Conjugate gradient methods and other optimization methods

Morten Hjorth-Jensen[1,2]

Department of Physics, University of Oslo[1]

Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University[2]

Mar 5, 2018

Motivation

Our aim with this part of the project is to be able to

- find an optimal value for the variational parameters using only some few Monte Carlo cycles
- use these optimal values for the variational parameters to perform a large-scale Monte Carlo calculation

To achieve this will look at methods like *Steepest descent* and the *conjugate gradient method*. Both these methods allow us to find the minima of a multivariable function like our energy (function of several variational parameters). Alternatively, you can always use Newton's method. In particular, since we will normally have one variational parameter, Newton's method can be easily used in finding the minimum of the local energy.

Simple example and demonstration

Let us illustrate what is needed in our calculations using a simple example, the harmonic oscillator in one dimension. For the harmonic oscillator in one-dimension we have a trial wave function and probability

$$\psi_T(x) = e^{-\alpha^2 x^2} \qquad P_T(x)dx = \frac{e^{-2\alpha^2 x^2}dx}{\int dx e^{-2\alpha^2 x^2}}$$

with α being the variational parameter. We obtain then the following local energy

$$E_L[\alpha] = \alpha^2 + x^2\left(\frac{1}{2} - 2\alpha^2\right),$$

which results in the expectation value for the local energy

$$\langle E_L[\alpha]\rangle = \frac{1}{2}\alpha^2 + \frac{1}{8\alpha^2}$$

Simple example and demonstration

The derivative of the energy with respect to α gives

$$\frac{d\langle E_L[\alpha]\rangle}{d\alpha} = \alpha - \frac{1}{4\alpha^3}$$

and a second derivative which is always positive (meaning that we find a minimum)

$$\frac{d^2\langle E_L[\alpha]\rangle}{d\alpha^2} = 1 + \frac{3}{4\alpha^4}$$

The condition

$$\frac{d\langle E_L[\alpha]\rangle}{d\alpha} = 0,$$

gives the optimal $\alpha = 1/\sqrt{2}$, as expected.

a) Derive the local energy for the harmonic oscillator in one dimension and find its expectation value.

b) Show also that the optimal value of optimal $\alpha = 1/\sqrt{2}$.

c) Repeat the above steps in two dimensions for N bosons or electrons. What is the optimal value of α?

Variance in the simple model

We can also minimize the variance. In our simple model the variance is

$$\sigma^2[\alpha] = \frac{1}{2}\alpha^4 - \frac{1}{4} + \frac{1}{32\alpha^4},$$

with first derivative

$$\frac{d\sigma^2[\alpha]}{d\alpha} = 2\alpha^3 - \frac{1}{8\alpha^5}$$

and a second derivative which is always positive

$$\frac{d^2\sigma^2[\alpha]}{d\alpha^2} = 6\alpha^2 + \frac{5}{8\alpha^6}$$

Computing the derivatives

In general we end up computing the expectation value of the energy in terms of some parameters $\alpha_0, \alpha_1, \ldots, \alpha_n$ and we search for a minimum in this multi-variable parameter space. This leads to an energy minimization problem *where we need the derivative of the energy as a function of the variational parameters*.
In the above example this was easy and we were able to find the expression for the derivative by simple derivations. However, in our actual calculations the energy is represented by a multi-dimensional integral with several variational parameters. How can we can then obtain the derivatives of the energy with respect to the variational parameters without having to resort to expensive numerical derivations?

Expressions for finding the derivatives of the local energy

To find the derivatives of the local energy expectation value as function of the variational parameters, we can use the chain rule and the hermiticity of the Hamiltonian.
Let us define

$$\bar{E}_\alpha = \frac{d\langle E_L[\alpha]\rangle}{d\alpha}.$$

as the derivative of the energy with respect to the variational parameter α (we limit ourselves to one parameter only). In the above example this was easy and we obtain a simple expression for the derivative. We define also the derivative of the trial function (skipping the subindex T) as

$$\bar{\psi}_\alpha = \frac{d\psi[\alpha]\rangle}{d\alpha}.$$

Derivatives of the local energy

The elements of the gradient of the local energy are then (using the chain rule and the hermiticity of the Hamiltonian)

$$\bar{E}_\alpha = 2\left(\langle\frac{\bar{\psi}_\alpha}{\psi[\alpha]}E_L[\alpha]\rangle - \langle\frac{\bar{\psi}_\alpha}{\psi[\alpha]}\rangle\langle E_L[\alpha]\rangle\right).$$

From a computational point of view it means that you need to compute the expectation values of

$$\langle\frac{\bar{\psi}_\alpha}{\psi[\alpha]}E_L[\alpha]\rangle,$$

and

$$\langle\frac{\bar{\psi}_\alpha}{\psi[\alpha]}\rangle\langle E_L[\alpha]\rangle$$

a) Show that

$$\bar{E}_\alpha = 2\left(\langle\frac{\bar{\psi}_\alpha}{\psi[\alpha]}E_L[\alpha]\rangle - \langle\frac{\bar{\psi}_\alpha}{\psi[\alpha]}\rangle\langle E_L[\alpha]\rangle\right).$$

b) Find the corresponding expression for the variance.

