Linear System

Fabio Cannizzo

Linear Systems

- Solution of a system of linear equations
- Let's focus on the case where the number of equations is equal to the number of unknowns

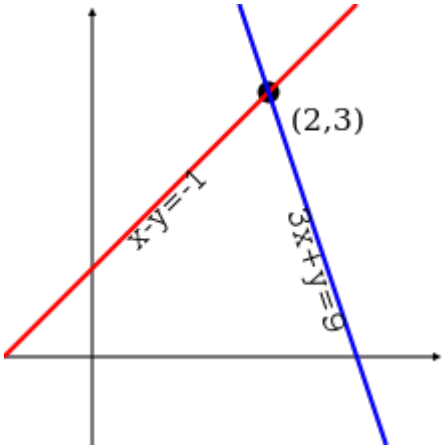
$$a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = b_1$$

$$\ldots$$

$$a_{n,1}x_1 + a_{n,2}x_2 + \ldots + a_{n,n}x_n = b_n$$

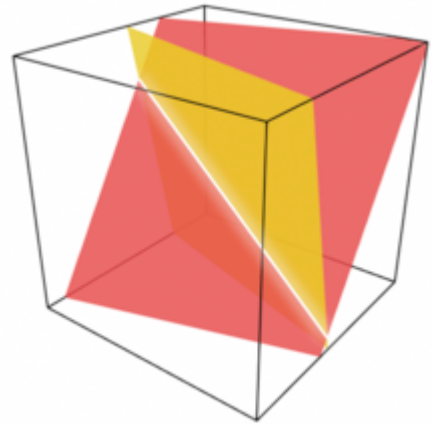
- Theory states that if the equations are linearly independent a solution exists and is unique

Geometric Interpretation



2 equations in 2 unknowns

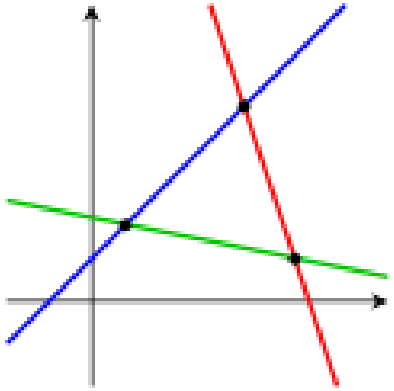
solution is a unique point



2 equations in 3 unknowns

solution is a line

indetermined

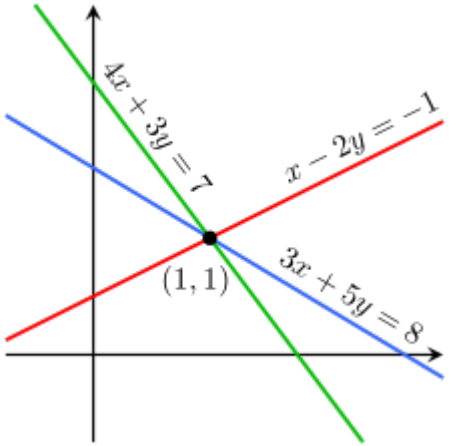


3 equations in 2 unknowns

solution does not exists, unless some equations are linearly dependent

over-determined

Geometric Interpretation

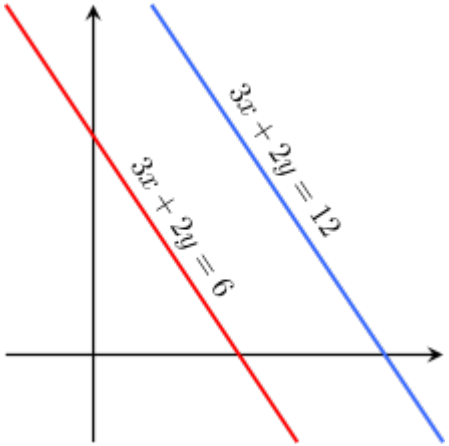


3 equations in 2 unknowns, but not linearly independent

rank of the system is 2

solution is unique

redundant



2 equations in 2 unknowns, but not linearly independent

rank of the system is 1

no solution

not consistent

Linear Algebra Review

- Determinant of a matrix: a unique number associated with a square matrix

$$\det(A) = \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i} \quad \sigma = permutation([1,2,...,n])$$

