Project 3: Potential and Efficient Solution Techniques

Zhipeng Yu

1. Introduction to the problem:

Solve the following boundary value problem, with domain (0, L), analytically, the conditions are given like this.

$$\frac{d}{dx}\left(E(x)\frac{du}{dx}\right) = xk^3 cos(\frac{2\pi kx}{L})$$

$$E(x) = 10 \; DIFFERENT \; SEGMENTS \; (SEE \; BELOW)$$

$$k = 12, \; L = 1, \; u(0) = -0.3, \; u(L) = 0.7$$

$$\begin{array}{l} FOR \; 0.0 < x < 0.1 \; E = 2.5 \\ FOR \; 0.1 < x < 0.2 \; E = 1.0 \\ FOR \; 0.2 < x < 0.3 \; E = 1.75 \\ FOR \; 0.3 < x < 0.4 \; E = 1.25 \\ FOR \; 0.4 < x < 0.5 \; E = 2.75 \\ FOR \; 0.5 < x < 0.6 \; E = 3.75 \\ FOR \; 0.6 < x < 0.7 \; E = 2.25 \\ FOR \; 0.7 < x < 0.8 \; E = 0.75 \\ FOR \; 0.8 < x < 0.9 \; E = 2.0 \\ FOR \; 0.9 < x < 1.0 \; E = 1.0 \end{array}$$

What am I going to do is to get a solution of "u" without direct integral.
How am I going to do is to combine some linear simple functions and add them up such that getting as closed as possible to the true solution.
For example: function like "(x-1)x/2,(1+x)(1-x),x(x+1)/2".
Although these functions seem simple, they are really powerful if you get fairly large number of them.

2. Objective

Goals: Solve this with the finite element method using linear equal-sized elements. Use 100, 1000 and 10000 elements. To write a Preconditioned Conjugate-Gradient solver. Use the diagonal preconditioning given in the notes. The data storage is to be element by element (symmetric) and the matrix vector multiplication is to be done element by element. And bellow, it is how to calculate the error.

$$e^N \overset{\text{def}}{=} \frac{||u-u^N||_{E(\Omega)}}{||u||_{E(\Omega)}},$$

$$||u||_{E(\Omega)} \overset{\text{def}}{=} \sqrt{\int_\Omega \frac{du}{dx} E \frac{du}{dx}\, dx},$$

3. My procedure

First we decide to use the Weak Formulation to solve it.

$$\text{Find } u \in H^1(\Omega)\ u|_{\Gamma_u} = d \text{ such that } \forall \nu \in H^1(\Omega), \nu|_{\Gamma_u} = 0$$

$$\int_\Omega \frac{d\nu}{dx} E \frac{du}{dx}\, dx = \int_\Omega f\nu\, dx + t\nu|_{\Gamma_t}.|$$

We approximate “u” by:

$$u^h(x) = \sum_{j=1}^{N} a_j \phi_j(x).$$

If we choose “v” with the same approximation functions, but a different linear combination, we get “v” like this:

$$\nu^h(x) = \sum_{i=1}^{N} b_i \phi_i(x),$$

Since the “v” are arbitrary (formulation definition), the “bi” are arbitrary, therefore

$$\sum_{i=1}^{N} b_i \left(\sum_{j=1}^{N} K_{ij} a_j - R_i \right) = 0 \Rightarrow [K]\{a\} = \{R\},$$

$$K_{ij} \overset{\text{def}}{=} \int_\Omega \frac{d\phi_i}{dx} E \frac{d\phi_j}{dx}\, dx \text{ and}$$

$$R_i \overset{\text{def}}{=} \int_\Omega \phi_i f\, dx + \phi_i t|_{\Gamma_t},$$

According to that, we will use some mathematical trick to simplify the integral calculation such as Gaussian quadrature (ϕi represents the simple

"function" that we build by ourselves). Then we will get [K] and {R} and add constrains to them. This time I will use PCG method to get solution from [K] and [R].

