Problem 1: Root Solving

Problem Statement Consider the "Rosenbrock" function for x_1 and x_2:

$$f(\mathbf{x}) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

Part 1.1

Problem Statement Demonstrate the topology of the function for the ranges of $x_1 = [-3, 3]$ and $x_2 = [-1, 5]$

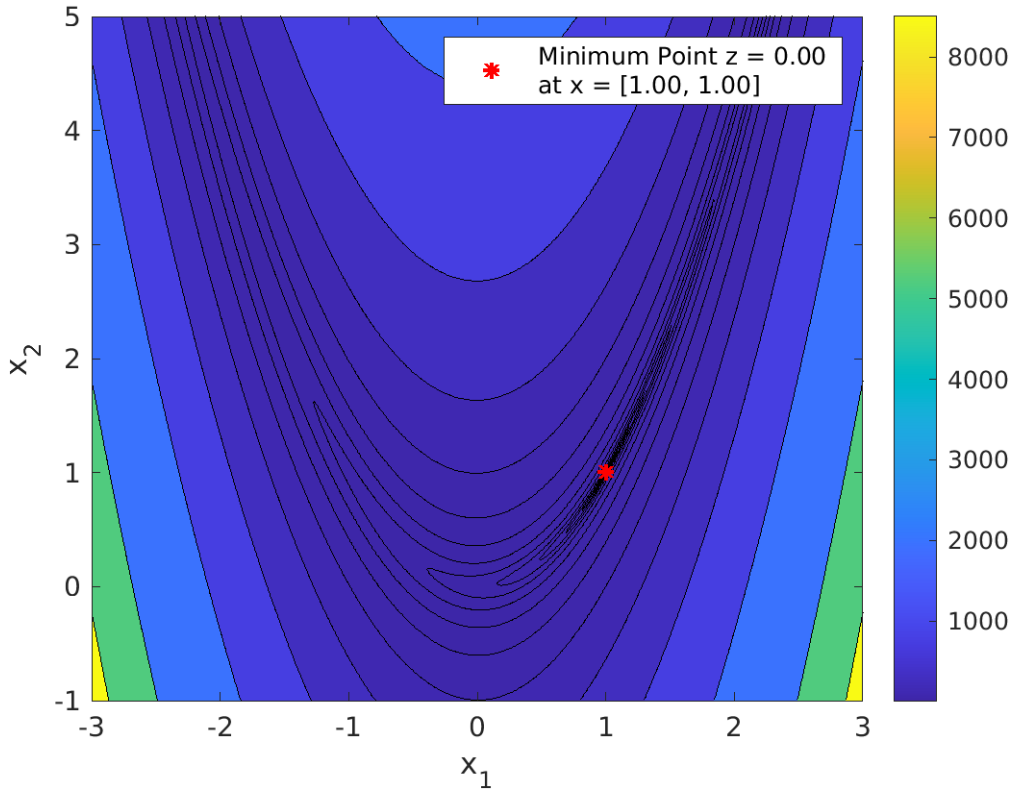


Figure 1: Contour map of Rosenbrock function over $x_1 = [-3, 3]$ and $x_2 = [-1, 5]$

Part 1.2

> **Problem Statement** Visually, where is the global minimum?

Looking specifically at the density of the contour lines, indicates a region around $[1,\ 1]$ being the minimum.

Part 1.3

> **Problem Statement** Using calculus, demonstrate the answer is a local minimum with the sufficient and necessary conditions.

Second Order Necessary Condition: $\nabla f(\mathbf{x}_*) = \vec{0}$

Second Order Sufficient Condition: $\nabla^2 f(\mathbf{x}_*)$ is positive definite

Calculating the first gradient of the function yields a 2×1 vector:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2\,x_1 - 400\,x_1\,(x_2 - {x_1}^2) - 2 \\ 200\,x_2 - 200\,{x_1}^2 \end{bmatrix}$$

$$\begin{bmatrix} 2\,x_1 - 400\,x_1\,(x_2 - {x_1}^2) - 2 \\ 200\,x_2 - 200\,{x_1}^2 \end{bmatrix}\Bigg|_{\mathbf{x}=[1,1]^T} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \checkmark$$

With the first order gradient having zeros in every index, the second order necessary condition is satisfied. Taking the second gradient, using the first, yields a 2×2 matrix of second order partial derivatives:

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 1200\,{x_1}^2 - 400\,x_2 + 2 & -400\,x_1 \\ -400\,x_1 & 200 \end{bmatrix}$$

$$\texttt{chol}(\ \nabla^2 f([1,\ 1]^T)\ ) = \begin{bmatrix} \sqrt{802} & -\frac{200\sqrt{802}}{401} \\ 0 & \frac{10\sqrt{2}\sqrt{401}}{401} \end{bmatrix} \checkmark$$

Seeing as the Cholesky factorization was completed successfully, the second order gradient evaluated at $[1, 1]^T$ is positive definite, satisfying the second order sufficient condition.

Problem 2: Single Line Search

Problem Statement Build a general line search subroutine that minimizes an arbitrary n-dimensional function along an arbitrary search direction **s** starting at an arbitrary location **x** with an initial stride t.

Part 2.1

Problem Statement Using the Quadratic Polynomial and Golden Ratio Methods, minimize the Rosenbrock function along $\mathbf{s} = [-1,\ 1]$ from a starting point $\mathbf{x} = [4,\ -1]$ using an initial stride length of $t_0 = 0.1$ and an $\epsilon = 10^{-4}$. Plot the position of each function call and use a different marker for the initial guess and final solution. Report the number of function calls required.

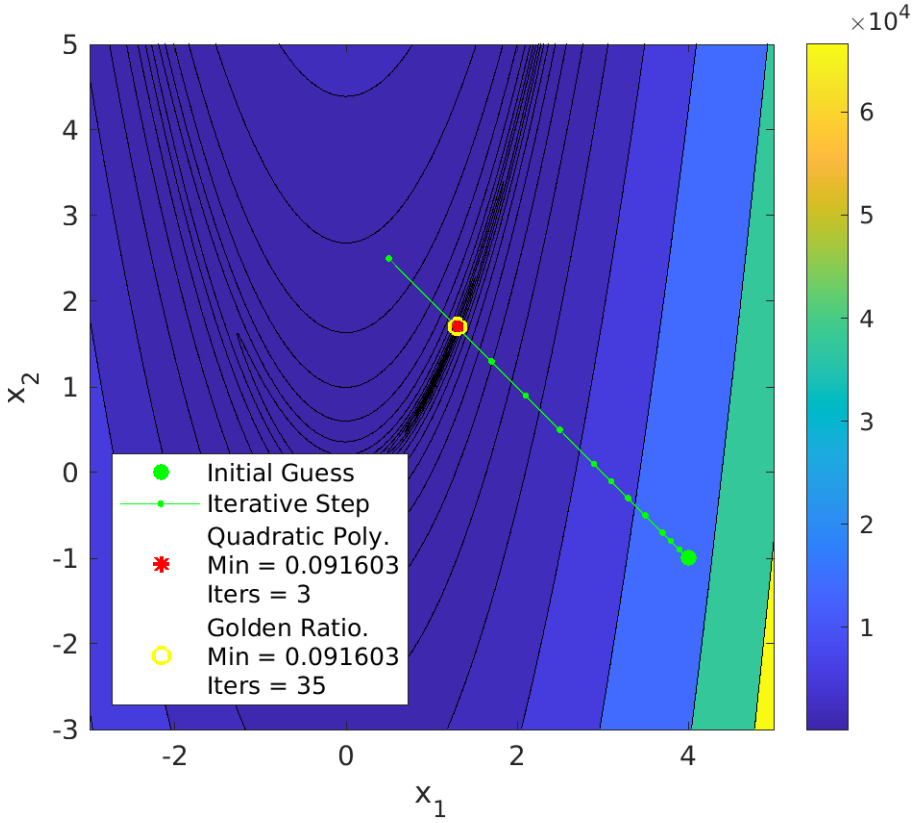


Figure 2: Stepping to solution space in green, then solving for minimum using both Quadratic Polynomial and Golden Ratio solvers.

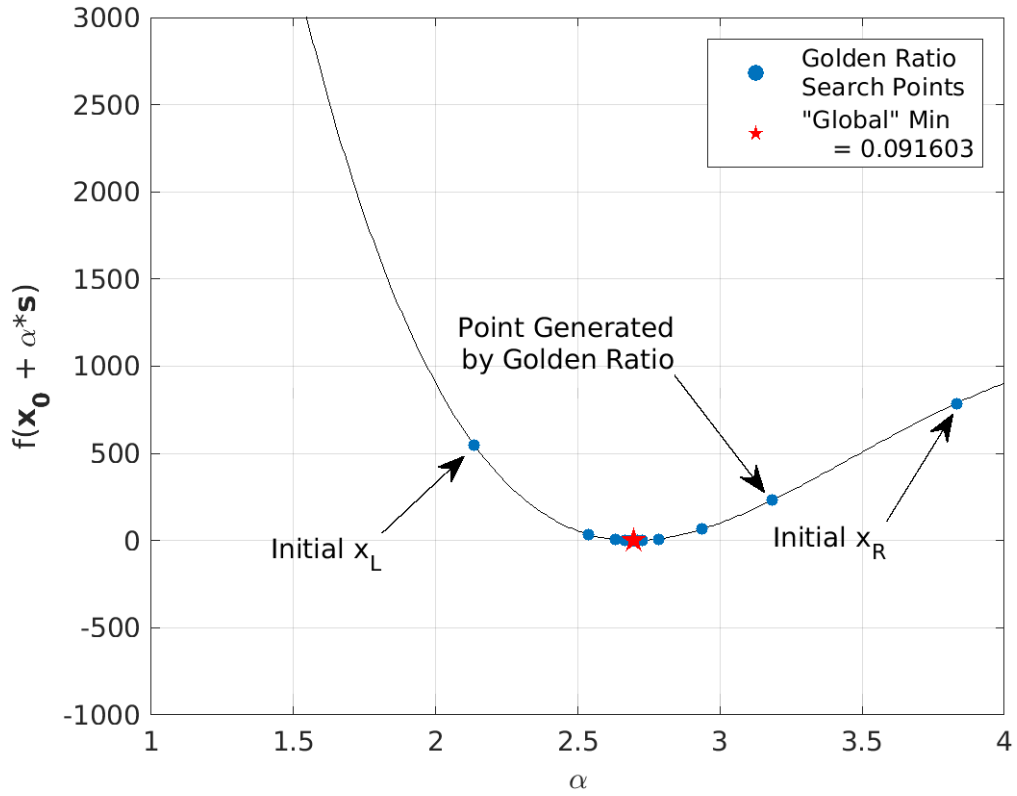


Figure 3: Slice of Rosenbrock function along $\mathbf{s} = [-1,\ 1]$ showing targeted minimum and search points created by Golden Ratio solver.

Part 2.2

> **Problem Statement** Use different strategies for stride adjustment, and discuss the results with regard to the number of function calls required. Present a summary table or plot of your investigation.

When adjusting the stride of the line search, the following method was used. Starting with the initial, given stride length, a certain number of steps were taken, and if the minimum was not bracketed, the stride length would be increased by some multiplication factor. The below table shows the results of varying the number of strides before increasing the step size, per row variance, and how much the stride was increased by, per column variance. Within each cell is the number of iterations to bracket the minimum, find the minimum using the quadratic polynomial method, and find the minimum using the golden ratio method, in that order.

Number Steps	1.1 Multiplier	1.25 Multiplier	1.5 Multiplier	2.0 Multiplier	2.5 Multiplier	3.0 Multiplier	5.0 Multiplier
2	19, 4, 33	14, 5, 34	11, 5, 35	9, 5, 36	8, 52, 37	7, 5, 37	7, 5, 44
4	23, 4, 32	19, 3, 33	16, 5, 33	13, 3, 35	11, 5, 35	10, 5, 35	9, 5, 37
8	26, 4, 32	23, 4, 32	21, 4, 33	18, 3, 34	17, 3, 34	16, 4, 33	13, 3, 34
16	28, 4, 31	27, 4, 31	25, 3, 32	23, 3, 32	22, 4, 33	21, 3, 33	19, 5, 34

One key result from the table is the inverse relationship between the number of iterations to bracket the minimum and find the minimum, with both methods, when varying either the multiplier or number of steps. The faster the minimum is bounded the longer it will take to find the minimum, due to the size of said bracket on the line. One outlier from the results is for the 2 step and 2.5 multiplier result of the quadratic polynomial method. Which hit an iteration limit, and bounced around very near the solution for infinity.

Problem 3: Adding the Outer Loop

Problem Statement Using the polynomial line search from Part 2, build wrapper routines for general n-dimensional unconstrained optimizers using:

1) Steepest Descent
2) Fletcher-Reeves
3) Polak-Ribiere
4) BFGS

Also build a full second order Levenberg-Marquardt solver with dynamic adjustment of the damping parameter. **Extra Credit:** Build a full trust region solver with dynamic trust region management.

Part 3.1

Problem Statement For each of the 5 methods, minimize the Rosenbrock function starting at $\mathbf{x} = [4, -1]$. Plot the **x** at each major iteration. Tune your codes and try to reduce the number of function calls to a minimum. State success or failure (using a maximum iteration limit of 1000), the final **x** and f to 8 digits, and the total number of function calls and derivative function calls (first and second order), also the CPU time required, and machine you are using. Discuss the results and your general strategy. Present a summary table with rows of each method and columns for the characteristics of each method mentioned above.

Algorithm	F_{min}	Func Calls	Grad Calls	Hess Calls	Run Time (seconds)
Steepest Descent	0.17807939	40863	1001	0	0.279271
Fletcher Reeves	1×10^{-8}	3766	93	0	0.079834
Polak Ribiere	0	3571	88	0	0.071204
BFGS	0	635	16	0	0.066949
Levenberg Marquardt	0	21	15	15	0.031304

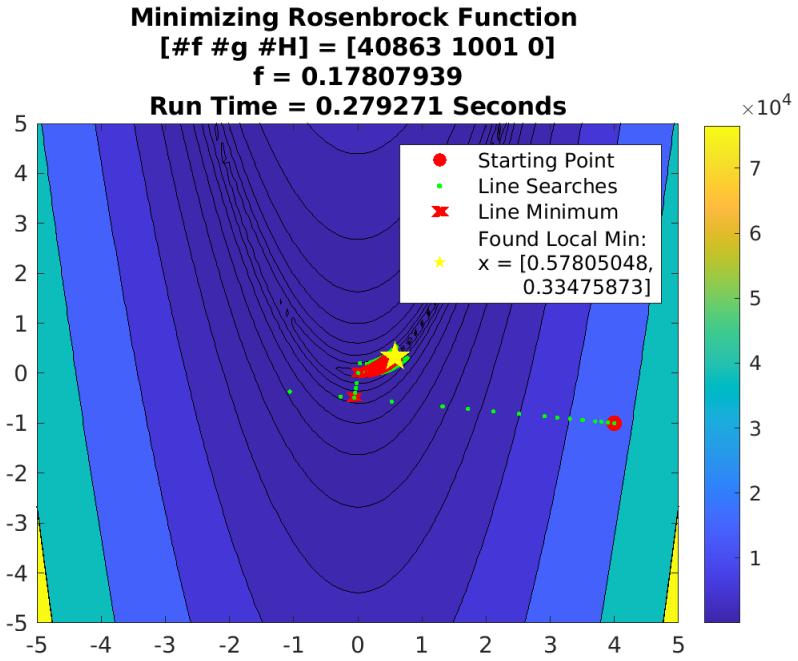


Figure 4: Finding minimum of Rosenbrock function using **Steepest Descent**. Solution fails after iteration budget is reached.

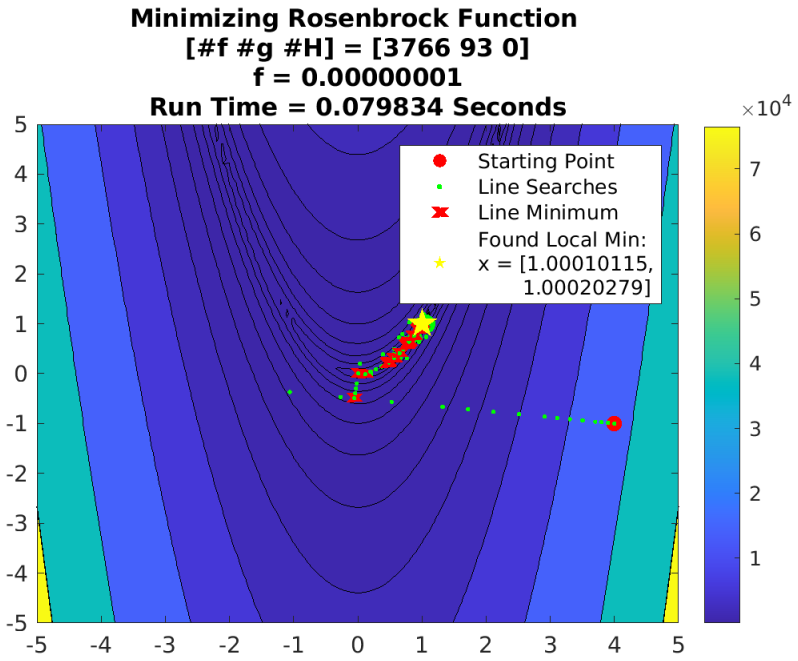


Figure 5: Finding minimum of Rosenbrock function using **Fletcher-Reeves** algorithm. Minimum successfully found within bounds of problem.

