Bogumiła **Mrozek**, Zbigniew **Mrozek**

MATLAB® and Simulink® examples

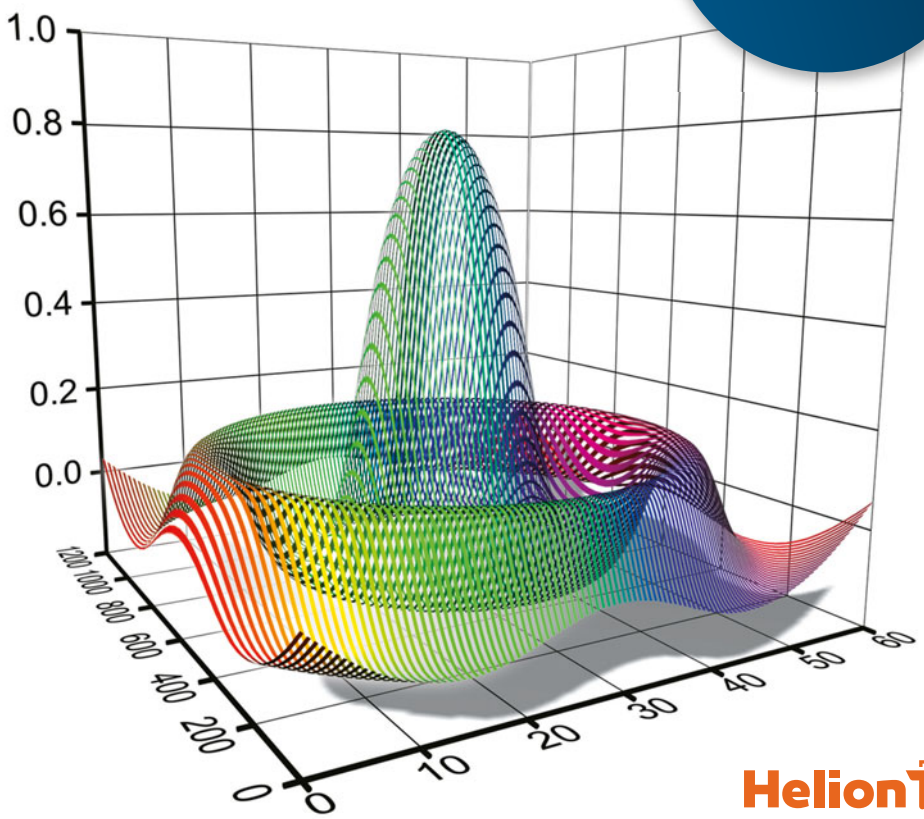
MATLAB i Simulink

PORADNIK UŻYTKOWNIKA

Modeluj, rozwiązuj problemy i wizualizuj wyniki!

Wykonuj złożone obliczenia i wykresy bez programowania
Poznaj elegancję programowania zorientowanego obiektowo
Wykorzystaj biblioteki Toolbox
Przeprowadź symulację procesów w Simulink

Wydanie IV



Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Grafika na okładce jest własnością autorów.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/matsi4_ebook
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-4260-6

Naszym Dzieciom i Wnukom:
Dorocie i Przemkowi
oraz Julii, Wandzi i Rysiowi

Spis treści

Przedmowa **13**

Rozdział 1. Wstęp **15**

1.1. Dlaczego MATLAB odnosi sukcesy? 15
1.1.1. MATLAB dla projektantów i naukowców 16
1.1.2. MATLAB rozumie matematykę 16
1.1.3. MATLAB udostępnia narzędzia rozwiązujące problemy matematyczne typowe dla wielu dziedzin nauki i techniki 16
1.1.4. MATLAB oferuje interaktywne aplikacje App 17
1.1.5. MATLAB integruje proces projektowania 17
1.1.6. MATLAB jest szybki, łatwy i wydajny 17
1.1.7. MATLAB jest wiarygodny 18
1.2. Środowisko MATLAB i Simulink 18
1.2.1. Interaktywna mapa środowiska MATLAB i Simulink 19
1.2.2. Rozszerzenia dostępne poprzez Add-On Explorer 20
1.2.3. Wybór licencji MATLAB-a 20
1.2.4. MATLAB w chmurze 22
1.2.5. MATLAB Distributed Computing Server 23
1.3. MATLAB i Simulink w internecie 24
1.3.1. Witryna producenta www.mathworks.com 24
1.3.2. MathWorks Account i logowanie do strony www.mathworks.com 25
1.3.3. Listy dyskusyjne i wyszukiwarki 25
1.4. MATLAB i Simulink w książkach 26

Rozdział 2. Pierwsze kroki w programie MATLAB **27**

2.1. Pierwsza sesja w programie MATLAB 27
2.1.1. Rozpoczęcie i zakończenie pracy z MATLAB-em 27
2.1.2. Pulpit MATLAB-a i jego okna 28
2.1.3. Przykład grafiki 3-D — funkcja peaks 29
2.1.4. Przykłady poleceń MATLAB-a 29
2.1.5. System pomocy doc i help — informacje wstępne 30
2.1.6. Zapisanie przebiegu sesji MATLAB-a 30
2.2. Matematyka i proste wykresy 30
2.2.1. Wyrażenia matematyczne, zmienne, zmienna ans 31
2.2.2. Funkcje arytmetyczne i trygonometryczne 31
2.2.3. Obliczanie wartości wyrażeń matematycznych 33
2.2.4. Błędy w zapisie poleceń i wyrażeń matematycznych 33
2.2.5. Powtórne użycie wcześniejszych poleceń 34
2.2.6. Formaty wypisywania liczb 34
2.2.7. Wizualizacja danych i wyników obliczeń bez programowania 35
2.2.8. Wykresy funkcji dwu- i trójwymiarowej z użyciem fplot i fplot3 36

2.2.9. Wektory, tablice i wykresy plot ... 38
2.2.10. Przykład rozwiązania układu równań algebraicznych ... 39
2.2.11. Konwersja układu współrzędnych ... 40
2.3. Zmienne w programie MATLAB ... 40
2.3.1. Tworzenie wektorów, tablic i macierzy ... 41
2.3.2. Zapisywanie zmiennych w plikach — MAT-plik i plik ASCII ... 42
2.3.3. Wczytywanie zmiennych z pliku ... 42
2.3.4. Usuwanie zmiennych z przestrzeni roboczej, czyszczenie ekranu ... 43
2.4. Dwukropek — operator generowania wektorów ... 44
2.4.1. Generowanie wektorów — dwukropek, linspace, logspace ... 44
2.4.2. Wybór żądanych wierszy, kolumn i elementów tablicy ... 44
2.4.3. Macierze — przykłady użycia notacji dwukropkowej ... 45
2.5. Operatory arytmetyczne i logiczne ... 47
2.5.1. Operatory arytmetyczne dla tablic i macierzy ... 47
2.5.2. Mnożenie i dzielenie wektorów i macierzy w MATLAB-ie ... 47
2.5.3. Przykłady operacji macierzowych i tablicowych ... 48
2.5.4. Dzielenie macierzowe i tablicowe ... 49
2.5.5. Operatory potęgowania macierzy i tablic ... 50
2.5.6. Sprzężenie i transponowanie macierzy i wektorów ... 50
2.5.7. Operatory relacji i operatory logiczne ... 51
2.5.8. Relacje i wyrażenia logiczne — przykłady ... 51
2.5.9. Funkcje logiczne ... 53
2.5.10. Priorytety operatorów arytmetycznych ... 54
2.6. Znaki i nazwy specjalne ... 54
2.7. System pomocy ... 56

Rozdział 3. Programowanie ... 59

3.1. Editor, Live Editor, pliki skryptowe ... 60
3.1.1. Editor — środowisko programistyczne MATLAB-a ... 60
3.1.2. Live Editor — nowe środowisko programistyczne ... 61
3.1.3. Uruchomienie skryptu z okna edytora lub Live Editor ... 63
3.1.4. Polecenia z okna History ... 63
3.1.5. Kotwiczenie edytora w panelu programu MATLAB ... 63
3.2. Pliki funkcyjne ... 64
3.2.1. Przygotowanie nowej funkcji w edytorze ... 64
3.2.2. Wywołanie pliku funkcyjnego ... 65
3.2.3. Zmienne lokalne, globalne i zmienna persistent ... 66
3.2.4. Funkcje lokalne ... 66
3.2.5. Funkcje zagnieżdżone ... 67
3.2.6. Funkcje prywatne ... 68
3.2.7. Funkcja anonimowa ... 69
3.2.8. Podpowiedzi w oknach Command i Editor ... 70
3.2.9. Priorytet wywołania funkcji ... 71
3.2.10. Polecenia i funkcje biblioteczne ... 72
3.3. Instrukcje sterujące przebiegiem programu ... 72
3.3.1. Instrukcje warunkowe if ... 73
3.3.2. Instrukcja wyboru switch ... 74
3.3.3. Instrukcje iteracyjne: while, for i dwukropek (:) ... 75
3.4. Wykrywanie błędów w MATLAB-ie ... 77
3.4.1. Lokalizacja błędów w pliku z programem ... 77
3.4.2. Błędy syntaktyczne i błędy wykonania ... 77
3.4.3. Programowa obsługa błędów — try-catch-end ... 78
3.5. Uruchamianie programu — debuger ... 79
3.5.1. Rozpoczęcie pracy z debugerem, wstawianie pułapki ... 79
3.5.2. Debuger — stan wstrzymania obliczeń ... 80

3.5.3. Debuger — zmienne lokalne funkcji ze stosu ... 81
3.5.4. Debuger — śledzenie kolejnych linii programu ... 81
3.6. Obsługa plików i folderów ... 82
3.6.1. Wykonywanie poleceń systemu operacyjnego ... 82
3.6.2. Folder aktualny i ścieżki dostępu — path ... 82
3.6.3. Pliki w MATLAB-ie ... 84
3.6.4. Zewnętrzne pliki z danymi — pliki ASCII i MAT-pliki ... 84
3.6.5. Binarne MEX-pliki wykonywalne ... 85
3.7. Poprawa wydajności MATLAB-a ... 86
3.7.1. Sposoby poprawiania wydajności programu ... 86
3.7.2. Wektoryzacja, operacje macierzowe i tablicowe ... 87
3.7.3. Rezerwowanie pamięci na macierze i wektory ... 87
3.7.4. Programowanie zorientowane obiektowo ... 88
3.8. Uwagi dla zaawansowanego użytkownika ... 88
3.8.1. Podział programu na sekcje (%% cell) ... 88
3.8.2. Wielokrotne wykorzystanie tworzonych funkcji ... 91
3.8.3. Raporty TODO/FIXME i inne ... 92
3.8.4. Shortcuts, czyli skróty ... 93
3.8.5. Preferencje — przygotowanie środowiska do pracy ... 93
3.8.6. Mysz czy klawiatura? ... 93
3.8.7. Funkcje eval, feval i podobne ... 94
3.8.8. Funkcje o zmiennej liczbie parametrów ... 96
3.9. Optymalizacja programu z użyciem profilera ... 97
3.10. Jaja wielkanocne ... 98

Rozdział 4. Wykresy w MATLAB-ie ... 99

4.1. Wykresy plot, fplot i inne ... 100
4.1.1. Funkcja plot ... 100
4.1.2. Fplot, ezplot i inne funkcje graficzne ... 101
4.1.3. Kolory, rodzaje linii i opisywanie wykresów ... 102
4.1.4. Dwie osie pionowe y, jedna oś pozioma x ... 103
4.1.5. Podział okna i modyfikowanie rysunków ... 104
4.1.6. Funkcje do opisywania i modyfikowania wykresów ... 105
4.2. Wykresy trójwymiarowe ... 106
4.2.1. Mapy dla funkcji trójwymiarowych ... 107
4.3. Interaktywne tworzenie i edycja rysunków ... 108
4.3.1. Przykład — przygotowanie danych do wykresu ... 109
4.3.2. Wykonanie wykresu z użyciem myszy ... 109
4.3.3. Interaktywna edycja wykresów ... 109
4.3.4. Narzędzia interaktywne — Plot Tools ... 110
4.3.5. Przygotowanie programu tworzącego grafikę ... 113
4.3.6. Przenoszenie rysunków i zapisywanie do pliku ... 114

Rozdział 5. Typy danych i programy obiektowo zorientowane ... 115

5.1. Fundamentalne typy danych ... 115
5.1.1. Numeryczne typy danych ... 116
5.1.2. Nienumeryczne typy danych ... 117
5.1.3. Funkcje konwersji typów i klas ... 117
5.2. Macierze pełne ... 117
5.2.1. Sposoby tworzenia macierzy ... 118
5.2.2. Wybrane funkcje i operacje macierzowe ... 120
5.3. Macierze rzadkie ... 122
5.3.1. Tworzenie macierzy rzadkich ... 122
5.3.2. Operacje na macierzach rzadkich ... 123
5.3.3. Uwagi dotyczące stosowania macierzy rzadkich ... 124

5.4. Łańcuchy i tablice znakowe ... 124
5.5. Tablice wielowymiarowe ... 126
5.5.1. Wizualizacja tablicy trójwymiarowej ... 127
5.5.2. Tworzenie tablicy trójwymiarowej przez indeksowanie i doklejanie warstw ... 128
5.6. Tablice komórkowe cell ... 128
5.6.1. Funkcje do obsługi tablic komórkowych ... 129
5.7. Struktury ... 130
5.7.1. Tworzenie struktury przez przypisanie ... 130
5.7.2. Tworzenie struktury z użyciem funkcji struct ... 130
5.8. Tabele i typ wyliczeniowy categorical ... 131
5.8.1. Tworzenie i obsługa tabel ... 131
5.8.2. Tablica wyliczeniowa, categorical array ... 132
5.9. Datatime, duration, calendarDuration ... 133
5.9.1. Typ datetime ... 133
5.9.2. Typ duration ... 133
5.9.3. Typ calendarDuration ... 134
5.9.4. Timetable ... 134
5.10. Tall Array i Tall Table — na duże zbiory danych ... 135

Rozdział 6. Klasy i programowanie obiektowo zorientowane ... 137

6.1. Definiowanie klasy — classdef ... 137
6.1.1. Przykład definiowania klasy — classdef ... 138
6.1.2. Tworzenie obiektu należącego do klasy ... 139
6.1.3. Tworzenie dodatkowych obiektów ... 140
6.1.4. Bloki properties, methods i events w definicji klasy ... 140
6.1.5. Pobieranie i zmiana wartości atrybutów obiektu ... 141
6.1.6. Atrybuty private i hidden, metody get i set ... 142
6.2. Dziedziczenie klas ... 143
6.2.1. Definiowanie podklasy ... 143
6.2.2. Obiekty należące do podklasy ... 145
6.2.3. Funkcje do obsługi klas i obiektów ... 145
6.3. Wizualizacja zmiany świateł drogowych ... 146
6.3.1. Synchronizacja pracy czterech sygnalizatorów ... 146
6.3.2. Skrypt uruchamiający wizualizację ... 148
6.4. Przeciążanie funkcji i operatorów ... 149
6.4.1. Przeciążanie w Control System Toolbox ... 150
6.4.2. Jednoznaczność wyboru operatora lub funkcji ... 150
6.5. Klasy języka Java ... 151
6.6. Zadania dla Czytelnika ... 151

Rozdział 7. App Designer i GUI ... 153

7.1. Obiekty graficzne MATLAB-a ... 153
7.1.1. Hierarchia obiektów graficznych ... 153
7.1.2. Atrybuty obiektu graficznego ... 155
7.1.3. Zmiana atrybutów modyfikuje obiekty graficzne ... 156
7.2. App Designer, tworzenie aplikacji z GUI ... 157
7.2.1. Panel App Designer ... 157
7.2.2. Automatyczne generowanie kodu aplikacji ... 159
7.2.3. Wywołania zwrotne, dodatkowe funkcje i zmienne ... 159
7.3. Przykład — interaktywny wykres cos(x) ... 160
7.3.1. Odczytanie wartości po zmianie położenia suwaka ... 160
7.3.2. Wyświetlenie wartości w okienku edycyjnym ... 161
7.3.3. Wykonanie wykresu w oknie graficznym ... 161
7.3.4. Nowy wykres po zmianie w oknie edycyjnym ... 162

7.3.5. Zamknięcie okna aplikacji — przycisk STOP ... 162
7.3.6. Wersja końcowa aplikacji ... 162
7.4. App Designer — interaktywna wizualizacja rozwiązań równań różniczkowych ... 163
7.4.1. Przykład — wizualizacja drgań z uwzględnieniem zmiany tłumienia ... 163
7.4.2. Scenariusz i aranżacja obiektów graficznych ... 164
7.4.3. Kodowanie wywołań zwrotnych i wykresów ... 165
7.4.4. Zalety użycia uchwytu zamiast nazwy funkcji ... 167
7.5. Instalowanie aplikacji ... 167
7.5.1. Instalowanie własnych aplikacji ... 168
7.5.2. Instalowanie rozszerzeń dostępnych poprzez Add-On Explorer ... 168

Rozdział 8. Metody numeryczne ... 169

8.1. Układy równań liniowych ... 169
8.1.1. Dzielenie prawostronne dla x·A=b ... 170
8.1.2. Automatyczny wybór algorytmu ... 170
8.1.3. Kilka rozwiązań dla różnych wektorów b ... 170
8.1.4. Przykład rozwiązania równania liniowego z liczbami zespolonymi ... 171
8.1.5. Równania liniowe źle uwarunkowane ... 171
8.1.6. Sprawdzenie poprawności rozwiązań ... 172
8.1.7. Pseudoodwrotność i rozwiązywanie równań nadokreślonych i niedookreślonych ... 173
8.2. Równania różniczkowe zwyczajne, zagadnienie początkowe ... 173
8.2.1. Zagadnienie początkowe, solvery odeXX ... 173
8.2.2. Wybór parametrów dla solvera odeXX ... 174
8.2.3. Modyfikowanie parametrów solvera odeXX ... 175
8.2.4. Wpływ parametrów solvera na obliczenia ... 176
8.2.5. Algorytmy dla układów źle uwarunkowanych ... 178
8.3. Równanie różniczkowe III rzędu, przykład ... 180
8.3.1. Przekształcenie równania różniczkowego III rzędu na układ trzech równań I rzędu ... 180
8.3.2. Przygotowanie pliku funkcyjnego obliczającego pochodne ... 181
8.3.3. Wywołanie solvera odeXX i wizualizacja rozwiązania ... 182
8.3.4. Symbolic Math Toolbox ... 183
8.3.5. Model równania różniczkowego w Simulinku ... 183
8.4. Inne równania różniczkowe i cząstkowe ... 184
8.4.1. Równania różniczkowe z macierzą M(t) ... 184
8.4.2. Zagadnienie brzegowe ... 185
8.4.3. Równania różniczkowe zwyczajne z opóźnieniem ... 186
8.4.4. Równania różniczkowe cząstkowe ... 186
8.4.5. Zadania do samodzielnego rozwiązania ... 186
8.5. Całkowanie i różniczkowanie ... 187
8.5.1. Całkowanie numeryczne ... 187
8.5.2. Całkowanie analityczne — Symbolic Math Toolbox ... 188
8.5.3. Różniczkowanie numeryczne i analityczne ... 188
8.6. Dekompozycja macierzy ... 189
8.6.1. Dekompozycja LU ... 189
8.6.2. Rozkład Cholesky'ego ... 190
8.6.3. Dekompozycja QR ... 190
8.6.4. Obliczenia równoległe w algebrze liniowej ... 190
8.7. Wartości własne i wektory własne ... 191
8.8. Analiza funkcji ... 191
8.8.1. Minimum, maksimum i miejsca zerowe funkcji ... 191
8.8.2. Obliczanie zer wielomianu ... 191

8.8.3. Równanie nieliniowe i źle uwarunkowane ... 192
8.8.4. Wielomian i funkcje wielomianowe ... 193
8.9. Interpolacja i aproksymacja ... 194
8.9.1. Interpolacja wielomianowa ... 194
8.9.2. Aproksymacja wielomianowa ... 194
8.9.3. Funkcja sklejana — spline function ... 194
8.9.4. Przykład interpolacji i aproksymacji ... 195
8.9.5. Interpolacja i aproksymacja w oknie Basic Fitting ... 196
8.10. Analiza statystyczna ... 196
8.11. Analiza sygnałów ... 198
8.11.1. Przykład analizy przebiegu odkształconego ... 199

Rozdział 9. Przetwarzanie obrazów ... 201
9.1. Zapis i odczyt obrazów, liczby 8- i 16-bitowe bez znaku ... 201
9.2. Grafika 24-bitowa (true color) ... 202
9.3. Palety barw i obrazy indeksowane ... 203
9.3.1. Palety barw ... 203
9.3.2. Obrazy indeksowane ... 205
9.3.3. Obrazy w skali szarości, pseudokolor ... 205
9.3.4. Przekodowywanie obrazów stało- i zmiennoprzecinkowych ... 205
9.3.5. Przekodowanie obrazu kolorowego na szary ... 206
9.4. Przetwarzanie obrazów rastrowych ... 206
9.4.1. Przygotowanie obrazu do testów ... 206
9.4.2. Przetwarzanie punktowe — negatyw ... 207
9.4.3. Przetwarzanie punktowe — rozjaśnianie i ściemnianie ... 207
9.4.4. Operacje logiczne — binaryzacja ... 208
9.4.5. Operacje morfologiczne ... 209
9.4.6. Operacje arytmetyczne. Filtry ... 210
9.5. Światło, odbicia i tekstury ... 212
9.5.1. Źródła światła i odbicia ... 212
9.5.2. Tekstura — nakładanie obrazu na powierzchnię ... 213

Rozdział 10. Simulink — pakiet do modelowania i symulacji ... 215
10.1. Pierwsza sesja z Simulinkiem ... 215
10.1.1. Otwarcie okna Simulink Editor ... 216
10.1.2. Pobieranie bloków z bibliotek do okna modelu ... 216
10.1.3. Łączenie bloków liniami przesyłania sygnałów ... 217
10.1.4. Uruchomienie symulacji ... 218
10.1.5. Wizualizacja wyników symulacji ... 219
10.1.6. Tryby przyspieszone symulacji — accelerated i rapid ... 220
10.1.7. Stepper ... 221
10.2. Druga sesja z Simulinkiem ... 221
10.3. Podsystemy — blok Subsystem ... 223
10.3.1. Przykład modelu definiowanego graficznie ... 223
10.3.2. Tworzenie podsystemów ... 224
10.3.3. Maskowanie podsystemów ... 225
10.4. Biblioteki bloków ... 227
10.4.1. Tworzenie własnych bibliotek bloków ... 227
10.4.2. Dodatkowe bloki i biblioteki z toolboksów ... 227
10.5. Modelowanie fizyczne ... 228
10.6. Stateflow — modelowanie i symulacja systemów sterowanych zdarzeniami ... 229
10.6.1. Simulink i blok Stateflow Chart ... 229
10.6.2. Diagram stanu ... 230
10.6.3. Symulacja sterowania poślizgowego ... 231
10.6.4. Generowanie kodu dla modeli ... 232
10.7. Dodatkowe moduły rozszerzające środowisko Simulinka ... 232

Rozdział 11. Dodatek 235
11.1. MATLAB Online i MATLAB Drive 235
11.2. Big data, obsługa dużych zbiorów danych 236
11.2.1. Zbiory danych używane do badań i testowania 237
11.2.2. Dostęp do danych przez obiekt datastore 238
11.2.3. Tall array i tall table 242
11.2.4. MapReduce 246
11.3. Obliczenia równoległe 246
11.3.1. Automatyczny tryb pracy równoległej 246
11.3.2. Obliczenia równoległe na komputerze wieloprocesorowym 246
11.3.3. Biblioteki toolbox z bezpośrednim wsparciem dla obliczeń równoległych 248
11.3.4. Obliczenia równoległe z użyciem GPU 248
11.3.5. Klastry, chmury, sieci lokalne i rozproszone 250
11.4. Modele obiektów używanych w Control System Toolbox™, System Identification Toolbox™ i w Robust Control Toolbox™ 250
11.4.1. Abstrakcyjne superklasy definiujące atrybuty i metody wykorzystane w modelach 251
11.4.2. Klasa numlti oraz modele typu tf, zpk, ss i inne 252
11.4.3. Liniowe i nieliniowe modele identyfikowalne 253
11.4.4. Modele uogólnione oraz z niepewnymi parametrami 254
11.4.5. Bloki Simulinka 257
11.5. Biblioteka Control System Toolbox™ 258
11.5.1. Ciągłe i dyskretne modele numlti w ControlSystem Toolbox 258
11.5.2. Model dyskretny i równanie w dziedzinie czasu 259
11.5.3. Pobieranie danych z modelu numlti 262
11.5.4. Zmiana nazwy zmiennej w polu Variable 262
11.5.5. Badanie właściwości modelu 263
11.6. ThingSpeak™ — obsługa urządzeń IoE w chmurze 264
11.6.1. Tworzenie nowego kanału dla danych IoE 265
11.6.2. Przykład przetwarzania danych w chmurze 266
11.6.3. Przetwarzanie danych pomiarowych w chmurze 267
11.6.4. Wsparcie dla różnych technologii 268

Bibliografia 269

Skorowidz 271

Przedmowa

Książka jest adresowana do osób, które nie mają doświadczenia w stosowaniu MATLAB-a lub chcą poszerzyć swoją wiedzę. Książka obejmuje opis języka i pakietu MATLAB, wybrane biblioteki *toolbox*, wprowadzenie do *Simulinka* oraz wybrane przez autorów *rozszerzenia* MATLAB-a. Jest to poradnik, pełen wskazówek i przykładów. Już po przeczytaniu drugiego rozdziału można zacząć *efektywne korzystanie* z MATLAB-a i jego wybranych rozszerzeń. Książka nie jest podręcznikiem i nie zawiera dokładnego opisu ponad 2500 funkcji MATLAB-a i jego rozszerzeń.

MATLAB® jest efektywnym i uniwersalnym językiem programowania wysokiego poziomu. Jest też interaktywnym środowiskiem do obliczeń naukowych i technicznych oraz do tworzenia algorytmów i analizy danych. Ma znakomite narzędzia do dwu- i trójwymiarowej wizualizacji danych i wyników obliczeń w postaci różnych wykresów, grafiki biznesowej i animacji. Jest też przyjaznym, interaktywnym środowiskiem integrującym niezawodne algorytmy matematyki stosowanej i liczne moduły rozszerzeń i narzędzi zorientowanych na określone dziedziny zastosowań.

Liczba użytkowników MATLAB-a przekroczyła na świecie milion, a środowisko MATLAB stało się faktycznym standardem i podstawowym narzędziem naukowca oraz inżyniera w różnych dziedzinach. W ostatnich latach wzrosło zainteresowanie wykorzystaniem MATLAB-a w finansach, biologii, medycynie, przemyśle samochodowym, lotnictwie oraz przy tworzeniu niezawodnych aplikacji i oprogramowania, w tym oprogramowania do urządzeń z wbudowanym procesorem lub układami FPGA.

MATLAB umożliwia rozwiązywanie różnorodnych problemów poprzez ułatwienie dostępu do efektywnych algorytmów obliczeniowych i pozwala na wizualizację wyników. W znacznym stopniu wyparł z obliczeń naukowo-technicznych języki uniwersalne (Fortran, C, C++), ograniczając ich zasadniczą rolę do funkcji oprogramowania narzędziowego. Wygrywa też z bezpłatnym oprogramowaniem open source, jak Octave, SciLab oraz Python — źródła sukcesu MATLAB-a wyjaśniono w podrozdziale 1.1. Środowisko MATLAB jest otwarte. Daje możliwość zarówno wykorzystywania oprogramowania istniejącego, jak i budowania oprogramowania własnego.

MATLAB jest oferowany dla systemów *Windows*, *Linux* i *Macintosh*. Znacznie poszerzono ofertę licencji: studenci i pracownicy kilku polskich uczelni mogą już instalować MATLAB-a na prywatnych komputerach w ramach licencji akademickiej *TAH*. Każdy może zakupić tanią licencję *Home*. Jest też licencja *STEM* dla szkół średnich lub podstawowych, a nie posiadając żadnej licencji można korzystać z usługi ThingSpeak.

Nieodpłatne *obliczenia w chmurze* są dostępne w licencjach TAH i TSH oraz (pod pewnymi warunkami) — w innych licencjach. Obliczenia w chmurze prowadzi się poprzez przeglądarkę WWW (*MATLAB Online*) lub poprzez aplikację *MATLAB Mobile* na smartfonach i tabletach. *MATLAB Drive* przechowuje pliki w chmurze i synchronizuje je z plikami na komputerach.

Niniejsze wydanie książki uwzględnia najnowsze udoskonalenia środowiska MATLAB. Opisano tu *App Designer* (ułatwia tworzenie aplikacji i zapewne zastąpi starszego GUIDE), nowy interaktywny *Live Editor* (integruje tworzony kod z jego dokumentacją i prezentacją wyników obliczeń) i inne.

Większość rozszerzeń MATLAB-a jest dostosowana do pracy równoległej na procesorach wielordzeniowych, klastrach, a także na wybranych kartach graficznych. Programowanie *zorientowane obiektowo* ma coraz większe znaczenie i jest omawiane w oddzielnym rozdziale.

Autorzy pragną podziękować firmom **Oprogramowanie Naukowo-Techniczne** z Krakowa oraz **The MathWorks, Inc.** z USA za udostępnienie najnowszych wersji MATLAB-a i jego rozszerzeń oraz za zgodę na wykorzystanie rysunków z dokumentacji omawianych produktów. Serdecznie dziękujemy pani Małgorzacie Kulik, redaktor prowadzącej z Grupy Helion SA, za pomoc w redagowaniu i edycji książki. Jej wyjątkowo duża wiedza informatyczna i doświadczenie edytorskie pozwalały na szybkie znalezienie właściwego sposobu prezentacji treści w IV wydaniu tej książki.

Prosimy Czytelników o nadsyłanie uwag i sugestii na adres autorów: *bmrozek@pk.edu.pl* lub *pemrozek@gmail.com*.

Autorzy
Kraków, sierpień 2017

Rozdział 1.

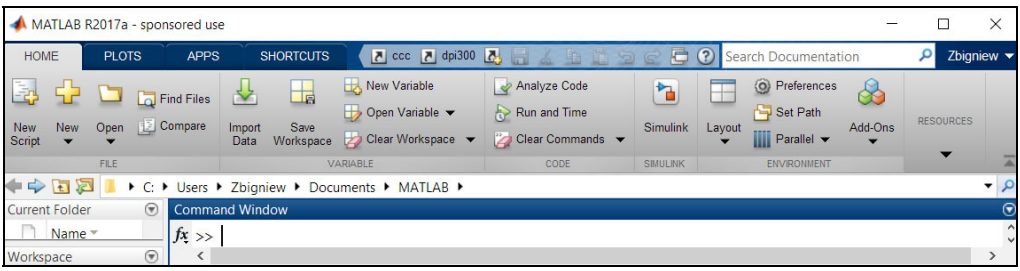
Wstęp

MATLAB® jest produktem firmy **The MathWorks, Inc.** z USA. Używanie MATLAB-a i jego rozszerzeń wymaga posiadania licencji (podrozdział 1.2.3). Niniejsza książka opisuje 64-bitową wersję MATLAB-a 2017a, ale uwzględnia też ostatnią 32-bitową wersję 2015b. ThingsSpeak™ ma bezpłatny ograniczony dostęp do MATLAB-a

- Wybrane przykłady programów z niniejszej książki są dostępne w internecie. Aby je pobrać, należy otworzyć stronę *www.mathworks.com/* i w okienku wyszukiwania wpisać nazwisko autorów książki: `Mrozek` lub otworzyć *http://www.mathworks.com/matlabcentral/fileexchange/?term=mrozek*.
- Niektóre przykłady są dostępne bezpośrednio z panelu MATLAB-a, po wybraniu zakładki *APPS/Get More Apps* (podrozdział 1.2.2) lub *HOME/Community* — i po wpisaniu nazwiska `Mrozek` w okienku *Search*.
- Przykłady dla starszych wydań MATLAB-a zamieszczono wraz z opisem wcześniejszych wydań tej książki na *https://www.mathworks.com/*.

1.1. Dlaczego MATLAB odnosi sukcesy?

MATLAB jest przyjazny dla użytkownika. Efektywne korzystanie z MATLAB-a i jego wybranych rozszerzeń jest możliwe bezpośrednio z panelu (rysunek 1.1-1), bez potrzeby programowania, już po przeczytaniu nawet tylko drugiego rozdziału tej książki. Jednakże przygotowanie prostego programu lub aplikacji App umożliwia wielokrotne wykorzystanie przygotowanej sekwencji obliczeń.



Rysunek 1.1-1. *Panel MATLAB-a — wstążka z ikonami narzędzi dostępnych z zakładki HOME*

MATLAB zawiera szeroki zestaw funkcji graficznych dostępnych z panelu po wybraniu zakładki *PLOTS*: generowanie wykresów funkcji jednej i dwóch zmiennych, wykresów kołowych, paskowych, konturowych, cieniowanych, wizualizację odwzorowań dwu- i trójwymiarowych. Posiada także bogate środki opisu tekstowego: różne czcionki, symbole specjalne, strzałki itp.

1.1.1. MATLAB dla projektantów i naukowców

Inżynierowie i naukowcy nie powinni tracić czasu na czytanie obszernych tomów dokumentacji oprogramowania ani na tworzenie własnego oprogramowania *ab ovo*. Potrzebują gotowego, niezawodnego i łatwego w użytkowaniu systemu, który wspomaga obliczenia matematyczne typowe dla ich pracy. MATLAB i jego rozszerzenia spełniają te wymagania.

MATLAB umożliwia rozwiązywanie różnorodnych problemów poprzez ułatwienie dostępu do efektywnych algorytmów obliczeniowych i możliwość wizualizacji wyników. W znacznym stopniu wyparł z obliczeń naukowo-technicznych języki uniwersalne (Fortran, C, C++), ograniczając ich zasadniczą rolę do funkcji oprogramowania narzędziowego. Środowisko MATLAB jest otwarte. Daje to możliwość zarówno wykorzystywania oprogramowania istniejącego, jak i budowania oprogramowania własnego, w tym aplikacji App, toolboksów i bibliotek modeli symulacyjnych.

1.1.2. MATLAB rozumie matematykę

MATLAB jest językiem wysokiego poziomu. Ma wbudowane operacje matematyczne na *wektorach*, *liczbach zespolonych* i *macierzach*. Nawet zaawansowane obliczenia można wykonać bez programowania, wpisując wyrażenia matematyczne w głównym oknie programu. Uzyskane wyniki można przedstawić na wykresach dwu- i trójwymiarowych lub dalej przetwarzać, korzystając z dostępnych aplikacji. Prócz operacji macierzowych zaimplementowano w MATLAB-ie operacje na macierzach rzadkich i *tablicach*.

Profesjonalna **biblioteka matematyczna i graficzna** jest oparta na bibliotekach optymalizowanych pod kątem operacji blokowo-macierzowych. LAPACK *Linear Algebra Package* i BLAS *Basic Linear Algebra Subroutines* stanowią bazę dla wszystkich elementów składowych środowiska MATLAB. Wbudowano ją częściowo do jądra MATLAB-a, a częściowo ma ona postać plików zewnętrznych, umieszczonych w folderach mających początek w *matlab\toolbox\matlab*. MATLAB wykorzystuje też bardzo szybką bibliotekę transformat Fouriera — FFTW (*Fastest Fourier Transform in the West*).

1.1.3. MATLAB udostępnia narzędzia rozwiązujące problemy matematyczne typowe dla wielu dziedzin nauki i techniki

Na bazie MATLAB-a, który ma wbudowane algorytmy matematyczne i obsługę grafiki, utworzono kilkadziesiąt rozszerzeń, w tym toolboksy, Simulink i jego biblioteki oraz narzędzia pozwalające rozwiązywać rzeczywiste problemy z wielu dziedzin. Powstają

też kolejne narzędzia uwzględniające nowe obszary zastosowań MATLAB-a. Wiele z nich można uzyskać nieodpłatnie. Dzięki temu użytkownik może w prosty sposób dobrać zestaw rozszerzeń adekwatnych do swoich potrzeb.

1.1.4. MATLAB oferuje interaktywne aplikacje App

Aplikacje App to interaktywne programy obsługujące pełny cykl obliczeń poprzez interfejs użytkownika, od pobrania danych po wykonanie obliczeń i prezentację wyników w postaci tekstowej lub graficznej. Aplikacje umożliwiają zmianę algorytmów obliczeniowych, ich parametrów oraz sposobów wizualizacji i zapisywania wyników. Pozwala to na powtarzanie obliczeń (iterację) aż do uzyskania zadowalających rezultatów.

Pewna liczba aplikacji App jest dostarczana wraz z MATLAB-em i jego rozszerzeniami. Dodatkowe aplikacje App można przygotować samodzielnie, korzystając z narzędzia *App Designer* (podrozdział 7.2). Można też przeszukać zasoby udostępniane poprzez opcję *Add-On Explorer* (podrozdziały 1.2.2 i 7.5.2).

1.1.5. MATLAB integruje proces projektowania

MATLAB ułatwia współpracę specjalistów z różnych dziedzin. Podczas realizacji projektów mogą oni wykorzystywać potrzebne im rozszerzenia MATLAB-a do budowy modeli i do badań symulacyjnych. Korzystając z Simscape można integrować modele z różnych dziedzin i badać je metodami symulacyjnymi (ang. *model based design*). Można też tworzyć modele czasu rzeczywistego i badać je w trybie HiL (ang. *hardware in the loop*). Algorytmy użyte w modelach można przetworzyć na kod, który pozwoli zaprogramować odpowiednio karty prototypowe, karty z układem FPGA (ang. *field-programmable gate array*), sterowniki PLC (ang. *programmable logic controller*) lub systemy z wbudowanym procesorem.

1.1.6. MATLAB jest szybki, łatwy i wydajny

MATLAB jest znacznie szybszy niż dawniej, gdyż ulepszono jego silnik LXE (ang. *language execution engine*). Co więcej, obliczenia mogą być jeszcze szybsze przy wykorzystaniu algorytmów równoległych (podrozdział 11.3). Ponadto wszystkie operacje są wykonywane z dużą dokładnością na liczbach o podwójnej precyzji — chyba że będą inne wymagania (np. przetwarzanie grafiki jest często efektywniejsze z użyciem liczb typu `uint8`).

Przy projektowaniu czas przygotowania i testowania programu jest często znacznie dłuższy niż czas faktycznych obliczeń. MATLAB jest przyjazny dla użytkownika, a wiele zadań można zrealizować w nim z poziomu pulpitu (rysunek 1.1-1), bez potrzeby programowania.

Jeśli przygotowanie programu jest konieczne, to programista jest wspierany przez wbudowane inteligentne edytory, bardzo bogate biblioteki oprogramowania, debuger, profiler i inne narzędzia.

1.1.7. MATLAB jest wiarygodny

Poprawność i dokładność używanych algorytmów są kontrolowane od wielu lat przez producenta (firma powstała w 1984 roku) oraz przez ponad milion użytkowników. Jak już wspomniano, operacje matematyczne w MATLAB-ie są wykonywane z dużą dokładnością na liczbach o podwójnej precyzji. Przy przetwarzaniu grafiki zazwyczaj wystarczające jest użycie liczb `uint8` (ang. *unsigned integer 8-bit*).

Jeśli MATLAB jest używany w projektowaniu systemów krytycznych z uwagi na bezpieczeństwo — przewidziano konieczność uzyskania certyfikatów wymaganych przez prawo. Służą do tego narzędzia do badania poprawności kodu (grupa *Polyspace*) oraz:

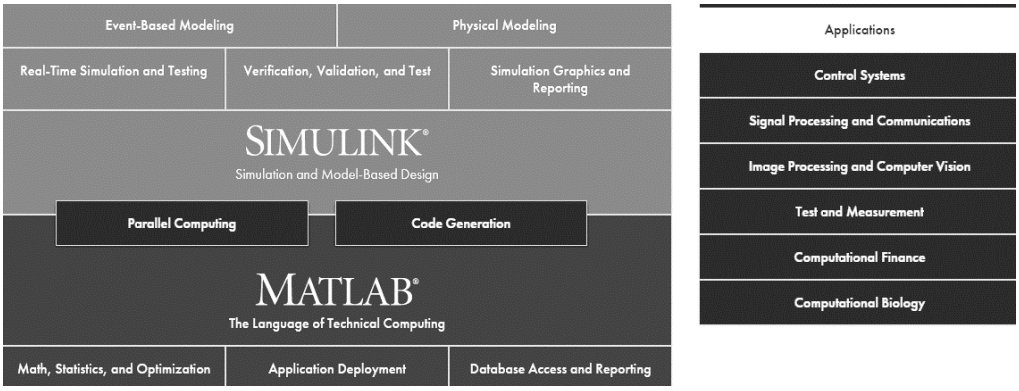
- IEC Certification Kit (dla IEC 61508 i ISO 26262), używany np. przy projektowaniu systemów bezpieczeństwa w samochodach: ABS, napinania i zwalniania pasów oraz obsługi poduszek powietrznych;
- DO Qualification Kit (certyfikacja awioniki lotniczej DO-178C, DO-278A).

O poziomie zaufania do MATLAB-a i Simulinka może świadczyć ich użycie do kontroli lotu i nawigacji sondy podczas misji kosmicznej New Horizons (rozpoczętej w 2006 roku; sonda dotarła do Plutona w roku 2015) — *http://blogs.mathworks.com/steve/2015/07/19/new-horizons-pluto-program-used-matlab-and-image-processing-for-navigation/*.

1.2. Środowisko MATLAB i Simulink

MATLAB, Simulink i ich rozszerzenia tworzą środowisko programistyczne pokazane na rysunku 1.2-1. Bazą tego środowiska są:

- MATLAB, który udostępnia narzędzia i oprogramowanie dostępne z panelu (rysunek 1.1-1), w tym algorytmy matematyczne i grafikę.
- Simulink, który wykorzystując MATLAB-a, tworzy środowisko do *graficznego projektowania systemów* w oparciu o modele (ang. *model based design*) oraz do symulacji i analizy tych systemów.



Rysunek 1.2-1. *Środowisko pakietu MATLAB/Simulink — kopia interaktywnej strony http://www.mathworks.com/products/pfo (reprinted with permission of The MathWorks, Inc.)*

Żaden z elementów tego środowiska (program, funkcja, biblioteka) nie będzie działać bez MATLAB-a lub bez MATLAB-a i Simulinka. Wyjątkiem są programy skompilowane przez *MATLAB Compiler*, do których dołączono niezbędne biblioteki oraz komputery i ich klastry pracujące równolegle w sieci zarządzanej przez *MATLAB Distributed Computing Server*.

Pozostałe zależności (np. *Econometrics Toolbox* wymaga MATLAB-a oraz toolboksów: *Optimization Toolbox i Statistics and Machine Learning Toolbox*) są podane w tabeli *Add-On Product Requirements & Platform Availability for R2017a*. Jest ona dostępna na *www.mathworks.com* po wpisaniu `Product Requirements` w okienku *Search.*

1.2.1. Interaktywna mapa środowiska MATLAB i Simulink

Na stronie *http://www.mathworks.com/products/pfo* (rysunek 1.2-1) jest dostępna interaktywna mapa odpłatnych rozszerzeń MATLAB-a (ok. 60), rozszerzeń Simulinka (ok. 40) oraz innych usług i produktów. Kliknięcie wybranego obszaru tej strony powoduje wyświetlenie linków i nazw rozszerzeń przeznaczonych do obsługi odpowiednich zastosowań. Kliknięcie kolejnych linków wyświetla szczegółowy opis wybranego rozszerzenia.

Rozszerzenia MATLAB-a to ok. 40 bibliotek *toolbox* (zwanych też przybornikami) oraz ok. 20 aplikacji i narzędzi, w tym do tworzenia oprogramowania, pracy równoległej, raportowania i innych usług. Toolboksy to wyspecjalizowane pakiety oprogramowania, które poszerzają MATLAB-a o zastosowania z zakresu automatyki, przetwarzania sygnałów i obrazów, optymalizacji, inżynierii finansowej, obliczeń symbolicznych, sieci neuronowych, logiki rozmytej i wielu innych.

Rozszerzeniami Simulinka (rozdział 10.) są:

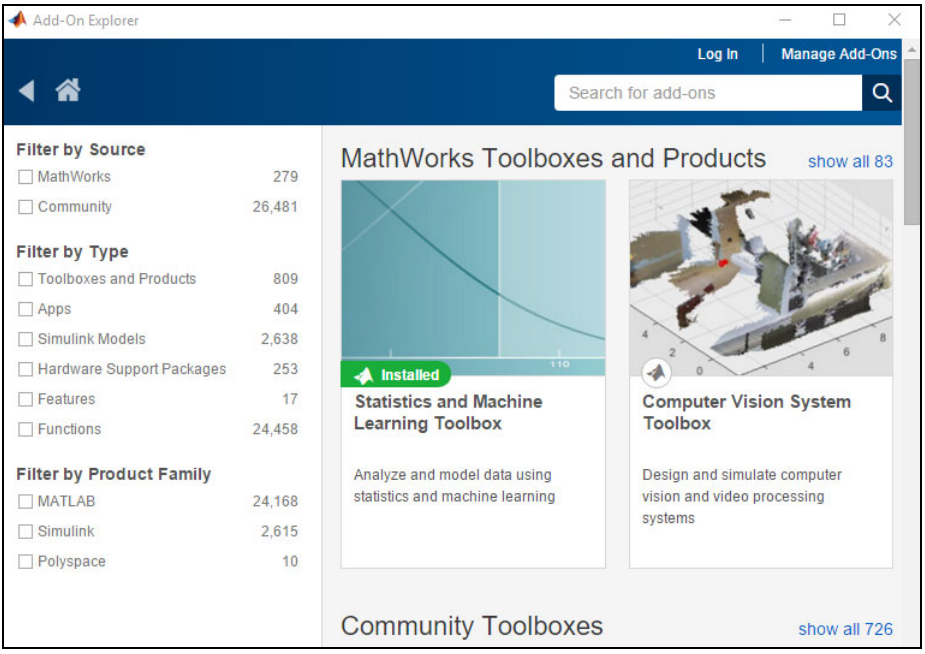
- *Blocksets*, czyli dodatkowe biblioteki bloków zawierające wyspecjalizowane modele i poszerzające zastosowania Simulinka;
- dodatkowe bloki Simulinka dostarczane wraz z *Control System Toolbox™*, *System Identification Toolbox™*, *Fuzzy Logic Toolbox™* i innymi toolboksami; niektóre z nich omówiono w podrozdziale 11.4.5;
- pakiety do wielodomenowego, wirtualnego modelowania fizycznego: *Simscape*, *Simscape Multibody*, *Simscape Driveline*, *Simscape Fluids*, *SimRF*, *Simscape Electronics i Simscape Power Systems*;
- *SimEvents* i *Stateflow®*, czyli zintegrowane z Simulinkiem środowisko do symulacji procesów ze zdarzeniami i maszyna stanów do symulacji systemów reaktywnych;
- narzędzia do symulacji i testowania w czasie rzeczywistym;
- narzędzia do weryfikacji, walidacji i testowania;
- aplikacje do zastosowania w obszarach wymienionych w prawej kolumnie na rysunku 1.2-1.

Wybrane rozszerzenia MATLAB-a i Simulinka oraz ich zastosowania omówiono w Dodatku, w rozdziale 11.

1.2.2. Rozszerzenia dostępne poprzez Add-On Explorer

MATLAB ma narzędzie do poszukiwania i instalowania rozszerzeń oferowanych zarówno przez producenta, The MathWorks Inc., jak i przez niezależne firmy, uniwersytety i użytkowników MATLAB-a.

Po wybraniu na panelu MATLAB-a zakładki *APPS* i ikony *Get More Apps* pojawia się okno *Add-On Explorer* (rysunek 1.2.2-1). Narzędzie to umożliwia przeglądanie wielu tysięcy opisów dodatkowych funkcji, ponad 2600 modeli Simulinka, ponad 800 toolboksów i innych produktów oraz ponad 250 pakietów oprogramowania do współpracy MATLAB-a i Simulinka z systemami *Android*, *Arduino*, *Lego Mindstorms*, *National Instruments*, *Raspberry Pi* i wieloma innymi.



Rysunek 1.2.2-1. *Okno Add-On Explorer umożliwiające przeglądanie wielu tysięcy opisów dodatkowego oprogramowania do MATLAB-a i Simulinka*

Niektóre przykłady programów udostępniane na *http://www.mathworks.com/matlabcentral/* są dostępne również w Add-On Explorer. Po wpisaniu nazwiska `Mrozek` w okienku *Search* pojawią się niektóre przykłady z tej książki (żadna z opcji *Filter by Type* nie powinna być zaznaczona).

1.2.3. Wybór licencji MATLAB-a

Koszt zakupu licencji MATLAB-a zależy od okresu jej ważności (3 miesiące, 12 miesięcy lub bezterminowo), od zakupionych rozszerzeń (toolboksy, toolkity i inne rozszerzenia) oraz od możliwości uzyskania ceny akademickiej, studenckiej, edukacyjnej czy

też domowej. Licencje bezterminowe mają 12-miesięczny okres wsparcia technicznego i aktualizacji oprogramowania. Po tym okresie producent zaleca zakupienie subskrypcji aktualizacji i wsparcia.

1.2.3.1. Licencja standardowa

Licencje standardowe są przeznaczone do wszelkich zastosowań komercyjnych. Licencje te mogą być 3-miesięczne, 12-miesięczne lub bezterminowe.

Dostępna jest też licencja *MATLAB Distributed Computing Server* (MDCS) na klaster komputerów. Umożliwia ona wykonywanie obliczeń przygotowanych na licencjach programu MATLAB i przesłanych do MDCS za pomocą modułu Parallel Computing Toolbox.

1.2.3.2. Licencja akademicka i dla szkół

Uczelnie i inne instytucje akademickie mogą zakupić indywidualne i sieciowe licencje edukacyjne (do celów dydaktycznych) oraz licencje do niekomercyjnych badań naukowych i publikacji ich wyników.

Najwyższe licencje zwane *Total Academic Headcount* i *Total Student Headcount* zezwalają na instalowanie kopii oprogramowania również na prywatnych komputerach odpowiednio studentom i naukowcom bądź tylko studentom. Umożliwiają też korzystanie z MATLAB Online (w chmurze).

Studenci i pracownicy kilku polskich uczelni mogą już instalować MATLAB-a na prywatnych komputerach w ramach licencji akademickiej *TAH*. Licencja *TSH* daje prawo instalowania MATLAB-a tylko studentom. Studentom udostępniane są jeszcze lepsze i tańsze oferty. Jest też licencja *STEM* dla szkół średnich i podstawowych.

1.2.3.3. Licencja studencka

Wersja studencka MATLAB-a jest wieloplatformowa, może być zainstalowana na 64-bitowych systemach Windows, Linux i Mac OS. Jest prawie identyczna z wersją standardową i jest oferowana w jednym z dwu wariantów:

- wersja poszerzona **MATLAB and Simulink Student Suite** zawiera MATLAB-a, Simulinka i 10 innych najczęściej używanych produktów oraz dodatkowo narzędzia do prototypowania, testowania i uruchamiania modeli na Arduino®, LEGO® MINDSTORMS® NXT™ i EV3™ oraz Raspberry Pi® i innych popularnych urządzeniach; cena rekomendowana to 69 euro + VAT (w 2017 roku);
- tylko MATLAB (cena rekomendowana 35 euro + VAT) — pozostałe elementy mogą być dokupione zależnie od kierunku studiów.

Najbardziej widoczne różnice pomiędzy wersjami studencką i standardową to znak zachęty `EDU>>` zamiast `>>` oraz dopisek *Student Version of MATLAB* umieszczany na wydrukach. W Simulinku liczba bloków nie może być większa niż 1000 i nie mogą być używane akceleratory. Do wersji studenckiej można dokupić prawie wszystkie toolboksy i toolkity oraz rozszerzenia *SimScape* i podobne.

1.2.3.4. MATLAB Home

Pełna wersja MATLAB-a dla użytkowników domowych jest oferowana w cenie 105 euro + VAT (w 2017 roku). Rozszerzenia (w tym Simulink) są oferowane w cenie 29 euro każde. Nie ma ograniczeń w funkcjonalności MATLAB-a. Jako zabezpieczenie na ekranie jest wyświetlana informacja o użytkowaniu wersji Home — nie wszystkie rozszerzenia MATLAB-a mogą wtedy być instalowane.

W Simulink Home ograniczono maksymalną liczbę bloków w modelu do 1000 i nie ma możliwości używania akceleratorów Simulinka.

1.2.4. MATLAB w chmurze

Obliczenia w chmurze wymagają dostępu do internetu. Korzyścią jest uniezależnienie użytkownika od:

- dostępu do komputera o dużej wydajności (procesor, RAM, dysk);
- konieczności instalowania i aktualizowania MATLAB-a i jego rozszerzeń oraz aktualizowania i konfigurowania systemu operacyjnego.

Pliki mogą być przechowywane w chmurze (usługa MATLAB Drive) i synchronizowane z plikami na lokalnym komputerze.

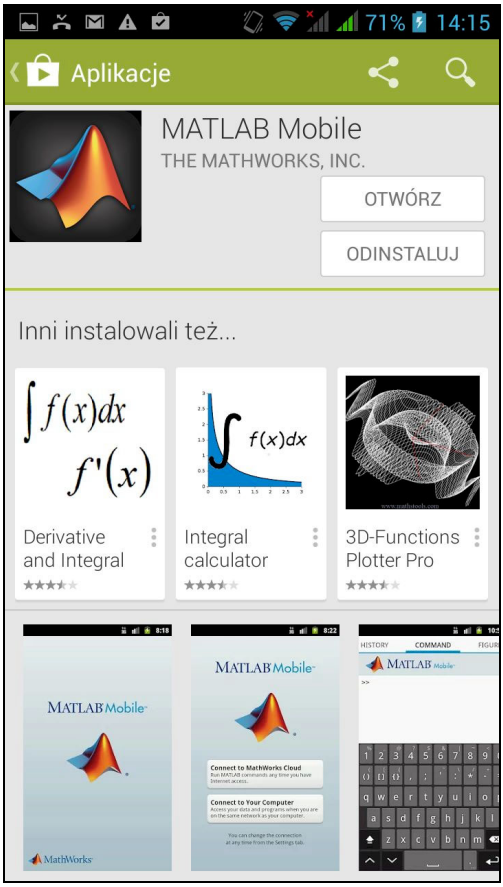
1.2.4.1. MATLAB Online

MATLAB Online udostępnia pulpit MATLAB-a w dowolnej przeglądarce internetowej pod adresem *matlab.mathworks.com*. Opis jest dostępny ma stronie *http://www.mathworks.com/products/matlab-online.html*. Przykład użycia MATLAB-a Online zamieszczono w rozdziale 11. Usługa jest dostępna dla użytkowników z indywidualną licencją, mających wykupioną usługę subskrypcji uaktualnień. Również licencje uczelniane zwane TAH i TSH dają dostęp do MATLAB-a Online. Zalecaną przeglądarką jest Google Chrome. Użytkownicy iPhone'ów, iPadów oraz systemu Android mogą korzystać tylko z MATLAB-a Mobile. Przykłady użycia MATLAB Online i MATLAB Drive zamieszczono w podrozdziałach 11.1 i 11.2.

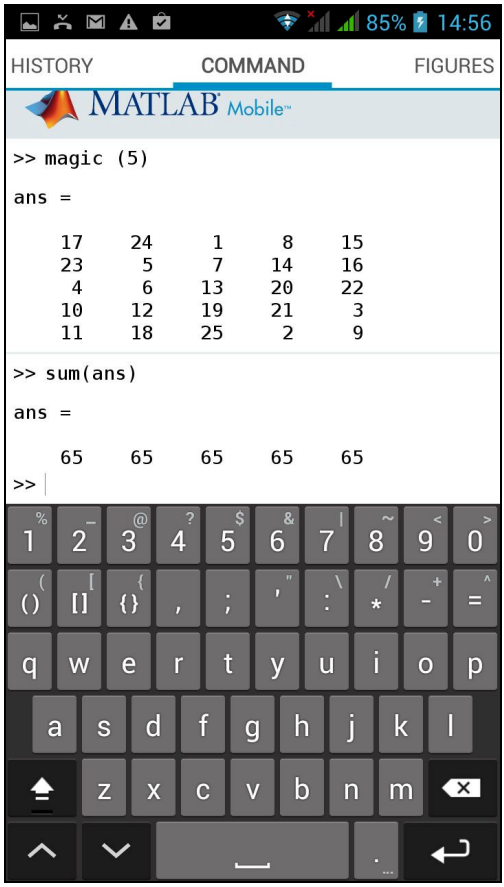
1.2.4.2. MATLAB Mobile na smartfony i tablety

MATLAB Mobile to lekki pulpit na iPhone, iPad lub urządzenie z systemem Android, który łączy się z oprogramowaniem MATLAB działającym w chmurze lub na lokalnym komputerze (rysunki 1.2.4.2-1 i 1.2.4.2-2). Pozwala to uruchamiać skrypty, tworzyć dane i wyświetlać wyniki. Usługi w chmurze są bezpłatne dla użytkowników mających wykupioną usługę subskrypcji uaktualnień. Połączenie z komputerem lokalnym wymaga wcześniejszego wykonania polecenia `connector on`. W aktualnej wersji MATLAB-a Mobile zapewniono:

- tworzenie i edycję plików w edytorze;
- wsparcie dla grafiki prezentacyjnej na iPad i Android;
- zgodność większości interfejsów GUI z iPhone'em, iPadem i Androidem;
- możliwość wykorzystania danych z czujników komórki lub tabletu:
  - przyspieszenie, prędkość kątowa i pole magnetyczne w trzech osiach;
  - orientacja (ang. *azimuth*, *pitch*, *roll*);
  - pozycja (ang. *latitude*, *longitude*, *altitude*, *horizontal accuracy*, *speed*, *course*).



Rysunek 1.2.4.2-1. *Pobieranie i instalowanie MATLAB-a Mobile na smartfonie z systemem Android*



Rysunek 1.2.4.2-2. *Okno MATLAB-a i jego klawiatura na ekranie smartfona*

Jako przykład (rysunek 1.2.4.2-2) na ekranie smartfona pokazano wyliczone w chmurze elementy kwadratu magicznego. Pokazano też, że suma elementów w każdej kolumnie tego kwadratu wynosi 65. Na smartfonie i na tabletach nie można korzystać z okna graficznego Simulinka. Można jednak użyć polecenia `sim`, aby w MATLAB-ie uruchomić model Simulinka.

1.2.5. MATLAB Distributed Computing Server

Klaster realizujący obliczenia równoległe pod nadzorem *MATLAB Distributed Computing Server* nie musi mieć zainstalowanego MATLAB-a. Oprogramowanie do pracy równoległej jest ładowane z komputera, na którym zainstalowano *Parallel Computing Toolbox*, MATLAB-a i inne potrzebne oprogramowanie, a koszt licencji MATLAB-a na klaster jest wliczony do ceny serwera. Powyższe reguły obowiązują także dla sieci komputerów i dla chmury realizującej obliczenia równoległe.

1.3. MATLAB i Simulink w internecie

Dystrybutor MATLAB-a w Polsce jest dostępny pod adresem *http://www.ont.com.pl*. Można tam uzyskać nieodpłatne materiały informacyjne, darmowe, czasowe licencje próbne (dla instytucji) oraz informacje o szkoleniach i konferencjach.

1.3.1. Witryna producenta www.mathworks.com

Pulpit pakietu MATLAB poprzez wstążkę i kartę *HOME/Resources/Help* prowadzi bezpośrednio do strony domowej producenta MATLAB-a, The MathWorks, Inc. W ten sposób bez wychodzenia z programu MATLAB uzyskuje się dostęp do opisanych poniżej stron witryny *http://www.mathworks.com*. Dostępne tam materiały (w języku angielskim) są podzielone na grupy widoczne po kliknięciu ikony menu  :

- *Products* — podano tu między innymi aktualny wykaz najnowszego oprogramowania (alfabetyczny i według zastosowań, z linkami do dokumentacji), opisy nowych produktów i wersji oprogramowania oraz produkty oferowane przez firmy współpracujące z producentem MATLAB-a.
- *Solutions* — przedstawiono tu możliwości MATLAB-a i przykłady jego zastosowania w celu poprawy efektywności produkcji w różnych gałęziach przemysłu. Omówiono tu obszary zastosowań MATLAB-a i zaprezentowano liczne przykłady jego wykorzystania.
- *Academia* — dostęp do materiałów edukacyjnych dla studentów i wykładowców, w tym:
  - *MATLAB for students* — omówienie wersji MATLAB-a dla studentów,
  - *MATLAB examples* — wybrane przykłady kodu MATLAB-a, modele Simulinka i materiały wideo,
  - *Tutorials* — poradniki, materiały wideo i bardzo przydatny 2-godzinny, bezpłatny kurs **MATLAB Onramp**, oferowany w ramach kursów **MATLAB Academy**,
  - *For Educators* — materiały dla wykładowców, w tym przykładowe programy nauczania z wielu przedmiotów wraz z materiałami dydaktycznymi,
  - link do *Makerzone* — zamieszczono tu między innymi linki do bezpłatnych kodów do sterowania robotem **LEGO MINDSTORMS** oraz sterowniki i oprogramowanie do popularnego sprzętu, jak Arduino® i wiele innych.
- *Support* — zamieszczono tu linki:
  - *Get Started* (pomoc, tutoriale i pliki do pobrania),
  - *Get Help* (dokumentacja, przykłady oraz pytania i odpowiedzi),
  - *Community* — przekierowanie, patrz niżej.
- *Community* (strona dla użytkowników) — przekierowanie do strony **MATLAB Central** z linkami:
  - *File Exchange* — wymiana plików (ang. *get and share code*). Strona ułatwia wymianę oprogramowania i wzajemny kontakt pomiędzy użytkownikami

MATLAB-a. Umieszczono tam bogate archiwum przykładów i pakietów przygotowanych przez autorów książek o MATLAB-ie, przez uczelnie, organizacje i osoby prywatne. *File Exchange* zawiera przykłady z tej książki.

- *MATLAB Answers* — pytania i odpowiedzi, grupa dyskusyjna.
- *Blogs* — blogi (ang. *learn from experts*).
- *Cody* — stały konkurs (*play coding game*), czyli problemy do rozwiązania.
- *Link Exchange* — pożyteczne linki.
- *Newsgroup* — grupa dyskusyjna *comp.soft-sys.matlab*.
- *ThingSpeak* (*Explore IoT Data*) — MathWorks udostępnia możliwość archiwizacji i przetwarzania danych pomiarowych w chmurze.

- *Events* — zamieszczono tu informacje na temat bezpłatnych konferencji i szkoleń internetowych (webinariów) oraz seminariów, szkoleń i konferencji klasycznych (płatnych i bezpłatnych).
- *About MathWorks* — linki w stopce strony, prowadzące do stron *Careers*, *Newsroom*, *Social Mission*, *About MathWorks*.

The MathWorks, Inc. stale ulepsza i zmienia swoje witryny internetowe. Może to spowodować, że niektóre podane tu informacje staną się nieaktualne.

1.3.2. MathWorks Account i logowanie do strony www.mathworks.com

Szczegółowe informacje i materiały szkoleniowe są dostępne po zalogowaniu się na *http://www.mathworks.com*. Potrzebne do zalogowania hasło otrzymuje się e-mailem po wypełnieniu krótkiej ankiety. Dla użytkowników MATLAB-a z wykupioną subskrypcją uaktualnień udostępniane są tam nowe wersje oprogramowania i możliwość pracy w chmurze.

1.3.3. Listy dyskusyjne i wyszukiwarki

W internecie jest kilka list dyskusyjnych dla użytkowników MATLAB-a:

- *http://www.matlab.pl* — polskie forum użytkowników pakietu MATLAB,
- *http://www.mathworks.com/matlabcentral/newsreader/*,
- *https://groups.google.com/forum/?hl=en#!forum/comp.soft-sys.matlab* oraz *http://comp.soft-sys.matlab.narkive.com/* (ta sama zawartość) — FAQ (ang. *frequently asked questions*), często zadawane pytania i odpowiedzi użytkowników.

Wpisując w wyszukiwarkach internetowych (np. Google) hasła zawierające słowo MATLAB, można znaleźć dodatkowe oprogramowanie, którego nie ma w ofercie producenta ani u dystrybutorów MATLAB-a.

1.4. MATLAB i Simulink w książkach

Co miesiąc ukazuje się kilka lub kilkanaście książek na temat MATLAB-a i jego zastosowań. Najlepsze książki są promowane i omawiane na *http://www.mathworks.com*. Najłatwiejszy dostęp do opisów książek można uzyskać, wchodząc na stronę *https://www.mathworks.com/support/books.html*. Można też wpisać słowo `books` w okienku *Search*. Następnie w kolejnym oknie należy wybrać *MATLAB and Simulink based books*.

Aby znaleźć potrzebną książkę, należy wskazać autora, słowo kluczowe lub język (np. *Polish*), w którym wydano książkę. Można też wybrać z listy tematykę poszukiwanej książki (lista jest w języku angielskim), np.:

- biologia i medycyna,
- chemia i inżynieria chemiczna,
- cyfrowe przetwarzanie sygnałów,
- ekonomia i finanse,
- elektronika,
- fizyka,
- przetwarzanie obrazów i wideo,
- identyfikacja systemów,
- matematyka,
- mechanika,
- modelowanie i symulacja,
- nauki o Ziemi (przyrodnicze),
- ogólnego przeznaczenia,
- pomiary i testowanie,
- programowanie i inżynieria programowania,
- sieci neuronowe i zbiory rozmyte,
- statystyka i funkcje losowe,
- systemy sterowania (automatyka),
- telekomunikacja,
- wykorzystanie MATLAB-a i Simulinka (w tym dziale można znaleźć tę książkę i poprzednie jej wydania).

W Polsce pakiet MATLAB jest szeroko wykorzystywany w edukacji oraz w wielu dziedzinach nauki i techniki, a także w medycynie, a nawet w muzyce. Wiele przykładów udanych zastosowań umieszczono na witrynie krajowego dystrybutora MATLAB-a *http://www.ont.com.pl*, w tym na stronach dotyczących konferencji i seminariów.

Rozdział 2.

Pierwsze kroki w programie MATLAB

Podane w tym rozdziale informacje umożliwią Czytelnikowi rozpoczęcie samodzielnego użytkowania pakietu MATLAB. Opisano tu wybrane funkcje matematyczne i graficzne oraz najważniejsze operacje, które można wykonać z pulpitu i w oknie *Command* — bez programowania. Zamieszczono też przykłady poleceń tworzących macierze o elementach rzeczywistych i zespolonych oraz przykłady użycia notacji kropkowej.

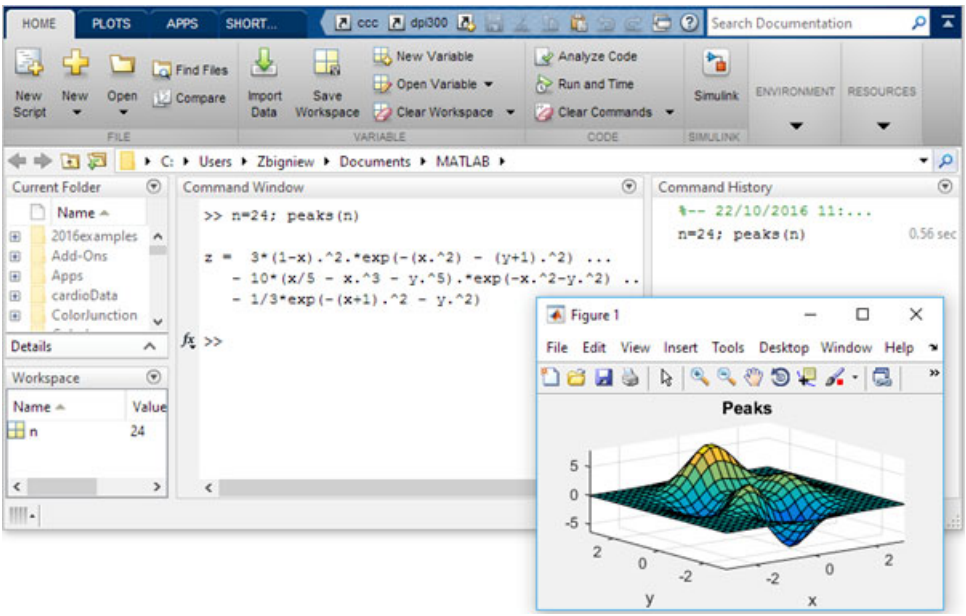
2.1. Pierwsza sesja w programie MATLAB

Ramy tej książki **nie pozwalają** na szczegółowe opisanie wszystkich dostępnych w MATLAB-ie funkcji. Dokładniejsze informacje o dostępnych funkcjach otrzymuje się za pomocą polecenia w postaci: `doc` *`nazwa_funkcji`*. Funkcje oraz operacje matematyczne opisano w podrozdziale 2.2.2 i dalszych.

2.1.1. Rozpoczęcie i zakończenie pracy z MATLAB-em

MATLAB-a można uruchomić przez podwójne kliknięcie myszą jego ikony  . W rezultacie na ekranie pojawi się pulpit MATLAB-a (rysunek 2.1.1-1). Plik *matlab.exe* jest zazwyczaj w folderze *C:\Program Files\MATLAB\R2017a\bin*. W miejsce *R2017a* należy wpisać oznaczenie aktualnie zainstalowanej wersji MATLAB-a. Pliki użytkownika są zapisywane w *Documents\MATLAB*.

Zakończenie sesji MATLAB-a następuje w wyniku zamknięcia okna pulpitu lub przez wykonanie polecenia `>> exit` albo równoważnego `>> quit` .



Rysunek 2.1.1-1. *Pulpit MATLAB-a oraz okno graficzne Figure z wykresem peaks(24)*

2.1.2. Pulpit MATLAB-a i jego okna

Pulpit MATLAB-a (ang. *desktop*, inne nazwy to: panel, ekran główny) pokazano na rysunku 2.1.1-1. Domyślne ustawienie okien pulpitu uzyskuje się przez wybór opcji *HOME/Layout/Default*; przydatne też jest ustawienie *Layout\Three Column*. W zależności od wybranych opcji na pulpicie MATLAB-a i poza nim mogą być otwarte okna:

- *Command Window* — okno do wpisywania i wykonywania poleceń MATLAB-a. Polecenia wpisuje się za znakiem zachęty >>, a zatwierdza się je klawiszem *Enter*. Znaku zachęty nie należy wpisywać samodzielnie. Pojawienie się znaku >> potwierdza gotowość MATLAB-a do realizacji kolejnego polecenia. Na rysunku 2.1.1-1 wpisano dwa polecenia w jednej linii: `n=24; peaks(n)`. W oknie *Command* pojawiają się też wyniki obliczeń i komunikaty MATLAB-a (tutaj program `peaks` wyświetlił wyrażenie matematyczne użyte do obliczeń i wykonania wykresu.
- W oknie *Workspace* widoczne są zmienne utworzone przez polecenia MATLAB-a oraz zmienne pobrane z plików poleceniem `load`. Tutaj mamy `n=24`. Zmienna `z` nie jest widoczna, bo jej wartość była obliczona i wykorzystana jedynie wewnątrz funkcji `peaks`.
- Okno *Current Folder* daje dostęp do plików w aktualnym folderze.
- Okno *Command History* pokazuje kopie wykonywanych wcześniej poleceń oraz umożliwia powtórne ich wykorzystanie.
- Okna *Editor* lub *Live Editor* (nie są teraz widoczne) są przeznaczone do pisania i uruchomiania programów.
- Dostęp do dodatkowych usług i okien można uzyskać, klikając ikonę w *Current Folder*.

Ikony umieszczone w górnej części okna MATLAB-a zajmują dużo miejsca. Można je ukryć (a w razie potrzeby przywrócić) przez podwójne kliknięcie dowolnej zakładki, np. *HOME*.

W prawym górnym narożniku okna może być umieszczony czarny trójkąt lub strzałki umożliwiające zakotwiczenie okna w pulpicie (ang. *dock*) lub jego wypchnięcie poza pulpit (ang. *undock*). Rozmiary okien i ich położenie można zmieniać, przeciągając myszą ich krawędzie.

Kliknięcie ikony z pytajnikiem (rysunek 2.1.1-1) otwiera okno systemu interaktywnej pomocy (patrz podrozdziały 2.1.5 i 2.7).

2.1.3. Przykład grafiki 3-D — funkcja peaks

Szybkość i jakość grafiki MATLAB-a można wstępnie ocenić, wykonując trójwymiarowy rysunek z użyciem polecenia `peaks`. Polecenie wpisuje się w oknie *Command* za znakiem zachęty >>:

```
>> peaks
```

Rezultatem wykonania polecenia `peaks` jest kolorowy trójwymiarowy rysunek powierzchni opisanej funkcją dwu zmiennych $z = f(x, y)$, widoczny w oddzielnym oknie graficznym na rysunku 2.1.1-1. Jeśli wykres nie jest widoczny na ekranie komputera, to może on być ukryty pod innym otwartym oknem. Kolory są zależne od wybranej mapy kolorów (palety barw). Można je zmienić na przykład poleceniem `colormap jet`. Więcej o mapach podaje polecenie `doc colormap`.

2.1.4. Przykłady poleceń MATLAB-a

Polecenia MATLAB-a należy wpisywać w oknie *Command* obok znaku zachęty, np.: `>>peaks`. Każde polecenie trzeba zatwierdzić (nacisnąć klawisz *Enter*).

Jeśli wpisywany tekst polecenia nie mieści się w jednej linii, można na końcu aktualnej linii wpisać spację i trzy kropki (…), nacisnąć klawisz *Enter* i kontynuować wpisywanie polecenia w kolejnej linii.

Zakończenie polecenia **średnikiem** (`;`) wstrzymuje wyświetlanie wyników na ekranie. Przyspiesza to wykonywanie poleceń i pozostawia w oknie *Command* więcej wolnego miejsca na kolejne polecenia. Zawartość okna *Command* można usunąć poleceniem `clc`.

Polecenie `ver` podaje aktualnie wersje używanego oprogramowania, w tym MATLAB-a, systemu operacyjnego, maszyny wirtualnej Java, oraz (co bardzo ważne) wykaz i wersje zainstalowanych toolboksów i innych rozszerzeń MATLAB-a.

Polecenie `demo` otwiera kartę *MATLAB Examples* w oknie *Help*. Umieszczono tam przykładowe programy, animacje i materiały szkoleniowe wideo w języku angielskim. Materiały szkoleniowe są też dostępne jako **webinaria** lub **tutoriale** na stronie *www.mathworks.com* — zakładki *Academia* i *Support*. Najłatwiejszy dostęp do tych stron jest poprzez menu *HOME/Help/Examples*, w grupie *RESOURCES*. Materiały szkoleniowe po polsku można pobrać ze strony *www.ont.com.pl/do-pobrania*.

Polecenie `bench` służy do testowania komputera. Polecenie `bench` mierzy czasy wykonania różnych zadań na dużych macierzach. Testowane są algorytmy `lu`, `fft`, `ode45`, `sparse` oraz **wykresy** 2D i **animacja** powierzchni 3D. Zadania przygotowano w taki sposób, aby czasy ich wykonania były prawie identyczne. **Duże różnice** w czasach wykonania kolejnych zadań najczęściej są spowodowane zbyt małą wielkością **pamięci RAM** lub nieodpowiednim **sterownikiem karty graficznej**.

Polecenia `dir`, `ls`, `del` i inne (z systemów operacyjnych Linux lub DOS) są wykonywane, jeśli mają swoje odpowiedniki w MATLAB-ie. Jeśli nie wykonają się, to należy je poprzedzić wykrzyknikiem, co przekieruje takie polecenie do systemu operacyjnego, np. `>> !dir`. Przydatne mogą być też polecenia `computer`, `perl`, `system`, `unix`, `dos`.

2.1.5. System pomocy doc i help — informacje wstępne

MATLAB automatycznie podpowiada, jakie parametry należy podać dla funkcji wpisywanych w oknie *Command* lub w **edytorze**. Po otwarciu nawiasu (po nazwie wpisywanej funkcji) pojawia się niewielkie okienko, a w nim przykłady listy parametrów dla danej funkcji.

Okno pomocy *Help* jest otwierane po kliknięciu ikony z pytajnikiem lub ikony *fx* poprzedzającej znak >> (rysunek 2.1.1-1, okno *Command Window*) bądź po wpisaniu polecenia `doc` z parametrami lub bez parametrów. Z kolei polecenie `help` wyświetla objaśnienia tekstowe w oknie *Command*. Parametrem dla `help` (w większości przypadków też dla `doc`) może być nazwa dowolnej funkcji MATLAB-a. Można również podać nazwę folderu zawierającego funkcje lub biblioteki MATLAB-a. Więcej w podrozdziale 2.7.

2.1.6. Zapisanie przebiegu sesji MATLAB-a

Zapisywanie przebiegu części lub całości sesji MATLAB-a (historia wykonanych poleceń i rezultatów ich działania) w zewnętrznym pliku rozpocznie się po wydaniu polecenia `diary` lub lepiej:

```
>>diary nazwa_pliku.m.
```

Zapis sesji wstrzymuje lub kończy polecenie `>>diary off`, a wznawia `>> diary on`. Wszystkie wykonane polecenia są także zapisywane w oknie *Command History*, skąd można je wygodnie pobrać przy użyciu myszy i powtórnie wykonać lub zapisać w pliku tekstowym zwanym **skryptem** (rozszerzenie *.m*) lub w pliku Live Editora (rozszerzenie *.mlx*). Szczegółowe informacje dotyczące skryptów podano w podrozdziałach 3.1 i 3.2.

2.2. Matematyka i proste wykresy

W MATLAB-ie można wykonać skomplikowane obliczenia oraz wizualizację wyników obliczeń w grafice dwu- i trójwymiarowej — bez potrzeby pisania jakichkolwiek programów. Operacje na **wektorach**, **macierzach** i **tablicach** są wykonywane bardzo szybko, gdyż odpowiednie **algorytmy są wbudowane** w MATLAB-a. Dodatkowo można przygotować animację i efekty dźwiękowe.

2.2.1. Wyrażenia matematyczne, zmienne, zmienna ans

MATLAB nie wymaga deklarowania zmiennych. Tworzenie zmiennych wybranego typu realizuje się przez przypisanie wartości do zmiennej lub przez użycie funkcji tworzących zmienne. Na przykład polecenie `x=1+2` tworzy zmienną `x` typu *double* i przypisuje jej wartość *n=3*. Polecenia wpisuje się w oknie *Command Window*, za znakiem zachęty `>>`.

Po naciśnięciu klawisza *Enter* wartość wyrażenia `1+2` jest obliczana i zostanie przypisana do nowo utworzonej zmiennej `x`. Jeśli zmienna `x` już istniała, to przyjmie nową wartość. Na ekranie pojawi się odpowiedź `x=3` oraz kolejny znak zachęty `>>` umożliwiający dalszy dialog. Utworzone zmienne są zapamiętane w przestrzeni roboczej MATLAB-a (ang. *workspace*).

Możliwe jest także wykonanie polecenia tak jak na kalkulatorze, bez podania nazwy zmiennej:

```
>> 1+2      % brak podstawienia, wynik będzie przypisany do zmiennej ans i wyświetlony na ekranie
```

Wtedy wynik obliczeń będzie przypisany do zmiennej `ans` w przestrzeni roboczej MATLAB-a. Na ekranie pojawi się:

```
ans =
     3
```

Znak `%` i tekst po tym znaku są komentarzem (nie są wykonywane).

2.2.2. Funkcje arytmetyczne i trygonometryczne

W wyrażeniach matematycznych można używać zaimplementowanych w MATLAB-ie operatorów i funkcji, np. funkcji **algebraicznych**, **trygonometrycznych**, **wykładniczych** i innych. Większość funkcji działa w razie potrzeby na liczbach zespolonych oraz na elementach tablicy. Oznacza to, że jeśli argumentem funkcji jest **wektor** lub **macierz**, to funkcja ta działa na każdym elemencie wektora lub macierzy. Skrócony opis wszystkich funkcji elementarnych podaje polecenie `doc elfun` (ang. *elementary math functions*).

2.2.2.1. Funkcje trygonometryczne

Funkcje trygonometryczne to: `sin`, `sind`, `sinh`, `asin`, `asind`, `asinh`, `cos`, `cosd`, `cosh`, `acos`, `acosd`, `acosh`, `tan`, `tand`, `tanh`, `atan`, `atand`, `atan2`, `atan2d`, `atanh`, `sec`, `secd`, `sech`, `asec`, `asecd`, `asech`, `csc`, `cscd`, `csch`, `acsc`, `acscd`, `acsch`, `cot`, `cotd`, `coth`, `acot`, `acotd`, `acoth`. Więcej informacji podaje polecenie `doc` *`nazwa_funkcji`*. Przykładowo:

- `acos`, `acosd`, `acosh`, `acot` — to odpowiednio: **arcus cosinus** w radianach, **arcus cosinus** w stopniach, **arcus cosinus** hiperboliczny, **arcus cotangens**,
- `acsc`, `acsch` — to odpowiednio **arcus cosecans**, **arcus cosecans hiperboliczny**.

2.2.2.2. Funkcje wykładnicze

Funkcje wykładnicze to: `exp`, `expm1`, `log`, `log1p`, `log10`, `log2`, `pow2`, `realpow`, `reallog`, `realsqrt`, `sqrt`, `nthroot`, `nextpow2`. Przykładowo:

`log10`, `log` — to odpowiednio logarytm **dziesiętny** i logarytm **naturalny**.

Do obliczenia logarytmu naturalnego w otoczeniu jedynki należy stosować funkcję `log1p`, gdyż dokładność obliczeń funkcji `log` spada bardzo w bliskim otoczeniu `x=1`. Na przykład dla `q=1e-10` wyniki obliczeń `log(1+q)` i `log1p(q)` różnią się już od ósmego miejsca po kropce dziesiętnej (zapis `1e-1` oznacza `0.1`, podobnie `e-3` to `0.001` itd.). Dla małych `q` funkcja `log1p(q)` zwraca lepsze przybliżenie wartości logarytmu naturalnego z liczby `(1+q)` niż wywołanie funkcji `log(1+q)`.

2.2.2.3. Funkcje dla liczb zespolonych

Większość funkcji MATLAB-a działa poprawnie na liczbach zespolonych. Więcej informacji o funkcjach `abs`, `angle`, `complex`, `conj`, `imag`, `real`, `isreal`, `cplxpair`, `unwrap` podaje polecenie `doc` *`nazwa_funkcji`*.

Przykład 2.2.2.3-1

Wynik działania funkcji `abs` zależy od rodzaju argumentu.

Dla argumentów o elementach rzeczywistych obliczana jest wartość bezwzględna każdego elementu, np. `abs(-5.2)` wynosi `5.2`.

Dla liczb zespolonych funkcja `abs` oblicza ich moduł, np. `abs(3+4i)` wynosi `5`.

Jeśli argumentem `abs` jest łańcuch, to otrzymuje się wektor kodów ASCII, które odpowiadają znakom składowym łańcucha, np. wektor `[77, 65, 84, 76, 65, 66, 45, 97]` to znaki ASCII tekstu *`'MATLAB-a'`*, uzyskane poleceniem `abs('MATLAB-a')`.

Uwaga: początek i koniec łańcucha jest oznaczony apostrofem, w druku mogą być przekłamania.

2.2.2.4. Zaokrąglanie

Do zaokrąglania wartości służą `fix`, `floor`, `ceil`, `round`, `mod`, `rem`, `sign`. Funkcje `round`, `fix`, `ceil`, `floor` — służą do zaokrąglania liczb, odpowiednio poprzez zaokrąglenie do najbliższej liczby całkowitej lub poprzez obcięcie odpowiednio w kierunku *zera*, $+\infty$, $-\infty$. Poniżej podano przykłady ich działania.

Przykład 2.2.2.4-1

Testowanie rezultatów zaokrąglania liczb: funkcja `round` zaokrągla do najbliższej liczby całkowitej:

```
>>xr= round([-1.7 -0.3 3.2 5.8 8.0])  % zaokrąglanie elementów wektora: liczby w nawiasie []
x =
    -2     0     3     6     8
```

Autorzy proponują przetestowanie pozostałych funkcji zaokrąglania liczb:

- `ceil` — zaokrąglenie w górę (ang. *ceiling* — sufit), np. `ceil(-1.7)=-1.0`, `ceil(1.7)=2.0`
- `fix` — zaokrąglenie przez obcięcie, np. `fix(-1.7)=-1.0`, `fix(+1.7)=+1.0`
- `floor` — zaokrąglenie w dół (ang. *floor* — podłoga), np. `floor(-1.7)=-2`, `floor(1.7)=1`

2.2.3. Obliczanie wartości wyrażeń matematycznych

W oknie *Command* po znaku zachęty >> należy wpisać nazwy i wartości danych. Następnie należy zapisać obliczane wyrażenie matematyczne z uwzględnieniem nazw funkcji dostępnych w MATLAB-ie.

Przykład 2.2.3-1

Oblicz wartość wyrażenia $x(t) = Ae^{(-\gamma t/2)}cos(\omega t + \phi)$, $\omega = 2\pi f$ dla zmiennej $t = 5$ i dla wartości współczynników $A = 1$, $\gamma = 0.2$, $f = 1$, $\phi = 0$.

```
>> A=1, gamma=0.2, f=1, omega =2*pi*f, fi=0, t=5
>> x=A*exp(-gamma*t/2)*cos(omega*t+fi)
 x =
    0.6065
```

Po podstawieniu wartości współczynników, wpisaniu powyższego wyrażenia i naciśnięciu *Enter* MATLAB niezwłocznie poda wynik obliczeń: `x=0.6065`. W zapisie danych i wyrażenia matematycznego litery greckie zastąpiono ich nazwami (`gamma`, `omega`, `fi`). Wartość liczby π jest zdefiniowana w MATLAB-ie pod nazwą `pi`. Funkcje $e^{-\gamma t/2}$ i $\cos(\omega t + \phi)$ zapisano z wykorzystaniem funkcji elementarnych MATLAB-a `exp()` i `cos()`. Drobna modyfikacja obliczanego wyrażenia (użycie notacji kropkowej `.*`), czyli `x=A.*exp(-gamma.*t/2).*cos(omega.*t+fi)`, pozwala wykonać obliczenia jednocześnie dla wielu wartości czasu `t` zadanego jako wektor, np. dla `t=0:0.1:5` (patrz podrozdziały 2.5.1 i 2.5.2). Więcej informacji o funkcjach elementarnych podaje polecenie `doc elfun`.

2.2.3.1. Liczby specjalne

W MATLAB-ie zdefiniowano liczby `pi`, `eps`, `realmax`, `realmin`, `intmax`, `intmin`, `flintmax`, `i`, `j`, `inf`, `Nan`. Są to stałe, zmienne lub funkcje o wartościach przydatnych w obliczeniach. Liczba `pi` to 3.141592653589793. Liczby `realmax`, `realmin`, `intmax`, `intmin` są obliczane dla wybranego typu zmiennych, np. `realmin('single')` = 1.1754944e-38, `inf` to wynik dzielenia 1/0, `nan` (ang. *not a number*) to wynik dzielenia 0/0. Funkcje `isnan`, `isinf`, `isfinite` mają wartość `TRUE`, jeśli ich argumentem jest odpowiednio `NaN`, liczba nieskończona, liczba skończona. Więcej informacji podaje polecenie `doc` *`nazwa_funkcji`*.

2.2.3.2. Funkcje specjalne

Funkcje z zakresu teorii liczb to `factor`, `isprime`, `primes`, `gcd`, `lcm`, `rat`, `rats`, `perms`, `nchoosek`, `factorial`. Przykładowo: `primes(200)` generuje wszystkie liczby pierwsze, nie większe od 200, `factor(200)` rozkłada liczbę 200 na czynniki pierwsze, a `factorial(4)` to silnia 4!, czyli $1 \cdot 2 \cdot 3 \cdot 4$. Dokładniejsze informacje o powyższych funkcjach podaje polecenie `doc` *`nazwa_funkcji`*.

2.2.4. Błędy w zapisie poleceń i wyrażeń matematycznych

Poniżej podano przykład próby obliczenia wartości wyrażenia $\text{arctg}\left(\log\left(\ln\left(4(3+2)\right)\right)\right)$.

Powyższy wzór zapisano błędnie jako polecenie MATLAB-a:

```
>> atan(log(ln(4*(3+2))))
```

Po naciśnięciu klawisza *Enter* MATLAB analizuje wprowadzone polecenie. Jeżeli nie może go wykonać, to podaje opis błędu, np.:

```
Undefined function or variable 'ln'
```

czyli: *nieokreślona funkcja lub zmienna 'ln'*. Rzeczywiście, w MATLAB-ie nie zdefiniowano funkcji `ln`. Logarytm naturalny oblicza funkcja `log`, a logarytm dziesiętny funkcja `log10`.

Aby poprawić błędy, należy nacisnąć klawisz ↑ (strzałka w górę). Pojawi się wtedy wydane wcześniej polecenie i będzie można skorygować popełnione tam błędy. Poprawnie napisane polecenie ma postać:

```
>> atan(log10(log(4*(3+2))))
```

Tym razem po naciśnięciu klawisza *Enter* uzyskuje się poprawny wynik:

```
ans = 0.4447
```

Ta sama wartość w formacie `long` będzie wyświetlona w pełnej dokładności:

```
>> format long
>> ans
ans =
   0.444673933514452
```

Domyślny format `short` będzie przywrócony przez polecenie `format` lub `format short`.

2.2.5. Powtórne użycie wcześniejszych poleceń

Korzystając ze strzałek, można odszukać dowolne wydane wcześniej polecenie. W tym celu po znaku zachęty należy wpisać kilka początkowych liter poszukiwanego polecenia, np.:

```
>> at
```

Po naciśnięciu klawisza ↑ (strzałka w górę) MATLAB odszuka wydane wcześniej polecenie rozpoczynające się od znaków `at`. W tym przykładzie jest to polecenie użyte w podrozdziale 2.2.4:

```
atan(log10(log(4*(3+2))))
```

Wydane wcześniej polecenie można wyświetlić, używając ↑ lub *Shift*+↑, bądź odszukać w oknie *History* i powtórnie wykonać po dwukrotnym kliknięciu wybranej linii historii lub po przekopiowaniu jej do okna *Command Window*.

2.2.6. Formaty wypisywania liczb

Wartość wyrażenia $\operatorname{arctg}\left(\log\left(\ln\left(4\left(3+2\right)\right)\right)\right)$ obliczanego w podrozdziale 2.2.4 wynosi `0.4447`, jeśli jest wyprowadzona w formacie domyślnym (`format short`). Aby zobaczyć wszystkie 15 cyfr znaczących (obliczenia są wykonane w podwójnej precyzji), należy zmienić format na `long` lub `long e` albo `long g`. Poniżej użyto polecenia `format long`. Można też wybrać potrzebny format z menu (grupa *ENVIRONMENT*) *HOME/Preferences/Command Window/Numeric Format/long*,

```
>> format long
>> atan(log10(log(4*(3+2))))
```

```
ans =
     0.444673933514452
```

Skutki zmiany formatu pokazuje też poniższy przykład.

Przykład 2.2.6-1

Wyświetlanie liczby $\pi = 3.141592653589793$ w różnych formatach:

```
format    % przywrócenia domyślnego formatu
pi,    format long, pi,      format rat, pi
```

Na ekranie otrzymamy kolejno: *3.1416 3.141592653589793*. Dla formatu `rat` otrzymamy ułamek: *355/113* (*rat* to skrót od *rational* — wymierny).

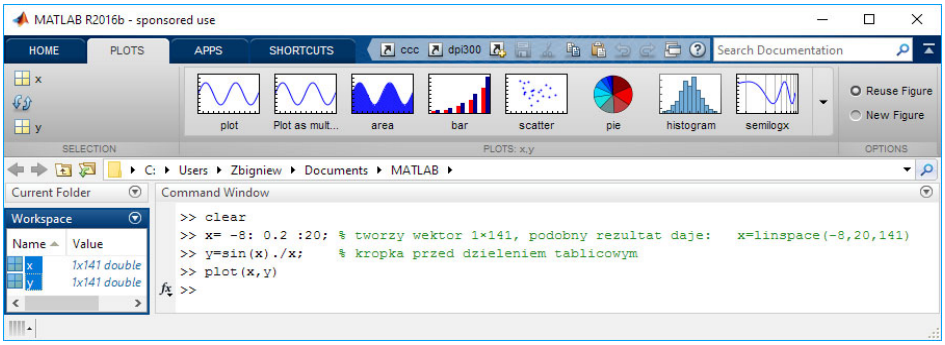
2.2.7. Wizualizacja danych i wyników obliczeń bez programowania

MATLAB umożliwia wizualizację danych i wyników obliczeń w grafice dwu- i trójwymiarowej — bez potrzeby pisania jakichkolwiek programów. Wykres można wykonać:

- poprzez zaznaczenie myszą danych do wizualizacji i wskazanie rodzaju wykresu,
- poprzez wpisanie w oknie *Command* nazwy funkcji graficznej i jej parametrów,
- poprzez umieszczenie w programie nazwy funkcji graficznej i jej parametrów.

Osie rysunku, punkty, linie, powierzchnie oraz wszelkie opisy rysunku są obiektami, co umożliwia łatwą modyfikację każdego z elementów gotowego rysunku.

Najprostszym sposobem jest zaznaczenie danych do wykresu w oknie *Workspace* pulpitu MATLAB-a. Następnie (w zakładce *PLOTS*) należy kliknąć ikonę z wizerunkiem potrzebnego wykresu, jak pokazano na rysunku 2.2.7-1.



Rysunek 2.2.7-1. *Przygotowanie danych do wykresu. Widoczne ikony grafiki prezentacyjnej, dostępne w zakładce PLOTS pulpitu MATLAB-a. W oknie Workspace zaznaczono zmienne x, y*

Przykład 2.2.7-1

Przygotowanie danych do testowania grafiki i przypisanie ich do wektorów `x`, `y`:

```
x= -8: 0.2 :20;    % tworzy wektor 1×141, podobny rezultat daje: x = linspace(-8,20,141)
y=sin(x)./x        % kropka przed dzieleniem tablicowym ./
```

Kolejnym krokiem jest zaznaczenie potrzebnych do wykresu zmiennych `(x,y)` w oknie *Workspace* oraz kliknięcie ikony (zatwierdzenie) wybranego wykresu w zakładce *PLOT*

na pulpicie MATLAB-a. Po zatwierdzeniu pojawi się w oknie *Command* dodatkowa linia, np. `plot(x,y)`.

Uwagi:

- W poleceniu `x= -8: 0.2 :20` użyto notacji dwukropkowej (podrozdział 2.4).
- We wzorze `y=sin(x)./x` użyto dzielenia tablicowego (notacja kropkowa `./`). Dzięki temu wektory `x` i `y` mają tę samą liczbę elementów i można wykonać wykres. Dokładniej opisano to w podrozdziałach 2.5.1 – 2.5.4).
- Ikony wykresów są *aktywne* (kolorowe) dopiero od chwili zaznaczenia w oknie *Workspace* pierwszej zmiennej z *danymi do wykresu*.
- Wpierw należy zaznaczyć zmienną dla osi poziomej `x`, a potem (przy wciśniętym klawiszu *Ctrl*) dla osi pionowej `y`.
- Wykres będzie wykonany niezwłocznie po kliknięciu ikony wybranego wykresu.
- W razie potrzeby można zamienić osie współrzędnych wykresu, klikając ikonę  .
- Opis funkcji graficznej wyświetla się po zatrzymaniu kursora myszy na ikonie wykresu.

W przykładzie użyto ikony *plot*. Dokładniejsze opisy wykresów w MATLAB-ie podano w rozdziale 4.

2.2.8. Wykresy funkcji dwu- i trójwymiarowej z użyciem fplot i fplot3

Wykres funkcji można wykonać, używając `plot`, ale wymaga to tablicowania rysowanej funkcji. Funkcja `fplot` nie wymaga tablicowania.

Pierwszym parametrem funkcji `fplot` jest funkcja, która będzie przedstawiona na wykresie. Zaleca się, aby były to funkcja **anonimowa** (patrz przykład 2.2.8-1 i podrozdział 3.2.7) lub **uchwyt tej funkcji**.

Drugim parametrem funkcji graficznej `fplot` może być przedział zmiennej niezależnej, np. `[0,20]`. Domyślna wartość przedziału wynosi `[-5,5]`. Więcej informacji podaje polecenie `doc fplot`.

Przykład 2.2.8-1 (dla zaawansowanego programisty)

Wykonanie wykresu funkcji $x(t) = Ae^{(-\gamma t/2)}cos(\omega t + \phi)$, $\omega = 2\pi f$ dla zmiennej $t = [0, 20]$ i dla wartości współczynników $A = 1$, $\gamma = 0.2$, $f = 1$, $\phi = 0$ — jak w przykładzie 2.2.3-1. W pierwszej kolejności podstawia się wartości współczynników do zmiennych, następnie tworzony jest wykres.

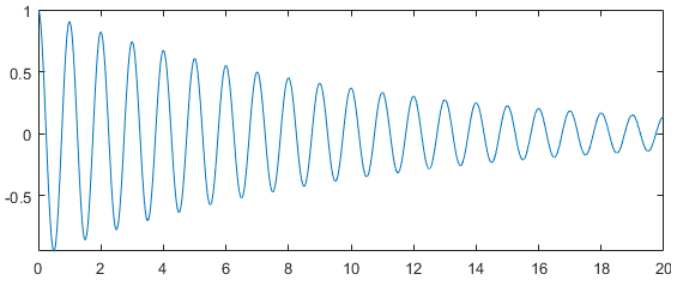
```
A=1, gamma=0.2,f=1,omega =2*pi*f,fi=0,
fplot  (@(t)  A.*exp(-gamma.*t./2).*cos(omega.*t+fi), [0,20] )   % @(t) definiuje funkcję
                                                                 % anonimową
```

Pierwszym parametrem `fplot` jest funkcja anonimowa, którą tworzy się, poprzedzając wyrażenie matematyczne przez `@(t)` i spację, gdzie `t` jest zmienną niezależną wyrażenia matematycznego.

Zaleca się, aby parametrami `fplot` nie były funkcje, ale ich *uchwyty* — jak pokazano poniżej:

```
A=1;, gamma=0.2; f=1,omega =2*pi*f; fi=0; t=5;
h1 = @(t) A.*exp(-gamma.*t./2).*cos(omega.*t+fi)    % zdefiniowanie funkcji anonimowej
                                                    % i jej uchwytu h1
fplot (h1, [0,20])
```

Dla obu wariantów uzyskuje się ten sam wykres (rysunek 2.2.8-1). W starszych wydaniach MATLAB-a zamiast `fplot` należy użyć funkcji graficznej `ezplot`.



Rysunek 2.2.8-1. *Wykres fplot z użyciem funkcji anonimowej*

Uchwyt `h1` przechowuje nie tylko funkcję, ale też wartości jej parametrów: `A`, `gamma`, `f`, `fi`, Utworzenie funkcji anonimowej i jej uchwytu potwierdza komunikat:

```
h1 =
  function_handle with value:
    @(t)A.*exp(-gamma.*t./2).*cos(omega.*t+fi)
```

Zwraca się uwagę, że:

- funkcja anonimowa nie ma nazwy, ale może mieć uchwyt;
- w wyrażeniu matematycznym użyto operacji tablicowych (np. .*) „z kropką” — zamiast operacji macierzowych (np. *). Operacja tablicowa nie zmienia wymiarów macierzy i wektorów, co umożliwia wykonanie wykresu. Dokładniej opisano to w podrozdziałach 2.5.1 – 2.5.4.

Funkcję `fplot` można wywołać na wiele sposobów, np. w celu narysowania okręgu można wykonać:

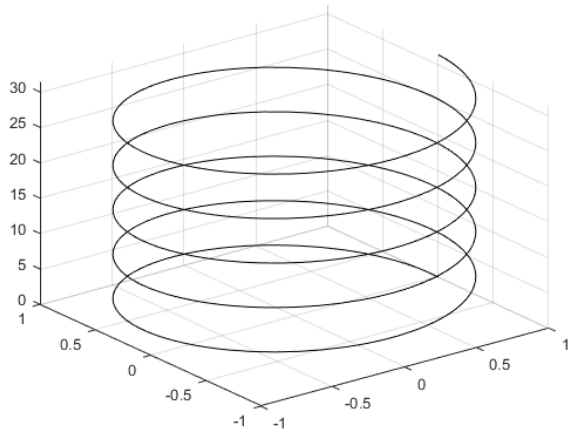
```
fplot(@(t) cos(t),@(t) sin(t), [0,2*pi])
```

Dodatkowe informacje o funkcji `fplot` podaje polecenie `doc fplot`.

