MATH 376: Numerical Analysis

Final Project

Quan Vu

October 26, 2017

Chapter 1

Greenhouse gases and pH

1.1 Abstract

This chapter aims at examining the relationship between the steady rise in atmospheric levels of several greenhouse gases and the pH of rainwater within the coressponding areas. In particular, this project looks at the annual levels of atmospheric carbon dioxide (CO_2) from the year 1959 to 2016 in Mauna Loa, Hawaiii. By computing the pH of water using the given data, it can be shown that the pH of rainwater has decreased from 5.63 to 5.58 over the years, and the trend shows that the pH level will likely drop lower.

1.2 Introduction

1.2.1 Background information

It is well documented that the atmospheric levels of several greenhouse gases have been increasing over the past 57 years. It is also know that within areas with generally low human activities, carbon dioxide is the primary determinant of the pH of rainwater.

1.2.2 Problem description

This project aims at using the existing data regarding the levels of atmospheric CO_2 around Mauna Loa to calculate the pH of rainwater in the same region over the years. This can be done with the help of five equations governing the chemistry of rainwater:

$$K_1 = \frac{10^6[H^+][HCO_3^-]}{K_H CO_2} \tag{1}$$

$$K_2 = \frac{[H^+][CO_3^{-2}]}{[HCO_3^-]} \tag{2}$$

$$K_\omega = [H^+][OH^-] \tag{3}$$

$$c_T = \frac{K_H CO_2}{10^6} + [HCO_3^-] + [CO_3^{-2}] \tag{4}$$

$$0 = [HCO_3^-] + 2[CO_3^{-2}] + [OH^-] - [H^+] \tag{5}$$

where K_H is Henry's constant, K_1, K_2 and K_ω are equilibrium coefficients. The five unknowns are c_T = total inorganic carbon, HCO_3^- = bicarbonate, $[CO_3^{-2}]$ = carbonate, $[H^+]$ = hydrogen ion, $[OH^-]$ = hydroxyl ion. One major assumption with this approach is that we fix CO_2 as the sole factor contributing to the pH of rainwater. In reality, many other greenhouse gases can also contribute to the fluctuation of pH.

1.2.3 Outline

Given the values $K_H = 10^{-1.46}$, $K_1 = 10^{-6.3}$, $K_2 = 10^{-10.3}$, $K_\omega = 10^{-14}$ and the annual CO_2, we can reduce equation (5) to be one that is in terms of $[H^+]$. We can compute $[H^+]$ and calculate the pH of rainwater using the equation:

$$pH = -log_{10}[H^+] \tag{6}$$

1.3 Numerical method

Firstly we convert the given equations so that the unknowns in (5) can be expressed in terms of $[H^+]$.
From (1):

$$[HCO_3^-] = \frac{K_H K_1 CO_2}{10^6 [H^+]} \tag{1a}$$

From (2) and from (1a):

$$[CO_3^{-2}] = \frac{K_2 [HCO_3^-]}{[H^+]} = \frac{K_H K_1 K_2 CO_2}{10^6 [H^+]^2} \tag{2a}$$

From (3):

$$[OH^-] = \frac{K_\omega}{[H^+]} \tag{3a}$$

From (4), (2a) and (3a):

$$c_T = \frac{K_H CO_2}{10^6} + \frac{K_H K_1 CO_2}{10^6 [H^+]} + \frac{K_H K_1 K_2 CO_2}{10^6 [H^+]^2} \tag{4a}$$

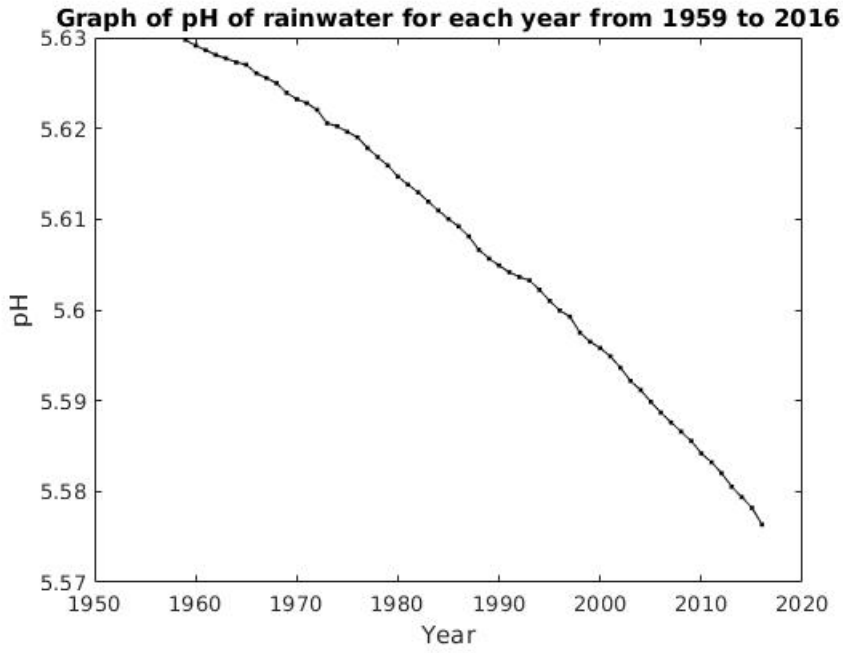
From 5, and from (1a), (2a), (3a):

