Final Report: Spectral Integration and Two-Point Boundary Value Problems

Shih-Ming Wang

1 Problem Description

We consider the numerical solution of linear second-order two-point boundary value problems (BVPs) with constant coefficients. The governing differential equation is given by:

$$Lu := u''(x) + \mu u'(x) + \nu u(x) = f(x), \quad x \in [-1, 1] \tag{1}$$

subject to the Dirichlet boundary conditions:

$$u(-1) = \alpha, \quad u(1) = \beta \tag{2}$$

where μ and ν are real constants, and $f(x)$ is a known function. While spectral methods typically approximate the solution $u(x)$ by a truncated Chebyshev series $u_N(x) = \sum_{k=0}^{N} a_k T_k(x)$, determining the coefficients by spectral differentiation matrix is a big numerical challenge. The differentiation matrix $\mathbf{D}_N$ becomes ill-conditioned as the number of nodes N increases, with an error amplification factor of $O(N^2)$. This instability leads to a loss of precision, particularly for stiff problems or when high-order derivatives are required.

2 Method Overview

To overcome the instability of differentiation, the method proposed by Greengard (1991) reformulates the differential equation as an integral equation. The key idea is that the integration operator is bounded and well-conditioned (error amplification < 2.4) which are mentioned in this paper, unlike the unbounded differentiation operator.

Integral Equation Formulation

Instead of solving for $u(x)$ directly, we constructed an integral equation by solving for

$$\sigma(x) = u''(x) \tag{3}$$

Integrating $\sigma(x)$ twice, we can express the first derivative $u'(x)$ and the function $u(x)$ as:

$$u'(x) = \int_{-1}^{x} \sigma(t)\, dt + C_1 \tag{4}$$

$$u(x) = \int_{-1}^{x} \int_{-1}^{t} \sigma(\tau)\, d\tau\, dt + C_1 x + C_0 \tag{5}$$

where C_1 and C_0 are arbitrary constants of integration that will be determined by the boundary conditions. Using the two equations into the original problem (1), we obtain the full integral equation for $\sigma(x)$:

$$\sigma(x) + \mu \left(\int_{-1}^{x} \sigma(t)\, dt + C_1 \right) + \nu \left(\int_{-1}^{x} \int_{-1}^{t} \sigma(\tau)\, d\tau\, dt + C_1 x + C_0 \right) = f(x) \tag{6}$$

This equation (6) relates the unknown density $\sigma(x)$ and the integration constants C_0, C_1 to the known function $f(x)$. Representing $\sigma(x)$ and $f(x)$ by truncated Chebyshev series:

$$\sigma(x) = \sum_{k=0}^{N}{}' a_k T_k(x), \quad f(x) = \sum_{k=0}^{N}{}' f_k T_k(x), \tag{7}$$

and using the spectral integration formulas in the paper, so we can find that the first integral becomes

$$\int_{-1}^{x} \sigma(t)dt = \sum_{k=1}^{N}{}' d_k T_k(x) + C_1, \tag{8}$$

where

$$d_k = \frac{a_{k-1} - a_{k+1}}{2k}, \quad k \geq 1. \tag{9}$$

If we integrate again, it becomes

$$\int_{-1}^{x} \int_{-1}^{t} \sigma(\tau)d\tau dt = \sum_{k=2}^{N} u_k T_k(x) + \left(C_1 - \frac{d_1}{2} \right) x + C_0, \tag{10}$$

with

$$u_k = \frac{d_{k-1} - d_{k+1}}{2k} = \frac{1}{2k} \left(\frac{a_{k-2} - a_k}{2k-2} - \frac{a_k - a_{k+2}}{2k+2} \right), \quad k \geq 2. \tag{11}$$

Compare the Chebyshev coefficients and then we will get the following linear system:

$$k = 0: \quad a_0 + \mu C_1 + \nu C_0 = f_0, \tag{12}$$

$$k = 1: \quad a_1 + \frac{\mu}{2}(a_0 - a_2) + \frac{\nu}{8}(8C_1 - a_1 + a_3) = f_1, \tag{13}$$

$$k \geq 2: \quad a_k + \frac{\mu}{2k}(a_{k-1} - a_{k+1}) + \frac{\nu}{2k} \left(\frac{a_{k-2} - a_k}{2k-2} - \frac{a_k - a_{k+2}}{2k+2} \right) = f_k. \tag{14}$$

This is a system of $N+1$ equations with $N+3$ unknowns. Two additional equations are obtained from the boundary conditions

$$C_0 - C_1 + \sum_{k=2}^{N} \frac{1}{2k} \left(\frac{1}{2k-2}(a_{k-2} - a_k) - \frac{1}{2k+2}(a_k - a_{k+2}) \right) (-1)^k = \alpha, \tag{15}$$

$$C_0 + C_1 + \sum_{k=2}^{N} \frac{1}{2k} \left(\frac{1}{2k-2}(a_{k-2} - a_k) - \frac{1}{2k+2}(a_k - a_{k+2}) \right) = \beta. \tag{16}$$

Now the $(N+3) \times (N+3)$ system is pentadiagonal except for the two rows derived from the boundary conditions, and can be solved using approximately 10N floating point operations.

3 Simple Implementation

We verify the method by solving a Poisson equation, which is a special case of (1) with $\mu = 0$ and $\nu = 0$:

$$u''(x) = e^x, \quad u(-1) = 0, \quad u(1) = 0 \tag{17}$$

The exact solution is $u(x) = e^x - \frac{e-e^{-1}}{2}x - \frac{e+e^{-1}}{2}$.

Python Code

The following Python implementation uses a pure NumPy approach. Note that special care is taken to multiply the zeroth coefficient (a_0) by 2 before applying the integration recurrence, ensuring consistent definitions with the integral operator.

