Lecture 4: Loops, Program Branching

BE0B17MTB – Matlab

Miloslav Čapek, Viktor Adler, Michal Mašek, and Vít Losenický

Department of Electromagnetic Field
Czech Technical University in Prague
Czech Republic
matlab@elmag.org

October 14, 2020
Winter semester 2020/21

Outline

1. Loops
2. Program Branching
3. Excercises

Loops I.

- Repeating certain operation multiple-times, one of the basic programming techniques.
- There are two types of cycles in MATLAB:
  - `for`: the most used one, number of repetitions is known in advance,
  - `while`: condition is known ensuring cycle continuation as long as it remains true.
- Essential programming principles to be observed:
  - memory allocation (matrix-related) of sufficient size,
  - cycles should be properly terminated,
  - to ensure terminating condition with `while` cycle,
  - (*more on it later*).
- Frequently, it is possible to modify array (1D → 2D, 2D → 3D using function `repmat` or implicit expansion of dimensions, and carry out a matrix-wise operation, vectorized code is faster under certain conditions and more understandable, possibility of utilization of GPU).
- Always ask the question: Is the cycle really necessary?

Program Flow

Loop – for I.

- for loop is applied to known number of repetitions of a group of commands:

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

- expression is a vector/matrix.
  - Columns of this vector/matrix are successively assigned to n/m.

```
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 as mPoints, mRows, nSymbols, etc.
  - For clarity, it is suitable to use, *e.g.*, mXX for rows and nXX for columns.

Program Flow

Loop – `for` II.

- Create a script calculating factorial N,
  - use a cycle, verify your result using MATLAB function `factorial`.

- Can you come up with other solutions (*e.g.*, using vectorizing)?

- Compare all possibilities for decimal input N as well.

300

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 it may be reduced.
    - Allocate the variable of the largest size first, then the smaller ones.
- Example (try it):

```
%% 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
```

Program Flow

Loop – `while` I.

- Keeps on executing commands contained in the body of the cycle depending on a logical condition.

```
while condition
   commands
end
```

- Keeps on executing commands as long as all elements of the expression (`condition` can be a multidimensional matrix) are non-zero.
  - 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, in can be reduced using function `any` or `all`.

Program Flow

Typical Application of Loops

```
%% script generates N experiments with M throws with a coin
clear;
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
clear;
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);
```

Loop – `while` II.

- Calculate the sum if integers from 1 to 100 using `while` cycle.
  - Apply any approach to solve the task, but use `while` cycle.

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

300

Infinite Loop

- Pay attention to conditions in `while` cycle that are always fulfilled ⇒ danger of infinite loop.
  - Mostly (not always) it is a semantic error.
- Trivial, but good example of a code:

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

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

  - These codes "never" ends. Shortcut to terminate: CTRL+C.

Interchange of an Index and Complex Unit

- Be careful not to confuse complex unit (`i`, `j`) with a 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 can, in principle, be written in one line:

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

  - Usually less understandable. In general, not as fast as commands written separately line by line.

Program Flow

Nested Loops, Loop Combining

- Often, there is a need for a nested loops.
  - Consider vectorizing instead.
  - Consider proper loop type.
- Loop nesting usually rapidly increases computational demands.

```
%% script generates N experiments with M throws with a coin
clear;
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
```

Loops II.

- Fill in the matrix $\mathbf{A} = [a_{mn}]$ using loops. The matrix entries read

$$a_{mn} = \frac{mn}{4} + \frac{m}{2n}.$$

- Consider $m \in \{1, \ldots, 100\}$, $n \in \{1, \ldots, 20\}$, allocate matrix first.
- To plot the matrix $\mathbf{A}$ use for instance function `pcolor(A)`.

480

Loops III.

- In the previous task the loops can be avoided by using vectorizing.
  - Try to eliminate inner loop.
  - Try to eliminate both loops using implicitly expansions of vectors with compatible sizes.

600

Loops IV.

- Visualize guitar string whose movement is described as

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

- in the interval $t \in (0, 4\pi)$, $x \in \left(\frac{-\pi}{2}, \frac{\pi}{2}\right)$, choose $N = 101$ samples.
- For visualization inside the loop use following piece of code:

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

600

Commands break and continue

- Function break enables to terminate execution of the loop.

```
% previous code ..
for k = 1:length(v)
   if v(k) > x
      break
   end
   % another code
end
```

if true

- Function continue passes control to the next iteration of the loop.