- Rank(B): dimension of the largest square matrix M which we can extract from B such that det(M) ≠ 0
- A is inversible if there exist A^{-1} such that: $A^{-1}A=I$
- A is inversible ⟷ det(A) ≠ 0

Linear Algebra Review

- $(AB)^T = B^T A^T$
- A linear system Ax=b has a unique solution ⬅➡ rank(A) = rank([A,b])
- A is singular ⬅➡ det(A)=0
- A is diagonal or triangular ➔ det(A) is the product of the elements on the main diagonal
- $Ax=b \rightarrow A^{-1}Ax=A^{-1}b \rightarrow I\,x = A^{-1}b \rightarrow x = A^{-1}b$

Linear Algebra Review

- A symmetric ⇐⇒ $a(i,j)=a(j,i)$
- A symmetric and positive definite

  $x^T Ax > 0$ for any vector $x \neq 0$
- A symmetric and semi-positive definite

  $x^T Ax \geq 0$ for any vector $x \neq 0$

Linear Systems

- In matrix or vector form the linear system can be written as:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ \cdots & & & \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \qquad Ax = b$$

$$\begin{bmatrix} a_{1,1} \\ \vdots \\ a_{n,1} \end{bmatrix} x_1 + \begin{bmatrix} a_{1,2} \\ \vdots \\ a_{n,2} \end{bmatrix} x_2 + \ldots + \begin{bmatrix} a_{n,1} \\ \vdots \\ a_{n,n} \end{bmatrix} x_n = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \qquad A_{:,1}x_1 + A_{:,2}x_2 + \ldots + A_{:,n}x_n = b$$

Linear Systems

- Let's focus on system of n equations with n unknowns and A matrix of full rank
- At school we learned how to solve a linear system
  - symbolically via variable elimination
  - linearly combining equations to eliminate variables
- Let's see how this can translate into a computer algorithm

Diagonal Case

- If the matrix A is diagonal, solution of the system is trivial

$$\begin{bmatrix} a_{1,1} & & \\ & \ddots & \\ & & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \qquad x_i = \frac{b_i}{a_{i,i}} \qquad i = 1,2,\ldots,n$$

- a_{ii} must be all different from zero, otherwise A would be singular
- Number of operations is n ➔ O(n)

Triangular Case

- If the matrix A is **lower** triangular, solution of the system is trivial via **forward** substitution

$$\begin{bmatrix} a_{1,1} & & \\ \vdots & \ddots & \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \qquad x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j} x_j}{a_{i,i}} \qquad i = 1,2,\ldots,n$$

- a_{ii} must be all different from zero, otherwise A would be singular
- Number of operations is $n+n(n-1)$ ➔ $O(n^2)$

Triangular Case

- If the matrix A is **upper** triangular, solution of the system is trivial via **backward** substitution

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ & \ddots & \vdots \\ & & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix},$$

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i,j} x_j}{a_{i,i}} \qquad i = n, n-1, \ldots, 1$$

- a_{ii} must be all different from zero, otherwise A would be singular
- Number of operations is $n+n(n-1)$ ➔ $O(n^2)$

General Case

- Cramer is too slow: $O(n^4)$
- We could computing the inverse A^{-1}, then multiply by b. This is affected by rounding errors, so linear system are solved directly, without computing the inverse
- At the contrary, the inverse is obtained via solving linear systems

Computing the Inverse

$$AA^{-1} = I \quad \Rightarrow \quad \begin{cases} AA^{-1}_{:,1} = e_1 \\ AA^{-1}_{:,2} = e_2 \\ \vdots \\ AA^{-1}_{:,n} = e_n \end{cases}, \qquad e_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \ \ldots$$

$$AX = I$$

- Assuming we have a method to solve a linear system Ax=b, we can compute the inverse solving the system Ax=b n times, each time with a different known term $b=e_i$

Gauss Elimination

- Gauss elimination uses linear combinations of equations to reduce the system to reduce the matrix of coefficients to upper triangular form, then is uses back substitution