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

# ------------------------------------------------------------
# 1.The Core Algorithm
# ------------------------------------------------------------

def cheb_nodes(N):
    return np.cos(np.pi * np.arange(N + 1) / N)

def cheb_coeffs_pure_numpy(vals):
    N = len(vals) - 1
    coeffs = np.zeros(N + 1)
    for k in range(N + 1):
        sum_val = 0.0
        for j in range(N + 1):
            weight = 1.0
            if j == 0 or j == N: weight = 0.5
            term = weight * vals[j] * np.cos(k * j * np.pi / N)
            sum_val += term
        coeffs[k] = (2.0 / N) * sum_val
    coeffs[0] /= 2.0
    return coeffs

def spectral_integrate(coeffs):
    N = len(coeffs) - 1
    int_coeffs = np.zeros(N + 1)
    a_pad = np.append(coeffs, [0, 0])

    a_pad[0] *= 2.0

    for k in range(1, N):
        int_coeffs[k] = (1.0 / (2.0 * k)) * (a_pad[k-1] - a_pad[k+1])
    return int_coeffs

def solve_poisson_exp_x(N):
    start_time = time.perf_counter()
    x = cheb_nodes(N)
    f = np.exp(x)

    sigma_coeffs = cheb_coeffs_pure_numpy(f)
    u_prime_coeffs = spectral_integrate(sigma_coeffs)
```

```
    u_coeffs = spectral_integrate(u_prime_coeffs)

    val_at_1 = np.sum(u_coeffs)
    val_at_neg1 = np.sum(u_coeffs * ((-1)**np.arange(N+1)))

    C0 = -0.5 * (val_at_1 + val_at_neg1)
    C1 = -0.5 * (val_at_1 - val_at_neg1)

    u_coeffs[0] += C0
    u_coeffs[1] += C1

    u_computed = np.polynomial.chebyshev.chebval(x, u_coeffs)
    end_time = time.perf_counter()
    return x, u_computed, end_time - start_time

# ------------------------------------------------------------
# 2.Plot
# ------------------------------------------------------------
N_values = [4, 8, 16, 32, 64]
mse_results = []
time_results = []

for N in N_values:
    x_nodes, u_comp, time_sec = solve_poisson_exp_x(N)
    u_exact = np.exp(x_nodes) - (np.exp(1)-np.exp(-1))/2 * x_nodes - (np
   .exp(1)+np.exp(-1))/2
    mse = np.mean((u_comp - u_exact)**2)
    mse_results.append(mse)
    time_results.append(time_sec)

# Picture
fig = plt.figure(figsize=(14, 6))
ax1 = fig.add_subplot(1, 2, 1)
ax1.semilogy(N_values, mse_results, 'o-', linewidth=2, markersize=8,
   label='Spectral Integration MSE')
ax1.set_title("Spectral Integration Convergence ($u''=e^x$)", fontsize
   =14)
ax1.set_xlabel("N", fontsize=12)
ax1.set_ylabel("MSE", fontsize=12)
ax1.grid(True, which="both", ls="--", alpha=0.4)
ax1.set_xticks(N_values)
ax1.legend()

# Table
ax2 = fig.add_subplot(1, 2, 2)
ax2.axis('off')
table_data = [[N, f"{t:.6f}", f"{m:.3e}"] for N, t, m in zip(N_values,
   time_results, mse_results)]
col_labels = ["N", "CPU Time (s)", "Mean Square Error"]
the_table = ax2.table(cellText=table_data, colLabels=col_labels, loc='
   center', cellLoc='center')
the_table.scale(1, 2)
the_table.set_fontsize(14)

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

Listing 1: Solving $u'' = e^x$ using spectral integration

Results

The numerical results for $u'' = e^x$ are presented in Table 1. As expected for a smooth function like e^x, the method exhibits **superalgebraic convergence**. The error drops to machine precision ($\approx 10^{-16}$) at $N = 16$, demonstrating the method's accuracy and stability. There is also a picture can be seen in github.

Table 1: Mean Square Error (MSE) for $u''(x) = e^x$

N (Nodes)	**Max Error**
4	4.654×10^{-5}
8	7.034×10^{-14}
16	7.196×10^{-32}
32	3.847×10^{-32}
64	3.494×10^{-32}

4 Conclusion

The advantage of the spectral integration method proposed in this paper, compared to the stabilized spectral approach, is its numerical stability. Unlike traditional methods where differentiation amplifies noise, this scheme allows for the simultaneous and precise computation of the first (u') and second (u'') derivatives of the solution. These derivatives are often of significant physical interest and are obtained without the instability inherent in spectral differentiation. But there are also some limitations about this method. The efficiency of the method depends heavily on the structure of the resulting linear system:

- **Constant Coefficients:** The system matrix is pentadiagonal and can be solved efficiently in $O(N)$ operations.
- **Variable Coefficients:** The system matrix becomes dense, and a direct solution using Gaussian elimination would require $O(N^3)$ operations, which is computationally prohibitive for large N.

To deal with the variable coefficient problem, two main approaches are considered:

1. **Iterative Methods:** These methods can utilize the Fast Fourier Transform (FFT) to perform matrix-vector multiplications, maintaining an operation count of $O(N \log N)$. However, for problems with complex behavior or poor conditioning, the number of iterations required may become excessively large.
2. **Direct Methods and Future Direction:** The author suggests that for ill-conditioned problems, direct methods maintaining $O(N)$ or $O(N \log N)$ complexity are preferable. The author also figures out that such algorithms have been designed for both second-order differential equations and more general first-order systems by researchers such as Greengard and Rokhlin recently. These fast direct solvers represent the future direction for efficiently handling variable coefficient boundary value problems.