CSC446 A3

Peiqi Wang 1001132561

April 4, 2019

Problem 1

Implement Ritz-Galerkin method on equidistant grid with piecewise linear hat function for

$$-y'' + y = (\pi^2+1)\sin(\pi x) \quad x \in (0,1)$$
$$y(0) = 1 \quad y'(1) = \frac{1}{2}(e - \frac{1}{e}) - \pi$$

where the real solution to the problem is given by

$$y(x) = \frac{1}{2}(e^x + e^{-x}) + \sin(\pi x)$$

solution. Note this BVP is a special case of BVP that appeared in the textbook, so

$$a_{k,l} = \langle \varphi_l', \varphi_k' \rangle + \langle \varphi_l, \varphi_k \rangle = \begin{cases} \frac{2(h^2+3)}{3h} & l = k \\ \frac{h^2-6}{6h} & l = k-1, k+1 \\ 0 & \text{otherwise} \end{cases}$$

for $k = 1, 2, \cdots, m-1$. Let

$$\varphi_0(x) = \left(\frac{1}{2}\left(e - \frac{1}{e}\right) - \pi\right) x + 1$$

satisfies the boundary conditions. Now we compute the right hand side of the linear system

$$\begin{aligned} b_k &= \langle f, \varphi_k \rangle - a_{k,0} = \langle f, \varphi_k \rangle - \langle \varphi_0, \varphi_k \rangle \\ &= \int_0^1 (\pi^2+1)\sin(\pi x)\varphi_k(x)dx - \int_0^1 \varphi_0(x)\varphi_k(x)dx \\ &= \int_{(k-1)h}^{kh} (\pi^2+1)\sin(\pi x)\left(1 - k + \frac{x}{h}\right)dx + \int_{kh}^{(k+1)h} (\pi^2+1)\sin(\pi x)\left(1 + k - \frac{x}{h}\right)dx \\ &\quad - \int_{(k-1)h}^{kh} \left(\left(\frac{1}{2}\left(e - \frac{1}{e}\right) - \pi\right)x + 1\right)\left(1 - k + \frac{x}{h}\right)dx \\ &\quad - \int_{kh}^{(k+1)h} \left(\left(\frac{1}{2}\left(e - \frac{1}{e}\right) - \pi\right)x + 1\right)\left(1 + k - \frac{x}{h}\right)dx \end{aligned}$$

for $k = 1, 2, \cdots, m-1$. To allow for arbitrary approximate $y_m(x)$ at $x = 1$, we have a basis function $\varphi_m(x)$ supported over $[x_{m-1}, x_m]$ only. The corresponding entries in A and b is as follows

$$\begin{aligned} a_{m,m-1} &= \frac{h^2-6}{6h} \\ a_{m,m} &= \frac{h^2+3}{3h} \\ b_m &= \langle f, \varphi_k \rangle - \left(- \langle \varphi_0'', \varphi_k \rangle + \langle \varphi_0, \varphi_k \rangle\right) \\ &= \int_{(m-1)h}^{mh} (\pi^2+1)\sin(\pi x)\left(1 - m + \frac{x}{h}\right)dx \\ &\quad - \int_{(m-1)h}^{mh} \left(\left(\frac{1}{2}\left(e - \frac{1}{e}\right) - \pi\right)x + 1\right)\left(1 - m + \frac{x}{h}\right)dx \end{aligned}$$

Due to complexity of integrands that arises in the problem, we will numerically integrate all integrals with 5-point Gaussian Quadrature. Implementation is in appendix. We have maximum error as follows

m	max error	ratio
10	0.0014590704	0.0000000000
20	0.0003643921	0.2497426391
40	0.0000910745	0.2499354110
80	0.0000227698	0.2500132232
160	0.0000056924	0.2499964951
320	0.0000014231	0.2499961108
640	0.0000003558	0.2500486353

We see as $h = 1/m$ halves, the maximum error decreases by a factor of 4. $\square$

Problem 2

Repeat qestion 1, but use cubic B-spline basis instead of piecewise linear hat basis function.

