MATH 610 Programming Assignment 1 Hints

Jordan Hoffart

July 25, 2025

Problem 1

First we set up the variational formulation. Multiply by a smooth test function v such that $v(0) = v(l) = 0$ and integrate by parts to get

$$\int_0^l W'(x)v'(x)\,\mathrm{d}x + \frac{S}{D}\int_0^l W(x)v(x)\,\mathrm{d}x = \frac{q}{2D}\int_0^l x(l-x)v(x)\,\mathrm{d}x$$

Now we introduce a mesh of the domain by setting $h = l/N$ and $x_j = jh$ for $0 \le j \le N$. This partitions $(0,l)$ into N subintervals (x_j, x_{j+1}) for $0 \le j \le N-1$. The collection $\{(x_j, x_{j+1})\}_{j=0}^{N-1}$ is called a mesh of $(0,l)$, and its members are called elements or cells. We let V_h be the space of continuous piecewise linear functions with respect to this mesh that vanish at the boundary. This has the following basis functions

$$\varphi_j(x) = \begin{cases} (x - x_{j-1})/h & x \in (x_{j-1}, x_j) \\ (x_{j+1} - x)/h & x \in (x_j, x_{j+1}) \\ 0 & \text{otherwise} \end{cases}$$

for $1 \le j \le N-1$. Furthermore, any $W_h \in V_h$ satisfies

$$W_h(x) = \sum_{j=1}^{N-1} W_h(x_j)\varphi_j(x),$$

so that we can identify $W_h \in V_h$ with its vector $\vec{W}_h$ of coefficients, i.e. $(\vec{W}_h)_j = W_h(x_j)$. Now we seek a discrete solution $W_h \in V_h$ such that

$$\int_0^l W_h'(x)v_h'(x)\,\mathrm{d}x + \frac{S}{D}\int_0^l W_h(x)v_h(x)\,\mathrm{d}x = \frac{q}{2D}\int_0^l x(l-x)v_h(x)\,\mathrm{d}x$$

for all $v_h \in V_h$. Using the basis of V_h, this is equivalent to solving the following matrix-vector problem for the coefficient vector $\vec{W}_h$:

$$A_h\vec{W}_h = \vec{b}_h,$$

where

$$
\begin{aligned}
(A_h)_{i,j} &= \int_0^l \varphi_j'(x)\varphi_i'(x)\,\mathrm{d}x + \frac{S}{D}\int_0^l \varphi_j(x)\varphi_i(x)\,\mathrm{d}x,\\
(\vec{b}_h)_i &= \frac{q}{2D}\int_0^l x(l-x)\varphi_i(x)\,\mathrm{d}x
\end{aligned}
$$

To assemble A_h and $\vec{b}_h$, we loop over the elements of the mesh, compute local contributions, and assemble them into the global system. More precisely, we observe that each cell $K_j = (x_j, x_{j+1})$ has only two basis functions that are nonzero on it, φ_j and φ_{j+1}, except for cell K_0 or K_{N-1}, in which case there is only one basis function that is nonzero on it. Furthermore, if we let T_j denote the affine linear map from the reference element $(-1, 1)$ onto the physical element (x_j, x_{j+1}), i.e.

$$
T_j(\widehat{x}) = \frac{x_j + x_{j+1}}{2} + \frac{x_{j+1} - x_j}{2}\widehat{x} = \frac{h}{2}(2j + 1 + \widehat{x}),
$$

then we have that

$$
\begin{aligned}
\varphi_j|_{(x_j,x_{j+1})}(T_j(\widehat{x})) &= \frac{1-\widehat{x}}{2},\\
\varphi_{j+1}|_{(x_j,x_{j+1})}(T_j(\widehat{x})) &= \frac{1+\widehat{x}}{2},
\end{aligned}
$$

for all $0 \le j \le N-1$ (without being too pedantic about the $j = 0$ and $j+1 = N$ cases). What is important is that the right hand sides do not depend on j at all, meaning that we can map all of our computations back to the reference element and then only do a couple computations on the reference element. This is a significant cost savings algorithmically, and one of the appealing features of finite element codes. Here is how we can exploit our observations. Let us first compute the matrix M_h with entries

$$
(M_h)_{i,j} = \int_0^l \varphi_j(x)\varphi_i(x)\,\mathrm{d}x.
$$

This matrix is usually referred to as the mass matrix for historical reasons. We split this integral up over our elements $K_k = (x_k, x_{k+1})$:

$$
(M_h)_{i,j} = \sum_{k=0}^{N-1}\int_{K_k}\varphi_j(x)\varphi_i(x)\,\mathrm{d}x =: \sum_{k=0}^{N-1}(M_h^k)_{i,j}
$$

