Motivation: Nonlinear Equations

So far we have mostly focused on linear phenomena

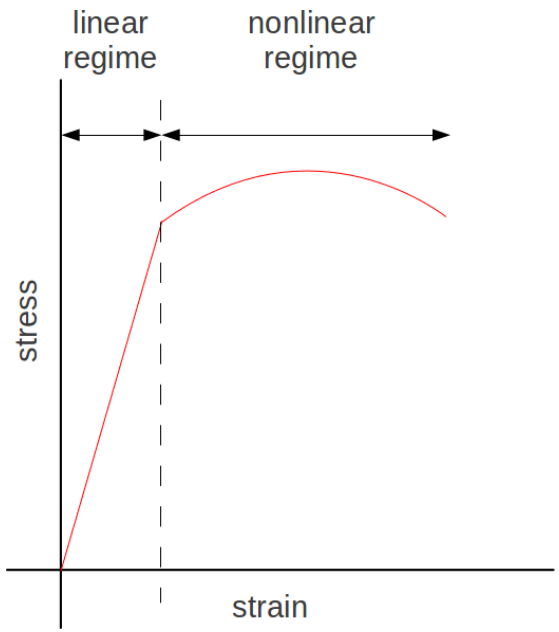
- Interpolation leads to a linear system $Vb = y$ (monomials) or $\mathrm{I}b = y$ (Lagrange polynomials)
- Linear least-squares leads to the normal equations $A^T Ab = A^T y$
- We saw examples of linear physical models (Ohm's Law, Hooke's Law, Leontief equations) $\implies Ax = b$
- F.D. discretization 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[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 $f \in C[a, b]$

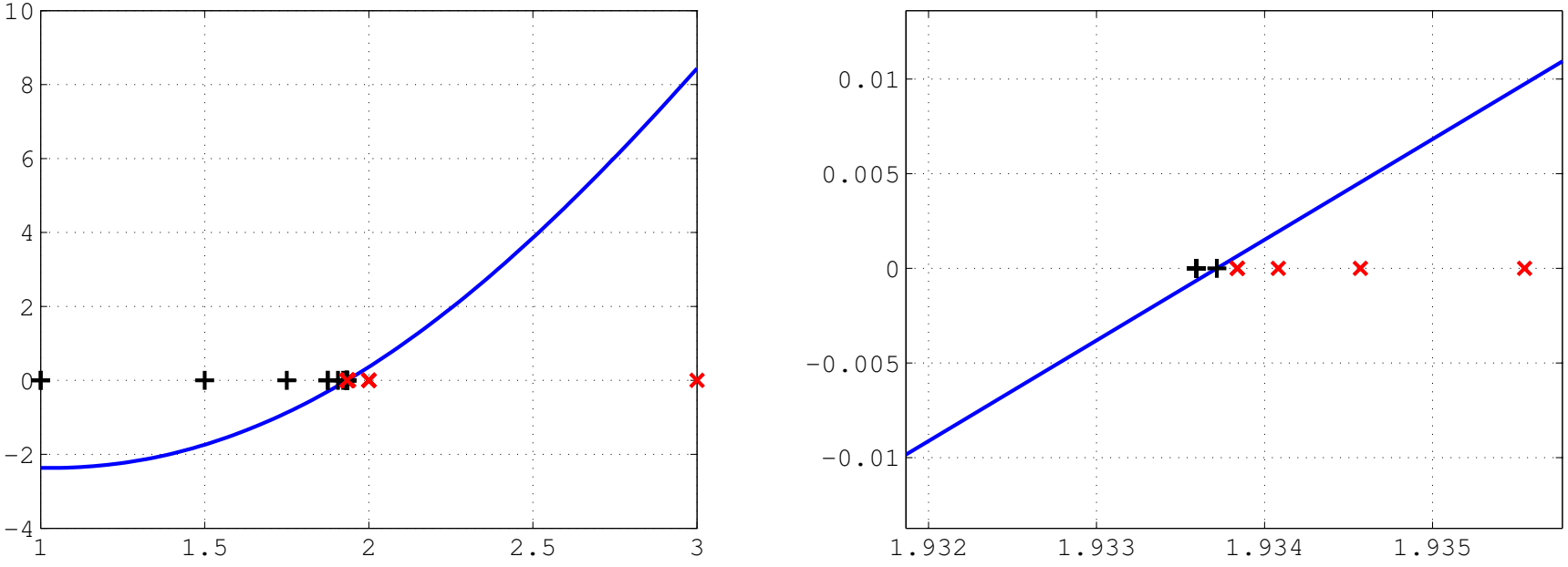
Look for a root in the interval $[a, b]$ by bisecting based on sign of f

Python example: [*bisection.py*]

Motivation: Nonlinear Equations

```
#!/usr/bin/python3
from math import sin

# 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" aspects 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

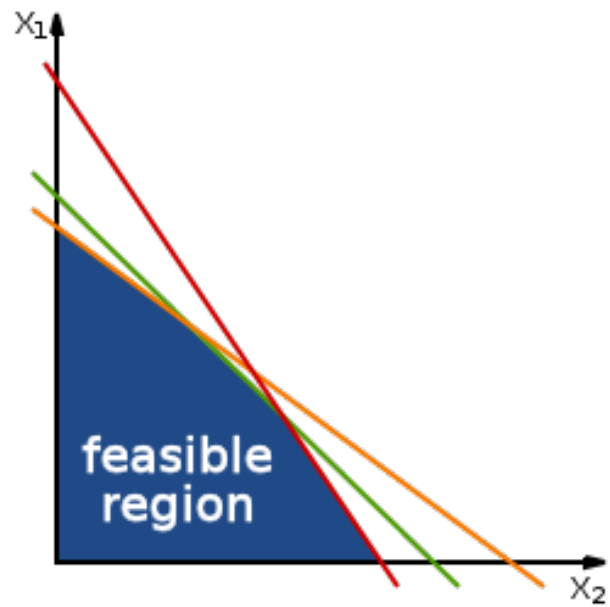
If f, g and h are all affine, then the optimization problem is called a linear program

(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

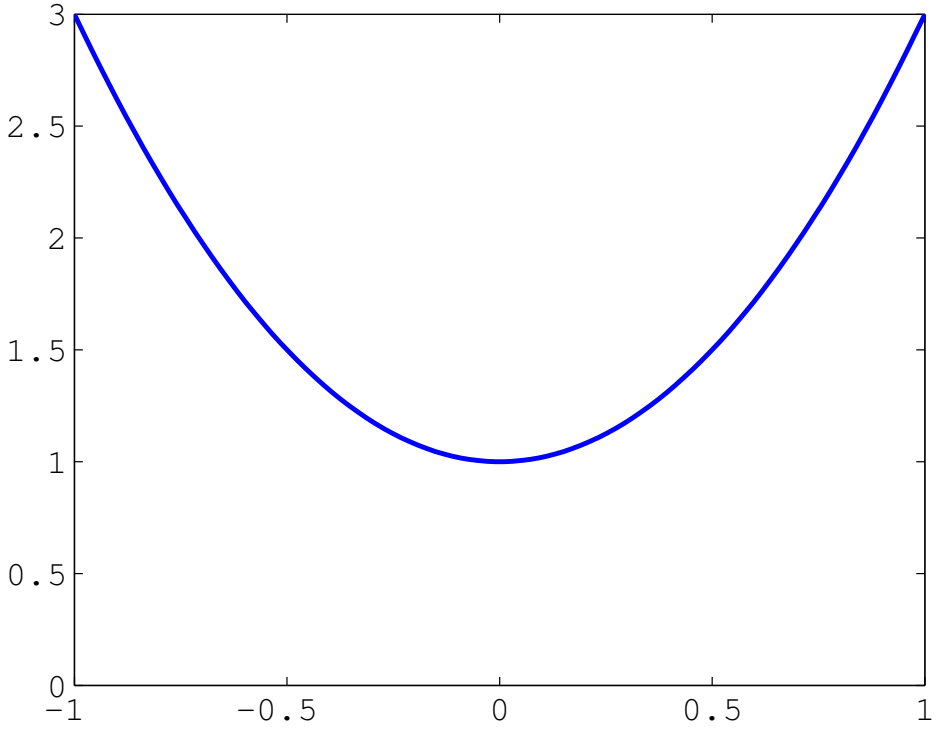
If the objective function or any of the constraints are nonlinear then we have a nonlinear optimization problem or nonlinear program

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

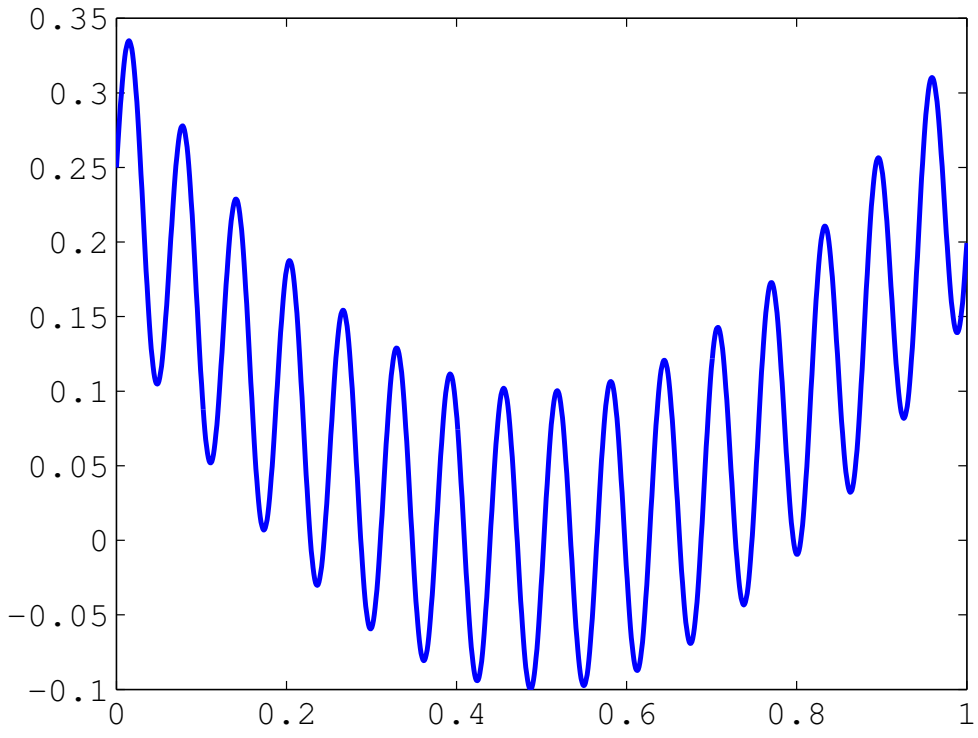
Motivation: Optimization

In some cases this easily gives a global minimum



Motivation: Optimization

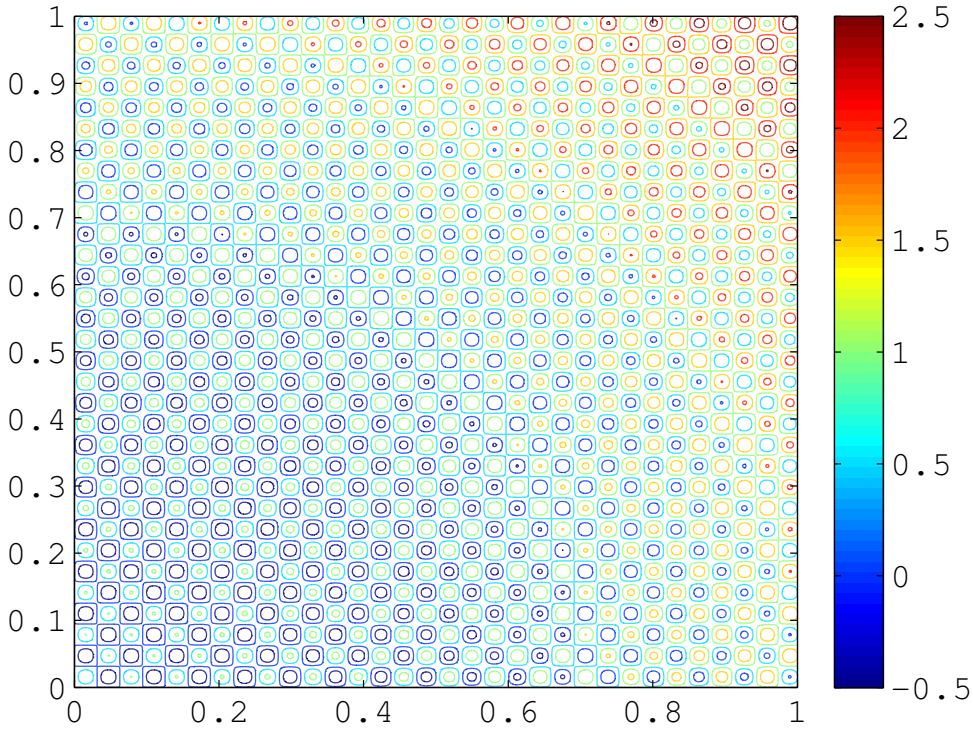
But in general, global optimization can be very difficult



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

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 result 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 $x \in (\alpha - \delta, \alpha + \delta)$:

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

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 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 $\mu \in (0, 1)$,

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

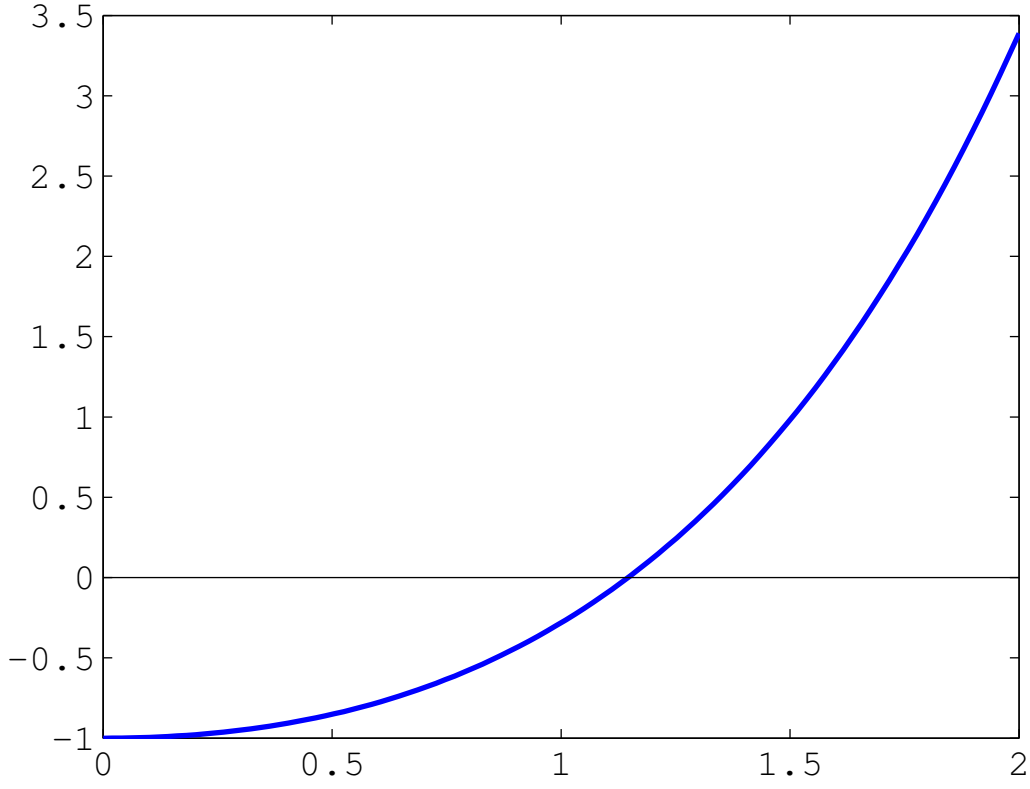
An iteration converges superlinearly if

