Response to the Reviews and Decision

Title: Controlling chaos in van der Pol dynamics using signal encoded deep learning[1]

(Originally: Physics-Informed Deep Operator Control: Controlling chaos in van der Pol oscillating circuits)

Manuscript Reference Number:
mathematics-1562236

Authors:
Hanfeng Zhai
Timothy Sands

Date: January 21, 2022

[1]We would like to change the original title based on both reviewers' comments

Message from the Authors

Dear Editors and Reviewers,

We thank you for your constructive comments, which have allowed us to improve the quality of the manuscript. We have addressed the comments and incorporated your valuable suggestions in the revised manuscript, in particular highlighting the key contributions of this work. The revised manuscript are attached.

We address each comment separately in the following detailed response. The comments we received are boxed, and our responses are written following each comment. All page and reference numbers in our response are based on the revised manuscript, unless otherwise stated. The page and reference numbers mentioned in the reviewers' comments are kept intact and are based on the original manuscript. The references that we used to create our review responses are listed in the reference section in the last page of this response document. We look forward to hearing from you and hope that you find the revised manuscript satisfactory. Specifically, inspired from both reviewers' comments, we renamed the title as "***Controlling chaos in van der Pol dynamics using signal encoded deep learning***" for a more straightforward and accurate description of our work.

Sincerely,
Hanfeng Zhai, Timothy Sands

Response To Reviewer #1

Overall Comments

> In this paper, a new controller for the chaotic van der Pol system is proposed. The results are interesting; however, there are some comments as follows:

Response

We appreciate your careful review and detailed feedback. Our focus in the revised manuscript was to clearly state the novelties and contributions. We hope that you find the following response satisfactory.

Reviewer Comment

> More discussions on the application of PINN-based methods for chaotic and nonlinear systems are needed in the introduction.

Response

Thank you for your valuable suggestions. We agree. We expand our *Introduction* section to cover more details in the revised manuscript. However, as already noted in the Introduction, until now there are still very limited works on applying PINN for implementing controls and we are trying our best to cover all the related works.

- We specifically include more details on the framework of PINC[2], which is strongly related to our works.
- We include more PINN-based controls for clearer introductions.

Reviewer Comment

> The authors claim that they are working on chaos, but I just can see the periodic dynamics of the van der Pol system. Can a two-dimensional system show chaotic dynamics? What conditions should be applied to the system to have chaotic dynamics?

Response

Thank you for pointing this out. We admit this is an important point to be noted. Over the years the definition of chaos has been of discussions the elicited many related explanations. More generally, chaos is defined as obeying the following points[3]:

[2] Antonelo et al., 2021, arXiv:2104.02556

[3] Boris & Katok, 2003, Cambridge University Press. DOI:10.1017/CBO9780511998188.020

- it must be sensitive to initial conditions,
- it must be topologically transitive,
- it must have dense periodic orbits.

where over the years the sensitivity on initial conditions was much more emphasized. One thing to note is that the original idea of our paper mainly arises from a follow-up to the previous work[4] as an attempt to control chaos in the same system so we adopt the same term tackling the same issue with novel attempts. Admittedly, many preferences on specific definitions of chaos exist regarding different systems and applications.

Reviewer Comment

> Can you tell that the controller is a model predictive control? You can compare your work with some important references like "Fuzzy modeling, prediction, and control of uncertain chaotic systems based on time series".

Response

Thank you for raising this important question. If our understanding is correct, the answer is yes that PIDOC can be considered as a type of MPC. Since PIDOC take the encoded signal as a **soft constraint**[5] that guide the framework to infer a **predicted output** calculated in a prediction horizon by comparing the mathematical model to the real process's output[6], the general framework can be viewed as a new type of MPC. We also greatly thank the reviewer for providing the important reference. We have accordingly added the comparison in our discussions.

Reviewer Comment

> What type of dynamics can be the desired dynamic in this control method? As I can see, in all the simulations, the desired dynamic is a cycle. Am I right?

