OptimC

Optimization / Unconstrained Minimization/ Nonlinear Least Squares Library in ANSI C

https://code.google.com/p/optimc/

http://sourceforge.net/projects/optimc/

Author : Rafat Hussain

Contact : rafat.hsn@gmail.com

Table Of Contents

01 Introduction

OptimC is a C software package to minimize any unconstrained multivariable function. The algorithms implemented are Nelder-Mead,Newton Methods (Line Search and Trust Region methods), Conjugate Gradient and BFGS (regular and Limited Memory). Brent method is also available for single variable functions if the bounds are known. The package also implements Levenberg-Marquardt method for nonlinear least squares optimization. This LM method is a C port of the MINPACK Fortran code.

How To Obtain The Library

Git Repository

```
git clone https://code.google.com/p/optimc/
```

or

```
git clone git://git.code.sf.net/p/optimc/code optimc-code
```

or

```
git clone http://git.code.sf.net/p/optimc/code optimc-code
```

Or Download Zip File From

```
https://code.google.com/p/optimc/source/browse/
```

- or -

https://sourceforge.net/projects/optimc/files/

License : BSD 3 Clause

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer
in the documentation and/or other materials provided with the distribution.

3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.

02 Usage : How To Integrate OptimC In Your Code

Optimc code consists of C source files and their corresponding headers. You can directly use these files in your code by including "optimc.h" header in your code. Just make sure that all the files are in the same folder and your program can "see" them. For example, something like

```
gcc -Wall -c *.c
```

will work with GNU gcc compiler. It will build object files of all the files in the /src folder and you can link your project against these object files. This is more straightforward if you are using one of the modern IDEs as you can just plug all the files in your code and link "optimc.h" to your project files. The IDE will do the rest. If you are an expert programmer then you may want to skip the rest of this section.

Building Shared and Static Libraries on Linux

A Simple Static Library

If you are using GNU GCC compiler then something like

```
gcc -c *.c
```

will build the object files in the src folder. You can then package the object files in a liboptimc.a static library package using

```
ar rcs liboptimc.a *.o
```

A Simple Shared Library

```
gcc -fPIC -c *.c
gcc -shared -Wl,-soname,liboptimc.so.1 -o liboptimc.so.1.0 *.o
```

You may want to move lipoptimc.so.1.0 to a separate folder before creating symlinks.

```
ln -sf liboptimc.so.1.0 liboptimc.so
ln -sf liboptimc.so.1.0 liboptimc.so.1
```

If your folder is not on the path then you will have to export the path before executing your program.

```
export LD_LIBRARY_PATH=/OPTIMCFOLDERLOCATION/
```

Some useful links.

http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html

http://codingfreak.blogspot.com/2010/01/creating-and-using-static-libraries-in.html

http://www.techytalk.info/c-cplusplus-library-programming-on-linux-part-one-static-libraries/

http://www.techytalk.info/c-cplusplus-library-programming-on-linux-part-two-dynamic-libraries/

Building Shared and Static Libraries on Windows

Use IDE to build the libraries

I am mentioning this approach as most Windows programmers use one or more IDEs for their programming. All modern IDEs can create static and DLLs from source codes. eg., In Visual Studio you start out by creating an empty project, then by adding all the source and header files followed by "Build Solution".

Link - http://msdn.microsoft.com/en-us/library/ms235636.aspx

http://msdn.microsoft.com/en-us/library/ms235627.aspx

It is equally straightforward to create libraries in Eclipse , Codeblocks and other IDEs.

Working with Cygwin

A Static library (.a) build is identical to that with linux.

A Simple Static Library

If you are using GNU GCC compiler then something like

```
gcc -c *.c
```

will build the object files in the src folder. You can then package the object files in a liboptimc.a static library package using

```
ar rcs liboptimc.a *.o
```

```

```

A Simple DLL in Cygwin

This will build a simple standalone DLL.

```
gcc -c -fPIC *.c
gcc -shared -o liboptimc.dll *.
```

Check this link for more options, specially if you want to build the DLL as an export library.

http://cygwin.com/cygwin-ug-net/dll.html

03 Optimize : Optimization Class And Functions

```
opt_object obj = opt_init(N);  // Initialize optimization object obj .
//N - Number of Variables
```

Optimization Object obj , thus created, can be used as many time as you wish for any problem with N variables.

