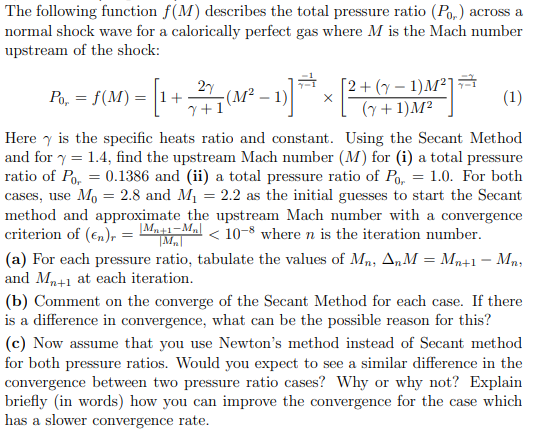
2021

# Test 1

AERO 5830 - S. HOSDER

MATT PAHAYO

Question 1



**Results**

**(a)**

****

**(b)**

**The case in which the total pressure ratio is 0.1386 converges much faster than the case where the total pressure ratio is 1. The reason is the function, the function iterations goes to zero without meeting the convergence criteria.**

**(c)**

**Newton’s method will converge faster because it has a higher rate of convergence than the Secant method. The convergence problem in case 2 can be fixed by using Newton’s method.**

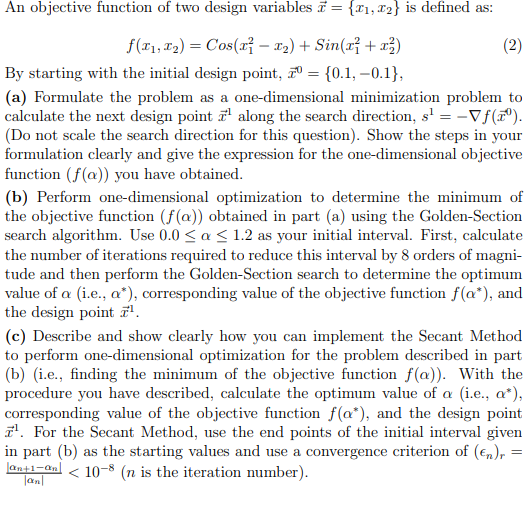
**Methodology**

1. Rearrange function so it equals zero
2. Use a root finder (secant method) – the only one in python, the rest in MATLAB



Solve the equation above until convergence criteria is met

**Question 2**



**Results**

1. **Text, letter

   Description automatically generated**

.09019

**First compute the gradient of the function:**

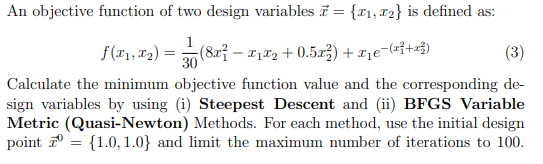
**Then get the search direction at**

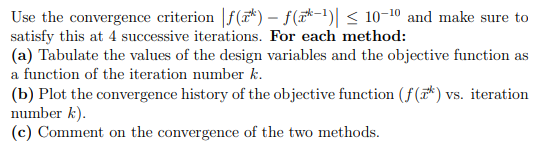
**Get F in terms of the guess, the search direction, and alpha.**

1. **The golden search algorithm is then used on the objective function to find the minimum value of alpha at the first iteration. It is found that the golden search algorithm takes 39 iterations to reduce the initial interval by eight orders of magnitude. The optimum value of alpha is found to be 0.626356740484508 at the first iteration.**
2. **Finding the minimum value of alpha is not limited to the golden search algorithm. A root finder may be used instead. To use a root finder such as secant method, the roots of the derivative of the objective function equal to zero are the optimum values of alpha.**
3. **Find derivative of the objective function and set it equal to zero.**
4. **Use a root finding technique such as secant method on the new objective function to find the roots/minimum.**

**Using secant method, it is found that the optimum value of alpha is 0.626356813471203 in 5 iterations.**

**Question 3**





1. 
2. Chart, line chart

   Description automatically generated
3. The BFGS algorithm converges faster than the steepest descent method.

## Methodology

Get the search direction from the gradient. Equation 1 will give a unit vector and is for the Steepest Descent Method.

