A0B17MTB – Matlab

Part #5

Miloslav Čapek
miloslav.capek@fel.cvut.cz

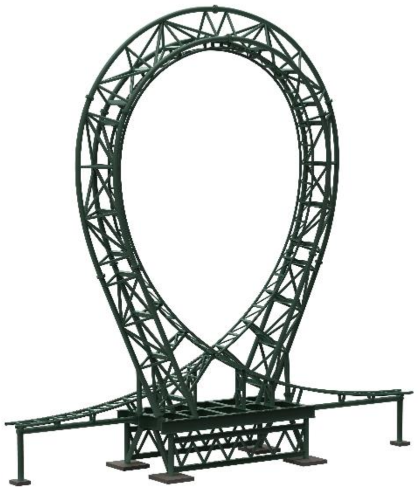
Filip Kozák, Viktor Adler, Pavel Valtr

Department of Electromagnetic Field
B2-626, Prague

Learning how to …

Loops

Program branching



Program branching – loops

- repeating certain operation multiple-times, one of the basic programming techniques
- There are 2 types of cycles in Matlab:
  - `for` – the most used one, number of repetitions is known in advance
  - `while` – condition is known ensuring cycle (dis)continuation as long as it remains true
- essential programing principles to be observed:
  - memory allocation (matrix-related) of sufficient size /see later.../
  - cycles should be properly terminated /see later.../
  - to ensure terminating condition with `while` cycle /see later.../
- frequently is possible to modify the array (1D → 2D, 2D → 3D using function `repmat` and carry out a matrix-wise operation, under certain conditions the vectorized code is faster and more understandable, possibility of utilization of GPU)
- we always ask the question: <u>is a cycle really necessary?</u>

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

`for` loop

- `for` loop is applied to <u>known number of repetitions</u> of a group of commands

```
for m = expression
    commands
end
```

- `expression` is a vector / matrix; columns of this vector / matrix are successively assigned to `m` / `n`

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

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

- frequently, `expression` is generated using `linspace` or using „:“, with the help of `length`, `size`, `numel`, etc.

- instead of `m` it is possible to use more relevant names like `mPoints`, `mRows`, `mSymbols`, …
  - for clarity, it is suitable to use e.g. `mXX` for rows and `nXX` for columns

Loops #1

400 s

- create a script to calculate factorial N!
  - use a cycle, verify your result using Matlab `factorial` function

```
%% script calculates factorial of N
close all; clear; clc;
...
...
...
...
...
...
```

  - can you come up with other solutions? (e.g. using vectorising…)

  - compare all possibilities for decimal input N as well

Memory allocation

- allocation can prevent perpetual increase of the size of a variable
  - Code Analyser (M-Lint) will notify you about the possibility of allocation by underlining the matrix's name
  - whenever you know the size of a variable, allocate!
    - sometimes, it pays off to allocate even when the final size is not known - then the worst-case scenario size of a matrix is allocated and then the size of the matrix is reduced
    - allocate the variables of the largest size first, then the smaller ones
- example:
  - try…

```
%% WITHOUT allocation
tic;
for m = 1:1e7
    A(m) = m + m;
end
toc;
% computed in 0.45s
```

```
%% WITH allocation
tic;
A = nan(1,1e7);
for m = 1:1e7
    A(m) = m + m;
end
toc;
% computed in 0.06s
```


Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

while loop

- keeps on executing commands contained in the body of the cycle (`commands`) depending on a logical condition

```
while condition
    commands
end
```

- keeps on executing commands as long as <u>all</u> elements of the expression (`condition` can be a multidimensional matrix) are <u>non-zero</u>
  - the condition is converted to a relational expression, i.e. till all elements are `true`
  - logical and relational operators are often used for condition testing

- if `condition` is not a scalar, it can be reduced using functions `any` or `all`

Typical application of loops

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

mThrows  = 1e3;
nTimes   = 1e2;
results  = nan(mThrows, nTimes);
for iTime = 1:nTimes % however, can be even further vectorized!
    results(:, iTime) = round(rand(mThrows, 1)); % vectorized
end
```

```
%% script finds out the number of lines in a 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);
```

Loops #2

360 s

- calculate the sum of integers from 1 to 100 using `while` cycle
  - apply any approach to solve the task, but use `while` cycle

