Assignment 1

Ludwig Tranheden

TMA372

2016/02/11

Contents

1 Problem 1

The description states that we are supposed to write a program which computes the contionus G(2) finite element approximation for the following BVP problem.

$$\begin{cases} u''(x) = f(x) \text{ in } (0,1), \\ u(0) = u(1) = 0. \end{cases}$$

Given that the user supplies the data vector b, that is f(x).

We tackle this problem by first dividing the intervall of interest into smaller subintervals. Let $\tau_h = \{0 = x_0 < x_1 < ... < x_M < x_M + 1 = 1\}$ be a partion of the intervall of the problem and define $I_k := [x_{\text{k-1}}, x_k]$ to be the different subintervalls. Next we restrict the length of each intervall to be constant because it simplifies things, that is we construct a uniform mesh: $h_k = x_k - x_{\text{k-1}} = h$. Restricting the approximate function and the the test function in the FEM-formulation to be a part of $V_h^{(0)}$ the formulation simplifies to:

$$\int_0^1 u'(x)v'(x)\,\mathrm{d}x = \int_0^1 f(x)v(x)\,\mathrm{d}x \qquad \forall v \in V_h^{(0)}$$

Where $V_h^{(0)}$ in the space of piecewise quadratic continous functions being zero at 0 and 1. We will use the lagrange basis and in addition to the two points $x_{\text{k-1}}$ and x_k we will use the midpoints $x_{\text{k-1/2}} = (x_{\text{k-1}} + x_k)/2 = x_{\text{k-1}} + h/2$. The definitions is seen below:

$$\varphi_{\text{k-1/2}}(x) = \begin{cases} \frac{4(x_k - x)(x - x_{\text{k-1}})}{h^2}, & x \in I_k \\ 0, & \textit{otherwise} \end{cases}$$

Where k = 1,2,..,M+1.

$$\varphi_{\text{k}}(x) = \begin{cases} \frac{2(x - x_{\text{k+1/2}})(x - x_{\text{k+1}})}{h^2}, & x \in I_{\text{k+1}} \\ \frac{2(x - x_{\text{k-1/2}})(x - x_{\text{k-1}})}{h^2}, & x \in I_{\text{k}} \\ 0, & \textit{otherwise} \end{cases}$$

Where k = 1,2,...,M.

This will be the basis of $V_h^{(0)}$ and yield functions with the behavior illustrated in figure 1.

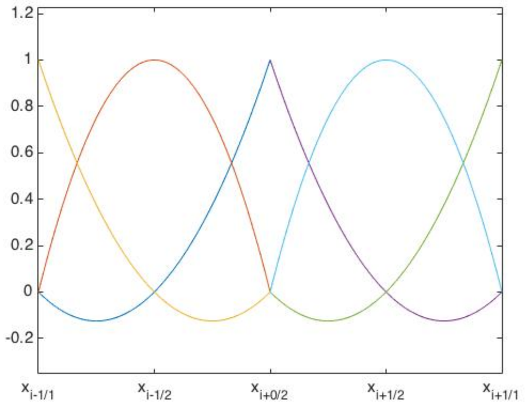

Figure 1: Basis functions

This means that the approximate solution to u(x) can be defined as:

$$\tilde{u}(x) = \sum_{k=1}^{2\text{M}+1} \xi_{\text{k}/2}(x)\varphi_{\text{k}/2}(x)$$

And we end up with a system of equations $A\xi = b$ to solve for the 2M+1 unknown coefficients of $\tilde{u}(x)$. I.e we end in fact up with 2M+1 nodes. To move on we calculate the derivatives for our basis functions.

$$\varphi'_{\text{k-1/2}}(x) = \begin{cases} \frac{4(x_k + x_{\text{k-1}} - 4x)}{h^2}, & x \in I_k \\ 0, & otherwise \end{cases}$$

$$\varphi'_{\text{k}}(x) = \begin{cases} \frac{2(2x - x_{\text{k+1/2}} - x_{\text{k+1}})}{h^2}, & x \in I_{\text{k+1}} \\ \frac{2(2x - x_{\text{k-1/2}} - x_{\text{k-1}})}{h^2}, & x \in I_{\text{k}} \\ 0, & otherwise \end{cases}$$

