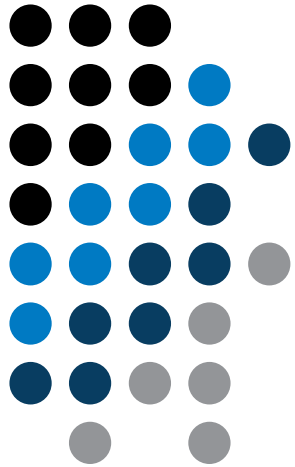
A0B17MTB – Matlab

Část #4

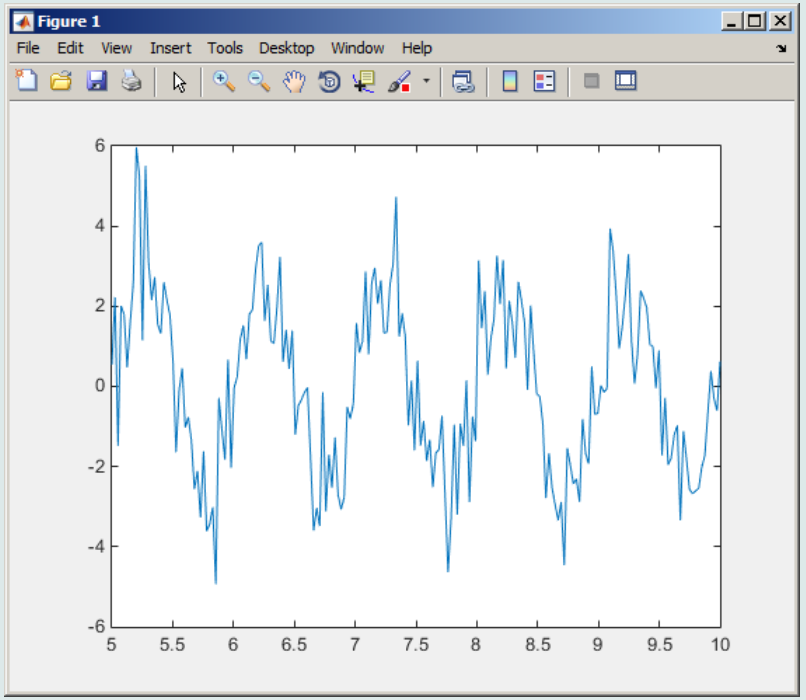
Miloslav Čapek
miloslav.capek@fel.cvut.cz

Filip Kozák, Viktor Adler

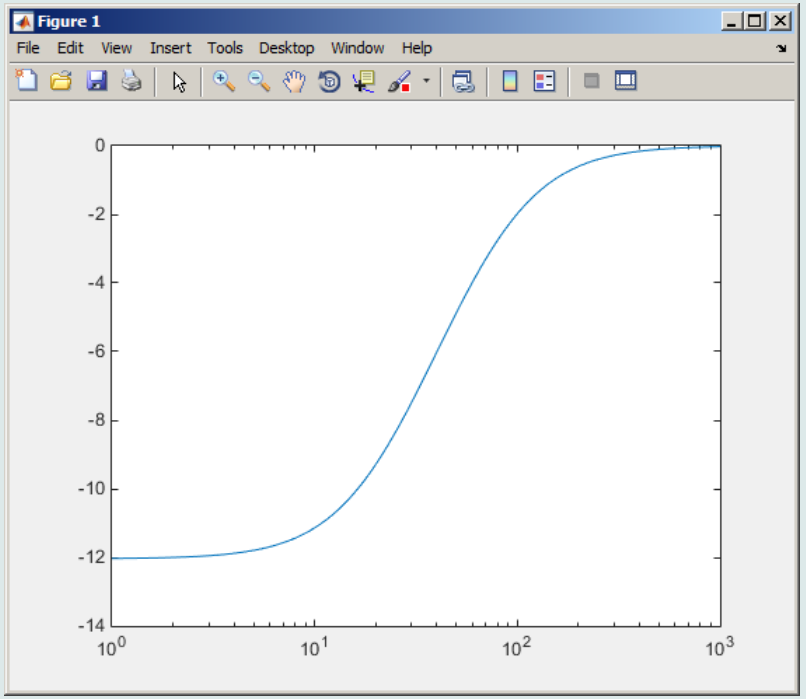
Katedra elektromagnetického pole
B2-626, Dejvice



