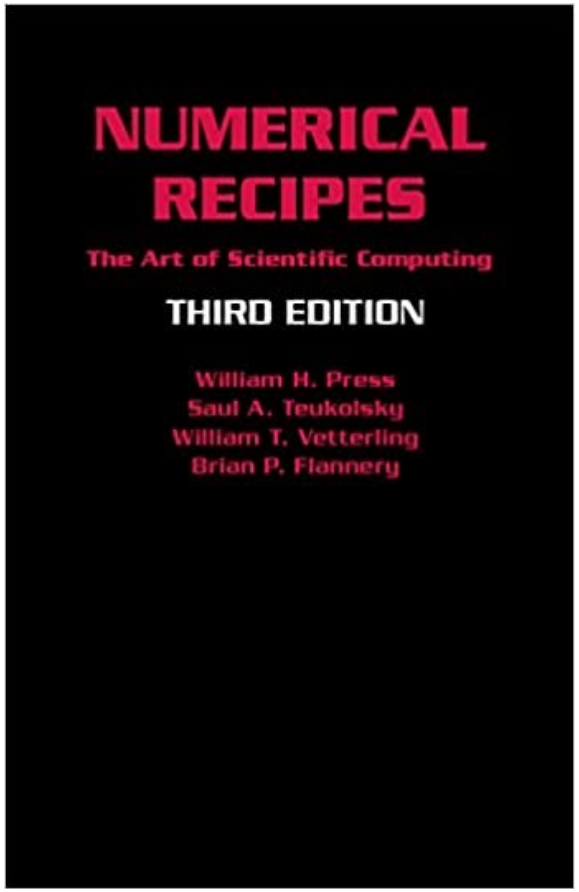
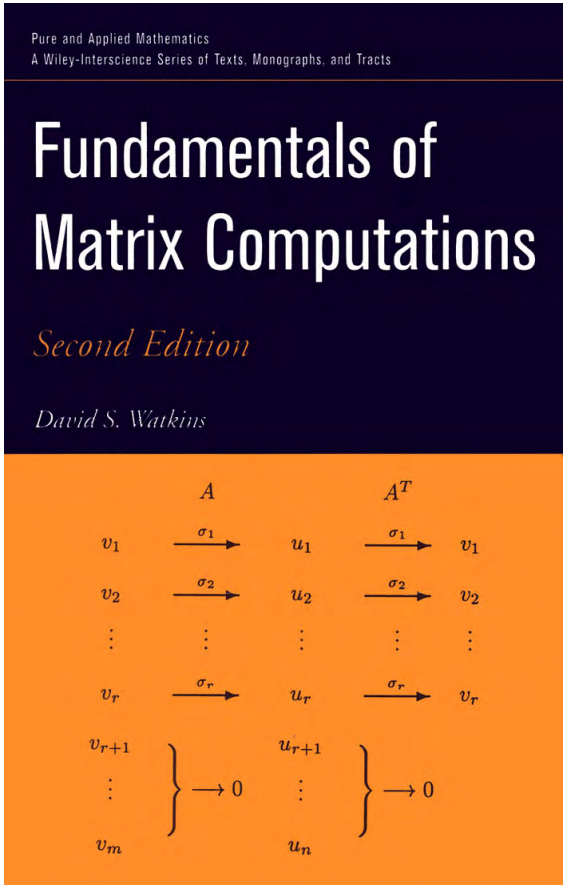
Numerical Methods for Linear Algebra

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

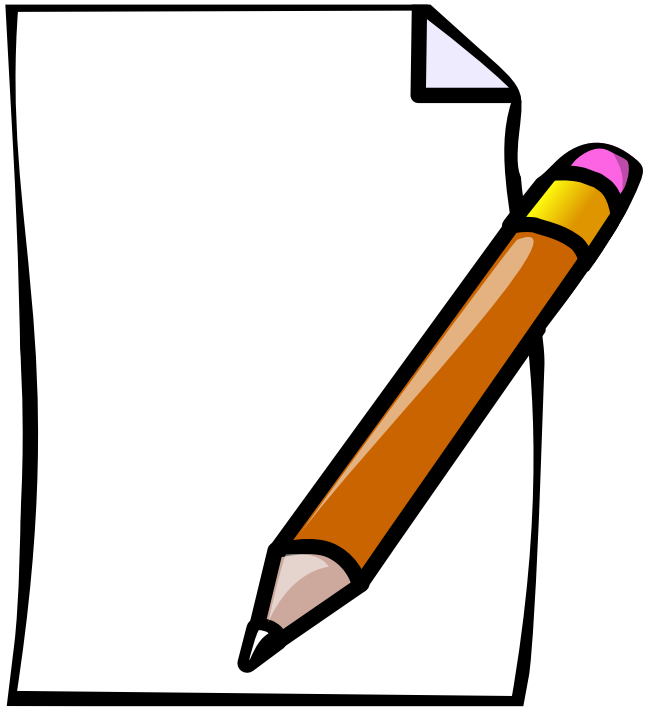




Notes and slides: "NM4LA-1/2" on TISS as pdf files

Today

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



04.04.2022

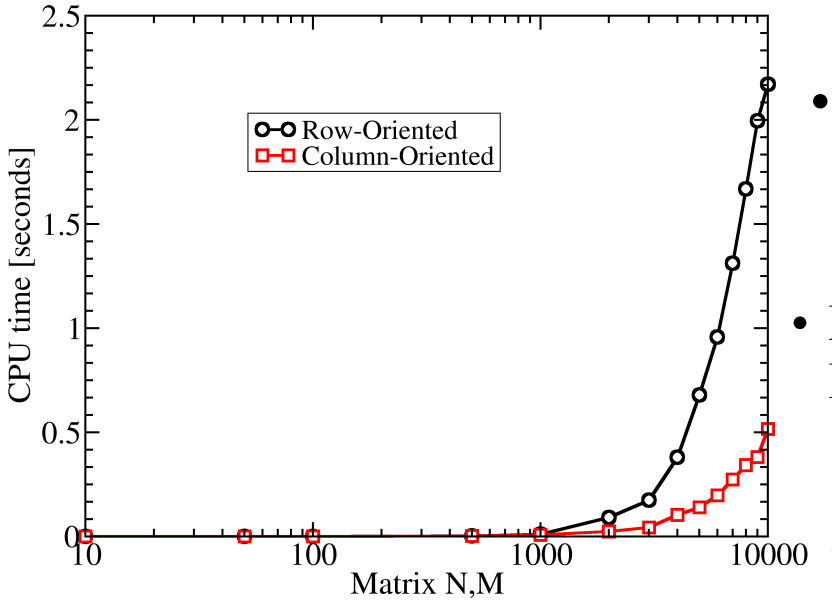
Matrix-Vector Multiplication

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

Row-Oriented Operation — [example: matrix-vector.f90] — **Column**-Oriented Operation

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

```
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:

by rows / by column

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

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	SP - Sum. Packed
HE - HErmitian	HB - Herm. Band	HP - Herm. Packed
TR - TRiangular	TB - Triang. Band	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	= '**N**o transpose', '**T**ranspose', '**C**onjugate transpose' (X, X^T, X^H)
UPLO	= '**U**pper triangular', '**L**ower triangular'
DIAG	= '**N**on-unit triangular', '**U**nit triangular'
SIDE	= '**L**eft', '**R**ight' (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

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

We focus on solvable systems with A_{nxn} and detA≠0

Note that Cramer's rule $x_i = \dfrac{\det(A_i)}{\det(A)}$ 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

condition stability

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

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 $\boldsymbol{Ax = b}$ 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 $K(A) = ||A||||A^{-1}||$

- $\sim$ 1 then A is well-conditioned
- $>>$ 1 then A is ill-conditioned

Forward/Backward substitution for triangular matrices

Lower Triangular

We solve the system from top to bottom

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

$$\boxed{a_{ij} \neq 0}$$

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

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

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

Forward substitution

```
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
```

2n(n-1) flops

n flops

$O(n^2)$ 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

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\
\vdots \quad \vdots \quad\quad \vdots \quad \vdots &= \vdots \\
a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n
\end{aligned}
$$

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

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 is upper triangular

$$
\begin{aligned}
u_{11}x_1 + u_{12}x_2 + \quad \ldots \quad + u_{1n}x_n &= z_1 \\
u_{22}x_2 + \quad \ldots \quad + u_{2n}x_n &= z_2 \\
\ldots \quad\quad\quad\quad\quad\quad &= \ldots \\
u_{nn-1}x_{n-1} + u_{nn}x_n &= z_2 \\
u_{nn}x_n &= z_n
\end{aligned}
$$

Gauss elimination method for generic matrices

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\
\vdots \quad \vdots \quad \vdots \quad \vdots &= \vdots \\
a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n
\end{aligned}
$$

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

new Row i = Row i − l_{i1} Row 1

$$\underbrace{(a_{i1} - l_{i1}a_{11})}_{=0}x_1 + \underbrace{(a_{i2} - l_{i1}a_{12})}_{=a'_{i2}}x_2 + \ldots + \underbrace{(a_{in} - l_{i1}a_{1n})}_{=a'_{in}}x_n = \underbrace{b_i - l_{i1}b_1}_{=b'_i}$$

$$
\begin{aligned}
a_{11}x_1 \quad + \quad & a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\
+ \quad & a'_{22}x_2 + \ldots + a'_{2n}x_n = b'_2 \\
& \vdots \qquad\quad \vdots \quad = \vdots \\
+ \quad & a'_{n2}x_2 + \ldots + a'_{nn}x_n = b'_n
\end{aligned}
$$

new Row i = Row i − $\dfrac{a'_{i2}}{a'_{22}}$ Row 2

Gauss elimination method for generic matrices

$$A^{(k)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & \dots & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \dots & \dots & \dots & a_{2n}^{(2)} \\ \vdots & 0 & \ddots & & & \vdots \\ \vdots & \vdots & 0 & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & a_{nk}^{(k)} & \dots & a_{nn}^{(k)} \end{pmatrix}$$

residual matrix

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

nonzero pivot element

$$\begin{aligned} l_{ik} &= a_{ik}^{(k)} / a_{kk}^{(k)} && \text{for } i = k+1, \dots, n \\ a_{ij}^{(k+1)} &= a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)} && \text{for } i, j = k+1, \dots, n \\ b_{j}^{(k+1)} &= b_{i}^{(k)} - l_{ik} b_{k}^{(k)} && \text{for } i = k+1, \dots, n \end{aligned}$$

