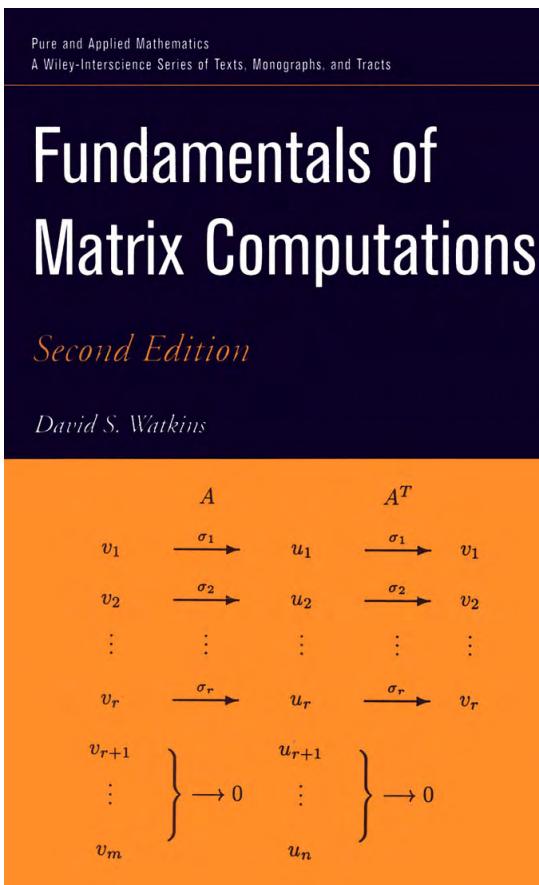
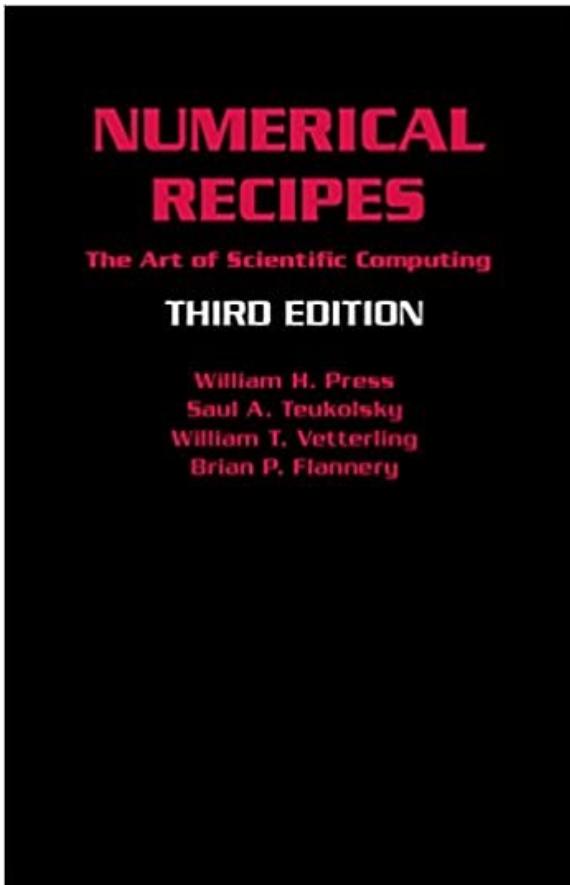


Numerical Methods for Linear Algebra

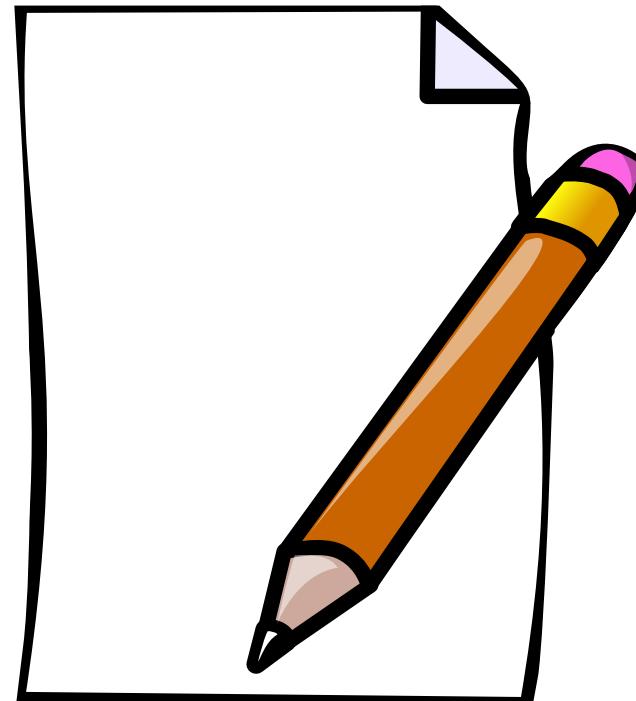
- Vector and Matrix Operations (BLAS)
- Systems of Linear Equations (LAPACK)
- [Exercise 1](#)
- Eigenvalues and Eigenvectors computation
- [Exercise 2](#)



Notes and slides: “NM4LA-1/2” on TISS as pdf files

Today

- Vector and Matrix Operations (BLAS)
- Systems of Linear Equations (LAPACK)



