Apéndice A

Aplicación del formalismo de Wigner-Weyl al oscilador armónico unidimensional.

El Hamiltoniano para el caso de unidimensional de una partícula de masa m, no relativista, sometida a un potencial de oscilador armónico está dado por

$$\hat{H} = \frac{1}{2m}\hat{p}^2 + \frac{1}{2}m\omega^2\hat{x}^2. \tag{A.1}$$

Los autoestados correspondientes a este operador, en la representación posición, son de la forma

$$\psi(x)_n = \sqrt{\frac{1}{2^n n!}}\left(\frac{m\omega}{\pi\hbar}\right)^{\frac{1}{4}} e^{-\frac{m\omega x^2}{2\hbar}} H_n\Big(\sqrt{\frac{m\omega}{\hbar}}x\Big). \tag{A.2}$$

Luego, la función de Wigner correspondiente al n-enésimo autoestado del operador Hamiltoniano (A.1) vendrá dada, de acuerdo con (2.14), por la siguiente expresión

$$W_n(x,p) = \frac{1}{2^n\, n!}\left(\frac{m\omega}{4\pi^3\hbar^3}\right)^{\frac{1}{2}} \int dy\, e^{-\frac{ipy}{\hbar}} e^{\frac{-m\omega}{\hbar}\left(x^2+\frac{y^2}{4}\right)} H_n\left(\sqrt{\frac{m\omega}{\hbar}}(x+y/2)\right) H_n\left(\sqrt{\frac{m\omega}{\hbar}}(x-y/2)\right), \tag{A.3}$$

donde, $H_n(x)$ se corresponde con el n-ésimo polinomio de Hermite.

De acuerdo con (A.2) los dos primeros autoestados de (A.1), en la representación posición, son

$$\Psi_0(x) = \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} e^{-m\omega x^2/2\hbar}, \tag{A.4}$$

$$\Psi_1(x) = \left(\frac{m\omega}{\hbar}\right)^{1/2}\left(\frac{m\omega}{4\pi\hbar}\right)^{1/4} x e^{-m\omega q^2/2\hbar}. \tag{A.5}$$

Empleando la ecuación (A.3) se puede calcular la correspondiente función de Wigner para

cada uno de estos autoestados,

$$W_0(x,p) = \frac{1}{\pi\hbar} e^{-\frac{m\omega}{\hbar}(x^2+(m\omega)^{-2}p^2)}, \tag{A.6}$$

$$W_1(x,p) = \frac{1}{\pi\hbar} e^{-\frac{m\omega}{\hbar}(x^2+(m\omega)^{-2}p^2)} \left(-1+\frac{2m\omega x^2}{\hbar}+\frac{2p^2}{\hbar m\omega}\right) \tag{A.7}$$

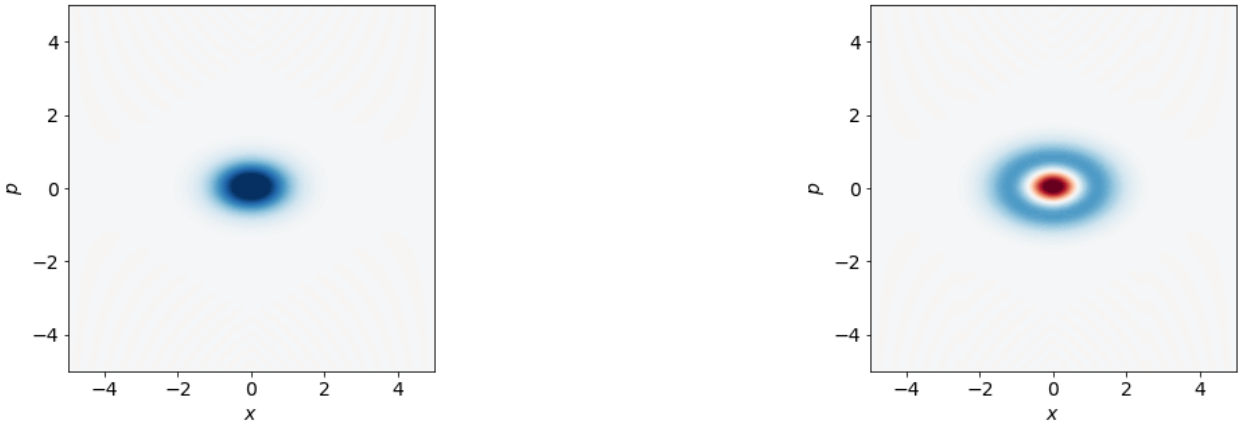

Figura A.1: Funciones de Wigner para los dos primeros autoestados del oscilador armónico cuántico. *Izquierda*: Función de Wigner para el estado base. *Derecha*: Función de Wigner para el primer estado excitado. El color azul denota partes positivas de la función de Wigner mientras que el color rojo denota parte negativas.