```
%% script calculates sum from 1 by 1 to 100
close all; clear; clc;
...
...
...
...
...
...
...
...
```

- are you able to come up with another solution (using a Matlab function and without cycle)?

`while` cycle – infinite loop

- pay attention to conditions in `while` cycle that are always fulfilled ⇒ danger of infinite loop
  - mostly, not always however(!!) it is a semantic error

- trivial, but good example of a code…

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

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

… that „never“ ends (shortcut to terminate: CTRL+C)

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Interchange of an index an complex unit

- be careful not to confuse complex unit (`i`, `j`) for cycle index
  - try to avoid using `i` and `j` as an index
    - overloading can occur (applies generally, e.g. `>> sum = 2` overloads the `sum` function)

- find out the difference in the following pieces of code:

```
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
```

- all the commands, in principle, can be written as one line

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

  - usually less understandable, not even suitable from the point of view of the speed of the code

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Nested loops, loop combining

- quite frequently there is a need for nested loops
  - consider vectorising instead
  - consider loop type
- loop nesting usually rapidly increases computational demands

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

mThrows  = 1e3;
nTimes   = 1e2;
results  = nan(mThrows, nTimes);
for iThrow = 1:mThrows
    for iExperiment = 1:nTimes % not vectorized (30 times slower!!)
        results(iThrow, iExperiment) = round(rand(1));
    end
end
```


Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Loops #3

600 s

- fill in the matrix using loops $\mathbf{A}(m,n) = \dfrac{mn}{4} + \dfrac{m}{2n}$
- consider $m \in \{1,\ldots,100\}$, $n \in \{1,\ldots,20\}$, allocate matrix first
- create a new script

```
%% script fills a matrix
close all; clear; clc;
...
...
...
...
...
...
...
...
...
```

- to plot the matrix **A** use for instance the function `pcolor(A)`

Loops #4

600 s

- in the previous task the loops can be avoided entirely by using vectorising
  - it is possible to use `meshgrid` function to prepare the matrices needed
  - `meshgrid` can be used for 3D arrays as well!!

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Loops #5

600 s

- visualize current distribution of a dipole antenna described as

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

- in the interval $t\in(0,4\pi),\quad x\in\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$ choose N = 101

for visualization inside the loop use following piece of code:

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

Loops #6

600 s

- try to write moving average code applied to following function

$$\mathrm{f}\left(x\right)=\sin^2\left(x\right)\cos\left(x\right)+0.1r\left(x\right),$$

where $r(x)$ is represented by function of uniform distribution (`rand()`)

- use following parameters

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

- and then plot:

