TMA375 Partial differential equations
Assignment 1

Simon Sigurdhsson ssimon@student.chalmers.se

10th March 2011

8.35 — The second-degree continuous Galerkin method

We are tasked with implementing the second-degree continuous Galerkin method solving the problem (1) in MATLAB, given that the user supplies the load vector b.

$$\begin{cases} -u'' = f \\ u(0) = u(1) = 0 \end{cases} \tag{1}$$

Since the problem states that the user has to supply the load vector b, the problem is effectively reduced to calculating the stiffness matrix A and solving the system of equations that arises. Using the basis functions $\varphi_{i-\frac{1}{2}}$ and φ_i given in Eriksson et al. (1996, p. 200), also seen in (2) and (3), we can compute the values that will feature in the banded and symmetrical matrix A.

$$\varphi_{i-\frac{1}{2}} = \begin{cases} \frac{4(x_i - x)(x - x_{i-1})}{h_i^2}, & x \in I_i \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$\varphi_i = \begin{cases} \frac{2(x - x_{i-\frac{1}{2}})(x - x_{i-1})}{h_i^2}, & x \in I_i \\ \frac{2(x - x_{i+\frac{1}{2}})(x - x_{i+1})}{h_{i+1}^2}, & x \in I_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Also keep in mind that the interval I_i is defined as $I_i = [x_{i-1}, x_i]$. We now assume a uniform mesh, i.e. $h_i = h$, $\forall i$, and calculate the derivatives of these basis functions as seen in (4) and (5).

$$\varphi'_{i-\frac{1}{2}} = \begin{cases} \frac{4x_i + 4x_{i-1} - 8x}{h^2}, & x \in I_i \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$\varphi'_i = \begin{cases} \frac{4x - 2x_{i-1} - 2x_{i-\frac{1}{2}}}{h^2}, & x \in I_i \\ \frac{4x - 2x_{i+1} - 2x_{i+\frac{1}{2}}}{h^2}, & x \in I_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

This gives us all we need to calculate the entries of the stiffness matrix A. We already know that the matrix is banded with 5 non-zero diagonals (this can be verified by comparing the basis functions and what intervals they are defined on, and is explained in Eriksson et al. (1996, p. 201)) and that it is symmetrical. The entries are given by (6), and have to be calculated for six pairs (k is an integer): (k,k), (k,$k+\frac{1}{2}$), (k,$k+1$), ($k+\frac{1}{2}$,$k+\frac{1}{2}$), ($k+\frac{1}{2}$,$k+1$), ($k+\frac{1}{2}$,$k+\frac{3}{2}$).

$$a_{ij} = \int_0^1 \varphi_i' \varphi_j' \, \mathrm{d}x \tag{6}$$

Letting Mathematica do the tedious grunt work, we arrive at the values seen in (7). Values not actually calculated by rather inferred by the symmetry and bandedness of our matrix are shown as "−". Note that the outermost diagonals aren't entirely non-zero — this is because the half-step basis functions $\varphi_{i-\frac{1}{2}}$ and $\varphi_{i+\frac{1}{2}}$ don't overlap.

$$\begin{array}{c} \\ k \\ k+\frac{1}{2} \\ k+1 \\ k+\frac{3}{3} \end{array} \begin{array}{c} \begin{array}{cccc} k & k+\frac{1}{2} & k+1 & k+\frac{3}{2} \end{array} \\ \begin{pmatrix} \frac{14}{3h} & -\frac{8}{3h} & \frac{1}{3h} & - \\ - & \frac{16}{3h} & -\frac{8}{3h} & 0 \\ - & - & \frac{14}{3h} & -\frac{8}{3h} \\ - & - & - & \frac{16}{3h} \end{pmatrix} \end{array} \tag{7}$$

We can now go about implementing our MATLAB function, which will solve the problem given in (1) given input data b from the user. We begin by defining the two basis functions given by (2) and (3) as MATLAB functions:

Listing 1: cG(2) basis functions implemented in MATLAB

```
function y=phi_half(x, x0, h)
  y = 4*(x0 - x).*(x - (x0 + h))/(h^2);
end

function y=phi_whole(x, x0, h)
  y = 2*(x - (x0 + h/2)).*(x - x0)/(h^2) .* ((x - x0) <= h);
  y = y + 2*(x - (x0 + 3*h/2)).*(x - (x0 + 2*h))/(h^2) .* ((x - x0) > h);
end
```

The functions given in listing 1 are relatively useless when setting up the stiffness matrix A, but highly relevant when the user has to compute b or plot the solution.

