Subject: Re: Added Eigen
Posted by Sender Ghost on Fri, 10 Aug 2012 10:45:43 GMT
View Forum Message <> Reply to Message

Hello, Luigi.
forlano wrote on Fri, 10 August 2012 08:42Now, and here comes the problems, I would like to use the same curve, with the same number of parameters, but with a NEW dataset [X,Y] of different size.
Following is a possible implementation:
Toggle Spoiler

```
#include <Core/Core.h>

using namespace Upp;

#include <plugin/Eigen/Eigen.h>
#include <plugin/Eigen/unsupported/Eigen/NonLinearOptimization>

using namespace Eigen;

// Generic functor
template<typename _Scalar, int nx = Dynamic, int ny = Dynamic>
struct Functor {
 typedef _Scalar Scalar;
 enum {
  InputsAtCompileTime = nx,
  ValuesAtCompileTime = ny
 };
 typedef Matrix<Scalar,InputsAtCompileTime,1> InputType;
 typedef Matrix<Scalar,ValuesAtCompileTime,1> ValueType;
 typedef Matrix<Scalar,ValuesAtCompileTime,InputsAtCompileTime> JacobianType;

 int m_inputs, m_values;
 void SetInputsCount(int count) { m_inputs = count; }
 void SetValuesCount(int count) { m_values = count; }

 Functor() : m_inputs(InputsAtCompileTime), m_values(ValuesAtCompileTime) {}
 Functor(int inputs, int values) : m_inputs(inputs), m_values(values) {}

 int inputs() const {return m_inputs;}
 int values() const {return m_values;}

 // you should define that in the subclass :
 virtual  void operator() (const InputType& x, ValueType* v, JacobianType* _j=0) const {};
};

class LogisticA_functor : public Functor<double> {
protected:
```

```
 double *vx;
 double *vy;
public:
 void Set(double *x, double *y) { vx = x; vy = y; }

 int operator()(const VectorXd &b, VectorXd &fvec) const {
  ASSERT(b.size()==m_inputs);
  ASSERT(fvec.size()==m_values);
  for(int i=0; i<m_values; i++)
   fvec[i] = b[0] / (1.0 + b[1]*exp(-1.0 * b[2] * vx[i])) - vy[i];
  return 0;
 }
};

void DoLevenbergMarquardt(LogisticA_functor& functor, VectorXd& x, double *j, double *k, int
inputs, int values)
{
 functor.Set(j, k);
 functor.SetInputsCount(inputs);
 functor.SetValuesCount(values);
 NumericalDiff<LogisticA_functor> numDiff(functor);
 LevenbergMarquardt<NumericalDiff<LogisticA_functor> > lm(numDiff);

 int ret = lm.minimize(x);
 if (ret == LevenbergMarquardtSpace::ImproperInputParameters ||
  ret == LevenbergMarquardtSpace::TooManyFunctionEvaluation)
  Cout() << "\nNo convergence!: " << ret << '\n';
 else
  for (int i = 0; i < inputs; ++i)
   Cout() << "Parameter: " << i << " = " << x[i] << '\n';
}

void NonLinearOptimization() {
 const int inputs = 3;
 VectorXd x(inputs);
 x << 5., 5., 0.01; // Initial values

 LogisticA_functor functor;
 Cout() << "First run\n";
 double jx[13] = {-280.0, -240.0, -200.0, -160.0, -120.0, -80.0, -40.0, 0.0, 40.0, 80.0, 120.0, 160.0,
200.0},
  jy[13] = {0.061276, 0.071429, 0.091574, 0.112821, 0.132959, 0.131597, 0.167887, 0.198380,
0.221380, 0.292292, 0.351831, 0.445803, 0.497754};
 VectorXd j = x;
 DoLevenbergMarquardt(functor, j, jx, jy, inputs, 13);

 Cout() << "Second run with new data of different length and same curve to fit\n";
 double kx[15] = {-280.0, -240.0, -200.0, -160.0, -120.0, -80.0, -40.0, 0.0, 40.0, 80.0, 120.0, 160.0,
```

```
200.0, 240.0, 280.0},
  ky[15] = {0.061276, 0.071429, 0.091574, 0.112821, 0.132959, 0.131597, 0.167887, 0.198380,
0.221380, 0.292292, 0.351831, 0.445803, 0.497754 ,  0.609337,0.632353};
 VectorXd k = x;
 DoLevenbergMarquardt(functor, k, kx, ky, inputs, 15);
}

CONSOLE_APP_MAIN
{
 NonLinearOptimization();
}
```