Week 11, March 11-15: Resampling Techniques, Bootstrap and Blocking

Morten Hjorth-Jensen Email morten.hjorth-jensen@fys.uio.no[1,2]

[1]Department of Physics and Center fo Computing in Science Education, University of Oslo, Oslo, Norway
[2]Department of Physics and Astronomy and Facility for Rare Ion Beams, Michigan State University, East Lansing, Michigan, USA

March 11-15

Overview of week 11, March 11-15

Topics.

1. Reminder from last week about statistical observables, the central limit theorem and bootstrapping, see notes from last week
2. Resampling TechniquesL Blocking
3. Discussion of onebody densities
4. Start discussion on optimization and parallelization

Teaching Material, videos and written material.

- Overview video on the Bootstrap method
- Marius Johnson's Master thesis on the Blocking Method

Why resampling methods ?

Statistical analysis.

- Our simulations can be treated as *computer experiments*. This is particularly the case for Monte Carlo methods
- The results can be analysed with the same statistical tools as we would use analysing experimental data.
- As in all experiments, we are looking for expectation values and an estimate of how accurate they are, i.e., possible sources for errors.

Statistical analysis

- As in other experiments, many numerical experiments have two classes of errors:
  1. Statistical errors
  2. Systematical errors
- Statistical errors can be estimated using standard tools from statistics
- Systematical errors are method specific and must be treated differently from case to case.

And why do we use such methods?

As you will see below, due to correlations between various measurements, we need to evaluate the so-called covariance in order to establish a proper evaluation of the total variance and the thereby the standard deviation of a given expectation value.

The covariance however, leads to an evaluation of a double sum over the various stochastic variables. This becomes computationally too expensive to evaluate. Methods like the Bootstrap, the Jackknife and/or Blocking allow us to circumvent this problem.

Central limit theorem

Last week we derived the central limit theorem with the following assumptions:

Measurement i. We assumed that each individual measurement x_{ij} is represented by stochastic variables which independent and identically distributed (iid). This defined the sample mean of of experiment i with n samples as

$$\overline{x}_i = \frac{1}{n}\sum_j x_{ij}.$$

and the sample variance

$$\sigma_i^2 = \frac{1}{n}\sum_j \left(x_{ij} - \overline{x}_i\right)^2.$$

Note that we use n instead of $n-1$ in the definition of variance. The sample variance and mean are not necessarily equal to the exact values we would get if we knew the corresponding probability distribution.

Running many measurements

Adding m measurements i. With the assumption that the average measurements i are also defined as iid stochastic variables and have the same probability function p, we defined the total average over m experiments as

$$\overline{X} = \frac{1}{m}\sum_i \overline{x}_i.$$

and the total variance

$$\sigma_m^2 = \frac{1}{m}\sum_i \left(\overline{x}_i - \overline{X}\right)^2.$$

These are the quantities we used in showing that if the individual mean values are iid stochastic variables, then in the limit $m \to \infty$, the distribution for $\overline{X}$ is given by a Gaussian distribution with variance σ_m^2.

Adding more definitions

The total sample variance over the mn measurements is defined as

$$\sigma^2 = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(x_{ij} - \overline{X}\right)^2.$$

We have from the equation for σ_m^2

$$\overline{x}_i - \overline{X} = \frac{1}{n}\sum_{j=1}^{n}\left(x_i - \overline{X}\right),$$

and introducing the centered value $\tilde{x}_{ij} = x_{ij} - \overline{X}$, we can rewrite σ_m^2 as

$$\sigma_m^2 = \frac{1}{m}\sum_i \left(\overline{x}_i - \overline{X}\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\left[\frac{i}{n}\sum_{j=1}^{n}\tilde{x}_{ij}\right]^2.$$

Further rewriting

We can rewrite the latter in terms of a sum over diagonal elements only and another sum which contains the non-diagonal elements

$$\begin{aligned}\sigma_m^2 &= \frac{1}{m}\sum_{i=1}^{m}\left[\frac{i}{n}\sum_{j=1}^{n}\tilde{x}_{ij}\right]^2 \\ &= \frac{1}{mn^2}\sum_{i=1}^{m}\sum_{j=1}^{n}\tilde{x}_{ij}^2 + \frac{2}{mn^2}\sum_{i=1}^{m}\sum_{j<k}^{n}\tilde{x}_{ij}\tilde{x}_{ik}.\end{aligned}$$

The first term on the last rhs is nothing but the total sample variance σ^2 divided by m. The second term represents the covariance.

