AM 205: lecture 16

- Last time: hyperbolic PDEs
- Today: parabolic and elliptic PDEs, introduction to optimization

The θ-Method: Accuracy

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

$$
\begin{aligned}
T_j^n \equiv\; & \frac{u_j^{n+1}-u_j^n}{\Delta t} - \theta \frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2} - (1-\theta)\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} \\
=\; & [u_t - u_{xx}] + \left[\left(\frac{1}{2}-\theta\right)\Delta t u_{xxt} - \frac{1}{12}(\Delta x)^2 u_{xxxx}\right] \\
& + \left[\frac{1}{24}(\Delta t)^2 u_{ttt} - \frac{1}{8}(\Delta t)^2 u_{xxtt}\right] \\
& + \left[\frac{1}{12}\left(\frac{1}{2}-\theta\right)\Delta t(\Delta x)^2 u_{xxxxt} - \frac{2}{6!}(\Delta x)^4 u_{xxxxxx}\right] + \cdots
\end{aligned}
$$

The term $u_t - u_{xx}$ in T_j^n vanishes since u solves the PDE

The θ-Method: Accuracy

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

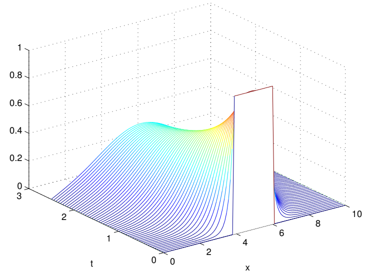
θ-method gives us consistency (at least first order) and stability (assuming Δt is chosen appropriately when $\theta \in [0, 1/2)$)

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 discontinous initial condition



This is very different to hyperbolic equations, *e.g.* the advection equation will just transport a discontinuity in u_0

Elliptic PDEs

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)

Elliptic PDEs

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

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

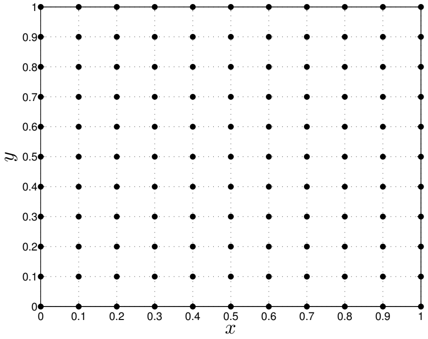
This is generally written more succinctly as $\Delta u = f$

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

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 Ω



Elliptic PDEs

Let $h = \Delta x = \Delta y$ denote the grid spacing

Then,

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

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

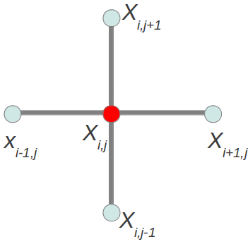
$$u_{xx}(x_i, y_j) + u_{yy}(x_i, y_j) =$$
$$\frac{u(x_i, y_{j-1}) + u(x_{i-1}, y_j) - 4u(x_i, y_j) + u(x_{i+1}, y_j) + u(x_i, y_{j+1})}{h^2} + O(h^2)$$

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”



Elliptic PDEs

As usual, we represent the numerical solution as a vector $\mathbb{U} \in \mathbb{R}^{n_x n_y}$

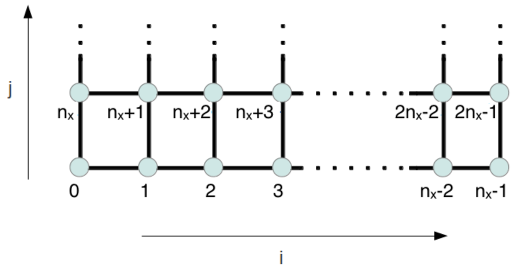
We want to construct a differentiation matrix $D_2 \in \mathbb{R}^{n_x n_y \times n_x n_y}$ that approximates the Laplacian

