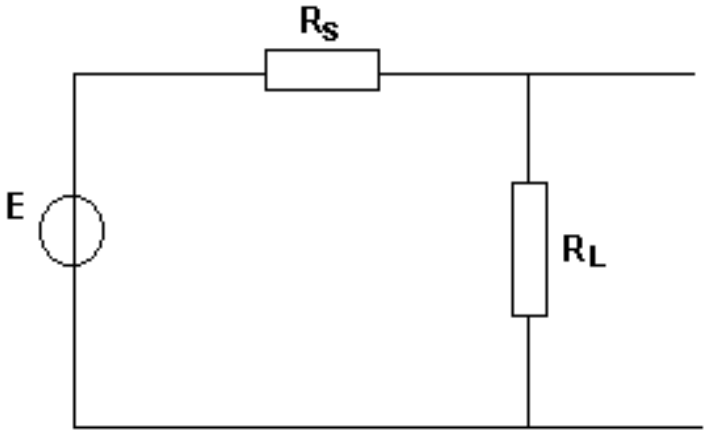
Mathematical Modelling - assignments Week 3: Functions in Matlab & Numerical integration

Functions

Given is the following electronical circuit:



The voltage source has a constant voltage E.

The power dissipated by R_L is given by the formula: $P = \left(\frac{E}{R_S + R_L} \right)^2 \cdot R_L$

Imagine that you would like to calculate the power P for a lot of different values E, R_S, and R_L.
Then you should make a function file, which is an m-file (extension ".m") that contains the definition of the function.
The advantage of such a function file is that you must create an implementation of the function only once, and that you can call that function with different values as often as you like.

An implementation of such a function is given below:

```
function P=calculatePower(E,Rs,RL)
% This function calculates P for all the given values E, Rs and RL.
% These values can also be given in vector form.
  P = ((E./(Rs+RL)).^2).*RL;
end
```

The specific "m-file name" must be equal to the name of the function.
So in this case the name of the function file should be `calculatePower.m`

The comment lines following the function header will appear as help text if your enter `help` followed by the function name on the command line of the command window. It is a good habit to always add those comment lines.

You can now make a script-file `scriptCalculatePower.m` in which you can call the predefined function `calculatePower`.

```
% This script calculates in different situations the power.
situation1 = calculatePower (10, 4, 1)
situation2 = calculatePower (10, 4, linspace(2,8,10)')
E = 12;
Rs = 2;
RL = 0.3:0.3:3;
P = calculatePower(E,Rs,RL);
```

```
output = [RL',P']
```

The resulting output of this script would be:

```
situation1 =

     4

situation2 =

    5.5556
    6.0000
    6.1983
    6.2500
    6.2130
    6.1224
    6.0000
    5.8594
    5.7093
    5.5556

output =

    0.3000    8.1664
    0.6000   12.7811
    0.9000   15.4102
    1.2000   16.8750
    1.5000   17.6327
    1.8000   17.9501
    2.1000   17.9893
    2.4000   17.8512
    2.7000   17.6007
    3.0000   17.2800
```

An example of a function that calculates the sum of all the positive elements of a vector, is given below:

```
function y = sumpos(x)
% This function calculates the sum of all positive elements of an input vector
  help1 = x>0;
  help2 = x.*help1;
  y = sum(help2);
end
```

The name of the function file should be `sumpos.m`

Example script:

```
clear
v = [1 -3 4 2 -4 -5 1 -1];
p = sumpos(v)
```

Result:

```
p = 8
```

Note that `x`, `y`, `help1` and `help2` are NOT known in the Workspace after calling the function; their scope is limited to the function only!