The covariance term

Using the definition of the total sample variance we have

$$\sigma_m^2 = \frac{\sigma^2}{m} + \frac{2}{mn^2}\sum_{i=1}^{m}\sum_{j<k}^{n}\tilde{x}_{ij}\tilde{x}_{ik}.$$

The first term is what we have used till now in order to estimate the standard deviation. However, the second term which gives us a measure of the correlations between different stochastic events, can result in contributions which give rise to a larger standard deviation and variance σ_m^2. Note also the evaluation of the second term leads to a double sum over all events. If we run a VMC calculation with say 10^9 Monte carlo samples, the latter term would lead to 10^{18} function evaluations. We don't want to, by obvious reasons, to venture into that many evaluations.

Note also that if our stochastic events are iid then the covariance terms is zero.

Rewriting the covariance term

We introduce now a variable $d = |j - k|$ and rewrite

$$\frac{2}{mn^2}\sum_{i=1}^{m}\sum_{j<k}^{n}\tilde{x}_{ij}\tilde{x}_{ik},$$

in terms of a function

$$f_d = \frac{2}{mn}\sum_{i=1}^{m}\sum_{k=1}^{n-d}\tilde{x}_{ik}\tilde{x}_{i(k+d)}.$$

We note that for $d =$ we have

$$f_0 = \frac{2}{mn}\sum_{i=1}^{m}\sum_{k=1}^{n}\tilde{x}_{ik}\tilde{x}_{i(k)} = \sigma^2!$$

Introducing the correlation function

We introduce then a correlation function $\kappa_d = f_d/\sigma^2$. Note that $\kappa_0 = 1$. We rewrite the variance σ_m^2 as

$$\sigma_m^2 = \frac{\sigma^2}{m}\left[1 + 2\sum_{d=1}^{n-1}\kappa_d\right].$$

The code here shows the evolution of κ_d as a function of d for a series of random numbers. We see that the function κ_d approaches 0 as $d \to \infty$.

Statistics, wrapping up from last week

Let us analyze the problem by splitting up the correlation term into partial sums of the form:

$$f_d = \frac{1}{n-d}\sum_{k=1}^{n-d}(x_k - \bar{x}_n)(x_{k+d} - \bar{x}_n)$$

The correlation term of the error can now be rewritten in terms of f_d

$$\frac{2}{n}\sum_{k<l}(x_k - \bar{x}_n)(x_l - \bar{x}_n) = 2\sum_{d=1}^{n-1} f_d$$

The value of f_d reflects the correlation between measurements separated by the distance d in the sample samples. Notice that for $d = 0$, f is just the sample variance, $\mathrm{var}(x)$. If we divide f_d by $\mathrm{var}(x)$, we arrive at the so called *autocorrelation function*

$$\kappa_d = \frac{f_d}{\mathrm{var}(x)}$$

which gives us a useful measure of pairwise correlations starting always at 1 for $d = 0$.

Statistics, final expression

The sample error can now be written in terms of the autocorrelation function:

$$\begin{aligned}
\mathrm{err}_X^2 &= \frac{1}{n}\mathrm{var}(x) + \frac{2}{n}\cdot\mathrm{var}(x)\sum_{d=1}^{n-1}\frac{f_d}{\mathrm{var}(x)} \\
&= \left(1 + 2\sum_{d=1}^{n-1}\kappa_d\right)\frac{1}{n}\mathrm{var}(x) \\
&= \frac{\tau}{n}\cdot\mathrm{var}(x) \qquad (1)
\end{aligned}$$

and we see that err_X can be expressed in terms the uncorrelated sample variance times a correction factor τ which accounts for the correlation between measurements. We call this correction factor the *autocorrelation time*:

$$\tau = 1 + 2\sum_{d=1}^{n-1}\kappa_d \qquad (2)$$

Statistics, effective number of correlations

For a correlation free experiment, τ equals 1.