⃗𝑠⃗⃗⃗𝑘 = ∇F(⃗𝑥⃗⃗⃗𝑘 )/‖∇𝐹(⃗𝑥⃗⃗⃗𝑘⃗⃗) ‖ (1)

Get alpha star by evaluating the function at the guess plus the search direction times alpha. Then find the minimum value of alpha by minimizing the alpha-function with a Golden Section algorithm and then fit a cubic function that is outputted by the Golden Section algorithm to get alpha star.

𝑑 𝐹(⃗𝑥⃗⃗⃗𝑘 +𝛼𝑘⃗𝑠⃗⃗⃗𝑘 ) = 0 (2)

𝑑𝛼

𝑥⃗⃗⃗⃗𝑘⃗⃗+⃗⃗⃗1 = ⃗𝑥⃗⃗⃗𝑘 +𝛼𝑘⃗𝑠⃗⃗⃗𝑘 (3)

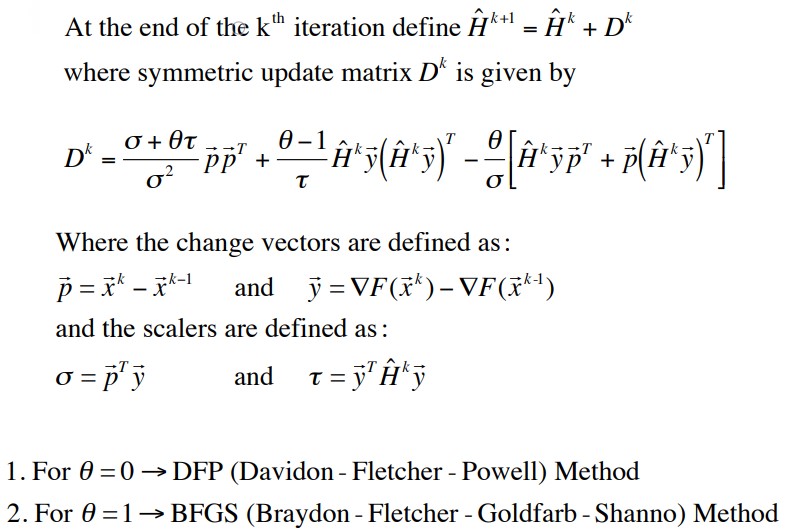
Everything on the right-hand side is known, so get the next iteration of x-vector. These steps are repeated until the stopping criteria is met (equation 4).

|𝐹(𝑥⃗⃗⃗⃗𝑘⃗⃗+⃗⃗⃗1 )−𝐹(⃗𝑥⃗⃗⃗𝑘⃗⃗) | < 𝑡𝑜𝑙 (4)

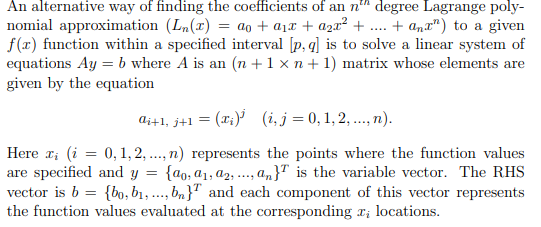
To use the BFGS algorithm all that needs to be changed is how the search direction is calculated. This is done by approximating the Hessian matrix at xk and multiplying it by the negative of the gradient of the function at xk.

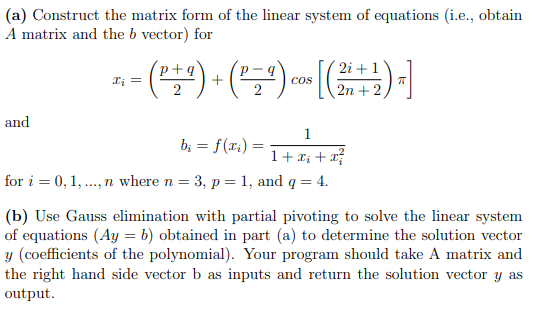
⃗𝑠⃗⃗⃗𝑘 = −𝐻𝑘∇𝐹(⃗𝑥⃗⃗⃗𝑘⃗⃗) (5)