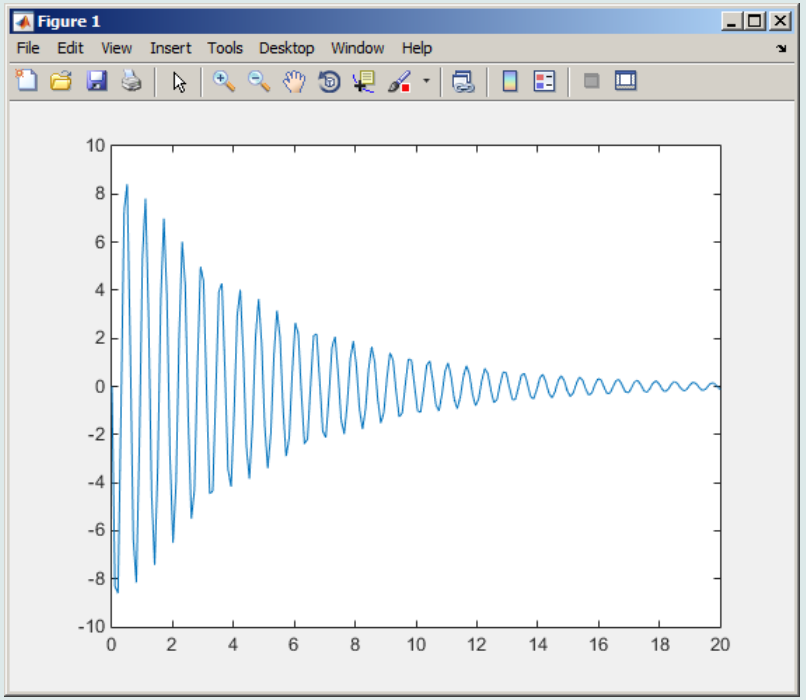
Řešení příkladu #3 z minulé hodiny



Řešení příkladu #5 z minulé hodiny



Řešení příkladu #7 z minulé hodiny

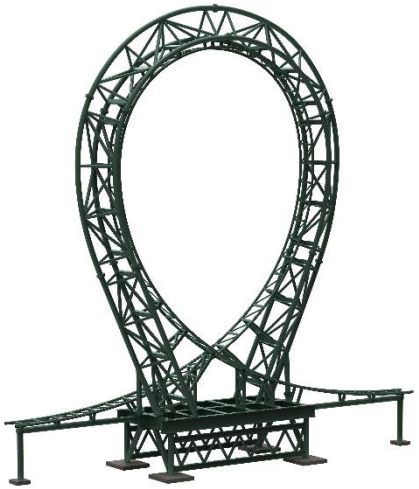


Naučíte se …

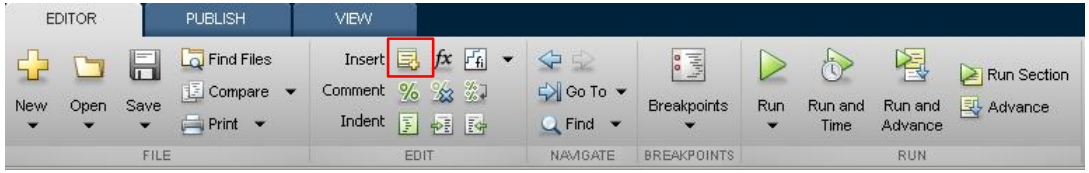
Relační a logické opetárory

Cykly

Větvení programu #1



Cell režim v Matlab Editoru



- celly umožňují dělit kód do menších, logicky ucelených částí
  - oddělovač: `%%`
  - oddělení je vizuální, ale je možné i vyhodnotit (spustit) jediný cell - zkratka CTRL+ENTER
- ve starších verzích Matlabu je cell režim zpravidla nutné aktivovat

Cell režim v Matlab Editoru

240 s ↑

- rozdělte předchozí skript (`loanRepayment.m`) na funkční části
  - využijte oddělovače `%%` (cellů)

Data ve skriptu

- skript může využívat data, která ve Workspace již byla
- proměnné zůstávají v pracovním prostoru i po skončení výpočtu
- s daty ve skriptu je operováno v základním pracovním prostoru Matlabu, viz Matlab Workspace

Jména skriptů a funkcí

- jména skriptů a funkcí
  - max. počet znaků v názvu je 63 (další znaky jsou ignorovány)
  - jména jsou omezeny podobně jako názvy proměnných
  - volte názvy, které přímo evokují, co daný skript / funkce počítá
  - vyhýbejte se existujícím názvům, neboť pak je skript / funkce volán(a) na úkor funkce zabudované (může docházet k tzv. přetěžování)
- více informací:
  - `http://www.mathworks.com/matlabcentral/fileexchange/2529-matlab-programming-style-guidelines`

- chcete-li aplikovat vektorovou funkci na řádky
  - podívejte se, zda funkce neumožňuje výpočet přes jinou dimenzi (`max`)
  - transponujte zdrojovou matici
  - některé funkce mají i „řádkovou“ mutaci (`sort` × `sortrows`)

Skript `startup.m`

- skript `startup.m`
  - je vždy proveden při spuštění Matlabu
  - lze sem umístit Vaše předdefinované konstanty a další operace, které se provedou (načtou) při startu Matlabu

- umístění (použijte `>> which startup`):
  - `...\Matlab\R201Xx\toolbox\local\startup.m`

- změna základního adresáře po startu Matlabu:

```
%% script startup.m in ..\Matlab\Rxxx\toolbox\local\
clc;
disp('Workspace is changing to:');
cd('d:\Data\Matlab\');
cd
disp(datestr(now, 'mmmm dd, yyyy HH:MM:SS.FFF AM'));
```

```
Workspace is changing to:

d:\Data\Matlab

February 25, 2014  3:36:03.347 PM
Keep on working...
>>
```

Skript `matlabrc.m`

- spuštěn při startu Matlabu (nebo lze spustit: `>> matlabrc`)
- obsahuje některé základní definice, např.
  - velikosti obrázků, nastavení některých grafických prvků
  - nastavuje Matlab path (viz později)
  - a další
- v případě multilicence lze do skriptu vložit zprávu, který se při startu zobrazí všem uživatelům
- umístění (použijte `>> which matlabrc`):
  - `...\Matlab\R201Xx\toolbox\local\matlabrc.m`

- na závěr volá skript `startup.m` (pokud ten existuje)

- skript `matlabrc.m` modifikovat jen v případě naprosté nutnosti!

