Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Week 2 Assignment
CS7CS2 Optimisation for Machine Learning

Ujjayant Kadian (22330954)

February 13, 2025

Introduction

This report documents a series of experiments and analyses related to:

1. Symbolic and Numerical Differentiation of the function $y(x) = x^4$.
2. Finite Difference Approximations and the effect of the step δ.
3. Gradient Descent (GD) applied to $y(x) = x^4$, $y(x) = \gamma x^2$ and $y(x) = \gamma|x|$.

(a) Analyzing $y(x) = x^4$

(a)(i) Symbolic Derivative

We begin with the function:

$$y(x) = x^4$$

Its exact derivative is:

$$\frac{dy}{dx} = 4x^3$$

This can be confirmed using the symbolic algebra package in python called **sympy**.

```
x = sympy.Symbol('x')
y = x**4
dy_dx = sympy.diff(y, x)  # yields 4*x^3
```

(a)(ii) Finite-Difference Derivative and Comparison

We compare the exact derivative $4x^3$ with **finite difference** approximation, calculated by:

$$D_\delta[f](x) = \frac{f(x+\delta) - f(x)}{\delta}$$

Using $\delta = 0.01$, we compute both values on an array of x-points $[-2, 2]$. The following code outlines the process:

```
def f(x):
    return x**4
def finite_diff(f, x, delta=0.01):
    return (f(x + delta) - f(x)) / delta

x_vals = np.linspace(-2, 2, 50)

exact_derivs = 4 * x_vals**3
delta = 0.01
fd_derivs = [finite_diff(f, xv, delta) for xv in x_vals]
```

We plot the exact derivative and finite-difference derivative and obtain the following plot:

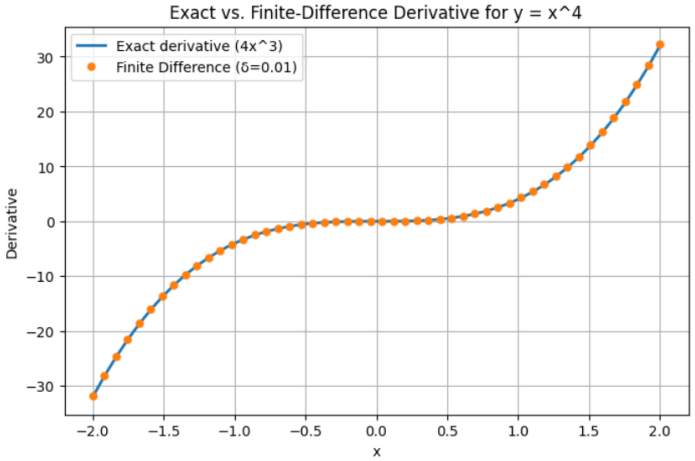

Figure 1: Exact vs. Finite-Difference for $y = x^4$

In this case, the finite-difference estimates (orange dots) lie very close (in fact, overlapping) to $4x^3$. However, when the value of δ is increased, offsets start to emerge and orange dots start to shift away from the exact derivatives.

This can be explained by calculating the error between exact derivative and finite-difference derivative:

$$Error = \frac{f(x+\delta) - f(x)}{\delta} - \frac{df(x)}{dx}$$

$$Error = \frac{(x+\delta)^4 - x^4}{\delta} - 4x^3$$

$$Error = \frac{x^4 + 4x^3\delta + 6x^2\delta^2 + 4x\delta^3 + \delta^4 - x^4}{\delta} - 4x^3$$

$$Error = 6x^2\delta + 4x\delta^2 + \delta^3$$

Based on this, we can deduce that error term grows with $|x|$ and δ (in the order of $x^2\delta$).

