AM 205: lecture 16

- Last time: hyperbolic PDEs
- Today: parabolic and elliptic PDEs, introduction to optimization
The $\theta$-Method: Accuracy

The truncation error analysis is fairly involved, hence we just give the result:

$$T_j^n \equiv \frac{u_{j+1}^n - u_j^n}{\Delta t} - \theta \frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2} - (1 - \theta) \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2}$$

$$= [u_t - u_{xx}] + \left[ \left( \frac{1}{2} - \theta \right) \Delta t u_{xxt} - \frac{1}{12} (\Delta x)^2 u_{xxxx} \right]$$

$$+ \left[ \frac{1}{24} (\Delta t)^2 u_{ttt} - \frac{1}{8} (\Delta t)^2 u_{xxtt} \right]$$

$$+ \left[ \frac{1}{12} \left( \frac{1}{2} - \theta \right) \Delta t(\Delta x)^2 u_{xxxxt} - \frac{2}{6!} (\Delta x)^4 u_{xxxxxx} \right] + \cdots$$

The term $u_t - u_{xx}$ in $T_j^n$ vanishes since $u$ solves the PDE
The \( \theta \)-Method: Accuracy

**Key point:** This is a first order method, unless \( \theta = 1/2 \), in which case we get a second order method!

\( \theta \)-method gives us consistency (at least first order) and stability (assuming \( \Delta t \) is chosen appropriately when \( \theta \in [0, 1/2) \))

Hence, from Lax Equivalence Theorem, the method is convergent.
The Heat Equation

Note that the heat equation models a **diffusive process**, hence it tends to smooth out discontinuities

**Python demo:** Heat equation with discontinuous initial condition

This is very different to hyperbolic equations, e.g. the advection equation will just transport a discontinuity in $u_0$
Elliptic PDEs
The canonical elliptic PDE is the Poisson equation

In one-dimension, for \( x \in [a, b] \), this is \(-u''(x) = f(x)\) with boundary conditions at \( x = a \) and \( x = b \)

We have seen this problem already: **Two-point boundary value problem**!

(Recall that Elliptic PDEs model steady-state behavior, there is no time-derivative)
In order to make this into a PDE, we need to consider more than one spatial dimension.

Let $\Omega \subset \mathbb{R}^2$ denote our domain, then the Poisson equation for $(x, y) \in \Omega$ is

$$u_{xx} + u_{yy} = f(x, y)$$

This is generally written more succinctly as $\Delta u = f$.

We again need to impose boundary conditions (Dirichlet, Neumann, or Robin) on $\partial \Omega$ (recall $\partial \Omega$ denotes boundary of $\Omega$).
Elliptic PDEs

We will consider how to use a finite difference scheme to approximate this 2D Poisson equation

First, we introduce a uniform grid to discretize $\Omega$
Elliptic PDEs

Let \( h = \Delta x = \Delta y \) denote the grid spacing.

Then,

\[ x_i = ih, \quad i = 0, 1, 2 \ldots, n_x - 1, \]
\[ y_j = jh, \quad j = 0, 1, 2, \ldots, n_y - 1, \]
\[ U_{i,j} \approx u(x_i, y_j) \]

Then, we need to be able to approximate \( u_{xx} \) and \( u_{yy} \) on this grid.

Natural idea: Use central difference approximation!
Elliptic PDEs

We have

\[ u_{xx}(x_i, y_j) = \frac{u(x_{i-1}, y_j) - 2u(x_i, y_j) + u(x_{i+1}, y_j)}{h^2} + O(h^2), \]

and

\[ u_{yy}(x_i, y_j) = \frac{u(x_i, y_{j-1}) - 2u(x_i, y_j) + u(x_i, y_{j+1})}{h^2} + O(h^2), \]

so that

\[ u_{xx}(x_i, y_j) + u_{yy}(x_i, y_j) = \frac{u(x_i, y_{j-1}) + u(x_{i-1}, y_j) - 4u(x_i, y_j) + u(x_{i+1}, y_j) + u(x_i, y_{j+1})}{h^2} + O(h^2) \]
Elliptic PDEs

Hence we define our approximation to the Laplacian as

\[
\frac{U_{i, j-1} + U_{i-1, j} - 4U_{i, j} + U_{i+1, j} + U_{i, j+1}}{h^2}
\]

This corresponds to a “5-point stencil”
As usual, we represent the numerical solution as a vector $U \in \mathbb{R}^{n_x n_y}$.

We want to construct a differentiation matrix $D_2 \in \mathbb{R}^{n_x n_y \times n_x n_y}$ that approximates the Laplacian.

