1 Poisson Equation

Our first boundary value problem will be the steady-state heat equation, which in two dimensions has the form

$$-\left(\frac{\partial}{\partial x}k\frac{\partial T}{\partial x}+\frac{\partial}{\partial y}k\frac{\partial T}{\partial y}\right)=q'''(\mathbf{x}),\qquad\text{plus BCs.}\tag{1}$$

If the thermal conductivity $k>0$ is constant, we can pull it outside of the partial derivatives and divide both sides by k to yield the 2D Poisson equation

$$-\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)=f(\mathbf{x}),\qquad\text{plus BCs.}\tag{2}$$

with $f:=q'''/k$ and $u:=T$. (We typically use u for scalar fields throughout the course for notational convenience. Later, it will pair well with the set of test functions, denoted as v.)

The short-hand notation for the Poisson equation (2) is

$$-\nabla^2 u \;=\; f(\mathbf{x})\text{ in }\Omega,\qquad u=u_b\text{ on }\partial\Omega_D,\ \nabla u\cdot\hat{\mathbf{n}}=g\text{ on }\partial\Omega_N,$$

which applies in any number of space dimensions d. Here, we've indicated a mixture of boundary conditions: Dirichlet on $\partial\Omega_D$, where u is prescribed, and Neumann on $\partial\Omega_N$, where the normal component of the gradient is prescribed. We take $\hat{\mathbf{n}}$ to be the outward pointing normal on the domain boundary $\partial\Omega=\partial\Omega_D\bigcup\partial\Omega_N$.

We will work with the Poisson equation and extensions throughout the course. At this point we want to introduce some simple cases in order to understand optimal solution strategies in the 3D case, which is arguably the most important in terms of compute cycles consumed throughout the world.

1.1 Finite Differences in 1D

The basic idea behind the finite difference approach to solving differential equations is to replace the differential operator with difference operators at a set of n gridpoints. In 1D, it is natural to order the points sequentially, as illustrated in Fig. 1. Here, we consider the two-point boundary value problem

$$-\frac{d^2u}{dx^2}\;=\;f(x),\qquad u(0)=u(1)=0.\tag{3}$$

We use the second-order centered difference approximation to the second derivative, which for uniform grid spacing $h:=x_j-x_{j-1}$ is

$$\begin{aligned}-\frac{u_{j+1}-2u_j+u_{j-1}}{h^2}&=-\left(\left.\frac{d^2u}{dx^2}\right|_{x_j}+\left.\frac{h^2}{12}\frac{d^4u}{dx^4}\right|_{x_j}+O(h^4)\right)&(4)\\&=f(x_j)-\left.\frac{h^2}{12}\frac{d^4u}{dx^4}\right|_{x_j}-O(h^4)&(5)\\&\approx f(x_j).&(6)\end{aligned}$$

Note that (4)–(6) comprises three steps. First, we use the Taylor series expansion to express our difference formula in terms of the desired derivative plus higher-order corrections. Second, we replace the desired derivative by the data (f), given by the differential equation. Finally, we incur the our truncation error by dropping the higher-order terms in the Taylor series. We equate the results to arrive at our discete system for the *numerical approximation*, u_j,

$$-\frac{u_{j+1}-2u_j+u_{j-1}}{h^2}\;=\;f_j,\qquad j=1,\ldots,n.\tag{7}$$

The set of equations (7) represents our discretization of the original differential equation and is an algebraic system consisting of n equations in n unknowns, u_j, j=1,...,n. Each equation j relates u_{j-1}, u_j, and u_{j+1} to f_j. For this reason, the resulting matrix system is *tridiagonal*,

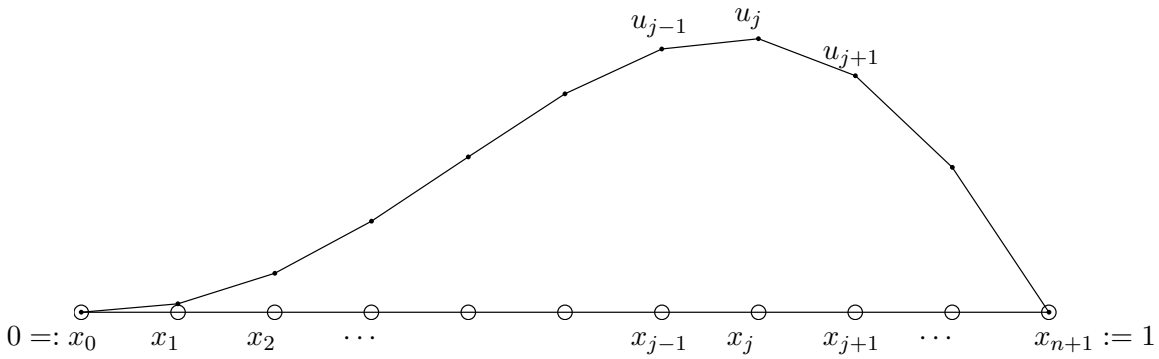


Figure 1: Finite difference grid on $\Omega := [0, 1]$.

