UMEÅ UNIVERSITY
Department of Computing Science
Assignment report, v.2

December 14, 2018

5DV005 - Project 2

Scientific Computing
Autumn 2018, 7.5 Credits

Approximation of real functions and solution of non-trivial equations

Name Betty Törnkvist (`et16btt@cs.umu.se`)
Jonas Sjödin (`id16jsn@cs.umu.se`)
Emil Söderlind (`id15esd@cs.umu.se`)

Path ~`et16btt/edu/tvb/5dv005ht18/`

Teacher
Carl Christian Kjelgaard Mikkelsen

Contents

1 Introduction **1**

2 The zero theorems **1**
- 2.1 Theorems 1
- 2.2 Procedure 1

3 Approximation of derivatives **2**
- 3.1 Taylor polynomials 3
- 3.2 Symmetric difference quotient when $h \to 0$ 3
- 3.3 Left special case when $h \to 0$ 3
- 3.4 Right special case when $h \to 0$ 4
- 3.5 Analysing the error on a minimal working example 4

4 Hermites piece-wise approximation **5**
- 4.1 Preparing the values from polynomials 5
  - 4.1.1 Deriving function values of $p_0(t)$ 5
  - 4.1.2 Deriving function values of $p_1(t)$ 6
  - 4.1.3 Deriving function values of $q_0(t)$ 6
  - 4.1.4 Deriving function values of $q_1(t)$ 6
- 4.2 Derivation of Hermite's approximation polynomial $p(x)$ 7
  - 4.2.1 Deriving $p(a) = f(a)$ 7
  - 4.2.2 Deriving $p(b) = f(b)$ 7
  - 4.2.3 Derivative of $p(x)$ 8
  - 4.2.4 Deriving $p'(a) = f'(a)$ 8
  - 4.2.5 Deriving $p'(a) = f'(a)$ 8
- 4.3 Hermite's piece-wise approximation method 8

5 Impact of a shell **9**
- 5.1 Procedure 9

6 Discussion **10**

References **11**

A MyZeroTheorem.m **12**

B MyDerivs.m **14**

C a2f3.m **15**

D MyPiecewiseHermite.m **16**

E MyEvent.m **17**

1 Introduction

In this project we set out to approximate a function f, $f : I \to \mathbb{R}$, with the interval is $I = [a, b] \subset \mathbb{R}$. We want to solve the equation where $f(x) = 0$ and construct reliable approximations of $f(t)$ for $t \in [x_o, x_n]$.

We start by creating a script that uses Runge's theorem and the mean value theorem to illustrate the zero theorems that assert the existence of a zero. We then create a script that approximates derivatives by using Taylor's theorem and expanding the function into a series, and show that the error committed by the script is of size $O(h^2)$. We move on to creating a script that approximates the function values with the help of Hermite's piece-wise approximation method and show that the error decays as $O(h^4)$. Lastly, we use our previous scripts, the Hermite approximation method and the derivatives approximation method, to solve a non-linear equation and therefore also approximate a so-called event equation. In this case, the event location of the calculated example is the point where a trajectory meets its target. In this case, error estimation is not a trivial problem and will not be presented.

2 The zero theorems

To establish a deeper understanding of central theorems regarding zero in the numerical approximations, a objective where all theorems are applied, was conducted.

2.1 Theorems

The theorems are the following:

The intermediate value theorem says that given a function $f : I \to R$ with a bracket (a_0, b_0) where $f(a_0) * f(b_0) < 0$ there must exist at least one zero $r \in (a_0, b_0)$.

Rolle's theorem says that if $a_0 \in I$ and $b_0 \in I$ are zeroes of a differentiable function $f : I \to R$, then by Rolle's theorem there exists a c between a_0 and b_0 such that $f'(c) = 0$.

Mean value theorem says that if $a_0 \in I$ and $b_0 \in I$ and $f : I \to R$, is differentiable, then by the mean value theorem there exist a c between a_0 and b_0 such that

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

2.2 Procedure

Given the function

$$f(x) = e^x * sin(x)$$

and its derivative

$$f'(x) = e^x * (sin(x) + cos(x))$$

The zero c where $f'(c) = 0$ was approximated using the bisection algorithm. At the point c a tangent t_1 was defined as a constant function. $f(x)$ and t_1 can be seen in figure 1 in blue receptively red.

