High Performance Linear Algebra I

M. D. Jones, Ph.D.

Center for Computational Research
University at Buffalo
State University of New York

High Performance Computing I, 2013

Computer Algebra Recapitulated

Consider an old friend,

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

where **A** is an $M \times N$ matrix, and (therefore) **x** and **b** are vectors of length N and M, respectively.

- Otherwise known as M equations for N unknowns
- Solution fails analytically when:
  - $M < N$ (underdetermined)
  - linear dependence exists (i.e. some row or column is a linear combination of the others), i.e. **A** is singular.
  - $M > N$ (overdetermined), seek "best fit" solution (linear least squares)

Potential Numerical Pitfalls

Eq. 1 has some special pitfalls numerically:

- *Near* linear dependence exists, in which the machine precision is unable to distinguish the linear independence
- Roundoff errors accumulate in an uncontrolled fashion, which usually leads to nonsensical results
  (Prime motivator for using higher precision arithmetic)

Numerical analysts have produced copious research in the above, and much of that work is incorporated into the various available algebra packages

Gauss-Jordan elimination is the 'standard' technique for solving linear systems by inverting the (square, non-singular) matrix **A**:

- Solution is unaffected by interchange of any two rows of **A**
- Solution is unaffected by replacing any row by a linear combination of other rows

Of course, in these operations we also have to keep track of the corresponding permutations in **x** and **b**

Augmentation with the identity matrix yields $\mathbf{A}^{-1}$,

$$\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1N} & 1 & 0 & \dots & 0 \\ A_{21} & A_{22} & \dots & A_{2N} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \dots & A_{NN} & 0 & 0 & \dots & 1 \end{pmatrix}$$

or you can augment with (a set of) solution vectors to solve for $\mathbf{x}$

$$\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1N} & b_{11} & b_{12} \\ A_{21} & A_{22} & \dots & A_{2N} & b_{21} & b_{22} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ A_{N1} & A_{N2} & \dots & A_{NN} & b_{N1} & b_{N2} \end{pmatrix}$$

Rearranging the rows of **A** to yield the identity matrix finds the solution (simultaneously $\mathbf{A}^{-1}$).

Attributes of Gauss-Jordan

Prominent features of Gauss-Jordan elimination:

- Simple to implement
- Stable (when used with pivoting, which we discuss next)
- More expensive when you do not need $\mathbf{A}^{-1}$
- Using $\mathbf{A}^{-1}$ for additional solution vectors can be unstable

Instability

Consider the following simple algorithm for the first steps in Gauss-Jordan elimination:

```
do i=1,N-1
   do j=i+1,N-1
      mult=A(j,i)/A(i,i)
      do k=i,N-1
         A(j,k) = A(j,k) - mult*A(i,k)
      end do
   end do
end do
```

See the problem? In this context $A(i,i)$ is called the **pivot** element.

Pivoting

We can use **pivoting** to eliminate the instability in Gauss-Jordan (and later plain Gaussian) elimination

- Always unstable without pivoting
- **Partial pivoting**, exchange rows to get desirable (large) value for pivot
- **Full pivoting**, exchange rows and columns to get desirable (large) value for pivot
- **Implicit pivoting**, use as pivots those elements that would have been largest if original equations were scaled to have unity as largest elements

Halfway There

Gauss-Jordan is sometimes called a **full elimination** scheme, i.e. the original matrix is (fully) reduced to the identity. Gaussian elimination (with or without pivoting) stops halfway:

$$\begin{pmatrix} A'_{11} & A'_{12} & \ldots & A'_{1N} & b'_1 \\ 0 & A'_{22} & \ldots & A'_{2N} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & A'_{NN} & b'_N \end{pmatrix}$$

when we have a **triangular** matrix,

and then proceeds through **backsubstitution** to obtain the solution

$$x_i = \frac{1}{A'_{ii}} \left(b'_i - \sum_{j=i+1}^{N} A'_{ij} x_j \right),$$

where *i* goes from *N* down to 1.

Gauss, or Gauss-Jordan?

So which technique, Gaussian or Gauss-Jordan elimination is preferable, computationally?

- Even if you do compute the inverse matrix, Gaussian elimination has a slight advantage in operation count ($N^3/3$ versus N^3, plus $N^2/2$ versus N^2 for each right-hand side).
- Big drawback for both is that all right-hand side vectors must be solved at the time of elimination
- Neither - the next method, **LU factorization** is just as efficient, without the drawbacks

Triangular Factors

Suppose that we can write **A** as a product of a lower triangular matrix, **L** and an upper triangular matrix, **U** (we can if **A** is invertible and its principal minors are non-zero),

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U},$$

then our original matrix equation becomes

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b}, \\ \mathbf{L} \cdot \mathbf{Ux} &= \mathbf{b}, \end{aligned}$$

and then we can arrange as two coupled equations:

$$\begin{aligned} \mathbf{Ly} &= \mathbf{b}, \\ \mathbf{Ux} &= \mathbf{y}. \end{aligned}$$

Given the triangular nature of these matrices, we can write the solutions as:

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

known as **forward substitution** and

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

as the **backward substitution** step.

