Cambridge University Engineering Department
Engineering Tripos Part IIA
PROJECTS: Interim and Final Report Coversheet

IIA Projects

TO BE COMPLETED BY THE STUDENT(S)

Project:	SF3: Machine Learning		
Title of report:	Dynamical Simulation Report Individual Report		
Name(s): (capitals)		crsID(s):	College(s):
XIAOQIAO HU		xh297	Trinity College

Declaration for: Interim Report

I confirm that, except where indicated, the work contained in this report is my own original work.

Instructions to markers of Part IIA project reports:

Grading scheme

Grade	A*	A	B	C	D	E
Standard	Excellent	Very Good	Good	Acceptable	Minimum acceptable for Honours	Below Honours

Grade the reports by ticking the appropriate guideline assessment box below, and provide feedback against as many of the criteria as are applicable (or add your own). Feedback is particularly important for work graded C-E. Students should be aware that different projects and reports will require different characteristics.

Penalties for lateness: Interim Reports: 3 marks per weekday; Final Reports: 0 marks awarded – late reports not accepted.

Guideline assessment (tick one box)

A*/A	**A/B**	**B/C**	**C/D**	**D/E**

Marker:		Date:	

Delete (1) or (2) as appropriate (for marking in hard copy – different arrangements apply for feedback on Moodle):

(1) Feedback from the marker is provided on the report itself.

(2) Feedback from the marker is provided on second page of cover sheet.

	Typical Criteria	Feedback comments
Project Skills, Initiative, Originality	Appreciation of problem, and development of ideas	
	Competence in planning and record-keeping	
	Practical skill, theoretical work, programming	
	Evidence of originality, innovation, wider reading (with full referencing), or additional research	
	Initiative, and level of supervision required	
Report	Overall planning and layout, within set page limit	
	Clarity of introductory overview and conclusions	
	Logical account of work, clarity in discussion of main issues	
	Technical understanding, competence and accuracy	
	Quality of language, readability, full referencing of papers and other sources	
	Clarity of figures, graphs and tables, with captions and full referencing in text	

SF3 Machine Learning Interim Report
Dynamical Simulation

Xiaoqiao Hu
xh297

I. Introduction

In the first week of this project, we are tasked to do a dynamical simulation on an inverted pendulum system. Our focus is on the time evolution of the system state (denoted by $X = [x, \dot{x}, \theta, \dot{\theta}]$), which includes cart location, cart velocity, pole angle, and pole angular velocity. The tasks include observing and simulating the system state update using the Euler method, implementing changes of state, applying linear regression, and evaluating the model. As an outcome of the first week, a linear model is developed and then compared with the true dynamics in the time evolution. All codes used can be found in the Appendix.

II. Tasks

1. Rollout simulation

In this task, different initial conditions on cart velocity and pole angular velocity are applied to the cartpole system, giving different behaviours.

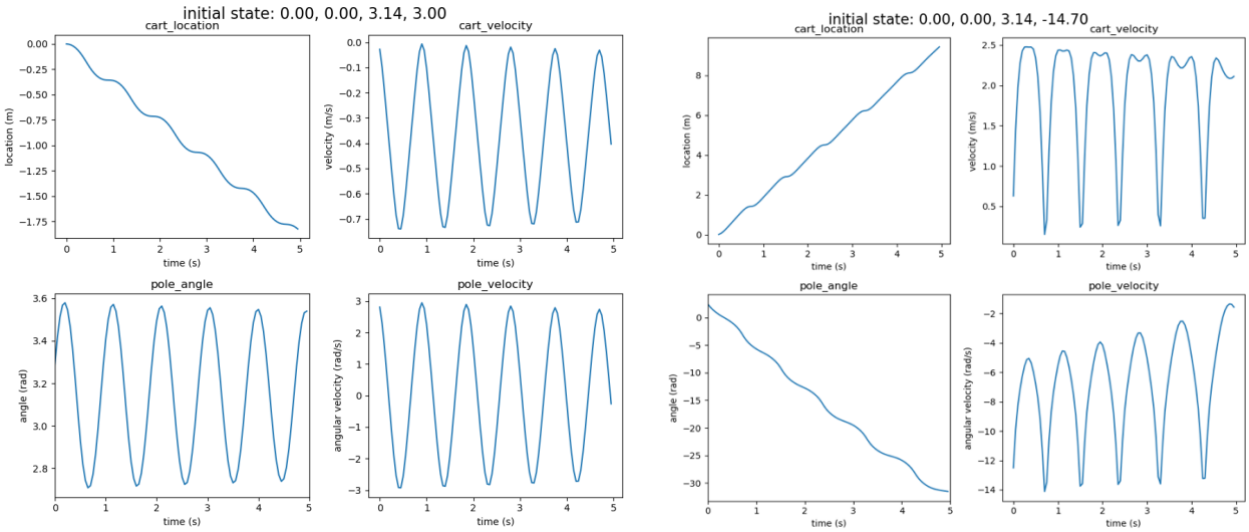

Fig 1. Time evolution of system variables
oscillation around stable equilibrium (left) and complete circular motion (right)

- Simple oscillation around the stable equilibrium given an initial state $X = [0, 0, \pi, 3]$, the pole angle value is oscillating around 3.14. Sinusoidal shape can be observed.
- Complete rotation of the pendulum given an initial state $X = [0, 0, \pi, -14.7]$, the pole angle (without remapping) monotonously decreasing as time evolves, indicating complete circular motion of the pendulum.