Response

Thank you for this important question. The answer is yes and in this paper, we apply sinusoidal signals for all the controls since it is one of the most applied controls for circuits and various dynamical systems[7,8]. However, by changing the encoded desired dynamics in $MSE_{\mathcal{D}}$ (as part of $\mathcal{L}$) one can also apply square wave signal (e.g., using the `scipy.signal.square()` in `Python`) or any other desired trajectories in control sciences.

---

[4] Cooper et al., *Mathematics*, 2017, DOI:10.3390/math5040070
[5] Lu et al., 2021, arXiv:2102.04626
[6] Antonelo et al., 2021, arXiv:2104.02556
[7] `https://www.sciencedirect.com/topics/engineering/sinusoidal-signal`
[8] `https://www.allaboutcircuits.com/video-tutorials/applications-of-sinusoidal-signals/`

Reviewer Comment

More discussions on the paper results should be added to the revised paper.

Response

Thank you for the valuable suggestion. We have revised the discussions considering your suggestions on the following points:

- We include the reviewer mentioned reference in the *Introduction* for a better overview of the backgrounds.
- We further expand the discussions on the nonlinearity part as to expand the limitations of PIDOC with more details.
- We expand more possibilities on comparing PIDOC with traditional controls for future researches.

Hope you finds our revisions satisfactory.

Response To Reviewer #2

Overall Comments

> The authors proposed a novel technics for controlling chaos based on recent advances in computer analysis by exploiting neural networks. The case study of the Van der pol oscillator is considered. The technics name physics-informed neural network proposed by previous authors is modified and adapted to control the dynamics of the van der pol oscillator to a desired state. Description of the method is provided and discussion of the results are supported by figures and tables. The work sound interesting and something new can be found. However, several concerns are to be addressed before acceptance.

Response

We would like to thank you for all the valuable feedback. Your detailed comments have considerably helped with improving the clarity of the revised manuscript. The addressed concerns and questions are responded to as follows.

Reviewer Comment

> 1- The abstract is more like the conclusion. One can't retrieve the goal of the work, the materials and methods exploited, and the major results obtained. Moreover, sentences in this abstract are too long and make the author lose comprehension.

Response

Thank you for pointing this out and we agree. We have accordingly revised the abstract to make it more concise and straightforward.

Reviewer Comment

> 2- English need to be reviewed.

Response

Thank you for pointing this out and we have modified the language as much as we can for concise presentations.

Reviewer Comment

> 3- How robust is the method to external disturbances?

Given that several authors (`https://doi.org/10.1007/s11071-012-0354-x`, `https://doi.org/10.1016/S0960-0779(01)00052-2`) have successfully controlled chaos based on external feedback, how relevant is this work compared to their own?

Response

Thank you for this question and this indeed is a very important question. To answer this question we design another sets of numerical experiments, with the van der Pol systematic chaotic input data contains "disturbance" (or noise). The disturbance were exerted using the MATLAB `randn` function to create the time series data with same size as the original training data based on normal (Gaussian) distribution, can be written as:

$$\mathtt{x_{train}\_with\_noise} = \mathtt{x_{train}} * (1 + n * \mathtt{randn}(3000, 1)) \tag{1}$$

where n is the level of noise, and we change it from 0% to 10%. The new $\mathtt{x_{train}\_with\_noise}$ was then input to PIDOC. The inherent dynamics, controlled schemes, and desired trajectory are shown in Figure 1.