Execution

```
optimize(obj,function,gradient,N,xi,method); // Optimize

obj - Optimization obj previously created
function and gradient - see testfunctions.c for more information on how to write functions and gradient functions in C. NULL is an acceptable value for gradient.
N - problem size
xi - Initial point
method accepts values between 0 and 6 where -
Method 0 - Nelder-Mead
Method 1 - Newton Line Search
Method 2 - Newton Trust Region - Hook Step
Method 3 - Newton Trust Region - Double Dog-Leg
Method 4 - Conjugate Gradient
Method 5 - BFGS
Method 6 - Limited Memory BFGS
```

Optimization Object Parameters

```
int N; // Problem size
double objfunc; // Objective function value at the minima or termination point
double eps;// Machine Epsilon
double gtol;
double stol;
double ftol;
double xtol;// These four tolerance values are set based on machine epsilon value but users can set these values using setTOL, See below.
int MaxIter;// Maximum Iterations. Typically set to a maximum of 200* N or 1000. Can be set by user using setMaxIter. See below
```

int Iter;// Number of Iterations at the point of termination
int Method;// Method. See Above for value keys
int retval;// Return Value.
char MethodName[50];// Method Name
double xopt[1];// N-vector termination point. Could be the minima depending on the return value

Return Code

On program termination, obj->retval contains a return code.

Return Codes
Code 1 denotes probable success.
Codes 2,3 and 6 denote possible success.
Codes 0, 4 and 15 denote failure. [Code 4 may also occur in cases where the minima is realized but
convergence is not achieved due to tolerance values being too small. It is recommended that you use
another method if you suspect that such a scenario has occured.]
0 - Input Error
1 - df(x)/dx <= gtol achieved so current point may be the local minima.
2 - Distance between the last two steps is less than stol or |xf - xi| <= stol so the point may be the local minima.
3 - Global Step failed to locate a lower point than the current point so it may be the local minima.
4 - Iteration Limit exceeded. Convergence probably not achieved.
6 - Function value drops below ftol (relative functional tolerance).
15 - Overflow occurs. Try a different method.

Functions associated with Optimization class

void summary(opt_object obj); // Returns concise imformation.
void setMaxIter(opt_object obj,int MaxIter);// Sets Maximum Iterations value. May be useful if the return value is 4.

```
void setTOL(opt_object obj,double gtol,double stol,double ftol,double xtol); //
Can be use to modify tolerance values.
```

Deallocate Object Obj when the computation is finished

```
free_opt(obj);
```

Example

A simple demonstration of optimization class using a two variable Freudenstein and Roth function starting at [0.5,2]. All 7 methods are used one by one and the result is printed using the summary function.

```
#include <stdio.h>
#include <stdlib.h>
#include "../header/optimc.h"
#include "testfunctions.h"

int main(void) {
      int N,j,i;
      double *xi,*xf;
      opt_object optim;

      N = 2;

      xi = (double*) malloc(sizeof(double) * N);
      xf = (double*) malloc(sizeof(double) * N);

      for (i = 0; i < N;++i) {
            xi[i] = 2;
      }
      xi[0] = 0.5;

      optim = opt_init(N);

      for(j = 0; j < 7;++j) {
            optimize(optim,froth,NULL,N,xi,j);
            summary(optim);
            for (i = 0; i < N;++i) {
                  xi[i] = 2;
            }
            xi[0] = 0.5;
```

```
        }

        free(xi);
        free(xf);
        free_opt(optim);

        return 0;
}
```

04 NLS : Nonlinear Least Squares Class

Nonlinear Least Squares using Levenberg-Marquardt Method

```
nls_object obj = nls_init(M,N);  // Initialize optimization object obj .
//M - Number of Functions
//N - Number of Variables
```

Nonlinear least squares optimization Object obj , thus created, can be used as many times as you wish for any problem with M functions and N variables.

Execution

```
nls(obj,function,jacobian,xi);

-OR-

nls_scale(obj,function,jacobian,diag,xi);

obj - Optimization obj previously created
function and jacobian - see testfunctions.c for more information on how to write functions and jacobian functions in C. NULL is an acceptable value for jacobian.
xi - Initial point
diag - is an array of length N. diag must contain positive entries that serve as multiplicative scale factors for the variables.
```