**Question**: How many non-zero diagonals will $D_2$ have?

To construct $D_2$, we need to be able to relate the entries of the vector $U$ to the “2D grid-based values” $U_{i,j}$.
Elliptic PDEs

Hence we need to number the nodes from 1 to \( n_x n_y \) — we number nodes along the “bottom row” first, then second bottom row, etc.

Let \( G \) denote the mapping from the 2D indexing to the 1D indexing. From the above figure we have:

\[
G(i, j; n_x) = jn_x + i, \quad \text{and hence} \quad U_{G(i,j;n_x)} = U_{i,j}
\]
Elliptic PDEs

Let us focus on node $(i,j)$ in our F.D. grid, this corresponds to entry $G(i,j; n_x)$ of $U$

Due to the 5-point stencil, row $G(i,j; n_x)$ of $D_2$ will only have non-zeros in columns

\[
\begin{align*}
G(i,j-1; n_x) & = G(i,j; n_x) - n_x, \\
G(i-1,j; n_x) & = G(i,j; n_x) - 1, \\
G(i,j; n_x) & = G(i,j; n_x), \\
G(i+1,j; n_x) & = G(i,j; n_x) + 1, \\
G(i,j+1; n_x) & = G(i,j; n_x) + n_x
\end{align*}
\]

- (2), (3), (4), give the same tridiagonal structure that we’re used to from differentiation matrices in 1D domains
- (1), (5) give diagonals shifted by $\pm n_x$
Elliptic PDEs

For example, sparsity pattern of $D_2$ when $n_x = n_y = 6$
Elliptic PDEs

Python demo: Solve the Poisson equation

\[ \Delta u = -\exp \left\{ -(x - 0.25)^2 - (y - 0.5)^2 \right\}, \]

for \((x, y) \in \Omega = [0, 1]^2\) with \(u = 0\) on \(\partial \Omega\)
Nonlinear Equations and Optimization
Motivation: Nonlinear Equations

So far we have mostly focused on linear phenomena

- Interpolation leads to a linear system $Vb = y$ (monomials) or $Ib = y$ (Lagrange polynomials)

- Linear least-squares leads to the normal equations $A^T Ab = A^T y$

- We saw examples of linear physical models (Ohm’s Law, Hooke’s Law, Leontief equations) $\implies Ax = b$

- F.D. discretization of a linear PDE leads to a linear algebraic system $AU = F$
Motivation: Nonlinear Equations

Of course, nonlinear models also arise all the time

- Nonlinear least-squares, Gauss–Newton/Levenberg–Marquardt

- Countless nonlinear physical models in nature, e.g. non-Hookean material models

- F.D. discretization of a non-linear PDE leads to a nonlinear algebraic system

\(^1\)Important in modeling large deformations of solids
Motivation: Nonlinear Equations

Another example is computation of Gauss quadrature points/weights

We know this is possible via roots of Legendre polynomials

But we could also try to solve the nonlinear system of equations for \( \{(x_1, w_1), (x_2, w_2), \ldots, (x_n, w_n)\} \)
Motivation: Nonlinear Equations

e.g. for $n = 2$, we need to find points/weights such that all polynomials of degree 3 are integrated exactly, hence

$$w_1 + w_2 = \int_{-1}^{1} 1dx = 2$$

$$w_1 x_1 + w_2 x_2 = \int_{-1}^{1} xdx = 0$$

$$w_1 x_1^2 + w_2 x_2^2 = \int_{-1}^{1} x^2dx = \frac{2}{3}$$

$$w_1 x_1^3 + w_2 x_2^3 = \int_{-1}^{1} x^3dx = 0$$
Motivation: Nonlinear Equations

We usually write a nonlinear system of equations as

\[ F(x) = 0, \]

where \( F : \mathbb{R}^n \rightarrow \mathbb{R}^m \)

We implicitly absorb the “right-hand side” into \( F \) and seek a root of \( F \)

In this Unit we focus on the case \( m = n \), \( m > n \) gives nonlinear least-squares
Motivation: Nonlinear Equations

We are very familiar with scalar \((m = 1)\) nonlinear equations

Simplest case is a quadratic equation

\[ ax^2 + bx + c = 0 \]

We can write down a closed form solution, the quadratic formula

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]
Motivation: Nonlinear Equations

In fact, there are also closed-form solutions for arbitrary cubic and quartic polynomials, due to Ferrari and Cardano (∼ 1540)

Important mathematical result is that there is no general formula for solving fifth or higher order polynomial equations

Hence, even for the simplest possible case (polynomials), the only hope is to employ an iterative algorithm

An iterative method should converge in the limit $n \to \infty$, and ideally yields an accurate approximation after few iterations
Motivation: Nonlinear Equations

There are many well-known iterative methods for nonlinear equations.