Another example of a function with two input values and two (!) output values is given below.
The function `multiply` calculates the product of two input matrixes, but only if the input matrixes have the correct dimensions, i.e. the inner-dimensions must agree.
In any other case it gives an error (as its second return value), and the output becomes **`NaN`**, which means **`Not a Number`**.

```
function [Product, Txt] = multiply(A,B)
% This function calculates the product of two matrices A and B
% It will give the also as second output value 'OK'
% If the values don't agree it will return NaN as first output value
% and as second output value the message 'Inner dimensions don't agree'
  sa = size(A);
  sb = size(B);
  if sa(2)==sb(1)
    Product = A*B;
    Txt = 'OK';
  else
    Product = NaN;
    Txt = 'Inner dimensions don't agree';
  end
end
```

The name of the function file should be `multiply.m`

Example script:

```
A =[2, 3; 4, 5]
B =[1, 2, 3; 4, 5, 6]
[AB, result1]= multiply(A,B)
[BA, result2]= multiply (B,A)
```

Result:

```
A =
     2     3
     4     5


B =
     1     2     3
     4     5     6


AB =
    14    19    24
    24    33    42


result1 =
    'OK'


BA =
   NaN


result2 =
    'Inner dimensions don't agree'
```

Final example:

Given is the following function:

$$fun(x)=\begin{cases}\tan^{-1}(x+0.34) & \text{if} \quad x\le -3\\ 1.2\cdot\ln\left(\dfrac{\sqrt{x^2+4}}{x^2+1}\right) & \text{if} \quad -3<x\le 0\\ 0.1664\cdot e^{-2x}\left(5\cos(3x)-4\sin(3x)\right) & \text{if} \quad x>0\end{cases}$$

You need to first to calculate separate individual function values for `x=-3, x=2, x=5` and `x=10`, and then for the interval `[-5,5]` you need to make a graph of the entire function. With a function file you only must define the function once and can be used to call it multiple times with ease by calling upon its name.

```
function y = fun(x)
% This function is composed of three different parts
  help1 = atan(x+0.34);
  help2 = 1.2*log(sqrt(x.^2+4)./(x.^2+1));
  help3 = 0.1664*exp(-2*x).*(5*cos(3*x)-4*sin(3*x));
  y = help1.*(x<=-3)+help2.*(x>-3).*(x<=0)+help3.*(x>0);
end
```

The name of the function file should be `fun.m`

Example script:

```
y_min3 = fun(-3)
y_2 = fun(2)
y_5 = fun(5)
y_10 = fun(10)
x = linspace(-5,5,100);
y = fun(x);
plot(x,y)
grid on
title('A function composed of three different parts')
xlabel('-5 \leq x \leq 5') % x-axis label
ylabel('y values') % y-axis label
shg
```

Results:

```
y_min3 =
   -1.2112
y_2 =
    0.0180
y_5 =
  -4.8346e-05
y_10 =
   1.6200e-09
```

A few notes about "user defined function files":

- Function header: function [output] = functionname (input)
- You can't 'run' or 'execute' a function (like in C)
- `Filename = functionname.m`
- Function name: don't use spaces, only use letters and/or numbers, don't start with a number.
- **Always add help text!**
- A function should not have side effects. It is not a very good idea to use plot-commands in a function. It is better to use functions for calculations of results and plot the results after having called the function.
- Each calculation with a function file ends also with a semicolon ; or else each time you call the function it will show the result of calculations made in the function file.
- Calling a function from a script is done like:
  `y = functionname(x);` or `[a, b] = functionname(x1, x2, …);`
- Variables inside the function are "local"; this means that they are only active inside the function and unknown outside the function.

For the following assignments you have to make several functions. For each function you have to make a separate MATLAB file. The scripts, which call these functions, should be collected in one single assignment file. This assignment file has to be submitted together with The MATLAB files of the functions.

Assignment 1

Make a function file for the function `squaredSum`.
The function has as input variable a vector x.
The output variable is the number y, which is the sum of the elements of x squared.
Don't forget to add a few comment lines, to explain what the function does.
Hint: you are allowed to use the command `sum` (see help).

Test your function with a script with x a vector of the integer numbers 1 to 10.

Assignment 2

Make a function file of the function `replacementResistance`
That function has the input variable: the vector R of resistances R1, R2, ... Rn.
The output variable should be Rv, which should be equivalent to the circuit in which all resistors R1 to Rn are in parallel.
Hence, the following formula applies: $\frac{1}{Rv} = \frac{1}{R1} + \frac{1}{R2} + \cdots + \frac{1}{Rn}$
Don't forget to add a few comment lines to explain what the function does.

Test your function for R = [20, 24, 27, 50, 32].

