1.1 Metody numeryczne w środowisku MATLAB

Wstęp

Pierwszy moduł kursu poświęcony zostanie podstawom programowania metod numerycznych oraz symulacji w środowisku MATLAB. Zintegrowane środowisko obliczeniowe MATLAB posiada ugruntowana pozycję w środowisku naukowców oraz inżynierów projektantów. Główną zaletą tego środowiska jest stosunkowo prosty język programowania (język skryptowy) nastawiony na obliczenia numeryczne i symulacje ze szczególnym nastawieniem na algebrę, stąd nazwa pakietu: - MATLAB od (MATrix LABoratory). Drugą niewątpliwą zaletą jest bardzo rozbudowane i dopracowane środowisko programistyczne z wbudowanym debugerem. Obecnie środowisko posiada bardzo rozbudowany zestaw bibliotek dedykowanych do konkretnych zastosowań inżynierskich oraz naukowych (ang. Toolbox). W ramach niniejszego przedmiotu będziemy poznawać zaawansowane metody numeryczne oraz ich implementację w MATLABie. Z uwagi na to nie będziemy posługiwać się wbudowanymi bibliotekami lecz samodzielnie starać się implementować wybrane algorytmy. Należy jednak zaznaczyć, że podejście to ma jedynie cel dydaktyczny, jakim jest zrozumienie, samodzielna implementacja oraz weryfikacja algorytmów numerycznych. W praktycznych zastosowaniach zdecydowanie zachęcamy do wykorzystywania gotowych funkcji wbudowanych w środowisko.

Wprowadzenie do MATLABa

Pierwsza część zostanie poświęcona wprowadzeniu w środowisko MATLAB. Środowisko w trybie graficznym składa się z następujących głównych elementów: okno komend, przeglądarka plików, przeglądarka zmiennych, okno edytora. W MATLABie możemy pracować interaktywnie wpisując bezpośrednio kolejne komendy i instrukcje w oknie komend lub wsadowo edytując plik tekstowy (skrypt) stanowiący zbiór komend. Skrypty mogą być zorganizowane w funkcje, czyli bloki kodu, które mogą być uruchomione poprzez przekazanie zestawu argumentów oraz, które zwracają określony wynik. Komendy MATLABa operują na wartościach stałych (literały liczbowe, napisy, itp.) oraz na zmiennych. W MATLABie deklaracja typu zmiennej jest dynamiczna i następuje w momencie przypisania wartości do jakiejś nazwy. Np.:

```
A = 5.12
B = [1, 2, 3.0]
C = ‘Przykładowy napis’
```

W linii 1. następuje przypisanie zmiennej A wartości liczby rzeczywistej 5.12, w linii 2. zmienna B staje się wektorem trzy elementowym, a w linii 3. zmiennej C zostaje przypisany ciąg znaków o wartości ‘Przykładowy napis’. Przejdźmy przez podstawowe, najczęściej wykorzystywane działania:

```
>> t = 0.0:0.25:2.0
t =
  Columns 1 through 7
         0    0.2500    0.5000    0.7500    1.0000    1.2500    1.5000
  Columns 8 through 9
    1.7500    2.0000
```

W powyższej linijce kodu zmiennej t przypisany zostanie wektor dziewięcioelementowy, z wartościami kolejno 0.0, 0.25, 0.5, 0.75, …, 1.75, 2.0. Instrukcja ta oznacza: wygeneruj wektor liczb równo-odległych zaczynając od 0.0 z krokiem 0.1 i nie większych niż 2.0.

Innym sposobem inicjalizacji wektorów z wartościami równo-odległymi jest wykorzystanie funkcji linspace:

```
t =
  Columns 1 through 7
    1.0000    1.4803    1.9606    2.4409    2.9212    3.4014    3.8817
  Columns 8 through 12
    4.3620    4.8423    5.3226    5.8029    6.2832
```

Powyższa instrukcja spowoduje utworzenie wektora z 12 elementami, rozpoczynającymi się od wartości 1, a kończącymi na wartości 2π, z krokiem $2\pi/11$. Wynika to z tego, że przy dwunastu punktach będzie 11 równoodległych odcinków.