Probably the simplest is the bisection method for a scalar equation $f(x) = 0$, where $f \in C[a, b]$.

Look for a root in the interval $[a, b]$ by bisecting based on sign of $f$. 
# Nonlinear Equations

```python
#!/usr/bin/python
from math import *

# Function to consider
def f(x):
    return x*x-4*sin(x)

# Initial interval: assume f(a)<0 and f(b)>0
a=1
b=3

# Bisection search
while b-a>1e-8:
    print a,b
    c=0.5*(b+a)
    if f(c)<0: a=c
    else: b=c

print "# Root at",0.5*(a+b)
```
Root in the interval [1.933716, 1.933777]
Motivation: Nonlinear Equations

Bisection is a robust root-finding method in 1D, but it does not generalize easily to $\mathbb{R}^n$ for $n > 1$

Also, bisection is a crude method in the sense that it makes no use of magnitude of $f$, only $\text{sign}(f)$

We will look at mathematical basis of alternative methods which generalize to $\mathbb{R}^n$:
- Fixed-point iteration
- Newton’s method
Optimization
Motivation: Optimization

Another major topic in Scientific Computing is optimization

Very important in science, engineering, industry, finance, economics, logistics,…

Many engineering challenges can be formulated as optimization problems, e.g.:

- Design car body that maximizes downforce\(^2\)
- Design a bridge with minimum weight

\(^2\)A major goal in racing car design
Motivation: Optimization

Of course, in practice, it is more realistic to consider optimization problems with constraints, e.g.:

- Design car body that maximizes downforce, subject to a constraint on drag
- Design a bridge with minimum weight, subject to a constraint on strength
Motivation: Optimization

Also, (constrained and unconstrained) optimization problems arise naturally in science

Physics:

- many physical systems will naturally occupy a minimum energy state
- if we can describe the energy of the system mathematically, then we can find minimum energy state via optimization
Motivation: Optimization

Biology:

- recent efforts in Scientific Computing have sought to understand biological phenomena quantitively via optimization
- computational optimization of, e.g. fish swimming or insect flight, can reproduce behavior observed in nature
- this jells with the idea that evolution has been “optimizing” organisms for millions of years
Motivation: Optimization

All these problems can be formulated as: Optimize (max. or min.) an objective function over a set of feasible choices, i.e.

Given an objective function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) and a set \( S \subset \mathbb{R}^n \), we seek \( x^* \in S \) such that \( f(x^*) \leq f(x), \forall x \in S \)

(It suffices to consider only minimization, maximization is equivalent to minimizing \(-f\))

\( S \) is the feasible set, usually defined by a set of equations and/or inequalities, which are the constraints

If \( S = \mathbb{R}^n \), then the problem is unconstrained
Motivation: Optimization

The standard way to write an optimization problem is

$$\min_{x \in S} f(x) \text{ subject to } g(x) = 0 \text{ and } h(x) \leq 0,$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, $h : \mathbb{R}^n \to \mathbb{R}^p$. 

Motivation: Optimization

For example, let $x_1$ and $x_2$ denote radius and height of a cylinder, respectively

Minimize the surface area of a cylinder subject to a constraint on its volume\(^3\) (we will return to this example later)

$$
\min_{x} f(x_1, x_2) = 2\pi x_1 (x_1 + x_2)
$$

subject to $g(x_1, x_2) = \pi x_1^2 x_2 - V = 0$

\(^3\)Heath Example 6.2
Motivation: Optimization

If \( f, g \) and \( h \) are all affine, then the optimization problem is called a linear program.

(Here the term “program” has nothing to do with computer programming; instead it refers to logistics/planning)

Affine if \( f(x) = Ax + b \) for a matrix \( A \), i.e. linear plus a constant\(^4\)

Linear programming may already be familiar

Just need to check \( f(x) \) on vertices of the feasible region

---

\(^4\)Recall that “affine” is not the same as “linear”, i.e. 
\( f(x + y) = Ax + Ay + b \) and \( f(x) + f(y) = Ax + Ay + 2b \)
Motivation: Optimization

If the objective function or any of the constraints are nonlinear then we have a **nonlinear optimization** problem or **nonlinear program**.

We will consider several different approaches to nonlinear optimization in this Unit.

Optimization routines typically use **local information** about a function to iteratively approach a **local minimum**.
Motivation: Optimization

In some cases this easily gives a global minimum
Motivation: Optimization

But in general, global optimization can be very difficult

We can get “stuck” in local minima!
Motivation: Optimization

And can get much harder in higher spatial dimensions
Motivation: Optimization

There are robust methods for finding local minimima, and this is what we focus on in AM205.

