Matlab lecture 3

Table of Contents

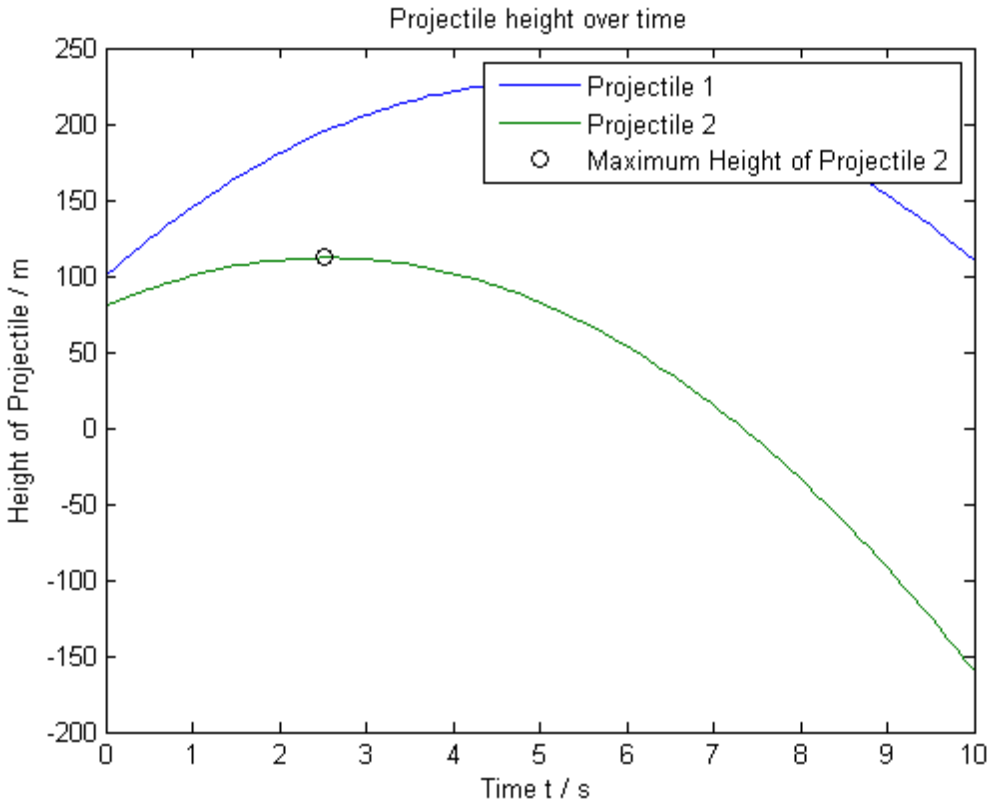
Version 0.1 by Markus Schellenberg, Winter Term 16/17

The find command

In Exercise 2 you had to set all negative values of a vector to zero. In that Exercise it should have been solved by using a loop. But there are other ways to do so.

Run the script `ProjectilePlot.m` from Exercise 2:

```
ProjectilePlot
```



Look at the values of variable `path2` by typing in `path2`

Click inside the figure and try to estimate when the function values of `path2` is smaller than zero

```
path2(70:80)
```

```
ans =

  Columns 1 through 7

   16.2167   11.7927    7.2686    2.6446   -2.0794   -6.9034  -11.8274

  Columns 8 through 11

  -16.8513  -21.9753  -27.1993  -32.5232
```

It can be seen, that path2 is below zero after position 74. To set it to zero you could just type in:

```
path2(74:end) = 0;
path2(70:80)
```