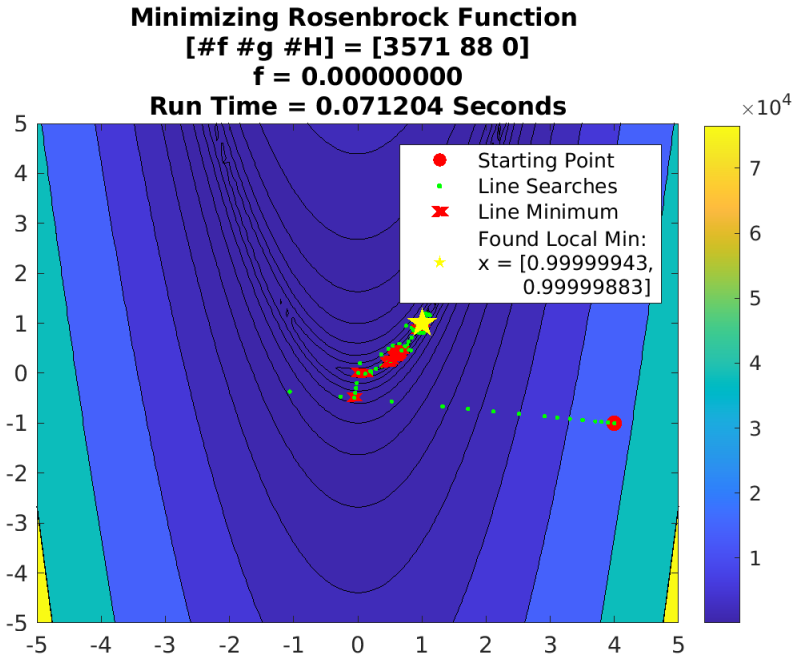


Figure 6: Finding minimum of Rosenbrock function using **Polak-Ribiere** algorithm. Minimum successfully found within bounds of problem.

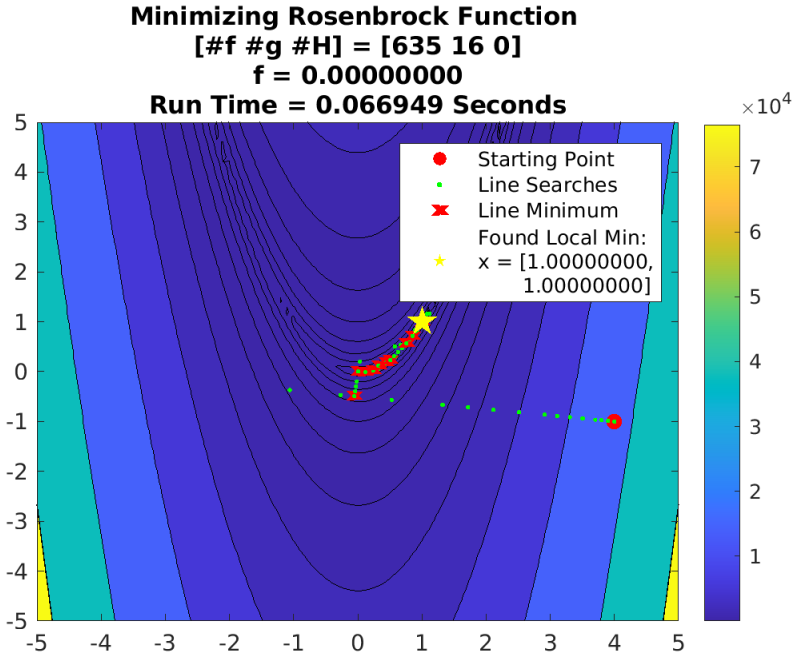


Figure 7: Finding minimum of Rosenbrock function using **BFGS** algorithm. Minimum successfully found within bounds of problem.

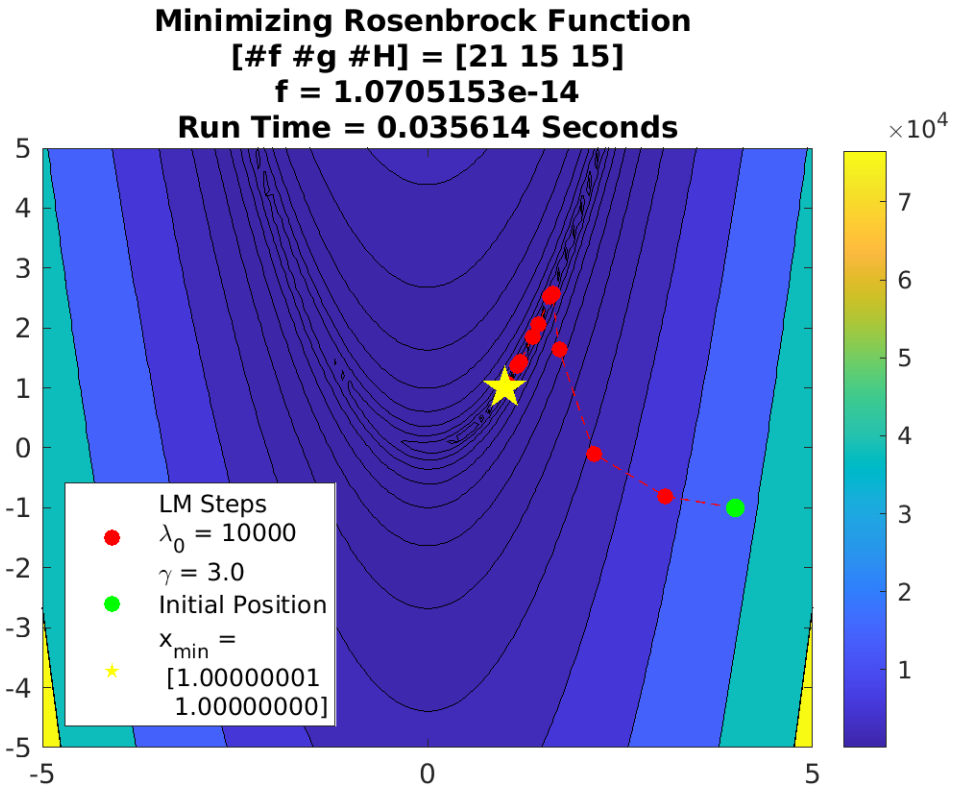


Figure 8: Finding minimum of Rosenbrock function using **Levenberg-Marquart** algorithm. Minimum successfully found within bounds of problem.

Part 3.2

Problem Statement Repeat Part 3.1 but instead minimize the function below with the two initial conditions: $\mathbf{x} = [4, -1]$ and $\mathbf{x} = [2, 3]$.

$$f = e^{u^2} + \sin(4x_1 - 3x_2)^4 + 0.5(2x_1 + x_2 - 10)^2$$

where

$$u = \frac{1}{2}(x_1^2 + x_2^2 + 25)$$

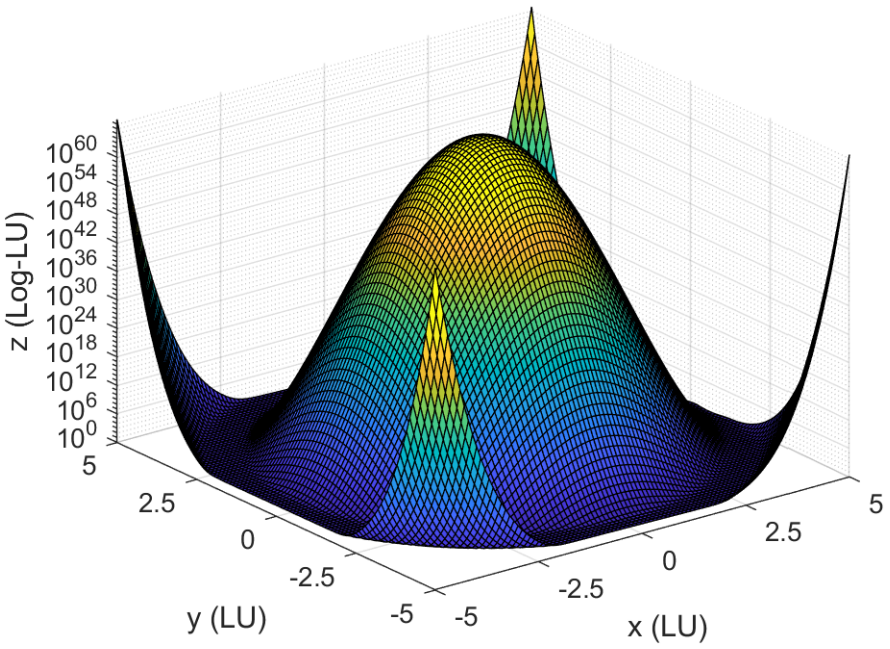


Figure 9: Plot of surface for above function, note log scale on z-axiz

Initial Condition: $x_0 = [-4, 1]^T$ For this specific initial condition it was found that the linear methods (even for BFGS's hessian approximation), converged onto a local minimum, and not the global minimum of the function where $F_{min} = 1$. On the contrary, the second order Levenberg-Marquardt method was able to find a much closer value in fewer iterations and run time.

Algorithm	F_{min}	Func Calls	Grad Calls	Hess Calls	Run Time (seconds)
Steepest Descent	1.18079703	408	11	0	0.110826
Fletcher Reeves	1.18084310	40316	1001	0	0.445786
Polak Ribiere	1.18079703	1742	47	0	0.060499
BFGS	1.18079703	279	8	0	0.030884
Levenberg Marquardt	1.0348054	49	24	24	0.039495

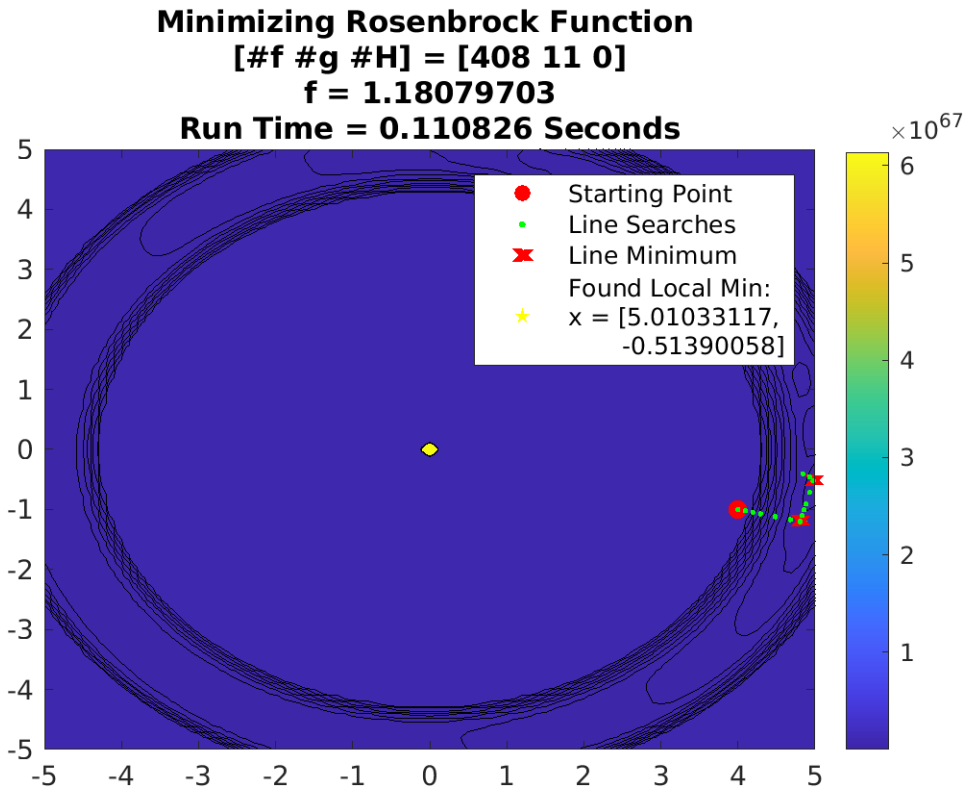


Figure 10: Finding minimum of given function using **Steepest Descent**. Solution converges to local minimum. **Apologies for the plot coloring, my MATLAB has been crashing trying to plot these as of late**

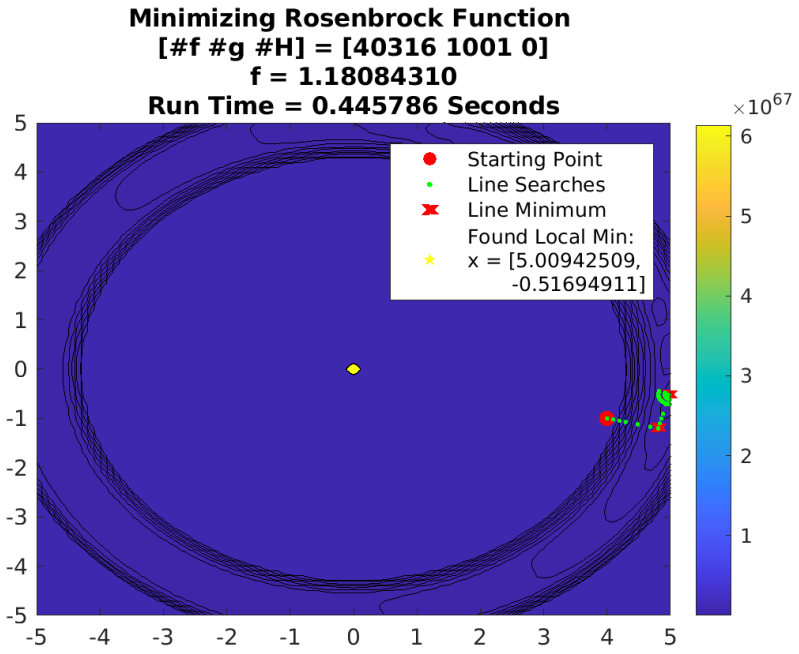


Figure 11: Finding minimum of given function using **Fletcher-Reeves** algorithm. Solution converges to local minimum.**Apologies for the plot coloring, my MATLAB has been crashing trying to plot these as of late**

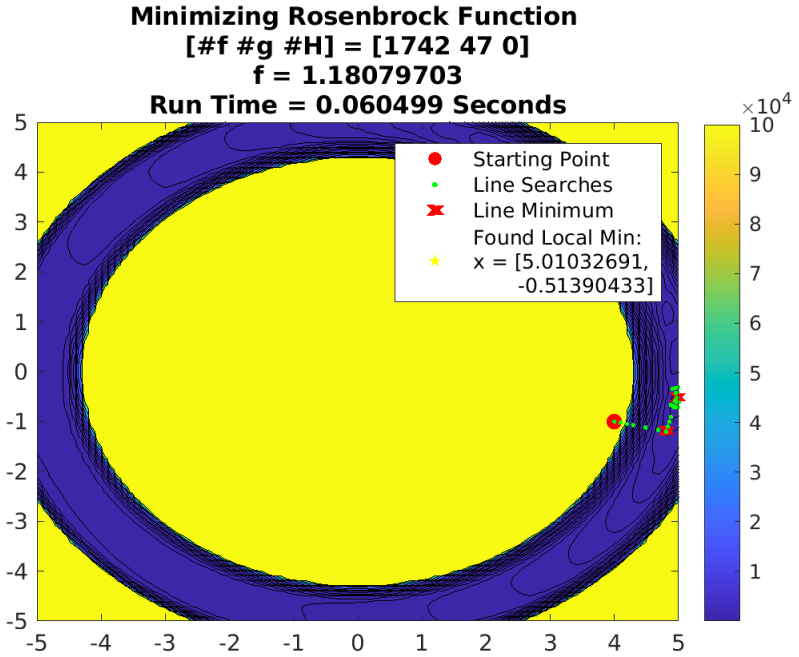


Figure 12: Finding minimum of given function using **Polak-Ribiere** algorithm. Solution converges to local minimum.

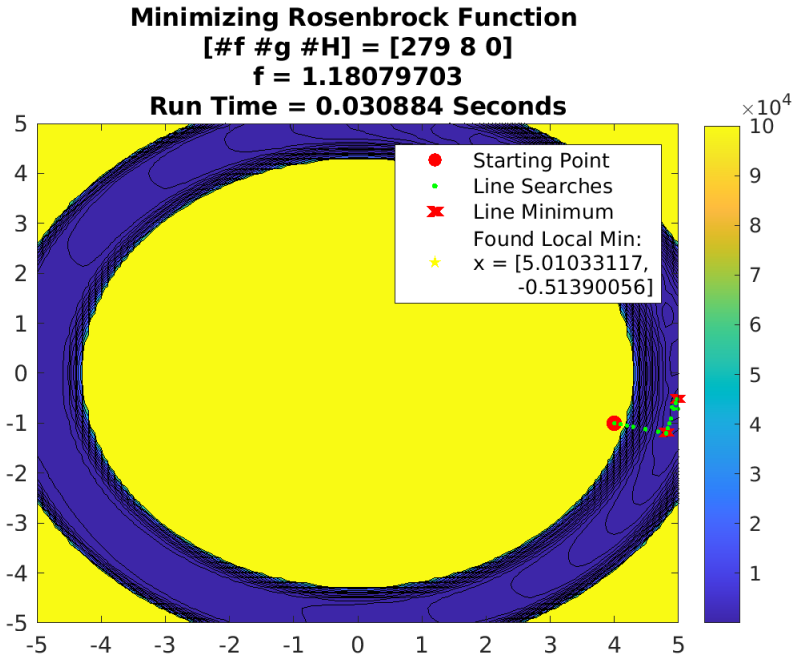


Figure 13: Finding minimum of given function using **BFGS** algorithm. Solution converges to local minimum.

