JiNP3: pakiety matematyczne

Piotr Krzyżanowski

Wydział Matematyki, Informatyki i Mechaniki
Uniwersytet Warszawski
http://mst.mimuw.edu.pl

27 października 2010

Plan przedmiotu

- 7–8 wykładów
- przegląd pakietów matematycznych
  - numerycznych
  - symbolicznych
- programowanie w MATLABie (**Metody numeryczne!**)
- programowanie w CAS

Zasady zaliczenia

- projekt zaliczeniowy w MATLABie (i zapewne w czymś jeszcze)

Materiały pomocnicze

- pakiety dostępne w LABie
- dokumentacja online (w trakcie wykładu)
- są także wersje demo oraz darmowe klony
- mnóstwo podręczników, ale zorientowanych na zastosowania

Pakiety matematyczne — po co?

- weryfikacja hipotez
- uciążliwe rachunki
- rysunki, wizualizacje

Pakiety matematyczne — jak?

- zwykle interpretowane — interakcja z użytkownikiem
- język bardzo wysokiego poziomu, proceduralny lub funkcyjny
- łatwa i nie zawsze konsekwentna składnia
- główna moc języka: specjalistyczne funkcje
- wewnętrznie oparte na „klasycznym" języku programowania proceduralnego

Pakiety matematyczne — mity

Reklama sugeruje, że pakiety mogą poważnie wesprzeć matematyka w takich dziedzinach, jak:

- dowodzenie twierdzeń
- bezbłędne rachunki
- wyniki szybciej niż na kartce
- uwolnienie od myślenia

To wszystko nieprawda, a przynajmniej gruba przesada!

Rachunek symboliczny

- **Rachunek symboliczny**: kwintesencja matematyki czystej
- Przeprowadzenie go od ogólnie sformułowanego zadania z parametrami do końcowego, eleganckiego rozwiązania (zazwyczaj: wzoru) wymaga zazwyczaj
  - dużej wiedzy matematycznej,
  - finezji,
  - szczęścia.
- Bywają zadania, w których nie jesteśmy w stanie podać precyzyjnego wzoru na rozwiązanie.

Obliczenia numeryczne

- **Rachunek numeryczny** jest koniem pociągowym matematyki stosowanej.
- Modele matematyczne realnych zjawisk są najczęściej na tyle skomplikowane, że badanie ich metodami matematyki teoretycznej często jest niemożliwe.
- Metody numeryczne i ogólniej przybliżone pozwalają uzyskać wgląd w naturę modelowanych zjawisk.

Rachunek symboliczny kontra obliczenia numeryczne

- Tam, gdzie rachunki numeryczne zawodzą — tam trzeba korzystać z metod matematyki czystej, „teoretycznej".
- Matematykę czystą najlepiej uprawia się przy użyciu giętkiego umysłu, kartki papieru i ołówka. Czasem może wspomóc ją oprogramowanie potrafiące przeprowadzić żmudne (te nudne!) rachunki symboliczne.
- Tam, gdzie finezja matematyka–teoretyka (lub pakietu algebry symbolicznej) nie jest w stanie sobie poradzić z zadaniem, znakomitym ratunkiem jest wykorzystanie rachunku numerycznego.
- Matematykę obliczeniową bezsprzecznie najlepiej uprawia się z pomocą szybkiego komputera.

Podsumowanie

Metoda rachunku	**numeryczny**	**symboliczny**
Możliwość rozwiązywania zadań praktycznych	zazwyczaj tak	zazwyczaj nie
Wielość metod o różnej skuteczności	tak	tak
Wymaga wiedzy wykraczającej poza zadanie	najczęściej tak	najczęściej nie
Wynik	liczby lub rysunek	wzór
Działa na abstrakcyjnych obiektach	nie	tak
Radzi sobie z nieskończonościami	zazwyczaj nie	zazwyczaj tak
Dobrze radzi sobie z mnogością parametrów	tak	nie
Precyzja wyniku	ograniczona	teoretycznie nieskończona
Ostateczna jakość wyniku	niepewna	niepewna

Rozdział I

Wprowadzenie do MATLABa

MATLAB — język i środowisko obliczeń numerycznych

- `http://www.mathworks.com`
- używany powszechnie przez inżynierów
- lista klientów imponująca
- oni rzeczywiście go stosują!
- ...do pary z Simulinkiem...

Krótka historia MATLABa

- ok. 1980 — Cleve Moler tworzy w Fortranie prosty interfejs do bibliotek numerycznych, język skryptów, narzędzia do manipulacji **macierzami**
- „**MAT**rix **LAB**oratory"
- żeby studenci nie musieli pisać programów w Fortranie
- darmowy! (dla studentów i kolegów)
- 1983 — przepisanie języka z Fortranu do C (Moler, Little, Bangert)
- 1984 — komercjalizacja i kolejne wersje
- 2000 — wersja 6.0, znacząco odświeżona
- 2009 — wersja 7.9
- co 6 miesięcy aktualizacja
- **drogi!**
- pakiety rozszerzające: Toolbox'y

MATLAB na naszym Wydziale

- 15 licencji na MATLAB i Optimization Tlbx
- zarówno pod Linuxem (students), jak i pod Windows
- klony MATLABa (darmowe, ale nie w 100% kompatybilne)
  - Octave: `http://www.octave.org` wersja webowa: `http://hara.mimuw.edu.pl/weoctave`
  - Scilab: `http://www.scilab.org`

MATLAB — IDE

Obecnie jest to w pełni zintegrowane środowisko obliczeniowe

- interpreter do pracy w trybie interaktywnym
- edytor z analizatorem składni
- kompilator JIT
- debuger
- profiler
- inspektor zmiennych
- pełna dokumentacja i help kontekstowy
- zaawansowana wizualizacja
- możliwość rozszerzeń
- ...itd...

