

pi-qmc Documentation

Release 1.0.beta

John Shumway	Ian Galbraith	Peter MacDonald
Jonathan DuBois	Saad Khairallah	Jianheng Liu
Mark Sanger	Nikhil Monga	Dan Quach

April 13, 2015

I	Introduction	1
1	Features	3
1.1	Applications	3
2	Overview	5
3	Getting Started	7
4	Example: A Simple Harmonic Oscillator	9
4.1	Introduction: Quantum Simple Harmonic Oscillator	9
II	Preliminaries	13
5	File Formats: XML and HDF5	15
5.1	XML	15
5.2	HDF5	15
6	Science: Units, Statistical Mechanics	17
7	Testing: Unit Tests, TDD, System Integration Tests	19
7.1	Unit Tests	19
7.2	System Integration Tests	19
8	Parallel Computing	21
8.1	MPI	21
8.2	OpenMP	21
III	The Input File	23
9	Structure of the Input File	25
9.1	Example	25
10	Simulation Info: Describing the System	27
11	Actions: Describing the Dynamics	29
11.1	Kinetic Action	29
11.2	External Potentials and Fields	29
11.3	Interactions	29

11.4	Coulomb Action	29
12	Estimators: Describing the Measurements	31
12.1	List of All Estimators	31
12.2	Density Estimators	32
12.3	Dynamic Response Functions	33
13	Algorithms: Describing the Sampling Strategy	37
IV	More Tutorials	39
14	Hydrogen and Helium Atoms	41
14.1	Hydrogen Atom	41
14.2	Helium Atom	42
15	Quantum Dots	43
16	Quantum Wires	45
17	Electron Gasses	47
18	Laser Trapped Atoms	49
V	Advanced Topics	51
19	Magnetic Fields	53
20	Fermions and the Fixed Node Approximation	55
20.1	Exact Fermions	55
21	Recombination Rates in Semiconductor Nanostructures	57
VI	Appendices	59
22	Building pi-qmc	61
22.1	Quick start	61
22.2	Required libraries	61
22.3	Advanced build using multiple directories	61
22.4	Platform specific instructions	62
22.5	HPC Centers	64
23	Frequently Asked Questions	67
23.1	How to raise issues and get help	67
23.2	How to help update the FAQ	67
23.3	The Questions	67
24	Publications	69

Part I

Introduction

Features

A primary motivation of pi-qmc is to have a framework so that features developed by one student or for one research project can be used in many different contexts. Some of the many features developed for pi-qmc include:

- Bosons
- Exact fermions
- Fixed-node fermions
- Coulomb Interactions
- Linear Response Theory
- Electron-Hole recombination rates.
- Exact magnetic fields and fixed-phase.
- Multilevel sampling.
- Collective sampling.
- Free-energy sampling.
- Spin flips.

We are developing system integration tests to document and verify the status of these features.

1.1 Applications

1.1.1 Small Molecules

We have performed careful calculations on a H₂ molecule.

1.1.2 Quantum Dots

1.1.3 Quantum Point Contacts

1.1.4 Plasmas

1.1.5 Ultracold Atomic Gases

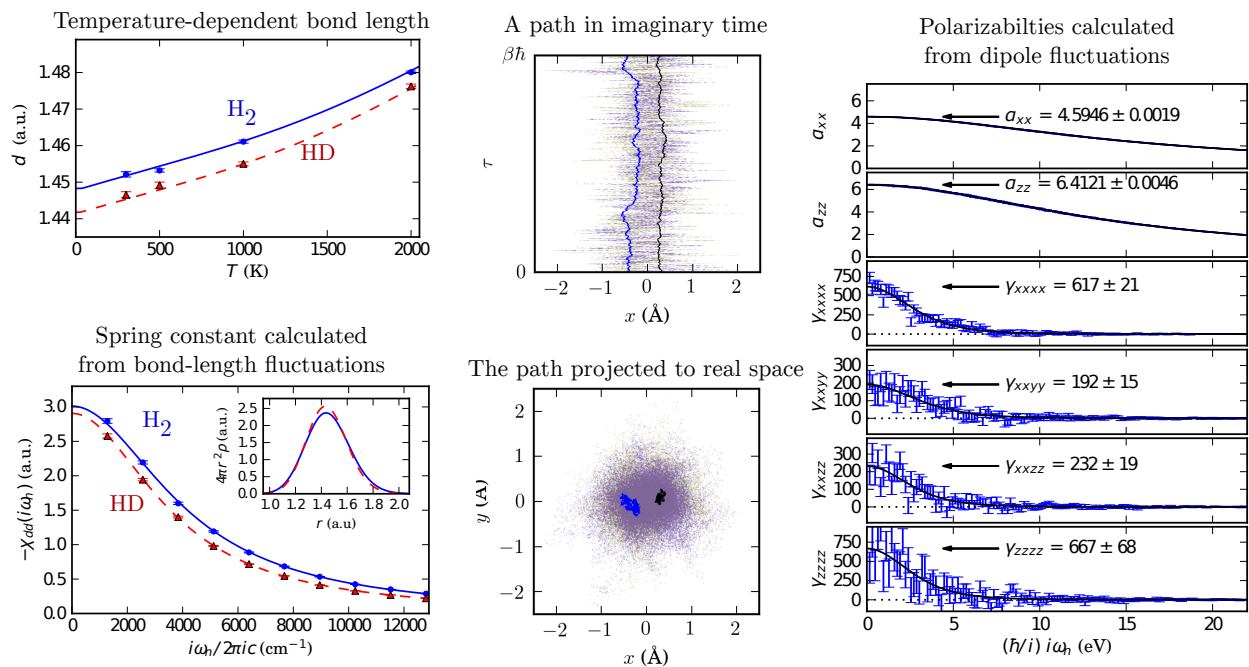


Figure 1.1: An illustrative calculation of a hydrogen molecule at 295 K. (*Top center*) A typical path in imaginary time for HD, showing low-mass electrons (faint lines), a proton (blue, left) and a heavier deuteron (black, right). (*Bottom center*) The same path, shown in real space. (*Top left*) Calculated bond length for H_2 and HD, showing zero-point expansion from the $d = 1.40 a_0$ potential minimum and thermal expansion due to molecular vibrations and rotations. (*Bottom left*) Dynamic fluctuations in the bond length give the linear-response to a stretching force (inverse spring constant) and vibrational frequency. (*Right side*) Fluctuations in the dipole give the polarizabilities α and hyperpolarizabilities γ .