Conjugate gradient (CG) method

The success of the CG method for finding solutions of non-linear problems is based on the theory of conjugate gradients for linear systems of equations. It belongs to the class of iterative methods for solving problems from linear algebra of the type

$$\hat{A}\hat{x} = \hat{b}.$$

In the iterative process we end up with a problem like

$$\hat{r} = \hat{b} - \hat{A}\hat{x},$$

where $\hat{r}$ is the so-called residual or error in the iterative process. When we have found the exact solution, $\hat{r} = 0$.

Conjugate gradient method

The residual is zero when we reach the minimum of the quadratic equation

$$P(\hat{x}) = \frac{1}{2}\hat{x}^T\hat{A}\hat{x} - \hat{x}^T\hat{b},$$

with the constraint that the matrix $\hat{A}$ is positive definite and symmetric. If we search for a minimum of the quantum mechanical variance, then the matrix $\hat{A}$, which is called the Hessian, is given by the second-derivative of the function we want to minimize. This quantity is always positive definite. In our case this corresponds normally to the second derivative of the energy.

Conjugate gradient method, Newton's method first

We seek the minimum of the energy or the variance as function of various variational parameters. In our case we have thus a function f whose minimum we are seeking. In Newton's method we set $\nabla f = 0$ and we can thus compute the next iteration point

$$\hat{x} - \hat{x}_i = \hat{A}^{-1}\nabla f(\hat{x}_i).$$

Subtracting this equation from that of $\hat{x}_{i+1}$ we have

$$\hat{x}_{i+1} - \hat{x}_i = \hat{A}^{-1}(\nabla f(\hat{x}_{i+1}) - \nabla f(\hat{x}_i)).$$

Simple example and demonstration

The function f can be either the energy or the variance. If we choose the energy then we have

$$\hat{\alpha}_{i+1} - \hat{\alpha}_i = \hat{A}^{-1}(\nabla E(\hat{\alpha}_{i+1}) - \nabla E(\hat{\alpha}_i)).$$

In the simple harmonic oscillator model, the gradient and the Hessian $\hat{A}$ are

$$\frac{d\langle E_L[\alpha]\rangle}{d\alpha} = \alpha - \frac{1}{4\alpha^3}$$

and a second derivative which is always positive (meaning that we find a minimum)

$$\hat{A} = \frac{d^2\langle E_L[\alpha]\rangle}{d\alpha^2} = 1 + \frac{3}{4\alpha^4}$$

Simple example and demonstration

We get then

$$\alpha_{i+1} = \frac{4}{3}\alpha_i - \frac{\alpha_i^4}{3\alpha_{i+1}^3},$$

which can be rewritten as

$$\alpha_{i+1}^4 - \frac{4}{3}\alpha_i\alpha_{i+1}^4 + \frac{1}{3}\alpha_i^4.$$

Using the conjugate gradient method

- Start your program with calling a function which implements for example the CG method or the steepest descent method.
- This function needs a function for the expectation value of the local energy and the derivative of the local energy.
- Your function **func** is now the Metropolis part with a call to the local energy function. For every call to the function **func** many practitionser use 1000-10000 Monte Carlo cycles for the trial wave function.
- This gives an expectation value for the energy which is returned by the function **func**.
- When one calls the local energy one also computes the first derivative of the expectaction value of the local energy

$$\frac{d\langle E_L[\alpha]\rangle}{d\alpha} = 2\langle\frac{\bar{\psi_{T\alpha}}}{\psi_T[\alpha]}(E_L[\alpha] - \langle E_L[\alpha]\rangle)\rangle.$$

The following two codes, to be found in the programs folder as well, implement first the conjugate gradient and steepest descent method to a simple 2×2 problem. Finally, we include a program based on

Simple codes for steepest descent and conjugate gradient using a 2×2 matrix

```
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>
#include "vectormatrixclass.h"
using namespace  std;
//   Main function begins here
int main(int  argc, char * argv[]){
  int dim = 2;
  Vector x(dim),xsd(dim), b(dim),x0(dim);
  Matrix A(dim,dim);

  // Set our initial guess
  x0(0) = x0(1) = 0;
  // Set the matrix
  A(0,0) =  3;    A(1,0) =  2;   A(0,1) =  2;   A(1,1) =  6;
  b(0) = 2; b(1) = -8;
  cout << "The Matrix A that we are using: " << endl;
  A.Print();
  cout << endl;
  x = ConjugateGradient(A,b,x0);
  xsd = SteepestDescent(A,b,x0);
  cout << "The approximate solution using Conjugate Gradient is: " <<
  x.Print();
  cout << endl;
  cout << "The approximate solution using Steepest Descent is: " << en
```

