Jack Ninteman & Aly Khater
ECEN524

‘ Weekly Assignment 2

Problem 1

Dynamic Movement Primitives (DMPs) represent a new and evolving mathematical approach to training artificial systems to complete complex tasks in an adaptable and creative manner. The inspiration for DMPs stems from the incredible versatility demonstrated by biological systems when attempting new and challenging tasks. As human-robot interaction continues to become increasingly more common, the ability for robots to complete tasks with dexterity and creativity is imperative to ensure the safety of the users. The further development of DMPs could lead to safer work environments in many high risk fields, as well as an increase in robot’s ability to adapt to meticulous and complex tasks.

The specific goals for the DMP in the development of these nonlinear attractor systems are as follows:

- point attractor(discrete) and cycle attractors(periodic) need representation.
- Autonomous, without explicit time independence.
- Stability through multiple systems
- Simplify parameters of the system
- Needs coupling terms for closed-loop
- Real-time computation of control parameters
- Time Invariance

In developing the model, researchers utilized a damped spring model system:

$$\tau \ddot{y} = \alpha_z(\beta_z(g - y) - \dot{y}) + f,$$

where tau is a time constant, alpha and beta are positive gain constants. These values are explored further in our testing of periodic DMPs in Problem 2. The spring-damp system showcases a simple motor control problem.
'
There exists two types of DMP formulations: discrete and periodic motions. A discrete DMP is used for point-to-point motions in a stable dynamical system utilizing a time constant. The pipeline of a discrete DMP starts with the classical in real space, where training data is stored into a linear, second order system with the addition of a forcing term. The next step in the pipeline moves to the Cartesian space in quaternion and rotation based orientation trajectory. The quaternion trajectory gives the orientation of the robot's end-effector, and the rotation trajectory provides the representation of the orientation trajectories. The pipeline ends with Symmetric Positive Definite (SPD) trajectory space to encode robotic manipulation profiles, where stiffness, inertia, and other data is encapsulated in the form of these SPD matrices [1].

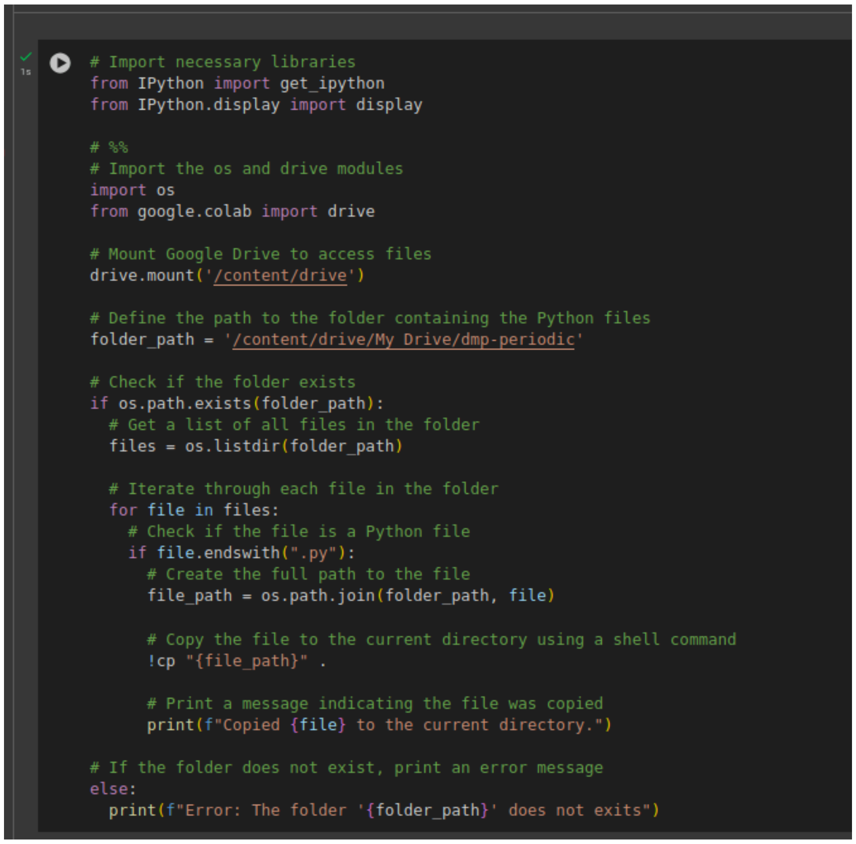
Periodic DMPs, unlike the discrete DMPs, utilizes frequency of trajectory execution instead of time constants. There exists the classical periodic DMP, similar to discrete with the addition of the frequency term, contains weights that are distributed equally along the phase space of the trajectory, and a term to modify the amplitude of the periodic signal. The periodic DMP also contains orientations in the form of quaternions and rotations, but unlike the Cartesian, the matrices are constrained. Quaternions utilize a new matrix of frequencies and amplitude modulators in the equations.

In addition to the two main types of DMPs, many strategies have been developed to make the method adapt better to specific needs and situations. One example is the addition of a dynamic scaling factor to allow the DMP to adapt to different start and goal points. Another technique involves using via points to guarantee the robot passes through a select number of points in space during its motion. This could be very useful for object avoidance when the robot needs to be mindful of its environment, as is the case with cooperative robotics. In many cases, we might require the robot to complete various unique movements in short succession to form one complex task, which may not be achievable using only one DMP. In order to implement this, a few different selection policies have been developed to switch between multiple DMPs, each one tuned to control a specific movement. One example of these policies is known as Velocity Threshold. This policy exploits the knowledge that the robot must slow to a stop as it approaches the end of a given movement. Once the robot’s velocity drops below some threshold value, a different DMP will be used to achieve a different movement, and this process will repeat until the complex task is completed. These strategies are just a small subset of the numerous DMP implementations being studied by researchers, each with their own unique benefits.

