In [1]:
```
import gym
import pybulletgym
import pybulletgym.envs
import numpy as np
import math
import matplotlib.pyplot as plt
from numpy.linalg import pinv
```

In [2]:
```
env = gym.make("ReacherPyBulletEnv-v0")
#env.render()
env.reset()
```

```
current_dir=/home/apurba/.virtualenvs/276c_assgn/lib/python3.6/site-p
ackages/pybullet_envs/bullet
options=
```

Out[2]:
```
array([-0.21537443, -0.07986251,  0.14044574,  0.18985098,  0.3391394
3,
        0.94073612,  0.        ,  0.59053606,  0.        ])
```

In [4]:
```
def getForwardModel (q0, q1, l0 = 0.1, l1 = 0.11):
    """
    Forward kinematics
    :param q0: Central Joint angle
    :param q1: Shoulder Joint angle
    :param l0: Arm Length l0
    :param l1: Arm Length l1
    :return: x1,x2 position
    """
    H_A_B = np.array([[np.cos(q0), -1 * np.sin(q0), l0 * np.cos(q0)],
                      [ np.sin(q0),  1 * np.cos(q0), l0 * np.sin(q0)],
                      [          0,              0,                1
]]);
    V_B =  np.array([[l1 * np.cos(q1)],
                     [l1 * np.sin(q1)],
                     [1]]);

    V_A =  np.matmul(H_A_B, V_B)
    final = V_A[:2, :]/V_A[2,:];
    return final
```

In [74]:
```
def getJacobian (q0, q1, l0 = 0.1, l1 = 0.11):
    """
    Jacobian matrix
    :param q0: Central Joint angle
    :param q1: Shoulder Joint angle
    :param l0: Arm Length l0
    :param l1: Arm Length l1
    :return: Jacobian matrix
    """

    #J = np.zeros(shape=(2,2))
    J =  np.array([[-1*l1*np.sin(q1+q0) - l0*np.sin(q0), -1*l1*np.sin
(q1+q0)],
                   [   l1*np.cos(q1+q0) + l0*np.cos(q0),    l1*np.cos
(q1+q0)]]);
    return J;
```

In [8]:
```
def getIK_analytical (x0 , x1, l0 = 0.1, l1 = 0.11):
    """
    Inverse kinematics (using Jacobian Inverse)
    :param x0: Current x0
    :param x1: Current x1
    :param l0: Arm Length l0
    :param l1: Arm Length l1
    """
    q1 = np.arccos((x0*x0 + x1*x1 - l0*l0 - l1*l1)/2*l1*l0)
    if((x0*x0 + x1*x1 - l0*l0 - l1*l1)/2*l1*l0 > 1):
        q1 = np.arccos(1)
    if((x0*x0 + x1*x1 - l0*l0 - l1*l1)/2*l1*l0 < -1):
        q1 = np.arccos(-1)
    q0 = np.arctan(x1/x0) - np.arctan(l1 * np.sin(q1)/(l0 + l1*np.cos
(q1)))
    return(q0,q1)
```

In [9]:
```
def getIK(q0, q1, delta_x, delta_y):
    """
    Inverse kinematics (using Jacobian Inverse)
    :param q0: Current Central Joint angle
    :param q1: Current Shoulder Joint angle
    :param delta_x: Error in position x
    :param delta_y: Error in position y

    :return: Joint angles
    """



    inv_jacobian = np.linalg.pinv(getJacobian(q0, q1))
    del_pos = np.array([[delta_x], [delta_y]])
    del_q = np.matmul(inv_jacobian,del_pos)

    return del_q
```

In [7]:
```
def x_ref(theta):
    """
    Path trajectory
    :param theta: Current theta angle
    :return: position x
    """
    return (0.19 + 0.02 * np.cos(4*theta))*np.cos(theta);


def y_ref(theta):
    """
    Path trajectory
    :param theta: Current theta angle
    :return: position y
    """
    return (0.19 + 0.02 * np.cos(4* theta))*np.sin(theta);

previous_ind = 0
steps = 0
done = False
```