A tangent t_2 was defined, touching the points $f(2)$ and $f(\pi)$. t_2 can be seen in figure 1 in yellow.

A tangent t_3 was defined to be parallel to t_2 and be zero in the point q where $f(q) = f'(q)$, approximated using bisection. t_3 can be seen in figure 1 in purple.

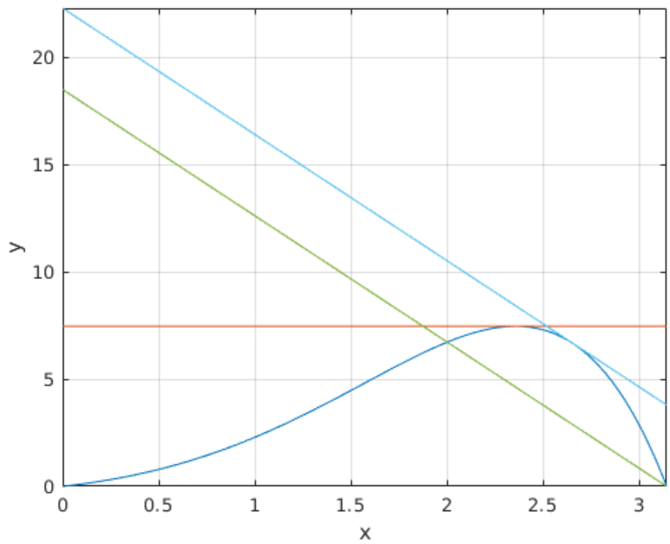

Figure 1: Zero theorem procedure

3 Approximation of derivatives

To perform numerical derivation of a derivable function f at a point x we have used a symmetric difference quotient

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} \tag{1}$$

where h is small. Section 3.2 derives how the expression behaves when $h \to 0$.There is special cases in the end points, where the neighbour value to the right respectively left is missing. In the left most end point we are using

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

since we lack points to the left. Section 3.3 derives how the expression behaves when $h \to 0$. In the right most end point we are using

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

since we lack points to the right. Section 3.4 derives how the expression behaves when $h \to 0$. The matlab code is provided in appendix B.

3.1 Taylor polynomials

Taylor's theorem tells us that any function can be expressed as a Taylor series [1]. To use the approximation expressions (1), (2) and (3) we will use the following expressions derived from Taylor's theorem:

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3) \tag{4}$$

$$f(x+2h) = f(x) + 2f'(x)h + 2f''(x)h^2 + O(h^3) \tag{5}$$

$$f(x-h) = f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3) \tag{6}$$

$$f(x-2h) = f(x) - 2f'(x)h + 2f''(x)h^2 + O(h^3) \tag{7}$$

3.2 Symmetric difference quotient when $h \to 0$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2), \quad h \to 0, \quad h > 0. \tag{8}$$

Since

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

$$\frac{f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3) - (f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3))}{2h} + O(h^2) =$$

$$\frac{f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3) - f(x) + f'(x)h - \frac{1}{2}f''(x)h^2 - O(h^3))}{2h} + O(h^2) =$$

$$\frac{2f'(x)h + O(h^3)}{2h} + O(h^2) = f'(x) + O(h^2) \to f'(x), \quad h \to 0$$

We can conclude that, as h approaches 0 the error $O(h^2)$ is insignificant.

3.3 Left special case when $h \to 0$

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} + O(h^2), \quad h \to 0, \quad h > 0. \tag{9}$$

Since

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

$$\frac{-3f(x) + 4(f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3)) - (f(x) + 2f'(x)h + 2f''(x)h^2 + O(h^3))}{2h} + O(h^2) =$$

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

$$\frac{2f'(x)h + O(h^3)}{2h} + O(h^2) = f'(x) + O(h^2) \to f'(x), \quad h \to 0$$

We can conclude that, as h approaches 0 the error $O(h^2)$ is insignificant.

