# Project 1
# FMN011

Robert Foss (dt08rf1@student.lth.se)

# Contents

# 1 Introduction

Newton's law of cooling is not solvable algebraically, due to it being a nonlinear equation, however it possible to produce a solution analytically.

The aim of this project is to analytically calculate a solution for Newton's law of cooling.

## 1.1 Problem background

The issue lies in determining the time of death for professor Sommar and thereby confirming the alibi of instructor Tögersson.

# 2 Numerical Considerations

Implementations of the following methods were created; Bisection, Secant root, Fixed point iteration and Newton-Raphson's method. Matlab and Ocatave[5] was the only software directly used. Specific matlab functions used include; fzero (control), anonymous functions, fplot, plot, vectors, sym, feval, inline, diff, xlabel, ylabel, title and plot.

# 3 Results

## 3.1 Task 1

$$T' = -k(T - T_{office}) = -k(T - (22 + 0,5t)) \leftrightarrow T' + kT = -k(22 + 0,5t)$$

Integrating factor produces:
$$T = e^{-kt}k(\int 22e^{kt}dt + \int \frac{t}{2}e^{kt}dt) + ce^{-kt}$$
$$= e^{-kt}k([\frac{22}{k}e^{kt}] + ([\frac{e^{kt}}{2k}] - \int \frac{e^{kt}}{2k}dt)) + ce^{-kt}$$
$$= 22 + 0.5t - \frac{1}{2k} + ce^{-kt}$$

## 3.2 Task 2

$T(0) = 32$, given in the story of the problem.
$T(0) \leftrightarrow 22 - \frac{1}{2k} + c = 32 \leftrightarrow c = (10 - \frac{1}{2k})$
$T(1) = 29$, given in the story of the problem.
$T(1) \leftrightarrow 22, 5 - \frac{1}{2k}ce^{-k} \leftrightarrow c = (7 + \frac{1}{2k})e^k$
$\leftrightarrow 7 + \frac{1}{2k} = (10 - \frac{1}{2k})e^{-k}$

## 3.3 Task 3

The bisection method is an iterative method which repeatedly selects a subinterval where the root can found.

The choices of a and b were based on the criteria that $f(a)$ and $f(b)$ have opposite signs. A and b were chosen after a $f(k)$ was plotted for roots in the interval $[0, 10]$, as can be seen in Appendix B, section 9.1. The tolerance was chosen according to definition 1.3[1], $\frac{10^{-p}}{2}$. Where $p$ is the number of significant digits. Nmax was chosen any way that won't limit the precision of the function.

e was chosen to produce 6 significant digits. The function used can found in Appendix A, section 8.1.

```
format long;
f=@(k)(10+1/(2*k))*exp(-k)-7-1/(2*k);
a=0.1;
b=1;
e=0.5*10^(-2);
Nmax=99;
[answer, iterations, err_qoutient, err] = bisection(f,a,b,e,Nmax);
answer = 0.2898
```

## 3.4  Task 4

The secant root method is an iterative method which assumes a function is approximately linear in an interval and iteratively approximates the root of secant.

The choices of p0 and p1 are due to them being near the solution, as indicated by Task 3. The tolerance $s = e = 10^{-7}$ was chosen to only tolerate errors smaller than the precision desired. Nmax was chosen in a way that won't limit the precision of the function. The function used can found in Appendix A, section 8.2.

```
format long;
f=@(k)(10+1/(2*k))*exp(-k)-7-1/(2*k);
p0=0.1;
p1=1;
s=10^-7;
e=10^-7;
Nmax=99;
[answer, iterations, err_qoutient, err] = secantroot(f,a,b,s,e,Nmax);
answer = 0.296703053360269
```

## 3.5  Task 5

The fixed point method is a method for iteratively finding a $x = f(x)$.

