

# HW7\_Code

March 17, 2023

```
[ ]: import random

from qutip import *
from qiskit.visualization import array_to_latex
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from scipy.optimize import curve_fit
from scipy.stats import chisquare
```

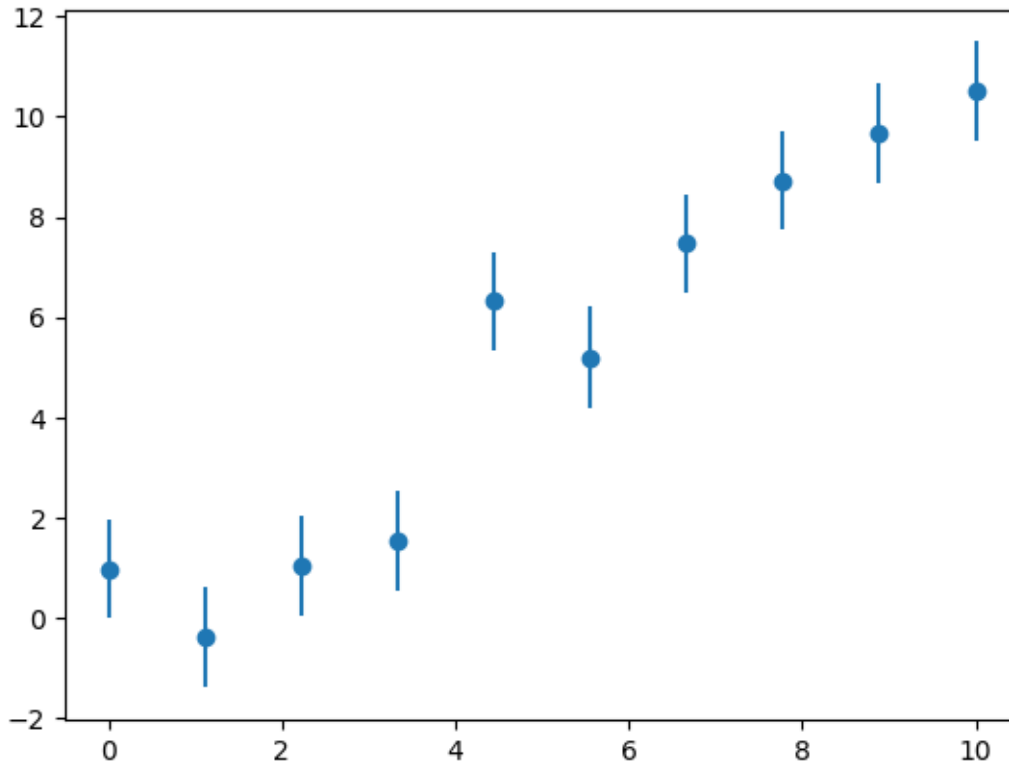
1 1)

1.1 1.A)

```
[ ]: slope = 1
offset = 0.25
time = np.linspace(0,10,10)
data = np.random.normal(slope*time + offset,1)
sigma = 1*np.random.normal(np.ones(len(data)),0.01)

plt.errorbar(time,data,yerr=sigma,fmt='o')
```

```
[ ]: <ErrorbarContainer object of 3 artists>
```



## 1.2 1.B)

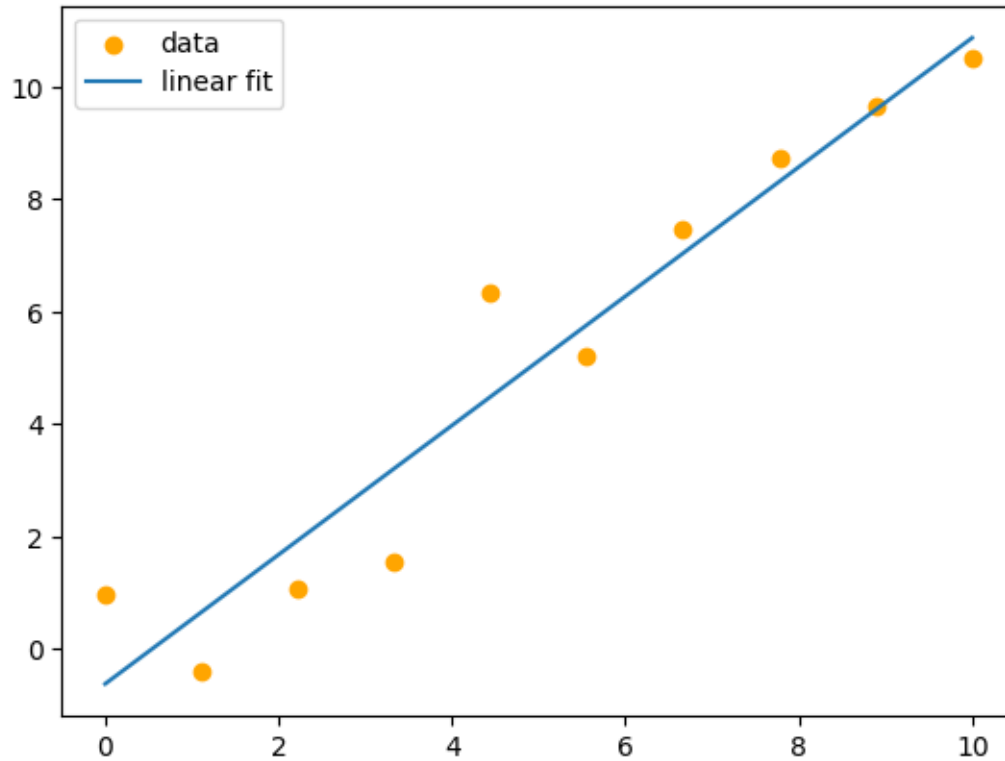
First, lets take a look at the linear fit

```
[ ]: def func_linear(x, a, b):
      return a*x + b
```

```
[ ]: lin_fit = curve_fit(func_linear, time, data, sigma=sigma)
      print("Linear fit parameters: a={:0.4f}, b={:0.4f}".format(lin_fit[0][0],
      ↪lin_fit[0][1]))
```

Linear fit parameters: a=1.1479, b=-0.6148

```
[ ]: plt.scatter(time, data, label='data', color='orange')
      plt.plot(time, func_linear(time, *lin_fit[0]), label='linear fit')
      plt.legend()
      plt.show()
```



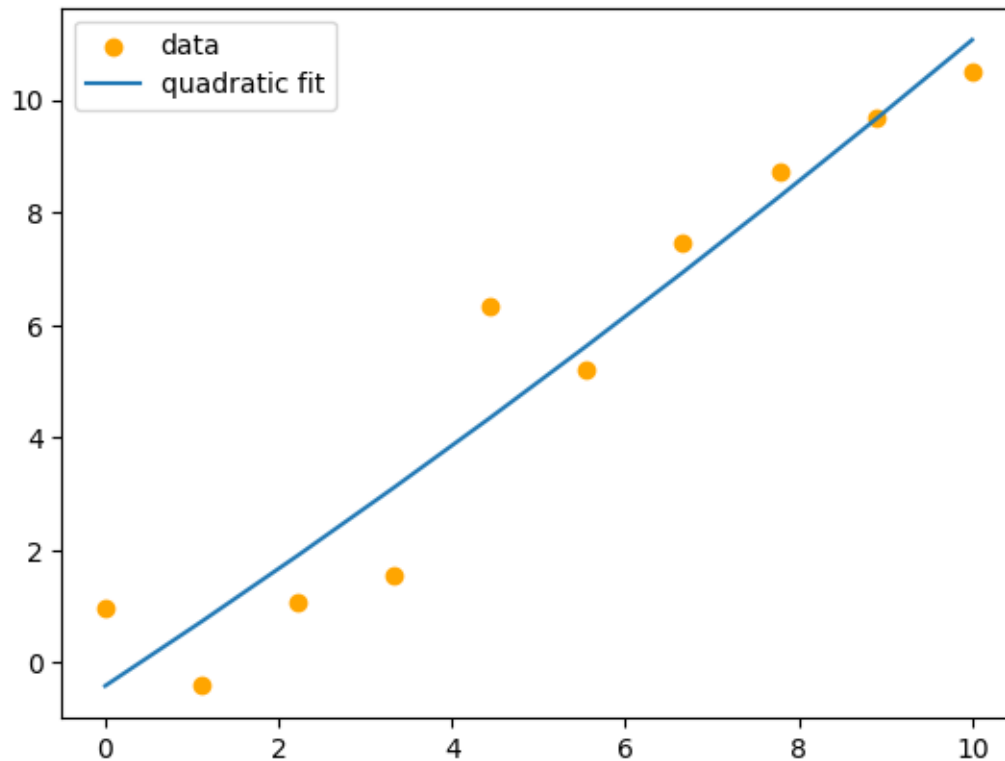
Next, lets take a look at the quadrtatic fit.

```
[ ]: def func_quadratic(x, a, b, c):
      return a * x**2 + b * x + c
```

```
[ ]: quad_fit = curve_fit(func_quadratic, time, data, sigma=sigma)
      print("Quadratic fit parameters: a={:0.4f}, b={:0.4f}, c={:0.4f}".
            ↪format(quad_fit[0][0], quad_fit[0][1], quad_fit[0][2]))
```

Quadratic fit parameters: a=0.0135, b=1.0132, c=-0.4167

```
[ ]: plt.scatter(time, data, label='data', color='orange')
      plt.plot(time, func_quadratic(time, *quad_fit[0]), label='quadratic fit')
      plt.legend()
      plt.show()
```



## 2 1.C)

```
[ ]: def chi_squared(data, model, sigma):
    sum = 0
    for i in range(len(data)):
        sum += ((data[i] - model[i])/sigma[i])**2
    return sum
```

```
[ ]: chi_square_linear = chi_squared(data, func_linear(time, *lin_fit[0]), sigma)
v_linear = len(data) - len(lin_fit[0])
print(v_linear)
print("X^2 / V for linear fit: {:.4f}".format(chi_square_linear/v_linear))
```

8

X<sup>2</sup> / V for linear fit: 1.4412

```
[ ]: chi_square_quadratic = chi_squared(data, func_quadratic(time, *quad_fit[0]),
    ↪sigma)
v_quadratic = len(data) - len(quad_fit[0])
print(v_quadratic)
```

```
print("X^2 / V for quadratic fit: {:.4f}".format(chi_square_quadratic/
↪v_quadratic))
```

7

X<sup>2</sup> / V for quadratic fit: 1.6257

Therefore, we can conclude that the linear fit is slightly better for fitting this data than the quadratic

### 3 3)

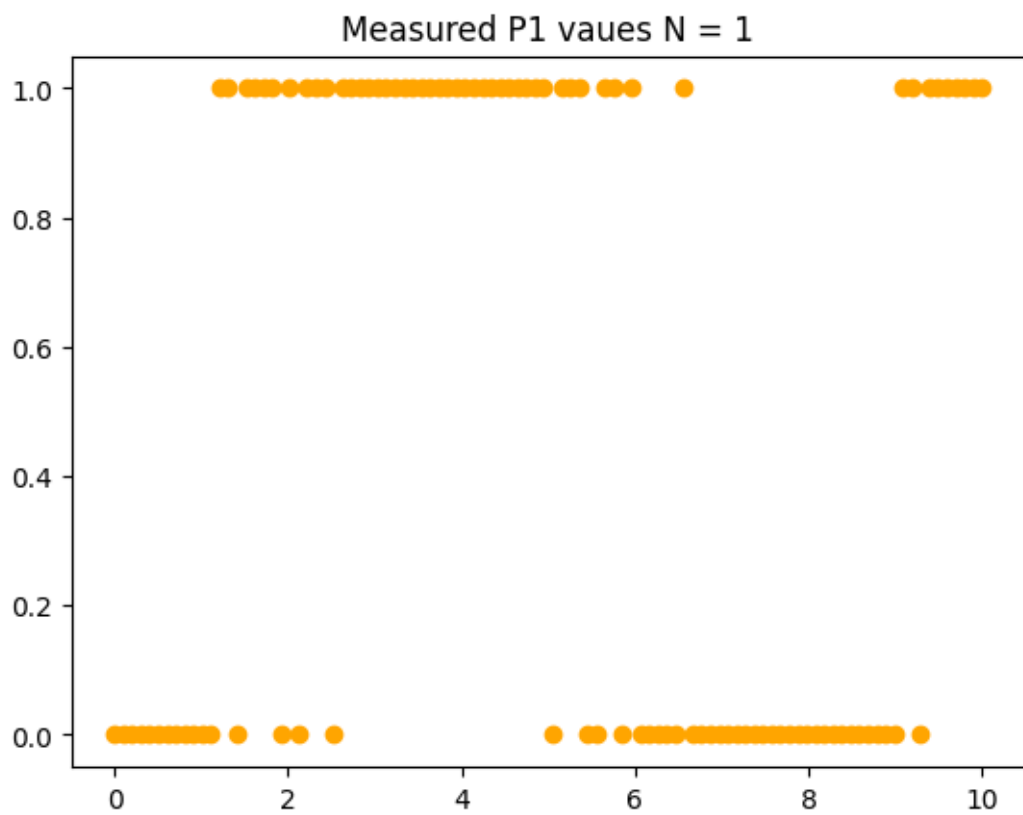
```
[ ]: t_p = 1
      delta = np.pi / 4
      Omega = 2 * np.pi
```