Relační operátory

- ptáme se, porovnáváme, zda je něco větší, menší, rovno atp.
- výsledek porovnání je vždy buďto
  - pravda (`true`), v Matlabu jako logická jednička „1“
  - nepravda (`false`), v Matlabu jako logická nula „0“

>	větší než
>=	větší nebo rovno
<	menší
<=	menší nebo rovno
==	rovno
~=	není rovno

- všechny relační operátory lze použít s vektorizací
  - můžeme porovnávat i vektory vs. vektory, matice vs. matice, …
- často kombinujeme s logickými operátory (viz dále)
  - klademe tak na více výrazů více relačních podmínek

Relační operátory

300 s ↑

- pro vektor $\mathbf{G} = \begin{pmatrix} \frac{\pi}{2} & \pi & \frac{3}{2}\pi & 2\pi \end{pmatrix}$ určete ty členy, které jsou
  - větší než π
  - menší nebo rovny π
  - nejsou rovny π
- zkuste podobné operace i pro $\mathbf{H} = \mathbf{G}^{\mathrm{T}}$
- zkuste použít relační operátory i v případě matice a skaláru
- porovnejte, zda je $\mathbf{V} \geq \mathbf{U}$:

$$\mathbf{V} = \begin{pmatrix} -\pi & \pi & 1 & 0 \end{pmatrix}$$

$$\mathbf{U} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$$

Relační operátory

200 s ↑

- zjistěte výsledky následujících porovnání
  - pokuste se interpretovat výsledky

```
>> 2 > 1 & 0 % ???
```

```
>> r = 1/2;
>> 0 < r < 1 % ???
```

```
>> (1 > A) <= true
```

Logické operátory

- ptáme se, zjišťujeme, zda je nějaká podmínka splněna
- výsledek je vždy buďto
  - pravda (`true`), v Matlabu jako logická jednička „1“
  - nepravda (`false`), v Matlabu jako logická nula „0“

`&`	`and`
`|`	`or`
`~`	`not`
	`xor`
	`all`
	`any`

- `all`, `any` využijeme při konverzi logického pole na skalár

- Matlab interpretuje každou `numeric` hodnotu krom `0` jako `true`
- všechny relační operátory lze použít s vektorizací
  - můžeme porovnávat i vektory vs. vektory, matice vs. matice, …

- funkce `is*` rozšiřují možnosti logického dotazování
  - probereme později

Logické operátory – příklady užití

- předpokládejme vektor 10 čísel náhodně generovaných mezi -10 a 10

```
>> a = 20*rand(10, 1) - 10
```

  - příkaz vrátí logickou hodnotu `true` pro prvky, které podmínky splňují:

```
>> a < -5 % relační oper.
```

  - příkaz vrátí prvky, které splňují podmínku (logická indexace):

```
>> a(a < -5)
```

  - příkaz přepíše hodnoty těch prvků, které splňují podmínku na -5:

```
>> a(a < -5) = -5
```

  - příkaz přepíše hodnoty prvků mezi -5 a 5 na 0 (opak prahování):

```
>> a(a > -5 & a < 5) = 0
```

  - prahovací funkce (hodnoty pod -5 nastaví na -5 pro hodnoty nad 5 na 5):

```
>> a(a < -5 | a > 5) = sign(a(a < -5 | a > 5))*5
```

Logické operátory

420 s ↑

- určete prvky z vektoru $\mathbf{A} = \left(\frac{\pi}{2} \quad \pi \quad \frac{3}{2}\pi \quad 2\pi \right)$, které
  - se rovnají π nebo se rovnají 2π
    - všimněte si, jaké hodnoty se Vám vrací (= logické hodnoty `true` / `false`)
  - jsou větší než π/2 a současně se nerovnají 2π
  - prvky z předchozího bodu přidejte do matice A

Logické operátory: &&, ||

- pokud potřebujeme porovnávat pouze skalární hodnoty, můžeme s výhodou využít zrychleného vyhodnocování

- vyhodnocování pokračuje pouze tak dlouho, dokud to má smysl
  - tj. pokud porovnáváme např.

```
>> clear; clc;
>> a = true;
>> b = false;
>> a && b && c && d
```

… nemáme problém s neznámými `c`, `d`, neboť příkaz je ukončen dříve

- avšak:
  - již končí chybou …

```
>> clear; clc;
>> a = true;
>> b = true;
>> a && b && c && d
```

Logické operátory

150 s ↑

- vytvořte řádkový vektor v intervalu 1 až 20 s krokem 3
  - do nového vektoru vyberte prvky větší než 10 a zároveň prvky menší nebo rovny 16 pomocí logických operátorů (tj. přímý výběr prvků z vektoru)

Logické operátory

240 s ↑

- vytvořte matici `A = magic(3)` a pomocí funkcí `all` a `any` zjistěte
  - které sloupce mají <u>všechny</u> členy větší než 2
  - které řádky mají <u>alespoň jeden</u> člen větší nebo roven 8
  - zda matice `A` obsahuje <u>pouze</u> kladná čísla

$$\mathbf{A} = \begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix}$$

$$\text{any}\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix},\quad \text{all}\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix},\quad \text{any}\left(\text{all}\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}\right) = \text{any}\begin{pmatrix} 0 & 1 & 0 \end{pmatrix} = 1$$

Logické operátory

240 s ↑

- zjistěte výsledky následujícího porovnání a interpretujte výsledek

```
>> ~(~[1 2 0 -2 0])
```

- ověřte zda *b* není rovno nule a poté ověřte zda současně $a / b > 3$
  - testujte, zda jsou obě podmínky splněny a vyhněte se dělení nulou!