2. Changes of state

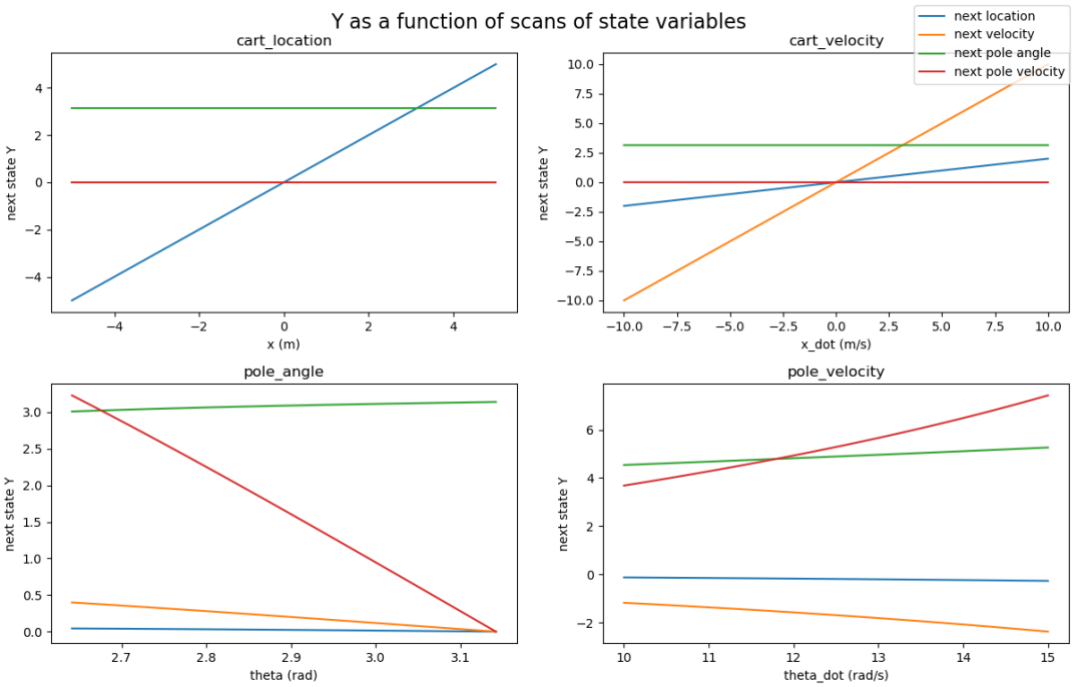

Fig 2. Next state (Y) as a function of scans of state variables

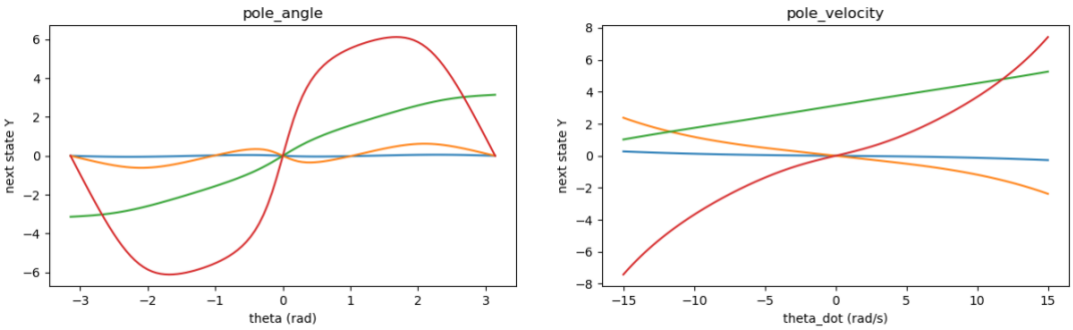

Fig 3. Next state (Y) as a function of scans of θ and $\dot{\theta}$ in a wider range

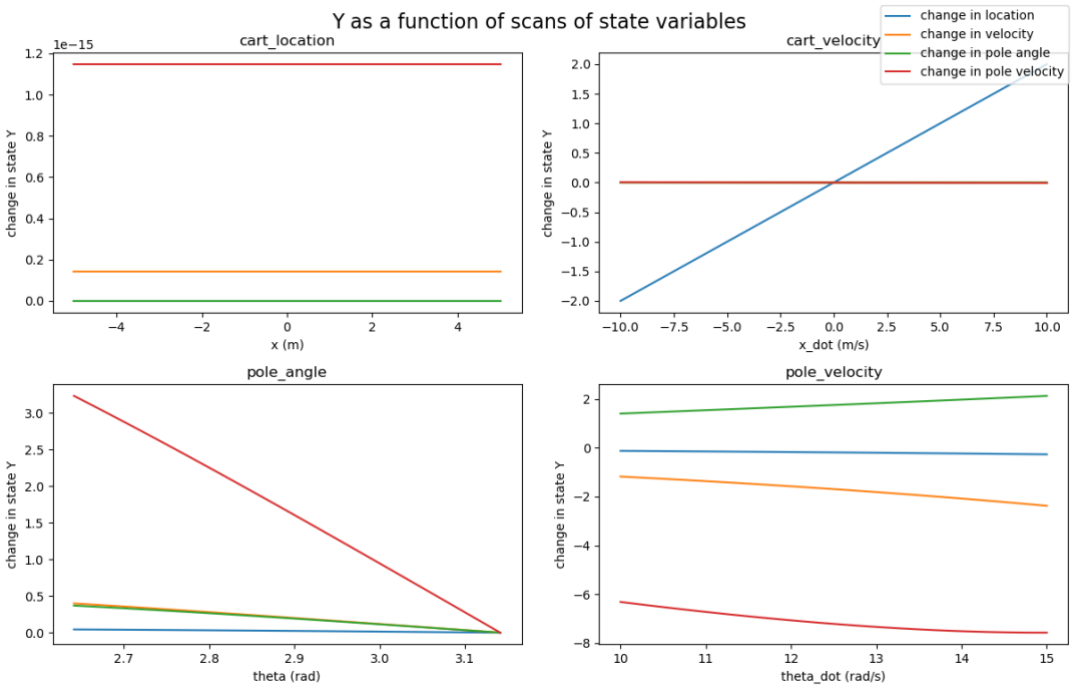

Fig 4. Change in state (Y) as a function of scans of sate variables

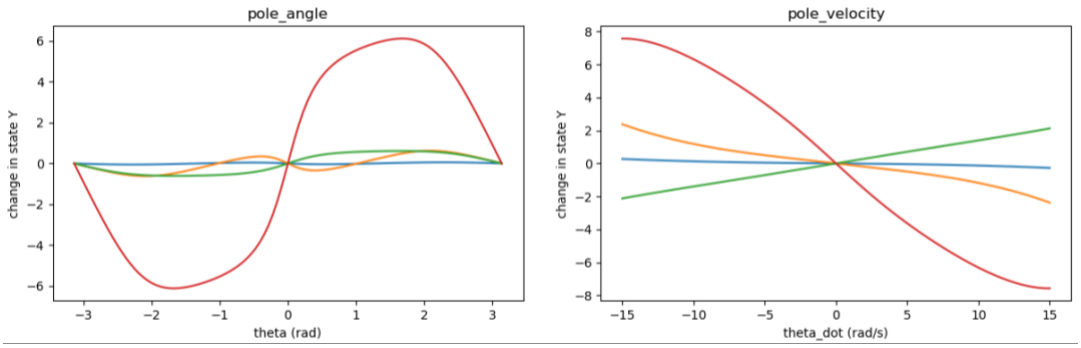

Fig 5. Change in state (Y) as a function of scans of θ and $\dot{\theta}$ in a wider range

2.1 Y modelled as next state

Given a random state X, let Y be the system state after one single call to performAction function (i.e. evolving for a time step) with zero force applied. As can be observed from Fig 2, the relationship between X and Y can be concluded as nearly linear as the change in one step is small. It is notable that for the cart location and cart velocity, the relationship stays nearly linear irrespective of the range of the variables on the horizontal axis when other variables are held fixed. For pole angle and pole velocity, where they have a generally nonlinear relationship with the next state variables, when the range of values on the x-axis is small, linearity can be assumed. And it can be observed from Fig 3 that, in a larger scale of scans, Y does not show valid linearity (especially for next velocity and next pole velocity).

