AMSC/CMSC 460 Computational Methods, Fall 2007
UNIT 4: Part 2: Linear Systems of Equations

Dianne P. O'Leary

Solution of Linear Systems

Read: Chapter 6. Skip: 6.2.

The plan

- Motivation
- Strategy
- Algorithms
- Analysis

Motivation

When are linear systems used?

Notation:

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

$$\mathbf{A} : n \times n, \qquad \mathbf{x} : n \times 1, \qquad \mathbf{b} : n \times 1.$$

Elements real or complex numbers.

The problem is to find $\mathbf{x}$, given $\mathbf{A}$ and $\mathbf{b}$.

Why study this problem?

- Mathematical models frequently yield linear systems.
- Linear systems are subproblems in
  - determining weights in numerical integration.
  - solving nonlinear equations (See Chapter 8.).
  - solving partial differential equations.
  - etc.

One large government lab reported that 75% of the calls to numerical software libraries were calls to linear system solvers!

Clearly, it is important to be able to solve linear systems quickly and accurately!

Strategy

- Exploit the structure of the problem.

  Example: If $\mathbf{A}$ has lots of zeros, we don't want to replace them by nonzeros. []
- Reduce the problem to one that has an obvious solution.
- Choose algorithms that have reasonable costs.
- Be able to evaluate the quality of our answer.

One algorithm: Cramer's rule

Suppose that we have 2 equations and 2 variables.

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}, \quad x_2 = \frac{\begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}$$

If we apply this algorithm to a problem with $n = 20$, it would take about 3×10^8 years to solve it on a computer.

Problems with $n =$ several million are important in airline scheduling, stock market analysis, weather prediction, power plant safety, traffic analysis, etc.

And Google searches make use of a matrix with size $n \approx 3.5$ billion.

Perhaps we should look for another algorithm....

A second algorithm: use the matrix inverse

We can solve

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

by multiplying both sides of the equation by $\mathbf{A}^{-1}$:

$$\mathbf{A}^{-1}\mathbf{A}\mathbf{x} = \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}\,.$$

Therefore, we can solve linear systems by multiplying the right-hand side $\mathbf{b}$ by $\mathbf{A}^{-1}$.

This is a BAD idea. It is more expensive than the algorithms we will discuss (the LU factorization) and it generally computes an answer that has larger error.

Whenever you see a matrix inverse in a formula, think "LU factorization".

Computing the inverse of a matrix is almost ALWAYS a bad idea!!!

What problems are easy?

2 examples.

Example 1: Suppose $\mathbf{A}$ is diagonal.

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix}$$

Then

$$\mathbf{x} = \begin{bmatrix} 5/2 \\ 6/3 \\ 7/5 \end{bmatrix}$$

[]

Example 2: Suppose $\mathbf{A}$ is triangular

$$\begin{bmatrix} 2 & 0 & 0 \\ 1 & 3 & 0 \\ 2 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ -1 \end{bmatrix}$$

Then the first equation tells us that $2x_1 = 2$, so

$$x_1 = 1\,.$$

The second equation says

$$x_1 + 3x_2 = 7\,.$$

Knowing that $x_1 = 1$, we see that $3x_2 = 7 - 1$, so

$$x_2 = 2\,.$$

Now that we know x_1 and x_2, the third equation tells us that

$$5x_3 = -1 - 2x_1 - x_2\,,$$

so

$$x_3 = -1\,.$$

So our solution is

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}$$

[]

Unquiz: Write two versions of the forward substitution algorithm for solving a lower triangular system $\mathbf{Lx} = \mathbf{b}$.

- In the first, use `sdot`.
- In the second, use `saxpy`.

Note that it is just as easy to solve upper triangular systems

$$\mathbf{A} = \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix}$$

Algorithms

The LU factorization and Gauss Elimination

If we could always reduce our problem to triangular form, we would be finished!

Our tool for this reduction is Gauss elimination and the related *LU* factorization.

See powerpoint example.

Can we always reduce a matrix to triangular form?

