how U doin'?

$$🐊^{🐊^{🐊}} = \int_{🎃}^{🎅} 🙊 \, d🍀$$

BVP for ODEs*

10170437 Mark Taylor

May 31, 2020

Contents

1 Problems **2**

2 Theory **3**

3 Solutions **3**

4 Code **4**

4.1 main function 4

4.2 PLRR 4

4.3 PLRR_intpol 6

4.4 Gaussquad 6

4.5 GauEli 7

5 Appendix **8**

1 Problems

问题：

$$
\begin{aligned}
-\frac{\mathrm{d}}{\mathrm{d}x}(p(x)\frac{\mathrm{d}u}{\mathrm{d}x})+q(x)u &= f(x), \quad x\in(a,b), & (1)\\
u(a) &= 0, \quad u'(b)=0. & (2)
\end{aligned}
$$

其中 $p(x)\in C^1(\bar{I}), p(x)\geq p_{\min}>0, q\in C(\bar{I}), q(x)\geq 0, f\in H^0(I)$, $\bar{I}=[a,b]$. 等式两边乘以 v 在 I 上积分，并运用分部积分可得

$$
a(u,v) = (f,v),
$$

其中，

$$
a(u,v)=\int_a^b\left(p\frac{\mathrm{d}u}{\mathrm{d}x}\frac{\mathrm{d}v}{\mathrm{d}x}+q\,u\,v\right)\,\mathrm{d}x, \quad (f,v)=\int_a^b f\,v\,\mathrm{d}x.
$$

对两点边值问题 (1)-(2)，在等距网格上（区间个数为 N 个），给出其相应的一次有限元程序。
要求：

1. 要求网格点个数 N 及区间 $[a,b]$ 在程序中可以变化；
2. 要求程序中可以很容易更换函数 p,q 和 f；
3. 先运行 $[0,1]$ 之间，$p=1, q=0, f=\pi^2 sin(\pi x), N=10$ 的情形，输出所有结点的值；

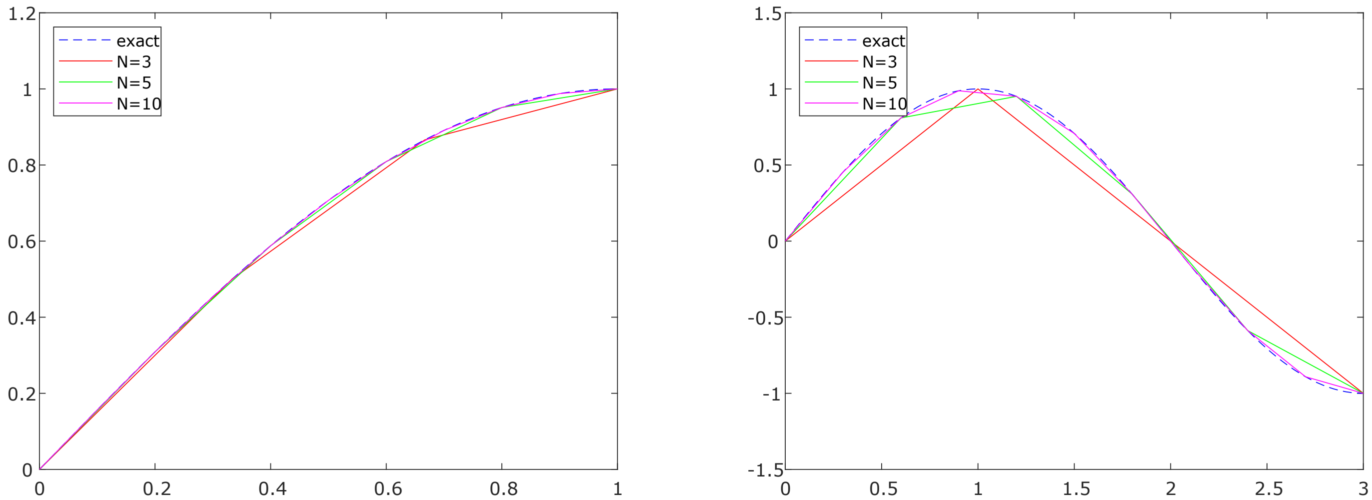
*This article was typeset by Mark Taylor using the LaTeX document processing system.