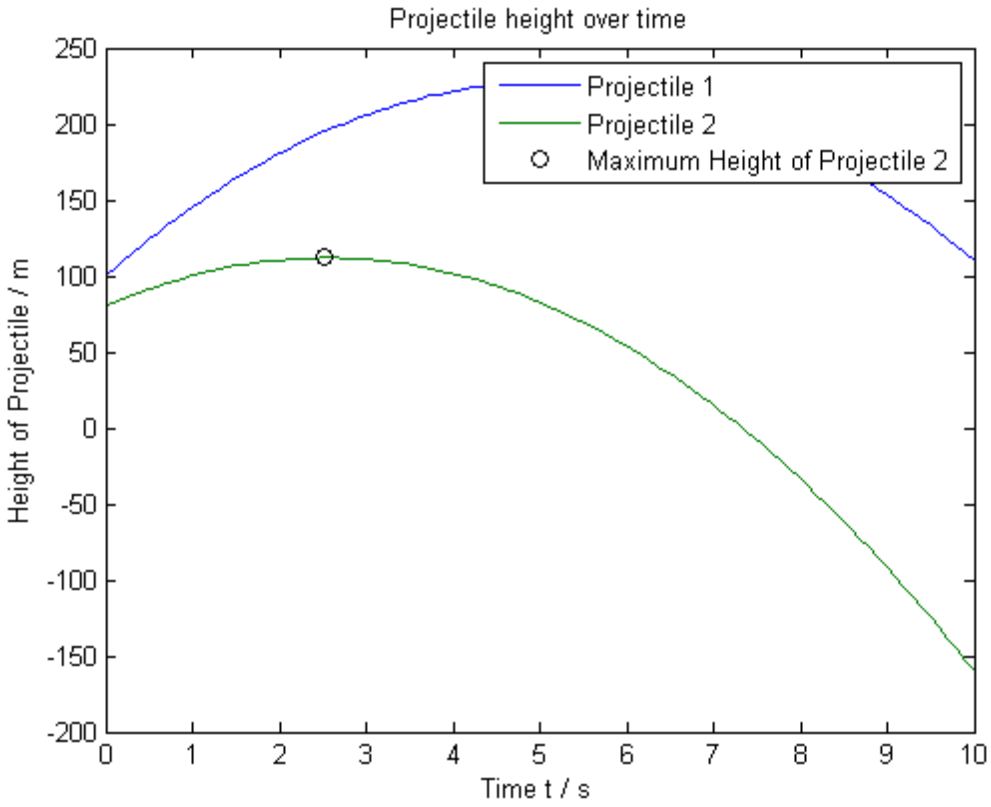
```
ans =

  Columns 1 through 7

   16.2167   11.7927    7.2686    2.6446         0         0         0

  Columns 8 through 11

         0         0         0         0
```

Rerun `ProjectilePlot.m` after line 20. The content of the figure will be replaced. Matlab always uses the last active figure for a plot. If you want to have a new (empty) figure just type in the keyword: `figure`

Is there a more elegant way to do the same operation? Try the `find` command:

```
ProjectilePlot;

negative_values_path2 = find(path2 < 0)
```

```
negative_values_path2 =

  Columns 1 through 13

    74    75    76    77    78    79    80    81    82    83    84    85

  Columns 14 through 26

    87    88    89    90    91    92    93    94    95    96    97    98

  Column 27

   100
```



Will find all positions where path2 is smaller than zero

```
path2(negative_values_path2)
```

```
ans =

  Columns 1 through 7

   -2.0794   -6.9034  -11.8274  -16.8513  -21.9753  -27.1993  -32.5232

  Columns 8 through 14

  -37.9471  -43.4711  -49.0950  -54.8189  -60.6428  -66.5667  -72.5906

  Columns 15 through 21

  -78.7144  -84.9383  -91.2621  -97.6860 -104.2098 -110.8336 -117.5574

  Columns 22 through 27

 -124.3812 -131.3050 -138.3287 -145.4525 -152.6763 -160.0000
```

Displays all values at the positions we found.

An easy and elegant way to replace all values smaller than zero with zero using only one line of code:

```
path2( find( path2 < 0 ) ) = 0;
path2(70:80)
```

```
ans =

  Columns 1 through 7

   16.2167   11.7927    7.2686    2.6446         0         0         0

  Columns 8 through 11

         0         0         0         0
```

Functions

Create a new function called f_compute_square.m as a separate file and save it inside your the folder your active folder:

```
function [ output_args ] = f_compute_square( input_args )
%F_COMPUTE_SQUARE multiplicates a number with itself.
%
% M. Schellenberg, 2016

output_args = input_args.^2;

end
```

You can run the function by typing: `f_compute_square(NUMBER)`, e.g.:

```
f_compute_square(5)
```

```
ans =

    25
```

or

```
result = f_compute_square(5)
```

```
result =

    25
```

Function Handles

You can shorten the input by adding a so called handle to any function. You will need the @ symbol to do so:

```
f = @f_compute_square;
result_2 = f(5)
```

```
result_2 =

    25
```

Anonymous Functions

If you want to create a function inside a script you can use so called anonymous functions. You also need the @ symbol to indicate what the variables of the anonymous function is. Try this:

```
f_compute_square2 = @(n) n.^2;
```

creates again a function that calculates the square of a value. n is the variable n.^2 is the desired calculation. The function can be called in the known way by typing in:

```
x = f_compute_square2(3)
```

```
x =

     9
```

Arrays of Function Handles

Anonymous functions can be stored in arrays. You can refer to any element of an array by using curly brackets: {}

```
f_sct = {@sin, @cos, @tan};

f_sct{1}(pi)
f_sct{2}(pi)
f_sct{3}(pi)
```

```
ans =

   1.2246e-16
```

```
ans =

    -1
```

```
ans =

  -1.2246e-16
```

Newton Method for finding the zero point

The Newton method for finding the zero point of a function is done by the following steps:

Choose a start point that is as close as possible to the expected zero point Define following iterative sequence:

$$x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)}$$

```
function [ output ] = f_newton( value, my_function, derivative )
%F_NEWTON calculates the root of a derivable function with the Newton
%method
%
% M. Schellenberg, 2106

output = value - my_function / derivative;

end
```

