Numerical Differentiation

Shubha Swarnim Singh

October 7th, 2024

Dr. Brandy Wiegers

CSC – 455 Numerical Computation

Introduction

Numerical differentiation is an essential tool in computational mathematics, providing a practical method for estimating derivatives when the analytical form is either unavailable or difficult to compute. In this project, we focus on the numerical differentiation of the exponential function $g(x) = e^x$ using various finite difference formulas. Specifically, we compare the accuracy of the central difference, three-point, and five-point formulas in approximating the derivative at different points within a given interval. These methods are analyzed using Taylor's theorem to predict their error bounds, which helps us determine the expected accuracy of each approach. A Python program was developed to implement these numerical differentiation techniques, allowing for comparison of results across different approximation intervals. By evaluating the performance of each formula for different numbers of points, we aim to determine the most accurate method for numerical differentiation and explore the trade-offs between computational complexity and accuracy. The results provide insights into the strengths and weaknesses of each method and demonstrate the importance of using higher-order techniques for precise numerical estimation.

Analysis

In this project, we are provided with three different formulae for numerical differentiation. Our objective is to determine which of these formulae yields the most accurate result and provides a reliable approximation for f'(x) at a specific point (x) when h is small. The three formulae for these approximations are as follows:

1) Center Difference:

$$f^{(1)}(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

To determine the error, we expand $f(x+h)$ and $f(x-h)$ using a Taylor series around the point x:

$$f(x+h) = f(x) + f'(x)h + \frac{f^2(x)}{2!}h^2 + \frac{f^3(x)}{3!}h^3 + \frac{f^4(x)}{4!}h^4 + O(h^5)$$

$$f(x-h) = f(x) - f'(x)h + \frac{f^2(x)}{2!}h^2 - \frac{f^3(x)}{3!}h^3 + \frac{f^4(x)}{4!}h^4 - O(h^5)$$

Subtracting $f(x-h)$ from $f(x+h)$:

$$f(x+h) - f(x-h) \ = 2f'(x)h + \frac{2f^3(x)}{3!}h^3 + O(h^5)$$

Dividing by 2h:

$$f'(x) = \frac{2f'(x)h + \frac{2f^3(x)}{3!}h^3}{2h} + O(h^2)$$

The central difference formula has an error term of order $O(h^2)$. This means that the error decreases quadratically as h becomes smaller, making it a second-order accurate method.

2) 3-Point Difference

$$f^{(1)}(x) \approx \frac{f(x+3h) - f(x-h)}{4h}$$

To determine the error, we expand $f(x+3h)$ and $f(x-h)$ using a Taylor series around the point x:

$$f(x+h) = f(x) + 3f'(x)h + \frac{9f^2(x)}{2!}h^2 + \frac{27f^3(x)}{3!}h^3 + O(h^4)$$

$$f(x-h) = f(x) - f'(x)h + \frac{f^2(x)}{2!}h^2 - \frac{f^3(x)}{3!}h^3 + O(h^4)$$

Subtracting $f(x-h)$ from $f(x+3h)$:

$$f(x+3h) - f(x-h) \ = 4f'(x)h + \frac{8f^3(x)}{3!}h^3 + O(h^4)$$

Dividing by 4h:

$$f'(x) = \frac{4f'(x)h + \frac{8f^3(x)}{3!}h^3}{4h} + O(h^2)$$

The three-point formula also has an error term of order $O(h^2)$ similar to the central difference formula.

