Project 2 - Approximation of real functions and solution of non-trivial equations

Aladdin Persson (alhi0008@student.umu.se)

Version 1, Username: tfy17api
Teknisk-vetenskapliga beräkningar
Supervisors: Carl Christian Kjelgaard Mikkelsen,
Niclas Börlin

Contents

1 Introduction

For this project we address the theory of important theorems and their visualisation in Section 2. Secondly methods of numerically approximating derivatives with the use of Taylor's polynomials and also how error decreases with the step size between the datapoints. This is then compared to the theorethical values in Section 3. Lastly we examine the unavoidable case of having a limited number of sample points for functions and how we can generate accurate approximations with the usage of Hermite's piece-wise approximation. In further depth this is found in Section 4 where the Hermite's approximation is determined given sample points. This can be used in realistic scenarios of different types, for example to estimate the position of collision for a trajectory and it's surrounding landscape which is further analyzed in Section 5.

All MATLAB files necessary to run the scripts for this project are included in a separate zip file named P2Code as well as included in the appendix in report.

2 The zero theorems

Rolle's theorem, the intermediate value theorem and the mean value theorem for derivatives are heavily utilized theorems in this part and are important to understand in detail. To prove these important theorems, their proofs are connected to another theorem which we will state below.

The Max-Min Theorem. *If $f(x)$ is continuous on the closed interval $[a, b]$, then there exist numbers $x_0, x_1 \in [a, b]$ such that for all $x \in [a, b]$,*

$$f(x_0) \leq f(x) \leq f(x_1). \tag{1}$$

Where $a, b \in \mathbb{R}$. Hence, f has the absolute minimum value $f(x_0)$ at the point x_0 and the absolute maximum value $f(x_1)$ at the point x_1.

With knowledge about this theorem we can now proceed.

Rolle's theorem. *Suppose that the function g is continuous on the closed, finite interval $[a, b]$. If additionally $g(a) = g(b)$, then there exists a point $c \in (a, b)$ such that $g'(c) = 0$.*

Proof. Case 1: $g(x) = g(a)$ $\forall x \in [a, b]$ then we know $g(x)$ is constant, hence it's derivative will be 0 $\forall c \in (a, b)$.

Case 2: $\exists$ $x \in (a, b)$ such that $g(x) \neq g(a)$. Let assume that $g(x) > g(a)$, then by the Max-Min Theorem, since g is continuous on $[a, b]$ there must exist a maximum value at some point $c \in [a, b]$. Since by known theorem, if the derivative exists at a maximum point then we know that the derivative is zero, hence $g'(c) = 0$.

When $g(x) < g(a)$ the proof is similar. This concludes the proof. □

The Mean-Value Theorem *Suppose that the function f is continuous on the closed, finite interval $[a, b]$, and that it is differentiable on (a, b). Then there exists a point $c \in (a, b)$ such that*

$$\frac{f(b) - f(a)}{b - a} = f'(c). \tag{2}$$

Proof. We know that the linear equation connecting $(a, f(a)), (b, f(b))$ can be written as

$$y = f(a) + \frac{f(b) - f(a)}{b - a}(x - a). \tag{3}$$

We can define

$$g(x) := f(x) - y \tag{4}$$

$$g(x) := f(x) - \Big(f(a) + \frac{f(b) - f(a)}{b - a}(x - a)\Big) \tag{5}$$

We can interpret $g(x)$ as the vertical distance between f and the linear equation y, hence at points a, b we know that $g(a) = g(b) = 0$. We therefore know by Rolle's theorem that $g'(c) = 0$

$$g'(x) = f'(x) - \Big(\frac{f(b) - f(a)}{b - a}\Big) = 0 \tag{6}$$

$$g'(c) = f'(c) - \Big(\frac{f(b) - f(a)}{b - a}\Big) = 0 \tag{7}$$

$$f'(c) = \Big(\frac{f(b) - f(a)}{b - a}\Big). \tag{8}$$

This concludes the proof. □

In the function `MyZeroTheorem.m` which can be found in (Appendix 8.1) we look at the function

$$f(x) = e^x sin(x). \tag{9}$$

We can illustrate these theorems in (Figure. 1) where the legend notation should be consistent with what was used previously in theorems.

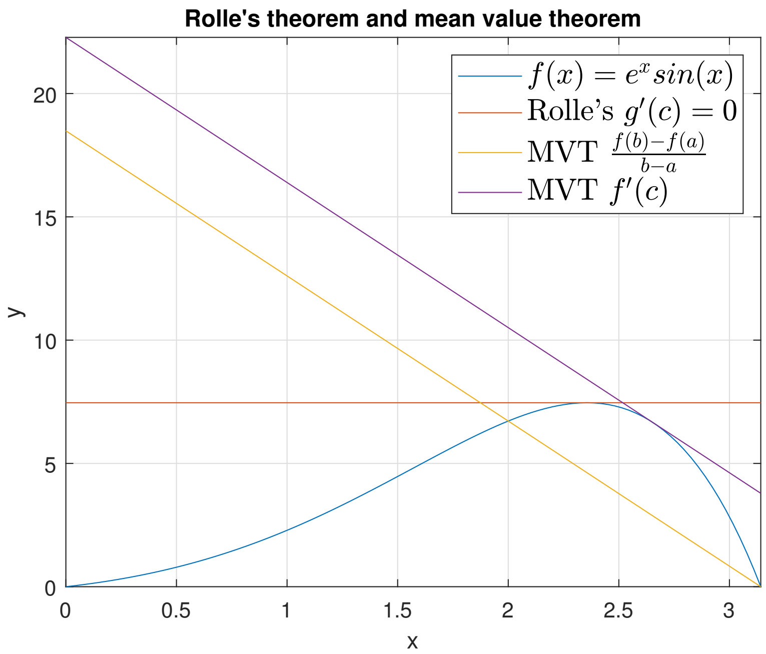

Figure 1 – Plot illustrating Rolle's Theorem and the Mean Value Theorem (MVT). Notice here that for the Mean Value Theorem $b = \pi$ and $a = 2$.

