PHYS 357 Pset 3. Due 11:59 PM Thursday Sep. 26

1. Starting with the angular momentum operators (Jx,Jy,Jz) you worked out in the last problem set (which should have been

$$J_z = \frac{\hbar}{2}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad J_x = \frac{\hbar}{2}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad J_y = \frac{\hbar}{2}\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{1}$$

show that you get the cannonical commutation relations

$$[J_x, J_y] = i\hbar J_z, \quad [J_y, J_z] = i\hbar J_x, \quad [J_z, J_x] = i\hbar J_y \tag{2}$$

where $[A, B] = AB - BA$ for matrices A and B.

2. Consider two coordinate systems (x, y, z) and (x', y', z) that have the same z-axis, but have their x- and y-axes rotated by some angle γ relative to each other (note the lack of a prime on z). Show that if I rotate a vector in the (x, y, z) coordinate system through some angle θ about the z-axis, I get the same physical vector if I transform the vector to the (x', y', z) coordinate system and carry out the rotation there.

3. If I have a set of arbitrary basis vectors (x', y', z'), where you have the (x, y, z) representation of each basis vector, write down the matrix that converts a vector in the (x, y, z) basis to one in the (x', y', z') basis, and vice-versa. How are these two matrices related? I suggest you write a function

```
genbasis(xyz)
```

that returns *a* valid matrix of basis vectors where xyz is one of the principal axes (a reminder that this isn't unique so you'll have to make a choice. The results of Problem 2 show that this choice won't matter so just do something easy.)

4. We're now ready to work out an rotation matrix in 3D about an arbitrary axis. Start by picking a random rotation axis in the $\hat{n}$, direction. Start with random θ, ϕ and convert that to a vector in (x, y, z) coordinates. Then generate a pair of basis vectors to make an

orthogonal space (hint - if you take the cross product of two vectors, you get something that's perpendicular to both, so e.g. $\hat{z} \times \hat{n}$ will be perpenciular to both $\hat{z}$ and $\hat{n}$). Don't forget that basis vectors must have unit length. Now write out the matrices that convert from (x, y, z) to the $\hat{n}$ basis, rotate by an angle γ in that basis, then convert back to (x, y, z). I suggest you do this on a computer, and write a function

```
genrot(xyz,gamma)
```

that generates the individual matrices, multiplies them together, and returns the final rotation matrix in the (x, y, z) coordinate system. What is the rotation matrix for $\theta = \pi/4$, $\phi = \pi/6$, and gamma=0.01?

5. Pretend, in a shocking turn of events, the Earth's crust gets rotated so that the Royal Greenwich Observatory, England (current latitude 51.476852, longitude=0.000) moves to the north pole. What are Montreal's new latitude and longitude? (current: 45.50884, -73.58781). Verify that the current angle between Montreal and Greenwich corresponds to Montreal's new latitude (as it must, because the distance to the pole depends only on latitude, not longitude, and your rotation matrix had better not change relative angles between vectors). Hint - think what axis we have to be rotating about to move Greenwich to the North pole.

6. Use your the results (and code!) to show numerically that the rotation commutation relations hold in an arbitrary coordinate system. If you haven't done this sort of thing before, you can show that as you make gamma smaller, the largest term in the error shrinks faster than the largest term in the commutator. I found gammas in the range of 0.01-0.001 were sufficient to show this clearly.

Audréanne Bernier (261100643)

Pset 3

1. Starting with the angular momentum operators (Jx,Jy,Jz) you worked out in the last problem set (which should have been

$$J_z = \frac{\hbar}{2}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}, \quad J_x = \frac{\hbar}{2}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, \quad J_y = \frac{\hbar}{2}\begin{bmatrix}0 & -i\\ i & 0\end{bmatrix} \tag{1}$$

show that you get the cannonical commutation relations

$$[J_x, J_y] = i\hbar J_z, \quad [J_y, J_z] = i\hbar J_x, \quad [J_z, J_x] = i\hbar J_y \tag{2}$$

where $[A, B] = AB - BA$ for matrices A and B.

code:

