Test 1

AERO 5830 - S. HOSDER

MATT PAHAYO

Question 1

The following function $f(M)$ describes the total pressure ratio (P_{0_r}) across a normal shock wave for a calorically perfect gas where M is the Mach number upstream of the shock:

$$P_{0_r} = f(M) = \left[1 + \frac{2\gamma}{\gamma+1}(M^2-1)\right]^{\frac{-1}{\gamma-1}} \times \left[\frac{2+(\gamma-1)M^2}{(\gamma+1)M^2}\right]^{\frac{-\gamma}{\gamma-1}} \quad (1)$$

Here γ is the specific heats ratio and constant. Using the Secant Method and for $\gamma = 1.4$, find the upstream Mach number (M) for **(i)** a total pressure ratio of $P_{0_r} = 0.1386$ and **(ii)** a total pressure ratio of $P_{0_r} = 1.0$. For both cases, use $M_0 = 2.8$ and $M_1 = 2.2$ as the initial guesses to start the Secant method and approximate the upstream Mach number with a convergence criterion of $(\epsilon_n)_r = \frac{|M_{n+1}-M_n|}{|M_n|} < 10^{-8}$ where n is the iteration number.

(a) For each pressure ratio, tabulate the values of M_n, $\Delta_n M = M_{n+1} - M_n$, and M_{n+1} at each iteration.

(b) Comment on the converge of the Secant Method for each case. If there is a difference in convergence, what can be the possible reason for this?

(c) Now assume that you use Newton's method instead of Secant method for both pressure ratios. Would you expect to see a similar difference in the convergence between two pressure ratio cases? Why or why not? Explain briefly (in words) how you can improve the convergence for the case which has a slower convergence rate.

Results

(a)

P_{0r} = 0.1386	M_n	M_{n+1}	$\Delta_n M$
1	2.8	2.2	-0.5999999999999996
2	2.2	3.4306480771137364	1.2306480771137362
3	3.4306480771137364	3.698619042421396	0.2679709653076596
4	3.698619042421396	3.932629587760294	0.2340105453388981
5	3.932629587760294	3.9923757655657788	0.059746177805484635
6	3.9923757655657788	4.001056114847878	0.008680349282099264
7	4.001056114847878	4.001332702778001	0.0002765879301227514
8	4.001332702778001	4.001333837214977	1.1344369763577333e-06
9	4.001333837214977	4.001333837358875	1.438982266677158e-10
10	4.001333837358875		

P_{0r} = 1	M_n	M_{n+1}	$\Delta_n M$
1	2.8	2.2	-0.5999999999999996
2	2.2	1.2651697984134973	-0.9348302015865029
3	1.2651697984134973	1.225692738526986	-0.03947705988651129
4	1.225692738526986	1.1497675333434785	-0.07592520518350754
5	1.1497675333434785	1.1119446188253057	-0.03782291451817277
6	1.1119446188253057	1.0814331519146798	-0.030511466910625895
7	1.0814331519146798	1.0605004505348425	-0.02093270137983727
8	1.0605004505348425	1.0449420189061367	-0.015558431628705849
9	1.0449420189061367	1.0335758569334714	-0.011366161972665267
10	1.0335758569334714	1.0251343372215227	-0.00844151971194873
11	1.0251343372215227	1.0188590211772832	-0.006275316044239476
12	1.0188590211772832	1.0141704780093277	-0.004688543167955528
13	1.0141704780093277	1.0106601460473243	-0.0035103319620033435
14	1.0106601460473243	1.0080261295491044	-0.002634016498219266
15	1.0080261295491044	1.0060468737630668	-0.001979255786037637
16	1.0060468737630668	1.0045578972039426	-0.0014889765591241666
17	1.0045578972039426	1.0034368185630342	-0.0011210786409083795
18	1.0034368185630342	1.0025921946997798	-0.0008446238632544567
19	1.0025921946997798	1.0019555478918263	-0.0006366468079535004
20	1.0019555478918263	1.0014754923674232	-0.0004800555244031113
21	1.0014754923674232	1.001113413640559	-0.00036207872686411235
22	1.001113413640559	1.0008402616705592	-0.0002731519699998852
23	1.0008402616705592	1.000634164149299	-0.00020609752126010683
24	1.000634164149299	1.0004786421127247	-0.00015552203657431818
25	1.0004786421127247	1.0003612737080978	-0.00011736840462694964
26	1.0003612737080978	1.0002726935370465	-8.858017105128901e-05
27	1.0002726935370465	1.0002058380776557	-6.685545939077997e-05
28	1.0002058380776557	1.0001553738493856	-5.0464228270152844e-05
29	1.0001553738493856	1.000117278364225	-3.8095485160649645e-05
30	1.000117278364225	1.0000885331033322	-2.8745260892693025e-05
31	1.0000885331033322	1.0000668288104353	-2.1704292896940203e-05
32	1.0000668288104353	1.0000504589420594	-1.6369868375942787e-05
33	1.0000504589420594	1.0000381197210328	-1.2339221026502756e-05
34	1.0000381197210328	1.0000287529328036	-9.366788229270284e-06
35	1.0000287529328036	1.000021676935174	-7.0759976296130844e-06
36	1.000021676935174	1.000016302759759	-5.374175414907256e-06
37	1.000016302759759	1.0000121930962065	-4.109663552576137e-06
38	1.0000121930962065	1.0000086705274471	-3.5225687593509747e-06
39	1.0000086705274471	1.0000069092430675	-1.7612843796754873e-06
40	1.0000069092430675	1.0000016253899284	-5.283853139026462e-06
41	1.0000016253899284	1.0000005686193005	-1.0567706278941102e-06
42	1.0000005686193005		