Then we can express the load vector elements as:

$$b_k = \int_0^1 f(x)\varphi_k dx \qquad \text{for i = 1/2,1,...,M+1/2}$$

and the entries af the A-matrix as:

$$a_{\text{k,l}} = \int_0^1 \varphi'_k \varphi'_l dx \qquad \text{for k,l = 1/2,1,...,M+1/2}$$

For the load vector elements i use a my own implemented function called "generateB" who uses two other functions called "Mysimpson" and "Generatebasis" all seen in subsection 4.4. The entries of the matrix will in most cases be equal to zero because how we have defined our basis. Computing the non-zero elements are not very hard but demends alot of time and effort. Choosing a uniform mesh simplifies things considerably. The nonzero combinations are
for k = i + 1/2 (i an integer): (k, i), (k, i + 1/2), (k, i + 1)
And for k = i: (k, i - 1), (k, i - 1/2), (k, i), (k, i + 1/2), (k, i + 1).
Implementing the filling of A we imagine us moving across the diagonal of A filling the elements around us.

Now we are left with solving the equation-system $A\xi = b$ for the 2M+1 unknown coefficients of $\tilde{u}(x)$. Since the matrix is not diagonally dominant we can't use the iteration method's in the lecture notes to solve the system. Instead we simply use LU factorization. The result using 21 nodal points can be seen in figure 2.

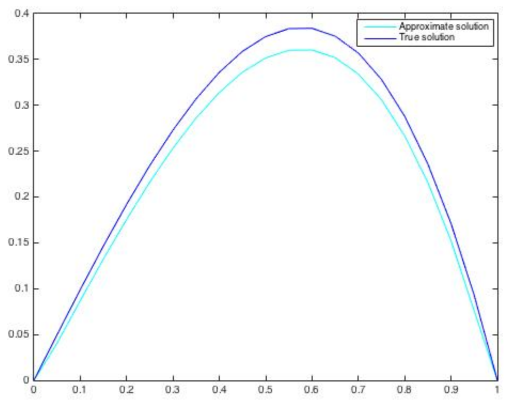

Figure 2: Approximate solution versus the true solution

2 Problem 2

Given the boundary value problem below we are supposed to solve some tasks regardning the problem using cG(1).

$$\begin{cases} \epsilon u''(x) + u'(x) = f(x) \text{ in } (0,1), \\ u(0) = 1 \qquad u(1) = 0. \end{cases}$$

In the same fashion as in the previous problem we create a uniform partion of [0,1] and define subintervalls I_k, with the same length. We then define the trialspace to be $H^1_{1,0}$ and the testspace as H^1_0. Writing the problem in variational form yields:

$$\int_0^1 \epsilon u'(x)v'(x) + u'(x)v(x)dx = \epsilon u'(0)v(0) = 0 \qquad \forall v \in H^1_0$$

To move on we use the partion τ_h and define two new function spaces.

$$V_h := \big\{w_h : w_h \; is\, piecewise\; linear,\; continous\; on\; \tau_h, w_h(0) = 1, w_h(1) = 0\big\}$$

and

$$V_h^0 := \big\{w_h : w_h \; is\, piecewise\; linear,\; continous\; on\; \tau_h, w_h(0) = 0, w_h(1) = 0\big\}$$

The basis for V_h is the hat functions φ_i i = 0,1...,n. The basis for V_h^0 is the same but without the first half-hat function φ_0. This is because of the nonzero boundary value at 0. The discrete variational formulation then end up to be the following with $u_h \in V_h$.

$$\int_0^1 \epsilon u_h' v' + u_h' v dx = 0 \qquad \forall v \in V_h^0$$

Where u_h will be of the form

$$u_h = \varphi_0(x) + \sum_{j=1}^{n} \xi_j \varphi_j(x)$$

Where the hat function are given by.

$$\varphi_0(x) = \begin{cases} \frac{h-x}{h}, & 0 \le x \le h \\ 0, & otherwise \end{cases}$$

and

$$\varphi_{\mathrm{j}}(x) = \begin{cases} \frac{x-x_{j-1}}{h}, & x \in I_{\mathrm{j}} \\ \frac{x_{j+1}-x}{h}, & x \in I_{\mathrm{j+1}} \\ 0, & otherwise \end{cases}$$