3 Approximation of derivatives

Approximating derivatives numerically is incredibly important because of the fact that they cannot always be found analytically. Also in many cases derivative function evaluations could be an incredibly expensive operation however numerical differentiation is in many cases a cheaper operation. To derive a formula for numerical differentiation we use Taylor's theorem.

Taylor's Theorem *Let $f : \mathbb{R} \to \mathbb{R}$ be an $(n+1)$ times differentiable on an open interval containing the points a and x. Then we know*

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + ... + \frac{f^n(a)}{n!}(x-a)^n + R_n(x) \quad (10)$$

where $R_n(x) = \frac{f^{n+1}(c)}{(n+1)!}(x-a)^{(n+1)}$ for some number c between a and x.

We wish to show that

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2) \quad (11)$$

Where $\mathcal{O}$ stands for the Big-O notation. We can make a taylor expansion for $f(x+h)$ with $a = x$ in formula for taylor polynomial (Eq. 10).

Proof. Let $n = 2$ in taylor expansion for $f(x+h)$ then when we let $h \to 0_+$ we get

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f^3(c)}{3!}h^3 \quad (12)$$

$$f(x-h) = f(x) + f'(x)(-h) + \frac{f''(x)}{2!}(-h)^2 + \frac{f^3(c)}{3!}(-h)^3. \quad (13)$$

Then we can see that

$$f(x+h) - f(x-h) = 2hf'(x) + 2\frac{f^3(c)}{3!}h^3 \quad (14)$$

since we know that c is just a constant we can use Big-O notation and rewrite this as

$$f(x+h) - f(x-h) = 2hf'(x) + \mathcal{O}(h^3) \implies \quad (15)$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2). \quad (16)$$

In the last step we used that $\frac{\mathcal{O}(2h^3)}{h} = \mathcal{O}(2h^2) = \mathcal{O}(h^2)$. Larger values of n for the taylor expansion would give the same result as the Big-O would be the same as $h \to 0_+$.

□

We wish to show that

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} + \mathcal{O}(h^2) \tag{17}$$

Proof.

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f^3(c)}{3!}h^3 \tag{18}$$

$$f(x+2h) = f(x) + f'(x)2h + \frac{f''(x)}{2!}(2h)^2 + \frac{f^3(c)}{3!}(2h)^3 \tag{19}$$

$$4f(x+h) - f(x+2h) = 3f(x) + 2hf'(x) + \mathcal{O}(h^3) \tag{20}$$

Hence,

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} + \mathcal{O}(h^2) \tag{21}$$

□

We wish to also show that

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} + \mathcal{O}(h^2) \tag{22}$$

Proof.

$$f(x-h) = f(x) + f'(x)(-h) + \frac{f''(x)}{2!}(-h)^2 + \frac{f^3(c)}{3!}(-h)^3 \tag{23}$$

$$f(x-2h) = f(x) + f'(x)(-2h) + \frac{f''(x)}{2!}(-2h)^2 + \frac{f^3(c)}{3!}(-2h)^3 \tag{24}$$

$$f(x-2h) - 4f(x-h) = -3f(x) + 2hf'(x) + \mathcal{O}(h^3) \tag{25}$$

Hence,

$$f'(x) = \frac{f(x-2h) - 4f(x-h) + 3f(x)}{2h} + \mathcal{O}(h^2) \tag{26}$$

□

One can note here that they the error term $\mathcal{O}(h^2)$ will decay equally as quickly for all three formulas derived for the approximated derivative. However they require different sample points, a formula such as (Eq. 16) requires to be in the center and having points to the left and right respectively. Formula such as (Eq. 17) requires only points to the right and (Eq. 26) requires left sample points only. These three equations can each be used to approximate the derivative with theory providing increasingly good accuracy depending on the sample point. This fact is used in

MyDerivs.m and code can be found in (Appendix 8.2).

A minimal working example for numerically approximating the derivative of

$$f(x) = e^x sin(x), \tag{27}$$

in the interval $[0, 1]$ can be found in MyDerivsMWE1.m and code can be found in (Appendix 8.3). When comparing log_{10} of the relative error between the approximated derivative true derivative

$$f'(x) = e^x sin(x) + e^x cos(x), \tag{28}$$

we obtain

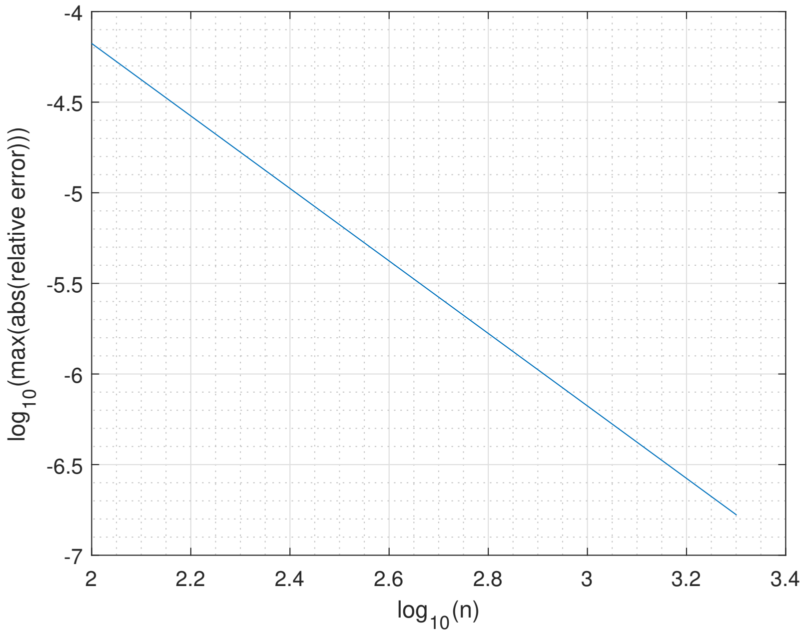

Figure 2 – This shows how the relative error is decreased when increasing the amount of sample points and therefore by decreasing the step size h.