04.04.2022

Matrix-Vector Multiplication

$$\mathbf{A}\vec{x} = \vec{y} \iff \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \quad y_i = \sum_{j=1}^n a_{ij}x_j$$

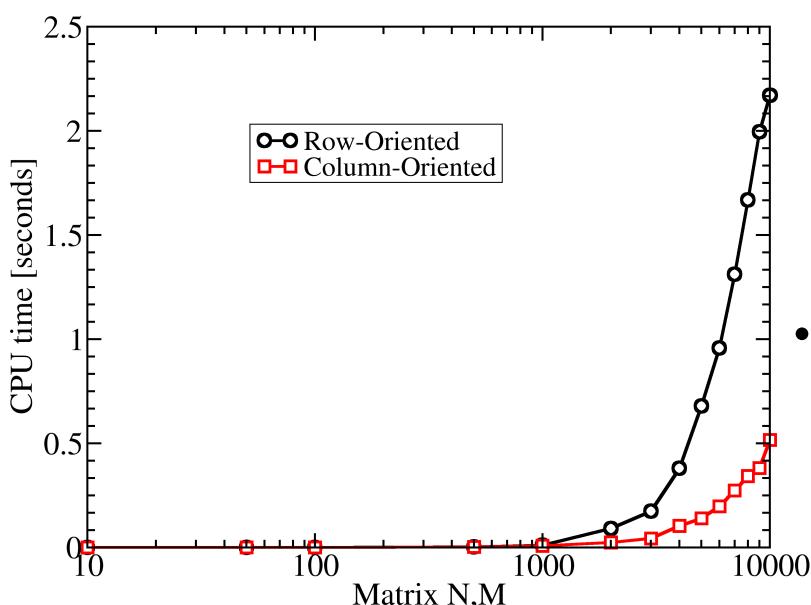
Row-Oriented Operation

```
DO i=1,m
  DO j=1,n
    y(i)=y(i)+a(i,j)*x(j)
  END DO
END DO
```

[example: matrix-vector.f90](#)

Column-Oriented Operation

```
DO j=1,n
  DO i=1,m
    y(i)=y(i)+a(i,j)*x(j)
  END DO
END DO
```



- **Number of flops** (floating point operations): $O(n^2)$ flops for square matrices
(one addition+one multiplication)*n*m = 2nm flops
- **Memory organization**: memory, cache, registers.
In FORTRAN90 the default storage is by columns:

$A = (a_{11}, a_{21}, \dots, a_{m1}, a_{12}, a_{22}, \dots, a_{m2}, \dots, a_{1n}, a_{2n}, \dots, a_{mn})$

by column by rows

Matrix-Matrix Multiplication

$$C_{mxn} = A_{mxr}B_{rxn}$$

$$\begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1r} \\ a_{21} & a_{22} & \dots & a_{2r} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mr} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{r1} & b_{r2} & \dots & b_{rn} \end{pmatrix}$$

$$c_{ij} = \sum_{k=1}^r a_{ik}b_{kj}$$

```
DO i=1,m
  DO j=1,n
    DO k=1,r
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
    END DO
  END DO
END DO
```

- **Number of flops:**
 $O(n^3)$ flops for square matrices
(one addition+one multiplication) $*n*m*r = 2nmr$ flops

- **Memory organization:**
partitioning matrices into blocks to decrease data movement.

BLAS (Basic Linear Algebra Subprograms)

Meaning of prefixes

S - REAL	C - COMPLEX
D - DOUBLE PRECISION	Z - COMPLEX*16
	(this may not be supported by all machines)

For the Level 2 BLAS a set of extended-precision routines with the prefixes ES, ED, EC, EZ may also be available.

Level 1 BLAS

In addition to the listed routines there are two further extended-precision dot product routines DQDOTI and DQDOTA.

Level 2 and Level 3 BLAS

Matrix types:

GE - GEneral	GB - General Band
SY - SYmmetric	SB - Sym. Band
HE - HErmitian	HB - Herm. Band
TR - TRiangular	TB - Triang. Band

SP - Sum. Packed
HP - Herm. Packed
TP - Triang. Packed

Level 2 and Level 3 BLAS Options

Dummy options arguments are declared as CHARACTER*1 and may be passed as character strings.

TRANx	= 'No transpose', 'Transpose', 'Conjugate transpose' (X, X^T, X^H)
UPLO	= 'Upper triangular', 'Lower triangular'
DIAG	= 'Non-unit triangular', 'Unit triangular'
SIDE	= 'Left', 'Right' (A or op(A) on the left, or A or op(A) on the right)

For real matrices, TRANSx = 'T' and TRANSx = 'C' have the same meaning.

For Hermitian matrices, TRANSx = 'T' is not allowed.

For complex symmetric matrices, TRANSx = 'H' is not allowed.

