CSC 2305 H1S
Numerical methods for Optimization
Assignment 1
Fri 22, Jan. 2016.
Sooa Lim, cdf c5limsoo, UID 994500731

Question 1

Given the order the convergence "*p*" ranges between 1 < p < 2 from the experimental results (*Table 1*), we can deduce that the rate of convergence of iteration is **superlinear (≈1.618)**.

Table(1)

Iter (=n)	x_n (=x)	x_n-x_{n-1} (=delta)	x_n-x^* (=e)	Order of convergence (=r_est)	Asymptotic error constant (=c_est)
1	32.000				
2	16.000				
3	10.708	5.292	9.294		
4	6.490	4.218	5.076	0.000	5.076
5	4.157	2.333	2.743	1.017	0.525
6	2.722	1.435	1.308	1.204	0.388
7	1.936	0.786	0.521	1.241	0.374
8	1.561	0.375	0.146	1.382	0.360
9	1.436	0.125	0.022	1.498	0.388
10	1.415	0.021	0.001	1.587	0.460
11	1.414	0.001	0.000	1.614	0.511
12	1.414	0.000	0.000	1.618	0.526

Rate of Convergence (p) = 1.618 → superlinear
Error constant (C) = 0.526

Matlab Code for Q1

```
clear

x(1) = 32;
x(2) = 16;
n=3
tol = 1e-5;
delta = ones(1,2)*1e-4

while delta(n-1)>tol
    x(n) = x(n-1)-((x(n-1)^2-2)/(x(n-1)+x(n-2)));
    delta(n) = abs(x(n)-x(n-1));
    e(n) = abs(x(n)-sqrt(2))
    r_est(n) = log(e(n)/e(n-1))/log(e(n-1)/e(n-2));
    c_est(n) = e(n)/(e(n-1)^r_est(n));
    n = n+1;
end
n = n-1

T = table(n, x, delta, e, r_est, c_est)
```

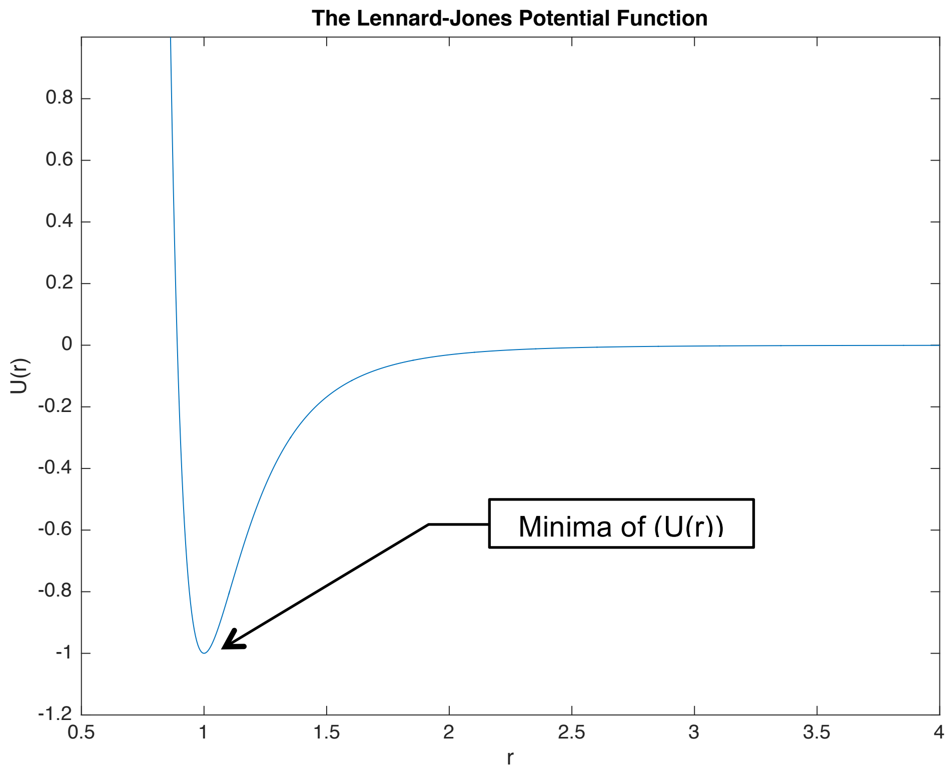
Question 2

Graph of (r, U(r)) to get a general idea of the curve and interval. Note r > 0

$$U(r) = r^{-12} - 2r^{-6}$$

As r → +∞ then U(r) → 0
And as r → 0 then U(r) → +∞

As the minima occurs between the interval 0.5 and 2, I will be setting my initial interval of interest to [0.5, 2] for my three approximation methods.



Matlab Code for graph U(r)

```
r = [0.5:0.005:4];
U = LJPotential(r);
plot(r,U)
axis([min(r), max(r), -1.2, 1])
xlabel('r')
ylabel('U(r)')
title('The Lennard-Jones Potential Function')
```

