Kwantowe Monte Carlo – metoda wariacyjna

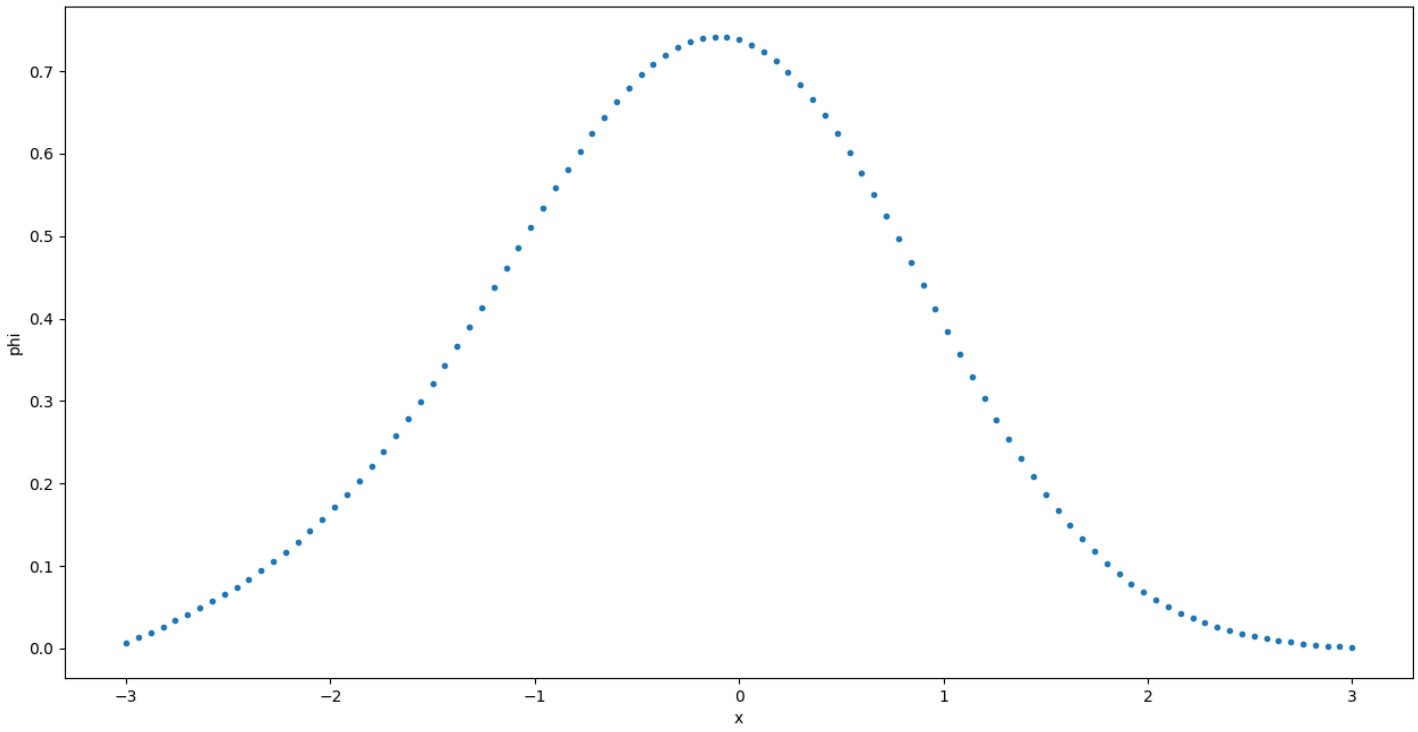
Janusz Twardak, 250333

Dane symulacji:

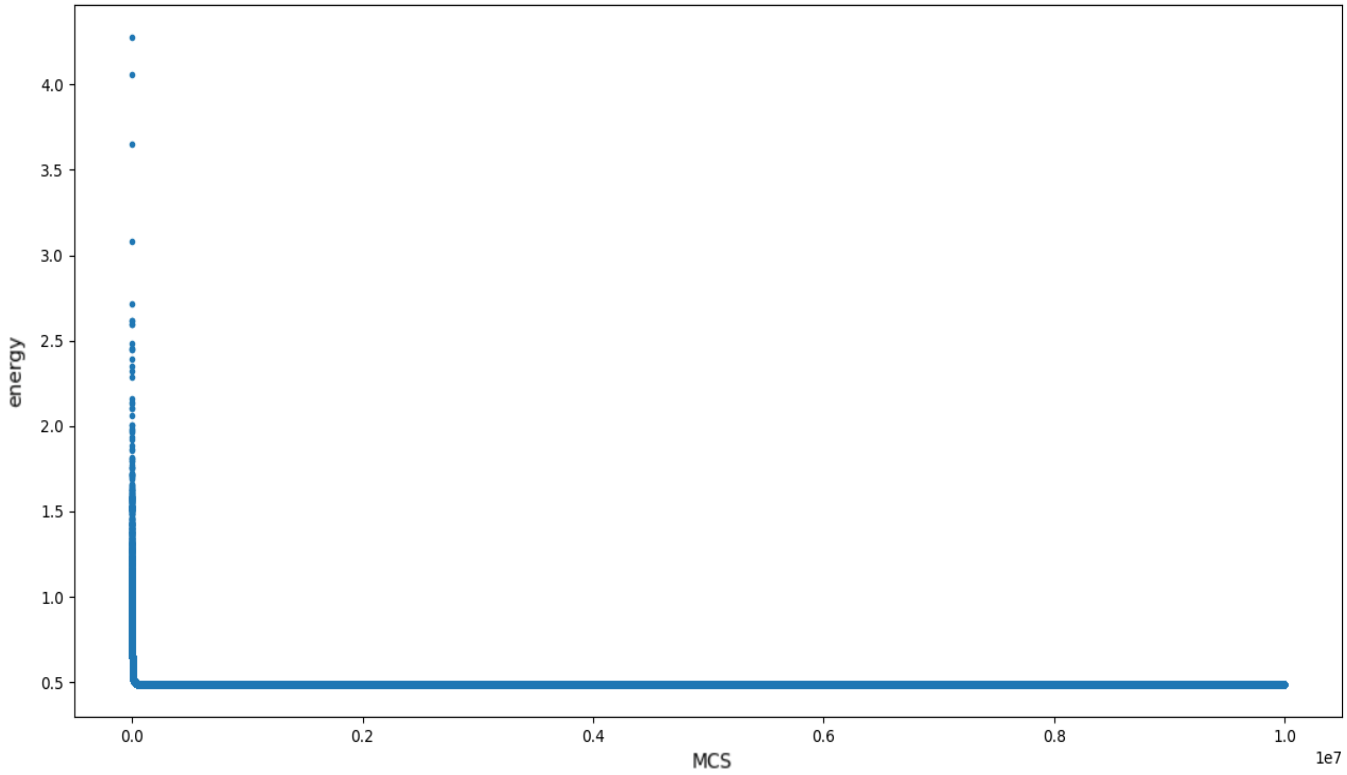
- liczba kroków Monte Carlo równa 10 000 000,
- przedział $< -3, 3 >$, liczba podziałów 100,
- początkowa funkcja falowa jest stała: $\varphi(x) = 0.2$,
- energia potencjalna cząstki w polu zewnętrznym: $V(x) = \frac{1}{2}x^2 + 0.1x^3$,
- stała przy obliczaniu nowych wartości funkcji falowej $d\,\varphi = 0.1$

