



how U doin'?

$$\text{🐳🐳🐳} = \int \text{👶} \text{🐏} d\text{🍀}$$

FEM for TPBVP *

10170437 Mark Taylor

December 18, 2020

Contents

1	Problems	3
2	Theory	3
3	Solutions	3
4	Code	5
4.1	Scripts (Main Functions)	5
4.2	Error Estimates	6
4.3	PLRR and PQRR	9
4.4	PLRR_test and PQRR_test	13
4.5	Gaussquad	14
4.6	GauEli	15
	References	16
	Appendices	16

*This article was typeset by Mark Taylor using the L^AT_EX document processing system.

1 Problems

问题:

$$-\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u = f(x), \quad x \in (a, b), \quad (1)$$

$$u(a) = 0, \quad u'(b) = 0. \quad (2)$$

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,

$$a(u, v) = \int_a^b \left(p \frac{du}{dx} \frac{dv}{dx} + q u v \right) dx, \quad (f, v) = \int_a^b f v dx.$$

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;

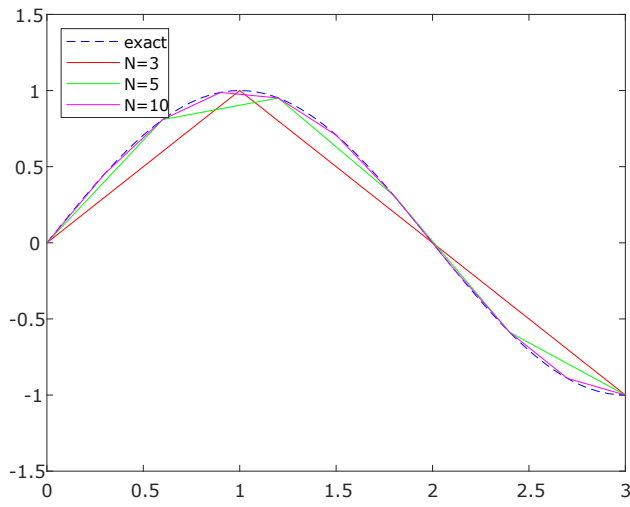
2 Theory

See [reference.mlx](#) [1] and [error.analysis.mlx](#) [3] in the *src* folder. Our text [2] also has a very detailed analysis on FEM error estimates.

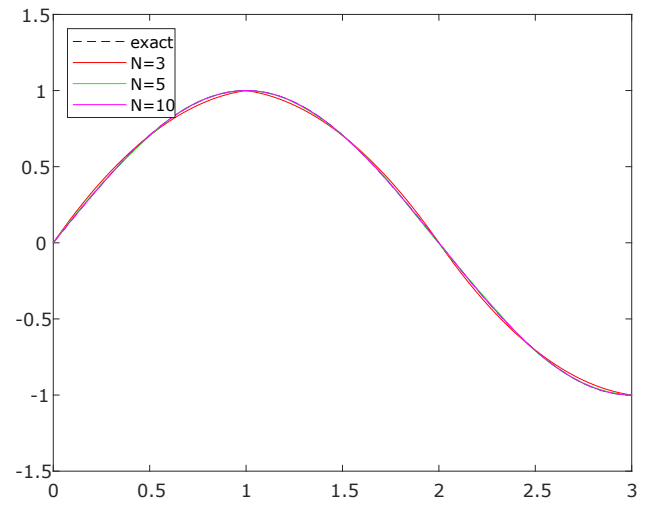
3 Solutions



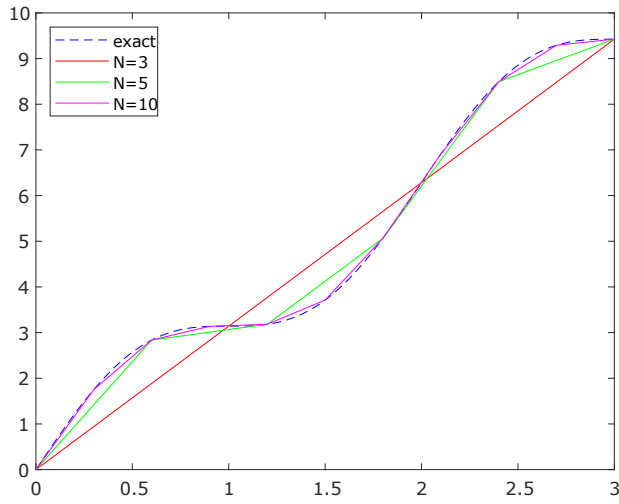
See graphs of those two test problems as follows. View code [here](#) .



