Matlab lecture 5

Table of Contents

Version 0.1 by Markus Schellenberg, Winter Term 16/17

Clean up

```
clc, clear, close all
```

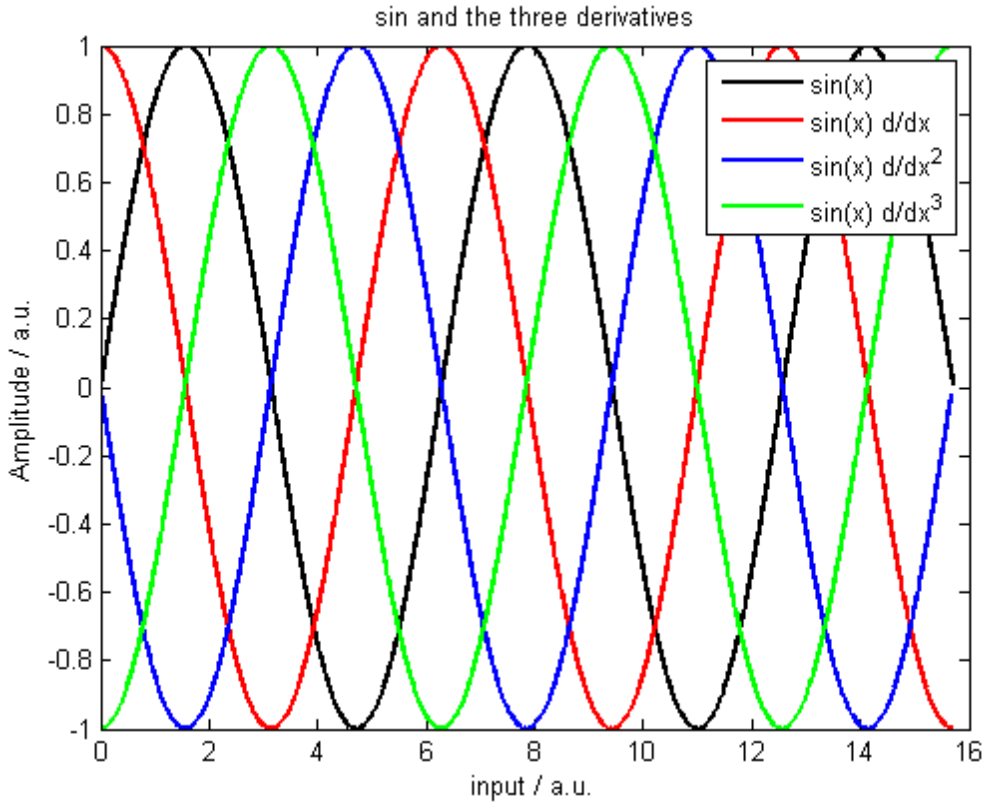
Derive functions with the diff command

The `diff` command gives you a difference inside a matrix and also an approximate derivative

Type in:

```
step_size = 0.01;                % step size
range = 0:step_size:5*pi;        % domain

% Functions to derive / Calculations
my_function = sin(range);       % range

% first derivative
my_function_first_derivative = diff(my_function)/step_size;

% second derivative
my_function_second_derivative = diff(my_function_first_derivative)/step_size;

% third derivative
my_function_third_derivative = diff(my_function_second_derivative)/step_size;


% plot the results
figure
plot(range(:,1:length(my_function)),my_function, 'black', 'linewidth', 2)

hold on

plot(range(:,1:length(my_function_first_derivative)),...
    my_function_first_derivative, 'red', 'linewidth', 2)

plot(range(:,1:length(my_function_second_derivative)),...
    my_function_second_derivative, 'blue', 'linewidth', 2)
```

```
plot(range(:,1:length(my_function_third_derivative)),...
    my_function_third_derivative, 'green', 'linewidth', 2)

% title, legend and labels
legend('sin(x)', 'sin(x) d/dx', 'sin(x) d/dx^2', 'sin(x) d/dx^3')
xlabel('input / a.u.')
ylabel('Amplitude / a.u.')
title('sin and the three derivatives')
```