Indexace matice vlastními hodnotami

300 s ↑

- vytvořte si matici `A`

```
>> N = 4;
>> A = magic(N)
```

```
A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

- nejprve se zamyslete, co bude výsledkem následující operace, a až poté ji realizujte

```
>> B = A(A)
```

  - je skutečně výsledkem, co jste očekávali?
  - umíte případně vysvětlit proč výsledek vypadá tak, jak vypadá?
  - všimněte si případně zajímavých matematických vlastností matice `A`, i `B`
  - umíte odhadnout co se stane dál, `C = B(B)`

- zkuste obdobný postup pro `N = 3` nebo `N = 5`

Větvení programu – cykly

- opakování určité činnosti vícekrát, jde o jednu ze základních struktur
- v Matlabu známe 2 druhy cyklů:
  - `for` – nejčastěji využíván, dopředu známe počet opakování
  - `while` – známe podmínku, za které (ne)pokračovat v cyklu

- zcela zásadní je dodržovat programátorské zásady:
  - alokujeme prostor (matice) dostatečné velikosti /více dále.../
  - cykly by měly být řádně ukončeny /více dále.../
  - u `while` cyklu zajistíme splnitelnou podmínku /více dále.../
- mnohdy stačí vhodně upravit pole (1D → 2D, 2D → 3D pomocí funkce `repmat` a operaci provést maticově, za jistých podmínek je vektorizovaný kód rychlejší i přehlednější, lze využít např. na GPU)

- vždy si položíme otázku: Je cyklus skutečně nezbytný?

Cyklus `for`

- pro předem daný počet opakování skupiny příkazů volíme `for`:

```
for m = vyraz
    prikazy
end
```

- `vyraz` je vektor / matice; sloupce tohoto vektoru / matice jsou postupně přiřazovány proměnné `m` / `n`

```
for n = 1:4
    n
end
```

```
for m = magic(4)
    m
end
```

- často `vyraz` generujeme pomocí funkcí `linspace`, nebo pomocí „`:`“, užíváme rovněž funkce `length`, `size`, atp.

- místo `m` můžeme psát výstižněji `mPoints`, `mRows`, `mSymbols`, …
  - pro přehlednost je vhodné užívat např. `mXX` pro řádky a `nXX` pro sloupce

Cykly #1

400 s ↑

- vypracujte skript, který vypočítá faktoriál *N*!
  - využijte cyklu, výsledek ověřte pomocí funkce Matlabu pro faktoriál

```
>> factorial(N)
```

  - napadne Vás i jiné řešení? (např. pomocí vektorizace…)
  - porovnejte všechny možnosti i pro neceločíselný argument *N*

Alokace proměnných

- alokací předejdete neustálému navyšování velikosti proměnné
  - na vhodnost alokace proměnných Vás v Matlab Editoru upozorní Code Analyser (M-Lint) podtržením dané matice
  - vždy, když víte, jak má být daná proměnná veliká, alokujte!
    - někdy se vyplatí alokovat i když konečnou velikost neznáme – pak alokujeme matici o velikosti „nejhoršího případu“ a pak ji zmenšíme
    - alokujte vždy nejprve ty největší proměnné, a až poté ty menší
- příklad:
  - vyzkoušejte si…

```
%% BEZ alokace
tic;
for m = 1:1e7
    A(m) = m + m;
end
toc;
% computed in 0.45s
```

```
%% S alokací
tic;
A = zeros(1,1e7);
for m = 1:1e7
    A(m) = m + m;
end
toc;
% computed in 0.06s
```

Cyklus `while`

- opakuje příkazy v těle cyklu (`prikazy`) v závislosti na logické podmínce

```
while podminka
    prikazy
end
```

- příkazy se opakují tak dlouho, dokud jsou <u>všechny</u> prvky ve výrazu (`podminka` může být i vícerozměrná matice) <u>nenulové</u>
  - podmínka je převedena na relační výraz, tj. dokud jsou všechny prvky `true`
  - pro testování podmínky často užíváme logické a relační operátory

- pokud `podminka` není skalár, můžeme ho redukovat pomocí funkcí `any` nebo `all`

Typické užití cyklů

```
%% script generates N experiments with M throws with a die
close all; clear all; clc;

Mthrows  = 1e3;
Ntimes   = 1e2;
Results  = NaN(Mthrows, Ntimes);
for mThrow = 1:Mthrows % however, can be even further vectorized!
    Results(mThrow, :) = round(rand(1, Ntimes)); % vectorized
end
```

```
%% script finds out the number of lines in a targed file
fileName = 'sin.m';
fid = fopen(fileName, 'r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    count = count + 1;
end
disp(['lines:' num2str(count)])
fclose(fid);
```

Cykly #2

360 s ↑

- pomocí cyklu `while` vypočtěte součet celých čísel od 1 do 100
  - úlohu si libovolně modifikujte, ale využijte `while` cyklus

- napadne Vás jiné řešení (pomocí funkce Matlabu a bez cyklu)?

Cyklus `while` – nekonečná smyčka

- pozor na podmínky v cyklu `while`, které jsou vždy splněny
  ⇒ hrozí nekonečná smyčka
  - ve většině, nikoliv však ve všech (!!), případech jde o sémantickou chybu

- triviální, avšak výstižný příklad …

```
while 1 == 1
    disp('ok');
end
```

```
while true
    disp('ok');
end
```

… který „nikdy“ neskončí (zkratka pro ukončení CTRL+C)

Záměna indexu za komplexní jednotku

- pozor na záměnu komplexní jednotky (`i`, `j`) s indexem cyklu
  - snažte se užití `i` a `j` jako indexů vyhnout
    - dochází k přetěžování (platí obecně, např. `>> sum = 2` přetíží funkci `sum`)

- určete rozdíl v následujících příkladech:

```
A = 0;
for i = 1:10
    A = A + 1i;
end
```

```
A = 0;
for i = 1:10
    A = A + i;
end
```

```
A = 0;
for i = 1:10
    A = A + j;
end
```

- všechny příkazy lze v principu zapsat na jednu řádku

