schulzd_Lab7

November 9, 2020

1 Lab 7 - Analytical and Numerical Differentiation

David Schulz

1.1 Introduction

The two previous labs focused on exploring cost functions and plotting the error over a given parameter space. We decided that the "solution" for our parameters is a set of parameters that produce the lowest error between our model and the data that we are given. While we used grid search in the previous lab to find these solutions, it should be obvious that grid search itself is a very costly approach. Because of this, we need to find other approaches that make more efficient use of our resources – enter the derivative.

The derivative is a tool that shows how a function is changing. Instead of searching all points over a grid, it would be much more helpful if we had a guide. A guide could tell us "Hey listen! I think the value (error) of our function is decreasing if we head over that way!" The derivative is our guide. We can't always rely on having an analytical solution to the functions that we are investigating. This is one reason that we might want to rely on numerical methods for solving the derivatives of a function. Let's use a trivial function as an example. If we have the following function, $\mathbf{f(x)=x^{2+e}x}$, we can appreciate that this function is dependent on x. To explore this function, we can control x and see how the function responds. In this case we can sweep x over the range 5 x -5. We can then use our calculus toolbox to solve for exact form of the first derivative and subsequently plot the derivative using the same range of x values while solving for f'(x). **f'(x)=2x+e^x**

This lab will explore some interesting properties of the first derivative that we can leverage for our own goals. While these are analytical solutions, we want to build out methods for solving these problems numerically. We can use the following formula to find an approximation of the first derivative without having to find an analytical solution. All we need to do this are values of the function (f(x) and f(x+h)) at two points (x and x+h). We can approximate the first derivative with a numerical solution using the following formula: **f'(x) (f(x+h)-f(x))/h**

Here f(x) refers to a function that is dependent on the x variable. The h (sometimes also represented as Δ) is a constant that we can specify that relates to our resolution, or step size.

1.2 Questions

1. For experiment 1, how did your approximated derivatives (numerical solutions) compare to the analytical solution. Describe what effect the h term had on the accuracy of the approximation.

Discuss considerations/tradeoffs when picking an optimal h value.

- They were the same shape as the analytical derivative, but as the h value increased, the numerical derivative shifted further left of the analytical one. So the larger the h value was, the worse the accuracy was.

2. In experiment 2 you should have found both local extrema of the function. Describe what information the derivative provides and how you found these extrema. If each numerical approximation of the derivative didn't always lead to both extrema of the function, explain why.
   - The derivative describes the slope of its function at a given x value. The min and max extrema occurr when the y-axis of the derivative hits zero. Not all of the numerical approximations lead to the extrema because the approximations with the larger h values shifted the derivative away from the true derivative, making its x values where y was zero incorrect.
3. Be creative - brainstorm a way that you could leverage the information of the derivative to only find the minima, not all extrema.
   - Find the x value where the first derivative is zero AND where its second derivative at that x value is positive.
4. For experiment 3, you performed another grid search. Compare and contrast what features you were looking for in the error-space grid search (lab 06) and the derivative-space grid search. Make sure to compare the computational complexity between these spaces. Depending on your answer do you think it makes more sense to perform a grid search in error space or derivative space?
   - In both labs, the features we searched through were the same: mu and sigma. In Lab 6, we searched for the point in one grid where the error from the parameters at the point was closest to zero. In this lab, we searched for the mu in one grid and the sigma in the other grid where the derivative was closest to zero. The computational complexity is probably larger for performing a grid search in derivative space than error space because instead of every set of parameters having to go through the function just once, every individual parameter in every parameter set has to go through the function twice, once as itself and once after it's slightly adjusted. Because of this, I think it makes more sense to perform a grid search in error space.
5. Be even more creative – brainstorm a way (or algorithm) that you could use to leverage the gradient, without having to use a grid search method.
   - Instead of testing many different combinations of parameters as a grid search, we could start out with an initial set of parameters and then, based on the sign of the resulting gradient, slightly adjust the parameters and repeat until the derivative reaches as close to zero as possible.

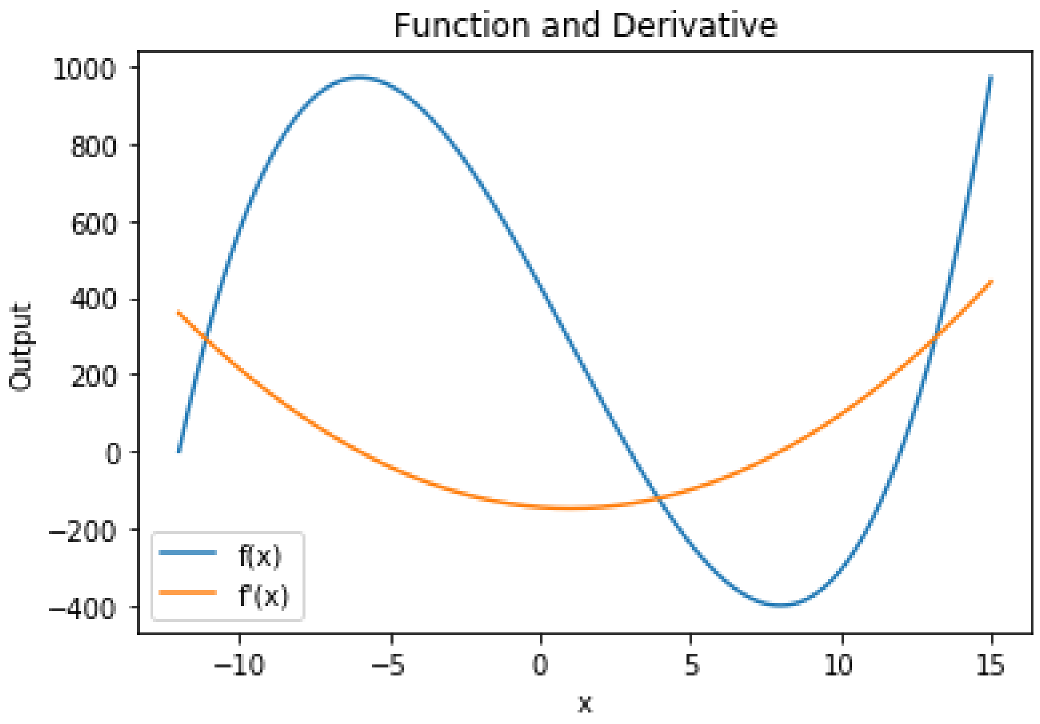
1.3 Experiment 1: Analytical vs Numerical