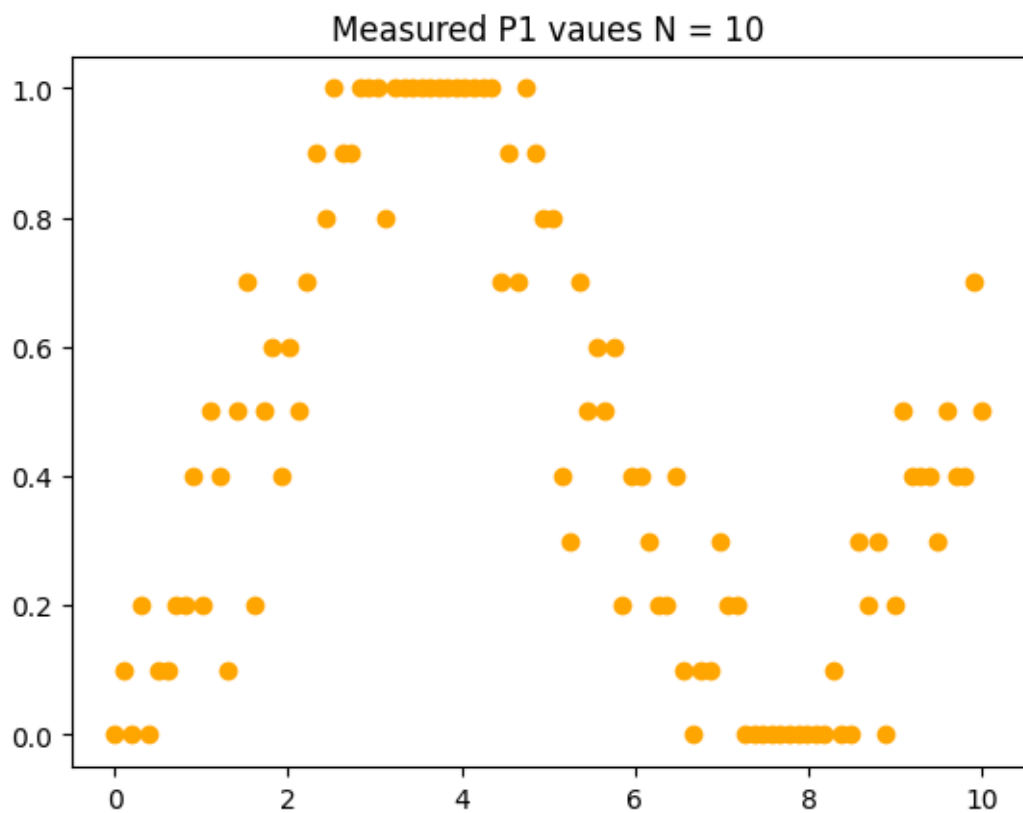
```
[ ]: def P1(t_w, Omega, delta):
      Omega_p = np.sqrt(Omega**2 + delta**2)
      return 1 - (Omega**2) / (Omega_p**4) * (Omega_p * np.cos(delta * t_w / 2) *
↪np.sin(np.pi/2) - 2 * delta * np.sin(delta * t_w / 2) * np.sin(np.pi/4)**2
↪)**2
```

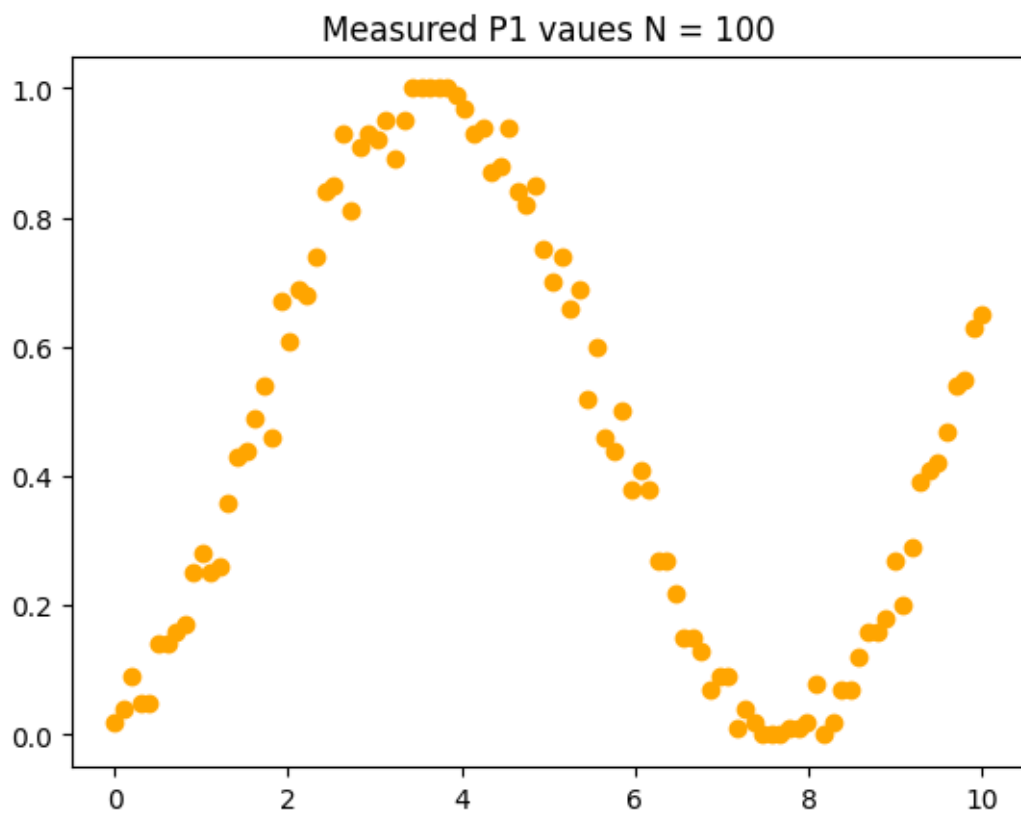
#### 3.1 3.A)

```
[ ]: for N in [1, 10, 100, 1000]:
      t_w = np.linspace(0, 10, 100)
      measured_results = []
      for t in t_w:
          true_p1 = P1(t, Omega, delta)
          p1_vals = np.sum(np.random.binomial(1, true_p1, N)) / N
          measured_results.append(p1_vals)

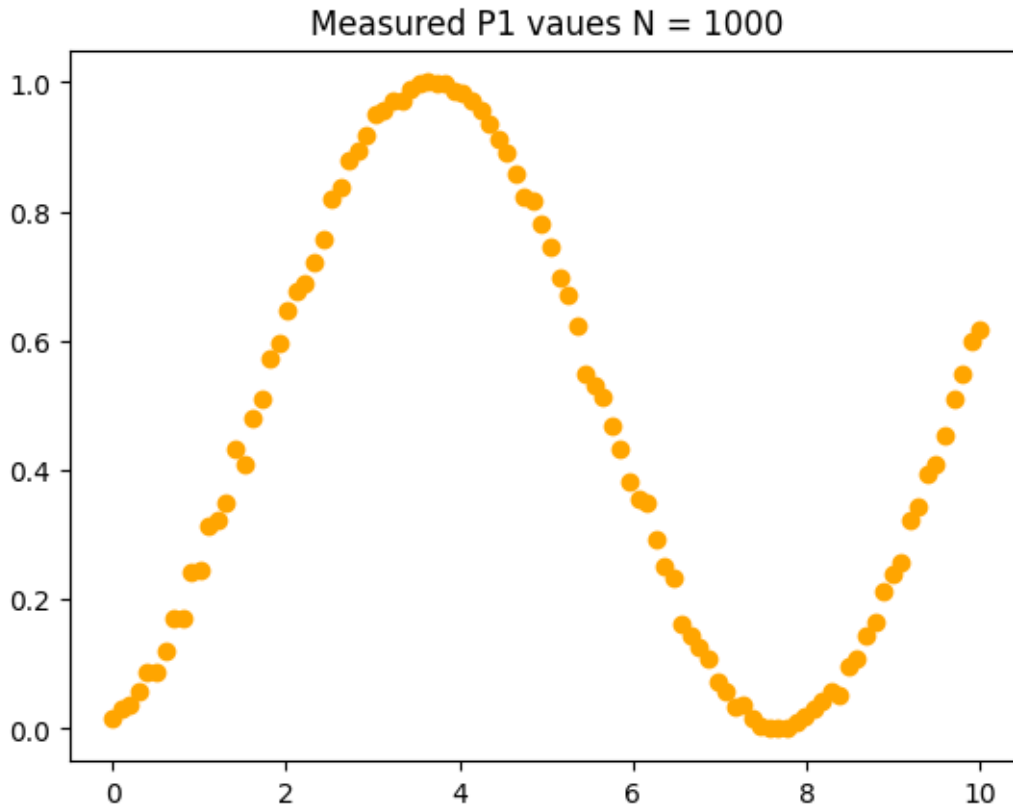
      plt.scatter(t_w, measured_results, label='data', color='orange')
      plt.title("Measured P1 vaues N = {}".format(N))
      plt.show()
```











### 3.2 3.B)

```
[ ]: def P1_field(t_w, Omega, gamma, phi):
    delta = (np.pi / 2 * t_w) + (2 * gamma * phi)
    Omega_p = np.sqrt(Omega**2 + delta**2)
    return 1 - (Omega**2) / (Omega_p**4) * (Omega_p * np.cos(delta * t_w / 2) *
    ↪ np.sin(np.pi/2) - 2 * delta * np.sin(delta * t_w / 2) * np.sin(np.pi/4)**2
    ↪)**2
```

```
[ ]: for N in [1, 10, 100, 1000]:
    t_w = np.linspace(1, 100, 1000)

    # First, find the best t by sampling around 2
    best_t = 0
    for t in t_w:
        true_p1 = P1(t, Omega, np.pi/4)
        p1_val = np.sum(np.random.binomial(1, true_p1, N)) / N
        if p1_val >= 0.5:
            best_t = t
            break
```

```

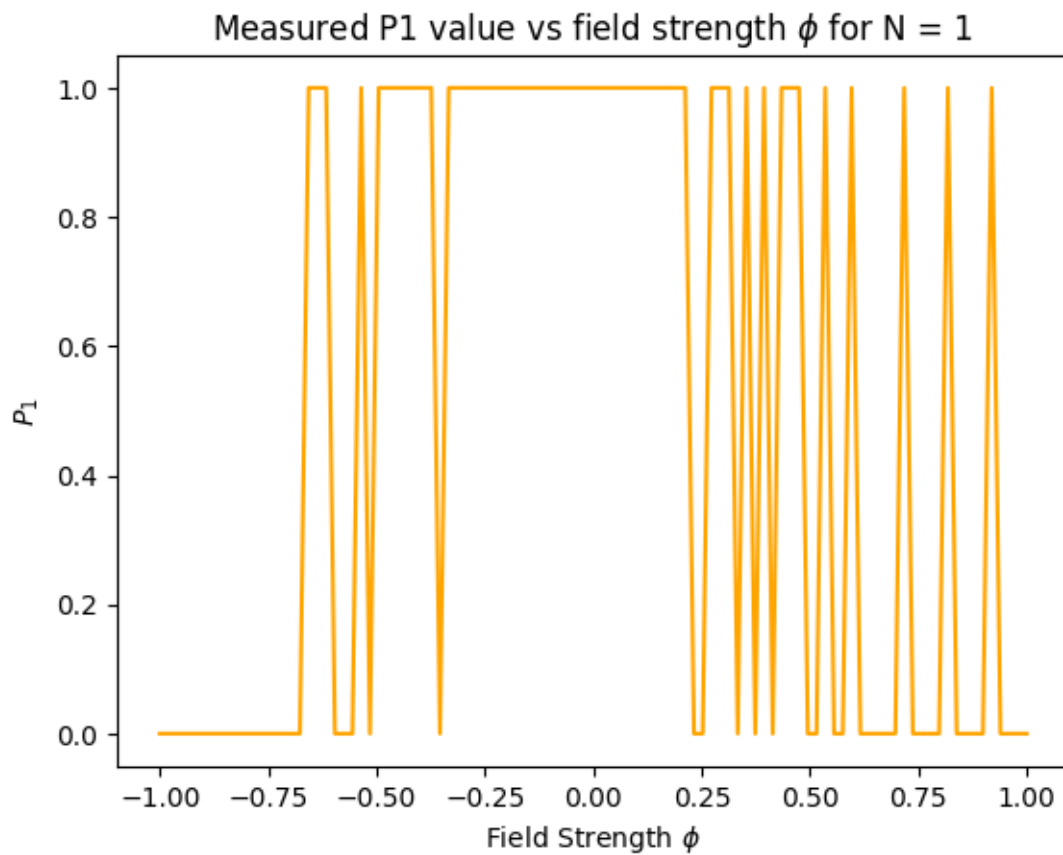
print("Best t for N = {} is {}".format(N, best_t))

# Now, lets plot our results relative to the strength of the field phi
gamma = 1
phis = np.linspace(-1, 1, 100)
measured_p1 = []
for phi in phis:
    true_p1 = P1_field(best_t, Omega, gamma, phi)
    p1_val = np.sum(np.random.binomial(1, true_p1, N)) / N
    measured_p1.append(p1_val)

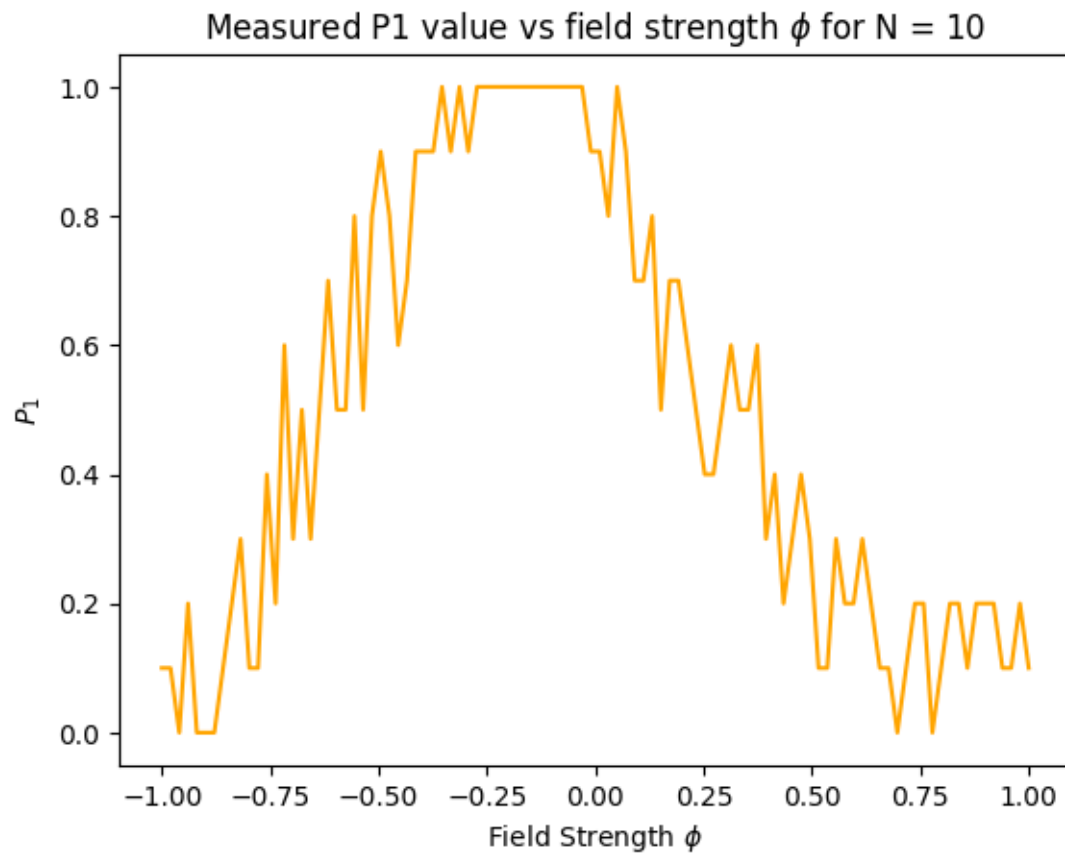
plt.plot(phis, measured_p1, label='data', color='orange')
plt.title("Measured P1 value vs field strength  $\phi$  for N = {}".format(N))
plt.xlabel("Field Strength  $\phi$ ")
plt.ylabel(" $P_1$ ")
plt.show()