Here we notice that as the amount of sample points go from $10^2 \rightarrow 10^3$ which means that $h = \frac{1-0}{10^2} \rightarrow \frac{1}{10^3}$. Hence h, the step size is decreased by a factor of $\frac{1}{10}$. We also notice that the error decreases with about 10^{-2} and hence the theorethical $\mathcal{O}(h^2)$ seems to be well supported in this example.

4 Hermite's piece-wise approximation

Given $x_0, x_1, ..., x_n$ and the corresponding $f(x_0), f(x_1), f(x_n)$ we can utilize methods of polynomial interpolation to approximate $f(t)$ where $x_j < t < x_{j+1}, j \in \{0, 1, ..., n-1\}$. A specific method of doing polynomial interpolation is called Hermite's piece-wise approximation which uses additional information in the form of using $f'(x_0), f'(x_1), f'(x_n)$ can be used to approximate functions in values of t with better accuracy. Specifically we can use the following polynomials

$$p_0(t) = (1+2t)(1-t)^2 \tag{29}$$
$$p_1(t) = t^2(3-2t) \tag{30}$$
$$q_0(t) = t(1-t)^2 \tag{31}$$
$$q_1(t) = t^2(t-1) \tag{32}$$

where it can be shown, either by differentiating by hand and plugging in values, or it is also possible to use MATLAB's built in differentiator that

$$\begin{aligned}
&p_0(0) = 1,\ p_0(1) = 1,\ p_0'(0) = 0,\ p_0'(1) = 1, \\
&p_1(0) = 0,\ p_1(1) = 1,\ p_1'(0) = 0,\ p_1'(1) = 0, \\
&q_0(0) = 0,\ q_0(1) = 0,\ q_0'(0) = 1,\ q_0'(1) = 0, \\
&q_1(0) = 0,\ q_1(1) = 0,\ q_1'(0) = 0,\ q_1'(1) = 1
\end{aligned}$$

and this can be seen in the script `derivatives_hermites.m` that can be found in (Appendix 8.6). By assuming that the function $f : [a, b] \to \mathbb{R}$ is differentiable and that f' is continuous we can write Hermite's approximation of the function as $p : [a, b] \to \mathbb{R}$ given by

$$p(x) = f(a)p_0(\phi(x)) + f(b)p_1(\phi(x)) + f'(a)(b-a)q_0(\phi(x)) + f'(b)(b-a)q_1(\phi(x)) \tag{33}$$

where $\phi(x) = \frac{x-a}{b-a}$. Therefor it is easy to see that $\phi(a) = 0, \phi(b) = 1$. Hence we can easily show that

$$p(a) = f(a)p_0(0) + f(b)p_1(0) + f'(a)(b-a)q_0(0) + f'(b)(b-a)q_1(0) \tag{34}$$
$$p(a) = f(a) \cdot 1 + 0 + 0 + 0 \tag{35}$$
$$p(a) = f(a) \tag{36}$$

and similarly

$$p(b) = f(a)p_0(1) + f(b)p_1(1) + f'(a)(b-a)q_0(1) + f'(b)(b-a)q_1(1) \tag{37}$$
$$p(b) = 0 + f(b) \cdot 1 + 0 + 0 \tag{38}$$
$$p(b) = f(b). \tag{39}$$

Note that

$$\phi'(x) = \frac{1}{b-a} \tag{40}$$

$$p'(x) = f(a)\frac{p_0(\phi(x))}{b-a} + f(b)\frac{p_1(\phi(x))}{b-a} + f'(a)q_0(\phi(x)) + f'(b)q_1(\phi(x)) \tag{41}$$

and therefore we can see that

$$p'(a) = f(a)\frac{p_0(0)}{b-a} + f(b)\frac{p_1(0)}{b-a} + f'(a)q_0(0) + f'(b)q_1(0) \tag{42}$$

$$p'(a) = 0 + 0 + f'(a) \cdot 1 + 0 \tag{43}$$

$$p'(a) = f'(a) \tag{44}$$

and similarly that

$$p'(b) = f(a)\frac{p_0(1)}{b-a} + f(b)\frac{p_1(1)}{b-a} + f'(a)q_1(0) + f'(b)q_1(0) \tag{45}$$

$$p'(b) = 0 + 0 + 0 + f'(b) \cdot 1 \tag{46}$$

$$p'(b) = f'(b). \tag{47}$$

We can therefore approximate f by using that

$$\forall x \in [x_{j-1}, x_j] : p(x) = p_j)(x) \tag{48}$$

where p_j is Hermite's approximation of f corresponding to sub-interval $[x_{j-1}, x_j]$. By design we have that p is both differentiable and that it's derivative p' is continuous. An implementation of Hermite's approximation can be found `MyPiecewiseHermite.m` which can be found in (Appendix. 8.4). A minimial working example for approximating the function

$$f(x) = e^x sin(x) \tag{49}$$

is found in `MyPiecewiseHermiteMWE1.m` and code at (Appendix. 8.3). The code generates the approximation with increasing number of sample points from $[10, 200]$. The approximations are compared to the true function values by a relative error which gives (Fig. 3).