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

- try to make the code more efficient

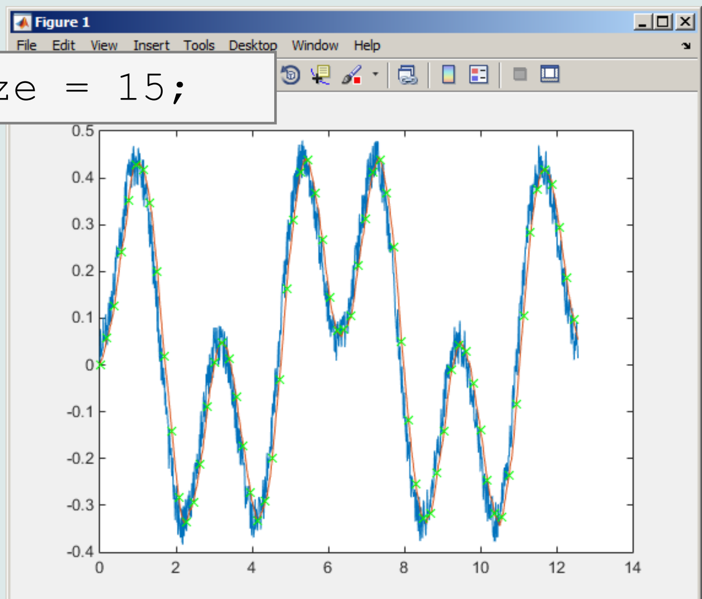
Loops #7

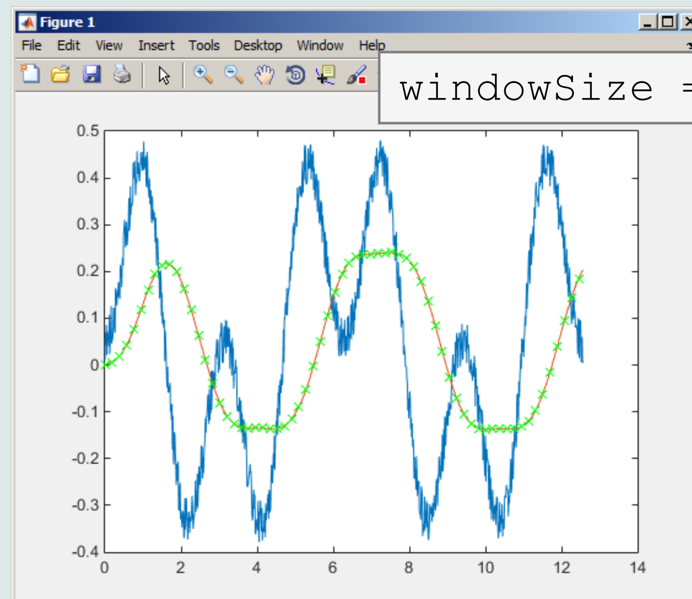
600 s ↑

- for comparison it is possible to use Matlab built-in function `filter`

- check how the result is influenced by parameter `windowSize`

`windowSize = 15;`





`windowSize = 150;`

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

break, continue

- function `break` enables to terminate execution of the loop

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

if (true)

- function `continue` passes control to next iteration of the loop

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

if (true)

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Loops vs. vectorizing #1

- since Matlab 6.5 there are two powerful hidden tools available
  - *Just-In-Time accelerator* (JIT accelerator)
  - *Real-Time Type Analysis* (RTTA)

- JIT enables partial compilation of code segments
  - precompiled loops are even faster than vectorizing
  - following rules have to be observed with respect to loops:
    - scalar index to be used with `for` loop
    - only built-in functions are called inside the body of `for` loop
    - the loop operates with scalar values only

- RTTA assumes the same data types as during the previous course of the code - significant speed up for standardized calculations
  - when measuring speed of the code, it is necessary to carry out so called warm-up (first run the code 2 or 3 times)

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Loops vs. vectorizing #2

- the motivation for introduction of JIT was to catch up with 3. generation languages
  - when fully utilized, JIT's computation time is comparable to that of C or Fortran
- highest efficiency (the highest speedup) in particular
  - when loops operate with scalar data
  - when no user-defined functions are called (i.e. only build-in functions are called)
  - when each line of the loop uses JIT

- as the result, some parts of the code don't have to be vectorized (or should not even be!)
- the whole topic is more complex (and simplified here)
  - for more details see `JIT_accel_Matlab.pdf` at the webpage of this course

Loops vs. vectorizing #3

- previous statement will be verified using a simple code - filling a band matrix
- conditions for using JIT are fulfilled …
  - working with scalars only, calling built-in functions only
  - HW and Matlab ver. dependent!
- try it yourself…

```
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 0.20s (2016b)
```

```
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 0.49s
(2016b)
```

Program branching

- if it is needed to branch program (execute certain part of code depending on whether a condition is fulfilled), there are two basic ways:
  - `if – elseif – else – end`
  - `switch – case – otherwise – end`

```
if condition
    commands
elseif condition
    commands
elseif condition
    commands
else
    commands
end
```

```
switch variable
 case value1
    commands
 case {value2a, value2b, ...}
    commands
 case ...
    commands
 otherwise
    commands
end
```


Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

`if` vs. `switch`

`if-elseif-else-end`	`switch-otherwise-end`
it is possible to create very complex structure (`&&` / `||`)	simple choice of many options
`strcmp` is used to compare strings of various lengths	test strings directly
test equality / inequality	test equality only
great deal of logical expressions is needed in the case of testing many options	enables to easily test one of many options using `{}`

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Program branching – `if` / `else` / `elseif`

- the most probable option should immediately follow the `if` statement
- only the `if` part is obligatory
- the `else` part is carried out only in the case where other conditions are not fulfilled

- if a M×N matrix is part of the condition, the condition is fulfilled only in the case it is fulfilled for each element of the matrix