2.2 Y modelled as change in state variables

We can now consider modelling Y as the change in the system state after one step: $Y \equiv X(T) - X(0)$, with T corresponding to one single call of performAction. In this case, the plots of Y as function of scans of system variables are as in Fig 4.

In this setting, the linear relationship remains as expected. Again, the linearity will not hold for some state variables on a wider scale in the pole angle and angular velocity (shown in Fig 5). In addition, from the horizontal lines in the left upper corner, it can be concluded that the cart location has no impact on the change in state variables. This can also be validated using contour plots including cart location as one of the variables (shown in Fig 6 below), where it shows horizontal stripes when cart location is on the horizontal axis.

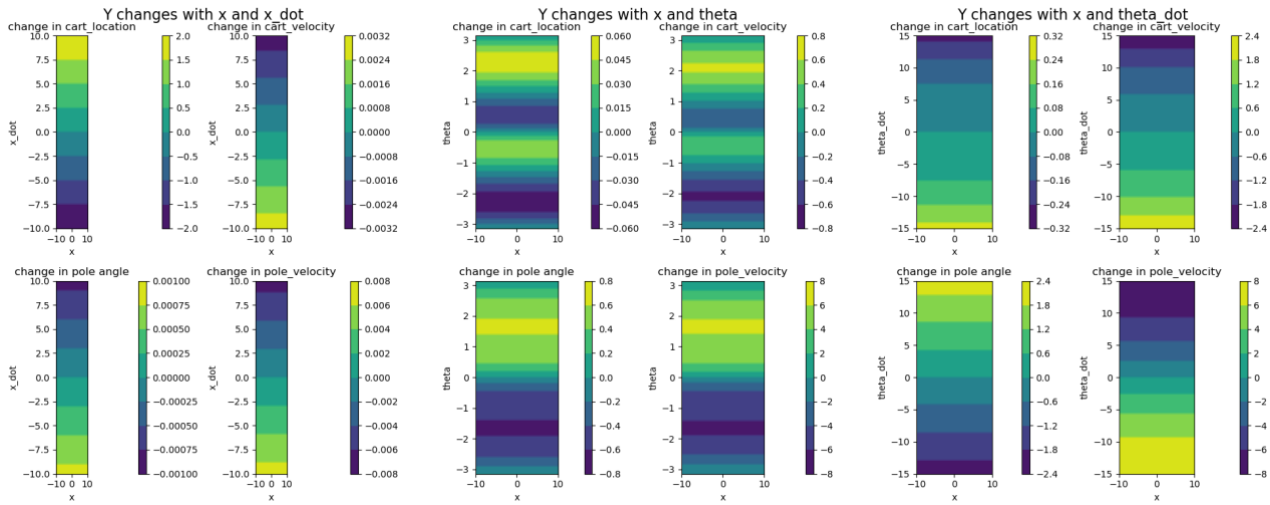

Fig 6. Contour plots of change in state Y with cart location as one changing variable (left: Y changes with x and x_dot, middle: Y changes with x and θ, right: Y changes with x and $\dot{\theta}$)

It is also notable that when cart velocity is one of the variables, its impact will only reflect on the change in cart location, having limited influence on the system. This is shown in Fig 7 below.

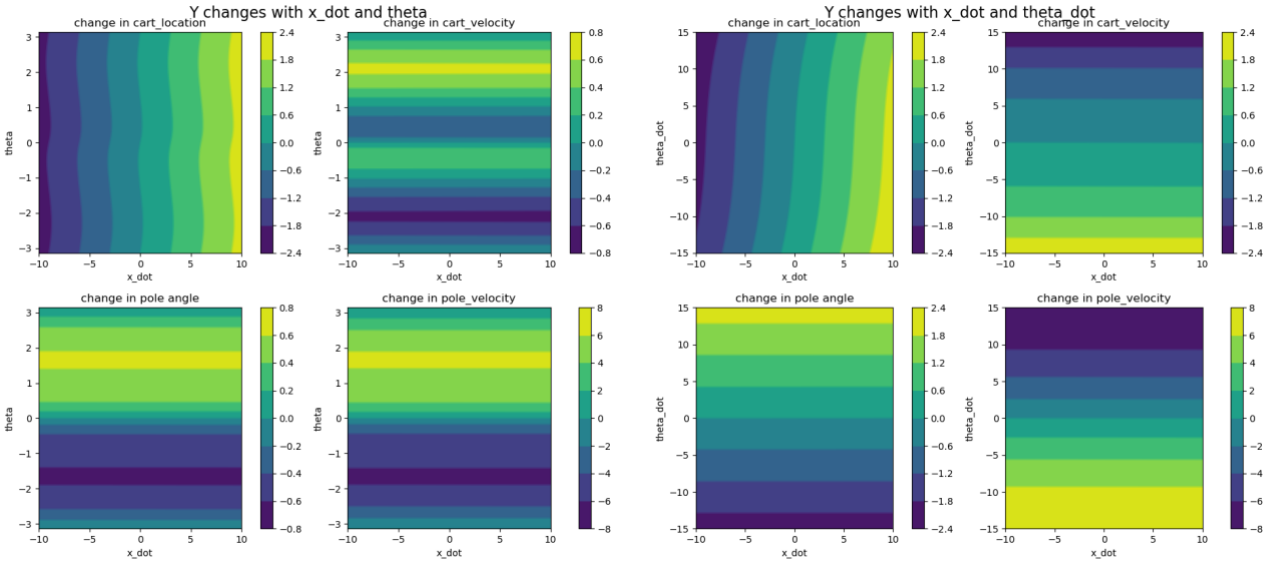

Fig 7. Contour plots of change in state Y with cart velocity as one changing variable (left: Y changes with $\dot{x}$ and θ, right: Y changes with $\dot{x}$ and $\dot{\theta}$)

In the case of pole angle and angular velocity, relatively complicated behaviour of the system can be observed. And this is also consistent with the nonlinearity between Y and these two variables in wider scales shown in previous graphs. In smaller ranges, for example $[2.54, 3.14]$ for pole angle and [10,15] for angular velocity, we can see that the stripes are nearly equally spaced, indicating the change in state can be seen as linear giving a small changing range in the state variables. The contours are shown below in Fig 8.