$$0 = \frac{K_H K_1 CO_2 + 10^6 K_\omega}{10^6 [H^+]} + \frac{2 K_H K_1 K_2 CO_2}{10^6 [H^+]^2}] - [H^+] \tag{5a}$$

By using the bisection method on the above equation, we can find $[H^+]$.
A short snippet of Matlab code is attached in the appendix for this section to show how the bisection method finds the root of the function. To put in simple terms, the bisection method starts out with 2 initial guesses on both sides of a root of a function. A caveat that we need to care of is that the function must change sign around this root. By iteratively assumming that the root is in the middle of the boundary, the method continues cutting the estimated interval in half until a satisfactory estimation for the root is found.

1.4 Results

Using the methodology discussed above, we obtain the following results of the pH levels in rainwater in Mauna Loa, from 1959 to 2016:



For detailed results, consult the csv file in the project directory.

1.5 Discussion

From the calculations taken, the pH in the year 1959 was 5.63, and the pH calculated in 2016 was 5.58. While the decline seems small, we must keep in mind that pH is calculated by taking $log_{10}[H^+]$. This means that the concentration of Hydrogen ions in rainwater has increased over the year, and the trend does not seem to be stopping. In fact this trend can be modeled using the polyfit

function in Matlab to give:

$$pH = -5.523 \times 10^{-6} t^2 + 0.02101t - 14.33$$

where t denotes the year. If the trend continues, the pH of rainwater will drop below the threshold of 5.0, creating acid rain.

1.6 Bibliography

Data for annual atmospheric levels of CO_2 are taken from:
https://www.esrl.noaa.gov/gmd/ccgg/trends/data.html

1.7 Appendix for Chapter 1

1.7.1 Code for the bisection method

```
function [xm] = bisectM(fun, xleft, xright, n, TOL)
    a = xleft;
    b = xright;
    fa = feval(fun, a);
    fb = feval(fun, b);

    fprintf('n \t approximation \n');

    for i = 1:n
        xm(i) = (a + b)./2;
        fm = feval(fun, xm(i));

        fprintf('%d \t %12.10f \n', i - 1, xm(i))

        if (sign(fm) == sign(fa))
            a = xm(i);
            fa = fm;
        else
            b = xm(i);
            fb = fm;
        end

        if (i >= 2)
            absE = abs(xm(i) - xm(i-1));
            if absE < TOL
                break;
            end
        end
    end
end
```

1.7.2 Code for processing data

```
clear all;
close all;
clc;

format long;

% File path is hard-coded. Please put the data file in
    the same folder
% for this to work
```

```
fid = fopen('MaunaLoa.dat','r');
data = textscan(fid, '%f%f%f', 'HeaderLines', 22);
year = data{1};
mean = data{2};
unc = data{3};
NO_OF_YEAR = length(year);

pH = zeros([1 NO_OF_YEAR]);

% Defining constants to use
KH = 10^(-1.46);
K1 = 10^(-6.3);
K2 = 10^(-10.3);
Ko = 10^(-14);

% Since pH lies between 2 and 12, [H+] lies between 1e-12 and 1e-2
upperBound = 1e-2;
lowerBound = 1e-12;

% Tolerance of 1e^-10 should be enough. This is TOL for [H+].
% Having 10 correct decimal places for [H+] ensures at least 2 correct for pH
% (as we are taking log10, a small change in [H+] leads to an even smaller
% change in pH). We are taking 1e-10 since our search space is inherently
% small, and taking 1e-2 would terminate the search prematurely
TOL = 1e-10;

% Loop through the year, extract current level of CO2, and calculate pH
for i = 1:NO_OF_YEAR
    currentYear = year(i);
    CO2 =  mean(i);
    f = @(x) (KH * K1 * CO2 + 10^6 * Ko) ./ (x * 10^6) + (2 * KH * K1 * K2 * CO2) ./ (x.^2 * 10^6) - x;
    pH(i) = -log10(bisectM(f, lowerBound, upperBound, 300, TOL));
end

pH(1)
pH(NO_OF_YEAR)
x = linspace(1959, 2016, NO_OF_YEAR);
```

```
fid = fopen('output.csv', 'wt');
fprintf(fid, '%s,%s\n', 'Year','pH');
fclose(fid);
output = horzcat(x(:), pH(:));
dlmwrite('output.csv', output, 'delimiter', ',','-append');
polyfit(x, pH, 2)
plot(x, pH, 'k.-');
xlabel('Year');
ylabel('pH');
title('Graph of pH of rainwater for each year from 1959 to 2016');

function result = bisectM(fun, xleft, xright, n, TOL)
    a = xleft;
    b = xright;
    fa = feval(fun, a);
    fb = feval(fun, b);

    %fprintf('n \t approximation \n');

    for i = 1:n
        xm(i) = (a + b)./2;
        result = xm(i);
        fm = feval(fun, xm(i));

        %fprintf('%d \t %12.12f \n', i - 1, xm(i))

        if (sign(fm) == sign(fa))
            a = xm(i);
            fa = fm;
        else
            b = xm(i);
            fb = fm;
        end

        if (i >= 2)
            absE = abs(xm(i) - xm(i-1));
            if absE < TOL
                return;
            end
        end
    end
end
```