Object Parameters

```
int M; //Number of Functions
int N; // Number of Variables
double eps; // Machine Epsilon
double epsfcn; //is an input variable used in determining a suitable
// step length for the forward-difference approximation. Default eps = epsfcn
double factor; //Determines the initial step bound
double gtol; // Tolerances
double ftol;
double xtol;
```

```
int MaxIter; // Maximum Number of Iterations allowed
int Maxfev; // Maximum function evaluations
int Iter; // Number of Iterations at termination
int nfev; // Number of function evaluations at termination
int njev; // Number of Jacobian evaluations at termination
int ldfjac; // Leading dimension of the Jacobian matrix >= M. (Default M)
int mode;// mode = 1 internal scaling (nls). mode = 2 for nls_scale
int retval;// Return Value
double xopt[1]; // Output. N-vector termination point.
```

Return Code

```
0  improper input parameters.
1 both actual and predicted relative reductions in the sum of squares are at most ftol.
2  relative error between two consecutive iterates is at most xtol.
3  conditions for info = 1 and info = 2 both hold.
4  the cosine of the angle between fvec and any column of the jacobian is at most gtol in absolute value.
5  number of calls to fcn with iflag = 1 has reached maxfev.
6 ftol is too small. no further reduction in the sum of squares is possible.
7  xtol is too small. no further improvement in the approximate solution x is possible.
8  gtol is too small. fvec is orthogonal to the columns of the jacobian to machine precision.
```

Functions associated with nls class

```
void setnlsTOL(nls_object obj,double gtol,double ftol,double xtol);// Can be use to modify tolerance values.

void nlssummary(nls_object obj);// Returns concise imformation.
```

Deallocate Object Obj when the computation is finished

```
free_nls(obj);
```

Example

```
#include <stdio.h>
#include <stdlib.h>
#include "../header/optimc.h"
#include "testfunctions.h"

int main(void) {

	int M,N,i;

	nls_object obj;

	//Matlab Example http://www.mathworks.in/help/optim/ug/lsqnonlin.html See
Examples

	M = 10; N = 2;
	obj = nls_init(M,N);// M >= N
	double xi[2] = { 0.3, 0.4 };
	double diag[2] = {1.0,1.0};

	// Using Custom Scaling nls_scale function that allows diag values to be set
by the user.
	// The program will only accept strictly positive diagonal values. diag has
the size N

	nls_scale(obj,fk10,fk10jac,diag,xi);
	for(i = 0; i < obj->N;++i) {
		printf("%g ",obj->xopt[i]);
	}
	nlssummary(obj);

	free_nls(obj);

	// Using Internal Scaling. Function nls.

	obj = nls_init(M,N);

	nls(obj,fk10,fk10jac,xi);
	for(i = 0; i < obj->N;++i) {
		printf("%g ",obj->xopt[i]);
	}
	nlssummary(obj);

	free_nls(obj);

	return 0;
}
```

05 List Of Other Functions

fminsearch

fminsearch is the Nelder-Mead method of minimizing a N-variable problem. No gradient information is needed so it can be useful in many situations where N is relatively small (N < 10-15) and gradient is hard to compute.

```
retval =  fminsearch(double (*funcpt)(double *,int),int N,double *xi,double *xf);
```

where

funcpt - function to be minimized
N - Number of variables
xi - Initial Point
xf - Minimized point
retval - Integer Value that denotes whether a minima was reached.

Return Code

On program termination, retval contains a return code.

Return Codes
Code 1 denotes probable success.
Codes 2,3 and 6 denote possible success.
Codes 0, 4 and 15 denote failure. [Code 4 may also occur in cases where the minima is realized but
convergence is not achieved due to tolerance values being too small. It is recommended that you use
another method if you suspect that such a scenario has occured.]
0 - Input Error
1 - df(x)/dx <= gtol achieved so current point may be the local minima.
2 - Distance between the last two steps is less than stol or |xf - xi| <= stol so the point may be the local minima.
3 - Global Step failed to locate a lower point than the current point so it may be the local minima.
4 - Iteration Limit exceeded. Convergence probably not achieved.

6 - Function value drops below ftol (relative functional tolerance).
15 - Overflow occured. Try a different method.