f'(x)=3x^2-6x-144

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt

def function(x):
    return np.power(x, 3) - (3 * np.power(x, 2)) - (144 * x) + 432
```

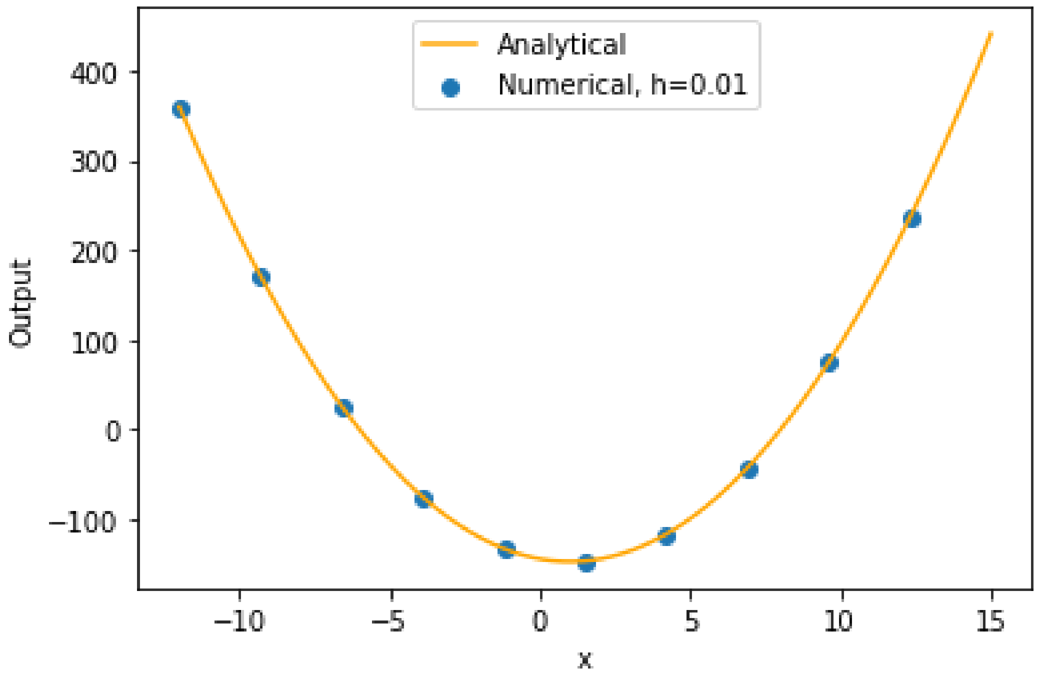
```
x = np.linspace(-12, 15, num=2700)
y = function(x)
an_deriv = (3 * np.power(x, 2)) - (6 * x) - 144
plt.plot(x, y, label='f(x)')
plt.plot(x, an_deriv, label='f\'(x)')
plt.title('Function and Derivative')
plt.xlabel('x')
plt.ylabel('Output')
plt.legend()
```

```
[1]: <matplotlib.legend.Legend at 0x7f8c9813ee90>
```



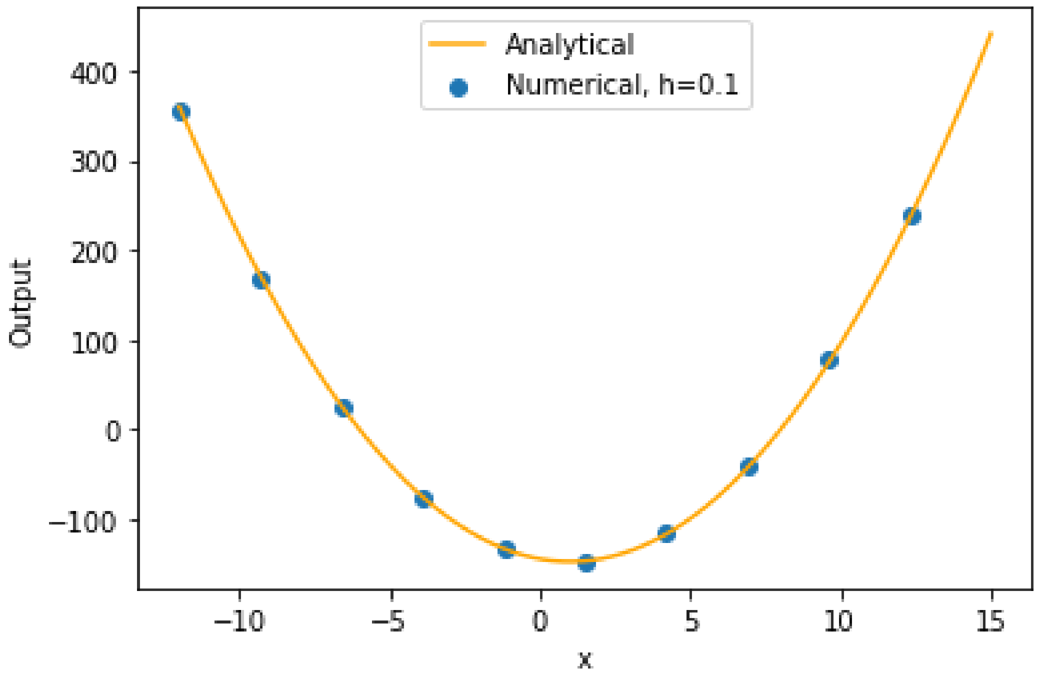
```
[2]: h = 0.01