STEP 1 : FOR i = 1 :$SELECT$ $\{a\}^1 \Rightarrow \{r\}^1 = \{R\} - [K]\{a\}^1 = \{z\}^1$

STEP 2 : COMPUTE (WITH $\{z\}^1 = \{r\}^1$)

$$\lambda^1 = \frac{\{z\}^{T,1}(\{R\}-[K]\{a\}^1)}{\{z\}^{T,1}[K]\{z\}^1} = \frac{\{z\}^{T,1}\{r\}^1}{\{z\}^{T,1}[K]\{z\}^1}$$

STEP 3 : COMPUTE $\{a\}^2 = \{a\}^1 + \lambda^1\{z\}^1$

STEP 4 : (FOR i > 1) COMPUTE $\{r\}^i = \{R\} - [K]\{a\}^i$

$$\theta^i = -\frac{\{r\}^{T,i}[K]\{z\}^{i-1}}{\{z\}^{T,i-1}[K]\{z\}^{T,i-1}} \qquad \{z\}^i \stackrel{\text{def}}{=} \{r\}^i + \theta^i\{z\}^{i-1}$$

$$\lambda^i = \frac{\{z\}^{T,i}(\{R\}-[K]\{a\}^i)}{\{z\}^{T,i}[K]\{z\}^i} = \frac{\{z\}^{T,i}\{r\}^i}{\{z\}^{T,i}[K]\{z\}^i}$$

COMPUTE $\{a\}^{i+1} = \{a\}^i + \lambda^i\{z\}^i$

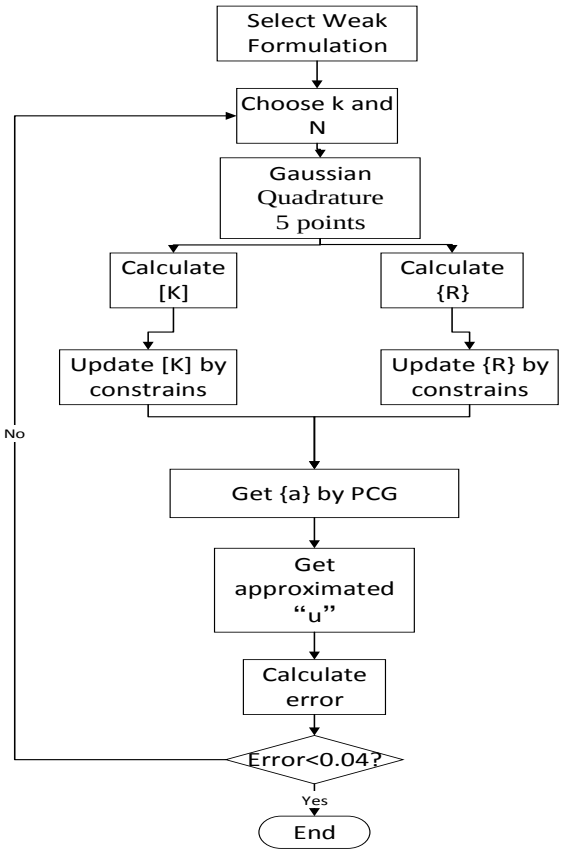
STEP 5 : COMPUTE $e^i \stackrel{\text{def}}{=} \frac{||\{a\}^{i+1}-\{a\}^i||_K}{||\{a\}^i||_K} = \frac{|\lambda^i|||\{z\}^i||_K}{||\{a\}^1||_K} \leq \tau \ \ (\tau = \text{TOL})$

IF $e^i < \tau \Rightarrow$ STOP

IF $e^i \geq \tau \Rightarrow$ GO TO STEP 4 AND REPEAT.

Particularly, when I calculate $\{z\}^T[K]\{z\}$,$\{r\}^T[K]\{z\}$,$\{a^{i+1}\text{-}a^i\}^T[K]\{a^{i+1}\text{-}a^i\}$, $\{a^i\}[K]\{a^i\}$, I will calculate element by element because there are many zeros in [K] which will waste the time.

