\( \mathbf{v}_1 \cdot \mathbf{v}_2 = 1 + 1 = 2 \) \( \| \mathbf{v}_1 \| \| \mathbf{v}_2 \| \cos \theta = 1 \) \( \sqrt{2} \) \( \cos \theta \)

\( \frac{2}{\sqrt{2} \sqrt{3}} = \cos \theta \)

\( \sqrt{\frac{2}{3}} = \cos \theta \rightarrow \theta = \cos^{-1} \left( \sqrt{\frac{2}{3}} \right) \approx 35.26^\circ \)

\( \approx 0.6155 \text{ rad} \)
\[ |\mathbf{a} + \mathbf{b}|^2 = (\mathbf{a} + \mathbf{b}) \cdot (\mathbf{a} + \mathbf{b}) = |\mathbf{a}|^2 + |\mathbf{b}|^2 + 2 \mathbf{a} \cdot \mathbf{b} \]
\[ (\mathbf{a} + \mathbf{b})^2 = \mathbf{a}^2 + \mathbf{b}^2 + 2 \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}|^2 + |\mathbf{b}|^2 + 2 |\mathbf{a}| |\mathbf{b}| \cos \theta \]

\[ \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta \]
\[
\Rightarrow |\mathbf{a} + \mathbf{b}|^2 \leq (|\mathbf{a}| + |\mathbf{b}|)^2
\]

Taylor 1.27

The numbers indicate times for puck positions, and the orientation of the turntable, as seen while standing on the ground.

This is in the frame of the turntable. Rotate the picture at left until the turntable edge marker is in the position, & note the position of the puck.
If the turntable rotates rapidly...
Taylor 1.30

Mass 2 is initially at rest, mass 0 has velocity \( \dot{V} \), so \( \dot{p}_{\text{init}} = 0 + m_0 \dot{V} \)

The final momentum is \( \dot{p}_{\text{final}} = (m_0 + m_2) \dot{V}' \)

by conservation of momentum \( \dot{p}_{\text{init}} = \dot{p}_{\text{final}} \)

\[ m_0 \dot{V} = (m_0 + m_2) \dot{V}' \]

\[ \dot{V}' = \frac{m_1}{m_1 + m_2} \dot{V} \]
\[ F_x = -mg \sin \phi = m \ddot{x} \rightarrow x = V_{xo} t - \frac{1}{2} g_x t^2 \]
\[ F_y = -mg \cos \phi = m \ddot{y} \]
\[ y = V_{yo} t - \frac{1}{2} g_y t^2 \]

\[ y = 0 : 0 = V_{yo} t - \frac{1}{2} g_y t^2 = t \left[ V_{yo} - \frac{1}{2} g_y t \right] \]

\[ t_F = \frac{2V_{yo}}{g_y} = \frac{2V_0}{g} \frac{\sin \theta}{\cos \phi} \]

Range = \[ X(t_F) = V_{xo} \left( \frac{2V_{yo}}{g_y} \right) - \frac{1}{2} g_x \left( \frac{2V_{yo}}{g_y} \right)^2 \]

\[ = \frac{2V_0^2 \cos \theta \sin \theta}{g \cos \phi} - \frac{1}{2} g \sin \phi \left[ 4V_0^2 \sin^2 \theta \right. \]
\[ \left. - \frac{1}{g^2 \cos^2 \phi} \right] \]

\[ = \frac{2V_0^2 \sin \theta}{g \cos^2 \phi} \left[ \cos \theta \cos \phi - \sin \theta \sin \phi \right] \]

\[ = \frac{2V_0^2 \sin \theta}{g \cos^2 \phi} \cos (\theta + \phi) \]
\[ O = \frac{d}{d\theta} \left( \frac{2V_0^2}{g} \sin \theta \cos (\theta + \phi) \cos^2 \phi \right) \]

\[ O = \frac{d}{d\theta} \sin \theta \cos (\theta + \phi) = \cos \theta \cos (\theta + \phi) - \sin \theta \sin (\theta + \phi) \]

\[ = \cos (\theta + \theta + \phi) \]

condition for \( \frac{dR}{d\theta} = 0 \) is \( 2\theta + \phi = \pi/2 \)

\[ \Rightarrow \theta_{optimal} = \frac{\pi}{4} - \frac{\phi}{2} \]