References

C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM Trans. on Math. Soft.* 5 (1979) 308-325

J.J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson, "An Extended Set of Fortran Basic Linear Algebra Subprograms," *ACM Trans. on Math. Soft.* 14,1 (1988) 1-32

J.J. Dongarra, I. Duff, J. DuCroz, and S. Hammarling, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans. on Math. Soft.* (1989)

Obtaining the Software via netlib@ornl.gov

To receive a copy of the single-precision software,
type in a mail message:

```
send sblas from blas
send sblas2 from blas
send sblas3 from blas
```

To receive a copy of the double-precision software,
type in a mail message:

```
send dblas from blas
send dblas2 from blas
send dblas3 from blas
```

To receive a copy of the complex single-precision software,
type in a mail message:

```
send cblas from blas
send cblas2 from blas
send cblas3 from blas
```

To receive a copy of the complex double-precision software,
type in a mail message:

```
send zblas from blas
send zblas2 from blas
send zblas3 from blas
```

Send comments and questions to lapack@cs.utk.edu.

Basic

Linear

Algebra

Subprograms

A Quick Reference Guide

University of Tennessee
Oak Ridge National Laboratory
Numerical Algorithms Group Ltd.

May 11, 1997

[example: matrix-matrix.f90](#)

Systems of Linear Equations

$$A\vec{x} = \vec{b} \iff \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

We focus on solvable systems with $A_{n \times n}$ and $\det A \neq 0$

Note that Cramer's rule $x_i = \frac{\det(A_i)}{\det(A)}$ is computationally inefficient already for $n=3$

Direct Methods

(exact result in a finite number of steps, apart from rounding errors)

- Forward/Backward substitution for triangular matrices
- Gauss elimination for generic matrices
- Thomas algorithm for tridiagonal matrices

Iterative Methods

(exact result can only be achieved after an infinite number of steps)

- Jacobi method
- Gauss-Seidel method
- (SOR- and JOR-methods)

Condition number of a problem

Given a problem f and the input data x , and an algorithm \tilde{f} with the disturbed input data \tilde{x} , then the absolute error

$$\|f(x) - \tilde{f}(\tilde{x})\| = \|f(x) - f(\tilde{x}) + f(\tilde{x}) - \tilde{f}(\tilde{x})\| \leq \|f(x) - f(\tilde{x})\| + \|f(\tilde{x}) - \tilde{f}(\tilde{x})\|$$

condition stability

The relative **condition number** of the problem $f(x)$ is the smallest number $K \geq 0$, such that

$$\frac{\|f(x) - f(\tilde{x})\|}{\|f(x)\|} \leq K \frac{\|\tilde{x} - x\|}{\|x\|}$$

The condition number associated with the linear equation $Ax = b$ is defined as the maximum ratio of the relative error in x to the relative error in b

$$\begin{aligned} \max_{e,b \neq 0} \left\{ \frac{\|A^{-1}e\|}{\|e\|} \frac{\|b\|}{\|A^{-1}b\|} \right\} &= \max_{e \neq 0} \left\{ \frac{\|A^{-1}e\|}{\|e\|} \right\} \max_{b \neq 0} \left\{ \frac{\|b\|}{\|A^{-1}b\|} \right\} = \\ &= \max_{e \neq 0} \left\{ \frac{\|A^{-1}e\|}{\|e\|} \right\} \max_{x \neq 0} \left\{ \frac{\|Ax\|}{\|x\|} \right\} = \\ &= \|A^{-1}\| \|A\| \end{aligned}$$

The condition number of a matrix A is $K(A) = \|A\| \|A^{-1}\|$

$\left\{ \begin{array}{l} \sim 1 \text{ then } A \text{ is well-conditioned} \\ \gg 1 \text{ then } A \text{ is ill-conditioned} \end{array} \right.$

Forward/Backward substitution for triangular matrices

Lower Triangular

$$A = \begin{pmatrix} a_{11} & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 \\ a_{31} & a_{32} & a_{33} & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix}$$

$$a_{ij} \neq 0$$

We solve the system from top to bottom

$$\begin{aligned} a_{11}x_1 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \\ &\vdots = \vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \dots + a_{nn}x_n &= b_n \end{aligned}$$

Once we have x_1, x_2, \dots, x_{i-1} , we can solve for x_i

$$x_i = \frac{b_i - a_{i1}x_1 - a_{i2}x_2 - \dots - a_{i,i-1}x_{i-1}}{a_{ii}} \leftrightarrow x_i = a_{ii}^{-1} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j \right)$$

```
do i=1, n
  do j=1, i-1
    b(i) = b(i)-a(i,j)*b(j)
  end do
  if(a(i,i).ne.0)then
    b(i)=b(i)/a(i,i)
  else
    write(*,*) "error: singular matrix"
  end if
end do
```

Forward substitution

2n(n-1) flops

n flops

O(n²) flops

Forward/Backward substitution for triangular matrices

Upper Triangular

We solve the system from bottom to top

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

Backward substitution

Note: it is easy to develop block variants of both forward and back substitution

[example: fwd-bwd-substitution.f90](#)

Gauss elimination method for generic matrices

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots = \vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n$$