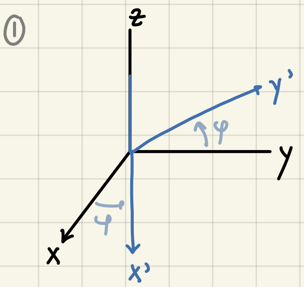
```python
import numpy as np
import scipy as sp

# constants
hbar = sp.constants.hbar

# Functions
def commutation(A,B):
    return A@B - B@A

# Momentum operators in the z basis
Jz = hbar/2 * np.array([[1,0],[0,-1]])
Jx = hbar/2 * np.array([[0,1],[1,0]])
Jy = hbar/2 * np.array([[0,-1j],[1j,0]])

# [Jx,Jy]
Jx_Jy = commutation(Jx,Jy)
print('Jx_Jy:\n', Jx_Jy==1j*hbar*Jz)
# [Jy,Jz]
Jy_Jz = commutation(Jy,Jz)
print('Jy_Jz:\n', Jy_Jz==1j*hbar*Jx)
# [Jz,Jx]
Jz_Jx = commutation(Jz,Jx)
print('Jz_Jx:\n', Jz_Jx==1j*hbar*Jy)
```

output:

```
Jx_Jy:
 [[ True  True]
 [ True  True]]
Jy_Jz:
 [[ True  True]
 [ True  True]]
Jz_Jx:
 [[ True  True]
 [ True  True]]
```

→ checking that $[J_x, J_y] = i\hbar J_z$

2. Consider two coordinate systems (x, y, z) and (x', y', z) that have the same z-axis, but have their x- and y-axes rotated by some angle γ relative to each other (note the lack of a prime on z). Show that if I rotate a vector in the (x, y, z) coordinate system through some angle θ about the z-axis, I get the same physical vector if I transform the vector to the (x', y', z) coordinate system and carry out the rotation there.

① 

$$\hat{x}' = \cos\varphi\,\hat{x} + \sin\varphi\,\hat{y}$$
$$\hat{y}' = -\sin\varphi\,\hat{x} + \cos\varphi\,\hat{y}$$
$$\hat{z}' = \hat{z}$$

write as a matrix: $\underbrace{\begin{pmatrix} x' \\ y' \\ z \end{pmatrix}}_{\text{vector in } x', y', z \text{ basis}} = \underbrace{\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{R_{n\to n'}} \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{\text{vector in } x, y, z \text{ basis}}$

② Rotation matrix around z: $R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$ in x, y, z basis

③ so if we have $v = x\hat{x} + y\hat{y} + z\hat{z}$, rotate by θ around z-axis:

$$v_{rotated} = R_z(\theta)\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ z \end{pmatrix}$$ *in x, y, z basis

Transform to x', y', z basis to compare:

$$R_{n\to n'} \cdot v_{rotated} = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ z \end{pmatrix}$$

$$= \begin{pmatrix} \cos\varphi(x\cos\theta - y\sin\theta) + \sin\varphi(x\sin\theta + y\cos\theta) \\ -\sin\varphi(x\cos\theta - y\sin\theta) + \cos\varphi(x\sin\theta + y\cos\theta) \\ z \end{pmatrix}$$

$$= \begin{pmatrix} x\cos\theta\cos\varphi + x\sin\theta\sin\varphi + y\cos\theta\sin\varphi - y\sin\theta\cos\varphi \\ -x\cos\theta\sin\varphi + x\sin\theta\cos\varphi + y\cos\theta\cos\varphi + y\sin\theta\sin\varphi \\ z \end{pmatrix}$$

④ other way: $v'_{rotated} = R_z(\theta)\, R_{n\to n'}\begin{pmatrix} x \\ y \\ z \end{pmatrix}$

$$= R_z(\theta)\begin{pmatrix} x\cos\varphi + y\sin\varphi \\ -x\sin\varphi + y\cos\varphi \\ z \end{pmatrix}$$

$$= \begin{pmatrix} (x\cos\varphi + y\sin\varphi)\cos\theta + (x\sin\varphi - y\cos\varphi)\sin\theta \\ (x\cos\varphi + y\sin\varphi)\sin\theta - (x\sin\varphi - y\cos\varphi)\cos\theta \\ 1 \end{pmatrix}$$ *in x', y', z basis

$$v'_{rotated} = \begin{pmatrix} x\cos\theta\cos\varphi + y\cos\theta\sin\varphi + x\sin\theta\sin\varphi - y\sin\theta\cos\varphi \\ x\sin\theta\cos\varphi + y\sin\theta\sin\varphi - x\cos\theta\sin\varphi + y\cos\theta\cos\varphi \\ 1 \end{pmatrix} = R_{n\to n'} \cdot v_{rotated} \quad ✓$$