Example:

$$\begin{bmatrix} 0 & 4 & 4 \\ 4 & 0 & 2 \\ 2 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 0 \end{bmatrix}$$

with solution $\mathbf{x} = [1, 2, -1]^T$.

Unquiz: Apply Gauss Elimination to this problem.

[]

A complication

We see that Gauss elimination breaks down if the pivot element (the main diagonal element a_{jj} at stage j is zero.

What happens if it is nearly zero?

Example:

$$\begin{bmatrix} 10^{-6} & 4 & 4 \\ 4 & 0 & 2 \\ 2 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 + 10^{-6} \\ 2 \\ 0 \end{bmatrix}$$

with solution $\mathbf{x} = [1, 2, -1]^T$.

Unquiz: Apply Gauss Elimination to this problem.

The fix

We have trouble with Gauss Elimination if the pivot element is close to zero.

How can we guarantee that this never happens?

Note that we have one extra tool available to us: if we interchange two rows of the problem, we don't change the answer:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix} \leftrightarrow \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \end{bmatrix}$$

So, to keep out of trouble, we choose the pivot element at stage j to be the largest magnitude element among those remaining in column j on or below the main diagonal.

This algorithm is called Gauss Elimination with (column) pivoting.

Unquiz: Try it on our example.

$$\begin{bmatrix} 0 & 4 & 4 \\ 4 & 0 & 2 \\ 2 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 0 \end{bmatrix}$$

with solution $\mathbf{x} = [1, 2, -1]^T$.

Implementation

We can implement this algorithm by

- overwriting $\mathbf{A}$ with L and U.
- keeping track of the pivot interchanges in an index array.

Some things to know about Gauss Elimination

- Operations count: $n^3/3$ multiplications.
- Matlab's backslash operator solves linear systems, using LU, without forming the inverse:

  ```
  x = A \ b;
  ```

- Matlab also has an `lu` function that returns the $\mathbf{L}$ and $\mathbf{U}$ factors.

  ```
  [L,U] = lu(A);
  y = L \ b;
  x = U \ y;
  ```

  The matrix `U` is upper triangular, and `L` is a permutation of a lower triangular matrix.

- If you have k right-hand sides involving the same matrix, store them as columns in a matrix $\mathbf{B}$ of size $n \times k$ and then solve using, for example

  ```
  X = A \ B;
  ```

  or

  ```
  [L,U] = lu(A);
  for i=1:k,
     y = L \ B(:,i);
     X(:,i) = U \ y;
  end
  ```

What about sparsity?

If $\mathbf{A}$ has lots of zeros, we would like our algorithms to take advantage of this, and not to ruin the structure by introducing many nonzeros.

If $\mathbf{A}$ is initialized as a sparse matrix in Matlab, then backslash and the `lu` algorithm both try to preserve sparsity.

How good are our answers?

Suppose we solve a linear system on the computer. How do we measure how good our answer is?

Due to round-off error, we probably do not compute the exact answer $\mathbf{x}_{true}$. We would like our computed $\mathbf{x}$ to be close to $\mathbf{x}_{true}$, but since $\mathbf{x}_{true}$ is unknown, we can't measure the distance. Instead, we settle for computing a bound on the distance.

A motivating example

Example 1: Suppose $\delta < .5 * \epsilon_{mach}$.

$$\begin{bmatrix} \delta & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

If we solve this system without pivoting, we'll get

$$\begin{bmatrix} \delta & 1 \\ 0 & -1/\delta \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1/\delta \end{bmatrix}$$

so

$$x_2 = 1, \;\; x_1 = 0.$$

The true solution is

$$\mathbf{x} = \begin{bmatrix} -\frac{1}{1-\delta} \\ \frac{1}{1-\delta} \end{bmatrix},$$

so our answer is very bad. This is because we used an unstable algorithm: Gauss elimination without pivoting.

If we use pivoting, our answer improves: the linear system

$$\begin{bmatrix} \delta & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

is rewritten as

$$\begin{bmatrix} 1 & 1 \\ \delta & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

so the elimination gives us

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

so

$$x_2 = 1, \;\; x_1 = -1\,.$$

This is quite close to the true solution!