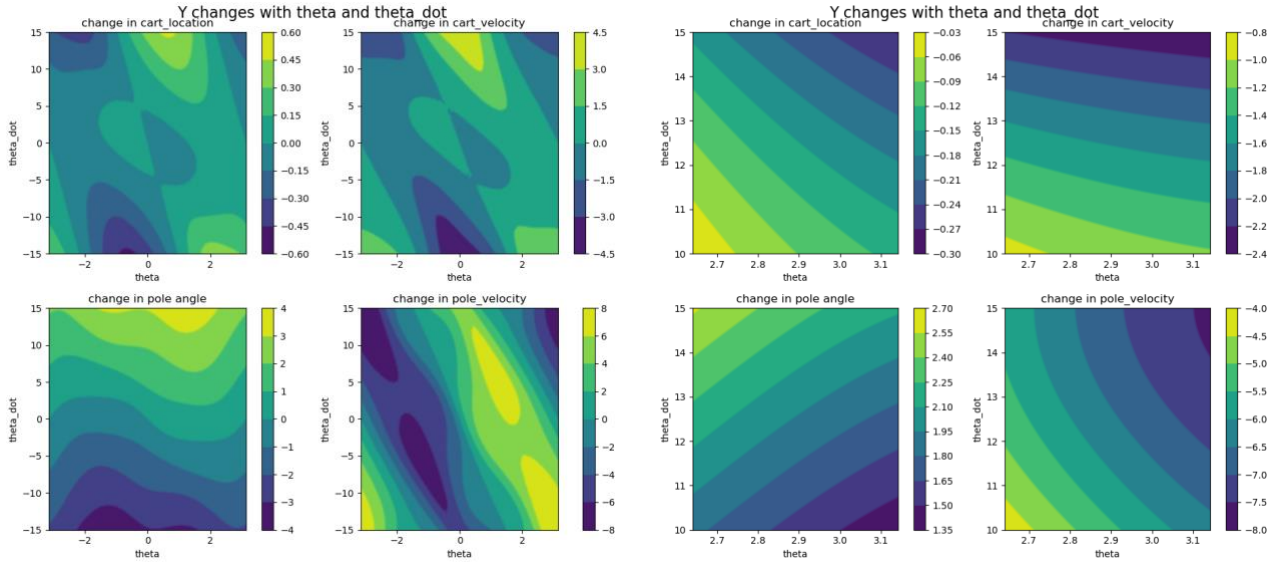

Fig 8. Contour plots of change in state Y with pole angle and pole angular velocity

From the above plots and contours we can conclude that the value in Y does not change with the cart location (denoted as x in the plots).

3. Linear regression

In this task, we set $X \equiv X(n)$, and $Y \equiv X(n+1) - X(n)$. We want to find the coefficients for the linear relationship $Y = f(X) = CX$. 1000 datapoints of X are generated from a uniform distribution in

the appropriate range of the variables and corresponding Y using the relation defined. Linear regression is applied on the dataset and an optimal coefficient matrix is found to be:

$$C = \begin{bmatrix} 0.00229529 & 0.09880289 & 0.0121259 & 0.2989933 \\ 0.20218987 & 0.03619563 & 0.01620113 & 0.18313903 \\ -0.01409282 & -0.40889898 & 0.06512129 & -0.23591583 \\ 0.01132967 & 0.08200457 & 0.20283474 & -0.02881604 \end{bmatrix}^T$$

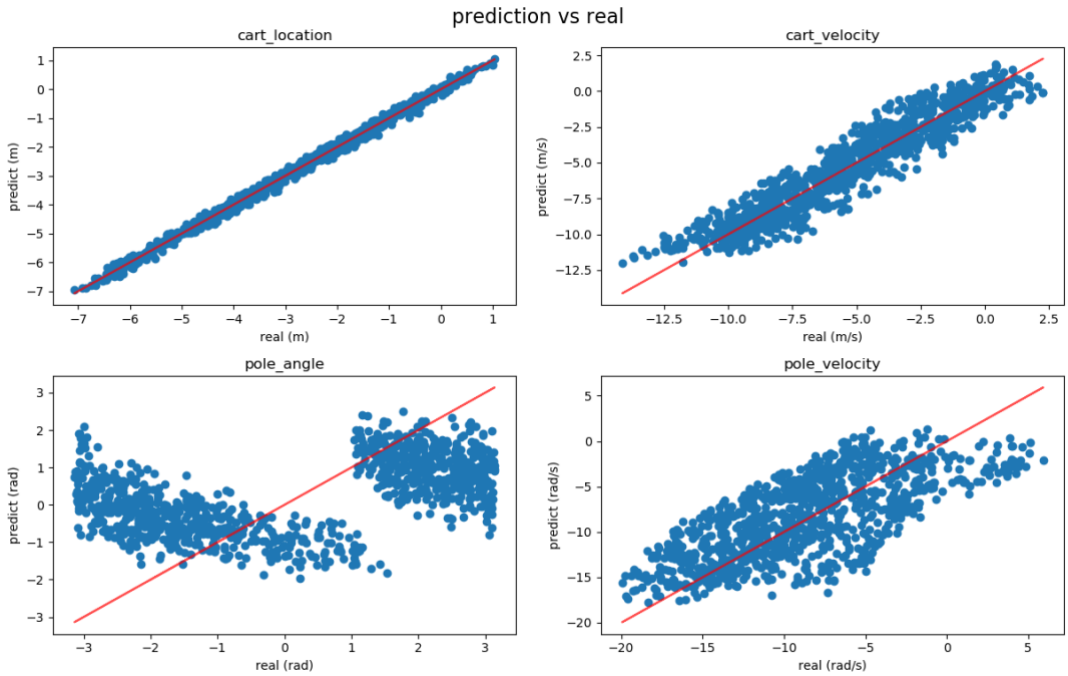

Fig 9. Linear model prediction in change of state vs real data (with the reference line in red)

To compare the model prediction result with the target, the scatter plots (model prediction as y value and real as x value) in Fig 9 are generated as above. The prediction error can be visualised as the distance of a point to the straight line in the y direction. From the plot we can see that, for the cart location, the points are centred on a straight line, indicating that the model gives good results within the limit of deviation. For the cart velocity, the points are scattered around the straight line, with relatively large errors. For pole angular velocity, the points are showing large deviations from the straight line, indicating that the model cannot give a desirable prediction. For pole angle, the points are centred in two clusters, showing poor performance in predicting.