1) Golden Section Search

Output for different initial conditions

Tried testing within range of x>0 per original condition

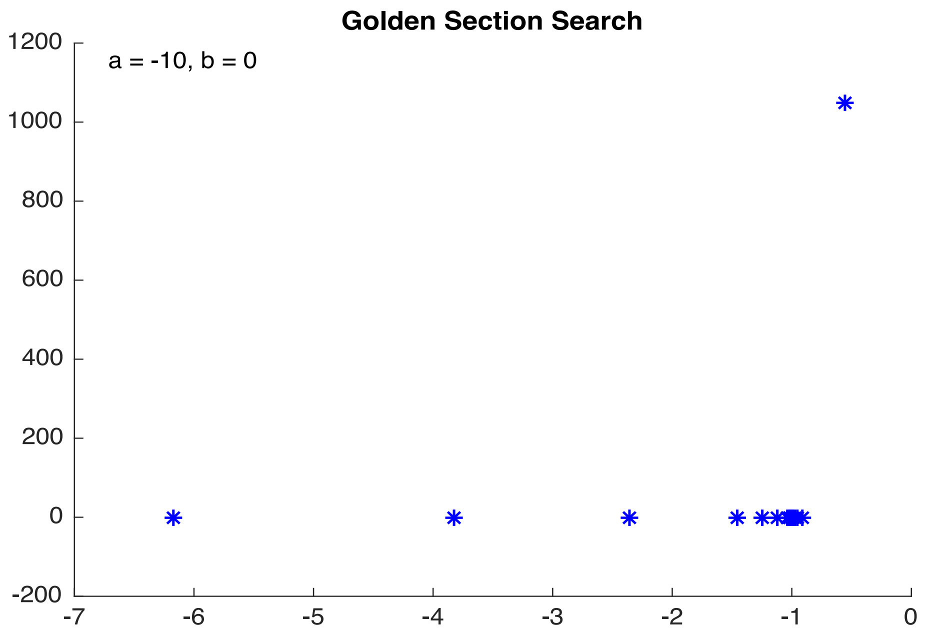
Table(2)

No.	a (start)	b (end)	Tolerance	x_{min}	$f(x)_{min}$	Iter
1	-10	0	1e-6	-1.000001	-1.000000	68
2	-1	1	1e-6	1.000000	-1.000000	62
3	0.5	1.5	1e-6	1.000001	-1.000000	58
4	1	10	1e-6	1.000000	-1.000000	68
5	0.5	1.5	1e-12	1.000000	-1.000000	116
6	0.5	1.5	1e-9	1.000000	-1.000000	88
7	0.5	1.5	1e-3	1.000033	-1.000000	30
8	0.5	1.5	0.1	1.006578	-0.998512	10
9	0.5	1.5	0.5	1.027864	-0.976890	4
10	0.5	1.5	0.8	1.118034	-0.761856	2

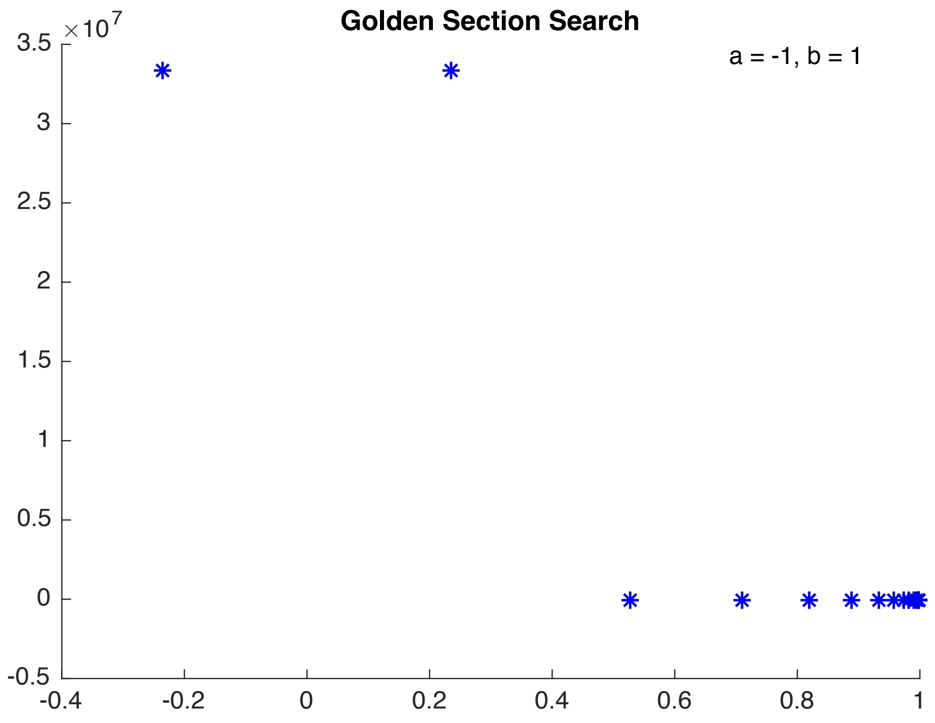
Observation: Lowering the tolerance level increases the rate of convergence and thus reduced the number of steps of iteration to reach convergence. On the other hand, after the tolerance increased above 1e-3 the accuracy of locating the minimal point deteriorated.

Even when the interval included a region where the function was undefined (x=0) the algorithm managed to converge to a minima and yield the correct answer (1, -1)