\[ R(\theta = \frac{\pi}{4} - \frac{\phi}{2}) = \frac{2V_0^2}{g} \frac{\sin \left( \frac{\pi}{4} - \frac{\phi}{2} \right)}{\cos^2 \phi} \cos \left( \frac{\pi}{4} - \frac{\phi}{2} + \phi \right) \]

\[ \cos \left( \frac{\pi}{4} + \phi \right) = \sin \left( \frac{\pi}{4} - \phi \right) \]

\[ R_{opt} = \frac{2V_0^2}{g} \frac{\cos^2 \left( \frac{\pi}{4} + \phi \right)}{\cos^2 \phi} = \frac{2V_0^2}{g} \frac{\cos \frac{\pi}{4} \cos \frac{\phi}{2} - \sin \frac{\pi}{4} \sin \frac{\phi}{2}}{\cos \phi} \]

\[ = \frac{V_0^2}{g} \left( \frac{\cos \frac{\phi}{2} - \sin \frac{\phi}{2}}{\cos \phi} \right)^2 = \frac{V_0^2}{g^2} \frac{\cos^2 \frac{\phi}{2} + \sin^2 \frac{\phi}{2} - 2 \cos \frac{\phi}{2} \sin \frac{\phi}{2}}{\cos^2 \phi} \]

\[ = \frac{V_0^2}{g} \frac{1 - 2 \cos \frac{\phi}{2} \sin \frac{\phi}{2}}{\cos^2 \phi} \]

\[ = \frac{V_0^2}{g} \frac{1 - \sin \phi}{\cos^2 \phi} \]
\[ R_{\text{opt}} = \frac{V_0^2}{g} \frac{1 - \sin \phi}{\cos^2 \phi} \times \frac{1 + \sin \phi}{1 + \sin \phi} \]

\[ = \frac{V_0^2}{g} \frac{1 - \sin \phi}{\cos^2 \phi (1 + \sin \phi)} \]

\[ = \frac{V_0^2}{g} \frac{1}{1 + \sin \phi} \]
Taylor 1.46

a) In the lab frame:

\[ r(t) = R - V_0 t \]
\[ \phi(t) = \tan^{-1} \left( \frac{0}{R - V_0 t} \right) \]
\[ = 0 \]

b) In the \( S' \) frame

\[ r'(t) = r = R - V_0 t \]
\[ \phi'(t) = \omega t \]

See 1.27 for the sketch. The \( S' \) frame is not inertial since a particle with no force acting on it does not follow a constant velocity path.
Evolving the equation of motion in the small angle approximation through two full periods. The angle should return to 0, so any deviation from 0 is due to the finite time step. The time interval is divided into \( N_t \) intervals. Start with \( N_t = 100 \), and then keep multiplying by 2, recalculating \( \phi(2T) \) each time. Continue until \( N_t > 500000 \), and print the result for each \( N_t \), giving a table of \( \phi(2T) \) vs the number of time steps:

<table>
<thead>
<tr>
<th>( N_t )</th>
<th>( \phi(2T) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.00592811984592</td>
</tr>
<tr>
<td>200</td>
<td>0.00147787393642</td>
</tr>
<tr>
<td>400</td>
<td>0.000369208867927</td>
</tr>
<tr>
<td>800</td>
<td>9.22859930853e-05</td>
</tr>
<tr>
<td>1600</td>
<td>2.30704842219e-05</td>
</tr>
<tr>
<td>3200</td>
<td>5.76755828919e-06</td>
</tr>
<tr>
<td>6400</td>
<td>1.44188426056e-06</td>
</tr>
<tr>
<td>12800</td>
<td>3.60471371971e-07</td>
</tr>
<tr>
<td>25600</td>
<td>9.01224193425e-08</td>
</tr>
<tr>
<td>51200</td>
<td>2.25325652761e-08</td>
</tr>
<tr>
<td>102400</td>
<td>5.55924864275e-09</td>
</tr>
<tr>
<td>204800</td>
<td>1.17618609246e-09</td>
</tr>
<tr>
<td>409600</td>
<td>2.1874207767e-10</td>
</tr>
</tbody>
</table>

The last value of \( N_t \) was 409600 which gave \( \phi(2T) \) very close to zero, so let's use that value for \( N_t \).

The initial conditions are \( \phi_0 = 0.349065850399 \) radians, \( d\phi/dt = 0.0 \) radians/sec.