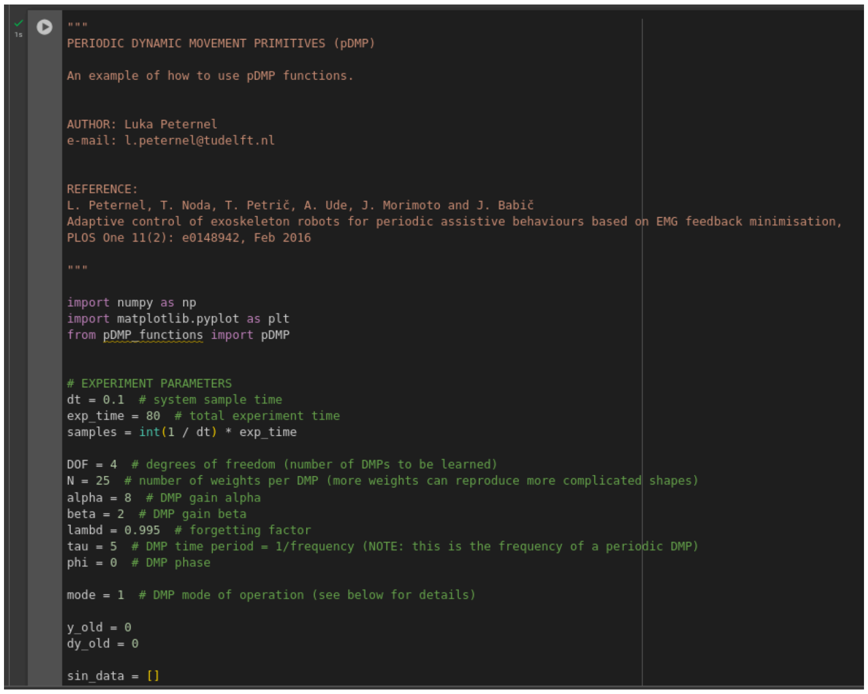
With the expansive selection of DMP variations, there needs to be some classification of use cases to help the user determine which one to implement. At the highest level, use cases are divided into two sections: passive environment tasks, where the environment is not actively introducing dynamics to the system, and co-manipulation tasks, where the user is in active contact with the robot. Additional subsections include human body augmentation/rehabilitation, teleoperation, motion analysis/recognition, high DoF robots, and autonomous driving and field robotics. In each of these cases, the use of DMPs must be carefully considered, and the correct variation must be chosen in order to fully exploit the benefits. In the case of passive environment tasks, most uses require control of both position and force. In order to achieve this, specific force and position trajectories must be derived to complete the task using human-like skills. Once this is done, DMPs may be used to control both position and force profiles and complete the task. Alternatively, in the case of co-manipulation tasks, there are many approaches to train the robot. One of the most common is kinesthetic guidance, where the human operator physically holds the robot and guides it along its motion. This is a very effective training approach, but also introduces some safety concerns due to the direct human-robot interaction. Other approaches include motion detection through optical sensors and audio recognition via spoken instructions. Regardless of the method, the robot must be extremely well trained to ensure the safety of the human user.

Problem 2

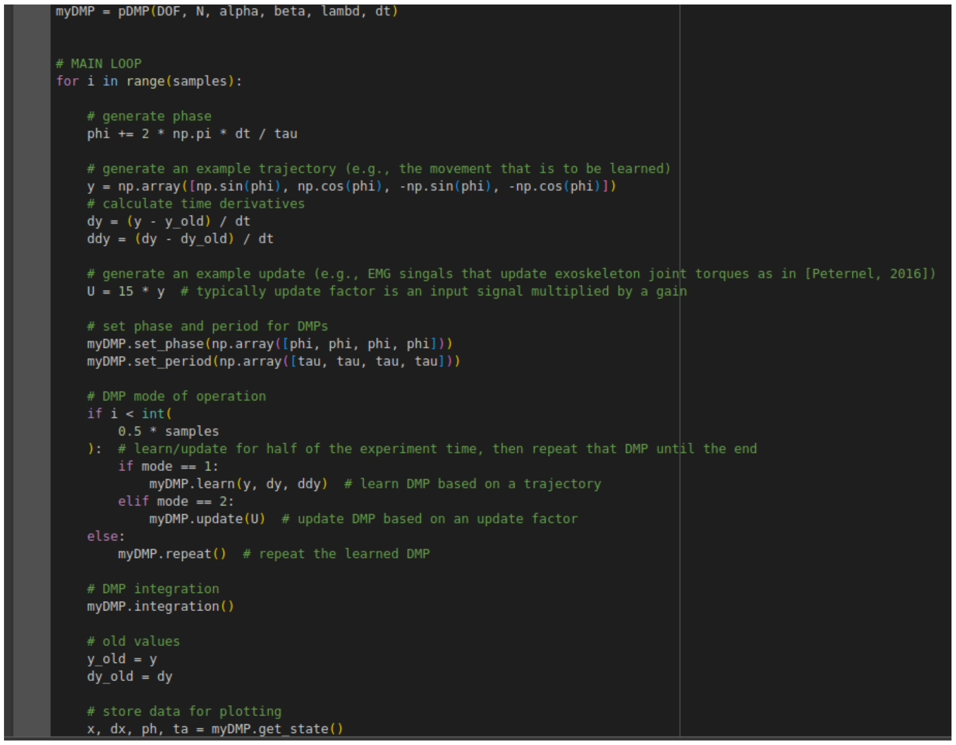
1. Link to our selected package: https://gitlab.com/dmp-codes-collection/dmp-periodic
2. Screenshots of code executed via Colab:



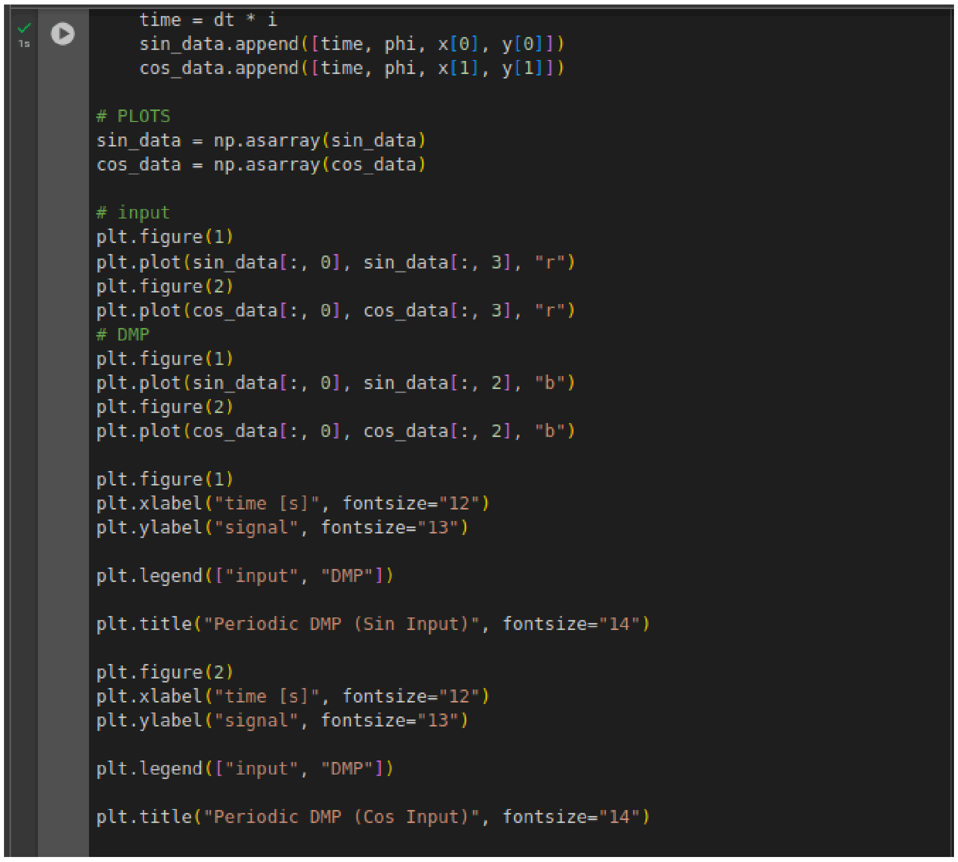
```python
# Import necessary libraries
from IPython import get_ipython
from IPython.display import display

# %%
# Import the os and drive modules
import os
from google.colab import drive

# Mount Google Drive to access files
drive.mount('/content/drive')

# Define the path to the folder containing the Python files
folder_path = '/content/drive/My Drive/dmp-periodic'

# Check if the folder exists
if os.path.exists(folder_path):
  # Get a list of all files in the folder
  files = os.listdir(folder_path)

  # Iterate through each file in the folder
  for file in files:
    # Check if the file is a Python file
    if file.endswith(".py"):
      # Create the full path to the file
      file_path = os.path.join(folder_path, file)

      # Copy the file to the current directory using a shell command
      !cp "{file_path}" .

      # Print a message indicating the file was copied
      print(f"Copied {file} to the current directory.")

# If the folder does not exist, print an error message
else:
  print(f"Error: The folder '{folder_path}' does not exits")
```



```python
"""
PERIODIC DYNAMIC MOVEMENT PRIMITIVES (pDMP)

An example of how to use pDMP functions.


AUTHOR: Luka Peternel
e-mail: l.peternel@tudelft.nl


REFERENCE:
L. Peternel, T. Noda, T. Petrič, A. Ude, J. Morimoto and J. Babič
Adaptive control of exoskeleton robots for periodic assistive behaviours based on EMG feedback minimisation,
PLOS One 11(2): e0148942, Feb 2016

"""

import numpy as np
import matplotlib.pyplot as plt
from pDMP_functions import pDMP


# EXPERIMENT PARAMETERS
dt = 0.1  # system sample time
exp_time = 80  # total experiment time
samples = int(1 / dt) * exp_time

DOF = 4  # degrees of freedom (number of DMPs to be learned)
N = 25  # number of weights per DMP (more weights can reproduce more complicated shapes)
alpha = 8  # DMP gain alpha
beta = 2  # DMP gain beta
lambd = 0.995  # forgetting factor
tau = 5  # DMP time period = 1/frequency (NOTE: this is the frequency of a periodic DMP)
phi = 0  # DMP phase

mode = 1  # DMP mode of operation (see below for details)

y_old = 0
dy_old = 0

sin_data = []
```