We can interpret a sequential correlation as an effective reduction of the number of measurements by a factor τ. The effective number of measurements becomes:

$$n_{\text{eff}} = \frac{n}{\tau}$$

To neglect the autocorrelation time τ will always cause our simple uncorrelated estimate of $\text{err}_X^2 \approx \text{var}(x)/n$ to be less than the true sample error. The estimate of the error will be too *good*. On the other hand, the calculation of the full autocorrelation time poses an efficiency problem if the set of measurements is very large.

Can we understand this? Time Auto-correlation Function

The so-called time-displacement autocorrelation $\phi(t)$ for a quantity $\mathbf{M}$ is given by

$$\phi(t) = \int dt' \left[\mathbf{M}(t') - \langle \mathbf{M} \rangle\right] \left[\mathbf{M}(t'+t) - \langle \mathbf{M} \rangle\right],$$

which can be rewritten as

$$\phi(t) = \int dt' \left[\mathbf{M}(t')\mathbf{M}(t'+t) - \langle \mathbf{M} \rangle^2\right],$$

where $\langle \mathbf{M} \rangle$ is the average value and $\mathbf{M}(t)$ its instantaneous value. We can discretize this function as follows, where we used our set of computed values $\mathbf{M}(t)$ for a set of discretized times (our Monte Carlo cycles corresponding to moving all electrons?)

$$\phi(t) = \frac{1}{t_{\text{max}} - t} \sum_{t'=0}^{t_{\text{max}}-t} \mathbf{M}(t')\mathbf{M}(t'+t) - \frac{1}{t_{\text{max}} - t} \sum_{t'=0}^{t_{\text{max}}-t} \mathbf{M}(t') \times \frac{1}{t_{\text{max}} - t} \sum_{t'=0}^{t_{\text{max}}-t} \mathbf{M}(t'+t).$$

Time Auto-correlation Function

One should be careful with times close to t_{max}, the upper limit of the sums becomes small and we end up integrating over a rather small time interval. This means that the statistical error in $\phi(t)$ due to the random nature of the fluctuations in $\mathbf{M}(t)$ can become large.

One should therefore choose $t \ll t_{\text{max}}$.

Note that the variable $\mathbf{M}$ can be any expectation values of interest.

The time-correlation function gives a measure of the correlation between the various values of the variable at a time t' and a time $t'+t$. If we multiply the values of $\mathbf{M}$ at these two different times, we will get a positive contribution if they are fluctuating in the same direction, or a negative value if they fluctuate in the opposite direction. If we then integrate over time, or use the discretized version of, the time correlation function $\phi(t)$ should take a non-zero value if the fluctuations are correlated, else it should gradually go to zero. For times a long way apart the different values of $\mathbf{M}$ are most likely uncorrelated and $\phi(t)$ should be zero.

Time Auto-correlation Function

We can derive the correlation time by observing that our Metropolis algorithm is based on a random walk in the space of all possible spin configurations. Our probability distribution function $\hat{\mathbf{w}}(t)$ after a given number of time steps t could be written as

$$\hat{\mathbf{w}}(t) = \hat{\mathbf{W}}^{\mathbf{t}}\hat{\mathbf{w}}(0),$$

with $\hat{\mathbf{w}}(0)$ the distribution at $t = 0$ and $\hat{\mathbf{W}}$ representing the transition probability matrix. We can always expand $\hat{\mathbf{w}}(0)$ in terms of the right eigenvectors of $\hat{\mathbf{v}}$ of $\hat{\mathbf{W}}$ as

$$\hat{\mathbf{w}}(0) = \sum_i \alpha_i \hat{\mathbf{v}}_i,$$

resulting in

$$\hat{\mathbf{w}}(t) = \hat{\mathbf{W}}^t\hat{\mathbf{w}}(0) = \hat{\mathbf{W}}^t \sum_i \alpha_i \hat{\mathbf{v}}_i = \sum_i \lambda_i^t \alpha_i \hat{\mathbf{v}}_i,$$

with λ_i the i^{th} eigenvalue corresponding to the eigenvector $\hat{\mathbf{v}}_i$.