3.4 Right special case when $h \to 0$

$$f'(x) = \frac{f(x-2h) - 4f(x-h) + 3f(x)}{2h} + O(h^2), \quad h \to 0, \quad h > 0. \tag{10}$$

Since

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

$$\frac{f(x) - 2f'(x)h + 2f''(x)h^2 + O(h^3) - 4(f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3)) + 3f(x)}{2h} + O(h^2) =$$

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

$$\frac{2f'(x)h + O(h^3)}{2h} + O(h^2) = f'(x) + O(h^2) \to f'(x), \quad h \to 0$$

We can conclude that, as h approaches 0 the error $O(h^2)$ is insignificant.

3.5 Analysing the error on a minimal working example

With a given minimal working example (appendix C) we could measure the error committed by the numerical derivation. A plot of the error can be seen in figure 2.

In equation (11) we can see that the logarithmic slope is -2 for each log(n) such that it is decreasing by 2 for each integer step in the x-direction.

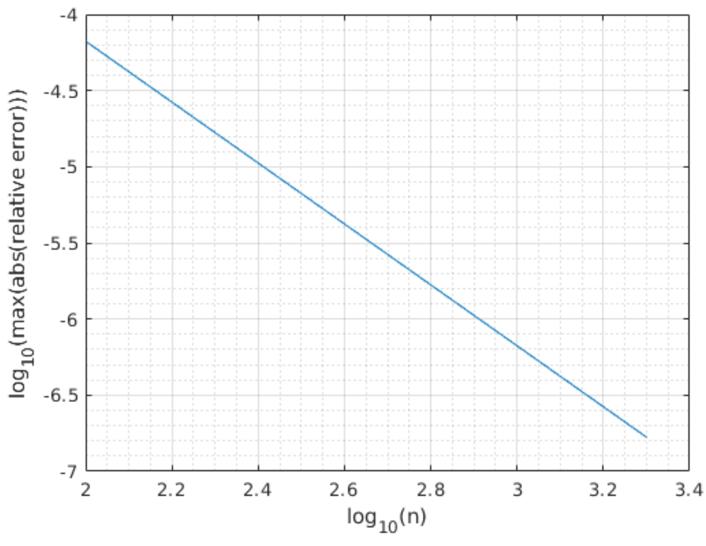

Figure 2: By decreasing the stepsize h the error of the first derivative decay rapidly.

$$E_h = O(h^2), \quad n * h = K \implies$$

$$\exists C > 0, \quad \exists h_0 > 0 : h \in (0, h_0], \quad |E_h| \leq h^2 \implies$$
$$log(|E_h|) \leq log(C) + 2 * log(h) = log(C) + 2(log(K) - log(n)) =$$
$$= log(C) + 2 * log(K) - 2 * log(n) \tag{11}$$

4 Hermites piece-wise approximation

Hermite's piece-wise approximation takes a set of given points and with their function values and derivative values at those points. It then approximates the function values in another given set of points and calculates them. It does that by using the following equations.

$$p_0(t) = (1 + 2t)(1 - t)^2, \quad p_1(t) = t^2(3 - 2t) \tag{12}$$

$$q_0(t) = t(1 - t)^2 \quad q_1(t) = t^2(t - 1) \tag{13}$$

From this we know that their derivatives are:

$$p_0'(t) = -6t(1 - t), \quad p_1'(t) = 6t(1 - t) \tag{14}$$

$$q_0'(t) = 3t^2 - 4t + 1, \quad q_1'(t) = 3t^2 - 2t \tag{15}$$

4.1 Preparing the values from polynomials

To use the polynomials (12), (13) and their derivatives (14), (15) in Hermite's piece-wise approximation function values of $t = 0$ and $t = 1$ was derived.

