Optimization Assignment 1
Line Search Algorithms and Gradient Descent

Kim Paolo Laberinto

Winter 2021

Contents

1 Q1. 1D Line Search on Rosenbrock Banana Function

1.1 Set-up

This section of the document discusses some different observations seen in 1D Line Searches on the Rosenbrock Banana Function. The equation for the Rosenbrock 2D Banana Function in shown in Eq. 1.

$$f_{\text{rosenbrock banana}}(x, y) = (a - x)^2 + 100b(y - x^2)^2 \tag{1}$$

In particular for this report, the 2D objective function examined is the Rosenbrock Banana Function where $a = 1$ and $b = 0.1$. This particular 2D Rosenbrock Banana Function is used to investigate the differences and similarities between the following 1D Line Search Methods:

- Swann's Bracketing Method
- Powell's Bracketing Method

Fig. 1 shows Lines A, B, and C along with the contours of the Rosenbrock Banana Function chosen. Note that Lines A and B pass through the global minimum at $(x, y) = (1, 1)$. The following subsection discusses how Swann's and Powell's Bracketing Methods find a suitable bracketing interval on the 1D line.

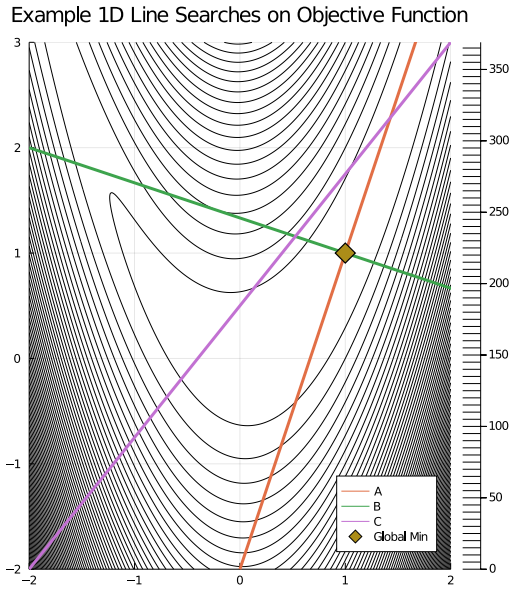


Figure 1: 1D Lines A, B, and C shown on top of the contours of the 2D Rosenbrock Banana Function.

1.2 Swann's vs Powell's for Initial Bracketing

In order to apply a 1D Line Search on an N-D objective function $f : \mathbb{R}^n \to \mathbb{R}$, a parametrization along a 1D line is necessary. This 1D parametrization can be done in the following way in Eq. 2. An initial point $\vec{x_0} \in \mathbb{R}^n$ along with a direction $\vec{d} \in \mathbb{R}^n$ serve as a way to specify which 1D line to select. The $\alpha \in \mathbb{R}$ parameter then is used to specify where along the 1D line to sample from for the 1D Line Search Methods.

$$f_{1\text{D}} : \alpha \mapsto f\left(\vec{x_0} + \alpha\vec{d}\right) \tag{2}$$

With this parametrization, the 1D Line Search Methods can try various values of α to sample the objective function. The goal of the 1D Line Search is then to determine a suitable bracketing interval in terms of α_{lower} and α_{upper} wherein a minimum exists for the objective function along the 1D line.

Applying Swann's Bracketing Method and Powell's Bracketing Methods on Lines A, B, and C from Fig. 1 result initial bracketing intervals shown in Fig. 2, Fig. 3, Fig. 4.

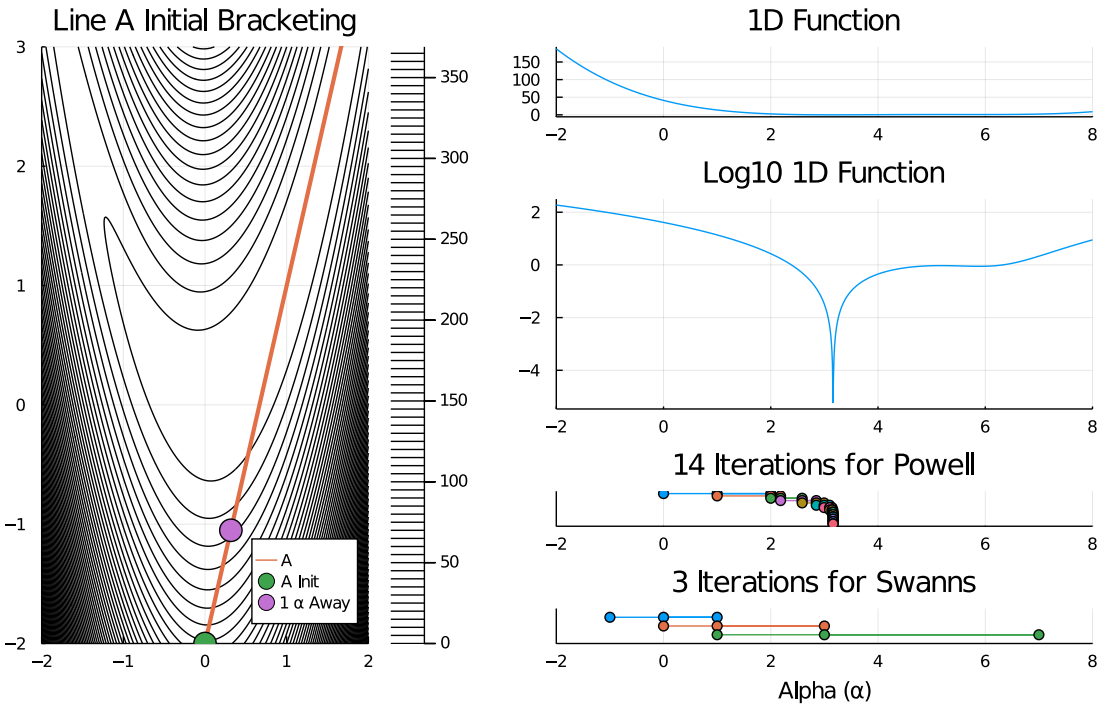


Figure 2: 1D Line Search Intervals on Line A found using Swann's and Powell's Bracketing Methods. Note that the location for global minimum $(1, 1)$ can be found on this line.

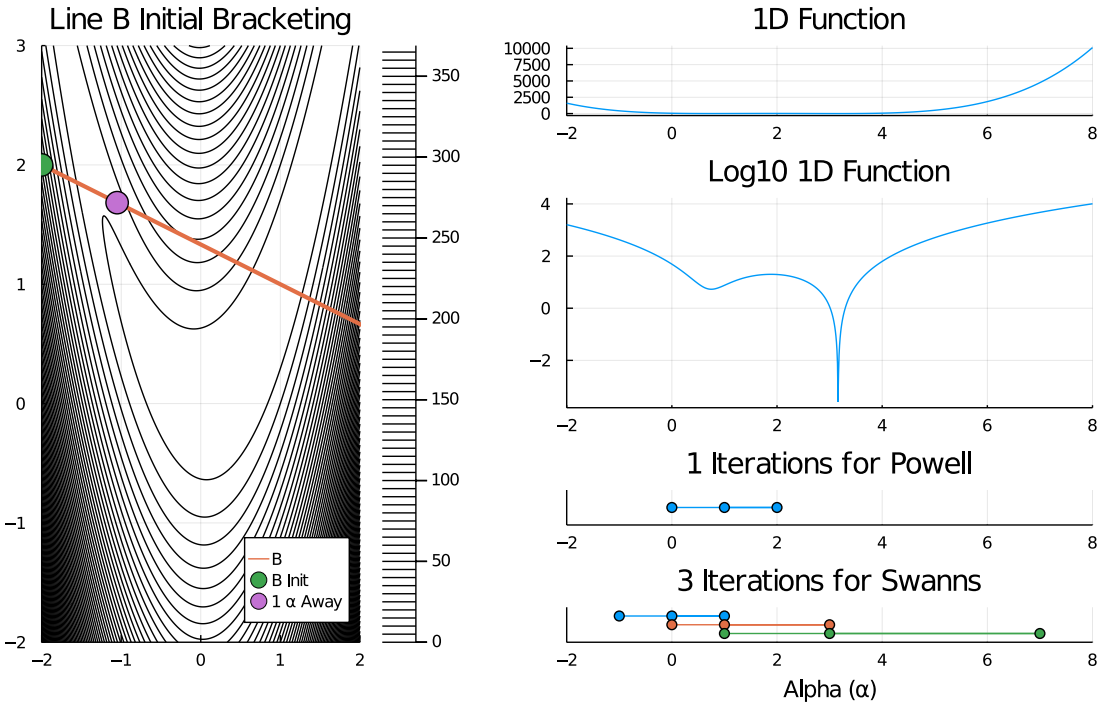


Figure 3: 1D Line Search Intervals on Line B found using Swann's and Powell's Bracketing Methods. Note that the location for global minimum $(1, 1)$ can be found on this line.

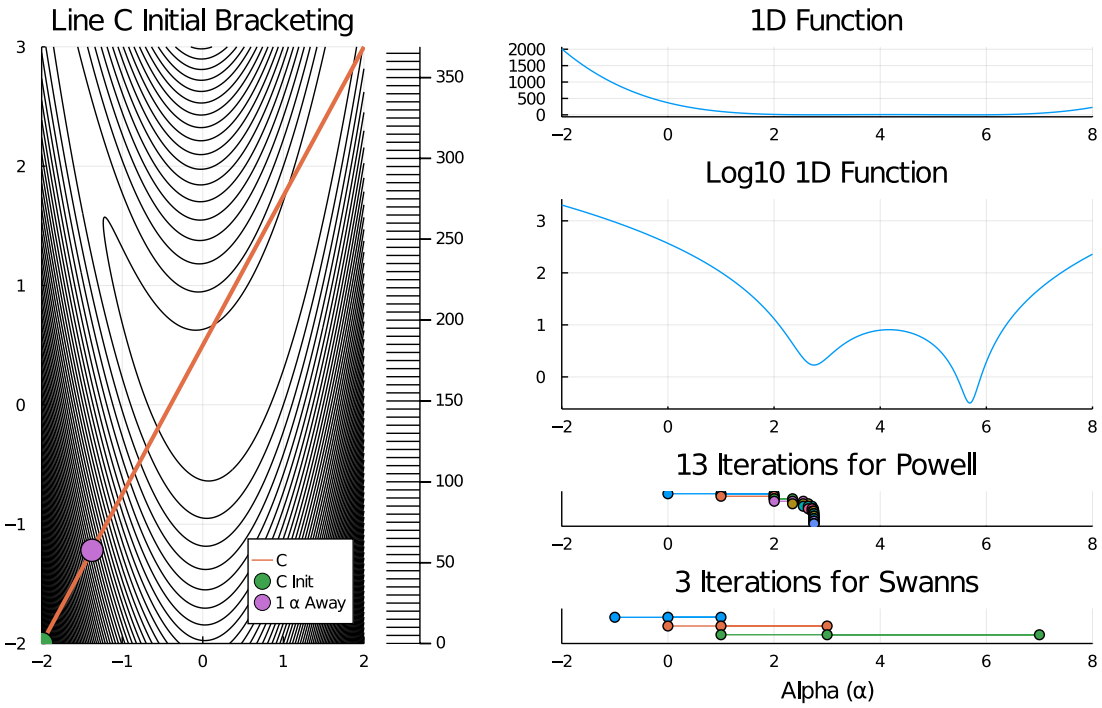


Figure 4: 1D Line Search Intervals on Line C found using Swann's and Powell's Bracketing Methods.

Fig. 2, Fig. 3, Fig. 4 all show the 1D line relative to the 2D contours of the Rosenbrock Banana function, as well as the function value along the 1D line in terms of α. Note that that the objective function value is also shown in terms of a log10 scale to help visualize the differences in function value along the 1D line.

There are several interesting observations to note from these results.

- As per the algorithm definition, the 1D Line Searches only stop once it finds a situable interval.
  - A suitable interval is such where the midpoint evaluated has a lower function value
    - An example of this can be seen in Fig. 2 for Line A, where the final iteration for Swann's has a middle evaluation point at $\alpha_{\text{middle}} = 3$ which has a lower function value than the interval endpoints $\alpha_{\text{lower}} = 1$ and $\alpha_{\text{upper}} = 7$
  - If a non-suitable interval is found, then a new iteration for the 1D search is done. Powell's Bracketing Method tests the next point using a quadratic fitting technique and testing the minimum of such a parabola (within bounds). Whereas Swann's Bracketing Method simply tests farther away with a magnification factor and step size.
    - This can be seen by the multiple intervals/iterations shown in the figures.
- For Lines A and B which contained the global minimum there were different results in the Bracketing Methods in whether or not the global minimum was captured.
  - For Line A both Powell's and Swann's Bracketing Intervals captured the global minimum because there was only one minimum in the nearby 1D function as seen on the log10 plot.
  - For Line B, only Swann's Bracketing Interval captured the global minimum inside its final interval. As can be seen for Powell's Bracketing Interval, only 1 interation was done as it formed a suitable interval.
- For Lines A and C, Powell's Bracketing Method made the intervals smaller and smaller. It seems like due to the curvature of the 1D function near the initial starting point, the fitted parabolas formed such that to only move a small amount further along the 1D function.

- For Lines B and C, the two different bracketing methods captured two different local minima.
  - As can be seen with Powell's Bracketing Method, due to the local function behavior near the starting point, only the first nearby minimum was captured by Powell's. Powell's missed the deeper minima.
  - Swann's Method was simple enough with its steps to capture the deeper minima in a large interval.