Overview

This is a quantum simulation program from the Shumway Research Group, which focuses on applications to nanoscience and technology. Path integral Monte Carlo (PIMC) simulates particles (often electrons and ions) by directly sampling the canonical partition function. In the path integral formulation of quantum statistical mechanics developed by Richard Feynman, particles get represented by closed imaginary-time trajectories of length $\hbar/k_B T$. PIMC simulations are able to compute total energies, correlation functions, charge distribution, and linear response functions for thermal equilibrium. As in many quantum Monte Carlo methods, PIMC has efficient scaling with system size, often order N^2 or N^3 .

Our application, pi-qmc, is well suited for modeling conduction electrons and holes in quantum dots, quantum wires, and quantum wells. For quantum dots and wires, we often generate realistic confining potentials using *qdot-tools*. We are also testing and developing pi for ab initio calculations, but at this point only hydrogen and helium systems work well.

Getting Started

Example: A Simple Harmonic Oscillator

4.1 Introduction: Quantum Simple Harmonic Oscillator

Pi-qmc is a computer program in the language of C++ that predicts the behavior of quantum, or atomically small, particles. These particles do not always behave in the same manner as larger bodies and must be studied using unique equations. Quantum particles behave more and more like larger bodies as their energy increases.

Pi-qmc is able to easily predict the motion of these particles by summing the potential motions found using a random walk where the motions are predicted using statistical equations. Normally, the behavior of quantum systems would have to be found using a far more difficult process where the energies of each eigenstate are summed and divided by the number of possible states in order to normalize the function and find the average energy. A simple example of a quantum system where this process could be applied is the Quantum Harmonic Oscillator (QHC). The mathematics behind this process start with calculating the total energy of the system.

4.1.1 Background

The total energy, or Hamiltonian, of a Quantum Harmonic Oscillator is represented by the function

$$\hat{H} = \frac{\hat{p}^2}{2m} + (1/2)m\omega^2\hat{x}^2 \quad (4.1)$$

The kinetic energy of the quantum harmonic oscillator is represented by the first term while the stored energy is represented by the second term. This may seem complex at first glance, but upon further inspection, it becomes apparent that this Hamiltonian is nearly identical to Hamiltonian of a regular harmonic oscillator which is represented by the function

$$E = \frac{1}{2}mv^2 + \frac{1}{2}kx^2 \quad (4.2)$$

In the Hamiltonian of a Quantum Harmonic Oscillator, the mass multiplied by the velocity can be substituted in place of the momentum and the first term can be rearranged to resemble the first term of the regular harmonic oscillator's Hamiltonian. Similarly, the square root of the spring constant divided by the mass can be substituted in for the wavelength and the second term can be rearranged to resemble the second term of the regular harmonic oscillator. After both terms are rearranged, it becomes clear that the two Hamiltonians are identical.

4.1.2 Eigenstates of the Simple Harmonic Oscillator

The Eigenstate of Quantum Harmonic Oscillator is the energy level it occupies. These energy levels are represented by whole numbers ($n=0, n=1, n=2, \dots$) and are separated by consistent amounts energy that increase with each

consecutive energy level. As the eigenstate increases, the period of the quantum harmonic oscillator decreases and the wave behaves differently. The energy of the eigenstate, or eigenvalue can be found using the function

$$E_n = \hbar\omega\left(n + \frac{1}{2}\right) \quad (4.3)$$

According to Heisenberg's Uncertainty Principle, the momentum and the position of a quantum particle cannot be known at the same time. Therefore, we can surmise that the energy of the eigenvalue will never be equal to zero since the no energy means no movement which, in turn, means that both the momentum and the position of the quantum particle will be known. When we substitute in a 0 for the eigenvalue, we find that the function simplifies to

$$E_0 = \frac{1}{2}\hbar\omega \quad (4.4)$$

which represents the lowest possible energy of the Eigenstate and agrees with Heisenberg's Uncertainty principle. When the eigenstate is multiplied by the Hamiltonian, the resulting value is the eigenvalue as shown in the function

$$\hat{H}\psi(x) = E\psi(x) \quad (4.5)$$

4.1.3 The Classical Partition Function

4.1.4 The Quantum Mechanical Partition Function

In one dimension, the partition function of the simple harmonic oscillator is

$$Z = \sum_{n=0}^{\infty} e^{-\beta\hbar\omega(n+\frac{1}{2})} = \left[2 \sinh\left(\frac{\hbar\omega}{2k_B T}\right)\right]^{-1} \quad (4.6)$$

For N oscillators in D dimensions, the partition function is

$$Z = \left[2 \sinh\left(\frac{\hbar\omega}{2k_B T}\right)\right]^{-ND} \quad (4.7)$$

The Helmholtz free energy is

$$F = E - TS = -k_B T \ln(Z) = NDk_B T \ln\left[2 \sinh\left(\frac{\hbar\omega}{2k_B T}\right)\right]. \quad (4.8)$$

The total energy is

$$E = -\frac{d}{d\beta} \ln(Z) = ND \frac{\hbar\omega}{2} \coth\left(\frac{\hbar\omega}{2k_B T}\right). \quad (4.9)$$

4.1.5 Normalizing the Function

The function is now normalized by dividing by the partition function

$$z = \sum_{n=0}^{\infty} e^{-\frac{E_n}{k_B T}} \quad (4.10)$$

This function can be simplified to the form

$$z = \frac{1}{2 \sinh\left(\frac{\beta\hbar\omega}{2}\right)} \quad (4.11)$$

where

$$\beta = \frac{1}{k_B T} \quad (4.12)$$

Our final equation is now

$$P_n = \frac{e^{-\frac{E_n}{k_B T}}}{z} \quad (4.13)$$

4.1.6 The Density Matrix

For one particle in a harmonic confining potential in three dimensions, the imaginary time propagator is

$$K(\mathbf{r}, \mathbf{r}'; \tau) = \left(\frac{m\omega}{2\pi\hbar \sinh \omega\tau} \right)^3 \exp \left(-\frac{m\omega((r^2 + r'^2) \cosh \omega\tau - 2\mathbf{r} \cdot \mathbf{r}')}{2\hbar \sinh \omega\tau} \right) \quad (4.14)$$