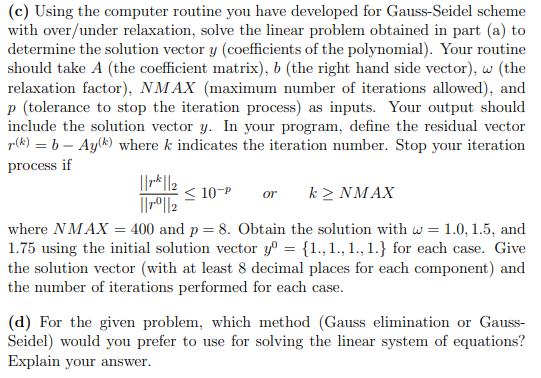
To approximate the Hessian, let the first iteration be the identity matrix at the first iteration and to get the next define D.



## Question 4







|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

## Results

1. The ‘A’ matrix was solved by hand and is;
2. Using a gauss elimination algorithm, the solution vector was calculated to be:
3. Using a gauss-seidel algorithm with different relaxation parameters



1. Gauss elimination is preferred because the matrix is not diagonally dominant which will cause issues with convergence for methods like Gauss-Seidel.

classdef rootFind

%rootFind is a class of functions that find the root of a function /

%data set

%------------------------------------------------------------------------%

methods (Static)

function x = Bisect(f,a,b,tol)

%Bisect uses the bisection algoritm using the interval

iter = 0;

while (b-a)/2 >= tol

c = (a+b)/2;

if f(c) > 0

b = c;

end

if f(c) < 0

a = c;

end

iter = iter + 1;

end

x = (a+b)/2

end

%------------------------------------------------------------------------%

function x = newRap(f,x0,eps,nmax)

%newRap is a function that utilizes the Newton-Raphson

%algorithm to find the roots of the function

%x0 is the initial guess

fp = diff(f);

x=x0;

n=0;

while eps>=1e-5&&n<=nmax

y=x-double(f(x))/double(fp(x));

eps=abs(y-x);

x=y;

n=n+1;

end

end

%------------------------------------------------------------------------%

% function x = secant\_Method(f,x0,eps,nmax)

% newRap is a function that utilizes the Newton-Raphson

% algorithm to find the roots of the function

% x0 is the initial guess

% y=x0(1);

% x=x0(2);

% n=0;

% while abs((x-y)/y)>eps

% y=x-double(f(x))/double(fp(x));

% x=y;

% n=n+1;

end

end

end

end

classdef mOpt

% mOpt is a multivariable optimizer

methods (Static)

function [Fq,q,iter,PE,q1,q2] = Steep(F,q0,tol)

syms x1 x2 x3 x4 as

% Steep is a function that utilizes the

% steepest descent method

q = q0;

q1(1) = q0(1);

q2(1) = q0(2);

q3(1) = q0(3);

q4(1) = q0(4);

i=2;

q1(2) = 10^99;

q2(2) = 10^99;

q3(2) = 10^99;

q4(2) = 10^99;

gradF(x1,x2,x3,x4) = gradient(F,[x1,x2,x3,x4]);

s = @(x1,x2,x3,x4) -gradF(x1,x2,x3,x4);

while abs(F(q1(i),q2(i),q3(i),q4(i))-F(q1(i-1),q2(i-1),q3(i-1),q4(i-1)))>tol

if q1(2) == 10^99

q1(2) = q0(1);

q2(2) = q0(2);

q3(2) = q0(3);

q4(2) = q0(4);

end

search = double(s(q1(i),q2(i),q3(i),q4(i)));

fas(as) = F(q1(i)+search(1)\*as,q2(i)+search(2)\*as,q3(i)+search(3)\*as,q4(i)+search(4)\*as);

[xlow,w2,w1,xhigh] = onedOpt.Gold(0,1.2,20,10^-8,-fas);

[alpha,y] = cubicFit(xlow,w2,w1,xhigh,fas);

q = q + alpha(1)\*search

PE(i-1,1) = F(q1(i),q2(i),q3(i),q4(i));

q1(i+1) = q(1);

q2(i+1) = q(2);

q3(i+1) = q(3);

q4(i+1) = q(4);

i = i + 1;

if i == 101

break;

end

end

Fq = F(q1(end),q2(end),q3(end),q4(end));

iter = i - 1;

q1 = transpose(q1);

q2 = transpose(q2);

end

function [Fq,q,iter,PE,q1,q2] = BFGS(F,q0,theta,tol)