1.3 Golden Section Search after Swann/Powell

After finding the initial bracketing interval, a Golden Section Search is performed. Fig. 5 below shows how the Powell and Swann's initial bracketing intervals are made smaller to some tolerance level. Note that the figure shows the 1D function values on a log10 scale to help to see the local function behavior.

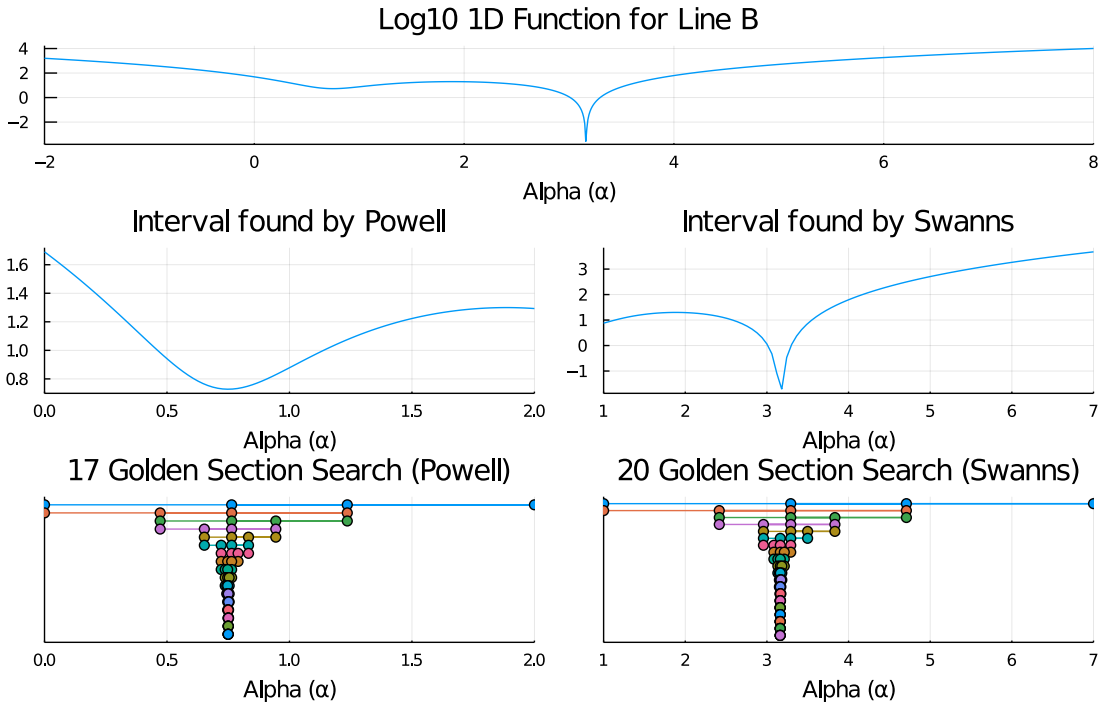


Figure 5: Golden Section Search applied to 1D Line Search Results on Line B. The 1D Function shown in the plots are on a log10 scale. Iterations for Golden Section Search shown where the top is the initial bracketing interval, and towards the bottom is the final bracketing interval.

There are some interesting observations on the results of the Golden Section Search on Line B

- Because Powell and Swann's found different intervals the Golden Section Search refined the intervals into two different minima.
- The global minimum of the 2D function is found near $\alpha \approx 3$. The Golden Section Search from the Swann's Bracketing Method correctly found this global minimum.
- The inner (non-end-point) function evaluations for the Golden Section Search are also shown in the plot. As can be seen, because of the golden ratio the previous functionevaluation points are able to be correctly reused.

Point Label	(x, y) Location
D	(-2, -2)
E	(-1.5, 1.5)
F	(-1, 3)
G	(-0.5, -1.5)
H	(2, 2)
Global Min	(1, 1)

Table 1: Initial Points and Global Minimum Locations for 2D Rosenbrock Objective Function Search

2 Q2. Search on 2D Rosenbrock Banana Function

2.1 Set-up

In this section, a couple of search methods are applied to the full 2D Rosenbrock Banana Function from various different initial points. The two major search methods are:

- Gradient Descent Method
  - Using the gradient (∇f) of the Rosenbrock function find the next direction to search
  - Then using Swann's and Powell's Methods (given some tolerance parameter), the algorithm searches in that 1D direction found using the gradient
  - Once a minimum point in that 1D direction is found, the algorithm loops until the $\|\nabla f\| < 10^{-4}$.
- (EXTRA) Hooke-Jeeves Method

These methods are applied to various initial positions on the 2D parameter search space. The labels for the initial points are found below in Table. 1. These points relative to the contours of the 2D Rosenbrock Function are also shown in the following section.

2.2 Gradient Descent Results

Each gradient step of the algorithm is recorded and plotted in Fig. 6 from the various different initial points in Table. 1. As seen in Fig. 6, various different settings were used to see if there were any differences in the resulting trajectories. The two line search initial bracketing methods used were Powell's and Swann's. Two different settings for the tolerance was also used, 10^{-4} shown on the top row, and 10^{-2} shown on the bottom row.

The values of the objective function for each gradient descent algorithm can be found in Fig. 7.

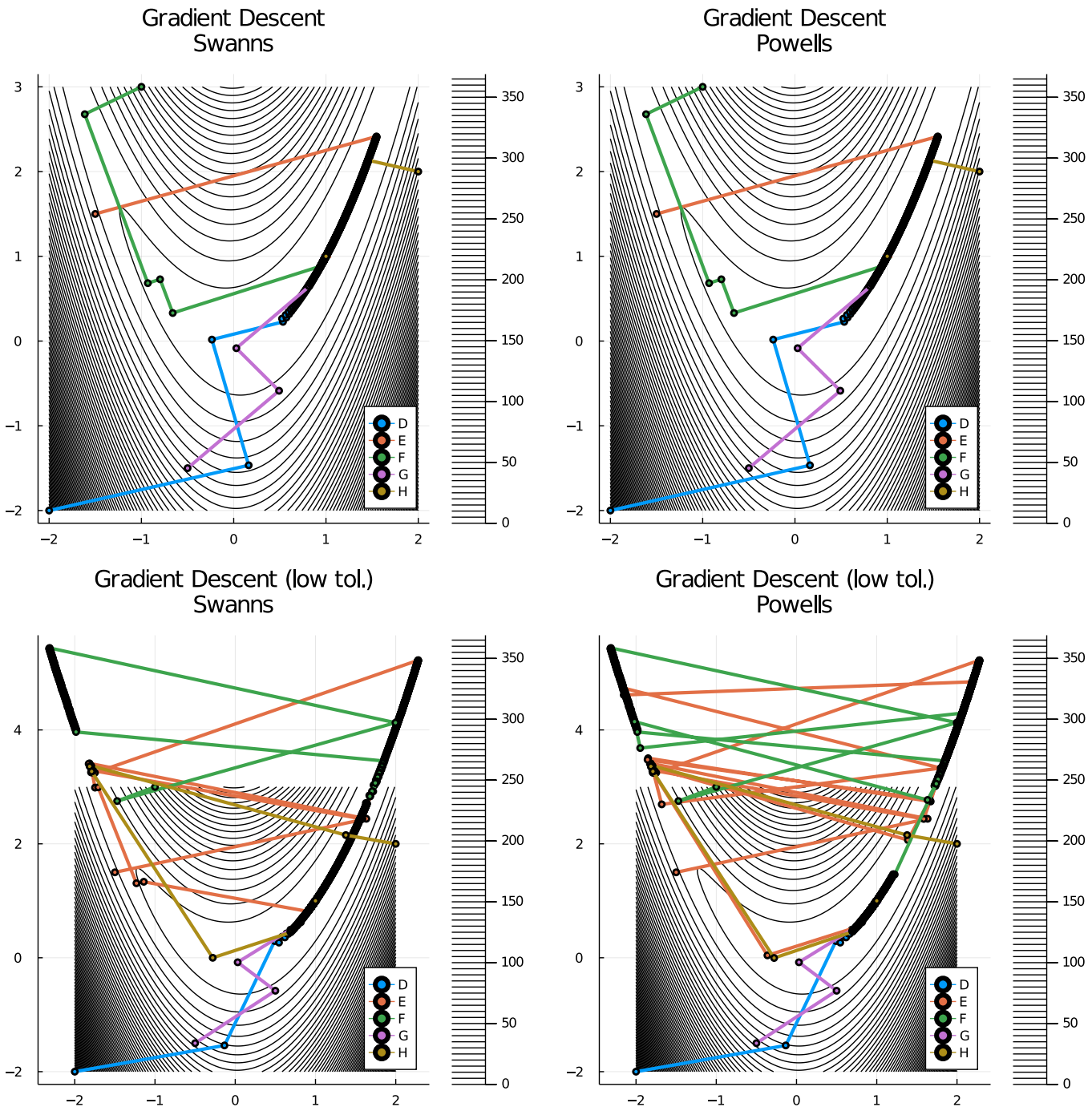


Figure 6: The trajectories of the Gradient Descent on Points D, E, F, G, and H using Powell's and Swann's Bracketing Methods. The top row uses a line search tolerance of 10^{-4} while the bottom row (low tol) uses a line search tolerance of 10^{-2}. Contours only shown in limited region for clarity. Global Minimum located at $(1, 1)$.

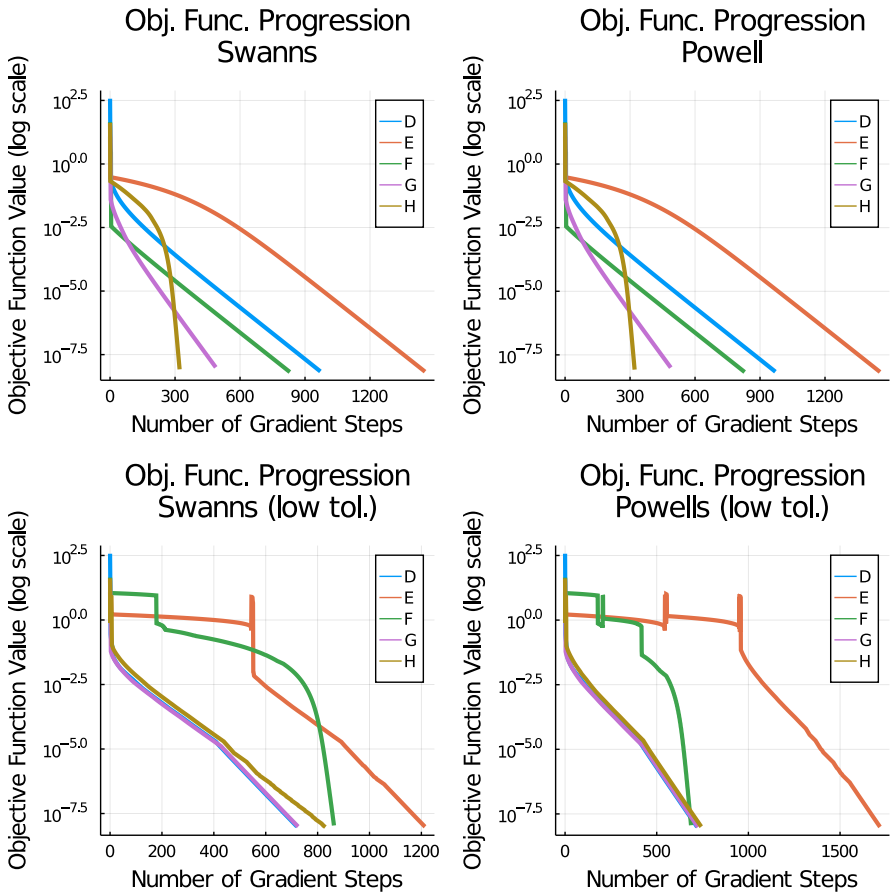


Figure 7: Objective Function Value vs Gradient Descent Steps for Points D, E, F, G, and H using Powell's and Swann's Bracketing Methods. The top row uses a line search tolerance of 10^{-4} while the bottom row (low tol) uses a line search tolerance of 10^{-2}. Global Minimum located at $(1, 1)$.

There are a few interesting observations to note from these Gradient Descent results seen in the contour plots on Fig. 6

- On the the top row (with the correct amount of tolerance) the algorithm was able to correctly finds the global minimum at $(1, 1)$ from the various initial starting positions.
- There is no noticeable difference between the gradient steps of Swann's and Powell's on the top row.
- As per the course notes, every step of the gradient descent should be orthogonal to the previous step as the gradient. This phenomena can be seen in the top row with the properly calibrated line search tolerance, where each gradient step is at 90 degrees to the previous gradient step.
- All gradient descent trajectories encountered difficulty in the valley of the Rosenbrock Banana Function. The trajectories had to zig-zag down the valley to try to find the minimum.

There are a few interesting observations to note from Fig. 7 with the objective function value plotted against the number the gradient descent steps.

- On the top row, the effect of zig-zag through the valley can be seen with the much slower descent taking much more gradient steps to find the global minimum.
- The curve jumps more erratically in the bottom row with the low tolerance.

2.3 (EXTRA) Hooke-Jeeves Result

The Hooke-Jeeves algorithm (with parameters $\Delta = 0.2$ and $\delta = 0.001$) was applied to the same 2D Rosenbrock Banana Function starting at the same initial values. The results from this can be seen in Fig. 8.

