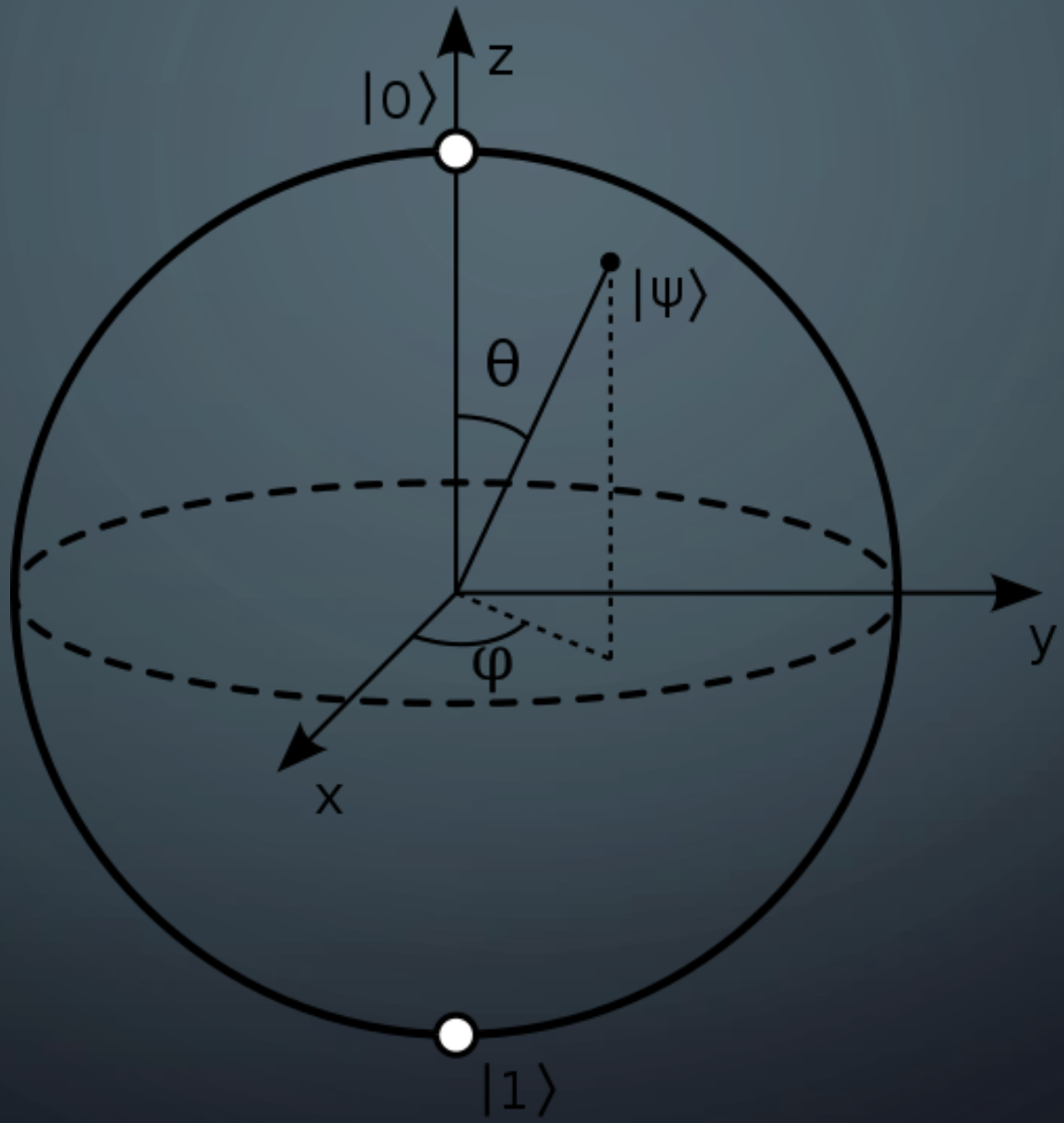
TEMATYKA PROJEKTU:
WSTĘP DO OBLICZEŃ KWANTOWYCH

PRZEMYSŁAW RYŚ

PODSTAWOWE POJĘCIA

- Kubit jest to bit kwantowy, który nie przybiera wartości zero-jedynkowych jak jest to w przypadku klasycznych komputerów, a jest superpozycją dwóch stanów ket 0 (|0>) i ket 1 (|1>). W myśl tej definicji kubitem może być dowolny obiekt, który możemy przedstawić w superpozycji stanów (spin elektronu, polaryzacja fotonu etc.)
- Sfera Blocha jest graficzną próbą przedstawienia stanu pojedynczego kubitu. W dużej mierze ułatwia ona wizualizację, a tym lepsze zrozumienie tego co liczymy.


z
|0⟩
|ψ⟩
θ
y
φ
x
|1⟩

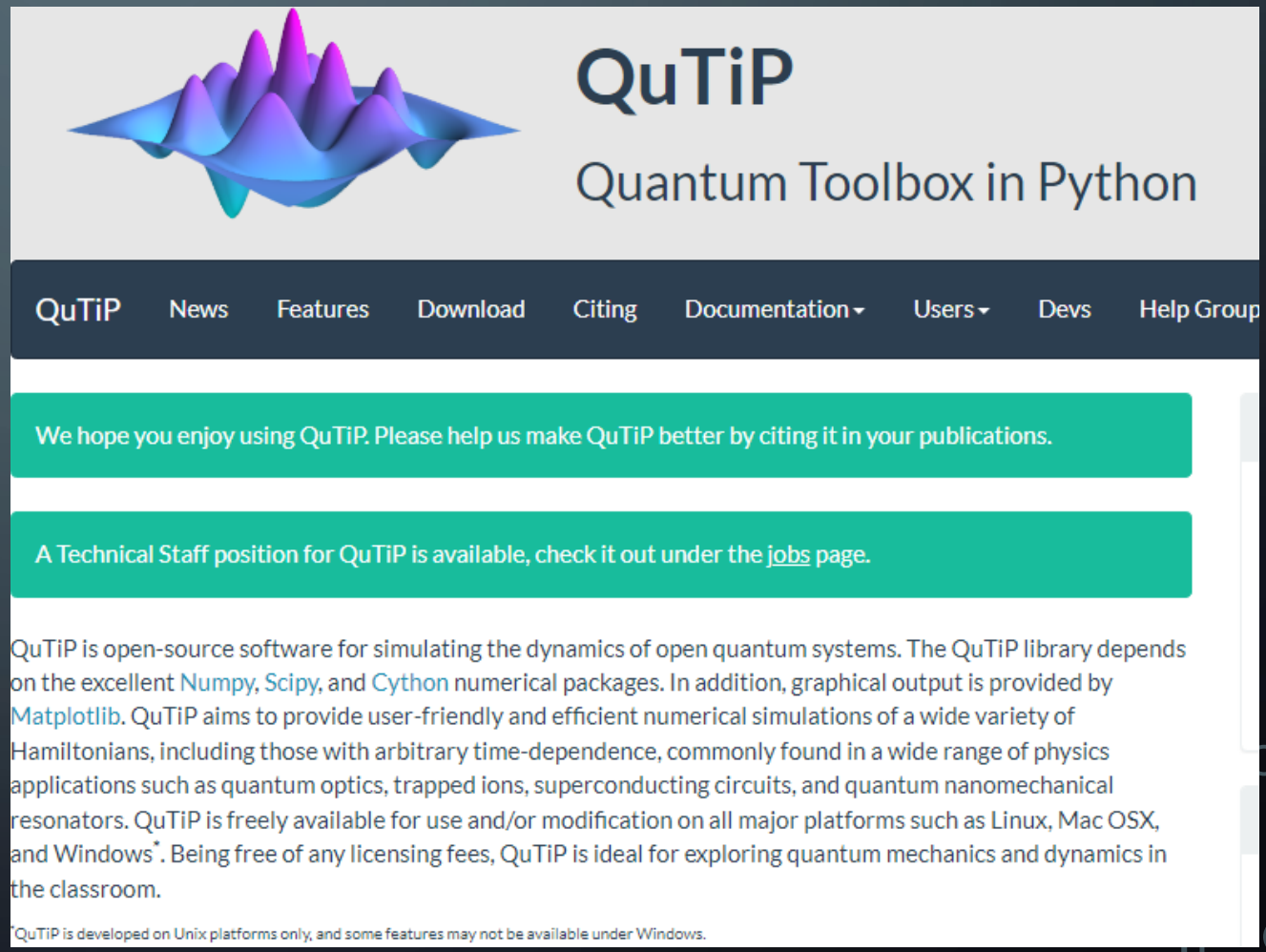
TEMATYKA PROJEKTU

- Przestawienie wektorów stanu na sferze Blocha.
- Przeprowadzenie obliczeń dla przypadku oscylacji Rabiego.
- Przeprowadzenie obliczeń dla przypadku impulsu gaussowskiego.