num_deriv = (function(x+h) - function(x)) / h

plt.scatter(x[::round(len(x)*.1)], num_deriv[::round(len(x)*.1)],␣
 ↪label='Numerical, h=0.01')
plt.plot(x, an_deriv, label='Analytical', c='orange')
plt.xlabel('x')
plt.ylabel('Output')
plt.legend()
```

[2]: <matplotlib.legend.Legend at 0x7f8c8c042a50>



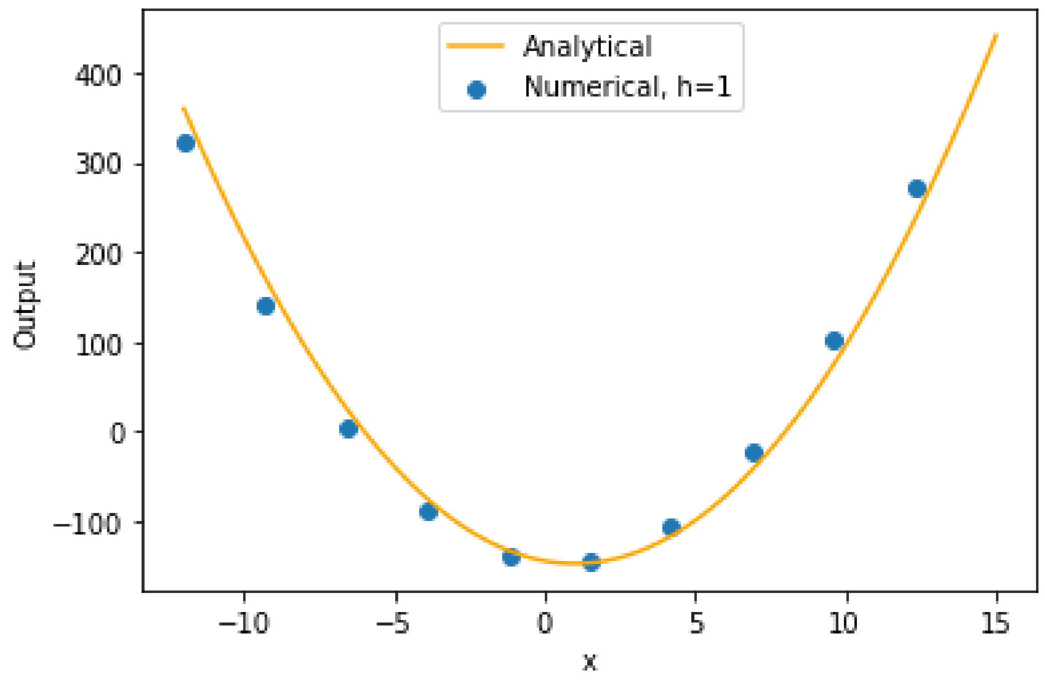
```
[3]: h = 0.1

     num_deriv = (function(x+h) - function(x)) / h

     plt.scatter(x[::round(len(x)*.1)], num_deriv[::round(len(x)*.1)],␣
      ↪label='Numerical, h=0.1')
     plt.plot(x, an_deriv, label='Analytical', c='orange')
     plt.xlabel('x')
     plt.ylabel('Output')
     plt.legend()
```

[3]: <matplotlib.legend.Legend at 0x7f8c8b0d9f90>



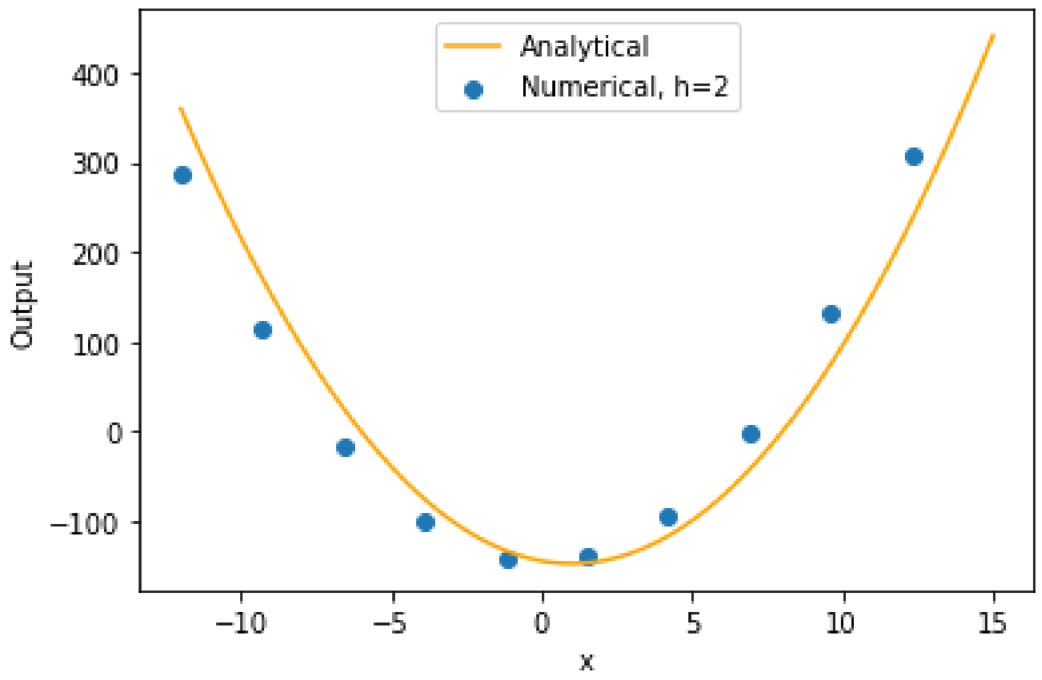
```
[4]: h = 1