- the condition may contain calling a function etc.
- conditions `if` may be nested

```
c = randi(1e2)
if mod(c, 2)
    disp('c is odd');
elseif c > 10
    disp('even, >10');
else
    disp('even, <=10');
end
```

Program branching – `if` / `else` / `elseif`

400 s

- generate random numbers

```
r = 2*rand(8, 1)-1;
```

- save the numbers in vectors `Neq` and `Pos` depending on whether each number is negative or positive; use `for` cycle, `if-else` statement and indexing for storing values of `r`

```
% your code
...
...
...
...
...
...
...
...
```

  - pay attention to growth in size of vectors `Pos` and `Neq` – how to solve the problem?

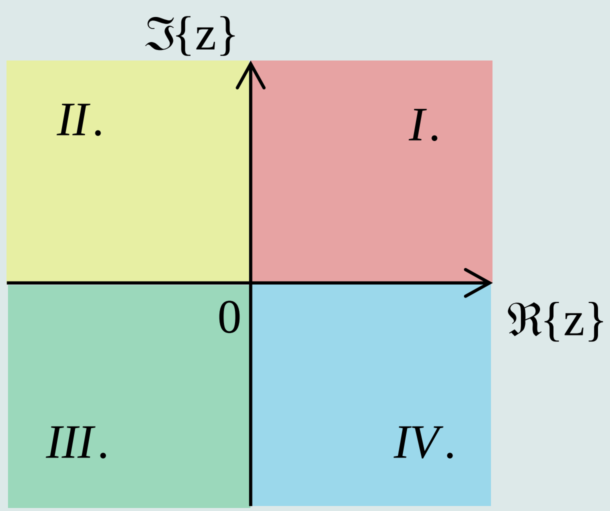
- can you come up with a more elegant solution? (`for` cycle is not always necessary)

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Program branching – `if` / `else` / `elseif`

500 s ↑

- write a script generating a complex number and determining to what quadrant the complex number belongs to



Department of Electromagnetic Field, CTU FEE, `miloslav.capek@fel.cvut.cz`

Program branching – switch / case

- does a variable correspond to one of (usually many) values?
- the commands in the part `otherwise` are carried out when none of the cases above applies (compare to `else` in the `if` statement)

- suitable to evaluate conditions containing strings
  - if you want to learn more details on when to use `if` and when to use `switch`, visit pages `blogs.mathworks.com`

- it is appropriate to always terminate the statement by `otherwise` part

```
c = 0.5*randi(1e2)
switch mod(c, 2)
 case 1
    disp('c is odd integer');
 case 0
    disp('c is even integer');
 otherwise
    disp('c is decimal number');
end
```


Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Program branching – `switch` / `case`

450 s

- create a script that, given lengths of two sides of a right triangle, calculates the length of the third side (Pythagorean theorem)
  - two sides are known together with string marking the type of unknown side (`'leg'` for leg or `'hyp'` for hypotenuse)

```
%% HINT:
% input variables will be here
%(including type of unknown side)
switch aaa % aaa denotes the type of unknown side
  case 'leg' % calculation for the first type of side
%     calculation1
  case 'hyp' % calculation for the second type of side
%     calculation2
  otherwise % unknown type
%     return empty (default) values
end
```

What does the script do?

300 s

- try to estimate what does the script below assign to `logResult` variable depending on input variable `vec` (a vector)
  - are you able to decide whether there is a Matlab function doing the same?

```
% vec is a given vector

logResult = false;
m = 1;
while (m <= length(vec)) && (logResult == false)
    if vec(m) ~= 0
        logResult = true;
    end
    m = m + 1;
end
```

Same as any !!!

What does the script do?

300 s

- try to estimate what does the script below assign to `logResult` variable depending on input variable `mat` (a matrix)
  - are you able to decide whether there is a Matlab function doing the same?

```
% mat is a given matrix
count  = 0;
[mRows, nColumns] = size(mat);
for m = 1:mRows
    for n = 1:nColumns
        if mat(m,n) ~= 0
            count = count + 1;
        end
    end
end
logResult = count == numel(mat);
```

Same as `all` (for matrices, i.e. `all(all())`)!!!