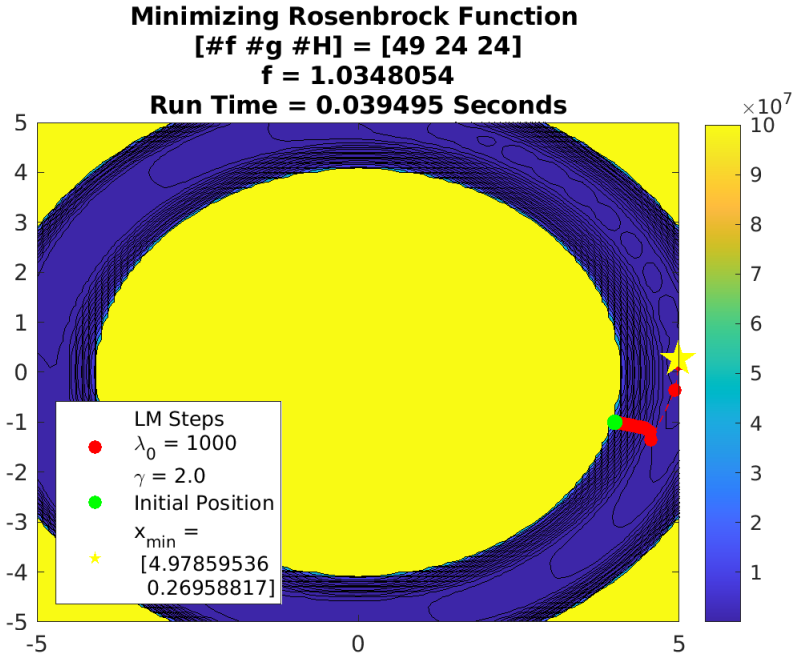


Figure 14: Finding minimum of given function using **Levenberg-Marquart** algorithm. Solution converges to local minimum.

Initial Condition: $x_0 = [2,\ 3]^T$ In the opposite case, the linear descent algorithms were able to, apart from the Fletcher-Reeves, quickly find the true minima within 30 iterations. While the second order method found a settling point not on the global minima, and attempts to perturb the resting state to evaluate if it was a saddle point did not change the result.

Algorithm	F_{min}	Func Calls	Grad Calls	Hess Calls	Run Time (seconds)
Steepest Descent	1.00000000	1130	30	0	0.0.62096
Fletcher Reeves	1.00000254	40474	1001	0	0.283291
Polak Ribiere	1.00000000	320	9	0	0.035704
BFGS	1.00474597	124	4	0	0.031874
Levenberg Marquardt	2.0522366	1001	1001	1001	0.049011

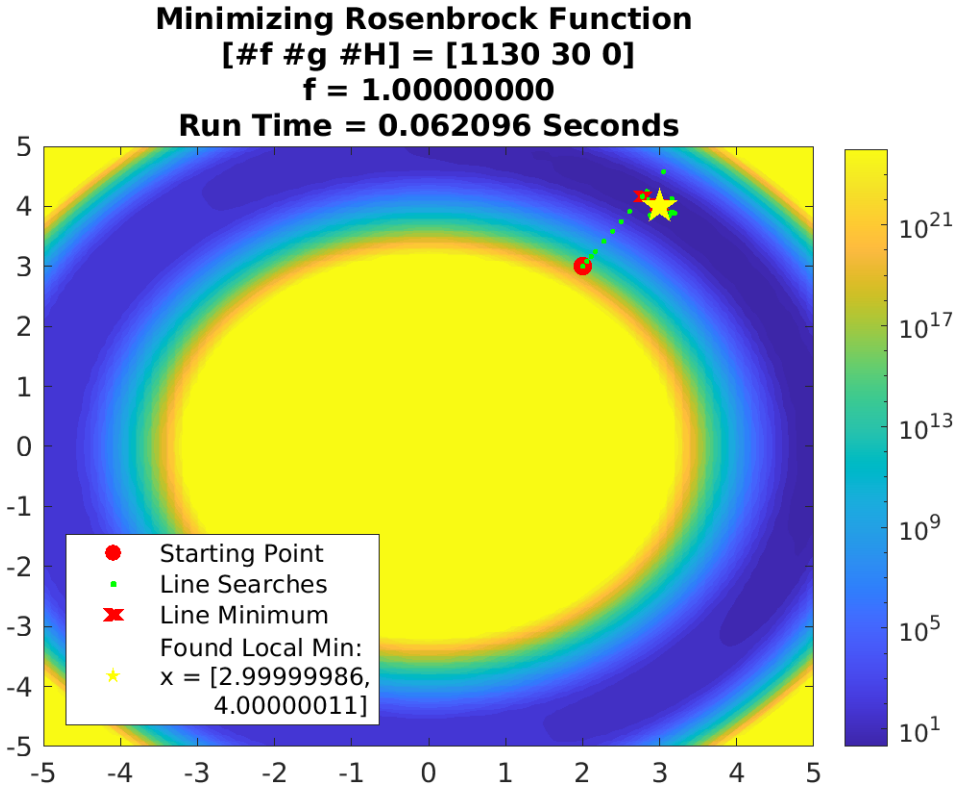


Figure 15: Finding minimum of given function using **Steepest Descent**. Solution converges to global minimum, although gets stuck in iteration loop.

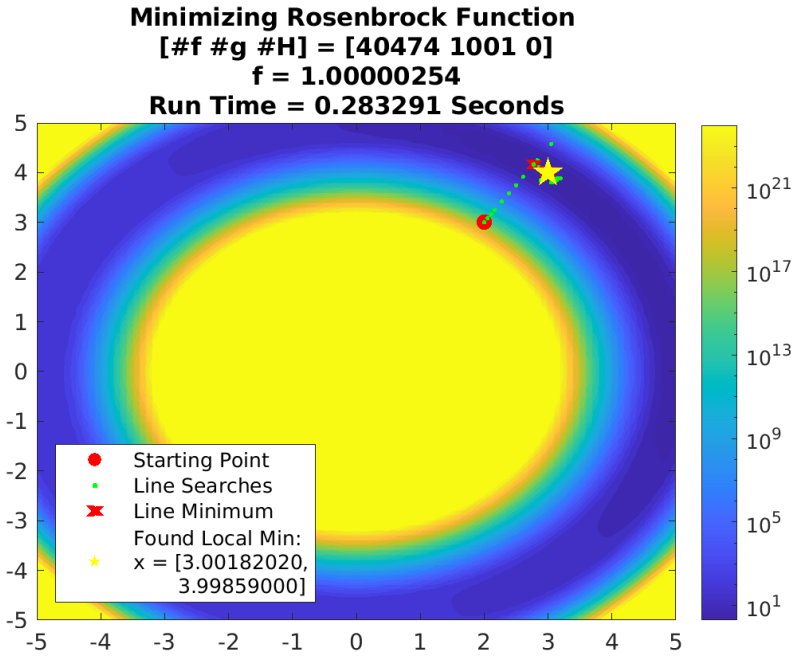


Figure 16: Finding minimum of given function using **Fletcher-Reeves** algorithm. Solution converges to global minimum.

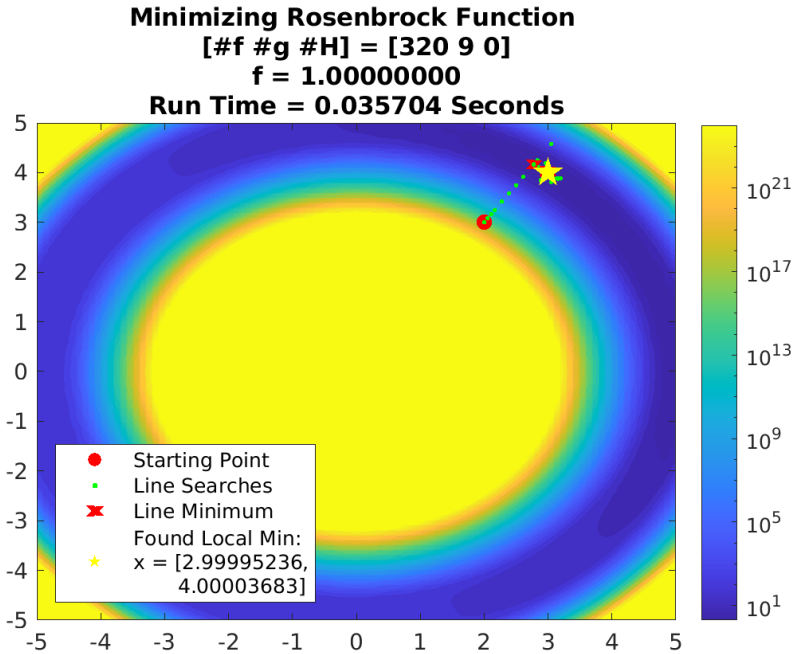


Figure 17: Finding minimum of given function using **Polak-Ribiere** algorithm. Solution converges to global minimum.

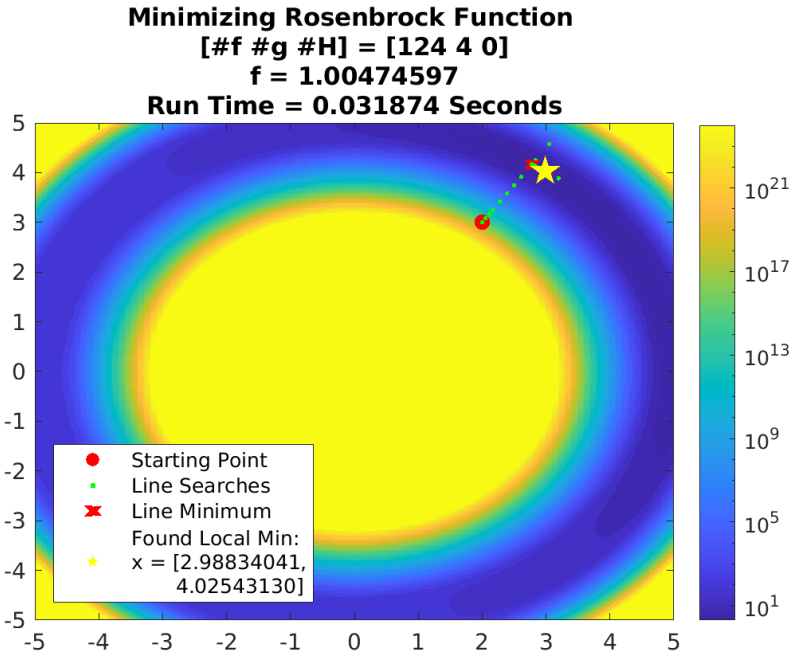


Figure 18: Finding minimum of given function using **BFGS** algorithm. Solution converges to global minimum.

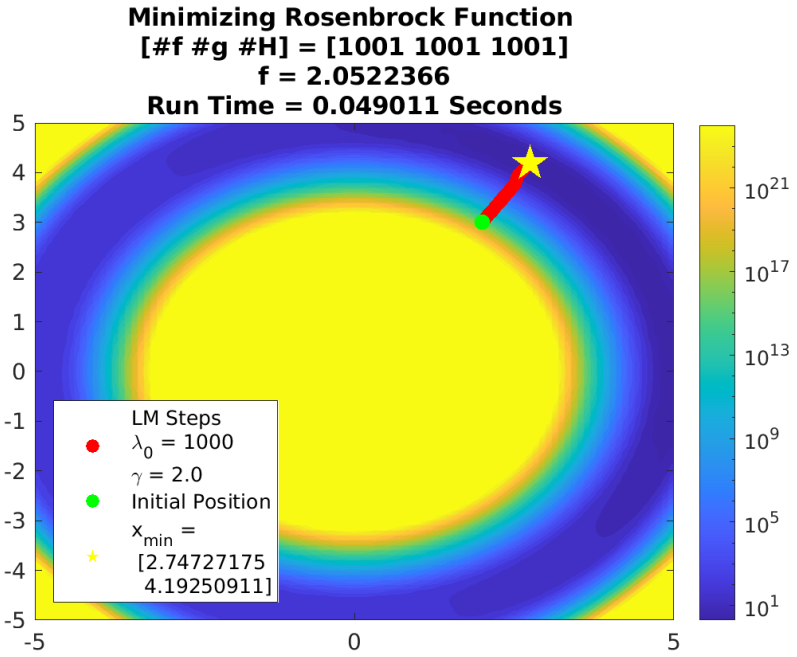


Figure 19: Finding minimum of given function using **Levenberg-Marquart** algorithm. Solution converges to local minimum.

Discussion of Results For each initial condition the linear and second order methods had different behaviors and found different minima, while the behavior is strange, it can be explained looking at the plot of the function as a surface rather than as a contour.

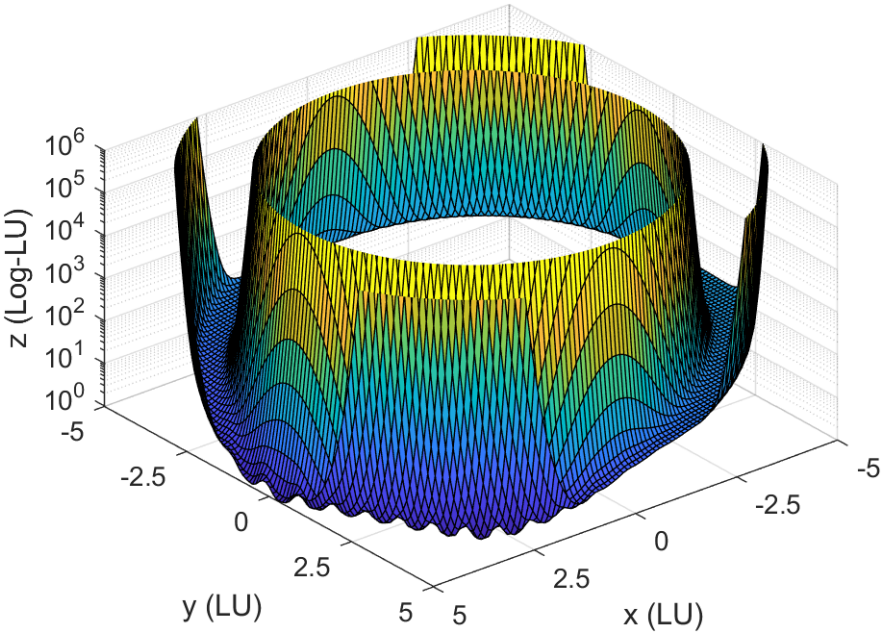


Figure 20: Isometric view of function minimized, note the ridged region closest to the front is where the global minima lies at $x = [3,\ 4]^T$.

The function proves difficult to minimize due to its steep slopes, very flat regions far from the global minimum (located in the other 3 corners from the solution), and global minimum surrounded by multiple different, isolated local minima. Depending the initial condition different minima are found, which is shown in the previous parts.

Problem 4: Orbits Problem

Problem Statement Find the initial velocity that solves the planar J_2-Lambert problem described below:

Input Constants:

$$J_2 = 0.1 \quad R = 0.9 \quad \mu = 1 \quad \text{TOF} = 83$$

$$\mathbf{r}_f = \begin{bmatrix} -2.891216226058043 \\ -1.254145998446107 \\ 0 \end{bmatrix}$$

Performance Index:

$$f = (\mathbf{r}_* - \mathbf{r}_f)^T(\mathbf{r}_* - \mathbf{r}_f) + 10^{10}(N - N_*)$$

Where N_* and $\mathbf{r}_*$ are the number of revolutions and final position of the propagated orbit, respectively.

Equations of Motion:

$$\ddot{\mathbf{r}} = \nabla U = \left[\frac{dU}{dx}, \frac{dU}{dy}, \frac{dU}{dz}\right]$$

$$U = \frac{\mu}{\sqrt{x^2+y^2+z^2}}\left[1 - \frac{R^2}{x^2+y^2+z^2}J_2\left(\frac{3z^2}{2(x^2+y^2+z^2)} - \frac{1}{2}\right)\right]$$

Initial Guesses:

$$\mathbf{r}_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{v}_0 = \begin{bmatrix} 0.05 \\ 1.305 \\ 0 \end{bmatrix} \quad \textbf{or} \quad \mathbf{v}_0 = \begin{bmatrix} -0.2 \\ 1.27 \\ 0 \end{bmatrix}$$

Show the solution space, and path for any method implemented successfully. Discuss the method used for computing derivatives, and solving the equations of motion. Discuss the approach including any failed or stalled efforts and generally discuss results of success. Plot the initial condition of the trajectory, the targeted position, and the propagated solution.

