Phase reconstruction from intensity or amplitude measurements of the electromagnetic field [1]

Alexey A. Novikov

July 16, 2009

[1]Special thanks to my supervisors - Professor Claude Dedaban and Professor Christian Pichot

1 Introduction

The phase problem is the name given to the problem of loss of information (the phase) from a physical measurement. The name itself comes from the field of x-ray crystallography, where the phase problem has to be solved for the determination of a structure from diffraction data. The phase problem is also met in the fields of imaging and signal processing. Various approaches have been developed over the years to solve it.

Let $S'(R)$ be the space of tempered distribution on R and set

$$A = \{g \in S'(R) : h \in L^1_{loc}(R)\}$$

Here h denotes the Fourier Transform, using the convention

$$h(k) = \int_{-\infty}^{+\infty} g(t) e^{ikt} dt$$

for $g \in L^1(R)$.

Recall that for any $g \in S'(R)$, the Fourier transform is defined as another member of $S'(R)$. A first statement of the problem in which we are interested is as follows.

Phase determination problem. Find $g \in A$ given $|h(k)|$ for $k \in R$

By the Fourier inversion theorem, g is uniquely determined by the complex valued function $h(k)$ for $k \in R$, thus the essence of the matter is to determine the phase $\phi(k) = \arg h(k)$ from the amplitude $r(k) = |h(k)|$. Problems of phase reconstuction arise in a number of interesting application areas.

Now stated in this form, the problem is quite hopeless. If, for example, we fix any non-negative $r \in L^2(R)$, and

$$G(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} r(k) e^{-i(\theta(k)+kt)}$$

then, for any measurable real valued function θ, we have $G \in L^2(R)$ and $|H(k)| = r(k)$ on R. That is to say, in this generality, the phase and the amplitude are completely independent of each other. On the other hand, particular phase determination problems always have more specific features which need to be taken into account, which is to say that, typically, there is a smaller class $B \subset A$ of admissible solutions g, so that we really mean to solve the following problem.

Constrained phase determination problem. Find $g \in B$ given $|h(k)|$ for $k \in R$

If the constraint set B is sufficiently small, then one may hope that the phase of h is uniquely detemined by its amplitude, or at least that the phase is constrained enough by the amplitude to considerably reduce the degree of non-uniqueness. Each different choice of admissible class B leads to a different problem with its own special features and difficulties. The admissibility criteria in the definition of B might be in the form of explicit conditions to be satisfied by the solution g, but could also incorporate other kinds of constraint, such as extra conditions related to the behaviour of h, which restrict the admissible solutions in an implicit way.

Phase identification problems of the type just described may be regarded as a special case of the more general type of problem in which one seeks to recover a function g from limited knowledge about both g and its Fourier transform h. If we write, for example, $g(t) = \rho(t)e^{i\theta(t)}$ and as above $h(k) = r(k)e^{i\phi(k)}$, then we may ask what information about the four functions ρ, θ, r, ϕ suffices to determine g uniquely. Obviously, either pair (ρ, θ) or (r, ϕ) is enough. In the applications discussed below, g is often real valued, so that we are attempting to recover g from the prescribed function $r(k)$ with $\theta(t) = 0$ or π. More generally, one might wish to recover g from the pair (r, ρ). Finally, it may happen that some or all of these functions are known only over a portion of their domains.