```

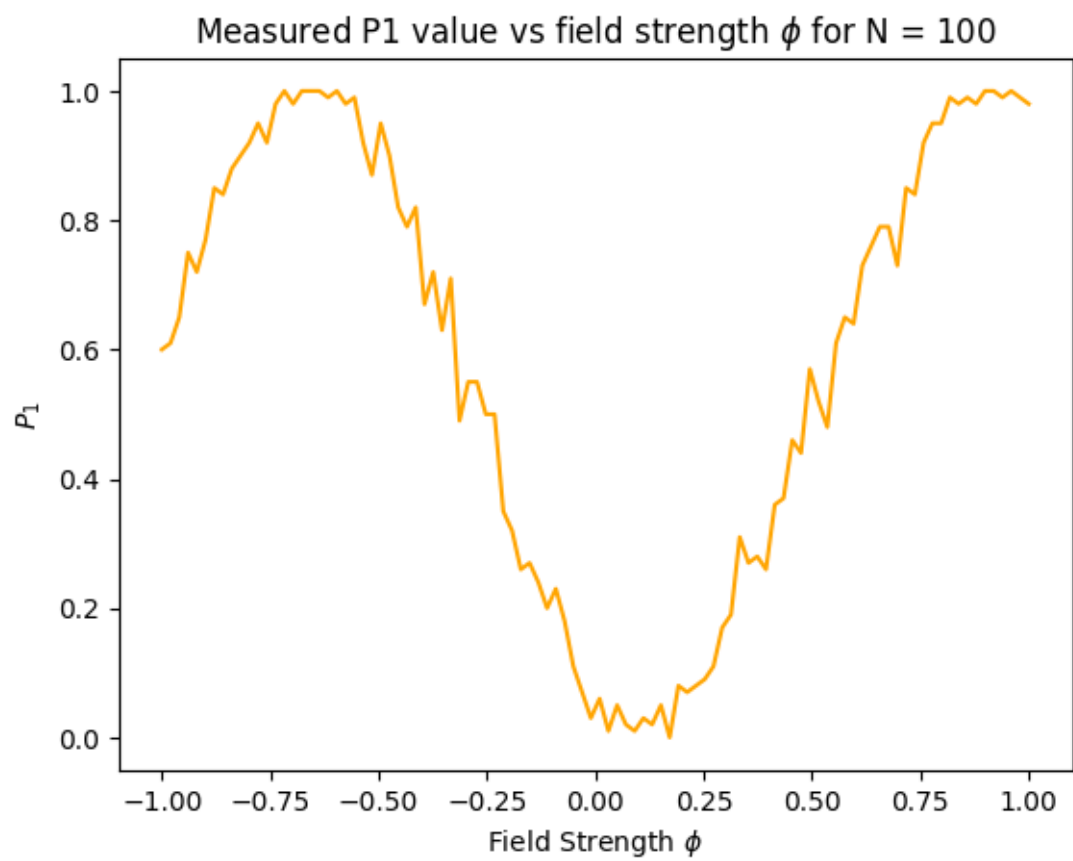
Best t for N = 1 is 1.2972972972972974



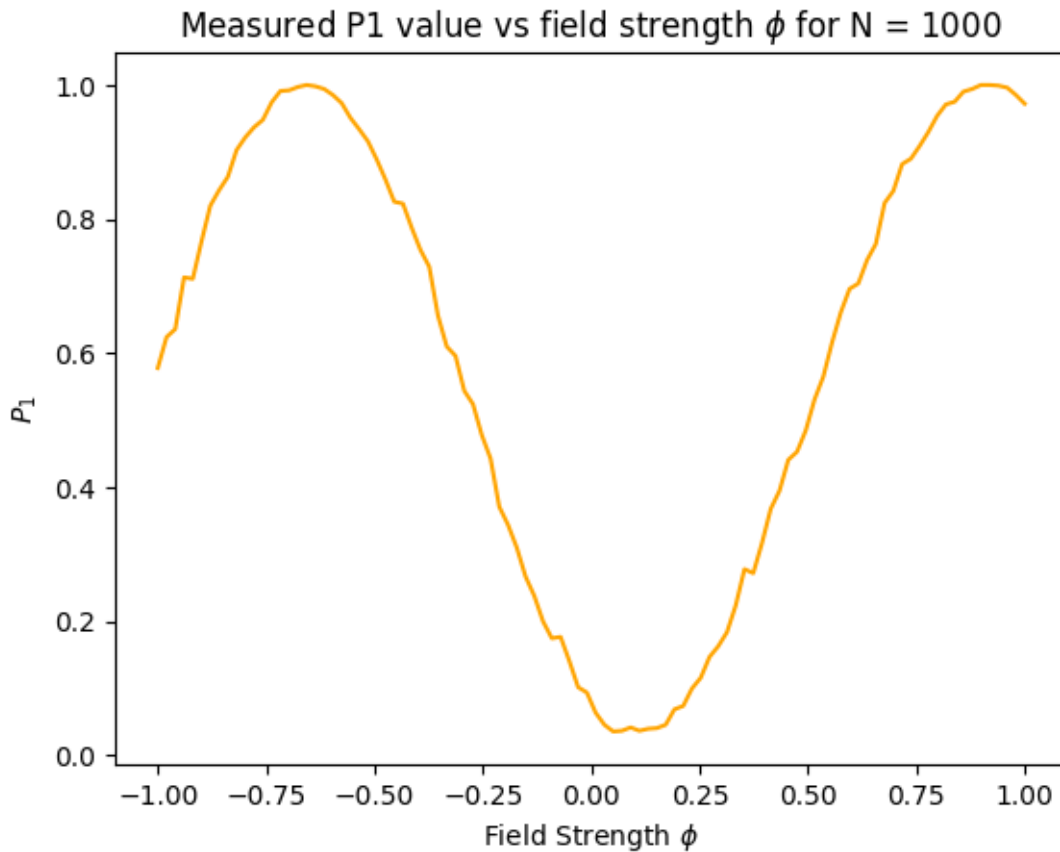
Best  $t$  for  $N = 10$  is 1.3963963963963963



Best  $t$  for  $N = 100$  is 1.7927927927927927



Best  $t$  for  $N = 1000$  is 1.7927927927927927



As we can see, we get a linear relationship between our field strength and our shifted slope probability at  $P_1 = 1/2$

### 3.3 3.C)

Now, let's plot our  $N=100$  solution with the std error bars

```
[ ]: N = 100
t_w = np.linspace(1, 100, 1000)

# First, find the best t by sampling around 2
best_t = 0
for t in t_w:
    true_p1 = P1(t, Omega, np.pi/4)
    p1_val = np.sum(np.random.binomial(1, true_p1, N)) / N
    if p1_val >= 0.5:
        best_t = t
        break
print("Best t for N = {} is {}".format(N, best_t))
```

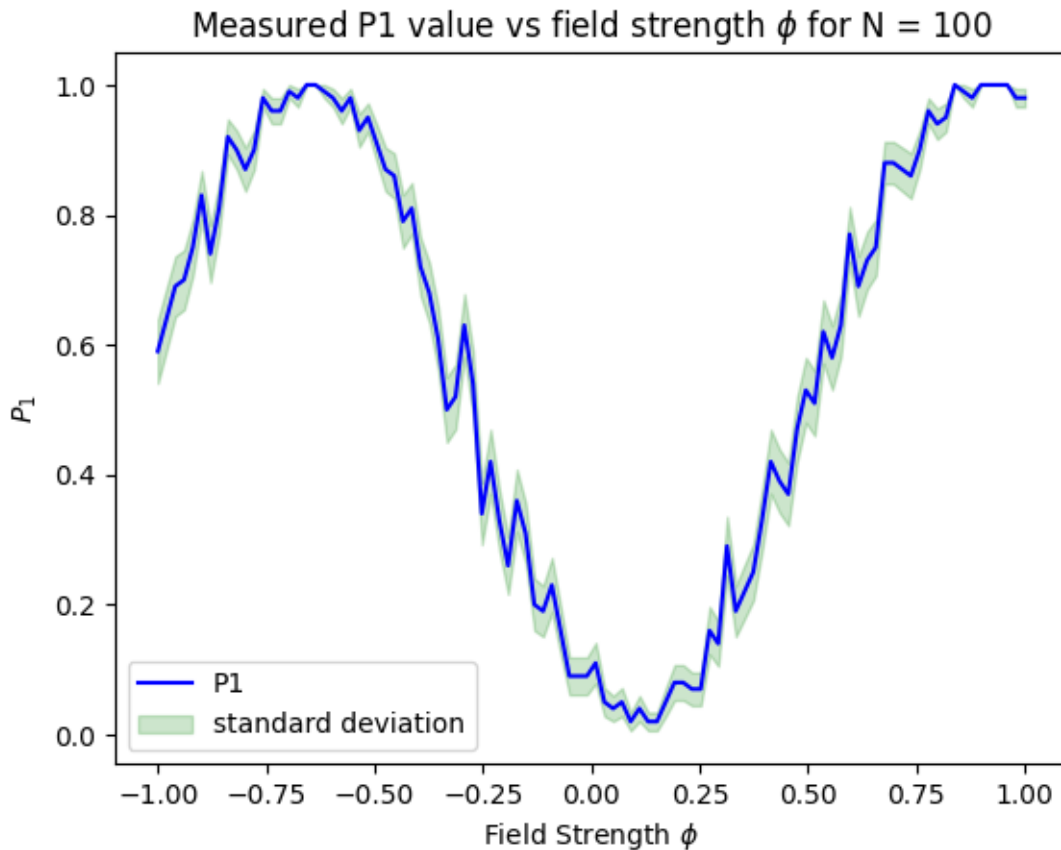
```

# Now, lets plot our results relative to the strengh of the field phi
gamma = 1
phis = np.linspace(-1, 1, 100)
measured_p1 = []
std_devs = []
for phi in phis:
    p1_field = P1_field(best_t, Omega, gamma, phi)
    p1_val = np.sum(np.random.binomial(1, p1_field, N)) / N
    measured_p1.append(p1_val)
    std_devs.append(np.sqrt(p1_val * (1 - p1_val) / N))

plt.plot(phis, measured_p1, label='P1', color='blue')
plt.fill_between(phis, np.array(measured_p1) - np.array(std_devs), np.
    ↪array(measured_p1) + np.array(std_devs), color='green', alpha=0.2,
    ↪label='standard deviation')
plt.title("Measured P1 value vs field strength  $\phi$  for N = {}".format(N))
plt.xlabel("Field Strength  $\phi$ ")
plt.ylabel(" $P_1$ ")
plt.legend()
plt.show()

```

Best t for N = 100 is 1.7927927927927927



From this graph we can see that the standard error is quite small around the peak and valley of the distribution. This is expected because we expect when the slope is near zero, the error will be small. However, when the slope is near 1 or -1, the error will be larger.