Gauss elimination method for generic matrices

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

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

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 detA≠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

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

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)!

Gauss elimination method for generic matrices

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

Simple Driver Routines for Standard and Generalized Linear Least Squares Problems

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

Gauss elimination method for generic matrices

Example of LU-factorization

with pivoting

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

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

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

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

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

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

Example of Gauss elimination

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

with pivoting

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

$$A = LU$$

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

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

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

without pivoting

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

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

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

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

example: gauss-elimination.f90

Thomas algorithm for tridiagonal matrices

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

where $a_1=0$ and $c_n=0$

$$\begin{aligned} b_1 x_1 + c_1 x_2 &= f_1 \quad \text{for} \quad i = 1 \\ a_i x_{i-1} + b_i x_i + c_i x_{i+1} &= f_i \quad \text{for} \quad i = 2, \dots, n-1 \\ a_n x_{n-1} + b_n x_n &= f_n \quad \text{for} \quad i = n \end{aligned}$$

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

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

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

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

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

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**

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

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

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

We focus on solvable systems with A_{nxn} and detA≠0

Note that Cramer's rule $x_i = \dfrac{\det(A_i)}{\det(A)}$ 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

$$\boxed{(1-a)x = b}$$

- Iteration rule
- Initial condition

$$x^{(k+1)} = ax^{(k)} + b \quad \text{where} \quad k = 0, 1, \ldots$$
$$x^{(0)} = 0$$

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

$$\boxed{|a| < 1}$$

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

- Weighted iteration rule

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

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

Iterative Methods for sparse matrices

The same concept applied to linear systems of equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned}$$

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

$$\begin{aligned} x_1 &= c_{12}x_2 + c_{13}x_3 + d_1 \\ x_2 &= c_{21}x_1 + c_{23}x_3 + d_2 \\ x_3 &= c_{31}x_1 + c_{32}x_2 + d_3 \end{aligned}$$

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

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

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

- 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

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

- 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

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

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

$$x^{(k+1)} = x^{(k)} + D^{-1}r^{(k)} \qquad r^{(k)} = b - Ax^{(k)}$$

If diagonal entries >> off-diagonal entries
convergence is aided

Iterative Methods for sparse matrices

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

- 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

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

- SOR method

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

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

- JOR method

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

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

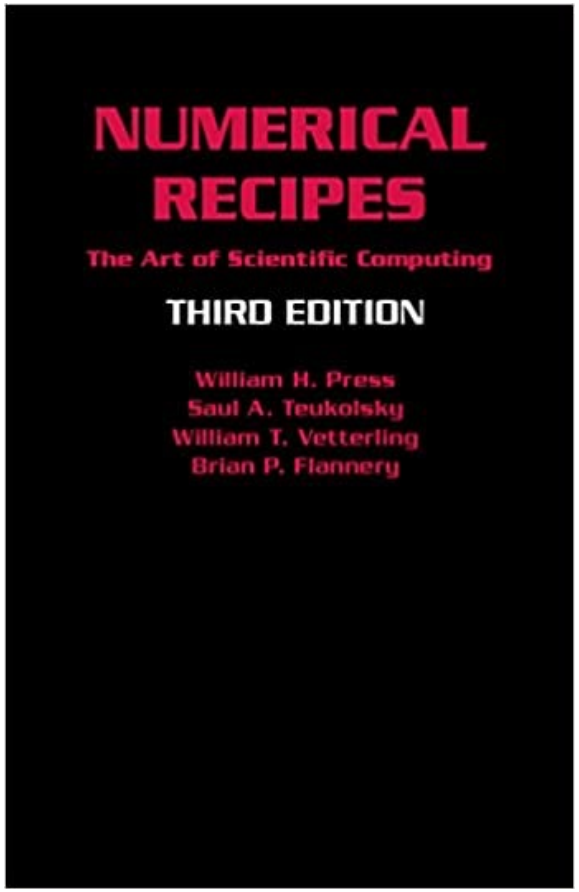
That's all for Today!

