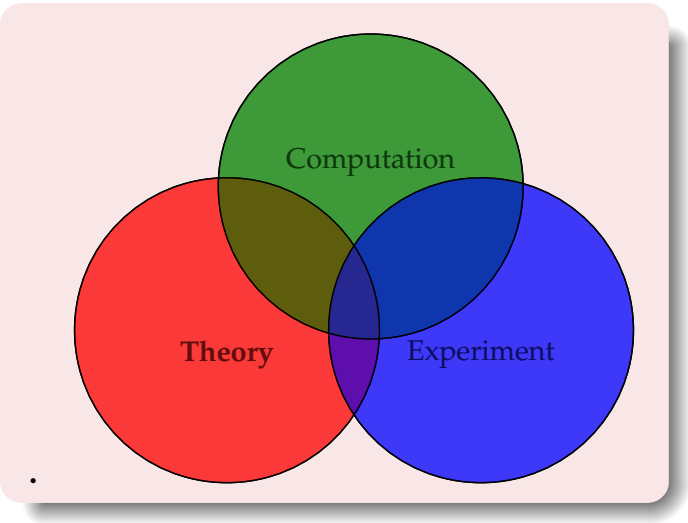
Efficient object orientation for many-body physics problems

Islen Vallejo Henao

Department of physics
University of Oslo

November 14, 2009

Why computational physics?



Advantages

- Better understanding of the processes.
- Cheaper and less dangerous than experiments.
- Allow experiments impossible practically.
- Need to routinely predict processes.

What are the difficulties?

- Numerical method: which one?, does it work properly? Is it efficient?,...
- Programming: howto, modularity, efficiency, maintainibility, reusability? ...
- Hardware: Speed, memory (storage)?

Methodology in computational physics

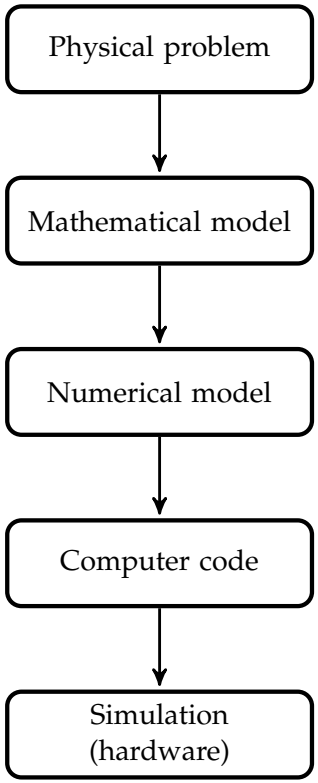


Figure: Simulation chart flow

The many-body problem in quantum mechanics

Goal

Solve the many-electron Schrödinger equation to obtain the ground state energy in quantum mechanical systems with the so-called *closed shell* model.

Time-independent Schrödinger eq.

- For a system of N-particles:

$$\hat{\mathbf{H}}\Psi = E\Psi \tag{1}$$

- Hamiltonian: Total energy

$$\hat{\mathbf{H}} = \hat{\mathbf{K}}(x) + \hat{\mathbf{V}}(x) \tag{2}$$

Physical systems

- Atoms: helium and beryllium.
- Quantum dots.

How to define our Hamiltonian...for atoms?

Assumptions

1. Assume non-relativistic energies.
2. Use Born-Oppenheimer approximation:
   - Nuclei are much more massive than electrons ($\sim 2000:1$ or more).
   - Electron motions much faster than nuclear motions.
   - Assume that electrons move instantly compared to nuclei.
3. Go from a molecular to an electronic Schröndinger equation.

Hamiltonian for hydrogen-like atoms.

$$\widehat{\mathbf{H}} = \underbrace{-\frac{\hbar^2}{2m}\sum_{i=1}^{N}\nabla_{r_i}^2}_{\text{Electronic kinetic energy}} \overbrace{\underbrace{-\frac{\hbar^2}{2m}\sum_{i=1}^{N}\frac{Z}{|\boldsymbol{r}_i-\boldsymbol{R}|}}_{\text{Attraction of electron } i \text{ by nucleus}} + \underbrace{\frac{1}{4\pi\epsilon_0}\sum_{i=1}^{N}\sum_{j=i+1}^{N}\frac{e^2}{|\boldsymbol{r}_i-\boldsymbol{r}_j|}}_{\text{Repulsion between electrons } i \text{ and } j}}^{\text{Potential energy}}$$

How to define our Hamiltonian...for quantum dots?

Assumptions

1. Assume non-relativistic energies.
2. Confining potential modelled by an harmonic oscillator potential.
3. Use the electronic Schröndinger equation.

Hamiltonian for quantum dots.

$$\widehat{\mathbf{H}} = \underbrace{\sum_{i=1}^{N} [-\frac{\hbar^2}{2m^*}\nabla_i^2}_{\text{Electronic kinetic energy}} + \underbrace{\frac{1}{2}m^*\omega^2(x_i^2+y_i^2)]}_{\text{Confining potential}} + \underbrace{\frac{e^2}{4\pi\epsilon_0\epsilon_r}\sum_{i=1}^{N}\sum_{j=i+1}^{N}\frac{1}{|\boldsymbol{r}_i-\boldsymbol{r}_j|}}_{\text{Repulsion}}$$

Why is it a hard problem to solve?

P.A.M. Dirac:

The fundamental laws necessary for the mathematical treatment of a large part of physics and the whole of chemistry are thus completely known, and the difficulty lies only in the fact that application of these laws leads to equations that are too complex to be solved.

Hamiltonian term	**Quantum nature**
Kinetic energy	One-body
Nuclei-electron	One-body
Electron-electron	Two-body (hard)

Table: Nature of the terms in the Hamiltonian.

The wave function contains explicit correlations that leads to non-factoring multi-dimensional integrals.

Quantum Variational Monte Carlo

Variational principle

The expectation value of the energy computed for a Hamiltonian $\hat{\mathbf{H}}$ given a (parametrized) trial wave function Ψ_T is an upper bound to the ground state energy E_0.

$$E_{VMC} = \langle \hat{\mathbf{H}} \rangle = \frac{\int \Psi_T^* \hat{\mathbf{H}} \Psi_T \, d\boldsymbol{R}}{\int \Psi_T^2 \, d\boldsymbol{R}} = \frac{\langle \Psi_T | \hat{\mathbf{H}} | \Psi_T \rangle}{\langle \Psi_T | \Psi_T \rangle} \geq E_0$$

How to compute high dimensional integrals efficiently?

Solution: Translate the problem into a Monte Carlo language.

$$E_{vmc} = \frac{\int \Psi_T^* \Psi_T \frac{\hat{\mathbf{H}} \Psi_T}{\Psi_T} \, d\boldsymbol{R}}{\int \Psi_T^2 \, d\boldsymbol{R}} = \frac{\int |\Psi_T|^2 \left[\frac{\hat{\mathbf{H}} \Psi_T}{\Psi_T} \right] d\boldsymbol{R}}{\int \Psi_T^2 \, d\boldsymbol{R}} = \int P(\boldsymbol{R}) E_L \, d\boldsymbol{R}$$

- Sample configurations, $\boldsymbol{R}$, stochastically.

...How to compute high dimensional integrals efficiently? [continued]

- Average of the local energy E_L over the probability distribution function $P(\boldsymbol{R})$.

$$\langle E \rangle = \int P(\boldsymbol{R}) E_L \, d\boldsymbol{R} \approx \frac{1}{M} \sum_{i=1}^{M} E_L(\boldsymbol{R}_i),$$

  where M is the number of Monte Carlo samples.
- Statistical variance (assuming uncorrelated data set)

$$var(E) = \langle E^2 \rangle - \langle E \rangle^2$$

- *Zero variance principle*.
- The optimization of the trial wave function has as goal to find the optimal set of parameters that gives the minimum energy/variance.
- BUT: We just need a (parametric) trial wave function...!

Trial wave function: $\Psi_T = \Psi_D \Psi_J$ (Slater-Jastrow)

Slater determinant

$$\Psi_D \propto \begin{vmatrix} \phi_1(\boldsymbol{r}_1) & \phi_2(\boldsymbol{r}_1) & \cdots & \phi_N(\boldsymbol{r}_1) \\ \phi_1(\boldsymbol{r}_2) & \phi_2(\boldsymbol{r}_2) & \cdots & \phi_N(\boldsymbol{r}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{r}_N) & \phi_2(\boldsymbol{r}_N) & \cdots & \phi_N(\boldsymbol{r}_N) \end{vmatrix}$$

- Pauli exclusion principle.
- Spin-independent Hamiltonian? $\Psi_D = |\boldsymbol{D}|_\uparrow |\boldsymbol{D}|_\downarrow$.
- Electron-nucleus cusp conditions.

Linear Padé-Jastrow correlation function

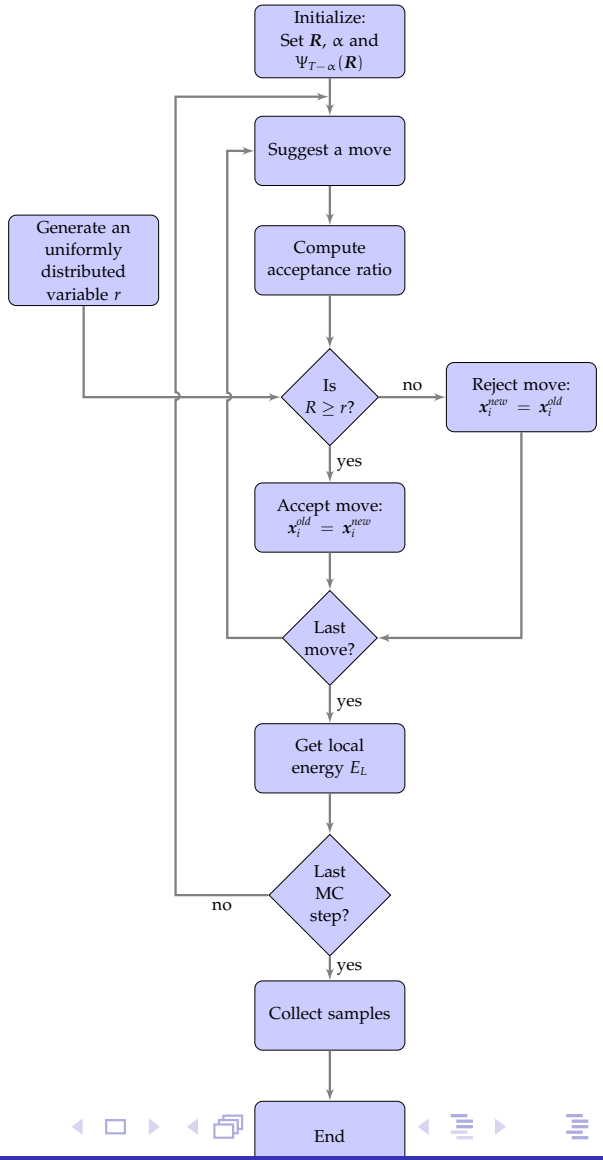
$$\Psi_J = \exp\left(\sum_{j<i} \frac{a_{ij} r_{ij}}{1 + \beta_{ij} r_{ij}}\right)$$

$$r_{ij} = |\boldsymbol{r}_j - \boldsymbol{r}_i|$$

- Linear correlation.
- Two-body correlation.
- Fullfils electron-electron cusp conditions.

