Lecture 4: Linear Algebra 1

Sourendu Gupta

TIFR Graduate School

Computational Physics 1
February 12, 2010

1. Linear problems
   - Motivation
   - Norms
   - Computing dot products: introduction to parallel computing

2. Solving linear equations
   - Gauss Elimination
   - LU decomposition
   - Odds and ends

3. References

Outline

1. Linear problems
   - Motivation
   - Norms
   - Computing dot products: introduction to parallel computing
2. Solving linear equations
   - Gauss Elimination
   - LU decomposition
   - Odds and ends
3. References

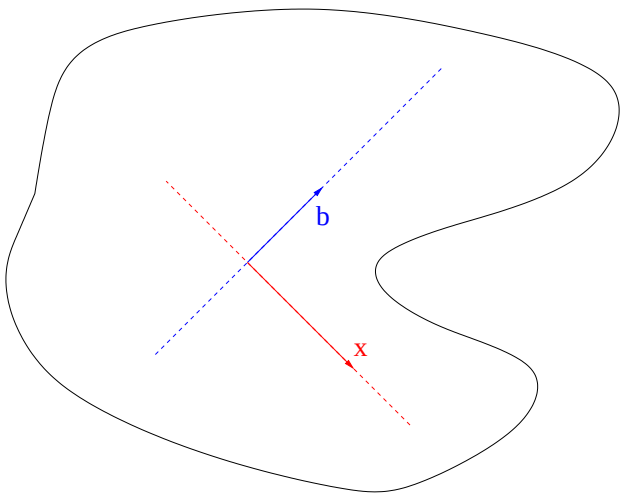
What is a linear problem?

The typical linear problem is solving a set of linear algebraic equations—

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_N \end{pmatrix}, \qquad i.e., \qquad A\mathbf{x} = \mathbf{b}.$$

But such problems can arise in a surprising variety of contexts. In the last lecture we saw that function interpolation involves solving linear equations. How do we recognize a linear problem? The simplest definition involves scaling the unknown quantities in the problem. In the above problem the unknown is the vector $\mathbf{x}$. If you scale this by some amount, $\mathbf{x}' = \lambda\mathbf{x}$, then this new $\mathbf{x}'$ is the solution of the equations for a new $\mathbf{b}' = \lambda\mathbf{b}$. Such scalings define linear problems.

Scaling a vector generates a line



For every vector **b** on the right there is a vector **x** which is the solution of the equations. Scaling a vector generates another parallel vector, and hence defines a line. A linear problem gives straight lines in the space of solutions when the input comes from straight lines in the space of data.

Problem 1: Is this linear problem well-conditioned on **b**? That is, if **b** is changed continuously by a small amount, then does **x** also change continuously by a small amount? Is it well-conditioned on the matrix?

Vector and matrix norms

A vector norm $||\mathbf{x}||$ must have the following properties, $||\lambda\mathbf{x}|| = \lambda||\mathbf{x}||$, $||\mathbf{x}|| = 0$ if and only if $\mathbf{x} = 0$, and $||\mathbf{x}+\mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||$. The p-norm of an N-dimensional vector is

$$||\mathbf{x}||_p = \left[\sum_{i=1}^{N} |x_i|^p\right]^{1/p}.$$

The 2-norm is the usual Euclidean norm. Sometimes one needs the ∞-norm, which is just the element of the vector with the largest absolute value. The 1-norm is sometimes called the Manhattan norm.
Every vector norm induces a matrix norm. The induced norm of a matrix, A, is

$$||A||_p = \sup_{\mathbf{x}, ||x||_p \neq 0} \frac{||A\mathbf{x}||_p}{||\mathbf{x}||_p}.$$

An obvious consequence is $||A\mathbf{x}|| \leq ||A||\ ||\mathbf{x}||$.

The triangle inequality

Problem 2: Examine the triangle inequality for the p-norm of vectors in one dimension. Is it satisfied? Is there any stronger condition which is satisfied? Are all real values of p allowed as a p-norm?

Problem 3: Examine the triangle inequality for the p-norm of vectors in two dimensions. Is it satisfied when the norm of the two vectors are equal? What happens when the norm of one vector changes? Can you establish a proof for the triangle inequality in this case? Are all real values of p allowed as a p-norm?

Problem 4: Why is it sufficient to prove the triangle inequality in two dimensions in order to establish it in any dimension?