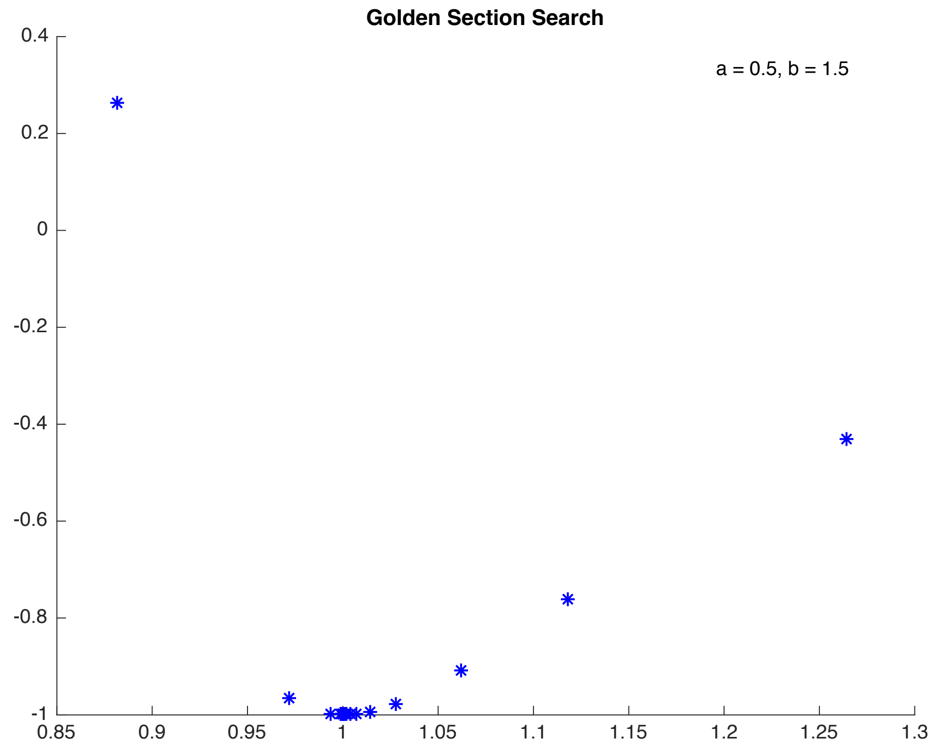
1. GSS(@LJPotential,-10, 0, 1e-6) -> interval included for experimental purpose



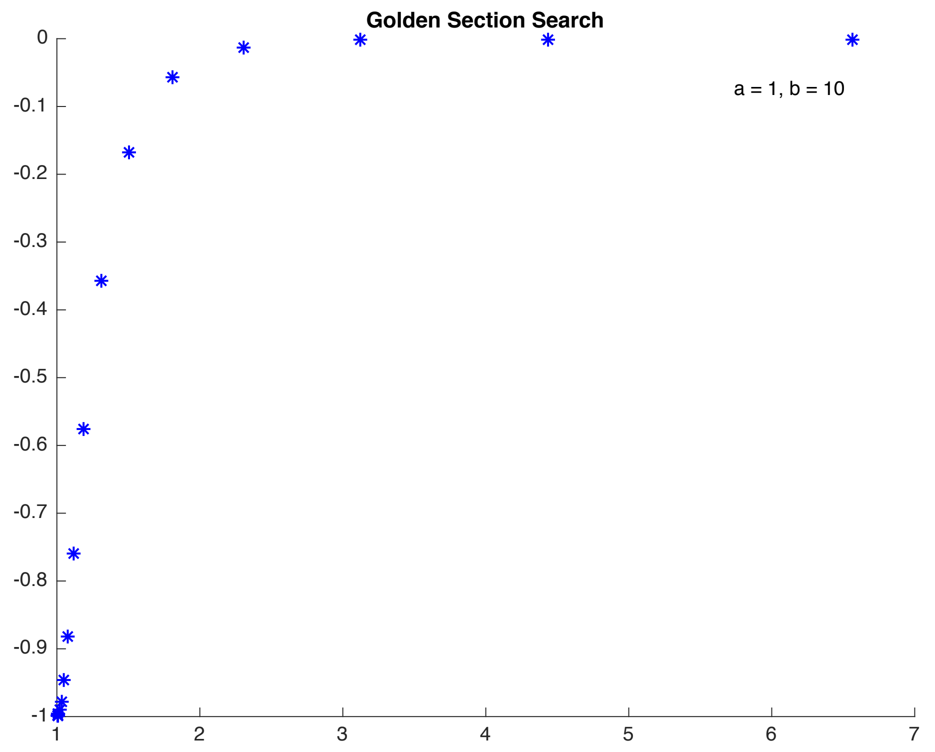
```
2. GSS(@LJPotential,-1, 1, 1e-6)
```



```
1. GSS(@LJPotential, 0.5, 1.5, 1e-6)
```

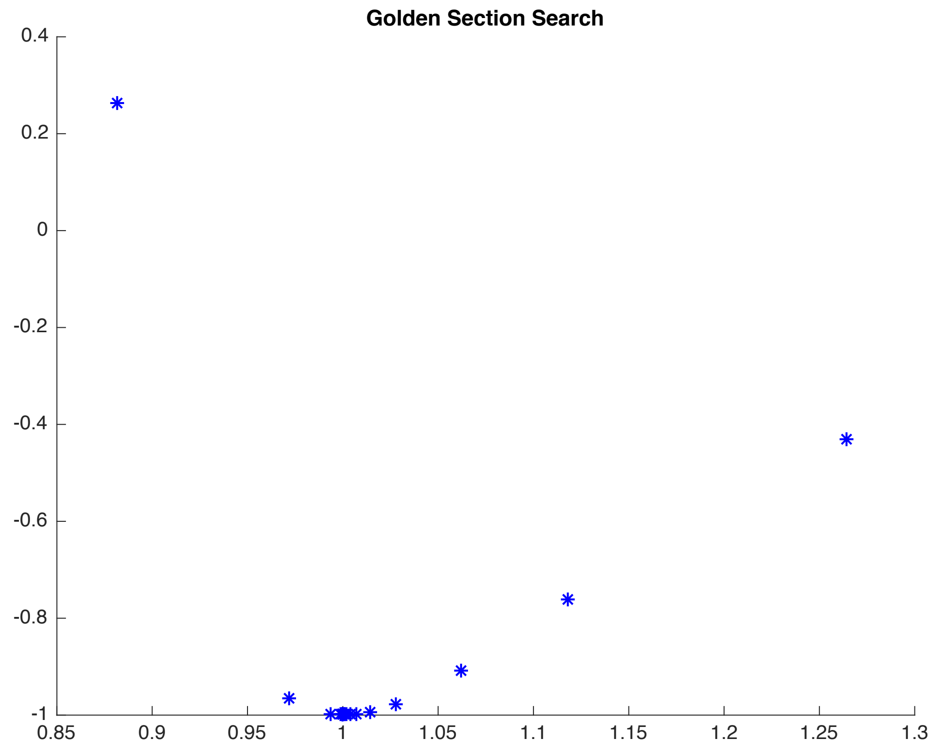


```
2. GSS(@LJPotential, 1, 10, 1e-6)
```