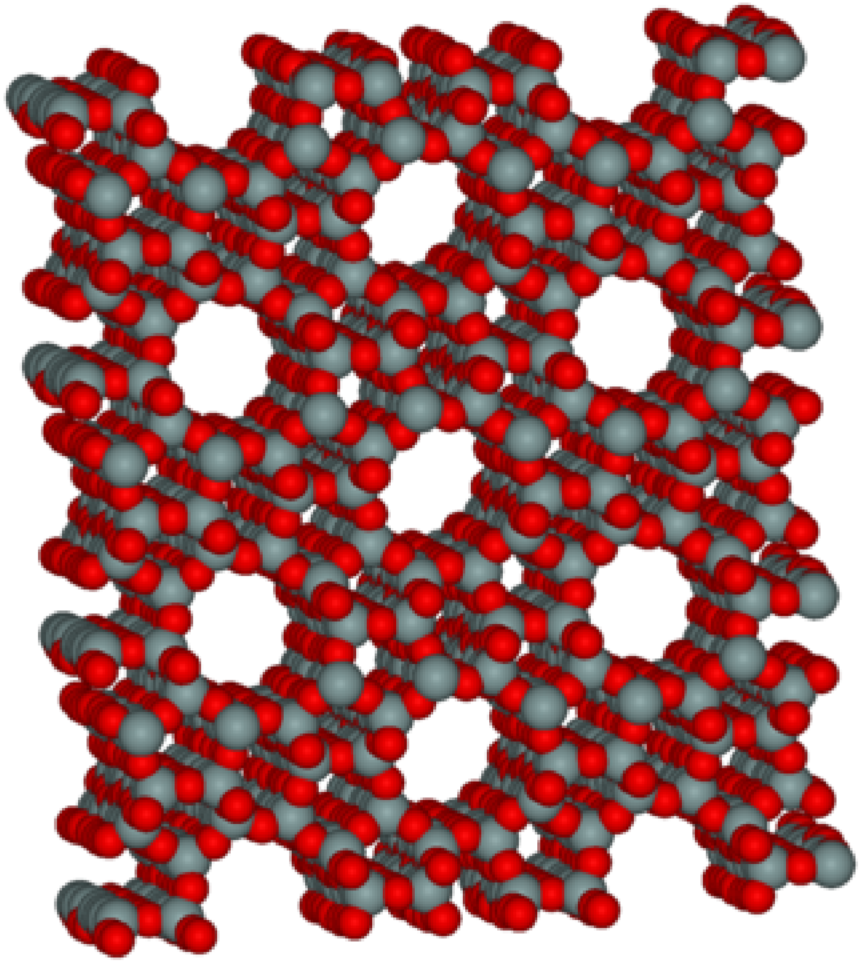
2 Examples

The phase retrieval problem is of wide interest because there are a variety of physically important problems in which essential quantities may be regarded as complex valued functions are thus characterized by amplitude and phase, where the phase is difficult or impossible to measure, while the amplitude is easily accessible to measurement or subject to some simple kind of inference. If knowledge of the phase is required for some reason, then we are faced with the constrained phase determination problem described in the first section, where the class B is defined in such a way as to incorporate constraints which are specific to each particular case.

Such problems arise frequently in optics, where the complex valued function represents a propagating electromagnetic wave which is generally a function of position. When such a wave is scattered off an object, as in an electron microscope, then information about the shape of the object is propated by the wave and is encoded in both the amplitude and the phase. Only the wave amplitude may be directly measured, and so one seeks to recover the lost phase information in order to characterize the object as thoroughly as possible. The underlying function $g(t)$ (here t may be a two- or three-dimensional variable) can often be assumed to be a non-negative function of compact support.

A related problem in lens design has recently been studied. One is given a real non-negative function $f(x),\ x \in \Omega \subset R^2$ representing the transparency of a lens, and a desired far-field intensity pattern $\Phi(\xi)$ and we seek to find a function $\phi(x),\ x \in \Omega$, representing the thickness of the lens so that the Fourier transform of $fe^{i\phi}$ has amplitude Φ. The constraint set B will then consist of functions g of compact support for which $|g|$ is prescribed.

One other important example arises in x-ray crystallography. Here one seeks to recover the electron density function ρ characterizing a certain crystal structure, which is taken to be a periodic function on R^3. In an x-ray diffraction experiment one measures the so-called structure factors, which are essentially the magnitude of the Fourier coefficients of ρ or, equivalently, the amplitude of the Fourier transform of ρ understood as a sum of delta functions supported on an appropriate lattice in R^3. The class B will consist of positive, periodic functions.



X-ray crystallography can locate every atom in a zeolite, an aluminosilicate with many important applications, such as water purification.

The most important application of phase retrieval is X-ray crystallography, awarded of several Nobel prizes.

3 Mathematical Model

Given $Y \in R^3$, a set of points; n and m are given integers, such that $m \subset N,\ n \subset N,\ n \leq m$

ϕ_k is a family of functions: $\phi_k\ :\ Y \rightarrow C \qquad k\ =\ 1, 2, ...,\ n$

Y_m is a set of m points: $Y_m = \{x_p \in Y p = 1, 2, ...,\ m\}$

Let

$$e(u, x) = \sum_{k=1}^{n} u_k\ \phi_k(x)$$

and

$$h(u, x) = |e(u, x)|^2 = e(u, x)\ \overline{e(u, x)}$$

Thus,

$$h(u, x) = \left(\sum_{k=1}^{n} u_k\ \phi_k(x)\right)\ \overline{\left(\sum_{k=1}^{n} u_k\ \phi_k(x)\right)}$$