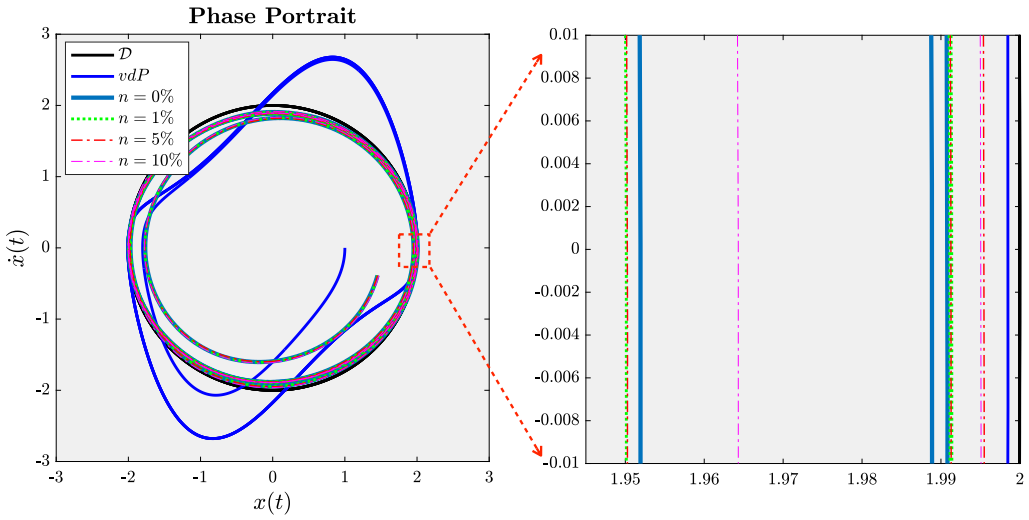

Figure 1: The PIDOC controls with different van der Pol systematic input contains different level of noise (or disturbance).

Based on the original and zoomed view in Figure 1 one can observe that with more noise added the PIDOC controlled schemes do not variate evidently, with seemingly overlapped controlled dynamics. The zoomed view in the right sub figure does show small differences given different levels of noise. We can hence deduce that PIDOC is insensitive to systematic noise, and hence say it has a generally high tolerance to disturbance.

If our understanding is correct, the work of Sun et al.[9] and Yang, Chen, & Yau[10] both adopt the method using external forced system via feedback (by adding a d term in the Yang paper and c_0 term in the Sun paper) to achieve the controlled state. In our original manuscript, we do mention the inspiration of our work by Cooper et al., who also employs forced terms in a similar way to achieve control as in Eq. (2) in our manuscript. We do provide a very brief explanation following

[9]Sun et al., 2012, *Nonlinear Dynamics*, DOI:10.1007/s11071-012-0354-x
[10]Yang, Chen, & Yau, 2002, *Chaos, Solitons & Fractals*, DOI:10.1016/S0960-0779(01)00052-2

that equation. As specified for the difference, if adopt an inappropriate analogy, the encoded signal in $\mathcal{L}$ ($MSE_{\mathcal{D}}$ & $MSE_{\mathcal{I}}$) can be considered as an "inner forced term", to guide the system to achieve the controlled state.

Reviewer Comment

> 4- The nonlinear dynamical system recently prove to experience the striking phenomenon of multistability, and such phenomenon deserves control (`https://doi.org/10.1142/S0218127419501190`). how does PINN help there?

Response

Thank you for this significant question. The multistability phenomenon has been an important issue in control sciences. If our understanding is correct, considering the deep learning approach, the multistability requires to make the systems controlled to the desired trajectory given different stabilized phases. To this extent, if the desired trajectory is still a sinusoidal signal then we believe the original PIDOC framework may still work for control. However, considering the just for NN the input is the time series and the output should be multistabilized states (rather than just $x(t)$ for our van der Pol system), then to the perspective of deep learning this should be a mapping between function(s) to function(s), i.e., functional spaces. Hence, in the fields of deep learning, the recently proposed DeepONet can solve the approximation of such mappings[11]; and correspondingly the modified version, Physics-Informed DeepONet[12], may allow us to implement controls with better accuracy compared with just a modified version of PINN for systems of multistability.

Reviewer Comment

> 5- The case study of van der pol is quite interesting since several electro-mechanical devices are operating satisfying its modelling, especially the pacemaker which controls the heartbeat rate. How long can this method take to control such a real situation? And how does it apply there (experiment approach)?

