ΥΦΥ201 | Υπολογιστική κβαντική φυσική και εφαρμογές

Παρουσίαση υπολογιστικής εργασίας / Computational Problems for Physics (with guided solutions using Python)

Ημ/νία: 21/06/2022

Όν/μο: Ελεάννα Χωραΐτη Σιδέρη

Αρ. Μητρώου: 4406

Listing 6.16

Το πρόγραμμα SqBilliardQM.py λύνει την χρονοεξαρτόμενη εξίσωση του Schrodinger για την κίνηση κυματοπακέτου σε μαθηματικό τετράγωνο τραπέζι μπιλιάρδου.

Το μαθηματικό μπιλιάρδο περιγράφει ένα δυναμικό σύστημα στο οποίο ένα σωματίδιο κινείται ελεύθερα σε μια ευθεία γραμμή μέχρι να χτυπήσει έναν οριακό τοίχο, υφίσταται ανάκλαση (με ίσα προσπίπτουσα γωνία και γωνία ανάκλασης), και στη συνέχεια συνεχίζει σε ευθεία γραμμή μέχρι την επόμενη σύγκρουση.

Η άσκηση αυτή είναι μια προσέγγιση του κβαντικού χάους. Το χάος είναι μια συχνά παρατηρούμενη συμπεριφορά στα κλασικά μη γραμμικά συστήματα. Η παρατήρηση του χάους σε κβαντικά συστήματα μπορεί να προσεγγιστεί αν εξετάσουμε ένα κβαντικό σύστημα του οποίου το κλασικό ανάλογο έχει χαοτική συμπεριφορά. Ως παράδειγμα, το βλήμα στο κλασικό μπιλιάρδο συχνά συγκρούεται ατελείωτα, γεμίζοντας όλο τον επιτρεπόμενο χώρο χωρίς ποτέ να επαναλαμβάνεται. Εδώ βάλαμε ένα γκαουσιανό κυματοπακέτο στην θέση του βλήματος.

Το γκαουσιανό κυματοπακέτο δίνεται από τον τύπο:

$$\psi(x, y, t = 0) = e^{i(k_x x + k_y y)} e^{-A(x-x_0)^2 - A(y-y_0)^2}$$

Στο πρόγραμμα διακριτοποιούμε το κυματοπακέτο ώς εξής:

$$Gauss = e^{-A(x-x_0)^2 - A(y-y_0)^2}$$

$$R_{i,j}^n = Gauss * \cos(K_x j + K_y i)$$

$$I_{i,j}^n = Gauss * \sin(K_x j + K_y i)$$

,

όπου i, j είναι τα σημεία x, y του τετράγωνου πλέγματους του τραπεζιού.

Στον z-άξονα έχουμε το πλάτος του κυματοπακέτου $Z = (I^2 + R^2)$

Αφού αρχικοποιηθεί το πλέγμα γίνεται η ολοκλήρωση στο χρόνο σύμφωνα με τις διακριτοποιημένες σχέσεις:

$$I_{i,j}^{n+1} = I_{i,j}^n + \frac{\Delta t}{\Delta x^2}(R_{i+,j}^n + R_{i-1,j}^n - 4R_{i,j}^n + R_{i,j+1}^n + R_{i,j-1}^n)$$

$$R_{i,j}^{n+1} = R_{i,j}^n + \frac{\Delta t}{\Delta x^2}(I_{i+,j}^n + I_{i-1,j}^n - 4I_{i,j}^n + I_{i,j+1}^n + I_{i,j-1}^n)$$

Αρχικά εισάγουμε τα πακέτα της python που θα χρειαστούμε.