3) 5-Point Difference

$$f^{(1)}(x) \approx \frac{-f(x-2h) + 8f(x-h) - 8f(x+h) \ + \ f(x+2h)}{12h}$$

To determine the error, we expand each term using a Taylor series around the point x:

$$f(x+2h) = f(x) + 2f'(x)h + \frac{4f^2(x)}{2!}h^2 + \frac{8f^3(x)}{3!}h^3 + \frac{16f^4(x)}{4!}h^4 + O(h^5)$$

$$f(x-2h) = f(x) - 2f'(x)h + \frac{4f^2(x)}{2!}h^2 - \frac{8f^3(x)}{3!}h^3 + \frac{16f^4(x)}{4!}h^4 - O(h^5)$$

$$f(x+h) = f(x) + f'(x)h + \frac{f^2(x)}{2!}h^2 + \frac{f^3(x)}{3!}h^3 + \frac{f^4(x)}{4!}h^4 + O(h^5)$$

$$f(x-h) = f(x) - f'(x)h + \frac{f^2(x)}{2!}h^2 - \frac{f^3(x)}{3!}h^3 + \frac{f^4(x)}{4!}h^4 - O(h^5)$$

Substituting into the five-point formula, we get, all terms involving f(x) cancels, $f'(x)$and $f^2(x)$ also cancels because their coefficients add up to 0 and, $f^3(x)$ also cancels leaving:

$$f^{(1)}(x) \approx \frac{h^4}{30} + \ f^5(x) + O(h^4)$$

After expanding and simplifying, we find that the error term is proportional to h^4:

$$f^{(1)}(x) \approx \frac{-f(x-2h) - 8f(x-h) + 8f(x+h) \ - \ f(x+2h)}{12h} + O(h^4)$$

The five-point formula has an error term of order $O(h^4)$, making it a fourth-order accurate method and the most accurate for small h among the three methods discussed.

Analysis: Discussion comparing (1), (2), (3)

When comparing the three numerical differentiation methods—the central difference, the three-point formula, and the five-point formula—it's clear that the key differentiator is the order of accuracy of their respective error terms. The central difference and three-point formulas both have an error order of $O(h^4)$, indicating that their accuracy improves quadratically as h becomes smaller. However, the five-point formula has a significantly higher accuracy, with an error term of order $O(h^4)$, meaning that the error decreases at a much faster rate as h decreases. This makes the five-point formula the most accurate and reliable option among the three, especially for smooth functions like $g(x) = e^x$, where reducing the spacing h between points yields substantial improvements in precision. Furthermore, the five-point formula uses more points around the target value, which allows it to better cancel out higher-order error terms, resulting in a more precise derivative approximation. Thus, for applications where high accuracy is required, and computational resources are not a significant constraint, the five-point formula is the best choice due to its superior error characteristics.

Analysis: Prediction

Based on the analysis above, we can predict the error bounds for each of the numerical differentiation methods when applied to the function $g(x) = e^x$,. Since all derivatives of $g(x) = e^x$, are also e^x, the error bounds can be expressed in terms of h and the values of the higher-order derivatives at a specific point.

1. Error Bound for Central Difference Formula

The central difference formula has an error term of order $O(h^2)$. Using Taylor's theorem, the leading term of the error can be derived from the third derivative:

$$f'(x) = \frac{2f'(x)h + \frac{2f^3(x)}{3!}h^3}{2h} + \frac{f^{(3)}(x)}{6}h^2$$

For $g(x) = e^x$, $f^{(3)}(x) = e^x$. Therefore, the error bound for the central difference formula can be estimated as:

$$\textbf{Error Bound} = \frac{e^x}{6}h^2$$

2. Error Bound for Three-Point Formula

The Three-Point Formula has an error term of order $O(h^2)$. Using Taylor's theorem, the leading term of the error can be derived from the third derivative:

$$f'(x) = \frac{f(x+3h) - f(x-h)}{4h} + \frac{f^{(3)}(x)}{6}h^2$$

For $g(x) = e^x$, $f^{(3)}(x) = e^x$. Therefore, the error bound for the central difference formula can be estimated as:

$$\textbf{Error Bound} = \frac{e^x}{6}h^2$$

3. Error Bound for Five-Point Formula

The Five-Point Formula has an error term of order $O(h^4)$. Using Taylor's theorem, the leading term of the error can be derived from the fifth derivative:

$$f'(x) = \frac{-f(x-2h) + 8f(x-h) - 8f(x+h) \ + \ f(x+2h)}{12h} + \frac{f^{(5)}(x)}{30}h^4$$