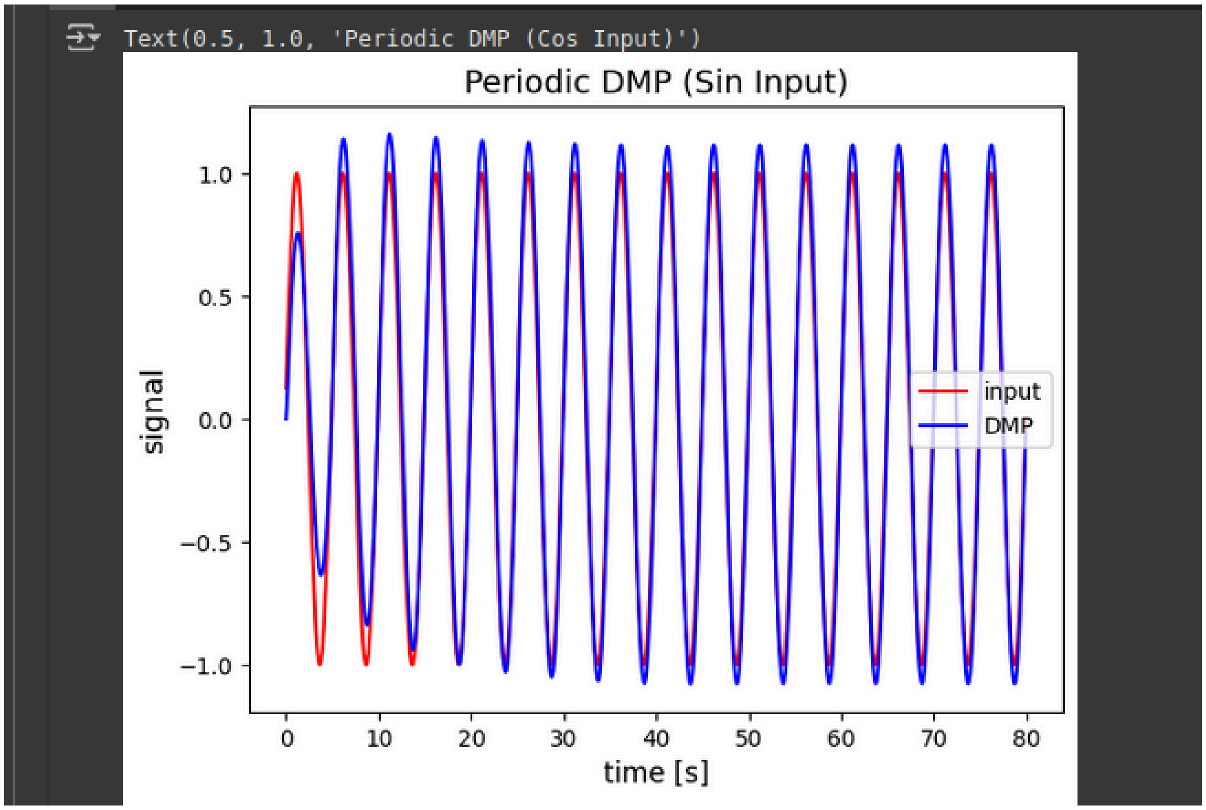


```python
myDMP = pDMP(DOF, N, alpha, beta, lambd, dt)


# MAIN LOOP
for i in range(samples):

    # generate phase
    phi += 2 * np.pi * dt / tau

    # generate an example trajectory (e.g., the movement that is to be learned)
    y = np.array([np.sin(phi), np.cos(phi), -np.sin(phi), -np.cos(phi)])
    # calculate time derivatives
    dy = (y - y_old) / dt
    ddy = (dy - dy_old) / dt

    # generate an example update (e.g., EMG singals that update exoskeleton joint torques as in [Peternel, 2016])
    U = 15 * y  # typically update factor is an input signal multiplied by a gain

    # set phase and period for DMPs
    myDMP.set_phase(np.array([phi, phi, phi, phi]))
    myDMP.set_period(np.array([tau, tau, tau, tau]))

    # DMP mode of operation
    if i < int(
        0.5 * samples
    ):  # learn/update for half of the experiment time, then repeat that DMP until the end
        if mode == 1:
            myDMP.learn(y, dy, ddy)  # learn DMP based on a trajectory
        elif mode == 2:
            myDMP.update(U)  # update DMP based on an update factor
    else:
        myDMP.repeat()  # repeat the learned DMP

    # DMP integration
    myDMP.integration()

    # old values
    y_old = y
    dy_old = dy

    # store data for plotting
    x, dx, ph, ta = myDMP.get_state()
```