The routine for the steepest descent method

```
Vector SteepestDescent(Matrix A, Vector b, Vector x0){
  int IterMax, i;
  int dim = x0.Dimension();
  const double tolerance = 1.0e-14;
  Vector x(dim),f(dim),z(dim);
  double c,alpha,d;
  IterMax = 30;
  x = x0;
  f = A*x-b;
  i = 0;
  while (i <= IterMax){
    z = A*f;
    c = dot(f,f);
    alpha = c/dot(f,z);
    x = x - alpha*f;
    f =  A*x-b;
    if(sqrt(dot(f,f)) < tolerance) break;
    i++;
  }
  return x;
}
```

Conjugate gradient method

In the CG method we define so-called conjugate directions and two vectors $\hat{s}$ and $\hat{t}$ are said to be conjugate if

$$\hat{s}^T \hat{A} \hat{t} = 0.$$

The philosophy of the CG method is to perform searches in various conjugate directions of our vectors $\hat{x}_i$ obeying the above criterion, namely

$$\hat{x}_i^T \hat{A} \hat{x}_j = 0.$$

Two vectors are conjugate if they are orthogonal with respect to this inner product. Being conjugate is a symmetric relation: if $\hat{s}$ is conjugate to $\hat{t}$, then $\hat{t}$ is conjugate to $\hat{s}$.

Conjugate gradient method

An example is given by the eigenvectors of the matrix

$$\hat{v}_i^T \hat{A} \hat{v}_j = \lambda \hat{v}_i^T \hat{v}_j,$$

which is zero unless $i = j$.

Conjugate gradient method

Assume now that we have a symmetric positive-definite matrix $\hat{A}$ of size $n \times n$. At each iteration $i + 1$ we obtain the conjugate direction of a vector

$$\hat{x}_{i+1} = \hat{x}_i + \alpha_i \hat{p}_i.$$

We assume that $\hat{p}_i$ is a sequence of n mutually conjugate directions. Then the $\hat{p}_i$ form a basis of R^n and we can expand the solution $\hat{A}\hat{x} = \hat{b}$ in this basis, namely

$$\hat{x} = \sum_{i=1}^{n} \alpha_i \hat{p}_i.$$

Conjugate gradient method

The coefficients are given by

$$\mathbf{A}\mathbf{x} = \sum_{i=1}^{n} \alpha_i \mathbf{A}\mathbf{p}_i = \mathbf{b}.$$

Multiplying with $\hat{p}_k^T$ from the left gives

$$\hat{p}_k^T \hat{A} \hat{x} = \sum_{i=1}^{n} \alpha_i \hat{p}_k^T \hat{A} \hat{p}_i = \hat{p}_k^T \hat{b},$$

and we can define the coefficients α_k as

$$\alpha_k = \frac{\hat{p}_k^T \hat{b}}{\hat{p}_k^T \hat{A} \hat{p}_k}$$

Conjugate gradient method and iterations

If we choose the conjugate vectors $\hat{p}_k$ carefully, then we may not need all of them to obtain a good approximation to the solution $\hat{x}$. We want to regard the conjugate gradient method as an iterative method. This will us to solve systems where n is so large that the direct method would take too much time.
We denote the initial guess for $\hat{x}$ as $\hat{x}_0$. We can assume without loss of generality that

$$\hat{x}_0 = 0,$$

or consider the system

$$\hat{A}\hat{z} = \hat{b} - \hat{A}\hat{x}_0,$$

instead.

Conjugate gradient method

One can show that the solution $\hat{x}$ is also the unique minimizer of the quadratic form

$$f(\hat{x}) = \frac{1}{2}\hat{x}^T \hat{A} \hat{x} - \hat{x}^T \hat{x}, \quad \hat{x} \in \mathbf{R}^n.$$

This suggests taking the first basis vector $\hat{p}_1$ to be the gradient of f at $\hat{x} = \hat{x}_0$, which equals

$$\hat{A}\hat{x}_0 - \hat{b},$$

and $\hat{x}_0 = 0$ it is equal $-\hat{b}$. The other vectors in the basis will be conjugate to the gradient, hence the name conjugate gradient method.

