1. Error in Finite Difference Approximations

a. $given\ f(x) = x^2 \rightarrow f'(x) = x^2$

$$using\ the\ approximation\ f'_{app}(x) = \frac{1}{h}\big(f(x) - f(x-h)\big)$$

$$f'_{app}(x) = \frac{1}{h}(x^2 - (x-h)^2) = \frac{1}{h}\big(x^2 - (x^2 + h^2 - 2xh)\big) = \frac{1}{h}(2xh - h^2) = 2x - h$$

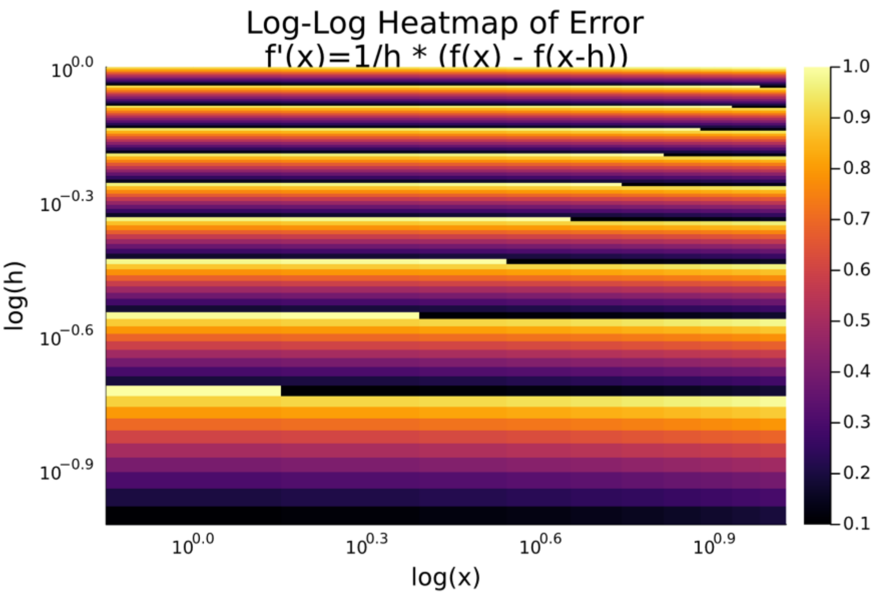
$$err(x,h) = f'(x) - \frac{1}{h}\big(f(x) - f(x-h)\big) = 2x - (2x - h) = \boldsymbol{h}$$

b.

```
## Define error term function for part a
function compute_err_a(x, h)
   return h
end
## Define ranges for x and h
x_range = 1:10
h_range = 0.1:0.01:1.0

## Compute error at each combination of x and h_range
err_a = [compute_err_a(x, h) for x in x_range, h in h_range]

## Plot log-log heatmap
heatmap(x_range, h_range, err_a, xlabel="log(x)", ylabel="log(h)",
       title="Log-Log Heatmap of Error\nf'(x)=1/h * (f(x) - f(x-h))",
       xscale=:log10,yscale=:log10)
```



c) $given\ f(x) = x^2, f'(x) = 2x$:

$$With\ the\ approximation\ f'_{app}(x) = \frac{1}{2h}\big(f(x+h) - f(x-h)\big)$$

$$f'_{app}(x) = \frac{1}{2h}\big(f(x+h) - f(x-h)\big) = \frac{1}{2h}((x+h)^2 - (x-h)^2)$$