$g(x) = x$ was calculated and found to be $g(k) = k = log(\frac{10+\frac{1}{2k}}{7+\frac{1}{2k}})$. x0 was chosen due to it being near the solution, as indicated by Task 4. The tolerance $e = 10^{-7}$ was chosen to only tolerate errors smaller than the precision desired. Nmax was chosen in a way that won't limit the precision of the function. The function used can found in Appendix A, section 8.3.

```
format long;
g=@(k)log((10+1/(2*k))/(7+1/(2*k)));
x0=1;
e=10^-7;
Nmax=99;
[answer, iterations, err_qoutient, err] = fixedpoint(g,x0,e,Nmax);
answer = 0.296703056932817
```

## 3.6   Task 6

Newton-Raphson's method is an iterative method that utilizes the first terms of the Taylor series for $f(x)$

x0 was chosen due to it being near the solution, as indicated by Task 4. The tolerance $e = 10^{-7}$ was chosen to only tolerate errors smaller than the precisionv desired. Nmax was chosen in a way that won't limit the precision of the function. The function used can found in Appendix A, section 8.4.

```
format long;
f=@(k)log((10+1/(2*k))/(7+1/(2*k)));
x0=1;
e=10^-7;
Nmax=99;
[answer, iterations, err_qoutient, err] = newton(f,x0,e,Nmax);
answer = 0.296703053214996
```

$k = 0.296703$ will be used for the approximation of $t_d(t) = 22 - 37\frac{t}{2} - \frac{1}{2k} + (10 + \frac{1}{2k})e^{-kt}$. x0 was chosen as the professor has been dead for atleast 0 hours. The tolerance, e, was chosen to provide a solution with minimal residual. Nmax was chosen in a way that won't limit the precision of the function.

```
format long;
f= @(t)22-37+t/2-1/(2*0.296703)+(10+1/(2*0.296703))*exp(-0.296703*t)
x0=0
e=10^-70;
Nmax=99;
[answer, iterations, err_qoutient, err] = newton(f,x0,e,Nmax);
answer = -1.332487352900656
```

Meaning that the professor died approximately 1.33 hours earlier than $T(0)$, about 6:40 p.m.

## 3.7   Task 7

9 significant digits were chosen. Leading to a tolerance of $10^{-10}$ for every method except for the bisection method, which's tolerance was set $\frac{10^{-9}}{2}$ in accordance with def. 1.3[1]. The points for the bisection method and the secant root method were chosen to be $-2$ and $0$, based on the root plot for $T(t)$ found in Appendix B, section 9.2. x0 for Newton-Raphson and the fixed point iteration method was chosen be $-1$, based the previously mentioned root plot for $T(t)$. Nmax was selected to a large number, $Nmax = 999$, to keep it from interfering. The function being approximated was once again $t_d(t) = 22 - 37\frac{t}{2} - \frac{1}{2k} + (10 + \frac{1}{2k})e^{-kt}$, with $k = 0.29670$ in accordance with Task 6. $g_d(t) = t = \frac{log(22 - 37 + \frac{t}{2} - \frac{1}{2k})}{log(10 + \frac{1}{2k})k}$ was used for the fixed point iteration method.

| Root finding method | Error | Iterations | Convergence speed, last iter. |
|---|---|---|---|
| Bisection method[8.1] | $1.1642 * 10^{-10}$ | 34 | 0.5 |
| Secant root method[8.2] | $8.2710 * 10^{-10}$ | 6 | 0.0002 |
| Fixed point iteration[8.3] | $1.1818 * 10^{-10}$ | 9 | 0.0458 |
| Newton-Raphson's method[8.4] | 0 | 4 | 0 |

# 4    Analysis

Error quotient plots can be found in Appendix: C, section 10. Bisection[10.1] and fixed point iteration[10.3] are shown to have a linear convergence rate and secant root[10.2] as well as Newton-Raphson's method[10.4] are shown to have a superlinear convergence rate, for the analyzed functions.

The points chosen for the respective methods clearly influence the number of iterations needed to reach a given number of significant digits.

# 5    Discussion

A weakness is using using specific points for the comparison of the different methods, which might produce suboptimal error quotient plots. Larger distances from the correct solution for points used in the root finding methods could be used to improve the accuracy of the error quotient plots.

The points used in Task 3–6 are also somewhat arbitrary, even though chosen from plots, but are only used to verify that the implemented functions are functional.