4 4)

4.1 4.A)

```
[ ]: def U(n, t, omega):
      return (-1.j * omega * t * (create(n) * destroy(n) + 0.5 * identity(n)) ).
      ↪ expm()
```

```
[ ]: n = 5
      m = 1
      alpha = 1
      omega = 2 * np.pi
      t_vals = np.linspace(0, 10, 100)
```

Here's what our evolved states look like from time 0 - 10 for 100 time steps.

```
[ ]: evolved_states = []
      for t in t_vals:
          state = U(n, t, omega) * coherent(n, alpha)
          state_p1_prob = np.abs((fock(n,0).dag() * state * fock(n,0))[0,0])**2

          evolved_states.append(state)

      print("Eveolved states:")
      print(evolved_states)
```

Eveolved states:

```
[Quantum object: dims = [[5], [1]], shape = (5, 1), type = ket
Qobj data =
[[0.60655682]
 [0.60628133]
 [0.4303874 ]
 [0.24104351]
 [0.14552147]], Quantum object: dims = [[5], [1]], shape = (5, 1), type = ket
Qobj data =
[[ 0.57627211-0.18926601j]
 [ 0.35167767-0.49386219j]
 [-0.00682851-0.43033322j]
 [-0.14597828-0.19181322j]
 [-0.13962682-0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
```

```

Qobj data =
[[ 0.48844216-0.35963235j]
 [-0.1982952 -0.57293635j]
 [-0.43017072+0.0136553j ]
 [-0.06423178+0.23232789j]
 [ 0.12242045+0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.35183747-0.4940866j ]
 [-0.58172267-0.17080919j]
 [ 0.02047866+0.42989992j]
 [ 0.22377706-0.08958682j]
 [-0.09529629-0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.18009908-0.57920246j]
 [-0.47656931+0.37477825j]
 [ 0.42952089-0.02729686j]
 [-0.20681133-0.1238186j ]
 [ 0.0604518 +0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.00962361-0.60648047j]
 [ 0.02884803+0.60559461j]
 [-0.03410818-0.42903373j]
 [ 0.02671683+0.23955831j]
 [-0.02070986-0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.1983853 -0.57319669j]
 [ 0.51003631+0.32778043j]
 [-0.42843857+0.04091092j]
 [ 0.17445139-0.16633906j]
 [-0.02070986+0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.36733668-0.48267477j]
 [ 0.56285214-0.22533201j]
 [ 0.04770336+0.42773555j]
 [-0.23801573-0.03808521j]
 [ 0.0604518 -0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.49960665-0.34395403j]
 [ 0.14293624-0.58919121j]
 [ 0.42692485-0.0544838j ]
 [ 0.11383788+0.21246861j]
 [-0.09529629+0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),

```



```

type = ket
Qobj data =
[[-0.581987 -0.1708868j ]
 [-0.39702983-0.45819685j]
 [-0.06125051-0.42600667j]
 [ 0.10013309-0.21926089j]
 [ 0.12242045-0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.60625144+0.01924479j]
 [-0.60353604+0.0576307j ]
 [-0.42498125+0.06800181j]
 [-0.23512102+0.05310442j]
 [-0.13962682+0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.56997697+0.20745465j]
 [-0.30314066+0.52505503j]
 [ 0.07473599+0.42384885j]
 [ 0.18465004+0.15493978j]
 [ 0.14552147+0.j          ]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.47678586+0.37494855j]
 [ 0.25185836+0.55149289j]
 [ 0.42260974-0.08145135j]
 [ 0.01146931-0.24077049j]
 [-0.13962682-0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.33598399+0.50500092j]
 [ 0.59532503+0.1147395j ]
 [-0.08814621-0.42126424j]
 [-0.19854189+0.1366861j ]
 [ 0.12242045+0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.16163151+0.58462503j]
 [ 0.43878643-0.41838202j]
 [-0.41981269+0.09481888j]
 [ 0.22900847+0.07521364j]
 [-0.09529629-0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.02886114+0.60586979j]
 [-0.08628283-0.60011026j]
 [ 0.10146767+0.41825545j]
 [-0.07883761-0.22778631j]

```

```

[ 0.0604518 +0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.21647177+0.56661375j]
 [-0.53888433-0.27781418j]
 [ 0.41659292-0.10809093j]
 [-0.13351883+0.20068556j]
 [-0.02070986-0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.38246602+0.47077693j]
 [-0.53888433+0.27781418j]
 [-0.11468697-0.41482552j]
 [ 0.24055821-0.01528792j]
 [-0.02070986+0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.51026807+0.32792937j]
 [-0.08628283+0.60011026j]
 [-0.41295368+0.12125414j]
 [-0.15784993-0.18216853j]
 [ 0.0604518 -0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.58711588+0.15233552j]
 [ 0.43878643+0.41838202j]
 [ 0.12779079+0.41097789j]
 [-0.04936732+0.23593397j]
 [-0.09529629+0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[0.60533562-0.03847021j]
 [0.59532503-0.1147395j ]
 [0.40889864-0.13429526j]
 [0.21764458-0.10359927j]
 [0.12242045-0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.5631079 -0.2254344j ]
 [ 0.25185836-0.55149289j]
 [-0.14076593-0.40671644j]
 [-0.21424801-0.11045253j]
 [-0.13962682+0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.46464948-0.38988721j]
 [-0.30314066-0.52505503j]
 [-0.40443186+0.14720116j]

```

```

[ 0.04185677+0.23738151j]
[ 0.14552147+0.j          ]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.3197922 -0.51540675j]
 [-0.60353604-0.0576307j ]
 [ 0.15359933+0.40204547j]
 [ 0.16355029-0.17706856j]
 [-0.13962682-0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.14300119-0.58945893j]
 [-0.39702983+0.45819685j]
 [ 0.39955786-0.15995884j]
 [-0.23995204-0.02291264j]
 [ 0.12242045+0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.0480696 -0.60464906j]
 [ 0.14293624+0.58919121j]
 [-0.16627807-0.39696966j]
 [ 0.12708428+0.2048208j ]
 [-0.09529629-0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.23434027-0.55946029j]
 [ 0.56285214+0.22533201j]
 [-0.39428154+0.17255545j]
 [ 0.08602511-0.22517028j]
 [ 0.0604518 +0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.39721024-0.45840506j]
 [ 0.51003631-0.32778043j]
 [ 0.17878939+0.39149415j]
 [-0.23127955+0.0679098j ]
 [-0.02070986-0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.52041569-0.31157452j]
 [ 0.02884803-0.60559461j]
 [ 0.3886082 -0.18497831j]
 [ 0.19410517+0.14291661j]
 [-0.02070986+0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.59165359-0.13363086j]
 [-0.47656931-0.37477825j]

```

```

[-0.19112067-0.38562443j]
[-0.00382439-0.24101317j]
[ 0.0604518 -0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.60381028+0.05765689j]
 [-0.58172267+0.17080919j]
 [-0.38254358+0.19721492j]
 [-0.18947299+0.14900321j]
 [-0.09529629+0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.55567183+0.24318715j]
 [-0.1982952 +0.57293635j]
 [ 0.20325952+0.37936642j]
 [ 0.23331775+0.06053759j]
 [ 0.12242045-0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.45204523+0.40443328j]
 [ 0.35167767+0.49386219j]
 [ 0.37609376-0.20925295j]
 [-0.09312598-0.22232751j]
 [-0.13962682+0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.30327841+0.52529361j]
 [ 0.60628133+0.j          ]
 [-0.2151937 -0.37272642j]
 [-0.12052175+0.2087498j ]
 [ 0.14552147+0.j          ]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.12422688+0.5936993j ]
 [ 0.35167767-0.49386219j]
 [-0.36926525+0.22108028j]
 [ 0.23910427-0.03051429j]
 [-0.13962682-0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.06722967+0.6028195j ]
 [-0.1982952 -0.57293635j]
 [ 0.2269112 +0.36571112j]
 [-0.16908597-0.1717903j ]
 [ 0.12242045+0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.25197281+0.55174349j]

```

```

[-0.58172267-0.17080919j]
[ 0.36206492-0.23268499j]
[-0.03430407+0.23859003j]
[-0.09529629-0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.41155451+0.44557161j]
[-0.47656931+0.37477825j]
[-0.23840021-0.35832757j]
[ 0.21063572-0.11719457j]
[ 0.0604518 +0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.5300393 +0.29490594j]
[ 0.02884803+0.60559461j]
[-0.35450002+0.24405542j]
[-0.22082199-0.0966417j ]
[-0.02070986-0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.59559555+0.11479163j]
[ 0.51003631+0.32778043j]
[ 0.24964918+0.35058322j]
[ 0.05682816+0.23424887j]
[-0.02070986+0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[0.60167695-0.07678552j]
[0.56285214-0.22533201j]
[0.34657817-0.2551801j ]
[0.15199063-0.18708507j]
[0.0604518 -0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.54767625-0.26069503j]
[ 0.14293624-0.58919121j]
[-0.26064678-0.34248587j]
[-0.24092215-0.00764781j]
[-0.09529629+0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.43898581-0.41857213j]
[-0.39702983-0.45819685j]
[-0.33830735+0.26604784j]
[ 0.13981895+0.19634824j]
[ 0.12242045-0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =

```

```

[[ 0.28645924-0.53465155j]
 [-0.60353604+0.0576307j ]
 [ 0.27138192+0.33404366j]
 [ 0.07157073-0.23017298j]
 [-0.13962682+0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.10532749-0.59734186j]
 [-0.30314066+0.52505503j]
 [ 0.32969587-0.27664769j]
 [-0.2265068 +0.08244173j]
 [ 0.14552147+0.j          ]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.08632204-0.60038294j]
 [ 0.25185836+0.55149289j]
 [-0.28184381-0.32526509j]
 [ 0.2027787 +0.13031796j]
 [-0.13962682-0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.26935163-0.54347113j]
 [ 0.59532503+0.1147395j ]
 [-0.32075243+0.28696898j]
 [-0.01910269-0.24028537j]
 [ 0.12242045+0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.42548438-0.43228951j]
 [ 0.43878643-0.41838202j]
 [ 0.2920219 +0.31615901j]
 [-0.17964116+0.16072034j]
 [-0.09529629-0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.5391292 -0.27794042j]
 [-0.08628283-0.60011026j]
 [ 0.31148601-0.29700131j]
 [ 0.23668754+0.04561778j]
 [ 0.0604518 +0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.59893779-0.09583682j]
 [-0.53888433-0.27781418j]
 [-0.30190595-0.30673459j]
 [-0.10703937-0.21597348j]
 [-0.02070986-0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket

```

```

Qobj data =
[[-0.59893779+0.09583682j]
 [-0.53888433+0.27781418j]
 [-0.30190595+0.30673459j]
 [-0.10703937+0.21597348j]
 [-0.02070986+0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.5391292 +0.27794042j]
 [-0.08628283+0.60011026j]
 [ 0.31148601+0.29700131j]
 [ 0.23668754-0.04561778j]
 [ 0.0604518 -0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.42548438+0.43228951j]
 [ 0.43878643+0.41838202j]
 [ 0.2920219 -0.31615901j]
 [-0.17964116-0.16072034j]
 [-0.09529629+0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.26935163+0.54347113j]
 [ 0.59532503-0.1147395j ]
 [-0.32075243-0.28696898j]
 [-0.01910269+0.24028537j]
 [ 0.12242045-0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.08632204+0.60038294j]
 [ 0.25185836-0.55149289j]
 [-0.28184381+0.32526509j]
 [ 0.2027787 -0.13031796j]
 [-0.13962682+0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.10532749+0.59734186j]
 [-0.30314066-0.52505503j]
 [ 0.32969587+0.27664769j]
 [-0.2265068 -0.08244173j]
 [ 0.14552147+0.j          ]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.28645924+0.53465155j]
 [-0.60353604-0.0576307j ]
 [ 0.27138192-0.33404366j]
 [ 0.07157073+0.23017298j]
 [-0.13962682-0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),

```

```

type = ket
Qobj data =
[[ 0.43898581+0.41857213j]
 [-0.39702983+0.45819685j]
 [-0.33830735-0.26604784j]
 [ 0.13981895-0.19634824j]
 [ 0.12242045+0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.54767625+0.26069503j]
 [ 0.14293624+0.58919121j]
 [-0.26064678+0.34248587j]
 [-0.24092215+0.00764781j]
 [-0.09529629-0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[0.60167695+0.07678552j]
 [0.56285214+0.22533201j]
 [0.34657817+0.2551801j ]
 [0.15199063+0.18708507j]
 [0.0604518 +0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.59559555-0.11479163j]
 [ 0.51003631-0.32778043j]
 [ 0.24964918-0.35058322j]
 [ 0.05682816-0.23424887j]
 [-0.02070986-0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.5300393 -0.29490594j]
 [ 0.02884803-0.60559461j]
 [-0.35450002-0.24405542j]
 [-0.22082199+0.0966417j ]
 [-0.02070986+0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.41155451-0.44557161j]
 [-0.47656931-0.37477825j]
 [-0.23840021+0.35832757j]
 [ 0.21063572+0.11719457j]
 [ 0.0604518 -0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.25197281-0.55174349j]
 [-0.58172267+0.17080919j]
 [ 0.36206492+0.23268499j]
 [-0.03430407-0.23859003j]