4.1.1 Deriving function values of $p_0(t)$

We want to show that

$$p_0(0) = 1, \quad p_0(1) = 0, \quad p_0'(0) = 0, \quad p_0'(1) = 0$$

Given equation 12 and 14 we can show

$$p_0(0) = (1 + 2 * 0)(1 - 0)^2 = 1$$
$$p_0(1) = (1 + 2 * 1)(1 - 1)^2 = 0$$
$$p_0'(0) = -6 * 0 * (1 - 0) = 0$$
$$p_0'(1) = -6 * 1(1 - 1) = 0$$

4.1.2 Deriving function values of $p_1(t)$

We want to show that

$$p_1(0) = 0, \quad p_1(1) = 1, \quad p_1'(0) = 0, \quad p_1'(1) = 0$$

Given equation 12 and 14 we can show

$$\begin{aligned} p1(0) &= 0^2(3 - 2 * 0) = 0 \\ p1(1) &= 1^2(3 - 2 * 1) = 1 \\ p'1(0) &= 6 * 0(1 - 0) = 0 \\ p'1(1) &= 6 * 1(1 - 1) = 0 \end{aligned}$$

4.1.3 Deriving function values of $q_0(t)$

We want to show that

$$q_0(0) = 0, \quad q_0(1) = 0, \quad q_0'(0) = 1, \quad q_0'(1) = 0$$

Given equation 13 and 15 we can show

$$\begin{aligned} q_0(0) &= 0 * (1 - 0)^2 = 0 \\ q_0(1) &= 0 * (1 - 1)^2 = 0 \\ q_0'(0) &= 3 * 0^2 - 4 * 0 + 1 = 1 \\ q_0'(1) &= 3 * 1^2 - 4 * 1 + 1 = 0 \end{aligned}$$

4.1.4 Deriving function values of $q_1(t)$

We want to show that

$$q_1(0) = 0, \quad q_1(1) = 0, \quad q_1'(0) = 1, \quad q_1'(1) = 1$$

Given equation 13 and 15 we can show

$$\begin{aligned} q_1(0) &= 0^3 - 0^2 = 0 \\ q_1(1) &= 1^3 - 1^2 = 0 \\ q_1'(0) &= 3 * 0^2 - 2 * 0 = 0 \\ q_1'(1) &= 3 * 1^2 - 2 * 1 = 1 \end{aligned}$$

4.2 Derivation of Hermite's approximation polynomial $p(x)$

We know that,

$$\phi(x) = \frac{x-a}{b-a} \tag{16}$$

where $a = 0$ and $b = 1$.
From this we can conclude the following:

$$\phi(a) = \frac{a-a}{b-a} = \frac{0-0}{1-0} = 0 \tag{17}$$

$$\phi(b) = \frac{b-a}{b-a} = \frac{1-0}{1-0} = 1 \tag{18}$$

and that the derivative of $\phi(x)$is:

$$\phi'(x) = \frac{1}{1-0} = 1 \tag{19}$$

We know that

$$p(x) = f(a)p_0(\phi(x)) + f(b)p_1(\phi(x)) + f'(a)(b-a)q_0(\phi(x)) + f'(b)(b-a)q_1(\phi(x)) \tag{20}$$

We want to show that

$$p(a) = f(a), \quad p(b) = f(b), \quad p'(a) = f'(a), \quad p'(b) = f'(b) \tag{21}$$

4.2.1 Deriving $p(a) = f(a)$

Given equation 17 and 18 we can show that $p(a) = f(a)$:

$$p(a) = f(a)p_0(\phi(a)) + f(b)p_1(\phi(a)) + f'(a)(b-a)q_0(\phi(a)) + f'(b)(b-a)q_1(\phi(a)) \implies$$

$$f(a)p_0\Big(\frac{a-a}{b-a}\Big) + f(b)p_1\Big(\frac{a-a}{b-a}\Big) + f'(a)(b-a)q_0\Big(\frac{a-a}{b-a}\Big) + f'(b)(b-a)q_1\Big(\frac{a-a}{b-a}\Big) \implies$$

$$p(a) = f(a)p_0(0) + f(b)p_1(0) + f'(a)(b-a)q_0(0) + f'(b)(b-a)q_1(0) \implies$$

$$p(a) = f(a) * 1 + f(b) * 0 + f'(a)(b-a) * 0 + f'(b)(b-a) * 0 \implies$$

$$p(a) = f(a)$$

4.2.2 Deriving $p(b) = f(b)$

Given equation 17 and 18 we can show that $p(b) = f(b)$:

$$p(b) = f(a)p_0(\phi(b)) + f(b)p_1(\phi(b)) + f'(a)(b-a)q_0(\phi(b)) + f'(b)(b-a)q_1(\phi(b)) \implies$$

$$f(a)p_0\Big(\frac{b-a}{b-a}\Big) + f(b)p_1\Big(\frac{b-a}{b-a}\Big) + f'(a)(b-a)q_0\Big(\frac{b-a}{b-a}\Big) + f'(b)(b-a)q_1\Big(\frac{b-a}{b-a}\Big) \implies$$

$$p(b) = f(a)p_0(1) + f(b)p_1(1) + f'(a)(b-a)q_0(1) + f'(b)(b-a)q_1(1) \implies$$

$$p(b) = f(a) * 0 + f(b) * 1 + f'(a)(b-a) * 0 + f'(b)(b-a) * 0 \implies$$

$$p(b) = f(b)$$

4.2.3 Derivative of $p(x)$

By the chain rule [2] we know that the derivative of $p(\phi(x))$ is:

$$p'(\phi(x)) = p'(\phi(x)) * \phi'(x) = p'(\phi(x)) * 1 \tag{22}$$

We can show that:

$$\begin{gathered} p'(x) = f(a) * p_0'(\phi(x)) + f(b)p_1'(\phi(x)) + \\ f'(a)(b-a)q_0'(\phi(x)) + f'(b)(b-a)q_1'(\phi(x)) \end{gathered} \tag{23}$$

When $a = 0$ and $b = 1$ the derivative is:

$$p'(x) = f(a) * p_0'(\phi(x)) + f(b)p_1'(\phi(x)) + f'(a)q_0'(\phi(x)) + f'(b)q_1'(\phi(x)) \tag{24}$$

We know that $\phi(a) = 0$ and can therefore show that

$$p'(a) = f(a) * p_0'(0) + f(b)p_1'(0) + f'(a)q_0'(0) + f'(b)q_1'(0) =$$

4.2.4 Deriving $p'(a) = f'(a)$

$$\begin{aligned} p'(a) &= f(a) * p_0'(0) + f(b)p_1'(0) + f'(a)q_0'(0) + f'(b)q_1'(0) = \\ &= f(a) * 0 + f(b) * 0 + f'(a) * 1 + f'(b) * 0 = f'(a) \end{aligned} \tag{25}$$

4.2.5 Deriving $p'(a) = f'(a)$

$$\begin{aligned} p'(b) &= f(a) * p_0'(1) + f(b)p_1'(1) + f'(a)q_0'(1) + f'(b)q_1'(1) = \\ &= f(a) * 0 + f(b) * 0 + f'(a) * 0 + f'(b) * 1 = f'(b) \end{aligned} \tag{26}$$

4.3 Hermite's piece-wise approximation method

A script was created to approximate a function f values, see Appendix D. The script uses the polynomials p_0,p_1, q_0 and q_1 given in equation (12) and equation (13). It then iterates over the given sample points t, finds its interval, uses a linear transformation and finally computes the approximation corresponding to the sub-interval. Finally, the error is calculated, and the result can be seen in Figure 3.

In equation (27) we can see that the logarithmic slope is -4 for each log(n) such that it is decreasing by 4 for each integer step in the x-direction.

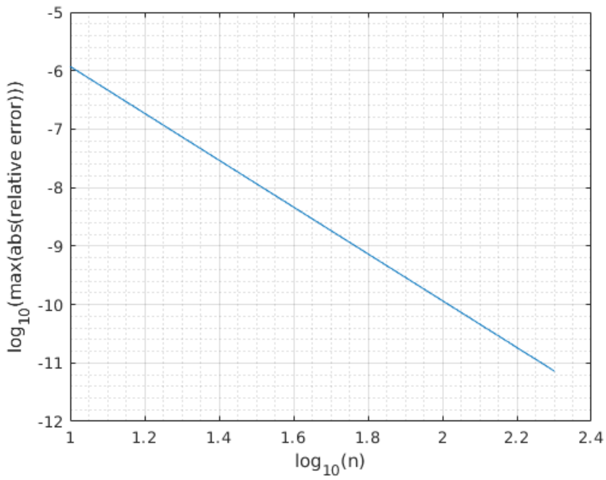

Figure 3: The error of Hermite approximation.