%METHOD1 Summary of this method goes here

% Detailed explanation goes here

syms x1 x2 x3 x4 as

q = q0;

q1(1) = q0(1);

q2(1) = q0(2);

q3(1) = q0(3);

q4(1) = q0(4);

i=2;

q1(2) = 10^99;

q2(2) = 10^99;

q3(2) = 10^99;

q4(2) = 10^99;

gradF(x1,x2,x3,x4) = gradient(F,[x1,x2,x3,x4]);

H = eye(length(q));

D = 0;

while abs(F(q1(i),q2(i),q3(i),q4(i))-F(q1(i-1),q2(i-1),q3(i-1),q4(i-1)))>tol

if q1(2) == 10^99

q1(2) = q0(1);

q2(2) = q0(2);

q3(2) = q0(3);

q4(2) = q0(4);

end

if i >2

p = [q1(i)-q1(i-1);q2(i)-q2(i-1);q3(i)-q3(i-1);q4(i)-q4(i-1)];

y = double(gradF(q1(i),q2(i),q3(i),q4(i))-gradF(q1(i-1),q2(i-1),q3(i-1),q4(i-1)));

sigma = transpose(p)\*y;

tau = transpose(y)\*H\*y;

D = (sigma+theta\*tau)/sigma^2\*p\*transpose(p)...

+(theta-1)/tau\*H\*y\*transpose(H\*y)-...

theta/sigma\*(H\*y\*transpose(p)+p\*transpose(H\*y));

end

H = H + D;

search = double(-H\*gradF(q1(i-1),q2(i-1),q4(i-1),q3(i-1))/norm(-H\*gradF(q1(i-1),q2(i-1),q3(i-1),q4(i-1))));

fas(as) = F(q1(i)+search(1)\*as,q2(i)+search(2)\*as,q3(i)+search(3)\*as,q4(i)+search(4)\*as);

[xlow,w2,w1,xhigh] = onedOpt.Gold(0,2,20,0,-fas);

[alpha,y] = cubicFit(xlow,w2,w1,xhigh,-fas);

q = q + alpha(1)\*search;

PE(i-1,1) = F(q1(i),q2(i),q3(i),q4(i));

q1(i+1) = q(1);

q2(i+1) = q(2);

q3(i+1) = q(3);

q4(i+1) = q(4);

i = i + 1;

if i == 201

break;

end

end

Fq = F(q1(end),q2(end),q3(end),q4(end));

iter = i - 1;

q1 = transpose(q1);

q2 = transpose(q2);

end

end

end

classdef onedOpt

% onedOpt is a class of 1-d optimization functions

methods (Static)

function [xlow,x2,x1,xhigh] = Gold(xlow,xhigh,n,es,f)

% Gold is the Golden section algorithm for 1-d opt.

% if using a set convergence target set es to the according

% value and set iter to any value else set it to zero for a targeted amount of iterations

R = (sqrt(5)-1)/2;

if es>0

n = log10(es)/log10(R);

end

iter = 1;

d = R\*(xhigh-xlow);

x1 = xlow + d;

x2 = xhigh - d;

f1 = double(f(x1));

f2 = double(f(x2));

if f1>f2

xopt = x1;

fx = f1;

else

xopt = x2;

fx = f2;

end

while iter<n

d = R\*d;

if f1>f2

xlow = x2;

x2 = x1;

x1 = xlow + d;

f2 = f1;

f1 = double(f(x1));

else

xhigh = x1;

x1 = x2;

x2 = xhigh - d;

f1 = f2;

f2 = double(f(x2));

end

if f1>f2

fx = f1;

else

fx = f2;

end

iter = iter + 1

end

end

end

end

% Iterative Methods class

classdef IM

methods (Static)

%=========================================================================%

% Gauss-Seidel Method

%=========================================================================%

function [x,w] = gauSei(A,b,n,x,imax,es,lambda)

for i = 1:n

dum = A(i,i);

for j = 1:n

A(i,j) = A(i,j)/dum;

end

b(i) = b(i)/dum;

end

for i = 1:n

sum = b(i);

for j = 1:n

if i~= j

sum = sum - A(i,j)\*x(j);

end

x(i) = sum;

end

end

iter = 1;

sen = 0;