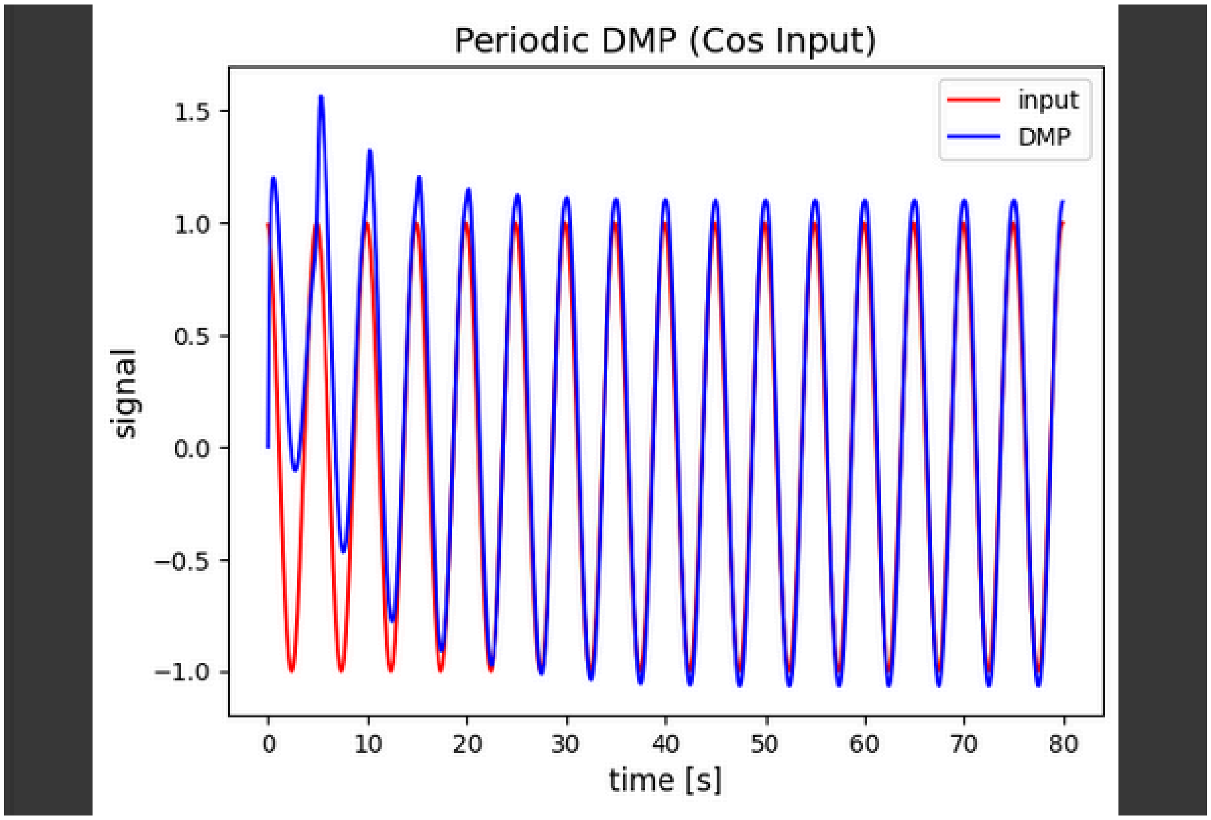


```python
    time = dt * i
    sin_data.append([time, phi, x[0], y[0]])
    cos_data.append([time, phi, x[1], y[1]])

# PLOTS
sin_data = np.asarray(sin_data)
cos_data = np.asarray(cos_data)

# input
plt.figure(1)
plt.plot(sin_data[:, 0], sin_data[:, 3], "r")
plt.figure(2)
plt.plot(cos_data[:, 0], cos_data[:, 3], "r")
# DMP
plt.figure(1)
plt.plot(sin_data[:, 0], sin_data[:, 2], "b")
plt.figure(2)
plt.plot(cos_data[:, 0], cos_data[:, 2], "b")

plt.figure(1)
plt.xlabel("time [s]", fontsize="12")
plt.ylabel("signal", fontsize="13")

plt.legend(["input", "DMP"])

plt.title("Periodic DMP (Sin Input)", fontsize="14")

plt.figure(2)
plt.xlabel("time [s]", fontsize="12")
plt.ylabel("signal", fontsize="13")

plt.legend(["input", "DMP"])

plt.title("Periodic DMP (Cos Input)", fontsize="14")
```

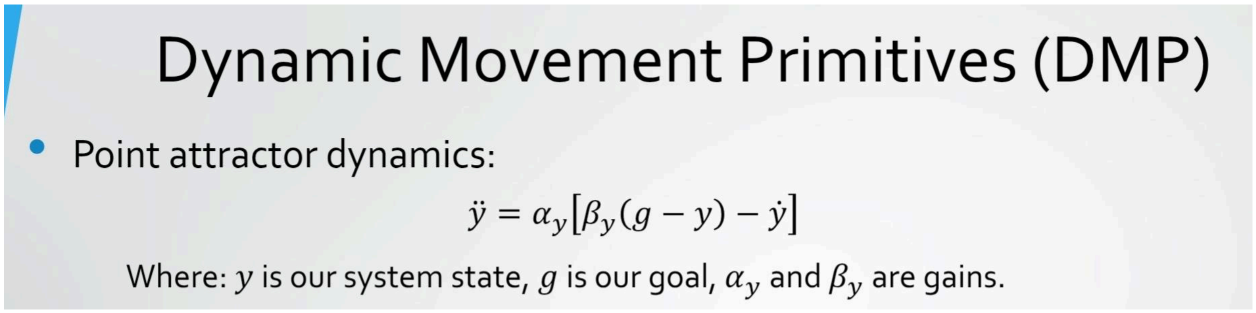
3 & 4. Time response plots for both Sin and Cos inputs:



```
Text(0.5, 1.0, 'Periodic DMP (Cos Input)')
```



Mode = 2. Alpha = 8, Beta = 2. DMP Equation given below for initial parameters.