Require: *nel, nmc, nes,* δt, $\boldsymbol{R}$ *and* $\Psi_\alpha(\boldsymbol{R})$.
Ensure: $\langle E_\alpha \rangle$.
  for $c = 1$ to *nmc* **do**
    for $p = 1$ to *nel* **do**
      $\boldsymbol{x}_p^{new} = \boldsymbol{x}_p^{cur} + \chi + D\boldsymbol{F}(\boldsymbol{x}_p^{cur})\delta t$

      Compute $\boldsymbol{F}(\boldsymbol{x}^{new}) = \dfrac{\nabla \Psi_T}{\Psi_T}$
      Accept trial move with probability
      min $\left[1, \dfrac{\omega(\boldsymbol{x}^{cur}, \boldsymbol{x}^{new})}{\omega(\boldsymbol{x}^{new}, \boldsymbol{x}^{cur})} \dfrac{|\Psi(\boldsymbol{x}^{new})|^2}{|\Psi(\boldsymbol{x}^{cur})|^2}\right]$
    end for
    Compute $E_L = \dfrac{\widehat{\mathbf{H}}\Psi_T}{\Psi_T} = -\dfrac{1}{2}\dfrac{\nabla^2\Psi_T}{\Psi_T} + V.$
  end for
  Compute $\langle E \rangle = \dfrac{1}{nmc}\sum_{c=1}^{nmc} E_L$ *and*
  $\sigma^2 = \langle E \rangle^2 - \langle E^2 \rangle$.



Wave function optimization, why and how?

Goal

Find an optimal set of parameters $\boldsymbol{\alpha}$ *in* Ψ_T *to minimized the estimated energy.*

Approaches

1. Minimization of the variance of the local energy.
2. Minimization of the energy.
3. A combination of both.

Examples optimization algorithms

1. Stochastic gradient approximation (SGA).
2. Quasi-Newton method.

Minimization of energy with quasi-Newton method

Objective function and its derivative

- Expectation value of the energy:

$$E_{vmc} = \frac{\int |\Psi_T|^2 \left[\frac{\hat{\mathbf{H}}\Psi_T}{\Psi_T}\right] d\boldsymbol{R}}{\int \Psi_T^2 \, d\boldsymbol{R}}$$

- Derivative of the expectation value of the energy:

$$\frac{\partial E}{\partial c_m} = 2\left[\left\langle E_L \frac{\frac{\partial \Psi_{T_{c_m}}}{\partial c_m}}{\Psi_{T_{c_m}}} \right\rangle - E\left\langle \frac{\frac{\partial \Psi_{T_{c_m}}}{\partial c_m}}{\Psi_{T_{c_m}}} \right\rangle\right]$$

Strategies for computing the derivative

- Direct analytical differentiation:

$$\frac{\partial \Psi_{T_{c_m}}}{\partial c_m} = \frac{\partial}{\partial c_m}\left[|\boldsymbol{D}|_\uparrow |\boldsymbol{D}|_\downarrow \prod_{i=1}^{N}\prod_{i=j+1}^{N} J(r_{ij})\right] = ...? \qquad \text{Enjoy it!}$$

- Numerical derivative: central differences: Just for Ψ_{SD} we have to do

$$\frac{d\Psi_{SD}}{d\alpha_m} = \frac{\Psi_{SD}(\alpha_m+\Delta\alpha_m)-\Psi_{SD}(\alpha_m-\Delta\alpha_m)}{2\Delta\alpha_m} + \mathcal{O}(\Delta\alpha_m^2)$$

(...to be done per parameter).

Minimization of energy with quasi-Newton method

Trick to compute the derivative of the energy

Goal:

$$\frac{\partial E}{\partial c_m} = 2\left[\left\langle E_L \frac{\frac{\partial \Psi_{T_{c_m}}}{\partial c_m}}{\Psi_{T_{c_m}}}\right\rangle - E\left\langle \frac{\frac{\partial \Psi_{T_{c_m}}}{\partial c_m}}{\Psi_{T_{c_m}}}\right\rangle\right]$$

1. Split the trial wave function: $\Psi_{T_{c_m}} = \Psi_{SD_{c_m}}\Psi_{J_{c_m}} = \Psi_{SD_{c_m}\uparrow}\Psi_{SD_{c_m}\downarrow}\Psi_{J_{c_m}}$.
2. Rewrite the derivatives as: $\frac{\partial \ln \Psi_{T_{c_m}}}{\partial c_m} = \frac{\partial \ln(\Psi_{SD_{c_m}\uparrow})}{\partial c_m} + \frac{\partial \ln(\Psi_{SD_{c_m}\downarrow})}{\partial c_m} + \frac{\partial \ln(\Psi_{J_{c_m}})}{\partial c_m}$
3. Divide he standard expression $\frac{\mathrm{d}}{\mathrm{d}t}(\det \mathbf{A}) = (\det \mathbf{A})\,\mathrm{tr}\left(\mathbf{A}^{-1}\frac{\mathrm{d}\mathbf{A}}{\mathrm{d}t}\right)$ by $\det \mathbf{A}$ to get:
   $\frac{d}{dt}\ln\det \mathbf{A}(t) = \mathrm{tr}\left(\mathbf{A}^{-1}\frac{d\mathbf{A}}{dt}\right) = \sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij}^{-1}\dot{A}_{ji}$
4. This expression inserted in step (2) gives:

$$\boxed{\frac{\partial \ln \Psi_{T_{c_m}}}{\partial c_m} = (\sum_{i=1}^{N}\sum_{j=1}^{N} D_{ij}^{-1}\dot{D}_{ji})_{\uparrow} + (\sum_{i=1}^{N}\sum_{j=1}^{N} D_{ij}^{-1}\dot{D}_{ji})_{\downarrow} + \frac{\partial \ln(\Psi_{J_{c_m}})}{\partial c_m}}$$

Implementing a QVMC simulator

How and why [later]?

1. Programming style?: Object-orientation.
2. Methodology?: Prototyping-test-extension/migration.
3. Programming languages?: Python/C++.
4. Structure of the program?: Define classes [next slide], methods and flux of information (algorithm).

Basic class structure of a QVMC simulator

- An administration class: VMC.
- A class computing energies: Energy.
- A class containing the trial wave function: PsiTrial.
- A class for administrate the configuration space: Particle.

Quick design of a QVMC simulator in Python

```
#Import some packages
...
class VMC():
  def __init__(self, _dim, _np, _charge, ...,_parameters):
    ...
    particle = Particle(_dim, _np, _step)
    psi      = Psi(_np, _dim, _parameters)
    energy   = Energy(..., particle, psi, _charge)
    self.mc  = MonteCarlo(psi, _ncycles, particle, energy,...)

  def doVariationalLoop(self):
    ...
    for var in xrange(nVar):
      self.mc.doMonteCarloImportanceSampling()
      self.mc.psi.updateVariationalParameters()
```

Easy creation/manipulation of matrices in Python

```
...
class Particle():
  def __init__(self, _dim, _np, _step):
    # Initialize matrices for configuration space
    r_old = zeros((_np, _dim))
     ...

  def acceptMove(self, i):
    self.r_old[i,0:dim] = self.r_new[i,0:dim]

  ...
  def setTrialPositionsBF(self):
    dt = self.step
    r_old = dt*random.uniform(-0.5,0.5,size=np*dim)
    r_old.reshape(np,dim)
  ...
```

Calling the code

```
from VMC import *

# Set parameters of simulation
nsd = 3          # Number of spatial dimensions
nVar= 10         # Number of variations (optimization method)
nmc = 10000      # Number of monte Carlo cycles
nel = 2          # Number of electrons
Z   = 2.0        # Nuclear charge
 ...
vmc = VMC(nsd, nel, Z,..., nmc, dt, nVar, varPar)
vmc.doVariationalLoop()
vmc.mc.energy.printResults()
```

What do you gain from using Python?

High level language?

- Clear and compact syntax.
- Support all the major program styles.
- Runs on all major platforms.
- Free, open source.
- Comprehensive standard library.
- Huge collection of free modules on the web.
- Good support for scientific computing.

However...

How well performs Python with respect to C++?

Comparing Python to C++ computing VMC

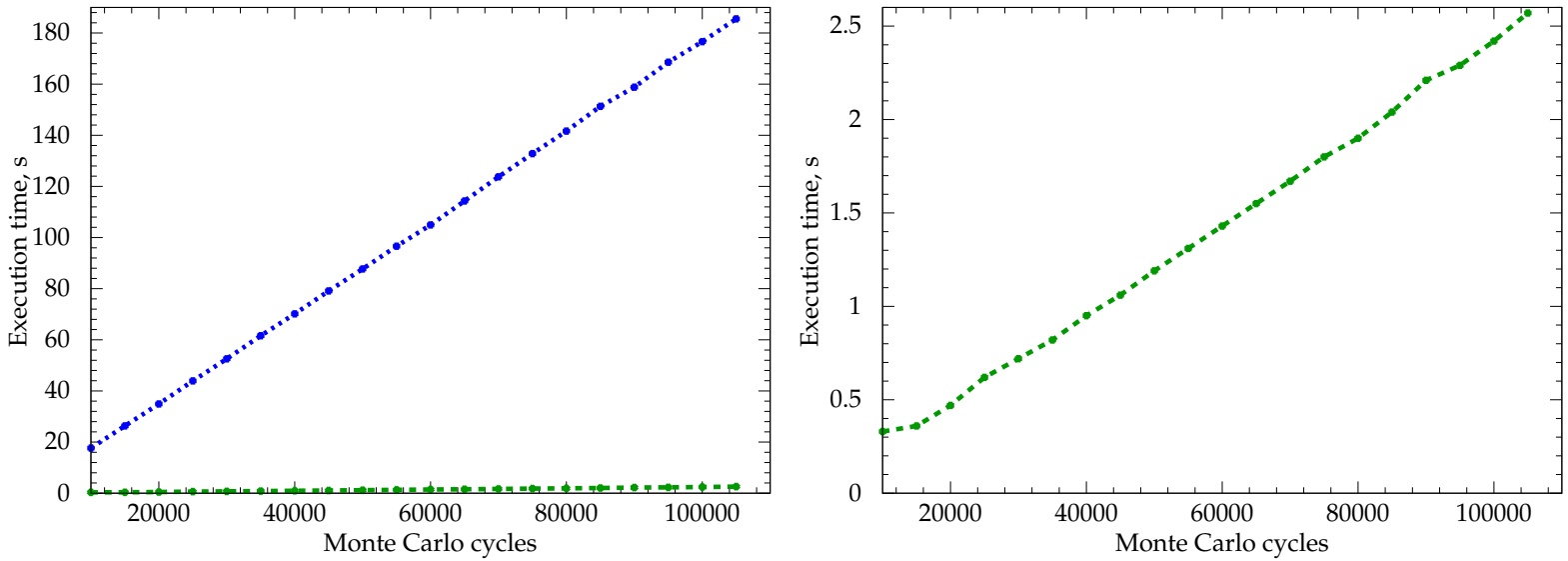


Figure: Execution time as a function of the number of Monte Carlo cycles for a Python (blue) and C++ (green) simulators implementing the QVMC method with importance sampling for the He atom.