ŚRODOWISKO PROGRAMISTYCZNE

Zastosowanym do realizacji projektu środowiskiem było rozszerzenie do języka Python, służące specjalnie do obliczeń kwantowych.

https://qutip.org



Na początek wraz z importem stosownych bibliotek, definiujemy początkowe stany naszych kubitów.

In [1]:
```
from qutip import *
import matplotlib.pyplot as plt
import numpy as np
import math
from pylab import *
import matplotlib.animation as animation
from mpl_toolkits.mplot3d import Axes3D
```

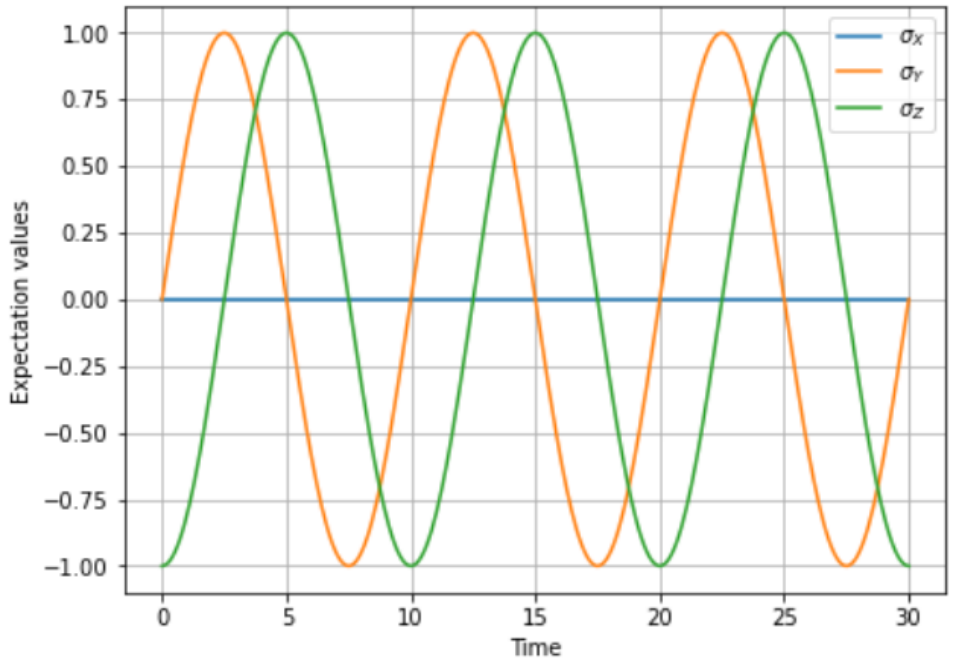
In [2]:
```
#wektory stanu na sferze Blocha w kierunku x, y, z, -x, -y, -z
psi_z = basis(2, 0)
psi_mz = basis(2, 1)
psi_y = (basis(2, 0)+1j*basis(2, 1)).unit()
psi_my = (basis(2, 0)-1j*basis(2, 1)).unit()
psi_x = (basis(2, 0)+basis(2, 1)).unit()
psi_mx = (basis(2, 0)-basis(2, 1)).unit()
```

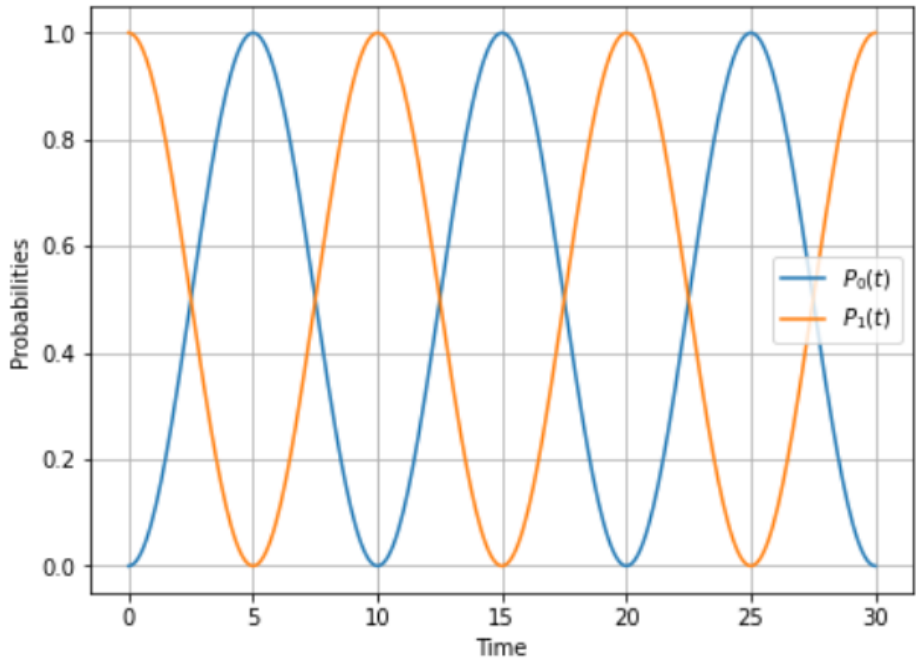
W kolejnym etapie definiujemy hamiltonian w zależności od zadanego parametru. Za stan w chwili t=0 przyjmujemy ket |1>. W zadanej liście e_ops znajdują się operatory, których będziemy liczyć wartości oczekiwane (Pierwsze trzy są to macierze Pauliego $\sigma_x, \sigma_y, \sigma_z$, a kolejne dwa są to operatory projekcji na odpowiednie stany).

In [13]:
```
Omega_x = 2*np.pi*0.05
H =  Omega_x * sigmax()
psi0 = psi_mz
e_ops = [sigmax(), sigmay(), sigmaz(), psi_z.proj(), psi_mz.proj()]
times = np.linspace(0.0, 30.0, 300)
result = sesolve(H, psi0, times, e_ops)
sx=result.expect[0]
sy=result.expect[1]
sz=result.expect[2]
P_0=result.expect[3]
P_1=result.expect[4]
```

In [4]:

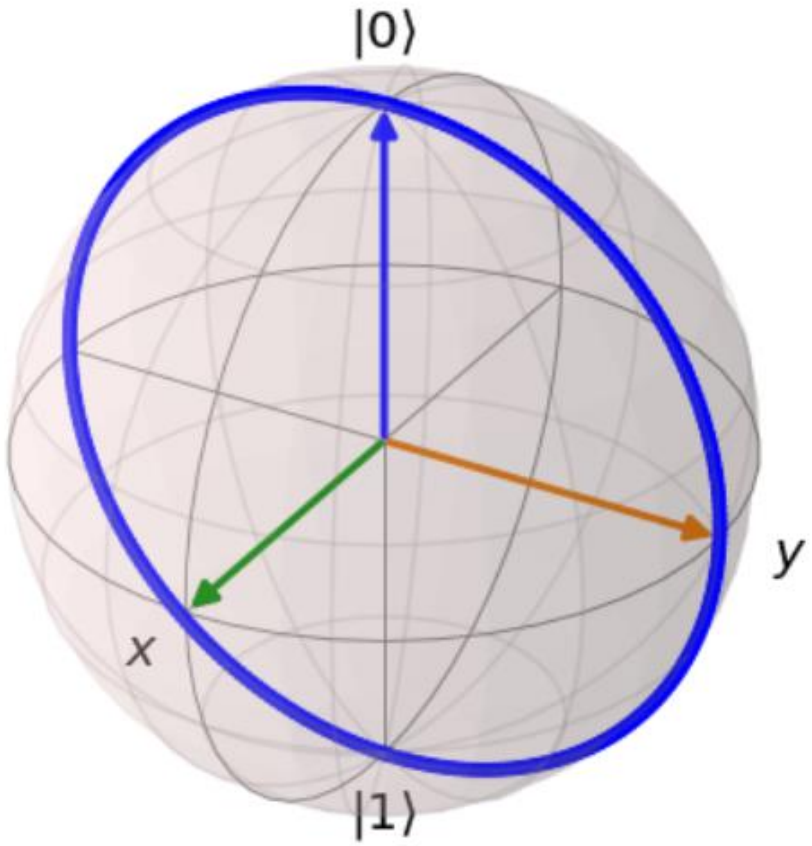
```
fig, axs = plt.subplots(1, 2, figsize=(15, 5))
axs[0].plot(result.times, sx)
axs[0].plot(result.times, sy)
axs[0].plot(result.times, sz)
axs[0].set_xlabel('Time')
axs[0].set_ylabel('Expectation values')
axs[0].legend(("$\sigma_X$", "$\sigma_Y$", "$\sigma_Z$"))
axs[0].grid()

axs[1].plot(result.times, P_0)
axs[1].plot(result.times, P_1)
axs[1].set_xlabel('Time')
axs[1].set_ylabel('Probabilities')
axs[1].legend(("$P_{0}(t)$", "$P_{1}(t)$"))
axs[1].grid()
plt.show()
```





In [15]:

```
b = Bloch()
vec = [[1,0,0],[0,1,0],[0,0,1]]
pnts = [sx, sy, sz]
b.add_points(pnts)
b.add_vectors(vec)
b.show()
```



OSCYLACJE RABIEGO

In [1]:
```
from qutip import *
import matplotlib.pyplot as plt
import numpy as np
import math
```

In [2]:
```
psi_z = basis(2, 0)
psi_mz = basis(2, 1)
psi_y = (basis(2, 0)+1j*basis(2, 1)).unit()
psi_my = (basis(2, 0)-1j*basis(2, 1)).unit()
psi_x = (basis(2, 0)+basis(2, 1)).unit()
psi_mx = (basis(2, 0)-basis(2, 1)).unit()
```

In [3]:
```
def hamiltonian(t, args):
    omega, omega0, omega1 = args[0], args[1], args[2]
    H = omega0*sigmaz() + omega1*(np.cos(omega*t)*sigmax()-np.sin(omega*t)*sigmay())
    H = -0.5*H
    return H
```

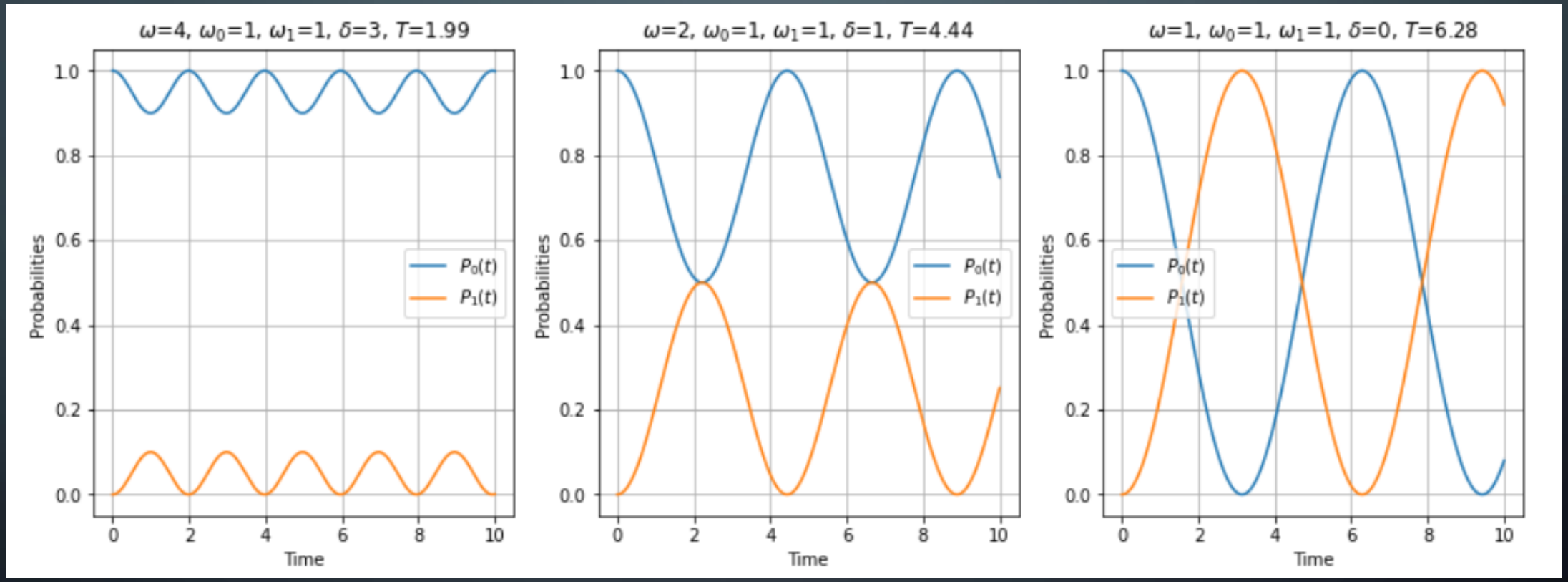
In [4]:
```
def dynamics(tList, psi_t0, omega, omega0, omega1):
    args    = (omega,omega0,omega1)
    delta   = omega-omega0
    T       = 2*math.pi/(math.sqrt((delta)**2 + omega1**2))
    c_ops = []
    e_ops = [sigmax(), sigmay(), sigmaz(), psi_z.proj(), psi_mz.proj()]
    output = mesolve(hamiltonian, psi_t0, tList, c_ops, e_ops, args=args, progress_bar=True)
    return {'omega': omega, 'omega0': omega0, 'omega1': omega1, \
            'sx': output.expect[0], 'sy': output.expect[1], 'sz': output.expect[2], \
            'probOfUpState': output.expect[3], 'probOfDownState': output.expect[4], 'delta': delta, 'T': T}
```

In [5]:
```
data = []
data1 = []
data2 = []
data3 = []
```

In [6]:
```
tList = np.linspace(0,10,300)
psi0=psi_z
data1=dynamics(tList, psi0, 4, 1, 1)
data2=dynamics(tList, psi0, 2, 1, 1)
data3=dynamics(tList, psi0, 1, 1, 1)
data.append(data1)
data.append(data2)
data.append(data3)
```

