Optimisation Algorithms Week 2 Assignment

(a) **Derivative**

(i) I first defined a sympy function $y = x^4$. Next, I used the sympy diff method to obtain $\frac{dy}{dx} = 4x^3$. Code can be found in Appendix A.

(ii) En estimate function is teted against the derivative. This function has the formula $estimate = \frac{(f(x+\delta) - f(x)}{\delta}$ where $f(x) = y$. This is equal to the derivative when $\delta \to 0$. Table 1 shows an example of the derivative and the estimate functions at work for $\delta = 0.01$. Values for $x \in 1, 2, 3$ are given.

x	y	$\frac{dy}{dx}$	estimate
1	1	4	4.060401
2	16	32	32.240801
3	81	108	108.541201

Table 1. *The values for the function, the derivative of the function, and an estimated derivative for some values of x*

Figure 1 shows the values of the derivative between -1 and 1 computed using the $\frac{dy}{dx}$ formula as well as using the estimate formula with $\delta = 0.01$. The second half of the figure shows the difference between the two values. The two plots show that the errors are non-linearly proportional to the absolute value of x - the closer x is to 0, the smaller the error.

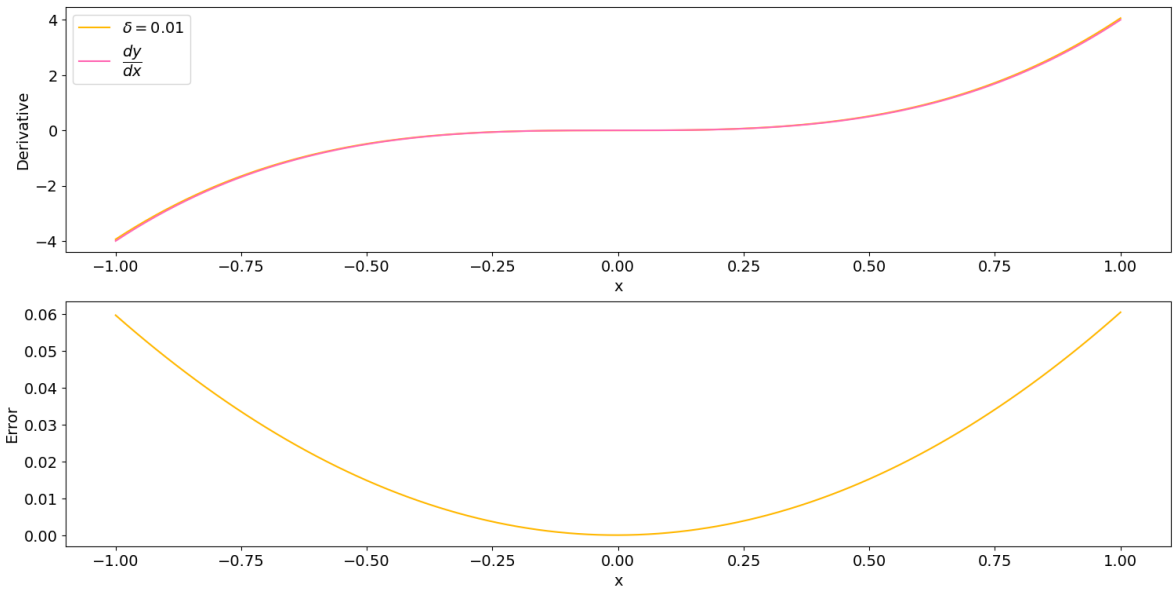

Figure 1: Values for the derivative and errors computing using the formula as well as the estimate

(iii) Figure 2 shows the effect of varying the δ. The second first plot shows the derivative values for a range of x between -1 and 1; the second plot shows the difference between each estimate and the actual value. The figure shows that the difference between the values is proportional to δ. The smaller the δ, the smaller the difference. This is because the derivative can be defined as $f'(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$.

