Gauss–Newton Method

Note that $\sum_{k=1}^{m} \frac{\partial r_k}{\partial b_j}\frac{\partial r_k}{\partial b_i} = (J_r(b)^T J_r(b))_{ij}$, so that when the residual is small $J_F(b) \approx J_r(b)^T J_r(b)$

Then putting all the pieces together, we obtain the iteration: $b_{k+1} = b_k + \Delta b_k$ where

$$J_r(b_k)^T J_r(b_k)\Delta b_k = -J(b_k)^T r(b_k), \qquad k = 1, 2, 3, \ldots$$

This is known as the Gauss–Newton Algorithm for nonlinear least squares

Gauss–Newton Method

This looks similar to Normal Equations at each iteration, except now the matrix $J_r(b_k)$ comes from linearizing the residual

Gauss–Newton is equivalent to solving the linear least squares problem $J_r(b_k)\Delta b_k \simeq -r(b_k)$ at each iteration

This is a common refrain in Scientific Computing: Replace a nonlinear problem with a sequence of linearized problems

Computing the Jacobian

To use Gauss–Newton in practice, we need to be able to compute the Jacobian matrix $J_r(b_k)$ for any $b_k \in \mathbb{R}^n$

We can do this "by hand", e.g. in our transmitter/receiver problem we would have:

$$[J_r(b)]_{ij} = -\frac{\partial}{\partial b_j}\sqrt{(b_1 - x_1^i)^2 + (b_2 - x_2^i)^2}$$

Differentiating by hand is feasible in this case, but it can become impractical if $r(b)$ is more complicated

Or perhaps our mapping $b \to y$ is a "black box" — no closed form equations hence not possible to differentiate the residual!

Computing the Jacobian

So, what is the alternative to "differentiation by hand"?
Finite difference approximation: for $h \ll 1$ we have

$$[J_r(b_k)]_{ij} \approx \frac{r_i(b_k + e_j h) - r_i(b_k)}{h}$$

Avoids tedious, error prone differentiation of r by hand!

Also, can be used for differentiating "black box" mappings since we only need to be able to evaluate $r(b)$

Gauss–Newton Method

We derived the Gauss–Newton algorithm method in a natural way:

- apply Newton's method to solve $\nabla\phi = 0$
- neglect the second derivative terms that arise

However, Gauss–Newton is not widely used in practice since it doesn't always converge reliably

Levenberg–Marquardt Method

A more robust variation of Gauss–Newton is the Levenberg–Marquardt Algorithm, which uses the update

$$[J^T(b_k)J(b_k) + \mu_k \operatorname{diag}(S^T S)]\Delta b = -J(b_k)^T r(b_k)$$

where[1] $S = \mathrm{I}$ or $S = J(b_k)$, and some heuristic is used to choose μ_k

This looks like our "regularized" underdetermined linear least squares formulation!

[1]In this context diag(A) means "zero the off-diagonal part of A"

Levenberg–Marquardt Method

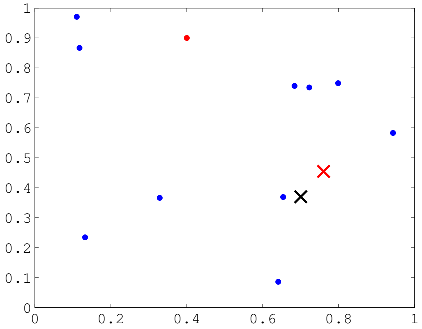
Key point: The regularization term $\mu_k \text{diag}(S^T S)$ improves the reliability of the algorithm in practice

Levenberg–Marquardt is implemented in Python and Matlab's optimization toolbox
We need to pass the residual to the routine, and we can also pass the Jacobian matrix or ask for a finite-differenced Jacobian