(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.

<u>Methodology</u>

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

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})} = \frac{x_{n-2}f(x_{n-1}) - x_{n-1}f(x_{n-2})}{f(x_{n-1}) - f(x_{n-2})}.$$

Solve the equation above until convergence criteria is met.

Question 2

An objective function of two design variables $\vec{x} = \{x_1, x_2\}$ is defined as:

$$f(x_1, x_2) = Cos(x_1^2 - x_2) + Sin(x_1^2 + x_2^2) \tag{2}$$

By starting with the initial design point, $\vec{x}^0 = \{0.1, -0.1\}$,

(a) Formulate the problem as a one-dimensional minimization problem to calculate the next design point $\vec{x}^1$ along the search direction, $s^1 = -\nabla f(\vec{x}^0)$. (Do not scale the search direction for this question). Show the steps in your formulation clearly and give the expression for the one-dimensional objective function ($f(\alpha)$) you have obtained.

(b) Perform one-dimensional optimization to determine the minimum of the objective function ($f(\alpha)$) obtained in part (a) using the Golden-Section search algorithm. Use $0.0 \leq \alpha \leq 1.2$ as your initial interval. First, calculate the number of iterations required to reduce this interval by 8 orders of magnitude and then perform the Golden-Section search to determine the optimum value of α (i.e., α^*), corresponding value of the objective function $f(\alpha^*)$, and the design point $\vec{x}^1$.

(c) Describe and show clearly how you can implement the Secant Method to perform one-dimensional optimization for the problem described in part (b) (i.e., finding the minimum of the objective function $f(\alpha)$). With the procedure you have described, calculate the optimum value of α (i.e., α^*), corresponding value of the objective function $f(\alpha^*)$, and the design point $\vec{x}^1$. For the Secant Method, use the end points of the initial interval given in part (b) as the starting values and use a convergence criterion of $(\epsilon_n)_r = \frac{|\alpha_{n+1} - \alpha_n|}{|\alpha_n|} < 10^{-8}$ (n is the iteration number).

Results

2

a) $\vec{s}^1 = -\nabla f(x^0)$ $\qquad \bar{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$$= \begin{bmatrix} -\dfrac{\partial f(\bar{x}^0)}{\partial x_1} \\ -\dfrac{\partial f(\bar{x}^0)}{\partial x_2} \end{bmatrix}$$

$$\nabla f(\bar{x}) = \begin{bmatrix} -2x_1 \sin(x_1^2 + x_2) + 2x_1 \cos(x_1^2 + x_2^2) \\ + \sin(x_1^2 - x_2) + 2x_2 \cos(x_1^2 + x_2^2) \end{bmatrix}$$

$$-\nabla f(x^0) = \begin{bmatrix} -.178 \\ .0901 \end{bmatrix}$$

$$f(\alpha) = f(x^0 + \alpha s^1)$$

$$f\left(\begin{bmatrix} -.178 \\ .9018 \end{bmatrix} \alpha + \begin{bmatrix} .1 \\ -.1 \end{bmatrix}\right) =$$

b) iter = 39

(a)

First compute the gradient of the function:

$$\begin{bmatrix} 2 * x1 * cos(x1\text{^}2 + x2\text{^}2) + 2 * x1 * sin(-x1\text{^}2 + x2) \\ 2 * x2 * cos(x1\text{^}2 + x2\text{^}2) - sin(-x1\text{^}2 + x2) \end{bmatrix}$$