CONCLUSION: Python is SLOW, except when it is not running!

Detecting bottlenecks in Python

# calls	Total time	Cum. time	Class:function
1	9.153	207.061	MonteCarlo.py:(doMCISampling)
1	0.000	207.061	VMC.py:(doVariationalLoop)
1910014	23.794	159.910	Psi.py:(getPsiTrial)
100001	12.473	117.223	Psi.py:(getQuantumForce)
1910014	58.956	71.704	Psi.py:(getModelWaveFunctionHe)
50000	0.864	66.208	Energy.py:(getLocalEnergy)
1910014	57.476	64.412	Psi.py:(getCorrelationFactor)
50000	8.153	62.548	Energy.py:(getKineticEnergy)
6180049	21.489	21.489	:0(sqrt)
900002	4.968	4.968	:0(copy)
300010	2.072	2.072	:0(zeros)
50000	2.272	2.796	Energy.py:(getPotentialEnergy)

Table: Profile of a QVMC simulator with importance sampling for the He atom implemented in Python. The run was done with 50000 Monte Carlo cycles.

Can "Python" do better?: Extending Python with C++

```
sys.path.insert(0,'./extensions') # Set the path to the extensions
import ext_QVMC                   # Extension module

class Vmc():
  def __init__(self, _Parameters):
    # Create an object of the 'conversion class'
    self.convert = ext_QVMC.Convert()

    # Get the paramters of the currrent simulation
    simParameters = _Parameters.getParameters()
    alpha = simParameters[6]
    self.varpar = array([alpha,beta])

    # Convert a Python array to a MyArray object
    self.v_p = self.convert.py2my_copy(self.varpar)

    # Create objects to be extended in C++
    self.psi  = ext_QVMC.Psi(self.v_p, self.nel, self.nsd)...
```

Calling code for the mixed Python/C++ simulator

```
from SimParameters import * # Class encapsulating the \
                           # parameters of simulation
from Vmc import *          # Import the simulator box.

# Create an object containing the
# parameters of the current simulation
simpar = SimParameters('Be.data')

# Create a Variational Monte Carlo simulation
vmc = Vmc(simpar)

vmc.doVariationalLoop()
vmc.energy.doStatistics("resultsBe.data", 1.0)
```

Comparing Python to C++

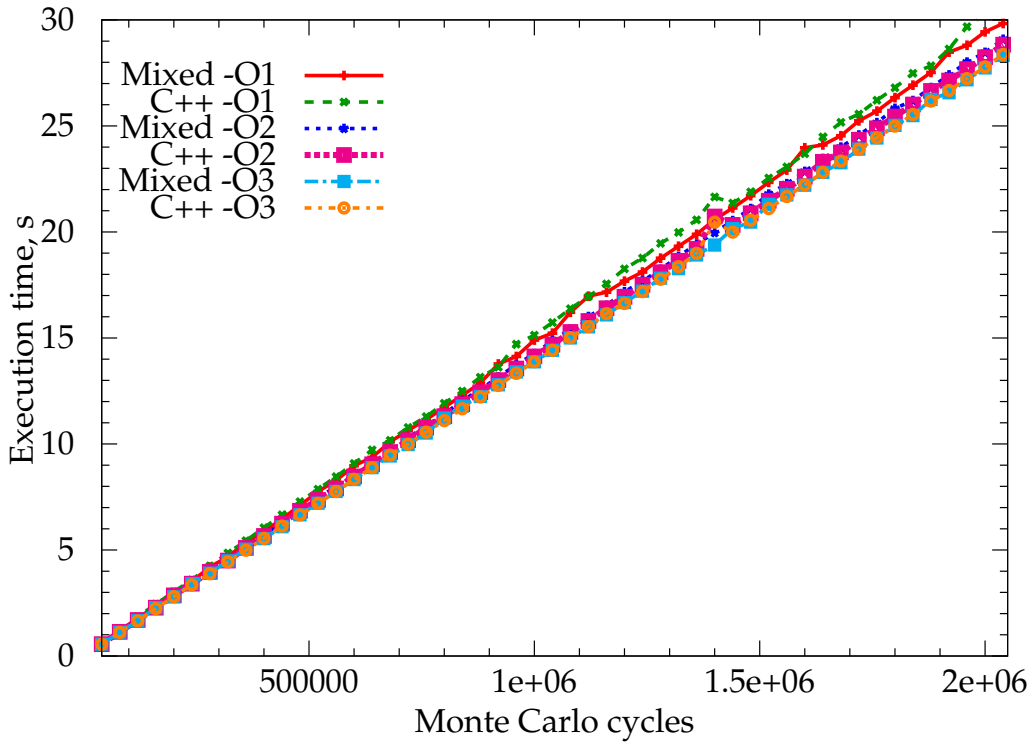


Figure: Execution time as a function of the number of Monte Carlo cycles for mixed Python/C++ and pure C++ simulators implementing the QVMC method with importance sampling for He atom.

Structuring a simulator with object-orientation?

Defining classes

1. From the mathematical model and the algorithms, **identify the main components of the problem** (classes).
   - Vmc simulation: `VmcSimulation.h`
   - Parameters in simulation: `Parameters.h`
   - Monte Carlo method: `MonteCarlo.h`
   - Energy: `Energy.h`
   - Potential:`Potential.h`
   - Trial wave function: `PsiTrial` ...
2. **Identify the mathematical and algorithmic parts changing from problem to problem** (superclasses).
   - Potential:`Potential.h`
   - Trial wave function: `PsiTrial`
   - Monte Carlo method: `MonteCarlo.h` ...

Enhance flexibility

3. **List what each of these classes should do**, IN GENERAL (member functions), e.g., for `PsiTrial.h`:
   - Compute the acceptance ratio: `getPsiPsiRatio()`.
   - Compute the quantum force: `getQuantumForce()`. ...

```
#include "SomeClass.h"

class PsiTrial{
  public:
    virtual double getAcceptanceRatio()=0;
    virtual MyArray<double> getQuantumForce()=0;
    virtual getLapPsiRatio()=0;
}
```

Inheritance (specializing behaviours): *is a (kind of)*-relationship

4. Be specific with the behaviour , i.e., create subclases by finding ***is a (kind of)*** -relationships.(subclasses) For example:
   - Slater-Jastrow (`SlaterJastrow.h`) ***is a (kind of)*** trial wave function (`PsiTrial.h`).
   - Slater alone (`SlaterAlone.h`) ***is a (kind of)*** trial wave function (`PsiTrial.h`).
   - ...
   - Coulomb one-body (`OneBodyCoulomb.h`) ***is a (kind of)*** potential (`Potential.h`).

Composition: *has a*-relationship

5. Connect the whole structure with relatonships of type ***has a*** (composition).

$$\Psi_T = |\boldsymbol{D}|_\uparrow |\boldsymbol{D}|_\downarrow |J(rij)$$

For example: Slater-Jastrow wave function (`SlaterJastrow.h`) ***has a*** Slater determinant (`SlaterDeterminant.h`) and a correlation function (`CorrelationFnc.h`).

```
#include "SlaterDeterminant.h"
#include "CorrelationFnc.h"

class SlaterJastrow: public PsiTrial{
  private:
    SlaterDeterminant *slater;
    CorrelationFnc *correlation;

  public:
    SlaterJastrow(SlaterDeterminant *sd,
                  CorrelationFnc *cor):
                  slater(sd), correlation(cor){}

    double getPsiPsiRatio(){
      return slater->getDetDetRatio(...)
           *correlation->getCorCorRatio(...);
    }
};
```

Creating trial wave functions

```
void VmcSimulator::setTrialWaveFnc(){
  ...
  SlaterDeterminant *sd = new ↩
      SlaterDeterminant(...);
  Correlation *pj = new PadeJastrow(...);

  PsiTrial *s = new SlaterJastrow(sd, pj);
}
```

or

```
void VmcSimulator::setTrialWaveFnc(){
  ...
    SlaterDeterminant *sd = new ↩
        SlaterDeterminant(...);
    PsiTrial *sj = new SlaterAlone(sd);
}
```

Analytical GS for atoms (without correlation)

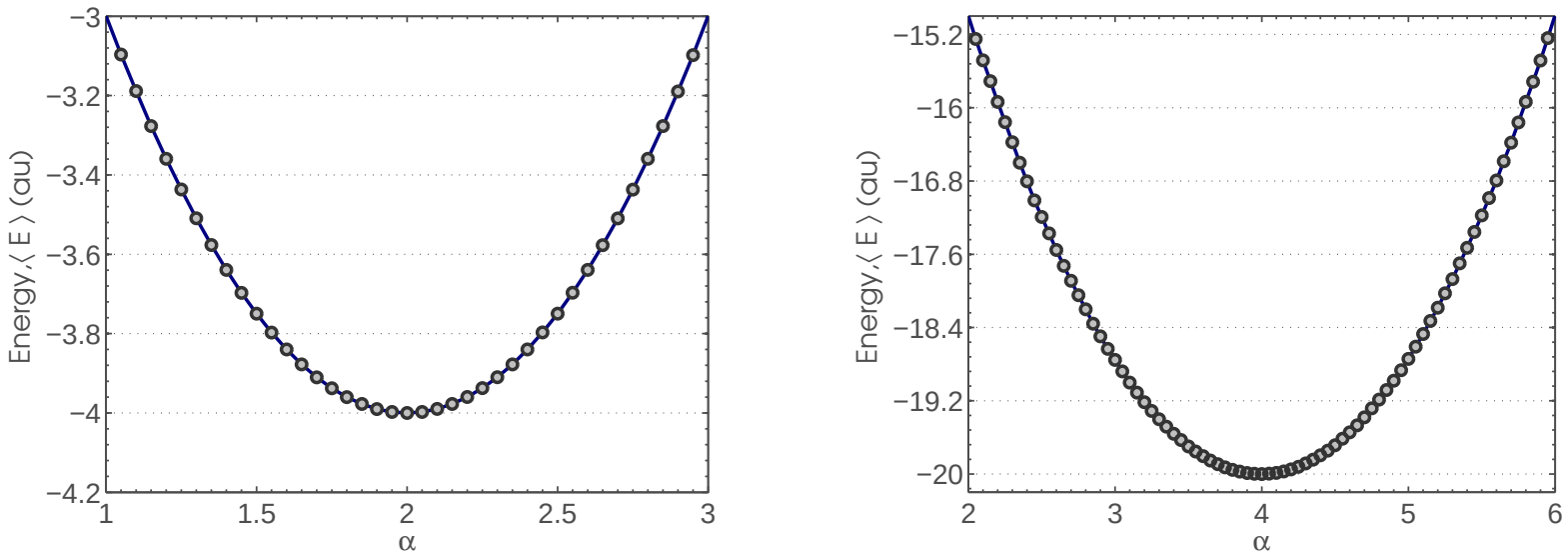


Figure: Dependence of the energy on the parameter α for He(left) and Be(right) atoms.