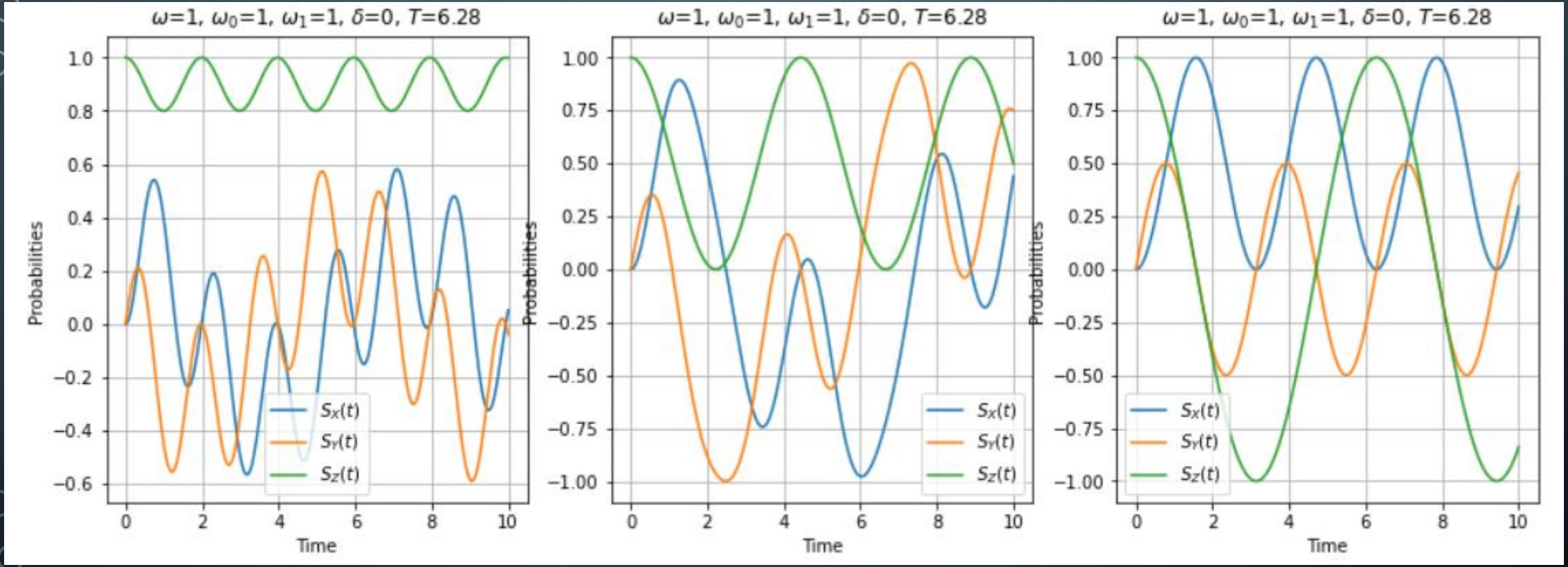
In [9]:

```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].plot(tList, data1['probOfUpState'], label='P_0(t)     Parameters:'+f'{label_1_Part1}.{label_1_Part2}')
axs[0].plot(tList, data1['probOfDownState'], label='P_1(t)     Parameters:'+f'{label_1_Part1}.{label_1_Part2}')
axs[0].set_xlabel('Time')
axs[0].set_ylabel('Probabilities')
axs[0].set_title(f'{label_1_Part1}, {label_1_Part2}')
axs[0].legend(("$P_{0}(t)$", "$P_{1}(t)$"))
axs[0].grid()

axs[1].plot(tList, data2['probOfUpState'], label='P_0(t)     Parameters:'+f'{label_2_Part1}, {label_2_Part2}')
axs[1].plot(tList, data2['probOfDownState'], label='P_1(t)     Parameters:'+f'{label_2_Part1}, {label_2_Part2}')
axs[1].set_xlabel('Time')
axs[1].set_ylabel('Probabilities')
axs[1].set_title(f'{label_2_Part1}, {label_2_Part2}')
axs[1].legend(("$P_{0}(t)$", "$P_{1}(t)$"))
axs[1].grid()

axs[2].plot(tList, data3['probOfUpState'], label='P_0(t)     Parameters:'+f'{label_3_Part1}, {label_3_Part2}')
axs[2].plot(tList, data3['probOfDownState'], label='P_1(t)     Parameters:'+f'{label_3_Part1}, {label_3_Part2}')
axs[2].set_xlabel('Time')
axs[2].set_ylabel('Probabilities')
axs[2].set_title(f'{label_3_Part1}, {label_3_Part2}')
axs[2].legend(("$P_{0}(t)$", "$P_{1}(t)$"))
axs[2].grid()

plt.show()
```


ω=4, ω_0=1, ω_1=1, δ=3, T=1.99
ω=2, ω_0=1, ω_1=1, δ=1, T=4.44
ω=1, ω_0=1, ω_1=1, δ=0, T=6.28
Probabilities
Time
$P_0(t)$
$P_1(t)$
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10

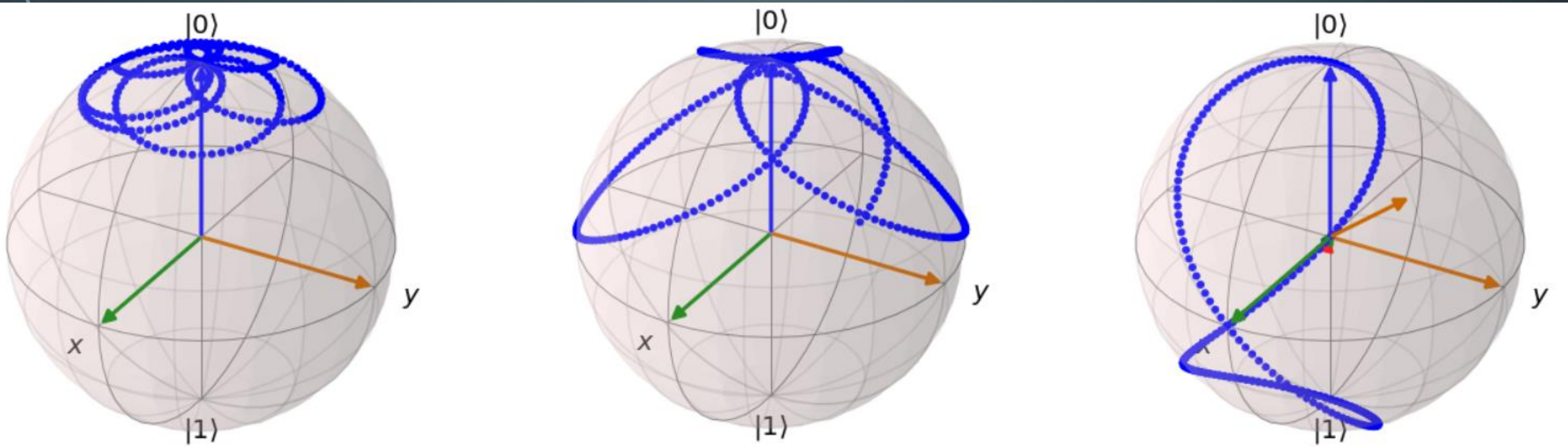
In [10]:

```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].plot(tList, data1['sx'])
axs[0].plot(tList, data1['sy'])
axs[0].plot(tList, data1['sz'])
axs[0].set_xlabel('Time')
axs[0].set_ylabel('Probabilities')
axs[0].set_title(f'{label_3_Part1}, {label_3_Part2}')
axs[0].legend(("$S_{X}(t)$", "$S_{Y}(t)$", "$S_{Z}(t)$"))
axs[0].grid()

axs[1].plot(tList, data2['sx'])
axs[1].plot(tList, data2['sy'])
axs[1].plot(tList, data2['sz'])
axs[1].set_xlabel('Time')
axs[1].set_ylabel('Probabilities')
axs[1].set_title(f'{label_3_Part1}, {label_3_Part2}')
axs[1].legend(("$S_{X}(t)$", "$S_{Y}(t)$", "$S_{Z}(t)$"))
axs[1].grid()

axs[2].plot(tList, data3['sx'])
axs[2].plot(tList, data3['sy'])
axs[2].plot(tList, data3['sz'])
axs[2].set_xlabel('Time')
axs[2].set_ylabel('Probabilities')
axs[2].set_title(f'{label_3_Part1}, {label_3_Part2}')
axs[2].legend(("$S_{X}(t)$", "$S_{Y}(t)$", "$S_{Z}(t)$"))
axs[2].grid()

plt.show()
```