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} - \cancel{\frac{a_{2,1}}{a_{1,1}} a_{1,1}} & a_{2,2} - \frac{a_{2,1}}{a_{1,1}} a_{1,2} & a_{2,3} - \frac{a_{2,1}}{a_{1,1}} a_{1,3} \\ a_{3,1} - \cancel{\frac{a_{3,1}}{a_{1,1}} a_{1,1}} & a_{3,2} - \frac{a_{3,1}}{a_{1,1}} a_{1,2} & a_{3,3} - \frac{a_{3,1}}{a_{1,1}} a_{1,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 - \frac{a_{2,1}}{a_{1,1}} b_1 \\ b_2 - \frac{a_{3,1}}{a_{1,1}} b_1 \end{bmatrix}$$

Gauss Elimination

- Cost of Gauss elimination is $O(n^3/3)$
- It must be performed at the same time on A and b.
- If I want to solve multiple rhs, they all need to be treated at the same time

Gauss Elimination with Pivoting

- What happens if the a_{11} is zero?
- Problem could actually occur also if a_{11} is not zero, but relatively small compared to the other numbers in the matrix
- We can use pivoting:
  - At every step j, choose the largest element in absolute value in the remaining rows, then swap the rows (**partial pivoting**)
  - At every step j, choose the largest element in absolute value in the whole remaining sub-matrix, then swap rows and columns (**full pivoting**). Note that swapping columns is equivalent to a permutation in the vector x, so we need to keep track of that!

LU Decomposition

- If A can be transformed in the product of a lower triangular matrix L and an upper triangular matrix U, then solution is trivial

$$Ax = b$$

$$LUx = b$$

let $z = Ux,$ (z is a vector)

$Lz = b,$ compute z via forward substitution

$Ux = z,$ compute z via backward substitution

- Computational cost: 2*TriangularCost, i.e. $O(n^2)$
- We have not considered the cost of obtaining L and U

LU Decomposition

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

- We have n^2 equations in n^2+n unknowns
- The problem is under-determined. By convention, we arbitrarily fix all elements of the diagonal of L to be 1
- The 1st equation is trivial: $1 \cdot u_{11} = a_{11}$
- The 2nd equation is also easy: $l_{21} \cdot u_{11} = a_{21}$

LU Decomposition

partition A, L, U as block matrices:

$$A = \begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \qquad L = \begin{bmatrix} 1 & 0 \\ L_{21} & L_{22} \end{bmatrix}, \qquad U = \begin{bmatrix} u_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

- a_{11} and u_{11} are scalars
- L_{22} unit lower-triangular, U_{22} upper triangular of order $n-1$

determine L and U from $A = LU$, *i.e.*,

$$\begin{aligned} \begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \\ &= \begin{bmatrix} u_{11} & U_{12} \\ u_{11}L_{21} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix} \end{aligned}$$

LU Decomposition

recursive algorithm:

- determine first row of U and first column of L

$$u_{11} = a_{11}, \qquad U_{12} = A_{12}, \qquad L_{21} = (1/a_{11})A_{21}$$

- factor the $(n-1) \times (n-1)$-matrix $A_{22} - L_{21}U_{12}$ as

$$A_{22} - L_{21}U_{12} = L_{22}U_{22}$$

this is an LU factorization (without pivoting) of order $n-1$

cost: $(2/3)n^3$ flops (no proof)

LU Decomposition

- Decomposition routines usually return the result in the storage space previously used by A
- Advantage over Gauss elimination is that we can reuse the obtained factorization, i.e. we do not need to process *b* together with *A*
- It suffers of the same stability issues as Gauss elimination, so pivoting is necessary (only partial pivoting can be carried out efficiently)
- On banded matrices, its computation cost is lower and the result obtained has the same band shape as the original matrix A (very fast for tridiagonal)

Cholesky Decomposition

- A decomposition of type $A=PP^T$, with P lower triangular
- It only exists if A is SPD
- We will look at it in the context of Monte Carlo
- Similar decomposition is $A=LDL^T$, with L lower triangular and D diagonal

Iterative Methods

Given a square system of n linear equations with unknown $\mathbf{x}$:

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

where:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Then A can be decomposed into a diagonal component D, and strictly lower and upper triangular components L and U:

$$A = D + L + U,$$

where

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

Gauss Seidel

$$Ax = b$$

$$\left(L + D + U\right)x = b$$

$$\left(L + D\right)x = b - Ux$$

$$x = \left(L + D\right)^{-1}\left(b - Ux\right)$$

$$x^{(k+1)} = \left(L + D\right)^{-1}\left(b - Ux^{(k)}\right) = \left(L + D\right)^{-1}b - \left(L + D\right)^{-1}Ux^{(k)}$$

