Matrix Algorithms (15)

Michael Littman

November 17th, 1998

1 MATRIX APPLICATIONS

1.1 Scientific Computing

Use computers to simulate the real world:

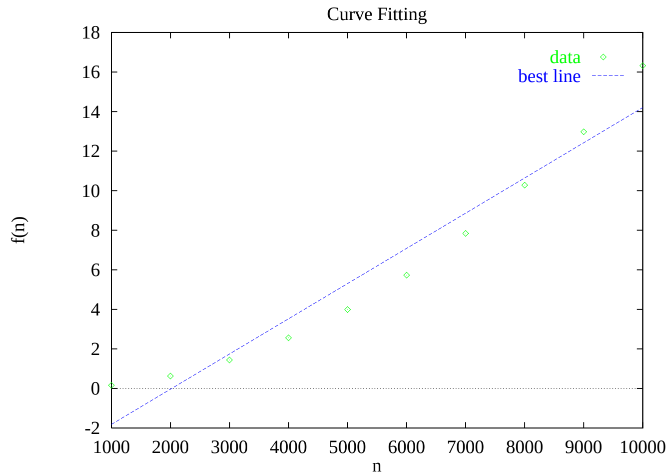
- Semiconductor process
- Aircraft/automobile design
- Electrocardiography (heart beats)

How?

1. Set up the mathematical model.
2. Parse the mathematical model into a system of linear equations.
3. **Solve the system of linear equations.**

1.2 Learning from Data

A common task in statistics, data mining, machine learning, etc. is to find a mathematical function that fits a set of data.
For example: Given temperature, wind speed, humidity, day of the year, etc., predict the strength of a hurricane.
Or: Given n, predict the running time of selection sort:



This shows the "best" linear function $(0.001779818n - 3.596)$.

2 MATRICES AND VECTORS

2.1 Definitions

Matrix:

- A 2-dimensional array of real numbers
- An m by n matrix has m rows and n columns

$$A_{m \times n} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- $A[i,j] = a_{ij}$, element at ith row and jth column.

Vector:

- A 1-dimensional array of real numbers
- A **row vector** (of size n) — a matrix with 1 row

$$u^T = [u_1 \; u_2 \; \cdots \; u_n]$$

- A **column vector** (of size m) — a matrix with 1 column

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix}$$

- By default, a vector is a **column** vector

2.2 Some Special Matrices

Square matrix: Same number of rows and columns ($n \times n$).
Identity matrix, $I_{n\times n}$:

$$I[i,j] = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{otherwise} \end{cases}$$

Diagonal matrix, $D_{n\times n}$

$$D[i,j] = 0 \quad \text{if } i \neq j$$

Tridiagonal matrices, $T_{n\times n}$

$$T[i,j] = 0 \quad \text{if } |i-j| > 1$$

Lower-triangular matrices, $L_{n\times n}$

$$L[i,j] = 0 \quad \text{if } i < j$$

Upper-triangular matrices, $U_{n\times n}$

$$U[i,j] = 0 \quad \text{if } i > j$$

Pictures...

2.3 Basic Matrix Operations

Transpose: A^T

- Algorithm: $A^T[i,j] = A[j,i]$, exchange the row and column indices.
- # operations = $\Theta(mn)$

Scalar multiplication: $c \cdot A_{m\times n}$

- Precondition: c is a real number.
- Algorithm: $(c \cdot A)[i,j] = c \cdot A[i,j]$, multiply each element of A by c.
- # operations = $\Theta(mn)$

Addition: $A_{m\times n} + B_{m\times n}$

- Precondition: A and B have the same shape.
- Algorithm: $(A+B)[i,j] = A[i,j] + B[i,j]$, add element by element.
- # operations = $\Theta(mn)$

Dot product (of vectors): $u^T v$

- Preconditions:
  - A row vector (u^T) times a column vector (v);
  - Two vectors have the same size (n).
- Algorithm: $u^T v = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n \equiv \sum_{i=1}^{n} u_i v_i$
- # operations $= \Theta(n)$

Norm (l^2-norm) of a vector: $\|u\| = \sqrt{u^T u}$.
How can we express matrix subtraction in terms of these operations?