$$= \frac{1}{2h}\big((x^2 + 2xh + h^2) - (x^2 - 2xh + h^2)\big)$$

$$= \frac{1}{2h}(x^2 - x^2 + 2xh + 2xh + h^2 - h^2) = \frac{1}{2h}(4xh) = 2x$$

$$err(x,h) = f'(x) - f'_{app}(x) = 2x - 2x = \mathbf{0}$$

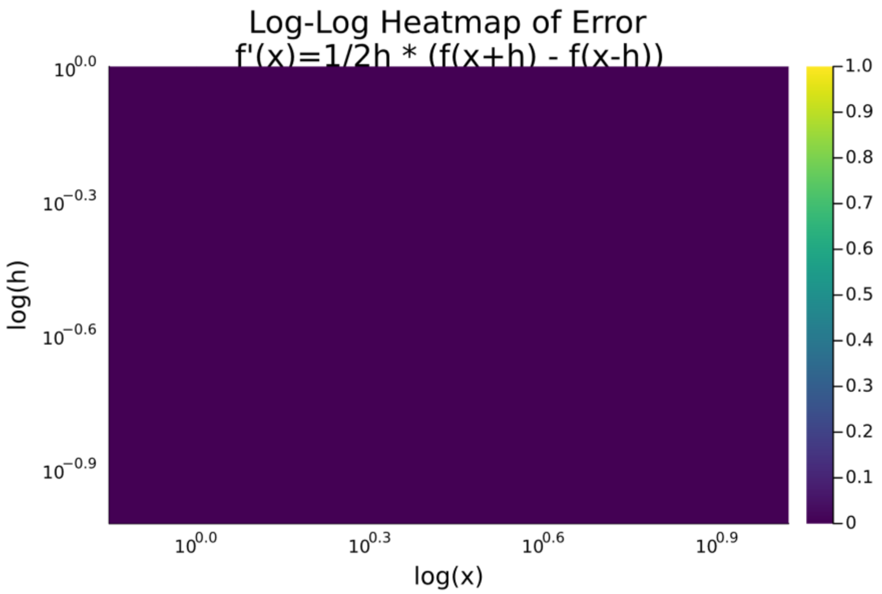
Plotting the heatmap

```
## Define error term function for part c
function compute_err_c(x, h)
   return 0
end

## Define ranges for x and h
x_range = 1:10
h_range = 0.1:0.01:1.0

## Compute error at each combination of x and h_range
err_c = [compute_err_c(x, h) for x in x_range, h in h_range]
```

```
## Plot log-log heatmap
heatmap(x_range, h_range, err_c, xlabel="log(x)", ylabel="log(h)",
        title="Log-Log Heatmap of Error\nf'(x)=1/2h * (f(x+h) - f(x-h))",
        xscale=:log10,yscale=:log10, color=:viridis)
```



2. Using Convex.jl and SCS for Non-negative Least Squares

For this problem, I decided to use the "Wine Quality" dataset from UC Irvine's Machine Learning repository to predict the quality rating of the red variant of the Portuguese "Vinho Verde" wine [1]. This is a great example for a non-negative least squares problem, as the factors considered are physicochemical properties – in other words, the column values are purely positive values.

Using the following code, I solved the function $\arg\min_{x}||Ax - y||_2^2$ using non-negative least squares (NNLS) and unconstrained least squares (ULS) on this dataset to see how well the minimized solution would work on predicting new, labeled values. The difference in application between the two is the constraint of $x \geq 0$, in context of the code, A is replaced with X, the features extracted from the data frame, and x is represented by beta to avoid confusion.

```
## Add required packages
using Statistics, StatsPlots, CSV, DataFrames, Convex, SCS, Random, HypothesisTests

## Import dataset as a DataFrame
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
```

```
wine_df = CSV.read(download(url), DataFrame; delim=';')

## Extract features and target and convert to matrix
X = Matrix{Float64}(wine_df[:, Not("quality")])
y = convert(Vector{Float64}, wine_df[:, "quality"])

## Train-test split
function train_test_split(X, y, train_split=0.7)
    n = size(X, 1)
    idx = idx = Random.shuffle(1:n)
    ## Identify the test and train indices
    train_idx = view(idx, 1:floor(Int, train_split*n))
    test_idx = view(idx, floor(Int, train_split*n)+1:n)

    X_train = X[train_idx, :]
    X_test = X[test_idx, :]
    y_train = y[train_idx]
    y_test = y[test_idx]
    return X_train, X_test, y_train, y_test
end
train_split = 0.8
X_train, X_test, y_train, y_test = train_test_split(X, y, train_split)
```