3. If I have a set of arbitrary basis vectors (x', y', z'), where you have the (x, y, z) representation of each basis vector, write down the matrix that converts a vector in the (x, y, z) basis to one in the (x', y', z') basis, and vice-versa. How are these two matrices related? I suggest you write a function

```
genbasis(xyz)
```

that returns *a* valid matrix of basis vectors where xyz is one of the principal axes (a reminder that this isn't unique so you'll have to make a choice. The results of Problem 2 show that this choice won't matter so just do something easy.)

$\hat{x}' = a\hat{x} + b\hat{y} + c\hat{z}$
$\hat{y}' = d \ \cdots \ e \ \cdots \ f \ \cdots$
$\hat{z}' = \cdots$

find $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \underbrace{\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}}_{A} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

$A \Rightarrow$ converts a vector from x, y, z basis to x', y', z' basis

so to get the x', y', z' to x, y, z basis, we have $\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \underbrace{\begin{pmatrix} \quad \end{pmatrix}}_{\text{This is just } A^{-1}} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$

```python
import numpy as np

def genbasis(xyz):
    row_1 = np.array([1,0,0]) if xyz=='x' else (np.array([0,1,0]) if xyz=='y' else np.array([0,0,1]))
    row_2 = np.array([0,1,0]) if xyz=='x' else np.array([1,0,0])
    row_3 = np.cross(row_1, row_2)
    row_3 = row_3/np.linalg.norm(row_3)

    A = np.array([row_1, row_2, row_3])
    return A


# Change of basis matrix
axis = 'y'
A = genbasis(axis)
print(f'Input axis is {axis}:\n{A}')

# Get inverse
A_inv = np.linalg.inv(A)

# Checking
x = np.array([1,0,0])
y = np.array([0,1,0])
z = np.array([0,0,1])

print(f'{A@y}')
print(f'{A_inv@A}')
```

```
Input axis is y:
[[ 0  1  0]
 [ 1  0  0]
 [ 0  0 -1]]
[1 0 0]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

x, y, z to x', y', z'

$\rightarrow$ y in x', y', z'

checking that matrices are inverses

4. We're now ready to work out an rotation matrix in 3D about an arbitrary axis. Start by picking a random rotation axis in the $\hat{n}$, direction. Start with random θ, ϕ and convert that to a vector in (x, y, z) coordinates. Then generate a pair of basis vectors to make an orthogonal space (hint - if you take the cross product of two vectors, you get something that's perpendicular to both, so e.g. $\hat{z} \times \hat{n}$ will be perpenciular to both $\hat{z}$ and $\hat{n}$). Don't forget that basis vectors must have unit length. Now write out the matrices that convert from (x, y, z) to the $\hat{n}$ basis, rotate by an angle γ in that basis, then convert back to (x, y, z). I suggest you do this on a computer, and write a function

genrot(xyz,gamma)

that generates the individual matrices, multiplies them together, and returns the final rotation matrix in the (x, y, z) coordinate system. What is the rotation matrix for $\theta = \pi/4$, $\phi = \pi/6$, and gamma=0.01?

want to rotate by γ around axis $\hat{n}$ in θ, φ direction so...

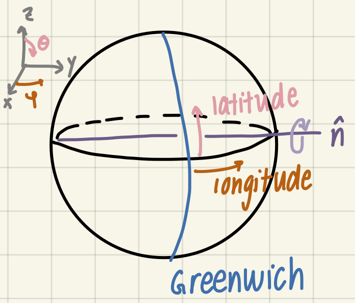
$$V_{rotated} = \underbrace{R_{n\to x} R_\gamma R_{x\to n}}_{\text{final rot. matrix}} \underset{\text{in } x,y,z}{V}$$