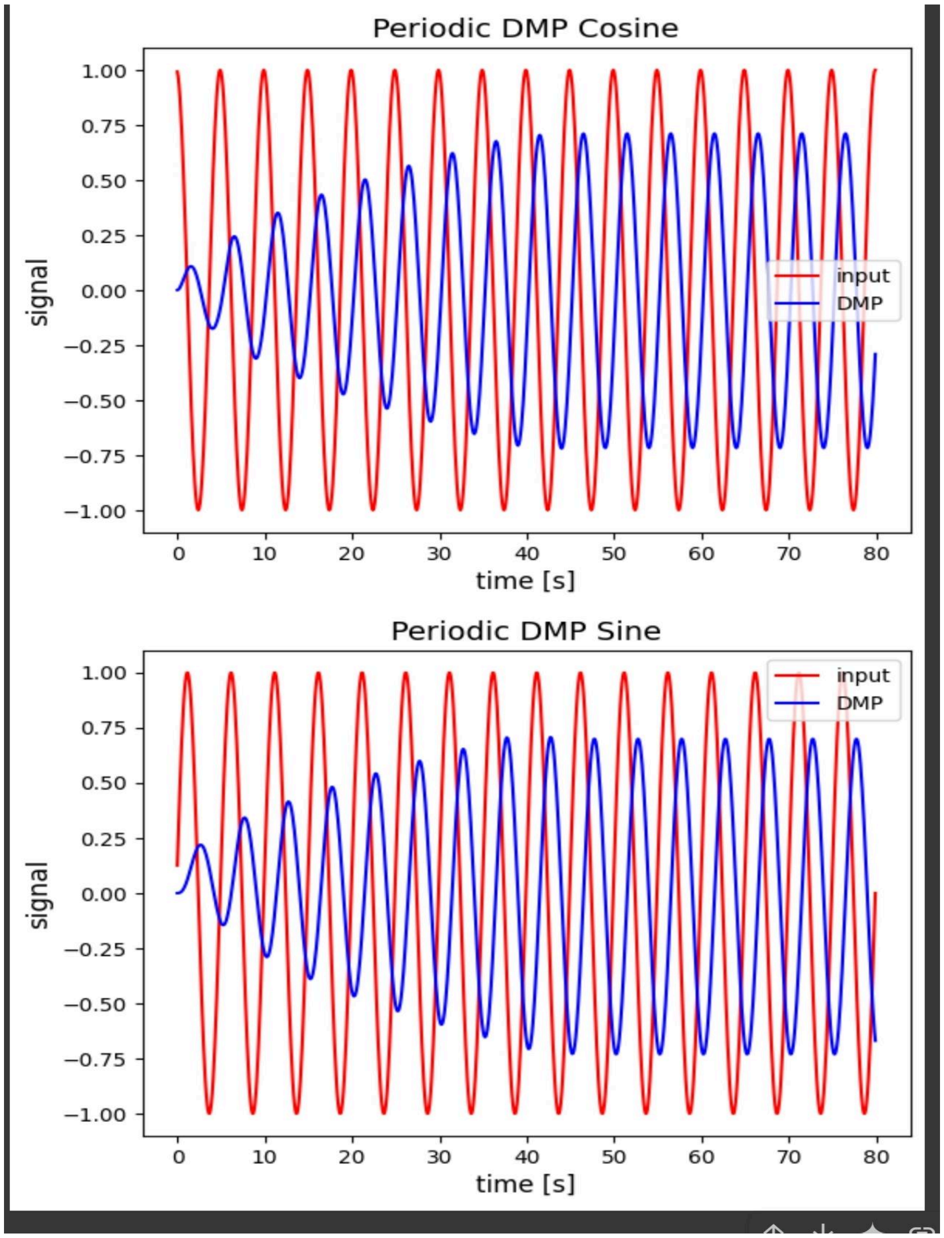


Dynamic Movement Primitives (DMP)

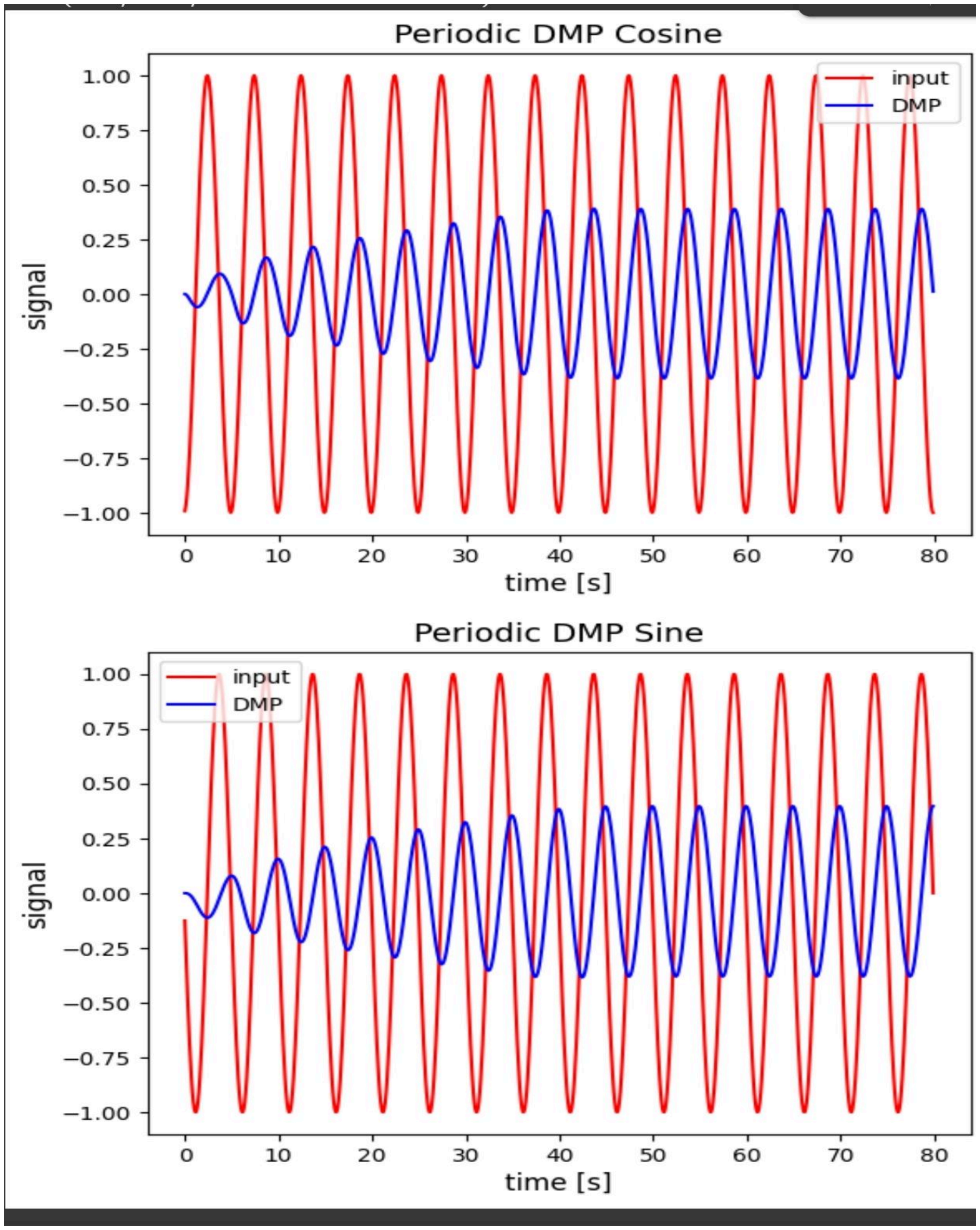
- Point attractor dynamics:

$$\ddot{y} = \alpha_y[\beta_y(g - y) - \dot{y}]$$

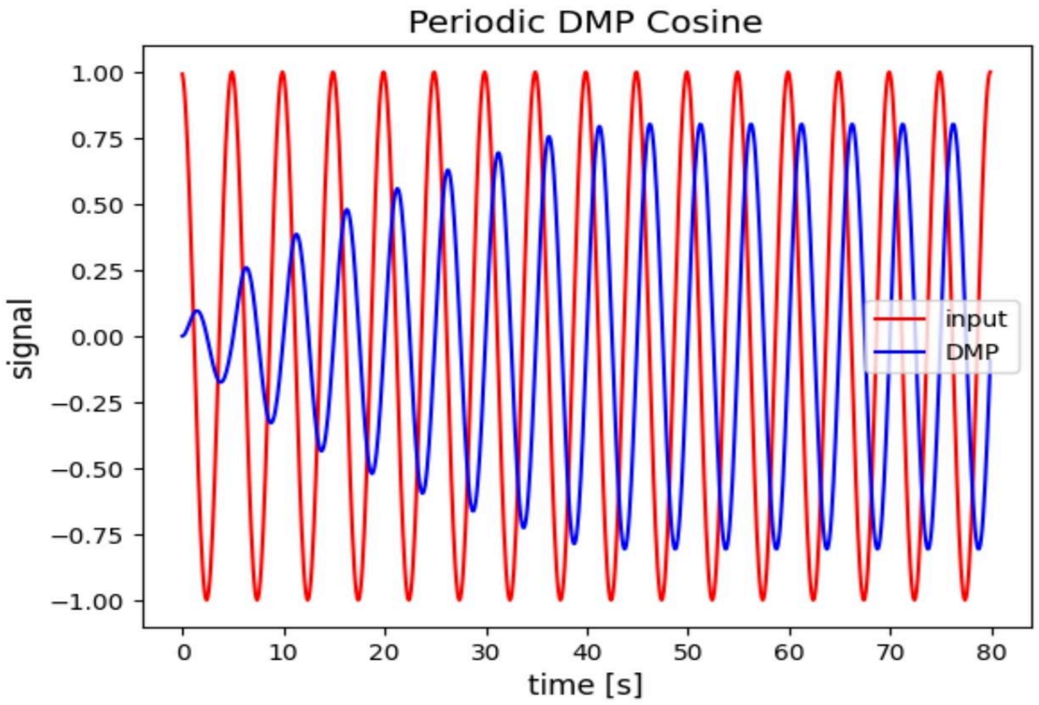
Where: y is our system state, g is our goal, α_y and β_y are gains.

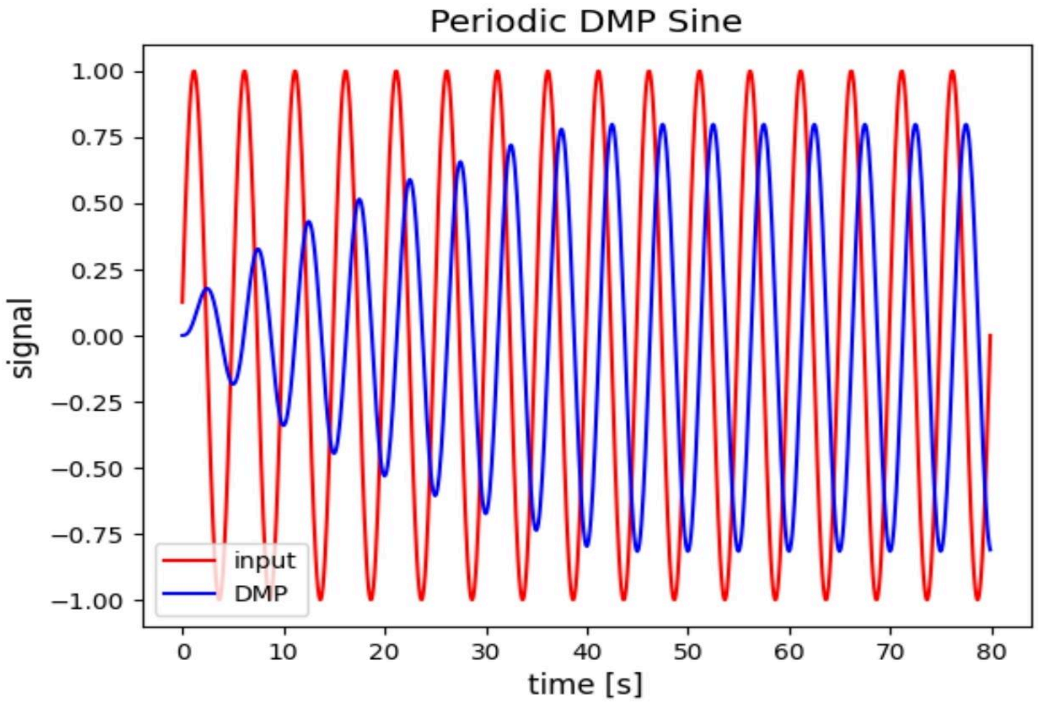


Parameter Alpha changed. Alpha = 16. Beta = 2. Mode = 2. Amplitude of the DMP wave takes longer to reach convergence, with a smaller amplitude of that at Alpha = 8 and Beta = 2. This matches the equation for the DMP.

