CFD HW 2

Semyon Lopatkin

# Information to know

**Root is considered to be a solution when error goes under 1%**

*And*

**Both codes incorporate 2 numerical methods and user is able to pick which to use **

# Part 1

## Code:

```
from __future__ import division
import time
import math
from operator import itemgetter

#import numpy

Re1 = 174981
Re2 = 94682
Re3 = 80281
Re4 = 43955
Re5 = 36309
Re6 = 21680
Re7 = 14628
Re8 = 36309
Re9 = 80281
n = 0 #probs needs to be inside the loop smh
pn = 1 #pipe number
nn = 1 #iteration counter
#u = #probs needs to be inside the loop smh
#l = #probs needs to be inside the loop smh


a = [Re1, Re2, Re3, Re4, Re5, Re6, Re7, Re8, Re9, 0] # array
L = [2,4,2,4,2,4,8,2,2]

method = int(input("1 for Secant Method, 2 for False Position Method"))

if method == 1:
    x0 = float(input("what is initial xi-1?"))
    x1 = float(input("what is initial xi?"))
```

```
    xx0 = x0
    xx1 = x1
if method == 2:
    xl = float(input("what is initial xl?"))
    xu = float(input("what is initial xu?"))
    xxl = xl
    xxu = xu
    xxr = 0 #placeholder, so the ERR function works correctly


#fof = 4*math.log(Re*f^0.5,10)-f^-.5-0.4 #refference, nstbh
#dfof = 2*(math.ln(10)*f+2x^-1.5) #derrivative of fof, nstbh
#fu = 4*math.log(Re*u^0.5,10)-u^-.5-0.4


#dif = x0-x1

#fr = fu - (fu*(dif))/(fx0-fx1)

while method == 1: #SECANT
    a = [Re1, Re2, Re3, Re4, Re5, Re6, Re7, Re8, Re9]
    time.sleep(0.1)
    ind_pos = [n] # indexing the positon of a value in the array
    Re = itemgetter(*ind_pos)(a) #get the needed value
    print (Re) #debug for my sanity

    time.sleep(0.1)
    fx0 = float(4*math.log(Re*x0**0.5,10)-x0**-.5-0.4)
    #print(fx0)
    fx1 = float(4*math.log(Re*x1**0.5,10)-x1**-.5-0.4)
    err = (x1-x0)/x1


    x2 = x1 - ((float(fx1)*(float(x0)-float(x1)))/((float(fx0)-float(x1))))

    err = (x1-x0)/x1
```

```
    print("Iteration: ", nn, "xi-1: ", x0, "xi: ", x1, "xi+1: ", x2, "f(xi): ", fx1, "f(xi-1) :", fx0, "ERROR: ", err)
    nn = nn +1

    x0 = x1
    x1 = x2

    if err < 0.005:

        print("SUCCESS, PIPE", pn, "- ERROR UNDER 1%")
        pn=pn+1
        n=n+1
        x0 = xx0
        x1 = xx1
    elif pn == 9 or n == 9:
        print("DONE")
        break

while method == 2: #FALSE POSITION
    a = [Re1, Re2, Re3, Re4, Re5, Re6, Re7, Re8, Re9, 0]
    time.sleep(0.1)
    ind_pos = [n] # indexing the positon of a value in the array
    Re = itemgetter(*ind_pos)(a) #get the needed value
    print (Re) #debug for my sanity
    fxu = 4*math.log(Re*xu**0.5,10)-xu**-.5-0.4
    fxl = 4*math.log(Re*xl**0.5,10)-xl**-.5-0.4
    xr = xu-((fxu*(xl-xu))/(fxl-fxu))
    fxr = 4*math.log(Re*xr**0.5,10)-xr**-.5-0.4
    fxrfxl = fxl*fxr
    ERR = abs((xxr-xr)/xr)
    xxr=xr
    if fxrfxl < 0:
        xu = xr
        #print("XU: ", xu)
    elif fxrfxl > 0:
        xl = xr
```

```
    #print("XL: ", xl)

elif fxrfxl == 0:
    print("root is:", xr)



print("Iteration: ", nn, "xl: ", xl, "xu: ", xu, "xr: ", xr, "f(xl): ", fxl, "f(xr) :", fxr, "f(xr)*f(xl): ", fxrfxl, "ERROR: ", ERR)
nn = nn +1
if ERR < 0.005:

    print("SUCCESS, PIPE", pn, "- ERROR UNDER 1%")
    pn=pn+1
    n=n+1
    xl = xxl
    xu = xxu
elif pn > 9 or n > 9:
    print("DONE")
    break
```