Time Auto-correlation Function

If we assume that λ_0 is the largest eigenvector we see that in the limit $t \to \infty$, $\hat{\mathbf{w}}(t)$ becomes proportional to the corresponding eigenvector $\hat{\mathbf{v}}_0$. This is our steady state or final distribution.

We can relate this property to an observable like the mean energy. With the probabilty $\hat{\mathbf{w}}(t)$ (which in our case is the squared trial wave function) we can write the expectation values as

$$\langle \mathbf{M}(t) \rangle = \sum_\mu \hat{\mathbf{w}}(t)_\mu \mathbf{M}_\mu,$$

or as the scalar of a vector product

$$\langle \mathbf{M}(t) \rangle = \hat{\mathbf{w}}(t)\mathbf{m},$$

with $\mathbf{m}$ being the vector whose elements are the values of $\mathbf{M}_\mu$ in its various microstates μ.

Time Auto-correlation Function

We rewrite this relation as

$$\langle \mathbf{M}(t) \rangle = \hat{\mathbf{w}}(t)\mathbf{m} = \sum_i \lambda_i^t \alpha_i \hat{\mathbf{v}}_i \mathbf{m}_i.$$

If we define $m_i = \hat{\mathbf{v}}_i \mathbf{m}_i$ as the expectation value of $\mathbf{M}$ in the i^{th} eigenstate we can rewrite the last equation as

$$\langle \mathbf{M}(t) \rangle = \sum_i \lambda_i^t \alpha_i m_i.$$

Since we have that in the limit $t \to \infty$ the mean value is dominated by the the largest eigenvalue λ_0, we can rewrite the last equation as

$$\langle \mathbf{M}(t) \rangle = \langle \mathbf{M}(\infty) \rangle + \sum_{i \neq 0} \lambda_i^t \alpha_i m_i.$$

We define the quantity

$$\tau_i = -\frac{1}{log \lambda_i},$$

and rewrite the last expectation value as

$$\langle \mathbf{M}(t) \rangle = \langle \mathbf{M}(\infty) \rangle + \sum_{i \neq 0} \alpha_i m_i e^{-t/\tau_i}.$$

Time Auto-correlation Function

The quantities τ_i are the correlation times for the system. They control also the auto-correlation function discussed above. The longest correlation time is obviously given by the second largest eigenvalue τ_1, which normally defines the correlation time discussed above. For large times, this is the only correlation time that survives. If higher eigenvalues of the transition matrix are well separated from λ_1 and we simulate long enough, τ_1 may well define the correlation time. In other cases we may not be able to extract a reliable result for τ_1. Coming back to the time correlation function $\phi(t)$ we can present a more general definition in terms of the mean magnetizations $\langle \mathbf{M}(t) \rangle$. Recalling that the mean value is equal to $\langle \mathbf{M}(\infty) \rangle$ we arrive at the expectation values

$$\phi(t) = \langle \mathbf{M}(0) - \mathbf{M}(\infty) \rangle \langle \mathbf{M}(t) - \mathbf{M}(\infty) \rangle,$$

resulting in

$$\phi(t) = \sum_{i,j \neq 0} m_i \alpha_i m_j \alpha_j e^{-t/\tau_i},$$

which is appropriate for all times.

Correlation Time

If the correlation function decays exponentially

$$\phi(t) \sim \exp(-t/\tau)$$

then the exponential correlation time can be computed as the average

$$\tau_{\exp} = -\langle \frac{t}{log|\frac{\phi(t)}{\phi(0)}|} \rangle.$$

If the decay is exponential, then

$$\int_0^\infty dt \phi(t) = \int_0^\infty dt \phi(0) \exp(-t/\tau) = \tau \phi(0),$$

which suggests another measure of correlation

$$\tau_{\text{int}} = \sum_k \frac{\phi(k)}{\phi(0)},$$

called the integrated correlation time.

Resampling methods: Blocking

The blocking method was made popular by Flyvbjerg and Pedersen (1989) and has become one of the standard ways to estimate $V(\widehat{\theta})$ for exactly one $\widehat{\theta}$, namely $\widehat{\theta} = \overline{X}$.