where $(M_h^k)_{i,j}$ represents the contribution from element k to the (i, j) entry of the mass matrix M_h. Now we ask ourselves which basis functions φ_i, φ_j are supported over element K_k. We already answered this earlier: it's when $i, j \in \{k, k+1\}$, and if $i \notin \{k, k+1\}$ or $j \notin \{k, k+1\}$, then $(M_h^k)_{i,j} = 0$. In other words, the element mass matrix M_h^k only has 4 possibly nonzero entries (except M_h^0 and M_h^N which only have 1 possibly nonzero entry). Furthermore, we know exactly which entries may possibly be nonzero: for $k \neq 0, N$, its $M_{k,k}^k$, $M_{k,k+1}^k$, $M_{k+1,k}^k$, and $M_{k+1,k+1}^k$; if $k = 0$, then it's $M_{1,1}^0$ (careful, our matrices start with index 0 here); and if $k = N$, it's $M_{N-1,N-1}^N$. Our problem has been reduced to computing at most 4 entries per element. Let us now see how to do this. We have that

$$(M_h^k)_{k,k} = \int_{x_k}^{x_{k+1}} \varphi_k(x)^2 \, \mathrm{d}x.$$

If we then map back to the reference element using T_k above, we have that

$$(M_h^k)_{k,k} = \frac{h}{2} \int_{-1}^{1} \varphi_k(T_k(\widehat{x}))^2 \, \mathrm{d}\widehat{x} = \frac{h}{8} \int_{-1}^{1} (1 - \widehat{x})^2 \, \mathrm{d}x.$$

Observe now that the last integral is independent of k. Thus, by computing this one integral over the reference element, we have computed the (k, k) entry of every M_h^k. We can proceed similarly for the other nonzero entries of M_h^k to get

$$(M_h^k)_{k,k} = \frac{h}{8} \int_{-1}^{1} (1 - \widehat{x})^2 \, \mathrm{d}\widehat{x} \qquad (M_h^k)_{k,k+1} = \frac{h}{8} \int_{-1}^{1} (1 + \widehat{x})(1 - \widehat{x}) \, \mathrm{d}\widehat{x}$$

$$(M_h^k)_{k+1,k} = (M_h)_{k,k+1}^k \qquad (M_h^k)_{k+1,k+1} = \frac{h}{8} \int_{-1}^{1} (1 + \widehat{x})^2 \, \mathrm{d}\widehat{x}$$

Thus, by computing 4 integrals (only 2 actually due to symmetry), we have computed every single nonzero entry in all of the element mass matrices. Adding these to the right spots in the global mass matrix gives us M_h. Now since the integrands are simply polynomials, we could compute these integrals by hand, but to be even more efficient, we can use a Gaussian quadrature rule to reduce computing the integrals above to evaluating the functions at a finite number of points. Indeed, if we let $p(\widehat{x})$ be one of the integrands above, and if we let $\widehat{x}_q$ and w_q be the Gaussian quadrature points and weights that are exact for polynomials of degree at least 2, then we have that

$$\int_{-1}^{1} p(\widehat{x}) \, \mathrm{d}x = \sum_q p(\widehat{x}_q) w_q.$$

A quick google search will tell you the weights and points for various order quadrature rules. Since these are usually given on the interval $(-1, 1)$, this is why I chose that as our reference element. A pseudocode for assembling the mass matrix is given below.

```
quad_pts = [x_1,...,x_n]
quad_wts = [w_1,...w_n]
basis_0(x) = (1 - x) / 2
basis_1(x) = (1 + x) / 2
for k in [1,...,N-2]: # handle the k = 0 and k = N-1 case separate
  M_k = array(2,2) # 2 x 2 array of zeros
  for i in [0,1]:
    basis_i = basis_0 if i == 0
    basis_i = basis_1 if i == 1
    for j in [0,1]:
      basis_j = basis_0 if j == 0
      basis_j = basis_1 if j == 1
      integrand(x) = basis_i(x) * basis_j(x) * h / 8
     M_k[i,j] = sum_q integrand(x_q) * w_q
  # now add to global matrix
  for i in [0,1]:
    for j in [0,1]:
      M[k+i,k+j] += M_k[i,j] # pay careful attention to the + here
# k = 0 case and k = N case are similar, so I leave that to you
```