The problem is: given m positive real numbers γ_p such that $\gamma_p \in R^+$. Find n complex numbers u_k so that

$$h(u, x_p) = \gamma_p \qquad p = 1, 2, ..., m$$

In order to solve the problem we should minimize the following functional:

$$J(u) = \sum_{p=1}^{m} (h(u, x_p) - \gamma_p)^2$$

For that we should perform the following steps:

1) Get the expressions $\frac{\partial J}{\partial \alpha_l}$

2) Get the descent direction d associated with Fletcher-Reeves modification

3) Use Newton method to find the mininum in the descent direction

First of all, we should write the expression of the derivative of the cost function with respect to the real parts of the vector u

$$\frac{\partial J}{\partial \alpha_R} = 2\sum_{p=1}^{m} \left(h(u, x_p) - \gamma_p\right) \frac{\partial h}{\partial \alpha_R}$$

where the derivatives of h with respect to the real parts of the vector u are as the following

$$\frac{\partial h}{\partial \alpha_R} = \frac{\partial}{\partial \alpha_R}\left(z\overline{z}\right) = \frac{\partial z}{\partial \alpha_R}\overline{z} + z\frac{\partial \overline{z}}{\partial \alpha_R}$$

where the derivatives of the auxiliary function z with respect to the real parts of the vector u are as the following

$$\frac{\partial z}{\partial \alpha_R} = \frac{\partial}{\partial \alpha_R}\left[\sum_{k=1}^{n}(\alpha_{2k-1} + i\alpha_{2k})\phi_k(x)\right] = \phi_{\frac{R}{2}+1}(x)$$

Its complex conjugate is

$$\frac{\partial \overline{z}}{\partial \alpha_R} = \overline{\phi_{\frac{R}{2}+1}}(x)$$

The expression of the derivative of the cost function with respect to the imaginary parts of the vector u

$$\frac{\partial J}{\partial \alpha_R} = 2\sum_{p=1}^{m} \left(h(u, x_p) - \gamma_p\right) \frac{\partial h}{\partial \alpha_I}$$

where the derivatives of h with respect to the imaginary parts of the vector u are as the following

$$\frac{\partial h}{\partial \alpha_I} = \frac{\partial}{\partial \alpha_I}\left(z\overline{z}\right) = \frac{\partial z}{\partial \alpha_I}\overline{z} + z\frac{\partial \overline{z}}{\partial \alpha_I}$$

where the derivatives of the auxiliary function z with respect to the imaginary parts of the vector u are as the following

$$\frac{\partial z}{\partial \alpha_I} = \frac{\partial}{\partial \alpha_I}\left[\sum_{k=1}^{n}(\alpha_{2k-1} + i\alpha_{2k})\phi_k(x)\right] = i\ \phi_{\frac{I+1}{2}}(x)$$

Its complex conjugate is

$$\frac{\partial \overline{z}}{\partial \alpha_I} = -i\ \overline{\phi_{\frac{I+1}{2}}}(x)$$

4 Optimization Methods

The class of conjugate direction methods can be viewed as being intermediate between the methods of steepest descent and Newton's method. The conjugate direction methods have the following properties:

1. Solve quadratics of n variables in n steps

2. The usual implementation, the conjugate gradient algorithm, requires no Hessian matrix evaluations

3. No matrix inversion and no storage of an $n \times n$ matrix required

Pts. 2 and 3 are very important in our case, because we have to handle with huge systems of equations.

The conjugate gradient algorithm does not use the prescribed conjugate directions, but instead computes the directions as the algorithm progresses. At each stage of the algorithm, the direction is calculated as a linear combination of the previous direction and the current gradient, in such a way that all the directions are mutually Q-conjugate, hence the name conjugate gradient algorithm. This calculation exploits the fact that for a quadratic function of n variables, we can locate the function minimizer by performing n searches along mutually conjugate directions.

The standard conjugate gradient algorithm is summarized below.

1. Select the initial point $u^{(0)}$

2. $g^{(0)} = \bigtriangledown J(u^{(0)})$

   If $g^{(0)} = 0$, stop, else set $d^{(0)} = -g^{(0)}$

3. $\alpha_k = -\dfrac{-g^{(k)T} d^{(k)}}{d^{(k)T} Q d^{k}}$

4. $u^{(k+1)} = u^{(k)} + \alpha_k \; d^{(k)}$

5. $g^{(k+1)} = \bigtriangledown J(u^{(k+1)})$

   If $g^{(k+1)} = 0$, stop

6. $\beta_k = \dfrac{g^{(k+1)T} Q d^{(k)}}{d^{(k)T} Q d^{k}}$

7. $d^{(k+1)} = -g^{(k+1)} + \beta_k \; d^{(k)}$

8. Go to step 3

For a quadratic function, the matrix Q, the hessian of the quadratic, is constant. However, for a general nonlinear function the Hessian is a matrix that has to be reevaluated at each iteration of the algorithm. This can be computationally very expensive. Thus, an efficient implementation of the conjugate gradient algorithm that eliminates the Hessian evaluation at each step is desirable.