In [27]:
```
import numpy as np
import matplotlib.pyplot as plt
```

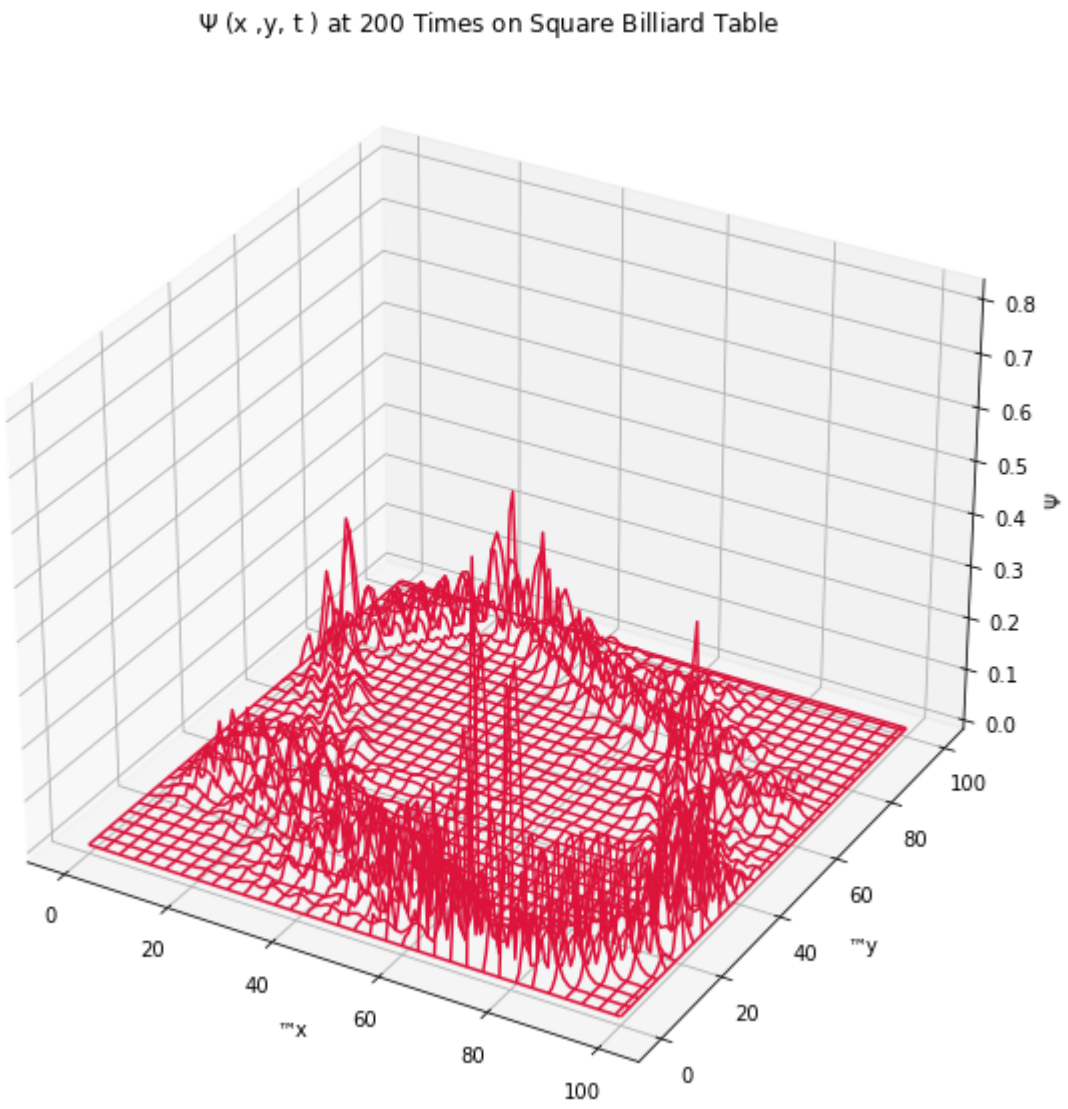
Το κύριο σώμα του προγράμματος.

In [28]:
```
Nmax = 101
dt = 0.01
dx = 0.2
dx2 = dx*dx
fc = dt/dx2
Tmax = 200
Kx = 10.
Ky = 15.
xin = 50.
yin = 50.

I = np.zeros((Nmax,Nmax),float)
R = np.zeros((Nmax,Nmax),float)
for i in range(1,Nmax-1): #Initial psi
    for j in range(1 ,Nmax-1):
        Gauss = np.exp(-.05*(i-yin)**2-.05*(j-xin)**2)
        R[i,j] = Gauss*np.cos(Kx*j + Ky*i)
        I[i,j] = Gauss*np.sin(Kx*j + Ky*i)
for t in range (0, Tmax): # Step through time
    R[1:-1 ,1:-1] = R[1:-1 ,1:-1] - fc*(I[2:,1:-1] + I[0:-2 ,1:-1]
                   -4*I[1:-1,1:-1] + I[1:-1,2:] + I[1:-1 ,0:-2])
    I[1:-1 ,1:-1] = I[1:-1 ,1:-1] + fc*(R[2:,1:-1] + R[0:-2,1:-1]
                    -4*R[1:-1,1:-1] + R[1:-1,2:] + R[1:-1,0:-2])
x = y = np.arange(0, Nmax)
X, Y = np.meshgrid (x,y)
Z = (I**2 + R**2)
```