solution. Due to complexity of integrands that arises in the problem, we will numerically integrate all integrals with 5-point Gaussian Quadrature. We use the same φ_0 as previously described. Generic formula for A and b is given by

$$a_{k,l} = \int_0^1 \left[B_l'(x)B_k'(x) + B_l(x)B_k(x)\right] dx$$
$$b_k = \int_0^1 \left[f(x)B_k(x) + \varphi_0''(x)B_k(x) - \varphi_0(x)B_k(x)\right] dx = \int_0^1 \left[f(x)B_k(x) - \varphi_0(x)B_k(x)\right] dx$$

since $\varphi_0''(x)$ is a zero function on $[0, 1]$. First derivates of φ_k are computed from online integral calculator. Implementation is in appendix. Maximum error is given as follows

m	max error	ratio
10	0.0000139296	0.0000000000
20	0.0000008599	0.0617307527
40	0.0000000535	0.0622524299
80	0.0000000033	0.0624312541
160	0.0000000002	0.0622318081
320	0.0000000000	0.0596110771
640	0.0000000000	0.6453057277

As h halves, the maximum error decreases by a factor of 16. This expected for cubic basis functions, which makes erray decrease proportional to h^4. Compared to problem 1, where the linear hat basis function makes error decrease proportional to h^2, error for solution using B-spline basis decreases much faster. $\square$

Problem 3

two-point bvp

$$-y'' + 10^4 y = 0 \qquad x \in (0, 1)$$
$$y(0) = y(1) = 1$$

where real solution is

$$y(x) = c_1 e^{100x} + c_2 e^{-100x}$$

where

$$c_1 = \frac{1 - e^{-100}}{e^1 00 - e^{-100}} \quad c_2 = \frac{e^1 00 - 1}{e^1 00 - e^{-100}}$$

solution. Use $\varphi_0(x) = 1$ and so $\varphi_0'(x) = 0$ for $x \in (0,1)$. We have

$$a_{k,l} = \langle \varphi_l', \varphi_k' \rangle + \langle 10^4 \varphi_l, \varphi_k \rangle = \int_0^1 \varphi_l'(x)\varphi_k'(x) + 10^4 \varphi_l(x)\varphi_k(x) dx$$
$$b_k = \langle f, \varphi_k \rangle - a_{k,0} = \langle 0, \varphi_k \rangle - \int_0^1 \varphi_0'(x)\varphi_k'(x) + 10^4 \varphi_0(x)\varphi_k(x) dx = -10^4 \int_0^1 \varphi_k(x) dx$$

