These exercises are intended to familiarize you with some tools used to derive, analyze, and understand the Galerkin formulation in 1D. Estimated write-up is about two to three pages, including figures.

1. Solve Laplace’s equation

\[-\nabla^2 u = 0 \text{ in } \Omega, \quad u = \tilde{u} \text{ on } \partial \Omega \quad (1)\]

in the domain shown in Fig. 1 using a single spectral element for \(N=5\) to 50. Take as the exact solution

\[\tilde{u}(x,y) = e^x \sin y. \quad (2)\]

By forcing your numerical solution to satisfy the Dirichlet condition on the boundary your solution will converge to \(\tilde{u}\) as \(N\) is increased. Here, \((x,y) \in \Omega\) is the image of \((\xi,\eta) \in \hat{\Omega}\) under the transformation

\[x(\xi,\eta) = \xi + \frac{3}{4} \left( \frac{\xi + 1}{2} \right) \left( \frac{\eta + 1}{2} \right) \quad (3)\]

\[y(\xi,\eta) = \eta + \frac{3}{4} \left( \frac{\eta + 1}{2} \right) \cos \frac{\pi}{2} \xi. \quad (4)\]

Figure 1: Computational domain for Problem 1.
Use preconditioned conjugate gradient iteration to solve the system to an iteration tolerance of $10^{-14}$ with preconditioners $M$ described below.

1. Plot using semilogy the maximum pointwise error vs $N$.

2. Plot using loglog the number of iterations vs $N$ for the case $M=I$.

3. Plot using loglog the number of iterations vs $N$ for the case where $M$ is a separable approximation to $A$.

Comment on the observed results. In particular, what do your iteration counts say about the asymptotic behavior of the condition number of your iterative system as a function of $N$? (Specifically, $\kappa(M^{-1}A)$).

Notes:

Here we give a brief review of the matrix system $A$ to be solved for a deformed spectral element, followed by a description of a separable approximation to $A$.

Recall, for any $u, v \in X^N$ we have

$$a(v, u) = \Psi_L^T A L u = \sum_{e=1}^{E} (\psi^e)^T A^e u^e$$

(5)

$$= (v^1)^T A^1 u^1 \quad \text{for } E=1,$$

(6)

$$= (v^1)^T D^T G^1 D u^1$$

(7)

$$= (v^1)^T \left( \begin{array}{c} D \xi \\ D \eta \end{array} \right)^T \left[ \begin{array}{cc} G^{1\xi} & G^{1\eta} \\ G^{\xi\xi} & G^{\eta\eta} \end{array} \right] \left( \begin{array}{c} D \xi \\ D \eta \end{array} \right) u^1$$

(8)

Here, $G_{ij} := G^{1}_{ij}$ is the geometric factor involving the Jacobian, the quadrature weights and the metrics associated with the geometric deformation:

$$G_{ij}|_{pq} = \left( \sum_{k=1}^{d} \frac{\partial \xi_i}{\partial x_k} \frac{\partial \xi_j}{\partial x_k} J \right)_{pq} \rho_p \rho_q$$

(9)

where, in this context $(\xi_1, \xi_2) := (\xi, \eta)$. The expression above is evaluated for $i, j \in \{1, \ldots, d\}$ ($d=2$.
or 3) for each quadrature point \( p \) and \( q \) ranging from 0 to \( N \). Recall that

\[
\frac{\partial x}{\partial \xi} = D_\xi x = (\hat{I} \otimes \hat{D})x, \\
\frac{\partial x}{\partial \eta} = D_\eta x = (\hat{D} \otimes \hat{I})x, \\
\frac{\partial y}{\partial \xi} = D_\xi y = (\hat{I} \otimes \hat{D})y, \text{ and} \\
\frac{\partial y}{\partial \eta} = D_\eta y = (\hat{D} \otimes \hat{I})y.
\]

From these relationships we find \( \frac{\partial \xi}{\partial x} \) and so forth by inverting this \( 2 \times 2 \) matrix at each gridpoint.

When solving this system by conjugate gradient iteration you will need to set up a function that evaluates the matrix-vector product implied by (7)–(8). Since we have only one element here this amounts to \( A = RAR^T \), with \( A = D^TGD \), and \( R = R_\eta \otimes R_\xi \) being the boundary-condition restriction matrix, here in tensor-product form. Note that if you form \( A \) it is completely full (for one element), while the factored form requires only \( \hat{D} \) and the three \( G_{ij} \) matrices, so storage that scales as \( N^2 + 3n \), where \( n \) is the number of gridpoints.

