libcmaes supports a range of transforms from the original parameter space, referred to as 'phenotype' space, to an internal parameter space, referred to as 'genotype' space.

These transforms have a variety of usages, that can be combined:

1- apply bound control to the original parameter space, see Bounds on Parameters for the steps for applying bounds only without diving into the more complex transforms;

2- apply a linear or non-linear transformation to the phenotype space in order to achieve similar sensitivity across all parameters (see Nikolaus Hansen's page for more hints about this);

3- apply a custom transform, as needed by external applications.

These usages can all be achieved through the generic genotype / phenotype transformation scheme built in libcmaes.

Achieving similar parameter sensitivity through linear scaling

The library has a built-in linear scaling scheme that can in many applications improve results significantly.

There are three cases to consider for using this scaling:

- when the parameter space is bounded with known bounds, the scaling can then be automatically determined by the library:

```
int dim = 10; // problem dimensions.
std::vector<double> x0(dim,1.0);
double sigma = 0.1;
double lbounds[dim],ubounds[dim]; // arrays for lower and upper parameter bounds, respectively
for (int i=0;i<dim;i++)
  {
    lbounds[i] = -2.0;
    ubounds[i] = 2.0;
  }
GenoPheno<pwqBoundStrategy,linScalingStrategy> gp(lbounds,ubounds,dim);
CMAParameters<GenoPheno<pwqBoundStrategy,linScalingStrategy>> cmaparams(dim,&x0.front(),sigma,-1,0,gp);
CMASolutions cmasols = cmaes<GenoPheno<pwqBoundStrategy,linScalingStrategy>>(fsphere,cmaparams);
```

- when the parameter space (or most of it) is unbounded, but that a scaling vector is known. This typically happens when the initial estimated error sigmai of each parameter i is known, and therefore the scaling vector is given by:

```
dVec scaling(dim);
for (int i=0;i<dim;i++)
  scaling[i]=1.0/sigmai;
```

In this case, the GenoPheno object can be built in this way:

```
dVec shift = dVec::Zero(dim);
GenoPheno<NoBoundStrategy,linScalingStrategy> gp(scaling,shift);
```

- when the parameter space is bounded and the scaling vector can be determined in advance:

```
dVec shift = dVec::Zero(dim);
GenoPheno<NoBoundStrategy,linScalingStrategy> gp(scaling,shift,lbounds,ubounds);
```

where lbounds and ubounds are arrays of double that contain the lower and upper bounds respectively.

Applying custom transforms

It is rather straightforward to define custom transforms. The following code is available in examples/sample-code-genopheno.cc and demonstrates how to use a simple bijective transform:

```
#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;
};

// dummy genotype / phenotype transform functions.
TransFunc genof = [](const double *ext, double *in, const int &dim)
{
  for (int i=0;i<dim;i++)
    in[i] = 2.0*ext[i];
};

TransFunc phenof = [](const double *in, double *ext, const int &dim)
{
  for (int i=0;i<dim;i++)
    ext[i] = 0.5*in[i];
};

int main(int argc, char *argv[])
{
  int dim = 10; // problem dimensions.
  std::vector<double> x0(dim,1.0);
  double sigma = 0.1;
  GenoPheno<> gp(genof,phenof);
  CMAParameters<> cmaparams(dim,&x0.front(),sigma,-1,0,gp); // -1 for automatically decided lambda.
  CMASolutions cmasols = cmaes<>(fsphere,cmaparams);
  std::cout << "best solution: " << cmasols << std::endl;
  std::cout << "optimization took " << cmasols._elapsed_time / 1000.0 << " seconds\n";
  return cmasols._run_status;
}
```

The only requirement in the code above are the two functions genof and phenof of type TransFunc.