Assignment 3

Make a function file of a function with input variable x and output variable y, in which y is defined as the function $y(x) = 0.9x^4 - 12x^2 - 5x$
Write the function in such a way that input `x` could also be a vector.
Don't forget to add a few comment lines, to explain what the function does.

Calculate with this function the function-values for `y(-3)` and `y(5)`.
Then make a plot of `y` as a function of `x`, when x has 100 values between −4 and 4.

Assignment 4

Make a function file of the function `calculateAngle`. Since already a function with the name `angle` exists, a different name has been chosen.
The function `calculateAngle` has two input variables: the vectors a and b.
Also it has two output variables: the real number y and the text txt.
In the function you should first check if the variables a and b have the same dimensions.
If that is the case, the text txt will get the value 'OK', and y will get the value of the angle (in rad) between the vector a and b.
If the dimensions do not agree txt should get the value 'Error', and y should become the value 'NaN'.
Don't forget to add a few comment lines, to explain what the function does.

Test your function with a script for the vectors a = `[1,2,4] and b = [2,-3,1]`

Hint: the formula to calculate the angle between two vectors is $\cos^{-1}\left(\frac{a \cdot b}{|a|\,|b|}\right)$

Assignment 5

Make a function file with the name `derivative.m`
The function has two input variables and one output variable

- Input variable x is the vector of the x-values
- Input variable y is the vector of the output values of a certain function applied to x.
- Output variable dydx is the vector with the numeric calculated derivative of the input vector y of x (by using the command `diff`), with an added value at the end, to make the dimension (length) of dydx the same as the dimension of x. Think of a smart way to fill in this last value so that it fits in with the rest of the values.

The function is given $f : x \to x \cdot \sin\left(\frac{x^2}{10}\right)$

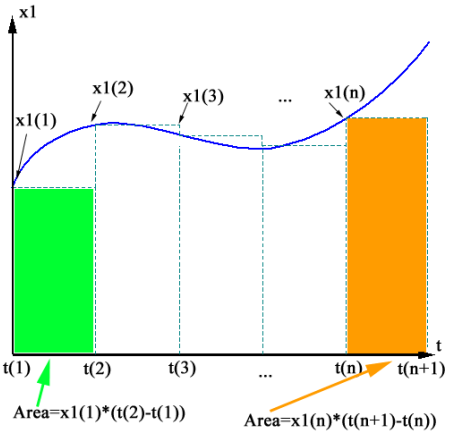
Make a script-file in which the following tasks are included:

- Make a vector x with 100 numbers ranging from -1 to 5, with equal spacing.
- Calculate the function values for y = f(x)
- Calculate the differentiated values of the function f for the interval [-1,5]. For calculating the differentiated values, you should use your previously defined function `derivative`.
- Generate two plots using subplot: one subplot of f as function of x, and one subplot of the derivative of f as function of x;

Numerical Integration

Sometimes integrals can easily be solved analytically. However, sometimes it is very hard or even not possible. In that case numerical integration can give an answer. There are several methods known. In this assignment we will see a few of them.

Riemann method



Matlab can also be used to approximate the process of integration. To find the surface below the curve we make an approximation by means of rectangles. This is called Riemann Integration. A summation of the rectangles gives an approximation of the integral.
To calculate the area below the curve we need to know the minimum and maximum t-value (t_min,t_max).
The area of the rectangle is not exactly equal to the area below the graph. For smaller Δt values the error will become smaller.
The number of rectangles has great influence on the error. Next example shows the Matlab code to calculate the following defined integral $\int_1^4 (2t+2)dt$

```
t_min=1; t_max=4; number_of_rectangles =1000;
delta_t=(t_max-t_min)/number_of_rectangles;
area=0;
t=t_min;
% each time we calculate the area of one rectangle + summation
for counter=1:number_of_rectangles
    area = area+(2*t+2)*delta_t;
    t = t+delta_t;
end
area % displays the area
```

Since the given integral is not that difficult, you can also calculate it manually and compare the numerical result with the exact result.