Analytical GS for quantum dots (without correlation)

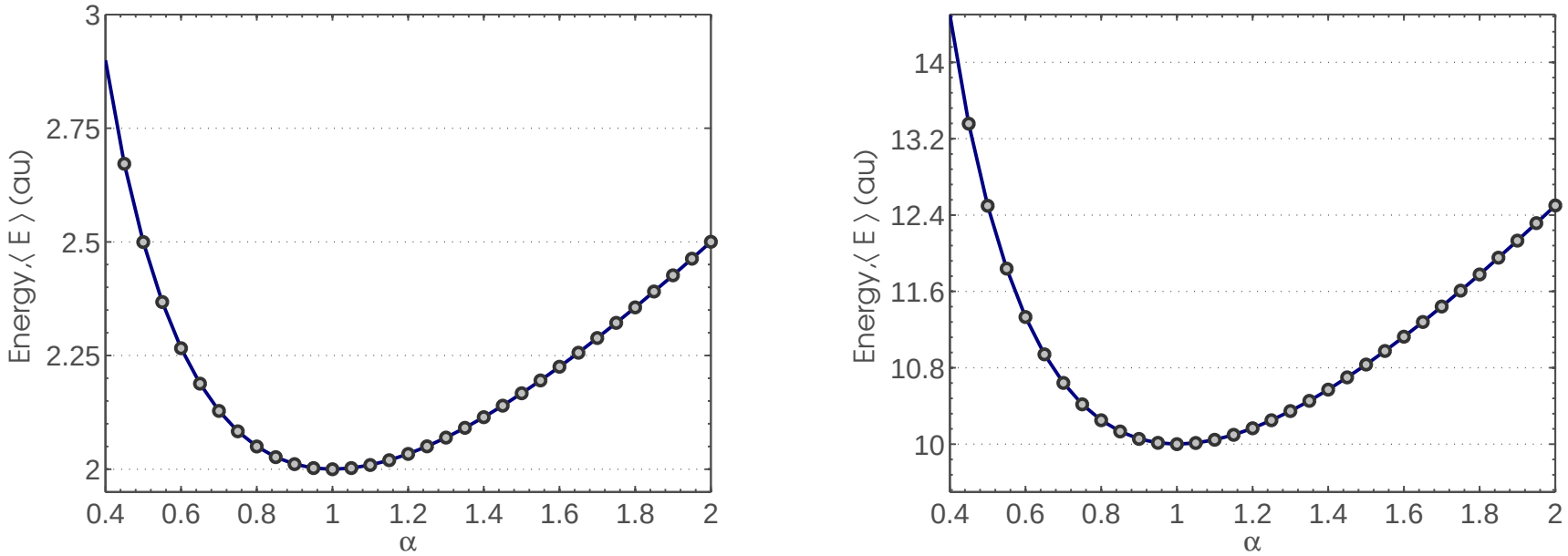
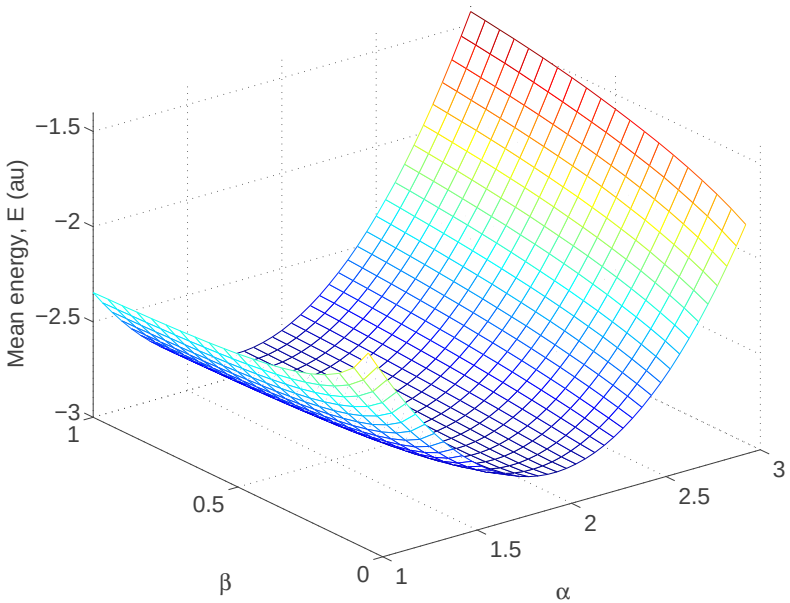


Figure: Dependence of the energy on the parameter α for a two dimensional harmonic oscillator with two and six electrons electrons, respectively.

Graphical estimation of the GS energy for atoms



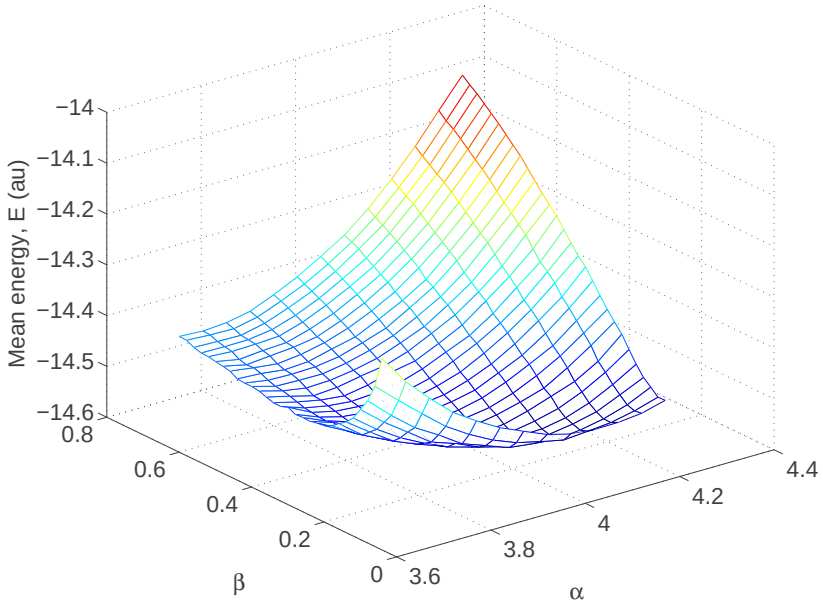
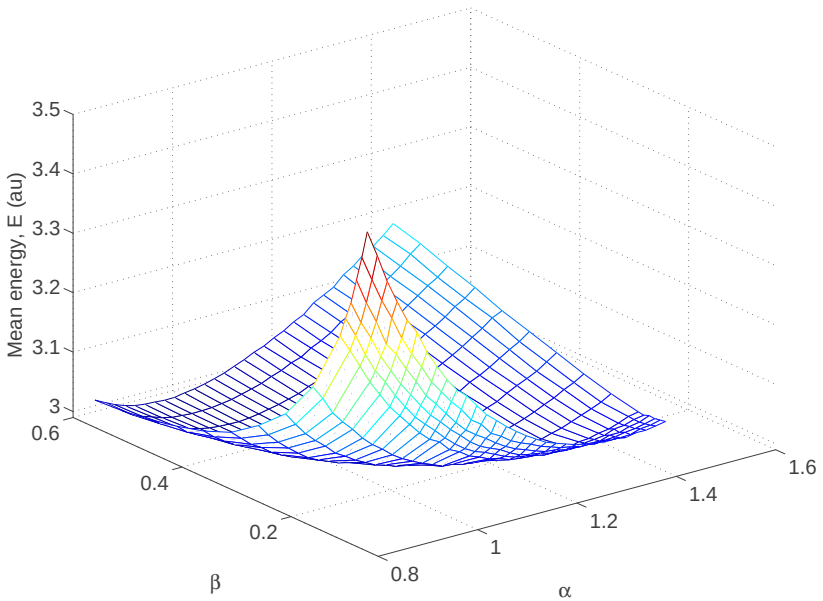


Figure: Dependence of the energy on the parameters α and β for He and Be atoms. Experiment was carried out with 10^7 Monte Carlo

Graphical estimation of the GS energy for QD



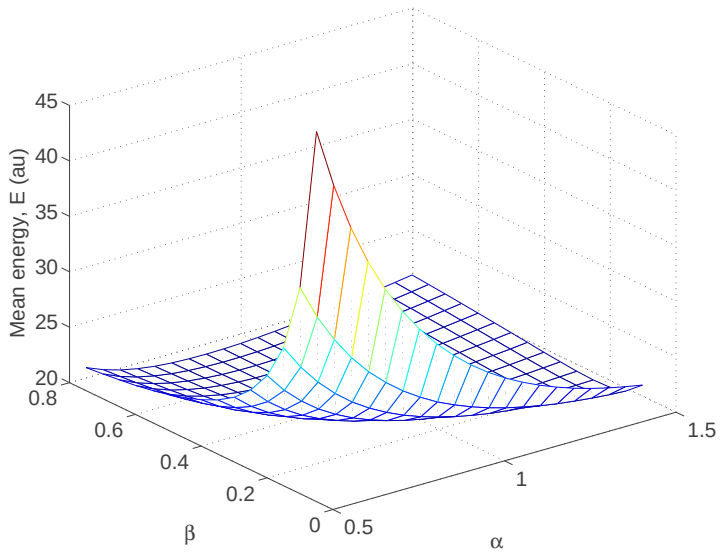


Figure: Dependence of the energy on the parameters α and β for a two-dimensional quantum dots with two and electrons, respectively.