```
A = 0; for i = 1:10, A = A + 1i; end,
```

  - většinou hůře pochopitelné, není vhodné ani z pohledu rychlosti kódu

Zanořené cykly, kombinování cyklů

- často potřebujeme cykly zanořit do sebe
  - zvažte vektorizaci
  - zvažte typ cyklu

- zanořování cyklů zpravidla rapidně zvyšuje výpočetní náročnost

```
%% script generates N experiments with M throws with a die
close all; clear all; clc;

Mthrows  = 1e3;
Ntimes   = 1e2;
Results  = NaN(Mthrows, Ntimes);
for mThrow = 1:Mthrows
    for nExperiment = 1:Ntimes % not vectorized (30 times slower!!)
        Results(mThrow, nExperiment) = round(rand(1));
    end
end
```

Cykly #3

600 s ↑

- pomocí cyklu (cyklů) naplňte matici $\mathbf{A}(m,n) = \frac{mn}{4} + \frac{m}{2n}$
- uvažujte $m \in \{1,\ldots,100\}$, $n \in \{1,\ldots,20\}$, matici před výpočtem alokujte
- vytvořte pro daný účel skript, využijte cykly

- pro vykreslení matice **A** můžete využít např. funkce `pcolor()`

Cykly #4

600 s ↑

- zobrazte rozložení proudu na dipólové anténě ve tvaru

$$\mathrm{I}(x,t) = \mathrm{I}_0(x)\mathrm{e}^{-\mathrm{j}\omega_0 t}, \quad \mathrm{I}_0(x) = \cos(x), \quad \omega_0 = 2\pi$$

- na intervalech $t \in (0, 4\pi)$, $x \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$ volte N = 101 prvků

pro vykreslení uvnitř těla cyklu využijte příkazů:

```
% ... Váš kód
    figure(1);
    plot(x,real(I));
    axis([x(1) x(end) -1 1]);
    pause(0.1);
% ... Váš kód
```

Cykly #5

600 s ↑

- zkuste si naprogramovat průměrování klouzavým oknem pro funkci

$$f(x) = \sin^2(x)\cos(x) + 0.1r(x),$$

kde $r(x)$ je reprezentována funkcí rovnoměrného rozložení (`rand()`)

  - využijte následujících parametrů

```
clear; clc;
signalSize = 1e3;
x  = linspace(0, 4*pi, signalSize);
f  = sin(x).^2.*cos(x) + 0.1*rand(1, signalSize);
windowSize = 50;
% Váš kód ...
```

  - a poté vykreslete:

```
plot(x, f, x, my_averaged);
```

  - zkuste kód dále zefektivnit

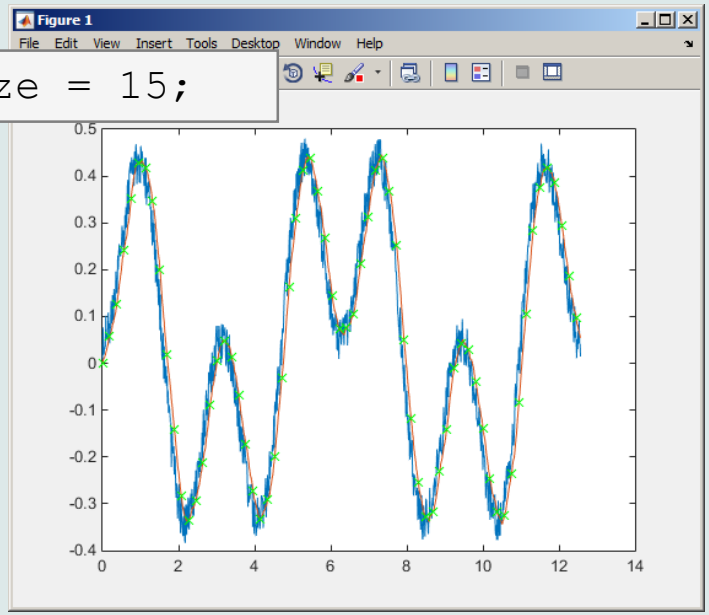
Cykly #6

600 s ↑

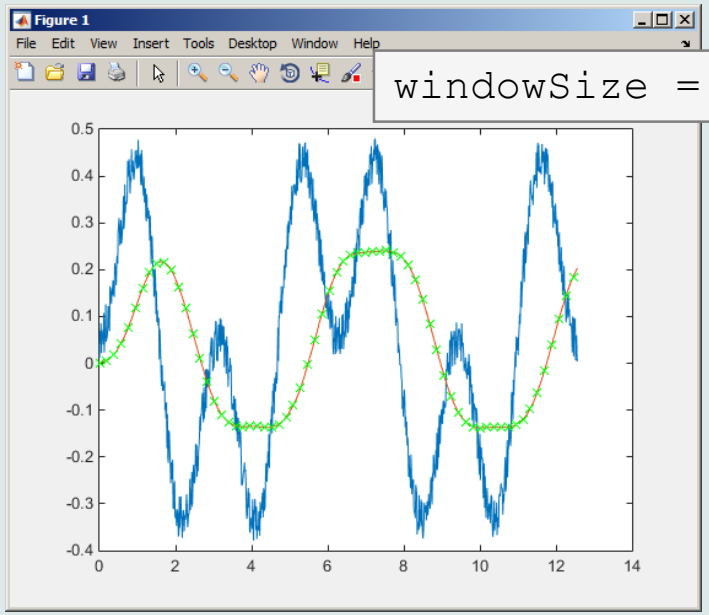
- pro porovnání lze využít funkce `filter`, kterou Matlab obsahuje

- zkuste se podívat, jak ovlivní nastavení `windowSize` výsledek