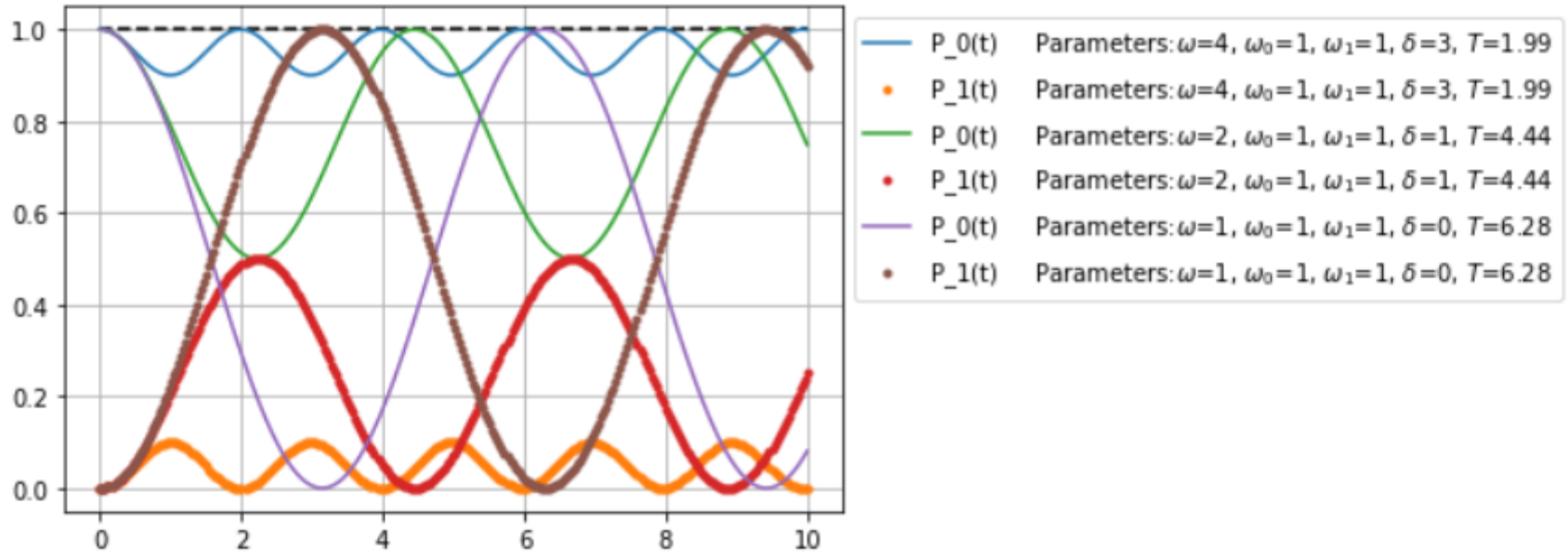

ω=1, ω_0=1, ω_1=1, δ=0, T=6.28
Probabilities
Time
$S_X(t)$
$S_Y(t)$
$S_Z(t)$
ω=1, ω_0=1, ω_1=1, δ=0, T=6.28
Probabilities
Time
$S_X(t)$
$S_Y(t)$
$S_Z(t)$
ω=1, ω_0=1, ω_1=1, δ=0, T=6.28
Probabilities
Time
$S_X(t)$
$S_Y(t)$
$S_Z(t)$

In [11]:

```
b1 = Bloch()
vec = [[1,0,0],[0,1,0],[0,0,1]]
pnts = [data1['sx'], data1['sy'], data1['sz']]
b1.add_points(pnts)
b1.add_vectors(vec)
b1.show()

b2 = Bloch()
vec = [[1,0,0],[0,1,0],[0,0,1]]
pnts = [data2['sx'], data2['sy'], data2['sz']]
b2.add_points(pnts)
b2.add_vectors(vec)
b2.show()

b3 = Bloch()
vec = [[1,0,0],[0,1,0],[0,0,1]]
pnts = [data3['sx'], data3['sy'], data3['sz']]
b3.add_points(pnts)
b3.add_vectors(vec)
b3.add_vectors(pnts)
b3.show()
```


|0⟩
y
x
|1⟩
|0⟩
y
x
|1⟩
|0⟩
y
|1⟩

In [12]:
```
plt.plot(tList, [1]*300, 'k--')
for d in data:
    labelPart1 = f"$\omega$={d['omega']}, $\omega_0$={d['omega0']}, $\omega_1$={d['omega1']}"
    labelPart2 = f"$\delta$={d['delta']}, $T$={d['T']:.2f}"
    plt.plot(tList, d['probOfUpState'], label='P_0(t)     Parameters:'+f'{labelPart1}, {labelPart2}')
    plt.plot(tList, d['probOfDownState'], '.', label='P_1(t)     Parameters:'+f'{labelPart1}, {labelPart2}')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
plt.show()
```



IMPULS GAUSSOWSKI

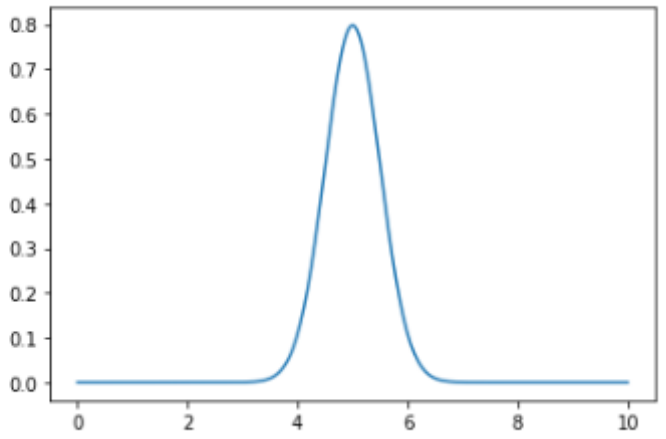
In [1]:
```
from qutip import *
import matplotlib.pyplot as plt
import numpy as np
import math
```

In [2]:
```
psi_z = basis(2, 0)
psi_mz = basis(2, 1)
psi_y = (basis(2, 0)+1j*basis(2, 1)).unit()
psi_my = (basis(2, 0)-1j*basis(2, 1)).unit()
psi_x = (basis(2, 0)+basis(2, 1)).unit()
psi_mx = (basis(2, 0)-basis(2, 1)).unit()
```

In [3]:
```
def gaussian(t, t_M, sigma):
    return 1./(sigma*math.sqrt(2*np.pi))*math.exp(-(((t-t_M)**2)/(2*sigma**2)))
```

In [4]:
```
tList = np.linspace(0,10,300)
F=[]
for t in tList:
    F.append(gaussian(t, 5, 0.5))
plt.plot(tList, F)
```

Out[4]: [<matplotlib.lines.Line2D at 0x7fa0d88971d0>]



In [5]:
```
def hamiltonian_x(t, args):
    theta, t_M, sigma = args[0], args[1], args[2]
    H_x = -0.5*theta*gaussian(t, t_M, sigma)*sigmax()
    return H_x
```

In [6]:
```
def hamiltonian_y(t, args):
    theta, t_M, sigma = args[0], args[1], args[2]
    H_y = -0.5*theta*gaussian(t, t_M, sigma)*sigmay()
    return H_y
```

In [7]:
```
def hamiltonian_z(t, args):
    theta, t_M, sigma = args[0], args[1], args[2]
    H_z = -0.5*theta*gaussian(t, t_M, sigma)*sigmaz()
    return H_z
```

In [8]:
```
def dynamics_x(tList, psi_t0, theta, t_M, sigma):
    args    = (theta,t_M,sigma)
    c_ops = []
    e_ops = [sigmax(), sigmay(), sigmaz(), psi_z.proj(), psi_mz.proj()]
    output = mesolve(hamiltonian_x, psi_t0, tList, c_ops, e_ops, args=args, progress_bar=True)
    return {'theta': theta, \
            'sx': output.expect[0], 'sy': output.expect[1], 'sz': output.expect[2],\
            'probOfUpState': output.expect[3], 'probOfDownState': output.expect[4],\
            't_M': t_M, 'sigma': sigma}
```