MATLAB — praca w trybie interaktywnym

Zobaczmy, jak to działa!

- proste komendy
- system pomocy
- MATLAB jako kalkulator
- części składowe desktopu MATLABa

Najprostsze działania w MATLABie

- podstawowy obiekt: macierz dwuwymiarowa (w rzeczywistości: **tablica *N*-wymiarowa**)
- podstawowe operacje na macierzach
- najprostszy zapis/odczyt danych

Rozdział II

MATLAB ogólnie

Matematyka w MATLABie

- algebra liniowa (MAT-LAB!)
- funkcje matematyczne
- równania nieliniowe, minimalizacja (Optimization Tlbx)
- całkowanie
- wielomiany, interpolacja
- splajny (Spline Tlbx)
- FFT (DSP Tlbx)
- ...itd...

Algebra liniowa

- rozwiązywanie układów równań liniowych
- LZNK
- wartości i wektory własne
- macierze gęste i rzadkie
- własności macierzy
- macierze szczególne
- ...itd...

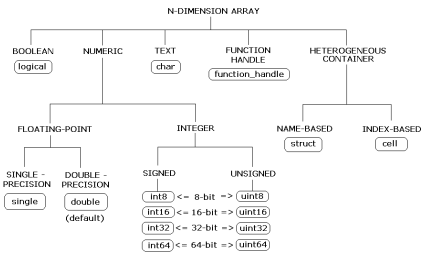
Matematyka w MATLABie

Chociaż stanowią o rzeczywistej sile MATLABa... nie będziemy się nimi zajmować więcej, niż jest to niezbędne.
Powód?
— Funkcje MATALBa służące rozwiązywania zadań obliczeniowych będą omawiane (mam nadzieję) na LABie z **Metod numerycznych**!

Programowanie w MATLABie

- typy danych
- operatory
- funkcje i skrypty
- zmienne
- instrukcje sterujące
- wyznaczanie wartości
- indeksy
- błędy i ostrzeżenia

Typy danych



Rysunek: Hierarchia klas w MATLABie. *Źródło: MATLAB 7 Programming Fundamentals.*.

Typy danych

- wszystko jest N-wymiarową ($N \geqslant 0$) tablicą
- realnie, podstawowym obiektem jest dwuwymiarowa tablica liczb rzeczywistych (podwójnej precyzji): macierz (...i tak się to zaczęło...)
- skalar: macierz jednowymiarowa
- czym jest macierz pusta, $Q = []$, zerowymiarowa?

Typ rzeczywisty — liczby rzeczywiste w komputerze

Tylko wybrane liczby wymierne można dokładnie reprezentować w arytmetyce procesora. Wszystkie inne — tylko w sposób przybliżony. Na przykład, liczba 0.1 **nie jest** dokładnie reprezentowana w procesorze.

Tabela: Typy zmiennoprzecinkowe w standardzie IEEE 754. Orientacyjne zakresy liczb (znormalizowanych, dodatnich) reprezentowalnych w tym typie oraz precyzja reprezentacji

Bity	Typ	Zakres	Precyzja
32	poj. prec.	$10^{-38} \ldots 10^{38}$	10^{-7}
64	podw. prec.	$10^{-308} \ldots 10^{308}$	10^{-16}
$\geqslant 80$	rozsz. podw. prec.	$10^{-308} \ldots 10^{308}$	10^{-20}

Typ rzeczywisty — liczby rzeczywiste w języku C

Tabela: Typy zmiennoprzecinkowe w C. Orientacyjne zakresy liczb (znormalizowanych, dodatnich) reprezentowalnych w tym typie oraz precyzja reprezentacji

Bity	Typ	Zakres	Precyzja
32	**float**	$10^{-38} \ldots 10^{38}$	10^{-7}
64	**double**	$10^{-308} \ldots 10^{308}$	10^{-16}
96	**long double**	$10^{-308} \ldots 10^{308}$	10^{-20}

Typ rzeczywisty

- podwójnej precyzji (domyślnie), double(x)
- pojedynczej precyzji, single(x)
- nie ma podwójnej rozszerzonej precyzji
- wszystkie obliczenia są prowadzone w podwójnej precyzji
- charakterystyka: **eps**, **realmax**, **realmin**, **eps**('single'), **realmax**('single'), **realmin**('single')
- double w wyrażeniach z single daje **single** (odwrotnie niż w C)
- double/single w wyrażeniach z typem całkowitym daje ten typ całkowity

Typ rzeczywisty — wartości specjalne

- inf, −inf — nieskończoności „$\pm\infty$"; mają pożądane własności: $1/0 = \infty$, $-1/0 = -\infty$, $\infty - 1 = \infty$
- **NaN**, „not a number": $0/0 = NaN$, $\infty - \infty = NaN$,
- **isinf**(x), **isnan**(x) sprawdzają, czy $x = \pm\infty$ lub, odpowiednio, $x = NaN$
- **uwaga: NaN'y są nieporównywalne!**

Typ rzeczywisty — liczby zespolone

- konstruktor: z = 5+3i z = complex(a, b) daje liczbę $z = a + i \cdot b$
- **real**(z), **imag**(z)
- domyślnie zmienne i oraz j mają wartość równą $\sqrt{-1}$, ale to zawsze może być zmienione przez programistę, np. i = [1 2 3], zatem 5+3*i może być ryzykowne...

