Computational Physics
Lecture 3 – Matrix Methods

Pat Scott

Department of Physics, Imperial College

October 9, 2018

Slides available from https://bb.imperial.ac.uk/

Goals

By the end of today's lecture, you should be able to

- Solve matrix equations numerically
- Obtain the inverse of a matrix numerically
- Implement a simple elimination scheme to do the above
- Numerically decompose a matrix into a product of upper and lower triangular matrices

Outline

Outline

Matrix equations

Want to solve a set of M linear equations in N unknowns $x_{j=1..N}$:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1,N}x_N &= b_1 \quad (1)\\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2,N}x_N &= b_2 \quad (2)\\
\ldots & \quad (3)\\
a_{M,1}x_1 + a_{M,2}x_2 + \ldots + a_{M,N}x_N &= b_M \quad (4)
\end{aligned}
$$

These are most easily represented as matrices and vectors

$$A\mathbf{x} = \mathbf{b} \quad (5)$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots \\ a_{21} & a_{22} & \ldots \\ \ldots & \ldots & \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \ldots \end{bmatrix}. \quad (6)$$

Desired outcomes

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

What might you want out of this system of equations?

Desired outcomes

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

What might you want out of this system of equations?

1. One or more solutions – find x for a given $\mathbf{b}$.
   Maybe you want to do this many times over for different $\mathbf{b}$, in order to obtain different solutions for x.

Desired outcomes

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

What might you want out of this system of equations?

1. One or more solutions – find x for a given **b**.
   Maybe you want to do this many times over for different **b**, in order to obtain different solutions for x.
2. The inverse of A.

Desired outcomes

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

What might you want out of this system of equations?

1. One or more solutions – find x for a given **b**.
   Maybe you want to do this many times over for different **b**, in order to obtain different solutions for x.

2. The inverse of A.
   Note that this is not the same as 1, and usually not the best way to achieve 1!

Which method is best for my problem?

Many, many, many^{many} canned matrix solution and inversion codes exist

- Highly optimised
- Well-know public free examples: LAPACK, BLAS
- Practically, you will usually want to use one of these

http://www.netlib.org/lapack/
http://www.netlib.org/blas/

Which method is best for my problem?

However, a few simple methods are worth knowing about:

Which method is best for my problem?

However, a few simple methods are worth knowing about:

1. Gauss-Jordan elimination
    - Computes set number of solutions and inverse in one hit
    - All needs to be done over again for new **b** values
    - super-simple

Which method is best for my problem?

However, a few simple methods are worth knowing about:

1. Gauss-Jordan elimination
   - Computes set number of solutions and inverse in one hit
   - All needs to be done over again for new **b** values
   - super-simple
2. LU decomposition and back-substitution
   - Faster when just solutions are needed
   - Fast and accurate to apply to new **b** values
   - Similar speed to Gauss-Jordan for inverses
   - more complicated but almost always better choice

Outline

How you would proceed without a computer...

Question:

What is the most naive way to solve $A\mathbf{x} = \mathbf{b}$?

Answer:

Replace rows (single equations) by linear combinations of each other until you've transformed A into the identity $\mathbf{1}$
$\implies$ you will have transformed $A\mathbf{x}$ into just $\mathbf{x}$ and $\mathbf{b}$ into $\mathbf{b}A^{-1}$, i.e. you will have

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

and be able to just read the solution off the righthand side.