Graphical estimation of the GS energy for helium

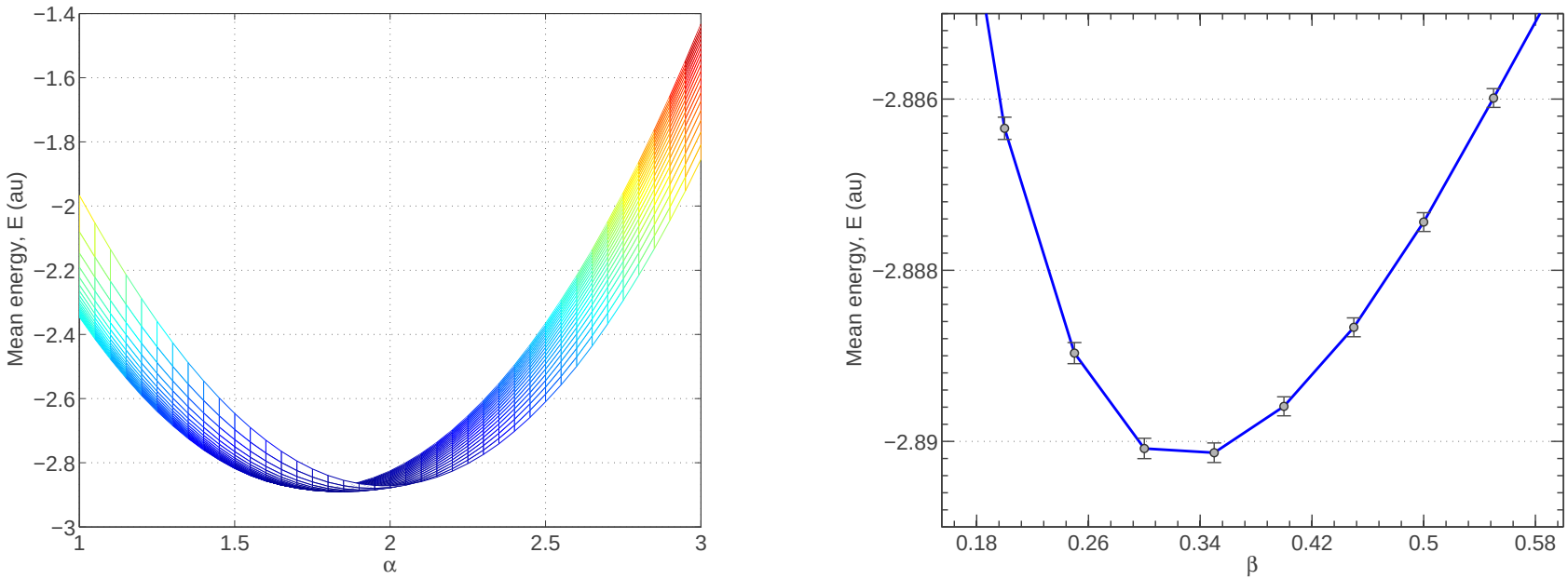


Figure: Dependence of the energy on α (left) along the value of β (right) that gives the minimum variational energy for a He atom.

Graphical estimation of the GS energy for beryllium

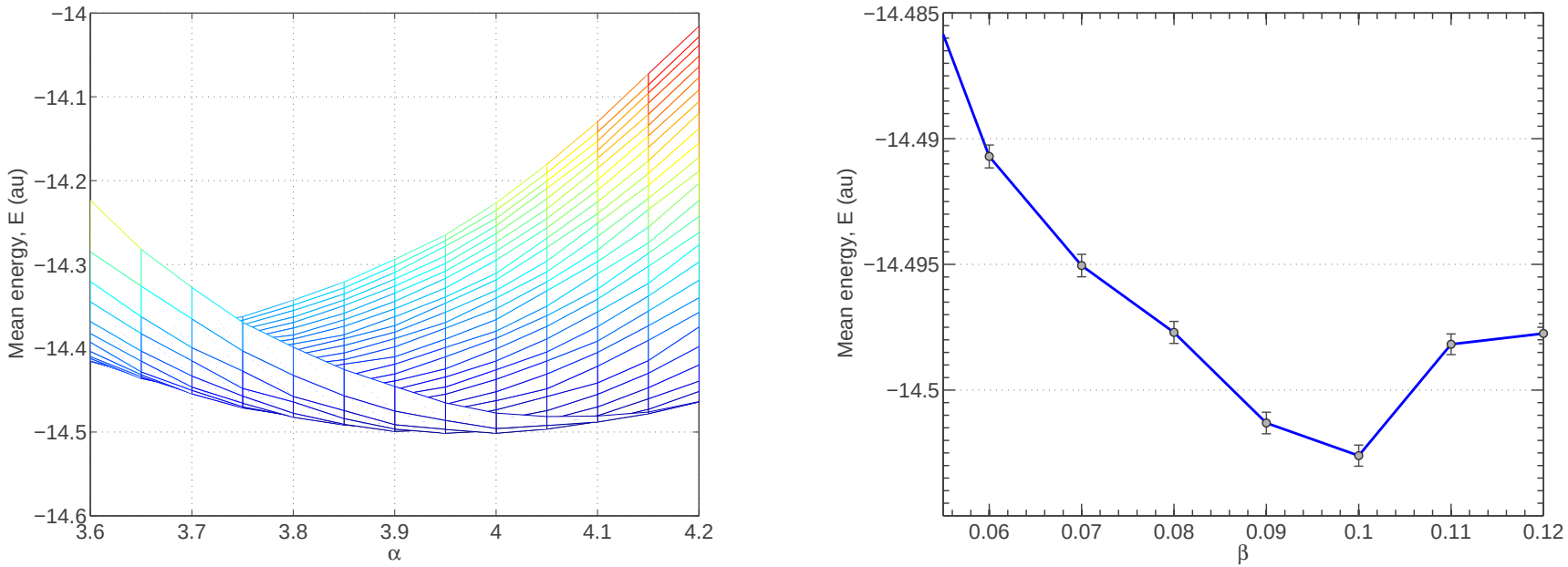


Figure: Dependence of the energy on α (left) along the value of β (right) that gives the minimum variational energy for a Be atom.

Graphical estimation of the GS energy for 2-e^- QD

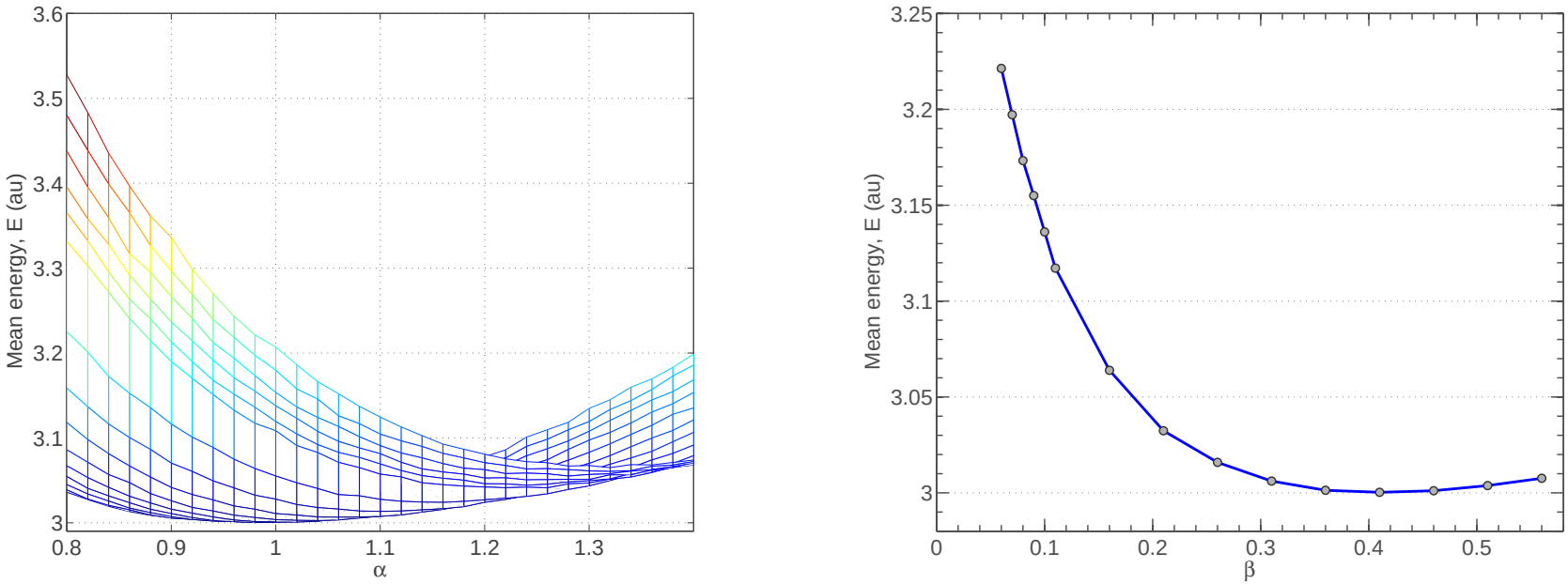


Figure: Dependence of the energy on α (left) along the value of β (right) that gives the minimum variational energy for a two-dimensional quantum dot with two electrons.

Graphical estimation of the GS energy for 6-e^- QD

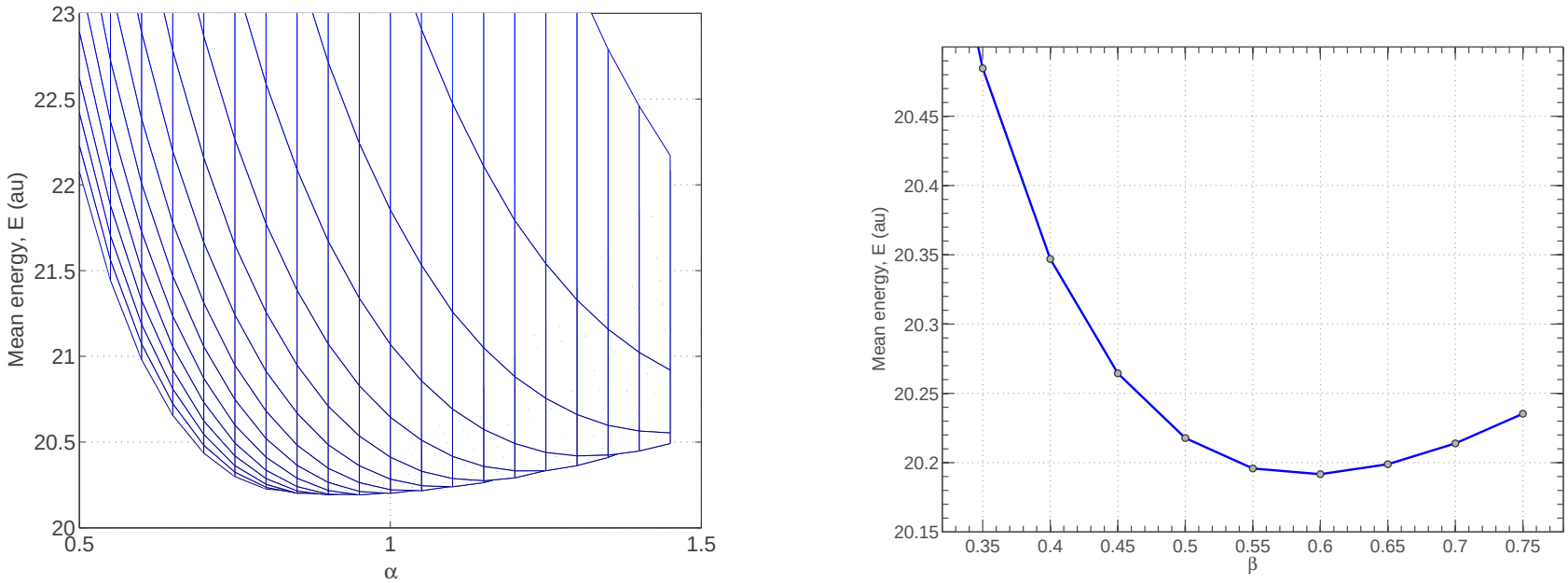


Figure: Dependence of the energy on α (left) along the value of β (right) that gives the minimum variational energy for a two-dimensional quantum dot with six electrons.