```



```

[-0.09529629+0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.06722967-0.6028195j ]
 [-0.1982952 +0.57293635j]
 [ 0.2269112 -0.36571112j]
 [-0.16908597+0.1717903j ]
 [ 0.12242045-0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.12422688-0.5936993j ]
 [ 0.35167767+0.49386219j]
 [-0.36926525-0.22108028j]
 [ 0.23910427+0.03051429j]
 [-0.13962682+0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.30327841-0.52529361j]
 [ 0.60628133+0.j          ]
 [-0.2151937 +0.37272642j]
 [-0.12052175-0.2087498j ]
 [ 0.14552147+0.j          ]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.45204523-0.40443328j]
 [ 0.35167767-0.49386219j]
 [ 0.37609376+0.20925295j]
 [-0.09312598+0.22232751j]
 [-0.13962682-0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.55567183-0.24318715j]
 [-0.1982952 -0.57293635j]
 [ 0.20325952-0.37936642j]
 [ 0.23331775-0.06053759j]
 [ 0.12242045+0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.60381028-0.05765689j]
 [-0.58172267-0.17080919j]
 [-0.38254358-0.19721492j]
 [-0.18947299-0.14900321j]
 [-0.09529629-0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.59165359+0.13363086j]
 [-0.47656931+0.37477825j]
 [-0.19112067+0.38562443j]

```

```

[-0.00382439+0.24101317j]
[ 0.0604518 +0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.52041569+0.31157452j]
 [ 0.02884803+0.60559461j]
 [ 0.3886082 +0.18497831j]
 [ 0.19410517-0.14291661j]
 [-0.02070986-0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.39721024+0.45840506j]
 [ 0.51003631+0.32778043j]
 [ 0.17878939-0.39149415j]
 [-0.23127955-0.0679098j ]
 [-0.02070986+0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.23434027+0.55946029j]
 [ 0.56285214-0.22533201j]
 [-0.39428154-0.17255545j]
 [ 0.08602511+0.22517028j]
 [ 0.0604518 -0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.0480696 +0.60464906j]
 [ 0.14293624-0.58919121j]
 [-0.16627807+0.39696966j]
 [ 0.12708428-0.2048208j ]
 [-0.09529629+0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.14300119+0.58945893j]
 [-0.39702983-0.45819685j]
 [ 0.39955786+0.15995884j]
 [-0.23995204+0.02291264j]
 [ 0.12242045-0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.3197922 +0.51540675j]
 [-0.60353604+0.0576307j ]
 [ 0.15359933-0.40204547j]
 [ 0.16355029+0.17706856j]
 [-0.13962682+0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.46464948+0.38988721j]
 [-0.30314066+0.52505503j]

```

```

[-0.40443186-0.14720116j]
[ 0.04185677-0.23738151j]
[ 0.14552147+0.j          ]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.5631079 +0.2254344j ]
 [ 0.25185836+0.55149289j]
 [-0.14076593+0.40671644j]
 [-0.21424801+0.11045253j]
 [-0.13962682-0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[0.60533562+0.03847021j]
 [0.59532503+0.1147395j ]
 [0.40889864+0.13429526j]
 [0.21764458+0.10359927j]
 [0.12242045+0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.58711588-0.15233552j]
 [ 0.43878643-0.41838202j]
 [ 0.12779079-0.41097789j]
 [-0.04936732-0.23593397j]
 [-0.09529629-0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.51026807-0.32792937j]
 [-0.08628283-0.60011026j]
 [-0.41295368-0.12125414j]
 [-0.15784993+0.18216853j]
 [ 0.0604518 +0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.38246602-0.47077693j]
 [-0.53888433-0.27781418j]
 [-0.11468697+0.41482552j]
 [ 0.24055821+0.01528792j]
 [-0.02070986-0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.21647177-0.56661375j]
 [-0.53888433+0.27781418j]
 [ 0.41659292+0.10809093j]
 [-0.13351883-0.20068556j]
 [-0.02070986+0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.02886114-0.60586979j]

```

```

[-0.08628283+0.60011026j]
[ 0.10146767-0.41825545j]
[-0.07883761+0.22778631j]
[ 0.0604518 -0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.16163151-0.58462503j]
[ 0.43878643+0.41838202j]
[-0.41981269-0.09481888j]
[ 0.22900847-0.07521364j]
[-0.09529629+0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.33598399-0.50500092j]
[ 0.59532503-0.1147395j ]
[-0.08814621+0.42126424j]
[-0.19854189-0.1366861j ]
[ 0.12242045-0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.47678586-0.37494855j]
[ 0.25185836-0.55149289j]
[ 0.42260974+0.08145135j]
[ 0.01146931+0.24077049j]
[-0.13962682+0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.56997697-0.20745465j]
[-0.30314066-0.52505503j]
[ 0.07473599-0.42384885j]
[ 0.18465004-0.15493978j]
[ 0.14552147+0.j          ]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.60625144-0.01924479j]
[-0.60353604-0.0576307j ]
[-0.42498125-0.06800181j]
[-0.23512102-0.05310442j]
[-0.13962682-0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.581987 +0.1708868j ]
[-0.39702983+0.45819685j]
[-0.06125051+0.42600667j]
[ 0.10013309+0.21926089j]
[ 0.12242045+0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =

```

```

[[-0.49960665+0.34395403j]
 [ 0.14293624+0.58919121j]
 [ 0.42692485+0.0544838j ]
 [ 0.11383788-0.21246861j]
 [-0.09529629-0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.36733668+0.48267477j]
 [ 0.56285214+0.22533201j]
 [ 0.04770336-0.42773555j]
 [-0.23801573+0.03808521j]
 [ 0.0604518 +0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.1983853 +0.57319669j]
 [ 0.51003631-0.32778043j]
 [-0.42843857-0.04091092j]
 [ 0.17445139+0.16633906j]
 [-0.02070986-0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[-0.00962361+0.60648047j]
 [ 0.02884803-0.60559461j]
 [-0.03410818+0.42903373j]
 [ 0.02671683-0.23955831j]
 [-0.02070986+0.14404027j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.18009908+0.57920246j]
 [-0.47656931-0.37477825j]
 [ 0.42952089+0.02729686j]
 [-0.20681133+0.1238186j ]
 [ 0.0604518 -0.13237098j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.35183747+0.4940866j ]
 [-0.58172267+0.17080919j]
 [ 0.02047866-0.42989992j]
 [ 0.22377706+0.08958682j]
 [-0.09529629+0.10997779j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[ 0.48844216+0.35963235j]
 [-0.1982952 +0.57293635j]
 [-0.43017072-0.0136553j ]
 [-0.06423178-0.23232789j]
 [ 0.12242045-0.07867484j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket

```

```

Qobj data =
[[ 0.57627211+0.18926601j]
 [ 0.35167767+0.49386219j]
 [-0.00682851+0.43033322j]
 [-0.14597828+0.19181322j]
 [-0.13962682+0.04099813j]], Quantum object: dims = [[5], [1]], shape = (5, 1),
type = ket
Qobj data =
[[0.60655682]
 [0.60628133]
 [0.4303874 ]
 [0.24104351]
 [0.14552147]]]

```

## 4.2 4.B)

First, lets create a helper function that gives us  $\sigma_x$  and  $\sigma_p$  for our given time evolved hamiltonian

```

[ ]: def get_std_dev(op):
    std_devs = []
    for t in t_vals:
        state = U(n, t, omega) * coherent(n, alpha)

        # Calculate the standard deviation of the operator
        std_dev = np.sqrt(expect(op*op, state) - expect(op, state)**2)
        std_devs.append(std_dev)

    return std_devs

```

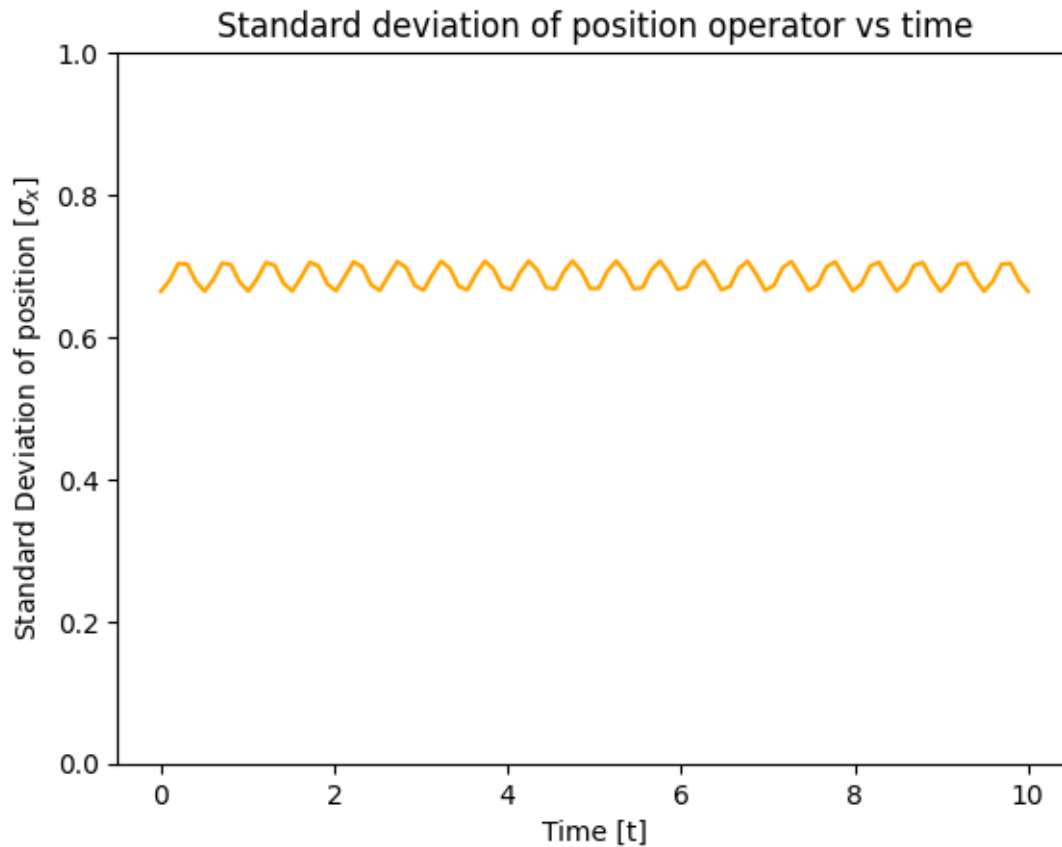
Now, lets see what  $\sigma_x$  vs time looks like:

```

[ ]: std_devs_x = get_std_dev(position(n))

plt.plot(t_vals, std_devs_x, label='data', color='orange')
plt.title("Standard deviation of position operator vs time")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of position [$\sigma_x$]")
plt.ylim(0, 1)
plt.show()

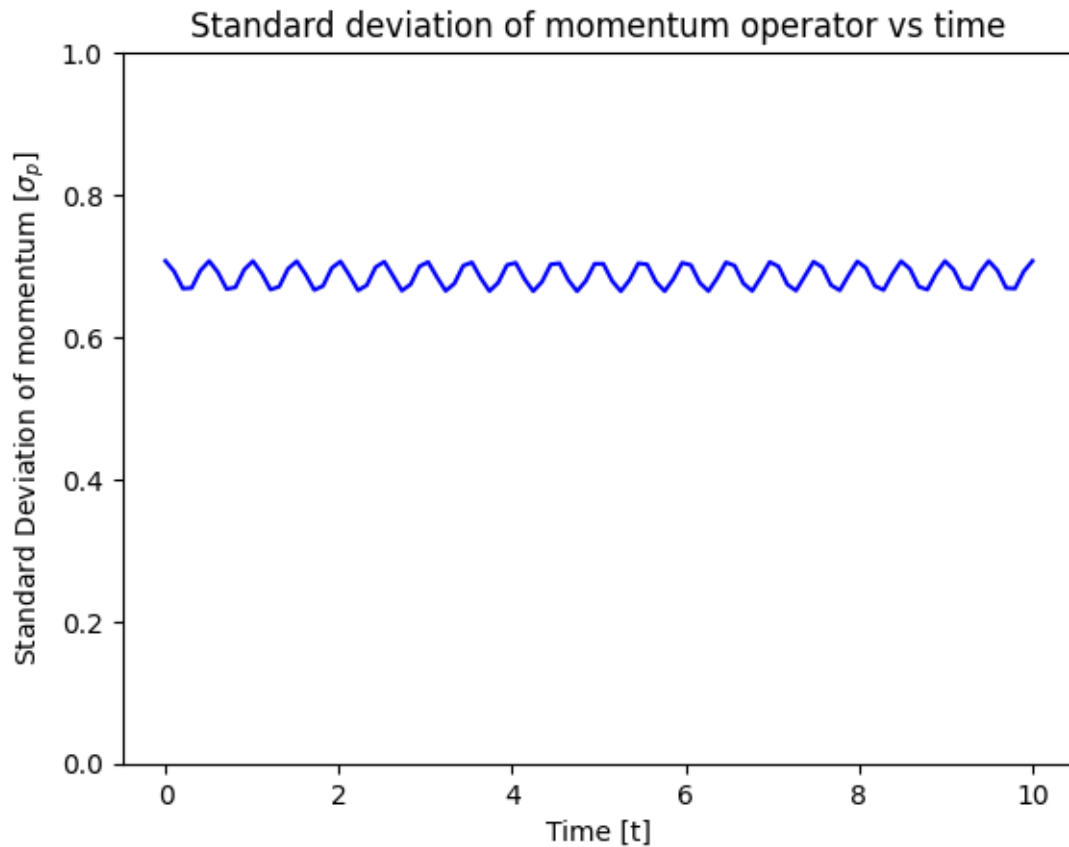
```



And here's see what  $\sigma_p$  vs time looks like:

```
[ ]: std_devs_p = get_std_dev(momentum(n))

plt.plot(t_vals, std_devs_p, label='data', color='blue')
plt.title("Standard deviation of momentum operator vs time")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of momentum [ $\sigma_p$ ]")
plt.ylim(0, 1)
plt.show()
```

