Project 3: Finite Element Method to Solve Partial Differential Equations with Boundary Conditions

Ji, Jingwei 5141209098 No. 10

June 7, 2018

1 The problem

Consider the following one-dimensional elliptic equations with boundary condition：

$$\begin{cases} -\left(p(x)u'(x)\right)' + u(x) = f(x), \quad 0 < x < 1, \\ u(0) = 0, \quad p(1)u'(1) + u(1) = \beta \end{cases} \tag{1.1}$$

where $p(x) = 1 + x^2$.

In this report, I set $u(x) = \sin(x)$. Thus the following are subsequently determined:

$$f(x) = 2\sin x - 2x\cos x + x^2 \sin x \tag{1.2}$$

$$\beta = 2\cos(1) + \sin(1) \tag{1.3}$$

2 The virtual work statement equivalent to 1.1

The space of admissible functions is $V = \{v | v \text{ is sufficiently smooth} on[0,1], v(0) = 0\}$。

It is to find $u \in V$, such that $\forall v \in V$:

$$\int_0^1 \left[-\left(pu'\right)' + u\right] v dx = \int_0^1 f v dx \tag{2.1}$$

After a bit calculus, the above turns to

$$\int_0^1 \left[pu'v' + uv\right] dx + u(1)v(1) = \int_0^1 f v dx + \beta v(1)$$

Define $a(u,v) \triangleq \int_0^1 [pu'v' + uv]\, dx + u(1)v(1)$ and $F(v) \triangleq \int_0^1 f v dx + \beta v(1)$. Now solving 1.1 is equivalent to finding $u \in V$, such that

$$a(u,v) = F(v), \quad \forall v \in V \tag{2.2}$$

3 The construction of finite element method

3.1 A general case

V has infinite degrees of freedom, which is impossible to solve. Consider the discretized space, $U_h = \{v | v \in C[0,1]; v|e_k \in P_1(e_k), 1 \le k \le N\}$, where e_k is a partition, $e_k = [x_{k-1}, x_k]$, and P_1 is polynomials of order one. Denote by $h_k = x_k - x_{k-1}$ the length of each interval, and define $h = \max_{1 \le k \le N} h_k$. Clearly $U_h \subseteq V$.

Denote by

$$\phi_0 = \begin{cases} 1 - \dfrac{1}{h_1}(x - x_0), & x_0 = 0 \le x \le x_1, \\ 0, & \text{else} \end{cases}$$

$$\phi_i = \begin{cases} \dfrac{1}{h_i}(x - x_{i-1}), & x_{i-1} \le x \le x_i, \\ -\dfrac{1}{h_{i+1}}(x - x_{i+1}), & x_i \le x \le x_{i+1} \\ 0, & \text{else} \end{cases}$$

$i = 1, 2, ..., N - 1$.

$$\phi_N = \begin{cases} \dfrac{1}{h_N}(x - x_{N-1}), & x_{N-1} \le x \le x_N = 1 \\ 0, & \text{else} \end{cases}$$

It can be shown that $\phi_i, i = 0, 1, ..., N$ is a set of basis of U_h. Thus $\forall v \in U_h$, $v = \sum_{i=0}^{N} v_i \phi_i$.

To further incorporate into the type I boundary condition, we constrain U_h to $V_h = \{v | v \in C[0, 1]; v|e_k \in P_1(e_k), 1 \le k \le N; v(0) = 0\}$. Clearly it has freedom of N, and $\forall v \in V_h$, $v = \sum_{i=1}^{N} v_i \phi_i$.