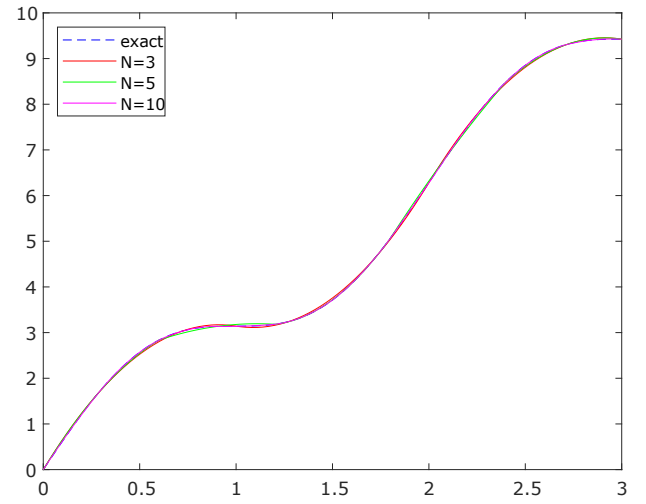
(a) Problem 1 by PLRR



(b) Problem 1 by PQRR



(c) Problem 2 by PLRR



(d) Problem 2 by PQRR

Figure 1: Fittings via PLRR & PQRR

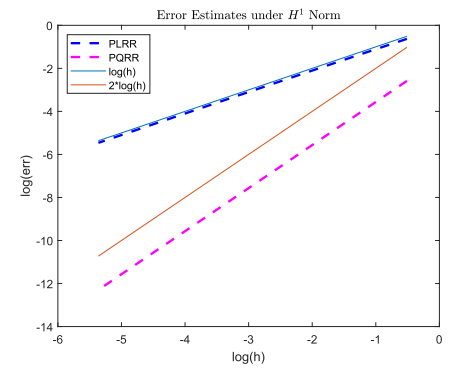
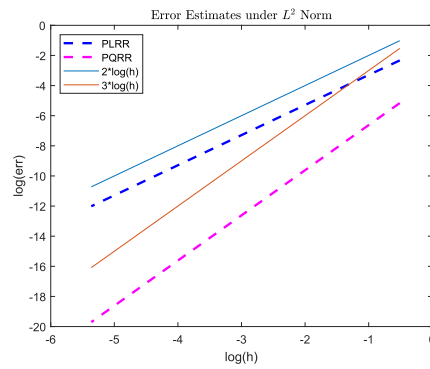
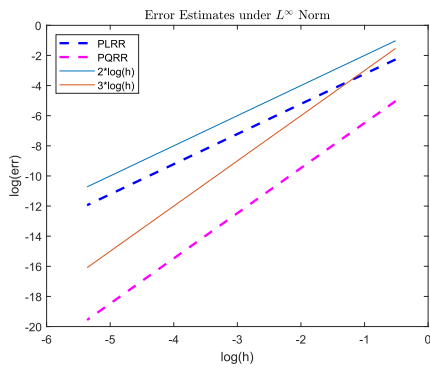


Figure 2: 1-d FEM error estimates under different norms

4 Code

4.1 Scripts (Main Functions)

```

1 % =====
2 % Solve a linear two-point boundary-value problem
3 %            $-(p(x)u'(x))' + q(x)u(x) = f(x)$ ,       $a \leq x \leq b$ ,
4 % with initial conditions
5 %            $u(a)=0, u'(b)=0$ .
6 % =====
7
8 % *****
9 % Test 1
10 isTest1 = true;
11 f=@(x)pi^2/4*sin(pi/2*x);
12 u=@(x)sin(pi/2*x);
13
14 % Test 2
15 % isTest1 = false;
16 % f=@(x)pi^2*sin(pi*x);
17 % u=@(x)sin(pi*x)+pi*x;
18 % *****
19
20 p=@(x)x.^0; % p(x)=1
21 q=@(x)x.*0; % q(x)=0
22 a=0;
23 b=3;
24 % b=1;
25 N=[3,5,10];
26 % N=2*[3,5,10]; % compare with quadratic element
27
28 % Since the dimensions are different, we can't
29 % simply use a matrix for storage. List?
30 uc1=PLRR(f,p,q,a,b,N(1));
31 uc2=PLRR(f,p,q,a,b,N(2));
32 uc3=PLRR(f,p,q,a,b,N(3));
33
34 % output coef vector of approx solution in command window
35 disp(uc3)
36
37 % plot graphs of exact solution & approximate solutions
38 t1=linspace(a,b,N(1)+1);
39 t2=linspace(a,b,N(2)+1);
40 t3=linspace(a,b,N(3)+1);
41 tt=linspace(a,b,100+1);
42
43 % approx solutions
44 u1=PLRR_intpol(t1,[0;uc1],tt); % Note that these interpolating points are also on
45 u2=PLRR_intpol(t2,[0;uc2],tt); % those line segments, which means we can actually
46 u3=PLRR_intpol(t3,[0;uc3],tt); % skip this phase, and replace with [0;uc] instead.
47
48 figure
49 plot(tt,u(tt),'--b',tt,u1,'r',...
50      tt,u2,'g',tt,u3,'m')
51 legend('exact',sprintf('N=%d',N(1)),sprintf('N=%d',N(2))...
52      ,sprintf('N=%d',N(3)),'Location','northwest')
53
54 if isTest1 && b==3
55     ylim([-1.5,1.5])
56 end

```

Listing 1: PLRR_test for solving two-point BVP

```

1 % =====
2 % Solve a linear two-point boundary-value problem
3 %            $-(p(x)u'(x))' + q(x)u(x) = f(x)$ ,       $a \leq x \leq b$ ,