Typ całkowity

Aby uzyskać w MATLABie liczbę całkowitą, musimy ją skonwertować z typu podwójnej precyzji do żądanego, używając poniższych funkcji:

Ze znakiem

Tabela: Typy całkowite ze znakiem w MATLABie.

Bity	Zakres	Funkcja
8	$-2^7 \ldots 2^7 - 1$	int8(x)
16	$-2^{15} \ldots 2^{15} - 1$	int16(x)
32	$-2^{31} \ldots 2^{31} - 1$	int32(x)
64	$-2^{63} \ldots 2^{63} - 1$	int64(x)

Bez znaku Wiadomo jak, funkcje zyskują przedrostek „u", np. uint32(x)

- Uwaga na konwersję liczby ujemnej do typu bez znaku... — zwracane jest zero!
- Uwaga na konwersję zbyt dużej liczby do typu całkowitego...
- Typy 64-bitowe tylko do indeksowania: nie można na nich prowadzić działań arytmetycznych

Typ rzeczywisty — funkcje zaokrąglające

- **round**(x) — do najbliższej
- **floor**(x) — w dół
- **fix**(x) — w stronę zera
- **ceil**(x) — w górę

Ich wynik **nie jest liczbą całkowitą!**

Typ numeryczny — weryfikacja

- isfloat(x) — czy jest tablicą typu rzeczywistego
- **isreal**(x) — czy jest tablicą typu nie-zespolonego
- isinteger(x) — czy jest tablicą typu całkowitego
- isnumeric(x) — czy jest tablicą typu numerycznego

Macierze: numeryczne tablice dwuwymiarowe

- dla typu podwójnej precyzji i logicznego, można wybrać reprezentację w formacie macierzy rozrzedzonej (więcej o tym — później)
- Wiele specjalizowanych funkcji matematycznych działa tylko na macierzach;
- operator konstrukcji: [], np.

```
A = [ x, y, z; P, q];
```

tworzy macierz

$$A = \begin{pmatrix} x & y & z \\ P & q & \end{pmatrix}$$

- funkcje zwracające popularne macierze: **eye**(n,m), ones(n,m), **zeros**(n,m), **rand**(n,m), itd.
- macierz innego typu niż domyślny: **zeros**(n,m,'uint8'), **rand**(n,m,'single'), itd.

Tablice numeryczne w MATLABie

- możliwość konstruowania tablic dowolnego wymiaru dla dowolnego typu numerycznego
- operator konstrukcji: [], w połączeniu z wprowadzeniem dodatkowego wymiaru np.

```
A(:,:,1) = [1 2 3; 4 5 6; 7 8 9]
A(:,:,2) = [1 0 4; 3 5 6; 9 8 7]
```

tworzy macierz $3 \times 3 \times 2$.

- ones($n_1,\ldots,n_N$), **zeros**($n_1,\ldots,n_N$), **rand**($n_1,\ldots,n_N$)
- duplikacja macierzy repmat(A, [$n_1,\ldots,n_N$])

Tablice numeryczne w MATLABie

- Dostęp do elementu x = A(3,4,2,10)
- Indeksowanie zakresami P = A(3:4,2:**end**,:,[1 3])
- Własności tablicy
  - **size**(A) — wymiary,
  - numel(A) — liczba elementów,
  - ndims(A) — ilowymiarowa jest tablica,
  - **whos**(A) — informacja m.in. o zajmowanej pamięci i klasie,
  - **length**(A) — długość wektora

Oczywiście, działa to także dla macierzy!

Typ logiczny

Zajmuje tyle miejsca, co uint8, ale przyjmuje tylko dwie wartości: 0 i 1 (znane również jako false i true)

- oczywiście można organizować w macierze i tablice
- islogical(x) sprawdza, czy obiekt jest typu logicznego

Stringi w MATLABie

- znak to faktycznie integer (Unicode)
- c = 'z' tworzy pojedynczy znak
- string to po prostu tablica znaków, c = 'zebra'; c(3:**end**)
- **num2str**(x) (**int2str**(x)) zamienia liczbę rzeczywistą (całkowitą) na string
- char(x) zamienia wektor liczb całkowitych na „prawdziwy" string z takimi samymi kodami
- uint8(s) zamienia string na „prawdziwy" wektor liczb

MATLAB golf — krótkie zasady

- gra składa się z zadań — „dołków"
- wynikiem jest długość kodu źródłowego: im mniejsza, tym lepszy wynik
- czas działania nie odgrywa istotnej roli (dlatego jest to tylko zabawa)
- tylko kod poprawnie działający na spreparowanym zestawie testowym jest dopuszczony do konkurencji
- nie wolno stosować sztuczek niegodnych gentlemana
- punktacja za każdy dołek:
  - **I miejsce** — 3 punkty
  - **II miejsce** — 2 punkty
  - **III miejsce** — 1 punkt

MATLAB golf: przykład

Kod przypisujący zmiennej x iloczyn zadanej macierzy A i wektora y.

