1 Introduction to Finite Difference Approximations

Often in scientific computing, we need to compute derivatives of a function f, but we might only have access to the function values at discrete sample points, or the function itself might be too complex to differentiate analytically. Numerical differentiation provides methods to approximate derivatives using these function samples. A common approach is using finite difference approximations, which rely on Taylor series expansions.

Suppose we have a scalar function f and we want to approximate its derivatives based on expressions involving several samples of f. A convenient starting point is Taylor's theorem. For a small value h, we can expand $f(x+h)$ around x:

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

Similarly, for $f(x-h)$:

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

By rearranging these expansions and truncating higher-order terms, we can derive various finite difference formulas.

1.1 Finite Difference Stencils

The term "stencil" encapsulates both the positions of the function samples used and their corresponding weights in a finite difference formula. Here are some common stencils:

- **Forward Difference:** Uses $f(x_i)$ and $f(x_{i+1})$. Approximates $f'(x_i)$.
- **Backward Difference:** Uses $f(x_{i-1})$ and $f(x_i)$. Approximates $f'(x_i)$.
- **Centered Difference (1st Derivative):** Uses $f(x_{i-1})$ and $f(x_{i+1})$. Approximates $f'(x_i)$.
- **Centered Difference (2nd Derivative):** Uses $f(x_{i-1})$, $f(x_i)$, and $f(x_{i+1})$. Approximates $f''(x_i)$.

These can be visualized as follows:

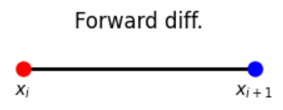

Figure 1: Forward difference stencil using x_i (red) and x_{i+1} (blue).

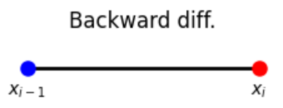

Figure 2: Backward difference stencil using x_{i-1} (blue) and x_i (red).

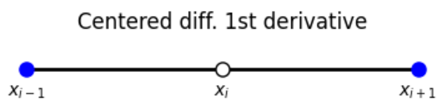

Figure 3: Centered difference stencil for the 1st derivative using x_{i-1} and x_{i+1} (blue). The value at x_i (white) is not directly used.

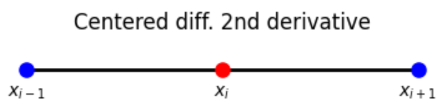

Figure 4: Centered difference stencil for the 2nd derivative using x_{i-1} (blue), x_i (red), and x_{i+1} (blue).

2 Derivation of Finite Difference Formulas

There are two primary methods for deriving finite difference formulas: using Taylor series expansions and differentiating Lagrange interpolants.

2.1 Taylor Series Approach

This approach involves writing Taylor expansions for the function at the stencil points and then linearly combining these expansions to isolate the desired derivative while canceling out lower-order terms.

2.1.1 Forward Difference Formula

From the Taylor expansion of $f(x+h)$:

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

Rearranging for $f'(x)$:

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

Truncating the terms involving h and higher powers gives the forward difference formula:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

The leading error term is $-\frac{f''(x)}{2}h$, so this formula is first-order accurate, with error $O(h)$.

2.1.2 Backward Difference Formula

From the Taylor expansion of $f(x-h)$:

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

Rearranging for $f'(x)$:

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

Truncating gives the backward difference formula:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

This is also first-order accurate, with error $O(h)$.

2.1.3 Centered Difference Formula (1st Derivative)

Subtract the Taylor expansion for $f(x-h)$ from the expansion for $f(x+h)$:

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

$$f(x+h)-f(x-h) = 2f'(x)h+\frac{2f'''(x)}{6}h^3+O(h^5)$$

Rearranging for $f'(x)$:

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

Truncating gives the centered difference formula for the first derivative:

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

The leading error term is $-\frac{f'''(x)}{6}h^2$, so this formula is second-order accurate, with error $O(h^2)$.

2.1.4 Centered Difference Formula (2nd Derivative)

Add the Taylor expansions for $f(x+h)$ and $f(x-h)$:

$$f(x+h)+f(x-h) = (f(x)+f'(x)h+\frac{f''(x)}{2}h^2+\frac{f'''(x)}{6}h^3+\frac{f^{(4)}(x)}{24}h^4+\dots)+(f(x)-f'(x)h+\frac{f''(x)}{2}h^2-\frac{f'''(x)}{6}h^3+\frac{f^{(4)}(x)}{24}h^4+\dots$$

$$f(x+h)+f(x-h) = 2f(x)+f''(x)h^2+\frac{2f^{(4)}(x)}{24}h^4+O(h^6)$$

Rearranging for $f''(x)$:

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

Truncating gives the centered difference formula for the second derivative:

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

The leading error term is $-\frac{f^{(4)}(x)}{12}h^2$, so this formula is second-order accurate, with error $O(h^2)$.