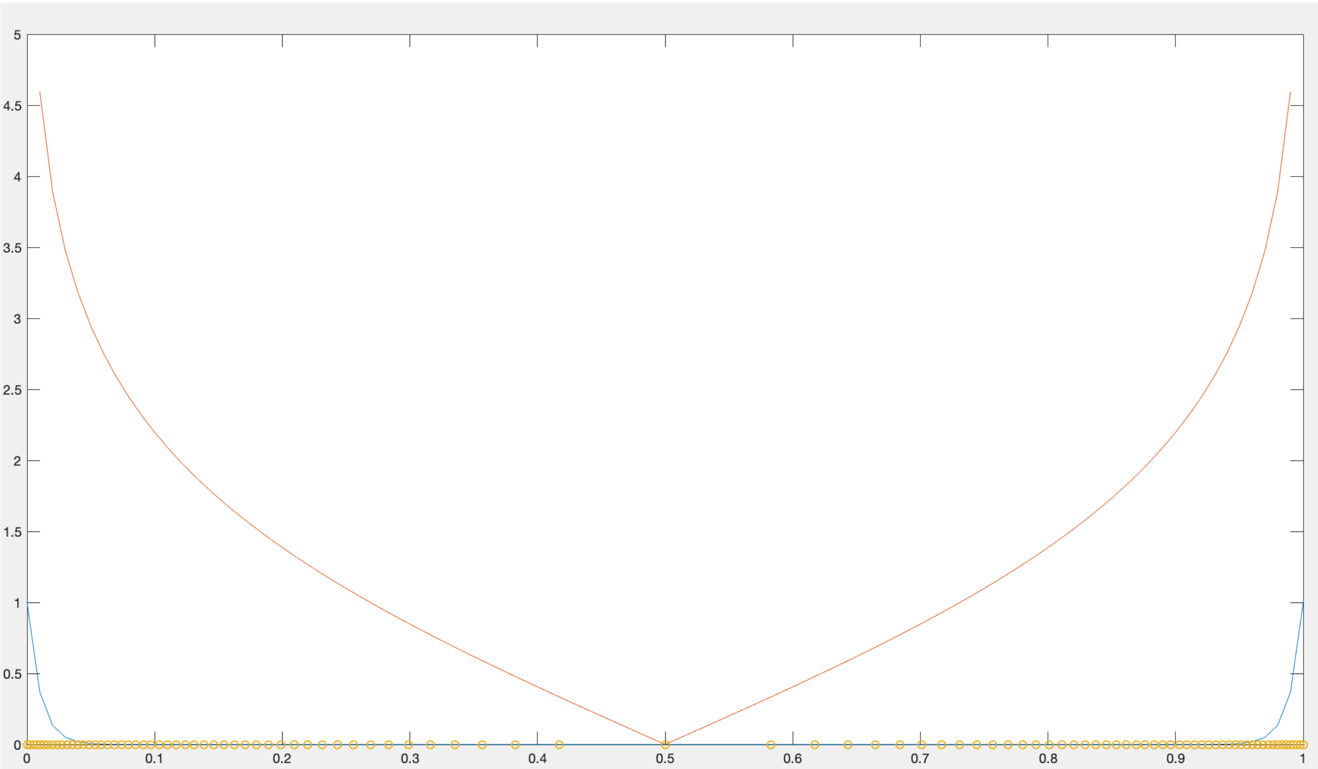
where

$$\varphi_k'(x) = \begin{cases} \frac{1}{x_k - x_{k-1}} & x \in [x_{k-1}, x_k] \\ \frac{1}{x_k - x_{k+1}} & x \in [x_k, x_{k+1}] \\ 0 & \text{otherwise} \end{cases}$$

We will numerically integrate all integrals with 5-point Gaussian Quadrature. We adapt new grid such that error over each element is approximately equal (reference). After plotting the real solution, we noticed that y' has large magnitude near 0 and 1. So we choose a monitor function such that the adapted grid point is dense near 0 and 1. In particular, we used absolute value of the logit function

$$M(x) = \left| \log \frac{x}{1-x} \right|$$

and constructed the grid as follows



Implementation is in appendix. We compare the maximum error using the equidistant (left) and adapted (right) grid.

m	max error	ratio
9	0.2415051781	0.0000000000
19	0.1815659068	0.7518095811
39	0.0888420639	0.4893102756
79	0.0269767436	0.3036483223
159	0.0060326729	0.2236249487
319	0.0015075440	0.2498965323
639	0.0003743138	0.2482937903

m	max error	ratio
9	0.1609113342	0.0000000000
19	0.0509483141	0.3166235267
39	0.0087430834	0.1716069225
79	0.0020603703	0.2356571727
159	0.0005347227	0.2595274714
319	0.0001357461	0.2538625282
639	0.0000340911	0.2511385551

We noticed that, with the same number of basis functions, the maximum error on the adapted grid is at least an order of magnitude smaller to that of the equidistant grid. □

Problem 4

2d bvp

$$-\nabla^2 u = -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 32x(1-x) + 32y(1-y) \qquad x \in (0,1) \quad y \in (0,1)$$

with Dirichlet boundary, where the real solution is

$$u(x,y) = 16x(1-x)y(1-y)$$

solution. Galerkin's equation for the above problem is derived in class, resulting in m^2 unknonwns. Note