Assume $n = 2^d$ for some integer $d > 1$ and $X_1, X_2, \cdots, X_n$ is a stationary time series to begin with. Moreover, assume that the time series is asymptotically uncorrelated. We switch to vector notation by arranging $X_1, X_2, \cdots, X_n$ in an n-tuple. Define:

$$\hat{X} = (X_1, X_2, \cdots, X_n).$$

The strength of the blocking method is when the number of observations, n is large. For large n, the complexity of dependent bootstrapping scales poorly, but the blocking method does not, moreover, it becomes more accurate the larger n is.

Blocking Transformations

We now define blocking transformations. The idea is to take the mean of subsequent pair of elements from $\vec{X}$ and form a new vector $\vec{X}_1$. Continuing in the same way by taking the mean of subsequent pairs of elements of $\vec{X}_1$ we obtain $\vec{X}_2$, and so on. Define $\vec{X}_i$ recursively by:

$$\begin{aligned} (\vec{X}_0)_k &\equiv (\vec{X})_k \\ (\vec{X}_{i+1})_k &\equiv \frac{1}{2}\Big((\vec{X}_i)_{2k-1} + (\vec{X}_i)_{2k}\Big) \qquad \text{for all} \qquad 1 \leq i \leq d-1 \end{aligned} \tag{3}$$

The quantity $\vec{X}_k$ is subject to k **blocking transformations**. We now have d vectors $\vec{X}_0, \vec{X}_1, \cdots, \vec{X}_{d-1}$ containing the subsequent averages of observations. It turns out that if the components of $\vec{X}$ is a stationary time series, then the components of $\vec{X}_i$ is a stationary time series for all $0 \leq i \leq d-1$

We can then compute the autocovariance, the variance, sample mean, and number of observations for each i. Let $\gamma_i, \sigma_i^2, \overline{X}_i$ denote the autocovariance, variance and average of the elements of $\vec{X}_i$ and let n_i be the number of elements of $\vec{X}_i$. It follows by induction that $n_i = n/2^i$.

Blocking Transformations

Using the definition of the blocking transformation and the distributive property of the covariance, it is clear that since $h = |i - j|$ we can define

$$\begin{aligned}\gamma_{k+1}(h) &= cov\left((X_{k+1})_i, (X_{k+1})_j\right) \\ &= \frac{1}{4} cov\left((X_k)_{2i-1} + (X_k)_{2i}, (X_k)_{2j-1} + (X_k)_{2j}\right) \\ &= \frac{1}{2}\gamma_k(2h) + \frac{1}{2}\gamma_k(2h+1) \text{ h} = 0 \qquad (4) \\ &= \frac{1}{4}\gamma_k(2h-1) + \frac{1}{2}\gamma_k(2h) + \frac{1}{4}\gamma_k(2h+1) \quad \text{else} \qquad (5)\end{aligned}$$

The quantity $\hat{X}$ is asymptotic uncorrelated by assumption, $\hat{X}_k$ is also asymptotic uncorrelated. Let's turn our attention to the variance of the sample mean $V(\overline{X})$.

Blocking Transformations, getting there

We have

$$V(\overline{X}_k) = \frac{\sigma_k^2}{n_k} + \underbrace{\frac{2}{n_k}\sum_{h=1}^{n_k-1}\left(1 - \frac{h}{n_k}\right)\gamma_k(h)}_{\equiv e_k} = \frac{\sigma_k^2}{n_k} + e_k \quad \text{if} \quad \gamma_k(0) = \sigma_k^2. \qquad (6)$$

The term e_k is called the **truncation error**:

$$e_k = \frac{2}{n_k}\sum_{h=1}^{n_k-1}\left(1 - \frac{h}{n_k}\right)\gamma_k(h). \qquad (7)$$

We can show that $V(\overline{X}_i) = V(\overline{X}_j)$ for all $0 \leq i \leq d-1$ and $0 \leq j \leq d-1$.