Observe that Q appears only in the computation of the scalars α_k and β_k. Because

$$\alpha_k = \arg\min_{\alpha \geq 0} J(u^{(k)} + \alpha \; d^{(k)})$$

the closed form formula for α_k in the algorithm can be replaced by a numerical line search procedure. Therefore, we only need to concern ourselves with the formula for β_k. Fortunately, elimination of Q from the formula is possible and results in algorithms that depend only on the function and gradient values at each iteration.

There are some modifications of the formula which are based on algebraically manipulating the formula β_k in such a way that Q is eliminated.

Three well-known modifications are:

1. The Hestenes-Stiefel formula

$$\beta_k = \frac{g^{(k+1)T}\left[g^{(k+1)} - g^{(k)}\right]}{d^{(k}\left[g^{(k+1)} - g^{(k)}\right]}$$

2. The Polak-Ribiere formula

$$\beta_k = \frac{g^{(k+1)T}\left[g^{(k+1)} - g^{(k)}\right]}{g^{(k)T}g^{(k)}}$$

3. The Fletcher Reeves formula

$$\beta_k = \frac{g^{(k+1)T}g^{(k+1)}}{g^{(k)T}g^{(k)}}$$

The above formulas give us conjugate gradient algorithms that do not require explicit knowledge of the Hessian matrix Q. All we need the objective function and gradient values at each iteration.

We need a few more slight modifications to apply the algorithm. The termination criterion $\bigtriangledown J(u^{(k+1)}) = 0$ is not practical. A suitable practical stopping criterion needs to be used.

For nonquadratic problems, the algorithm will not usually converge in n steps, and as the algorithm progresses, the "Q-conjugacy" of the direction vectors will tend to deteiorate. Thus, a common practice is to reinitialize the direction vector to the negative gradient after every few iterations (e.g., n or $n+1$), and continue until the algorithm satisfies the stopping critetion.

A very important issue in minimization problems of nonquadratic functions is the line search. The purpose of the line search is to minimize $\phi_k(\alpha) = J(u^{(k+1)} + \alpha d^{(k)})$ with respect to $\alpha \geq 0$. A typical approach is to bracket or box in the minimizer and then estimate it. The accuracy of the line search is a critical factor in the performance of the conjugate gradient algorithm. If the line search is known to be inaccurate, the Hestenes-Stiefel formula for β_k is recommended.

In general, the choice of which formula for β_k to use depends on the objective function. For example, the Polak-Ribiere formula is known to perform far better than the Fleetcher-Reeves formula in some cases, but not in others.

In order to find the minimum in the descent direction we need to solve the following problem

$$\alpha_k = \arg\min_{\alpha \geq 0} J(u^{(k)} + \alpha\ d^{(k)})$$

Recall that the method of steepest descent uses only the first derivatives (gradients) in selecting a suitable search direction. This strategy is not always the most effective. If higher derivatives are used, the resulting iterative algorithm may perform better than the steepest descent method. Newton's method (sometimes called Newton-Raphson method) uses first and second derivatives and indeed does perform better than the steepest descent method if the initial point is close to the minimizer. The idea behind this method as follows. Given a starting point, we construct a quadratic approximation to the objective function that matches the first and second derivative values at the point. We then minimize the approximate (quadratic) function instead of the original objective function. We use the minimizer of the approximate function as the starting point in the next step and repeat the procedure iteratively.

Observe that the k-th iteration of Newton's method can be written in two steps as

1. Solve $F(u^{(k)})d^{(k)} = -g^{(k)}$ for $d^{(k)}$
2. Set $u^{(k+1)} = u^{(k)} + d^{(k)}$

To avoid confusions let us use ξ instead of α. So we need to solve the following problem:

$$\xi_k = \arg\min_{\xi \geq 0} J(u^{(k)} + \xi\ d^{(k)})$$

Assume that

$$v^{(k)} = u^{(k)} + \xi d^{(k)}$$

$$J(v^{(k)}) = 2 \sum_{p=1}^{m} \Big(h(v^{(k)}, x_p) - \gamma_p \Big)^2$$

Newton's method requires computing the gradient of J with respect to ξ on each iteration

$$\frac{\partial J(v^{(k)})}{\partial \xi} = 2 \sum_{p=1}^{m} \Big(h(v^{(k)}, x_p) - \gamma_p \Big) \frac{\partial h(v^{(k)}, x_p)}{\partial \xi}$$