$$\langle \varphi_k, \varphi_l \rangle = \begin{cases} \frac{2h}{3} & k = l \\ \frac{h}{6} & |k-l| = 1 \\ 0 & \text{otherwise} \end{cases} \qquad \langle \varphi_k', \varphi_l' \rangle = \begin{cases} \frac{2}{h} & k = l \\ -\frac{1}{h} & |k-l| = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned}
a_{(k,l),(i,j)} &= \langle \varphi_i', \varphi_k' \rangle \langle \varphi_j, \varphi_l \rangle + \langle \varphi_i, \varphi_k \rangle \langle \varphi_j', \varphi_l' \rangle \\
&= \begin{cases} \frac{8}{3} & i = k \land j = l \\ -\frac{1}{3} & (j = l \land |i-k| = 1) \lor (i = k \land |j-l| = 1) \\ -\frac{1}{3} & |k-i| = 1 \land |l-j| = 1 \\ 0 & \text{otherwise} \end{cases} \\
b_{(k,l)} &= \langle f, \varphi_{k,l} \rangle \\
&= \int_0^1 \int_0^1 f(x,y)\varphi_{k,l}(x,y)dxdy \\
&= \int_0^1 \int_0^1 (32x(1-x) + 32y(1-y))\varphi_k(x)\varphi_l(y)dxdy \\
&= \int_0^1 \int_0^1 \left[32x(1-x)\varphi_k(x)\right]\varphi_l(y) + \varphi_k(x)\left[32y(1-y)\varphi_l(y)\right] dxdy \\
&= \left(\int_0^1 32x(1-x)\varphi_k(x)dx\right)\left(\int_0^1 \varphi_l(y)dy\right) + \left(\int_0^1 \varphi_k(x)dx\right)\left(\int_0^1 32y(1-y)\varphi_l(y)dy\right) \\
&= h\left(\int_0^1 32x(1-x)\varphi_k(x)dx + \int_0^1 32y(1-y)\varphi_l(y)dy\right)
\end{aligned}$$

for $k, l, i, j = 1, 2, \cdots, m$. We can still use 5-point Gaussian Quadrature to integrate $b_{k,l}$ by using Fubini's theorem to simplify the expression of $b_{k,l}$ as shown above. Implementation is in appendix. Maximum error is given as follows

m	max error	ratio
9	0.0079210491	0.0000000000
19	0.0019684524	0.2485090538
39	0.0004913844	0.2496298242
79	0.0001228007	0.2499076098
159	0.0000306973	0.2499769097
319	0.0000076742	0.2499941906
639	0.0000019185	0.2499979504

As h halves, the maximum error decreases by a factor of 4. □

Appendix

```
% 'n'-point Gaussian Quadrature
%       given function 'f' with lower/upper limit 'a'/'b'
function int = gq(f,a,b,order)
    assert(any(order == [5]), 'only support {5}-point gauss quadrature');

    gq5_points = [
        0
        -1/3*sqrt(5-2*sqrt(10/7))
        +1/3*sqrt(5-2*sqrt(10/7))
        -1/3*sqrt(5+2*sqrt(10/7))
        +1/3*sqrt(5+2*sqrt(10/7))
    ];

    gq5_weights = [
        128/225
        (322+13*sqrt(70)) / 900
        (322+13*sqrt(70)) / 900
        (322-13*sqrt(70)) / 900
        (322-13*sqrt(70)) / 900
    ];

    % over [-1.1]
    fx = arrayfun(@(x,w) w*f(x*(b-a)/2+(a+b)/2), ...
        gq5_points, gq5_weights);
    int = (b-a)/2 * sum(fx);
end
```

Problem 1 code

```
clear all;
global m;

ms = [10,20,40,80,160,320,640];
% ms = [10];

fprintf('m\tmax_error\tratio\n');
for iter = 1:size(ms,2)
    m=ms(iter);
    h=1/m;

    [A, b] = assembly();
    c = A \ b;

    e = arrayfun(@(i) abs(y(i*h) - (c(i)+varphi0(i*h))), ...
        1:m);
    max_e = max(e);

    xs = arrayfun(@(i)i*h,1:m);
    plot(xs, arrayfun(@(i) y(i*h), 1:m), '-', ...
         xs, arrayfun(@(i) (c(i)+varphi0(i*h)), 1:m), '--');

    ratio = 0;
    if iter ~= 1
        ratio = max_e / pre_max_e;
    end
    fprintf('%d\t%.10f\t%.10f\n', m, max_e, ratio);
    pre_max_e = max_e;
end

function [A, b] = assembly()
    global m;

    h = 1/m;
    A = sparse(m,m);
    b = zeros(m,1);

    for k = 1:m
        if k ~= 1
            A(k,k-1) = (h^2-6)/(6*h);
        end
        if k ~= m
            A(k,k) = 2*(h^2+3)/(3*h);
            A(k,k+1) = (h^2-6)/(6*h);
            b(k) = gq(@(x) (f(x)-varphi0(x)).*(1-k+x./h), (k-1)*h, k*h, 5) ...
                + gq(@(x) (f(x)-varphi0(x)).*(1+k-x./h), k*h, (k+1)*h, 5);
        else
            A(k,k) =   (h^2+3)/(3*h);
            b(k) = gq(@(x) (f(x)-varphi0(x)).*(1-k+x./h), (k-1)*h, k*h, 5);
        end
    end
end
```

```
function fx = f(x)
    fx = (pi^2+1)*sin(pi*x);
end

function yx = y(x)
    yx = 1/2*(exp(x)+exp(-x)) + sin(pi*x);
end

function varphi0x = varphi0(x)
    varphi0x = (1/2*(exp(1)-exp(-1)) - pi)*x + 1;
end

function varphi0p = varphi0prime()
    varphi0p = (1/2*(exp(1)-exp(-1)) - pi);
end
```