Equations of Motion For computing the equations of motion, the equations were symbolically solved using MATLAB's symbolic toolbox. Starting with the original U equation, the derivatives were first taken with respect to x and y as their equations will yield very similar answers where only the numerators have their x terms swapped with y terms. Since z requires an additional quotient rule it will yield different results. All equations are shown below:

$$\frac{dU}{dx} = \frac{\frac{2\,J_2\,R^2\,x\,\sigma_2}{{\sigma_1}^2} + \frac{12\,J_2\,R^2\,x\,z^2}{\left(2\,x^2+2\,y^2+2\,z^2\right)^2\,\sigma_1}}{\sqrt{\sigma_1}} + \frac{x\left(\frac{J_2\,R^2\,\sigma_2}{\sigma_1}-1\right)}{{\sigma_1}^{3/2}}$$

$$\frac{dU}{dy} = \frac{\frac{2\,J_2\,R^2\,y\,\sigma_2}{{\sigma_1}^2} + \frac{12\,J_2\,R^2\,y\,z^2}{\left(2\,x^2+2\,y^2+2\,z^2\right)^2\,\sigma_1}}{\sqrt{\sigma_1}} + \frac{y\left(\frac{J_2\,R^2\,\sigma_2}{\sigma_1}-1\right)}{{\sigma_1}^{3/2}}$$

$$\frac{dU}{dz} = \frac{z\left(\frac{J_2\,R^2\,\sigma_2}{\sigma_1}-1\right)}{{\sigma_1}^{3/2}} - \frac{\frac{J_2\,R^2\left(\frac{6\,z}{\sigma_3}-\frac{12\,z^3}{{\sigma_3}^2}\right)}{\sigma_1} - \frac{2\,J_2\,R^2\,z\,\sigma_2}{{\sigma_1}^2}}{\sqrt{\sigma_1}}$$

where

$$\sigma_1 = x^2 + y^2 + z^2$$

$$\sigma_2 = \frac{3\,z^2}{\sigma_3} - \frac{1}{2}$$

$$\sigma_3 = 2x^2 + 2y^2 + 2z^2$$

Derivatives To calculate the derivatives of the performance index, the integration of the trajectory is passed through a complex step derivative function. This incurs 3 additional functions for each gradient computed when minimizing.

Minimization Methods Searching the space and descending to the minima was left to the BFGS (Broyden-Fletcher-Goldfarb-Shanno Algorithm) to approximate the Hessian without requiring finite differencing the complex step method used previously. After using BFGS to find the direction of travel, a line search is conducted and the Golden Ratio method was used to minimize the line search. The number of iterations to find the solution showed high sensitivity to the initial stride length, t_0, which had to be tweaks from the value used earlier in the problem set. Associated plots available in Appendix. Among all tested values, $t_0 = 0.01$ yielded the fastest results with fewest iterations.

t_0	f Calls	g Calls	Run Time (sec)
0.1	736	21	5.1968
0.05	254	8	1.8634
0.01	229	8	1.8084
0.005	219	8	1.8685

Part 4.1

Problem Statement Solving the trajectory using the first initial condition for velocity:

$$\mathbf{v}_0 = \begin{bmatrix} 0.05 \\ 1.305 \\ 0 \end{bmatrix}$$

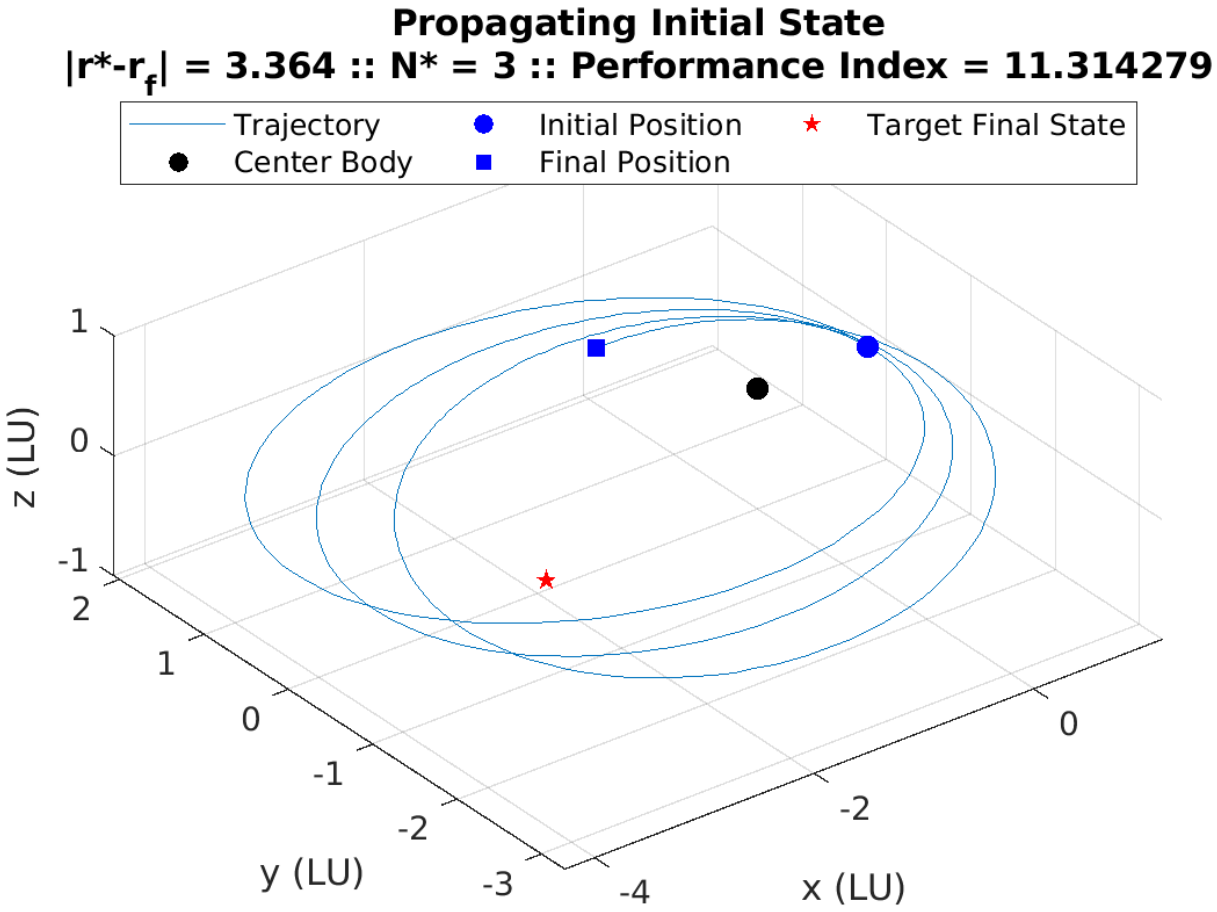


Figure 21: Initial state propagated for first $\mathbf{v}_0$ condition

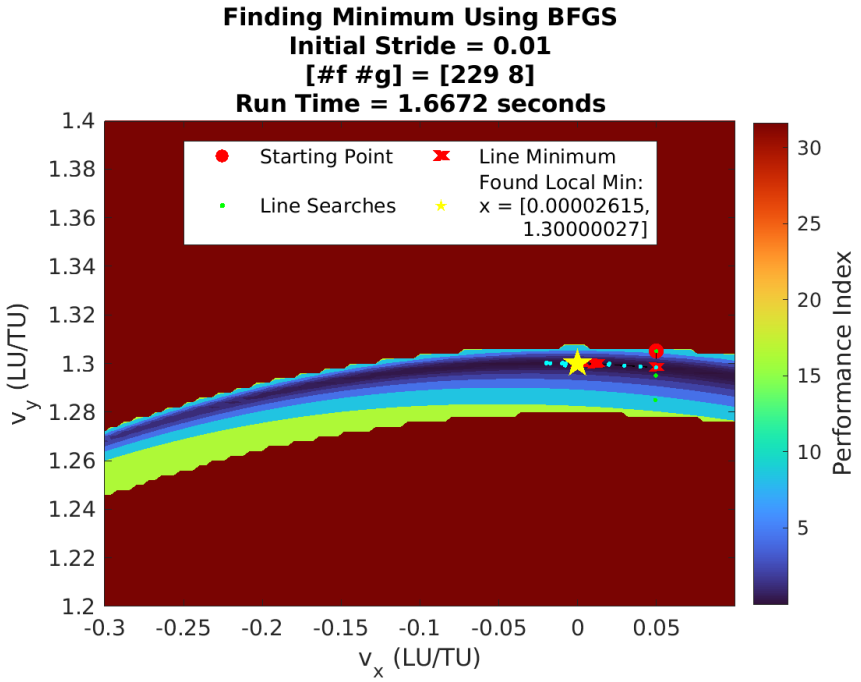


Figure 22: Finding minimum using BFGS algorithm with initial stride optimized for quick convergence. Red region indicates where propagated trajectory no longer has three revolutions.

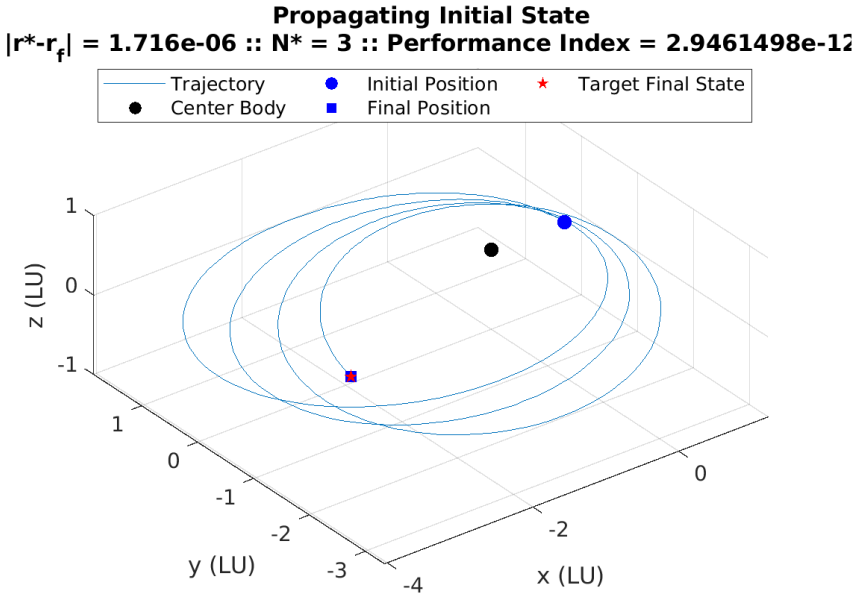


Figure 23: Optimized trajectory propagated terminating at targeted final state.

Part 4.2

Problem Statement Solving the trajectory using the second initial condition for velocity:

$$\mathbf{v}_0 = \begin{bmatrix} -0.2 \\ 1.27 \\ 0 \end{bmatrix}$$

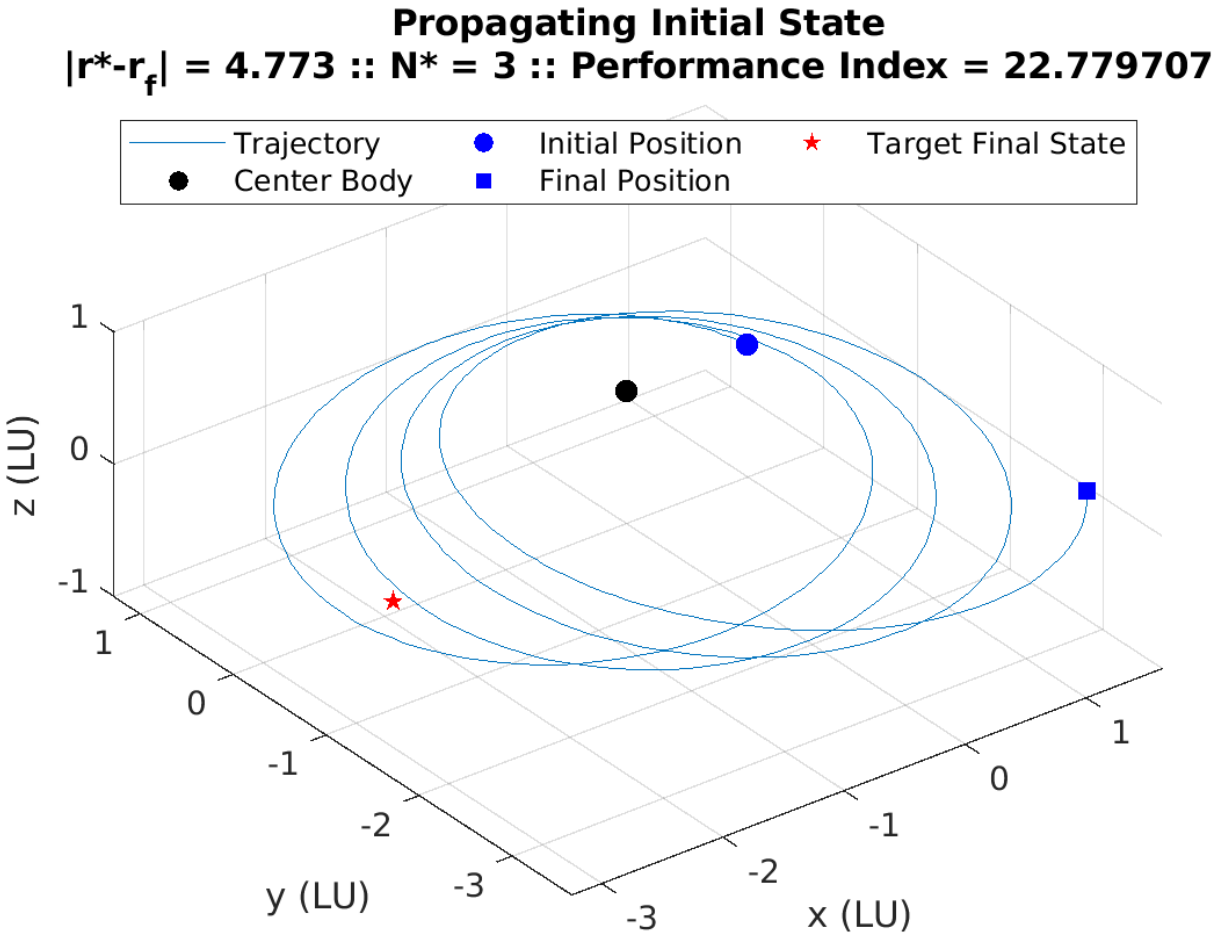


Figure 24: Initial state propagated for second $\mathbf{v}_0$ condition

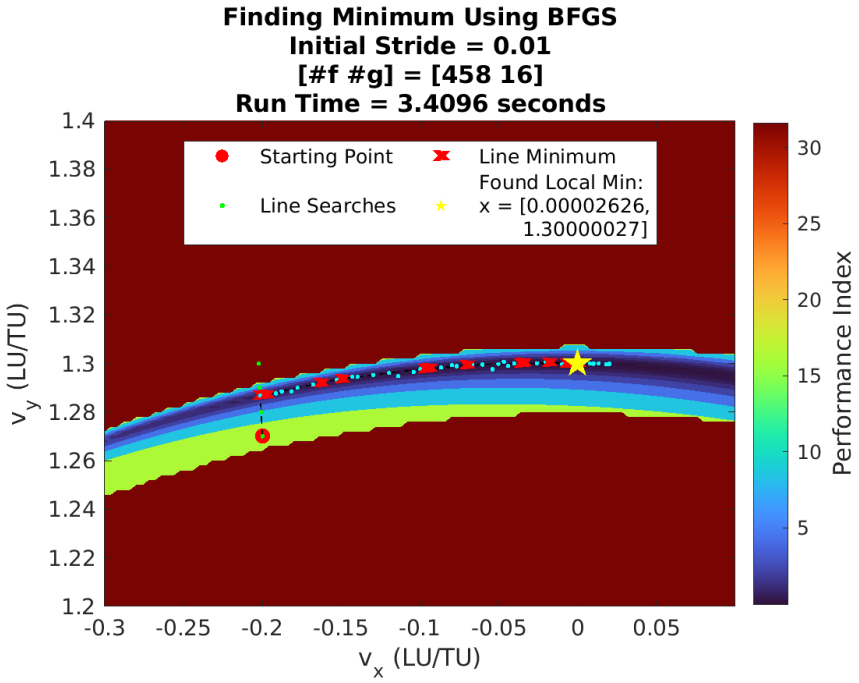


Figure 25: Finding minimum using BFGS algorithm with initial stride optimized for quick convergence. Takes more iterations due to initial distance from minimum. Red region indicates where propagated trajectory no longer has three revolutions.

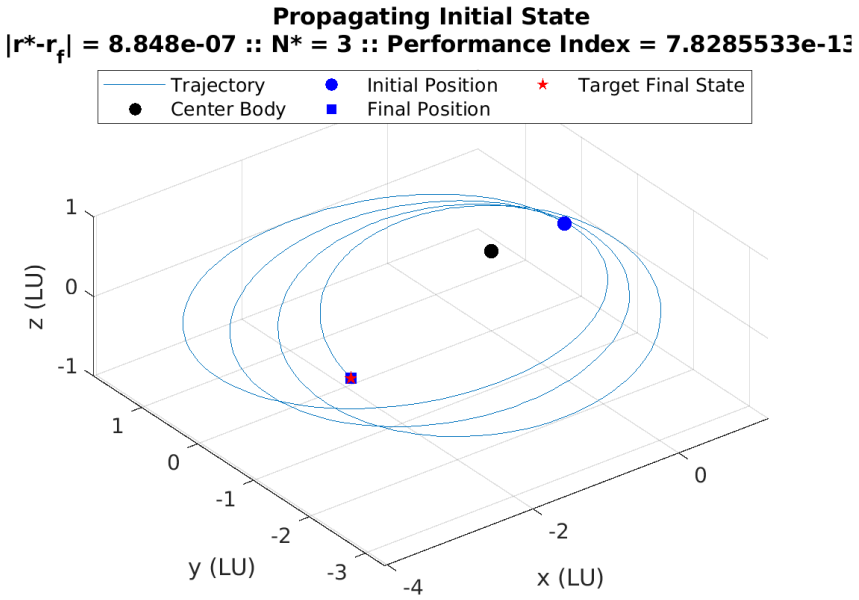


Figure 26: Optimized trajectory propagated terminating at targeted final state.

