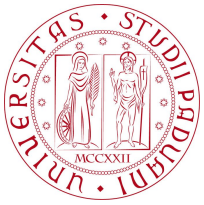
Librerie per problemi di algebra lineare

Università degli studi di Padova

18 Novembre 2013



Librerie di Algebra Lineare : BLAS

- BLAS : Basic Linear Algebra Subprograms

  http://www.netlib.org/blas/

- Insieme di procedure che forniscono funzioni di base per operazioni tra vettori e matrici.
  Costruite su tre livelli:
  - Lvl 1 : operazioni scalari-vettori e vettori-vettori;
  - Lvl 2 : operazioni vettori-matrici;
  - Lvl 3 : operazioni tra matrici;

- Sviluppate inizialmente in Fortran 77 e poi Fortran 90 Le librerie

  BLAS sono efficienti, portabili tra archietture diverse e comunemente utilizzate per sviluppare programmi di algebra lineare di alta qualità (le librerie LAPACK usano le BLAS)

Librerie di Algebra Lineare : LAPACK

- LAPACK : Linear Algebra PACKage

  http://www.netlib.org/lapack/

- routines scritte in Fortran 90 che permttono di risolvere sistemi di equazioni lineari, calcolo di autovalori ed autovettori, fattorizzazioni di matrici, etc. Le librerie trattano matrici dense e a banda, ma non matrici sparse in generale. Le librerie sfruttano pesantemente le BLAS delle quali espandono il Livello 3. Esistono delle API in C che permettono di utilizzare le LAPACK, sviluppate in collaborazione con INTEL

  http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation/

Librerie di Algebra Lineare : BOOST

- BOOST : Free Peer-reviewed portable C++ source libraries

  http://www.boost.org

- insieme di librerie che integrano molto bene la Standard Library del C++.
- recentemente dieci librerie sono state incluse nello standard del linguaggio (C++ 11)
- sono divise in sotto librerie che coprono argomenti i più disparati (algoritmi, array, operazioni atomiche, contenitori, date-time, grafici, logging, lambda, network, funzioni matematiche, etc).

Librerie di Algebra Lineare : uBLAS

- uBLAS : una template class library che incorpora le funzionalità BLAS per matrici dense, packed e sparse.

  http://www.boost.org/doc/libs/1_54_0/libs/numeric/ublas/doc/index.htm

- it provides provides templated C++ classes for dense, unit and sparse vectors, dense, identity, triangular, banded, symmetric, hermitian and sparse matrices. Views into vectors and matrices can be constructed via ranges, slices, adaptor classes and indirect arrays. The library covers the usual basic linear algebra operations on vectors and matrices: reductions like different norms, addition and subtraction of vectors and matrices and multiplication with a scalar, inner and outer products of vectors, matrix vector and matrix matrix products and triangular solver. The glue between containers, views and expression templated operations is a mostly STL conforming iterator interface.

I Vettori in uBLAS

- L'utilizzo di vettori e matrici è intuitivo e semplice.
- Un vettore di `double` viene creato e inizializzato da

  ```
  vector<double> v(unsigned int n, double x)
  ```

  con `n` numero di elementi del vettore e `x` valori di default.
- si possono creare vettori speciali

  ```
  unit_vector<double> v(unsigned int n)
  zero_vector<double> v(unsigned int n)
  ```

- si possono eseguire operazioni tra vettori e tra vettori e scalari.

```
#include <boost/numeric/ublas/vector.hpp>
using namespace boost::numeric::ublas;
```

uBLAS: vector class

```
[3](2,-5,2)
-1
9
5.74456
5
1
```

```
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/io.hpp>
using namespace boost::numeric::ublas;

int main( )
{
 vector <double> v(3, 2.0);
    v[1] = -5.0

    std::cout << v << std::endl;

    std::cout << sum(v) << std::endl;
    std::cout << norm_1(v) << std::endl;
    std::cout << norm_2(v) << std::endl;
    std::cout << norm_inf(v) << std::endl;
    std::cout << index_norm_inf(v) << std::endl;

}
```