The diagonal of the density matrix is the probability density,

$$\begin{aligned} \rho(\mathbf{r}) &= K(\mathbf{r}, \mathbf{r}; \tau) \\ &= \left(\frac{m\omega}{2\pi\hbar \sinh \omega\tau} \right)^3 \exp \left(-\frac{m\omega((r^2 + r'^2) \cosh \omega\tau - 2\mathbf{r} \cdot \mathbf{r}')}{2\hbar \sinh \omega\tau} \right) \end{aligned} \quad (4.15)$$

4.1.7 Simulating a Simple Harmonic Oscillator

4.1.8 Setting up the Input file

You can simulate a simple harmonic oscillator with the following input file, which we copy from the test/system/sho/ directory.

```

1  <?xml version="1.0"?>
2  <Simulation>
3      <SuperCell a="20.0" x="1" y="1" z="1"/>
4      <Species name="e" count="1" mass="1" charge="-1"/>
5      <Temperature value="1.0" nslice="32"/>
6      <Action>
7          <SpringAction/>
8          <SHOAction omega="0.5 Ha"/>
9      </Action>
10     <Estimators>
11         <ThermalEnergyEstimator/>
12     </Estimators>
13     <PIMC>
14         <RandomGenerator/>
15         <!-- Thermalize -->
16         <Loop nrepeat="100">
17             <ChooseSection nlevel="5">
18                 <Sample npart="1" mover="Free" species="e"/>
19             </ChooseSection>
20         </Loop>
21         <!-- Sample data -->
22         <Loop nrepeat="500">
23             <Loop nrepeat="200">
24                 <ChooseSection nlevel="5">
25                     <Sample npart="1" mover="Free" species="e"/>
26                 </ChooseSection>
27                 <Measure estimator="all"/>
28             </Loop>
29             <Collect estimator="all"/>
30         </Loop>
31     </PIMC>
32 </Simulation>

```

Here we have set the temperature $k_B T = 1.0$ (line 5), and we are simulating a harmonic oscillator with $\hbar\omega = 0.5$ Hartrees (line 8). From Eq. (4.9), the energy of this oscillator in three dimensions should be $\frac{3}{2} \coth \frac{1}{4} \approx 3.06224$.

4.1.9 Calculating the Energy

4.1.10 Calculating the Density

4.1.11 Calculating the Polarizability

Part II

Preliminaries

File Formats: XML and HDF5

5.1 XML

5.2 HDF5

Science: Units, Statistical Mechanics

Testing: Unit Tests, TDD, System Integration Tests

7.1 Unit Tests

Unit testing uses [GoogleTest](#). The unit tests are in the *pi-qmc/unit-test/* subdirectory, which mirrors the structure of the *pi-qmc/src* directory.

Each unit test should execute in a few milliseconds, so that the entire suite can be run in a few seconds.

Right now the unit tests are only included in the cmake build.

7.2 System Integration Tests

System integration tests are run with python scripts. We use the python [unittest](#) module to organize the test cases. These system tests can be run using [nosetests](#), like

```
nosetests -v --rednose
```

System integration tests run the *pi-qmc* executable on real test systems, and can take a few minutes to run.

Parallel Computing

8.1 MPI

8.2 OpenMP

Part III

The Input File

Structure of the Input File

9.1 Example

The pimc.xml files are xml files. The root tag contains six elements. Order doesn't matter to the parser, but we always order them in our pimc.xml input files (extended whitespace indicates omitted details):

```
<?xml version="1.0"?>
<Simulation>
  <SuperCell    />
  <Species      />
  <Temperature   />
  <Action>

  </Action>
  <Estimators>

  </Estimators>
  <PIMC    >

  </PIMC>
</Simulation>
```

As you can see, the file is grouped by six tags that describe the PIMC simulation:

- SuperCell
- Species (this tag is repeated for each species)
- Temperature
- Action (contains all the ActionTags).
- Estimators (contains all the EstimatorTags).
- PIMC (specifies the parallelism and contains the algorithm tree as PIMCTags).

All these tags are required. Notes: The code uses atomic units (Ha, a0), but has some unit conversion capabilities.

9.1.1 SuperCell

The SuperCell tag is parsed in SimInfoParser.cc.

9.1.2 Species

The Species tag is parsed in SimInfoParser.cc.

9.1.3 Temperature

The Temperature tag is parsed in SimInfoParser.cc.

9.1.4 Actions

In path integrals, the action plays the same role as the Hamiltonian plays in Schrödinger's equation. The action section of a pimc.xml file is denoted

```
<Actions> </Actions>
```

and contains any number of ActionTags. Most simulations contain at least SpringAction for free-particle kinetic action, but in some cases even that may be replaced. Action tags are parsed in ActionParser.cc.

and contains any number of ActionTags. Most simulations contain at least SpringAction for free-particle kinetic action, but in some cases even that may be replaced. Action tags are parsed in ActionParser.cc.

9.1.5 Estimators

Estimators are the mathematical and algorithmic tools to extract physical information from the path integral. The estimator section is denoted

```
<Estimators> </Estimators>
```

and may contain any number of EstimatorTags. Estimator tags are parsed in EstimatorParser.cc.

9.1.6 PIMC Commands

The PIMC commands are denoted

```
<PIMC Commands> </PIMC Commands>
```

and may contain any number of sequential or nested PIMCTags. PIMC commands describe the how the paths are sampled, when measurements are performed, and when data is written to disk. PIMC tags are parsed in PIMCParser.cc.