In [9]:
```
def dynamics_y(tList, psi_t0, theta, t_M, sigma):
    args    = (theta,t_M,sigma)
    c_ops = []
    e_ops = [sigmax(), sigmay(), sigmaz(), psi_z.proj(), psi_mz.proj()]
    output = mesolve(hamiltonian_y, psi_t0, tList, c_ops, e_ops, args=args, progress_bar=True)
    return {'theta': theta, \
            'sx': output.expect[0], 'sy': output.expect[1], 'sz': output.expect[2],\
            'probOfUpState': output.expect[3], 'probOfDownState': output.expect[4],\
            't_M': t_M, 'sigma': sigma}
```

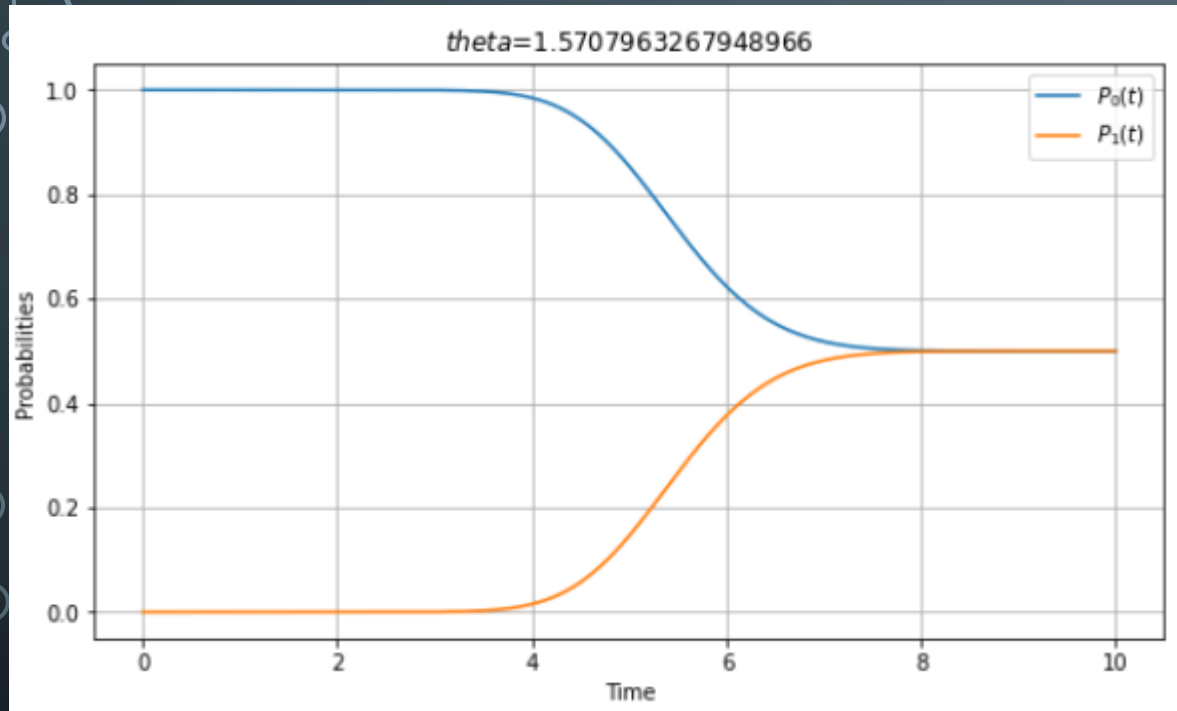
In [10]:
```
def dynamics_z(tList, psi_t0, theta, t_M, sigma):
    args    = (theta,t_M,sigma)
    c_ops = []
    e_ops = [sigmax(), sigmay(), sigmaz(), psi_z.proj(), psi_mz.proj()]
    output = mesolve(hamiltonian_z, psi_t0, tList, c_ops, e_ops, args=args, progress_bar=True)
    return {'theta': theta, \
            'sx': output.expect[0], 'sy': output.expect[1], 'sz': output.expect[2],\
            'probOfUpState': output.expect[3], 'probOfDownState': output.expect[4],\
            't_M': t_M, 'sigma': sigma}
```

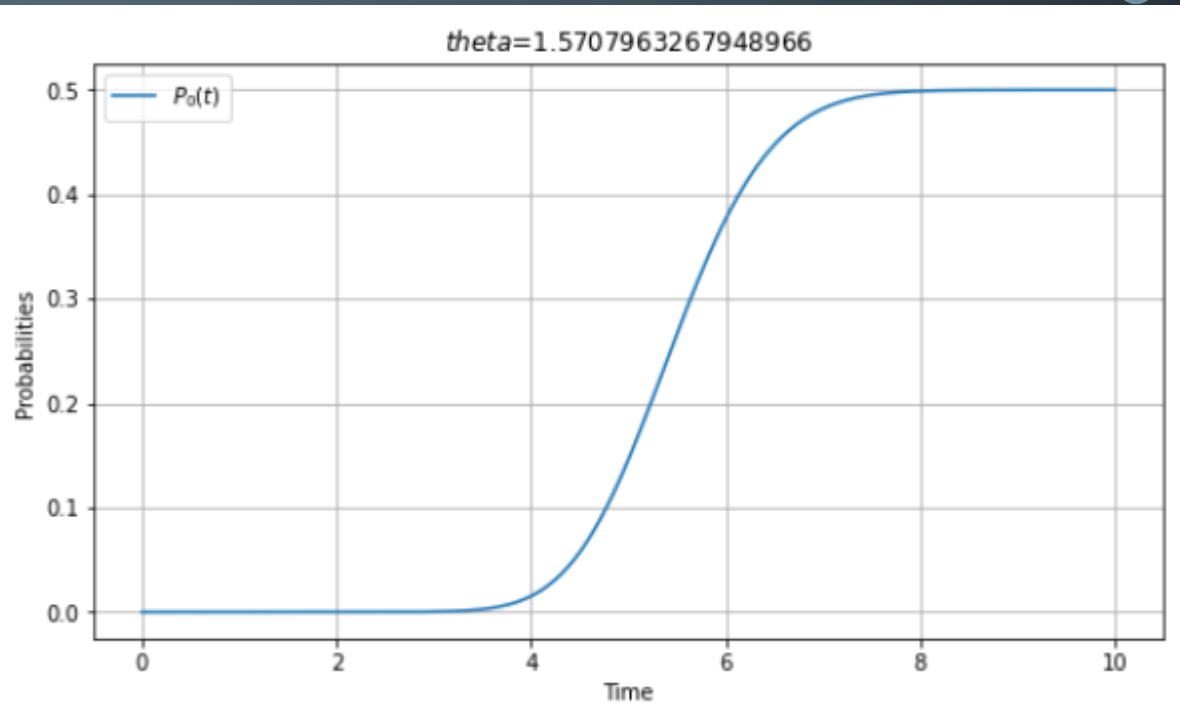
In [11]:
```
data = []
data1 = []
data2 = []
data3 = []
```

In [12]:
```
psi0=psi_z
data1=dynamics_x(tList, psi0, 0.5*np.pi, 5, 1)
data.append(data1)
data2=dynamics_y(tList, psi0, 2*np.pi, 5, 1)
data.append(data2)
data3=dynamics_z(tList, psi0, np.pi, 5, 1)
data.append(data3)
```

In [13]:
```
label_1 = f"$\ttheta$={data1['theta']}"
label_2 = f"$\ttheta$={data2['theta']}"
label_3 = f"$\ttheta$={data3['theta']}"
```