The derivative of the function h with respect to ξ is

$$\frac{\partial h(v^{(k)}, x_p)}{\partial \xi} = \frac{\partial \theta}{\partial \xi} \overline{\theta} + \frac{\partial \overline{\theta}}{\partial \xi} \theta$$

The auxiliary function θ is

$$\theta = \sum_{k=1}^{n} v^{(k)} \phi_k(x)$$

The first derivative of θ with respect to ξ is

$$\frac{\partial \theta}{\partial \xi} = \frac{\partial}{\partial \xi} \sum_{k=1}^{n} v^{(k)} \phi_k(x) = \sum_{k=1}^{n} d_k \phi_k(x)$$

Its complex conjugate is

$$\frac{\partial \overline{\theta}}{\partial \xi} = \frac{\partial}{\partial \xi} \sum_{k=1}^{n} \overline{v^{(k)}}\ \overline{\phi_k(x)} = \sum_{k=1}^{n} \overline{d_k}\ \overline{\phi_k(x)}$$

Newton's method also requires computing of the second derivative of J with respect to ξ

$$\frac{\partial^2 J}{\partial \xi^2} = 2 \sum_{p=1}^{m} \left(\frac{\partial h(v^{(k)}, x_p)}{\partial \xi} \right)^2 + 2 \sum_{p=1}^{m} \left(h(v^{(k)}, x_p) - \gamma_p \right) \frac{\partial^2 (v^{(k)}, x_p)}{\partial \xi^2}$$

The second derivative of the function h with respect to ξ

$$\frac{\partial^2 h(v^{(k)}, x_p)}{\partial \xi^2} = 2 \frac{\partial \theta}{\partial \xi} \frac{\partial \overline{\theta}}{\partial \xi} = 2 \left| \frac{\partial \theta}{\partial \xi} \right|^2$$

The k-th iteration of the Newton's method is as follows

$$\xi_k = \xi_{k-1} + \frac{\partial J(\xi_{k-1})}{\partial \xi} \left(\frac{\partial^2 J(\xi_{k-1})}{\partial \xi^2} \right)^{-1}$$

5 Implementation and Technologies

Microsoft Visual Studio is an Integrated Development Environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer. It allows plug-ins to be added that enhance the functionality at almost every level - including adding support for source control systems (like Subversion and Visual SourceSafe) to adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

In the problem we need to handle with complex-valued functionals. Microsoft Visual Studio gives the possibility to work with them easily.

The standard library $< complex >$ defines the container template class complex and its supporting templates.

The most important functions of the library are

1. $abs()$ (extracts the modulus of a complex number)
2. $conj()$ (returns the complex conjugate of a complex number)
3. $exp()$ (returns the exponential function of a complex number)
4. $imag()$ (extracts the imaginary component of a complex number)
5. $real()$ (extracts the real component of a complex number)