Simulation Info: Describing the System

Actions: Describing the Dynamics

11.1 Kinetic Action

11.2 External Potentials and Fields

11.3 Interactions

11.4 Coulomb Action

Estimators: Describing the Measurements

12.1 List of All Estimators

- **AngularMomentumEstimator** (*scalar/angular-momentum*) Measure the momentum in two-dimensions in a magnetic field.
- **BondLengthEstimator** (*scalar-length/bond-length*) Measure the length of a molecular bond.
- **BoxEstimator** (*scalar/in-box*) Measure the fraction of time a particle is in a boxed range.
- **CoounductanceEstimator** (*dynamic-array/conductance*) Measure dynamic correlations of current for polarizability.
- **CoounductivityEstimator** (*dynamic-array/conductivity*) Measure dynamic correlations of current density for conductivity.
- **CoounductivityEstimator2D** (*dynamic-array/conductivity-2D*) Measure dynamic correlations of current density for conductivity in two dimensions.
- **CoulombEnergyEstimator** (*scalar-energy/coulomb-energy*) Measure the Coulomb energy using the thermodynamic estimator.
- **CountCountEstimator** (*dynamic-array/count-count*) Measure the dynamic correlation of occupations on a grid.
- **DensCountEstimator** (*dynamic-array/density-count*) Measure the dynamic correlation of density with occupations on a grid. (Not implemented.)
- **DensDensEstimator** (*dynamic-array/density-density*) Measure dynamic correlations of density.
- **DensityEstimator** (*array/density*) Measure particle density.
- **DiamagneticEstimator** (*scalar/diamagnetism*) Measure diamagnetic susceptibility using path area.
- **DipoleEstimator** (*scalar-length/dipole*) Measure the electric dipole of a localized, non-period system.
- **DynamicPCFEstimator** (*dynamic-array/pair-correlation*) Measure the dynamic correlations of a pair correlation function.
- **EIndEstimator** (*dynamic-array/induced-e-field*) Measure the induced electric field correction for the ConductivityEstimator.
- **EMARateEstimator** (*scalar/ema-recombination-rate*) Measure electron and hole recombination rate.
- **FreeEnergyEstimator** (*histogram/free-energy*) Measure relative free energies for different actions.
- **FrequencyEstimator** (*dynamic-scalar/frequency*) Measure relative free energies for different actions.

- **JEstimator** (*array/singlet-triplet*) Measure singlet-triplet splitting with magnetic field dependence.
- **PairCFEstimator** (*array/pair-correlation*) Measure pair correlation functions.
- **PermutationEstimator** (*histogram/permutation*) Measure distribution of permutations.
- **PositionEstimator** (*scalar-length/position*) Measure average position of a particle.
- **SpinChargeEstimator** (*dynamic-array/spin-charge*) Measure dynamic correlation of currents for different species.
- **ThermoEnergyEstimator** (*scalar-energy/thermo-energy*) Measure the total energy using the thermodynamic estimator.
- **VIndEstimator** (*dynamic-scalar/induced-voltage*) Measure the induced voltage correction for the ConductivityEstimator.
- **VirialEnergyEstimator** (*scalar-energy/virial-energy*) Measure the total energy using the virial estimator.
- **WindingEstimator** (*histogram/winding*) Measure distribution of periodic windings.
- **ZeroVarDensityEstimator** (*array/zero-var-density*) Measure point-contact density with zero-variance within pair approximation.

12.2 Density Estimators

12.2.1 Real-space on a rectangular grid

The simplest way to collect the density is to create a rectangular array of bins and histogram the beads of the paths. For example, a grid defined by

- `xmin, xmax, nx`
- `ymin, ymax, ny`
- `zmin, zmax, nz`

has `nx*ny*nz` rectangular bins, each with dimension `dx = (xmax-xmin)/nx` by `dy = (ymax-ymin)/ny` by `dz = (zmax-zmin)/nz`. The bin with indices `(i, j, k)` is centered at `(xmin+(i+0.5)*dx, ymin+(j+0.5)*dy, zmin+(k+0.5)*dz)`.

Each time a measurement is made, the position of all `npart*nslice` beads is checked, where `npart` is the number of particles of the species whose density is being measured. If a bead is inside one of the bins, that bin is increased by `1./nslice`. If all beads lie in the grid bins, then the bins sum to `npart`. Otherwise the total is less than `npart`; this can happen if the grid dimensions do not fill the simulation supercell. To convert the measurement to density, divide by the volume of a bin, `dx * dy * dz`.

Sample pimc.xml code

```
<DensityEstimator>
  <Cartesian dir="x" nbin="500" min="-250 nm" max="250 nm"/>
  <Cartesian dir="y" nbin="500" min="-255 nm" max="250 nm"/>
</DensityEstimator>
```

12.2.2 Real-space on arbitrary grids

12.2.3 Density in k-space

Since pi-qmc uses a position basis, we often collect density fluctuations in real space. However, most textbook descriptions of density fluctuations are in k-space, and results for homogeneous systems are often best represented in k-space. Here we give a brief summary of common definitions for pedagogical purposes. For simplicity we write all formulas for spinless particles.

The dimensionless Fourier transform of the density operator is (Eqs. 1.11 and 1.66 of Giuliani and Vignale)

$$\begin{aligned} n_{\mathbf{k}} &= \sum_j e^{-i\mathbf{k}\cdot\mathbf{r}_j} \\ &= \sum_{\mathbf{q}} a_{\mathbf{q}-\mathbf{k}}^\dagger a_{\mathbf{q}}. \end{aligned}$$

Note that $n_0 = N$, the total number of particles.

The pi code does not presently calculate this expectation value. If it is implemented in the future, it should return a complex expectation value for each k-vector. The imaginary part of this estimator will be zero for systems with inversion symmetry about the origin.

Homogeneous systems, such as liquid helium or the electron gas, will have $\langle n_{\mathbf{k}} \rangle = 0$ for all $\mathbf{k} \neq 0$. In those cases, it is better to calculate the static structure factor (see *static structure factor*).

12.3 Dynamic Response Functions

We define a dynamic correlation function as

$$\chi_{AB}(\tau) = -(1/\hbar)\langle A(\tau)B(0) \rangle, \quad (12.1)$$

where A and B are operators.

12.3.1 Density-density response

Since pi-qmc uses a position basis, we often collect density fluctuations in real space. However, most textbook descriptions of density fluctuations are in k-space, and results for homogeneous systems are often best represented in k-space. Here we give a brief summary of common definitions for pedagogical purposes. For simplicity we write all formulas for spinless particles.

The dimensionless Fourier transform of the density operator is (Eqs. 1.11 and 1.66 of Giuliani and Vignale)

$$\begin{aligned} n_{\mathbf{q}} &= \sum_j e^{-i\mathbf{q}\cdot\mathbf{r}_j} \\ &= \sum_{\mathbf{k}} a_{\mathbf{k}-\mathbf{q}}^\dagger a_{\mathbf{k}}. \end{aligned} \quad (12.2)$$

Note that $n_0 = N$, the total number of particles. To get back to real-space density use

$$n(\mathbf{r}) = \frac{1}{V} \sum_{\mathbf{q}} n_{\mathbf{q}} e^{i\mathbf{q}\cdot\mathbf{r}}. \quad (12.3)$$

For each of these, we define frequency-dependent density operators,

$$n(\mathbf{r}, i\omega_n) = \int_0^{\beta\hbar} n(\mathbf{r}, \tau) e^{i\omega_n \tau} d\tau, \quad (12.4)$$

and

$$n_{\mathbf{q}}(i\omega_n) = \int_0^{\beta\hbar} n_{\mathbf{q}} e^{i\omega_n \tau} d\tau, \quad (12.5)$$

where $i\omega_n = 2\pi i n k_B T / \hbar$ are the Matsubara frequencies. Within the pi-qmc code, these frequency-dependent densities are easily calculated with fast Fourier transforms, which are most efficient when the number of slices is a power of two.