Inserting our expression for u_h into the discrete variational formulation and choosing $v = \varphi_i(x),\ i = 1, 2, ..., n$ we get.

$$\sum_{j=1}^{n} \Big(\epsilon \int_0^1 \varphi_j' \varphi_i' dx + \int_0^1 \varphi_j' \varphi_i dx\Big)\xi_j = -\Big(\epsilon \int_0^1 \varphi_0' \varphi_i' dx + \int_0^1 \varphi_0' \varphi_i dx\Big)$$

And this leads to the equation system of the form $A\xi = b$.

Implementing the solution of the BVP in matlab is basicly the same as the previous problem. The matrix entries can be solved rather easily but are quite timeconsuming. I tried to use my "Mysimpson-method" to do the integrals.

When compared to the analytical answers i had some problem along the diagonal of the convection matrix, an error of $\approx 10^{-16}$ but just rounding the numbers down a few decimals did the trick. I do realize that this makes the efficiency slightly worse then just entering the analytical values. To solve the equation system i implemented a Gauss-seidel method called "GaussSeidel" seen in subsection 4.4. The program solves the system with gauss seidel if and only if the matrix is diagonally dominant.

The analytical solution to the problem is

$$u(x) = \frac{e^{\frac{1}{\epsilon}} - e^{\frac{x}{\epsilon}}}{e^{\frac{1}{\epsilon}} - 1}$$

Below are the approximate solution plotted against the analytical solution for $\epsilon = 0.01$, M=10,11,100.

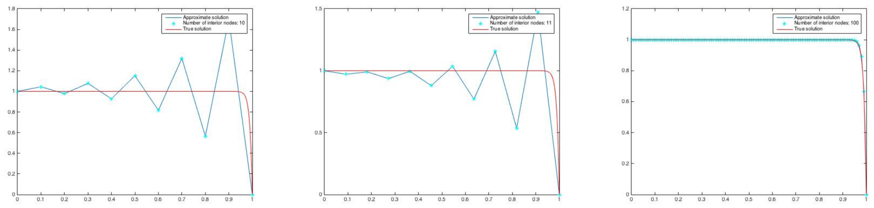

Figure 3: 10,11 and 100 nodes

We see that 10 or 11 nodes is insufficient to approximate the solution in a good way. But when you use 100 nodes the approximate solution is a very good fit to the analytical solution.

3 Problem 3

In this problem we are given a initial value problem to solve.

$$\begin{cases} \dot{u}(t) + 4u(t) = f(t) \text{ for } 0 < t \leq T, \\ u(0) = u_0. \end{cases}$$

Where $u_0 = 1$ and $f(t) = t^2$. We are going to use cG(1), i.e the trial functions are piecewise linear and continous and the the test functions are piecewise constant and discontinous. For cG(1) we can restrict v to $v \equiv 1$. Since $U(t)$ is a linear function on we can express it as

$$U(t) = U(0)\frac{T-t}{T} + U(T)\frac{t}{T}$$

on an intervall. We then for a uniform partion τ_k of the intervall $[0, T]$ define subintervals $I_k = (t_{k-1}, t_k]$ for k = 1,2,...,n. The variational formulation on a subintervall is.

$$U(t_k) - U(t_{k-1}) + \int_{t_{n-1}}^{t_n} 4U(t)dt = \int_{t_{n-1}}^{t_n} t^2 dt$$

Integrating the variational formulation and using the definition $U_k := U(t_k)$ and the uniform stepsize (for area) we get the following expression where h is the stepsize.

$$U_k - U_{k-1} + 2(U_{k-1} + U_k)h = \int_{t_{n-1}}^{t_n} t^2 dt$$

Integrating the right hand side and solving for U_k yields.

$$U_k = \frac{\left(\frac{t_k^3 - t_{k-1}^3}{3} - U_{k-1}(2h-1)\right)}{(1+2h)}$$

Starting at U_1, using the value at U_0 which is a value we know from the problem we can iterate our way to the endpoint value U_n. Having both U_{k-1} and U_k we have $U(t)$ for $t \in I_k$ using the linearity. So doing this to all the subintervalls we get U(t) of the whole intervall.