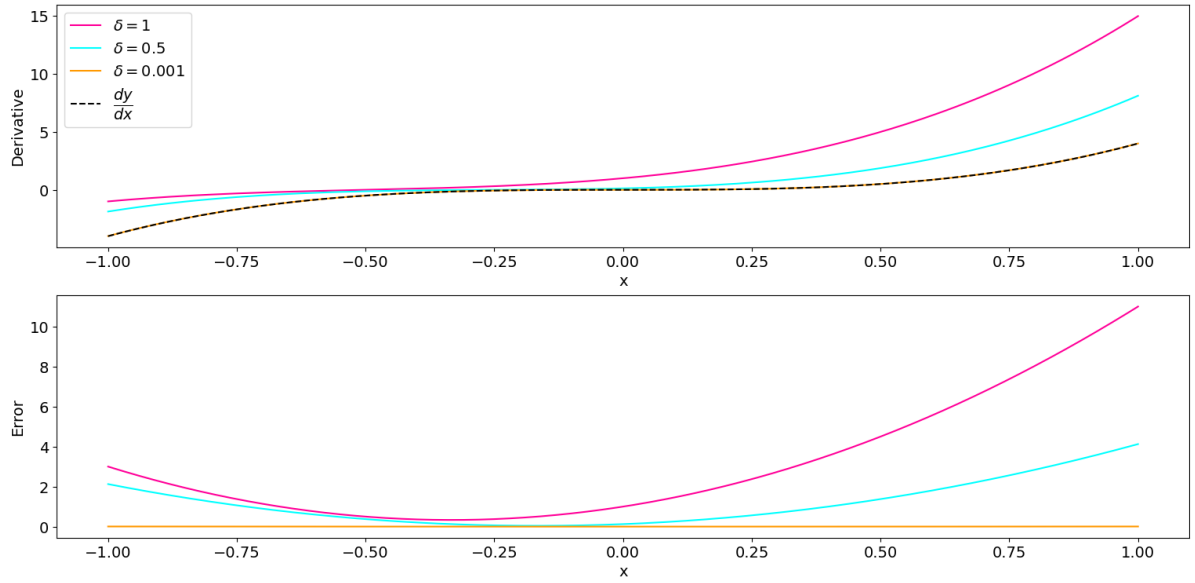

Figure 2: Values for the derivative computing using the formula as well as the estimate

(b) **Gradient Descent**

(i) The code for the gradient descent function is shown in Appendix A. It is based on the code shown in class. The function takes the function f to minimise, the gradient df of f, the starting point x_0, the step size α, and allows for the specification of the number of iterations, with a default set at 50. The current point is initialised at x_0 and is then updated by making a step equal to the opposite of the gradient in that point scaled by the step size. If the function is increasing in the current point, the gradient will be positive, and by taking a step equal to the scaled opposite value, which is negative, the next point will correspond to a smaller x and a smaller f(x). If the function is decreasing in the current point, the opposite happens, but the next point still corresponds to a smaller f(x). By doing this process a number of times (here capped at 50), the hope is to reach the minimum value of the function. In order to plot the process, the values of x at each point are stored in the list X which is returned at the end. The last value in the list is the value corresponding to the smallest function value found.

(ii) Figure 3 shows the gradient descent for the function $f(x) = x^4$ when the starting point x_0 is 1 and the step size α is 0.01. The first plot shows the function together with the steps taken towards minimising it; the second plot shows the difference between the current function value and the minimum at each step. The figure shows that with each step, the step size becomes smaller and the decrease in the error becomes smaller. The rate of convergence is slower because of the gradient becoming smaller towards zero.