$$\underbrace{\frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}}_{A_x} \underbrace{\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_n \end{pmatrix}}_{\underline{u}} = \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_n \end{pmatrix}}_{\underline{f}}, \tag{8}$$

which has the shorthand $A_x\underline{u} = \underline{f}$, where $\underline{u}$ is the vector of unknowns and $\underline{f}$ the vector of data values, as indicated in (8). We will often simply use A instead of A_x for notational convenience. Here, we explicitly call out the x-coordinate association in preparation for the 2D development coming in the next section.

We list several attributes of $A = A_x$ that carry over to higher space dimensions.

- A is *symmetric*, which implies it has real eigenvalues and an orthonormal set of eigenvectors satisfying $A\underline{s}_j = \lambda_j \underline{s}_j$, $\underline{s}_j^T \underline{s}_i = \delta_{ij}$, where the Kronecker δ_{ij} equals 1 when $i = j$ and 0 when $i \neq j$.
- A is also *positive definite*, which means that $\underline{x}^T A\underline{x} > 0$ for all $\underline{x} \neq 0$. It also implies $\lambda_j > 0$. Symmetric positive definite (SPD) systems are particularly attractive because they can be solved without pivoting using Cholesky factorization, $A = LL^T$, or iteratively using preconditioned conjugate gradient (PCG) iteration. (For large sparse systems, PCG is typically the best option.)
- A is *sparse*. It has a fixed maximal number of nonzeros per row (3, in the case of A_x), which implies that the total number of nonzeros in A is linear in the problem size, n. We say that the storage cost for A is $O(n)$, meaning that there exists a constant C independent of n such that the total number of words to be stored is $< Cn$.
- A is *banded* with bandwidth $w = 1$, which implies that $k_{ij} = 0$ for all $|i - j| > w$. A consequence is that the storage bound for the Cholesky factor L is $< (w + 1)n$. For the 1D case with w=1, the storage for L is thus $O(n)$. As we shall see, the work to compute the factors is $O(w^2 n)$.
- A^{-1} is completely full. We know this from the physics of the problem. Consider $\underline{f} \equiv 0$ save for one point, where $f_j = 1$ (i.e., f_j is the nth column of the $n \times n$ identity matrix). This case corresponds to a point heat source at x_j and, as we know, the temperature will be nonzero everywhere except at the endpoints. In fact, it will exhibit a linear decay from x_j to the boundaries. This is the exact Green's function for both the continuous and the discrete case. It's an easy exercise to show that, for any matrix $A = (\underline{a}_1\, \underline{a}_2\, \ldots\, \underline{a}_n)$ we have $\underline{a}_j = A\underline{e}_j$ when $\underline{e}_j$ is the jth column of I. The preceding arguments establish that A^{-1} must be completely full.

1.2 Poisson Equation in $\mathbb{R}^2$

Our principal concern at this point is to understand the (typical) matrix structure that arises from the 2D Poisson equation and, more importantly, its 3D counterpart. The essential features of this structure will be similar for other discretizations (i.e., FEM, SEM), other PDEs, and other space dimensions, so there is merit to starting with this relatively simple system.

The steady-state heat equation in two dimensions is:

$$-\nabla \cdot k\nabla T = q'''(x,y), \qquad \text{plus BCs.} \tag{9}$$

For constant thermal conductivity k this equation reduces to the standard Poisson equation in $\Omega := [0,1]^2 \subset \mathbb{R}^2$, which we usually express in terms of u for notational convenience:

$$\begin{aligned} -\nabla^2 u &= f(x,y), \qquad \text{plus BCs} \\ &= -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \\ &= -\left(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2}\right) + O(h^2), \end{aligned} \tag{10}$$

where we have substituted the finite difference approximations, assumed to be about the point $\mathbf{x}_{ij} := (x_i, y_j)$,

