MA3227 Numerical Analysis II

Lecture 3: Finite Differences

Simon Etter

Semester II, AY 2020/2021

Finite Differences

Problem statement

Given $\Omega \subset \mathbb{R}^d$ and $f : \Omega \to \mathbb{R}$, determine $u : \Omega \to \mathbb{R}$ such that

$$-\Delta u(x) = f(x) \qquad \text{for all } x \in \Omega.$$

Terminology and notation

- $\Delta = \frac{\partial^2}{\partial x_1^2} + \ldots + \frac{\partial^2}{\partial x_d^2}$ is called the *Laplace operator* or *Laplacian*.
- The above problem is known as *Poisson's equation*.
- Poisson's equation is a particular example of a *partial differential equation (PDE)*, i.e. an equation in terms of an unknown function $u(x)$ and its partial derivatives.

Finite Differences

Remark

Even though Poisson's equation is just a particular example of a partial differential equation, it is the only example that I will consider in this module. There are two, partially overlapping reasons for this.

▶ Discussing PDEs in general runs a high risk that you will fail to see the forest for the trees. Instead, I believe it is more productive to focus on a special case and trust that you will be able to adapt the ideas once the need arises.

▶ Most practically relevant PDEs can be summarised as "Poisson + some extra complications". There is a high chance that you can pursue an entire career in applied mathematics without venturing far beyond Poisson's equation.

Finite Differences

Example

For $\Omega = (0,1)$ and $f(x) = x - x^2$, Poisson's equation becomes

$$-u''(x) = x - x^2 \qquad \text{for all } x \in (0,1).$$

Taking antiderivatives, we obtain

$$-u'(x) = \tfrac{1}{2} x^2 - \tfrac{1}{3} x^3 + c_1,$$
$$-u(x) = \tfrac{1}{6} x^3 - \tfrac{1}{12} x^4 + c_1 x + c_2,$$

where $c_1, c_2 \in \mathbb{R}$ are some unspecified parameters.

Discussion

The above example shows that additional constraints are required to ensure that Poisson's equation has a unique solution $u(x)$.

These additional constraints typically take the form of boundary conditions, see next slide.

Finite Differences

Terminology: Boundary conditions

▶ Dirichlet boundary conditions: $u(x) = g(x)$ for all $x \in \partial\Omega$.

▶ Neumann boundary conditions: $\frac{\partial u}{\partial n}(x) = g(x)$ for all $x \in \partial\Omega$.

If $g(x) = 0$, then these boundary conditions are called *homogeneous*.

$\partial\Omega$ denotes the boundary of $\Omega \subset \mathbb{R}$.
$\frac{\partial u}{\partial n}$ denotes the directional derivative of $u(x)$ in the direction normal to the boundary.

Example (continued)

For $\Omega = (0, 1)$, the homogeneous Dirichlet boundary conditions are

$$u(0) = u(1) = 0.$$

These conditions provide two equations for determining the parameters c_1, c_2 in the formula

$$u(x) = \tfrac{1}{6} x^3 - \tfrac{1}{12} x^4 + c_1 x + c_2$$

derived on the previous slide.

Finite Differences

Example (continued)

For $\Omega = (0, 1)$, the homogeneous Neumann boundary conditions are

$$u'(0) = u'(1) = 0.$$

These conditions provide two equations for determining the c_1 in

$$u(x) = \tfrac{1}{6} x^3 - \tfrac{1}{12} x^4 + c_1 x + c_2,$$

and no equation for determining c_2 since

$$u'(x) = \tfrac{1}{2} x^2 - \tfrac{1}{3} x^3 + c_1$$

is independent of c_2. Poisson's equation with Neumann boundary conditions can therefore have none or many solutions.

This example shows that Neumann boundary conditions are somewhat more complicated than Dirichlet boundary conditions. I will focus on homogeneous Dirichlet boundary conditions for most of this module.

Finite Differences

Derivation of Poisson's equation (not examinable)

Poisson's equation $-\Delta u = f$ may look like a fairly arbitrary combination of mathematical operations, but there is a good reason why much of applied mathematics is dedicated to studying precisely this particular equation: Poisson's equation models diffusion, and diffusion features prominently in many fields of science and technology. I will list some examples in a moment.

The following slides will illustrate the relationship between Poisson's equation and diffusion by demonstrating how Poisson's equation arises in a macroscopic model of ants in a sand pit. As you will soon realise, this model is somewhat silly, but I believe it is the clearest way to describe the physical principles underlying Poisson's equation.

Finite Differences

Derivation of Poisson's equation (not examinable)

Consider the following model.

- $\Omega \subset \mathbb{R}^2$ describes a sandpit containing a colony of ants.

- $u : \Omega \to \mathbb{R}$ describes the *concentration* of ants.

- $f : \Omega \to \mathbb{R}$ describes the net flow of ants moving from the nest onto the surface of the sandpit, i.e. if $f(x) = 1$, then we have one more ant per second appearing on the surface of the sandpit than disappearing into the nest at the point $x \in \Omega$.

  I will refer to $f(x)$ as a *source term*, since from the point of view of an birds-eye-view observer, it describes the rate at which ants appear or disappear at any given location $x \in \Omega$.

- $J : \Omega \to \mathbb{R}^2$ describes the *net flow* of ants, i.e. if $J(x) = (1,0)$, then one more ant crosses the point $x \in \Omega$ from left to right than from right to left per second, and the number of ants crossing from top to bottom equals the number of ants crossing from bottom to top.

Finite Differences

Derivation of Poisson's equation (not examinable, continued)

In the above model, the number of ants in a domain $\Omega' \subset \Omega$ can only change if ants either enter or leave the nest within Ω, or if ants move into or out of Ω.

In mathematical terms, this means that we must have

$$\frac{\partial}{\partial t} \underbrace{\int_{\Omega'} u \, dx}_{\#\text{ ants in } \Omega'} = - \underbrace{\int_{\partial\Omega'} n \cdot J \, dx}_{\#\text{ ants crossing } \partial\Omega'} + \underbrace{\int_{\Omega'} f \, dx}_{\substack{\#\text{ ants appearing or} \\ \text{disappearing in } \Omega'}}.$$

$n = n(x)$ denotes the exterior normal vector at $x \in \partial\Omega'$.

This relationship between concentration $u(x)$, flow $J(x)$ and source term $f(x)$ is called *conservation of mass*.

Finite Differences

Derivation of Poisson's equation (not examinable, continued)

Applying the divergence law

$$\int_{\partial \Omega'} n \cdot J \, dx = \int_{\Omega'} \nabla \cdot J \, dx$$

to the conservation of mass equation

$$\frac{\partial}{\partial t} \int_{\Omega'} u \, dx = - \int_{\partial \Omega'} n \cdot J \, dx \; + \int_{\Omega'} f \, dx$$

and moving all terms to the left, we obtain

$$\int_{\Omega'} \left(\frac{\partial u}{\partial t} + \nabla \cdot J - f \right) dx = 0.$$

The next slide demonstrates how to translate this statement into a PDE.

Finite Differences

Theorem (not examinable)

The following statements are equivalent for any function $g : \Omega \to \mathbb{R}$.

1. $\displaystyle\int_{\Omega'} g(x)\, dx = 0$ for all $\Omega' \subset \Omega$

2. $g(x) = 0$ except on some set $\Omega_N \subset \Omega$ of measure 0.

   A set $\Omega_N \subset \mathbb{R}^n$ is said to have measure 0 if $\int_{\Omega_N} 1\, dx = 0$.

Proof. $(1) \impliedby (2)$:

$$\int_{\Omega'} g(x)\, dx = \int_{\Omega' \setminus \Omega_N} 0\, dx + \int_{\Omega' \cap \Omega_N} g(x)\, dx = 0.$$

The second integral is 0 because $\Omega' \cap \Omega_N \subset \Omega_N$ has measure 0.

$(1) \implies (2)$: We observe:

▶ $\Omega_+ = \{x \in \Omega \mid g(x) > 0\}$ has measure 0 since otherwise $\int_{\Omega_+} g(x)\, dx > 0$ in contradiction to (1).

▶ $\Omega_- = \{x \in \Omega \mid g(x) < 0\}$ has measure 0 for the same reason.

▶ $\Omega_N = \Omega_+ \cup \Omega_-$ and hence Ω_N has measure 0.

Finite Differences

The above theorem motivates the following definition.

Definition: Almost all and almost everywhere

The equation $g(x) = 0$ is said to be satisfied for *almost all* $x \in \Omega$ or *almost everywhere on* Ω if it is satisfied for all $x \in \Omega \setminus \Omega_N$ where $\Omega_N \subset \Omega$ denotes some unspecified set of measure 0.

Finite Differences

PDEs and the notion of almost everywhere

Applying the above theorem to the conservation of mass equation in divergence form,

$$\int_{\Omega'} \left(\frac{\partial u}{\partial t} + \nabla \cdot J - f \right) dx = 0,$$

we conclude that

$$\frac{\partial u}{\partial t} + \nabla \cdot J - f = 0 \qquad \text{for almost all } x \in \Omega.$$

Such almost-everywhere-satisfied PDEs can be handled using a rigorous mathematical theory (Lebesgue and Sobolev spaces), but this theory involves many technical complications which are well beyond the scope of this module. To avoid these complications, I will ignore the "almost everywhere" in the following and instead assume that

$$\frac{\partial u}{\partial t} + \nabla \cdot J - f = 0 \qquad \text{for } \textit{all } x \in \Omega.$$

Doing so is common in science and engineering, but it comes at the price that there will be some phenomena in the theory of PDEs which cannot be explained in this simplified framework.

Finite Differences

Derivation of Poisson's equation (not examinable, continued)

We have seen on the previous slide that conservation of mass implies that the ant concentration $u(x) \in \mathbb{R}$, the ant entry and exits rate $f(x) \in \mathbb{R}$ and the and flow $J(x) \in \mathbb{R}^2$ are related by

$$\frac{\partial u}{\partial t} + \nabla \cdot J - f = 0.$$

Two further steps are required to turn the above into Poisson's equation.

▶ We must assume that we are in a steady state, i.e. $\frac{\partial u}{\partial t} = 0$.

▶ We must relate the ant flow $J(x)$ to the ant concentration $u(x)$.

The second step will be tackled on the next slide.

Finite Differences

Derivation of Poisson's equation (not examinable, continued)

Let us assume that the ants try to spread out as evenly as possible, i.e. if $u(x)$ is large at some point x_1 but low at a nearby point x_2, then the ants will move from x_1 to x_2.
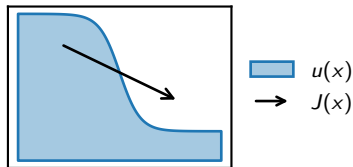
One way to formalise this idea is to assume that the ant flow $J(x)$ satisfies the so-called *Fick's law*

$$J(x) = -D \nabla u(x) \qquad \text{for some } D > 0.$$

Fick's law occurs frequently in physics because it describes the correct macroscopic ant flow $J(x)$ in particular if each ant moves around randomly.

The constant D in Fick's law is called the *diffusion coefficient*.