```

```

4 % with initial conditions
5 %                                     u(a)=0, u'(b)=0.
6 % =====
7
8 % *****
9 % Test 1
10 isTest1 = true;
11 f=@(x)pi^2/4*sin(pi/2*x);
12 u=@(x)sin(pi/2*x);
13
14 % Test 2
15 % isTest1 = false;
16 % f=@(x)pi^2*sin(pi*x);
17 % u=@(x)sin(pi*x)+pi*x;
18 % *****
19
20 p=@(x)x.^0; % p(x)=1
21 q=@(x)x.*0; % q(x)=0
22 a=0;
23 b=3;
24 % b=1;
25 N=[3,5,10];
26
27 % Since the dimensions are different, we can't
28 % simply use a matrix for storage. List?
29 uc1=PQRR(f,p,q,a,b,N(1));
30 uc2=PQRR(f,p,q,a,b,N(2));
31 uc3=PQRR(f,p,q,a,b,N(3));
32
33 % output coef vector of approx solution in command window
34 disp(uc3)
35
36 % plot graphs of exact solution & approximate solutions
37 t1=linspace(a,b,N(1)+1);
38 t2=linspace(a,b,N(2)+1);
39 t3=linspace(a,b,N(3)+1);
40 tt=linspace(a,b,100+1);
41
42 % approx solutions
43 u1=PQRR_intpol(t1,[0;uc1],tt);
44 u2=PQRR_intpol(t2,[0;uc2],tt);
45 u3=PQRR_intpol(t3,[0;uc3],tt);
46
47 figure
48 plot(tt,u(tt),'--b',tt,u1,'r',...
49      tt,u2,'g',tt,u3,'m')
50 legend('exact',sprintf('N=%d',N(1)),sprintf('N=%d',N(2))...
51        ,sprintf('N=%d',N(3)),'Location','northwest')
52
53 if isTest1 && b==3
54     ylim([-1.5,1.5])
55 end

```

Listing 2: PQRR_test for solving two-point BVP

4.2 Error Estimates

```

1 % Error Estimates for PLRR and PQRR
2 % See error analysis at <error_analysis.mlx>
3
4 % (1) infinity norm --> infinity norm
5 % ||u-u_h||_{L^{\infty}}=sqrt(int((u-u_h)^2,a,b))
6 % p.w. linear: O(h^2); cubic spline: O(h^4)
7 % So we can guess quadratic element is O(h^3)
8
9 % (2) L^2 norm --> L^2 norm
10 % ||u-u_h||_{L^2}=sqrt(int((u-u_h)^2,a,b))

```

```

11 % p.w. linear: O(h^2);    cubic spline: O(h^4)
12 % So we can guess quadratic element is O(h^3)
13
14 % (3)  H^1 norm --> L^2 norm
15 % ||u-u_h||_{H^1}=sqrt(int((u'-u_h')^2,a,b)+int((u-u_h)^2,a,b))
16 % p.w. linear: O(h);      cubic spline: O(h^3)
17 % So we can guess quadratic element is O(h^2)
18
19
20 % =====
21 % Solve a linear two-point boundary-value problem
22 %          -(p(x)u'(x))'+q(x)u(x)=f(x),    a<=x<=b,
23 % with initial conditions
24 %                      u(a)=0, u'(b)=0.
25 % =====
26
27 % *****
28 % Test 1
29 f=@(x)pi^2/4*sin(pi/2*x);
30 u=@(x)sin(pi/2*x);
31 u1=@(x)pi/2*cos(pi/2*x); % direvative of u
32
33 % Test 2
34 f=@(x)pi^2*sin(pi*x);
35 u=@(x)sin(pi*x)+pi*x;
36 u1=@(x)pi*(1+cos(pi*x)); % direvative of u
37 % *****
38
39 p=@(x)x.^0; % p(x)=1
40 q=@(x)x.*0; % q(x)=0
41 a=0;
42 b=3;
43
44
45 % err = O(h^n), log(err)=n*log(h), slope k = n, which represents
46 % its convergence order. So we need to let log(h) be the x axis,
47 % log(err) be the y axis, and see how the slopes of those norms
48 % look like.
49 n=5;
50 N=8; % no. of points of log(h)
51 h=zeros(N,1);
52 err_inf=zeros(N,2); % infinity norm
53 err_L2 = zeros(N,2); % L^2 norm
54 err_H1 = zeros(N,2); % H^1 norm
55 for i=1:N
56     h(i)=(b-a)/n;
57     t = linspace(a,b,n+1).';
58     tt=linspace(a,b,5*n+1).'; % interpolation every 5 points
59
60     uc1=PLRR(f,p,q,a,b,n); % try also 2*n parts. See footnote for analysis
61     uc2=PQRR(f,p,q,a,b,n);
62     [y1, y1_1]=PLRR_intpol(t,[0;uc1],tt);
63     [y2, y2_1]=PQRR_intpol(t,[0;uc2],tt);
64
65     e = (u(tt)-y1).^2;
66     e1 = (u1(tt)-y1_1).^2;
67     err_inf(i,1)=max(abs(u(tt)-y1)); % norm(u(tt)-y1, inf)
68     err_L2(i,1)=sqrt(sum(e(1:end-1))*h(i)/5); % approx equal to:
69                                     % norm(u(tt)-y1)*sqrt(h(i)/5)
70     err_H1(i,1)=sqrt(sum(e(1:end-1)+e1(1:end-1))*h(i)/5);
71
72     e = (u(tt)-y2).^2;
73     e1 = (u1(tt)-y2_1).^2;
74     err_inf(i,2)=max(abs(u(tt)-y2));
75     err_L2(i,2)=sqrt(sum(e(1:end-1))*h(i)/5);
76     err_H1(i,2)=sqrt(sum(e(1:end-1))*h(i)/5+sum(e1(1:end-1))*h(i)/5);
77