Δημιουργία διαγράμματος του πλάτους του κυματοπακέτου μετά από 200

βήματα χρόνου.

In [29]:
```
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(projection="3d")
ax.plot_wireframe(X ,Y ,Z, color='crimson')
ax.set_xlabel("™x")
ax.set_ylabel("™y")
ax.set_zlabel("$\Psi$")
ax.set_title("$\Psi$ (x ,y, t ) at 200 Times on Square Billiard Table")
plt.show()
```

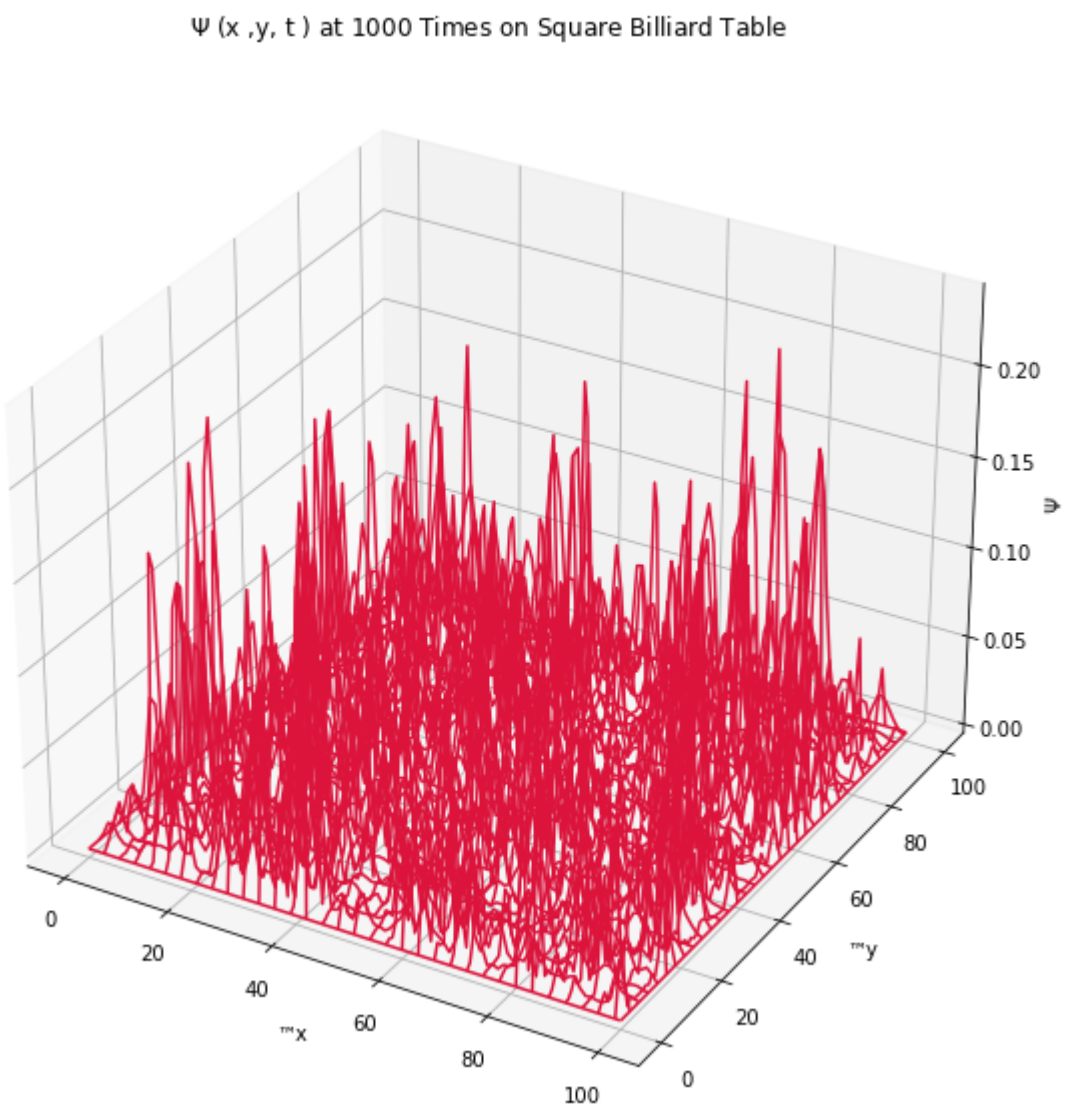


Δημιουργία διαγράμματος του πλάτους του κυματοπακέτου μετά από 2000 βήματα χρόνου.

In [30]:
```
Tmax = 1000
I = np.zeros((Nmax,Nmax),float)
R = np.zeros((Nmax,Nmax),float)
for i in range(1,Nmax-1): #Initial psi
    for j in range(1 ,Nmax-1):
        Gauss = np.exp(-.05*(i-yin)**2-.05*(j-xin)**2)
        R[i,j] = Gauss*np.cos(Kx*j + Ky*i)
        I[i,j] = Gauss*np.sin(Kx*j + Ky*i)
```

```
for t in range (0, Tmax): # Step through time
    R[1:-1 ,1:-1] = R[1:-1 ,1:-1] - fc*(I[2:,1:-1] + I[0:-2 ,1:-1]
                   -4*I[1:-1,1:-1] + I[1:-1,2:] + I[1:-1 ,0:-2])
    I[1:-1 ,1:-1] = I[1:-1 ,1:-1] + fc*(R[2:,1:-1] + R[0:-2,1:-1]
                    -4*R[1:-1,1:-1] + R[1:-1,2:] + R[1:-1,0:-2])
x = y = np.arange(0, Nmax)
X, Y = np.meshgrid (x,y)