(a)(iii) Varying δ

To better illustrate how the finite difference error depends on δ, we can fix $x = 1$ and evaluate the error. Based on the earlier derived expression, here:

$$Error = 6\delta + 4\delta^2 + \delta^3$$

We can calculate the error over $\delta \in [10^(-3), 1]$ on a logarithmic scale:

```
deltas = np.logspace(-3, 0, 50)  # 0.001 to 1
x_test = 1.0
errors = []

for d in deltas:
    fd_approx = finite_diff(f, x_test, d)
    exact = 4 * x_test**3  # 4(1)^3 = 4
    errors.append(abs(fd_approx - exact))
```

We can obtain the following graph:

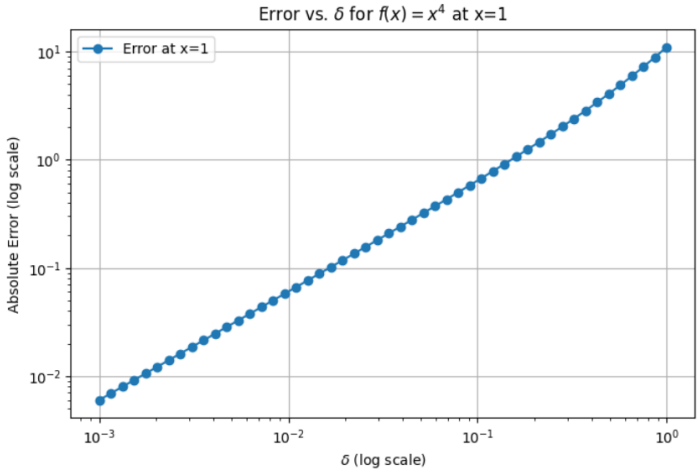

Figure 2: Error vs. δ for $f(x) = x^4$ at x=1

- The error grows monotonically from $\delta = 10^{-3}$ to $\delta = 1$. This is expected because in the range, 6δ dominates and it simply gets larger with δ.
- If δ were much smaller (e.g 10^{-10}), floating-point round-off eventually causes the error to rise again, forming a shape similar to "U".

(b) Gradient Descent on $y(x) = x^4$

(b)(i) Fixed Step-Size Algorithm

Recalling $y(x) = x^4$. Its gradient or derivative is $4x^3$. The gradient descent update (with step size α) is:

$$x_{\text{new}} = x_{\text{old}} - \alpha.(4x_{\text{old}}^3)$$

The following code shows the gradient descent update:

```
def grad_y(x):
    return 4 * x**3

def gradient_descent(initial_x, alpha, n_steps=20):
    x_current = initial_x
    xs = [x_current]
    for _ in range(n_steps):
        g = grad_y(x_current)
        x_current = x_current - alpha * g
        xs.append(x_current)
    return xs
```

We are storing each iteration's x-value in **xs**.

(b)(ii) Example: $x_0 = 1, \alpha = 0.1$

Below shows how gradient descent can be calculated for 20 iterations:

```
x_vals_gd = gradient_descent(initial_x=1.0, alpha=0.1, n_steps=20)
y_vals_gd = [xv**4 for xv in x_vals_gd]
```

Using this, we can obtain the following graphs showing how x and $y(x)$ vary with every gradient descent iteration:

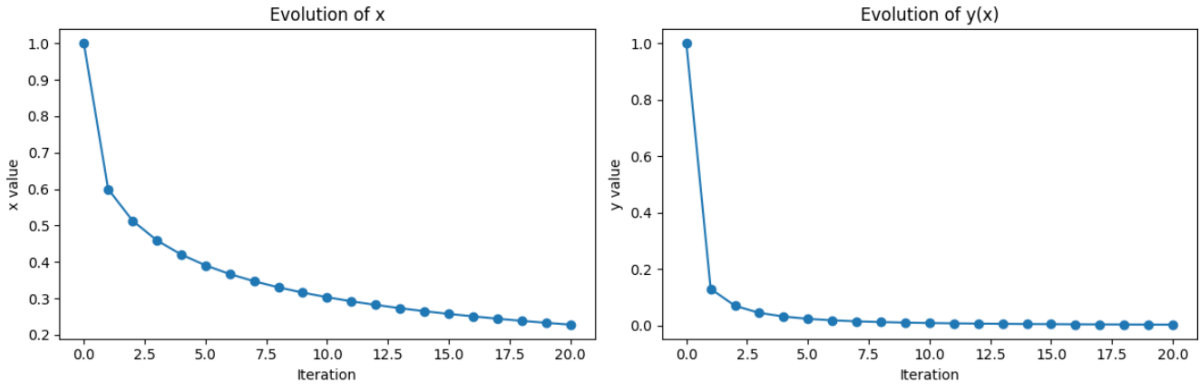

Figure 3: Evolution of x and $y(x)$

- x quickly descends from 1.0 to 0 (almost).
- Similarly, $y = x^4$ starts at 1.0 and drops near 0 (almost suddenly) and gets plateaued earlier when compared to x. This indicates that with current selection of concerned values, we are successful in minimizing x^4.

(b)(iii) Sensitivity to initial x and α

We now vary x_0 (e.g., $-2.0, 1.0, -0.5, 1.0, 1.5$) and α (e.g., $0.01, 0.1, 0.2, 0.5$) to see the convergence behavior of this GD algorithm. The following code shows how we can iterate over these values and calculate gradient descent:

```
initial_values = [2.0, -1.0, -0.5, 1.0, 1.5]
alphas = [0.01, 0.1, 0.2, 0.5]

results = []
for x0 in initial_values:
    for alpha in alphas:
        xs_gd = gradient_descent(x0, alpha, n_steps=30)
        final_x = xs_gd[-1]
        results.append({
            "Initial x0": x0,
            "Alpha": alpha,
            "Final x": final_x,
            "Iterations": len(xs_gd)
        })
```

We obtain the following values:

	Initial x0	Alpha	Final x	Iterations
0	2.0	0.01	5.971686e-01	31
1	2.0	0.10	-1.907218e-01	31
2	2.0	0.20	-2.071761e+05	4
3	2.0	0.50	5.474000e+03	3
4	-1.0	0.01	-5.363442e-01	31
5	-1.0	0.10	-1.909326e-01	31
6	-1.0	0.20	-1.172800e-01	31
7	-1.0	0.50	-1.000000e+00	31
8	-0.5	0.01	-3.944026e-01	31
9	-0.5	0.10	-1.849222e-01	31
10	-0.5	0.20	-1.343476e-01	31
11	-0.5	0.50	-8.487700e-02	31
12	1.0	0.01	5.363442e-01	31
13	1.0	0.10	1.909326e-01	31
14	1.0	0.20	1.172800e-01	31
15	1.0	0.50	1.000000e+00	31
16	1.5	0.01	5.809849e-01	31
17	1.5	0.10	1.213568e-01	31
18	1.5	0.20	1.147177e-01	31
19	1.5	0.50	-4.588798e+07	4

Figure 4: Table showing GD with varying x and α

- The initial value x_0 and α combination has a significant influence on convergence or divergence as seen in the table. With poor selection of x_0 and α, it can very easily diverge. For example, when $x_0 = 2.0$ and $\alpha = 0.20$, it diverges very quickly resulting in a very high $|x|$ value.
- For small α, the method generally converges but slowly.
- For moderate (or 'optimal') α, it converges faster. For larger values, the next iteration can overshoot and diverge/oscillate.
- Some runs produce overflow because of x-value getting too large (because x starts to diverge) and thus a tolerance value (see appendix) must be used to detect divergence.

(c) Changing the function

(c) (i) $y(x) = \gamma x^2$

Here, γ is a parameter. Thus,

$$\frac{dy}{dx} = 2x\gamma \text{ and } x \leftarrow x - \alpha(2x\gamma) = (1 - 2\alpha\gamma)x$$

Using this expression, the gradient descent algorithm for this function can be defined as below:

```
def gradient_descent_gamma_x2(initial_x, alpha, gamma, n_steps=20):
    x_current = initial_x
    xs = [x_current]
    for _ in range(n_steps):
        grad = 2 * gamma * x_current
        x_current = x_current - alpha * grad
        xs.append(x_current)
    return xs
```

We can deduce that:

- If $|1 - 2\alpha\gamma| < 1$, it converges to $x = 0$.
- If the magnitude of the factor exceeds 1, GD diverges.