Wykres trójwymiarowy może być wykonany za pomocą funkcji `fplot3` lub `plot3`. Na rysunku 2.2.8-2 funkcje $x = \cos(t)$ oraz $y = \sin(t)$ rysują koło na płaszczyźnie `xy`.

Funkcja `z=t` przesuwa kolejne punkty koła w górę i rysuje sprężynę. Parametr `t` jest wyrażony w radianach i zmienia się w zakresie od 0 do 10π, czyli będzie wyrysowane 5 kół. Efektem równoczesnego działania tych funkcji jest trójwymiarowy rysunek spirali rozciągniętej w pionie. Rysunek uzyskano po wykonaniu polecenia z funkcjami anonimowymi:

```
fplot3(@(t) cos(t),@(t) sin(t), @(t) t, [0,10*pi], 'k')
```



Rysunek 2.2.8-2. *Wykres trójwymiarowy wykonany przy użyciu fplot3 i trzech funkcji anonimowych*

Przykład 2.2.8-2 (bez funkcji anonimowej)

Można też utworzyć wektor zmiennej niezależnej `t` i użyć funkcji graficznej `plot3`. Jednak z uwagi na planowane zmiany w MATLAB-ie zaleca się użycie `fplot` funkcji anonimowych.

```
t=0:pi/20:10*pi;  plot3(cos(t),sin(t),t, 'k')
```

Więcej informacji dotyczących tworzenia wykresów funkcji znajduje się w rozdziale 4. Można też wejść do systemu pomocy poprzez `doc fplot3` i wyszukać potrzebne informacje.

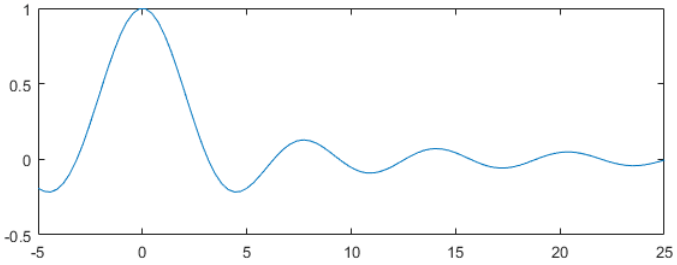
2.2.9. Wektory, tablice i wykresy plot

Użycie funkcji `plot` wymaga przygotowania danych numerycznych, czyli tablicowania funkcji. Zazwyczaj tworzy się wektor `x`, po czym dla każdego elementu tego wektora oblicza się wartość `y = f(x)`. W MATLAB-ie korzystamy z operacji na wektorach, tablicach i macierzach.

Przykład 2.2.9-1

Wykonanie wykresu funkcji $y = sin(x)./x$ w przedziale $x \in [-5,+25]$ (rysunek 2.2.9-1):

```
x= linspace(-5,25);     % linspace(X1, X2, N) generuje N-elementowy wektor
y = sin(x)./x;     % oblicza y=sin(x)./x dla każdego elementu wektora x. Kropka (./) jest potrzebna
plot(x,y)     % wykonanie wykresu
```



Rysunek 2.2.9-1. *Wykres funkcji sin(x)./x*

MATLAB ma wbudowane operacje na wektorach, tablicach i macierzach. Pierwszy element wektora `x` jest równy `-5`, a ostatni `25`. Użycie kropki przed operacją dzielenia `y=sin(x)./x` powoduje dzielenie każdego elementu wektora sin(`x`) przez odpowiadający mu element wektora `x`, co gwarantuje, że wymiary wektorów `x` i `y` będą identyczne, a to jest pomocne przy wykonywaniu wykresu. Wyniki obliczeń (wartości wektorów `x`, `y`) nie pojawiają się na ekranie z uwagi na użycie średnika (`;`). Wyświetlanie niepotrzebnych wyników pośrednich znacznie wydłużyłoby czas obliczeń. Funkcja graficzna `plot` otwiera okno graficzne i rysuje w nim wykres zadany przez wektory `x`, `y` o tej samej długości. Zwraca się uwagę na konieczność użycia dzielenia tablicowego (z kropką `./`) — patrz podrozdziały 2.5.1 i 2.5.2.

Przykład 2.2.9-2

Wykonanie wykresu funkcji $x(t) = Ae^{-\gamma t/2}\cos(\omega t + \phi)$, $\omega = 2\pi f$ z użyciem `plot`.

W pierwszej kolejności dla zmiennej niezależnej $t \in [0, 20]$ wygenerowano wektor `t` o 401 elementach (polecenie `t=0:0.05:20;`), po czym obliczono 401 wartości funkcji $x(t)$. Reasumując, należy wykonać następujące polecenia:

```
t=0:0.05:20;    % generuje wektor o wartościach od zero do 20, z krokiem 0.05
A=1; gamma=0.2; f=1; omega =2*pi*f; fi=0;
x=A*exp(-gamma.*t/2).*cos(omega.*t+fi);
plot(t,x)    % wykonanie wykresu
```

Polecenia `A=1; gamma=0.2; f=1;…` nadają wartości parametrom wyrażenia matematycznego. Polecenie `x=A*exp(-gamma.*t/2).*cos(omega.*t+fi);` zostało dostosowane do wymogów mnożenia tablicowego wektorów (z kropką `.*`) — patrz podrozdziały 2.5.1 i 2.5.2.

Wykres wykonany za pomocą funkcji `plot` jest prawie identyczny z rysunkiem 2.2.9-1 i nie jest tutaj pokazany (plot nie wpisuje tytułu rysunku). Więcej informacji o funkcji `plot` można uzyskać poprzez `doc plot`.

2.2.10. Przykład rozwiązania układu równań algebraicznych

Rozwiązywanie układu równań algebraicznych, nawet o bardzo dużej liczbie zmiennych (mogą to być nawet liczby zespolone), jest w MATLAB-ie wykonywane szybko i dokładnie bez potrzeby programowania.

Przykład 2.2.10-1

Obliczanie wektora `x` będącego rozwiązaniem układu trzech równań algebraicznych:

$$\begin{aligned} -x_1 + 3x_2 + 2x_3 &= 0 \\ x_1 + 0x_2 - x_3 &= 1 \\ 2x_1 - 3x_2 + x_3 &= 3 \end{aligned} \qquad (2.2.10\text{-}1)$$

Powyższe równanie wyrażamy macierzowo jako *A*x = b*. Poniżej podano sposób zapisania danych: macierzy `A` i wektora `b`:

```
>> A=[-1 3 2; 1 0 -1; 2 -3 -1], b=[0 1 3]
```

Macierz `A` i wektor `b` mają postać:

```
A =
    -1     3     2
     1     0    -1
     2    -3    -1
b =
     0     1     3
```

Wektor `b` jest poziomy. Aby uzyskać zgodność z podanym wyżej zapisem macierzowym, użyty będzie wektor transponowany, czyli zostanie zapisany jako `b'` — z apostrofem (`'`). Jeśli wektor `b` jest liczbą zespoloną, przed apostrofem należy wpisać kropkę, czyli `b.'` (patrz podrozdział 2.5.6). Rozwiązanie `x` otrzymuje się jako wynik zdefiniowanego w MATLAB-ie dzielenia lewostronnego: `x=A\b'`.

```
>> x=A\b'          % dzielenie lewostronne oblicza rozwiązanie równania Ax=b)
```

Rozwiązywaniem układu równań algebraicznych (2.2.10-1) jest $x_1 = 2,\ \ x_2 = 0,\ \ x_3 = 1$.

Użycie macierzy odwrotnej `inv(A)` nie jest zalecane, gdyż obliczenia są bardziej czasochłonne, a wynik jest mniej dokładny. Przykład rozwiązania układu *równań zespolonych* podano w podrozdziale 8.1.4.

2.2.11. Konwersja układu współrzędnych

Funkcje `cart2sph`, `cart2pol` służą do zmiany współrzędnych kartezjańskich odpowiednio na sferyczne i biegunowe. Funkcje `pol2cart`, `sph2cart` służą do zmiany współrzędnych biegunowych (*polar*) lub sferycznych (*spherical*) na kartezjańskie.

Przykład 2.2.11-1

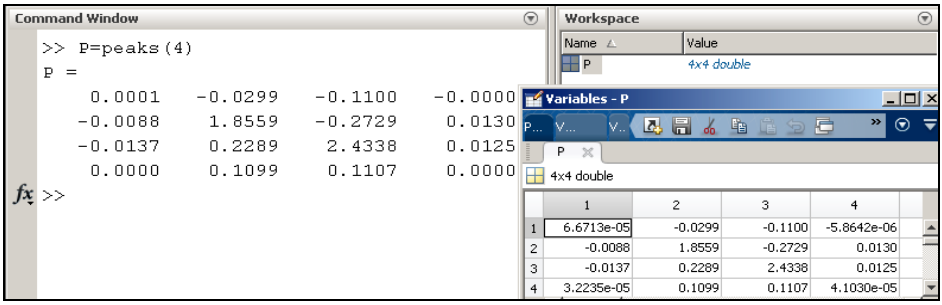
Przeliczenie radianów na stopnie.

```
>> [fi,a]=cart2pol(1,1)    % oblicza kąt i długość przekątnej kwadratu o bokach (1,1)
fi =
    0.7854
a =
    1.4142
>> fi*180/pi    % przeliczenie radianów na stopnie
ans =
    45
```

Przekątna kwadratu o boku 1 cm ma długość 1.4142 i kąt 0.7854 radiana, czyli 45º.

2.3. Zmienne w programie MATLAB

Zmienne są przechowywane w przestrzeni roboczej MATLAB-a. Nazwy zmiennych są widoczne w oknie *Workspace*, mogą też być wyświetlone w oknie *Command* za pomocą poleceń `who` i `whos`. W oknie *Workspace* możemy kliknąć ikonę, która znajduje się przed nazwą każdej zmiennej. Powoduje to otwarcie okna edytora zmiennych *Variables* (rysunek 2.3-1) i daje możliwość zmiany wartości elementów wybranego wektora lub tablicy.



Rysunek 2.3-1. *Pulpit MATLAB-a. Otwarte okna: Command Window, Workspace i Variables. Liczba jest wypisana w różnych formatach*

2.3.1. Tworzenie wektorów, tablic i macierzy

Zmiennych w MATLAB-ie nie deklaruje się. Są one tworzone i zapamiętane w momencie pierwszego użycia ich nazwy lub przypisania im nowej wartości. Aktualna wartość zmiennej jest zapamiętana w przestrzeni roboczej i jest widoczna w oknie *Workspace* na pulpicie MATLAB-a. Zmienne są też tworzone w trakcie wykonywania obliczeń oraz mogą być wczytane z pliku (podrozdział 2.3.3).

Przykład 2.3.1-1

Utwórz zmienną `a` i przypisz do niej wektor o wartościach: `1`, `2`, `3` do `6`.

```
>>a = [1 2 3 4 5 6]
```

Elementy wektora poziomego oddziela się spacją lub przecinkiem, a całość umieszcza się w nawiasie **kwadratowym**. Potwierdzeniem wykonania polecenia jest wyświetlenie na ekranie nazwy zmiennej i jej aktualnej wartości w postaci:

```
a =
1    2    3    4    5    6
```

Jeśli wartości elementów wektora są równo odległe, używa się notacji **dwukropkowej** `a = 1:6;` (patrz podrozdział 2.4.1).

Przykład 2.3.1-2

Przy tworzeniu macierzy z klawiatury każdy wiersz macierzy wpisuje się jako wektor poziomy. Wiersze oddziela się średnikiem (`;`) lub przejściem do nowego wiersza (klawisz *Enter*). Całość umieszcza się w nawiasie **kwadratowym**. Jeśli na końcu polecenia nie ma średnika, utworzona macierz będzie wyświetlona na ekranie.

```
>> A = [1 2 0; 2 5 -1; 4 10 -1]
A =
     1     2     0
     2     5    -1
     4    10    -1
```

Przykład 2.3.1-3

Zmiennej można przypisać ciąg znaków, czyli łańcuch (ang. *string*):

```
>> s = 'MATLAB'
s =
MATLAB
```

2.3.2. Zapisywanie zmiennych w plikach — MAT-plik i plik ASCII

Zmienne z przestrzeni roboczej są tracone z chwilą zakończenia pracy z MATLAB-em. Jeśli są one potrzebne, to należy je zapisać do pliku poleceniem `save`. Przykładowo polecenie `save a1` zapisuje wszystkie aktualne zmienne z przestrzeni roboczej do pliku *a1.mat* w postaci binarnej, skompresowanej i z zachowaniem pełnej ich dokładności.

Polecenie `save a1 A B C` (bez przecinków) zapisze do pliku *a1.mat* tylko zmienne `A B C`. MAT-pliki są często używane do przechowywania dużych ilości danych pomiarowych oraz do wymiany danych pomiędzy MATLAB-em i innymi programami, np. *dSPACE*, *Dymola*, *COMSOL*.

Polecenie `save a1.dat A -ascii` zapisuje do pliku *a1.dat* w formacie ASCII wartości tylko jednej zmiennej `A`. Rozszerzenie nazwy pliku jest dowolne, ale zwyczajowo używa się *.dat*. Dodatkowe użycie opcji `-double` (czyli `save a1.dat A -ascii -double`) zwiększy dokładność zapisu do 16 cyfr. Dane w kodzie ASCII zajmują dużo miejsca, gdyż nie są skompresowane, ale można je przeglądać i modyfikować w dowolnym edytorze tekstowym. Więcej szczegółów podaje polecenie `doc save`.

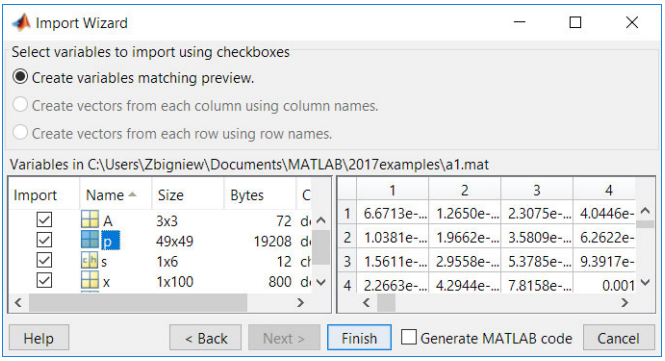
Plik z danymi można też utworzyć poprzez kliknięcie odpowiedniej ikony (lub kilku ikon) w oknie *Workspace*. Można też wybrać zakładkę *Home* i opcję *SaveWorkspace* z menu pulpitu MATLAB-a. W obu przypadkach następuje otwarcie okna, które zawiera listę już istniejących MAT-plików i okienko dialogowe, do którego wpisuje się nazwę nowego MAT-pliku (tzn. pliku z rozszerzeniem *.mat*, dopisywanym automatycznie).

2.3.3. Wczytywanie zmiennych z pliku

Utworzony w poprzednim podrozdziale MAT-plik *a1.mat* można wykorzystać w kolejnych sesjach MATLAB-a. Można też kliknąć *Import Data* na karcie *HOME* pulpitu MATLAB-a lub wczytać go poleceniem:

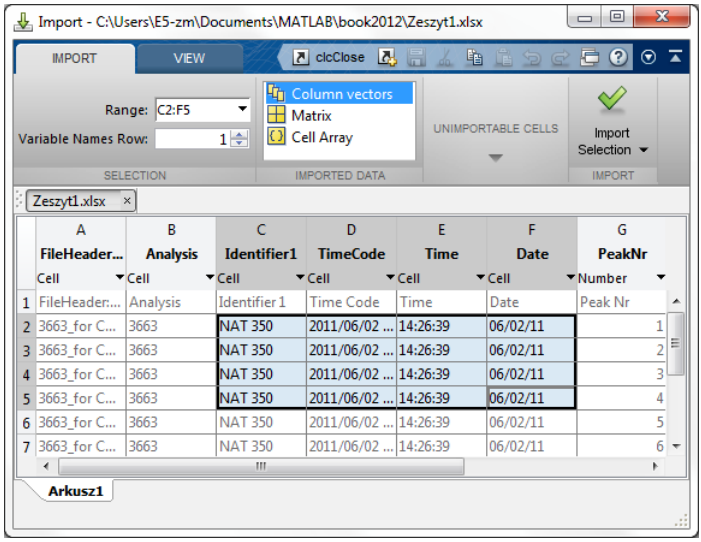
```
load a1
```

Innym sposobem wczytania danych jest podwójne kliknięcie wybranego pliku w oknie *Current Folder* w panelu MATLAB-a. MAT-plik może zawierać kilka zmiennych, w tym wektory i macierze. Jego zawartość będzie wtedy pokazana w oknie *Import Wizard* (rysunek 2.3.3-1).



Rysunek 2.3.3-1. *Okno Import Wizard — wizualizacja danych z MAT-pliku MATLAB-a*

Tym sposobem można wczytać dane w różnych formatach: dane tekstowe, MAT-pliki, arkusze Excela, pliki graficzne, a także pliki audio i wideo oraz większość automatycznie zarejestrowanych wyników badań i pomiarów laboratoryjnych. *Import Wizard* inteligentnie wczytuje dane, rozpoznaje ich formaty i separatory danych (spacje, tabulacje, znaki: :, .). Wczytane dane są zazwyczaj poprawnie umieszczane w macierzy `data`, a nagłówki kolumn i inne opisy są wpisane do zmiennych `textdata` i `colheaders`. Wygląd okna zależy od wczytywanego pliku. Na rysunku 2.3.3-2 pokazano okno *Import Wizard* dla Excela.



Rysunek 2.3.3-2. *Okno Import Wizard — wizualizacja danych z Excela*

2.3.4. Usuwanie zmiennych z przestrzeni roboczej, czyszczenie ekranu

Wszystkie zmienne z przestrzeni roboczej usuwa polecenie `clear`. Najłatwiej jest kliknąć myszą (prawym przyciskiem) wewnątrz okna *Workspace* i potwierdzić polecenie `clear`. Wybrane zmienne można też usunąć poleceniem `clear` w postaci:

```
>>clear nazwy_zmiennych_oddzielone_spacjami
```

Polecenie `clear` ma wiele opcji: `global`, `fun`, `all`, `import`, `classes` (patrz `doc clear`). Polecenie `clear` jest często używane przed rozpoczęciem nowych obliczeń lub gdy MATLAB zakomunikuje brak wolnej pamięci. Należy unikać polecenia `clear all`, gdyż usuwa ono również kody półskompilowane, co może spowolnić pracę MATLAB-a.

Większość okien tekstowych MATLAB-a można wyczyścić, klikając myszą (prawym przyciskiem) wewnątrz okna i potwierdzając polecenie jego wyczyszczenia. Okno *Command* najczęściej czyści się poleceniem `cls`.

Wszystkie okna graficzne zamyka polecenie `close all`. Aktualne (czyli ostatnio użyte, np. kliknięte) okno graficzne zamyka się poleceniem `close` bez parametrów; `close 3` zamyka okno *Figure 3*.

2.4. Dwukropek — operator generowania wektorów

Dwukropek (:) jest jednym z częściej używanych operatorów w MATLAB-ie. Jest on wykorzystywany do niejawnego indeksowania, na przykład przy tworzeniu wektorów o równomiernie rozłożonych elementach. Umożliwia też wyselekcjonowanie żądanych wierszy, kolumn lub elementów tablic. Jest stosowany w algorytmach iteracyjnych, a w szczególności w instrukcji `for` lub jako jej alternatywa.

2.4.1. Generowanie wektorów — dwukropek, linspace, logspace

Wektor o elementach wzrastających, z krokiem równym 1, tworzy się, podając wartość pierwszego elementu oraz najwyższą dopuszczalną wartość ostatniego elementu. Pomiędzy te wartości wstawia się dwukropek (:):

```
>> n=5; i=1:n
i =
     1     2     3     4     5
```

Jeśli wartość kroku ma być inna niż 1, to wpisuje się ją pomiędzy dwoma dwukropkami (krok może być ujemny), np.:

```
>> i=0:0.3:1
i =
         0    0.3000    0.6000    0.9000
```

Innym, wygodniejszym sposobem utworzenia podobnego wektora jest użycie funkcji `linspace`:

```
>> linspace(0,1,4)
ans =
         0    0.3333    0.6667    1.0000
```

Liczba `4` oznacza tu liczbę elementów tworzonego wektora. Skalę logarytmiczną generuje `logspace`:

```
>> A=logspace(0,3,4)
A =
       1            10           100          1000    .
```

Pierwsze dwa parametry funkcji `logspace(0,3,4)` oznaczają wartość $10^0 = 1$ oraz $10^3 = 1000$, trzeci parametr to liczba elementów (domyślnie 50). Sposób użycia `freqspace` podaje polecenie `doc freqspace`.

2.4.2. Wybór żądanych wierszy, kolumn i elementów tablicy

MATLAB umożliwia łatwe i precyzyjne wybranie potrzebnych wierszy, kolumn i innych fragmentów macierzy. Przy braku wprawy warto utworzyć niewielką macierz `A=magic(4)` i poeksperymentować. Ogólnie:

1. Dwukropek pomiędzy liczbami lub nazwami zmiennych oznacza `od:do`.
2. Sam dwukropek (`:`) oznacza wszystkie dopuszczalne wartości.
3. Słowo `end` oznacza ostatni element.

Przykłady ogólne i indeksy podaje się w nawiasach okrągłych:

- `A(i,j)` oznacza element w `i`-tym wierszu oraz jednocześnie w `j`-tej kolumnie.
- `A(:, j)` oznacza wybranie całej `j`-tej kolumny macierzy `A`.
- `A(:, j:k)` oznacza wszystkie elementy kolumn `A(j)`, `A(j+1)`, …, `A(k)`. Liczba `j` jest numerem pierwszej, `k` numerem ostatniej wypisywanej kolumny macierzy `A`. Jeśli `j>k`, to uzyskuje się macierz pustą.
- `A(i, :)` oznacza wybranie całego `i`-tego wiersza macierzy `A`.
- `A(i, j:k)` oznacza wybranie z macierzy `A` tylko tych elementów, które należą do `i`-tego wiersza i jednocześnie znajdują się w kolumnach `j`, `j+1`, …, `j+k`.
- `A(:)` tworzy wektor pionowy zawierający wszystkie elementy macierzy `A`. Najwyżej jest umieszczona kolumna pierwsza, a pod znajdują się nią kolejne, aż do ostatniej kolumny.
- `A(j:k)` oznacza odwołanie się do elementów macierzy `A(:)`, od elementu o indeksie `j` aż do indeksu `k`. Gdy `j>k`, to powstaje wektor pusty.
- Użycie nawiasów kwadratowych pozwala wskazać wiersze i kolumny w dowolnej kolejności. Przykładowo `A(:,[3,1])` oznacza wybór kolumn `3` i `1`.

W MATLAB-ie **nie ma elementu** o indeksie *zero*. Powinny o tym pamiętać osoby programujące w języku C.

2.4.3. Macierze — przykłady użycia notacji dwukropkowej

Poniżej podano przykłady użycia notacji dwukropkowej do wybierania wierszy, kolumn i elementów zdefiniowanej tu macierzy `A`:

```
>>A=[1 2 3; 4 5 6; 7 8 9]    % zdefiniowano macierz A
A =
     1     2     3
     4     5     6
     7     8     9
```

Przykład 2.4.3-1

Wybór kolumn macierzy `A`:

```
>>A(:,3)    % wszystkie wiersze z trzeciej kolumny macierzy A
ans =
     3
     6
     9
>>A(:,2:3)    %% wszystkie wiersze z drugiej i trzeciej kolumny macierzy A
ans =
     2     3
     5     6
     8     9
```

```
>>A(:,2:-1:1)    %% wszystkie wiersze z drugiej, a potem pierwszej kolumny
ans =
     2     1
     5     4
     8     7
>>A(:,2:1)    % 2:1 generuje pusty wektor, bo 2>1, czyli brak kolumn. Rezultatem jest pusta macierz
ans =
   Empty matrix: 3-by-0
```

Przykład 2.4.3-2

Wybór wierszy macierzy `A`:

```
>>A(2,:)    % wypisywanie drugiego wiersza macierzy A
ans =
     4     5     6
>>A(2:3,:)    % wypisywanie drugiego i trzeciego wiersza
ans =
     4     5     6
     7     8     9
```

Przykład 2.4.3-3

Usunięcie wybranych elementów macierzy (tutaj drugi wiersz) poprzez wstawianie elementów pustych `[]`:

```
>> A(2,:)=[]    % usuwanie drugiego wiersza
A =
     1     2     3
     7     8     9
```

Przykład 2.4.3-4

Przekształcenie macierzy w wektor kolumnowy

```
>>b=A(:)    % wszystkie elementy aktualnej macierzy A jako wektor pionowy
b =
     1
     7
     2
     8
     3
```

Przykład 2.4.3-5

Użycie słowa kluczowego `end` jako oznaczenie ostatniego wiersza lub ostatniej kolumny macierzy:

```
>> A(end,:)    % wypisanie ostatniego wiersza macierzy A

ans =
     7     8     9
>>sum(A(:,end))    % suma elementów ostatniej kolumny macierzy A
ans =
    12
```

2.5. Operatory arytmetyczne i logiczne

Informacje dotyczące sposobu działania tych operatorów można uzyskać, wykonując polecenia: `doc arith`, `doc slash`, `doc ctranspose`, `doc kron`.

2.5.1. Operatory arytmetyczne dla tablic i macierzy

W MATLAB-ie za pomocą operatorów wykonuje się dwa rodzaje operacji na wektorach i macierzach. Pierwszy rodzaj to arytmetyczne **operacje macierzowe** (ang. *matrix operation*) określone regułami algebry liniowej. Drugi rodzaj to tzw. arytmetyczne **operacje tablicowe** (ang. *array operation*), które są wykonywane na elementach macierzy. W tabeli 2.5.1-1 zestawiono wybrane operatory arytmetyczne pakietu MATLAB z uwzględnieniem wspomnianego powyżej podziału. Więcej informacji można uzyskać za pomocą polecenia `doc` z parametrami `matfun`, `elmat` lub `specfun`.

Tabela 2.5.1-1. *Operatory arytmetyczne (macierzowe i tablicowe)*

Symbol operacji macierzowej	Nazwa operacji	Symbol operacji tablicowej (z kropką)
`+`	dodawanie	
`-`	odejmowanie	
`*`	mnożenie	`.*`
`^`	potęgowanie	`.^`
`/`	dzielenie prawostronne	`./`
`\`	dzielenie lewostronne	`.\`
`'`	transpozycja i sprzężenie macierzy zespolonej, transpozycja macierzy rzeczywistej	
`.'`	transpozycja dowolnej macierzy	
`kron`	funkcja, iloczyn tensorowy Kroneckera	
`dot`	funkcja, iloczyn skalarny wektorów	
`cross`	funkcja, iloczyn wektorowy	
`bsxfun`	funkcja, operacja binarna na elementach macierzy	

Operacje tablicowe są używane przy tworzeniu wykresów, gdyż nie zmieniają wymiarów wyniku operacji.

2.5.2. Mnożenie i dzielenie wektorów i macierzy w MATLAB-ie

W MATLAB-ie są dostępne dwa operatory **mnożenia** (`*`) i (`.*`) oraz cztery operatory **dzielenia** (`/`), (`./`), (`\`), (`.\`). Zestawiono je w tabeli 2.5.1-1. Opis operatorów można otrzymać za pomocą poleceń `doc arith` oraz `doc slash` i `doc mldivide`. Operatorów mnożenia i dzielenia tablicowego użyto w przykładach 2.2.3-1, 2.2.7-1, 2.2.8-1, 2.2.9-1, 2.2.9-2.

Mnożenie A*B można wykonać, jeśli liczba kolumn macierzy A jest równa liczbie wierszy macierzy B lub gdy jedna z macierzy (mnożna lub mnożnik) jest skalarem, czyli macierzą o wymiarze 1×1.

Poprzedzenie operatora mnożenia, dzielenia lub potęgowania kropką (**notacja kropkowa**) powoduje zmianę operacji macierzowej na tablicową, jak w prawej kolumnie tabeli 2.5.1-1. Taki sposób zapisu umożliwia niejawne indeksowanie elementów wektora lub macierzy i wykonanie operacji dla elementów o tych samych indeksach, np. (A(i,j)*B(i,j)). Operacja ta jest wykonywana, jeśli rozmiary A i B są takie same lub gdy jeden z czynników jest skalarem. **Operacje tablicowe** na wektorach i macierzach **nie zmieniają wymiaru** tych elementów i są często wykorzystywane do **przygotowania wykresów**.

2.5.3. Przykłady operacji macierzowych i tablicowych

Zamieszczony poniżej fragment sesji MATLAB-a ilustruje efekty działania poszczególnych operatorów. Pokazano różnice między pojęciami: operacja **na macierzach** i operacja **na tablicach**. Operacje te wykonano na przykładzie obliczania iloczynu dwóch wektorów i dwóch macierzy.

Przykład 2.5.3-1

Mnożenie wektorów (błędne i poprawne) oraz mnożenie tablicowe.

```
>>x=[1 2 3 ]; y=[4 5 6];    % zdefiniowano dwa wektory

>>x*y    % to działanie jest niewykonalne
Error using ==> * Inner matrix dimensions must agree.


>>x*y'    % y', czyli transponowane y jest wektorem pionowym, wynikiem mnożenia jest jedna liczba
ans
    32
>> A=x'*y    % wynikiem mnożenia wektora pionowego i poziomego jest macierz kwadratowa
A =
     4     5     6
     8    10    12
    12    15    18
% wiersze i kolumny tej macierzy są liniowo zależne, a jej wyznacznik det(A)=0
>>x.*y    % mnożenie tablicowe — wynikiem jest wektor, zachowano rozmiary wektorów
ans =
     4    10    18
```

Przykład 2.5.3-2

Dzielenie tablicowe wektora z wartościami funkcji sin(x) przez wektor x o tej samej długości.

```
x=-10:pi/10:16*pi;    % generuje wektor x = –10, –10+pi/10, –10+2*pi/10,...
y = sin(x)./x;    % oblicza wartość y=sin(x)/x dla każdego elementu wektora x
```

Rezultatem są dwa wektory x, y o tych samych wymiarach 1×192, użyte do wykonania wykresu w przykładzie 2.2.9-1. Ewentualne dzielenie przez zero da wynik *NaN* (ang. *not a number*) i może być widoczne jako nieciągłość (dziura) na wykresie.

Przykład 2.5.3-3

Mnożenie macierzowe. Macierze `A`, `B` wprowadzono z klawiatury do przestrzeni roboczej, po czym wykonano mnożenie macierzowe `C=A*B` i tablicowe `C=A.*B`.

```
>> A= [1 2 3; 4 5 6]    % dane
A =
     1     2     3
     4     5     6
>> B= [1 1 1; 2 2 2; 3 3 3]    % dane
B =
     1     1     1
     2     2     2
     3     3     3
>> C=A*B    % mnożenie macierzowe: identyczne długości wiersza A i kolumny B
C =
    14    14    14
    32    32    32
>> D=B*A    % mnożenie niewykonalne, długość wiersza B inna niż kolumny A
Error using  *
Inner matrix dimensions must agree.
```

Mnożenie tablicowe macierzy jest wykonalne, jeśli macierze mają takie same wymiary.

2.5.4. Dzielenie macierzowe i tablicowe

W MATLAB-ie są dostępne cztery operatory dzielenia (tabela 2.5.1-1). Należy zwrócić uwagę na to, że algorytm użyty do wykonania dzielenia macierzy `X = A\B` lub `X = B/A` jest zależny od struktury macierzy `A`, która może być macierzą trójkątną, symetryczną, kwadratową, prostokątną, pełną lub rzadką (podrozdział 8.1.2). Opis działania operatorów dzielenia uzyskuje się za pomocą polecenia `doc slash` oraz `doc mldivide` i `doc mrdivide`.

2.5.4.1. Lewostronne dzielenie macierzy (\)

W poleceniu `x = A\b` użyto operatora `\` (ang. *backslash*). Wywołuje on funkcję `mldivide`, która oblicza rozwiązanie równania $Ax = b$. Operator ten opisano dokładniej w rozdziale 8. Opis podają też polecenia `doc slash` oraz `doc mldivide`.

2.5.4.2. Prawostronne dzielenie macierzy (/)

W poleceniu `x = b/A` użyto operatora `/` (ang. *slash*). Wywołuje on funkcję `mrdivide`, która oblicza rozwiązanie równania $xA = b$. Operator ten opisano dokładniej w rozdziale 8. Opis podają też polecenia `doc slash` oraz `doc mldivide`.

2.5.4.3. Prawostronne dzielenie tablic (./)

Wynikiem operacji `A./B` jest macierz o elementach `A(i,j)/B(i,j)`. Wszystkie macierze mają te same wymiary.

2.5.4.4. Lewostronne dzielenie tablic (.\)

Wynikiem operacji `A.\B` jest macierz o `elementach B(i,j)/A(i,j)`. Wszystkie macierze mają te same wymiary.

Przykład 2.5.4.4-1

Dzielenie tablic jest wykonywane, jeśli `A` i `B` mają ten sam rozmiar lub gdy jedna z tych wartości jest skalarem. Format wyprowadzania wyników zmieniono za pomocą polecenia `format rat`, aby pomijać zera po kropce dziesiętnej:

```
>>format rat    % wyświetla wynik jako iloraz dwu liczb całkowitych
>>x=[1 2 3 ]; y=[4 5 6];    % zdefiniowano dwa wektory
>>x\y    % dzielenie lewostronne (operacja macierzowa)
ans =
      0            0            0
      0            0            0
     4/3          5/3          2

>>x.\y    % dzielenie lewostronne tablic
ans =
      4           5/2           2

>>x/y    % dzielenie prawostronne (operacja macierzowa)
ans =
    32/77

>>x./y    % dzielenie prawostronne tablic
ans =
     1/4          2/5          1/2
```

2.5.5. Operatory potęgowania macierzy i tablic

Operacja potęgowania macierzy jest wykonywana następująco:

- `Z = X^y` (`X` do potęgi `y`) — jeśli `X` jest macierzą kwadratową i `y` jest liczbą całkowitą większą od 1, to wynik uzyskuje się poprzez powtarzanie operacji mnożenia. Dla innych wartości `y` obliczenia są wykonywane przy użyciu wartości i wektorów własnych.
- `Z = x^Y` (`x` do potęgi `Y`) — jeśli `x` jest skalarem i `Y` jest macierzą kwadratową, to obliczenia prowadzi się z wykorzystaniem wartości i wektorów własnych.
- `Z = X^Y` (`X` do potęgi `Y`) — jeśli żaden z elementów `X` i `Y` nie jest skalarem, to pojawia się komunikat o błędzie.
- `Z = X.^Y` (potęgowanie tablic) — operacja ta jest wykonywana poprawnie, gdy `X` i `Y` są macierzami kwadratowymi i mają ten sam rozmiar lub jedna z nich jest wartością skalarną.

Przykład 2.5.5-1

Potęgowanie tablic:

```
>>x.^y
ans =
      1           32          729
```

2.5.6. Sprzężenie i transponowanie macierzy i wektorów

Odmienne działanie operatorów sprzężenia i transponowania macierzy ujawnia się jedynie dla macierzy lub wektorów zawierających wartości zespolone:

- Sprzężenie macierzy, operator: **apostrof** (') — zamiana wierszy macierzy z jej kolumnami oraz **zmiana znaków** dla części **urojonej** elementów zespolonych,
- Transpozycja macierzy, operator: kropka i apostrof (. ') — zamiana wierszy macierzy z jej kolumnami, operator: kropka i apostrof (. ').

Przykład 2.5.6-1

Utworzono macierz zespoloną `Z`, a następnie wykonano sprzężenie i transpozycję tej macierzy:

```
Z= [1+5i, 2+6i; 3-i,  4]
A=Z'
B=Z.'
```

Otrzymane wyniki: macierz `Z`, macierz sprzężona `A`, macierz transponowana `B`:

```
Z =
   1.0000 + 5.0000i   2.0000 + 6.0000i
   3.0000 - 1.0000i   4.0000 + 0.0000i
A =
   1.0000 - 5.0000i   3.0000 + 1.0000i
   2.0000 - 6.0000i   4.0000 + 0.0000i
B =
   1.0000 + 5.0000i   3.0000 - 1.0000i
   2.0000 + 6.0000i   4.0000 + 0.0000i
```

2.5.7. Operatory relacji i operatory logiczne

Stosowane w MATLAB-ie operatory relacji i operatory logiczne zestawiono w tabeli 2.5.7-1. Dokładniejszy ich opis można uzyskać za pomocą polecenia `doc relop`. MATLAB zawiera także zestaw funkcji umożliwiających wykonanie bitowych operacji logicznych (`doc ops`).

Tabela 2.5.7-1. *Operatory relacji i operatory logiczne*

Symbol	Operator relacji	Symbol	Operator logiczny
<	mniejszy od	&	koniunkcja
<=	mniejszy lub równy	|	alternatywa
>	większy od		negacja
>=	większy lub równy	xor	alternatywa wykluczająca
==	równy	~~=	nierówny (różny od)

2.5.8. Relacje i wyrażenia logiczne — przykłady

Relacje i wyrażenia logiczne zapisuje się, korzystając z operatorów podanych w tabeli 2.5.7-1. Wynikiem relacji i wyrażenia logicznego może być wartość skalarna lub wektor bądź macierz o wartościach *zero* lub *jeden*, czyli `FALSE` lub `TRUE`.

Argumentami relacji mogą być macierze. Wymiary porównywanych macierzy muszą być takie same. Przy porównywaniu wartości skalarnej i macierzy wartość skalarna jest porównywana z każdym elementem macierzy.

Łańcuchy można porównywać poprzez użycie operatorów relacji oraz za pomocą funkcji logicznych `strcmp`, `strcmpi`, `strncmpi`, `regexp`. Poniżej zamieszczono przykład prezentujący rezultaty obliczeń kilku relacji i wyrażeń logicznych dla zadanych macierzy `A` i `B`.

Przykład 2.5.8-1

```
>>A=[-1 2  0; 4 0 6 ],B=[1 2 3; 1 2 3]    % zadane macierze
A =
    -1     2     0
     4     0     6
B =
     1     2     3
     1     2     3

>>A>=B
ans =
     0     1     0
     1     0     1

>>A&B    % iloczyn logiczny macierzy A i B
ans =
     1     1     0
     1     0     1

>>~A    % negacja macierzy A
ans =
     0     0     1
     0     1     0

>>x=5; x>=A
ans =
     1     1     1
     1     1     0
```

Operatory relacji mogą być stosowane do porównywania łańcuchów lub tablic znakowych o tych samych wymiarach. Operatory relacji (>, >=, <, <=, ==, ~= lub !=) dokonują porównania wartości odpowiednich znaków w kodzie ASCII. Stąd możliwe jest porównywanie łańcucha lub tablicy znakowej z wartością skalarną.

Przykład 2.5.8-2

```
>>s3=['Ala';'Ola']
  s3 =
     Ala
     Ola

>>s3(1,:)==s3(2,:)    % porównanie wierszy tablicy znakowej s3
  ans =
       0     1     1
```

Zachęca się Czytelnika do sprawdzenia, że dla macierzy `A=[-1,2,0;4,0,4];` polecenie `A(A==4)=-55` spowoduje zamianę w macierzy `A` wszystkich elementów o wartości `4` na wartość `-55`.

2.5.9. Funkcje logiczne

Funkcje logiczne stanowią istotne uzupełnienie operatorów z tabeli 2.5.7-1 i są stosowane na przykład przy konstruowaniu relacji i wyrażeń logicznych dla instrukcji warunkowej `if` oraz iteracyjnej `while`.

Argumentami funkcji logicznych mogą być: wektory, łańcuchy, macierze, tablice blokowe i znakowe, struktury, obiekty, zmienne oraz wyrażenia. Wynikiem funkcji logicznej może być wartość **jeden** lub **zero** (`TRUE` lub `FALSE`) albo macierz o elementach **jeden** i **zero**. Funkcje logiczne przyjmują wartość **jeden** (`TRUE`), jeżeli ich argument spełnia żądany warunek.

Przy niespełnionym warunku funkcje logiczne reprezentują `FALSE` i mają wartość **zero**. Przykładowo funkcja `isreal` ma wartość **jeden**, gdy jej argument jest macierzą, której wszystkie elementy są rzeczywiste.

Funkcje logiczne (podano nazwę i warunek, przy którym funkcja jest `TRUE`):

- `all` — wszystkie elementy wektora są niezerowe,
- `any` — jakikolwiek element wektora jest niezerowy,
- `isfinite` — element argumentu jest skończony,
- `isinf` — element argumentu jest równy `+Inf` lub `-Inf`,
- `isnan` — element argumentu jest nieokreślony (`NaN`),
- `isempty` — argument jest macierzą pustą,
- `isletter` — element argumentu jest literą wielką lub małą,
- `ischar` — argument jest łańcuchem lub tablicą znakową,
- `strcmp` — porównywane łańcuchy są identyczne,
- `iscell` — argument jest tablicą blokową,
- `isstruct` — argument jest strukturą,
- `isobject` — argument jest obiektem,
- `isreal` — wszystkie elementy argumentu są rzeczywiste,
- `issparse` — argument jest macierzą rzadką,
- `isglobal` — argument jest zmienną globalną,
- `ishold` — jest włączone,
- `exist` — ustalona liczba, gdy zmienna lub funkcja istnieje,
- `find` — podaje indeksy niezerowych elementów (np. w macierzy rzadkiej).

Funkcje `all` i `any` przyjmują wartość jeden lub zero, jeżeli ich argumentami są wektory. Natomiast jeśli argumentami tych funkcji są macierze, to w wyniku uzyskuje się wektor o elementach jeden i zero.

Funkcje `exist` i `find` mogą przyjmować wartości inne niż jeden i zero. Mimo to stosuje się je przy tworzeniu relacji i wyrażeń logicznych.

Niektóre funkcje logiczne działają na elementach argumentu (np. `isfinite`, `isnan`, `isletter`). Rezultatem działania tych funkcji jest wektor lub macierz o elementach zero i jeden. Funkcja `strcmp` ma dwa argumenty, które mogą być łańcuchami lub tablicami znakowymi. Jeśli porównywane łańcuchy lub wiersze tablic znakowych są identyczne,

to jako wynik działania funkcji uzyskuje się odpowiednio wartość 1 (TRUE) lub wektor o elementach 1. Porównywane łańcuchy nie muszą być tej samej długości. Poniżej pokazano, jak działają funkcje isletter i strcmp.

Przykład 2.5.9-1

Testowanie funkcji isletter i strcmp:

```
>>s1='Ala?', s2='Ola!!'    % zdefiniowano łańcuchy s1 i s2
  s1 =
      Ala?
  s2 =
      Ola!!
>>isletter(s1)
  ans =
       1   1   1    0
>>strcmp(s1,s2)    % łańcuchy s1 i s2 nie są identyczne
  ans =
        0
>>strcmp(s1(2:3),s2(2:3))    % elementy (2:3) łańcuchów są identyczne
  ans =
        1
```

2.5.10. Priorytety operatorów arytmetycznych

Operatory arytmetyczne mają zróżnicowane priorytety działania. Poniżej zestawiono operatory arytmetyczne, od priorytetu najwyższego (1) do najniższego (4):

- potęgowanie macierzowe (^) i tablicowe (.^), transpozycja (.'), sprzężenie macierzy ('),
- jednoargumentowa zmiana znaku (+),
- mnożenie macierzowe (*) i tablicowe (.*), dzielenie macierzowe prawo- (/) i lewostronne (\), dzielenie tablicowe prawo- (./) i lewostronne (.\),
- dodawanie (+) i odejmowanie (-).

2.6. Znaki i nazwy specjalne

MATLAB zawiera pewną liczbę znaków oraz wartości i zmiennych specjalnych, które są reprezentowane przez nazwy (identyfikatory) specjalne. Poniżej podano wybrane znaki specjalne MATLAB-a. Można je uzyskać za pomocą polecenia doc ops. Użycie nazw stałych i zmiennych specjalnych do określania własnych zmiennych może prowadzić do błędnego wykonania programu.

Znaki specjalne:

- = — przypisanie wartości (zwraca się uwagę, że == jest operatorem logicznym),
- [] — tworzenie tablic, argumenty wyjściowe funkcji, łączenie macierzy,
- {} — indeksy struktur i tablic komórkowych,
- () — argumenty wejściowe funkcji, indeksy tablic, nawiasy do określenia kolejności działań,
- . — kropka dziesiętna, separator pola dla klasy, obiektu lub struktury,

- ... — kontynuacja polecenia w następnej linii,
- , — separator indeksów, argumentów funkcji, poleceń,
- ; — koniec wiersza macierzy; wstrzymanie wypisania wyniku,
- % — początek komentarza w danej linii,
- : — generowanie wektorów, indeksowanie macierzy,
- ' — łańcuch; operator transpozycji lub sprzężenia macierzy,
- ! — wykonanie komendy systemu operacyjnego.

Poniżej podano wybrane nazwy specjalne stosowane w MATLAB-ie jako zmienne i stałe specjalne. Pełniejsze informacje uzyskuje się za pomocą poleceń `doc elmat` oraz `doc timefun`. Poniżej znajduje się przykład użycia zmiennych specjalnych, które dostarczają informacji o aktualnym czasie i dacie.

Zmienne i stałe specjalne — podano nazwę oraz opis zmiennej lub stałej:

- `ans` — zmienna robocza; przypisuje się jej automatycznie wartość ostatnio wyliczonego wyrażenia, jeśli nie użyto podstawienia,
- `computer` — oznaczenie komputera, na którym pracuje MATLAB,
- `eps` — precyzja zmiennoprzecinkowa, zależy od wartości liczby i jej typu (pojedyncza lub podwójna precyzja),
- `i,j` — jednostka urojona,
- `Inf` — nieskończoność,
- `NaN` — wartość nieokreślona (ang. *Not-a-Number*),
- `nargin` — liczba argumentów wejściowych funkcji,
- `nargout` — liczba argumentów wyjściowych funkcji,
- `pi` — liczba $\pi = 3.1415926535897$ w reprezentacji maszynowej,
- `realmax realmax('double')` — największa dostępna liczba rzeczywista,
- `realmin realmin('double')` — najmniejsza dostępna liczba.

Zmienne specjalne i funkcje czasu:

- `clock` — aktualna data i czas jako elementy wektora,
- `cputime` — liczba sekund od ostatniego startu MATLAB-a,
- `date` — aktualna data w postaci łańcucha,
- `now` — aktualna data i czas w postaci liczby,
- `etime` — podaje wartość wybranego przedziału czasu,
- `tic`, `toc` — funkcje do odmierzania czasu obliczeń.

Zmienna `clock` jest wektorem o sześciu elementach. Reprezentują one kolejno aktualne wartości roku, miesiąca, dnia, godziny, minuty i sekundy.

Zmienna `date` podaje aktualną datę w postaci łańcucha. Zmienna `now` jest liczbą reprezentującą aktualną datę i czas dla chwili, w której ją wywołano. Funkcję `fix` objaśniono w przykładzie 2.2.2.4-1. Przekształcenie zmiennej `now` w łańcuch według formatu wybranego z dziewiętnastu możliwych uzyskuje się za pomocą funkcji `datestr`.

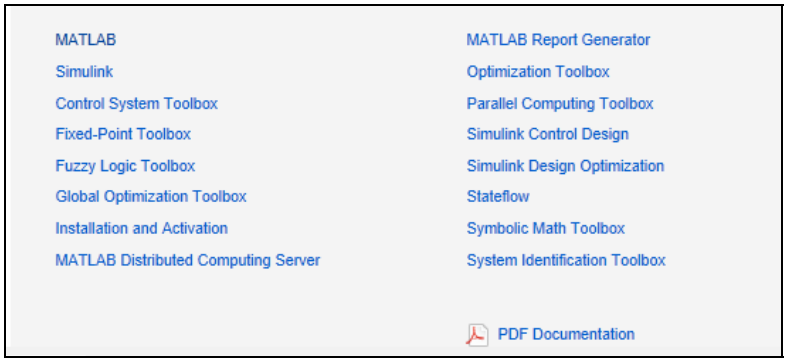
Przykład 2.6-1

```
>>format long, czas=now
  czas =
        7.317623584932638e+005

>>kalendarz=datestr(czas)
  kalendarz =
        30-Jun-2003 08:36:14

>>zegar=fix(clock)
  zegar =
        2003     6     30     8     36     15
```

2.7. System pomocy

Pełna dokumentacja MATLAB-a, Simulinka, bibliotek toolbox i innych składników środowiska MATLAB jest dostępna w postaci plików HTML oraz PDF. Pliki HTML są wykorzystywane przez interaktywny system pomocy, dostępny lokalnie poprzez polecenie `doc`. Są one też dostępne pod adresem: *http://www.mathworks.com/help*. Wiele przydatnych linków, w tym do plików wideo, przykładów i webinariów udostępniono na stronie *http://www.mathworks.com/products/matlab/*. Dokumentację PDF można pobrać ze strony *http://www.mathworks.com/help/pdf_doc/allpdf.html/*. Korzystanie z dokumentacji niekiedy wymaga utworzenia bezpłatnego konta użytkownika lub posiadania licencji na dany produkt.

Lokalny, interaktywny system pomocy uruchamia się z pulpitu MATLAB-a przez polecenie `doc` lub przez kliknięcie myszą ikony *Help*. Podane tam informacje są starannie przygotowane i bogato ilustrowane przykładami oraz rysunkami. Potrzebną informację wyszukuje się przez podanie hasła w okienku *search* lub przeglądając spis treści (rysunek 2.7-1).



Rysunek 2.7-1. *Okno systemu pomocy (Help)*

Tradycyjny, tekstowy system pomocy, oparty na wykorzystaniu polecenia `help`, jest także bardzo przydatny. Jego główną zaletą jest to, że nie zabiera dużo miejsca na ekranie. Jest też najbardziej wiarygodny, gdyż potrzebne informacje pobierane są bezpośrednio

z komentarzy w plikach MATLAB-a. Informacje tekstowe uzyskiwane za pomocą polecenia `help` są zawsze dostępne i odnoszą się do **aktualnie zainstalowanej** na komputerze **wersji oprogramowania**. Mogą one być bardziej aktualne niż opisy pochodzące z innych dokumentów.

Polecenie `>>help` podaje informację tekstową, a `>>doc` *`słowo_kluczowe`* podaje ładnie sformatowane i bardziej szczegółowe informacje tekstowe i graficzne z plików HTML. Słowem kluczowym może być nazwa skryptu, nazwa funkcji lub nazwa folderu przeszukiwanego przez MATLAB-a (patrz opis polecenia `path`). Polecenia `doc help` oraz `help help` objaśniają działanie systemu pomocy oraz podają przykłady, jak można uzyskać informacje na temat klas i metod.

W podobny sposób, poprzez polecenie `help`, mogą być dostępne objaśnienia zawarte w funkcjach i w skryptach tworzonych przez użytkownika. W tym celu należy wpisać do własnych plików odpowiednie komentarze, poczynając od drugiej linii każdego z programów i podprogramów. Polecenie `help` wypisze teksty komentarzy, od linii 2. aż do linii poprzedzającej kod lub linię pustą. Można też przygotować plik *Contents.m* i umieścić go w tym samym folderze co własne pliki. Folder z plikami utworzonymi przez użytkownika będzie przeszukiwany i widoczny dla MATLAB-a dopiero po wpisaniu jego ścieżki dostępu w *PathBrowser* (rysunek 3.6-2).

Polecenie `>> lookfor` *`słowo_kluczowe`* pozwala na odszukanie potrzebnej funkcji, jeśli nie jest znana jej nazwa. Przeszukuje ono pierwszą linię pomocy wszystkich funkcji (linia H1, zazwyczaj druga linia programu, jeśli zawiera tylko komentarz) we wszystkich folderach dostępnych dla MATLAB-a. Na ekranie są wypisywane wszystkie odszukane linie H1 zawierające dane słowo kluczowe.

Polecenie `>>lookfor -all` *`słowo_kluczowe`* działa wolniej, gdyż przeszukuje cały pierwszy blok linii komentarzy, a nie tylko pierwszą linię. Zwiększa to szansę na odszukanie potrzebnej informacji.

Uwaga: w dalszej części książki **znak zachęty** >> będzie zazwyczaj **pomijany**.

Rozdział 3.
Programowanie

MATLAB jest językiem wysokiego poziomu, nie wymaga deklarowania zmiennych ani określania typu danych. Jego polecenia, operatory i funkcje stosuje się do obliczeń numerycznych (w tym na macierzach i na liczbach zespolonych) oraz do wizualizacji wyników w grafice dwu- i trójwymiarowej. Ponad 1300 opisanych w dokumentacji MATLAB-a funkcji realizuje algorytmy numeryczne, operacje na macierzach, wielomianach, metody interpolacji i aproksymacji, transformacje Fouriera, algorytmy całkowania równań różniczkowych, implementacje specjalizowanych algorytmów dla macierzy rzadkich i wiele innych. W MATLAB-ie jest ponad 1300 dostępnych funkcji (ok. 2500 — po uwzględnieniu funkcji w rozszerzeniach MATLAB-a).

MATLAB realizuje funkcje typowe dla języków programowania, w tym pętle `for` i `while`, instrukcje warunkowe `if-then-else` i `try-catch`, oraz wspiera programowanie *zorientowane obiektowo* . MATLAB jest językiem wysokiego poziomu, nie wymaga deklarowania zmiennych ani określania typu danych. Jego polecenia, operatory i funkcje są wykorzystywane do obliczeń numerycznych (w tym *na macierzach* i na *liczbach zespolonych*) oraz do *wizualizacji* i *animacji* wyników w grafice dwu- i trójwymiarowej.

Okno *Command Window* jest przeznaczone do wykonywania poleceń MATLAB-a oraz do wywoływania skryptów, funkcji i aplikacji *Apps*. Jeśli wykonuje się większą liczbę poleceń, należy wpisać je do pliku. Rozróżnia się dwa rodzaje plików: *skryptowe* i *funkcyjne*.

Używane dotychczas określenie M-*pliki* (pliki z rozszerzeniem *.m*) traci na znaczeniu, gdyż pojawiły się nowe narzędzia i nowe rozszerzenia. Na przykład *Live Editor* nadaje edytowanym plikom rozszerzenie *.mlx*.

Przykład 3-1

Skrypt *pitagoras1.m* oblicza długość przeciwprostokątnej c trójkąta prostokątnego w oparciu o twierdzenie Pitagorasa. Obliczenia są wykonywane w przestrzeni roboczej MATLAB-a. Po wykonaniu skryptu wszystkie zmienne pozostają w przestrzeni roboczej.

```
a=3, b=4
c2=a^2 +b^2, c=sqrt(c2)
```

Skrypt jest plikiem tekstowym ASCII. Może zawierać sekwencje poleceń MATLAB-a, wywołania skryptów, funkcji, aplikacji oraz komentarze. Od MATLAB-a 2016b na końcu skryptu można umieścić funkcje lokalne. Skrypt może wywoływać sam siebie.

Przykład 3-2

Funkcja *pitagoras.m* oblicza długość przeciwprostokątnej trójkąta prostokątnego w oparciu o twierdzenie Pitagorasa. Obliczenia są wykonywane w przestrzeni roboczej funkcji. Po wykonaniu zapisanego w funkcji programu wszystkie zmienne, prócz argumentu wyjściowego *c*, są usuwane. Zapobiega to kolizjom nazw zmiennych.

```
function [c]=pitagoras(a,b)
c2=a^2 +b^2
c=sqrt(c2)
```

Pierwsza linia funkcji zawiera słowo kluczowe `function`. Funkcje omówiono w podrozdziale 3.2.

Programy mogą być też tworzone i prezentowane wraz z wynikami obliczeń w interaktywnym środowisku *Live Editor* (podrozdział 3.1.2) lub jako *aplikacje App* (podrozdział 7.2). Programy zorientowane obiektowo omówiono w rozdziale 6.

Kod w języku C (także C++ albo Fortran) może być skompilowany do funkcji MEX (ang *MATLAB EXecutable*) i może być wywoływany bezpośrednio z poziomu MATLAB-a. Podobnie można wywołać skompilowany kod MATLAB-a z C, C++ lub Fortranu.

3.1. Editor, Live Editor, pliki skryptowe

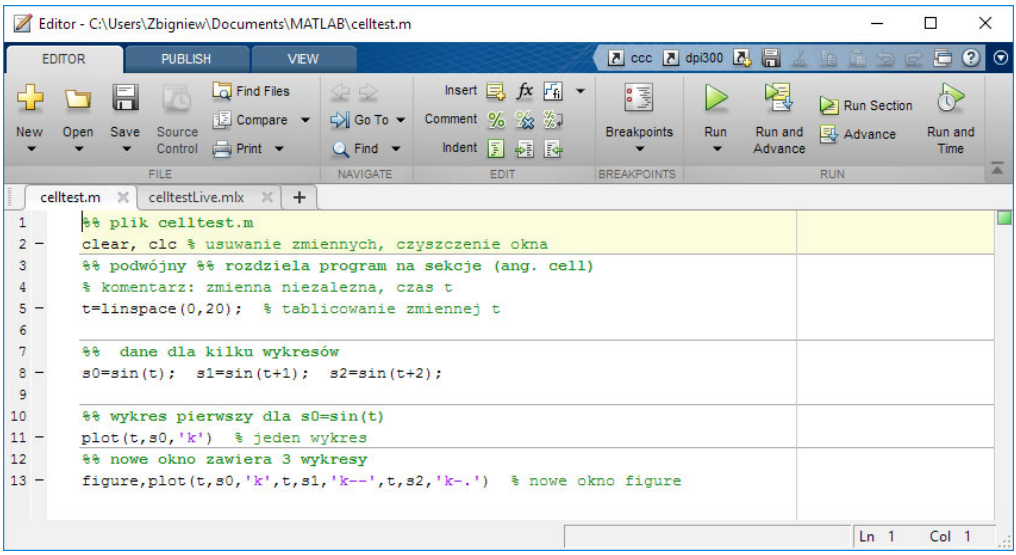
Skrypt najlepiej przygotować w edytorze wbudowanym w MATLAB-a. Jeśli chcemy przeglądać lub utworzyć skrypty lub funkcje poza MATLAB-em, godny polecenia jest dostępny na stronie *http://notepad-plus.sourceforge.net* bezpłatny edytor *Notepad++*. Obsługuje on wiele różnych języków programowania, wyróżniając kolorem słowa zastrzeżone i komentarze, a po kliknięciu myszą dowolnego nawiasu pokazuje otwarcie i domknięcie wybranej pary nawiasów.

Edytory wbudowane w MATLAB-a podświetlają słowa zastrzeżone MATLAB-a na *niebiesko*, łańcuchy domknięte apostrofami na *fioletowo*, łańcuchy niedomknięte na *bordowo*, a komentarze na *zielono*.

3.1.1. Editor — środowisko programistyczne MATLAB-a

Puste okno edytora MATLAB-a można otworzyć poprzez kliknięcie ikony *New* i wybranie opcji *Script* lub *Live Script*.

Na rysunku 3.1.1-1 pokazano okno *Editor* z wpisanym skryptem (plik *celltest.m*). Program tablicuje (oblicza wartości w różnych punktach) trzy funkcje `s0(t)`, `s1(t)`, `s2(t)`. Następnie tworzy wykres funkcji `s0=sin(t)`, po czym otwiera drugie okno graficzne i umieszcza w nim trzy wykresy.



Rysunek 3.1.1-1. *W edytorze można zapisywać skrypty, funkcje i klasy. W chwili zapisu edytor dopisuje do nazwy pliku rozszerzenie .m*

Aby zmodyfikować już istniejący skrypt lub funkcję, należy go (ją) otworzyć przez dwukrotne kliknięcie jego (jej) nazwy w oknie *Current Folder* lub wyszukać go (ją) po kliknięciu ikony *Open*. W oknie edytora można jednocześnie wyświetlić kilka plików obok siebie lub na zakładkę. Sposób wyświetlania określa się w zakładce *VIEW* panelu MATLAB-a.

3.1.2. Live Editor — nowe środowisko programistyczne

Live Editor pozwala na interaktywne tworzenie plików skryptowych zwanych *live scripts*. Są one zapisywane w plikach z rozszerzeniem *.mlx* i mogą zawierać:

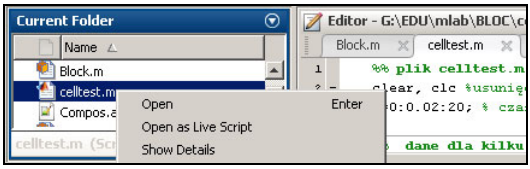
- *skrypty*, czyli sekwencje poleceń MATLAB-a,
- *funkcje lokalne* (od wersji MATLAB 2016b),
- *dokumentację* — grafikę i odpowiednio sformatowane teksty (tytuły, nagłówki rozdziałów, linki i wzory matematyczne zapisane w formacie LaTeX; dołączenie dokumentacji do pliku z kodem ułatwia jej aktualizowanie i wykorzystanie;
- *wyniki obliczeń* w formie cyfrowej i graficznej — są aktualizowane po każdym wykonaniu live scriptu; wykresy, tytuły, osie, legenda i inne adnotacje mogą być edytowane interaktywnie.

Ponadto *Live Editor*, podobnie jak MATLAB Editor:

- sugeruje sposób *korekty błędów* w poleceniach i nazwach zmiennych,
- podaje *aktualną wartość zmiennych* wskazywanych myszą.

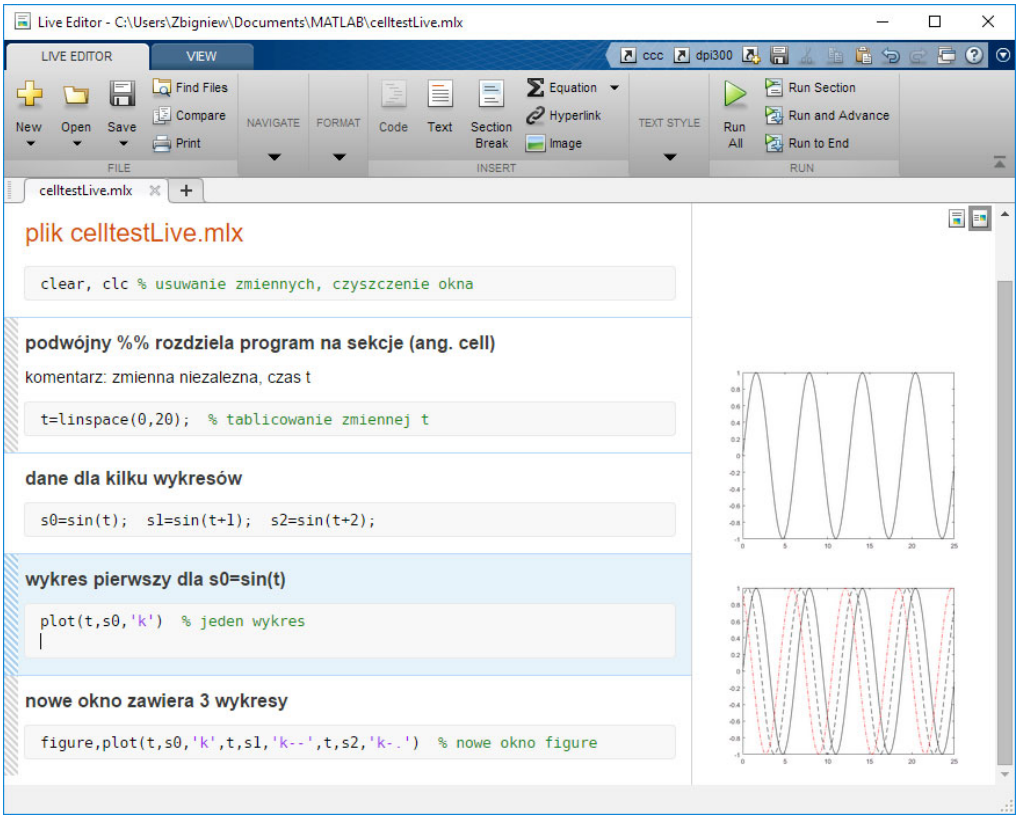
Puste okno *Live Editor* można otworzyć poprzez kliknięcie ikony *New* ▼ i wybranie opcji *Live Script*.

Skrypt może być wczytany i przetworzony do *Live Editor* po jego kliknięciu prawym przyciskiem myszy w oknie *Editor* lub *Current Folder* (rysunek 3.1.2-1).



Rysunek 3.1.2-1. *Otwieranie skryptu jako live script*

Na rysunku 3.1.2-2 pokazano przetworzony plik *celltest.m* w oknie *Live Editor*. Jego nazwę zmieniono na *celltestLive.mlx*, aby uniknąć kolizji nazw pliku i mlx-pliku.



Rysunek 3.1.2-2. *Live script otrzymany z pliku celltest.m z użyciem opcji Open As Live Script*

Plik w oknie *Live Editor* jest wstępnie sformatowany i ma rozszerzenie *.mlx*. Można go uzupełnić o dodatkowe teksty i ilustracje, a następnie wykonać cały program, korzystając z ikony . Wyniki obliczeń (teksty i rysunki) pojawią się po prawej stronie okna, ale można je umieścić pomiędzy liniami programu, klikając jedną z ikon .

Zawartość okna *Live Editor* jest prawie gotową do drukowania dokumentacją edytowanego pliku. W razie potrzeby można powrócić do wcześniejszych etapów i dokonać korekty linii programu, komentarzy lub formatowania.

Podczas zapisywania utworzony będzie plik *.mlx* i zostaną do niego dołączone wyniki obliczeń i wykresy. Taki zintegrowany dokument zawiera w jednym pliku *program i jego dokumentację*.

- Jeśli dokona się zmian w skrypcie, to w tym samym oknie można niezwłocznie uruchomić obliczenia i *zaktualizować wyniki* obliczeń i wykresy (rysunek 3.1.2-2).
- Dodatkowe linie komentarza z podwójnym procentem `%%` dzielą skrypt na sekcje (ang. *cell*) i stają się tytułami w dokumentacji.
- Sekcje mogą być uruchomione indywidualnie, co ułatwia i przyspiesza tworzenie programu.

3.1.3. Uruchomienie skryptu z okna edytora lub Live Editor

Zawartość okna *edytora* lub *Live Editor* zostanie uruchomiona po kliknięciu zielonej ikony w pasku narzędziowym edytora (rysunek 3.1.2-1 i 3.1.2-2). Można też uruchomić fragment programu (ang. *section*) zawartego pomiędzy liniami z podwójnym znakiem komentarza `%%`. Jeśli zawartość okna edytora została zmieniona, to przed jego uruchomieniem będzie zapisana aktualna wersja programu. Poprzednia wersja pliku zostanie zapisana z rozszerzeniem *.asv* — jeśli ustawiono opcję *autosave* w preferencjach.

Zapisując skrypt na dysku, należy unikać nazw, które są słowami zastrzeżonymi MATLAB-a lub nazwami funkcji MATLAB-a. W razie wątpliwości można użyć polecenia `which` *`nazwa`*. Jeśli funkcja o podanej nazwie już istnieje i jest widoczna dla MATLAB-a, to `which` pokaże ścieżkę dojścia do tej funkcji.

Skrypt można wykonać poprzez wpisanie jego nazwy (bez rozszerzenia) w oknie *Command Window*. Jeśli jednak skrypt nie znajduje się w folderze aktualnym lub w folderze przeszukiwanym (ang. *search path*, podrozdział 3.6.2), to MATLAB nie znajdzie go i nie zostanie wykonany.

3.1.4. Polecenia z okna History

Pisząc program, warto wykorzystać widoczne w oknie *Command History* kopie wykonywanych wcześniej poleceń. Można tam wybrać kilka pojedynczych poleceń (z użyciem klawisza *Ctrl* i myszy) lub zaznaczyć grupę kolejnych poleceń (z użyciem klawisza *Shift* i myszy). Kliknięcie prawym przyciskiem myszy zaznaczonego polecenia otwiera menu, z którego wybieramy potrzebną opcję, na przykład *Create M-File*. W rezultacie wszystkie zaznaczone polecenia zostaną przekopiowane do nowego okna w edytorze lub do okna *Command Window*.

3.1.5. Kotwiczenie edytora w panelu programu MATLAB

W prawym górnym rogu okna edytora (a także w innych oknach) jest umieszczona ikona menu lub są umieszczone ikony kotwiczenia, uwalniania i zamykania okna . Kliknięcie pierwszej z grupy trzech ikon powoduje wstawienie i zakotwiczenie okna

edytora do panelu programu MATLAB, obok lub powyżej okna *Command Window*. W razie potrzeby położenie i wielkość każdego z okien można modyfikować przy użyciu myszy.

3.2. Pliki funkcyjne

Przykładem pliku funkcyjnego jest funkcja `pitagoras` zapisana w pliku *pitagoras.m*. Utworzono ją, modyfikując plik skryptowy *pitagoras1.m* z przykładu 3-2.

Przykład 3.2

Plik funkcyjny:

```
function c=pitagoras(a,b)
% Plik funkcyjny pitagoras.m
% a,b — parametry wejściowe, długości boków trójkąta prostokątnego
% program oblicza długość przeciwprostokątnej c
c2 = a.^2 +b.^2;      % zmienna lokalna c2
c = sqrt(c2);
```

Funkcja `pitagoras` jest dostępna w folderze, w którym ją zapisano. Jeśli jednak folder jest przeszukiwany przez MATLAB-a (podrozdział 3.6.2), to funkcja może być wywołana z dowolnego folderu. Można też utworzyć *skrót* lub użyć *uchwytu funkcji*.

Pliki funkcyjne można kompilować z użyciem *MATLAB Compiler*. Skompilowane funkcje działają szybciej. Mają one rozszerzenie *.mex* (ang. *MATLAB executable*) i są nazywane MEX-plikami. W Windows mogą mieć rozszerzenie *.dll*.

3.2.1. Przygotowanie nowej funkcji w edytorze

Aby utworzyć nową funkcję w edytorze, należy w oknie *MATLAB* lub w oknie *Editor* wybrać menu *New ▼/Function*. Otworzy się wtedy nowe okno edytora zawierające szablon funkcji, jak poniżej:

```
function [ output_args ] = Untitled( input_args )
% UNTITLED Summary of this function goes here
% Detailed explanation goes here

end     % ta linia jest konieczna tylko w funkcji zagnieżdżonej
```

Jak widać, pierwsza linia pliku funkcyjnego zawiera słowo kluczowe `function`. W miejsce słowa *`Untitled`* należy wpisać wybraną dla funkcji nazwę. Tej samej nazwy należy użyć do nazwania pliku. Listę parametrów wyjściowych podaje się w nawiasie kwadratowym zamiast `output_args`. Jeśli parametry wyjściowe nie są potrzebne, to należy **pominąć** nawiasy kwadratowe i znak równości.

Zamiast (`input_args`) należy podać listę parametrów wejściowych funkcji, np. (`x`, `y`). Parametry oddziela się przecinkami. Parametrem funkcji może być także nazwa funkcji (tej samej lub innej) w postaci łańcucha, np. `'pitagoras'`, lub uchwyt funkcji, np. `@pitagoras`. Jeśli parametry wejściowe nie są potrzebne, to należy pozostawić puste nawiasy.

Funkcję można też edytować i testować w *Live Editor*, jeśli będzie poprzedzona co najmniej jedną linią skryptu, np. wywołaniem tej funkcji.

3.2.2. Wywołanie pliku funkcyjnego

Funkcje można wywoływać, podając ich nazwę (bez rozszerzenia *.m*). Jeśli funkcja ma uchwyt, można ją także wywołać, podając nazwę jej uchwytu. Po nazwie pliku należy *w nawiasie* okrągłym podać *listę jego aktualnych parametrów*.

W wywołaniu funkcji można zastąpić znakami tyldy (~) nazwy aktualnie *niepotrzebnych zmiennych* wyjściowych oraz niepotrzebnych parametrów wejściowych. Takie postępowanie pozwala przyspieszyć działanie programu i zaoszczędza miejsce w przestrzeni roboczej MATLAB-a. Na przykład:

```
function [out1, out2, out3]=oblicz(par1,par2,par3)
```

Jeśli potrzebny jest tylko wynik `out2`, funkcja `oblicz` może być wywołana jako:

```
[~,out2]=oblicz(par1,par2,par3)
```

3.2.2.1. Wywołanie pliku funkcyjnego z edytora

Aby poprawnie wywołać funkcję z przykładu 3.2-1 przez kliknięcie myszą, należy zapisać parametry używane do jej testowania. W tym celu należy kliknąć czarny trójkąt  (pod ikoną  i napisem *Run*) i wybrać z menu opcję *type code to run*. Następnie w niewielkim okienku należy wpisać wywołanie funkcji: `q=pitagoras(3,4)`. Po zatwierdzeniu, ikona  zostanie zastąpiona przez ikonę z kulką  . Po kliknięciu będzie ona wywoływać `q=pitagoras(3,4)`.

Plik funkcyjny może zawierać więcej niż jedną funkcję zdefiniowaną przez użytkownika. Funkcja zapisana w pliku jako pierwsza jest funkcją główną (ang. *primary function*), jej nazwa powinna być taka sama jak nazwa pliku. Inne funkcje zapisane w tym samym pliku (tak jak w przykładzie 3.2-1) to tak zwane *funkcje lokalne*. Opisano je w podrozdziale 3.2.4. Można je wywoływać wyłącznie z wnętrza pliku, w którym są umieszczone.

3.2.2.2. Uchwyt funkcji

Uchwyt funkcji (ang. *function handle*) jest obiektem przechowującym nazwę funkcji, pełną ścieżkę dostępu do niej i inne dane. Uchwyt umożliwia wywołanie funkcji w sytuacji, gdy wywołanie przez użycie nazwy tej funkcji nie jest skuteczne. Uchwyt dla funkcji `pitagoras` z przykładu 3.2-1 można utworzyć następująco:

```
>> h=@pitagoras
h =
  function_handle with value:
    @pitagoras
```

Uchwyt został przypisany do zmiennej `h` i może być użyty zamiast funkcji `pitagoras`, na przykład:

```
>> c=h(3,4)
c =
     5
```

daje poprawny wynik.

3.2.3. Zmienne lokalne, globalne i zmienna persistent

Funkcje działają na zmiennych lokalnych i globalnych. Komunikują się z przestrzenią roboczą poprzez parametry formalne i zmienne globalne. Jednak *nie zaleca się* używania zmiennych *globalnych* — z uwagi na ryzyko powstania trudnych do wykrycia błędów.

3.2.3.1. Zmienne lokalne

Wszystkie **zmienne** używane **wewnątrz** funkcji oraz **nazwy formalne** parametrów wejściowych i wyjściowych użytych podczas definiowania tej funkcji mają charakter **lokalny**. Istnieją one od chwili wywołania funkcji do chwili wyjścia z niej. Oznacza to, że nie posiadają one żadnych powiązań ze zmiennymi z przestrzeni roboczej, nawet jeżeli ich nazwy są identyczne. Zasada ta **nie dotyczy** następujących przypadków:

- funkcja jest zagnieżdżona (ang. *nested*) i ma wspólne zmienne z funkcją główną,
- gdy zmienne są definiowane za pomocą nazw zmiennych specjalnych (należy tego unikać) lub są zadeklarowane jako `persistent` (wtedy nie są usuwane),
- gdy zmienne są określone jako globalne, czyli wpisane jako argumenty polecenia `global` wewnątrz tych funkcji, w których są używane (należy tego unikać).

3.2.3.2. Zmienne globalne

Wartości zmiennych globalnych są widziane w przestrzeni roboczej i wewnątrz funkcji, jeżeli zarówno w przestrzeni roboczej, jak i wewnątrz funkcji wpisano je jako argumenty deklaracji `global`.

Używanie zmiennych globalnych może być źródłem trudnych do wykrycia błędów wykonania. Ponadto wszelkie modyfikacje programu lub stosowanie tych samych nazw zmiennych globalnych w innych programach mogą być źródłem dodatkowych błędów. Dlatego zaleca się zrezygnowanie z używania zmiennych *globalnych*, a co najmniej znaczne ograniczenie ich liczby. Alternatywnym sposobem udostępniania wartości zmiennych są funkcje zagnieżdżone (patrz podrozdział 3.2.5).

3.2.3.3. Zmienna persistent

W MATLAB-ie jest możliwe definiowanie zmiennych lokalnych, które zachowują swoje wartości pomiędzy kolejnymi wywołaniami danej funkcji. Takie zmienne można zdefiniować tylko wewnątrz funkcji poprzez ich wpisanie po deklaracji `persistent`.

3.2.4. Funkcje lokalne

Funkcje lokalne mogą być wpisywane w dowolnej kolejności, jedna po drugiej:

- w pliku funkcyjnym po funkcji głównej;
- (od MATLAB-a 2016b) również w pliku *live script* oraz w *pliku skryptowym* po ostatniej wykonywalnej linii skryptu.

Funkcje lokalne są widoczne jedynie wewnątrz pliku, do którego zostały wpisane. Nie mogą być wywołane z okna *Command* ani z innych programów.

Użycie funkcji lokalnych zmniejsza liczbę plików, poprawia czytelność programu i ułatwia jego modyfikowanie.

Przykład 3.2.4-1

Funkcja *główna* i jedna funkcja *lokalna*. Pojedyncze i podwójne procenty oznaczają komentarz, a dodatkowo są użyte do formatowania dokumentacji tego skryptu w *Live Editor* (podrozdział 3.1.2) oraz w narzędziu do automatycznego przygotowania raportów PUBLISH, dostępnym w edytorze MATLAB-a (podrozdział 3.8.1.2). Raport PUBLISH do tego przykładu umieszczono w podrozdziale 3.8.1.3.

```
%% Funkcja główna i lokalna — oblicz nowy kapitał po oprocentowaniu lokaty odnawialnej
function kapital = lokataOdnaw(oprocRok,okres,kapital,ileMiesiecy_wBanku)
% przykład   % kapital = lokataOdnaw(4,3,100,6) % 4%, 3 mies., 100 PLN, 6 mies.
zainwestowano = kapital;
ileOkresowLokaty = fix(ileMiesiecy_wBanku/okres);    % liczba całkowita
%% Obliczanie zysku za każdy okres lokaty
while(ileOkresowLokaty)    % wykonanie tylko gdy ileOkresowLokaty ≠ 0
    kapital = lokata(oprocRok,okres,kapital);
    ileOkresowLokaty = ileOkresowLokaty-1;
end   % while end
zysk = kapital-zainwestowano;
end   % koniec funkcji głównej lokataOdnaw

%% Funkcja lokata oblicza nowy kapitał po jednym okresie lokaty
function [kapital] = lokata(oprocRok,okresLokaty,kapital)
% [nowyKapital,zysk] = lokata( 3.5,3,100) % przykład wywołania
%% Obliczenia
procentyZaOkresLokaty = oprocRok/12*okresLokaty;
zysk = procentyZaOkresLokaty/100*kapital;
kapital = kapital+zysk;   % kapitał powiększony o zysk
end   % koniec funkcji lokalnej lokata
```

Zmienne używane w *funkcji głównej* i w *funkcjach lokalnych* są traktowane jako zmienne lokalne każdej z tych funkcji i *nie mają ze sobą żadnego związku*, nawet gdy mają te same nazwy. Słowa `end` na końcu funkcji mogą zostać pominięte (wszystkie jednocześnie i tylko w pliku funkcyjnym), gdyż koniec funkcji jest jednoznacznie określony pierwszą linią kolejnej funkcji lokalnej lub końcem pliku.

3.2.5. Funkcje zagnieżdżone

Funkcja jest zagnieżdżona (ang. *nested*), jeśli jest umieszczona wewnątrz innej funkcji (będzie ona nazwana funkcją główną), czyli przed słowem `end` oznaczającym koniec funkcji głównej. Słowo `end` wskazuje zakończenie funkcji i *jest wymagane* zarówno w funkcji głównej, jak i w funkcji w niej zagnieżdżonej. Funkcja lokalna może być zarazem funkcją główną dla funkcji w niej zagnieżdżonej.

Zaletą funkcji zagnieżdżonej jest to, że funkcja *główna* i funkcje w niej *zagnieżdżone* mają *wspólną przestrzeń zmiennych* roboczych. Oznacza to, że jeśli w funkcji głównej lub zagnieżdżonej przypisano zmiennej nową wartość, to będzie ona dostępna (czyli można ją odczytać i zmienić jej wartość) we wszystkich funkcjach w niej zagnieżdżonych oraz w funkcji, w której ta funkcja jest zagnieżdżona.

Funkcja zagnieżdżona jest *widoczna* tylko dla swojej *funkcji głównej* oraz dla innych *funkcji zagnieżdżonych* na tym samym poziomie w tej samej funkcji głównej. Funkcje zagnieżdżone nieznacznie spowalniają obliczenia i nie są zalecane, jeśli szybkość obliczeń jest parametrem krytycznym (podrozdział 3.7).

Przykład 3.2.5-1

Funkcja zagnieżdżona `lokata` nie *ma żadnych parametrów* wejściowych ani wyjściowych, gdyż funkcja główna i funkcje w niej zagnieżdżone mają *wspólną przestrzeń zmiennych* i wszystkie zmienne używane w każdej z tych funkcji są dostępne w pozostałych funkcjach

```
%% Funkcja główna i zagnieżdżona — oblicz nowy kapitał po oprocentowaniu lokaty odnawialnej
function kapital= lokataOdnawZagn(oprocRok,okres,kapital,ileMiesiecy_wBanku)
% kapital = lokataOdnawZagn(4,3,100,6) % przykład 4%, 3 mies., 100 PLN, 6 mies. w banku
ileOkresowLokaty=fix(ileMiesiecy_wBanku/okres);     % odrzuca resztę z
%% Obliczanie zysku za każdy okres lokaty
while(ileOkresowLokaty)     % pętla while
    lokata();
    ileOkresowLokaty=ileOkresowLokaty-1;
end    % while end

%% Funkcja lokata oblicza nowy kapitał po jednym okresie lokaty
    function lokata     % ta funkcja zagnieżdżona nie ma parametrów
        % funkcja zagnieżdżona
        %% Obliczenia
        procentyZaOkresLokaty=oprocRok/12*okres;
        zysk=procentyZaOkresLokaty/100*kapital;
        kapital=kapital+zysk; %
    end    % koniec funkcji zagnieżdżonej lokata
end        % koniec funkcji głównej lokataOdnawZagn
```

Przy porównaniu powyższego przykładu z przykładem 3.2.4-1 widać, że dzięki wspólnej przestrzeni roboczej można było pominąć parametry `(oprocRok,okresLokaty,kapital)` w definicji i wywołaniu funkcji zagnieżdżonej. Ogólnie funkcje zagnieżdżone można wywołać:

- z poziomu bezpośrednio wyższego,
- na tym samym poziomie, w tej samej funkcji głównej,
- z poziomu niższego,
- z dowolnego poziomu i z dowolnego folderu, jeśli użyje się *function handle* (podrozdział 3.2.2.2).

Dodatkowe informacje na temat funkcji zagnieżdżonych można znaleźć poprzez polecenie `doc nested-functions`.

3.2.6. Funkcje prywatne

Korzystanie z funkcji prywatnych jest szczególnie przydatne przy tworzeniu i testowaniu nowych wersji funkcji.

W wybranym folderze (*folder macierzysty*) należy *utworzyć podfolder* o nazwie specjalnej *private* i umieścić w nim nową wersję funkcji do testowania. Funkcje umieszczone w podfolderze *private* są funkcjami prywatnymi. Są one widoczne tylko dla funkcji

znajdujących się w folderze macierzystym. Jeśli w folderze prywatnym jest funkcja wywołana przez program z folderu macierzystego, to będzie ona wykonana, nawet jeśli funkcja o tej samej nazwie jest w folderze macierzystym lub w którymkolwiek z folderów przeszukiwanych.

Funkcja prywatna może być wywołana tylko przez funkcję umieszczoną w folderze macierzystym. Dzięki temu unika się kolizji z innymi funkcjami o tej samej nazwie, które mogą się znajdować w folderach przeszukiwanych przez MATLAB-a (podrozdział 3.6.2).

3.2.7. Funkcja anonimowa

Funkcje anonimowe (ang. *anonymous*) są używane w bibliotekach MATLAB-a i w automatycznie generowanych programach. Są też używane jako parametry funkcji. Funkcja anonimowa nie jest przechowywana w pliku, lecz jest skojarzona z uchwytem, czyli ze zmienną typu *function_handle*. Funkcja anonimowa może zawierać tylko jedno wyrażenie i ma postać:

```
@(arglist) expr
```

gdzie:

- `arglist` to lista argumentów w nawiasie,
- `expr` to wyrażenie lub polecenie.

Funkcja anonimowa może być przypisana do zmiennej o dowolnej nazwie. Zmienna ta, nazwana poniżej *fhandle*, staje się uchwytem (ang. *function handle*):

```
fhandle = @(arglist)
```

Przykład 3.2.7-1

Utworzona będzie funkcja anonimowa obliczająca wyrażenie $z = a^2 + 3b$. Uchwyt tej funkcji `fh` może być przypisany do nowej zmiennej `gg` i wtedy obie zmienne będą typu *function_handle* i będą wskazywać tę samą funkcję anonimową. W następnych liniach:

- zdefiniowano funkcję anonimową obliczającą $\frac{sin(x)}{x}$,
- podano przykład wykonania obliczeń i wykresu *fplot* z użyciem funkcji anonimowej:

```
fh=@(a,b)  a^2+b;          % funkcja anonimowa
gg=fh                      % przypisanie uchwytu fh do zmiennej gg
h=@(x) sin(x)./x;          % inna funkcja anonimowa
z =fh(3,4)                 % funkcja z=3^2+4;
fplot(h,[-2*pi,10*pi])     % wykres h funkcji anonimowej
figure
fplot(@(x) sin(x)./x,[-2*pi,10*pi])    % ten sam wykres
whos                       % wykaz zmiennych w Workspace
```

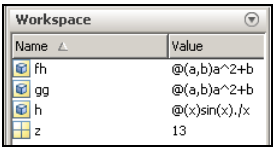
Rezultatem wykonania powyższego przykładu jest poniższy wydruk oraz wykres podobny do pokazanego na rysunku 2.2.9-1:

```
gg =
  function_handle with value:
    @(a,b)a^2+b
z =
    13
```

```
Name      Size            Bytes  Class             Attributes

fh        1x1                32  function_handle
gg        1x1                32  function_handle
h         1x1                32  function_handle
z         1x1                 8  double
```

Zmienne `fh`, `gg`, `h`, `z` są też widoczne w oknie *Workspace* (rysunek 3.2.7-1).

Rysunek 3.2.7-1. *Zmienne w oknie Workspace*



Przykład pokazuje, że uchwyt do funkcji anonimowej zapamiętuje wartości, jakie miały zmienne w momencie tworzenia tej funkcji, a nie same nazwy tych zmiennych.

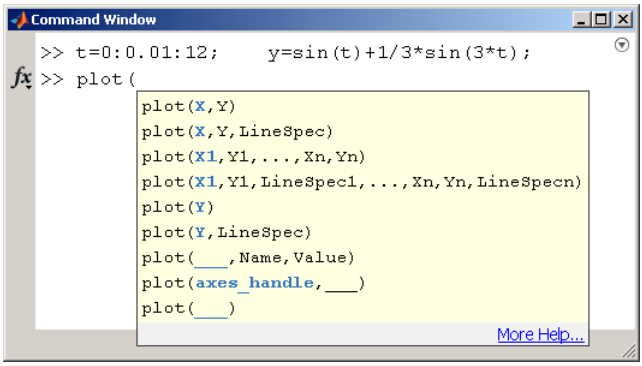
```
% anonym.m
a=3; b=5; c=7;
y=@(t) sin(t) +1/a*sin(a*t) +1/b*sin(b*t) +1/c*sin(c*t)    % funkcja anonimowa
clear a b c     % usunięto zmienne a, b, c; ich wartości są jednak zapamiętane w uchwycie
t=0:0.01:12;    % utworzono wektor t
plot(t,y(t))    % oblicza wartość funkcji y(t) z użyciem a, b, c oraz tworzy wykres
```

Jak widać, jedynym argumentem wejściowym funkcji `y=@(t)` jest zmienna `t`. Wartości `a`, `b`, `c` są zapamiętane w funkcji anonimowej.

3.2.8. Podpowiedzi w oknach Command i Editor

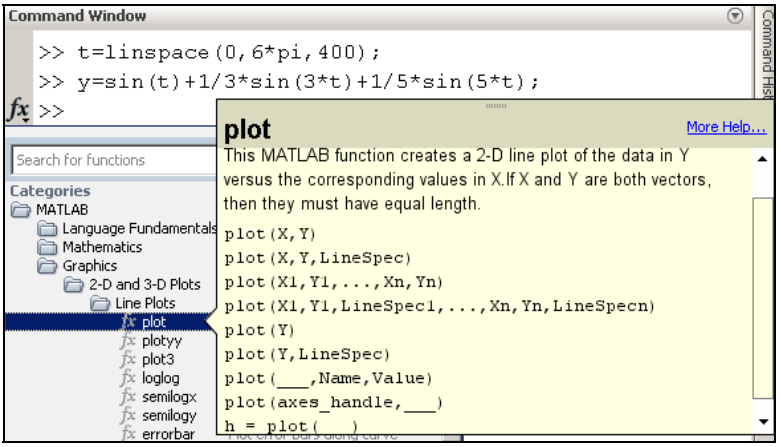
Po wpisaniu w oknie *Command Window* (lub w edytorze MATLAB-a) nazwy funkcji i nawiasu otwierającego pojawi się okno z podpowiedzią w formie wykazu dozwolonych parametrów tej funkcji (rysunek 3.2.8-1).

Rysunek 3.2.8-1. *Po wpisaniu nazwy funkcji i po otwarciu nawiasu pojawi się okno z podpowiedzią w formie wykazu dozwolonych parametrów tej funkcji*



Z podpowiedzi na rysunku 3.2.8-1 widać, że poprawne są wywołania z dwoma parametrami `plot(t,y)`, z jednym parametrem `plot(y)` oraz z wieloma parametrami.

Jeśli nie znamy nazwy potrzebnej funkcji lub jeśli potrzebne są dodatkowe informacje, należy kliknąć ikonę fx >> w oknie *Command Window* (rysunek 3.2.8-2). Spowoduje to otwarcie okna *Function Browser*. Jest to narzędzie do wyszukiwania funkcji



Rysunek 3.2.8-2. *Function Browser „fx" ułatwia wyszukanie potrzebnej funkcji i jej opisu*

bibliotecznych i ich opisów oraz podpowiadające parametry wybranej funkcji. Można je też otworzyć kombinacją klawiszy *Shift+F1*.

3.2.9. Priorytet wywołania funkcji

Może się zdarzyć, że nazwa funkcji i nazwa zmiennej są identyczne. Może być też kilka funkcji o tej samej nazwie, ale różniących się rozszerzeniem bądź zapisanych w innym folderze. MATLAB wykonuje zawsze **pierwszą funkcję** o danej nazwie, znalezioną według podanych poniżej zasad.

Jeśli w trakcie realizacji pewnej sekwencji poleceń MATLAB natrafi na nową nazwę zmiennej lub funkcji, to przed wykonaniem polecenia z udziałem tej nazwy sprawdza ją w następujący sposób:

1. Czy nazwa jest *zmienną*?
2. Czy nazwa jest *funkcją* z pakietu dołączonego przez `import`?
3. Czy nazwa jest *funkcją zagnieżdżoną* (podrozdział 3.2.5)?
4. Czy nazwa jest *funkcją lokalną* (podrozdział 3.2.3)?
5. Czy nazwa jest *funkcją prywatną* (podrozdział 3.2.6)?
6. Czy nazwa jest *obiektem* (rozdział 6.)?
7. Czy nazwa jest *konstruktorem klasy w @-folderze* (podrozdział 6.1)?
8. Czy nazwa jest *przeładowaną metodą* klasy MATLAB-a (podrozdział 6.4)?
9. Czy nazwa jest *funkcją* umieszczoną w *aktualnym* folderze (podrozdział 3.6.2)?
10. Czy nazwa jest *funkcją* umieszczoną w innym *folderze dostępnym* dla MATLAB-a?
11. Jeśli w wybranym folderze jest kilka funkcji o tej samej nazwie, to kolejność ich wyboru jest następująca:
    - funkcja *wbudowana*,
    - *MEX*-plik (podrozdział 3.6.5),
    - *MDL*-plik lub SLX-plik (model Simulinka, podrozdział 10.1.3),
    - *P-plik* (podrozdział 3.6.3),
12. *plik* skryptowy i funkcyjny (podrozdział 3.1).

Wybór właściwej funkcji spośród kilku o tej samej nazwie można też zagwarantować poprzez stosowanie obiektów (rozdział 6.).

3.2.10. Polecenia i funkcje biblioteczne

Poniżej podano wybrane polecenia i funkcje ułatwiające programowanie w MATLAB-ie:

- `echo on` — podaje nazwy poleceń wykonywanych w pliku,
- `end` — zakończenie bloku kodu lub ostatni element tablicy,
- `eval`, `evalin` — wykonanie łańcucha jako polecenia MATLAB-a (podrozdział 3.8.7),
- `feval` — wykonanie funkcji o nazwie podanej tekstowo jako łańcuch,
- `function` — początek nagłówka pliku funkcyjnego,
- `global` — definiowanie zmiennych globalnych,
- `input` — wczytuje dane z klawiatury,
- `inputname` — aktualna nazwa parametru wejściowego funkcji,
- `mfilename` — podaje nazwę ostatnio wykonywanego pliku,
- `mlock` — chroni plik przed usunięciem przez `clear`,
- `munlock` — znosi ochronę plików przed usunięciem,
- `nargin` — liczba parametrów wejściowych funkcji,
- `nargout` — liczba parametrów wyjściowych funkcji,
- `persistent` — definiowanie zmiennych statycznych,
- `varargin` — lista wprowadzająca zmienną liczbę parametrów wejściowych,
- `varargout` — lista wyprowadzająca zmienną liczbę parametrów wyjściowych.

3.3. Instrukcje sterujące przebiegiem programu

Instrukcje sterujące przebiegiem programu to:

- instrukcje warunkowe: `if-elseif-else-end` (podrozdział 3.3.1),
- instrukcja wyboru: `switch-case-otherwise-end` (podrozdział 3.3.2),
- instrukcje pętli: `while-end` oraz `for-end`, (podrozdziały 3.3.1 i 3.3.2),
- instrukcje warunkowe: `try-catch-end` (podrozdział 3.4.3),
- instrukcje przerywające wykonywanie sekwencji instrukcji: (podrozdział 3.4.1),
  - `break` powoduje wyjście z wnętrza aktualnej pętli `while` lub `for`,
  - `continue` powoduje przejście do kolejnego obiegu pętli,
  - `pause` chwilowe zatrzymanie, wznowienie wykonywania programu klawiszem *Enter*,
  - `return` (wpisane do programu) powoduje przerwanie wykonywania tego programu i powrót do miejsca, z którego program wywołano,

- `return` (wpisane po znakach `K>>` w trybie *keyboard*) powoduje powrót do wykonywania programu przerwanego przez `keyboard`,
- `keyboard` wstrzymuje wykonanie programu, zmienia znak zachęty `>>` na `K>>` i umożliwia wykonanie dowolnych poleceń z klawiatury w przestrzeni roboczej aktualnego programu lub funkcji,
- `error` zazwyczaj kończy wykonanie programu, patrz `doc error`,
- *Ctrl+C* z klawiatury, również przerywa wykonanie programu.

Więcej informacji o instrukcjach sterujących podaje `doc lang`.

3.3.1. Instrukcje warunkowe if

Ogólna postać instrukcji warunkowej `if` jest następująca:

```
if wyrażenie_1
   % polecenia1
elseif wyrażenie_2    % elseif można użyć wielokrotnie
   % polecenia2
else
   % polecenia3
end
```

Sposób działania instrukcji warunkowej `if` pokazano poniżej na przykładzie algorytmu rozpoznawania małych liter.

Przykład 3.3.1-1

Algorytm określa, czy znak przypisany do zmiennej `zn` jest małą literą i dodatkowo czy znak jest samogłoską, czy spółgłoską. Użyto instrukcje warunkowe: `if-elseif-else-end`.

```
zn= input('Wpisz z klawiatury znak małej litery --> ','s');
if (isempty(zn)) disp('koniec'),
  return,
end
    disp(['wpisano znak:  "' zn ,'"',])
    samogl =['a' 'e' 'i' 'o' 'u' 'y'];
if any(zn == samogl)  % lub all(zn ~= samogl)
    wyn =['  ' zn  '  jest samogłoską'];
elseif ('a'<=zn) & (zn <='z')
    wyn =['  ' zn  '  jest spółgłoską'];
else
    wyn =['  "'  zn  '"  nie jest małą literą'];
end
disp(wyn)
```

Objaśnienia do przykładu:

- `input` wczytuje znaki z klawiatury i przypisuje je do zmiennej `zn`, zgodnie z przyjętym formatem `'s'`, czyli jako łańcuch (ang. *string*). Wpisywanie należy **zakończyć klawiszem *Enter***.
- `disp` wypisuje zadany tekst. Nawias kwadratowy `[]` łączy kilka tekstów w jeden łańcuch.
- Zmienna `samogl` jest wektorem, do którego wpisano samogłoski: `a`, `e`, `i`, `o`, `y`.

- Operator = = bada, czy w wektorze samogl i zmiennej zn są identyczne elementy. Wynik jest wektorem zawierającym zera i jedynki. Funkcja any bada, czy w wektorze jest element niezerowy. Jeśli to prawda, to do wyn będzie wstawiony tekst: 'zn ' jest samogłoską' i program wykona skok do end.
- Jeśli poprzedni warunek nie był spełniony (nie wykryto samogłoski), to elseif bada, czy wprowadzony z klawiatury znak ma kod odpowiadający małej literze z zakresu od 'a' do 'z'. Jeśli to prawda, to do wyn będzie wstawiony odpowiedni tekst i program wykona skok do end.
- elseif może być użyte wielokrotnie.
- Jeśli żaden z warunków dla if lub elseif nie był spełniony, to zostanie wykonane polecenie po else.

3.3.2. Instrukcja wyboru switch

Instrukcja switch-case-otherwise ma postać:

```
switch  wyrażenie lub zmienna sterująca wyborem
    case  lista stałych wyboru
        % polecenia
    case  lista1 stałych wyboru
        % polecenia
    case  lista2 stałych wyboru
        % polecenia
    otherwise
        % polecenia
end
```

Instrukcja switch-case-otherwise może zastąpić kilka instrukcji if. Po słowie switch umieszcza się wyrażenie lub *zmienną sterującą wyborem* jednego z kilku możliwych wariantów przebiegu obliczeń. Zmienna sterująca wyborem (lub wartość wyrażenia) może być wartością skalarną lub łańcuchem.

Słowo case może być wielokrotnie powtórzone. Po case umieszcza się listę stałych wyboru (w nawiasie klamrowym) lub pojedynczy element bez nawiasów. Jeżeli *zmienną sterującą wyborem* jest równa którejś z wartości wymienionych po słowach kluczowych case, to wykonywane są *polecenia* podane po odpowiednim słowie case (jednorazowo, pozostałe warunki nie są sprawdzane). Jeśli nie stwierdzono zgodności po którymkolwiek case, to wykonywane są polecenia po słowie kluczowym otherwise.

Działanie instrukcji wyboru switch ilustruje poniższy przykład.

Przykład 3.3.2-1

Kalkulator do wykonywania podstawowych operacji arytmetycznych + - * /, przykład dla danych a=4; b=0.5.

```
%% Demo: switch-case-otherwise
a=4; b=0.5;    % dane
operator= input('Wpisz z klawiatury znak: *,+,- lub /, wciśnij Enter','s');
switch operator
    case '+'
       w=a+b    % pominięto średniki, aby lepiej pokazać działanie CASE
    case {'*','.*'}
       w=a*b
```

```
    case {'/','./'}
       w=a/b
    case '-'
       w=a-b
    otherwise
       disp(['wpisano niepoprawny znak w= ',operator]), return
end
disp(['dane a= ', num2str(a),', b= ', num2str(b)])
disp(['a',operator,'b = ',num2str(w)])
```

Wartość zmiennej `w` uzyskiwana po wykonaniu instrukcji `switch` zależy od wartości zmiennej `operator`, czyli od tego, który ze znaków *,.*+,-,/ został wpisany z klawiatury. Jeśli na przykład wpisano * lub .*, to wykonywane jest mnożenie zapisane po `case` z listą wyboru {'*','.*'} w nawiasach klamrowych. Jeśli wpisano inny znak niż *,.*+,-,/, to wykonywane jest polecenie podane po `otherwise`, czyli `w = w-s`.

