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

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

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.

`https://www.shutterstock.com/video/clip-9873545-many-ants-on-sand`

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

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

$n = n(x)$ 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. $\int_{\Omega'} g(x)\,dx = 0 \quad$ 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) $\Longleftarrow$ (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) $\Longrightarrow$ (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.

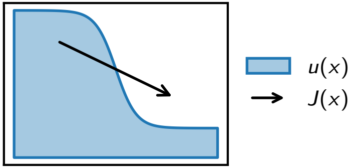
- We must assume that we are in a steady state, i.e. $\frac{\partial u}{\partial t} = 0$.
- We must relate the ant flow $J(x)$ to the ant concentration $u(x)$.

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

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

  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 = 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` : `Float64` $\to$ `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

$$\big\{\mathtt{u} : \mathtt{Float64}^d \to \mathtt{Float64}\big\};$$

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.

`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)

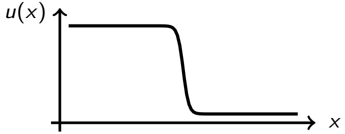
Possible reasons for this include:

- 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.

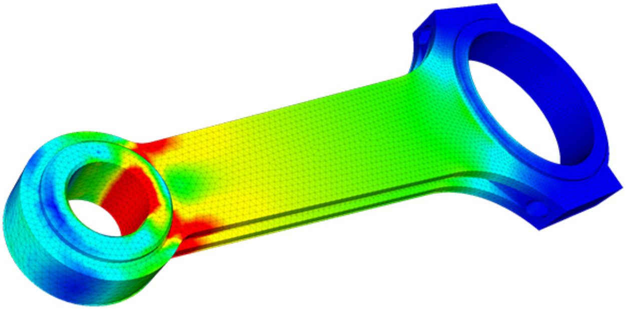
  This indicates that polynomials are often not the most efficient tool for solving partial differential equations.

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.



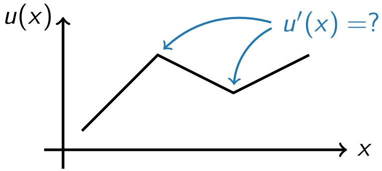
`https://www.simscale.com/blog/2016/10/what-is-finite-element-method/`

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

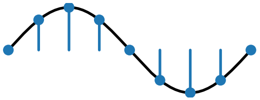
Representing functions (continued)

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

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



.

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 $A_{i+i;(j-1)/2}$.

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

Example. Instead of $A_{i+i;(j-1)/2}$, 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

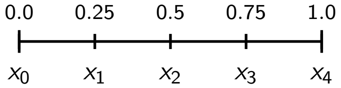
Grids

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

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

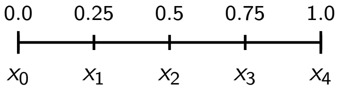


in this lecture. This choice is based on several assumptions presented on the next slide.

Finite Differences

Grids (continued)

Assumptions underlying the grid

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

- 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.)

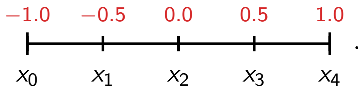
Finite Differences

Grids (continued)

A few variations may help to further clarify the above.

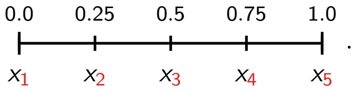
- If the domain of the PDE was $\Omega = [-1, 1]$ instead of $\Omega = [0, 1]$, then we would choose the grid

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



- 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

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



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 $-\Delta \mathtt{u} = \mathtt{f}$, 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

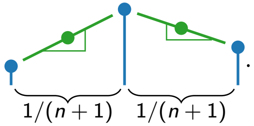
Numerically solving Poisson's equation (continued)

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 $u\big(\frac{i}{n+1}\big)$ as desired. We can make this point even clearer by replacing the point values in

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

with their corresponding vector entries, which yields

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

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

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

These findings can 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.

I demonstrate how to do this in `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 consists of two separate statements regarding the scaling of the runtime and error as functions of the grid size n.

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

Finite Differences

[To be continued]