Response

Thank you for raising this question. The aim of this paper is to provide a theoretically feasible model for future control design. Also, limited by the accessible resources we are not able to apply such a design in real practice. However, we can envision the possibility of applying PIDOC to real-world engineering systems, say circuits. We can either recreate PIDOC in Simulink or directly apply our `Python` code to connect with LabVIEW and exert the control, just to replace the traditional PI or PID controls with the PIDOC framework, and connect with the circuits' wires as to tune the current flow[13]. Note that the framework has to "consume" some nonlinear data from the chaotic system first. Regarding specifically to the question on **how long** even though we do not actually

[11] Lu et al., *Nature Machine Intelligence*, 2021, DOI:10.1038/s42256-021-00302-5

[12] Wang, Wang, & Perdikaris, *Science Advances*, 2021, DOI:10.1126/sciadv.abi8605

[13] Referring to tune the capacitor in `https://services.math.duke.edu/education/ccp/materials/diffeq/vander/vand1.html`

conduct the experiments we here provide a theoretical analysis in Figure 2. Note that in our response to Q. 8 we also provide an abstract comparison of PIDOC with traditional methods.

Figure 2: Time analysis of the system control.

In our attempts, we only estimate the NN training time t_{train} whereas in real practice one needs to also take the algorithmic implementation time t_{alg} and hardware implementation time t_{hw} into account. Hence, implementing PIDOC in an experiment will no doubt take longer than we estimated.[14]

Reviewer Comment

> 6- I instead view the interest of this method in harnessing the complex dynamical behaviours in a new system or existing one by engineers in case of any unexpected behaviour in the functioning of the existing one. However, moving from theory to practice may be a challenge.

Response

Thank you for providing this interesting view. We totally agree. The actual **practice** requires much more technical analysis to overcome humongous possible hardware issues.

Reviewer Comment

> 7- Fig.3 is unclear to me since we get two outputs (one for feedback loop and another one for controlled dynamics)

Response

Thank you for pointing this out. We apologize for the implicit description in the paper. The output of the PIDOC framework should be the controlled dynamics. However, the PIDOC framework includes a neural network (or a multi-layer perceptron, MLP), in the blue box in Fig. 3 in our original manuscript. The output of MLP, x_{pred}, was first included in the loss (see our provided algorithm for Q. 9), in both $MSE_{\mathcal{D}}, MSE_{\mathcal{I}}, MSE_{NN}$. Then the whole PIDOC framework, including both MLP & physics-informed control output the controlled dynamics. A schematic was created to illustrate our framework in detail: the feedback loop including the automatic differentiation to the physics-informed loss, and the MLP are all part of the controller that output the eventually controlled dynamics.

---

[14] One thing to note that the t_{train} we computed in our manuscript considered the whole training time including NN initialization, weights randomization, etc., whereas one can also only count the time for exempting the command `model.train()`

