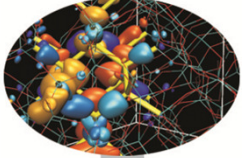
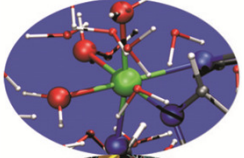
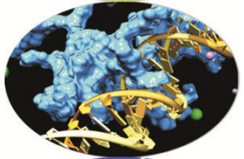
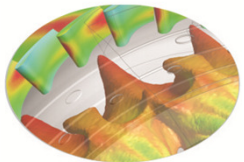
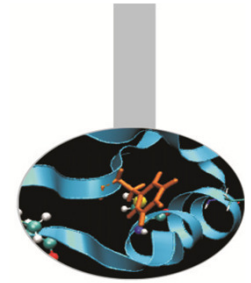


Numerical libraries

- Exercises -

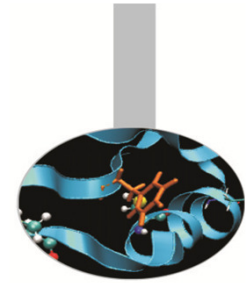




Library installation

As a beginning effort try download and install LAPACK from NETLIB:

- Download from <http://www.netlib.org/lapack/lapack-3.4.1.tgz>
- Configure *make.inc* (just copy *make.inc.example* for GNU compilers)
- `cd SRC`
- Run *make*
- `cd ../lapacke`
- Run *make*
- `cd ../BLAS/SRC`
- Run *make*

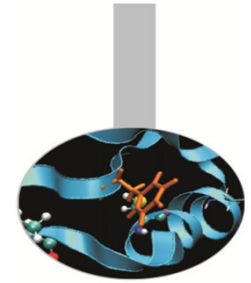


E1 – exercise – Matrix Multiply

It should be straightforward, although usually rather inefficient to implement a matrix multiplication routine. Try doing it and later add an alternative implementation that uses the BLAS routine:

```
FUNCTION DDOT(N,X,INCX,Y,INCY) RESULT(XtY)
  REAL(8) :: XtY
  INTEGER, INTENT(IN) :: N, INCX, INCY
  REAL(8), DIMENSION(:), INTENT(IN) :: X,Y
END FUNCTION DDOT
```

Source code: *MatrMult*



E2 – example - Rotation

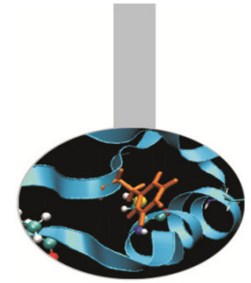
Wallace Givens was a researcher at Argonne National Laboratory when he introduced the so called *Givens rotation algorithm*, a rotation in the plane that is useful for transforming matrices in linear algebra computations.

The BLAS library contains routines to compute plane rotations and we could use them with the goal of rotating geometrical figures.

As an example, given a matrix $PLANE(:, :)$ that represents a rectangle in a bi-dimensional plane and a set of points within the rectangle that defines a figure, at first we compute the angular parameters of the rotation, given by $c = \cos(\theta)$ and $s = \sin(\theta)$, then we calculate the result of the rotation applied to the figure.

The already used routine `IntData2pgm()` will help to visualize the procedure.

Source code: *Rotation*



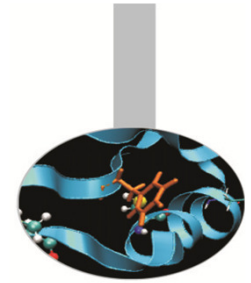
Matrix norms

Consider \mathcal{F} to be the space of *Real* or *Complex* numbers and $\mathcal{F}_{m,n}$ the space of matrices with $M \times N$ elements belonging to \mathcal{F} .