The summation is done by the command area=area+(2*t+2)*delta_t inside the for loop. An easier way to perform the summation can be done with the built-in sum function, giving the following code:

```
t_min=1; t_max=4; number_of_rectangles = 1000;
delta_t=(t_max-t_min)/number_of_rectangles;
t=t_min:delta_t:t_max;          % this results in 1001 time values
t=t(1: number_of_rectangles); % this results in 1000 time values
f_t=2*t+2;
total_area=sum(f_t.*delta_t)  % displays the area
```

In the code the time `t_max` itself is removed, since only the left corners of the rectangles are used.

Trapezium method

Working with rectangles only is not always accurate enough. A better approximation can be obtained by using the trapezium rule. Matlab has a built-in function called trapz.
The code using this function is:

```
t_min=1; t_max=4; number_of_trapeziums = 1000;
delta_t=(t_max-t_min)/number_of_trapeziums;
t=t_min:delta_t:t_max;          % this results in 1001 time values
f_t=2*t+2;
total_area=trapz(t,f_t)         % displays the area
```

Note that `t_max` is not removed, since both left and right corners of the trapeziums are used.

Assignment 6

Make a plot of the graph of the function $f(x) = x^2 + 4x$ on the interval of 2 to 5.

First manually calculate the outcome of the integral of $\int_2^5 \left(x^2 + 4x\right)dx$

Write your manual calculation as comment in the Matlab file; you should not only give the value, but also the calculation steps as comment.

Now we want to calculate the integral of $\int_2^5 \left(x^2 + 4x\right)dx$ using numerical integration by means of the built-in function `trapz` to estimate the numerical value of this integral.

First do this with 2 values for x, i.e. a single trapezium.
Is the result too big or too small, compared to the actual value of this integral? Give your answer in the form of Matlab comment.
Next do this with 3 values for x, i.e. two trapeziums.
Then do this with 100 values for x
And finally do this with 1000 values for x.

Assignment 7

Estimate the value of the following integral by means of numerical integration with the help of `trapz`.
First, use 10 values for x to calculate the integral, then double the amount of values for x until the result is no longer changing for the first four decimal digits.

$$\int_0^2 \left(1 + \cos\left(x^2 + 1\right)\right)dx$$

Intermezzo: Inline functions

In Matlab you have the possibility to define inline functions (also called anonymous functions), if you don't want to create a separate function-file. There are also other useful applications as we will see later.

An example is the following definition.

```
>> sqr = @(x) x.^2;
```

This means that `sqr` is a function with argument x and has as result the x squared. In the way `sqr` is defined, it can also take a vector as input.

You now can use type

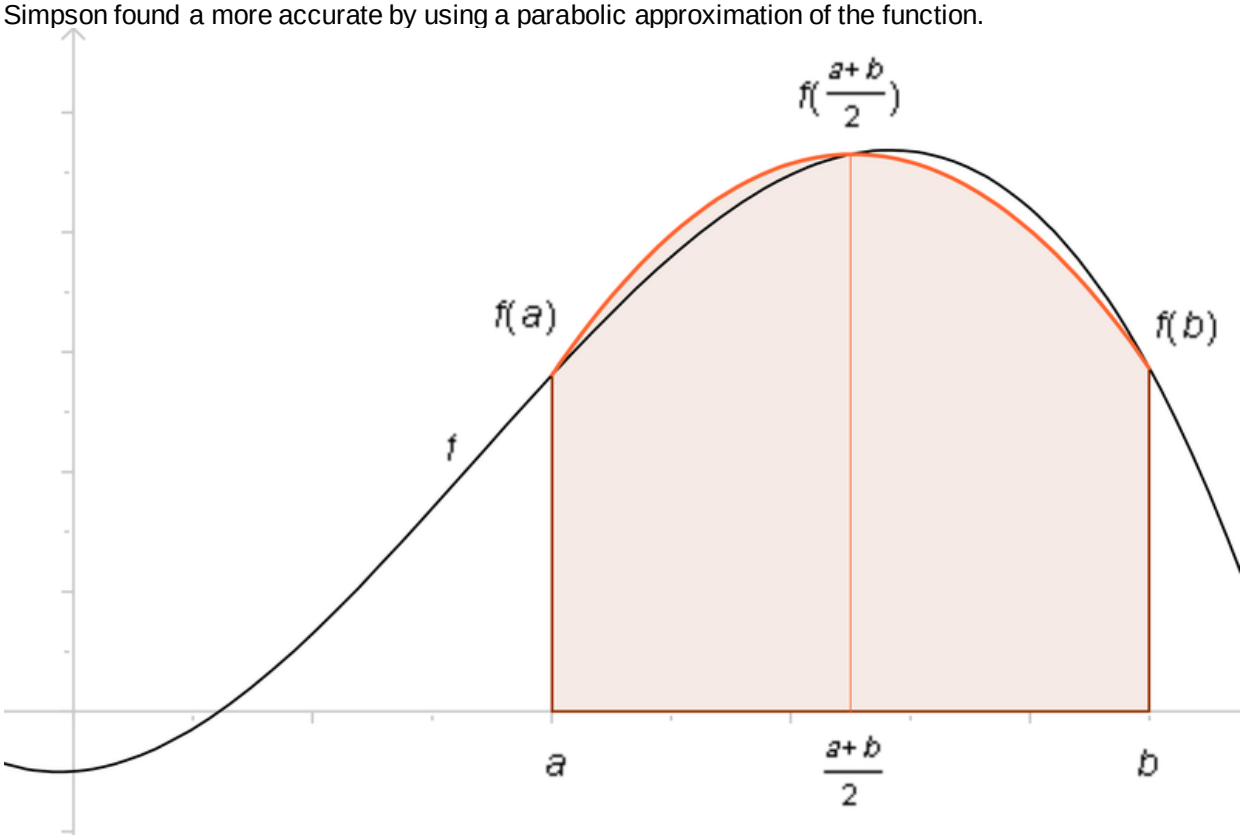
```
>> sqr(1:4)

ans =

     1     4     9    16
```