$$
\begin{gathered}
E_h = O(h^4), \quad n * h = K \implies \\
\exists C > 0, \quad \exists h_0 > 0 : h \in (0, h_0], \quad |E_h| \leq h^4 \implies \\
log(|E_h|) \leq log(C) + 4 * log(h) = log(C) + 4(log(K) - log(n)) = \\
= log(C) + 4 * log(K) - 4 * log(n) \qquad (27)
\end{gathered}
$$

5 Impact of a shell

To sum the projects concepts up, a more concrete problem scenario was solved. The scenario was to approximate the point p when a shell launched in a trajectory hits the ground of a fictional hill, see figure 4. To solve the problem the Hermite piece-wise method was used, see section 4.3.

Note: The computed time of impact on the hill is only an approximation and not the true time of impact.

5.1 Procedure

To find the point p of impact we used the given "range_rkx"-script which gives us a trajectory $tra(x)$ given external parameters. To create a hill the given "a2f6"-script was used giving us $hill(x)$, see figure 4 and Appendix E.

Since the $tra(x)$ contained less function values than $hill(x)$ we had to approximate the function values of $tra(x)$ on all function values of $hill(x)$. To do this we used the recently implemented "Piece-wis Hermites"-procedure. This gave us the trajectory with more points approximated $tra_*(x)$

The procedure required the derivative of the trajectory $tra'(x)$. To get this we used our own implementation of numerical derivation (see appendix B). Note that this requires the function $tra(x)$ to be continuous in the set interval

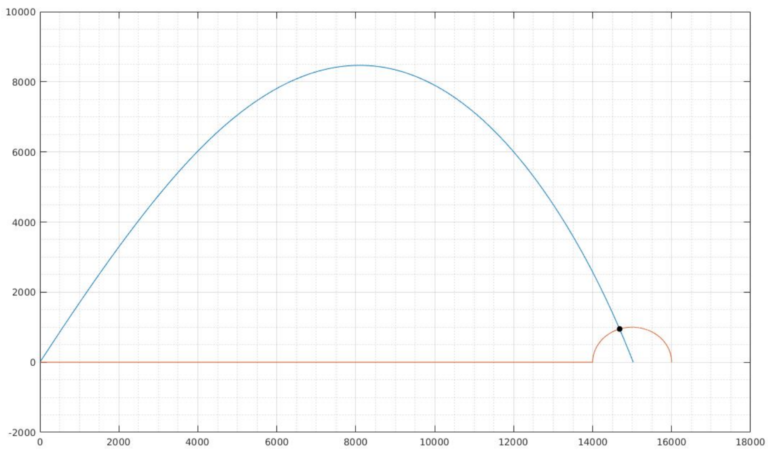

Figure 4: Shell impact on hill.

and differentiable in that interval, since Hermite's approximation requires a differentiable function.

When $tra_*(x)$ and $hill(x)$ share the same interval we could calculate the distance between the trajectory and the hill with

$$diff(x) = tra_*(x) - hill(x)$$

To find the impact point p we searched for the smallest point in $diff(x)$ with

$$min\{|diff(x)|\}$$

This gave us that the shell will impact the hill at $x = 14683$, see figure 4.

6 Discussion

During the project we have strengthen our knowledge about zero theorems, numerical derivation and Hermites piece-wise approximation. We have used this knowledge to solve a concrete projectile scenario of a shell and it's point of impact. This time we really figured out how important it is to draw a sketch over the problem, to easier solve hard problems.

References

[1] B. Taylor, *Methodus incrementorum directa & inversa.* impensis Gulielmi Innys, 1717.