Instrukcję `input` wykorzystano i opisano w przykładach 3.3.1-1 i 3.3.2-1. Funkcja `num2str(a)` przekodowuje liczbę `a=4` na zmienną tekstową, która będzie jednym z elementów tekstu wypisanego przez `disp` w linii:

```
disp(['dane a= ', num2str(a),', b= ', num2str(b)])
```

Wydruk z przykładu 3.3.2-1 po wybraniu operatora `'+'` może wyglądać następująco:

```
w =
   4.500000000000000
dane a= 4, b= 0.5
a+b = 4.5
```

3.3.3. Instrukcje iteracyjne: while, for i dwukropek (:)

Instrukcje iteracyjne pozwalają na wielokrotne wykonanie pętli z sekwencją poleceń. Obieg pętli może być przerwany i modyfikowany przez instrukcje `break`, `keyboard` `return` i `continue`, omówione w podrozdziale 3.3. W MATLAB-ie używamy instrukcji iteracyjnych:

- `while` — wykonywanie sekwencji poleceń jest powtarzane w pętli tak długo, jak długo wyrażenie po `while` ma wartość (`TRUE`), czyli jest różne od zera.
- `for` — wykonuje ściśle określoną liczbę obiegu pętli. Liczba powtórzeń sekwencji poleceń zależy od wartości początkowej i końcowej *zmiennej sterującej* oraz od *kroku* tej zmiennej.
- `parfor` — jest odmianą `for` dla obliczeń równoległych (podrozdział 11.3.2).
- *Dwukropek* — w MATLAB-ie wykorzystuje się **niejawne indeksowanie** do szybkiego generowania wektorów, macierzy i tablic z użyciem operatora `:` (dwukropek), opisanego w podrozdziałach 2.4 oraz 3.3.3.3.
- Niejawne indeksowanie jest także używane w operacjach tablicowych (`.*`, `./`, `.\`, `.^`, `'`) z kropką, zazwyczaj do przygotowania danych do wykresów. Operacje arytmetyczne są wykonywane kolejno na elementach o tych samych indeksach (podrozdziały 2.5.3 i 3.3.3.3).

3.3.3.1. Instrukcja iteracyjna while

Postać ogólna tej instrukcji jest następująca:

```
while wyrażenie
```

```
polecenia
end
```

Wynik wyrażenia pisanego po `while` *musi być liczbą* (zero lub $\neq 0$). Jeśli wyrażenie po `while` jest tablicą pustą, to polecenia zawarte w tej instrukcji nigdy nie będą wykonane. Jeśli mamy `while 1` to zdefiniowano pętlę o nieskończonej liczbie obiegów. Sposób jej przerwania podano w podrozdziale 3.3.

Sposób działania instrukcji `while` ilustruje algorytm, który wypisuje w odwrotnej kolejności cyfry liczby dodatniej i całkowitej. Algorytm ten zrealizowano, stosując funkcje `rem`, `fix` z podrozdziału 2.2.2.4.

Przykład 3.3.3.1-1

Poniższy program wypisuje cyfry zadanej liczby w odwrotnej kolejności. Użyto pętli `while`.

```
zadana_liczba= 27859;     % dane do przykładu
x=zadana_liczba; i=0;
   while x>0,    i=i+1;   % początek pętli
      y(i) = rem(x,10);   % reszta z dzielenia x/10, czyli ostatnia cyfra do y(i)
      x    = fix(x/10);   % obcina ostatnią cyfrę
   end                    % koniec pętli
disp(['Cyfry liczby  ',int2str(zadana_liczba),' - wypisano'])
disp(['w odwrotnej kolejności  ',int2str(y)])
```

Do wyprowadzenia wyniku liczbowego na ekran użyto funkcji `int2str`, która przekształca liczbę całkowitą w łańcuch (patrz przykład 3.3.3.1-1).

3.3.3.2. Instrukcja iteracyjna for

Postać ogólna instrukcji `for` jest następująca:

```
for zmienna sterująca = wyrażenie     % np. for i=1:100
polecenia
end
```

Zmienna sterująca w instrukcji `for` jest najczęściej obliczana z użyciem dwukropka:

```
zmienna sterująca = wartość_początkowa : krok : wartość_końcowa
```

Jeśli `krok` ma wartość `1`, to zapisuje się `wartość_początkowa : wartość_końcowa`. Wyrażenie może być również tablicą o rozmiarach $m \times n$. Wtedy do zmiennej sterującej wstawia się i-tą kolumnę tej tablicy, a liczba obiegów pętli jest równa liczbie kolumn w tej tablicy.

Sposób działania instrukcji `for` pokazano w przykładzie 3.3.3.2-1.

Przykład 3.3.3.2-1

Utworzenie tablicy trójkątnej dolnej o rozmiarze $n = 6$. Elementy, które leżą na głównej przekątnej i poniżej, zawierają znaki przypisane zmiennej `zn ='X'`. Pozostałe elementy są spacjami.

```
clear              % czyszczenie przestrzeni roboczej
n = 6; zn = 'X';   % dane do przykładu
for i=1:n          % początek pętli1
    for j=1:i      % początek pętli2
        a(i,j) = zn;
    end            % koniec pętli2
```

```
end                    % koniec pętli1
disp(a)    % wyprowadzanie wyniku
```

Powyższy algorytm nie jest optymalny. W MATLAB-ie można go zapisać w jednej linii programu, tworząc macierz kwadratową z jedynek (funkcja `ones`) i korzystając z `tril`, które tworzy podmacierz trójkątną dolną.

```
n = 6; zn = 'X'; a=char(tril(zn*ones(n))); disp(a)
```

Zachęcamy Czytelnika do sprawdzenia obu wariantów zapisu algorytmu.

3.3.3.3. Iteracje realizowane przez dwukropek lub operacje tablicowe

Należy zwrócić uwagę, że w MATLAB-ie instrukcję iteracyjną realizuje także operator *dwukropek* (`:`) oraz operatory tablicowe, np. mnożenie (`.*`). Zapisanie operacji w notacji *kropkowej* lub *dwukropkowej* jest łatwe, a obliczenia są wykonywane znacznie szybciej niż obliczenia *jawnie indeksowane* w pętli `while` lub `for` (porównaj podrozdział 3.7.2).

Przykładowo, operacja tablicowa `C=A.*B` jest znacznie szybsza od:

```
for i=1:size(A(:)); C(i)=A(i)*B(i); end    % indeksem jest zmienna i
```

3.4. Wykrywanie błędów w MATLAB-ie

W MATLAB-ie nawet bardzo złożone operacje macierzowo-wektorowe wymagają mniej linii programu niż w innych językach. Z drugiej strony brak wymogu deklarowania zmiennych powoduje, że program nie jest automatycznie sprawdzany pod kątem zgodności typów zmiennych. Przyspiesza to tworzenie programów, ale zarazem zwiększa ryzyko powstania trudnych do wykrycia błędów.

3.4.1. Lokalizacja błędów w pliku z programem

Plik, którego nazwa została podana w komunikacie o błędzie, można odszukać za pomocą polecenia: `which` *nazwa_pliku*.

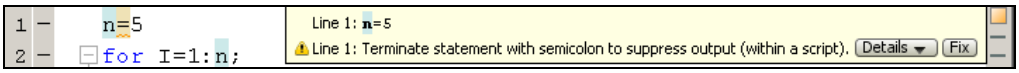
Należy pamiętać, że *wartość zmiennej* (niezadeklarowanej jako globalna) może się zasadniczo *różnić* od wartości tak samo nazwanej *zmiennej lokalnej*, użytej wewnątrz pliku *funkcyjnego*. Różnice mogą dotyczyć również typu zmiennej (na przykład `double`, `int8`, `char`) lub jej wymiarów (na przykład skalar, wektor, tablica dwu- lub trójwymiarowa). Wątpliwości w tym zakresie można wyjaśnić, wstawiając polecenia `whos` do programu lub przez użycie debugera.

W celu zatrzymania wykonania programu przed wybraną instrukcją wstawia się instrukcję `keyboard` lub (niezalecane) instrukcję `error`. W pętli `while` lub `for` można użyć `break`. Zaawansowani programiści używają do tego celu opisanych dalej *pułapek* debugera (podrozdział 3.5.1).

3.4.2. Błędy syntaktyczne i błędy wykonania

Błędy syntaktyczne (składniowe) są wykrywane przez interpreter bądź kompilator. Komunikat o błędzie syntaktycznym ułatwia jego lokalizację i poprawienie. Narzędzie

Analyse_Code (dostępne w oknie MATLAB-a, zakładka *HOME*, grupa *CODE*) sprawdza poprawność formalną programu i dopisuje uwagi do niewłaściwie użytych instrukcji MATLAB-a. Uwagi są wyświetlane w edytorze po najechaniu myszą na kreskę widoczną przy prawej krawędzi okna edytora. Na rysunku 3.4.2-1 uwaga dotyczy braku średnika (ang. *semicolon*) w linii `n=5`. Średnik wstrzymuje niepotrzebne wyświetlanie wyników na ekranie.



Rysunek 3.4.2-1. *Narzędzie Analyse_Code sprawdza poprawność formalną programu i dopisuje uwagi do niewłaściwie użytych instrukcji*

Błędy wykonania można łatwo wykryć, jeśli prowadzą do zatrzymania obliczeń lub do wyników niezgodnych z oczekiwaniami. Lokalizacja błędów wykonania bywa jednak trudna i pracochłonna. Często wymaga sprawdzenia poprawności użytych algorytmów oraz weryfikacji pośrednich wyników obliczeń. Polecenie `echo on` powoduje, że na ekranie są kolejno wyświetlane linie poleceń z aktualnie wykonywanych plików i wszystkie wyniki. Echo wstrzymuje się przez `echo off`.

Plik, którego nazwa została podana w komunikacie o błędzie, można odszukać za pomocą polecenia: `which nazwa_pliku`.

Należy pamiętać, że **wartość zmiennej** (niezadeklarowanej jako globalna) może się **różnić** od wartości tak samo nazwanej **zmiennej lokalnej**, używanej wewnątrz pliku **funkcyjnego**. Różnice mogą dotyczyć również typu zmiennej (na przykład `double`, `int8`, `char`) lub jej wymiarów (na przykład skalar, wektor, tablica *n*-wymiarowa). Wątpliwości w tym zakresie można wyjaśnić, wstawiając polecenia `whos` do programu.

W celu zatrzymania wykonania programu można przed wybraną linią programu wstawić instrukcję `keyboard`. Wstrzymuje to wykonanie programu, zmienia znak zachęty `>>` na `K>>` i umożliwia wykonanie dowolnych poleceń z klawiatury z użyciem zmiennych lokalnych zatrzymanego programu. Lepszym (i wygodniejszym) sposobem zatrzymania programu jest wstawienie opisanej dalej pułapki debugera (podrozdział 3.5).

3.4.3. Programowa obsługa błędów — try-catch-end

Instrukcja `try-catch-end` jest przeznaczona do programowej obsługi błędów. Ma ona postać:

```
try
    % testowany fragment programu
catch
    % polecenia wykonywane w przypadku wystąpienia błędu
end
```

Testowany fragment programu umieszcza się pomiędzy `try` i `catch`. Jeśli MATLAB nie wykryje błędów wykonania w tym fragmencie, to polecenia umieszczone pomiędzy `catch` i `end` nie będą wykonywane.

Jeśli jednak błąd zostanie wykryty (na przykład nieodpowiednie wymiary macierzy przy ich mnożeniu), to zamiast standardowego komunikatu i ewentualnego zatrzymania programu zostaną wykonane polecenia zawarte pomiędzy `catch` i `end`. Mogą one na przykład drukować wartości zmiennych, jak w przykładzie 3.4.3-1.

Przykład 3.4.3-1

Sekwencja `try-catch` wyświetla komunikat, jeśli mnożenie nie zostało wykonane.

```
for n=3:5
    A=magic(4); B=eye(n);
    try
        C=A*B; disp('Wynik mnożenia C=A*B'),disp(C)
    catch
        disp('mnożenia C=A*B nie wykonano, złe wymiary macierzy')
    end
end
```

3.5. Uruchamianie programu — debuger

Debuger jest narzędziem ułatwiającym poprawianie błędów w programie komputerowym.

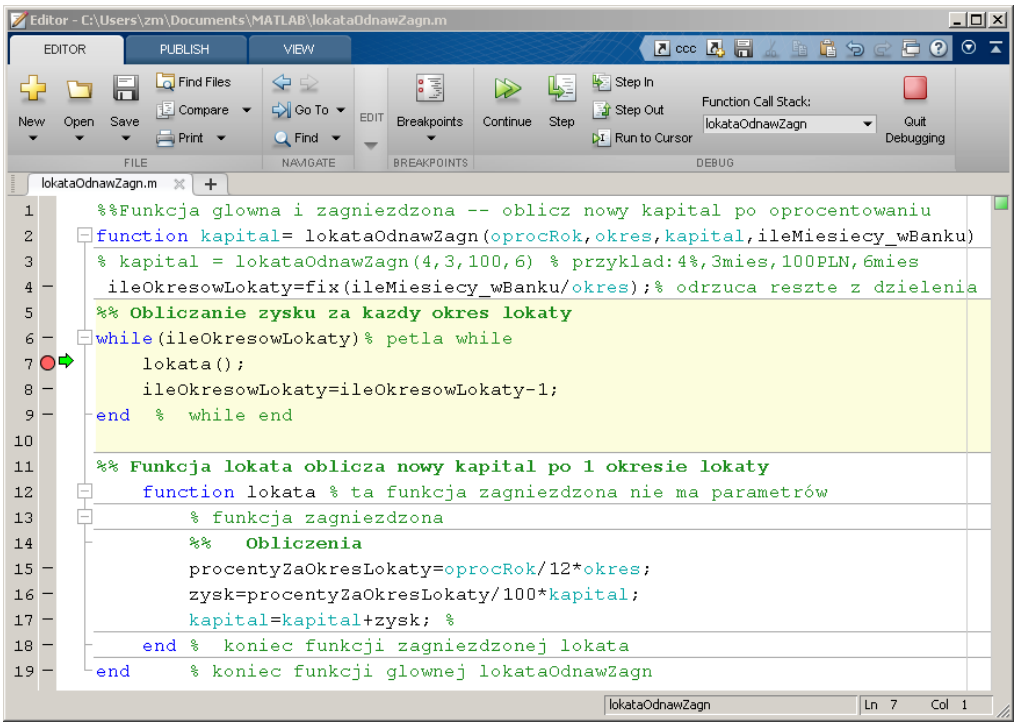
3.5.1. Rozpoczęcie pracy z debugerem, wstawianie pułapki

Na prawo od ikon grupy *FILE*, przeznaczonych do obsługi plików, w tym do ich edytowania i drukowania, znajdują się ikony pokazane na rysunku 3.5.1-1. W grupie *RUN* zielony trójkąt uruchamia wykonanie kodu widocznego oknie edytora. Klikając czarny trójkąt znajdujący się pod tą ikoną, można podać parametry dla uruchamianej funkcji, np. `lokataOdnawZagn(4,3,100,6)`. Po podaniu parametrów na ikonie uruchamiającej funkcję pojawi się dodatkowa ciemna kulka. Ikony *Run Section* i *Run and Advance* uruchamiają wybrane fragmenty (ang. *section*) kodu zawartego pomiędzy liniami z podwójnym znakiem komentarza `%%`, np. linie od 6. do 9. z rysunku 3.5.1-2. Warunkiem koniecznym poprawnego działania debugera jest umieszczenie badanych plików w aktualnym folderze lub w folderach przeszukiwanych przez MATLAB-a (podrozdział 3.6.2).

Rysunek 3.5.1-1. *Wybrane ikony edytora MATLAB-a*

Aby rozpocząć pracę z debugerem, należy otworzyć plik z badanym programem w oknie edytora MATLAB-a i wstawić pułapkę w linii, w której chcemy zatrzymać obliczenia. Pułapkę (ikona w postaci czerwonej kulki) wstawia się, klikając wybraną linię programu przy jej numerze (np. linia 7. na rysunku 3.5.1-2).

Można też kliknąć ikonę *Breakpoints* i wybrać z menu jedną z dodatkowych możliwości:



Rysunek 3.5.1-2. *Debuger — program lokataOdnawZagn podczas pracy krokowej*

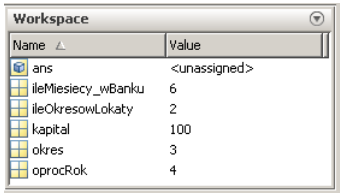
- *Set Condition* — ustaw warunek zadziałania pułapki (np. t>1.5);
- *Stop on Errors* — zatrzymaj w razie błędu;
- *Stop on Warnings* — zatrzymaj w razie komunikatu z ostrzeżeniem;
- *More... Options* — jeszcze więcej możliwości i dodatkowy *Help*.

Pułapki **warunkowe** mają kulkę w kolorze **żółtym**. Po wstawieniu wszystkich zaplanowanych pułapek należy uruchomić badany program.

3.5.2. Debuger — stan wstrzymania obliczeń

Program pokazany na rysunku 3.5.1-2 doszedł do pułapki i został zatrzymany przed wykonaniem linii 7. W oknie *Command Window* zamiast znaku >> pojawia się znak zachęty klawiatury K>>. Oznacza to, że klawiatura umożliwia teraz wprowadzenie dowolnych poleceń do wykonania w lokalnej przestrzeni aktualnego podprogramu z użyciem zmiennych lokalnych tego programu. Wartości zmiennych lokalnych są widoczne w oknie *Workspace* i są dostępne do obliczeń (rysunek 3.5.2-1).

Rysunek 3.5.2-1. *Zmienne lokalne w momencie zatrzymania programu przed linią 7.*



Wartości zmiennych są też pokazywane w oknie edytora. Po ustawieniu kursora myszy na wybranej zmiennej — w niewielkiej ramce obok kursora zostanie wyświetlana wartość tej zmiennej (dotyczy to także wektorów). Dodatkowo po kliknięciu prawym przyciskiem myszy pojawi się bogate menu kontekstowe.

Nazwy i wartości zmiennych lokalnych będą też widoczne w oknie *Workspace* na panelu MATLAB-a.

3.5.3. Debuger — zmienne lokalne funkcji ze stosu

Debuger umożliwia równoczesną *zmianę przestrzeni roboczej* debugera i okna *Command Window*. Po kliknięciu myszą ikony (pod napisem *Function Call Stack*) ukaże się zawartość stosu wywołań funkcji (ang. *stack*). Pozwala to na powrót do miejsca, skąd aktualny program został wywołany, i do zmiennych lokalnych programu w tej lokalizacji. Umożliwia to oglądanie i zmianę wartości zmiennych zarówno *w lokalnej przestrzeni* należącej do kolejnych *funkcji*, które można wybrać *ze stosu*, jak i w przestrzeni roboczej MATLAB-a (ang. *Base Workspace*).

3.5.4. Debuger — śledzenie kolejnych linii programu

Aby wykonać kolejne linie programu, należy kliknąć jedną z ikon z grupy *DEBUG* (rysunek 3.5.4-1). Najczęściej używane są *Step* (jeden krok aktualnego programu, bez wchodzenia do podprogramu), *Step_In* (jeden krok, z ewentualnym wejściem do podprogramu), *Continue* (wykonanie programu do końca lub do kolejnej pułapki). Okno edytora jest na bieżąco aktualizowane, a zielona strzałka (rysunek 3.5.4-1) przy numerach linii) wskazuje aktualną linię wykonywanego programu bądź podprogramu.

Rysunek 3.5.4-1. *Ikony edytora z grupy DEBUG są widoczne podczas pracy debugera*

Do obsługi debugera można wykorzystać wymienione niżej ikony i funkcje (w nawiasie):

- ikona *Set/Clear* — wstaw lub usuń pułapkę (`dbstop`, `dbclear`),
- ikona *Clear All* — usuń wszystkie pułapki (`dbclear all`),
- ikona *Step* — wykonaj jeden krok w aktualnym pliku (`dbstep`),
- ikona *Step In* — wykonaj jeden krok w aktualnie wywoływanym pliku (`dbstep in`),
- ikona *Step Out* — wykonaj pozostałe linie kodu wewnątrz pliku, na przykład po wcześniejszym użyciu *Step In* (`dbstep out`),
- ikona *Continue* — kontynuuj obliczenia aż do napotkania błędu lub pułapki (`dbcont`),
- ikona *Quit Debugging* — koniec pracy z debugerem (`dbquit`).

Aby wyjść z trybu debugowania, należy kliknąć *Quit Debugging*. Wpisanie polecenia `return` odpowiada kliknięciu *Continue*. Listę poleceń debugera można uzyskać, używając polecenia `doc debug`.

3.6. Obsługa plików i folderów

Polecenia `dir`, `ls`, `type`, `delete`, `cd`, `matlabpath`, `pwd`, `what` wykonują podstawowe operacje na plikach i folderach. Poniżej, w tabeli 3.6, podano zestawienie wybranych poleceń w MATLAB-ie i w systemach operacyjnych DOS i UNIX.

Tabela 3.6-1. *Polecenia w MATLAB-ie i w systemach operacyjnych DOS i UNIX*

MATLAB	DOS	Linux i UNIX
`dir, ls`	`dir`	`ls`
`mkdir`	`md, mkdir`	`mkdir`
`type`	`type`	`cat, more`
`delete`	`del`	`rm`
`cd`	`cd`	`cd`
`help, doc, helpbrowser`	`help`	`man`
`which`		`which`
`lookfor`	`find`	`apropos`
`matlabpath`	`path, set`	`set, setenv`

Powyższe polecenia mogą wymagać użycia odpowiednich parametrów.

3.6.1. Wykonywanie poleceń systemu operacyjnego

Z poziomu MATLAB-a użytkownik ma możliwość wykonania dowolnej komendy systemu operacyjnego bądź uruchomienia dowolnego programu zewnętrznego. Komendę taką (lub wywołanie programu) należy poprzedzić znakiem wykrzyknika (`!`) lub podać w apostrofach jako parametr polecenia `dos`, `winopen` albo `unix`. Rezultaty tak wykonywanych komend pojawiają się w oknie poleceń MATLAB-a lub w dodatkowym oknie. Zachęcamy Czytelnika do porównania rezultatów przykładowych poleceń.

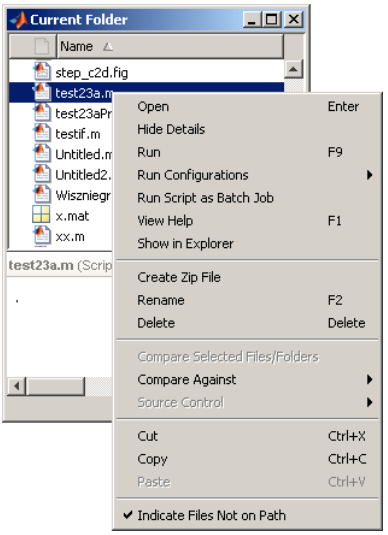
```
>> !type/?
>> [s,w] = dos('type/? &')      % tylko Windows
>> winopen('pitagoras.m')       % otwiera plik w Windows
>> dir
>> !dir                         % tylko Windows
>> !man cat &                   % tylko Unix, Linux
>> [s,w]= unix('man cat &')     % tylko Unix, Linux
```

Znak `&` może być pominięty (niezalecane), jeśli wywoływany program nie powinien otwierać własnego okna. Szczegóły podaje `doc dos`.

3.6.2. Folder aktualny i ścieżki dostępu — path

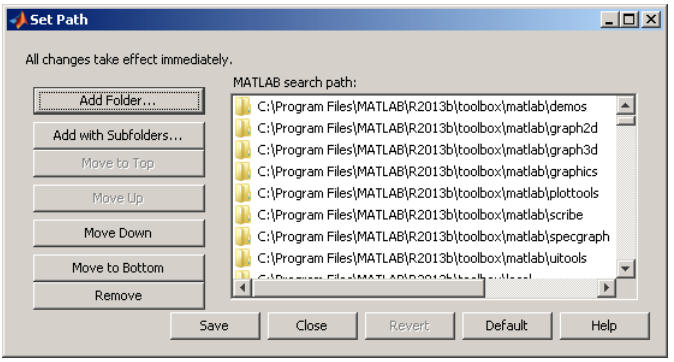
Aktualny folder (ang. *current directory*, *current folder*) i ścieżki dostępu (przeszukiwania, ang. *search path*) są istotne przy wyszukiwaniu plików i funkcji w MATLAB-ie. Plik, który chcemy edytować lub uruchomić, musi znajdować się w *folderze aktualnym* bądź w folderze wskazanym przez jedną ze *ścieżek dostępu*. Dodatkowo można uruchomić pliki, dla których określono *skrót* lub *uchwyt*.

Pełna ścieżka dostępu do aktualnego folderu jest wyświetlana w niewielkim podłużnym okienku panelu *MATLAB*, umieszczonym pod ikonami dostępnych narzędzi. Po prawej stronie tego okienka znajduje się przycisk ▼, pozwalający na wybranie innego, używanego wcześniej folderu. Pliki znajdujące się w aktualnym folderze są wyświetlane w oknie przeglądarki *Current Folder* (rysunek 3.6.2-1). Klikając prawym przyciskiem myszy dowolny plik, otwieramy menu kontekstowe zawierające wiele użytecznych opcji. Nazwę aktualnego folderu podaje też polecenie `pwd` (ang. *print work directory*). Polecenie `cd` pozwala zmienić aktualny folder.



Rysunek 3.6.2-1. *Menu kontekstowe plików w oknie Current Folder*

Jeśli zachodzi konieczność uzyskania stałego dostępu do własnych programów bądź obiektów, należy *zmodyfikować wykaz ścieżek dostępu*. Dokonuje się tego przez wybranie opcji *HOME/Set Path*. Rezultatem jest otwarcie okna *Set Path* (rysunek 3.6.2-2).



Rysunek 3.6.2-2. *Okno Set Path do ustawiania ścieżek przeszukiwania*

Aby dopisać dodatkowy folder do *ścieżek przeszukiwania*, należy kliknąć przycisk *Add Folder*, wyszukać potrzebny folder i zatwierdzić go przyciskami *OK* i *Save*. Ścieżki przeszukiwania są przechowywane w pliku *pathdef.m* w folderze *toolbox\local*.

3.6.3. Pliki w MATLAB-ie

MATLAB korzysta z plików binarnych i tekstowych. Zawartość pliku rozpoznaje się po jego rozszerzeniu. Poniżej zestawiono rozszerzenia najczęściej stosowane w MATLAB-ie.

- *.m* — pliki tekstowe z takim rozszerzeniem to pliki skryptowe, funkcyjne i definiujące klasę:
  - *pliki skryptowe* zawierają sekwencje poleceń MATLAB-a i (lub) wyrażeń matematycznych (podrozdział 3.1);
  - *pliki funkcyjne* rozpoczynają się od zarezerwowanego słowa `function` i mają własną lokalną przestrzeń roboczą dla zmiennych (podrozdział 3.2);
  - pliki definiujące klasę zazwyczaj rozpoczynają się od zarezerwowanego słowa `classdef`.
- *.mlx* — interaktywne pliki skryptowe *live script*. Mogą one zawierać:
  - sekwencje poleceń MATLAB-a (skrypty) oraz funkcje lokalne;
  - dokumentację w postaci sformatowanych tekstów i grafiki, w tym tytuły, nagłówki rozdziałów i podrozdziałów, rysunki i fotografie oraz wzory matematyczne zapisane w programie LaTeX;
  - wyniki obliczeń, także w postaci tabel i wykresów.
- *.mlapp*, *mlappinstall*, *mlpkginstall* — aplikacja *App* i pliki instalacyjne.
- *.mat* (MAT-pliki) — są to skompresowane pliki binarne służące do przechowywania danych (w tym obrazów). Są obsługiwane przez `save` i `load` (podrozdział 3.6.4). Dodatkowe bezpłatne biblioteki w C i w Fortranie ułatwiają obsługę MAT-plików w tych językach.
- *.dat* — zalecane rozszerzenie plików tekstowych zawierających dane zapisane w kodzie ASCII (podrozdział 2.3.2).
- *.mdl*, *.slx*, *.mdlp*, *.slxp* — modele Simulinka i modele *proteced*. Format *.slx* jest zgodny z *Open Packaging Conventions* (OPC) i *Unicode® UTF-8* (rozdział 10.).
- *.slxc*, *.req*, *.slreqx*, *.slmx* — inne pliki Simulinka.
- *.mexa64*, *.mexmaci64*, *.dll* — MEX-pliki (ang. *MATLAB executable*) stosowane w celu przyspieszenia krytycznych fragmentów obliczeń i do sterowania w czasie rzeczywistym (podrozdział 3.6.5).
- *.p* — (ang. *protected*) pliki ze skompilowanym kodem. Więcej podaje `doc pcode`.
- *.asv* — (ang. *autosave*) kopia bezpieczeństwa dla pliku skryptowego, funkcji lub klasy.
- toolboksy — mogą używać własnych formatów plików, np. *Fuzzy Logic Toolbox* tworzy modele *.fis*, a *Simulink 3D Animation* wykorzystuje pliki zgodne ze standardem VRML97.

3.6.4. Zewnętrzne pliki z danymi — pliki ASCII i MAT-pliki

MATLAB może być użytkowany jako wyspecjalizowany element rozproszonego systemu obliczeniowego. Wymaga to wymiany danych pomiędzy MATLAB-em a innymi programami oraz urządzeniami. Obejmuje to pobieranie danych (import) oraz zapisywanie wyników obliczeń i danych na urządzeniu zewnętrznym (eksport).

Małe ilości danych można wprowadzić do MATLAB-a z klawiatury. Większe zbiory danych wygodnie jest przechowywać na dysku — w postaci binarnych MAT-plików lub plików znakowych ASCII.

Pliki ASCII mają zazwyczaj rozszerzenie *.dat*. Istotną wadą formatu ASCII jest konieczność zamiany liczby binarnej na dziesiętną przed jej zapisem. Powoduje to utratę dokładności danych. Z drugiej strony pliki ASCII są łatwe do odczytania. W razie potrzeby dane można korygować dowolnym edytorem.

Binarne MAT-pliki są dla MATLAB-a standardową postacią zapisu danych liczbowych i graficznych (polecenie `save nazwa_pliku lista_zmiennych`). Są też używane w wielu produktach i urządzeniach wykorzystywanych np. do archiwizacji danych pomiarowych, do modelowania i symulacji oraz do sterowania.

Zapis danych do MAT-pliku jest dokonywany z pełną dokładnością, bez zaokrągleń i bez potrzeby przeliczania liczb binarnych na dziesiętne. Dane binarne są lepiej upakowane. Ponadto do jednego MAT-pliku można zapisać wiele zmiennych (macierzy), a do pliku ASCII tylko jedną. Do odczytu lub zapisu MAT-plików (poza MATLAB-em) używa się dostarczanych wraz z MATLAB-em podprogramów (w C lub Fortranie). Znajdują się one w folderze *matlab\extern*. Eksport i import danych binarnych oraz tworzenie MAT-plików omówiono w podrozdziałach 2.3.2 i 2.3.3.

3.6.5. Binarne MEX-pliki wykonywalne

Użytkownik MATLAB-a może korzystać z pewnej liczby gotowych, skompilowanych i dynamicznie ładowanych programów zwanych MEX-plikami (ang. *Matlab EXecutable)*. Stosowanie MEX-plików zwiększa szybkość obliczeń i poprawia ochronę własności intelektualnej w kodzie skompilowanym, przeznaczonym do rozpowszechnienia. MEX-pliki wywołuje się w taki sam sposób jak funkcje MATLAB-a.

MEX-pliki są skutecznym narzędziem w zastosowaniach czasu rzeczywistego oraz do przetwarzania informacji zapisanych w różnych formatach. Korzystanie z MEX-plików jest bardzo efektywne, lecz ich przygotowanie wymaga użycia kompilatora, dostępnego za opłatą.

MEX-pliki są stosowane:

- jeśli konieczne jest zwiększenie szybkości obliczeń;
- gdy MATLAB współpracuje w czasie rzeczywistym z obiektami sprzężonymi z komputerem, takimi jak: czujniki i przetworniki, mikrofon, kamera TV, robot i inne;
- do obsługi obiektów regulacji w czasie rzeczywistym z poziomu MATLAB-a (poprzez specjalizowane układy sprzęgające);
- jeśli korzysta się z istniejącej biblioteki oprogramowania w językach C lub Fortran; programy te należy dostosować do specyfiki MATLAB-a, a w szczególności uzupełnić o podprogramy do zapisu i odczytu macierzy w formacie MAT-pliku.

3.7. Poprawa wydajności MATLAB-a

Aktualna wersja MATLAB-a jest szybsza od jego wersji wcześniejszych. W roku 2015 w MATLAB-ie zastosowano nowy silnik *MATLAB execution engine* i wciąż ulepszaną technologię JIT (ang. *just-in-time*). W MATLAB-ie nie ma wydzielonego etapu kompilacji. Kompilowane są kolejne linie interpretowanego programu, a uzyskany kod jest niezwłocznie realizowany i zapamiętywany — w celu ewentualnego późniejszego użycia.

Gdy skompilowany wcześniej moduł jest powtórnie obliczany, przebiega to znacznie szybciej.

Przyspieszenia nie będzie, jeśli skompilowana postać kodu została usunięta przez niepotrzebne użycie `clear all` zamiast `clear` bez parametrów.

Wprowadzono też inne ulepszenia:

- znaczne zmniejszenie czasu potrzebnego na *wywołanie funkcji* i powrót;
- szybsze działanie programów *obiektowo zorientowanych*;
- *memoize* — zapamiętanie wyniku obliczeń i ewentualne wykorzystanie go przy powtórnym wywołaniu funkcji, np. przy obliczeniach w pętli;
- stałe usuwanie niedoskonałości historycznych MATLAB-a.

3.7.1. Sposoby poprawiania wydajności programu

Poprawę wydajności MATLAB-a można uzyskać między innymi poprzez:

- niestosowanie `clear all`, ponieważ usuwany jest także skompilowany kod; wystarcza `clear` bez parametrów;
- prealokację, czyli rezerwowanie potrzebnej ilości miejsca na macierze i tablice już przed pierwszym ich użyciem, aby unikać czasochłonnego powiększania ich wymiarów;
- wektoryzację — zamiast obliczeń w pętli, należy stosować operacje macierzowe;
- umieszczanie w pętli tylko tego, co niezbędne, aby uniknąć wielokrotnych, zbędnych obliczeń;
- zastępowanie skryptu lub dużych funkcji przez niewielkie, wyspecjalizowane funkcje;
- stosowanie funkcje lokalnych, jeśli nie jest konieczne użycie innego typu funkcji;
- nieprzypisywanie danych innego typu do istniejącej zmiennej; należy użyć nowej zmiennej;
- unikanie zmiennych globalnych — zmienne globalne mogą zmniejszyć wydajność kodu MATLAB-a i są często powodem trudnych do wykrycia błędów;
- unikanie przeciążenia funkcji wbudowanych w MATLAB-a;
- używanie warunkowych operatorów logicznych (ang. *short-circuit*) `&&` i `||`, kiedy jest to możliwe; MATLAB nie musi badać drugiego argumentu, jeśli:
  - którykolwiek z operandów wyrażenia `&&` jest równy `0` lub
  - którykolwiek z operandów wyrażenia `||` jest równy `1`;
- nieobliczanie danych w programie — należy je obliczyć jednorazowo i zapisać jako MAT-plik do wielokrotnego użycia.

3.7.1.1. Unikanie konkretnych funkcji i poleceń w programie MATLAB

- Niestosowanie `clear all`, ponieważ usuwany jest także skompilowany kod. Wystarcza `clear` bez parametrów.
- Unikanie funkcji: `inputname`, `which`, `whos`, `exist(var)`, `dbstack` — są one kosztowne obliczeniowo.
- Unikanie funkcji `eval`, `evalc`, `evalin` i `feval` — są one jest kosztowne obliczeniowo.
- Unikanie korzystania w programach z poleceń: `cd`, `addpath` i `rmpath`. Powodują one potrzebę rekompilacji kodu.
- Unikanie poleceń wykonywanych przez system operacyjny.

3.7.2. Wektoryzacja, operacje macierzowe i tablicowe

Język MATLAB umożliwia efektywną realizację operacji tablicowych i macierzowych (w tym na macierzach rzadkich).

Dobry program w języku MATLAB nie powinien zawierać pętli `for` ani `while`, gdyż zazwyczaj można je zastąpić notacją dwukropkową, operacjami tablicowymi, funkcjami do generowania macierzy (na przykład `ones`) i operacjami na macierzach (na przykład `tril`). Stosowanie pętli może być jednak konieczne przy obsłudze macierzy wielowymiarowych.

Istotą wbudowanych w MATLAB-a operacji wektorowych i macierzowych jest **niejawna indeksacja** elementów wektora lub macierzy. Pozwala to na efektywną wektoryzację obliczeń i zwarty ich zapis poprzez konstrukcje z użyciem dwukropka, nawiasów kwadratowych oraz połączenie kropki i operatorów arytmetycznych (*notacja kropkowa*). W operacji tablicowej `C=A.*B` nie używa się jawnie indeksów, pomimo że operacje mnożenia są wykonane na parach elementów wektora lub macierzy. W starych wersjach MATLAB-a (bez akceleratora JIT) instrukcje iteracyjne nie były przekształcane na operacje wektorowe. Powodowało to znaczne zmniejszenie szybkości tych obliczeń. Zalecenia:

- Stosowanie notacji kropkowej, czyli operacji tablicowych (np. `.*  ./  .^  .\`) do zapisu działań na elementach wektorów, macierzy lub tablic, np.: `C=A.*B`.
- Stosowanie operatora (`:`) do wskazania kolumny, wiersza lub fragmentu macierzy.
- Stosowanie funkcji macierzowych z bibliotek MATLAB-a (np. `B=tril(C)`). Funkcje te (patrz `doc elmat`) mogą być użyte zarówno dla macierzy pełnych, jak i rzadkich. Opis funkcji przeznaczonych tylko dla macierzy rzadkich uzyskuje się za pomocą polecenia `doc sparfun`.

3.7.3. Rezerwowanie pamięci na macierze i wektory

Domyślne określanie typów i wymiarowanie tablic umożliwia szybkie przygotowanie obliczeń w MATLAB-ie. Powoduje to jednak istotne spowolnienie obliczeń, jeśli MATLAB musi zwiększać rozmiar macierzy po każdorazowym obliczeniu wartości kolejnego

jej elementu. Jeśli jednak znany jest rozmiar (lub maksymalny rozmiar) generowanej macierzy, to można uzyskać niekiedy znaczne przyspieszenie obliczeń. Pamięć można **rezerwować** poprzez

- **utworzenie** dowolnej *macierzy specjalnej* o wymaganej wielkości (np. `ones(6)`);
- generowanie wektorów i macierzy z użyciem notacji dwukropkowej (np. `n=1:5`);
- wykorzystanie funkcji generujących macierze specjalne, na przykład `ones`, `zeros`, `eye`.

3.7.4. Programowanie zorientowane obiektowo

Zaleca się stosowanie programowania obiektowo zorientowanego (rozdział 6.) we wszystkich większych pracach wykonywanych z użyciem MATLAB-a. Dotyczy to szczególnie przypadków, gdy oprogramowanie jest wykonywane zespołowo lub jeśli przewiduje się wielokrotne modyfikowanie programu, aby go dostosować do zmiany wymagań użytkownika lub do zmiany norm i przepisów prawa.

3.8. Uwagi dla zaawansowanego użytkownika

3.8.1. Podział programu na sekcje (%% cell)

Program w MATLAB-ie może być podzielony na sekcje (ang. *cell*). Są one oddzielone liniami komentarza z podwójnym znakiem i nazwą sekcji (patrz przykład 3.2.4-1). Nazwy sekcji i komentarze są też wykorzystywane do automatycznego tworzenia dokumentacji (podrozdziały 3.8.1.2 oraz 3.8.1.3).

Przykład 3.8.1-1

Podział kodu programu *celltest.m* na trzy sekcje z użyciem linii z komentarzem `%%` (rysunek 3.8.1-1).

Rysunek 3.8.1-1. *Podział kodu na sekcje z użyciem linii rozpoczętej przez %%*

celltest.m

```
1    %% plik celltest.m
2 -  clear, clc %usunięcie  zmiennych  z pamięci roboczej
3 -  t=0:0.02:20; % czas t, zmienna niezależna
4
5    %%  dane dla kilku wykresów
6 -  s0=sin(t);  s1=sin(t+1);  s2=sin(t+2);
7
8    %% wykresy
9 -  plot(t,s0)  % jeden wykres
10   %% nowe okno i 3 wykresy
11 - figure,plot(t,s0,t,s1,':',t,s2,'-.')  % nowe okno i 3 wykresy
```

3.8.1.1. Testowanie sekcji programu w edytorze

Podział programu na sekcje poprawia efektywność pracy programisty z uwagi na możliwość wielokrotnego modyfikowania i testowania wybranej sekcji — bez potrzeby wykonywania całego programu.

Kliknięcie myszą dowolnej sekcji spowoduje, że stanie się ona sekcją aktualną i będzie mieć żółte tło. Kliknięcie w edytorze ikony uruchamia wykonanie zawartości aktualnej sekcji. Ikona uruchomi wykonanie aktualnej sekcji oraz dodatkowo zaznaczy sekcję następną jako aktualną. Uruchomienie aktualnej sekcji jest też możliwe przez kombinację klawiszy *Ctrl+Enter*.

3.8.1.2. Tworzenie dokumentacji — Live Editor i raport PUBLISH

Po opracowaniu poprawnej wersji programu warto go wydrukować w formie eleganckiego raportu. Skrypty i funkcje można otworzyć i wydrukować w *Live Editor* w sposób opisany w podrozdziale 3.8.1.3.

Więcej możliwości daje usługa *PUBLISH* dostępna dla skryptów, funkcji i klas. *PUBLISH* opisano w podrozdziale 3.8.1. Bardziej zaawansowane raporty można przygotować z użyciem dostępnych za opłatą narzędzi: MATLAB Report Generator i Simulink Report Generator.

3.8.1.3. Live Editor i funkcja z przykładu 3.2.4-1

Wykorzystanie *Live Editor* do obsługi *plików skryptowych* przedstawiono w podrozdziale 3.1.2. Skrypt może zawierać funkcje lokalne, co umożliwia edytowanie funkcji poprzedzonej co najmniej jedną linią skryptu. W rezultacie pierwsze linie nowej wersji pliku z przykładem 3.2.4-1 będą wyglądać następująco:

Przykład 3.8.1.3-1

Pierwsze linie przykładu 3.2.4-1. Do pierwszej linii wstawiono skrypt, który wywołuje edytowaną funkcję. Nazwę funkcji zmieniono na `lokataOdnaw3`, aby uniknąć kolizji nazw.

```
kapital = lokataOdnaw3(4,3,100,9) % 4%, 3 mies, 100 PLN, 9 mies
%% skrypt i funkcje lokalne - oblicz nowy kapitał po oprocentowaniu lokaty odnawialnej
function kapital = lokataOdnaw3(oprocRok,okres,kapital,ileMiesiecy_wBanku)
%
%  dalsze linie programu jak w przykładzie 3.2.4-1
```

Jak widać, skrypt składa się z jednej linii `kapital = lokataOdnaw3(4,3,100,9)` oraz z pozostawionej linii komentarza. W linii 3. rozpoczyna się `function kapital`. Ostatnia pokazana linia to początek kolejnego segmentu kodu rozpoczynający się komentarzem *%% Obliczanie zysku*; dalsze linie programu jak w przykładzie 3.2.4-1.

Skrypt z przykładu 3.2.4-1 ze zmianami opisanymi w przykładzie 3.8.1-3 należy zapisać pod dowolną nazwą:

- jako *Live Script* z użyciem zakładki *EDITOR* panelu MATLAB oraz menu *Save/Save As*, wybierając w kolejnym oknie *Save as type*: *MATLAB Live Scripts*;
- lub jako skrypt z rozszerzeniem *.m*, a następnie kliknąć prawym przyciskiem myszy jego nazwę w oknie *Current Folder* i z menu kontekstowego wybrać *Open as Live Script*.

Dalsze czynności są opisane w podrozdziale 3.1.2.

3.8.1.4. Raport PUBLISH dla przykładu 3.2.4-1 — przygotowanie

Narzędzie *PUBLISH* przygotowuje raport opisujący strukturę i działanie kodu przygotowanego w języku MATLAB. Można określić wygląd opublikowanego raportu i jego format (do wyboru *HTML*, *XML*, *LATEX*, *PDF*, *DOC*, *PPT*). Jeśli w pliku są funkcje, to wymagają one określenia wartości parametrów formalnych w chwili ich wywołania.

Kod powinien być podzielony na sekcje z użyciem komentarzy rozpoczynających nową linię podwójnym znakiem *%%*. Komentarze rozpoczynające się od *%%* będą użyte jako tytuły rozdziałów tego raportu. Mogą one opisywać zadania wykonywane w kolejnych sekcjach programu. Wszystkie sekcje będą kolejno uruchomione, opisane i zilustrowane wynikami obliczeń i wykresem — jeśli wykres był wykonany.

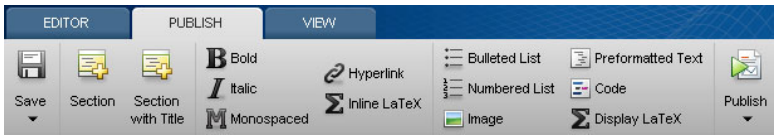
Polecenie publish w oknie Command

Narzędzie *PUBLISH* można wywołać z okna *Command*. Pokazane niżej polecenie wykona raport dla pliku *lokataOdnaw.m* z przykładu 3.2.4-1. Pierwszym parametrem jest nazwa pliku, dalej umieszcza się pary: nazwa parametru i jego wartość. Tutaj użyto dwóch par: kodu *wywołania funkcji* oraz *formy raportu*. Parametr *outputDir* może być użyty do wskazania folderu, gdzie raport będzie zapisany. Tutaj folderu nie wskazano i ścieżka dojścia do raportu została wpisana do zmiennej `ans`.

```
>> publish('lokataOdnaw.m','codeToEvaluate',...
           'kapital = lokataOdnaw(4,3,100,6)','format','doc')
ans = 'C:\Users\Zbigniew\Documents\MATLAB\2017examples\03\html\lokataOdnaw.doc'
```

Zakładka PUBLISH na panelu MATLAB

Innym sposobem wygenerowania tego raportu jest otwarcie pliku *lokataOdnaw.m* w edytorze MATLAB-a. Następnie po wybraniu zakładki PUBLISH, a w menu *Publish/Edit Publishing Options* i kliknięciu ikony *Publish* (rysunek 3.8.1.4-1) należy wybrać menu *Edit Publishing Options.*



Rysunek 3.8.1.4-1. *Zakładka PUBLISH z opcjami do formatowania publikacji pliku*

Otworzy się okno *Edit Configurations* i w okienku *MATLAB expression* należy wpisać linię wywołania edytowanej funkcji.

```
kapital = lokataOdnaw(4,3,100,6)     % 4%, 3 mies., 100 PLN, 6 mies.
```

W okienku *Publish settings* należy sprawdzić i skorygować inne parametry raportu, w tym jego format. Można wybrać: *HTML*, *LATEX*, *DOC*, *XML*, *PPT*, *PDF*.

Po sprawdzeniu poprawności ustawień można kliknąć przycisk *Publish*. MATLAB wygeneruje raport w wybranym formacie. Poniżej podano tekst raportu utworzonego przez narzędzie PUBLISH.

3.8.1.5. Raport PUBLISH dla przykładu 3.2.4-1 — wydruk

Pierwszym elementem raportu jest spis treści, utworzony z komentarzy umieszczonych w liniach rozpoczynających się od *%%*. Następnie pojawią się opisy oraz wyniki obliczeń i rysunki wykonywane w kolejnych segmentach kodu.

Funkcja główna i lokalna — oblicz nowy kapitał po oprocentowaniu lokaty odnawialnej

```
function kapital = lokataOdnaw(oprocRok,okres,kapital,ileMiesiecy_wBanku)
% kapital = lokataOdnaw(4,3,100,6) % 4%, 3 mies., 100 PLN, 6 mies.
zainwestowano = kapital;
ileOkresowLokaty = fix(ileMiesiecy_wBanku/okres);    % liczba całkowita
```

Obliczanie zysku za każdy okres lokaty

```
while(ileOkresowLokaty)    % wykonanie tylko gdy ileOkresowLokaty ? 0
    kapital = lokata(oprocRok,okres,kapital);
    ileOkresowLokaty = ileOkresowLokaty-1;
end    % while end
zysk = kapital-zainwestowano;
end    % koniec funkcji głównej lokataOdnaw
```

Funkcja lokata oblicza nowy kapitał po pierwszym okresie lokaty

```
function [kapital] = lokata(oprocRok,okresLokaty,kapital)
% [nowyKapital,zysk] = lokata( 3.5,3,100) % przykład wywołania
```

Obliczenia

```
procentyZaOkresLokaty = oprocRok/12*okresLokaty;
zysk = procentyZaOkresLokaty/100*kapital;
kapital = kapital+zysk;    % kapitał powiększony o zysk
end    % koniec funkcji lokalnej lokata
kapital =
  102.0100
```

3.8.2. Wielokrotne wykorzystanie tworzonych funkcji

Wielokrotne wykorzystanie tworzonych funkcji (ang. *reusability*) pozwala użyć tworzonego oprogramowania w następnych projektach. Poniżej podano wskazówki, które ułatwiają wielokrotne wykorzystanie kodu:

- Stosowanie komentarzy objaśniających przeznaczenie segmentów kodu, jego parametrów wejściowych i wyjściowych oraz zastosowanych algorytmów i użytych jednostek (np. mm, cm). Odpowiednio umieszczone komentarze są niezbędne do prawidłowego działania poleceń `help`, `where`, `what`, `lookfor` (podrozdział 2.7) oraz do automatycznego przygotowania dokumentacji.
- Skrypty powinno się traktować jako tymczasową (łatwą do zapisu i testowania) postać algorytmu przed jego przetworzeniem na funkcje.
- Funkcje, w odróżnieniu od skryptu, operują na zmiennych lokalnych (nieprzechowywanych w przestrzeni roboczej MATLAB-a). Chroni to przed błędami, które są powodowane niezamierzonym użyciem tych samych nazw dla różnych zmiennych.
- Skrypty i funkcje powinny poprawnie działać zarówno dla wielkości skalarnych, jak i macierzowych. Dobrym sprawdzianem poprawności programu jest użycie macierzy specjalnych (`ones`, `zeros`, `magic`), macierzy pustej `[]` oraz macierzy stochastycznych lub zespolonych jako dane testowe.

- Stosowanie zrozumiałych dla przyszłego użytkownika nazw zmiennych. Przykładem mogą być nazwy: `napiecie_wejsciowe` lub krócej `NapWejsc` — dla zmiennej reprezentującej wartości napięcia sygnału wejściowego.
- Tworzenie plików *Contents.m* w folderach zawierających własne pliki użytkownika. Umożliwia to uzyskanie informacji o plikach umieszczonych w danym folderze za pomocą polecenia *help nazwa_folderu* (podrozdział 2.7).
- Prawie gotowe, przetestowane programy powinny być przenoszone do oddzielnego folderu. Należy dopisać ten folder do ścieżek przeszukiwanych przez MATLAB-a (podrozdział 3.6.2). Dodatkowo powinno się zweryfikować program z użyciem narzędzia *M-Lint* oraz poprawić jego czytelność poprzez użycie opcji *indent* edytora. Należy też uzupełnić i uaktualnić komentarze, wydruk gotowego programu i dodatkową jego kopię archiwalną na przykład na pendrivie lub w chmurze. Weryfikację programów zaleca się ponawiać z użyciem kolejnych wersji M-Lint, dostarczanych z coraz nowszymi wydaniami MATLAB-a.

3.8.3. Raporty TODO/FIXME i inne

Notatki TODO są przeznaczone do opisywania prac, które powinny być wykonane w przyszłości. Są to umieszczone w programach komentarze zawierające słowo *TODO* lub *FIXME*, np. *% zerowy wektor 2x1, potrzebny tylko jeden raz (TODO)*.

Program z tą notatką zostanie odszukany i wyświetlony w edytorze po wybraniu opcji *TODO/FIXME Report*, dostępnej po kliknięciu ikony (mały trójkąt w okręgu) po prawej stronie nagłówka *Current Folder* w panelu MATLAB-a. Następnie wybiera się opcję *Reports/TODO/FIXME Report*. Na tym etapie można wskazać inne, dodatkowe słowo kluczowe. Przykładowy raport, obejmujący wszystkie pliki MATLAB-a z aktualnego folderu, pokazano na rysunku 3.8.3-1.

Rysunek 3.8.3-1. *Raport TODO/FIXME dla folderu zawierającego plik z notatką TODO*

MATLAB File List

kalkulator	
ode1000	6 xdot=zeros(2,1);% zerowy wektor 2x1, (TODO) potrzebny tylko jeden raz
test23a	
test23aProfil	

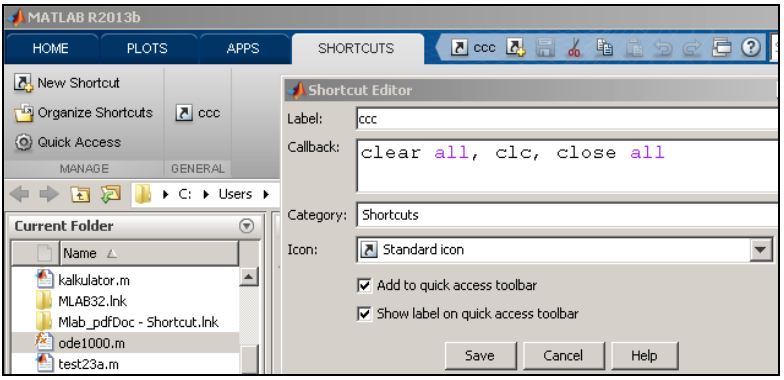
Zaletą tego narzędzia jest możliwość łatwego odszukania notatek — nawet jeśli nie zna się nazwy ani położenia pliku, w którym umieszczono notatkę.

Opisane wyżej menu daje dostęp do raportów:

- *Code Analyzer Report*
- *TODO/FIXME Report*
- *Help Report*
- *Contents Report*
- *Dependency Report*
- *Coverage Report*

3.8.4. Shortcuts, czyli skróty

Często używane sekwencje poleceń można zapamiętać w postaci skryptu lub skrótu. Skrót jest wygodniejszy, gdyż uruchamia się go, klikając ikonę umieszczoną w górnej części okna MATLAB-a. Autor często używa własnego skrótu `ccc`, realizującego: (1) usunięcie wszystkich zmiennych z przestrzeni roboczej, (2) zamknięcie wszystkich okien graficznych i (3) wyczyszczenie zawartości okna *Command Window*. Taki skrót można utworzyć, klikając ikonę New Shortcut i wypełniając pola w oknie *Shortcut Editor*, pokazanym na rysunku 3.8.4-1.



Rysunek 3.8.4-1. *Okno Shortcut Editor i nowy skrót ccc na pasku szybkiego dostępu i na karcie Shortcuts*

3.8.5. Preferencje — przygotowanie środowiska do pracy

Już po kilku tygodniach pracy z MATLAB-em użytkownik czuje potrzebę poprawy własnego komfortu oraz zwiększenia wydajności swojej pracy.

W razie potrzeby można ustawić w *Home/Preferences/MATLAB* własne preferencje dotyczące np. folderu roboczego, wielkości czcionek i formatu A4 do wydruków. Jeśli z komputera korzysta więcej niż jedna osoba, to każdy użytkownik może przygotować własny skrypt z potrzebnymi ustawieniami.

3.8.6. Mysz czy klawiatura?

Kolejne wersje MATLAB-a są coraz bardziej przyjazne dla użytkownika. Wiele zaawansowanych czynności można wykonać bez znajomości poleceń MATLAB-a, obsługując myszą graficzny interfejs użytkownika. Jest to bardzo wygodne i przyspiesza wykonanie wielu zadań.

Jeśli jednak zachodzi potrzeba wielokrotnego wykonania identycznych lub podobnych zadań, na przykład wykonanie setki podobnych wykresów — korzystanie z myszy staje się uciążliwe. Lepszym wariantem jest jednorazowe ustawienie wszystkich potrzebnych parametrów wykresu w postaci skryptu lub funkcji i następnie wielokrotne użycie go do przetworzenia kolejnego zestawu danych.

Mysz może niekiedy zastąpić klawiaturę podczas pisania programu. Jeśli potrzebne nam polecenia były już wcześniej użyte, to prawdopodobnie mogą być odnalezione w oknie *History* i przeniesione myszą do okna edytora — bez potrzeby ich powtórnego zapisywania z użyciem klawiatury.

3.8.7. Funkcje eval, feval i podobne

Funkcje `eval`, `evalc`, `evalin` oraz `feval` pozwalają na użycie utworzonego przez program łańcucha jako kolejnego polecenia MATLAB-a. Są to bardzo *mocne instrukcje*, gdyż do wykonywanych poleceń, mogą dopisać argumenty, które *nie były znane podczas pisania programu* (na przykład nazwy plików).

- Argumentem wejściowym większości omawianych funkcji jest *wyrażenie tworzące łańcuch*. Ten łańcuch to jedno lub kilka poleceń MATLAB-a oddzielonych przecinkiem lub średnikiem. Łańcuch tworzy się, sklejając (*konkatenacja*) *w nawiasie kwadratowym* łańcuchy i zmienne zawierające łańcuchy. Polecenia zawarte w utworzonym łańcuchu są niezwłocznie wykonywane.
- `eval` ma tylko jeden parametr wejściowy: wyrażenie tworzące łańcuch.
- `evalc` działa jak `eval`, ale to, co `eval` wyświetla na ekranie (poza ewentualnym komunikatem o błędzie), będzie wpisane do zmiennej wyjściowej.
- `evalin` działa jak `eval`, z tym że *wyrażenie tworzące łańcuch* jest drugim parametrem. Pierwszy parametr ma wartość `'base'` lub `'caller'` i decyduje o tym, czy polecenia będą wykonane w bazowej przestrzeni roboczej, czy w lokalnej przestrzeni funkcji wywołującej.
- `feval` — jej pierwszym argumentem wejściowym jest nazwa funkcji lub uchwyt do funkcji. Drugi argument `feval` to lista argumentów tej funkcji. Lista argumentów jest *wyrażeniem tworzącym łańcuch*. Efektem działania `feval` jest wykonanie funkcji z zadaną listą parametrów.

Funkcja `eval` (oraz `evalc`, `evalin`, `feval`) jest bardzo mocna, ale spowalnia wykonywanie programu. Wyjaśniono to w podrozdziale 3.8.7.2.

Dalszą część podrozdziału 3.8.7 można pominąć przy pierwszym czytaniu, gdyż omawiane tu funkcje nie są często używane.

Przykład 3.8.7-1

Przygotowanie skryptu tworzącego pliki `x1`, `x11`, `x2`,`x22`, `x3`, `x33` z danymi do testowania w następnym przykładzie:

```
% przygotowuje pliki z danymi dla przykładu 3.8.7-2
clear
x1 = 1:2:11; x11 = x1.^2;
   save dane1    % plik dane1.mat zawiera wszystkie zmienne z przestrzeni roboczej
   clear         % usuwa wszystkie zmienne z przestrzeni roboczej
x2 = 2:3:20; x22=x2+2;    % generuje nowe zmienne
   save dane2
   clear
x3 = 3:9:39; x33=x3+3;    % generuje nowe zmienne
   save dane3
   clear
dir dane*        % podaje nazwy plików rozpoczynające się od "dane"
```

Ostatnie polecenie poda wykaz plików o nazwie rozpoczynającej się od *dane*. Są to: *dane1.mat*, *dane2.mat* oraz *dane3.mat.* Poniżej pokazano, jak można utworzyć nazwy plików z danymi przeznaczone do testowania funkcji `ldanef`.

Przykład 3.8.7-2

Przygotowano skrypt, który generuje nazwy plików z danymi i wczytuje je. Liczba pobieranych plików jest wpisana do zmiennej `n`. Nazwa pierwszego pliku to *dane1.mat*, następne pliki to *dane2.mat* oraz *dane3.mat*. W tym skrypcie wczytywane są pliki utworzone w przykładzie 3.8.7-1. Użyto funkcji `feval`.

```
% wczytywanie plików z przykładu 3.8.7-1
% potrzebne pliki: dane1, ... dane3 — wpierw wykonaj przykład 3.8.7-1
clear, clc    % usuwa zmienne, czyści okno Command
name0 = 'dane';    % pierwsze 4 litery nazwy pliku
n=3;    % wczyta 3 pliki z danymi
for i = 1:n
   feval('load' ,[name0, int2str(i),'.mat'])    % feval wczyta kolejny plik
end
%
who
```

Należy zwrócić uwagę, że w razie potrzeby liczbę plików i wspólną część ich nazw można wczytać z klawiatury poleceniami:

```
name0 = input('podaj pierwsze znaki nazwy pliku ','s');    % np. dane
n=input('podaj liczbę, ile plików z danymi będzie wczytane? ');    % podaj liczbę, np. 1
                                                                    % lub 2, lub 3
```

Drugi parametr w postaci `'s'` oznacza, że z klawiatury będzie wpisany tekst; brak parametru oznacza, że wprowadzona będzie liczba (np. 5) lub wyrażenie mające wartość liczbową (3*8).

3.8.7.1. Uwagi do przykładu 3.8.7-2

Drugim parametrem `feval` jest nazwa pliku do wczytania; jest ona tworzona przez sklejenie łańcuchów:

- wartości przypisanej do zmiennej `file0` (łańcuch `'dane'`),
- liczby (`i`) przekształconej w łańcuch poleceniem `int2str(i)`,
- rozszerzenia nazwy pliku *'.mat'* (ten łańcuch można pominąć, gdyż *.mat* jest domyślnym rozszerzeniem polecenia `load`).

Siła funkcji `eval` (i jej podobnych) polega na możliwości wygenerowania (podczas wykonywania programu) dodatkowych poleceń, które będą niezwłocznie wykonane w tym samym programie.

3.8.7.2. Funkcja eval spowalnia działanie MATLAB-a

Fragmenty programu zawierające funkcje `eval` (i jej podobne) są na etapie kompilacji pomijane, gdyż polecenia, które będą wygenerowane przez `eval`, nie są jeszcze wtedy znane. W rezultacie funkcje te podczas kolejnych przebiegów programu będą wykonywane wolniej od kodu skompilowanego. Z drugiej strony proces przygotowania i uruchomienia kodu wykorzystującego funkcje `eval` (i jej podobne) może być znacznie szybszy niż bez ich użycia. Może się to okazać ważniejsze od późniejszego spowolnienia obliczeń.

Artykuł *http://www.mathworks.com/help/matlab/matlab_prog/string-evaluation.html* pokazuje sposoby zastąpienia `eval` przez inny kod.

3.8.8. Funkcje o zmiennej liczbie parametrów

W wywołaniu funkcji można zastąpić znakiem tyldy (~) nazwy aktualnie *niepotrzebnych zmiennych* wyjściowych oraz niepotrzebnych argumentów wejściowych. Takie postępowanie przyspiesza działanie programu i oszczędza miejsce w przestrzeni roboczej MATLAB-a. Można też pominąć potrzebne argumenty, pozostawiając w razie potrzeby przecinki.

Jeśli sposób działania funkcji ma zależeć od liczby argumentów użytych przy jej wywołaniu, należy użyć zmiennych `nargin` oraz `nargout`. Reprezentują one aktualną dla danego wywołania funkcji liczbę użytych argumentów wejściowych (`nargin`) i wyjściowych (`nargout`).

Zmienna `varargin` na liście parametrów wejściowych umożliwia użycie dowolnej liczby parametrów, których nie określa się w definicji tej funkcji. Podobną rolę dla parametrów wyjściowych odgrywa `varargout`.

3.8.8.1. Przykład użycia zmiennej nargin oraz funkcji feval

Funkcja `nargin_test` wczytuje pliki utworzone w przykładzie 3.8.7-1. Ma ona trzy argumenty wejściowe. Pierwszy, `fileName`, jest łańcuchem reprezentującym nazwy plików z zestawami danych, które mogą różnić się między sobą jedynie liczbą porządkową. Drugi parametr podaje liczbę wczytywanych zestawów. Trzeci parametr, `folder`, jest łańcuchem określającym ścieżkę dostępu do wczytywanych danych.

Zmienna `nargin` umożliwia warunkowe wykonanie wybranych poleceń zależnie od liczby argumentów wejściowych użytych w wywołaniu funkcji.

Przykład 3.8.8.1-1

Zmienna `nargin` jest równa liczbie argumentów użytych w wywołaniu funkcji, w tym przykładzie może mieć wartość 1, 2 lub 3. Zakłada się, że nazwa pliku składa się ze słowa `dane` i kolejnej liczby, zaczynając od `dane1`. Rozszerzeniem pliku jest `.mat`. Działanie omawianego programu zależy od liczby parametrów użytych przy jego wywołaniu.

- Jeśli jedynym parametrem będzie *nazwa pliku* (skrócona, bez kolejnego numeru i bez rozszerzenia), to wskazany plik będzie wczytany, jeśli tylko znajduje się w folderze aktualnym.
- Jeśli parametrami będą *nazwa pliku* oraz `n` — *liczba plików*, to wczytane będzie `n` kolejnych plików, jeśli tylko znajdują się w folderze aktualnym.
- Jeśli będą użyte trzy parametry: *nazwa pliku*, `n` — *liczba plików* oraz *ścieżka dojścia do folderu z plikami*, to wczytane będzie `n` kolejnych plików ze wskazanego folderu.

Sterowanie wykonaniem funkcji `nargin_test` jest realizowane poprzez wartość zmiennej `nargin`. Dalsze modyfikowanie funkcji `nargin_test` poprzez wprowadzenie zmiennej `nargout` oraz `varargin` i `varargout` pozostawia się Czytelnikowi.

3.9. Optymalizacja programu z użyciem profilera

Profiler jest narzędziem analizującym zużycie czasu procesora podczas realizacji kolejnych linii badanego programu. Korzyścią ze stosowania profilera jest to, że nawet niewielkie usprawnienie programu w miejscu wskazanym przez profiler daje dużo lepsze efekty niż optymalizowanie innych elementów programu. Profiler można uruchomić ikoną *Run_and_Time* (trójkąt z zegarkiem) w zakładce *HOME* lub *EDITOR*.

Przykład 3.9-1

Wykorzystano profiler do zbadania czasów realizacji każdej linii skryptu *test23a.m* i funkcji `ode1000` oraz czasów realizacji funkcji bibliotecznej `ode23` podczas rozwiązania równania różniczkowego zadanego funkcją `ode1000`. Rozwiązywanie *równań różniczkowych* omówiono w podrozdziale 8.2. Skrypt *test23a.m* przygotowuje dane dla `ode23`.

```
% test23a.m
t0= 0; t2= 3;       % zadany przedział czasu
options=odeset('AbsTol',1e-1,'stat','on');  % opcje dla ode23
x0= [1 -1];         % warunki początkowe, wartości dla t=0
[t,x] = ode23s('ode1000', [t0,t2], x0, options);
```

Funkcja `ode1000` oblicza pierwszą i drugą pochodną `xdot(1)` i `xdot(2)` prawej strony równania różniczkowego (8.2.5-1), w zadanej chwili `t` dla aktualnego punktu `x`.

```
%% Ordinary differential eq./ Równanie różniczkowe zwyczajne
function xdot=ode1000(t,x); % called from ode23
% use ode23 or ode45 solver / użyj ode23 lub ode45
% x''= -1001x' -1000x; Bjorck & Dalquist "Num. Methods, 1974"

xdot=zeros(2,1);    % zerowy wektor 2x1, (TODO) potrzebny jeden raz
xdot(1)=x(2);
xdot(2)=-1001*x(2) -1000*x(1);
```

W profilu funkcji `test23a` (rysunek 3.9-1) podano, że *na łączny czas obliczeń 0.734 s przypada 0.539 s na ode23 i 0.147 s na ode1000*. Rysunek 3.9-2 pokazuje czas obliczeń każdej z trzech wykonywanych linii funkcji `ode1000.m`. Najwięcej czasu pochłonęło tu zerowanie wektora `xdot` przy każdym obiegu pętli tej funkcji: 8, 6, 7. Linia 6. jest potrzebna tylko przy pierwszym wywołaniu, a była wykonana 2872 razy. To należy poprawić.

Rysunek 3.9-1. *Raport profilowania programu test23a*

Profile Summary

Generated 28-Feb-2017 22:09:33 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
test23a	1	0.734 s	0.003 s	
ode23	1	0.710 s	0.539 s	
ode1000	2872	0.147 s	0.147 s	
odeset	1	0.021 s	0.013 s	

Z raportu profilera widać, że w porównaniu z własnym podprogramem `ode1000` program biblioteczny `ode23` wymaga znacznie więcej czasu na obliczenia. W tej sytuacji nie warto zaczynać optymalizacji od podprogramu `ode1000`. W zamian należy przyspieszyć

Rysunek 3.9-2. *Raport profilowania dla funkcji ode1000, która jest wywoływana z ode23 (linia 5. w ode23a)*

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
6	`xdot=zeros(2,1);% zerowy wekto...`	2872	0.057 s	39.0%	
8	`xdot(2)=-1001*x(2) -1000*x(1);`	2872	0.021 s	14.6%	
7	`xdot(1)=x(2);`	2872	0.005 s	3.3%	
All other lines			0.063 s	43.1%	
Totals			0.147 s	100%	

pracę programu bibliotecznego `ode23` poprzez dobranie mu odpowiednich opcji *odeset*. Należy też wypróbować solvery inne niż `ode23`, na przykład `ode45`, `ode23s`.

Zmiana solvera `ode23` na `ode23s` dała radykalną poprawę, co widać na rysunku 3.9-3.

Rysunek 3.9-3. *Raport profilowania test23a — po zmianie solvera ode23 na ode23s*

Profile Summary

Generated 28-Feb-2017 23:02:45 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
test23a	1	0.172 s	0.001 s	
ode23s	1	0.151 s	0.062 s	

Nie było żadnych zmian w `ode1000`, ale z uwagi na mniejszą liczbę wywołań z solvera (tylko 92 razy zamiast 2872) czas obliczeń w tej funkcji zmalał radykalnie (rysunek 3.9-4).

Rysunek 3.9-4. *Raport profilowania dla funkcji ode1000, która jest wywoływana z ode23s, a nie z ode23 — jak poprzednio (linia 5. w ode23a)*

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
6	`xdot=zeros(2,1);% zerowy wekto...`	52	0.002 s	49.3%	
8	`xdot(2)=-1001*x(2) -1000*x(1);`	52	0.000 s	13.8%	
7	`xdot(1)=x(2);`	52	0.000 s	3.1%	

3.10. Jaja wielkanocne

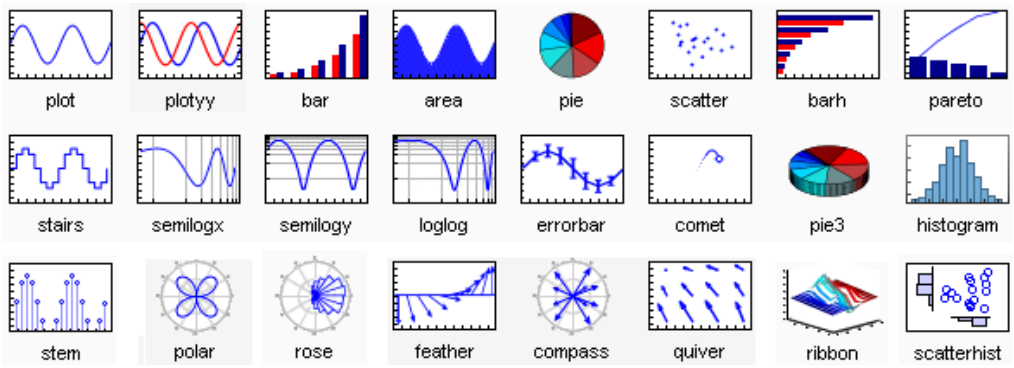
Jaja wielkanocne (ang. *Easter eggs*) to zazwyczaj żartobliwa, nieudokumentowana wstawka do programu. Aktualnie jaja są widoczne w MATLAB-ie po wywołaniu:

- `why` — losowane są zwięzłe i dowcipne odpowiedzi na pytanie: *dlaczego?*

Na stronach internetowych można znaleźć opisy kilkunastu innych jaj z MATLAB-a, ale zazwyczaj są to opisy udokumentowanych plików demo z MATLAB-a i Simulinka, które bez istotnej przyczyny rozbawiły internautów.

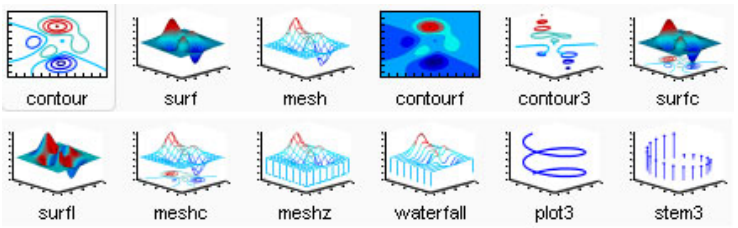
Rozdział 4.
Wykresy w MATLAB-ie

MATLAB udostępnia zestaw łatwych w stosowaniu narzędzi do wizualizacji w postaci wykresów. Sposób wykonywania prostych rysunków z użyciem ikon z zakładki *PLOTS* opisano w podrozdziale 2.2.7. Na rysunku 4-1 zestawiono ikony wybranych funkcji, które wykonują różnego rodzaju wykresy w skali liniowej, logarytmicznej, półlogarytmicznej oraz we współrzędnych biegunowych. Skrócony opis funkcji reprezentowanej przez wybraną ikonę pojawia się w dodatkowym okienku po zatrzymaniu myszy na tej ikonie.



Rysunek 4-1. *Wybrane ikony wykresów 2D dostępnych w zakładce PLOTS MATLAB-a*

Sposoby wywołania tych funkcji są podobne do wariantów wywołania funkcji `plot`. Dokładny opis funkcji wykonujących wykresy dwuwymiarowe uzyskuje się poleceniami: `doc graph2d` i `doc specgraph`. Wizualizację powłok i brył trójwymiarowych pokazano na rysunku 4-2. Sposoby wywołania tych funkcji podaje `doc graph3d`.



Rysunek 4-2. *Wybrane ikony powłok, brył i wykresów 3D dostępnych w zakładce PLOTS*

Wykresy mogą być automatycznie aktualizowane po każdej zmianie wartości danych. Najprościej uzyskuje się to poprzez kliknięcie ikony *link plot* w oknie z gotowym wykresem. W MATLAB-ie dostępne są też proste możliwości animacji obrazu oraz generowania dźwięków stereo. Opis graficznego interfejsu użytkownika **GUI**, czyli sposobów tworzenia przycisków, suwaków, okienek dialogowych i menu, podano w rozdziale 6.

Większość funkcji graficznych zapewnia automatyczne skalowanie osi. Jeśli wykonuje się kilka wykresów w tym samym układzie osi współrzędnych, to kolor kolejnego wykresu jest zgodny z domyślną kolejnością kolorów mapy barw *parula*: niebieski, czerwony, żółty, fioletowy itd. Funkcja `plotyy` tworzy dwa wykresy i każdy z nich ma własną pionową oś współrzędnych, po lewej lub po prawej stronie wykresu (patrz przykład 4.1.4-2).

Przykłady zastosowań grafiki zawierają pliki z folderu *matlab\toolbox\matlab\demos*. Opis zawartości tych plików uzyskuje się za pomocą polecenia `doc demos`. Zaawansowane funkcje akwizycji i przetwarzania obrazów są oferowane w *Image Processing Toolbox*.

4.1. Wykresy plot, fplot i inne

Użytkownik MATLAB-a może wykorzystać wiele funkcji graficznych do prezentacji biznesowych, do wizualizacji i animacji wyników obliczeń oraz symulacji.

4.1.1. Funkcja plot

Wykres dwuwymiarowy z liniową skalą na obu osiach można wykonać za pomocą funkcji `plot` lub opisanej w podrozdziałach 2.2.8 i 4.1.2 funkcji `fplot`. Wywołanie funkcji `plot` powoduje otwarcie okna graficznego, w którym pojawia się wykres. Funkcja `plot` może być wywoływana z różną liczbą argumentów wejściowych. Warianty wywołania tej funkcji są następujące:

```
plot(y)     % oś x jest opisywana liczbami 1, 2, 3, ...
plot(x,y)     % pojedynczy wykres y=f(x) - jeśli y jest wektorem; n wykresów - jeśli y jest macierzą
              % n-kolumnową,
plot(x,y,'typ_linii')   % zmiana typu i koloru linii
plot(x1,y1,'typ_linii1',x2,y2,'typ_linii2',...)     % kilka wykresów
plot(x1,y1,x2,y2,...)     % kilka wykresów
plot(x,y,'PropertyName',PropertyValue)     % np. plot(x,y,'LineWidth',3)
plot(axisHandle,...)      % np. plot(gca,x1,y1,'typ_linii1',x2,y2,'typ_linii2',...) - tworzy wykres
                          % w układzie współrzędnych wskazanym przez uchwyt axisHandle
```

gdzie:

`x`, `y` — wektory *n*-elementowe lub wektor i macierz o wymiarach *m* x *n*. W razie niezgodności wymiarów MATLAB dokonuje transpozycji wektora lub macierzy `y` i ponawia próbę wykonania wykresu.

typ_linii — łańcuch określający kolor i (lub) rodzaj linii wykresu — zgodnie z symbolami podawanymi przez `doc plot`. Na przykład łańcuch `'r:'` oznacza, że wykres będzie wykonany linią kropkową (:) w kolorze czerwonym (ang. *red*). Parametr `typ_linii` można pominąć. Wtedy kolory kolejnych wykresów są określane przez pole `DefaultColorOrder`.

`x2,y2,'typ_linii2', ...` — parametry kolejnych wykresów.

'PropertyName',PropertyValue — para parametrów wykresu: *nazwa* i *wartość* dla obiektu *Handle Graphics*, np. 'LineWidth',3.

axisHandle — uchwyt lub funkcja gca (ang. *get current axis*) dla aktualnych osi.

Przykład 4.1.1-1

Zachęca się Czytelnika do wykonania kilku wykresów.

```
x=0:pi/12:3*pi;        % przygotowanie wektora x
y=sin(x);              % tablicowanie funkcji y(x)
plot(y)                % numery elementów na osi x
plot(x,y)              % poprawny opis osi x
h=plot(x,y)            % do zmiennej h wpisano uchwyt (ang. handle) do wykresu
plot(x,y,x,cos(x))     % dwa wykresy, wektor x podany dwukrotnie
plot(x,y,'r:',x,cos(x),'g.-')    % podano kolor i rodzaj linii wykresu
plot(x,[y; cos(x)])    % dwa wykresy, dane w kolumnach macierzy [ ]
yy=[1 3 5;2 3 4],plot(yy) % macierz 3-kolumnowa, trzy wykresy: 1:2, 3:3, 5:4
```

Funkcja plot może być użyta do rysowania grafów (więcej podaje doc graph/plot). Znanym przykładem jest graf struktury węgla C-60, którą można wyrysować poleceniami:

```
G = graph(bucky)
plot(G)
```

4.1.1.2. Osie logarytmiczne — semilogx, semilogy lub loglog

Funkcja plot wykonuje wykres z liniowymi osiami *x*, *y*. Jedną lub dwie osie logarytmiczne można uzyskać, stosując odpowiednio semilogx, semilogy lub loglog.

4.1.2. Fplot, ezplot i inne funkcje graficzne

Niektóre funkcje z grupy *ez* (w dokumentacji: *easy to use*), np. ezplot, ezplot3, ezcontour, ezsurf, ezmesc, nie powinny być używane, gdyż poczynając od wersji MATLAB-a 2016a, ich funkcjonalność jest dostępna odpowiednio w fplot, fplot3, fcontour, fsurf, fmesc. Nie dotyczy to funkcji ezpolar, ezcontour, ezsurfc, ezmeshc, ezcontour, ezcontourf oraz funkcji ezsurf w wersji zaimplementowanej w *Symbolic Math Toolbox*. Służą one do wykonania dwu- i trójwymiarowych wykresów dla wyrażeń matematycznych, w tym wyrażeń symbolicznych z *Symbolic Math Toolbox*. Funkcje fplot i fplot3 były już opisywane w podrozdziale 2.2.8.

Używając opisanych wyżej funkcji, wystarczy podać ich nazwę i ewentualnie przedział dla argumentu x. Dla przykładu poniżej podano opis funkcji fplot. Opisy pozostałych funkcji są dostępne poprzez doc *nazwa_funkcji*. Funkcja fplot jest podobna do fplot, ale ma większe możliwości. Można ją wywołać z jednym parametrem lub z większą liczbą parametrów w następującej postaci:

- fplot(*f, [xmin, xmax]*) — wykonuje wykres funkcji $y = f(x)$ w przedziale $xmin<x<xmax$, Jeśli przedziału nie podano, wykres jest wykonywany dla $-2\pi<x<2\pi$.
- fplot(*f, [xmin, xmax, ymin, ymax]*) — wykonuje wykres funkcji $z = f(x,y)$ w przedziale $xmin<x<xmax$, $ymin<y<ymax$. Jeśli przedziału nie podano, wykres jest wykonywany dla $-2\pi<x<2\pi$, $-2\pi<y<2\pi$.

- `fplot(funx,funy, [tmin, tmax])` — wykonanie wykresu krzywej parametrycznej $funx = x(t), funy = y(t),\ t \in [tmin, tmax]$ lub dla $t \in [-2*\pi, 2*\pi]$, jeśli $[tmin, tmax]$ nie podano.
- `fplot(…,figure_handle)` — wykres jest wykonywany we wskazanym oknie.
- `fplot(axes_handle,…)` — wykres jest wykonywany we wskazanym układzie współrzędnych.

Więcej wariantów i dokładne opisy podaje `doc nazwa funkcji`.

4.1.3. Kolory, rodzaje linii i opisywanie wykresów

Kolory kolejnych wykresów są pobierane automatycznie z mapy barw *parula* utworzonej pod kątem łatwego odróżnienia kolejnych wykresów. Parula to niewielki, kolorowy ptak wędrowny występujący w Ameryce Północnej. Można też użyć na wykresie dowolnych innych barw. Standardowe kolory RGB i CMYK, ich symbole oraz symbole rodzajów linii podano w tabeli 4.1.3-1.

Tabela 4.1.3-1. *Zestawienie kolorów, rodzajów linii i znaczników wykresu*

Symbol	Kolor i rodzaj linii	Symbol	Oznaczenia punktów
r	czerwony (*red*)	s	kwadrat
g	zielony (*green*)	d	diament ◊
b	niebieski (*blue*)	v	trójkąt ∇
c	błękit (*cyan*)	^	trójkąt Δ
m	purpura (*magenta*)	>	trójkąt prawy
y	żółty (*yellow*)	<	trójkąt lewy
k	czarny (*blacK*)	+	plus +
w	biały (*white*)	.	punkt ●
-	ciągła	o	okrąg ○
:	kropkowa	h	gwiazda sześciokątna *
-.	kreskowo-kropkowa	*	gwiazdka *
--	kreskowa	p	gwiazdka pięciokątna l*
		x	znak ×

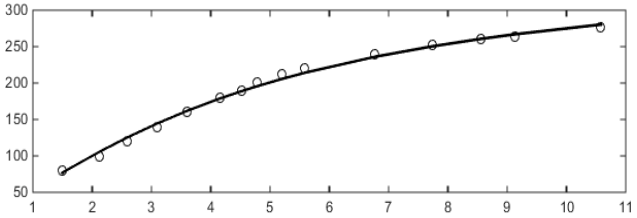
Opis możliwości zmian rodzaju i kolorów linii podano także przy omawianiu zasad grafiki zorientowanej obiektowo *Handle Graphics* (podrozdział 7.1.3).

Przykład 4.1.3-1

Punkty pomiarowe i aproksymacja biegu jałowego silnika — wartości pomiarowe dla osi odciętych (oś *x*) zawiera wektor `Im`, a dla osi rzędnych (oś *y*) wektor `Ugen`. W pracy [Mrozek, 1984] pokazano, że takie pomiary dobrze przybliża funkcja w postaci: `Ua=239.984 * atan(0.222*Im);`. Wykres *Ugen = f(Im)* wykonano następująco:

```
Im=[1.5 2.13 2.6 3.09 3.6 4.15 4.52 4.78 5.2 5.59 6.77 7.74 8.55 9.13 10.58]
Ugen=[80 100 120 140 160 180 190 200 212 220 240 252  260 264 276]
Uapr=239.984 * atan(0.222*Im);
h=plot(Im, Ugen,'ok', Im,Uapr,'k-')     % do zmiennej h przypisano uchwyt handle
set(h(2),'LineWidth',2)     % uchwyt h(2) wskazuje na drugi wykres:, Im,Uapr,'k-'
% dla drugiego wykresu ustawiono grubość linii =2
```

Na powstałym w ten sposób rysunku 4.1.3-1 punkty pomiarowe zaznaczono małymi okręgami (parametr `'ok'`). Przebieg funkcji *arcus tangens* narysowano linią pogrubioną.



Rysunek 4.1.3-1. *Charakterystyka biegu jałowego silnika i jej aproksymacja*

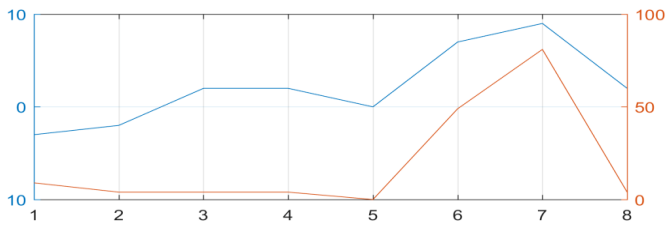
4.1.4. Dwie osie pionowe y, jedna oś pozioma x

Dwie liniowe osie *y*, odpowiednio po lewej i po prawej stronie wykresu, są rysowane po wywołaniu `plotyy` lub `yyaxis`.

Przykład 4.1.4-1

Funkcja `plotyy` tworzy dwa wykresy i oddzielne osie pionowe (lewa i prawa) dla każdego z wykresów (rysunek 4.1.4-1).

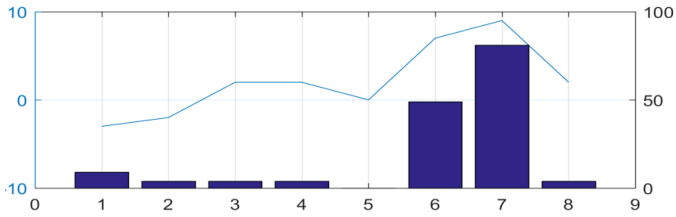
```
x= 1:8
y1=[-3 -2  2  2 0  7 9 2]        % funkcja y1
y2=y1.*y1                        % funkcja y2
plotyy(x,y1,'-',x,y2)            % dwa wykresy, oddzielne osie: y1, y2
grid on                          % siatka
```



Rysunek 4.1.4-1. *Wykres plotyy — dwie osie y i dwa wykresy*

Użycie uchwytów np. `@plot,@bar` jako dodatkowych parametrów funkcji plotyy daje nowe możliwości pokazane na rysunku 4.1.4-2 — druga funkcja została wyrysowana w postaci wykresu słupkowego. Znak `@` przed nazwą funkcji oznacza, że jest to uchwyt funkcji (ang. *function handle*). Omówiono je w podrozdziale 3.2.2.2.

```
% dodatkowe dwie linie kodu do przykładu 4.1.4-1
plotyy(x,y1,x,y2,@plot,@bar)     % dwie osie, różne funkcje graficzne
grid on    % siatka
```



Rysunek 4.1.4-2. *Wykres plotyy — dwie osie y i dwie funkcje graficzne: plot i bar*

Opisana powyżej funkcja `plotyy` tworzy dwa wykresy i oddzielne osie pionowe dla każdego z tych wykresów. Natomiast funkcja `yyaxis` (dostępna od wersji MATLAB-a 2016a) umożliwia łatwe umieszczenie większej liczby wykresów pomiędzy dwoma osiami pionowymi. Poniżej podano przykład kodu z użyciem `yyaxis`.

Przykład 4.1.4-2

```
clear, close
x=linspace(0,pi*4);
y1=sin(x); y2=cos(x); y3=sin(4*x)./x;

yyaxis left        % lewa, niebieska oś y
plot(x,y1,x,y2)    % wykresy niebieskie

yyaxis right       % prawa, czerwona oś y
plot(x,y3,'-.')    % wykres czerwony
```

4.1.5. Podział okna i modyfikowanie rysunków

Poniżej omówiono przykład podziału okna graficznego na obszary, w których wykonano niezależne wykresy. Do nadania wartości wektorowi `x` użyto notacji dwukropkowej. Oddzielone dwukropkiem wyrażenia to kolejno: wartość początkowa, krok równy $\pi/15$ i dopuszczalna wartość maksymalna.

Polecenie `subplot(2,1,1)` dzieli okno graficzne na dwa obszary w pionie. Natomiast `subplot(2,3,1)` realizuje podział okna na dwa obszary w pionie oraz na trzy obszary w poziomie, razem na sześć obszarów. Trzeci parametr (tu: równy 1) wskazuje, że aktualnie wykonywany wykres będzie wstawiony do pierwszego obszaru. Rezultat działania funkcji `subplot(2,1,1)` i `subplot(2,1,2)` pokazano na rysunku 4.1.5-1. Zawartość pliku, który wykonuje te wykresy, podano poniżej.

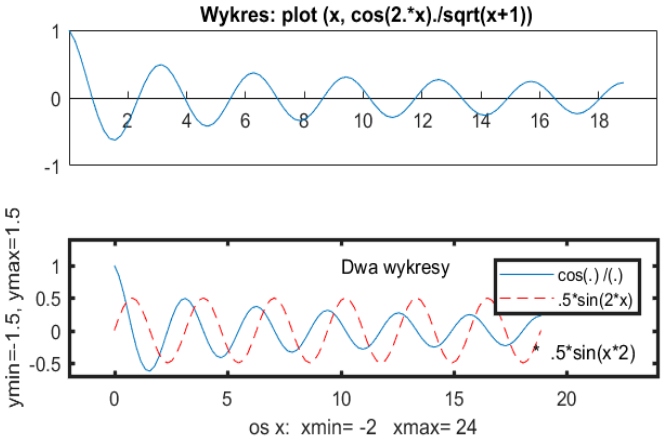
Przykład 4.1.5-1

Podział okna graficznego podzielonego na obszary z użyciem `subplot`.

```
x=0:pi/16:6*pi;
hh1 = subplot(2,1,1)                % hh1 to uchwyt (handle) do nowego okna
plot(hh1,x,cos(2*x)./sqrt(x+1));    % pierwszy parametr to hh1, uchwyt
set(gca,'XAxisLocation','origin')   % os x przechodzi przez 0 (MATLAB 2015b)
title(' Wykres: plot (x, cos(2.*x)./sqrt(x+1))')
hh2 = subplot(2,1,2)                % hh2 to uchwyt (handle) do nowego okna
plot(hh2,x,cos(2*x)./sqrt(x+1),  x,0.5*sin(2*x),'r--');    % dwa wykresy
xmin=-2; xmax=24; axis([xmin,xmax,-0.7,1.4])    % [xmin,xmax ymin,ymax]
text(10,1,'Dwa wykresy'), legend('cos(.) /(.)','.5*sin(2*x)')
x1=18.5;
text(x1,0.5*sin(x1*2),'*  .5*sin(x*2)')    % (x,y,'tekst')
```

```
ylabel('ymin=-1.5, ymax=1.5')
xlabel(['os x:  xmin= ', num2str(xmin),'   xmax= ',num2str(xmax)])
set (hh2,'LineWidth',2)    % zmiana grubości osi i ramki drugiego okna
```

Do wykonania wykresu pokazanego w górnej części rysunku 4.1.5-1 użyto funkcji `plot` z uchwytem `hh1` jako pierwszego parametru. Uchwyt `hh2` określa położenie osi rysunku dolnego.



Rysunek 4.1.5-1. *Dwa układy współrzędnych i trzy funkcje*

Funkcja `title` służy do umieszczenia tytułu nad wykresem. Funkcja `axis` zmienia skalowanie osi wykresów, wydłużając oś *x* do 30 jednostek. Funkcje `xlabel` i `ylabel` wykonują opisy osi `x` i `y` aktualnego wykresu. Funkcja `text` umieszcza tekst na aktualnym rysunku, w miejscu określonym przez wartości jej pierwszych dwóch parametrów `(x,y)`. Jej odmianą jest funkcja `gtext('napis')`, umieszczająca napis w miejscu wskazanym myszą. Podstawienia `hh1=` oraz `hh2=` mają na celu przechowanie uchwytu do tworzonego obiektu graficznego.

Użycie funkcji `axes` z parametrem `position` daje możliwość dowolnego rozmieszczania osi i wykresów wewnątrz okna graficznego. Parametry liczbowe `'position',[x1,y1,x,y]` to podane w skali 0..1 odpowiednio: współrzędne `(x1,y1)` dla lewego dolnego narożnika wykresu oraz długość obu osi: `x`, `y` (rysunek 4.1.5-1).

4.1.6. Funkcje do opisywania i modyfikowania wykresów

Funkcje do opisywania wykresów:

- `grid` — nałożenie siatki współrzędnych na wykres,
- `gtext` — umieszczenie napisu w miejscu wskazanym myszą,
- `text` — umieszczenie napisu w punkcie (*x*, *y*),
- `legend` — legenda do wykresu,
- `title` — tekst opisujący wykres (tytuł),
- `xlabel` — opis osi *x*,
- `ylabel` — opis osi *y*.

W opisach można używać wybranych poleceń edytora LaTeX, co daje możliwość wprowadzania do tych opisów wzorów matematycznych i liter alfabetu greckiego (przykład takiego opisu znajduje się na rysunku 7.4.5-1; zmienna β).

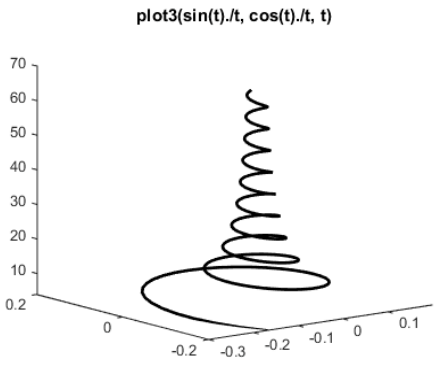
Funkcje stosowane do modyfikowania rysunków:

- `axis` — określa zakres wartości liczbowych na osiach,
- `axes` — określa położenie i wygląd osi,
- `zoom` — powiększanie lub zmniejszanie rysunku dwuwymiarowego,
- `box` — `on/off` — tworzenie tła obszaru rysunku,
- `hold` — umożliwia nakładanie kolejnych wykresów na rysunku,
- `ishold` — 1 lub 0, odpowiednio dla stanu on/off funkcji `hold`,
- `get(gca,…), set(gca, …)`— odczytuje lub zmienia atrybuty osi wykresu,
- `subplot` — podział okna na obszary dla kolejnych rysunków.

4.2. Wykresy trójwymiarowe

Ikony używane do wykonywania różnego rodzaju wykresów i rysunków trójwymiarowych pokazano na rysunku 4-2 na początku tego rozdziału. Standardowe rysunki są zazwyczaj kolorowane lub cieniowane.

Na rysunku 4.2-1 pokazano trójwymiarową spiralę o malejącej średnicy, którą narysowano z zastosowaniem funkcji `plot3`. Argumentami tej funkcji są sinus i cosinus o malejących amplitudach oraz **zmienna** `t`. Funkcje `sin(t)` i `cos(t)` powodują obrót, a zmienna `t` unosi w górę punkt rysujący spiralę. Do obliczania funkcji użyto dzielenia tablicowego z kropką, np. `sin(t)./t`. Funkcje są zależne od parametru `t`.



Rysunek 4.2-1. *Spirala trójwymiarowa — funkcja plot3(x,y,z)*

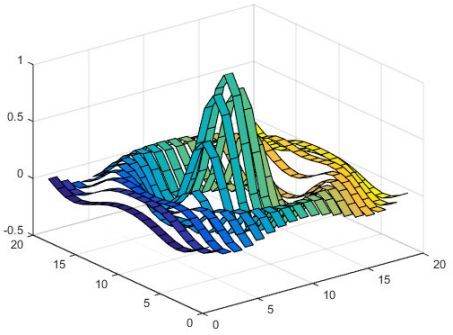
Przykład 4.2-1

Wykres funkcji dwóch zmiennych w przestrzeni trójwymiarowej (rysunek 4.2-1).

```
t=[pi:pi/18:20*pi];                 % wektor z wartościami parametru t
plot3(sin(t)./t, cos(t)./t, t)      % oblicza wartości x, y, z
axis([-0.3 0.2  -0.2 0.2   4 70])   % ustawia zakresy dla osi x, y, z
title('plot3(sin(t)./t, cos(t)./t, t)')
```

Funkcja `ribbon` buduje przestrzenny obraz z kolorowych tasiemek, co także pokazano na rysunku 4.2-2. Rysunek ten utworzono następującym zestawem poleceń:

Rysunek 4.2-2. *Wykres trójwymiarowy wykonany z użyciem ribbon(z)*



Przykład 4.2-2

```
% Funkcja ribbon - kolorowe tasiemki
[x,y]=meshgrid(-9:1:9);
r=sqrt(x.^2+y.^2);
z=sin(r+eps)./(r+eps);
ribbon(z)
```

Pozostawia się Czytelnikowi samodzielne wykonanie rysunków do przykładu 4.2-3.

Przykład 4.2-3

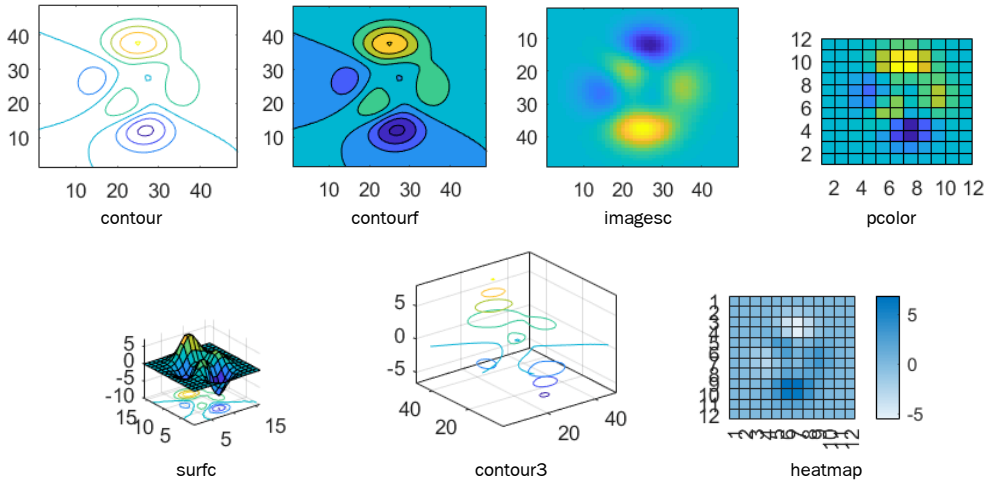
```
[x,y]= meshgrid(0:0.1:1, 0:0.1:1);
mesh((1-x).*(1-y))

t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1); mesh(X), title('mesh(X)')
subplot(2,2,2); mesh(Y), title('mesh(Y)')
subplot(2,2,3); mesh(Z), title('mesh(Z)')
subplot(2,2,4); mesh(X,Y,Z), title('mesh(X,Y,Z)')
```

4.2.1. Mapy dla funkcji trójwymiarowych

Na rysunku 4.2.1-1 pokazano przykłady map dla funkcji trójwymiarowej. Dane dla map obliczono z użyciem funkcji `peaks`.

```
z12=peaks(12)
z49=peaks
```

W oknie *Workspace* panelu MATLAB-a wybrano zakładkę *PLOTS* i zaznaczono zmienną `z49`. Wykres był wykonywany automatycznie po kliknięciu ikony z wybranym wykresem. Jeśli widoczna na niektórych wykresach mapa była zbyt gęsta, wykonywano ją powtórnie dla danych `z12`.



Rysunek 4.2.1-1. *Mapy dla funkcji trójwymiarowych wykonano dla wartości obliczonych przez z=peaks(n) dla n=12 lub n=49*

4.3. Interaktywne tworzenie i edycja rysunków

MATLAB udostępnia środowisko *Plot Tools* przeznaczone do interaktywnego tworzenia rysunków, a przede wszystkim do zmiany własności elementów składowych gotowego rysunku. Wygodnym sposobem zmiany parametrów rysunku jest zmiana parametrów obiektów graficznych. Opisane narzędzia interaktywne pozwalają:

- tworzyć dowolne wykresy dwu- i trójwymiarowe,
- eksperymentować z danymi pobieranymi bezpośrednio z okna *Workspace*,
- dowolnie rozmieszczać i przesuwać wykresy w oknie graficznym,
- dowolnie umieszczać elementy objaśniające wykresy, jak: teksty, linie, strzałki, chmurki itp.,
- określać i zmieniać własności (ang. *properties*) wszystkich elementów gotowego rysunku, w tym grubość, kolor i typ wykresu, jego osi i jego opisów,
- zapisać gotowy rysunek w postaci pliku graficznego (w tym *.jpg*, *.tiff*, *.png*, a nawet *.eps*) oraz w postaci umożliwiającej późniejszą edycję (*.fig*) i jego powtórne i w pełni automatyczne wykonanie dla kolejnych zestawów danych.

