To find the minimum of an objective function is a four steps process:

- define an objective function,
- define a CMAParameters object that contains input details to the algorithm,
- call on the optimizer,
- finally get a CMASolutions object that contains the optimization results.

These three steps are reviewed below.

Objective function definition

An objective function is defined as a of type FitFunc:

```
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;
};
```

where x is the vector of parameters to the objective function, and N its dimension. Therefore N is the dimensionality of the problem (see [[Practical-hints]] on how to deal with large parameter spaces).

Optimization parameters

Next comes the instanciation of the CMAParameters object:

```
int dim = 10; // problem dimensions.
double sigma = 0.1; // initial step-size, i.e. estimated initial parameter error.
std::vector<double> x0(dim,10.0); // initialize x0 as 10.0 in all 10 dimensions
CMAParameters<> cmaparams(dim,&x0.front(),sigma); // &x0.front() allows to turn the vector into a double*
cmaparams._algo = aCMAES;  // select active CMA-ES as algorithm (default is CMA-ES).
```

The full API of the CMAParameters object has more abilities such as control of tolerance, function evaluation budget, number of offsprings per generation etc... For instance:

```
cmaparams.set_max_fevals(10000); // limits the number of function evaluations to 10000
cmaparams.set_max_iter(100000); // limits the number of iterations of the algorithms to 1000000
cmaparams.set_ftarget(1e-8); // stops the optimization whenever the objective function values gets below 1e-8
```

See CMAParameters API for more details.

Running the optimizer

Next step is to run the optimizer:

```
CMASolutions cmasols = cmaes<>(fsphere,cmaparams);
```

This yields a CMASolutions object that allows to check on the optimization final status, some statistics, and to study and capture the final and best solution.

```
  std::cout << "best solution: " << cmasols << std::endl;
  std::cout << "optimization took " << cmasols._elapsed_time / 1000.0 << " seconds\n";
  std::cout << cmasols._run_status; // the optimization status, failed if < 0
}
```

One of the main element to first check is the final status of the optimization process.

Error codes

The following error codes are available, as defined in CMAStopCriteria:

```
CONT = 0,
AUTOMAXITER = 7,
TOLHISTFUN = 1, // convergence
EQUALFUNVALS = 5, // partial success, user error
TOLX = 2, // partial success
TOLUPSIGMA = -13,
STAGNATION = 6, // partial success
CONDITIONCOV = -15, // error, user action needed
NOEFFECTAXIS = 3, // partial success
NOEFFECTCOOR = 4, // partial success
MAXFEVALS = 8,
MAXITER = 9,
FTARGET = 10 // success
```

Solution, Error covariance matrix and Expected Distance to the Minimum (EDM)

The best solution is obtained as a Candidate object:

```
Candidate bcand = cmasols.best_candidate();
double fmin = bcand.get_fvalue(); // min objective function value the optimizer converged to
const double* x = bcand.get_x(); // vector of objective function parameters at minimum.
double edm = cmasols.edm(); // expected distance to the minimum.
```

A copy of the error covariance matrix can be obtained from the CMASolutions object:

```
dMat err_cov = cmasols.cov();
```

where dMat is an Eigen3 matrix. It can easily be converted to a array of double:

```
double* derr_cov = err_cov.data();
```