Real-space response

The imaginary-frequency response of the density to an external perturbation is given by (Ch 3.3 of Giuliani and Vignale),

$$\delta n(\mathbf{r}, i\omega_n) = \int d\mathbf{r}' \chi_{nn}(\mathbf{r}, \mathbf{r}', i\omega_n) V_{\text{ext}}(\mathbf{r}', i\omega_n). \quad (12.6)$$

In k-space this takes the convenient form,

$$\delta n(\mathbf{q}, i\omega_n) = \sum_{\mathbf{q}'} \chi_{nn}(\mathbf{q}, \mathbf{q}', i\omega_n) V_{\text{ext}}(\mathbf{q}', i\omega_n). \quad (12.7)$$

where the external potential in k-space satisfies

$$V_{\text{ext}}(\mathbf{r}') = \frac{1}{V} \sum_{\mathbf{q}'} V_{\text{ext}}(\mathbf{q}') e^{i\mathbf{q}' \cdot \mathbf{r}'}, \quad (12.8)$$

and

$$V_{\text{ext}}(\mathbf{q}') = \int d\mathbf{r}' e^{-i\mathbf{q}' \cdot \mathbf{r}'} V_{\text{ext}}(\mathbf{r}').$$

These response functions are related to imaginary-frequency dynamic correlation functions,

$$\chi_{nn}(\mathbf{r}, \mathbf{r}', i\omega_n) = -\frac{1}{\beta\hbar^2} \langle n(\mathbf{r}, i\omega_n) n(\mathbf{r}', -i\omega_n) \rangle,$$

and

$$\chi_{nn}(\mathbf{q}, \mathbf{q}', i\omega_n) = -\frac{1}{\beta\hbar^2 V} \langle n_{\mathbf{q}}(i\omega_n) n_{-\mathbf{q}'}(-i\omega_n) \rangle.$$

For a homogeneous system,

$$\chi_{nn}(\mathbf{q}, \mathbf{q}', i\omega_n) = -\frac{1}{\beta\hbar^2 V} \langle n_{\mathbf{q}}(i\omega_n) n_{-\mathbf{q}}(-i\omega_n) \rangle \delta_{\mathbf{q}\mathbf{q}'}.$$

Structure factor

The dynamic structure factor $S(\mathbf{k}, i\omega_n)$ measures the density response of the system,

$$S(\mathbf{k}, i\omega_n) = -\frac{V}{\hbar N} \chi_{nn}(\mathbf{k}, \mathbf{k}, i\omega_n) \quad (12.9)$$

The static structure factor is defined for equal time, not for $\omega_n \rightarrow 0$,

$$S(\mathbf{k}) = \frac{1}{N} \langle n_{\mathbf{k}}(\tau = 0) n_{-\mathbf{k}}(\tau = 0) \rangle.$$

In terms of $\chi_{nn}(\mathbf{q}, \mathbf{q}', i\omega)$, the static structure factor is given by (*prefactor is wrong*)

$$S(\mathbf{k}) = -\frac{V}{\hbar N} \sum_n \omega_n \chi_{nn}(\mathbf{k}, \mathbf{k}, i\omega_n) e^{-i\omega_n \tau}. \quad (12.10)$$

Polarizability

Algorithms: Describing the Sampling Strategy

Part IV

More Tutorials

Hydrogen and Helium Atoms

14.1 Hydrogen Atom

```

1  <?xml version="1.0"?>
2  <Simulation>
3    <SuperCell a="25 A" x="1" y="1" z="1"/>
4    <Species name="eup" count="1" mass="1 m_e" charge="-1"/>
5    <Species name="H" count="1" mass="1.006780 amu" charge="+1"/>
6    <Temperature value="10000 K" nslice="512"/>
7    <Action>
8      <SpringAction/>
9      <CoulombAction norder="3" rmin="0.001" rmax="5." ngridPoints="1000"/>
10   </Action>
11   <Estimators>
12     <ThermalEnergyEstimator unit="eV"/>
13     <VirialEnergyEstimator unit="eV" nwindow="500"/>
14     <CoulombEnergyEstimator unit="eV"/>
15   </Estimators>
16   <PIMC>
17     <RandomGenerator/>
18     <SetCubicLattice nx="2" ny="2" nz="2" a="1."/>
19     <!-- Thermalize -->
20     <Loop nrepeat="10000">
21       <ChooseSection nlevel="6">
22         <Sample npart="1" mover="Free" species="H"/>
23         <Sample npart="1" mover="Free" species="eup"/>
24       </ChooseSection>
25     </Loop>
26     <!-- Sample data -->
27     <Loop nrepeat="100">
28       <Loop nrepeat="300">
29         <ChooseSection nlevel="7">
30           <Sample npart="1" mover="Free" species="H"/>
31           <Sample npart="1" mover="Free" species="eup"/>
32         </ChooseSection>
33         <Loop nrepeat="3">
34           <ChooseSection nlevel="6">
35             <Sample npart="1" mover="Free" species="eup"/>
36           </ChooseSection>
37         </Loop>
38         <Loop nrepeat="5">
39           <ChooseSection nlevel="5">

```

```
40         <Sample npart="1" mover="Free" species="eup"/>
41     </ChooseSection>
42 </Loop>
43     <Measure estimator="all"/>
44 </Loop>
45     <Collect estimator="all"/>
46 </Loop>
47 </PIMC>
48 </Simulation>
```

14.2 Helium Atom

Quantum Dots

Quantum Wires

Electron Gasses

Laser Trapped Atoms

Part V

Advanced Topics

Magnetic Fields

Fermions and the Fixed Node Approximation

20.1 Exact Fermions

20.1.1 Example: Two fermions in a one-dimensional simple harmonic oscillator

For two distinguishable particles in a harmonic confining potential in three dimensions, the imaginary time propagator is

$$K(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}'_1, \mathbf{r}'_2; \tau) = \left(\frac{m\omega}{2\pi\hbar \sinh \omega\tau} \right)^3 \exp \left(-\frac{m\omega((r_1^2 + r_2^2 + r_1'^2 + r_2'^2) \cosh \omega\tau - 2\mathbf{r}_1 \cdot \mathbf{r}'_1 - 2\mathbf{r}_2 \cdot \mathbf{r}'_2)}{2\hbar \sinh \omega\tau} \right) \quad (20.1)$$