We can compare $\gamma = 1$ vs. $\gamma = 8$ with $\alpha = 0.1$ and $x_0 = 1$. For small $\gamma.\alpha$, we get slow and stable convergence to 0 as shown below. For larger γ, we can see oscillations.

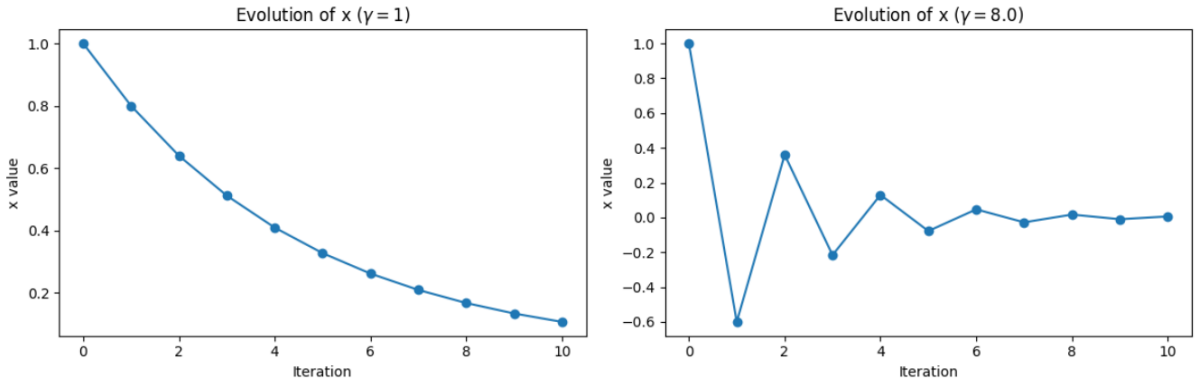

Figure 5: Evolution of x with different γ

(c)(ii) $y(x) = \gamma|x|$

Here, as we know, the derivative is piece-wise:

- For $x > 0$, $\frac{d}{dx}(\gamma|x|) = \gamma$.
- For $x < 0$, $\frac{d}{dx}(\gamma|x|) = -\gamma$.
- At $x = 0$, the function is not differentiable but we can pick any sub-gradient $\in [-\gamma, \gamma]$.

Using this expression, the gradient descent algorithm for this function can be defined as below:

```
def gradient_descent_gamma_abs(initial_x, alpha, gamma, n_steps=20):
    x_current = initial_x
    xs = [x_current]
    for _ in range(n_steps):
        if x_current > 0:
            grad = gamma
        elif x_current < 0:
            grad = -gamma
        else:
            grad = 0.0  # e.g. pick subgradient=0
        x_current = x_current - alpha * grad
        xs.append(x_current)
    return xs
```

We can deduce that:

- If $\alpha\gamma$ is "small", GD should monotonically decreases towards 0.
- If $\alpha\gamma$ is large, GD can overshoot and keep flipping the sign of x (oscillate).

We can compare $\gamma = 1$ vs. $\gamma = 2$ with $\alpha = 0.1$ and $x_0 = 1$ to prove the above points. As shown below, when $\gamma = 2$, we observe oscillations and when $\gamma = 1$, it monotonically moves toward 0.