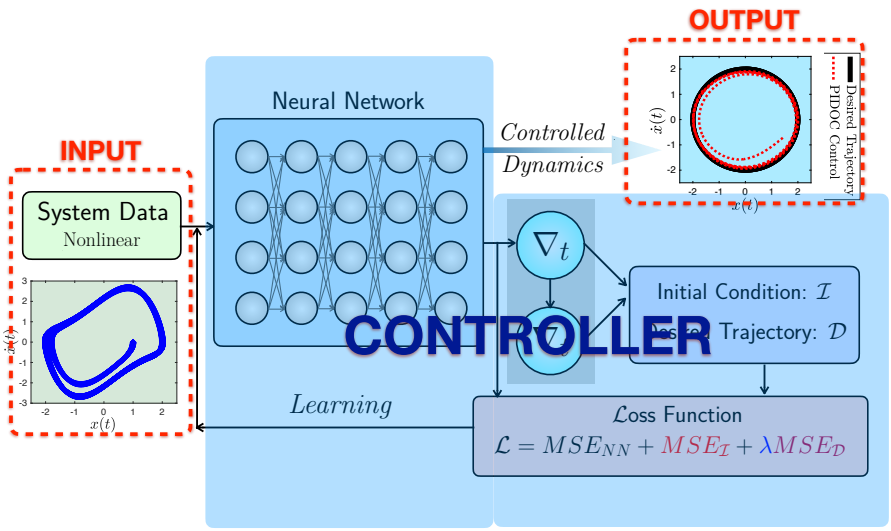

Figure 3: Explanation of the PIDOC system.

Reviewer Comment

> 8- How can Fig3 be applied to fig 1 experimentally without disturbing the dynamical behaviour of the system?

Response

Thank you for pointing out this important question. If we understand the question correctly, the term *disturbing* can be referred to as tuning the nonlinear system to behave like the desired output. Then in PIDOC, the *disturbing* action can be understood as the encoded signal (physics-informed loss) for implementing controls. However, since the aim of this paper is to provide a theoretical methodology, exact experimental implementations might be limited to the exact cases, thus are not detailed elaborated in the manuscript. To better explain our thoughts as referring to Fig. 1 (as a possible example of how to apply PIDOC) in the *Introduction*, we generate Figure 4: the van der Pol circuit generated nonlinear data (blue) was input to the generalized "controller", which might be PIDOC or traditional methods. For PIDOC: the system nonlinear behavior (represented with data in simulation) was first input to the NN then the MSE_{NN} were to output to the physics-informed loss (PI loss), $MSE_{\mathcal{D}}$ and $MSE_{\mathcal{I}}$ were also encoded to PI loss through automatic differentiation (AD). For a closed-loop feedback control system, a control signal is first executed to the plant and the nonlinear signal is fed back to control adjustment (either with PID, PI, or other methods). Note that one specific characteristic of PIDOC is there is no constant input of control signals required since it is encoded in the physics-informed loss.

Reviewer Comment

> 9- Implemented algorithm may be a huge contribution to this work (although provided by the author under acceptance of the manuscript). At least provide a flowchart describing the implemented algorithm for now.

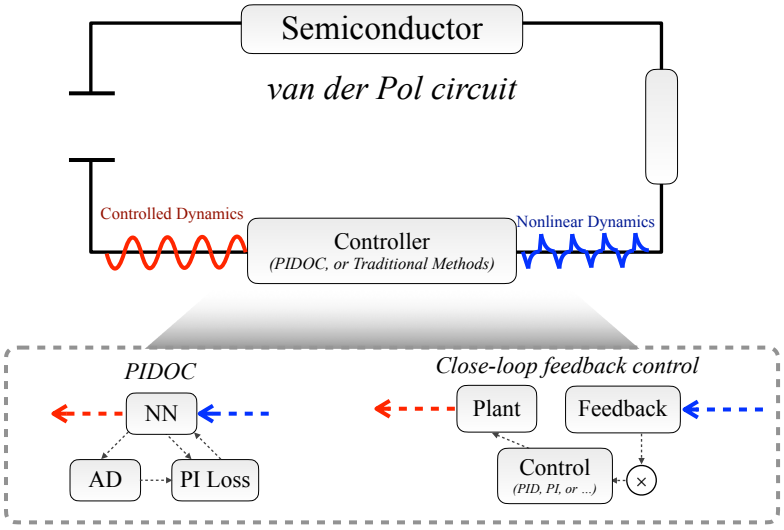

Figure 4: A simplified schematic for explaining how PIDOC is applied to the van der Pol circuits.

Response

Thank you for this important point. We totally agree. We hence provide a simplified process of our algorithm listed as follows in Algorithm 1. We also attached our `Python` code at the end of our response letter.