Problem 2 code

```
clear all;
global m;

ms = [10,20,40,80,160,320,640];
% ms = [10];

fprintf('m\tmax_error\tratio\n');
for iter = 1:size(ms,2)
    m=ms(iter);
    h=1/m;

    [S, P] = getSP();
    [A, b] = assembly();
    c = A \ b;

    nodes = zeros(1, m);    % at x_{i=1} to 1
    for k = -2:(m-1)
        Bk_support = S(k+3,:);
        for i = Bk_support(1):Bk_support(2)
            nodes(i+1) = nodes(i+1) + c(k+3)*B(k, P(k+3,i+1), (i+1)*h);
        end
    end

    e = arrayfun(@(i) abs(y(i*h) - (nodes(i)+varphi0(i*h))), ...
        1:m);
    max_e = max(e);

    xs = arrayfun(@(i)i*h,1:m);
    plot(xs, arrayfun(@(i) y(i*h), 1:m), '-', ...
         xs, arrayfun(@(i) (c(i)+varphi0(i*h)), 1:m), '--');

    ratio = 0;
    if iter ~= 1
        ratio = max_e / pre_max_e;
    end
    fprintf('%d\t%.10f\t%.10f\n', m, max_e, ratio);
    pre_max_e = max_e;
end


function [A, b] = assembly()
    global m;

    h = 1/m;
    N = m+2;

    A = sparse(N,N);
    b = zeros(N,1);

    % k = Bk; l = Bl;
    % a_{k,l} on [x_i, x_{i+1}]
    %        contributed by B_{k}^Pk, B_{l}^Pl
```

```
    akl = @(i,Bk,Pk,Bl,Pl) ...
        gq(@(x) (Bp(Bl,Pl,x)*Bp(Bk,Pk,x) + B(Bl,Pl,x)*B(Bk,Pk,x)), ...
            i*h, (i+1)*h, 5);

    % b_k on [x_i, x_{i+1}]
    %         contributed by B_{k}^Pk, B_{l}^Pl
    bk = @(i,Bk,Pk) ...
        gq(@(x) (f(x) - varphi0(x))*B(Bk,Pk,x), ...
            i*h, (i+1)*h, 5);

    [S, P] = getSP();

    for k = -2:(m-1)
        for l = -2:(m-1)
            Bk_support = S(k+3,:);
            Bl_support = S(l+3,:);
            support = [
                max(Bk_support(1),Bl_support(1)), ...
                min(Bk_support(2),Bl_support(2)), ...
            ];

            % support does not overlap, so a_{k,l} = 0
            if support(1)>support(2)
                continue;
            end

            A(k+3,l+3) = 0;
            for i = support(1):support(2)
                A(k+3,l+3) = A(k+3,l+3) + akl(i,k,P(k+3,i+1),l,P(l+3,i+1));
            end
        end

        Bk_support = S(k+3,:);
        for i = Bk_support(1):Bk_support(2)
            b(k+3) = b(k+3) + bk(i,k,P(k+3,i+1));
        end
    end
end


function fx = f(x)
    fx = (pi^2+1)*sin(pi*x);
end

function yx = y(x)
    yx = 1/2*(exp(x)+exp(-x)) + sin(pi*x);
end

function varphi0x = varphi0(x)
    varphi0x = (1/2*(exp(1)-exp(-1)) - pi)*x + 1;
end

function varphi0p = varphi0prime()
    varphi0p = (1/2*(exp(1)-exp(-1)) - pi);
```

```
end

function [S, P] = getSP()
    global m;
    N = m+2;

    % support (m+2)x2
    %        where [S(k+3,1), S(k+3,2)+1] is support for basis B_k
    S = zeros(N,2);
    S(1,:)    = [0, 1];        % B_{-2}
    S(2,:)    = [0, 2];        % B_{-1}
    for i = 3:(m-1)
        S(i,:) = [i-3, i];
    end
    S(m,:)    = [m-3, m-1];   % B_{m-3}
    S(m+1,:) = [m-2, m-1];   % B_{m-2}
    S(m+2,:) = [m-1, m-1];   % B_{m-1}

    % polynomial (m+2)xm
    %        where P(k+3, i+1) is {0,1,2,3}-th polynomial
    %            for cubic B_k (k=-2,...,m-1)
    %            on [x_i, x_{i+1}] (i=0,...,m-1)
    P = zeros(N,m);
    P(:,:) = -1;
    P(1,1:2) = [0, 1];        % B_{-2}