Carrying out the transformation

Set the problem up as follows:

$$\begin{bmatrix} a_{11} & a_{12} & \dots \\ a_{21} & a_{22} & \dots \\ \dots & \dots & \end{bmatrix} \left[\begin{pmatrix} x_{11} \\ x_{21} \\ \dots \end{pmatrix} \begin{pmatrix} x_{12} \\ x_{22} \\ \dots \end{pmatrix} \dots \right] = \left[\begin{pmatrix} b_{11} \\ b_{21} \\ \dots \end{pmatrix} \begin{pmatrix} b_{12} \\ b_{22} \\ \dots \end{pmatrix} \dots \right] \tag{7}$$

So, how do you achieve the transformation $A \rightarrow \mathbf{1}$? You need to work one column at a time. For a given column j,

1. Divide the jth row by a_{jj}, ensuring that $a_{jj} \rightarrow 1$. OK, so now you have the jth entry on the diagonal of the identity.
2. Subtract an appropriate multiple of the jth row from all the other entries in the column, so that all the other entries in column get set to zero. Now you have the non-diagonal elements for the jth column.

Carrying out the transformation

Set the problem up as follows:

$$\begin{bmatrix} a_{11} & a_{12} & \dots \\ a_{21} & a_{22} & \dots \\ \dots & \dots & \end{bmatrix} \left[\begin{pmatrix} x_{11} \\ x_{21} \\ \dots \end{pmatrix} \begin{pmatrix} x_{12} \\ x_{22} \\ \dots \end{pmatrix} \dots \right] = \left[\begin{pmatrix} b_{11} \\ b_{21} \\ \dots \end{pmatrix} \begin{pmatrix} b_{12} \\ b_{22} \\ \dots \end{pmatrix} \dots \right]$$

So, how do you achieve the transformation $A \rightarrow \mathbf{1}$? You need to work one column at a time. For a given column j,

1. Divide the jth row by a_{jj}, ensuring that $a_{jj} \rightarrow 1$. OK, so now you have the jth entry on the diagonal of the identity.
2. Subtract an appropriate multiple of the jth row from all the other entries in the column, so that all the other entries in column get set to zero. Now you have the non-diagonal elements for the jth column.

Carrying out the transformation

Set the problem up as follows:

$$\begin{bmatrix} 1 & a_{12}/a_{11} & \dots \\ a_{21} & a_{22} & \dots \\ \dots & \dots & \end{bmatrix} \left[\begin{pmatrix} x_{11} \\ x_{21} \\ \dots \end{pmatrix} \begin{pmatrix} x_{12} \\ x_{22} \\ \dots \end{pmatrix} \right] = \left[\begin{pmatrix} b_{11}/a_{11} \\ b_{21} \\ \dots \end{pmatrix} \begin{pmatrix} b_{12}/a_{11} \\ b_{22} \\ \dots \end{pmatrix} \right]$$

So, how do you achieve the transformation $A \rightarrow \mathbf{1}$? You need to work one column at a time. For a given column j,

1. Divide the jth row by a_{jj}, ensuring that $a_{jj} \rightarrow 1$. OK, so now you have the jth entry on the diagonal of the identity.
2. Subtract an appropriate multiple of the jth row from all the other entries in the column, so that all the other entries in column get set to zero. Now you have the non-diagonal elements for the jth column.

Carrying out the transformation

Set the problem up as follows:

$$\begin{bmatrix} 1 & a_{12}/a_{11} & \dots \\ a_{21} & a_{22} & \dots \\ \dots & \dots & \end{bmatrix} \left[\begin{pmatrix} x_{11} \\ x_{21} \\ \dots \end{pmatrix} \begin{pmatrix} x_{12} \\ x_{22} \\ \dots \end{pmatrix} \right] = \left[\begin{pmatrix} b_{11}/a_{11} \\ b_{21} \\ \dots \end{pmatrix} \begin{pmatrix} b_{12}/a_{11} \\ b_{22} \\ \dots \end{pmatrix} \right]$$

So, how do you achieve the transformation $A \to \mathbf{1}$? You need to work one column at a time. For a given column j,

1. Divide the jth row by a_{jj}, ensuring that $a_{jj} \to 1$. OK, so now you have the jth entry on the diagonal of the identity.
2. Subtract an appropriate multiple of the jth row from all the other entries in the column, so that all the other entries in column get set to zero. Now you have the non-diagonal elements for the jth column.