```
% previous code ..
for k = 1:length(v)
   if v(k) > x
      continue
   end
   % another code
end
```

if true

Loops vs. Vectorizing I.

- Since MATLAB 6.5 there are two powerful hidden tools available:
  - *Just-In-Time accelerator* (JIT),
  - *Run-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 has to be used with `for` loop.
    - Only build-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).

Loops vs. Vectorizing II.

- The motivation for introduction of JIT was to catch up with third-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 vectorized (or should not even be!).
- The whole topic is more complex (and simplified here).

▶ More information

Loops vs. Vectorizing III.

- Previous statement is verified using a simple code – filling a band matrix.
- Conditions for using JIT are fulfilled (working with scalars only, calling build-in functions only).
- HW and MATLAB version dependent!

```
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 0.2182 s
```

```
mat = toeplitz([1, 3, zeros(1, N-2)], ...
  [1, 2, zeros(1, N-2)]);
% computed in 0.3428 s (2019a, Win10, i5)
```

```
clear;clc;
N = 5e3;
mat = nan(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 0.3407 s
```

Program Branching

- If it is needed to branch the 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 value3
      commands
   otherwise
      commands
end
```

Program Flow

Cell data type "{ }" will be explained in detail later.

if vs. switch

if – elseif – else – end	switch – case – otherwise – end
It is possible to create very complex structure (`&&` / `||`).	Simple choice of many options.
Function `strcmp` is used to compare strings of various lengths.	Test string directly.
Test equality / inequality.	Test equality only.
Great deal of logical expression is needed in the case of testing many options.	Enables to easily test many options using cell data type (*more on later*).

Program Branching – if – elseif – else I.

- 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 $N \times M$ 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.
- `if` conditions can be nested.

```
c = randi(100)
if mod(c, 2)
   disp('c is odd');
elseif c > 10
   disp('c is even, > 10')
else
   disp('c is even, <= 10')
end
```

Program Flow

Program Branching – if – elseif – else II.

- 40 students pass the test with following points:

```
points = randi(100, nStudents,
1);
```

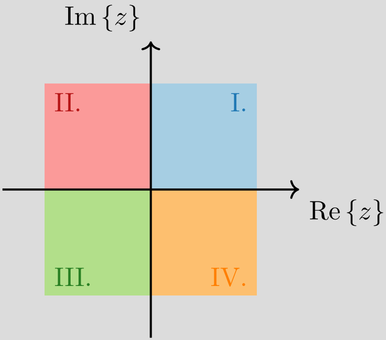
- Count how many:
  - excellent students (with 80+ points),
  - average students (with 35-79 points),
  - poor students (with less than 35 points)

  are visiting the class.
- Use `for` cycle, `if–else` statement and indexing for storing values of points.
- Can you come up with a more elegant solution? (`for` cycle is not always necessary.)

400

Program Branching – `if` – `elseif` – `else` III.

- Write a script generating a complex number z and determining to what quadrant the complex number belongs to.



500

Program Branching – switch – case I.

- Does a variable correspond to one of (usually many) values?
- Each switch must have at leas one case, otherwise part is not required, but highly recommended (to deal somehow with unpredictable options).
- The commands in the part otherwise are carried out when none of the cases above apply.
- Suitable to evaluate conditions containing strings.
- If you want to learn more details on when to use if and when to use switch, visit:
  ▶ blogs.mathworks.com

```
c = 0.5*randi(100)
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
```

Program Branching – `switch` – `case` II.

- Depending on the given grade from important test, select the amount of money a student will receive as an reward:

Grade	Reward
1	1000
2	500
3	200
4	50
5	0

- Use `switch` – `case` statement.

300

Program Branching – switch – case III.

- 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:
clear, clc
% input variables will be here
% including type of unknown side
switch type
   case 'hyp'
      % calculation here
   case 'leg'
      % calculation here
   otherwise % unknown values
      % return empty (default) values
end
```

450

What Does the Script Do? I.

- Try to estimate what the script below assigns 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
```

90

What Does the Script Do? II.