The partition function is the trace of the propagator for $\tau = \beta\hbar$,

$$\begin{aligned} Z &= \text{tr } K \\ &= \int d\mathbf{r}_1^3 \int d\mathbf{r}_2^3 K(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_1, \mathbf{r}_2; \beta\hbar) \\ &= \left(\frac{m\omega}{2\pi\hbar \sinh \beta\hbar\omega} \right)^3 \int d\mathbf{r}_1^3 \int d\mathbf{r}_2^3 \exp \left(-\frac{2m\omega(\cosh \beta\hbar\omega - 1)(r_1^2 + r_2^2)}{2\hbar \sinh \omega\tau} \right) \\ &= (2(\cosh \beta\hbar\omega - 1))^{-3} \\ &= \left[2 \sinh \left(\frac{\hbar\omega}{2k_B T} \right) \right]^{-6} \end{aligned} \quad (20.2)$$

For identical particles, we need to symmetrize the states for fermions, or antisymmetrize the states for bosons. The trace of the permuted propagator is,

$$\begin{aligned} Z_P &= \text{tr } PK \\ &= \int d\mathbf{r}_1^3 \int d\mathbf{r}_2^3 K(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_2, \mathbf{r}_1; \beta\hbar) \\ &= \left(\frac{m\omega}{2\pi\hbar \sinh \beta\hbar\omega} \right)^3 \int d\mathbf{r}_1^3 \int d\mathbf{r}_2^3 \exp \left(-\frac{2m\omega(\cosh \beta\hbar\omega(r_1^2 + r_2^2) - \mathbf{r}_1 \cdot \mathbf{r}_2)}{2\hbar \sinh \omega\tau} \right) \end{aligned}$$

Next we change coordinates to $\mathbf{R} = \frac{1}{2}(\mathbf{r}_1 + \mathbf{r}_2)$ and $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$. Then $r_1^2 + r_2^2 = R^2 + r^2/2$ and $\mathbf{r}_1 \cdot \mathbf{r}_2 = R^2 - r^2/2$, and we find,

$$Z_P = \left[2 \cosh \left(\frac{\hbar\omega}{2k_B T} \right) \sinh \left(\frac{\hbar\omega}{2k_B T} \right) \right]^{-3} \quad (20.3)$$

Recombination Rates in Semiconductor Nanostructures

Part VI

Appendices

Building pi-qmc

22.1 Quick start

The easiest way to build is to use:

```
./configure make
```

For a parallel build

```
./configure --enable-mpi MPICXX=mpic++ MPICC=mpicc MPIF77=mpif77
```

where you should use the names of your MPI enabled compilers.

You can also build for different numbers of physical dimensions (default is NDIM=3)

```
./configure --with-ndim=2
```

22.2 Required libraries

We use the following libraries in the pi code:

- libxml2
- blitz++
- hdf5
- fftw3
- BLAS / LAPACK
- gsl

22.3 Advanced build using multiple directories

In research, we often want different versions of the executables, for example, versions with and without MPI, or versions compiled for two-dimensional systems. To accomplish this, we make a `pibuilds` directory beside our svn checkout directory (`pi` or `pi-qmc`). We then make empty subdirectories for each build, for example `ndim2mpi` for a two dimensional MPI version. A typical directory structure is:

```
codes/  
  pi-qmc/  
    configure  
    src/  
    lib/  
  pibuilds/  
    ndim1/  
    ndim2/  
    ndim3/  
    ndim1mpi/  
    ndim2mpi/  
    ndim3mpi/  
    debug/
```

To build, go into the empty build directory,

```
cd ~/codes/pibuilds/ndim2mpi
```

Then run the configure script with the desired options

```
../../configure --with-ndim=2 --enable-mpi
```

You will probably want more configure options; see the platform specific instructions below for some examples.

Then, make the code in that directory,

```
make -j2
```

For convenience, you can make a soft link to the executable

```
ln -sf ~/codes/pibuilds/ndim3mpi ~/bin/pi2Dmpi
```

22.4 Platform specific instructions

22.4.1 Mac OS X

All the dependencies are available through [<http://www.macports.org/> macports]. It is also handy to install the latest gcc compilers (with gfortran), openmpi, and python utilities for data analysis and plotting.

```
$ port installed  
  libxml2 @2.7.3_0 (active)  
  blitz @0.9_0 (active)  
  hdf5-18 @1.8.3_0 (active)  
  gsl @1.12_0 (active)  
  fftw-3 @3.2.2_0 (active)  
  
  gcc44 @4.4.0_0 (active)  
  
  python26 @2.6.2_3 (active)  
  py26-numpy @1.3.0_0 (active)  
  py26-ipython @0.9.1_0+scientific (active)  
  py26-scipy @0.7.0_0+gcc44 (active)  
  py26-tables @2.1_0 (active)
```

A bash script to download the pi-qmc source from git hub, compile on Mac OS X, and run all tests is included in doc/deploy/macosx/start.sh:

```
#!/bin/bash

git clone git@github.com:phys-tools/pi-qmc.git

mkdir pibuild
cd pibuild
export CC=gcc-mp-4.7
export CXX=g++-mp-4.7
export CXXFLAGS="-O3 -g -Wall -ffast-math -march=native"
export CXXFLAGS+=" -ftree-vectorize -fomit-frame-pointer -pipe"
cmake ../pi-qmc
make -j2

echo; echo " Running unit tests..."; echo
make -j2 unittest
bin/unittest_pi

cd test/system
echo; echo "Running system integration tests..."; echo
nosetests-2.7 -v --rednose --with-xunit
cd ../../
```