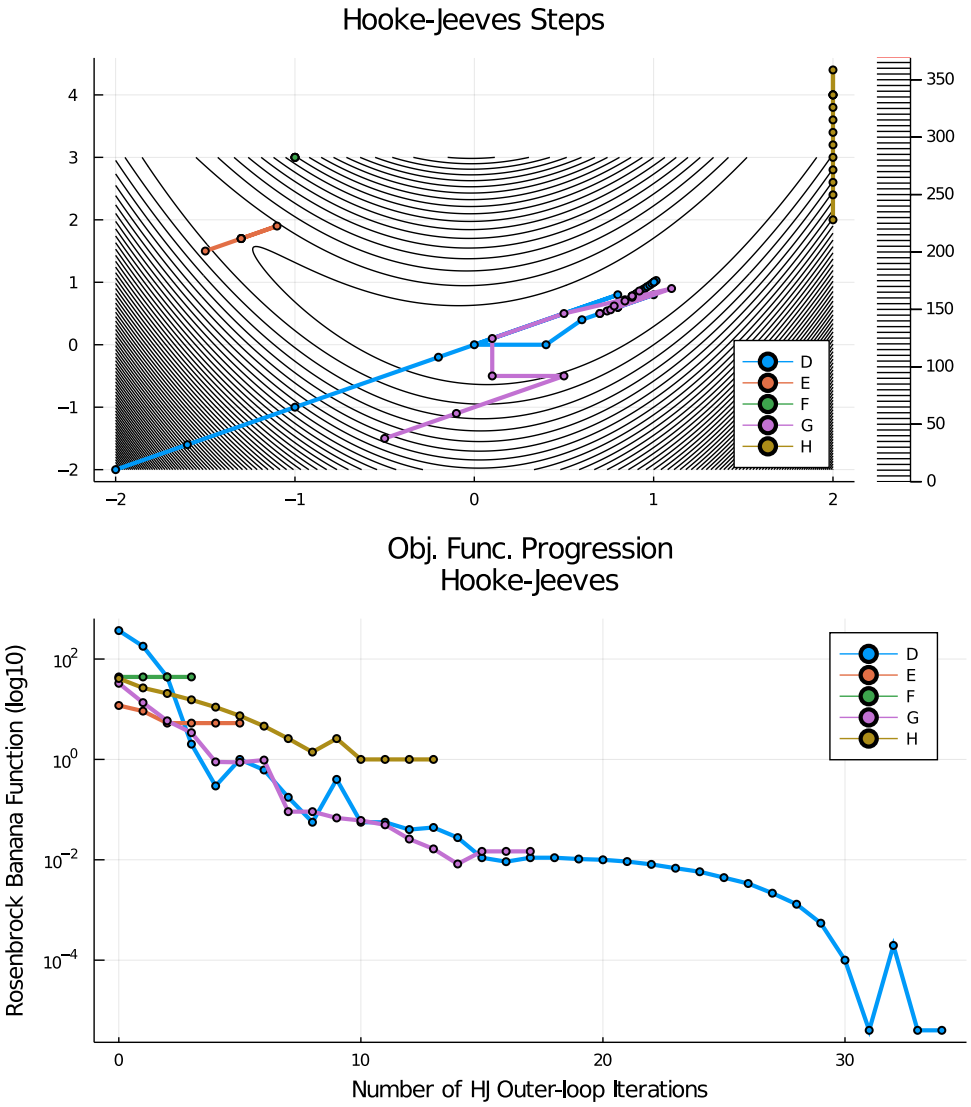


Figure 8: Hooke-Jeeves steps visualized when applied to the 2D Rosenbrock Function.

There are a few interesting observations from this.

- Because Hooke-Jeeves does not have access to the gradient information, the algorithm seems to have a harder time finding the global minimum from points E, F, and H.
- The behavior of bouncing around in the valley can still be seen with D and G.
- The trajectory from point H went towards the valley, but away from the global minimum.

t	0	1	2	3	4	5	6	7	8	9
y(t)	1.75	1.65	1.56	1.49	1.43	1.37	1.33	1.29	1.26	1.23

Table 2: Data used for Model Fitting

3 Q3. Model Fitting using Optimization Techniques

3.1 Set-up

Given a set of data point (also called measurements) and a model, optimization techniques can be applied to find the model parameters which best fit the data.

In this lab assignment, the model f used will be the following where the parameter vector is $x \in \mathbb{R}^5$. The data used for fitting is found in Table 2.

$$f(t;x) = y = x_1 + x_2 e^{x_4 t} + x_3 e^{x_5 t} \tag{3}$$

The objective function used for optimization will be the sum of squared errors, and as such the optimization will try to find the parameters that best fit the data in the least squares sense.

In this section, model fitting will be done using two different optimization techniques:

- Gradient Descent (as described in Q2)
- Hooke-Jeeves

3.2 Gradient Descent Method

3.2.1 Sum Squares Function and Gradient Derivation

In order to apply the Gradient Descent method to the objective function, the gradient function must be obtained. The following steps show the symbolic derivation of the gradient of the objective function.

First, we define the objective function L, which is way to evaluate a model's fit of the data (t_i, y_i) in a least squares sense.

$$L = \sum_i (y_{\text{data}} - y_{\text{model}})^2 \tag{4}$$

$$L = \sum_i (y_i - (x_1 + x_2 e^{x_4 t_i} + x_3 e^{x_5 t_i}))^2 \tag{5}$$

We can now derive the gradient ∇L.

$$\nabla L = \nabla \left(\sum_i (y_i - (x_1 + x_2 e^{x_4 t_i} + x_3 e^{x_5 t_i}))^2 \right) \tag{6}$$

$$\nabla L = \sum_i \left(\nabla \left((y_i - (x_1 + x_2 e^{x_4 t_i} + x_3 e^{x_5 t_i}))^2 \right) \right) \tag{7}$$

$$\nabla L = \sum_i \left(2(y_i - (x_1 + x_2 e^{x_4 t_i} + x_3 e^{x_5 t_i})) \cdot \nabla (y_i - x_1 + x_2 e^{x_4 t_i} + x_3 e^{x_5 t_i}) \right) \tag{8}$$

$$\nabla L = \sum_i \left(2(y_i - (x_1 + x_2 e^{x_4 t_i} + x_3 e^{x_5 t_i})) \cdot \nabla (x_1 + x_2 e^{x_4 t_i} + x_3 e^{x_5 t_i}) \right) \tag{9}$$

Label	$[x_1, x_2, x_3, x_4, x_5]$ **Initial Parameter Vector**
Initial Guess 1	$[0.1, 0.1, 0.1, 0.1, 0.1]$
Initial Guess 2	$[-0.1, -0.1, -0.1, -0.1, -0.1]$
Initial Guess 3	$[0.0, 0.1, -0.1, 0.0, 0.1]$

Table 3: Labels for Initial Parameters used in Model Fitting

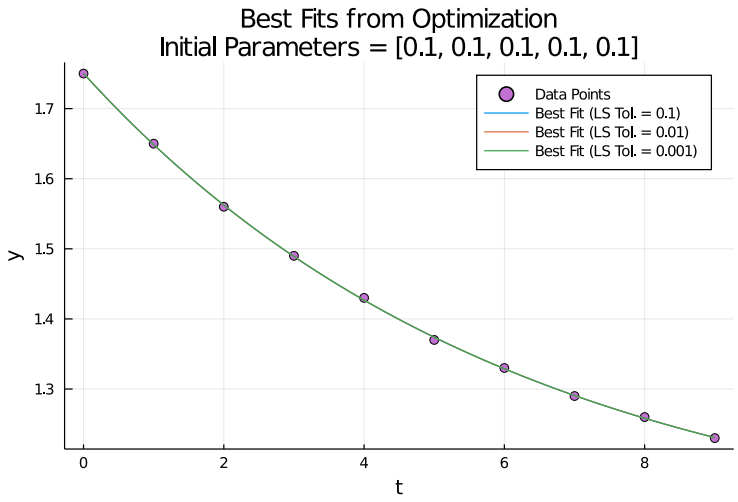
$$\nabla(x_1 + x_2 e^{x_4 t_i} + x_3 e^{x_5 t_i}) = \begin{bmatrix} 1 \\ e^{x_4 t_i} \\ e^{x_5 t_i} \\ x_2 t_i e^{x_4 t_i} \\ x_3 t_i e^{x_5 t_i} \end{bmatrix} \tag{10}$$

The above expressions for ∇L and $\nabla(x_1 + x_2 e^{x_4 t_i} + x_3 e^{x_5 t_i})$ define the gradient in full given some data and a location in the parameter space to evaluate the gradient.

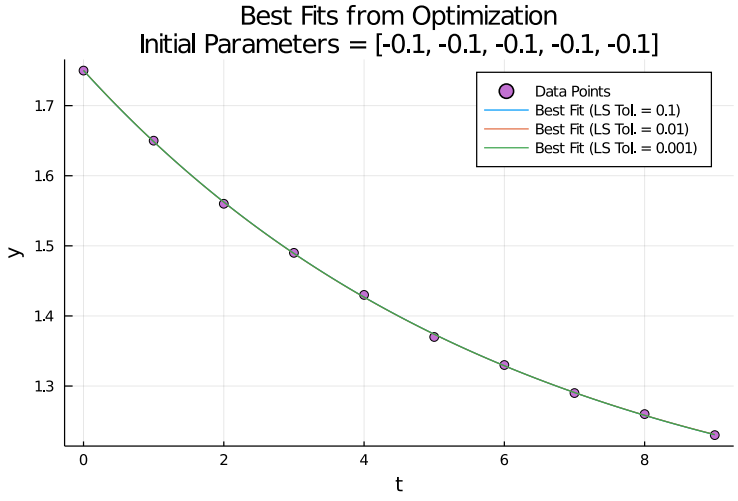
3.2.2 Gradient Descent Results

Using the gradient defined above, the following figures are results from the gradient descent algorithm from various different starting guesses. The starting guesses can be found in Table 3.

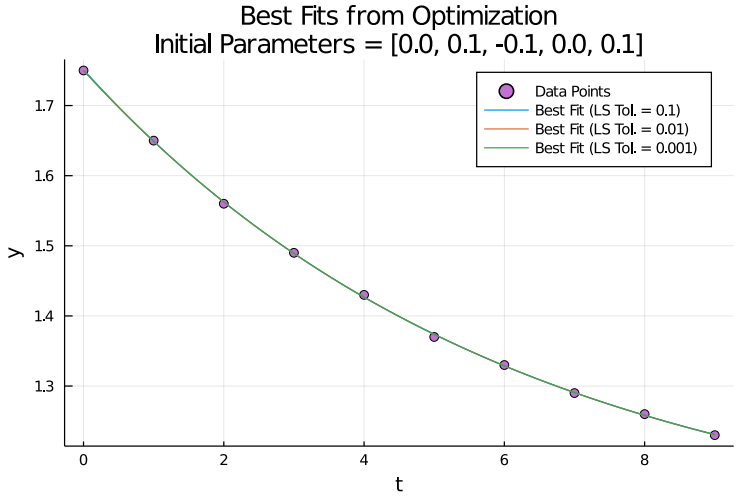
The resulting models are shown in Fig. 9. They are plotted against the data.



(a) Initial Guess 1

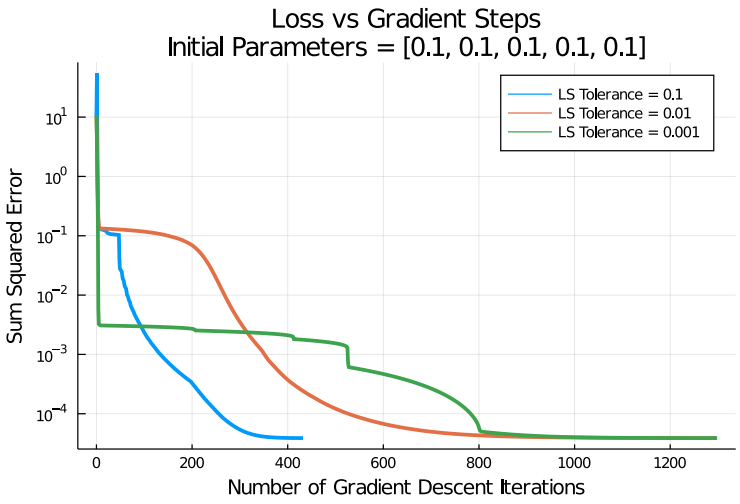


(b) Initial Guess 2

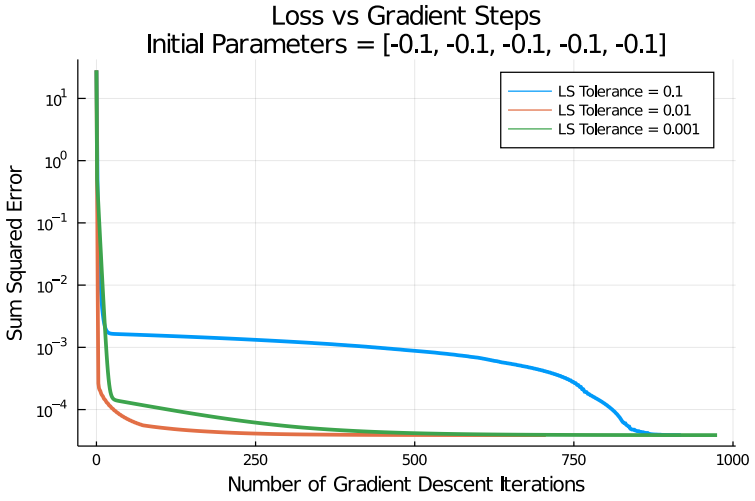


(c) Initial Guess 3

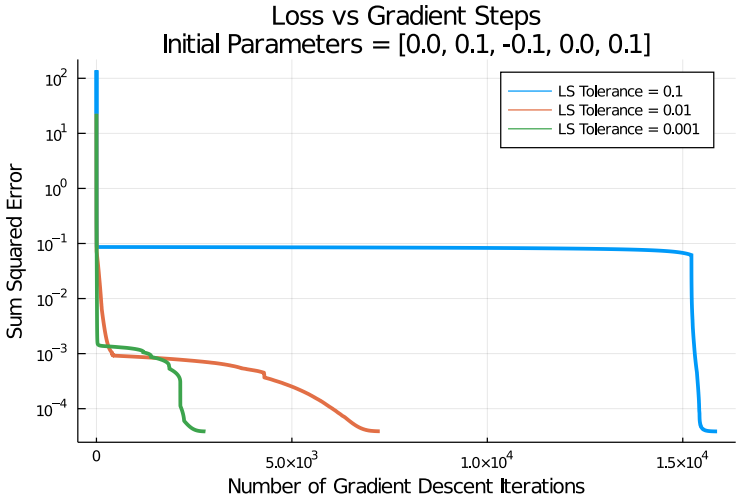
Figure 9: Result of optimization of model parameters using various initial guesses. Fitted model plotted against data. Various Line Search tolerances were used to do the gradient descent as shown in the legend of the plots.