     num_deriv = (function(x+h) - function(x)) / h

     plt.scatter(x[::round(len(x)*.1)], num_deriv[::round(len(x)*.1)],␣
      ↪label='Numerical, h=1')
     plt.plot(x, an_deriv, label='Analytical', c='orange')
     plt.xlabel('x')
     plt.ylabel('Output')
     plt.legend()
```

```
[4]: <matplotlib.legend.Legend at 0x7f8c8b04e7d0>
```



```
[5]: h = 2

     num_deriv = (function(x+h) - function(x)) / h

     plt.scatter(x[::round(len(x)*.1)], num_deriv[::round(len(x)*.1)],␣
      ↪label='Numerical, h=2')
     plt.plot(x, an_deriv, label='Analytical', c='orange')
     plt.xlabel('x')
     plt.ylabel('Output')
     plt.legend()
```

```
[5]: <matplotlib.legend.Legend at 0x7f8c8afc9f50>
```



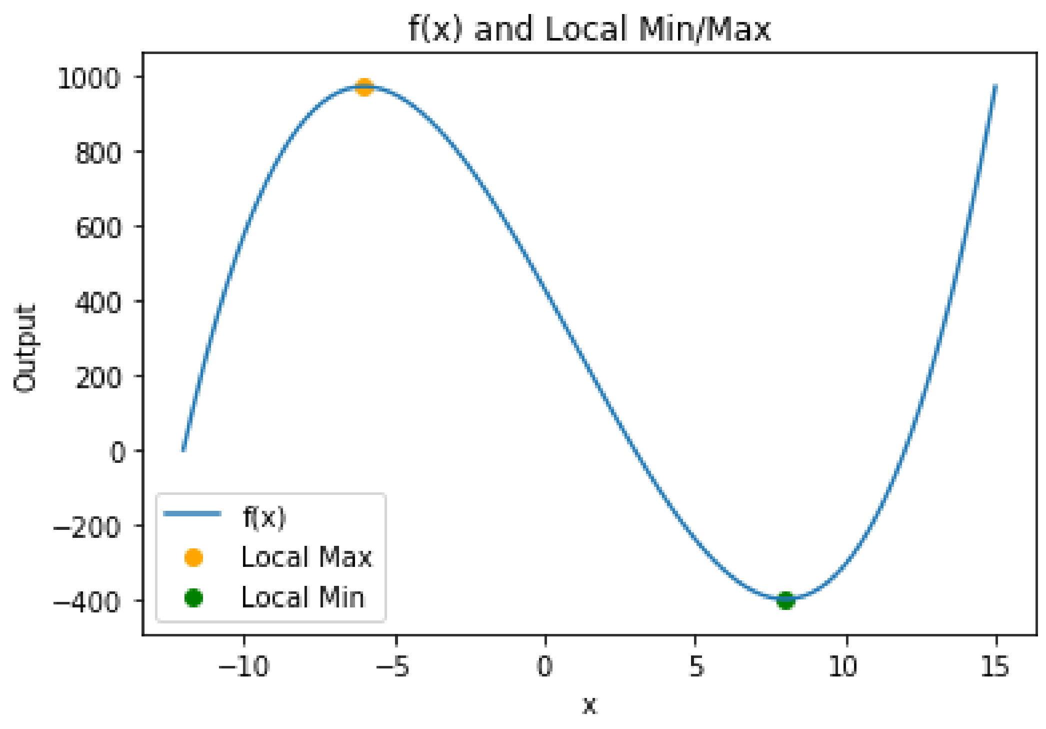
1.4 Experiment 2: 1-Dimensional Grid Search - Derivative Space

f'(x)=3x^2-6x-144=3(x+6)(x-8)

x=-6,8

[6]:
```
plt.plot(x, y, label='f(x)')
plt.scatter(-6, function(-6), c='orange', label='Local Max')
plt.scatter(8, function(8), c='green', label='Local Min')
plt.title('f(x) and Local Min/Max')
plt.xlabel('x')
plt.ylabel('Output')
plt.legend()
```

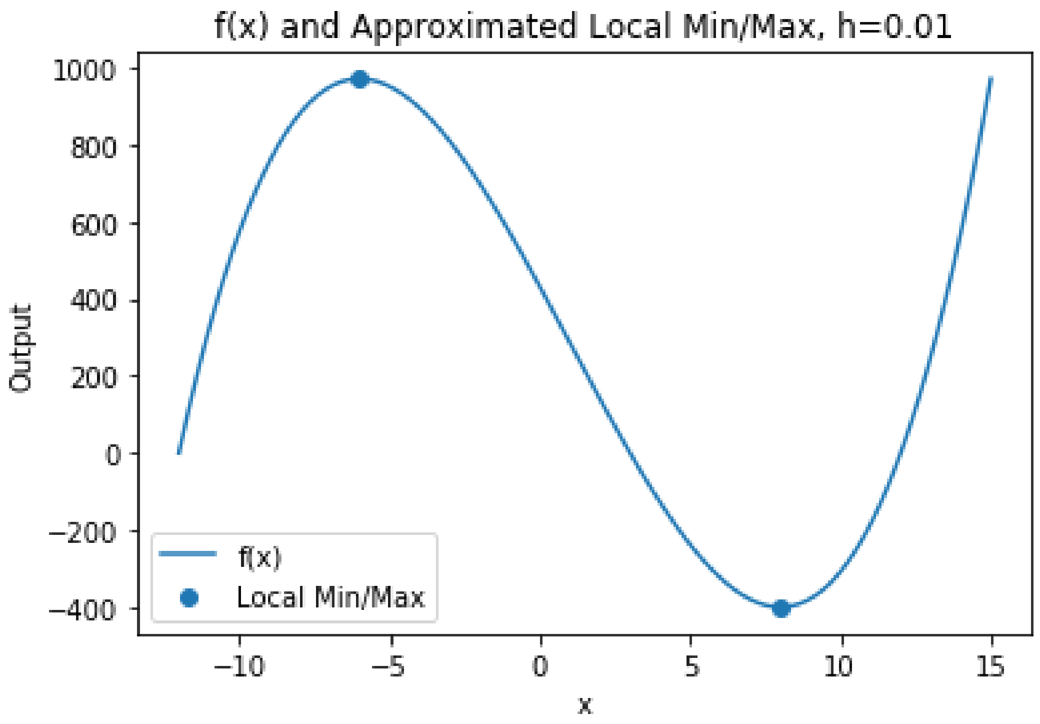
[6]:
```
<matplotlib.legend.Legend at 0x7f8c8af3d950>
```