    P(2,1:3) = [0, 1, 2];     % B_{-1}
    for i = 3:(m-1)
        P(i,(i-2):(i+1)) = [0, 1, 2, 3];
    end
    P(m,(m-2):m)     = [0,1,2];% B_{m-3}
    P(m+1,(m-1):m)  = [0,1];   % B_{m-2}
    P(m+2,m:m)       = [0];     % B_{m-1}
end


% polynomial function for 'i'-th basis function 'B_i'
%        and 'k'-th polynomial 'P_k' for 'B_i' where k \in {0,1,2,3}
function Bx = B(i, k, x)
    global m;
    h = 1/m;

    if i == -2
        switch k
        case 0
            Bx = (3)*(x/h) - (9/2)*(x/h)^2 + (7/4)*(x/h)^3;
        case 1
            Bx = (1/4)*((2*h-x)/h)^3;
        otherwise
            warning('k ~\in {0,1} for B_{-2)\n');
        end
        return;
    end

    if i == -1
```

```
        switch k
        case 0
            Bx = (3/2)*(x/h)^2 - (11/12)*(x/h)^3;
        case 1
            Bx = (-3/2) + (9/2)*(x/h) - (3)*(x/h)^2 + (7/12)*(x/h)^3;
        case 2
            Bx = (1/6)*((3*h-x)/h)^3;
        otherwise
            warning('k ~\in {0,1,2} for B_{-1}\n');
        end
        return;
    end

    if  (i == m-1 && any(k == [1 2 3])) || ...
        (i == m-2 && any(k == [2 3])) || ...
        (i == m-3 &&      k == 3)
        warning('k not right value for for B_{m-1,m-2,m-3}\n');
        return;
    end

    assert((i >= 0 && i <= m-1),'invalid basis i');
    l = i*h;
    r = (i+4)*h;

    switch k
    case 0
        Bx = (1/6)*((x-l)/h)^3;
    case 1
        Bx = (2/3)   - (2)*((x-l)/h)  + (2)*((x-l)/h)^2  - (1/2)*((x-l)/h)^3;
    case 2
        Bx = (-22/3) + (10)*((x-l)/h) - (4)*((x-l)/h)^2  + (1/2)*((x-l)/h)^3;
    case 3
        Bx = (1/6)*((r-x)/h)^3;
    otherwise
        warning('k ~\in {0,1,2,3} for B_i\n');
        return;
    end
end

% first order derivative of polynomial function for `i`-th basis function `B_i`
%       and `k`-th polynomial `P_k` for `B_i`  where k \in {0,1,2,3}
function Bx = Bp(i, k, x)
    global m;
    h = 1/m;

    if i == -2
        switch k
        case 0
            Bx = (21*x^2)/(4*h^3) - (9*x)/(h^2) + 3/h;
        case 1
            Bx = -(3*(2*h-x)^2)/(4*h^3);
        otherwise
            warning('Bp: k ~\in {0,1} for B_{-2}\n');
        end
```

```
        return;
    end

    if i == -1
        switch k
        case 0
            Bx =  -x*(11*x-12*h)/(4*h^3);
        case 1
            Bx = 9/(2*h) - (6*x)/(h^2) + (7*x^2)/(4*h^3);
        case 2
            Bx = -(3*h-x)^2/(2*h^3);
        otherwise
            warning('Bp: k ~\in {0,1,2} for B_{-1)\n');
        end
        return;
    end

    if  (i == m-1 && any(k == [1 2 3])) || ...
        (i == m-2 && any(k == [2 3])) || ...
        (i == m-3 &&        k == 3)
        warning('Bp: k not right value for for B_{m-1,m-2,m-3}\n');
        return;
    end

    assert((i >= 0 && i <= m-1),'Bp: invalid basis i');
    l = i*h;
    r = (i+4)*h;

    switch k
    case 0
        Bx =  (x-l)^2/(2*h^3);
    case 1
        Bx = -2/h + 4*(x-l)/h^2 - 3*(x-l)^2/(2*h^3);
    case 2
        Bx = 10/h - 8*(x-l)/h^2 + 3*(x-l)^2/(2*h^3);
    case 3
        Bx =  -(r-x)^2/(2*h^3);
    otherwise
        warning('Bp: k ~\in {0,1,2,3} for B_i\n');
    end
end
```