Discussion of Results

For both initial condition cases, the final state was correctly optimized to. In this case, the method of minimization was the BFGS algorithm. This choice was motivated by the fact that the derivatives of the cost function were found through complex step derivatives, which requires two propagation call for every gradient found. Also tested was variation of the initial stride length during the line search. Starting with the t_0 from the previous problems, 0.1, Figure 27 shows the path taken to find the minimum. This initial stride length was large enough to jump completely over the band of solutions where the number of revolutions, N, was 3. From there, it proceeded to find the solution, taking a much longer than average route. The next figures in the appendix show the increasingly smaller values of the initial stride.

Appendix

Problem 4 Stride Length Variation

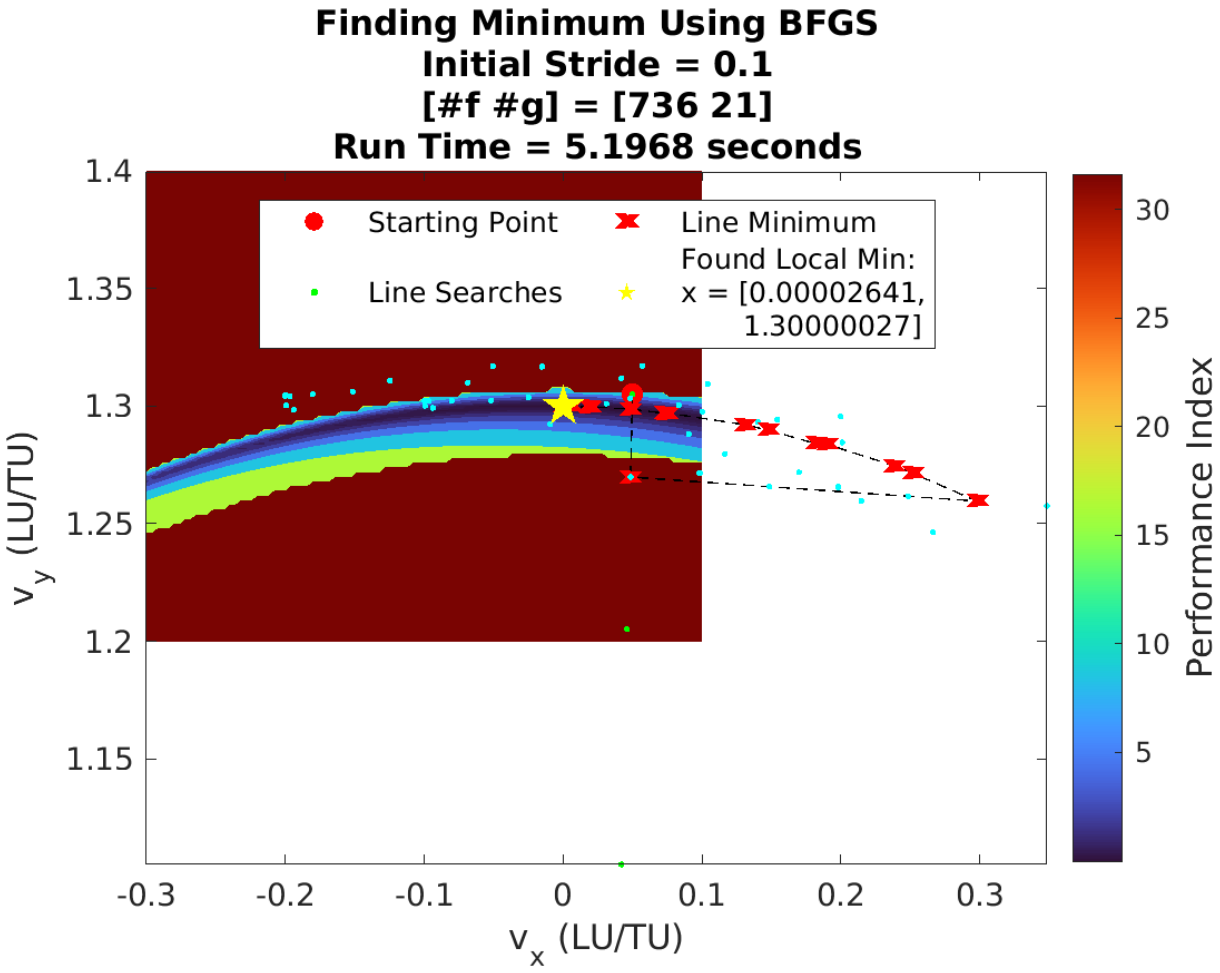
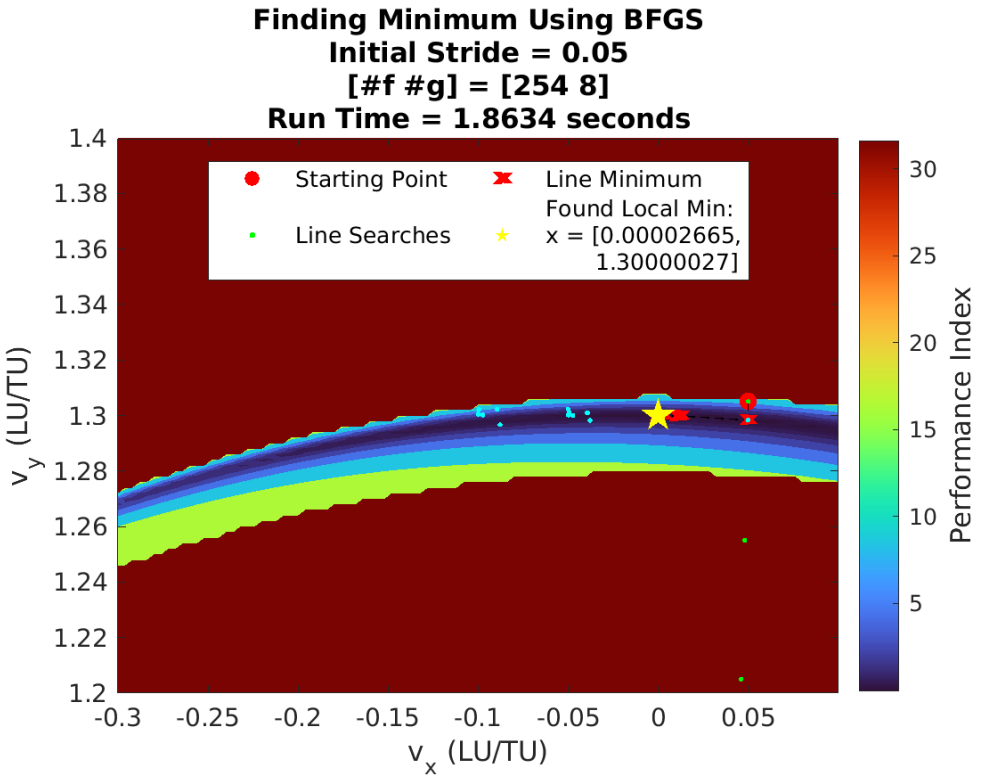
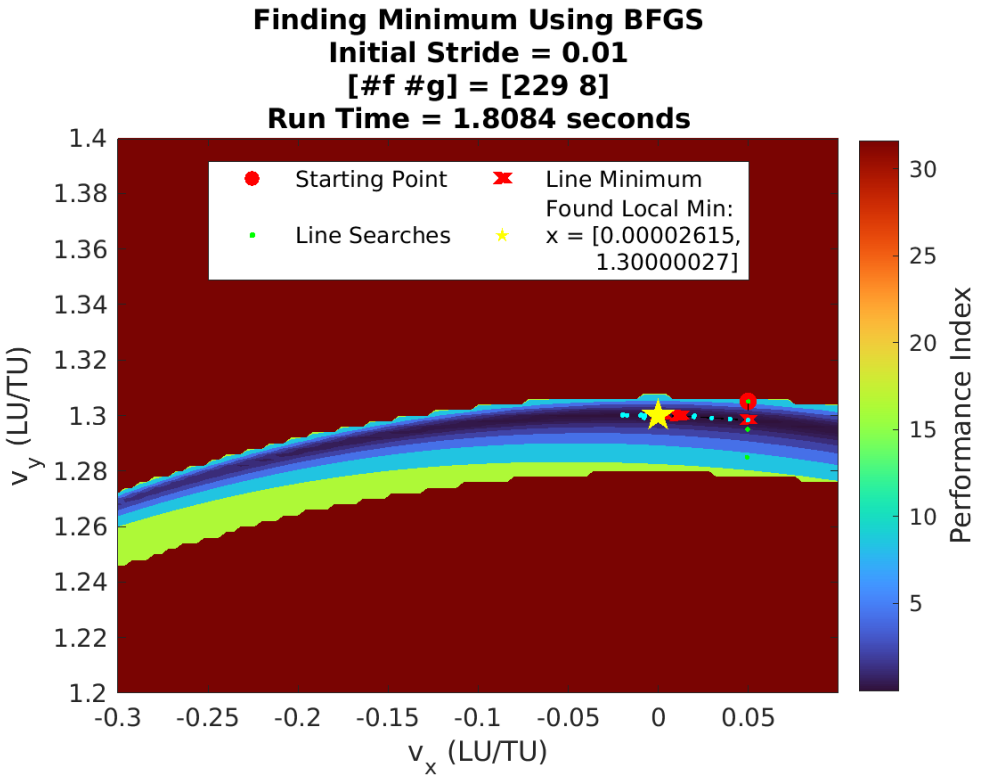
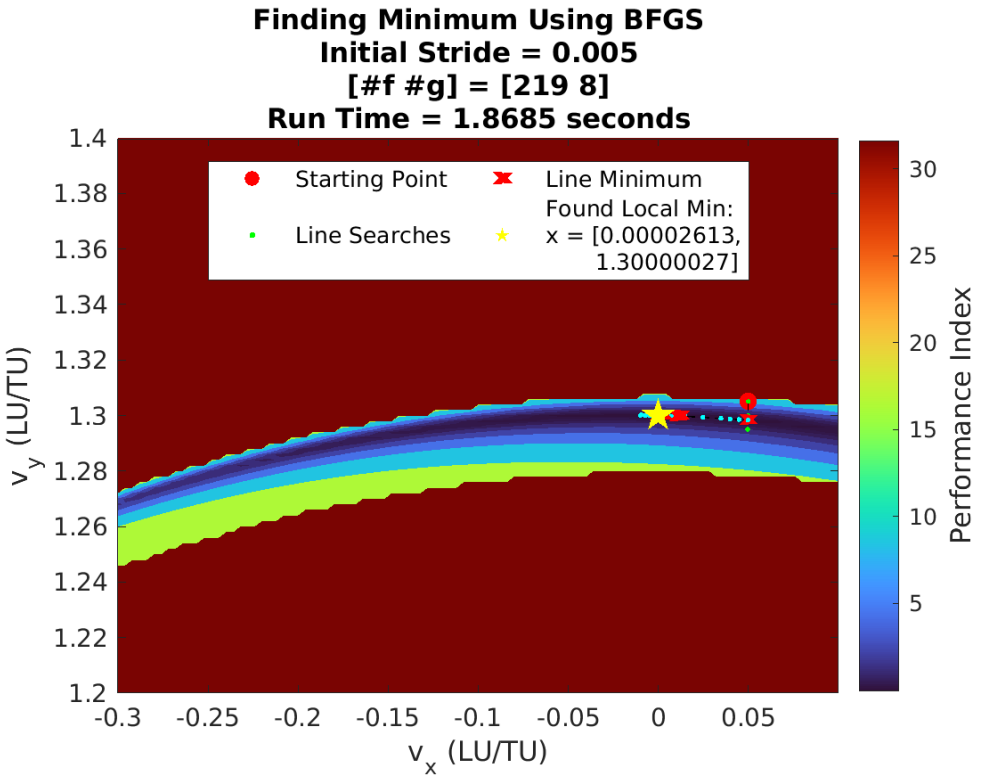


Figure 27: Finding minimum using BFGS algorithm with a variable initial stride length. Red region indicates where propagated trajectory no longer has three revolutions.


Finding Minimum Using BFGS
Initial Stride = 0.05
[#f #g] = [254 8]
Run Time = 1.8634 seconds
Starting Point
Line Minimum
Found Local Min:
x = [0.00002665,
1.30000027]
Line Searches
1.4
1.38
1.36
1.34
1.32
1.3
1.28
1.26
1.24
1.22
1.2
v_y (LU/TU)
-0.3
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
v_x (LU/TU)
30
25
20
15
10
5
Performance Index


Finding Minimum Using BFGS
Initial Stride = 0.01
[#f #g] = [229 8]
Run Time = 1.8084 seconds
Starting Point
Line Minimum
Found Local Min:
x = [0.00002615,
1.30000027]
Line Searches
1.4
1.38
1.36
1.34
1.32
1.3
1.28
1.26
1.24
1.22
1.2
v_y (LU/TU)
-0.3
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
v_x (LU/TU)
30
25
20
15
10
5
Performance Index



MATLAB Live Script Code Listings

(next page)

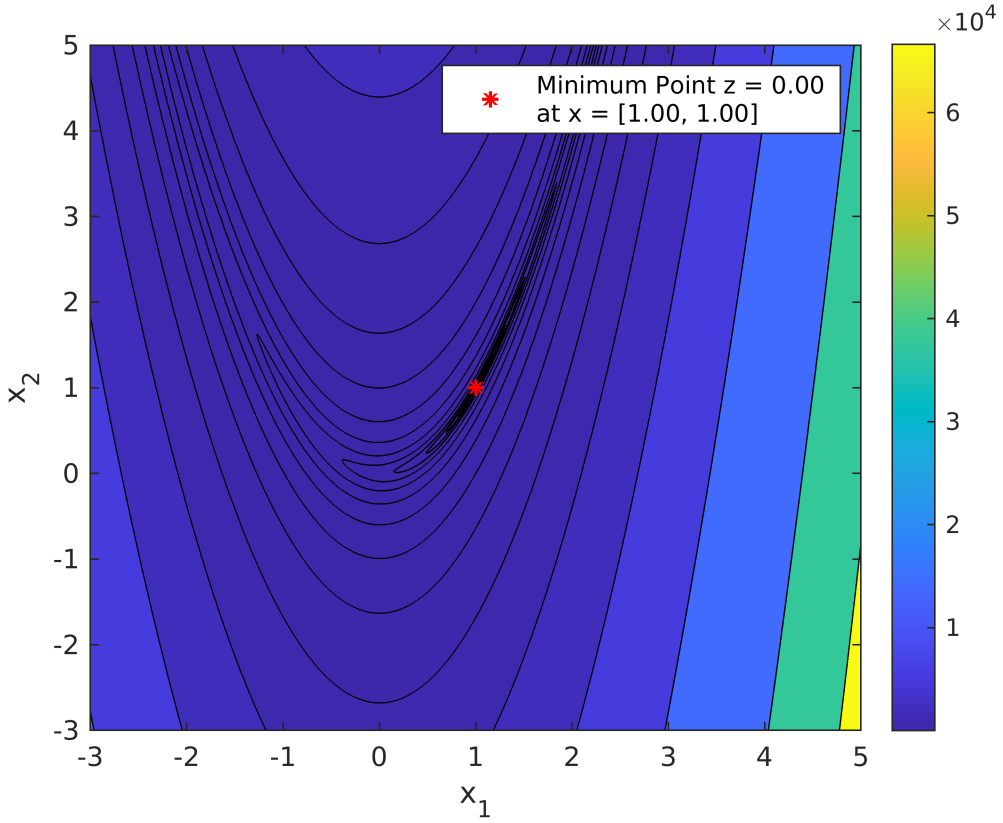
ASE387P Homework 3

Problem 1

```
clear; clc
```

Part a-b

```
f = @(x) (1-x(1))^2 + 100*(x(2)-x(1)^2)^2;
del = 0.01;
x1 = -3:del:5;
x2 = -3:del:5;
z = zeros(length(x1));
for i = 1:length(x1)
    for j = 1:length(x1)
        z(j, i) = f([x1(i), x2(j)]);
    end
end
contourf(x1, x2, z, [logspace(-4, 5, 22), max(max(z))*0.85], 'HandleVisibility',"off")
colorbar
[val, idx1] = min(z);
[val, idx2] = min(val);
idx1 = idx1(idx2);
hold on
scatter(x1(idx2), x2(idx1), 'r*', 'DisplayName', ...
    sprintf('Minimum Point z = %0.2f \nat x = [%0.2f, %0.2f]', val, x1(idx2), x2(idx1))
    'linewidth', 1.5)
legend
hold off
xlabel('x_1'); ylabel('x_2')
```




```
% exportgraphics(gcf, 'hw3p1a.png', 'Resolution', 200)
```

Part c

```
syms x_1 x_2
f_sym = (1-x_1)^2 + 100*(x_2-x_1^2)^2
```

f_sym = $(x_1 - 1)^2 + 100\ \left(x_2 - {x_1}^2\right)^2$

```
eqs = [diff(f_sym, x_1) == 0; diff(f_sym, x_2) == 0]
```

eqs =

$$\begin{pmatrix} 2\,x_1 - 400\,x_1\ \left(x_2 - {x_1}^2\right) - 2 = 0 \\ 200\,x_2 - 200\,{x_1}^2 = 0 \end{pmatrix}$$

```
sol = solve(eqs); fprintf('Derivatives = 0 \nat x = [%0.2f, %0.2f]', sol.x_1, sol.x_2)
```

```
Derivatives = 0
at x = [1.00, 1.00]
```

