Metodi Computazionali della Fisica
Secondo Modulo: C++

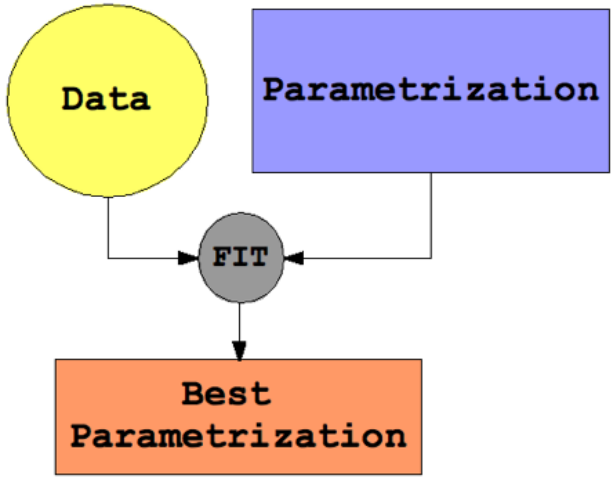
Seconda Lezione

Introduzione ai fit di dati

Contenitori: `vector` e `valarray`

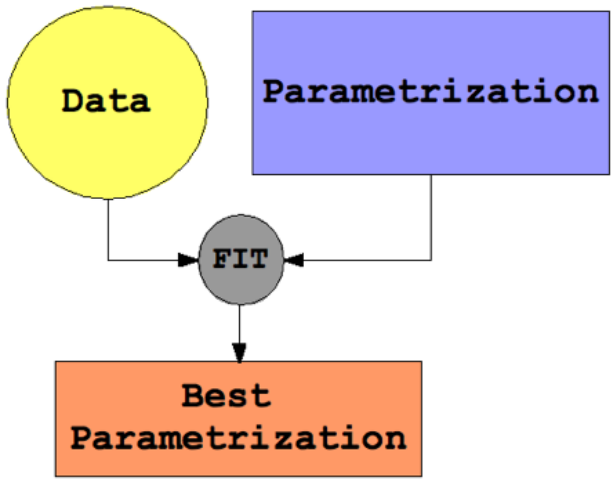
Definizione della classe Data

In breve



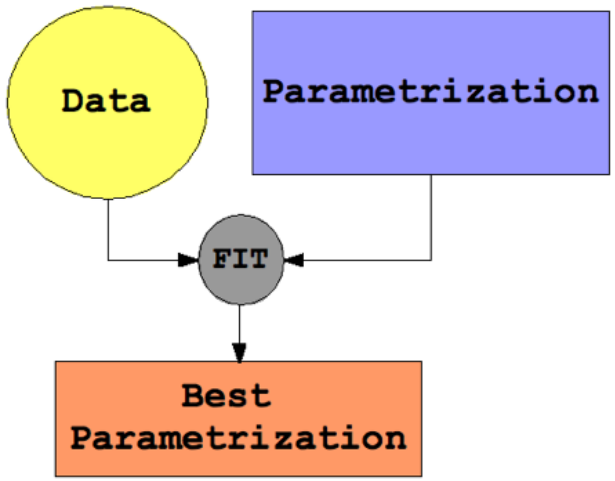
- Lo spazio delle possibili misure (Data) è infinito.
- Lo spazio delle possibili funzioni (Parametrization) è infinito.
- In pratica avremo un numero finito N di dati e un numero finito M di parametri.

In breve



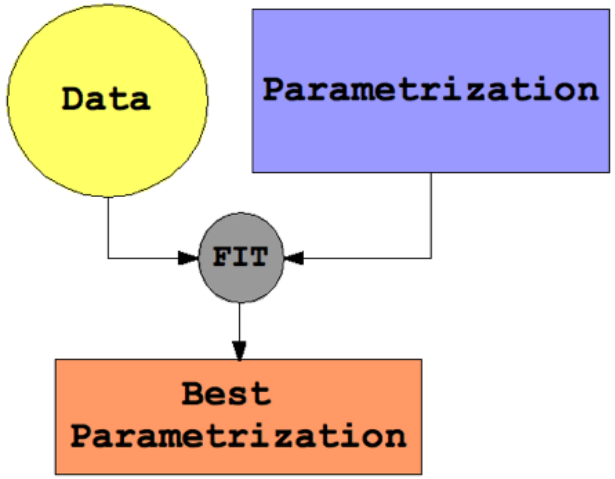
- Lo spazio delle possibili misure (Data) è infinito.
- Lo spazio delle possibili funzioni (Parametrization) è infinito.
- In pratica avremo un numero finito N di dati e un numero finito M di parametri.