```
[7]: num_001 = (function(x+0.01) - function(x)) / 0.01
     num_01 = (function(x+0.1) - function(x)) / 0.1
     num_1 = (function(x+1) - function(x)) / 1
     num_2 = (function(x+2) - function(x)) / 2
```

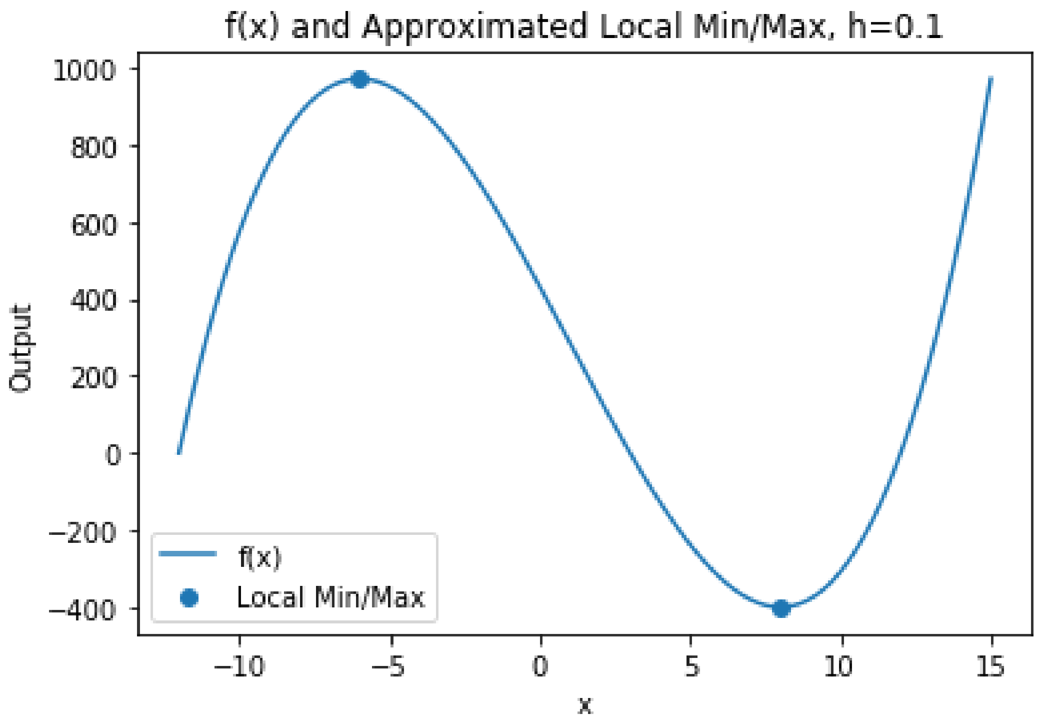
```
[8]: ind = np.argpartition(np.abs(num_001), 2)
     plt.scatter(x[ind[:2]], y[ind[:2]], label='Local Min/Max')
     plt.plot(x, y, label='f(x)')
     plt.title('f(x) and Approximated Local Min/Max, h=0.01')
     plt.xlabel('x')
     plt.ylabel('Output')
     plt.legend()
```