Then get the search direction at $\overrightarrow{s^1}$

$$\overrightarrow{s^1} = \begin{bmatrix} -0.178004341165881 \\ 0.0901817004961407 \end{bmatrix}$$

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

$$f(\alpha) = F(\vec{x}^0 + \alpha \overrightarrow{s^1})$$

$$\begin{aligned} f(\alpha) = cos(((1603320569089981 * as)/9007199254740992 - 1/10)^2 \\ - (3249138182000457 * as)/36028797018963968 + 1/10) \\ + sin(((3249138182000457 * as)/36028797018963968 - 1/10)^2 \\ + ((1603320569089981 * as)/9007199254740992 - 1/10)^2) \end{aligned}$$

(b) **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.**

(c) **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.**

1. **Find derivative of the objective function and set it equal to zero.**

$$\frac{df(\alpha)}{d\alpha} = 0$$

2. **Use a root finding technique such as secant method on the new objective function to find the roots/minimum.**

$$\begin{aligned} \frac{df(\alpha)}{d\alpha} = 0 = cos(((3249138182000457 * as)/36028797018963968 - 1/10)^2 \\ + ((1603320569089981 * as)/9007199254740992 - 1/10)^2) \\ * ((51687088482005563432241941494625 \\ * as)/649037107316853453566312041152512 \\ - 9662420458360381/180143985094819840) \\ - sin(((1603320569089981 * as)/9007199254740992 \\ - 1/10)^2 - (3249138182000457 * as)/36028797018963968 \\ + 1/10) * ((2570636847267020537246474580361 \\ * as)/40564819207303340847894502572032 \\ - 22658973186362209/180143985094819840) \end{aligned}$$

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

The objective function and it's derivative are found with MATLAB.

Question 3

An objective function of two design variables $\vec{x} = \{x_1, x_2\}$ is defined as:

$$f(x_1, x_2) = \frac{1}{30}(8x_1^2 - x_1 x_2 + 0.5x_2^2) + x_1 e^{-(x_1^2+x_2^2)} \tag{3}$$

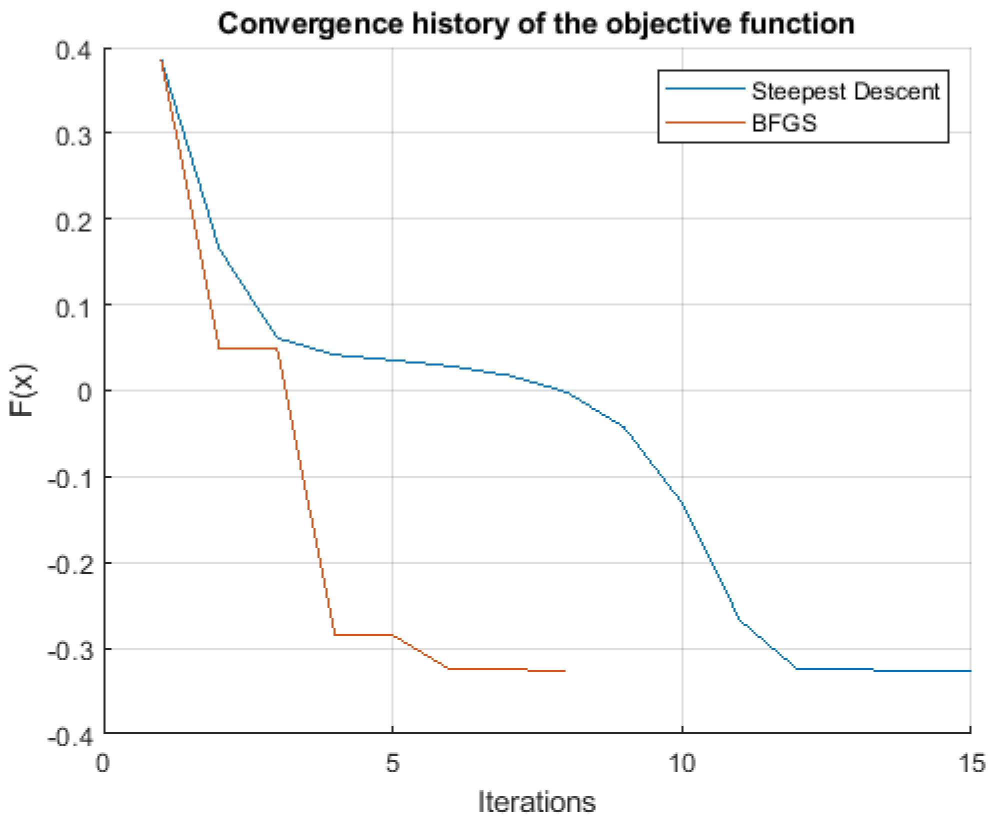
Calculate the minimum objective function value and the corresponding design variables by using (i) **Steepest Descent** and (ii) **BFGS Variable Metric (Quasi-Newton)** Methods. For each method, use the initial design point $\vec{x}^0 = \{1.0, 1.0\}$ and limit the maximum number of iterations to 100.