The idea of finite element method is to find a $u_h = \sum_{i=1}^{N} u_i \phi_i$ in V_h to approximate the real solution u. And note that 2.2 turns to

$$a(u_h, v) = F(v), \quad \forall v \in V_h \tag{3.1}$$

So it suffices to find $u_h \in V_h$, such that

$$a(u_h, \phi_i) = F(\phi_i), \quad i = 1, 2, ..., N$$

Substituting $u_h = \sum_{i=1}^{N} u_i \phi_i$ into above we have

$$\sum_{i=1}^{N} a(\phi_i, \phi_j) u_i = F(\phi_j), \quad j = 1, 2, ..., N$$

Denote by $A = [a(\phi_i, \phi_j)]_{ij}$, $\overrightarrow{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix}$, $\overrightarrow{F} = \begin{bmatrix} F(\phi_1) \\ F(\phi_2) \\ \vdots \\ F(\phi_N) \end{bmatrix}$, and then we can write

$$A\overrightarrow{u} = \overrightarrow{F} \tag{3.2}$$

3.2 A computable form

Now consider how to express the integration in $a(u, v)$ and $F(v)$ in a computable manner, where $u, v \in V_h$, meaning $u(x) = \sum_{k=1}^{N} u_k \phi_k(x)$, $v(x) = \sum_{k=1}^{N} v_k \phi_k(x)$. Define $N_k(x) = \frac{x_k - x}{h_k}$, $M_k(x) = \frac{x - x_{k-1}}{h_k}$, then $u(x) = u_{k-1} N_k(x) + u_k M_k(x)$, $v(x) = v_{k-1} N_k(x) + v_k M_k(x)$ respectively on e_k.

$$\begin{aligned} a(u, v) &\triangleq \int_0^1 \left[p u' v' + u v \right] dx + u(1) v(1) \\ &= \sum_{k=1}^{N} \left[\int_{e_k} p u' v' + u n d x \right] + u(1) v(1) \\ &\triangleq \sum_{k=1}^{N} a_k(u, v) \end{aligned}$$

where

$$a_k(u,v) = \int_{e_k} pu'v' + uvdx, \quad 1 \le k \le N-1$$

$$a_N(u,v) = \int_{e_N} pu'v' + uvdx + u(1)v(1)$$

For $1 \le k \le N-1$,

$$\begin{aligned}
a_k(u,v) &= \left[\int_{e_k} p(x)N_k'(x)^2 + N_k(x)^2dx\right] u_{k-1}v_{k-1} \\
&+ \left[\int_{e_k} p(x)N_k'(x)M_k'(x) + N_k(x)M_k(x)dx\right] u_{k-1}v_k \\
&+ \left[\int_{e_k} p(x)N_k'(x)M_k'(x) + N_k(x)M_k(x)dx\right] u_k v_{k-1} \\
&+ \left[\int_{e_k} p(x)M_k'(x)^2 + M_k(x)^2dx\right] u_k v_k \\
&= \left[\frac{\frac{2}{3}x_k^3 - \frac{2}{3}x_{k-1}^3 + x_k x_{k-1}^2 - x_k^2 x_{k-1} + x_k - x_{k-1}}{(x_k - x_{k-1})^2}\right] u_{k-1}v_{k-1} \\
&+ \left[\frac{-\frac{1}{6}x_k^3 + \frac{1}{6}x_{k-1}^3 - x_k + x_{k-1} - \frac{1}{2}x_{k-1}x_k^2 + \frac{1}{2}x_k x_{k-1}^2}{(x_k - x_{k-1})^2}\right] u_{k-1}v_k \\
&+ \left[\frac{-\frac{1}{6}x_k^3 + \frac{1}{6}x_{k-1}^3 - x_k + x_{k-1} + \frac{1}{2}x_k x_{k-1}^2 - \frac{1}{2}x_{k-1}x_k^2}{(x_k - x_{k-1})^2}\right] u_k v_{k-1} \\
&+ \left[\frac{\frac{2}{3}x_k^3 - \frac{2}{3}x_{k-1}^3 + x_k - x_{k-1} + x_{k-1}^2 x_k - x_{k-1}x_k^2}{(x_k - x_{k-1})^2}\right] u_k v_k
\end{aligned}$$