2.2 Lagrange Interpolant Approach

An alternative approach is to construct a polynomial interpolant $P(x)$ that passes through the function samples at the stencil points and then differentiate this polynomial. The derivative of the interpolant $P'(x)$ serves as an approximation to $f'(x)$.

For example, consider a linear interpolant $P_1(x)$ through $(x_0, f(x_0))$ and $(x_0+h, f(x_0+h))$. Using Lagrange basis functions:

$$P_1(x) = f(x_0)\frac{x-(x_0+h)}{x_0-(x_0+h)}+f(x_0+h)\frac{x-x_0}{(x_0+h)-x_0}$$

$$P_1(x) = f(x_0)\frac{x_0+h-x}{h}+f(x_0+h)\frac{x-x_0}{h}$$

Differentiating with respect to x:

$$P_1'(x) = f(x_0)\left(-\frac{1}{h}\right)+f(x_0+h)\left(\frac{1}{h}\right) = \frac{f(x_0+h)-f(x_0)}{h}$$

Evaluating at $x = x_0$ (or anywhere, since $P_1'(x)$ is constant), we recover the forward difference formula. Similarly, using points x_0-h and x_0 yields the backward difference formula.

Using a quadratic interpolant $P_2(x)$ through $(x_0-h, f(x_0-h))$, $(x_0, f(x_0))$, and $(x_0+h, f(x_0+h))$, and evaluating $P_2'(x_0)$ would yield the centered difference formula for the first derivative. Evaluating $P_2''(x_0)$ (which is constant for a quadratic) yields the centered difference formula for the second derivative.

2.3 Example: Second-Order Accurate Formula from Three Points

Let's derive a formula for $f'(x)$ using samples at x, $x+h$, and $x+2h$, aiming for $O(h^2)$ accuracy.

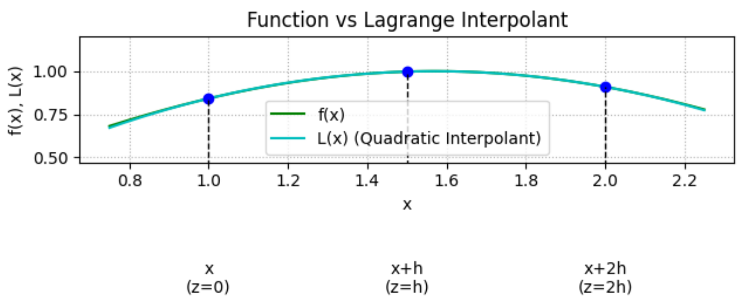

Figure 5: A function $f(x)$ (green) sampled at x, $x+h$, $x+2h$ (blue dots), and the quadratic Lagrange interpolant $L(x)$ (cyan) passing through these points.

2.3.1 Taylor Series Approach

We write the Taylor expansions around x:

$$\begin{aligned} f(x) &= f(x) + 0 \cdot f'(x) + 0 \cdot f''(x) + O(h^3) \\ f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2} f''(x) + O(h^3) \\ f(x+2h) &= f(x) + (2h)f'(x) + \frac{(2h)^2}{2} f''(x) + O(h^3) \\ &= f(x) + 2hf'(x) + 2h^2 f''(x) + O(h^3) \end{aligned}$$

We seek coefficients α, β, γ such that the linear combination:

$$f'_{tay}(x) = \alpha f(x) + \beta f(x+h) + \gamma f(x+2h)$$

approximates $f'(x)$ with $O(h^2)$ error. That is, $f'_{tay}(x) = f'(x) + O(h^2)$. Substituting the Taylor expansions into the stencil formula:

$$\begin{aligned} f'_{tay}(x) = &\alpha(f(x)) \\ &+ \beta(f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \dots) \\ &+ \gamma(f(x) + 2hf'(x) + 2h^2 f''(x) + \dots) \end{aligned}$$

Grouping terms by derivatives of f at x:

$$f'_{tay}(x) = (\alpha + \beta + \gamma) f(x) + (h\beta + 2h\gamma) f'(x) + (\frac{h^2}{2}\beta + 2h^2\gamma) f''(x) + O(h^3)$$

To match $f'(x) + O(h^2)$, we require the coefficients of $f(x)$ and $f''(x)$ to be zero, and the coefficient of $f'(x)$ to be one:

1. Coefficient of $f(x)$: $\alpha + \beta + \gamma = 0$
2. Coefficient of $f'(x)$: $h\beta + 2h\gamma = 1$
3. Coefficient of $f''(x)$: $\frac{h^2}{2}\beta + 2h^2\gamma = 0 \implies \beta + 4\gamma = 0$ (assuming $h \neq 0$)

From (3), $\beta = -4\gamma$. Substituting into (2):

$$h(-4\gamma) + 2h\gamma = 1 \implies -2h\gamma = 1 \implies \gamma = -\frac{1}{2h}$$

Then $\beta = -4\gamma = -4(-\frac{1}{2h}) = \frac{2}{h}$. From (1):

$$\alpha + \frac{2}{h} - \frac{1}{2h} = 0 \implies \alpha + \frac{3}{2h} = 0 \implies \alpha = -\frac{3}{2h}$$

Thus, the finite difference formula is:

$$f'_{tay}(x) = -\frac{3}{2h}f(x) + \frac{2}{h}f(x+h) - \frac{1}{2h}f(x+2h)$$
$$f'_{tay}(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h}$$

This formula is second-order accurate, $f'_{tay}(x) = f'(x) + O(h^2)$.