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

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

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

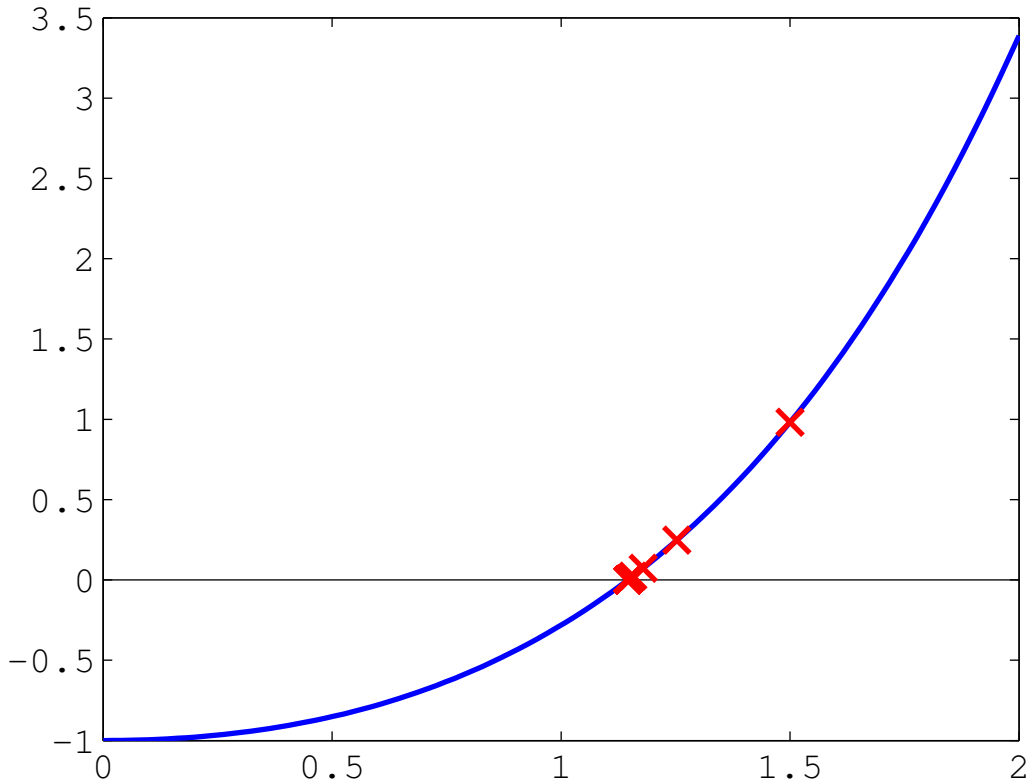
Therefore $g(x) \equiv e^x - 2$, and $g'(x) = e^x$

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

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

Fixed-Point Iteration

Python demo: [*iter.py*] 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 evaluation of $f(x)$ per solution (Newton's method also requires $f'(x_k)$ each iteration)

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 $q \approx 1.6$

Python demo: [*n_secant.py*] Newton's method versus 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

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

Python Example: [*iter_2d.py*] 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

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

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

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

Multivariate Taylor Theorem

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

$$\begin{aligned} |E_k| &\leq \frac{1}{(k+1)!} \left| \left[\left(\|\delta\|_\infty \frac{\partial}{\partial x_1} + \ldots + \|\delta\|_\infty \frac{\partial}{\partial x_n} \right)^{k+1} F \right] (x + \eta\delta) \right| \\ &= \frac{1}{(k+1)!} \|\delta\|_\infty^{k+1} \left| \left[\left(\frac{\partial}{\partial x_1} + \ldots + \frac{\partial}{\partial x_n} \right)^{k+1} F \right] (x + \eta\delta) \right| \\ &= \frac{n^{k+1}}{(k+1)!} A \|\delta\|_\infty^{k+1} \end{aligned}$$

where the last line follows from the fact that there are n^{k+1} terms in the 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 $\|x_{k+1} - \alpha\|_\infty \leq \text{const.} \|x_k - \alpha\|_\infty^2$, *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 Python 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 Python's built-in `fsolve` function

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

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

Matlab has an equivalent `fsolve` function.