In this code above, I start the process by extracting the red wine .CSV file from the link, and converting it to a data frame. I also defined a train-test split function and used an 80-20 split of training samples to testing samples. I then defined the variables for the problem as a 1x11 variable and defined both minimization problems. The NNLS problem has the additional constraint of beta being >= 0. We then solve using the SCS.Optimizer, and evaluate both beta variables. The predictions made by using the beta variables are the product of the beta variables and the testing subset, as shown in the code below.

```
## Define variables
β_nnls = Variable(size(X_train, 2))
β_uls = Variable(size(X_train, 2))

## Define problem
nnls_prob = minimize(sumsquares(X_train*β_nnls – y_train), β_nnls >= 0)
uls_prob = minimize(sumsquares(X_train*β_uls – y_train))
```

```
## Solve the problem for training data
nnls_sol = solve!(nnls_prob, SCS.Optimizer)
uls_sol = solve!(uls_prob, SCS.Optimizer)
β_nnls_val = evaluate(β_nnls)
β_uls_val = evaluate(β_uls)

β_nnls_vals = β_nnls.value
β_uls_vals = β_uls.value

## Evaluate againt test data
y_pred_nnls = X_test * β_nnls_val
y_pred_uls = X_test * β_uls_val

## Print coefficients and optimal values
println("Non-Negative Least Squares coefficients:", β_nnls_val)
println("Unconstrained Least Squares coefficients:", β_uls_val)
println("Minimized value of |X*β-y|^2 for NNLS: ", nnls_prob.optval)
println("Minimized value of |X*β-y|^2 for ULS: ", uls_prob.optval)

## Compute MSE for both solutions
mse_nnls = mean((y_test - y_pred_nnls).^2)
mse_uls = mean((y_test - y_pred_uls).^2)
println("MSE for NNLS:", mse_nnls)
println("MSE for ULS:", mse_uls)
println()
```

Executing this results in the following beta coefficients for the NNLS and ULS solutions:

```
Non-Negative Least Squares coefficients: [0.05841462426266634, 0.0015777667575407833,
0.14577558005758115, 0.0034266966333227457, 3.259840390972343e-5,
-0.0037407788616860157, 0.003133485891270362, 0.6449644306447966,
-0.00032092469675769504, 0.8060238191398399, 0.36916382617959026]
Unconstrained Least Squares coefficients: [0.013248225042778053, -1.08049349255975,
-0.19528723443635504, 0.0031471434049468302, -2.0582754449699503, 0.00425183591851079, -
0.003022188257538222, 4.481108703764953, -0.5382351169279859, 0.9118183543725774,
0.29869825527440885]
```

With optimal minimized values of:

```
Minimized value of |y-X*β|^2 for NNLS: 570.9673577815196
Minimized value of |y-X*β|^2 for ULS: 516.7777488086313
```

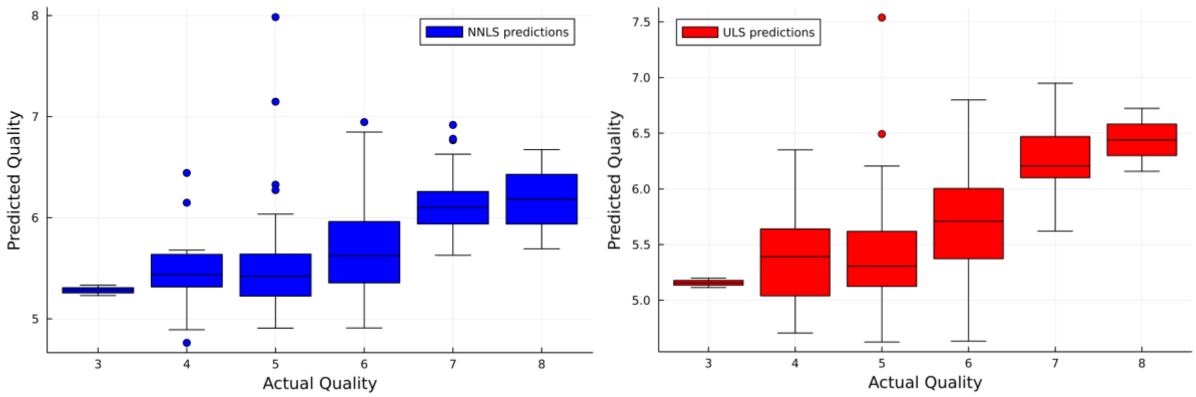
After calculating the MSE for both methods, I found that both methods for this dataset had similar error terms, which indicates that there may not be a significant difference in the results.

```
MSE for NNLS:0.5405554808011127
MSE for ULS:0.4741761656251321
```