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

Reduction to a
triangular
form

(n-1) reduction steps with elementary row operations

1. swapping rows
2. multiplying rows by nonzero numbers
3. adding a multiple of one row to another row

$$U\vec{x} = \vec{z}$$

$$u_{11}x_1 + u_{12}x_2 + \dots + u_{1n}x_n = z_1$$

$$u_{22}x_2 + \dots + u_{2n}x_n = z_2$$

$$\dots = \dots$$

$$u_{nn-1}x_{n-1} + u_{nn}x_n = z_n$$

$$u_{nn}x_n = z_n$$

U is upper triangular

Gauss elimination method for generic matrices

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots = \vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n$$

$$l_{i1} = a_{i1}/a_{11}$$

$$\text{new Row } i = \text{Row } i - l_{i1} \text{ Row 1}$$

$$(a_{i1} - l_{i1}a_{11})x_1 + \underbrace{(a_{i2} - l_{i1}a_{12})x_2}_{=a'_{i2}} + \dots + \underbrace{(a_{in} - l_{i1}a_{1n})x_n}_{=a'_{in}} = \underbrace{b_i - l_{i1}b_1}_{=b'_i}$$

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$+ a'_{22}x_2 + \dots + a'_{2n}x_n = b'_2$$

$$\vdots \quad \vdots \quad = \vdots$$

$$+ a'_{n2}x_2 + \dots + a'_{nn}x_n = b'_n$$

$$\text{new Row } i = \text{Row } i - \frac{a'_{i2}}{a'_{22}} \text{ Row 2}$$

Gauss elimination method for generic matrices

$$A^{(k)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & \cdots & \cdots & a_{2n}^{(2)} \\ \vdots & 0 & \ddots & & & \vdots \\ \vdots & \vdots & 0 & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & a_{nk}^{(k)} & \cdots & a_{nn}^{(k)} \end{pmatrix} \quad \xrightarrow{\hspace{10em}} \text{residual matrix}$$

From $A^{(k)}\vec{x} = \vec{b}^{(k)}$ to $A^{(k+1)}\vec{x} = \vec{b}^{(k+1)}$

$$\begin{aligned} l_{ik} &= a_{ik}^{(k)} / a_{kk}^{(k)} && \text{nonzero pivot element} \quad \text{for } i = k + 1, \dots, n \\ a_{ij}^{(k+1)} &= a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)} && \text{for } i, j = k + 1, \dots, n \\ b_j^{(k+1)} &= b_i^{(k)} - l_{ik} b_k^{(k)} && \text{for } i = k + 1, \dots, n \end{aligned}$$

Gauss elimination method for generic matrices

- $A = LU$, A is decomposed into a lower and an upper triangular matrix

$$A\vec{x} = \vec{b} \implies LU\vec{x} = \vec{b} \quad \text{LU-factorization}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{21} & 1 & 0 & \dots & 0 \\ l_{31} & l_{32} & 1 & \dots & \vdots \\ \vdots & \vdots & \vdots & \dots & 0 \\ l_{n1} & l_{n2} & \dots & l_{nn-1} & 1 \end{pmatrix} \quad U = \begin{pmatrix} u_{11} & u_{12} & \dots & \dots & u_{1n} \\ 0 & u_{22} & \dots & \dots & u_{2n} \\ 0 & 0 & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & u_{nn} \end{pmatrix}$$

- $L\vec{z} = \vec{b}$ is solved by forward substitution
- $U\vec{x} = \vec{z}$ is solved by backward substitution

$O(n^3)$ flops for the triangular decomposition

$O(n^2)$ flops for the forward and backward substitution

Gauss elimination method for generic matrices

- In some cases the LU-factorization cannot be carried out even though $\det A \neq 0$

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \det A = -1, a_{11} = 0$$

As the matrix is not singular we search for a pivot element that is not zero

$$A^1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathbb{1} = LU \text{ where } L = U = \mathbb{1}$$

- The **pivot search** improves the numerical accuracy

$$A = \begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \quad \text{within our numerical accuracy} \quad \left(1 - \frac{1}{\epsilon}\right) \approx -\frac{1}{\epsilon}$$

without pivoting

$$LU = \begin{pmatrix} 1 & 0 \\ \frac{1}{\epsilon} & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 1 \\ 0 & 1 - \frac{1}{\epsilon} \end{pmatrix} \approx \begin{pmatrix} 1 & 0 \\ \frac{1}{\epsilon} & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 1 \\ 0 & -\frac{1}{\epsilon} \end{pmatrix} = \begin{pmatrix} \epsilon & 1 \\ 1 & 0 \end{pmatrix} \neq A$$

with pivoting

$$LU = \begin{pmatrix} 1 & 0 \\ \epsilon & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 - \epsilon \end{pmatrix} \approx \begin{pmatrix} 1 & 0 \\ \epsilon & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ \epsilon & 1 \end{pmatrix} \equiv A$$

Gauss elimination method for generic matrices

- The choice of the pivot is thus crucial for the minimization of the rounding errors

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

If $\epsilon = 10^{-4}$, then the exact solution is

$x_1=1.0$ and $x_2=0.9999$
with a 4-decimal accuracy

$x_1=1.0$ and $x_2=1.0$
with a 3-decimal accuracy

LU-factorization without pivoting

$$\begin{pmatrix} \epsilon & 1 \\ 0 & 1 - \frac{1}{\epsilon} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 - \frac{1}{\epsilon} \end{pmatrix}$$

$x_1=0.0$ wrong
 $x_2=1.0$ correct

LU-factorization with pivoting

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 - \epsilon \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 - 2\epsilon \end{pmatrix}$$

$x_1=1.0$ correct
 $x_2=1.0$ correct

If we assume a
3-decimal accuracy

$$\left(1 - \frac{1}{\epsilon}\right) \approx -10^4$$

$$(1 - \epsilon) \approx 1$$

Attention: column pivoting vs total pivoting (largest matrix element of the residual matrix, swapping of columns & rows)!