$\text{norm\_1} = \sum_i |v_i|$

$\text{norm\_2} = \sqrt{\sum_i v_i^2}$

$\text{norm\_inf} = \text{MAX}(|v_i|)$

uBLAS: vector operations

```
v1: [3](2,-2,2) | v2: [3](1,-1.5,1)
v1 + v2 : [3](3,-3.5,3) | v1 - v2 :
[3](1,-0.5,1)
inner_prod(v1,v2) : 7
outer_prod(v1,v2) :
[3,3]((2,-3,2),(-2,3,-2),(2,-3,2))
v1 * k : [3](4,-4,4) | v1 / k : [3](4,-4,4)
```

```
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/io.hpp>
using namespace boost::numeric::ublas;
int main( )
{
 vector <double> v1(3, 2.0);
    v1[1] = - v1[1];
    vector <double> v2(3, 1.0);
    v2[1] = -1.5;
    std::cout << "v1: " << v1;
              << "v2: " << v2 << std::endl;

    std::cout << " v1 + v2 : " << v1 + v2 << " | ";
    std::cout << " v1 - v2 : " << v1 - v2 << std::endl;
    std::cout << inner_prod(v1,v2) << std::endl;
    std::cout << outer_prod(v1,v2) << std::endl;
    double k = 2.0;
    std::cout << " v1 * k : " << v1 * k << " | ";
    std::cout << " v1 / k : " << v1 * k << std::endl;
}
```

Le Matrici in uBLAS

- Un matrice è creata tramite

  `matrix<T> A(unsigned int righe, unsigned int col)`

  con `T` classe templata
- ci sono due funzioni che ritornano il numero di righe `size1()` e di colonne `size2()` della matrice.
- È possibile creare matrici speciali, quella identità e la matrice nulla:

```
identity_matrix<T> I(unsigned int nrighe, unsigned
                     int ncol)
zero_matrix<T> O(unsigned int nrighe, unsigned int
                 ncol)
```

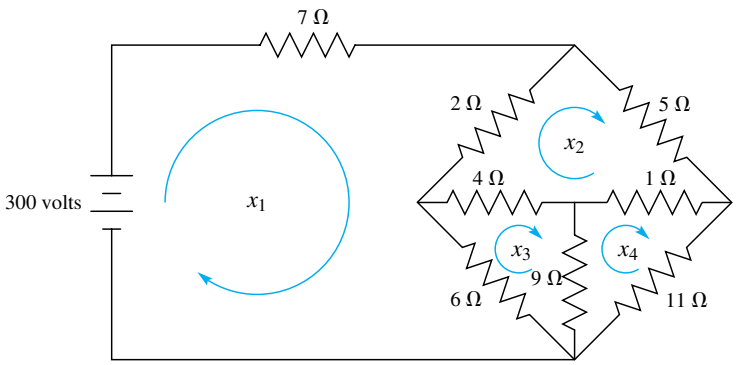
```
#include <boost/numeric/ublas/matrix.hpp>
using namespace boost::numeric::ublas;
```

Esempio: soluzione di un sistema lineare

A simple electrical network contains a number of resistances and a single source of electromotive force (a battery) as shown in Figure 7.1. Using Kirchhoff's laws and Ohm's law, we can write a system of linear equations that govern this circuit. If x_1, x_2, x_3, and x_4 are the loop currents as shown, then the equations are

$$\begin{cases} 15x_1 - 2x_2 - 6x_3 = 300 \\ -2x_1 + 12x_2 - 4x_3 - x_4 = 0 \\ -6x_1 - 4x_2 + 19x_3 - 9x_4 = 0 \\ - x_2 - 9x_3 + 21x_4 = 0 \end{cases}$$

$x_1 = 26.5 \quad x_2 = 9.35 \quad x_3 = 13.3 \quad x_4 = 6.13$



```
x = [4](26.5492,9.3537,13.255,6.12613)
A * x = [4](300,1.42e-14,-1.42e-14,-2.84e-14)
b = [4](300,0,0,0)
```

```cpp
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/lu.hpp>
#include <boost/numeric/ublas/io.hpp>
using namespace boost::numeric::ublas;

int main()
{
    matrix<double> A(4, 4, 0.0);
    A(0,0) = 15;  A(0,1) = -2;  A(0,2) = -6;  A(0,3) = 0;
    A(1,0) = -2;  A(1,1) = 12;  A(1,2) = -4;  A(1,3) = -1;
    A(2,0) = -6;  A(2,1) = -4;  A(2,2) = 19;  A(2,3) = -9;
    A(3,0) = 0;   A(3,1) = -1;  A(3,2) = -9;  A(3,3) = 21;
    matrix<double> Acopy(A);
    permutation_matrix<double> P(4);

    vector<double> b(4, 0.0);
    b[0] = 300.0;
    vector<double> x(4, 0);

    lu_factorize(Acopy, P);  // A e P contengono la fattorizzazione LU di A
    x = b;
    lu_substitute(Acopy, P, x);

    std::cout << " x = " << x << std::endl;
    std::cout << " A * x =  " << prod(A, x) << std::endl;
    std::cout << " b = " << b << std::endl;
    return 0;
}
```