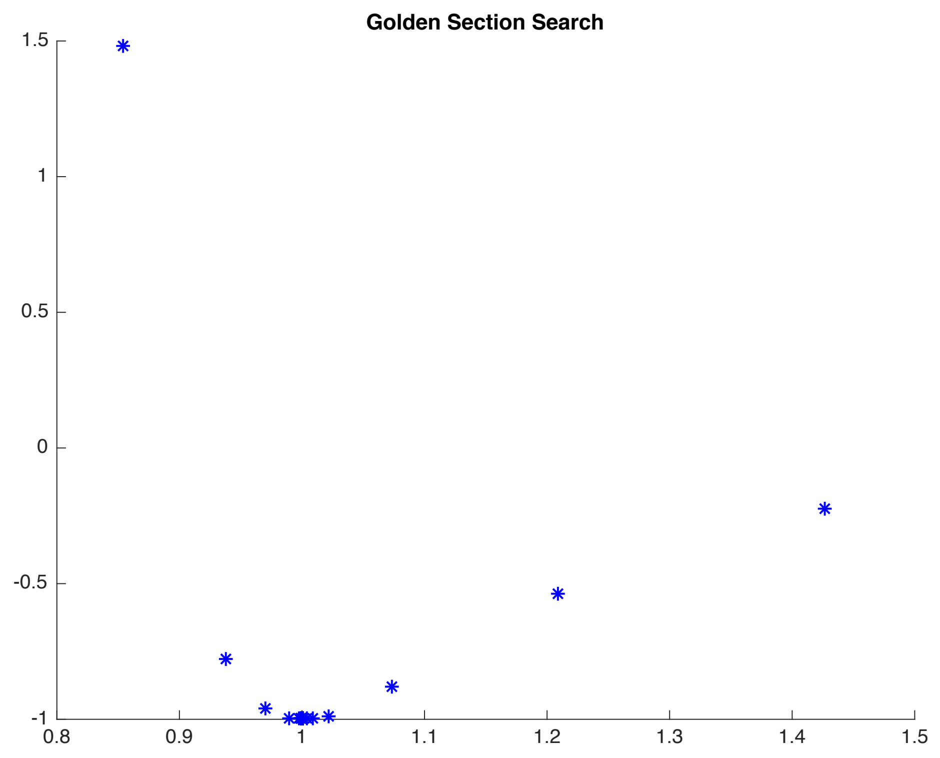


Vary Tolerance (3rd parameter of function input)

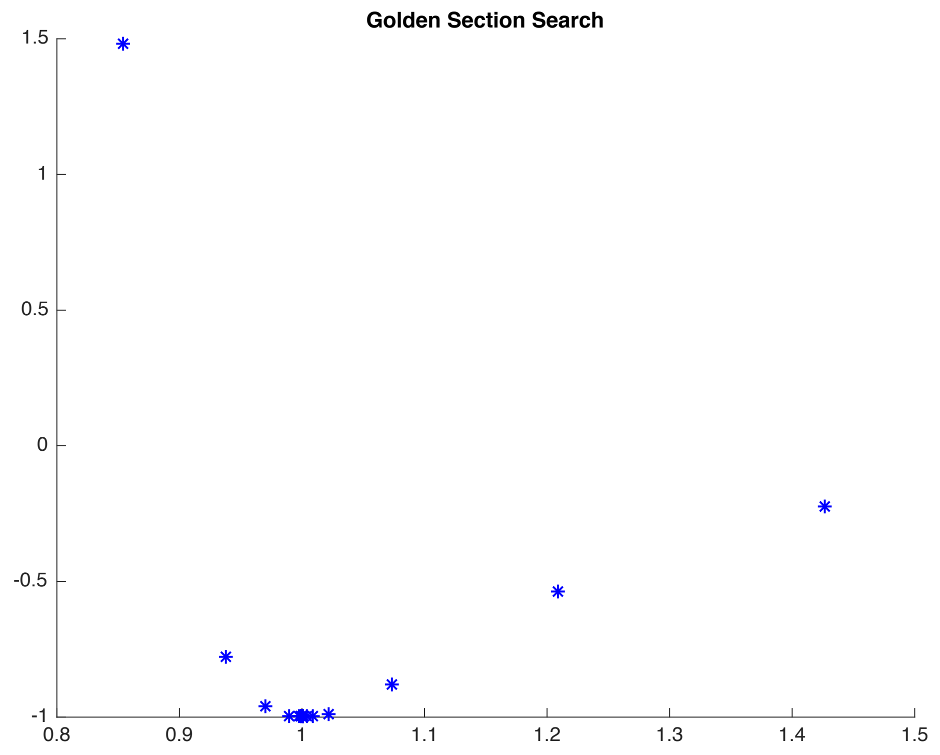
```
   3. GSS(@LJPotential,0.5, 1.5, 1e-12)
```