```

```

78     n=2*n; % delta(log(h)) = -log(2) = -0.6931
79 end
80
81 logh=log(h);
82 loge_inf=log(err_inf);
83 loge_L2=log(err_L2);
84 loge_H1=log(err_H1);
85
86 % L^{inf} norm
87 figure
88 plot(logh,loge_inf(:,1),'b--',logh,loge_inf(:,2),'m--','LineWidth',2)
89 hold on
90 plot(logh,2*logh,logh,3*logh)
91 legend('PLRR','PQRR','2*log(h)','3*log(h)','Location','northwest')
92 xlabel('log(h)')
93 ylabel('log(err)')
94 title('Error Estimates under $L^{\infty}$ Norm','interpreter','latex')
95
96 % Get convergence orders of these methods (i.e. their slopes)
97 % Or use built-in func polyfit instead
98 P1_inf=lsq(logh,loge_inf(:,1),1); % use linear polynomial to fit, i.e. y = a1*x + a0
99     , and P1 = [a0, a1]
100 k_PLRR_inf = P1_inf(2) % its convergence order should amount to its slope
101 % it may not be an integer, but it is pretty
102 % much close to one, say 1.9961 or whatever
103
104 P2_inf=lsq(logh,loge_inf(:,2),1);
105 k_PQRR_inf = P2_inf(2)
106
107 % L^2 norm
108 figure
109 plot(logh,loge_L2(:,1),'b--',logh,loge_L2(:,2),'m--','LineWidth',2)
110 hold on
111 plot(logh,2*logh,logh,3*logh)
112 legend('PLRR','PQRR','2*log(h)','3*log(h)','Location','northwest')
113 xlabel('log(h)')
114 ylabel('log(err)')
115 title('Error Estimates under $L^2$ Norm','interpreter','latex')
116
117 P1_L2=lsq(logh,loge_L2(:,1),1);
118 k_PLRR_L2 = P1_L2(2)
119
120 P2_L2=lsq(logh,loge_L2(:,2),1);
121 k_PQRR_L2 = P2_L2(2)
122
123 % H^1 norm
124 figure
125 plot(logh,loge_H1(:,1),'b--',logh,loge_H1(:,2),'m--','LineWidth',2)
126 hold on
127 plot(logh,logh,logh,2*logh)
128 legend('PLRR','PQRR','log(h)','2*log(h)','Location','northwest')
129 xlabel('log(h)')
130 ylabel('log(err)')
131 title('Error Estimates under $H^1$ Norm','interpreter','latex')
132
133 P1_H1=lsq(logh,loge_H1(:,1),1);
134 k_PLRR_H1 = P1_H1(2)
135
136 P2_H1=lsq(logh,loge_H1(:,2),1);
137 k_PQRR_H1 = P2_H1(2)
138
139 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% footnote %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
140 % err = O(h^2) = C*h^2
141 % log(err) = log(C*(h/2)^2)
142 %           = log(err) - log(4)
143 % So doubling the no. of intervals
144 % just makes 'PLRR' translate down,

```



```

144 % remaining its slope unchanged.
145 %%%%%%%%%%% footnote %%%%%%%%%%%

```

Listing 3: Error Estimates for FEM

4.3 PLRR and PQRR

These are the exact methods we used for solving this two-point boundary value problem.

```

1 % Piecewise Linear Rayleigh-Ritz method
2 % =====
3 % Solve a linear two-point boundary-value problem
4 %          -(p(x)u'(x))'+q(x)u(x)=f(x),    a<=x<=b,
5 % with initial conditions
6 %                      u(a)=0, u'(b)=0.
7 % =====
8 % See the theory part at <reference.mlx>
9 function uc=PLRR(f,p,q,a,b,N)
10 % INPUT:
11 %   f: f(x)
12 %   p: p(x); optional, by default p(x)=1
13 %   q: q(x); optional, by default q(x)=0
14 %   a, b: interval [a,b]; optional, by default [a,b]=[0,1]
15 %   N: # of evenly spaced intervals; optional, by default N=10
16 % OUTPUT:
17 %   uc: coef vector of approx solution
18
19 % =====
20 % ***** NOTE that f, p, q must be element-wise functions *****
21 % e.g. f(x)=@(x)x.^2+1 (NOT x^2+1)
22 % =====
23
24 % set default args
25 if nargin<6
26     N=10;
27     if nargin<5
28         b=1;
29         if nargin<4
30             a=0;
31             if nargin<3
32                 q=@(x)x.*0;
33                 if nargin<2
34                     p=@(x)x.^0;
35                     if nargin<1
36                         error('Error! Function f is not set.')
37                     end
38                 end
39             end
40         end
41     end
42 end
43 if b<a
44     tmp=a; a=b; b=tmp;
45 end
46
47 h=(b-a)/N; % step length. no check for N, hhh
48 A=zeros(N); % coef matrix, symmetric tridiagonal
49 c=zeros(N,1); % right-hand side (RHS) vector
50 xx=linspace(a,b,N+1);
51
52 % determine tridiagonal elements of A, and RHS vector c
53 for i=1:N-1
54     % Or use MATLAB func <integral> instead of <Gaussquad>.
55     % --> <Ctrl + F> --> Replace All. Then go to PLRR_test
56     % script file & run it again, it shall get same graph.
57     A(i,i)=Gaussquad(p,xx(i),xx(i+2))+...
58         Gaussquad(@(x)(x-xx(i)).^2.*q(x),xx(i),xx(i+1))+...
59         Gaussquad(@(x)(xx(i+2)-x).^2.*q(x),xx(i+1),xx(i+2));