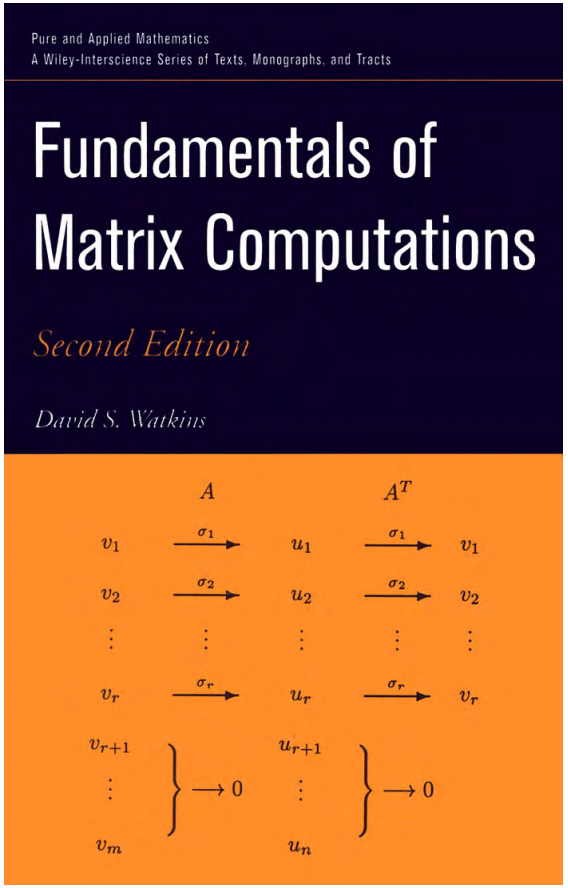
- Vector and Matrix Operations (BLAS)
- Systems of Linear Equations (LAPACK)
- Exercise 1

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:

$$u''(x_i) = -f(x_i) \qquad i = 1, \ldots, n-1$$

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

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

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

1. $\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\epsilon}$

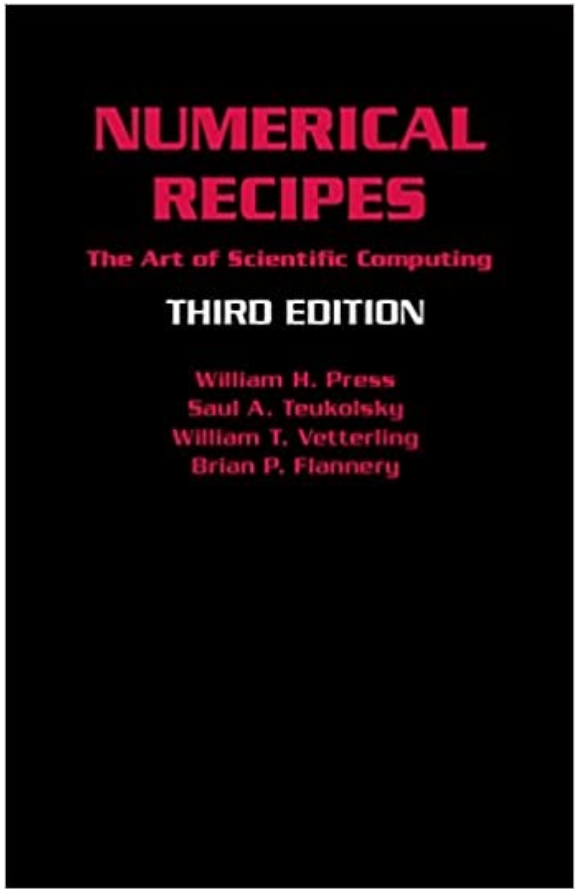
3. $\vec{\nabla} \mathrm{x} \vec{E} = \frac{\partial \vec{B}}{\partial t} = 0$

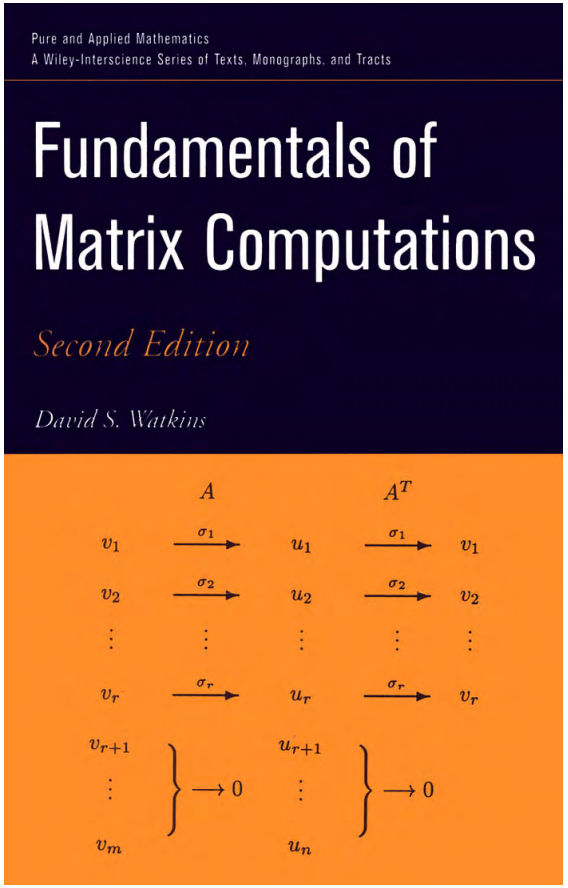
$$\vec{E} = -\nabla\phi$$

$$\nabla^2\phi = -\frac{\rho}{\epsilon}$$

Numerical Methods for Linear Algebra

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

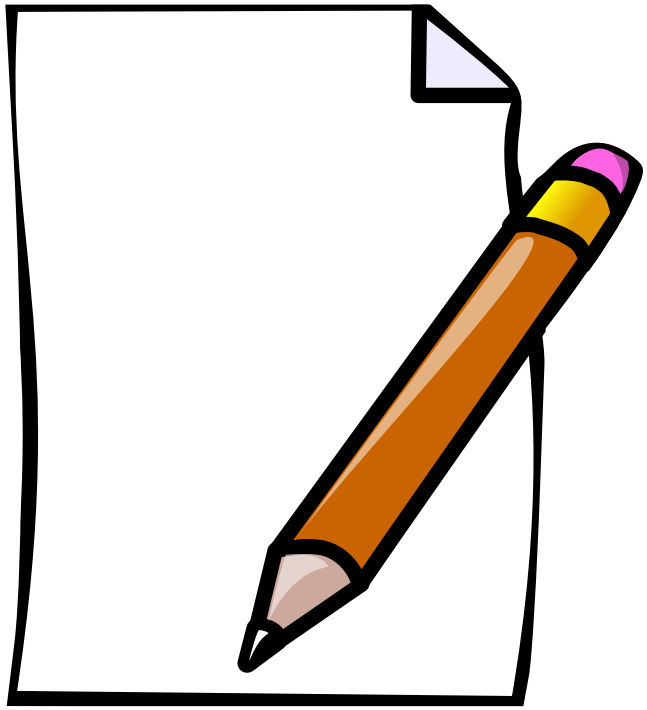




Notes and slides: "NM4LA-1/2" on TISS as pdf files

Today

- Eigenvalues and Eigenvectors computation



25.04.2022

Computation of eigenvalues and eigenvectors

Reminder on the eigenvalue and eigenvector problem in linear algebra

- Almost all vectors **x** change direction when they are multiplied by a matrix A: $A\vec{x} = \vec{b}$
- Certain exceptional vectors are in the same direction as A**x: eigenvectors**
- When you multiply an eigenvector by A, the vector A**x** is a number times the original:

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

- The number λ is an **eigenvalue** of A
- The eigenvalue tells whether the special vector is stretched or shrunk or reversed or left unchanged when it is multiplied by A

When do eigenvalues and eigenvectors emerge in Physics? Always!

- Find the normal modes in crystalline solids (e.g. elastic properties)
- Solve the time-dependent Schrödinger equation (e.g. ground states of atomic systems)
- Determine the shape parameters characterizing large macromolecules

Computation of eigenvalues and eigenvectors

Reminder on the eigenvalue and eigenvector problem in linear algebra

- Almost all vecto… $\vec{x} = \vec{b}$
- Certain exceptio…
- When you multi… ginal:

The solution of eigensystems is a fairly complicated business!

Rely on publicly available routines!

The purpose of this lecture is to give you some appreciation of what is going on inside such routines, so that you can make intelligent choices about using them and intelligent diagnoses when something goes wrong.

- The number λ is …
- The eigenvalue t… or left unchanged when it is multip…

…s!