fminsearch example

```
/*
ex1_fminsearch : demonstrates usage of fminsearch. fminsearch is a simple
implementation of
Nelder-Mead method and it requires only the knowledge of function values and an
initial
point.

retval = fminsearch(double (*funcpt)(double *,int),int N,double *xi,double *xf);

funcpt - function to be minimized. See below and check testfunctions.c for
function examples.
N - Problem size. Number of Variables. fminsearch works best for N <= 15
xi - Initial point (N-dimensional)
xf - Minimized point (N-dimensional)
retval - Return Value ( 1 for success. 0 and 4 for failure. 0 - Input error. 4 -
Maximum
Iterations exceeded)

*/

#include <stdio.h>
#include <stdlib.h>
#include "../header/optimc.h"
#include "testfunctions.h"

int main(void) {
  int N,i,retval;
  double *xi,*xf;


  N = 4;

  xi = (double*) malloc(sizeof(double) * N);
  xf = (double*) malloc(sizeof(double) * N);

  xi[0] = 3; xi[1] = -1; xi[2] = 0; xi[3] = -1;


  retval = fminsearch(quartic,N,xi,xf);

  printf("Powell's Quartic Function \n");
  printf("Return Value %d \nObjective Function %g \n",retval,quartic(xf,N));

  printf("Function minimized at : ");
```

```
  for(i = 0; i < N;++i) {
    printf("%g ",xf[i]);
  }
  printf("\n \n");
  free(xi);
  free(xf);
  printf("\nRosenbrock Function \n");
  N = 2;
  xi = (double*) malloc(sizeof(double) * N);
  xf = (double*) malloc(sizeof(double) * N);

  xi[0] = -1.2; xi[1] = 1;
        retval = fminsearch(rosenbrock,N,xi,xf);

  printf("Return Value %d \nObjective Function %g \n",retval,rosenbrock(xf,N));

  printf("Function minimized at : ");
  for(i = 0; i < N;++i) {
    printf("%g ",xf[i]);
  }


  return 0;
}
```

fminbnd

fminbnd is the method implemented by Richard P Brent to minimize a single-variable function within a pair of bounds.

```
oup =  fminbnd(double (*funcpt)(double),double a, double b);
```

where

funcpt - Single-variable function to be minimized
a - lower bound
b - upper bound
oup - minima

The function outputs a minima only.

fminbnd example

```
/*
fminbnd - minimizes a single variable function within a bound given by a and b.

 */

#include <stdio.h>
#include <stdlib.h>
#include "../header/optimc.h"
#include "testfunctions.h"

int main(void) {
  double a,b,oup;

  a = 0.3;
  b = 1;

  oup = fminbnd(humps,a,b);
  printf("OUP %g \n",oup);

  return 0;
}
```

fminunc

fminunc is the functional equivalent of optimize object but with lesser flexibility. It implements all 7 methods.

```
retval fminunc(double (*funcpt)(double *,int),void(*funcgrad)(double *,
int,double *),int N,double *xi,int method,double *xf);
```

where

funcpt - function to be minimized
funcgrad - Numerical Gradient. NULL is an acceptable value but it is recommended that you provide a function to calculate the gradient.
N - problem size
xi - Initial point
method accepts values between 0 and 6 where -
Method = 0 - Nelder-Mead
Method = 1 - Newton Line Search
Method = 2 - Newton Trust Region - Hook Step
Method = 3 - Newton Trust Region - Double Dog-Leg
Method = 4 - Conjugate Gradient

Method = 5 - BFGS
Method = 6 - Limited Memory BFGS
xf - Minimized Point
retval - Integer Value that denotes whether a minima was reached.

fminunc example

```
#include <stdio.h>
#include <stdlib.h>
#include "../header/optimc.h"
#include "testfunctions.h"

int main(void) {
        int N,j,i,retval;
        double *xi,*xf;

        N = 4;

        xi = (double*) malloc(sizeof(double) * N);
        xf = (double*) malloc(sizeof(double) * N);

        for (i = 0; i < N;++i) {
                xi[i] = 1;
        }
        printf("\n\n%-25s%-20s%-20s \n","Method","Return Value","Objective
Function");



  for(j = 0; j < 7;++j) {
      retval = fminunc(myvalue,myvaluegrad,N,xi,j,xf);
      printf("\n\n%-25d%-20d%-20g \n",j,retval,myvalue(xf,N));
      printf("Function Minimized At : ");
      for (i = 0; i < N;++i) {
        printf("%g ",xf[i]);
      }

      for (i = 0; i < N;++i) {
    xi[i] = 1;
      }

  }

  free(xi);
  free(xf);
  return 0;
}
```

fminnewt

fminnewt is based on the Dennis/Schnabel's implementation of Newton line search and trust region methods as developed in their book.