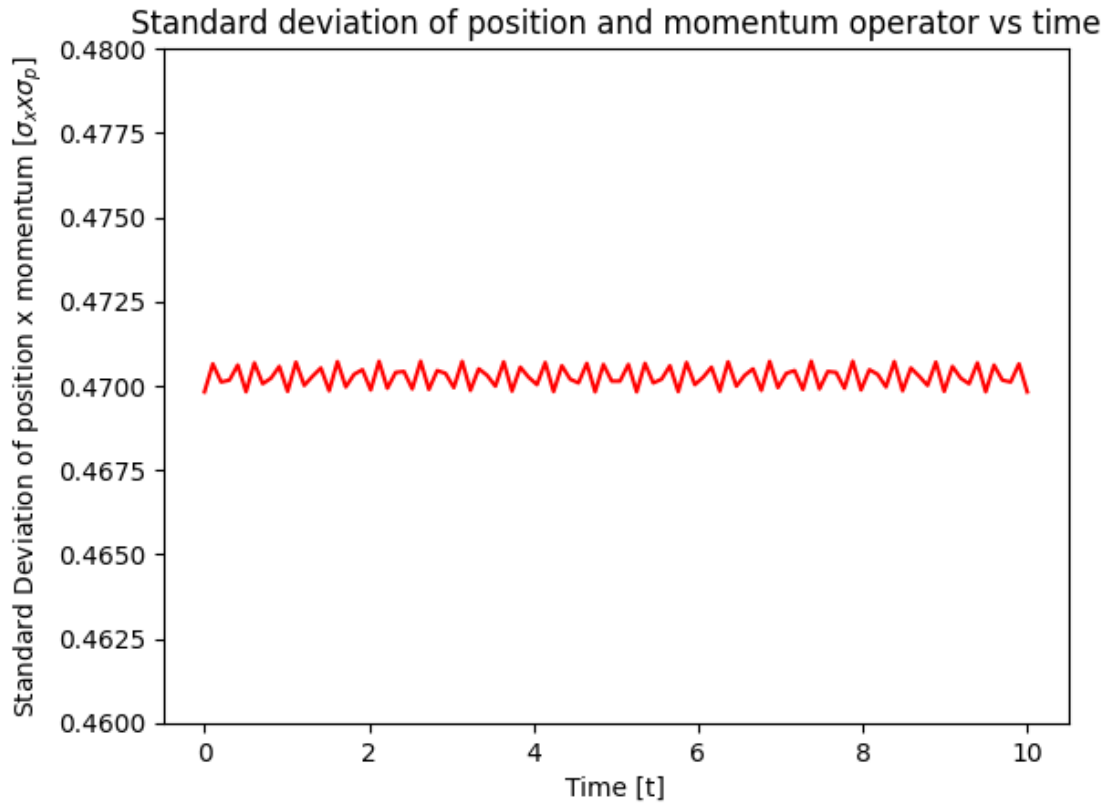


Lastly, lets see what  $\sigma_x * \sigma_p$  looks like:

```
[ ]: std_devs_xp = np.multiply(std_devs_x, std_devs_p)

plt.plot(t_vals, std_devs_xp, label='data', color='red')
plt.title("Standard deviation of position and momentum operator vs time")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of position x momentum [ $\sigma_x \times \sigma_p$ ]")
plt.ylim(0.46, 0.48)
plt.show()
```





### 4.3 4.C)

```
[ ]: def get_std_dev_squeezed(op):
    r = 1
    std_devs = []
    for t in t_vals:
        state = squeeze(n, r) * U(n, t, omega) * coherent(n, alpha)

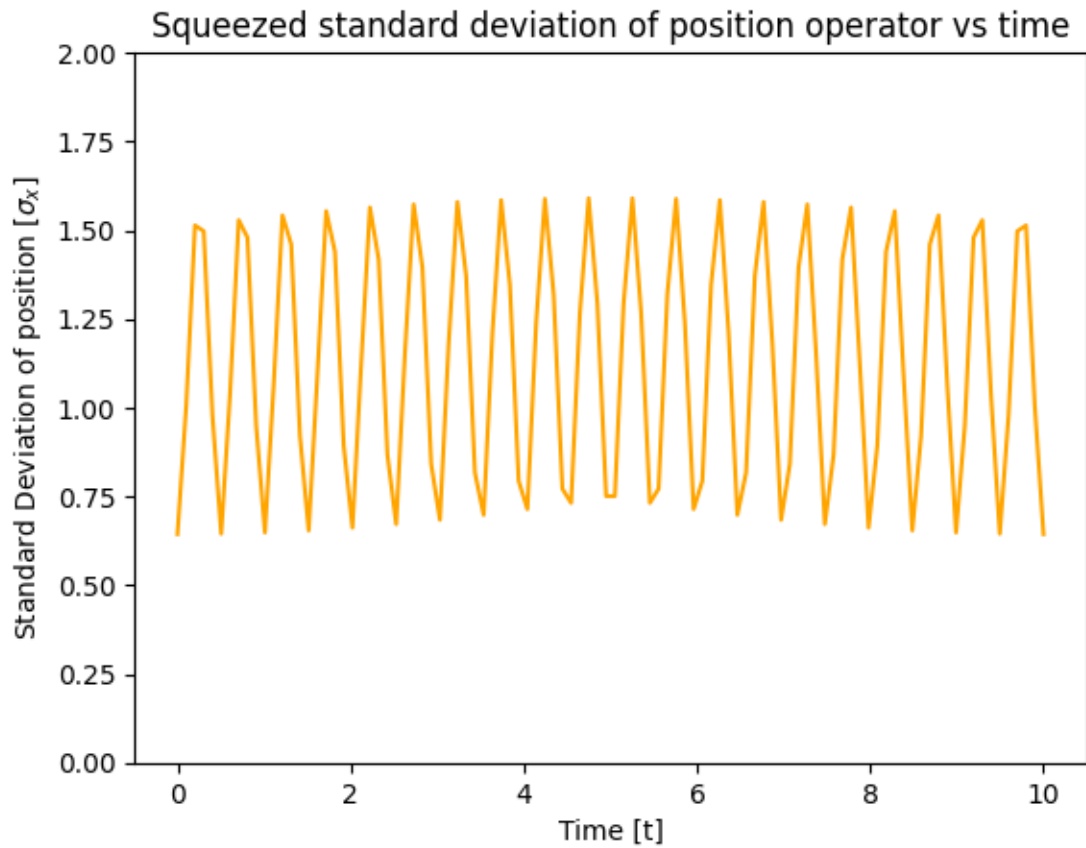
        # Calculate the standard deviation of the operator
        std_dev = np.sqrt(expect(op*op, state) - expect(op, state)**2)
        std_devs.append(std_dev)

    return std_devs
```

```
[ ]: squeezed_std_devs_x = get_std_dev_squeezed(position(n))

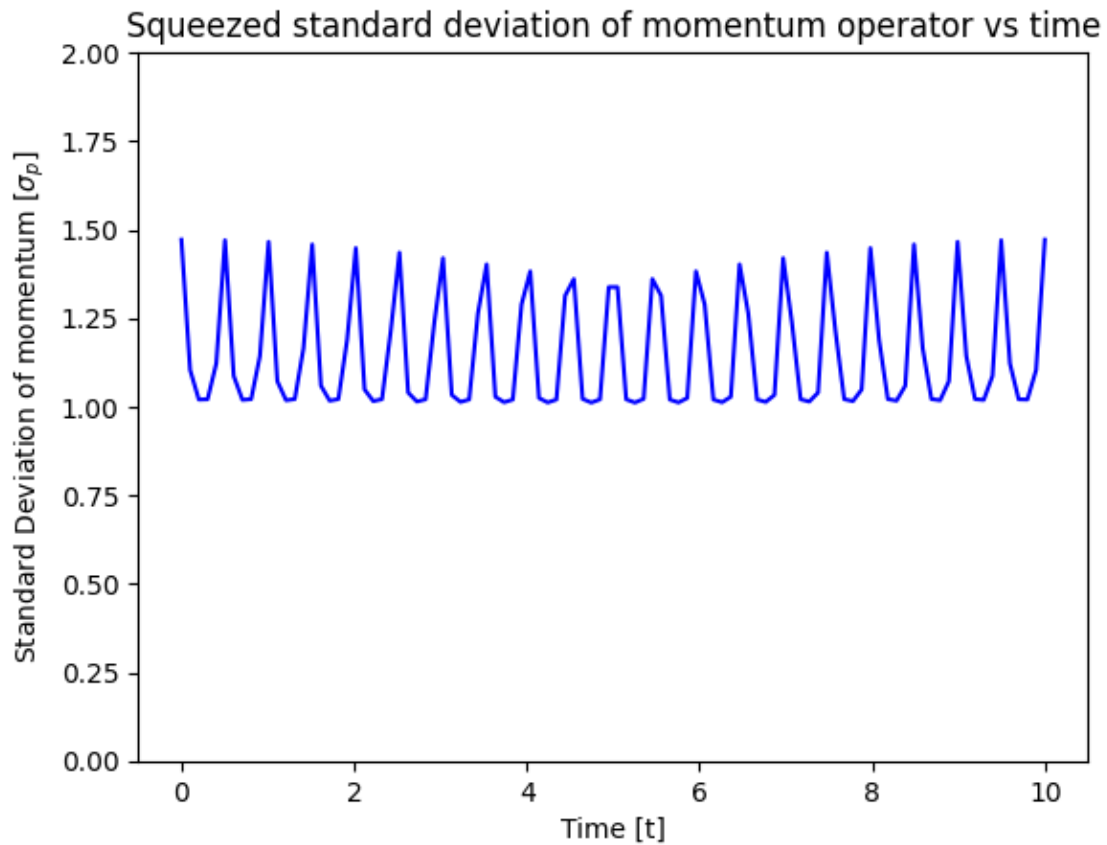
plt.plot(t_vals, squeezed_std_devs_x, label='data', color='orange')
plt.title("Squeezed standard deviation of position operator vs time")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of position [$\sigma_x$]")
plt.ylim(0, 2)
```

```
plt.show()
```



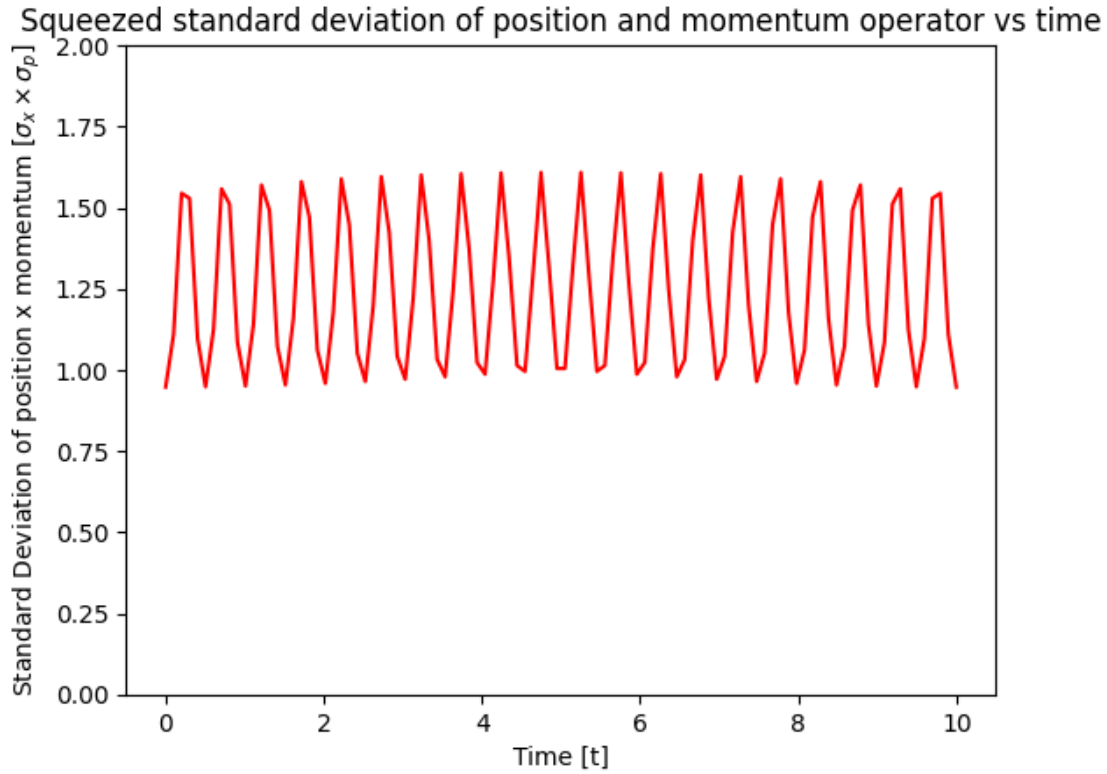
```
[ ]: squeezed_std_devs_p = get_std_dev_squeezed(momentum(n))

plt.plot(t_vals, squeezed_std_devs_p, label='data', color='blue')
plt.title("Squeezed standard deviation of momentum operator vs time")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of momentum [ $\sigma_p$ ]")
plt.ylim(0, 2)
plt.show()
```



```
[ ]: squeezed_std_devs_xp = np.multiply(squeezed_std_devs_x, squeezed_std_devs_p)

plt.plot(t_vals, squeezed_std_devs_xp, label='data', color='red')
plt.title("Squeezed standard deviation of position and momentum operator vs␣
↪time")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of position x momentum [ $\sigma_x$  ␣\times␣
↪\sigma_p$]")
plt.ylim(0, 2)
plt.show()
```



We can clearly see that the values of  $\sigma_x$  and  $\sigma_p$  are no longer the same. This is due to the fact that we ‘squeezed’ the system. This means that as we evolve the system, the values of  $\sigma_x$  and  $\sigma_p$  will oscillate between the same and different. This allows us to get a better measurement for position or momentum at a given time.