```
[8]: <matplotlib.legend.Legend at 0x7f8c8aeb2a90>
```



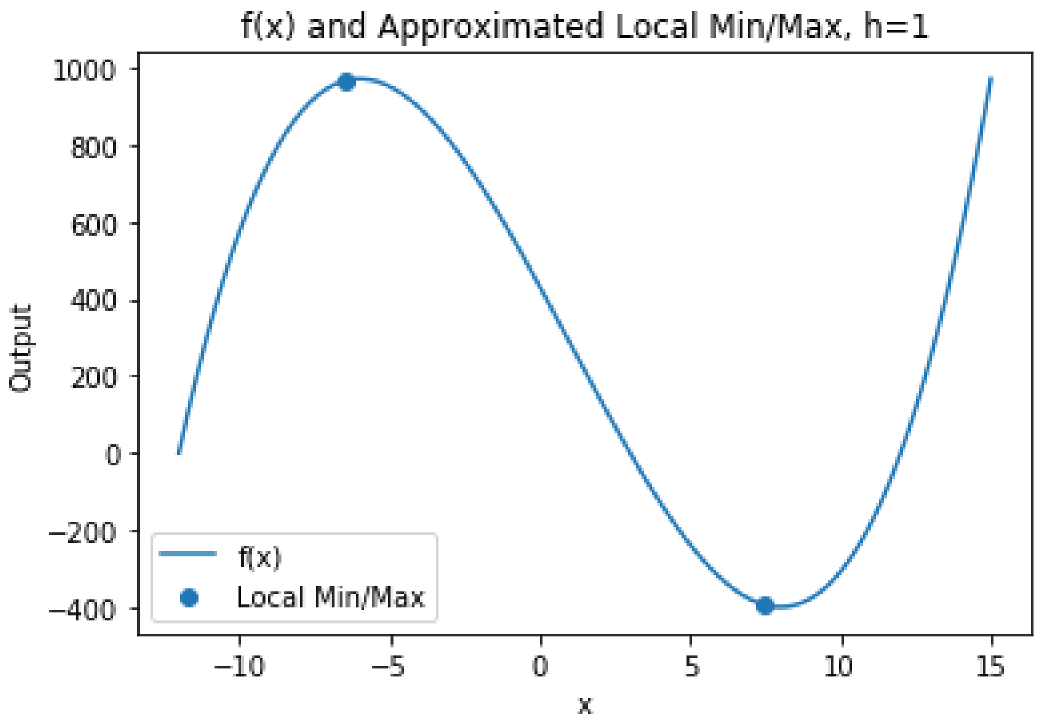
```
[9]: ind = np.argpartition(np.abs(num_01), 2)
     plt.scatter(x[ind[:2]], y[ind[:2]], label='Local Min/Max')
     plt.plot(x, y, label='f(x)')
     plt.title('f(x) and Approximated Local Min/Max, h=0.1')
     plt.xlabel('x')
     plt.ylabel('Output')
     plt.legend()
```

```
[9]: <matplotlib.legend.Legend at 0x7f8c8ae362d0>
```



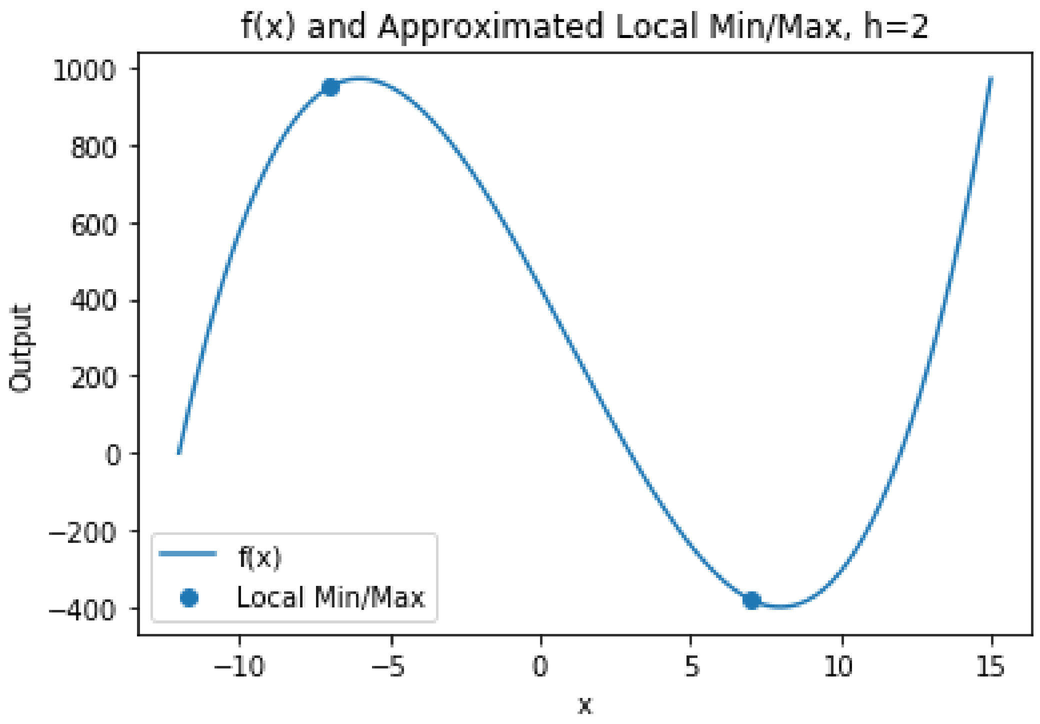
```
[10]: ind = np.argpartition(np.abs(num_1), 2)
      plt.scatter(x[ind[:2]], y[ind[:2]], label='Local Min/Max')
      plt.plot(x, y, label='f(x)')
      plt.title('f(x) and Approximated Local Min/Max, h=1')
      plt.xlabel('x')
      plt.ylabel('Output')
      plt.legend()
```

```
[10]: <matplotlib.legend.Legend at 0x7f8c8adac890>
```



```
[11]: ind = np.argpartition(np.abs(num_2), 2)
      plt.scatter(x[ind[:2]], y[ind[:2]], label='Local Min/Max')
      plt.plot(x, y, label='f(x)')
      plt.title('f(x) and Approximated Local Min/Max, h=2')
      plt.xlabel('x')
      plt.ylabel('Output')
      plt.legend()
```