## Methods Comparison

Equation of the interest in the **Part 1** is:

$$4Log\left(Re * \sqrt{f}\right) - f^{-0.5} - 0.4 = 0$$

Equation 1.

## False Position Method

```
Iteration:  623 xl:  0.0001 xu:  0.004969345097634994 xr:  0.004969345097634994 f(xl):  -88.78154890588534 f(xr) : 0.42536083736859565 f(xr)*f(xl):  -37.76419398548832 ERROR:  0.005145107865
435038
80281
Iteration:  624 xl:  0.0001 xu:  0.004946126843419846 xr:  0.004946126843419846 f(xl):  -88.78154890588534 f(xr) : 0.38803656590847735 f(xr)*f(xl):  -34.45048735347528 ERROR:  0.004694229434
4992416
SUCCESS, PIPE 9 - ERROR UNDER 1%
```

Image 1. False Position Method

Only took 624 iterations for the <u>False Position</u> to calculate friction factor for all 9 pipes.

## Secant Method

```
Iteration:  37 xl:  0.004701125375499179 xu:  0.0035 xr:  0.004701125375499179 f(xl):  -0.11506361128425058 f(xr) : -0.02189839459738574 f(xr)*f(xl):  0.0025197083637027258 ERROR:  0.0113223
48835791843
80281
Iteration:  38 xl:  0.004711341611950063 xu:  0.0035 xr:  0.004711341611950063 f(xl):  -0.02189839459738574 f(xr) : -0.004191247298807377 f(xr)*f(xl):  9.178158720451103e-05 ERROR:  0.002168
4346609404507
SUCCESS, PIPE 9 - ERROR UNDER 1%
```

Image 2. Secant Method

<u>Unlike</u> for the <u>False Position,</u> it only takes 38 iterations to reach the root for all 9 pipes.

## Problem Analysis

| Q1 (m3/s) | p (kg/m3) | mu (Ns/m2) |
|---|---|---|
| 1 | 1.23 | 1.79E-05 |

| Pipe | Length (m) | D (m) | S (m3) | Q (m3/s) | V (m/s) |
|---|---|---|---|---|---|
| 2 | 4 | 0.5 | 0.1963 | 0.5411 | 2.7571975 |
| 3 | 2 | 0.5 | 0.1963 | 0.4588 | 2.3378344 |
| 4 | 4 | 0.5 | 0.1963 | 0.2512 | 1.28 |
| 5 | 2 | 0.5 | 0.1963 | 0.2075 | 1.0573248 |
| 6 | 4 | 0.5 | 0.1963 | 0.1239 | 0.6313376 |
| 7 | 8 | 0.5 | 0.1963 | 0.0836 | 0.4259873 |
| 8 | 2 | 0.5 | 0.1963 | 0.2075 | 1.0573248 |
| 9 | 2 | 0.5 | 0.1963 | 0.4588 | 2.3378344 |