What we have to construct is the function (let's call it `cG2`) that creates the stiffness matrix and solves the system of equations that results. It also has to check that the input data has $2N + 1$ nodes (i.e. and odd number of elements) and calculate a suitable step size h. To summarize, our function will:

1. Check that input data b has $2N + 1$ elements
2. Calculate the step size h given the input data b
3. Create a $2N + 1 \times 2N + 1$ stiffness matrix A
4. Solve the system of equations $Ax = b$

The first two steps are easy. Assuming our input load vector is named `b` (a *column* vector) and out output variable is called `U`, we simply use the `mod` function to perform the first task and simple arithmetic to perform the second one:

Listing 2: Input validation and step size calculation

```
function U = cG2( b )
  % Check that input data b has 2N+1 elements
  if(mod(length(b), 2) ~= 1)
    U = []; % Return an empty result if input data is invalid
    return
  end
  % Calculate the step size h given the input data b
  h = 1/((length(b)-1)/2);
```

The step size is given by $1/N$, where N is the number of intervals we've split the range (0,1) into. Since N intervals imply $N+1$ nodes, and by extension $2N$ intervals imply $2N+1$ nodes (incidentally the number of nodes we have in b), the step length is simply given by $h = (\text{len}(b) - 1)/2$.

Continuing on, we must now create the stiffness matrix. We do this by first creating an empty matrix of the correct size, then looping over one of its indices to fill it with the relevant information. After we do this, we set the first and last element of the diagonal to a very large number, to enforce the boundary conditions given in (1). Solving the system is then done using the simple MATLAB backslash operator; `U = A\b`.

To save ourselves some trouble, we only visit the *even* indices. This means we won't have to care about wether we're visiting a half node or a whole node.

Each step, we visit the even diagonal element we're at and the odd diagonal element preceding it. We populate these, as well as the two closest elements to the right and underneath them:

Listing 3: Setting up the stiffness matrix A and solving the system

```
  % Set up the stiffness matrix
  A = zeros(length(b), length(b));
  for i=2:2:length(b)
    A(i-1,i-1) = 16/(3*h); % Half-index diagonal element
    A(i-1,i) = -8/(3*h);   % Shift right by one
                           % Shift right by two (element is 0)
    A(i,i-1) = -8/(3*h);   % Shift down by one
                           % Shift down by two (element is 0)

    A(i, i) = 14/(3*h);    % Whole-index diagonal element
    A(i, i+1) = -8/(3*h);  % Shift right by one
    if(i < length(b)-1) A(i, i+2) = 1/(3*h); end % Shift right by two
    A(i+1, i) = -8/(3*h);  % Shift down by one
    if(i < length(b)-1) A(i+2, i) = 1/(3*h); end % Shift down by two
  end
  % Enforce boundary conditions
  A(1,1) = 1000;
  A(end,end) = 1000;
  % Solve system
  U = A\b;
end
```

Remember, adding one to the index in this matrix is equivalent to adding 1/2 to the index of the correspoinding basis function. Note that we also safeguard for out-of-bounds errors on

the elements furthest from the diagonal. Since we don't fill the last diagonal element (since we're going to override that value), we don't have to care about the elements directly adjacent to the diagonal.

We enforce the boundary conditions by setting the first and last diagonal element to a very large value. This weighs the edge values so that they are almost guaranteed to stay at 0.