In fact, we can calculate the residual, the remainder that we get when we substitute our approximate $\mathbf{x}$ into the equations:

$$\begin{aligned}\mathbf{r} &\equiv \mathbf{b} - \mathbf{A}\mathbf{x} \\ &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ \delta & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ \delta \end{bmatrix}.\end{aligned}$$

This is quite small! So we have (exactly) solved the problem

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

and this is quite close to what we wanted to solve.

[]

Important notes

This is an example of backward error analysis, discussed at the beginning of the course, in the notes on errors. If the data in the problem had some uncertainty, then we may have solved a problem just as good as the original one.

So we now have two ways to measure error: the difference between the true solution and the computed solution gives us the forward error, uncomputable unless we know the true solution, and the residual gives us the backward error, which is computable.

In this first example, Gauss elimination gave us both small forward error and small backward error. Does this always happen?

A second example

Let's assume 3-digit decimal arithmetic.

$$\begin{bmatrix} .780 & .563 \\ .913 & .659 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} .217 \\ .254 \end{bmatrix}$$

If we compute the solution with pivoting, we obtain

$$\mathbf{x} = \begin{bmatrix} -.443 \\ 1.000 \end{bmatrix}, \quad \mathbf{r} = \begin{bmatrix} -.000460 \\ -.000541 \end{bmatrix}$$