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 $\mathbb{U}$ 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 $\mathcal{G}$ denote the mapping from the 2D indexing to the 1D indexing. From the above figure we have:

$$\mathcal{G}(i,j;n_x) = jn_x + i, \quad \text{and hence} \quad \mathbb{U}_{\mathcal{G}(i,j;n_x)} = U_{i,j}$$

Elliptic PDEs

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

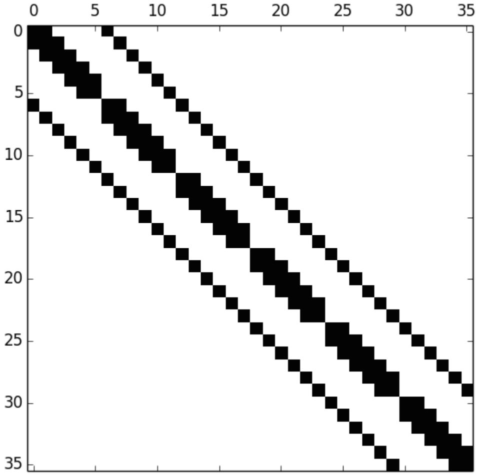
Due to the 5-point stencil, row $\mathcal{G}(i, j; n_x)$ of D_2 will only have non-zeros in columns

$$
\begin{aligned}
\mathcal{G}(i, j-1; n_x) &= \mathcal{G}(i, j; n_x) - n_x, && (1)\\
\mathcal{G}(i-1, j; n_x) &= \mathcal{G}(i, j; n_x) - 1, && (2)\\
\mathcal{G}(i, j; n_x) &= \mathcal{G}(i, j; n_x), && (3)\\
\mathcal{G}(i+1, j; n_x) &= \mathcal{G}(i, j; n_x) + 1, && (4)\\
\mathcal{G}(i, j+1; n_x) &= \mathcal{G}(i, j; n_x) + n_x && (5)
\end{aligned}
$$

- (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 $n_x = n_y = 6$

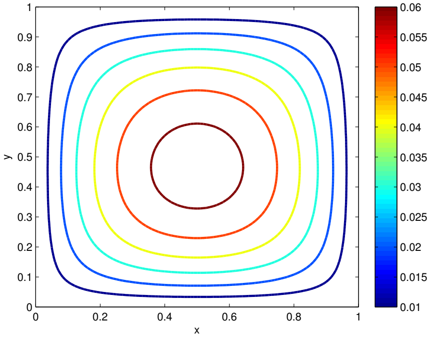


Elliptic PDEs

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

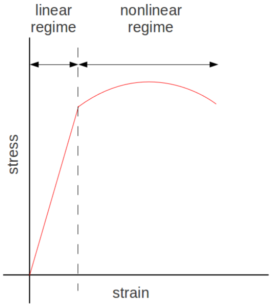
So far we have mostly focused on linear phenomena

- Interpolation leads to a linear system $Vb = y$ (monomials) or $\mathrm{I}b = y$ (Lagrange polynomials)
- Linear least-squares leads to the normal equations $A^T A b = A^T y$
- 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[1]



- 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

$$\begin{aligned} w_1 + w_2 &= \int_{-1}^{1} 1 \mathrm{d}x = 2 \\ w_1 x_1 + w_2 x_2 &= \int_{-1}^{1} x \mathrm{d}x = 0 \\ w_1 x_1^2 + w_2 x_2^2 &= \int_{-1}^{1} x^2 \mathrm{d}x = 2/3 \\ w_1 x_1^3 + w_2 x_2^3 &= \int_{-1}^{1} x^3 \mathrm{d}x = 0 \end{aligned}$$