The method tries to circumvent Newton methods problems in dealing with badly scaled functions. The method requires three additional parameters.

```
int fminnewt(double (*funcpt)(double *,int),void(*funcgrad)(double *, int,double *),int N,double *xi,
             double delta,double *dx,double fsval,int method,double *xf);
```

dx - Expected order of magnitude of the solution. dx is a vector of size N with default value [1,1,...,1]. If you expect the minima to occur in the vicinity of [0.007,1100] for N = 2, then dx = [0.001,1000] should suffice. For most problems dx = [1,1,1..,1] should work.

fsval - The anticipated value of the objective function near the minimizer. Default Value is 1.0 and it should work in most cases.

delta - Trust Region Initial Radius. Default Value is -1

funcpt - function to be minimized
funcgrad - Numerical Gradient. NULL is an acceptable value but it is recommended that you provide a function to calculate the gradient.
N - problem size
xi - Initial point
method accepts values between 0 and 6 where -
Method = 1 - Newton Line Search
Method = 2 - Newton Trust Region - Hook Step
Method = 3 - Newton Trust Region - Double Dog-Leg
xf - Minimized Point
retval - Integer Value that denotes whether a minima was reached.

Return Code

On program termination, retval contains a return code.

Return Codes
Code 1 denotes probable success.
Codes 2,3 and 6 denote possible success.
Codes 0, 4 and 15 denote failure. [Code 4 may also occur in cases where the minima is realized but
convergence is not achieved due to tolerance values being too small. It is recommended that you use
another method if you suspect that such a scenario has occured.]
0 - Input Error
1 - df(x)/dx <= gtol achieved so current point may be the local minima.
2 - Distance between the last two steps is less than stol or |xf - xi| <= stol so the point may be the local minima.
3 - Global Step failed to locate a lower point than the current point so it may be the local minima.
4 - Iteration Limit exceeded. Convergence probably not achieved.
6 - Function value drops below ftol (relative functional tolerance).
15 - Overflow occured. Try a different method.

fminnewt example

```
#include <stdio.h>
#include <stdlib.h>
#include "../header/optimc.h"
#include "testfunctions.h"

int main(void) {
        int N,j,i,retval;
  double fsval,delta;
        double *xi,*xf,*dx;

        N = 2;

        xi = (double*) malloc(sizeof(double) * N);
        xf = (double*) malloc(sizeof(double) * N);
  dx = (double*) malloc(sizeof(double) * N);

        for (i = 0; i < N;++i) {
                xi[i] = 1;
```

```
    //dx[i] = 1;
        }
  fsval = 1;// Default Value
  delta = -1.0;//Default Value
  dx[0] = 1.0e05; dx[1] = 1.0e-05;
        printf("\n\n%-25s%-20s%-20s \n","Method","Return Value","Objective
Function");



  for(j = 1; j < 4;++j) {
      retval = fminnewt(brown,browngrad,N,xi,delta,dx,fsval,j,xf);
      printf("\n\n%-25d%-20d%-20g \n",j,retval,brown(xf,N));
      printf("Function Minimized At : ");
      for (i = 0; i < N;++i) {
        printf("%g ",xf[i]);
      }

      for (i = 0; i < N;++i) {
    xi[i] = 1;
      }
  }

  free(xi);
  free(xf);
  free(dx);
  return 0;
}
```

levmar

Function levmar is a simple Levenberg-Marquardt C routine to access C translations of MINPACK lmder and lmdif functions. levmar simply returns the end point and the termination code. For more options use the object-oriented approach employed by nls and nls_scale.

```
int levmar(void (*funcmult)(double *,int,int,double *),void(*jacobian)(double *,
int,int,double *),
            double *xi,int M, int N,double *xf);}}}
```

where

funcmult - Routine that returns M function values for M functions in N variables.
jacobian - of function funcmult. NULL is an acceptable value and the code will compute finite difference Jacobians.
xi - Initial Point. Length N.

M - Number of Functions
N - Number of Variables where M >= N
xf - Termination Point
retval - Return Value.

Return Code

0 improper input parameters.
1 both actual and predicted relative reductions in the sum of squares are at most ftol.
2 relative error between two consecutive iterates is at most xtol.
3 conditions for info = 1 and info = 2 both hold.
4 the cosine of the angle between fvec and any column of the jacobian is at most gtol in absolute value.
5 number of calls to fcn with iflag = 1 has reached maxfev.
6 ftol is too small. no further reduction in the sum of squares is possible.
7 xtol is too small. no further improvement in the approximate solution x is possible.
8 gtol is too small. fvec is orthogonal to the columns of the jacobian to machine precision.

levmar example

```
#include <stdio.h>
#include <stdlib.h>
#include "../header/optimc.h"
#include "testfunctions.h"

int main(void) {
	int M,N,i,ret;

	M = N = 3;

	double xi[3] = {-1,0,0};
	double xf[3] = {0,0,0};

	//levmar simply returns the end point and the termination code
	// For more options use the object-oriented approach employed by nls and
nls_scale
	ret = levmar(fpowell,NULL,xi,M,N,xf);

	printf("Return Value : %d \n",ret);
	printf("\n");
	printf("Termination At : ");
```

```
	for(i = 0; i < N;++i) {
		printf("%g ",xf[i]);
	}

	return 0;
}
```

06 References

This list is a work in progress.