W pierwszej kolejności zostanie omówione przygotowanie danych i wykonanie wykresu, a następnie edycja rysunków z użyciem ikon okna *Figure*. Główne narzędzia środowiska *Plot Tools* to: *Figure Palette*, *Plot Browser* i *Property Editor*; są omówione w podrozdziale 4.3.4.

4.3.1. Przykład — przygotowanie danych do wykresu

Przygotowano krótki plik skryptowy *testdata.m*, który definiuje wektor `t` (zmienna niezależna) oraz oblicza wartości zmiennych potrzebnych do wykonania różnych wykresów.

Przykład 4.3-1

Przygotowanie danych do wykresu.

```
%% plik TestData.m
clear all    % usunięcie zmiennych i funkcji z pamięci roboczej
t=0:0.02:20;    % zmienna niezależna czas
%% poniżej obliczono dane dla kilku wykresów
s0=sin(t);  s1=sin(t+1);  s2=sin(t+2);
%% dzielenie tablicowe „z kropką"
sx=sin(t)./t;
c0=3*cos(t); c1=3*cos(t+1); c2=3*cos(t+2);
cp=cos(t) -cos(3.*t)/3 +cos(5.*t)/5;
```

Poniższy skrypt należy wpisać do edytora i zapisać w aktualnym folderze MATLAB-a jako *TestData.m*. Efektem działania tego skryptu jest utworzenie wektorów {`t`, `c0`, `c1`, `c2`, `cp`, `s1`, `s2`, `sx`}. W oknie *Workspace* widać nazwę każdego wektora, jego wymiar (tutaj 1×1001) oraz wartości największego i najmniejszego elementu (na przykład –3 +3). Jedynie dla wektora `sx`, z uwagi na dzielenie przez zero podczas obliczania pierwszego elementu tego wektora, wartość minimalną i maksymalną oznaczono jako `NaN`, czyli *Not a Number*. Pomimo to użycie wektora `sx` do wykonania wykresu jest możliwe i daje poprawne rezultaty.

4.3.2. Wykonanie wykresu z użyciem myszy

Aby wykonać wykres, wystarczy w oknie *Workspace* zaznaczyć myszą wektor z danymi (na przykład `sx`), a następnie w zakładce *PLOTS* kliknąć ikonę .

Wykres będzie poprawny, ale na osi *x* pojawią się numery elementów wektora `sx`, a nie wartości zmiennej `t`. Aby to poprawić, należy (jako pierwszy) wybrać wektor zmiennej niezależnej `t`, a następnie `cp`. Jeśli zajdzie taka potrzeba, to można przestawić zmienne, używając ikony lub x↔y. Kliknięcie czarnego trójkąta pokazuje dodatkowe ikony lub opcje menu.

4.3.3. Interaktywna edycja wykresów

Edycję okna graficznego można wykonywać z użyciem menu i ikon dostępnych w pasku narzędziowym tego okna (rysunek 4.3.3-1), a także z użyciem narzędzi *Plot Tools* opisanych w podrozdziale 4.3.4. Innym podejściem jest grafika uchwytów opisana w podrozdziale 7.1.

Rysunek 4.3.3-1. *Menu i ikony okna graficznego Figure*

Dodatkowe dwa paski narzędziowe, dostępne jako opcje menu *View*, pokazano na rysunku 4.3.3-2. Pasek *Plot Edit Toolbar* zawiera ikony narzędzi do opisywania rysunków. Umożliwia rysowanie linii ze strzałką i bez strzałki, chmurek oraz wprowadzenie komentarza w miejscu wskazanym przez kursor myszy. Aby obramowanie tekstu stało się niewidoczne, należy nadać mu kolor tła (zazwyczaj biały) z użyciem opcji *Edge Color*, dostępnej po kliknięciu utworzonego tekstu prawym przyciskiem myszy.

Rysunek 4.3.3-2. *Dodatkowe paski narzędziowe okna graficznego: Plot Edit Toolbar oraz Camera Toolbar*

Kolejne ikony  powodują odpowiednio powiększanie zaznaczonego prostokątnego obszaru, pomniejszanie rysunku, jego przesuwanie, obracanie rysunku w dwóch lub w trzech wymiarach oraz podanie wartości zmiennych we wskazanym punkcie wykresu.

Następna ikona to kolorowy pędzel  do zaznaczania wybranych grup danych (*Data brushing*) na przykład o wartościach poza poprawnym przedziałem wartości zmiennej. Ikona *Data link*  aktywuje odświeżanie wykresu przy każdorazowej zmianie wartości zaznaczonych danych. Kolejne dwie ikony  służą do narysowania słupka ze skalą barw (ang. *colorbar*) oraz legendy do wykresów.

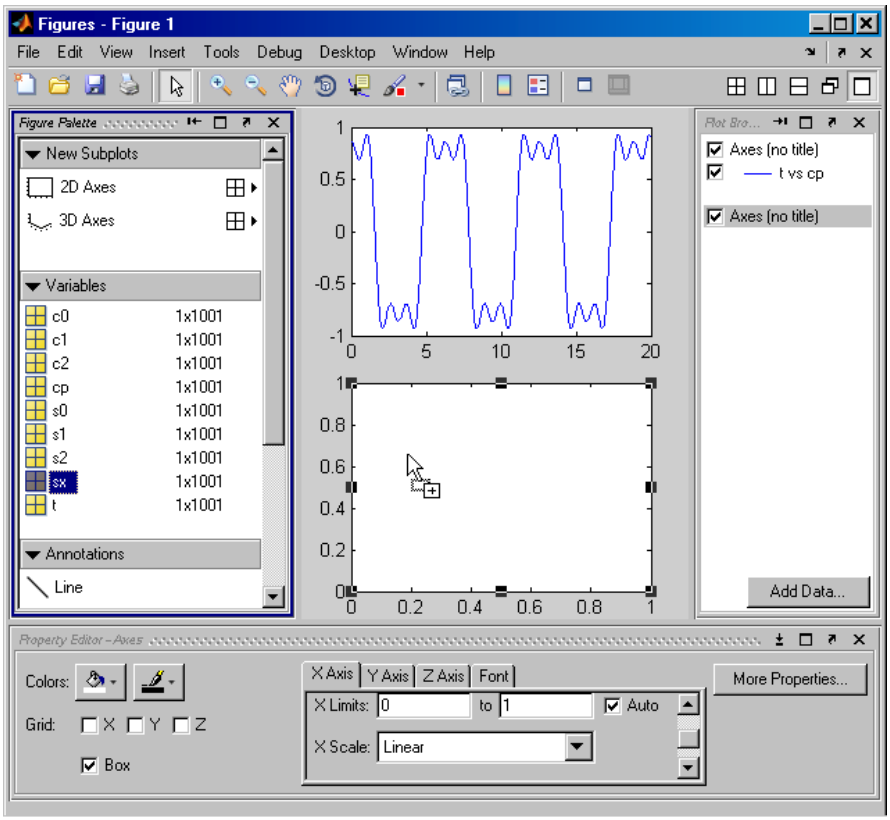
Zmiana własności elementów rysunku, takich jak położenie i skalowanie, opisywanie osi, grubość, kolor i typ linii wykresu, wymaga kliknięcia ikony w kształcie ukośnej strzałki  , co spowoduje przejście w tryb edycji. Wyboru elementu do edycji (oś, wykres, opis itp.) dokonuje się w oknie graficznym prawym przyciskiem myszy. Powoduje to zaznaczenie edytowanego elementu i otwarcie menu, którego opcje umożliwiają zmianę własności (wartości parametrów) tego elementu (rysunek 4.3.4-1).

Dodatkowy pasek narzędziowy *Camera Toolbar* (rysunek 4.3.3-2) jest przydatny do modyfikacji rysunków zawierających bryły trójwymiarowe. Można go uaktywnić, wybierając z menu okna graficznego opcję *View/Camera_Toolbar*. Zawiera on między innymi narzędzia do modyfikowania oświetlenia i do określenia kierunku oraz odległości, z jakiej trójwymiarowa bryła będzie widoczna. Przykładową bryłę na potrzeby testowania tych narzędzi można utworzyć za pomocą polecenia `peaks`.

4.3.4. Narzędzia interaktywne — Plot Tools

Używanie narzędzi *Plot Tools* jest bardzo łatwe, gdyż można je stosować bez korzystania z dokumentacji, a efekty pracy są natychmiast widoczne na ekranie. *Plot Tools* można otworzyć:

- za pomocą polecenia `plottols` (jeśli na ekranie nie ma okien *Figure* — otworzy się nowe okno zawierające tylko *Figure Palette* oraz puste miejsce na wykres);
- lub wybierając opcje (nazwę narzędzia) z menu *View* okna *Figure*;
- lub klikając ikonę  umieszczoną w prawej części okna *Figure*.



Rysunek 4.3.4-1. *Okno Figure z wykresem i narzędziami środowiska PlotTools — Figure Palette (po lewej), Plot Browser (po prawej) i Property Editor (z dołu). Pokazano wykres funkcji cp(t) z przykładu 4.3-1 oraz puste okienko na dodatkowy wykres*

Aby objaśnić działanie *Plot Tools*, w oknie *Command* wykonano przykład 4.3-1 i obliczone tam wartości zmiennych `t`, `cp` wykorzystano w poleceniach podanych poniżej.

```
% dane t, cp obliczono w przykładzie 4.3-1
plot(t,cp)     % wykonuje wykres cp(t)
plottools      % otwiera okna plottools
```

W efekcie pojawiło się okno *Figure* z wykresem oraz po jego lewej stronie *Figure Palette*. Następnie w menu *View* wybrano:

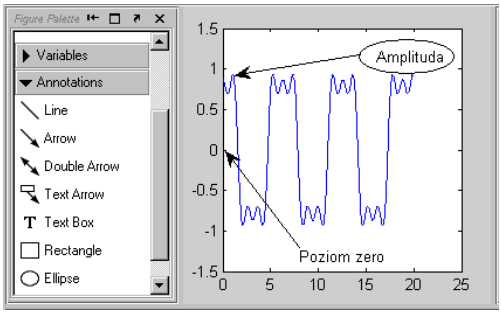
- *Plot Browser* — pojawi się po prawej stronie wykresu;
- *Property Editor* — pojawi się poniżej wykresu, jak na rysunku 4.3.4-1.

Otwarte narzędzia *Plot Tools* można zamknąć, klikając ikonę ☐ na belce narzędziowej, i powtórnie je otworzyć, klikając ikonę .

4.3.4.1. Okno Figure Palette

Aby w oknie graficznym umieścić dodatkowy wykres, wystarczy przesunąć do tego okna ikonę zmiennej zawierającej odpowiednie dane. Można ją pobrać z okna *Workspace* lub z okna *Figure Palette*. Tutaj zaznaczono zmienną `sx` i została ona przesunięta myszą do pustego okienka pod wykresem. Dodatkowe okienko otrzymano, klikając ikonę 2D Axes w oknie *Figure Palette*.

Dodatkowe wykresy można też umieścić w oknie, w którym już jest inny wykres. Aby wszystkie wykresy były czytelne, należy zadbać o odpowiednie zróżnicowanie typu i koloru linii, a także o takie dobranie danych, aby na każdej osi wartości minimalne i maksymalne dla każdego z wykresów były zbliżone. W dolnej części okna *Figure Palette* **po przesunięciu suwaka w dół** (rysunek 4.3.4.1-1) widać ikony linii, tekstów i chmurek, które można wykorzystać do opisania wykresów.



Rysunek 4.3.4.1-1. *Okno Figure Palette: ikony do opisywania rysunków*

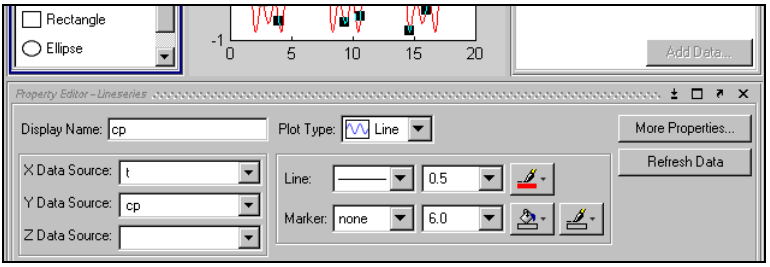
4.3.4.2. Okno Plot Browser

Plot Browser (prawe okienko na rysunku 4.3.4-1) podaje nazwy okienek graficznych i wykresów. Obok nazwy znajduje się niewielkie kwadratowe okienko wyboru. Jeśli ten kwadrat jest pusty, to dany wykres (lub całe okienko graficzne) jest niewidoczny. Zaznaczenie kwadratu zmienia się lewym przyciskiem myszy. Klikając natomiast prawym przyciskiem, można całkowicie usunąć wykres bądź osie współrzędnych wykresu.

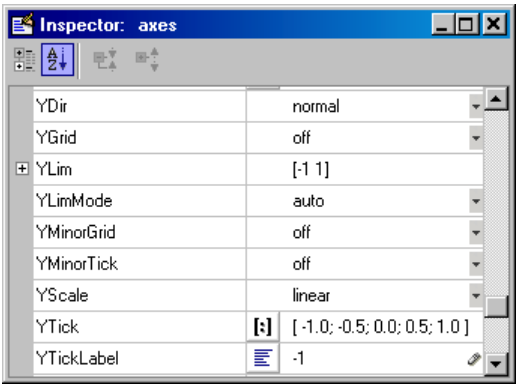
W dolnej części okna *Plot Browser* znajduje się przycisk *Add Data*, który uaktywnia się po wybraniu lub utworzeniu nowego okienka graficznego. Umożliwia on wybranie danych oraz wykonanie dowolnego typu wykresu (liniowy, schodkowy, biegunowy itp.).

4.3.4.3. Edycja osi, linii i tekstu w Property Editor i Inspector

Wygląd okna *Property Editor* zmienia się w zależności od edytowanych elementów (rysunki 4.3.4-1 – 4.3.4.3-1). Element przeznaczony do edycji (linia, opis, okno) można wybrać w oknie *Plot Browser* lub klikając go bezpośrednio w oknie graficznym. Dokonanie wyboru jest potwierdzone umieszczeniem niewielkich kwadratów na wybranym elemencie. W razie potrzeby można uaktywnić tryb graficzny, klikając ikonę. Jeśli atrybuty dostępne w aktualnym oknie *Property Editor* nie są wystarczające, należy kliknąć przycisk *More Properties*, co otworzy dodatkowe okno *Inspector* (rysunek 4.3.4.3-2).



Rysunek 4.3.4.3-1. *Okno Property Editor w trybie edycji linii wykresu*



Rysunek 4.3.4.3-2. *Przycisk More Properties otwiera dodatkowe okno Inspector do edycji parametrów graficznych*

Property Editor w trybie edycji linii wykresu umożliwia określenie jej grubości, rodzaju (ciągła, kropkowa itd.) oraz umożliwia wybór dodatkowych znaczników identyfikujących ten wykres. *Property Editor* w trybie edycji osi wykresu umożliwia nadanie lub zmianę nazwy osi *x*, *y*, *z*, określenie maksymalnej i minimalnej wartości skali na każdej osi oraz kroku dla skali i opisu na każdej osi. Ponadto można określić, czy skala ma być liniowa, czy logarytmiczna i czy będzie rysowana siatka współrzędnych.

4.3.5. Przygotowanie programu tworzącego grafikę

Przy większej liczbie wykresów opłaca się przygotować pliki zawierające polecenia potrzebne do wykonania grafiki. Warto też przygotować automatyczne opisywanie wykresów i ich osi w oparciu np. o datę, godzinę, użyte parametry i nazwę pliku z danymi do wykresu.

MATLAB może wygenerować plik, który można użyć do powtórnego utworzenia aktualnego okna graficznego. W tym celu należy w oknie *Figure* wybrać menu *File/Generate Code*. Efektem będzie plik, którego pierwsze linie mogą być podobne do poniższych.

```
function createfigure(X1, Y1)
% CREATEFIGURE(X1,Y1)
% X1: vector of x data
% Y1: vector of y data
% Auto-generated by MATLAB on 06-Sep-2013
```

Jak widać, utworzona została funkcja, która może wykonać wykres dla nowych kompletów danych (`X1,Y1`). Utworzenie biblioteki własnych plików pozwoli na powtórne wykonanie różnych rysunków.

4.3.6. Przenoszenie rysunków i zapisywanie do pliku

Rysunki zapisane do pliku można drukować lub wykorzystać w innym programie. Można je wykonać, korzystając z menu okna graficznego względnie polecenia `saveas` lub `print`.

4.3.6.1. Zapisywanie rysunków do pliku z użyciem myszy

Po użyciu opcji *File/Save* aktualny rysunek można zapisać do pliku w różnych formatach. Pliki *.fig* lub *.m* umożliwiają powtórne utworzenie i modyfikację rysunku w MATLAB-ie. Formaty graficzne, takie jak *.png*, *.jpg*, *.eps* i inne, umożliwiają użycie rysunku poza pakietem MATLAB.

Kompletne okno wraz z ramką można skopiować, wciskając jednocześnie *Alt+Print Screen* lub wykorzystując program *Windows\system32\SnippingTool*.

4.3.6.2. Zapisywanie rysunków do pliku z użyciem polecenia saveas lub print

Jeśli polecenie zapisania rysunku ma być umieszczone w pliku, to można użyć polecenia `saveas` lub `print`, na przykład:

- `saveas (h,'nazwa_pliku.rozszerzenie')`
- `print (nazwa_pliku, -opcje)`

gdzie:

h — *handle*, identyfikator rysunku; dla aktualnego rysunku można użyć `gcf` (ang. *get current figure*).

Rozszerzenie: *.ai*, *.bmp*, *.emf*, *.eps*, *.fig*, *.jpg*, *.m*, *.pcx*, *.png*, *.tif* (i wiele innych).

4.3.6.3. Zapisywanie rysunków do pliku w zadanej rozdzielczości

Polecenie `print` udostępnia więcej opcji niż menu i między innymi pozwala określić rozdzielczość zapisywanego rysunku. Przygotowując tę książkę, zapisywano rysunki w formacie *.png* z rozdzielczością 300 dpi poleceniem:

```
print('C:\Users\zm\Documents\MATLAB\examples\x300dpi','-dpng','-r300')
```

Pierwszym parametrem jest nazwa pliku `x300dpi` wraz z pełną ścieżką dostępu, następnie jest rozszerzenie `-dpng` określające format *png* pliku graficznego. Ostatni parametr `-r300` określa rozdzielczość zapisanego rysunku: 300 dpi.

4.3.6.4. Zapisywanie rysunków do pliku w formacie wektorowym

Polecenia `saveas` oraz `print` umożliwiają zapisanie grafiki w formacie wektorowym, np. *.eps*.

```
print('print2epsc','-depsc')
saveas(gcf,'saveas2epsc','epsc')
```

Dodatkowa litera *c* w *epsc* oznacza rysunek kolorowy.

Rozdział 5.

Typy danych i programy obiektowo zorientowane

W MATLAB-ie podstawowym typem danych jest tablica (ang. *array*). Jej elementami mogą być liczby rzeczywiste podwójnej precyzji `double`, liczby zespolone, znaki albo inne tablice. **Macierz** jest szczególnym przypadkiem tablicy — jest to *tablica dwuwymiarowa*, na której elementach wykonuje się operacje algebry liniowej.

Filozofię działania MATLAB-a oparto na operacjach wektorowo-macierzowych. Pojedyncza wartość liczbowa (całkowita, rzeczywista lub zespolona) jest traktowana jak macierz o wymiarach 1×1. Macierz pusta `[]` ma wymiar 0×0. Wektor to macierz, która składa się tylko z jednego wiersza lub tylko z jednej kolumny.

W MATLAB-ie nie ma potrzeby deklarowania zmiennych. Zmienne tworzy się poprzez przypisanie odpowiedniej wartości do zmiennej lub poprzez użycie funkcji tworzącej odpowiednią zmienną. Na przykład:

- funkcja `eye(3)` tworzy macierz pełną z trzema jedynkami na przekątnej,
- funkcja `sparse` tworzy macierze rzadkie,
- funkcja `datetime` tworzy zmienną odpowiednią dla zapisania daty i czasu.

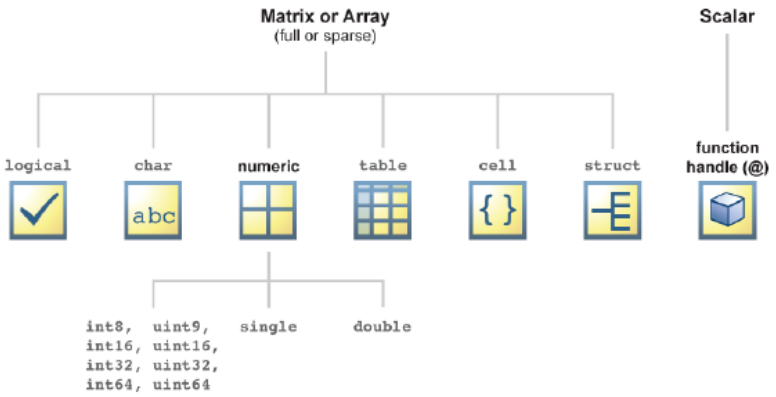
Nazwy i typy aktualnie używanych zmiennych są widoczne w panelu MATLAB-a, w oknie *Workspace* (rysunek 2.1.1-1). Polecenie `who` podaje *nazwy*, a `whos` podaje *nazwy* i *typy* aktualnie używanych zmiennych. Więcej informacji o tych poleceniach podaje `doc` *`nazwa_polecenia`*. Linki do informacji o typach danych podaje `doc datatypes`.

5.1. Fundamentalne typy danych

W MATLAB-ie zdefiniowano 16 fundamentalnych typów danych, które są zarazem klasami zdefiniowanymi w MATLAB-ie (rysunek 5.1-1):

- typ numeryczny obejmuje 10 fundamentalnych typów danych:
  - liczby zmiennoprzecinkowe 64- i 32-bitowe: `double`, `single`,
  - liczby całkowite (ang. *integer*) ze znakiem: `int8`, `int16`, `int32`, `int64`,
  - liczby całkowite bez znaku (ang. *unsigned*): `uint8`, `uint16`, `uint32`, `uint64`,

- znaki i łańcuchy: `char`,
- zmienne logiczne: `logical`,
- uchwyty funkcji: `function_handle`,
- prostokątne tabele na mieszane typy danych: `table` (ang. *mixed-type tabular data*),
- struktury: `struct`,
- tablice komórkowe: `cell`.



Rysunek 5.1-1. *Fundamentalne typy danych w MATLAB-ie. Multidimensional Array to tablica wielowymiarowa (reprinted with permission of The MathWorks)*

5.1.1. Numeryczne typy danych

Tworzenie zmiennych numerycznych, to jest liczb stało- lub zmiennoprzecinkowych wybranego typu, realizuje się z użyciem funkcji tworzących odpowiednie zmienne lub przez przypisanie wartości do istniejącej lub obliczanej zmiennej. Na przykład:

- przypisanie `a=[2 1 3]` tworzy wektor typu *double*,
- przypisanie `z=3+4j` tworzy zmienną typu *double complex*,
- `b=uint8(a)` konwertuje wektor `a` na 8-bitowy wektor `[2 1 3]` bez znaku.

Fundamentalne typy danych numerycznych pokazano na rysunku 5.1-1. Więcej informacji podaje `doc datatypes`.

Użytkownik może stosować typy danych, które zdefiniowano w MATLAB-ie i w jego rozszerzeniach. Typ macierzy lub tablicy zależy od typu jej elementów składowych. Macierze mogą być pełne, rzadkie i wielowymiarowe:

- ***Matrix***, **macierz**, **macierz pełna** — to tablica dwuwymiarowa o elementach rzeczywistych lub zespolonych. W macierzy pełnej MATLAB przechowuje wartości wszystkich jej elementów.
- ***Sparse***, **macierz rzadka** — w macierzy rzadkiej zapamiętywane są jedynie wartości elementów niezerowych oraz ich indeksy. Macierz rzadka może być utworzona przez funkcję `sparse` lub powstaje jako wynik operacji na macierzach rzadkich.
- ***Multidimensional matrix***, **tablica wielowymiarowa** — elementy składowe takiej tablicy mają *ten sam typ i wymiar*. Mogą to być liczby rzeczywiste lub zespolone, znaki, komórki, struktury i inne. Do wskazania elementu takiej tablicy używa się więcej niż dwóch indeksów.

5.1.2. Nienumeryczne typy danych

- Typ `char` obejmuje *znaki* i *łańcuchy ASCII* oraz *Unicode*. Prócz przypisania tekstu do zmiennej, np. `txt='ala ma kota'`, używana jest konwersja liczby na tekst, np. `c=num2str(12)` konwertuje wartość liczby *12* na łańcuch `'12'`, czyli na dwuelementowy wektor typu `char` o wymiarach 1×2.
- Typ `logical` — *zmienna logiczna*; ma wartości `prawda` i `fałsz` (ang. *true* i *false*), co jest zapisywane jako `1` lub `0`. Każda liczba rzeczywista *różna od zera* jest traktowana jako logiczne `true`.
- Typ `function_handle` — *uchwyt funkcji*, identyfikator; jest używany do wywołania funkcji. Aktualnie (od MATLAB-a 2014b), `function handle` jest *obiektem* przechowującym nazwę funkcji, pełną ścieżkę dostępu do niej i inne dane. *Function handle* umożliwia wywołanie funkcji w sytuacji, gdy wywołanie przez użycie nazwy tej funkcji nie jest skuteczne.
- Typ `cell array` — *tablica komórkowa*; przechowuje dane różnych typów w komórkach (ang. *cell*). W komórkach można umieszczać tablice o *dowolnych wymiarach* i *typach* (podrozdział 5.6).
- Typ `struct` — *struktura*; ma zdefiniowane pola (ang. *fields*) na różne typy danych. Struktury, podobnie jak tablice komórkowe, mogą przechowywać *inne tablice* o dowolnych wymiarach i elementach.

Użytkownik może zdefiniować własne typy danych (czyli *klasy*) za pomocą funkcji `classdef` (podrozdział 6.1).

5.1.3. Funkcje konwersji typów i klas

Polecenie w postaci `b=class_name(c)` jest konwersją, czyli przekształceniem jednego typu danych (lub obiektu) oznaczonego jako `c` w inny, należący do klasy `class_name` i oznaczony jako `b`. Typowym przykładem jest funkcja `char()`, która przekształca informację przekazaną w postaci parametru w łańcuch, czyli w zmienną należącą do klasy `char`, np.:

```
>> tekst=char([77,65,84,76,65,66])    % wektor z kodami ASCII słowa MATLAB
MATLAB
```

Innym przykładem jest funkcja `class`, która przekształca strukturę w obiekt.

5.2. Macierze pełne

Macierz jest to tablica dwuwymiarowa, na której elementach wykonuje się operacje algebry liniowej. Każda zmienna tworzona jako macierz jest macierzą pełną. Oznacza to, że w pamięci jest przechowywana wartość każdego jej elementu. W macierzy rzadkiej przechowuje się tylko wartości różne od zera.

5.2.1. Sposoby tworzenia macierzy

W MATLAB-ie macierze można definiować na kilka różnych sposobów:

Wprowadzanie elementów macierzy z klawiatury, jak pokazano w podrozdziale 2.3.1:

- Użycie notacji dwukropkowej (`:`) pokazano w podrozdziale 2.2.7.
- Tworzenie macierzy elementarnych z użyciem funkcji, np. `magic(5)`.
- Tworzenie macierzy specjalnych z użyciem funkcji: `hadamard`, `hankel`, `hilb`, `invhilb`, `peaks`, `rosser`, `toeplitz`, `vander` i `wilkinson` (patrz `doc elmat`).
- Użycie funkcji konwersji typów, np.: `char`, `int2str`, `mat2str`, `num2str`, `str2double`, `str2num`, `native2unicode`, `unicode2native`, `base2dec`, `bin2dec`, `dec2base`, `dec2bin`, `dec2hex`, `hex2dec`, `hex2num`, `num2hex`, `cell2mat`. Funkcje `cell2struct`, `cellstr`, `mat2cell`, `num2cell`, `struct2cell` tworzą struktury, łańcuchy i macierze komórkowe.
- Konstruowanie macierzy poprzez łączenie (sklejanie) ze sobą mniejszych macierzy (ang. *concatenation*). Operatorem wykonującym takie łączenie jest para nawiasów prostokątnych `[]`. Poniżej utworzono macierz `C` poprzez sklejenie trzech macierzy elementarnych.

```
C=[zeros(3),eye(3),ones(3)]
```

- Wczytanie poleceniem `load` macierzy zapisanej na dysku (plik binarny *.mat* lub plik zapisany w kodzie ASCII, jak pokazano w podrozdziale 2.3.3).
- Importowanie macierzy utworzonej w obcym formacie, np. *Excel*.

5.2.1.1. Macierze elementarne tworzone z użyciem funkcji

Macierze elementarne są używane do obliczeń, do rezerwowania pamięci dla innych macierzy i do przygotowywania wykresów. Macierze elementarne są tworzone przez funkcje `zeros`, `ones`, `eye`, `repmat`, `linspace`, `logspace`, `freqspace`, `meshgrid`, `accumarray`. Wykaz macierzy elementarnych (i wiele innych) wraz z krótkimi objaśnieniami uzyskuje się poleceniem `doc elmat`. Dokładniejsze informacje o poszczególnych funkcjach otrzymuje się poleceniem w postaci: `doc` *`nazwa_funkcji`*.

Wybrane funkcje do konstruowania macierzy opisane w `doc elmat`:

- `eye`, `ones`, `zeros` — odpowiednio: macierz jednostkowa (jedynki na przekątnej), macierz o wszystkich elementach równych `1` lub równych `0`;
- `magic` — tworzy kwadrat magiczny, przydatny do testowania wyrażeń matematycznych;
- `pascal` — tworzy macierz Pascala;
- `linspace`, `logspace`, `freqspace` — wektor o wartościach rozłożonych równomiernie lub nierównomiernie do opisywania osi na wykresach;
- `accumarray` — tablica z akumulacją wartości;
- `meshgrid` — tablica dla wykresów trójwymiarowych (siatkowych);
- `rand`, `randn` — macierz losowa o rozkładzie równomiernym i normalnym;
- `repmat` — tworzy dużą macierz z wielu kopii danej macierzy;
- `compan`, `gallery` — macierz stowarzyszona wielomianu, różne macierze testowe.

Przykład 5.2.1.1-1

Tworzenie macierzy z użyciem funkcji MATLAB-a. Cztery macierze elementarne o wymiarach 3×3 umieszczone wewnątrz nawiasów kwadratowych i przypisane do zmiennej `B`:

```
B=[zeros(3),eye(3),ones(3),rand(3)]
```

W rezultacie utworzono macierz `B` i przypisano do niej elementy zawarte pomiędzy nawiasami prostokątnymi. Rezultat: utworzono nową macierz o 3 wierszach i 12 kolumnach.

```
Columns 1 through 6
       0         0         0    1.0000         0         0
       0         0         0         0    1.0000         0
       0         0         0         0         0    1.0000

Columns 7 through 12
  1.0000    1.0000    1.0000    0.8147    0.9134    0.2785
  1.0000    1.0000    1.0000    0.9058    0.6324    0.5469
  1.0000    1.0000    1.0000    0.1270    0.0975    0.9575
```

5.2.1.2. Rezerwowanie pamięci z użyciem zeros, ones, eye

Funkcje `eye`, `ones`, `zeros` są używane do rezerwowania pamięci, co znacznie przyspiesza tworzenie dużych macierzy różnych typów. Aktualnie funkcje `eye`, `ones`, `zeros` tworzą nie tylko macierze `double`, ale też macierze pojedynczej precyzji `single` oraz wszystkie typy macierzy `integer`. Polecenie `A= ones([1280,1024,3],'uint8');` tworzy trójwymiarową macierz liczb *8-bitowych bez znaku* o wymiarach 1280×1024×3. Macierze `uint8` są stosowane do zapisu obrazów w grafice komputerowej.

5.2.1.3. Macierze specjalne tworzone z użyciem funkcji

Macierze specjalne są tworzone przez funkcje: `airy`, `besselj`, `bessely`, `besselh`, `besseli`, `besselk`, `beta`, `betainc`, `betaincinv`, `betaln`, `ellipj`, `ellipke`, `erf`, `erfc`, `erfcx`, `erfinv`, `expint`, `gamma`, `gammainc`, `gammaincinv`, `gammaln`, `psi`, `legendre`, które służą najczęściej do testowania funkcji i algorytmów. Poniżej podano ich krótkie opisy. Dokładniejsze informacje podaje `doc` *`nazwa_funkcji`*.

- Funkcja `airy` jest jednym z rozwiązań równania Schrödingera.
- Funkcja `besselj`, `bessely`, `besselh` — funkcja Bessela 1., 2., 3. rodzaju.
- Funkcja `besseli`, `besselk` — modyfikowana funkcja Bessela 1., 2. rodzaju.
- Funkcja `beta`, `betainc` — funkcja Beta, niepełna funkcja Beta.
- Funkcja `betaln` — logarytm funkcji Beta.
- Funkcja `ellipj`, `ellipke` — funkcja eliptyczna Jacobiego, całka eliptyczna.
- Funkcje `erf`, `erfc`, `erfcx`, `erfinv` — funkcje błędu Erf.
- Funkcja `expint` — całka z $\left(\exp(-t)/t\right)dt$ w granicach od x do ∞.
- Funkcja `gamma`, `gammainc` — funkcja Gamma, niepełna funkcja Gamma.
- Funkcja `gammaln` — logarytm z funkcji Gamma.
- Funkcja `legendre` — funkcja Legendre'a.

5.2.1.4. Macierze specjalizowane do testowania funkcji i algorytmów

Macierze generowane przez `compan`, `gallery`, `hadamard`, `hankel`, `hilb`, `invhilb`, `magic`, `pascal`, `peaks`, `rosser`, `toeplitz`, `vander`, `wilkinson` są używane do testowania funkcji i algorytmów. Dokładniejsze informacje podaje `doc nazwa_funkcji`.

5.2.2. Wybrane funkcje i operacje macierzowe

Funkcje arytmetyczne i trygonometryczne opisano w podrozdziale 2.2.2. Oprócz funkcji przeznaczonych do wykonywania operacji na macierzach MATLAB zawiera obszerny zestaw funkcji realizujących algorytmy numeryczne z zakresu algebry liniowej.

Wykaz najważniejszych funkcji macierzowych podano w kolejnych podrozdziałach. Dokładniejsze informacje o każdej funkcji podaje `doc nazwa_funkcji`.

5.2.2.1. Manipulowanie elementami macierzy

Funkcje przeznaczone do manipulowania elementami macierzy to: `cat`, `reshape`, `diag`, `blkdiag`, `tril`, `triu`, `fliplr`, `flipud`, `flipdim`, `rot90`, `find`, `end`, `sub2ind`, `ind2sub`, `bsxfun` i `:` (dwukropek). Dokładniejsze informacje o poszczególnych funkcjach podaje `doc nazwa_funkcji` i `doc elmat`. Najczęściej używane są:

- `cat` — sklejanie macierzy (ang. *concatenate*), ten sam rezultat daje użycie `[]`;
- `diag` — macierze diagonalne lub wektory elementów na przekątnej;
- `fliplr`, `flipud` — odbicie lustrzane macierzy w pionie lub poziomie;
- `reshape`, `rot90` — zmiana wymiaru macierzy, obrót macierzy o kąt prosty;
- `triu`, `tril` — macierz trójkątna z elementów ponad lub pod główną przekątną.

Przykład 5.2.2.1-1

Funkcji `tril` można użyć do operacji wyodrębnienia macierzy trójkątnej z elementów pod główną przekątną danej macierzy.

```
>> tril(eye(3,3) + 4*ones(3,3))
ans =
     5     0     0
     4     5     0
     4     4     5
```

Macierze można modyfikować za pomocą następujących funkcji:

- funkcje do operowania na tablicach wielowymiarowych to `ndgrid`, `permute`, `ipermute`, `shiftdim`, `circshift`, `squeeze`;
- funkcje podające wartość stałych i zmiennych specjalnych to `eps`, `realmax`, `realmin`, `intmax`, `intmin`, `flintmax`, `flintmin`, `pi`, `i`, `j`, `inf`, `nan`, `isnan`, `isfinite`;
- funkcje sprawdzające wartość zmiennych specjalnych `isnan`, `isfinite`;
- funkcje sprawdzające, czy zmienna jest skalarem, wektorem, kolumną, czy macierzą `isscalar`, `isvector`, `isrow`, `iscolumn`, `ismatrix`;
- analizę macierzową, poszukiwanie wartości własnych i osobliwych oraz rozwiązywanie równań liniowych można uzyskać poleceniem `doc matfun`.

Wybrane elementarne funkcje macierzowe i funkcje analizy macierzowej (pełna lista po wykonaniu `doc matfun`).

5.2.2.2. Funkcje analizy macierzowej

Funkcje `det`, `norm`, `normest`, `rank`, `det`, `trace`, `null`, `orth`, `rref`, `subspace` służą do analizy macierzowej, przykładowo:

- `det` — wyznacznik macierzy;
- `norm` — normy wektora i macierzy.

5.2.2.3. Funkcje macierzowe

Funkcje `expm`, `logm`, `sqrtm`, `funm` służą odpowiednio do obliczania

- `expm` — macierzowej funkcji wykładniczej;
- `logm` — macierzowej funkcji logarytmicznej;
- `sqrtm` — macierzowej funkcji pierwiastkowej;
- `funm` — zadanej funkcji macierzowej, np. *cos*;

Dokładniejsze informacje podaje `doc` *`nazwa_funkcji`*.

5.2.2.4. Badanie macierzy

Funkcje `size`, `length`, `ndims`, `numel` podają odpowiednio rozmiar macierzy, długość wektora (lub `max(size)` dla macierzy), liczba wymiarów i liczba elementów macierzy. Funkcja `disp` wyświetla elementy macierzy lub łańcucha, jest często używana do wyświetlania wyników obliczeń. Funkcja `isempty` sprawdza, czy macierz jest pusta, a `isequal` porównuje, czy macierze są równe. Funkcja `isequaln` działa jak `isequal`, ale wszystkie wartości `NaN` traktuje jako sobie równe. Dokładniejsze informacje o poszczególnych funkcjach podaje `help` *`nazwa_funkcji`*.

5.2.2.5. Obliczenia i badanie własności macierzy

Funkcję `norm` używa się do obliczania różnych norm macierzy. Funkcja `normest` jest używana do oszacowania normy kwadratowej $\|\cdot\|_2$ dla macierzy rzadkich oraz dla dużych macierzy pełnych. Funkcja `rank` oblicza rząd macierzy. Funkcje `det`, `trace` obliczają odpowiednio wyznacznik i ślad macierzy. Funkcje `null`, `orth` tworzą bazy ortonormalne. Funkcja `subspace(`*`A`*`,`*`B`*`)` oblicza kąt pomiędzy podprzestrzeniami zdefiniowanymi przez kolumny macierzy `A` i `B`. Wartość kąta równa zeru oznacza liniową zależność tych podprzestrzeni, a wartość $\pi/2$ oznacza ich ortogonalność. Funkcja `rref` używa eliminacji Gausa z częściowym wyborem elementu głównego (ang. *partial pivoting*). Nie zaleca się jej stosowania do rozwiązywania równań, gdyż użycie polecenia $x = A\backslash b$ automatycznie wybiera najlepszą metodę i oblicza rozwiązanie. Dokładniejsze informacje o omawianych funkcjach podaje `doc` *`nazwa_funkcji`*.

5.2.2.6. Wartości własne i osobliwe

Funkcje `eig`, `svd`, `gsvd`, `eigs`, `svds`, `poly`, `polyeig`, `condeig`, `hess`, `schur`, `qz`, `ordschur`, `ordqz`, `ordeig` obliczają wartości własne i osobliwe dla macierzy. Przykładowo `eig` oblicza wartości własne i wektory własne macierzy kwadratowej. Dokładniejsze informacje podaje `doc` *`nazwa_funkcji`* lub `doc backslash`.

5.2.2.7. Równania liniowe i rozkład (faktoryzacja) macierzy

Funkcje `\`, `linsolve inv`, `rcond`, `cond`, `condest`, `normest1`, `chol`, `ldl`, `lu`, `qr`, `pinv`, `lscov` służą do rozwiązywania równań liniowych i do faktoryzacji. Przykładowo:

- `\` — nazywana też *backslash* lub *mldivide*, rozwiązuje równania liniowe $Ax = b$, wybierając najlepszy algorytm dla danej macierzy. Więcej informacji podają podrozdział 8.1, `doc backslash` oraz `doc slash`.
- `linsolve` — rozwiązuje równania liniowe $Ax = b$, nie testuje macierzy A w celu wybrania optymalnego algorytmu. Jeśli jako parametr wywołania będzie podany rodzaj macierzy A (np. `opts=POSDEF`, czyli A jest dodatnio określona), to zostanie wybrany najlepszy algorytm dla takiej macierzy. Brak lub nietrafna informacja o macierzy A pogorszy dokładność i szybkość obliczeń.
- `inv` — macierz odwrotna. Nie zaleca się jej używania do rozwiązywania równań typu $Ax = b$, gdyż dokładniejsze rozwiązanie uzyskuje się prościej jako $x = A\backslash b$.
- `lu` — rozkład LU (ang. *lower*, *upper*) na macierze trójkątne, dolne i górne. Więcej informacji podaje podozdział 8.6.

Funkcje pomocnicze używane do faktoryzacji to `qrdelete`, `qrinsert`, `rsf2csf`, `cdf2rdf`, `balance`, `planerot`, `cholupdate`, `qrupdate`.

Dokładniejsze informacje o omawianych funkcjach podaje `doc nazwa_funkcji`.

5.3. Macierze rzadkie

Macierze o dużej liczbie zer warto definiować jako macierze rzadkie. Zapamiętywane są wtedy tylko elementy niezerowe macierzy, wraz z informacją o ich adresie (numer wiersza i kolumny). MATLAB wykonuje operacje na macierzach rzadkich (na przykład mnożenie) inteligentnie i szybko. Działań na elementach zerowych macierzy rzadkich nie wykonuje się, zakładając, że wynik jest zerem. Może to jednak prowadzić do błędów, bo np. `cos(0)=1`.

W MATLAB-ie macierze rzadkie nie powstają automatycznie. Są one tworzone w następujących przypadkach:

- użycie funkcji generującej macierz rzadką, np. `sparse`, `sprand`,
- przekształcenie macierzy pełnej na macierz rzadką (funkcje `sparse` lub `spconvert`),
- jako wynik operacji na macierzach rzadkich.

Typowymi przykładami macierzy rzadkich są: macierze pasmowe (w tym diagonalna), macierze blokowe oraz macierze trójkątne — dolna i górna (np. uzyskane jako wynik rozkładu LU). Pełny zestaw funkcji do generowania, przekształcania, wizualizacji oraz algebry liniowej macierzy rzadkich uzyskuje się za pomocą polecenia `doc sparfun`.

5.3.1. Tworzenie macierzy rzadkich

Przykład 5.3.1-1

Generowanie losowej macierzy rzadkiej:

```
>> S=sprand(6,5,0.1)   % macierz
S =
   (1,4)       0.035711678574190
   (6,4)       0.933993247757551
   (3,5)       0.849129305868777
>> A=full(S);
>> whos
  Name      Size            Bytes  Class     Attributes

  A         6x5               240  double
  S         6x5                96  double    sparse
```

Jak widać powyżej, zera są przechowywane tylko w macierzy pełnej `A`. Funkcja `sparse` może być wywoływana z następującymi parametrami:

- `S = sparse(i,j,s,m,n,nzmax)` — generuje macierz rzadką o wymiarach `m×n` i rezerwuje miejsce dla `nzmax` elementów niezerowych;
- `S = sparse(i,j,s,m,n)` — jak wyżej, ale nie przewidziano rezerwy dla elementów niezerowych;
- `S = sparse(i,j,s)` — wymiar macierzy określają zależności: `m = max(i)`, `n = max(j)`;
- `S = sparse(m,n)` lub `sparse([],[],[],m,n,0)` — generuje pustą macierz rzadką o wymiarach `n×m`,

gdzie:

`[i,j,s]` — kolumny, określające odpowiednio indeksy `i,j` oraz wartości elementów `s` macierzy rzadkiej; elementy `s` mogą być liczbami zespolonymi;

`m, n` — wymiar macierzy rzadkiej;

`nzmax` — maksymalna liczba elementów niezerowych macierzy `S`.

Różnica (`nzmax` — *liczba_elementów_niezerowych_macierzy*) jest rezerwą na dodatkowe elementy niezerowe.

5.3.2. Operacje na macierzach rzadkich

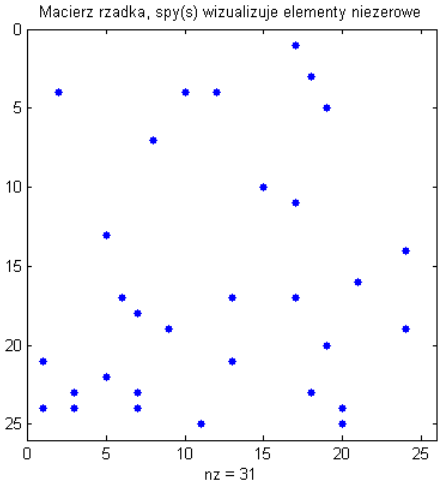
Funkcje i operatory MATLAB-a przeznaczone dla macierzy pełnych mogą być w większości zastosowane także dla macierzy rzadkich. Należy jednak zachować ostrożność, bo przykładowo może zostać pominięty wynik `cos(0)=1`.

Przykład 5.3.2-1

Wizualizacja elementów niezerowych macierzy rzadkiej (rysunek 5.3.2-1).

```
% c5sparse.m
clear, close   % czyszczenie pamięci roboczej i zamknięcie okien
S25=sprandn(25,25,.05)   % utworzenie macierzy rzadkiej S
spy(S25)                 % wizualizacja elementów niezerowych
title('Macierz rzadka, spy(s) wizualizuje elementy niezerowe')
A25=full(S25);           % przekształcenie na macierz pełną
whos                     % listowanie zmiennych
```

Wizualizacja z użyciem funkcji `spy` pozwala ocenić gęstość macierzy (stosunek liczby elementów niezerowych do wszystkich elementów macierzy). Każdy niezerowy element macierzy jest oznaczony punktem. Indeksy tych elementów można określić za pomocą



Rysunek 5.3.2-1. *Wizualizacja elementów niezerowych macierzy rzadkiej*

funkcji `find`, na przykład `[i,j]=find(S25)`. Polecenie `whos` potwierdza ponad dziesięciokrotną oszczędność miejsca potrzebnego dla macierzy rzadkiej `S25` w porównaniu z `A25`. Oszczędność miejsca zależy od wielkości macierzy i stopnia jej wypełnienia zerami.

Efektownym przykładem wykorzystania macierzy rzadkiej jest dostępne w MATLAB-ie demo programu `life` — wizualizacja *automatu komórkowego*, gry z regułami życia komórkowego podanymi przez brytyjskiego matematyka Johna Hortona Conwaya w roku 1970. Użycie macierzy rzadkich, wszędzie tam, gdzie to jest uzasadnione niewielką liczbą niezerowych elementów macierzy, gwarantuje obniżenie czasu symulacji i sprawniejszą animację wyników.

5.3.3. Uwagi dotyczące stosowania macierzy rzadkich

Ryzyko stosowania macierzy rzadkich polega na tym, że typowe dla tych macierzy pomijanie elementów zerowych może prowadzić do błędów. Dobrym przykładem jest użycie funkcji $cos(x)$ oraz e^x, które dla $x = 0$ mają wartość 1, a nie zero.

Przy przekształceniach macierzy rzadkiej (np. po jej odwróceniu) może wzrosnąć liczba jej elementów niezerowych. Należy więc przewidzieć odpowiedni zapas wolnych miejsc w macierzy rzadkiej poprzez ustawienie wystarczająco dużej wartości parametru `nzmax` w funkcji `sparse`.

- `S = sparse(i,j,s,m,n,nzmax)` — generuje macierz rzadką o wymiarach `m×n` i rezerwuje dla niej `nzmax` elementów niezerowych.

5.4. Łańcuchy i tablice znakowe

Łańcuch jest to tekst w postaci ciągu znaków, który ograniczono apostrofami. Informacje dotyczące łańcuchów w MATLAB-ie można uzyskać za pomocą polecenia `doc strings`. Natomiast poprzez polecenie `doc strfun` podaje funkcje wykonujące operacje na łańcuchach i tablicach znakowych.

Łańcuch jest wektorem. Pojedynczy znak łańcucha to jeden element wektora typu `char`. Wymiar wektora, który przechowuje łańcuch, określa się za pomocą funkcji `size` lub `length`. Każdy znak łańcucha reprezentuje sobą wartość według kodu ASCII. Inne znaki, np. *ćłś*, nie są poprawnie przetwarzane.

Stosując funkcję `abs`, `double`, `uint8` lub podobne, można przekształcić łańcuch w wektor wartości numerycznych kodu ASCII. Funkcja `char` wykonuje operację odwrotną.

Przykład 5.4-1

Łańcuchy i wektory: tekst *Odkryć urok MATLAB-a* jest przekształcony na liczby kodu ASCII, po czym powtórnie na tekst. Litera *ć* nie jest przetwarzana.

```
s  = 'Odkryć urok MATLAB-a'    % tekst zawiera znak ć
n1 = abs(s);  n2 = uint8(s)    % wektory: n1 typu double i n2 typu uint8
s1 = char(n1);s2 = char(n2)    % teksty otrzymane z kodu ASCII
```

Tak wyglądają wyniki z przykładu 5.4-1:

```
s =
  1×20 char array
Odkry? urok MATLAB-a
n2 =
  1×20 uint8 row vector
  Columns 1 through 10
    79   100   107   114   121    63    32   117   114   111
  Columns 11 through 20
   107    32    77    65    84    76    65    66    45    97
s2 =
  1×20 char array
Odkry? urok MATLAB-a
```

Łańcuchy można ze sobą łączyć i tworzyć z nich dłuższe napisy, np.:

```
s = [ s,' z nami']   % lub s=strcat( s,' z nami')
s =
    Odkry? urok MATLAB-a  z nami
```

Macierz tekstową można utworzyć poleceniem `char(s,' z nami')`.

Konwersja wartości numerycznych na tekst z użyciem funkcji `num2str` lub `int2scr` pozwala na łączenie tekstu z wynikami obliczeń. Jest to wykorzystywane do generowania tytułów i opisów na rysunkach oraz do wydruku wyników obliczeń, jak poniżej.

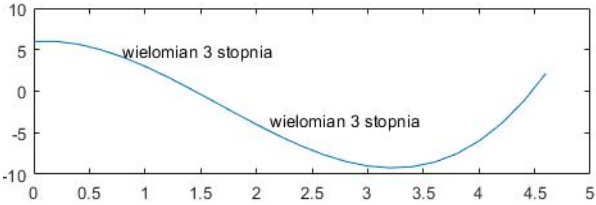
Przykład 5.4-2

Łączenie tekstu z wynikami obliczeń, `num2str` lub `int2scr` (rysunek 5.4-1):

```
n =4; txt=['wielomian ', int2str(n-1), '-go stopnia']
% użyto nawiasów prostokątnych [ ]
disp (txt)   % tekst na ekranie
x=0:0.2:4.7; plot(x, x.^3 -5*x.^2 +x+6)   % wielomian
text(0.8, 5, txt),   % tekst w punkcie (0.8, 5)
gtext(txt),   % tekst w miejscu kliknięcia myszy
```

Wybrane funkcje konwersji łańcuchów i liczb (pełna lista po wykonaniu `doc strfun`)

- `int2str` — konwersja liczby całkowitej na łańcuch,
- `num2str` — konwersja liczby na łańcuch,
- `sprintf` — konwersja liczby na łańcuch o wybranym formacie,



Rysunek 5.4-1. *Tekst i num2str lub int2scr w opisie rysunku*

- `sscanf` — konwersja łańcucha na liczbę o wybranym formacie,
- `str2num` — konwersja łańcucha na liczbę,
- `mat2str` — konwersja macierzy na łańcuch.

Poniżej zamieszczono wybrane funkcje, których argumenty są łańcuchami.

Wybrane funkcje działające na łańcuchach (pełna lista po wykonaniu `doc strfun`)

- `blanks` — generuje łańcuch spacji,
- `deblank` — usuwa spacje w łańcuchu,
- `eval` — wykonuje łańcuch jako wyrażenie lub polecenie MATLAB-a,
- `findstr` — znajduje jeden łańcuch w obrębie innego,
- `strcat` — łączy ciąg łańcuchów w wiersz,
- `str2mat` — z pojedynczych łańcuchów tworzy macierz,
- `strcmp` — porównuje łańcuchy,
- `strrep` — wyszukuje łańcuch i zamienia go,
- `lower` — zamiana liter na małe,
- `upper` — zamiana liter na wielkie.

Łańcuch może być wykonywany jako polecenie MATLAB-a. Przykłady zamieszczono w podrozdziale 3.8.7.

5.5. Tablice wielowymiarowe

Tablice wielowymiarowe są używane do przechowywania zestawów tablic o jednakowych wymiarach (mogą być wielowymiarowe). Elementy tablic wielowymiarowych wskazuje się przez podanie nazwy tablicy i indeksów w nawiasie okrągłym, na przykład `AAA(2,3,5)`.

Tablice wielowymiarowe można przekształcać przy użyciu podanych niżej funkcji. Można również wykonywać na nich operacje za pomocą funkcji i operatorów, które działają na pojedynczych elementach tablic. Natomiast nie można stosować bezpośrednio wielu funkcji, które realizują algorytmy algebry liniowej. Algorytmy te mogą być stosowane dla wybranych tablic dwuwymiarowych, będących elementami składowymi tablic wielowymiarowych. Podstawowe własności takich tablic objaśnia polecenie `doc Multidimensional Arrays` oraz `doc` *`nazwa_funkcji`*.

Funkcje dla tablic wielowymiarowych

- `cat` — powiększa tablice przez doklejanie innych tablic,
- `ndims` — podaje liczbę wymiarów, pomija wymiary nieistotne,

- `ndgrid` — generuje siatkę współrzędnych dla interpolacji wielowymiarowej,
- `permute` — przestawia wiersze, kolumny i warstwy tablicy,
- `ipermute` — odwraca efekt permutacji,
- `shiftdim` — przesuwa indeksy tablicy (+/– w lewo lub prawo),
- `circshift` — przesuwa cyklicznie wiersze, kolumny lub warstwy,
- `squeeze` — usuwa nieistotne (ang. *singleton*) indeksy tablicy.

Dokładniejsze informacje podaje `doc` *`nazwa_funkcji`*.

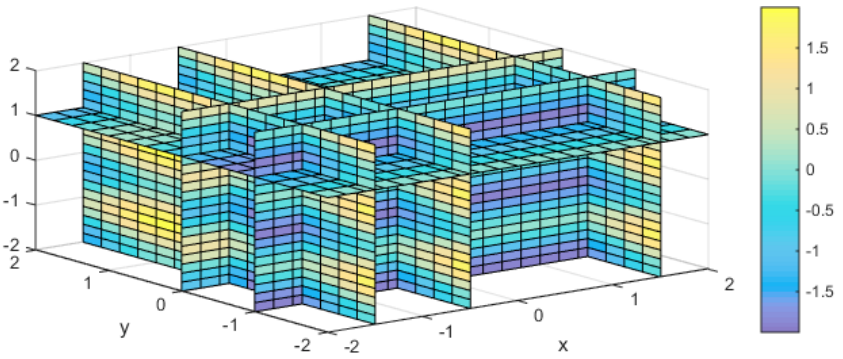
Tablice wielowymiarowe można tworzyć na kilka sposobów:

- indeksowanie — wymaga określenia wartości i indeksów wybranych elementów tablicy. Zalecane jest użycie dwukropka (`:`), tj. operatora generowania wektorów i tablic,
- zastosowanie funkcji generujących tablice wielowymiarowe, na przykład: `zeros(4,5,2)`, `rand(3,5,8)`,
- warstwowe łączenie tablic o mniejszym wymiarze za pomocą funkcji `cat` (ang. *catenate* lub *concatenate*).

Tablice wielowymiarowe umożliwiają organizowanie uporządkowanych zestawów danych dwuwymiarowych w postaci kolejnych warstw tablicy trójwymiarowej.

5.5.1. Wizualizacja tablicy trójwymiarowej

Wizualizację wartości elementów tablicy trójwymiarowej można wykonać, stosując przekroje, jak pokazano na rysunku 5.5.1-1. Najpierw należy utworzyć siatkę trójwymiarową, wykorzystując funkcję `meshgrid`, po czym punktom siatki nadaje się odpowiednie wartości.



Rysunek 5.5.1-1. *Wizualizacja danych w tablicy trójwymiarowej poprzez przekroje*

Przykład 5.5.1-1

```
% Wizualizacja tablicy trójwymiarowej
[x,y,z] = meshgrid(-2:.2:2, -2:.25:2, -2:.2:2);
v=sin(2.*y)+sin(4*z);    % obiekt, powierzchnia trójwymiarowa
h=slice(x,y,z,v,[-1.5 -0.5 1.5],[-1 0],[1])    % przekroje dla x, y, z
xlabel('x'), ylabel('y'),    % opis osi x,y
brighten(.6), ,colorbar    % zmniejszenie intensywności barw, skala barw
```

5.5.2. Tworzenie tablicy trójwymiarowej przez indeksowanie i doklejanie warstw

Poniżej podano przykład utworzenia tablicy trójwymiarowej przez indeksowanie.

Przykład 5.5.2-1

Utworzenia tablicy trójwymiarowej o wymiarze 5×4×2 poprzez określenie wartości i indeksów wybranych elementów tej tablicy.

```
BBB(5,4,1)=541;  BBB(4,2,1)=421;    % warstwa 1., dwa niezerowe elementy
BBB(:,:,2)=222;  BBB(1,4,2)=14;  BBB(4,4,2)=44    % warstwa 2.
```

Tablicę trójwymiarową `BBB` można powiększyć, dopisując dodatkowy (np. zerowy) element w punkcie przestrzeni *n*-wymiarowej, np. `BBB(5,5,5,2)=0;` powiększa tablicę do 4 wymiarów. Liczbę wymiarów można też powiększyć, doklejając dodatkowe warstwy.

Przykład 5.5.2-2

Tworzenie tablicy trójwymiarowej przez doklejanie warstw. Tablicę trójwymiarową `C` utworzono poprzez warstwowe ułożenie tablic dwuwymiarowych `A` i `B` przy użyciu funkcji `cat`.

```
A=[1,2,3;21,22,23]; B=A+100;    % dwie macierze 2x3
C=cat(3,A,B),    % elementy będą sklejane wzdłuż osi z, czyli warstwowo
```

Pierwszy parametr funkcji `cat()` to numer określający oś, wzdłuż której będą sklejane elementy.

5.6. Tablice komórkowe cell

Elementami składowymi tych tablic są komórki (ang. *cell*). Są one pojemnikami, w których można przechowywać inne tablice o różnych wymiarach, w tym inne tablice komórkowe. Komórki wskazuje się przez nazwę tablicy komórkowej i indeksy komórki podawane w nawiasie klamrowym {}.

Elementami tablicy komórkowej są *kopie innych tablic*, a nie wskaźniki (ang. *pointers*) do tych tablic. Oznacza to, że *zmiana wartości* elementu składowego tablicy komórkowej *nie przenosi się do oryginału* tej tablicy i na odwrót.

Tablice komórkowe tworzy się poprzez przypisanie indywidualnym komórkom ich zawartości, jak pokazano w przykładzie 5.6-1.

Komórki składowe tego samego wiersza oddziela się spacjami lub przecinkami, a wiersze średnikami (;).

Funkcja `cell` umożliwia wstępny przydział pamięci dla pustej tablicy komórkowej o żądanym wymiarze.

Przykład 5.6-1

Tworzenie tablicy komórkowej przez przypisanie.

```
BB = {magic(3), 6.9, 'hello'}
CC={'to jest tekst'; BB; pi; ones(5,3)}
```

Polecenie whos pokazuje aktualną zawartość przestrzeni roboczej MATLAB-a.

```
Name      Size            Bytes  Class    Attributes
BB        1x3               426  cell
CC        4x1              1028  cell
```

Tablica komórkowa BB ma trzy komórki: macierz 3×3, liczba 6.900 i tekst 'hello'.

```
BB =
  1×3 cell array
    [3×3 double]    [6.9000]    'hello'
```

Tablica komórkowa CC ma 4 komórki, w tym komórkę BB opisaną jako {1x3 cell }.

```
CC =
  4×1 cell array
    'to jest tekst'
    {1×3 cell  }
    [       3.1416]
    [5×3 double]
```

Żądany element tablicy przechowywanej w komórce wskazuje się, stosując *nawiasy klamrowe* dla wyboru komórki oraz *nawiasy okrągłe* dla indeksów tablicy umieszczonej w komórce. Ponieważ komórka może zawierać inną tablicę komórkową, *nawiasy klamrowe można zagnieżdżać*.

Przykład 5.6-2

Wskazanie wybranych elementów z tablic przechowywanych w komórkach:

```
x = BB{1}(3,2)    % element (3,2) tablicy z pierwszej komórki BB
y = CC{2}{1}(3,2)    % element {1}(3,2) z komórki CC{2}
```

Aby wydrukować tablicę z pierwszej komórki BB, należy wykonać podaną niżej instrukcję.

```
>> BB{1}(:,:)
ans =
     8     1     6
     3     5     7
     4     9     2
```

5.6.1. Funkcje do obsługi tablic komórkowych

Funkcje do obsługi i wizualizacji tablic komórkowych (więcej podaje doc datatypes)

- cell — utworzenie pustej tablicy komórkowej,
- celldisp — wypisanie zawartości tablicy komórkowej,
- cell2mat — konwersja tablicy komórkowej na macierz,
- cellplot — graficzna wizualizacja zawartości komórek,
- num2cell — przekształca tablicę liczbową w komórkową,
- deal — kopia argumentu wejściowego do argumentów wyjściowych,
- cell2struct — przekształca tablicę komórkową w strukturę,
- struct2cell — przekształca strukturę w tablicę komórkową,
- iscell — ma wartość 1 (TRUE) dla tablicy komórkowej.

5.7. Struktury

Struktura `struct` to tablica, której zawartość jest uporządkowana poprzez umieszczanie jej elementów składowych w *polach* (ang. *fields*). Cechą charakterystyczną struktury jest zapis jej elementów z użyciem *notacji kropkowej*, np. `student.nazwisko`, czyli kropka oddziela nazwę struktury od nazwy pola:

```
nazwa_struktury.nazwa_pola
```

Strukturę można utworzyć poprzez:

- przypisanie wartości do pól struktury,
- użycie funkcji `struct`.

5.7.1. Tworzenie struktury przez przypisanie

Poniżej utworzono strukturę o nazwie `Auto`, z polami określającymi typ samochodu, zużycie paliwa i zapas paliwa w baku.

Przykład 5.7.1-1

Tworzenie struktury przez przypisanie wartości do pól struktury.

```
Auto.model='golf';
Auto.zuzyciePaliwa100km=6;
Auto.rezerwaPaliwa=40;
Auto(2).model='pick-up';    % przypisuje model do Auto(2)
Auto(2).rezerwaPaliwa=60;
Auto(2).zuzyciePaliwa100km=8.6;
```

Wartości przypisane do pól drugiego elementu tej struktury podaje polecenie `Auto(2)`.

```
struct with fields:
                 model: 'pick-up'
    zuzyciePaliwa100km: 8.6000
         rezerwaPaliwa: 60
```

Elementami struktury mogą być także tablice, podobnie jak w tablicach komórkowych `cell`.

5.7.2. Tworzenie struktury z użyciem funkcji struct

Wybrane funkcje obsługujące struktury (pełna lista: `doc datatypes`)

- `struct` — utworzenie struktury, przekształcenie obiektu w strukturę,
- `fieldnames` — podaje nazwy pól struktury,
- `getfield` — pobiera wartość pola.

Parametrami funkcji `struct` są pary: nazwa pola i wartość przypisywana temu polu.

```
S = struct('field1',value1, 'field2',value2, …).
```

Polu struktury można także przypisać tablicę wielowymiarową lub komórkową. Funkcja `struct` służy do tworzenia struktur oraz do przekształcania obiektów na odpowiadające im struktury (obiekty opisano w rozdziale 6.).

Przykład 5.7.2-1

Tworzenie struktury `Auto` z wykorzystaniem funkcji `struct`:

```
Auto=struct('model',' Smart','zuzyciePaliwa100km',4.9,'rezerwaPaliwa',22)
```

Wynikiem tego polecenia jest:

```
Auto =
                model: ' Smart'
    zuzyciePaliwa100km: 4.9
         rezerwaPaliwa: 22
```

Przykład 5.7.2-2

Wartość przechowywaną we wskazanym polu struktury podaje `getfield`:

```
getfield(Auto(1), zuzyciePaliwa100km)    % nawias okrągły
```

lub po prostu `Auto(1).zuzyciePaliwa100km`.

W obu powyższych przypadkach odpowiedź będzie identyczna: *ans = 4.9*. Więcej informacji podaje polecenie `doc struct`.

5.8. Tabele i typ wyliczeniowy categorical

Tabela jest zbudowana z kolumn o tej samej długości. Kolumna ma swoją nazwę i jej elementy należą do tego samego typu danych.

5.8.1. Tworzenie i obsługa tabel

Do tworzenia i konwersji tabel używane są funkcje `table`, `array2table`, `cell2table`, `struct2table`, `table2array`, `table2cell`, `table2struct`. Do importu i exportu tabel używa się `readtable`, `writetable`, `write`.

Rozmiary tabeli podają polecenia `size`, `width`, `height`, `ndims`, `numel`, `horzcat`, `vertcat`, a do sprawdzenia typu zmiennej można użyć `whos` oraz `istable`.

Polecenia dotyczące przynależności to `intersect`, `ismember`, `setdiff`, `setxor`, `unique`, `union`, `join`, `innerjoin`, `outerjoin`.

Metadane tabeli są dostępne w polach: *Description*, *DimensionNames*, *VariableNames*, *VariableDescription*, *VariableUnits*, *RowNames*, *UserData*. W przykładzie 5.8.1-3 zmieniono nazwę w polu *VariableNames*.

Funkcje ułatwiające zarządzanie danymi w tabeli to: `summary`, `sortrows`, `stack`, `unstack`, `ismissing`, `standardizeMissing`. Zmianę wartości elementów tabeli dokonuje się podobnie jak w strukturze, ale można też użyć funkcji `varfun`, `rowfun`.

Przykład 5.8.1-1

Utworzenie tabeli z dostępnego w MATLAB-ie arkusza Excela *patients.xls* poleceniem:

```
patientTab=readtable('patients.xls','ReadRowNames',true);
% file MATLAB\toolbox\matlab\demos\patients.xls
```

Zawartość wierszy od 1. do 3. i kolumn od 1. do 4. utworzonej tabeli podaje polecenie `patientTab(1:3,1:4)`.

```
             Gender     Age    Location                      Height
Smith        'Male'     38     'County General Hospital'     71
Johnson      'Male'     43     'VA Hospital'                 69
Williams     'Female'   38     'St. Mary's Medical Center'   64
```

Tabelę można też zbudować z wektorów dostępnych w pliku *patients.mat*. Po załadowaniu tego pliku w przestrzeni roboczej jest dziesięć 100-elementowych wektorów o nazwach: *Age*, *Diastolic*, *Gender*, *Height*, *LastName*, *Location*, *SelfAssessedHealthStatus*, *Smoker*, *Systolic*, *Weight*. Wybrane wektory będą użyte w przykładzie 5.8-2.

Przykład 5.8.1-2

Załadowanie do przestrzeni roboczej MATLAB-a danych z pliku *patients.mat* i utworzenie tabeli:

```
clear, clc    % usunięcie zmiennych, czyszczenie okna Command
load  patients    % załadowanie 10 wektorów o wymiarze 100×1.
patientTab = table(Age, Height, Weight, Smoker, Gender, LastName);
patientTab(1:3,:)    % listowanie: wiersze 1-3, wszystkie kolumny
```

Pierwsze trzy linie nowo utworzonej tabeli `patientTab` mają następującą postać:

```
    Age    Height    Weight    Smoker     Gender      LastName
    38     71        176       true      'Male'      'Smith'
    43     69        163       false     'Male'      'Johnson'
    38     64        131       false     'Female'    'Williams'
```

Przykład 5.8.1-3

Powyższe dane pochodzą z USA, należy więc dokonać przeliczeń, uwzględniając, że *1 lb* to 0.45359 kg oraz *1 in* odpowiada 0.0254 m. Dokonano też zmiany nagłówków na polskie.

```
patientTab.Height=patientTab.Height*0.0254;    % konwersja cali na metry
patientTab.Weight=patientTab.Weight*0.45359;    % konwersja lb na kg
patientTab.Properties.VariableNames{'Height'}='Wzrost';    % zmiana nagłówka
patientTab.Properties.VariableNames{'Weight'}='Waga';    % zmiana nagłówka
patientTab(1:3,:)    % listowanie: wiersze 1-3, wszystkie kolumny
patientTab.Properties.RowNames = LastName;    % nazwisko jest nazwą wiersza
patientTab.LastName =[]    % usunięcie kolumny LastName
          Age    Wzrost      Waga      Smoker     Gender
Smith     38     1.8034    79.83184    true      'Male'
Johnson   43     1.7526    73.93517    false     'Male'
Williams  38     1.6256    59.42029    false     'Female'
```

5.8.2. Tablica wyliczeniowa, categorical array

Elementy tablicy wyliczeniowej `categorical array` przyjmują wartości należące do skończonego zbioru elementów. Elementami takiej tablicy mogą być łańcuchy, wartości liczbowe lub logiczne.

Przykład 5.8.2-1

Korzystając z przygotowanej w przykładzie 5.8.1-2 tabeli `patientTab`, przekształcamy kolumny *Smoker* i *Gender* na tablice *categorical array*:

```
patientTab.Gender = categorical(patientTab.Gender);
patientTab. Smoker = categorical(patientTab. Smoker);
```

Jednym z widocznych efektów tego przekształcenia jest zmniejszenie pamięci zajmowanej przez tabelę patientTab. Poprzednio zajmowała ona 29796 bajtów, a obecnie zajmuje 18582 bajty.

5.9. Datatime, duration, calendarDuration

Typy danych datetime, duration i calendarDuration służą do określania daty i czasu oraz do obliczania upływu czasu pomiędzy określonymi datami i godzinami.

5.9.1. Typ datetime

Typ datetime to klasa macierzy określających datę i czas, np.:

```
>> teraz = datetime
teraz =
  datetime
   05-Apr-2017 08:23:42
```

lub data jako cyfry oznaczające rok, miesiąc, dzień itd.

```
t1=datetime(2017,12,31,9,30,0), t2=datetime(2017,1,1)
t1 =
  datetime
   31-Dec-2017 09:30:00
t2 =
  datetime
   01-Jan-2017
```

Parametrem dla datetime jest określony moment:

- w postaci tekstowej, np. now, today, tomorrow, yesterday,
- jako 3- lub 6-elementowy *wektor* liczbowy (np. 2015,02,18,7,30,0),
- jako *macierz* mająca 3 kolumny lub 6 kolumn (względnie 3 lub 6 *oddzielnych macierzy*).

Przykładem macierzy datetime o 3 elementach jest 1×3 datetime array.

```
>> 2017,1,1,9:11,30,0
tt3 =
  1×3 datetime array
   01-Jan-2017 09:30:00   01-Jan-2017 10:30:00   01-Jan-2017 11:30:00
```

Więcej informacji podaje doc datetime.

5.9.2. Typ duration

Wynikiem odejmowania zmiennych typu datetime jest utworzenie zmiennej typu duration, która służy do określenia upływu czasu. Wynik jest podawany w godzinach, minutach i sekundach, np.:

```
>> t1=datetime(2016,12,31,9,30,0); t2=datetime(2017,1,1);
>> time=t2-t1
time =
  duration
   14:30:00
```

Wynik można przeliczyć na godziny lub inne jednostki: `y`, `d`, `h`, `m`, `s`. Miesięcy się nie oblicza. Przyjmuje się, że rok ma 365.2425 dni. Dokładniej będzie w typach `timetable` i `calendarDuration`.

```
>> hours=duration(time, 'Format','h')
hours =
  duration
   14.5 hr
```

Więcej informacji podaje `doc datetime` i `doc duration`.

5.9.3. Typ calendarDuration

Typ `calendarDuration`, podobnie jak `timeTable`, uwzględnia zmianę długości kolejnych miesięcy w roku i lata przestępne. Będzie to pokazane na przykładzie.

Przykład 5.9.3-1

Ile czasu upływa pomiędzy datami 6.02.2017 a 7.03.2017?

```
>> L2 = calendarDuration(2017,2,6);L3=calendarDuration(2017,3,7);
>> L32=L3-L2
L32 =
  calendarDuration
   1mo 1d
```

Jest to prawidłowa odpowiedź: `1mo 1d` to 1 miesiąc + 1 dzień. Typ `duration` poda tą odpowiedź w innej formie: 29 dni to także 1 miesiąc + 1 dzień (luty 2017 ma 28 dni).

```
>> d2=datetime(2017,2,6);d3=datetime(2017,3,7);
>> duration(d3-d2, 'Format','d')
ans =
  duration
   29 days
```

5.9.4. Timetable

`Timetable` jest rodzajem tabeli, która łączy czas z każdym wierszem tabeli. `Timetable` może przechowywać w kolumnach różne typy danych, ale wszystkie kolumny muszą mieć taką samą liczbę wierszy. Poniżej (a także w przykładzie 11.2.3.2-2) pokazano przykład tworzenia timetable:

```
>> cisnienie = [1004;1008;1002;998];    % wektor kolumnowy, 4 elementy
>> czas = datetime(2017,4,1:4,8,30,0); % 4 dni: 1-4.04.2017, godz. 8:30 --- wektor poziomy
>> temp = [15;14;12;13];                % temperatury
>> tt=timetable(czas',temp,cisnienie)
tt =
  4×2 timetable
           Time            temp    cisnienie
    ____________________    ____    _________
    01-Apr-2017 08:30:00    15      1004
```

```
02-Apr-2017 08:30:00    14      1008
03-Apr-2017 08:30:00    12      1002
04-Apr-2017 08:30:00    13       998
```

5.10. Tall Array i Tall Table — na duże zbiory danych

Tall Table, *tall array* i obiekty typu *datastore* umożliwiają pracę z dużymi zbiorami danych, które są ładowane do pamięci w małych kawałkach. Dzięki temu tablice mogą być dowolnie duże, w szczególności mogą mieć dowolną liczbę wierszy. Jeśli dane w *tall array* mają postać tabeli, to używa się nazwy *tall table*. *Tall table* są omówione w podrozdziale 11.2.3.2.

Rozdział 6.

Klasy i programowanie obiektowo zorientowane

Programowanie obiektowo zorientowane (ang. *object oriented programming* — OOP) jest bardzo efektywną metodyką programowania. Obiekty to instancje danej klasy. Klasa to struktura zawierająca atrybuty oraz metody. Atrybuty to dane określające aktualny stan obiektu. Metoda to funkcja przeznaczona do obsługi obiektów należących do danej klasy. Podczas przygotowania oprogramowania obiektowo zorientowanego można korzystać z diagramów języka **UML**.

Wszystkie typy danych w środowisku MATLAB są klasami, a zmienne są obiektami należącymi do tych klas. Przykładami klas są typy: `double`, `sparse`, `char`, `struct`, `cell`, `uint8`. Programy obiektowo zorientowane zawierają dodatkowe klasy definiowane przez programistę. Wiele klas zdefiniowano w bibliotekach toolbox.

6.1. Definiowanie klasy — classdef

W języku MATLAB programista ma możliwość definiowania własnych klas. Można to zrobić na dwa sposoby:

- Definiując klasy z użyciem `classdef`, jak opisano poniżej. Taką klasę można zapisać w dowolnym, przeszukiwanym przez MATLAB folderze. Nazwa pliku powinna być taka sama jak nazwa klasy i powinna mieć rozszerzenie *.m*.
- Definiując każdą klasę w oddzielnym folderze *@nazwaKlasy*. Ten sposób jest nadal używany w bibliotekach MATLAB-a i był opisany w poprzednich wydaniach tej książki.

Utworzenie klasy jest proste, jeśli korzysta się z pokazanego poniżej gotowego formularza:

```
classdef Untitled
  % UNTITLED Summary of this class goes here
  %   Detailed explanation goes here
    properties
    end
```

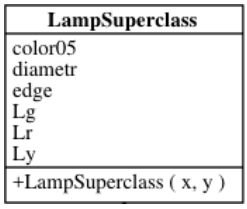
```
    methods
    end
end
```

Formularz jest dostępny w panelu MATLAB-a, w zakładkach *HOME* lub *EDIT*. Po wybraniu menu *New*▼, a następnie *Class*, czyli *HOME/New/Class* lub *EDITOR/New/Class* — otworzy się okno edytora z gotowym formularzem.

6.1.1. Przykład definiowania klasy — classdef

Opisana poniżej klasa będzie użyta do modelowania sygnalizatora świetlnego do sterowania ruchem drogowym. Nie jest to konieczne, ale w nazwie klasy `LampSuperclass` umieszczono słowo *Superclass*, aby zaznaczyć, że ta klasa będzie zawierała podklasy. W bloku `properties` zdefiniowano *atrybuty* (czyli zmienne) określające *właściwości* przyszłych obiektów. W bloku `methods` jedyną metodą jest `LampSuperclass` (ta sama nazwa co nazwa klasy). Ta metoda *tworzy obiekty* i jest zwana *konstruktorem*.

Na rysunku 6.1.1-1 pokazano diagram UML (ang. *Unified Modeling Language*). Ikona klasy `LampSuperclass` ma trzy przedziały: w górnym przedziale umieszcza się *nazwę klasy*, w środkowym podaje się potrzebne *atrybuty*, czyli zmienne, które będą deklarowane w bloku `properties`, a w dolnym przedziale podaje się nazwy *metod*, czyli usług lub funkcji, które są potrzebne w tej klasie.



Rysunek 6.1.1-1. *Diagram UML dla LampSuperclass, wykonany z użyciem programu UMLgui z www.mathworks.com/matlabcentral/fileexchange/*

Diagramy klas są jednym z istotnych elementów projektu informatycznego i są zazwyczaj tworzone oraz starannie weryfikowane, zanim przystąpi się do tworzenia kodów programu komputerowego [Mrozek, 2011].

Przykład 6.1.1-1

Poniżej podano kod definiujący klasę `LampSuperclass`, przygotowany w oparciu o diagram UML pokazany na rysunku 6.1.1-1. Dokładniejszy opis kodu podano w podrozdziale 6.1.2.

```
classdef LampSuperclass
    % superklasa, czyli przodek innych klas np. trafficLamp
    % clear, close, obj=LampSuperclass(3,4), axis equal % wywołanie konstruktora

    properties
        diametr=[0.8  0.8]       % diametr of lamp / średnica lampy
        color05=[0.5 0.5 0.5]    % gray color / kolor szary
        Lg, Ly, Lr    % green, yellow and red lamp / zielona, żółta i czerwona
    end

    properties (Access=private)
        edge=2.5                 % edge width / szerokość krawędzi lampy
    end
```

```
    methods
        function obj=LampSuperclass(x,y)
            % konstruktor, tworzy obiekty obj należące do tej klasy
          % constructor method, creates object obj, instance of this class
            obj.Lr= rectangle('Position',[x,y+2,obj.diametr],…
                'Curvature',[1,1],'FaceColor',obj.color05,…
                'LineWidth',obj.edge,'LineStyle','-');    % red lamp /lampa czerwona
            obj.Ly= rectangle('Position',[x,y+1,obj.diametr],…
                'Curvature',[1,1],'FaceColor',obj.color05,…
                'LineWidth',obj.edge,'LineStyle','-');    % yellow lamp / lampa żółta
            obj.Lg= rectangle('Position',[x,y,obj.diametr],…
                'Curvature',[1,1],'FaceColor',obj.color05,…
                'LineWidth',obj.edge,'LineStyle','-');    %green lamp / lampa zielona
        end

    end
end
```

6.1.2. Tworzenie obiektu należącego do klasy

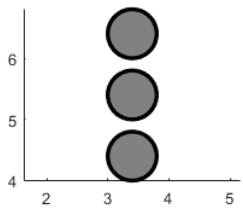
Metoda `function obj=LampSuperclass(x,y)` służy do tworzenia obiektów i jest nazywana *konstruktorem*. Konstruktor ma nazwę identyczną z nazwą klasy. Obiekt to *instancja klasy*, czyli *zmienna* utworzona *zgodnie z opisem* podanym przez *klasę*.

Aby utworzyć obiekt `obj` należący do klasy `LampSuperclass`, należy wywołać konstruktora tej klasy, czyli wykonać `obj=LampSuperclass(x,y)` — z odpowiednimi parametrami, np. wywołanie `lampObj = LampSuperclass(3,4)` tworzy obiekt `lampObj`. Jeśli obiekt o podanej nazwie już istnieje, polecenie nie wykona się. Dlatego przed testowaniem programów należy usunąć niepotrzebne zmienne (polecenie `clear`) oraz zamknąć okna graficzne (polecenie `close all` — jak pokazano poniżej).

```
clear, close all, LampObj=LampSuperclass(3,4), axis equal
```

Polecenie to można wpisać z klawiatury w oknie *Command*, jeśli plik *LampSuperclass.m* jest dostępny publicznie oraz jeśli znajduje się on w folderze aktualnym lub w folderze przeszukiwanym przez MATLAB-a. Polecenie `axis equal` zapobiega owalizacji okręgów.

Obiekt `lampObj` pokazano na rysunku 6.1.2-1. Jest modelem sygnalizatora świetlnego służącego do sterowania ruchem drogowym. Składa się on z trzech okrągłych lamp *Lr*, *Ly*, *Lg* wyrysowanych przez `rectangle`. Lewy dolny narożnik obiektu `lampObj` jest umieszczony w punkcie (3, 4) przekazanym jako parametry wejściowe konstruktora `LampSuperclass(x,y)`. Polecenie `axis equal` wyrównuje skalowanie osi pionowej i poziomej rysunku, aby okręgi nie miały kształtu elipsy.



Rysunek 6.1.2-1. *Obiekt lampObj należy do klasy LampSuperclass, jego elementami są trzy lampy: Lr, Ly, Lr. Lampy są wygaszone i mają kolor szary*

Obiekty należące do klasy `LampSuperclass` są mało użyteczne, gdyż nie mają metod do zmiany koloru świateł. Dodatkowe metody będą przygotowane dopiero w podrozdziale 6.2 jako elementy klasy `ColorLampClass`, będącej *potomkiem*, czyli *podklasą* klasy `LampSuperclass`.

Dziedziczenie klas (ang. *inheritance*) jest techniką pozwalającą na łatwą modyfikację i rozbudowę oprogramowania obiektowo zorientowanego — bez ponoszenia ryzyka uszkodzenia wcześniejszych jego wersji.

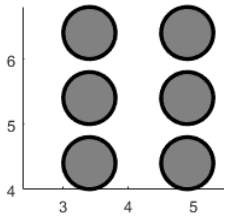
6.1.3. Tworzenie dodatkowych obiektów

Dodatkowe obiekty można utworzyć przez kolejne wywołania konstruktora, np.:

```
lampObj2=LampSuperclass(4.5, 4);
```

utworzy dodatkowy obiekt `lampObj2` (rysunek 6.1.3-1). Nazwy utworzonych obiektów są widoczne w oknie *Workspace* oraz jako wynik polecenia `whos`.

```
Name          Size            Bytes  Class             Attributes
lampObj       1x1                72  LampSuperclass
lampObj2      1x1                72  LampSuperclass
```



Rysunek 6.1.3-1. *Dwa obiekty — lampObj i lampObj2*

6.1.4. Bloki properties, methods i events w definicji klasy

W bloku `properties` definiuje się *zmienne*, zwane też polami lub atrybutami. Blok `properties` może mieć nadane atrybuty określające dodatkowe własności definiowanych w nim zmiennych (ang. *property attributes*). Można utworzyć bloki z różnymi atrybutami, np.:

- `properties (Constant)` — w bloku zdefiniowano stałe,
- `properties (Access=private)` — dostęp tylko dla obiektów danej klasy,
- `properties (Access=protected)` — poszerza dostęp o potomków danej klasy.

Dopuszczalne atrybuty bloku `properties` to: `AbortSet`, `Abstract`, `Access`, `GetAccess`, `SetAccess`, `Constant`, `Dependent`, `GetObservable`, `SetObservable`, `Hidden`, `NonCopyable`, `Transient`. Atrybuty `Access`, `GetAccess`, `SetAccess` mogą mieć jedną z wartości: `private`, `protected` lub `public`, gdzie `public` jest wartością domyślną.

6.1.4.1. Bloki methods

W bloku `methods` definiuje się konstruktora oraz inne metody, czyli funkcje obsługujące obiekty należące do definiowanej klasy. Do bloku `methods` można przypisać atrybuty `Abstract`, `Access`, `Hidden`, `Sealed`, `Static`. Więcej informacji o blokach `properties`

i `methods` podaje `doc classdef` i dostępne w nim linki do *Property Syntax* oraz do *How to Use Methods*.

6.1.4.2. Blok events

Jeśli klasa ma obsługiwać zdarzenia (np. kliknięcie myszą), to jednym z jej przodków powinna być klasa `handle`, pobierana z bibliotek MATLAB-a. Klasa dziedzicząca od `handle` może mieć dodatkowy blok:

```
events … end
```

służący do obsługi zdarzeń (ang. *events*). Więcej informacji o obsłudze zdarzeń podaje `doc classdef` i dostępny tam link do *Events and Listeners Syntax*.

6.1.5. Pobieranie i zmiana wartości atrybutów obiektu

Poniżej podano przykłady pobrania i ewentualnej zmiany wartości atrybutów obiektu, czyli wartości zmiennych zadeklarowanych w bloku `properties`. W celu wylistowania atrybutów wystarczy utworzyć obiekt i wpisać jego nazwę w oknie *Command*. Ewentualnie można wpisać nazwę atrybutu w notacji kropkowej, np. `lampObj.Lg`.

```
>> lampObj    % podaje atrybuty (zmienne, właściwości, pola) obiektu lampObj

lampObj =
     LampSuperclass with properties:

    diametr: [0.8000 0.8000]
    color05: [0.5000 0.5000 0.5000]
         Lg: [1×1 Rectangle]
         Ly: [1×1 Rectangle]
         Lr: [1×1 Rectangle]
```

Dla zmiennych `diametr` i `color05` otrzymano ich nazwy i przypisane do nich wartości. Zmienne `Lg`, `Ly`, `Lr` były utworzone z użyciem metody `rectangle` i są one jednocześnie obiektami należącymi do klasy `matlab.graphics.primitive.Rectangle` i atrybutami obiektu `lampObj`. Z kolei zmienna `edge` ma przypisany atrybut `private` i nie jest widoczna na powyższym listingu. Próba odczytania wartości tej zmiennej przez wpisanie w oknie *Command* nazwy `lampObj.edge` zakończy się niepowodzeniem.

```
>> lampObj.edge
No public property 'edge' for class 'LampSuperclass'.
```

Atrybuty lampy `Lg` nie są prywatne i można je wyświetlić, wpisując w oknie *Command* pełną nazwę tej zmiennej w notacji kropkowej, czyli wpisując polecenie `lampObj.Lg`.

```
>> lampObj.Lg
ans =
  Rectangle with properties:

    FaceColor: [0.5000 0.5000 0.5000]
    EdgeColor: [0 0 0]
    LineWidth: 2.5000
    LineStyle: '-'
    Curvature: [1 1]
     Position: [3 4 0.8000 0.8000]
```

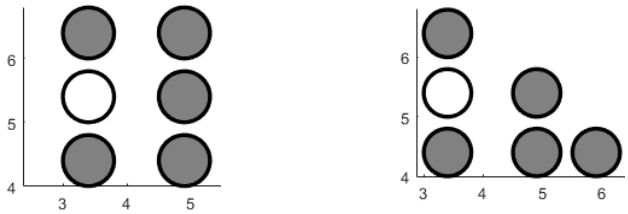
Wartości przypisane do atrybutów obiektu można zmienić, podając jego pełną nazwę i nową wartość.

Przykład 6.1.5-1

Zmiana koloru lampy *Lg*, należącej do obiektu `lampObj` (kolor biały `[1 1 1]` przypisano do pola `FaceColor`), oraz przesunięcie lampy *Lr*, należącej do obiektu `lampObj2` (nowe wartości pola `Position`).

```
lampObj.Lg.FaceColor =[1 1 1]
lampObj2.Lr.Position = [5.5  4 .8000 0.8000]
```

Efekt tych zmian jest widoczny po porównaniu rysunków 6.1.3-1 i 6.1.5-1.



Rysunek 6.1.5-1. *Zmiany atrybutów (pól) obiektu lampObj po wykonaniu pierwszej linii kodu z przykładu 6.1.5-1 oraz atrybutów (pól) obiektu lampObj2 po wykonaniu drugiej linii kodu*

Zmiana koloru sygnalizatorów (zielone, czerwone lub żółte) będzie realizowana przez podklasę `ColorLampClass`, opisaną w podrozdziale 6.2. Klasa `LampSuperclass` nie będzie modyfikowana.

6.1.6. Atrybuty private i hidden, metody get i set

Istotną cechą programowania obiektowo zorientowanego jest możliwość hermetyzacji, czyli zablokowania bezpośredniego dostępu do wskazanych danych. Przykładem jest zmienna `edge` w pliku *LampSuperclass.m*. Jest to zmienna prywatna, zadeklarowana w bloku `properties` (`Access=private`). Jej wartość jest dostępna wewnątrz wszystkich obiektów należących do klasy `LampSuperclass`. Natomiast **próba odczytania wartości** tej zmiennej poza tymi obiektami, np. w oknie *Command*, zakończy się komunikatem o błędzie, jak pokazano w podrozdziale 6.1.5.

```
>> lampObj.edge
No public property 'edge' for class 'LampSuperclass'.
```

Bezpiecznym sposobem udostępnienia zmiennej prywatnej jest utworzenie tak zwanych metod GET i SET.

Przykład 6.1.6-1

Dodatkowy blok kodu `methods(Hidden=true)` został umieszczony wewnątrz klasy `LampSuperclass`, przed ostatnią linią `end` w pliku *LampSuperclass.m*. Użycie metod `GETedge` i `SETedge` umożliwia dostęp do zmiennej prywatnej `edge` spoza obiektów należących do klasy `LampSuperclass`, w tym z okna *Command*.

```
methods(Hidden=true)
        function edge=GETedge(obj)
            edge=obj.edge;
        end
```

```
        function obj=SETedge(obj,edgeNew)
            obj.edge=edgeNew;
        end
    end   %
```

Przykład 6.1.6-2

Poniższy skrypt pozwala przetestować wywołanie metod GET i SET w oknie *Command*.

```
% file / plik getsetTest.m
clear, close, lampObj=LampSuperclass(3,4); axis equal % create object
                                                      % utworzenie obiektu
ee=GETedge(lampObj)    % reading private variable / odczyt zmiennej prywatnej
lampObj=SETedge(lampObj,5); % new value of edge / nowa wartość zmiennej
ee=GETedge(lampObj)    % reading private variable / odczyt zmiennej
```

Metody `GETedge` i `SETedge` są dostępne publicznie, ale *nie są widoczne* (ang. *Hidden*). Jeśli użytkownik programu nie ma dostępu do kodu źródłowego i nie został poinformowany, jakie nazwy mają metody GET i SET, oraz nie wie, jaką nazwę ma zmienna prywatna, to nie może odczytać ani zmienić wartości zmiennej prywatnej.

Należy zwrócić uwagę, że jednym z argumentów obu metod GET i SET jest nazwa obiektu. Oznacza to, że korzystając z tych metod, można zmienić wartość zmiennej przypisanej do wskazanego obiektu, a nie wartość tej zmiennej w konstruktorze, używanej przy tworzeniu nowych obiektów.

6.2. Dziedziczenie klas

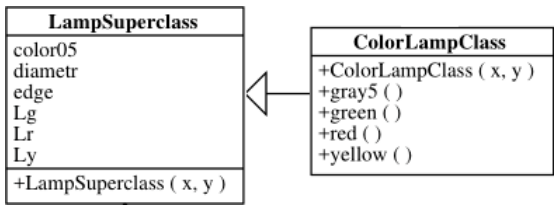
Dziedziczenie klas (ang. *inheritance*) jest techniką pozwalającą na łatwą modyfikację i rozbudowę oprogramowania obiektowo zorientowanego — bez ponoszenia ryzyka uszkodzenia wcześniejszych jego wersji.

Klasa `LampSuperclass` z poprzedniego podrozdziału została przetestowana i nie będzie już modyfikowana. Każda modyfikacja może spowodować wadliwe działanie programu. Zamiast tego zostanie zdefiniowana *podklasa* (ang. *subclass*), czyli nowa klasa dziedzicząca metody i atrybuty od swojego *przodka*, czyli od *superklasy*. W nowej klasie mogą być umieszczone nowe metody i atrybuty. Nową klasę nazwano `ColorLampClass`. Klasy `LampSuperclass` i `ColorLampClass` tworzą hierarchię przodek-potomek (ang. *parent-child*).

Ta sama klasa, jeśli ma zarówno swojego przodka, jak i potomków, może być jednocześnie podklasą i superklasą.

6.2.1. Definiowanie podklasy

Polecenie `classdef ColorLampClass < LampSuperclass` definiuje klasę `ColorLampClass` jako potomka klasy `LampSuperclass`. Podklasa `ColorLampClass` dziedziczy wszystkie pola i metody swojego przodka. Klasa potomna ma zazwyczaj dodatkowe własne atrybuty i metody (rysunek 6.2.1-1). Jeśli jest taka potrzeba, to wybrane atrybuty i metody mogą być powtórnie zdefiniowane, czyli przeciążone (ang. *overloading*). Przeciążanie (podrozdział 6.4) umożliwia zmianę sposobu działania metod i operacji nawet w sytuacji, gdy nie ma się dostępu do kodu źródłowego przodka.



Rysunek 6.2.1-1. *Przodkiem podklasy ColorLampClass jest LampSuperclass. Diagramy wykonano z użyciem programu UMLgui z www.mathworks.com/matlabcentral/fileexchange/*

Przykład 6.2.1-1

Definiowanie klasy `ColorLampClass`. Klasa ma dziedziczyć pola i metody po klasie `LampClass` oraz posiadać dodatkowe funkcje służące do zmiany koloru świateł `red`, `green`, `yellow`, `gray5`.

```
classdef ColorLampClass < LampClass
% inherits methods and properties from LampClass /dziedziczy atrybuty i metody od LampClass
% clear,close, lampW=ColorLampClass(3,4), lampW=red(lampW) % example/przykład
    properties
    end

    methods
        function obj=ColorLampClass(x,y)
            obj=obj@LampClass (x,y)    % wywołanie przodka
                                       % using superclass
        end
        function obj=red(obj)
            obj=gray0(obj);              % gasi wszystkie światła
            set(obj.Lr,'FaceColor','r'); % zapala światło czerwone / red light on
        end
        function obj=green(obj)
            obj=gray0(obj);
            set(obj.Lg,'FaceColor','g');    % zapala światło zielone / green light on
        end
        function obj=yellow(obj)
            obj=gray0(obj);
            set(obj.Ly,'FaceColor','y');    % zapala światło żółte / yellow light on
        end %function
        function obj=gray0(obj)
            % changes all FaceColor to Gray /zmienia kolor elementów obiektu na szary
            set(obj.Lr,'FaceColor',obj.colorGray);
            set(obj.Lg,'FaceColor',obj.colorGray);
            set(obj.Ly,'FaceColor',obj.colorGray);
        end
    end % methods
end %    classdef
```

Zadaniem klasy dziedziczącej jest tworzenie nowych obiektów z użyciem klasy przodka. Tutaj przodkiem jest `LampClass`, a umieszczone w *konstruktorze* `ColorLampClass(x,y)` wywołanie przodka ma postać `obj=obj@LampClass(x,y)`. W wywołaniu użyto dwukrotnie nazwy lokalnej tworzonego obiektu (tutaj: `obj`) oraz separatora `@` i *nazwy przodka* `LampClass`.

Obiekty należące do klasy `ColorLampClass` *dziedziczą* wszystkie *pola* i *metody* swojego *przodka* `LampClass` oraz mają *dodatkowe metody* zdefiniowane w klasie `ColorLampClass`.

6.2.2. Obiekty należące do podklasy

Aby utworzyć obiekt należący do klasy `ColorLampClass`, należy wywołać konstruktora `ColorLampClass(x,y)`, a utworzony obiekt przypisać do nowej zmiennej, np. `lampW`, jak w przykładzie poniżej. Wywoływana klasa `ColorLampClass` musi być dostępna w folderze aktualnym MATLAB-a lub w folderach przeszukiwanych.

Przykład 6.2.2-1

Utworzenie dwu obiektów o nazwach `lampW` i `LampE` należących do klasy `ColorLampClass`.

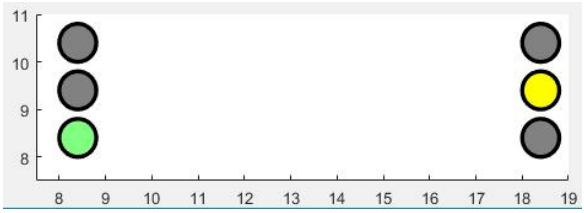
```
clear, close all    % ewentualne usunięcie starych obiektów
lampW= ColorLampClass(8,8);    % lamp, West direction /obsługuje kierunek zachodni
lampE= ColorLampClass(18,8);   % lamp, East direction /obsługuje kierunek wschodni
axis([7.5,19, 7.5, 11])    % [xmin xmax ymin ymax], plot area / obszar rysunku
```

Przykład 6.2.2-2

Metody `obj=green(obj)`, `obj=red(obj)` i `obj=yellow(obj)` zapalają kolorowe lampy na wybranych obiektach: `lampW` i `lampE`.

```
lampW = green(lampW); % lampW is input and output / dwukrotnie użyto LampW
lampE = yellow(lampE); % yellow lamp is on/ tu zapalono żółty
```

Zapalono światła zielone i żółte na obiektach `lampW` i `lampE` (rysunek 6.2.2-1).



Rysunek 6.2.2-1. *Dwa obiekty, lampW i lampE, należące do klasy ColorLampClass*

Polecenie `whos` pokazuje aktualną zawartość przestrzeni roboczej: dwa obiekty należące do klasy `ColorLampClass` są utworzone przez skrypt z przykładu 6.2.2-1.

```
Name        Size              Bytes  Class          Attributes
lampE       1x1                 480  ColorLampClass
lampW       1x1                 480  ColorLampClass
```

Podanie nazwy obiektu jako polecenia, na przykład `lampE`, spowoduje wyświetlenie wartości jego atrybutów: `diametr`, `colorGray`, `Lg`, `Ly`, `Lr`. Każdy z atrybutów `Lg`, `Ly`, `Lr` ma wartość `[1×1 Rectangle]`, co oznacza, że jest to obiekt należący do klasy `Rectangle`, dokładniej do wbudowanej (ang. *build-in*) klasy `matlab.graphic.primitive.Rectangle`. Atrybuty tego obiektu można odczytać poleceniem `lampW.Lg` — po wykonaniu przykładów 6.2.2-1 i 6.2.2-2. Opis klasy `Rectangle` podaje `doc Rectangle`.

6.2.3. Funkcje do obsługi klas i obiektów

Poniżej podano nazwy wybranych funkcji używanych do tworzenia programów obiektowo zorientowanych w języku MATLAB.

- `classdef` — tworzy klasy,
- `class` — określa typ zmiennej,
- `methods` — lista metod obsługujących daną klasę, w tym metody dziedziczone,

- `isa` — ma wartość 1, jeśli obiekt należy do danej klasy,
- `isobject` — ma wartość 1, jeśli zmienna jest obiektem,
- `superiorto` — relacja nadrzędna klas: ma pierwszeństwo,
- `inferiorto` — relacja podrzędna klas: ustępuje pierwszeństwa.

6.2.3.1. Funkcje class i isa

Do określenia typu pojedynczej zmiennej służy funkcja `class`, na przykład:

```
>> cotojest=pi;
>> cc=class(cotojest)
cc =
  1×6 char array
double
```

Faktycznie, zmienna `cotojest` jest liczbą typu `double` i jak wiadomo, ma wartość 3.141592653589793. Typ zmiennej podaje też polecenie `who` lub `whos`. Jednak używając `class`, można łatwo wpisać otrzymaną nazwę klasy do zmiennej (tutaj zmienna `cc`) i wykorzystać ją wewnątrz kodu tworzonego programu.