```
windowSize = 15;
```




```
windowSize = 150;
```



break, continue

- funkce `break` umožní vyskočit ze smyčky předčasně

```
% another code ...
for k = 1:length(A)
    if A(k) > threshold
        break;
    end
    % another code ...
end
```

if (true)

- funkce `continue` přeskočí na další iteraci cyklu

```
% another code ...
for k = 1:length(A)
    if A(k) > threshold
        continue;
    end
    % another code ...
end
```

if (true)

Použití cyklů vs. vektorizace #1

- od verze Matlabu 6.5 máme k dispozici dvě mocné skryté technologie
  - *Just-In-Time accelerator* (JIT akcelerátor)
  - *Real-Time Type Analysis* (RTTA)

- JIT umožňuje částečnou kompilaci segmentů kódu
  - předkompilované cykly jsou rychlejší i než vektorizace
  - v cyklech je mj. nutné dodržet zásady:
    - cyklus `for` se opakuje přes skalární index
    - uvnitř `for` cyklu jsou volány pouze built-in funkce
    - cyklus pracuje typicky pouze se skalárními hodnotami

- RTTA předpokládá stejné datové typy jako při předchozím běhu programu – znatelné urychlení pro typizované výpočty
  - při měření rychlosti kódu je potřeba provést tzv. warm-up (pustíme nejprve kód 2-3× a poté měříme rychlost)

Použití cyklů vs. vektorizace #2

- motivací vzniku JIT bylo vyrovnat se rychlostně jazykům 3. generace
  - pokud je JIT plně využit, je čas výpočtu srovnatelný s C nebo Fortranem
- nejvyšší účinnost (největší dosažené zrychlení) zejm. pokud
  - cykly operují na skalárními daty
  - nejsou volány funkce napsané uživatelem (resp. jsou volány pouze build-in funkce)
  - každá řádka cyklu využívá JIT
- výsledkem je, že některé segmenty kód již není potřeba vektorizovat (nebo by se dokonce vektorizovat na úkor cyklů ani neměly!)
- uvedené téma je výrazně složitější (zde zjednodušeno)
  - detaily na stránce předmětu v dokumentu `JIT_accel_Matlab.pdf`

Použití cyklů vs. vektorizace

- předchozí tvrzení ověříme na jednoduchém kódu – naplnění pásové matice
- podmínky pro využití JIT jsou splněny…
  - práce se skaláry, voláme pouze built-in funkce
- naplnění matice pomocí `for` cyklů je rychlejší!
  - vyzkoušejte si…

```
clear; clc;
N   = 5e3;

tic,
mat = diag(ones(N, 1)) + ...
    2*diag(ones(N-1, 1), 1) + ...
    3*diag(ones(N-1, 1), -1);
toc,
% computed in 0.411
```

```
clear; clc;
N   = 5e3;
mat = NaN(N, N);

tic,
for n1=1:N
    for n2=1:N
        mat(n1, n2)=0;
    end
end
for n1 = 1:N
    mat(n1, n1)=1;
end
for n1 = 1:(N-1)
    mat(n1, n1+1)=2;
end
for n1 = 2:N
    mat(n1, n1-1)=3;
end
toc,
% computed in 0.331
```

Větvení programu

- potřebujeme-li větvit program (provést určitou část pouze za dané podmínky), máme k dispozici dvě základní konstrukce:
  - `if – elseif – else – end`
  - `switch – case – otherwise – end`

```
if podminka
    prikazy
elseif podminka
    prikazy
elseif podminka
    prikazy
else
    prikazy
end
```

```
switch promenna
 case hodnota1
    prikazy
 case {hodnota2a, hodnota2b, ...}
    prikazy
 case ...
    prikazy
 otherwise
    prikazy
end
```

`if` vs. `switch`

`if-elseif-else-end`	`switch-otherwise-end`
Lze vytvořit i velmi složitou strukturu (`&&` / `||`)	Snáze dělíme více variant
Je potřeba `strcmp` pro porovnání textových řetězců různých délek	Přímo testuje textové řetězce
Testuje rovnost / nerovnost	Pouze test na rovnost
Pro test mnoha možností je potřeba mnoho logických výrazů	Pomocí `{}` umožňuje elegantně testovat na jednu z mnoha možností

Větvení programu – `if` / `else` / `elseif`

- přímo za `if` by měla následovat nejpravděpodobnější možnost
- pouze `if` větev je povinná
- část za `else` je provedena pouze v případě, že ostatní podmínky nejsou splněny

- pokud je součástí podmínky matice typu M×N, je podmínka pravdivá pouze tehdy, pokud je pravdivá pro všechny prvky

- podmínka může obsahovat volání funkcí atp.
- lze vnořovat více `if` podmínek do sebe