LU Code

LU code fragment:

```
for (k=1;k<N;k++) {
   for (i=k+1;i<=N;i++) {
       L(i,k) = A(i,k)/A(k,k)
   }
   for (j=k+1;j<=N;j++) {
      for (i=k+1;i<=N;i++) {
          A(i,j) = A(i,j) - L(i,k)*A(k,j)
      }
   }
}
```

Note that the inner loops (over *i*) have no dependencies, i.e. they can more easily be executed in parallel.

Singular Value Decomposition

- Useful when Gaussian elimination and LU decomposition fails
- Diagnose problematic matrices, and often can yield useful numerical results
- Also useful for nonlinear least-squares problems, although that is a separate topic

Foundations of SVD

SVD is based on a linear algebraic theorem:

Theorem

*Any $M \times N$ matrix **A** with $M \geq N$ can be written as the product of a $M \times N$ column orthogonal matrix **U**, a $N \times N$ diagonal matrix **w** with nonnegative elements, and the transpose of an $N \times N$ orthogonal matrix **V**:*

$$\boldsymbol{A} = \boldsymbol{U} \cdot \boldsymbol{w} \cdot \boldsymbol{V}^T. \tag{1}$$

The column-wise orthogonality of **U** and **V** simply means

$$\sum_{i=1}^{M} U_{ij} U_{ik} = \delta_{jk},$$
$$\sum_{i=1}^{N} V_{ij} V_{ik} = \delta_{jk}.$$

SVD For Square Matrices

- For $M = N$, it is “easy” to invert the orthogonal matrices **U** and **V** (just take the transpose) and show that

$$\mathbf{A}^{-1} = \mathbf{V} \cdot \left[\mathrm{diag}(1/w_j)\right] \cdot \mathbf{U}^{\mathrm{T}}$$

- If some of the w_j are quite small (or even zero), that is as indication that **A** is **ill-conditioned.**
- The condition number of a matrix is simply given by

$$\mathrm{cond}(\mathbf{A}) = \frac{\max(w_j)}{\min(w_j)}.$$

- Condition numbers approaching machine preicision (10^6 single, 10^{12} double) indicate ill-condition **A**

- Columns of **U** whose corresponding values w_j are nonzero form an orthonormal **basis** (i.e. span) the range of **A**
- Columns of **V** whose corresponding values w_j are zero form an orthonormal **basis** (i.e. span) the nullspace of **A**

Iterative Improvements to “Poor” Solutions

Sometimes you have a nearly singular matrix that leads to roundoff errors even under the best of circumstances. This can happen for large matrices as well. But all hope is not lost, as you can apply the following trick to improve the quality of an existing solution.
Start with

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

but suppose that $\mathbf{x}$ is the *exact* solution, and ours is just a bit off, $\mathbf{x} + \delta\mathbf{x}$.

So our poor solution satisfies

$$\mathbf{A} \cdot (\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b},$$

which also implies

$$\mathbf{A} \cdot \delta\mathbf{x} = \delta\mathbf{b},$$

but substituting for $\delta\mathbf{b}$ gives:

$$\mathbf{A} \cdot \delta\mathbf{x} = \mathbf{A} \cdot (\mathbf{x} + \delta\mathbf{x}) - \mathbf{b}, \tag{2}$$

but we know the right-hand side of Eq. 2 - it is the “residual” of the problem we originally solved, and now want to improve. So solving Eq. 2 for $\delta\mathbf{x}$ is as simple as backsubstituting the right-hand side using our original $\mathbf{A}$ (if LU decomposed). Rinse and repeat!

Iterative Improvement Scheme

The iterative improvement scheme:

```
• Solve for residual, $\mathbf{r} = -(\mathbf{Ax} - \mathbf{b})$,
• while $(\text{norm}(\mathbf{r}) > \epsilon)$ do
          Solve $\mathbf{A}\delta\mathbf{x} = \mathbf{r}$
          Let $\mathbf{x} = \mathbf{x} + \delta\mathbf{x}$
          $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$
• end while
```

When the matrix **A** is already LU decomposed, solving for $\delta\mathbf{x}$ is relatively straightforward.