Tarea # 2. Matrix Squaring, Trotter and Path-Integral Quantum Monte Carlo

Johans Restrepo Cárdenas

Instituto de Física. Universidad de Antioquia.

15 de abril de 2020

Matrix squaring (Convolution)

Modifique el siguiente programa, el cual hace uso de la matriz densidad de una partícula libre (free off-diagonal density matrix) y de la fórmula de Trotter para obtener $\pi(x)$ en un β final escribiendo los resultados en un archivo.

Tenga en cuenta que $\pi(x) = \rho(x, x, \beta)/Z$, con $Z(\beta) = \int \rho(x, x, \beta) dx$.

matrix_square_harmonic.py

```python
import math, numpy

# Free off-diagonal density matrix
def rho_free(x, xp, beta):
    return (math.exp(-(x - xp) ** 2 / (2.0 * beta)) /
            math.sqrt(2.0 * math.pi * beta))

# Harmonic density matrix in the Trotter approximation (returns the full matrix)
def rho_harmonic_trotter(grid, beta):
    return numpy.array([[rho_free(x, xp, beta) * \
                         numpy.exp(-0.5 * beta * 0.5 * (x ** 2 + xp ** 2)) \
                         for x in grid] for xp in grid])

x_max = 5.0
nx = 100
dx = 2.0 * x_max / (nx - 1)
x = [i * dx for i in range(-(nx - 1) / 2, nx / 2 + 1)]
beta_tmp = 2.0 ** (-5)                  # initial value of beta (power of 2)
beta     = 2.0 ** 4                     # actual value of beta (power of 2)
rho = rho_harmonic_trotter(x, beta_tmp)  # density matrix at initial beta
while beta_tmp < beta:
    rho = numpy.dot(rho, rho)
    rho *= dx
    beta_tmp *= 2.0
    print 'beta: %s -> %s' % (beta_tmp / 2.0, beta_tmp)
```

Matrix squaring (Convolution)

Para ello, tenga en cuenta además el siguiente retazo de programa. En el código que entregue haga los respectivos comentarios:

```
Z = sum(rho[j, j] for j in range(nx + 1)) * dx
pi_of_x = [rho[j, j] / Z for j in range(nx + 1)]
f = open('data_harm_matrixsquaring_beta' + str(beta) + '.dat', 'w')
for j in range(nx + 1):
    f.write(str(x[j]) + ' ' + str(rho[j, j] / Z) + '\n')
f.close()
```

Corra su programa y adjunte una gráfica de $\pi(x)$ para un valor final $\beta = 4$. En la misma gráfica muestre la curva teórica de $\pi(x)$ (i.e. $\pi(x)_{quant}$ de la tarea 1). Explique por qué a alta temperatura, la matriz densidad del oscilador armónico cuántico $\rho(x, x, \beta)$ es casi clásica. Varíe los parámetros de temperatura inversa inicial y de discretización dx para explicar cuáles deben ser elecciones iniciales adecuadas de estos parámetros.

Matrix squaring (Convolution)

Con base en la expresión para la función partición $Z(\beta)$ dada por $Z(\beta) = \int \rho(x, x, \beta) dx$, o lo que es lo mismo:

```
Z = sum(rho[j, j] for j in range(nx + 1)) * dx
```

Corra su programa para calcular dicha función partición para un conjunto amplio de valores de β correspondientes a temperaturas cercanas al cero absoluto y adjunte una gráfica de $\ln Z(\beta)$ en función de la temperatura T con $k_B = 1$, y a partir de ella, calcule el valor esperado de la energía usando $< E >= -\partial \ln Z / \partial \beta$.

Path-integral Monte Carlo para el oscilador armónico

Modifique el siguiente programa para generar un camino de Feynman (β vs. x) y un histograma normalizado solo de las posiciones $x[0]$. Para ello, no almacene valores de $x[0]$ en cada iteración. Hágalo cada 10 pasos, introduciendo la condición **if step %10 == 0**.

naive_harmonic_path.py

```
1  import math, random
2
3  def rho_free(x, y, beta):    # free off-diagonal density matrix
4      return math.exp(-(x - y) ** 2 / (2.0 * beta))
5
6  beta = 4.0
7  N = 8                                          # number of slices
8  dtau = beta / N
9  delta = 1.0                                    # maximum displacement on one slice
10 n_steps = 1000000                              # number of Monte Carlo steps
11 x = [0.0] * N                                  # initial path
12 for step in range(n_steps):
13     k = random.randint(0, N - 1)               # random slice
14     knext, kprev = (k + 1) % N, (k - 1) % N    # next/previous slices
15     x_new = x[k] + random.uniform(-delta, delta)  # new position at slice k
16     old_weight  = (rho_free(x[knext], x[k], dtau) *
17                    rho_free(x[k], x[kprev], dtau) *
18                    math.exp(-0.5 * dtau * x[k] ** 2))
19     new_weight  = (rho_free(x[knext], x_new, dtau) *
20                    rho_free(x_new, x[kprev], dtau) *
21                    math.exp(-0.5 * dtau * x_new ** 2))
22     if random.uniform(0.0, 1.0) < new_weight / old_weight:
23         x[k] = x_new
24     print x
```

Path-integral Monte Carlo para el oscilador armónico

Modifique su programa para leer los datos del archivo generado en la sección anterior (matrix-squaring) y grafíquelos junto con el histograma. Para ello, haga uso de la siguiente función la cual lee un archivo de dos columnas:

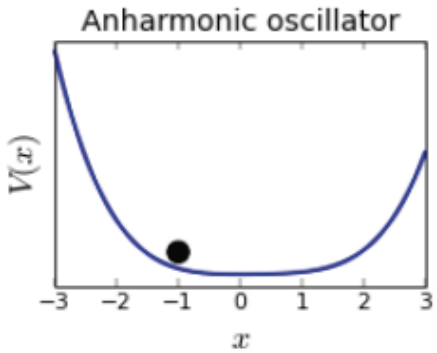
```
def read_file(filename):
    list_x = []
    list_y = []
    with open(filename) as f:
        for line in f:
            x, y = line.split()
            list_x.append(float(x))
            list_y.append(float(y))
    f.close()
    return list_x, list_y
```

Corra su programa con $\beta = 4$ y $N = 10$. Compare el histograma generado para $x[0]$ con el correspondiente a un $x[k]$ cualquiera con k entre 1 y $N - 1$. Compare a su vez cualquiera de estos histogramas con el que se obtendría de considerar todo el camino entero: $x[0], x[1], \cdots, x[N-1]$.

Nota: Al graficar el histograma, restrinja el rango de visualización al intervalo [-2.0,2.0] usando **pylab.xlim(-2.0,2-0)**.

Partícula en un potencial anarmónico

Considere un potencial anarmónico como el que se muestra en la siguiente figura:



y dado por la siguiente expresión:

$$V(x) = \frac{x^2}{2} - x^3 + x^4$$

Modifique los programas anteriores (matrix-squaring y path-integral) para tener en cuenta el nuevo potencial. Adjunte los programas y gráficas respectivas para $\beta = 4$. Realice también las comparaciones de $\pi(x)$ entre un potencial armónico y uno anarmónico en la misma gráfica para los programas de las secciones matrix-squaring y path-integral.