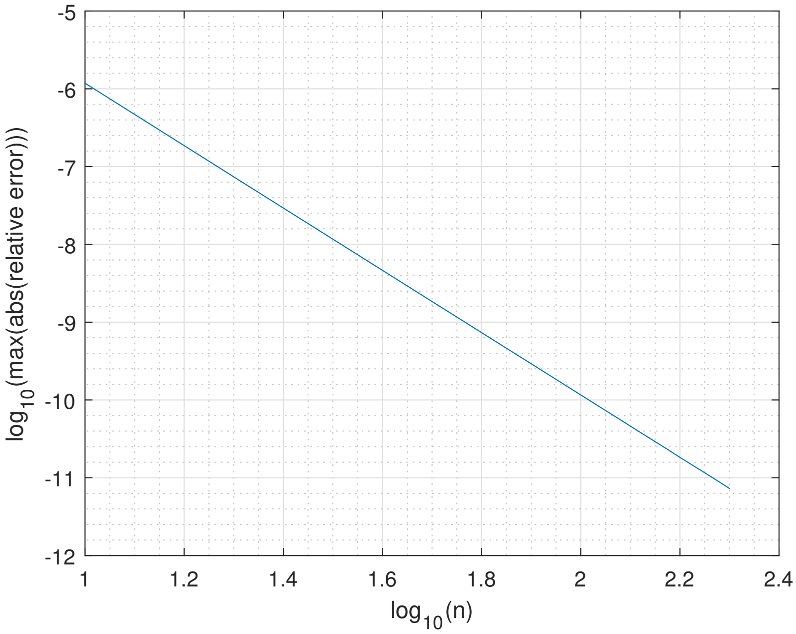

Figure 3 – Illustration of the relative error decay between $f(x)$ and approximated $p(x)$ by Hermite's approximation, as the stepsize is decreased by a larger amount of sample points.

The interval for the comparison is in $b = 1, a = 0$ which corresponds to when we have 10 sample points $h = 0.1$ and similarly when we have $10^2 = 100$ sample points then $h = 0.01$. By (Fig. 3) we notice that the relative error decreases by 10^{-4} for a $\frac{1}{10}$ rate decay in step size. This suggests that the error for this example decays as $\mathcal{O}(h^4)$.

5 Event location for ordinary differential equations

Ordinary differential equations has many practical applications, among them is solving equations related to trajectories. By solving an ordinary differential equation of the form

$$\gamma'(t) = f(t, \gamma(t)) \tag{50}$$

where $\gamma(t) = (x(t), y(t), x'(t), y'(t))^T$.

If we imagine the trajectory displaying an artillery shell then by solving the differential equation (Eq. 50) we obtain the position $(x(t), y(t))$ as well as the velocity $(x'(t), y'(t))$. We can define

$$g(\gamma(t)) := 0 \tag{51}$$

where g is called an event function and we say that an event has occured at time t if $g(\gamma(t)) = 0$. We assume here that g is defined for all $z \in \mathbb{R}$, where $g(z) = g(z_1, z_2, z_3, z_4)$. For example we can solve

$$g(z) = z_2 - c \implies y(t) = c \tag{52}$$

which is equivalent to finding the time t where the shell reaches height c. Let us consider the case where we wish to find the position where the shell hits an arbitrary landscape. If we consider $h(x)$ be a function which represents the height of the landscape then if we let the event function be

$$g(z)z_2 - h(z_1) \implies y(t) - h(x(t)) = 0 \tag{53}$$

which is equivalent to solving t where the shell hits the landscape, i.e the ground. In realistic scenarios when simulating these shells, since we are dealing with computers which do not have infinite memory, it is an impossibility of having continuous functions. Therefore it is not possible to have exactly $y(t)$ or even $h(x(t))$ but rather we have a limited amount of sample points and how many is depedent mainly upon the step size in the differential equation solver used to solve (50). It is however possible to utilize Hermite's approximation to obtain an approximation of these trajectories for $t \in \mathbb{R}^+$. This means that we can also find a very good approximation where the shell hits the ground. Function that defines landscape is given by `simple_landscape.m` in (Appendix. 8.7). A minimal working example of solving (53) is found in `MyEvent.m`. The resulting figure when this is solved is found by (Fig.4).

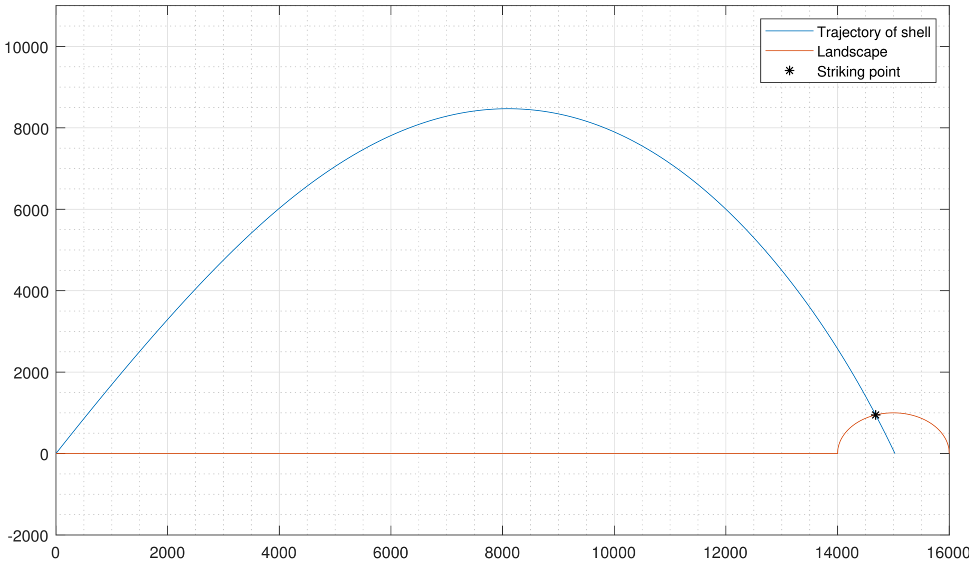

Figure 4 – The trajectory of the shell and the spherical landscape as well as the hit point where the shell collides with the landscape.