Newton's Method

Python example: [*fsolve.py*] With either approach and with starting guess $x_0 = [-1, 1, 1, 1]$, we get

```
x_k =
  -0.57735026918962
   0.57735026918962
   1.00000000000000
   1.00000000000000
```

Conditions for Optimality

Existence of Global Minimum

In order to guarantee existence and uniqueness of a global min. we need to make assumptions about the objective function

e.g. if f is continuous on a closed[7] and bounded set $S \subset \mathbb{R}^n$ then it has global minimum in S

In one dimension, this says f achieves a minimum on the interval $[a, b] \subset \mathbb{R}$

In general f does not achieve a minimum on (a, b), *e.g.* consider $f(x) = x$

(Though $\inf_{x \in (a,b)} f(x)$, the largest lower bound of f on (a, b), is well-defined)

[7] A set is closed if it contains its own boundary

Existence of Global Minimum

Another helpful concept for existence of global min. is coercivity

A continuous function f on an unbounded set $S \subset \mathbb{R}^n$ is coercive if

$$\lim_{\|x\| \to \infty} f(x) = +\infty$$

That is, $f(x)$ must be large whenever $\|x\|$ is large

Existence of Global Minimum

If f is coercive on a closed, unbounded[8] set S, then f has a global minimum in S

Proof: From the definition of coercivity, for any $M \in \mathbb{R}$, $\exists r > 0$ such that $f(x) \geq M$ for all $x \in S$ where $\|x\| \geq r$

Suppose that $0 \in S$, and set $M = f(0)$

Let $Y \equiv \{x \in S : \|x\| \geq r\}$, so that $f(x) \geq f(0)$ for all $x \in Y$

And we already know that f achieves a minimum (which is at most $f(0)$) on the closed, bounded set $\{x \in S : \|x\| \leq r\}$

Hence f achieves a minimum on S $\square$

[8] *e.g.* S could be all of $\mathbb{R}^n$, or a "closed strip" in $\mathbb{R}^n$

Existence of Global Minimum

For example:

- $f(x, y) = x^2 + y^2$ is coercive on $\mathbb{R}^2$ (global min. at $(0, 0)$)
- $f(x) = x^3$ is not coercive on $\mathbb{R}$ ($f \to -\infty$ for $x \to -\infty$)
- $f(x) = e^x$ is not coercive on $\mathbb{R}$ ($f \to 0$ for $x \to -\infty$)

Convexity

An important concept for uniqueness is convexity

A set $S \subset \mathbb{R}^n$ is convex if it contains the line segment between any two of its points

That is, S is convex if for any $x, y \in S$, we have

$$\{\theta x + (1-\theta)y : \theta \in [0,1]\} \subset S$$

Convexity

Similarly, we define convexity of a function $f : S \subset \mathbb{R}^n \to \mathbb{R}$

f is convex if its graph along any line segment in S is on or below the chord connecting the function values

i.e. f is convex if for any $x, y \in S$ and any $\theta \in (0, 1)$, we have

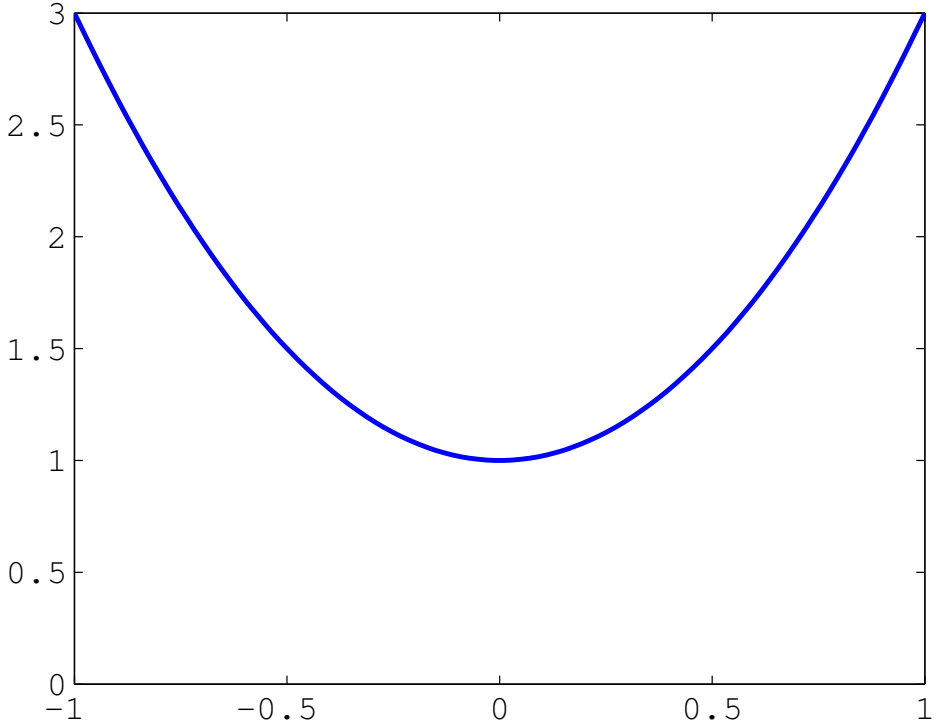
$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$$

Also, if

$$f(\theta x + (1 - \theta)y) < \theta f(x) + (1 - \theta)f(y)$$

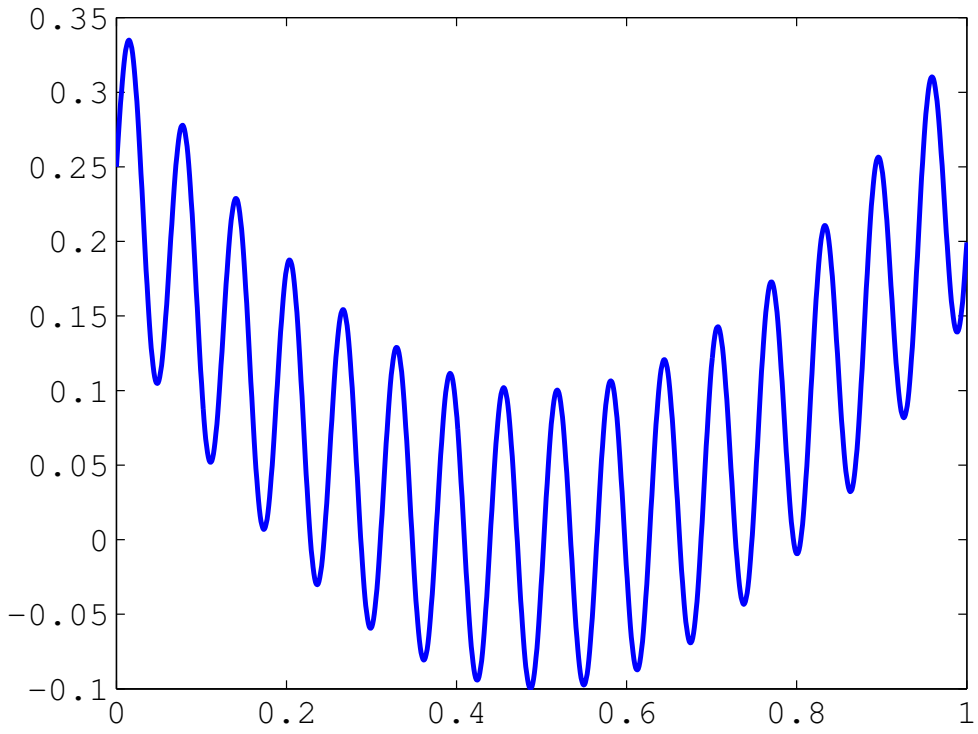
then f is strictly convex

Convexity



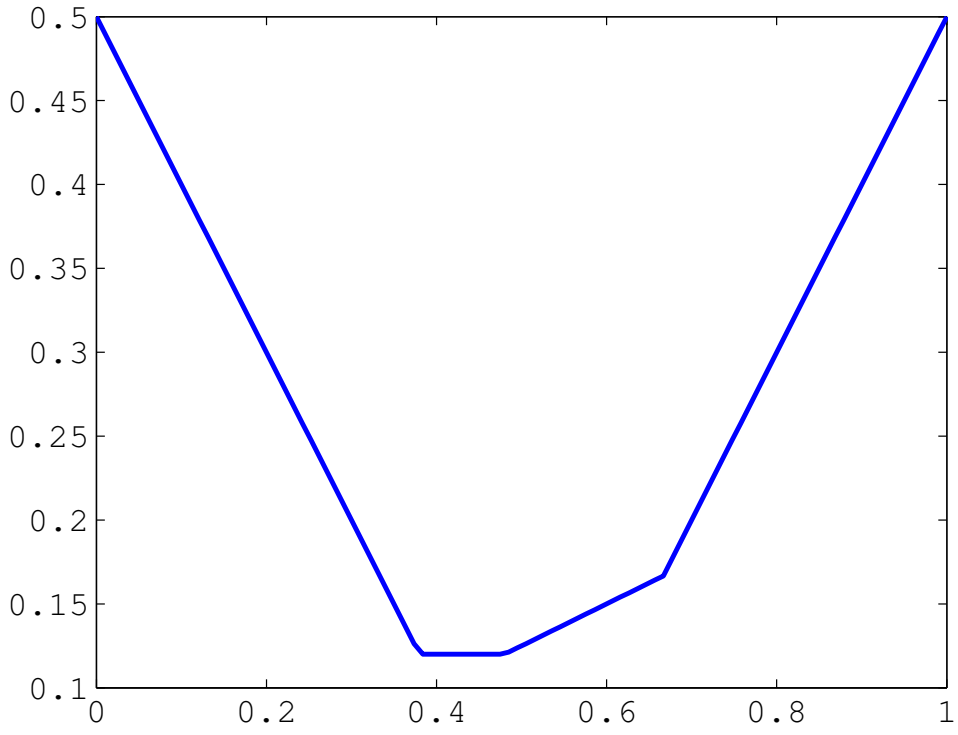
Strictly convex

Convexity



Non-convex

Convexity



Convex (not strictly convex)

Convexity

If f is a convex function on a convex set S, then any local minimum of f must be a global minimum[9]

Proof: Suppose x is a local minimum, *i.e.* $f(x) \leq f(y)$ for $y \in B(x, \epsilon)$ (where $B(x, \epsilon) \equiv \{y \in S : \|y - x\| \leq \epsilon\}$)

Suppose that x is not a global minimum, *i.e.* that there exists $w \in S$ such that $f(w) < f(x)$

(Then we will show that this gives a contradiction)

[9] A global minimum is defined as a point z such that $f(z) \leq f(x)$ for all $x \in S$. Note that a global minimum may not be unique, *e.g.* if $f(x) = -\cos x$ then 0 and 2π are both global minima.

Convexity

Proof (continued . . .):

For $\theta \in [0,1]$ we have $f(\theta w + (1-\theta)x) \leq \theta f(w) + (1-\theta)f(x)$

Let $\sigma \in (0,1]$ be sufficiently small so that

$$z \equiv \sigma w + (1-\sigma)\, x \in B(x, \epsilon)$$

Then

$$f(z) \leq \sigma f(w) + (1-\sigma)\, f(x) < \sigma f(x) + (1-\sigma)\, f(x) = f(x),$$

i.e. $f(z) < f(x)$, which contradicts that $f(x)$ is a local minimum!

Hence we cannot have $w \in S$ such that $f(w) < f(x)$ $\quad \square$

Convexity

Note that convexity does not guarantee uniqueness of global minimum

e.g. a convex function can clearly have a "horizontal" section (see earlier plot)

If f is a strictly convex function on a convex set S, then a local minimum of f is the unique global minimum