To be able to test this using $f = 6x$ with the known solution $u = x - x^3$, we need to be able to calculate the load vector as well as plot the solution. Plotting the solution is not complex; we give each sub-interval, say, 10 nodes and calculate the values of our basis functions at these point. We then multiply by the correct value from the solution value, and add half nodes and whole nodes to get the actual value of the solution at that point:

Listing 4: Plotting the solution using the basis function implementation

```
function plot_cG2( U )
  N = (length(U)-1)/2;
  h = 1/N;
  x = linspace(0, 1, 10*2*N+1);
  u = zeros([1 10*2*N+1]);

  for i=1:N
    sxh = 10*2*(i-1)+1;
    exh = 10*2*(i+0)+1;
    u(sxh:exh) = u(sxh:exh) + phi_half(x(sxh:exh), x(sxh), h) * U(2*i-1);
  end
  for i=1:(N-1)
    sxw = 10*2*(i-1)+1;
    exw = 10*2*(i+1)+1;
    u(sxw:exw) = u(sxw:exw) + phi_whole(x(sxw:exw), x(sxw), h) * U(2*i);
  end

  plot(x, u);
end
```

We also have to compute the load vector b. This is done using (8), and with $f = 6x$ this is a simple integral.

$$b_i = \int_0^1 f\varphi_i \,\mathrm{d}x \tag{8}$$

This gives us that $b_{i-1/2} = 2h(x_i + x_{i-1})$ and $b_i = 2hx_i$. Thus, we can easily set up a $2N+1$ load vector using the following MATLAB code:

Listing 5: Setting up the load vector for $f = 6x$

```
x = linspace(0, 1, 2*N+1)';
b = zeros([1 2*N+1])'; % note that h = 1/N
b(3:2:end) = 2/N*(x(3:2:end)+x(1:2:end-2));
b(2:2:end-2) = 2/N*x(2:2:end-2);
```

Testing it using $N = 100$ and plotting against the known solution yields the result seen in figure 1. We can draw the conclusion that the cG(2) method is very accurate.

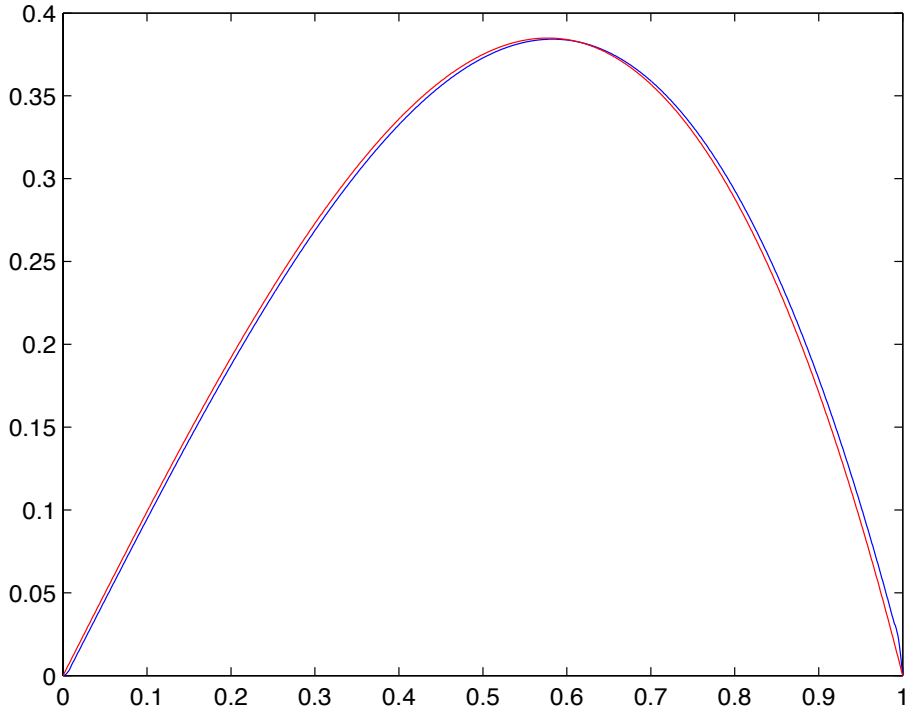


Figure 1: The problem FEM solution (blue) compared to the actual solution (red)