In breve



- Lo spazio delle possibili misure (Data) è infinito.
- Lo spazio delle possibili funzioni (Parametrization) è infinito.
- In pratica avremo un numero finito N di dati e un numero finito M di parametri.

In breve



- Lo spazio delle possibili misure (Data) è infinito.
- Lo spazio delle possibili funzioni (Parametrization) è infinito.
- In pratica avremo un numero finito N di dati e un numero finito M di parametri.

Rappresentare e generalizzare.

Un fit (l'*adattamento* di una funzione ai dati attraverso l'*aggiustamento* dei parametri liberi della funzione) deve rappresentare e generalizzare:

- rappresentare: riprodurre i dati entro un certo limite (ad esempio, l'errore sperimentale o l'accuratezza numerica) secondo quanto richiesto dalla figura di merito utilizzata (si veda la lezione sulla minimizzazione);
- generalizzare:interpolare ed estrapolare (vogliamo qualcosa che ci permetta di fare predizioni quali, ad esempio, quanta energia elettrica servirà domani al Comune di Milano, quale massa potrebbe avere il bosone di Higgs).

Rappresentare e generalizzare.

Un fit (l'*adattamento* di una funzione ai dati attraverso l'*aggiustamento* dei parametri liberi della funzione) deve rappresentare e generalizzare:

- rappresentare: riprodurre i dati entro un certo limite (ad esempio, l'errore sperimentale o l'accuratezza numerica) secondo quanto richiesto dalla figura di merito utilizzata (si veda la lezione sulla minimizzazione);
- generalizzare:interpolare ed estrapolare (vogliamo qualcosa che ci permetta di fare predizioni quali, ad esempio, quanta energia elettrica servirà domani al Comune di Milano, quale massa potrebbe avere il bosone di Higgs).

Rappresentare e generalizzare.

Un fit (l'*adattamento* di una funzione ai dati attraverso l'*aggiustamento* dei parametri liberi della funzione) deve rappresentare e generalizzare:

- rappresentare: riprodurre i dati entro un certo limite (ad esempio, l'errore sperimentale o l'accuratezza numerica) secondo quanto richiesto dalla figura di merito utilizzata (si veda la lezione sulla minimizzazione);
- generalizzare:interpolare ed estrapolare (vogliamo qualcosa che ci permetta di fare predizioni quali, ad esempio, quanta energia elettrica servirà domani al Comune di Milano, quale massa potrebbe avere il bosone di Higgs).

STL - vector

```
#include <vector>
vector();
vector( const vector& c );
vector( size_type num, const TYPE& val = TYPE() );
vector( input_iterator start, input_iterator end );
```

Per tutti i dettagli:

- http://www.sgi.com/tech/stl/index.html
- http://www.cppreference.com/cppvector/index.html

Esempio: [# 1]

```
#include <iostream>
#include <vector>
using namespace std;

int main(void){

  int nmax;
  cout<<"Insert vector size:"<<endl;
  cin>>nmax;

  vector<int> index;
  vector<int> vec;

  for(int i=0;i<nmax;i++){
    int tmp=i+2;
    index.push_back(i);
    vec.push_back(tmp);
    //cout<<i<<" "<<vec[i]<<endl;
  };
```

Esempio: [# 2]

```
  int ncols=2;

  vector<int>* both;
  both = new vector<int> [ncols];

  both[0]=index;
  both[1]=vec;

  //cout<<" "<<both[0][nmax-1]<<" "<<both[1][nmax-1]<<endl;

};
```

STL (ext) - valarray

Il contenitore `valarray` permette di rappresentare e manipolare vettori unidimensionali.
A differenza di altri contenitori della STL può essere usato con un numero ristretto di tipi, ma questa restrizione assicura efficienza nelle operazioni numeriche dal momento che evita ambiguità.
Noi useremo questo contenitore per generare maschere di verità. Per tutti i dettagli:

- `http://incubator.apache.org/stdcxx/doc/stdlibref/valarray.html`
- `http://incubator.apache.org/stdcxx/doc/stdlibref/mask-array.html`