Pivot search by columns

The Gauss elimination method with column pivoting, implies that in the elimination step from $A^{(k)}$ to $A^{(k+1)}$

- we first search by columns for a suitable pivot element , *i.e.* we look for the element with the largest absolute value from the k -th sub-column $a_{jk}^{(k)}$ for $j = k, \dots, n$; if this element is in the r -th row, then the pivot element is $a_{rk}^{(k)}$
- we swap the r -th with the k -th row in $A^{(k)}$ and $\vec{b}^{(k)}$
- as the pivot element $a_{kk}^{(k)}$ is not zero, we proceed with the Gauss elimination step

For each invertible square matrix A , there exist a permutation matrix P such that the LU-factorization can be written as $PA=LU$

Reference Guide

to the

Driver Routines

Release 3.0

Simple Drivers

Simple Driver Routines for Linear Equations

Matrix Type	Routine			
General	SGESV(N, CGESV(N,	NRHS, A, LDA, NRHS, A, LDA,	IPIV, B, LDB, IPIV, B, LDB,	INFO) INFO)
General Band	SGBSV(N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, CGBSV(N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB,			INFO) INFO)
General Tridiagonal	SGTSV(N, CGTSV(N,	NRHS, DL, D, DU, NRHS, DL, D, DU,	B, LDB, B, LDB,	INFO) INFO)
Symmetric/Hermitian Positive Definite	SPOSV(UPLO, N, CPOSV(UPLO, N,	NRHS, A, LDA, NRHS, A, LDA,	B, LDB, B, LDB,	INFO) INFO)
Symmetric/Hermitian Positive Definite (Packed Storage)	SPPSV(UPLO, N, CPPSV(UPLO, N,	NRHS, AP, NRHS, AP,	B, LDB, B, LDB,	INFO) INFO)
Symmetric/Hermitian Positive Definite Band	SPBSV(UPLO, N, KD, CPBSV(UPLO, N, KD,	NRHS, AB, LDAB, NRHS, AB, LDAB,	B, LDB, B, LDB,	INFO) INFO)
Symmetric/Hermitian Positive Definite Tridiagonal	SPTSV(N, CPTSV(N,	NRHS, D, E, NRHS, D, E,	B, LDB, B, LDB,	INFO) INFO)
Symmetric/Hermitian Indefinite	SSYSV(UPLO, N, CSYSV(UPLO, N, CHESV(UPLO, N,	NRHS, A, LDA, NRHS, A, LDA, NRHS, A, LDA,	IPIV, B, LDB, WORK, LWORK, IPIV, B, LDB, WORK, LWORK, IPIV, B, LDB, WORK, LWORK,	INFO) INFO) INFO)
Symmetric/Hermitian Indefinite (Packed Storage)	SSPSV(UPLO, N, CSPSV(UPLO, N, CHPSV(UPLO, N,	NRHS, AP, NRHS, AP, NRHS, AP,	IPIV, B, LDB, IPIV, B, LDB, IPIV, B, LDB,	INFO) INFO) INFO)

Simple Driver Routines for Standard and Generalized Linear Least Squares Problems

Problem Type	Routine			
Solve Using Orthogonal Factor, Assuming Full Rank	SGELS(TRANS, N, NRHS, A, LDA, B, LDB, CGELS(TRANS, N, NRHS, A, LDA, B, LDB,			WORK, LWORK, INFO) WORK, LWORK, INFO)
Solve LSE Problem Using GRQ	SGGLSE(H, N, P, A, LDA, B, LDB, C, D, X, CGGLSE(H, N, P, A, LDA, B, LDB, C, D, X,			WORK, LWORK, INFO) WORK, LWORK, INFO)
Solve GLM Problem Using GQR	SGGGLH(N, H, P, A, LDA, B, LDB, D, X, Y, CGGGLH(N, H, P, A, LDA, B, LDB, D, X, Y,			WORK, LWORK, INFO) WORK, LWORK, INFO)

Gauss elimination method for generic matrices

Example of LU-factorization with pivoting

$$A = \begin{pmatrix} 1 & 6 & 1 \\ 2 & 3 & 2 \\ 4 & 2 & 1 \end{pmatrix} \xrightarrow{\text{swap 1st&3rd}} \begin{pmatrix} 4 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 6 & 1 \end{pmatrix} \xrightarrow{\text{swap 2nd&3rd}} \begin{pmatrix} 4 & 2 & 1 \\ 0 & 2 & \frac{3}{2} \\ 0 & \frac{11}{2} & \frac{3}{4} \end{pmatrix} \xrightarrow{\quad} \begin{pmatrix} 4 & 2 & 1 \\ 0 & \frac{11}{2} & \frac{3}{4} \\ 0 & 2 & \frac{3}{2} \end{pmatrix} \xrightarrow{\quad} \begin{pmatrix} 4 & 2 & 1 \\ 0 & \frac{11}{2} & \frac{3}{4} \\ 0 & 0 & \frac{27}{22} \end{pmatrix}$$