Kod	Wynik (pkt)
x=A*y	5
x = A*y;	8
A*y	nie klasyfikowany
system('mnozenie.exe')	zdyskwalifikowany

MATLAB golf — dołek pierwszy

Napisz skrypt, przypisujący zmiennej x wartość największego co do modułu elementu zadanej tablicy A. Uwaga: ta wartość może być ujemna!

Rozdział III

Programowanie w MATLABie

Zmienne

- zmiennych nie trzeba deklarować (są wyjątki od tej reguły)
- zmienna to w zasadzie etykieta dla wartości znajdującej się w pamięci komputera (lub ich zestawu — np. tablice)
- zmienne są takiego typu, jak wartości, które reprezentują (domyślnie typu double, ale można to zmienić, jak wcześniej widzieliśmy)
- usuwamy zmienną (i wartości jej przypisane) z pamięci poleceniem **clear**, np. **clear** x ala B usunie zmienne x, ala, B (**uwaga**: elementów tej listy nie wolno oddzielać przecinkami!)
- zmienne zapisujemy do pliku poleceniem wysokiego poziomu **save**, np. **save** 'filename.mat'x ala B zapisuje do pliku `filename.mat` zmienne x, ala, B
- zmienne odczytujemy z pliku poleceniem wysokiego poziomu **load**, np. **load** 'filename.mat' wczytuje z pliku `filename.mat` wszystkie zapisane w nim zmienne; **load** 'filename.mat'x wczytuje tylko zmienną x
- format tego pliku jest specyficzny dla MATLABa, ale jego specyfikacja jest jawna

Operatory

Wartości wyrażeń są wyznaczane od lewej do prawej.

- macierzowe operatory arytmetyczne $+$, $-$, $*$, $/$, $\backslash$, $\hat{}$ — mają znaczenie macierzowe (mnożenie macierzy, rozwiązywanie układu równań, itd.)
- tablicowe operatory arytmetyczne $+$, $-$, $.*$, $./$, $.\hat{}$ — są przykładane do elementów macierzy
- operatory logiczne (tablicowe) &, |, ~ oraz wersje zwarte (tylko skalary) &&, ||
- operatory relacyjne $==$, $\tilde{}=$, $>$, $>=$ (i w stronę przeciwną) (dodatkowo, funkcje **any**, **all**)

Kolejność działań jest standardowa; w razie wątpliwości nie zaszkodzi użyć nawiasów...

Konstrukcje sterujące

- pętle: **for**..**end**, **while**..**end**
- warunki: **if**..**elseif**..**else**..**end**, switch..case..**end**
- obsługa błędów: try..catch..**end**

Staramy się unikać pętli po elementach tablic na rzecz operacji/instrukcji tablicowych (efektywność).
Kompilator JIT nieco osłabia efekt wektoryzacji, ale w sytuacjach odbiegających od banału traci moc.

Skrypty

- zestaw komend MATLABa możemy zorganizować w zewnętrznym pliku w formie skryptu
- nazwa skryptu musi mieć rozszerzenie `.m`
- skrypt jest tak traktowany jak zestaw komend, które zawiera
- skrypty wykonujemy z poziomu linii komend
- skrypty mogą wywoływać inne skrypty

Skrypty — przykład z macierzą

Przypuśćmy, że naszym zadaniem jest utworzenie macierzy postaci:

$$L_N = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2 & \cdots & 2 \\ 1 & 2 & 3 & \cdots & 3 \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & 2 & 3 & \cdots & N \end{pmatrix}_{N \times N} \tag{1}$$

Właśnie takiej postaci jest macierz odwrotna do tzw. macierzy jednowymiarowego Laplasjanu z mieszanymi warunkami brzegowymi:

$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}^{-1}_{N \times N} = L_N$$

Skrypt wyznaczający macierz

Dla $N = 5$, moglibyśmy użyć na przykład następującej sekwencji instrukcji:

```
N = 5;
for i=1:N
  for j=1:i
    L(i,j) = j;
  end
  for j=i:N
    L(i,j) = i;
  end
  L
end
```

Te same instrukcje możemy wpisać do skryptu `lap1dscript.m`!

Kilka wniosków

- znakomite do tworzenia *ad hoc* i edycji z boku
- prealokacja dużych obiektów jest koniecznością!
- wady strukturalne:
  - efekty uboczne skyptu: zmienne w skrypcie istnieją wprost w przestrzeni roboczej użytkownika (ale jednocześnie mogą korzystać z innych zmiennych z przestrzeni roboczej!)
  - kłopotliwa „parametryzacja"

Funkcje

```
function [a, b, c] = myfun(x, y)
  ...instrukcje...
end
```

- podobnie jak skrypt, muszą być zdefiniowane w pliku
- nazwa funkcji musi być tożsama z nazwą pliku
- wewnętrzne zmienne są lokalne (są wyjątki) i kończą żywot z chwilą opuszczenia funkcji (są wyjątki)
- argumenty są formalnie przekazywane przez wartość (są wyjątki), ale...
- „...internally, MATLAB optimizes away any unnecessary copy operations." *Źródło: MATLAB 7 Programming Fundamentals.* ale co MATLAB zrobi z poniższym kodem:

```
function HugeMatrix = myfun(HugeMatrix, k)
  HugeMatrix(range(k)) = 2; % być może tylko 1 element jest modyfikowany
end
```

- „In the case of passing structures or cell arrays, only the field or cell data being modified by the function will be passed "by value"."