(a) Initial Guess 1



(b) Initial Guess 2



(c) Initial Guess 3

Figure 10: Loss vs Gradient Descent Iterations plot for various guesses. Different line search tolerances were used as shown in legend of plots.

Initial Guess	Line Search Tolerance	Number of Function Evals	Number of Gradient Steps	Final Parameter Vector [x1, x2, x3, x4, x5] (rounded)
	0.1	6113	433	[1.076, 0.337, 0.337, -0.163, -0.163]
Initial Guess 1	0.01	24643	1298	[1.076, 0.337, 0.337, -0.163, -0.163]
	0.001	29904	1299	[1.074, 0.338, 0.338, -0.162, -0.163]
	0.1	12918	919	[1.074, 0.338, 0.338, -0.163, -0.163]
Initial Guess 2	0.01	13416	707	[1.076, 0.337, 0.337, -0.163, -0.163]
	0.001	22425	976	[1.074, 0.338, 0.338, -0.163, -0.163]
	0.1	222340	15879	[1.078, -0.002, 0.674, -51.631, -0.165]
Initial Guess 3	0.01	137726	7249	[1.074, 0.000, 0.676, -1.693, -0.163]
	0.001	64230	2791	[1.075, -0.001, 0.675, -4.498, -0.163]

Table 4: Summary of Results and Number of Function Evaluations for Gradient Descent Model Fitting.

There are a few interesting observations to make from these results and from the table of number of function evaluations on Table 4.

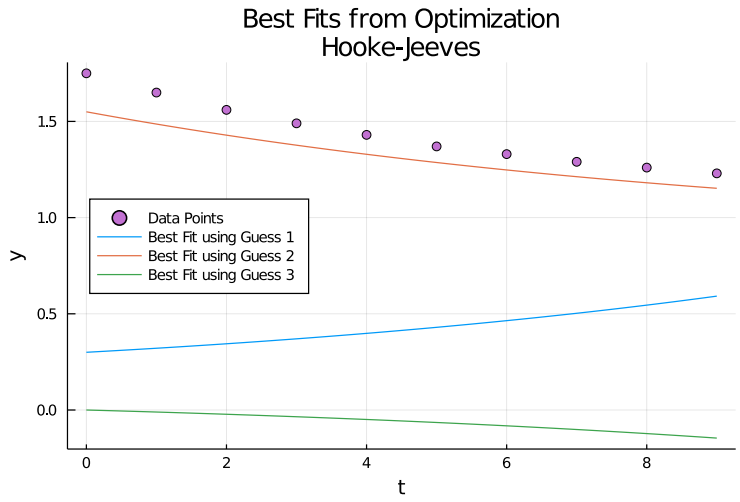
- The different initial guesses and line search tolerances used did not change the final fit of the model parameter.
- The different initial guesses and line search tolerences did make a difference into the exact final parameter vector that the technique converged to. This seems to indicate multiple local minima that all have various similar fits (least squares sense). In fact as seen with some of the final parameter vectors having $x_2 \approx 0$ term (from Initial Guess 3), a single constant term summed up with an exponential term may be another suitable model instead of two different exponential terms as given in the assignment. Recall that the original model given the assignment is $f(t;x) = x_1 + x_2 e^{x_4 t} + x_3 e^{x_5 t}$.
- Not shown here, however comparing the results between using Swann's vs Powell's for the line search bracketing did not make a difference between the final fit of the model.
- All the trials was able to reduce the sum of squared error to 10^{-1} very quickly in a short amount of gradient steps. The rest of the gradient steps were used to reduce that error down lower than 10^{-4}.
- The run using Initial Guess 3 and Line Search tolerance of 0.1 was stuck for a large amount of steps as seen in the plot and the number of function evaluations and gradient descent evaluations.

3.3 Hooke-Jeeves Direct Search Method Results

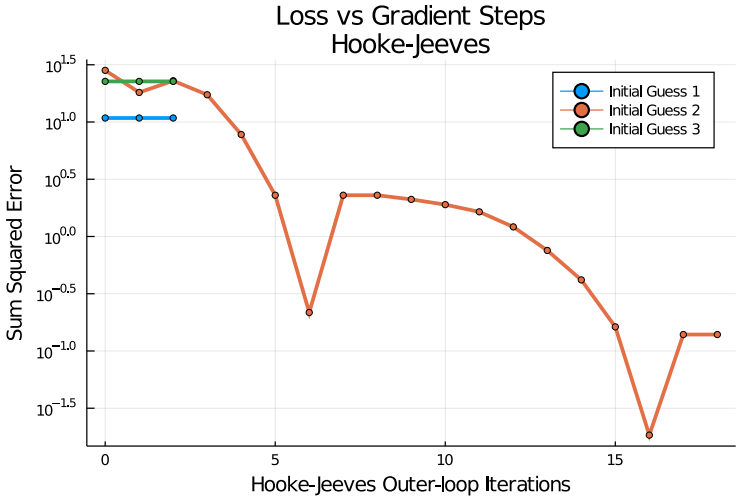
The Hooke-Jeeves algorithm (with parameters $\Delta = 0.1$ and $\delta = 0.01$) was applied using the same initial guesses as in Table 3 to the Model Fitting optimization problem. A visualization of the results can be found on Fig. 11. A table summarizing the number of function evaluations and the final parameters can be seen in Table 5. (Note zero gradient function evaluations were used because the Hooke-Jeeves algorithm does not use the gradient function.)

Initial Guess	Number of Function Evals	Number of Gradient Steps	Final Parameter Vector [x1, x2, x3, x4, x5] (rounded)
Initial Guess 1	35	0	[0.1, 0.1, 0.1, 0.1, 0.1]
Initial Guess 2	308	0	[0.880, 0.770, -0.1, -0.1, -0.1]
Initial Guess 3	35	0	[0.0, 0.1, -0.1, 0.0, 0.1]

Table 5: Table Summary of results from Hooke-Jeeves



(a) Final Models found using Hooke-Jeeves compared with data.



(b) Loss vs Iterations for Hooke-Jeeves

Figure 11: Results from Hooke-Jeeves on Model Fitting

When compared with the results from the Gradient Descent algorithm, there are a few observations to make.

- Clearly, Hooke-Jeeves was not able to properly fit the model to the data.
- Two of the initial starting guesses (1, and 3) was not able to move away properly as seen in the Loss vs Iterations plot. With the given orthogonal directions and the Δ and δ parameter, the Hooke-Jeeves algorithm was stuck and did not proceed.
- The number of function evaluations for Hooke-Jeeves was lower than the the number of function evaluations for Gradient Descent. However clearly, Hooke-Jeeves did not work.
- Gradient-Descent with the gradient information was much better at finding the best fitting parameters.

3.4 (EXTRA) Automatic Differentiation Gradient Descent

As an alternative to deriving the gradient of the objective function by hand, one can apply a technique called Automatic Differentiation. Note that automatic differentiation is not symbolic differentiation nor

finite difference methods. My understanding of automatic differentiation uses the source code representation of the objective function and applies a concept called dual numbers (similar to imaginary numbers but with a quantity called ϵ where $\epsilon^2 = 0$) to do the automatic differentiation on all the sub-operations in the source code (e.g. addition, subtraction, products, etc.).

The implementation for automatic differentiation in this lab assignment used the ForwardDiff.jl package in the programming language called Julia. The ForwardDiff.jl package can be used to compute the gradient automatically. An example of such an implementation is the following piece of code.

```
function autodiff_grad_objective_function(params)
    global N_grad_f_eval += 1
    return ForwardDiff.gradient(objective_function, params)
end
```

The above function as then used as the gradient function for the Gradient Descent Algorithm. The result from this technique to the Model Fitting question can be seen in Fig. 12. As seen in the error vs gradient steps plot, the progression to find the minimum is identical.

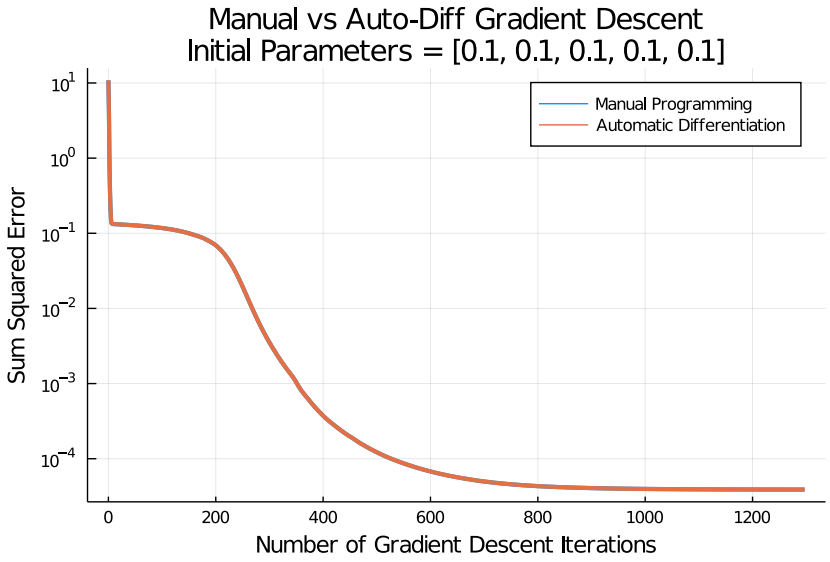


Figure 12: Comparison of Error vs Steps between the manually programmed Gradient Descent vs Automatic Differentiation

4 Q4. Vector Space Exercises (Graduate Student - EXTRA)

Omitted for the sake of time.

4.1 Proving B_1 and B_1 are both a basis of V

4.1.1 Proof: B_1 is Linearly Independent

4.1.2 Proof: B_1 spans V

4.1.3 Proof: B_2 is Linearly Independent

4.1.4 Proof: B_2 spans V

4.2 Transformation between B_1 and B_2

4.3 D Derivative Operator

4.3.1 Proof: D is linear

4.3.2 Matrix Representation of D in the bases B_1 and B_2

5 Q5. Linear Transformation and Coordinates Exercise (Graduate Student - EXTRA)

Omitted for sake of time.

5.1 x in full $\mathbb{R}^3$ standard basis

5.2 Coordinates of x in B_X

5.3 Coordinates of y in B_Y using L ($c_y = Lc_x$)

5.4 y in full $\mathbb{R}^4$ standard basis

6 Q6. Matrix Calculus / Derivative Exercise (Graduate Student - EXTRA)

6.1 Df derivation

Let $f : \mathbb{R}^4 \to \mathbb{R}$ be defined as $f(x) = \|Ax - b\| + \lambda\|Cx\| + \gamma\|Eb\|$ where $A, C, E \in \mathbb{R}^{4\times4}$, $x, b \in \mathbb{R}^4$ and $\lambda, \gamma \in \mathbb{R}$.

Let D denote the differential operator.

$$
\begin{aligned}
Df(x) &= D\left(\|Ax - b\| + \lambda\|Cx\| + \gamma\|Eb\|\right) \\
&= D\left(\|Ax - b\|\right) + \lambda D\left(\|Cx\|\right) + \gamma D\left(\|Eb\|\right) && \text{[using linearity of D]} \\
&= D\left(\|Ax - b\|\right) + \lambda D\left(\|Cx\|\right) + 0 \\
&= D\left(\|Ax - b\|\right) + \lambda D\left(\|Cx\|\right) \\
&= \frac{(Ax - b)^T A}{\|Ax - b\|} + \lambda\frac{(Cx)^T C}{\|Cx\|} && \text{[using derivations from next sections]}
\end{aligned}
$$

The full derivation of $D\left(\|Ax - b\|\right)$ and $D\left(\|Cx\|\right)$ can be found in the following sections.