$$A - B = A + ((-1) \cdot B).$$

2.4 Matrix Multiplication

$A_{m\times n} \cdot B_{n\times p}$:

- Precondition: # columns of A equals # rows of B.
- Algorithm: $(A \cdot B)[i, j] = A[i, 1]B[1, j] + A[i, 2]B[2, j] + \cdots + A[i, n]B[n, j]$
- Alternatively, write A as an array of **row** vectors

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix}$$

  and B as an array of **column** vectors

$$B = [b_1 \; b_2 \; \cdots \; b_p],$$

  then

$$A \cdot B = \begin{bmatrix} a_1^T b_1 & a_1^T b_2 & \cdots & a_1^T b_p \\ a_2^T b_1 & a_2^T b_2 & \cdots & a_2^T b_p \\ \vdots & \vdots & & \vdots \\ a_m^T b_1 & a_m^T b_2 & \cdots & a_m^T b_p \end{bmatrix}.$$

- # operations $= \Theta(mnp)$:
  - $A \cdot B$ involves $(m \times p)$ dot products, and
  - Each dot product needs $O(n)$ operations.

2.5 Matrix Algebra

For all matrices $A_{m\times n}$, $B_{m\times n}$ and scalars c:

- $c(A+B) = cA + cB$
- $A + B = B + A$
- $(A^T)^T = A$
- $A - A = 0$
- $(A+B)^T = A^T + B^T$

In addition, for all matrices $C_{p\times m}$:

- $(AB)^T = B^T A^T$
- $C(A+B) = CA + CB$

However, $AB \neq BA$ in general (even if both are square):

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 4 \\ 24 & 10 \end{bmatrix},$$

but

$$\begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 10 & 16 \\ 6 & 10 \end{bmatrix}.$$

2.6 Solving Equations

Say A and C are $m \times n$ and b and x are n-vectors. Solve for vector x:

$$\begin{aligned} (C^T A)^T b + x &= b - 2x \\ A^T C b + x &= b - 2x \\ x + 2x &= b - A^T C b \\ 3x &= b - A^T C b \\ x &= 1/3(b - A^T C b) \end{aligned}$$

Can solve lots of equations this way. What about $Ax = b$? Need a notion of matrix inverse.

3 SOLVING LINEAR SYSTEMS

3.1 Gaussian Elimination

Basic form: $A \cdot x = b$.
An example:

$$A = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix},$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix},$$

$$b = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}.$$

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 12 \\ 12x_1 - 8x_2 + 6x_3 + 10x_4 = 34 \\ 3x_1 - 13x_2 + 9x_3 + 3x_4 = 27 \\ -6x_1 + 4x_2 + 1x_3 - 18x_4 = -38 \end{cases}$$

1. Eliminate x_1 in equations 2,3 and 4

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 12 \\ - 4x_2 + 2x_3 + 2x_4 = 10 \\ - 12x_2 + 8x_3 + x_4 = 21 \\ 2x_2 + 3x_3 - 14x_4 = -26 \end{cases}$$

2. Eliminate x_2 in equations 3 and 4

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 12 \\ - 4x_2 + 2x_3 + 2x_4 = 10 \\ 2x_3 - 5x_4 = -9 \\ 4x_3 - 13x_4 = -21 \end{cases}$$

3. Eliminate x_3 in equation 4

$$\begin{cases} 6x_1 - 2x_2 + 2x_3 + 4x_4 = 12 \\ - 4x_2 + 2x_3 + 2x_4 = 10 \\ 2x_3 - 5x_4 = -9 \\ - 3x_4 = -3 \end{cases}$$

4. Solve the above system starting from the last equation

$$\begin{cases} x_1 & = & 1 \\ x_2 & = & -3 \\ x_3 & = & -2 \\ x_4 & = & 1 \end{cases}$$

3.2 Expressing the Algorithm

There is a more systematic way of describing the Gaussian elimination process.

- Easy-to-solve systems
  - A is a *lower-triangular* matrix

$$\begin{bmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}.$$

It can be solved as

$$\begin{aligned} x_1 &= b_1/a_{11}, \\ x_2 &= (b_2 - a_{21}x_1)/a_{22}, \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33}, \\ \vdots \quad & \quad \vdots \\ x_n &= (b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n\,n-1}x_{n-1})/a_{nn}. \end{aligned}$$

This is called the **forward substitution**.