2 Theory

See reference.mlx in the *src* folder.

3 Solutions

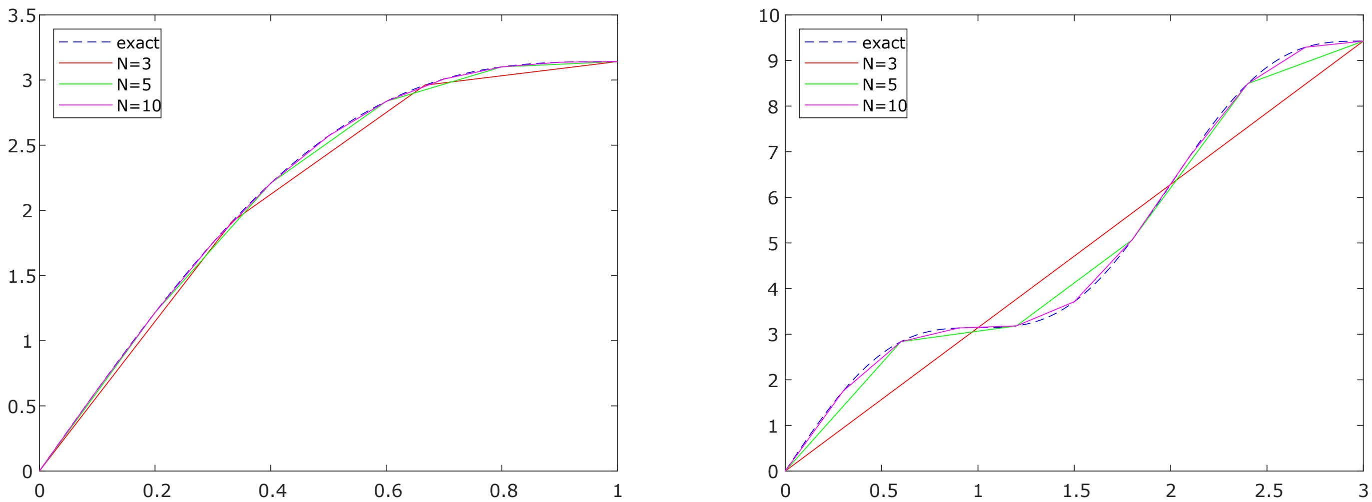
See graphs of those two test problems as follows. View code here



(a) Fitting for problem 1 on $[0, 1]$

(b) Fitting for problem 1 on $[0, 3]$

Figure 1: BVP Test 1



(a) Fitting for problem 2 on $[0, 1]$

(b) Fitting for problem 2 on $[0, 3]$

Figure 2: BVP Test 2

4 Code

4.1 main function

```
% ============================================================
% Solve a linear two-point boundary-value problem
%                   -(p(x)u'(x))'+q(x)u(x)=f(x),    a<=x<=b,
% with initial conditions
%                           u(a)=0, u'(b)=0.
% ============================================================

% ************************************************************
% Test 1
% f=@(x)pi^2/4*sin(pi/2*x);
% u=@(x)sin(pi/2*x);

% Test 2
f=@(x)pi^2*sin(pi*x);
u=@(x)sin(pi*x)+pi*x;
% ************************************************************

p=@(x)x.^0; % p(x)=1
q=@(x)x.*0; % p(x)=0
a=0;
% b=1;
b=3;
N=[3,5,10];

% Since the dimensions are different, we can't
% simply use a matrix for storage. List?
uc1=PLRR(f,p,q,a,b,N(1));
uc2=PLRR(f,p,q,a,b,N(2));
uc3=PLRR(f,p,q,a,b,N(3));

% output coef vector of approx solution in command window
disp(uc3)

% plot graphs of exact solution & approximate solution
t1=linspace(a,b,N(1)+1);
t2=linspace(a,b,N(2)+1);
t3=linspace(a,b,N(3)+1);
tt=linspace(a,b,100+1);

% approx solutions
u1=PLRR_intpol(uc1,a,b);
u2=PLRR_intpol(uc2,a,b);
u3=PLRR_intpol(uc3,a,b);

figure
plot(tt,u(tt),'--b',t1,u1,'r',...
    t2,u2,'g',t3,u3,'m')
legend('exact',sprintf('N=%d',N(1)),sprintf('N=%d',N(2))...
    ,sprintf('N=%d',N(3)),'Location','northwest')
```

Listing 1: Main function for solving BVP of ODEs.