To further elaborate, the scans of variables are used again to illustrate the model quality, as shown in Fig 10 below:

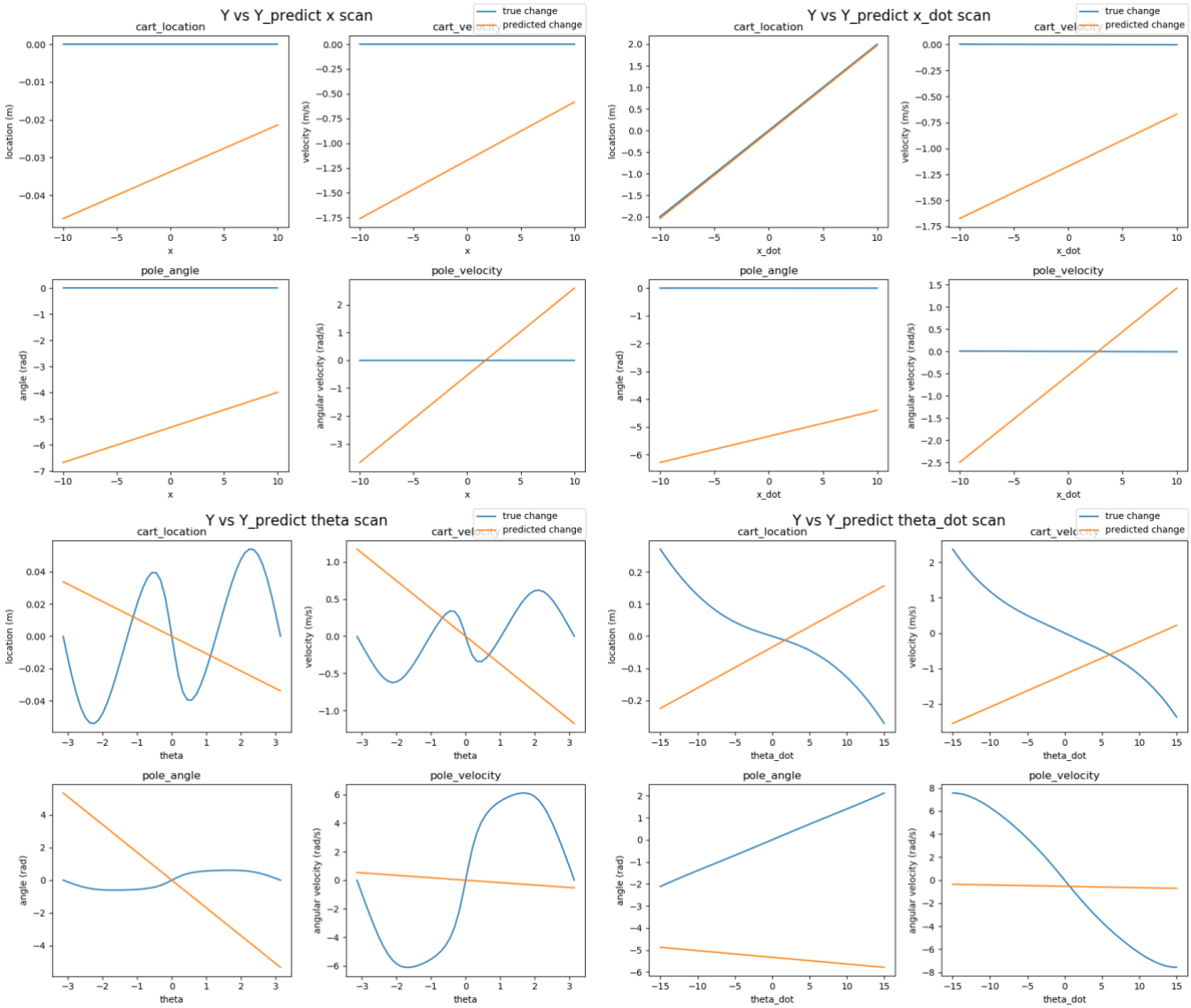

Fig 10. True value vs predicted value in change in state, varying with scans of state variables

As can be concluded from the plots above, the linear model generally will not work and will lead to enormous divergence from the real dynamics. The prediction is accurate when scanning through the cart velocity and giving the predicted change in cart location. And all the other plots show a poor fitting. This is expected since the model is trained using data generated from across the whole available range of the variables, where the data itself doesn't show linearity in general. Although it is discussed earlier that for the cart location and cart velocity, the linearity of Y with the two variables will hold on both wide and narrow scales of ranges, the coefficients generated in linear regression minimise the overall sum of squared error (including the ones with angle and angular velocity), which will lead to large deviations in the predictions for cart location and cart velocity as well.

4. Modelling time evolution

In this task, we used the model to simulate the time evolution of the system and compared it with the true dynamics. The 2 initial conditions as described in section 1 are used. The plots are generated in Fig11 as follows:

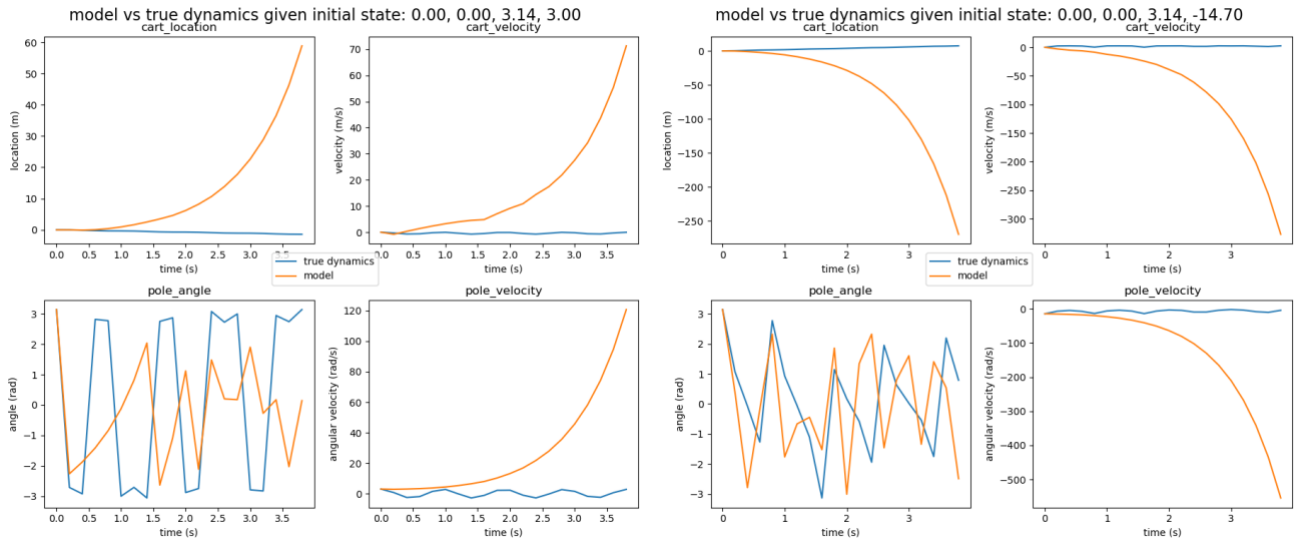

Fig 11. Time evolution modelling vs true dynamics of the system
Left: initial state of oscillation around equilibrium
Right: complete circular motion

Since simple linear model cannot deal with the periodicity in trigonometry, overall divergence shown above does not appear as surprising. In the first several time instances, the model results show relatively small deviations from the target data, and then it diverges very fast. Without remapping, the model solution with the pole angle will diverge. This is because when we fit the linear model to the data, we assume the relationship between the pole angle and Y is linear and non-periodic (Y being the change in the state variables after a single time step). In that case, the value of the pole angle in the time evolution will keep accumulating since the value of Y is adding up to get the system state at a given time instance.