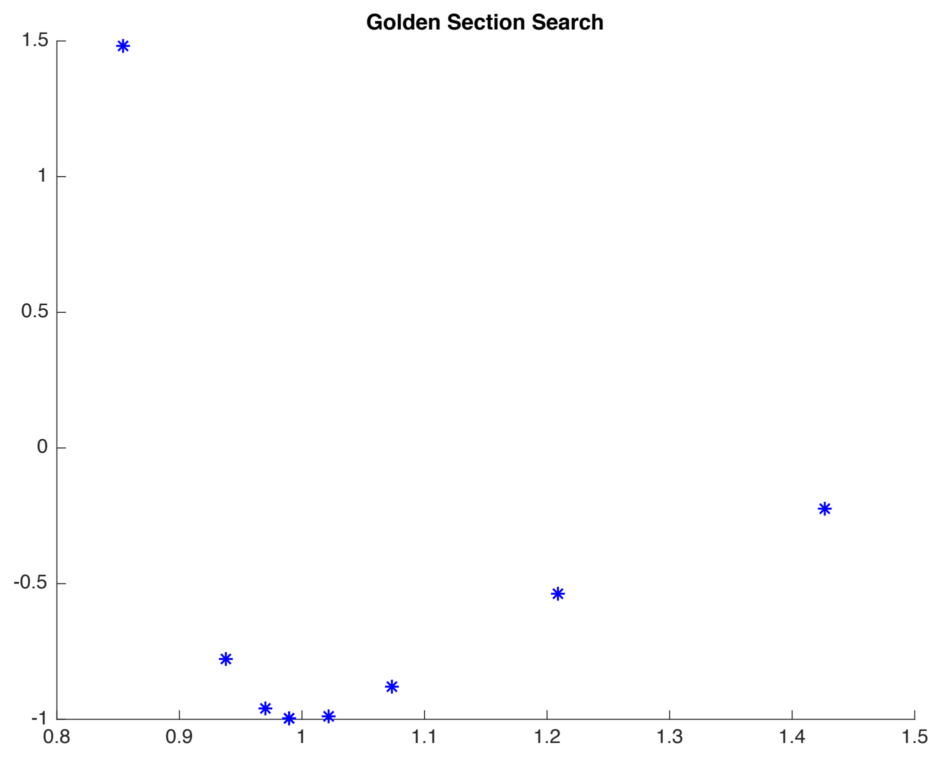
4. `GSS(@LJPotential,0.5, 1.5, **1e-9**)`



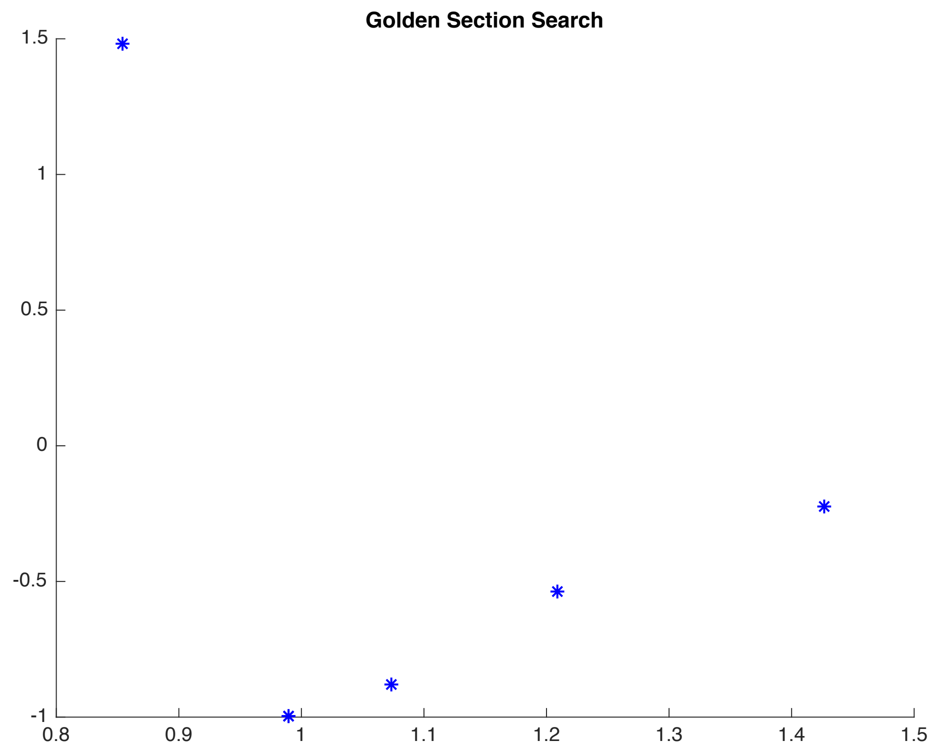
5. `GSS(@LJPotential,0.5, 1.5, **1e-3**)`



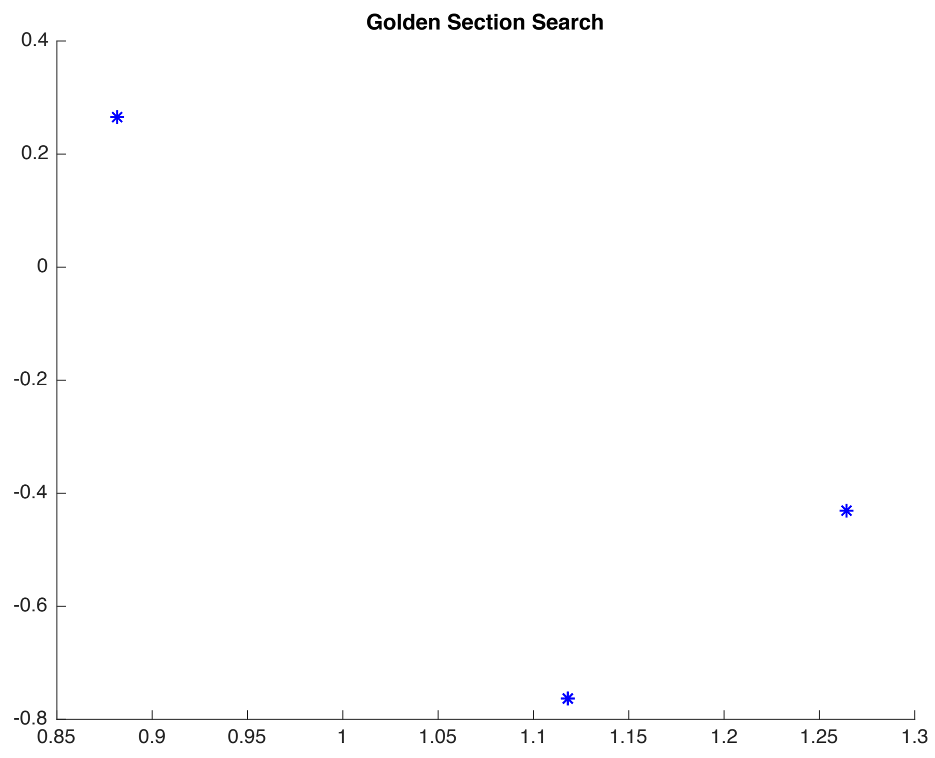
6. `GSS(@LJPotential,0.5, 1.5, `**`1e-1`**`)`