Carrying out the transformation

Set the problem up as follows:

$$\left[\begin{array}{ccc} 1 & a_{12}/a_{11} & \dots \\ 0 & a_{22} - a_{21}a_{12}/a_{11} & \dots \\ \dots & \dots & \end{array}\right] \left[\left(\begin{array}{c} x_{11} \\ x_{21} \\ \dots \end{array}\right) \left(\begin{array}{c} x_{12} \\ x_{22} \\ \dots \end{array}\right) \right]$$

$$= \left[\left(\begin{array}{c} b_{11}/a_{11} \\ b_{21} - a_{21}b_{11}/a_{11} \\ \dots \end{array}\right) \left(\begin{array}{c} b_{12}/a_{11} \\ b_{22} - a_{21}b_{12}/a_{11} \\ \dots \end{array}\right) \right]$$

So, how do you achieve the transformation $A \to \mathbf{1}$? You need to work one column at a time. For a given column j,

1. Divide the jth row by a_{jj}, ensuring that $a_{jj} \to 1$. OK, so now you have the jth entry on the diagonal of the identity.
2. Subtract an appropriate multiple of the jth row from all the other entries in the column, so that all the other entries in column get set to zero. Now you have the non-diagonal elements for the jth column.

Carrying out the transformation

Perform this algorithm yourself by hand for a 2×2 example (2–3 mins, feel free to discuss):

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

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = ?? \tag{8}$$

Work one column at a time. For a given column j,

1. Divide the jth row by a_{jj}, ensuring that $a_{jj} \to 1$. OK, so now you have the jth entry on the diagonal of the identity.
2. Subtract an appropriate multiple of the jth row from all the other entries in the column, so that all the other entries in column get set to zero. Now you have the non-diagonal elements for the jth column.

Carrying out the transformation

Perform this algorithm yourself by hand for a 2×2 example (2–3 mins, feel free to discuss):

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \end{bmatrix}$$
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \frac{10}{3} \\ -\frac{2}{3} \end{bmatrix} \tag{8}$$

Work one column at a time. For a given column j,

1. Divide the jth row by a_{jj}, ensuring that $a_{jj} \to 1$. OK, so now you have the jth entry on the diagonal of the identity.
2. Subtract an appropriate multiple of the jth row from all the other entries in the column, so that all the other entries in column get set to zero. Now you have the non-diagonal elements for the jth column.

Pivoting

Well, you're almost done.

- Notice that you're dividing by a_{jj} at each step.
- What if $a_{jj} = 0$? Everything breaks.
- You need to pivot.
- $\rightarrow$ pre-shuffle the rows so that you never get a zero on the diagonal.
- Actually the most accuracy comes from shuffling so that the *biggest* value of each column ends up on the diagonal.
- Some codes renormalise each row to maximum value 1 to start with $\rightarrow$ *implicit pivoting*

Pivoting is a bit fiddly and annoying, as you need to keep track of the row swaps – but it's essential for making a robust matrix solver.

Outline

Triangular Matrices

Upper triangular matrix U has diagonal and upper elements.
Lower triangular matrix L has diagonal and lower elements.

$$U = \begin{bmatrix} u_{11} & u_{12} & \dots \\ 0 & u_{22} & \dots \\ 0 & 0 & \dots \end{bmatrix}, L = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \dots \end{bmatrix}. \tag{9}$$

Imagine if we could write $A = LU$. Then,

$$A\mathbf{x} = LU\mathbf{x} = L(U\mathbf{x}) = \mathbf{b}, \tag{10}$$

i.e. if we can solve

$$L\mathbf{y} = \mathbf{b} \tag{11}$$

$$U\mathbf{x} = \mathbf{y} \tag{12}$$

then we can solve $A\mathbf{x} = \mathbf{b}$.

