Università della Svizzera italiana

Faculty of Informatics

Introduction to Computational Science

Assignment3

Zhixiang Dai, Daniele del Pozzo

Spring Semester, March 27, 2025

1 Lagrange Basis Polynomials

1.1

We want to show that:

$$\sum_{i=0}^{n} L_i(x) = 1$$

for any $x \in \mathbb{R}$.

Step 1

The Lagrange basis polynomials are defined as:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

The interpolation polynomial is:

$$P(x) = \sum_{i=0}^{n} f(x_i) L_i(x)$$

Step 2

If we set $f(x_i) = 1$ for all i, then:

$$P(x) = \sum_{i=0}^{n} L_i(x)$$

Since $P(x)$ is a polynomial of degree at most n that is equal to 1 at every x_i. Therefore, for any x:

$$\sum_{i=0}^{n} L_i(x) = 1$$

2 Barycentric Interpolation

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


def barycentric_weights(x_nodes):
    n = len(x_nodes)
    tilde_rho = np.zeros(n)
    sigma = np.ones(n)

    for i in range(n):
        log_sum = 0.0
        sign_product = 1
        for j in range(n):
            if j != i:
                diff = x_nodes[i] - x_nodes[j]
                log_sum += np.log(abs(diff))
                if diff < 0:
                    sign_product = -sign_product
        tilde_rho[i] = log_sum
        sigma[i] = sign_product

    min_tilde_rho = np.min(tilde_rho)
    rho = sigma * np.exp(-(tilde_rho - min_tilde_rho))
    return rho

def barycentric_interpolation(f_values, rho, x, x_nodes):
    p = np.zeros_like(x, dtype=float)
    n = len(x_nodes)

    for j in range(len(x)):
        node_index = np.where(x_nodes == x[j])[0]
        if len(node_index) > 0:
            p[j] = f_values[node_index[0]]
        else:
            denom = 0.0
            numer = 0.0
            for i in range(n):
                weight = rho[i] / (x[j] - x_nodes[i])
                denom += weight
                numer += weight * f_values[i]
            p[j] = numer / denom
    return p

if __name__ == "__main__":
    x_nodes = np.array([0, 1, 2, 3, 4], dtype=float)
    rho = barycentric_weights(x_nodes)

    x_plot = np.linspace(0, 4, 401)

    plt.figure()
    for i in range(len(x_nodes)):
        f_vals = np.zeros_like(x_nodes, dtype=float)
        f_vals[i] = 1.0
        L_i = barycentric_interpolation(f_vals, rho, x_plot, x_nodes)
        plt.plot(x_plot, L_i, label=f"L_{i}")
```