Przykład prostej funkcji

```
function y = average(x)
% AVERAGE Mean of vector elements.
% AVERAGE(X), where X is a vector, is the mean of vector
% elements. Nonvector input results in an error.
[m,n] = size(x);
if (~((m == 1) | (n == 1)) | (m == 1 & n == 1))
    error('Input must be a vector');
end
y = sum(x)/length(x); % Actual computation
```

Źródło: MATLAB 7 Programming Fundamentals.

- komentarze
- komentarz blokowy

```
%{
...komentowane instrukcje...
%}
```

- tekst pomocy
- weryfikacja argumentów, **error**
- nowe zmienne, wyniki

Obsługa argumentów

Parametry wewnętrzne **nargin**, **nargout** pozwalają reagować na liczbę argumentów na wejściu i wyjściu:

```
if (nargin < 2)
  p = 2;
  if (nargin < 1)
    usage('[L, norma] = invlap1d(N,p); N - wymiar macierzy p - która norma');
  end
end
...
if (nargout > 1)
norma = norm(L,p);
end
```

„Argument" wejściowy varargin i analogiczny wyjściowy varargout pozwala tworzyć funkcje o zmiennej liczbie argumentów (raczej egzotyczne w MATLABie).

```
function [out1,out2] = example1(a,b,varargin)
function [i,j,varargout] = example2(x1,y1,x2,y2,flag)
```


Funkcje pomocnicze

Zapisane do pliku `myfun1.m`

```
function y = myfun1(a,b)
...
end

function [y, z] = myfun2(a,b,c)
...
end
```

- tylko pierwsza (główna) funkcja jest dostępna na zewnątrz.
- funkcje pomocnicze widzą się nawzajem i mogą się wywoływać
- nie współdzielą swoich zmiennych lokalnych

Funkcje zagnieżdżone

Zapisane do pliku `myfun1.m`

```
function y = myfun1(a,b)
...

  function [y, z] = myfun2(a,b,c)
  ...
  end
end
```

- tylko zewnętrzna funkcja jest dostępna na zewnątrz.
- funkcje wewnętrzne widzą tylko funkcje swojego poziomu, wyższego, lub **o jeden** niższego
- mają dostęp do wszystkich zmiennych lokalnych funkcji w których są zagnieżdżone

Funkcje prywatne

Zwyczajne funkcje, ale zapisane do podkatalogu `private`

- funkcje prywatne są widoczne tylko dla funkcji z katalogu bezpośrednio nad `private`
- ich nazwy przesłaniają ewentualne identyczne nazwy innych funkcji

Zmienne globalne i statyczne

- zmienne globalne — widoczne wszędzie i dostępne dla wszystkich (co może być ryzykowne, ale bywa szalenie wygodne)
- zmienne statyczne — zachowują swoją wartość pomiędzy wywołaniami używającej jej funkcji

MATLAB golf — dołek drugi

Napisz skrypt, przypisujący zmiennej x macierz, będącą uśrednieniem zadanej macierzy (dwuwymiarowej) A. Jest to macierz tego samego rozmiaru, co A. Elementy macierzy wynikowej są zadane wzorem:

$$x_{i,j} = \frac{1}{4}\left(a_{i,j+1} + a_{i,j-1} + a_{i+1,j} + a_{i-1,j}\right).$$

Należy przyjąć ((nad)używając terminologii MATLABa), że

$$a_{i,0} = a_{i,\text{end}+1} = a_{0,j} = a_{\text{end}+1,j} = 0.$$

Rozdział IV

Programowanie w MATLABie — c.d.

Funkcje jako argumenty innych funkcji

Oczywiście, każdą funkcję dostępną w bieżącym katalogu lub w ścieżce poszukiwań, można użyć wewnątrz innej funkcji w tym położeniu (w tym sensie takie funkcje są globalne. Ale jak przekazywać funkcję jako **argument**?
Dwa sposoby przekazywania;

- jako nazwę (staromodne)
- jako uchwyt, czyli w rzeczywistości (opatentowany) wskaźnik do funkcji

Funkcje jako argumenty — przez nazwę

- Definicja funkcji wywoływanej

```
function [a, b] = groza( x, y, z )
..
end
```

- Definicja funkcji wywołującej

```
function r = myfun( fname, p, w)
..
[z, t] = feval(fname, a1, a2, a3);
..
end
```

- Wywołanie

```
X = myfun('groza', P(1), P(2));
```


A propos feval: wywrotowe polecenie eval

Polecenie **eval**(string) powoduje wykonanie poleceń MATLABa zawartych w tym stringu, np.

```
V0 = pi;
for i=1:4
  eval(['V',num2str(i),'␣=␣i+V',num2str(i−1)]);
end
```

Uwaga. W w/w przypadku lepiej użyć **tablicy**:

```
V = zeros(4,1);
V(1) = pi;
for i=1:4
  V(i+1) = i+V(i);
end
```


Funkcje jako argumenty — przez uchwyt

- Definicja funkcji wywoływanej

```
function [a, b] = groza( x, y, z )
..
end
```

- Definicja funkcji wywołującej

```
function r = myfun( fname, p, w)
..
[z, t] = fname( a1, a2, a3);
..
end
```

- Wywołanie

```
X = myfun(@groza, P(1), P(2));
```


Przykłady realnych funkcji numerycznych w MATLABie

Na przykładzie znanych nam metod dla równań nieliniowych, przyjrzymy się dokładniej praktyce implementowania funkcji w MATLABie.