Blocking Transformations, final expressions

We can then wrap up

$$\begin{aligned}n_{j+1}\overline{X}_{j+1} &= \sum_{i=1}^{n_{j+1}}(\hat{X}_{j+1})_i = \frac{1}{2}\sum_{i=1}^{n_j/2}(\hat{X}_j)_{2i-1} + (\hat{X}_j)_{2i} \\ &= \frac{1}{2}\left[(\hat{X}_j)_1 + (\hat{X}_j)_2 + \cdots + (\hat{X}_j)_{n_j}\right] = \underbrace{\frac{n_j}{2}}_{=n_{j+1}}\overline{X}_j = n_{j+1}\overline{X}_j. \qquad (8)\end{aligned}$$

By repeated use of this equation we get $V(\overline{X}_i) = V(\overline{X}_0) = V(\overline{X})$ for all $0 \leq i \leq d-1$. This has the consequence that

$$V(\overline{X}) = \frac{\sigma_k^2}{n_k} + e_k \qquad \text{for all} \qquad 0 \leq k \leq d-1. \qquad (9)$$

Flyvbjerg and Petersen demonstrated that the sequence $\{e_k\}_{k=0}^{d-1}$ is decreasing, and conjecture that the term e_k can be made as small as we would like by making k (and hence d) sufficiently large. The sequence is decreasing (Master of Science thesis by Marius Jonsson, UiO 2018). It means we can apply blocking transformations until e_k is sufficiently small, and then estimate $V(\overline{X})$ by $\widehat{\sigma}_k^2/n_k$.

For an elegant solution and proof of the blocking method, see the recent article of Marius Jonsson (former MSc student of the Computational Physics group).