Kolejnym typowym zadaniem podczas programowania algorytmów numerycznych jest ‘indeksowanie’ elementów macierz oraz wektorów. W MATLABie indeksowanie odbywa się poprzez podanie numeru elementu w nawiasach okrągłych, i tak np.:

```
>> t(6)
ans =
    1.2500
```

wyświetli szósty element wektora t. Należy przy zwrócić szczególną uwagę na to, że w MATLABie indeksowanie rozpoczyna się od wartości 1. Czyli pierwszy element wektora ma numer 1. W odróżnieniu, w praktycznie wszystkich innych językach programowania indeksowanie rozpoczyna się od wartości 0.

Operacje indeksowania w MATLABie posiadają szereg udogodnień. I tak na przykład do ostatniego elementu macierzy możemy odwołać się poprzez konstrukcję wykorzystującą słowo kluczowe end:

```
>> t(end)

ans =

     2
```

Możemy również efektywnie wycinać fragmenty wektorów i macierzy podając zakresy indeksów z separatorem w postaci znaku dwukropka np.:

```
>> t(3:6)

ans =

    0.5000    0.7500    1.0000    1.2500
```

spowoduje zwrócenie nowego wektora, który będzie zawierał 4 elementy z wektora t o indeksach kolejno 3, 4, 5, 6. Inną wygodną konstrukcją jest wycięcie fragmentu wektora z pominięciem ostatniego elementu, np.:

```
>> t(1:end-1)

ans =

  Columns 1 through 7

         0    0.2500    0.5000    0.7500    1.0000    1.2500    1.5000

  Column 8

    1.7500
```

Możemy również utworzyć nowy wektor na podstawie zbioru indeksów podanych jako elementy wektora indeksującego, np.:

```
>> t([1 4 5])

ans =

         0    0.7500    1.0000
```

Spowoduje wycięcie wartości elementów o indeksach 1, 4, 5 i zwrócenie ich w postaci nowego trójelementowego wektora.

Bardziej zaawansowane operacje indeksowania uwzględniają operacje logiczne na wartościach wektorów. Możemy na przykład zwrócić w nowym wektorze wszystkie elementy, które są większe od 0.5 i mniejsze od 1.5:

```
>> t( t > 0.5 & t < 1.5 )

ans =

    0.7500    1.0000    1.2500
```

gdzie w nawiasach okrągłych wpisaliśmy dwa warunki połączone operatorem logicznym ‘i’ zapisanym w postaci &.

Operatory indeksujące możemy również wykorzystywać w połączeniu z operatorem przypisania:

```
>> t(t == 1.0) = 100

t =

  Columns 1 through 7

         0    0.2500    0.5000    0.7500  100.0000    1.2500    1.5000

  Columns 8 through 9

    1.7500    2.0000
```

Powyższa komenda spowoduje zastąpienie w wektorze t wszystkich elementów o wartości 1.0 wartością 100.

Inicjalizacja macierzy typowo odbywa się poprzez wykorzystanie funkcji zeros(), ones(), eye(), itp. Funkcje te przyjmują albo jeden, albo dwa argumenty oznaczające odpowiednio liczbę wierszy macierzy oraz liczbę kolumn. W przypadku podania jednego argumentu tworzona jest macierz kwadratowa. Funkcja zeros() tworzy macierz zainicjalizowaną wartościami 0, ones() macierz wypełniona jedynkami oraz eye() macierz jednostkową. Często chcemy zainicjalizować macierze jakimiś wartościami. Wtedy wystarczy wykorzystać funkcję ones() i wymnożyć wynik przez wartość skalarną, np.:

```
>> eye(3) * 1.3

ans =

    1.3000         0         0

         0    1.3000         0

         0         0    1.3000
```

Programy w MATLABie mogą być dzielone na bloki kodu zamknięte w funkcjach, instrukcjach warunkowych oraz pętlach. W przypadku instrukcji warunkowych mamy do dyspozycji klasyczne wyrażenie:

```
>> if (t(1) == 0.25)

  disp('wykona się gdy prawda')

else

  disp('wykona się gdy fałsz')

end
```

W powyższym przykładzie sprawdzana jest wartość pierwszego elementu wektora `t`. Jeżeli wartość będzie równa `0.25` wówczas wykona się pierwszy blok programu aż do napotkania instrukcji `else`, w przeciwnym razie wykona się drugi blok programu zawarty między `else` oraz `end`. Użycie `else` jest opcjonalne, natomiast słowo kluczowe end jest obowiązkowe i oznacza koniec instrukcji warunkowej `if`.