Problem 3 code

```
clear all;
global m grid;

logit = @(x) log(x/(1-x));
abs_logit = @(x) abs(logit(x));
abs_logit_scaled = @(x) 0.1*abs(logit(x));

% ms = [9];
ms = [9 19 39 79 159 319 639];

fprintf('m\tmax_error\tratio\n');
for iter = 1:size(ms,2)
    m = ms(iter);
    h = 1/(m+1);
    grid = 0:h:1;
    grid = error_equidistribution(abs_logit_scaled, grid);

    [A, b] = assembly();
    c = A \ b;

    e = arrayfun(@(i) abs(y(grid(i+1)) - (c(i)+1)), ...
        1:m);
    max_e = max(e);

    plot(grid(2:end-1), arrayfun(@(i) y(grid(i+1)), 1:m), '-', ...
         grid(2:end-1), arrayfun(@(i) (c(i)+1), 1:m), '--');

    ratio = 0;
    if iter ~= 1
        ratio = max_e / pre_max_e;
    end
    fprintf('%d\t%.10f\t%.10f\n', m, max_e, ratio);
    pre_max_e = max_e;
end


function [A, b] = assembly()
    global m grid;

    A = sparse(m,m);
    b = zeros(m,1);

    % formula over [x_{i}, x_{i+1}] for lhs/rhs
    %         note `i` is 0-indexed
    akl = @(i,k,l) ...
        gq(@(x) Bp(k,x)*Bp(l,x) + (10^4)*B(k,x)*B(l,x), ...
            grid(i        +1), ...
            grid(i+1      +1), 5);

    bk = @(i,k) ...
        -(10^4)* gq(@(x) B(k,x), ...
            grid(i        +1), ...
```

```
            grid(i+1      +1), 5);

    for k = 1:m
        for l = 1:m
            if l == k
                A(k,l) = akl(k-1,k,l) + akl(k,k,l);
            elseif l == k-1
                A(k,l) = akl(l,k,l);
            elseif l == k+1
                A(k,l) = akl(k,k,l);
            end
        end
        b(k) = bk(k-1,k) + bk(k,k);
    end
end


% basis B_k at `x`
%       where `k` is 0-indexed
function Bx = B(k,x)
    global grid;

    l = grid(k-1     +1);
    c = grid(k       +1);
    r = grid(k+1     +1);

    if x >= l && x <= c
        Bx = (x-l)/(c-l);
    elseif x >= c && x <= r
        Bx = (r-x)/(r-c);
    else
        warning('B: out of support');
    end
end


% basis B_k' at `x`
%       where `k` is 0-indexed
function Bx = Bp(k,x)
    global grid;

    l = grid(k-1     +1);
    c = grid(k       +1);
    r = grid(k+1     +1);

    if x >= l && x <= c
        Bx = 1/(c-l);
    elseif x >= c && x <= r
        Bx = 1/(c-r);
    else
        warning('B: out of support');
    end
end
```

```
function yx = y(x)
    c_1 = (1-exp(-100))/(exp(100)-exp(-100));
    c_2 = (exp(100)-1)/(exp(100)-exp(-100));
    yx = c_1*exp(100*x) + c_2*exp(-100*x);
end

function yx = yp(x)
    yx = (100*exp(-100*x)*(exp(200*x)-exp(100)))/(exp(100)+1);
end

function yx = ypp(x)
    yx = (10000*exp(-100*x)*(exp(200*x)+exp(100)))/(exp(100)+1);
end

function phix = varphi0(x)
    phix = 1;
end


% Given monitor function that approximates erorr
%       and original grid, returns a new grid with equal error distribution
%       reference: https://www.math.uci.edu/~chenlong/226/Ch4AFEM.pdf
function xs = error_equidistribution(M, xs)
    cdf = arrayfun(@(i) gq(M, xs(i), xs(i+1), 5), 1:(max(size(xs))-1));
    cdf = [0, cumsum(cdf)];
    cdf = cdf/cdf(end);
    ys = 0:1/(length(xs)-1):1;
    [cdf, index] = unique(cdf);
    xs = interp1(cdf,xs(index),ys);
end

function explore_monitor()
    xs1 = 0:1/100:1;
    logit = @(x) log(x/(1-x));
    abs_logit = @(x) abs(logit(x));
    xs2 = error_equidistribution(abs_logit, xs1);
    plot(xs1, arrayfun(@y,xs1), ...
     xs1, arrayfun(abs_logit,xs1), ...
     xs2, zeros(size(xs2)), 'o');
end
```