Finite Differences

Derivation of Poisson's equation (not examinable, continued)

We now have all the ingredients in place to derive Poisson's equation from the ants-in-sandpit model:

▶ We have seen on slide 13 that conservation of mass implies

$$\frac{\partial u}{\partial t} + \nabla \cdot J - f = 0.$$

▶ Inserting the steady state assumption $\frac{\partial u}{\partial t} = 0$ and Fick's law

$$J(x) = -D \nabla u(x),$$

we obtain

$$0 - D \nabla \cdot \nabla u - f = 0 \quad \Longleftrightarrow \quad -D \Delta u = f.$$

This is exactly Poisson's equation. (We can get rid of the diffusion coefficient D simply by replacing $f(x)$ with $D f(x)$.)

Given the above, we conclude:

> Poisson's equation $-\Delta u = f$ relates the steady state concentration $u(x)$ of a conserved, diffusing quantity with the rate $f(x)$ at which this quantity is produced and/or destroyed.

Finite Differences

Meaning of boundary conditions

Now that we understand the physical meaning of the Poisson equation $-\Delta u = f$, let us next look into the meaning of the boundary conditions introduced on slide 5. In the context of our "ants in sandpit" model, these boundary conditions can be interpreted as follows.

▶ Homogeneous Dirichlet boundary conditions, $u(\partial\Omega) = 0$.

No ants at the boundary. One way to achieve this would be to paint the sandpit walls with glue such that any ant touching the wall gets stuck and no longer counts as a freely moving ant.

(Note that $u(x)$ must describe the concentration of freely moving ants since otherwise $J(x) = -\nabla u(x)$ may result in a positive ant flow out of a region where there are only glued ants left.)

▶ Homogeneous Neumann boundary conditions, $\frac{\partial u}{\partial n}(\partial\Omega) = 0$.

No net ant flow $J(x) = -\nabla u$ across the boundary. This condition is satisfied if every ant hitting the boundary simply turns around and continues walking around randomly.

Finite Differences

Applications of Poisson's equation

The physical principles which give rise to Poisson's equation, namely conservation of mass and Fick's law, are ubiquitous in physics. Correspondingly, Poisson's equation plays an important role in many different fields of science and engineering. Here are some examples.

- ▶ Conductive heat transfer: $\frac{\partial T}{\partial t} = \Delta T + f$ where T denotes temperature and f denotes heat sources and sinks.
- ▶ Electrostatics / gravity: $-\Delta \phi = \rho$ where ϕ denotes the electric / gravitational potential and ρ denotes the charge / mass density.
- ▶ Fluid dynamics (Navier-Stokes equations): $\frac{\partial u}{\partial t} = \nu \, \Delta u - u \cdot \nabla u$ where u denotes the flow velocity and ν denotes the viscosity.
- ▶ Quantum mechanics (Schrödinger equation): $i \frac{\partial \psi}{\partial t} = -\Delta \psi + V \, \psi$ where ψ denotes the wave function and V the potential energy.

Finite Differences

Outlook

Our main goal in this lecture is to develop numerical methods for evaluating the map

$$\big(f : \Omega \to \mathbb{R}\big) \quad \mapsto \quad \big(u : \Omega \to \mathbb{R} \text{ such that } -\Delta u = f\big).$$

To do so, we must address the following fundamental question:

How do we represent an arbitrary function $\Omega \to \mathbb{R}$ on a computer?

The following slides will go through several possible answers to this question and discuss their respective strengths and weaknesses.

Finite Differences

Representing functions

Option 1: Function handles

Most programming languages treat functions like any other piece of data; hence it is possible to write a function

$$u = \texttt{solve\_poisson(f)}$$

where the input `f` and output `u` are themselves arbitrary functions.

Such a representation of $f(x)$ and $u(x)$ may look promising at first because it is clearly the most general possible. In particular, this representation would allow for `solve_poisson(f)` to return a $u(x)$ which is exact up to the granularity of `Float64`.

Unfortunately, the apparent generality of this approach breaks down once we look into it more closely: there are $N = (2^{64})^{2^{64}}$ distinct functions $f : \texttt{Float64} \to \texttt{Float64}$; thus if `solve_poisson(f)` is to be surjective in this space then it must sample at least $\log_2(N) = 64 \times 2^{64} \approx 10^{21}$ bits of $f(x)$, or put differently, it must sample $f(x)$ at at least $2^{64} \approx 10^{19}$ different points x. This is not feasible.

Finite Differences

Representing functions (continued)

Option 1: Function handles (continued)

We therefore conclude that while it is certainly possible and perhaps even convenient to let solve_poisson(f) -> u operate at the level of function handles, we should not be fooled by the apparent generality suggested by this approach: in practice, u(x) must necessarily be chosen from a space which is much smaller than

$$\{\texttt{u} : \texttt{Float64}^d \to \texttt{Float64}\};$$

hence we continue our quest for what this space should be.

Finite Differences

Representing functions (continued)

Option 2: Polynomials

Polynomials can approximate any continuous functions arbitrarily closely (Weierstrass approximation theorem), and they can easily be differentiated and integrated. With a bit of work, these operations can be used to compute an approximate solution $u_n(x)$ to the Poisson equation $-\Delta u = f$ as follows.

- Approximate $f : \Omega \to \mathbb{R}$ with a polynomial $f_n \in \mathcal{P}_n^d$.

  $\mathcal{P}_n$ denotes the space of univariate polynomials of degree $\leq n$.

- Determine $u_n \in \mathcal{P}_{n+2}^d$ such that $-\Delta u_n = f_n$.

This idea has been explored in the mathematical literature, see e.g.

$$\texttt{www.chebfun.org},$$

but it is not used very often in the applied sciences and industry.

Finite Differences

Representing functions (continued)

Option 2: Polynomials (continued)

Possible reasons for this include:

- ▶ Using polynomials effectively is quite challenging.

  This circumstance is apparent e.g. in Runge's phenomenon (see
  https://demonstrations.wolfram.com/RungesPhenomenon).

- ▶ Polynomials do not allow for adaptive approximation.

  In many real-world applications, $u(x)$ is "simple" in most of Ω but
  varies rapidly in some small regions.
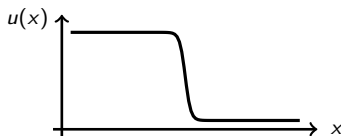
  Polynomial approximation of such functions requires large degrees,
  and increasing the polynomial degree increases the computational
 burden everywhere even though $u(x)$ is simple in most of Ω.

  This indicates that polynomials are often not the most efficient tool
  for solving partial differential equations.

Finite Differences

Representing functions (continued)

Option 3: Piecewise polynomials
The aforementioned problems can be avoided by splitting Ω into many small pieces and using a low-degree polynomial on each such piece. This approach is known as the finite element method (FEM) and the current state of the art for solving complicated real-world PDEs.


https://www.simscale.com/blog/2016/10/what-is-finite-element-method/
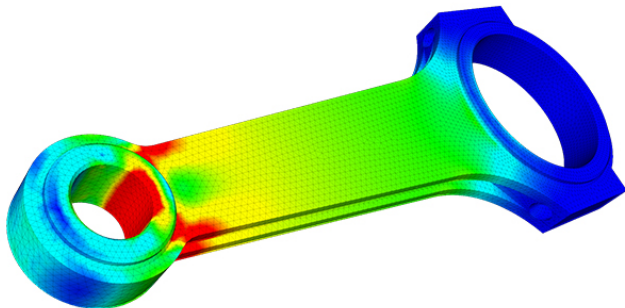
Finite Differences

Representing functions (continued)

Option 3: Piecewise polynomials (continued)

The main obstacle to using piecewise polynomials for solving partial differential equations is to make sense of the derivatives of $u(x)$ at the intersection between two neighbouring pieces where $u(x)$ may fail to be differentiable.
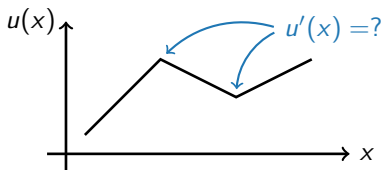
Answering this question requires the notion of almost-everywhere-defined functions introduced on slide 12 which I declared to be beyond the scope of this module. Correspondingly, I will not discuss the finite element method here.
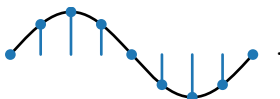
Finite Differences

Representing functions (continued)

Option 4: Point values

The issues of the function handle approach from slide 20 are ultimately due to the fact that `Float64` contains far more points than we can reasonably handle.

Once formulated this way, a possible solution to the problem presents itself: instead of sampling $u(x)$ at the resolution of `Float64` numbers, we can artificially restrict the number of samples to some manageable $n \in \mathbb{N}$ so we end up with only a manageable number of unknown point values

$$\left(\, u[i] = u(x_i) \, \right)_{i=1}^{n} \qquad \longleftrightarrow \qquad$$.

This is the representation that I will be using for the remainder of this lecture, so let us establish some notation and conventions regarding this representation.

Finite Differences

Square brackets for array indexing

The ith entry of a vector $v \in \mathbb{R}^n$ is usually denoted by v_i, but this subscript notation can be hard to read in formulae like $A_{i+1;(j-1)/2}$.

Instead, I will indicate array indexing using square brackets like in Julia, Python, R or C++.

Example. Instead of $A_{i+1;(j-1)/2}$, I will write $A[i+1, (j-1)/2]$.

Functions vs. vectors of point values

Throughout this lecture, I will use the same symbol u to denote both the function $u : \Omega \to \mathbb{R}$ and the vector of point values $u \in \mathbb{R}^n$.

It turns out that this is not ambiguous because it will always be clear from context whether a particular occurrence of u refers to the function $u(x)$ or the associated vector of point values $u[i] = u(x_i)$.

Example. Square brackets $[\cdot]$ indicate array indexing, so the u in $u[i]$ must refer to the vector-of-point-values interpretation of u. Similarly, parentheses $(\cdot)$ indicate function evaluation, so the u in $u(x)$ must refer to the function interpretation of u.

Finite Differences

Grids

Going from a function $u : \Omega \to \mathbb{R}$ to a vector of point values $u \in \mathbb{R}^n$ requires us to choose points x_i so we can relate the function and vector of point values using the formula $u[i] = u(x_i)$.

A collection $(x_i)_{i=1}^n$ of such points is called a *grid*.

For the sake of simplicity and concreteness, I will only consider the grid

$$x_i = \frac{i}{n+1} \qquad \xrightarrow{(n=3)}$$

in this lecture. This choice is based on several assumptions presented on the next slide.

Finite Differences

Grids (continued)

Assumptions underlying the grid

$$x_i = \frac{i}{n+1} \qquad \xrightarrow{(n=3)}$$

0.0	0.25	0.5	0.75	1.0
x_0	x_1	x_2	x_3	x_4

- ▶ The domain of the PDE is $\Omega = [0,1]$.
- ▶ We have homogeneous Dirichlet BC, i.e. $u(0) = u(1) = 0$.
- ▶ We want $x_a = 0$, $x_b = 1$ to be grid points so we can implement the boundary conditions by requiring that $u[a] = u[b] = 0$.
- ▶ We want the x_i to be equally spaced.
- ▶ We want the "unknown" entries of $u[i]$ to be indexed by $\{1, \ldots, n\}$ so our notation is consistent with array indexing in Julia.
  (The "unknown" qualifier serves to exclude $u[0] = u(0)$ and $u[n+1] = u(1)$ since these entries are fixed by the BC.)