La función de Wigner para el estado base es una gaussiana, siendo positiva en todo el espacio de fase, mientras que, la función de Wigner asociada al primer estado excitado no es siempre positiva, en particular, en el punto $x = p = 0$ toma como valor -1. Una representación gráfica de estas dos funciones se muestra en la Fig. A.1.

La función de Wigner asociada al Hamiltoniano (A.1) está dada simplemente por $H(x,p) = \frac{p^2}{2m} + \frac{m\omega^2}{2}x^2$ puesto que,

$$\tilde{x}^2(x,p) = \int dy e^{\frac{-ipy}{\hbar}} \langle x - \frac{y}{2}|\hat{x}^2|x+\frac{y}{2}\rangle = x^2, \tag{A.8}$$

$$\tilde{p}^2(x,p) = \int dy e^{\frac{-ipy}{\hbar}} \langle x - \frac{y}{2}|\hat{p}^2|x+\frac{y}{2}\rangle = p^2. \tag{A.9}$$

Entonces, empleando la ecuación (2.12) y la correspondiente función de Wigner asociada al estado base tendremos el conocido valor,

$$\langle H_0\rangle = \int\int dy\, dp\, W_0(x,p)\left(\frac{p^2}{2m}+\frac{m\omega^2}{2}x^2\right) = \frac{\hbar\omega}{2}. \tag{A.10}$$

Considerando que la evolución del oscilador armónico es unitaria, tenemos que los estados del sistema evolucionan mediante $|\Psi(t)\rangle = e^{i\hat{H}t/\hbar}|\Psi\rangle$. Luego, podemos escribir los autoestados (A.4), (A.5) dependientes del tiempo como

$$\Psi_0(x,t) = \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} e^{-m\omega x^2/2\hbar}\, e^{-i\omega t/2}, \tag{A.11}$$

$$\Psi_1(x,t) = \left(\frac{m\omega}{\hbar}\right)^{1/2} \left(\frac{m\omega}{4\pi\hbar}\right)^{1/4} x e^{-m\omega x^2/2\hbar}\, e^{-3i\omega t/2}. \tag{A.12}$$

La combinación lineal de estos dos autoestados dependientes del tiempo está dada por

$$\Psi_{01}(x,t) \equiv \frac{1}{\sqrt{2}} \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} e^{-m\omega x^2/2\hbar} \left(e^{-i\omega t/2} + \sqrt{\frac{2m\omega}{\hbar}} x e^{-3i\omega t/2} \right), \tag{A.13}$$

y su correspondiente función de Wigner, ahora dependiente también del tiempo, será

$$W_{01}(x,p,t) = \frac{1}{2}\left(W_0(x,p) + W_1(x,p)\right) + \sqrt{\frac{m\omega}{2\pi^2\hbar^3}}\, e^{-m\omega/\hbar(x^2+(m\omega)^{-2}p^2)} \left(x\cos\omega t - \frac{p}{m\omega}\sin\omega t \right) \tag{A.14}$$

La parte dependiente del tiempo en (A.14) indica que la función de Wigner rotará sobre el espacio de fase con cambios del parámetro t. Gráficas de la función de Wigner para diferentes valores del parámetro t pueden verse en Fig. A.2.

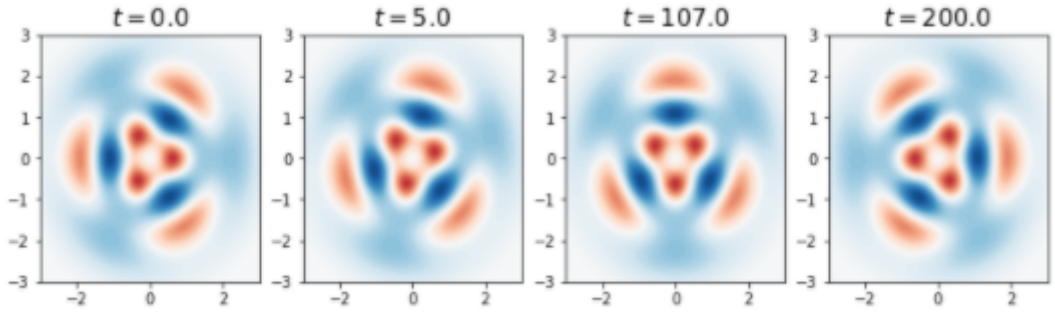

Figura A.2: Evolución de la función de Wigner para la combinación lineal (A.13)