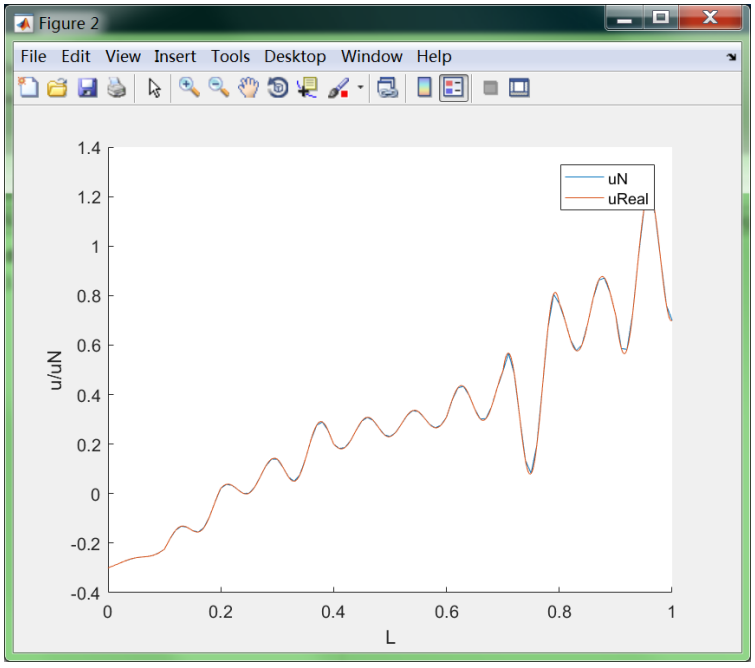
Once we get {a}, we get approximated "u". Based on that, we can do further test like errors and potential energy.



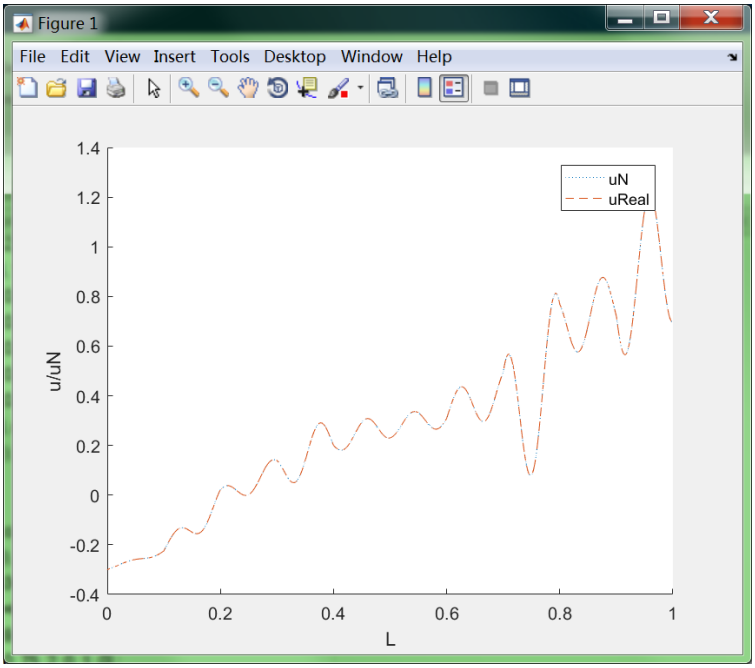
4. Findings

1) Solutions for N=100, N=1000, 10000:

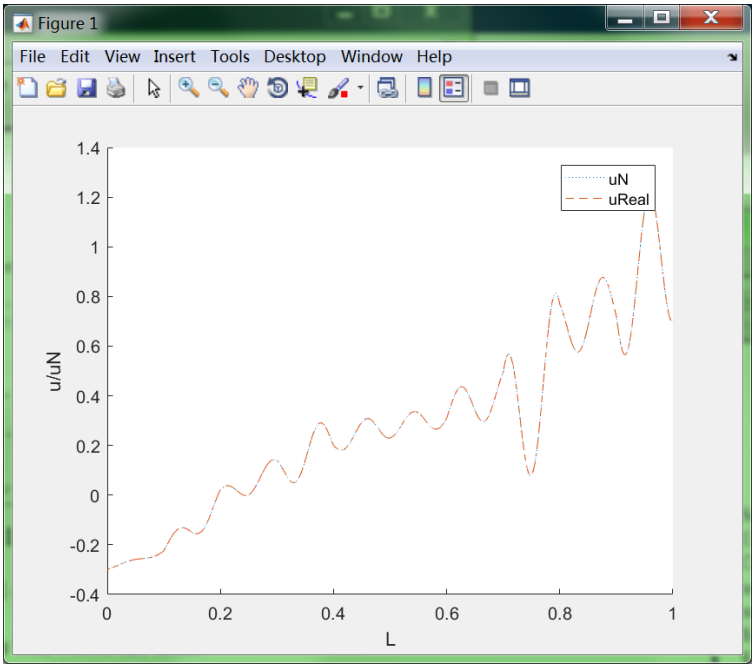
N=100



N=1000



N=10000



2) Errors when N=100;N=1000;N=10000.