and the true solution is

$$\mathbf{x}_{true} = \begin{bmatrix} 1.000 \\ -1.000 \end{bmatrix}.$$

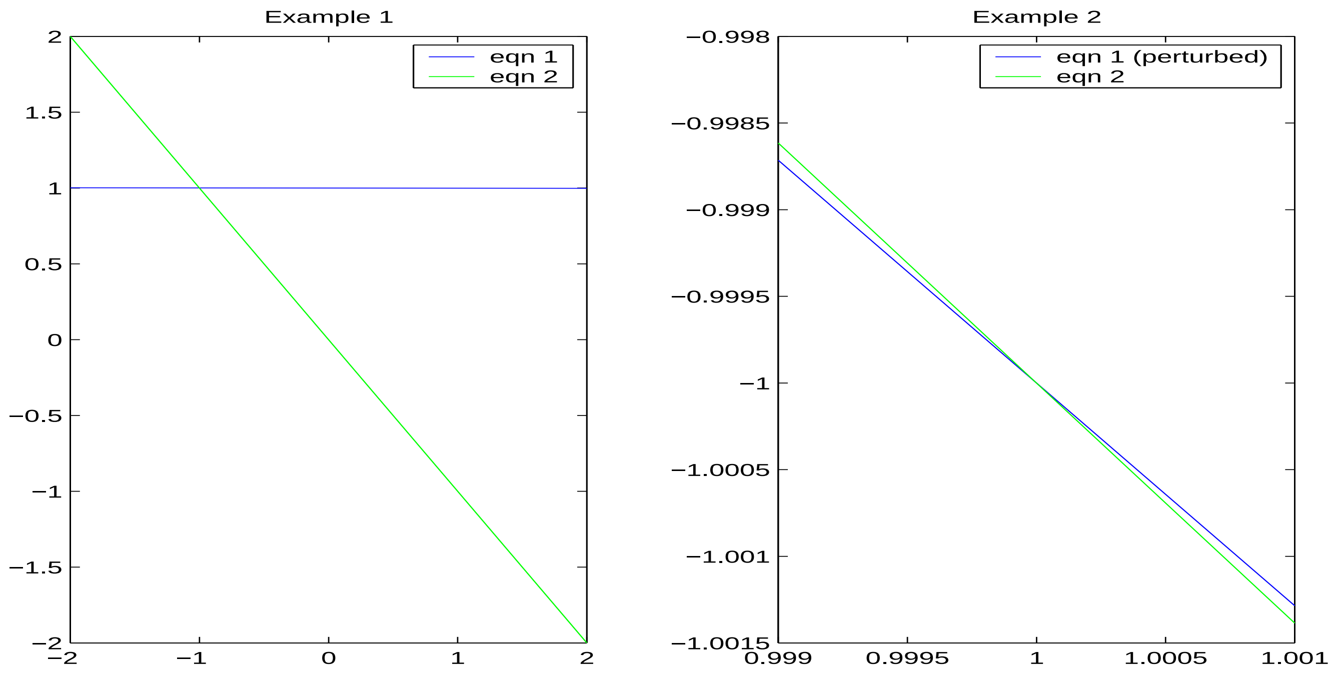

Figure 1: Graphs of the equations for the two problems. For the second one, I needed to perturb the first line a little to make it visible. Note the difference in scales.

In this case, Gauss elimination with pivoting produced a small residual but not a small x-error.

[]

What characterizes the difference between the two problems?

The first problem is well-conditioned; small changes in the data produce small changes in the answer.

The second problem is ill-conditioned; small changes in the data can produce large changes in the answer.

We can illustrate this by graphing the two systems.

A guarantee

Gauss Elimination with pivoting is guaranteed to produce a small residual

The small print:

- need reasonable floating point hardware (IEEE will do)
- "small" relative to the
  - original data.
  - intermediate data.
  - computed solution.

Note that this does not guarantee that the x-error will be small. For this, we need a well conditioned problem.

Measuring conditioning

We measure the conditioning of the linear system by measuring the condition number of the matrix $\mathbf{A}$, using the 1-norm:

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

Matlab has a function to estimate this, without computing $\mathbf{A}^{-1}$, at rather small cost. It is called `condest`.

How do we estimate x-error?

Consider

$$\begin{array}{rcl} \mathbf{A}\mathbf{x}_{true} = \mathbf{b} & \rightarrow & \|\mathbf{b}\| = \|\mathbf{A}\mathbf{x}_{true}\| \le \|\mathbf{A}\|\,\|\mathbf{x}_{true}\| \\ \|\mathbf{x}_{true}\| \ge \dfrac{\|\mathbf{b}\|}{\|\mathbf{A}\|} & \rightarrow & \dfrac{1}{\|\mathbf{x}_{true}\|} \le \dfrac{\|\mathbf{A}\|}{\|\mathbf{b}\|} \end{array}$$

Also:

$$\begin{array}{rcl} \mathbf{A}\mathbf{x} = \mathbf{b} - \mathbf{r} & \rightarrow & \mathbf{A}(\mathbf{x}_{true} - \mathbf{x}) = \mathbf{r} \\ (\mathbf{x}_{true} - \mathbf{x}) = \mathbf{A}^{-1}\mathbf{r} & \rightarrow & \|\mathbf{x}_{true} - \mathbf{x}\| \le \|\mathbf{A}^{-1}\|\,\|\mathbf{r}\| \end{array}$$

Therefore,

$$\begin{aligned} \frac{\|\mathbf{x}_{true} - \mathbf{x}\|}{\|\mathbf{x}_{true}\|} &\le \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}\|\mathbf{A}\|\|\mathbf{A}^{-1}\| \\ &= \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}\kappa(\mathbf{A})\,. \end{aligned}$$

Some properties:

- $\kappa(\mathbf{A}) \geq 1$ for all matrices.
- $\kappa(\mathbf{A}) = \infty$ for singular matrices.
- $\kappa(c\mathbf{A}) = \kappa(\mathbf{A})$ for any nonzero scalar c.
- $\kappa(\mathbf{D}) = \max |d_{ii}| / \min |d_{ii}|$ if $\mathbf{D}$ is diagonal.
- κ measures closeness to singularity better than the determinant.

Final words

- Never compute a matrix inverse unless you really want to look at the entries of the matrix.
- Always use a stable algorithm, like Gauss elimination with pivoting.
- Check the residual and the condition number for the problem to see how good your solution is.
- Question the solution if the condition number is large, and try to reformulate the problem in a better way.