Apéndice B

Código implementado

Código B.1: Código para el cálculo de la función de Wigner semi-clásica

```
1
2  from sympy import *
3  import sympy as sp
4  from sympy.abc import Q,q,P,p,z,k,y,a,c,x
5  from scipy.special import airy
6  from scipy.integrate import quad, dblquad
7  from qutip import *
8  from sympy.solvers import solve
9  import cmath
10 import math
11 import numpy as np
12 from numpy import exp,arange
13
14 #Grafica de Wigner semiclasica
15
16 from mpl_toolkits.mplot3d import Axes3D
17 from matplotlib import cm
18 from matplotlib.ticker import LinearLocator, FormatStrFormatter
19 import matplotlib.animation as ani
20 import matplotlib.colors as clr
21 import types
22 import IPython.display as ipyd
23 from qutip.ipynbtools import plot_animation
24 import matplotlib.pyplot as plt
25 plt.rcParams["figure.figsize"] = (12,9)
26 from pylab import meshgrid,cm,imshow,contour,clabel,colorbar,axis,title,
27 show
28 %matplotlib widget
29
30 #Valor de hbarra
31 hbarra=(10)**(-6)
32
33 #Retorna el area como funcion de q, para un valor dado de la energia,
34 #frecuencia y masa."
35
```

```
36 def Area(e,x0,m=4,w=1):
37     area = abs(integrate(sqrt(m*(2*e-m*(w**2)*(Q**2))), (Q, (q - x0),
38     (q + x0))) - 2*p*x0)
39     return area
40 
41 #Calcula los extremos de una elipse de energia e
42 def elipse_ext(e,m=4,w=1):
43     return(sqrt(2*e/(m*w**2)))
44 
45 def extremo_p(e,m=4,w=1):
46     return(sqrt(2*m*e))
47 
48 #Devuelve la expresion momentum como funcion de q, dados m y w
49 def momentum(e,m=4,w=1):
50     return sqrt(2*e*m - (m**2)*(w**2)*(q**2))
51 
52 #Devuelve un arreglo de puntos  +-momentum
53 def Points_momentum(e,x,m=4,w=1):
54         y = []
55         xfinal = []
56         for i in range(len(x)):
57             momento = momentum(e,m,w).subs(q,x[i])
58             momento_menos = momento*(-1)
59             y_inside = []
60             y_inside.append(momento)
61             y_inside.append(momento_menos)
62             y.append(y_inside)
63             xfinal.append(x[i])
64         return xfinal, y
65 
66 #Retorna la funcion de Airy para un array de puntos.
67 def Ai(x):
68     (ai, ai_prime, bi, bi_prime) = airy(x)
69     return ai
70 
71 #Calcula el denominador de la funci n de Wigner semiclasica.
72 def denominator(x,y,x0,e,pcuadrant):
73     return I_q2(x,x0,pcuadrant)* I_p1(x, x0, e,pcuadrant)
74     - I_p2(x,x0,e, pcuadrant)*I_q1(x,x0,pcuadrant)
75 
76 #Derivada de la funcion variable de accion con respecto a q, evaluada en 1
77 def I_q1(x,xo,pcuadrant,m=4,w=1):
78     if (pcuadrant):
79         return m*w*(x-xo)
80     else:
81         return m*w*(x+xo)
82 
83 #Derivada de la funcion variable de accion con respecto a q, evaluada en 2
84 def I_q2(x,xo, pcuadrant,m=4,w=1):
85     if(pcuadrant):
86         return m*w*(x+xo)
```

```
87      else:
88          return m*w*(x-xo)
89
90  #Derivada de la funcion variable de accion con respecto a p, evaluada en
91  #1
92  def I_p1(x,xo,e,pcuadrant, m=4,w=1):
93      pvalue = momentum(e).subs(q,x+xo)
94      if not pcuadrant:
95          pvalue = pvalue*(-1)
96      return (1/(m*w))*pvalue
97
98  # Derivada de la funcion variable de accion con respecto a p, evaluada en
99  #2
100 def I_p2(x,xo,e,pcuadrant, m=4,w=1 ):
101     pvalue = momentum(e).subs(q,x+xo)
102     if not pcuadrant:
103         pvalue = pvalue*(-1)
104     return (1/(m*w))*pvalue
105
106 #Calcula las raices de la condicion de estacionariedad
107 def Point_xo(q,p,e,m=4,w=1):
108     return 16*(p**4) - 32*(p**2)*m*e + 16*(m**3)*(w**2)*e*(q**2+z**2)
109     + 16*((q**2+z**2)**2)*((m**2)*(w**2)*(p**2)-(m**3)*e*(w**2))
110     - 4*(m**4)*(w**4)*((q**2 - z**2)**2)
111     + 4*(m**4)*(w**4)*((q**2+z**2)**4)
112
113 # Funcion Energia
114 def Energy(q,p,m=4,w=1):
115     return (1/2)*((p**2)/m) + (1/2)*m*(w**2)*(q**2)
116
117 """
118 ################################################################
119                 FUNCION DE WIGNER SEMICLASICA
120 ################################################################
121 """
122
123 """
124
125 Devuelve la funcion de wigner calculada por cuadrantes.
126
127 INPUT:
128     e-- number; Energia
129     extremo-- number; Energia para calcular la discretizacion
130     pun-- number; Cantidad de puntos
131     qpositive-- number; Indica si las q son positivas o negativas
132     ppositive-- number; Indica si las p son positivas o negativas
133     m-- number; Masa
134     w-- number; Frecuencia
135 OUTPUT:
136     qs-- list de number;
137         representa los puntos q donde se calculara la funcion de wigner
```

```
    ps-- list de number;