L2norm\_0 = norm(b-A\*x);

while sen == 0

sen = 1;

for i = 1:n

old = x(i);

sum = b(i);

for j = 1:n

if i~= j

sum = sum - A(i,j)\*x(j);

end

end

x(i) = lambda\*sum + (1-lambda)\*old;

L2norm = norm(b-A\*x);

if sen == 1 && x(i) ~= 0

ea = abs(L2norm/L2norm\_0)/1;

if ea > es

sen = 0;

end

end

end

iter = iter + 1;

if iter >= imax

break

end

end

w = [lambda iter];

end

%=========================================================================%

% Newton-Raphson Method

%=========================================================================%

function [q,t] = newRap(f,q,p,kmax)

% f is the 'A' matrix

% q is the 'b' vector

% p is the precision goal

% kmax is the maximum allowable iterations

syms x1 x2 x3 x4

fp = jacobian(f,[x1 x2 x3 x4]);

b = transpose(double(f(q(1),q(2),q(3),q(4))));

b\_0 = b ;

k = 0;

while (norm(b)/norm(b\_0)) > 10^p && k<kmax;

L2norm = (norm(b)/norm(b\_0));

l = norm(b);

A = double(fp(q(1),q(2),q(3),q(4)));

b = transpose(double(f(q(1),q(2),q(3),q(4))));

del = linsolve(A,-b); % apparently linsolve uses LU Factorization

q = q+del;

t(k+1,1) = k;

t(k+1,2) = l;

t(k+1,3) = L2norm;

k = k + 1;

end

end

end

end

function [x,y] = cubicFit(xlow,x2,x1,xhigh,f)

%cubicFit fits a cubic function into to the specified points

% takes values from Gold

q1 = x1^3\*(x2-xlow)-x2^3\*(x1-xlow)+xlow^3\*(x1-x2);

q2 = xhigh^3\*(x2-xlow)-x2^3\*(xhigh-xlow)+xlow^3\*(xhigh-x2);

q3 = (x1-x2)\*(x2-xlow)\*(x1-xlow);

q4 = (xhigh-x2)\*(x2-xlow)\*(xhigh-xlow);

q5 = double(f(x1))\*(x2-x1)-double(f(x2))\*(x1-xlow)+double(f(x1-x2));

q6 = double(f(xhigh))\*(x2-x1)-double(f(x2))\*(xhigh-xlow)+double(f(xhigh-x2));

a3 = (q3\*q6-q4\*q5)/(q2\*q3-q1\*q4);

a2 = (q5-a3\*q1)/q3;

a1 = (double(f(x2)-f(xlow)))/(x2-xlow)-a3\*(x2^3-xlow^3)/(x2-xlow)-...

a2\*(xlow+x2);

del = a2^2-3\*a1\*a3;

x(1) = double(-a2+sqrt(del))/3/a3;

x(2) = double(-a2-sqrt(del))/3/a3;

y(1) = double(f(x(1)));

y(2) = double(f(x(2)));

end

function [x] = gauss(a,b)

% gauss elimination

n = length(a);

k = 1 ;

p = k ;

big = abs(a(k,k));

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% pivoting portion

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for ii=k+1:n

dummy = abs(a(ii,k));

if dummy > big

big = dummy;

p = ii ;

end

end

if p ~= k

for jj = k:n

dummy = a(p,jj);

a(p,jj) = a(k,jj);

a(k,jj) = dummy;

end

dummy = b(p);

b(p)=b(k);

b(k) = dummy;

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% elimination step

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for k=1:(n-1)

for i=k+1:n

factor = a(i,k)/a(k,k);

for j=k+1:n

a(i,j) = a(i,j) - factor\*a(k,j);

end

b(i) = b(i) - factor\*b(k);

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% back substitution

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

x(n,1) = b(n)/a(n,n);

for i = n-1:-1:1

sum = b(i);

for j = i + 1:n

sum = sum - a(i,j)\*x(j,1);

end

x(i,1) = sum/a(i,i);

end

end

clc

clear all

close all

format longg

syms x1 x2 x3 x4 as

%=========================================================================%

% q2

%=========================================================================%

F = @(x1,x2,x3,x4) cos(x1^2-x2)+sin(x1^2+x2^2)

gradF(x1,x2,x3,x4) = gradient(F,[x1,x2,x3,x4])

s = @(x1,x2,x3,x4) -gradF(x1,x2,x3,x4)

search = double(s(.1,-.1,0,0))

fas(as) = F(.1+search(1)\*as,-.1+search(2)\*as,search(3)\*as,search(4)\*as)