18.6 — Difficult problems for the cG(1) method

We will now investigate the problem (9) using the first-degree continuous Galerkin method, and we'll se that this problem is hard to solve using the Galerkin method. The problem has a known solution (10).

$$\begin{cases} -\epsilon u'' + u' = 0 \\ u(0) = 1 \\ u(1) = 0 \end{cases} \tag{9}$$

$$u(x) = \frac{e^{\frac{1}{\epsilon}} - e^{\frac{x}{\epsilon}}}{e^{\frac{1}{\epsilon}} - 1} \tag{10}$$

Our first issue is to find basis functions and set up the variational formulation of the problem. We choose the hat basis functions (11), but add a first basis function φ_0 that consists of the decreasing part of the hat function only (12).

$$\varphi_i = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}}, & x_{i-1} \leq x \leq x_i \\ \frac{x_{i+1} - x}{x_{i+1} - x_i}, & x_i \leq x \leq x_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

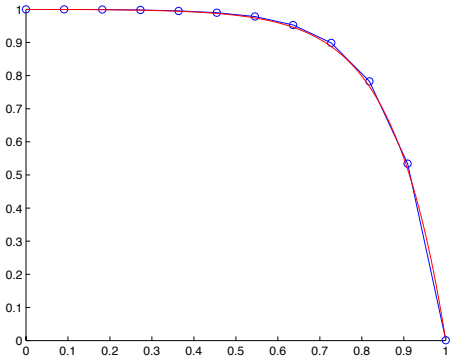
$$\varphi_0 = \begin{cases} \frac{x_1 - x}{x_i - x_0}, & 0 \leq x \leq x_1 \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

The variational formulation is obtained by multiplying by a test function and integrating as usual, and after partial integration a fairly simple relation is found:

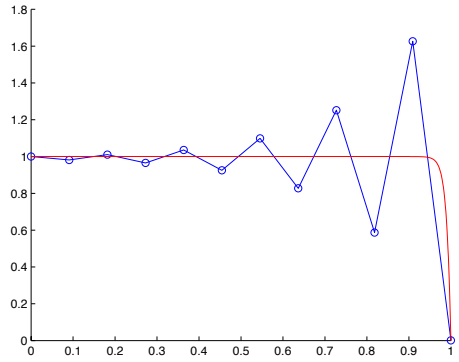
$$\int_0^1 \epsilon u'v' + u'v \, \mathrm{d}x = \epsilon u'(0) \tag{13}$$

Note that the right-hand side of this equation will be equal to 0 for all the hat basis functions, and equal to $\frac{\epsilon}{h}$ for $v = \varphi_0$. We can also calculate the elements of the stiffness matrix. With a uniform step size h, we have $A_{i,i} = \frac{2\epsilon}{h}$ and $A_{i,i+1} = -\frac{\epsilon}{h} + \frac{1}{2}$, $A_{i+1,i} = -\frac{\epsilon}{h} - \frac{1}{2}$ for all i except 0, where we have $A_{0,0} = \frac{\epsilon}{h} - \frac{1}{2}$ and $A_{0,1}$, $A_{1,0}$ as above. This, along with enforcing boundary conditions by setting the first and last diagonal elements to a large number L, results in a matrix problem as seen in (14).