6 Conclusions

From this project we have seen examples of how Rolle's theorem and the mean value theorem of differentiation can be visualized. We have also seen how numerical differentiation can be done in different ways depending on the sample point position and how it can achieve quite good accuracy by decreasing stepsize h. Perhaps most interestingly we have also seen the usage of how approximating functions can be done with Hermite's piece-wise approximation as well as how this can be used in practice for calculating accurate approximations of trajectories of shell colliding with a landscape. One interesting improvement of this project would be to increase additional functionality when approximating derivatives by also finding the optimal stepsize h. Also it would be interesting to see how Hermite's approximation could be extended with additional information about the second derivative, third, etc.

7 References

References

[1] Carl Christian Kjelgaard Mikkelsen: *An Introduction to Scientific Computing*, Department of Computing Science, Umeå University (2018)

8 Appendix

8.1 Appendix 1 - MyZeroTheorem.m

Source code for MyZeroTheorem.m.

```
clear all; close all; clc;

% Purpose of this script is to illustrate Rolle's theorem and the mean
% value theorem for differentiation.

% Programming by Aladdin Persson (alhi0008@student.umu.se)
%   2018-12-08 Initial programming

% Define a nice function
f=@(x)exp(x).*sin(x);

% Define the derivative fp (fprime) of f
fp=@(x) exp(x).*sin(x)+exp(x).*cos(x);

% Interval
a=0; b=pi;

% Number of subintervals
n=100;

% Sample points for plotting
x=linspace(a,b,n+1);

% Plot the graph
h=figure; plot(x,f(x));

% Hold the graph
hold on;

% Turn on grid
grid on;

% Axis tight
axis tight

```

```
% ///////////////////////////////////////////////////////////////////////
% Illustration of Rolle's theorem
% ///////////////////////////////////////////////////////////////////////

%% Initial search bracket
x0=0;
x1=pi;

% The function values corresponding to the initial search bracket
% fp0, fp1 different signs means we can use bisection for this
fp0=fp(x0);
fp1=fp(x1);

% Tolerances and maxit for bisection.
delta=1e-8; eps=1e-8; maxit=1000;

% Run the bisection algorithm to find the zero c of fp
[c,flag, it, a, b, his, res] = bisection(fp,x0,x1,fp0,fp1,delta,eps,maxit,false);

% Define the tangent at this point; this a constant function.
w=@(x)ones(size(x))*f(c);

% Plot the tangent
plot(x,w(x))

% ///////////////////////////////////////////////////////////////////////
% Illustration of the mean value theorem
% ///////////////////////////////////////////////////////////////////////
```

```
% Define points for corde
x0=2; x1=pi;

% Compute corresponding function values
f0=f(x0);
f1=f(x1);

% Define the linear function which connects (x0,f0) with (
    x1,f1)
% Using point-slope formula
m = (f1-f0)./(x1-x0);
p = @(s) m*(x-x0) + f(x0);

% Plot the straight line between (x0,f0) with (x1,f1)
plot(x,p(x))

% Compute the slope of the corde
yp = m;

% Define an auxiliary function which is zero when fp equals
    yp
g=@(x) fp(x) - yp;

% Run the bisection algorithm to find a zero c of g
[c, flag, it, a, b, his, res] = bisection(g,x0,x1,g(x0),g(
    x1),delta,eps,maxit,false);

% Define the line which is tangent to the graph of f at the
    point (c,f(c))
% Using point slope formula
q = @(s) m.*(x - c) + f(c);

% % Plot the tangent line
plot(x,q(x));

% Labels
xlabel('x'); ylabel('y');

% legend
```

```
h=legend('$f(x) = e^xsin(x)$','Rolle''s $g''(c) = 0$','MVT
    $\frac{f(b)-f(a)}{b-a}$','MVT $f''(c)$', 'FontSize',14);
set(h,'Interpreter','latex');

% title
title('Rolle''s theorem and mean value theorem')

% Print the figure to a file
% print('MyZeroTheorems','-depsc2');
```

8.2 Appendix 2 - MyDerivs.m

Source code `MyDerivs.m`.

```
function fp=MyDerivs(y,h)

% MyDerivs  Computes approximations of derivatives
%
% CALL SEQUENCE: fp=MyDerivs(y)
%
% INPUT:
%   y      a one dimensional array of function values, y = f
%   (x)
%   h      the spacing between the sample points x
%
% OUTPUT
%   fp     a one dimension array such that fp(i)
%   approximates f'(x(i))
%
% ALGORITHM: Space central and asymmetric finite difference
%    as needed
%
% MINIMAL WORKING EXAMPLE: MyDerivsMWE1

% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen (
%   spock@cs.umu.se)
%   2018-11-26 Extracted from a working code

% Programming by Aladdin Persson (alhi0008@student.umu.se)
%   2018-12-08 Initial programming