```
[11]: <matplotlib.legend.Legend at 0x7f8c8ad28ad0>
```



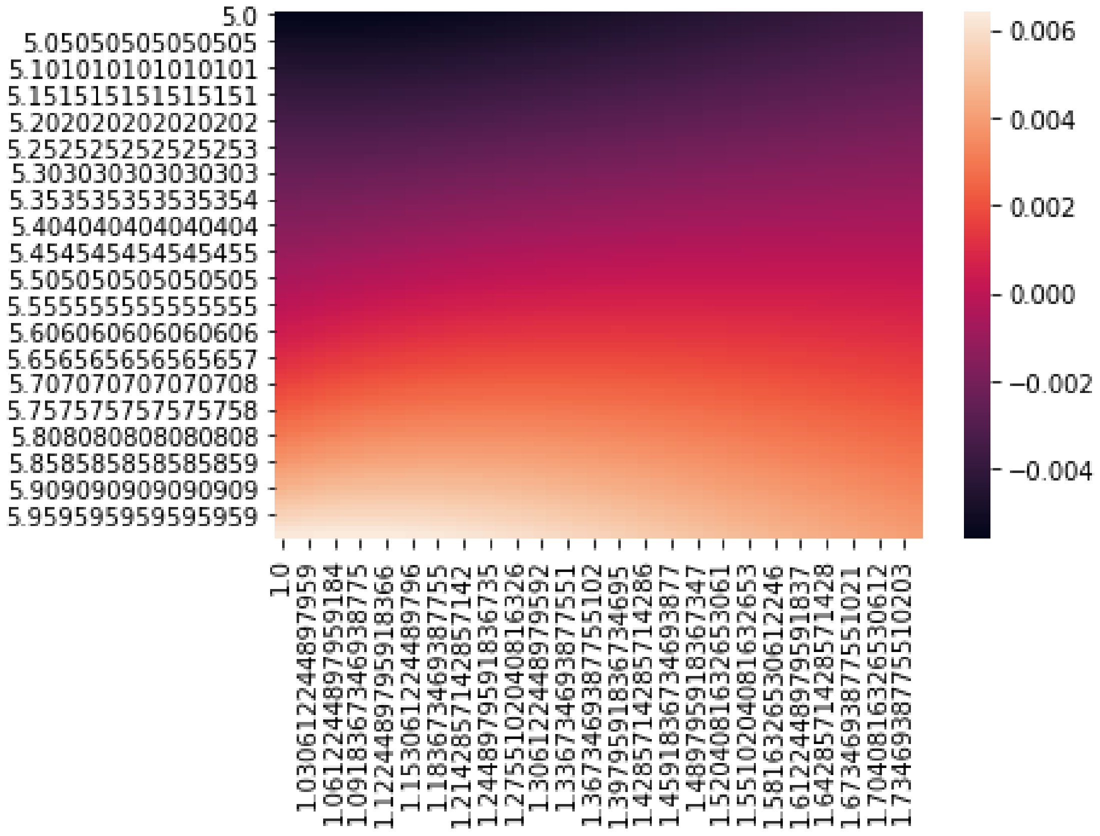
1.5 Experiment 3: Gaussian Model Grid Search - Derivative Space

[12]:
```python
import pandas as pd

from cost_functions import GaussianCostFunction
from test_Numerical_Differentiation import TestOptimizer

gauss = pd.read_csv('gaussdist.csv', header=None)
gauss = gauss.to_numpy()

features = gauss[:, 0]
responses = gauss[:, 1]

gcf = GaussianCostFunction(features, responses)

tester = TestOptimizer()
tester.test_gradient_parabola()
tester.test_gradient_paraboloid()
print("All tests passed")
```

```
All tests passed
```

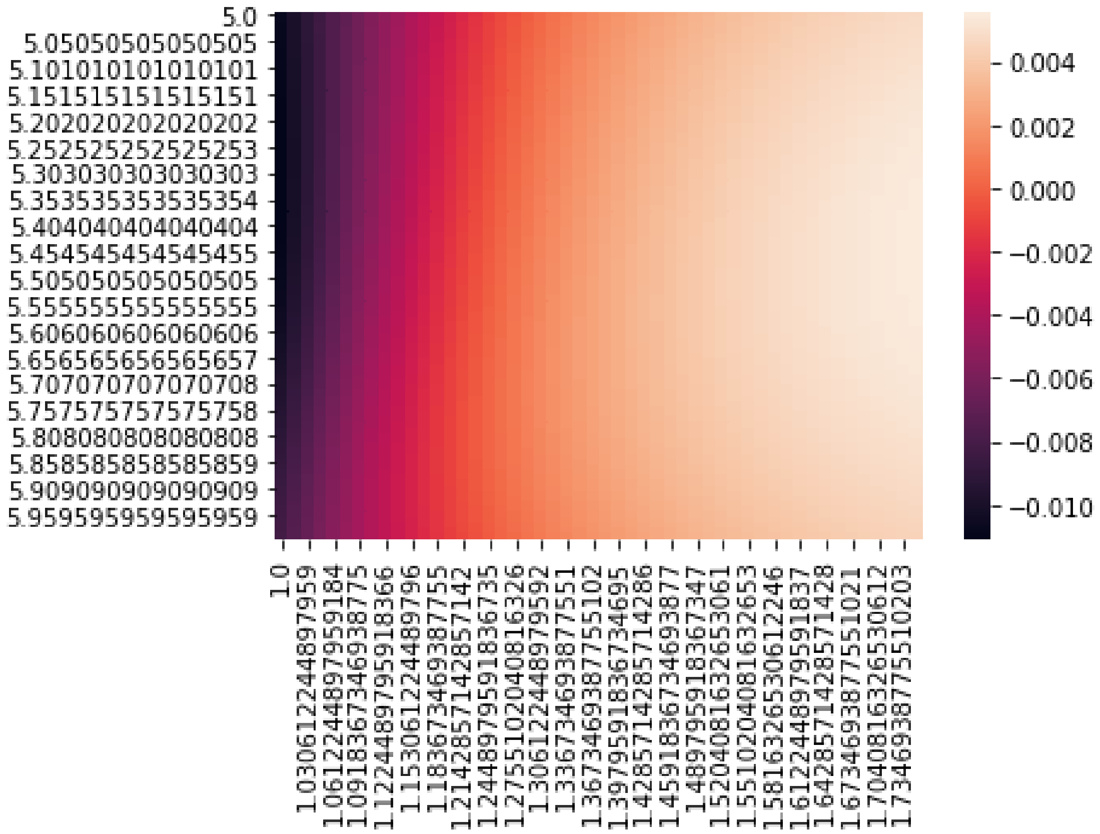
```
[13]: from seaborn import heatmap
      from Numerical_Differentiation import NumericalDifferentiation