Example code form last week

```
# 2-electron VMC code for 2dim quantum dot with importance sampling
# Using gaussian rng for new positions and Metropolis- Hastings
# Added energy minimization
from math import exp, sqrt
from random import random, seed, normalvariate
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from scipy.optimize import minimize
import sys
import os

# Where to save data files
PROJECT_ROOT_DIR = "Results"
DATA_ID = "Results/EnergyMin"

if not os.path.exists(PROJECT_ROOT_DIR):
    os.mkdir(PROJECT_ROOT_DIR)

if not os.path.exists(DATA_ID):
    os.makedirs(DATA_ID)

def data_path(dat_id):
    return os.path.join(DATA_ID, dat_id)

outfile = open(data_path("Energies.dat"),'w')


# Trial wave function for the 2-electron quantum dot in two dims
def WaveFunction(r,alpha,beta):
    r1 = r[0,0]**2 + r[0,1]**2
    r2 = r[1,0]**2 + r[1,1]**2
    r12 = sqrt((r[0,0]-r[1,0])**2 + (r[0,1]-r[1,1])**2)
    deno = r12/(1+beta*r12)
    return exp(-0.5*alpha*(r1+r2)+deno)

# Local energy  for the 2-electron quantum dot in two dims, using analytical local energy
def LocalEnergy(r,alpha,beta):

    r1 = (r[0,0]**2 + r[0,1]**2)
    r2 = (r[1,0]**2 + r[1,1]**2)
    r12 = sqrt((r[0,0]-r[1,0])**2 + (r[0,1]-r[1,1])**2)
    deno = 1.0/(1+beta*r12)
    deno2 = deno*deno
```

```
    return 0.5*(1-alpha*alpha)*(r1 + r2) +2.0*alpha + 1.0/r12+deno2*(alpha*r12-deno2+2*beta*deno-1

# Derivate of wave function ansatz as function of variational parameters
def DerivativeWFansatz(r,alpha,beta):

    WfDer  = np.zeros((2), np.double)
    r1 = (r[0,0]**2 + r[0,1]**2)
    r2 = (r[1,0]**2 + r[1,1]**2)
    r12 = sqrt((r[0,0]-r[1,0])**2 + (r[0,1]-r[1,1])**2)
    deno = 1.0/(1+beta*r12)
    deno2 = deno*deno
    WfDer[0] = -0.5*(r1+r2)
    WfDer[1] = -r12*r12*deno2
    return  WfDer

# Setting up the quantum force for the two-electron quantum dot, recall that it is a vector
def QuantumForce(r,alpha,beta):

    qforce = np.zeros((NumberParticles,Dimension), np.double)
    r12 = sqrt((r[0,0]-r[1,0])**2 + (r[0,1]-r[1,1])**2)
    deno = 1.0/(1+beta*r12)
    qforce[0,:] = -2*r[0,:]*alpha*(r[0,:]-r[1,:])*deno*deno/r12
    qforce[1,:] = -2*r[1,:]*alpha*(r[1,:]-r[0,:])*deno*deno/r12
    return qforce


# Computing the derivative of the energy and the energy
def EnergyDerivative(x0):


    # Parameters in the Fokker-Planck simulation of the quantum force
    D = 0.5
    TimeStep = 0.05
    # positions
    PositionOld = np.zeros((NumberParticles,Dimension), np.double)
    PositionNew = np.zeros((NumberParticles,Dimension), np.double)
    # Quantum force
    QuantumForceOld = np.zeros((NumberParticles,Dimension), np.double)
    QuantumForceNew = np.zeros((NumberParticles,Dimension), np.double)

    energy = 0.0
    DeltaE = 0.0
    alpha = x0[0]
    beta = x0[1]
    EnergyDer = 0.0
    DeltaPsi = 0.0
    DerivativePsiE = 0.0
    #Initial position
    for i in range(NumberParticles):
        for j in range(Dimension):
            PositionOld[i,j] = normalvariate(0.0,1.0)*sqrt(TimeStep)
    wfold = WaveFunction(PositionOld,alpha,beta)
    QuantumForceOld = QuantumForce(PositionOld,alpha, beta)

    #Loop over MC MCcycles
    for MCcycle in range(NumberMCcycles):
        #Trial position moving one particle at the time
        for i in range(NumberParticles):
            for j in range(Dimension):
                PositionNew[i,j] = PositionOld[i,j]+normalvariate(0.0,1.0)*sqrt(TimeStep)+\
                                   QuantumForceOld[i,j]*TimeStep*D
```

```
            wfnew = WaveFunction(PositionNew,alpha,beta)
            QuantumForceNew = QuantumForce(PositionNew,alpha, beta)
            GreensFunction = 0.0
            for j in range(Dimension):
                GreensFunction += 0.5*(QuantumForceOld[i,j]+QuantumForceNew[i,j])*\
                                  (D*TimeStep*0.5*(QuantumForceOld[i,j]-QuantumForceNew[i,j])-
                                  PositionNew[i,j]+PositionOld[i,j])

            GreensFunction = exp(GreensFunction)
            ProbabilityRatio = GreensFunction*wfnew**2/wfold**2
            #Metropolis-Hastings test to see whether we accept the move
            if random() <= ProbabilityRatio:
                for j in range(Dimension):
                    PositionOld[i,j] = PositionNew[i,j]
                    QuantumForceOld[i,j] = QuantumForceNew[i,j]
                wfold = wfnew
        DeltaE = LocalEnergy(PositionOld,alpha,beta)
        DerPsi = DerivativeWFansatz(PositionOld,alpha,beta)
        DeltaPsi += DerPsi
        energy += DeltaE
        DerivativePsiE += DerPsi*DeltaE

    # We calculate mean values
    energy /= NumberMCcycles
    DerivativePsiE /= NumberMCcycles
    DeltaPsi /= NumberMCcycles
    EnergyDer  = 2*(DerivativePsiE-DeltaPsi*energy)
    return EnergyDer


# Computing the expectation value of the local energy
def Energy(x0):
    # Parameters in the Fokker-Planck simulation of the quantum force
    D = 0.5
    TimeStep = 0.05
    # positions
    PositionOld = np.zeros((NumberParticles,Dimension), np.double)
    PositionNew = np.zeros((NumberParticles,Dimension), np.double)
    # Quantum force
    QuantumForceOld = np.zeros((NumberParticles,Dimension), np.double)
    QuantumForceNew = np.zeros((NumberParticles,Dimension), np.double)

    energy = 0.0
    DeltaE = 0.0
    alpha = x0[0]
    beta = x0[1]
    #Initial position
    for i in range(NumberParticles):
        for j in range(Dimension):
            PositionOld[i,j] = normalvariate(0.0,1.0)*sqrt(TimeStep)
    wfold = WaveFunction(PositionOld,alpha,beta)
    QuantumForceOld = QuantumForce(PositionOld,alpha, beta)

    #Loop over MC MCcycles
    for MCcycle in range(NumberMCcycles):
        #Trial position moving one particle at the time
        for i in range(NumberParticles):
            for j in range(Dimension):
                PositionNew[i,j] = PositionOld[i,j]+normalvariate(0.0,1.0)*sqrt(TimeStep)+\
                                   QuantumForceOld[i,j]*TimeStep*D
            wfnew = WaveFunction(PositionNew,alpha,beta)
```

```
                QuantumForceNew = QuantumForce(PositionNew,alpha, beta)
                GreensFunction = 0.0
                for j in range(Dimension):
                    GreensFunction += 0.5*(QuantumForceOld[i,j]+QuantumForceNew[i,j])*\
                                      (D*TimeStep*0.5*(QuantumForceOld[i,j]-QuantumForceNew[i,j])-
                                      PositionNew[i,j]+PositionOld[i,j])

                GreensFunction = exp(GreensFunction)
                ProbabilityRatio = GreensFunction*wfnew**2/wfold**2
                #Metropolis-Hastings test to see whether we accept the move
                if random() <= ProbabilityRatio:
                    for j in range(Dimension):
                        PositionOld[i,j] = PositionNew[i,j]
                        QuantumForceOld[i,j] = QuantumForceNew[i,j]
                    wfold = wfnew
            DeltaE = LocalEnergy(PositionOld,alpha,beta)
            energy += DeltaE
            if Printout:
               outfile.write('%f\n' %(energy/(MCcycle+1.0)))
        # We calculate mean values
        energy /= NumberMCcycles
        return energy

    #Here starts the main program with variable declarations
    NumberParticles = 2
    Dimension = 2
    # seed for rng generator
    seed()
    # Monte Carlo cycles for parameter optimization
    Printout = False
    NumberMCcycles= 10000
    # guess for variational parameters
    x0 = np.array([0.9,0.2])
    # Using Broydens method to find optimal parameters
    res = minimize(Energy, x0, method='BFGS', jac=EnergyDerivative, options={'gtol': 1e-4,'disp': True
    x0 = res.x
    # Compute the energy again with the optimal parameters and increased number of Monte Cycles
    NumberMCcycles= 2**19
    Printout = True
    FinalEnergy = Energy(x0)
    EResult = np.array([FinalEnergy,FinalEnergy])
    outfile.close()
    #nice printout with Pandas
    import pandas as pd
    from pandas import DataFrame
    data ={'Optimal Parameters':x0, 'Final Energy':EResult}
    frame = pd.DataFrame(data)
    print(frame)
```