- bisekcja dla równań skalarnych
- implementacja równania Chandrasekhara

$$x_i - \left(1 - \frac{c}{2N}\sum_{j=1}^{N}\frac{\mu_i \cdot x_j}{\mu_i + \mu_j}\right)^{-1} = 0, \qquad \forall i = 1, \ldots, N \qquad (2)$$

gdzie $c \in (0,1)$ oraz $\mu_j = \frac{1}{N}\left(i - \frac{1}{2}\right)$.

Ciąg dalszy — demonstracja: pisanie funkcji na żywo w trakcie wykładu, z komentarzami, co, dlaczego i dlaczego nie. Dokończenie na następnym wykładzie.

Rozdział V

Programowanie w MATLABie — c.d.

Struktury i zmienne tablicowe

- struktura:

```
options.maxit = 20;
options.rtol = 1e-5;
x = banach(@ff, x0, options);
```

  - Podstawowe funkcje dostępu: getfield(options, 'maxit') (pole maxit musi istnieć!) setfield(options, 'maxit', 22) ustawia wartość pola w istniejącej strukturze options.
  - Sprawdzenie, czy istnieje pole: isfield(options, 'maxit')
  - Sprawdzenie, czy istnieje zmienna: **exist**('options', 'var') == 1
- zmienna tablicowa: kontener na dowolne obiekty

```
msg = cell(4,1); % tworzymy pustą tablicę z miejscem na 4 komórki
msg{1} = 'Sukces!';
msg{2} = 'Osiągnięto maksymalną liczba iteracji';
msg{3} = 'Stagnacja';
msg{4} = 'Nieznany błąd';
msg{5} = @sin;
error_code = 2;
msg{5}(pi)
```


Przykłady realnych funkcji numerycznych w MATLABie

Na przykładzie znanych nam metod dla równań nieliniowych, przyjrzymy się dokładniej praktyce implementowania funkcji w MATLABie.

- bisekcja dla równań skalarnych: opcje i ich sprawdzanie
- implementacja równania Chandrasekhara — c.d.

$$x_i - \left(1 - \frac{c}{2N}\sum_{j=1}^{N}\frac{\mu_i \cdot x_j}{\mu_i + \mu_j}\right)^{-1} = 0, \qquad \forall i = 1, \ldots, N \qquad (3)$$

  gdzie $c \in (0,1)$ oraz $\mu_j = \frac{1}{N}\left(i - \frac{1}{2}\right)$.
- implementacja wielowymiarowej metody Newtona

Demonstracja na żywo w trakcie wykładu.

Instrukcje wyjścia/wejścia

- **dlmwrite**(nazwapliku, macierz, separator)/**dlmread** — np. wygodne by zapisać plik w formacie CSV
- **disp**, **input**, **menu**, **load**, **save**, ...

Instrukcje wyjścia/wejścia niskiego poziomu

fid = **fopen**(nazwapliku,tryb,endian) otwiera plik, **fclose**(fid) — zamyka. nazwapliku to string z nazwą, tryb to string 'r', 'w', itd., endian to string 'l' (*little endian*, x86 i x86-64) albo 'b' (*big endian*, np. SPARC, PowerPC), domyślnie — właściwy hostowi

- tekstowe **fprintf**/**fscanf** obsługują także wektory i macierze; brak deskryptora pliku w **fprintf** oznacza stdout
- tekstowe **fgetl** (wczytanie linii) i **fgets** (wczytanie stringa)
- binarne **fwrite**/**fread** (jeśli nietypowa *endianness*, należy ją podać jako parametr)

Rozdział VI

Inne możliwości programistyczne i merytoryczne MATLABa

Funkcje anonimowe

Definiowane pojedynczym wyrażeniem

- jeszcze jeden sposób konstrukcji funkcji

```
sqr = @(x) x.^2; % sqr jest wskaźnikiem do zdefiniowanej funkcji
z = sqr(4);
pwr = @(x,p) x.^p; % sqr jest wskaźnikiem do zdefiniowanej funkcji
z = pwr(4,);
```

- wbrew pozorom mocne narzędzie, pozwalające (wreszcie!) tworzyć nowe funkcje „w locie", nie tylko w funkcjach, ale też w skryptach
- wykorzystują zmienne w wyrażeniu zostają zamrożone

```
q = 2;
pqr = @(x) x.^q; % pqr jest wskaźnikiem do zdefiniowanej funkcji
pqr(3)
q = 13;
pqr(3)
```

- alternatywny (i „zawsze" skuteczny) sposób przekazania dodatkowych argumentów do wnętrza funkcji

funkcja wyznaczająca normę L^p zadanej funkcji, ...

Wskaźniki do funkcji — c.d.