Motivation: Nonlinear Equations

We usually write a nonlinear system of equations as

$$F(x) = 0,$$

where $F : \mathbb{R}^n \to \mathbb{R}^m$

We implicity 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

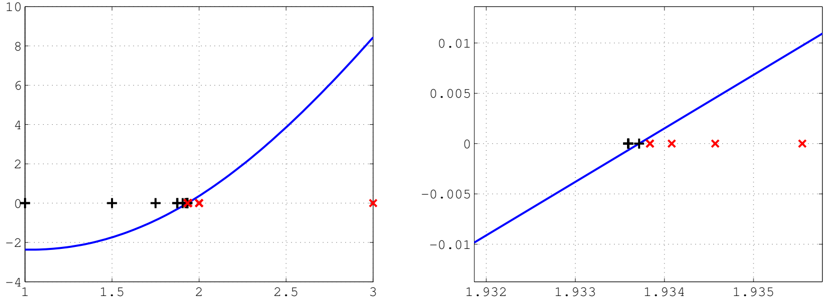
Probably the simplest is the bisection method for a scalar equation $f(x) = 0$, where $f \in C[a, b]$

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

Motivation: Nonlinear Equations

```
#!/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)
```

Motivation: Nonlinear Equations



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 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 year

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 $f : \mathbb{R}^n \to \mathbb{R}$ 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)

$$\min_x f(x_1, x_2) = 2\pi x_1(x_1 + x_2)$$

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

[3] Heath Example 6.2

Motivation: Optimization

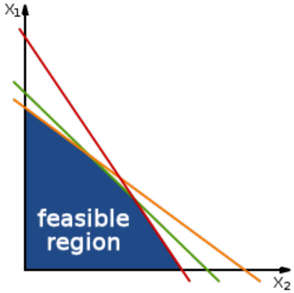
If f, g and h are all affine, then the optimization problem is called a linear program

(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

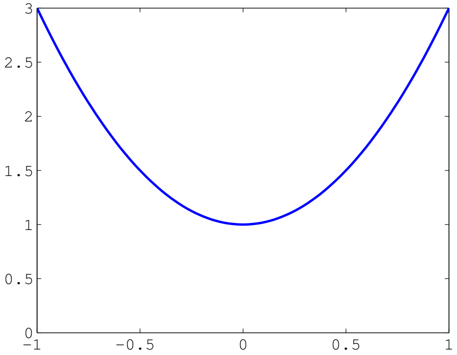
If the objective function or any of the constraints are nonlinear then we have a nonlinear optimization problem or nonlinear program

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

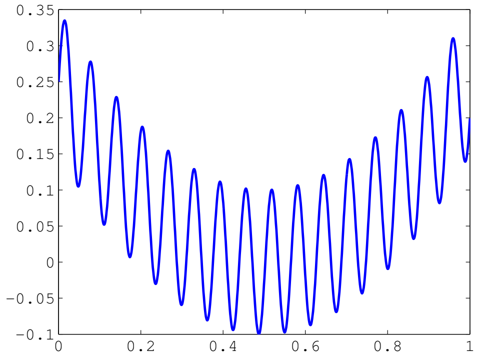
Motivation: Optimization

In some cases this easily gives a global minimum



Motivation: Optimization

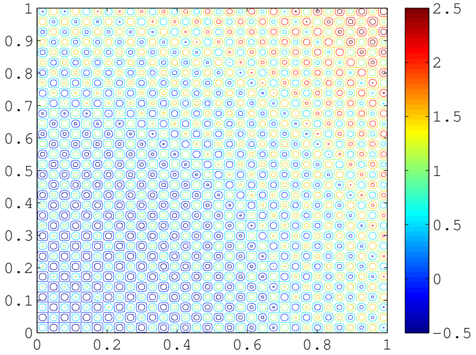
But in general, global optimization can be very difficult



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

$$x_{k+1} = g(x_k) \qquad (*)$$

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

Fixed-Point Iteration

Suppose α is such that $g(\alpha) = \alpha$, then we call α a fixed point of g

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

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

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[6]

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

[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 $L < 1$

Fixed-Point Iteration

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 α.