Z = (I**2 + R**2)
```

In [31]:
```
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(projection="3d")
ax.plot_wireframe(X ,Y ,Z, color='crimson')
ax.set_xlabel("™x")
ax.set_ylabel("™y")
ax.set_zlabel("$\Psi$")
ax.set_title("$\Psi$ (x ,y, t ) at 1000 Times on Square Billiard Table")
plt.show()
```



Listing 6.1

Το πρόγραμμα HOnumeric.py λύνει αριθμητικά την κυματική εξίσωση του Schrödinger για τον 1D αρμονικό ταλαντωτή χρησιμοποιώντας τον αλγόριθμο Runge-Kutta 4ης τάξης και αυθαίρετες αρχικές συνθήκες.

Η χρονοανεξάρτητη εξίσωση Schrodinger του αρμονικού ταλαντωτή μιας διάστασης προκύπτει από τη σχέση: $H\psi = E\psi$, όπου H η Χαμιλτονιανή του αρμονικού ταλαντωτή και E η ενέργεια μιας κατάστασης.

Η ενέργειακές καταστάσεις δίνονται από τον τύπο $E_n = \hbar\omega(n + \frac{1}{2})$.

Τελικά η εξίσωση που επιλύεται βρίσκεται σε αδιάστατη μορφή και είναι η:

$$\frac{d^2u}{dx^2} + (2n + 1 - x^2)u = 0, n = 0, 1, 2, \ldots$$

Αρχικά εισάγουμε τα πακέτα της python που θα χρειαστούμε.

In [32]:
```
import numpy as np
import matplotlib.pylab as plt
```

Η συνάρτηση rk4Algor εκτελεί την ολοκλήρωση της u με τη μέθοδο Runge-Kutta 4ης τάξης.

