Matrix Operations - A Study

Surya Raghav B

June 2024

1 Solving systems of linear equations

We have a set of linear equations in n unknowns $x_1, x_2, \ldots, x_n$. We can write them in the matrix form

$$Ax = b$$

if A is nonsingular, then it has an inverse A^{-1}, and

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

We can prove that this is a unique solution as follows. Assume there are two solutions x and x', then $Ax = Ax' = b$, and let I be identity matrix,

$$\begin{aligned} x &= Ix \\ &= (A^{-1}A)x \\ &= A^{-1}(Ax) \\ &= A^{-1}(Ax') \\ &= (A^{-1}A)x' \\ &= x' \end{aligned}$$

This is the case where A is nonsingular or,equivalently, the rank of A is equal to the number n of unknowns. If rank of A is less than n then the system is underdetermined and may have infinitely many solutions. If the number of equations exceeds n, then the system is overdetermined, and there may not exist any solutions. When finding $x = A{-}1b$, we face numerical instability. So we look at LUP decomposition which is numerically stable.

1.1 Overview of LUP decomposition

We find three nxn matrices L,U, and P such that

$$PA = LU$$

where

- L is a unit lower-triangular matrix

- U is an upper-triangular matrix
- P is a permutation matrix

This involves only triangular linear systems, which is easier to solve.

$$Ax = b$$

$$PAx = Pb$$

$$LUx = Pb$$

This can be solved by solving two triangular linear systems. Define $y = Ux$ and first solve the lower-triangular system for y known as forward substitution.

$$Ly = Pb$$

Having solved for y, we then solve the upper-triangular system for known as back substitution.

$$Ux = y$$

Now we show that x is indeed the solution

$$PA = LU$$

$$A = P^{-1}LU$$

$$Ax = P^{-1}LUx$$

$$Ax = P^{-1}Ly$$

$$Ax = P^{-1}Pb$$

$$Ax = b$$

Next we show how forward and back substitution work

1.2 Forward and Back substitution

The forward substitution $Ly = Pb$ system can be solved in $\Theta(n^2)$ time. Let the Permutation P be an array $\pi[1 \ldots n]$, where $\pi[i]$ indicates that $P_{ij} = 1$ if $j = \pi[i]$ and $P_{ij} = 0$ if $j \neq \pi[i]$. This shows that multiplying P with A transforms a_{ij} to $a_{\pi[i]j}$ and multiplying P with b transforms b_i to $b_{\pi[i]}$. Since L is a unit-triangular matrix, we can write the equations as follows.

$$\begin{aligned} y_1 &= b_1 \\ l_{21}y_1 + y_2 &= b_2 \\ l_{31}y_1 + l_{32}y_2 + y_3 &= b_3 \\ &\vdots \\ l_{n1}y_1 + l_{n2}y_2 + l_{n3}y_3 + \cdots + y_n &= b_n \end{aligned}$$

In general, we substitute $y_1, y_2, \ldots, y_{i-1}$ into the ith equation to solve for y_i:

$$y_i = b_{\pi[i]} - \sum_{j=1}^{i-1} l_{ij} y_j$$

After solving for y we use back substitution to solve $Ux = y$ by working from the nth equation to the first. This also takes $\Theta(n^2)$ time.

$$\begin{aligned}
u_{11}x_1 + u_{12}x_2 + \cdots + u_{1,n-2}x_{n-2} + u_{1,n-1}x_{n-1} + u_{1n}x_n &= y_1, \\
u_{22}x_2 + \cdots + u_{2,n-2}x_{n-2} + u_{2,n-1}x_{n-1} + u_{2n}x_n &= y_2, \\
&\vdots \\
u_{n-2,n-2}x_{n-2} + u_{n-2,n-1}x_{n-1} + u_{n-2,n}x_n &= y_{n-2}, \\
u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= y_{n-1}, \\
u_{n,n}x_n &= y_n.
\end{aligned}$$

In general we can solve for x_i using $x_n, x_{n-1}, \ldots, x_{i-1}$ using,

$$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) / u_{ii}$$

Algorithm 1 LUP-SOLVE(L, U, P, b)

1: $n = L$.rows
2: let x be a new vector of length n
3: **for** $i = 1$ **to** n **do**
4: $\quad y_i = b_{\pi(i)} - \sum_{j=1}^{i-1} l_{ij} y_j$
5: **end for**
6: **for** $i = n$ **downto** 1 **do**
7: $\quad x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) / u_{ii}$
8: **end for**
9: **return** x

The run-time of LUP-SOLVE is $\Theta(n^2)$

1.3 Computing an LU decomposition

We show a LU decomposition where $P = I_n$. We use **Gaussian elimination**. To get U we form a upper triangle matrix by subsequent elimination of the variables. The matrix is made of the row multiplier that cause the elimination.

We perform a recursive procedure. Break A into four parts

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$
$$= \begin{pmatrix} a_{11} & w^T \\ v & A' \end{pmatrix}$$

where v and w are column vectors and A' is a square matrix all of sizes $(n-1)$. We can further factor A as

$$A = \begin{pmatrix} a_{11} & w^T \\ v & A' \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{pmatrix}$$

The resulting $(n-1)x(n-1)$ matrix

$$A' - vw^T/a_{11}$$

is called the **Schur complement** of A with respect to a_{11}. We claim that the Schur complement is also nonsingular. Suppose to the contrary, let Schur complement be singular, then it has row rank strictly less than $n-1$. Now in the matrix A in the form

$$\begin{pmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{pmatrix}$$

has 0s in the bottom n-1 rows, thus the bottom n-1 rows have row rank strictly less than n-1. The the total matrix must have row rank strictly less than n. This contradicts that A in nonsingular.