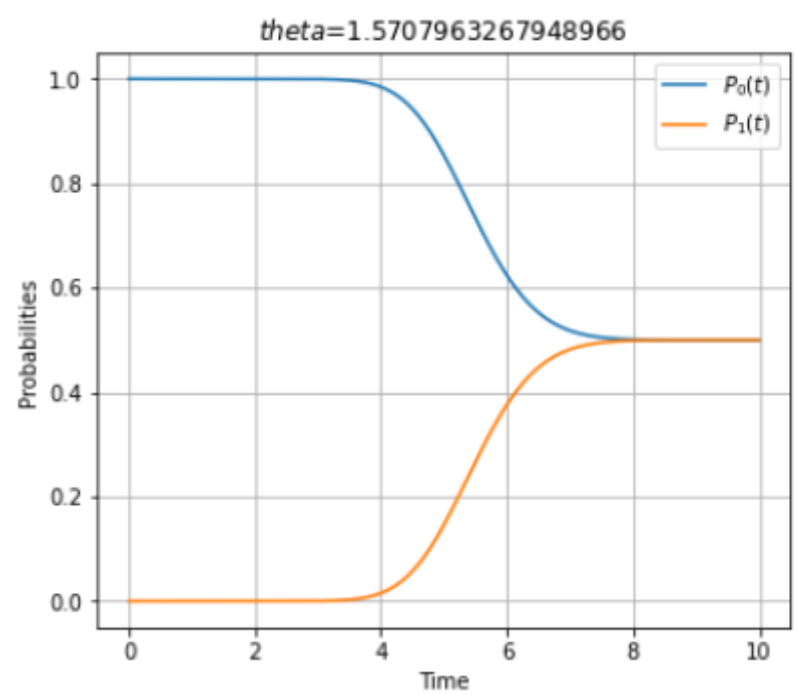
In [14]:
```
fig, axs = plt.subplots(1, 2, figsize=(20, 5))
axs[0].plot(tList, data1['probOfUpState'], label='P_0(t)     Parameters:'+f'{label_1}')
axs[0].plot(tList, data1['probOfDownState'], label='P_1(t)     Parameters:'+f'{label_1}')
axs[0].set_xlabel('Time')
axs[0].set_ylabel('Probabilities')
axs[0].set_title(f'{label_1}')
axs[0].legend(("$P_{0}(t)$", "$P_{1}(t)$"))
axs[0].grid()

axs[1].plot(tList, data1['probOfDownState'], label='P_1(t)     Parameters:'+f'{label_1}')
axs[1].set_xlabel('Time')
axs[1].set_ylabel('Probabilities')
axs[1].set_title(f'{label_1}')
axs[1].legend(("$P_{0}(t)$", "$P_{1}(t)$"))
axs[1].grid()

plt.show()
```

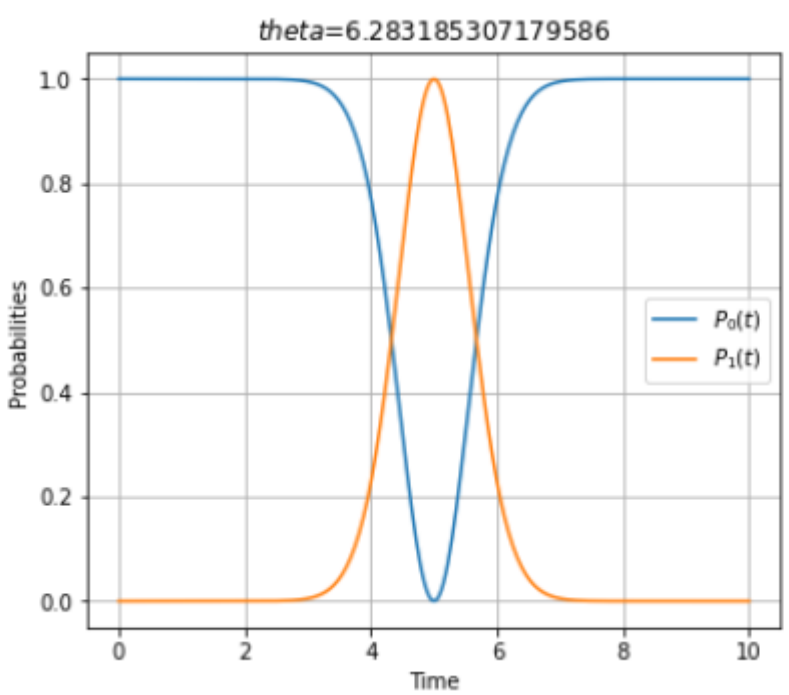

theta=1.5707963267948966
$P_0(t)$
$P_1(t)$
Probabilities
Time
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10

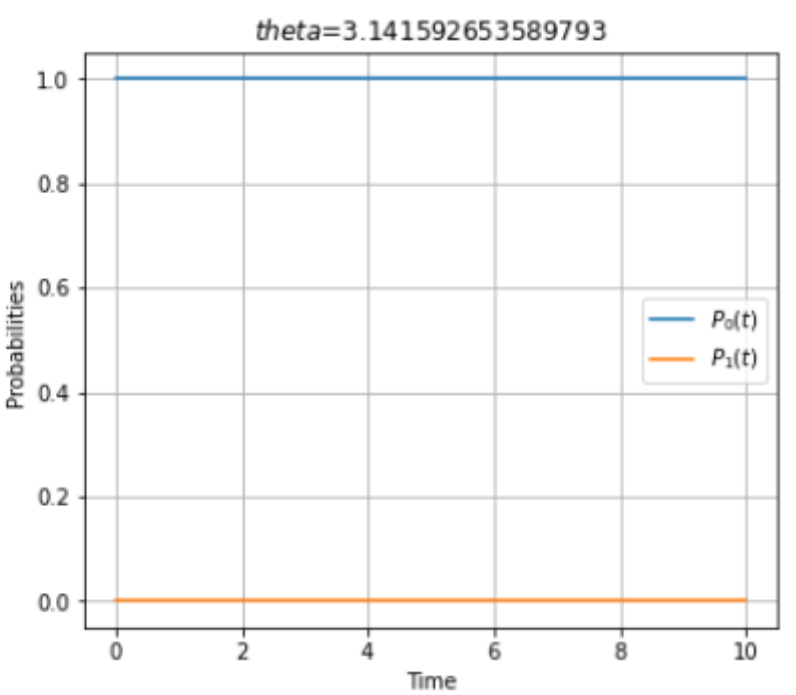

theta=1.5707963267948966
$P_0(t)$
Probabilities
Time
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
10

In [15]:

```
fig, axs = plt.subplots(1, 3, figsize=(20, 5))
axs[0].plot(tList, data1['probOfUpState'], label='P_0(t)     Parameters:'+f'{label_1}')
axs[0].plot(tList, data1['probOfDownState'], label='P_1(t)     Parameters:'+f'{label_1}')
axs[0].set_xlabel('Time')
axs[0].set_ylabel('Probabilities')
axs[0].set_title(f'{label_1}')
axs[0].legend(("$P_{0}(t)$", "$P_{1}(t)$"))
axs[0].grid()

axs[1].plot(tList, data2['probOfUpState'], label='P_0(t)     Parameters:'+f'{label_2}')
axs[1].plot(tList, data2['probOfDownState'], label='P_1(t)     Parameters:'+f'{label_2}')
axs[1].set_xlabel('Time')
axs[1].set_ylabel('Probabilities')
axs[1].set_title(f'{label_2}')
axs[1].legend(("$P_{0}(t)$", "$P_{1}(t)$"))
axs[1].grid()

axs[2].plot(tList, data3['probOfUpState'], label='P_0(t)     Parameters:'+f'{label_3}')
axs[2].plot(tList, data3['probOfDownState'], label='P_1(t)     Parameters:'+f'{label_3}')
axs[2].set_xlabel('Time')
axs[2].set_ylabel('Probabilities')
axs[2].set_title(f'{label_3}')
axs[2].legend(("$P_{0}(t)$", "$P_{1}(t)$"))
axs[2].grid()

plt.show()
```