Reviewer Comment

> 10- From fig.9 and this also holds as a general observation, the desired trajectory does not match the inherent one. How can this be explained? This becomes complicated when the nonlinearity is strong (large mu) (see Fig.7)

Response

Thank you for this crucial question. In our *Results and discussion* we point out that weak controllability on high nonlinear systems remains an "unsolved" characteristic of PIDOC even by increasing the neurons and layers. We report such as part of the PIDOC's weaknesses and limitations that may lead to future better design of using PINN-based controls. Currently, we provide the explanation: physics-informed control is only a soft constraint inserted, whereas when the systematic data (training input) is highly nonlinear, a soft constraint is not robust enough for implementing controls. The recent work on PINNs with hard constraints (hPINNs)[15] may be a potential tool for implementing controls with PINN.

Reviewer Comment

[15] Lu et al., 2021, arXiv:2102.04626

Algorithm 1 PHYSICS-INFORMED DEEP OPERATOR CONTROL

class PIDOC($self,\ t,\ x,\ layers$)
    $\hat{t} = \text{UPDATE}(t)$
    $(\hat{weights},\ \hat{biases},\ \hat{layers}) = self.\text{INITIALIZENN}(weights,\ biases,\ layers)$
    $\{x_{pred},\ \dot{x}_{pred},\ \ddot{x}_{pred}\} = self.\text{NEURALNET}(t)$
    $x_{control} = \sin(t) \longrightarrow \dot{x}_{control} = \cos(t), \ddot{x}_{control} = -\sin(t)$;
    $MSE_{\mathcal{D}} = \texttt{MSE}([\ddot{x}_{\mathcal{D}}, \ddot{x}_{pred}], [x_{\mathcal{D}}, x_{pred}])$*#Eq. (9) in our manuscript*
    $MSE_{\mathcal{I}} = \texttt{MSE}(x_{pred}[0],\ x_{control}[0])$
    $MSE_{NN} = \texttt{MSE}(x_{pred},\ x_{train})$
    $self.\text{Loss} = MSE_{NN} + MSE_{\mathcal{I}} + \lambda MSE_{D}$ *#adjust the Lagrangian multiplier*
    Specify Optimizer: 'L-BFGS-B'
    def INITIALIZENN($self,\ layers$)
        Initialize all the *weights* & *biases* for NEURALNET.
    def NEURALNET($self,\ weights,\ biases$)
        Build NN for mapping: $t \to x$ with random *weights* & *biases*.
        $\{x,\ \dot{x},\ \ddot{x}\} = self.\text{NEURALNET}(t,\ weights,\ biases)$
    #gradients calculated by recalling `tf.gradients`
        Return $\{x,\ \dot{x},\ \ddot{x}\}$
    def TRAIN($self,\ iterations$)
        Define the dictionary for storing variable x and t during iterations.
        Recall the optimizer to minimize the loss and train the NN model.
    def PREDICT $\{x\}$ ($self,\ iterations$)
        $\{x_{pred}\} = self.$ `sess`.run(t)
Input = $\{t\}$, Output = $\{x\}$
Specify the model hidden layers: `Layers` = [Neurons × Layers] *#specify the hyper-parameters*
Load fields data of the nonlinear van der Pol system, $\{t_{train},\ x_{train},\ \dot{x}_{train},\ \ddot{x}_{train}\}$, as `Data`. *#considered as the controller taking the chaotic input*
`model` = PIDOC(`Data`, `Layers`)
`model`.TRAIN(Iterations) *#Specify the training iterations*
Obtain $\{x_{pred}\}$ = `model.`PREDICT(t). *#obtain the predicted* **controlled** *output* $\longrightarrow$ *considered as the PIDOC controlled scheme*
Save all the data & post-processing.

11- The method applied here seems to be specific to low dimension systems. Moreover, the choice of inherent parameters of the NN was not well-motivated, given that this can vary from one system to another.