- Find the normal …
- Solve the time-de… …tems)
- Determine the shape parameters characterizing large macromolecules

Basic facts and definitions

An *nxn* matrix A has an eigenvector **x** if

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

Basic facts and definitions

An *nxn* matrix A has an eigenvector **x** if

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

The associated characteristic polynomial p(λ) must be zero

$$\det|A - \lambda\mathbb{1}| = 0$$

Basic facts and definitions

An *nxn* matrix A has an eigenvector **x** if

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

The associated characteristic polynomial p(λ) must be zero

$$\det|A - \lambda\mathbb{1}| = 0$$

For each eigenvalue the corresponding eigenvector is calculated by

$$(A - \lambda\mathbb{1})\vec{x} = 0$$

The eigenvalue problem could be solved by searching for the roots of p(λ)
…
but this is not recommended as soon as *n* is bigger than 2!

A = normal matrix (Hermitian matrix & real symmetric matrix):

- real eigenvalues
  - distinct eigenvalues: eigenvectors form a complete and orthogonal basis
  - degenerate eigenvalues: associated eigenvectors can be orthogonalized (Gram-Schmidt)

Basic facts and definitions

Improvement of the eigenvalue calculation via the following transformations

Shift	$(A-\sigma\mathbb{1})\vec{x}$	$=$	$(\lambda-\sigma)\vec{x}$
Inversion	$A^{-1}\vec{x}$	$=$	$\lambda^{-1}\vec{x}$
Power	$A^2\vec{x}$	$=$	$\lambda^2\vec{x}$
Polynomial expansion	$p_n(A)\vec{x}$	$=$	$p_n(\lambda)\vec{x}$

Basic facts and definitions

Improvement of the eigenvalue calculation via the following transformations

$$
\begin{array}{lrcl}
\text{Shift} & (A-\sigma\mathbb{1})\vec{x} & = & (\lambda-\sigma)\vec{x} \\
\text{Inversion} & A^{-1}\vec{x} & = & \lambda^{-1}\vec{x} \\
\text{Power} & A^{2}\vec{x} & = & \lambda^{2}\vec{x} \\
\text{Polynomial expansion} & p_n(A)\vec{x} & = & p_n(\lambda)\vec{x}
\end{array}
$$

We will often rely on similarity transformations

$$\boxed{A \rightarrow A' = Z^{-1}AZ}$$

as such transformation leaves the eigenvalues of a matrix unchanged

$$
\begin{aligned}
\det|Z^{-1}AZ-\lambda\mathbb{1}| &= \det|Z^{-1}(A-\lambda\mathbb{1})Z| \\
&= \det|Z|\det|(A-\lambda\mathbb{1})|\det|Z^{-1}| \\
&= \det|(A-\lambda\mathbb{1})|
\end{aligned}
$$

Condition number of the eigenvalues 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

condition stability

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

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*

$$\max_{e,b\neq 0}\left\{\frac{||A^{-1}e||}{||e||}\frac{||b||}{||A^{-1}b||}\right\} = \max_{e\neq 0}\left\{\frac{||A^{-1}e||}{||e||}\right\}\max_{b\neq 0}\left\{\frac{||b||}{||A^{-1}b||}\right\} =$$

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

$$= ||A^{-1}||\,||A||$$

Condition number of the eigenvalues 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

condition stability

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

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*

$$\max_{e,b\neq 0}\left\{\frac{||A^{-1}e||}{||e||}\frac{||b||}{||A^{-1}b||}\right\} = \max_{e\neq 0}\left\{\frac{||A^{-1}e||}{||e||}\right\}\max_{b\neq 0}\left\{\frac{||b||}{||A^{-1}b||}\right\} =$$

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

$$= ||A^{-1}||\,||A||$$

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

- ~ 1 then *A* is well-conditioned
- >> 1 then *A* is ill-conditioned (e.g., A not invertible)

Condition number of the eigenvalues problem

$$K(A) = ||A||||A^{-1}||$$

What is the (induced) **norm of matrix** A?
Measure of how much matrix A magnifies the length of a nonzero vector **x**

$$||A|| \equiv \sup_{x \neq 0} \frac{||Ax||}{||x||}$$

Proof The proof depends on a result from real analysis (sometimes called "advanced calculus") that states that $\sup_{x \in S} f(x)$ is attained for some vector $x \in S$ as long as f is continuous and S is a compact (closed and bounded) set. For any norm $\|\cdot\|$, the unit ball $\|x\| = 1$ is a compact set. When a supremum is an element in S, it is called the maximum instead and $\sup_{x \in S} f(x)$ can be restated as $\max_{x \in S} f(x)$.

Condition number of the eigenvalues problem

$$K(A) = ||A||||A^{-1}||$$

What is the (induced) **norm of matrix** A?
Measure of how much matrix A magnifies the length of a nonzero vector **x**

$$||A|| \equiv \sup_{x \neq 0} \frac{||Ax||}{||x||}$$

Proof The proof depends on a result from real analysis (sometimes called "advanced calculus") that states that $\sup_{x \in S} f(x)$ is attained for some vector $x \in S$ as long as f is continuous and S is a compact (closed and bounded) set. For any norm $\|\cdot\|$, the unit ball $\|x\| = 1$ is a compact set. When a supremum is an element in S, it is called the maximum instead and $\sup_{x \in S} f(x)$ can be restated as $\max_{x \in S} f(x)$.

$$||A|| \equiv \max_{||x||=1} ||Ax||$$

$$||A^{-1}|| \equiv \max_{||x||=1} ||A^{-1}x||$$

Condition number of the eigenvalues problem

$$K(A) = ||A||||A^{-1}||$$

$$||A|| \equiv \max_{||x||=1} ||Ax||$$

$$||A^{-1}|| \equiv \max_{||x||=1} ||A^{-1}x||$$

If A has only real eigenvalues $\lambda_1, \ldots, \lambda_n$ and if there is an orthonormal basis of eigenvectors $x_1, \ldots, x_n$, then $Ax_i = \lambda_i x_i$, *i.e.*

$$AQ = Q\Lambda \quad \text{where} \quad \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n) \quad \text{and} \quad Q = [x_1, \ldots, x_n]$$

$$A^{-1} = Q\Lambda^{-1}Q^T$$

Condition number of the eigenvalues problem

$$K(A) = ||A||||A^{-1}||$$

$$||A|| \equiv \max_{||x||=1} ||Ax||$$

$$||A^{-1}|| \equiv \max_{||x||=1} ||A^{-1}x||$$

If A has only real eigenvalues $\lambda_1, \ldots, \lambda_n$ and if there is an orthonormal basis of eigenvectors $x_1, \ldots, x_n$, then $Ax_i = \lambda_i x_i$, *i.e.*