Optimization of convex functions over convex sets is called convex optimization, which is an important subfield of optimization

Optimality Conditions

We have discussed existence and uniqueness of minima, but haven't considered how to find a minimum

The familiar optimization idea from calculus in one dimension is: set derivative to zero, check the sign of the second derivative

This can be generalized to $\mathbb{R}^n$

Optimality Conditions

If $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable, then the gradient vector $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ is

$$\nabla f(x) \equiv \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

The importance of the gradient is that ∇f points "uphill," *i.e.* towards points with larger values than $f(x)$

And similarly $-\nabla f$ points "downhill"

Optimality Conditions

This follows from Taylor's theorem for $f : \mathbb{R}^n \to \mathbb{R}$

Recall that

$$f(x + \delta) = f(x) + \nabla f(x)^T \delta + \text{H.O.T.}$$

Let $\delta \equiv -\epsilon \nabla f(x)$ for $\epsilon > 0$ and suppose that $\nabla f(x) \neq 0$, then:

$$f(x - \epsilon \nabla f(x)) \approx f(x) - \epsilon \nabla f(x)^T \nabla f(x) < f(x)$$

Also, we see from Cauchy–Schwarz that $-\nabla f(x)$ is the steepest descent direction

Optimality Conditions

Similarly, we see that a necessary condition for a local minimum at $x^* \in S$ is that $\nabla f(x^*) = 0$

In this case there is no "downhill direction" at x^*

The condition $\nabla f(x^*) = 0$ is called a first-order necessary condition for optimality, since it only involves first derivatives

Optimality Conditions

$x^* \in S$ that satisfies the first-order optimality condition is called a critical point of f

But of course a critical point can be a local min., local max., or saddle point

(Recall that a saddle point is where some directions are "downhill" and others are "uphill", *e.g.* $(x, y) = (0, 0)$ for $f(x, y) = x^2 - y^2$)

Optimality Conditions

As in the one-dimensional case, we can look to second derivatives to classify critical points

If $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable, then the Hessian is the matrix-valued function $H_f : \mathbb{R}^n \to \mathbb{R}^{n \times n}$

$$H_f(x) \equiv \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n x_1} & \frac{\partial^2 f(x)}{\partial x_n x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

The Hessian is the Jacobian matrix of the gradient $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$

If the second partial derivatives of f are continuous, then $\partial^2 f / \partial x_i \partial x_j = \partial^2 f / \partial x_j \partial x_i$, and H_f is symmetric

Optimality Conditions

Suppose we have found a critical point x^*, so that $\nabla f(x^*) = 0$

From Taylor's Theorem, for $\delta \in \mathbb{R}^n$, we have

$$\begin{aligned} f(x^* + \delta) &= f(x^*) + \nabla f(x^*)^T \delta + \frac{1}{2}\delta^T H_f(x^* + \eta\delta)\delta \\ &= f(x^*) + \frac{1}{2}\delta^T H_f(x^* + \eta\delta)\delta \end{aligned}$$

for some $\eta \in (0, 1)$

Optimality Conditions

Recall positive definiteness: A is positive definite if $x^T Ax > 0$

Suppose $H_f(x^*)$ is positive definite

Then (by continuity) $H_f(x^* + \eta\delta)$ is also positive definite for $\|\delta\|$ sufficiently small, so that: $\delta^T H_f(x^* + \eta\delta)\delta > 0$

Hence, we have $f(x^* + \delta) > f(x^*)$ for $\|\delta\|$ sufficiently small, *i.e.* $f(x^*)$ is a local minimum

Hence, in general, positive definiteness of H_f at a critical point x^* is a second-order sufficient condition for a local minimum

Optimality Conditions

A matrix can also be negative definite: $x^T Ax < 0$ for all $x \neq 0$

Or indefinite: There exists x, y such that $x^T Ax < 0 < y^T Ay$

Then we can classify critical points as follows:

- $H_f(x^*)$ positive definite $\implies$ x^* is a local minimum
- $H_f(x^*)$ negative definite $\implies$ x^* is a local maximum
- $H_f(x^*)$ indefinite $\implies$ x^* is a saddle point

Optimality Conditions

Also, positive definiteness of the Hessian is closely related to convexity of f

If $H_f(x)$ is positive definite, then f is convex on some convex neighborhood of x

If $H_f(x)$ is positive definite for all $x \in S$, where S is a convex set, then f is convex on S

Question: How do we test for positive definiteness?