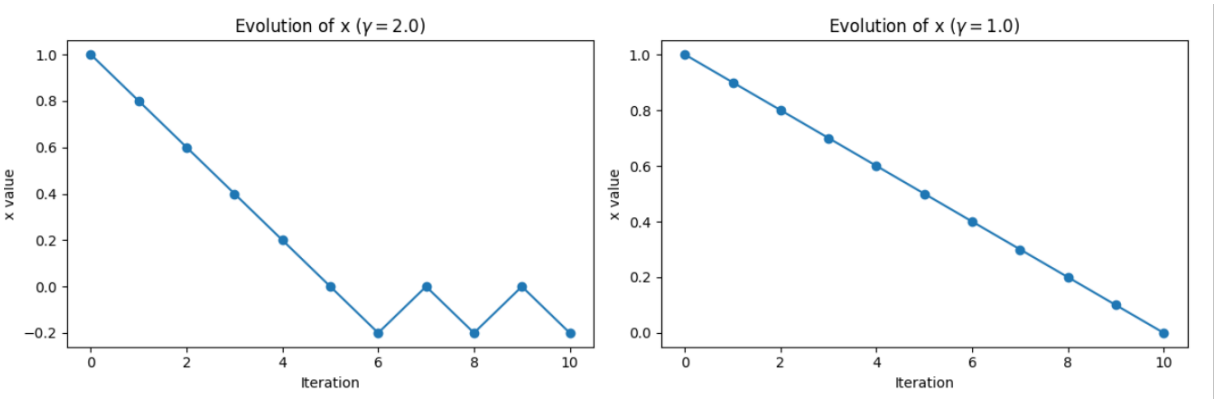

Figure 6: Evolution of x with different γ

Conclusion

Symbolic vs. Numerical Derivatives

For x^4, the exact derivative is $4x^3$. A forward-difference approximation introduces an error of order δ.
If δ is too large, the approximation is coarse; if δ is too small, floating-point errors may emerge.

Gradient Descent on x^4

The update rule is:

$$x \leftarrow x - \alpha(4x^3)$$

Convergence depends critically on both α and x_0.
- A modest α converges quickly.
- A large α may cause divergence.

Scaling by γ

For $y(x) = \gamma x^2$: The update rule is:

$$x \leftarrow (1 - 2\alpha\gamma)x$$

Convergence occurs only if:

$$|1 - 2\alpha\gamma| < 1$$

For $y(x) = \gamma|x|$: The gradient is piecewise:

$$\frac{d}{dx}(\gamma|x|) = \pm\gamma$$

- Convergence behavior depends on $\alpha\gamma$.
- Larger $\alpha\gamma$ may cause oscillations.
- Moderate $\alpha\gamma$ ensures quick setting towards zero.

Summary: These experiments highlight how finite-difference methods and gradient-based optimization are sensitive to step sizes, scaling factors, and floating-point constraints, underscoring the need for careful choice of parameters in numerical methods.