- Wskaźnik do funkcji zawiera pełną, globalną informację o niej
- functions(h) wyświetla częściową informację of funkcji wskazywanej przez h
- Można to wykorzystać do wyprowadzenia funkcji lokalnej/prywatnej na zewnątrz

```
function h = myfun(a, b, c)
h = @locfun;
    function z = locfun(x)
    z = a*x.^3 + b*x.^2 + c*x';
    end
end
```

- Czy h jest wskaźnikiem do funkcji: isa(h, 'function_handle')
- h = str2func('fname') zamienia nazwę funkcji na wskaźnik; fname = func2str(h) działa w drugą stronę

Wskaźniki do funkcji — c.d.

- Porównanie wskaźników do funkcji: isequal(h1,h2)
- Mimo identyczności wskazywanych funkcji, MATLAB może twierdzić, że wskaźniki nie są równe (np. w przypadku funkcji anonimowych)
- Wskaźniki można zapisywać do plików, ale może to być nieskuteczne (np, wskaźnik korzystał z funkcji, której źródło zostało w międzyczasie usunięte)
- Efekty uboczne tego, że wskaźnik zawiera pełną informację o funkcji

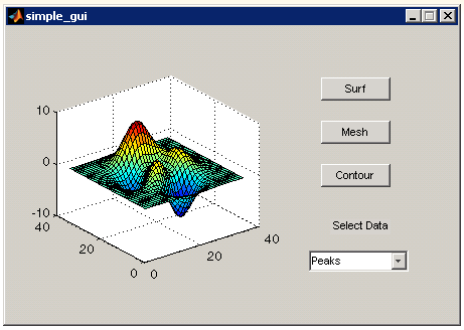
```
function h = counter
x = 0;
h = @addOne;
    function y = addOne;
    x = x + 1;
    y = x;
    end
end
h = counter; h()
    1
h()
```


Nie działa w przypadku funkcji anonimowych

Programowanie obiektowe

- w starszych wersjach niemożliwe
- podobieństwo do struktury
- definicja jednej klasy (np. chebyshev) jej metod w osobnych plikach: folder `@chebyshev`
- przykład: pakiet CHEBFUN, implementujący w trybie numerycznym działania „symboliczne" na funkcjach jednej zmiennej
- dwa rodzaje klas: *value classes* i *handle classes*, te ostatnie ponoć obiecują przekazywanie argumentów przez referencję
- manual: *Źródło: MATLAB 7 Object-Oriented Programming.*

Programowanie GUI



Rysunek: Przykładowy GUI *Źródło: MATLAB 7 Creating Graphical User Interfaces...*

- manual: *Źródło: MATLAB 7 Creating Graphical User Interfaces.*

Garść praktycznych porad

- łamanie linii kodu: ...
- builtin wywołuje funkcję wbudowaną o zadanej nazwie (nawet wtedy, jeśli została przesłonięta przez inną);
- rozkłady macierzy mają swoje wersje ekonomiczne [L U p] = **lu**(A, 'vector'), [Q R] = **qr**(A, 0)
- macierze rzadkie **sparse**(i,j,v,N,M) i jak z nimi postępować
- wektoryzacja kluczowa dla efektywności, a także dla użyteczności kodu

```
g = @(s)1;
>> quad(g,0,1)
??? Index exceeds matrix dimensions.

Error in ==> quad at 85
if ~isfinite(y(7))
```


Garść praktycznych porad — wektoryzacja

- indeksowanie zakresami
- indeksowanie logiczne Vgood = V(D>=0);
- używać funkcji zwektoryzowanych z argumentami macierzowymi
- używać funkcji wbudowanych **cumsum**, **cumprod**, **diag**, **zeros**, ones, repmat, **reshape**, **find**, **diff**, **max**, **min**, **sort**, unique
- proste rekurencje zastępować operacjami macierzowymi,

```
% gorzej:
A = 1;
L = 1000;
for i = 1:L
  A(i+1) = 2*A(i)+1;
end
% lepiej:
L = 1000;
A = filter([1],[1 -2],ones(1,L+1));
```


Wizualizacja

Bardzo dużo funkcji służących rysowaniu najrozmaitszych wykresów.

- dwuwymiarowe:
  - krzywe: **plot**, **semilogy**, **loglog**, **polar**, **contour**, ...
  - punktowe: scatter, **spy**, **bar**
  - wypełnione: pie, contourf, **imagesc**, ...
  - strzałkowe: **quiver**, **feather**
- trójwymiarowe:
  - krzywe: **plot3**, **contour3**, **waterfall**
  - powierzchnie: **surf**, **mesh**, ...
  - punktowe: scatter3
  - strzałkowe: quiver3, coneplot
  - linie prądu: streamline, streamribbon, streamtube, ...

Rozdział VII

System obliczeń symbolicznych (i nie tylko): Mathematica

Pakiety CAS

CAS = Computer Algebra System

- Macsyma/Maxima
  - prekursor pozostałych
  - Lisp, lata 70-te XX w.
  - niszowa wersja open-source Maxima, bez rozszerzeń z wersji komercyjnych
- Maple
- Mathematica
- MuPAD
  - zamienił się w Symbolic Computation Toolbox w MATLABie
- Axiom
- ...i wiele, wiele innych

Maple i Mathematica to obecnie najwięksi gracze.