```
c = randi(1e2);
if mod(c, 2)
    disp('c je liche');
elseif c > 10
    disp('sude, >10');
end
```

Větvení programu – `if` / `else` / `elseif`

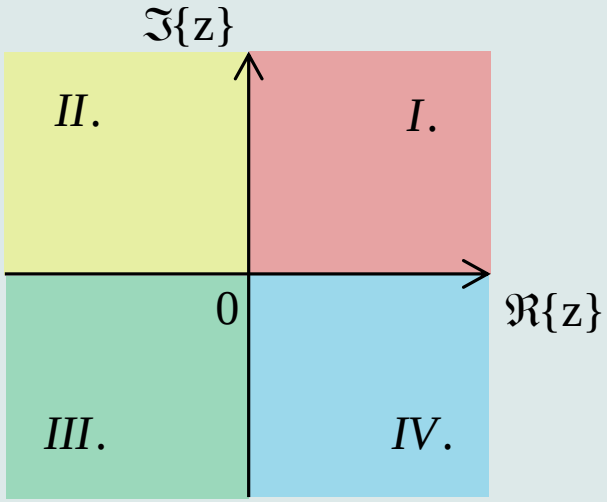
400 s ↑

- vygenerujte si čísla `r = 2*rand(8, 1)-1;`
- podle toho, zda jsou záporné nebo kladné je uložte buďto do matice `Neq` nebo `Pos`, užijte `for` cyklus, `if-else` a pro uložení indexujte
  - pozor na narůst velikosti matic `Pos` a `Neq` – jak problém vyřešíte?
- napadne Vás elegantnější řešení? (ne vždy potřebujeme cyklus)

Větvení programu – `if` / `else` / `elseif`

500 s ↑

- napište skript, který určí do kterého kvadrantu patří komplexní číslo, které si sami vygenerujete



Probrané funkce

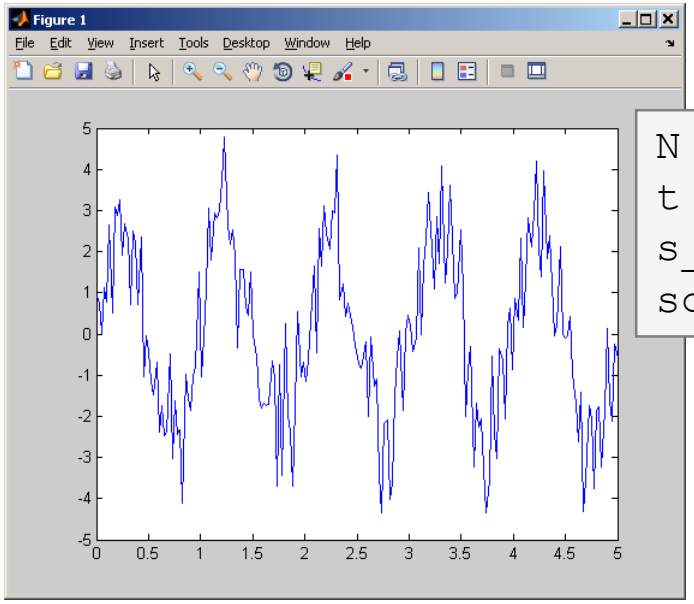
`edit`	otevře Matlab Editor	●
`disp, pause`	vypíše výsledek do příkazové řádky, pozastaví vyhodnocování kódu	●
`num2str`	převod z datového typu `numeric` na `char`	●
`for-end, while-end`	cykly	●
`factorial`	vypočte faktoriál	
`break, continue`	ukončí zpracovávání cyklu, přeskočí na další iteraci	
`and, or, not, xor`	funkce přetěžující logické operátory	
`all, any`	vyhodnocování logických polí („pro všechny z“, „alespoň jeden z“)	
`sign`	znaménková funkce	
`if-elseif-else-end`	podmínka	●

Cvičení #1

360 s ↑

- vzpomeňte si na náš signál z 2. části materiálů
  - signál se opět pokuste prahovat k hodnotám, $s_{\min}$ a $s_{\max}$
  - místo funkcí (`max`, `min`) využijte relačních operátorů (> / <) a logického indexování (`s(a>b) = c`)
    - klidně úlohu řešte postupně

$$s_p(t) = \begin{cases} s_{\min} \Leftrightarrow s(t) < s_{\min} \\ s_{\max} \Leftrightarrow s(t) > s_{\max} \\ s(t) \ldots \text{ jinak} \end{cases} \qquad s_{\min} = -\frac{9}{10} \qquad s_{\max} = \frac{\pi}{2}$$

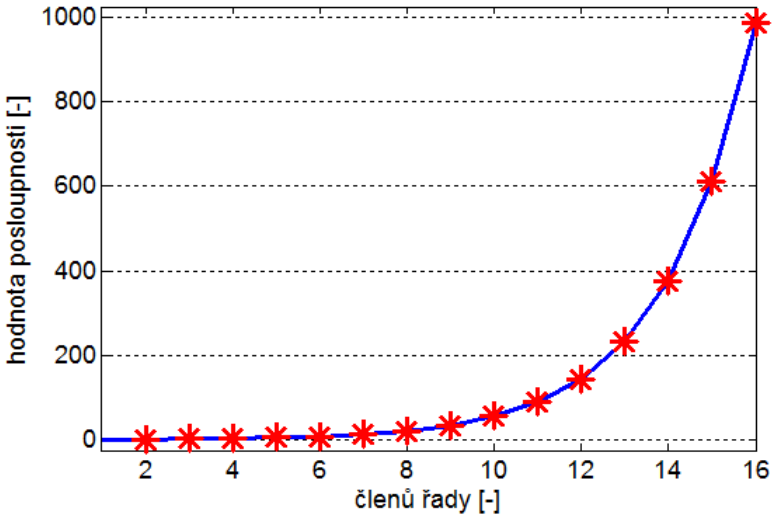