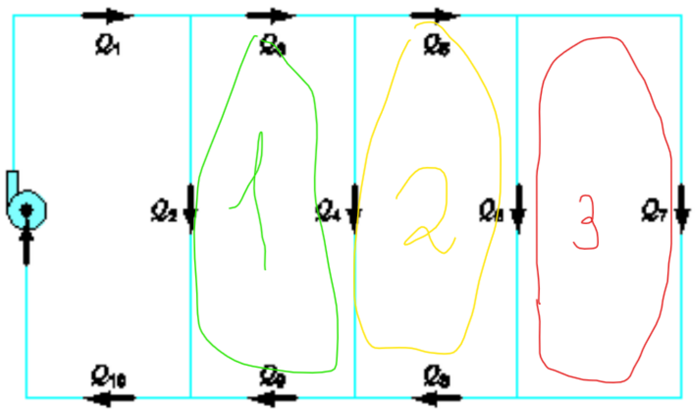
| Re | f (secant method) | f (false position) | dP (kPa) | dP (kPa) | Method difference |
|---|---|---|---|---|---|
| 9.47E+04 | 0.00478 | 0.00449 | 0.1786 | 0.1678 | 6% |
| 8.03E+04 | 0.00495 | 0.00464 | 0.0665 | 0.0623 | 7% |
| 4.40E+04 | 0.00567 | 0.00522 | 0.0457 | 0.0420 | 9% |
| 3.63E+04 | 0.00593 | 0.00544 | 0.0163 | 0.0149 | 9% |
| 2.17E+04 | 0.00674 | 0.00604 | 0.0132 | 0.0118 | 12% |
| 1.46E+04 | 0.00743 | 0.00653 | 0.0133 | 0.0117 | 14% |
| 3.63E+04 | 0.00593 | 0.00544 | 0.0163 | 0.0149 | 9% |
| 8.03E+04 | 0.00495 | 0.00464 | 0.0665 | 0.0623 | 7% |

Table 1.

False Position estimated the root to be lower than the Secant Method root estimates, therefore pressure drops ended up being smaller across the pipes. Additionally values are checked through established physical relationships:

## Checking

Physical relationships are established inside of these loops:




That gets us:

$$\text{Loop 3}: P_6 + P_7 = 0$$

$$\text{Loop 2}: P_4 + P_5 + P_6 + P_8 = 0$$

$$\text{Loop 1}: P_2 + P_3 + P_4 + P_9 = 0$$

*Set Of Equations 1. Pressure drop relations*

<u>Secant Method</u>

**Loop 3** 0.0133-0.0132 = **0.0001**

**Loop 2** 0.0163-0.0457+0.0163+0.0132 = **0.0001**

**Loop 1** 0.1786-0.0665-0.0457-0.0665 = **-0.0001**

<u>False Position Method</u>

**Loop 3** 0.0118-0.0117 = **0.0001**

**Loop 2** 0.0420-0.0149-0.0118-0.0149 = **0.0004**

**Loop 1** 0.1678-0.0623-0.0420-0.0623 = **0.0012**

This tells us that both methods are pretty accurate at getting the root of the function, yet <u>Secant Method</u> is more accurate as it deviates less from the *Set Of Equations 1.*

# First 2 Iterations by Hand

| iter | $x_{i-1}$ | $x_i$ | $f(x_i)$ | $f(x_{i-1})$ | $x_{i+1}$ | $\varepsilon_a$ | |
|---|---|---|---|---|---|---|---|
| 1 | 0,003 | 0,0035 | -1,24 | -2,731 | 0,0037 | 100% | pipe 1 |
| 1 | 0,003 | 0,0035 | -2,31 | -3,798 | 0,0038 | | pipe 2 |
| 1 | 0,003 | 0,0035 | -2,596 | -4,085 | 0,0038 | | pipe 3 |
| 1 | 0,003 | 0,0035 | -2,596 | -4,085 | 0,0038 | | pipe 4 |
| 1 | 0,003 | 0,0035 | -3,643 | -5,131 | 0,0038 | | pipe 5 |
| 1 | 0,003 | 0,0035 | -3,915 | -5,463 | 0,0039 | | pipe 6 |
| 1 | 0,003 | 0,0035 | -4,871 | -6,39 | 0,0039 | | pipe 7 |
| 1 | 0,003 | 0,0035 | -5,55 | ~~[illegible]~~ -7,04 | 0,0039 | | pipe 8 |
| 1 | 0,003 | 0,0035 | ~~-5,981~~ -3,98 | ~~[illegible]~~ [illegible],463 | 0,0039 | | pipe 9 |