Esempio: [# 1]

```
#include <iostream>
#include <valarray>

using namespace std;

int main(void){

  float af1 [] = {8, 5, 2, 3, 9, 1};
  valarray <float> vf1 (af1, 6);

  bool b [] = {true, true, false, true, false, true};
  valarray <bool> vb (b, 6);

  cout<<"Original vector and bool flags:"<<endl;
  for(int i=0;i<vf1.size();i++){
    cout<<i<<" "<<vf1[i]<<" "<<b[i]<<endl;
  };
```

Esempio: [# 2]

```
  valarray<float> vf2 = vf1[vb];

  cout<<"True subvector:"<<endl;
  for(int i=0;i<vf2.size();i++){
    cout<<i<<" "<<vf2[i]<<endl;
  };

  valarray<float> vf3 = vf1[!vb];

  cout<<"False subvector:"<<endl;
  for(int i=0;i<vf3.size();i++){
    cout<<i<<" "<<vf3[i]<<endl;
  };

  return 0;
};
```

Caratteristiche dei dati:

1. numero di colonne del file (numero di input, output, errori);
2. numero di dati;
3. tipo (float, int, bool);
4. valori massimi e valori minimi;

Nella definizione della classe adotteremo le seguenti ipotesi di lavoro:

- numero di dati è il numero di righe che legge il programma;
- la prima colonna rappresenta interi (numero sequenziale), tutte la altre saranno colonne di float/double.

Caratteristiche dei dati:

1. numero di colonne del file (numero di input, output, errori);
2. numero di dati;
3. tipo (float, int, bool);
4. valori massimi e valori minimi;

Nella definizione della classe adotteremo le seguenti ipotesi di lavoro:

- numero di dati è il numero di righe che legge il programma;
- la prima colonna rappresenta interi (numero sequenziale), tutte la altre saranno colonne di float/double.

Data.hpp: [# 1]

```
#ifndef _DATA_HPP
#define _DATA_HPP

#include <iostream>
#include <vector>
#include <fstream>
#include <valarray>

using namespace std;
```

Data.hpp: [# 2]

```
class Data {
public:

  Data();
  Data(const size_t&,const size_t&,const size_t&);
  Data(const Data&);
  Data(const Data&, const valarray<bool>&);
  ~Data();

  size_t read_data(string,size_t,size_t,size_t,size_t);
  void find_range();
```

Data.hpp: [# 3]

```
double get_data(size_t i,size_t np){return data[i][np];};
double get_min(size_t i){return datamin[i];};
double get_max(size_t i){return datamax[i];};

size_t get_npoints(){return npoints;};
size_t get_nin(){return nin;};
size_t get_nout(){return nout;};
size_t get_nerr(){return nerr;};
```

Data.hpp: [# 4]

```
private:

  vector<double> *data;

  vector<double> datamin;
  vector<double> datamax;

  size_t npoints,nin,nout,nerr;

};

#endif
```

Esercizio

1. Implementate i costrutori della classe.
2. Implementate i metodi della classe.
3. Compilate senza generare l'eseguibile per controllare la sintassi (`g++ -c nomefile.cpp -o tmp.o`).
4. Scrivete un programma di test tipo quello mostrato nell'esempio che segue.
5. Utilizzate i dati generati con il codice della scorsa lezione per controllare che tutto funzioni.
6. Fate una sottomissione del codice nella repository.

Data.cpp:

```
Data::Data(){

  cout<<"Inside Data empty constructor"<<endl;

}

Data::Data(const size_t& ni,const size_t& no,const size_t& ne){

  cout<<"Inside Data constructor"<<endl;

[...]

}
```

test_data.cpp: [# 1]

```
int main(void) {

  Data MyData(1,1,1);

  size_t totdata=MyData.read_data("prova",0,3,1,3);

  MyData.find_range();

  cout<<"Check copy constructor"<<endl;

  Data NewData(MyData);

  cout<<MyData.get_data(1,5)<<" "<<NewData.get_data(1,5)<<endl;
  cout<<MyData.get_data(2,5)<<" "<<NewData.get_data(2,5)<<endl;
```

test_data.cpp: [# 2]

```
  cout<<"Check bool copy constructor"<<endl;

  double t_frac=0.2;
  bool b[totdata];
  size_t k=0;

  for(size_t i=0; i<totdata; i++) {
    b[k]=true;
    if(RandomGenerator::RandomUniform()<t_frac) b[k]=false;
    k++;
  };

  valarray<bool> mask(b,totdata);

  Data Train(MyData,mask);
  Data Validation(MyData,!mask);

  return 0;
}
```