Finite Differences

Grids (continued)

A few variations may help to further clarify the above.

▶ If the domain of the PDE was $\Omega = [-1, 1]$ instead of $\Omega = [0, 1]$, then we would choose the grid

$$x_i = -1 + 2\frac{i}{n+1} \qquad \xrightarrow{(n=3)}$$

▶ If we used Neumann instead of Dirichlet boundary conditions, then the two boundary values $u(0)$ and $u(1)$ are also unknown and hence we would change the grid to

$$x_i = \frac{i-1}{n-1} \qquad \xrightarrow{(n=5)}$$

Finite Differences

Numerically solving Poisson's equation

Now that we have established that we want to represent functions $u : [0, 1] \to \mathbb{R}$ as vectors $u \in \mathbb{R}^n$, we can start looking into how to implement the `solve_poisson(f)` function.

A good starting point for this endeavour is to ask the following:

> If someone else were to implement `solve_poisson(f)` for us, how would we convince ourselves that their implementation is correct?

Clearly, we would consider a function `u = solve_poisson(f)` to be correct if its output satisfies $-\Delta u = f$, but at this point this condition does not makes sense because we have just decided that `u` is now a vector of point values rather than a twice-differentiable function.

To overcome this issue, we will next look into how we can approximate $\Delta u = u''$ given only the vector of point values $u \in \mathbb{R}^n$.
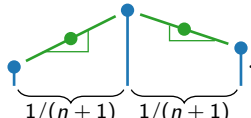
Finite Differences

Numerically solving Poisson's equation (continued)

The derivative of a function $u : \mathbb{R} \to \mathbb{R}$ is given by

$$u'(x) = \lim_{\Delta x \to 0} \frac{u(x + \Delta x/2) - u(x - \Delta x/2)}{\Delta x}.$$

It therefore seems likely that the best we can do to approximate this quantity given the point values $u[i] = u\left(\frac{i}{n+1}\right)$ is to set

$$u'\left(\tfrac{i+1/2}{n+1}\right) \approx \frac{u\left(\frac{i+1}{n+1}\right) - u\left(\frac{i}{n+1}\right)}{1/(n+1)} \qquad \longleftrightarrow$$

Iterating this idea for the second derivative, we obtain

$$
\begin{aligned}
u''\left(\tfrac{i}{n+1}\right) &\approx \frac{u'\left(\frac{i+1/2}{n+1}\right) - u'\left(\frac{i-1/2}{n+1}\right)}{1/(n+1)} \\
&= (n+1)^2 \left(u\left(\tfrac{i+1}{n+1}\right) - 2\,u\left(\tfrac{i}{n+1}\right) + u\left(\tfrac{i-1}{n+1}\right) \right).
\end{aligned}
$$

Finite Differences

Numerically solving Poisson's equation (continued)

The above is an approximation to $u''(x)$ based on only the available point values $u\left(\frac{i}{n+1}\right)$ as desired. We can make this point even clearer by replacing the point values in

$$u''\left(\tfrac{i}{n+1}\right) \approx (n+1)^2 \left(u\left(\tfrac{i+1}{n+1}\right) - 2\,u\left(\tfrac{i}{n+1}\right) + u\left(\tfrac{i-1}{n+1}\right)\right)$$

with their corresponding vector entries, which yields

$$u''[i] \approx (n+1)^2 \left(u[i-1] - 2\,u[i] + u[i+1]\right),$$

and we can shorten our notation by realising that the above is just a matrix-vector product $u'' \approx \Delta_n u$ where the matrix $\Delta_n \in \mathbb{R}^{n \times n}$ is given by

$$\Delta_n = (n+1)^2 \begin{pmatrix} \text{-2} & 1 & & & \\ 1 & \text{-2} & 1 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & \text{-2} \end{pmatrix}.$$

The findings of the last two slides can hence be summarised as follows:

> We want `u = solve_poisson(f)` to be such that $-\Delta_n u = f$.

Finite Differences

Terminology: Discrete Laplacian

I will call the matrix Δ_n introduced above the *discrete Laplacian* or *Laplace matrix* (as opposed to continuous Laplacian / Laplace operator).

Terminology: Finite difference discretisation

Replacing $-\Delta u = f$ with $-\Delta_n u = f$ is known as the *finite difference discretisation* of Poisson's equation.

Numerically solving Poisson's equation (conclusion)

We arrived at the condition $-\Delta_n u = f$ by asking how we would recognise a "correct" implementation of `u = solve_poisson(f)`, but it turns out that $-\Delta_n u = f$ also immediately tells how to implement this function: all we need to do is assemble and solve this linear system of equations. We hence conclude:

> Solving Poisson's equation using the finite difference
> method amounts to assembling and solving $-\Delta_n u = f$.

See also `solve_poisson_1d()` and `example_1d()`.

Finite Differences

Remark: Boundary conditions in the discrete Laplacian

You may have noticed that the left / blue 1 is missing in the first row of

$$\Delta_n = (n+1)^2 \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -2 \end{pmatrix}.$$

This is because the first row corresponds to the equation

$$u''[1] \approx (n+1)^2 \left(u[0] - 2\,u[1] + u[2] \right)$$

and the homogeneous Dirichlet boundary conditions specify that

$$u[0] = u(0) = 0.$$

The right / green 1 in the last row is missing for the same reason.

Finite Differences

Performance of finite differences

Now that we have an algorithm for solving Poisson's equations, let us next address the question: how fast is this algorithm?

Because the finite difference method depends on an effort parameter (the number of grid points n), the answer to this question comes in the form of two separate statements regarding the scaling of the runtime and error, respectively, as functions of the grid size n.

The following slides will elaborate on each of these statements in turn.

Finite Differences

Thm: Runtime of finite differences

Assembling and solving $-\Delta_n u = f$ requires at most $O(n^3)$ operations.

Proof. We observe:

- $f \in \mathbb{R}^n$ can be assembled in $O(n)$ runtime.
- $-\Delta_n \in \mathbb{R}^{n \times n}$ can be assembled in $O(n^2)$ runtime.
- The $n \times n$ linear system $-\Delta_n u = f$ can be solved in $O(n^3)$ runtime using the LU factorisation.

The overall runtime is hence dominated by the $O(n^3)$ operations required for the LU factorisation.

Finite Differences

Remark: Exploiting the zeros in Δ_n

We will see later that $-\Delta_n u = f$ can actually be solved in much fewer than $O(n^3)$ operations by exploiting that most entries of

$$\Delta_n = (n+1)^2 \begin{pmatrix} \text{-2} & 1 & & & \\ 1 & \text{-2} & 1 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & \text{-2} \end{pmatrix}$$

are zero, but we will also see that doing so requires quite a bit of work. The above is hence only a stop-gap statement until we find the time to look into more advanced algorithms.

Finite Differences

Error measure for finite differences

Our goal in the following is to derive a statement of the form

$$\|u_n - u\| = O\big(f(n)\big)$$

where $u_n \in \mathbb{R}^n$ denotes the solutions to the discretised Poisson equation $-\Delta_n u_n = f$ and $u : [0,1] \to \mathbb{R}$ denotes the solution to the continuous Poisson equation $-\Delta u = f$.

However, before we can do so we must first settle on a norm $\|\cdot\|$ in which to measure the difference between u_n and u.

Finite Differences

Error measure for finite differences (continued)

Since the discrete solutions $u_n \in \mathbb{R}^n$ are vectors, it may seem natural to measure this difference using the well-known Euclidian / 2-norm

$$\|u\|_2 = \sqrt{\sum_{i=1}^{n} u[i]^2},$$

but this choice of norm leads to some undesirable consequences: if two solutions $u_n \in \mathbb{R}^n$ and $u_{2n} \in \mathbb{R}^{2n}$ have roughly the same pointwise error, i.e.

$$\left| u_n[i] - u\left(\tfrac{i}{n+1}\right) \right| \approx \varepsilon \qquad \text{and} \qquad \left| u_{2n}[i] - u\left(\tfrac{i}{2n+1}\right) \right| \approx \varepsilon,$$

then their respective error 2-norms are

$$\left\| u_n - u \right\|_2 \approx \sqrt{n}\,\varepsilon \qquad \text{and} \qquad \left\| u_{2n} - u \right\|_2 \approx \sqrt{2n}\,\varepsilon,$$

i.e. the 2-norm of the error in u_{2n} is $\sqrt{2}$ times larger than the 2-norm of the error in u_n even though u_n and u_{2n} are arguably roughly equally accurate. Measuring errors in the 2-norm thus puts an unfair disadvantage on approximate solutions u_n which sample in a larger number of points n.

Finite Differences

Error measure for finite differences (continued)

I eliminate this issue by measuring errors using a *weighted* 2-norm $\| \cdot \|_{2,n}$ defined as follows.

Def: Weighted 2-norm

Given $u \in \mathbb{R}^n$, we define

$$\|u\|_{2,n} = \tfrac{1}{\sqrt{n+1}} \sqrt{\sum_{i=1}^n u[i]^2} = \frac{\|u\|_2}{\sqrt{n+1}}.$$

This choice of norm can be interpreted as a trapezoidal-rule approximation to

$$\|u\|_{L^2([0,1])} = \sqrt{\int_0^1 u(x)^2 \, dx}.$$

Finite Differences

Error measure for finite differences (conclusion)

Now that we have settled the choice of norm, we are ready to formulate a convergence result for the finite difference method.

Thm: Convergence of finite differences

Let $u : [0, 1] \to \mathbb{R}$ be the solution to the continuous Poisson equation $-\Delta u = f$ with homogeneous Dirichlet boundary conditions, and let $u_n \in \mathbb{R}^n$ be the solution to the discretised Poisson equation $-\Delta_n u_n = f$. Then,

$$\|u - u_n\|_{2,n} = O(n^{-2}) \qquad \text{for } n \to \infty.$$

The proof of this result will make use of the notion of induced matrix norms introduced on the next slide.

Finite Differences

Thm: Induced matrix norm

Let $\| \cdot \|$ be a norm on $\mathbb{R}^n$. Then

$$\|A\| = \sup_{x \in \mathbb{R}^n \setminus \{0\}} \frac{\|Ax\|}{\|x\|}$$

is a norm on the space of matrices $A \in \mathbb{R}^{n \times n}$.

Proof. Omitted.

Induced matrix norms are usually written using the same symbol as their inducing vector norms. For example, I will write $\|A\|_2$ to denote the matrix norm induced by the vector 2-norm, and I will write $\|A\|_{2,n}$ to denote the matrix norm induced by the weighted vector 2-norm.

Lemma: $\|A\|_2 = \|A\|_{2,n}$.