Using the following code to plot the results:

```
## Plot results
## Distributions of actual quality vs predicted quality
display(boxplot(y_test, y_pred_nnls, label="NNLS predictions", xlabel="Actual Quality", ylabel="Predicted Quality",
color=:blue))
display(boxplot(y_test, y_pred_uls, label="ULS predictions", xlabel="Actual Quality", ylabel="Predicted Quality",
color=:red))
```

I drew the following boxplots:



On the left, we have the predictions in quality by the NNLS optimizer, and on the right the ULS optimizer. It is apparent from the graphs that there is significant overlap between the two models. I wanted to validate this hypothesis so I used the following code to check the means by using an ANOVA test from the HypothesisTesting library under the following:

$$H_0: mean(NNLS) = mean(ULS), \qquad H_1: mean(NNLS) \neq mean(ULS)$$

```
## Significance testing
OneWayANOVATest(y_pred_nnls, y_pred_uls)

Test summary:
    outcome with 95% confidence: fail to reject h_0
    p-value:                  0.8322
```

This outcome means that the quality predicted by the NNLS and ULS methods are not significantly different, but the minimized values between the two methods are different by approximately 54.

[1] Cortez, Paulo, Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). Wine Quality. UCI Machine Learning Repository. https://doi.org/10.24432/C56S3T.

3. Illustration of Function Extrema

a. Define and illustrate a global function maximizer

A global maximizer of a function f(x) is a point x* in the domain of f such that $f(x^*) \geq f(x)$ for all x in the domain of f. In other words, x* is the point where the function attains its highest value in its entire domain. For example, let's consider the function $f(x) = -x^2 + 4$. We can tell that this function reaches its maximum at x = 0, when f(x) = 4. To illustrate this, let's look at this graph defined in this Julia code:

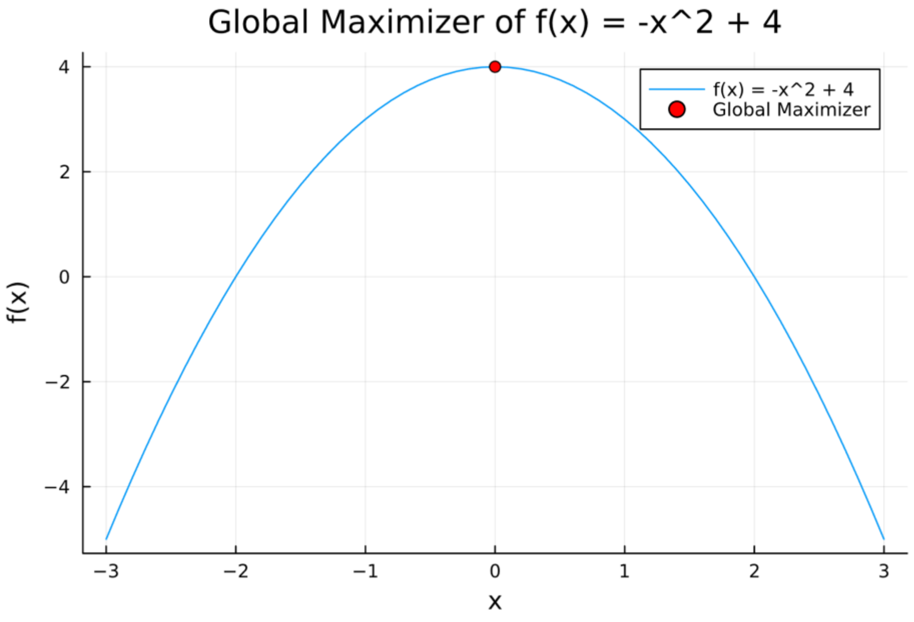
```
using Plots

# Define the function
f(x) = -x^2 + 4

# Generate x values
x = -3:0.1:3
y = f.(x)

# Plot the function
plot(x, y, label="f(x) = -x^2 + 4", xlabel="x", ylabel="f(x)", title="Global Maximizer of f(x) = -x^2 + 4")
scatter!([0], [f(0)], label="Global Maximizer", color=:red, markersize=4)

# Show the plot
display(plot!())
```