      nd = NumericalDifferentiation(1e-5)

      mus = np.linspace(5, 6, num=100)
      sigmas = np.linspace(1, 1.75, num=50)

      grads = np.ndarray((len(mus), len(sigmas), 2))
      for i in range(len(mus)):
          for j in range(len(sigmas)):
              grad = nd.gradient(gcf, np.array([mus[i], sigmas[j]]))
              grads[i][j] = grad

      mu_grads_df = pd.DataFrame(data=grads[:,:,0], index=mus, columns=sigmas)
      heatmap(mu_grads_df)
```

```
[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c86bbad10>
```



```
[14]: sig_grads_df = pd.DataFrame(data=grads[:,:,1], index=mus, columns=sigmas)
      heatmap(sig_grads_df)
```

```
[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c88f42850>
```

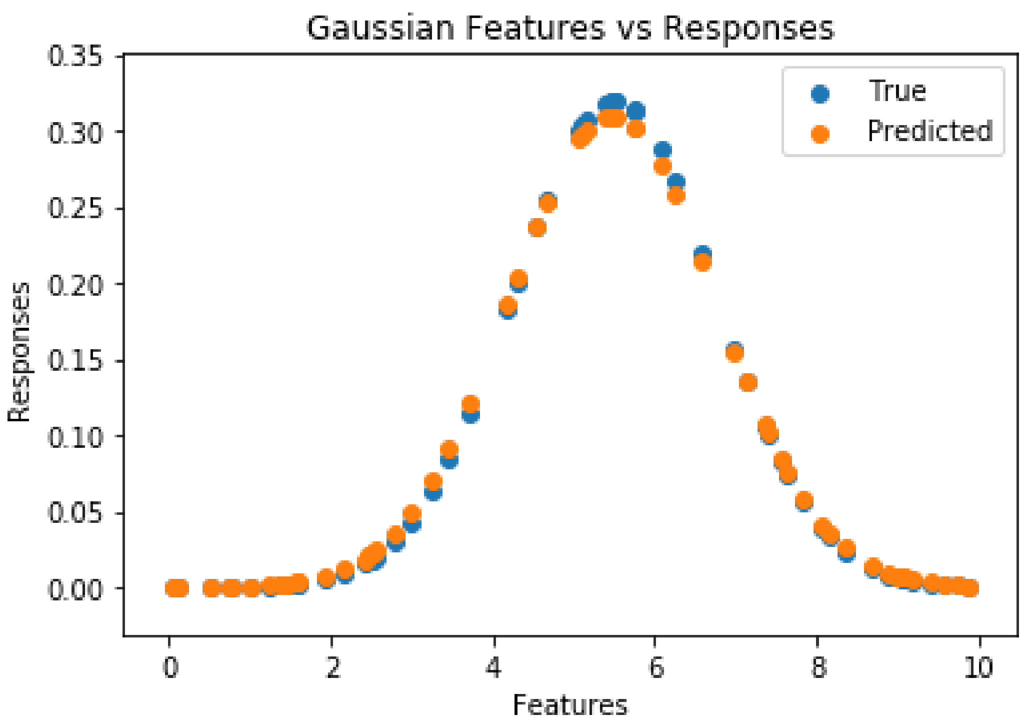


```
[15]: mu_derivs = grads[:,:,0]
      sig_derivs = grads[:,:,1]

      mu_min_ind = np.where(np.abs(mu_derivs) == np.abs(mu_derivs).min())
      mu_min = mu_derivs[mu_min_ind][0]
      mu = mus[mu_min_ind[0][0]]

      sig_min_ind = np.where(np.abs(sig_derivs) == np.abs(sig_derivs).min())
      sig_min = sig_derivs[sig_min_ind][0]
      sig = sigmas[sig_min_ind[1][0]]

      print(mu_min)
      print(sig_min)

      params = np.array([mu, sig])
```

```
print(params)
pred = gcf._predict(features, params)

plt.scatter(features, responses, label='True')
plt.scatter(features, pred, label='Predicted')
plt.title('Gaussian Features vs Responses')
plt.xlabel('Features')
plt.ylabel('Responses')
plt.legend()
```

```
6.764058714178733e-08
-1.3565795622255683e-07
[5.47474747 1.29081633]
```

```
[15]: <matplotlib.legend.Legend at 0x7f8c86da1050>
```


Gaussian Features vs Responses
Responses
Features
True
Predicted