[xlow,w2,w1,xhigh] = onedOpt.Gold(0,1.2,20,10^-8,-fas)

[x,y] = cubicFit(xlow,w2,w1,xhigh,fas)

diff(fas)

=========================================================================%

q3

=========================================================================%

theta = 1

q0 = [1;1;0;0]

F = @(x1,x2,x3,x4) 1/30\*(8\*x1^2-x1\*x2+.5\*x2^2)+x1\*exp(-x1^2-x2^2)

[Fq,q,iter,PE,q1,q2] = mOpt.BFGS(F,q0,theta,10^-6);

[Fs,qs,iters,PEs,q1s,q2s] = mOpt.Steep(F,q0,10^-6);

hold on

plot(1:length(PEs),PEs)

plot(1:length(PE),PE)

xlabel('Iterations')

legend('Steepest Descent','BFGS')

ylabel('F({x})')

title('Convergence history of the objective function')

grid on

%=========================================================================%

% q4

%=========================================================================%

x0 = [1;1;1;1];

imax = 400;

es = 10^-8;

lambda = 1.75;

n=3;

p=1;

q=4;

A = [1 1.14 1.219 1.317;1 1.925 2.228 2.415;...

1 3.074 2.28 2.584;1 3.886 1.2196 3.6829];

b = zeros(n+1,1);

x = b;

i=0;

while i<n+1

x(i+1) = (p+q)/2+(p-q)/2\*cos(((2\*(i)+1)/(2\*n+2))\*pi);

b(i+1) = 1/(1+x(i+1)+x(i+1)^2);

i=i+1;

end

y = gauss(A,b)

[w,z] = IM.gauSei(A,b,n,x0,imax,es,lambda)

# Matthew Pahayo

# 2/11/2021

# computational methods

# hw 1

# question 2

# secant method

# q2.py

import numpy as np

def f(x):

    gam = 1.4

    P0r = 1

    return np.cos(((3249138182000457\*x)/36028797018963968 - 1/10)\*\*2 + ((1603320569089981\*x)/9007199254740992 - 1/10)\*\*2)\*((51687088482005563432241941494625\*x)/649037107316853453566312041152512 - 9662420458360381/180143985094819840) - np.sin(((1603320569089981\*x)/9007199254740992 - 1/10)\*\*2 - (3249138182000457\*x)/36028797018963968 + 1/10)\*((2570636847267020537246474580361\*x)/40564819207303340847894502572032 - 22658973186362209/180143985094819840)

# define tol, guess a\_0 and a\_1

# a\_0 != 0, because of stopping criteria

tol = 10\*\*-8

a\_0 = .001

a\_1 = 1.2

# declare lists

al = [a\_0, a\_1]

fl = [f(a\_0), f(a\_1)]

dal = []

# secant method

# evaluates function at a\_n

n = 0

while (abs(al[n+1]-al[n])/abs(al[n])) > tol:

    if (f(al[n+1])-f(al[n]))\*((al[n+1])-al[n]) ==0:

        break

    a\_np1 = al[n+1] - f(al[n+1])/(f(al[n+1])-f(al[n]))\*((al[n+1])-al[n])

    al.append(a\_np1)

    fl.append(f(al[n]))

    n += 1

# defines dal list

for k in range(len(al)-1):

    dal.append(al[k+1]-al[k])

i=0

data\_fl = open("fl.txt", "a")

data\_al = open("al.txt", "a")

for i in range(len(al)):

    data\_al.write(str(al[i]) + "\n")

    data\_fl.write(str(fl[i]) + "\n")

data\_al.close()

data\_fl.close()

i = 0

data\_dal = open("dal.txt", "a")

for i in range(len(dal)):

    data\_dal.write(str(dal[i]) + "\n")

data\_dal.close()

print("a = " + str(a\_np1))

print("total iterations = " + str(n))

print("final delta a = " + str(dal[n-1]))