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)

Fixed-Point 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 α 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

Fixed-Point Iteration

By continuity of g' (and hence continuity of $|g'|$), for any $\epsilon > 0$ $\exists \delta > 0$ such that for $x \in (\alpha - \delta, \alpha + \delta)$:

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

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 α

Fixed-Point Iteration

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

Fixed-Point Iteration

We say that an iteration converges linearly if, for some $\mu \in (0, 1)$,

$$\lim_{k\to\infty} \frac{|x_{k+1} - \alpha|}{|x_k - \alpha|} = \mu$$

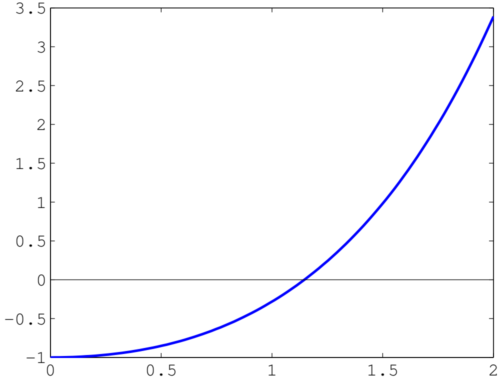
An iteration converges superlinearly if

$$\lim_{k\to\infty} \frac{|x_{k+1} - \alpha|}{|x_k - \alpha|} = 0$$

Fixed-Point Iteration

We can use these ideas to construct practical fixed-point iterations for solving $f(x) = 0$

e.g. suppose $f(x) = e^x - x - 2$



From the plot, it looks like there's a root at $x \approx 1.15$

Fixed-Point Iteration

$f(x) = 0$ is equivalent to $x = \log(x+2)$, hence we seek a fixed point of the iteration

$$x_{k+1} = \log(x_k + 2), \quad k = 0, 1, 2, \ldots$$

Here $g(x) \equiv \log(x+2)$, and $g'(x) = 1/(x+2) < 1$ for all $x > -1$, hence fixed point iteration will converge for $x_0 > -1$

Hence we should get linear convergence with factor approx. $g'(1.15) = 1/(1.15+2) \approx 0.32$

Fixed-Point Iteration

An alternative fixed-point iteration is to set

$$x_{k+1} = e^{x_k} - 2, \quad k = 0, 1, 2, \ldots$$

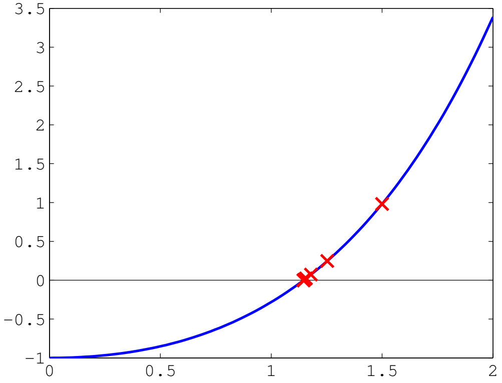
Therefore $g(x) \equiv e^x - 2$, and $g'(x) = e^x$

Hence $|g'(\alpha)| > 1$, so we can't guarantee convergence

(And, in fact, the iteration diverges...)

Fixed-Point Iteration

Python demo: Comparison of the two iterations



Newton's Method

Constructing fixed-point iterations can require some ingenuity

Need to rewrite $f(x) = 0$ in a form $x = g(x)$, with appropriate properties on g

To obtain a more generally applicable iterative method, let us consider the following fixed-point iteration

$$x_{k+1} = x_k - \lambda(x_k)f(x_k), \quad k = 0, 1, 2, \ldots$$

corresponding to $g(x) = x - \lambda(x)f(x)$, for some function λ

A fixed point α of g yields a solution to $f(\alpha) = 0$ (except possibly when $\lambda(\alpha) = 0$), which is what we're trying to achieve!