N	100	1000	10000
E	0.2196	0.0222	0.0027

3) Energy when N=100;N=1000;N=10000.

N	100	1000	10000

J	-29.0266	-30.5994	-30.6153

4) Iteration when N=100; N=1000; N=10000.

N	100(t1=0.000001)	1000(t2=t1/10)	10000(t3=t2/10)
Iteration	101	1001	10001

5. Observations and discussion

According to tables and runtime,

a) I find when N increases, the errors decreases significantly and potential energy decreases slightly. And actually 1000 points is good enough for this problem. N=10000 will take 100 times run-time to solve it.

b) Just as mentioned above, the time complexity of solving this problem should be $O(N^2)$, because when N=1000, it takes about 15 seconds and when N=10000, it takes about 30 minutes, nearly 100 times compared with N 10 times bigger.

c) And as conjugate gradient decent method is aiming to make every update of {a} not affect previous update. So ideally iteration times should be dimension N. And my results fit this well.

d) It is happy to find that P-CG is faster than normal gradient decent but it is also upset to find that matlab always does things better than you. Using matlab original commands only takes 10 seconds to solve it.(like sparse)

e) So I wonder what is the meaning of P-CG methods and why for FEA

problems people want to use gradient decent method. Why people don’t directly use closed-form like {a}={R}/{K} as {K} is positive defined. For close form, it must have one local minimum (also is global minimum).

f) So over-fitting is the only disadvantage of closed-form that I could imagine regardless of local or global minimum. But actually I don’t think over-fitting is a bad thing for this problem.