Optimality Conditions

Answer: A is positive (resp. negative) definite if and only if all eigenvalues of A are positive (resp. negative)[10]

Also, a matrix with positive and negative eigenvalues is indefinite

Hence we can compute all the eigenvalues of A and check their signs

[10] This is related to the Rayleigh quotient, see Unit 5

Heath Example 6.5

Consider

$$f(x) = 2x_1^3 + 3x_1^2 + 12x_1x_2 + 3x_2^2 - 6x_2 + 6$$

Then

$$\nabla f(x) = \begin{bmatrix} 6x_1^2 + 6x_1 + 12x_2 \\ 12x_1 + 6x_2 - 6 \end{bmatrix}$$

We set $\nabla f(x) = 0$ to find critical points[11] $[1, -1]^T$ and $[2, -3]^T$

[11] In general solving $\nabla f(x) = 0$ requires an iterative method

Heath Example 6.5, continued ...

The Hessian is

$$H_f(x) = \begin{bmatrix} 12x_1 + 6 & 12 \\ 12 & 6 \end{bmatrix}$$

and hence

$$H_f(1,-1) = \begin{bmatrix} 18 & 12 \\ 12 & 6 \end{bmatrix}, \text{ which has eigenvalues } 25.4, -1.4$$

$$H_f(2,-3) = \begin{bmatrix} 30 & 12 \\ 12 & 6 \end{bmatrix}, \text{ which has eigenvalues } 35.0, 1.0$$

Hence $[2,-3]^T$ is a local min. whereas $[1,-1]^T$ is a saddle point

Optimality Conditions: Equality Constrained Case

So far we have ignored constraints

Let us now consider equality constrained optimization

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad g(x) = 0,$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$, with $m \leq n$

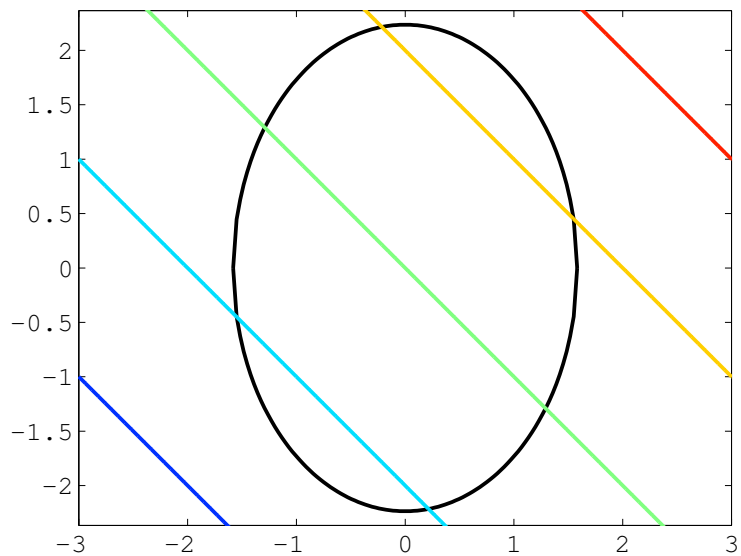
Since g maps to $\mathbb{R}^m$, we have m constraints

This situation is treated with Lagrange mutlipliers

Optimality Conditions: Equality Constrained Case

We illustrate the concept of Lagrange multipliers for $f, g : \mathbb{R}^2 \to \mathbb{R}$

Let $f(x, y) = x + y$ and $g(x, y) = 2x^2 + y^2 - 5$



∇g is normal to S:[12] at any $x \in S$ we must move in direction $(\nabla g(x))_\perp$ (tangent direction) to remain in S

[12] This follows from Taylor's Theorem: $g(x + \delta) \approx g(x) + \nabla g(x)^T \delta$

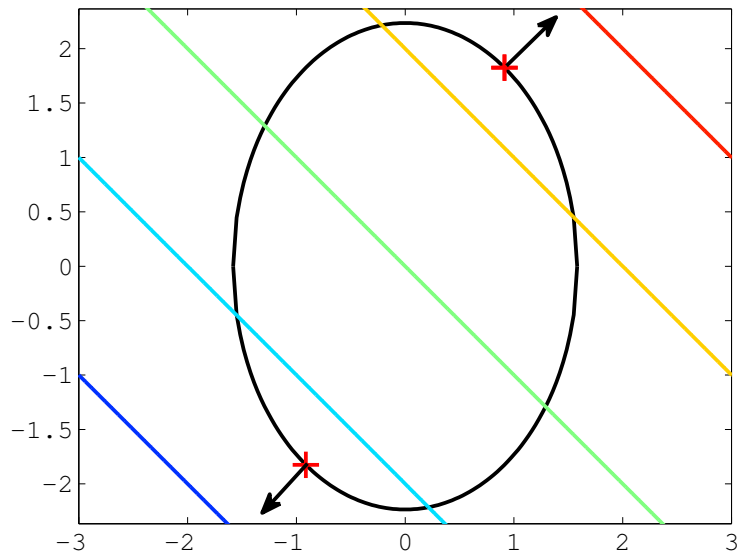
Optimality Conditions: Equality Constrained Case

Also, change in f due to infinitesimal step in direction $(\nabla g(x))_\perp$ is

$$f(x \pm \epsilon(\nabla g(x))_\perp) = f(x) \pm \epsilon \nabla f(x)^T (\nabla g(x))_\perp + \text{H.O.T.}$$

Hence stationary point $x^* \in S$ if $\nabla f(x^*)^T (\nabla g(x^*))_\perp = 0$, or

$$\nabla f(x^*) = \lambda^* \nabla g(x^*), \quad \text{for some } \lambda^* \in \mathbb{R}$$



Optimality Conditions: Equality Constrained Case

This shows that for a stationary point with $m = 1$ constraints, ∇f cannot have any component in the "tangent direction" to S

Now, consider the case with $m > 1$ equality constraints

Then $g : \mathbb{R}^n \to \mathbb{R}^m$ and we now have a set of constraint gradient vectors, ∇g_i, $i = 1, \ldots, m$

Then we have $S = \{x \in \mathbb{R}^n : g_i(x) = 0, i = 1, \ldots, m\}$

Any "tangent direction" at $x \in S$ must be orthogonal to all gradient vectors $\{\nabla g_i(x), i = 1, \ldots, m\}$ to remain in S

Optimality Conditions: Equality Constrained Case

Let $\mathcal{T}(x) \equiv \{v \in \mathbb{R}^n : \nabla g_i(x)^T v = 0, i = 1, 2, \ldots, m\}$ denote the orthogonal complement of $\{\nabla g_i(x), i = 1, \ldots, m\}$

Then, for $\delta \in \mathcal{T}(x)$ and $\epsilon \in \mathbb{R}_{>0}$, $\epsilon\delta$ is a step in a "tangent direction" of S at x

Since we have

$$f(x^* + \epsilon\delta) = f(x^*) + \epsilon \nabla f(x^*)^T \delta + \text{H.O.T.}$$

it follows that for a stationary point we need $\nabla f(x^*)^T \delta = 0$ for all $\delta \in \mathcal{T}(x^*)$

Optimality Conditions: Equality Constrained Case

Hence, we require that at a stationary point $x^* \in S$ we have

$$\nabla f(x^*) \in \text{span}\{\nabla g_i(x^*), i = 1, \ldots, m\}$$

This can be written succinctly as a linear system

$$\nabla f(x^*) = (J_g(x^*))^T \lambda^*$$

for some $\lambda^* \in \mathbb{R}^m$, where $(J_g(x^*))^T \in \mathbb{R}^{n \times m}$

This follows because the columns of $(J_g(x^*))^T$ are the vectors $\{\nabla g_i(x^*), i = 1, \ldots, m\}$

Optimality Conditions: Equality Constrained Case

We can write equality constrained optimization problems more succinctly by introducing the Lagrangian function, $\mathcal{L} : \mathbb{R}^{n+m} \to \mathbb{R}$,

$$\begin{aligned} \mathcal{L}(x, \lambda) &\equiv f(x) + \lambda^T g(x) \\ &= f(x) + \lambda_1 g_1(x) + \cdots + \lambda_m g_m(x) \end{aligned}$$

Then we have,

$$\frac{\partial \mathcal{L}(x,\lambda)}{\partial x_i} = \frac{\partial f(x)}{\partial x_i} + \lambda_1 \frac{\partial g_1(x)}{\partial x_i} + \cdots + \lambda_n \frac{\partial g_n(x)}{\partial x_i}, \quad i = 1, \ldots, n$$

$$\frac{\partial \mathcal{L}(x,\lambda)}{\partial \lambda_i} = g_i(x), \quad i = 1, \ldots, m$$

Optimality Conditions: Equality Constrained Case

Hence

$$\nabla \mathcal{L}(x, \lambda) = \begin{bmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ \nabla_\lambda \mathcal{L}(x, \lambda) \end{bmatrix} = \begin{bmatrix} \nabla f(x) + J_g(x)^T \lambda \\ g(x) \end{bmatrix},$$

so that the first order necessary condition for optimality for the constrained problem can be written as a nonlinear system:[13]

$$\nabla \mathcal{L}(x, \lambda) = \begin{bmatrix} \nabla f(x) + J_g(x)^T \lambda \\ g(x) \end{bmatrix} = 0$$

(As before, stationary points can be classified by considering the Hessian, though we will not consider this here . . .)