```
N = 5; V = 40;
t = linspace(0, N, N*V);
s_t = randn(1, N*V) + ...
sqrt(2*pi)*sin(2*pi*t);
```

Cvičení #2

600 s ↑

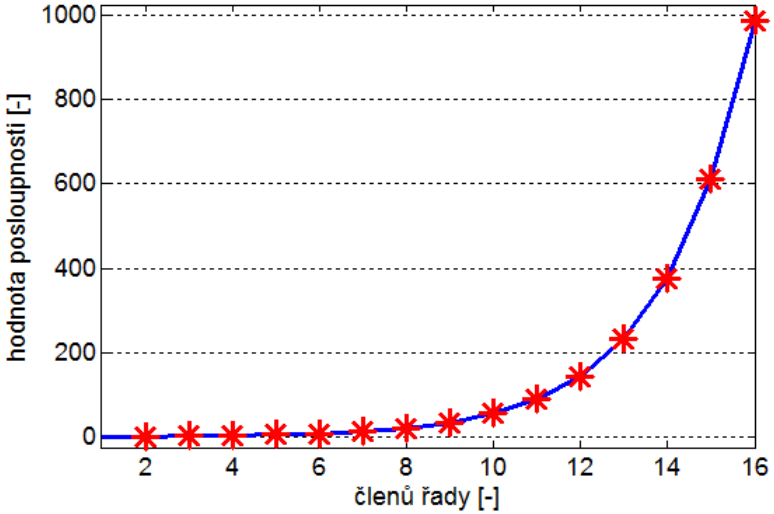
- navrhněte skript, který vypočítá hodnoty Fibonacciho posloupnosti do určité hodnoty `limit`
  - setkali jste se již s touto posloupností?
  - pokud ne, najděte její předpis
  - implementace:
    - jaký použijete cyklus (pokud nějaký)?
    - které matice / vektory alokujete?
  - výslednou posloupnost vykreslete pomocí funkce `plot`



Cvičení #3

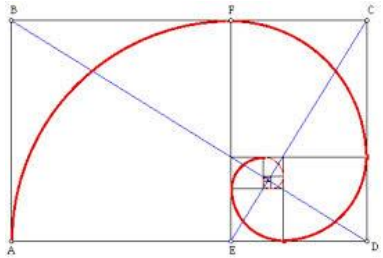
240 s ↑

- takhle rychle se množí králíci:



- zkuste zjistit, jak řada souvisí s hodnotou zlatého řezu
- pokuste se ho vypočítat:

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618033\ldots$$



Cvičení #4

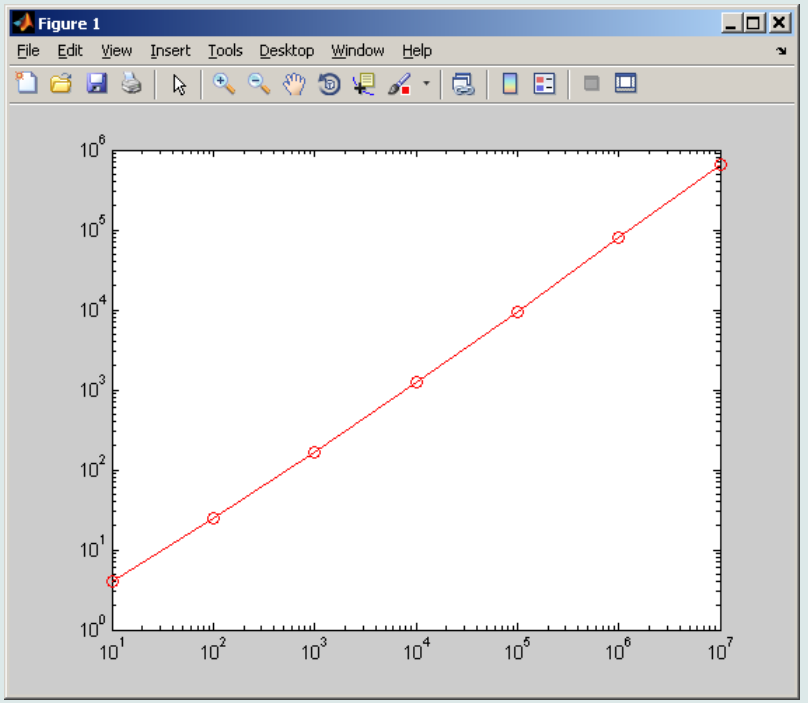
300 s ↑

- mějme následující matici: $\mathbf{A} = \begin{pmatrix} 1 & 1 & 2 \\ 2 & 3 & 5 \end{pmatrix}$
- napište podmínku, která otestuje, zda jsou všechny prvky matice **A** kladné a zároveň všechny prvky v 1. řádku celočíselné?
  - pokud ano, vypište pomocí `disp` oznámení
- porovnejte s
  - jaký je v příkazech rozdíl?

Cvičení #5

600 s ↑

- pokuste se určit hustotu prvočísel
  - pro generování prvočísel si prostudujte funkci `primes`
  - pro řády 10^1 – 10^7 určete hustotu prvočísel (tj. počet prvočísel do čísla 10, do čísla 100, …, do čísla 10^7)
  - vykreslete závislosti pomocí funkce `plot`
  - vykreslete průběh v logaritmickém měřítku (funkce `loglog`)
    - jak se změní průběh grafu?



Cvičení #6

- využili jste cyklus?
- je zde výhodný (nezbytný)?
- alokujete si prázdné matice?
- čím je podle Vás dominantně ovlivněna doba výpočtu?

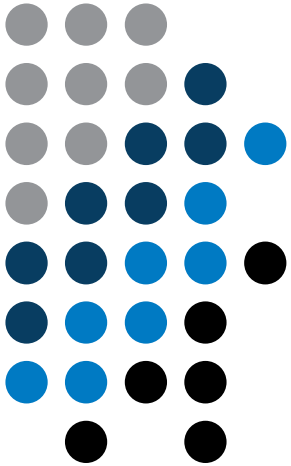
Cvičení #7

- skript lze dále zrychlit
  - funkce `primes`, která je časově „drahá“ je počítána pouze 1×

- dokázali byste skript zrychlit ještě více?

Děkuji!

ver. 3.4 (09/03/2015)
Miloslav Čapek
miloslav.capek@fel.cvut.cz



Materiál vytvořen v rámci předmětu A0B17MTB.