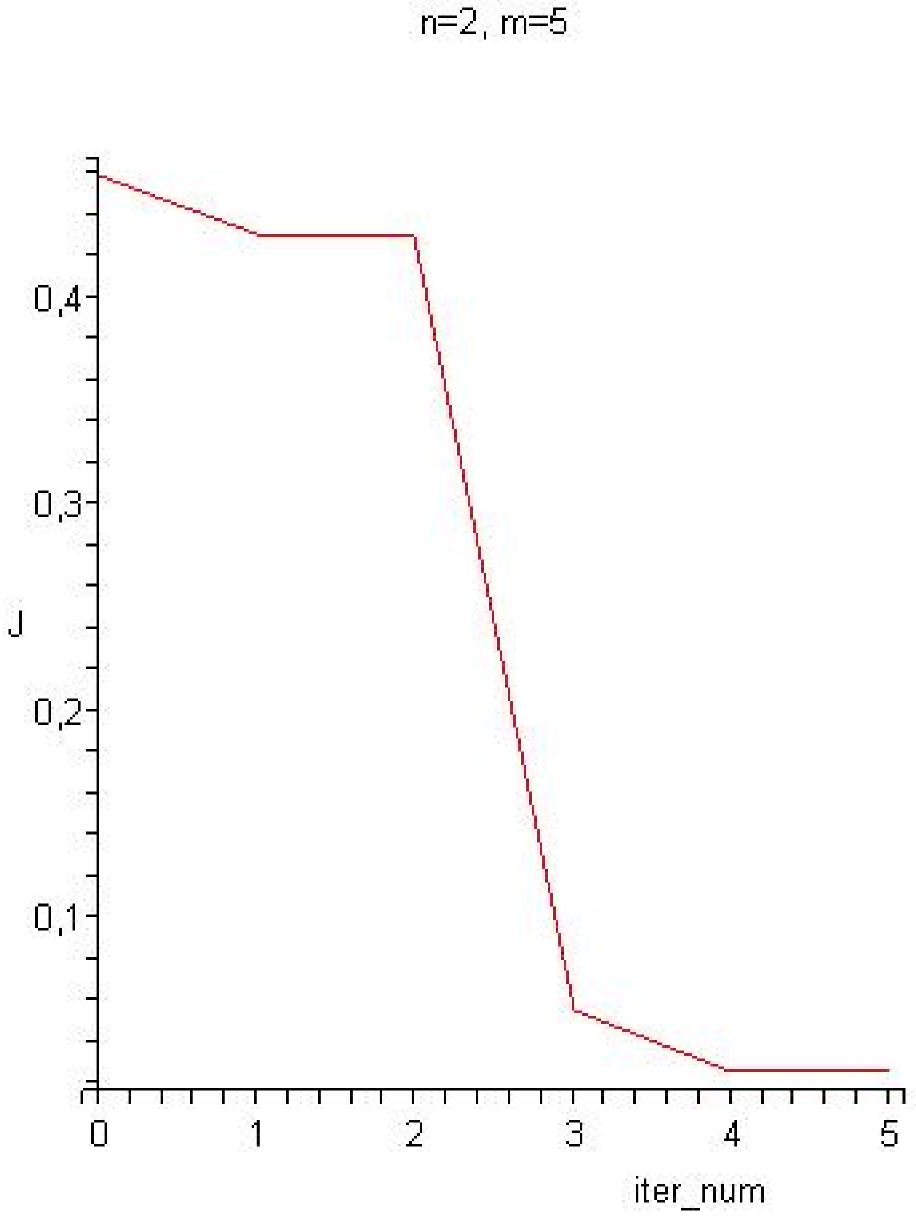
The construction $complex < double >$ describes an object that stores an ordered pair of objects both of type double, the first representing the real part of a complex number and the second representing the imaginary part.

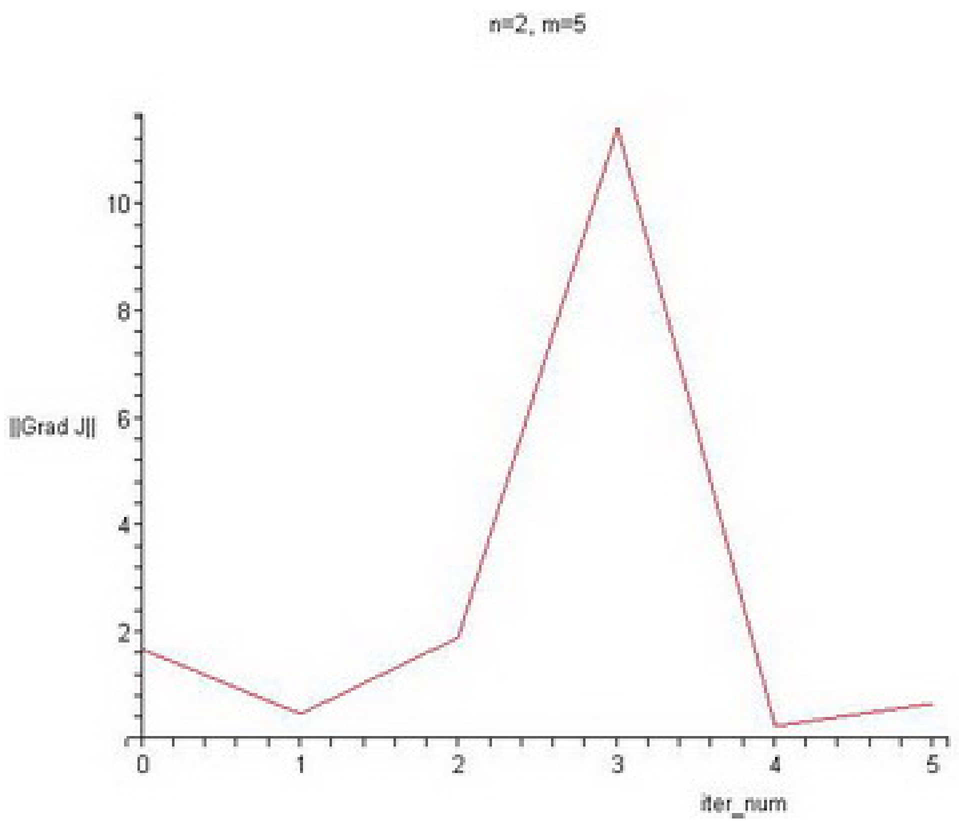
The explicit specialization of the template class complex to a complex class of type double differs from the template class only in the constructors it defines. The conversion from float to double is allowed to be implicit, but the conversion from long double to double is required to be explicit. The use of explicit rules out the initiation with type conversion using assignment syntax