Wartość energii stanu podstawowego:

$$E = 0.48606906247578346\ \hbar\omega$$



Wykres 1: wykres unormowanej funkcji falowej odpowiadającej stanowi podstawowemu



Wykres 2: wykres energii układu w toku symulacji E(MCS) (energia w jednostkach $\hbar\omega$)

Kod programu:

```
import random


def main():

    #DANE WEJSCIOWE:
    STEPS = 1000    # liczba krokow Monte Carlo
    DIVISIONS = 100 # liczba podzialow przedzialu
    D_PHI = 0.1 #


    phi = [0.2] * (DIVISIONS + 1) # poczatkowa funkcja phi - funkcja stala, wszystkie
                                                                    wartosci rowne 0.2

    divisionLength = 6 / DIVISIONS  # dlugosc jednego podziału
```

```
U = 0.0     # obliczanie U poczatkowego
for i in range(DIVISIONS + 1):
    U += phi[i]**2 * V(-3 + i * divisionLength)

T = 0.0     # obliczanie T poczatkowego
for i in range(1, 100):     # petla nie uwzglednia skrajnych punktow
    T += 0.5 * (phi[i] * (2 * phi[i] - phi[i - 1] - phi[i + 1]))/(divisionLength**2)
 T += 0.5 / divisionLength**2 * (phi[0] * (2 * phi[0] - phi[1]) + phi[DIVISIONS] * (2
                                                * phi[DIVISIONS] - phi[DIVISIONS-1]))

numerator = U + T   # licznik
denominator = 0     # mianownik (inicjalizacja)

for value in phi:   # obliczanie mianownika poczatkowego
    denominator += value**2

energy = numerator / denominator    # energia poczatkowa
saveToFile(energy)

for _ in range(STEPS):   # petla wykonujaca kroki Monte Carlo
    for _ in range(DIVISIONS + 1):
        randomX = random.randint(DIVISIONS)     # wylosowanie punktu z osi X
        randomValue = random.random()   # przyjmuje wartosc od 0 do 1

        phiTrial = phi[randomX] + (randomValue - 0.5) * D_PHI
        deltaPhi = phiTrial - phi[randomX]
        deltaPhiSquared = phiTrial**2 - phi[randomX]**2

        if(randomX != 0 and randomX != DIVISIONS):  # w przypadku wylosowania punktu
             dT = (deltaPhiSquared - deltaPhi * (phi[randomX + 1] + phi[randomX - 1])) / (div
                                                                                  isionLength**2)
        elif(randomX == 0):
             dT = (deltaPhiSquared - deltaPhi * (phi[randomX + 1])) / (divisionLength
                                                                               **2)
        elif(randomX == DIVISIONS):
           dT = (deltaPhiSquared - deltaPhi * (phi[randomX - 1])) / (divisionLength**2)

        dU = deltaPhiSquared * V(-3 + randomX * divisionLength)     # testowe dU
        newEnergy = (numerator + dU + dT) / (denominator + deltaPhiSquared)
                                                    # obliczenie energii po zmianie

        if (energy > newEnergy):    # zaakceptowanie zmian
            energy = newEnergy
            numerator += dU + dT
            denominator += deltaPhiSquared
            phi[randomX] = phiTrial
            print(energy)

    saveToFile(energy)

g.write(str(phi))
```

```
def saveToFile(energy):
    f.write(str(energy))
    f.write("\n")



def V(x):   # obliczanie V = 0.5 * x^2 + 0.1 * x^3
    V = 0.5 * x**2 + 0.1 * x**3
    return V



if __name__ == "__main__":

    f = open("energytest.txt", 'w')
    g = open("phitest.txt", 'w')

    main()

    f.close()
    g.close()
```