Numerical Analysis Project 1
Non-Adiabatic Explosion

Zachary Vogel Matt Hansen
Instructor: Adam Norris

December 1, 2015

Introduction

The non-adiabatic explosion problem describes a mixture of fuel and oxidizer, at temperature T, in a container. A chemical reaction converts the fuel and oxidizer into products and results in a release of heat. This heat raises the temperature of the container and the materials inside it, which causes the reaction to proceed at a faster rate. The rate of transformation of fuel can be modeled by:

$$\frac{dA}{dt} = -\bar{c}Ae^{\frac{E}{RT}}$$

A is the amount of fuel, E is the activation energy for the reaction, R is the universal gas constant and c is a constant of proportionality. Initially RT is much smaller than E so the exponential term is small and the reaction proceeds slowly with a small temperature rise. As T increases, the exponential term will increase, and the reaction will proceed more rapidly. Then by conversation of energy we get:

$$c_v\frac{dT}{dt} = -\bar{c}\frac{dA}{dt} - h(T - T_0)$$

c_v is the heat capacity at constant volume, To is the surrounding temperature and h is a convective heat transfer coefficient associated with the heat lost to the surrounding environment from the exterior of the container. The left side of the equation represents the rate of change if internal energy, and the term?s on the right side represent heat release from the reaction and loss from the container.
We can non-dimensionalize the problem by defining the following variables, delta is proportional to $\frac{1}{h}$. $\epsilon = \frac{T_o R}{E}, \sigma = \frac{t}{\delta t_{\text{ref}}}$ with t_{ref} being a reference time. $\bar{T} = \frac{T}{To} + \epsilon\theta$. epsilon is much smaller than 1 because we have a high activation energy problem, and θ is a variable of order one. By substituting in these variables into the differential equation we obtain the following:

$$\frac{d\theta}{d\sigma} = \delta e^{\theta} - \theta$$

with $\theta(0) = 0$.
In this report we will explore two outcomes, one an explosion and one a fizzle and the magnitude of δ will determine which outcome will happen. $\delta < \frac{1}{e}$ corresponds to a fizzle and $\delta > \frac{1}{e}$ corresponds to an explosion. We will analyze the system when $\delta = \frac{1}{5}$ and $\delta = 1$.

Fizzle Case ($\delta = 0.2$)

To begin, we wanted to find out the asymptotic value of the fizzle solution. This was found by finding the root of the equation:

$$\frac{e^{\theta_{\text{fiz}}}}{\theta} - \frac{1}{\delta} = 0$$

This represents an accurate approximation to the long term solution of the problem. This equation was solved using Brent's Method of root finding, which combines the bisection method, secant method, and inverse quadratic interpolation to find the root. Code for this method can be seen in the Appendix. The value that θ appraoches asymptotically is 0.25917. Next, we wanted to determine the exact solution to the ode found in the problem statement, $\frac{d\theta}{d\sigma} = \delta e^{\theta} - \theta$, subject to the initial condition $\theta(0) = 0$. To do this, the Runge-Kutta 4 method of ode solving was used. The values of θ were evaluated until the 6th significant figure was unchanging at $\sigma = 20$. Finally, this solution was plotted along with the late solution found above, and the early solution to the problem $\theta = \frac{\delta}{\delta - 1}\left(e^{(\delta-1)\sigma} - 1\right)$. The graph for this can be seen in figure 1.