Problem 4 code

```
clear all;
global m;

% ms = [9];
ms = [9 19 39 79 159 319 639];

fprintf('m\tmax_error\tratio\n');
for iter = 1:size(ms,2)
    m = ms(iter);
    h = 1/(m+1);

    [A, b] = assembly();
    c = A \ b;

    actual = zeros(m,m);
    expected = zeros(m,m);
    for p = 1:m^2
        [i,j] = p2ij(p);
        actual(i,j) = c(p);
        expected(i,j) = u(i*h,j*h);
    end

    e = arrayfun(@(x) abs(x), expected-actual);
    max_e = max(e,[],'all');

    % [X,Y] = meshgrid(h:h:1-h);
    % mesh(X,Y,e);

    ratio = 0;
    if iter ~= 1
        ratio = max_e / pre_max_e;
    end
    fprintf('%d\t%.10f\t%.10f\n', m, max_e, ratio);
    pre_max_e = max_e;
end

function [A, b] = assembly()
    global m;
    h = 1/(m+1);

    % construct `A`

    % 3 integral values for `a_{kl}`
    dfull   = zeros(m^2,1) + 8/3;
    dside   = zeros(m^2,1) - 1/3;
    dcorner = zeros(m^2,1) - 1/3;

    % pad zeros at certain locations -> block diagonal
    fillzerosat = m:m:m^2;
    b1 = dside;
    b1(fillzerosat) = 0;
    b2 = dcorner;
```

```
    b2(fillzerosat) = 0;

    b = [-m-1,-m,-m+1,      -1,0,1,          m-1,m,m+1];
    B = [b2,dside,b2,    b1,dfull,b1,    b2,dside,b2];
    B(:,3) = flip(B(:,1));
    B(:,6) = flip(B(:,4));
    B(:,9) = flip(B(:,7));
    A = spdiags(B,b,m^2,m^2);

    % construct `b`

    b = zeros(m^2,1);

    % part of (half) the function for `bkl`
    bkl_ = @(k) ...
        gq(@(x) 32*x*(1-x)*hat(k,x), (k-1)*h, k*h, 5) + ...
        gq(@(x) 32*x*(1-x)*hat(k,x), k*h, (k+1)*h, 5);

    % evaluate the integral for `b_{k,l}`
    bkl = @(k,l) ...
        h*(bkl_(k) + bkl_(l));

    for j = 1:m
        for i = 1:m
            p = ij2p(i,j);
            b(p) = bkl(i,j);
        end
    end
end

function [i,j] = p2ij(p)
    global m;
    j = floor((p-1) / m) + 1;
    i = mod(p-1, m) + 1;
end
function p = ij2p(i,j)
    global m;
    p = i + (j-1)*m;
end

function fx = hat(k, x)
    global m;
    h = 1/(m+1);

    l = (k-1)*h;
    c = k*h;
    r = (k+1)*h;

    if x >= l && x <= c
        fx = 1-k+x/h;
    elseif x >= c && x <= r
        fx = 1+k-x/h;
    else
        % warning('hat function should not reach here ...');
```

```
        fx = 0;
    end
end

function fxy = f(x,y)
    fxy = 32*x*(1-x) + 32*y*(1-y);
end

function uxy = u(x,y)
    uxy = 16*x*(1-x)*y*(1-y);
end
```