        representa los puntos p donde se calculara la funcion de wigner
    wigner-- list de number;
        representa los valores de la funcion de wigner en los puntos q,p

"""
def Semi_classical_Wigner_funct_cuadrant(e, extremo,  pun, qpositive,
ppositive,m=4,w=1):

    #Calculamos los puntos extremos de la elipse para un determinado
    #estado de energ a y par metros m y w.
    extrem = elipse_ext(extremo,m,w)

    #Hacemos un arreglo con los puntos dentro del intervalo de extremos.
    if(qpositive):
        x = np.linspace(0, float(extrem) -0.0000000000000001, pun)
    else:
        x = np.linspace(-float(extrem)+0.0000000000000001, 0, pun)
    #Hacemos un arreglo de puntos p que son dependientes de los puntos q
    #a trav s de la ecuaci n de la elipse
    x , y = Points_momentum(extremo,x,m,w)

    # Hacemos un array de puntos p para cada uno de los p extremos
    #encontrados a trav s de momentum para cada punto q
    array_puntos_p = []
    for j in range(len(y)):
        if(ppositive):
            array_puntos_p.append(np.linspace(0, float(y[j][0]),pun))
        else:
            array_puntos_p.append(np.linspace(float(y[j][1]), 0,pun))

    #Calculamos x0 a trav s de Point_xo
    finalq = []
    finalp = []
    finalx0 = []
    removed = []
    x0cantidad = 0
    for i in range(len(x)):

        for k in array_puntos_p[i]:
            xo1 = solve(Point_xo(x[i],k,e),z)
            if("I" not in str(xo1)):
                finalq.append(x[i])
                finalp.append(k)
                if (len(xo1) == 2):
                    finalx0.append(xo1[1])
                else:
                    finalx0.append(xo1[0])
            else:
                removed.append(xo1)
        x0cantidad = x0cantidad + len(array_puntos_p[i])
```

```
189     print (x0cantidad)
190 
191     # Evaluamos el area en cada punto de los arrays finalq,finalp y
192     #finalx0 correspondientes
193     area_array = []
194     for i in range(len(finalq)):
195         area = Area(e,finalx0[i])
196         variable = area.subs(q,finalq[i]).subs(p,finalp[i])
197         area_array.append(variable)
198 
199     final_area_array = []
200     for i in area_array:
201         variable = str(i).replace("asin", "math.asin").
202         replace("sqrt","np.sqrt")
203         final_area_array.append(eval(variable))
204 
205     final_area_array  = np.array(final_area_array)
206 
207     #Calcula la funcion de Airy para cada elemento de pointsEvaluated
208     #multiplicado por las respectivas constantes
209     argument_airy = -((3/(2*hbarra))*(np.array(final_area_array)))**(2/3)
210 
211     ai = Ai(argument_airy)
212 
213     wigner_array = []
214     filteredq = []
215     filteredp = []
216     for i in range(len(ai)):
217         denominatorValue =
218         denominator(finalq[i], finalp[i], finalx0[i], e,ppositive)
219         if denominatorValue != 0:
220             wigner_variable = hbarra*(np.sqrt(2)
221                     *(((3/2)*final_area_array[i])**(1/6))*ai[i])/
222                     (pi*hbarra**(2/3)*(denominator(finalq[i], finalp[i],
223                     finalx0[i], e,ppositive)**(1/2)))
224             wigner_variable =
225                         eval(str(wigner_variable).replace("pi", "np.pi"))
226             filteredq.append(finalq[i])
227             filteredp.append(finalp[i])
228             wigner_array.append(wigner_variable)
229     fig = plt.figure()
230     ax = fig.gca(projection='3d')
231 
232     ax.plot_trisurf(filteredq, filteredp, wigner_array, linewidth=0.2,
233     antialiased=True,cmap=cm.coolwarm)
234     ax.set_title('Semiclassical Wiger function')
235     ax.set_xlabel('q')
236     ax.set_ylabel('p')
237     ax.set_zlabel('$W(q,p)$')
238     plt.show()
239     return filteredq, filteredp, wigner_array
```

```
240
241 "-----------------------------------------------
242
243 #   EJEMPLO CALCULO PARA ESTADO DE ENERGIA E_0
244
245 -----------------------------------------------"
246
247 firstq, firstp, firstwigner = Semi_classical_Wigner_funct_cuadrant
248 (hbarra,hbarra,30,true, true)
249 secondq, secondp, secondwigner = Semi_classical_Wigner_funct_cuadrant
250 (hbarra,hbarra, 30, false, true)
251 thirdq, thirdp, thirdwigner = Semi_classical_Wigner_funct_cuadrant
252 (hbarra,hbarra, 30,false, false)
253 fourthq, fourthp, fourthwigner = Semi_classical_Wigner_funct_cuadrant
254 (hbarra,hbarra,30,true, false)
255
256 totalq = firstq + secondq + thirdq + fourthq
257 totalp = firstp + secondp + thirdp + fourthp
258 totalwigner = firstwigner + secondwigner + thirdwigner + fourthwigner
```