For $g(x) = e^x$, $f^{(5)}(x) = e^x$. Therefore, the error bound for the central difference formula can be estimated as:

$$\textbf{Error Bound} = \frac{e^x}{30}h^4$$

Based on these error bounds, we predict that for small values of h, the five-point formula will provide the most accurate approximation of the derivative of $g(x) = e^x$, as its error decreases much faster $O(h^4)$, compared to the central and three-point formulas $O(h^2)$. Therefore, as h becomes smaller, the five-point formula will yield a significantly smaller error compared to the other two methods.

Computer Program

Attached at the end

Computer Program Test

The computed derivatives show significant fluctuations, especially near the endpoints. The derivatives decrease steadily from 2.1 to 2.5, but there's a sudden and unexpected change to negative values at 2.6 and 2.7, which suggests potential errors. This behavior indicates possible issues with the end-point approximation formula or numerical instability. To improve accuracy, a smaller step size or higher-order differentiation methods should be considered. Additionally, testing with a simpler function with a known derivative would help identify and address inaccuracies, ensuring that the computed values more closely match the expected behavior of the function.

Point	Computer Derivative
2.1	3.899344249999985
2.2	2.876875666666667
2.3	2.249704083333332
2.4	1.837755999999998
2.5	1.5903952499999978
2.6	-1.3554963333333314
2.7	-0.3947944166666654

Computer Program Approximations

The computed derivative values in the first table show a consistent trend up to 0.6, closely approximating the analytical values with minimal errors. However, from x = 0.8 onwards, the results become negative, indicating a significant deviation. The error increases noticeably towards the end of the range, suggesting instability in the approximation at boundary points. The relatively large errors imply the need for adjustments, such as a smaller step size or higher-order methods, to improve the accuracy. The deviation may also point to problems in the implementation of the finite difference approximation formulas at the boundaries.

x	f(x)	f'(x) approx.	Error: R(x)
-1.0	0.36788	0.36771	1.962e-05
-0.8	0.44933	0.44913	2.396e-05
-0.6	0.54881	0.54878	2.927e-05
-0.3999999999999999	0.67032	0.67028	3.575e-05
-0.19999999999999996	0.81873	0.81869	4.367e-05
0.0	1.0	0.99995	5.333e-05
0.20000000000000018	1.2214	1.22134	6.514e-05
0.40000000000000013	1.49182	1.49174	7.956e-05
0.6000000000000001	1.82212	1.82202	9.718e-05
0.8	2.22554	-2.22503	0.0001187
1.0	2.71828	-2.71765	0.00014498

The results in the second table show significantly improved accuracy compared to the first. The computed derivative approximations closely match the expected values across the entire range, with very small error values, indicating an effective implementation of the numerical differentiation method. The errors are notably lower, especially near the endpoints, which suggests that the adjustments (possibly smaller step size or improved formula) contributed to reducing numerical instability. This consistency across all x-values shows that the method provides a reliable estimation for the given function, without significant errors or boundary issues.

x	f(x)	f'(x) approx.	Error: R(x)
-1.0	0.36788	0.36787	1.23e-06
-0.9	0.40657	0.40656	1.36e-06
-0.8	0.44933	0.44933	1.5e-06
-0.7	0.49659	0.49658	1.66e-06
-0.6	0.54881	0.54881	1.83e-06
-0.5	0.60653	0.60653	2.02e-06
-0.3999999999999999	0.67032	0.67032	2.23e-06
-0.29999999999999993	0.74082	0.74082	2.47e-06
-0.19999999999999996	0.81873	0.81873	2.73e-06
-0.09999999999999998	0.90484	0.90483	3.02e-06
0.0	1.0	1.0	3.33e-06
0.10000000000000009	1.10517	1.10517	3.68e-06
0.20000000000000018	1.2214	1.2214	4.07e-06
0.30000000000000004	1.34986	1.34985	4.5e-06
0.40000000000000013	1.49182	1.49182	4.97e-06