$$l_{21} = a_{21}^{(1)} / a_{11}^{(1)} = 1/2$$

$$l_{31} = a_{31}^{(1)} / a_{11}^{(1)} = 1/4$$

$$l_{32} = a_{32}^{(2)} / a_{22}^{(2)} = 4/11$$

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{4} & \frac{4}{11} & 1 \end{pmatrix} \begin{pmatrix} 4 & 2 & 1 \\ 0 & \frac{11}{2} & \frac{3}{4} \\ 0 & 0 & \frac{27}{22} \end{pmatrix} PA = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 6 & 1 \\ 2 & 3 & 2 \\ 4 & 2 & 1 \end{pmatrix}$$

without pivoting

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & \frac{22}{9} & 1 \end{pmatrix} \begin{pmatrix} 1 & 6 & 1 \\ 0 & -9 & 0 \\ 0 & 0 & -3 \end{pmatrix}$$

[example: lu-factorization.f90](#)

Gauss elimination method for generic matrices

Example of Gauss elimination

with pivoting

$$\begin{pmatrix} 1 & 6 & 1 \\ 2 & 3 & 2 \\ 4 & 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{4} & \frac{4}{11} & 1 \end{pmatrix} \begin{pmatrix} 4 & 2 & 1 \\ 0 & \frac{11}{2} & \frac{3}{4} \\ 0 & 0 & \frac{27}{22} \end{pmatrix}$$

$$A = LU$$

$$\begin{pmatrix} 4 & 2 & 1 \\ 0 & \frac{11}{2} & \frac{3}{4} \\ 0 & 0 & \frac{27}{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ \frac{1}{4} \\ \frac{9}{22} \end{pmatrix}$$

$$U\vec{x} = \vec{z}$$

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \frac{2}{3} \\ 0 \\ \frac{1}{3} \end{pmatrix}$$

without pivoting

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & \frac{22}{9} & 1 \end{pmatrix} \begin{pmatrix} 1 & 6 & 1 \\ 0 & -9 & 0 \\ 0 & 0 & -3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 1 \\ 0 & -9 & 0 \\ 0 & 0 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

[example: gauss-elimination.f90](#)

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & \frac{22}{9} & 1 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$L\vec{z} = \vec{b} \quad \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

Thomas algorithm for tridiagonal matrices

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & \dots & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & \dots & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & \dots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & \dots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \dots & \dots & \dots & 0 & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix} \quad \text{where } a_1=0 \text{ and } c_n=0$$

$$b_1 x_1 + c_1 x_2 = f_1 \quad \text{for } i = 1$$

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = f_i \quad \text{for } i = 2, \dots, n-1$$

$$a_n x_{n-1} + b_n x_n = f_n \quad \text{for } i = n$$

Let us assume

$$x_{i+1} = \alpha_i x_i + \beta_i$$

Then

$$a_i x_{i-1} + b_i x_i + c_i (\alpha_i x_i + \beta_i) = a_i x_{i-1} + (b_i + c_i \alpha_i) x_i + c_i \beta_i = f_i$$

Solving for x_i

$$x_i = -\frac{a_i}{b_i + c_i \alpha_i} x_{i-1} + \frac{f_i - c_i \beta_i}{b_i + c_i \alpha_i}$$

Thomas algorithm for tridiagonal matrices

The beginning of the recursion can be obtained by looking at the last line of the system

$$a_n x_{n-1} + b_n x_n = f_n \implies x_n = -\frac{a_n}{b_n} x_{n-1} + \frac{f_n}{b_n} = \alpha_{n-1} x_{n-1} + \beta_{n-1}$$

Thus $\alpha_n = 0$ and $\beta_n = 0$

Then α_i and β_i (careful: $i = n - 1, n - 2, \dots, 1$)

$$\alpha_{i-1} = -\frac{a_i}{b_i + c_i \alpha_i} \quad \text{and} \quad \beta_{i-1} = \frac{f_i - c_i \beta_i}{b_i + c_i \alpha_i}$$

With the knowledge of α_1 and β_1 , we calculate x_1 , and then all x_i

$$x_{i+1} = \alpha_i x_i + \beta_i$$

Thomas algorithm vs Gauss elimination

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ l_2 & 1 & 0 & 0 & 0 \\ 0 & l_3 & 1 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ 0 & \dots & 0 & l_n & 1 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} d_1 & c_1 & 0 & 0 & 0 \\ 0 & d_2 & c_2 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 0 & d_n \end{pmatrix}$$

where $l_i = a_i/d_{i-1}$ and $d_i = b_i - l_i c_{i-1}$ and the recursion starts with $d_1 = b_1$

Systems of Linear Equations

$$A\vec{x} = \vec{b} \iff \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

We focus on solvable systems with $A_{n \times n}$ and $\det A \neq 0$

Note that Cramer's rule $x_i = \frac{\det(A_i)}{\det(A)}$ is computationally inefficient already for $n=3$

Direct Methods

(exact result in a finite number of steps, apart from rounding errors)

- Forward/Backward substitution for triangular matrices
- Gauss elimination for generic matrices
- Thomas algorithm for tridiagonal matrices

Iterative Methods

(exact result can only be achieved after an infinite number of steps)

- Jacobi method
- Gauss-Seidel method
- (SOR- and JOR-methods)