7. `GSS(@LJPotential,0.5, 1.5, `**`0.5`**



8. GSS(@LJPotential,0.5, 1.5, **0.8**)



Matlab code **LJPotential.m**

```
function U = LJPotential(r)
U = r.^(-12)-2*r.^(-6);
end
```

Matlab code **GSS.m**

```
function GSS(fi, ai, bi, tol)

% fi: function handler
% ai: start of interval
% bi: end of interval

figure; hold on;

a = ai;                         % start of interval
b = bi;                         % end of interval
f = fi;                         % function handler
epsilon = tol                   % accuracy value
iter= 50;                       % maximum number of
iterations
tau=double((sqrt(5)-1)/2);      % golden proportion
coefficient, around 0.618
n=0;                            % number of iterations

x1=a+(1-tau)*(b-a);             % computing x values
x2=a+tau*(b-a);

f_x1=f(x1);                     % computing values in x
```

```
points
f_x2=f(x2);

plot(x1,f_x1,'b*')                    % plotting x
plot(x2,f_x2,'b*')

while ((abs(b-a)>epsilon) && (n<iter))
    n=n+1;
    if(f_x1<f_x2)
        b=x2;
        x2=x1;
        x1=a+(1-tau)*(b-a);

        f_x1=f(x1);
        f_x2=f(x2);

        plot(x1,f_x1,'b*');
    else
        a=x1;
        x1=x2;
        x2=a+tau*(b-a);

        f_x1=f(x1);
        f_x2=f(x2);

        plot(x2,f_x2,'b*')
    end
    n=n+1;
end

% chooses minimum point
if(f_x1<f_x2)
    sprintf('x_min=%f', x1)
    sprintf('f(x_min)=%f ', f_x1)
    plot(x1,f_x1,'b*')
else
    sprintf('x_min=%f', x2)
    sprintf('f(x_min)=%f ', f_x2)
    plot(x2,f_x2,'b*')
end
title('Golden Section Search')
n
end
```

2) Successive Parabolic Interpolation method

Output for different initial conditions
Tried testing within range of x>0 per original condition

Table (3)

No.	a (start)	b (end)	Tolerance	x_{min}	$f(x)_{min}$	Iter
1	-10.0	0.0	1e-6	NaN	NaN	NaN
2	-1.0	1.0	1e-6	NaN	NaN	NaN
3	0.5	1.5	1e-6	1.000000	-1.00000	19
4	1.0	10.0	1e-6	5.538000	-0.00007	11
5	0.5	1.5	1e-12	1.000000	-1.00000	37
6	0.5	1.5	1e-9	1.000000	-1.00000	30
7	0.5	1.5	1e-3	1.001300	-0.99994	10
8	0.5	1.5	0.1	1.124800	-0.74378	3
9	0.5	1.5	0.5	1.249900	-0.45576	2
10	0.5	1.5	0.8	1.249900	-0.45576	2

Note: NaN indicates that the sequence diverges

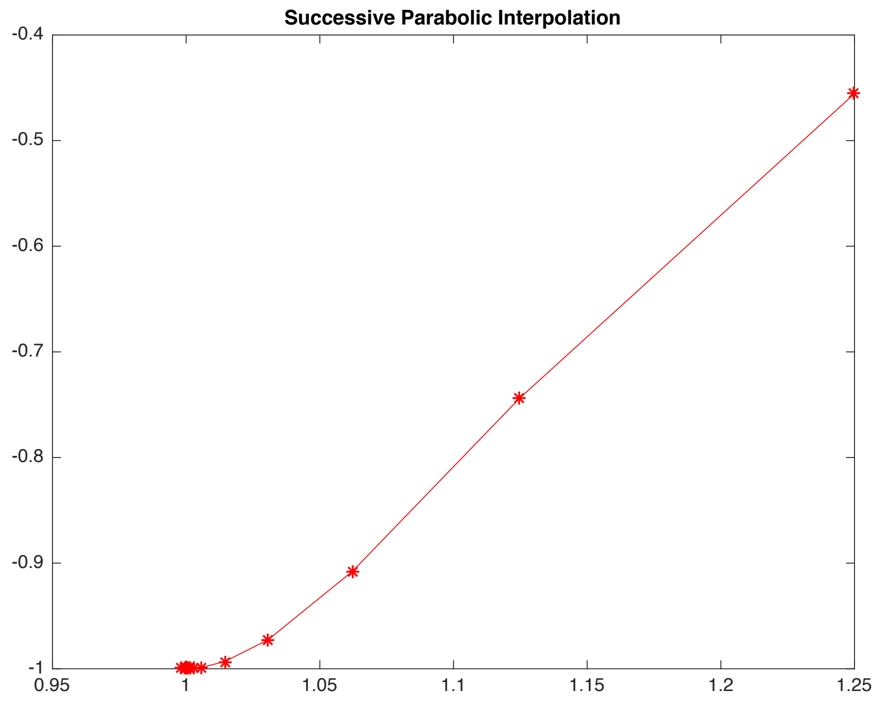
Observation: Unlike the Golden Search method, the Successive Parabolic Interpolation method diverges when the domain interval includes an asymptotic region (x = 0 for this case) and causes the sequence to never converge. This is illustrated for initial conditions case no. 1 and 2 from Table(3).