~~pipe 10~~

$$x_{i+1} = x_i - \frac{f(x_i)\cdot(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

initial $x_{i-1} = 0{,}003$
$x_i = 0{,}0035$

$$f(x_i) = 4\log_{10}(Re\cdot f^{\frac{1}{2}}) - f^{-0{,}5} - 0{,}4 = 0$$

Pipe 1 $f(x_i) = 4\log_{10}(174981{,}5\cdot f^{\frac{1}{2}}) - f^{-0{,}5} - 0{,}4 = -1{,}243$

$x_{i+1} = 0{,}0035 - \frac{-1{,}24\,(0{,}0035 - 0{,}003)}{-2{,}731 + 1{,}24} = 0{,}0037$

Pipe 2 $f(x_i) = 4\log_{10}(94682{,}5\, f^{\frac{1}{2}}) - f^{-0{,}5} - 0{,}4 = -$

Pipe 3 $f(x_i) = 4\log_{10}(802815\cdot x_i^{\frac{1}{2}}) - x_i^{-0{,}5} - 0{,}4 = -2{,}596$

Pipe 4 $f(x_i) = 4\log_{10}(43955{,}4\cdot x_i^{\frac{1}{2}}) - x_i^{-0{,}5} - 0{,}4 =$

Pipe 5 $f(x_i) = 4\log_{10}(86308{,}7\cdot x_i^{\frac{1}{2}}) - x_i^{-0{,}5} - 0{,}4 = -3{,}643$

Pipe 6 $f(x_i) = 4\log_{10}(21680{,}21\cdot x_i^{\frac{1}{2}}) - x_i^{-0{,}5} - 0{,}4 =$

Pipe 7 $f(x_i) = 4\log_{10}(14624{,}5\cdot x_i^{\frac{1}{2}}) - x_i^{-0{,}5} - 0{,}4 = -4{,}871$

Pipe 8 $f(x_i) = 4\log_{10}(36308{,}7\cdot x_i^{\frac{1}{2}}) - x_i^{-0{,}5} - 0{,}4 =$

Pipe 9 $f(x_i) = 4\log_{10}(80281{,}5\cdot x_i^{\frac{1}{2}}) - x_i^{-0{,}5} - 0{,}4 = -5{,}431$

# Part 2

## Code:

```
import time
import math

print("Starting")
print(" The equation of interest is: 1-40.775*((3+xl)/(3*xl+((xl**2)/2))**3)=0 ")
xl = float(input("what is initial xl?"))
xu = float(input("what is initial xu?"))
method = int(input("1 for Bisection Method, 2 for False Position Method"))

n=1
xr = xl/2+xu/2
xxr = 0 #placeholder, so the ERR function works correctly

while method == 1:
    time.sleep(1)
    xr = xl/2+xu/2
    ERR = abs((xxr-xr)/xr)
    print (ERR)
    fxl = 1-40.775*((3+xl)/(3*xl+((xl**2)/2))**3)
    fxr = 1-40.775*((3+xr)/(3*xr+((xr**2)/2))**3)
    fxrfxl = fxl*fxr
    xxr = xr #store old xr
    print("Iteration: ", n, "xl: ", xl, "xu: ", xu, "xr: ", xr, "f(xl): ", fxl, "f(xr) :", fxr, "f(xr)*f(xl): ", fxrfxl, "ERROR: ", ERR)

    n = n+1
    if fxrfxl < 0:
        xu = xr
        #print("XU: ", xu)
    elif fxrfxl > 0:
        xl = xr
```

```
        #print("XL: ", xl)
    elif fxrfxl == 0:
        print("root is:", xr)



    #ERR = (xu-xr)/xr

    if ERR < 0.01:
        print("SUCCESS - ERROR UNDER 1%")
        break

while method == 2:
    time.sleep(1)
    fxu = 1-40.775*((3+xu)/(3*xu+((xu**2)/2))**3)
    fxl = 1-40.775*((3+xl)/(3*xl+((xl**2)/2))**3)
    xr = xu-((fxu*(xl-xu))/(fxl-fxu))
    fxr = 1-40.775*((3+xr)/(3*xr+((xr**2)/2))**3)
    fxrfxl = fxl*fxr
    ERR = abs((xxr-xr)/xr)
    #print("LOOK HEREEEEE:  ", xxr, xr) #debug
    xxr = xr#store old xr
    print("Iteration: ", n, "xl: ", xl, "xu: ", xu, "xr: ", xr, "f(xl): ", fxl, "f(xr) :", fxr, "f(xr)*f(xl): ", fxrfxl, "ERROR: ", ERR)
    if fxrfxl < 0:
        xu = xr
    elif fxrfxl > 0:
        xl = xr
    elif fxrfxl == 0:
        print("root is:", xr)

    n = n+1
    if ERR < 0.01:
        print("SUCCESS - ERROR UNDER 1%")
        break
```