6.2 $D\left(\|Cx\|\right)$ Derivation

$$
\begin{aligned}
D\left(\|Cx\|\right) &= D\left(\sqrt{(Cx)^T(Cx)}\right) \\
&= D\left(\left((Cx)^T(Cx)\right)^{\frac{1}{2}}\right) \\
&= \frac{1}{2}\left((Cx)^T(Cx)\right)^{\frac{-1}{2}} D\left((Cx)^T(Cx)\right) && \text{[using Chain Rule]} \\
&= \frac{1}{2\sqrt{(Cx)^T(Cx)}} D\left((Cx)^T(Cx)\right) \\
&= \frac{1}{2\|Cx\|} D\left((Cx)^T(Cx)\right) \\
&= \frac{\left((Cx)^T D(Cx)\right) + \left((Cx)^T D(Cx)\right)}{2\|Cx\|} && \text{[using Product Rule]} \\
&= \frac{2\left((Cx)^T D(Cx)\right)}{2\|Cx\|} \\
&= \frac{(Cx)^T C}{\|Cx\|}
\end{aligned}
$$

6.3 $D\left(\|Ax-b\|\right)$ Derivation

$$
\begin{aligned}
D\left(\|Ax-b\|\right) &= D\left(\sqrt{(Ax-b)^T(Ax-b)}\right) \\
&= D\left(\left((Ax-b)^T(Ax-b)\right)^{\frac{1}{2}}\right) \\
&= \frac{1}{2}\left((Ax-b)^T(Ax-b)\right)^{\frac{-1}{2}} D\left((Ax-b)^T(Ax-b)\right) \quad \text{[using Chain Rule]} \\
&= \frac{1}{2\sqrt{(Ax-b)^T(Ax-b)}} D\left((Ax-b)^T(Ax-b)\right) \\
&= \frac{1}{2\|Ax-b\|} D\left((Ax-b)^T(Ax-b)\right) \\
&= \frac{\left((Ax-b)^T D(Ax-b)\right) + \left((Ax-b)^T D(Ax-b)\right)}{2\|Ax-b\|} \quad \text{[using Product Rule]} \\
&= \frac{2(Ax-b)^T D(Ax-b)}{2\|Ax-b\|} \\
&= \frac{2(Ax-b)^T A}{2\|Ax-b\|} \\
&= \frac{(Ax-b)^T A}{\|Ax-b\|}
\end{aligned}
$$