Since the Schur complement is nonsingular , we can recursively find an LU decomposition for it, say

$$A' - vw^T/a_{11} = L'U'$$

Then we have

$$A = \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & L'U' \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 \\ v/a_{11} & L' \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & U' \end{pmatrix}$$
$$= LU.$$

The elements that we divide by are called **pivots**. In LU decomposition they can be zero. So we include a Permutation P matrix to avoid dividing by zero called **pivoting** in LUP decomposition.

LU decomposition always for symmetric positive-definite matrices. We give a tail-recursive procedure for this decomposition.

Algorithm 2 LU Decomposition

1: $n = A.rows$
2: let L and U be new nxn matrices.
3: Initialize U with 0s below the diagonal
4: Initialize L with 1s on the diagonal and 0s above the diagonal
5: **for** $k = 1$ TO n **do**
6: $u_{kk} = a_{kk}$
7: **for** $i = k + 1$ TO n **do**
8: $l_{ik} = \frac{a_{ik}}{u_{kk}}$ $\{l_{ik}$ HOLDS $\ell_i\}$
9: $u_{ki} = a_{ki}$ $\{u_{ki}$ HOLDS $w_i^T\}$
10: **end for**
11: **for** $i = k + 1$ TO n **do**
12: **for** $j = k + 1$ TO n **do**
13: $a_{ij} = a_{ij} - l_{ik} \cdot u_{kj}$
14: **end for**
15: **end for**
16: **end for**
17: **Return** L AND U

We can perform a standardization by storing significant elements of L and U in place in the matrix A.

1.4 Computing an LUP decomposition

Here we use a permutation matrix P to overcome the division by zero problem for non-symmetric-positive definite matrices. Before partitioning for the LU decomposition, we move a nonzero element with the greatest absolute value, say a_{k1} to $(1, 1)$. This is similar to multiplying A with a permutation matrix Q.

$$QA = \begin{pmatrix} a_{k1} & w^T \\ v & A' \end{pmatrix}$$

where $a_{k1} \neq 0$, so we can now divide by a_{k1}.

$$\begin{aligned} QA &= \begin{pmatrix} a_{k1} & w^T \\ v & A' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix} \end{aligned}$$

Similar to LU decomposition the Schur complement can be recursively used to find an LUP decomposition.

$$P'(A' - vw^T/a_{k1}) = L'U'$$

where the final P matrix can be recursively be defined as,

$$P = \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} Q$$

$$\begin{aligned}
PA &= \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} QA \\
&= \begin{pmatrix} 1 & 0 \\ 0 & P' \end{pmatrix} \begin{pmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & P' \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & A' - vw^T/a_{k1} \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & P'(A' - vw^T/a_{k1}) \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & L'U' \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ P'v/a_{k1} & L' \end{pmatrix} \begin{pmatrix} a_{k1} & w^T \\ 0 & U' \end{pmatrix} \\
&= LU.
\end{aligned}$$

In the below algorithm we the permutation matrix P as an array π where $\pi[i] = j$, We also use in place treatment to store the values of L and U. This algorithm has three nested loops thus results in a running time of $\Theta(n^3)$

Algorithm 3 LUP Decomposition

```
1: n = A.rows
2: let π[1...n] be a new array
3: for i = 1 to n do
4:    π[i] = i
5: end for
6: for k = 1 to n do
7:    p = 0
8:    for i = k to n do
9:       if |a_ik| > p then
10:         p = |a_ik|
11:         k' = i
12:      end if
13:   end for
14:   if p == 0 then
15:      error "singular matrix"
16:   end if
17:   exchange π[k] with π[k']
18:   for i = 1 to n do
19:      exchange a_ki with a_k'i
20:   end for
21:   for i = k + 1 to n do
22:      a_ik = a_ik / a_kk
23:      for j = k + 1 to n do
24:         a_ij = a_ij − a_ik a_kj
25:      end for
26:   end for
27: end for
```

2 Inverting matrices

We show in this section the matrix multiplication and computing inverses are equally hard problems. Thus Strassen's algorithms can be used to find inverses.

2.1 Computing Inverse from an LUP decomposition

The LUP decomposition only depends on A in the equation $Ax = b$, and solving using the LUP decomposition takes $\Theta(n^2)$ time. Thus the equation with X as the inverse of A

$$AX = I_n$$

can be split into n distinct set of linear equations of the form

$$AX_i = e_i$$

where X_i and e_i denote the ith column of X and I respectively. Here the LUP decomposition takes $\Theta(n^3)$ time. Finding one column of X takes $\Theta(n^2)$ time, so in total this algorithm takes $\Theta(n^3)$ time to calculate the inverse of A.

2.2 Matrix multiplication and inversion

We show that matrix inversion and matrix multiplication are equivalent via two theorems.

Theorem 2.1 (Multiplication is no harder than inversion) *If we can invert a matrix in time $I(n)$, and $I(3n) = O(I(n))$, the we can multiply tow matrices in time $O(I(n))$.*

Proof. Let C be the product of A and B, define a 3nx3n matrix D by

$$D = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix}$$

The inverse of D is

$$D^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix}$$

Here the upper-right nxn matrix contains the product of A and B. We can construct matrix D in $\Theta(n^2)$ D can be constructed in $\Theta(n^2)$ time, which is $O(I(n))$, and it can be inverted in $O(I(3n)) = O(I(n))$ time.

Theorem 2.2 (Inversion is no harder than multiplication) *If we can multiply two matrices in $M(n)$ time, and $M(n + k) = O(M(n))$ for k in range $0 \leq k \leq n$ and $M(n/2) \leq cM(n)$ for some $c \leq 1/2$. Then we can compute the inverse in $O(M(n))$ time.*