For $k = N$,

$$\begin{aligned}
a_N(u,v) &= \left[\frac{\frac{2}{3}x_N^3 - \frac{2}{3}x_{N-1}^3 + x_N x_{N-1}^2 - x_N^2 x_{N-1} + x_N - x_{N-1}}{(x_N - x_{N-1})^2}\right] u_{N-1}v_{N-1} \\
&+ \left[\frac{-\frac{1}{6}x_N^3 + \frac{1}{6}x_{N-1}^3 - x_N + x_{N-1} - \frac{1}{2}x_{N-1}x_N^2 + \frac{1}{2}x_N x_{N-1}^2}{(x_N - x_{N-1})^2}\right] u_{N-1}v_N \\
&+ \left[\frac{-\frac{1}{6}x_N^3 + \frac{1}{6}x_{N-1}^3 - x_N + x_{N-1} + \frac{1}{2}x_N x_{N-1}^2 - \frac{1}{2}x_{N-1}x_N^2}{(x_N - x_{N-1})^2}\right] u_N v_{N-1} \\
&+ \left[\frac{\frac{2}{3}x_N^3 - \frac{2}{3}x_{N-1}^3 + x_N - x_{N-1} + x_{N-1}^2 x_N - x_{N-1}x_N^2}{(x_N - x_{N-1})^2} + 1\right] u_N v_N
\end{aligned}$$

Similarly, for $F(v)$,

$$\begin{aligned}
F(v) &\triangleq \int_0^1 fvdx + \beta v(1) \\
&= \sum_{k=1}^{N} \int_{e_k} fvdx + \beta v(1) \\
&\triangleq \sum_{k=1}^{N} F_k(v)
\end{aligned}$$

For $1 \le k \le N-1$,

$$
\begin{aligned}
F_k(v) &= \left[\int_{x_{k-1}}^{x_k} f(x)N_k(x)dx\right] v_{k-1} + \left[\int_{x_{k-1}}^{x_k} f(x)M_k(x)dx\right] v_k \\
&= \left[\frac{x_k}{x_k - x_{k-1}}(-2\cos x - x^2\cos x)|_{x=x_{k-1}}^{x=x_k} - \frac{1}{x_k - x_{k-1}}(x^2\sin x - x^3\cos x)|_{x=x_{k-1}}^{x=x_k}\right] v_{k-1} \\
&+ \left[-\frac{x_{k-1}}{x_k - x_{k-1}}(-2\cos x - x^2\cos x)|_{x=x_{k-1}}^{x=x_k} + \frac{1}{x_k - x_{k-1}}(x^2\sin x - x^3\cos x)|_{x=x_{k-1}}^{x=x_k}\right] v_k
\end{aligned}
$$

For $k = N$,

$$
\begin{aligned}
F_N(v) &= \int_{x_{N-1}}^{x_N} fvdx + \beta v(1) \\
&= \left[\frac{x_N}{x_N - x_{N-1}}(-2\cos x - x^2\cos x)|_{x=x_{N-1}}^{x=x_N} - \frac{1}{x_N - x_{N-1}}(x^2\sin x - x^3\cos x)|_{x=x_{N-1}}^{x=x_N}\right] v_{N-1} \\
&+ \left[-\frac{x_{N-1}}{x_N - x_{N-1}}(-2\cos x - x^2\cos x)|_{x=x_{N-1}}^{x=x_N} + \frac{1}{x_N - x_{N-1}}(x^2\sin x - x^3\cos x)|_{x=x_{N-1}}^{x=x_N} + \beta\right] v_N
\end{aligned}
$$

The above expressions provide a computable way to specify the matrix A without computing integration numerically. For implementation codes, please see the appendix.

4 Numerical experiments

I implement this specific finite element method using Python. Figure 1 to 4 show the computed solution under different partition size. "Method 1" refers to evenly discretization of the x axis. Table 1 exhibits the relative error [1] under different N. Clearly we find that the finite element method has a nice precision, even when N is rather small. And the max relative error decreases as N increases, except at the case when $N = 10000$.