W przypadku pętli występują ich dwa zasadnicze rodzaje: pętle w których przypadku znamy liczbę powtórzeń przed jej uruchomieniem, oraz takie dla których nie jesteśmy w stanie przed uruchomieniem jednoznacznie określić liczby ‘iteracji’. W pierwszym przypadku stosuje się instrukcję `for`, a w drugim instrukcję `while`.

```
>> for i=1:4
        disp(i)
 end
     1
     2
     3
     4
```

Zasada działania instrukcji for polega na wskazaniu zmiennej, która będzie ‘iteratorem’ oraz zbioru wartości dla których wewnętrzny blok pętli będzie kolejno wykonany. W powyższym przykładzie iteratorem jest zmienna ‘`i`’, która kolejno będzie przyjmować wartości z wektora [1, 2, 3, 4]. Wewnętrzny blok wykorzystuje funkcję `disp`(), która w MATLABie służy do wyświetlania komunikatów tekstowych.

Drugim przykładem pętli są pętle `while`():

```
>> i = 1;
>> while (i<=4)
        disp(i)
        i = i + 1;
 end
     1
     2
     3
     4
```

W przypadku pętli `while` musimy przed jej wykonaniem zainicjalizować własny iterator i wartością 1, oraz zadbać wewnątrz jej o zwiększanie jego wartości na końcu wewnętrznego bloku pętli. Pętla `while` będzie się wykonywać dopóki jest spełniony warunek podany w nawiasach okrągłych przy instrukcji `while`. W przypadku instrukcji `while` należy szczególnie być ostrożnym na zagrożenie związane z powstaniem pętli nieskończonej wynikającym z ograniczonej precyzji. Na przykład poniższy kod wykona pętlę dopóki wartość i będzie różna od 2 (instrukcja `i ~= 2`). Pomimo tego, że wygląda poprawnie spowoduje wykonanie pętli nieskończonej i nie zatrzyma się na wartości 2.

```
i = 0;
while (i ~= 2)
```

```
    disp(i)
    i = i + 1/3;
end
     0
    0.3333
    0.6667
     1
    1.3333
    1.6667
    2.0000   <<<<
    2.3333
    2.6667
    …
```

Takie zachowanie programu związane jest z przybliżoną reprezentacją liczb rzeczywistych. Współczesne komputery reprezentują liczby rzeczywiste za pomocą skończonej sumy potęg liczby 2. Wartość przyrostu w wewnętrznym bloku kodu jest w tym przypadku liczbą wymierną 1/3, która nie ma dokładnej reprezentacji za pomocą skończonej liczby potęg liczby 2. Dlatego rzeczywista wartość i po 6 iteracjach będzie bliska 2 ale nie równa dokładnie. Dlatego bezpieczniejsze konstrukcje warunku w powyższej pętli `while` są następujące:

```
while (abs(i - 2) > e)
```

lub

```
while (i < 2)
```

Aczkolwiek, drugie rozwiązanie, chociaż bezpieczne z punktu widzenia pętli nieskończonej niesie za sobą ryzyko pominięcia wartości 2 w przypadku błędu zaokrągleń odrobinę mniejszego od wartości 2, tak jak w poniższym przykładzie.

```
i = 0;
while (i < 2)
    disp(i)
    i = i + 1/3;
end
       0
    0.3333
    0.6667
     1
    1.3333
    1.6667
    2.0000
```

Bloki programu mogą i zasadniczo powinny być zamykane w funkcje. Powinno się unikać pisania tzw. skryptów czyli plików tekstowych w MATLABie, które zawierają zbiory instrukcji niezorganizowane w funkcje. Pliki funkcyjne w MATLABie tworzy się w postaci plików tekstowych posiadających rozszerzenie z literką ‘m’. Nazwy plików funkcyjnych nie mogą rozpoczynać się od cyfry oraz nie mogą zawierać znaków spacji. Oto przykłady poprawnych nazw plików:

```
a123.m

pierwszy_i_drugi_algorytm.m
```

Oto niewłaściwe nazwy plików:

```
1aaa.m

nazwa_pliku

nazwa ze spacjami.m
```

W MATLABie funkcje deklaruje i definiuje się w następujący sposób:

```
function [zwracana1, zwracana2] = nazwa_funkcji(argument1, argument2)

 % tutaj powinien być blok funkcji

end
```