Newton's Method

Recall that the asymptotic convergence rate is dictated by $|g'(\alpha)|$, so we'd like to have $|g'(\alpha)| = 0$ to get superlinear convergence

Suppose (as stated above) that $f(\alpha) = 0$, then

$$g'(\alpha) = 1 - \lambda'(\alpha)f(\alpha) - \lambda(\alpha)f'(\alpha) = 1 - \lambda(\alpha)f'(\alpha)$$

Hence to satisfy $g'(\alpha) = 0$ we choose $\lambda(x) \equiv 1/f'(x)$ to get Newton's method:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \ldots$$

Newton's Method

Based on fixed-point iteration theory, Newton's method is convergent since $|g'(\alpha)| = 0 < 1$

However, we need a different argument to understand the superlinear convergence rate properly

To do this, we use a Taylor expansion for $f(\alpha)$ about $f(x_k)$:

$$0 = f(\alpha) = f(x_k) + (\alpha - x_k)f'(x_k) + \frac{(\alpha - x_k)^2}{2} f''(\theta_k)$$

for some $\theta_k \in (\alpha, x_k)$

Newton's Method

Dividing through by $f'(x_k)$ gives

$$\left(x_k - \frac{f(x_k)}{f'(x_k)}\right) - \alpha = \frac{f''(\theta_k)}{2f'(x_k)}(x_k - \alpha)^2,$$

or

$$x_{k+1} - \alpha = \frac{f''(\theta_k)}{2f'(x_k)}(x_k - \alpha)^2,$$

Hence, roughly speaking, the error at iteration $k+1$ is the square of the error at each iteration k

This is referred to as quadratic convergence, which is very rapid!

Key point: Once again we need to be sufficiently close to α to get quadratic convergence (result relied on Taylor expansion near α)

Secant Method

An alternative to Newton's method is to approximate $f'(x_k)$ using the finite difference

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Substituting this into the iteration leads to the secant method

$$x_{k+1} = x_k - f(x_k)\left(\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}\right), \quad k = 1, 2, 3, \ldots$$

The main advantages of secant are:

- does not require us to determine $f'(x)$ analytically
- requires only one extra function evaluation, $f(x_k)$, per iteration (Newton's method also requires $f'(x_k)$)

Secant Method

As one may expect, secant converges faster than a fixed-point iteration, but slower than Newton's method

In fact, it can be shown that for the secant method, we have

$$\lim_{k\to\infty} \frac{|x_{k+1} - \alpha|}{|x_k - \alpha|^q} = \mu$$

where μ is a positive constant and $q \approx 1.6$

Matlab demo: Newton's method vs. secant method for $f(x) = e^x - x - 2 = 0$

Multivariate Case

Systems of Nonlinear Equations

We now consider fixed-point iterations and Newton's method for systems of nonlinear equations

We suppose that $F : \mathbb{R}^n \to \mathbb{R}^n$, $n > 1$, and we seek a root $\alpha \in \mathbb{R}^n$ such that $F(\alpha) = 0$

In component form, this is equivalent to

$$\begin{aligned} F_1(\alpha) &= 0 \\ F_2(\alpha) &= 0 \\ &\vdots \\ F_n(\alpha) &= 0 \end{aligned}$$

Fixed-Point Iteration

For a fixed-point iteration, we again seek to rewrite $F(x) = 0$ as $x = G(x)$ to obtain:

$$x_{k+1} = G(x_k)$$

The convergence proof is the same as in the scalar case, if we replace $|\cdot|$ with $\|\cdot\|$

i.e. if $\|G(x) - G(y)\| \leq L\|x - y\|$, then $\|x_k - \alpha\| \leq L^k\|x_0 - \alpha\|$

Hence, as before, if G is a contraction it will converge to a fixed point α

Fixed-Point Iteration

Recall that we define the Jacobian matrix, $J_G \in \mathbb{R}^{n\times n}$, to be

$$(J_G)_{ij} = \frac{\partial G_i}{\partial x_j}, \quad i, j = 1, \ldots, n$$

If $\|J_g(\alpha)\|_\infty < 1$, then there is some neighborhood of α for which the fixed-point iteration converges to α