#### 4.4 4.D)

Lets create a fuction that will give us the driven QHO with respect to the position operator:

```
[ ]: def H_B1_drive_coeff(t, args):
    omega = args['omega']
    B1 = args['B1']

    return B1 * np.exp(np.sin(2 * omega * t))

def H_B2_drive_coeff(t, args):
    omega = args['omega']
    B2 = args['B2']

    return B2 * np.exp( np.sin(2 * omega * t))
```

Next, lets plug this into a solver starting at the ground state. Our drive will create a coherent state and squeeze our QHO - which will offset the values of  $\sigma_x$  and  $\sigma_p$  over time.

```
[ ]: psi0 = fock(n, 0)
t_vals = np.linspace(0, 100, 1000)
B1 = 1
B2 = 1

QHO_H = create(n) * destroy(n) + 0.5 * identity(n)
H_B1 = position(n)
H_B2 = position(n)**2

H = [QHO_H, [H_B1, H_B1_drive_coeff], [H_B2, H_B2_drive_coeff]]

res = sesolve(H, psi0, t_vals, args={'n': n, 'omega': omega, 'B1': B1, 'B2': B2})
res_states = res.states
```

```
[ ]: def get_driven_QHO_exp_val(op, states):
    exp_vals = []
    for state in states:
        exp_val = expect(op, state)
        exp_vals.append(exp_val)

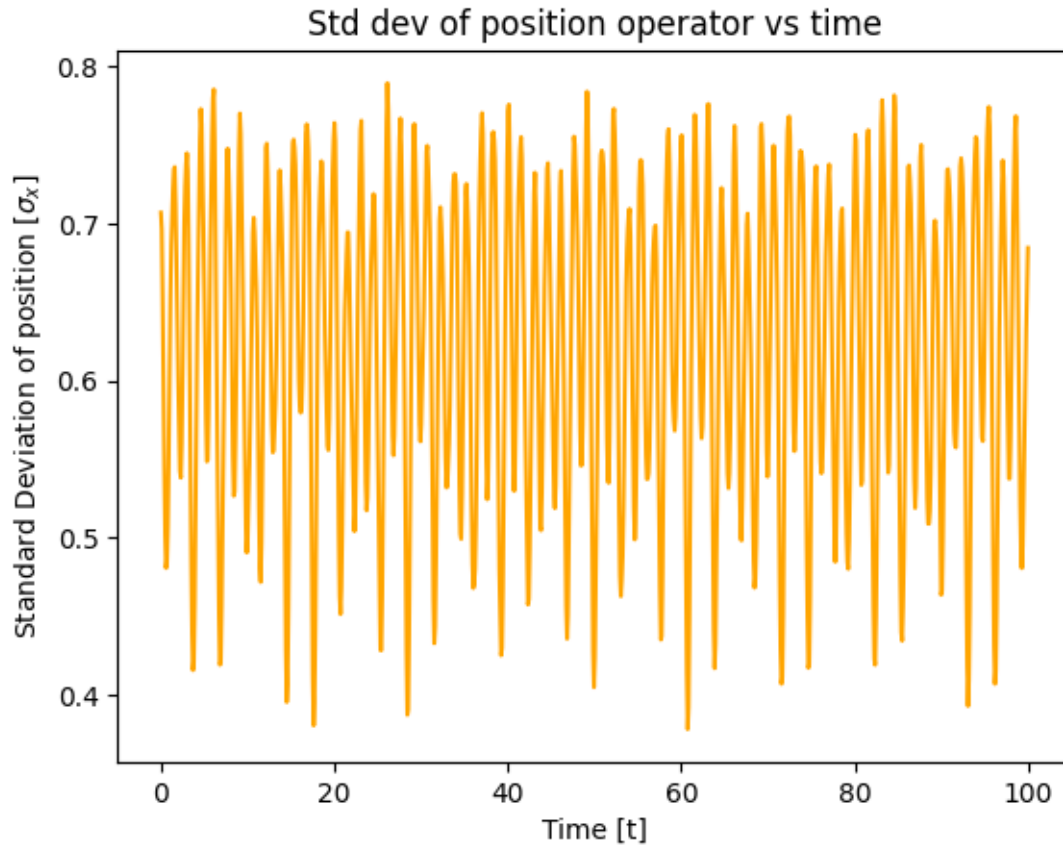
    return exp_vals
```

```
[ ]: def get_driven_QHO_std_dev(op, states):
    std_devs = []
    for state in states:
        # Calculate the standard deviation of the operator
        std_dev = np.sqrt(expect(op*op, state) - expect(op, state)**2)
        std_devs.append(std_dev)

    return std_devs
```

First, lets look at  $\sigma_x$  vs time:

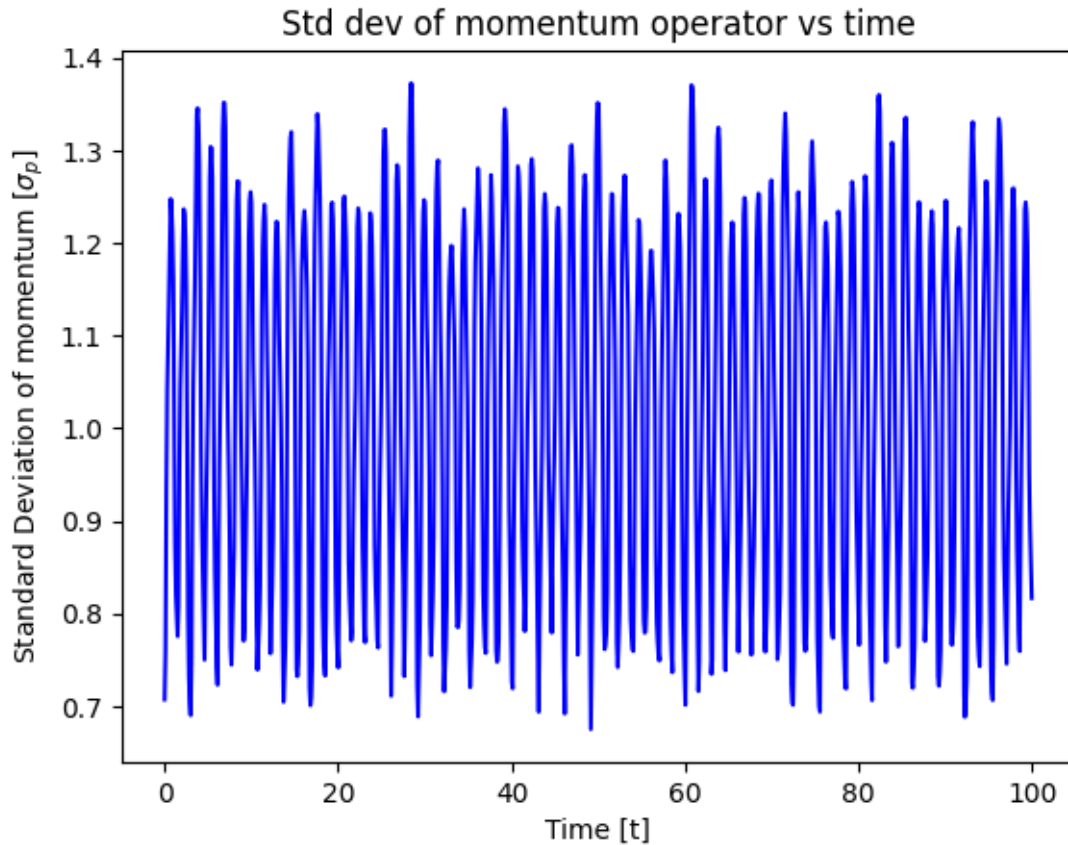
```
[ ]: driven_x_std_dev = get_driven_QHO_std_dev(position(n), res_states)
plt.plot(t_vals, driven_x_std_dev, label='data', color='orange')
plt.title("Std dev of position operator vs time")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of position [ $\sigma_x$ ]")
plt.show()
```



We see that our  $\sigma_x$  is oscillating between the same and different values. This is due to the fact that we are squeezing the system. This means that we are offsetting the values of  $\sigma_x$  and  $\sigma_p$  over time.

Next, let's look at  $\sigma_p$  vs time:

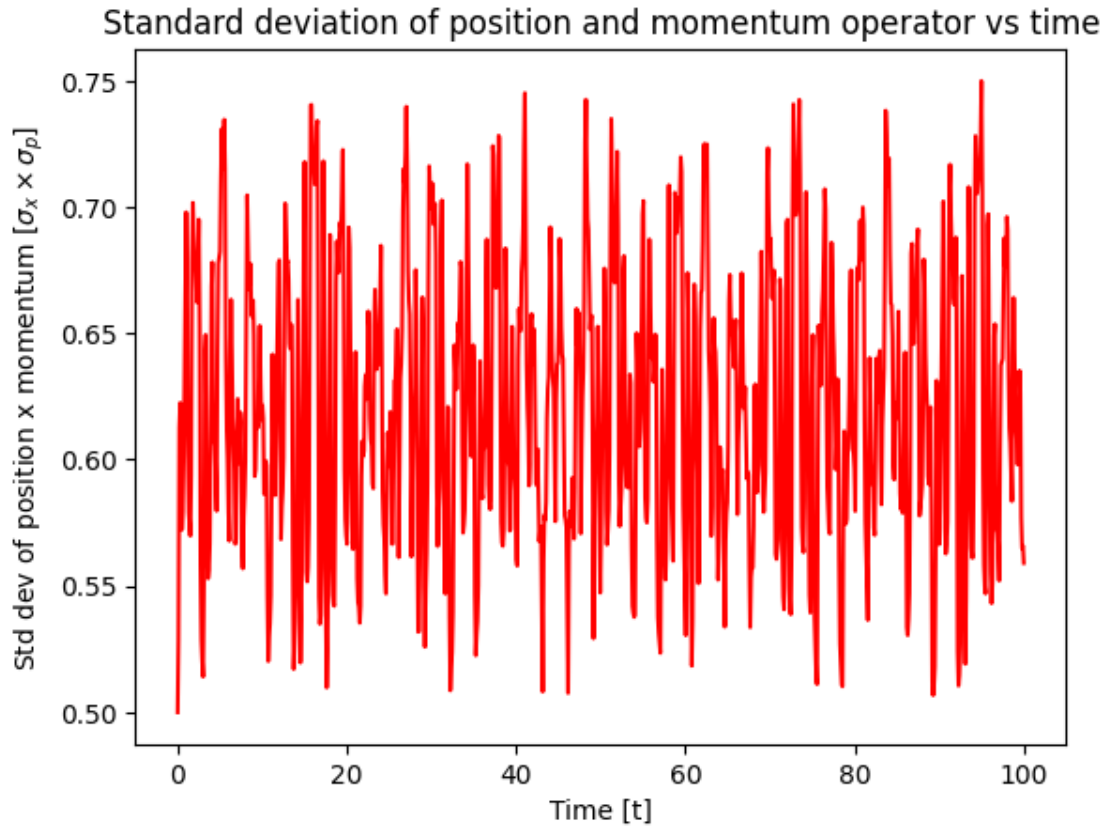
```
[ ]: driven_p_std_dev = get_driven_QHO_std_dev(momentum(n), res_states)
plt.plot(t_vals, driven_p_std_dev, label='data', color='blue')
plt.title("Std dev of momentum operator vs time")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of momentum [\\sigma_p]")
plt.show()
```



We can clearly see that  $\sigma_p$  also changes over time.

Lastly, lets look at the combined relation:

```
[ ]: driven_xp_std_dev = np.multiply(driven_x_std_dev, driven_p_std_dev)
plt.plot(t_vals, driven_xp_std_dev, label='data', color='red')
plt.title("Standard deviation of position and momentum operator vs time")
plt.xlabel("Time [t]")
plt.ylabel("Std dev of position x momentum [ $\sigma_x \times \sigma_p$ ]")
plt.show()
```



From these results, we can clearly see that our system is in fact being ‘squeezed’. This means that we are offsetting the values of  $\sigma_x$  and  $\sigma_p$  over time.