The poor time resolution in the third figures is partly due to remapping. The time resolution overall can be improved by increasing the number of Euler integration steps.

III. Conclusion

In the first part of this project, we observed and analysed the local linearity of the inverted pendulum system when there is a small deviation from a given state. However, it is not adequate to apply a simple linear model to the wider scans of the variables since linearity only holds in small segments. And when simulating the dynamics, in the time evolution, the error will accumulate over the process, which leads to a large divergence of the model output, as can be observed. It can be concluded that the linear regression model, in general, cannot work well for an inverted pendulum system. To improve the results, nonlinear modelling is required, which will be explored in the next stage.

IV. Appendix

All codes used for the dynamical simulation and plotting are included, also available at Deepnote.

```
from CartPole import *
import numpy as np
import matplotlib.pyplot as plt

"""task1.1 rollout simulation"""
cartpole1 = CartPole()
cartpole1.delta_time = 0.05
max_t = 5.0 # terminate time for the simulation
n = int(max_t/cartpole1.delta_time) # number of iterations for performAction
state1 = [0, 1, np.pi, 0] # nonzero cart velocity
state2 = [0, 0, np.pi, -14.7] # nonzero angular velocity
init_state = state1
cartpole1.setState(state=init_state)
# list of variables
x, x_dot, theta, theta_dot = [],[],[],[]
t = np.arange(0,max_t,cartpole1.delta_time)
# simulation
for i in range(n):
    cartpole1.performAction()
    # cartpole1.remap_angle()
    current_state = cartpole1.getState()
    x.append(current_state[0])
    x_dot.append(current_state[1])
    theta.append(current_state[2])
    theta_dot.append(current_state[3])
# plotting the results
fig, axs = plt.subplots(2,2,figsize=(9,16),constrained_layout=True)
axs[0,0].plot(t,x)
axs[0,0].set_title('cart_location')
axs[0,0].set_xlabel('time (s)')
axs[0,0].set_ylabel('location (m)')
axs[0,1].plot(t,x_dot)
axs[0,1].set_title('cart_velocity')
axs[0,1].set_xlabel('time (s)')
axs[0,1].set_ylabel('velocity (m/s)')
axs[1,0].plot(t,theta)
axs[1,0].set_title('pole_angle')
axs[1,0].set_xlabel('time (s)')
axs[1,0].set_ylabel('angle (rad)')
axs[1,0].set_xlim([0,5.0])
axs[1,1].plot(t,theta_dot)
axs[1,1].set_title('pole_velocity')
axs[1,1].set_xlabel('time (s)')
axs[1,1].set_ylabel('angular velocity (rad/s)')
fig.suptitle('initial state: {0:.2f}, {1:.2f}, {2:.2f},
{3:.2f}'.format(init_state[0],init_state[1],
init_state[2],init_state[3]), fontsize=16)
plt.show()
```

```
"""task 1.2 change of state"""
cartpole1.reset()
x_scan = np.linspace(-5,5,100)
x_dot_scan = np.linspace(-10,10,100)
theta_scan = np.linspace(np.pi-0.5,np.pi,100)
theta_dot_scan = np.linspace(10,15,100)
# scans of variables
Y0, Y1, Y2, Y3 = [],[],[],[] # next state
Y01,Y11,Y21,Y31 = [],[],[],[] # change in state
for x in x_scan:
    X = [x,0,np.pi,0]
    cartpole1.setState(X)
    cartpole1.performAction()
    Y = np.array(cartpole1.getState())
    Y0.append(Y)
    Y01.append(Y-X)
Y0 = np.array(Y0)
Y01 = np.array(Y01)
for x_dot in x_dot_scan:
    X = [0,x_dot,np.pi,0]
    cartpole1.setState(X)
    cartpole1.performAction()
    Y = np.array(cartpole1.getState())
    Y1.append(Y)
    Y11.append(Y-X)
Y1 = np.array(Y1)
Y11 = np.array(Y11)
for theta in theta_scan:
    X = [0,0,theta,0]
    cartpole1.setState(X)
    cartpole1.performAction()
    Y = np.array(cartpole1.getState())
    Y2.append(Y)
    Y21.append(Y-X)
Y2 = np.array(Y2)
Y21 = np.array(Y21)
for theta_dot in theta_dot_scan:
    X = [0,0,np.pi,theta_dot]
    cartpole1.setState(X)
    cartpole1.performAction()
    Y = np.array(cartpole1.getState())
    Y3.append(Y)
    Y31.append(Y-X)
Y3 = np.array(Y3)
Y31 = np.array(Y31)
# plotting scans of variables
# can change Y0, Y1, Y2, Y3 -> Y01,Y11,Y21,Y31 in the setting of Y = X(T)-X(0)
fig, axs = plt.subplots(2,2,figsize=(12,8),constrained_layout=True)
axs[0,0].plot(x_scan,Y0[:,0],label='next location')
axs[0,0].plot(x_scan,Y0[:,1],label='next velocity')
axs[0,0].plot(x_scan,Y0[:,2],label='next pole angle')
```

```
axs[0,0].plot(x_scan,Y0[:,3],label='next pole velocity')
axs[0,0].set_title('cart_location')
axs[0,0].set_xlabel('x (m)')
axs[0,0].set_ylabel('next state Y')
axs[0,1].plot(x_dot_scan,Y1[:,0],label='next location')
axs[0,1].plot(x_dot_scan,Y1[:,1],label='next velocity')
axs[0,1].plot(x_dot_scan,Y1[:,2],label='next pole angle')
axs[0,1].plot(x_dot_scan,Y1[:,3],label='next pole velocity')
axs[0,1].set_title('cart_velocity')
axs[0,1].set_xlabel('x_dot (m/s)')
axs[0,1].set_ylabel('next state Y')
axs[1,0].plot(theta_scan,Y2[:,0],label='next location')
axs[1,0].plot(theta_scan,Y2[:,1],label='next velocity')
axs[1,0].plot(theta_scan,Y2[:,2],label='next pole angle')
axs[1,0].plot(theta_scan,Y2[:,3],label='next pole velocity')