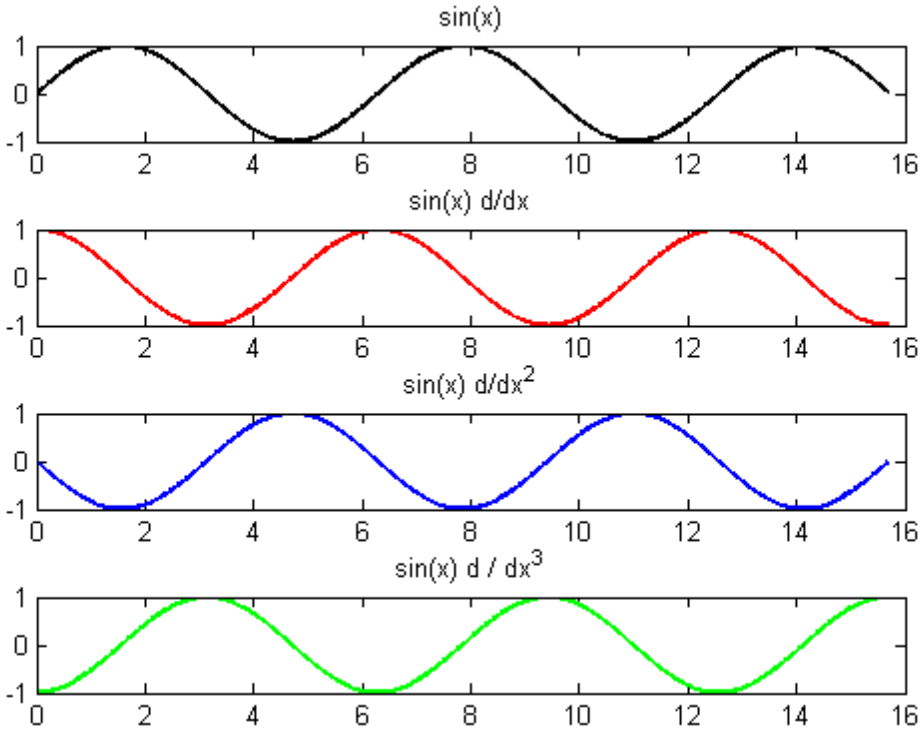


Subplot

With the `subplot` command you can plot several plots side by side among each other or both. `subplot(m,n,p)` , divides the figure into an m-by-n grid with an axes in the pth grid location. The grids are numbered along each row.

```
figure,
subplot(4,1,1)
plot(range(:,1:length(my_function)),my_function, 'black', 'linewidth', 2)
title('sin(x)')

subplot(4,1,2)
plot(range(:,1:length(my_function_first_derivative)),...
    my_function_first_derivative, 'red', 'linewidth', 2)
title('sin(x) d/dx')
```

```
subplot(4,1,3)
plot(range(:,1:length(my_function_second_derivative)),...
    my_function_second_derivative, 'blue', 'linewidth', 2)
title('sin(x) d/dx^2')

subplot(4,1,4)
plot(range(:,1:length(my_function_third_derivative)),...
    my_function_third_derivative, 'green', 'linewidth', 2)
title('sin(x) d / dx^3')
```



Curve scetching

This example shows you, how to do a curve scetching with on any derivable polynomial function:

$$x^n + x^{n-1} + \ldots x^2 + x + C$$

```
function [x, y, y_1, y_2, zero_points, extreme_points, extreme_points_y, ...
   turning_points, turning_points_y] = ...
   f_curve_scetching(polynom, range)
%F_CURVE_SCETCHING
% This function performs a curvature discussion for a polynomial and
% presents the result graphically.

% Input:
% Polynomial: coefficient vector of the polynomial range
% [start, end, step width]
```

```
%
% Output:
% Zeros: vector with the x-values of the zeros
% Turnpoints: vector with the x-values of the inflection points
% Extreme points: Vector with the x-values of the extreme points

% create x-values
x = [range(1):range(3):range(2)];

% create y-values with polyval. polyval(p,x) returns the value of a
% polynomial of degree n evaluated at x. For more details look it up:
% doc polyval.

y = polyval(polynom, x);

% calculation of the zero points
zero_points = roots(polynom);

%% First derivative and their zero points (extreme points of the function)
%
% polyder does an polynomial differentiation k = polyder(p) returns the
% derivative of the polynomial represented by the coefficients in p.
% It can
% also be used in combination with a product or a qutient of two
% polynomals. Please look it up: |doc polyder| .

polynom_1 = polyder(polynom); % Coefficients of the 1st derivative
y_1 = polyval(polynom_1, x);

% calculation of the zero points (extreme points)
extreme_points = roots(polynom_1);
extreme_points_y = polyval(polynom, extreme_points);

%% Second derivative and their zero points (turning points of the function)

polynom_2 = polyder(polynom_1); % % Coefficients of the 2nd derivative
y_2 = polyval(polynom_2, x);

% calculation of the zero points (turning points)
turning_points = roots(polynom_2);
turning_points_y = polyval(polynom, turning_points);

end
```