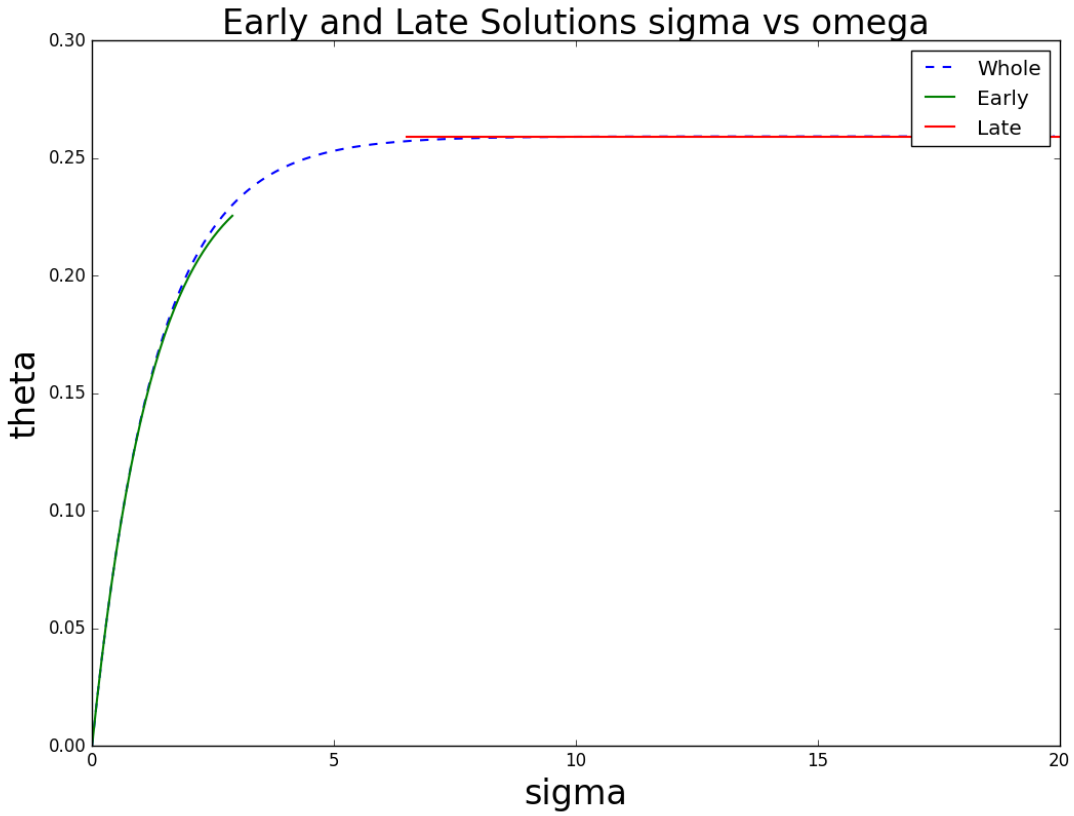

Figure 1: Graph of the early solution, late solution, and RK4 solution

In this graph, σ is proportional to time, and θ is proportional to energy and related to temperature. The fact that it has an asymptotic value for $\delta = 0.2$ means that as much energy is being lost to the environment as is being generated in the chemical reaction. As time increases, the energy in the container approaches the asymptote. As the energy gets closer to the asymptotic value more and more energy is lost to the environment until the values become equal at infinite time. One can also see that the short and long term solutions, shown as green and red lines, are accurate for large portions of the problem. With the numerical solution really only being necessary between 2.5 and 7.5.

Explosion Case ($\delta = 1$)

In the explosion case, delta is equal to 1 which is greater than delta critical. Since, delta is proportional to 1/h, this physically means that the amount of heat lost to the surrounding environment was small enough to allow the explosion to occur. Due to this, the temperature will increase asymptotically to infinity which would

render our integration solver ineffective. In order to overcome this limitation, we reverse the the variables and invert the differential equation to get $\frac{d\sigma}{d\theta} = \frac{1}{\delta e^{\theta} - \theta}$. Using fourth order Runge-Kutta to integrate the dimensionless differential equation, we found a numeric approximation of sigma in terms of theta with the initial condition $\sigma(0) = 0$. In order to determine the asymptotic value sigma explosion we used Simpson's method to approximate the integral from 0 to infinity of the differential equation. We found this value to be 1.3951 which matched the limit of sigma found in the numeric approximation. Using this value for sigma explosion, we were able to create a long term approximation for sigma valid for sufficiently large values of theta. We also used the same short term approximation as in the fizz case to approximate sigma for theta close to zero.