Forward and back-substitution

Actually, $L\mathbf{y} = \mathbf{b}$ and $U\mathbf{x} = \mathbf{y}$ are very easy to solve:

$$y_1 = \frac{b_1}{\ell_{11}} \tag{13}$$

$$y_i = \frac{1}{\ell_{ii}} \left(b_i - \sum_{j=1}^{i-1} \ell_{ij} y_j \right), i \neq 1 \tag{14}$$

$$x_N = \frac{y_N}{u_{N,N}} \tag{15}$$

$$x_i = \frac{1}{u_{ii}} \left(y_i - \sum_{j=i+1}^{N} u_{ij} x_j \right), i \neq N \tag{16}$$

So, all that remains is to work out how to get L and U...

The actual decomposition

If we actually write out $A = LU$ we get:

$$i \leq j: \qquad \ell_{i1}u_{1j} + \ell_{i2}u_{2j} + \ldots + \ell_{ii}u_{ij} = a_{ij} \tag{17}$$

$$i \geq j: \qquad \ell_{i1}u_{1j} + \ell_{i2}u_{2j} + \ldots + \ell_{ij}u_{jj} = a_{ij} \tag{18}$$

This is N^2 equations in $N^2 + N$ unknowns $\implies$ N free entries to choose at will.

So, just choose L to have entries of 1 on the diagonal, for simplicity.

The system of equations can then be easily solved with *Crout's Algorithm*.

Crout's Algorithm

Essentially just reshuffles the rows in the equation $A = LU$:

1. Set all $\ell_{ii} = 1$ (i.e. for $1 \leq i \leq N$).
2. For each $j = 1..N$:
   a) Use first equation (17) with $\ell_{ii} = 1$ to get

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} \ell_{ik} u_{kj} \tag{19}$$

   b) Use second equation (18) to get

$$\ell_{ij} = u_{jj}^{-1} \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} u_{kj} \right) \tag{20}$$

If working up in j, each u and ℓ entry gets calculated before it is needed.

Crout's Algorithm

Essentially just reshuffles the rows in the equation $A = LU$:

1. Set all $\ell_{ii} = 1$ (i.e. for $1 \leq i \leq N$).
2. For each $j = 1..N$:
   a) Use first equation (17) with $\ell_{ii} = 1$ to get

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} \ell_{ik} u_{kj} \tag{19}$$

   b) Use second equation (18) to get

$$\ell_{ij} = u_{jj}^{-1} \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} u_{kj} \right) \tag{20}$$

If working up in j, each u and ℓ entry gets calculated before it is needed.

Note that pivoting is still needed for robustness! (Luckily you get an example in your assignment that is OK without.)

Putting it all together

Now you have L and U.

- $\implies$ can solve for $\mathbf{y}$ using using forward substitution.
- $\implies$ can solve for $\mathbf{x}$ using back-substitution.
- $\rightarrow$ linear system is solved.

You can get $\det(A)$ just by multiplying diagonals of U

$$\det(A) = \prod_{i=1}^{N} u_{ii} \tag{21}$$

...and you can get the jth column of A^{-1} just by solving for $\mathbf{x}$ with $\mathbf{b}$ set to the unit vector $\mathbf{b} = \mathbf{e}_j$.

Housekeeping

- Problem Sheet for Lecture 3 available now
  - Manipulate a few matrices
  - Implement Gauss-Jordan Elimination (+ its variant Gaussian Elimination)
- Later you will get the Assignment
  - Q2 covers LU decomposition
- Next lecture: Interpolation ()

Bonus: Coding search tips

- Try googling errors directly, minus any project-specific output
- https://stackoverflow.com/
- Know whether you are using Python 2.7 or 3.x
  https://docs.python.org/3/
  https://docs.python.org/2/
- Read the documentation of the command carefully
- Read the documentation of anything you don't understand in the documentation of the command carefully
- Read the documentation of anything you don't understand in the documentation of anything you don't understand in the documentation of the command carefully
- Read the. . . (just read a lot)
- Copy the documented example into your session. Run it. Keep changing it to be more like your code until it breaks. That's your error.