axs[1,0].set_title('pole_angle')
axs[1,0].set_xlabel('theta (rad)')
axs[1,0].set_ylabel('next state Y')
axs[1,1].plot(theta_dot_scan,Y3[:,0],label='next location')
axs[1,1].plot(theta_dot_scan,Y3[:,1],label='next velocity')
axs[1,1].plot(theta_dot_scan,Y3[:,2],label='next pole angle')
axs[1,1].plot(theta_dot_scan,Y3[:,3],label='next pole velocity')
axs[1,1].set_title('pole_velocity')
axs[1,1].set_xlabel('theta_dot (rad/s)')
axs[1,1].set_ylabel('next state Y')
fig.suptitle('Y as a function of scans of state variables', fontsize=16)
handles, labels = axs[1,1].get_legend_handles_labels()
fig.legend(handles, labels)
# fig.tight_layout()
plt.show()

# generating contours
Y01,Y02,Y03,Y12,Y13,Y23 = [],[],[],[],[],[]
for x_dot in x_dot_scan: # x & x_dot
    Y0 = []
    for x in x_scan:
        X = np.array([x,x_dot,np.pi,0])
        cartpole1.setState(X)
        cartpole1.performAction()
        Y = np.array(cartpole1.getState())
        Y0.append((Y-X))
    Y01.append(Y0)
Y01 = np.array(Y01)
for theta in theta_scan: # x & theta
    Y0 = []
    for x in x_scan:
        X = np.array([x,0,theta,0])
        cartpole1.setState(X)
        cartpole1.performAction()
        Y = np.array(cartpole1.getState())
        Y0.append((Y-X))
```

```
    Y02.append(Y0)
Y02 = np.array(Y02)
for theta_dot in theta_dot_scan: # x & theta_dot
    Y0 = []
    for x in x_scan:
        X = np.array([x,0,np.pi,theta_dot])
        cartpole1.setState(X)
        cartpole1.performAction()
        Y = np.array(cartpole1.getState())
        Y0.append((Y-X))
    Y03.append(Y0)
Y03 = np.array(Y03)
for theta in theta_scan: # x_dot & theta
    Y1 = []
    for x_dot in x_dot_scan:
        X = np.array([0,x_dot,theta,0])
        cartpole1.setState(X)
        cartpole1.performAction()
        Y = np.array(cartpole1.getState())
        Y1.append((Y-X))
    Y12.append(Y1)
Y12 = np.array(Y12)
for theta_dot in theta_dot_scan: # x_dot & theta_dot
    Y1 = []
    for x_dot in x_dot_scan:
        X = np.array([0,x_dot,np.pi,theta_dot])
        cartpole1.setState(X)
        cartpole1.performAction()
        Y = np.array(cartpole1.getState())
        Y1.append((Y-X))
    Y13.append(Y1)
Y13 = np.array(Y13)
for theta_dot in theta_dot_scan: # theta & theta_dot
    Y2 = []
    for theta in theta_scan:
        X = np.array([0,0,theta,theta_dot])
        cartpole1.setState(X)
        cartpole1.performAction()
        Y = np.array(cartpole1.getState())
        Y2.append((Y-X))
    Y23.append(Y2)
Y23 = np.array(Y23)
```

```
def contour_plots(x,y,z,name):
    fig, axs = plt.subplots(2,2,figsize=(9,16),constrained_layout=True)
    x, y = np.meshgrid(x,y)
    triang = tri.Triangulation(x.flatten(), y.flatten())
    cntr1 = axs[0,0].tricontourf(triang, z[:,:,0].flatten())
    fig.colorbar(cntr1, ax=axs[0,0])
    axs[0,0].tricontour(triang,z[:,:,0].flatten())
    axs[0,0].set_xlabel(name[0])
    axs[0,0].set_ylabel(name[1])
    axs[0,0].set_title('change in cart_location')
    cntr2 = axs[0,1].tricontourf(triang, z[:,:,1].flatten())
    fig.colorbar(cntr2, ax=axs[0,1])
    axs[0,1].tricontour(triang,z[:,:,1].flatten())
    axs[0,1].set_xlabel(name[0])
    axs[0,1].set_ylabel(name[1])
    axs[0,1].set_title('change in cart_velocity')
    cntr3 = axs[1,0].tricontourf(triang, z[:,:,2].flatten())
    fig.colorbar(cntr3, ax=axs[1,0])
    axs[1,0].tricontour(triang,z[:,:,2].flatten())
    axs[1,0].set_xlabel(name[0])
    axs[1,0].set_ylabel(name[1])
    axs[1,0].set_title('change in pole angle')
    cntr4 = axs[1,1].tricontourf(triang, z[:,:,3].flatten())
    fig.colorbar(cntr4, ax=axs[1,1])
    axs[1,1].tricontour(triang,z[:,:,3].flatten())
    axs[1,1].set_xlabel(name[0])
    axs[1,1].set_ylabel(name[1])
    axs[1,1].set_title('change in pole_velocity')
    fig.suptitle('Y changes with {} and {}'.format(name[0],name[1]),fontsize=16)
    plt.show()

contour_plots(x_scan,x_dot_scan,Y01,['x','x_dot'])
contour_plots(x_scan,theta_scan,Y02,['x','theta'])
contour_plots(x_scan,theta_dot_scan,Y03,['x','theta_dot'])
contour_plots(x_dot_scan,theta_scan,Y12,['x_dot','theta'])
contour_plots(x_dot_scan,theta_dot_scan,Y13,['x_dot','theta_dot'])
contour_plots(theta_scan,theta_dot_scan,Y23,['theta', 'theta_dot'])
```

```
"""task 1.3 linear regression"""
def lin_reg(X,Y):
    X = np.matrix(X)
    XT = np.matrix.transpose(X)
    Y = np.matrix(Y)
    XT_X = np.matmul(XT, X)
    XT_Y = np.matmul(XT, Y)
    betas = np.matmul(np.linalg.inv(XT_X), XT_Y)
    return betas

cartpole1 = CartPole()
# generating dataset for training
X = []
Y = []
N = 500 # no of datapoints
for i in range(N):
    x = random.uniform([-5,5,1])[0]
    x_dot = random.uniform([-10,10,1])[0]
    theta = random.uniform([-np.pi,np.pi,1])[0]
    theta_dot = random.uniform([-15,15,1])[0]
    Xn = np.array([x,x_dot,theta,theta_dot])
    X.append(Xn)
    cartpole1.setState(Xn)
    cartpole1.performAction()