Funkcja `isa` sprawdza, czy badana zmienna należy do wskazanej klasy. Funkcja `isa` ma postać `isa(nazwa_zmiennej, 'nazwa_typu')` i przyjmuje wartość 1 (`TRUE`), jeśli typ zmiennej pokrywa się z typem podanym w wywołaniu tej funkcji. W poniższym przykładzie polecenie `obj=green(obj)` będzie wykonywane tylko wtedy, gdy funkcja `isa` potwierdzi, że `obj` jest obiektem należącym do klasy `ColorLampClass`.

```
obj = lampS;
if isa(obj,'ColorLampClass')
    obj=green(obj);
end
```

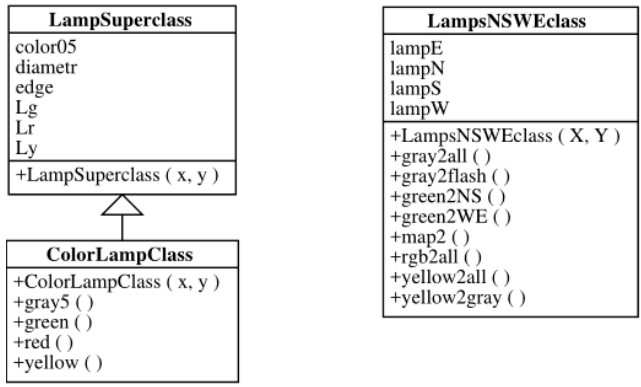
Obiekt ma dostęp tylko do metod, które zdefiniowano w jego klasie i w klasach nadrzędnych. Klasa `LampSuperclass` nie ma ani nie dziedziczy metody `green`. Jeśli obiekt `obj` należy do klasy, która nie obsługuje żądanej operacji, to efektem będzie błąd. Próba wykonania polecenia `obj=green(obj)` zakończy się wtedy komunikatem: *Undefined function 'green' for input arguments of type 'LampSuperclass '.*

6.3. Wizualizacja zmiany świateł drogowych

Poniżej opisano obiektowo zorientowaną wizualizację zmiany świateł sterujących ruchem pojazdów nadjeżdżających do skrzyżowania z kierunków północ N, południe S, zachód W i wschód E.

6.3.1. Synchronizacja pracy czterech sygnalizatorów

Klasa `LampsNSWEclass` definiuje wirtualny model czterech sygnalizatorów. Konstruktor tej klasy `LampsNSWEclass(X,Y)` tworzy obiekt, w którym zdefiniowano cztery odpowiednio rozmieszczone sygnalizatory należące do klasy `ColorLampClass`. Na rysunku 6.3.1-1 pokazano diagram klas dla omawianego programu.



Rysunek 6.3.1-1. *Diagram klas dla obiektowo zorientowanej wizualizacji zmiany świateł na skrzyżowaniu. Diagramy wykonano z użyciem programu UMLgui dostępnego na serwerze www.mathworks.com/matlabcentral/fileexchange/*

Przykład 6.3.1-1

Metoda `obj=LampsNSWEclass(X,Y)` jest konstruktorem tworzącym obiekt. Obiekt będzie nazwany `lampsAll`. Atrybutami obiektu są cztery sygnalizatory `lampN`, `LampS`, `lampW` i `LampE` (zadeklarowano je w bloku `properties`). Każdy z sygnalizatorów jest równocześnie obiektem należącym do klasy `ColorLampClass`. Metoda `gray2flash` uruchamia światła *żółte migające*, a metoda `rgb2all` odpowiednio zapala światła *zielone*, *żółte* i *czerwone*. Metoda `map2` tworzy uproszczoną mapę skrzyżowania. Pozostałe metody wspomagają zapalanie i gaszenie świateł.

```
classdef LampsNSWEclass    % needs / używa ColorLampClass<LampSuperclass
% Visualisation of 4 traffic lights / Wizualizacja zmiany 4 świateł drogowych
    properties
        lampN, lampS, lampW, lampE
    end

    methods
        function obj=LampsNSWEclass(X,Y)
            obj=map2(obj);          % draws simplified map / rysuje skrzyżowanie
            obj.lampN=ColorLampClass(X(1),Y(1));    % create / tworzy lampy
            obj.lampS=ColorLampClass(X(2),Y(2));
            obj.lampW=ColorLampClass(X(3),Y(3));
            obj.lampE=ColorLampClass(X(4),Y(4));
        end
        function obj=gray2flash(obj)    % flash yellow / żółte migające
            obj=yellow2all(obj);
            pause(.6)
            obj=yellow2gray(obj);
            pause(.3)
        end
        function obj=yellow2gray(obj)    % yellow off / gasi żółte
            set(obj.lampN.Ly,'FaceColor',obj.lampN.color05);
            set(obj.lampS.Ly,'FaceColor',obj.lampS.color05);
            set(obj.lampW.Ly,'FaceColor',obj.lampW.color05);
            set(obj.lampE.Ly,'FaceColor',obj.lampE.color05);
        end
         function obj=rgb2all(obj)
            %% green for NS on road crossing / zielone dla północ-południe
            obj=green2NS(obj);
            pause(2)
            %% yellow for All / żółte
            obj=yellow2all(obj);
```

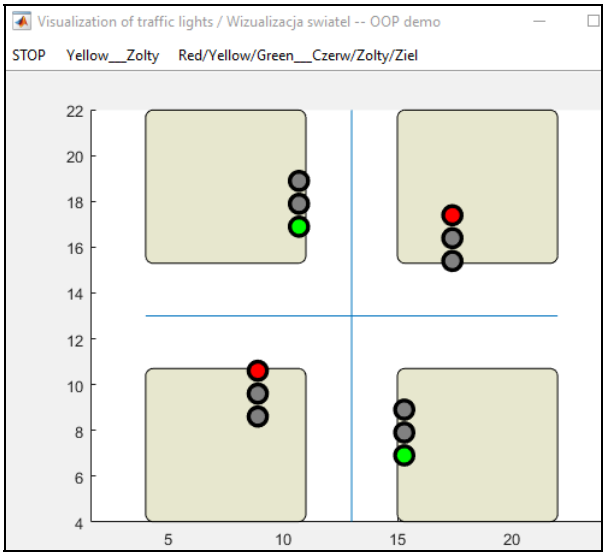
```
            pause(.5)
            %% green for WE on road crossing / zielone dla wschód-zachód
            obj=green2WE(obj);
            pause(2)
            %% yellow for All / żółte
            obj=yellow2all(obj);
            pause(.5)
        end

        function obj=gray2all(obj)    %% all lights off /gasi wszystkie lampy
            obj.lampN=gray(obj.lampN);  obj.lampS=gray(obj.lampS);
            obj.lampE=gray(obj.lampE);  obj.lampW=gray(obj.lampW);
        end

        function obj=green2NS(obj)    % NS Green / zielone dla północ-południe
            obj.lampN=red(obj.lampN);   obj.lampS=red(obj.lampS);
            obj.lampE=green(obj.lampE); obj.lampW=green(obj.lampW);
        end

        function obj=green2WE(obj)    % WE green / zielone dla wschód-zachód
            obj.lampN=green(obj.lampN); obj.lampS=green(obj.lampS);
            obj.lampE=red(obj.lampE);   obj.lampW=red(obj.lampW);
        end

        function obj=yellow2all(obj)    % yellow for All / żółte
            obj.lampN=yellow(obj.lampN); obj.lampS=yellow(obj.lampS);
            obj.lampE=yellow(obj.lampE); obj.lampW=yellow(obj.lampW);
        end

        function obj=map2(obj)
            % draws simplified map of roads / rysuje uproszczone skrzyżowanie
            g9=[0.9 0.9 0.8];
            rectangle('Position',[4,4,7,6.7],'FaceColor',g9,…
                'Curvature',0.1);
            rectangle('Position',[15,4,7,6.7],'FaceColor',g9,…
                'Curvature',0.1);
            rectangle('Position',[15,15.3,7,6.7],'FaceColor',g9,…
                'Curvature',0.1);
            rectangle('Position',[4,15.3,7,6.7],'FaceColor',g9,…
                'Curvature',0.1);
            line([4,22],[13,13],'LineStyle','-');
            line([13,13],[4,22],'LineStyle','-');
        end
    end
end
```

6.3.2. Skrypt uruchamiający wizualizację

Sygnalizatory są umieszczone w punktach określonych przez czteroelementowe wektory X i Y. Są one parametrami wejściowymi konstruktora. Mapa skrzyżowania (pokazana na rysunku 6.3.2-1) jest dostępna do pobrania wraz z całym programem pod adresem *https://www.mathworks.com/matlabcentral/fileexchange/?term=mrozek*.

Przykład 6.3.2-1

```
clear,close,
X=[14.9,10.3,8.5,17]; Y=[6.5,16.5,8.2,15];
%% define graphic window / definiuje okno graficzne
```



Rysunek 6.3.2-1. *Zestaw czterech sygnalizatorów to jeden obiekt LampsAll należący do klasy LampsNSWEclass. Lampy zostały umieszczone na mapie skrzyżowania*

```
set(gcf,'NumberTitle','off','MenuBar','none', 'Name',…
   'Visualization of traffic lights / Wizualizacja swiatel -- OOP demo')
%% control menu / menu do sterowania światłami
menu1=uimenu(gcf,'Label','STOP', 'Callback', 'cont=0;');    % STOP
menu1=uimenu(gcf,'Label','Yellow___Zolty', 'Callback', 'cont=1');
menu1=uimenu(gcf,'Label','Red/Yellow/Green___Czerw/Zolty/Ziel',…
'Callback','cont=2');
%% create object with 4 traffic lights / utwórz obiekt z 4 sygnalizatorami
lampsAll=LampsNSWEclass(X,Y); axis equal
%% start visualization in while loop / rozpoczyna wizualizację w pętli while
cont=2;        % start with Red_Yellow_Green / czerwone, zielone i żółte
while cont
    switch cont
        case 1     % yellow_flash / żółte migające
            lampsAll=gray2flash(lampsAll);
        case 2     % Red_Yellow_Green / czerwone, zielone i żółte
            lampsAll=rgb2all(lampsAll);
    end
end
disp ('STOP'), clear, close, return % STOP if cont == 0
```

Kompletny kod programu do wizualizacji zmiany świateł drogowych można pobrać z serwera *https://www.mathworks.com/matlabcentral/fileexchange/?term=mrozek*. Można go również odszukać po wpisaniu nazwiska autora `Mrozek` w oknie *Search* na stronie: *http://www.mathworks.com/*.

6.4. Przeciążanie funkcji i operatorów

Użytkownik ma możliwość modyfikowania funkcji i operatorów wbudowanych w MATLAB-a. W literaturze określenia takie jak **przedefiniowanie** operatora, funkcji lub metody oraz **przeciążenie** (ang. *overloading*) lub **przeładowanie** są używane zamiennie.

6.4.1. Przeciążanie w Control System Toolbox

Przeciążanie (przedefiniowanie) metod jest stosowane między innymi w bibliotekach toolbox. Przykładem może być klasa `numlti` (ang. *numerical linear time invariant*) zdefiniowana w *Control System Toolbox*. Potomkami superklasy `numlti` są trzy klasy (rysunek 11.4.2-1):

- `ss` (*state space*) — model przestrzeni stanu,
- `tf` (*transfer function*) — transmitancja,
- `zpk` (*zero-pole-gain*) — zera, bieguny, współczynnik wzmocnienia.

W każdej z klas potomnych są dostępne metody superklasy `numlti`. Ich listę uzyskuje się za pomocą polecenia `methods numlti` (*konieczne* jest zainstalowanie biblioteki *Control System Toolbox*).

Niektórych z metod klasy `numlti` (np. `bode`, `damp`, `impulse`) można używać bez jakiejkolwiek modyfikacji. Inne, na przykład `plus`, `minus`, `c2d`, `d2c`, muszą być dostosowane do specyfiki każdej z klas: `ss`, `tf`, `zpk`.

Można się o tym przekonać, analizując np. zestawienie metod zdefiniowanych dla klasy `tf`. Można je uzyskać za pomocą polecenia `methods tf`.

Metody wymagające modyfikacji zostały powtórnie zdefiniowane w klasie `tf` i będą przeładowane, czyli zmienione. Metoda `tf` jest konstruktorem klasy `tf`. Przeglądając plik *tf.m* umieszczony w podfolderze *toolbox\control\ctrlmodels\@tf*, można odszukać linie wskazujące bezpośrednio na klasę `numlti` jako przodka klasy `tf`.

```
classdef (InferiorClasses = {?
matlab.graphics.axis.Axes, ?
matlab.ui.control.UIAxes})...
                tf < numlti
```

6.4.2. Jednoznaczność wyboru operatora lub funkcji

Reguły przeszukiwania nazw gwarantują każdorazowo jednoznaczność wyboru operatora lub funkcji — spośród wszystkich aktualnie dostępnych, mających tę samą nazwę. Jeśli argumenty funkcji są numeryczne lub znakowe, to stosuje się standardową kolejność poszukiwania funkcji o danej nazwie: wpierw subfunkcje, potem funkcje prywatne, funkcje wbudowane, MEX-pliki, P-pliki itd. — jak opisano w podrozdziale 3.2.9.

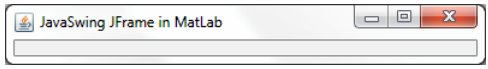
Jeśli przynajmniej jeden z argumentów funkcji jest obiektem, to ustalany jest obiekt najwyższy w hierarchii. Hierarchię ustala się w konstruktorach za pomocą słów kluczowych `superiorto/inferiorto`. Klasy obiektów wskazane jako `superiorto` będą miały pierwszeństwo użycia swoich metod i funkcji. Wskazane jako `inferiorto` ustępują pierwszeństwa metodom i funkcjom innego obiektu. Jeśli hierarchia nie została ustalona (obiekty są *równe*), to brany jest pierwszy (od lewej).

Dla wybranego obiektu sprawdzane jest, czy w określeniu jego klasy nie występuje metoda o danej nazwie. Jeśli tak, to jest ona wywoływana. Jeśli nie, to szuka się odpowiedniej metody na ewentualnych wyższych szczeblach hierarchii klas aż do skutku. Jeśli to się nie uda, stosuje się standardową kolejność poszukiwania.

6.5. Klasy języka Java

W omawianej wersji MATLAB-a biblioteki języka Java (pliki *.jar* i *.jarext*) są dostępne w folderze: *C:\Program Files\MATLAB\R2017a\java*.

Poniżej podano sposób uzyskania wykazu metod dla danej klasy (polecenie `methods`) oraz prosty przykład wykorzystania metod należących do klasy `javax.swing.JFrame` do utworzenia okna Javy w środowisku MATLAB (rysunek 6.5-1).



Rysunek 6.5-1. *Okno języka Java w środowisku MATLAB*

Przykład 6.5-1

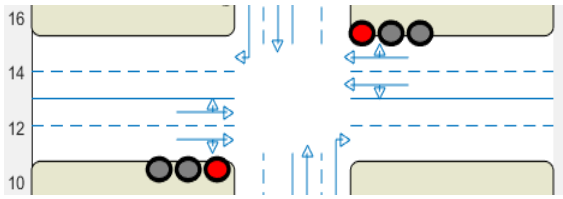
Uzyskanie wykazu metod dla wskazanej klasy i wygenerowanie okna Javy:

```
methods javax.swing.JFrame       % podaje metody klasy JFrame
%
box=javax.swing.JFrame(''); % tworzy obiekt 'box', klasa JFrame
box.setSize(400,50)          % metoda określa rozmiary box
box.setTitle('JavaSwing JFrame in MatLab') % nadanie tytułu
box.setLocation(200,160);    % określa położenie box na ekranie
box.setResizable(true);      % można zmieniać wymiary myszą
box.setVisible(true);    % metoda rysująca obiekt box na ekranie
```

Obiekty i metody języka Java mogą być też tworzone za pomocą poleceń: `javaArray`, `javaMethod`, `javaObject`, `javaMethodEDT`, `javaObjectEDT`, gdzie `EDT` oznacza *Swing Event Dispatch Thread*.

6.6. Zadania dla Czytelnika

1. Opracowanie funkcji `lampsAll=lampRotateFun(lampsAll)`, która dokona obrotu sygnalizatorów `lampsAll.lampW` i `lampsAll.lampE` z pozycji pionowej do pozycji poziomej, jak pokazano na rysunku 6.6-1. Funkcja będzie wywoływana z okna *Command*.



Rysunek 6.6-1. *Wskazówka do zadania 1. i 2.: poziome ustawienie sygnalizatorów*

2. Opracowanie dodatkowej metody `obj= lampRotate(obj)` dla klasy `LampsNSWEclass`, która dokona obrotu sygnalizatora `lampW` i sygnalizatora `lampE` do pozycji poziomej, jak pokazano na rysunku 6.6-1. Modyfikowanie innych klas nie jest dozwolone. Dozwolonym wariantem jest utworzenie podklasy zawierającej metodę do obracania sygnalizatora.
3. Przygotowanie obiektowo zorientowanego programu obliczającego wartość lokaty bankowej z uwzględnieniem oprocentowania, oparte na ofercie jednego z dużych banków krajowych.

Rozdział 7.

App Designer i GUI

Każdy obiekt graficzny w MATLAB-ie ma swój *uchwyt* (ang. *handle*), który jest *obiektem* (w MATLAB-ie R2014a i wcześniej uchwyt był liczbą). Metodyka tworzenia i obsługi obiektów graficznych w MATLAB-ie nosi nazwę *Handle Graphics*, po polsku *grafika uchwytów*. Pojęcia klasy i obiektu omawiano w rozdziale 6.

Zaprojektowanie interfejsu graficznego, czyli umieszczenie na rysunku suwaków, przycisków, okien edycyjnych i innych elementów *GUI* (ang. *Graphical User Interface*) nie jest trudne. Znacznym ułatwieniem jest możliwość użycia narzędzia *App Designer*. W poprzednim wydaniu książki omawiano starsze, mniej przyjazne narzędzie *GUIDE*.

7.1. Obiekty graficzne MATLAB-a

Obiektami graficznymi są *linie*, *powierzchnie*, *osie*, *teksty*, *okna graficzne* oraz *cały ekran*. *Podobne obiekty* (na przykład linie) należą do tej samej *klasy*. W klasie definiuje się *atrybuty* (nazwy i typy zmiennych, czyli parametrów obiektu) oraz *metody*, czyli funkcje, które może realizować każdy obiekt należący do danej klasy.

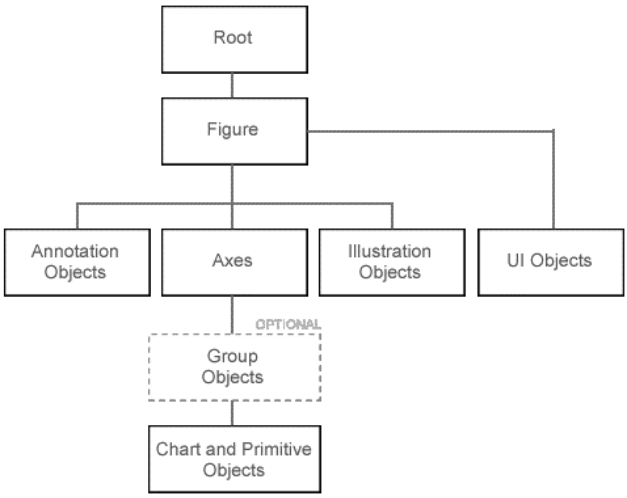
Obiekt przechowuje wartości swoich atrybutów. *Atrybuty* określają *aktualny stan i właściwości obiektu*, na przykład długość, rodzaj i kolor linii oraz jej położenie. MATLAB umożliwia modyfikowanie elementów już wykonanego rysunku poprzez zmianę wartości atrybutów obiektów graficznych.

7.1.1. Hierarchia obiektów graficznych

Obiekty graficzne tworzą strukturę hierarchiczną. Ekran komputera to obiekt `Root`, czasem określany jako `groot`. Każde okno graficzne to obiekt należący do klasy `Figure` (rysunek 7.1.1-1).

Potomkami klasy `Figure` mogą być:

- `Axes` — kontener, czyli klasa będąca pojemnikiem na atrybuty osi współrzędnych x, y, z (określają one położenie obiektów umieszczonych w oknie `Figure`) oraz na atrybuty innych obiektów, np. punktów, linii itp.,
- `UI Objects` to kontener na elementy interfejsu graficznego użytkownika. Są to różne przyciski, suwaki, rozwijane menu, okna dialogowe i okna tekstowe,



Rysunek 7.1.1-1. *Hierarchia obiektów graficznych — published with permission of The Mathworks, Inc.*

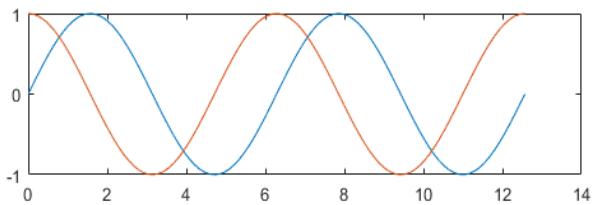
- `Annotation Object` to linie, linie ze strzałką lub strzałką podwójną, prostokąty i elipsy oraz teksty; jeśli atrybut `interpreter` ma wartość `latex`, to można w tekście umieścić znaki alfabetu greckiego oraz symbole matematyczne,
- `Illustration Object` to skala barw lub legenda (ang. *Colorbar*, *legend*).

Poleceniem utworzenia obiektu graficznego może być każde polecenie MATLAB-a, które otwiera okno graficzne bądź wykonuje dowolny fragment rysunku. Nazwy klas obiektów są identyczne z nazwami poleceń utworzenia danego obiektu. Rezultatem takiego polecenia, oprócz utworzenia nowego obiektu graficznego, jest przypisanie mu uchwytu (ang. *handle*). Uchwyt może być przypisany do zmiennej podczas tworzenia tego obiektu.

Przykład 7.1.1-1

W poniższym fragmencie programu do zmiennej `h1` przypisano uchwyt do wykresu utworzonego przez funkcję `plot`.

```
clear, close
x=linspace(0,pi*4);              % generuje wektor 1x100
h1=plot(x,sin(x),x,cos(x));      % dwa wykresy, rysunek 7.1.1-2
```



Rysunek 7.1.1-2. *Do zmiennej h1 przypisano uchwyt wykresu utworzonego przez plot*

Zmienne utworzone w przestrzeni roboczej MATLAB-a mogą być wyświetlone przez polecenie `whos`.

```
Name        Size     Bytes  Class      Attributes
h1          2x1         16  matlab.graphics.chart.primitive.Line
x           1x100      800  double
```

Zmienna `h1` jest uchwytem i ma wymiar `2x1`, gdyż przypisano ją do dwóch wykresów: sin(*x*) i cos(*x*). Oba uchwyty `h1(1)` i `h1(2)` są obiektami typu `Line` należącymi do klasy `matlab.graphics.chart.primitive.Line`. Faktycznie, po wpisaniu nazwy `h1` jako polecenia w oknie `Command` na ekranie wyświetli się informacja o uchwytach `h1(1)` i `h1(2)`:

```
>> h1
h1 =
  2×1 Line array:
  Line
  Line
```

7.1.2. Atrybuty obiektu graficznego

Nazwy i liczba atrybutów (inaczej *pól*) obiektu są dostosowane do potrzeb. Atrybuty określają kolor, wielkość, położenie i inne właściwości obiektu.

Wpisanie nazwy pierwszego uchwytu `h1(1)` jako polecenia spowoduje wyświetlenie na ekranie atrybutów pierwszego wykresu z przykładu 7.1.1-1, czyli sin(*x*).

```
>> h1(1)
ans =
  Line with properties:
              Color: [0 0.4470 0.7410]
          LineStyle: '-'
          LineWidth: 0.5000
             Marker: 'none'
         MarkerSize: 6
    MarkerFaceColor: 'none'
              XData: [1×100 double]
              YData: [1×100 double]
              ZData: [1×0 double]
```

Pokazane w podrozdziale 4.3.3 interaktywne edytowanie rysunków polega na odpowiedniej zmianie tych atrybutów. Przykładowo:

- atrybut `h1(1).Color` określa barwę linii tego wykresu jako RGB `[0  0.4470  0.7410]`,
- atrybut `h1(1).LineWidth` określa grubość tej linii,
- atrybuty `h1(1).XData` i `h1(1).YData` są wektorami `1×100 double` i określają współrzędne punktów, przez które przebiega linia wykresu.

Uchwyt `h1(2)` daje dostęp do atrybutów drugiego wykresu z przykładu 7.1.1-1, czyli cos(*x*).

Obiekt jest *aktualny*, jeśli był używany jako ostatni. *Kliknięcie* myszą *wybranego obiektu* (np. wykresu) powoduje, że *staje się on aktualny*. Funkcje `gcf`, `gca` i `gco` pobierają uchwyty aktualnych obiektów:

- `gcf` — pobiera uchwyt aktualnego okna (ang. *get current figure*),
- `gca` — pobiera uchwyt aktualnego układu osi współrzędnych (ang. *axes*),
- `gco` — pobiera uchwyt aktualnego obiektu (ang. *object*),
- `0` (zero) — jest uchwytem obiektu `Root`, czyli całego ekranu.

Funkcja (`get(`*`handle`*`)`) pobiera, a (`set(`*`handle`*`)`) zmienia atrybuty obiektu wskazanego przez *handle*, czyli uchwyt. Dla aktualnego okna graficznego można napisać `get(gcf)`.

Podobnie dla aktualnego układu współrzędnych lub dla aktualnego obiektu stosuje się odpowiednio polecenia: `get(gca)`, `get(gco)`.

Do każdego obiektu można się odwołać, podając zmienną zawierającą jego uchwyt (np. `h1(1)`) lub odpowiednią funkcję (np. `gca`), która pobierze taki uchwyt.

Przykładowe operacje na obiektach *Handle Graphics*:

- `get`, `set` — pobiera lub ustawia parametry obiektu,
- `reset` — przywraca domyślne parametry obiektu,
- `delete` — usuwa obiekt,
- `gcbo` — uchwyt dla *callback* z aktualnego obiektu (ang. *object*),
- `gcbf` — uchwyt dla *callback* z aktualnego okna graficznego (*figure*),
- `drawnow` — wykonuje operacje graficzne, które były wstrzymane,
- `findobj` — poszukuje obiektu o zadanych parametrach,
- `copyobj` — tworzy duplikat obiektu i jego potomków,
- `allchild` — podaje uchwyty potomków.

Polecenie `get(0)` podaje atrybuty przypisane do całego ekranu komputera. Jednym z jego parametrów jest rozdzielczość `ScreenSize`, która może mieć wartość `[1  1 1280  800]`.

7.1.3. Zmiana atrybutów modyfikuje obiekty graficzne

Wskazanym atrybutom obiektu można nadać nową wartość, stosując funkcję w postaci: `set(`*`handle, 'nazwa_pola', wartość`*`)`.

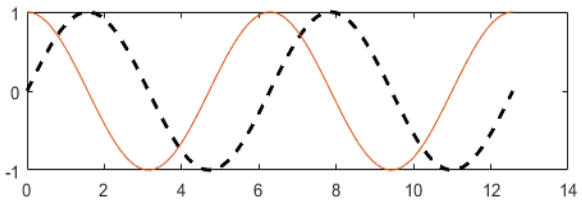
Zmianę koloru na czarny w obiekcie z uchwytem `h1(1)` wykonuje polecenie:

```
set(h1(1), 'Color', 'k')
```

W ten sposób kolor linii pierwszego wykresu w dolnej części rysunku 7.1.1-2 zmienia się na czarny (ang. *blacK*, litera *b* jest ewentualnie używana dla koloru niebieskiego, ang. *blue*). Liczba par *'nazwa_pola', wartość* w funkcji `set` nie jest ograniczona i można jednocześnie ustawić więcej atrybutów, zmieniając przykładowo kolor, grubość linii i rodzaj linii.

```
set(h1(1), 'Color', 'k', 'LineWidth',2.0, 'LineStyle','--')
```

Rezultat wszystkich zmian pokazano na rysunku 7.1.3-1. Wykres funkcji sinus jest czarny, linia jest przerywana i pogrubiona. Jeśli uchwyt modyfikowanego elementu nie jest znany, można kliknąć wybrany obiekt (aby stał się aktualny) i użyć `gco` lub `gca` zamiast uchwytu.



Rysunek 7.1.3-1. *Modyfikowanie wykresu: pogrubienie, zmiana koloru i typu linii*

Atrybuty obiektów w postaci *'nazwa_pola', wartość* mogą być dopisywane do wielu innych poleceń, np.:

```
>> x=linspace(0,pi*4); plot(x,cos(x),'LineWidth',3)
```

7.2. App Designer, tworzenie aplikacji z GUI

Zaprojektowanie interfejsu graficznego, czyli umieszczenie na rysunku suwaków, przycisków, okien edycyjnych i innych elementów GUI, nie jest trudne. Znacznym ułatwieniem jest możliwość użycia dołączonego do MATLAB-a narzędzia *App Designer*. Podane dalej przykłady pomogą zaprojektować własny interfejs graficzny, dostosowany do potrzeb konkretnego programu i wymaganego scenariusza obliczeń.

Elementy interfejsu graficznego GUI to suwaki, okienka dialogowe oraz różnego rodzaju przyciski i paski narzędziowe.

Aplikacja MATLAB-a to program, który wykorzystuje interfejs graficzny GUI do pobrania danych, sterowania przetwarzaniem danych oraz do prezentacji rezultatów obliczeń. Aplikacje są zawarte w wielu produktach MATLAB-a, można ją łatwo wyszukać i zainstalować (podrozdziały 1.2.2 i 7.5). Można też tworzyć własne aplikacje, jak opisano w podrozdziałach 7.2 – 7.4. *App Designer* jest nowym, łatwym w użytkowaniu środowiskiem do tworzenia interaktywnych aplikacji. Umożliwia on projektowanie i implementację własnych interfejsów graficznych z użyciem elementów obiektowo zorientowanej grafiki uchwytów (ang. *Handle Graphics*) oraz wywołań zwrotnych (ang. *callbacks*).

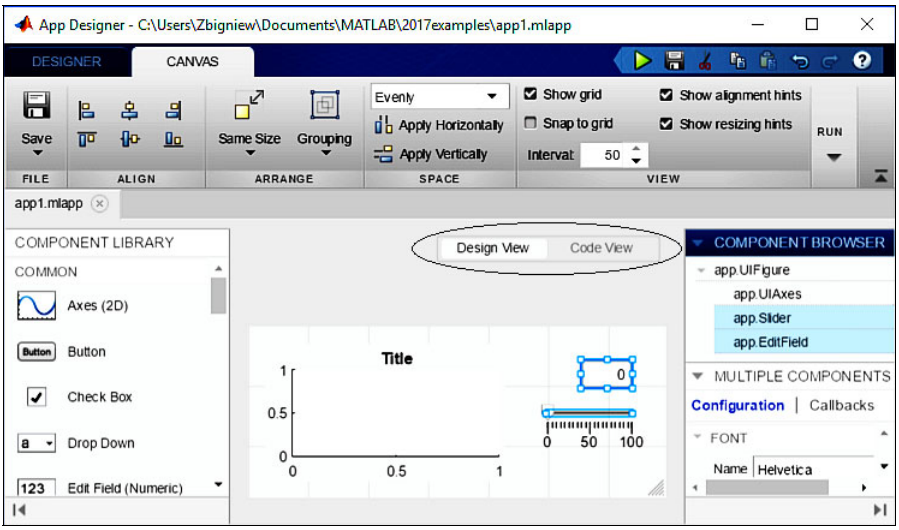
Stosowanie narzędzia *App Designer* daje dużą oszczędność czasu oraz zapewnia wysoki komfort i precyzję pracy w budowaniu interaktywnego środowiska graficznego. *App Designer* jest nieodpłatnie dołączany do MATLAB-a.

7.2.1. Panel App Designer

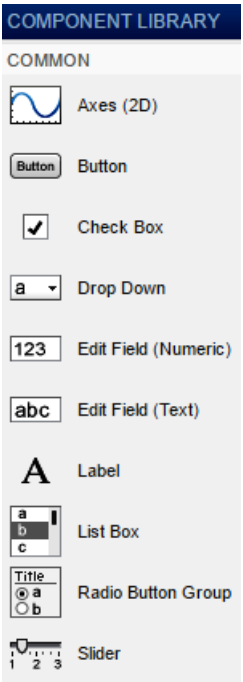
Po wywołaniu `>>appdesigner` otwiera się panel przeznaczony na utworzenie nowego projektu (rysunek 7.2.1-1). W jego centralnej części, znajduje się duże okno, które może pracować w dwu trybach. W trybie *Design View* jest to okno graficzne *UIFigure* i umieszcza się w nim ikony pobrane myszką z *Component Library,* czyli z lewej strony panelu. W trybie *Code View*, w centralnym oknie jest wyświetlany kod, omawiany w podrozdziale 7.2.2.

Elementy interfejsu graficznego GUI (rysunki 7.2.1-2, 7.2.1-3) pobiera się myszą z widocznej po lewej stronie biblioteki *Component Library.* Służą one do budowy konsoli do interaktywnego sterowania przebiegiem obliczeń, symulacji lub pomiarów. Używane tu okno *UIFigure* i osie współrzędnych różnią się od wcześnie omówionych obiektów *Figure* i *Axes.*

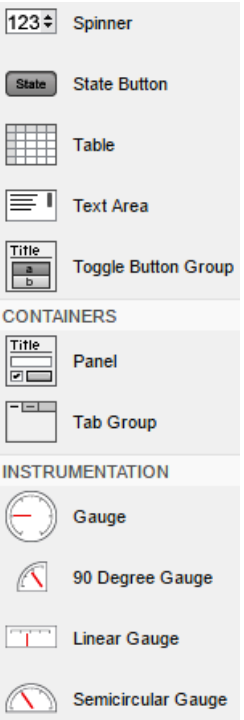
Ikony grupy *ALIGN* są widoczne i aktywne tylko przy równoczesnym zaznaczeniu co najmniej dwóch elementów GUI, np. suwaka *app.Slider* i okienka edycyjnego *app.EditField*, jak pokazano na rysunku 7.2.1-1.



Rysunek 7.2.1-1. *Panel App Designera — środowisko do tworzenia interaktywnych aplikacji*



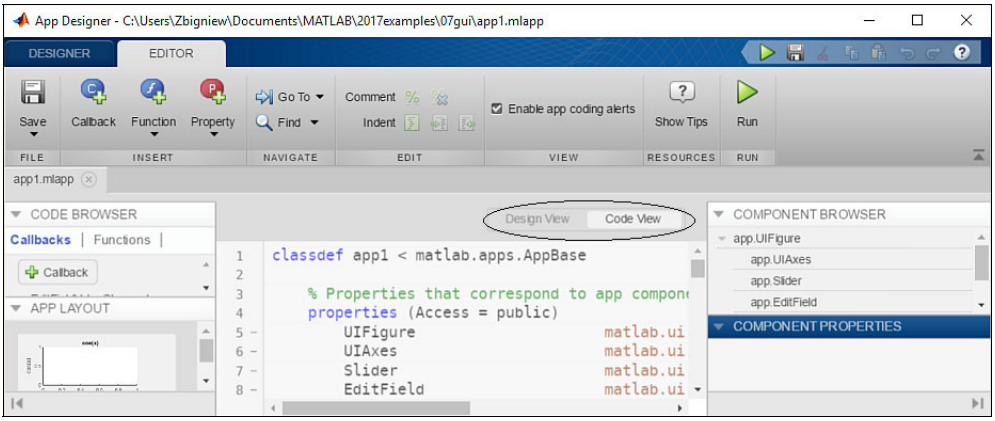
Rysunek 7.2.1-2. *Biblioteka Component Library, elementy GUI należące do grupy COMMON*



Rysunek 7.2.1-3. *Biblioteka Component Library, elementy GUI należące do grupy COMMON, CONTAINERS i INSTRUMENTATION*

7.2.2. Automatyczne generowanie kodu aplikacji

App Designer automatycznie generuje oprogramowanie do obsługi elementów graficznych pobranych z *Component Library*. Nie jest to jednak kompletny program, gdyż *App Designer* nie może odgadnąć ani zaprogramować wzajemnych relacji pomiędzy tymi elementami. Aktualny program jest widoczny po kliknięciu w centralnym oknie przycisku *Code View*, powyżej centralnego okna graficznego. Dodatkowo nazwa zakładki *CANVAS* zmienia się na *EDITOR* (rysunek 7.2.2-1).



Rysunek 7.2.2-1. *Panel App Designera w trybie Code View*

7.2.3. Wywołania zwrotne, dodatkowe funkcje i zmienne

Użytkownik może dopisać i edytować *wywołania zwrotne* oraz dodatkowe *funkcje* i *zmienne* — tylko w przewidzianych do tego miejscach programu wygenerowanego dla użytych elementów GUI.

Pozostała część automatycznie generowanego kodu jest wyświetlana *na szarym tle* i *nie może być edytowana*. Zwiększa to pewność działania kodu, gdyż jego istotne fragmenty nie mogą być przypadkowo zmienione przez użytkownika.

Wywołania zwrotne służą do automatycznego wywołania odpowiednich poleceń, funkcji lub metod w odpowiedzi na zdarzenia (ang. *event*). Przykładem zdarzenia jest kliknięcie myszą lub ustawienie nowej wartości na suwaku.

App Designer proponuje miejsce wstawienia wywołania zwrotnego i jego kod dla każdego elementu GUI. Rolą użytkownika jest modyfikacja i ewentualne zaakceptowanie tych wywołań oraz wpisanie nazw i parametrów funkcji, które będą wykonane w odpowiedzi na zdarzenia przewidziane w projekcie aplikacji. Wywołania zwrotne opisano w podrozdziale 7.4.3.

7.3. Przykład — interaktywny wykres cos(x)

Tworzona aplikacja ma wykonać wykres funkcji cos(*x*) dla wartości *x* zmieniających się od zera do wartości `xMax`, ustawianej suwakiem. Odczytana z suwaka wartość `xMax` ma być wyświetlona obok suwaka.

Scenariusz

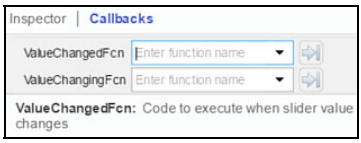
Tworzona aplikacja powinna wykonać następujące czynności:

1. Odczytać wartość ustawioną na suwaku, czyli na obiekcie *app.Slider*.
2. Wykonać wykres w oknie *UIAxes*.
3. Wyświetlić wartość `xMax` w okienku *app.EditField*, obok suwaka.
4. Ewentualnie wykonać odpowiednie czynności po wpisaniu nowej wartości `xMax` bezpośrednio z klawiatury do okienka *app.EditField*.
5. Zamknąć program po kliknięciu przycisku *Exit*.

7.3.1. Odczytanie wartości po zmianie położenia suwaka

App Designer używa mechanizmu *callback*, aby odpowiednio zareagować na zdarzenia zewnętrzne, np. ustawienie nowej wartości na suwaku lub wpisanie danych w oknie edycyjnym. Aby utworzyć *callback*, który odczyta nowo ustawioną wartość na suwaku, należy rozmieścić elementy GUI jak na rysunku 7.2.1-1 i wykonać poniższe czynności:

- Kliknąć prawym przyciskiem myszy ikonę suwaka *Slider* (lub linię *app.Slider* w oknie *COMPONENT BROWSER*).
- Poniżej *COMPONENT BROWSER* pokaże się nagłówek *SLIDER PROPERTIES* i zakładki. Należy wybrać *Callbacks/ValueChangedFcn callback* (nie pomylić z *ValueChangingFcn*) (rysunek 7.3.1-1).



Rysunek 7.3.1-1. *Wybór ValueChangedFcn callback z menu kontekstowego*

Po automatycznym przełączeniu centralnego okna z trybu *Design View* w tryb *Code View* wyświetlany jest fragment automatycznie wygenerowanego kodu (rysunek 7.3.1-2) z funkcją `SliderValueChanged`. Część wygenerowanego kodu jest wyświetlona na szarym tle i nie może być edytowana. Jedynie linia `value = app.Slider.Value;` może być zmieniona, usunięta lub zastąpiona dowolną liczbą własnych linii kodu.

Nazwa zmiennej `value` będzie zmieniona na `xMax`, gdyż suwak *app.Slider* służy do ustawienia wartości maksymalnej zmiennej `x` na wykresie. Nowy kod ma następującą postać.

```
xMax = app.Slider.Value;
```

```
% Value changed function: Slider
function SliderValueChanged(app, event)
    value = app.Slider.Value;

end
```

Rysunek 7.3.1-2. *Funkcja SliderValueChanged została dodana przez App Designer w celu obsługi zmiany ustawienia suwaka app.slider*

7.3.2. Wyświetlenie wartości w okienku edycyjnym

Kolejnym zadaniem jest przygotowanie wywołania zwrotnego do wyświetlania wartości `xMax` w okienku edycyjnym *EditField*. Po kliknięciu *app.EditField* w oknie *COMPONENT BROWSER* pojawi się okno *EDIT FIELD PROPERTIES*, w którym należy wybrać *Callbacks*. Następnie, po kliknięciu ikony ▼, która znajduje się poniżej, należy wybrać *<add ValueChangedFcn Callback>*. W rezultacie w centralnej części panelu *App Designer* pojawi się nowa funkcja.

```
% Value changed function: EditField
function EditFieldValueChanged(app, event)
value = app.EditField.Value;
end
```

Edytować można tylko linię `value = app.EditField.Value;`. Chwilowo nie jest to potrzebne. Natomiast użyteczna jest nazwa `app.EditFie.Valueld`.

Polecenie:

```
app.EditField.Value= xMax;    % wpisze xMax do pola Value w app.EditField
```

przypisze wartość `xMax` do zmiennej `app.EditField.Value`, a efektem tego będzie wyświetlenie tej wartości w okienku edycyjnym. Powyższa linia będzie wpisana do `function SliderValueChanged`.

7.3.3. Wykonanie wykresu w oknie graficznym

Kolejnym zadaniem jest wykonanie wykresu w oknie *UIAxes*. W tym celu będzie użyte:

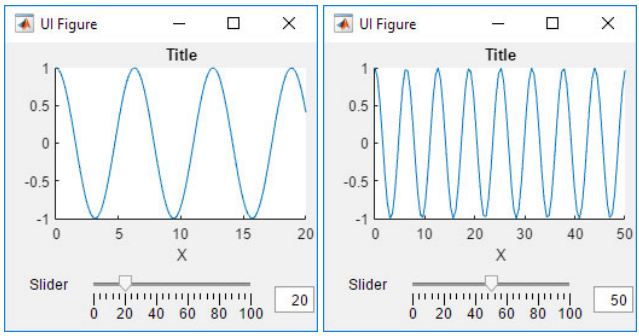
```
x=linspace(0, xMax);          % tablicowanie zmiennej x
plot(app.UIAxes,x,cos(x))     % wykonanie wykresu w oknie UIAxes
```

Pierwszym parametrem funkcji `plot` jest uchwyt okna *app.UIAxes*, co przekieruje wykres funkcji cos(*x*) do tego okna.

Opisane powyżej działania, to jest odczyt wartości `xMax` ustawionej na suwaku, wpisanie jej do okienka i wykonanie nowego wykresu ma być powtarzane po każdorazowej zmianie ustawienia suwaka. Z tego powodu wszystkie omówione wyżej polecenia wpisano do `function SliderValueChanged`.

```
% Value changed function: Slider
function SliderValueChanged(app, event)
xMax = app.Slider.Value;      % przypisanie nowej wartości do xMax
app.EditField.Value=xMax;     % wpisanie nowej wartości xMax do okienka
x=linspace(0,xMax);           % tablicowanie zmiennej x
plot(app.UIAxes,x,cos(x))     % wykonanie wykresu
end
```

Na rysunku 7.3.3-1 pokazano wykresy wykonane przez opisaną wyżej aplikację. Można ją uruchomić, klikając zielony trójkącik *Run* w oknie *App Designer* lub wpisując `app1` nazwę aplikacji w oknie *Command Window* MATLAB-a.



Rysunek 7.3.3-1. *Wykresy są wykonywane po każdorazowej zmianie pozycji suwaka*

7.3.4. Nowy wykres po zmianie w oknie edycyjnym

Jeśli nowa wartości `xMax` będzie wpisana do okienka *app.EditField* bezpośrednio z klawiatury, to odpowiedni callback wywoła funkcję `EditFieldValueChanged`. Do tej funkcji należy wpisać dodatkowe linie, aby otrzymać kod podany poniżej.

```
% Value changed function: EditField
function EditFieldValueChanged(app, event)
xMax = app.EditField.Value;
app.Slider.Value=xMax
x=linspace(0, xMax, 512);    % tablicowanie zmiennej x
plot(app.UIAxes,x,cos(x))    % wykonanie wykresu
end
```

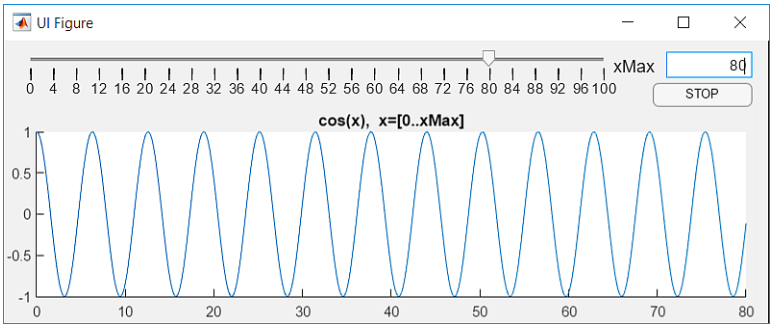
7.3.5. Zamknięcie okna aplikacji — przycisk STOP

W oknie *CANVAS* umieszczono przycisk *Button* po kliknięciu *app.Button* w *COMPPONENT BROWSER*. Następnie w *Button Properties* wybrano *Configuration* — w polu *Text* wpisano `STOP`. Dodatkowym efektem była *zmiana nazwy* przycisku na *app.STOPButton*. Następnie w *Callbacks* wybrano opcję utworzenia funkcji `STOPButtonPushed`. W pustej linii wpisano polecenie `delete(app)`.

```
% Button pushed function: STOPButton
function STOPButtonPushed(app, event)
delete(app)
end
```

7.3.6. Wersja końcowa aplikacji

Po sprawdzeniu zgodności zbudowanej aplikacji ze wszystkimi wymaganiami scenariusza skorygowano rozmieszczenie elementów GUI. Ostateczny efekt pokazano na rysunku 7.3.6-1.



Rysunek 7.3.6-1. *Wersja końcowa aplikacji*

7.4. App Designer — interaktywna wizualizacja rozwiązań równań różniczkowych

W porównaniu z poprzednim przykładem wizualizacja tłumienia drgań harmonicznych wymaga rozwiązywania równań różniczkowych po każdej zmianie wartości tłumienia oraz przekazywania aktualnej wartości współczynnika tłumienia do funkcji obliczającej pochodne.

7.4.1. Przykład — wizualizacja drgań z uwzględnieniem zmiany tłumienia

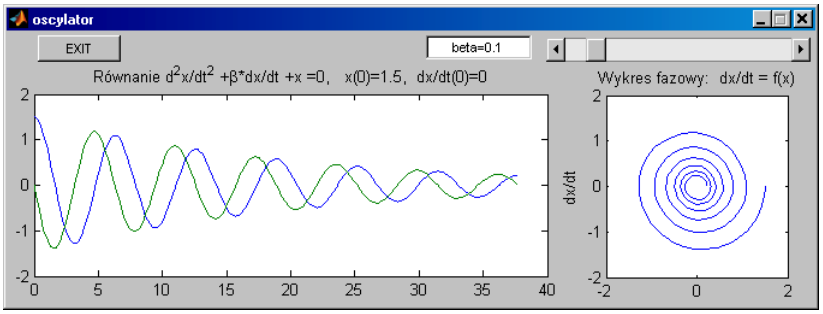
Układ drgający z dwoma elementami magazynującymi energię (na przykład ciężarek zawieszony na sprężynie) jest opisany przez równanie różniczkowe II rzędu.

$$\ddot{x}+\beta\dot{x}+x=0 \qquad (7.4.1\text{-}1)$$

Chcemy, aby w oknie podobnym do pokazanego na rysunku 7.4.1-1 dla ustawionej na suwaku wartości parametru β i dla warunków początkowych $x(0)=1.5$ $\dot{x}(0)=0$ pokazano wykresy rozwiązania równania różniczkowego 7.4.1-1 dla czasu $t=[0,12\cdot\pi]$. Po lewej stronie (w dużym okienku) umieszczono dwa wykresy: *przebieg czasowy rozwiązania x(t)* i jego *pierwszej pochodnej dx/dt* w funkcji czasu *t*. Po prawej stronie umieszczono dodatkowe kwadratowe okno zawierające *wykres fazowy*, czyli zależność wartości pochodnej *dx/dt* od wartości zmiennej *x*. W górnej części okna umieszczono elementy interfejsu GUI: *przycisk EXIT*, *okno* podające aktualną wartość współczynnika tłumienia β oraz *suwak* przeznaczony do wygodnej zmiany wartości tego współczynnika.

Przykład 7.4.1-1

Równanie różniczkowe II rzędu (równanie 7.4.1-1), opisujące drgania harmoniczne, przekształcono na dwa równania różniczkowe I rzędu w sposób pokazany w podrozdziale 8.3.1. Zapisano je jako `function xdot=mech_ode(t,x)`, w postaci wymaganej przez `ode23` i inne solvery rozwiązujące równania różniczkowe (patrz podrozdział 8.3.2).



Rysunek 7.4.1-1. *Przykład wizualizacji rozwiązania równania różniczkowego. Interfejs GUI tworzą: przycisk Exit, okno edycyjne i suwak. Użyto narzędzia GUIDE [Mrozek, 2010]*

```
function xdot=mech_ode(t,x)
% model układu drgającego II rzędu
% dx2/dt2 + beta*x + x = 0
% mech_ode jest wielokrotnie wywoływane przez solver ode23

xdot=zeros(2,1);                % zerowy wektor dwuelementowy
xdot(1)= x(2);                  % x(1)= x, x(2)= dx/dt
xdot(2)= -1*x(1) -beta*x(2);
```

Należy zwrócić uwagę, że zmienna `beta` (tłumienie) wykorzystana w ostatniej linii kodu nie jest parametrem wejściowym funkcji `mech_ode(t,x)`. Wartość `beta` nie jest znana podczas kodowania, bo będzie określana położeniem suwaka w oknie graficznym. W przykładzie 7.4.3.2-1 wartość zmiennej `beta` będzie przekazywana wewnątrz *funkcji zagnieżdżonej*.

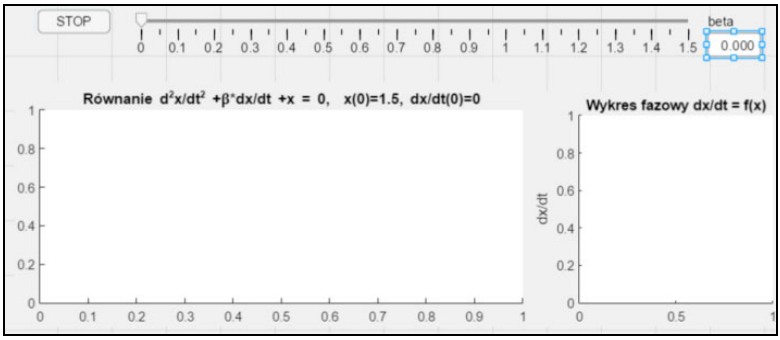
7.4.2. Scenariusz i aranżacja obiektów graficznych

Każda zmiana położenia suwaka powinna zapoczątkować wykonanie czynności podanych w scenariuszu poniżej:

1. Odczytanie z suwaka nowej wartości parametru `beta`.
2. Wpisanie aktualnej wartości `beta` do okienka edycyjnego umieszczonego obok suwaka.
3. Rozwiązanie równania różniczkowego zadanego przez `function xdot=mech_ode(t,x)` z podrozdziału 7.4.1.
4. Wykonanie dwóch wykresów dla aktualnych danych.
5. Proponowane rozszerzenie scenariusza — wpisanie z klawiatury nowej zawartości do okienka edycyjnego powinno spowodować:
   a) przesunięcie suwaka do właściwego położenia,
   b) realizację punktów 3. i 4. scenariusza.

Jeśli wpisana do okienka liczba jest poza zakresem ustawień suwaka, to należy wypisać odpowiedni komunikat i przejść do punktów a) i b) dopiero po skorygowaniu tej wartości.

Pierwszym krokiem jest uruchomienie MATLAB-a i wywołanie narzędzia *App Designer*. Na rysunku 7.4.2-1 pokazano, jak należy rozmieścić elementy GUI w centralnym oknie graficznym (*App Designer*, zakładka *CANVAS*).



Rysunek 7.4.2-1. *Rozmieszczenie elementów GUI w oknie App Designer CANVAS*

7.4.3. Kodowanie wywołań zwrotnych i wykresów

Po rozmieszczeniu w oknie *App Designer CANVAS* elementów GUI (zgodnie z rysunkiem 7.4.2-1) należy utworzyć wywołania zwrotne *callback* do obsługi tych elementów. Sposób utworzenia wywołań zwrotnych *callback* dla suwaka *app.Slider*, dla okienka edycyjnego *app.EditField* i dla przycisku *app.STOPButton* omówiono w podrozdziale 7.3. W ten sposób będą utworzone funkcje `SliderValueChanged`, `EditFieldValueChanged`, `STOPButtonPushed`. W podrozdziale 7.3.5 do funkcji `STOPButtonPushed` dodano linię `delete(app)`. W rezultacie kliknięcie przycisku STOP zakończy pracę aplikacji i zamknie jej okno.

7.4.3.1. Obsługa suwaka i okienka edycyjnego dla parametru beta

Funkcje `SliderValueChanged` i `EditFieldValueChanged` są uruchamiane odpowiednio po poruszeniu suwakiem lub wpisaniu i zatwierdzeniu nowej wartości w okienku edycyjnym.

Kod wewnątrz tych funkcji należy dostosować do zadań umieszczonych w scenariuszu. Jednak nie można przy tym zmienić ani *nazwy*, ani *parametrów* tych funkcji W pierwszej kolejności nowa wartość *beta* pobrana przez pierwszy obiekt (np. suwak) jest przekazywana do drugiego obiektu (np. okienko edycyjne), co skutkuje uaktualnieniem (synchronizacją) wartości wskazywanej przez suwak oraz wyświetlanej w okienku edycyjnym. Jak to zrobić, pokazano w podrozdziałach 7.3.3 i 7.3.4. Szkielet tych funkcji został wygenerowany przez *App Designera* w sposób opisany w podrozdziałach 7.3.1 i 7.3.2.

7.4.3.2. Wykonanie wykresów, funkcje DoPlots i nested_ode

Zadaniem `DoPlots` jest wykonanie nowych wykresów po każdej zmianie wartości `beta`. Funkcja `DoPlots` będzie wywoływana każdorazowo po zakończeniu obsługi suwaka i okienka edycyjnego.

Przykład 7.4.3.2-1

Funkcje `DoPlots` i `nested_ode`. Z uwagi na wykorzystanie `App Designera` pierwszym parametrem funkcji `DoPlots` musi być *app*.

```
function DoPlots(app,beta)
t0=0; t2=12*pi;      % czas rozpoczęcia i zakończenia obliczeń
x0= [1.5 0];         % wartości początkowe dla t=0
[t,x]=nested_ode(app,[t0,t2], x0,beta);
```

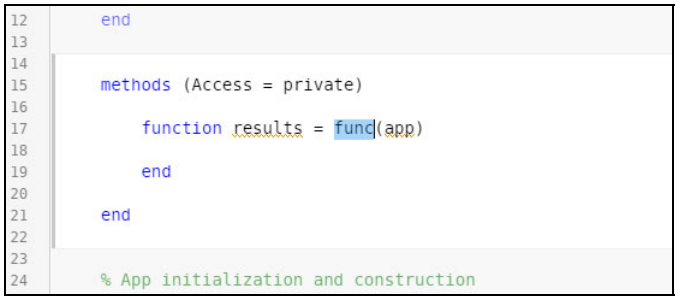
```
plot(app.UIAxes,t,x(:,1),'r',t,x(:,2),'k--')
legend(app.UIAxes,'x(t)','dx/dt');
plot(app.UIAxes2,x(:,1),x(:,2))
end

function [t,x]=nested_ode(app,tspan, x0,beta);
[t,x] = ode23 (@mech_ode, tspan, x0); %Runge-Kutta solver
    function xdot=mech_ode(t,x);       %nested in nested_ode
        % mech_ode.m  to model układu drugiego rzędu
        % dx2/dt2 + beta*x' + x = 0
        % handle @mech_ode jest wielokrotnie wołane przez ode23
        xdot=zeros(2,1);   % zerowy wektor dwuelementowy
        xdot(1)=x(2);      % x(1)= x  x(2)= dx/dt
        xdot(2)= -1*x(1) -beta*x(2);
    end              % internal nested function end
end                  % external nested function end
```

Kluczową rolę w obliczeniu danych do obu wykresów odgrywa funkcja `nested_ode` i zagnieżdżona w niej funkcja `mech_ode`. Zagnieżdżenie powoduje, że funkcje te mają wspólną przestrzeń roboczą, a w szczególności wartość tłumienia `beta`, wprowadzona jako parametr wejściowy funkcji `nested_ode`, może być wykorzystana wewnątrz funkcji `xdot=mech_ode(t,x)`.

Funkcja `DoPlots(app,beta)` przypisuje wartości zmiennym `t0, t2, x0`, wywołuje funkcję `nested_ode`, która korzystając z `mech_ode` i odpowiedniego solvera (np. `ode23`), oblicza rozwiązanie równania 7.4.1-1 w postaci wektora czasu *t* i macierzy dwukolumnowej *x*, po czym wykonuje dwa wykresy w obszarze wskazanym przez odpowiednie uchwyty osi współrzędnych: *app.UIAxes* i *app.UIAxes2*.

7.4.3.3. Umieszczenie własnych funkcji w wygenerowanym kodzie

Edytowalny obszar do umieszczenia własnych funkcji można utworzyć w trybie *Code View*, klikając zakładkę *Functions* w *CODE BROWSER*, następnie ramkę *+Function / Private Function*. W efekcie w *CODE BROWSER* pojawia się nowa funkcja `func(app,…)`. Funkcję tę można odszukać samodzielnie w oknie *Code View*, można też ją zlokalizować, klikając `func1(app,…)`. Jest ona edytowalna i ma postać pokazaną na rysunku 7.4.3.3-1.



Rysunek 7.4.3.3-1. *Okno Code View w App Designer. Brak szarego tła oznacza, że funkcja prywatna results=func(app) może być edytowana, a nawet usunięta i zastąpiona innymi funkcjami*

Funkcję `results = func(app)` (rysunek 7.4.3.3-1) należy usunąć i zastąpić przez funkcje `DoPlots` i `nested_ode` z przykładu 7.4.3.2-1. W ten sposób obie funkcje zostały zintegrowane z programem tworzonym przez *App Designer*, co pokazano na rysunku 7.4.3.3-2.

```
12 -    end
13
14
15      methods (Access = private)
16          function DoPlots(app,beta)
17 -            t0=0; t2=12*pi;  % czas rozpoczęcia i zakończenia obliczeń
18 -            x0= [1.5 0];     % wartości początkowe dla t=0
19 -            [t,x]=nested_ode(app,[t0,t2], x0,beta);
20 -            plot(app.UIAxes,t,x(:,1),'r',t,x(:,2),'k--')
21 -            legend(app.UIAxes,'x(t)','dx/dt');
22 -            plot(app.UIAxes2,x(:,1),x(:,2))
23 -        end
24          function [t,x]=nested_ode(app,tspan, x0,beta);
25 -            [t,x] = ode23 (@mech_ode, tspan, x0); %Runge-Kutta solver
26              function xdot=mech_ode(t,x); %nested in nested_ode
27                  % mech_ode.m  to model układu drugiego rzędu
28                  % dx2/dt2 + beta*x' + x = 0
29                  % handle @mech_ode jest wielokrotnie wołane przez ode23
30 -                xdot=zeros(2,1);         % zerowy wektor dwuelementowy
31 -                xdot(1)=x(2);            % x(1)= x   x(2)= dx/dt
32 -                xdot(2)= -1*x(1) -beta*x(2);
33 -            end % internal nested function end
34 -        end % external nested function end
35      end
36
37
38      methods (Access = private)
```

Rysunek 7.4.3.3-2. *Okno Code View w App Designer. Funkcja results=func(app) została zastąpiona przez dwie funkcje: DoPlots i nested_ode*

7.4.4. Zalety użycia uchwytu zamiast nazwy funkcji

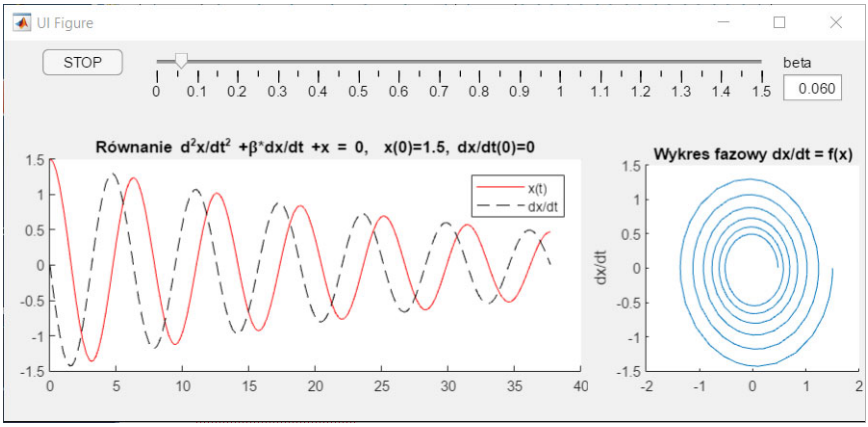
Zaleca się używanie uchwytu zamiast nazwy funkcji, jeśli ta funkcja jest parametrem innej funkcji. Tutaj pierwszym parametrem funkcji `ode23` jest uchwyt `@mech_ode` do funkcji `mech_ode`. Uchwyt to struktura, w której między innymi zapisano ścieżkę dojścia do funkcji. Daje gwarancję znalezienia tej funkcji (jeśli tylko nie została ona usunięta).

Użycie uchwytu do funkcji `mech_ode` jest konieczne, gdyż `mech_ode` została wpisana do *funkcji zagnieżdżonej*, a więc nie jest widoczna poza funkcją zagnieżdżoną.

Na rysunku 7.4.4-1 pokazano ostateczny rezultat: ekran programu z wykresami wyrysowanymi po poruszeniu suwakiem. W stanie oczekiwania na pierwsze poruszenie suwakiem okna wykresów są puste.

7.5. Instalowanie aplikacji

Aplikację można uruchomić, wpisując jej nazwę w oknie *Command*. Będzie ona odnaleziona i wykonana, jeśli znajduje się w aktualnym lub przeszukiwanym folderze. Jeśli aplikacja jest zainstalowana — to można ją uruchomić przez kliknięcie ikony dostępnej po wybraniu zakładki *APPS* w panelu MATLAB-a (rysunek 7.5-1). Aplikacje należące do zainstalowanych toolboksów (na przykład widoczny na rysunku *Control System Designer*) są zainstalowane.

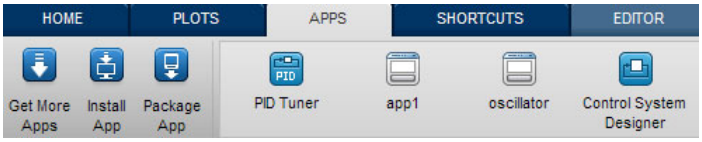


Rysunek 7.4.4-1. *Ostateczny rezultat: ekran z wykresami wyrysowanymi po poruszeniu suwakiem*

7.5.1. Instalowanie własnych aplikacji

Aplikacje przygotowane w podrozdziałach 7.3 i 7.4 mogą być łatwo zainstalowane. Pierwszym krokiem jest przygotowanie pliku instalacyjnego po kliknięciu ikony *Package Apps*. Wymaga to:

- wskazania *Main File* — głównego pliku aplikacji; może to być plik z rozszerzeniem *.mlapp*, utworzony przez *App Designera* lub *.m*, jeśli aplikację przygotowano z użyciem GUIDE lub w inny sposób;
- ewentualnego wskazania innych potrzebnych plików i wypełnienia krótkiej ankiety;
- rezultatem jest utworzenie pliku z rozszerzeniem *.mlappinstall*.



Rysunek 7.5-1. *Zakładka APPS w panelu MATLAB-a*

Plik z rozszerzeniem *.mlappinstall* może być zainstalowany w MATLAB-ie uruchomionym na dowolnym komputerze po kliknięciu ikony *Install App*.

7.5.2. Instalowanie rozszerzeń dostępnych poprzez Add-On Explorer

Ikona *Get More Apps* otwiera okno *Add-On Explorer* i inicjuje przeszukiwanie bazy danych firmy MathWorks zawierającej opisy i linki do ponad 400 aplikacji i wielu innych rozszerzeń do MATLAB-a (dane z lutego 2017 r.). Wybrane rozszerzenia można niezwłocznie pobrać i zainstalować (większość z nich jest bezpłatna).

Rozdział 8.

Metody numeryczne

W niniejszym rozdziale przedstawiono wybrane metody numeryczne z zakresu rozwiązywania układów równań algebraicznych i różniczkowych (zwyczajnych i cząstkowych), interpolacji, aproksymacji oraz analizy sygnałów. Wcześniej, w podrozdziale 2.2.10, zamieszczono przykład numerycznego rozwiązywania układu trzech równań algebraicznych oraz sposób użycia macierzy odwrotnej `inv(A)`.

Poczynając od roku 2000, funkcje algebry liniowej i operacje macierzowe MATLAB-a wykorzystują algorytmy z bibliotek LAPACK i BLAS. Uzyskano w ten sposób zwiększenie szybkości i dokładności obliczeń. Dodatkowe informacje o algorytmach obliczeniowych zaimplementowanych w MATLAB-ie można znaleźć poprzez polecenia: `doc matfun`, `doc datafun`, `doc polyfun`, `doc funfun`.

Obliczenia symboliczne wymagają użycia biblioteki *Symbolic Math Toolbox*. Biblioteka ta jest dołączona do wersji studenckiej MATLAB-a oraz niekiedy jest oferowana bezpłatnie w zestawie z MATLAB-em i Simulinkiem.

8.1. Układy równań liniowych

MATLAB automatycznie dokonuje wyboru najodpowiedniejszej metody rozwiązywania układu równań liniowych w postaci:

$$A \cdot x = b \tag{8.1-1}$$

gdzie:

- A jest macierzą kwadratową współczynników o wymiarze $n \times n$; będzie też omawiany wariant o n wierszach i m kolumnach,
- b jest wektorem kolumnowym o n elementach,
- wektor x jest poszukiwanym rozwiązaniem układu równań (8.1-1).

Rozwiązaniem układu równań (8.1-1) $A \cdot x = b$ jest `x=A\b`. Operator `\` (ang. *backslash*) wywołuje funkcję `mldivide`, która *wybierze najlepszy* dostępny *algorytm* i użyje go do obliczenia rozwiązania x. Operacja ta jest zwana *dzieleniem lewostronnym*, bo w równaniu (8.1-1) macierz A jest na lewo od x. Użycie właściwego algorytmu zwiększa szybkość i dokładność obliczeń.

Jeśli macierz *A* jest źle uwarunkowana lub bliska osobliwej, w oknie poleceń MATLAB-a pojawi się komunikat ostrzegawczy.

8.1.1. Dzielenie prawostronne dla x·A=b

Rozwiązanie równania w postaci

$$x \cdot A = b \tag{8.1.1-1}$$

oblicza się poleceniem `x = b/A` z użyciem operatora / (ang. *slash*). Wywołuje on funkcję `mrdivide`, która wybierze najlepszy dostępny algorytm i użyje go do obliczenia rozwiązania *x*. Operacja ta jest zwana *dzieleniem prawostronnym*, bo w równaniu 8.1.1-1 macierz *A* jest na prawo od *x*. Użycie właściwego algorytmu zwiększa szybkość i dokładność obliczeń.

W równaniach (8.1-1) i (8.1.1-1) dopuszcza się użycie macierzy prostokątnej, czyli rozwiązywanie układu nadokreślonego i niedookreślonego równań liniowych. Problem będzie omówiony w podrozdziale 8.1.7.

8.1.2. Automatyczny wybór algorytmu

Funkcje `mldivide` i `mrdivide` wybierają najlepszy dostępny algorytm dla zadanej macierzy *A*. W przypadku macierzy pełnej testowane jest kolejno, czy macierz *A*:

- jest kwadratowa,
- jest trójkątna,
- jest trójkątna spermutowana,
- jest macierzą Hamiltona,
- jest górną macierzą Hessenberga,
- ma tylko dodatnie lub tylko ujemne elementy na głównej przekątnej.

W zależności od rezultatu kolejnego testu podejmowana jest decyzja o wyborze algorytmu lub wykonywany jest kolejny test i podejmowana jest kolejna decyzja. Podobne, ale bardziej złożone testy są wykonywane dla macierzy rzadkich. Więcej informacji podają `doc mldivide` i `doc mrdivide`.

Użytkownik ma możliwość wyboru algorytmu według własnego uznania, np. `lu`, `chol`, `pinv`. Wyznaczanie rozwiązań poprzez odwracanie macierzy *A* nie jest zalecane. Wykonanie funkcji `inv(A)` trwa dość długo i nie daje dokładności większej niż algorytmy opisane wyżej.

8.1.3. Kilka rozwiązań dla różnych wektorów b

W jednym kroku można wyliczyć kilka rozwiązań równania $Ax = b$ dla różnych wartości wektora *b*. Wystarczy zamiast wektora *b* użyć macierzy *b*, a w jej kolumnach wpisać potrzebne wektory. Po wykonaniu obliczeń *x* będzie macierzą, której kolumny zawierają odpowiednie rozwiązania.

8.1.4. Przykład rozwiązania równania liniowego z liczbami zespolonymi

Dany jest układ równań liniowych zespolonych $Z \cdot I = U$ opisujący 3-oczkowy układ elektryczny. Przykład rozwiązywania równania z liczbami rzeczywistymi podano w podrozdziale 2.2.10.

$$\begin{pmatrix} z_1+z_{12}+z_{13} & -z_{12} & -z_{13} \\ -z_{12} & z_2+z_{12}+z_{23} & -z_{23} \\ -z_{13} & -z_{23} & z_3+z_{13}+z_{23} \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 230 \end{pmatrix} \qquad (8.1.4\text{-}1)$$

Przykład 8.1.4-1

Należy obliczyć wartości i_1, i_2, i_3 dla danych:

```
z1=50+50j;  z2=5+15j;   z3=1;
z12=-50j;   z23=-25j;   z13=50;
```

Poniższy skrypt oblicza elementy macierzy `Z`. Następnie rozwiązuje się układ równań $Z \cdot I = U$, wykorzystując operator dzielenia lewostronnego. Szukane rozwiązanie (wektor `I`) jest wynikiem dzielenia `I=Z\U`.

```
format rat                                  % rat poprawia czytelność wydruku
Z=[z1+z12+z13         -z12      -z13;       % obliczanie macierzy Z
         -z12   z2+z12+z23      -z23;       % kontynuacja linii
         -z13         -z23       z3+z13+z23]
U=[0 0 230];                                % zadany wektor U
format                              % przywrócenie standardowego formatu
disp ('rozwiązanie: I=Z\U')         % disp drukuje tekst na ekranie
I=Z\U'                              % oblicza i drukuje rozwiązanie
```

Obliczone elementy macierzy `Z` (liczby zespolone):

```
Z =
     100          0 + 50i     -50
       0 +50i     5 - 60i       0 + 25i
     -50          0 + 25i      51 - 25i
```

Rozwiązaniem jest wektor: `I=Z\U'`.

```
I =
   1.3928  - 2.0689i
   3.1236  - 2.4070i
   5.1925  - 1.0142i
```

Uwagi: MATLAB w liczbach zespolonych akceptuje zarówno litery `i` oraz `j`. W wynikach obliczeń używane jest `i`.

8.1.5. Równania liniowe źle uwarunkowane

Dokładność rozwiązania układu równań liniowych może być znacznie obniżona przy złym uwarunkowaniu macierzy *A*. Do oszacowania uwarunkowania macierzy służy funkcja `cond(A)` (ang. *condition* — warunek). Duże wartości tej funkcji świadczą o złym uwarunkowaniu, na przykład `cond(A)=10000` oznacza, że tylko cztery cyfry znaczące rezultatu obliczeń $x = A\backslash b$ są wiarygodne. Pozostałe cyfry powinny być odrzucone.

Dla równań źle uwarunkowanych operator \ użyje funkcji opartych na rozkładzie ortogonalnym **QR** lub **SVD** (podrozdziały 8.6.3 i 8.6.4) — pozwala to uzyskać dokładne rozwiązanie, ale powoduje około 10-krotne wydłużenie czasów obliczeń.

[Bjorck, Dahlquist, 1987] podają przykład źle uwarunkowanego układu równań liniowych w postaci:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & .0001 & 0 \\ 1 & 1 & 9999 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 10000 \end{pmatrix} \qquad (8.1.5\text{-}1)$$

Macierz *A* i wektor *b*, są zapisane w następujący sposób:

```
>> A=[1 0 0; 1 .0001 0; 1 1 9999]; b=[1 1 10000]'
```

Duża wartość funkcji `cond(A) = 1.4141e+008` potwierdza złe uwarunkowanie układu. Wartość wyznacznika `det(A)=0.99` nie jest bliska zeru, co może niesłusznie sugerować dobre uwarunkowanie macierzy. Rozwiązanie układu równań można wyznaczyć na dwa sposoby: `x=A\b` lub `x=inv(A)*b`.

Jednakże, pomimo złego uwarunkowania równania, w obu przypadkach uzyskuje się prawidłowy wynik `x=[1;0;1]`, tak jak to opisali [Bjorck, Dahlquist, 1987].

8.1.6. Sprawdzenie poprawności rozwiązań

[Bjorck, Dahlquist, 1987] wykazują, że mała wartość *residuum* ***r*** (definiowanego dla $A \cdot x = b$ jako $r = A \cdot \overline{x} - b$, gdzie $\overline{x}$ jest rozwiązaniem przybliżonym) może nie mieć żadnego związku z poprawnością i dokładnością uzyskanego wyniku. Dla układu równań liniowych:

$$\begin{pmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.8642 \\ 0.1440 \end{pmatrix} \qquad (8.1.6\text{-}1)$$

stwierdzono, że liczby $(\overline{x_1} = .9911, \overline{x_2} = -0.4870)$ spełniają to równanie z bardzo małym błędem. Wartość residuum wynosi $r = [10^{-8}, 10^{-8}]$. Dokładna wartość rozwiązania jest jednak zupełnie inna i można ją uzyskać, obliczając: `x=A\b` lub `x=inv(A)*b`.

W obu przypadkach uzyskuje się $x_1 = 2, \quad x_2 = -2$. Są to dokładne współrzędne punktu przecięcia prostych:

$$\begin{aligned} x_1 &= (-0.8648 \cdot x_2 + 0.8642) / 1.2969 \\ x_1 &= (-0.1441 \cdot x_2 + 0.1440) / 0.2161 \end{aligned} \qquad (8.1.6\text{-}2)$$

Otrzymane proste są prawie równoległe, co wyjaśnia małą wartość *residuum*.

8.1.7. Pseudoodwrotność i rozwiązywanie równań nadokreślonych i niedookreślonych

Zazwyczaj macierz A w równaniu $A \cdot x = b$ jest kwadratowa. Jednak na przykład w terenowych pomiarach geodezyjnych czy też w pomiarach laboratoryjnych liczba pomiarów może być większa niż liczba nieznanych wartości parametrów funkcji użytej do aproksymowania tych pomiarów.

Wiadomo, że wszystkie pomiary są obarczone błędami, a brakuje podstaw do wskazania, które z nich należy odrzucić. W takiej sytuacji zaleca się użyć wszystkich danych. Macierz A jest wtedy prostokątna (liczba pomiarów jest większa od liczby niewiadomych), a otrzymany układ równań określa się wtedy jako nadokreślony (ang. *overdetermined*). Szuka się wtedy „najlepszego" przybliżenia rozwiązania w sensie średniokwadratowym.

Możliwa jest też sytuacja odwrotna — równań jest mniej niż liczba potrzebnych parametrów i układ równań jest niedookreślony (ang. *underdetermined*). Rozwiązanie w takim przypadku nie jest jednoznaczne — będzie to na przykład dowolny punkt należący do wskazanej prostej, płaszczyzny lub bryły w przestrzeni wielowymiarowej.

Dla równań o niedostatecznej lub nadmiernej liczbie danych poszukuje się rozwiązania średniokwadratowego przy najmniejszej normie $\|A \cdot x - b\|$. Takie rozwiązania można wyznaczyć za pomocą funkcji `pinv`, która określa macierz pseudoodwrotną (ang. *pseudoinverse*).

Jeśli A jest macierzą o rozmiarach m, n i $m \neq n$ oraz b jest wektorem kolumnowym o m elementach, to wektor $x = A\backslash b$ jest rozwiązaniem średniokwadratowym nadokreślonego lub niedookreślonego układu równań w postaci $A \cdot x = b$. Wyznaczone w ten sposób rozwiązanie może różnić się od rezultatu `x=pinv(A)*b`.

8.2. Równania różniczkowe zwyczajne, zagadnienie początkowe

Poniżej omówiono rozwiązywanie zadania początkowego dla równań różniczkowych zwyczajnych oraz podano krótkie informacje dotyczące problemu brzegowego i równań różniczkowych cząstkowych.

8.2.1. Zagadnienie początkowe, solvery odeXX

Zadanie początkowe można zapisać w postaci równania różniczkowego z warunkiem początkowym:

$$\frac{dy}{dt} = F(t, y)$$
$$y(t_o) = y0 \qquad (8.2.1\text{-}1)$$

Do rozwiązywania zagadnienia początkowego są używane algorytmy pakietu ODE Suite, który jest częścią składową MATLAB-a. Algorytmy te będą oznaczane jako `odeXX` i są reprezentowane przez podane niżej funkcje, tak zwane *solvery*:

- `ode45` — zmodyfikowana (Bogacki i Shampine) jednokrokowa metoda Rungego-Kutty, rzędu IV i V.
- `ode23` — zmodyfikowana (Dormand-Prince) jednokrokowa metoda Rungego-Kutty, rzędu II i III. Dla równań o niewielkiej sztywności może być efektywniejsza niż `ode45`.
- `ode113` — metoda Adamsa-Bashfortha-Moultona (PECE). Jest wielokrokowa i bardzo efektywna, ale może być stosowana tylko dla równań mających *gładkie* rozwiązania.
- `ode15i` — aktualnie jest to *jedyny algorytm* dla równania różniczkowego **w postaci niejawnej** $f(t,x,x')=0$ oraz do równań różniczkowo-algebraicznych (ang. *differential algebraic equations* — DAEs), patrz też `doc ode15i` oraz `doc decic` — funkcja służąca do obliczenia spójnych warunków początkowych dla `ode15i`.
- `ode15s` — metoda wielokrokowa NDFs (opcjonalnie BDFs, tj. Geara), z możliwością zmiany rzędu, dla układów sztywnych (patrz objaśnienie poniżej). Rozwiązuje też równania z macierzą $M(t)$, opisane w podrozdziale 8.3.1.
- `ode23s` — zmodyfikowana jednokrokowa metoda Rosenbrocka, rzędu II, dla układów sztywnych. Rozwiązuje też równania z macierzą $M(t)$, opisane w podrozdziale 8.3.1.
- `ode23t` — metoda trapezowa dla układów umiarkowanie sztywnych oraz do równań różniczkowo-algebraicznych (ang. *differential algebraic equations* — DAEs).
- `ode23tb` — metoda TR-BDF2 dla układów sztywnych. Jest połączeniem metody trapezowej i wstecznego różniczkowania II rzędu.

Pojęcie układu lub zadania *sztywnego* (dosłowne tłumaczenie angielskiego *stiff*) można spotkać w publikacjach dotyczących metod numerycznych. Przykładowo tłumacz książki [Bjorck, Dahlquist, 1987] określa tak równanie różniczkowe, które opisuje *procesy (...) z bardzo różnymi skalami czasu*. Inni polscy autorzy zadanie takie określają jako sztywno stabilne lub *źle uwarunkowane* numerycznie.

8.2.2. Wybór parametrów dla solvera odeXX

Ogólna postać wywołania solverów `odeXX` dla zadania początkowego równań różniczkowych zwyczajnych jest następująca:

```
[tout,yout] = odeXX(odefun, [t0 tf],y0)
```

lub

```
[tout,yout,varargout] = odeXX(odefun, [t0 tf],y0,options,varargin)
```

lub

```
sol = solver(odefun,[t0 tf],y0…)
```

gdzie:

- `[tout,yout]` — wyniki obliczeń: wektor czasu `tout` i macierz `yout`: jej kolumny zawierają wartości rozwiązań $y(t)$, $\dot{y}(t)$,…
- `odeXX` — nazwa solvera, czyli funkcji reprezentującej algorytm rozwiązywania równań różniczkowych zwyczajnych. Przy pierwszym podejściu zaleca się użycie `ode45` lub `ode23`, gdyż dają one poprawne rozwiązania dla dużej grupy równań.
- `odefun` — nazwa pliku funkcyjnego (lub lepiej jego uchwyt `@odefun`, ang. *function handle*), w którym zapisano układ równań różniczkowych I rzędu. Dokładniej `odefun` jest to plik przygotowywany przez użytkownika, w którym oblicza się wartości pochodnych poszukiwanego rozwiązania i ewentualnie inne parametry, na przykład wartości jakobianu.
- `[t0 tf]` — wektor, w którym określa się początkową i końcową wartość zmiennej niezależnej (zwykle jest to czas). Można też podać wszystkie punkty (w kolejności rosnącej lub malejącej), dla których potrzebne są wartości rozwiązania $y(t)$. Jeśli znane są warunki końcowe (a nie początkowe), to można wykonać całkowanie wstecz. Wtedy początkowa wartość czasu jest większa od jego wartości końcowej.
- `y0` — wartości początkowe dla układu równań.
- `options` — zmienna typu `struct`, w jej polach zapisuje się wartości parametrów całkowania równań różniczkowych. Wartości domyślne (ang. *default*) można zmienić za pomocą funkcji `odeset` (podrozdział 8.2.3).
- `varargout`, `varargin` — parametry dodatkowe. Są one objaśnione w opisie każdej z metod rozwiązywania równań poprzez `doc`, na przykład `doc ode23`.
- `sol` — *struktura*, czyli zmienna typu `struct`, do której mogą być zapisywane wyniki obliczeń. Więcej informacji podano w podrozdziałach 8.4.2 i 8.4.3.

Szczegółowy opis stosowania poszczególnych algorytmów uzyskuje się za pomocą polecenia `doc solver`, na przykład `doc ode113`.

8.2.3. Modyfikowanie parametrów solvera odeXX

Funkcje `odeXX` (ang. *ordinary differential equation solver*) zawierają algorytmy rozwiązywania równań różniczkowych zwyczajnych. Odpowiedni dobór parametrów dla poszczególnych algorytmów pozwala dostosować ich działanie do specyfiki konkretnego układu równań. Do ustawiania i odczytywania aktualnych wartości parametrów solvera służą polecenia `odeset` i `odeget`. Pokazano rezultat wpisania polecenia `odeset`:

```
          AbsTol: [ positive scalar or vector {1e-6} ]
          RelTol: [ positive scalar {1e-3} ]
    NormControl: [ on | {off} ]
    NonNegative: [ vector of integers ]
      OutputFcn: [ function_handle ]
      OutputSel: [ vector of integers ]
         Refine: [ positive integer ]
          Stats: [ on | {off} ]
    InitialStep: [ positive scalar ]
        MaxStep: [ positive scalar ]
```

```
              BDF: [ on | {off} ]
         MaxOrder: [ 1 | 2 | 3 | 4 | {5} ]
         Jacobian: [ matrix | function_handle ]
         JPattern: [ sparse matrix ]
      Vectorized: [ on | {off} ]
             Mass: [ matrix | function_handle ]
MStateDependence: [ none | {weak} | strong ]
       MvPattern: [ sparse matrix ]
    MassSingular: [ yes | no | {maybe} ]
    InitialSlope: [ vector ]
          Events: [ function_handle ]
```

Wartość pól tej struktury można zmienić za pomocą polecenia w postaci:

```
options = odeset('pole_1',wartość_1,'pole_2',wartość_2,…)
```

gdzie: pary (*'pole', wartość*) reprezentują odpowiednio nazwę pola powyższej struktury i jego nową wartość. Wybrane pola reprezentują następujące parametry algorytmów:

- `RelTol` — oszacowanie błędu względnego, domyślna wartość `e-3`. Szacowana wartość błędu jest utrzymywana na poziomie zapewniającym spełnienie warunku: `e(i) <= max(RelTol*abs(y(i)), AbsTol(i)`.
- `AbsTol` — oszacowanie maksimum błędu bezwzględnego, domyślnie `e-6`.
- `Refine` — parametr, który zwiększa *n*-krotnie liczbę obliczanych wartości rozwiązania, co poprawia jakość wykresów rozwiązania.
- `MaxStep` — maksymalna długość kroku całkowania, domyślnie jest to jedna dziesiąta przedziału określonego przez `[t0 tf]`. Dla rozwiązań okresowych `MaxStep` powinien być mniejszy od jednej czwartej okresu.
- `InitialStep` — początkowa wartość kroku całkowania.
- `OutputFcn` — nazwa funkcji, która jest wywoływana po każdym kroku całkowania. Może to być funkcja przygotowana przez użytkownika albo jedna z następującego zestawu funkcji: `odeplot`, `odephas2`, `odephas3`, `odeprint`.
- `OutputSel` — wektor, który podaje indeksy składowych wektora rozwiązań przekazywanych dla wyżej opisanego parametru `OutputFcn`. Domyślnie są to wszystkie składowe.
- `Stats` — domyślna wartość `'off'`. Po zmianie `'on'` wypisuje informacje statystyczne o przebiegu obliczeń.

Pozostałe parametry, z których większość dotyczy rozwiązywania układów sztywnych, podaje polecenie `doc odeset`. Przykład użycia funkcji `odeset` zamieszczono w podrozdziałach 8.2.4 i 8.2.5.

8.2.4. Wpływ parametrów solvera na obliczenia

Wybór algorytmu rozwiązywania równań różniczkowych oraz właściwy dobór jego parametrów mają istotny wpływ na czas trwania obliczeń i możliwość uzyskania dokładnych wyników. Wykazano to na przykładzie rozwiązania równania różniczkowego II rzędu [Mrozek, 1997] w następującej postaci:

$$\begin{aligned} &\ddot{y} - y = 4 \cdot \sin(t) + 5 \cdot \cos(2t) \\ &y(0) = -1, \quad \dot{y}(0) = -2 \end{aligned} \tag{8.2.4-1}$$

Rozwiązaniem dokładnym tego równania jest funkcja $y_a = -2 \cdot sin(t) - cos(2t)$. Aby rozpocząć obliczenia, należy przekształcić to równanie na układ równań I rzędu, jak opisano w podrozdziale 8.3.1.

$$\begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ 4 \cdot sin(t) + 5 \cdot cos(2t) + y_1 \end{pmatrix} \tag{8.2.4-2}$$

Równania (8.2.4-2) zapisano w funkcji o nazwie `eqdif1`.

```
function dy =eqdif1(t,y)
dy = [y(2); 4.*sin(t) + 5.*cos(2.*t)+ y(1)];
```

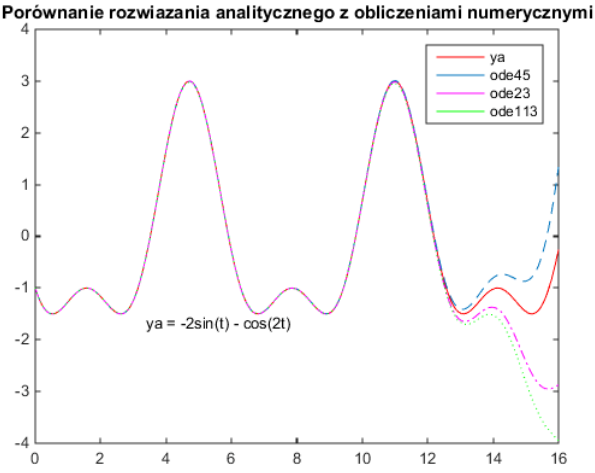
Obliczenia wykonano kolejno z użyciem solverów: `ode45`, `ode23`, `ode113`. W polu `'RelTol'` struktury `odeset` ustawiono nową wartość błędu względnego. Otrzymane rezultaty pokazano na wykresach, które wykonano w przykładzie 8.2.4-1.

Przykład 8.2.4-1

Badanie zgodności rozwiązania analitycznego i rozwiązań numerycznych uzyskanych z użyciem solverów `ode45`, `ode23` i `ode113`.

```
options= odeset('RelTol',1e-5);
       [t,y]= ode45('@eqdif1',[0 16],[-1 -2],options);
   [t23,y23]= ode23('@eqdif1',[0 16],[-1 -2],options);
   [t13,y13]= ode113('@eqdif1',[0 16],[-1 -2],options);
         ya= -2.*sin(t)-cos(2.*t);
plot(t,ya,'r',t,y(:,1),'--',t23,y23(:,1),'m-.',t13,y13(:,1),'g:')
s='Porównanie rozwiązania analitycznego z obliczeniami numerycznymi'
title(s)
legend('ya','ode45','ode23','ode113')    % legenda do linii wykresu
text(3.4,-1.7,'ya = -2sin(t) - cos(2t)')
```

Na rysunku 8.2.4-1 pokazano wykres rozwiązania dokładnego (linia ciągła) i rezultaty obliczeń numerycznych. Widać, że dla końcowego przedziału czasu uzyskane wyniki znacznie się różnią pomiędzy sobą i nie pokrywają się z rozwiązaniem dokładnym. Potwierdza to **konieczność weryfikowania rezultatów** obliczeń, na przykład poprzez powtórzenie obliczeń z użyciem innej metody lub innego kroku całkowania.



Rysunek 8.2.4-1. *Rozwiązania równania różniczkowego (8.2.4-2) tracą zbieżność dla czasu powyżej 12 s*

Poprawne rezultaty (dla każdego z zastosowanych algorytmów) uzyskano po zmniejszeniu wartości błędów względnego i bezwzględnego oraz maksymalnego kroku całkowania za pomocą polecenia:

```
options= odeset('RelTol',1e-7,'AbsTol',1e-8,'MaxStep',1e-1);
```

Dla algorytmu ode45 poprawny wynik osiągnięto w krótszym czasie i przy większych wartościach RelTol i AbsTol niż dla ode23:

```
options= odeset('RelTol',1e-5,'AbsTol',1e-7,'MaxStep',1e-1);
```

Sprawdzenie innych wariantów tych parametrów i ich wpływu na czas obliczeń pozostawia się Czytelnikowi.

Należy zwrócić uwagę na wykazany powyżej wpływ maksymalnej wartości kroku całkowania (parametr 'MaxStep') na czas obliczeń i na poprawność uzyskiwanych wyników. Domyślna wartość 'MaxStep' jest wyznaczana jako jedna dziesiąta przedziału określonego przez [t0 tf], czyli $(t_0 - t_f)/10$. Stwierdzono, że dla dużych wartości czasu symulacji oraz dla rozwiązań o przebiegach okresowych lub bliskich okresowym — domyślna wartość parametru 'MaxStep' jest nadmiernie wysoka.

8.2.5. Algorytmy dla układów źle uwarunkowanych

Algorytmy ode23 i ode23s dodatkowo przetestowano na specjalnie dobranym układzie równań różniczkowych, podanym przez [Bjorck, Dahlquist, 1987].

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} x_2 \\ -1000 \cdot x_1 - 1001 \cdot x_2 \end{pmatrix} \qquad (8.2.5\text{-}1)$$
$$x_1(0) = 1; \quad x_2(0) = -1$$

Równania (8.2.5-1) są układem sztywnym (ang. *stiff*), czyli źle uwarunkowanym numerycznie. Jego wartości własne λ (a dokładniej ich moduły) różnią się znacznie. Rozwiązanie takiego układu posiada składowe szybkozmienne i wolnozmienne. Typowe algorytmy rozwiązywania równań różniczkowych (na przykład metoda Rungego-Kutty ode23) są w takim przypadku mało skuteczne.

Układ równań (8.2.5-1) zapisano w przykładzie 8.2.5-1 w postaci funkcji o nazwie ode1000. Jest ona umieszczona w końcowej części pliku funkcyjnego *stiff_test.m* jako funkcja lokalna, a pierwszą linią tej funkcji jest function xdot= ode1000(t,x). Należy zwrócić uwagę, że pierwszy parametr funkcji ode23, nie jest ode1000, ale @ode1000, czyli uchwyt tej funkcji (*function handle*, podrozdział 3.2.2.2). W uchwycie jest zapamiętana ścieżka dostępu do ode1000, co umożliwia skuteczne wywołanie tej funkcji z wnętrza solverów ode23 i ode23s. Użycie uchwytu pozwala na umieszczenie funkcji ode1000 wewnątrz pliku *stiff_test.m*. Funkcje tic i toc służą do pomiaru czasu obliczeń.

Przykład 8.2.5-1

```
function [t,x]=stiff_test;    % stiff_test
t0=0; t2=3;          % zadany przedział
x0= [1  -1];         % warunki początkowe dla t=0
options=odeset('AbsTol',1e-1,'stat','on');
%% ode23
tic,[t,x] = ode23(@ode1000, [t0, t2], x0, options);toc
```

```
subplot(1,3,1), plot(t,x), title('ode23')
%% ode45
tic,[t,x] = ode45(@ode1000, [t0, t2], x0, options);toc
subplot(1,3,2), plot(t,x) , title('ode45')
%% ode23s
tic,[t,x] = ode23s(@ode1000, [t0, t2], x0, options);toc
subplot(1,3,3), plot(t,x) , title('ode23s')
%% równanie  różniczkowe zapisano w funkcji ode1000
function xdot=ode1000(t,x);
xdot=zeros(2,1);     % zerowy wektor 2-elementowy
xdot(1)=x(2);  xdot(2)=-1001*x(2) -1000*x(1);
```

Rozwiązaniem dokładnym równania (8.2.5-1) są funkcje:

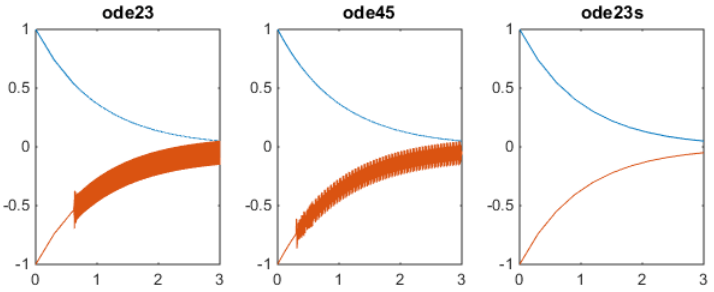
$$x_1 = e^{-t},\ x_2 = -e^{-t}$$

Do testowania przyjęto dokładność `AbsTol=0.10`. Dla `ode45` i `ode23` są widoczne oscylacje rozwiązania (dolna krzywa — lewy i środkowy wykres na rysunku 8.2.5-1. Solver Rungego-Kutty'ego `ode45` traci stabilność absolutną dla tego równania już przy kroku większym od 0.027.

Dokładniejsze informacje o pracy algorytmu uzyskano, ustawiając parametr `'stats','on'`. Z informacji statystycznych (tabela 8.2.5-1) wynika, że `ode23` (lewy wykres) rozwiązuje równania źle uwarunkowane znacznie wolniej i mniej efektywnie niż `ode23s`. Używając metody Rosenbrocka `ode23s`, uzyskano przeszło 20-krotne przyspieszenie obliczeń i zupełny brak oscylacji rozwiązań. Wyższa efektywność `ode23s` jest skutkiem wykorzystania w tym algorytmie rozkładu LU i wartości pochodnych cząstkowych.

Tabela 8.2.5-1. *Porównanie efektywności solverów ode23 i ode23s dla źle uwarunkowanego równania (8.2.5-1)*

solver ode23	solver ode23s
950 successful steps	10 successful steps
7 failed attempts	0 failed attempts
2872 function evaluations	52 function evaluations
0 partial derivatives	10 partial derivatives
0 LU decompositions	10 LU decompositions
0 solutions of linear systems	30 solutions of linear systems
elapsed_time = 1.0400	elapsed_time = 0.0500



Rysunek 8.2.5-1. *Równanie różniczkowe źle uwarunkowane, parametr AbsTol=1e-1, solver ode45 (lewy wykres), ode23 (środkowy wykres) i ode23s (prawy wykres)*

8.3. Równanie różniczkowe III rzędu, przykład

Dane jest równanie różniczkowe zwyczajne **III rzędu** z **trzema** warunkami początkowymi:

$$\begin{aligned} &\dddot{y} + 6\ddot{y} + 11\dot{y} + 6y = 0 \\ &y(0) = 0,\ \dot{y}(0) = 0,\ \ddot{y}(0) = 1 \end{aligned} \tag{8.3-1}$$

Należy obliczyć rozwiązanie tego równania dla czasów z przedziału [0,10].

Aby użyć któregokolwiek z opisanych w podrozdziale 8.2.1 *solverów*, czyli algorytmów rozwiązujących równania różniczkowe (na przykład `ode23`), należy przygotować własną funkcję `odefun`. Funkcja ta będzie wielokrotnie wywoływana przez *solver* podczas rozwiązywania równania (8.3-1). Z uwagi na dużą liczbę błędów popełnianych przez studentów proces tworzenia tej funkcji będzie **szczegółowo** omówiony.

8.3.1. Przekształcenie równania różniczkowego III rzędu na układ trzech równań I rzędu

Pierwszym krokiem jest przekształcenie równania różniczkowego **III rzędu** (8.3-1) na układ **trzech** równań różniczkowych I rzędu. W tym celu definiuje się **trzy** nowe zmienne y_1, y_2, y_3, które będą użyte zamiast zmiennej y i jej pochodnych: $\dot{y}$, $\ddot{y}$.

$$\begin{aligned} y &= y_1 \\ \dot{y} &= y_2 \Rightarrow \quad \dot{y}_1 = y_2 \\ \ddot{y} &= y_3 \Rightarrow \quad \dot{y}_2 = y_3 \\ \dddot{y} &= \dot{y}_3 \Rightarrow \quad \dot{y}_3 = \ldots \end{aligned} \tag{8.3.1-1}$$

W nowym układzie równań po lewej stronie znaku równości ma być wektor pierwszych pochodnych nowych zmiennych, czyli $\dot{y}_1, \dot{y}_2, \dot{y}_3$. Pierwsze dwa wiersze nowego równania, czyli $\dot{y}_1 = y_2$, $\dot{y}_2 = y_3$, są już przygotowane w (8.3.1-1), po znaku $\Rightarrow$. Ostatni, **trzeci** wiersz otrzymuje się z oryginalnego równania różniczkowego (8.3-1), przenosząc na prawą stronę znaku równości wszystkie elementy tego równania poza $\dddot{y}$, czyli poza najwyższą pochodną. Otrzymuje się:

$$\dddot{y} = -6\ddot{y} - 11\dot{y} - 6y \tag{8.3.1-2}$$

W równaniu 8.3.1-2 użyto starych zmiennych. Pamiętając, że $\dddot{y} = \dot{y}_3$, zmienne y, $\dot{y}$, $\ddot{y}$ należy odpowiednio zastąpić przez y_1, y_2, y_3, korzystając z 8.3.1-1. W ten prosty sposób otrzymuje się ostatni **trzeci** wiersz nowego układu trzech równań różniczkowych I rzędu (8.3.1-3). Ten układ jest odpowiednikiem jednego równania różniczkowego zwyczajnego **III rzędu** 8.3-1. Warunki początkowe nie wymagają przeliczania, lecz są przepisane z użyciem nowych zmiennych. Nowe zadanie, równoważne równaniu (8.3-1), ma postać:

$$\begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \end{pmatrix} = \begin{pmatrix} y_2 \\ y_3 \\ -6y_3 - 11y_2 - 6y_1 \end{pmatrix} \qquad (8.3.1\text{-}3)$$

$$y_1(0) = 0,\ y_2(0) = 0,\ y_3(0) = 1$$

Macierz po prawej stronie znaku równości jest odpowiednikiem funkcji $F(t, y)$ z równania (8.2.1-1). Powyższe równanie będzie zapisane w przykładzie 8.3.2-1 jako `function [dy] = ode1rz(t,y)`.