4.2 PLRR

This is the exact method we used for solving this Boundary Value Problems (BVP) of ODEs.

```
% Piecewise Linear Rayleigh-Ritz method
% ============================================================
% Solve a linear two-point boundary-value problem
%                   -(p(x)u'(x))'+q(x)u(x)=f(x),    a<=x<=b,
% with initial conditions
%                           u(a)=0, u'(b)=0.
% ============================================================
```

```
% See the theory part at <reference.mlx>
function uc=PLRR(f,p,q,a,b,N)
% INPUT:
%   f: f(x)
%   p: p(x); optional, by default p(x)=1
%   q: q(x); optional, by default q(x)=0
%   a, b: interval [a,b]; optional, by default [a,b]=[0,1]
%   N: # of evenly spaced intervals; optional, by default N=10
% OUTPUT:
%   uc: coef vector of approx solution

% ==============================================================
% *****  NOTE that f, p, q must be element-wise functions  *****
% e.g. f(x)=@(x)x.^2+1 (NOT x^2+1)
% ==============================================================

% set default args
if nargin<6
    N=10;
    if nargin<5
        b=1;
        if nargin<4
            a=0;
            if nargin<3
                q=@(x)x.*0;
                if nargin<2
                    p=@(x)x.^0;
                    if nargin<1
                        error('Error! Function f is not set.')
                    end
                end
            end
        end
    end
end
if b<a
    tmp=a; a=b; b=tmp;
end

h=(b-a)/N;  % step length. no check for N, hhh
A=zeros(N); % coef matrix, symmetric tridiagonal
c=zeros(N,1); % right-side vector
xx=linspace(a,b,N+1);

% determine tridiagonal elements of A, and right-side vector c
for i=1:N-1
    % Or use MATLAB func <integral> instead of <Gaussquad>.
    % --> <Ctrl + F> --> Replace All. Then go to PLRR_test
    % script file & run it again, it shall get same graph.
    A(i,i)=Gaussquad(p,xx(i),xx(i+2))+...
        Gaussquad(@(x)(x-xx(i)).^2.*q(x),xx(i),xx(i+1))+...
        Gaussquad(@(x)(xx(i+2)-x).^2.*q(x),xx(i),xx(i+1));
    A(i,i+1)=-Gaussquad(p,xx(i+1),xx(i+2))+...
        Gaussquad(@(x)(x-xx(i+1)).*(xx(i+2)-x).*q(x),xx(i+1),xx(i+2));

    c(i)=Gaussquad(@(x)(x-xx(i)).*f(x),xx(i),xx(i+1))+...
        Gaussquad(@(x)(xx(i+2)-x).*f(x),xx(i+1),xx(i+2));
end
A(N,N)=Gaussquad(p,xx(N),xx(N+1))+...
       Gaussquad(@(x)(x-xx(N)).^2.*q(x),xx(N),xx(N+1));
c(N)=Gaussquad(@(x)(x-xx(N)).*f(x),xx(N),xx(N+1));

A=A+diag(diag(A,1),-1); % let A_{i,i-1}=A_{i-1,i}
A=A/h^2;
c=c/h;

% uc=A\c; % coef vector of approx solution
```

```
% or use user-defined function <GauEli>
uc=GauEli(A,c);

end
```