Comparing the approximate solution in matlab to the analytic solution $u(t) = \frac{1}{32}(8t^2 - 4t + 31e^{-4t} + 1)$ using T=1 and 10 steps we get the result shown in figure 4.

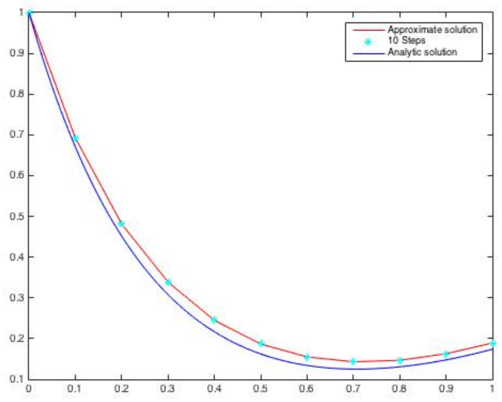

Figure 4: Approximate solution versus the true solution

Moving on to the stepsize, in our problem any stepsize will guarantee a solution exists because $a = 4$ is a constant positive number. The general condition on the stepsize for a general a is the following:

$$\int_{I_k} \mid a(t) \mid dt < 1$$

4 Appendix

4.1 Code: Problem 1

Listing 1: Code for problem 1

```
n = 10;
f = @(x) 6*x; %User Supplied function.
B = zeros(1, 2*n+1);

Nodes = linspace(0,1,length(B)); %Node positions
B = generateB(f,length(B),Nodes)'; %Generate loadvector

h = 2/((length(B)-1)/2);
A = zeros(length(B),length(B));

%Generate A Matrix.
%One step in matrix = 1/2 to the index of basisfunctions.
for i=2:2:length(B)
    A(i,i) = 14/(3*h);
    A(i,i-1) = -8/(3*h);
    A(i,i+1) = -8/(3*h);
    A(i-1,i) = -8/(3*h);
    A(i+1,i) = -8/(3*h);
    A(i-1,i-1) = 16/(3*h);
    if i < length(B) - 1    %To not exceed size
        A(i,i+2) = 1/(3*h);
        A(i+2,i) = 1/(3*h);
    end
end

%Force boundary values.
A(1,1) = realmax;
A(length(B),length(B)) = realmax;

x = all((2*abs(diag(A))) - sum(abs(A),2)>=0); %Diagonally dominant? It's not, use LU
[L,X] = lu(A); % LU decomposition
U = X\(L\B); %Solve
```

4.2 Code: Problem 2

Listing 2: Code for problem 2

```
eps = 0.01;
M = 11;
h = 1/M;
x=0:h:1;
n=length(x);

S = zeros(n,n);
C = zeros(n,n);
B = zeros(n,1);

for i=1:1:length(x)-1
```

```
    if i < 2
        S(1,1) = Mysimpson(0,h,@(x) x*0 + 1/h^2,0.001);
        S(2,1) = Mysimpson(0,h,@(x) x*0 - 1/h^2,0.001);
        S(1,2) = Mysimpson(0,h,@(x) x*0 - 1/h^2,0.001);
    else
        S(i,i) = Mysimpson(x(i-1),x(i),@(x) x*0 + 1/h^2,0.001) +
        Mysimpson(x(i),x(i+1),@(x) x*0 + 1/h^2,0.001);
        S(i+1,i) = Mysimpson(x(i),x(i+1),@(x) x*0 - 1/h^2,0.001);
        S(i,i+1) = Mysimpson(x(i),x(i+1),@(x) x*0 - 1/h^2,0.001);
        if i < length(x)-1
            S(i+1,i+1) = Mysimpson(x(i),x(i+1),@(x) x*0 + 1/h^2,0.001) +
            Mysimpson(x(i+1),x(i+2),@(x) x*0 + 1/h^2,0.001);
        else
            S(length(x),length(x)) = Mysimpson(0,h,@(x) x*0 + 1/h^2,0.001) +
            Mysimpson(0,h,@(x) x*0 + 1/h^2,0.001);
            S = S*eps;
        end
    end

    if i < 2
        C(1,2) = Mysimpson(0,h,@(t)(1/h)*(x(i+1)- t)/h,0.001);
        C(2,1) = Mysimpson(0,h,@(t)(-1/h)*(t-x(i))/h,0.001);
    else
        C(i,i) = round((Mysimpson(x(i-1),x(i),@(t) ((t-x(i-1))/h)*(1/h),0.001) +
        Mysimpson(x(i),x(i+1),@(t) ((x(i+1)-t)/h)*(-1/h),0.001)),-100000);
        C(i+1, i) = Mysimpson(x(i),x(i+1),@(t)(-1/h)*(t-x(i))/h,0.001);
        C(i,i+1)= Mysimpson(x(i),x(i+1),@(t)(1/h)*(x(i+1)- t)/h,0.001);
    end
end

%Will force boundaryconditions
A = C+S;
B(1,1) = realmax;
A(1,1) = realmax;
A(length(A),length(A)) = realmax;

if all((2*abs(diag(A))) - sum(abs(A),2)>=0) %Diagonally dominant (eps > 0.8)? Use Gaussseidel.
    U = GaussSeidel(A,B,1000)';
else
    [L,X] = lu(A); % LU decomposition
    U = X\(L\B); %Solve
end
```