At the end of 2 periods, the error in the approximate solution is:

\[
\frac{[\phi(\text{approx}) - \phi(\text{exact})]/\phi(\text{exact})}{0.00455821910438}
\]

So the small angle approximation is better than 1% accurate in this case.
#!/usr/bin/python
from __future__ import division, print_function

import numpy as np
import matplotlib.pylab as plt

R = 5.  # radius in m
g = 9.8  # acceleration of gravity in m/s^2
w2 = g / R  # omega^2 in the small angle approximation
approx = False  # if True, use approximation f(x) = -w2 * x

# return the force as a function of position and time
def f(phi):
    if approx:
        return -w2 * phi
    return -w2 * np.sin(phi)

# integrate d^2phi/dt^2 = f(phi) from time t_i to time t_f using Euler's method
# phi_i    = initial position at t_i
# phidot_i = initial velocity at t_i
# Nt  = number of time steps to use
# this is the same as step() except that instead of returning the final phi value
# it fills numpy arrays with the values of t and phi to be plotted
def evolve(phi_i, phidot_i, t_i, t_f, Nt, t_array, phi_array):
    dt = ( t_f - t_i ) / Nt
    phi0 = phi_i                  # phi(0) initial position
    phi1 = phi_i + phidot_i * dt  # phi(dt) from initial phidot

    npoints = t_array.size    # number of points to be plotted
    m = Nt / npoints         # plot a point every m time steps

    for n in range(1,Nt):
        t = t_i + n * dt
        phi2 = f(phi1)*(dt**2) - phi0 + 2.*phi1  # phi_{n+1}
        phi0 = phi1
        phi1 = phi2
    return phi1   # this is the value at the final time, t = t_f
phi₀ = phi₁  # φ_{n-1}
phi₁ = phi₂  # φ_{n}

if n % m == 0:
    i = int(n/m)
    t_array[i] = t_n
    phi_array[i] = phi₁

    t_array[0], t_array[npoints-1] = t_i , t_f
    phi_array[0], phi_array[npoints-1] = phi₀ , phi₁

# Start off at x(0) = 0
phi₀ = 0.
phidot₀ = 1.  # has an initial velocity v(0) = 1 m/s

# the period

# try increasing Nt to see how the accuracy increases
Nt = 100
approx = True
print("Evolve the equation of motion in the small angle approximation through")
print("two full periods. The angle should return to 0, so any deviation from 0")
print("is due to the finite time step. The time interval is divided into Nt")
print("intervals.")
print("Start with Nt = 100, and then keep multiplying by 2, recalculating phi(2T)")
print("each tim.")
print("Continue until Nt > 500000, and print the result for each Nt, giving a table")
print("phi(2T) vs the number of time steps:
")
print('Nt	phi(2T)')
while Nt < 500000:
    # print solution at t = 2 * period (where it should be 0 for the approx. case)
    print(Nt,'\t',step(phi₀, phidot₀, t_i, t_f, Nt) )
    Nt *= 2
    # keep the maximum value of Nt for the next step...

print("\nThe last value of Nt was 409600 which gave phi(2T) very close to zero,"")
print("so let's use that value for Nt")

    t_approx = np.linspace(0.,0.,100) # create an array for 100 t-values to plot
    phi_approx = np.linspace(0.,0.,100) # create an array for 100 x-values to plot

    # Now do the exact case. Initial conditions need to be set:
    phi₀ = (20./360.)*2.*np.pi # the initial angle of 20 degrees
    phidot₀ = 0.  # put something else here
    print("the initial conditions are phi₀ = ",phi₀," radians, dphi/dt = ", phidot₀," radians/sec")

approx = True
evolve(phi0, phidot0, t_i, t_f, Nt, t_approx, phi_approx )

t_exact = np.linspace(0,0,100) # create an array for 100 t-values to plot
phi_exact = np.linspace(0,0,100) # create an array for 100 x-values to plot

approx = False
evolve(phi0, phidot0, t_i, t_f, Nt, t_exact, phi_exact )

plt.plot(t_exact, phi_exact, linestyle='-',color='red',label='exact')
plt.plot(t_approx, phi_approx, linestyle='-',color='blue',label='approx')

print("at the end of 2 periods, the error in the approximate solution is:")
print("[phi(approx) - phi(exact)]/phi(exact) = ",(phi_approx[99]-phi_exact[99])/phi_exact[99])
print("So the small angle approximation is better than 1\% accurate in this case")
plt.xlabel('$t$ [sec]')
plt.ylabel('$\phi$')
plt.legend(frameon=False)
plt.show()
exit(0)