A Extra Plots

A.1 Q1 Extra Plots

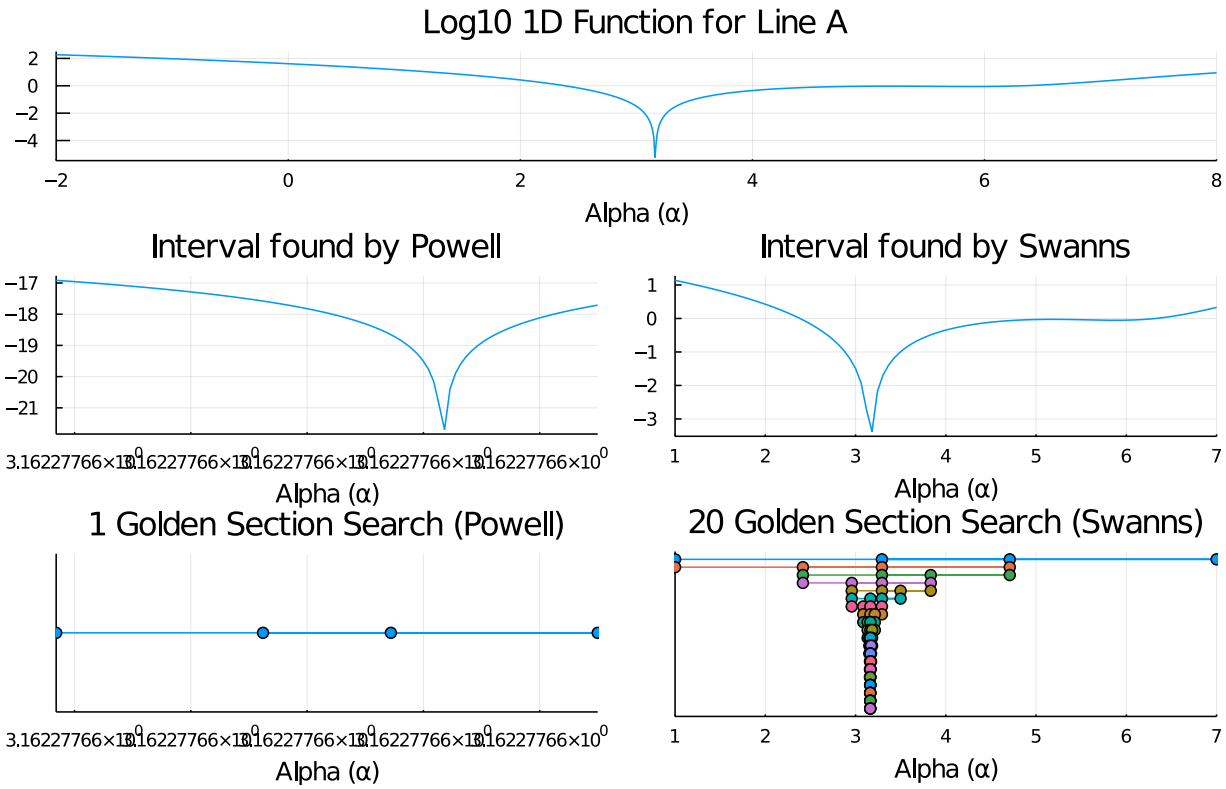


Figure 13: Golden Section Comparison for Line A in Q1

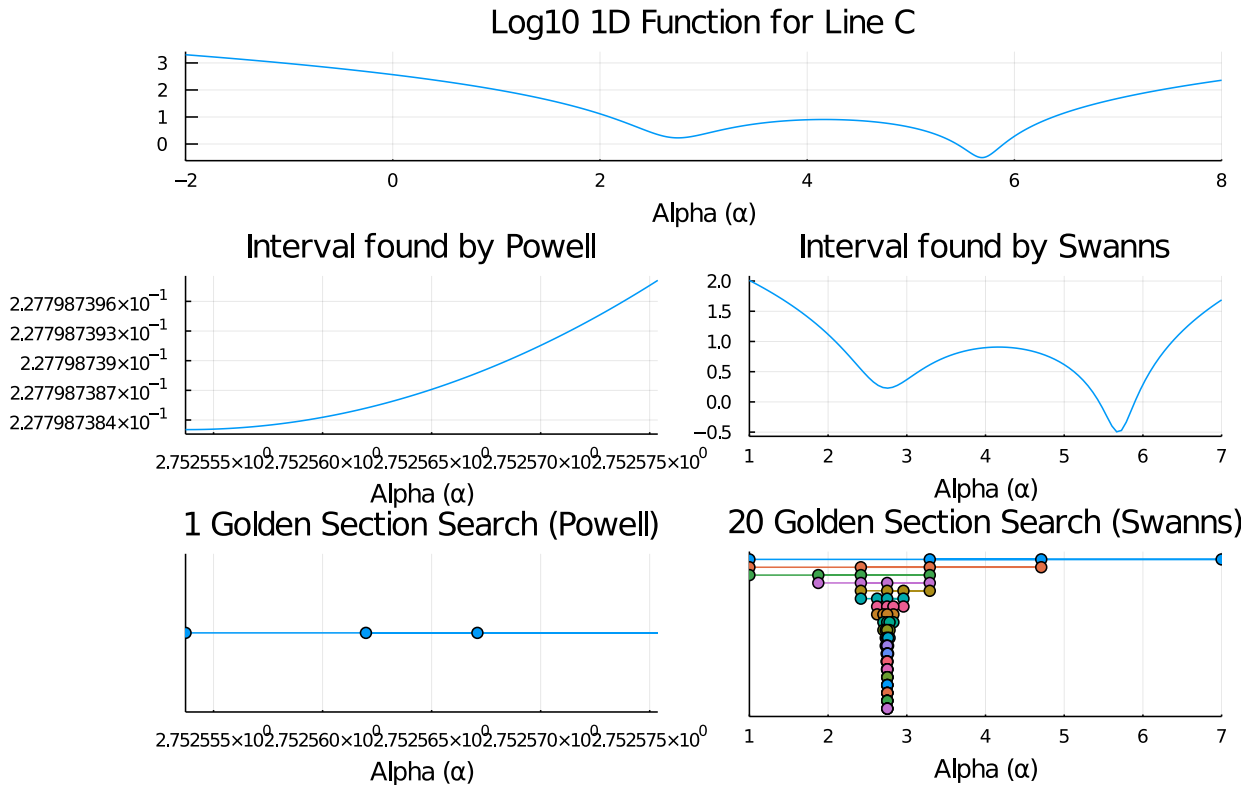


Figure 14: Golden Section Comparison for Line C in Q1

B Source Code

B.1 All Optimization Algorithms (A1Module.jl)

```
module A1Module

using LinearAlgebra #External module for taking norm
using Memento #Invenia Module for logging
using Printf #External module for formatting strings
using Plots
using ValueHistories #External Package for keeping track of values

export SwannsBracketingMethod
export PowellsBracketingMethod
export GoldenSectionSearch
export Q1LineSearch
export Q2SteepestDescent
export HookeJeeves

# Set up Memento Logger
const LOGGER = getlogger(@__MODULE__)
function __init__()
    Memento.register(LOGGER)
end

function SwannsBracketingMethod(f, x_0, initial_step_length; magnification = 2)
    info(LOGGER, "Inside Swann's Bracketing Method")

    #Define test points
    x_l = x_0 - abs(initial_step_length) #lower
    x_u = x_0 + abs(initial_step_length) #upper
    x_m = x_0 #middle


    #Do function evaluations
```

```
    debug(LOGGER, "Performing initial function evaliations")
    f_l = f(x_l) #lower
    f_u = f(x_u) #upper
    f_m = f(x_0) #middle

    history = History(Tuple{Float64, Float64, Float64})
    push!(history, 0, convert(Tuple{Float64, Float64, Float64}, (x_l, x_u, x_m)))

    # 4 Cases.
    # 1) Keep moving right
    # 2) Keep moving left
    # 3) Initial interval is bracket
    # 4) Error (non-unimodal)

    debug(LOGGER, @sprintf "x_l x_m x_u = [%3.2f, %3.2f %3.2f]" x_l x_m x_u)
    if f_l >= f_m >= f_u
        debug(LOGGER, "Case 1: Keep Moving Right")
        i = 1
        while f_u < f_m
            debug(LOGGER, @sprintf "x_l to x_m = [%3.2f, %3.2f]" x_l x_m)

            (x_l, x_m, f_l, f_m) = (x_m, x_u, f_m, f_u)
            x_u = x_u + (magnification^i) * abs(initial_step_length)
            f_u = f(x_u)

            push!(history, i, convert(Tuple{Float64, Float64, Float64}, (x_l, x_u, x_m)))
            i += 1
        end
        # f_u is now higher than f_m
        # This now forms a valid bracketing interval

        info(LOGGER, @sprintf "Outputting x_l x_u = [%3.2f %3.2f]" x_l x_u)
        return (x_l, x_u), history

    elseif f_l <= f_m <= f_u
        debug(LOGGER, "Case 2: Keep Moving Left")
        i = 1
        while f_l < f_m
            debug(LOGGER, @sprintf "x_u to x_m = [%3.2f, %3.2f]" x_l x_m)

            (x_u, x_m, f_u, f_m) = (x_m, x_l, f_m, f_l)
            x_l = x_l - (magnification^i) * abs(initial_step_length)
            f_l = f(x_l)

            push!(history, i, convert(Tuple{Float64, Float64, Float64}, (x_l, x_u, x_m)))
            i += 1
        end
        # f_l is now higher than f_m
        # This forms a valid bracketing interval
        info(LOGGER, @sprintf "Outputting x_l x_u = [%3.2f %3.2f]" x_l x_u)
        return (x_l, x_u), history

    elseif f_l >= f_m <= f_u
        debug(LOGGER, "Case 3: Initial interval is bracket")
        info(LOGGER, @sprintf "Outputting x_l x_u = [%3.2f %3.2f]" x_l x_u)
        return (x_l, x_u), history

    else # 4) Error  f_l <= f_m >= f_u
        error(LOGGER, "Case 4: Error (non-unimodal)")
    end

end

function PowellsBracketingMethod(f, a_1, delta, delta_max)
    #TODO: Gets an error when the function is perfectly quadratic e.g x -> x^2
    info(LOGGER, "Inside Powell's Bracketing Method")

    c_1 = a_1 + delta
    F_a = f(a_1)
    F_c = f(c_1)

    if F_a > F_c
        b_1 = a_1 + 2*delta
        F_b = f(b_1)
        forward = true
    else
        (b_1, c_1, a_1) = (c_1, a_1, a_1 - delta)
        (F_b, F_c, F_a) = (F_c, F_a, f(a_1))
        forward = false
    end
```

```
    debug(LOGGER, @sprintf "Direction is forward: %s" forward)
    debug(LOGGER, @sprintf " a_1, b_1, c_1 = %s" (a_1, b_1, c_1))
    debug(LOGGER, @sprintf " F_a, F_b, F_c = %s" (F_a, F_b, F_c))

    a_current = a_1
    b_current = b_1
    c_current = c_1

    a_next = a_1
    b_next = b_1
    c_next = c_1

    history = History(Tuple{Float64, Float64, Float64})
    push!(history, 0, convert(Tuple{Float64, Float64, Float64}, (a_current, b_current, c_current)))

    N_iterations = 0
    while !(F_c < F_a && F_c < F_b)
        debug(LOGGER, @sprintf "New loop iteration...")
        debug(LOGGER, @sprintf "F_a F_b F_c = %5.3f %5.3f %5.3f" F_a F_b F_c)
        a_current = a_next
        b_current = b_next
        c_current = c_next
        N_iterations += 1

        debug(LOGGER, @sprintf "a_current, b_current, c_current = [%5.3f, %5.3f, %5.3f]" a_current b_current c_current)


        p_numerator = (c_current - b_current)*F_a + (a_current - c_current)*F_b + (b_current-a_current)*F_c
        p_denominator = ((b_current - c_current)*(c_current - a_current)*(a_current - b_current))
        p = p_numerator / p_denominator
        debug(LOGGER, @sprintf "p = %5.3f" p)

        # Cases Summarized
        # 1) Moving Forward
        # 1.1) Quadratic has maximum (p <= 0)
        # 1.2) Quadratic has no maximum
        # 1.2.1) Quadratic minimum is too far away
        # 1.2.2) Quadratic minimum is within reach
        # 2) Moving Backward
        # 2.1) Quadratic has maximum (p <= 0)
        # 2.2) Quadratic has no maximum
        # 2.2.1) Quadratic minimum is too far away
        # 2.2.2) Quadratic minimum is within reach

        if forward #Moving Forward
            debug(LOGGER, @sprintf "Moving Forward...")
            if p <= 0 #Quadratic has a maximum, move as far as possible
                debug(LOGGER, @sprintf "Quadratic has a maximum. Move as far forward as possible.")
                a_next = a_current
                b_next = b_current + delta_max
                c_next = c_current

                F_b = f(b_next)
            else
                #p > 0 concave up
                x_star_num = (b_current^2 - c_current^2)*F_a + (c_current^2 - a_current^2)*F_b + (a_current^2 - b_current^2)*F_c
                x_star_denom = (b_current - c_current)*F_a + (c_current - a_current)*F_b + (a_current - b_current)*F_c
                x_star = (1/2) * x_star_num / x_star_denom

                debug(LOGGER, @sprintf "p > 0, with x_star = %5.3f" x_star)

                if (x_star - b_current) > delta_max #Quadratic minimum is too far
                    debug(LOGGER, @sprintf "Quadratic minimum is too far. Moving as far forward as possible." )
                    b_next = b_current + delta_max
                else
                    debug(LOGGER, @sprintf "Quadratic minimum is reachable. (forward)" )
                    b_next = x_star
                end

                a_next = c_current
                c_next = b_current

                (F_a, F_c, F_b) = (F_c, F_b, f(b_next))
            end
        else #Moving Backward
            debug(LOGGER, @sprintf "Moving Backward...")
            if p <= 0 #Quadratic has a maximum, move as far as possible
                debug(LOGGER, @sprintf "Quadratic has a maximum. Move as far back as possible.")
                a_next = b_current - delta_max
                b_next = b_current
                c_next = c_current
```

```
                F_a = f(a_next)
            else
                #p > 0 concave up
                x_star_num = (b_current^2 - c_current^2)*F_a + (c_current^2 - a_current^2)*F_b + (a_current^2 - b_current^2)*F_c
                x_star_denom = (b_current - c_current)*F_a + (c_current - a_current)*F_b + (a_current - b_current)*F_c
                x_star = (1/2) * x_star_num / x_star_denom

                debug(LOGGER, @sprintf "p > 0, with x_star = %5.3f" x_star)

                if (a_current - x_star) > delta_max #Quadratic minimum is too far
                    debug(LOGGER, @sprintf "Quadratic minimum is too far. Moving as far back as possible." )
                    a_next = a_current - delta_max
                else
                    debug(LOGGER, @sprintf "Quadratic minimum is reachable. (backwards)" )
                    a_next = x_star
                end

                b_next = c_current
                c_next = a_current
                (F_b, F_c, F_a) = (F_c, F_a, f(a_next))
            end
        end

        push!(history, N_iterations, convert(Tuple{Float64, Float64, Float64}, (a_next, b_next, c_next)))

        debug(LOGGER, @sprintf "a_next b_next c_next = %5.3f %5.3f %5.3f" a_next b_next c_next)
        debug(LOGGER, @sprintf "End of loop iteration...")
    end

    a_current = a_next
    b_current = b_next
    c_current = c_next

    debug(LOGGER, @sprintf "Exiting with a_current, b_current c_current = %5.3f %5.3f %5.3f" a_current b_current c_current)
    debug(LOGGER, @sprintf "Exiting with F_a F_b F_c = %5.3f %5.3f %5.3f" F_a F_b F_c)

    return (a_current, b_current), history
end

function GoldenSectionSearch(f, a, b, tolerance)
    info(LOGGER, "Inside Golden Section Search")

    TAU = 0.618_033_988_7

    F_a = f(a)
    F_b = f(b)

    c = a + (1 - TAU)*(b - a)
    F_c = f(c)

    d = b - (1 - TAU)*(b - a)
    F_d = f(d)

    a_current = a
    b_current = b
    c_current = c
    d_current = d

    a_next = a
    b_next = b
    c_next = c
    d_next = d

    history = History(Tuple{Float64, Float64, Float64, Float64})
    push!(history, 0, convert(Tuple{Float64, Float64, Float64, Float64}, (a_current, b_current, c_current, d_current)))

    N_iterations = 0
    interval_size = abs(b - a)
    while !(interval_size < tolerance)
        a_current = a_next
        b_current = b_next
        c_current = c_next
        d_current = d_next
        N_iterations += 1

        debug(LOGGER, @sprintf "Start of loop iteration... (a_current, b_current, c_current, d_current) = (%5.3f %5.3f %5.3f %5.3f)" 
        debug(LOGGER, @sprintf "Start of loop iteration... (F_a, F_b, F_c, F_d) = (%5.3f %5.3f %5.3f %5.3f)" F_a F_b F_c F_d)

        if F_c < F_d
            debug(LOGGER, "F_c < F_d case")
```

```
            a_next = a_current
            b_next = d_current
            c_next = a_next + (1 - TAU)*(b_next - a_next)
            d_next = c_current

            (F_a, F_b) = (F_a, F_d)
            (F_c, F_d) = (f(c_next), F_c)

        else
            debug(LOGGER, "F_c > F_d case")
            a_next = c_current
            b_next = b_current
            c_next = d_current
            d_next = b_next - (1 - TAU)*(b_next - a_next)

            (F_a, F_b) = (F_c, F_b)
            (F_c, F_d) = (F_d, f(d_next))

        end

        push!(history, N_iterations, convert(Tuple{Float64, Float64, Float64, Float64}, (a_next, b_next, c_next, d_next)))
        interval_size = abs(b_next - a_next)
        debug(LOGGER, @sprintf "End of loop iteration... interval_size = %5.3f" interval_size)
    end

    a_current = a_next
    b_current = b_next
    c_current = c_next
    d_current = d_next

    info(LOGGER, @sprintf "Exiting Golden Section Search with a_current, b_current = %5.3f %5.3f" a_current b_current)
    info(LOGGER, @sprintf "Exiting Golden Section Search with F_a, F_b = %5.3f %5.3f" F_a F_b)
    return (a_current, b_current), history

end

function Q1LineSearch(f, d, x_0, desired_interval_size; linesearch_method = "")
    info(LOGGER, "Entering Q1LineSearch...")
    info(LOGGER, @sprintf "Entering with d = %s" d)
    info(LOGGER, @sprintf "Entering with x_0 = %s" x_0)

    one_dimensional_function = alpha -> f((x_0 .+ alpha .* d))

    bracketing_history = undef
    golden_history = undef
    a_l_smaller, a_u_smaller = undef, undef
    if linesearch_method == "SwannsBracketingMethod"
        swanns_step_length = 1 #HARDCODED
        alpha_init = 0 #HARDCODED
        (alpha_lower, alpha_upper), swanns_history = SwannsBracketingMethod(one_dimensional_function, alpha_init, swanns_step_length)
        (a_l_smaller, a_u_smaller), golden_history = GoldenSectionSearch(one_dimensional_function, alpha_lower, alpha_upper, desired_
        bracketing_history = swanns_history
    elseif linesearch_method == "PowellsBracketingMethod"
        alpha_init = 0 #HARDCODED
        powells_delta = 1 #HARDCODED
        powells_delta_max = 16 #HARDCODED
        (alpha_lower, alpha_upper), powell_history = PowellsBracketingMethod(one_dimensional_function, alpha_init, powells_delta, pow
        (a_l_smaller, a_u_smaller), golden_history = GoldenSectionSearch(one_dimensional_function, alpha_lower, alpha_upper, desired_
        bracketing_history = powell_history
    else
        error(LOGGER, "Line Search Method not recognized: $linesearch_method")
    end

    a_mid = (a_l_smaller + a_u_smaller) / 2 #along line
    debug(LOGGER, @sprintf "a_middle %5.3f" a_mid)

    full_middle_point = @. x_0 + a_mid * d #In full N-D space
    info(LOGGER, @sprintf "Exiting Q1LineSearch with middle point in N-D as %s" full_middle_point)
    return full_middle_point, bracketing_history, golden_history
end

function Q2SteepestDescent(f, grad_f, x_0, tolerance_for_1D_search; linesearch_method = "")
    info(LOGGER, "Entering Q2SteepestDescent...")
    info(LOGGER, @sprintf "Entering with x_0 = %s" x_0)

    x_0 = convert(Array{Float64, 1}, x_0)

    current_point = x_0
    next_point = x_0
    steepest_descent_direction = -1 * grad_f(x_0)
```

```
    Q2_history = MVHistory()
    push!(Q2_history, :Nd_point, 0, current_point)

    N_iterations = 0
    while !(norm(steepest_descent_direction) < 10^(-4))
        info(LOGGER, @sprintf "Q2 Start of Loop Iteration... current_point = %s" current_point)
        info(LOGGER, @sprintf "Q2 Start of Loop Iteration... steepest = %s" steepest_descent_direction)
        N_iterations += 1

        next_point, bracketing_history, golden_history = Q1LineSearch(f, steepest_descent_direction, current_point, tolerance_for_1D_
linesearch_method)

        steepest_descent_direction = -1 * grad_f(next_point)
        current_point = next_point

        push!(Q2_history, :Nd_point, N_iterations, current_point)
        push!(Q2_history, :bracketing_history, N_iterations, bracketing_history)
        push!(Q2_history, :golden_history, N_iterations, golden_history)
        debug(LOGGER, @sprintf "End of Loop Iteration... norm of grad %s" norm(steepest_descent_direction))
    end

    info(LOGGER, @sprintf "Exiting with next_point = %s" next_point)
    return next_point, Q2_history

end

function HookeJeeves(f, x_0, big_delta, small_delta, orthogonal_directions)
    info(LOGGER, "Entering Hooke-Jeeves...")
    @assert length(x_0) == length(orthogonal_directions)

    x_i_array = zeros(length(x_0), length(orthogonal_directions)+1)

    x_0_current = x_0
    x_i_array[:, 1] = x_0

    history = MVHistory()
    push!(history, :x_1, 0, x_i_array[:, 1])
    push!(history, :x_0_current, 0, x_0_current)

    iteration_number = 0
    while !(big_delta < small_delta)
        debug(LOGGER, "Start of Iteration Loop...")
        iteration_number += 1

        #Exploratory Moves
        debug(LOGGER, "Performing Exploratory Moves...")
        for (index, direction) in enumerate(orthogonal_directions)
            x_current = x_i_array[:, index] .+ big_delta .* direction

            F_x_i =  f(x_i_array[:, index])
            if f(x_current) < F_x_i
                x_i_array[:, index+1] = x_current
            else
                x_current = x_i_array[:, index] .- big_delta .* direction

                if f(x_current) < F_x_i
                    x_i_array[:, index+1] = x_0_current
                else
                    x_i_array[:, index+1] = x_i_array[:, index]
                end
            end
        end

        # Pattern Move
        debug(LOGGER, "Performing Pattern Move...")
        if f(x_i_array[:, end]) < f(x_0_current)
            x_i_array[:, 1] = x_i_array[:, end] .+ (x_i_array[:, end] .- x_0_current)
            x_0_current = x_i_array[:, end]

        elseif x_i_array[:, 1] == x_0_current
            big_delta = big_delta / 10.

        else
            x_i_array[:, 1] = x_0_current

        end

        push!(history, :x_1, iteration_number, x_i_array[:, 1])
        push!(history, :x_0_current, iteration_number, x_0_current)
        push!(history, :x_end, iteration_number, x_i_array[:, end])
        debug(LOGGER, "End of Iteration Loop...")
```

```
    end

    info(LOGGER, "Exiting Hooke-Jeeves...")
    return x_i_array[:, 1], history
end

end
```