Use the convergence criterion $\left|f(\vec{x}^k) - f(\vec{x}^{k-1})\right| \leq 10^{-10}$ and make sure to satisfy this at 4 successive iterations. **For each method:**
(a) Tabulate the values of the design variables and the objective function as a function of the iteration number k.
(b) Plot the convergence history of the objective function ($f(\vec{x}^k)$ vs. iteration number k).
(c) Comment on the convergence of the two methods.

(a)

Steepest	x1	x2	F({xk})		BFGS	x1	x2	F({xk})
0	1	1	0.385335		0	1	1	0.385335
1	0.575834	1.314835	0.165363		1	0.06135	1.696709	0.048949
2	0.219674	1.510587	0.061209		2	0.05791	1.699027	0.048944
3	0.039731	1.535634	0.041442		3	-0.69172	0.170816	-0.28433
4	-0.03488	1.49104	0.03534		4	-0.70534	0.134375	-0.28507
5	-0.0809	1.418793	0.028384		5	-0.56692	-0.0577	-0.32506
6	-0.12806	1.325208	0.017543		6	-0.56316	-0.06138	-0.32508
7	-0.18837	1.201789	-0.00181		7	-0.54653	-0.02598	-0.32594
8	-0.27358	1.022476	-0.04253		8	-0.54623	-0.02508	-0.32595
9	-0.39042	0.751655	-0.13068					
10	-0.51758	0.362264	-0.26737					
11	-0.54558	0.025162	-0.32502					
12	-0.54622	0.022201	-0.32514					
13	-0.55496	-0.02361	-0.32593					
14	-0.54961	-0.02299	-0.32596					
15	-0.54962	-0.02295	-0.32596					

(b)



(c) 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. In this problem the search direction was not scaled with the L2 norm of the function

$$\vec{s}^{k} = -\nabla \mathrm{F}(\overrightarrow{x^{k}})/||\nabla \mathrm{F}(\overrightarrow{x^{k}})|| \qquad (1)$$

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.

$$\frac{d}{d\alpha} F(\vec{x}^{k} + \alpha^{k} s^{k}) = 0 \qquad (2)$$

$$\vec{x}^{k+1} = \vec{x}^{k} + \alpha^{*k} s^{k}. \qquad (3)$$

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).

$$\left|\mathrm{F}(\overrightarrow{x^{k+1}}) - \mathrm{F}(\overrightarrow{x^{k}})\right| < tol \qquad (4)$$

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 x^k and multiplying it by the negative of the gradient of the function at x^k.

$$s^k = -H^k \nabla F(\vec{x}^k) \qquad (5)$$

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

At the end of the k$^{\text{th}}$ iteration define $\hat{H}^{k+1} = \hat{H}^k + D^k$

where symmetric update matrix D^k is given by

$$D^k = \frac{\sigma + \theta\tau}{\sigma^2}\vec{p}\vec{p}^T + \frac{\theta - 1}{\tau}\hat{H}^k\vec{y}\left(\hat{H}^k\vec{y}\right)^T - \frac{\theta}{\sigma}\left[\hat{H}^k\vec{y}\vec{p}^T + \vec{p}\left(\hat{H}^k\vec{y}\right)^T\right]$$

Where the change vectors are defined as:

$\vec{p} = \vec{x}^k - \vec{x}^{k-1}$ and $\vec{y} = \nabla F(\vec{x}^k) - \nabla F(\vec{x}^{k-1})$

and the scalers are defined as:

$\sigma = \vec{p}^T\vec{y}$ and $\tau = \vec{y}^T\hat{H}^k\vec{y}$

1. For $\theta = 0 \rightarrow$ DFP (Davidon - Fletcher - Powell) Method
2. For $\theta = 1 \rightarrow$ BFGS (Braydon - Fletcher - Goldfarb - Shanno) Method

Question 4

An alternative way of finding the coefficients of an n^{th} degree Lagrange polynomial approximation $(L_n(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n)$ to a given $f(x)$ function within a specified interval $[p, q]$ is to solve a linear system of equations $Ay = b$ where A is an $(n+1 \times n+1)$ matrix whose elements are given by the equation

$$a_{i+1,\ j+1} = (x_i)^j \quad (i, j = 0, 1, 2, ..., n).$$

Here x_i $(i = 0, 1, 2, ..., n)$ represents the points where the function values are specified and $y = \{a_0, a_1, a_2, ..., a_n\}^T$ is the variable vector. The RHS vector is $b = \{b_0, b_1, ..., b_n\}^T$ and each component of this vector represents the function values evaluated at the corresponding x_i locations.