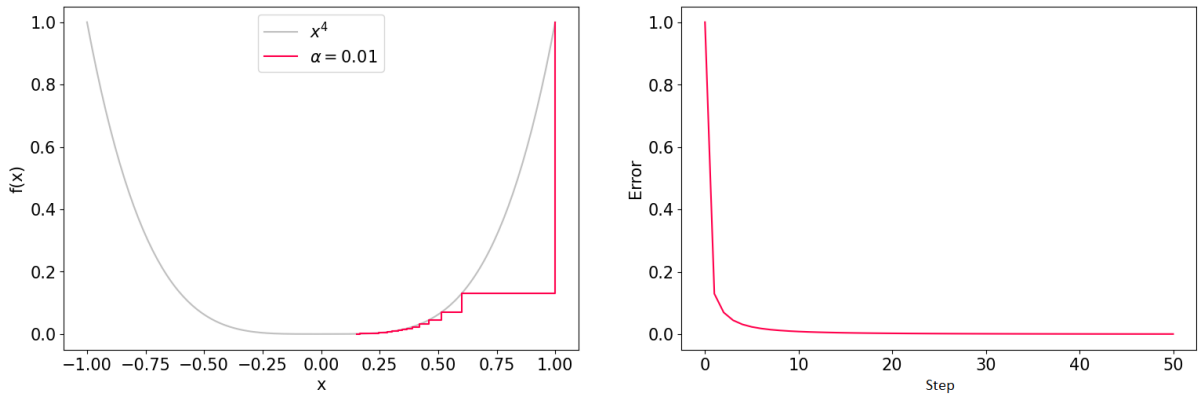

Figure 3: Values for the derivative computing using the formula as well as the estimate

(iii) Figure 4 shows the same process from (ii) for three different x_0 values, -1.15, 1 and -0.8, each

with 5 different alpha values. The plots on the left have a fixed x range between -1 and 1, while the plots on the right have a fixed y axis between 0 and 2. This helps to show how, for a lower starting point values, the rate of convergence is always slow, while for larger values, the rate starts high and slows down with each step. Moreover, the alphas are chosen to show a range of outcomes. For an alpha that is too large, the steps end up being too large and jump from one side of the curve to the other, slowly converging towards the middle. The largest alpha chosen for each x_0 never converges and instead just leads to jumping back and forth between the same points. Conversely, the small alpha values lead to very small steps being take. Due to the maximum number of steps allowed being capped at 50, the smaller the alpha, the further away from reaching the minimum is the result. Finding a balance between these effects leads to the best results.

