Runge polynomial approximation

*Report 1 on the course "Numerical Analysis".

1st Chen Yihang
Peking University
1700010780

***Abstract*—This report implements Newton, Lagrange, linear, Hermite polynomial and natural spline interpolation over the Runge function. We detail our implementation and plot the interpolation function as well as interpolation error in a series of figures.**

I. Problem Statement

Use the following method to interpolate the Runge function $R(x) = \frac{1}{1+x^2}, x \in [-5, 5]$, and compare the interpolation function with $R(x)$.

1) For nodes $x_i = -5+i (i = 0, 1, \cdots, 10)$, plot its Newton polynomial interpolation function.
2) For nodes $x_i = 5\cos(\frac{2i+1}{42}\pi) (i = 0, 1, \cdots, 20)$, plot its Lagrange polynomial interpolation function.
3) For nodes $x_i = -5 + i (i = 0, 1, \cdots, 10)$, plot its piecewise linear interpolation function.
4) For nodes $x_i = -5 + i (i = 0, 1, \cdots, 10)$, plot its piecewise 3-rd order Hermite polynomial interpolation function.
5) For nodes $x_i = -5 + i (i = 0, 1, \cdots, 10)$, plot its piecewise 3-rd order natural spline interpolation function.

II. Interpolation methods

A. Newton polynomial

Define the divided difference by

$$\begin{aligned} &f[x_i, x_{i+1}, \cdots, x_{i+j}] \\ =&\frac{f[x_{i+1}, x_{i+2}, \cdots, x_{i+j}] - f[x_i, x_{i+1}, \cdots, x_{i+j}]}{x_{i+j} - x_i} \end{aligned} \tag{1}$$

We can recursively calculate thef difference. Then, we can directly write the Newton interpolation polynomial:

$$N_n(x) = f[x_0] + \sum_{m=1}^{n} f[x_0, x_1, \cdots, x_m](x - x_0)(x - x_1)\cdots(x - x_{m-1}) \tag{2}$$

We use a matrix to encode the divided difference.

B. Lagrange polynomial

We can write the Lagrange interpolation polynomial directly:

$$L_n(x) = \sum_{i=1}^{n} f(x_i)\frac{\prod_{j\neq i}(x - x_j)}{\prod_{j\neq i}(x_i - x_j)} \tag{3}$$

C. Linear polynomial

Similarly, we can write

$$\phi(x) = \frac{f(x_{i+1})(x - x_i) - f(x_i)(x - x_{i+1})}{x_{i+1} - x_i}, \quad x \in [x_i, x_{i+1}], \quad 0 \leq i < n. \tag{4}$$

D. Hermite polynomial

If we have f on the endpoints of the interval $[x_i, x_{i+1}]$

$$y_i = f(x_i), \quad y_{i+1} = f(x_{i+1}), \quad m_i = f'(x_i), \quad m_{i+1} = f'(x_{i+1}) \tag{5}$$

We need to find a third order polynomial $H_3(x)$, s.t.

$$H_3(x_i) = y_i, \quad H_3(x_{i+1}) = y_{i+1}, \quad H_3(x_i) = m_i, \quad H_3(x_{i+1}) = m_{i+1} \tag{6}$$

Then we plan to construt a basis of $\mathcal{P}_3$ explicitly

$$\begin{aligned} \alpha_i(x) &= (1 + 2\frac{x - x_i}{x_{i+1} - x_i})(\frac{x - x_{i+1}}{x_{i+1} - x_i})^2 \\ \alpha_{i+1}(x) &= (1 + 2\frac{x - x_{i+1}}{x_{i+1} - x_i})(\frac{x - x_i}{x_{i+1} - x_i})^2 \\ \beta_i(x) &= (x - x_i)(\frac{x - x_{i+1}}{x_{i+1} - x_i})^2 \\ \beta_{i+1}(x) &= (x - x_{i+1})(\frac{x - x_i}{x_{i+1} - x_i})^2 \end{aligned} \tag{7}$$

We can easily represent the Hermite polynomial on the interval

$$H_3(x) = \sum_i [y_i\alpha_i(x) + m_i\beta_i(x)] \tag{8}$$

E. Natural spline interpolation

The key difference is that we do not use $m_i = f'(x_i)$. Instead, we should use the condition

$$\begin{cases} S_0''(x_0) = S_{n-1}''(x_n) = 0 \\ S_{i-1}''(x_i) = S_i''(x_i), \quad (1 \leq i \leq n-1) \end{cases} \tag{9}$$