Global optimization is very important in practice, but in general there is no way to guarantee that we will find a global minimum.

Global optimization basically relies on heuristics:

▶ try several different starting guesses ("multistart" methods)
▶ simulated annealing
▶ genetic methods\(^5\)

\(^5\)Simulated annealing and genetic methods are covered in AM207
Root Finding: Scalar Case
Fixed-Point Iteration

Suppose we define an iteration

\[ x_{k+1} = g(x_k) \]  \quad (*)

e.g. recall Heron’s Method from Assignment 0 for finding \( \sqrt{a} \):

\[ x_{k+1} = \frac{1}{2} \left( x_k + \frac{a}{x_k} \right) \]

This uses \( g_{\text{heron}}(x) = \frac{1}{2} (x + a/x) \)
Suppose $\alpha$ is such that $g(\alpha) = \alpha$, then we call $\alpha$ a fixed point of $g$.

For example, we see that $\sqrt{a}$ is a fixed point of $g_{\text{heron}}$ since

$$g_{\text{heron}}(\sqrt{a}) = \frac{1}{2} (\sqrt{a} + a/\sqrt{a}) = \sqrt{a}$$

A fixed-point iteration terminates once a fixed point is reached, since if $g(x_k) = x_k$ then we get $x_{k+1} = x_k$.

Also, if $x_{k+1} = g(x_k)$ converges as $k \to \infty$, it must converge to a fixed point: Let $\alpha \equiv \lim_{k \to \infty} x_k$, then

$$\alpha = \lim_{k \to \infty} x_{k+1} = \lim_{k \to \infty} g(x_k) = g \left( \lim_{k \to \infty} x_k \right) = g(\alpha)$$

\[6\] Third equality requires $g$ to be continuous.
Fixed-Point Iteration

Hence, for example, we know if Heron’s method converges, it will converge to $\sqrt{a}$

It would be very helpful to know when we can guarantee that a fixed-point iteration will converge

Recall that $g$ satisfies a Lipschitz condition in an interval $[a, b]$ if $\exists L \in \mathbb{R}_{>0}$ such that

$$|g(x) - g(y)| \leq L|x - y|, \quad \forall x, y \in [a, b]$$

$g$ is called a contraction if $L < 1$
**Theorem:** Suppose that \( g(\alpha) = \alpha \) and that \( g \) is a contraction on \([\alpha - A, \alpha + A]\). Suppose also that \(|x_0 - \alpha| \leq A\). Then the fixed point iteration converges to \( \alpha \).

**Proof:**

\[
|x_k - \alpha| = |g(x_{k-1}) - g(\alpha)| \leq L|x_{k-1} - \alpha|,
\]

which implies

\[
|x_k - \alpha| \leq L^k|x_0 - \alpha|
\]

and, since \( L < 1 \), \( |x_k - \alpha| \to 0 \) as \( k \to \infty \). (Note that \(|x_0 - \alpha| \leq A\) implies that all iterates are in \([\alpha - A, \alpha + A]\).) \( \square \)

(This proof also shows that error decreases by factor of \( L \) each iteration)
Recall that if \( g \in C^1[a, b] \), we can obtain a Lipschitz constant based on \( g' \): 

\[
L = \max_{\theta \in (a, b)} |g'(\theta)|
\]

We now use this results to show that if \( |g'(\alpha)| < 1 \), then there is a neighborhood of \( \alpha \) on which \( g \) is a contraction.

This tells us that we can verify convergence of a fixed point iteration by checking the gradient of \( g \).
By continuity of $g'$ (and hence continuity of $|g'|$), for any $\epsilon > 0$ \( \exists \delta > 0 \) such that for \( x \in (\alpha - \delta, \alpha + \delta) \):

\[
|g'(x)| - |g'(\alpha)| \leq \epsilon \implies \max_{x \in (\alpha - \delta, \alpha + \delta)} |g'(x)| \leq |g'(\alpha)| + \epsilon
\]

Suppose $|g'(\alpha)| < 1$ and set $\epsilon = \frac{1}{2}(1 - |g'(\alpha)|)$, then there is a neighborhood on which $g$ is Lipschitz with $L = \frac{1}{2}(1 + |g'(\alpha)|)$.

Then $L < 1$ and hence $g$ is a contraction in a neighborhood of $\alpha$. 

Furthermore, as $k \to \infty$, 

\[
\frac{|x_{k+1} - \alpha|}{|x_k - \alpha|} = \frac{|g(x_k) - g(\alpha)|}{|x_k - \alpha|} \to |g'(\alpha)|,
\]

Hence, asymptotically, error decreases by a factor of $|g'(\alpha)|$ each iteration.