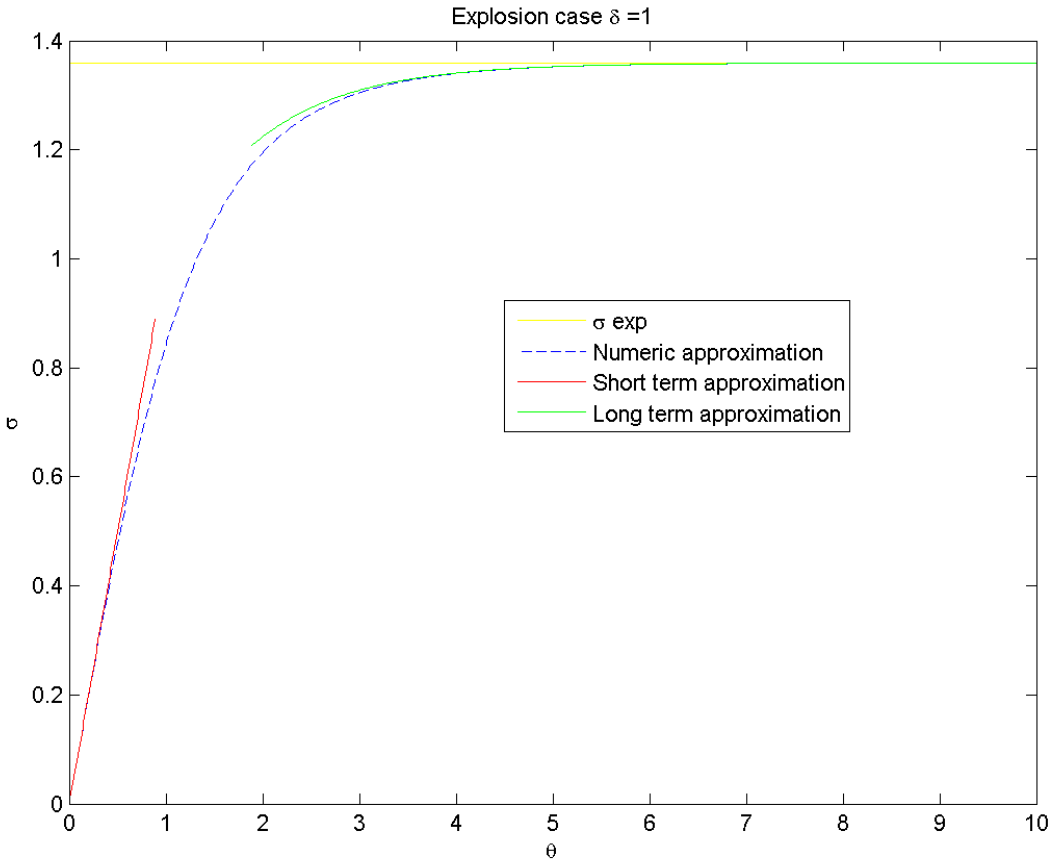

Figure 2: The graph of σ and θ for the explosion case, showing that σ asymptotes at 1.3951

When theta approaches zero the differential equation approaches $\frac{1}{e^{0} - 0}$ which is 1 so the short term solution is the line sigma = theta. When we plug in the dimensional variables we find that time and temperature are linearly related for time close to 0. When theta goes to infinity the differential equation goes to $\frac{1}{e^{\theta}} = 0$. This means that as sigma (which is proportional to time) approaches the critical value of sigma explosion theta (proportional to temperature) ?blows up? to a very large value as the explosion happens and large amounts of heat generated by the reaction. As the system approaches the critical time, enough fuel has been burned up to sufficiently increase the energy of enough particles past the activation energy for the problem and the temperature increases exponentially as the energy used in activation the reaction is more than replaced by the energy given off in the reaction.

Conclusion

We have analysed the non-adiabatic explosion problem in two separate instances using many numerical techniques, in one instance the heat was allowed to diffuse into the environment enough to cause the reaction to fizzle out and approach a limiting temperature. In the other instance, heat was retained sufficiently enough to allow the reaction to give off enough heat to cause an explosion. In both instances, similar asymptotic behavior was displayed, with the explosion having a critical time where temperature grew toward infinity and the fizzle having a critical temperature that was never surpassed as time grew to infinity. We learned the value that numerical methods have in modeling when analytic solutions are difficult or impossible to find.