B.2 Plot Generation Script (makeplots.jl)

```
using Revise

using ForwardDiff # Library for Extra Plot - Calculating Gradients and Hessians Automatically from Code (not symbolic!)
using Plots # Library for plotting
using Printf # Library for formatting strings
using LinearAlgebra # Library for calculating norm and linear algebra
using LaTeXStrings # Library for supporting LaTeX symbols in strings

includet("A1_module/A1Module.jl")
using .A1Module # Custom Library written for Assignment. Written by Kim Laberinto

const ROSENBROCK_A = 1.0
const ROSENBROCK_B = 0.1

N_f_eval = 0
N_grad_f_eval = 0

function rosenbrock_banana(input; a = ROSENBROCK_A, b = ROSENBROCK_B)
    global N_f_eval += 1
    x = input[1]
    y = input[2]
    return (a - x)^2 + 100 * b * (y - x^2)^2
end

function grad_rosenbrock_banana(input; a = ROSENBROCK_A, b = ROSENBROCK_B)
    global N_grad_f_eval += 1
    x = input[1]
    y = input[2]

    grad_x = -2*a - 400*b*x*(y-x^2) + 2*x # 2*(a - x)*(-1) + (100*b)*(2)*(y-x^2)*(-2*x)
    grad_y = 200*b*(y-x^2) #(100*b)*(2)*(y-x^2)*(1)
    return [grad_x, grad_y]
end

# Block to plot the "All lines" plot for Q1
begin
    x_plot = -2:0.01:2
    y_plot = -2:0.01:3
    contour(x_plot, y_plot, (x, y) -> rosenbrock_banana([x, y]),
        levels = 0:5:400,
        fill=false,
        label = "Rosenbrock",
        c=:black,
        aspect_ratio = :equal,
        size = (600, 600),
        legend = :bottomright)

    title!("Example 1D Line Searches on Objective Function")

    plot!([0, 2], [-2, 4], label="A", lw = 3)
    plot!([-2, 4], [2, 0], label="B", lw = 3)
    plot!([-2, 2], [-2, 3], label="C", lw = 3)

    scatter!([1], [1], label="Global Min", shape = :diamond, markersize = 10)

    xlims!(-2, 2)
    ylims!(-2, 3)

    savefig("A1/assets/AllLines.svg")
end

# Function for plot file
function make_bracketing_plot(string_for_plotfile, letter, initial_point, away_point, alpha_minmax, golden_plot_string)
    l = @layout [a{0.45w} [b{0.2h} ; c{0.6h}; d; e]]
```

```
    LINE_START = initial_point #Custom
    LINE_DIRECTION = (away_point .- initial_point) #custom
    LINE_DIRECTION = LINE_DIRECTION / norm(LINE_DIRECTION)
    POINT_PLUS_ALPHA = LINE_START .+ 1 .* LINE_DIRECTION
    OneD_function = alpha -> rosenbrock_banana(LINE_START .+ alpha .* LINE_DIRECTION)

    begin
        x_plot = -2:0.01:2
        y_plot = -2:0.01:3
        p1 = contour(x_plot, y_plot, (x, y) -> rosenbrock_banana([x, y]),
            levels = 0:5:400,
            fill=false,
            label = "Rosenbrock",
            c=:black,
            legend = :bottomright)

        plot!([initial_point[1], away_point[1]], [initial_point[2], away_point[2]], label=letter, lw = 3, title="Line $letter Initial
        scatter!([LINE_START[1]],[LINE_START[2]], label="$letter Init", shape=:circle, markersize = 9)
        scatter!([POINT_PLUS_ALPHA[1]],[POINT_PLUS_ALPHA[2]], label="1 α Away", shape=:circle, markersize = 9)
        println(POINT_PLUS_ALPHA)
        xlims!(-2, 2)
        ylims!(-2, 3)
    end

    #Plot 1D
    begin
        xs = alpha_minmax[1]:0.01:alpha_minmax[2]
        ys = @. OneD_function(xs)
        p_1D = plot(xs,ys, legend=false, title="1D Function")
        xlims!(p_1D, minimum(xs), maximum(xs))
    end

    #Plot 1D Log
    begin
        xs = alpha_minmax[1]:0.01:alpha_minmax[2] #Custom
        ys = @. log10(OneD_function(xs))
        p_1Dlog = plot(xs,ys, legend=false, title = "Log10 1D Function")
        xlims!(p_1Dlog, minimum(xs), maximum(xs))
    end

    # Plot Powell
    begin
        desired_interval_size = 1e-3
        _, powells_bracketing_history, golden_powell_history = Q1LineSearch(rosenbrock_banana, LINE_DIRECTION, LINE_START, desired_in
"PowellsBracketingMethod")

        num_iterations_powell = length(powells_bracketing_history)

        p_powell = plot(title="$num_iterations_powell Iterations for Powell")
        for (i, interval) in enumerate(powells_bracketing_history)
            plot!([interval[1], interval[2], interval[3]], [-i, -i, -i], label="$i", shape=:circle, markersize=4, ytick=[], legend=fa
        end
        xlims!(p_powell, minimum(xs), maximum(xs))
        ylims!(p_powell, -1*(length(powells_bracketing_history)), 1)

    end

    # Plot Swanns
    begin
        _, swanns_bracketing_history, golden_swanns_history  = Q1LineSearch(rosenbrock_banana, LINE_DIRECTION, LINE_START, desired_in
"SwannsBracketingMethod")

        num_iterations_swanns = length(swanns_bracketing_history)

        p_swanns = plot(title="$num_iterations_swanns Iterations for Swanns")
        for (i, interval) in enumerate(swanns_bracketing_history)
            plot!([interval[1], interval[2], interval[3]], [-i, -i, -i], label="$i", shape=:circle, markersize=4, ytick=[], legend=fa
        end
        xlims!(p_swanns, minimum(xs), maximum(xs))
        ylims!(p_swanns, -1*(length(swanns_bracketing_history)), 1)

    end

    xlabel!("Alpha (α)")
    plot(p1, p_1D, p_1Dlog, p_powell, p_swanns, layout=l, size=(800, 500))
    savefig("A1/assets/$string_for_plotfile.svg")

    layout_golden = @layout [a{0.2h} ; [b ; c] [d ; e]]

    #Plot Golden for Powell
    begin
```

```
        _, initial_interval = first(golden_powell_history)
        print("Golden Initial = $initial_interval")
        xs = range(initial_interval[1], initial_interval[2], length=100)
        ys = @. log10(OneD_function(xs))
        plot_zoomed_powell  = plot(xs, ys, legend=false, title="Interval found by Powell")
        xlims!(plot_zoomed_powell, minimum(xs), maximum(xs))
    end
    begin
        num_iterations_powell_gold = length(golden_powell_history)

        p_powell_goldensearch = plot(title="$num_iterations_powell_gold Golden Section Search (Powell)")
        for (i, interval) in enumerate(golden_powell_history)
            plot!([interval[1], interval[2], interval[3], interval[4]], [-i, -i, -i, -i], label="$i", shape=:circle, markersize=4, yt
        end
        xlims!(p_powell_goldensearch, minimum(xs), maximum(xs))
        ylims!(p_powell_goldensearch, -1*(length(golden_powell_history)), 1)
    end

    begin
        _, initial_interval = first(golden_swanns_history)
        print("Golden Initial = $initial_interval")
        xs = range(initial_interval[1], initial_interval[2], length=100)
        ys = @. log10(OneD_function(xs))
        plot_zoomed_swanns  = plot(xs, ys, legend=false, title="Interval found by Swanns")
        xlims!(plot_zoomed_swanns, minimum(xs), maximum(xs))
    end
    begin
        num_iterations_powell_gold = length(golden_swanns_history)

        p_swanns_goldensearch = plot(title="$num_iterations_powell_gold Golden Section Search (Swanns)")
        for (i, interval) in enumerate(golden_swanns_history)
            plot!([interval[1], interval[2], interval[3], interval[4]], [-i, -i, -i, -i], label="$i", shape=:circle, markersize=4, yt
        end
        xlims!(p_swanns_goldensearch, minimum(xs), maximum(xs))
        ylims!(p_swanns_goldensearch, -1*(length(golden_swanns_history)), 1)
    end
    title!(p_1Dlog, "Log10 1D Function for Line $letter")
    plot(p_1Dlog, plot_zoomed_powell, p_powell_goldensearch, plot_zoomed_swanns, p_swanns_goldensearch, layout=layout_golden, size=(8
    xlabel!("Alpha (α)")
    savefig("A1/assets/$golden_plot_string.svg")
end

# Block for Q1 Plots
begin
    make_bracketing_plot("LineA_initialbracketing", "A", [0, -2], [2, 4], [-2, 8], "LineA_GoldenComparison")
    make_bracketing_plot("LineB_initialbracketing", "B", [-2, 2], [4, 0], [-2, 8], "LineB_GoldenComparison")
    make_bracketing_plot("LineC_initialbracketing", "C", [-2, -2], [2, 3], [-2, 8], "LineC_GoldenComparison")
end

# Block for Q2 Plots - Gradient Descent
begin
    x_plot = -2:0.01:2
    y_plot = -2:0.01:3

    plot_Q2_swanns = contour(x_plot, y_plot, (x, y) -> rosenbrock_banana([x, y]),
        levels = 0:5:400,
        fill=false,
        label = "Rosenbrock",
        c=:black,
        legend = :bottomright)
    plot_Q2_powells = contour(x_plot, y_plot, (x, y) -> rosenbrock_banana([x, y]),
        levels = 0:5:400,
        fill=false,
        label = "Rosenbrock",
        c=:black,
        legend = :bottomright)

    plot_Q2_swanns_lowtol = contour(x_plot, y_plot, (x, y) -> rosenbrock_banana([x, y]),
        levels = 0:5:400,
        fill=false,
        label = "Rosenbrock",
        c=:black,
        legend = :bottomright)
    plot_Q2_powells_lowtol = contour(x_plot, y_plot, (x, y) -> rosenbrock_banana([x, y]),
        levels = 0:5:400,
        fill=false,
        label = "Rosenbrock",
        c=:black,
        legend = :bottomright)

    plot_Q2_loss_vs_iter_swanns = plot()
```

```
    plot_Q2_loss_vs_iter_powells = plot()
    plot_Q2_loss_vs_iter_swanns_lowtol = plot()
    plot_Q2_loss_vs_iter_powells_lowtol = plot()

    N_f_eval = 0
    N_grad_f_eval = 0
    tolerance = 1e-4
    low_tolerance = 1e-2
    points = [[-2, -2], [-1.5, 1.5], [-1, 3], [-0.5, -1.5], [2, 2]]
    labels = ["D", "E", "F", "G", "H"]
    for (i, (init_point, descent_label)) in enumerate(zip(points, labels))
        result, history = Q2SteepestDescent(rosenbrock_banana, grad_rosenbrock_banana, init_point, tolerance;  linesearch_method =
"SwannsBracketingMethod")

        _, all_points = get(history, :Nd_point)
        all_points = transpose(hcat(all_points...))
        plot!(plot_Q2_swanns, all_points[:, 1], all_points[:, 2], label = descent_label, shape=:circle, markersize = 3, lw=3, color=i

        iterations, all_points = get(history, :Nd_point)
        loss = []
        for point2d in all_points
            push!(loss, rosenbrock_banana(point2d))
        end
        plot!(plot_Q2_loss_vs_iter_swanns, iterations, loss, label = descent_label, lw=3, color=i)
    end
    title!(plot_Q2_swanns, "Gradient Descent\nSwanns")
    title!(plot_Q2_loss_vs_iter_swanns, "Obj. Func. Progression\nSwanns")

    for (i, (init_point, descent_label)) in enumerate(zip(points, labels))
        result, history = Q2SteepestDescent(rosenbrock_banana, grad_rosenbrock_banana, init_point, tolerance;  linesearch_method =
"PowellsBracketingMethod")

        _, all_points = get(history, :Nd_point)
        all_points = transpose(hcat(all_points...))
        plot!(plot_Q2_powells, all_points[:, 1], all_points[:, 2], label = descent_label, shape=:circle, markersize = 3, lw=3, color=

        iterations, all_points = get(history, :Nd_point)
        loss = []
        for point2d in all_points
            push!(loss, rosenbrock_banana(point2d))
        end
        plot!(plot_Q2_loss_vs_iter_powells, iterations, loss, label = descent_label, lw=3, color=i)
    end
    title!(plot_Q2_powells, "Gradient Descent\nPowells")
    title!(plot_Q2_loss_vs_iter_powells, "Obj. Func. Progression\nPowell")

    for (i, (init_point, descent_label)) in enumerate(zip(points, labels))
        result, history = Q2SteepestDescent(rosenbrock_banana, grad_rosenbrock_banana, init_point, low_tolerance;  linesearch_method =
"SwannsBracketingMethod")

        _, all_points = get(history, :Nd_point)
        all_points = transpose(hcat(all_points...))
        plot!(plot_Q2_swanns_lowtol, all_points[:, 1], all_points[:, 2], label = descent_label, shape=:circle, markersize =
3, lw=3, color=i)

        iterations, all_points = get(history, :Nd_point)
        loss = []
        for point2d in all_points
            push!(loss, rosenbrock_banana(point2d))
        end
        plot!(plot_Q2_loss_vs_iter_swanns_lowtol, iterations, loss, label = descent_label, lw=3, color=i)
    end
    title!(plot_Q2_swanns_lowtol, "Gradient Descent (low tol.)\nSwanns")
    title!(plot_Q2_loss_vs_iter_swanns_lowtol, "Obj. Func. Progression\nSwanns (low tol.)")

    for (i, (init_point, descent_label)) in enumerate(zip(points, labels))
        result, history = Q2SteepestDescent(rosenbrock_banana, grad_rosenbrock_banana, init_point, low_tolerance;  linesearch_method =
"PowellsBracketingMethod")

        _, all_points = get(history, :Nd_point)
        all_points = transpose(hcat(all_points...))
        plot!(plot_Q2_powells_lowtol, all_points[:, 1], all_points[:, 2], label = descent_label, shape=:circle, markersize =
3, lw=3, color=i)

        iterations, all_points = get(history, :Nd_point)
        loss = []
        for point2d in all_points
            push!(loss, rosenbrock_banana(point2d))
        end
        plot!(plot_Q2_loss_vs_iter_powells_lowtol, iterations, loss, label = descent_label, lw=3, color=i)
    end
```

```
    title!(plot_Q2_powells_lowtol, "Gradient Descent (low tol.)\nPowells")