The 2-norm of a matrix

For any matrix A, the 2-norm is

$$||A||_2 = \sup_{\mathbf{x}, ||x||_2 \neq 0} \sqrt{\frac{\mathbf{x}^\dagger A^\dagger A \mathbf{x}}{\mathbf{x}^\dagger \mathbf{x}}},$$

where $A^\dagger$ denotes Hermitean conjugation— $(A^\dagger)_{ij} = A^*_{ji}$ and $\mathbf{x}^\dagger \mathbf{x}$ is just the dot product of the vector with itself. Note also that $B = A^\dagger A$ is a Hermitean matrix, *i.e.*, $B^\dagger = B$. Therefore, the eigenvalues of B are real. Call them σ_i. If the orthonormalized eigenvectors of B are $\mathbf{v}_i$, then any vector $\mathbf{x} = \sum_i x_i \mathbf{v}_i$. As a result,

$$||A||_2 = \sup_{\mathbf{x}, ||x||_p \neq 0} \sqrt{\frac{\sum_i \sigma_i |x_i|^2}{\sum_i |x_i|^2}} = \max_i \sqrt{\sigma_i}.$$

The eigenvalues of $A^\dagger A$ are called the singular values of A. If A is Hermitean, then the singular values are the squares of the eigenvalues of A.

The ∞-norm of a matrix

The ∞-norm of a matrix A is

$$||A||_\infty = \sup_{\mathbf{x}, ||x||_\infty \neq 0} \operatorname{Lim}_{p\to\infty} \left(\frac{\sum_{ij} |A_{ij}x_j|^p}{\sum_j |x_j|^p} \right)^{1/p}.$$

The numerator is the term in the sum $|A_{ij}x_j|$ which is the largest and the denominator is the component of $\mathbf{x}$ which is the largest in magnitude. If the maximum in the numerator does not come from that ij for which $|A_{ij}|$ is the largest, then by just increasing the corresponding $|x_j$ one can increase the ratio. Therefore, as one varies the components of $\mathbf{x}$, the largest term in the numerator will be the one which picks out the largest component of A. By varying over such $\mathbf{x}$ it is clear that one will eventually have

$$||A||_\infty = \max_{ij}|A_{ij}|.$$

Is $||A||_\infty$ invariant under unitary transformations? What is an efficient method for finding $||A^{-1}||_\infty$?

Dot products of vectors

The dot product of two N-dimensional complex vectors $\mathbf{x}$ and $\mathbf{y}$ is

$$\mathbf{x}^\dagger \mathbf{y} = \sum_{i=1}^{N} x_i^* y^i.$$

This requires N complex conjugations, complex additions and complex multiplications. Each complex conjugation requires one sign flip, a complex addition is two real additions, a complex multiplication is four real multiplications and two real additions. Hence the operation count is $9N$. For the dot product of two real vectors the operation count is $2N$. We say that the complexity of the dot product is $\mathcal{O}(N)$.

The sum is performed by sending one element of each vector from memory to the arithmetic-logic unit (ALU) in the CPU, and performing the sum into an accumulator in the ALU. This pipeline from memory to ALU is the slowest part of the computation. A typical CPU on a desktop today has 3 GHz clock, but the memory transfer is less than 1 MHz-bit. So one 4-byte real number takes more than 12 CPU cycles to transfer, but only 2 cycles to process.

Parallel dot products

Let's think of parallelizing the operation. Assume we have P processors and we distribute the components of the vector equally over them, then the dot product becomes

$$\mathbf{x}^\dagger \mathbf{y} = \sum_{j=0}^{P-1} \left[\sum_{i=j*(N/P)+1}^{(j+1)*(N/P)} x_i^* y_i \right].$$

The j-th processor does the partial sum given within brackets. The outer sum needs $2P$ operations no matter how we break it up, so the total is $9(N/P) + 2P$ operations. By this count the optimum number of processors is $P = \sqrt{9N/2}$. A similar counting for real vectors shows that the optimum number of processors for that case is $P = \sqrt{2N}$. The parallel complexity of a dot product is $\mathcal{O}(\sqrt{N})$.

Consider again the transfer speeds. The bandwidth of the network between processors can be factors of 10–1000 slower than that from memory to ALU. Hence the parallel operation is dominated by communication speed.