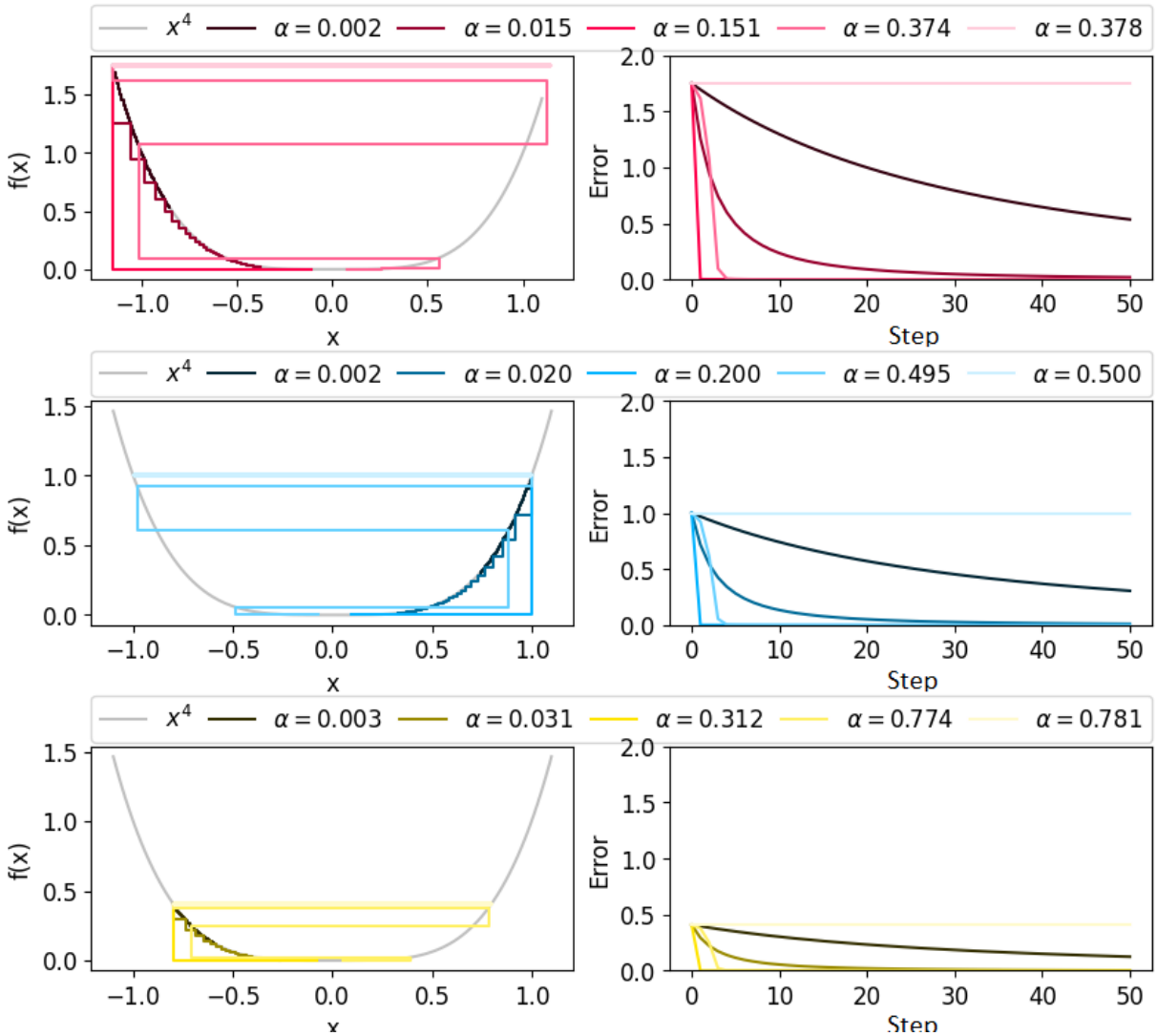

Figure 4: Values for the derivative computing using the formula as well as the estimate

Figure 5 shows the same thing as figure 4 with one exception - the axes are scaled for the left hand plots to the x range $[-x_0, x_0]$, and for the right hand plots to the upper y limit corresponding with the starting error value. Moreover, in both figures the alpha values depend on the starting point x_0. Specifically, each step size is set as $alpha = c * \frac{x_0^2}{2}$ where c is some fraction smaller than 1. This was done to ensure that for each starting value, the maximum alpha doesn't cause an error. However, since the plots end up looking identical, it also shows how regardless of the choice of x_0, the rate of convergence and success of the minimisation problem depends on the fraction c chosen.