Table 1: Evenly discretized

N	Max relative error
10	8.93E-05
100	8.93E-07
1000	8.56E-09
10000	2.76E-07

[1] The relative error here refers to $\max_i |\frac{\widetilde{u(x_i)} - u(x_i)}{u(x_i)}|$

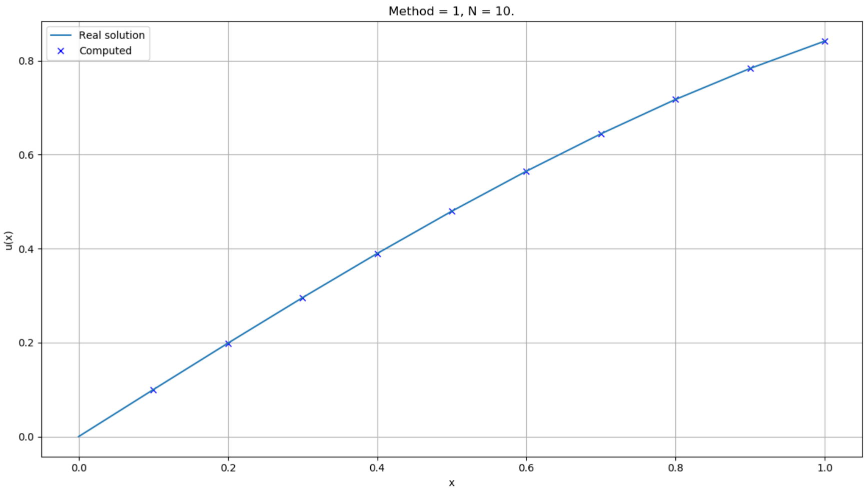

Figure 1: Computed solution

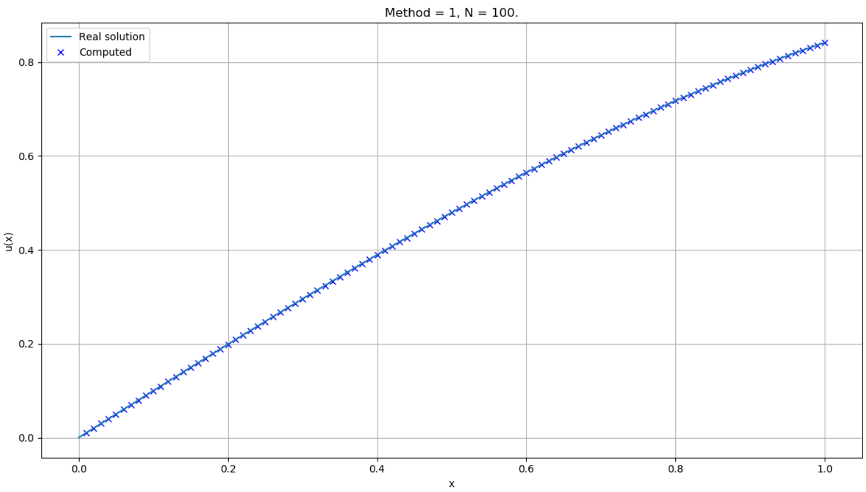

Figure 2: Computed solution

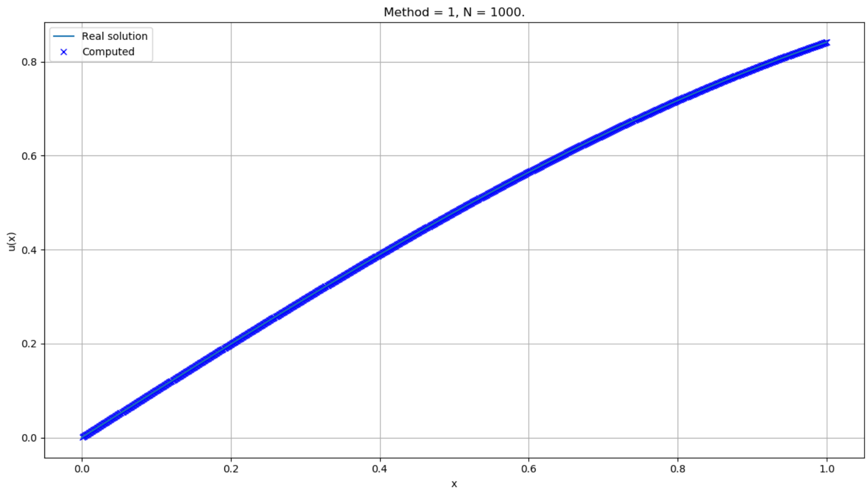

Figure 3: Computed solution

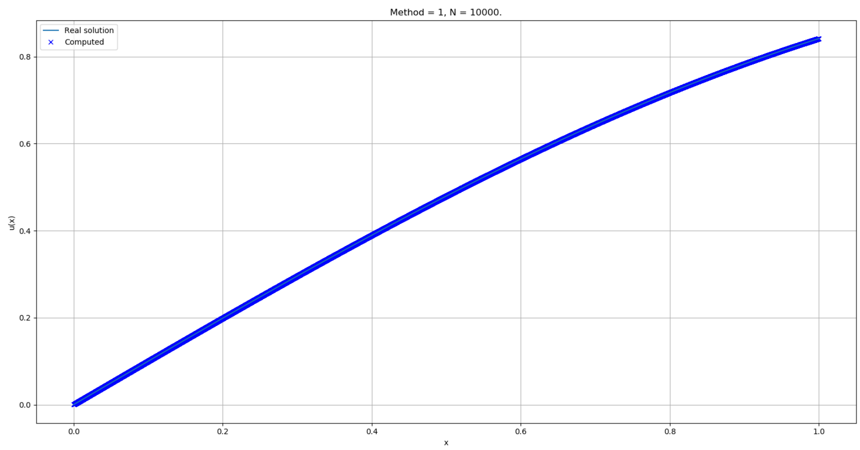

Figure 4: Computed solution

5 Appendix

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

class Solver:
    def __init__(self, N, method):
        self.N = N

        if method == 1:
            self.x_list = []
            self.x_list.append(0)
            stepSize = 1 / self.N
            for i in range(N):
                self.x_list.append(stepSize+self.x_list[-1])

        self.A_matrix = np.zeros((self.N, self.N))
        self.F_vector = np.zeros(self.N)
        self.U_vector = np.zeros(self.N)

        self.realU_vector = np.sin(self.x_list)
        self.beta = 2*np.cos(1) + np.sin(1)
        self.set_A()
        self.set_F()
        self.solve_engine()

    def a(self, i, j):
        sum = 0
        for k in range(1, self.N+1):
            # u_{k-1}: a
            # v_{k-1}: b
            # u_k: c
            # v_k: d
            a = b = c = d = 0
            if k-1 == i:
                a = 1
            if k == i:
                c = 1
            if k-1 == j:
                b = 1
            if k == j:
                d = 1
            # print(i,j,a,b,c,d)

            if a*b != 0:
                sum += ( 2/3*self.x_list[k]**3 - 2/3*self.x_list[k-1]**3 + self.x_list[k] * self.
                    x_list[k-1]**2 - self.x_list[k]**2 * self.x_list[k-1] + self.x_list[k] - self
                    .x_list[k-1] ) / ( (self.x_list[k] - self.x_list[k-1] )**2 )

            if a*d != 0:
                sum += ( -1 * self.x_list[k]**3/6 + self.x_list[k-1]**3 / 6 -self.x_list[k] +self
                    .x_list[k-1] - 0.5*self.x_list[k-1]*self.x_list[k]**2 + 0.5 * self.x_list[k]*
                    self.x_list[k-1]**2 ) / ( (self.x_list[k] - self.x_list[k-1] )**2 )

            if c*b != 0:
                sum += ( -1 * self.x_list[k]**3/6 + self.x_list[k-1]**3 / 6 -self.x_list[k] +self
                    .x_list[k-1] - 0.5*self.x_list[k-1]*self.x_list[k]**2 + 0.5 * self.x_list[k]*
                    self.x_list[k-1]**2 ) / ( (self.x_list[k] - self.x_list[k-1] )**2 )

            if c*d != 0:
                if k == self.N:
                    sum += ( 2/3* self.x_list[k]**3 - 2/3*self.x_list[k-1]**3 +self.x_list[k] -
                        self.x_list[k-1] + self.x_list[k-1]**2 * self.x_list[k] - self.x_list[k-1]
```