```

```

60     A(i,i+1)=-Gaussquad(p,xx(i+1),xx(i+2))+...
61         Gaussquad(@(x)(x-xx(i+1)).*(xx(i+2)-x).*q(x),xx(i+1),xx(i+2));
62
63     c(i)=Gaussquad(@(x)(x-xx(i)).*f(x),xx(i),xx(i+1))+...
64         Gaussquad(@(x)(xx(i+2)-x).*f(x),xx(i+1),xx(i+2));
65 end
66 A(N,N)=Gaussquad(p,xx(N),xx(N+1))+...
67     Gaussquad(@(x)(x-xx(N)).^2.*q(x),xx(N),xx(N+1));
68 c(N)=Gaussquad(@(x)(x-xx(N)).*f(x),xx(N),xx(N+1));
69
70 A=A+diag(diag(A,1),-1); % let A_{i,i-1}=A_{i-1,i}
71 A=A/h^2;
72 c=c/h;
73
74 uc = solveTridiag(A,c); % O(n) operations
75 % uc=A\c; % coef vector of approx solution
76 % or use user-defined function <GauEli>
77 % uc=GauEli(A,c); % O(n^3) operations
78
79 end

```

Listing 4: Piecewise Linear Rayleigh-Ritz method

```

1 % Piecewise Quadratic element Rayleigh-Ritz method
2 % =====
3 % Solve a linear two-point boundary-value problem
4 %      -(p(x)u'(x))'+q(x)u(x)=f(x),    a<=x<=b,
5 % with initial conditions
6 %      u(a)=0, u'(b)=0.
7 % =====
8 % See the theory part under folder <quadratic_element_img>
9 function uc=PQRR(f,p,q,a,b,N)
10 % INPUT:
11 %   f: f(x)
12 %   p: p(x); optional, by default p(x)=1
13 %   q: q(x); optional, by default q(x)=0
14 %   a, b: interval [a,b]; optional, by default [a,b]=[0,1]
15 %   N: # of evenly spaced intervals; optional, by default N=10
16 % OUTPUT:
17 %   uc: coef vector of approx solution
18
19 % =====
20 % ***** NOTE that f, p, q must be element-wise functions *****
21 % e.g. f(x)=@(x)x.^2+1 (NOT x^2+1)
22 % =====
23
24 % set default args
25 if nargin<6
26     N=10;
27     if nargin<5
28         b=1;
29         if nargin<4
30             a=0;
31             if nargin<3
32                 q=@(x)x.*0;
33                 if nargin<2
34                     p=@(x)x.^0;
35                     if nargin<1
36                         error('Error! Function f is not set.')
37                     end
38                 end
39             end
40         end
41     end
42 end
43 if b<a
44     tmp=a; a=b; b=tmp;