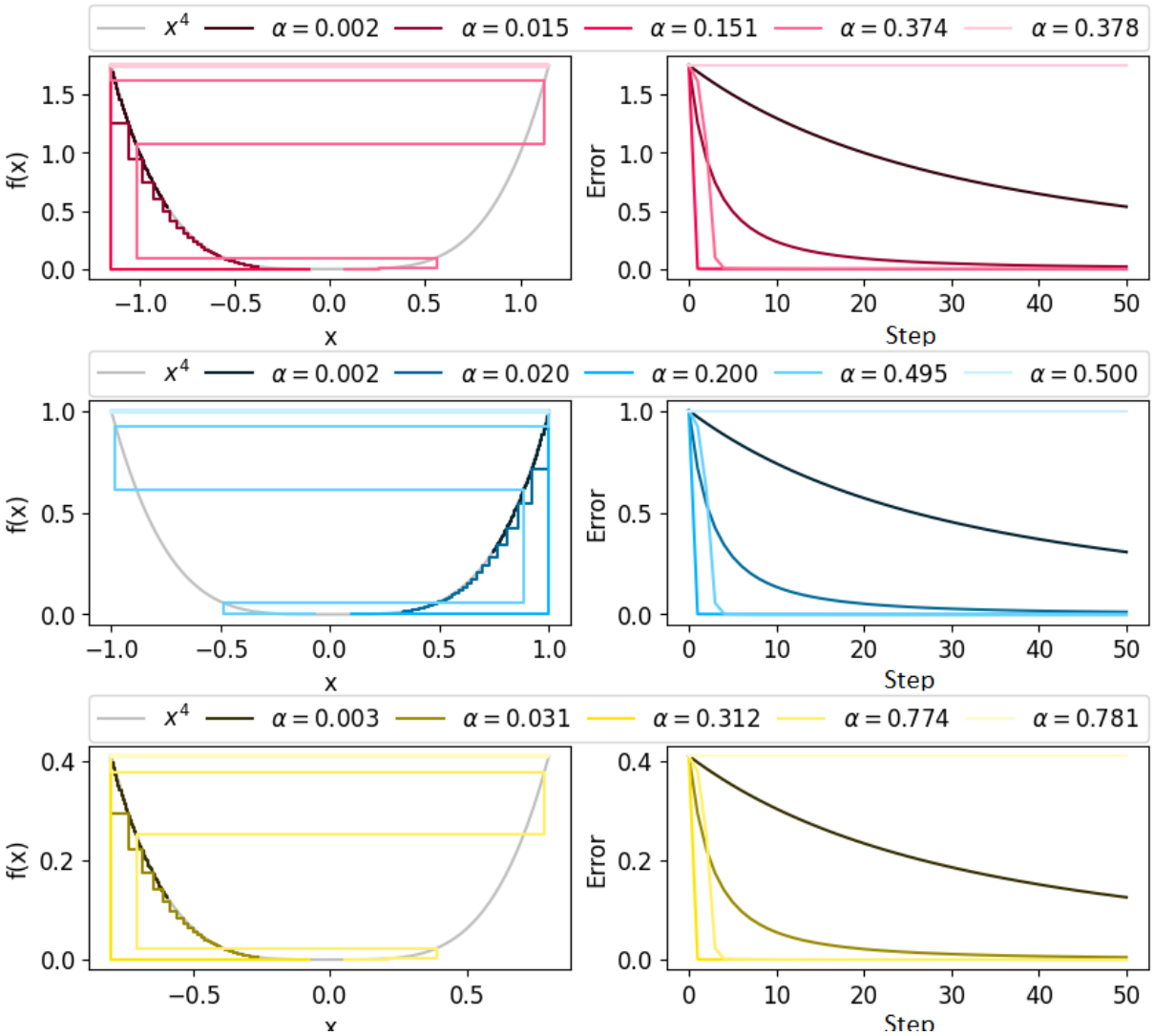

Figure 5: Values for the derivative computing using the formula as well as the estimate

(c) **Function Scaling**

(i) Figure 6 shows the gradient descent process on the function $f(x) = \gamma x^2$ for different values of gamma. The starting point in each case is $x_0 = 2$ and the alpha is fixed as 0.01. It can be seen that for a larger gamma values, the step size is larger. This is because the derivative of the function scaled by gamma is also scaled by gamma. Thus, the larger the gamma, the larger the gradient, and the larger the step size and the faster the convergence rate. For a number of iterations capped at 50, this means that the larger the gamma, the closer the result gets to the minimum.