2nd Order Necessary and Sufficient Condition

Necessary: $\nabla f(x_*) = \vec{0}$

Sufficient: $\nabla^2 f(x_*)$ is positive definite

```
grad = [diff(f_sym, x_1); diff(f_sym, x_2)]
```

grad =

$$\begin{pmatrix} 2x_1 - 400x_1\left(x_2 - x_1^2\right) - 2 \\ 200x_2 - 200x_1^2 \end{pmatrix}$$

```
f_grad_ = matlabFunction(grad);
fprintf('Norm of the first gradient of f is %0.2f,\n thus the necessary condition is sa
```

```
Norm of the first gradient of f is 0.00,
 thus the necessary condition is satisfied
```

```
grad = [diff(grad(1), x_1) diff(grad(1), x_2);
    diff(grad(2), x_1) diff(grad(2), x_2)]
```

grad =

$$\begin{pmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}$$

```
f_grad2_ = matlabFunction(grad);
[A, flag] = chol(f_grad2_(sol.x_1, sol.x_2))
```

A =

$$\begin{pmatrix} \sqrt{802} & -\frac{200\sqrt{802}}{401} \\ 0 & \frac{10\sqrt{2}\sqrt{401}}{401} \end{pmatrix}$$

```
flag = 0
```

```
fprintf('As the Cholesky factorization was successful, and returned a flag of %i\n the
```

```
As the Cholesky factorization was successful, and returned a flag of 0
 the sufficient condition was satisfied
```

Problem 2

```
% SETUP
s = [-1 1].';
x0 = [4 -1].';
t0 = 0.1;
f_grad = @(x) f_grad_(x(1), x(2));
f_grad2 = @(x) f_grad2_(x(1), x(2));

% BRACKETING MINIMUM
[pts, line, iters] = lineSearch(x0, s, f, t0, 4, 2)
```

pts = 1×3 struct

Fields	x	z
1	[1.7000;...	253.3000
2	[1.3000;...	0.1000
3	[0.5000;...	506.5000

line = 2×13

```
    4.0000    3.9000    3.8000    3.7000    3.5000    3.3000    3.1000    2.9000 ···
   -1.0000   -0.9000   -0.8000   -0.7000   -0.5000   -0.3000   -0.1000    0.1000
iters = 13
```

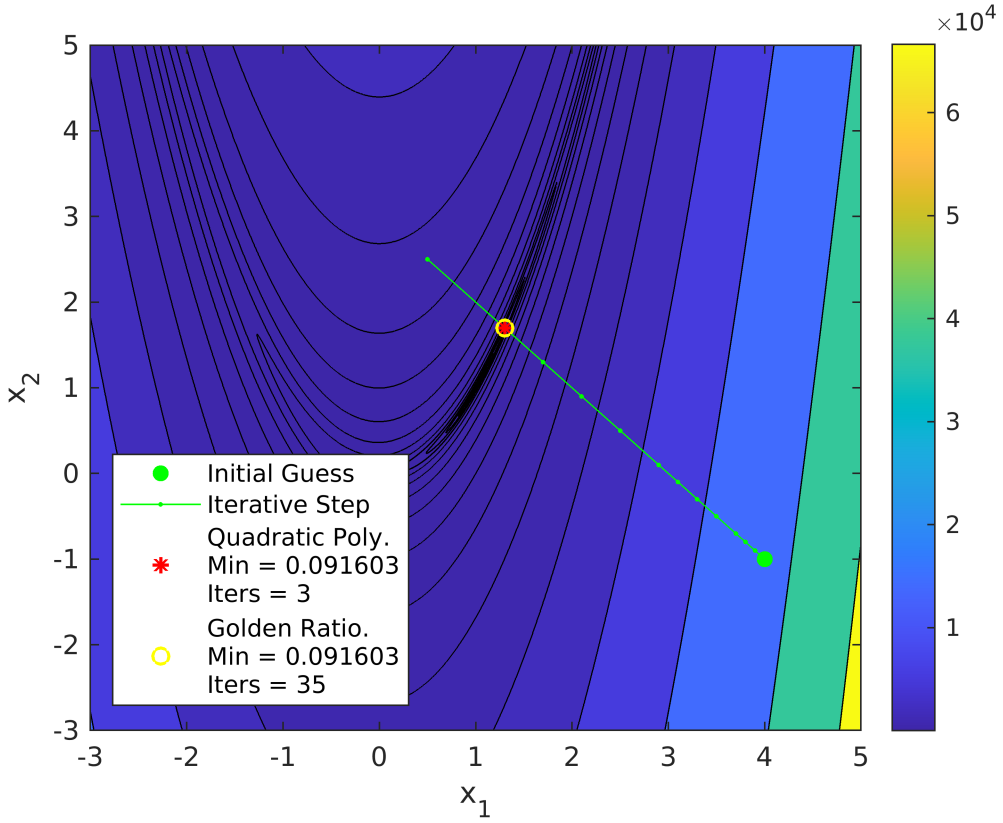
```
% QUADRATIC POLYNOMIAL LINE SEARCH
[x_min_quad, z_min_quad, iter_quad] = quadMin([pts(:).x], [pts(:).z], s, f, f_grad, f_g
```

```
x_min_quad = 2×1
    1.3025
    1.6975
z_min_quad = 0.0916
iter_quad = 3
```

```
% GOLDEN RATIO LINE SEARCH
[x_min_gr, z_min_gr, iter_gr, allX] = grMin([pts(:).x], [pts(:).z], s, f)
```

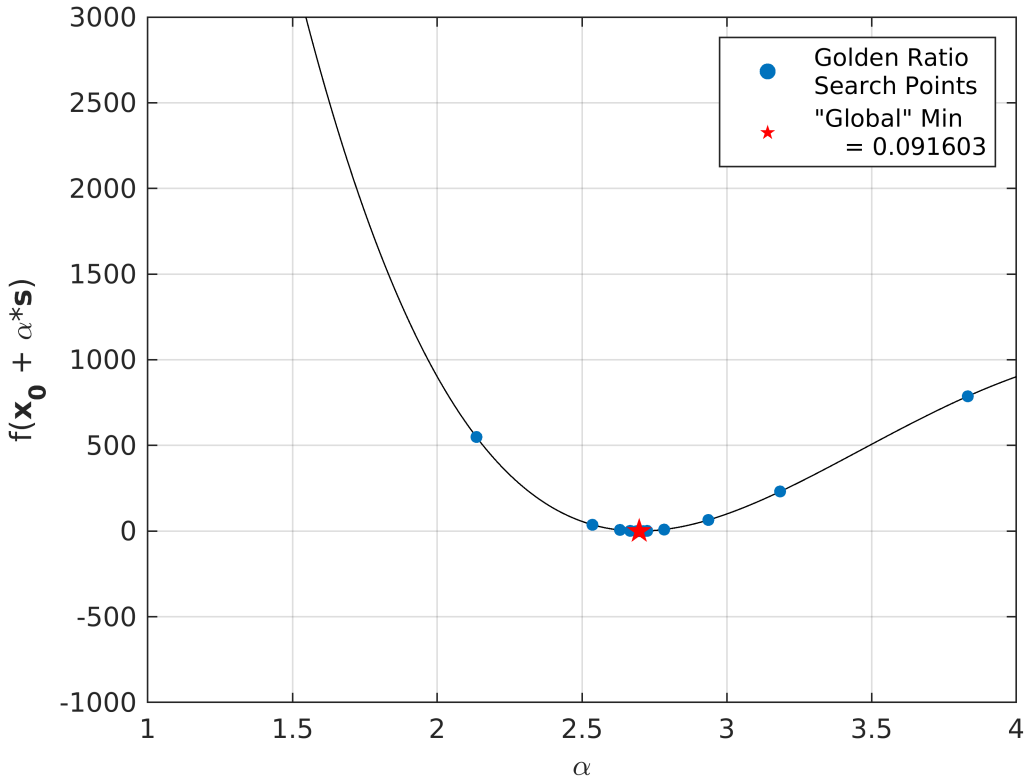
```
x_min_gr = 2×1
    1.3025
    1.6975
z_min_gr = 0.0916
iter_gr = 35
allX = 2×39
    1.3025    1.3025    1.3025    1.3025    1.3025    1.3025    1.3025    1.3025 ···
    1.6975    1.6975    1.6975    1.6975    1.6975    1.6975    1.6975    1.6975
```

```
% PLOTTING
contourf(x1, x2, z, [logspace(-4, 5, 22), max(max(z))*0.85], 'HandleVisibility',"off")
colorbar
[val, idx1] = min(z);
[val, idx2] = min(val);
idx1 = idx1(idx2);
hold on
% scatter(x1(idx2), x2(idx1), 'r*', 'DisplayName', ...
%     sprintf('Minimum Point z = %0.2f \nat x = [%0.2f, %0.2f]', val, x1(idx2), x2(idx1
%     'linewidth', 1.5)
scatter(line(1, 1), line(2, 1), 'go', 'filled', 'DisplayName', 'Initial Guess')
plot(line(1, :), line(2, :), 'g.-', 'DisplayName', 'Iterative Step')
scatter(x_min_quad(1), x_min_quad(2), 'r*', 'DisplayName', ...
    sprintf('Quadratic Poly.\nMin = %0.6f\nIters = %i', z_min_quad, iter_quad), ...
    'linewidth', 1.5)
scatter(x_min_gr(1), x_min_gr(2), 'yo', 'DisplayName', ...
    sprintf('Golden Ratio.\nMin = %0.6f\nIters = %i', z_min_gr, iter_gr), ...
    'linewidth', 1.5)
lgn = legend; lgn.Location = 'southwest';
% axis equal
hold off
xlabel('x_1'); ylabel('x_2')
```




```
% exportgraphics(gcf, 'hw3p2.png', 'Resolution', 200)

% ADDITIONAL PLOT
line = linspace(0, 7, 50000);
y = zeros(length(line), 1);
for i = 1:length(y); y(i) = f(x0 + line(i).*s); end
plot(line(1:50:end), y(1:50:end), 'k-', 'HandleVisibility', "off")
hold on
[val, idx] = min(y);
ylim([-1000 3000])
legend
ylabel('f({\bfx_0} + \alpha*{\bfs})', "Interpreter","tex")
xlabel('\alpha')
dist = zeros(size(allX, 2), 1); ydist = dist;
for i = 1:size(allX, 2)
    dist(i) = norm(allX(:, i)-x0)-1.1173;
    ydist(i) = f(x0 + dist(i).*s);%*1.85;
end
scatter(dist, ydist, 20, 'o', 'filled', 'MarkerFaceColor', [0 0.4470  0.7410],...
    'DisplayName', sprintf('Golden Ratio \nSearch Points'))
scatter(line(idx), y(idx), 100, 'rp', 'filled',...
    'DisplayName', sprintf('"Global" Min\n    = %0.6f', val))
xlim([1 4])
grid on
```




```
% CREATING TABLE OF RESULTS
% tMult = [1.1 1.25 1.5 2 2.5 3 5];
% tSteps = [2 4 8 16];
% C = cell(length(tSteps), length(tMult));
% for i = 1:length(tSteps)
%     for j = 1:length(tMult)
%         % FINDING ITERATIONS
%         [pts, ~, iters_line] = lineSearch(x0, s, f, t0, tSteps(i), tMult(j));
%         [~, ~, iter_quad] = quadMin([pts(:).x], [pts(:).z], s, f, f_grad, f_grad2);
%         [~, ~, iter_gr] = grMin([pts(:).x], [pts(:).z], s, f);
%
%         % COLLECTING
%         C{i, j} = [iters_line, iter_quad, iter_gr];
%     end
% end
% C = cat(2, num2cell(tSteps.'), C);
% colNames = {'NumSteps', '1.1 Multiplier', '1.25 Multiplier', '1.5 Multiplier', '2.0 M
% rowNames = {'2 Steps', '4 Steps', '8 Steps', '16 Steps'};
% dimNames = ["Num Steps","Multiplier"];
% T = cell2table(C(:, 2:end), 'DimensionNames', dimNames, 'VariableNames', colNames(2:e
```

```
function [pts, x, iterTot] = lineSearch(x0, s, f, t0, steps, stepMod)
    % SETUP
    z = f(x0);
    x = zeros(length(x0), 1);
    x(:, end) = x0;           % Enforcing column vectors
```

```
    t = t0;
    iter = 1; iterTot = 1;

    % ITERATING UNTIL MINIMUM BRACKETED
    while length(z) < 2 || z(end) < z(end-1)
        % INCREASING STEP SIZE IF NOT BRACKETING
        if iter == steps
            t = stepMod*t;
            iter = 0;
        end

        % STEPPING AND EVALUATING
        x = cat(2, x, x(:, end) + t*s);
        z(end+1) = f(x(:, end));
        iter = iter+1;
        iterTot = iterTot+1;

        if iter == 31
            pts = [];
            warning("lineSearch() ran out of iterations")
            return
        end
    end

    % OUTPUTTING POINTS
    pts = struct('x', x(:, end-2), 'z', z(end-2));
    pts(end+1) = struct('x', x(:, end-1), 'z', z(end-1));
    pts(end+1) = struct('x', x(:, end), 'z', z(end));
end

function [xmin, zmin, iters] = quadMin(x, z, s, f, f_grad, f_grad2)
    % SETUP
    alpha = inf;
    iters = 0;

    % ITERATING
    while abs(alpha) > 1e-8
        [~, idx] = min(z);

        % FINDING STEP SIZE
        alpha = -(f_grad(x(:, idx)).'*s) / (s.' * f_grad2(x(:, idx)) * s);

        % STEPPING
        x = cat(2, x, x(:, idx) + alpha*s);
        z(end+1) = f(x(:, end));
        iters = iters+1;
        if iters > 51; break; end
    end

    % OUTPUTTING
    [~, idx] = min(z);
    xmin = x(:, idx);
    zmin = z(idx);
end
```

```
function [xmin, zmin, iters, allX] = grMin(x0, z0, s, f)
    % SETUP
    alpha = (3 - sqrt(5))/2;   % Step Size

    xL = x0(:, 1);             % Creating Points
    xR = x0(:, 3);             % |
    x1 = xL + alpha*(xR - xL); % |
    x2 = xR - alpha*(xR - xL); % #

    fL = z0(1);                % Evaluating at Points
    f1 = f(x1);                % |
    f2 = f(x2);                % |
    fR = z0(3);                % #

    X = [xL x1 x2 xR];         % Creating Test Matrix
    dx = inf;                  % Initializing Error
    iters = 0;

    % ITERATING
    allX = X;
    while abs(dx) > 1e-8
        if f1 > f2
            % MOVING BOUNDS
            xL = x1; fL = f1;
            x1 = x2; f1 = f2;

            % RE-EVALUTING POINT
            x2 = xR - alpha*(xR - xL);
            allX = cat(2, x2, allX);
            f2 = f(x2);
        else
            % MOVING BOUINDS
            xR = x2; fR = f2;
            x2 = x1; f2 = f1;

            % RE-EVALUTING POINT
            x1 = xL + alpha*(xR - xL);
            allX = cat(2, x1, allX);
            f1 = f(x1);
        end

        Xnew = [xL, x1, x2, xR];
        dx = norm(Xnew, 'fro') - norm(X, 'fro');
        X = Xnew;
        iters = iters+1;
        if iters > 51; break; end
    end

    % FINDING RESULTS
    [zmin, idx] = min([fL, f1, f2, fR]);
    xmin = X(:, idx);
end
```

```
clear; clc;
```