The proof of this is a natural extension of the corresponding scalar result

Fixed-Point Iteration

Once again, we can employ a fixed point iteration to solve $F(x) = 0$

e.g. consider

$$\begin{aligned} x_1^2 + x_2^2 - 1 &= 0 \\ 5x_1^2 + 21x_2^2 - 9 &= 0 \end{aligned}$$

This can be rearranged to $x_1 = \sqrt{1 - x_2^2}$, $x_2 = \sqrt{(9 - 5x_1^2)/21}$

Fixed-Point Iteration

Hence, we define

$$G_1(x_1, x_2) \equiv \sqrt{1 - x_2^2}, \;\; G_2(x_1, x_2) \equiv \sqrt{(9 - 5x_1^2)/21}$$

Matlab Example: This yields a convergent iterative method

Newton's Method

As in the one-dimensional case, Newton's method is generally more useful than a standard fixed-point iteration

The natural generalization of Newton's method is

$$x_{k+1} = x_k - J_F(x_k)^{-1} F(x_k), \quad k = 0, 1, 2, \ldots$$

Note that to put Newton's method in the standard form for a linear system, we write

$$J_F(x_k)\Delta x_k = -F(x_k), \quad k = 0, 1, 2, \ldots,$$

where $\Delta x_k \equiv x_{k+1} - x_k$

Newton's Method

Once again, if x_0 is sufficiently close to α, then Newton's method converges quadratically — we sketch the proof below

This result again relies on Taylor's Theorem

Hence we first consider how to generalize the familiar one-dimensional Taylor's Theorem to $\mathbb{R}^n$

First, we consider the case for $F : \mathbb{R}^n \to \mathbb{R}$

Multivariate Taylor Theorem

Let $\phi(s) \equiv F(x + s\delta)$, then one-dimensional Taylor Theorem yields

$$\phi(1) = \phi(0) + \sum_{\ell=1}^{k} \frac{\phi^{(\ell)}(0)}{\ell!} + \phi^{(k+1)}(\eta), \quad \eta \in (0,1),$$

Also, we have

$$\begin{aligned}
\phi(0) &= F(x) \\
\phi(1) &= F(x+\delta) \\
\phi'(s) &= \frac{\partial F(x+s\delta)}{\partial x_1}\delta_1 + \frac{\partial F(x+s\delta)}{\partial x_2}\delta_2 + \cdots + \frac{\partial F(x+s\delta)}{\partial x_n}\delta_n \\
\phi''(s) &= \frac{\partial^2 F(x+s\delta)}{\partial x_1^2}\delta_1^2 + \cdots + \frac{\partial^2 F(x+s\delta)}{\partial x_1 x_n}\delta_1\delta_n + \cdots + \\
&\quad \frac{\partial^2 F(x+s\delta)}{\partial x_1 \partial x_n}\delta_1\delta_n + \cdots + \frac{\partial^2 F(x+s\delta)}{\partial x_n^2}\delta_n^2 \\
&\vdots
\end{aligned}$$

Multivariate Taylor Theorem

Hence, we have

$$F(x+\delta) = F(x) + \sum_{\ell=1}^{k} \frac{U_\ell(\delta)}{\ell!} + E_k,$$

where

$$U_\ell(x) \equiv \left[\left(\frac{\partial}{\partial x_1}\delta_1 + \cdots + \frac{\partial}{\partial x_n}\delta_n \right)^\ell F \right](x), \quad \ell = 1, 2, \ldots, k,$$

and

$$E_k \equiv U_{k+1}(x+\eta\delta), \quad \eta \in (0,1)$$

Multivariate Taylor Theorem

Let A be an upper bound on the abs. values of all derivatives of order $k+1$, then

$$\begin{aligned}
|E_k| &\leq \frac{1}{(k+1)!}\left|(A,\ldots,A)^T\left(\|\delta\|_\infty^{k+1},\ldots,\|\delta\|_\infty^{k+1}\right)\right| \\
&= \frac{1}{(k+1)!}A\|\delta\|_\infty^{k+1}\left|(1,\ldots,1)^T(1,\ldots,1)\right| \\
&= \frac{n^{k+1}}{(k+1)!}A\|\delta\|_\infty^{k+1}
\end{aligned}$$

where the last line follows from the fact that there are n^{k+1} terms in the inner product (*i.e.* there are n^{k+1} derivatives of order $k+1$)