- Try to estimate what the script below assigns 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);
```

90

Example of Listing More Options In switch - case

- switch supports options listing
  - Evaluation of values A1 and A2 in the same way:

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

Program Flow

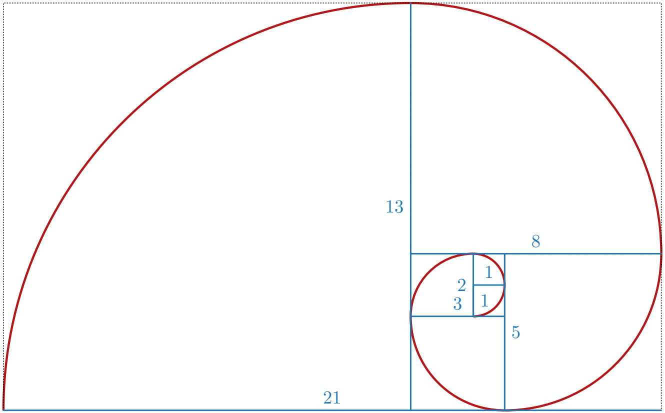
Infinite Loop – for Cycle (A Riddle)

- In this lecture we learned how to construct infinite loop with `while` cycle command (`>> while true, disp('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?

Exercises

Exercise I.a

- Fibonacci sequence: $F_1 = 1, \quad F_2 = 1, \quad F_n = F_{n-1} + F_{n-2}$ for $n > 2$.

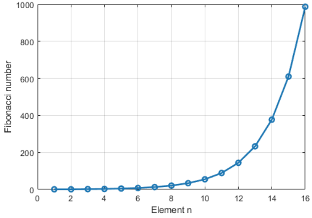


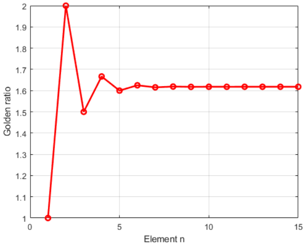
Exercise I.b

- Draw a script to calculate values of Fibonacci sequence up to certain value `limit`.
  - plot the resulting series using function: `figure(1), plot(F, '-o')`
- Calculate length of Fibonacci spiral.
- Calculate approximations to the golden ratio:

$$\varphi = \lim_{n \to \infty} \frac{F_{n+1}}{F_n} = \frac{1+\sqrt{5}}{2} \approx 1.618033$$

  - plot it: `figure(2), plot(phi, '-or')`





600

Exercise II.a

- Create vector $\mathbf{v} \in \mathbb{C}^{4\times 1}$ which contains one complex number from each quadrant.

$$
\begin{aligned}
&\mathrm{Re}\{v_1\} > 0, \mathrm{Im}\{v_1\} > 0\\
&\mathrm{Re}\{v_3\} < 0, \mathrm{Im}\{v_3\} < 0\\
&\mathrm{Re}\{v_2\} > 0, \mathrm{Im}\{v_2\} < 0\\
&\mathrm{Re}\{v_4\} < 0, \mathrm{Im}\{v_4\} > 0
\end{aligned}
$$

- Prepare a code which can generate random complex number. Determine its quadrant and save it into $\mathbf{v}$. Repeat the process until vector $\mathbf{v}$ is not full.

500

Exercise II.b

- Can you do the same without loop and program branching?

360

Exercise III.a

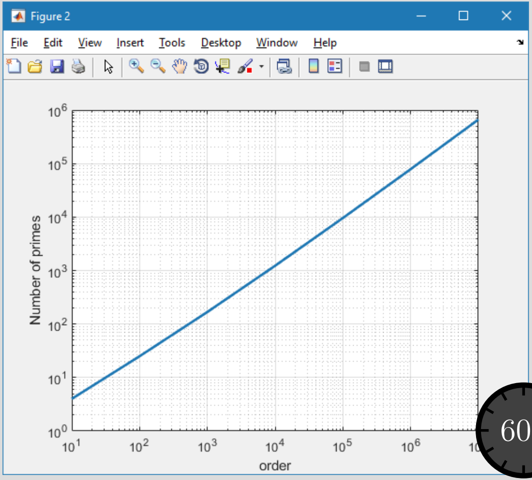
- Try to determine the density of prime numbers:
  - examine the functions `primes` generating prime numbers,
  - for the orders $10^1 - 10^7$ determine the primes density (*i.e.*, the number of primes up to 10, to 100, ..., to 10^7).
  - Outline the dependence using `plot`.
  - Display results in logarithmic scale,
    - `loglog(xData, yData)`.
    - How does the plot change?

```
%% density of prime numbers
clear, clc, close all
N = 7;
nPrimes = % alocate here
orders = % vector of orders 10^1-10^7
% your code here..
% ..
% ..
% ..
figure, plot(nPrimes)
figure, loglog(orders, nPrimes)
```

300

Exercise III.b

- Can the script be speeded-up?
- What does, in your view, have the dominant impact on computation time?
- Is it necessary to compute primes in every loop?



600

Exercise IV.a

- Following expansion holds true:

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

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

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

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

500

Exercise IV.b

- Estimate value of π using following expansion:

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

- Approximate value of π using following expansion with the expansion for $\arctan(x)$ from the previous slide:

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

- Determine the number of elements of the sum and computational time required to achieve estimation accuracy better than $1 \cdot 10^{-6}$.
- Compare all three solutions.

500

Exercise IV. – Solution

Questions?

BE0B17MTB – Matlab

matlab@elmag.org

October 14, 2020
Winter semester 2020/21

Acknowledgement: Filip Kozak, Pavel Valtr.