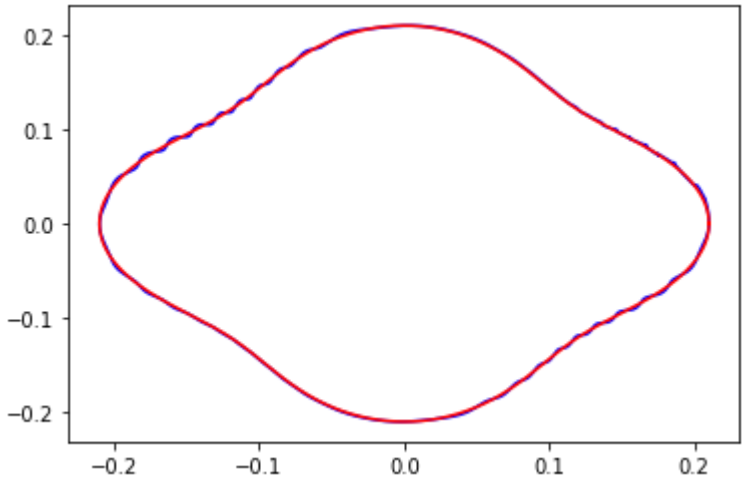
In [112]:
```
def mse (referencex , generatedx , referencey , generatedy):
    """
    MSE error
    :param referencex: xreference trajectory
    :param generatedx : xgenerated trajectory
    :param referencey: yreference trajectory
    :param generatedy : ygenerated trajectory
    :return: MSE error
    """
    sum = 0
    for i in range(0, len(referencex)):
        sum = sum + (referencex[i] - generatedx[i]) ** 2 + (reference
y[i] - generatedy[i]) ** 2
    return np.sqrt(sum/len(referencex))
```

In [113]:
```
def problem_1(k_pos, k_vel, steps = 5000, ini_theta = -np.pi):

    """
    problem_1 trajectory with input as error in end effector position
    :param k_pos: Position error gain value
    :param k_val: Velocity error gain value
    :steps: No of steps to sample
    :param ini_theta: Initial theta
    """
    step = float((np.pi*2)/steps);
    x = []
    y = []
    xref = []
    yref = []

    xref0 = x_ref(-1* np.pi);
    yref0 = y_ref(-1* np.pi);


    env.unwrapped.robot.central_joint.reset_position(0,0);
    env.unwrapped.robot.elbow_joint.reset_position(0,0);

    for theta in np.arange(0,2*np.pi,step):

        k_pos = 25;
        k_vel = 0.9;
        k_pos = np.diag([k_pos, k_pos])
        k_vel = np.diag([k_vel, k_vel])
        q0 , q0dot = env.unwrapped.robot.central_joint.current_positi
on();
        q1 , q1dot = env.unwrapped.robot.elbow_joint.current_position
();
        #2*1
        end_effector = getForwardModel(q0,q1);
        x.append(end_effector[0][0]);
        y.append(end_effector[1][0]);
        Jacobian = getJacobian(q0,q1);
        end_effector_vel = np.matmul(Jacobian,np.array([[q0dot],[q1do
t]]))

        e_pos = np.array([x_ref(theta)-end_effector[0][0], y_ref(thet
a)-end_effector[1][0]]).reshape(-1, 1)
        e_vel = np.array([-1 * end_effector_vel[0][0], -1 * end_effec
tor_vel[0][0]]).reshape(-1,1)


        F = np.matmul(k_pos, e_pos) + np.matmul(k_vel, e_vel)
        xref.append(x_ref(theta))
        yref.append(y_ref(theta))

        Torque = np.matmul(Jacobian.T, F);
        action = [Torque[0][0], Torque[1][0]];
        env.step(action)

    print("The MSE error is", mse(xref,x, yref, y))
```

```
    plt.plot(x,y, color='b');
    plt.plot(xref, yref, color='r')
    plt.show()
```