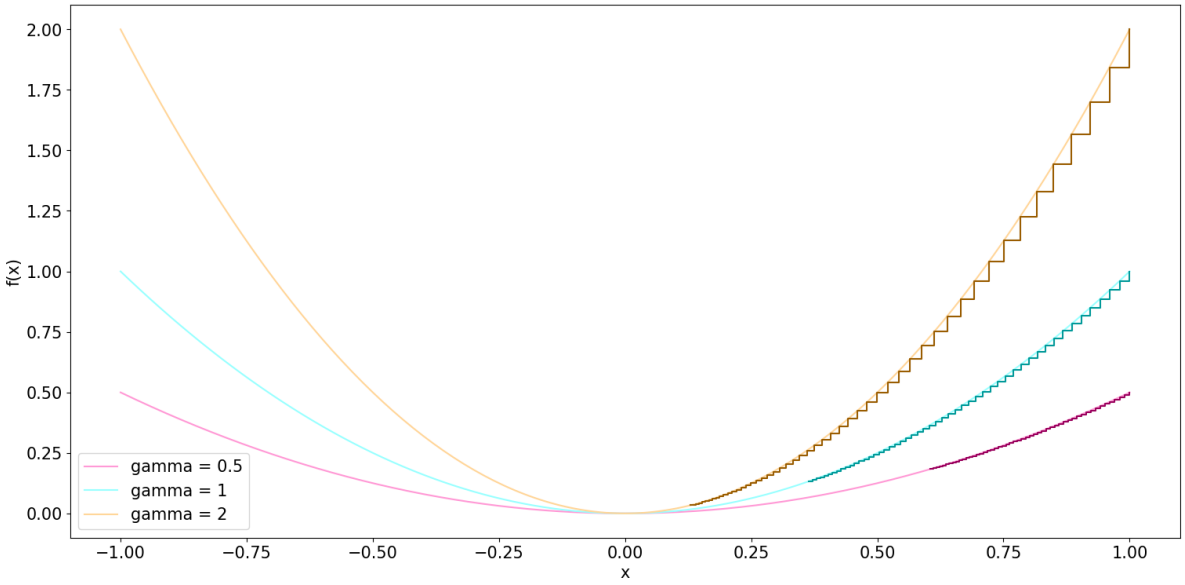

Figure 6: Values for the derivative computing using the formula as well as the estimate

(ii) Figure 7 shows the gradient descent process for $f(x) = \gamma|x|$, $x_0 = 1$, and $\alpha = 0.1$. This choice of alpha results in oscillations as the descent approaches the minimum. This is because the function is non-differentiable in 0. Moreover, these oscillations are themselves proportional to gamma since gamma partly determines the step size.

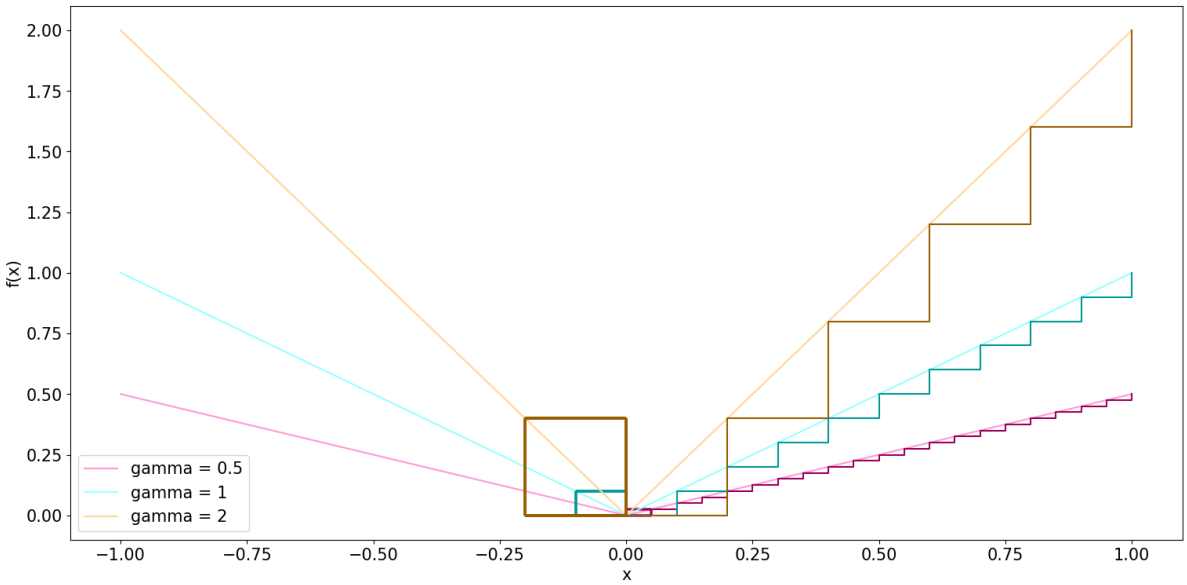

Figure 7: Values for the derivative computing using the formula as well as the estimate