Calculating $g(x) = x$, for the fixed point method, proved somewhat time demanding since only a few $g(x) = x$ had $|g'(x)| < 1$, while $x \in [a, b]$[6].

# 6    Lessons learned

Improved Matlab skills, improved latex skills. Implementing and modifying the functions have improved my understanding of how these root finding methods work as well as my overall understanding for root finding methods. Hands-on understanding of convergence rates has helped me understand in which situation a certain root finding method might be appropriate.

Finding implementations of the root finding methods at the beginning stages of the project proved to be a good idea due to it widening my understanding of Matlab and the possibilities of Matlab. Understanding of the functions was aquired while modifying the functions and reasoning about how to best use them.

I've learned how to use the Matlab functions; feval, sym, inline and anonymous functions.

# 7    Acknowledgements

# References

[1] *Numerical Analysis*, Timothy Sauer. Pearson Education, 2005.

[2] *Newton-Raphson's method*, `http://www.bukisa.com/articles/78586_newton-method-and-bisection-method-matlab-scripts`
*Fetched 30/3 -11*

[3] *Bisection method*, Carmen Arévalo Matematikcentrum, Numerisk Analys, Lunds Universitet
`http://www.maths.lth.se/na/courses/FMN011/media/material/p1_2011____.pdf`
*Fetched 30/3 -11*

[4] *Secant root method*, `http://en.wikipedia.org/wiki/Secant_method`
*Fetched 30/3 -11*

[5] *Octave*, `http://www.gnu.org/software/octave/`
*Fetched 30/3 -11*

[6] *Fixed point itration*, `http://www.math.usm.edu/lambers/mat460/lecture9.pdf`
*Fetched 31/3 -11*

# 8 Appendix: A

## 8.1 Bisection

```
function [c, iter, e_quote, err] = bisection(f, a, b, tol, Nmax)
% Source: Slides from lecture #1
%
% Input variables:
% f      Function
% a,b    f(a), f(b) must have opposite signs
% tol    Tolerance
% Nmax   Maximum number of iterations
%
% Output variables:
% c      Solution
% iter   Number of iterations that were run
% e_quote Error quotient e_(n)/e_(n-1)
% err    Error

i = 1;
iter = 0;
cOld = abs(a-b);
e_quote(1) = 0;
err = 0;
while ((b-a)/2>tol) && (i<Nmax)
    iter = i;
    c = (a+b)/2; % midpoint
    err = abs(cOld-c);
    if (i > 1) % Create and save error quotient
        e_quote(i-1)=err/errOld;
    end

    if f(c)*f(a)>0
        a=c;
    elseif f(c)*f(b)>0
        b=c;
    else
        cOld = c;
    end
    i=i+1;
    errOld = err;
    cOld = c;
end
end
```

Author: Robert Foss - 8707174051

## 8.2 Secant root

```
function [Xs, iter, e_quote, err] = secantroot(f, p0, p1, atol, rtol, Nmax)
% Source: Wikipedia.
%
% secanttoot finds the root of Fun = 0 using the Secant method.
% Input variables:
% f       Function
% p0 p1   Two points in the neighborhood of the root (on either side, or the
%         same side of the root).
% atol    Absolute error tolerance.
% rtol    Residual error tolerance.
% Nmax    Maximum number of iterations
%
% Output variables:
% Xs      Solution
% iter    Number of iterations that were run
% e_quote Error quotient e_(n)/e_(n-1)
% err     Error

e_quote(1) = 0;
err = 0;
for i = 1:Nmax
    iter=i;
    fp1 = feval(f,p1);
    pi = p1 - fp1*(p0-p1)/(feval(f,p0)-fp1);
    err = abs(pi - p1);

    if (i > 1) % Create and save error quotient
        e_quote(i-1)=err/errOld;
    end

    if (err < atol) || (abs(feval(f,pi)) < rtol)
        Xs = pi;
        break
    end
    errOld=err;
    p0 = p1;
    p1 = pi;
end
end
```