    title!(plot_Q2_loss_vs_iter_powells_lowtol, "Obj. Func. Progression\nPowells (low tol.)")

    layout_Q2 =  @layout [a b; c d]
    plot(plot_Q2_swanns, plot_Q2_powells, plot_Q2_swanns_lowtol, plot_Q2_powells_lowtol, layout = layout_Q2, size=(1000, 1000))
    savefig("A1/assets/Q2_stepsvisualized.svg")

    layout_Q2_loss_vs_steps = @layout [a b; c d]
    plot(plot_Q2_loss_vs_iter_swanns, plot_Q2_loss_vs_iter_powells, plot_Q2_loss_vs_iter_swanns_lowtol, plot_Q2_loss_vs_iter_powells_
layout_Q2_loss_vs_steps, legend=true, size=(650, 650))
    plot!(yscale=:log10)
    xlabel!("Number of Gradient Steps")
    ylabel!("Objective Function Value (log scale)")
    savefig("A1/assets/Q2_loss_vs_steps.svg")
end

# Block for Q2 Plots - HookeJeeves
begin
    x_plot = -2:0.01:2
    y_plot = -2:0.01:3

    plot_Q2_HJ = contour(x_plot, y_plot, (x, y) -> rosenbrock_banana([x, y]),
        levels = 0:5:400,
        fill=false,
        label = "Rosenbrock",
        c=:black,
        legend = :bottomright)

    plot_Q2_loss_vs_iter_HJ = plot()

    points = [[-2, -2], [-1.5, 1.5], [-1, 3], [-0.5, -1.5], [2, 2]]
    labels = ["D", "E", "F", "G", "H"]
    for (i, (init_point, descent_label)) in enumerate(zip(points, labels))
        result, history = HookeJeeves(rosenbrock_banana, init_point, .2, 0.001, [[1, 0], [0, 1]])

        _, all_points = get(history, :x_1)
        all_points = transpose(hcat(all_points...))
        plot!(plot_Q2_HJ, all_points[:, 1], all_points[:, 2], label = descent_label, shape=:circle, markersize = 3, lw=3, color=i)

        iterations, all_points = get(history, :x_1)
        loss = []
        for point2d in all_points
            push!(loss, rosenbrock_banana(point2d))
        end
        min = minimum(loss)
        println("Minimum ($descent_label) : $min")
        plot!(plot_Q2_loss_vs_iter_HJ, iterations, loss, yscale = :log10, label = descent_label, lw=3, color=i, shape=:circle, marker
3)
    end
    title!(plot_Q2_HJ, "Hooke-Jeeves Steps")
    title!(plot_Q2_loss_vs_iter_HJ, "Obj. Func. Progression\nHooke-Jeeves")
    xlabel!(plot_Q2_loss_vs_iter_HJ, "Number of HJ Outer-loop Iterations")
    ylabel!(plot_Q2_loss_vs_iter_HJ, "Rosenbrock Banana Function (log10)")


    layout_Q2_HJ =  @layout [a; b]
    plot(plot_Q2_HJ, plot_Q2_loss_vs_iter_HJ, layout = layout_Q2_HJ, size = (700, 800))
    savefig("A1/assets/Q2_HookeJeeves_visualized.svg")
end

# Block for Q3 Plots
begin
    ##
    ## SET UP
    ##
    const TIME_DATA = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]
    const Y_DATA = [1.75; 1.65; 1.56; 1.49; 1.43; 1.37; 1.33; 1.29; 1.26; 1.23]

    N_f_eval = 0
    N_grad_f_eval = 0

    function Q3Model(t, params)
        return @. params[1] + params[2]*exp(params[4]*t) + params[3]*exp(params[5]*t)
    end

    function Q3_Grad_Model(t, params)
        gradient = [1;
                    exp(params[4]*t);
                    exp(params[5]*t);
                    params[2]*t*exp(params[4]*t);
                    params[3]*t*exp(params[5]*t) ]
```

```
        return gradient
    end

    function Q3SumSquaredError(params, data_t, data_y)
        global N_f_eval += 1

        squared_errors = (data_y - Q3Model(data_t, params)).^2
        return sum(squared_errors)
    end

    function Q3_Grad_SumSquaredError(params, data_t, data_y)
        global N_grad_f_eval += 1
        gradient = zeros(5)

        for (t_i, y_i) in zip(data_t, data_y)
            gradient += 2*(y_i - Q3Model(t_i, params))*Q3_Grad_Model(t_i, params)
        end
        @assert size(gradient) == size(params)

        return gradient
    end

    ##
    ## Plotting
    ##

    objective_function = params -> Q3SumSquaredError(params, TIME_DATA, Y_DATA)
    grad_objective_function = params -> Q3_Grad_SumSquaredError(params, TIME_DATA, Y_DATA)

    methods_array = ["SwannsBracketingMethod"]
    method_simple_name_array = ["Swanns"] #For Plot Labels
    tolerances_array = [1e-1, 1e-2, 1e-3] #tolerances for line search
    init_params_array = [[0.1, 0.1, 0.1, 0.1, 0.1], [-0.1, -0.1, -0.1, -0.1, -0.1], [0.0, 0.1, -0.1, 0.0, 0.1]]
    for (index_param, init_params) in enumerate(init_params_array)

        outputs_N_f_eval = []
        outputs_graf_f_eval = []
        outputs_final_param_vector = []

        plot_all_tolerances_loss_vs_steps = plot()
        plot_all_curves = plot()
        scatter!(plot_all_curves, TIME_DATA, Y_DATA, label="Data Points", color=4)

        for (method, simple_method_name) in zip(methods_array, method_simple_name_array)
            for (index_tol, tol) in enumerate(tolerances_array)
                global N_f_eval = 0
                global N_grad_f_eval = 0
                result, history = Q2SteepestDescent(objective_function, grad_objective_function, init_params, tol;  linesearch_method
"SwannsBracketingMethod")

                push!(outputs_N_f_eval, N_f_eval)
                push!(outputs_graf_f_eval, N_grad_f_eval)
                push!(outputs_final_param_vector, result)

                begin
                    best_params = result
                    t_bestfit = 0:0.02:9
                    y_bestfit = Q3Model(t_bestfit, best_params)
                    plot!(plot_all_curves, t_bestfit, y_bestfit, label="Best Fit (LS Tol. = $tol)", color=index_tol)
                end

                begin
                    is, points = get(history, :Nd_point)
                    errors = []
                    for (i, current_params) in enumerate(history, :Nd_point)
                        error = Q3SumSquaredError(current_params, TIME_DATA, Y_DATA)
                        push!(errors, error)
                    end
                    plot!(plot_all_tolerances_loss_vs_steps, is, errors, yscale=:log10, lw=3, label="LS Tolerance = $tol", color=inde
                    title!(plot_all_tolerances_loss_vs_steps, "Loss vs Gradient Steps\nInitial Parameters = $init_params")
                    xlabel!(plot_all_tolerances_loss_vs_steps, "Number of Gradient Descent Iterations")
                    ylabel!(plot_all_tolerances_loss_vs_steps, "Sum Squared Error")
                    savefig("A1/assets/Q3_LossVsSteps_$index_param.svg")

                    title!(plot_all_curves, "Best Fits from Optimization\nInitial Parameters = $init_params")
                    xlabel!(plot_all_curves, "t")
                    ylabel!(plot_all_curves, "y")
                    savefig("A1/assets/Q3_BestFits_$index_param.svg")
                end

                output_file = open("A1/assets/Q3_OUTPUTGradient_$index_param.txt", "w")
```

```
                write(output_file, "Initial Parameter Guess for whole file: $init_params\n")
                write(output_file, "Tolerances : $tolerances_array\n")
                write(output_file, "Num Function Evals : $outputs_N_f_eval\n")
                write(output_file, "Num Gradient Steps/Evals : $outputs_graf_f_eval\n")
                write(output_file, "Final Parameter Vectors : $outputs_final_param_vector\n")
                close(output_file)
            end
        end
    end
end

begin
    plot_HJ_lossvssteps = plot()
    title!(plot_HJ_lossvssteps, "Loss vs Gradient Steps\nHooke-Jeeves")
    xlabel!(plot_HJ_lossvssteps, "Hooke-Jeeves Outer-loop Iterations")
    ylabel!(plot_HJ_lossvssteps, "Sum Squared Error")

    plot_HJ_curves_all_inits = plot()
    title!(plot_HJ_curves_all_inits, "Best Fits from Optimization\nHooke-Jeeves")
    xlabel!(plot_HJ_curves_all_inits, "t")
    ylabel!(plot_HJ_curves_all_inits, "y")
    scatter!(plot_HJ_curves_all_inits, TIME_DATA, Y_DATA, label="Data Points", color=4)

    init_params_array = [[0.1, 0.1, 0.1, 0.1, 0.1], [-0.1, -0.1, -0.1, -0.1, -0.1], [0.0, 0.1, -0.1, 0.0, 0.1]]
    HJ_tolerance = 1e-2
    for (index_param, init_params) in enumerate(init_params_array)

        global N_f_eval = 0
        global N_grad_f_eval = 0

        orthogonal_directions = [[1, 0, 0, 0, 0],
                                 [0, 1, 0, 0, 0],
                                 [0, 0, 1, 0, 0],
                                 [0, 0, 0, 1, 0],
                                 [0, 0, 0, 0, 1]]
        result, history = HookeJeeves(rosenbrock_banana, init_params, .1, HJ_tolerance, orthogonal_directions)
        final_param_vector = result

        # Plot curve
        begin
            t_bestfit = 0:0.02:9
            y_bestfit = Q3Model(t_bestfit, final_param_vector)
            plot!(plot_HJ_curves_all_inits, t_bestfit, y_bestfit, label="Best Fit using Guess $index_param", legend=:left, color=inde
        end

        # Plot loss curve
        begin
            is, _ = get(history, :x_1)
            errors = []
            for (i, current_params) in enumerate(history, :x_1)
                error = Q3SumSquaredError(current_params, TIME_DATA, Y_DATA)
                push!(errors, error)
            end
            plot!(plot_HJ_lossvssteps, is, errors, yscale=:log10, lw=3, label="Initial Guess $index_param", color=index_param, shape
:circle, markersize=3)
        end

        output_file = open("A1/assets/Q3HookeJeeves_$index_param.txt", "w")
        write(output_file, "Initial Parameter Guess for whole file: $init_params\n")
        write(output_file, "Num Function Evals : $N_f_eval\n")
        write(output_file, "Num Gradient Steps/Evals : $N_grad_f_eval\n")
        write(output_file, "Final Parameter Vectors : $final_param_vector\n")
        close(output_file)
    end
    savefig(plot_HJ_lossvssteps, "A1/assets/Q3HookeJeeves_LossVsSteps.svg")
    savefig(plot_HJ_curves_all_inits, "A1/assets/Q3HookeJeeves_BestFits.svg")
end

begin
    function autodiff_grad_objective_function(params)
        global N_grad_f_eval += 1
        return ForwardDiff.gradient(objective_function, params)
    end

    plot_loss_manual_vs_autodiff = plot()
    init_params = [0.1, 0.1, 0.1, 0.1, 0.1]
    tol = 1e-2
    title!(plot_loss_manual_vs_autodiff, "Manual vs Auto-Diff Gradient Descent\nInitial Parameters = $init_params")
    xlabel!(plot_loss_manual_vs_autodiff, "Number of Gradient Descent Iterations")
    ylabel!(plot_loss_manual_vs_autodiff, "Sum Squared Error")
    begin
```

```
        global N_f_eval = 0
        global N_grad_f_eval = 0
        result, history = Q2SteepestDescent(objective_function, grad_objective_function, init_params, tol;  linesearch_method =
"SwannsBracketingMethod")
        output_num_eval_manual = N_f_eval
        output_num_grad_eval_manual = N_grad_f_eval
        output_result_manual = result

        is, points = get(history, :Nd_point)
        errors = []
        for (i, current_params) in enumerate(history, :Nd_point)
            error = Q3SumSquaredError(current_params, TIME_DATA, Y_DATA)
            push!(errors, error)
        end
        plot!(plot_loss_manual_vs_autodiff, is, errors, yscale=:log10, lw=3, label="Manual Programming")


    end

    begin
        global N_f_eval = 0
        global N_grad_f_eval = 0
        _, history = Q2SteepestDescent(objective_function, autodiff_grad_objective_function, init_params, tol;  linesearch_method =
"SwannsBracketingMethod")
        output_num_eval_autodiff = N_f_eval
        output_num_grad_eval_autodiff = N_grad_f_eval
        output_result_autodiff = result

        is, points = get(history, :Nd_point)
        errors = []
        for (i, current_params) in enumerate(history, :Nd_point)
            error = Q3SumSquaredError(current_params, TIME_DATA, Y_DATA)
            push!(errors, error)
        end
        plot!(plot_loss_manual_vs_autodiff, is, errors, yscale=:log10, lw=3, label="Automatic Differentiation")
    end

    output_file = open("A1/assets/Q3_Autodiff_Comparison.txt", "w")
    write(output_file, "Initial Parameter Guess for whole file: $init_params\n")
    write(output_file, "Tolerance: $tol\n")
    write(output_file, "\n")
    write(output_file, "Num Function Evals for Manual : $output_num_eval_manual\n")
    write(output_file, "Num Gradient Steps/Evals for Manual : $output_num_grad_eval_manual\n")
    write(output_file, "Final Parameter Vectors for Manual: $output_result_manual\n")
    write(output_file, "\n")
    write(output_file, "Num Function Evals for Autodiff : $output_num_eval_autodiff\n")
    write(output_file, "Num Gradient Steps/Evals for Autodiff : $output_num_grad_eval_autodiff\n")
    write(output_file, "Final Parameter Vectors Autodiff : $output_result_autodiff\n")
    close(output_file)

    savefig(plot_loss_manual_vs_autodiff, "A1/assets/Q3_Autodiff_Comparison_LossVsSteps.svg")
end
```