In [114]:
```
problem_1(k_pos = 15, k_vel = 0.95);
```

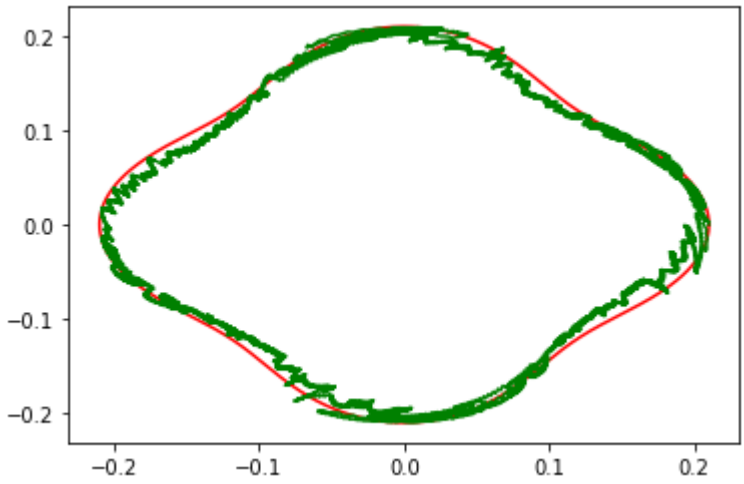
```
The MSE error is 0.0013336163494978 38
```



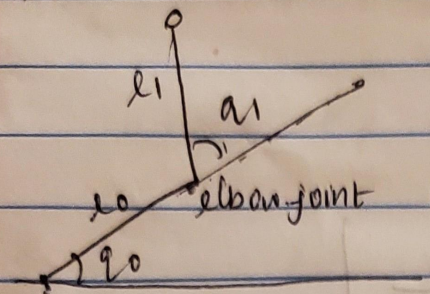
In [115]:
```
def problem_2(k_pos, k_vel, steps = 5000, ini_theta = -np.pi):
    """
    problem_2 trajectory with input as error in joint angles
    :param k_pos: Angular Position error gain value
    :param k_val: Angular Velocity error gain value
    :steps: No of steps to sample
    :param ini_theta: Initial theta
    """

    step = float((np.pi*2)/steps);
    x = []
    y = []
    xref = []
    yref = [];
    env.unwrapped.robot.central_joint.reset_position(0,0);
    env.unwrapped.robot.elbow_joint.reset_position(0,0);


    for theta in np.arange(0,2*np.pi,step):
        k_pos  = 15
        k_vel = 0.95

        k_pos = np.diag([k_pos, k_pos])
        k_vel = np.diag([k_vel, k_vel])
        #this gives the joint angles and the joint angle velocity
        q0 , q0dot = env.unwrapped.robot.central_joint.current_positi
on();
        q1 , q1dot = env.unwrapped.robot.elbow_joint.current_position
();
        #2*1
        end_effector = getForwardModel(q0,q1);
        Jacobian = getJacobian(q0,q1);
        x.append(end_effector[0][0]);
        y.append(end_effector[1][0]);
        xref.append(x_ref(theta))
        yref.append(y_ref(theta))

        #get the errors
        del_x, del_y = x_ref(theta) - end_effector[0][0], y_ref(theta
) - end_effector[1][0]
        angleref_error = getIK(q0 ,q1 , del_x, del_y);
        e_angle = np.array([angleref_error[0][0], angleref_error[1][0
] ]).reshape(-1, 1)
        e_vel = np.array([-q0dot, -q1dot]).reshape(-1, 1)
        #PD controller
        Torque = np.matmul(k_pos, e_angle) + np.matmul(k_vel, e_vel)
# PD Controller
        Torque = Torque.reshape(-1)
        env.step(Torque)

    print("The MSE error is", mse(xref,x, yref, y))

    plt.plot(xref, yref, color='r')
    plt.plot(x,y, color = 'g')

    plt.show()
```

In [116]:
```
problem_2(k_pos = 15, k_vel = 0.95)
```

```
The MSE error is 0.018212144470918967
```



Q1) Forward Kinematics



Suppose the B frame is the one with origin as elbow joint

$$\therefore V_B = \begin{bmatrix} l_1 \cos q_1 \\ l_1 \sin q_1 \end{bmatrix}$$

To transform the vector into the A frame with origin as at central joint, the Transformation matrix is

$$= \left[\begin{array}{cc|c} \cos q_0 & -\sin q_0 & l_0 \cos q_0 \\ \sin q_0 & \cos q_0 & l_0 \sin q_0 \\ \hline 0 & 0 & 1 \end{array}\right]$$