Proof.

$$\|A\|_{2,n} = \sup_{u \in \mathbb{R}^n} \frac{\|Au\|_{2,n}}{\|u\|_{2,n}} = \sup_{u \in \mathbb{R}^n} \frac{\frac{1}{\sqrt{n+1}} \|Au\|_2}{\frac{1}{\sqrt{n+1}} \|u\|_2} = \sup_{u \in \mathbb{R}^n} \frac{\|Au\|_2}{\|u\|_2} = \|A\|_2.$$

Finite Differences

The proof of the converge estimate of the finite difference method will further make use of the following auxiliary result.

Stability and consistency lemma

Let $u : [0,1] \to \mathbb{R}$ be the solution to $-\Delta u = f$, and let $u_n \in \mathbb{R}^n$ be the solution to $-\Delta_n u_n = f$. Then,

$$\|u - u_n\|_{2,n} \leq \|\Delta_n^{-1}\|_{2,n} \|\Delta_n u + f\|_{2,n}.$$

Proof.
$$\begin{aligned}
\|u - u_n\|_{2,n} &= \|\Delta_n^{-1} \left(\Delta_n u - \Delta_n u_n\right)\|_{2,n} \\
&= \|\Delta_n^{-1} \left(\Delta_n u + f\right)\|_{2,n} \\
&\leq \|\Delta_n^{-1}\|_{2,n} \|\Delta_n u + f\|_{2,n}
\end{aligned}$$

where on the last line I used that $\|Ax\| \leq \|A\| \|x\|$, which is an immediate consequence of the definition of induced matrix norms.

Finite Differences

Stability factor and consistency error

The power of the lemma on the previous slide is that it allows us to estimate the error in u_n by estimatig separately the two factors $\|\Delta_n^{-1}\|_{2,n}$ and $\|\Delta_n u + f\|_{2,n}$. These factors have a fairly intuitive meaning.

- $\|\Delta_n^{(-1)}\|_{2,n}$ provides an upper bound on how much the solution u_n to $-\Delta_n u_n = f$ can change for a given change in the right-hand side f: if u_n and v_n solve the discrete Poisson equations

$$-\Delta_n u_n = f \qquad \text{and} \qquad -\Delta_n v_n = g,$$

  then the difference between u_n and v_n is upper bounded by

$$\|u_n - v_n\|_{2,n} \leq \|\Delta_n^{-1}\|_{2,n} \, \|f - g\|_{2,n}.$$

  For this reason, $\|\Delta_n^{-1}\|_{2,n}$ is called the *stability factor*.

- $\|\Delta_n u + f\|_{2,n}$ quantifies how well the solution u to the continuous problem satisfies the discretised equation: if $-\Delta_n u = f$ were true, then we would have $\|\Delta_n u + f\|_{2,n} = 0$.

  For this reason, $\|\Delta_n u + f\|_{2,n}$ is called the *consistency error*.

Finite Differences

I will next prove the finite difference convergence estimate from slide 42 by analysing separately the scaling of the stability factor $\|\Delta_n^{-1}\|_{2,n}$ and the consistency error $\|\Delta_n u + f\|_{2,n}$.

Theorem: Stability of Δ_n

We have $\|\Delta_n^{-1}\|_{2,n} = O(1)$ for $n \to \infty$.

Proof. Δ_n is a symmetric matrix; hence according to the lemma from slide 43 and standard results from linear algebra, we have

$$\|\Delta_n^{-1}\|_{2,n} = \|\Delta_n^{-1}\|_2 = |\lambda_{\min}^{-1}|$$

where $\lambda_{\min}$ denotes the eigenvalue of smallest absolute value of Δ_n. We can thus estimate the stability factor $\|\Delta_n^{-1}\|_{2,n}$ by determining the eigenvalues of Δ_n. This problem will be addressed on the following slides.

Finite Differences

In order to determine the eigenvalues of Δ_n, it is useful to determine the eigenvalues of its continuous analogue Δ first.

Thm: Eigenpairs of continuous Laplace operator

The eigenpairs of the continuous Laplace operator with homogeneous Dirichlet boundary conditions on $(0, 1)$ are given by

$$\left(\quad \lambda_k = -\pi^2 k^2, \quad v_k(x) = \sin(\pi kx) \quad \right)_{k=1}^{\infty}.$$

Proof. The above (λ_k, v_k) satisfy the eigenvalue equation

$$v_k''(x) = \tfrac{\partial^2}{\partial x^2} \sin(\pi kx) = \pi k \tfrac{\partial}{\partial x} \cos(\pi kx) = -\pi^2 k^2 \sin(\pi kx) = \lambda_k v_k(x),$$

and the v_k additionally satisfy the boundary condition $v_k(0) = v_k(1) = 0$. The latter property is what sets the above functions apart from e.g. $w_k(x) = \cos(\pi kx)$ which also satisfy $w_k''(x) = \lambda_k w_k(x)$ but for which we have $w_k(0) = w_k(1) = \pm 1$.

Note that I do not prove here that the above are all the eigenpairs of Δ. Doing so is beyond the scope of this module.

Finite Differences

The above result leads us to the educated guess that $v_k[i] = \sin\left(\pi k \frac{i}{n+1}\right)$ may be the eigenvectors of Δ_n. This is indeed the case.

Thm: Eigenpairs of the discrete Laplace matrix

The n eigenpairs of Δ_n are given by

$$\left(\quad \lambda_k = 2\,(n+1)^2 \left(\cos\left(\pi \frac{k}{n+1}\right) - 1\right), \quad v_k[i] = \sin\left(\pi k \frac{i}{n+1}\right) \quad \right)_{k=1}^n.$$

Proof. The above statement makes two related but distinct claims:

▶ The pairs (λ_k, v_k) satisfy the eigenvalue equation $\Delta_n v_k = \lambda_k v_k$.

▶ These are all eigenpairs, i.e. if there is any other pair $(\lambda, v \neq 0)$ such that $\Delta_n v = \lambda v$ then $\lambda \in \{\lambda_k\}_{k=1}^n$ and $v \in \text{span}\{v_k\}_{k=1}^n$.

The second property is important for our purposes because we are looking for the eigenvalue of smallest absolute value and it is possible that we are missing this smallest eigenvalue unless we have found all of them.

Finite Differences

Proof of eigenpairs of Δ_n (continued).

Let us first show that the pairs (λ_k, v_k) indeed satisfy the eigenvalue equation $\Delta_n v_k = \lambda_k v_k$. We have seen on slide 33 that

$$(\Delta_n v_k)[i] = (n+1)^2 \left(v_k[i-1] - 2\,v_k[i] + v_k[i+1] \right) \qquad (1)$$

for all $i \in \{1, \ldots, n\}$ except for $i = 1$ where the $v[i-1]$ term is missing and $i = n$ where the $v[i+1]$ term is missing. However, for $i = 1$ we have

$$v_k[i-1] = v_k[0] = \sin\left(\pi k \tfrac{0}{n+1}\right) = 0,$$

and likewise for $i = n$ we have

$$v_k[i+1] = v_k[n+1] = \sin\left(\pi k \tfrac{n+1}{n+1}\right) = 0.$$

We therefore conclude that (1) holds for all $i \in \{1, \ldots, n\}$, and it remains to show that

$$(\Delta_n v_k)[i] = (n+1)^2 \left(v_k[i-1] - 2\,v_k[i] + v_k[i+1] \right) = \lambda_k\,v_k[i].$$

I will do so on the next slide.

Finite Differences

Proof of eigenpairs of Δ_n (continued).

$$(\Delta_n v_k)[j] = (n+1)^2 \left(v_k[j-1] - 2v_k[j] + v_k[j+1] \right)$$

$$= (n+1)^2 \left(\sin\!\left(\pi k \tfrac{j-1}{n+1}\right) - 2\sin\!\left(\pi k \tfrac{j}{n+1}\right) + \sin\!\left(\pi k \tfrac{j+1}{n+1}\right) \right)$$

$$= \frac{(n+1)^2}{2i} \left(e^{\pi i k \frac{j-1}{n+1}} - e^{-\pi i k \frac{j-1}{n+1}} - 2e^{\pi i k \frac{j}{n+1}} + 2e^{-\pi i k \frac{j}{n+1}} + \ldots \right.$$
$$\left. e^{\pi i k \frac{j+1}{n+1}} - e^{-\pi i k \frac{j+1}{n+1}} \right)$$

$$= \frac{(n+1)^2}{2i} \left(e^{\pi i k \frac{j}{n+1}} \left(e^{-\pi i k \frac{1}{n+1}} - 2 + e^{\pi i k \frac{1}{n+1}} \right) - \ldots \right.$$
$$\left. e^{-\pi i k \frac{j}{n+1}} \left(e^{\pi i k \frac{1}{n+1}} - 2 + e^{-\pi i k \frac{1}{n+1}} \right) \right)$$

$$= \frac{(n+1)^2}{2i} \left(e^{-\pi i k \frac{1}{n+1}} - 2 + e^{\pi i k \frac{1}{n+1}} \right) \left(e^{\pi i k \frac{j}{n+1}} - e^{-\pi i k \frac{j}{n+1}} \right)$$

$$= (n+1)^2 \left(2\cos\!\left(\pi \tfrac{k}{n+1}\right) - 2 \right) \sin\!\left(\pi k \tfrac{j}{n+1}\right)$$

$$= \lambda_k \, v_k[j]$$

Finite Differences

Proof of eigenpairs of Δ_n (continued).

I now move on to showing the second claim, namely that $(\lambda_k, v_k)_{k=1}^n$ are indeed all eigenpairs of Δ_n.

This problem can be reduced to showing that

1. $(\lambda_k)_{k=1}^n$ are distinct, and
2. $v_k \neq 0$ for all $k \in \{1, \ldots, n\}$

by recalling that eigenvectors paired with distinct eigenvalues are linearly independent and linear independence implies that $\text{span}\{v_k\}_{k=1}^n = \mathbb{R}^n$.

(Explicitly checking that $v_k \neq 0$ is necessary because $(\lambda, 0)$ is by definition not an eigenpair and hence distinctness of the λ_k does not rule out that at least some of the associated vectors v_k could be equal to 0.)

Finite Differences

Proof of eigenpairs of Δ_n (continued).

We observe:

1. $\lambda_k = 2\,(n+1)\left(\cos(\pi\frac{k}{n+1}) - 1\right)$ are distinct because the points $x_k = \frac{k}{n+1}$ are distinct and $\cos(\pi x_k) - 1$ is monotonous for $x_k \in [0, 1]$.
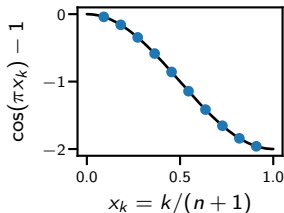
2. $v_k[i] = \sin\left(\pi k\frac{i}{n+1}\right) \neq 0$ as long as k is not a multiple of $n+1$.

$(\lambda_k, v_k)_{k=1}^{n}$ are hence indeed all eigenpairs of Δ_n, and the proof of the result on slide 48 is complete.

Finite Differences

Terminology: Fourier analysis

The above technique of guessing that $v_k[i] = \sin\left(\pi k \frac{i}{n+1}\right)$ should be the eigenvectors of Δ_n and then verifying our guess through explicit calculations is known as *Fourier analysis*.