Simpson rule

Simpson found a more accurate by using a parabolic approximation of the function.



When using 2 trapeziums, a good approximation can be calculated by:

$$S = \frac{(b-a)}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right]$$

When using $n \geq 4$, with n an even number, the approximation would be:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + ... + 2f_{n-2} + 4f_{n-1} + f_n \right]$$

There is no built-in function in Matlab to perform integration with the Simpson rule.
The source code of a user defined function simpson is given below:

```
function result = simpson(f_x,lower,upper,number_of_trapeziums)
% % Syntax:
% % simpson(f_x,lower_limit,upper_limit,number_of_trapeziums)
% % f_x should be an inline function
% % Beware: number of trapeziums must be even (2, 4, 6, ...)
% %
% % Example: simpson(@(x) x.^2,1,4,4)
  a = lower; % lower limit
  b = upper; % upper limit
  n = number_of_trapeziums;
  h = (b-a)/n; % integration stepsize
  x = a:h:b;   % arguments for f_x

  sum= f_x(x(1)); % assign first function value to sum
  for i=2:n
      if mod(i,2)==0
          sum = sum +4* f_x(x(i));
      else
          sum = sum +2* f_x(x(i));
      end
  end
  sum = sum + f_x(x(n+1)); % add last function value to sum

  result=h/3*sum;
end
```

Assignment 8

Estimate the value of the following integral by means of numerical integration using the Simson rule (making use of the function `simpson` defined above).
Take for each integral first 10 trapeziums, then double the number of trapeziums until the result is no longer changing for the first four decimal digits.

$$\int_0^2 e^{1-x^2}\,dx$$

Built-in function *integral*

Matlab has also a built-in function `integral`, which makes it even easier to perform numerical integration.

To be able to use this built-in function `integral`, we must define an inline function for the function of which we want to calculate the integral, just like when using the Simpson function.

Example:
Let us take the function $f(x)=1+\cos\left(x^2+1\right)$, and calculate the integral of this function using lower limit 0 and upper limit 2, i.e. $\int_0^2 1+\cos\left(x^2+1\right)dx$

```
>> fun = @(x) 1+cos(x.^2+1);
>> integral(fun,0,2)
```

Assignment 9

Use the built-in function `integral` to calculate

$$\int_0^2 e^{1-x^2}\,dx$$

Assignment 10

Use the built-in function `integral` to calculate

$$\int_2^5 \left(x^2 + 4x\right)dx \text{ d}$$