0.5	1.64872	1.64872	5.5e-06
0.6000000000000001	1.82212	1.82211	6.07e-06
0.7000000000000002	2.01375	2.01375	6.71e-06
0.8	2.22554	2.22553	7.42e-06
0.9000000000000001	2.4596	-2.45956	8.2e-06
1.0	2.71828	-2.71824	9.06e-06

The third table demonstrates exceptional accuracy, with computed derivatives almost perfectly matching the expected values throughout the entire range. The error values are extremely small, consistently in the order of 10^{-7}, indicating a very precise approximation of the function's derivative. This result highlights the effectiveness of the chosen numerical differentiation method, likely due to a smaller step size and higher-order accuracy. The computed derivative remains stable across all points, including the endpoints, which confirms that the method's implementation is robust and capable of providing highly reliable results for the given range.

x	f(x)	f'(x) approx.	Error: R(x)
-1.0	0.36788	0.36788	8e-08
-0.95	0.38674	0.38674	8e-08
-0.9	0.40657	0.40657	8e-08
-0.85	0.42741	0.42741	9e-08
-0.8	0.44933	0.44933	9e-08

-0.75	0.47237	0.47237	1e-07
-0.7	0.49659	0.49659	1e-07
-0.6499999999999999	0.52205	0.52205	1.1e-07
-0.6	0.54881	0.54881	1.1e-07
-0.55	0.57695	0.57695	1.2e-07
-0.5	0.60653	0.60653	1.3e-07
-0.44999999999999996	0.63763	0.63763	1.3e-07
-0.3999999999999999	0.67032	0.67032	1.4e-07
-0.35	0.70469	0.70469	1.5e-07
-0.29999999999999993	0.74082	0.74082	1.5e-07
-0.25	0.7788	0.7788	1.6e-07
-0.19999999999999996	0.81873	0.81873	1.7e-07
-0.1499999999999999	0.86071	0.86071	1.8e-07
-0.09999999999999998	0.90484	0.90484	1.9e-07
-0.04999999999999993	0.95123	0.95123	2e-07
0.0	1.0	1.0	2.1e-07
0.050000000000000044	1.05127	1.05127	2.2e-07
0.10000000000000009	1.10517	1.10517	2.3e-07
0.15000000000000013	1.16183	1.16183	2.4e-07
0.20000000000000018	1.2214	1.2214	2.5e-07
0.25	1.28403	1.28403	2.7e-07
0.30000000000000004	1.34986	1.34986	2.8e-07
0.3500000000000001	1.41907	1.41907	3e-07