For the preconditioning step in conjugate gradients you need a routine that solves \( Mz = r \), where \( M \approx A \). A reasonable (and very good) choice for \( M \) in this single element case is to provide a routine that returns

\[
Z = S_x \left( D^{-1} \left( S_x^T[r]S_y \right) \right) S_y^T,
\]

where \([r]\) is simply the residual vector \( r \) in array form. (I didn’t want to use \( R \) here because we are using it for the restriction matrix.)

Once your code is working, you can test the \( M = I \) case by simply returning \( z = r \) from your preconditioning routine.

**Think carefully** about how you will test your code as you develop it. You have already implemented (14). You can test your PCG code by returning \( z = r \), which gives you just plain conjugate gradient iteration rather than the preconditioned version. If this version works but your PCG one does not, then you know the problem is in your preconditioner. You can test the CG code by setting \( M = I \) (in the manner suggested) and setting, say, \( A = \text{diag}(1 + 1/j) \). A critical point in PCG, and one of the most likely places for error, is that \( A \) and \( M \) must implement matrix-vector products that emulate multiplication by an SPD matrix. Otherwise, PCG will fail to converge. You can also do things like test your PCG code on a \( 2 \times 2 \) matrix, etc. Feel free to use canned PCG routines (e.g., pcg is available in matlab) or to write your own. It’s only about 10 lines to write your own. We will go over this in class at some point but you can find the algorithm on line or talk to me directly.
A theoretical note: We can see that (14) will be an excellent preconditioner for $A$ from the following analysis. First, note that for PCG we are interested in the condition number of $M^{-1}A$, which for SPD $M$ and $A$ corresponds to $\kappa(M^{-1}A) = \lambda_{\max}/\lambda_{\min}$, where $\lambda_k$ solves the eigenvalue problem:

$$M^{-1}A\bar{z} = \lambda\bar{z},$$

which is equivalent to the generalized eigenvalue problem,

$$A\bar{z} = \lambda M\bar{z}.$$ (16)

Here, we are interested only in $\lambda_{\max}$ and $\lambda_{\min}$. Suppose $\bar{z}$ is the eigenvector associated with $\lambda_{\max}$ and consider the Rayleigh quotient

$$\frac{\bar{z}^T A\bar{z}}{\bar{z}^T M\bar{z}} = \lambda_{\max}. $$ (17)

We know from power iteration (in fact, Rayleigh-quotient iteration) that (17) provides a definition of $\lambda_{\max}$:

$$\lambda_{\max} = \max_{\bar{z} \in \mathbb{R}^n} \frac{\bar{z}^T A\bar{z}}{\bar{z}^T M\bar{z}}, $$ (18)

which is just like the standard Rayleigh quotient form save that the inner product in the denominator is now the $M$ inner product. Similarly, we have

$$\lambda_{\min} = \min_{\bar{z} \in \mathbb{R}^n} \frac{\bar{z}^T A\bar{z}}{\bar{z}^T M\bar{z}}, $$ (19)

Now, let’s look at the numerator and denominator in detail. Starting with $\lambda_{\max}$ we have,

$$\lambda_{\max} = \max_{\bar{z} \in \mathbb{R}^n} \frac{\bar{z}^T A\bar{z}}{\bar{z}^T M\bar{z}}$$

$$= \max_{\bar{z} \in \mathbb{R}^n} \frac{\bar{z}^T RD^T G D R^T \bar{z}}{\bar{z}^T BD R^T \bar{z}}$$

$$= \max_{y \in \mathbb{R}(DR^T)} \frac{y^T G y}{y^T B y},$$

with a similar result for $\lambda_{\min}$. Note that the diagonal mass matrix, $M = I_d \otimes (\hat{B} \otimes \hat{B})$, is also present in the numerator of (22), so the $B$ terms drop out, leaving only the Jacobian/metric terms in the
numerator. Here, we have the remarkable result that $\kappa(M^{-1}A)$ is asymptotically independent of $N$. It depends only on the geometry and I think the relationship is fairly simple to analyze. Note that one can seek a better approximation to the geometry than $\hat{\Omega}$ and still have a separable form. So, at no extra cost per iteration one could do a bit of optimization to find a better preconditioner.

**Finally,** it’s also worth mentioning that PCG will give you an almost-free estimate of the spectrum of $M^{-1}A$ and its eigenvectors. Essentially, PCG is a Lanczos iteration. You can get excellent estimates of the condition number of $M^{-1}A$ by looking at the eigenvalues of the tridiagonal matrix comprising $(\alpha_k, \beta_k)$ from the PCG algorithm.