```python
# Functions
def R_matrix_n(angle):
    """
    When defining n, the n vector gets mapped to (1,0,0), so the equiv. to the x axis
    So we want to rotate around 'x'
    """
    r1 = [1, 0, 0]
    r2 = [0,np.cos(angle), -np.sin(angle)]
    r3 = [0,np.sin(angle), np.cos(angle)]
    return np.array([r1,r2,r3])

def genbasis(vector):
    """
    Create new basis with the input vector as the first basis vector
    Returns the change of basis matrix from xyz to the new basis
    """
    # Normalize the input vector
    vector = vector/np.linalg.norm(vector)

    # Arbitrary vector
    v = np.array([0,1,0]) if np.all(vector[1:]==[0,0]) else np.array([1,0,0])

    # Get the other basis vectors (normalized)
    r2 = np.cross(vector,v)
    r2 = r2/np.linalg.norm(r2)
    r3 = np.cross(vector,r2)

    # Change of basis matrix
    M = np.array([vector,r2,r3])

    return M
```

```python
def genrot(n, gamma):
    """
    xyz: axis we rotate around
    gamma: angle of rotation
    """
    # generate a basis with n
    M = genbasis(n)  # go from x,y,z to n basis
    M_inv = np.linalg.inv(M)  # go from n basis to x,y,z

    # generate the rotation matrix
    R = R_matrix_n(gamma)  # rotates around n by gamma, in the n basis

    return M_inv@R@M
```

```
Rotation matrix:
[[ 0.99996875 -0.00706012  0.00355713]
 [ 0.00708178  0.99995625 -0.00611112]
 [-0.00351382  0.00613612  0.999975  ]]
```

5. Pretend, in a shocking turn of events, the Earth's crust gets rotated so that the Royal Greenwich Observatory, England (current latitude 51.476852, longitude=0.000) moves to the north pole. What are Montreal's new latitude and longitude? (current: 45.50884, -73.58781). Verify that the current angle between Montreal and Greenwich corresponds to Montreal's new latitude (as it must, because the distance to the pole depends only on latitude, not longitude, and your rotation matrix had better not change relative angles between vectors). Hint - think what axis we have to be rotating about to move Greenwich to the North pole.



New coordinates: Greenwich lat = 90°
long unchanged

① convert latitude & longitude to xyz coordinates

② want to rotate along — to have north pole at Greenwich

→ Rotate around $\hat{n}$ = Greenwich × north pole

→ Get rotation matrix for rotation by an angle around n

→ So need change of basis matrix + rot. matrix

use functions from Q4

```python
import numpy as np

# Coordinates
greenwich_lat = 51.476852
greenwich_long = 0.000
mtl_lat = 45.50884
mtl_long = -73.58781
```

```python
def cartesian_to_spherical(x,y,z):
    # Radial distance
    r = np.sqrt(x**2 + y**2 + z**2)
    # Polar angle theta
    theta = np.arccos(z/r)
    # Azimuthal angle phi
    phi = np.arctan2(y,x)

    return theta, phi

def earth_to_cartesian(lat, lon):
    theta = np.pi/2-np.radians(lat)
    phi = np.radians(lon)
    x = np.sin(theta) * np.cos(phi)
    y = np.sin(theta) * np.sin(phi)
    z = np.cos(theta)
    return np.array([x, y, z])

def angle_between_vectors(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm1 = np.linalg.norm(vec1)
    norm2 = np.linalg.norm(vec2)
    cos_theta = dot_product / (norm1 * norm2)
    return np.degrees(np.arccos(cos_theta))
```

```python
# Question 5
print('\nQuestion 5')
# Convert to cartesian
g_cart = earth_to_cartesian(greenwich_lat, greenwich_long)  # Greenwich
m_cart = earth_to_cartesian(mtl_lat, mtl_long)  # Montreal

# Angle we want to rotate by (to get Greenwich at north pole)
g_theta = cartesian_to_spherical(*g_cart)[0]

# Get the axis we want to rotate around
n = np.cross(g_cart, [0,0,1])  # g x NP

# Rotation matrix in xyz coordinates
G = genrot(n,g_theta)  # to rotate around n by g_theta
print(f'Rotation matrix for greenwich at NP:\n{G}')

# New Montreal coordinates, in cartesian
m_cart_rotated = G@m_cart
# Convert back to spherical coordinates
m_theta_rotated, m_phi_rotated = cartesian_to_spherical(*m_cart_rotated)

# Get new lat & long of Montreal
mtl_lat_new = np.degrees(np.pi/2 - m_theta_rotated)
mtl_long_new = np.degrees(m_phi_rotated)
print(f'New Montreal lat: {mtl_lat_new}, long: {mtl_long_new}')

# Compare angles between Montreal and Greenwich
initial_angle = angle_between_vectors(m_cart, g_cart)
current_angle = angle_between_vectors(m_cart_rotated, G@g_cart)
print(f'Initial angle between Montreal and Greenwich: {initial_angle}')
print(f'Angle between Montreal and Greenwich: {current_angle}')
```

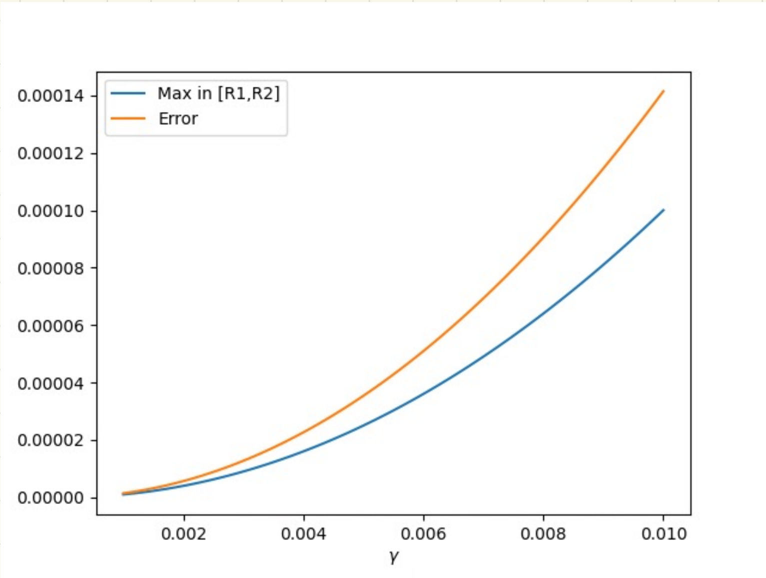
output:

```
Question 5
Rotation matrix for greenwich at NP:
[[ 0.78235659  0.         -0.62283077]
 [ 0.          1.          0.        ]
 [ 0.62283077  0.          0.78235659]]
New Montreal lat: 42.95518235099854, long: -113.29100393617745
Initial angle between Montreal and Greenwich: 47.04481764900146
Angle between Montreal and Greenwich: 47.04481764900146
```

so 90 - new latitude = angle b/w MTl & Greewich

90 - 43 = 47 ✓

6. Use your the results (and code!) to show numerically that the rotation commutation relations hold in an arbitrary coordinate system. If you haven't done this sort of thing before, you can show that as you make gamma smaller, the largest term in the error shrinks faster than the largest term in the commutator. I found gammas in the range of 0.01-0.001 were sufficient to show this clearly.



error goes to 0 faster

```
# Question 6
print('\nQuestion 6')

gamma = np.linspace(0.01, 0.001, 100)

# Get orthogonal vectors
n1 = np.array([1,0,0])
n2 = np.array([0,1,0])
n3 = np.array([0,0,1])

# Get rotation matrices, shape (N_gamma,3,3)
R1 = np.array([genrot(n1, g) for g in gamma])
R2 = np.array([genrot(n2, g) for g in gamma])
R3 = np.array([genrot(n3, g) for g in gamma])

# Commutators
R1R2 = np.array([R1[i]@R2[i] - R2[i]@R1[i] for i in range(len(gamma))])
pred = np.array([np.eye(3)-R3[i] for i in range(len(gamma))])

# Errors
errors = np.array([np.linalg.norm(R1R2[i]) for i in range(len(gamma))])
R1R2_max = np.array([np.max(np.abs(R1R2[i])) for i in range(len(gamma))])
R1R2_max_err = np.array([np.max(np.abs(R1R2[i]-pred[i])) for i in range(len(gamma))])

# Plot
plt.plot(gamma, R1R2_max, label='Max in [R1,R2]')
plt.plot(gamma, errors, label='Error')
# plt.plot(gamma, R1R2_max_err, label='Max error')
plt.xlabel(r'$\gamma$')
plt.legend()
plt.savefig('question6.png')
plt.show()
```