The following configure works well on an intel mac:

```
../pi/configure CXX=g++-mp-4.7 CC=gcc-mp-4.7 \
  CXXFLAGS="-O3 -g -Wall -ffast-math -ftree-vectorize \
  -march=native -fomit-frame-pointer -pipe" \
  F77=gfortran-mp-4.7
```

or, for an MPI enabled build,

```
../pi/configure --enable-mpi CXX=g++-mp-4.4 CC=gcc-mp-4.4 F77=gfortran-mp-4.4 \
  MPICC=openmpicc MPICXX=openmpicxx MPIF77=openmpif77 \
  CXXFLAGS="-O3 -g -Wall -ffast-math -ftree-vectorize \
  -march=native -fomit-frame-pointer -pipe"
```

On a G5 mac, try:

```
../pi/configure --with-ndim=3 F77=gfortran-mp-4.4 CC=gcc-mp-4.4 CXX=g++-mp-4.4 \
  CXXFLAGS="-g -O3 -ffast-math -ftree-vectorize -maltivec -mpowerpc-gpopt \
  -mpowerpc64 falign-functions=32 -falign-labels=32 -falign-loops=32 -falign-jumps=32 -funroll-loops"
```

or, for an MPI enabled build,

```
../pi/configure --with-ndim=3 --enable-mpi \
  CXXFLAGS="-g -O3 -ffast-math -ftree-vectorize -maltivec -mpowerpc-gpopt \
  -mpowerpc64 falign-functions=32 -falign-labels=32 -falign-loops=32 -falign-jumps=32 -funroll-loops"
  F77=gfortran-mp-4.4 CC=gcc-mp-4.4 CXX=g++-mp-4.4 MPICC=openmpicc MPICXX=openmpicxx MPIF77=openmpif77
```

22.4.2 Linux (CentOS 5.3)

You can download dependencies using yum. First, you may need to add access to the fedora [http://fedoraproject.org/wiki/EPEL Extra Packages for Enterprise Linux (EPEL)].

```
sudo rpm -Uvh http://download.fedora.redhat.com/pub/epel/5/i386/epel-release-5-3.noarch.rpm
```

Then install the required packages for `_pi_`. (You probably want to compile atlas yourself to get automatic performance tuning for your hardware, but the yum install will work if you are impatient.) Note: replace `x86_64` with `i386` if you are on a 32 bit machine.

```
sudo yum install libxml2-devel-versionXXX.x86_64 (here I don't know the correct version)
sudo yum install blitz-devel.x86_64
sudo yum install fftw3-devel.x86_64
sudo yum install hdf5-devel.x86_64
sudo yum install atlas-sse3-devel.x86_64
sudo yum install lapack-devel.x86_64
sudo yum install gsl-devel.x86_64
```

It is useful to install the gcc 4.3 compilers.

```
sudo yum install gcc43.x86_64
sudo yum install gcc43-c++.x86_64
sudo yum install gcc43-gfortran.x86_64
```

Also, you will want an MPI implementation if you want to run in parallel,

```
sudo yum install openmpi-devel.x86_64
```

The openmpi package will require that you run `mpi-selector` and open a new terminal to get the executables. Use the `mpi-selector --list` option to see what is available, then set a system-wide default.

```
sudo mpi-selector --system --set openmpi-1.2.7-gcc-x86_64
```

When you configure pi, you will probably need to specify the location of your BLAS and LAPACK routines,

```
../../pi/configure CXX=g++43 CC=gcc43 F77=gfortran43 CXXFLAGS=\
"-g -O3 -ffast-math -ftree-vectorize -march=native -fomit-frame-pointer -pipe"\
--with-blas="-L/usr/lib64/atlas -llapack -lf77blas"
```

For mpi, just add `--enable-mpi`.

For the python analysis utilities, you'll want to install ipython and matplotlib.

```
sudo yum install python-matplotlib
sudo yum install ipython
sudo yum install scipy
```

The python package `pytables` for reading HDF5 files is also required for the analysis scripts, but it is not available through yum, so you'll have to download it and install it yourself.

22.5 HPC Centers

22.5.1 ASU Fulton: saguaro

For a serial build in two dimensions,

```
../../pi/configure --with-ndim=2 --enable-sprng CXX=icpc CC=icc CXXFLAGS="-O3 -xP -ipo" --with-blas="-L$MKL_LIB -lmkl_lapack -lmkl_intel_lp64 -lmkl_sequential -lmkl_core" F77=ifort AR="xild -lib"
```

or for a parallel version,

```
../../pi/configure --with-ndim=2 --enable-sprng --enable-mpi MPICC=mpicc MPICXX=mpicxx CXX=icpc CC=icc F77=ifort CXXFLAGS="-O3 -xP -ipo" AR="xild -lib" --with-blas="-L$MKL_LIB -lmkl_lapack -lmkl_intel_lp64 -lmkl_sequential -lmkl_core"
```

Omit the `--enable-sprng` option if you do not have the SPRNG library.

22.5.2 LONI-LSU: queenbee

You need to add some lines to your `.soft` file to include some required libraries,

```
#My additions (CPATH mimics -I include directories).
CPATH += /usr/local/packages/hdf5-1.8.1-intel10.1/include
+gsl-1.9-intel10.1
+sprng4-mvapich-1.1-intel-10.1
+fftw-3.1.2-intel10.1
CPATH += /usr/local/packages/fftw-3.1.2-intel10.1/include
+intel-mkl
CPPFLAGS += -DMPICH_IGNORE_CXX_SEEK
```

For an MPI build, use,

```
../pi/configure --with-ndim=3 --enable-mpi MPICC=mpicc MPICXX=mpicxx \
CXX=icpc CC=icc F77=ifort AR="xild -lib" CXXFLAGS="-O3 -xP -ipo" \
--with-blas="-lmkl_lapack -lmkl_intel_lp64 -lmkl_sequential -lmkl_core"
```

22.5.3 NCSA: abe

You need to add some lines to your `.soft` file to include some required libraries,

```
#My additions (CPATH mimics -I include directories).
+libxml2-2.6.29
+libxml2
+intel-mkl
+gsl-intel
+hdf5-1.8.2
CPATH += /usr/apps/hdf/hdf5/v182/include
LD_LIBRARY_PATH += /usr/apps/hdf/zip/lib
+fftw-3.1-intel
LD_LIBRARY_PATH += /usr/apps/math/fftw/fftw-3.1.2/intel10/lib
CPATH += /usr/apps/math/fftw/fftw-3.1.2/intel10/include
+intel-mkl
CPPFLAGS = "${CPPFLAGS} -DMPICH_IGNORE_CXX_SEEK"
Also have blitz installed locally with --prefix=(your dir choice)
```