Reasumując, przekształcając równanie różniczkowe *dowolnego rzędu* ***n*** (np. równanie (8.3-1)), zawsze definiuje się *dokładnie* ***n*** nowych zmiennych, nawet jeśli w oryginalnym równaniu niektóre pochodne nie występują. Rząd równania różniczkowego to *rząd najwyższej pochodnej* występującej w tym równaniu. Po przekształceniach zawsze otrzymuje się układ ***n*** *równań* różniczkowych *I rzędu* oraz ***n*** warunków początkowych.

Najczęstsze błędy popełniane przy powyższych przekształceniach to:

- niewłaściwa liczba nowych zmiennych,
- niewłaściwa liczba wierszy w równaniu (8.3.1-3),
- błędne numerowanie nowych zmiennych lub pozostawienie nawet kilku starych zmiennych w (8.3.1-3).

8.3.2. Przygotowanie pliku funkcyjnego obliczającego pochodne

W oparciu o układ równań (8.3.1-3) należy przygotować *plik funkcyjny* obliczający pochodne $\dot{y}_1, \dot{y}_2, \dot{y}_3$ dla zadanych wartości czasu `t` oraz dla zadanych wartości zmiennej niezależnej `y`. W MATLAB-ie nowe zmienne i ich pochodne można zapisać jako pionowe wektory `y` i `dy`. Dla równania *III rzędu* będą to *wektory 3-elementowe*, a równanie (8.3.1-3) będzie zapisane jak niżej:

```
dy(1)= y(2);
dy(2)= y(3);
dy(3)= -6*y(3)  -11*y(2)  -6*y(1);
```

lub w jednym wierszu, z użyciem nawiasów kwadratowych.

```
dy= [y(2);, y(3); -6*y(3)  -11*y(2)  -6*y(1)];
```

Funkcja obliczająca pochodne `dy(1)`, `dy(2)`, `dy(3)` może mieć dowolną nazwę (w dokumentacji używa się `odefun`) — tutaj funkcję nazwano `ode1rz`. Należy ją zapisać w pliku z rozszerzeniem *.m* o tej samej nazwie, czyli *ode1rz.m*. Aby algorytmy typu `ode`*XX* działały poprawnie, pierwsza linia funkcji musi być zgodna z opisem podanym w podrozdziale 8.2.2, np. `function [dy] = ode1rz(t,y)`.

Przykład 8.3.2-1

Przygotowano funkcję obliczającą pochodne `dy(1)`, `dy(2)`, `dy(3)`dla (8.3.1-3), czyli też dla (8.3-1).

```
function [ dy ] = ode1rz(t,y)
%file ode1rz.m
% oblicza pochodne dy(1), dy(2), dy(3)
```

```
dy=[0;0;0];              % użyto średników, aby wektor zerowy był pionowy
dy(1)= y(2);
dy(2)= y(3);
dy(3)=  -6*y(3)  -11*y(2)  -6*y(1);
%
% dy=[y(2); y(3); -6*y(3)-11*y(2)- 6*y(1)]   % inny zapis: dy jako wektor
```

Uchwyt tej funkcji `@ode1rz` jest pierwszym parametrem wejściowym solvera *odeXX*.

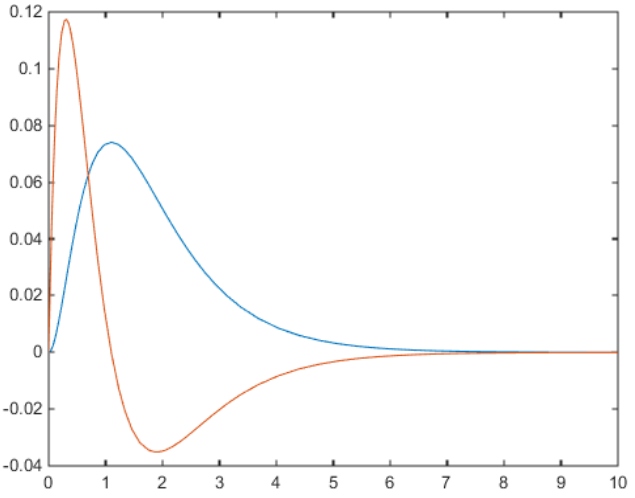
8.3.3. Wywołanie solvera odeXX i wizualizacja rozwiązania

Gdy funkcja obliczająca pochodne jest zapisana w pliku *ode1rz.m*, można rozwiązać numerycznie równanie różniczkowe (8.3-1) w zadanym przedziale czasu, np. `[0,10]`, wywołując w oknie *Command Window* dowolny solver, np. `ode23`:

```
[t,y]=ode23(@ode1rz,[0,10],[0,0,1]);
plot(t,y(:,1:2))     % wykres rozwiązania, funkcje y(t) i dy/dt
```

Parametrami `ode23` są odpowiednio: nazwa pliku *ode1rz* lub lepiej uchwyt `@ode1rz`, przedział czasu `[0,10]` oraz warunki początkowe — wektor zawierający $y(0)$, a następnie wartości kolejnych pochodnych, poczynając od $\dot{y}(0)$. W tym przykładzie nie użyto dodatkowych parametrów, np. *options*.

Na rysunku 8.3.3-1 pokazano wykres rozwiązania równania (8.3-1), wykonany przez `plot(t,y(:,1:2))`. Wyrażenie `y(:,1:2)` wybiera do wykresu tylko pierwszą i drugą kolumnę macierzy `y`. Dzięki temu na rysunku będzie pokazany tylko wykres funkcji $y(t)$ i jej pierwszej pochodnej $\dot{y}(t)$. Trzecia kolumna macierzy `y`, zawierająca drugą pochodną, będzie pominięta. Czytelnik może jednoznacznie zidentyfikować wykresy pokazane na rysunku 8.3.3-1, wiedząc, że funkcja $y(t)$ osiąga swoje ekstremum lokalne wtedy, gdy jej pochodna ma wartość zero. Można też odczytać z wykresu warunki początkowe dla `t=0`.



Rysunek 8.3.3-1. *Rozwiązanie równania różniczkowego zwyczajnego (8.3-1); pokazano funkcję i jej pierwszą pochodną*

8.3.4. Symbolic Math Toolbox

Biblioteka *Symbolic Math Toolbox* daje możliwość wykonywania obliczeń na zmiennych symbolicznych. Zestawienie dostępnych funkcji podaje polecenie `doc symbolic`. Jego zastosowania obejmują:

- obliczenia z możliwością dowolnego określenia dokładności, aktualnie do 49 cyfr znaczących. Funkcja `vpa` to skrót od *variable precision arithmetics*. Polecenie `vpa(pi,49)` oblicza wartość liczby *pi* z zadaną precyzją, tj. 49 cyfr: .1415926535897931159979634685441851615905761718 75
- całkowanie i różniczkowanie analityczne wzorów matematycznych,
- upraszczanie i przekształcanie wyrażeń, na przykład przekształcenie złożonego układu równań analitycznych w postać równań stanu,
- obliczenia z zakresu algebry liniowej, w tym symboliczne obliczanie wartości własnych i wektorów własnych,
- analityczne rozwiązanie równań algebraicznych i różniczkowych.

8.3.4.1 Rozwiązanie analityczne równania różniczkowego

Mając zainstalowany *Symbolic Math Toolbox*, można wyliczyć rozwiązanie analityczne równania (8.3-1). Wystarczy w oknie *Command* wpisać i zatwierdzić polecenia:

```
y=dsolve('D3y+6*D2y+11*Dy+6*y=0','2y(0)=1,Dy(0)=0,y(0)=0')
```

Parametrami `dsolve` są kolejno: wyrażenie matematyczne jako zmienna tekstowa (pomiędzy apostrofami) oraz ewentualnie warunki początkowe. Bywa, że zamiast rozwiązania pojawia się komunikat `Error: The input character is not valid`…. Najczęstszym powodem tego błędu jest użycie znaków akcentu (‘) zamiast znaku apostrofu (’). Jeśli nie ma błędów, to otrzymuje się rozwiązanie w postaci analitycznej, czyli wzór matematyczny:

$$y=1/2*\exp(t))-1/\exp(2t)+1/(2*\exp(3t)) \qquad (8.3.4\text{-}1)$$

Czas obliczeń zmierzony funkcjami `tic`, `toc` wyniósł za pierwszym razem ponad 6 s. Powtórzenie obliczeń wymagało już tylko 0.18 s. Jeśli jednak usunięto z pamięci kod półskompilowanego poleceniem `clear all`, to czas obliczeń powiększył się znacząco (do 3.40 s). Korzystając z MATLAB Online, uzyskano praktycznie te same czasy.

Wykres uzyskanego rozwiązania w zadanym przedziale `[0,10]` wykona `fplot(y,[0,10])` lub, dla starszych wersji MATLAB-a, `ezplot(y,[0,10])`.

Przykłady użycia *Symbolic Math Toolbox* do całkowania i różniczkowania przedstawiono w podrozdziałach 8.5.2 i 8.5.3. Więcej informacji podaje `doc symbolic`.

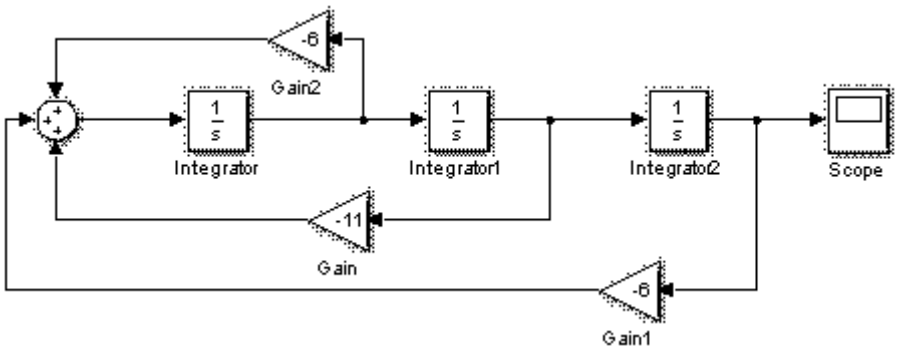
8.3.5. Model równania różniczkowego w Simulinku

Najprostszym sposobem numerycznego rozwiązania równania (8.3-1) jest metoda modelowania graficznego z użyciem Simulinka (patrz rozdział 10.). Nie trzeba niczego programować, gdyż biblioteki Simulinka zawierają gotowe bloki z modelami elementów równania różniczkowego: integratory, wzmacniacze i sumatory.

Dla równania III rzędu należy użyć trzech integratorów (blok całkujący) połączonych szeregowo, wzmacniaczy oraz jednego sumatora. Zakłada się, że na wejście pierwszego integratora podawany jest sygnał pochodnej najwyższego rzędu (tutaj: $\dddot{y} = dy^3 / dt^3$). Kolejne całkowania w integratorach obniżają rząd pochodnej o jeden, kolejno do $\ddot{y}$, $\dot{y}$, aż do uzyskania szukanego rozwiązania $y(t)$ na wyjściu trzeciego, ostatniego integratora.

Aby taki model działał poprawnie, należy spełnić założenie, to znaczy należy utworzyć i podać na wejście pierwszego integratora sygnał najwyższej pochodnej, utworzony zgodnie ze wzorem (8.3.1-2): $\dddot{y} = -6\ddot{y} - 11\dot{y} - 6y$. Sygnały występujące po prawej stronie tego równania są pobierane z kolejnych integratorów i po przemnożeniu przez odpowiednie współczynnik doprowadza się je na wejście sumatora, umieszczonego przed pierwszym integratorem.

W każdym integratorze po jego kliknięciu pojawia się okienko, w którym określamy warunki początkowe zgodnie z tematem zadania (wzór 8.3-1). Na rysunku 8.3.5-1 pokazano otrzymany model równania różniczkowego zapisany graficznie w Simulinku (patrz też rozdział 10.).



Rysunek 8.3.5-1. *Model równania różniczkowego III rzędu zapisany graficznie w Simulinku*

8.4. Inne równania różniczkowe i cząstkowe

8.4.1. Równania różniczkowe z macierzą M(t)

Algorytmy `ode23s` i `ode15s` mogą rozwiązywać równania z zależną od czasu macierzą $M(t)$ (*macierz masowa*) w postaci:

$$M(t) \cdot \frac{dy}{dt} = F(t, y) \tag{8.4.1-1}$$

Umożliwia to wykorzystanie powyższych algorytmów do przybliżonego rozwiązywania pewnej klasy równań różniczkowych cząstkowych metodą elementów skończonych.

Efektywność algorytmów `ode23s` i `ode15s` znacznie wzrasta, jeśli podany zostanie wzór analityczny na jakobian prawej strony rozwiązywanego zadania. Użycie jakobianu opisuje `doc ode23s`.

8.4.2. Zagadnienie brzegowe

Zagadnienie brzegowe ma postać:

$$\frac{dy}{dt} = F(t, y) \tag{8.4.2-1}$$

z *warunkiem brzegowym* $g(y(a), y(b)) = 0$ określonym na przedziale $[a, b]$.

Do rozwiązania równania (8.4.2-1) używa się:

- solverów `bvp4c` i `bvp5c` (ang. *boundary value problem*) — rozwiązują one zadanie brzegowe dla równania różniczkowego zwyczajnego; parametrami przy wywołaniu tych solverów są uchwyty (ang. *function handle*) oraz wektory i macierze opisujące rozwiązywane zadanie,
- funkcji `bvpinit` — przekształca wstępne przybliżenie (ang. *initial guess*) rozwiązania w strukturę,
- równania różniczkowego i jego warunku brzegowego — zapisuje się je w postaci plików funkcyjnych `odefun()` i `bcfun()`,
- funkcji `bvpset` i `bvpget` — służą one do ustawiania i odczytu opcji solverów,
- funkcji `deval` — oblicza wartości rozwiązania, używając wyników obliczeń zapisanych do struktury `sol`.

Równanie (8.4.2-2) może zawierać dodatkowe parametry (`p1`, `p2`). Wartości tych parametrów są przekazywane do funkcji `odefun()` i `bcfun()`, które opisują wtedy równania w postaci:

$$\begin{aligned} \frac{dy}{dt} &= F(t, y, p1) \\ g(y(a), y(b), p2) &= 0 \end{aligned} \tag{8.4.2-2}$$

Wyniki obliczeń zadań brzegowych *nie są zapisywane* do macierzy `[tout,yout]`, lecz do struktury `sol` (czyli do zmiennej typu *struct*). Aby pokazać wyniki obliczeń na wykresie, należy użyć funkcji `deval`. Przykładowo, aby przygotować wykres rozwiązania dla czasu $t \in [0, 5]$, należy wykonać dodatkowe polecenia podane poniżej:

```
tt = linspace(0,5);
yy = deval(sol,tt);
plot(tt,yy)
```

Więcej informacji podają polecenia `doc bvp4c` i `doc bvp5c`.

8.4.3. Równania różniczkowe zwyczajne z opóźnieniem

Równanie różniczkowe zwyczajne z opóźnieniem ma postać:

$$y'(t) = f(t, y(t), y(t-\tau_1), ..., y(t-\tau_k)) \qquad (8.4.3\text{-}1)$$

gdzie $\tau_i > \tau_{i+1}, i = 1..k-1$.

Należy zwrócić uwagę, że wartość pochodnej y' w chwili (t_0) zależy zarówno od wartości funkcji w zerze $y(t_0)$, jak i od wcześniejszych wartości $y(t_0-\tau_1)...y(t_0-\tau_k)$ tej funkcji.

Do rozwiązania równania (8.4.3-1) używa się:

- solvera `dde23` (ang. *delay differential equations*) — rozwiązuje równania ze stałym opóźnieniem, gdzie opóźnienia $\tau_i, i = 1,k$ są nieujemnymi liczbami rzeczywistymi,
- solvera `ddesd` — rozwiązuje bardziej ogólne równanie, w którym opóźnienie $\tau_i, i = 1,k$ jest zależne od czasu *t* i wartości funkcji $y(t)$,
- funkcji `deval` — oblicza wartości rozwiązania, używając wyników obliczeń zapisanych w strukturze `sol`.

Wyniki obliczeń wykonanych z użyciem solverów `dde23` i `ddesd` nie są zapisywane do `[tout,yout]`, lecz do struktury `sol`, jak opisano w podrozdziale 8.4.2.

8.4.4. Równania różniczkowe cząstkowe

MATLAB może być wykorzystany do rozwiązywania układów **równań różniczkowych cząstkowych** parabolicznych i eliptycznych, określonych dla jednej zmiennej przestrzennej ***x*** i dla czasu ***t***. Co najmniej jedno równanie musi być paraboliczne. Korzystając z symetrii zmiennej przestrzennej, można rozwiązywać zadania dwu- i trójwymiarowe z symetrią, w tym dla walca i dla kuli. Sposób rozwiązywania równań różniczkowych cząstkowych jest podawany przez `doc pdepe`.

Inne problemy oraz równania hiperboliczne wymagają użycia *Partial Differential Equation Toolbox*, a zadania wielodomenowe (ang. *multiphysics*) można rozwiązać z użyciem pakietu *COMSOL multiphysics*. Jest on dostępny odpłatnie, współpracuje z MATLAB-em i z Simulinkiem.

8.4.5. Zadania do samodzielnego rozwiązania

Dla każdego zadania oblicz **trzema sposobami**, to jest z użyciem jednego z solverów `odeXX`, np. `ode23`, z zastosowaniem *Symbolic Math Toolbox* oraz przygotowując i wykorzystując model równania w Simulinku. Nie drukuj wartości liczbowych rozwiązania, ale wykonaj wykresy pokazujące rozwiązanie i jego pierwszą pochodną dla czasu $t \in [0,15]$.

Zadanie A

Dane jest równanie różniczkowe $\dddot{y}+0\cdot\ddot{y}+11\cdot\dot{y}+6\cdot y=0$ (zwyczajne III rzędu) z trzema warunkami początkowymi: $y(0)=0,\ \dot{y}(0)=0,\ \ddot{y}(0)=1$. Układ równań I rzędu będzie miał trzy wiersze, a trzeci wiersz należy obliczyć w oparciu o przykład rozwiązania równania różniczkowego zwyczajnego III rzędu z podrozdziału 8.3.

Zadanie B

Dane jest równanie różniczkowe zwyczajne III rzędu $\dddot{y}+11\cdot\dot{y}=0$ z trzema warunkami początkowymi: $y(0)=0,\ \dot{y}(0)=0,\ \ddot{y}(0)=1$. Przeczytaj też uważnie opis zadania A.

Zadania C

Dla równania różniczkowego zwyczajnego III rzędu (8.3-1) wykonaj obliczenia i wykresy dla czasu $t\in[0,15]$, dla każdego z podanych warunków początkowych:

1. $y(0)=-1,\ \dot{y}(0)=0,\ \ddot{y}(0)=0$
2. $y(0)=+1,\ \dot{y}(0)=0,\ \ddot{y}(0)=0$
3. $y(0)=0,\ \dot{y}(0)=0,\ \ddot{y}(0)=0$
4. $y(0)=-0.5,\ \dot{y}(0)=0,\ \ddot{y}(0)=0$
5. $y(0)=-0.5,\ \dot{y}(0)=0,\ \ddot{y}(0)=-1$

Po wykonaniu każdego z zadań należy sprawdzić, czy dla $t = 0$ wartości funkcji i jej pierwszej pochodnej na wykresie są zgodne z zadanymi warunkami początkowymi.

8.5. Całkowanie i różniczkowanie

MATLAB umożliwia wykonywanie całkowania i różniczkowania metodami numerycznymi. Całkowanie i różniczkowanie analityczne wymaga biblioteki *Symbolic Math Toolbox*.

Zakupienie biblioteki *Symbolic Math Toolbox* umożliwia przekształcanie wyrażeń matematycznych w postaci symbolicznej, w tym całkowanie i różniczkowanie.

8.5.1. Całkowanie numeryczne

Całki oznaczone można obliczać przy użyciu funkcji `integral`, np. aby obliczyć całkę oznaczoną:

$$q=\int_0^1\frac{1}{1+x^2}dx \tag{8.5.1-1}$$

należy zdefiniować funkcję anonimową obliczającą wartość funkcji podcałkowej i wykorzystać ją jako parametr w poleceniu `integral`. Dodatkowe parametry `integral` to dolna i górna granica całkowania.

```
>> fun=@(x) 1./(1+ x.^2)
fun =
  function_handle with value:
    @(x)1./(1+x.^2)
>> q1=integral(fun,0,1)
q1 =
    0.7854
```

Funkcje `integral2` i `integral3` obliczają odpowiednio wartości całek podwójnych i potrójnych. Wartość całki można również obliczyć metodą trapezów (funkcje `trapz`, `cumtrapz`). Więcej informacji podają polecenia `doc` *nazwa_funkcji* oraz `doc funfun`.

Nie należy używać funkcji `quad`, `quadgk`, `quadl`, `quadv`, `quad2d` oraz `dblquad`, gdyż nie będą one dostępne w przyszłych wydaniach MATLAB-a.

8.5.2. Całkowanie analityczne — Symbolic Math Toolbox

Całkowanie analityczne z użyciem *Symbolic Toolbox* pokazano na przykładzie całki oznaczonej (8.5.1-1).

Wartość całki oblicza się następującym zestawem poleceń:

```
>> syms x                    % deklarowanie x jako zmiennej niezależnej
>> y = int(1./(1+ x.^2))     % całka nieoznaczona
       y =atan(x)
>> y = int(1./(1+ x.^2),0,1)    % całka oznaczona, granice x [0,1]
       y =1/4*pi
```

Funkcja `int` podaje rezultat całkowania w postaci wzoru. Jeśli w wywołaniu funkcji `int` nie podano granic całkowania, to otrzymuje się rozwiązanie analityczne dla całki nieoznaczonej. Jeśli poda się granice całkowania, to obliczana jest wartość dokładna całki oznaczonej. Tutaj obliczenia wykonano dla $x \in [0,1]$. Wynik obliczeń całki oznaczonej jest taki sam jak wynik uzyskany w podrozdziale 8.5.1.

Osobliwość w wyrażeniu podcałkowym może spowodować zaburzenie procesu obliczeń, co prowadzi do błędnych wyników. Zaleca się uprzednie przekształcenie takiego wyrażenia do postaci niezawierającej osobliwości.

8.5.3. Różniczkowanie numeryczne i analityczne

Funkcja `diff` oblicza różnice pomiędzy sąsiadującymi elementami wektora. Jeśli wektory x i y zawierają współrzędne kolejnych punktów krzywej, to przybliżone wartości pochodnej uzyskuje się ze wzoru: `dydx = diff(y)./ diff(x)`. Funkcja `gradient` oblicza przybliżoną wartość gradientu.

Korzystając z *Symbolic Math Toolbox*, można wyznaczyć postać analityczną pochodnej, np. pochodną wyrażenia $y = x(x^3 - 1)$ można obliczyć następująco:

```
syms x                  % x zmienna dla Symbolic Math Toolbox
y=diff(x.*(x.^3-1))     % obliczenie pochodnej dla wyrażenia
```

rozwiązaniem jest `y =4*x^3 -1`.

8.6. Dekompozycja macierzy

Dekompozycja (rozkład) macierzy pozwala na zwiększenie dokładności i przyspieszenie obliczeń macierzowych, a w szczególności w przypadku rozwiązywania macierzowych równań liniowych, w tym z macierzami zespolonymi. W MATLAB-ie można dokonać dekompozycji macierzy na kilka sposobów. Zależnie od zastosowanego algorytmu dekompozycji uzyskuje się:

- `lu` — dwie macierze trójkątne: dolna `L` i górna `U`,
- `chol` — macierz trójkątną górną `R` — rozkład Cholesky'ego,
- `qr` — macierz ortogonalną i trójkątną górną `Q R`,
- `svd` (ang. *singular value decomposition*) — macierz diagonalną `S` i macierze unitarne `U V`.

Algorytmy działające na macierzach wykorzystują zazwyczaj taki algorytm dekompozycji macierzy, który w danych warunkach jest najefektywniejszy.

Poniżej podano przykłady dekompozycji macierzy `A` zdefiniowanej następująco:

```
>> A=[1 0 0; 1 .0001 0; 1 1 9999]
A =
   1.0e+03 *
    0.0010         0         0
    0.0010    0.0000         0
    0.0010    0.0010    9.9990
```

8.6.1. Dekompozycja LU

Rozkład *LU* (ang. *lower upper*) jest stosowany przy rozwiązywaniu układów równań liniowych. Funkcja `lu` tworzy dwie macierze trójkątne *LU* takie, że $A = LU$. *L* jest macierzą trójkątną dolną (ang. *lover*) z jedynkami na przekątnej głównej, a *U* jest macierzą trójkątną górną (ang. *upper*). W MATLAB-ie dla poprawienia dokładności i szybkości obliczeń macierz *L* jest permutowana. Po permutacji *L* może już nie być macierzą trójkątną, co pokazano w przykładzie 8.6.1-1.

Przykład 8.6.1-1

```
>> [l,u]=lu(A)
l =
1.0000  0  0
1.0000  0.0001  1.0000
1.0000  1.0000  0

u =
1.0e+003 *
0.0010  0  0
0  0.0010  9.9990
0  0  -0.0010
```

Poprawność rozkładu można sprawdzić, wykonując mnożenie `l*u`. Dekompozycja `LU` może być wykorzystywana w funkcjach, np.:

```
inv(A) = inv(u)*inv(l)          % macierz odwrotna
det(A) = det(l)*det(u)          % wyznacznik
x = u\(l\b)                     % rozwiązanie równania A*x=b
```

8.6.2. Rozkład Cholesky'ego

Jeśli macierz kwadratowa *A* jest *symetryczna* oraz *dodatnio określona*, to istnieje rozkład macierzy *A* na iloczyn macierzy trójkątnej i jej transpozycji, nazywany rozkładem Cholesky'ego. Rozkład Cholesky'ego można stosować też do macierzy *zespolonych*, *Hermitowskich*, *dodatnio określonych*.

Macierz `R=chol(A)` spełnia warunek: `A=R'*R`, gdzie `R` to macierz trójkątna górna.

Więcej podaje `doc(chol)`.

Przykład 8.6.2-1

```
>> P=pascal(4),     R=chol(P)
P =
     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20
R =
     1     1     1     1
     0     1     2     3
     0     0     1     3
     0     0     0     1
```

8.6.3. Dekompozycja QR

Rozkład na macierze ortogonalną `Q` i trójkątną górną `R` realizuje funkcja `qr`:

```
>> [q,r]=qr(A)
q =
-0.5774  -0.4083  -0.7071
-0.5774  -0.4082  0.7071
-0.5774  0.8165  -0.0001
r =
1.0e+003 *
-0.0017  -0.0006  -5.7729
0  0.0008  8.1641
0  0  -0.0007
```

Poprawność rozkładu można stwierdzić, wykonując mnożenie `q*r`.

Rozkład *QR* zastosowano w funkcjach `orth`, `null`, `eig` oraz przy dzieleniu macierzy prostokątnych (np. przy rozwiązywaniu nadokreślonego układu równań).

8.6.4. Obliczenia równoległe w algebrze liniowej

MATLAB obsługuje obliczenia równoległe dla niektórych funkcji. Są one automatycznie uruchamiane — bez potrzeby instalowania *Parallel Computing Toolbox*.

Można się o tym przekonać, obserwując okno *Task Managera* podczas używania funkcji `lu` lub `qr` na dużych tablicach o podwójnej precyzji, mających 10 000 i więcej elementów. W tych warunkach obserwuje się też znaczny wzrost prędkości obliczeń.

8.7. Wartości własne i wektory własne

Wartości własne i *wektory własne* oblicza się przy pomocy funkcji `eig`. Wywołując ją w postaci `Lambda=eig(A)`, uzyskuje się tylko wektor wartości własnych. Natomiast przy wywołaniu `[x,Lambda]=eig(A)` obliczany jest *wektor własny* `x` oraz λI — *wartości własne* umieszczone na przekątnej głównej macierzy `Lambda`. Więcej informacji podaje `doc eig`.

8.8. Analiza funkcji

8.8.1. Minimum, maksimum i miejsca zerowe funkcji

Do analizy funkcji używa się między innymi:

- `fplot` — wykonuje wykres funkcji; pozwala lepiej określić punkt startowy dla algorytmu poszukującego miejsc zerowych lub minimum funkcji oraz zweryfikować otrzymane rozwiązania;
- `fminbnd` — wyznacza minimum funkcji jednej zmiennej; aby znaleźć *maksimum*, szuka się minimum funkcji ze zmienionym znakiem;
- `fminsearch` — wyznacza minimum lokalne funkcji wielu zmiennych metodą Nelder-Mead w otoczeniu zadanego punktu startowego;
- `fzero` — wyznacza miejsce, w którym funkcja zmienia znak; w podrozdziale 8.8.3 podano przykład wykorzystania tej funkcji;
- `roots` — wyznacza pierwiastki rzeczywiste i zespolone wielomianu:

```
[x,fval,exitflag] = fminbnd(fun,x1,x2,options,P1,P2,…)
```

8.8.2. Obliczanie zer wielomianu

Wielomian trzeciego stopnia $y = x^3 + x^2 - 3x - 3$ zapisano w pliku *yy.m*:

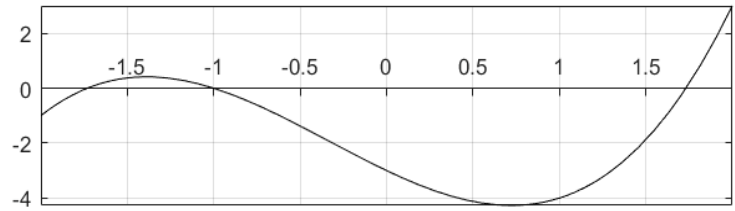
```
function y=yy(x)
% oblicza wartość wielomianu, zmienna x może być wektorem
y=x.^3 +x.^2 -3.*x -3;    % wielomianu
```

Wykres wykonano funkcją `fplot`, jak opisano w przykładzie 8.8.2-1.

Przykład 8.8.2-1

```
fplot(@yy,[-2,2],'k'), grid on
set(gca,'XAxisLocation','origin')    % ustawia oś x na wysokości zero
```

Funkcja `fplot(@yy,[-2,2],'k')` wykonuje wykres funkcji `yy`, symbol `@` tworzy uchwyt tej funkcji, `'k'` to kolor czarny (ang. *blacK*). Oś poziomą `x` ustawiono na wysokości `y=0`, czyli na poziomie początku układu współrzędnych (ang. *origin*). Wykres z rysunku 8.8.2-1 przecina poziom `y=0` w trzech punktach. Są to trzy pierwiastki — miejsca zerowe wielomianu:



Rysunek 8.8.2-1. *Miejsca zerowe wielomianu w przedziale [–2, 2]*

Pierwiastki wielomianu oblicza funkcja `roots`. Parametrem `roots` jest wektor zawierający współczynniki wielomianu.

```
format long       % zwiększenie liczby wyświetlanych cyfr
zera=roots([1 1 -3 -3])    % współczynniki wielomianu
```

W wyniku otrzymuje się wartości: *–1.73205080756888*, *–1*, *1.73205080756888*.

8.8.3. Równanie nieliniowe i źle uwarunkowane

Rozważmy problem znalezienia miejsc zerowych wielomianu 13. stopnia:

$$(x+2)(x^2-1)^6 - 3\cdot 10^6 x^{11} = 0 \qquad (8.8.3\text{-}1)$$

Problem jest trudny numerycznie [Bjorck, Dahlquist, 1987]. Wynika to ze stosunkowo wysokiego stopnia wielomianu, w którym dodatkowo wprowadzono wyrażenie $3\cdot 10^6 x^{11}$, zmieniające liczbę i położenie miejsc zerowych. Problem można rozwiązać, korzystając z funkcji `roots`, obliczającej zera wielomianów, ale poznanie funkcji `fzero` umożliwi szukanie zer funkcji w postaci niejawnej f(*x*)=0. Wyrażenie (8.8.3-1) zapisano jako funkcję anonimową. Działania matematyczne zapisano jako tablicowe (z kropką), co umożliwia wykonywanie obliczeń na wektorach i macierzach:

```
f3=@(x) (x+2).*(x.*x-1).^6-(3e-6).*x.^11    % funkcja anonimowa, f3 to uchwyt
```

Po kilku próbach stwierdzono, że zera tej funkcji będą poszukiwane w przedziale $x \in [-2.02,\ +1.52]$. Na rysunku 8.8.3-1 pokazano wykres funkcji (8.8.3-1) wykonany przez:

```
fplot(f3,[-2.02, 1.52]),  set(gca,'XAxisLocation','origin')
% origin to punkt (0,0) na wykresie, os "x" jest na poziomie y=0 (od MATLAB 2015b)
```

Następnie użyto funkcję `fzero`. Poczynając od punktu startowego, `fzero` poszukuje punktu, w którym badana funkcja zmienia znak. Oznacza to, że `fzero`:

- nie wykrywa zer, przy których nie ma zmiany znaku, np. *x* = 0 dla *x***x*,
- wykrywa tylko jedno zero,
- kolejne zera wykrywa po zmianie punktu startowego.

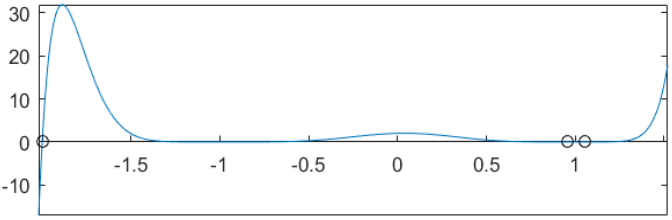
Startując kolejno z *x – 5*, *x = 1*, *x = 1.3*, wykonano:

```
a1=fzero(f3,-5),  2=fzero(f3,1), a3=fzero(f3,1.3)
hold on, plot(a3,0,'ko',a2,0,'ko',a1,0,'ko'),hold off    % k to kolor czarny
```

i wykryte trzy miejsca zerowe zaznaczono kółeczkami na rysunku 8.8.3-1.

Uzyskane wartości to:

```
a1= 1.053416973822623, a2 =– 2.000008427805969, a3 = 0.952998658572976.
```



Rysunek 8.8.3-1. *Wykres badanej funkcji z zaznaczonymi trzema miejscami zerowymi*

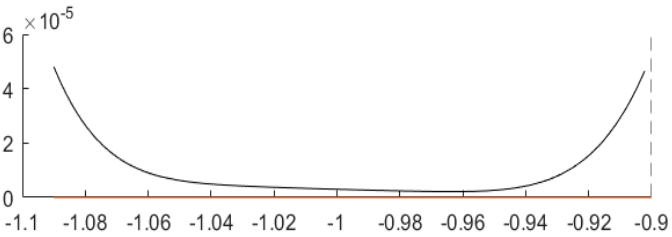
Na rysunku 8.8.3-1 zaznaczono ich położenie kółkami, poprzez użycie poleceń:

```
hold on, plot(a3,0,'ko',a2,0,'ko',a1,0,'ko'),hold off    % k to kolor czarny
```

Wobec braku pierwiastków w przedziale `[-1, -0.9]` wykonano dodatkowy wykres, wywołując funkcję `fplot` i `plot`, jak poniżej:

```
t=[-1.09,-0.9];hold on, fplot(f3,t,'k'),plot(t,[0,0]), hold off
```

Uzyskany wykres (rysunek 8.8.3-2) potwierdził brak miejsc zerowych w tym przedziale.



Rysunek 8.8.3-2. *Badana funkcja nie ma miejsc zerowych w przedziale [–1.1, –0.9]. Należy zwrócić uwagę na zmianę skali na osiach*

Zweryfikowanie wyników obliczeń przy użyciu funkcji `roots` pozostawiamy Czytelnikowi. Będzie to wymagało obliczenia wartości wszystkich współczynników wielomianu danego równaniem (8.8.3-1), np. z użyciem funkcji `conv` lub *Symbolic toolbox*. Sposób użycia `fzero` i `roots` podaje `doc` *`nazwa_funkcji`*.

8.8.4. Wielomian i funkcje wielomianowe

Wybrane funkcje działające na wielomianach zamieszczono w tabeli 8.8.4-1. Współczynniki wielomianów są zapisywane w wektorach. Pierwszym elementem wektora jest współczynnik przy zmiennej w najwyższej potędze. Ostatni element zawiera wyraz wolny. Z uwagi na wyraz wolny długość wektora jest o jeden większa od stopnia wielomianu. Wielomian $p(y)=y^4+2y+5$ można zapisać w postaci wektorowej jako `p = [1 0 0 2 5]`.

Tabela 8.8.4-1. *Wybrane funkcje wektorowe i wielomianowe*

Nazwa	Opis funkcji	Nazwa	Opis funkcji
`roots`	pierwiastki wielomianu	`poly`	wielomian charakterystyczny
`cross`	iloczyn wektorowy	`dot`	iloczyn skalarny
`conv`	iloczyn wielomianów	`deconv`	iloraz wielomianów
`polyder`	pochodna wielomianu	`residue`	residuum
`polyfit`	aproksymacja wielomianu	`polyeig`	wartości własne wielomianu
`polyval`	wartość wielomianu	`polyvalm`	wartość wielomianu w sensie macierzowym

8.9. Interpolacja i aproksymacja

Zadanie aproksymacji lub interpolacji można wygodnie zrealizować przy użyciu *Basic Fitting Interface*, którego okno pokazano w podrozdziale 8.9.5.

8.9.1. Interpolacja wielomianowa

Zadane są wartości funkcji w punktach dyskretnych. Problem interpolacji to poszukiwanie funkcji, która przybliża funkcję zadaną. Wartości obu tych funkcji w zadanych punktach są identyczne. Zadane punkty są nazywane węzłami interpolacji, a poszukiwana funkcja — funkcją interpolującą. Określenie tej funkcji pozwala na obliczanie przybliżonych wartości funkcji zadanej w punktach pomiędzy węzłami.

Klasycznym rozwiązaniem tego problemu jest **wielomian interpolacyjny** Lagrange'a stopnia *n*, który przechodzi przez *n+1* zadanych punktów. Ponieważ wielomian jest funkcją ciągłą, można obliczyć jego wartość w dowolnym punkcie. Jednak wartość przybliżona może znacznie odbiegać od rzeczywistej wartości zadanej funkcji. Wadą interpolacji wielomianowej są duże oscylacje wielomianu interpolacyjnego (linia kropkowa — na rysunku 8.9.4-1).

8.9.2. Aproksymacja wielomianowa

Jeśli liczba zadanych punktów jest większa od stopnia wielomianu *n* plus jeden, to zazwyczaj nie można przeprowadzić krzywej przez wszystkie zadane punkty. Mamy wtedy do czynienia z zadaniem **aproksymacji**. Rozwiązaniem może być wielomian, który najlepiej przybliża zadaną krzywą w sensie minimum błędu **średniokwadratowego**. W MATLAB-ie powyższe zadanie rozwiązuje funkcja `polyfit`, która wyznacza współczynniki poszukiwanego wielomianu aproksymującego:

```
p = polyfit(x,y,n)
```

Wartości takiego wielomianu w dowolnych punktach zapisanych w wektorze `xx` oblicza funkcja `polyval`.

```
ypi= polyval(p,xx)
```

Argumenty obu tych funkcji są następujące:

- `x`, `y` — wektory punktów zadanych (węzły interpolacji, aproksymacji),
- `n` — stopień wielomianu interpolującego lub aproksymującego,
- `p` — wektor współczynników wielomianu,
- `ypi` — wektor wartości wielomianu w dowolnych punktach `xx`.

8.9.3. Funkcja sklejana — spline function

Interpolacja za pomocą funkcji sklejanych ma właściwości lepsze od interpolacji za pomocą wielomianów. Jej teorię opracowano w latach 40.

Cechą charakterystyczną funkcji sklejanej jest nieciągłość jej wyższych pochodnych w węzłach. Na przykład trzecia pochodna jest nieciągła w punktach węzłowych dla interpolacji za pomocą funkcji sklejanych III rzędu (ang. *cubic spline*).

W MATLAB-ie do utworzenia funkcji sklejanej przechodzącej przez węzły zadane wektorami x, y oraz do obliczenia wartości funkcji sklejanej w punktach zadanych wektorem xx używa się funkcji spline. Wywołuje się ją w następującej postaci:

```
sp = spline (x,y,xx)
```

Współrzędne węzłów są podane w wektorach x, y, a wynikiem jest wektor sp, który zawiera obliczone wartości funkcji sklejanej w punktach wektora xx. Bardziej zaawansowane rodzaje interpolacji za pomocą funkcji sklejanych można stosować przy wykorzystaniu opcjonalnej biblioteki MATLAB-a *Spline Toolbox*.

8.9.4. Przykład interpolacji i aproksymacji

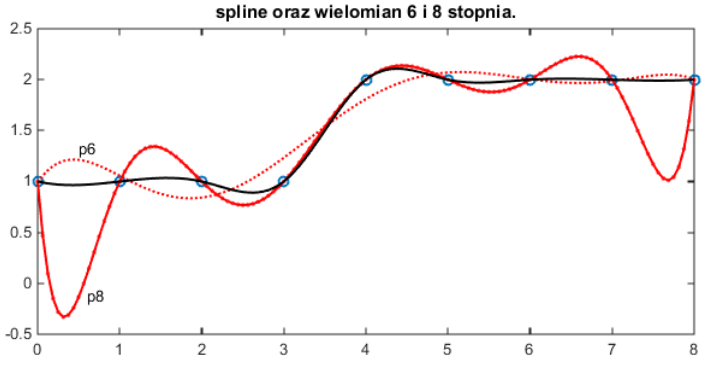
Poniżej zamieszczono plik definiujący zadaną funkcję. Wartości funkcji dla kolejnych całkowitych argumentów od 0 do 8 zadano wektorem y = [1 1 1 1 2 2 2 2 2] (kółka na rysunku 8.9.4-1). Uzyskanie dobrego przybliżenia takiej funkcji jest trudne. Przetestowano funkcje MATLAB-a realizujące przybliżenie funkcji zadanej z użyciem wielomianów oraz funkcji sklejanej.

Przykład 8.9.4-1

Aproksymacja funkcji schodkowej z użyciem spline i wielomianów stopnia 6 i 8.

```
y = [1 1 1 1 2 2 2 2 2];    % zadano punkty funkcji schodkowej
x = [0:1:8];                % w przedziale [0, 8]
xx = 0:.0625:8 ;      % podział na wiele podprzedziałów
p8=polyfit(x,y,8);    % interpolacja wielomianowa (Lagrange'a)
p6=polyfit(x,y,6);    % aproksymacja wielomianowa
% wektory p6 i p8 zawierają współczynniki wielomianów
yp6=polyval(p6,xx);   % interpolacja, wartości wielomianu w punktach xx
yp8=polyval(p8,xx);
sp=spline(x,y,xx);    % interpolacja, wartości funkcji sklejanej w punktach xx
plot(x,y,'o',xx,yp6,':r',xx,yp8,'.-r',xx,sp, 'k','linewidth',1.5)
title ('spline oraz wielomian 6 i 8 stopnia.')
text(0.5,polyval(p6,0.5)+0.1,'p6')    % tekst p6 powyżej wykresu
text(0.6,polyval(p8,0.5),'p8')    % tekst p8 obok wykresu
```

Funkcję zadaną dziewięcioma punktami interpolowano funkcją sklejaną stopnia III i wielomianem stopnia VIII oraz aproksymowano wielomianem stopnia VI. Na wykresie (rysunek 8.9.4-1) widać, że wielomian niskiego stopnia ma mniejsze oscylacje, ale nie przechodzi przez wszystkie zadane punkty.



Rysunek 8.9.4-1. *Interpolacja i aproksymacja zadanej funkcji. Zadane punkty oznaczono na wykresie kółkami*

Funkcja sklejana i wielomian VIII-go stopnia przechodzą przez punkty węzłowe, jednak wielomian robi to kosztem znacznego zwiększenia oscylacji (tzw. efekt Rungego).

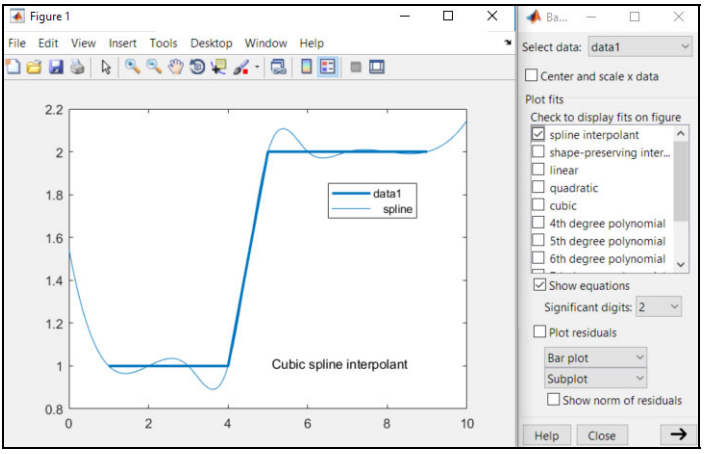
Tylko funkcja sklejana spełnia równocześnie dwa postulaty: gwarantuje dużą dokładność i małe oscylacje. Może też służyć do dość dobrego oszacowania pochodnej zadanej funkcji.

8.9.5. Interpolacja i aproksymacja w oknie Basic Fitting

Interpolację i aproksymację można wykonać bez programowania, korzystając z okna interfejsu **Basic Fitting** (rysunek 8.9.5-1). Dostęp do tego interfejsu uzyskuje się poprzez okno graficzne. Pierwszym krokiem jest narysowanie wartości zadanych (przyszłe węzły interpolacji lub aproksymacji). Dla danych z przykładu 9.4-1 wystarczy wykonać.

```
y = [1 1 1 1 2 2 2 2 2];    plot(y)
```

Następnie w oknie graficznym, w którym pojawił się rysunek, należy wybrać z menu opcję *Tools/Basic Fitting*. Otwiera się wtedy okno z nagłówkiem *Basic Fitting*, z interfejsem umożliwiającym bardzo łatwe wykonanie różnego rodzaju interpolacji bądź aproksymacji zadanych wartości funkcji — przez zaznaczenie myszą wybranych algorytmów aproksymacji. Większe możliwości oferuje *Curve Fitting Toolbox*.



Rysunek 8.9.5-1. *Interpolacja i aproksymacja funkcji z przykładu 8.9.4-1 z użyciem interfejsu Basic Fitting*

8.10. Analiza statystyczna

W wersji standardowej MATLAB oferuje kilkanaście funkcji przydatnych do analizy statystycznej i wizualizacji danych (tabela 8.10-1). Funkcje te działają na wektorach lub kolumnach macierzy. Jeśli argumentem funkcji jest wektor, to jej wynik jest skalarem. Natomiast jeśli argumentem jest macierz, to jako rezultat uzyskuje się wektor. Dalsze informacje podaje polecenie `doc datafun`. Poszerzenie możliwości analizy statystycznej można uzyskać, wykorzystując *Statistics Toolbox*.

Tabela 8.10-1. *Wybrane funkcje statystyczne*

Nazwa	Opis funkcji	Nazwa	Opis funkcji
max	maksymalny element	min	minimalny element
mean	wartość średnia	median	wartość środkowa
std	odchylenie standardowe	sort	uporządkowanie rosnące
corrcoef	współczynnik korelacji	hist, histc	histogram
sum	suma elementów	prod	iloczyn elementów
cumsum	suma kumulatywna	cumprod	iloczyn kumulatywny
cov	kowariancja	var	wariancja

Przykładowo funkcja max podaje wartość największego elementu wektora lub największego elementu każdej z kolumn macierzy. Największy element macierzy A można odszukać jako max(A(:)).

Funkcje rand, randn generują odpowiednio liczby pseudolosowe o rozkładzie równomiernym lub o rozkładzie normalnym z przedziału [0, 1].

Funkcje te można wywołać w sposób podany poniżej:

```
xy=rand(2,50);          % rozkład równomierny, 2x50 próbek
xi=randi(100,[1,50]);   % liczby<100, integer, 50 próbek
ss=randn(1,5000);       % rozkład normalny, 5000 próbek
mean(ss), std(ss)       % wartość średnia i odchylenie standardowe dla 5000 próbek
```

Funkcja r=randi(imax,n,classname) generuje macierz $n \times n$ liczb pseudolosowych typu *classname* (np. single, double, uint8).

Niekiedy zachodzi potrzeba wielokrotnego wygenerowania tego samego ciągu liczb losowych. Można to uzyskać przez utworzenie uchwytu do klasy @RandStream lub przez użycie nasienia seed.

Przykład 8.10-1

Powtórne wygenerowanie tego samego ciągu liczb losowych z użyciem RandStream.

```
h=RandStream.getGlobalStream    % uchwyt do klasy @RandStream
% h=RandStream.getDefaultStream w starszych wersjach MATLAB-a
myState=h.State;    % zapisanie stanu strumienia
r1=randn([1,5])     % generowanie 5 liczb losowych
r2=randn([1,5])     % generowanie kolejnych 5 liczb
h.State=myState;    % odtworzenie zapamiętanego stanu strumienia mState
r3=randn([1,5])     % powtórne wygenerowanie liczb losowych jak w r1
```

Innym sposobem zapewnienia kontroli nad powtarzalnością wygenerowanego ciągu liczb losowych jest użycie rng i nasienia seed, jak w przykładzie poniżej.

Przykład 8.10-2

Powtórne wygenerowanie tego samego ciągu liczb losowych z użyciem seed

```
format short, format compact    % 4 cyfry dziesiętne, bez pustych linii
seed =1;    % ziarno nieujemna liczba całkowita, np. 0,1,2...
rng(seed);    % użycie ziarna w generatorach
r1=randn([1,5]), r2=randn([1,5])    % generowanie liczb losowych
```

```
rng('shuffle'), r1shuffle=randn([1,5])    % ziarno wygenerowane z użyciem zegara
rng(seed);    % powtórnie użyto seed = 1
r1seed1=randn([1,5]), , r2=randn([1,5])    % wygenerowanie tego samego liczb
```

Użycie tej samej wartości ziarna w `rng(seed)` powoduje wygenerowanie tego samego ciągu liczb losowych. Z kolei `rng('shuffle')` używa aktualnej daty i czasu do utworzenia ziarna, co ma zapewnić niepowtarzalność otrzymanych ciągów liczb pseudolosowych.

Wykaz wszystkich dostępnych w MATLAB-ie algorytmów generowania liczb pseudolosowych można otrzymać za pomocą `RandStream.list`.

8.11. Analiza sygnałów

Próbki zdyskretyzowanych sygnałów mogą być zapisywane w wektorach lub (dla sygnałów wielokanałowych) w macierzach. Można je przechować w skompresowanych *mat-plikach* binarnych lub w *formacie Excela*. W tabeli 8.11-1 podano wybrane funkcje używane do przetwarzania sygnałów, w tym szybką transformatę Fouriera (ang. *Fast Fourier Transform* — FFT). Znacznie więcej możliwości w zakresie przetwarzania sygnałów udostępnia *Signal Processing Toolbox*.

Tabela 8.11-1. *Wybrane funkcje przetwarzania sygnałów w MATLAB-ie*

Nazwa	Opis funkcji
`conv`, `conv2`	splot i mnożenie wektorów
`deconv`	dzielenie wektorów
`cov`	kowariancja
`fft`, `fft2`, `fftn`	szybka transformata Fouriera
`ifft`, `iftt2`, `ifftn`	odwrotna transformata Fouriera
`fftshift`	przesuwa składową stałą do środka macierzy
`decimate`	rozrzedzanie próbek (skraca wektor)
`interp`	zagęszczanie próbek (wydłuża wektor)
`abs`	amplituda wektora
`angle`	faza wektora (kąt)

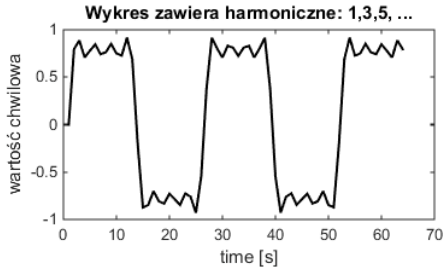
Funkcje `y=decimate(x,r)` i `y=interp(x,r)` przetwarzają ciąg próbek sygnału `x` na wektor `r` razy krótszy lub dłuższy, o elementach odpowiadających mniejszej lub większej częstotliwości próbkowania. Przed przetworzeniem wektor jest filtrowany dolnoprzepustowo, tak aby ograniczyć zakłócenia powodowane wyższymi harmonicznymi (ang. *aliasing*). Funkcja `decimate(x,r)` realizuje filtr Czebyszewa I typu, rzędu VIII. Dodatkowe parametry zmieniają rząd filtru lub jego typ na FIR.

W folderze *matlab\toolbox\matlab\demos* zamieszczono przykład użycia analizy Fouriera do badania wieloletnich pomiarów aktywności Słońca (cykle krótkie i wieloletnie).

8.11.1. Przykład analizy przebiegu odkształconego

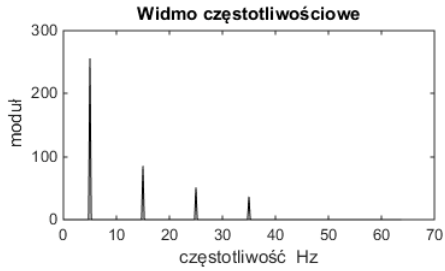
Poniżej zamieszczono przykład zastosowania funkcji `fft` do obliczania widma częstotliwościowego niesinusoidalnej funkcji okresowej. Wykres analizowanej funkcji pokazano na rysunku 8.11.1-1.

Rysunek 8.11.1-1. *Analizowany przebieg odkształcony*



Funkcja `fft` realizuje algorytm szybkiej dyskretnej transformaty Fouriera. Użyto jej do wykonania analizy widma przebiegu odkształconego o częstotliwości 5 Hz, który zawiera harmoniczne 1, 3, 5, 7. Rezultaty obliczeń uzyskane w przykładzie 8.11.1-1, z wykorzystaniem funkcji `fft`, przedstawiono na rysunku 8.11.1-2.

Rysunek 8.11.1-2. *Harmoniczne 1, 3, 5, 7 w widmie częstotliwościowym*



Przykład 8.11.1-1

Analiza przebiegu odkształconego:

```
n=512        % liczba próbek, najlepiej potęga liczby 2
t=1/128;     % krok dyskretyzacji czasu: okres próbkowania [s]
k=0:n-1;     % numery próbek  k=0 ... 511
fo=5;        % częstotliwość podstawowa [Hz]
fprintf('harmoniczna podstawowa    % g Hz \n',fo)
fun=  sin(2*pi*fo*(k*t))       + sin(2*pi*(3*fo)*(k*t))/3 …
    + sin(2*pi*(5*fo)*(k*t))/5 + sin(2*pi*(7*fo)*(k*t))/7;
plot (fun(1:n/8))        % tylko n/8 punktów
xlabel('time [s]'), ylabel('wartość chwilowa')
title('Wykres zawiera harmoniczne: 1,3,5, …')

widmo=fft(fun);                        % analiza częstotliwościowa
modul=abs(widmo); freq_hz=k*(1/(n*t));
figure,plot(freq_hz(1:n/2), modul(1:n/2))    % połowa widma 1:n/2
title('Widmo częstotliwościowe')
xlabel('częstotliwość  Hz'), ylabel('moduł')
```

Rozdział 9.

Przetwarzanie obrazów

Środowisko pakietu MATLAB posiada szybkie algorytmy do operacji na dużych macierzach. Pozwala na tworzenie zaawansowanych procedur do komputerowej wizualizacji wyników pomiarów i obliczeń, a także do efektywnego przetwarzania grafiki.

Gotowe narzędzia do akwizycji, tworzenia i przetwarzania obrazów są dostępne w *Image Processing Toolbox™*, *Image Acquisition Toolbox™*, jak również w *Bioinformatics Toolbox™*. Służą one do rozwiązywania złożonych problemów graficznych w wielu dziedzinach, jak lotnictwo i przemysł obronny, teledetekcja, biotechnologia, medyczne i naukowe obrazy oraz materiałoznawstwo. Szczegółowe omówienie tych toolboksów wykracza poza ramy tej książki.

Niniejszy rozdział przedstawia podstawowe możliwości przetwarzania grafiki rastrowej bez użycia toolboksów. Raster to sieć, w której oczkach znajdują się piksele. Obraz rastrowy jest opisany przez podanie koloru każdego piksela tworzącego mapę bitową obrazu.

Grafiki wektorowej nie omawia się w tej książce.

9.1. Zapis i odczyt obrazów, liczby 8- i 16-bitowe bez znaku

Funkcje stosowane do zapisu i odczytu obrazu to:

- `imread`, `imwrite` — odczyt i zapis pliku graficznego w różnych formatach,
- `image`, `imagesc` — wyświetla obraz, wyświetla i skaluje obraz,
- `imfinfo` — analizuje informacje z pliku graficznego,
- `exifread` — podaje ustawienia aparatu fotograficznego.

Pliki graficzne wczytuje się poleceniami:

```
 X=imread('nazwa_pliku')          % wczytanie obrazu 'true color' lub w poziomach szarości
[X,map]=imread('nazwa_pliku')     % wczytanie obrazu indeksowanego
[X,map]=load('nazwa_pliku')       % wczytanie obrazu jako grafiki z pliku binarnego .mat
```

Obrazy wczytane funkcją `imread` zajmują niewiele miejsca w pamięci z uwagi na zmianę typu danych. W miejsce zazwyczaj używanych liczb 64-bitowych o podwójnej precyzji z zakresu [0, 1], używa się tutaj liczb 8-bitowych z zakresu [0, 255].

Polecenie `imread` rozpoznaje i wczytuje kilkanaście formatów graficznych, a dodatkowo poprawnie interpretuje ich odmiany różniące się liczbą bitów, kodowaniem barw i sposobem kompresji. Wczytanie obrazu za pomocą polecenia `imread` nie powoduje jego wyświetlenia, co daje możliwość przetworzenia obrazu przed jego wyświetleniem.

Przetworzony obraz można zapisać w dowolnie wybranym formacie graficznym za pomocą polecenia `imwrite`, na przykład `imwrite(X,'pict.tif')` zapisze obraz jako plik graficzny *pict.tif* w formacie TIFF.

Do pakietu MATLAB można wczytać obrazy w wielu standardach (patrz `doc imread`), w tym:

- *bmp* — bitmapy: 1-, 4-, 8- i 24-bitowe; może być użyta bezstratna kompresja RLE (ang. *run lengh encoded*),
- *cur* — (ang. *cursor file*),
- *gif* — (ang. *graphics interchange format*) — 1- i 8-bitowe,
- *hdf4* — (ang. *hierarchical data format*),
- *ico* — ikony 1-, 4-, 8-bitowe,
- *jpeg* — (ang. *joint photographic experts group*),
- *jpeg2000* — (ang. *joint photographic experts group 2000*),
- *pbm* — (ang. *portable bitmap*),
- *pgm* — (ang. *portable graymap*),
- *pcx* — (ang. *format Paintbrush*) — 1-, 8- i 24-bitowe,
- *png* — (ang. *portable network graphics*) — 1-bitowe – 48-bitowe,
- *ppm* — (ang. *portable pixmap*),
- *ras* — (ang. *sun raster*),
- *tiff* — (ang. *tagged image file format*) — 1-, 8-, 24-bitowe, z kompresją lub bez,
- *xwd* — (ang. *X Window Dump*).

Biblioteka *Image Processing Toolbox* obsługuje jeszcze więcej formatów.

9.2. Grafika 24-bitowa (true color)

Obrazy *true color* wymagają dużej pamięci, gdyż dla każdego piksela przechowuje się jego barwę w postaci trzech liczb 8-bitowych. Daje to łącznie 24 bity na każdy piksel i oznacza możliwość wyboru spośród ponad 16 milionów barw ($2^{24} = 16777216$). Obrazy *true color* są przechowywane w tablicy trójwymiarowej m×n×3. Powstaje ona ze złożenia trzech warstw o wymiarach m×n. Każda warstwa zawiera informację o intensywności jednego z trzech kolorów RGB (czerwony, zielony, niebieski). Obrazy *true color* wyświetla się za pomocą polecenia `image(X)`.

Przykład 9.2-1

Udostępniony przez NASA obraz *ngc6543a.jpg* jest dostępny bez potrzeby podawania jego ścieżki dostępu. Poniżej pokazano sposób jego wczytania.

```
X=imread('ngc6543a.jpg');    % wczytanie obrazu 'truecolor' do X
image(X)                     % wyświetla obraz na ekranie
```

W przestrzeni roboczej będzie tylko jedna macierz 3-kolumnowa:

```
Name        Size                Bytes  Class    Attributes
X         650x600x3           1170000  uint8
```

Wymiary tego obrazu to 650×600 pikseli. Należy zwrócić uwagę na użycie trzech liczb 8-bitowych typu `uint8` do zapisu składowych RGB barwy każdego piksela. Liczby 8-bitowe zabierają znacznie mniej pamięci w porównaniu z liczbami zmiennoprzecinkowymi `single` lub `double`.

Przykład 9.2-2

Polecenie `imfinfo` podaje dodatkowe informacje o pliku graficznym (w tym dane EXIF — jeśli są dostępne).

```
imfinfo('ngc6543a.jpg')    % odczytanie informacji o rysunku
ans =
           Filename: [1x64 char]
        FileModDate: '01-paź-1996 16:19:44'
           FileSize: 27387
             Format: 'jpg'
      FormatVersion: ''
              Width: 600
             Height: 650
           BitDepth: 24
          ColorType: 'truecolor'
    FormatSignature: ''
    NumberOfSamples: 3
       CodingMethod: 'Huffman'
      CodingProcess: 'Sequential'
            Comment: {[1x69 char]}
```

Należy zwrócić uwagę na wysoki stopień kompresji pliku *.jpg*, mającego tylko 27 387 bajtów — w porównaniu do 1 170 000 bajtów macierzy X.

9.3. Palety barw i obrazy indeksowane

Obrazy mogą być przechowywane także w formie obrazów indeksowanych. Są one gorszej jakości, gdyż w wyniku ograniczenia liczby dostępnych kolorów (256 lub mniej, np. 16) zajmują mniej miejsca. Wybrane kolory są zapisywane w pomocniczej tablicy, zwanej *paletą barw*, w postaci trzech liczb: R, G, B. Barwa punktów obrazu jest określana przez podanie numeru wiersza (indeksu) palety barw, w którym zapisano wybraną kombinację barwy czerwonej *R*, zielonej *G* i niebieskiej *B*.

9.3.1. Palety barw

Palety barw są stosowane w pakiecie MATLAB do barwienia wykresów (aktualnie służy do tego paleta *parula*) oraz do barwienia obrazów. Funkcje *parula*, *jet*, *hsv*, *hot*, *cool*,

spring, *summer*, *autumn*, *winter*, *gray*, *bone*, *copper*, *pink* generują palety, w których podobne barwy sąsiadują ze sobą. Funkcje *lines*, *colorcube*, *prism*, *flag* umieszczają obok siebie krótkie sekwencje kontrastujących kolorów, co niekiedy pozwala dostrzec słabo widoczne elementy obrazu. Paleta *white* przypisuje biały kolor do wszystkich pikseli, co powoduje zniknięcie obrazu bez jego usuwania.

Każda z palet zawiera standardowo 64 kolory, chyba że w wywołaniu poda się inną liczbę kolorów, na przykład `hsv(16)` lub `hsv(512)`. Paleta jest aktywna dopiero po jej zainstalowaniu za pomocą funkcji `colormap('nazwa_palety')`.

Nazwę palety można pominąć lub podać `'default'`. Pasek z aktualnymi barwami wyświetla funkcja `colorbar`, a wykres intensywności barw składowych podaje `rgbplot`.

Przykład 9.3.1-1

Poniżej podano przykład palety `hsv(6)`, składającej się z kolorów: czerwony, żółty, zielony, cyan, niebieski, magenta. W kolumnach podano liczby przypisane do kolorów RGB, a wiersze definiują kolejne barwy palety.

```
paleta1=hsv(6)

paleta1 =

     1     0     0
     1     1     0
     0     1     0
     0     1     1
     0     0     1
```

Do tej palety dodano dwa dodatkowe kolory: morski i szary.

```
paleta2=(paleta1;.49 1 .83; .5 .5 .5)
```

Wygenerowaną tapetę można zainstalować poleceniem `colormap(paleta1)` i wykorzystywać do tworzenia wykresów. Palety, a także obrazy korzystające z tych palet oraz obrazy *true color* można badać i modyfikować z użyciem funkcji:

- `colormap` — instaluje wskazaną macierz jako aktualną paletę barw,
- `colorbar` — wyświetla skalę barw w postaci słupka,
- `brighten` — rozjaśnia lub przyciemnia paletę barw,
- `spinmap` — zamiana barw (rotacja kolorów w palecie barw),
- `rgbplot` — wykres palety barw,
- `hsv2rgb` — transformuje `hsv` na kolory `rgb`,
- `rgb2hsv` — transformuje `rgb` na kolory `hsv`,
- `shading` — cieniowanie,
- `image` — wyświetla obraz bez skalowania,
- `imagesc` — skaluje barwy i wyświetla obraz,
- `colordef` — cieniowanie kolorów tła obiektu `root`.

9.3.2. Obrazy indeksowane

Obraz indeksowany składa się z dwóch części: z 3-kolumnowej palety barw oraz 2-wymiarowej tablicy obrazu, w której zamiast koloru piksela (np. 24 bity) przechowuje się indeks do koloru, czyli 8-bitowy numer wiersza w trójkolumnowej palecie barw, dołączonej do danego obrazu. Zmniejsza to zapotrzebowanie na pamięć.

Obrazy indeksowane są wyświetlane na ekranie za pomocą poleceń:

```
image(X); colormap(map)
```

gdzie `X` jest dwuwymiarową tablicą obrazu, a `map` jest 3-kolumnową paletą barw.

9.3.3. Obrazy w skali szarości, pseudokolor

Obrazy w skali szarości są przechowywane w jednej dwuwymiarowej tablicy i nie muszą mieć własnej palety kolorów.

Przykład 9.3.3-1

Przykładowy obraz (czarno-białą fotografię) można wczytać za pomocą polecenia:

```
load gatlin2
```

Nazwy plików wczytanych przez `load gatlin2` podaje polecenie `whos`.

```
Name        Size            Bytes    Class     Attributes
X           176x260         366080   double
map         64x3            1536     double
```

Obraz będzie prawidłowo wyświetlony po wykonaniu:

```
imagesc(X); colormap(map)
```

gdzie `imagesc(x)` skaluje i wyświetla obraz, a `colormap(map)` instaluje wczytaną mapę kolorów. Do skalowania obrazu będą użyte wartości minimalne i maksymalne pobrane z tablicy `X`. Pominięcie polecenia `colormap(map)` spowoduje nienaturalne zabarwienie obrazu wywołane użyciem domyślnej mapy `default` zamiast poziomów szarości.

W celu zwrócenia uwagi na szczegóły, które są słabo widoczne na oryginalnym, czarno-białym obrazie (lub dla uzyskania specjalnych efektów), można celowo zamienić paletę barw, na przykład poleceniem `colormap(jet)` lub `colormap(copper)`. Efektem jest *pseudokolor*, czyli przypisanie nienaturalnych barw różnym poziomom szarości.

Korzystając z generatora liczb losowych, można tworzyć tablice barwne przetwarzające fotografie na obrazy podobne do prac impresjonistów.

9.3.4. Przekodowywanie obrazów stało- i zmiennoprzecinkowych

Obrazy szare i obrazy *true color* zapisane z użyciem liczb 64-bitowych o podwójnej precyzji z przedziału [0, 1] można przekodować do standardu `uint8` lub `uint16` na liczby z przedziału [0, 255] lub [0, 65535]. Do przetwarzania używa się wzorów:

```
RGB8 = uint8(round(RGB64*255));    % tablica 8-bitowa [0, 255]
RGB16 = uint16(round(RGB64*65535));    % tablica 16-bitowa [0, 65535]
RGB64 = double(RGB8)/255;    % tablica 64-bitowa  [0, 1]
```

W obrazach **indeksowanych** w palecie barwnej używa się liczb 64-bitowych z przedziału [0, 1] i się ich nie przekodowuje. Należy jedynie skorygować używany zakres indeksów obrazu z użyciem podanych niżej wyrażeń:

```
X8 = uint8(X64 - 1);    % 8-bitowa macierz obrazu indeksowanego
X64 = 1 + double(X8);   % 64-bitowa macierz obrazu
```

Zakres dla liczb `uint8` to [0, 255], a dozwolone numery wierszy palety barw to [1, 256].

9.3.5. Przekodowanie obrazu kolorowego na szary

Przekształcenie obrazu **kolorowego na czarno-biały** (w skali szarości) wykonuje się następująco:

- Jeśli obraz jest indeksowany, to wstępnie można zamienić aktualnie używaną paletę barwną na paletę odcieni szarości *gray*: `figure; colormap(gray(256)); image(ss)`. Jeśli wynik nie jest zadowalający, to należy utworzyć nową mapę barwną z podanego poniżej wzoru na obliczanie `ss`.
- Jeśli obraz jest nieindeksowany, to macierz (*true color*) `rgb` typu `double` przekształca się na macierz `ss` skali szarości. Poniższe wzory uwzględniają niejednakową czułość oka na barwę, zgodnie ze standardem NTSC.

```
ss = .2989*rgb(:,:,1)+.5870*rgb(:,:,2)+.1140*rgb(:,:,3);
```

9.4. Przetwarzanie obrazów rastrowych

Profesjonalnym pakietem do przetwarzania obrazów w środowisku MATLAB-a jest *Image Processing Toolbox*. Większość podanych tu algorytmów nie wymaga użycia tego toolboksa.

Wszystkie przykłady dotyczą obrazów czarno-białych, nieindeksowanych. Można je przystosować do obrazów kolorowych, powtarzając kolejne operacje oddzielnie dla każdego koloru RGB, a następnie scalając otrzymane trzy obrazy.

9.4.1. Przygotowanie obrazu do testów

Do testów wybrano powszechnie znany plik graficzny *lena.jpg* o rozdzielczości 256×256. Można go łatwo wyszukać w internecie. Parametry pliku można odczytać za pomocą polecenia `imfinfo('lena.jpg')`.

Przykład 9.4.1-1

Poniżej podano kilka poleceń, które usuwają z pamięci wszystkie zmienne i stare rysunki, wczytują plik graficzny *lena.jpg* i tworzą używane dalej tablice z danymi graficznymi:

```
clear    % usuwa zmienne z przestrzeni roboczej / clear all variables
close all    % zamyka wszystkie okna graficzne / close all figures
X=imread ('lena.jpg');    % wczytanie pliku jako tablicy typu int8
x=single(X);              % konwersja z uint8 na typ single
xmax=max(max(x))          % wyszukanie maksymalnej wartości
```

```
xnorm=x/xmax;    % normalizacja liczb single do zakresu [0,1]
figure, imagesc(x), colormap(gray),title('obraz oryginalny')
figure, imagesc(xnorm), colormap(gray),title('obraz normalizowany')
```

Rezultat: wczytano plik graficzny *lena.jpg* i przekodowano go do postaci:

- `X` — tablica `uint8`, 8-bitowe liczby bez znaku z zakresu [0, 255] są używane, gdy przetwarzanie (np. binaryzacja, negatyw, LUT) nie wymaga mnożenia ani dzielenia;
- `xnorm` — tablica `single`, 32-bitowe liczby zmiennoprzecinkowe z zakresu [0, 1] są używane np. do filtracji, z uwagi na wykonywane operacje mnożenia i dzielenia.

9.4.2. Przetwarzanie punktowe — negatyw

Negatyw obrazu rastrowego (rysunek 9.4.2-1) uzyskuje się, zastępując kolor czarny — białym, a kolor biały — czarnym oraz przeliczając odpowiednio wszystkie pozostałe poziomy szarości. Przeliczanie polega na odejmowaniu aktualnej wartości piksela od wartości przypisanej do koloru białego.

Rysunek 9.4.2-1. *Oryginalny rysunek lena.jpg i jego negatyw*

Przykład 9.4.2-1

Przetworzenie obrazu na negatyw — kontynuacja przykładu 9.4.1-1.

```
figure, imagesc(1-xnorm), colormap(gray),title('negatyw')
imwrite(x,'lena.negat.jpg','jpg')    % zapisuje plik z negatywem
```

9.4.3. Przetwarzanie punktowe — rozjaśnianie i ściemnianie

9.4.3.1. Rozjaśnienie obrazu rastrowego

Pierwiastkowanie elementów normalizowanej macierzy `xnorm` powoduje rozjaśnienie obrazu, jak pokazano na rysunku 9.4.3.1-1. Pierwiastkowanie nie zmienia wartości zer ani jedynek w macierzy obrazu. Nie trzeba więc powtórnie skalować tej macierzy.

Rysunek 9.4.3.1-1. *Obraz oryginalny xnorm (środkowy) oraz obrazy rozjaśnione: norm=0.5 (lewy) i 0.2 (prawy)*

Przykład 9.4.3.1-1

Przetworzenie obrazu — rozjaśnienie. Kontynuacja przykładu 9.4.1-1:

```
figure,imagesc(xnorm), colormap gray,
xsq=sqrt(xnorm); figure,imagesc(xsq), colormap gray,
xsq=xnorm.^0.2; figure,imagesc(xsq), colormap gray,
```

9.4.3.2. Przyciemnianie obrazu rastrowego

Podnoszenie do potęgi elementów macierzy `xnorm` powoduje przyciemnienie obrazu, jak pokazano na rysunku 9.4.3.2-1. Potęgowanie nie zmienia wartości 0 ani wartości 1 w macierzy obrazu.

Rysunek 9.4.3.2-1. *Obraz oryginalny xnorm (lewy) oraz po potęgowaniu: xnorm.^1.5 i xnorm.^2*

Przykład 9.4.3.2-1

Przetworzenie obrazu — przyciemnienie. Kontynuacja przykładu 9.4.1-1:

```
figure,imagesc(xnorm), colormap gray,
xsq=xnorm.^1.5; figure,imagesc(xsq), colormap gray
xsq=xnorm.^2; figure,imagesc(xsq), colormap gray,
```

9.4.4. Operacje logiczne — binaryzacja

Obraz binarny ma tylko dwa kolory: biały i czarny. Po binaryzacji macierz obrazu zawiera tylko zera i jedynki. Binaryzacja może poprawić czytelność wybranego elementu obrazu, np. napisu.

Przykład 9.4.4-1

Z pokazanej na rysunku 9.4.4-1 fotografii wybrano niewielki fragment z tekstem wydrukowanym na obwodzie scalonym i poddano go binaryzacji.

```
clear, close all
% pliki dostępne na serwerze www.mathworks.com/matlabcentral/fileexchange
% w oknie search należy wpisać: mrozek

image(imread('printedBoard1.jpg')) % read and show picture /wczytaj i pokaż obraz
ccm=imread('ccm.jpg');       % integrated circut photo / fotografia płytki drukowanej
ccm=single(ccm);
ccmGray=.2989*ccm(:,:,1)+.5870*ccm(:,:,2)+.1140*ccm(:,:,3);% gray/szary, NTSC
ccmMax=max(max(ccmGray));  % 0..ccmMax
ccm1=ccmGray./ccmMax;        % 0..1
ccm05=(ccm1>0.5);            % binarization level 0.5 / poziom odcięcia 0.5
ccm1OVERccm05=[ccm1;ccm05];    % dwa obrazy: oryginalny szary, a pod nim binarny
imwrite(ccm05,'ccm05.png')
% new window/ nowe okno
figure('NumberTitle','off','Name','gray & binary /szary lub binarny')
imagesc(ccm1OVERccm05), colormap gray    % show picture 2 /wyświetl obraz 2
```

Na rysunku 9.4.4-1 (po lewej) stronie rysunku pokazano fotografię płytki drukowanej z obwodem scalonym. Niewielki fragment oryginalnego obrazu poddano obróbce w MATLAB-ie. Obrazek górny (po prawej) jest małym fragmentem fotografii w odcieniach szarości — zmienna `ccmGray` w programie. Obrazek dolny `ccm05` jest dodatkowo przetworzony przez binaryzację na poziomie 0.5. Jest on czarno-biały i bardziej czytelny.

Rysunek 9.4.4-1. *Płytka drukowana (po lewej stronie) oraz jej powiększony fragment: przed binaryzacją (górny) i po binaryzacji na poziomie 0.5 (dolny)*

9.4.5. Operacje morfologiczne

Wybrane operacje morfologiczne (erozja i dylatacja) będą podane tylko przykładowo. Rezultaty tych operacji, wykonanych z użyciem *Image Processing Toolbox*, pokazano na rysunkach 9.4.5-1 i 9.4.5-2.

Rysunek 9.4.5-1. *W środku pokazano napis z rysunku 9.4.4-1 przetworzony przez binaryzację na poziomie 0.5. Po lewej stronie pokazano rezultat binaryzacji i erozji, a po prawej rezultat binaryzacji i dylatacji*

Rysunek 9.4.5-2. *Otwarcie, czyli erozja, a potem dylatacja (po lewej) oraz domknięcie, czyli dylatacja, a następnie erozja (po prawej stronie)*

9.4.6. Operacje arytmetyczne. Filtry

Filtrację można wykonać, korzystając z funkcji bibliotecznej `conv2`. Wybierając odpowiedni filtr, uzyskuje się wyostrzenie lub rozmycie obrazu, wykrycie krawędzi pionowych, poziomych lub ukośnych. Można też uzyskać wiele innych efektów opisywanych jako artystyczne.

Filtrowanie polega na obliczeniu nowej wartości piksela jako sumy ważonej jego aktualnej wartości i wartości pikseli z nim sąsiadujących. Współczynniki wagi są wpisane do niewielkiej macierzy zwanej **filtrem** lub **maską**. Wymiar filtru to zazwyczaj 3×3. Efekty działania filtrów o wymiarach większych, np. (5×5, 7×7…), można uzyskać poprzez kilkakrotne powtórzenie filtracji z małymi filtrami.

Aby dokonać filtracji, należy punkt centralny filtru umieszczać kolejno nad każdym pikselem oryginalnego obrazu. Nową wartość tego piksela oblicza się jako sumę iloczynów wartości każdego z elementów filtru i wartości umieszczonego pod nim piksela. Następnie przesuwa się filtr i powtarza obliczenia dla kolejnego piksela. Operacje te wykonuje funkcja `conv2`. Punkty na krawędzi obrazu wymagają specjalnego potraktowania.

Nawet proste filtry pozwalają na radykalną zmianę wyglądu przetwarzanego obrazu. Poniżej pokazano kilka przykładów.

9.4.6.1. Filtry dolnoprzepustowe

Filtry dolnoprzepustowe uśredniają wartości sąsiadujących pikseli. Zmniejsza to gwałtowność przejścia pomiędzy bielą i czernią, powodując nieznaczne rozmycie (zmiękczenie) obrazu i zmniejszenie jego ostrości (efekt *blur*). Poniżej podano przykłady typowych filtrów dolnoprzepustowych oraz efekty ich działania.

$$\begin{matrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{matrix} \qquad \begin{matrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{matrix} \qquad \begin{matrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{matrix}$$

```
X=imread ('lena.jpg');    % wczytanie pliku jako liczby int8
x=single(X(50:200,50:200));    % liczby single, wybrano twarz
x=x/max(max(x));
imwrite(x,'twarz.jpg','jpg')
figure, imagesc(x), colormap(gray),,title('oryginalny')
filtr=ones(3)/9;
xf=conv2(x,filtr,'same');
figure, imagesc(xf), colormap gray,title('filtr=ones(3)/9'),
imwrite(xf,'dolnop9.jpg','jpg')
filtr=ones(3)/10;
xf=conv2(x,filtr,'same'); imwrite(xf,'dolnop10.jpg','jpg')
filtr=ones(3)/10; filtr(2,2)=.2
xf=conv2(x,filtr,'same'); imwrite(xf,'dolnop102.jpg','jpg')
filtr=[1 2 1;2 4 2;1 2 1]/16
xf=conv2(x,filtr,'same'); imwrite(xf,'dolnop242.jpg','jpg')
```

W powyższym programie do zmiennej `filtr` należy wstawić jedną z podanych wyżej tablic, a efekty filtracji będą zbliżone do pokazanych na rysunku 9.4.6.1-1.

Rysunek 9.4.6.1-1. *Powiększona fotografia twarzy: oryginalna i filtrowana dolnoprzepustowo (obraz rozmyty, mniej ostry)*

9.4.6.2. Filtry górnoprzepustowe

Podano tu przykłady typowych filtrów górnoprzepustowych (ang. *high pass filter*) oraz efekty ich działania.

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix} \qquad \begin{matrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{matrix}$$

W poniższym programie do zmiennej `filtr` wstawia się jedną z podanych wyżej tablic, a efekty filtracji będą zbliżone do pokazanych na rysunku 9.4.6.2-1.

```
clear, close all
% plik dostępny na serwerze www.mathworks.com/matlabcentral/fileexchange
% w oknie search należy wpisać: mrozek
%
X=imread ('lena.jpg');    % wczytanie pliku jako liczby int8
x=single(X(50:200,50:200));    % liczby single, wybrano twarz
x=x/max(max(x));    % normalizacja [0,1]
filtr=-ones(3); filtr(2,2)=8
xf=conv2(x,filtr,'same');    % filtracja
xf8=uint8(256*xf);    % odrzuca wartości poza zakresem [0,255]
figure, imagesc(xf8), colormap gray,title('filtr=-ones(3);
filtr(2,2)=8'),
imwrite(xf8,'fgp8.jpg','jpg')
% return
filtr=-ones(3); filtr(2,2)=9
xf=conv2(x,filtr,'same');    % filtracja
xf8=uint8(256*xf);    % odrzuca wartości poza zakresem [0,255]
figure, imagesc(xf8), colormap gray,title('filtr=-ones(3);
filtr(2,2)=9'),
imwrite(xf,'fgp9.jpg','jpg')
```

9.4.6.3. Inne filtry

Pokazane powyżej przykłady filtrów dolno- i górnoprzepustowych prezentują tylko niewielką część możliwych do uzyskania efektów. Zachęca się Czytelnika do wypróbowania filtrów gradientowych, różniczkowych, a także filtrów Robertsa, Sobela, Robinsona, Laplace'a oraz tak zwanych filtrów artystycznych. Można je znaleźć w specjalistycznych publikacjach i w internecie.

Rysunek 9.4.6.2-1. *Powiększona fotografia twarzy: oryginalna oraz filtrowana górnoprzepustowo. Filtr z punktem centralnym 8 pokazuje białe krawędzie na czarnym tle. Filtr z punktem centralnym 9 uwypukla szczegóły bez zaczerniania obrazu*

9.5. Światło, odbicia i tekstury

MATLAB pozwala na tworzenie realistycznych obrazów brył trójwymiarowych z uwzględnieniem ich oświetlenia i różnorodnych parametrów odbijania światła od zakrzywionej powierzchni. Stosowanie tekstur pozwala nadać powierzchniom bryły charakterystyczny rysunek struktury materiału, na przykład marmuru lub drewna.

9.5.1. Źródła światła i odbicia

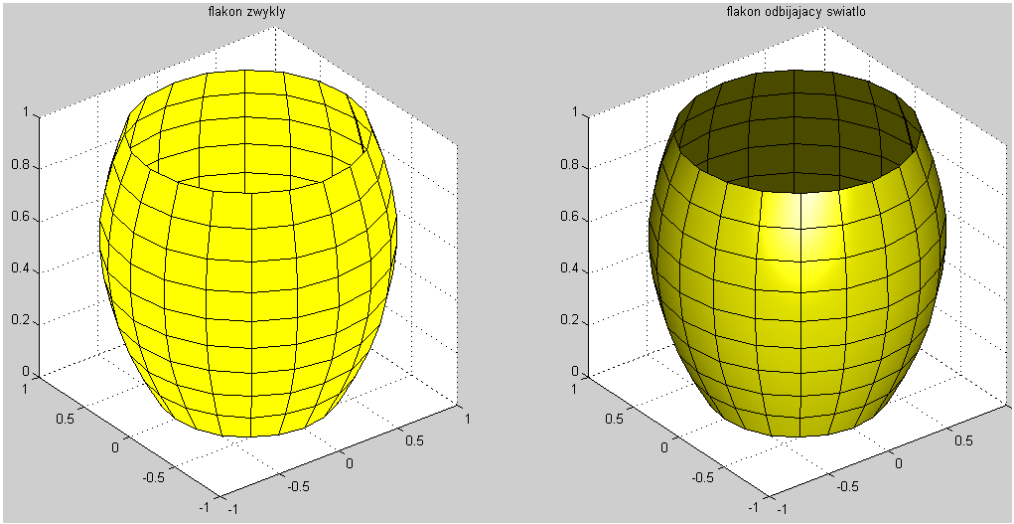
Istotę problemu oświetlenia bryły i odbijania światła przez jej powierzchnię ilustruje rysunek 9.5.1-1. Pokazano na nim dwie bryły. Siatka nałożona na bryły daje w obu wypadkach efekt przestrzenny. Dodatkowe efekty: oświetlenie, odbicie światła oraz zaciemnienie wnętrza — w sposób znaczący poprawiają wygląd prawej bryły. Wybrane funkcje, które realizują różne sposoby oświetlenia i zmieniają parametry odbicia światła, to:

- `surfl`, `surf` — powierzchnia trójwymiarowa z oświetleniem i bez,
- `lighting` — określa rozpraszanie światła na powierzchni,
- `material` — współczynnik odbicia materiału powierzchni,
- `specular` — odbicie lustrzane,
- `diffuse` — odbicie z rozproszeniem,
- `camlight` — źródło światła, współrzędne kamery.

Przykład 9.5.1-1

Oświetlenie bryły (`lighting`) i odblaski typu `phong`:

```
t=2/3:1/6:.7*pi;
y=sin(t);
[a,b,c]=cylinder(y);              % dane do narysowania bryły
subplot(1,2,1),  h1=surf(a,b,c),  title('flakon zwykly')
 set (h1,'facecolor','y'),        % nadaje kolor żółty (yellow)
subplot(1,2,2),  h2=surf(a,b,c)   % druga bryła (prawy rysunek)
 set (h2,'facelighting','phong','backfacelight','lit')    % odblaski
```



Rysunek 9.5.1-1. *Dzięki oświetleniu (lighting) i odblaskom typu phong (po prawej stronie) uzyskano realistyczny wygląd powierzchni bryły trójwymiarowej*

```
set (h2,'facecolor','y')          % nadaje kolor żółty (yellow)
light('position',[-2,-3,.6])      % tworzy obiekt: źródło światła
title('flakon odbijajacy swiatlo')
```

9.5.2. Tekstura — nakładanie obrazu na powierzchnię

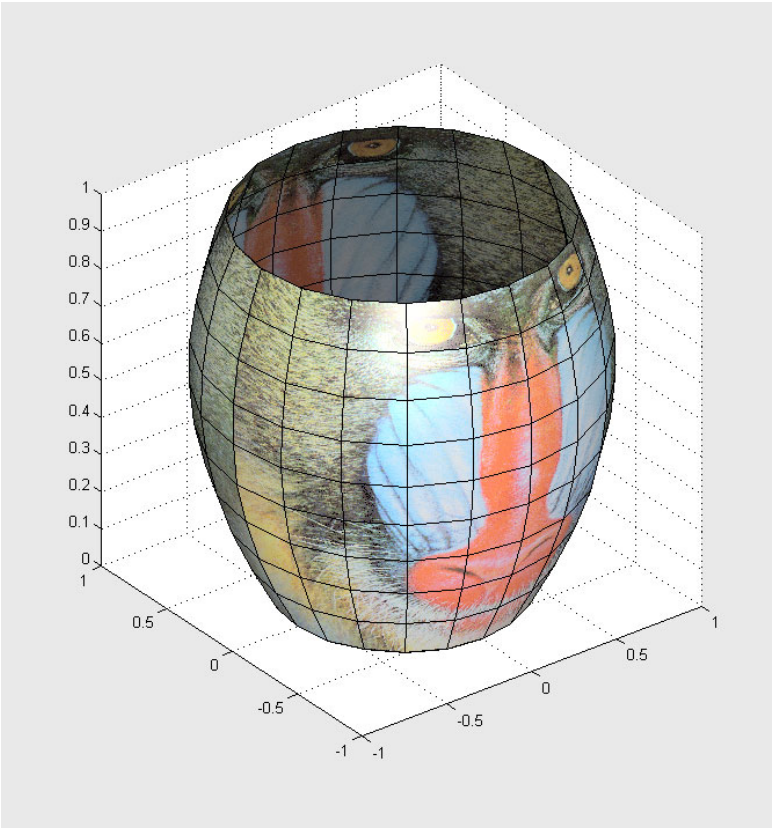
Teksturami nazywamy obrazy płaskie nałożone na zakrzywioną powierzchnię trójwymiarową. Bardzo efektowne jest nałożenie rysunku lub fotografii na powierzchnię kuli lub walca. Poniżej zamieszczono przykład nałożenia ilustracji na powierzchnię bryły. Wykorzystano plik z rysunkiem *mandrill.mat*, który można znaleźć w folderze *matlab\toolbox\matlab\demos*.

Przykład 9.5.2-1

Na powierzchnię bryły nałożono plik z rysunkiem *mandrill.mat*. Uzyskany rezultat pokazano na rysunku 9.5.2-1. Funkcji `findobj` użyto celem znalezienia identyfikatora powierzchni narysowanej wcześniej bryły (rysunek 9.5.1-1). Nałożenie tekstury na obiekt graficzny wymaga wpisania odpowiednich wartości w pola tego obiektu:

- `'texturemap'` do pola `'FaceColor'`,
- `x=[X,X]` do pola `'CData '`.

```
% wymaga wcześniejszego narysowania bryły lub powierzchni
load mandrill      % załadowanie ilustracji, tablice (X,map)
x=[X X];           % dwukrotne powtarzanie ilustracji
x=flipud(x);       % odbicie lustrzane ilustracji: góra/dół
h=findobj('type','surface');    % wyszukanie identyfikatora powierzchni
set(h,'CData',x,'FaceColor','texturemap')    % nałożenie x jako tekstury
colormap(map)                                % paleta barw
```



Rysunek 9.5.2-1. *Tekstura — na powierzchnię bryły nałożono obraz (mandrill.mat)*

```
set(h,'facelighting','phong','backfacelight','lit')    % odbicie światła
light('position',[-2,-3,.6])      % utworzenie obiektu: źródło światła
brighten(.05)                                  % rozjaśnienie rysunku
```

Rozdział 10.

Simulink — pakiet do modelowania i symulacji

Simulink jest środowiskiem graficznym do projektowania systemów. Posiada edytor graficzny, biblioteki bloków i solvery (algorytmy rozwiązujące równania różniczkowe) przeznaczone do modelowania i symulacji systemów dynamicznych. Jest zintegrowany z MATLAB-em, umożliwia wykorzystanie algorytmów MATLAB-a do symulacji oraz eksport wyników symulacji do MATLAB-a i innych programów do dalszej analizy.

Simulink wykonuje symulacje, automatyczne generowanie kodu, umożliwia jego testowanie oraz weryfikację, w tym pod kątem użycia dedykowanego kodu w systemach wbudowanych i w systemach krytycznych dla bezpieczeństwa.

Badanie systemów dynamicznych metodami symulacyjnymi oraz prototypowanie znacznie zmniejsza koszty i czas niezbędny do przygotowania lub modernizacji prototypów urządzeń i systemów. Do Simulinka można dokupić liczne rozszerzenia, niektóre z nich są omówione w tym rozdziale.

10.1. Pierwsza sesja z Simulinkiem

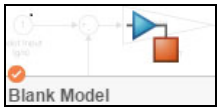
Przy użytkowaniu Simulinka istotną umiejętnością jest definiowanie modeli w postaci schematu blokowego. Pierwszym krokiem jest uruchomienie MATLAB-a (**Simulinka nie można uruchomić bez MATLAB-a**). Następnie uruchamia się Simulink poprzez:

- kliknięcie ikony *Simulink* w zakładce *Home* panelu MATLAB,
- lub wybranie z menu *New/SimulinkModel*,
- lub wpisanie polecenia `simulink` w oknie *Command Window*.

Rezultatem jest otwarcie okna *Simulink Start Page*.

10.1.1. Otwarcie okna Simulink Editor

Konstruowanie modelu należy rozpocząć od otwarcia nowego okna modelu z nagłówkiem *untitled*. Można je otworzyć kliknięciem ikony *Blank Model* (rysunek 10.1.1-1) w oknie *Simulink Start Page*.



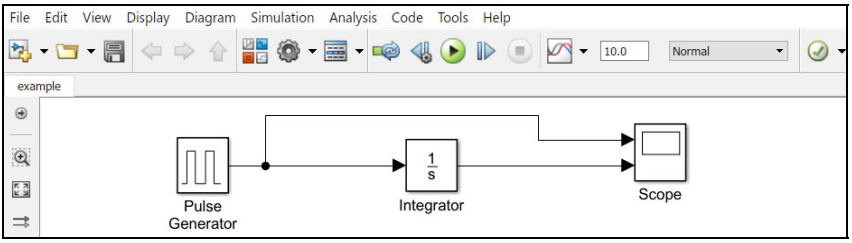
Rysunek 10.1.1-1. *Ikona Blank Model w oknie Simulink Start Page*

Otworzy się wtedy okno edytora graficznego *Simulink Editor*, nazwane *untitled*, w którym można umieścić bloki Simulinka i połączyć je celem utworzenia modelu. Klikając w menu *File* lub ikonę *Save*, można zapisać to okno pod nową nazwą w wybranym folderze (na przykład *example* w folderze *Documents\MATLAB*).

10.1.2. Pobieranie bloków z bibliotek do okna modelu

Na rysunku 10.1.2-1 pokazano model Simulinka. Do jego budowy użyto bloków:

- *Pulse Generator* — generator funkcji prostokątnej,
- *Integrator* — blok całkujący,
- *Scope* — oscyloskop, blok do wizualizacji wyników symulacji.



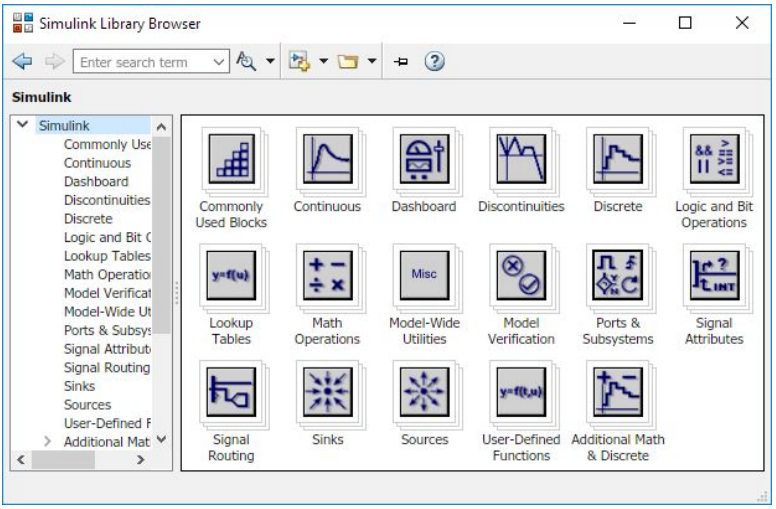
Rysunek 10.1.2-1. *Okno edytora graficznego Simulink Editor, model example*

Bloki można pobierać z bibliotek dostępnych w oknie przeglądarki *Simulink Library Browser* (rysunek 10.1.2-2) lub z bibliotek własnych użytkownika.

Przeglądarkę otwiera się, klikając ikonę w oknie *Simulink Editor* (rysunek 10.1.2-1). Bloki potrzebne do budowy modelu pobiera się myszą z odpowiednich bibliotek na kilka sposobów. Na przykład, aby pobrać generator przebiegów prostokątnych *Pulse Generator*:

- w lewej części okna należy wybrać bibliotekę źródeł sygnału *Sources*, po czym odszukać blok *Pulse Generator* i przeciągnąć go myszą do okna tworzonego modelu,
- można też wpisać nazwę `Pulse Generator` w okienku *Enter serach term* okna *Simulink Library Browser*; gdy potrzebny blok zostanie odszukany — trzeba:
  - przeciągnąć go myszą do okna tworzonego modelu,
  - lub kliknąć go prawym przyciskiem myszy i wybrać z menu kontekstowego *Add Block to…*,

Rysunek 10.1.2-2. *Okno przeglądarki Simulink Library Browser*



- można kliknąć w dowolnym pustym obszarze wewnątrz okna *Simulink Editor* i następnie wpisać z klawiatury początek nazwy potrzebnego bloku. Pojawi się lista bloków, z której wybiera się potrzebny blok.

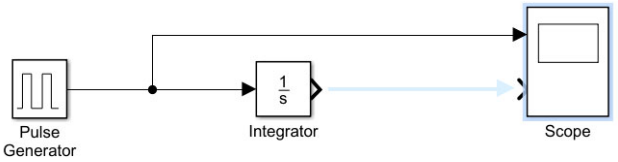
Podobnie pobiera się pozostałe bloki:

- *Integrator* pobiera się z biblioteki *Continuous* lub z *Comonly Used Blocks*.
- *Scope* pobiera się z biblioteki *Sinks* lub z *Comonly Used Blocks*.
- Po pobraniu pierwszego bloku *Scope* należy kliknąć go prawym przyciskiem myszy i wybrać z menu kontekstowego opcję *Copy*. Rezultatem jest pojawienie się dodatkowego bloku *Scope*.

10.1.3. Łączenie bloków liniami przesyłania sygnałów

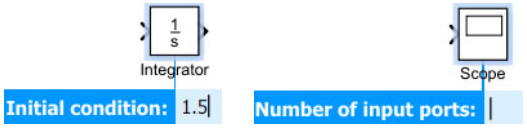
Po operacji kopiowania bloków bibliotecznych okno modelu zawiera trzy bloki, które można ustawić myszą jak na rysunku 10.1.3-1.

Rysunek 10.1.3-1. *Łączenie bloków liniami przesyłania sygnałów*



Jeśli w modelu są bloki całkujące *Integrator*, to należy wpisać ich warunki początkowe do okienka, które pojawi się w momencie wstawienia integratora do okna modelu (rysunek 10.1.3-2). Podobnie można zmienić liczbę wejść bloku *Scope*. Parametry bloków można też zmienić w oknie dialogowym bloku. Okno to otwiera się zazwyczaj dwukrotnym kliknięciem bloku.

Rysunek 10.1.3-2. *Zmiana parametrów bloków bibliotecznych*



Połączenia pomiędzy blokami można wykonać na kilka sposobów:

- kliknięcie portu wyjściowego (miejsce wyjścia sygnału) bloku poprzedzającego i bez zwalniania lewego przycisku myszy przeciągnięcie linii do wejścia bloku następnego; po dotknięciu tego bloku linia zmienia kolor i na jej końcu pojawia się strzałka — wtedy zwalnia się przycisk myszy;
- lub kliknięcie bloku poprzedzającego (oznacza to jego wybranie i zaznaczenie), następnie wciśnięcie i trzymanie klawisza *Ctrl* klawiatury oraz kliknięcie bloku następnego (lub jego nazwy) — spowoduje połączenie tych bloków linią sygnałową;
- lub przy precyzyjnym wyrównaniu dwu bloków w poziomie lub w pionie pojawi się pomiędzy nimi *cienka, niebieska linia* ze strzałką (na rysunku 10.1.3-1 wyrównano bloki *Integrator* i *Scope*); kliknięcie tej linii spowoduje wrysowanie linii połączenia tych bloków;
- rozdzielenie sygnału w celu jego doprowadzenia do kilku bloków wymaga kliknięcia prawym przyciskiem myszy w miejscu, gdzie linia sygnałowa ma być rozdzielona; następnie, nie zwalniając prawego przycisku myszy, przeciąga się dodatkową linię do kolejnego bloku, na przykład do *Scope*.

Po podwójnym kliknięciu linii sygnałowej pojawia się migający kursor, co umożliwia jej opisanie. Napis zatwierdza się kliknięciem myszy poza tym napisem.

Po zapamiętaniu i zamknięciu okna modelu można go ponownie otworzyć:

- z okna poleceń MATLAB-a, pisząc nazwę modelu jako polecenie np. `>> example`,
- z dowolnego otwartego okna modelu Simulinka należy wybrać opcję *File/Open* lub kliknąć ikonę *Open model* paska narzędziowego.

Modele Simulinka są zgodne z *Packaging Conventions (OPC)* i *UnicodeUTF-8*. Mają one rozszerzenie *.slx* lub *.slxp* (ang. *protected* — chronione). W starszych wersjach Simulinka używano rozszerzeń *.mdl* i *.mdlp*.