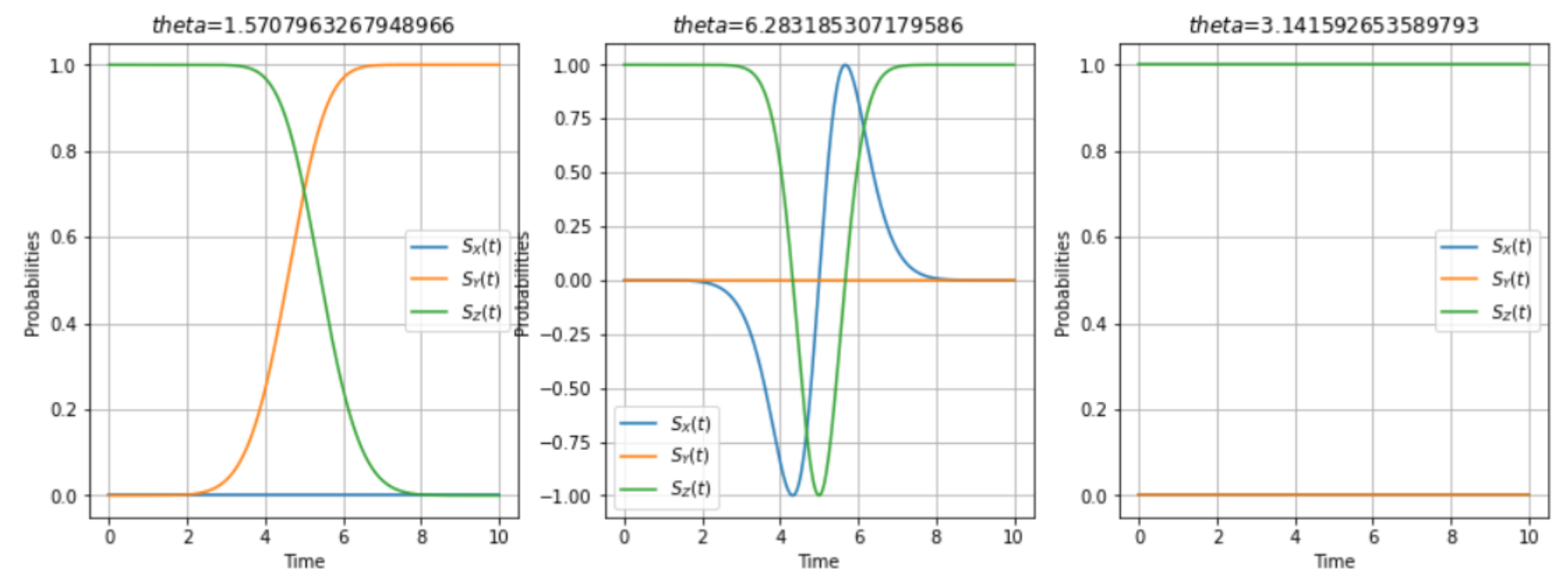

theta=1.5707963267948966
Probabilities
Time
$P_0(t)$
$P_1(t)$
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10


theta=6.283185307179586
Probabilities
Time
$P_0(t)$
$P_1(t)$
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10


theta=3.141592653589793
Probabilities
Time
$P_0(t)$
$P_1(t)$
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10

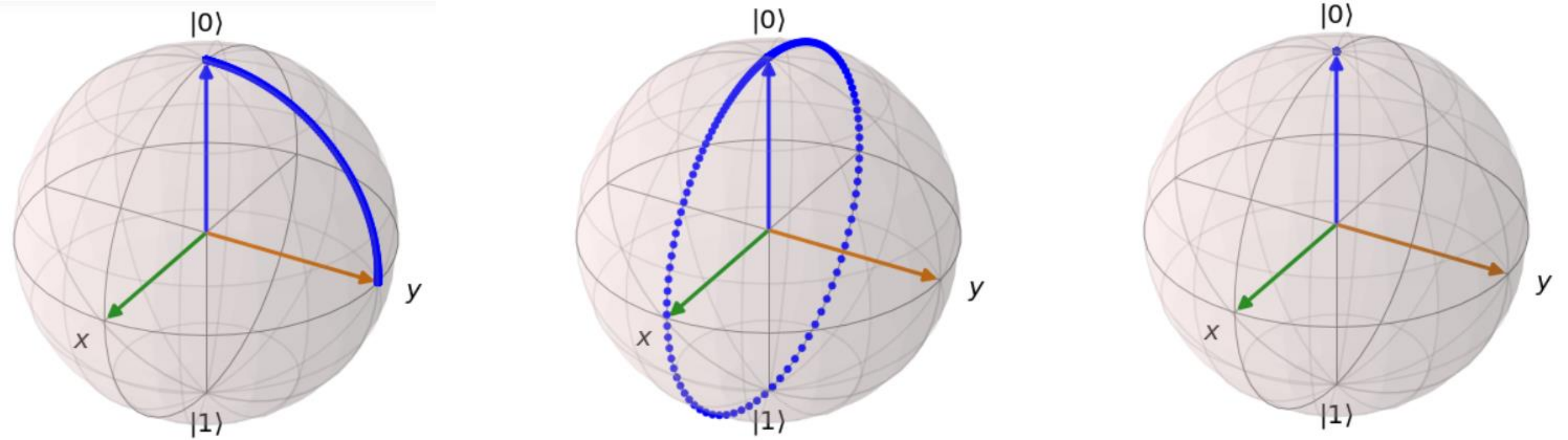
In [16]:

```
fig, axs = plt.subplots(3, 1, figsize=(5, 15))
axs[0].plot(tList, data1['sx'])
axs[0].plot(tList, data1['sy'])
axs[0].plot(tList, data1['sz'])
axs[0].set_xlabel('Time')
axs[0].set_ylabel('Probabilities')
axs[0].set_title(f'{label_1}')
axs[0].legend(("$S_{X}(t)$", "$S_{Y}(t)$", "$S_{Z}(t)$"))
axs[0].grid()

axs[1].plot(tList, data2['sx'])
axs[1].plot(tList, data2['sy'])
axs[1].plot(tList, data2['sz'])
axs[1].set_xlabel('Time')
axs[1].set_ylabel('Probabilities')
axs[1].set_title(f'{label_2}')
axs[1].legend(("$S_{X}(t)$", "$S_{Y}(t)$", "$S_{Z}(t)$"))
axs[1].grid()

axs[2].plot(tList, data3['sx'])
axs[2].plot(tList, data3['sy'])
axs[2].plot(tList, data3['sz'])
axs[2].set_xlabel('Time')
axs[2].set_ylabel('Probabilities')
axs[2].set_title(f'{label_3}')
axs[2].legend(("$S_{X}(t)$", "$S_{Y}(t)$", "$S_{Z}(t)$"))
axs[2].grid()

plt.show()
```


theta=1.5707963267948966
theta=6.283185307179586
theta=3.141592653589793
Probabilities
Time
$S_x(t)$
$S_y(t)$
$S_z(t)$

In [19]:

```
b1 = Bloch()
vec = [[1,0,0],[0,1,0],[0,0,1]]
pnts = [data1['sx'], data1['sy'], data1['sz']]
b1.add_points(pnts)
b1.add_vectors(vec)
b1.show()

b2 = Bloch()
vec = [[1,0,0],[0,1,0],[0,0,1]]
pnts = [data2['sx'], data2['sy'], data2['sz']]
b2.add_points(pnts)
b2.add_vectors(vec)
b2.show()

b3 = Bloch()
vec = [[1,0,0],[0,1,0],[0,0,1]]
pnts = [data3['sx'], data3['sy'], data3['sz']]
b3.add_points(pnts)
b3.add_vectors(vec)
b3.show()
```


|0⟩
y
x
|1⟩
|0⟩
y
x
|1⟩
|0⟩
y
x
|1⟩

KONIEC