```
plt.scatter(x_nodes, np.eye(len(x_nodes))[range(len(x_nodes)),
 ↪  range(len(x_nodes))], color='black')
plt.legend()
plt.title("Lagrange Basis Polynomials (Barycentric Form)")
plt.show()
```

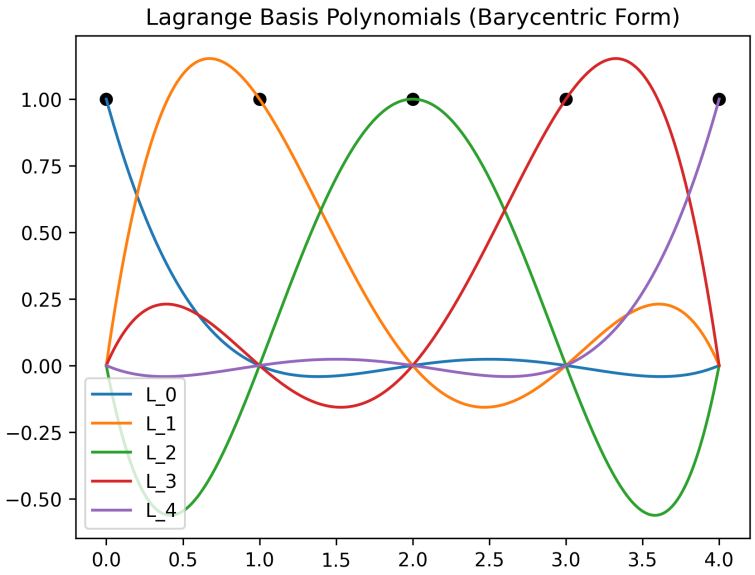

Figure 1: Barycentric Interpolation

3 Interpolation Polynomial and Error Estimate

3.1

We are given the function:

$$f(x) = \frac{2}{3+2x}$$

And nodes:

$$x_0 = -1, \quad x_1 = -0.5, \quad x_2 = 0.5, \quad x_3 = 1$$

Step 1

The Lagrange interpolation polynomial is:

$$p(x) = \sum_{i=0}^{3} f(x_i) L_i(x)$$

We start by evaluating $f(x)$ with their corresponding values:

$$f(x_0) = 2, \quad f(x_1) = 1, \quad f(x_2) = 0.5, \quad f(x_3) = 0.4$$

As a result:

$$p(x) = 2L_0(x) + 1L_1(x) + 0.5L_2(x) + 0.4L_3(x)$$

Step 2

The error bound is given by:

$$\max_{x\in[-1,1]} |f(x) - p(x)| \leq \frac{\max_{x\in[-1,1]} |f^{(4)}(x)|}{4!} \max_{x\in[-1,1]} |(x-x_0)(x-x_1)(x-x_2)(x-x_3)|$$

We now evaluate the fourth derivative:

$$f^{(4)}(x) = \frac{768}{(3+2x)^5}, \quad \max_{x\in[-1,1]} |f^{(4)}(x)| = 768$$

We evaluate the product term:

$$\max_{x\in[-1,1]} |(x+1)(x+0.5)(x-0.5)(x-1)| = \frac{1}{4}$$

Step 3

Final error estimate:

$$\max_{x\in[-1,1]} |f(x) - p(x)| \leq \frac{768 \times \frac{1}{4}}{24} = 8$$

Ultimately, the estimate matches the given bound.

4 Runge function

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


def runge_function(x):
    return 1.0 / (1.0 + 25.0 * x ** 2)


def equidistant_nodes(n):
    return np.linspace(-1, 1, n)


def chebyshev_nodes_first_kind(n):
    k = np.arange(n)
    return np.cos((2 * k + 1) * np.pi / (2 * n))


def chebyshev_nodes_second_kind(n):
    k = np.arange(n)
    return np.cos(k * np.pi / (n - 1))


def barycentric_weights(x_nodes):
    n = len(x_nodes)
    tilde_rho = np.zeros(n)
    sigma = np.ones(n)
```

```
    for i in range(n):
        log_sum = 0.0
        sign_product = 1
        for j in range(n):
            if j != i:
                diff = x_nodes[i] - x_nodes[j]
                log_sum += np.log(abs(diff))
                if diff < 0:
                    sign_product = -sign_product
        tilde_rho[i] = log_sum
        sigma[i] = sign_product

    min_val = np.min(tilde_rho)
    rho = sigma * np.exp(-(tilde_rho - min_val))
    return rho

def barycentric_interpolation(f_values, rho, x_eval, x_nodes):
    p = np.zeros_like(x_eval, dtype=float)
    n = len(x_nodes)

    for i, x in enumerate(x_eval):
        # Check if x == any node
        idx = np.where(np.isclose(x_nodes, x))[0]
        if len(idx) > 0:
            p[i] = f_values[idx[0]]
        else:
            denom = 0.0
            numer = 0.0
            for k in range(n):
                weight = rho[k] / (x - x_nodes[k])
                denom += weight
                numer += weight * f_values[k]
            p[i] = numer / denom
    return p


if __name__ == "__main__":
    node_counts = [5, 10, 15]
    x_fine = np.linspace(-1, 1, 400)
    f_exact = runge_function(x_fine)

    plt.figure(figsize=(10, 8))

    node_types = [
        ("Equidistant", equidistant_nodes),
        ("Chebyshev-1", chebyshev_nodes_first_kind),
        ("Chebyshev-2", chebyshev_nodes_second_kind),
    ]

    index_plot = 1

    for n in node_counts:
        for (label, node_func) in node_types:
            x_nodes = node_func(n)
            y_nodes = runge_function(x_nodes)
```

```
            rho = barycentric_weights(x_nodes)
            p_approx = barycentric_interpolation(y_nodes, rho, x_fine, x_nodes)

            plt.subplot(len(node_counts), len(node_types), index_plot)
            plt.plot(x_fine, f_exact, 'k-', label="Exact f(x)")
            plt.plot(x_fine, p_approx, '--', label="Interp")
            plt.scatter(x_nodes, y_nodes, color='red', zorder=5, label="Nodes")
            plt.ylim([-0.2, 1.2])
            plt.title(f"N={n}, {label}")
            plt.legend(fontsize=7, loc="upper center")

            index_plot += 1

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

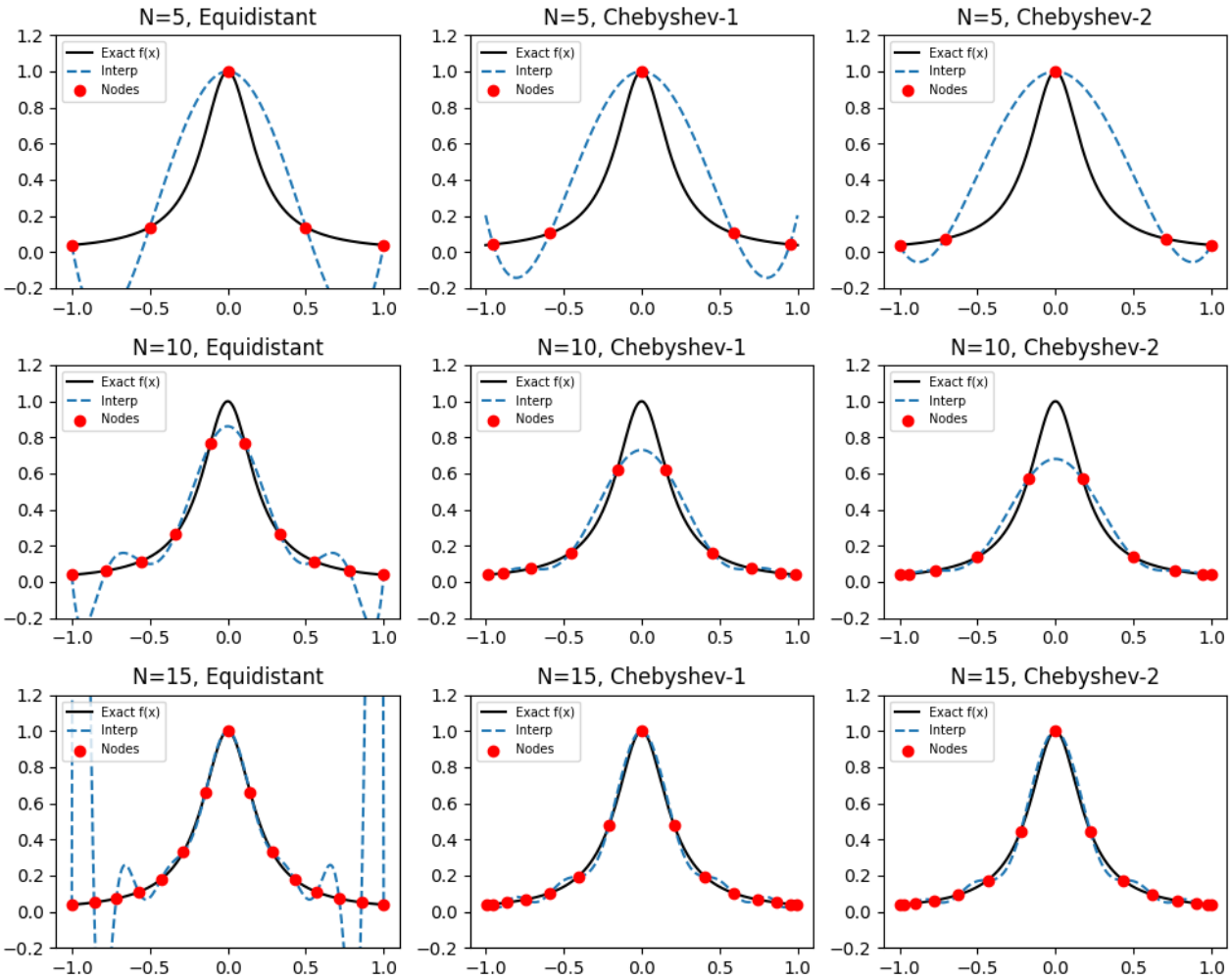

Figure 2: Runge function