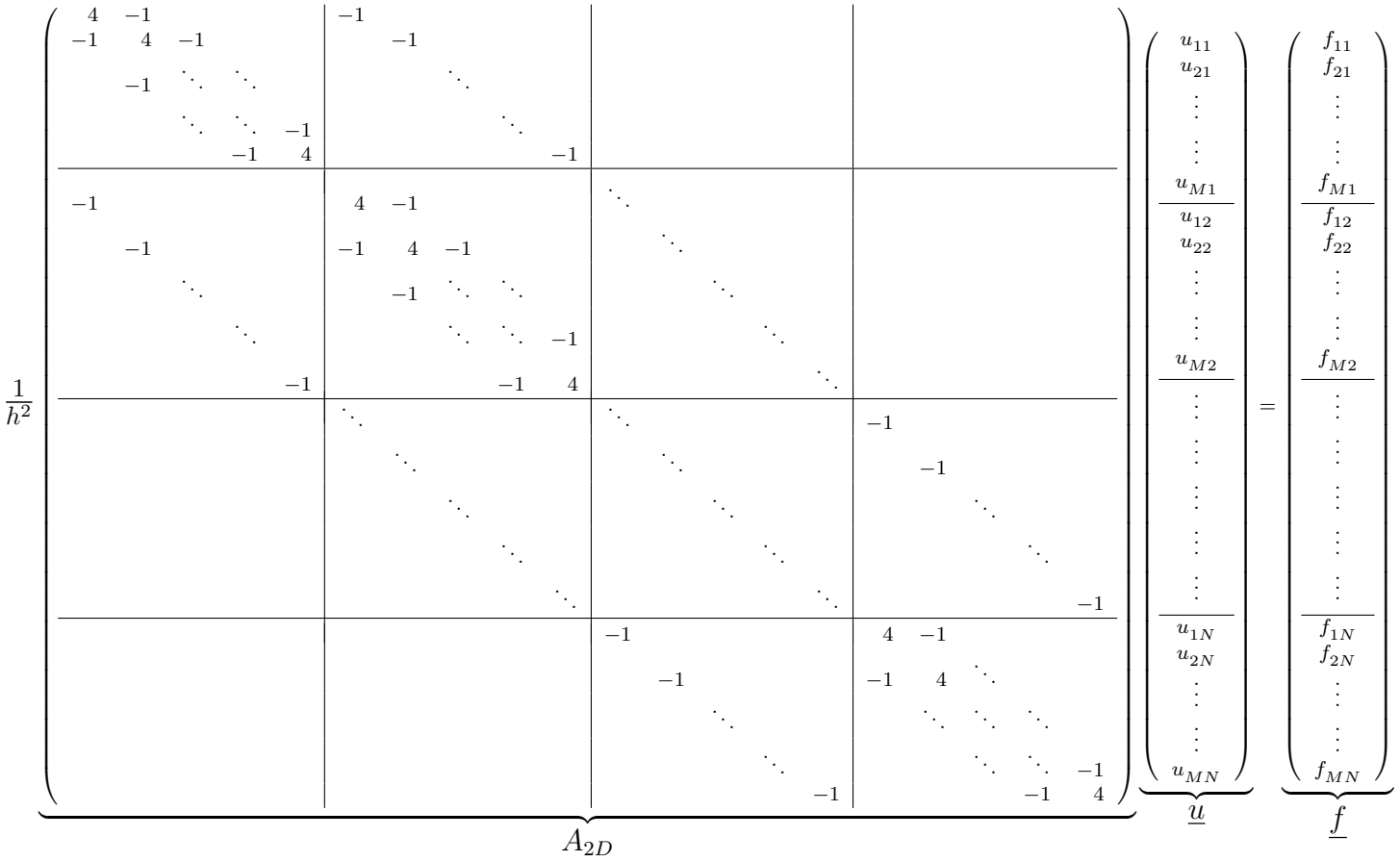
$$\begin{aligned} \frac{\delta^2 u}{\delta x^2} &:= \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} \\ \frac{\delta^2 u}{\delta y^2} &:= \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}, \end{aligned} \tag{11}$$

with the further assumption of uniform grid spacing, $\Delta x = \Delta y = h$. We'll also consider homogeneous Dirichlet boundary conditions, that is, $u(x,y)|_{\partial\Omega} \equiv 0$. The respective unknowns and data in this case are u_{ij} and f_{ij}, governed by the following system of equations

$$-\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}\right) = f_{ij}, \tag{12}$$

for $i,j \in [1,\ldots,N]^2$.

Assuming a *lexicographical ordering* in which the i- (x-) index advances fastest, the system takes on the following matrix structure for $\Delta x = \Delta y = h$.