```

```

45 end
46
47 h=(b-a)/N; % step length. no check for N, hhh
48 A=zeros(2*N); % coef matrix, symmetric
49 c=zeros(2*N,1); % right-hand side (RHS) vector
50 xx=linspace(a,b,N+1);
51
52 % k = ( x- x_{j-1} ) / h_j, 0 <= k <=1
53 % u_h = u_{j-1}*(2k-1)(k-1) + u_{j-1/2}*4k(1-k) + u_j*k(2k-1)
54 % u_h' = u_{j-1}*(4k-3)/h_j + u_{j-1/2}*(-8k+4)/h_j + u_j*(4k-1)/h_j
55
56 % hopefully inlined
57 phi1 = @(k)(2*k-1).*(k-1);
58 phi2 = @(k)4*k.*(1-k);
59 phi3 = @(k)k.*(2*k-1);
60 k1 = @(k)(4*k-3)/h;
61 k2 = @(k)(4-8*k)/h;
62 k3 = @(k)(4*k-1)/h;
63
64 % determine tridiagonal elements of A, and RHS vector c
65 for j = 1 : N
66     % Or use MATLAB func <integral> instead of <Gaussquad>.
67     % --> <Ctrl + F> --> Replace All. Then go to PQRR_test
68     % script file & run it again, it shall get same graph.
69     A(2*j-1,2*j-1) = Gaussquad(@(k)(p(k*h+xx(j)).*k2(k).^2+...
70         q(k*h+xx(j)).*phi2(k).^2)*h, 0, 1); % omit *h in each item for optimization
71     A(2*j-1,2*j) = Gaussquad(@(k)(p(k*h+xx(j)).*k2(k).*k3(k)+...
72         q(k*h+xx(j)).*phi2(k).*phi3(k))*h, 0, 1);
73     A(2*j,2*j-1) = A(2*j-1, 2*j); % symmetric
74
75     A(2*j,2*j) = Gaussquad(@(k)(p(k*h+xx(j)).*k3(k).^2+...
76         q(k*h+xx(j)).*phi3(k).^2)*h, 0, 1);
77     if(j<N)
78         A(2*j,2*j)=A(2*j,2*j)+Gaussquad(@(k)(p(k*h+xx(j+1)).*k1(k).^2+...
79             q(k*h+xx(j+1)).*phi1(k).^2)*h, 0, 1);
80
81         A(2*j,2*j+1) = Gaussquad(@(k)(p(k*h+xx(j+1)).*k1(k).*k2(k)+...
82             q(k*h+xx(j+1)).*phi1(k).*phi2(k))*h, 0, 1);
83
84         A(2*j,2*j+2) = Gaussquad(@(k)(p(k*h+xx(j+1)).*k1(k).*k3(k)+...
85             q(k*h+xx(j+1)).*phi1(k).*phi3(k))*h, 0, 1);
86
87         A(2*j+1,2*j) = A( 2*j ,2*j+1);
88         A(2*j+2,2*j) = A( 2*j ,2*j+2);
89     end
90
91     c(2*j-1) = Gaussquad(@(k)(f(k*h+xx(j)).*phi2(k))*h, 0, 1);
92     c( 2*j ) = Gaussquad(@(k)(f(k*h+xx(j)).*phi3(k))*h, 0, 1);
93     if(j<N)
94         c(2*j)=c(2*j)+Gaussquad(@(k)(f(k*h+xx(j+1)).*phi1(k))*h, 0, 1);
95     end
96 end
97
98 % uc=A\c; % coef vector of approx solution
99 % or use user-defined function <GauEli>
100 uc=GauEli(A,c);
101
102 end

```

Listing 5: Piecewise Quadratic element Rayleigh-Ritz method global stiffness matrix version

```

1 % Piecewise Quadratic element Rayleigh-Ritz method
2 % element stiffness matrix version
3 % Inspired by Miss White ;)
4 function u = PQRR_es(f,p,q,a,b,n)
5
6 h = (b-a)/n; % step length. no check for N, hhh

```

```

7 A = zeros(2*n+1); % coef matrix, symmetric
8 F = zeros(2*n+1,1); % right-hand side (RHS) vector
9 x = linspace(a,b,n+1);
10
11 % k = ( x - x_{j-1} ) / h_j, 0 <= k <=1
12 % u_h = u_{j-1}*(2k-1)(k-1) + u_{j-1/2}*4k(1-k) + u_j*k(2k-1)
13 % u_h' = u_{j-1}*(4k-3)/h_j + u_{j-1/2}*(-8k+4)/h_j + u_j*(4k-1)/h_j
14
15 % hopefully inlined
16 phi1 = @(k)(2*k-1).*(k-1);
17 phi2 = @(k)4*k.*(1-k);
18 phi3 = @(k)k.*(2*k-1);
19 k1 = @(k)(4*k-3)/h;
20 k2 = @(k)(4-8*k)/h;
21 k3 = @(k)(4*k-1)/h;
22
23 K = zeros(3); % element stiffness matrix
24 for j = 1 : n
25     K(1,1) = integral(@(k)(p(k*h+x(j)).*k1(k).^2+...
26         q(k*h+x(j)).*phi1(k).^2)*h, 0, 1); % omit *h in each item for optimization
27
28     K(1,2) = integral(@(k)(p(k*h+x(j)).*k1(k).*k2(k)+...
29         q(k*h+x(j)).*phi1(k).*phi2(k))*h, 0, 1);
30
31     K(1,3) = integral(@(k)(p(k*h+x(j)).*k1(k).*k3(k)+...
32         q(k*h+x(j)).*phi1(k).*phi3(k))*h, 0, 1);
33
34     K(2,2) = integral(@(k)(p(k*h+x(j)).*k2(k).^2+...
35         q(k*h+x(j)).*phi2(k).^2)*h, 0, 1);
36
37     K(2,3) = integral(@(k)(p(k*h+x(j)).*k2(k).*k3(k)+...
38         q(k*h+x(j)).*phi2(k).*phi3(k))*h, 0, 1);
39
40     K(3,3) = integral(@(k)(p(k*h+x(j)).*k3(k).^2+...
41         q(k*h+x(j)).*phi3(k).^2)*h, 0, 1);
42
43     % using symmetry
44     K(2,1) = K(1,2);
45     K(3,1) = K(1,3);
46     K(3,2) = K(2,3);
47
48     % assemble the element stiffness matrix
49     A(2*j-1,2*j-1) = A(2*j-1,2*j-1) + K(1,1);
50     A(2*j-1, 2*j ) = A(2*j-1, 2*j ) + K(1,2);
51     A(2*j-1,2*j+1) = A(2*j-1,2*j+1) + K(1,3);
52     A( 2*j ,2*j-1) = A( 2*j ,2*j-1) + K(2,1);
53     A( 2*j , 2*j ) = A( 2*j , 2*j ) + K(2,2);
54     A( 2*j ,2*j+1) = A( 2*j ,2*j+1) + K(2,3);
55     A(2*j+1,2*j-1) = A(2*j+1,2*j-1) + K(3,1);
56     A(2*j+1, 2*j ) = A(2*j+1, 2*j ) + K(3,2);
57     A(2*j+1,2*j+1) = A(2*j+1,2*j+1) + K(3,3);
58
59     % calculate & (implicitly) assemble right-hand side vector
60     F(2*j-1) = F(2*j-1) + integral(@(k)(f(k*h+x(j)).*phi1(k))*h, 0, 1);
61     F( 2*j ) = F( 2*j ) + integral(@(k)(f(k*h+x(j)).*phi2(k))*h, 0, 1);
62     F(2*j+1) = F(2*j+1) + integral(@(k)(f(k*h+x(j)).*phi3(k))*h, 0, 1);
63 end
64
65 A = A(2:2*n+1,2:2*n+1);
66 F = F(2:2*n+1);
67
68 u = A\F;
69 end