Listing 2: Piecewise Linear Rayleigh-Ritz method

4.3 PLRR_intpol

This is the function used for interpolation after obtaining those coefficients.

```
% % Interpolation for PLRR
% function u=PLRR_intpol(uc,xq,a,b)
% N=length(uc);   % No. of coefs of approx solution
% m=length(xq);   % No. of query points
% u=zeros(m,1);   % approximations at these n query points
% xx=linspace(a,b,N+1);
% h=(b-a)/N;
% for i=1:N
%     for j=1:m
%         if xq(j)>=xx(i) && xq(j)<=xx(i+1)
%             if i==1
%                 u(j)=uc(1)*(xq(j)-xx(1))/h;
%                 continue
%             end
%             u(j)=uc(i-1)*(-(xq(j)-xx(i+1))/h)+uc(i)*(xq(j)-xx(i))/h;
%         end
%     end
% end
% end

% Interpolation for PLRR only at those N+1 evenly spaced points
function u=PLRR_intpol(uc,a,b)
N=length(uc);    % No. of coefs of approx solution
u=zeros(N+1,1);  % approximations at these N+1 evenly spaced points
xx=linspace(a,b,N+1);
h=(b-a)/N;
% u(0)=0;
for i=2:N
    u(i)=uc(i-1)*(-(xx(i)-xx(i+1))/h);
end
u(N+1)=uc(i)*(xx(N+1)-xx(N))/h;

end
```

Listing 3: Interpolation for PLRR

4.4 Gaussquad

This function is a use-defined function that servers the purpose of numerical integration, which can be simply identically substituted by MATLAB build-in function *integral*.

```
% Gaussian quadrature
function T = Gaussquad(f,a,b,n)
if nargin<4
    n=8;
end
% Gaussian quarature points & weights on [-1,1]
if n == 2
    w = [1,1];                          % weights
    p = [-1/sqrt(3),1/sqrt(3)];         % points
elseif n == 4
    w = [0.3478548451,0.3478548451,0.6521451549,0.6521451549];
    p = [0.8611363116,-0.8611363116,0.3399810436,-0.3399810436];
elseif n == 8
```

```
    w = [0.1012285363,0.1012285363,0.2223810345,0.2223810345,0.3137066459,...
           0.3137066459,0.3626837834,0.3626837834];
    p = [0.9602898565,-0.9602898565,0.7966664774,-0.7966664774,0.5255324099,...
      -0.5255324099,0.1834346425,-0.1834346425];
else
    error('n must be either 2 or 4 or 8')
end

% take a linear transform: x = t*(b-a)/2 + (a+b)/2, where t belongs to [-1,1]
% int(f,[a,b]) = (b-a)/2 * int(f(t*(b-a)/2 + (a+b)/2), [-1,1])
w = 0.5*(b-a)*w;
p = 0.5*(b-a)*p+0.5*(a+b);

T = w*f(p).';
end
```

Listing 4: Gaussian quadrature

4.5 GauEli

```
% Gaussian Elimination: solve linear systems
% of algebraic equations of the form Ax=b.
% 10170437 Mark Taylor

function [x,U] = GauEli(A, b)

[m,n]=size(A);
if m ~= n
    error('A must be a square matrix!')
end

U=[A,b];          % Augemented matrix of A

for j=1:n
    k=maxIndex(U(:,j),j,n);
    if abs(U(k,j))>eps
        if k~=j
            temp=U(j,j:n+1);
            U(j,j:n+1)=U(k,j:n+1);
            U(k,j:n+1)=temp;
        end
    else
        x='The system has infinitely many solutions!';
        U=NaN;
        return;
    end

    % Perform Gauss elimination.
    for i=j+1:n
        if abs(U(i,j))>eps
            t=U(i,j)/U(j,j);
            U(i,j:n+1)=U(i,j:n+1)-t*U(j,j:n+1);
        end
    end
end

x=zeros(n,1);
% Sovle upper diagonal linear system, i.e. U(1:n,1:n)x=U(:,n+1).
x(n)=U(n,n+1)/U(n,n);
for i=n-1:-1:1
    x(i)=(U(i,n+1)-U(i,i+1:n)*x(i+1:n))/U(i,i);
end

end
```

Listing 5: Gaussian Elimination

5 Appendix

Hello from the Beatles.😃