Chapter 2

Fluid flows in pipes and tubes

2.1 Abstract

This chapter aims at examining the flow of fluid through pipes and tubes. The project computes the dimensionless *friction factor* in turbulent flows. By computing this factor using the bisection method, the false position method, Newton's method and lastly the fixed point iteration method, this paper draws out comparisons between how effective these methods are and what are the constraints associated with them.

2.2 Introduction

2.2.1 Background information

Determining the friction factor is of great relevance to many field of engineering and science. Some of these include the flow of liquid and gases through pipelines and cooling systems. Scientists are interested in topics ranging from the flow in blood vessels to nutrient transmission through a plant's vascular system.

2.2.2 Problem description

In turbulent flows, the *Colebrook equation* provides a means to calculate the friction factor using the equation

$$0 = \frac{1}{\sqrt{f}} + 2.0 \log_{10}\left(\frac{\varepsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}}\right)$$

where ε is the roughness (m), D is the diameter (m), and Re is the Reynold's

number, as calculated by

$$Re = \frac{\rho V D}{\mu}$$

where ρ is the fluid's density (kg/m^3), V is the velocity (m/s), and μ is the dynamic viscosity ($N.s/m^2$)

2.2.3 Outline

By computing Reynold's number using the given values, we can substitute it back into the equation and using the numerical methods discussed, we can find the value of the friction factor. The given values are $\rho = 1.23kg/m^3$, $\mu = 1.79 \times 10^{-5} N.s/m^2$, $D = 0.005m$, $V = 40m/s$, and $\varepsilon = 0.0015mm$

2.3 Numerical methods

The numerical methods involved in the calculation of the friction factor are the bisection method, false position method, Newton's method, and fixed-point iteration method. Consult the attached Matlab project file to see the details of the calculations

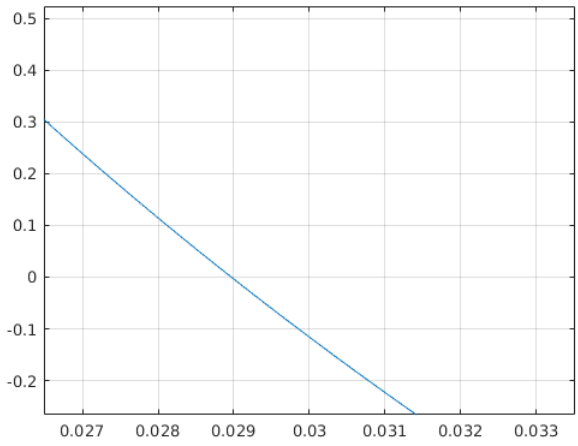
2.4 Results

2.4.1 Reynold's number

Using Matlab, we are able to determine Reynold's number = 1.374×10^4

2.4.2 Plotting graph and estimation

Using Matlab to plot the graph, the friction factor seems to be somewhere around 0.029



2.4.3 Bisection method and False position method

For both of these methods, we approximate the value of the friction factor by using a = 0.008 and b = 0.08, with a tolerance of 10^{-8}

Bisection method

The bisection method was discussed in chapter 1. For a quick reference as to how this method finds the root of a function, refer to this section in chapter 1 as well as the attached appendix for detailed code in Matlab.