A matrix norm is a function $\|\cdot\| : \mathcal{F}_{m,n} \rightarrow \mathcal{F}$ with the following properties:

- $\|A\| = 0$ iff $A = 0$
- $\|A\| \geq 0$
- $\|x \cdot A\| = |x| \cdot \|A\|$
- $\|A+B\| \leq \|A\| + \|B\|$

With: $A, B \in \mathcal{F}_{m,n}$; $x \in \mathcal{F}$



Matrix norms

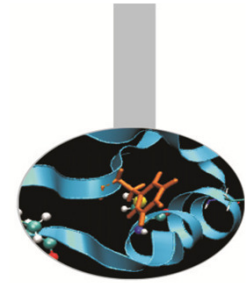
It follows that $(A, B \in \mathcal{F}_{m,n} ; x, y \in \mathcal{F})$:

- $|x| \cdot ||A||' \leq |x| \cdot ||A||'' \leq |y| \cdot ||A||'$ [equivalence of matrix norm]
- $||\cdot|| : \mathcal{F}_{m,n} \rightarrow \mathcal{F}$ is a continuous function

Relevant norms:

$$||A||_1 = \text{Max}_{1 \leq j \leq N} \text{Sum}_{1 \leq i \leq M} |A_{i,j}| \text{ [max abs sum col elements]}$$

$$||A||_\infty = \text{Max}_{1 \leq i \leq M} \text{Sum}_{1 \leq j \leq N} |A_{i,j}| \text{ [max abs sum row elements]}$$



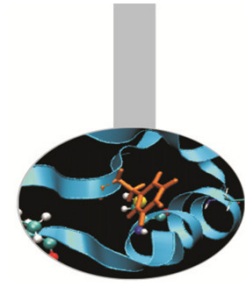
Condition number

A system of linear equations is solved when a vector $x \in \mathcal{F}_n$ is found such that $A \cdot x = b$ for given $A \in \mathcal{F}_{n,n}$ and $b \in \mathcal{F}_n$.

Suppose that $A \cdot y = b + e$ for a given e : $\|e\| \ll \|b\|$

It follows that $\|y - x\| = \|A^{-1} \cdot e\|$ and

$$\frac{\|y - x\| / \|x\|}{\|e\| / \|b\|} = \frac{\|A^{-1} \cdot e\| / \|A^{-1} \cdot b\|}{\|e\| / \|b\|} = \|A^{-1}\| \cdot \|A\| = K(A)$$



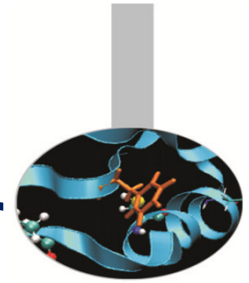
Condition number

$K(A)$ is the condition number with respect to the inversion of the matrix A or simply the *condition number* of A .

The value of the condition number depends on the used norm but, since matrix norms are equivalent, the order of magnitude of the condition numbers are similar.

The condition number gives a hint of how accurate is the solution of a linear system or how fast the solution varies as rhs values change.

E3 – exercise - Condition number



It can be demonstrated that $1 \leq K(A) \leq \infty$

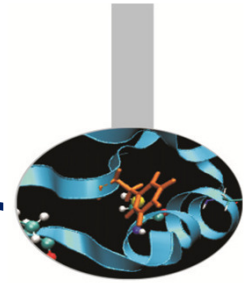
The nearest $K(A)$ is to 1 the more precise is the solution.

As an exercise a program could be written that calculates the condition number of a real $N \times N$ matrix A .