[2] E. Weisstein. Chain rule. [Online]. Available: http://mathworld.wolfram.com/ChainRule.html

A MyZeroTheorem.m

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

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

% Interval
a=0; b=pi;

% Number of subintervals
n=100;

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

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

% Hold the graph
hold on;

% Turn on grid
grid on;

% Axis tight
axis tight

% ///////////////////////////////////////////////////////////////////////////
%  Illustration of Runge's theorem
% ///////////////////////////////////////////////////////////////////////////

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

% The function values corresponding to the initial search bracket
fp0=fp(x0);
fp1=fp(x1);

% Tolerances and maxit for bisection.
tol=10^-13; maxit = 1000;

% Run the bisection algorithm to find the zero c of fp
c=bisection(fp, x0, x1, fp0, fp1, 10^-10, 10^-10, maxit, 1);

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

% Plot the tangent
hold on
plot(s, w(s));


% ///////////////////////////////////////////////////////////////////////////
%  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)
k = (f1-f0)/(x1 -x0);
m = f1 -k * x1;

p=@(x)(k*x + m); % slope is here
```

```

% Plot the straight line between (x0,f0) with (x1,f1)
hold on
plot(s,p(s));

% Compute the slope of the corde
yp=k;

% 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 = bisection(g, x0, x1, g(x0), g(x1), tol, tol, maxit, 1);

% Define the line which is tangent to the graph of f at the point (c,f(c))
m = f(c) - yp*c;
q=@(x)(m +  k * x);

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

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

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

B 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: MyDerivsMWE

% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen (spock@cs.umu.se)
%   2018-11-26 Extracted from a working code
%   2018-12-11 Extracted and finished code by
%                Betty Tornkvist (et16btt@cs.umu.se)
%                Emil Soderlind (id15@esd@cs.umu.se)
%                Jonas Sjodin (id16jsn@cs.umu.se)

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

% The exercise is pointless unless there are at least 3 points
if m < 3
    fp = 0;
    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

for i=2:(m-1)
    fp(i) = (y(i + 1) - y(i - 1))/(2*h);
end

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

C a2f3.m

```
% A2F3  Minimal working example for A2F2

% 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=y+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');
```

D MyPiecewiseHermite.m

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

% A2F4 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
%   2018-12-11 Extracted and finished code by
%              Betty Tornkvist (et16btt@cs.umu.se)
%              Emil Soderlind (id15@esd@cs.umu.se)
%              Jonas Sjodin (id16jsn@cs.umu.se)

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

% Define the polynomial p0
p0=@(x)((1+2.*x).*((1-x).^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);
z = zeros(1, m);

% 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
```

E MyEvent.m

```
% MWE for range_rkx
% PROGRAMMING by
%   2018-12-11 Extracted and finished code by
%               Betty Tornkvist (et16btt@cs.umu.se)
%               Emil Soderlind (id15@esd@cs.umu.se)
%               Jonas Sjodin (id16jsn@cs.umu.se)

% Load shells models
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

% 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, ~, tra]=range_rkx(param,v0,theta,method,dt,maxstep);

% 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,:));

% Turn of the major grid lines and set the axis
grid ON; axis([0 18000 -2000 10000]); grid MINOR;

% ----------------------------------
% Our new implementation
% ----------------------------------

%Define the number of points
m = 16000;
xpoints = 1:m;

% Create a hill and add it to the plot
hill = a2f6(xpoints);
hold on;
plot(xpoints, hill);

% Define function input
spacing = 1 / 100;
s = tra(1, :);
y = tra(2, :);
yp = MyDerivs(y, tra(1, 2) - tra(1, 1));
t = 14000:tra(1, end);

% Approximate the shell on t points
tra_app = MyPiecewiseHermite(s, y, yp, t);

```

```
% Calculate the difference in heigh of the approximated trajectory
% and the hill
diff = tra_app - hill(14000:(14000 + length(tra_app) - 1));

% Find the point closest to the intersection point of the shell and
% the hill.
[~, ind] = min(abs(diff));

% Scatter plot the intersection point
hold on
scatter(14000 + ind, tra_app(ind),'filled', 'MarkerFaceColor',[0 0 0]);
```