Conjugate gradient method

Let $\hat{r}_k$ be the residual at the k-th step:

$$\hat{r}_k = \hat{b} - \hat{A}\hat{x}_k.$$

Note that $\hat{r}_k$ is the negative gradient of f at $\hat{x} = \hat{x}_k$, so the gradient descent method would be to move in the direction $\hat{r}_k$. Here, we insist that the directions $\hat{p}_k$ are conjugate to each other, so we take the direction closest to the gradient $\hat{r}_k$ under the conjugacy constraint. This gives the following expression

$$\hat{p}_{k+1} = \hat{r}_k - \frac{\hat{p}_k^T \hat{A} \hat{r}_k}{\hat{p}_k^T \hat{A} \hat{p}_k} \hat{p}_k.$$

Conjugate gradient method

We can also compute the residual iteratively as

$$\hat{r}_{k+1} = \hat{b} - \hat{A}\hat{x}_{k+1},$$

which equals

$$\hat{b} - \hat{A}(\hat{x}_k + \alpha_k \hat{p}_k),$$

or

$$(\hat{b} - \hat{A}\hat{x}_k) - \alpha_k \hat{A}\hat{p}_k,$$

which gives

$$\hat{r}_{k+1} = \hat{r}_k - \hat{A}\hat{p}_k,$$

Simple implementation of the Conjugate gradient algorithm

```
Vector ConjugateGradient(Matrix A, Vector b, Vector x0){
int dim = x0.Dimension();
const double tolerance = 1.0e-14;
Vector x(dim),r(dim),v(dim),z(dim);
double c,t,d;

x = x0;
r = b - A*x;
v = r;
c = dot(r,r);
int i = 0; IterMax = dim;
while(i <= IterMax){
  z = A*v;
  t = c/dot(v,z);
  x = x + t*v;
  r = r - t*z;
  d = dot(r,r);
  if(sqrt(d) < tolerance)
    break;
  v = r + (d/c)*v;
  c = d;  i++;
}
return x;
}
```

Broyden–Fletcher–Goldfarb–Shanno algorithm

The optimization problem is to minimize $f(\mathbf{x})$ where $\mathbf{x}$ is a vector in R^n, and f is a differentiable scalar function. There are no constraints on the values that $\mathbf{x}$ can take.
The algorithm begins at an initial estimate for the optimal value $\mathbf{x}_0$ and proceeds iteratively to get a better estimate at each stage.
The search direction p_k at stage k is given by the solution of the analogue of the Newton equation

$$B_k \mathbf{p}_k = -\nabla f(\mathbf{x}_k),$$

where B_k is an approximation to the Hessian matrix, which is updated iteratively at each stage, and $\nabla f(\mathbf{x}_k)$ is the gradient of the function evaluated at x_k. A line search in the direction p_k is then used to find the next point x_{k+1} by minimising

$$f(\mathbf{x}_k + \alpha \mathbf{p}_k),$$

over the scalar $\alpha > 0$.

Codes from numerical recipes

You can however use codes we have adapted from the text Numerical Recipes in C++, see chapter 10.7. Here we present a program, which you also can find at the webpage of the course we use the functions **dfpmin** and **lnsrch**. This is a variant of the Broyden et al algorithm discussed in the previous slide.

- The program uses the harmonic oscillator in one dimensions as example.
- The program does not use armadillo to handle vectors and matrices, but employs rather my own vector-matrix class. These auxiliary functions, and the main program *model.cpp* can all be found under the program link here.

Below we show only excerpts from the main program. For the full program, see the above link.

Finding the minimum of the harmonic oscillator model in one dimension

```
//   Main function begins here
int main()
{
     int n, iter;
     double gtol, fret;
     double alpha;
     n = 1;
//   reserve space in memory for vectors containing the variational
//   parameters
     Vector g(n), p(n);
     cout << "Read in guess for alpha" << endl;
     cin >> alpha;
     gtol = 1.0e-5;
//   now call dfmin and compute the minimum
     p(0) = alpha;
     dfpmin(p, n, gtol, &iter, &fret, Efunction, dEfunction);
     cout << "Value of energy minimum = " << fret << endl;
     cout << "Number of iterations = " << iter << endl;
     cout << "Value of alpha at minimum = " << p(0) << endl;
      return 0;
}  // end of main program
```

Functions to observe

The functions **Efunction** and **dEfunction** compute the expectation value of the energy and its derivative. These functions need to be changed when you want to your own derivatives.

```
//  this function defines the expectation value of the local energy
double Efunction(Vector  &x)
{
  double value = x(0)*x(0)*0.5+1.0/(8*x(0)*x(0));
  return value;
} // end of function to evaluate

//  this function defines the derivative of the energy
void dEfunction(Vector &x, Vector &g)
{
  g(0) = x(0)-1.0/(4*x(0)*x(0)*x(0));
} // end of function to evaluate
```

You need to change these functions in order to compute the local energy for your system. I used 1000 cycles per call to get a new value of $\langle E_L[\alpha]\rangle$. When I compute the local energy I also compute its derivative. After roughly 10-20 iterations I got a converged result in terms of α.