Note that A_{2D} can be expressed as the sum of two systems, one associated with A_x coming from $\frac{\delta^2 u}{\delta x^2}$, and one associated with one associated with A_y coming from $\frac{\delta^2 u}{\delta y^2}$. Specifically, we can write

$$A_{2D} \;=\; (I_y \otimes A_x) \;+\; (A_y \otimes I_x), \tag{13}$$

where we have introduced the Kronecker (or *tensor*) product, $\otimes$. For two matrices A and B, their Kronecker product $C = A \otimes B$ is defined as the block matrix

$$C \;:=\; \begin{pmatrix} a_{11}B & a_{12}B & \cdots & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & \cdots & a_{2n}B \\ \vdots & \vdots & & & \vdots \\ a_{m1}B & a_{m2}B & \cdots & \cdots & a_{mn}B \end{pmatrix}. \tag{14}$$

We will soon explore a few properties of this form, but for now simply note that it allows a clean expression of the discretized Poisson operator in 2D. Consider the following splitting of A_{2D}.

$$
A_{2D} = \frac{1}{h^2}
\left(\begin{array}{c|c|c|c}
\begin{array}{ccccc}
2 & -1 & & & \\
-1 & 2 & -1 & & \\
 & -1 & \ddots & \ddots & \\
 & & \ddots & \ddots & -1 \\
 & & & -1 & 2
\end{array} & & & \\
\hline
 & \begin{array}{ccccc}
2 & -1 & & & \\
-1 & 2 & -1 & & \\
 & -1 & \ddots & \ddots & \\
 & & \ddots & \ddots & -1 \\
 & & & -1 & 2
\end{array} & & \\
\hline
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & \ddots \\
 & & & & \ddots \\
 & & & & \ddots
\end{array}
\right.
$$

$$
\left.\begin{array}{c|c|c|c}
 & & & \\
\hline
 & & & \\
\hline
 & & & \\
\hline
\begin{array}{ccccc}
2 & -1 & & & \\
-1 & 2 & \ddots & & \\
 & \ddots & \ddots & \ddots & \\
 & & \ddots & \ddots & -1 \\
 & & & -1 & 2
\end{array} & & &
\end{array}\right)
$$

$$
+ \frac{1}{h^2}
\left(\begin{array}{c|c|c|c}
\begin{array}{ccccc}
2 & & & & \\
 & 2 & & & \\
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & 2
\end{array} & \begin{array}{ccccc}
-1 & & & & \\
 & -1 & & & \\
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & -1
\end{array} & & \\
\hline
\begin{array}{ccccc}
-1 & & & & \\
 & -1 & & & \\
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & -1
\end{array} & \begin{array}{ccccc}
2 & & & & \\
 & 2 & & & \\
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & 2
\end{array} & \begin{array}{ccccc}
\ddots & & & & \\
 & \ddots & & & \\
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & \ddots
\end{array} & \\
\hline
 & \begin{array}{ccccc}
\ddots & & & & \\
 & \ddots & & & \\
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & \ddots
\end{array} & \begin{array}{ccccc}
\ddots & & & & \\
 & \ddots & & & \\
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & \ddots
\end{array} & \begin{array}{ccccc}
-1 & & & & \\
 & -1 & & & \\
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & -1
\end{array} \\
\hline
 & & \begin{array}{ccccc}
-1 & & & & \\
 & -1 & & & \\
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & -1
\end{array} & \begin{array}{ccccc}
2 & & & & \\
 & 2 & & & \\
 & & \ddots & & \\
 & & & \ddots & \\
 & & & & 2
\end{array}
\end{array}\right)
$$

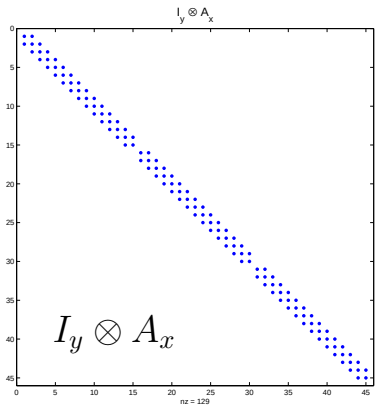
$$A_{2D} = \begin{pmatrix} A_x & & & \\ & A_x & & \\ & & \ddots & \\ & & & A_x \end{pmatrix} + \frac{1}{h^2} \begin{pmatrix} 2I_x & -I_x & & \\ -I_x & 2I_x & \ddots & \\ & \ddots & \ddots & -I_x \\ & & -I_x & 2I_x \end{pmatrix}$$

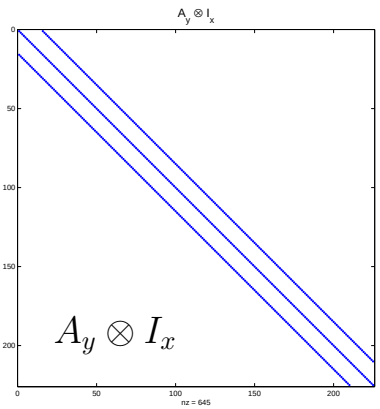
$$= (I_y \otimes A_x) + (A_y \otimes I_x)$$

We see that we can express A_{2D} as $(I_y \otimes A_x) + (A_y \otimes I_x)$. The first term is nothing other than $\frac{\delta^2}{\delta x^2}$ being applied to each row (j) of u_{ij} and the second term amounts to applying $\frac{\delta^2}{\delta y^2}$ to each column (i) on the grid.