Finite Differences

Proof of $\|\Delta_n^{-1}\|_{2,n} = O(1)$ (continued from slide 46)

Recall that our motivation for determining the eigenpairs of Δ_n was to show that $\|\Delta_n^{-1}\|_{2,n} = |\lambda_{\min}| = O(1)$ as claimed on slide 46. We now have all the ingredients in place to complete the proof of this claim:

▶ It follows from the plot on slide 52 that the eigenvalue of smallest absolute value of Δ_n is obtained by setting $k = 1$ in the formula

$$\left(\quad \lambda_k = 2\,(n+1)^2 \left(\cos\left(\pi \tfrac{k}{n+1}\right) - 1 \right) \quad \right)_{k=1}^n.$$

▶ Using the Taylor series expansion $\cos(x) = 1 - \frac{x^2}{2!} + O(x^4)$ we obtain

$$\lambda_1 = 2\,(n+1)^2 \left(1 - \tfrac{1}{2!} \tfrac{\pi^2}{(n+1)^2} + O(n^{-4}) - 1 \right) = -\pi^2 + O(n^{-2})$$

and hence $\quad \|\Delta_n^{-1}\|_{2,n} = \pi^{-2} + O(n^{-2}) = O(1) \quad$ for $n \to \infty$.

This completes the proof of the $\|\Delta_n^{-1}\|_{2,n} = O(1)$ estimate from slide 46.

Finite Differences

I now move on to showing consistency of the discrete Poisson equation.

Thm: Consistency of the discrete Poisson equation

Assume u is a four times differentiable solution to the continuous Poisson equation $-\Delta u = f$ on $\Omega = [0, 1]$ with homogeneous Dirichlet boundary conditions $u(0) = u(1) = 0$. Then,

$$\|\Delta_n u + f\|_{2,n} = O(n^{-2}) \qquad \text{for } n \to \infty.$$

Proof. See next slide.

Finite Differences

Proof of $\|\Delta_n u + f\|_{2,n} = O(n^{-2})$.

According to Taylor's theorem, we have

$$u\left(\tfrac{i\pm 1}{n+1}\right) = u\left(\tfrac{i}{n+1}\right) \pm u'\left(\tfrac{i}{n+1}\right)\tfrac{1}{n+1} + \tfrac{1}{2!}u''\left(\tfrac{i}{n+1}\right)\tfrac{1}{(n+1)^2} + \ldots$$
$$\pm \tfrac{1}{3!}u'''\left(\tfrac{i}{n+1}\right)\tfrac{1}{(n+1)^3} + O\left(n^{-4}\right)$$

and hence

$$\begin{aligned}
(\Delta_n u)[i] &= (n+1)^2 \left(u\left(\tfrac{i-1}{n+1}\right) - 2\,u\left(\tfrac{i}{n+1}\right) + u\left(\tfrac{i+1}{n+1}\right) \right) \\
&= (n+1)^2 \Big((1-2+1)\,u\left(\tfrac{i}{n+1}\right) + \ldots \\
&\qquad\qquad (-1+1)\,u'\left(\tfrac{i}{n+1}\right)\tfrac{1}{n+1} + \ldots \\
&\qquad\qquad (1+1)\,\tfrac{1}{2!}u''\left(\tfrac{i}{n+1}\right)\tfrac{1}{(n+1)^2} + \ldots \\
&\qquad\qquad (-1+1)\,\tfrac{1}{3!}u'''\left(\tfrac{i}{n+1}\right)\tfrac{1}{(n+1)^3} + O\left(n^{-4}\right) \Big) \\
&= u''(\tfrac{i}{n+1}) + O\left(n^{-2}\right).
\end{aligned}$$

(Note that these computations are correct even for $i \in \{1, n\}$ because

$$u[0] = u(0) = u[n+1] = u(1) = 0$$

according to the Dirichlet boundary conditions.)

Finite Differences

Proof of $\|\Delta_n u + f\|_{2,n} = O(n^{-2})$ (continued).
Combining the above with the assumption that $-\Delta u = -u'' = f$, we conclude that

$$\|\Delta_n u + f\|_{2,n} = \|\Delta_n u - u''\|_{2,n} = \sqrt{\tfrac{n}{n+1}\, O\big(n^{-2}\big)^2} = O(n^{-2})$$

as claimed.

Proof of $\|u - u_n\|_{2,n} = O(n^{-2})$ (see slide 42).
Finally, we can show the convergence estimate from slide 42 by combining the stability and consistency lemma from slide 44 with the stability estimate from slide 46 and the consistency estimate from slide 55 to conclude that

$$\|u - u_n\|_{n,2} \le \|\Delta_n^{-1}\|_{2,n}\, \|\Delta_n u + f\|_{2,n} = O(1)\, O\big(n^{-2}\big) = O(n^{-2})$$

as claimed.

Finite Differences

Summary of finite difference convergence proof
The proof of the finite difference convergence estimate from slide 42 is somewhat convoluted. Here is a high-level summary of the argument for your convenience.

▶ Stability and consistency lemma (slide 44):

$$\|u - u_n\|_{2,n} \le \|\Delta_n^{-1}\|_{2,n} \, \|\Delta_n u + f\|_{2,n}.$$

   ▶ Stability estimate $\|\Delta_n^{-1}\|_{2,n} = O(1)$ (slide 46)
      ▶ Eigenvalues of Δ_n via Fourier analysis (slide 48).
   ▶ Consistency estimate $\|\Delta_n u + f\|_{2,n} = O(n^{-2})$ via Taylor series (slide 55).

Numerical demonstration of convergence estimate
See `convergence_1d()`.

Finite Differences

The energy method for showing stability

Using Fourier analysis to show stability of Δ_n amounts to doing too much work in some sense: all that is needed to estimate $\|\Delta_n^{-1}\|_{2,n}$ is the eigenvalue of smallest absolute value $\lambda_{\min}$, but by proceeding via Fourier analysis we end up computing *all* eigenvalues of Δ_n rather than only this smallest one.

One consequence of this fact that Fourier analysis gives us more information than what we need is that Fourier analysis no longer applies once we move from the very simple setting considered above to more complicated settings where e.g.

- ▶ the grid is not uniformly spaced,
- ▶ the domain $\Omega \subset \mathbb{R}^d$ is not of the form $\Omega = [a, b]^d$, or
- ▶ the diffusion coefficients D in the generalised Poisson equation $-\nabla \cdot (D \nabla u) = f$ is not spatially constant, i.e. $D = D(x) \neq$ const.

I decided to at first establish the stability of Δ_n via Fourier analysis because understanding the eigenvalues of Δ_n will be important in the later lecture on Krylov subspace methods. However, the above indicates that other, more targeted techniques for showing stability will be useful as well. Deriving one such technique will be the topic of the following slides.

Finite Differences

The energy method for showing stability (continued)

Our alternative technique for showing stability of Δ_n will be based on the following result.

Rayleigh quotient theorem

The smallest eigenvalue $\lambda_{\min}$ of a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is given by

$$\lambda_{\min} = \min_{v \in \mathbb{R}^n} \frac{v^T A v}{v^T v}.$$

Proof. See any linear algebra textbook.

Terminology: Rayleigh quotient

The ratio $v^T A v / (v^T v)$ is called the *Rayleigh quotient* of (A, v).

Finite Differences

The energy method for showing stability (continued)

Aiming to estimate the smallest eigenvalue of $-\Delta_n$ via the Rayleigh quotient theorem, I compute

$$
\begin{aligned}
v^T(-\Delta_n)\,v &= (n+1)^2 \sum_{i=1}^{n} \Big(-v[i]\,v[i-1] + 2\,v[i]^2 - v[i]\,v[i+1] \Big) \\
&= (n+1)^2 \sum_{i=1}^{n+1} \Big(v[i]^2 - 2\,v[i]\,v[i-1] + v[i-1]^2 \Big) \\
&= (n+1)^2 \sum_{i=1}^{n+1} \Big(v[i] - v[i-1] \Big)^2,
\end{aligned}
$$

where as usual I assume $v[0] = v[n+1] = 0$.

Note that the step from first to second line corresponds to going from

 to .
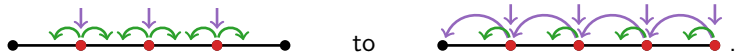
Finite Differences

The energy method for showing stability (continued)

In order to use the above result to estimate the Rayleigh quotient

$$\frac{v^T(-\Delta_n)\,v}{v^T v},$$

we next need to relate the sum of squared differences in the numerator

$$v^T(-\Delta_n)\,v = (n+1)^2 \sum_{i=1}^{n+1}\Big(v[i] - v[i-1]\Big)^2$$

with the sum of squared values in the denominator

$$v^T v = \sum_{i=1}^{n} v[i]^2.$$

This is tackled on the next slide.

Finite Differences

The energy method for showing stability (continued)

Using a telescopic sum argument and setting $v[0] = 0$, we obtain

$$v[i] = \sum_{j=1}^{i} \big(v[j] - v[j-1]\big) \leq \sum_{j=1}^{i} \big|v[j] - v[j-1]\big|$$

$$\leq \sum_{j=1}^{n+1} \big|v[j] - v[j-1]\big| = \big\|v[j] - v[j-1]\big\|_1$$

$$\leq \sqrt{n+1} \, \big\|v[j] - v[j-1]\big\|_2$$

$$= \sqrt{(n+1) \sum_{j=1}^{n+1} \Big(v[j] - v[j-1]\Big)^2}.$$

Inserting this result into the formula for $v^T v$, we conclude

$$v^T v = \sum_{i=1}^{n} v[i]^2 \leq (n+1)^2 \sum_{j=1}^{n+1} \Big(v[j] - v[j-1]\Big)^2.$$

Finite Differences

The energy method for showing stability (continued)

Combining all of the above, we conclude that

$$v^T(-\Delta_n)\,v \;=\; (n+1)^2 \sum_{i=1}^{n+1}\Big(v[i] - v[i-1]\Big)^2 \;\geq\; v^T v$$

and hence

$$\lambda_{\min} = \min_{v\in\mathbb{R}^n} \frac{v^T(-\Delta_n)\,v}{v^T v} \geq 1 \qquad \text{and} \qquad \|\Delta_n^{-1}\|_{2,n} = |\lambda_{\min}|^{-1} \leq 1.$$

We have thus succeeded in showing stability of Δ_n without computing the eigenvalues of Δ_n.

This technique for showing stability is known as the *energy method* because $v^T(-\Delta_n)\,v$ can often be related to the energy of the physical system under consideration.

Finite Differences

From one dimension to many dimensions

We have now completed the discussion of the finite difference method in one dimension.

I will next demonstrate how we can generalise this technique from one to two dimensions, and later on also to three dimensions.

Two-dimensional finite differences

A good starting point for deriving a two-dimensional finite difference method is to recapitulate how we derived the one-dimensional method.

We therefore recall that we obtained the one-dimensional method by

- ▶ deciding that we want to represent functions $u : [0, 1] \to \mathbb{R}$ as vectors of point values $u[i] = u\left(\frac{i}{n+1}\right)$,

- ▶ deriving a matrix Δ_n which mimics the action of the continuous Laplacian within this vector-of-point-values representation, and

- ▶ constructing a sequence of finite-difference approximations $u_n \in \mathbb{R}^n$ by demanding that these approximations satisfy $-\Delta_n u_n = f$.