$$AQ = Q\Lambda \quad \text{where} \quad \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n) \quad \text{and} \quad Q = [x_1, \ldots, x_n]$$

$$A^{-1} = Q\Lambda^{-1}Q^T$$

$$||A|| \equiv \max_{||x||=1} ||Ax|| = \max|\lambda_i|$$

$$||A^{-1}|| \equiv \max_{||x||=1} ||A^{-1}x| = 1/\min|\lambda_i|$$

$$K(A) = \frac{\max|\lambda_i|}{\min|\lambda_i|}$$

The calculation with matrices becomes more difficult the **broader** the eigenvalue spectrum is

Condition number of the eigenvalues problem

$$K(A) = \frac{\max|\lambda_i|}{\min|\lambda_i|}$$

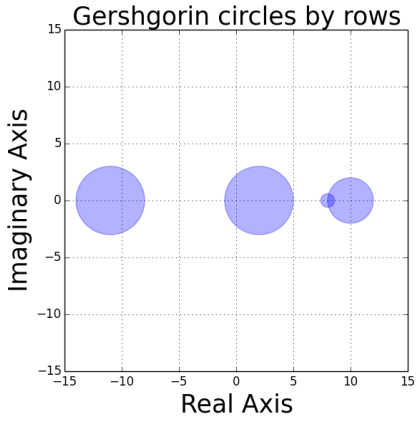
A first estimate of the eigenvalue spectrum is given by the **Gershgorin circles theorem**

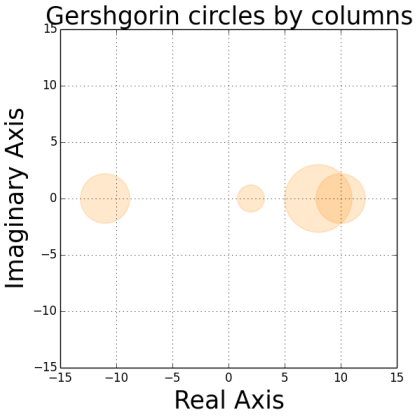
$$|\lambda_i - c_{ii}| \leq \sum_{j \neq i} |c_{ij}| \quad \text{by rows}$$

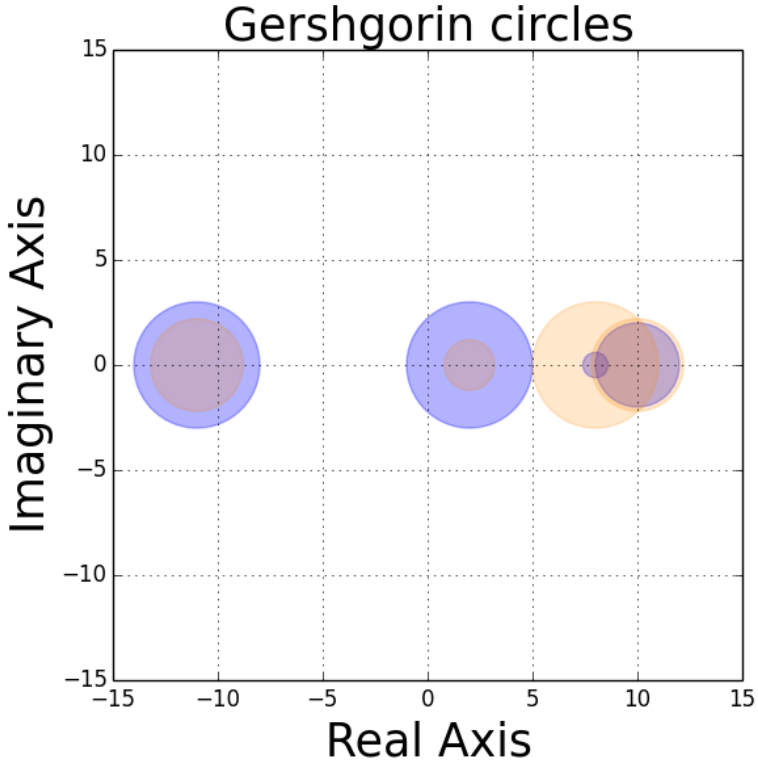
$$|\lambda_i - c_{ii}| \leq \sum_{j \neq i} |c_{ji}| \quad \text{by columns}$$

$$A = \begin{pmatrix} 10 & -1 & 0 & 1 \\ 0.2 & 8 & 0.2 & 0.2 \\ 1 & 1 & 2 & 1 \\ -1 & -1 & -1 & -11 \end{pmatrix}$$

$$|\lambda_1 - 10| \leq 2 \quad \text{and} \quad |\lambda_1 - 10| \leq 2.2$$
$$|\lambda_2 - 8| \leq 0.6 \quad \text{and} \quad |\lambda_2 - 8| \leq 3$$
$$|\lambda_3 - 2| \leq 3 \quad \text{and} \quad |\lambda_3 - 2| \leq 1.2$$
$$|\lambda_4 + 11| \leq 3 \quad \text{and} \quad |\lambda_4 + 11| \leq 2.2$$







The grand strategy to deal with eigensystem problems

Any matrix with a complete set of eigenvectors (normal matrices) can be diagonalized by a similarity transformation!

Eigensystem routines bring A towards its diagonal form by a sequence of similarity transformations

$$A \to P_1^{-1} A P_1 \to P_2^{-1} P_1^{-1} A P_1 P_2 \to P_3^{-1} P_2^{-1} P_1^{-1} A P_1 P_2 P_3 \to \dots$$

If we get all the way to the diagonal form, then

$$X_R = P_1 P_2 P_3 \dots$$

$$X_R^{-1} A X_R = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$$

- eigenvectors = columns of the accumulated transformation matrix
- eigenvalues = the non-zero, diagonal elements of the final matrix

The grand strategy to deal with eigensystem problems

Any matrix with a complete set of eigenvectors (normal matrices) can be diagonalized by a similarity transformation!

Eigensystem routines bring A towards its diagonal form by a sequence of similarity transformations

$$A \to P_1^{-1} A P_1 \to P_2^{-1} P_1^{-1} A P_1 P_2 \to P_3^{-1} P_2^{-1} P_1^{-1} A P_1 P_2 P_3 \to \dots$$

If we get all the way to the diagonal form, then

$$X_R = P_1 P_2 P_3 \dots$$

$$X_R^{-1} A X_R = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$$

- eigenvectors = columns of the accumulated transformation matrix
- eigenvalues = the non-zero, diagonal elements of the final matrix

- all eigenvalues and all eigenvectors
  - A diagonal: **Jacobi transformation**
  - A tridiagonal or triangular form:
  - ◉ **Given/Householder reduction** + **recursion** (eigenvalues) and **inverse iteration** (eigenvectors)
  - ◉ **QR algorithm** (eigenvalues and eigenvectors)

- only the maximum (or minimum) eigenvalue (and associated eigenvector): **power method**

Jacobi transformation: from symmetric to diagonal

A is a real and symmetric matrix

$$A = \begin{pmatrix} * & & \cdots & & * \\ & \ddots & & & \\ & & a_{pp} \; \cdots \; a_{pq} & & \\ \vdots & & \vdots \quad \ddots \quad \vdots & & \vdots \\ & & a_{qp} \; \cdots \; a_{qq} & & \\ & & & \ddots & \\ * & & \cdots & & * \end{pmatrix} \rightarrow A' = \begin{pmatrix} * & & \cdots & & * \\ & \ddots & & & \\ & & a'_{pp} \; \cdots \; 0 & & \\ \vdots & & \vdots \quad \ddots \quad \vdots & & \vdots \\ & & 0 \; \cdots \; a'_{qq} & & \\ & & & \ddots & \\ * & & \cdots & & * \end{pmatrix}$$

$$A \rightarrow A' = P_{pq}^T A P_{pq}$$

a Jacobi transformation is a plane rotation designed to annihilate one of the off-diagonal elements

$$P_{pq} = \begin{pmatrix} 1 & & & & & & \\ & \cdots & & & & & \\ & & c & \cdots & s & & \\ & & \vdots & 1 & \vdots & & \\ & & -s & \cdots & c & & \\ & & & & & \cdots & \\ & & & & & & 1 \end{pmatrix}$$

c and s are the cosine and sine of the rotation angle Φ

$$c^2+s^2=1$$

the matrix P_{pq} describes a rotation with angle Φ in the (p,q)-plane

Jacobi transformation: from symmetric to diagonal

$$A' = \begin{pmatrix} & \cdots & a'_{1p} & \cdots & a'_{1q} & \cdots & \\ \vdots & & \vdots & & \vdots & & \vdots \\ a'_{p1} & \cdots & a'_{pp} & \cdots & a'_{pq} & \cdots & a'_{pn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a'_{q1} & \cdots & a'_{qp} & \cdots & a'_{qq} & \cdots & a'_{qn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ & \cdots & a'_{np} & \cdots & a'_{nq} & \cdots & \end{pmatrix}$$

$$\begin{aligned} a'_{rp} &= ca_{rp} - sa_{rq} \quad r \neq p, r \neq q \\ a'_{rq} &= ca_{rq} + sa_{rp} \quad r \neq p, r \neq q \\ a'_{pp} &= c^2 a_{pp} + s^2 a_{qq} - 2sca_{pq} \\ a'_{qq} &= s^2 a_{pp} + c^2 a_{qq} + 2sca_{pq} \\ a'_{pq} &= (c^2 - s^2)a_{pq} + sc(a_{pp} - a_{qq}) \end{aligned}$$

If we set $a'_{pq} = 0$ then $\cot 2\phi \equiv \dfrac{c^2 - s^2}{2sc} = \dfrac{a_{qq} - a_{pp}}{2a_{pq}}$

Jacobi transformation: from symmetric to diagonal

$$A' = \begin{pmatrix} & \cdots & a'_{1p} & \cdots & a'_{1q} & \cdots & \\ \vdots & & \vdots & & \vdots & & \vdots \\ a'_{p1} & \cdots & a'_{pp} & \cdots & a'_{pq} & \cdots & a'_{pn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a'_{q1} & \cdots & a'_{qp} & \cdots & a'_{qq} & \cdots & a'_{qn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ & \cdots & a'_{np} & \cdots & a'_{nq} & \cdots & \end{pmatrix}$$

$$\begin{aligned} a'_{rp} &= ca_{rp} - sa_{rq} \quad r \neq p, r \neq q \\ a'_{rq} &= ca_{rq} + sa_{rp} \quad r \neq p, r \neq q \\ a'_{pp} &= c^2 a_{pp} + s^2 a_{qq} - 2sca_{pq} \\ a'_{qq} &= s^2 a_{pp} + c^2 a_{qq} + 2sca_{pq} \\ a'_{pq} &= (c^2 - s^2)a_{pq} + sc(a_{pp} - a_{qq}) \end{aligned}$$

If we set $a'_{pq} = 0$ then $\cot 2\phi \equiv \dfrac{c^2 - s^2}{2sc} = \dfrac{a_{qq} - a_{pp}}{2a_{pq}}$