Define

$$\begin{cases} \lambda_0 = 1, \quad \lambda_n = 0, \\ \lambda_i = \frac{\Delta x_{i-1}}{\Delta x_{i-1} + \Delta x_i} \end{cases} \tag{10}$$

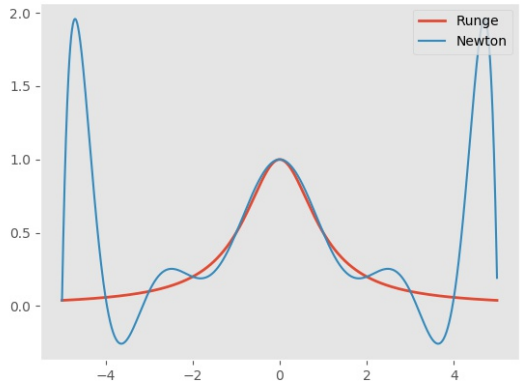
$$\begin{cases} \mu_0 = 3\frac{y_1 - y_0}{\Delta x_0}, \quad \mu_n = 3\frac{y_n - y_{n-1}}{\Delta x_{n-1}} \\ \mu_i = 3[\frac{1-\lambda_i}{\Delta x_{i-1}}(y_i - y_{i-1}) + \frac{\lambda_i}{\Delta x_i}(y_{i+1} - y_i)] \end{cases} \tag{11}$$

We can stack the equations into $Am = \mu$, where A is a tridiagnoal matrix. In the code, we implement the numerical method to solve the tridiagnoal linear system.

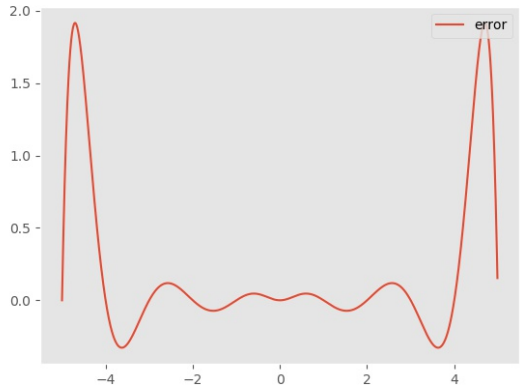
III. Results

To reproduce the results, please execute "interp.py".

We can clearly observe the Runge phenonmenon in the Newton polynomial interpolation of uniform distance in Figure 1. If we use Chebyshev's interpolation nodes, the error is sigificantly reduced, as Figure 2 shows. The piecewise linear interpolation, in Figure 3, is not C^1 smooth, and the piecewise Hermite interpolation is not C^2 smooth. However, natural spline is more computationally onerous than the Hermite interpolation, and if we compare the Figure 4 and 5, it seems that Hermite slightly outperforms the natural spline in terms of l^∞ error.