[13] $n + m$ variables, $n + m$ equations

Optimality Conditions: Equality Constrained Case

See Lecture: Constrained optimization of cylinder surface area

Optimality Conditions: Equality Constrained Case

As another example of equality constrained optimization, recall our underdetermined linear least squares problem from Unit 1

$$\min_{b \in \mathbb{R}^n} f(b) \quad \text{subject to} \quad g(b) = 0,$$

where $f(b) \equiv b^T b$, $g(b) \equiv Ab - y$ and $A \in \mathbb{R}^{m \times n}$ with $m < n$

Optimality Conditions: Equality Constrained Case

Introducing Lagrange multipliers gives

$$\mathcal{L}(b, \lambda) \equiv b^T b + \lambda^T (Ab - y)$$

where $b \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}^m$

Hence $\nabla \mathcal{L}(b, \lambda) = 0$ implies

$$\begin{bmatrix} \nabla f(b) + J_g(b)^T \lambda \\ g(b) \end{bmatrix} = \begin{bmatrix} 2b + A^T \lambda \\ Ab - y \end{bmatrix} = 0 \in \mathbb{R}^{n+m}$$

Optimality Conditions: Equality Constrained Case

Hence, we obtain the $(n+m) \times (n+m)$ square linear system

$$\begin{bmatrix} 2\mathrm{I} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} b \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix}$$

which we can solve for $\begin{bmatrix} b \\ \lambda \end{bmatrix} \in \mathbb{R}^{n+m}$

Optimality Conditions: Equality Constrained Case

We have $b = -\frac{1}{2}A^T\lambda$ from the first "block row"

Subsituting into $Ab = y$ (the second "block row") yields
$\lambda = -2(AA^T)^{-1}y$

And hence

$$b = -\frac{1}{2}A^T\lambda = A^T(AA^T)^{-1}y$$

which was the solution we introduced (but didn't derive) in Unit 1

Optimality Conditions: Inequality Constrained Case

Similar Lagrange multiplier methods can be developed for the more difficult case of inequality constrained optimization

Steepest Descent

We first consider the simpler case of unconstrained optimization (as opposed to constrained optimization)

Perhaps the simplest method for unconstrained optimization is steepest descent

Key idea: The negative gradient $-\nabla f(x)$ points in the "steepest downhill" direction for f at x

Hence an iterative method for minimizing f is obtained by following $-\nabla f(x_k)$ at each step

Question: How far should we go in the direction of $-\nabla f(x_k)$?

Steepest Descent

We can try to find the best step size via a subsidiary (and easier!) optimization problem

For a direction $s \in \mathbb{R}^n$, let $\phi : \mathbb{R} \to \mathbb{R}$ be given by

$$\phi(\eta) = f(x + \eta s)$$

Then minimizing f along s corresponds to minimizing the one-dimensional function ϕ

This process of minimizing f along a line is called a line search[14]

[14] The line search can itself be performed via Newton's method, as described for $f : \mathbb{R}^n \to \mathbb{R}$ shortly, or via a built-in function

Steepest Descent

Putting these pieces together leads to the steepest descent method:

1: choose initial guess x_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: $\quad s_k = -\nabla f(x_k)$
4: $\quad$ choose η_k to minimize $f(x_k + \eta_k s_k)$
5: $\quad x_{k+1} = x_k + \eta_k s_k$
6: **end for**

However, steepest descent often converges very slowly

Convergence rate is linear, and scaling factor can be arbitrarily close to 1

(Steepest descent will be covered on Assignment 5)

Newton's Method

We can get faster convergence by using more information about f

Note that $\nabla f(x^*) = 0$ is a system of nonlinear equations, hence we can solve it with quadratic convergence via Newton's method[15]

The Jacobian matrix of $\nabla f(x)$ is $H_f(x)$ and hence Newton's method for unconstrained optimization is:

1: choose initial guess x_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: solve $H_f(x_k)s_k = -\nabla f(x_k)$
4: $x_{k+1} = x_k + s_k$
5: **end for**

[15] Note that in its simplest form this algorithm searches for stationary points, not necessarily minima

Newton's Method

We can also interpret Newton's method as seeking stationary point based on a sequence of local quadratic approximations

Recall that for small δ

$$f(x+\delta) \approx f(x) + \nabla f(x)^T \delta + \frac{1}{2}\delta^T H_f(x)\delta \equiv q(\delta)$$

where $q(\delta)$ is quadratic in δ (for a fixed x)

We find stationary point of q in the usual way:[16]

$$\nabla q(\delta) = \nabla f(x) + H_f(x)\delta = 0$$

This leads to $H_f(x)\delta = -\nabla f(x)$, as in the previous slide

[16] Recall Unit 1 for differentiation of $\delta^T H_f(x)\delta$

Newton's Method

Python example: Newton's method for minimization of Himmelblau's function

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

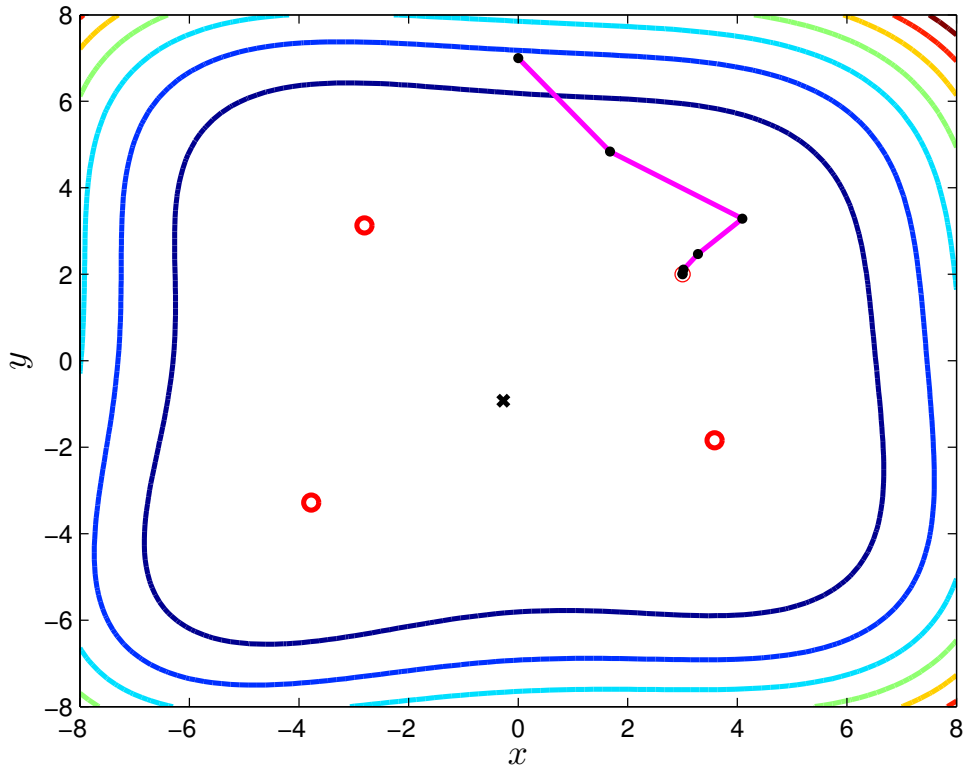
Local maximum of 181.617 at $(-0.270845, -0.923039)$

Four local minima, each of 0, at

$$(3, 2),\ (-2.805, 3.131),\ (-3.779, -3.283),\ (3.584, -1.841)$$

Newton's Method

Python example: [*h_ newton.py*] Newton's method for minimization of Himmelblau's function



Newton's Method: Robustness

Newton's method generally converges much faster than steepest descent

However, Newton's method can be unreliable far away from a solution

To improve robustness during early iterations it is common to perform a line search in the Newton-step-direction

Also line search can ensure we don't approach a local max. as can happen with raw Newton method

The line search modifies the Newton step size, hence often referred to as a damped Newton method

Newton's Method: Robustness

Another way to improve robustness is with trust region methods

At each iteration k, a "trust radius" R_k is computed

This determines a region surrounding x_k on which we "trust" our quadratic approx.