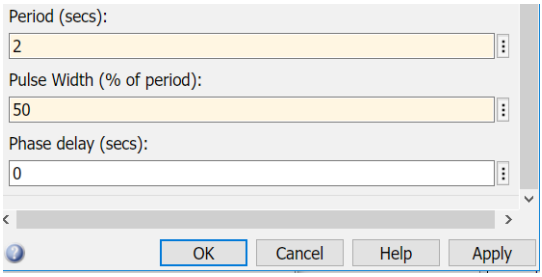
10.1.4. Uruchomienie symulacji

Symulację modelu pokazanego na rysunku 10.1.2-1 uruchamia się, klikając ikonę *Run* lub z menu *Simulation/Run* w oknie zawierającym uruchamiany model. Domyślny czas symulacji wynosi 10. Można go zmienić w okienku na pasku narzędziowym tego okna lub w menu *Simulation/Model Configuration Parameters.*

Bloki Simulinka mają wstępne (domyślne) wartości parametrów; można je odczytać i zmienić po dwukrotnym kliknięciu bloku. Wartości domyślne parametrów bloku *Pulse Generator* (*Period* = 10, *Pulse Width* = 10%) zmieniono na wartości pokazane na rysunku 10.1.4-1.

Menu *Simulation/Model Configuration Parameters* w oknie *Simulink Editor* (rysunek 10.1.2-1) umożliwia zmianę parametrów symulacji.

- Opcja *Solver* pozwala na określenie:
  - czasu rozpoczęcia i zakończenia symulacji, domyślnie $t = [0, 10]$,



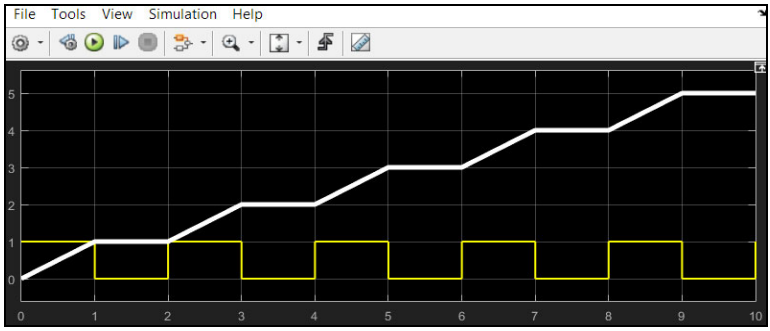
Rysunek 10.1.4-1. *Zmienione parametry bloku Pulse Generator*

- wybór pomiędzy symulacją stało- i zmiennokrokową,
- wybór solvera zmiennokrokowego: *wybór automatyczny*, *ode45*, *ode23*, *ode113*, *ode15s*, *ode23t*, *ode23s*, *ode23tb*, (podrozdział 8.2) oraz solver dyskretny,
- wybór solvera stałokrokowego: *wybór automatyczny*, *ode8*, *ode5*, *ode4*, *ode3*, *ode2*, *ode1*, *ode14x*.

♦ Pozostałe opcje w *Model Configuration Parameters* to: *Data Import/Export*, *Optimization*, *Diagnostics*, *Hardware Implementation*, *Model Referencing*, *Simulation Target*. Więcej opcji jest dostępne po wybraniu *All Parameters*.

10.1.5. Wizualizacja wyników symulacji

Otwarcie okna z wizualizacją wymaga dwukrotnego kliknięcia bloku *Scope* (rysunek 10.1.5-1).



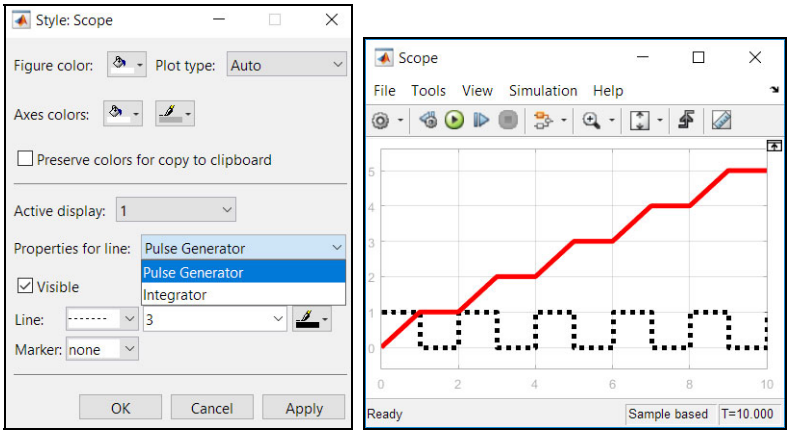
Rysunek 10.1.5-1. *Wyniki symulacji w oknie graficznym bloku Scope*

Domyślny kolor tła wykresu jest czarny. Zmianę kolorów tła oraz grubości, typu i koloru linii umożliwia menu *View/Style* okna *Scope* (rysunek 10.1.5-2).

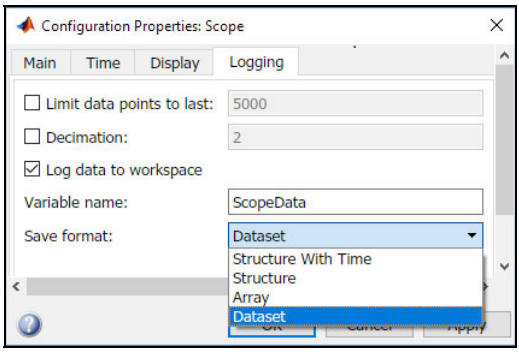
Opcja *View/Configuration Properties/Logging* okna *Scope* umożliwia zapisanie wyników symulacji do macierzy lub struktury w przestrzeni roboczej MATLAB-a (rysunek 10.1.5-3). Umożliwia to zapisanie wyników do pliku (zalecany jest format binarny *.mat* — z uwagi na kompresję danych i zachowanie pełnej ich dokładności) oraz przetworzenie i opracowanie wyników w trybie offline w MATLAB-ie lub w innym oprogramowaniu.

W przypadku wybrania opcji *Dataset* wyniki symulacji są dostępne jako:

```
czas = ScopeData{1}.Values.Time
dane = ScopeData{1}.Values.Data;
```



Rysunek 10.1.5-2. *Zmiana koloru i grubości linii oraz tła wykresów bloku Scope*



Rysunek 10.1.5-3. *Zapisywanie wyników symulacji do macierzy lub struktury*

10.1.6. Tryby przyspieszone symulacji — accelerated i rapid

Symulację można przyspieszyć, zastępując interpretację kodu zawartego w modelach Simulinka (tryb normalny) przez wykorzystanie kodu skompilowanego, dostępnego dla większości bloków Simulinka. Tryby przyspieszone nie wymagają posiadania kompilatora ani koderów, ale ich posiadanie pozwala na przygotowanie kodu skompilowanego dla bloków, dla których taki kod nie jest dostępny.

Tryby przyspieszone uruchamia się przez wybór trybu *Normal*, *Accelerator* lub *Rapid Accelerator* w menu *Simulation/Mode*.

W trybie *Normal* oraz w trybie *Accelerator* w modelu mogą być użyte zarówno bloki, dla których jest dostępny kod skompilowany, jak i bez takiego kodu. W trybie tym można korzystać z *debugera* i *profilera*, co pozwala na wykrywanie błędów oraz na wskazanie, które części modelu i w jakim stopniu spowalniają symulację.

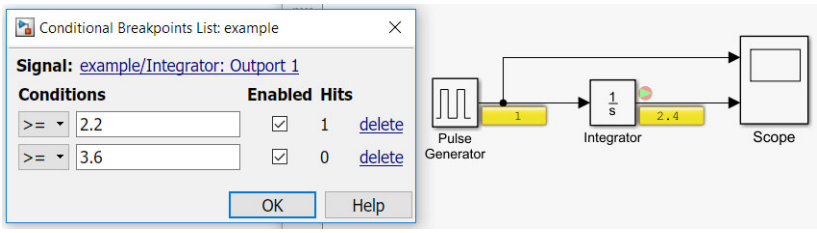
Tryb *Rapid Accelerator* współpracuje wyłącznie z modelami, dla których jest dostępny kod skompilowany. Nie można w tym trybie korzystać z debugera ani z profilera. Bloki wyjściowe, np. *Scope*, są uruchamiane dopiero po zakończeniu przebiegu symulacji. Jeśli komputer ma procesor dwurdzeniowy, to na jednym rdzeniu działa środowisko

MATLAB i Simulink, a drugi jest wykorzystywany do symulacji. Niektóre z algorytmów zaimplementowanych w blokach Simulinka mogą używać więcej rdzeni bez potrzeby posiadania *Parallel Computing Toolbox*.

10.1.7. Stepper

W Simulinku są dostępne *stepper* i *debuger*. Oba narzędzia analizują przebieg symulacji w celu odnalezienia i identyfikacji zawartych w nich błędów, a ich możliwości częściowo się pokrywają. *Debuger* opisano w poprzednim wydaniu książki. Uruchamia się go przez wybór opcji *Simulation/Debug*. Jest on znacznie bardziej zaawansowanym narzędziem niż *stepper*.

Stepper pozwala na zatrzymanie symulacji po upływie wskazanego czasu lub po spełnieniu przez wskazany sygnał warunku wyrażonego przez wartość liczbową i wybraną relację, jak pokazano na rysunku 10.1.7-1.



Rysunek 10.1.7-1. *Stepper zatrzymał symulację po spełnieniu warunku — sygnał >= 2.2. Żółte chorągiewki z wartością sygnału pojawiają się po dwukrotnym kliknięciu linii sygnałowej*

Pracą *steppera* sterują ikony  (rysunek 10.1.2-1). Pierwsza z lewej umożliwia szybki restart po zatrzymaniu, druga określa możliwość wykonania kroków wstecz, strzałka w kółku uruchamia symulację, a ostatnia ikona wykonuje krok do przodu.

Przed rozpoczęciem symulacji można zaprogramować *stepper*. Po kliknięciu ikony  zaznacza się *Enable stepping back*. Pozostałe wartości można pozostawić bez zmiany. Aby ustawić pułapkę *zatrzymaj, jeśli* sygnał integratora >= 2.2 lub >= 3.6, należy prawym przyciskiem myszy kliknąć wybraną linię sygnału i z menu kontekstowego wybrać *Add Conditional Breakpoint*. Otworzy się okienko, które należy wypełnić zgodnie z rysunkiem 10.1.7-1. Na wskazanej linii pojawi się czerwona kulka potwierdzająca ustawienie pułapki.

Po uruchomieniu symulacji zostanie ona zatrzymana po spełnieniu ustawionego warunku. Następnie zależnie od potrzeby można wykonać jeden krok lub więcej kroków w przód lub wstecz. Można też kontynuować symulację lub ją zakończyć.

10.2. Druga sesja z Simulinkiem

Simulink daje możliwość łatwego zapisu równań różniczkowych w postaci graficznej:

Przykład 10.2-1

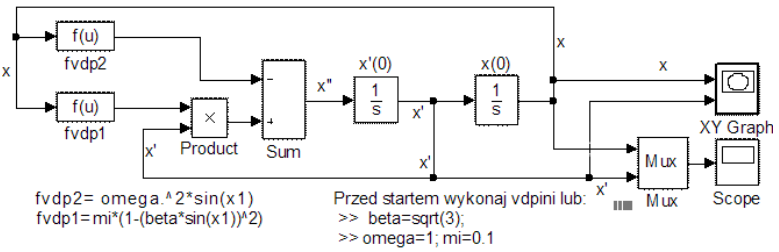
Równanie różniczkowe II rzędu w postaci:

$$\ddot{x} + \omega^2 \sin x = \mu(1 - b^2 \sin^2 x)\dot{x} \quad \text{dla } \mu \ll 1 \tag{10.2-1}$$

można zapisać jako układ równań I rzędu:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \mu(1 - b^2 \sin^2 x_1)x_2 - \omega^2 \sin x_1 \end{aligned} \tag{10.2-2}$$

Powyższy układ równań zapisano w postaci modelu Simulinka o nazwie *vdp2n*, który pokazano na rysunku 10.2-1.



Rysunek 10.2-1. *Model równania różniczkowego drugiego rzędu vdp2n*

Do budowy tego modelu użyto następujących bloków bibliotecznych:

- Blok sumatora *Sum* — w oknie dialogowym bloku *Sum* wpisano znaki sumowanych sygnałów. Ponadto zmieniono kształt ikony tego bloku, wybierając z menu *Icon shape* opcję *rectangular*.
- W bloku mnożenia *Product* pozostawiono domyślne mnożenie wartości wejściowych (element razy element .*) i liczbę wejść 2.
- Bloki *Fcn* (biblioteka *User-Defined Functions*) realizują wyrażenia wpisane w okna dialogowe tych bloków:
  - *fvdp2* — `omega*omega*sin(u[1])` — `u[1]` to sygnał na wejściu;
  - *fvdp1* — `mi*(1 - beta*sin(u[1]) * beta*sin(u[1]))`.
- Bloki *Integrator* całkują wartości pochodnych i do każdego z nich należy odpowiednio wpisać wartości warunków początkowych: $x' = 2.5$ oraz $x = 0$.
- *Mux* (biblioteka *Signal Routing*) przekształca dwa sygnały `x` i `x'` w jeden sygnał wektorowy `[x, x']`.
- *Scope* i inne często używane bloki można pobrać z *Comonly Used Blocks*.

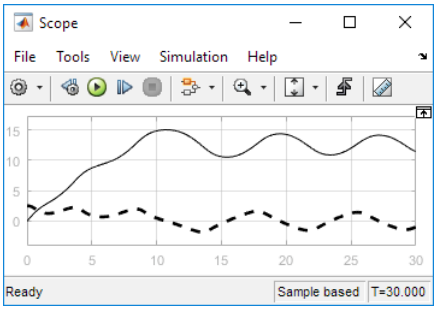
Dla bloków *Sum*, *Product*, *Scope*, *Mux* zachowano nazwy biblioteczne. Natomiast blokom *Fcn* i *Integrator* zmieniono nazwy. Wprowadzono etykiety sygnałów `x`, `x'` oraz `x"`. Przebiegi czasowe pochodnej *x'* i rozwiązania *x* można obserwować w oknie graficznym bloku *Scope* (rysunek 10.2-2), a wykres fazowy w oknie graficznym bloku *XY graph* (rysunek 10.2-3).

Aby uzyskać wyniki symulacji, niezbędne jest wprowadzenie do przestrzeni roboczej wartości parametrów modelu (równania): `beta`, `omega`, `mi`. Można je wpisać z klawiatury lub zapisać w pliku np. o nazwie *vdpini.m*.

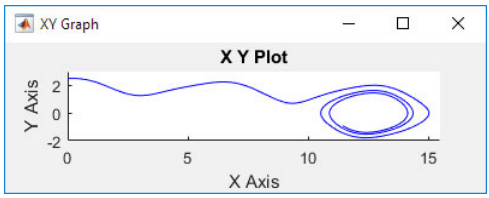
```
>>ts=ScopeData(:,1); xs=ScopeData(:,2); plot(ts,xs)
```

W podrozdziale 8.3.5 opisano model Simulinka dla równania różniczkowego III rzędu.

Rysunek 10.2-2. *Wykresy rozwiązania równania różniczkowego w dziedzinie czasu*



Rysunek 10.2-3. *Rozwiązanie równania różniczkowego w postaci diagramu fazowego w oknie XY_Plot*



10.3. Podsystemy — blok Subsystem

Jeśli rozpatrywany model zawiera wiele bloków składowych, to staje się mało przejrzysty i może nie mieścić się na ekranie. W takich przypadkach celowe jest tworzenie modeli hierarchicznych, to jest modeli, w których występują podsystemy.

Podsystem jest reprezentowany przez blok *Subsystem* i zastępuje w modelu wybraną grupę bloków. Bloki podsystemów mogą być maskowane: użytkownik może zaprojektować ikonę i okienko dialogowe dla swojego podsystemu.

W Simulinku jest dostępna biblioteka *Ports & Subsystems*. Zawiera ona bloki podsystemów działające warunkowo, bloki z zaimplementowanymi instrukcjami `if-else`, `for`, `while`, `switch-case` oraz bloki podsystemów uruchamiane tymi instrukcjami.

Bloki realizujące warunkowe działanie podsystemów ułatwiają współpracę Simulinka z pakietem *Stateflow*. Sygnały wejściowe dla modeli tworzonych w pakiecie *Stateflow* są pobierane z modeli Simulinka. Pakiet określa działania do wykonania, zmienia odpowiednie stany logiczne i wysyła sygnały wyzwalające lub (i) przełączające podsystemy w Simulinku.

10.3.1. Przykład modelu definiowanego graficznie

Do dalszych rozważań przyjęto jako przykład model układu *masa-sprężyna-tłumik*. Przyjęto, że charakterystyka sprężyny jest liniowa, a tarcie jest proporcjonalne do prędkości. Przesunięcie x mierzy się od położenia odpowiadającego zerowej sile sprężyny. Do masy przyłożono siłę zewnętrzną $F(t)$.

Rozpatrywany układ jest opisany równaniem różniczkowym II rzędu:

$$\ddot{x} + \beta\dot{x} + \omega x = F(t)$$
$$\dot{x}(0) = x_{01}, \quad x(0) = x_{02} \tag{10.3.1-1}$$

gdzie:

x — przesunięcie masy drgającej,

$\dot{x}$ — prędkość masy drgającej,

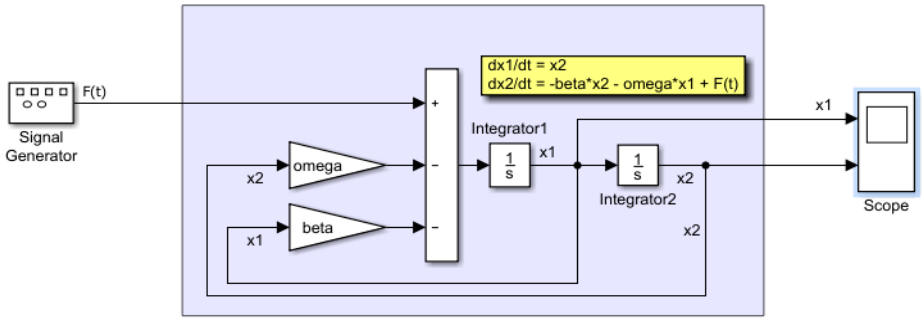
k — współczynnik sztywności sprężyny,

b — współczynnik tłumienia,

m — masa, $\beta = b/m$, $\omega = k/m$,

$\dot{x}(0)$, $x(0)$ — warunki początkowe.

Na rysunku 10.3.1-1 pokazano model Simulinka, który reprezentuje równanie różniczkowe drugiego rzędu zapisane powyżej i spełnia przyjęte założenia.



Rysunek 10.3.1-1. *Model układu masa-sprężyna-tłumik z zaznaczonym obrysem podsystemu*

W blokach *Integrator1* i *Integrator2* wpisano zmienne `x01` i `x02`, które odpowiednio określają warunki początkowe. Bloki *omega* i *beta* są to bloki *Gain*, którym zmieniono nazwę. Blok *Signal Generator* z biblioteki *Source* realizuje oddziaływanie na układ siły zewnętrznej $F(t)$, stałej lub zmiennej w czasie. Do wizualizacji wyników użyto bloku *Scope*.

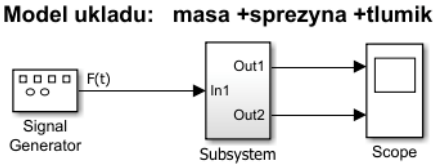
Sprawdzenie poprawności tak zdefiniowanego modelu wykonano dla następujących parametrów: `beta=5`, `omega=4`, `x01=50`, `x02=0`. Parametry te wpisano do pliku o nazwie `ini_mst.m`. Wykonanie tego pliku musi nastąpić przed rozpoczęciem symulacji.

10.3.2. Tworzenie podsystemów

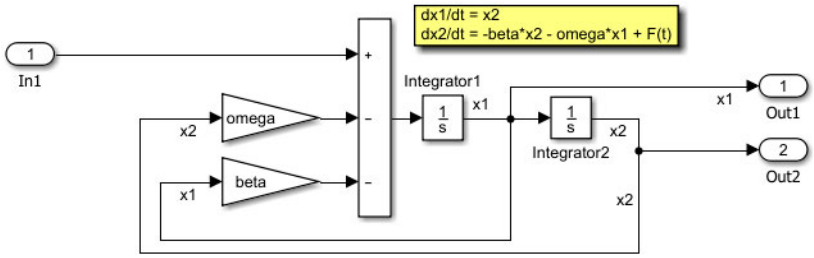
Podsystemy można tworzyć na dwa sposoby. Pierwszy sposób wymaga, aby elementy wybrane do utworzenia podsystemu były rozmieszczone wewnątrz wybranego prostokątnego obszaru, jak na rysunku 10.3.1-1. Następnie należy:

- wskazać myszą dowolny narożnik tego obszaru,
- przy wciśniętym lewym przycisku myszy przesunąć mysz do przeciwległego narożnika i zwolnić przycisk; wybrany obszar będzie nadal zaznaczony,
- wybrać menu: *Diagram/Subsystem & Model Reference/Create Subsystem from Selection*; zamiast menu można użyć skrótu klawiszowego: *Ctrl+G*,
- w miejsce zaznaczonego obszaru pojawi się blok *Subsystem* (rysunek 10.3.2-1).

Rysunek 10.3.2-1. *Model układu masa-sprężyna-tłumik jako blok Subsystem*



Dwukrotne kliknięcie bloku *Subsystem* otwiera okno podsystemu. Połączenia przechodzące przez granicę ramki stanowiącej obrys podsystemu zostają zachowane. W oknie podsystemu, w miejscach takich połączeń, pojawiają się automatycznie bramy wejściowe lub wyjściowe (rysunek 10.3.2-2).



Rysunek 10.3.2-2. *Wnętrze bloku Subsystem układu: masa+sprężyna+tłumik*

Inny sposób tworzenia podsystemu wymaga pobrania bloku *Subsystem* z biblioteki *Ports & Subsystems*. Okno bloku *Subsystem* otwiera się poprzez dwukrotne kliknięcie jego ikony. Następnie przekopiowuje się do niego potrzebne bloki tworzonego podsystemu i łączy się je ze sobą oraz z bramami wejścia *In* i wyjścia *Out*, zgodnie z rysunkiem 10.3.2-2.

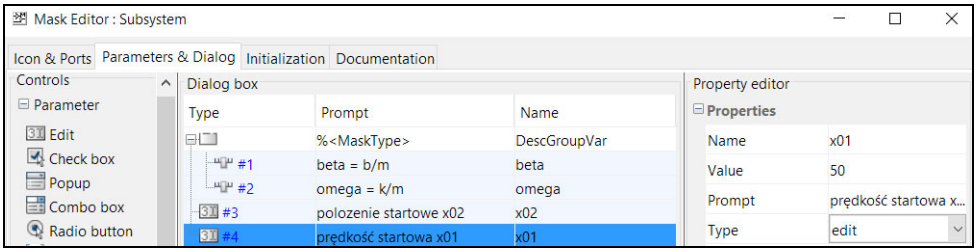
10.3.3. Maskowanie podsystemów

Mask Editor umożliwia zaprojektowanie okna dialogowego oraz ikony, odpowiednich dla tworzonego podsystemu. Maskowanie ma następujące zalety:

- łatwa zmiana parametrów podsystemu w oknie dialogowym,
- nadanie wartości początkowych parametrom i wykonanie poleceń inicjujących działanie podsystemu,
- możliwość przygotowania ikony bloku z własnym rysunkiem i tekstem.

Aby utworzyć maskę dla podsystemu, należy kliknąć wybrany podsystem i wpisać skrót klawiszowy *Ctrl+M*. Zamiast skrótu można kliknąć blok prawym przyciskiem myszy i wybrać z menu kontekstowego *Mask/Create*. Pojawi się okno *Mask Editor: Subsystem* z zakładkami:

- *Parameters & Dialog* — należy wypełnić zgodnie z rysunkiem 10.3.3-1. Wartości parametrów `beta` i `omega` będą ustawiane *suwakiem*, warunki początkowe `x02` i `x01` są wpisywane do okienka *Edit*. Ikony suwaka i okienka *Edit*, widoczne w kolumnie *Type*, przeciągnięto myszą z kolumny *Controls*.
- *Icon & Ports* — aby na ikonie podsystemu umieścić rysunek pokazujący masę drgającą i sprężynę, należy wpisać polecenia MATLAB-a jak poniżej w polu *Icon drawing commands*:



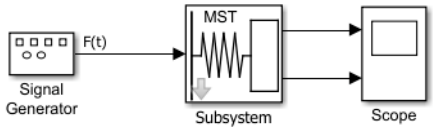
Rysunek 10.3.3-1. *Mask Editor: zakładka Parameters & Dialog*

```
plot([.12 .125 .125 .12 .12],[.4 .4 .6 .6 .4])          % wall
plot([.57 .78  .78   .57  .57],[.42 .42 .58 .58 .42])   % mass
plot([.125 .19 .21 .25 .29 .33 .37 .41 .45 .49 .51 .57],  % x_spring
     [.50  .50 .55 .45 .55 .45 .55 .45 .55 .45 .50 .50])  % y_spring
text(0.25,0.58,'MST')                                   % MST: masa, sprężyna, tarcie
```

- Zakładka *Initialization* nie będzie użyta; wartości początkowe parametrów wpisano w zakładce *Parameters & Dialog*.
- Zakładka *Documentation* — możliwość umieszczenia opisu podsystemu i podpowiedzi *help* pozostawia się Czytelnikowi.

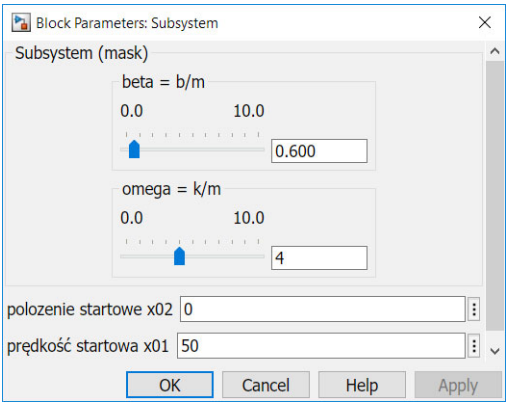
Zapamiętanie utworzonej maski i zamknięcie okna *Mask Editora* należy zatwierdzić, klikając przycisk *OK*, w dole tego okna. Na rysunku 10.3.3-2 pokazano model z *zamaskowanym podsystemem*.

Rysunek 10.3.3-2. *Model z zamaskowanym podsystemem*



Na rysunku 10.3.3-3 pokazano okno dialogowe *zamaskowanego podsystemu*. Korzystając z suwaków lub odpowiednich rubryk, można określić wartości parametrów `beta`, `omega`, `x02` oraz `x01`.

Rysunek 10.3.3-3. *Okno dialogowe zamaskowanego podsystemu*



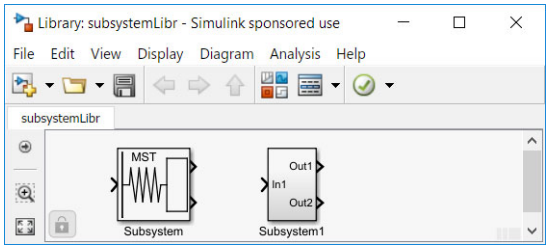
10.4. Biblioteki bloków

Własne bloki i modele Simulinka, a w szczególności bloki podsystemów *Subsystem* mogą być elementami składowymi innych, bardziej skomplikowanych modeli.

Łatwy dostęp do własnych bloków i modeli można uzyskać poprzez umieszczenie ich we własnej bibliotece.

10.4.1. Tworzenie własnych bibliotek bloków

Aby utworzyć nową bibliotekę, należy wybrać opcję *File/New/Library* w oknie dowolnego modelu i wybrać ikonę *Blank Library*. Do nowo otwartego okna biblioteki użytkownik może przenieść swoje bloki (rysunek 10.4.1-1).



Rysunek 10.4.1-1. *Utworzono bibliotekę i umieszczono w niej dwa podsystemy*

Zapisanie nowej biblioteki opcją *File/Save* powoduje jej zablokowanie. Modyfikacje biblioteki (w tym umieszczanie w niej nowych bloków) mogą być wykonywane dopiero po jej odblokowaniu. W oknie komunikatu informującego o blokadzie biblioteki znajduj się przycisk służący do jej odblokowania.

10.4.2. Dodatkowe bloki i biblioteki z toolboksów

Prócz bibliotek, których ikony pokazano na rysunku 10.1.2-2 (w oknie przeglądarki *Simulink Library Browser*), wraz z Simulinkiem instalowane są biblioteki *Simulink Extras* (rysunek 10.4.2-1).

Rysunek 10.4.2-1. *Ikony bibliotek Simulink Extras*

Po zainstalowaniu toolboksów (rozszerzeń MATLAB-a) w oknie przeglądarki *Simulink Library Browser* mogą się pojawić dodatkowe bloki lub biblioteki. Pozwalają one na korzystanie w modelach Simulinka z narzędzi i algorytmów pochodzących np. z *System Identification Toolbox*, *Fuzzy Logic Toolbox*, *Neural Network Toolbox* lub z *Control System Toolbox*.

10.5. Modelowanie fizyczne

Dostępne w środowisku MATLAB (Simulink) narzędzia modelowania fizycznego umożliwiają szybką budowę modeli systemów fizycznych. Ponadto modele te można łatwo połączyć z algorytmami sterowania i przetwarzania sygnałów. Wszystko to działa w jednym środowisku. Konstruowanie modeli instalacji przemysłowych może być wykonywane łącznie z ich systemami sterowania i z wykorzystaniem algorytmów dostępnych w Simulinku i w MATLAB-ie.

Modelowanie systemów fizycznych wymaga rozległej wiedzy fizycznej, matematycznej i znajomości języków programowania. Natomiast budowanie takich modeli w środowisku graficznym z uwzględnieniem połączeń fizycznych jest bardzo intuicyjne, zatem jest łatwiejsze do zrozumienia. Narzędzia modelowania fizycznego działające w środowisku MATLAB (Simulink) umożliwiają budowę modeli tak, aby odwzorowywały one strukturę inżynierską modelowanego układu. Nie jest wymagane określanie równań takiego układu ani pisanie programu.

Od wielu lat inżynierowie i naukowcy opracowują dokładne równania dla wielu elementów składowych układów fizycznych, takich jak silniki, zawory hydrauliczne albo sprzęgła. Istotna jest też identyfikacja parametrów krytycznych i łatwość doboru ich wartości. Dobór (realistycznych) poprawnych wartości parametrów jest elementem decydującym o dokładności budowanych modeli.

W wielu praktycznych zastosowaniach tworzy się algorytmy sterowania w oparciu o modele, które nie odwzorowują całej złożoności sterowanego obiektu, gdy działa on w warunkach rzeczywistych. Istnieje potrzeba testowania takich algorytmów w sposób powtarzalny, z możliwością kontroli uzyskiwanych rezultatów i dla warunków pracy identycznych z rzeczywistymi. Takie podejście pozwala zrozumieć i udoskonalić algorytm sterowania bez narażania personelu i urządzeń na uszkodzenia.

Modele fizyczne tworzone w środowisku Simulinka mogą być konwertowane do kodu w C celem wykonania testów w pętli, czyli testów HIL (ang. *hardware-in-the-loop*).

Dla Simulinka w wersji 8.9 (R2017a) są dostępne następujące narzędzia do modelowania fizycznego:

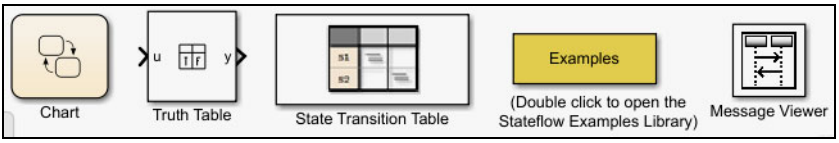
- *Simscape*™ — modelowanie i symulacja wielodziedzinowych systemów fizycznych, czyli takich, na które składają się modele z wymienionych poniżej narzędzi.
- *SimscapeDriveline*™ — modelowanie i symulacja mechanicznych układów napędowych, w tym przekładnie, wały obrotowe, sprzęgła, standardowe układy przenoszenia napędu oraz modele silników spalinowych i opon.
- *Simscape Electronics*™ — modelowanie i symulacja układów elektronicznych i elektromechanicznych, w tym półprzewodników, obwodów scalonych, układów logicznych, silników elektrycznych, napędów i serwomotorów małej mocy.
- *Simscape Fluids*® — modelowanie i symulacja hydraulicznych układów sterowania oraz pomp, zaworów i silników hydraulicznych.

- *Simscape Multibody* — modelowanie i symulacja układów mechanicznych, w tym narzędzia do budowy modelu składającego się z ciał sztywnych, połączeń, więzów i sił wewnętrznych, które odwzorowują strukturę układu mechanicznego.
- *Simscape Power Systems™* — jest rozszerzeniem Simulinka, umożliwia modelowanie i symulację procesów wytwarzania, przesyłu, dystrybucji i konsumpcji energii elektrycznej.

10.6. Stateflow — modelowanie i symulacja systemów sterowanych zdarzeniami

Stateflow to zintegrowane z Simulinkiem środowisko do modelowania i symulacji procesów sterowanych zdarzeniami, zwanych też systemami reaktywnymi. *Stateflow* wykorzystuje reprezentację graficzną maszyn skończenie wymiarowych w postaci zbliżonej do diagramów stanu (ang. *state charts*).

Pakiet *Stateflow* otwiera się za pomocą polecenia `stateflow`. Następuje wtedy otwarcie dwu okien: biblioteki *sflib* (rysunek 10.6-1) i modelu Simulinka z blokiem *Chart*.



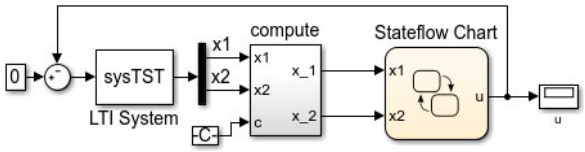
Rysunek 10.6-1. *Okno biblioteki sflib*

Do *Stateflow* dołączono liczne przykłady, w tym modele sprzęgła `sf_clutch.mdl`, windy `sf_elevator.mdl`, gry tetris `sf_tetris2.mdl`, sterownika do systemu klimatyzacji w mieszkaniu `sf_climate_control.mdl`, systemu alarmowego dla budynku mieszkalnego `sf_security.mdl`, układu napędowego (ang. *drive train*) samochodu z automatyczną skrzynią biegów `sldemo_autotrans.mdl` i wiele innych.

10.6.1. Simulink i blok Stateflow Chart

Diagramy stanu są umieszczane w modelach Simulinka jako bloki *Chart*. Struktura wewnętrzna bloku *Chart* jest niewidoczna. Model Simulinka pokazany na rysunku 10.6.1-1 realizuje układ sterowania ślizgowego (ang. *sliding mode*) ruchem modelu platformy mobilnej z robotem spawającym [Mrozek, Tarasiewicz, 2001]. Zmienne deklarowane jako wejście i wyjście do bloków Simulinka są widoczne jako bramy wejściowe i wyjściowe bloku *Stateflow Chart*.

Rysunek 10.6.1-1. *Okno modelu Simulinka z blokiem Stateflow Chart*



Podwójne kliknięcie ikony bloku *Stateflow Chart* w Simulinku otwiera okno edytora graficznego. W razie potrzeby można pobrać myszą pusty *Stateflow Chart* z biblioteki *sflib*. *Stateflow chart editor* służy do konstruowania diagramów stanu.

10.6.2. Diagram stanu

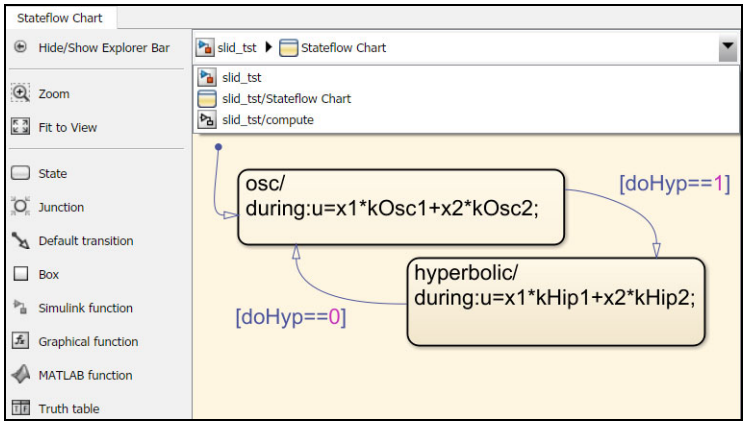
Diagram stanu opisuje logikę działania modelowanego systemu. Kolejne stany obiektu są pokazane jako prostokątne bloki *State* o zaokrąglonych krawędziach. Mogą być one w stanie aktywnym lub nieaktywnym. Przejście (ang. *transition*) z jednego stanu aktywnego do następnego zachodzi w wyniku określonego zdarzenia (ang. *event*) i po spełnieniu podanego warunku.

Na każdej linii przejścia, łączącej sąsiadujące stany, należy podać warunek realizacji tego przejścia. Ma on postać: nazwa zdarzenia (ang. *event*), warunek jako wyrażenie logiczne (ang. *guard condition*) i ewentualna akcja realizowana przed wejściem do nowego stanu, czyli *zdarzenie[warunek]/akcja*. Przejście może zachodzić tylko we wskazanym kierunku. Stan początkowy systemu jest wskazany przez przejście domyślne (linia z czarną kulką).

Diagram stanu buduje się z ikon umieszczonych przy lewej krawędzi okna edycyjnego. Przejścia rysuje się, łącząc wybrane stany za pomocą myszy.

Działanie systemu programuje się graficznie, poprzez odpowiednie połączenie i opisanie elementów diagramu. Każdy blok stanu powinien mieć nadaną nazwę. Widoczny na rysunku 10.6.2-1 ukośnik / jest separatorem i nie należy do nazwy. Akcje, czyli opis tego, co się dzieje podczas aktywności danego stanu, wpisuje się wewnątrz bloku danego stanu. Akcję `entry` można też zapisać obok linii przejścia, po zakończonym znakiem / warunku uaktywniającym dany stan. Akcje poprzedza się prefiksami. Najważniejsze prefiksy podano poniżej:

- `entry:` — skrót `en:`, akcja realizowana *przy wchodzeniu* w stan aktywności,
- `during:` — skrót `do:`, *w trakcie* stanu aktywności,
- `exit:` — skrót `ex:`, *przy wychodzeniu* z aktywności,
- *`nazwa_zdarzenia:`* — w razie zajścia określonego *zdarzenia*.

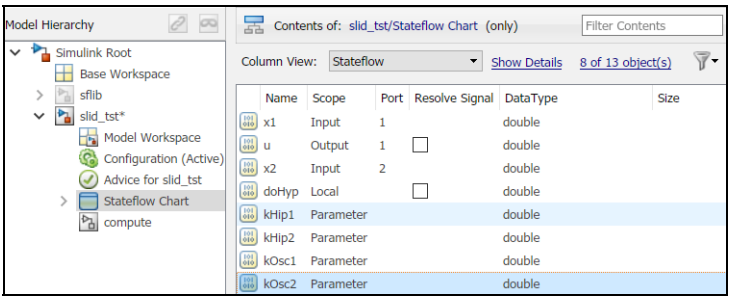


Rysunek 10.6.2-1. *Diagram stanu realizujący logikę sterownika poślizgowego doHyp=(x1*s<=0) &(x1*x2<0)*

W opisie akcji i warunków wykorzystuje się sygnały z Simulinka oraz zmienne z przestrzeni roboczej MATLAB-a. Muszą one być zadeklarowane przed ich użyciem. Zmienne deklarowane jako wejście i wyjście z Simulinka są widoczne jako bramy wejściowe i wyjściowe bloku *Stateflow Chart* (rysunek 10.6.1-1).

Zadeklarowane zmienne można przeglądać w oknie *Model Explorer* (rysunek 10.6.2-2) po wybraniu z menu opcji *Tools/Model Explorer*. W diagramie *Stateflow Chart* zadeklarowano zmienne: `x1`, `x2`, `s` jako wejścia z Simulinka, zmienną `u` jako wyjście z Simulinka. Stałe `kHip1`, `kHip2`, `kOsc1`, `kOsc2` są pobierane z przestrzeni roboczej MATLAB-a.

Rysunek 10.6.2-2. *Okno Model Explorer pokazuje hierarchię obiektów diagramu i zmienne wybranego obiektu*

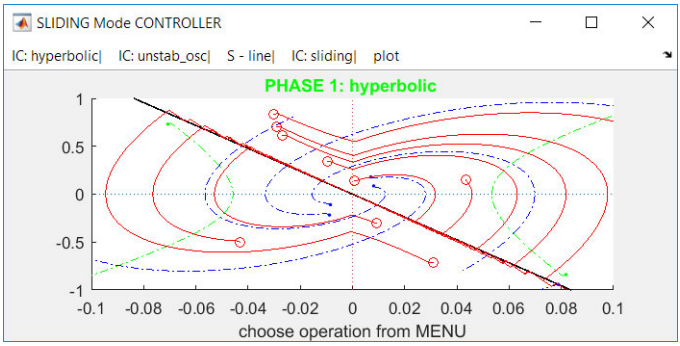


Celem zadeklarowania zmiennych wybiera się z menu opcję *Chart/Add Inputs & Outputs*. Powoduje to otwarcie okna *Data data*. Zaleca się, aby unikać deklarowania macierzy, gdyż działania na macierzach bądź wektorach nie są w *Stateflow* tak proste jak w MATLAB-ie.

10.6.3. Symulacja sterowania poślizgowego

Uruchomienie symulacji ruchu platformy mobilnej ze sterownikiem poślizgowym wymaga pobrania kilku dodatkowych, niewielkich plików. Po wpisaniu nazwiska autora `Mrozek` w okienku *Search Files* na stronie *http://www.mathworks.com/matlabcentral/fileexchange* należy wybrać link do *Sliding Mode controller* i pobrać umieszczony tam plik zawierający oprogramowanie i opis jego uruchomienia. Wyniki symulacji zależą od nachylenia linii przełączającej oraz od wybranych warunków początkowych (kółka na wykresie) i są podobne do pokazanych na rysunku 10.6.3-1.

Rysunek 10.6.3-1. *Linie ciągłe pokazują trajektorię platformy na wykresie fazowym*



10.6.4. Generowanie kodu dla modeli

Simulink Coder lub *Embedded Coder* mogą wygenerować kod w języku C lub C++ z modeli Simulinka oraz dla zawartych w nich bloków Stateflow.

Kod wygenerowany przez *Simulink Coder* może być zastosowany do szybkiego prototypowania i do pracy w czasie rzeczywistym, w tym do symulacji *HiL* (ang. *hardware in the loop*).

Embedded Coder jest natomiast używany do tworzenia oprogramowania przeznaczonego do uruchomienia na mikroprocesorach i urządzeniach docelowych (ang. *target system*).

Wykorzystanie *HDL Coder* pozwala na wygenerowanie przenośnego kodu *Verilog* lub *VHDL* dla funkcji MATLAB-a, dla modeli Simulinka i modeli *Stateflow*.

Wygenerowany kod *HDL* może być używany do programowania *FPGA* (ang. *field-programmable gate array*) oraz do prototypowania i projektowania *ASIC* (ang. *application-specific integrated circuit*).

10.7. Dodatkowe moduły rozszerzające środowisko Simulinka

Producent MATLAB-a oferuje za opłatą dodatkowe moduły rozszerzające środowisko Simulinka:

- **Modelowanie systemów sterowanych zdarzeniami** — narzędzia *Stateflow* i *SimEvents*.
- **Wirtualne modelowanie fizyczne** oraz integracja podsystemów tworzonych w różnych technologiach (ang. *interdysciplinary*, *multiphysics*, *multidomain*) w jeden system — narzędzia *Simscape*, *Simscape Multibody*, *Simscape Driveline*, *Simscape Fluids*, *Simscape Electronics*, *Simscape Power Systems*.
- **Analiza i projektowanie systemów sterowania** — narzędzia *Simulink Control Design*, *Simulink Design Optimization*, *Aerospace Blockset*, *Robotics System Toolbox*, *Powertrain Blockset*.
- **Przetwarzanie sygnałów i telekomunikacja** — narzędzia *DSP System Toolbox*, *Audio System Toolbox*, *Communications System Toolbox*, *Phased Array System Toolbox*, *SimRF*, *Computer Vision System Toolbox*.
- **Automatyczne generowanie kodu** — narzędzia *Simulink Coder*, *Embedded Coder*, *HDL Coder*, *Vision HDL Toolbox*, *Simulink PLC Coder*, *Fixed — Point Designer*, *DO Qualification Kit (for DO-178)*, *IEC Certification Kit (for IEC 61508 and ISO 26262)*.
- **Przygotowanie kodu oraz bezpośrednie sterowanie** urządzeniami mikroprocesorowymi (i monitorowanie ich), takimi jak *ARM*, *Arduino*, *Altera*, *National Instruments*, *Raspberry Pi*, *Xilinx*, *Android*, *STMicroelectronics*, *Keysight* i ponad 100 innych z wykazu *Explore hardware by vendor* na stronie *www.mathworks.com/hardware-support/home.html*.

- **Szybkie prototypowanie i symulacja HiL** (sprzęt w pętli, ang. *hardware in the loop*) — narzędzia *Simulink Real-Time*, *Simulink Desktop Real-Time*.
- **Weryfikacja, walidacja i testowanie** — narzędzia *Simulink Verification and Validation*, *Simulink Design Verifier*, *Simulink Test*, *Simulink Code Inspector*, *HDL Verifier*, *Polyspace Bug Finder*, *Polyspace Code Prover*.
- **Animacja 3D i generowanie raportów** — narzędzia *Simulink 3D Animation*, *Simulink Report Generator*.

Na stronie *http://www.mathworks.com/products/pfo* (rysunek 1.2-1) jest interaktywna mapa *dostępnych odpłatnie* rozszerzeń MATLAB-a (ok. 60), rozszerzeń Simulinka (ok. 40) oraz innych usług i produktów (prawa kolumna). Kliknięcie wybranego obszaru tej strony powoduje wyświetlenie linków i nazw rozszerzeń przeznaczonych do obsługi danych zastosowań. Kliknięcie kolejnych linków wyświetla szczegółowe opisy wybranych rozszerzeń.

Rozdział 11.

Dodatek

W dodatku opisano wybrane rozszerzenia i funkcje MATLAB-a. Dokumentacja wszystkich zainstalowanych rozszerzeń jest dostępna w postaci plików PDF oraz poprzez polecenia `help` i `doc`, a także w postaci *demo* i *pokazów video*. Wiele informacji oraz zaproszenia na webinaria można znaleźć na stronach *www.mathworks.com* i *www.ont.com.pl*.

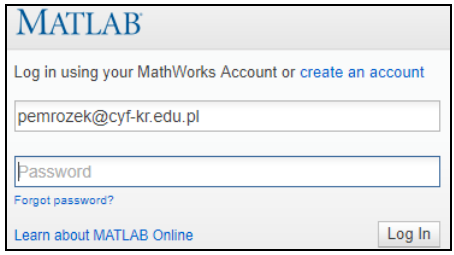
Na stronie *http://www.mathworks.com/products/pfo* jest interaktywna mapa *dostępnych odpłatnie* rozszerzeń MATLAB-a (ok. 60), rozszerzeń Simulinka (ok. 40) oraz innych usług i produktów. Kopia mapy jest na rysunku 1.2-1, ale nie widać tam szczegółów, które pojawiają się po kliknięciu elementów rysunku.

11.1. MATLAB Online i MATLAB Drive

MATLAB Online udostępnia możliwość programowania, uruchamiania i wykonywania programów w MATLAB-ie w chmurze bez potrzeby instalowania i aktualizowania MATLAB-a na komputerze. Pulpit MATLAB-a jest udostępniany w oknie przeglądarki internetowej pod adresem *matlab.mathworks.com*.

Z usługi *MATLAB Online* i *MATLAB Drive* (synchronizuje pliki lokalne i pliki w chmurze) mogą korzystać użytkownicy licencji indywidualnych, mający wykupioną subskrypcję uaktualnień MATLAB-a. Również licencje uczelniane zwane *TAH* i *TSH* dają dostęp do *MATLAB Online*. Zalecaną przeglądarką jest Google Chrome. Smartfony i tablety (w tym iPhone, iPad oraz Android) zamiast do *MATLAB Online* mogą się logować do *MATLAB Mobile*.

Opis *MATLAB Online* jest dostępny na stronie *www.mathworks.com/products/matlab-online.html*, a *MATLAB Drive* pod adresem *drive.mathworks.com*. Okienko logowania do *MATLAB Online* pokazano na rysunku 11.1-1. Przykłady użycia *MATLAB Online* i *MATLAB Drive* zamieszczono w podrozdziale 11.2.2 i pokazano na rysunkach 11.2.2.1-1 i 11.2.2.2-1.



Rysunek 11.1-1. *Okno logowania do MATLAB Online*

11.2. Big data, obsługa dużych zbiorów danych

Przetwarzanie zbiorów danych o dużej objętości (ang. *volume*), zmienności (ang. *velocity*), różnorodności (ang. *variety*) oraz przydatności i wiarygodności (ang. *value*) wymaga użycia nowych, przeznaczonych do tego narzędzi. Zmienność związana jest zazwyczaj ze stałym zwiększaniem objętości danych i ewentualną zmianą ich struktury w kolejnych plikach z danych. Dane, a w szczególności rezultaty eksploracji tych danych (ang. *data mining*) powinny być weryfikowane pod kątem ich wiarygodności i przydatności. Użyte powyżej określenia (*volume*, *velocity*, *variety*, *value*) są znane jako *model 4V*, a dane opisywane tym modelem są nazywane *Big data*.

Przetwarzanie dużych ilości danych wymaga użycia nowych narzędzi, które są wciąż tworzone i ulepszane. Klasyczne techniki stosowane przy obsłudze baz danych są mało efektywne lub nieprzydatne, głównie z uwagi na dużą objętość i różnorodność danych.

Programy przygotowane na *lokalnym komputerze* mogą być *przeskalowane* do pracy w systemach do rozproszonego przetwarzania i składowania dużych zbiorów w chmurze. Znanymi przykładami są Apache Hadoop (wykorzystuje on rozproszony system plików HDFS, ang. *Hadoop Distributed File System*), Amazon S3 (ang. *Simple Storage Service*) oraz implementacja `MapReduce` i wiele innych narzędzi. Apache Hadoop nie będzie tu omawiany.

W MATLAB-ie do obsługi dużych zbiorów danych używa się następujących narzędzi:

- Obiekt `datastore` — to wirtualny kontener umożliwiający łatwy dostęp do dużych danych, które mogą być dostępne w postaci jednego lub wielu plików. W podrozdziale 11.2.2 opisano jego warianty: `TabularTextDatastore`, `ImageDatastore`, `SpreadsheetDatastore`, `FileDatastore`, `DatabaseDatastore`. Dodatkowo `KeyValueDatastore` jest używane przez `MapReduce`.
- *Tall table* — umożliwia pracę z dużymi zbiorami danych, które nie mieszczą się w pamięci. *Tall table* może mieć dowolną liczbę wierszy. Użycie *Tall table* powoduje, że większość operatorów i funkcji macierzowych MATLAB-a działa tak samo jak z macierzami w pamięci, choć w rzeczywistości MATLAB przetwarza kolejno małe fragmenty dużych plików z danymi. Nie ma przeszkód, aby macierz lokalną zadeklarować jako *Tall table* (podrozdział 11.2.3).

- `MapReduce` — jest techniką do analizowania danych, które nie mieszczą się w pamięci. Jest ona dostępna w MATLAB-ie bez dodatkowych opłat. `Map` przetwarza kolejne porcje danych pobieranych przez obiekt `datastore`, a wyniki pośrednie są przekazywane do `Reduce` (podrozdział 11.2.4).

11.2.1. Zbiory danych używane do badań i testowania

11.2.1.1. Pliki dostępne lokalnie

W folderze *toolbox\matlab\demos* znajdują się pliki *airlinesmall.csv*, *airlinesmall_subset.xlsx*, *patients.xls* i *outages.csv*, których można użyć jako lokalnych danych do testowania (użyto ich w podrozdziałach 11.2.2.1 i 11.2.2.4 do 11.2.2.5). Plik *airlinesmall_subset.xlsx* jest podzbiorem danych z pliku *airlinesmall.csv*. Liczne pliki graficzne z folderu *\help\matlab* wykorzystano jako dane w podrozdziale 11.2.2.3.

11.2.1.2. Dane biznesowe, geograficzne i inne udostępniane publicznie

Liczne agencje rządowe, firmy i uczelnie udostępniają swoje wybrane zbiory danych, szczególnie jeśli są one rezultatem projektów finansowanych ze środków publicznych. Przykładowo:

- Amazon Web services, *https://aws.amazon.com/free*, roczny bezpłatny dostęp do baz danych AWS Lambda, Amazon S3, Amazon RDS, Amazon Quick Sight, Amazon EC2 i inne;
- *http://snap.stanford.edu/data/*, Stanford Large Network Dataset Collection;
- na stronie *http://stat-computing.org/dataexpo/2009/the-data.html* dostępne są linki do danych o lądowaniach i startach samolotów amerykańskich linii lotniczych w latach od 1987 do 2008; dane są skompresowane (rozszerzenie *.bz2*); dołączony do MATLAB-a plik *airlinesmall.csv* zawiera dane ze znajdujących się tam plików;
- na stronie *https://www.data.gov/* umieszczono linki do danych udostępnianych przez agendy rządowe USA;
- swoje dane udostępnia miasto Nowy Jork; linki do danych archiwalnych o kursach taksówek Yellow Taxi i Green Taxi od roku 2009 są dostępne po wpisaniu *Taxi Trip Data* w okienku *search* na stronie *https://data.cityofnewyork.us/*;
- na stronie *https://catalog.data.gov/dataset* umieszczono linki do ponad 190 zbiorów danych;
- na stronie *http://climatedataapi.worldbank.org/climateweb/rest/v1/* są dostępne dane banku światowego o klimacie;
- na stronie *http://www.stateair.net/web/historical/1/1.html* są dostępne linki do danych z pomiarów zanieczyszczenia atmosfery w Chinach; dane te zostały użyte w podrozdziale 11.2.2.2;
- wiele serwerów z danymi można znaleźć po wpisaniu słów *free data* w przeglądarce internetowej.

11.2.2. Dostęp do danych przez obiekt datastore

Obiekt `datastore` to wirtualny kontener umożliwiający łatwy dostęp do dużych danych, które mogą być dostępne w postaci jednego lub wielu plików — jeśli każdy z nich zawiera dane tego samego typu (np. numeryczne, tekstowe), występujące w tej samej kolejności i oddzielone tym samym separatorem. Polecenie

```
>>ds = datastore(location,Name,Value)
```

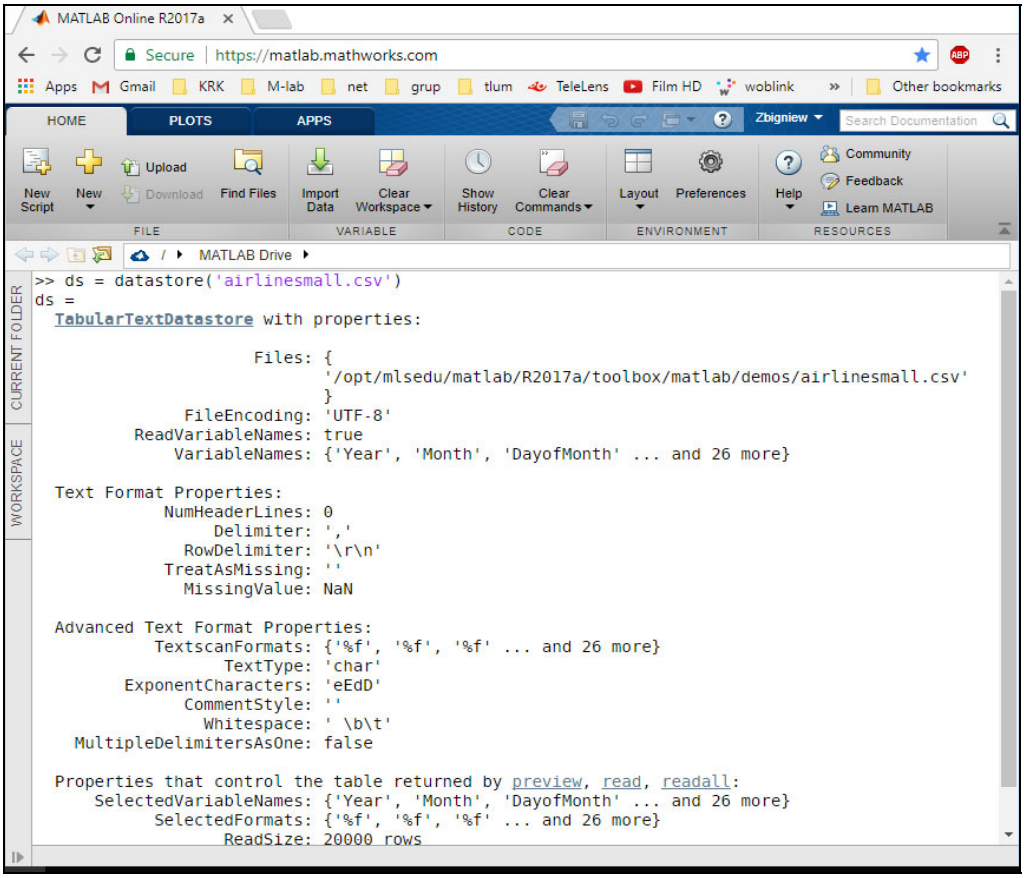
tworzy wirtualny kontener, obiekt `ds`.

11.2.2.1. Dane lokalne, jeden plik csv — TabularTextDatastore

Datastore może udostępniać także dane lokalne, co pozwala na łatwe przetestowanie sposobów pracy z dużymi plikami danych. W celu przetestowania dostępu do jednego pliku *.csv* (ang. *coma separated variable* — dane tekstowe oddzielane przecinkiem) wykonano polecenie:

```
>> ds = datastore('airlinesmall.csv')
```

Rezultat wykonania tego polecenia w chmurze MATLAB Online pokazano na rysunku 11.2.2.1-1.

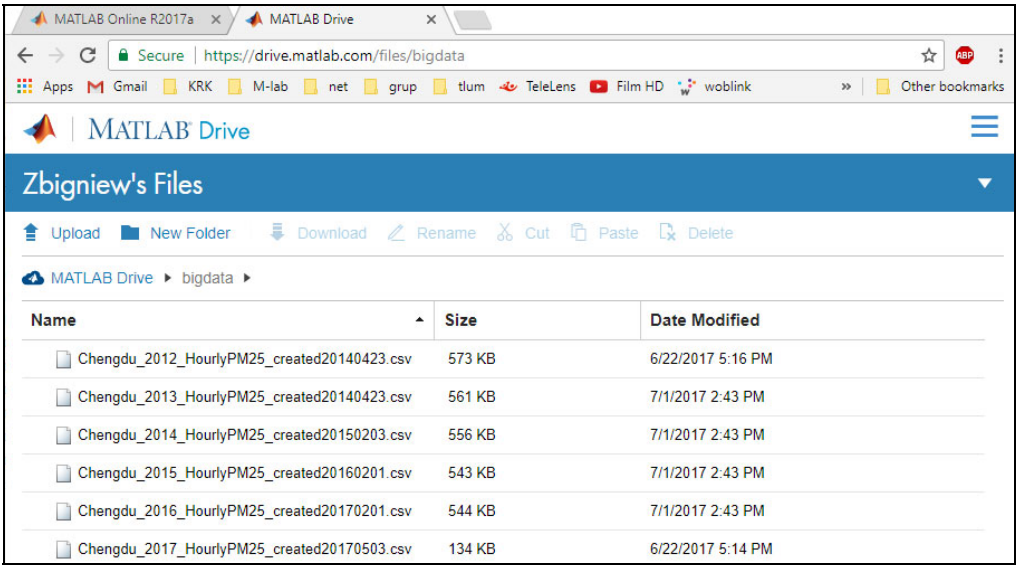


Rysunek 11.2.2.1-1. *Utworzenie obiektu TabularTextDatastore dla pliku lokalnego airlinesmall.csv*

11.2.2.2. Pliki w formacie csv — MATLAB Drive i sześć plików w chmurze MATLAB Online

Obiekt `TabularTextDatastore` utworzono dla kilku lokalnych plików z wynikami pomiaru zanieczyszczenia powietrza pyłem zawieszonym PM 2.5, mierzonym co godzinę w miejscowości Chengdu w Chinach. Dane pomiarowe (w oddzielnych plikach na każdy rok) pobrano na komputer lokalny ze zlokalizowanego w USA serwera *http://www.stateair.net/web/historical/1/1.html*.

Aby te pliki były dostępne w MATLAB Online i w MATLAB Mobile, przekopiowano je do chmury z użyciem MATLAB Drive. Pierwszym krokiem było otwarcie strony *https://drive.matlab.com* i zalogowanie się hasłem do MATLAB Online. Następnie utworzono w chmurze folder *bigdata* i po kliknięciu ikony *Upload* skopiowano sześć plików z danymi o smogu w Chengdu komputera lokalnego do chmury (rysunek 11.2.2.2-1). Aplikacja MATLAB Drive nie była pobierana.



Rysunek 11.2.2.2-1. *MATLAB Drive — pliki zaimportowane z lokalnego komputera*

Obiekt typu `datastore` może być utworzony poleceniami:

```
varnames = {'Date_LST_','Site', 'Parameter',  'Value','QCName'}
ds = datastore('bigdata/Chengdu*.csv','SelectedVariableNames', varnames)
```

Aby utworzyć obiekt typu `datastore` na komputerze lokalnym, należy podać ścieżkę dojścia do lokalnych plików *Chengdu*.csv*. Więcej szczegółów podano w podrozdziale 11.2.3.2.

11.2.2.3. Lokalne pliki graficzne — ImageDatastore na MATLAB Online

Funkcja `datastore` obsługuje różne typy danych. Poniżej podano przykład utworzenia obiektu należącego do klasy `ImageDatastore`. Przykład można wykonać na komputerze lokalnym lub w chmurze *MATLAB Online*.

```
>> dsImg =  datastore(fullfile(matlabroot, 'help', 'matlab'),...
    'IncludeSubfolders', true,'FileExtensions', {'.png','.jpg'}','Type', 'image')
```

Pierwszym parametrem `datastore` jest pełna ścieżka dojścia do folderu *help/matlab*, określona przez funkcję `fullfile` i jej parametry: główny folder MATLAB-a *matlabroot* oraz jego podfoldery *help/matlab*. Chmura MATLAB Online pracuje w systemie Linux i ścieżki dojścia zapisuje się z użyciem ukośnika `/`.

Kolejne parametry mają postać `key-value`, np. `'FileExtensions', {'.png','.jpg'}'`, gdzie klucz to rozszerzenie pliku, a wartość to tablica dwuelementowa typu *cell array* o wartościach `{'.png','.jpg'}`.

Potwierdzenie utworzenia przez funkcję `datastore` wirtualnego magazynu danych `dsImg` typu `ImageDatastore` zawierającego 3+2528 plików graficznych z folderu */opt/mlsedu/matlab/R2017a/help/matlab/* i ewentualnie jego *podfolderów* wygląda następująco:

```
dsImg =

  ImageDatastore with properties:

       Files: {
              '/opt/mlsedu/matlab/R2017a/help/matlab/12b-demos.png';
              '/opt/mlsedu/matlab/R2017a/help/matlab/12b-new_tab.png';
              '/opt/mlsedu/matlab/R2017a/help/matlab/12b_cftoolbar.png'
               ... and 2528 more
              }
    ReadSize: 1
      Labels: {}
     ReadFcn: @readDatastoreImage
```

11.2.2.4. Arkusz kalkulacyjny SpreadsheetDatastore — lokalnie lub na MATLAB Online

Utworzono wirtualny magazyn typu `SpreadsheetDatastore` dla pliku lokalnego *tsunamis.xlsx*. Jedynym parametrem funkcji `datastore` jest ścieżka dojścia do pliku.

```
>> dsX =  >> dsX = datastore('tsunamis.xlsx')
```

Rezultatem jest utworzenie `dsX`, wirtualnego magazynu `SpreadsheetDatastore` z polami `{'Latitude', 'Longitude', 'Year'` ... i 17 innymi`}`.

11.2.2.5. Lokalne MAT-pliki binarne — fileDatastore na MATLAB Online lub lokalnie

Utworzono obiekt typu `FileDatastore` dla plików w chmurze. Koniecznym warunkiem utworzenia takiego obiektu jest podanie dwóch par `key-value`. Pierwsza para to klucz `'ReadFcn'` i wartość: `@load`, uchwyt do funkcji czytającej te pliki. Druga para to klucz `'FileExtensions'` i jego wartość `'.mat'`.

```
>> fds = fileDatastore(fullfile(matlabroot,'toolbox','matlab','demos'),...
         'ReadFcn',@load,'FileExtensions','.mat')
fds =
  FileDatastore with properties:

      Files: {
             '/opt/mlsedu/matlab/R2017a/toolbox/matlab/demos/accidents.mat';
             '/opt/mlsedu/matlab/R2017a/toolbox/matlab/demos/airfoil.mat';
```

```
            '/opt/mlsedu/matlab/R2017a/toolbox/matlab/demos/airlineResults.mat'
             ... and 37 more
            }
  ReadFcn: @load
```

11.2.2.6. Lokalne, binarne MAT-pliki — fileDatastore na komputerze lokalnym

Utworzono obiekt typu `FileDatastore` dla plików lokalnych. Koniecznym warunkiem utworzenia takiego obiektu jest podanie dwóch par `key-value`. Pierwsza para to klucz `'ReadFcn'` i wartość: `@load`, uchwyt do funkcji czytającej te pliki. Druga para to klucz `'FileExtensions'` i jego wartość `'.mat'`. Wczytano te same pliki co w poprzednim podrozdziale, ale inne są ścieżki dojścia.

```
>> fds = fileDatastore(fullfile(matlabroot,'toolbox','matlab','demos'),...
         'ReadFcn',@load,'File Extensions','.mat')
fds =
  FileDatastore with properties:

          Files: {
                 'G:\Mlab\R2017a\toolbox\matlab\demos\accidents.mat';
                 'G:\Mlab\R2017a\toolbox\matlab\demos\airfoil.mat';
                 'G:\Mlab\R2017a\toolbox\matlab\demos\airlineResults.mat'
                  ... and 37 more
                 }
    UniformRead: 0
        ReadFcn: @load
```

11.2.2.7. Utworzenie obiektu DatabaseDatastore do obsługi lokalnej i odległej bazy danych

Obiekt `DatabaseDatastore` może być utworzony do współpracy z bazami relacyjnymi, ale wymaga to zainstalowania *Database Toolbox*. Korzystanie z *HDFS* (ang. *Hadoop Distributed File System*) lub Amazon S3 (ang. *Simple Storage Service*) wymaga dostępu do odpowiedniego oprogramowania.

Więcej informacji można znaleźć, podając w okienku *Search* na stronie *www.mathworks.com* odpowiedni tekst: „Read Remote Data” w przypadku Amazon S3 i HDFS lub „DatabaseDatastore” w przypadku baz relacyjnych.

11.2.2.8. Polecenia do obsługi obiektów typu datastore

Podane poniżej polecenia mogą być użyte do obsługi każdego obiektu typu `datastore`:

- `preview` — wczytuje kilka lini danych z początku `datastore`,
- `read` — wczytuje porcję danych, rozpoczynając od aktualnego położenia,
- `readall` — wczytuje wszystkie dane,
- `hasdata` — ma wartość `TRUE`, jeśli w `datastore` pozostały niewczytane dane,
- `reset` — przewija `datastore` na początek danych.

Więcej informacji podaje `doc datastore`.

11.2.3. Tall array i tall table

Użycie zmiennych typu *tall array* radykalnie ułatwia pracę z dużymi zbiorami danych, które nie mieszczą się w pamięci. *Tall array* może mieć dowolną liczbę wierszy, gdyż MATLAB ładuje do pamięci tylko jej niewielkie fragmenty i przetwarza je kolejno po kawałku. Pomimo to większość operatorów i funkcji macierzowych MATLAB-a działa poprawnie, czyli tak, jakby miały stały i pełny dostęp do wszystkich elementów. Jeśli dane w *tall array* są tabelą, to używa się nazwy *tall table.* Wykaz operacji i funkcji, które są poprawnie realizowane na *tall table*, podaje polecenie `methods(tall)`.

Użycie *tall array* powoduje, że tworzenie programów do obsługi dużych danych jest znacznie szybsze i łatwiejsze, bo programista nie musi uwzględniać, że MATLAB przetwarza kolejno małe fragmenty dużych plików z danymi.

11.2.3.1. Utworzenie tall table dla macierzy lokalnej

Korzystając z MATLAB Online, utworzono dużą macierz A o wymiarach 10 000 000×8. Następnie poleceniem `tA = tall(A)` utworzono *Tall table*, w której umieszczono zawartość macierzy A — zalecany etap pośredni z użyciem `datastore` pominięto.

Przykład 11.2.3.1-1

Utworzenie *tall table* dla macierzy lokalnej A o wymiarach 10 000 000×8.

```
>> A = rand(10000000,8);
tA = tall(A)
tA =
  10,000,000×8 tall double matrix
    0.9270    0.8462    0.4771    0.0220    0.6223    0.5153    0.2537    0.1040
    0.0695    0.0035    0.4154    0.7303    0.0369    0.7841    0.1255    0.4501
    0.7848    0.5550    0.4879    0.7565    0.2262    0.6718    0.6661    0.5628
    0.9381    0.8127    0.3259    0.1056    0.4044    0.2315    0.5801    0.0700
    0.1797    0.7964    0.0588    0.7534    0.7891    0.6118    0.7222    0.5074
    0.2431    0.4922    0.3494    0.2116    0.9714    0.5912    0.4117    0.5795
    0.6647    0.3436    0.1467    0.2900    0.7446    0.4409    0.7072    0.7618
    0.7060    0.2454    0.3319    0.7524    0.1108    0.0632    0.3551    0.7133
    :         :         :         :         :         :         :         :
    :         :         :         :         :         :         :         :
```

Korzyści z użycia *Tall table* pokazuje polecenie `whos` w rubryce `size`.

```
>> whos
  Name              Size                Bytes  Class     Attributes
  A          10000000x8            640000000  double
  tA       10,000,000x8                   45  tall
```

Zalecanym sposobem tworzenia *tall table* jest wcześniejsze użycie `datastore`. W tym celu macierz `A` zapisano w pliku (polecenie `save A8 A % A8` jest nazwą pliku *.mat*), po czym wykonano jedno z równoważnych poleceń.

```
>> dsA8fd=fileDatastore('A8.mat','ReadFcn',@load,'FileExtensions','.mat')
```

lub

```
>> dsA8ds=datastore('A8.mat','Type','fileDatastore','ReadFcn',@load,…
        'FileExtensions','.mat')
```

Istotne jest zadeklarowanie, że z pliku typu *.mat* będzie utworzone `FileDatastore`, a odczyt z *MAT*-pliku realizuje funkcja z uchwytem `@load`.

Cały proces tworzenia dużej tablicy *tall table* z pliku *.mat* wygląda następująco:

```
>> dsA8fd=fileDatastore('A8.mat','ReadFcn',@load,'FileExtensions','.mat');
>> tA=tall(dsA8fd)
```

11.2.3.2. Utworzenie tall table dla dużego zestawu danych

Aby utworzyć *tall table* dla dużego zestawu danych w formacie CSV (ang. *comma separated values*), należy wpierw utworzyć obiekt typu `TabularTextDatastore`.

Przykład 11.2.3.2-1

Utworzenie obiektu typu `TabularTextDatastore` dla danych z Chengdu.

```
>> varnames = {'Date_LST_','Site', 'Parameter',  'Value','QCName'} %wybór kolumn
varnames =
  1×5 cell array
    'Date_LST_'    'Site'    'Parameter'    'Value'    'QCName'

>> ds = datastore('bigdata/Chengdu*.csv','SelectedVariableNames', varnames)
Warning: Variable names were modified to make them valid MATLAB identifiers.
ds =
  TabularTextDatastore with properties:

                     Files: {
                            '/MATLAB Drive/bigdata/...
                            Chengdu_2012_HourlyPM25_created20140423.csv';
                            '/MATLAB Drive/bigdata/...
                            Chengdu_2013_HourlyPM25_created20140423.csv';
                            '/MATLAB Drive/bigdata/...
                            Chengdu_2014_HourlyPM25_created20150203.csv'
                             ... and 3 more
                            }
              FileEncoding: 'UTF-8'
        ReadVariableNames: true
            VariableNames: {'Site', 'Parameter', 'Date_LST_' ... and 8 more}

  Text Format Properties:
            NumHeaderLines: 3
                 Delimiter: ','
              RowDelimiter: '\r\n'
            TreatAsMissing: ''
              MissingValue: -999

  Advanced Text Format Properties:
          TextscanFormats: {'%q', '%q', '%D' ... and 8 more}
                 TextType: 'char'
       ExponentCharacters: 'eEdD'
             CommentStyle: ''
               Whitespace: ' \b\t'
  MultipleDelimitersAsOne: false

  Properties that control the table returned by preview, read, readall:
    SelectedVariableNames: {'Date_LST_', 'Site', 'Parameter' ... and 2 more}
          SelectedFormats: {'%D', '%q', '%q' ... and 2 more}
                 ReadSize: 20000 rows
```

Przykład 11.2.3.2-2

Utworzenie *tall table* dla danych z Chengdu.

```
>> tChengdu = tall(ds)
tChengdu =
  M×5 tall table
       Date_LST_          Site       Parameter    Value     QCName
    _______________    _________    _________    _____    _________
    2012-01-01 00:00    'Chengdu'    'PM2.5'      -999     'Missing'
    2012-01-01 01:00    'Chengdu'    'PM2.5'      -999     'Missing'
    2012-01-01 02:00    'Chengdu'    'PM2.5'      -999     'Missing'
    2012-01-01 03:00    'Chengdu'    'PM2.5'      -999     'Missing'
    2012-01-01 04:00    'Chengdu'    'PM2.5'      -999     'Missing'
    2012-01-01 05:00    'Chengdu'    'PM2.5'      -999     'Missing'
    2012-01-01 06:00    'Chengdu'    'PM2.5'      -999     'Missing'
    2012-01-01 07:00    'Chengdu'    'PM2.5'      -999     'Missing'
    :                   :            :            :        :
    :                   :            :            :        :
```

Zamiana na `NaN` wartości `-999` w polu `Value` wpisywanej przy braku wyniku pomiaru.

```
>> tChengdu.Value(tChengdu.Value==-999)=NaN % wstaw NaN jeśli TRUE
tChengdu =
  M×5 tall table
       Date_LST_          Site       Parameter    Value     QCName
    _______________    _________    _________    _____    _________
    2012-01-01 00:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 01:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 02:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 03:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 04:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 05:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 06:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 07:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    :                   :            :            :        :
    :                   :            :            :        :
```

Utworzenie *Tall timetable* dla danych z Chengdu; wymiar `Talltimetable` jest równy `M×4`.

```
>> ttimeChengdu=table2timetable(tChengdu)
ttimeChengdu =
  M×4 tall timetable
        Date_LST_          Site       Parameter    Value     QCName
    _______________    _________    _________    _____    _________
    2012-01-01 00:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 01:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 02:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 03:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 04:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 05:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 06:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    2012-01-01 07:00    'Chengdu'    'PM2.5'      NaN      'Missing'
    :                   :            :            :        :
    :                   :            :            :        :
```

Polecenie `whos` pokazuje zmienne w przestrzeni roboczej MATLAB-a: jeden obiekt typu `TabularTextDatastore`, dwa obiekty typu `tall table` i jeden obiekt typu `cell`.

```
>> whos
  Name              Size            Bytes  Class
Attributes

  ds                1x1                 8  matlab.io.datastore.TabularTextDatastore
```

```
tChengdu          Mx5              2366  tall
ttimeChengdu      Mx4              2144  tall
varnames          1x5               626  cell
```

Jeśli jest zainstalowany *Parallel Computing Toolbox™*, to po wydaniu polecenia: `tChengdu = tall(ds)` zostanie uruchomiony tryb pracy równoległej i pojawi się komunikat:

```
Starting parallel pool (parpool) using the 'local' profile ...
```

Polecenie `tail` nie może pokazać ostatnich wierszy *tall table*, bo operacje są odroczone.

```
>> tail (ttimeChengdu)
ans =
  M×4 tall timetable
    Date_LST_     Site    Parameter    Value    QCName
    _________     ____    _________    _____    ______
    ?             ?       ?            ?        ?
    ?             ?       ?            ?        ?
    ?             ?       ?            ?        ?
    :             :       :            :        :
    :             :       :            :        :
Preview deferred. Learn more.
>> gather(tChengdu)    % wykonanie operacji zakolejkowanych dla tChengdu
```

Odroczenie obliczeń pozwala zaplanować operacje w taki sposób, aby zmniejszyć liczbę czasochłonnych przebiegów przez dane. Odroczone operacje dla danej tablicy będą wykonane dopiero po wywołaniu funkcji `gather` z nazwą tablicy jako parametrem. Wyniki obliczeń są zapisywane do pamięci. Należy się liczyć z tym, że wolnej pamięci może zabraknąć.

11.2.3.3 Polecenia do obsługi obiektów typu tall array

Podane poniżej polecenia są używane do obsługi obiektów typu *tall array*:

- `tall` — tworzy obiekt typu *tall array*,
- `mapreducer` — modyfikuje środowisko dla *tall array* i *map reduce*,
- `gather` — wykonanie operacji zakolejkowanych dla obiektu typu *tall array*,
- `head` — pobiera pierwsze linie danych z obiektów typu *tall array*, *table*, *timetable*,
- `tail` — pobiera końcowe linie danych z obiektów typu *tall array*, *table*, *timetable*,
- `topkrows` — pobiera *k* pierwszych linii danych,
- `istall` — TRUE jeśli badany obiekt jest typu *tall array*,
- `classUnderlying` — podaje, do jakiej klasy należą elementy *tall array*,
- `isaUnderlying` — podaje, czy elementy *tall array* należą do wskazanej klasy,
- `write` — zapisuje obiekt typu *tall array* na dysku.

Więcej informacji podaje `doc datastore`.

11.2.4. MapReduce

`MapReduce` jest techniką programowania do analizy danych, które nie mieszczą się w pamięci. `MapReduce` pobiera dane do przetwarzania w małych fragmentach (ang. *chunks*) z obiektu `datasore`, który jest wirtualnym magazynem danych. Kolejne fragmenty danych są wstępnie przetwarzane przez funkcję `Map`. Wyniki pośrednie w postaci par *`klucz_pośredni-obliczona_wartość`* są przez funkcje `add` lub `addmulti` dodawane do magazynu pośredniego, którym jest obiekt `KeyValueStore`. Liczba wywołań funkcji `Map` przez `MapReduce` jest równa liczbie fragmentów danych.

Następnie `MapReduce` wywołuje funkcję `Reduce` w celu przetworzenia wartości przypisanych dla każdego *klucza pośredniego* przekazanego przez `Map` do obiektu `KeyValueStore`. `Reduce` przetwarza wyniki pośrednie i zapisuje wyniki w magazynie wyjściowym.

Szczegółowy opis algorytmu `MapReduce` wraz z przykładami funkcji `Map` i `Reduce` obliczających średnią długość trasy lotniczej w danych z pliku *airlinesmall.csv* można znaleźć w opracowaniu *Getting Started with MapReduce* dostępnym po wpisaniu tytułu tego opracowania w okienku *Search* na stronie *www.mathworks.com*.

11.3. Obliczenia równoległe

Obliczenia równoległe wykorzystują jednoczesną, równoległą pracę wielu procesorów w celu przyspieszenia obliczeń. Z uwagi na upowszechnienie komputerów z procesorami 2 – 8-rdzeniowymi (komputery wieloprocesorowe i komputery z procesorem wielordzeniowym nie będą tu rozróżniane) oraz wieloprocesorowych kart graficznych z technologią *CUDA* obliczenia równoległe stały się łatwo dostępne.

Prócz wymagań sprzętowych niekiedy konieczne jest zakupienie i zainstalowanie *Parallel Computing Toolbox™* oraz posiadanie lub wytworzenie oprogramowania przeznaczonego do pracy równoległej. Jeśli używa się klastra lub komputerów w sieci albo w chmurze, to konieczne jest użycie *MATLAB Distributed Computing Server™*.

11.3.1. Automatyczny tryb pracy równoległej

Jak podano w podrozdziale 8.6.4, obliczenia **równoległe** są **automatycznie** uruchamiane dla niektórych operacji *algebry liniowej* (np. rozkłady LU i QR) oraz dla operacji *tablicowych* (z kropką) na dużych tablicach o podwójnej precyzji, *bez potrzeby* instalowania *Parallel Computing Toolbox™*. Wyraźny efekt przyspieszenia jest widoczny dla macierzy i tablic mających ok. 10 000 i więcej elementów.

Znacznie wyższą wydajność można uzyskać, realizując obliczenia równoległe na klastrach, w sieciach i w chmurach — pod nadzorem *MATLAB Distributed Computing Server™*.

11.3.2. Obliczenia równoległe na komputerze wieloprocesorowym

Parallel Computing Toolbox™ umożliwia znaczne przyspieszenie wykonania zadań intensywnych obliczeniowo w MATLAB-ie, w Simulinku, w *Stateflow* oraz z użyciem

toolboxów. Poniżej (przykład 11.3.2-1) pokazano skrypt zawierający pętlę `for`. Zawarte w skrypcie instrukcje będą wykonywane sekwencyjnie.

Przykład 11.3.2-1

Obliczenia sekwencyjne na jednym rdzeniu procesora, bez obliczeń równoległych:

```
%% 'single processor used / uzyto jednego procesora'
tic                 % start timer / początek pomiaru czasu obliczeń
for tt=10:20,       %% for loop / wykonaj wielokrotnie
    [~,~]=ode23('ode1000',[0,tt],[1,-1]);
    % solving ordinary differential equation given by ode1000.m
    % rozwiązywanie równania różniczkowego (plik ode1000.m)
    % [~,~]  % no output / wyniki obliczeń nie są zapamiętane
end
toc                 % stop timer / koniec pomiaru czasu obliczeń
disp('single processor used / uzyto jednego procesora')
```

Funkcję `ode1000` opisano w podrozdziale 8.2.5. Plik zawierający funkcję `ode1000` należy umieścić w tym samym folderze co powyższy przykład.

```
function xdot=ode1000(t,x);
xdot=zeros(2,1);      % zerowy wektor 2-elementowy
xdot(1)=x(2);  xdot(2)=-1001*x(2) -1000*x(1);
```

Czas sekwencyjnego wykonania tych obliczeń był mierzony przez funkcje `tic` i `toc` i wynosił około 13 sekund.

Aby wykonać te obliczenia równolegle na dwóch rdzeniach procesora, należy otworzyć nową sesję MATLAB-a dla obliczeń równoległych poleceniem `parpool(2)` oraz zastąpić instrukcję `for` przez `parfor`, odpowiednik `for` dla obliczeń równoległych.

Przykład 11.3.2-2

Obliczenia równoległe na procesorze 2-rdzeniowym. Przykład pokazujący łatwość wdrożenia obliczeń równoległych, nie był jednak optymalizowany pod kątem uzyskania dużej szybkości obliczeń.

```
%% Obliczenia równolegle
nc=2    % liczba rdzeni lub procesorów nc = 1,2,3…
parpool('local',nc)    % otwarcie sesji równoległej dla nc rdzeni
tic          % start timer / początek pomiaru czasu obliczeń
%% parallel loop / obliczenia równolegle w pętli for
parfor tt=10:20    % pętla równoległa, dowolna kolejność wykonania pętli
    [~,~]=ode23('ode1000',[0,tt],[1,-1]);
    % solving ordinary differential equation given by ode1000.m
    % rozwiązywanie równania różniczkowego (plik ode1000.m)
    % [~,~] % no output / wyniki obliczeń nie są zapamiętane
end
toc    % Czas obliczeń
delete(gcp)    % Close parallel session / Zamyka sesje równoległą
```

Czas obliczeń równoległych na procesorze 2-rdzeniowym był ok. 40% krótszy i wynosił przy każdym z kilku powtórzeń około 9 sekund (pierwszy przebieg obliczeń trwał dłużej). Obliczenia równoległe są opłacalne tylko dla dużych zadań, na przykład dla macierzy o 10 000 elementów. Tutaj zysk z obliczeń równoległych został utracony, gdyż czynności wstępne (polecenie `parpool`) zajęły ponad dwukrotnie więcej czasu niż obliczenia w pętli `parfor`.

11.3.3. Biblioteki toolbox z bezpośrednim wsparciem dla obliczeń równoległych

Przykład 11.3.2-1 wymagał modyfikacji, aby obliczenia mogły być wykonane równolegle na dwóch rdzeniach procesora. Podobne, lecz bardziej zaawansowane modyfikacje zostały już zaimplementowane w wielu elementach pakietu MATLAB i Simulink. Mogą być one bez żadnych przeróbek użyte do przetwarzania równoległego.

W celu uruchomienia obliczeń równoległych wymagane jest tylko dodatkowe zainstalowanie *Parallel Computing Toolbox* na komputerze wieloprocesorowym oraz zmiana ustawienia kilku parametrów.

Przykładowo w *Optimization Toolbox™* solvery `fmincon`, `fminunc`, `fgoalattain`, `fminimax`, `fsolve`, `lsqcurvefit`, `lsqnonlin` automatycznie przechodzą w tryb pracy równoległej przy numerycznej estymacji gradientów i nieliniowych ograniczeń, jeśli:

- zainstalowano *Parallel Computing Toolbox*,
- co najmniej jedna z opcji: `SpecifyObjectiveGradient` albo `SpecifyConstraintGradient` ma wartość `false`.

Jeśli powyższe warunki są spełnione, obliczenia przejdą w tryb pracy równoległej z chwilą przydzielenia procesorów poleceniem `parpool`.

Aktualne obliczenia *równoległe* są zaimplementowane w *prawie wszystkich* rozszerzeniach MATLAB-a, **nie są** one zaimplementowane tylko w:

- *Filter Design HDL Coder*;
- *MATLAB Compiler* i *MATLAB Compiler SDK*;
- *Spreadsheet Link* (for *Microsoft Excel*);
- *HDL Coder* i *HDL Verifier*;
- *Simulink Design Verifier*, *Simulink Desktop Real-Time* i *Simulink PLC Coder*;
- *Polyspace Client for Ada* i *Polyspace Server for Ada*.

Obliczenia równoległe są dostępne tylko w trybie interaktywnej sesji `parpool` dla:

- *Embedded Coder*;
- *Simulink Coder* i *Simulink Code Inspector*;
- *Simulink Design Verifier*, *Simulink Desktop Real-Time* i *Simulink Report Generator*.

Ponadto *Polyspace Bug Finder* i *Polyspace Code Prover* wymagają przy pracy równoległej użycia dodatkowych kluczy licencyjnych, po jednym dla każdego *workera*. Więcej informacji podaje strona *https://www.mathworks.com/products/parallel-computing/parallel-support.html*.

11.3.4. Obliczenia równoległe z użyciem GPU

CUDA (ang. *compute unified device architecture*) to opracowana przez firmę Nvidia uniwersalna architektura procesorów wielordzeniowych (głównie kart graficznych) umożliwiająca wykorzystanie ich mocy obliczeniowej do rozwiązywania ogólnych

problemów numerycznych. Aby sprawdzić, czy zainstalowana karta graficzna może być użyta do obliczeń równoległych, należy wydać polecenie:

```
>> gpuDevice
```

Jeśli zainstalowana karta graficzna (ang. *graphics processing unit — GPU*) udostępnia technologię *CUDA* i ma odpowiedni driver, to zostanie wyświetlona jej nazwa i lista kilkunastu jej parametrów, np.:

```
>> gpuDevice
ans =
  CUDADevice with properties:
                      Name: 'GeForce GTX 750 Ti'
                     Index: 1
        ComputeCapability: '5.0'
```

Przykład 11.3.4-1

Porównanie czasu obliczeń filtracji cyfrowej z użyciem procesora CPU oraz obliczeń z użyciem procesora graficznego *GPU* w technologii *CUDA*. Przykład pokazuje łatwość wdrożenia obliczeń równoległych. Nie był on jednak optymalizowany pod kątem uzyskania dużej szybkości obliczeń.

```
% Copyright 2014 The MathWorks, Inc.
%% Filter a matrix on CPU
A = magic(5000);
f = ones(1,20)/20;

tic;
B = filter(f, 1, A);
tCPU = toc;

disp(['Total time on CPU:       ' num2str(tCPU)])

%% Filter a matrix on GPU
tic;
AonGPU = gpuArray(A);
BonGPU = filter(f, 1, AonGPU);
BonCPU = gather(BonGPU);
wait(gpuDevice)
tGpu = toc;

disp(['Total time on GPU:       ' num2str(tGpu)])

%% Look at computation time only
tic;
BonGPU = filter(f, 1, AonGPU);
wait(gpuDevice)
tCompGpu = toc;

disp(['Computation time on GPU: ' num2str(tCompGpu)])
```

Czasy obliczeń dla powyższego przykładu z użyciem procesora CPU i karty graficznej GPU typu GeForce GTX 750 Ti wyniosły:

```
Total time on CPU:       0.27161
Total time on GPU:       0.29202
Computation time on GPU: 0.093536
```

Jak widać, czas obliczeń na GPU był krótszy niż na CPU. Jednak zysk z obliczeń równoległych został utracony, gdyż czynności wstępne (funkcje `gpuArray` i `gather`) zajęły więcej czasu niż obliczenia w pętli `parfor`. Potwierdza to, że obliczenia równoległe są opłacalne tylko dla dużych zadań, na przykład dla macierzy o 10 000 elementów.

https://www.mathworks.com/help/distcomp/run-built-in-functions-on-a-gpu.html

11.3.5. Klastry, chmury, sieci lokalne i rozproszone

MATLAB Distributed Computing Server umożliwia znaczne przyspieszenie intensywnych obliczeniowo zadań poprzez ich przeskalowanie i realizację na klastrach, w sieciach i w chmurach. Zadania te powinny być wcześniej uruchomione i przetestowane na sprzęcie wieloprocesorowym z wykorzystaniem *Parallel Computing Toolbox*.

Główne zadania realizowane przez *MATLAB Distributed Computing Server* to:

- Przeskalowanie zadań przygotowanych z wykorzystaniem *Parallel Computing Toolbox* do realizacji przez klastry, sieci i chmury.
- Rozszerzenie licencji z komputera, na którym zainstalowano *Parallel Computing Toolbox*, na komputery i klastry pracujące pod nadzorem *MATLAB Distributed Computing Server*. Pewne ograniczenia obowiązują dla produktów *PolySpace*.
- Obsługa zadań wsadowych, obliczeń równoległych i dużych, rozproszonych zbiorów danych.
- Wykonywanie funkcji obsługujących *GPU* w rozproszonych zasobach komputerowych.
- Wykonanie równoległych obliczeń z aplikacji i komponentów programowych generowanych przy użyciu *MATLAB Compiler™* na zasobach obliczeniowych rozproszonych.

MATLAB Distributed Computing Server posiada własny program szeregujący (ang. *scheduler*, *The MathWorks job manager*). Może on pracować na tych samych platformach sprzętowych i systemach operacyjnych co MATLAB i Simulink. Omawiany serwer jest bezpośrednio kompatybilny z serwerami: *Platform LSF*, *Microsoft Windows HPC Server*, *Altair PBS Pro* i *TORQUE*. Może też współpracować z innymi serwerami, np. *Grid Engine*, poprzez *generic scheduler API*.

11.4. Modele obiektów używanych w Control System Toolbox™, System Identification Toolbox™ i w Robust Control Toolbox™

Omawiane modele należą do klas zdefiniowanych w *Control System Toolbox™*, *System Identification Toolbox™* i w *Robust Control Toolbox™*. Modele należące do klas, których nazwa rozpoczyna się od:

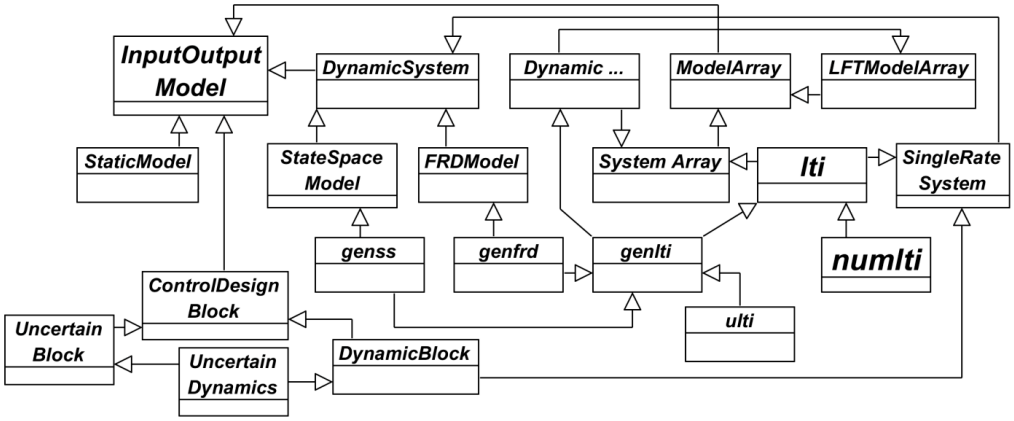
- litery *id* (ang. *identified)* wymagają zainstalowania *System Identification Toolbox™*;
- litery *u* (ang. *uncertain*, niepewne) wymagają zainstalowania *Robust Control Toolbox™*.

Pozostałe modele są zdefiniowane w *Control System Toolbox™* i zazwyczaj mogą być użyte we wszystkich trzech toolboksach. Wszystkie typy danych w środowisku MATLAB są klasami, a zmienne są obiektami należącymi do tych klas. Klasy i obiekty były omawiane w rozdziale 6.

Zestawienie typów modeli LTI oraz ich właściwości podaje polecenie `doc lti`. Listę funkcji tej biblioteki, informacje o jej interfejsie graficznym i przykłady (wraz z krótkim opisem) uzyskuje się za pomocą polecenia `help control`. Przeciążanie (ang. *overloading*) operatorów zdefiniowanych dla obiektu LTI omówiono w podrozdziale 6.4.1. Używa się też nazwy *przesłanianie*.

11.4.1. Abstrakcyjne superklasy definiujące atrybuty i metody wykorzystane w modelach

Modele używane w *Control System Toolbox™*, *System Identification Toolbox™* i w *Robust Control Toolbox™* mają wspólnych przodków w postaci klas pokazanych na rysunku 11.4.1-1. Są to klasy abstrakcyjne, bo w jej parametrach umieszczono atrybut *abstract*, co uniemożliwia tworzenie obiektów należących do tej klasy. Klasy abstrakcyjne odgrywają ważną rolę pojemnika (ang. *container*), w którym umieszcza się atrybuty i metody używane w klasach potomnych. Zmniejsza to liczbę linii programu (nie umieszcza się podobnych fragmentów kodu w wielu miejscach), ułatwia zrozumienie kodu oraz rozbudowę i konserwację oprogramowania. Na diagramach nazwa *klasy abstrakcyjnej* jest pisana *czcionką pochyłą*.



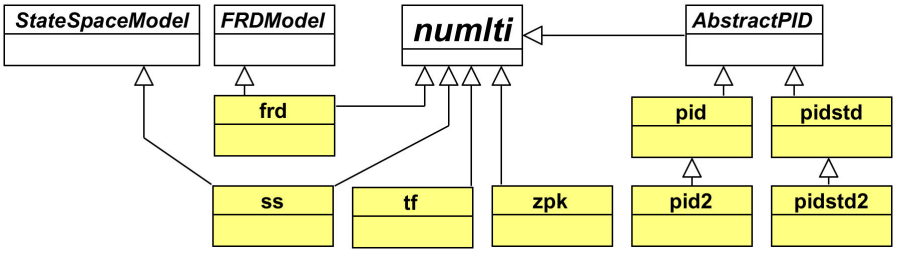
Rysunek 11.4.1-1. *Uproszczona hierarchia klas abstrakcyjnych definiujących atrybuty i metody wykorzystywane w Control System Toolbox, System Identification Toolbox i w Robust Control Toolbox. Diagram wykonano z użyciem programu Visual Paradigm*

Zadaniem tych klas jest zdefiniowanie atrybutów i metod, które będą dziedziczone w klasach potomnych. Takie podejście ułatwia wymianę danych, modeli i wyników obliczeń pomiędzy omawianymi toolboksami, a także ich przekazywanie do innych elementów środowiska MATLAB, w tym do Simulinka.

11.4.2. Klasa numlti oraz modele typu tf, zpk, ss i inne

Obiekty należące do klas `tf`, `zpk`, `ss`, `frd` są potomkami klasy abstrakcyjnej `numlti` (ang. *numerical lti*). Klasy dziedziczące od `numlti` mają stałe parametry, zapisane jako liczby (ang. *fixed numerical values*). Są one używane jako modele *liniowe* systemów lub elementów automatyki. Ich parametry są *numeryczne* i *niezmienne w czasie* (ang. *time inwariant*) (rysunek 11.4.2-1).

Obiekty należące do klas `pid`, `pid2`, `pidstd`, `pidstd2` są różnymi wariantami regulatorów proporcjonalno-całkująco-różniczkujących. Są one potomkami klasy abstrakcyjnej `AbstractPID`, a ona z kolei jest potomkiem `numlti`. Jej klasy nadrzędne pokazano na rysunku 11.4.1-1.



Rysunek 11.4.2-1 *Uproszczony diagram klas pokazujący klasy tf, zpk, ss, frd, pid, pid2, pidstd, pidstd2 oraz wybranych tych klas. Diagram wykonano z użyciem programu Visual Paradigm*

Poniżej objaśniono nazwy klas, które są potomkami superklasy abstrakcyjnej `numlti` (rysunek 11.4.2-1):

- `tf` — transmitancja (ang. *transfer function*), model w postaci transmitancji:
  - `filt` — funkcja tworząca filtr dyskretny typu `tf`, zależny od zmiennej `z^-1` (pominięto na diagramie, gdyż nie jest to klasa);
- `zpk` — (ang. *zero/pole/gain*) model z wartościami zer, biegunów i wzmocnienia;
- `ss` — (ang. *state-space*) w postaci równań stanu; na diagramie pominięto:
  - `dss` — funkcja tworząca zmodyfikowany model `ss` (ang. *descriptor state-space*);
  - `rss`, `drss` — funkcje tworzące model typu `ss` ciągły lub dyskretny, z losowymi (ang. *random*) wartościami liczbowymi w macierzach A, B, C, D;
- `frd` — model z wykorzystaniem odpowiedzi w dziedzinie częstotliwości (ang. *frequency response data*);
  - funkcja `chgFreqUnit` zmienia jednostki częstotliwości, np. `'rad/TimeUnit'`, `rpm`, `MHz` itp.

- `pid, pidstd, pid2, pidstd2` — modele sterowników PID;
  - `pid` jest definiowany jako `sys=pid(Kp,Ki,Kd,Tf)` lub dyskretny `sysd=pid(Kp,Ki,Kd,Tf,Ts);`
  - `pidstd` jest definiowany jako `sys=pidstd(Kp,Ti,Td,N)` lub dyskretny `sysd=pid(Kp,Ti,Td,N,Ts);`
  - modele należące do `pid2` oraz do `pidst2` są określane jako 2DOF (o dwu stopniach swobody); mają one dwa wejścia: pierwsze na *sygnał zadany* (ang. *reference*) oraz drugie na *sygnał z wyjścia* sterowanego obiektu (ang. *plant output*).

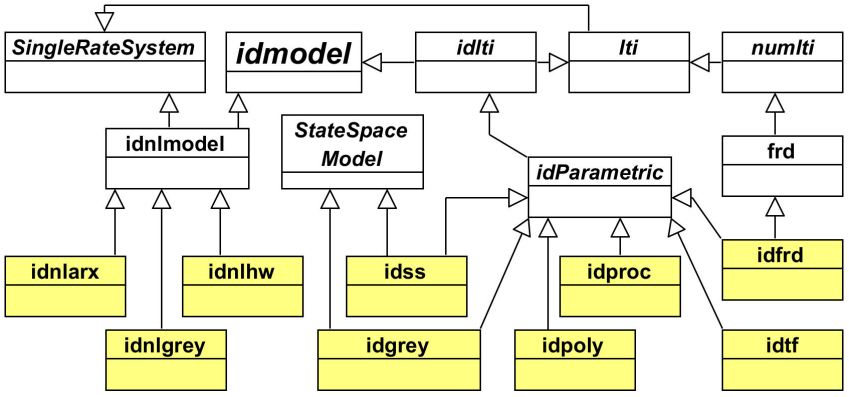
11.4.3. Liniowe i nieliniowe modele identyfikowalne

Nazwy liniowych i nieliniowych modeli identyfikowalnych rozpoczynają się od liter *id*. Ich użycie wymaga zainstalowania *System Identification Toolbox*. Przodkiem wszystkich modeli o identyfikowalnych parametrach jest superklasa `idmodel` (rysunek 11.4.3-1). Ma ona dwóch potomków — `idlti` i `idnlmodel`:

- `idlti` jest superklasą dla modeli liniowych, dziedziczy on dodatkowe metody i atrybuty od `lti`. Potomkiem `idlti` jest klasa abstrakcyjna `idParametric`, a kolejne pokolenie to omówione dalej klasy modeli typu: `idfrd`, `idtf`, `idproc`, `idss`, `idpoly` i `idgrey`. Klasy te uwzględniają wpływ sterowania mierzalnego i zakłóceń na sygnał wyjściowy, zgodnie ze wzorem $y(t) = G\,u(t) + H\,e(t)$.
- `idnlmodel` jest superklasą dla modeli nieliniowych, dziedziczy on dodatkowe metody i atrybuty od `SingleRateSystem`. Jest przodkiem dla identyfikowalnych modeli nieliniowych typu `idnlarx`, `idnlgrey` i `idnlhw`.