A sequence of successive transformations undo previously set zeros, but the off-diagonal elements nevertheless get smaller and smaller, until the matrix is diagonal to machine precision.

$$D = V^T A V \qquad V = P_1 P_2 P_3 \ldots$$

eigenvalues = the non-zero, diagonal elements of *D*
eigenvectors = columns of *V*

Jacobi transformation: from symmetric to diagonal

What is the best order of annihilation?

- Jacobi original (1846): search for the largest off-diagonal element in the whole upper triangular matrix and set it to zero

- Cyclic Jacobi: elements put to zero in strict order (*e.g.* down the rows P_{12}, P_{13}, …, P_{1n}, P_{23}, P_{24}…)

Iteration until convergence: the finite sequence of transformations is iterated over and over until the deviation of the matrix from a diagonal form is negligible

Jacobi method is computationally inefficient already for n=10!

Givens reduction: from symmetric to tridiagonal

A is a real and symmetric matrix

The rotation angle puts to zero an element that is **not** at one of the four "corners"

$$A = \begin{pmatrix} * & & \cdots & & * \\ & \ddots & & & \\ & & a_{pp} & \cdots & a_{pq} \\ \vdots & & \vdots & \ddots & \vdots & \vdots \\ & & a_{qp} & \cdots & a_{qq} & \\ & & & & \ddots & \\ * & & \cdots & & * \end{pmatrix}$$

$$\begin{aligned} a'_{rp} &= ca_{rp} - sa_{rq} \quad r \neq p, r \neq q \\ a'_{rq} &= ca_{rq} + sa_{rp} \quad r \neq p, r \neq q \\ a'_{pp} &= c^2 a_{pp} + s^2 a_{qq} - 2sca_{pq} \\ a'_{qq} &= s^2 a_{pp} + c^2 a_{qq} + 2sca_{pq} \\ a'_{pq} &= (c^2 - s^2)a_{pq} + sc(a_{pp} - a_{qq}) \end{aligned}$$

The method works because a'_{rp} and a'_{rq} are linear combinations of a_{rp} and a_{rq}, if those are already set to zero, they remain zero as the reduction proceeds.

$$a'_{q-1,p} = 0 = ca_{q-1,p} - sa_{q-1,q} \rightarrow \tan\phi = \frac{a_{q-1,p}}{a_{q-1,q}}$$

Householder reduction: from symmetric to tridiagonal

A is a real and symmetric matrix

$$H = \mathbb{1} - 2\frac{\vec{u}\cdot\vec{u}^T}{|\vec{u}|^2}$$

$$\vec{u} = \vec{a_i} \mp |\vec{a_i}|\vec{e}_1$$

1st column of A unit vector $(1, 0, \ldots, 0)^T$

$$\begin{aligned} H\vec{a_i} &= \vec{a_i} - 2\frac{\vec{u}}{|\vec{u}^2|}(\vec{a_i} \mp |\vec{a_i}|\vec{e}_1)^T\vec{a_i} \\ &= \vec{a_i} - \frac{2\vec{u}(|\vec{a_i}|^2 \mp |\vec{a_i}|a_1)}{(|2\vec{a_i}|^2 \mp 2|\vec{a_i}|a_1)} \\ &= \vec{a_i} - \vec{u} \\ &= \pm|\vec{a_i}|\vec{e}_1 \end{aligned}$$

H sets to zero all elements of a given vector except the first one

Householder reduction: from symmetric to tridiagonal

The vector for the first Householder matrix is the lower (n-1) elements of the 1st column of A

$$\vec{u}_1 = \begin{pmatrix} a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{pmatrix} \mp \left\| \begin{pmatrix} a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{pmatrix} \right\| \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \rightarrow H_1^{(n-1)} = \mathbb{1} - \frac{2\vec{u}_1 \cdot \vec{u}_1^T}{|\vec{u}_1|^2}$$

Householder matrix with dimensions (n-1)x(n-1)

$$H_1 A = \left(\begin{array}{c|ccc} 1 & 0 & \dots & 0 \\ \hline 0 & & & \\ 0 & & & \\ \vdots & & H_1^{(n-1)} & \\ 0 & & & \end{array}\right) \left(\begin{array}{c|ccc} a_{11} & a_{12} & \dots & a_{1n} \\ \hline a_{21} & & & \\ a_{31} & & & \\ \vdots & & a_{ij} & \\ a_{n1} & & & \end{array}\right) = \left(\begin{array}{c|ccc} a_{11} & a_{12} & \dots & a_{1n} \\ \hline k & & & \\ 0 & & & \\ \vdots & & a'_{ij} & \\ 0 & & & \end{array}\right)$$

Complete orthogonal transformation

$$A' = H_1 A H_1 = \left(\begin{array}{c|ccc} a_{11} & k & 0 & \dots & 0 \\ \hline k & a'_{22} & a'_{23} & \dots & a'_{2n} \\ 0 & a'_{32} & a'_{33} & \dots & a'_{3n} \\ \vdots & \vdots & & & \vdots \\ 0 & a'_{n2} & a'_{n3} & \dots & a'_{nn} \end{array}\right)$$

The lower (n-2) elements of column 1 are zeroed — k is simply plus or minus the magnitude of the vector $(a_{21}, \dots, a_{n1})^T$

Householder reduction: from symmetric to tridiagonal

$$\vec{u}_2 = \begin{pmatrix} a_{32} \\ a_{42} \\ \vdots \\ a_{n2} \end{pmatrix} \mp \left\| \begin{pmatrix} a_{32} \\ a_{42} \\ \vdots \\ a_{n2} \end{pmatrix} \right\| \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \rightarrow H_2^{(n-2)} = \mathbb{1} - \frac{2\vec{u}_2 \cdot \vec{u}_2^T}{|\vec{u}_2|^2}$$

$$H_2 = \left(\begin{array}{cc|ccc} 1 & 0 & 0 & \dots & 0 \\ \hline 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & & & \\ \vdots & & & H_2^{(n-2)} & \\ 0 & 0 & & & \end{array}\right)$$

$$A'' = H_2 A' H_2 = H_2 H_1 A H_1 H_2$$

Note: the identity block in the upper left corner guarantees that the tridiagonalization achieved in the first step is not spoiled by the second step

Householder method reduces *A* to a tridiagonal form by *(n-2)* orthogonal transformations

Eigenvalues of a tridiagonal matrix: recursion

$$p(\lambda) = \det \begin{pmatrix} a_{11}-\lambda & a_{21} & 0 & 0 & \dots & \dots & 0 \\ a_{21} & a_{22}-\lambda & a_{23} & 0 & \dots & \dots & 0 \\ 0 & a_{32} & a_{33}-\lambda & a_{34} & 0 & \dots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & \dots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & a_{n-1n-2} & a_{n-1n-1}-\lambda & a_{n-1n} \\ 0 & \dots & \dots & \dots & 0 & a_{nn-1} & a_{nn}-\lambda \end{pmatrix}$$

The characteristic polynomial of a tridiagonal matrix can be evaluated for any trial value of λ by the **recursion relation**

$$p_i(\lambda) = (a_{ii} - \lambda)p_{i-1}(\lambda) - a_{ii-1}^2 p_{i-2}(\lambda)$$

Trial λ by the Gershgorin circles

$$\min\left\{a_{ii} - \sum_{j\neq i}|a_{ij}|\right\} \leq \lambda \leq \max\left\{a_{ii} + \sum_{j\neq i}|a_{ij}|\right\}$$

Computation of eigenvectors: inverse iteration

System of linear equations to be solved by LU-factorization

$$(A - (\lambda_i + \epsilon)\mathbb{1})\vec{x'}_i^{(j+1)} = \vec{x}_i^{(j)}$$

Attention: system nearly singular!