```

Listing 6: Piecewise Quadratic element Rayleigh-Ritz method element stiffness matrix version

4.4 PLRR_test and PQRR_test

Interpolations for PLRR and PQRR.

```

1 % Interpolation for PLRR
2 function [u, u1]=PLRR_intpol(xx,uc,xq)
3 N = length(xx)-1;
4 m = length(xq);
5 h = (xx(end)-xx(1))/N; % evenly spaced step length
6 u = zeros(m,1);
7 u1 = zeros(m,1); % derivative of u
8
9 % a lazy solution
10 %{
11 for i=1:N
12     for j=1:m
13         if xq(j)>=xx(i) && xq(j)<=xx(i+1)
14             u(j)=uc(i)*(xx(i+1)-xq(j))/h+uc(i+1)*(xq(j)-xx(i))/h;
15             u1(j)=(uc(i+1)-uc(i))/h;
16         end
17     end
18 end
19 %}
20
21 % When xq is an *ordered* further division based on xx,
22 % for example, xq = linspace(a,b,10*N+1), then an eager
23 % version can be developed like this:
24 % {
25 k = (m-1)/N;
26 for i=1:N
27     % Note that, we are better off using floor, since floor(k * N)
28     % might NOT be equal to m-1, in turn, xq(m) remains what it is.
29     k1 = 1 + ceil(k*(i-1)); % Get upper bounds, at most 1 element of
30     k2 = 1 + ceil(k*i);      % xq(k1:k2) NOT in [xx(i), xx(i+1)].
31     for j=k1:k2
32         if xq(j)>=xx(i) && xq(j)<=xx(i+1)
33             u(j)=uc(i)*(xx(i+1)-xq(j))/h+uc(i+1)*(xq(j)-xx(i))/h;
34             u1(j)=(uc(i+1)-uc(i))/h;
35         end
36     end
37 end
38 %}
39
40 % *****FOOTNOTE*****
41 % Say, m=101, N=3, floor(k*N)=100?? Luckily, this is true in
42 % MATLAB (C, C++, Python, etc.) due to the fact that:
43 % >> k=100/3; % k = 33.333333333333336
44 % >> k*3==100
45 % ans = (logical) 1
46 % That's how double-precision numbers store and operate.
47 % Notice that floor(33.3333*3) = 99, NOT 100.
48 % *****FOOTNOTE*****
49 end

```

Listing 7: Interpolation for PLRR

```

1 % Interpolation for PQRR
2 function [u, u1]=PQRR_intpol(xx,uc,xq)
3 N=(length(uc)-1)/2; % No. of coefs of integer points
4 m=length(xq);       % No. of query points
5 h=(xx(end)-xx(1))/N; % evenly spaced step length
6 u=zeros(m,1);       % approximations at these m query points
7 u1=zeros(m,1);      % derivative of u
8
9 phi1 = @(k)(2*k-1).*(k-1);
10 phi2 = @(k)4*k.*(1-k);
11 phi3 = @(k)k.*(2*k-1);