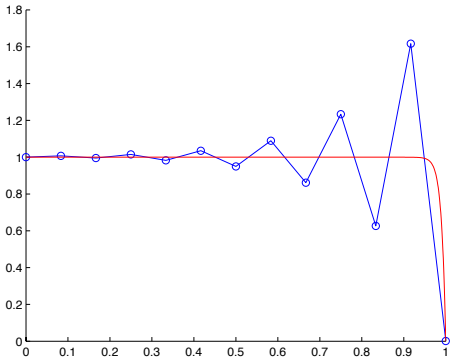
$$\begin{pmatrix} L & -\frac{\epsilon}{h} + \frac{1}{2} & 0 & 0 & 0 & 0 \\ -\frac{\epsilon}{h} - \frac{1}{2} & \frac{2\epsilon}{h} & -\frac{\epsilon}{h} + \frac{1}{2} & 0 & 0 & 0 \\ 0 & -\frac{\epsilon}{h} - \frac{1}{2} & \frac{2\epsilon}{h} & -\frac{\epsilon}{h} + \frac{1}{2} & 0 & 0 \\ 0 & 0 & -\frac{\epsilon}{h} - \frac{1}{2} & \frac{2\epsilon}{h} & -\frac{\epsilon}{h} + \frac{1}{2} & 0 \\ 0 & 0 & 0 & -\frac{\epsilon}{h} - \frac{1}{2} & \frac{2\epsilon}{h} & -\frac{\epsilon}{h} + \frac{1}{2} \\ 0 & 0 & 0 & 0 & -\frac{\epsilon}{h} - \frac{1}{2} & L \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} \frac{\epsilon}{h} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{14}$$



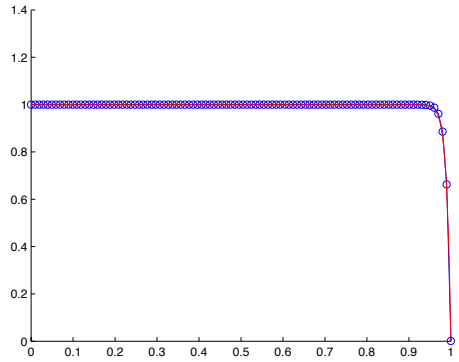
(a) The problem solved for $\epsilon = 0.125$ and $M = 10$



(b) The problem solved for $\epsilon = 0.01$ and $M = 10$



(c) The problem solved for $\epsilon = 0.01$ and $M = 11$



(d) The problem solved for $\epsilon = 0.01$ and $M = 100$

Figure 2: The problem (9) solved using the cG(1) method

Adapting the cG(1) MATLAB code published on the course website to solve this problem is fairly trivial and not covered here. We can see that the cG(1) method is fairly accurate for large ϵ (figure 2a), but that the solution is amost completely incorrect for smaller ϵ (figure 2b and 2c).

Upon further investigation, increasing the step size so that $M \geq 1/\epsilon$ produces a much better solution, as seen in figure 2d. For even smaller ϵ, this would mean increased computation time to achieve an accurate solution, which isn't desirable.

9.12 — Conditions on step-size for the cG(1) method

We now want to investigate the solution of another problem (15) for some function $a(t)$ to see what conditions the step size must satisfy to yield an acceptable solution. We only consider the case $a(t) = 4$.

$$\begin{cases} u'(t) + a(t)u(t) = f(t) & \text{for } 0 < t \leq T \\ u(0) = u_0 \end{cases} \tag{15}$$

Now, since this is an initial value problem rather than a boundary value problem, our cG(1) method doesn't look the same. This time the test functions v_i are discontinuous functions of degree 0 (i.e. piecewise linear functions). The trial functions u_i are replaced with a single, unknown continuous piecewise linear function U, the values of which we calculate at each node.

Regardless, the variational formulation (16) is still obtained by integrating and multiplying by a test function. Using our test functions v and trial function U we obtain the system of equations seen in (17) (using the notation $U(t_n) = U_n$.

$$\int_0^T \left(U'(t) + aU(t)\right) v(t)\,\mathrm{d}t = \int_0^T f(t)v(t)\,\mathrm{d}t \tag{16}$$

$$U_n - U_{n-1} + \int_{t_{n-1}}^{t_n} a(t)U(t)\,\mathrm{d}t = \int_{t_{n-1}}^{t_n} f(t)v(t)\,\mathrm{d}t \tag{17}$$

