Lecture 5: Functions, Branching, Cycles

A8B17CAS

Miloslav Čapek

Department of Electromagnetic Field
Czech Technical University in Prague
Czech Republic
miloslav.capek@fel.cvut.cz

November 8
Winter semester 2022/23

Outline

1. Functions
2. Program Branching
3. Cycles

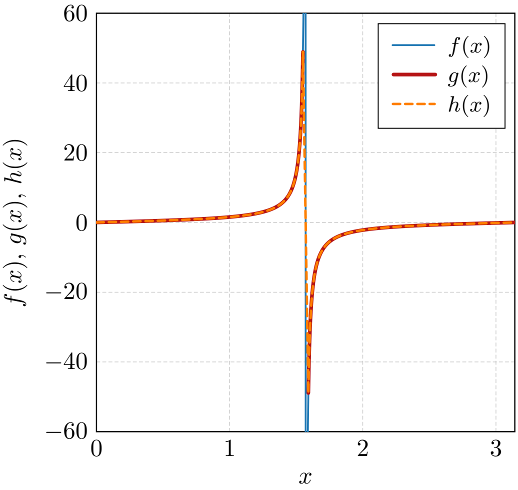
Warm Up: Function Thresholding and Linear Interpolation

Consider the following code snippet with the function $f(x) = \tan(x)$ defined for $x \in [0, \pi]$.

- Define function $g(x)$ as thresholded $f(x)$, with all values $|f(x)| > 50$ being replaced by `NaN`.
- Define function $h(x)$, where `NaN` values are replaced by a linear approximation.

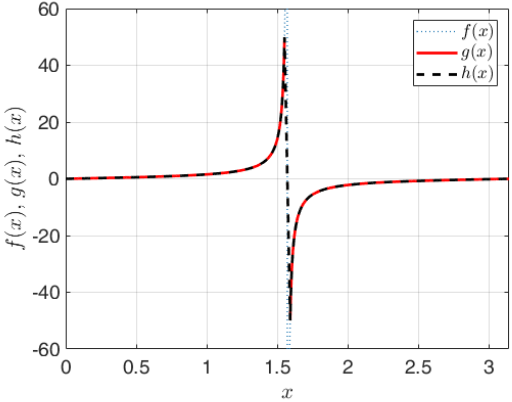
```
%% f(x) = tan(x)
x  = linspace(0, pi, 1e5);
fx = tan(x);

figure;
plot(x, fx, ':', 'LineWidth', 1);
ylim([-60 60]);
grid on;
```



Exercise

Warm Up: Function Thresholding and Linear Interpolation



Exercise

Function Header

- Function header:

```
function [out1, out2, ..]  = functionName(in1, in2, ..)
```

↑ keyword ↑ function's output parameters ↑ function's name ↑ function's input parameters

  - `functionName` has to follow the same rules as a variable's name.
  - `functionName` cannot be identical to any of its parameters' names.
  - `functionName` is usually typed as `lowerCamelCase` or using underscore character (`my_function`).

Functions – Notes

- More efficient, more transparent, and faster than scripts.
- Defined input and output, comments → function header is necessary.
- Can be called from Command Window, script, or another function (in all cases the function has to be accessible).
- Each function has its Workspace created upon the function's call and terminated with the last line of the function.
- Name of file × name of function. MATLAB sees file name!
- Be careful with overloading of build-in functions (`sum(v)` × `sum = 5`).

Adopt Code Verifying Dirichlet's Theorem as a Function

Generate all prime numbers which can be expressed as a summation of consecutive even and odd numbers (*e.g.*, $1 + 2 = 3$, $3 + 4 = 7$).