2.3.2 Lagrange Interpolant Approach

To simplify calculations, introduce a new variable z such that $z = 0$ corresponds to x, $z = h$ corresponds to $x + h$, and $z = 2h$ corresponds to $x + 2h$. We construct the quadratic Lagrange interpolant $L(z)$ through the points $(0, f(x))$, $(h, f(x+h))$, and $(2h, f(x+2h))$. The Lagrange basis polynomials are:

$$L_0(z) = \frac{(z-h)(z-2h)}{(0-h)(0-2h)} = \frac{(z-h)(z-2h)}{2h^2}$$
$$L_1(z) = \frac{(z-0)(z-2h)}{(h-0)(h-2h)} = \frac{z(z-2h)}{-h^2} = -\frac{z(z-2h)}{h^2}$$
$$L_2(z) = \frac{(z-0)(z-h)}{(2h-0)(2h-h)} = \frac{z(z-h)}{2h^2}$$

The interpolant is $L(z) = f(x)L_0(z) + f(x+h)L_1(z) + f(x+2h)L_2(z)$. We need the derivative $L'(z)$ evaluated at $z = 0$. Let's find the derivatives of the basis polynomials:

$$L_0'(z) = \frac{1}{2h^2}[(1)(z-2h) + (z-h)(1)] = \frac{2z-3h}{2h^2}$$
$$L_1'(z) = -\frac{1}{h^2}[(1)(z-2h) + z(1)] = -\frac{2z-2h}{h^2} = \frac{2h-2z}{h^2}$$
$$L_2'(z) = \frac{1}{2h^2}[(1)(z-h) + z(1)] = \frac{2z-h}{2h^2}$$

Now evaluate these derivatives at $z = 0$:

$$L_0'(0) = \frac{-3h}{2h^2} = -\frac{3}{2h}$$
$$L_1'(0) = \frac{2h}{h^2} = \frac{2}{h}$$
$$L_2'(0) = \frac{-h}{2h^2} = -\frac{1}{2h}$$

The derivative of the interpolant at $z = 0$ (which corresponds to x) is:

$$f'_{lag}(x) = L'(0) = f(x)L_0'(0) + f(x+h)L_1'(0) + f(x+2h)L_2'(0)$$
$$f'_{lag}(x) = f(x)\left(-\frac{3}{2h}\right) + f(x+h)\left(\frac{2}{h}\right) + f(x+2h)\left(-\frac{1}{2h}\right)$$
$$f'_{lag}(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h}$$

2.3.3 Comparison

Both the Taylor series approach and the Lagrange interpolant approach yield the exact same second-order accurate formula.

- **Taylor Series Approach:** Perhaps more direct for simpler cases. Requires setting up and solving a linear system for the coefficients (α, β, γ). This system can become larger and more complex for higher-order formulas or wider stencils.
- **Lagrange Interpolant Approach:** Involves constructing and differentiating the interpolating polynomial. This avoids solving a linear system directly, which might be advantageous for deriving more complex or higher-order stencils, as the process is more algorithmic.

3 Differentiation Matrices

So far, we've focused on approximating the derivative at a single point x_i. If we want to approximate the derivative $f'(x)$ over an entire interval $[a, b]$, we can discretize the interval into points $x_1, x_2, \ldots, x_n$ (usually equally spaced, $x_i = a + (i-1)h$ with $h = (b-a)/(n-1)$) and calculate the derivative approximation at each point simultaneously.

Let $\mathbf{f}$ be the vector of function values at these points:

$$\mathbf{f} = [f(x_1), f(x_2), \ldots, f(x_n)]^T$$

Let $\mathbf{f}'$ be the vector of true derivatives:

$$\mathbf{f}' = [f'(x_1), f'(x_2), \ldots, f'(x_n)]^T$$

And let $\tilde{\mathbf{f}}'$ be the vector of finite difference approximations to the derivatives:

$$\tilde{\mathbf{f}}' = [\tilde{f}'(x_1), \tilde{f}'(x_2), \ldots, \tilde{f}'(x_n)]^T$$

Since differentiation is a linear operation, and finite difference formulas are linear combinations of function values, the mapping from $\mathbf{f}$ to $\tilde{\mathbf{f}}'$ can be represented by an $n \times n$ matrix $\mathbf{D}$, called the **differentiation matrix**:

$$\tilde{\mathbf{f}}' \approx \mathbf{Df}$$

Each row i of the matrix $\mathbf{D}$ contains the coefficients of the finite difference stencil used to approximate $f'(x_i)$.

3.1 Example: Forward Difference Matrix

Using the forward difference formula $\tilde{f}'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h}$, the i-th row of $\mathbf{D}$ needs to implement this calculation: $\tilde{f}'(x_i) = -\frac{1}{h} f(x_i) + \frac{1}{h} f(x_{i+1})$. This means that for a typical row i (where $1 \le i < n$), the non-zero entries are:

- $D_{i,i} = -\frac{1}{h}$ (coefficient of $f(x_i)$)
- $D_{i,i+1} = \frac{1}{h}$ (coefficient of $f(x_{i+1})$)

All other entries in row i are zero. This results in a sparse matrix with two non-zero diagonals: the main diagonal and the first super-diagonal.

Here is Python code using NumPy to construct such a matrix (initially ignoring the last row issue):

```python
import numpy as np

n = 100 # Number of points
h = 1.0 / (n - 1) # Grid spacing assuming interval [0, 1]

# Create diagonals
```

```
main_diag = -np.ones(n) / h
super_diag = np.ones(n - 1) / h

# Construct matrix using np.diag
D_forward_incomplete = np.diag(main_diag) + np.diag(super_diag, 1)

# Note: D_forward_incomplete is missing the correct last row
```

Visualizing the sparsity pattern using ‘matplotlib.pyplot.spy‘:

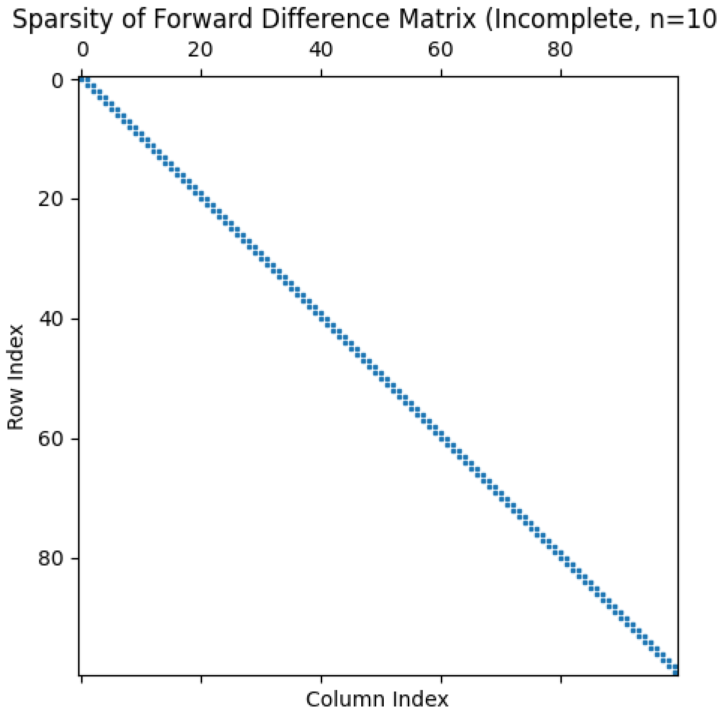

Figure 6: Sparsity pattern of the forward difference matrix before correcting the last row. Note the missing entry in the super-diagonal for the last row.

3.2 Handling Boundary Conditions

The forward difference formula $f'(x_i) \approx \frac{f(x_{i+1})-f(x_i)}{h}$ requires the value $f(x_{i+1})$. For the last point x_n, this would require $f(x_{n+1})$, which is outside our interval $[a, b]$.

To remedy this, we can use a different formula for the last row (or first row if using backward differences). A common choice is to use a formula of the same order of accuracy. Since the forward difference is $O(h)$, we can use the first-order accurate backward difference formula for the last point:

$$\tilde{f}'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{h}$$

This means the non-zero entries for the last row ($i = n$) are:

- $D_{n,n-1} = -\frac{1}{h}$
- $D_{n,n} = \frac{1}{h}$

With this modification, the differentiation matrix $\mathbf{D}$ is fully defined.

Corrected Python code:

```
import numpy as np

n = 100 # Number of points
h = 1.0 / (n - 1) # Grid spacing assuming interval [0, 1]

# Create matrix using diagonals
D = np.diag(-np.ones(n) / h) + np.diag(np.ones(n - 1) / h, 1)

# Fix the last row using backward difference
D[n-1, n-1] = 1.0 / h
D[n-1, n-2] = -1.0 / h

# Now D is the complete forward difference matrix with backward diff at
    ↪ boundary
```

The sparsity pattern now looks like this:

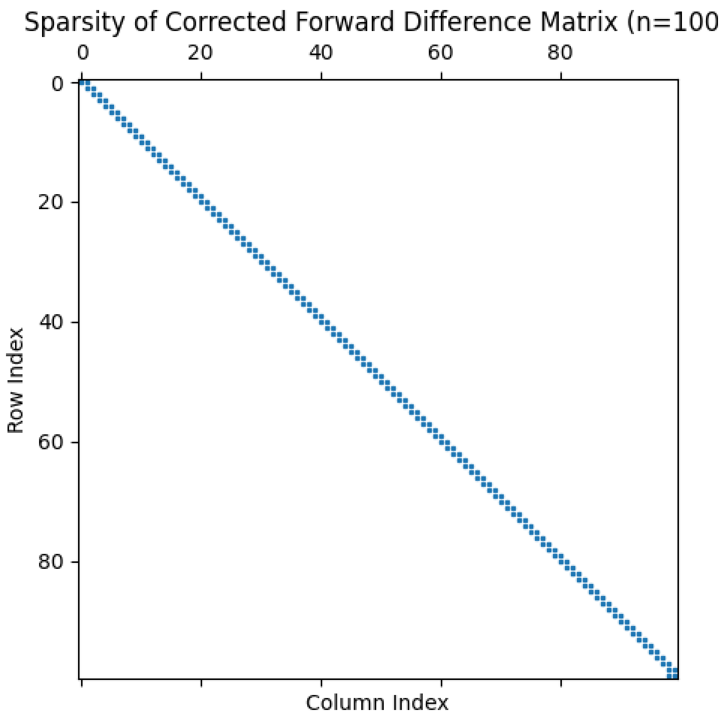

Figure 7: Sparsity pattern of the forward difference matrix after correcting the last row using a backward difference formula.

Note that with this specific correction, the last row ($i = n$) uses the stencil $D_{n,n-1} = -1/h, D_{n,n} = 1/h$, and the second-to-last row ($i = n-1$) uses the stencil $D_{n-1,n-1} = -1/h, D_{n-1,n} = 1/h$. These are essentially the same stencil (backward difference) applied at x_n and x_{n-1} respectively (relative to their own index). This might result in the last two rows of the matrix being identical, depending on the exact construction. However, this is still acceptable as both rows represent valid $O(h)$ estimates for the derivative at their respective points x_n and x_{n-1}.

For higher accuracy, one could use higher-order finite difference formulas for the interior points and specialized one-sided formulas (like the one derived in the example) near the boundaries to maintain the overall order of accuracy.

4 Application and Error Analysis

Let's apply the differentiation matrix to approximate the derivative of the function $f(x) = \frac{1}{2+\cos(10x)}$ on the interval $[0, 1]$. The analytical derivative is $f'(x) = \frac{10\sin(10x)}{(2+\cos(10x))^2}$.

We will use the forward difference matrix with the backward difference correction at the boundary.

4.1 Implementation Steps

1. Define the function $f(x)$ and its analytical derivative $f'(x)$. 2. Choose the number of points n and calculate the grid spacing h. 3. Create the grid points $x_1, \ldots, x_n$. 4. Evaluate the function at the grid points to get the vector $\mathbf{f}$. 5. Construct the differentiation matrix $\mathbf{D}$ (e.g., forward difference with boundary correction). 6. Compute the numerical derivative vector $\tilde{\mathbf{f}}' = \mathbf{D}\mathbf{f}$. 7. Evaluate the analytical derivative at the grid points to get $\mathbf{f}'$. 8. Calculate the error vector $\mathbf{e} = \tilde{\mathbf{f}}' - \mathbf{f}'$. 9. Plot the results: sparsity pattern, function, numerical vs analytical derivative, and error.

4.2 Results for n=100

Python code structure for analysis:

```
import numpy as np
import matplotlib.pyplot as plt

# Function and derivative
def f(x):
    return 1.0 / (2.0 + np.cos(10.0 * x))

def df(x):
    return (10.0 * np.sin(10.0 * x)) / ((2.0 + np.cos(10.0 * x))**2)

# Setup
n = 100
x = np.linspace(0, 1, n)
h = x[1] - x[0]
y = f(x)
dy_analytical = df(x)

# Differentiation Matrix (Forward + Backward boundary)
D = np.diag(-np.ones(n) / h) + np.diag(np.ones(n - 1) / h, 1)
D[n-1, n-1] = 1.0 / h
D[n-1, n-2] = -1.0 / h

# Numerical derivative
dy_numerical = D @ y

# Error
error = dy_numerical - dy_analytical

# Plotting (omitted here, see verbatim blocks below)
# plt.spy(D) ...
# plt.plot(x, y) ...
# plt.plot(x, y, label='f(x)')
# plt.plot(x, dy_numerical, label='f\'(x) numerical') ...
# plt.plot(x, error) ...
```

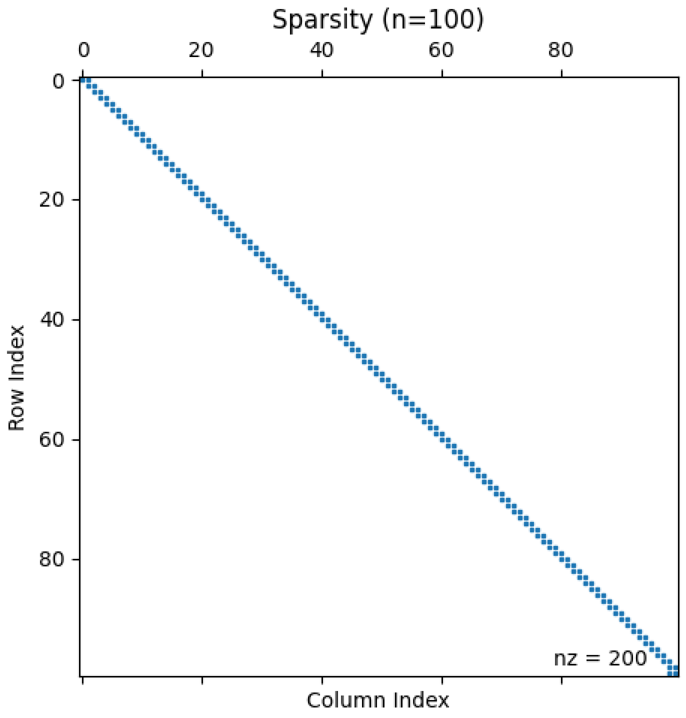

Figure 8: Sparsity pattern of the differentiation matrix for n=100.

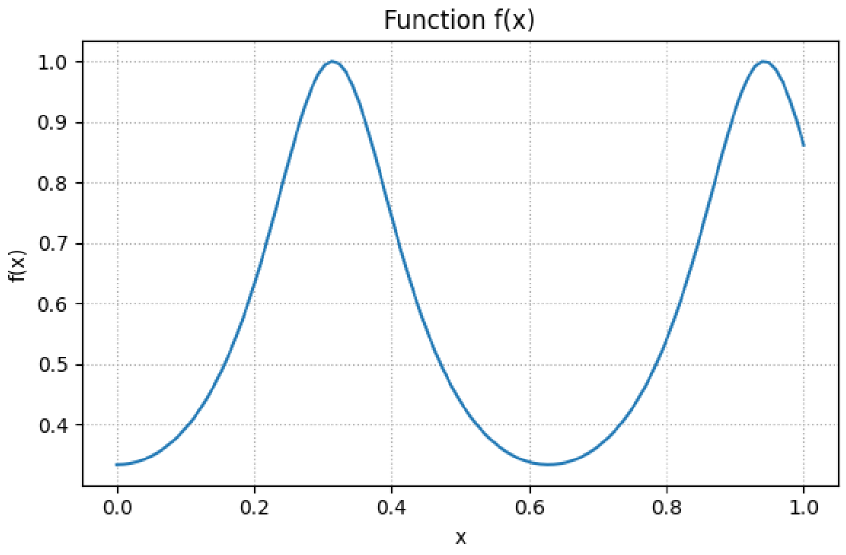

Figure 9: Plot of the function $f(x) = 1/(2 + \cos(10x))$ for n=100 points.

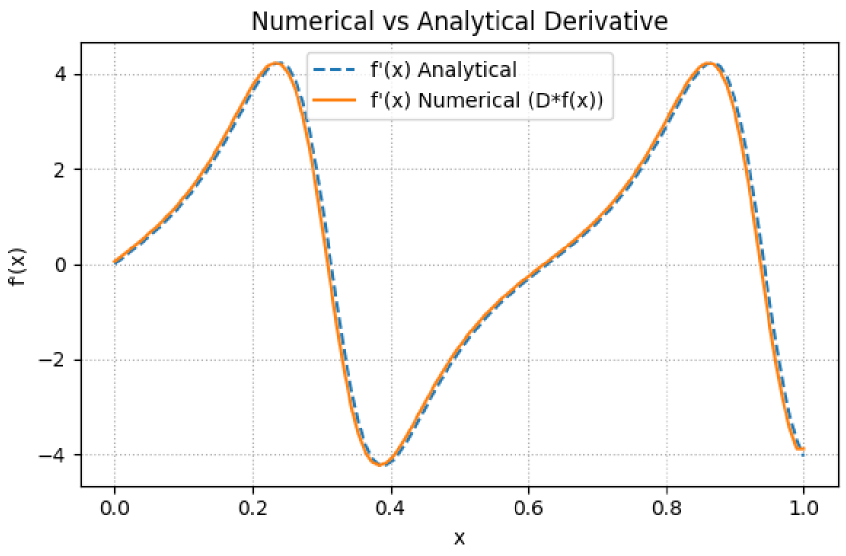

Figure 10: Numerical derivative (solid line) computed using the differentiation matrix vs the analytical derivative (dashed line) for n=100.

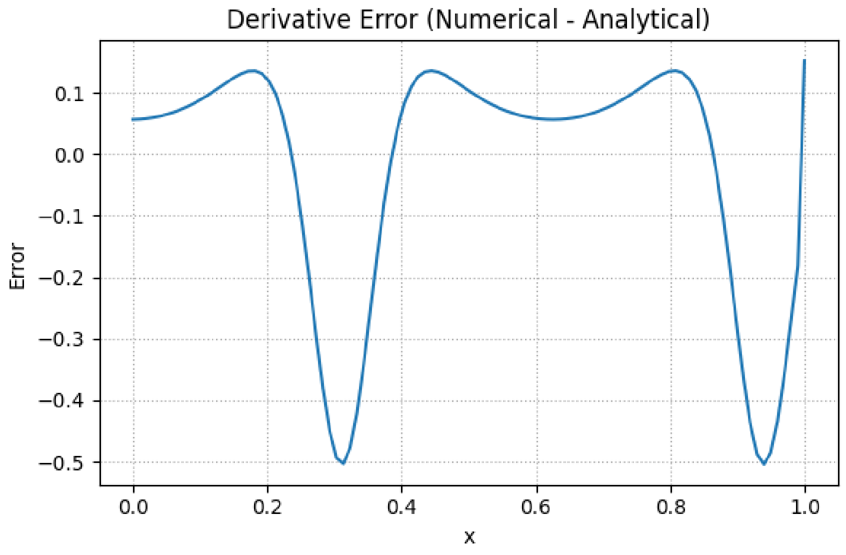

Figure 11: Error between the numerical and analytical derivative for n=100.

Analysis (n=100): The maximum error magnitude is around 0.5. The largest errors occur near the peaks of the original function $f(x)$. Since the derivative magnitude reaches about 4, the relative error is roughly $0.5/4 \approx 12.5\%$, or around 10% as mentioned in the transcript analysis.

4.3 Results for n=200

Increasing the number of points should decrease the error for a convergent method.

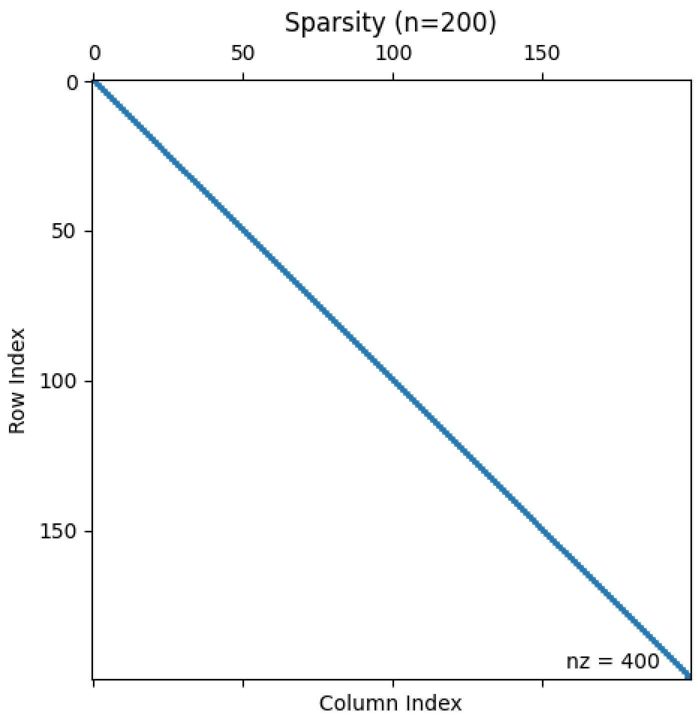

Figure 12: Sparsity pattern of the differentiation matrix for n=200.

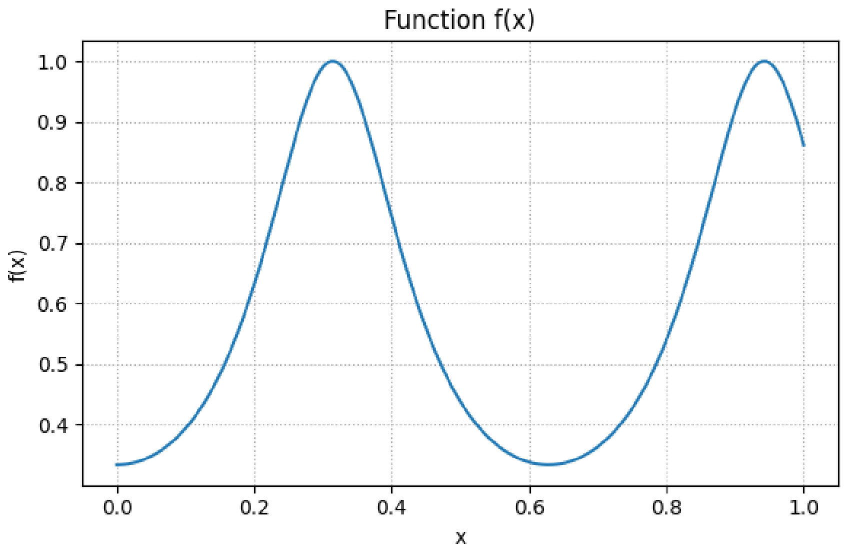

Figure 13: Plot of the function $f(x)$ for n=200 points.

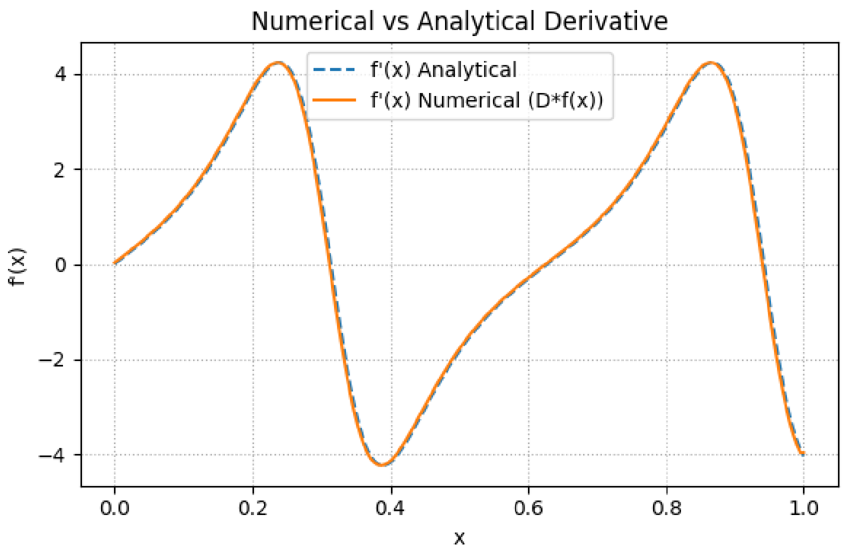

Figure 14: Numerical derivative vs analytical derivative for n=200.

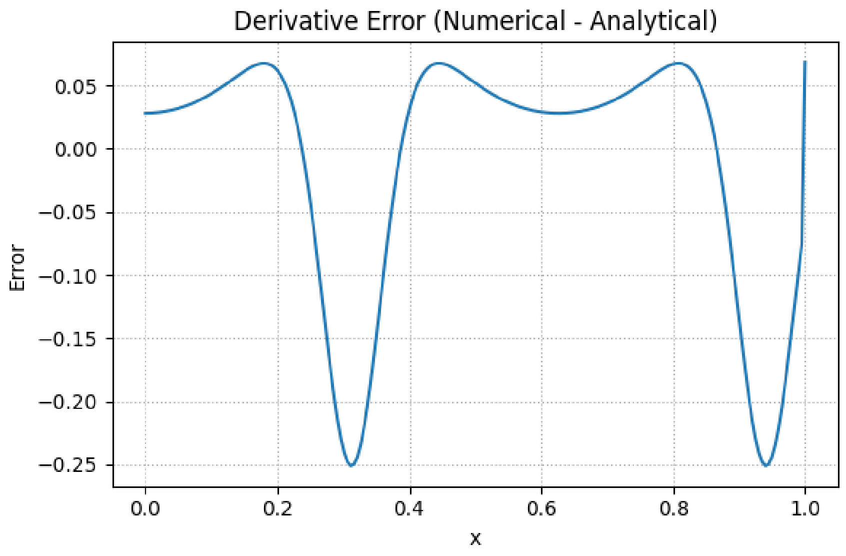

Figure 15: Error between the numerical and analytical derivative for n=200.

Analysis (n=200): Doubling the number of points (halving h) reduces the maximum error magnitude to about 0.25. This halving of the error is consistent with the $O(h)$ accuracy of the first-order forward/backward difference formulas used in constructing the matrix $\mathbf{D}$.

4.4 Efficiency Considerations

While conceptually simple, constructing and using $\mathbf{D}$ as a dense $n \times n$ matrix can be inefficient, especially for large n. The matrix-vector multiplication $\mathbf{Df}$ involves $O(n^2)$ operations, even though $\mathbf{D}$ is sparse (only $O(n)$ non-zero elements). In practice, it is much more efficient to store $\mathbf{D}$ using a sparse matrix format (e.g., Compressed Sparse Row (CSR) or Diagonal format (DIA) from SciPy's sparse module) and use specialized algorithms for sparse matrix-vector multiplication, which typically take only $O(n)$ operations, proportional to the number of non-zero elements.