W powyższym przykładzie użyliśmy słowa kluczowego `function` i zadeklarowaliśmy funkcję o nazwie ‘nazwa_funkcji’, która pobiera jako argumenty wywołania dwie wartości do których możemy odwoływać się wewnątrz bloku funkcji za pomocą zmiennych `argument1` i `argument2`. Zadeklarowana funkcja zwraca dwie wartości: `zwracana1` i `zwracana2`. Zmienne te muszą zostać zadeklarowane tzn. muszą być im przypisane wartości wewnątrz bloku funkcji w dowolnym momencie. Dodatkowo wprowadziliśmy do naszego kodu komentarz. Komentarz to dowolny tekst, który jest pomijany podczas wykonywania programu lub funkcji. Komentarz jednego wiersza w MATLABie rozpoczyna się znakiem %. Jeśli chcemy oznaczyć blok kodu jako komentarz wówczas należy ująć go w znaki %{ i %} `odpowiednio` na początku i na końcu bloku. Pełny, przykładowy kod funkcji może być następujący:

```
function [zwracana1, zwracana2] = nazwa_funkcji(argument1, argument2)

 zwracana1 = argument1 + argument2;

 zwracana2 = argument1 * argument2;

end
```

W powyższym przykładzie zmienna zwracana1 to suma argumentów funkcji a zmienna zwracana2 to ich iloczyn. Wywołanie funkcji następuje poprzez nazwę z nawiasami okrągłymi oraz podanymi argumentami.

```
[a1, a2] = nazwa_funkcji(2,3)
```

Jeżeli chcemy w jednym pliku tekstowym w MATLABie umieścić kilka funkcji, to plik ten powinien zawierać jedną funkcję, która powinna nazywać się tak samo jak plik bez rozszerzenia ‘m’ i funkcja ta powinna znajdować się na początku pliku tekstowego. Zazwyczaj funkcja ta nie pobiera i nie zwraca argumentów. Pozostałe funkcje umieszczamy za tą pierwszą funkcją główna, która jest odpowiednikiem funkcji startowej. Na przykład kompletna zawartość pliku tekstowego o nazwie plik_z_funkcja.m może być następująca:

```
function plik_z_funkcja
```

```
    [a1, a2] = nazwa_funkcji(2,3)
end


function [zwracana1, zwracana2] = nazwa_funkcji(argument1, argument2)
 zwracana1 = argument1 + argument2;
 zwracana2 = argument1 * argument2;
end
```

Uruchomienie programu bezpośrednio z edytora MALABA lub po wpisaniu w linii komend nazwy pliku: `plik_z_funkcja`, spowoduje wyświetlenie na ekranie:

```
>> plik_z_funkcja
a1 =
     5
a2 =
     6
```

Należy zwrócić uwagę, że bieżący katalog linii komend MATLABa musi zawierać utworzony plik tekstowy.

MATLAB posiada bardzo wiele funkcji pozwalających w szybki sposób wizualizować dane. Podstawowa komenda, która umożliwia rysowanie wyników to plot. W najprostszym ujęciu wystarczy podać jako argument wektor, który zostanie wyrysowany w przestrzeni 2D:

```
>> plot(1:10,'k.');
```

Weźmy przykład z dziedziny elektrotechniki, w którym należy zwizualizować jedną fazę trójfazowego odkształconego układu napięć. Gdy zatem chcemy wyrysować przebieg czasowy wymuszenia zawierający składową zerową, podstawową i wyższe harmoniczne można zrobić to w następujący sposób. Definiujemy wektor czasu t, zależny od niego wektor wymuszenia `e_a`, a wszystko rysujemy np. czerwoną linią ciągłą o grubości dwóch punktów:

```
>> t = 0:pi/200:4*pi; omega = 1;
>> e_a = 10 + sin(omega*t) + sin(3*omega*t) + sin(5*omega*t);
>> h=plot(t, e_a);
>> set(h, Color, [1 0 0], 'LineWidth', 2);
>> title('Napiecie odksztalcone');
>> xlabel('Czas t [s]'); ylabel('U [V]');
```

Dodatkowe, przydatne funkcje to np. `subplot` umożliwiający przedstawienie kilku przebiegów w jednym oknie graficznym. Poniższy kod podzieli okno graficzne na dwie części (dwa wiersze w jednej kolumnie) i wyrysuje w nich odpowiednio dwa przebiegi: sygnał `s1` i `s2`:

```
>> subplot(2,1,1), plot(t,s1);
>> subplot(2,1,2), plot(t,s2);
```