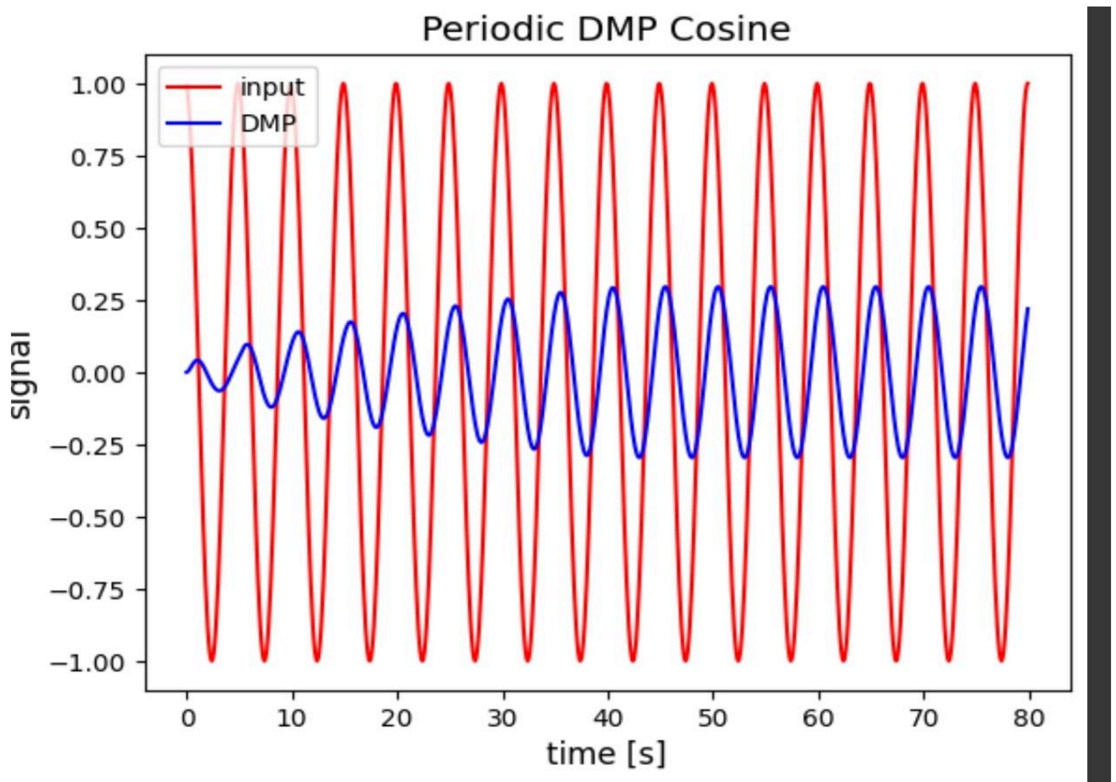


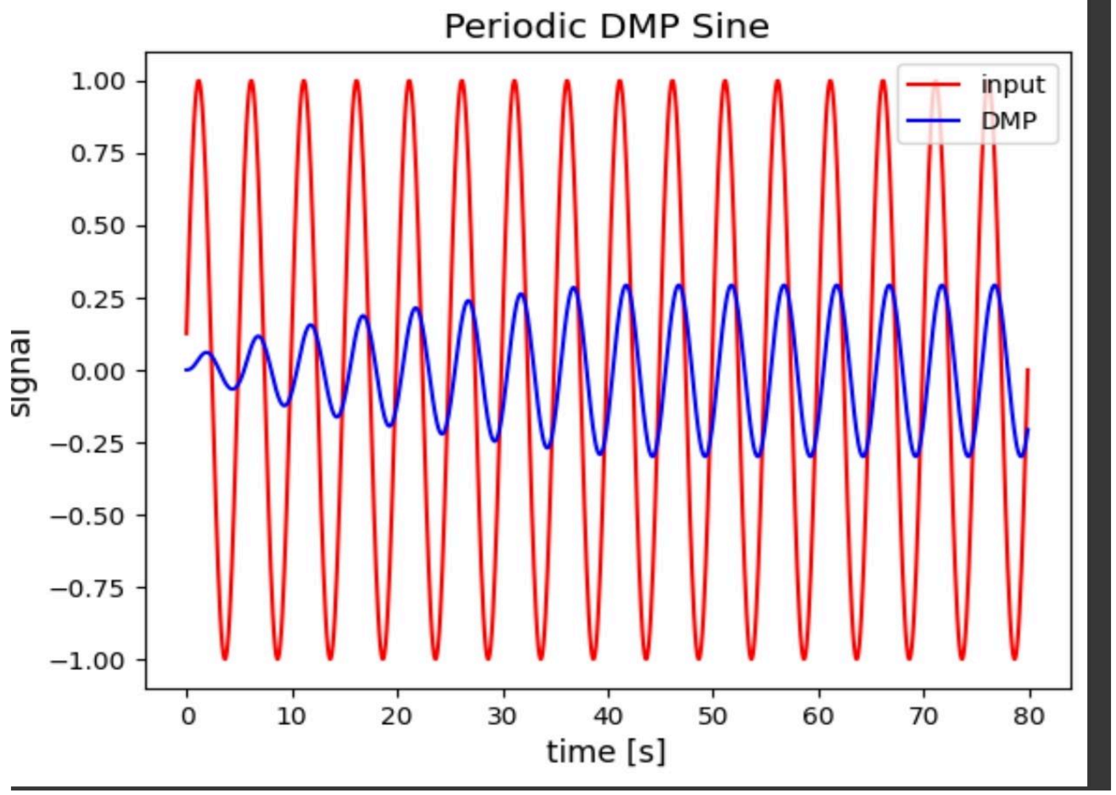
Alpha = 8, Beta = 4





Alpha = 16, Beta = 8


Periodic DMP Cosine
input
DMP
signal
time [s]


Periodic DMP Sine
input
DMP
signal
time [s]