Parallel speedup and scaling

Parallel speedup, SP, is the ratio of time spent on a problem by a serial computer and a parallel one. In this counting we found that the parallel speedup was $SP = 9N/[9N/P + (2+b)P]$, where b is the number of clock cycles needed to transfer one word. It is of practical importance to understand how SP scales.

When N is fixed and P changes, the scaling of work with changing P is called strong scaling. In this case, we find that for $P \ll N$ one has linear speedup, *i.e.*, $SP \propto \mathcal{O}(P)$. However, when $P \simeq \sqrt{9N/(2+b)}$ the speed up is sublinear, at best $SP \propto \mathcal{O}(\sqrt{P})$. For much larger P there is no speedup, since $SP \propto \mathcal{O}(1/P)$. It is cost-efficient to work in the regime of linear speedup.

Weak scaling is when the problem size, N, is changed as P changes in order to solve large problems efficiently. For the dot product, if $N \propto P^{\alpha}$, with $\alpha \geq 1$, then one has linear speedup.

Outline

1. Linear problems
   - Motivation
   - Norms
   - Computing dot products: introduction to parallel computing
2. Solving linear equations
   - Gauss Elimination
   - LU decomposition
   - Odds and ends
3. References

When should one even bother?

If the equations $A\mathbf{x} = \mathbf{b}$ are such that small changes in $\mathbf{b}$ lead to enormous changes in $\mathbf{x}$, then it may not be worthwhile trying to solve the equations. If $\delta\mathbf{b}$ is a change in $\mathbf{b}$ and $\delta\mathbf{x}$ is the induced change in $\mathbf{x}$, then

$$||\mathbf{b}|| \leq ||A||||\mathbf{x}|| \qquad \text{and} \qquad ||\delta\mathbf{x}|| \leq ||A^{-1}||||\delta\mathbf{b}||.$$

Therefore one can bound the fractional change in $\mathbf{x}$—

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

The condition number of a matrix A is

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

If the condition number of a matrix is large, then small changes in the right hand side of the equations cause large changes in the solutions. If the condition number is comparable to the inverse of the machine precision then any solution of the system of equations is essentially meaningless.

Gauss elimination

Given a system of N linear equations in N variables, $A\mathbf{x} = \mathbf{b}$, a systematic solution exists. Add an appropriate multiple of the first equation to each of the others to eliminate the first variable from the remainder. Then add a multiple of the second to the equations below to eliminate the second variable, *etc.*. At the N-th step, we have the upper triangular form

$$\begin{pmatrix} - & - & - & \cdots & - & - \\ 0 & - & - & \cdots & - & - \\ 0 & 0 & - & \cdots & - & - \\ 0 & 0 & 0 & \cdots & - & - \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & - \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} - \\ - \\ - \\ - \\ \vdots \\ - \end{pmatrix},$$

where the $-$ are non-zero numbers. This triangular system can be solved by back substitution from the last equation up. Programming this is straightforward.

Problem 5: Give an example of a system with non-singular A for which Gauss elimination fails.

A problem with Gauss elimination

A problem with the simple Gauss elimination process is illustrated by this example—

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix},$$

where ϵ is a number close to the machine precision. Note that the problem is perfectly solvable: the determinant is $-1+\epsilon$ and the condition number is not far from unity. The exact answer is

$$y = \frac{a - b\epsilon}{1-\epsilon} = a + \mathcal{O}(\epsilon), \quad x = \frac{b-a}{1-\epsilon} = b - a + \mathcal{O}(\epsilon).$$

The elimination gives

$$\left(\begin{array}{cc|c} \epsilon & 1 & a \\ 1 & 1 & b \end{array}\right) \longrightarrow \left(\begin{array}{cc|c} \epsilon & 1 & a \\ 0 & 1-\frac{1}{\epsilon} & b-\frac{a}{\epsilon} \end{array}\right) \approx \left(\begin{array}{cc|c} \epsilon & 1 & a \\ 0 & -\frac{1}{\epsilon} & -\frac{a}{\epsilon} \end{array}\right),$$

where the last step is the most likely result after rounding. Back substitution will then give $y \approx a$ and $x \approx 0/\epsilon$! During back-substitution the small relative error in elimination is magnified: this is catastrophic loss of precision.