Código B.2: Código para el cálculo del primer orden el la expansión asintótica de la función de Wigner semi-clásica

```
1
2
3  from sympy import *
4  import sympy as sp
5  from sympy.abc import Q,q,P,p,z,k,y,a,c,x
6  from scipy.special import airy
7  from scipy.integrate import quad, dblquad
8  from qutip import *
9  from sympy.solvers import solve
10 import cmath
11 import math
12 import numpy as np
13 from numpy import exp,arange
14
15 #Grafica de Wigner semiclasica
16
17 from mpl_toolkits.mplot3d import Axes3D
18 from matplotlib import cm
19 from matplotlib.ticker import LinearLocator, FormatStrFormatter
20 import matplotlib.animation as ani
21 import matplotlib.colors as clr
22 import types
23 import IPython.display as ipyd
24 from qutip.ipynbtools import plot_animation
25 import matplotlib.pyplot as plt
26 plt.rcParams["figure.figsize"] = (12,9)
27 from pylab import meshgrid,cm,imshow,contour,clabel,colorbar,axis,title,
28 show
```

```
%matplotlib widget

#Valor de hbarra
hbarra=(10)**(-6)

# Devuelve la expresion de la variable de accion en funcion de q y p.
def ActionVariable(m=4,w=1):
    return (1/w)* (((p)**2/(2*m)) + ((1/2)*m *(w**2)*(q)**2))

#Devuelve la derivada de la variable de accion respecto a q
def DerivativeActionVariableq(m=4,w=1):
    return m*w*q

#Devuelve la derivada de la variable de accion respecto a p
def  DerivativeActionVariablep(m=4,w=1):
    return p/(m*w)

#Devuelve la segunda derivada de la variable de accion respecto a q
def SecondDerivativeActionVariableq(m=4,w=1):
    return m*w

#Devuelve la segunda derivada de la variable de accion respecto a p
def SecondDerivativeActionVariablep(m=4,w=1):
    return 1/(m*w)

#Devuelve la funcion B(q,p)
def FunctionB(m=4,w=1):
    return (DerivativeActionVariableq(m,w))**2
    *(SecondDerivativeActionVariablep(m,w))
    + ((DerivativeActionVariableq(m,w)**2)*
    SecondDerivativeActionVariableq(m,w))

"""
#####################################################################
            FUNCION DE WIGNER SEMICLASICA ORDEN 1
#####################################################################
"""

"""

Devuelve el primer orden de la expansion asintotica de la funcion de
#Wigner semiclasica.

INPUT:
    e-- number; Energia
    ext-- number; Energia para calcular la discretizacion
    m-- number; Masa
    w-- number; Frecuencia
OUTPUT:
    qs-- list de number;
    representa los puntos q donde se calculara la funcion de wigner
```

```
    ps-- list de number;
    representa los puntos p donde se calculara la funcion de wigner
    wigner-- list de number;
    representa los valores de la funcion de wigner en los puntos q,p

"""

def FirstOrderWigner(e,ext,m,w):
    extrem = elipse_ext(ext,m,w)
    newarrayq =
    np.linspace(-float(extrem)+0.00000000000001,
    float(extrem)-0.00000000000001,200)
    pointsq, pointsp = Points_momentum(ext,newarrayq,m,w)

    array_puntos_p = []
    array_puntos_pn = []
    for j in range(len(pointsp)):
        array_puntos_p.append(np.linspace(0, float(pointsp[j][0]),200))
        array_puntos_pn.append(np.linspace(0, float(pointsp[j][1]),200))