  - A is an *upper-triangular* matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}.$$

It can be solved as

$$\begin{aligned} x_n &= b_n/a_{nn}, \\ x_{n-1} &= (b_{n-1} - a_{n-1,n}x_n)/a_{n-1,n-1}, \\ x_{n-2} &= (b_{n-2} - a_{n-2,n}x_n - a_{n-2,n-1}x_{n-1})/a_{n-2,n-2}, \\ \vdots \quad & \quad \vdots \\ x_1 &= (b_1 - a_{1n}x_n - a_{1,n-1}x_{n-1} - a_{1,n-2}x_{n-2} \cdots - a_{12}x_2)/a_{11}. \end{aligned}$$

This is called the **back substitution**

- Constraint: $a_{11}, a_{22}, \ldots, a_{nn} \neq 0$
- \# operations for both cases — $\Theta(n^2)$.

- Strategy
  If we can write $A = LU$, with L lower-triangular and U upper-triangular, then what we need to do is

  1. Solve z in $Lz = b$, then
  2. Solve x in $Ux = z$

3.3 LU-factorization

- Algorithm

  1. If $a_{11} = 0$, fail!
  2. If $n = 1$, return $L = 1$ and $U = a_{11}$.
  3. Write A as

$$A = \left[\begin{array}{c|cccc} a_{11} & a_{12} & \cdots & a_{1n} \\ \hline a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}\right] = \begin{bmatrix} a_{11} & r^T \\ c & A' \end{bmatrix},$$

  4. Define $A_1 = A' - \dfrac{c \cdot r^T}{a_{11}}$. (Schur complement, see CLR p.755)
  5. Recursively compute the LU-factorization of A_1

$$A_1 = L_1 U_1$$

  6. Return

$$L = \begin{bmatrix} 1 & 0 \\ c/a_{11} & L_1 \end{bmatrix}, \quad U = \begin{bmatrix} a_{11} & r^T \\ 0 & U_1 \end{bmatrix}$$

- Correctness of the algorithm

  - Show $L \cdot U = A$
    1. Base case — $L \cdot U = 1 \cdot a_{11} = a_{11} = A$
    2. Otherwise —

$$\begin{aligned} L \cdot U &= \begin{bmatrix} 1 & 0 \\ c/a_{11} & L_1 \end{bmatrix} \cdot \begin{bmatrix} a_{11} & r^T \\ 0 & U_1 \end{bmatrix} \\ &= \begin{bmatrix} a_{11} & r^T \\ c & L_1 \cdot U_1 + c \cdot r^T / a_{11} \end{bmatrix} \end{aligned}$$

$$
\begin{aligned}
&= \begin{bmatrix} a_{11} & r^T \\ c & A_1 + c \cdot r^T / a_{11} \end{bmatrix} \\
&= \begin{bmatrix} a_{11} & r^T \\ c & A' \end{bmatrix} \\
&= A
\end{aligned}
$$

- Show L is lower-triangular and U is upper-triangular (Exercise)
- Termination: # recursive calls $= n-1$

- Operation count
  - Computing A_1 (Schur complement) costs $\Theta(n^2)$ operations
  - $T(n) = T(n-1) + \Theta(n^2)$
  - Solution?

    $T(n) = \Theta(n^3)$.

3.4 PLU-factorization

PLU is LU with pivoting (although "P" stands for permutation).

- Permutation matrices $P_{n\times n}$:
  - Each row or each column has exactly one 1 and 0 for the rest.
  - Example: $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
  - Property: $P^T P = PP^T = I$
- What if $a_{11} = 0$ happens in LU-factorization?
  - Want to find a permutation matrix P, such that $A = PLU$.
- Algorithm
  1. If $n = 1$, return $P = 1, L = 1, U = a_{11}$.
  2. Find a non-zero element a_{i1} in the first row of A. (If not, then the system is not solvable.)
  3. Define a permutation matrix $Q_{n\times n}$ such that
     - $q_{1i} = q_{i1} = 1$,
     - $q_{jj} = 1$ for $j \neq 1, j \neq i$,
     - All the other elements of Q are 0.
  4. Let $B = QA$. Notice that $b_{11} = a_{i1} \neq 0$.

5. Write B as

$$B = \begin{bmatrix} b_{11} & r^T \\ c & B' \end{bmatrix}.$$

6. Define $B_1 = B' - c \cdot r^T/b_{11}$. Recursively compute $B_1 = P_1 L_1 U_1$
7. Return

$$P = Q^T \begin{bmatrix} 1 & 0 \\ 0 & P_1 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 \\ P_1^T c/b_{11} & L_1 \end{bmatrix}, \quad U = \begin{bmatrix} b_{11} & r^T \\ 0 & U_1 \end{bmatrix}$$

- Correctness of the algorithm? By induction.
- Operation count? Same as LU.

3.5 Summary

For an $n \times n$ matrix A, the following are equivalent.

1. A has a PLU-factorization.
2. For any n-vector b, the system $Ax = b$ has a solution.
3. A has an inverse.
4. A has non-zero determinant.
5. A has full rank.

Read CLR Ch 31.1 for definitions of inverse, determinant and rank.

4 LEAST SQUARES

4.1 Overconstrained Systems

We discussed how to solve $Ax = b$, and it only works when A is square (same number of equations as variables).
Consider the case of finding the best line to fit a set of points. We want to find x_1 and x_2 such that $x_1 n_i + x_2 = t_i$ where n_i is problem size and t_i is time.
We can't solve this exactly (in general) if we have more than two data points (which is the only useful case anyway!).

4.2 Least Squares

So, what solution do we pick? A classic example is to find the x_1 and x_2 that give the best values in a "least squares" sense:

$$\min_{x_1,x_2} \sum_i (x_1 n_i + x_2 - t_i)^2.$$

This scoring scheme is reasonable because it is optimized for a perfect fit, and the worse the predictions, the worse the score.

Note: You can think of this as being a generalization of the notion of an average. Recall that $\min_{x_1} \sum_i (x_1 - t_i)^2 = \sum_i t_i/n$.

4.3 Least Squares in Matrices

We can also express the problem this way: Let A be a matrix with the n_i column as its first column and a column of all 1s for its second column. Let x be the column vector containing x_1 and x_2 and b be a column vector of the y_i values.

We want to find the x that minimizes $(Ax - b)^2$.

In this general form, we can also consider fitting other functions. For example, $x_1 n_i^2 + x_2 n_i + x_3 \approx t_i$ can be expressed by putting n_i^2 values in the first column of A, n_i in the second column of A, and 1s in the third column.

4.4 Minimizing The Score

Recall from calculus that a differentiable function is minimized at the point at which the derivative is zero. Thus, we want to find

$$\begin{aligned} D((Ax-b)^2) &= 0 \\ 2D(Ax-b)^T(Ax-b) &= 0 \\ A^T(Ax-b) &= 0 \\ A^T Ax &= A^T b. \end{aligned}$$

Cool! It's just another system of linear equations to solve. (Thank goodness derivatives are linear operators!)

4.5 Example

$$A = \begin{bmatrix} 1000 & 1 \\ 2000 & 1 \\ 3000 & 1 \\ 4000 & 1 \\ 5000 & 1 \\ 6000 & 1 \\ 7000 & 1 \\ 8000 & 1 \\ 9000 & 1 \\ 10000 & 1 \end{bmatrix},$$

$$b = \begin{bmatrix} 0.16 \\ 0.63 \\ 1.44 \\ 2.56 \\ 3.99 \\ 5.73 \\ 7.84 \\ 10.28 \\ 12.98 \\ 16.32 \end{bmatrix}.$$

Form $A^T Ax = A^T b$, solve for x:

$$x = \begin{bmatrix} 0.001779818 \\ -3.596 \end{bmatrix}.$$

Thus, $t \approx 0.001779818n - 3.596$ is the best fit to the data in a least squares sense.
The best quadratic is much better: $t \approx 1.67803 \times 10^{-7}n^2 - 6.601515 \times 10^{-5}n + 0.09566667$.