## Methods Comparison

Equation of the interest in the **Part 2** is:

$$1 - 40.775\frac{3+y}{\left(3y+\frac{y^2}{2}\right)^3} = 0$$

Equation 2.

## Bisection Method

```
Iteration:  7 xl:  1.5 xu:  1.53125 xr:  1.515625 f(xl):  -0.030953086419752918 f(xr) : 0.0033762383758055847 f(xr)*f(xl):  -0.00010450499821999649 ERROR:  0.010309278350515464
0.0051813471502590676
Iteration:  8 xl:  1.5 xu:  1.515625 xr:  1.5078125 f(xl):  -0.030953086419752918 f(xr) : -0.013602161649019173 f(xr)*f(xl):  0.00042102888501753934 ERROR:  0.0051813471502590676
SUCCESS - ERROR UNDER 1%
```

Image 3. Bisection Method

From *Image. 3* it can be seen that only 8 iterations are needed for the Bisection method to get the solution for *Equation 2.*

The solution found by this numerical method is $x_r = 1.5078125$

Plugging this value into *Equation 2.* yields -0.0136 rather than 0.

## False Position

```
Iteration:  22 xl:  0.5 xu:  1.8149375383549944 xr:  1.7968643374581967 f(xl):  -32.25844333181611 f(xr) : 0.43096943889639083 f(xr)*f(xl):  -13.902403222383809 ERROR:  0.010058188879391
LOOK HEREEEEE:   1.7968643374581967 1.779766786646888
Iteration:  23 xl:  0.5 xu:  1.7968643374581967 xr:  1.779766786646888 f(xl):  -32.25844333181611 f(xr) : 0.4126436949605158 f(xr)*f(xl):  -13.31124325011501 ERROR:  0.009606624272116455
SUCCESS - ERROR UNDER 1%
```

Image 4. False Position

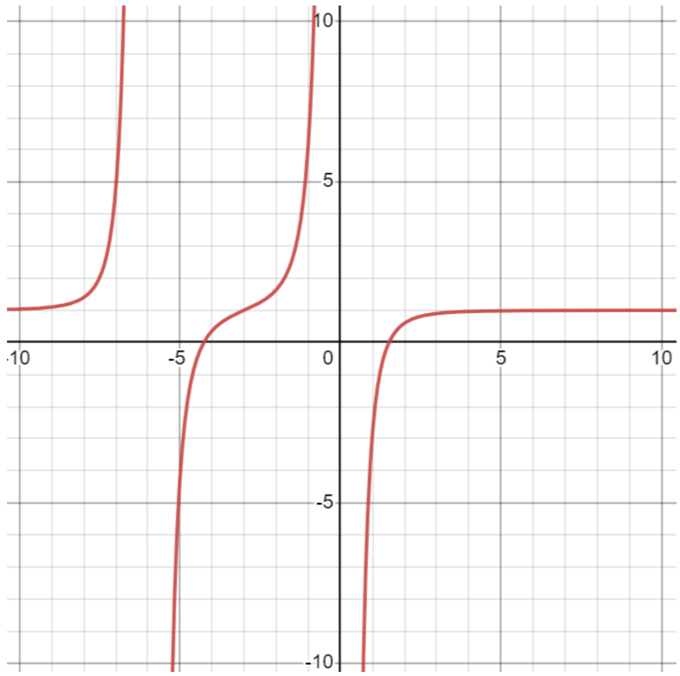
From *Image. 4* it can be seen that 23 iterations are needed for the False Position method to get the solution for *Equation 1.*