To do this the function DGECON of the LAPACK library can be used. Other useful functions are: DGETRF (LAPACK), DASUM (BLAS)

Source code: *CondNumber*

E3 – exercise - Condition number



```
FUNCTION DASUM( N, X, INCX ) RESULT(S)
```

```
! Sum of absolute values of a vector
```

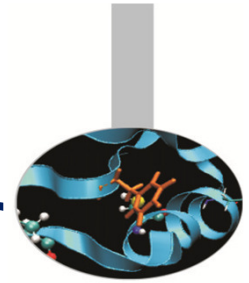
```
INTEGER, INTENT(IN) :: N, INCX
```

```
REAL(8), DIMENSION(N), INTENT(IN) :: X
```

```
REAL(8) :: S
```

```
END FUNCTION DASUM
```

E3 – exercise - Condition number



```
SUBROUTINE DGETRF ( M, N, A, LDA, IPIV, INFO)
```

```
!      Matrix factorization
```

```
INTEGER, INTENT(IN) :: M, N, LDA
```

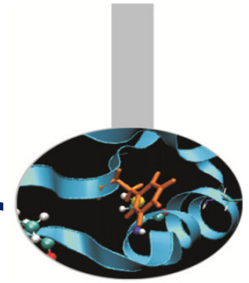
```
REAL(8), dimension( LDA, N ), INTENT(IN OUT) :: A
```

```
INTEGER, dimension( * ), INTENT(OUT) :: IPIV
```

```
INTEGER, INTENT(OUT) :: INFO
```

```
END SUBROUTINE DGETRF
```

E3 – exercise - Condition number



SUBROUTINE DGECON (NORM, N, A, LDA, ANORM, RCOND, WORK, &

& IWORK, INFO)

CHARACTER(1), INTENT(IN) :: NORM ! May be “1” or “I”

INTEGER, INTENT(IN) :: LDA, N

REAL(8), DIMENSION(LDA, N), INTENT(IN) :: A

REAL(8), INTENT(IN) :: ANORM

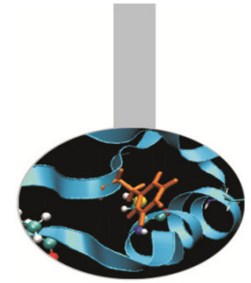
REAL(8), INTENT(OUT) :: RCOND

REAL(8), DIMENSION(4*N), INTENT(IN OUT) :: WORK

INTEGER, DIMENSION(N), INTENT(IN OUT) :: IWORK

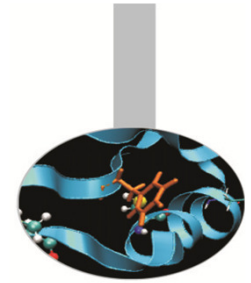
INTEGER, INTENT(OUT) :: INFO

END SUBROUTINE DGECON



E4 – example – Linear equation

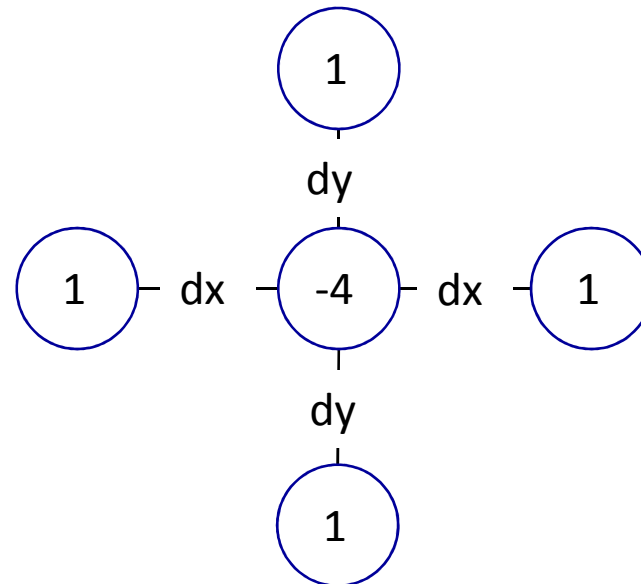
- Let us take into consideration the following equation in the $[(1,1), (1,2), (2,2), (2,1)]$ 2D-square:
 - $\partial^2 F(x,y) / \partial x^2 + \partial^2 F(x,y) / \partial y^2 = x + y$
- The exact solution is $F(x,y) = (x^3 + y^3) / 6$
- Let us pretend we do not know the exact solution but we know the values of the function on the perimeter of the square only and we are interested in computing the value of the function in a sufficient number of points within the square.
- A finite difference approach could be easily implemented.