The red dot represents the global maximum of the function, as it is the point where the function reaches its peak value of 4.

b. Define and illustrate a local function minimizer

A local minimizer of a function f(x) is a point x* in the domain of f such that there exists a neighborhood around x* where $f(x^*) \leq f(x)$ for all x in that neighborhood. This is distinct from the global minimizer, because while the global minimum is a local minimum, a local minimum may not be the global minimum.

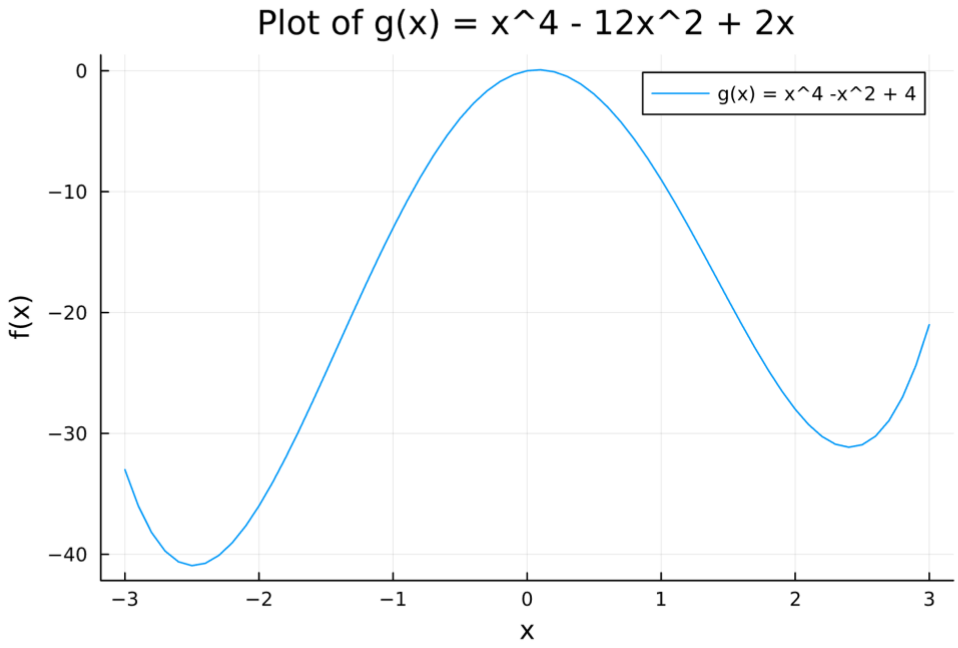
Taking the function $g(x) = x^4 - 12x^2 + 2x$ and using Julia to visualize it, we can observe the following:

```
# Define the function
g(x) = x^4 - 12*x^2 + 2*x

# # Generate x and y values for global function
x = -3:0.1:3
y = g.(x)

# Show the plot
plot(x, y, label="g(x) = x^4 -x^2 + 4", xlabel="x", ylabel="f(x)", title="Plot of g(x) = x^4 - 12x^2 + 2x")

display(plot!())
```



This is a great plot to identify the local minimizers of this function as there are two troughs in the plot. If we take the neighborhoods as being $x_1 = (-3,0)$ $x_2 = (0, 3)$ inclusive, we can then define the local minimizers as the smallest value in each domain. I've implemented that using these lines in the Julia code above