Also it is interesting to see the how the direction of which the x converges toward x_min differ depending on the initial conditions. Refer to the two graphs below. (graph (2)-1 and graph (2)-2)

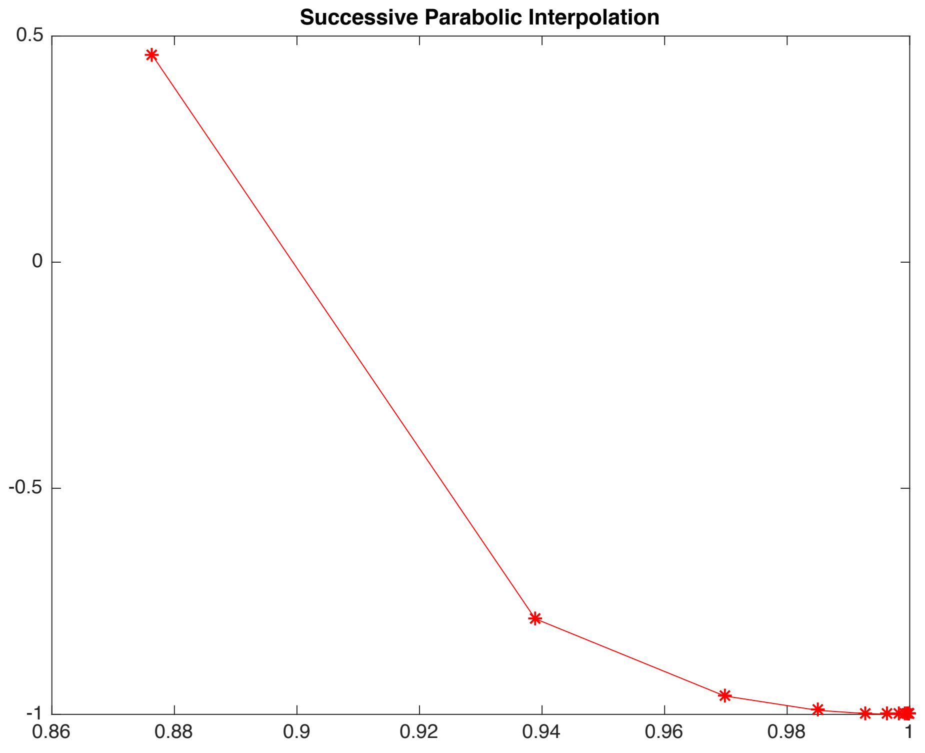
Matlab code for SPI.m

```
% ---------Parabolic Successive Interpolation.m----------
function PSI(fi, ai, bi, tol)
%initialization
a = ai;                         % start of interval
b = bi;                         % end of interval
mid = (a+b)*0.5;
f = fi;                         % function handler
epsilon = tol;                  % accuracy value
iter= 1000;                     % maximum number of
iterations
x = [a mid b];                  % initialize x
y = f(x);                       % initialize f(x)
k = 1;                           % number of iterations

while (k<iter)
  % compute new X
  Xnew = mid - 0.5*((mid-a).^2*(y(2)-y(3))-(mid-
b).^2*(y(2)-y(1)))./((mid-a)*(y(2)-y(3))-(mid-b)*(y(2)-
y(1)));
  X_vec(k) = Xnew;
  if Xnew > mid
      if f(Xnew)<f(mid)
          x = [mid, Xnew, b];
      else
          x = [a, mid, Xnew];
      end
  else
      if f(Xnew)<f(mid)
          x = [a, Xnew, mid];
      else
          x = [Xnew, mid, b];
      end
  end

  % assign new bracket values for x
  a = x(1);
  mid = x(2);
  b = x(3);

  % check tolerance condition error < epsilon
  if k > 1
      error = abs(X_vec(k)-X_vec(k-1))/abs(X_vec(k));
      if error<epsilon
          break
      end
  end
  k =  k+1;
end
plot(X_vec, f(X_vec), 'r*-')
title('Successive Parabolic Interpolation');
k, x_min = X_vec(k), f_min = f(X_vec(k))
end
```

Graph(2)-1
Graph plot for **PSI(@LJPotential, 0.5, 1.5, 1e-6)**
Initial conditions interval = [0.5, 1.5] tol = 1e-6
Converges in 19 iterations with x = 1.0 and f(x) = -1.0



Graph(2)-2
Graph plot for **PSI(@LJPotential, 0.5, 1.0, 1e-6)**
Initial conditions interval = [0.5, 1.0] tol = 1e-6
Converges in 18 iterations with x = 1.0 and f(x) = -1.0



3) Newton's method

Output for different initial conditions

Tried testing within range of x>0 per original condition

Table (4)

No.	X0 (start)	Tolerance	x_{min}	$f(x)_{min}$	Iter
1	0.1	1e-12	1	-1	38
2	0.1	1e-09	1	-1	37
3	0.1	1e-06	1	-1	37
4	0.1	1e-03	1	-1	35
5	0.1	0.01	0.1077	41094 e+7	1
6	0.1	0.1	0.1077	41094 e+7	1
7	0.1	0.2	0.1077	41094 e+7	1
8	0.1	0.3	0.1077	41094 e+7	1
9	0.1	0.5	0.1077	41094 e+7	1
10	0.1	0.8	0.1077	41094 e+7	1