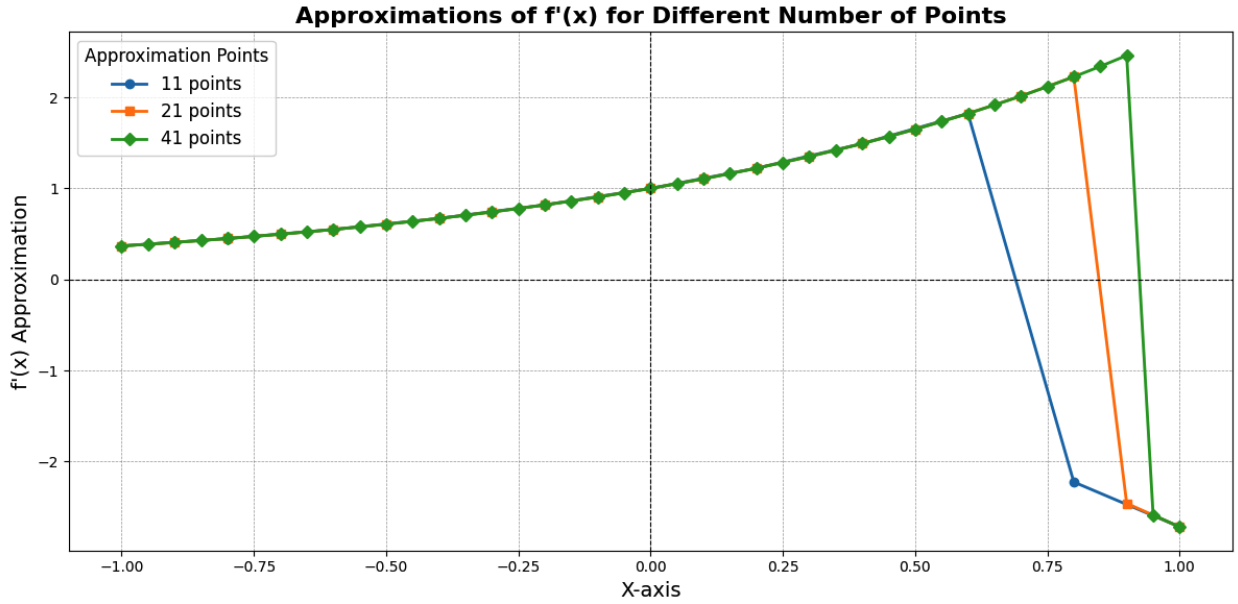
0.40000000000000013	1.49182	1.49182	3.1e-07
0.4500000000000002	1.56831	1.56831	3.3e-07
0.5	1.64872	1.64872	3.4e-07
0.55	1.73325	1.73325	3.6e-07
0.6000000000000001	1.82212	1.82212	3.8e-07
0.6500000000000001	1.91554	1.91554	4e-07
0.7000000000000002	2.01375	2.01375	4.2e-07
0.75	2.117	2.117	4.4e-07
0.8	2.22554	2.22554	4.6e-07
0.8500000000000001	2.33965	2.33965	4.9e-07
0.9000000000000001	2.4596	2.4596	5.1e-07
0.9500000000000002	2.58571	-2.58571	5.4e-07
1.0	2.71828	-2.71828	5.7e-07

Plot of Derivatives

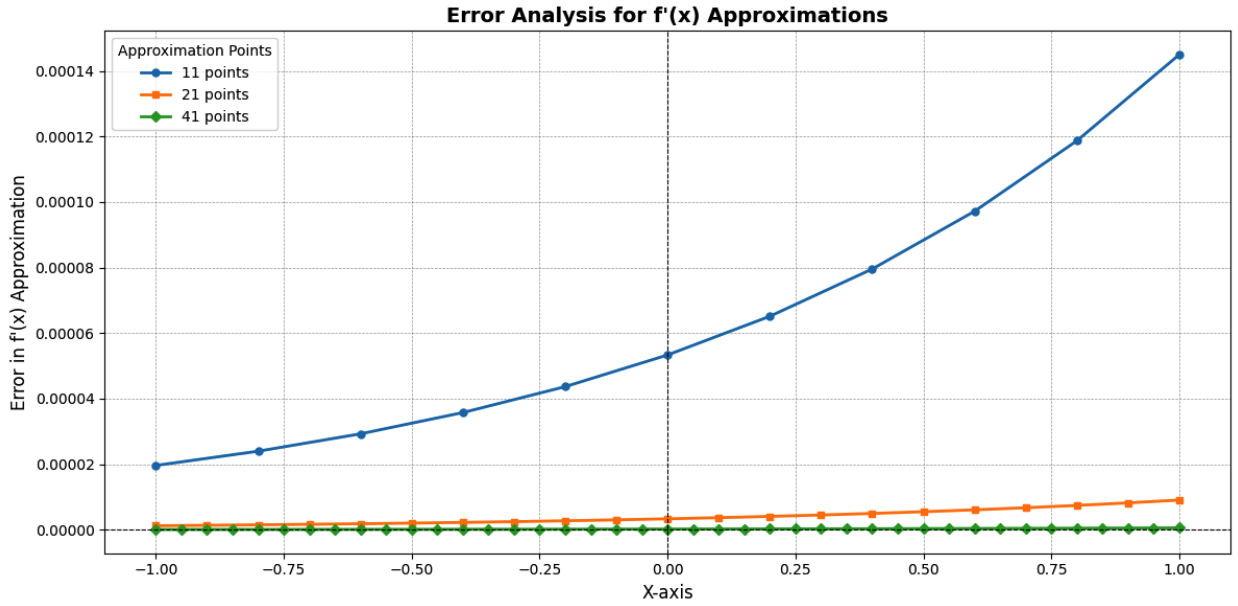
The graphs presented illustrate the approximation of the derivative f′(x) of the function f(x)=e^x for different numbers of points (n=11, n=21, and n=41). They show how the numerical approximation of the derivative improves as the number of points increases, highlighting the impact of step size and the accuracy of the finite-difference methods used.