The test fuctions are

$\sin(x)$ with it's derivative $\cos(x)$

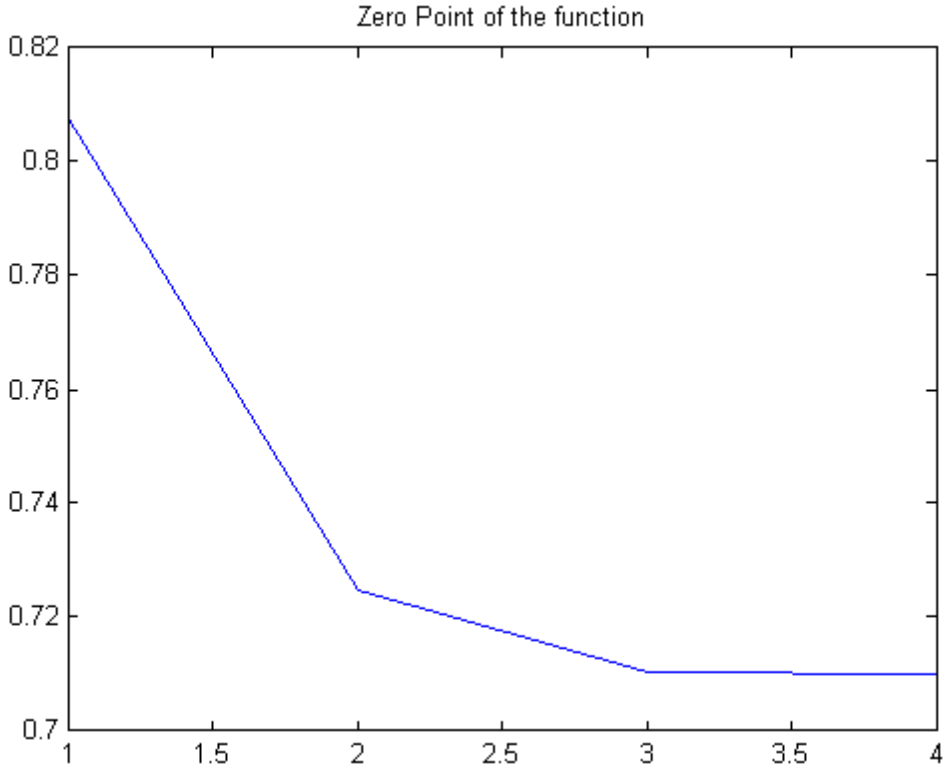
and

$x^5 - 14x^4 + x^2 - 3x + 5$ with with it's derivative $5x^4 + 56x^3 + 2x - 3$

```
% Variables and Constants
start_value = 1;    % Choose a start point, that is near the expected zero point
iterations = 4;     % Number of iterations used in the for loop

% Calculation of the root
format long % changes the ouput to more digits

for i = 1:iterations
    % First example function
    % start_value = f_newton(start_value, sin(start_value),cos(start_value));

    % Second example function
    start_value = f_newton(start_value, start_value.^5 - ...
                           14*start_value.^4+...
                           start_value.^2-...
                           3*start_value + 5, ...
                           5*start_value.^4 - ...
                           56*start_value.^3+2*start_value-3);

    % List of the results
    result_newton(i) = start_value;
end

% Output
```

```
formatSpec = ' Iteration number: %3.0f\n Actual value for the zero point: %4.10f\n
fprintf(formatSpec,iterations,start_value)

% Graphical representation of the quality of the result
figure, plot(result_newton)
title('Zero Point of the function')

format short % Back to normal output
```

```
Iteration number:   4

Actual value for the zero point: 0.7098473309
```



Import data from spreadsheet

the easiest way to import data from a spreadsheet (e.g. an Excel spreadsheet) is to drag and drop the file into the Matlab workspace and follow the instructions that are given by the wizard.

You have different opportunities to import the date (as a matrix, as several coulumns) and also the choise to eiter auto-create an import script or function. The import script from the lecture is displayed here:

```
% Script for importing data from the following spreadsheet:
%
%    Workbook: H:\Spectrum_Exp_35.xlsx Worksheet: Tabelle1
%
% To extend the code for use with different selected data or a different
% spreadsheet, generate a function instead of a script.
```

```
% Auto-generated by MATLAB on 2016/10/30 16:08:05


% Variables and constants
name_experiment = 'Spectrum_Exp_35.xlsx';   % Please change the filename


% Import the data
[~, ~, raw] = xlsread(name_experiment,'Tabelle1','B7:C21');

% Create output variable
Spectrum = reshape([raw{:}],size(raw));

% Clear temporary variables
clearvars raw;
```

Published with MATLAB® R2013a