I will now repeat this line of thought to derive a two-dimensional finite difference method.

Finite Differences

Two-dimensional finite differences (continued)

In one dimension, we discussed several options for numerically representing functions $u : [0, 1] \to \mathbb{R}$ and decided that the vector-of-point-values representation would be the most suitable for our purposes (see slide 20 and following).

Following the same reasoning in two dimensions, we conclude that we want to represent functions $u : [0, 1]^2 \to \mathbb{R}$ as matrices of point values

$$\left(\; u[i_1, i_2] = u\left(\tfrac{i_1}{n+1}, \tfrac{i_2}{n+1}\right) \; \right)_{i_1, i_2=1}^{n} \; \in \; \mathbb{R}^{n \times n}.$$

This formula corresponds to the following grid:
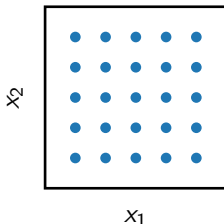
Finite Differences

Two-dimensional finite differences (continued)

In one dimension, we then moved on to derive

$$u''\left(\tfrac{i}{n+1}\right) \approx (n+1)^2 \left(u[i-1] - 2u[i] + u[i+1]\right)$$

as an approximation to $u''(x)$ within the framework of the vector-of-point-values representation. Applying this formula to the partial derivatives

$$\tfrac{\partial^2 u}{\partial x_1^2}\left(\tfrac{i_1}{n+1}, \tfrac{i_2}{n+1}\right) \approx (n+1)^2 \left(u[i_1-1, i_2] - 2u[i_1, i_2] + u[i_1+1, i_2]\right),$$

$$\tfrac{\partial^2 u}{\partial x_2^2}\left(\tfrac{i_1}{n+1}, \tfrac{i_2}{n+1}\right) \approx (n+1)^2 \left(u[i_1, i_2-1] - 2u[i_1, i_2] + u[i_1, i_2+1]\right),$$

we conjecture that the function $\Delta_n^{(2)} : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}$ given by

$$(\Delta_n^{(2)} u)[i_1, i_2] = (n+1)^2 \left(u[i_1-1, i_2] + u[i_1, i_2-1] - 4u[i_1, i_2] + u[i_1+1, i_2] + u[i_1, i_2+1]\right)$$

should be a reasonable approximation to the continuous Laplace operator

$$\Delta : \left([0,1]^2 \to \mathbb{R}\right) \to \left([0,1]^2 \to \mathbb{R}\right), \qquad \Delta u = \tfrac{\partial^2 u}{\partial x_1^2} + \tfrac{\partial^2 u}{\partial x_2^2}.$$

Finite Differences

Two-dimensional finite differences (conclusion)

Combining the above, we conclude:

> Finite differences in two dimensions amounts to
> finding $u_n \in \mathbb{R}^{n \times n}$ such that $-\Delta_n^{(2)} u_n = f$.

Notation: Laplacian matrices in different dimensions

From now on, I will distinguish discrete Laplacians $\Delta_n^{(d)}$ of different
dimensionalities d using a superscript $\Delta_n^{(d)}$ to avoid confusion.

Translating 2d finite differences into a linear system

As in one-dimensions, I would next like to interpret $-\Delta_n^{(2)} u_n = f$ as a
linear system in terms of the unknown point values $u_n \in \mathbb{R}^{n \times n}$.

However, doing so is not as straightforward as in one-dimension because
at this point u_n is a matrix rather than a vector and Δ_n is a linear
function on the space of $n \times n$ matrices rather than a matrix (i.e. a linear
function on the space of vectors).

Finite Differences

Translating 2d finite differences into a linear system (continued)

To overcome this issue, I will next introduce a bijective function

$$\text{vec} : \mathbb{R}^{m \times n} \to \mathbb{R}^{mn}$$

which maps matrices of point values $u \in \mathbb{R}^{n \times n}$ into vectors $\text{vec}(u) \in \mathbb{R}^{n^2}$. After that, I will derive a corresponding matricisation

$$\text{mat}\big(\Delta_n^{(2)}\big) \in \mathbb{R}^{n^2 \times n^2}$$

of the two-dimensional discrete Laplacian $\Delta_n^{(2)} : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}$ such that

$$\text{vec}\big(\Delta_n^{(2)} u_n\big) = \text{mat}(\Delta_n^{(2)}) \, \text{vec}(u_n).$$

This will then allow us to solve the two-dimensional finite difference problem $-\Delta_n^{(2)} u_n = f$ by applying standard linear system solvers to

$$-\text{mat}\big(\Delta_n^{(2)}\big) \, \text{vec}(u_n) = \text{vec}(f).$$

Finite Differences

Def: Vectorisation of a matrix

The *vectorisation* $\text{vec}(u) \in \mathbb{R}^{mn}$ of a matrix $u \in \mathbb{R}^{m \times n}$ is given by

$$\text{vec}(u)[i_1 + m\,(i_2 - 1)] = u[i_1, i_2].$$

Example 1

Sloppily speaking, the above formula says that $\text{vec}(u)$ stacks all the columns of u into one long vector. For example, we have

$$\text{vec} \begin{pmatrix} a & c \\ b & d \end{pmatrix} = \begin{pmatrix} a & b & c & d \end{pmatrix}^T.$$

Example 2

$\text{vec}(u)$ enumerates the entries of a 4×4 matrix u as follows:

$1 = 1 + 4 \times 0$	$5 = 1 + 4 \times 1$	$9 = 1 + 4 \times 2$	$13 = 1 + 4 \times 3$
$2 = 2 + 4 \times 0$	$6 = 2 + 4 \times 1$	$10 = 2 + 4 \times 2$	$14 = 2 + 4 \times 3$
$3 = 3 + 4 \times 0$	$7 = 3 + 4 \times 1$	$11 = 3 + 4 \times 2$	$15 = 3 + 4 \times 3$
$4 = 4 + 4 \times 0$	$8 = 4 + 4 \times 1$	$12 = 4 + 4 \times 2$	$16 = 4 + 4 \times 3$

Finite Differences

I now move on to deriving the matricisation $\mathrm{mat}(\Delta_n^{(2)})$ introduced above, and I will do so using the following operation.

Def: Kronecker product

The *Kronecker product* $A \otimes B \in \mathbb{R}^{(m_a m_b) \times (n_a n_b)}$ of two matrices

$$A \in \mathbb{R}^{m_A \times n_A}, \qquad B \in \mathbb{R}^{m_B \times n_B}$$

is given by

$$(A \otimes B)[i_B + m_B\,(i_A - 1),\ j_B + n_B\,(j_A - 1)] = A[i_A, j_A]\, B[i_B, j_B],$$

or put differently,

$$A \otimes B = \begin{pmatrix} A[1,1]\,B & \cdots & A[1, n_A]\,B \\ \vdots & \ddots & \vdots \\ A[m_A, 1]\,B & \cdots & A[m_A, n_A]\,B \end{pmatrix}.$$

Finite Differences

The Kronecker product will be useful for our purposes because of the following result.

Lemma

Let $A \in \mathbb{R}^{m_A \times n_A}$, $B \in \mathbb{R}^{m_B \times n_B}$ and $C \in \mathbb{R}^{n_B \times n_A}$. Then,

$$\text{vec}(BCA^T) = (A \otimes B)\,\text{vec}(C).$$

Proof (not examinable).

$$\text{vec}(BCA^T)[i_B + m_B\,(i_A - 1)]$$
$$= \sum_{j_B=1}^{n_B} \sum_{j_A=1}^{n_A} B[i_B, j_B]\, C[j_B, j_A]\, A[i_A, j_A]$$
$$= \sum_{j_B=1}^{n_B} \sum_{j_A=1}^{n_A} A[i_A, j_A]\, B[i_B, j_B]\, C[j_B, j_A]$$
$$= \sum_{j_B=1}^{n_B} \sum_{j_A=1}^{n_A} (A \otimes B)[i_B + m_B\,(i_A - 1),\; j_B + n_B\,(j_A - 1)]\, \text{vec}(C)[j_B + m_B\,(j_A - 1)]$$
$$= \Big((A \otimes B)\,\text{vec}(C)\Big)[i_B + m_B\,(i_A - 1)].$$

Finite Differences

Corollary: Matricisation of the two-dimensional discrete Laplacian

We have

$$\text{mat}\big(\Delta_n^{(2)}\big) = I_n \otimes \Delta_n^{(1)} + \Delta_n^{(1)} \otimes I_n,$$

i.e. the two-dimensional discrete Laplacian $\Delta_n^{(2)}$ from slide 67 satisfies

$$\text{vec}\big(\Delta_n^{(2)} u_n\big) = \big(I_n \otimes \Delta_n^{(1)} + \Delta_n^{(1)} \otimes I_n\big) \text{vec}(u_n).$$

$(I_n \in \mathbb{R}^{n \times n}$ denotes the $n \times n$ identity matrix.$)$

Proof. We observe that

$$\tfrac{\partial^2 u}{\partial x_1^2}\big(\tfrac{i_1}{n+1}, \tfrac{i_2}{n+1}\big) \approx (n+1)^2 \big(u[i_1 - 1, i_2] - 2u[i_1, i_2] + u[i_1 + 1, i_2]\big)$$

corresponds to the matrix product $\tfrac{\partial^2 u}{\partial x_1^2} \approx \Delta_n^{(1)} u$, and likewise

$$\tfrac{\partial^2 u}{\partial x_2^2}\big(\tfrac{i_1}{n+1}, \tfrac{i_2}{n+1}\big) \approx (n+1)^2 \big(u[i_1, i_2 - 1] - 2u[i_1, i_2] + u[i_1, i_2 + 1]\big)$$

corresponds to the matrix product $\tfrac{\partial^2 u}{\partial x_2^2} \approx u \Delta_n^{(1)}$.

Finite Differences

Proof (continued).
The two-dimensional Laplacian from slide 67 is thus given by

$$\Delta_n^{(2)} u_n = \Delta_n^{(1)} u_n + u_n \Delta_n^{(1)},$$

and hence we have according to the lemma on slide 72 that

$$
\begin{aligned}
\text{vec}\big(\Delta_n^{(2)} u_n\big) &= \text{vec}\big(\Delta_n^{(1)} u_n I_n\big) + \text{vec}\big(I_n u_n \Delta_n^{(1)}\big) \\
&= \big(I_n \otimes \Delta_n^{(1)}\big) \text{vec}(u_n) + \big(\Delta_n^{(1)} \otimes I_n\big) \text{vec}(u_n) \\
&= \underbrace{\big(I_n \otimes \Delta_n^{(1)} + \Delta_n^{(1)} \otimes I_n\big)}_{\text{mat}(\Delta_n^{(2)})} \text{vec}(u_n)
\end{aligned}
$$

Implementation of two-dimensional finite differences
See `solve_poisson_2d()` and `example_2d()`.