Kończąc to bardzo zwięzłe wprowadzenie do podstaw programowania w MATLABie zwróćmy jeszcze uwagę na znak średnika znajdujący się na końcu instrukcji. Umieszczenie

średnika na końcu instrukcji spowoduje tzw. ‘ciche’ wykonanie instrukcji, czyli MATLAB wykona instrukcję i nie wypisze nic na ekranie. Jeżeli chcemy wyświetlić na ekranie w linii komend wynik danej instrukcji to wystarczy pominąć znak średnika. Przykład:

```
>> a = 3;

>> a = 3

a =

     3
```

Jesteśmy świadomi, że powyższe wprowadzenie jest bardzo, krótkie, ale w znaczącej większości wystarcza ono do rozpoczęcia przygody z programowaniem metod numerycznych w środowisku MATLAB.

Wydajność implementacji programów w MATLABie

Zasadniczo w MATLABie wszelkie implementacje pętli nie są wskazane. Powinny one być zastąpione operacjami wektorowymi.

Zadania

1. Podaj przykład skryptu tzw. m-pliku, który jednocześnie powinien móc pełnić rolę funkcji. Przykładowa treść takiego pliku.

Odp.
Plik musi nazywać się: nazwaPliku

Zawartość pliku:

```
function [ arg_wyjsciowe ] = nazwaPliku( arg_wejsciowe )

 % tutaj zaczyna się właściwa treść skryptu

 arg_wyjsciowe = arg_wejsciowe * 2;

end
```

2. Podaj przykład deklaracji zmiennych oraz inicjalizowania ich wartościami (zwłaszcza dotyczy to wektorów i macierzy).

Odp.
Przykładowe instrukcje do inicjalizacji zmiennych wektorowych i macierzowych:

```
u = [-1 -0.75 -0.5 -0.25 0 0.25 0.5 0.75 1];

u = zeros(3, 4)

u = linspace(1, 3, 0.1)
```

3. Podaj przykład pętli dla której znamy liczbę powtórzeń przed jej wykonaniem.

Odp.

```
N = 10;
```

```
for i=0:N-1
     display(i*2);
 end
```

4. Podaj przykład pętli, dla której nie jesteśmy przed jej wywołaniem określić liczby iteracji.

Odp:

Używamy tutaj petli while w połączeniu z funkcją rand, która zwraca losową wartośc z zakresu 0 do 1.

```
i = 0;
while (i < N)
     display(i);
     i = i + rand(1);
end
```

5. Podaj przykłady wyświetlania wartości zmiennych na ekranie.

Odp.
a) pominięcie średnika na końcu instrukcji,

```
a=1
b=2
c = a+b
```

b) użycie funkcji display

```
display(c)
```

c) użycie funkcji fprintf

```
fprintf('Suma = %f\n',c); % 3. sposób
```

6. Podaj przykład automatycznego tworzenia wektorów zawierających ciągi arytmetyczne (np. [0 0.1 0.2 0.3 0.4]).

Odp.:

```
a=1:0.1:3*pi;
x=linspace(-1,1);
b=-10:10;
```

7. Podaj przykład instrukcji do rysowania wykresów jednowymiarowych zawierających kilka przebiegów wraz z legendą dotyczącą poszczególnych przebiegów

Odp.

```
pi=3.14;
```

```
a=1:0.1:3*pi;

s=sin(a);

c=cos(a);

plot(a,s,'r-o',a,c,'b-x');

legend('sin(x)','cos(x)');
```

8. Podaj przykład kodu wykonującego operacje na wektorach i macierzach takich jak: mnożenie, dodawanie, odejmowanie, mnożenie, odwracanie macierzy, transpozycja macierzy

Odp.:

```
a=[1 2 3];

b= [1

 2

 3];

c = a + b % matlab wyświetli błąd, ponieważ nie zgadzają się

          % wymiary wektorów - próbujemy

          % dodać wektor wierszowy do kolumnowego

c = a'+b  % operator ' oznacza transpozycję - jaki będzie

          % wynikowy wektor - kolumnowy czy wierszowy?

c = a+b'  % operator ' oznacza transpozycję -

          % jaki będzie wynikowy wektor - kolumnowy czy wierszowy?

d = a'-b;

iloczyn = a*b % wskaż wynik: a) błąd matlaba, b) wektor c) macierz d) skalar?

iloczyn = b*a % wskaż wynik: a) błąd matlaba, b) wektor c) macierz d) skalar?

iloczyn = b'*a % wskaż wynik: a) błąd matlaba, b) wektor c) macierz d) skalar?

A = [1 -4

4 5];

macierz_odwrotna = inv(A)
```