Summary of the results with graphical estimation

System	$\alpha_{optimal}$	$\beta_{optimal}$	**Energy**, $\langle E \rangle$ (au)
He	1.85	0.35	$-2.8901 \pm 1.0 \times 10-4$
Be	3.96	0.09	$-14.5043 \pm 4.0 \times 10^{-4}$
2DQDot2e ($\omega = 1.0$)	0.98	0.41	$3.0003 \pm 1.2 \times 10^{-5}$
2DQDot6e ($\omega = 1.0$)	0.9	0.6	$20.19 \pm 1.2 \times 10^{-4}$

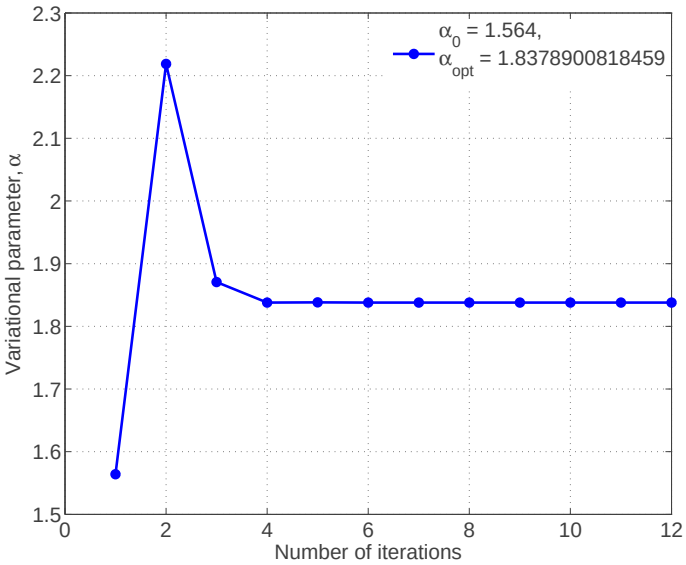
Table: Ground state energy and corresponding variational parameters α and β estimated graphically. (2DQDot2e stands for two-dimensional quantum dot with two electrons).

Optimization with Quasi-Newton method

System	α_0	β_0	α_{opt}	β_{opt}	**Energy**, (au)
He	1.564	0.134	1.838	0.370	-2.891
Be	3.85	0.08	3.983	0.104	-14.503

Table: Optimized variational parameters and corresponding energy minimization using a quasi-Newton method. Simulation parameters: 10^7 Monte Carlo cycles with 10 % equilibration steps, $dt = 0.01$.

Convergence to the optimal parameters for helium



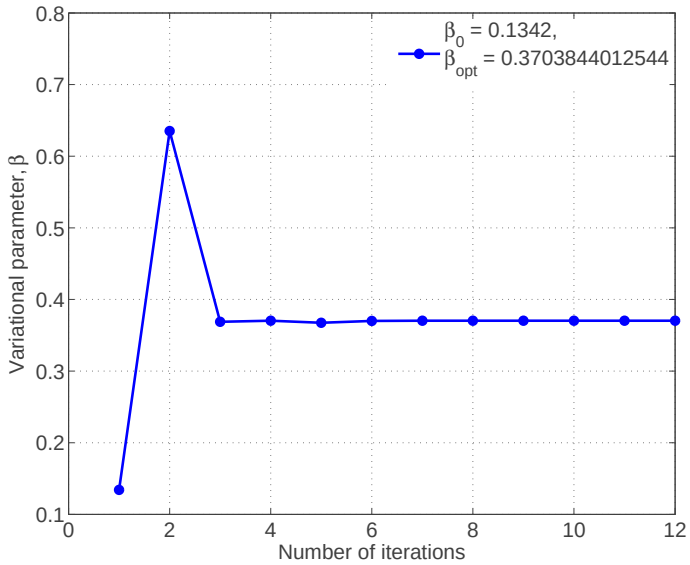


Figure: Evolution of the variational parameters α and β as a function of the number of iterations during the optimization of the trial wave function of He atom with quasi-Newton method. The experiment was carried out with 10^7 Monte Carlo cycles and 10% equilibration steps in f

Convergence to the minimal energy for helium

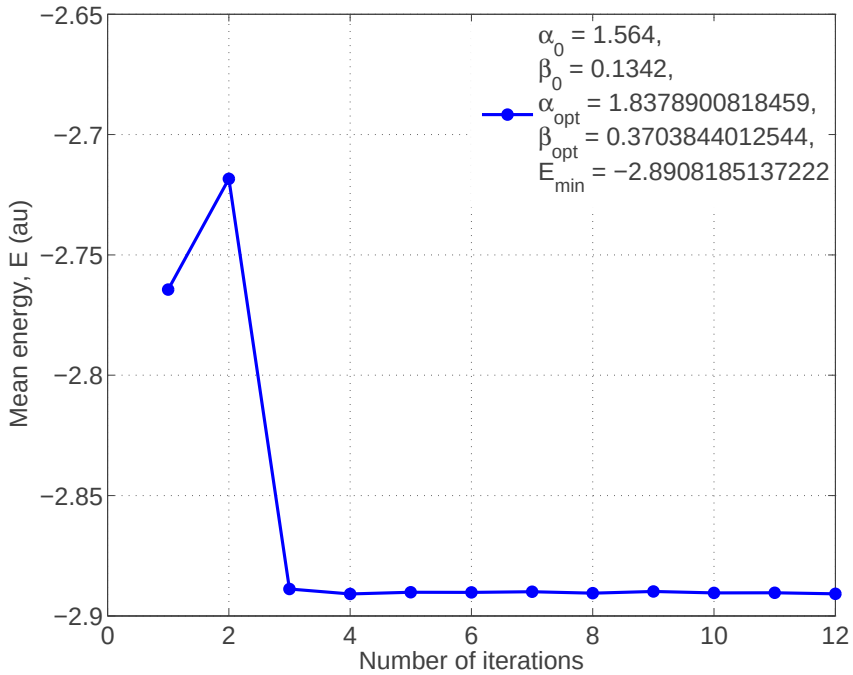
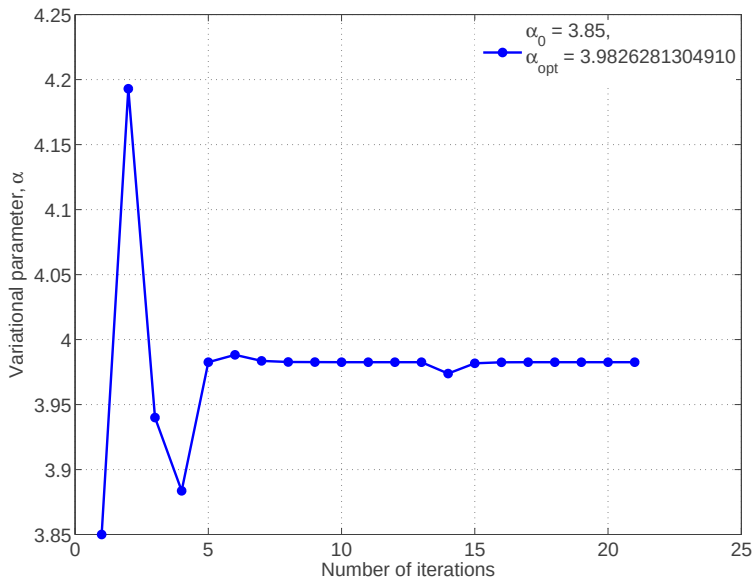


Figure: Evolution of the energy as a function of the number of iterations during the optimization of the trial wave function of He atom with the quasi-Newton method. The experiment was carried out with 10^7 Monte Carlo cycles and 10% equilibration steps in four

Convergence to the optimized parameters for Be



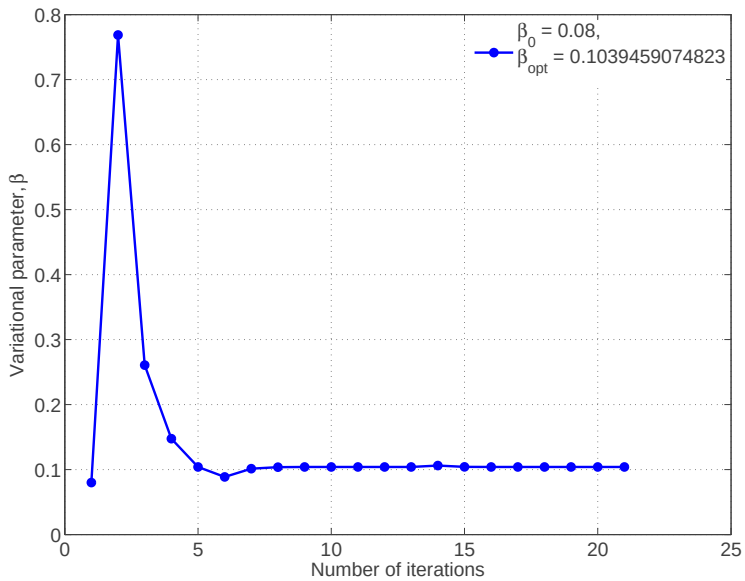


Figure: Evolution of the variational parameters α and β with the number of iterations during the optimization of the trial wave function of Be atom with the quasi-Newton method. The experiment was carried out with 10^7 Monte Carlo cycles and 10% equilibration

Convergence to the minimal energy for Be

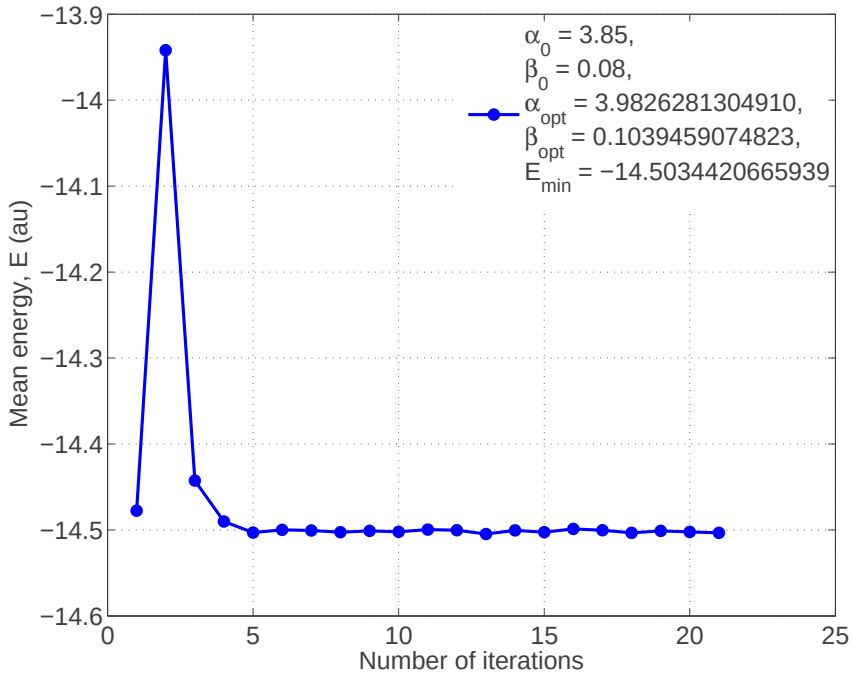


Figure: Evolution of the energy with the number of iterations during the optimization of the trial wave function of Be atom with the quasi-Newton method. The experiment was carried out with 10^7 Monte Carlo cycles and 10% equilibration steps in four nodes.

Extrapolation of energy to zero dt for He

Time step	**Energy**, (au)	**Error**	**Accepted moves**, (%)
0.002	-2.891412637854	5.5e-4	99.97
0.003	-2.890797649433	4.5e-4	99.94
0.004	-2.890386198895	4.0e-4	99.91
0.005	-2.890078440930	3.5e-4	99.88
0.006	-2.890463490951	3.2e-4	99.84
0.007	-2.890100432462	2.8e-4	99.81
0.008	-2.889659923905	2.7e-4	99.77

Table: Energy computed for the He atom and the error associated as a function of the time step. Parameters: 10^7 Monte Carlo cycles with 10 % equilibration steps, $\alpha = 1.8379$ and $\beta = 0.3704$.

Extrapolation of energy to zero dt for He

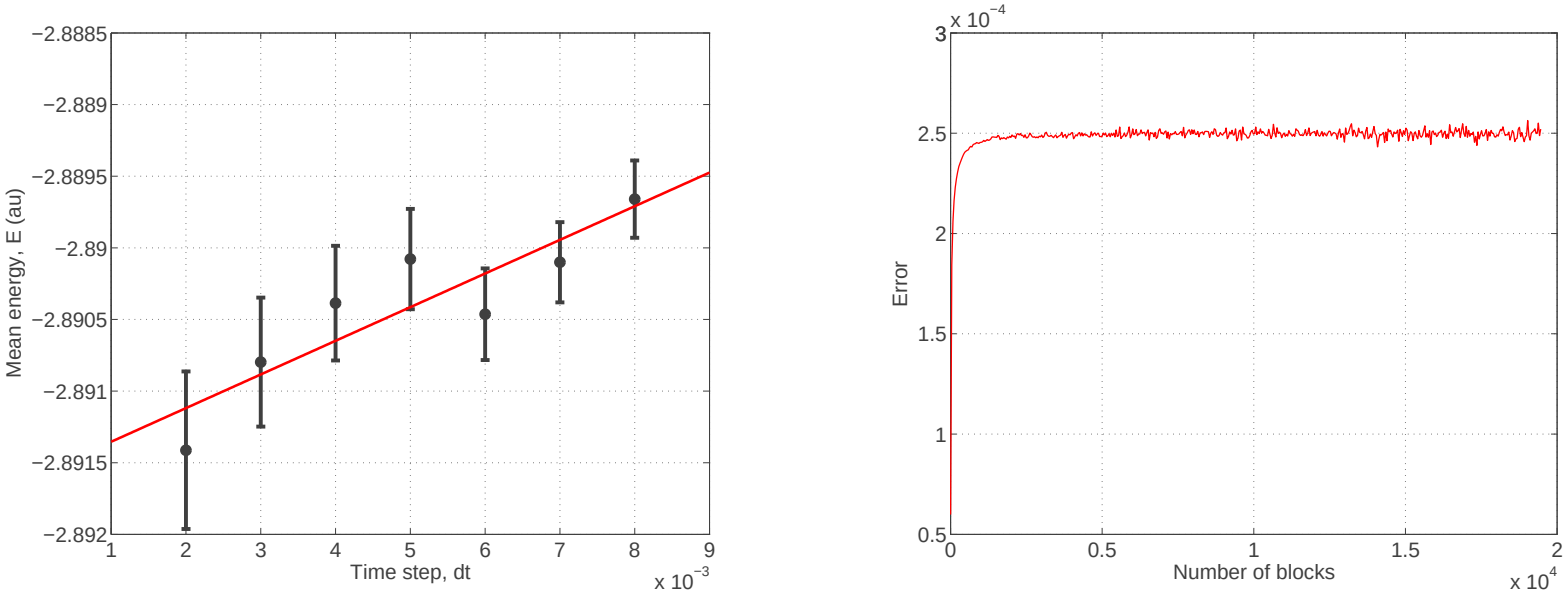


Figure: To the left we show the extrapolation to $dt-$zero of the energy for a He atom. To the right we display the details of the blocking analysis at $dt = 0.01$ where the energy $-2.89039 \pm 2.5 \times 10^{-4}\, au$. Experimental setup: 10^7 Monte Carlo cycles, 10% equilibration steps

Extrapolation of energy to zero dt for Be

Time step	**Energy**, (au)	**Error**	**Accepted moves**, (%)
0.004	-14.50321303316	1.3e-3	99.61
0.005	-14.50266236227	1.2e-3	99.47
0.006	-14.50136820967	1.1e-3	99.32
0.007	-14.50314292468	1.0e-3	99.17
0.008	-14.50206184582	9.5e-4	99.01
0.009	-14.50164368104	8.5e-4	98.85
0.01	-14.50145748870	8.0e-4	98.68

Table: Energy computed for the Be atom and the error associated as a function of the time step. Parameters: 10^7 Monte Carlo cycles with 10 % equilibration steps, $\alpha = 3.983$ and $\beta = 0.103$.

Extrapolation of energy to zero dt for Be

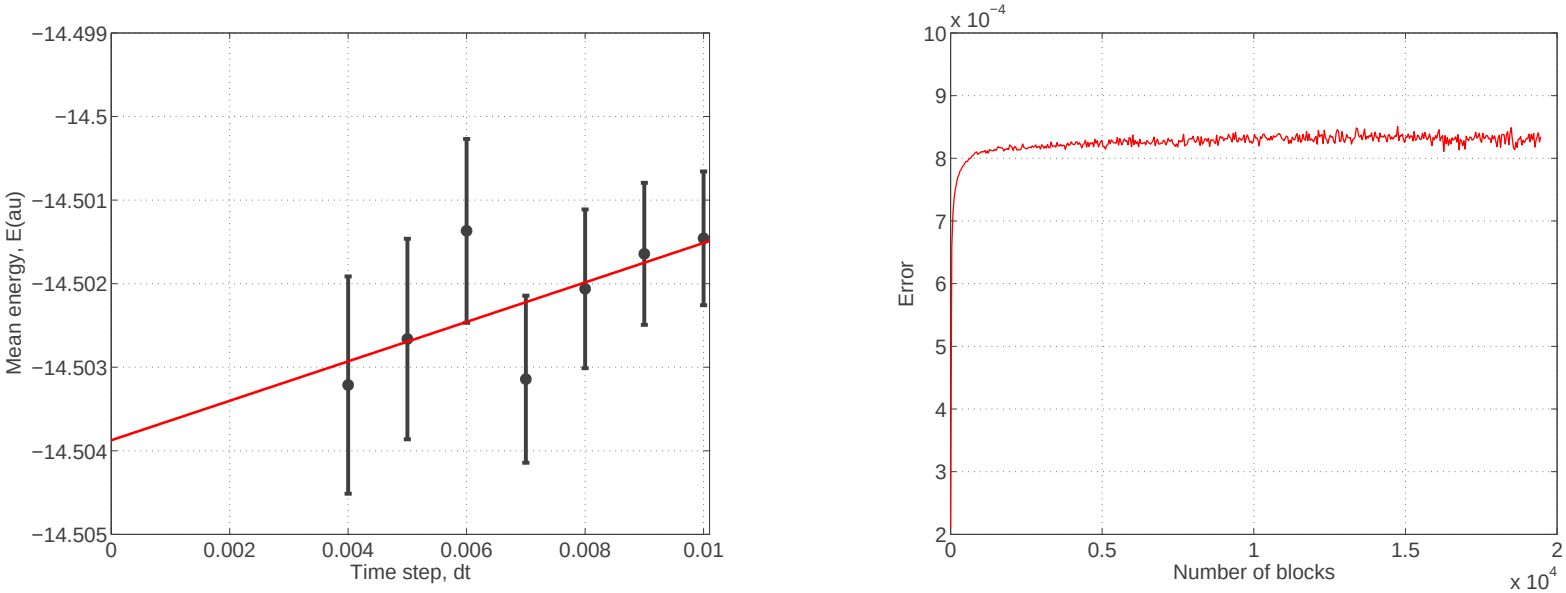


Figure: Extrapolation to $dt-$zero of the energy (left) and blocking analysis at $dt = 0.01$ for the Be atom where the energy $E = -14.50146 \pm 8.5 \times 10^{-4}\, au$. Experimental setup: 10^7 Monte Carlo cycles, 10% equilibration steps for four nodes, with $\alpha = 3.983$ and $\beta = 0.193$

Extrapolation of energy to zero dt for 2-e^- electron QD

Time step	**Energy**, (au)	**Error**	**Accepted moves**, (%)
0.01	3.000340072477	4.5e-5	99.95
0.02	3.000357900850	3.2e-5	99.87
0.03	3.000364180564	2.6e-5	99.77
0.04	3.000384908560	2.2e-5	99.65
0.05	3.000370330692	2.0e-5	99.52
0.06	3.000380980039	1.8e-5	99.37
0.07	3.000402836533	1.7e-5	99.21

Table: Results of a blocking analysis for several time steps. The system was a two-dimensional quantum dot with two electrons and $\omega = 1.0$. The rest of the parameters were: 10^7 Monte Carlo cycles with 10 % equilibration steps, and $\alpha = 0.99044$ and $\beta = 0.39994$ taken from Albrigtsen(2009).