Table (5)

No.	X0 (start)	Tolerance	x_{min}	$f(x)_{min}$	Iter
1	-1.0	1e-6	-1.0	-1.0	1
2	-0.8	1e-6	-1.0	-1.0	8
3	-0.6	1e-6	-1.0	-1.0	12
4	-0.4	1e-6	-1.0	-1.0	18
5	-0.2	1e-6	-1.0	-1.0	27
6	0.0	1e-6	NaN	NaN	1
7	0.2	1e-6	1.0	-1.0	27
8	0.4	1e-6	1.0	-1.0	18
9	0.6	1e-6	1.0	-1.0	12
10	0.8	1e-6	1.0	-1.0	8
11	1.0	1e-6	1.0	-1.0	1
12	1.2	1e-6	NaN	NaN	698

Note: NaN indicates that the sequence diverged

Observation:

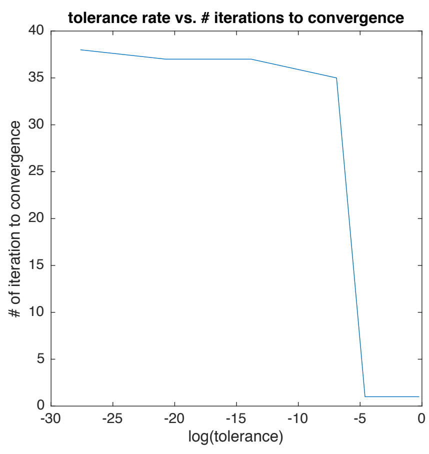
Newton's method converges properly for a tolerance value 0< tol <0.01 otherwise the function f(x) diverges.

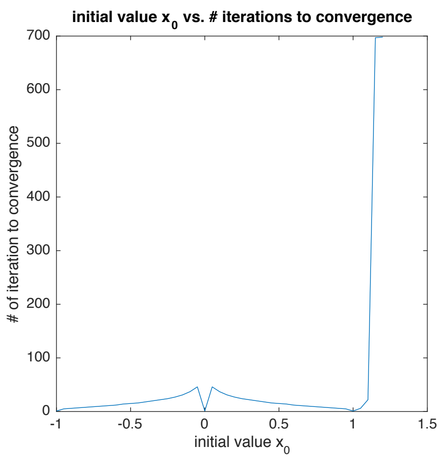
By fixing the tolerance value and varying the initial value x0, we saw that the function diverges at the asymptote x = 0 and also for values larger than 1.1 (x >= 1.1)

Also we can notice that the number of iterations it takes to converge decreases as the initial value is further away from 0

The graph below illustrates the relationship between thenumber of iteration it takes to converge to a local minima versus the initial conditions (tolerance value and initial input value x0)

Graph of Table(4) and Table(5)





Matlab code for Newton's method

```
% ------------------------NEWTON METHOD----------------------------

function [x, ex, fx] = Newton(x0, tol, iter)
  f = @(x)(x.^(-12))-(2.*x.^(-6));
  df = @(x)(-12*x.^(-13))+(12.*x.^(-7));
  ddf = @(x)(12*13*x.^(-14))-(12*7.*x.^(-8));

  x(1) = x0 - (df(x0)/ddf(x0));
  ex(1) = abs(x(1)-x0);
  k = 1;
  while (ex(k) >= tol) && (k-1 <= iter)
      x(k+1) = x(k) - (df(x(k))/ddf(x(k)));
      ex(k+1) = abs(x(k+1)-x(k));
      k = k+1;
  end
  fx = f(x)
end
```

Question 3

Matlab Code for finding min point fminbnd

```
fun = @(x)(x.^(-12))-(2.*x.^(-6))
x = fminbnd(fun, 1e-6,1,optimset('TolX',1e-
20,'Display','iter'))
```

function fminbnd calculation results

n	**x**	**f(x)**	**Procedure**
1	0.8820	0.2649	initial
2	1.1180	-0.7619	golden
3	1.2639	-0.4304	golden
4	1.1255	-0.7420	parabolic
5	1.0279	-0.9769	golden
6	0.9721	-0.9659	golden
7	1.0056	-0.9989	parabolic

8	1.0028	-0.9997	parabolic
9	0.9990	-1.0000	parabolic
10	1.0000	-1.0000	parabolic
11	1.0000	-1.0000	parabolic
12	1.0000	-1.0000	parabolic
13	1.0000	-1.0000	parabolic
14	1.0000	-1.0000	parabolic

Output of function fminbnd for condition

```
Optimization terminated:
 the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-06

x =    1.0000
```

Observation:

The **fminbnd** (hybrid) procedure which uses parabolic and golden method alternatively took **12 steps** to converge to a stabilized value of (1, -1) = (x, f(x))

The **golden section search method (GSS)** using the same initial condition of interval (a, b) = (0.5, 1.5) and tolerance level = 1e-6 took **58 steps.** And the **successive parabolic interpolation (SPI)** method took **19 steps** and the **Newton method** (with initial value x_0 = 0.5 and tolerance = 1e-5) took **15 steps to converge to the minima.**

Performace ranking in terms of speed(# of iterations)
fminbnd (SPI + GSS hybrid) > Newton method > SPI > GSS

On the other hand, when we compare the robustness of the algorithms, the golden section search method is more reliable. GSS can guarantee convergence regardless of the interval value, (as long as range includes the point of interest). The parabolic method diverges if the asymptote is included in the interval of interest and the accuracy of decreases as the tolerance value increases. Also, newton’s method was even more unreliable as it is sensitive to both tolerance and initial value of computation (x0).