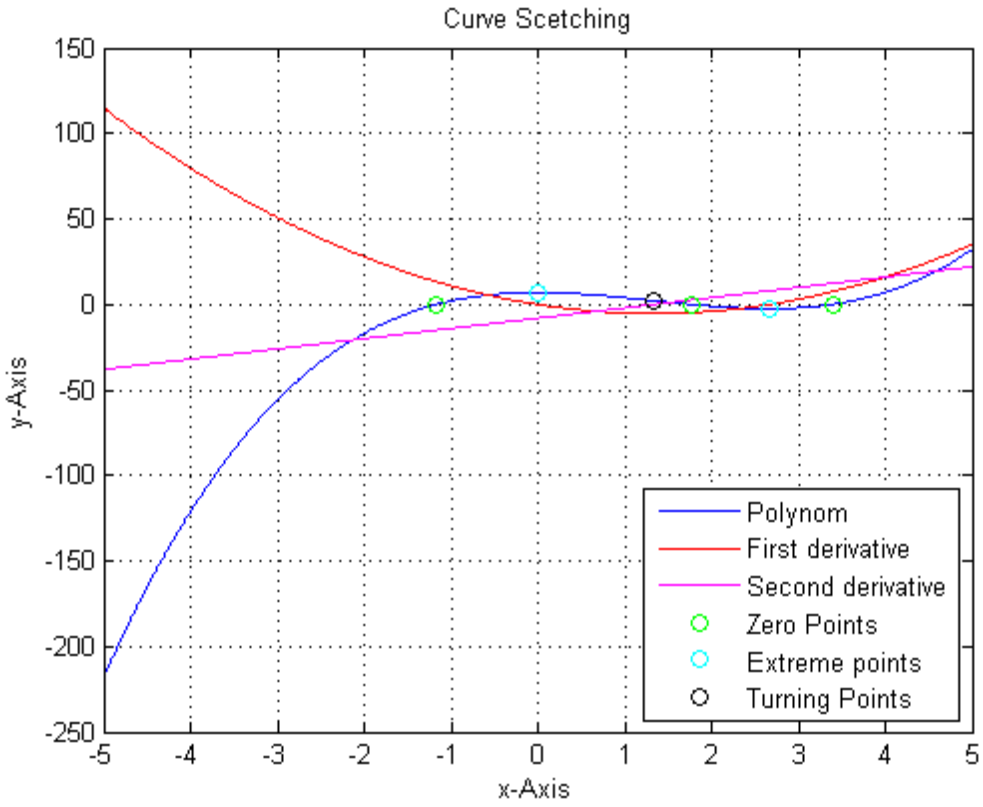
The example function is $x^3 - 4x^2 + 7$. The values for x will be -5 to 5 in steps of 0.01.

```
[x, y, y_1, y_2, zero_points, extreme_points, extreme_points_y, ...
    turning_points, turning_points_y] = ...
    f_curve_scetching([1, -4, 0, 7] , [-5, 5 0.01]);

zero_points
extreme_points
turning_points
```

```
zero_points =

    3.3914
    1.7729
   -1.1642


extreme_points =

         0
    2.6667


turning_points =

    1.3333
```

The functions will now be plotted and the extreme points will be marked inside the plot:

```
function f_plot_curve_scetching( x, y, y_1, y_2, zero_points,...
   extreme_points, extreme_points_y, turning_points, turning_points_y)
%F_PLOT_CURVE_SCETCHING Plots the output of f_curve_scetching

figure,

% Plot the function and its two derivatives
plot(x, y, 'b-');
hold('on');
plot(x, y_1, 'r-');
plot(x, y_2, 'm-');

% Label the extreme points
scatter(zero_points, zeros(1,length(zero_points)), 'go');
scatter(extreme_points, extreme_points_y, 'co');
scatter(turning_points, turning_points_y, 'ko');

% Title, axes, etc.
grid('on');
title('Curve Scetching');
xlabel('x-Axis');
ylabel('y-Axis');
legend('Polynom','First derivative', 'Second derivative','Zero Points',...
  'Extreme points', 'Turning Points',4);
hold('off');

end

f_plot_curve_scetching(x, y, y_1, y_2, zero_points, extreme_points, ...
    extreme_points_y, turning_points, turning_points_y)
```



Debugging

Diagnose problems with programs

You can diagnose problems with your MATLAB® program either graphically or programmatically. Both approaches allow you to set breakpoints to pause the execution of your MATLAB program so you can examine values where you think a problem could be.

To set a breakpoint you can click on the Set/Clear in the Breakpoint menue or simply click right next to the number you want the script or function to pause. A red dot indicates a valid breakpoint, if the point is grey, you have probably not saved your file.

When you run your program it will stop at the first breakpoint you have set and you will discover some new menue items. The most important are **Step** (F10) and **Step in** (F11). Step will go step by step through your code but will not follow the code into functions. Step in will do so.

Please get familiar with the different debugging opportunities. To stop the debug mode, simply click on **Quit Debugging** and clear all breakpoints.

Published with MATLAB® R2013a