9. Podaj przykład układu równań liniowych w postaci macierzowej i rozwiązania go za pomocą narzędzi Matlaba.

Odp.:

```
A = [ 1 3
 -4 8];
b = [0 4]';
x = A\b; % specjalny operator dzielenia (odwrotnie pisany)
         % do rozwiązywania układu równań, w x będzie
         % rozwiązanie układu równań Ax=b
```

10. Podaj przykłady generowania danych losowych.

Odp.:

```
A = rand(2)       % macierz 2x2 liczb losowych z zakresu 0-1
A = rand(2,1)     % wektor 2x1 liczb losowych z zakresu 0-1
A = floor(rand(5,1)*100) % wektor o pięciu elementach liczb
                            % całkowitych (funkcja floor) z zakresu
                         % 0 do 100
```

11. Podaj przykład konstrukcji warunkowych (if) oraz odwoływania się do pojedynczych elementów macierzy (A(i) - odwołanie do i-tego elementu wektora A.

Odp.:

```
A = floor(rand(5,1)*100) % wygenerowany
for i=1:5
  if (mod( A(i), 2) == 1) % mod - dzielenie modulo
      fprintf('Element A(%d) macierzy o wartości %d jest nieparzysty\n',i,A(i));
  else
     fprintf('Element A(%d) macierzy o wartości %d jest parzysty\n',i,A(i));
    end
end
```

12. Podaj przykłady wykorzystania mnożenia i dzielenia macierzowego odpowiadających sobie elementów w macierzach i wektorach za pomocą specjalnego operatora .* oraz ./

Odp.:

```
x = 0.1:0.01:10;
y = exp(1./x) .* sin(x);
plot(x,y);
```

13. Podaj przykład pliku tekstowego zawierającego co najmniej dwie funkcje.

Odp.:
Poniżej zamieszczona jest treść jednego m-pliku Matlaba. Proszę zwrócić uwagę, że słowo kluczowe function występuje w nim dwuktrotnie: na początku oraz w połowie skryptu.

```
function nazwaPliku( arg_wejsciowe )
   x = 0.1:0.01:10;
    y = f(x);   % x jest wektorem!, który w całości zostanie przekazany
                % do funkcji f, która zwróci wektor wyników,
                % będących wartościami jakieś funkcji dla każdego,
                % odpowiadającego danemu y'kowi x'a
    plot(x,y);
```

```
end


% ponizej przykladowa funkcja, która w zależności od argumentu czy jest
% wektorem czy skalarem, zwraca wektor lub skalar
function [y] = f(x)
    y = exp(1./x) .* sin(x); % Uwaga! Tutaj są wykonywane operacje
                             % dzielenia i mnożenia nie wektorowego
                             % lecz odpowiadających sobie elementów
                             % tych wektorów!
end
```

Przykładowe pytania egzaminacyjne

Zadanie 1

Napisz funkcję w Matlabie realizującą metodę interpolacji Lagrange'a. Funkcja ma wyliczać wartość wielomianu interpolacyjnego Lagrange'a dla argumentu *x*.

$$W_n(x) = y_0 L_0(x) + y_1 L_1(x) + \ldots + y_n L_n(x)$$

$$gdzie: L_i(x) = \prod_{k=0;k\neq j}^{n} \frac{(x - x_k)}{(x_j - x_k)}$$

$$ostatecznie: W_n(x) = \sum_{j=0}^{n} y_j \prod_{k=0;k\neq j}^{n} \frac{(x - x_k)}{(x_j - x_k)}$$

Rozwiązanie:

```
% xd, yd - wektory definiujące węzły interpolacji
% x - współrzędna dla której zostanie obliczona
% y - wartość interpolacji
function y = lagr(xd, yd, x)
    y = 0;
    for j = 1:length(xd)
        Li = 1;
        for k = 1:length(xd)
            if (k ~= j)
                Li = Li * (x - xd(k)) / (xd(j) - xd(k));
            end
        end
        y = y + yd(j) * Li;
    end
end
```