- Modify the existing code (on right) as a function `getDirichlet.m`.
  - Input: `N` (the highest value checked), outputs: `v` (the Dirichlet series), `Nv` (number of members).
- Implement function `plotDirichlet.m` which plots the series.
- Run both functions from a script.

```
N  = 100;
oddeven = (1:2:N) + (2:2:N);
v  =  intersect(primes(N), oddeven);
Nv = length(v)
```

```
% Dirichlet series
clear;
clc;

%% Calculate series
v = getDirichlet(1e3);

%% Plot series
hndl = plotDirichlet(v, 'k');
```

Exercise

Adopt Code Verifying Dirichlet's Theorem as a Function

Script to run the functions:

Functions:

Exercise

When is function terminated?

- MATLAB interpreter reaches the last line.
- Interpreter comes across the keyword `return`.
- Interpreter encounters an error (can be evoked by `error` as well).
- On pressing CTRL+C.

```
function res = myFcn2(matrixIn)

if isempty(matrixIn)
   error('matrixInCannotBeEmpty');
end
normMat = matrixIn - max(max(matrixIn));

if matrixIn == 5
   res = 20;
   return
end
end
```

Anonymous Functions I.

Anonymous functions make it possible to create handle references to a function that is not defined as a standalone file.

- The function has to be defined as one executable expression.

```
>> sqr = @(x) x.^2; % create anonymous function (handle)
>> res = sqr(5);    % x ~ 5, res = 5^2 = 25;
```

- Anonymous function can have more input parameters.

```
>> A = 4; B = 3; % parameters A,B have to be defined
>> sumAxBy = @(x, y) (A*x + B*y); % function definition
>> res2 = sumAxBy(5,7);    % x = 5, y = 7
% res2 = 4*5+3*7 = 20+21 = 41
```

- Anonymous function stores variables required as well as prescription.

- More information: » `doc Anonymous Functions`

Anonymous Functions II.

- Create anonymous function $\mathbf{A}(p) = [A_1(p) \quad A_2(p) \quad A_3(p)]$ so that

$$A_1(p) = \cos^2(p),$$
$$A_2(p) = \sin(p) + \cos(p),$$
$$A_3(p) = 1.$$

- Calculate and display its components for range $p = [0, 2\pi]$.

- Check the function $\mathbf{A}(p)$ with MATLAB built-in function functions, *i.e.*, `functions(A)`.

Exercise

Namespace

Namespace is a simple way how to create a package and organize functions into logical blocks.

1. Create a folder starting with "+".
2. Place selected functions inside.
3. Any function is accessed using namespace prefix.

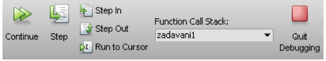
```
...\+foo
    bar.m
    baz.m
```

```
foo.bar();
foo.baz();
```

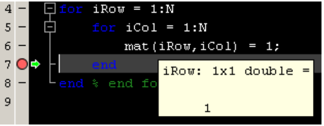
Debugging

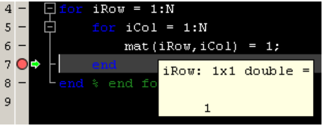
Possible errors:

- syntactical (MATLAB m-lint),
- semantical (debugging),
- unexpected (`try–catch`).



- Set breakpoint (click on the dash next to the line number),
- run the script (F5),
- check the status of variables,
- keep on tracing the script.
- use Continue, Step (F10) etc.





Numerical Integration of a Function

Using `integral` function calculate integral of a function $G = \int g(x)\,\mathrm{d}t$ in the interval $t \in [0, 1]\,\mathrm{s}$. The function has the following time dependency, where $f = 50\mathrm{Hz}$:

$$g(t) = 10\cos^2(2\pi f t) + 5\cos(4\pi f t)$$

- Solve the problem using an anonymous function.

- Solve the problem using classical function.

Exercise

Program Branching

- If it is needed to branch the program (execute a 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
```

Loop – for

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, nSymbols, etc.

Factorial

- Create a script calculating the factorial of 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.

Exercise

Loop – while

- 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.
- `condition` has to be a scalar. Use `any` or `all` if needed.

Memory Allocation

- Allocation can prevent a 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.
- 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
```

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.

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

Questions?

A8B17CAS
miloslav.capek@fel.cvut.cz

November 8
Winter semester 2022/23