We require $\|x_{k+1} - x_k\| \leq R_k$, hence constrained optimization problem (with quadratic objective function) at each step

Newton's Method: Robustness

Size of R_{k+1} is based on comparing actual change, $f(x_{k+1}) - f(x_k)$, to change predicted by the quadratic model

If quadratic model is accurate, we expand the trust radius, otherwise we contract it

When close to a minimum, R_k should be large enough to allow full Newton steps $\implies$ eventual quadratic convergence

Quasi-Newton Methods

Newton's method is effective for optimization, but it can be unreliable, expensive, and complicated

- Unreliable: Only converges when sufficiently close to a minimum
- Expensive: The Hessian H_f is dense in general, hence very expensive if n is large
- Complicated: Can be impractical or laborious to derive the Hessian

Hence there has been much interest in so-called quasi-Newton methods, which do not require the Hessian

Quasi-Newton Methods

General form of quasi-Newton methods:

$$x_{k+1} = x_k - \alpha_k B_k^{-1} \nabla f(x_k)$$

where α_k is a line search parameter and B_k is some approximation to the Hessian

Quasi-Newton methods generally lose quadratic convergence of Newton's method, but often superlinear convergence is achieved

We now consider some specific quasi-Newton methods

BFGS

The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is one of the most popular quasi-Newton methods:

1: choose initial guess x_0
2: choose B_0, initial Hessian guess, *e.g.* $B_0 = \mathrm{I}$
3: **for** $k = 0, 1, 2, \ldots$ **do**
4: solve $B_k s_k = -\nabla f(x_k)$
5: $x_{k+1} = x_k + s_k$
6: $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
7: $B_{k+1} = B_k + \Delta B_k$
8: **end for**

where

$$\Delta B_k \equiv \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

BFGS

See lecture: derivation of the Broyden root-finding algorithm

See lecture: derivation of the BFGS algorithm

Basic idea is that B_k accumulates second derivative information on successive iterations, eventually approximates H_f well

BFGS

Actual implementation of BFGS: store and update inverse Hessian to avoid solving linear system:

1: choose initial guess x_0
2: choose H_0, initial inverse Hessian guess, *e.g.* $H_0 = \mathrm{I}$
3: **for** $k = 0, 1, 2, \ldots$ **do**
4: calculate $s_k = -H_k \nabla f(x_k)$
5: $x_{k+1} = x_k + s_k$
6: $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
7: $H_{k+1} = \Delta H_k$
8: **end for**

where

$$\Delta H_k \equiv (I - s_k \rho_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \qquad \rho_k = \frac{1}{y_k^T s_k}$$

BFGS

[*h_bfgs.py*] BFGS is implemented as the `fmin_bfgs` function in `scipy.optimize`

Also, BFGS (+ trust region) is implemented in MATLAB's `fminunc` function, *e.g.*

```
x0 = [5;5];
options = optimset('GradObj','on');
[x,fval,exitflag,output] = ...
  fminunc(@himmelblau_function,x0,options);
```

Conjugate Gradient Method

The conjugate gradient (CG) method is another alternative to Newton's method that does not require the Hessian:[17]

choose initial guess x_0
$g_0 = \nabla f(x_0)$
$x_0 = -g_0$
for $k = 0, 1, 2, \ldots$ **do**
    choose η_k to minimize $f(x_k + \eta_k s_k)$
    $x_{k+1} = x_k + \eta_k s_k$
    $g_{k+1} = \nabla f(x_{k+1})$
    $\beta_{k+1} = (g_{k+1}^T g_{k+1})/(g_k^T g_k)$
    $s_{k+1} = -g_{k+1} + \beta_{k+1} s_k$
end for

[17] We will look at this method in more detail in Unit 5.

Constrained Optimization

Equality Constrained Optimization

We now consider equality constrained minimization:

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad g(x) = 0,$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$

With the Lagrangian $\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x)$, we recall from that necessary condition for optimality is

$$\nabla \mathcal{L}(x, \lambda) = \begin{bmatrix} \nabla f(x) + J_g^T(x)\lambda \\ g(x) \end{bmatrix} = 0$$

Once again, this is a nonlinear system of equations that can be solved via Newton's method

Sequential Quadratic Programming

To derive the Jacobian of this system, we write

$$\nabla \mathcal{L}(x, \lambda) = \begin{bmatrix} \nabla f(x) + \sum_{k=1}^{m} \lambda_k \nabla g_k(x) \\ g(x) \end{bmatrix} \in \mathbb{R}^{n+m}$$

Then we need to differentiate wrt to $x \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}^m$

For $i = 1, \ldots, n$, we have

$$(\nabla \mathcal{L}(x, \lambda))_i = \frac{\partial f(x)}{\partial x_i} + \sum_{k=1}^{m} \lambda_k \frac{\partial g_k(x)}{\partial x_i}$$

Differentiating wrt x_j, for $i, j = 1, \ldots, n$, gives

$$\frac{\partial}{\partial x_j}(\nabla \mathcal{L}(x, \lambda))_i = \frac{\partial^2 f(x)}{\partial x_i \partial x_j} + \sum_{k=1}^{m} \lambda_k \frac{\partial^2 g_k(x)}{\partial x_i \partial x_j}$$

Sequential Quadratic Programming

Hence the top-left $n \times n$ block of the Jacobian of $\nabla \mathcal{L}(x, \lambda)$ is

$$B(x, \lambda) \equiv H_f(x) + \sum_{k=1}^{m} \lambda_k H_{g_k}(x) \in \mathbb{R}^{n \times n}$$

Differentiating $(\nabla \mathcal{L}(x, \lambda))_i$ wrt λ_j, for $i = 1, \ldots, n$, $j = 1, \ldots, m$, gives

$$\frac{\partial}{\partial \lambda_j} (\nabla \mathcal{L}(x, \lambda))_i = \frac{\partial g_j(x)}{\partial x_i}$$

Hence the top-right $n \times m$ block of the Jacobian of $\nabla \mathcal{L}(x, \lambda)$ is

$$J_g(x)^T \in \mathbb{R}^{n \times m}$$

Sequential Quadratic Programming

For $i = n+1, \ldots, n+m$, we have

$$(\nabla \mathcal{L}(x, \lambda))_i = g_i(x)$$

Differentiating $(\nabla \mathcal{L}(x, \lambda))_i$ wrt x_j, for $i = n+1, \ldots, n+m$, $j = 1, \ldots, n$, gives

$$\frac{\partial}{\partial x_j}(\nabla \mathcal{L}(x, \lambda))_i = \frac{\partial g_i(x)}{\partial x_j}$$

Hence the bottom-left $m \times n$ block of the Jacobian of $\nabla \mathcal{L}(x, \lambda)$ is

$$J_g(x) \in \mathbb{R}^{m \times n}$$

. . . and the final $m \times m$ bottom right block is just zero (differentiation of $g_i(x)$ w.r.t. λ_j)

Sequential Quadratic Programming

Hence, we have derived the following Jacobian matrix for $\nabla \mathcal{L}(x, \lambda)$:

$$\begin{bmatrix} B(x,\lambda) & J_g^T(x) \\ J_g(x) & 0 \end{bmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}$$

Note the 2×2 block structure of this matrix (matrices with this structure are often called KKT matrices[18])

[18] Karush, Kuhn, Tucker: did seminal work on nonlinear optimization

Sequential Quadratic Programming

Therefore, Newton's method for $\nabla \mathcal{L}(x, \lambda) = 0$ is:

$$\begin{bmatrix} B(x_k, \lambda_k) & J_g^T(x_k) \\ J_g(x_k) & 0 \end{bmatrix} \begin{bmatrix} s_k \\ \delta_k \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) + J_g^T(x_k)\lambda_k \\ g(x_k) \end{bmatrix}$$

for $k = 0, 1, 2, \ldots$

Here $(s_k, \delta_k) \in \mathbb{R}^{n+m}$ is the k^{th} Newton step

Sequential Quadratic Programming

Now, consider the constrained minimization problem, where (x_k, λ_k) is our Newton iterate at step k:

$$\min_s \left\{ \frac{1}{2} s^T B(x_k, \lambda_k) s + s^T (\nabla f(x_k) + J_g^T(x_k)\lambda_k) \right\}$$

$$\text{subject to} \quad J_g(x_k)s + g(x_k) = 0$$

The objective function is quadratic in s (here x_k, λ_k are constants)

This minimization problem has Lagrangian

$$\begin{aligned} \mathcal{L}_k(s, \delta) &\equiv \frac{1}{2} s^T B(x_k, \lambda_k) s + s^T (\nabla f(x_k) + J_g^T(x_k)\lambda_k) \\ &+ \delta^T (J_g(x_k)s + g(x_k)) \end{aligned}$$

Sequential Quadratic Programming

Then solving $\nabla \mathcal{L}_k(s, \delta) = 0$ (i.e. first-order necessary conditions) gives a linear system, which is the same as the kth Newton step

Hence at each step of Newton's method, we exactly solve a minimization problem (quadratic objective fn., linear constraints)

An optimization problem of this type is called a quadratic program

This motivates the name for applying Newton's method to $\mathcal{L}(x, \lambda) = 0$: Sequential Quadratic Programming (SQP)