Using that $a(t) = 4$, and using the fact that v_i are pieceise linear, we can reduce this further (denoting the step length as h):

$$U_n - U_{n-1} + 2\left(U_{n-1} + U_n\right)h = \int_{t_{n-1}}^{t_n} f(t)\,\mathrm{d}t \tag{18}$$

We consider the case $u_0 = 1$ and $f(t) = t^2$, giving us that the RHS of () is $\frac{t_n^3 - t_{n-1}^3}{3}$. Also note that the system of equations given is defined for $n = 1,\ldots,N$, and that $U_0 = u_0$. Rearranging to isolate U_i on the LHS enables us to easily solve this using MATLAB:

Listing 6: The cG(1) method for the IVP given in (15), with $a(t) = 4$, $f(t) = t^2$ and $u_0 = 1$

```
N = 100;        % interior nodes
T = 5;          % final value T
h = T/(N+1);    % uniform step size
x = 0:h:T;      % node x values

U = zeros(size(x));
```

```
U(1) = 1;       % initial condition u_0 = 1
for i=2:length(x)
  U(i) = ((x(i)^3 - x(i-1)^3)/3 + U(i-1) - 2*h*U(i-1))/(1 + 2*h);
end

plot(x,U);
```

As stated by Eriksson et al. (1996, p. 211), we can easily determine conditions on the step size for a constant function $a(t)$ — in fact in our case a unique solution exists as long as $4h < 1$, i.e. $h < \frac{1}{4}$. The calculated solution for $T = 5$ and $h = 0.05$ can be seen in figure 3.

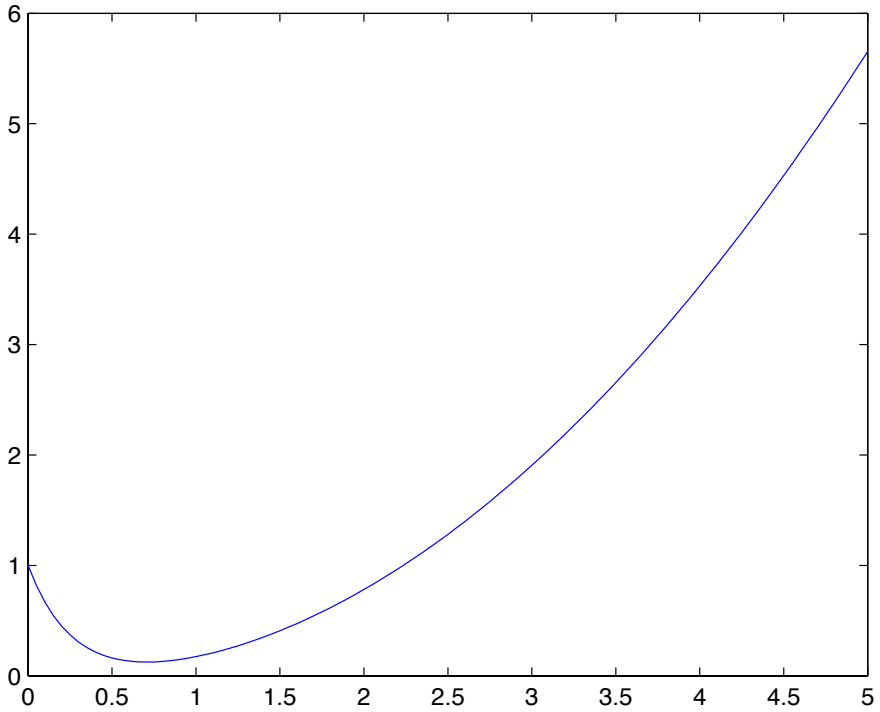


Figure 3: Solution of the problem (15) for $h = 0.05$, $0 < t \leq T$, $a(t) = 4$, $f(t) = t^2$ and $u_0 = 1$.

References

Eriksson, K., Estep, D., Hansbo, P. and Johnson, C. (1996) *Computational Differential Equations.* 2008. Lund: Studentlitteratur.