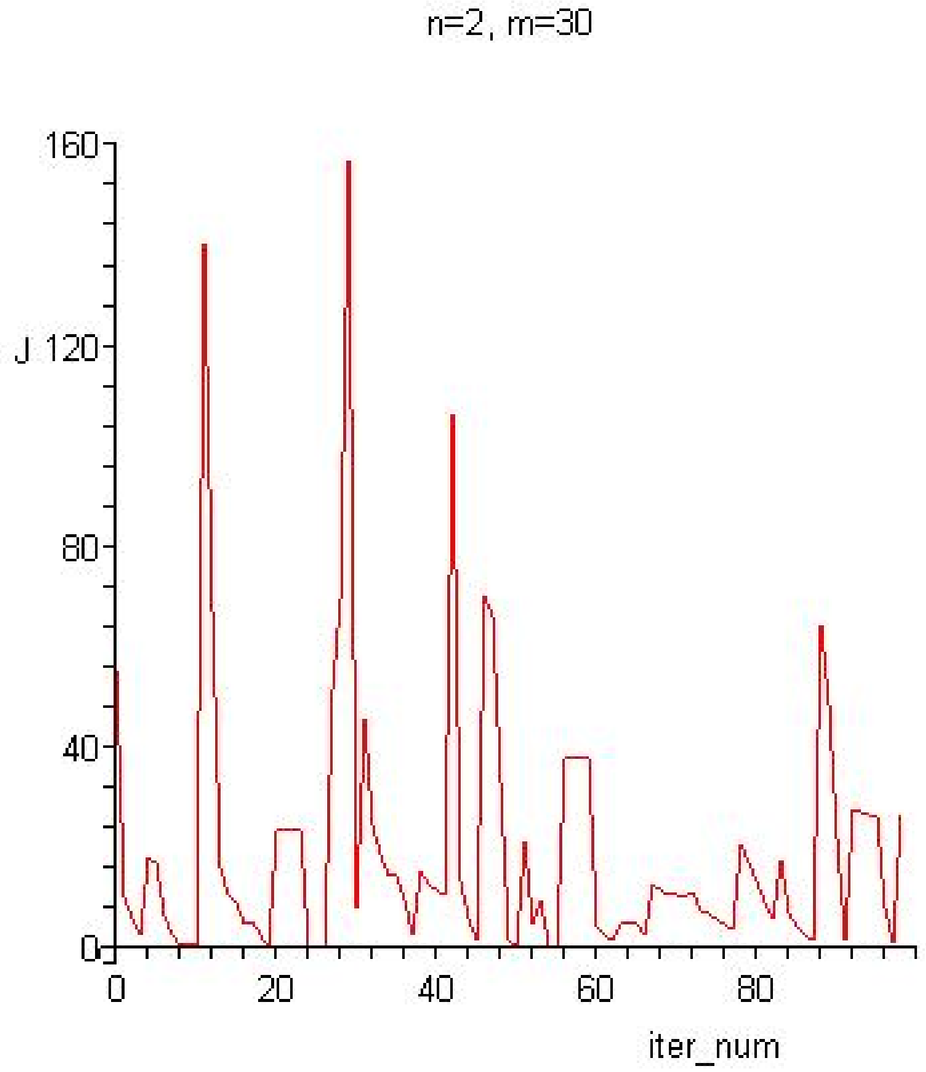
A simple example of using the basic functions and operations of the library $< complex >$ is shown below