```

```

12 kk1 = @(k)(4*k-3)/h;
13 kk2 = @(k)(4-8*k)/h;
14 kk3 = @(k)(4*k-1)/h;
15
16 % a lazy solution
17 %{
18 for i=1:N
19     for j=1:m
20         if xq(j)>=xx(i) && xq(j)<=xx(i+1)
21             u(j)=uc(2*i)*phi2((xq(j)-xx(i))/h)+... % central part
22                 uc(2*i+1)*phi3((xq(j)-xx(i))/h)+... % left half of current
23                 uc(2*i-1)*phi1((xq(j)-xx(i))/h); % right half of previous
24
25             u1(j)=uc(2*i)*kk2((xq(j)-xx(i))/h)+...
26                 uc(2*i+1)*kk3((xq(j)-xx(i))/h)+...
27                 uc(2*i-1)*kk1((xq(j)-xx(i))/h);
28         end
29     end
30 end
31 %}
32
33 % When xq is an *ordered* further division based on xx,
34 % for example, xq = linspace(a,b,10*N+1), then an eager
35 % version can be developed like this:
36 % {
37 k = (m-1)/N;
38 for i=1:N
39     % Note that, we are better off using floor, since floor(k * N)
40     % might NOT be equal to m-1, in turn, xq(m) remains what it is.
41     k1 = 1 + ceil(k*(i-1)); % Get upper bounds, at most 1 element of
42     k2 = 1 + ceil(k*i); % xq(k1:k2) NOT in [xx(i), xx(i+1)].
43     for j=k1:k2
44         if xq(j)>=xx(i) && xq(j)<=xx(i+1)
45             u(j)=uc(2*i)*phi2((xq(j)-xx(i))/h)+... % central part
46                 uc(2*i+1)*phi3((xq(j)-xx(i))/h)+... % left half of current
47                 uc(2*i-1)*phi1((xq(j)-xx(i))/h); % right half of previous
48
49             u1(j)=uc(2*i)*kk2((xq(j)-xx(i))/h)+...
50                 uc(2*i+1)*kk3((xq(j)-xx(i))/h)+...
51                 uc(2*i-1)*kk1((xq(j)-xx(i))/h);
52         end
53     end
54 end
55 %}
56
57 % *****FOOTNOTE*****
58 % Say, m=101, N=3, floor(k*N)=100?? Luckily, this is true in
59 % MATLAB (C, C++, Python, etc.) due to the fact that:
60 % >> k=100/3; % k = 33.333333333333336
61 % >> k*3==100
62 % ans = (logical) 1
63 % That's how double-precision numbers store and operate.
64 % Notice that floor(33.3333*3) = 99, NOT 100.
65 % *****FOOTNOTE*****
66 end

```

Listing 8: Interpolation for PQRR

4.5 Gaussquad

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

```

1 % Gaussian quadrature
2 function T = Gaussquad(f,a,b,n)
3 if nargin<4

```

```

4     n=8;
5 end
6 % Gaussian quadrature points & weights on [-1,1]
7 if n == 2
8     w = [1,1]; % weights
9     p = [-1/sqrt(3), 1/sqrt(3)]; % points
10 elseif n == 4
11     w = [0.3478548451, 0.3478548451, 0.6521451549, 0.6521451549];
12     p = [0.8611363116, -0.8611363116, 0.3399810436, -0.3399810436];
13 elseif n == 8
14     w = [0.1012285363, 0.1012285363, 0.2223810345, 0.2223810345, 0.3137066459, ...
15           0.3137066459, 0.3626837834, 0.3626837834];
16     p = [0.9602898565, -0.9602898565, 0.7966664774, -0.7966664774, 0.5255324099, ...
17           -0.5255324099, 0.1834346425, -0.1834346425];
18 else
19     error('n must be either 2 or 4 or 8')
20 end
21
22 % take a linear transform: x = t*(b-a)/2 + (a+b)/2, where t belongs to [-1,1]
23 % int(f,[a,b]) = (b-a)/2 * int(f(t*(b-a)/2 + (a+b)/2), [-1,1])
24 w = 0.5*(b-a)*w;
25 p = 0.5*(b-a)*p+0.5*(a+b);
26
27 T = w*f(p).';
28 end

```

Listing 9: Gaussian quadrature

4.6 GauEli

A routine for solving linear equations of the form $Ax = b$.

```

1 % Gaussian Elimination: solve linear systems
2 % of algebraic equations of the form Ax=b.
3 % 10170437 Mark Taylor
4
5 function [x,U] = GauEli(A, b)
6
7 [m,n]=size(A);
8 if m ~= n
9     error('A must be a square matrix!')
10 end
11
12 U=[A,b]; % Augmented matrix of A
13
14 for j=1:n
15     k=maxIndex(U(:,j),j,n);
16     if abs(U(k,j))>eps
17         if k~=j
18             temp=U(j,j:n+1);
19             U(j,j:n+1)=U(k,j:n+1);
20             U(k,j:n+1)=temp;
21         end
22     else
23         x='The system has infinitely many solutions!';
24         U=NaN;
25         return;
26     end
27
28 % Perform Gauss elimination.
29 for i=j+1:n
30     if abs(U(i,j))>eps
31         t=U(i,j)/U(j,j);
32         U(i,j:n+1)=U(i,j:n+1)-t*U(j,j:n+1);
33     end
34 end

```



```
35 end
36
37 x=zeros(n,1);
38 % Solve upper diagonal linear system, i.e.  $U(1:n,1:n)x=U(:,n+1)$ .
39 x(n)=U(n,n+1)/U(n,n);
40 for i=n-1:-1:1
41     x(i)=(U(i,n+1)-U(i,i+1:n)*x(i+1:n))/U(i,i);
42 end
43
44 end
```

Listing 10: Gaussian Elimination

References

- [1] R. L. Burden and J. D. Faires. *Numerical Analysis*. Cengage Learning, 9th edition, 2010.
- [2] L. Ronghua. *Numerical Methods for Partial Differential Equations*. Higher Education Press, 2th edition, 2010.
- [3] C. University. Chapter 5: Error estimates for the finite element method. page 4, 2002. URL <http://pi.math.cornell.edu/~demlow/425/chap5.pdf>.

Appendices



Hello from the Beatles. 😊