Appendix A - Code

```
import sympy
from sympy import print_latex
import numpy as np
import matplotlib.pyplot as plt
import colorsys

# Obtaining derivative for y = x^4
x = sympy.symbols('x', real=True)
y = x ** 4
dydx = sympy.diff(y, x)
print(y, dydx)

# Calculating and printing some values
y = sympy.lambdify(x, y)
dydx = sympy.lambdify(x, dydx)
est = ((x + 0.01) ** 4 - x ** 4) / 0.01
est = sympy.lambdify(x, est)
i = np.array([1,2,3])
print(y(i),dydx(i), est(i))

# Plotting estimates
deltas = [1,0.5,0.001] #[0.01]
plt.rcParams['font.size'] = 14
fig,ax=plt.subplots(2)
xx = np.arange(-1, 1.01, 0.01)
for i,delta in enumerate(deltas):
    est = ((x + delta) ** 4 - x ** 4) / delta
    est = sympy.lambdify(x, est)
    colour = colorsys.hls_to_rgb(*(0.9 - i * 0.4,0.5,1))
    ax[0].plot(xx, est(xx), color=colour,label = (r'$\delta = $' +str(delta)))
    ax[1].plot(xx, est(xx)-dydx(xx),color = colour)
ax[0].plot(xx, dydx(xx), color='black',linestyle = '--',label = r'$\dfrac{dy}{dx}$')
ax[0].set_xlabel('x')
ax[0].set_ylabel('Derivative')
ax[1].set_xlabel('x')
ax[1].set_ylabel('Error')
ax[0].legend()
plt.show()

# Gradient descent
def gradDescent(f, df, x0, alpha, iters=50):
    x = x0
    X = [x]
    for k in range(iters):
        step = alpha * df(x)
        x = x - step
        X.append(x)
    return X

#Plotting gradient descent
plt.rcParams['font.size'] = 12
fig,ax = plt.subplots(3,2)
for i, x in enumerate([-1.15,1,-0.8]):
    xx = np.arange(-abs(x), abs(x)+0.01, 0.01)
    ax[i,0].plot(xx, y(xx), color='silver',label=(r'$x^4$'))
    for j, c in enumerate([250,25,2.5,1.01,1]):
```

```
        alpha = 0.5/x**2/c
        X = gradDescent(y, dydx, x, alpha)
        colour=colorsys.hls_to_rgb(*(0.95-i*0.4,(0.1+0.2*j) ,1))
        ax[i,0].step(X,y(np.array(X)),color=colour,label=(r'$\alpha = %.3f$'%alpha))
        ax[i,0].set_xlabel('x')
        ax[i,0].set_ylabel('f(x)')
        ax[i,1].plot(y(np.array(X)), label=(r'$\alpha = %.2f$'%alpha), color=colour)
        ax[i,1].set_xlabel('x')
        ax[i,1].set_ylabel('Error')
        ax[i,0].legend(bbox_to_anchor=(0., 1.02, 2.2, .102), loc=3,ncol=6, mode="expand",
        borderaxespad=0)
plt.show()

#Plotting gradient descent for scaled functions
plt.rcParams['font.size'] = 15
x = sympy.symbols('x', real=True)
for i, gamma in enumerate([0.5,1,2]):
    y = gamma*abs(x) #x**2
    dydx = sympy.lambdify(x,sympy.diff(y,x))
    y = sympy.lambdify(x, y)
    xx = np.arange(-1, 1.01, 0.01)
    colour = colorsys.hls_to_rgb(*(0.9 - i * 0.4, 0.8, 1))
    plt.plot(xx, y(xx), color=colour,label = ('gamma = '+str(gamma)))
    X = gradDescent(y, dydx, 1, 0.1)
    colourstep = colorsys.hls_to_rgb(*(0.9 - i * 0.4, 0.3, 1))
    plt.step(X, y(np.array(X)),color=colourstep)
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend()
plt.show()
```