# Nonlinear optimization in Eigen

Thomas Capricelli

InModelia http://www.inmodelia.com

Eigen Meeting
February 20th, 2010

It all started from InModelia needs with respect to numerical processing. Our software is written in C++ and is cross-platform.

## Linear Algebra

*After a careful internal study, Eigen stepped as a clear winner on top of other solutions (yeah !).*

## Non Linear optimization (Levenberg Marquardt)

*The same kind of exhaustive search on the net brings minpack as the best freely available LM implementation. The original code is Fortran but there exist "automatic translation" in C, using f2c, tested and validated.*

Challenge : how to integrate minpack in our code ?

## The Levenberg–Marquardt algorithm

General algorithm for minimizing a (non linear) function over a space of parameters of the function.

Most important use : least squares curve fitting problem, this is what InModelia needs :

$$S(\beta) = \sum_{i=1}^{m} [y_i - f(x_i, \beta)]^2$$

Can be compared to Gauss-Newton, gradient (or "steepest") descent, quasi newton, BFGS, ...

## The Levenberg–Marquardt algorithm

Notations :

- $J$ is the jacobian matrix,
- $\lambda$ is the *dumping parameter*.

$$(J^T J + \lambda \, diag(J^T J))\boldsymbol{\delta} = J^T [y - f(\boldsymbol{\beta})]$$

Notes :

- can be done in very few lines in Eigen,
- a very tricky part is how to compute the *dumping parameter*,
- lot of difficulty in numerical details, too.

## What does minpack provide ?

- hybrj : find a zero of a system of nonlinear function, using the analytical jacobian.
- hybrd : the same using an approximation for the jacobian.
- lmder : Levenberg-Marquardt using the analytical jacobian.
- lmdif : the same using an approximation for the jacobian.
- lmstr : the same, but the whole jacobian is not stored ("storage efficient").
- 'simplified' variants with default parameters.

All of this in two variations : float/double.

Minpack, in fortran, several publications.

```
subroutine lmpar(n,r,ldr,ipvt,diag,qtb,delta,par,x,
    sdiag,wa1,wa2)
integer n,ldr
integer ipvt(n)
double precision delta,par
double precision r(ldr,n),diag(n),qtb(n),x(n),
    sdiag(n),wa1(n),wa2(n)
```

Someone used f2c to create some cminpack, which is kinda *raw*

```
int lmpar_(integer *n, doublereal *r__, integer *ldr,
    integer *ipvt, doublereal *diag, doublereal *qtb,
    doublereal *delta, doublereal *par, doublereal *x,
    doublereal *sdiag, doublereal *wa1, doublereal *wa2)
```

The code was later somehow cleaned, but only so that the API looks less uglier, the code is still ugly, weird, fortran-translated stuff.

```
void lmpar(int n, double *r__, int ldr, const int *ipvt,
const double *diag, const double *qtb, double delta,
double *par, double *x, double *sdiag, double *wa1,
double *wa2)
```

It handles in/out, constness, and most importantly include a port of fortrant examples. It's still called cminpack.

Starting from this latest cminpack :

1. transform all examples into tests. I test result value, number of function evaluations and number of jacobian evaluation,
2. add tests from NIST,
3. do all adaptations, being very careful not to break those tests.

The modification I've done :

- Remove working arrays, gotos.
- Porting to C++ : classes, templates, references, functor.
- Use eigen vectors/matrices objects and API (stable norms, triangular solvers).
- Further porting to Eigen : ei_*, traits, epsilon.
- Provide a generic way of computing numerical differentiation and merge variants.
- Use QR from Eigen and remove the minpack implementation.

Before

```
i__1 = n;
for (j = 1; j <= i__1; ++j) {
wa3[j] = 0.;
l = ipvt[j];
temp = wa1[l];
i__2 = j;
for (i__ = 1; i__ <= i__2; ++i__)
    wa3[i__] += fjac[i__ + j * fjac_dim1] * temp;
}
temp1 = enorm(n, &wa3[1]) / fnorm;
temp2 = sqrt(par) * pnorm / fnorm;
```

After :

```
wa3 = fjac.template triangularView<Upper>()*
    (colsPermutation * wa1);
temp1 = ei_abs2(wa3.stableNorm() / fnorm);
temp2 = ei_abs2(ei_sqrt(par) * pnorm / fnorm);
```

The good news is that it works and I'm very confident that is provides algorithms as close as possible to the original ones, with the following *added* benefits :

- the code is a lot simpler and easier to read,
- it works with any kind of Scalar,
- it works on a broad range of platforms/compilers,
- eigen optimization (cache friendliness, vectorization),
- it uses correct machine precision (cminpack has it hardcoded).

(meanwhile two bugs were found, fixed, reported to, and acknowledged by cminpack maintainer).

Some remains to be done, but the most important steps are :

- Switch to eigen's QR Givens decompozition for lmstr.
- Improve API (sparse, fixed-size, ReturnByValue,...).