```
                    * self.x_list[k]**2 ) / ( (self.x_list[k] - self.x_list[k-1] )**2 ) + 1
            else:
                sum += ( 2/3* self.x_list[k]**3 - 2/3*self.x_list[k-1]**3 +self.x_list[k] -
                    self.x_list[k-1] + self.x_list[k-1]**2 * self.x_list[k] - self.x_list[k-1]
                    * self.x_list[k]**2 ) / ( (self.x_list[k] - self.x_list[k-1] )**2 )

        return sum

    def set_A(self):
        for i in range(1, self.N+1):
            for j in range(1, self.N+1):
                if abs(i-j) >= 2:
                    pass
                else:
                    self.A_matrix[i-1][j-1] = self.a(i, j)

    def auxiliary_a(self, up_x, down_x):
        # a = -2 \cos x - x^2 \cos x
        return -2*np.cos(up_x)-up_x*up_x*np.cos(up_x) - (-2*np.cos(down_x)-down_x*down_x*np.cos(
            down_x))

    def auxiliary_b(self, up_x, down_x):
        # b = x^2 \sin x - x^3 \cos x
        return up_x*up_x*np.sin(up_x)-up_x**3*np.cos(up_x) - (down_x*down_x*np.sin(down_x)-down_x
            **3*np.cos(down_x))

    def set_F(self):
        for i in range(1, self.N+1):
            # a = -2 \cos x - x^2 \cos x
            # b = x^2 \sin x - x^3 \cos x
            sum = 0
            for k in range(1, self.N+1):
                c1 = c2 = 0
                if k-1 == i:
                    c1 = 1 # v_{k-1}
                if k == i:
                    c2 = 1 # v_k
                if c1 == 1:
                    sum += self.x_list[k]/(self.x_list[k]-self.x_list[k-1])*self.auxiliary_a(up_x=
                        self.x_list[k], down_x=self.x_list[k-1]) \
                    - 1/(self.x_list[k]-self.x_list[k-1]) * self.auxiliary_b(up_x=self.x_list[k],
                        down_x=self.x_list[k-1])
                if c2 == 1:
                    if k == self.N:
                        sum += 1*self.beta -self.x_list[k-1]/(self.x_list[k]-self.x_list[k-1])*
                            self.auxiliary_a(up_x=self.x_list[k], down_x=self.x_list[k-1]) \
                        + 1 / (self.x_list[k]-self.x_list[k-1]) * self.auxiliary_b(up_x=self.
                            x_list[k], down_x=self.x_list[k-1])
                    else:
                        sum += -1 * self.x_list[k-1]/(self.x_list[k]-self.x_list[k-1])*self.
                            auxiliary_a(up_x=self.x_list[k], down_x=self.x_list[k-1]) \
                        + 1 / (self.x_list[k]-self.x_list[k-1]) * self.auxiliary_b(up_x=self.
                            x_list[k], down_x=self.x_list[k-1])
            self.F_vector[i-1] = sum

    def solve_engine(self):
        self.U_vector = np.dot(np.linalg.inv(self.A_matrix), self.F_vector)
        fig = plt.figure()
        plt.plot(self.x_list[1:self.N+1], self.U_vector, 'bx')
        plt.plot(self.x_list[0:self.N+1], self.realU_vector)
        plt.xlabel("x")
        #
        # for i in range(self.N):
```

```python
        # line_string = ""
        # for j in range(self.N):
        # line_string += " "+str(self.A_matrix[i][j])+" "
        # print(line_string)
        # print(self.F_vector)

        plt.show()

if __name__ == '__main__':
    N = 10
    method = 1
    s = Solver(N=N, method=method)
```