A Code

```
import sympy

x = sympy.Symbol('x', real=True)

y = x**4

dy_dx = sympy.diff(y, x)

print(f"y(x) = {y}")
print(f"dy/dx = {dy_dx}")


import numpy as np
import matplotlib.pyplot as plt

# the function
def f(x):
    return x**4

# Finite-difference approximation
def finite_diff(f, x, delta=0.01):
    return (f(x + delta) - f(x)) / delta

x_vals = np.linspace(-2, 2, 50)

# Exact derivative = 4x^3
exact_derivs = 4 * x_vals**3

# Finite-difference derivative
delta = 0.01
fd_derivs = [finite_diff(f, xv, delta) for xv in x_vals]

# Plot
plt.figure(figsize=(8,5))
plt.plot(x_vals, exact_derivs, label='Exact derivative (4x^3)', linewidth=2)
plt.plot(x_vals, fd_derivs, 'o', label=r'Finite Difference ($\delta$=0.01)',
    markersize=5)
plt.xlabel('x')
plt.ylabel('Derivative')
plt.title('Exact vs. Finite-Difference Derivative for y = x^4')
plt.legend()
plt.grid(True)
plt.show()


deltas = np.logspace(-3, 0, 50)  # \delta from 0.001 to 1.0
x_test = 1.0

errors = []
for d in deltas:
    fd_approx = finite_diff(f, x_test, d)
    exact = 4 * x_test**3
    errors.append(abs(fd_approx - exact))

plt.figure(figsize=(8,5))
plt.plot(deltas, errors, 'o-', label='Error at x=1')
plt.xscale('log')
```

```
plt.yscale('log')
plt.xlabel(r'$\delta$ (log scale)')
plt.ylabel('Absolute Error (log scale)')
plt.title(r'Error vs. $\delta$ for $f(x) = x^4$ at x=1')
plt.grid(True)
plt.legend()
plt.show()


def grad_y(x):
    return 4 * x**3

def gradient_descent(initial_x, alpha, n_steps=30, tolerance=1e10):
    x_current = initial_x
    xs = [x_current]
    for _ in range(n_steps):
        g = grad_y(x_current)
        x_current = x_current - alpha * g
        if abs(x_current) > tolerance:
            # Stop if values become too large
            break
        xs.append(x_current)
    return xs


# Perform gradient descent
x_vals_gd = gradient_descent(initial_x=1.0, alpha=0.1, n_steps=20)

# Compute y = x^4 at each iteration
y_vals_gd = [xv**4 for xv in x_vals_gd]

# Plot x and y over iterations
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(x_vals_gd, 'o-', label='x')
plt.xlabel('Iteration')
plt.ylabel('x value')
plt.title('Evolution of x')

plt.subplot(1,2,2)
plt.plot(y_vals_gd, 'o-', label='y(x) = x^4')
plt.xlabel('Iteration')
plt.ylabel('y value')
plt.title('Evolution of y(x)')

plt.tight_layout()
plt.show()

initial_values = [2.0, -1.0, -0.5, 1.0, 1.5]
alphas = [0.01, 0.1, 0.2, 0.5]

results = []

for x0 in initial_values:
    for alpha in alphas:
        xs_gd = gradient_descent(x0, alpha, n_steps=30)
        final_x = xs_gd[-1]
        results.append({
            "Initial x0": x0,
            "Alpha": alpha,
            "Final x": final_x,
            "Iterations": len(xs_gd)
        })

import pandas as pd

results_df = pd.DataFrame(results)
print(results_df)

def gradient_descent_gamma_x2(initial_x, alpha, gamma, n_steps=20):
    x_current = initial_x
    xs = [x_current]
    for _ in range(n_steps):
        grad = 2 * gamma * x_current
        x_current = x_current - alpha * grad
```

```
        xs.append(x_current)
    return xs

# Example with gamma=1, alpha=0.1
test_xs = gradient_descent_gamma_x2(1.0, alpha=0.1, gamma=1.0, n_steps=10)
test_xs_2 = gradient_descent_gamma_x2(1.0, alpha=0.1, gamma=8.0, n_steps=10)

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(test_xs, 'o-', label='x')
plt.xlabel('Iteration')
plt.ylabel('x value')
plt.title(r'Evolution of x ($\gamma=1.0$)')

plt.subplot(1,2,2)
plt.plot(test_xs_2, 'o-', label='x')
plt.xlabel('Iteration')
plt.ylabel('x value')
plt.title(r'Evolution of x ($\gamma=8.0$)')

plt.tight_layout()
plt.show()

def gradient_descent_gamma_abs(initial_x, alpha, gamma, n_steps=20):
    x_current = initial_x
    xs = [x_current]
    for _ in range(n_steps):
        if x_current > 0:
            grad = gamma
        elif x_current < 0:
            grad = -gamma
        else:
            # At x=0, any subgradient in [-gamma, gamma] could be used.
            # Here, we pick 0 for simplicity (makes x stay at 0 if reached)
            grad = 0.0

        x_current = x_current - alpha * grad
        xs.append(x_current)
    return xs

test_abs = gradient_descent_gamma_abs(1.0, alpha=0.1, gamma=2.0, n_steps=10)
test_abs_2 = gradient_descent_gamma_abs(1.0, alpha=0.1, gamma=1.0, n_steps=10)

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(test_abs, 'o-', label='x')
plt.xlabel('Iteration')
plt.ylabel('x value')
plt.title(r'Evolution of x ($\gamma=2.0$)')

plt.subplot(1,2,2)
plt.plot(test_abs_2, 'o-', label='x')
plt.xlabel('Iteration')
plt.ylabel('x value')
plt.title(r'Evolution of x ($\gamma=1.0$)')

plt.tight_layout()
plt.show()
```