Mathematica

- Stephen Wolfram, 1988
- wzorowana na Macsymie, ale ze znacznie bardziej konsekwentną składnią
- w chwili wprowadzenia oszałamiała możliwościami graficznymi
- początkowo dla Macintosh, ale szybko dla MS Windows, SunOS, NeXT
- znakomity marketing, popularna na uniwersytetach

Możliwości systemu

- manipulacja symbolami + zaszyta wiedza „matematyczna” (Shift + Enter)
- obliczenia numeryczne
- programowanie funkcyjne i proceduralne

Demonstracja na żywo w trakcie wykładu.

Ograniczenia systemu

- dla obliczeń symbolicznych: nie zawsze wyniki są zgodne z regułami matematycznymi
- wprowadzenie kropki dziesiętnej oznacza przejście na obliczenia numeryczne
- w przypadku obliczeń numerycznych: najpierw wykonywane są numeryczne uproszczenia, a potem dopiero — numeryczne obliczenia
- trudno wymusić na pakiecie symbolicznym obliczenia w zadanej dziedzinie
- większość zadań nie daje się rozwiązać za pomocą symbolicznych rachunków na komputerze
- do obliczeń numerycznych lepiej stosować bardziej wyspecjalizowane pakiety (np. MATLAB)

Programowanie w Mathematice

- podstawowa struktura danych: lista (symboli, liczb, tekstów, wyrażeń) {1,2,3} lub {a,b,4}
- wszystko jest **wyrażeniem** działającym na pewnym zestawie argumentów: a+b+c jest w rzeczywistości wyrażeniem **Plus**[a,b,c], a {a,b,c} jest w rzeczywistości wyrażeniem **List**[a,b,c]
- łatwo więc zmienić operację wykonywaną na danej liście argumentów: **Apply**[**Plus**, {1,2,c}] zamieni **List**[1,2,c] na **Plus**[1,2,c], czyli 3+c
- **FullForm**[wyrażenie] daje jawną postać wyrażenia, np. **FullForm**[1+2+c]

Wyrażenia

Proste symbole, liczby (całkowite, wymierne, rzeczywiste (dziesiętne), zespolone), teksty

Złożone pozostałe; ekstrakcja fragmentu wyrażenia: **Part**[{a,b,c}, 3] lub równoważnie {a,b,c}[[3]]; czym jest {a,b,c}[[0]]?

Head[wyrażenie] to (prawie) to samo, co **Part**[wyrażenie, 0]; dla wyrażenia prostego zwraca typ wyrażenia

wyznaczanie wartości wyrażeń: *term rewriting*,

- zgodnie z zaszytymi w systemie regułami
- według reguł opisanych przez użytkownika („funkcje”)

czasem warto w systemie odłożyć na później wyznaczanie wartości wyrażenia (**Hold**, **Unevaluated**, itd.)

Funkcje

Dwa rodzaje funkcji: o działaniu natychmiastowym **Set**, tzn. = oraz działaniu odwleczonym **SetDelayed**, tzn. :=

- **Set** to praktycznie znana nam wersja operatora przypisania; A = {a,b,c}; **Apply**[**Plus**,A]
- **SetDelayed** bardziej odpowiada naszemu pojęciu „funkcji”: MyFun[x_] := **Sin**[x^2]^3, lub MyFunFun[x_,p_] := **Sin**[x^2]^p_
- _ (*blank*) oznacza „dowolne wyrażenie”, a x_ nadaje nazwę temu dowolnemu wyrażeniu — do wykorzystania w definicji funkcji
- różnica: x = 5; MyFun[x_] := **Sin**[x^2]^3 kontra x = 5; MyFun[x_] = **Sin**[x^2]^3 (ostatnie podstawi „5” pod x...)
- __ (*double blank*) oznacza „niepustą sekwencję dowolnych wyrażeń”; ___ (*triple blank*) oznacza „dowolną (także pustą) sekwencję dowolnych wyrażeń”; przyrostek dopisany do dowolnego blanka ogranicza go do wyrażenia z nim jako nagłówkiem

Funkcje — c.d.

- zmienne lokalne w funkcji są globalne: MyFun[x_] := (y = x^2; **Sin**[y]^3); aby ograniczyć ich zasięg: MyFun[x_] := **Module**[{y}, (y = x^2; **Sin**[y]^3)]
- można formułować reguły (tzn. funkcje) tylko dla specjalnych typów argumentów, itp.
- są też funkcje anonimowe (*pure*) oraz tymczasowe

Reguły tymczasowe dla wyrażeń

Czasem nie chcemy tworzyć specjalnej funkcji, tylko zastosować pewną regułę do konkretnego wyrażenia.

- reguła natychmiastowa: wyrażenie /. co -> naco, np. {(a+3)^2,b,c} /. a -> 7 („naco” jest wyznaczane jednorazowo)
- reguła odwleczona: wyrażenie /. co :> naco, np. {x,x,x} /. x -> RandomReal[] kontra {x,x,x} /. x :> RandomReal[]

Funkcje przykładające funkcje

- **Apply**[f, wyrażenie] zastępuje funkcję nagłówkową wyrażenia funkcją f. Skrótowo: f @@ wyrażenie
- **Map**[f, lista] przykłada f do kolejnych elementów listy. Skrótowo: f /@ lista
- **Nest**[f, wyrażenie, n], wyznacza f^n(wyrażenie) (ma różne warianty, np. **FixedPoint**, **Fold**, **NestList**, **NestList**, **FoldList**, **FixedPointList**)