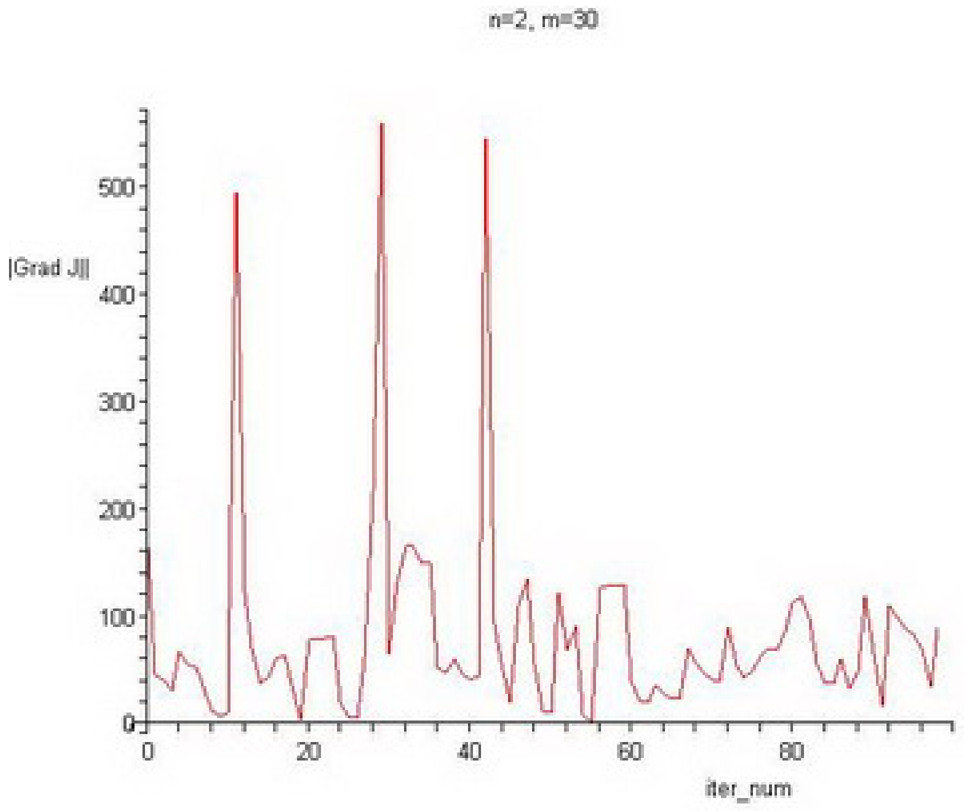
```
#include <complex>
#include <iostream>

int main( )
{
   using namespace std;
   double pi = 3.14159265359;

   // The first constructor specifies real & imaginary parts
   complex <double> c1 ( 4.0 , 5.0 );
   cout << "Specifying initial real & imaginary parts,\n"
        << " as type double gives c1 = " << c1 << endl;

   // The second constructor initializes values of the real &
   // imaginary parts using those of complex number of type float
   complex <float> c2float ( 4.0 , 5.0 );
   complex <double> c2double ( c2float );
   cout << "Implicit conversion from type float to type double,"
        << "\n gives c2double = " << c2double << endl;

   // The third constructor initializes values of the real &
   // imaginary parts using those of a complex number
   // of type long double
   complex <long double> c3longdouble ( 4.0 , 5.0 );
   complex <double> c3double ( c3longdouble );
   cout << "Explicit conversion from type float to type double,"
        << "\n gives c3longdouble = " << c3longdouble << endl;

   // The modulus and argument of a complex number can be recovered
   double absc3 = abs ( c3longdouble );
   double argc3 = arg ( c3longdouble );
   cout << "The modulus of c3 is recovered from c3 using: abs ( c3 ) = "
        << absc3 << endl;
   cout << "Argument of c3 is recovered from c3 using:\n arg ( c3 ) = "
        << argc3 << " radians, which is " << argc3 * 180 / pi
        << " degrees." << endl;
}
```

6 Results




n=2, m=5
||Grad J||
10
8
6
4
2
0
1
2
3
4
5
iter_num


n=2, m=30
J
160
120
80
40
0
0
20
40
60
80
iter_num



References

[1] M.S.Bazaraa, H.D.Sherali, and C.M.Shetty, Nonlinear Programming: Theory and Algorithms. New York: Wiley, Second ed., 1993

[2] Edwin K.P.Chong, Stanislaw H. Zak, An Introduction To Optimization. New York: Wiley, Second ed., 2001

[3] Michael V. Klibanov, Paul E. Sacks, and Alexander V. Tikhonravov, The Phase Retrieval Problem. Inverse Problems 11 (1-28), 1995

[4] `http://en.wikipedia.org/wiki/Phase_retrieval`

[5] `http://en.wikipedia.org/wiki/Phase_problem`

[6] `http://ca.wikipedia.org/wiki/Difraccio_de_raigs_X`

[7] http://en.wikipedia.org/wiki/Conjugate_gradient_method

[8] http://ca.wikipedia.org/wiki/Metode_de_Newton

[9] http://en.wikipedia.org/wiki/Newton's_method_in_optimization