- It is a **fixed point** algorithm
- Sufficient conditions (not necessary) for Gauss Seidel to converge are:
  - A positive definite
  - or A strictly diagonally dominant

Successive Over Relaxation

$$Ax = b$$

$$\theta\left(L + D + U\right)x = \theta b$$

$$\left(\theta L + \theta D + \left(1-\theta\right)D - \left(1-\theta\right)D\right)x = \theta b - \theta U x$$

$$x = \left(\theta L + D\right)^{-1}\left(\theta b + \left[\left(1-\theta\right)D - \theta U\right]x\right)$$

$$x^{(k+1)} = \left(\theta L + D\right)^{-1}\left(\theta b + \left[\left(1-\theta\right)D - \theta U\right]x^{(k)}\right)$$

- It is a **fixed point** algorithm
- Sufficient conditions (not necessary) for Gauss Seidel to converge are:
  - A positive definite
  - or A strictly diagonally dominant
  - $0<\theta<2$
- for $1<\theta<2$ it is faster then Gauss Seidel
- we do not know the optimal θ, can try 1.5

Newton Method

$$F(x) = Ax - b = 0$$

$$x^{(k+1)} = x^{(k)} - \left(J\left(x^{(k)}\right)\right)^{-1} F\left(x^{(k)}\right)$$

$$x^{(k+1)} = x^{(k)} - A^{-1}\left(Ax^{(k)} - b\right)$$

$$x^{(k+1)} = x^{(k)} - x^{(k)} + A^{-1}b$$

$$x^{(k+1)} = A^{-1}b$$

- Newton is a 2nd order method. It is exact for a linear problem. It converges to the exact solution in one single iteration, regardless of the initial guess
- Not very useful in practice, as it requires we compute A^{-1} !

Condition Number

- Some linear systems are ill-conditioned
- Small errors in the inputs will cause large error in the outputs
- The condition number is a quantitative measure of “how bad” the situation is
- It can be computed from the matrix A

Sparse Matrices

- Sometimes we deal with large matrices which contains lot of zeros.
- It is possible to store then in more compact formats, skipping the zero elements
- There are several of such formats (e.g., see LAPACK user guide)
- There are algorithms optimized to work on sparse matrices

LAPACK User Guide Extract

Band Storage

An ***m***-by-***n*** band matrix with ***kl*** subdiagonals and ***ku*** superdiagonals may be stored compactly in a two-dimensional array with ***kl***+***ku***+**1** rows and ***n*** columns. Columns of the matrix are stored in corresponding columns of the array, and diagonals of the matrix are stored in rows of the array. This storage scheme should be used in practice only if $kl, ku \ll \min(m, n)$, although LAPACK routines work correctly for all values of ***kl*** and ***ku***. In LAPACK, arrays that hold matrices in band storage have names ending in 'B'.

To be precise, a_{ij} is stored in AB(***ku***+**1**+*i*-*j*,*j*) for $\max(1, j - ku) \le i \le \min(m, j + kl)$. For example, when ***m*** = ***n*** = **5**, ***kl*** = **2** and ***ku*** = **1**:

Band matrix A	Band storage in array AB
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

The elements marked $*$ in the upper left and lower right corners of the array AB need not be set, and are not referenced by LAPACK routines.

Many More

- The field of Linear Algebra is one of the most heavily researched in the last 100 years??
- Linear Algebra problems appears a sub-problem pretty much everywhere
- The solution of linear system is just one of the problems
  - Singular Value Decomposition
  - Eigensystem
  - Least squares
- There are several free source and commercial libraries BLAS, LAPACK
- Sparse matrices
  - ARPACK, TAUCS, PARDISO, SUPERLU

BLAS Quick Reference Extract

Name	Operation	Prefixes
_GEMV	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
_GBMV	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
_HEMV	$y \leftarrow \alpha Ax + \beta y$	C, Z
_HBMV	$y \leftarrow \alpha Ax + \beta y$	C, Z
_HPMV	$y \leftarrow \alpha Ax + \beta y$	C, Z
_SYMV	$y \leftarrow \alpha Ax + \beta y$	S, D
_SBMV	$y \leftarrow \alpha Ax + \beta y$	S, D
_SPMV	$y \leftarrow \alpha Ax + \beta y$	S, D
_TRMV	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
_TBMV	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z