Example of listing more options

- `switch` supports options listing
  - evaluation of options A1 a A2 in the same way:

```
switch my_expression
   case {'A1', 'A2'}
       % do something
   otherwise
       % do something else
end
```

Inifinite loop – `for` cycle (a riddle)

- in the last lecture we learned how to construct the infinite loop with the `while` command (`>> while true, 'ok', end`)
  - Do you think, that the infinite loop can be constructed with the `for` cycle as well?
  - How?
  - Are there any restrictions? How many cycles will be performed and why?

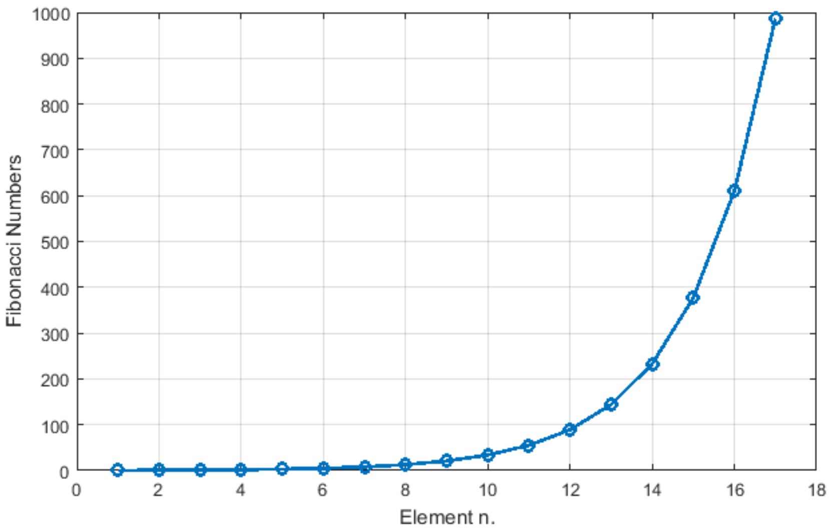
Discussed functions

`cell`	create cell array	●
`factorial`	calculate factorial	
`switch-case-otherwise-end`	condition statement	●
`for-end`	loop over distributed range	●
`while-end`	repeat loop while condition is true	●
`break, continue`	terminate loop, pass control to next iteration of loop	●
`if-elseif-else-end`	branching statement	●

Exercise #1

600 s

- draft a script to calculate values of Fibonacci sequence up to certain value `limit`
  - have you come across this sequence already?
  - if not, find its definition
  - implementation:
    - what kind of loop you use (if any)?
    - what matrices / vectors do you allocate?
  - plot the resulting series using function `plot(f, '-o')`

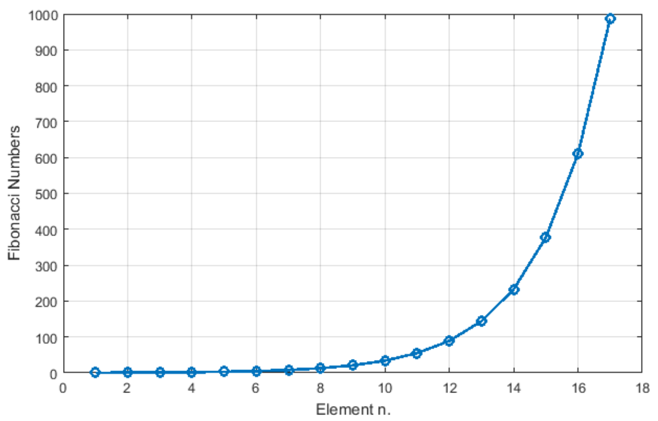


Exercise #2

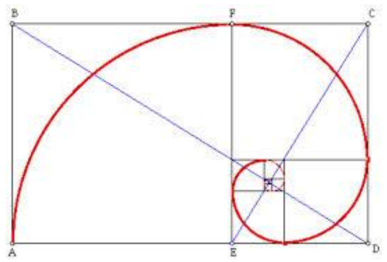
240 s

- rate of reproduction of rabbits:

```
%% fibonacci sequence
% your code
...
...
...
...
...
...
plot(f, '-o');
xlabel('Element n.')
ylabel('Fibonacci Numbers')
```