Finite Differences

Outlook

As in 1d, the theory of finite differences in two dimensions consists mainly of two parts:

- A heuristic argument why $-\Delta_n^{(2)} u_n = f$ could be a reasonable approximation to $-\Delta u = f$.
- A rigorous proof that the discrete solutions u_n converge to the continuous solution u with speed $O(n^{-2})$.

We have now completed the first part and will soon move on to the second part (see slide 77), but before we do so we should settle some notational issues first.

Finite Differences

Notation for two-dimensional finite differences

On slide 67, I introduced the two-dimensional discrete Laplacian as a function $\Delta_n^{(2)} : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}$.

This convention was intended mainly to help us get started with finite differences in two dimensions. From now on, $\Delta_n^{(2)}$ will always denote the Laplacian matrix

$$\Delta_n^{(2)} = I_n \otimes \Delta_n^{(1)} + \Delta_n^{(1)} \otimes I_n \quad \in \quad \mathbb{R}^{n^2 \times n^2}.$$

The rationale for doing so is that we will be studying algorithms for solving the linear system $-\Delta_n^{(2)} u_n = f$ in the upcoming lectures, and for this purpose it is much more convenient to work with a matrix $\Delta_n^{(2)} \in \mathbb{R}^{n^2 \times n^2}$ rather than a function $\Delta_n^{(2)} : \mathbb{R}^{n^2} \to \mathbb{R}^{n^2}$.

On the other hand, I will use the symbol u_n to denote both the matrix of point values $u_n \in \mathbb{R}^{n \times n}$ and its vectorisation $u_n \in \mathbb{R}^{n^2}$. Context will clarify the intended meaning.

Finite Differences

Convergence of two-dimensional finite differences

Let us now address the question of estimating the speed of convergence
of the two-dimensional finite difference method.

As in one dimension, the first step towards this goal is to choose a norm
for measuring errors.

Def: Weighted 2-norm for two-dimensional finite differences

Given $u \in \mathbb{R}^{n \times n}$, we define

$$\|u\|_{2,n} = \frac{1}{n+1} \sqrt{\sum_{i_1=1}^{n} \sum_{i_2=1}^{n} u[i_1, i_2]^2} = \frac{\|\text{vec}(u)\|_2}{n+1}.$$

As before, the purpose of the $1/(n+1)$ prefactor is to ensure that
$u \in \mathbb{R}^{n \times n}$ with larger n are not unfairly punished for sampling the error in
more points. Moreover, $\|u\|_{2,n}$ can again be interpreted as a
trapezoidal-rule approximation to

$$\|u\|_{L^2([0,1]^2)} = \sqrt{\int_0^1 \int_0^1 u(x_1, x_2) \, dx_1 \, dx_2}.$$

Finite Differences

We can now formulate the following result.

Thm: Convergence of 2d finite differences
Let $u : [0,1]^2 \to \mathbb{R}$ be the solution to the continuous Poisson equation
$-\Delta u = f$ with homogeneous Dirichlet boundary conditions, and let
$u_n \in \mathbb{R}^{n^2}$ be the solution to the discretised equation $-\Delta_n^{(2)} u_n = f$.
Then,
$$\|u - u_n\|_{2,n} = O(n^{-2}) \qquad \text{for } n \to \infty.$$

Proof. As in 1d, the proof of this result will be based on the stability and
consistency theorem from slide 44, i.e.

$$\|u - u_n\|_{2,n} \leq \left\|(\Delta_n^{(2)})^{-1}\right\|_{2,n} \left\|\Delta_n^{(2)} u + f\right\|_{2,n}.$$

I will therefore next show that $\Delta_n^{(2)}$ is stable (i.e. $\|\Delta_n^{(2)}\|_{2,n} = O(1)$) and
consistent (i.e. $\|\Delta_n^{(2)} u + f\|_{2,n} = O(n^{-2})$).
As before the proof of stability will proceed by determining all eigenvalues
of $\Delta_n^{(2)}$ and then estimating the smallest one.

Finite Differences

The following result will be useful to determine the eigenvalues of $\Delta_n^{(2)}$.

Lemma

Let $A \in \mathbb{R}^{m_a \times n_a}$, $B \in \mathbb{R}^{m_b \times n_b}$ and $a \in \mathbb{R}^{n_a}$, $b \in \mathbb{R}^{n_b}$. Then,

$$(A \otimes B)(a \otimes b) = (Aa) \otimes (Bb)$$

Proof (not examinable).

$$
\begin{aligned}
&\left((A \otimes B)(a \otimes b)\right)[i_B + m_B(i_A - 1)] \\
&\quad = \sum_{j_A=1}^{n_a} \sum_{j_B=1}^{n_b} (A \otimes B)[i_B + m_B(i_A-1), j_B + n_B(j_A-1)] \, (a \otimes b)[j_B + n_B(j_A-1)] \\
&\quad = \sum_{j_A=1}^{n_a} \sum_{j_B=1}^{n_b} A[i_A, j_A] \, B[i_B, j_B] \, a[j_A] \, b[j_B] \\
&\quad = \left(\sum_{j_A=1}^{n_a} A[i_A, j_A] \, a[j_A] \right) \left(\sum_{j_B=1}^{n_b} B[i_B, j_B] \, b[j_B] \right) \\
&\quad = (Aa)[i_A] \, (Bb)[i_B] \\
&\quad = \left((Aa) \otimes (Bb)\right)[i_B + m_B(i_A - 1)].
\end{aligned}
$$

Finite Differences

Thm: Eigenpairs of discrete two-dimensional Laplacian

Let $(\lambda_k, v_k)_{k=1}^n$ be the eigenpairs of $\Delta_n^{(1)}$.

Then, the eigenpairs of $\Delta_n^{(2)} = I_n \otimes \Delta_n^{(1)} + \Delta_n^{(1)} \otimes I_n$ are given by

$$\left(\quad \lambda_{k_1,k_2} = \lambda_{k_1} + \lambda_{k_2}, \qquad v_{k_1,k_2} = v_{k_1} \otimes v_{k_2} \quad \right)_{k_1,k_2=1}^n.$$

Proof.
$$\begin{aligned}
\Delta_n^{(2)} v_{k_1,k_2} &= \left(I_n \otimes \Delta_n^{(1)} + \Delta_n^{(1)} \otimes I_n \right) \left(v_{k_1} \otimes v_{k_2} \right) \\
&= v_{k_1} \otimes \left(\Delta_n^{(1)} v_{k_2} \right) + \left(\Delta_n^{(1)} v_{k_1} \right) \otimes v_{k_2} \\
&= \lambda_{k_2} v_{k_1} \otimes v_{k_2} + \lambda_{k_1} v_{k_1} \otimes v_{k_2} \\
&= \left(\lambda_{k_1} + \lambda_{k_2} \right) v_{k_1} \otimes v_{k_2} \\
&= \lambda_{k_1,k_2} v_{k_1,k_2}.
\end{aligned}$$

Finite Differences

Showing stability of $\Delta_n^{(2)}$ is straightforward now that we know the eigenvalues of $\Delta_n^{(2)}$.

Thm: Stability of $\Delta_n^{(2)}$

We have $\left\|(\Delta_n^{(2)})^{-1}\right\|_{2,n} = O(1)$ for $n \to \infty$.

Proof. We know from slide 46 that

$$\min_k |\lambda_k| = \lambda_1 = \pi^2 + O(n^{-2}),$$

and we have according to the result on the previous slide that

$$\min_{k_1,k_2} |\lambda_{k_1 k_2}| = 2\lambda_1 = 2\pi^2 + O(n^{-2}).$$

Therefore,

$$\left\|\left(\Delta_n^{(2)}\right)^{-1}\right\|_{2,n} = |\lambda_{\min}^{-1}| = (2\pi^2)^{-1} + O(n^{-2}) = O(1) \qquad \text{for } n \to \infty.$$

Finite Differences

Let us next show consistency of 2d finite differences.

Thm: Consistency of 2d finite differences

Assume $u : [0,1]^2 \to \mathbb{R}$ is a four times differentiable solution to the continuous Poisson equation $-\Delta u = f$ with homogeneous Dirichlet boundary conditions. Then,

$$\|\Delta_n^{(2)} u + f\|_{2,n} = O(n^{-2}) \qquad \text{for } n \to \infty.$$

Proof. We know from the one-dimensional consistency estimate from slide 55 that

$$\frac{\partial^2 u}{\partial x_1^2} = (I_n \otimes \Delta_n^{(1)})\, u + O(n^{-2}) \quad \text{and} \quad \frac{\partial^2 u}{\partial x_2^2} = (\Delta_n^{(1)} \otimes I_n)\, u + O(n^{-2});$$

hence we conclude that

$$
\begin{aligned}
\big\|\Delta_n^{(2)} u + f\big\|_{2,n} &= \big\|\Delta_n^{(2)} u - \Delta u\big\|_{2,n} \\
&= \big\|(I_n \otimes \Delta_n^{(1)})\, u - \tfrac{\partial^2 u}{\partial x_1^2} + (\Delta_n^{(1)} \otimes I_n)\, u - \tfrac{\partial^2 u}{\partial x_2^2}\big\|_{2,n} \\
&= \big\|O(n^{-2})\big\|_{2,n} \\
&= O(n^{-2}).
\end{aligned}
$$

Finite Differences

Proof of $\|u - u_n\|_{2,n} = O(n^{-2})$ (slide 78).

Inserting the above stability and consistency estimates into the stability and consistency lemma, we obtain

$$\|u - u_n\|_{n,2} \leq \|\Delta_n^{-1}\|_{2,n} \|\Delta_n u + f\|_{2,n} = O(1) \, O(n^{-2}) = O(n^{-2})$$

as claimed.

Numerical demonstration

See `convergence_2d()`.

Finite Differences

Finite differences in three dimensions

Everything I have said about finite differences in two dimensions generalises straightforwardly to finite differences in three dimensions:

- Functions $u : [0,1]^3 \to \mathbb{R}$ are represented as "cubes" of point values

$$u \in \mathbb{R}^{n \times n \times n}.$$

- The Laplacian matrix becomes

$$\Delta_n^{(3)} = I_n \otimes I_n \otimes \Delta_n^{(1)} + I_n \otimes \Delta_n^{(1)} \otimes I_n + \Delta_n^{(1)} \otimes I_n \otimes I_n.$$

- The finite difference solutions $u_n = -\left(\Delta_n^{(3)}\right)^{-1} f$ satisfy

$$\|u_n - u\|_{2,n} = O(n^{-2}) \qquad \text{for } n \to \infty,$$

where now $\|u\|_{2,n} = (n+1)^{-3/2} \|u\|_2$. This can be shown using the stability and consistency lemma just like in 2d.

See `solve_poisson_3d()` and `convergence_3d()` for numerical demonstration.

Finite Differences

Summary of finite differences

The above concludes our discussion of the core theory of finite differences. To summarize, the key take-away points are:

▶ Finite differences transforms the partial differential equation $-\Delta u = f$ into a linear system of equations $-\Delta_n^{(d)} u_n = f$.

▶ The difference between the finite-difference solutions u_n and the exact solution u satisfies $\|u - u_n\|_{2,n} = O(n^{-2})$.