Now let's solve our transmitter/receiver problem

Nonlinear Least Squares: Example

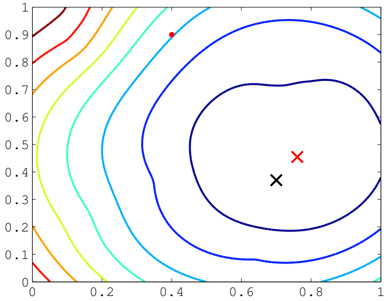
Python example: Using `lsqnonlin.py` in Matlab we provide an initial guess (●), and converge to the solution (×)



Nonlinear Least Squares: Example

Levenberg–Marquardt minimizes $\phi(b)$, as we see from the contour plot of $\phi(b)$ below

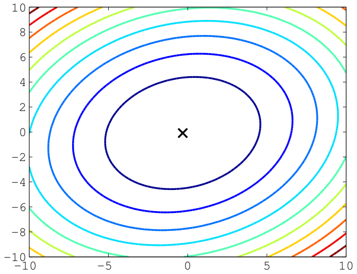
Recall $\times$ is the true transmitter location, $\times$ is our best-fit to the data; $\phi(\times) = 0.0248 < 0.0386 = \phi(\times)$.

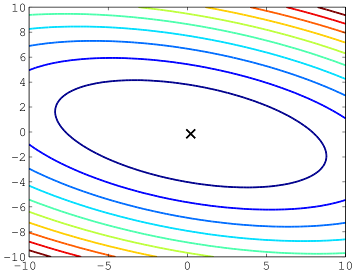


These contours are quite different from what we get in linear problems

Linear Least-Squares Contours

Two examples of linear least squares contours for $\phi(b) = \|y - Ab\|_2^2$, $b \in \mathbb{R}^2$





In linear least squares $\phi(b)$ is quadratic, hence contours are "hyperellipses"

Unit II: Numerical Linear Algebra

Motivation

Almost everything in Scientific Computing relies on Numerical Linear Algebra!

We often reformulate problems as $Ax = b$, e.g. from Unit I:

- Interpolation (Vandermonde matrix) and linear least-squares (normal equations) are naturally expressed as linear systems
- Gauss-Newton/Levenberg-Marquardt involve approximating nonlinear problem by a sequence of linear systems

Similar themes will arise in remaining Units (Numerical Calculus, Optimization, Eigenvalue problems)

Motivation

The goal of this Unit is to cover:

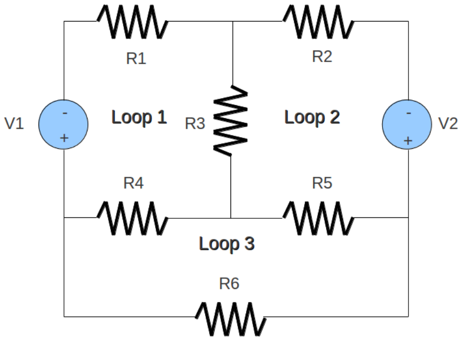
- key linear algebra concepts that underpin Scientific Computing
- algorithms for solving $Ax = b$ in a stable and efficient manner
- algorithms for computing factorizations of A that are useful in many practical contexts (LU, QR)

First, we discuss some practical cases where $Ax = b$ arises directly in mathematical modeling of physical systems

Example: Electric Circuits

Ohm's Law: Voltage drop due to a current i through a resistor R is $V = iR$

Kirchoff's Law: The net voltage drop in a closed loop is zero



Example: Electric Circuits

Let i_j denote the current in "loop j"

Then, we obtain the linear system:

$$\begin{bmatrix} (R_1 + R_3 + R_4) & R_3 & R_4 \\ R_3 & (R_2 + R_3 + R_5) & -R_5 \\ R_4 & -R_5 & (R_4 + R_5 + R_6) \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ 0 \end{bmatrix}$$