Response

Thank you for the important comment. We agree. We did not include applications of PIDOC (or different modifications of PINNs) on controlling high dimensional systems, regarding the limited volume of the manuscript. However, we do believe that by reformulating PINNs there is a possibility of controlling high-dimensional systems such as the Lorenz System, which could be included in our future directions. We admit that in our approach the selection of the hyperparameters was mostly empirical. Fine-tuning the hyperparameters using grid search or bayesian optimization may be considered as future research directions[16]. Here, we chose the NN structure as 6×30 initially considering the size of the training data (or the chaotic data generated from the van der Pol system). In our problem there are only 3000 data points for $x, \dot{x}, \ddot{x}$ and t respectively. Hence, empirically, a small FNN with layers < 10 and neurons < 100 are generally sufficient to fit the model. We have added the related parts into our manuscript correspondingly. The tuning of the NN hyper-parameters was also based on such a criterion to check how the model behaves given different NN sizes.

[16] Liu and Wang, 2021, arXiv:2106.09204

```
# !pip uninstall tensorflow==2.7.0
# !pip install tensorflow==1.15.0
import tensorflow as tf
# import deepxde as dde
import numpy as np
from scipy import linspace
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import scipy.io
import time
import timeit
# import tensorflow_probability as tfp

# tf.disable_v2_behavior()
print(tf.__version__)

class DeepvdP:
    # Initialize the class
    def __init__(self, x, t, layers):

        self.lb = t.min(0)
        self.ub = t.max(0)

        self.x = x
        self.t = t

        self.layers = layers

        # Initialize NN
        self.weights, self.biases = self.initialize_NN(layers)

        # tf placeholders and graph
        self.sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True,
                                                     log_device_placement=True))

        self.t_tf = tf.placeholder(tf.float32, shape=[None, self.t.shape[1]])
        self.x_tf = tf.placeholder(tf.float32, shape=[None, self.x.shape[1]])

        self.x_control = 5 * tf.math.sin(self.t_tf)#5.00 *
        self.x_dd_control = - 5 * tf.math.sin(self.t_tf) #5.00 *

        self.x_pred, self.x_dd_pred, self.ICs = self.vdP(self.t_tf)

        self.x_res = self.x_control - self.x_pred
        self.x_dd_res = self.x_dd_control - self.x_dd_pred

        self.control = self.x_res + self.x_dd_res
```

```
        self.loss = 1 * tf.reduce_sum(tf.square(self.control)) + tf.reduce_sum(tf.s
        tf.reduce_sum(tf.square( self.x_tf - (self.x_pred * 2 / 5 ) )) #+ \* 2 | *


        self.optimizer = tf.contrib.opt.ScipyOptimizerInterface(self.loss,

        method = 'L-BFGS-B', options = {'maxiter': 200000,

        'maxfun': 200000,

        'maxcor': 50,

        'maxls': 50,

        'ftol' : 1.0 * np.finfo(float).eps})


        self.optimizer_Adam = tf.train.AdamOptimizer()
        self.train_op_Adam = self.optimizer_Adam.minimize(self.loss)

        init = tf.global_variables_initializer()
        self.sess.run(init)
# ==============================================================================
    def initialize_NN(self, layers):
        weights = []
        biases = []
        num_layers = len(layers)
        for l in range(0,num_layers-1):
            W = self.xavier_init(size=[layers[l], layers[l+1]])
            b = tf.Variable(tf.zeros([1,layers[l+1]], dtype=tf.float32), dtype=tf.f
            weights.append(W)
            biases.append(b)
        return weights, biases

    def xavier_init(self, size):