For an MPI build, use,

```
../pi/configure --with-ndim=3 --enable-mpi MPICC=mpicc MPICXX=mpicxx CXX=icpc CC=icc CXXFLAGS="-O3 -xP -ipo" LD_FLAGS="-lsz" --with-blas="-lmkl_lapack -lmkl_intel_lp64 -lmkl_sequential -lmkl_core" F77=ifort AR="xild -lib"
```

22.5.4 TACC: Ranger

22.5.5 Cornell CNF: nanolab

The svn client wasn't working for me, so I built one in my `~/packages/bin` directory. You need to specify the most recent C++ and Fortran compilers by including the following in your `.bash_profile`,

```
# Version 10 compilers
source /opt/intel/cc/10.1.017/bin/iccvars.sh
source /opt/intel/fc/10.1.017/bin/ifortvars.sh
source /opt/intel/idb/10.1.017/bin/idbvars.sh
source /opt/intel/mkl/10.0.4.023/tools/environment/mklvars32.sh
```

Also, make sure that `/usr/lam-7.4.1_intelv10/bin` is in your path to get the correct MPI compilers.

You need to build blitz (again, in my `~/packages` directory). For a serial pi build,

```
../../pi/configure --with-ndim=3 CXX=icpc CC=icc CXXFLAGS="-O3 -ipo" \  
--with-blas="-Wl,-rpath,$MKLROOT/lib/32 -L/opt$MKLROOT/lib/32 -lmkl_intel \  
-lmkl_sequential -lmkl_core -lpthread -lm" F77=ifort AR="xild -lib" \  
  
../../pi/configure --with-ndim=3 CXX=icpc CC=icc CXXFLAGS="-O3 -ipo" \  
--with-blas="-Wl,-rpath,$MKLROOT/lib/32 -L$MKLROOT/lib/32 -lmkl_intel \  
-lmkl_sequential -lmkl_core -lpthread -lm" F77=ifort AR="xild -lib" \  
--enable-mpi MPICXX=mpic++ MPICC=mpicc MPIF77=mpif77
```

Frequently Asked Questions

23.1 How to raise issues and get help

This list of frequently asked questions will be built base on discussion on the [pi-qmc forum](#) and on issues that arise on the [bug tracker](#).

23.2 How to help update the FAQ

If you are a project member, simply checkout the code and go to the `pi-qmc/doc/sphinx/faq.rst` file and start editing. The markup language is ReStructuredText, and running `make html` will build the documentation on your machine so you can proofread the resulting HTML. When you commit your changes to GitHub, the websites will be automatically updated with your changes, including the FAQ page.

23.3 The Questions

23.3.1 How do I get help?

Read this manual and go to the [pi-qmc forum](#) to search for discussions or post your own questions.

Publications

A reverse chronological list of publications using pi-qmc.

- Peter G. McDonald, Edward J. Tyrrell, John Shumway, Jason M. Smith, and Ian Galbraith, “Tuning biexciton binding and anti-binding in core/shell quantum dots,” *Phys. Rev. B* 86, 125310, (2012).
- J. Shumway and M. J. Gilbert, “Effects of Fermion Flavor on Exciton Condensation in Double Layer Systems,” *Phys. Rev. B* 85, 033103, (2012).
- P. G. McDonald, J. Shumway, and I. Galbraith, “Lateral spatial switching of excitons using vertical electric fields in semiconductor quantum rings,” *Appl. Phys. Lett.*, 97, 173101 (2010).
- Jesper Pedersen, Lei Zhang, Matthew J. Gilbert, and J. Shumway, “Path integral study of the role of correlation in exchange coupling of spins in double quantum dots and optical lattices,” *J. Phys.: Condens. Matter* 22, 145301 (2010).
- J. Shumway and M. J. Gilbert, “Formation and Transport of Correlated Electron States at Room Temperature in Graphene Bilayers,” *ECS Transactions* 28, 29–37 (2010).
- M. J. Gilbert and J. Shumway, “Probing quantum coherent states in bilayer graphene,” *J. Comput. Electron.* 8, 51–59 (2009).
- Sutharsan Ketharanathan, Sourabh Sinha, John Shumway, and Jeff Drucker, “Electron charging in epitaxial Ge quantum dots on Si(001),” *J. Appl Phys.* 105, 044312 (2009).
- M. Wimmer, S. V. Nair, and J. Shumway, “Biexciton recombination rates in self-assembled quantum dots,” *Phys. Rev. B* 73, 165305 (2006).
- J. Shumway and Matthew J. Gilbert, “Path Integral Monte Carlo Simulations of Nanowires and Quantum Point Contacts,” *J. Phys.: Conf. Series* 35, 190-196 (2006).
- J. Shumway, “All-Electron Path Integral Monte Carlo Simulations of Small Atoms and Molecules,” pp. 181-195 in *Computer Simulations Studies in Condensed Matter Physics XVII*, edited by D. P. Landau, S. P. Lewis, and H. B. Schütter, (Springer Verlag, Heidelberg, Berlin, 2006).
- J. Shumway and D. M. Ceperley, “Quantum Monte Carlo Methods in the Study of Nanostructures,” in *Handbook of Theoretical and Computational Nanotechnology*, edited by Michael Rieth and Wolfram Schommers, Volume 3: pp. 605–641, ISBN: 1-58883-045-4 (American Scientific Publishers, 2006).
- M. Harowitz, Daejin Shin, and J. Shumway, “Path-Integral Quantum Monte Carlo Techniques for Self-Assembled Quantum Dots,” *J. of Low Temp. Phys.* 140, 211-226 (2005).
- J. Shumway, “A Quantum Monte Carlo Method for Non-Parabolic Electron Bands in Semiconductor Heterostructures,” *J. Phys.: Condens. Matter* 17, 2563-2570 (2005).

- M. Harowitz and J. Shumway, “Path Integral Simulations of Charged Multiexcitons in InGaAs/GaAs Quantum Dots,” pp. 697-698 in *Physics of Semiconductors: 27th International Conference on the Physics of Semiconductors*, edited by José Menéndez and Chris G. Van de Walle (AIP, 2005).
- *genindex*
- *search*

B

Bose-Einstein condensate, 3

D

density, 32

dynamic structure factor, 34

M

molecules, 3

P

partition function

 simple harmonic oscillator, 10

Q

quantum dot, 3

quantum point contact, 3

R

response function, 33

S

simple harmonic oscillator

 partition function, 10

structure factor, 34