$$\vec{x}_i^{(j+1)} = \frac{\vec{x'}_i^{(j+1)}}{|\vec{x'}_i^{(j+1)}|}$$

Why does it work?

Computation of eigenvectors: inverse iteration

System of linear equations to be solved by LU-factorization

$$(A - (\lambda_i + \epsilon)\mathbb{1})\vec{x'}_i^{(j+1)} = \vec{x}_i^{(j)}$$

Attention: system nearly singular!

$$\vec{x}_i^{(j+1)} = \frac{\vec{x'}_i^{(j+1)}}{|\vec{x'}_i^{(j+1)}|}$$

When to use it?
well-known, eigenvalues and few selected eigenvectors needed

First reduce A to a simple form!

Why does it work?

$$\vec{x'}_i^{(j+1)} = \sum_k \alpha_k^{(j)} \vec{x}_k \quad \text{and} \quad \vec{x}_i^{(j)} = \sum_k \beta_k^{(j)} \vec{x}_k$$

$$\sum_k \alpha_k^{(j)} (\lambda_k - \lambda_i - \epsilon)\vec{x}_k = \sum_k \beta_k^{(j)} \vec{x}_k$$

$$\alpha_k = \frac{\beta_k^{(j)}}{\lambda_k - \lambda_i - \epsilon} \qquad \vec{x'}_i^{(j+1)} = \sum_k \frac{\beta_k^{(j)} \vec{x}_k}{\lambda_k - \lambda_i - \epsilon}$$

When $\epsilon \to 0$, then the sum is dominated by the term $\lambda_i = \lambda_k$

Eigenvalues of a normal matrix: QR algorithm

Note: now A is not necessarily symmetric!

A decomposed into the product of an orthogonal and an upper triangular matrix

$$A = QR \qquad R = Q^T A \qquad A' = RQ = Q^T AQ$$

The QR-algorithm consists of a sequence of orthogonal transformations

$$\begin{aligned} A^{(j)} &= Q^{(j)} R^{(j)} \\ A^{(j+1)} &= R^{(j)} Q^{(j)} = Q^{(j)T} A^{(j)} Q^{(j)} \end{aligned}$$

The decomposition is constructed by applying orthogonal similarity transformations to annihilate successive columns of A below the diagonal

$$A = \begin{pmatrix} * & * & * & \dots & * \\ * & * & * & \dots & * \\ * & * & * & \dots & * \\ \vdots & & \ddots & & \vdots \\ * & \dots & * & * & * \end{pmatrix} \rightarrow A' = \begin{pmatrix} * & * & * & \dots & * \\ 0 & * & * & \dots & * \\ 0 & 0 & * & \dots & * \\ \vdots & & \ddots & & \vdots \\ 0 & \dots & 0 & 0 & * \end{pmatrix}$$

- Givens rotation sets to zero a single off-diagonal element in the matrix
- Householder rotation sets to zero a whole particular row or column

Eigenvalues of a tridiagonal matrix: QR algorithm

$$A^{(j\to\infty)} = \begin{pmatrix} \lambda_1 & * & * & & \dots & * \\ 0 & \lambda_2 & * & & \dots & \vdots \\ 0 & 0 & \lambda_3 & & & \\ \vdots & & \ddots & \ddots & & \\ \vdots & & & & & \\ 0 & \dots & & & 0 & \lambda_n \end{pmatrix}$$

A with non-degenerate eigenvalues

Note that:
similarity transformations keep eigenvalues
eigenvalues of a triangular matrix are the diagonal elements

A with a degenerate eigenvalue of multiplicity p

$$A^{(j\to\infty)} = \begin{pmatrix} \lambda_1 & * & * & \dots & & * \\ 0 & \lambda_2 & * & \dots & & \vdots \\ 0 & 0 & \lambda_3 & \dots & & \\ \vdots & & \ddots & \boxed{\begin{matrix} a_{ii} & \dots & a_{ii+p} \\ \vdots & \ddots & \vdots \\ a_{i+pi} & \dots & a_{i+pi+p} \end{matrix}} & & * \\ \vdots & & & & \ddots & \\ 0 & \dots & & & 0 & \lambda_n \end{pmatrix}$$

p-degenerate eigenvalue

Note: you recall the eigenvectors as columns of $\prod iQi$.

Note: QR method is computationally inefficient for sparse matrices!