Problem 3 Steepest Descent -> BFGS

```
% DERIVING GRADIENTS
syms x_1 x_2
u = (1/2)*(x_1^2 + x_2^2 - 25);
f_sym = exp(u^2) + sin(4*x_1 - 3*x_2)^4 + 0.5*(2*x_1 + x_2 - 10)^2;
% f_sym = (1-x_1)^2 + 100*(x_2-x_1^2)^2;
f = matlabFunction(f_sym); f = @(x) f(x(1), x(2));

grad = [diff(f_sym, x_1); diff(f_sym, x_2)];
f_grad = matlabFunction(grad); f_grad = @(x) f_grad(x(1), x(2));

grad = [diff(grad(1), x_1) diff(grad(1), x_2);
        diff(grad(2), x_1) diff(grad(2), x_2)];
f_grad2 = matlabFunction(grad); f_grad2 = @(x) f_grad2(x(1), x(2));

% SETUP
x0 = [4; -1]; x = x0;
% x0 = [2; 3]; x = x0;
dx = Inf;
fCalls = [0 0 0];
t0 = 0.1;
i = 0;
tic
filename = 'hw3p3_2SD.png';
while dx > 1e-8
    i = i+1;
    % FINDING DIRECTION
    if i == 1; dir0 = []; else; dir0 = dir(i-1); end
%     dir(i) = steepestDescent(x, f_grad); % Steepest Descent
%     dir(i) = fletchReev(x, f_grad, dir0); % Fletcher-Reeves
%     dir(i) = polakRibiere(x, f_grad, dir0); % Polak Ribiere
    dir(i) = BFGS(x, f_grad, dir0); % Broyden-Fletcher-Goldfarb-Shanno
    s = dir(i).s;
    fCalls = fCalls + dir(i).fCalls;

    % LINE SEARCHING
    lin(i) = lineSearch(x, s, f, t0, 4, 2);
    fCalls = fCalls + lin(i).fCalls;

    % MINIMIZING
    mini(i) = grMin([lin(i).pts(:).x], f);
%     mini(i) = quadMin([lin(i).pts(:).x], s, f, f_grad, f_grad2);
    fCalls = fCalls + mini(i).fCalls;

    % ALIGNING NEW MINIMUM
    %     if norm(x - mini(i).xmin) < 1e-8; warning('WARNING: dx = 0'); break; end
    dx = norm(x - mini(i).xmin);
    x = mini(i).xmin;
%     grad = f_grad(x);
%     fCalls(2) = fCalls(2) + 1;
```

```
    if i == 1000; break; end
end
fCalls(2) = fCalls(2) + 1;
if norm(f_grad(x)) > 1e-4
    warning('WARNING: May not have found minimum');
end
time = toc;

% PLOTTING
x1_space = -5:0.1:5;
x2_space = -5:0.1:5;
z_space = zeros(length(x1_space));
for i = 1:length(x1_space)
    for j = 1:length(x1_space)
        z_space(j, i) = f([x1_space(i), x2_space(j)]);
    end
end

% Contour
contourf(x1_space, x2_space, z_space, [logspace(0, 24, 40)], ...
    'HandleVisibility',"off")%, 'LineStyle',"none")
set(gca, 'ColorScale', 'log');
caxis([2, 1e24])
hold on

% Start Point
scatter(x0(1), x0(2), 50, 'ro', 'filled', 'DisplayName', 'Starting Point')

% Creating Initial Plots and Legend Entries
x = lin(1).allPts;
scatter(x(1, :), x(2, :), 20, 'g.', 'DisplayName', 'Line Searches')
xmin = mini(1).xmin;
scatter(xmin(1), xmin(2), 30, 'rx', 'DisplayName', 'Line Minimum', 'linewidth', 5)

% Rest of Line Searches
l = length(lin); if l > 100; l = 100; end
for i = 2:l
    % Line Searches
    x = lin(i).allPts;
    scatter(x(1, :), x(2, :), 20, 'g.', 'HandleVisibility', 'off')

    % Minimums
    xmin = mini(i).xmin;
    scatter(xmin(1), xmin(2), 30, 'rx', 'HandleVisibility', 'off', 'linewidth', 5)
end

% Final Point
scatter(xmin(1), xmin(2), 200, 'yp', 'filled', 'linewidth', 1, ...
    'DisplayName', sprintf('Found Local Min:\nx = [%0.8f,\n       %0.8f]', xmin(1), xmin(2)))
hold off
xticks(-5:5)
xlim([-5 5])
ylim([-5 5])
C = colorbar('peer', gca, "eastoutside", 'Ticks', logspace(1, 25, 7));
```

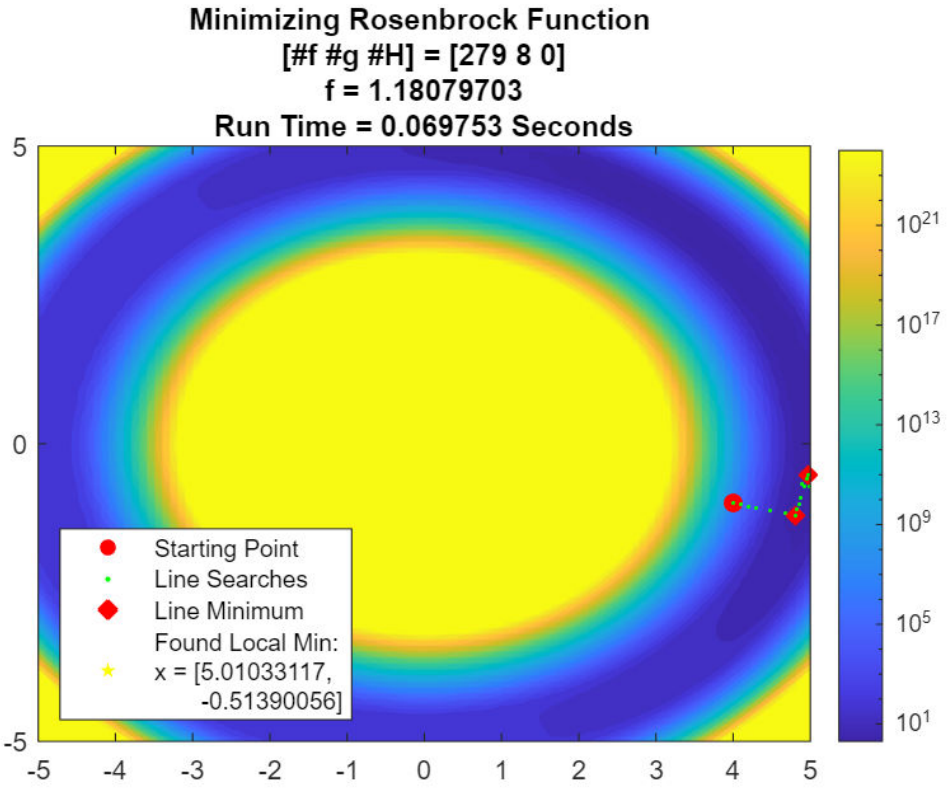
```
% set(C, )
% legend('Location', 'southwest')
% title(sprintf('Minimizing Rosenbrock Function \n[#f #g #H] = [%i %i %i]\nf = %0.8f\nRun Time
exportgraphics(gcf, filename, 'Resolution', 200)
```

```
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
```




```
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
```

```
function out = BFGS(x, g, prev)
    if isempty(prev)
        gk1 = g(x);
        Qk1 = eye(length(x));
    else
        % EVALUTING NEW TERMS
        gk1 = g(x);
        Qk = prev.Q;
```

```
        % CREATING CONSTANTS
        p = x - prev.x;
        y = gk1 - prev.g;
        sig = p.' * y;
        tau = y.' * Qk * y;
        A = Qk * y * p.';

        % COMPUTING UPDATE
        dQ = ((sig + tau)/sig^2)*(p * p.') - (1/sig)*(A + A.');
        Qk1 = Qk + dQ; % Creating approximated Hessian
    end

    out.s = -Qk1*gk1; out.s = out.s/norm(out.s);
    out.x = x;
    out.fCalls = [0 1 0];
    out.g = gk1;
    out.Q = Qk1;
end

function out = polakRibiere(x, g, prev)
    if isempty(prev)
        g1 = -g(x);
        s = g1;
    else
        g0 = prev.g;
        s = prev.s;
        g1 = -g(x);
        y1 = g1 - g0;
        beta = (g1.' * y1) / norm(g0)^2;
        s = g1 + beta*s;
    end

    out.s = s/norm(s);
    out.fCalls = [0 1 0];
    out.g = g1;
end

function out = fletchReev(x, g, prev)
    if isempty(prev)
        g1 = -g(x);
        s = g1;
    else
        g0 = prev.g;
        s = prev.s;
        g1 = -g(x);
        beta = norm(g1)^2 / norm(g0)^2;
        s = g1 + beta*s;
    end

    out.s = s/norm(s);
    out.fCalls = [0 1 0];
    out.g = g1;
end
```

```
function out = steepestDescent(x, g)
    s = zeros(length(x), 1);
    s = -g(x);
    out.s = s/norm(s);
    out.fCalls = [0 1 0];
end

function out = lineSearch(x0, s, f, t0, steps, stepMod)
    % SETUP
    fCalls = 0;
    z = f(x0); fCalls = fCalls+1;
    x = zeros(length(x0), 1);
    x(:, end) = x0;           % Enforcing column vectors
    t = t0;
    iter = 1; iterTot = 1;

    % ITERATING UNTIL MINIMUM BRACKETED
    while length(z) < 3 || z(end) < z(end-1)
        % INCREASING STEP SIZE IF NOT BRACKETING
        if iter == steps
            t = stepMod*t;
            iter = 0;
        end

        % STEPPING AND EVALUATING
        x = cat(2, x, x(:, end) + t*s);
        z(end+1) = f(x(:, end)); fCalls = fCalls+1;
        iter = iter+1;
        iterTot = iterTot+1;

        if iter == 31
            pts = [];
            warning("lineSearch() ran out of iterations")
            return
        end
    end

    % OUTPUTTING POINTS
    pts = struct('x', x(:, end-2), 'z', z(end-2));
    pts(end+1) = struct('x', x(:, end-1), 'z', z(end-1));
    pts(end+1) = struct('x', x(:, end), 'z', z(end));
    out.pts = pts;
    out.iters = iterTot;
    out.allPts = x;
    out.fCalls = [fCalls 0 0];
end
function out = quadMin(x, s, f, f_grad, f_grad2)
    % SETUP
    alpha = inf;
    iters = 0;
    fCalls = 3;
    dfCalls = 0;
    d2fCalls = 0;
```

```
    % ITERATING
    for i = 1:3; z(i) = f(x(:, i)); end
    while abs(alpha) > 1e-8
        [~, idx] = min(z);

        % FINDING STEP SIZE
        alpha = -(f_grad(x(:, idx)).'*s) / (s.' * f_grad2(x(:, idx)) * s);
        dfCalls = dfCalls+1;
        d2fCalls = d2fCalls+1;

        % STEPPING
        x = cat(2, x, x(:, idx) + alpha*s);
        z(end+1) = f(x(:, end));
        fCalls = dfCalls+1;
        iters = iters+1;
        if iters > 51; break; end
    end

    % OUTPUTTING
    [~, idx] = min(z);
    out.xmin = x(:, idx);
    out.zmin = z(idx);
    out.iters = iters;
    out.fCalls = [fCalls, dfCalls, d2fCalls];
end

function out = grMin(x0, f)
    % SETUP
    alpha = (3 - sqrt(5))/2;   % Step Size

    xL = x0(:, 1);             % Creating Points
    xR = x0(:, 3);             % |
    x1 = xL + alpha*(xR - xL); % |
    x2 = xR - alpha*(xR - xL); % #

    fL = f(xL);                % Evaluating at Points
    f1 = f(x1);                % |
    f2 = f(x2);                % |
    fR = f(xR);                % #

    X = [xL x1 x2 xR];         % Creating Test Matrix
    dx = inf;                  % Initializing Error
    iters = 0;
    fCalls = 4;

    % ITERATING
    allX = X;
    while abs(dx) > 1e-8
        if f1 > f2
            % MOVING BOUNDS
            xL = x1; fL = f1;
            x1 = x2; f1 = f2;

            % RE-EVALUTING POINT
```

```
            x2 = xR - alpha*(xR - xL);
            allX = cat(2, x2, allX);
            f2 = f(x2);
        else
            % MOVING BOUINDS
            xR = x2; fR = f2;
            x2 = x1; f2 = f1;

            % RE-EVALUTING POINT
            x1 = xL + alpha*(xR - xL);
            allX = cat(2, x1, allX);
            f1 = f(x1);
        end

        fCalls = fCalls+1;
        Xnew = [xL, x1, x2, xR];
        dx = norm(Xnew, 'fro') - norm(X, 'fro');
        X = Xnew;
        iters = iters+1;
        if iters > 51; break; end
    end

    % FINDING RESULTS
    [out.zmin, idx] = min([fL, f1, f2, fR]);
    out.xmin = X(:, idx);
    out.allPts = allX;
    out.iters = iters;
    out.fCalls = [fCalls 0 0];
end
```

```
clear; clc;
```

Problem 3 Levenberg-Marquardt

```
% DERIVING GRADIENTS
syms x_1 x_2
% f_sym = (1-x_1)^2 + 100*(x_2-x_1^2)^2;
u = (1/2)*(x_1^2 + x_2^2 - 25);
f_sym = exp(u^2) + sin(4*x_1 - 3*x_2)^4 + 0.5*(2*x_1 + x_2 - 10)^2
```

f_sym =

$$e^{\left(\frac{x_1^2}{2}+\frac{x_2^2}{2}-\frac{25}{2}\right)^2}+\frac{(2x_1+x_2-10)^2}{2}+\sin(4x_1-3x_2)^4$$

```
f = matlabFunction(f_sym); f = @(x) f(x(1), x(2));

grad = [diff(f_sym, x_1); diff(f_sym, x_2)]
```

grad =

$$\begin{pmatrix} 16\cos(4x_1-3x_2)\,\sigma_2+4x_1+2x_2+2x_1\,e^{\sigma_1^2}\,\sigma_1-20 \\ -12\cos(4x_1-3x_2)\,\sigma_2+2x_1+x_2+2x_2\,e^{\sigma_1^2}\,\sigma_1-10 \end{pmatrix}$$

where

$$\sigma_1=\frac{x_1^2}{2}+\frac{x_2^2}{2}-\frac{25}{2}$$

$$\sigma_2=\sin(4x_1-3x_2)^3$$

```
f_grad = matlabFunction(grad); f_grad = @(x) f_grad(x(1), x(2));

grad = [diff(grad(1), x_1) diff(grad(1), x_2);
        diff(grad(2), x_1) diff(grad(2), x_2)]
```

grad =

$$\begin{pmatrix} 192\,\sigma_5\,\sigma_4+\sigma_2+2\,x_1{}^2\,e^{\sigma_6}-64\,\sigma_3+4\,x_1{}^2\,e^{\sigma_6}\,\sigma_6+4 & \sigma_1 \\ \sigma_1 & 108\,\sigma_5\,\sigma_4+\sigma_2+2\,x_2{}^2\,e^{\sigma_6}-36\,\sigma_3+4\,x_2{}^2\,e^{\sigma_6}\,\sigma_6+1 \end{pmatrix}$$

where

$$\sigma_1 = 48\,\sigma_3-144\,\sigma_5\,\sigma_4+2\,x_1\,x_2\,e^{\sigma_6}+4\,x_1\,x_2\,e^{\sigma_6}\,\sigma_6+2$$

$$\sigma_2 = 2\,e^{\sigma_6}\left(\frac{x_1{}^2}{2}+\frac{x_2{}^2}{2}-\frac{25}{2}\right)$$

$$\sigma_3 = \sin(4\,x_1-3\,x_2)^4$$

$$\sigma_4 = \sin(4\,x_1-3\,x_2)^2$$

$$\sigma_5 = \cos(4\,x_1-3\,x_2)^2$$

$$\sigma_6 = \left(\frac{x_1{}^2}{2}+\frac{x_2{}^2}{2}-\frac{25}{2}\right)^2$$

```
f_grad2 = matlabFunction(grad); f_grad2 = @(x) f_grad2(x(1), x(2));

% SETUP
% x0 = [4; -1]; x = x0;
x0 = [2; 3]; x = x0;
xout = x0;
dx = Inf;
fCalls = [0 0 0];
t0 = 0.1;
iters = 0;
tic
filename = 'hw3p3_4LM.png';

% START CONDITIONS
lambda = 1e-2; gamma = 1;
lambdaList = lambda;
J = f(x); Jlast = J;
g = f_grad(x);
H = f_grad2(x);
fCalls = fCalls + [1 1 1];
lamMin = min(eig(H));

% ITERATING
while ~(norm(g) < 1e-8 && lamMin > 0)

    % FINDING TEST POINT
    xtest = x + (H + lambda*eye(length(x))) \ -g;
```

```
    % TESTING
    J = f(xtest);
    fCalls(1) = fCalls(1) + 1;
    if J > Jlast
        lambda = lambda*gamma;
%         if lambda < 1 || lambda > 1e12; break; end
        iters = iters + 1;

        if iters == 1000; break; else; continue; end
    end

    % ACCEPTING STEP
    x = xtest;
    Jlast = J;
    lambda = lambda / gamma^2;

%     UPDATING GRADIENTS
%     J = f(x);
    g = f_grad(x);
    H = f_grad2(x);
    fCalls = fCalls + [0 1 1];
    lamMin = min(eig(H));
    xout = cat(2, xout, x);

%     lambda = max(max(-lamMin + 1e-8, 1e3), lambda);

    iters = iters+1;
    lambdaList(end+1) = lambda;
    if iters == 1000; break; end
    if norm(g) < 1e-8
        xold = x;
        xnew = x - 1e-4*(g/norm(g));
        if f(xnew) < f(xold)
            x = xnew;
            g = f_grad(x);
            H = f_grad2(x);
        end
    end
end
time = toc;

% PLOTTING
x1_space = -5:0.1:5;
x2_space = -5:0.1:5;
z_space = zeros(length(x1_space));
for i = 1:length(x1_space)
    for j = 1:length(x1_space)
        z_space(j, i) = f([x1_space(i), x2_space(j)]);
    end
end

x
```

```
x = 2×1
```

```
    2.7473
    4.1925
```

```
f(x)
```

```
ans = 2.0522
```

```
% Contour
contourf(x1_space, x2_space, z_space, [logspace(0, 24, 40)], ...
    'HandleVisibility',"off")%, 'LineStyle',"none")
set(gcf,'Visible','on')
hold on
set(gca, 'ColorScale', 'log');
caxis([2, 1e24])
scatter(xout(1, :), xout(2, :), 'ro', 'filled', 'DisplayName', ...
    ['LM Steps', newline, '\lambda_0 = ', sprintf('%0.6g', lambdaList(1)), newline, '\gamma =
plot(xout(1, :), xout(2, :), 'r--', 'HandleVisibility',"off")
scatter(xout(1, 1), xout(2, 1), 50, 'go', 'filled', 'DisplayName',"Initial Position")
C = colorbar('peer', gca, "eastoutside", 'Ticks', logspace(1, 25, 7));
scatter(x(1, end), x(2, end), 300, 'yp', 'filled', 'DisplayName', sprintf('x_{min} =\n [%0.8f\n
hold off
legend('Location', 'southwest')
title(sprintf('Minimizing Rosenbrock Function \n[#f #g #H] = [%i %i %i]\nf = %0.8g\nRun Time =
exportgraphics(gcf, filename, 'Resolution', 200);
```

```
% clear; clc
```