LAPACK User Guide Extract

- Linear Equations
- Linear Least Squares (LLS) Problems
- Generalized Linear Least Squares (LSE and GLM) Problems
- Standard Eigenvalue and Singular Value Problems
  - Symmetric Eigenproblems (SEP)
  - Nonsymmetric Eigenproblems (NEP)
  - Singular Value Decomposition (SVD)
- Generalized Eigenvalue and Singular Value Problems
  - Generalized Symmetric Definite Eigenproblems (GSEP)
  - Generalized Nonsymmetric Eigenproblems (GNEP)
  - Generalized Singular Value Decomposition (GSVD)

LAPACK User Guide Extract

Table 2.2: Driver routines for linear equations

Type of matrix	Operation	Single precision		Double precision	
and storage scheme		real	complex	real	complex
general	simple driver	SGESV	CGESV	DGESV	ZGESV
	expert driver	SGESVX	CGESVX	DGESVX	ZGESVX
general band	simple driver	SGBSV	CGBSV	DGBSV	ZGBSV
	expert driver	SGBSVX	CGBSVX	DGBSVX	ZGBSVX
general tridiagonal	simple driver	SGTSV	CGTSV	DGTSV	ZGTSV
	expert driver	SGTSVX	CGTSVX	DGTSVX	ZGTSVX
symmetric/Hermitian	simple driver	SPOSV	CPOSV	DPOSV	ZPOSV
positive definite	expert driver	SPOSVX	CPOSVX	DPOSVX	ZPOSVX
symmetric/Hermitian	simple driver	SPPSV	CPPSV	DPPSV	ZPPSV
positive definite (packed storage)	expert driver	SPPSVX	CPPSVX	DPPSVX	ZPPSVX
symmetric/Hermitian	simple driver	SPBSV	CPBSV	DPBSV	ZPBSV
positive definite band	expert driver	SPBSVX	CPBSVX	DPBSVX	ZPBSVX
symmetric/Hermitian	simple driver	SPTSV	CPTSV	DPTSV	ZPTSV
positive definite tridiagonal	expert driver	SPTSVX	CPTSVX	DPTSVX	ZPTSVX
symmetric/Hermitian	simple driver	SSYSV	CHESV	DSYSV	ZHESV

LAPACK Routine Example

```
*  -- LAPACK routine (version 3.3.1) --
*  -- LAPACK is a software package provided by Univ. of Tennessee,    --
*  -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
*  -- April 2011                                                      --
*
*     .. Scalar Arguments ..
      CHARACTER          UPLO
      INTEGER            INFO, LDA, N
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * )
*     ..
*
*  Purpose
*  =======
*
*  DPOTRF computes the Cholesky factorization of a real symmetric
*  positive definite matrix A.
*
*  The factorization has the form
*     A = U**T * U,  if UPLO = 'U', or
*     A = L  * L**T,  if UPLO = 'L',
*  where U is an upper triangular matrix and L is lower triangular.
*
*  This is the block version of the algorithm, calling Level 3 BLAS.
*
*  Arguments
*  =========
*
*  UPLO    (input) CHARACTER*1
*          = 'U':  Upper triangle of A is stored;
*          = 'L':  Lower triangle of A is stored.
*
```

Further Readings

- Prof Binegar's lecture notes
  http://www.math.okstate.edu/~binegar/4513-F98/4513-l09.pdf
  http://www.math.okstate.edu/~binegar/4513-F98/4513-l10.pdf
  http://www.math.okstate.edu/~binegar/4513-F98/4513-l11.pdf
  http://www.math.okstate.edu/~binegar/4513-F98/4513-l12.pdf

- Numerical Recipes in C++

- BLAS, LAPACK Reference Guides
  http://www.netlib.org/lapack/lug/node145.html
  http://www.netlib.org/lapack/lug/

- Iterative Methods
  http://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method
  http://en.wikipedia.org/wiki/Successive_over-relaxation

- Condition number
  http://www.math.ufl.edu/~kees/ConditionNumber.pdf