Zadanie 2

Napisz funkcję w MATLABie wyznaczającą wartość współczynników aproksymacji wielomianowej według poniższego wzoru. Uwaga, do rozwiązania układu równań można zastosować wbudowany w MATLABa operator \. (Np.: a = A\b). Funkcja jako argument powinna otrzymać dwa wektory x_k i f_k definiujące węzły aproksymacji.

$$\begin{bmatrix} m+1 & \sum x_k & \sum x_k^2 & \sum x_k^3 & \dots & \sum x_k^n \\ \sum x_k & \sum x_k^2 & \sum x_k^3 & & & \\ \sum x_k^2 & \sum x_k^3 & \dots & \dots & & \vdots \\ \sum x_k^3 & & \dots & \dots & \dots & \\ \vdots & & & \dots & \dots & \sum x_k^{(2n-1)} \\ \sum x_k^n & & \dots & & \sum x_k^{(2n-1)} & \sum x_k^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ \dots \\ \dots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum f_k \\ \sum f_k x_k \\ \sum f_k x_k^2 \\ \dots \\ \dots \\ \sum f_k x_k^n \end{bmatrix}$$

Rozwiązanie:

```
% x, f - wektory węzłów aproksymacji
% order - rząd wielomianu aproksymującego
% a - wektor współczynników aprosymacji
function a = approx(x, f, order)
    N = order+1;
    n = length(x);
    A = zeros(N, N);
    b = zeros(N, 1);
    for i=1:N
        for j = 1:N
            for k = 1:n
                A(i,j) = A(i,j) + x(k)^(i+j-2);
            end
        end
        for k = 1:n
            b(i) = b(i) + x(k)^(i-1) * f(k);
        end
    end
    a = A\b;
end
```

Zadanie 3

Proszę napisać funkcję, która będzie wykonywać eliminację Gaussa dla zadanego układu równań w postaci macierzy A oraz wektora prawych stron b (Ax=b). Zgodnie z algorytmem:

$$a_{i,j}^{k} = a_{i,j}^{(k-1)} - \frac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}} a_{k,j}^{(k-1)}$$

- for
$$i = k+1, k+2, \dots, n$$
$$j = k, k+1, \dots, n+1$$
$$k = 1, 2, \dots, n-1$$

Rozwiązanie:

```
function Ag = gaussian(A,b)
  Ag = [A b]
  n = size(Ag,1);
  for k=1:n-1
    for i = k+1:n
      l = Ag(i,k) / Ag(k,k);
      for j = k:n+1
        Ag(i,j) = Ag(i,j) - l * Ag(k,j);
      end
    end
  end
end
```

Zadanie 4

Proszę napisać funkcję, która zastosuje algorytm wstecznego podstawienia do układu równań, którego macierz U jest górnotrójkątna (Ux=c). Zgodnie z algorytmem:

$$x_i = \frac{c_i - \sum_{j=i+1}^{n} (u_{ij} \cdot x_j)}{u_{ii}}$$

Rozwiązanie:

```
function x = backward(U,c)
  n = size(U,1);
  x = zeros(n,1);
  for i = n:-1:1
    s = 0;
    for j = i+1:n
      s = s + U(i,j)*x(j);
    end
    x(i) = (c(i) - s) / U(i,i);
  end
end
```

Słownik

Iterator – zmienna służąca do śledzenia kolejnego numeru uruchomienia danego bloku pętli, lub zmienna umożliwiająca kolejne odwoływanie się do elementów listy, tablicy, macierzy.

Pętla – konstrukcja programu umożliwiająca wielokrotne wykonanie wskazanego bloku kodu. Występują dwa rodzaje pętli: pętle, które od początku posiadają informację o całkowitej liczbie iteracji oraz pętle (pętle `for`), oraz które wykonują się dopóki spełnione jest pewne określone kryterium (`while`).

Blok kodu - zbiór instrukcji, które będą wykonywane w ściśle określonej kolejności (po linijkach).

Bibliografia:

1. Cleve Moler: Numerical Computing with MATLAB, 2004 (Free book available online at: http://www.mathworks.com/moler/chapters.html)
2. Kermit Sigmon: http://users.tem.uoc.gr/~spapadem/NLA_2016/Biblio/MATLABPrimer.pdf