#### 4.5 4.E)

Lastly, lets see if we can make sense of these B1 and B2 values in our drive. Lets start by adjusting B1 and see how it varies our squeezing:

```
[ ]: psi0 = fock(n, 0)
t_vals = np.linspace(0, 100, 1000)
B1 = 0
B2 = 1

QH0_H = create(n) * destroy(n) + 0.5 * identity(n)
H_B1 = position(n)
H_B2 = position(n)**2

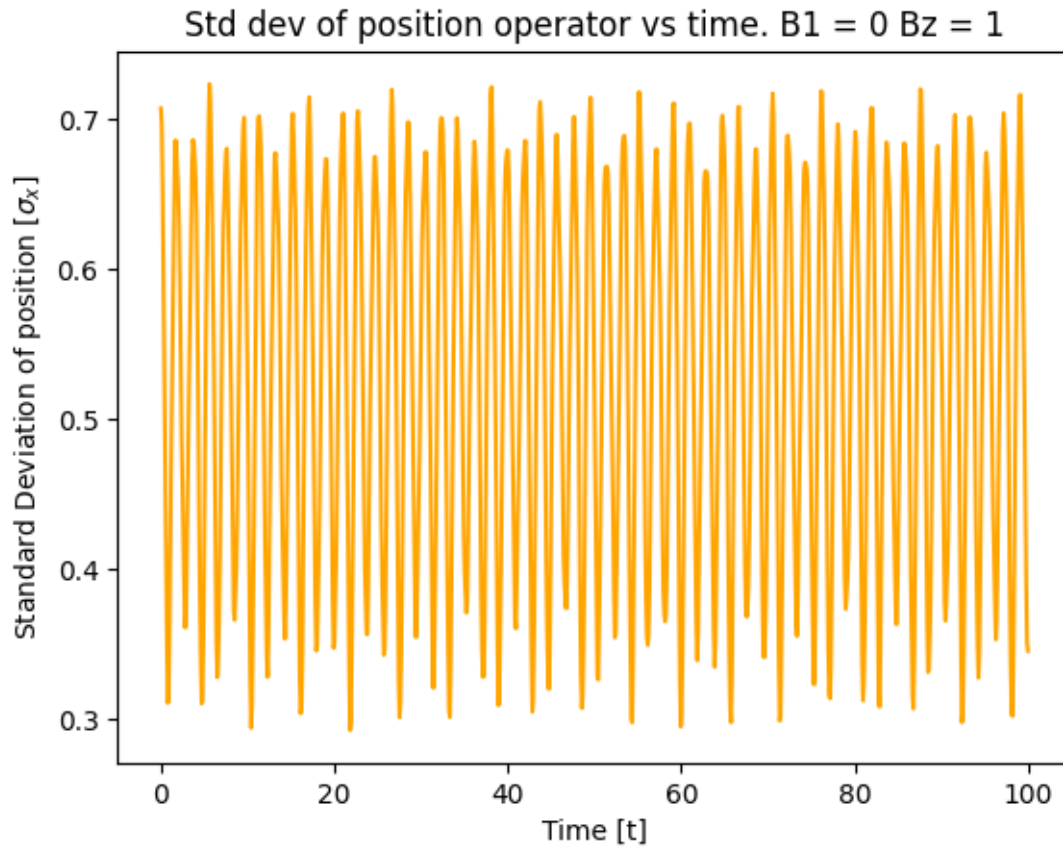
H = [QH0_H, [H_B1, H_B1_drive_coeff], [H_B2, H_B2_drive_coeff]]

res_B_vals = sesolve(H, psi0, t_vals, args={'n': n, 'omega': omega, 'B1': B1,
↪ 'B2': B2})
```

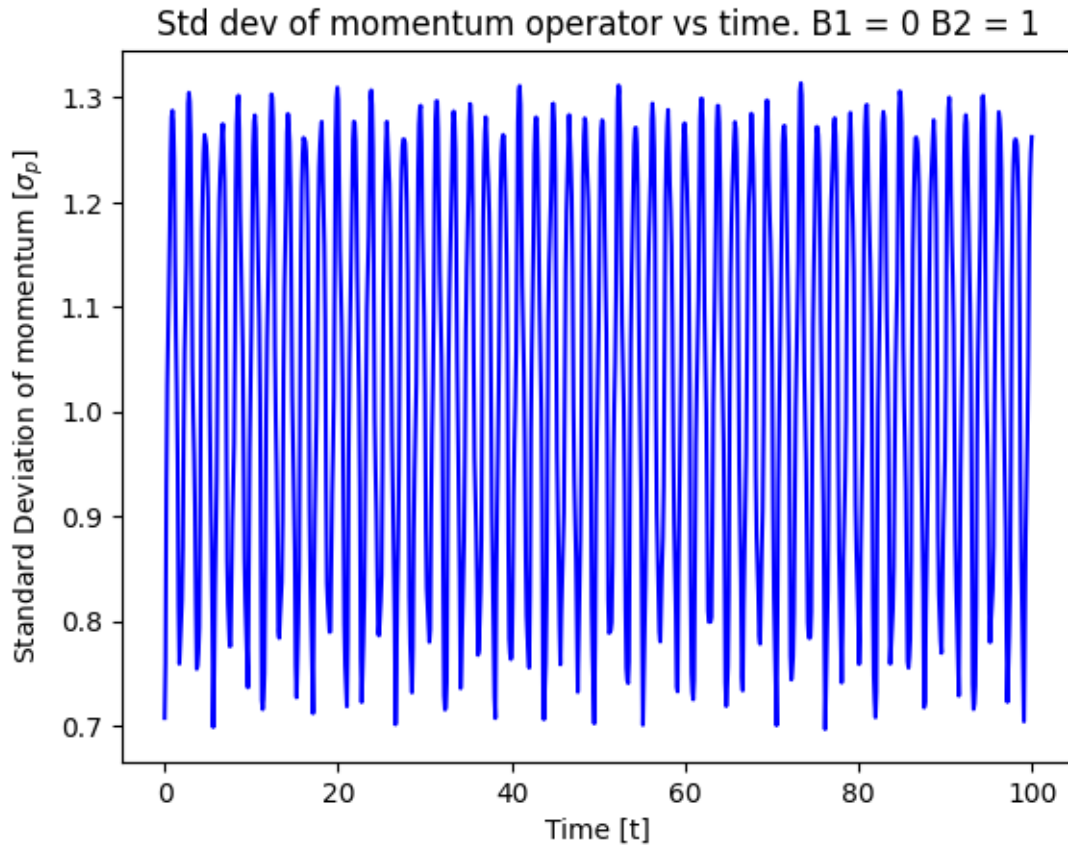


```
res_states_B_vals = res_B_vals.states
```

```
[ ]: driven_x_std_dev_B_vals = get_driven_QHO_std_dev(position(n), res_states_B_vals)
plt.plot(t_vals, driven_x_std_dev_B_vals, label='data', color='orange')
plt.title(f"Std dev of position operator vs time. B1 = {B1} Bz = {B2}")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of position [ $\sigma_x$ ]")
plt.show()
```



```
[ ]: driven_p_std_dev_B_vals = get_driven_QHO_std_dev(momentum(n), res_states_B_vals)
plt.plot(t_vals, driven_p_std_dev_B_vals, label='data', color='blue')
plt.title(f"Std dev of momentum operator vs time. B1 = {B1} B2 = {B2}")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of momentum [ $\sigma_p$ ]")
plt.show()
```

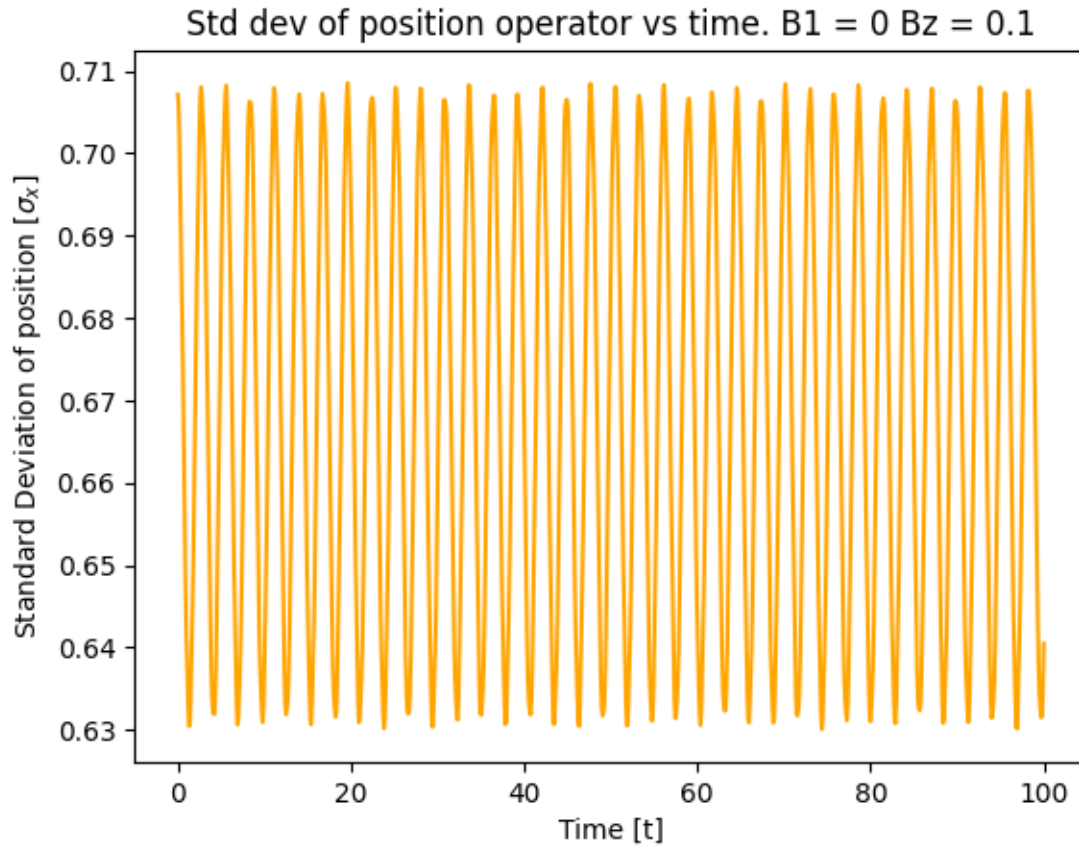


When we set  $B_1$  to 0, it appears that our position stay relatively constant with each other. This implies that our  $B_1$  strength must be linked to the squeezing of our system.

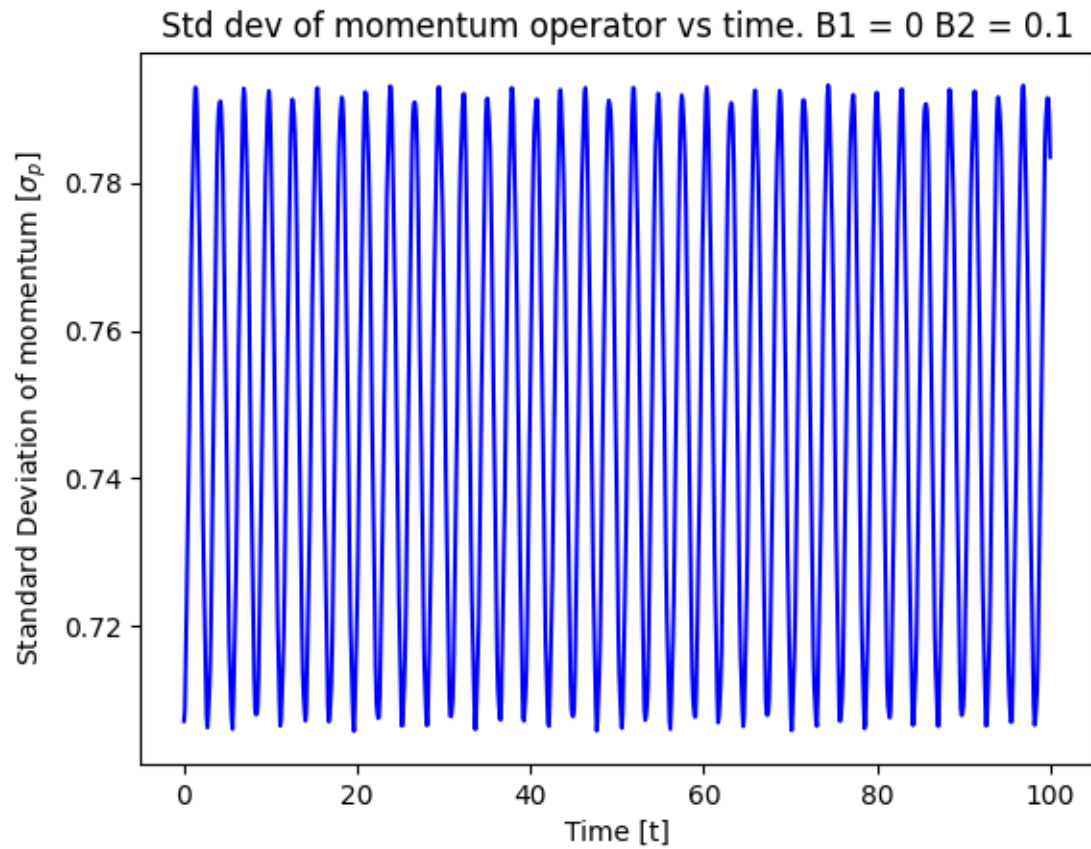
```
[ ]: B1 = 0
      B2 = 0.1

      res_B_vals = sesolve(H, psi0, t_vals, args={'n': n, 'omega': omega, 'B1': B1,
      ↪ 'B2': B2})
      res_states_B_vals = res_B_vals.states

[ ]: driven_x_std_dev_B_vals = get_driven_QHO_std_dev(position(n), res_states_B_vals)
      plt.plot(t_vals, driven_x_std_dev_B_vals, label='data', color='orange')
      plt.title(f"Std dev of position operator vs time. B1 = {B1} Bz = {B2}")
      plt.xlabel("Time [t]")
      plt.ylabel("Standard Deviation of position [ $\sigma_x$ ]")
      plt.show()
```



```
[ ]: driven_p_std_dev_B_vals = get_driven_QHO_std_dev(momentum(n), res_states_B_vals)
plt.plot(t_vals, driven_p_std_dev_B_vals, label='data', color='blue')
plt.title(f"Std dev of momentum operator vs time. B1 = {B1} B2 = {B2}")
plt.xlabel("Time [t]")
plt.ylabel("Standard Deviation of momentum [$\sigma_p$]")
plt.show()
```



When we make B2 nearly zero, we see that our oscillations are much smaller. This implies that B2 is linked to the oscillations of our squeezing.