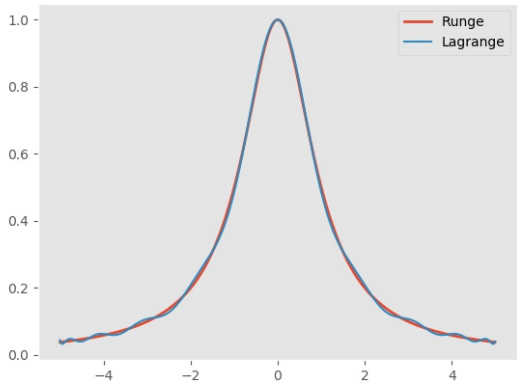


(a) Newton polynomial

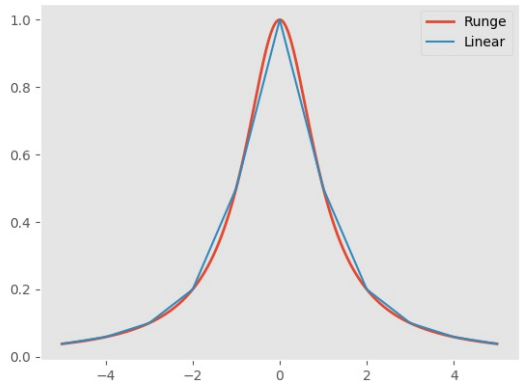


(b) Newton polynomial, error

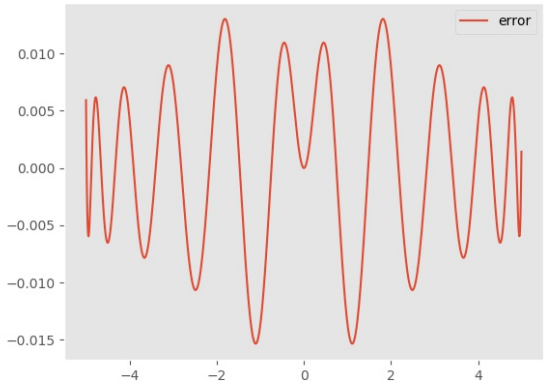
Fig. 1. Newton polynomial



(a) Lagrange polynomial

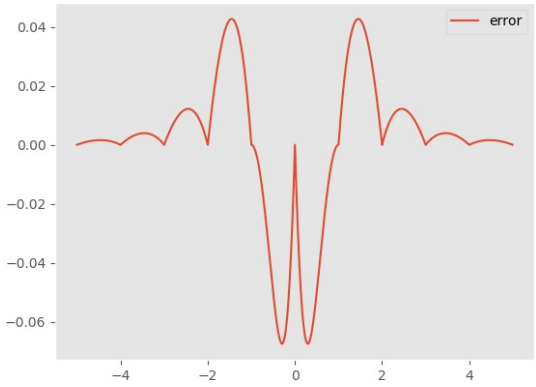


(a) Linear polynomial



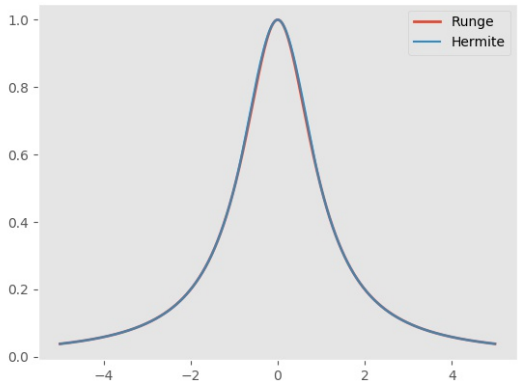
(b) Lagrange polynomial, error

Fig. 2. Lagrange polynomial

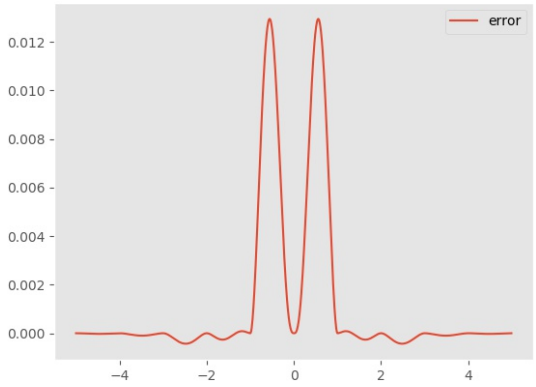


(b) Linear polynomial, error

Fig. 3. Linear polynomial

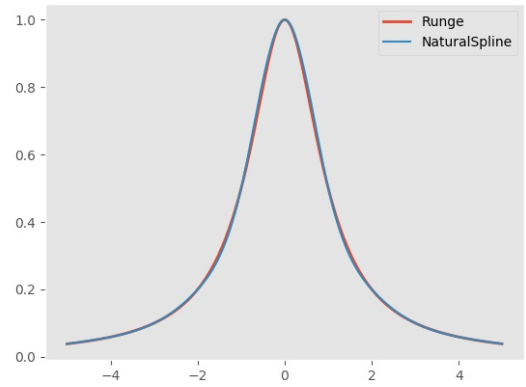


(a) Hermite polynomial

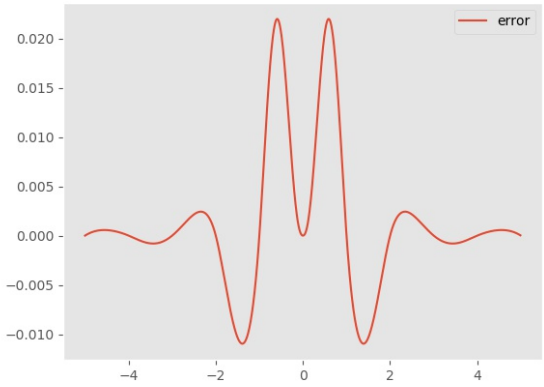


(b) Hermite polynomial, error

Fig. 4. 3-rd order Hermite polynomial



(a) Natural Spline



(b) Natural Spline, error

Fig. 5. 3-rd order natural spline polynomial