Iterative Methods for sparse matrices

The concept of iterative methods can be demonstrated by considering the simple linear equation

$$(1 - a)x = b$$

- Iteration rule
- Initial condition

$$\begin{aligned}x^{(k+1)} &= ax^{(k)} + b \quad \text{where } k = 0, 1, \dots \\x^{(0)} &= 0\end{aligned}$$

$$\begin{aligned}x^{(1)} &= ax^{(0)} + b \\x^{(2)} &= ax^{(1)} + b = a^2x^{(0)} + ab + b \\&\dots = \dots \\x^{(k+1)} &= ax^{(k)} + b = a^{k+1}x^{(0)} + (a^k + a^{k-1} + \dots + a + 1)b\end{aligned}$$

$$|a| < 1$$

$$\lim_{k \rightarrow \infty} a^{k+1} = 0 \quad \text{and} \quad \sum_{k=0}^{\infty} a^k = \frac{1}{1-a} \longrightarrow$$

$$x = \frac{b}{1-a}$$

- Weighted iteration rule

$$x^{(k+1)} = \omega(ax^{(k)} + b) + (1 - \omega)x^{(k)} = (1 - \omega(1 - a))x^{(k)} + \omega b \quad \text{where } k = 0, 1, \dots$$

$$|1 - \omega(1 - a)| < 1$$



$$x = \frac{\omega b}{1 - [1 - \omega(1 - a)]} = \frac{b}{1 - a}$$

Iterative Methods for sparse matrices

The same concept applied to linear systems of equations

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

In order to arrive at an iteration rule, one separates an unknown in each equation

$$x_1 = c_{12}x_2 + c_{13}x_3 + d_1$$

$$x_2 = c_{21}x_1 + c_{23}x_3 + d_2$$

$$x_3 = c_{31}x_1 + c_{32}x_2 + d_3$$

where $c_{ii} = 0$, $c_{ij} = -a_{ij}/a_{ii}$ for $i \neq j$ and $d_i = b_i/a_{ii}$ for $i, j = 1, 2, 3$

The system of equations can be thus divided into two parts

$$L = \begin{pmatrix} 0 & c_{12} & c_{13} \\ 0 & 0 & c_{23} \\ 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 0 & 0 & 0 \\ c_{21} & 0 & 0 \\ c_{31} & c_{32} & 0 \end{pmatrix}$$

$$\vec{x} = U\vec{x} + L\vec{x} + \vec{d}$$

Iterative Methods for sparse matrices

- the overall-step method or Jacobi method (J-method)

$$\vec{x}^{(k+1)} = U\vec{x}^{(k)} + L\vec{x}^{(k)} + \vec{d}$$

- the single-step method or Gauss and Seidel method (GS-method)

$$\vec{x}^{(k+1)} = U\vec{x}^{(k)} + L\vec{x}^{(k+1)} + \vec{d}$$

- the single-step method with over-relaxation, often referred to as successive over-relaxation method (SOR-method)

$$\vec{x}^{(k+1)} = \omega(U\vec{x}^{(k)} + L\vec{x}^{(k+1)} + \vec{d}) + (1 - \omega)\vec{x}^{(k)}$$

- the overall-step method with over-relaxation, that is a generalization of the Jacobi method with over-relaxation (often referred to as JOR-method)

$$\vec{x}^{(k+1)} = \omega(U\vec{x}^{(k)} + L\vec{x}^{(k)} + \vec{d}) + (1 - \omega)\vec{x}^{(k)}$$

Iterative Methods for sparse matrices

$$A\vec{x} = \vec{b} \quad \sum_{j=1}^n a_{ij}x_j = b_i \quad x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij}x_j \right)$$

- Jacobi method

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij}x_j^{(k)} \right)$$

Let us write $A=D+R$, where

$$D = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix} \quad \text{and} \quad R = \begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}$$

$$\begin{aligned} x^{(k+1)} &= D^{-1} \left(b - Rx_j^{(k)} \right) \\ x^{(k+1)} &= x^{(k)} + D^{-1}r^{(k)} \end{aligned} \quad r^{(k)} = b - Ax^{(k)}$$

If diagonal entries \gg off-diagonal entries
convergence is aided

Iterative Methods for sparse matrices

$$A\vec{x} = \vec{b}$$
$$\sum_{j=1}^n a_{ij}x_j = b_i$$
$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij}x_j \right)$$

- Gauss-Seidl method

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(k)} \right)$$

Let us write $A=D+L+U$

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

If we define the residual after k iterations as before, we get

$$x^{(k+1)} = x^{(k)} + (D + L)^{-1}r^{(k)}$$

J vs GS: J is parallel, GS is sequential but saves memory

Iterative Methods for sparse matrices

$$A\vec{x} = \vec{b} \quad \sum_{j=1}^n a_{ij}x_j = b_i \quad x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij}x_j \right)$$

- SOR method

$$x_i^{(k+1)} = \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(k)} \right) + (1 - \omega)x_i^{(k)}$$

$$x^{(k+1)} = \omega x_{\text{GS}}^{(k+1)} + (1 - \omega)x^{(k)}$$

- JOR method

$$x_i^{(k+1)} = \frac{\omega}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij}x_j^{(k)} \right) + (1 - \omega)x^{(k)}$$

$$x^{(k+1)} = \omega x_{\text{J}}^{(k+1)} + (1 - \omega)x^{(k)}$$

That's all for Today!