Problem 4

```
% SETTING INPUTS
global EoM cost
J2 = 0.1;
R = 0.9;
mu = 1;
ToF = 83;
N = 3;
r0 = [1 0 0].';
rf = [-2.891216226058043 -1.254145998446107 0].';
cost = @(rstar, Nstar) (rstar - rf).' * (rstar - rf) + 1e10 * (N - Nstar)^2;

% INITIAL GUESS
ICnum = 1;
v0 = [0.05 1.305 0;
      -0.2 1.27 0];
v0 = v0(ICnum, :).';

% FINDING EOMs
syms x y z
F1 = x^2 + y^2 + z^2;
U = mu/sqrt(F1) * (1 - R^2/F1 * J2 * ((3*z^2)/(2*F1) - 1/2));
rddot = matlabFunction([diff(U, x); diff(U, y); diff(U, z)]);
EoM = @(t, y) [y(4:6); rddot(y(1), y(2), y(3))];

% TEST PROPAGATING
opts = odeset('RelTol', 1e-6, 'AbsTol', 1e-6, 'Events', @eventFcn);
[t, Y, te, ye, ie] = ode45(EoM, [0 ToF], [r0; v0], opts);
Nstar = length(te) - 1;
rstar = Y(end, 1:3).';

% INTERPRETTING
plot3(Y(:, 1), Y(:, 2), Y(:, 3), 'DisplayName', 'Trajectory')
grid on
axis equal
hold on
scatter3(0, 0, 0, 50, 'ko', 'filled', 'DisplayName', 'Center Body')
scatter3(Y(1, 1), Y(1, 2), Y(1, 3), 50, 'bo', 'filled', 'DisplayName', 'Initial Position')
scatter3(Y(end, 1), Y(end, 2), Y(end, 3), 50, 'bs', 'filled', 'DisplayName', 'Final Position')
scatter3(rf(1), rf(2), rf(3), 50, 'rp', 'filled', 'DisplayName', 'Target Final State')
hold off
xlabel('x (LU)'); ylabel('y (LU)'); zlabel('z (LU)');
title(sprintf('Propagating Initial State\n|r*-r_f| = %0.4g :: N* = %i :: Performance Index = %0
legend('Location','north', 'NumColumns',3)
exportgraphics(gcf, sprintf('hw3p4_%i_Initial.png', ICnum), 'Resolution', 200);
```

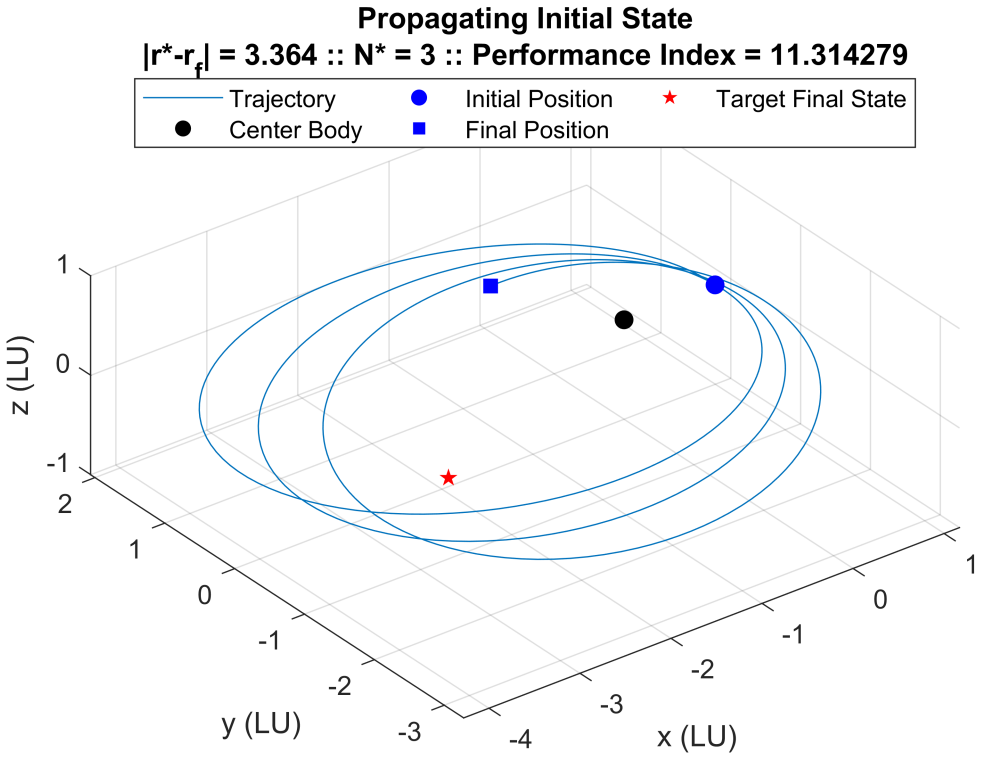
```
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
```

```
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
```




```
% CREATING CALLS FOR MINIMIZATION
func = @(x) propWithCost(x, r0, ToF); func(v0);
grad = @(x) costGradient(x, r0, ToF); grad(v0);

% ITERATION SETUP
dx = Inf;
i = 0;
fCalls = [0 0 0];
x = v0;
t0 = 0.01;

% ITERATING
tic
while dx > 1e-8
    i = i+1;
    % FINDING DIRECTION
    if i == 1; dir0 = []; else; dir0 = dir(i-1); end
    dir(i) = BFGS(x, grad, dir0); % Broyden-Fletcher-Goldfarb-Shanno
    s = dir(i).s;
    fCalls = fCalls + dir(i).fCalls;
```

```
    % LINE SEARCHING
    lin(i) = lineSearch(x, s, func, t0, 4, 2);
    fCalls = fCalls + lin(i).fCalls;

    % MINIMIZING
    mini(i) = grMin([lin(i).pts(:).x], func);
    fCalls = fCalls + mini(i).fCalls;

    % ALIGNING NEW MINIMUM
    %     if norm(x - mini(i).xmin) < 1e-8; warning('WARNING: dx = 0'); break; end
    dx = norm(x - mini(i).xmin);
    x = mini(i).xmin;
%     grad = f_grad(x);
%     fCalls(2) = fCalls(2) + 1;
    if i == 1000; break; end
end
time = toc;

% PLOTTING PATH
% Contour
% load hw3p4.mat
%{
contourf(x1_space, x2_space, z_space, [logspace(-4, 1.5, 20)], ...
    'HandleVisibility',"off", 'LineStyle',"none")
colormap turbo
hold on

% Start Point
scatter(v0(1), v0(2), 50, 'ro', 'filled', 'DisplayName', 'Starting Point')
stringMin = cat(2, v0, [mini(:).xmin]);
plot(stringMin(1, :), stringMin(2, :), 'k--', 'HandleVisibility',"off")

% Creating Initial Plots and Legend Entries
x = lin(1).allPts;
scatter(x(1, :), x(2, :), 20, 'g.', 'DisplayName', 'Line Searches')
xmin = mini(1).xmin;
scatter(xmin(1), xmin(2), 30, 'rx', 'DisplayName', 'Line Minimum', 'linewidth', 5)

% Rest of Line Searches
l = length(lin); if l > 100; l = 100; end
for i = 2:l
    % Line Searches
    x = lin(i).allPts;
    scatter(x(1, :), x(2, :), 20, 'c.', 'HandleVisibility', 'off')
end
for i = 2:l
    % Minimums
    xmin = mini(i).xmin;
    scatter(xmin(1), xmin(2), 30, 'rx', 'HandleVisibility', 'off', 'linewidth', 5)
end

% Final Point
scatter(xmin(1), xmin(2), 200, 'yp', 'filled', 'linewidth', 1, ...
```

```
    'DisplayName', sprintf('Found Local Min:\nx = [%0.8f,\n       %0.8f]', xmin(1), xmin(2)))
hold off
legend('Location','north', 'NumColumns',2)
C = colorbar;
C.Label.String = 'Performance Index';
C.Label.FontSize = 11;
xlabel('v_x (LU/TU)'); ylabel('v_y (LU/TU)')
title(sprintf('Finding Minimum Using BFGS\nInitial Stride = %0.3g\n[#f #g] = [%i %i]\nRun Time
exportgraphics(gcf, sprintf('hw3p4_%i_MinSearch.png', ICnum), 'Resolution', 200);
%}

% PROPAGATING FOUND SOLUTION
opts = odeset('RelTol', 1e-6, 'AbsTol', 1e-6, 'Events', @eventFcn);
[t, Y, te, ye, ie] = ode45(EoM, [0 ToF], [r0; mini(i).xmin], opts);
Nstar = length(te) - 1;
rstar = Y(end, 1:3).';

% PLOTTING SOLUTION
plot3(Y(:, 1), Y(:, 2), Y(:, 3), 'DisplayName', 'Trajectory')
```

```
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
```

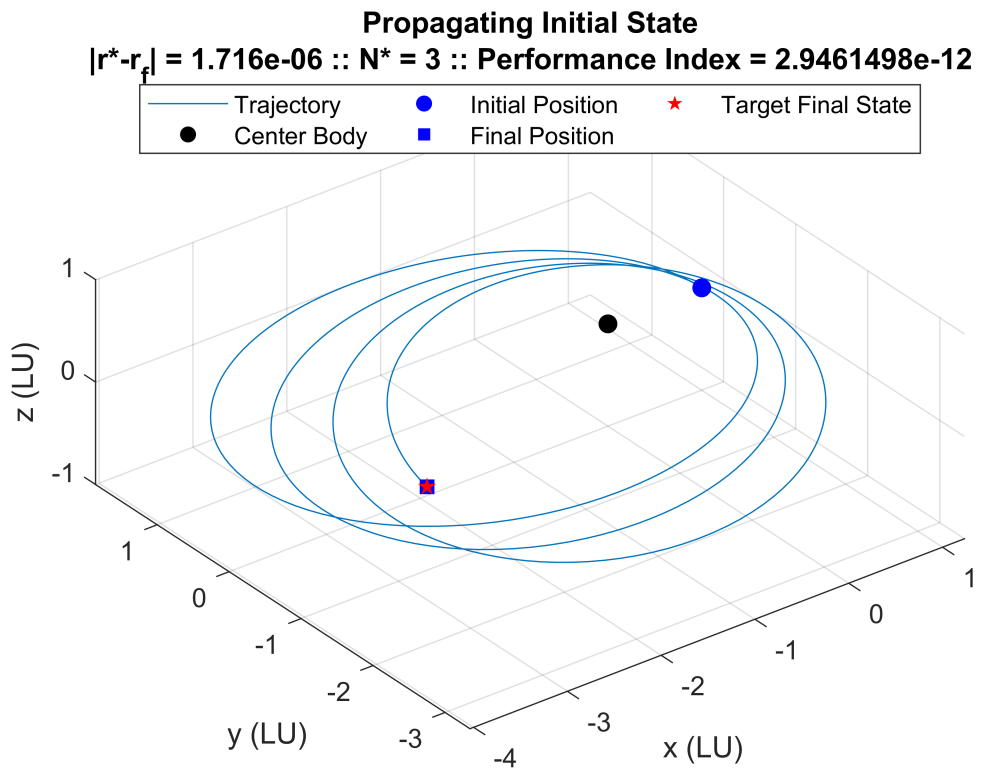
```
grid on
axis equal
hold on
scatter3(0, 0, 0, 50, 'ko', 'filled', 'DisplayName', 'Center Body')
scatter3(Y(1, 1), Y(1, 2), Y(1, 3), 50, 'bo', 'filled', 'DisplayName', 'Initial Position')
scatter3(Y(end, 1), Y(end, 2), Y(end, 3), 50, 'bs', 'filled', 'DisplayName', 'Final Position')
scatter3(rf(1), rf(2), rf(3), 50, 'rp', 'filled', 'DisplayName', 'Target Final State')
hold off
xlabel('x (LU)'); ylabel('y (LU)'); zlabel('z (LU)');
title(sprintf(['Propagating Initial State\n' ...
    '|r*-r_f| = %0.4g :: N* = %i :: ' ...
    'Performance Index = %0.8g'], ...
    norm(rstar - rf), Nstar, cost(rstar, Nstar)))
legend('Location','north', 'NumColumns',3)
exportgraphics(gcf, sprintf('hw3p4_%i_Optimal.png', ICnum), 'Resolution', 200);
```

```
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
```




```
Warning: Error updating Legend.

 Unrecognized method, property, or field 'InteractionsManager' for class
'matlab.graphics.primitive.canvas.JavaCanvas'.
```

```
%}
```

```
function [cond, flag, dir] = eventFcn(t, y)
    cond = [real(y(2)); norm(real(y(1:3))) - 0.1];
    flag = [false; true];
    dir = [1; 0];
end

function out = BFGS(x, g, prev)
    if isempty(prev)
        gk1 = g(x);
        Qk1 = eye(length(x));
    else
        % EVALUTING NEW TERMS
        gk1 = g(x);
        Qk = prev.Q;

        % CREATING CONSTANTS
        p = x - prev.x;
        y = gk1 - prev.g;
        sig = p.' * y;
        tau = y.' * Qk * y;
        A = Qk * y * p.';
```

```
        % COMPUTING UPDATE
        dQ = ((sig + tau)/sig^2)*(p * p.') - (1/sig)*(A + A.');
        Qk1 = Qk + dQ; % Creating approximated Hessian
    end

    out.s = -Qk1*gk1; out.s = out.s/norm(out.s);
    out.x = x;
    out.fCalls = [0 1 0];
    out.g = gk1;
    out.Q = Qk1;
end

function f = propWithCost(v, r0, ToF)
    global EoM cost
    opts = odeset('RelTol', 1e-6, 'AbsTol', 1e-6, 'Events', @eventFcn);
    [~, Y, te, ~, ~] = ode45(EoM, [0 ToF], [r0; v], opts);
    Nstar = length(te) - 1;
    rstar = Y(end, 1:3).';
    f = cost(rstar, Nstar);
end

function g = costGradient(v0, r0, ToF)
    g = zeros(length(v0), 1);
    h = 1e-20;
    for i = 1:length(v0)
        v = v0;
        v(i) = v(i) + h*1i;
        g(i) = imag(propWithCost(v, r0, ToF))/h;
    end
end

function out = lineSearch(x0, s, f, t0, steps, stepMod)
    % SETUP
    fCalls = 0;
    z = f(x0); fCalls = fCalls+1;
    x = zeros(length(x0), 1);
    x(:, end) = x0;           % Enforcing column vectors
    t = t0;
    iter = 1; iterTot = 1;

    % ITERATING UNTIL MINIMUM BRACKETED
    while length(z) < 3 || z(end) < z(end-1)
        % INCREASING STEP SIZE IF NOT BRACKETING
        if iter == steps
            t = stepMod*t;
            iter = 0;
        end

        % STEPPING AND EVALUATING
        x = cat(2, x, x(:, end) + t*s);
        z(end+1) = f(x(:, end)); fCalls = fCalls+1;
        iter = iter+1;
        iterTot = iterTot+1;
```

```
        if iter == 31
            pts = [];
            warning("lineSearch() ran out of iterations")
            return
        end
    end

    % OUTPUTTING POINTS
    pts = struct('x', x(:, end-2), 'z', z(end-2));
    pts(end+1) = struct('x', x(:, end-1), 'z', z(end-1));
    pts(end+1) = struct('x', x(:, end), 'z', z(end));
    out.pts = pts;
    out.iters = iterTot;
    out.allPts = x;
    out.fCalls = [fCalls 0 0];
end

function out = grMin(x0, f)
    % SETUP
    alpha = (3 - sqrt(5))/2;   % Step Size

    xL = x0(:, 1);             % Creating Points
    xR = x0(:, 3);             % |
    x1 = xL + alpha*(xR - xL); % |
    x2 = xR - alpha*(xR - xL); % #

    fL = f(xL);                % Evaluating at Points
    f1 = f(x1);                % |
    f2 = f(x2);                % |
    fR = f(xR);                % #

    X = [xL x1 x2 xR];         % Creating Test Matrix
    dx = inf;                  % Initializing Error
    iters = 0;
    fCalls = 4;

    % ITERATING
    allX = X;
    while abs(dx) > 1e-8
        if f1 > f2
            % MOVING BOUNDS
            xL = x1; fL = f1;
            x1 = x2; f1 = f2;

            % RE-EVALUTING POINT
            x2 = xR - alpha*(xR - xL);
            allX = cat(2, x2, allX);
            f2 = f(x2);
        else
            % MOVING BOUINDS
            xR = x2; fR = f2;
            x2 = x1; f2 = f1;

            % RE-EVALUTING POINT
```

```
            x1 = xL + alpha*(xR - xL);
            allX = cat(2, x1, allX);
            f1 = f(x1);
        end

        fCalls = fCalls+1;
        Xnew = [xL, x1, x2, xR];
        dx = norm(Xnew, 'fro') - norm(X, 'fro');
        X = Xnew;
        iters = iters+1;
        if iters > 51; break; end
    end

    % FINDING RESULTS
    [out.zmin, idx] = min([fL, f1, f2, fR]);
    out.xmin = X(:, idx);
    out.allPts = allX;
    out.iters = iters;
    out.fCalls = [fCalls 0 0];
end
```