        in_dim = size[0]
        out_dim = size[1]
        xavier_stddev = np.sqrt(2/(in_dim + out_dim))
        return tf.Variable(tf.truncated_normal([in_dim, out_dim], stddev=xavier_std

    def neural_net(self, t, weights, biases):
        num_layers = len(weights) + 1

        H = 2.0*(t - self.lb)/(self.ub - self.lb) - 1.0
        for l in range(0,num_layers-2):
            W = weights[l]
            b = biases[l]
            H = tf.tanh(tf.add(tf.matmul(H, W), b))
```

```
        W = weights[-1]
        b = biases[-1]
        Y = tf.add(tf.matmul(H, W), b)
        return Y
# ==============================================================================
    def vdP(self, t):
      x = self.neural_net(tf.concat([t],1), self.weights, self.biases)
      # dx_t = dde.grad.jacobian(x, t, i=0)
      # dx_tt = dde.grad.hessian(x, t, i=0)
      dx_t = tf.compat.v1.gradients(x, t)
      dx_tt = tf.compat.v1.gradients(dx_t, t)
      # x_desire = tf.math.sin(t) #5 *
      # x_dot_desire = tf.math.cos(t) #5 *
      # x_ddot_desire = - tf.math.sin(t) #-5 *
      # control = (x_desire - x) + (x_ddot_desire - dx_tt)
      ICs = x[0] - 1# Initial condition: 1, 5, 10
      return x, dx_tt, ICs
# ==============================================================================
    def callback(self, loss): #, betta
        print('%.3e' % (loss)) #, betta B: %.5f
        return loss

    def train(self, nIter):

        tf_dict = {self.x_tf: self.x, self.t_tf: self.t}

        # var_loss = tf.Variable(tf_dict)
        # loss = lambda: (var_loss ** 2)/2.0

        self.sess.run(self.train_op_Adam, feed_dict = tf_dict)

        self.optimizer.minimize(self.sess,
                                feed_dict = tf_dict,
                                fetches = [self.loss],
                                loss_callback = self.callback)

        # self.optimizer.minimize(self.loss)
        # self.optimizer.minimize(self.loss, global_step=None, var_list=var_loss,
    # aggregation_method=None, colocate_gradients_with_ops=False, name=None,
    # grad_loss=None)

        # (self.sess, feed_dict = tf_dict, fetches = [self.loss], loss_callback =

# ==============================================================================
    def predict(self, t_star):

        tf_dict = {self.x_tf: self.x, self.t_tf: self.t}
        x_star = self.sess.run(self.x_pred, tf_dict)
```

```
        return x_star

if __name__ == "__main__":

    layers = [1, 30, 30, 30, 30, 30, 30, 1]#30, 30, 30, 30, \
                #  30, 30, 30, 30, 30, 30, 30, 30, 30, 30, \
                #  30, 30, 30, 30, 30, 30, 30, 30, 30, 30, \
                #  30, 30, 30, 30, 30, 30, 30, 30, 30, 30, \
                #  30, 30, 30, 30, 30, 30, 30, 30, 30, 30,
    # Set NN Structure

    data = scipy.io.loadmat('data_mu3.mat')
    t_obtain = data['t'] #
    X_obtain = data['x'] #
    t_star = t_obtain[0:5000]
    X_star = X_obtain[0:5000]
    # print(sol.t)
    # print(sol.y[0])
    N = X_star.shape[0]
    T = t_star.shape[0]
    # print(X_star)
    # print(t_star)
    # Rearrange Data
    XX = np.tile(X_star, (1,T)) # [0:3000]
    TT = np.tile(t_star, (1,N)).T #

    x = XX.flatten()[:,None] #
    t = TT.flatten()[:,None] #

    # Training Data
    idx = np.random.choice(N*T, int(N*T*0.75), replace=False)
    x_train = X_star #x[:,:] # [idx,:]
    t_train = t_star #t[:,:]

    # Training
    t_tic = time.time()
    model = DeepvdP(x_train, t_train, layers)
    model.train(200000)
    # cpu_time = timeit.default_timer()
    # Prediction
    x_pred = model.predict(t_star)
    # loss_hist = model.callback()
    elapsed_toc = time.time() - t_tic

    np.savetxt("prediction.txt", np.hstack(x_pred))
    print(elapsed_toc)
```