The solution found by this numerical method is $x_r = 1.77976$

Plugging this value into *Equation 1* yields 0.4126 rather than 0.

## **Analysis**

In the current situation Bisection Method wins over almost by 3-fold. And even if we plug in the xr values into the equation Bisection Method will be more accurate. But why is that? We can take a closer look at the graph of the equation of interest:




Graph 2.

While the Bisection Method is getting xr that is in the middle of "search" boundaries (xu and xl), which does not complicate things, unless any of the asymptotes were selected as boundary locations.

False Position on the other hand, does not tend to zero the bracket, rather it employs Secant lines to find the xr, which makes only 1 point move closer to the solution. This with conjunction of the specific bracket chosen (0.5-2.5, f' is very big) makes it take more iterations and makes them less accurate.

**First 2 Iterations by Hand**

| iter | $\varepsilon$ | $x_L$ | $x_u$ | $x_r$ | $f(x_L)$ | $f(x_r)$ | $f(x_L)\cdot f(x_r)$ |
|---|---|---|---|---|---|---|---|
| 1 | 100% | 0,5 | 2,5 | 1,5 | −32,26 | −0,031 | + |
| 2 | ~~33,3%~~ 25 | 1,5 | 2,5 | 2 | −0,031 | 0,6 | — |
| 3 | | 1,5 | 2 | | | | |

Bisection

$$\boxed{1-40{,}775\cdot\frac{3+y}{\left(3y+\frac{y^2}{2}\right)^3}=0}$$

$x_u = 2{,}5$

$x_L = 0{,}5$

$$x_r=\frac{2{,}5+0{,}5}{2}=1{,}5$$

$$\varepsilon=\frac{x_r-x_{r\,last}}{x_r}=0{,}\cancel{33{,}3}\,25\,\%$$

$$0=1-\frac{Q^2}{g\cdot S_c^3}\cdot B$$

$B=3+y$

$S_c=3y+\frac{y^2}{2}$

$g=9{,}8$

$Q=20\ \frac{m^3}{s}$

| iter | $\varepsilon$ | $x_L$ | $x_u$ | $x_r$ | $f(x_L)$ | $f(x_r)$ | $f(x_L)\cdot f(x_r)$ |
|---|---|---|---|---|---|---|---|
| 1 | 100% | 0,5 | 2,5 | 2,4508 | −32,26 | 1,974 | — |
| 2 | ~~2,65%~~ 1,96% | 0,5 | 2,4508 | ~~2,4508~~ 2,403 | −32,26 | 0,7812 | — |

False Position

$f(x_{u_2})=0{,}7998$

$$f(x_u)=1-40{,}775\cdot\frac{3+2{,}5}{\left(3\cdot 2{,}5+\frac{2{,}5^2}{2}\right)^3}=0{,}813$$

$x_u = 2{,}5$

$x_L = 0{,}5$

$$x_r=2{,}5-\left(\frac{0{,}813\cdot(0{,}5-2{,}5)}{\underbrace{f(x_L)}_{-32{,}26}-\underbrace{f(x_u)}_{0{,}818}}\right)=2{,}4508$$

$$\varepsilon=\frac{\cancel{2{,}3875}\ 2{,}4508\ \ (2{,}403)}{\cancel{2{,}3875}\ \ (2{,}403)}=0{,}0196$$