(Rotation: $\cos q_0$, $-\sin q_0$, $\sin q_0$, $\cos q_0$, 0, 0 — Translation: $l_0 \cos q_0$, $l_0 \sin q_0$, 1)

$$= \begin{bmatrix} \cos q_0 & -\sin q_0 & l_0 \cos q_0 \\ \sin q_0 & \cos q_0 & l_0 \sin q_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l_1 \cos q_1 \\ l_1 \sin q_1 \\ 1 \end{bmatrix}$$

$$F_x = l_1 \cos(q_0 + q_1) + l_0 \cos q_0$$

$$F_y = l_1 \sin(q_0 + q_1) + l_0 \sin q_0$$

2. Jacobian expression

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial q_0} & \frac{\partial F_1}{\partial q_1} \\ \frac{\partial F_2}{\partial q_0} & \frac{\partial F_2}{\partial q_1} \end{bmatrix}$$

$$= \begin{bmatrix} -l_1 \sin(q_1 + q_0) - l_0 \sin(q_0) & -l_1 \sin(q_1 + q_0) \\ l_1 \cos(q_1 + q_0) + l_0 \cos(q_0) & l_1 \cos(q_1 + q_0) \end{bmatrix}$$

3. For the inverse kinematics equation, we can compute the q_1, q_2 from the (x, y) position of the end effector. The following equations compute the q_0, q_1

$$q_1 = \cos^{-1}\left(\frac{x^2 + y^2 - l_0^2 - l_1^2}{2 l_0 l_1}\right)$$

$$q_0 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{l_1 \sin q_1}{l_1 + l_0 \cos q_1}\right)$$

In this case

$$\begin{bmatrix} \Delta x_1 \\ \Delta y_1 \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial F_1}{\partial q_0} & \frac{\partial F_1}{\partial q_1} \\ \frac{\partial F_2}{\partial q_0} & \frac{\partial F_2}{\partial q_1} \end{bmatrix}}_{J} \begin{bmatrix} \Delta q_0 \\ \Delta q_1 \end{bmatrix}$$

Therefore in inverse kinematics, we consider the

$$\Delta x = x_{target} - x_{current}$$

$$\Delta y = y_{target} - y_{current}$$

$$\begin{bmatrix} \Delta q_0 \\ \Delta q_1 \end{bmatrix} = [J^{-1}] \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \end{bmatrix}$$

From the initial q_0, q_1 we step to $q_0 + \Delta q_0, q_1 + \Delta q_1$

This is done in steps to get the final q_0 and q_1

In the following question, we take the no. of steps = 2000

4. Gains used $K_p = 15$ and $K_d = 0.95$

5. Gains_used $K_p = 15$, $K_d = 0.95$

For forward kinematics ~~$\Delta q_0, \Delta q_1$ = get_forward_mode~~
$\Delta x, \Delta x_1$

we reset the bot to the initial position. Post that we compute the error with x_ref(theta) - end_effector(x) and
y_ref(theta) - end_effector(y)

1.4 $\Rightarrow$ $$F = \begin{bmatrix} K_p & 0 \\ 0 & K_p \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} + \begin{bmatrix} K_d & 0 \\ 0 & K_d \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

Error in velocity = v_{x_ref} (theta) $\to 0$ - end_effector_vel(x)
v_{y_ref} (theta) $\to 0$ - end_effector_vel(y)

$$\text{end\_eff\_vel} = J * \begin{bmatrix} q_0 dot \\ q_1 dot \end{bmatrix}$$

$$\boxed{\tau = J^T F}$$ which is the input.

1.5 $\Rightarrow$ For inverse kinematics, we record the q_0 and q_1 angles $\begin{bmatrix} \Delta q_0 \\ \Delta q_1 \end{bmatrix} = [J^{-1}] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$, so corresponding to the delx and dely we get the error in angl

The error in velocity = $\begin{bmatrix} 0 - q_0 dot \\ 0 - q_1 dot \end{bmatrix}$

$$\tau = \begin{bmatrix} K_p & 0 \\ 0 & K_p \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} + \begin{bmatrix} K_d & 0 \\ 0 & K_d \end{bmatrix} \begin{bmatrix} -q_0 dot \\ q_1 dot \end{bmatrix}$$