11.4.3.1. Identyfikowalne liniowe modele parametryczne

Identyfikowalne (lub zidentyfikowane) modele LTI to modele układów liniowych o uzmiennionych współczynnikach, których wartości można zidentyfikować przez pomiar i analizę sygnału wyjściowego i sygnału wejściowego badanego obiektu.



Rysunek 11.4.3-1. *Identyfikowalne modele liniowe to (kolor szary lub żółty): idgrey, idtf, idss, idfrd, idpoly, idproc. Uproszczony diagram wykonano z użyciem programu Visual Paradigm*

Na rysunku 11.4.3-1 pokazano klasy abstrakcyjne, które nie tworzą obiektów (ikony z białym tłem) oraz ciągłe i dyskretne modele z identyfikowanymi współczynnikami (ikony z żółtym lub szarym tłem). Są to:

- `idfrd` (ang. *identified frequency response data model*) — identyfikowalny model odpowiedzi w dziedzinie częstotliwości;
- `idtf` (ang. *transfer function model with identifiable parameters*) — identyfikowalny model w postaci transmitancji;
- `idproc` (ang. *process model with identifiable parameters*) — identyfikowalny model procesu;
- `idss` (ang. *state-space model with identifiable parameters*) — identyfikowalny model w postaci równań stanu;
- `idpoly` (ang. *polynomial model with identifiable parameters*) — identyfikowalny model w postaci wielomianu;
- `idgrey` (ang. *grey box model*) — ciągły lub dyskretny model w postaci równań stanu z identyfikowalnymi lub estymowanymi współczynnikami.

Przykładowo model ciągły `idss` z wektorem wejściowym u, wektorem wyjściowym y i zakłóceniami e ma postać:

$$\begin{aligned} \frac{dx}{dt} &= Ax(t) + Bu(t) + Ke(t) \\ y(t) &= Cx(t) + Du(t) + e(t) \end{aligned} \qquad (11.4.3.1\text{-}1)$$

11.4.3.2. Identyfikowalne modele nieliniowe

Identyfikowalne nieliniowe modele o uzmiennionych współczynnikach, których wartości można zidentyfikować przez pomiar i analizę sygnału wyjściowego i sygnału wejściowego badanego obiektu, to:

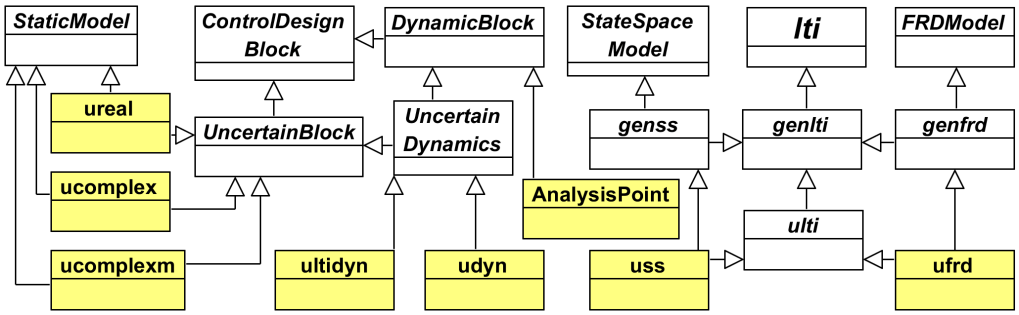
- `idnlarx` (ang. *Auto-Regresive eXogenous)* — identyfikowalny model autoregresyjny z wejściem sterującym;
- `idnlhw` (ang. *Hammerstein-Wiener model*) — identyfikowalny nieliniowy model Hammerstein-Wiener;
- `idnlgrey` — identyfikowalny nieliniowy model szarej skrzynki, modelem jest układ nieliniowych równań różniczkowych I rzędu.

Modele oraz ich przodków pokazano na diagramie klas (rysunek 11.4.3-1).

11.4.4. Modele uogólnione oraz z niepewnymi parametrami

11.4.4.1. Uogólnione modele LTI

Uogólnione (ang. *generalized*) modele LTI (pokazano je na diagramie klas, rysunek 11.4.4.1-1) mają parametry wymagające dostrojenia (ang. *tunable*) lub parametry, których wartości nie są pewne (ang. *uncertain*), ewentualnie mogą mieć zarówno parametry niepewne, jak i wymagające dostrojenia.



Rysunek 11.4.4.1-1. *Uproszczony diagram klas pokazujący klasy modeli uogólnionych oraz statycznych i dynamicznych klas z niepewnym parametrem. Diagram wykonano z użyciem programu Visual Paradigm*

Uogólnione modele LTI powstają przez połączenie modelu należącego do klasy `numlti` oraz specjalnego bloku konstrukcyjnego (ang. *Control Design Block*) zawierającego elementy *dostrajane* (ang. *tuned*). Bloki `genss` i `genfrd` są to uogólnione (ang. *generalized*) modele LTI, przeznaczone dla obiektów, których parametry są *dostrajane* i ewentualnie posiadają *niepewne* współczynniki.

- *genfrd* — jeśli są kombinacją modelu *frd* i obiektu typu *Control Design Block*;
- *genss* — jeśli są kombinacją modelu *lti* (innego niż *frd*) i obiektu typu *Control Design Block*. Utworzenie bloku *genss* pokazano w przykładzie 11.4.4.3-1.

Jeśli uogólnione modele LTI mają tylko parametry niepewne, to będą one nazywane odpowiednio:

- *ufrd* — jeśli są kombinacją modelu *frd* i obiektu typu *Control Design Block* oraz nie mają parametrów dostrajanych;
- *uss* — jeśli są kombinacją modelu *lti* (innego niż *frd*) i obiektu typu *Control Design Block* oraz nie mają parametrów dostrajanych.

Modele z niepewnymi parametrami wymagają zainstalowania *Robust Control Toolbox*.

11.4.4.2. Control Design Block — bloki dynamiczne

Bloki konstrukcyjne *Control Design Blocks* są używane do wprowadzenia do modeli parametrów niepewnych lub wymagające dostrojenia.

Bloki dynamiczne typu *Control Design Block* to

- *ultidyn* — niepewny model dynamiczny *lti*;
- *udyn* — niepewny model dynamiczny o nieokreślonej strukturze;
- *AnalysisPoint* — punkty zainteresowania (pomiaru) dla analizy liniowej lub strojenia systemu sterowania (pokazane na rysunku 11.4.4.1-1).

11.4.4.3. Control Design Block — bloki statyczne

Bloki statyczne typu *Control Design Block*:

- `realp` — parametr skalarny lub macierz;
- `ureal` — niepewny skalar typu Real (używa Robust Control Toolbox);
- `ucomplex`— niepewny skalar typu *complex* (używa Robust Control Toolbox);
- `ucomplex`— niepewna macierz typu *complex* (używa Robust Control Toolbox).

Modele statyczne uogólniają pojęcia macierzy do tablic parametrycznych lub niepewnych. Są one używane

- do tworzenia wyrażeń statycznych parametrycznych lub niepewnych typu *realp*;
- do tworzenia macierzy uogólnionych (obiekty typu *genmat*);
- do tworzenia uogólnionych modeli LTI, których współczynniki są parametryczne lub niepewne;
- do tworzenia niepewnych zmiennych skalarnych i macierzy typu *ureal*, *umat*, *ucomplex*, *ucomplexm* (wymaga oprogramowania *Robust Control Toolbox*).

Przykład 11.4.4.3-1

```
>> a = realp('a',4)        % P = realp(NAME,VALUE), VALUE to wartość początkowa
a =
       Name: 'a'
      Value: 4
    Minimum: -Inf
    Maximum: Inf
       Free: 1

>> F = tf(a,[1 0 a])       % utworzenie obiektu transmitancja z parametrem a

>> whos
  Name      Size            Bytes  Class    Attributes

  F         1x1              1432  genss
  a         1x1                35  realp
```

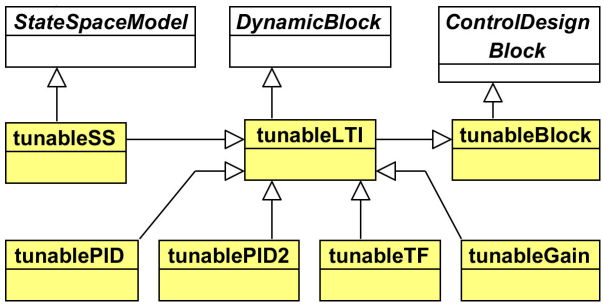
Polecenie `whos` pokazuje, że obiekt typu *tf* z parametrem `a` typu *realp* został przekształcony w obiekt typu *genss*.

Bloki `uss` i `ufrd` są to uogólnione modele LTI posiadające *tylko niepewne* współczynniki.

11.4.4.4. Dostrajane modele dynamiczne

Dostrajane (ang. *tunable*) modele są przeznaczone do projektowania bloków sterujących. Elementy dostrajane (ang. *tunable*) typu *Control Design Block*, np. element `realp`, `tunablePID`, `tunableTF` lub `tunableSS`, umożliwiające dostrajanie parametrów tego systemu (rysunek 11.4.4.4-1):

- modele dostrajane to `tunableGain`, `tunableTF`, `tunableSS`, `tunablePID`, `tunablePID2`;
- `tunableGain` — Tunable gain block (dostrajane wzmocnienie);
- `tunableTF` — SISO fixed-order transfer function with tunable coefficients (transmitancja z dostrajanymi współczynnikami);
- `tunableSS` — Fixed-order state-space model with tunable coefficients (model w przestrzeni stanu z dostrajanymi współczynnikami);
- `tunablePID` — One-degree-of-freedom PID controller with tunable coefficients (regulator PID z dostrajanymi współczynnikami);
- `tunablePID2` — Two-degree-of-freedom PID controller with tunable coefficients (regulator PID2 z dostrajanymi współczynnikami).



Rysunek 11.4.4.4-1. *Modele dynamiczne do projektowania bloków sterujących. Diagram wykonano z użyciem programu Visual Paradigm*

11.4.4.5. Modele z losowymi współczynnikami

- `rss` funkcja generująca ciągły stabilny model typu *ss* (w przestrzeni stanu) z parametrami o wartościach losowych, przydatny do projektowania i testowania systemów automatyki;
- `drss` funkcja generująca dyskretny stabilny model typu *ss*, o nieokreślonym czasie próbkowania `T = -1`.

Przykład 11.4.4.5-1

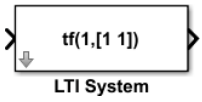
Utworzenie modelu typu *ss* z parametrami o wartościach losowych:

```
>>sys432=rss(4,3,2);   % tworzy ciągły stabilny model typu ss z losowymi parametrami
>>get(sys432)          % podaje własności obiektu sys432 (pokazano fragment wydruku)
                A: [4×4 double]
                B: [4×2 double]
                C: [3×4 double]
                D: [3×2 double]
                E: []
```

11.4.5. Bloki Simulinka

W *Control System Toolbox* (i w innych toolboksach) zdefiniowano biblioteki modeli które są dostępne w Simulinku. Zawartość biblioteki *Control System Toolbox* obejmuje:

- *LTI System* — umieszcza w Simulinku model LTI; model można utworzyć, wpisując odpowiednią funkcję, np. `tf(1,[1 1])`, a jeśli model istnieje, to wystarczy podać jego nazwę (rysunek 11.4.5-1);



Rysunek 11.4.5-1. *Ikona bloku LTI System w Simulinku*

- *LPV System* (ang. *Linear Parameter-Varying*) — umieszcza w Simulinku wybrany model liniowy o zmiennych parametrach;
- *State Estimation* — umieszcza w Simulinku wybrany *filtr Kalmana* lub inny model.

Modele zdefiniowane w *Control System Toolbox* mogą być wstawione do wymienionych wyżej bloków i używane w Simulinku (patrz rozdział 10.).

11.5. Biblioteka Control System Toolbox™

Biblioteka *Control System Toolbox™* udostępnia algorytmy i aplikacje do analizowania, projektowania i dostrajania liniowych (*modele LTI*, ang. *linear time invariant*) i nieliniowych modeli systemów.

Control System Toolbox posiada ulepszone narzędzia, w tym: `controlSystemDesigner` (dawniej SISO Design Tool, do projektowania regulatorów z wykorzystaniem linii pierwiastkowych i wykresów Bodego) oraz `linearSystemAnalyzer` (dawniej LTI Viewer, do analizy własności układów LTI).

11.5.1. Ciągłe i dyskretne modele numlti w ControlSystem Toolbox

Model numlti (nadaje mu się dowolną nazwę, na przykład `sys`) tworzy się z użyciem parametrów typowych dla danego typu modelu:

```
sys = tf(num,den)          % transmitancja: współczynniki licznika i mianownika
sys = zpk(z,p,k)           % wartości zer, biegunów i wsp. wzmocnienia
sys = ss(a,b,c,d)          % macierze przestrzeni stanu
sys = dss(a,b,c,d,e)       % macierze przestrzeni stanu oraz dodatkowa macierze
sys = frd(response,freq)   % wektor odpowiedzi i wektor częstotliwości
```

Funkcja `dss(a,b,c,d,e)` opisuje model w postaci:

$$e\frac{dx}{dt} = ax + bu,\ y = cx + du$$

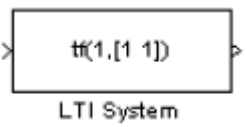
Macierz *e* może być osobliwa.

W celu utworzenia **modelu dyskretnego** należy dodatkowo podać okres próbkowania, na przykład `motor=tf(num,den,Ts)`.

Obiekt `numlti` w postaci transmitancji typu `tf` o nazwie `motor` tworzy się następująco:

```
>> motor=tf(0.74,[14 1])

Transfer function:
  0.74
--------
14 s + 1
```

Modele `numlti` mogą być użyte do symulacji w Simulinku, jeśli zostaną wczytane do bloku LTI dostarczanego wraz z *Control System Toolbox* (rysunek 11.5.1-1).



Rysunek 11.5.1-1. *Blok LTI System, do symulacji w Simulink®*

11.5.1.1. Systemy MIMO

Dla systemu MIMO (ang. *Multi Input Multi Output*) stosuje się te same funkcje (`ss`, `tf` lub `zpk`), jednak parametrami są tablice komórkowe, jak poniżej:

```
>> num={.7 0 ; 2 .4 }
>> den={[1 9] 1 ; [1 8] [1 6]}

num =

    [0.700000000000000]    [                0]
    [                2]    [0.400000000000000]


den =

    [1x2 double]    [         1]
    [1x2 double]    [1x2 double]

>> systemMIMO = tf(num,den)

systemMIMO =

  From input 1 to output…
       0.7
   1:  -----
       s + 9

         2
   2:  -----
       s + 8

  From input 2 to output…
   1:  0

        0.4
   2:  -----
       s + 6

Continuous-time transfer function.
```

11.5.2. Model dyskretny i równanie w dziedzinie czasu

Model dyskretny obiektu pokazanego jako *Proces* na rysunku 11.5.2-1 można zdefiniować z użyciem każdej z opisanych wyżej funkcji `ss`, `tf` lub `zpk`, podając okres próbkowania `Ts` jako dodatkowy parametr `sys = tf(num,den,Ts)`, na przykład:

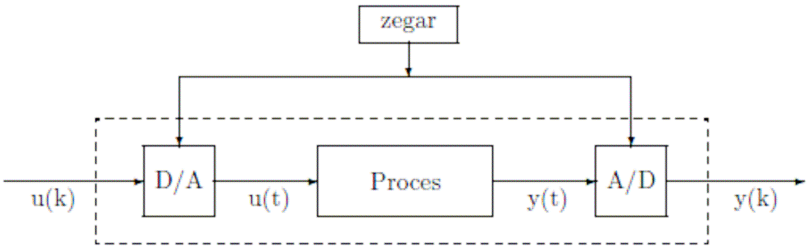
```
Ts=1;
dd=tf([0.08428 -0.01615], [1 -1.795 0.9311],Ts)
```

tworzy transmitancję dyskretną systemu:

$$dd = \frac{0.08428z - 0.01615}{z^2 - 1.795z + 0.9311} \tag{11.5.2-1}$$

```
Sample time: 1 seconds
Discrete-time transfer function.
```

gdzie: zmienna *z* jest operatorem opóźnienia w dziedzinie czasu, $z * f(k) = f(k-1)$.



Rysunek 11.5.2-1. *Fragment dyskretnego systemu sterowania procesem ciągłym*

11.5.2.1. Przekształcanie modelu ciągłego na dyskretny i odwrotnie

Odpowiedź obiektu ciągłego na wymuszenie dyskretne zależy nie tylko od wartości tego sygnału w dyskretnych momentach, ale też od czasu próbkowania i od użytego ekstrapolatora. Funkcja `c2d` przekształca model ciągły na dyskretny:

```
sysd = c2d(sysc,Ts,method)
```

gdzie:

`sysc`, `sysd` — nazwa obiektu ciągłego i obiektu dyskretnego,

`Ts` — czas próbkowania,

`method { 'zoh', 'foh', 'tustin', 'prewarp', 'matched'}` — ekstrapolator.

Jest wiele kryteriów, których można użyć do określenia dyskretnego odpowiednika dla obiektu ciągłego, stąd potrzeba stosowania wielu ekstrapolatorów. Domyślnym i najczęściej stosowanym ekstrapolatorem jest `zoh` (ang. *zero order hold*, znane też jako: *sample and hold*). Odpowiada to ekstrapolacji wielomianem zerowego rzędu (funkcja schodkowa). Otrzymany model jest określany jako *zoh-równoważny*.

Metoda `foh` aproksymuje przebieg odcinkami linii (łamaną). Jej odmianą jest metoda Tustina, często interpretowana jako przybliżenie zależności $z = e^{sT_s}$ poprzez jej rozwinięcie w szereg Taylora. Metoda `prewarp` jest stosowana przy badaniu filtrów dyskretnych. Wymaga ona dodatkowo podania częstotliwości krytycznej jako czwartego parametru. Poniżej podano przykłady dyskretyzacji obiektu ciągłego.

Przykład 11.5.2.1-1

Dyskretyzacja obiektu ciągłego z użyciem ekstrapolatora rzędu zerowego `zoh` dla czasu próbkowania `Ts=1`:

$$mm(s) = \frac{0.74s + 1}{14s^2 + s + 2}$$

```
>> mm=tf([0.74 1],[14 1 2]);    % obiekt ciągły

>> mmd=c2d(mm,1,'zoh')    % dyskretny, zoh-równoważny, krok Ts=1

Transfer function:
          0.08428 z - 0.01615
```

```
        ----------------------
        z^2 - 1.795 z + 0.9311
Sampling time: 1
```

Czytelnik może powtórzyć obliczenia dla innych ekstrapolatorów:

```
>> mmdf=c2d(mm,1,'foh')          % wykorzystuje dwie próbki
>> mmdt=c2d(mm,1,'tustin')       % wykorzystuje dwie próbki
>> mmdm=c2d(mm,1,'matched')   % z=exp(-sT), zachowuje zera i bieguny
```

Przykłady pokazują, że użycie ekstrapolatorów `zoh` i `matched` daje najniższy stopień licznika otrzymanej transmitancji dyskretnej.

Przekształcenia modelu dyskretnego na ciągły dokonuje funkcja `d2c`. Jest ona wywoływana w postaci: `sysc = d2c(sysd,method)`. Otrzymany wynik może wymagać normalizacji.

11.5.2.2. Pola obiektu numlti

Parametry modelu *numlti* są zapamiętane w jego polach. Wykaz nazw pól dla danego obiektu i ich wartości podaje funkcja `get(nazwa_obiektu)`. Model dyskretny `mmd` (otrzymany w podrozdziale 11.5.3 poprzez dyskretyzację modelu ciągłego `mm` metodą `zoh`) ma następujące wartości w swoich polach:

```
>> v=get(mmd)    % pokazano fragment wydruku dla obiektu typu tf
v =
            num: {[0 0.0843 -0.0162]}
            den: {[1 -1.7948 0.9311]}
       Variable:  'z'
             Ts: 1
       TimeUnit: 'seconds'
```

Okres próbkowania oznaczono jako `Ts`, jego wartość to `Ts=1`. Wartość `Ts=0` oznacza obiekt ciągły, a `Ts=-1` oznacza obiekt dyskretny o nieznanym czasie próbkowania. Pole `Variable` występuje tylko w modelach `tf` i `zpk`. Do odczytania wartości pól `z`, `p`, `k` można użyć `get`, jak poniżej:

```
>> v=get(zpk(mmd))
v =
  struct with fields:
                Z: {[0.1917]}
                P: {[2×1 double]}
                K: 0.0843
    DisplayFormat: 'roots'
         Variable: 'z'
```

Wynik jest wpisany do struktury, a elementy `Z` i `P`, czyli zero i biegun (ang. *pole*), są typu `cell`. Odczytanie ich wartości może wymagać użycia funkcji `cell2mat`, która przekształca tablice komórkowe na macierze:

```
>> vp=cell2mat(v.P)
vp =
   0.8974 + 0.3546i
   0.8974 - 0.3546i
```

Obiekt typu `ss`, czyli obiekt należący do przestrzeni stanu, np. `mmssd=ss(mmd)` ma pola A, B, C, D, a czas próbkowania `Ts=1`.

11.5.3. Pobieranie danych z modelu numlti

Dane o modelu już istniejącym można odczytać, podając jego nazwę np. `mmd` jako polecenie:

```
>> mmd
```

Jeśli wartości te mają być wykorzystane w dalszych obliczeniach, lepiej jest wpisać je do odpowiednich macierzy. Dla modelu `mmd` można w tym celu użyć jednego z poleceń:

```
[num,den,Ts] = tfdata(mmd)
[Z,P,K,Ts] = zpkdata(mmd)
[A,B,C,D,Ts] = ssdata(mmd)
[A,B,C,D,E,Ts] = dssdata(mmd)
```

Dla modeli ciągłych parametr `Ts` ma wartość zero. Należy zwrócić uwagę, że typ obiektu `mmd` nie jest istotny i nie musi być znany. W razie potrzeby obiekt będzie przekształcony w typ wymagany przez użytą funkcję. Należy mieć na uwadze, że każde przekształcenie typu może prowadzić do pogorszenia dokładności modelu.

11.5.4. Zmiana nazwy zmiennej w polu Variable

Funkcja `set` pozwala na dowolną zmianę wartości każdego pola obiektu, ale może to niekiedy prowadzić do istotnych efektów ubocznych. W szczególności nazwa zmiennej w polu `'Variable'` modelu typu `tf` nie jest dowolna, lecz musi być wybrana spośród nazw podanych przez `set (tf, 'Variable')`, czyli:

- `s` lub `p` dla obiektów ciągłych;
- `z` lub `q` dla obiektów dyskretnych;
- `z^-1` lub `q^-1` (czyli odwrotność `z` lub `q`) dla obiektów dyskretnych.

Najczęściej stosowane są zmienne `s` lub `z`. Obiekt dyskretny typu `tf`, na przykład:

$$mmd = \frac{0.08428z - 0.01615}{z^2 - 1.795z + 0.9311} \tag{11.5.4-1}$$

można przekształcić w jedną z poniższych postaci:

```
>> set(mmd, 'var','z^-1')    % 'var' to pierwsze litery 'Variable'
>> mmd
Transfer function:
        0.08428 z^-1 - 0.01615 z^-2
        ---------------------------
        1 - 1.795 z^-1 + 0.9311 z^-2
Sampling time: 1
```

czyli uzyskuje się (11.2.6-2):

$$mmd = \frac{0.08428z^{-1} - 0.01615z^{-2}}{1 - 1.795z^{-1} + 0.9311z^{-2}} \tag{11.5.4-2}$$

```
>> set(mmd, 'var','q') % operator opóźnienia  q^-1=z
>> mmd
Transfer function:
        0.08428 q - 0.01615 q^2
        -----------------------
        1 - 1.795 q + 0.9311 q^2
Sampling time: 1
```

czyli otrzyma się (11.2.6-3):

$$mmd = \frac{0.08428q - 0.01615q^2}{1 - 1.795q + 0.9311q^2} \qquad (11.5.4\text{-}3)$$

Należy zwrócić uwagę, że:

- zamiana zmiennych nie zmienia własności obiektu;
- użycie operatora q lub z^{-1} w miejsce z zmienia (zgodnie z oczekiwaniem) wartości potęg w liczniku oraz mianowniku transmitancji. Jest to wynik podzielenia licznika i mianownika transmitancji przez najwyższą potęgę operatora z w mianowniku.

11.5.5. Badanie właściwości modelu

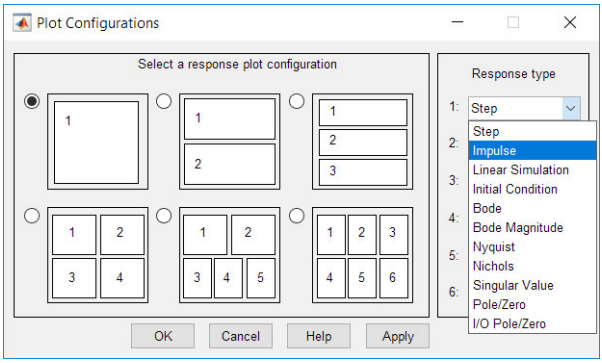
Do badania właściwości modelu używa się opisanego dalej narzędzia *Linear System Analyzer*, posiadającego możliwość jednoczesnego pokazania wielu wykresów: *odpowiedzi skokowej*, impulsowej, wykresów *Bodego* (faza i amplituda), *Nyquista*, *Nicholsa* oraz *zer i biegunów* na płaszczyźnie zespolonej. Markery pozwalają na identyfikację wielkości widocznych na wykresach. Można też wykorzystać funkcje opisane poniżej:

- `pole`, `zero`, `pzmap`, `damp`, `esort` i `dsort` używa się do badania zer i biegunów transmitancji,
- `eig` podaje wartości własne i wektory własne,
- `damp` podaje wartości własne, częstotliwości i tłumienia,
- `step` rysuje odpowiedź w dziedzinie czasu na wymuszenia funkcją $1(t)$,
- `impulse` rysuje odpowiedź w dziedzinie czasu na wymuszenia impulsem $\delta(t)$,
- `gensig` generuje typowe przebiegi sterujące: sinusoidę, ciąg prostokątny lub impulsy,
- `square` i `sawtooth` generują odpowiednio przebiegi: prostokątny lub piłokształtny,
- `initial` generuje odpowiedź systemu o niezerowych warunkach początkowych,
- `lsim` generuje odpowiedź systemu na dowolny zadany przebieg sterujący,
- `bode`, `nyquist`, `nichols` tworzą odpowiednie wykresy,
- `evalfr` podaje wartość odpowiedzi dla zadanej częstotliwości,
- `freqresp` podaje odpowiedzi dla zadanego zbioru częstotliwości.

11.5.5.1. Badanie właściwości modelu z użyciem Linear System Analyzer

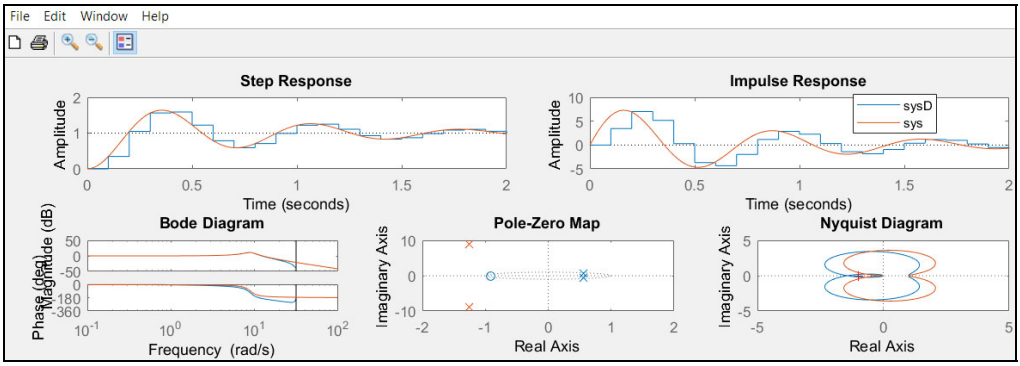
Do badania modeli systemów zaleca się stosowanie interfejsu graficznego *Linear System Analyzer*. Jest to interakcyjne środowisko do analizy systemów LTI. Jego okno otwiera się za pomocą polecenia `linearSystemAnalyzer` lub `ltiview`. Liczbę i zawartość okien interfejsu określa się samodzielnie poprzez wybór menu *Edit/Plot Configurations* (rysunek 11.5.5.1-1).

Rysunek 11.5.5.1-1. *Okno LTI Viewer — wybór wykresów i ich rozmieszczenia*



Linear System Analyzer pokazuje przebiegi czasowe i dokonuje analizy częstotliwościowej systemów. Wykonuje wykresy Bodego, Nyquista i Nicholsa. Dobiera automatycznie odpowiednie częstotliwości i skale na osiach wykresów. Podaje odpowiedź czasową systemu na wymuszenie impulsowe i skokowe.

Na rysunku 11.5.5.1-2 pokazano odpowiedzi systemu `sys = tf(160,[2,5,160])` i odpowiadającego mu systemu dyskretnego `sysD = c2d(sys,0.1)`. Zgodnie z przewidywaniem zgodność odpowiedzi obu systemów jest bardzo dobra przy wymuszeniu jednostkowym, lecz pogarsza się dla wymuszenia impulsowego — z uwagi na duży udział wysokich częstotliwości. Zachęcamy również Czytelnika do samodzielnego zbadania systemu `sysD = c2d(sys,0.1,'prewarp',2)`.



Rysunek 11.5.5.1-2. *LTI Viewer — wykresy dla sys= tf(160,[2,5,160]) i sysD= c2d(sys,.1), step(sysD)*

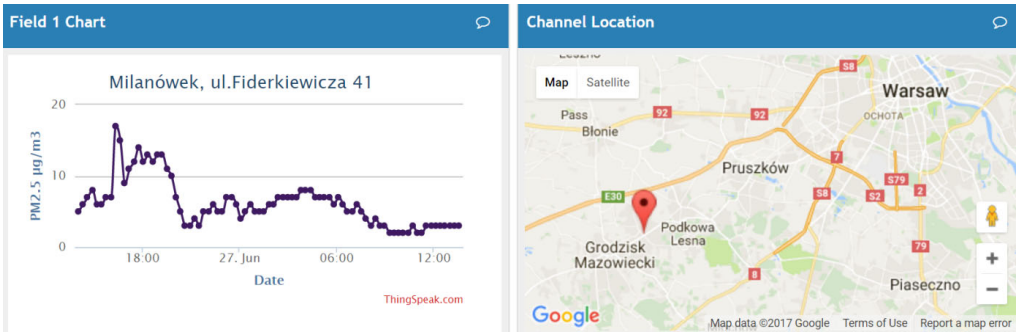
11.6. ThingSpeak™ — obsługa urządzeń IoE w chmurze

ThingSpeak™ to otwarta platforma do archiwizacji, przetwarzania, analizy oraz wizualizacji „na żywo” strumienia danych dostarczonych automatycznie przez czujniki połączone z internetem poprzez systemy mikroprocesorowe lub smartfony. Systemy te są znane jako *IoT* (ang. *Internet of Things*), ale coraz powszechniej używane jest określenie *IoE* (ang. *Internet of Everything*). Nazwa *IoT* nie obejmowała takich zastoso-

wań, jak zdalne monitorowanie pracy serca i innych narządów oraz lokalizacja ludzi i zwierząt.

ThingSpeak udostępnia kanał komunikacyjny i chmurę, w której można archiwizować oraz przetwarzać i wizualizować napływające dane oraz stronę internetową do publikowania wykresów i mapki, pokazującej lokalizację monitorowanego obiektu. *ThingSpeak* umożliwia też wysyłanie i odbieranie alertów poprzez usługi internetowe *Twitter*® lub *Twilio*®. Alert może informować o dowolnym zdarzeniu, np. zaniku lub przekroczenia wartości sygnału pomiarowego.

Wysyłane dane mogą być zadeklarowane jako prywatne. Wówczas dostęp do danych, wykresów i innych informacji mają tylko użytkownicy znający hasło. Jednak wiele kanałów z danymi jest dostępnych publicznie po wybraniu menu *Channels/PublicChannels* na stronie *https://thingspeak.com*. Takich kanałów jest obecnie ponad 15 000, w tym pewna liczba z Polski (rysunek 11.6-1).



Rysunek 11.6-1. *Prezentacja pomiarów zanieczyszczenia powietrza przygotowana przez Społeczne Liceum Ogólnokształcące nr 5 w Milanówku, https://thingspeak.com/channels/224426, tag Milanówek*

Podstawowe usługi *ThingSpeak* są bezpłatne i umożliwiają obsługę niewielkich, niekomercyjnych projektów z monitorowaniem maksymalnie 8 sygnałów, odświeżanych co 15 sekund. Do przetwarzania danych w chmurze można używać dostępnych tam aplikacji i języka MATLAB. Czas przetwarzania kolejnych porcji danych jest ograniczony do 20 sekund.

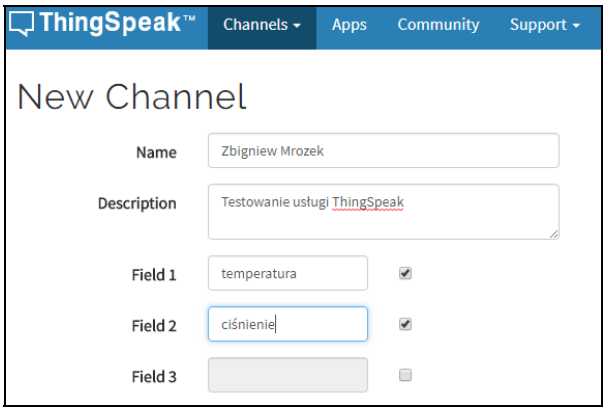
Jeśli bezpłatny dostęp nie jest wystarczający, to można zakupić licencję studencką, domową, akademicką lub standardową. Ceny zaczynają się od 55 lub 75 euro rocznie.

11.6.1. Tworzenie nowego kanału dla danych IoE

Posiadanie kanału *ThingSpeak* umożliwia przetestowanie procesu przetwarzania i wizualizacji danych w chmurze oraz poznanie języka MATLAB. Procedurę tworzenia kanału rozpoczyna się na stronie *https://thingspeak.com/*. Po kliknięciu przycisku *Get Started For Free* można utworzyć nowe konto lub zalogować się swoim hasłem z konta MathWorks lub *ThingSpeak*. Po zalogowaniu się należy wstępnie wypełnić formularz utworzenia kanału, pokazany na rysunku 11.6.1-1. Puste lub źle wypełnione rubryki będzie można później skorygować. Jeśli oprócz pól z nazwami kilku sygnałów (maksymalnie 8) zostaną wpisane współrzędne geograficzne miejsca pomiarów, to prócz wykresów będzie utworzona mapka Google z zaznaczoną lokalizacją miejsca pomiaru, jak na rysunku 11.6-1.

Formularz zatwierdza się przyciskiem *Save Chanell*. Po zatwierdzeniu pojawi się informacja z numerem kanału i hasłami do jego obsługi. Pojawią się też podpowiedzi, jak korzystając z *MQTT* API (ang. *message queueing telemetry transport, application programming interface*) wysyłać i pobierać kolejne dane z kanału oraz jak użyć *REST* API (ang. *representational state transfer*) do zdalnego zarządzania tym kanałem.

Kanał utworzono jako prywatny, ale aktualnie jest on dostępny publicznie pod adresem *https://thingspeak.com/channels/295141*.



Rysunek 11.6.1-1. *Fragment formularza do tworzenia nowego kanału w ThingSpeak*

11.6.2. Przykład przetwarzania danych w chmurze

Poniżej pokazano przykład pobrania danych z kanału dostępnego publicznie w celu ich przetworzenia i wizualizacji we własnym kanale.

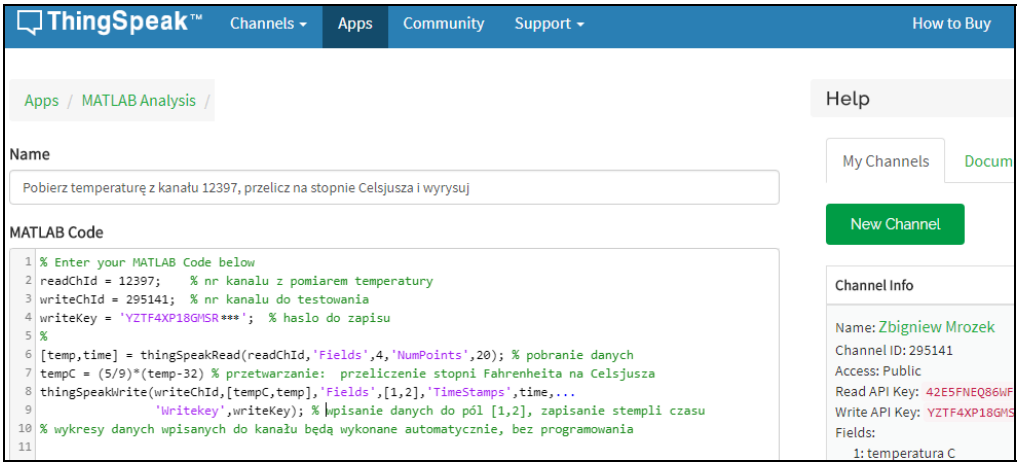
Przykład 11.6.2-1

Pobranie danych (temperatura i stempel czasu) z dostępnego publicznie kanału nr 12397 oraz, po przeliczeniu, wpisanie ich do własnego kanału nr 295141 w celu ich umieszczenia na stronie WWW.

Aby wpisać do chmury pokazany poniżej program w MATLAB-ie, należy wybrać zakładkę *Apps* **widoczną na** rysunku 11.6.1-2 i następnie, przechodząc przez kolejne strony: *MATLAB Analysis*, a potem *New/Templates/Custom(no starter code)*, kliknąć zielony przycisk *Create.*

Na nowej stronie należy wpisać nazwę (lub krótki opis) tworzonego programu oraz jego kod, czyli kolejne linie programu w języku MATLAB, zgodnie z rysunkiem 11.6.2-1. W liniach 3. i 4. należy wpisać numer *własnego kanału* i własny *kod zapisu*, zgodnie z podpowiedziami w *Channel Info*, po prawej stronie ekranu.

Program wykonano dwukrotnie, w odstępie 30 minut, pobierając każdorazowo 20 najnowszych wartości danych. Oba wykresy (temperatura w skali *Celsjusza* i *Fahrenheita)* oraz *mapa Google*, pokazująca miejsce zamieszkania autorów książki, zostały automatycznie umieszczone na stronie *https://thingspeak.com/channels/295141* i są publicznie dostępne. Widoczna na wykresie ciągła, pochyła linia łączy punkty pierwszej i drugiej grupy danych.



Rysunek 11.6.2-1. *Okno do edycji i realizacji programów w języku MATLAB-a*

11.6.3. Przetwarzanie danych pomiarowych w chmurze

Po kliknięciu zakładki *Apps* pojawia się strona z ikonami aplikacji do przetwarzania i wizualizacji danych oraz ikona okna do programowania w języku MATLAB. Są to:

- użycie MATLAB-a oraz aplikacje i wtyczki do wizualizacji danych;
- działania związane ze zdarzeniami, jak upływ czasu, komunikaty, przekroczenie zadanych wartości sygnału, w tym:
  - *ThingTweet* i *TweetControl* — działania związane z wysłaniem lub przyjęciem komunikatu z Twittera;
  - *TimeControl* — działania związane z upływem czasu;
  - *React* — działania związane z przekroczeniem zadanych wartości sygnału, w tym nadmierne oddalenie się od wskazanej lokalizacji GPS;
  - *TalkBack* — warunkowe dokończenie rozpoczętej sekwencji działań;
  - *ThingHTTP* — komunikacja pomiędzy urządzeniami z użyciem Prowl, Twilio i ThingHTTP.

W aplikacjach wykorzystywane są różne języki i protokoły, jak: MATLAB, JavaScript, HTML, CSS, MQTT API, REST API, JSON (ang. *JavaScript Object Notation*), Prowl (dla iOS, iPhone, iPad), Twilio i ThingHTTP.

Dodatkowe funkcje *thingSpeakArea*, *thingSpeakAuthenticate*, *thingSpeakAuthenticatedList*, *thingSpeakPlot*, *thingSpeakPlotYY*, *thingSpeakRead*, *thingSpeakScatter*, *thingSpeakStem*, *thingSpeakWrite*, *urlfilter thingSpeakClearAuthentication*, można pobrać z serwera *https://www.mathworks.com/matlabcentral/* po wpisaniu nazwy *ThingSpeak Support Toolbox* w okienku *Search*.

Na stronie *https://www.mathworks.com/help/thingspeak/examples.html* podano kilkadziesiąt linków do przykładów wykorzystania różnych technologii (Arduino, Raspberry Pi, ESP8266 Wi-Fi Module, Electric Imp) i wielu języków do współpracy z *ThingSpeak*.

Przetwarzanie danych z użyciem MATLAB-a przedstawiono wstępnie w przykładzie 11.6.2-1. Do przetwarzania danych można wykorzystać dostępne w *ThingSpeak* toolboksy, ale warunkiem jest zalogowanie się do *ThingSpeak* hasłem z MathWorks, które jest sprzężone z licencją na potrzebne toolboksy.

11.6.4. Wsparcie dla różnych technologii

Urządzenia oparte na technologiach Arduino®, Raspberry Pi™, BeagleBone Black i innych mogą się łatwo komunikować z *ThingSpeak*™. Warunkiem jest wsparcie dla protokołów TCP/IP, HTTP MQTT lub REST. Wartością dodaną dla użytkowników *ThingSpeak* jest bezpłatny dostęp do okrojonej wersji MATLAB-a poprzez okno pokazane na rysunku 11.6.2-1.

Na stronie *https://www.mathworks.com/hardware-support.html* można wyszukać oprogramowanie wspierające odczyt danych z czujników przyspieszenia (3 osie), pola magnetycznego (3 osie), prędkości kątowej (3 osie), orientacji (ang. *azimuth*, *roll*, *pitch*) oraz różnych danych *GPS* w systemach *Android* oraz *iPhone* i *iPad*.

Artykuły opisujące stacje pogodowe zbudowane i zainstalowane w siedzibie firmy The Mathworks, Inc. można odszukać, wpisując słowa `Arduino`, `Weather`, `Station` w okienku *Search* na stronie *www.mathworks.com*. Wynikiem jest kilkanaście linków. Inne przykłady są na: *https://www.mathworks.com/help/thingspeak/examples.html*.

Bibliografia

1. [Bjorck, Dahlquist, 1987] A. Bjorck, G. Dahlquist, *Metody numeryczne*, PWN, Warszawa 1987.
2. [Krupowicz, 1986] A. Krupowicz, *Metody numeryczne*, PWN, Warszawa 1986.
1. [Mrozek, 1984] B. Mrozek, Z. Mrozek, *Aproksymacja charakterystyk magnesowania silnika indukcyjnego z wykorzystaniem wielomianowej funkcji sklejanej*, „Przegląd Elektrotechniczny" nr 4, Warszawa 1984.
4. [Mrozek, 1996] B. Mrozek, Z. Mrozek, *MATLAB. Uniwersalne środowisko do obliczeń naukowo-technicznych*, wydanie III, PLJ, Warszawa 1996.
5. [Mrozek, 1997] B. Mrozek, Z. Mrozek, *MATLAB environment as CAL tool* [w:] M. Chrzanowski, E. Nawarecki (red.), *Proceedings of 4-th International Conference Computer Aided Engineering Education CAEE'97*, Kraków, 1997.
6. [Mrozek, 1998] B. Mrozek, Z. Mrozek, *MATLAB 5.x Simulink 2.x, poradnik użytkownika*, PLJ, Warszawa 1998.
7. [Mrozek, 2001] B. Mrozek, Z. Mrozek, *MATLAB 6, poradnik użytkownika*, PLJ, Warszawa 2001.
8. [Mrozek, Tarasiewicz, 2001] Z. Mrozek, S. Tarasiewicz, *Attempting sliding mode controller to mobile robot arc welding process*, str. 369-373. [w:] M.R. Tadeusiewicz, A.Ligęza, editor, *III Krajowa Konferencja: Metody i Systemy Komputerowe w Badaniach Inżynierskich*, CCATIE AGH, Kraków 2001.
9. [Mrozek, 2004] B. Mrozek, Z. Mrozek, *MATLAB i Simulink, Poradnik użytkownika*, Wydanie II, Helion, Gliwice 2004.
10. [Mrozek, 2010] B. Mrozek, Z. Mrozek, *MATLAB i Simulink, Poradnik użytkownika*, Wydanie III, Helion, Gliwice 2010.
11. [Mrozek, 2011] Z. Mrozek, *Wprowadzenie do inżynierii oprogramowania i języka UML*, Abaton, Kraków 2011.
12. *comp.soft.sys.matlab* — lista dyskusyjna.
13. *http://www.mathworks.com* — serwer producenta MATLAB-a i Simulinka.
14. *https://www.mathworks.com/matlabcentral/fileexchange* — platforma udostępniająca oprogramowanie środowiska MATLAB i Simulink.

15. *https://www.mathworks.com/hardware-support.html* — strona poświęcona współpracy pakietu MATLAB i Simulink ze sprzętem i oprogramowaniem ARM, Arduino, Altera, National Instruments, Raspberry Pi, Xilinx, Android i innymi.

16. *http://www.ont.com.pl* — serwer firmy ONT, dystrybutora pakietu MATLAB w Polsce.

Skorowidz

A

abs, 32, 125, 198
AbsTol, 97
abstrakcyjne superklasy, 251
accumarray, 118
acos, 31
acosd, 31
acosh, 31
acot, 31
acotd, 31
acoth, 31
acsc, 31
acscd, 31
acsch, 31
Add-On Explorer, 20
  rozszerzenia, 20
addpath, 87
airy, 119
aktualny folder, 82
algorytm, wybór, 170
all, 53
allchild, 156
alternatywa, 51
  wykluczająca, 51
analiza
  funkcji, 191
    maksimum, 191
    miejsca zerowe, 191
    minimum, 191
  obliczanie zer wielomianu, 191
  przebiegu odkształconego, 199
  równania nieliniowego, 192
  statystyczna, 196
  sygnałów, 198
angle, 198
Annotation Object, 154
ans, 55
any, 53
aplikacja
  App, 17, 60, 84
  instalowanie, 167
  wersja końcowa, 162
apostrof, 51
App, 17, 60
App Designer, 17, 153, 157
  interaktywny wykres, 160
  panel, 157
  wizualizacja interaktywna, 163
Apps, 15
aproksymacja, 59, 194
  funkcji schodkowej, 195
  funkcji, 195
  w Basic Fitting, 196
  wielomianowa, 194
  wielomianu, 193
apropos, 82
arcus cosecans, 31
arcus cosecans hiperboliczny, 31
arcus cosinus, 31
arcus cosinus hiperboliczny, 31
arcus cotangens, 31
Arduino®, 268
argumenty wejściowe funkcji, 54
arith, 47
array operation, 47
array2table, 131
arytmetyczne operacje macierzowe, 47
arytmetyczne operacje tablicowe, 47
asec, 31
asecd, 31
asech, 31
asin, 31
asind, 31
asinh, 31
atan, 31
atan2, 31
atan2d, 31
atand, 31
atanh, 31
atrybut, 137, 142, 143, 153, 155
  hidden, 142
  obiektu graficznego, 155
  private, 142
autumn, 204
axes, 105, 106, 138, 139
axis equal, 139
axis, 105, 106
axisHandle, 100, 101, 138, 139

B

backslash, 122
badanie macierzy, 121
balance, 122
Base Workspace, 81
base2dec, 118
Basic Fitting
  aproksymacja, 196
  interpolacja, 196
Basic Linear Algebra Subroutines, 16
bazy ortonormalne, 121
BeagleBone Black, 268
besselh, 119
besseli, 119
besselj, 119
besselk, 119
bessely, 119
beta, 119
betainc, 119
betaincinv, 119
betaln, 119
biblioteka
  bloków, 19, 227
  Control System Toolbox, 258
  FFTW, 16
  graficzna, 16
  matematyczna, 16
  sflib, 229
  toolbox, 19, 248
Big data, 236
bin2dec, 118
binaryzacja obrazu, 208
blacK, 102, 156
blanks, 126
BLAS, 16
blkdiag, 120
blocksets, 19
blok
  całkujący, 216
  do wizualizacji, 216
  events, 141
  genfrd, 255
  genss, 255
  Integrator, 216, 217, 222
  konstrukcyjny, 255
  maskowanie, 225
  methods, 140
  mnożenia, 222
  Mux, 222
  Product, 222
  properties, 140, 141
  Pulse Generator, 216, 219
  realp, 255
  Scope, 216, 217, 219, 222
  State, 230
  Stateflow Chart, 229
  Stateflow, 232
  Subsystem, 223, 225
  Sum, 222
  sumatora, 222
  ufrd, 256
  ureal, 255
  uss, 256
bloki
  biblioteki bloków, 227
  całkujące, 217
  dynamiczne, 255
  Fcn, 222
  łączenie, 217
  Simulinka, 257
  statyczne, 255
błędy, 77
  debuger, 79
  składniowe, 77
  syntaktyczne, 77
  w zapisie, 33
  wykonania, 77, 78
bode, 263
bone, 204
box, 106
break, 72
brighten, 204
bsxfun, 47, 120
bvpget, 185
bvpinit, 185
bvpset, 185

C

calendarDuration, 133, 134
callback, 149, 156, 157, 165
całkowanie, 187
  analityczne, 188
  numeryczne, 187
camlight, 212
cart2pol, 40
Camera Toolbar, 110
CANVAS, zakładka, 164
cart2sph, 40
case, 74, 75, 149
cat, 82, 120, 126, 127
categorical array, 132
cd, 82, 87
cdf2rdf, 122
ceil, 32
cell array, 117, 129
cell, 88, 116, 128, 129, 137
cell2mat, 118, 129

cell2struct, 118, 129
cell2table, 131
celldisp, 129
certyfikacja awioniki DO-178C, 18
cellplot, 129
char, 21, 22, 25, 77, 78, 92, 116, 118, 125, 137, 146
chmura, 239, 250, 264
  dane pomiarowe, 267
  przetwarzanie danych, 266
chol, 122
cholupdate, 122
circshift, 120, 127
class, 117, 137, 139, 141, 142, 145, 146, 150
classdef, 117, 137, 141, 144, 145
clc, 95, 132
clear all, 43
clear, 43, 87
clock, 55
close all, 139
cls, 43
Code Analyzer Report, 92
color, 138, 141, 155, 156
colorbar, 110, 127, 154
ColorLampClass, 144
Command Window, 28, 41
Component Library, 157, 159
concatenate, 120
constructor method, 139
cont, 149
contour, 108
Control System Toolbox, 19, 150
coder HDL Coder, 232
colorbar, 204
colorcube, 204
colordef, 204
colormap, 204
compan, 118, 120
computer, 55
cond, 122, 172
condeig, 121
condest, 122
Contents Report, 92
continue, 72, 73
Control Design Block
  bloki dynamiczne, 255
  bloki statyczne, 255
Control System Toolbox™, 250, 258
  modele LTI, 258
conv, 193, 198
conv2, 198
cool, 203
copper, 204
copyobj, 156
corrcoef, 197
cos, 31
cosd, 31
cosh, 31
cot, 31
cotd, 31
coth, 31
cov, 197, 198
Coverage Report, 92
cputime, 55
cross, 47, 119, 193
csc, 31
cscd, 31
csch, 31
ctranspose, 47
CUDA, 248
cumprod, 197
cumsum, 197
czas obliczeń, 178
czyszczenie
  ekranu, 43
  przestrzeni roboczej, 76

D

damp, 263
dane
  lokalne, 238
  pomiarowe, 267
  typy fundamentalne, 115
DatabaseDatastore, 241
data
  aktualna, 55
  mixed-type tabular, 116
datastore, 135, 236, 238
datatypes, 130
date, 55
datestr, 55
datetime, 115, 133
dblquad, 188
dbstack, 87
deal, 129
deblank, 126
debuger, 17, 79, 221
  stan wstrzymania obliczeń, 80
  śledzenie linii programu, 81
  wstawianie pułapki, 80
  zmiana przestrzeni roboczej, 81
  zmienne lokalne, 81
  znak zachęty K>>, 80
dec2base, 118
dec2bin, 118
dec2hex, 118
decimate, 198
deconv, 193, 198
definiowanie podklasy, 143
dekompozycja
  LU, 189
  macierzy, 189
  QR, 190
del, 82
delete, 82, 156
Dependency Report, 92
det, 121
deval, 185
diag, 120
diagram
  stanu, 229, 230
  UML, 138
diffuse, 212
dir, 82
disp, 73, 75, 77, 79, 125
dll, 64, 84
DO Qualification Kit, 18
doc, 82
dokumentacja, 61, 89
DoPlots, 165
dos, 82
dot, 47, 119, 193
double, 115, 125, 137
drawnow, 156
drss, 252
dsort, 263
dss, 252
duration, 133
dwukropek, 44, 75, 77, 120
dziedziczenie klas, 140, 143
dzielenie
  lewostronne, 49, 169
  macierzowe, 49
  macierzy, 47
  prawostronne, 49, 170
  tablic, 49
  tablicowe, 49
  wektorów, 47, 198

E

easter eggs, 98
echo on, 72
echo, 78
Editor, 60, 70, 93, 159
edytor
  Live Editor, 60
  Notepad++, 60
efekt
  blur, 210
  Rungego, 196
eig, 121, 263
eigs, 121
element
  maksymalny, 197
  minimalny, 197
elfun, 31
ellipj, 119
ellipke, 119
elmat, 47, 55, 87, 118
elseif, 74
end, 46, 72, 120
eps, 33, 55, 107, 108, 114, 120
erf, 119
erfc, 119
erfcx, 119
erfinv, 119
error, 73, 77
esort, 263
etime, 55
eval, 72, 87, 94, 95, 126
evalc, 87, 94
evalfr, 263
evalin, 72, 87, 94
event, 159
exifread, 201
exist, 53, 87
exp, 31, 33, 36, 37, 39
expint, 119
expm, 31, 121
expm1, 31
eye, 79, 88, 115, 118, 119
ezcontour, 101
ezcontourf, 101
ezmesc, 101
ezmeshc, 101
ezplot, 37, 101
ezplot3, 101
ezpolarezcontour, 101
ezsurf, 101
ezsurfc, 101

F

factor, 33
factorial, 33
fcontour, 101
feval, 72, 87, 94, 96
fft, 198, 199
fft2, 198
fftn, 198
fftshift, 198
FFTW, Fastest Fourier Transform in the West, 16
fgoalattain, 248
fieldnames, 130
Figure, 153
filt, 252
filtry, 210, 211
  dolnoprzepustowe, 210
  górnoprzepustowe, 211
find, 53, 82, 120, 124
findobj, 156, 213
findstr, 126
fix, 32
flag, 204
flintmax, 33, 120
flintmin, 120
flipdim, 120
fliplr, 120
flipud, 120
floor, 32
fmesc, 101
fminbnd, 191
fmincon, 248
fminimax, 248
fminsearch, 191
fminunc, 248
foh, 260
for, 72, 75, 87
for-end, 72
format
  ASCII, 42
  DOC, 90
  HTML, 90
  long e, 34
  long g, 34
  long, 34
  PDF, 90
  PPT, 90
  rat, 35, 50
  short, 34
  wektorowy, 114
  XML, 90
  graficzny, 114
FPGA, 17
fplot, 36, 69, 101, 191
fplot3, 101
frd, 252
freqresp, 263
freqspace, 44, 118
fsolve, 248
fsurf, 101
fullfile, 240
function, 60, 64, 72
function handle, 64, 103, 117, 175

funkcja, 71, 91, 131
abs, 125, 198
accumarray, 118
acos, 31
acosd, 31
acosh, 31
acot, 31
acotd, 31
acoth, 31
acsc, 31
acscd, 31
acsch, 31
airy, 119
all, 53
allchild, 156
angle, 198
any, 53
arctg, 33, 34
arcus cosecans
hiperboliczny, 31
arcus cosecans, 31
arcus cosinus
hiperboliczny, 31
arcus cosinus, 31
arcus cotangens, 31
array2table, 131
asec, 31
asecd, 31
asech, 31
asin, 31
asind, 31
asinh, 31
asv, 63, 84
atan, 31, 33, 34
atan2, 31
atan2d, 31
atand, 31
atanh, 31
autumn, 204
axes, 105, 106
axis, 105, 106
backslash, 122
balance, 122
base2dec, 118
besselh, 119
besselj, 119
besselk, 119
bessely, 119
beta, 119
betainc, 119
betaincinv, 119
betaln, 119
bin2dec, 118
blanks, 126
blkdiag, 120
bode, 263
bone, 204
box, 106
brighten, 204
bsxfun, 47, 120
bvpget, 185
bvpinit, 185
bvpset, 185
c2d, 260
camlight, 212
cart2pol, 40
cart2sph, 40
cat, 120, 126, 127
cdf2rdf, 122
ceil, 32
cell, 128, 129
cell2mat, 118, 129
cell2struct, 118, 129
cell2table, 131
celldisp, 129
cellplot, 129
char, 118
chgFreqUnit, 252
chol, 122
cholupdate, 122
circshift, 120, 127
class, 117, 145, 146
classdef, 117, 137, 141, 145
clock, 55
colorbar, 204
colorcube, 204
colordef, 204
colormap, 204
compan, 118, 120
cond, 122
condeig, 121
condest, 122
conv, 193, 198
conv2, 198
cool, 203
copper, 204
copyobj, 156
corrcoef, 197
cos, 31
cosd, 31
cosh, 31
cot, 31
cotd, 31
coth, 31
cov, 197, 198
cputime, 55
cross, 47, 119, 193
csc, 31
cscd, 31
csch, 31
cumprod, 197
cumsum, 197
damp, 263
datatypes, 130
date, 55
datestr, 55
datetime, 115
dblquad, 188
deal, 129
deblank, 126
dec2base, 118
dec2bin, 118
dec2hex, 118
decimate, 198
deconv, 193, 198
delete, 156
det, 121
deval, 185
diag, 120
diffuse, 212
DoPlots, 165
dot, 47, 119, 193
double, 125
drawnow, 156
drss, 252
dsort, 263
dss, 252
eig, 121, 263
eigs, 121
ellipj, 119
ellipke, 119
elmat, 87, 118
end, 120
eps, 120
erf, 119
erfc, 119
erfcx, 119
erfinv, 119
esort, 263
etime, 55
eval, 94, 95, 126
evalc, 94
evalfr, 263
evalin, 94
exifread, 201
exist, 53
exp, 31
expint, 119
expm, 121
expm1, 31
eye, 115, 118, 119
ezcontour, 101
ezcontourf, 101
ezmesc, 101
ezmeshc, 101
ezplot, 37, 101
ezplot3, 101
ezpolarezcontour, 101
ezsurf, 101
ezsurfc, 101
factor, 33
factorial, 33
fcontour, 101
feval, 94, 96
fft, 198, 199
fft2, 198
fftn, 198
fftshift, 198
fieldnames, 130
filt, 252
find, 53, 120, 124
findobj, 156, 213
findstr, 126
fix, 32
flag, 204
flintmax, 120
flintmin, 120
flipdim, 120
fliplr, 120
flipud, 120
floor, 32
fmesc, 101
fminbnd, 191
fminsearch, 191
foh, 260
fplot, 36, 101, 191
fplot3, 101
freqresp, 263
freqspace, 44, 118
fsurf, 101
fullfile, 240
funm, 121
fzero, 191
gallery, 118, 120
gamma, 119
gammainc, 119
gammaincinv, 119
gammaln, 119
gcbf, 156
gcbo, 156
gcd, 33
gcf, 114
gensig, 263
get, 106, 156, 261
getfield, 130
graph2d, 99
graph3d, 99
gray, 204
grid, 105
gsvd, 121
gtext, 105
hadamard, 118, 120
hankel, 118, 120
hess, 121
hex2dec, 118
hex2num, 118
hilb, 118, 120
hist, 197
histc, 197
hold, 106
hot, 203
hsv, 203
hsv2rgb, 204
hypot, 31
i, 120
ifft, 198
ifftn, 198
ifft2, 198
image, 201, 204
imagesc, 201, 204
imfinfo, 201
impulse, 263
imread, 201, 202
imwrite, 201, 202
ind2sub, 120
inf, 120
inferiorto, 146
initial, 263
int, 188
int2scr, 125
int2str, 118, 125
integral, 187
interp, 198
intmax, 120
intmin, 120
inv, 122
invhilb, 118, 120
ipermute, 120, 127
isa, 146
iscell, 53, 129
ischar, 53
iscolumn, 120
isempty, 53, 121
isequal, 121
isequaln, 121
isfinite, 33, 53, 120
isglobal, 53
ishold, 53, 106
isinf, 33, 53
isletter, 53
ismatrix, 120
ismissing, 131
isnan, 33, 53, 120
isobject, 53, 146
isprime, 33
isreal, 53

funkcja
isrow, 120
isscalar, 120
issparse, 53
isstruct, 53
isvector, 120
j, 120
jet, 203
kron, 47
lcm, 33
ldl, 122
legend, 105
legendre, 119
length, 121, 125
lighting, 212
lines, 204
linsolve, 122
linspace, 44, 118
log, 31
log10, 31
log1p, 31
log2, 31
loglog, 101
logm, 121
logspace, 44, 118
lower, 126
lscov, 122
lsim, 263
ltiview, 263
lu, 122
magic, 118, 120
mat2cell, 118
mat2str, 118, 126
matched, 260
material, 212
matfun, 120
max, 197
max, 197
mean, 197
median, 197
meshgrid, 118, 127
methods, 145
min, 197
mldivide, 122, 169, 170
mod, 32
mrdivide, 170
nan, 120
native2unicode, 118
nchoosek, 33
ndgrid, 120, 127
ndims, 121, 126
nested_ode, 165
nextpow2, 31
nichols, 263
norm, 121
normest, 121
normest1, 122
now, 55
nthroot, 31
null, 121
num2cell, 118, 129
num2hex, 118
num2str, 118, 125
numel, 121
nyquist, 263
ode113, 174
ode15i, 174
ode23, 174
ode45, 174
odefun, 180
ones, 118, 119
ordeig, 121
ordqz, 121
ordschur, 121
orth, 121
parula, 203
pascal, 118, 120
peaks, 29, 107, 118, 120
perms, 33
permute, 120, 127
pi, 120
pink, 204
pinv, 122
planerot, 122
plot, 38, 99, 100
plotyy, 103, 104
pol2cart, 40
pole, 263
poly, 121, 193
polyder, 193
polyeig, 121, 193
polyfit, 193
polyval, 193, 194
polyvalm, 193
pow2, 31
prewarp, 260
primary function, 65
primes, 33
prism, 204
prod, 197
psi, 119
pzmap, 263
qr, 122
qrdelete, 122
qrinsert, 122
qrupdate, 122
quad, 188
quad2d, 188
quadgk, 188
quadl, 188
quadv, 188
qz, 121
rand, 118, 197
randn, 118, 197
rank, 121
rat, 33
rats, 33
rcond, 122
readtable, 131
reallog, 31
realmax, 33, 120
realmin, 33, 120
realpow, 31
realsqrt, 31
rectangle, 139
regexp, 52
rem, 32
repmat, 118
reset, 156
reshape, 120
residue, 193
return , 72, 149
rgb, 102
rgb2all, 147
rgb2hsv, 204
rgbplot, 204
ribbon, 107
roots, 191, 193
rosser, 118, 120
rot90, 120
round, 32
rowfun, 131
rref, 121
rsf2csf, 122
rss, 252
sawtooth, 263
schur, 121
sec, 31
secd, 31
sech, 31
semilogx, 101
semilogy, 101
set, 156, 262, 261
shading, 204
shiftdim, 120, 127
sign, 32
sin, 31
sind, 31
sinh, 31
size, 121, 125
slash, 122
sort, 197
sortrows, 131
sparfun, 122
sparfun, 87
sparse, 115, 116, 122, 124
spconvert, 122
specgraph, 99
specular, 212
sph2cart, 40
spinmap, 204
spline, 195
sprand, 122
spring, 204
sprintf, 125
spy, 123
sqrt, 31
sqrtm, 120, 121
square, 263
squeeze, 120, 127
sscanf, 126
stack, 131
standardizeMissing, 131
std, 197
step, 263
str2double, 118
str2mat, 126
str2num, 118, 126
strcat, 126
strcmp, 53, 126
strcmpi, 52
string , 41, 73, 124
strncmpi, 52
strrep, 126
struct, 130, 131
struct2cell, 118, 129
sub2ind, 120
subplot, 106
subspace, 121
sum, 197
summary, 131
summer, 204
superiorto, 146
surf, 212
surfl, 212
svd, 121
svds, 121
table, 131
table2array, 131
table2cell, 131
table2struct, 131
tan, 31
tand, 31
tanh, 31
text, 105
tic, 55, 178
title, 105
toc, 55, 178
toeplitz, 118, 120
trace, 121
tril, 120
triu, 120
tustin, 260
uint8, 125
unicode2native, 118
unstack, 131
upper, 126
vander, 118, 120
var, 197
varfun, 131
wilkinson, 118, 120
winter, 204
write, 131
writetable, 131
xlabel, 105
ylabel, 105
yyaxis, 103
zero, 263
zeros, 118, 119
zoh, 260
zoom, 106
funkcje, 159
analiza, 191
analizy macierzowej, 121
anonimowe, 36, 37, 69
biblioteczne, 72
czasu, 55
dla liczb zespolonych, 32
do badania macierzy, 121
do konstruowania macierzy, 118
do opisywania wykresów, 105
działające na łańcuchach, 126
generujące macierze, 88
główne, 65, 66, 68
graficzne, 36, 101
jednoznaczność wyboru, 150
konwersji łańcuchów, 125
konwersji, 117
lokalne, 53, 61, 65, 66, 71
macierzowe, 121
niepotrzebne parametry, 65, 96
obsługi klas, 145
obsługi obiektów, 145
odeXX, 175
priorytety, 71
prywatne, 68, 71, 150
przeciążanie, 149
przetwarzania sygnałów, 198
rozkład macierzy, 122
równania liniowe, 122
sklejane, 194

specjalne, 33
statystyczne, 197
trygonometryczne, 31
uchwyt, 65, 103
wartości osobliwe, 121
wartości własne, 121
wbudowane, 71, 150
wektorowe, 193
wielokrotne
wykorzystanie, 91
wielomianowe, 193
wykładnicze, 31, 32
zagnieżdżone, 66, 67, 68, 71, 164
zespolone, 32
zmienna liczba parametrów, 96
Fuzzy Logic Toolbox, 84

G

gallery, 118, 120
gamma, 119
gammainc, 119
gammaincinv, 119
gammaln, 119
gca, 101, 155, 156
gcbf, 156
gcbo, 156
gcd, 33
gcf, 114, 149, 155
gco, 155, 156
generowanie wektorów, 44, 55, 88
genfrd, 255
gensig, 263
genss, 255
get, 106, 142, 156
GetAccess, 140
GETedge, 142, 143
getfield, 130
global, 66, 72
GPU, graphics processing unit, 249
graficzne formaty, 114
graficzny interfejs użytkownika, GUI, 100
grafika 24-bitowa, 202
grafika 3-D, 29
grafika uchwytów, 153
App Designer, 157
PlotTools, 108
graph2d, 99
graph3d, 99
gray, 144, 148, 204
grid, 105
groot, 153
gsvd, 121
gtext, 105
GUI, 100, 153

H

hadamard, 118, 120
handle, 141, 153
Handle Graphics, 153
hankel, 118, 120
height, 131, 132
help, 24, 30, 56, 57, 82, 91
Help Report, 92
helpbrowser, 82
hermetyzacja, 142
hess, 121
hex2dec, 118
hex2num, 118
hidden, 142
hierarchia
inferiorto, 150
klas abstrakcyjnych, 251
obiektów graficznych, 153, 154
superiorto, 150
HiL, 17, 232
hilb, 118, 120
hist, 197
histc, 197
histogram, 197
hold, 106
horzcat, 131
hot, 203
hsv, 203
hsv2rgb, 204
hypot, 31

I

i, 33, 55, 120
identyfikowalne modele liniowe, 253
identyfikowalne modele nieliniowe, 254
idfrd, 254
idgrey, 254
idnlarx, 254
idnlgrey, 254
idnlhw, 254
idpoly, 254
idproc, 254
idss, 254
IEC Certification Kit, 18
if-elseif-else-end, 72
ifft, 198
ifftn, 198
iftt2, 198
ikona Get More Apps, 168
iloczyn
elementów, 197
kumulatywny, 197
skalarny, 193
skalarny wektorów, 47
tensorowy Kroneckera, 47
wektorowy, 47, 193
wielomianów, 193
iloraz wielomianów, 193
Image Processing Toolbox, 201, 206, 209
image, 201, 204
ImageDatastore, 239
imagesc, 201, 204
imfinfo, 201
impulse, 263
imread, 201, 202
imwrite, 201, 202
ind2sub, 120
indeksowanie
jawne, 77
macierzy, 55
niejawne, 75
struktur, 54
tablic, 54
tablic komórkowych, 54
inf, 33, 120
Inf, 55
inferiorto, 146, 150
initial, 263
innerjoin, 131
input, 72, 75
inputname, 72, 87
instalowanie
aplikacji, 167
rozszerzeń, 168
instrukcja
break, 72
continue, 72, 73
error, 73, 77
for, 75, 76
input, 75
instrukcja keyboard, 73, 77
iteracyjna, 75, 76
iteracyjne, 75
parfor, 75
pause, 72
przerywające, 72
return, 72
sterująca while-end, 76
sterujące, 72
try-catch-end, 78
warunkowa if, 73
warunkowe, 72, 73
while, 75
while-end, 75
wyboru switch, 74
wyboru, 72
int, 188
int2scr, 125
int2str, 118, 125
integer, 115
integral, 187
interaktywna mapa środowiska, 19
interfejs Basic Fitting, 196
interp, 198
interpolacja, 194
funkcji, 195
w Basic Fitting, 196
wielomianowa, 194
intersect, 131
intmax, 33, 120
intmin, 33, 120
inv, 122
invhilb, 118, 120
IoE (ang. Internet of Everything)., 264
IoT (ang. Internet of Things), 264
ipermute, 120, 127
isa, 146
iscell, 53, 129
ischar, 53
iscolumn, 120
isempty, 53, 121
isequal, 121
isequaln, 121
isfinite, 33, 53, 120
isglobal, 53
ishold, 53, 106
isinf, 33, 53
isletter, 53
ismatrix, 120
ismember, 131
ismissing, 131
isnan, 33, 53, 120
isobject, 53, 146
isprime, 33
isreal, 53
isrow, 120
isscalar, 120
issparse, 53
isstruct, 53
istable, 131
isvector, 120

J

j, 33, 55, 120
jaja wielkanocne, 98
Java, 151
javaArray, 151
javaMethod, 151
javaMethodEDT, 151
javaObject, 151
javaObjectEDT, 151
jawne indeksowanie, 77
jednokrokowa metoda Rungego-Kutty, 174
jet, 203
język Java, 151
JIT, just-in-time, 86
join, 131

K

karta z układem FPGA, 17
keyboard, 73, 77
klasa, 137
Figure, 153
frd, 252
handle, 141
idfrd, 254
idgrey, 254
idpoly, 254
idproc, 254
idss, 254
idtf, 254
lti, 150
numlti, 252, 255
ss, 252
tf, 252
zpk, 252
klastry, 250
klasy
definiowanie, 137
definiowanie podklasy, 143
dziedziczenie, 140, 143
języka Java, 151
nadrzędna, 143
podklasa, 143
subclass, 143
superklasa, 143
tworzenie, 137
klawiatura, 93
kodowanie wywołań zwrotnych, 165
kolor linii, 102

komentarze, 91
komórka, cell, 128
koniec wiersza macierzy, 55
koniunkcja, 51
konkatenacja, 94
konstruktor, 138, 144
  klasy, 71
kontynuacja polecenia, 55
konwersja
  liczby, 125
  łańcuchów, 125
  układu współrzędnych, 40
kopia bezpieczeństwa, 84
kotwiczenie edytora, 63
kowariancja, 197, 198
kron, 47
kropka dziesiętna, 54

L

lang, 73
LAPACK, 16
LATEX, 90
lcm, 33
ldl, 122
legend, 105
legenda, 154
legendre, 119
length, 121, 125
licencja
  akademicka, 21
  dla szkół, 21
  MATLAB Home, 22
  standardowa, 21
  studencka, 21
  TAH, 21
  TSH, 21
liczba
  Inf, 55
  NaN, 55
  nargin, nargout, 55, 96
  pi, 55
  plików, 96
  realmax, 55
  realmin, 55
liczby
  całkowite, 115
  całkowite bez znaku, 115
  specjalne, 33
  zespolone, 171
  zmiennoprzecinkowe, 115
lighting, 212
Linear Algebra Package, 16
Linear System Analyzer, 263
lines, 204
linia
  ciągła, 102
  kreskowa, 102
  kreskowo-kropkowa, 102
  kropkowa, 102
linie przesyłania sygnałów, 217
linsolve, 122
linspace, 44, 118
Live Editor, 60, 61, 89
live script, 61, 66, 84
load, 28, 95, 118
log, 31
log10, 31
log1p, 31
log2, 31
logarytrn
  funkcji Beta, 119
  naturalny, 31, 34
logical, 116, 117
loglog, 101
logm, 121
logspace, 44, 118
lokalizacja błędów, 77
lokalne pliki graficzne, 239
lookfor, 82, 91
lower, 126
ls, 82
lscov, 122
lsim, 263
lsqcurvefit, 248
lsqnonlin, 248
lti, 150
lu, 122
LXE, language execution engine, 17

Ł

łańcuch, 41, 55, 116, 124

M

macierz, 41, 45, 115
  bsxfun, 47
  cross, 47
  dekompozycja, 189
  diagonalna, 120, 189
  dodawanie, 47
  dzielenie lewostronne, 47
  dzielenie macierzowe, 49
  dzielenie prawostronne, 47
  dzielenie tablicowe, 47, 49
  elementarna, 118
  faktoryzacja, 122
  full, 123
  Hamiltona, 170
  Hessenberga, 170
  iloczyn skalarny, 47
  iloczyn tensorowy Kroneckera, 47
  iloczyn wektorowy, 47
  jednostkowa, 118
  kwadratowa, 170
  komórkowa, 118
  losowa, 118
  łączenie, 54, 118
  masowa, 184
  mnożenie, 47
  odejmowanie, 47
  odwrotna, 122
  ortogonalna, 189
  pełna, 116, 117
  potęgowanie, 47, 50
  potęgowanie tablicowe, 47
  pseudoodwrotna, 173
  pseudoodwrotność, 47
  pusta, 91, 115
  rzadka, 116, 122, 124
  sklejanie, 120
  sparse, 123
  specjalizowana, 120
  specjalna, 88, 91, 119
  sprand, 123
  sprzężenie, 47, 50
  stochastyczna, 91
  stowarzyszona wielomianu, 118
  transponowanie, 47, 50, 51
  trójkątna, 120, 170
  trójkątna dolna, 189
  trójkątna górna, 189
  trójkątna spermutowana, 170
  tworzenie, 118
  unitarna, 189
  wartości własne, 47
  wybór wiersza, kolumny, 45
  zespolona, 50, 91
magic, 118, 120
manipulowanie elementami macierzy, 120
mapa środowiska, 19
MapReduce, 237, 246
mapy, 107
Mask Editor, 225
maska, 210
maskowanie podsystemów, 225
mat2cell, 118
mat2str, 118, 126
material, 212
materiały szkoleniowe, 25, 29
matfun, 47, 120
MathWorks, 15
MathWorks Account, 25
MATLAB, 15
MATLAB Compiler, 19
MATLAB Distributed Computing Server, 19, 23
MATLAB Drive, 235, 239
MATLAB executable, 64, 84, 85
MATLAB Home, 22
MATLAB Mobile, 22, 239
MATLAB Online, 22, 235, 239
MATLAB Report Generator, 89
MATLAB w chmurze, 22
matlabpath, 82
MAT-plik, 42, 84, 240, 241
matrix operation, 47
max, 197
md, 82
mean, 197
median, 197
meshgrid, 118, 127
method, Abstract, Access, 141
methods, 145, 151
methods lti, 150
metoda
  Adamsa-Bashfortha-Moultona (PECE), 174
  foh, 260
  get, 142
  jednokrokowa Rosenbrocka, 174
  prewarp, 260
  set, 142
  trapezowa, 174
  wielokrokowa NDFs, 174
metody numeryczne, 169
MEX-pliki, 84, 150
mfilename, 72
MIMO, Multi Input Multi Output, 259
min, 197
mkdir, 82
mlappinstall, 84
mldivide, 47, 122, 169, 170
mlock, 72
mlpkginstall, 84
mniejszy lub równy, 51
mniejszy od, 51
mnożenie
  macierzy, 47
  tablicowe, 47
  wektorów, 47, 198
mod, 32
model
  based design, 17, 18
  ciągły, przekształcanie, 260
  definiowany graficznie, 223
  dyskretny, 258, 259, 260
  identyfikowalny liniowy, 253
  identyfikowalny nieliniowy, 253
  LTI, 254
  numlti, 262
  obiektów, 250
  parametryczny, 253
  proteced, 84
  przestrzeni stanu, 150
  Simulinka, 84
  ss, 252
  tf, 252
  właściwości, 263
  z losowymi współczynnikami, 257
  z niepewnymi parametrami, 254
  zpk, 252
modelowanie, 215
modelowanie fizyczne, 228
moduły rozszerzające, 232
modyfikowanie
  parametrów solvera odeXX, 175
  rysunków, 104
  wykresów, 105
more, 82
MQTT API, 266
mrdivide, 170
munlock, 72
mysz, 93

N

NaN, 33, 48, 55, 120
nargin, 55, 72, 96
nargout, 55, 72, 96
narzędzia
  App Designer, 157
  do symulacji, 19
  do testowania, 19
  do weryfikacji, 19
  GUIDE, 153
  interaktywne Plot Tools, 110
native2unicode, 118

nawias kwadratowy, 41
nazwy
  pliku, 96
  formalne, 66
  specjalne, 54
nchoosek, 33
ndgrid, 120, 127
ndims, 121, 126, 131
negacja, 51
negatyw, 207
nested_ode, 165
nextpow2, 31
nichols, 263
niejawna indeksacja, 75, 87
nierówny (różny od), 51
norm, 121
norma macierzy, 121
norma wektora, 121
normest, 121
normest1, 122
notacja dwukropkowa, 41, 45
notacja kropkowa, 48, 87, 130
now, 55
nthroot, 31
null, 121
num2cell, 118, 129
num2hex, 118
num2str, 118, 125
numel, 121, 131
numlti, 255
nyquist, 263
nzmax, 123, 124

O

obiekt, 71, 88, 139
  aktualny, 155
  DatabaseDatastore, 241
  datastore, 236, 238
  należący do podklasy, 145
  numlti, 261
  tworzenie, 139, 140
  usuwanie, 156
  zmiana wartości atrybutów, 141
obiekty graficzne, 153
  atrybuty, 155
  hierarchia, 153
  scenariusz i aranżacja, 164
  zmiana atrybutów, 156
obliczanie
  pochodnych, 181
  wartości wyrażeń, 33
  zer wielomianu, 191
obliczenia równoległe, 190, 246, 248
obraz
  binaryzacja, 208
  dylatacja obrazu, 209
  erozja obrazu, 209
  filtr dolnoprzepustowy, 210
  filtr górnoprzepustowy, 211
  filtracja, 210
  funkcje, 201
  imfinfo, 206
  indeksowany, 203, 205, 206
  nakładanie na powierzchnię, 213
  negatyw, 207
  nieindeksowany, 206
  odczyt, 201
  palety barw, 203
  przekodowanie, 205, 206
  przyciemnianie, 208
  pseudokolor, 205
  rastrowy, 206
  rozjaśnianie, 207, 208
  ściemnianie, 207
  testy, 206
  w skali szarości, 205
  zapis, 201
obsługa
  błędów, 78
  folderów, 82
  plików, 82
  tabel, 131
  zdarzeń, 141
odbicia, 212
odchylenie standardowe, 197
ode113, 174
ode15i, 174
ode23, 174
ode45, 174
odefun, 180
odeXX, 175
okienko edycyjne, obsługa, 165
okno
  Basic Fitting, 196
  Command History, 28
  Command Window, 28, 34, 59, 70
  Command, 43
  Current Folder, 28, 83
  Edit Configurations, 90
  Editor, 28, 60
  Figure Palette, 112
  Figure, 109, 111
  Help, 30
  History, 63
  Live Editor, 28
  Plot Browser, 112
  Property Editor, 113
  Set Path, 83
  Workspace, 28
ones, 88, 118, 119
OOP, object oriented programming, 137
opcja autosave, 63
operacje
  arytmetyczne, 210
  logiczne, 208
  macierzowe, 47, 48, 120
  matematyczne, 16
  morfologiczne, 209
  na macierzach, 30
  na obiektach, 156
  na tablicach, 30
  na wektorach, 30
  tablicowe, 47, 48, 77
  wbudowane, 39
operatory, 75
  alternatywa, 51
  alternatywa wykluczająca, 51
  apostrof, 51
  arith, 47
  arytmetyczne, 47
  ctranspose, 47
  dzielenia, 47
  generowania wektorów, 44
  jednoznaczność wyboru, 150
  koniunkcja, 51
  kron, 47
  kropka, 51
  logiczne, 47, 51
  lub równy, 51
  mldivide, 47
  mniejszy od, 51
  mnożenia, 47
  negacja, 51
  nierówny (różny od), 51
  notacja dwukropkowa, 118
  operator opóźnienia z, 260
  potęgowania, 50
  priorytety, 54
  przeciążanie, 149
  relacji, 51
  równy, 51
  slash, 47
  sprzężenia macierzy, 55
  transpozycji, 55
  większy lub równy, 51
  większy od, 51
  xor, 51
Optimization Toolbox, 19
optymalizacja programu, 97
ordeig, 121
ordqz, 121
ordschur, 121
orth, 121
oscyloskop, 216
osie logarytmiczne, 101
osie pionowe, 103
oświetlenie, 213
outerjoin, 131
overloading, 149

P

pakiety, 19
paleta
  barw, 203
  parula, 203
  white, 204
parametr beta, 165
parfor, 75
parpool, 248
parula, 203
pascal, 118, 120
path, 57, 82
pause, 72
pcode, 84
peaks, 107, 110, 118, 120
perms, 33
permute, 120, 127
persistent, 66, 72
pętla for, 72
pętla while, 72
pętle, 72
pi, 33, 55
pierwiastki wielomianu, 193
pink, 204
pinv, 122
planerot, 122
PLC, programmable logic controller, 17
pliki, 84
  ASCII, 42, 84
  demo, 98
  funkcyjne, 64, 84
  funkcyjny, 71
  graficzne, 201
  instalacyjne, 84
  MAT-plik, 42, 85
  MEX-plik, 85
  polecenia, 82
  Simulinka, 84
  skryptowe, 60, 71, 84, 89
  tekstowe, 59
plot, 38, 99, 100
Plot Tools, 110
PLOTS, 16
plottols, 110
plotyy, 103
pochodna, 181
  obliczanie, 181
  wielomianu, 193
początek komentarza, 55
podklasa, 143
podpowiedzi, 70
podsystem, 223, 224
  maskowanie, 225
  tworzenie, 224
podział okna, 104
  subplot, 104, 107
pol2cart, 40
pola, 117
pole, 263
polecenia
  obsługi pliku, 82
  systemu operacyjnego, 82
polecenie
  addpath, 87
  apropos, 82
  axis equal, 139
  cat, 82
  cd, 82, 87
  clc, 29
  clear all, 43
  clear, 43, 87
  close all, 139
  cls, 43
  dbstack, 87
  del, 30, 82
  delete, 82
  demo, 29
  diary off, 30
  diary on, 30
  dir, 30, 82
  doc, 30, 56, 82
  dos, 82
  echo on, 72
  end, 72
  eval, 72, 87
  evalc, 87
  evalin, 72, 87
  exist, 87
  exit, 27
  eye, 88
  feval, 72, 87
  find, 82
  function, 72
  global, 66, 72
  height, 131
  help, 30, 57, 82, 91
  helpbrowser, 82

polecenie
horzcat, 131
innerjoin, 131
input, 72
inputname, 72, 87
intersect, 131
ismember, 131
istable, 131
javaArray, 151
javaMethod, 151
javaMethodEDT, 151
javaObject, 151
javaObjectEDT, 151
join, 131
lookfor, 57, 82, 91
ls, 30, 82
matlabpath, 82
md, 82
methods lti, 150
methods, 151
mfilename, 72
mkdir, 82
mlock, 72
more, 82
munlock, 72
nargin, 72
nargout, 72
ndims, 131
numel, 131
ones, 88
outerjoin, 131
parpool, 248
path, 57, 82
peaks, 29, 110
persistent, 72
plottols, 110
print, 114
publish, 90
pwd, 83
quit, 27
rm, 82
rmpath, 87
save, 42
saveas, 114
set, 82
setdiff, 131
setenv, 82
setxor, 131
simulink, 215
size, 131
subplot, 104
tail, 245
type, 82
union, 131
unique, 131
unix, 82
varargin, 72
varargout, 72
ver, 29
vertcat, 131
what, 91
where, 91
which, 82, 87
who, 40, 146
whos, 40, 78, 87, 131, 145, 146, 244
width, 131
winopen, 82
zeros, 88
poly, 121, 193
polyder, 193
polyeig, 121, 193
polyfit, 193
polyval, 193, 194
polyvalm, 193
poprawianie wydajności, 86
pow2, 31
P-pliki, 150
prealokacja, 86
prewarp, 260
primes, 33
print, 114
priorytety operatorów, 54
prism, 204
private, 142
prod, 197
profiler, 97
program tworzący grafikę, 113
programowa obsługa błędów, 78
programowanie, 59
programowanie zorientowane obiektowo, OOP, 88, 115, 137
properties, 141
próbki
rozrzedzanie, 198
zagęszczanie, 198
przeciążanie, 143, 149, 150
funkcji, 149
operatorów, 149
przedefiniowanie, 149
przekodowanie obrazu, 205, 206
przenoszenie rysunków, 114
przetwarzanie obrazów, 201
przetwarzanie punktowe, 207
przyciemnianie obrazu rastrowego, 208
przypisanie wartości, 54
pseudokolor, 205
pseudoodwrotność, 173
psi, 119
pulpit, desktop, 28
Pulse Generator, 216
punkty
diament, 102
gwiazda sześciokątna, 102
gwiazdka, 102
gwiazdka pięciokątna, 102
kwadrat, 102
okrąg, 102
plus, 102
punkt, 102
trójkąt, 102
trójkąt lewy, 102
trójkąt prawy, 102
pwd, 83
pzmap, 263

Q

qr, 122
qrdelete, 122
qrinsert, 122
qrupdate, 122
quad, 188
quad2d, 188
quadgk, 188
quadl, 188
quadv, 188
qz, 121

R

rand, 118, 197
randn, 118, 197
rank, 121
raport
Code Analyzer Report, 92
Contents Report, 92
Coverage Report, 92
Dependency Report, 92
Help Report, 92
PUBLISH, 90
TODO/FIXME Report, 92
Raspberry Pi™, 268
rat, 33
rats, 33
rcond, 122
readtable, 131
reallog, 31
realmax, 33, 55, 120
realmin, 33, 55, 120
realp, 255
realpow, 31
realsqrt, 31
regexp, 52
relacje, 51
relop, 51
rem, 32
repmat, 118
reset, 156
reshape, 120
residue, 193
residuum, 172, 193
return, 72
reusability, 91
rezerwowanie pamięci, 87, 119
rgb2hsv, 204
rgbplot, 204
ribbon, 107
rm, 82
rmpath, 87
robot, 85
Robust Control Toolbox™, 250
Root, 153
roots, 191, 193
rosser, 118, 120
rot90, 120
round, 32
rowfun, 131
rozjaśnienie obrazu rastrowego, 207
rozkład Cholesky'ego, 189, 190
rozkład LU, 189
rozpoczęcie pracy, 27
rozszerzenia, instalowanie, 168
rozszerzenia Simulinka, 19
rozszerzenie
.ai, 114
.asv, 63, 84
.bmp, 114, 202
.csv, 238
.cur, 202
.dat, 84, 85
.dll, 64, 84
.emf, 114
.eps, 108, 114
.fig, 108, 114
.gif, 202
.hdf4, 202
.ico, 202
.jar, 151
.jarext, 151
.jpeg, 202
.jpeg2000, 202
.jpg, 108, 114
.m, 30, 59, 65, 84, 114
.mat, 84, 242
.mdl, 84, 218
.mdlp, 84, 218
.mex, 64
.mexa64, 84
.mlapp, 84
.mlappinstall, 168
.mlx, 30, 59, 61, 62, 84
.p, 84
.pbm, 202
.pcx, 114, 202
.pgm, 202
.png, 108, 114, 202
.ppm, 202
.ras, 202
.req, 84
.slreqx, 84
.slx, 84, 218
.slxp, 84, 218
.tif, 114
.tiff, 108, 202
.xwd, 202
rozwiązanie
analityczne, 183
sprawdzenie poprawności, 172
układu równań, 39
równania
algebry liniowej, 169
cząstkowe, 184
liniowe, 169
rozwiązanie, 171
układy, 169
źle uwarunkowane, 171, 192
nadokreślone, 173
niedookreślone, 173
nieliniowe, 192
różniczkowe, 163, 221
AbsTol, 178
cząstkowe, 186
III rzędu, 180
MaxStep, 178
model, 183
ode23, 164
ode45, 179
odeset, 177, 178
przekształcenie, 180
RelTol, 178
solwer odeXX, 174
stats, 179
weryfikowanie wyników, 177
z macierzą, 184

zwyczajne
z opóźnieniem, 186
zwyczajne, 173
w dziedzinie czasu, 259
równy, 51
różniczkowanie, 187
analityczne, 188
numeryczne, 188
rref, 121
rsf2csf, 122
rss, 252

S

save, 42
saveas, 114
sawtooth, 263
schur, 121
search path, 63
sec, 31
secd, 31
sech, 31
sekcje, 88
sekcje programu, 88
sekwencje poleceń, 61
semilogx, 101
semilogy, 101
separator
argumentów funkcji, 55
indeksów, 55
pola, 54
poleceń, 55
set, 82, 142, 156, 262
setdiff, 131
setenv, 82
setxor, 131
shading, 204
shiftdim, 120, 127
sieci lokalne, 250
sieci rozproszone, 250
sign, 32
Signal Processing Toolbox., 198
silnik LXE, 17
SimRF, 19
Simscape, 17, 19, 228
Simscape Driveline, 19, 228
Simscape Electronics, 19, 228
Simscape Fluids, 19, 228
Simscape Multibody, 19, 229
Simscape Power Systems, 19, 229
Simulink, 16, 18, 215
pakiety, 19
rozszerzenia, 19
Simulink 3D Animation, 84
Simulink Editor, 216
Simulink Report Generator, 89
Simulink Start Page, 215
sin, 31
sind, 31
single, 115
sinh, 31
size, 121, 125, 131
skala barw, 154
skróty, shortcuts, 82, 93
skrypty, 61, 91
uruchamianie, 63
slash, 47, 122
słowo kluczowe
end, 46
function, 60, 64
inferiorto, 150
superiorto, 150
solver
dyskretny, 219
fgoalattain, 248
fmincon, 248
fminimax, 248
fminunc, 248
fsolve, 248
lsqcurvefit, 248
lsqnonlin, 248
ode23, 179
odeXX, 174, 175, 182
stałokrokowy, 219
użycie, 177
wywołanie, 182
zmiennokrokowy, 219
sort, 197
sortrows, 131
sparfun, 87, 122
sparse, 115, 116, 122, 124, 137
spconvert, 122
specfun, 47
specgraph, 99
specular, 212
sph2cart, 40
spinmap, 204
spirala trójwymiarowa, 106
spline, 195
spline function, 194
splot, 198
sprand, 122
spring, 204
sprintf, 125
sprzężenie, 50
macierzy, 47
spy, 123
sqrt, 31
sqrtm, 121
square, 263
squeeze, 120, 127
ss, 252
sscanf, 126
stack, 131
stałe specjalne, 55
standardizeMissing, 131
state charts, 229
Stateflow, 229
Statistics and Machine Learning Toolbox, 19
std, 197
step, 263
Stepper, 221
sterowanie poślizgowe, 231
sterownik PLC, 17
str2double, 118
str2mat, 126
str2num, 118, 126
strcat, 126
strcmp, 52, 53, 126
strcmpi, 52
string, 41
strncmpi, 52
strrep, 126
struct, 117, 130, 131, 137
struct2cell, 118, 129
struct2table, 131
struktura struct, 130
struktury, 116, 130
sub2ind, 120
subclass, 143
subfunkcja, 150
subplot, 104, 106
subspace, 121
sum, 197
suma elementów, 197
suma kumulatywna, 197
summary, 131
summer, 204
superiorto, 146, 150
superklasa, 143
surf, 212
surfl, 212
suwak, 160
obsługa, 165
położenie, 160
svd, 121
svds, 121
switch-case-otherwise, 74
switch-case-otherwise-end, 72
Symbolic Math Toolbox, 183, 187, 188
symulacja, 215, 218
tryby przyspieszone, 220
uruchomienie, 218
wizualizacja wyników, 219
System Identification Toolbox™, 250
system pomocy, 30, 56
system sterowany zdarzeniami, 229
systemy MIMO, 259
szybka transformata Fouriera, 16

Ś

ścieżka dojścia, 96
ścieżki dostępu, 82
średnik, 29
środowisko programistyczne, 61
światło, 212
źródła, 212

T

tabela timetable, 134
tabele, 131
tworzenie, 131
table, 116, 131
table2array, 131
table2cell, 131
table2struct, 131
tablic komórkowych, 129
tablica, 38, 41, 115
dwuwymiarowa, 115
komórkowa, 116, 128
potęgowanie, 50
wielowymiarowa, 116, 126
wyliczeniowa, 132
znakowa, 124
tail, 245
tall array, 135
tall table, 135, 236, 242, 243
tan, 31
tand, 31
tanh, 31
tekstury, 212, 213
testowanie
bench, 30
funkcji, 120
grafiki, 35
sekcji programu, 88
text, 105
tf, 252
ThingSpeak™, 264
tic, 55, 178
timefun, 55
timetable, 134
title, 105
toc, 55, 178
TODO/FIXME Report, 92
toeplitz, 118, 120
toolboksy, 19, 84
toolbox
Control System Toolbox, 258
Image Processing Toolbox, 209
Signal Processing, 198
Symbolic Math Toolbox, 188
trace, 121
transformata Fouriera FFT, 198
transponowanie, 50
macierzy, 47, 51
tril, 120
triu, 120
true color, 202
try-catch-end, 72, 78
tunableGain, 256
tunablePID, 256
tunablePID2, 256
tunableSS, 256
tunableTF, 256
tworzenie
aplikacji z GUI, 157
dokumentacji, 89
funkcji anonimowej, 37
funkcji, 64
macierzy rzadkich, 122
macierzy, 41, 88, 118
tablic, 41, 54
tall table, 243
wektorów, 41
typ
calendarDuration, 133, 134
cell array, 117
cell, 116, 137
char, 116, 117, 137
datetime, 133
double, 115, 137
duration, 133
function_handle, 117
integer, 115
logical, 116, 117
numeryczny, 115
single, 115
sparse, 137

typ
struct, 116, 117, 137
table, 116
tall table, 242
uint8, 137
wyliczeniowy categorical, 131
type, 82
typy danych, 115, 117
fundamentalne, 115
nienumeryczne, 117
numeryczne, 116
struktura, 117
uchwyt funkcji, 117

U

uchwyt, 69, 82, 153, 167, 175
funkcji, 36, 65, 103, 116
Handle Graphics, 154
użycie, 167
ucomplex, 255
ufrd, 256
uint8, 18, 125, 137
układ równań, 180
algebraicznych, 39
liniowych, 169
uwarunkowanych, 178
układ współrzędnych, 40
UML, Unified Modeling Language, 137, 138
unicode2native, 118
union, 131
unique, 131
unix, 82
unsigned integer 8-bit, 18
unstack, 131
uogólnione modele LTI, 254
uporządkowanie rosnące, 197
upper, 126
ureal, 255
uruchamianie programu, 79
uruchomienie skryptu, 63
uss, 256
usuwanie zmiennych, 43
użycie GPU, 248

V

vander, 118, 120
var, 197
varargin, 72, 96
varargout, 72, 96
varfun, 131
vertcat, 131

W

wariancja, 197
wartości
osobliwe, 121
własne, 121, 191
własne wielomianu, 193
wyświetlanie, 161
wartość
średnia, 197
środkowa, 197
wielomianu, 193
wyrażenia, 74
zmiennej, 77
wczytywanie zmiennych, 42
wektory, 38, 41, 44
amplituda, 198
dzielenie, 198
faza, 198
skracanie, 198
sprzężenie, 50
transponowanie, 50
własne, 191
wydłużanie, 198
wektoryzacja, 86
obliczeń, 87
weryfikowanie programu
M-Lint, 92
what, 91
where, 91
which, 82, 87
while, 72, 75, 87
while-end, 72
who, 40, 146
whos, 40, 78, 87, 131, 145, 146, 244
why, 98
width, 131
wielomian, 193
aproksymacja, 193
charakterystyczny, 193
iloczyn, 193
iloraz, 193
interpolacyjny Lagrange'a, 194
pierwiastki, 193
pochodna, 193
wartości własne, 193
wartość, 193
większy lub równy, 51
większy od, 51
wilkinson, 118, 120
winopen, 82
winter, 204
witryna producenta, 24
wizualizacja
danych, 35
meshgrid, 127
rozwiązania, 182
wyników symulacji, 219
write, 131
writetable, 131
wskaźniki, 128
współczynnik korelacji, 197
współrzędne
biegunowe, 40
kartezjańskie, 40
sferyczne, 40
wydajność, 86
wykres, 99
edycja osi, 112
edycja linii, 112
edycja tekstu, 112
fazowy, 163
fplot, 36, 69, 101
fplot3, 36, 37
function handle, 103
grid, 106
gtext, 106
Inspector, 112
edycja interaktywna, 109
kolor, 102
legend, opis, 106
loglog, 101
M-plik, 113
Plot Browser, 112
plot, 38, 100, 105
PlotTools, 108
plotyy, 103
Property Editor, 112
semilogx, 101
semilogy, 101
text, 106
title, 106
TypLinii wykresu, 100
użycie myszki, 109
wykonanie, 165
xlabel, ylabel, 106
znakowanie linii, 102
wykres trójwymiarowy, 3D, 37, 106
lighting, 213
material, 213
mesh, 107
meshgrid, 107
plot3, 106
ribbon, 107
tekstura, 213
wykres interaktywny, 160
okno graficzne, 161
przycisk STOP, 162
suwak, 160
wyświetlenie wartości, 161
wykrywanie błędów, 77
wyrażenia, 33
logiczne, 51
matematyczne, 31
wywołania zwrotne, 159
kodowanie, 165
wywołanie pliku funkcyjnego, 65
wyznacznik macierzy, 121

X

xlabel, 105
xor, 51

Y

ylabel, 105
yyaxis, 103

Z

zadania wielodomenowe, 186
zagadnienie brzegowe, 185
zagadnienie początkowe, 173
zakładka
APPS, 168
PLOTS, 35, 99
PUBLISH, 90
zakończenie pracy, 27
zaokrąglanie, 32
ceil, 32
fix, 32
floor, 32
round, 32
zapisywanie
do MAT-pliku, 42
do pliku, 114
przebiegu sesji, 30
rysunków do pliku, 114
zmiennych, 42
zdarzenie, 159
zero, 263
zeros, 88, 118, 119
zmienna, 31, 40, 71, 159
ans, 31, 55
globalne, 66
liczba parametrów, 96
logiczne, 116
lokalne, 66, 77, 81
nargin, 96
operator, 75
persistent, 66
specjalne, 55, 66
sterującą wyborem, 74
zmiana nazwy, 262
zrozumiała nazwa, 92
znaki, 54, 116
@, 103
specjalne, 54, 66
tyldy, 65
zachęty, 29
zoom, 106
zpk, 252

Ź

źródła światła, 212