Multivariate Taylor Theorem

We shall only need an expansion up to first order terms for analysis of Newton's method

From our expression above, we can write first order Taylor expansion succinctly as:

$$F(x + \delta) = F(x) + \nabla F(x)^T \delta + E_1$$

Multivariate Taylor Theorem

For $F : \mathbb{R}^n \to \mathbb{R}^n$, Taylor expansion follows by developing a Taylor expansion for each F_i, hence

$$F_i(x + \delta) = F_i(x) + \nabla F_i(x)^T \delta + E_{i,1}$$

so that for $F : \mathbb{R}^n \to \mathbb{R}^n$ we have

$$F(x + \delta) = F(x) + J_F(x)\delta + E_F$$

where $\|E_F\|_\infty \leq \max_{1\leq i\leq n} |E_{i,1}| \leq \frac{1}{2} n^2 \left(\max_{1\leq i,j,\ell\leq n} \left| \frac{\partial^2 F_i}{\partial x_j \partial x_\ell} \right| \right) \|\delta\|_\infty^2$

Newton's Method

We now return to Newton's method

We have

$$0 = F(\alpha) = F(x_k) + J_F(x_k)\,[\alpha - x_k] + E_F$$

so that

$$x_k - \alpha = [J_F(x_k)]^{-1} F(x_k) + [J_F(x_k)]^{-1} E_F$$

Newton's Method

Also, the Newton iteration itself can be rewritten as

$$J_F(x_k)\,[x_{k+1}-\alpha] = J_F(x_k)\,[x_k-\alpha] - F(x_k)$$

Hence, we obtain:

$$x_{k+1}-\alpha = [J_F(x_k)]^{-1}E_F,$$

so that $\|x_{k+1}-\alpha\|_\infty \leq \mathrm{const.}\|x_k-\alpha\|_\infty^2$, *i.e.* quadratic convergence!

Newton's Method

Example: Newton's method for the two-point Gauss quadrature rule

Recall the system of equations

$$\begin{aligned}
F_1(x_1, x_2, w_1, w_2) &= w_1 + w_2 - 2 = 0 \\
F_2(x_1, x_2, w_1, w_2) &= w_1 x_1 + w_2 x_2 = 0 \\
F_3(x_1, x_2, w_1, w_2) &= w_1 x_1^2 + w_2 x_2^2 - 2/3 = 0 \\
F_4(x_1, x_2, w_1, w_2) &= w_1 x_1^3 + w_2 x_2^3 = 0
\end{aligned}$$

Newton's Method

We can solve this in Matlab using our own implementation of Newton's method

To do this, we require the Jacobian of this system:

$$J_F(x_1, x_2, w_1, w_2) = \begin{bmatrix} 0 & 0 & 1 & 1 \\ w_1 & w_2 & x_1 & x_2 \\ 2w_1x_1 & 2w_2x_2 & x_1^2 & x_2^2 \\ 3w_1x_1^2 & 3w_2x_2^2 & x_1^3 & x_2^3 \end{bmatrix}$$

Newton's Method

Alternatively, we can use Matlab's built-in `fsolve` function

```
>> help fsolve
FSOLVE solves systems of nonlinear equations of
several variables.

FSOLVE attempts to solve equations of the form:

F(X) = 0 where F and X may be vectors or matrices.
```

Note that `fsolve` computes a finite difference approximation to the Jacobian by default

(Or we can pass in an analytical Jacobian if we want)

Newton's Method

Matlab example: With either approach and with starting guess $x_0 = [-1, 1, 1, 1]$, we get

```
x_k =
   -0.577350269189626
    0.577350269189626
    1.000000000000000
    1.000000000000000
```