1.3 Matlab Kronecker Product Demos

```
close all; format compact;
% Kronecker Product Demo
%
%    NOTE:  It is important to use SPARSE matrices throughout.
%
%           Otherwise, your run times will be very long and
%           you will likely run out of memory!

Lx=2; Ly=1;
nx=15; ny=3;  % Number of _interior_ points

dx=Lx/(nx+1); dy=Ly/(ny+1);

%  USE help spdiags

e = ones(nx,1); Ax = spdiags([-e 2*e -e], -1:1, nx, nx)/(dx*dx);
e = ones(ny,1); Ay = spdiags([-e 2*e -e], -1:1, ny, ny)/(dy*dy);

Ix=speye(nx); Iy=speye(ny);

A = kron(Iy,Ax) + kron(Ay,Ix);   %%% FINITE DIFFERENCE STIFFNESS MATRIX

% A couple of demo cases without the 1/(dx*dx) scaling.

nd= 5;
e = ones(nd,1); Ad = spdiags([-e 2*e -e], -1:1, nd, nd);
T = kron(Iy,Ad); full(T)

nd= 15;
e = ones(nd,1); Ad = spdiags([-e 2*e -e], -1:1, nd, nd);
T = kron(Iy,Ad); spy(T)
title('I_y \otimes A_x','fontsize',16)
set(gcf,'PaperUnits','normalized');set(gcf,'PaperPosition',[0 0 1 1])
print -dpdf iyax.pdf

pause; figure
nd= 5;
e = ones(nd,1); Ad = spdiags([-e 2*e -e], -1:1, nd, nd);
T = kron(Ad,Ix); full(T)

nd= 15;
e = ones(nd,1); Ad = spdiags([-e 2*e -e], -1:1, nd, nd);
T = kron(Ad,Ix); spy(T)
title('A_y \otimes I_x','fontsize',16)
set(gcf,'PaperUnits','normalized');set(gcf,'PaperPosition',[0 0 1 1])
print -dpdf ayix.pdf
```





Note that our finite-difference stiffness matrix in matlab would be written as

```
A = kron(Iy,Ax) + kron(Ay,Ix)
```

where `Ax` and `Ay` are formed using the matlab `spdiags` command (`help spdiags`), and Iy and Ix are formed using `speye`.

It is important to use *sparse matrices* in matlab for these higher-dimensional (2D and 3D) problems or you will run out of memory and it will take *very long* to solve these problems.

This problem is known in scientific computing and *the curse of dimensionality*.

1.4 Poisson Equation in $\mathbb{R}^3$

We now extend the 1D and 2D concepts to the most important 3D case. The short story is that the 3D stiffness matrix takes the wonderfully symmetric form

$$\begin{aligned} A_{3D} &= (I_z \otimes A_{2D}) + (A_z \otimes I_{2D}) \\ &= (I_z \otimes I_y \otimes A_x) + (I_z \otimes A_y \otimes I_x) + (A_z \otimes I_y \otimes I_x). \end{aligned} \tag{15}$$

and the discrete system is as before $A_{3D}\underline{u} = \underline{f}$. This of course is the form that arises for a finite difference discretization of $-\nabla^2 u = f$ in $\Omega = [0,1]^3$, $u = 0$ on $\partial\Omega$, or, more explicitly,

$$-\left(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} + \frac{\delta^2 u}{\delta z^2}\right) = f(x_i, y_j, z_k), \tag{16}$$

with

$$\left.\frac{\delta^2 u}{\delta z^2}\right|_{ijk} := \frac{u_{ij,k+1} - 2u_{ijk} + u_{ij,k-1}}{\Delta z^2}, \tag{17}$$

and equivalent expressions for $\frac{\delta^2 u}{\delta x^2}$ and $\frac{\delta^2 u}{\delta y^2}$.

Note that, with (15), we really have not restricted ourselves to uniform mesh spacing in each direction. We could have different grid spacing Δx, Δy, and Δz and number of mesh points N_x, N_y, and N_z, in each of the space directions. Then, I_x would be the $N_x \times N_x$ identity matrix and A_x would be the corresponding stiffness matrix, as would also be the case for y and z.