Τα ορίσματα της συναρτησης είναι:

t: η μεταβλητή ως προς την οποία ολοκληρώνουμε

h: το βήμα ολοκλήρωσης

N: ο αριθμός των εξισώσεων προς ολοκλήρωση

y: το διάνυσμα των τιμών στο προηγούμενο βήμα ολοκλήρωσης

f: η συνάρτηση που ολοκληρώνεται

In [33]:
```
def rk4Algor(t,h,N,y,f):
    k1=np.zeros(N); k2=np.zeros(N); k3=np.zeros(N); k4=np.zeros(N)
    k1 = h*f(t,y)
    k2 = h*f(t+h/2.,y+k1/2.)
    k3 = h*f(t+h/2.,y+k2/2.)
    k4 = h*f(t+h,y+k3)
    y=y+(k1+2*(k2+k3)+k4)/6.
    return y
```

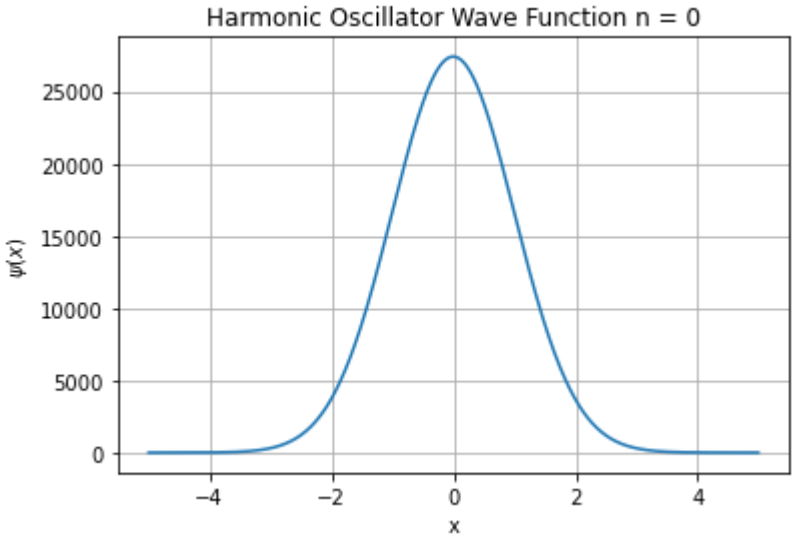
Το κύριο σώμα του προγράμματος.

In [34]:
```
rVec = np.zeros((1000),float) # x values for plot
psiVec = np.zeros((1000),float) # Wave function values
```

```
fVec = np.zeros(2) #the functions vector
y = np.zeros((2)) #initial conditions vector

n = 0 # n of the bound state energy

def f(x,y): # ODE RHS
    fVec[0] = y[1]
    fVec[1] = -(2*n+1-x**2)*y[0]
    return fVec

if(n%2==0): y[0]=1e-8  # Set parity
else: y[0]=-1e-8

y[1] = 1.

i=0
f (0.0, y)  # RHS at r = 0
dr = 0.01

for r in np.arange(-5 ,5 , dr):     # Compute WF steps of dr
    rVec[i] = r
    y = rk4Algor(r ,dr ,2 ,y ,f)
    psiVec[i] = y[0]
    i = i+1       # Advance i
```

Δημιουργία διαγράμματος με την κυματοσυνάρτηση.

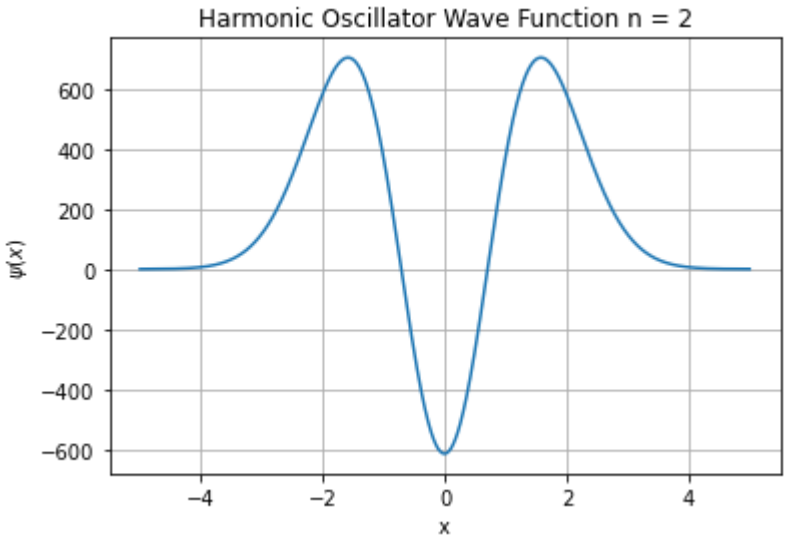
In [35]:
```
plt.figure()
plt.plot(rVec, psiVec)
plt.grid()
plt.title("Harmonic Oscillator Wave Function n = 0")
plt.xlabel("x")
plt.ylabel("$\psi(x)$")
plt.show()
```