Algorithms For Minimization Without Derivatives, Richard P. Brent
Numerical Optimization (Springer Series in Operations Research and Financial Engineering) , Jorge Nocedal (Author), Stephen Wright
Argonne National Laboratory. MINPACK Project. June 1983 ,Jorge J. More', David J. Thuente
J. A. Nelder and R. Mead, A simplex method for function minimization. Compuer Journal,7,308-313
Journal of the Royal Statistical Society. Series C (Applied Statistics), Vol. 20, No. 3 (1971), pp. 338-345, R O'Neill
Numerical Methods for Unconstrained Optimization and Non-Linear Equations, Dennis, Schnabel
"A Limited Memory Algorithm for Bound Constrained Optimization", SIAM Journal on Scientific Computing 16 (5): 1190–1208, Byrd, Richard H.; Lu, Peihuang; Nocedal, Jorge; Zhu, Ciyou
Practical methods of optimization (2nd ed.),Fletcher, Roger (1987),

MINPACK http://www.netlib.org/minpack/
EISPACK http://www.netlib.org/eispack/

07 Appendix : How To Calculate Function , Gradient And Jacobian Values

Function And Gradient Values Calculations Example

Function Evaluation is of the type

```
f = (*funcpt)(double *x,int N)
```

where f returns the value (double) of the function at the N-dimensional point x.

Powell's quartic function is a function with N = 4 variables.

```
double quartic(double *x,int N) {
        double f;
        // Powell's Quartic Function
        f = (x[0] + 10 * x[1] ) * (x[0] + 10 * x[1] ) + 5 * (x[2] - x[3]) * (x[2]
- x[3])
        + (x[1] - 2 * x[2]) * (x[1] - 2 * x[2]) * (x[1] - 2 * x[2]) * (x[1] - 2 *
x[2])
        + 10 * (x[0]-x[3]) * (x[0]-x[3]) * (x[0]-x[3]) * (x[0]-x[3]) ;
        return f;
}
```

Gradient evaluation function is of the type.

```
(*funcgrad)(double *x,int N,double *g)
```

where g is the N-dimensional gradient vector calculated at point x. It is recommended that you provide gradient calculation as shown below but NULL is an acceptable value and the code will calculate the gradient numerically using forward difference method.

```
void quarticgrad(double *x,int N,double *g) {
        double t1,t2,t3,t4;
        // Gradient of Powell's Quartic Function
        t1 = (x[0] + 10 * x[1]);
        t2 = (x[2] - x[3]);
        t3 = (x[1] - 2 * x[2]);
        t4 = (x[0] - x[3]);

        g[0] = 2*t1 + 40*t4*t4*t4;
        g[1] = 20*t1 + 4*t3*t3*t3;
        g[2] = 10*t2 - 8*t3*t3*t3;
        g[3] = -10*t2 - 40*t4*t4*t4;
```

```
}
```

M Functions in N variables and Jacobian Evaluation Example

Multiple Functions evaluation is of the type

```
(*funcmult)(double *x,int M,int N,double *f)
```

M-dimensional vector f returns values of M functions in N variables at N-dimensional point x. eg.,

```
void fk10(double *x, int M, int N, double *f) {
      int i;

      for (i = 1; i <= M; ++i) {
            f[i - 1] = 2 + 2*i - exp(i*x[0]) - exp(i*x[1]);
      }
}
```

M*N Jacobian evaluation is of the type.

```
(*jacobian) (double *x,int M,int N,double *jac)
```

jac is a M*N vector that contains values of the Jacobian matrix at N-dimensional point x. eg.,

```
void fk10jac(double *x, int M, int N, double *f) {
      int i;

      for(i = 1; i <= M;++i) {
            f[(i-1)*N] = -i * exp(i*x[0]);
            f[(i-1)*N+1] = -i * exp(i*x[1]);
      }

}
```

NULL is an acceptable value instead of a C routine calculating Jacobian matrix but it is recommended that you provide such a function for better performance.