Resampling analysis

The next step is then to use the above data sets and perform a resampling analysis using the blocking method The blocking code, based on the article of Marius Jonsson is given here

```
    # Common imports
    import os
```

```
# Where to save the figures and data files
DATA_ID = "Results/EnergyMin"

def data_path(dat_id):
    return os.path.join(DATA_ID, dat_id)

infile = open(data_path("Energies.dat"),'r')

from numpy import log2, zeros, mean, var, sum, loadtxt, arange, array, cumsum, dot, transpose, dia
from numpy.linalg import inv

def block(x):
    # preliminaries
    n = len(x)
    d = int(log2(n))
    s, gamma = zeros(d), zeros(d)
    mu = mean(x)

    # estimate the auto-covariance and variances
    # for each blocking transformation
    for i in arange(0,d):
        n = len(x)
        # estimate autocovariance of x
        gamma[i] = (n)**(-1)*sum( (x[0:(n-1)]-mu)*(x[1:n]-mu) )
        # estimate variance of x
        s[i] = var(x)
        # perform blocking transformation
        x = 0.5*(x[0::2] + x[1::2])

    # generate the test observator M_k from the theorem
    M = (cumsum( ((gamma/s)**2*2**arange(1,d+1)[::-1])[::-1] )  )[::-1]

    # we need a list of magic numbers
    q =array([6.634897,9.210340, 11.344867, 13.276704, 15.086272, 16.811894, 18.475307, 20.090235,

    # use magic to determine when we should have stopped blocking
    for k in arange(0,d):
        if(M[k] < q[k]):
            break
    if (k >= d-1):
        print("Warning: Use more data")
    return mu, s[k]/2**(d-k)


x = loadtxt(infile)
(mean, var) = block(x)
std = sqrt(var)
import pandas as pd
from pandas import DataFrame
data ={'Mean':[mean], 'STDev':[std]}
frame = pd.DataFrame(data,index=['Values'])
print(frame)
```