That's essentially how every finite element matrix is assembled. We loop over elements, we compute local contribution matrices by mapping back to the reference element and using quadrature, and then we add these to the right spots in the global matrices. Try to do the same thing with the *stiffness matrix* S_h, which is the matrix with entries

$$(S_h)_{i,j} = \int_0^l \varphi_j'(x)\varphi_i'(x)\,\mathrm{d}x$$

Once you have that, you can get A_h via

$$A_h = S_h + \frac{S}{D}M_h$$

Assembling the right-hand side vector $\vec{b}_h$ is also similar. I leave those details to you as well, along with the following pseudocode.

```
quad_pts = [x_1,...,x_n]
quad_wts = [w_1,...w_n]
basis_0(x) = (1 - x) / 2
basis_1(x) = (1 + x) / 2
for k in [1,...,N-2]: # handle the k = 0 and k = N-1 case separate
  b_k = vector(2) # length 2 vector of local contributions
  T_k(x) = (x_k + x_{k+1})/2 + h * x / 2 # map onto reference element
  for i in [0,1]:
    basis_i = basis_0 if i == 0
    basis_i = basis_1 if i == 1
    integrand(x) = basis_i(x) * f(T_k(x)) * h / 4
    b_k[i] = sum_q integrand(x_q) * w_q
  # now add to global rhs vector
  for i in [0,1]:
      M[k+i] += b_k[i] # pay careful attention to the + here
# k = 0 case and k = N case are similar, so I leave that to you
```

Now that we know how to assemble A_h and $\vec{b}_h$, we can solve for $\vec{W}_h$ using any numerical linear algebra solver. Since $\vec{W}_h$ is just the vector of values for W_h at the x_j, we can just plot the $(x_j, (\vec{W}_h)_j)$ data in order to visualize the solution. Now suppose that we have an exact solution W to compare to. To compute the L^2 error, we do something pretty similar to how we assembled our matrices. We observe that since

$$W_h = \sum_{j=1}^{N-1} (\vec{W}_h)_j \varphi_j$$

we have that, on element k,

$$W_h|_{K_k} = (\vec{W}_h)_k \varphi_k + (\vec{W}_h)_{k+1} \varphi_{k+1},$$

and if we map back to the reference element,

$$W_h|_{K_k}(T_k(\widehat{x})) = (\vec{W}_h)_k \frac{1(-\widehat{x}}{2} + (\vec{W}_h)_{k+1} \frac{1+\widehat{x}}{2} =: \widehat{W}_{h,k}.$$

Therefore, the element L^2 error is

$$\begin{aligned} \|W - W_h\|_{L^2_k}^2 &= \int_{K_k} |W - W_h|^2 \, \mathrm{d}x \\ &= \int_{-1}^{1} \underbrace{|W(T_k(\widehat{x})) - \widehat{W}_{h,k}(\widehat{x})|^2}_{\widehat{E}(\widehat{x})} \, dx \\ &\approx \sum_q \widehat{E}(\widehat{x}_q) w_q. \end{aligned}$$

Therefore, we just loop over the elements, compute the element L^2 errors, and add them all up to get the global L^2 error. We can do something similar with the derivatives to also get the H^1 error. For the L^∞ error, we first estimate

$$\|W - W_h\|_{L^\infty_k} \approx \max_q |W(T_k(\widehat{x}_q)) - \widehat{W}_{h,k}(\widehat{x}_q)|$$

and then we take

$$\|W - W_h\|_{L^\infty} \approx \max_k \max_q |W(T_k(\widehat{x}_q)) - \widehat{W}_{h,k}(\widehat{x}_q)|.$$

If we compute errors e_1 and e_2 with mesh sizes h_1 and h_2, and if we assume that the error $e(h)$ decays like $e(h) \approx Ch^r$, then we can estimate the error rate r as

$$r \approx \frac{\log(e_1/e_2)}{\log(h_1/h_2)}.$$

When you plot your errors versus the mesh size, you should make them log-log plots, since that way the slope of the plot represents your error rate.

1 Problem 2

This is simpler than problem 1, so I have nothing to add here.

2 Problem 3

You do essentially the same thing as problems 1 and 2, but now you have one more basis function $\varphi_N(x)$ which is 1 at x_N, 0 at all the other x_j, and is linear.

3 Problem 4

When you define your mesh points x_j, you should line them up with the discontinuities of the coefficient function k. Also, we need to lift the Dirichlet boundary condition at $x = 1$. We do this as follows: pick a function v that is 0 at $x = 0$ and $4/\pi + 3/2$ at $x = 1$, for instance, $v(x) = x(x - 1 + 4/\pi + 3/2)$. Now we set $w = u - v$, so that $u = w + v$. Then upon inserting u into the ODE we get that $-(kw')' = f$ where $f = -(kv')'$. Furthermore, by how we constructed v, we have that $w(0) = w(1) = 0$. Now we can approximate w using the stuff we did in the previous problems to get w_h. Then setting $u_h = w_h + v$ gives us an approximation to u.