6. Appendix

Weak formulation:

```
function output=weakform(N)
Gaussian=[
0.000,0.568;
0.538,0.478;
0.906,0.236;
-0.538,0.478;
-0.906,0.237];
N=10000;
L=1;
k=12;
he=L/N;
J=he/2;
%f=@(x)-1*k^2*sin(2*pi*k*x/L);
%u=@(x)(-1*L/(4*E*pi^2))*sin(2*pi*k*x/L)+L*x;
%du=@(x)(-1*L*k/(2*E*pi))*cos(2*pi*k*x/L)+L;
%Xkes=@(x,i)J*x+(2*i-1)*L/N;
%GaussianF=@(x)x;
Ke=zeros(2,2,N);
K=zeros(N+1,N+1);
Re=zeros(2,N);
a=zeros(N+1,1);
for i=1:N
    Ke(:,:,i)=[E1((i-1)*he)/(J*2),-1*E1((i-1)*he)/(J*2);
        -1*E1((i-1)*he)/(J*2),E1((i-1)*he)/(J*2)];
%       cc=(Ke(1,1,i)*Ke(2,2,i))^0.5;
%       Ke(:,:,i)=[1,Ke(1,2,i)/(cc);
%           Ke(2,1,i)/(cc),1];
    Re(1,i)=J*Gaussian(1,2)*thetahat1(Gaussian(1,1))*f(Xkes(Gaussian(1,1),i,J,L,N),k,L);
    Re(1,i)=Re(1,i)+J*Gaussian(2,2)*thetahat1(Gaussian(2,1))*f(Xkes(Gaussian(2,1),i,J,L,N),k,L);
    Re(1,i)=Re(1,i)+J*Gaussian(3,2)*thetahat1(Gaussian(3,1))*f(Xkes(Gaussian(3,1),i,J,L,N),k,L);
    Re(1,i)=Re(1,i)+J*Gaussian(4,2)*thetahat1(Gaussian(4,1))*f(Xkes(Gaussian(4,1),i,J,L,N),k,L);
    Re(1,i)=Re(1,i)+J*Gaussian(5,2)*thetahat1(Gaussian(5,1))*f(Xkes(Gaussian(5,1),i,J,L,N),k,L);
    Re(2,i)=J*Gaussian(1,2)*thetahat2(Gaussian(1,1))*f(Xkes(Gaussian(1,1),i,J,L,N),k,L);
    Re(2,i)=Re(2,i)+J*Gaussian(2,2)*thetahat2(Gaussian(2,1))*f(Xkes(Gaussian(2,1),i,J,L,N),k,L);
    Re(2,i)=Re(2,i)+J*Gaussian(3,2)*thetahat2(Gaussian(3,1))*f(Xkes(Gaussian(3,1),i,J,L,N),k,L);
    Re(2,i)=Re(2,i)+J*Gaussian(4,2)*thetahat2(Gaussian(4,1))*f(Xkes(Gaussian(4,1),i,J,L,N),k,L);
    Re(2,i)=Re(2,i)+J*Gaussian(5,2)*thetahat2(Gaussian(5,1))*f(Xkes(Gaussian(5,1),i,J,L,N),k,L);
end
for i=1:N+1
```

```
    if i==1
        K(i,1)=Ke(1,1,i);
        K(i,2)=Ke(1,2,i);
        R(i)=Re(1,i);
        continue
    end
    if(i==N+1)
        K(i,N)=Ke(2,1,i-1);
        K(i,N+1)=Ke(2,2,i-1);
        R(i)=Re(2,i-1);
        continue
    else
        K(i,i-1)=Ke(2,1,i-1);
        K(i,i)=Ke(2,2,i-1)+Ke(1,1,i);
        K(i,i+1)=Ke(1,2,i);
        R(i)=Re(1,i)+Re(2,i-1);
    end
end
Kc=K(2:N,2:N);
Rc=R(2:N)';
Rc(N-1)=Rc(N-1)-0.7*K(N,N+1);
Rc(1)=Rc(1)+0.3*K(2,1);
T=zeros(N-1,N-1);
KcT=zeros(N-1,N-1);
for i=1:N-1
    T(i,i)=1/sqrt(Kc(i,i));
    KcT(i,i)=1;
    if(i==N-1)
        KcT(i,i-1)=Kc(i,i-1)/(Kc(i,i)*Kc(i-1,i-1))^0.5;
        continue;
    end
    if(i==1)
        KcT(i,i+1)=Kc(i,i+1)/(Kc(i,i)*Kc(i+1,i+1))^0.5;
        continue;
    end
    KcT(i,i-1)=Kc(i,i-1)/(Kc(i,i)*Kc(i-1,i-1))^0.5;
    KcT(i,i+1)=Kc(i,i+1)/(Kc(i,i)*Kc(i+1,i+1))^0.5;
end
KeT=zeros(2,2,N-2);
for i=1:N-2
    KeT(:,:,i)=KcT(i:i+1,i:i+1);
    KeT(2,2,i)=0;
    if(i==N-2)
        KeT(2,2,i)=1;
```

```
    end
end
RcT=T'*Rc;
KcT=sparse(KcT);
a=zeros(N-1,1);
j=1;
rT=zeros(N-1,1);
zT=rT;
e=1;
while(e>0.000001)
    apre=a;
    RR=zeros(N-1,1);
    for i=1:N-2
        RR(i:i+1)=RR(i:i+1)+KeT(:,:,i)*a(i:i+1);
    end
    rT=RcT-RR;
    if(j==1)
        zT=rT;
        zKz=0;
        for k=1:N-2
            zKz=zKz+zT(k:k+1)'*KeT(:,:,k)*zT(k:k+1);
        end
        lamda=zT'*rT/zKz;
        a=a+lamda*zT;
    end
    if(j>1)
        zKz=0;
        for k=1:N-2
            zKz=zKz+zT(k:k+1)'*KeT(:,:,k)*zT(k:k+1);
        end
        rKz=0;
        for k=1:N-2
            rKz=rKz+rT(k:k+1)'*KeT(:,:,k)*zT(k:k+1);
        end
        theta=-rKz/zKz;
        zT=rT+theta*zT;
        zKz=0;
        for k=1:N-2
            zKz=zKz+zT(k:k+1)'*KeT(:,:,k)*zT(k:k+1);
        end
        lamda=zT'*rT/zKz;
        a=a+lamda*zT;
    end
    if(j==1000)
```

```
            aeKae=0;
            for k=1:N-2
                aeKae=aeKae+(a(k:k+1)-apre(k:k+1))'*KeT(:,:,k)*(a(k:k+1)-apre(k:k+1));
            end
            ppKpp=0;
            for k=1:N-2
                ppKpp=ppKpp+apre(k:k+1)'*KeT(:,:,k)*apre(k:k+1);
            end
            e=aeKae/ppKpp;
        end
        j=j+1;
    end
```