The method took 26 iterations to calculate the answer. The esitmations converge to 0.02896781

False position method

The method of false position combines both the bisection method and the secant method. Starting with the same two initial guesses, the method draws the secant line to find a new estimation for the root, and depending on the sign of the function at the root, the boundaries for root estimation change accordingly. Sample code for the False position method is provided in the appendix.

The method took 31 iterations to calculate the answer. The estimations converge to 0.02896782

Discussion

We see that both method converges to the solution, however the bisection method is faster in this case.

2.4.4 Newton's method

Newton's method relies on using the tangent line at any point on the function. By finding the intersection of the line with the x-axis, we have a new estimation for the root, with which we can find a new tangent line and repeat the prosess until we find a reasonable estimate for the actual root.

We can calculate the derivative of the function quite easily:

$$h'(f) = \frac{-1}{2f\sqrt{f}} + 2.0 \left(\frac{log_{10}e \times \frac{-2.51}{2Ref\sqrt{f}}}{\frac{\varepsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}}} \right)$$

This is then used for the calculation for Newton's method in the Matlab file provided in the appendix.

With initial guess 0.008

Took 6 iteration to calculate the answer. The estimations converge to 0.02896781

With initial guess 0.08

The approximations do not converge. They fluctuate and eventually go to infinity

Discussion

We see that Newton's method converges extremely fast if we choose the correct initial guess. This is due to the nature of the function around the root.

2.4.5 Using fzero

We use the built-in fzero function with options = optimset('Display','iter', 'TolX', 1e-8).This function searches for a point near the guess where the sign of the function changes

With initial guess 0.008

The method does not converge to the solution. Complex function value encountered during search

With initial guess 0.08

The method manages to find an interval [0.0288, 0.116204] where the function changes sign. It then continues to search for the root by evaluating possible values in this interval, and eventually got to f = 0.0289678

Discussion

As opposed to Newton's method, fzero was able to find the root with initial guess 0.08, but was unable to do so with initial guess 0.008.

2.4.6 Fixed Point Iteration

Iteration function

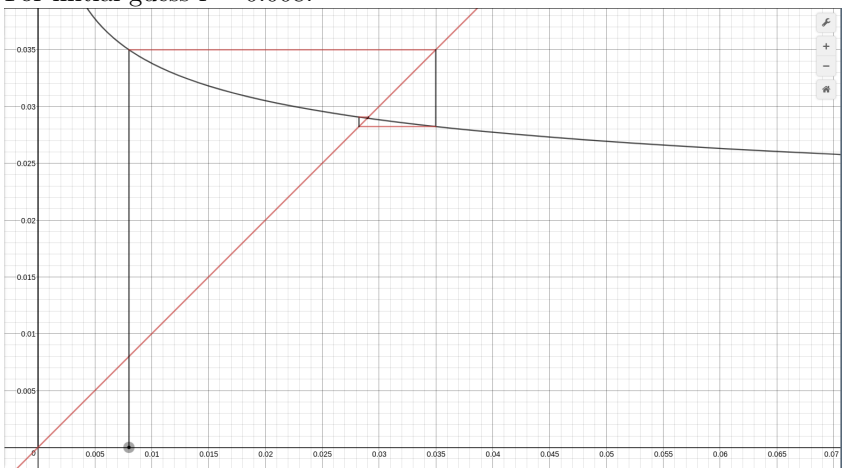
The fixed point method relies on whether we can find an iteration function $g(x)$ such that $f(x) = x - g(x)$ and $|g(xR)| < 1$ where xR is the root of the function. By repeatedly finding x for which $x = g(x)$, a reasonable estimate for the root is found. The code illustrating this method is attached in the appendix.

One of the possible iteration functions is:

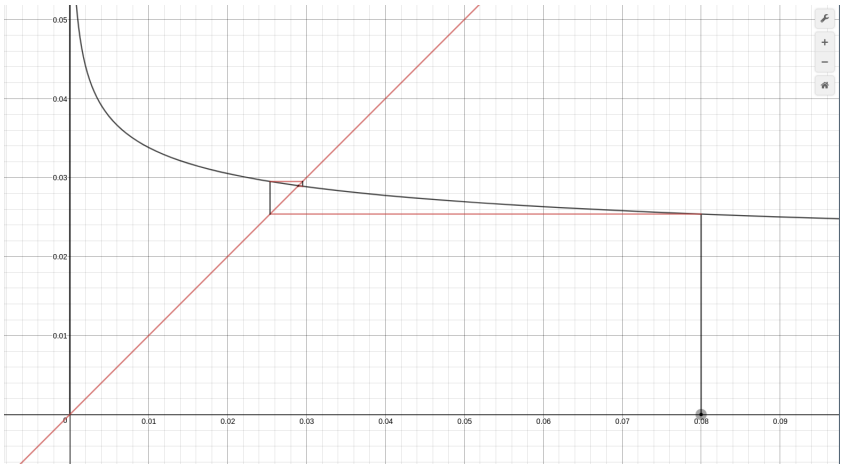
$$g(f) = \left(\frac{1}{2.0 \log_{10} \left(\frac{\varepsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}} \right)} \right)^2$$