In [36]:
```
rVec = np.zeros((1000),float) # x values for plot
psiVec = np.zeros((1000),float) # Wave function values

fVec = np.zeros(2) #the functions vector
y = np.zeros((2)) #initial conditions vector
```

```
n = 2 # n of the bound state energy

def f(x,y): # ODE RHS
    fVec[0] = y[1]
    fVec[1] = -(2*n+1-x**2)*y[0]
    return fVec

if(n%2==0): y[0]=1e-8  # Set parity
else: y[0]=-1e-8

y[1] = 1.

i=0
f (0.0, y)  # RHS at r = 0
dr = 0.01

for r in np.arange(-5 ,5 , dr):     # Compute WF steps of dr
    rVec[i] = r
    y = rk4Algor(r ,dr ,2 ,y ,f)
    psiVec[i] = y[0]
    i = i+1       # Advance i
```

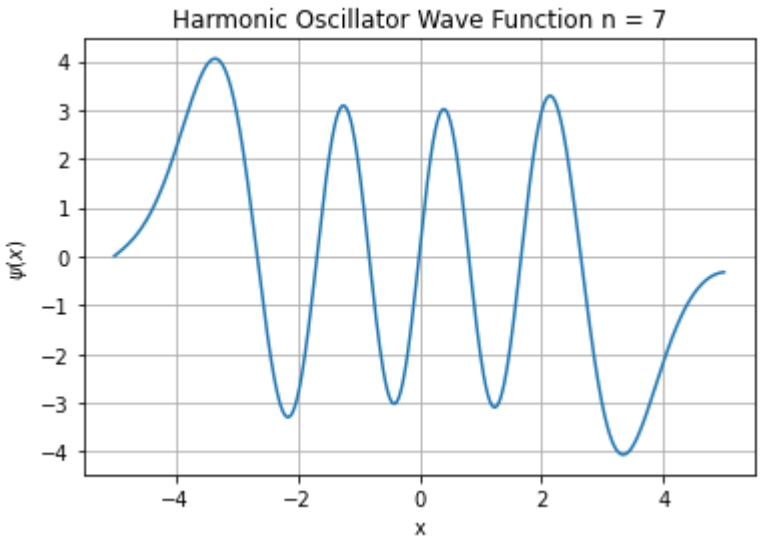
In [37]:
```
plt.figure()
plt.plot(rVec, psiVec)
plt.grid()
plt.title("Harmonic Oscillator Wave Function n = 2")
plt.xlabel("x")
plt.ylabel("$\psi(x)$")
plt.show()
```



In [38]:
```
rVec = np.zeros((1000),float) # x values for plot
psiVec = np.zeros((1000),float) # Wave function values

fVec = np.zeros(2) #the functions vector
y = np.zeros((2)) #initial conditions vector

n = 7 # n of the bound state energy

def f(x,y): # ODE RHS
    fVec[0] = y[1]
```

```
    fVec[1] = -(2*n+1-x**2)*y[0]
    return fVec

if(n%2==0): y[0]=1e-8  # Set parity
else: y[0]=-1e-8

y[1] = 1.

i=0
f (0.0, y)  # RHS at r = 0
dr = 0.01

for r in np.arange(-5 ,5 , dr):     # Compute WF steps of dr
    rVec[i] = r
    y = rk4Algor(r ,dr ,2 ,y ,f)
    psiVec[i] = y[0]
    i = i+1       # Advance i
```

In [39]:
```
plt.figure()
plt.plot(rVec, psiVec)
plt.grid()
plt.title("Harmonic Oscillator Wave Function n = 7")
plt.xlabel("x")
plt.ylabel("$\psi(x)$")
plt.show()
```



In [ ]: