Sometimes, the objective function's symbolic gradient, or some approximation of it, is known. In this case, black box optimization methods such as CMA-ES may prove costly compared to others.

However, in these cases, libcmaes allows to use the gradient as extra information to speed-up convergence.

Below is a basic example that can be found in examples/sample-code-gradient.cc :

```
#include "cmaes.h"
#include <iostream>

using namespace libcmaes;

FitFunc fsphere = [](const double *x, const int N)
{
  double val = 0.0;
  for (int i=0;i<N;i++)
    val += x[i]*x[i];
  return val;
};

GradFunc grad_fsphere = [](const double *x, const int N)
{
  dVec grad(N);
  for (int i=0;i<N;i++)
    grad(i) = 2.0*x[i];
  return grad;
};

int main(int argc, char *argv[])
{
  int dim = 10; // problem dimensions.
  std::vector<double> x0(dim,10.0);
  double sigma = 0.1;
  //int lambda = 100; // offsprings at each generation.
  CMAParameters<> cmaparams(dim,&x0.front(),sigma);
  cmaparams._algo = aCMAES;
  CMASolutions cmasols = cmaes<>(fsphere,cmaparams,
                                 CMAStrategy<CovarianceUpdate>::_defaultPFunc,  // use default progress fu
                                 grad_fsphere);
 std::cout << "best solution: " << cmasols << std::endl;
  std::cout << "optimization took " << cmasols._elapsed_time / 1000.0 << " seconds\n";
  return cmasols._run_status;
}
```

In summary, injection of the gradient is as easy as the definition of the GradFunc function above, and its passing as parameter to the cmaes<> call.