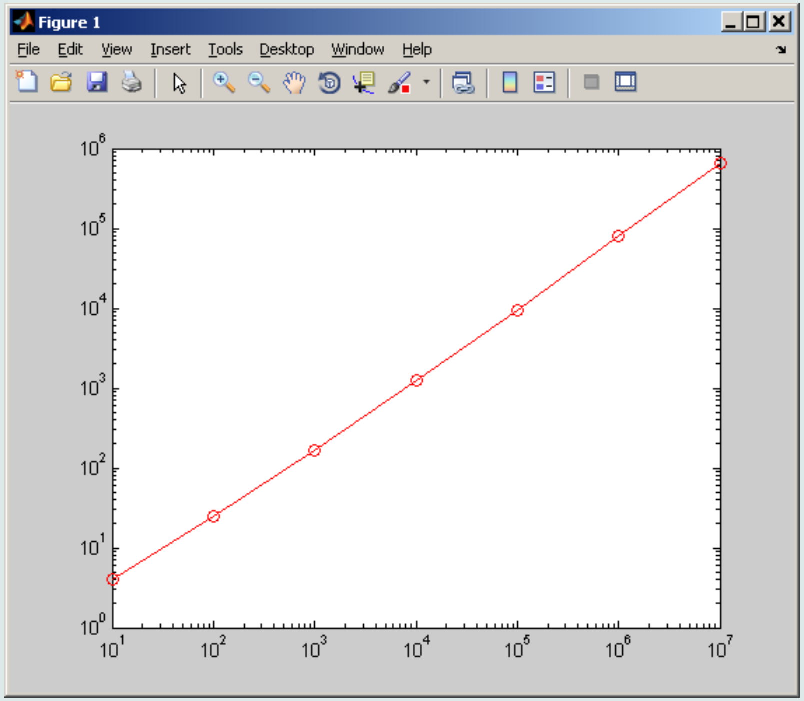
- try to find out the relation of the series to the value of golden ratio
- try to calculate it:



Exercise #3

600 s

- try to determine the density of prime numbers
  - examine the function `primes` generating prime numbers
  - for the orders 10^1 – 10^7 determine the primes density (i.e. the number of primes up to 10, to 100, …, to 10^7)
  - outline the dependence using `plot`
  - use logarithmic scale (function `loglog`)
    - how does the plot change?



Exercise #4

- did you use loop?
- is it advantageous (necessary) to use a loop?
- do you allocate matrices?
- what does, in your view, have the dominant impact on computation time?

Exercise #5

- the script can be further speeded-up
  - function `primes` is costly and can be run just once:

- would you be able to speed-up the script even more?

Exercise #6

600 s ↑

- following expansion holds true:

$$\arctan\left(x\right)=\sum_{n=0}^{\infty}\left(-1\right)^{n}\frac{\left(x\right)^{2n+1}}{2n+1}=x-\frac{x^{3}}{3}+\frac{x^{5}}{5}-\frac{x^{7}}{7}+\frac{x^{9}}{9}-\ldots$$

- based on the expansion for $x = 1$ estimate value of π:

$$\arctan\left(1\right)=\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\ldots$$

- determine the number of elements of the sum and computational time required to achieve estimation accuracy better than $1\cdot 10^{-6}$

Exercise #7

600 s ↑

- estimate value of π using following expansion

$$\frac{\pi}{8}=\sum_{n=0}^{\infty}\frac{1}{(4n+1)(4n+3)}=\frac{1}{1\cdot 3}+\frac{1}{5\cdot 7}+\frac{1}{9\cdot 11}+\ldots$$

- determine the number of elements of the sum and computational time required to achieve estimation accuracy better than $1\cdot 10^{-6}$

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Exercise #8

600 s ↑

- use following expression to approximate π :

$$\frac{\pi}{4} = 6\arctan\left(\frac{1}{8}\right) + 2\arctan\left(\frac{1}{57}\right) + \arctan\left(\frac{1}{239}\right)$$

- use following expression to implement the arctan function :

$$\arctan(x) = \sum_{n=0}^{\infty} (-1)^n \frac{(x)^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \dots$$

- determine the number of elements of the sum and computational time required to achieve estimation accuracy better than $1\cdot 10^{-6}$ and compare the solution with previous solutions

Thank you!

ver. 7.1 (20/3/2017)
Miloslav Čapek, Miloslav Čapek
miloslav.capek@fel.cvut.cz

Document created as part of A0B17MTB course.