## 8.3 Fixed point iteration

```
function [xc, iter, e_quote, err] = fixedpoint(g, x0, tol, Nmax)
% Source: Numerical Analysis by Timthy Sauer
%
% Input variables:
% g       Iteration function
% x0      Approximate solution
% tol     Tolerance
% Nmax    The maximum number of iterations
%
% Output variables:
% xc      Solution
% iter    Number of iterations that were run
% e_quote Error quotient e_(n)/e_(n-1)
% err     Error

x(1)=x0;
e_quote(1) = 0;
err = 0;
for i=1:Nmax
    k=i;
    x(i+1)=g(x(i));
    abs_error=abs(x(i)-x(i+1));
    rel_error=2*abs(x(i)-x(i+1))/(abs(x(i)+abs(x(i+1))));

    if (i > 1) % Create and save error quotient
        e_quote(i-1)=abs_error/abs_errorOld;
    end
    abs_errorOld = abs_error;
    if ((abs_error < tol) || (rel_error < tol))
        break;
    end
end
err = abs_error;
iter = k;
xc=x(k+1);
```
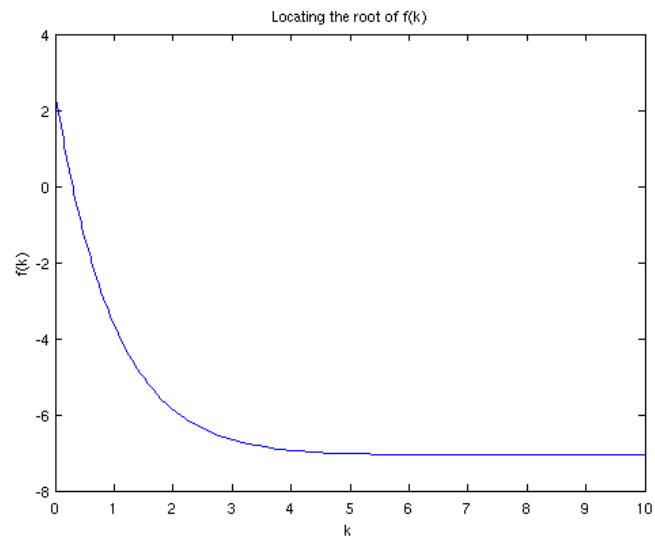
## 8.4  Newton-Raphson

```
function [c, iter, e_quote, err] = bisection(f, a, b, tol, Nmax)
% Source: Slides from lecture #1
%
% Input variables:
% f       Function
% a,b     f(a), f(b) must have opposite signs
% tol     Tolerance
% Nmax    Maximum number of iterations
%
% Output variables:
% c       Solution
% iter    Number of iterations that were run
% e_quote Error quotient e_(n)/e_(n-1)
% err     Error

i = 1;
iter = 0;
cOld = abs(a-b);
e_quote(1) = 0;
err = 0;
while ((b-a)/2>tol) && (i<Nmax)
    iter = i;
    c = (a+b)/2; % midpoint
    err = abs(cOld-c);
    if (i > 1) % Create and save error quotient
        e_quote(i-1)=err/errOld;
    end

    if f(c)*f(a)>0
        a=c;
    elseif f(c)*f(b)>0
        b=c;
    else
        cOld = c;
    end
    i=i+1;
    errOld = err;
    cOld = c;
end
end
```
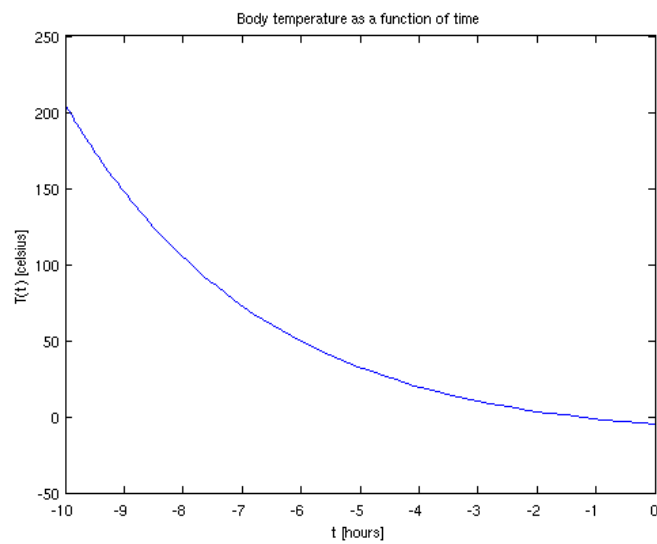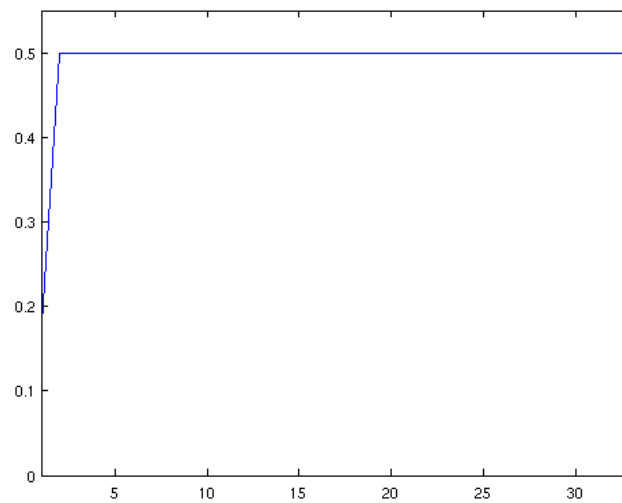
# 9 Appendix: B