```

```
% Extract the number of points
m=numel(y);

% The exercise is pointless unless there are at least 3 points
 if m < 3
     return
 end

% Allocate space for derivatives
 fp=zeros(size(y));

% Do asymmetric approximation of the derivative at the left endpoint
 fp(1) = (-3*y(1)+4*y(2)-y(3))./(2*h);

% Do space central approximation of all derivatives at the internal points
% Do a for-loop *before* you attempt to do this as an array operation

%% Following code for using for-loops. Saved here for comparison to
%% vectorized code below.
% for i = 2:m-1
%     fp(i) = (y(i+1)-y(i-1))./(2*h);
% end

% Vectorized code
y1=y(3:m);y2=y(1:m-2);
 fp(2:m-1)=(y1-y2)./(2*h);

% Do asymmetric approximation of the derivatives at the right endpoint
 fp(m) = (3*y(m)-4*y(m-1)+y(m-2))./(2*h);
```

8.3 Appendix 3 - MyDerivsMWE1.m

Source code for `MyDerivsMWE1.m`.

```
% MyDerivsMWE1  Minimal working example for MyDerivs
    function.

% Programming by Aladdin Persson (alhi0008@student.umu.se)
%   2018-12-08 Initial programming

clear all; close all; clc;

% Interval
a=0; b=1;

% Maximum number of iterations
maxit=20;

% Allocate space
n=zeros(maxit,1); mre=zeros(maxit,1);

% Loop over the number of sample points
for j=1:maxit

    % Number of sample points
    n(j)=100*j;

    % Sample points
    x=linspace(a,b,n(j)+1);

    % Function values
    y=exp(x).*sin(x);

    % Separation between points
    h=(b-a)/n(j);

    % Approximate first order derivative
    yp=MyDerivs(y,h);

    % Exact derivative
    z=exp(x).*sin(x) +exp(x).*cos(x);

    % Relative error
    re=(z-yp)./z;
```

```
    % Maximum relative error
    mre(j)=max(abs(re));
end

% Plot maximum relative error as a function of n
plot(log10(n),log10(mre));

% Grids
grid on; grid minor;

% Labels
xlabel('log_{10}(n)'); ylabel('log_{10}(max(abs(relative
    error)))');

% Print the figure to a file
% print('MyDerivs','-depsc2');
```

8.4 Appendix 4 - MyPiecewiseHermite.m

Source code for `MyPiecewiseHermite.m`.

```
function z=MyPiecewiseHermite(s,y,yp,t)

% MyPiecewiseHermite Evaluate Hermite's piecewise
    approximation

% INPUT:
%   s    a linear array of m points where f and f' are
    known
%   f    the function values, y = f(s)
%   fp   the derivatives, yp = f'(s)
%   t    a linear array of sample points where z=p(t) is
    sought
%
% OUTPUT:
%   z    the values of Hermite's piecewise approximation, z
     = p(t)
%
%
```

```
% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen (
    spock@cs.umu.se)
%   2018-11-25 Initial programming and testing
%
% Programming by Aladdin Persson (alhi0008@student.umu.se)
%   2018-12-08 Initial programming


% Determine the number of points
m=numel(t);

% Define the polynomial p0
p0 = @(t)(1+2.*t).*((1-t).^2);
% Define the polynomial p1
p1 = @(t)(t.^2).*(3-2*t);
% Define the polynomial q0
q0 = @(t) t.*((1-t).^2);
% Define the polynomial q1
q1 = @(t)t.^2 .* (t - 1);

% Determine the number of sample points where we know f and
    f'
n=numel(s);

% Loop over all points of t
for i=1:m
    % Isolate the ith value of t into a variable tau
    tau = t(i);
    % Find the interval s(j), s(j+1) which contains tau
    j=find(s(1:n-1)<=tau,1,'last');
    % Isolate the endpoints of the interval which contains
        tau into a, b
    a=s(j); b=s(j+1);
    % Map tau into a point x in [0,1] using the linear
        transformation
    % which maps a into 0 and b into 1
    x=(tau-a)./(b-a);
    % Compute Hermite's approximation of f(tau)
        corresponding to the
    % sub-interval [a,b]
```

```
    z(i)=y(j).*p0(x)+y(j+1).*p1(x)+yp(j).*(b-a).*q0(x)+yp(j+1).*(b-a).*q1(x);
end
```

8.5 Appendix 5 - MyPiecewiseHermiteMWE1.m

Source code for `MyPiecewiseHermiteMWE1.m`.

```
% Minimal working example for function MyPiecewiseHermiteMWE1

% Programming by Aladdin Persson (alhi0008@student.umu.se)
%   2018-12-08 Initial programming

clear all; close all; clc;

% Interval
a=0; b=1;

% Maximum number of iterations
maxit=20;

% Allocate space
n=zeros(maxit,1); mre=zeros(maxit,1);

% Points for comparison
t=linspace(a,b,100*maxit+1);

% Target function
f=@(x)exp(x).*sin(x);

% Derivative of target function
fp=@(x)exp(x).*sin(x)+exp(x).*cos(x);

% Loop over the number of sample points
for j=1:maxit

    % Number of sample points
    n(j)=10*j;

    % Sample points
```

```
    s=linspace(a,b,n(j)+1);

    % Separation between points
    h=(b-a)/n(j);

    % Evaluate y=f(s)
    y=f(s);

    % Evaluate yp=f'(s) exactly
    yp=fp(s);

    % Evaluate yp=f'(s) using an approximation
    % This has surprising consequences
    %yp=MyDerivs(y,h);

    % Define Hermite approximation
    z=@(t)MyPiecewiseHermite(s,y,yp,t);

    % Define relative error function
    R=@(t)(f(t)-z(t))./f(t);

    % Maximum relative error
    mre(j)=max(abs(R(t)));
end

% Plot a suitable transformation of the data
plot(log10(n),log10(mre));

% Labels
xlabel('log_{10}(n)'); ylabel('log_{10}(max(abs(relative
    error)))');

% Turn on the grid
grid on; grid minor;

% Print the figure to a file
% print('MyPiecewiseHermite','-depsc2');
```