Circuit simulators solve large linear systems of this type

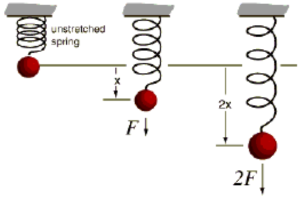
Example: Structural Analysis

Common in structural analysis to use a linear relationship between force and displacement, Hooke's Law

Simplest case is the Hookean spring law

$$F = kx,$$

- k: spring constant (stiffness)
- F: applied load
- x: spring extension



Example: Structural Analysis

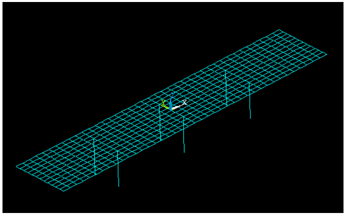
This relationship can be generalized to structural systems in 2D and 3D, which yields a linear system of the form

$$Kx = F$$

- $K \in \mathbb{R}^{n \times n}$: "stiffness matrix"
- $F \in \mathbb{R}^{n}$: "load vector"
- $x \in \mathbb{R}^{n}$: "displacement vector"

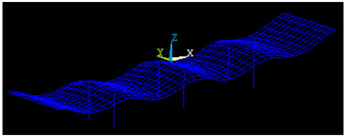
Example: Structural Analysis

Solving the linear system yields the displacement (x), hence we can simulate structural deflection under applied loads (F)



Unloaded structure

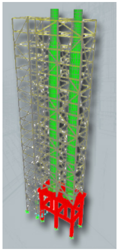
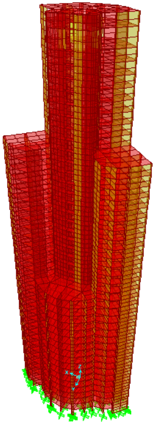
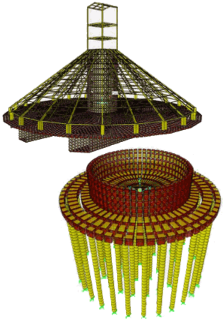
$\xrightarrow{Kx=F}$



Loaded structure

Example: Structural Analysis

It is common engineering practice to use Hooke's Law to simulate complex structures, which leads to large linear systems

(From SAP2000, structural analysis software)

Example: Economics

Leontief awarded Nobel Prize in Economics in 1973 for developing linear input/output model for production/consumption of goods

Consider an economy in which n goods are produced and consumed

- $A \in \mathbb{R}^{n \times n}$: a_{ij} represents amount of good j required to produce 1 unit of good i
- $x \in \mathbb{R}^n$: x_i is number of units of good i produced
- $d \in \mathbb{R}^n$: d_i is consumer demand for good i

In general $a_{ii} = 0$, and A may or may not be sparse

Example: Economics

The total amount of x_i produced is given by the sum of consumer demand (d_i) and the amount of x_i required to produce each x_j

$$x_i = \underbrace{a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n}_{\text{production of other goods}} + d_i$$

Hence $x = Ax + d$ or,

$$(\mathrm{I} - A)x = d$$

Solve for x to determine the required amount of production of each good

If we consider many goods (*e.g.* an entire economy), then we get a large linear system

Summary

Matrix computations arise all over the place!

Numerical Linear Algebra algorithms provide us with a toolbox for performing these computations in an efficient and stable manner

In most cases, can use these tools as black boxes, but it's important to understand what the linear algebra black boxes do:

- Pick the right algorithm for a given situation (*e.g.* exploit structure in a problem: symmetry, bandedness, *etc.*)
- Understand how and when the black box can fail

Preliminaries

In this chapter we will focus on linear systems $Ax = b$ for $A \in \mathbb{R}^{n \times n}$ and $b, x \in \mathbb{R}^n$

Recall that it is often helpful to think of matrix multiplication as a linear combination of the columns of A, where x_j are the weights

That is, we have $b = Ax = \sum_{j=1}^{n} x_j a_{(:,j)}$ where $a_{(:,j)} \in \mathbb{R}^n$ is the j^{th} column of A and x_j are scalars

Preliminaries

This can be displayed schematically as

$$
\begin{aligned}
\begin{bmatrix} \\ b \\ \\ \end{bmatrix} &= \left[\begin{array}{c|c|c|c} & & & \\ a_{(:,1)} & a_{(:,2)} & \cdots & a_{(:,n)} \\ & & & \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\
&= x_1 \begin{bmatrix} \\ a_{(:,1)} \\ \\ \end{bmatrix} + \cdots + x_n \begin{bmatrix} \\ a_{(:,n)} \\ \\ \end{bmatrix}
\end{aligned}
$$

Preliminaries

We therefore interpret $Ax = b$ as: "x is the vector of coefficients of the linear expansion of b in the basis of columns of A"

Often this is a more helpful point of view than conventional interpretation of "dot-product of matrix row with vector"

e.g. from "linear combination of columns" view we immediately see that $Ax = b$ has a solution if

$$b \in \mathrm{span}\{a_{(:,1)}, a_{(:,2)}, \cdots, a_{(:,n)}\}$$

(this holds even if A isn't square)

Let us write image$(A) \equiv$ span$\{a_{(:,1)}, a_{(:,2)}, \cdots, a_{(:,n)}\}$

Preliminaries

Existence and Uniqueness:

Solution $x \in \mathbb{R}^n$ exists if $b \in \text{image}(A)$

If solution x exists and the set $\{a_{(:,1)}, a_{(:,2)}, \cdots, a_{(:,n)}\}$ is linearly independent, then x is unique[2]

If solution x exists and $\exists z \neq 0$ such that $Az = 0$, then also $A(x + \gamma z) = b$ for any $\gamma \in \mathbb{R}$, hence infinitely many solutions

If $b \notin \text{image}(A)$ then $Ax = b$ has no solution

[2] Linear independence of columns of A is equivalent to $Az = 0 \implies z = 0$

Preliminaries

The inverse map $A^{-1} \colon \mathbb{R}^n \to \mathbb{R}^n$ is well-defined if and only if $Ax = b$ has unique solution for all $b \in \mathbb{R}^n$

Unique matrix $A^{-1} \in \mathbb{R}^{n\times n}$ such that $AA^{-1} = A^{-1}A = I$ exists if any of the following equivalent conditions are satisfied:

- $\det(A) \neq 0$
- $\operatorname{rank}(A) = n$
- For any $z \neq 0$, $Az \neq 0$ (null space of A is $\{0\}$)

A is non-singular if A^{-1} exists, and then $x = A^{-1}b \in \mathbb{R}^n$

A is singular if A^{-1} does not exist

Norms

A norm $\|\cdot\| : V \to \mathbb{R}$ is a function on a vector space V that satisfies

- $\|x\| \geq 0$ and $\|x\| = 0 \implies x = 0$
- $\|\gamma x\| = |\gamma|\|x\|$, for $\gamma \in \mathbb{R}$
- $\|x + y\| \leq \|x\| + \|y\|$

Norms

Also, the triangle inequality implies another helpful inequality: the "reverse triangle inequality", $|\|x\| - \|y\|| \leq \|x - y\|$

Proof: Let $a = y$, $b = x - y$, then

$$\|x\| = \|a+b\| \leq \|a\|+\|b\| = \|y\|+\|x-y\| \implies \|x\|-\|y\| \leq \|x-y\|$$

Repeat with $a = x$, $b = y - x$ to show $|\|x\| - \|y\|| \leq \|x - y\|$

Vector Norms

Let's now introduce some common norms on $\mathbb{R}^n$

Most common norm is the Euclidean norm (or 2-norm):

$$\|x\|_2 \equiv \sqrt{\sum_{j=1}^{n} x_j^2}$$

2-norm is special case of the p-norm for any $p \geq 1$:

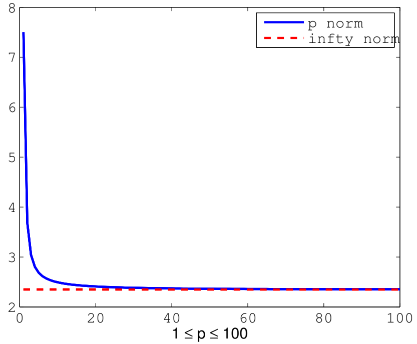
$$\|x\|_p \equiv \left(\sum_{j=1}^{n} |x_j|^p \right)^{1/p}$$

Also, limiting case as $p \to \infty$ is the ∞-norm:

$$\|x\|_\infty \equiv \max_{1 \leq i \leq n} |x_i|$$

Vector Norms

$\|x\|_\infty = 2.35$, we see that p-norm approaches ∞-norm: picks out the largest entry in x



Vector Norms

We generally use whichever norm is most convenient/appropriate for a given problem, *e.g.* 2-norm for least-squares analysis

Different norms give different (but related) measures of size

In particular, an important mathematical fact is:

All norms on a finite dimensional space (such as $\mathbb{R}^n$) are equivalent

Vector Norms

That is, let $\| \cdot \|_a$ and $\| \cdot \|_b$ be two norms on a finite dimensional space V, then $\exists c_1, c_2 \in \mathbb{R}_{>0}$ such that for any $x \in V$

$$c_1 \|x\|_a \leq \|x\|_b \leq c_2 \|x\|_a$$

(Also, from above we have $\frac{1}{c_2}\|x\|_b \leq \|x\|_a \leq \frac{1}{c_1}\|x\|_b$)

Hence if we can derive an inequality in an arbitrary norm on V, it applies (after appropriate scaling) in any other norm too

Vector Norms

In some cases we can explicitly calculate values for c_1, c_2:

e.g. $\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2$, since

$$\|x\|_2^2 = \left(\sum_{j=1}^{n} |x_j|^2\right) \leq \left(\sum_{j=1}^{n} |x_j|\right)^2 = \|x\|_1^2 \implies \|x\|_2 \leq \|x\|_1$$

[e.g. consider $|a|^2 + |b|^2 \leq |a|^2 + |b|^2 + 2|a||b| = (|a| + |b|)^2$]

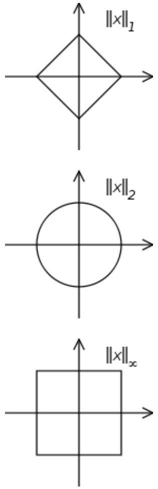
$$\|x\|_1 = \sum_{j=1}^{n} 1 \times |x_j| \leq \left(\sum_{j=1}^{n} 1^2\right)^{1/2} \left(\sum_{j=1} |x_j|^2\right)^{1/2} = \sqrt{n}\,\|x\|_2$$

[We used Cauchy-Schwarz inequality in $\mathbb{R}^n$:
$\sum_{j=1}^{n} a_j b_j \leq (\sum_{j=1}^{n} a_j^2)^{1/2} (\sum_{j=1}^{n} b_j^2)^{1/2}$]

Vector Norms

Different norms give different measurements of size

The "unit circle" in three different norms: $\{x \in \mathbb{R}^2 : \|x\|_p = 1\}$ for $p = 1, 2, \infty$



Matrix Norms

There are many ways to define norms on matrices

For example, the Frobenius norm is defined as:

$$\|A\|_F \equiv \left(\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2}$$

(If we think of A as a vector in $\mathbb{R}^{n^2}$, then Frobenius is equivalent to the vector 2-norm of A)

Matrix Norms

Usually the matrix norms induced by vector norms are most useful, e.g.:

$$\|A\|_p \equiv \max_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p=1} \|Ax\|_p$$

This definition implies the useful property $\|Ax\|_p \leq \|A\|_p \|x\|_p$, since

$$\|Ax\|_p = \frac{\|Ax\|_p}{\|x\|_p} \|x\|_p \leq \left(\max_{v\neq 0} \frac{\|Av\|_p}{\|v\|_p} \right) \|x\|_p = \|A\|_p \|x\|_p$$

Matrix Norms

The 1-norm and ∞-norm can be calculated straightforwardly:

$$\begin{aligned} \|A\|_1 &= \max_{1\le j\le n} \|a_{(:,j)}\|_1 && \text{(max column sum)} \\ \|A\|_\infty &= \max_{1\le i\le n} \|a_{(i,:)}\|_1 && \text{(max row sum)} \end{aligned}$$

We will see how to compute the matrix 2-norm next chapter

Condition Number

Recall from Unit 0 that the condition number of $A \in \mathbb{R}^{n\times n}$ is defined as

$$\kappa(A) \equiv \|A\|\|A^{-1}\|$$

The value of $\kappa(A)$ depends on which norm we use

Both Python and Matlab can calculate the condition number for different norms

If A is square then by convention A singular $\implies \kappa(A) = \infty$

The Residual

Recall that the residual $r(x) = b - Ax$ was crucial in least-squares problems

It is also crucial in assessing the accuracy of a proposed solution $(\hat{x})$ to a square linear system $Ax = b$

Key point: The residual $r(\hat{x})$ is always computable, whereas in general the error $\Delta x \equiv x - \hat{x}$ isn't

The Residual

We have that $\|\Delta x\| = \|x - \hat{x}\| = 0$ if and only if $\|r(\hat{x})\| = 0$

However, small residual doesn't necessarily imply small $\|\Delta x\|$

Observe that

$$\|\Delta x\| = \|x - \hat{x}\| = \|A^{-1}(b - A\hat{x})\| = \|A^{-1}r(\hat{x})\| \leq \|A^{-1}\|\|r(\hat{x})\|$$

Hence

$$\frac{\|\Delta x\|}{\|\hat{x}\|} \leq \frac{\|A^{-1}\|\|r(\hat{x})\|}{\|\hat{x}\|} = \frac{\|A\|\|A^{-1}\|\|r(\hat{x})\|}{\|A\|\|\hat{x}\|} = \kappa(A)\frac{\|r(\hat{x})\|}{\|A\|\|\hat{x}\|} \quad (*)$$

The Residual

Define the relative residual as $\|r(\hat{x})\|/(\|A\|\|\hat{x}\|)$

Then our inequality states that “relative error is bounded by condition number times relative residual”

This is just like our condition number relationship from Unit 0:

$$\kappa(A) \geq \frac{\|\Delta x\|/\|x\|}{\|\Delta b\|/\|b\|}, \qquad \textit{i.e.} \qquad \frac{\|\Delta x\|}{\|x\|} \leq \kappa(A)\frac{\|\Delta b\|}{\|b\|} \quad (**)$$

The reason $(*)$ and $(**)$ are related is that the residual measures the “input pertubation” in $Ax = b$

To see this, let’s consider $Ax = b$ to be a map $b \in \mathbb{R}^n \to x \in \mathbb{R}^n$

The Residual

Then we can consider $\hat{x}$ to be the exact solution for some perturbed input $\hat{b} = b + \Delta b$, *i.e.* $A\hat{x} = \hat{b}$

The residual associated with $\hat{x}$ is $r(\hat{x}) = b - A\hat{x} = b - \hat{b} = -\Delta b$, i.e. $\|r(\hat{x})\| = \|\Delta b\|$

In general, a numerically stable algorithm gives us the exact solution to a slightly perturbed problem, i.e. a small residual[3]

This is a reasonable expectation for a stable algorithm: rounding error doesn't accumulate, so effective input perturbation is small

[3]More precisely, this is called a "backward stable algorithm"