(a) Construct the matrix form of the linear system of equations (i.e., obtain A matrix and the b vector) for

$$x_i = \left(\frac{p+q}{2}\right) + \left(\frac{p-q}{2}\right) cos\left[\left(\frac{2i+1}{2n+2}\right)\pi\right]$$

and

$$b_i = f(x_i) = \frac{1}{1 + x_i + x_i^2}$$

for $i = 0, 1, ..., n$ where $n = 3$, $p = 1$, and $q = 4$.

(b) Use Gauss elimination with partial pivoting to solve the linear system of equations $(Ay = b)$ obtained in part (a) to determine the solution vector y (coefficients of the polynomial). Your program should take A matrix and the right hand side vector b as inputs and return the solution vector y as output.

(c) Using the computer routine you have developed for Gauss-Seidel scheme with over/under relaxation, solve the linear problem obtained in part (a) to determine the solution vector y (coefficients of the polynomial). Your routine should take A (the coefficient matrix), b (the right hand side vector), ω (the relaxation factor), $NMAX$ (maximum number of iterations allowed), and p (tolerance to stop the iteration process) as inputs. Your output should include the solution vector y. In your program, define the residual vector $r^{(k)} = b - Ay^{(k)}$ where k indicates the iteration number. Stop your iteration process if

$$\frac{||r^k||_2}{||r^0||_2} \leq 10^{-p} \qquad or \qquad k \geq NMAX$$

where $NMAX = 400$ and $p = 8$. Obtain the solution with $\omega = 1.0, 1.5$, and 1.75 using the initial solution vector $y^0 = \{1., 1., 1., 1.\}$ for each case. Give the solution vector (with at least 8 decimal places for each component) and the number of iterations performed for each case.

(d) For the given problem, which method (Gauss elimination or Gauss-Seidel) would you prefer to use for solving the linear system of equations? Explain your answer.

Results

(a) The 'A' matrix was solved by hand and is;

$$A = \begin{bmatrix} 1 & 1.114180 & 1.24139863500022 & 1.38314240165432 \\ 1 & 1.925 & 3.70937912842696 & 7.14417091585262 \\ 1 & 3.074 & 9.44963061390331 & 29.0484021516244 \\ 1 & 3.866 & 15.0995916226695 & 58.6742845308687 \end{bmatrix}$$

$$\vec{b} = \begin{bmatrix} 0.298011133041045 \\ 0.150707860203441 \\ 0.0739445026970109 \\ 0.0500364993209819 \end{bmatrix}$$

(b) Using a gauss elimination algorithm, the solution vector was calculated to be:

$$\vec{y} = \begin{bmatrix} 0.807156858499322 \\ -0.620722356642715 \\ 0.180093126458478 \\ -0.018139564092641 \end{bmatrix}$$

(c) Using a gauss-seidel algorithm with different relaxation parameters

ω	1	1.5	1.75
x1	8.07112849E-01	8.07156342E-01	8.07156138E-01
x2	-6.20660277E-01	-6.20721756E-01	-6.20721114E-01
x3	1.80066750E-01	1.80092930E-01	1.80092525E-01
x4	-1.81361377E-02	-1.81395457E-02	-1.81394811E-02
iterations	400	108	389

(d) Gauss elimination is preferred because the matrix is small so it will take less time and Gauss elimination is a direct method, while Gauss-Seidel is an iterative method.

```matlab
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
```

```matlab
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;

n=3;
p=1;
q=4;
A = [1 1.14 1.24139863500022 1.38314240165432;1 1.925 3.70937912842696
7.14417091585262;...
     1 3.074 9.44963061390331 29.0484021516244;1 3.886 15.0995916226695
58.6742845308687];
```

```
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+1,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)*((5168708848200556343224194149462 5*x)/64903710731685345356631204115 2512 - 9662420458360381/18014398509481984 0) - np.sin(((1603320569089981*x)/9007199254740992 - 1/10)**2 - (3249138182000457*x)/36028797018963968 + 1/10)*((2570636847267020537246474580361*x)/4056481920730334084789450257203 2 - 2265897318636220 9/18014398509481984 0)
```

```
# 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()
```

```python
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]))
```