subsectionAppendix 6 - MyEvent.m Source code for `MyEvent.m`.

```
% MWE for range_rkx with the usage of Hermite's
    approximation
```

```
% Programming by Aladdin Persson (alhi0008@student.umu.se)
%  2018-12-09 Initial programming

clear all; close all; clc;

load shells.mat

% Specify shell and enviroment
param=struct('mass',10,'cali',0.088,'drag',@(x)mcg7(x),'atmo',@(x)atmosisa(x),'grav',@(x)9.82,'wind',@(t,x)[0,0]);

% Set the muzzle velocity and the elevation of the gun
v0=780; theta=60*pi/180;

% Select the method which will be used to integrate the trajectory
method='rk2';

% Select the basic time step size and the maximum number of time steps
dt=0.1; maxstep=2000;

% Compute the range of the shell
[r, flag, t, tra]=range_rkx(param,v0,theta,method,dt,maxstep);
flag;

% Below follows a long sequence of commands which demonstrates how to get
% a very nice plot of the trajectory automatically

% Obtain the coordinates of the corners of the screen
screen=get(groot,'Screensize');

% Isolate the width and height of the screen measured in pixels
sw=screen(3); sh=screen(4);

```

```
% Obtain a handle to a new figure
hFig=gcf;

% Set the position of the desired window
set(hFig,'Position',[0 sh/4 sw/2 sh/2]);

% Plot the trajectory of the shell.
plot(tra(1,:),tra(2,:));

hold on;

x=linspace(0.0e4,1.6e4,1e6);
plot(x,simple_landscape(x))

% We want to solve y(t) - h(x(t)) = 0, i.e shell hit ground
    .
% We can approx. y(t), x(t), with Hermite's piece-wise
    approximation.

y=tra(2,:);yp=tra(4,:);
x=tra(1,:);xp=tra(3,:);
s=t;

% Create function that generates any value t given points y
    ,yp,x,xp and
% points s where y, y' is known.
y_approx=@(t) MyPiecewiseHermite(s,y,yp,t);
x_approx=@(t) MyPiecewiseHermite(s,x,xp,t);

% Create event function g(t) which is height position of
    shell - landscape
% height. When this is zero it means the shell has hit the
    ground.
g=@(t) y_approx(t)- simple_landscape(x_approx(t));

% Tolerances for bisection
delta=1e-12;eps=1e-12;maxit=100;

% By plotting g(t), we see that it switches sign ~between t
    =70, and t=80.
```

```
t0=70;t1=80;

% Find better approximation of time c, where shell collides
    with ground
[c, flag, it, a, b, his, res] = bisection(g,t0,t1,g(t0),g(
    t1),delta,eps,maxit,false);

% Plot the point where they collide
plot(x_approx(c), y_approx(c), 'k*')

% Add legend to make it easier to distinguish between
    landscape and traj.
legend('Trajectory of shell', 'Landscape', 'Striking point'
    )

% Turn of the major grid lines and set the axis
grid ON; axis([0 16000 -2000 11000]); grid MINOR;

% Save plot as 'shelltrajectory.eps'
print('shelltrajectory','-depsc2');
```

8.6 Appendix 8 - derivatives_hermites.m

Source code for `derivatives_hermites.m`.

```
% Purpose of this script is to evaluate polynomials p0,p1,
    q0,q1 and their
% derivatives at specific values of t which are used in
    Hermite's
% piece-wise approximation

% PROGRAMMED by Aladdin Persson (alhi0008@student.umu.se)
%   Initial programming 2018-12-08

clear all; close all; clc;
syms t

% Initialize polynomials p0(t),p1(t),q0(t),q1(t)
p0 = @(t)(1+2*t)*((1-t).^2);
p1 = @(t)(t.^2)*(3-2*t);
q0 = @(t) t.*((1-t).^2);
```

```
q1 = @(t)t.^2 .* (t - 1);

% Use MatLab built in differentiate function (diff) and
    evaluate derivative
% at val.
p0_prime = @(val) eval(subs(diff(p0,t),t,val));
p0_prime = @(val) eval(subs(diff(p0,t),t,val));

p1_prime = @(val) eval(subs(diff(p0,t),t,val));
p1_prime = @(val) eval(subs(diff(p0,t),t,val));

q0_prime = @(val) eval(subs(diff(p0,t),t,val));
q0_prime = @(val) eval(subs(diff(p0,t),t,val));

q1_prime = @(val) eval(subs(diff(p0,t),t,val));
q1_prime = @(val) eval(subs(diff(p0,t),t,val));

% Example p0(0), p0'(0), similarly to others.
p0(0)
p0_prime(0)
```

8.7 Appendix 9 - simple_landscape.m

Source code for `simple_landscape.m`.

```
function y=simple_landscape(x)

% simple_landscape - Computes the height of a simple
    landscape given x.

% CALL SEQUENCE: y=simple_landscape(x)
%
% INPUT:
%   length x
%
% OUTPUT:
%   height y
%
% EXAMPLE: Used in MyEvent.m

% Programming by Aladdin Persson (alhi0008@student.umu.se)
```

```
%  2018-12-09  Initial  programming

% Fill the array y with zeros
y=zeros(size(x));

% Isolate all the indices of x values between 13000 and
    15000
idx=14000<=x & x<=16000;

% Define a half-circle with center at 14000 and radius 1000
% Admittedly, this is an odd hill ...
y(idx)=sqrt(1e6-(x(idx)-15000).^2);
```