Appendix: Part 1

Code for part 1, the fizzle case

```
#numpy is what we will use for the majority of our calculations
import numpy as np

#library for ploting
import matplotlib.pyplot as plt

#library for exiting if stuff is bunk
import sys


#delta for the fizzle
delta=0.2

#diff eq to approximate with rk4, x is theta
dfizzle="0.2*np.exp(x)-x"


#initial condition
x0=0.0
sig0=0.0

#step size to use
h=0.1

#steps
step=200

#a nice thing for generating for loops
def my_range(start,end,step):
    while start<=end:
        yield start
        start+=step

#initialize arrays
omega=np.zeros((step,1))
sig=np.zeros((step,1))

omega[0]=x0
sig[0]=sig0


#now to do rk4 for this problem
for i in my_range(0,step-2,1):
    x=omega[i]
    sig[i+1]=sig[i]+h
    k1=eval(dfizzle)
    x=omega[i]+h*k1/2
    k2=eval(dfizzle)
    x=omega[i]+h*k2/2
    k3=eval(dfizzle)
    x=omega[i]+h*k3
```

```
52      k4=eval(dfizzle)
53      omega[i+1]=omega[i]+h/6*(k1+2*k2+2*k3+k4)
54
55  #print(omega[step-2])
56  #print(omega[step-1])
57
58
59  #gotta install the plotting stuff first
60  #plt.plot(sig,omega)
61  #plt.show()
62
63
64  ####################################
65  #Now we will do the root finding method to find the theta that
66  #gives the approximate llong term solution
67
68  #Brent's method cause doing new things is good for you
69
70  approx="np.exp(x)/x-5"
71
72  a=0.2
73  b=2.0
74
75
76
77  x=a
78  fa=eval(approx)
79  x=b
80  fb=eval(approx)
81
82
83  if fa*fb>=0:
84      sys.exit()
85
86  fa1=np.absolute(fa)
87  fb1=np.absolute(fb)
88  if fa1<fb1:
89      r=a
90      a=b
91      b=r
92      r=fa
93      fa=fb
94      fb=r
95  c=a
96  fc=fa
97
98  s=0
99  fs=50
100
101 delt=0.000001
102
103 flag=1
104 d=0
105
```

```
while (np.absolute(fb)>=0.0000001) & (np.absolute(fs)>=0.0000001)
    :
    if ((fa!=fc) & (fb!=fc)):
        s=a*fb*fc/((fa-fb)*(fa-fc))+b*fa*fc/((fb-fa)*(fb-fc))+c*
           fa*fb/((fc-fa)*(fc-fb))
    else:
        s=b-fb*(b-a)/(fb-fa)
    if  (~((3*a+b)/4<s<b))|(flag&(np.absolute(s-b)>=(np.absolute(
       b-c)/2)))|(~flag&(np.absolute(s-b)>=np.absolute(c-d)/2))|(
       flag&(np.absolute(b-c)<delt))|(flag&(np.absolute(c-d)<delt
       )):
        s=(a+b)/2
        flag=1
    else:
        flag=0
    x=s
    fs=eval(approx)
    d=c
    c=b
    x=c
    fc=eval(approx)
    if (fa*fs)<0:
        b=s
        x=b
        fb=eval(approx)
    else:
        a=s
        x=a
        fa=eval(approx)
    if np.absolute(fa)<np.absolute(fb):
        r=a
        a=b
        b=r
        r=fa
        fa=fb
        fb=r

if fb<fs:
    print (b)
else:
    print (s)


#########################
#here, we will be plotting the early and late solutions with the
   numeric solution.
shortt="(-.25)*(np.exp(-.8*y)-1)"
step=30
early=np.zeros((step,1))
yx=np.zeros((step,1))
for n in my_range(0,step-2,1):
    yx[n+1]=yx[n]+h
    y=yx[n]
    early[n]=eval(shortt)
```

```
y=yx[n+1]
early[n+1]=eval(shortt)

step=135
late=np.ones((step,1))*b
yr=np.linspace(6.5,20,step)



line1=plt.plot(sig,omega,label="Whole",linewidth=1.5,linestyle="
    --")
line2=plt.plot(yx,early,label="Early",linewidth=1.5)
line3=plt.plot(yr,late,label="Late",linewidth=1.5)
plt.title("Early and Late Solutions sigma vs omega",fontsize=24)
plt.xlabel("sigma",fontsize=24)
plt.ylabel("theta",fontsize=24)
plt.legend()
plt.show()
```

Appendix: Part 2

Helper functions for part 2, the explosion case
The function to be integrated:

```
function [ sigmaprime ] = F( theta )
%differential equation of dimensionless variables
sigmaprime = 1/(exp(theta) - theta) ;

end
```

Helper function to run Runge Kutta 4:

```
function [ m1 ] = RangeK( thetai, sigmai, h, N )
%Uses other RK4 function as a helper to run RK4 the specified number of
    iterations
% This code was used to create a 2 column matrix which was plotted as the
% numeric solution
m1 = zeros(N, 2);
m1(1,1) = thetai;
m1(1,2) = sigmai;
th = thetai;
sig = sigmai;
for i = 2:N
    sig = RK4(th, sig, h);
    th = th + h;
    m1(i, 1)= th;
    m1(i, 2) = sig;



end
```

Runga Kutta 4:

```
function [ sigmanew ] = RK4( theta, sigma, h )
%Run RK4 runs a single iteration
%   Detailed explanation goes here
k1 = h * F(theta);
k2 = h * F(theta + h/2);
k3 = h * F(theta + h/2);
k4 = h * F(theta + h);
sigmanew = sigma + k1/6 + k2/3 + k3/3 + k4/6;

end
```

Simpson's Method:

```
function [ I ] = Simpson( xi,h, N)
%Apply Simpson's method to F(x) [F is the differential equation defined in
%F.m]
%   Simpson's method is used to integrate the differential equation to find
%   sigma explosion.
sum = 0;
for i = 1:N/2
    sum = sum + F(xi + h * (2*i -2)) + 4 *F(xi + h*(2*i -1)) + F(xi + h *(2*i)
        );

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

Short time solution function:

```
function [ y ] = ST( x )
%Short term approximation for explosion found using short term
%approximation from fizz case, because delta = 1 L'Hospital's rule was used
%to derive the solution (sigma = theta)
y = x;

end
```

Long time solution function:

```
function [ y ] = LT( th )
%After finding the asymptotic value from integration, it is used for the
%long term solution, which takes in a vector and returns a vector with
% each element pased through the long term approximation
for i = 1:length(th)
    y(i) = 1.3591 - 1/(exp(th(i)));
end
```