Sequential Quadratic Programming

SQP is an important method, and there are many issues to be considered to obtain an efficient and reliable implementation:

- Efficient solution of the linear systems at each Newton iteration — matrix block structure can be exploited
- Quasi-Newton approximations to the Hessian (as in the unconstrained case)
- Trust region, line search etc to improve robustness
- Treatment of constraints (equality and inequality) during the iterative process
- Selection of good starting guess for λ

Penalty Methods

Another computational strategy for constrained optimization is to employ penalty methods

This converts a constrained problem into an unconstrained problem

Key idea: Introduce a new objective function which is a weighted sum of objective function and constraint

Penalty Methods

Given the minimization problem

$$\min_x f(x) \quad \text{subject to} \quad g(x) = 0$$

we can consider the related unconstrained problem

$$\min_x \phi_\rho(x) = f(x) + \frac{1}{2}\rho g(x)^T g(x) \qquad (**)$$

Let x^* and x_ρ^* denote the solution of $(*)$ and $(**)$, respectively

Under appropriate conditions, it can be shown that

$$\lim_{\rho \to \infty} x_\rho^* = x^*$$

Penalty Methods

In practice, we can solve the unconstrained problem for a large value of ρ to get a good approximation of x^*

Another strategy is to solve for a sequence of penalty parameters, ρ_k, where $x^*_{\rho_k}$ serves as a starting guess for $x^*_{\rho_{k+1}}$

Note that the major drawback of penalty methods is that a large factor ρ will increase the condition number of the Hessian H_{ϕ_ρ}

On the other hand, penalty methods can be convenient, primarily due to their simplicity

Linear Programming

Linear Programming

As we mentioned earlier, the optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \text{ subject to } g(x) = 0 \text{ and } h(x) \leq 0, \qquad (*)$$

with f, g, h affine, is called a linear program

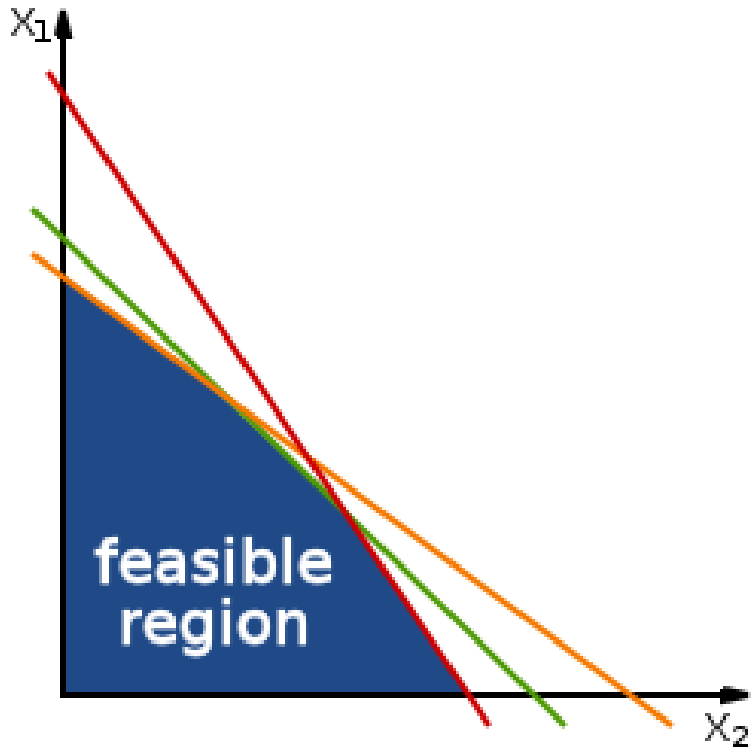
The feasible region is a convex polyhedron[19]

Since the objective function maps out a hyperplane, its global minimum must occur at a vertex of the feasible region

[19] Polyhedron: a solid with flat sides, straight edges

Linear Programming

This can be seen most easily with a picture (in $\mathbb{R}^2$)



Linear Programming

The standard approach for solving linear programs is conceptually simple: examine a sequence of the vertices to find the minimum

This is called the simplex method

Despite its conceptual simplicity, it is non-trivial to develop an efficient implementation of this algorithm

We will not discuss the implementation details of the simplex method . . .

Linear Programming

In the worst case, the computational work required for the simplex method grows exponentially with the size of the problem

But this worst-case behavior is extremely rare; in practice simplex is very efficient (computational work typically grows linearly)

Newer methods, called interior point methods, have been developed that are polynomial in the worst case

Nevertheless, simplex is still the standard approach since it is more efficient than interior point for most problems

Linear Programming

Python example: [*linprog.py*] Using `cvxopt`,[20] solve the linear program

$$\min_x f(x) = -5x_1 - 4x_2 - 6x_3$$

subject to

$$\begin{aligned} x_1 - x_2 + x_3 &\leq 20 \\ 3x_1 + 2x_2 + 4x_3 &\leq 42 \\ 3x_1 + 2x_2 &\leq 30 \end{aligned}$$

and $0 \leq x_1, 0 \leq x_2, 0 \leq x_3$

(LP solvers are efficient, main challenge is to formulate an optimization problem as a linear program in the first place!)

[20] [*linprog_alt.py*] An alternative version using SciPy is also provided

PDE Constrained Optimization

Here we will focus on the form we introduced first:

$$\min_{p \in \mathbb{R}^n} \mathcal{G}(p)$$

Optimization methods usually need some derivative information, such as using finite differences to approximate $\nabla \mathcal{G}(p)$

PDE Constrained Optimization

But using finite differences can be expensive, especially if we have many parameters:

$$\frac{\partial \mathcal{G}(p)}{\partial p_i} \approx \frac{\mathcal{G}(p + he_i) - \mathcal{G}(p)}{h},$$

hence we need $n + 1$ evaluations of $\mathcal{G}$ to approximate $\nabla \mathcal{G}(p)$!

We saw from the Himmelblau example that supplying the gradient $\nabla \mathcal{G}(p)$ cuts down on the number of function evaluations required

The extra function calls due to F.D. isn't a big deal for Himmelblau's function, each evaluation is very cheap

But in PDE constrained optimization, each $p \to \mathcal{G}(p)$ requires a full PDE solve!

PDE Constrained Optimization

Hence for PDE constrained optimization with many parameters, it is important to be able to compute the gradient more efficiently

There are two main approaches:

- the direct method
- the adjoint method

The direct method is simpler, but the adjoint method is much more efficient if we have many parameters

PDE Output Derivatives

Consider the ODE BVP

$$-u''(x;p) + r(x;p)u(x;p) = f(x), \qquad u(a) = u(b) = 0$$

which we will refer to as the primal equation

Here $p \in \mathbb{R}^n$ is the parameter vector, and $r : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$

We define an output functional based on an integral

$$g(u) \equiv \int_a^b \sigma(x)u(x)\mathrm{d}x,$$

for some function σ; then $\mathcal{G}(p) \equiv g(u(p)) \in \mathbb{R}$

The Direct Method

We observe that

$$\frac{\partial \mathcal{G}(p)}{\partial p_i} = \int_a^b \sigma(x) \frac{\partial u}{\partial p_i} \mathrm{d}x$$

hence if we can compute $\frac{\partial u}{\partial p_i}$, $i = 1, 2, \ldots, n$, then we can obtain the gradient

Assuming sufficient smoothness, we can "differentiate the ODE BVP" wrt p_i to obtain,

$$-\frac{\partial u}{\partial p_i}''(x; p) + r(x; p) \frac{\partial u}{\partial p_i}(x; p) = -\frac{\partial r}{\partial p_i} u(x; p)$$

for $i = 1, 2, \ldots, n$

The Direct Method

Once we compute each $\frac{\partial u}{\partial p_i}$ we can then evaluate $\nabla \mathcal{G}(p)$ by evaluating a sequence of n integrals

However, this is not much better than using finite differences: We still need to solve n separate ODE BVPs

(Though only the right-hand side changes, so could LU factorize the system matrix once and back/forward sub. for each i)

Adjoint-Based Method

However, a more efficient approach when n is large is the adjoint method

We introduce the adjoint equation:

$$-z''(x;p) + r(x;p)z(x;p) = \sigma(x), \qquad z(a) = z(b) = 0$$

Adjoint-Based Method

Now,

$$\begin{aligned}
\frac{\partial \mathcal{G}(p)}{\partial p_i} &= \int_a^b \sigma(x) \frac{\partial u}{\partial p_i} \mathrm{d}x \\
&= \int_a^b \left[-z''(x;p) + r(x;p) z(x;p) \right] \frac{\partial u}{\partial p_i} \mathrm{d}x \\
&= \int_a^b z(x;p) \left[-\frac{\partial u}{\partial p_i}''(x;p) + r(x;p) \frac{\partial u}{\partial p_i}(x;p) \right] \mathrm{d}x,
\end{aligned}$$

where the last line follows by integrating by parts twice (boundary terms vanish because $\frac{\partial u}{\partial p_i}$ and z are zero at a and b)

(The adjoint equation is defined based on this "integration by parts" relationship to the primal equation)

Adjoint-Based Method

Also, recalling the derivative of the primal problem with respect to p_i:

$$-\frac{\partial u}{\partial p_i}''(x;p) + r(x;p)\frac{\partial u}{\partial p_i}(x;p) = -\frac{\partial r}{\partial p_i}u(x;p),$$

we get

$$\frac{\partial \mathcal{G}(p)}{\partial p_i} = -\int_a^b \frac{\partial r}{\partial p_i} z(x;p)u(x;p)\mathrm{d}x$$

Therefore, we only need to solve two differential equations (primal and adjoint) to obtain $\nabla\mathcal{G}(p)$! Each component of the gradient requires a single integration.

For more complicated PDEs the adjoint formulation is more complicated but the basic ideas stay the same