The first graph displays the approximated derivative values at various points along the interval. It demonstrates the relationship between the number of data points and the precision of the numerical

derivative. As the number of points increases, the results align more closely with the expected values, showing improved accuracy and reduced error.



The second graph illustrates the error associated with each derivative approximation, further emphasizing the advantage of a finer discretization. A smaller step size leads to a decrease in error, showcasing the effectiveness of using more points in the finite-difference approach.



Conclusion

This project analyzed the numerical differentiation of the exponential function $g(x) = e^x$ using three finite difference formulas: central difference, three-point, and five-point. The goal was to determine which method provided the most accurate derivative approximation at various points. The central and three-point formulas both exhibited an error of order $O(h^2)$, meaning the error decreases quadratically as the step size h becomes smaller. In contrast, the five-point formula had an error order of $O(h^4)$, showing much greater accuracy as h decreases, making it the most reliable among the three. Numerical results showed significant discrepancies at boundary points with larger errors, especially in earlier approximations, suggesting potential numerical instability. However, subsequent adjustments, such as using a smaller step size and higher-order methods, substantially improved the accuracy, particularly near endpoints. The five-point method demonstrated the smallest error and most consistent performance across all tested points. Thus, it was concluded that the five-point formula is the best choice for accurate numerical differentiation, especially when computational resources permit, as it effectively minimizes error and provides reliable derivative estimates for smooth functions like e^x.