    cartpole1.remap_angle()
    Xn_1 = np.array(cartpole1.getState())
    Y.append(Xn_1-Xn)
X = np.array(X)
Y = np.array(Y)
# linear regression
coef = lin_reg(X,Y)
Y_predict = np.matmul(X,coef)
Y_predict = np.array(Y_predict)
# plotting scatter point contrsting real and predict data for next state
fig, axs = plt.subplots(2,2,figsize=(12,8),constrained_layout=True)
axs[0,0].scatter(Y[:,0]+X[:,0],Y_predict[:,0]+X[:,0])
axs[0,0].plot(Y[:,0]+X[:,0],Y[:,0]+X[:,0],color='r',alpha=0.7)
axs[0,0].set_title('cart_location')
axs[0,0].set_xlabel('real (m)')
axs[0,0].set_ylabel('predict (m)')
axs[0,1].scatter(Y[:,1]+X[:,1],Y_predict[:,1]+X[:,1])
axs[0,1].plot(Y[:,1]+X[:,1],Y[:,1]+X[:,1],color='r',alpha=0.7)
axs[0,1].set_title('cart_velocity')
axs[0,1].set_xlabel('real (m/s)')
axs[0,1].set_ylabel('predict (m/s)')
axs[1,0].scatter(Y[:,2]+X[:,2],Y_predict[:,2]+X[:,2])
axs[1,0].plot(Y[:,2]+X[:,2],Y[:,2]+X[:,2],color='r',alpha=0.7)
axs[1,0].set_title('pole_angle')
axs[1,0].set_xlabel('real (rad)')
axs[1,0].set_ylabel('predict (rad)')
```

```
axs[1,1].scatter(Y[:,3]+X[:,3],Y_predict[:,3]+X[:,3])
axs[1,1].plot(Y[:,3]+X[:,3],Y[:,3]+X[:,3],color='r',alpha=0.7)
axs[1,1].set_title('pole_velocity')
axs[1,1].set_xlabel('real (rad/s)')
axs[1,1].set_ylabel('predict (rad/s)')
fig.suptitle('prediction vs real',fontsize=16)
plt.show()

# scans of variables can use the same segment of codes with a few changes in lines,
# so omitted here. the plot is generated using the following:
scan = 'x'
t = x_scan
y = Y0
yp = Y0_predict
fig, axs = plt.subplots(2,2,figsize=(9,16),constrained_layout=True)
axs[0,0].plot(t,y[:,0],label='true change')
axs[0,0].plot(t,yp[:,0],label='predicted change')
axs[0,0].set_title('cart_location')
axs[0,0].set_xlabel(scan)
axs[0,0].set_ylabel('location (m)')
axs[0,1].plot(t,y[:,1],label='true change')
axs[0,1].plot(t,yp[:,1],label='predicted change')
axs[0,1].set_title('cart_velocity')
axs[0,1].set_xlabel(scan)
axs[0,1].set_ylabel('velocity (m/s)')
axs[1,0].plot(t,y[:,2],label='true change')
axs[1,0].plot(t,yp[:,2],label='predicted change')
axs[1,0].set_title('pole_angle')
axs[1,0].set_xlabel(scan)
axs[1,0].set_ylabel('angle (rad)')
axs[1,1].plot(t,y[:,3],label='true change')
axs[1,1].plot(t,yp[:,3],label='predicted change')
axs[1,1].set_title('pole_velocity')
axs[1,1].set_xlabel(scan)
axs[1,1].set_ylabel('angular velocity (rad/s)')
fig.suptitle('Y vs Y_predict {} scan'.format(scan), fontsize=16)
handles, labels = axs[1,1].get_legend_handles_labels()
fig.legend(handles, labels)
axs[0,0].autoscale()
axs[0,1].autoscale()
axs[1,0].autoscale()
axs[1,1].autoscale()
# fig.tight_layout()
plt.show()
```

```
"""task 1.4 simulating time evolution"""
Xn = np.array([0,0,np.pi,-14.7]) # change initial state here
max_t = 4
steps = int(max_t/cartpole1.delta_time) # 0.2s per step
# time evolution
X_cartpole = []
X_model = []
Xn1_new = Xn
Xn2_new = Xn
for i in range(steps):
    Xn1 = Xn1_new
    Xn2 = Xn2_new
    # Yn1 = model.predict([Xn1])[0]
    Yn1 = np.matmul(Xn1,coef)
    Yn1 = np.array(Yn1)
    Xn1_new = Xn1 + Yn1
    # remapping the angle
    # print(Xn1_new)
    Xn1_new = Xn1_new.flatten()
    # print(Xn1_new)
    Xn1_new[2] = remap_angle(Xn1_new[2])
    X_model.append(Xn1_new)
    cartpole1.setState(Xn2)
    cartpole1.performAction()
    # # remapping the angle for performAction
    # cartpole1.remap_angle()
    Xn2_new = np.array(cartpole1.getState())
    X_cartpole.append(Xn2_new)
X_cartpole = np.array(X_cartpole)
X_model = np.array(X_model)

# plotting the results
t = np.arange(0,max_t,cartpole1.delta_time)
fig, axs = plt.subplots(2,2,figsize=(9,16),constrained_layout=True)
axs[0,0].plot(t,X_cartpole[:,0],label='true dynamic')
axs[0,0].plot(t,X_model[:,0],label='model')
axs[0,0].set_title('cart_location')
axs[0,0].set_xlabel('time (s)')
axs[0,0].set_ylabel('location (m)')
axs[0,0].set_xlim([0,5])
axs[0,0].autoscale()

axs[0,1].plot(t,X_cartpole[:,1],label='true dynamics')
axs[0,1].plot(t,X_model[:,1],label='model')
axs[0,1].set_title('cart_velocity')
axs[0,1].set_xlabel('time (s)')
axs[0,1].set_ylabel('velocity (m/s)')
axs[0,1].set_xlim([0,5])
axs[0,1].autoscale()
```

```
axs[1,0].plot(t,X_cartpole[:,2],label='true dynamics')
axs[1,0].plot(t,X_model[:,2],label='model')
axs[1,0].set_title('pole_angle')
axs[1,0].set_xlabel('time (s)')
axs[1,0].set_ylabel('angle (rad)')
axs[1,0].set_xlim([0,5])
axs[1,0].autoscale()

axs[1,1].plot(t,X_cartpole[:,3],label='true dynamics')
axs[1,1].plot(t,X_model[:,3],label='model')
axs[1,1].set_title('pole_velocity')
axs[1,1].set_xlabel('time (s)')
axs[1,1].set_ylabel('angular velocity (rad/s)')
axs[1,1].set_xlim([0,5])
axs[1,1].autoscale()

handles, labels = axs[1,1].get_legend_handles_labels()
fig.legend(handles, labels, loc='center')
fig.suptitle('model vs true dynamics given initial state: {0:.2f}, {1:.2f},
{2:.2f}, {3:.2f}'.format(Xn[0],Xn[1],Xn[2],Xn[3]), fontsize=16)

plt.show()
```