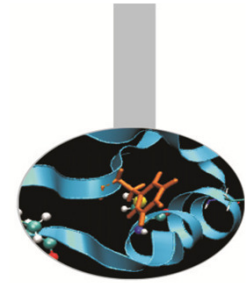


E4 – example – Linear equation

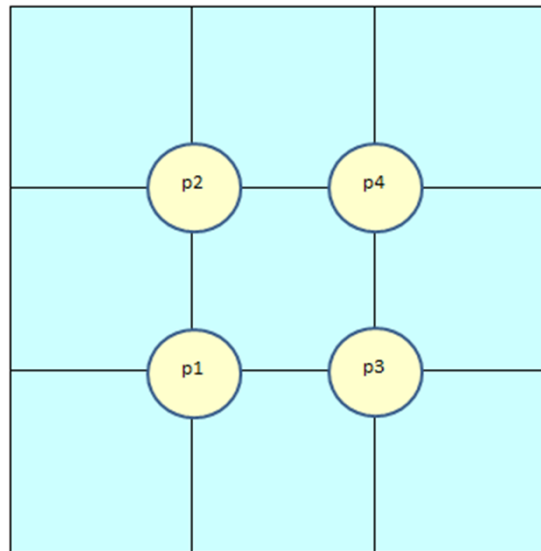
$\partial^2 F(x,y) / \partial x^2$ approximated by $[F(x+dx,y) + F(x-dx,y) - 2 * F(x,y)] / dx^2$

$\partial^2 F(x,y) / \partial x^2 + \partial^2 F(x,y) / \partial y^2 \Rightarrow [F(x+dx,y) + F(x-dx,y) + F(x,y+dy) + F(x,y-dy) - 4 * F(x,y)] / dx^2$

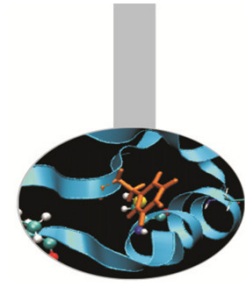




E4 – example – Linear equation



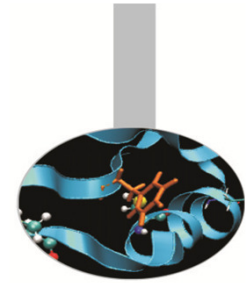
- Divide the square in NxN grid cells
- Inner points represent the unknown values. $LA = (N-1)*(N-1)$
- $A(LA,LA)$ matrix (finite difference coefficients) and $B(LA)$ RHS vector (rhs x+y value and peripheral values) can be generated.



E4 – example – Linear equation

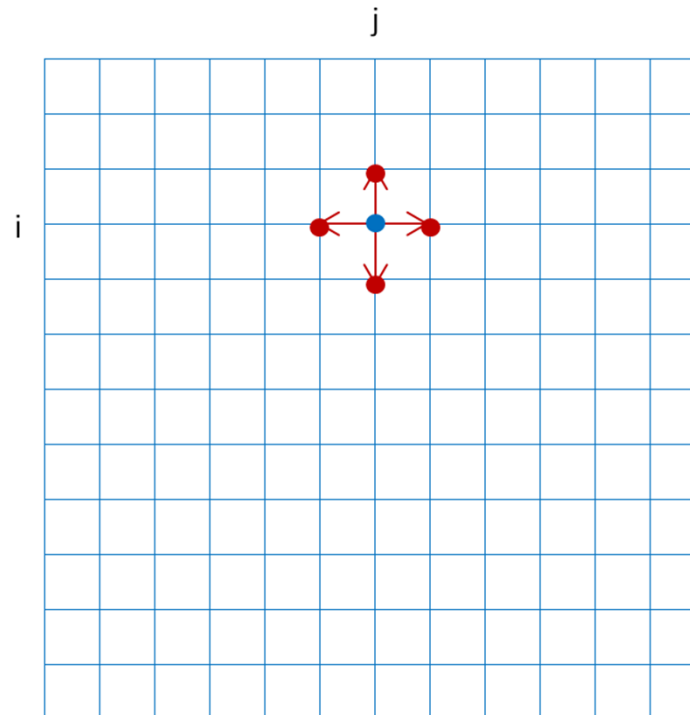
- The program could be designed as follow:
- Memorize inner point coords in Points(LA,2) vector
- Build A(LA,LA) matrix
- Build $B(LA) = (x+y)/(dx*dy)-p$; $p = \text{sum values peripheral points}$
- Factorize matrix: $A \Rightarrow L \cdot U$
- Solve the system
- Compare computed solution to exact known solution