The solution: pivoting

A small change will solve this problem. Start by interchanging the order of the equations.

$$\begin{pmatrix} 1 & 1 \\ \epsilon & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ a \end{pmatrix},$$

This is already approximately in the form required. If ϵ were small enough, the 21 element would be rounded to zero, and you could do the back-substitution straight away. Elimination gives

$$\left(\begin{array}{cc|c} 1 & 1 & b \\ \epsilon & 1 & a \end{array}\right) \longrightarrow \left(\begin{array}{cc|c} \epsilon & 1 & b \\ 0 & 1-\epsilon & a-\epsilon b \end{array}\right).$$

No problems arise at any point. The crucial point is that during Gauss elimination whenever you might have to divide by a small number, just interchange that row with one of the rows below which doesn't have a small number in that column. This is called pivoting.

Another problem with Gauss elimination

Consider the equations

$$\begin{pmatrix} 1 & 10^{14} & 10^{14} \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}.$$

The first step of Gauss elimination gives

$$\left(\begin{array}{ccc|c} 1 & 10^{14} & 10^{14} & a \\ 0 & 1-2\times 10^{14} & 3-2\times 10^{14} & b-2a \\ 0 & 1-10^{14} & 1-10^{14} & c-a \end{array}\right).$$

Since the three row vectors are nearly parallel, further steps will result in catastrophic loss of accuracy. The solution is to equilibrate the matrix, *i.e.*, multiplying each equation by an overall number such as to reduce the largest element to something of order unity. After pivoting, this gives

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & c \\ z & 1 & 1 & az \\ 2 & 1 & 3 & b \end{array}\right) \quad \text{and} \quad \left(\begin{array}{ccc|c} 1 & 1 & 1 & c \\ 0 & 1-z & 1-z & (a-c)z \\ 0 & -1 & 1 & b-2c \end{array}\right),$$

where $z = 10^{-14}$. Clearly, the rest has no any numerical instability.

Complexity of Gauss elimination

Consider Gauss elimination for a system of N equations in N variables. Eliminating at the i-th row requires $N - i + 2$ multiplications and the same number of subtractions. Back substituting in the i-th row requires $N - i$ multiplications, $N - i + 1$ subtractions, and one division. The total count of operations is

$$\sum_{i=1}^{N} 2(N - i + 2) + N - i + N - i + 1 + 1 = \sum_{i=1}^{N} (4N - 4i + 6) = \mathcal{O}(N^2).$$

A straightforward application of Kramer's rule would have resulted in an operation count of $\mathcal{O}(N^3)$.

If one moves the numbers required for pivoting from one location to another, then that could require potentially $\mathcal{O}(N^2)$ operations. Instead of this overhead, one therefore keeps an array of pointers: `row(i)`. Initially, `row(i)`$\rightarrow$`i`. Every time you need to interchange rows α and β just set `row(`α`)`$\rightarrow \beta$ and `row(`β`)`$\rightarrow \alpha$.

Problem 6: Is the description of the pointers above correct? If one of the rows involved has been interchanged already, what should you do? What is the correct way to incorporate pointers for pivoting in Gauss elimination.

The theory of Gauss elimination

Write the coefficient matrix, A, in the form of a product, $A = LU$, where L is lower triangular (all elements above the diagonal are zero) and U is upper triangular (all elements below the diagonal are zero). Then the elimination process converts the equation $LU\mathbf{x} = \mathbf{b}$ to $U\mathbf{x} = L^{-1}\mathbf{b}$. It is possible for the condition number of A to be okay, but that of U to be very large. In that case, the inversion of U can be problematic.
For example,

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{1}{\epsilon} & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 1 \\ 0 & 1 - \frac{1}{\epsilon} \end{pmatrix}, \qquad \begin{pmatrix} 1 & 1 \\ \epsilon & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \epsilon & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 - \epsilon \end{pmatrix}.$$

For the first form of A, the two row vectors of U are nearly parallel! As a result, $\kappa(U) = \mathcal{O}(\epsilon^{-2})$! Since $\kappa(L) = 1$, the elimination process is non-problematic; only the back substitution step (*i.e.*, obtaining U^{-1}) is numerically unstable. In the second case both $\kappa(L)$ and $\kappa(U)$ are near unity. When $\kappa(A)$ is acceptable, the controlling factor for accuracy of Gauss elimination is $\kappa(U)$. Pivoting helps to control this.