    actionVariableValuesPositive = []
    actionVariableValuesNegative = []
    finalQ = []
    finalPPositive = []
    finalPNegative = []
    for i in range(len(newarrayq)):
        for j in range(len(array_puntos_p[i])):
            qvalue = pointsq[i]
            ppositive = array_puntos_p[i][j]
            pnegative = array_puntos_pn[i][j]
            bexpression = FunctionB(m,w)
            bvaluepositive = bexpression.subs(q,qvalue).subs(p,ppositive)
            bvaluenegative = bexpression.subs(q,qvalue).subs(p,ppositive)
            ie = e/w
            expression = ActionVariable(m,w)
            valuepositive = expression.subs(q,qvalue).subs(p,ppositive)
            valuenegative = expression.subs(q,qvalue).subs(p,pnegative)

            fullValuepositive = 2*(valuepositive -

            ie)*(1/(((hbarra**2)*bvaluepositive)**(1/3)))
            fullValuenegative = 2*(valuenegative -

            ie)*(1/(((hbarra**2)*bvaluepositive)**(1/3)))
            actionVariableValuesPositive.append(float(fullValuepositive))
            actionVariableValuesNegative.append(float(fullValuenegative))
            finalQ.append(qvalue)
            finalPPositive.append(ppositive)
            finalPNegative.append(pnegative)
    expressionValuePositive = []
    expressionValueNegative = []
    for i in range(len(newarrayq)):
```

```
131         for j in range(len(array_puntos_p[i])):
132             qvalue = pointsq[i]
133             ppositive = array_puntos_p[i][j]
134             pnegative = array_puntos_pn[i][j]
135             expression = (1/np.pi)*
136                         (1/((hbarra**2)*bvaluenegative))**(1/3)
137             valuePositive = expression.subs(q,qvalue).subs(p,ppositive)
138             valueNegative = expression.subs(q,qvalue).subs(p,pnegative)
139             expressionValuePositive.append(valuePositive)
140             expressionValueNegative.append(valueNegative)
141     ai = Ai(np.array(actionVariableValuesPositive))
142     aiNega = Ai(np.array(actionVariableValuesNegative))
143 
144     finalValuesPositive = np.array(expressionValuePositive)*np.array(ai)
145     finalValuesNegative = np.array(expressionValueNegative)
146                         *np.array(aiNega)
147 
148     pPositives = []
149     pNegatives = []
150     for i in pointsp:
151         pPositives.append(i[0])
152         pNegatives.append(i[1])
153     resultq = finalQ + finalQ
154     resultp = finalPPositive + finalPNegative
155     resultFinal = []
156     for i in ai:
157         resultFinal.append(i)
158     for i in aiNega:
159         resultFinal.append(i)
160     fig = plt.figure()
161     ax = fig.gca(projection='3d')
162     plt.rcParams["figure.figsize"] = (15,12)
163     ax.plot_trisurf(resultq, resultp, resultFinal, linewidth=0.2,
164     antialiased=True, cmap=cm.coolwarm, alpha=0.5)
165     ax.set_title('Semiclassical Wigner function-First order correction')
166     ax.set_xlabel('q')
167     ax.set_ylabel('p')
168     ax.set_zlabel('$W(q,p)$')
169 
170     plt.show()
171 
172 
173     return resultq, resultp, resultFinal
```