

Lecture 12

Solving Linear Systems of Equations

L. Olson

Department of Computer Science
University of Illinois at Urbana-Champaign

Slides based on slides from Numerical Methods with Matlab by Gerald Recktenwald

March 9, 2006

Today:

Objectives

- to motivate where we're going: interpolation
- build a tool that we'll need: conditioning
- wrap up old stuff: LU factorization

Material

- First few pages of chapter 10
- wrap-up sections 8.3 and 8.4

Interpolation?

Example

Given a set of data

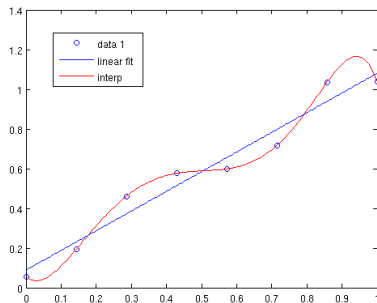
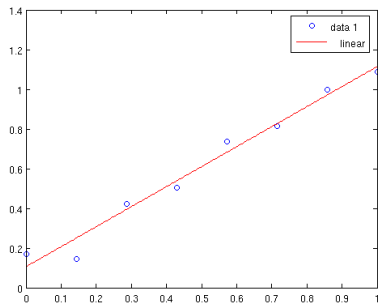
$$y_i \text{ at } x_i \quad i = 1 \dots n$$

find a function $\mathcal{F}(x)$ that represents this data

We have two options:

- 1 to get close to having $y_i = \mathcal{F}(x_i)$ for each i (fitting)
- 2 exactly pass through y_i with $\mathcal{F}(x_i)$ for each i (interpolation)

Interpolation versus Data Fitting?



Interpolation

Obvious attmps: try picking

$$\mathcal{F}(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1}$$

So for each x_i we have

$$\mathcal{F}(x_i) = c_0 + c_1x_i + c_2x_i^2 + \cdots + c_{n-1}x_i^{n-1} = y_i$$

OR

$$c_0 + c_1x_1 + c_2x_1^2 + \cdots + c_{n-1}x_1^{n-1} = y_1$$

$$c_0 + c_1x_2 + c_2x_2^2 + \cdots + c_{n-1}x_2^{n-1} = y_2$$

$$c_0 + c_1x_3 + c_2x_3^2 + \cdots + c_{n-1}x_3^{n-1} = y_3$$

\vdots

$$c_0 + c_1x_n + c_2x_n^2 + \cdots + c_{n-1}x_n^{n-1} = y_n$$

Interpolation: Vandermonde

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ & & & \vdots & \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Questions

- Can we always solve this system?
- Is it likely to have round-off? Lots of it?
- Do we get unique solutions?

Interpolation: Vandermonde

Take n large and look at the last two columns:

$$\begin{bmatrix} x_1^{n-2} \\ x_2^{n-2} \\ \vdots \\ x_n^{n-2} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} x_1^{n-1} \\ x_2^{n-1} \\ \vdots \\ x_n^{n-1} \end{bmatrix}$$

!!!

Nearly linearly dependent. How bad?

Limits on Numerical Solution to $Ax = b$

Limits of Floating Point Arithmetic

- Exact singularity
- Effect of perturbations to b
- Effect of perturbations to A
- The condition number

Geometric Interpretation of Singularity (1)

Consider a 2×2 system describing two lines that intersect

$$y = -2x + 6$$

$$y = \frac{1}{2}x + 1$$

The matrix form of this equation is

$$\begin{bmatrix} 2 & 1 \\ -1/2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \end{bmatrix}$$

The equations for two **parallel** but **not intersecting** lines are

$$\begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \end{bmatrix}$$

Here the coefficient matrix is singular ($\text{rank}(A) = 1$), and the system is inconsistent

Geometric Interpretation of Singularity (2)

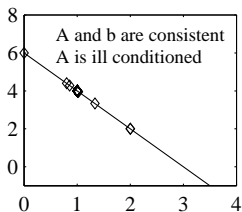
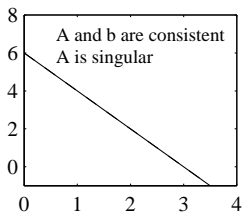
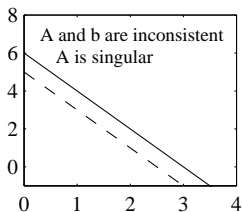
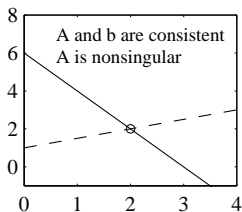
The equations for two **parallel** and **coincident** lines are

$$\begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

The equations for two **nearly parallel** lines are

$$\begin{bmatrix} 2 & 1 \\ 2 + \delta & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 + \delta \end{bmatrix}$$

Geometric Interpretation of Singularity (3)



Effect of Perturbations to b

Consider the solution of a 2×2 system where

$$b = \begin{bmatrix} 1 \\ 2/3 \end{bmatrix}$$

One expects that the *exact* solutions to

$$Ax = \begin{bmatrix} 1 \\ 2/3 \end{bmatrix} \quad \text{and} \quad Ax = \begin{bmatrix} 1 \\ 0.6667 \end{bmatrix}$$

will be different. Should these solutions be a **lot different** or a **little different**?

Effect of Perturbations to b

Perturb b with δb such that

$$\frac{\|\delta b\|}{\|b\|} \ll 1,$$

The perturbed system is

$$A(x + \delta x_b) = b + \delta b$$

The perturbations satisfy

$$A\delta x_b = \delta b$$

Analysis shows (see next two slides for proof) that

$$\frac{\|\delta x_b\|}{\|x\|} \leq \|A\| \|A^{-1}\| \frac{\|\delta b\|}{\|b\|}$$

Thus, the effect of the perturbation is small *only if* $\|A\| \|A^{-1}\|$ is small.

$$\frac{\|\delta x_b\|}{\|x\|} \ll 1 \quad \text{only if} \quad \|A\| \|A^{-1}\| \sim 1$$

Effect of Perturbations to A

Perturb A with δA such that

$$\frac{\|\delta A\|}{\|A\|} \ll 1,$$

The perturbed system is

$$(A + \delta A)(x + \delta x_A) = b$$

Analysis shows that

$$\frac{\|\delta x_A\|}{\|x + \delta x_A\|} \leq \|A\| \|A^{-1}\| \frac{\|\delta A\|}{\|A\|}$$

Thus, the effect of the perturbation is small *only if* $\|A\| \|A^{-1}\|$ is small.

$$\frac{\|\delta x_A\|}{\|x + \delta x_A\|} \ll 1 \quad \text{only if} \quad \|A\| \|A^{-1}\| \sim 1$$

Effect of Perturbations to both A and b

Perturb both A with δA and b with δb such that

$$\frac{\|\delta A\|}{\|A\|} \ll 1 \quad \text{and} \quad \frac{\|\delta b\|}{\|b\|} \ll 1$$

The perturbation satisfies

$$(A + \delta A)(x + \delta x) = b + \delta b$$

Analysis shows that

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leq \frac{\|A\| \|A^{-1}\|}{1 - \|A\| \|A^{-1}\| \frac{\|\delta A\|}{\|A\|}} \left[\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right]$$

Thus, the effect of the perturbation is small *only if* $\|A\| \|A^{-1}\|$ is small.

$$\frac{\|\delta x\|}{\|x + \delta x\|} \ll 1 \quad \text{only if} \quad \|A\| \|A^{-1}\| \sim 1$$

Condition number of A

The **condition number**

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

indicates the sensitivity of the solution to perturbations in A and b . The condition number can be measured with any p -norm.

The condition number is always in the range

$$1 \leq \kappa(A) \leq \infty$$

- $\kappa(A)$ is a mathematical property of A
- Any algorithm will produce a solution that is sensitive to perturbations in A and b if $\kappa(A)$ is large.
- In exact math a matrix is either singular or non-singular.
 $\kappa(A) = \infty$ for a singular matrix
- $\kappa(A)$ indicates how close A is to being numerically singular.
- A matrix with large κ is said to be **ill-conditioned**

Computational Stability

In Practice, applying Gaussian elimination with partial pivoting and back substitution to $Ax = b$ gives the **exact solution**, \hat{x} , to the **nearby problem**

$$(A + E)\hat{x} = b \quad \text{where} \quad \|E\|_{\infty} \leq \varepsilon_m \|A\|_{\infty}$$

*Gaussian elimination with partial pivoting and back substitution
“gives exactly the right answer to nearly the right question.”*

— Trefethen and Bau

Computational Stability

- An algorithm that gives the exact answer to a problem that is near to the original problem is said to be **backward stable**.
- Algorithms that are not backward stable will tend to amplify roundoff errors present in the original data.
- As a result, the solution produced by an algorithm that is not backward stable will not necessarily be the solution to a problem that is close to the original problem.
- Gaussian elimination without partial pivoting is *not* backward stable for arbitrary A . If A is symmetric and positive definite, then Gaussian elimination without pivoting is backward stable.

The Residual

Let \hat{x} be the numerical solution to $Ax = b$. $\hat{x} \neq x$ (x is the exact solution) because of roundoff.

The **residual** measures how close \hat{x} is to satisfying the original equation

$$r = b - A\hat{x}$$

It is not hard to show that

$$\frac{\|\hat{x} - x\|}{\|\hat{x}\|} \leq \kappa(A) \frac{\|r\|}{\|b\|}$$

Small $\|r\|$ does not guarantee a small $\|\hat{x} - x\|$.

If $\kappa(A)$ is large the \hat{x} returned by Gaussian elimination and back substitution (or any other solution method) is not guaranteed to be anywhere near the true solution to $Ax = b$.

Rules of Thumb (1)

- Applying Gaussian elimination with partial pivoting and back substitution to $Ax = b$ yields a numerical solution \hat{x} such that the residual vector $r = b - A\hat{x}$ is small *even if* the $\kappa(A)$ is large.
- If A and b are stored to machine precision ϵ_m , the numerical solution to $Ax = b$ by any variant of Gaussian elimination is correct to d digits where

$$d = |\log_{10}(\epsilon_m)| - \log_{10}(\kappa(A))$$

Rules of Thumb (2)

$$d = |\log_{10}(\varepsilon_m)| - \log_{10}(\kappa(A))$$

Example:

MATLAB computations have $\varepsilon_m \approx 2.2 \times 10^{-16}$. For a system with $\kappa(A) \sim 10^{10}$ the elements of the solution vector will have

$$\begin{aligned}d &= |\log_{10}(2.2 \times 10^{-16})| - \log_{10}(10^{10}) \\ &= 16 - 10 \\ &= 6\end{aligned}$$

correct digits

Summary of Limits to Numerical Solution of $Ax = b$

- 1 $\kappa(A)$ indicates how close A is to being numerically singular
- 2 If $\kappa(A)$ is “large”, A is **ill-conditioned** and *even the best* numerical algorithms will produce a solution, \hat{x} that cannot be guaranteed to be close to the true solution, x
- 3 In practice, Gaussian elimination with partial pivoting and back substitution produces a solution with a small residual

$$r = b - A\hat{x}$$

even if $\kappa(A)$ is large.

Factorization Methods

- LU factorization
- Cholesky factorization
- Use of the backslash operator

LU Factorization (1)

Find L and U such that

$$A = LU$$

and L is lower triangular, and U is upper triangular.

$$L = \begin{bmatrix} 1 & 0 & \cdots & & 0 \\ \ell_{2,1} & 1 & 0 & & 0 \\ \ell_{3,1} & \ell_{3,2} & 1 & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ \ell_{n,1} & \ell_{n,2} & \cdots & \ell_{n-1,n} & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & & & u_{n-1,n} \\ 0 & 0 & & & u_{n,n} \end{bmatrix}$$

Since L and U are triangular, it is easy to apply their inverses.

LU Factorization (2)

Since L and U are triangular, it is easy to apply their inverses.
Consider the solution to $Ax = b$.

$$A = LU \implies (LU)x = b$$

Regroup, matrix multiplication is associative

$$L(Ux) = b$$

Let $Ux = y$, then

$$Ly = b$$

Since L is triangular it is easy (without Gaussian elimination) to compute

$$y = L^{-1}b$$

This expression should be interpreted as “Solve $Ly = b$ with a forward substitution.”

LU Factorization (3)

Now, since y is known, solve for x

$$x = U^{-1}y$$

which is interpreted as “Solve $Ux = y$ with a backward substitution.”

LU Factorization (4)

Listing 1: Solve Ax

```
1 Factor  $A$  into  $L$  and  $U$   
2 Solve  $Ly = b$  for  $y$            use forward substitution  
3 Solve  $Ux = y$  for  $x$            use backward substitution
```

The Built-in `lu` Function

- Refer to `luNopiv` and `luPiv` functions in the NMM toolbox for expository implementations of LU factorization
- Use the built-in `lu` function for routine work

Cholesky Factorization (1)

- A must be symmetric and positive definite (SPD)
- For SPD matrices, pivoting is not required
- Cholesky factorization requires one half as many flops as LU factorization. Since pivoting is not required, Cholesky factorization will be more than twice as fast as LU factorization since data movement is avoided.
- Refer to the `Cholesky` function in NMM Toolbox for a view of the algorithm
- Use built-in `chol` function for routine work

Backslash Redux

The `\` operator examines the coefficient matrix before attempting to solve the system.

`\` uses:

- A triangular solve if A is triangular, or a permutation of a triangular matrix
- Cholesky factorization and triangular solves if A is symmetric and the diagonal elements of A are positive (*and* if the subsequent Cholesky factorization does not fail.)
- LU factorization if A is square and the preceding conditions are not met.
- QR factorization to obtain the least squares solution if A is not square.