Uniqueness of the LU decomposition

Given a matrix A, we construct a factorization in terms of a product of a lower triangular (L) and an upper triangular (U) matrix, $A = LU$, such that the diagonal element of L are all unity. Such a factorization is unique. Suppose there is another factorization $A = L'U'$. As long as A is non-singular, so are L, U, L' and U', since $\det A = \det L \times \det U$. Hence the inverses of the upper and lower triangular matrices exist. Since $LU = L'U'$, one has $L^{-1}L' = UU'^{-1}$. Now the left hand side is lower triangular whereas the right hand side is upper triangular. Hence one must have both sides equal to the identity. Therefore $L' = L$ and $U' = U$.

1. Note that $\det L = 1$, so $\det U = \det A$. Gauss elimination allows us to find the determinant of a matrix in $\mathcal{O}(N^2)$ operations.
2. If one allows the diagonal elements of L to be arbitrary then the factorization is not unique.
3. An explicit construction of L is useful if there are several right hand sides, $\mathbf{b}_1$, $\mathbf{b}_2$, *etc.*.

The LU decomposition

In the decomposition $A = LU$, consider the r-th row of A. Elements of this row are obtained by the dot products of the r-th row of L with successive columns of U—

$$a_{ri} = l_{r1}u_{1i} + l_{r2}u_{2i} + \cdots + l_{r,r-1}u_{r-1,i} + u_{ri}.$$

In these N equations, the unknowns are $r-1$ of the l_{ri} (for $i < r$) and $N+1-r$ of the u_{ji}. The remaining u_{ji} are known from the previous rows. The first row gives $u_{1i} = a_{1i}$. Starting from this the LU decomposition may be constructed systematically: $l_{r1} = a_{r1}/u_{11}$, $l_{r2} = (a_{r2} - l_{r1}u_{12})/u_{22}$, *etc.*. This fails only when one of the u_{ii} vanishes. If A is non-singular, then this can be avoided by pivoting.
Since the diagonal elements of L are all unity, they need not be stored. Therefore, L and U can be fitted into the same storage as A.
The error analysis for this construction is exactly the same as that for Gauss elimination.

Tridiagonal and band-diagonal matrices

Matrices with the only non-zero elements restricted to the diagonal and a few rows above and below it are called band-diagonal. If the only non-zero elements are in the diagonal and exactly one row above and below it, then the matrix is called tridiagonal. Gauss elimination with pivoting can increase the band width of these matrices.
However, if the matrices are diagonal dominated, *i.e.*, $|a_{ii}| > |a_{ij}$ for all $j \neq i$, then there are simpler methods to solve the equations.

Problem 7: Find an iterative algorithm to solve tridiagonal systems of equations.

Iterative improvement of solutions

On a computer with machine precision $\epsilon_m = 10^{-p}$, if one solves a set of equations with condition number $\kappa(A) = \mathcal{O}(10^k)$ with $k < p$, then one has a solution where the last k digits are essentially random. One could try to recover this precision iteratively.

Define the residual $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ where $\mathbf{x}$ is the solution that has been obtained. Since $\mathbf{x}$ has only $p - k$ significant digits, $\mathbf{r}$ has k significant non-zero digits. Now solve for $\mathbf{y}$ in $A\mathbf{y} = \mathbf{r}$. If an LU decomposition had been performed, then this cost is small. By the same logic as before, the last k digits of $\mathbf{y}$ could be undetermined. However, since $\mathbf{r}$ was 10^{-k} times smaller than $\mathbf{b}$, the new residual error in $\mathbf{r} - A\mathbf{y}$ is 10^{-2k} times smaller. Hence $\mathbf{x} + \mathbf{y}$ has higher precision.

We have assumed that A was equilibrated, so there was no loss of precision in storing it. The only loss of precision came from inverting it.

Outline

1. Linear problems
   - Motivation
   - Norms
   - Computing dot products: introduction to parallel computing
2. Solving linear equations
   - Gauss Elimination
   - LU decomposition
   - Odds and ends
3. References

References and further reading

1. The Algebraic Eigenvalue Problem, by J. H. Wilkinson, Clarendon Press, 1965. This is *the* book on linear algebra: with detailed theory, very good explanations, and, most of all, beautifully constructed examples. Very highly recommended as a book you should read through carefully at some time in your life, preferably during the course.
2. Numerical Recipes, by W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Cambridge University Press. Treat this as a well written manual for a ready source of building blocks. However, be ready to go into the blocks to improve their performance.
3. E. H. Neville, "Iterative interpolation", *Journal of the Indian Mathematical Society*, Vol 20, p. 87 (1934).