8.8 Appendix 10 - range_rkx.m

Source code for `range_rkx.m`.

```
function [r, flag, t, tra]=range_rkx(param,v0,theta,method,
    dt,maxstep)

% RANGE_RKX Computes the range of a shell using a Runge-
    Kutta method
%
% All time steps have the same size, except the last which
    is adjusted to
% put the shell exactly on the ground.
%
% CALL SEQUENCE: [r, flag, t, tra]=range_rkx(param,v0,theta
    ,method,dt,maxstep)
%
% INPUT:
%   param     a structure describing of the environment and
    the shell
%                param.mass    the mass of the shell
%                param.cali    the caliber of the shell
%                param.drag    a function computing the drag
      coeffient
%                param.atmo    a function computing the
    atmosphere
%                param.grav    a function computing gravity
%                param.wind    a function computing the wind
```

```
%   v0        the muzzle velocity of the shell
%   theta     the elevation of the gun in radians
%   method    a string describing the method, see "help RK" for options
%   dt        the standard time step, the last step will be shorter
%   maxstep   the maximum number of time steps allowed, a safety valve.
%
% OUTPUT:
%   r         the computed range if flag=1;
%   t         the time instances where the trajectory was approximated
%   tra       the computed trajectory, y(:,i) corresponds to time t(i)
%                 tra(1,i) is the x-component of the shells position
%                 tra(2,i) is the y-component of the shells position
%                 tra(3,i) is the x-component of the shells velocity
%                 tra(4,i) is the y-component of the shells velocity
%    flag     flag=0 if the shell did not hit the ground
%             flag=1 if the shell hit the ground
%
% MINIMAL WORKING EXAMPLE: range_rkx_mwe1
%
% See also: COMPUTE_RANGE, COMPUTE_ELEVATION, FIRE, RANGE_RK1, TARGET

% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen (spock@cs.umu.se)
%  Fall 2014    Initial programming and testing
%  2015-09-22   Globals m, k, and g integrated into structure CONST
%  2015-10-31   Replaced structure CONST with mandatory PARAM
%  2015-10-31   Minor error in the inline comments fixed during yearly review
```

```
% 2015-11-01 Extended the description of the initial condition
% 2015-12-08 Added support for wind to shell4.m
% 2016-06-22 Adapted routine to improved bisection method
% 2016-06-23 Added logical check for bad elevations
% 2016-09-09 Adapted to more flexible SHELL4A

% TODO: Prior to HT-2016
%   a) Add support for wind to shell8.m and target.m
%   b) Add shell data to the simulation using interpolation from tables
%   c) Investigate if a military grade 6 DOF model is feasible

% ///////////////////////////////////////////////////////////////////////
% Select the relative tolerance being use by the nonlinear solver
% ///////////////////////////////////////////////////////////////////////

% The default value of tol is the double precision unit round off.
%   ... change this value at your own peril
%   ... or if you are told to do so
%   ... or if you are feeling adventurous
tol=2^-53;

% Select a shell model, feeding it the parameters of the simulation
shell=@(t,x)shell4a(param,t,x);

% Define the initial condition; must be compatible with the simple shell model
% There are four coordinates:
% 1st coor. is the x coordinate of the muzzle of the gun
% 2nd coor. is the y coordinate of the muzzle of the gun
```

```
%  3rd coor. is the x coordinate of the velocity of the
   shell when it exits
%  4th coor. is the y coordinate of the velocity of the
   shell when it exits
tra0=[0; 0; v0*cos(theta); v0*sin(theta)];

% Allocate space for trajectory
tra=zeros(4,maxstep+1);

% Initialize the trajectory
tra(:,1)=tra0;

% Allocate space to record the time instances
t=zeros(1,maxstep+1);

% Anticipate failure or bad input.
r=NaN; flag=0;

% Check for bad elevation
if (sin(theta)<=0)
    % The shell is fired into the ground
    r=0; flag=1; t=0; tra=tra(:,1);
    % Quick return
    return;
end

% Pickup the method to use.
switch lower(method)
    case 'rk1'
        phi=@phi1;
    case 'rk2'
        phi=@phi2;
    case'rk3'
        phi=@phi3;
    case 'rk4'
        phi=@phi4;
    otherwise
        fprintf('Invalid method specified! Aborting\n');
           return;
end
```

```

% Loop over the time steps
for it=1:maxstep
    % Advance the clock a single time step
    t(it+1)=it*dt;

    % Advance the solution a single step using the selected method.
    tra(:,it+1)=phi(shell,t(it),tra(:,it),dt);

    % Test to see if we are below ground level, tra(2,it+1)<0
    if (tra(2,it+1)<0)
        % ------------------------------------------------------------
        % We passed through the ground! Go back and compute the time step
        % which will put the shell exactly on the ground.
        % ------------------------------------------------------------

        % Isolate the last point above ground level.
        z0=tra(:,it); t0=t(it);

        % Define the function psi(x) which isolates the y coordinate
        % of the shell after a step of size x*dt.
        psi=@(x)[0 1 0 0]*phi(shell,t0,z0,x*dt);

        % Find the 'exact' timestep which will put the shell on the ground
        rho=bisection(psi,0,1,tra(2,it),tra(2,it+1),tol*t(it+1),tol*tra(1,it),60);

        % Calculate the 'exact' point of impact;
        aux=phi(shell,t0,z0,rho*dt);

        % Save the time and point of impact
```

```
            t(it+1)=t0+rho*dt; tra(:,it+1)=aux;

            % The range has now been computed, signal succes
                ...
            flag=1;

            % ... and break from the for-loop.
            break;
        end
end

% Remove any trailing zeros from output arrays
tra=tra(:,1:it+1); t=t(1,1:it+1);

% Only define the range if the shell hit the ground !!!
if flag==1
  % Isolate the range
    r=tra(1,it+1);
end
```