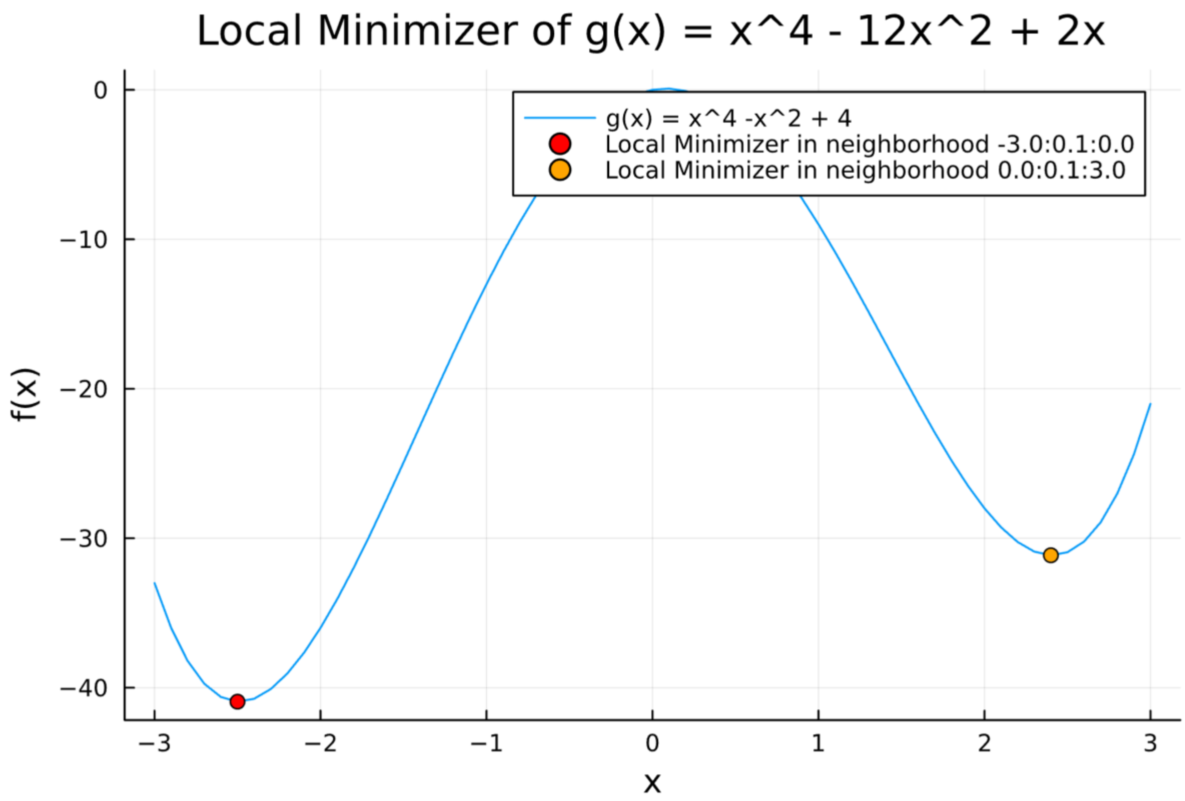
```julia
# define neighborhoods
x1 = -3:0.1:0
y1 = g.(x1)

x2 = 0:0.1:3
y2 = g.(x2)

# Identify local minima
loc1 = x1[findmin(y1)[2]]
loc2 = x2[findmin(y2)[2]]

# Show the plot
plot(x, y, label="g(x) = x^4 -x^2 + 4", xlabel="x", ylabel="f(x)", title="Plot of g(x) = x^4 - 12x^2 + 2x")
scatter!([loc1], [g(loc1)], label="Local Minimizer in neighborhood $x1", color=:red, markersize=4)
scatter!([loc2], [g(loc2)], label="Local Minimizer in neighborhood $x2", color=:orange, markersize=4)

display(plot!())
```



Now when we look at this modified plot, there are two dots to represent the minimizers of the neighborhoods bounded by -3 <= x1 <= 0 and 0 <= x2 <= 3, which we have identified as being at the bottom of the troughs split across the graph. This also illustrates the point of the local minimizers not being the global minimizers, as the orange dot, the minimizer for the non-negative neighborhood, is larger than the minimizer for the negative neighborhood.