Eigenvalues of a sparse matrix: Power method

A is a real and symmetric matrix

A with eigenvalues λ_i and eigenvectors $\vec{x}_i$:

$$\lambda_1 = \lambda_{\max} \qquad \lambda_n = \lambda_{\min} \qquad |\lambda_1| \gg |\lambda_i| \text{ for } i = 2, \ldots, n$$

Rayleigh quotient

$$\rho(\vec{z}, A) = \frac{\vec{z}^T A \vec{z}}{\vec{z}^T \vec{z}}$$

$$\max \rho(\vec{z}, A) = \lambda_1 \qquad \min \rho(\vec{z}, A) = \lambda_n$$

if an eigenvector

$$\rho(\vec{z}, A) = \frac{\vec{z}^T A \vec{z}}{\vec{z}^T \vec{z}} = \frac{\lambda \vec{z}^T \vec{z}}{\vec{z}^T \vec{z}} = \lambda$$

if not $\vec{z} = \sum_i c_i \vec{x}_i$

$$V = (\vec{x}_1, \ldots, \vec{x}_n)$$

$$V^T A V = \mathrm{diag}(\lambda_1, \ldots, \lambda_n),$$

$$\rho(\vec{z}, A) = \frac{\vec{z}^T V^T A V \vec{z}}{\vec{z}^T \vec{z}} = \frac{\sum_i c_i^2 \lambda_i}{\sum_i c_i^2}$$

weighted average dominated by the largest eigenvalue

Eigenvalues of a sparse matrix: Power method

Initial vector

$$\vec{z}^{(0)} = \sum_i c_i \vec{x}_i$$

Iteration

$$\vec{z}^{(j)} = A\vec{z}^{(j-1)} = A^{(j)}\vec{z}^{(0)} = \sum_i \lambda_i^{(j)} c_i \vec{x}_i$$

$$= \lambda_1^{(j)} \left(c_1 x_1 + c_2 x_2 \left(\frac{\lambda_2}{\lambda_1}\right)^{(j)} + \cdots + c_n x_n \left(\frac{\lambda_n}{\lambda_1}\right)^{(j)} \right)$$

Since $\lim_{j\to\infty} \left(\frac{\lambda_i}{\lambda_1}\right)^{(j)} = 0$ then on increasing j $\vec{z}^{(j)} = \lambda_1^{(j)} c_1 \vec{x}_1$

$$\rho(\vec{z}, A) = \frac{\vec{z}^{(j)T} A \vec{z}^{(j)}}{\vec{z}^{(j)T} \vec{z}^{(j)}} = \frac{\vec{z}^{(j)T} \vec{z}^{(j+1)}}{\vec{z}^{(j)T} \vec{z}^{(j)}} = \lambda_1 + O\left(\left| \frac{\lambda_i}{\lambda_1} \right|^{2j} \right)$$

Note: minimum eigenvalue and eigenvector by inverse power method (power method to A^{-1})

Eigenvalues of a sparse matrix: Power method

Define z_0
Calculate the unit vector y_0=$z_0/|z_0|$
Calculate the new vector $z_1 = Ay_0$
Calculate the first estimate of $\lambda = y_0 z_1/(y_0 y_0)$
Define a convergence criteria $tol = 10^{-6}$

do while (err.gt.(tol*abs(lambda)).and.(k.lt.100))

Calculate the new unit vector $y_k = z_k/|z_k|$ with $k = 1, ...kmax$
Calculate the new vector $z_{k+1} = Ay_k$
Calculate the new estimate of $\lambda = y_k * z_{k+1}/(y_k y_k)$

err=$|\lambda' - \lambda|$
$\lambda = \lambda'$
k=k+1

end do

The maxmimum eigenvalue is λ' after k iterations
The associated eigenvector is y_k after k iterations

The grand strategy to deal with eigensystem problems

• all eigenvalues and all eigenvectors

- A diagonal:
◉ **Jacobi transformation**

- A tridiagonal or triangular form:
◉ **Given/Householder reduction + recursion** (eigenvalues) and **inverse iteration** (eigenvectors)
◉ **QR algorithm** (eigenvalues and eigenvectors)

• only the maximum (or minimum) eigenvalue and associated eigenvector
◉ **Power method**

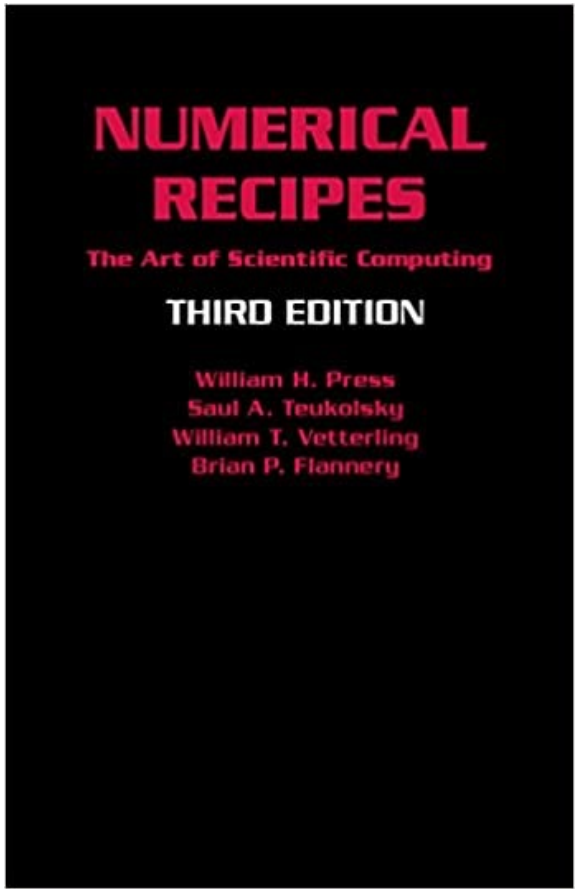
That’s all for Today!

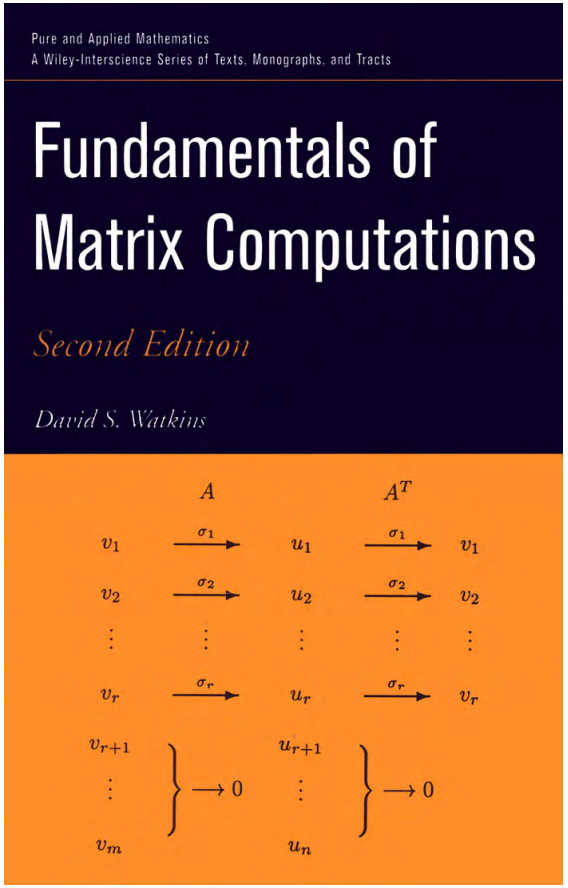
- Eigenvalues and Eigenvectors computation
- Exercise 2

25.04.2022

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

- Eigenvalues and Eigenvectors computation
- Exercise 2

25/26.04.2022

Today

Given the matrix A

$$A = \begin{pmatrix} 6 & -2 & 3 \\ 1 & 10 & -4 \\ -2 & 1 & 0 \end{pmatrix}$$

1. find the geometric estimate of its eigenvalues by the Gershgorin circles theorem and plot the circles in the complex plane. On this basis, provide an estimate of the condition number of the problem; is the problem well-conditioned?

2. find its largest eigenvalue by applying the power method, with initial vector $z_0 = (1, 1, 1)^T$ and convergence criteria $\epsilon = 10^{-6}$. Tabulate and plot the sequence of λ_{max}-values as a function of the iteration step i, until convergence is reached within the chosen criterium; also tabulate and plot the difference $|\lambda_i - \lambda_{i-1}|$ at each iteration step and check that it goes to zero within the chosen criterium. How many steps do you need to reach convergence within the chosen tolerance criterium?

3. find its complete set of eigenvalues with a self-developed QR-algorithm and with a publicly available routine. How many steps do you need to reach convergence within a tolerance of $\epsilon = 10^{-6}$? Plot the sequence of $(\lambda_1, \lambda_2, \lambda_3)$-values as a function of the iteration step i, until convergence is reached within the chosen criterium. Add $(\lambda_1, \lambda_2, \lambda_3)$ to the plot(s) of the Gershgorin circles.