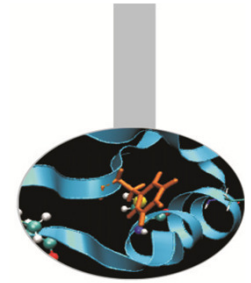
- Source code: LinearEquation



E5 – exercise – Band matrix

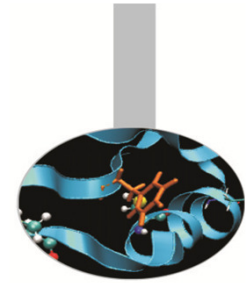
In the previous example a matrix has been generated to solve a PDE equation in a 2D-square. Grid points that are related can be far in the generated matrix \Rightarrow A is a band matrix





E5 – exercise – Band matrix

- Matrix $A(LA,LA)$, $LA=(N-1)*(N-1)$ of former example should have lower and upper bands with length N .
- Try modifying the source code in order to use Lapack routines `DGBTRF` and `DGBTRS` for band matrices.
- Source code: `BandMatrix`



E5 – exercise – Band matrix

```
SUBROUTINE DGBTRF( M, N, KL, KU, AB, LDAB, IPIV, INFO )
```

```
!      Band matrix factorization
```

```
INTEGER, INTENT(IN) :: M, N  ! Rows and columns
```

```
INTEGER, INTENT(IN) :: KL, KU  ! Number of lower and upper bands
```

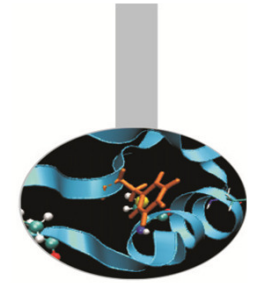
```
REAL(8), DIMENSION( LDAB, N ), INTENT(IN OUT) :: AB
```

```
INTEGER, INTENT(IN) :: LDAB    ! LDAB >= 2*KL+KU+1
```

```
INTEGER, DIMENSION( * ), INTENT(OUT) :: IPIV
```

```
INTEGER, INTENT(OUT) :: INFO
```

```
END SUBROUTINE DGBTRF
```



E5 – exercise – Band matrix

```
SUBROUTINE DGBTRS( TRANS, N, KL, KU, NRHS, AB, LDAB, IPIV, &  
& B, LDB, INFO )
```

```
! Solve a system of equation with band matrix
```

```
CHARACTER(1) :: TRANS ! "N" = no transpose
```

```
INTEGER, INTENT(IN) :: N, KL, KU ! Matrix order and bands
```

```
INTEGER, INTENT(IN) :: NRHS ! Righthand sides
```

```
REAL(8), DIMENSION( LDAB, N ), INTENT(IN) :: AB
```

```
INTEGER, INTENT(IN) :: LDAB ! LDAB >= 2*KL+KU+1
```

```
INTEGER, DIMENSION( * ), INTENT(IN) :: IPIV
```

```
REAL(8), DIMENSION(LDB,NRHS), INTENT(IN OUT) :: B
```

```
INTEGER, INTENT(IN) :: LDB
```

```
INTEGER, INTENT(OUT) :: INFO
```

```
END SUBROUTINE DGBTRS
```