## 9.1 F(k)



## 9.2 T(t)
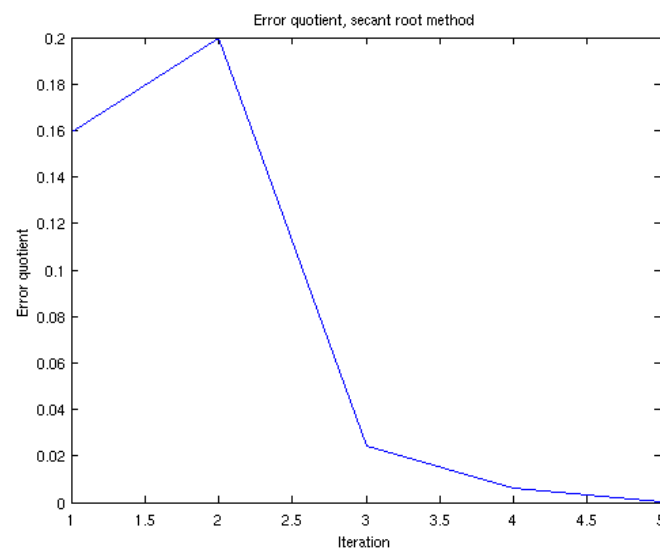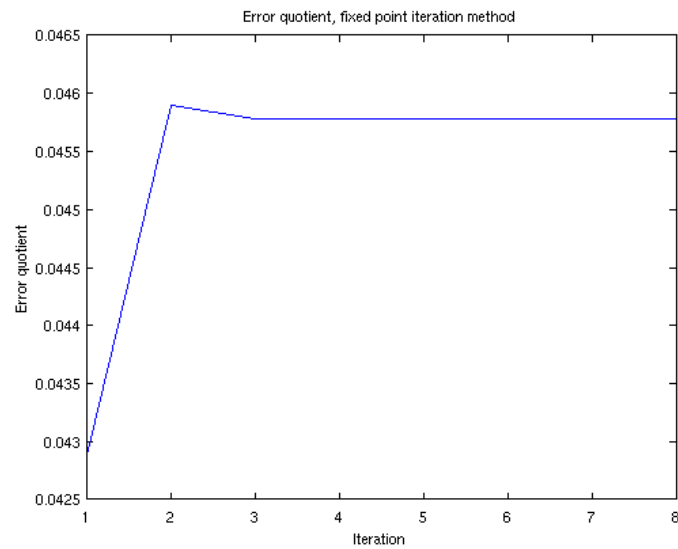
# 10 Appendix: C

## 10.1 Error quotient, bisection method



## 10.2 Error quotient, secant root method

## 10.3   Error quotient, fixed point iteration method



## 10.4   Error quotient, Newton-Raphson's method



Author: Robert Foss - 8707174051