Cobweb Diagrams

For initial guess f = 0.008:



For initial guess f = 0.08:



2.4.7 Discussion

With both initial guesses at 0.008 and 0.08, the method was able to find the approximation in 9 iterations. This is because $|g'(x)| < 1$ at the root and therefore guarantees convergence.

2.4.8 Appendix for chapter 2

2.4.9 Code for the false position method

```
function [xm] = falsePositionM(f, x0, x1, n, TOL)
    xm(1) = x0;
    xm(2) = x1;
    fprintf('n \t approximation \t error \n');
    b = x0;
    a = x1;
    for i = 3:n
        xm(i) =  a - f(a) * (b - a) / (f(b) - f(a));
        error = Inf;
        if i > 3
            error = abs(xm(i) - xm(i-1));
        end
        relerror = error / abs(xm(i-1));
        fprintf('%d \t %12.8f \t %12.8f\n', i - 3, xm(i),
            relerror);
        if (error < TOL || f(xm(i)) == 0)
            break;
        end
        if (sign(f(xm(i))) == sign(f(a)))
```

```
19              a = xm(i);
20          else
21              b = xm(i);
22          end
23
24      end
25  end
```

2.4.10 Code for Newton's method

```
1   function [xm] = NewtonsM(f, fder, x0, n, TOL)
2       xm(1) = x0;
3       fprintf('n \t approximation \t error \n');
4       for i = 2:n
5           xm(i) = xm(i-1) - f(xm(i-1)) / fder(xm(i-1));
6           error = abs(xm(i) - xm(i-1));
7           relerror = error / abs(xm(i-1));
8           fprintf('%d \t %12.8f \t %12.8f \n', i - 1, xm(i)
                , relerror)
9           if (error < TOL)
10              break;
11          end
12      end
13  end
```

2.4.11 Code for fixed point method

```
1   function [xm] = fixedpointM(g, x0, n, TOL)
2       xm(1) = x0;
3       for i = 2:n
4           xm(i) = g(xm(i-1));
5           absE = abs(xm(i) - xm(i-1));
6           relerror = absE / abs(xm(i-1));
7           fprintf('%d \t %12.8f \t %12.8f \n', i - 1, xm(i)
                , relerror)
8           if (absE < TOL)
9               break;
10          end
11      end
12  end
```

2.4.12 Code for this chapter

```
1   clear all;
2   close all;
3   format long;
4
```

```
% Project 2
% Part a
    rho = 1.23;
    V = 40;
    D = 0.005;
    mu = 1.79e-5;
    % Gives Re = 1.374301675977654e+04
    Re = rho * V * D / mu;

% Part b
    eps = 0.0015e-3; % Given in millimeters
    f = @(x) 1 ./ (sqrt(x)) + 2 * log10(eps / (3.7 * D) +
        2.51 ./ (Re .* sqrt(x)));
    x = linspace(0.008, 0.08, 1000);
    plot(x, f(x));
    grid on;
    % From the estimation, it looks like the root is
        somewhere around 0.029

% Part c
    close all;
    bisectM(f, 0.008, 0.8, 50, 1e-8);
    falsePositionM(f, 0.008, 0.8, 50, 1e-8);

% Part d
    df = @(x) (-1) ./ (2 *x * sqrt(x)) + 2 * (log10(exp
        (1)) * 1 / (eps / (3.7 * D) + 2.51 ./ (Re .* sqrt(
        x))) * (-2.51) ./ (2 * Re * x * sqrt(x)));
    NewtonsM(f, df, 0.008, 50, 1e-8);
    NewtonsM(f, df, 0.08, 50, 1e-8);

% Part e
    options = optimset('Display','iter', 'TolX', 1e-8);
    fzero(f, 0.008, options);
    fzero(f, 0.08, options);

% Part f
    % This function does not work
    g = @(x) (-1 ./ (2 * log10(eps / (3.7 * D) + 2.51 ./
        (Re .* sqrt(x))))).^2;
    fixedpointM(g, 0.008, 50, 1e-8);
    fprintf('\n');
    fixedpointM(g, 0.08, 50, 1e-8);
```