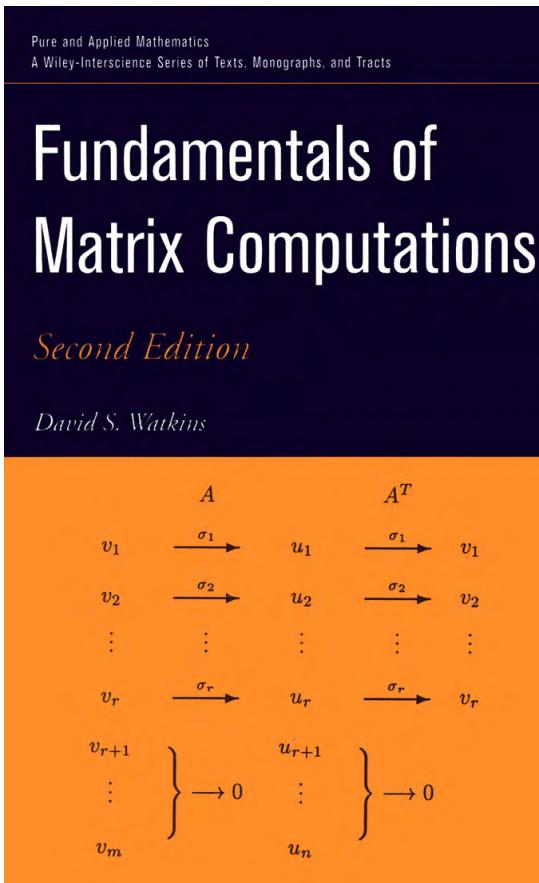
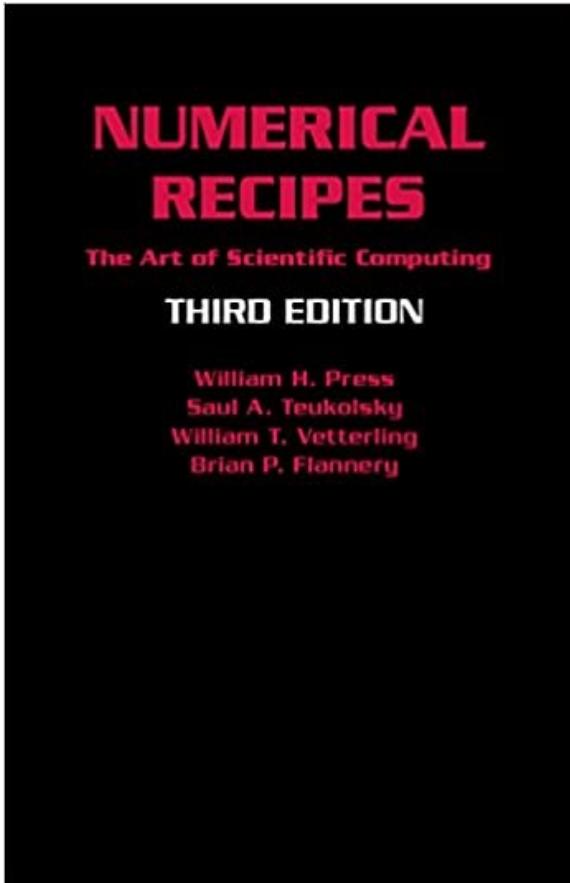
- Vector and Matrix Operations (BLAS)
- Systems of Linear Equations (LAPACK)
- [Exercise 1](#)



04.04.2022

Numerical Methods for Linear Algebra

- Vector and Matrix Operations (BLAS)
- Systems of Linear Equations (LAPACK)
- [Exercise 1](#)
- Eigenvalues and Eigenvectors computation
- [Exercise 2](#)



Notes and slides: “NM4LA-1/2” on TISS as pdf files

Today!

- Vector and Matrix Operations (BLAS)
- Systems of Linear Equations (LAPACK)
- [Exercise 1](#)



05/06.04.2022

Poisson equation in 1D

$$u''(x) = -f(x)$$

If we discretize the equation, we get a system of linear equations:

$$u''(x_i) = -f(x_i) \quad i = 1, \dots, n-1$$

We now use an approximation for the second derivative ...

$$u''(x_i) \approx \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{\Delta x^2}$$

... and the set of equation becomes

$$-u_{i-1} + 2u_i - u_{i+1} = \Delta x^2 f(x_i)$$

$$\begin{aligned} 1. \quad & \vec{\nabla} \cdot \vec{E} = \frac{\rho}{\epsilon} \\ 3. \quad & \vec{\nabla}_x \vec{E} = \frac{\partial \vec{B}}{\partial t} = 0 \\ & \vec{E} = -\nabla \phi \\ & \downarrow \\ \nabla^2 \phi = -\frac{\rho}{\epsilon} \end{aligned}$$

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & \ddots & \ddots & \ddots & 0 \\ \vdots & \dots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ \dots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \Delta x^2 \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \dots \\ \dots \\ f(x_{n-2}) \\ f(x_{n-1}) \end{pmatrix}$$