This summary answers two of the three standard questions about a numerical algorithm, namely it tells us:

▶ What is the algorithm?

▶ What is the speed of convergence of this algorithm?

However, notably missing from this summary is a statement regarding the runtime required to solve $-\Delta_n^{(d)} u_n = f$. The reason for this is to be found on slide 37, where I pointed out that $-\Delta_n^{(1)} u_n = f$ could be solved in $O(n^3)$ runtime using the standard LU factorisation but that doing so is likely not optimal due to the many zeros in $\Delta_n^{(1)}$.

Finite Differences

Summary of finite differences (continued)
Let us therefore conclude this lecture by first quantifying exactly how many entries in $\Delta_n^{(d)}$ are nonzero and then investigating how we work with such matrices in Julia. Both of these topics will serve as preparation for what is to come in the next two lectures.

Number of nonzero entries in $\Delta_n^{(1)}$
We have seen that the one-dimensional Laplacian matrix is given by

$$\Delta_n = (n+1)^2 \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -2 \end{pmatrix};$$

hence this matrix contains exactly $\boxed{3n-2}$ nonzero entries.

Finite Differences

Number of nonzero entries in $\Delta_n^{(2)}$

We can further verify that

$$\Delta_4^{(2)} = I_4 \otimes \Delta_4^{(1)} + \Delta_4^{(1)} \otimes I_4$$

is given by

$$
\Delta_4^{(2)} = (4+1)^2
\begin{pmatrix}
-4 & 1 & & & 1 & & & & & & & & & & & \\
1 & -4 & 1 & & & 1 & & & & & & & & & & \\
& 1 & -4 & 1 & & & 1 & & & & & & & & & \\
& & 1 & -4 & & & & 1 & & & & & & & & \\
1 & & & & -4 & 1 & & & 1 & & & & & & & \\
& 1 & & & 1 & -4 & 1 & & & 1 & & & & & & \\
& & 1 & & & 1 & -4 & 1 & & & 1 & & & & & \\
& & & 1 & & & 1 & -4 & & & & 1 & & & & \\
& & & & 1 & & & & -4 & 1 & & & 1 & & & \\
& & & & & 1 & & & 1 & -4 & 1 & & & 1 & & \\
& & & & & & 1 & & & 1 & -4 & 1 & & & 1 & \\
& & & & & & & 1 & & & 1 & -4 & & & & 1 \\
& & & & & & & & 1 & & & & -4 & 1 & & \\
& & & & & & & & & 1 & & & 1 & -4 & 1 & \\
& & & & & & & & & & 1 & & & 1 & -4 & 1 \\
& & & & & & & & & & & 1 & & & 1 & -4
\end{pmatrix}
$$

and that $\Delta_n^{(2)}$ contains $\boxed{5n^2 + O(n)}$ nonzero entries.

I will refer to this shape as "tridiagonal plus two far off-diagonals".

Finite Differences

Number of nonzero entries in $\Delta_n^{(3)}$

Finally, we observe that

$$\Delta_3^{(3)} = I_3 \otimes I_3 \otimes \Delta_3^{(1)} + I_3 \otimes \Delta_3^{(1)} \otimes I_3 + \Delta_3^{(1)} \otimes I_3 \otimes I_3$$

is given by

$$\Delta_3^{(3)} = (3+1)^2 \begin{pmatrix} \text{matrix} \end{pmatrix},$$

and that $\Delta_n^{(3)}$ contains $\boxed{7n^3 + O(n^2)}$ nonzero entries.

I will refer to this shape as "tridiagonal plus two far off-diagonals plus two far far off-diagonals".

Finite Differences

Numerical analysts use a special term to refer to matrices like the above.

Terminology: Dense and sparse matrices

▶ A matrix with only few nonzero entries is called *sparse*.

▶ A matrix where most or all entries are nonzero is called *dense*.

Note that whether a matrix has "many" or "few" nonzero entries is subjective and depends on the context; hence the terms "dense" and "sparse" are themselves subjective and context-dependent.

I will further use the following abbreviation.

Notation: Number of nonzero entries

I will write $\mathrm{nnz}(A)$ to denote the number of nonzero entries in A.

Finite Differences

The importance of sparsity

Recognising and characterising the sparsity of $\Delta_n^{(d)}$ is important because finite differences would be not nearly as powerful if the discrete Laplacians were not sparse.

I will elaborate further in a moment, but before I can do so we first need to look into how to work with sparse matrices in Julia.

Finite Differences

Sparse matrices in Julia

Julia allows us to represent sparse matrices using only $O(n + \text{nnz}(A))$ rather than $O(n^2)$ memory by means of the SparseMatrixCSC type.

Matrices of this type can be created using the sparse() function:

```julia
julia> i = [1,1,2]
       j = [1,2,2]
       v = [3,4,5]
       A = sparse(i,j,v)

2×2 SparseMatrixCSC{...} with 3 stored entries:
  [1, 1]  =  3
  [1, 2]  =  4
  [2, 2]  =  5
```

Finite Differences

Sparse matrices in Julia (continued)

The above output is indicative of how `SparseMatrixCSC` matrices are stored, but it can be a bit hard to read. We can obtain a more readable output by converting to a dense matrix type as follows:

```julia
julia> Matrix(A)
2×2 Array{...}:
 3  4
 0  5
```

Finite Differences

Sparse matrices in Julia (continued)

Other functions for creating sparse matrices include:

▶ sparse(I,n,n) creates the $n \times n$ identity matrix.

```
julia> sparse(I,2,2)
2×2 SparseMatrixCSC{...} with 2 stored entries:
  [1, 1]  =  1
  [2, 2]  =  1
```

▶ spdiagm(d=>v, ...) creates a sparse matrix with vector v on the
dth diagonal.

```
julia> Matrix(spdiagm(-1 => [1,2], 0 => [3,4,5]))
3×3 Array{...}:
 3  0  0
 1  4  0
 0  2  5
```

Finite Differences

Sparse matrices in Julia (continued)

▶ sparse(A) converts any matrix A to a sparse representation:

```julia
julia> sparse([
           4 0 0
           0 0 6
           0 5 0
       ])
3×3 SparseMatrixCSC{...} with 3 stored entries:
  [1, 1]  =  4
  [3, 2]  =  5
  [2, 3]  =  6
```

▶ sprand(m,n,p) creates an $m \times n$ random matrix where each entry is nonzero with probability p:

```julia
julia> Matrix(sprand(3,3,0.5))
3×3 ArrayFloat64,2:
 0.0       0.958872  0.0
 0.846538  0.0       0.366945
 0.715418  0.41126   0.286191
```

Finite Differences

The importance of exploiting sparsity

Now that we know how to work with sparse matrices in Julia, we can start looking into why doing so is of fundamental importance for making the finite difference method practically feasible.

Let us begin by observing that we can solve $-\Delta_n^{(1)} u_n = f$ very quickly even for a very large $n = 10^6$ as long as we represent $\Delta_n^{(1)}$ as a sparse matrix:

```julia
julia> typeof(laplacian_1d(2))
SparseMatrixCSC{...}
```

```julia
julia> n = Int(1e6)
       @time laplacian_1d(n) \ zeros(n);
  0.053375 seconds (...)
```

However, if we convert $\Delta_n^{(1)}$ to a dense matrix then we immediately get an error:

```julia
julia> Matrix(laplacian_1d(n))
ERROR: OutOfMemoryError()
```

Finite Differences

The importance of exploiting sparsity

The reason for this is easily understood: storing $\Delta_{10^6}^{(1)}$ as a dense matrix requires a memory capacity of

$$\left(10^6\right)^2 \texttt{Float64} \times 8 \, \tfrac{\text{bytes}}{\texttt{Float64}} \quad = \quad 8 \, \text{terabytes},$$

which is much more than the 15 gigabytes available on my computer.

On the other hand, if we store $\Delta_{10^6}^{(1)}$ in the $(i, j, A[i,j])$ format, then we require at most

$$3 \times 3 \times 10^6 \, [\texttt{Float64 or Int64}] \times 8 \, \tfrac{\text{bytes}}{[\texttt{Float64 or Int64}]} \quad = \quad 72 \, \text{megabytes},$$

which is small enough to easily fit on any reasonable computing platform.

In the above formula,

▶ The first 3 denotes the number of variables $i, j, A[i,j]$ per nonzero entry,

▶ The second 3 denotes the maximal number of nonzeros per column, and

▶ 10^6 denotes the number of columns.

Finite Differences

The importance of exploiting sparsity (continued)

The above shows that exploiting sparsity is important to ensure that the finite difference method stays within the memory limits of our hardware.

In addition, sparsity is also important for ensuring that finite differences runs within a reasonable amount of time: if we reduce n to $n = 10^4$ so we can solve $-\Delta_n^{(1)} u_n = f$ in both the sparse and dense representation, then we observe that the sparse version is about 300x faster than the dense one.

```julia
julia> n = Int(1e4)
       @time laplacian_1d(n) \ zeros(n);
  0.003764 seconds (...)

julia> @time Matrix(laplacian_1d(n)) \ zeros(n);
  9.082081 seconds (...)
```

We therefore conclude:

> Solving $-\Delta_n^{(d)} u_n = f$ would not be feasible both for runtime and memory reasons if $\Delta_n^{(d)}$ was not sparse.

Finite Differences

From finite differences to fast linear solvers

Unfortunately, recognising the importance of sparsity is only the first step towards making the finite difference method practical; in addition, we also need algorithms which can exploit this sparsity for computational benefits.

In particular, we need linear system solvers which can reduce the runtime of solving $-\Delta_n^{(d)} u_n = f$ the $O(n^3)$ runtime required by the standard LU factorisation. Deriving such algorithms will be the topic of the next two lectures.

▶ In Lecture 4, we will study how sparsity can be exploited in the LU factorisation to reduce the runtime below the $O(n^3)$ runtime of the standard algorithm.

▶ In Lecture 5, we will see that instead of computing an LU factorisation, we can approximately solve $Ax = b$ by setting $x \approx p(A) b$ for some polynomial $p(x) \approx \frac{1}{x}$.
Doing so helps with exploiting sparsity because $p(A) b$ can be evaluated using only matrix-vector products, and the runtime of one such product is proportional to the number of nonzero entries in A.

Finite Differences

Summary

▶ Poisson eq. = mass conservation + Fick's law + steady state.

▶ Finite elements $\leftrightarrow$ approximation with piecewise polynomials.
  Finite differences $\leftrightarrow$ approximation with vectors of point values.

▶ Derivation of $\Delta_n^{(1)}$ starting from

$$u'\left(\tfrac{i+1/2}{n+1}\right) \approx \frac{u\left(\tfrac{i+1}{n+1}\right) - u\left(\tfrac{i}{n+1}\right)}{1/(n+1)}.$$

▶ Stability and consistency lemma:

$$\|u - u_n\| \le \|\Delta_n^{-1}\|\,\|\Delta_n u + f\|.$$

▶ Fourier analysis and energy method for showing stability.

▶ Taylor series for showing consistency.

▶ Vectorisation and Kronecker product for finite differences in $d > 1$ dimensions.

▶ Sparsity of $\Delta_n^{(d)}$ makes finite differences work in practice.