4.3 Code: Problem 3

Listing 3: Code for problem 3

```
    T=1;
    n = 10;
    h = 1/(n+1);
    t = linspace(0,T,1/h);
    u = zeros(1,length(t));
    u(1) = 1;

    for i=2:length(u)
        u(i) = (((t(i)^3 - t(i-1)^3)/3 - (u(i-1).*(2*h-1)))/(1 + 2*h));
    end
```

4.4 Functions

Listing 4: Function to generate loadvector

```
 %f function
 %Size, number of nodes
 %T node vector
function B = generateB(f,size,T)
    B = zeros(1,size);
    for i = 2:size
        if mod(i,2) == 1 %If we are at a odd node
            f1 = @(x) f(x).*Generatebasis(x,T(i-1),T(i),3);
            B(i) = Mysimpson(T(i-1),T(i),f1,0.0001);
        else
            f1 = @(x) f(x).*Generatebasis(x,T(i-1),T(i),0);
            f2 = @(x) f(x).*Generatebasis(x,T(i),T(i+1),1);
            B(i) = Mysimpson(T(i-1),T(i),f1,0.0001) + Mysimpson(T(i),T(i+1),f2,0.0001);
        end
    end
end
```

Listing 5: Function to generate the basisfunctions

```
%If i=0 -> Whole indice phi type I_(i)
%Elseif i = 1 -> Whole indice phi type I_(i+1)
%Else -> Half indice phi
function y = Generatebasis(x,a,b,i) %x variable,intervall I_i=[a,b]
    h = b-a;
    cent = (a+b)/2;
    if i == 0
        y = (2*(x-cent).*(x-a))/(h^2);
    elseif i == 1
        y = (2*(x-cent).*(x-b))/(h^2);
    else
        y = (4*(b-x).*(x-a))/(h^2);
    end

end
```

Listing 6: Simpson numerical integration

```
% start - startingpoint of integral
% endd - endpoint of integral
% f - function to integrate
% step - stepsize between start and endd, determine accuracy
function S = Mysimpson(start,endd,f,step)
    S = 0;
    x = start:step:endd;
    for k=2:length(x)
        S = S + ((step/6)*(f(x(k-1))+4*f((x(k-1)+x(k))/2) + f(x(k))));
    end
```

```
end
```

Listing 7: Gauss Seidel

```
function U = GaussSeidel(A,B,N)
%Solves the system of equations AU=B with N iterations
%returns the coefficient in U

    x = ones(1,length(B)); %Intital values

        for ire=1:N                     %iterations
            for j=1:length(x)
                temp = 0;
                for i=1:length(x)
                    if j~=i
                        temp = temp + A(j,i)*x(i);
                    end
                end
                x(j) = (-1/A(j,j))*(temp-B(j)); %Next value x^(ire+1)_j
            end                                 %Use in next coefficient -> Faster convergence.
        end
        U = x;
end
```