Extrapolation of energy to zero dt for 2-e^- electron QD

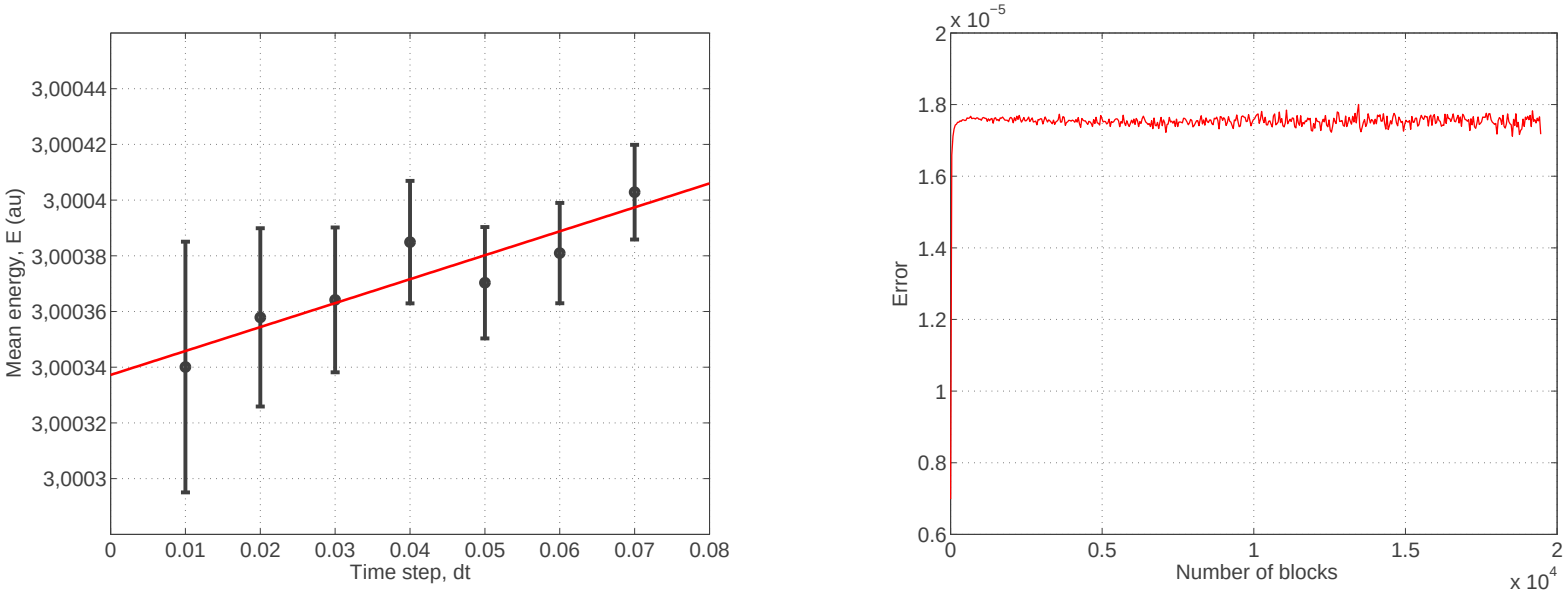


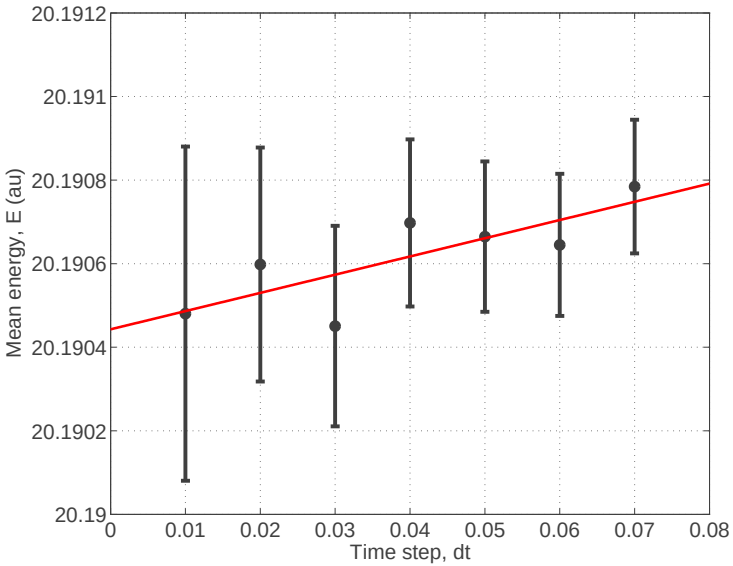
Figure: Extrapolation to $dt-$zero of the energy for a two-dimensional quantum dot with two electrons (left) and mean energy error ($E_{min} = 3.000381 \pm 1.8 \times 10^{-5}\, au$) at $dt = 0.06$ as a function of the number of blocks for a 2DQDot2e (right). Parameters of simulation: 1×10^7 Monte Carlo steps with 10 % equilibration steps, $\alpha = 0.99044$

Extrapolation of energy to zero dt for 6-e^- electron QD

Time step	**Energy**, (au)	**Error**	**Accepted moves**, (%)
0.01	20.19048030567	4.0e-4	99.90
0.02	20.19059799459	2.8e-4	99.75
0.03	20.19045049792	2.4e-4	99.55
0.04	20.19069748408	2.0e-4	99.34
0.05	20.19066469178	1.8e-4	99.10
0.06	20.19064491561	1.7e-4	98.85
0.07	20.19078449010	1.6e-4	98.58

Table: Results of a blocking analysis for several time steps. The system was a two-dimensional quantum dot with six electrons and $\omega = 1.0$. The rest of the parameters were: 10^7 Monte Carlo cycles with 10 % equilibration steps, and $\alpha = 0.926273$ and $\beta = 0.561221$ taken from Albrigtsen (2009).

Extrapolation of energy to zero dt for 6-e^- electron QD



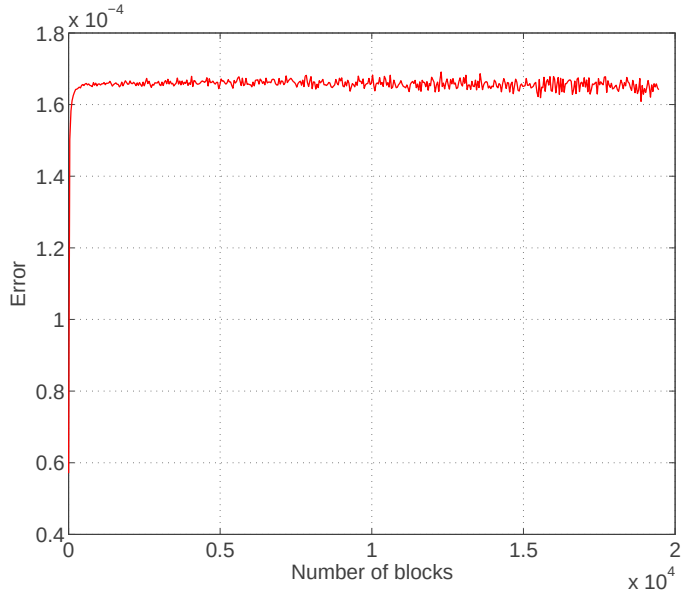


Figure: Extrapolation to $dt-$zero of the energy for a two-dimensional quantum dot with six electrons (left) and mean energy error ($E_{min} = 20.190645 \pm 1.7 \times 10^{-4}\, au$) at $dt = 0.06$ as a function of the number of blocks for a 2DQDot6e (right). Parameters of simulation: 10^7 Monte Carlo steps with 10 % equilibration steps. $\omega = 0.926272$

Extrapolated energy to zero dt for atoms and QD

System	**Energy**, (au)
He	-2.8913
Be	-14.5039
2DQDot2e	3.0003
2DQDot6e	20.1904

Table: Energies estimated using zero-dt extrapolation.

Influence of the # MC cycles on the statistical error

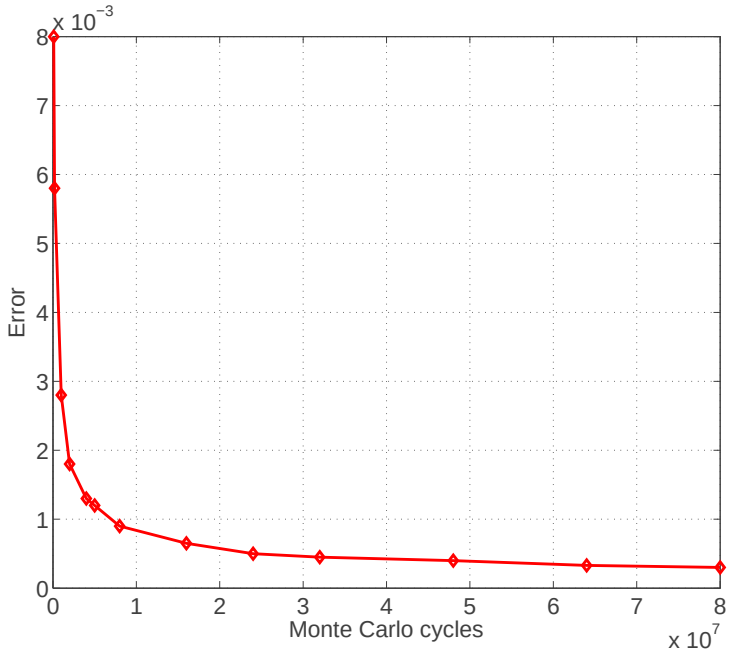


Figure: Error in the energy obtained by blocking for a Be atom as a function of the number of Monte Carlo cycles. The set up for the experiment was: $dt = 0.01$, $\alpha = 3.981$, $\beta = 0.09271$, 10 % equilibration steps by run in parallel with four processors.

Concluding remarks

What has be done?

- Prototyping of a QVMC simulator with Python.
- Extension of the Python simulator with C++.
- Development of a code for QVMC in C++ applied to atoms and quantum dots within the so-called closed shell model.
- Description of analytical expressions for validation of code.
- Proposal of an efficient algorithm to compute the analytical derivative of the energy.

...Concluding remarks [continued]

Some reflections

- What was Python good for in this thesis?
- Potential use of Python in scientific computing, and its limitations.
- What about C++ compared to Python?
- Parametric optimization of trial wave functions in QVMC.
  - Graphical method, uses and limitations.
  - Quasi-Newton method, uses and limitations.
- Energy computations with QVMC using C++.
  - Extrapolation of the energy to $zero - dt$ is expensive.
  - Observation of the acceptance to locate the region with quasilinear $energy - dt$ plot.

...Concluding remarks [continued]

Further work

- Work needed by SWIG to enhance compatibility Python/C++/legacy code, and automatize the conversion of C++/Python/C++ arrays, functions, etc.
- Development of better tutorials explaining the integration of Python arrays and functions with other languages.
- Efforts to make Python faster.
- Try alternative algorithms.
  - Updating the inverse of the Slater matrix.
  - Evaluation of quantum force and kinetic energy.
  - Sampling of core and valence electrons with different δt.
- Implementation of optimization methods adapted for handling stochastic noise.
  - Stochastic Newton-based methods.
  - Stochastic gradient approximation (SGA).
- Improve the code with functors and templates, optimize `Jastrow` class, add a *load balance checker* and a class for computing the statistical part separated from `energy` class.