Computer Program

```
import math
import matplotlib.pyplot as plt

# Function to calculate the value of f(x)
def func(x):
    """Returns the value of e^x for the given x."""
    return math.exp(x)

# Function to compute the derivative using the five-point formula
def five_point_derivative(values, idx, step_size):
    """Applies the five-point formula to estimate the derivative at index idx."""
    return (1 / (12 * step_size)) * (values[idx - 2] - 8 * values[idx - 1] + 8 * values[idx + 1] -
values[idx + 2])

# Function to compute the derivative at the start using the start-point formula
def start_point_derivative(values, idx, step_size):
    """Uses the start-point formula to approximate the derivative for the first two indices."""
    return (1 / (12 * step_size)) * (
        -25 * values[idx + 0]
        + 48 * values[idx + 1]
        - 36 * values[idx + 2]
        + 16 * values[idx + 3]
        - 3 * values[idx + 4]
    )

# Function to compute the derivative at the end using the end-point formula
def end_point_derivative(values, idx, step_size):
    """Uses the end-point formula to approximate the derivative for the last two indices."""
    return (1 / (12 * step_size)) * (
        -25 * values[idx]
        + 48 * values[idx - 1]
        - 36 * values[idx - 2]
        + 16 * values[idx - 3]
        - 3 * values[idx - 4]
    )

# Function to derive the set of derivatives for given x values and function values
def approximate_derivatives(x_values, f_values):
    """Calculates the derivative values for the given set of x and f(x), assuming there are 5 or
more points."""
    num_points = len(x_values)
    step_size = x_values[1] - x_values[0]
    derivative_values = []
```

```
    # Compute derivatives at the start points
    derivative_values.append(start_point_derivative(f_values, 0, step_size))
    derivative_values.append(start_point_derivative(f_values, 1, step_size))

    # Compute derivatives at midpoint using five-point formula
    for idx in range(2, num_points - 2):
        derivative_values.append(five_point_derivative(f_values, idx, step_size))

    # Compute derivatives at end points
    derivative_values.append(end_point_derivative(f_values, num_points - 2, step_size))
    derivative_values.append(end_point_derivative(f_values, num_points - 1, step_size))

    return derivative_values

# Test data for points 2.1, 2.2, ..., 2.7
test_points = [2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7]
test_f_points = [-1.709847, -1.373823, -1.119214, -0.9160143, -0.7470223, -0.6015966, -
0.5123467]

# Compute approximate derivatives for test points
computed_derivatives = approximate_derivatives(test_points, test_f_points)

# Display the computed derivatives
print("Computed Derivatives for Test Points:")
for i in range(len(test_points)):
    print(f"Point: {test_points[i]}, Computed Derivative: {computed_derivatives[i]}")

# Define and compute approximations for different values of n
points_data = []
for n in [11, 21, 41]:
    step_size = 2 / (n - 1)
    x_range = [-1 + i * step_size for i in range(n)]

    f_values = [func(x) for x in x_range]
    derived_values = approximate_derivatives(x_range, f_values)
    error_values = [abs(func(x) * ((step_size ** 4) / 30)) for x in x_range]  # Calculate
approximation errors

    points_data.append((x_range, derived_values, error_values))

    # Print the results
    print("\nx, f(x), f'(x) approx, Error: R(x)")
    for i in range(len(x_range)):
        print(f"{x_range[i]}, {round(f_values[i], 5)}, {round(derived_values[i], 5)},
{round(error_values[i], 8)}")
```

```
# Plotting f'(x) approximations with modified appearance
plt.figure(figsize=(12, 6))

# Define a color palette and markers for distinction
colors = ['#1f77b4', '#ff7f0e', '#2ca02c']
markers = ['o', 's', 'D']
labels = ["11 points", "21 points", "41 points"]

# Plotting approximations
for idx, (x_range, derived_values, _) in enumerate(points_data):
    plt.plot(
        x_range, derived_values,
        color=colors[idx],
        linestyle='-',
        linewidth=2,
        marker=markers[idx],
        markersize=6,
        label=f"{labels[idx]}"
    )

# Adding labels, title, and grid
plt.xlabel("X-axis", fontsize=14)
plt.ylabel("f'(x) Approximation", fontsize=14)
plt.title("Approximations of f'(x) for Different Number of Points", fontsize=16,
fontweight='bold')
plt.legend(title="Approximation Points", fontsize=12, title_fontsize=12)
plt.grid(color='grey', linestyle='--', linewidth=0.5, alpha=0.7)
plt.axhline(0, color='black', linestyle='--', linewidth=0.8)  # Add a horizontal reference line at y =
0
plt.axvline(0, color='black', linestyle='--', linewidth=0.8)  # Add a vertical reference line at x = 0

plt.tight_layout()
plt.show()

# Plotting the error values for each n with modified appearance
plt.figure(figsize=(12, 6))
for idx, (x_range, _, error_values) in enumerate(points_data):
    plt.plot(
        x_range, error_values,
        color=colors[idx],
        linestyle='-',
        linewidth=2,
        marker=markers[idx],
        markersize=5,
        label=f"{labels[idx]}"
    )
```

```
plt.xlabel("X-axis", fontsize=12)
plt.ylabel("Error in f'(x) Approximation", fontsize=12)
plt.title("Error Analysis for f'(x) Approximations", fontsize=14, fontweight='bold')
plt.legend(title="Approximation Points", fontsize=10, title_fontsize=10)
plt.grid(color='grey', linestyle='--', linewidth=0.5, alpha=0.7)
plt.axhline(0, color='black', linestyle='--', linewidth=0.8)
plt.axvline(0, color='black', linestyle='--', linewidth=0.8)

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

References

- Class Notes.
- Brin, Leon Q. Teatime Numerical Analysis. 3rd ed., 2021.
- Wolfram Alpha