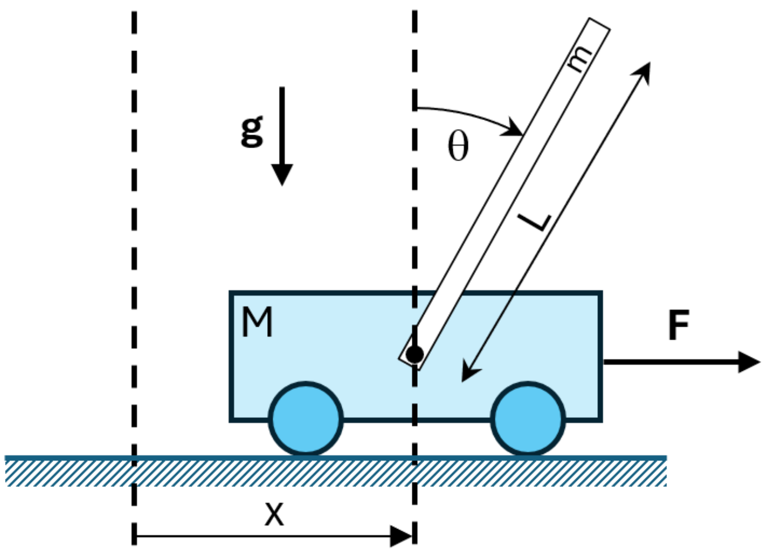
ENGINEERING TRIPOS PART IIA

SF3: MACHINE LEARNING

Interim Report

Hana Iza Kim

Pembroke College, May 2025



1 Introduction

In this interim report, we model the dynamics of a cart-pole system. The system can be described with the following four state variables: the cart position x, the cart velocity x', the pole angle θ and the pole angular velocity θ'. The dynamics of the cart-pole system are studied under various initial conditions and a linear model is developed, although its applicability is shown to be limited, as its predictions differ from those obtained by integrating the equations of motion.

A computational model that integrates the governing equations of motion of the cart-pole system is provided in *CartPole.py*. To improve the accuracy and temporal resolution of the simulation, the time step of the Euler integrator (*self.delta_time*) was reduced from 0.1 to 0.01. Additional temporary changes were made (e.g. changing the mass ratios and removing friction) to gain a better understanding of the system, but they were then reverted, and results presented here are obtained with the default values in the original file.

Task 1.1 - Dynamical Simulation

In this task, we use the provided *performAction* function with zero external force, i.e. free dynamics, to simulate the motion of the cart and pole for different initial conditions. Interesting dynamics are observed when the initial pole angle and velocity are varied.

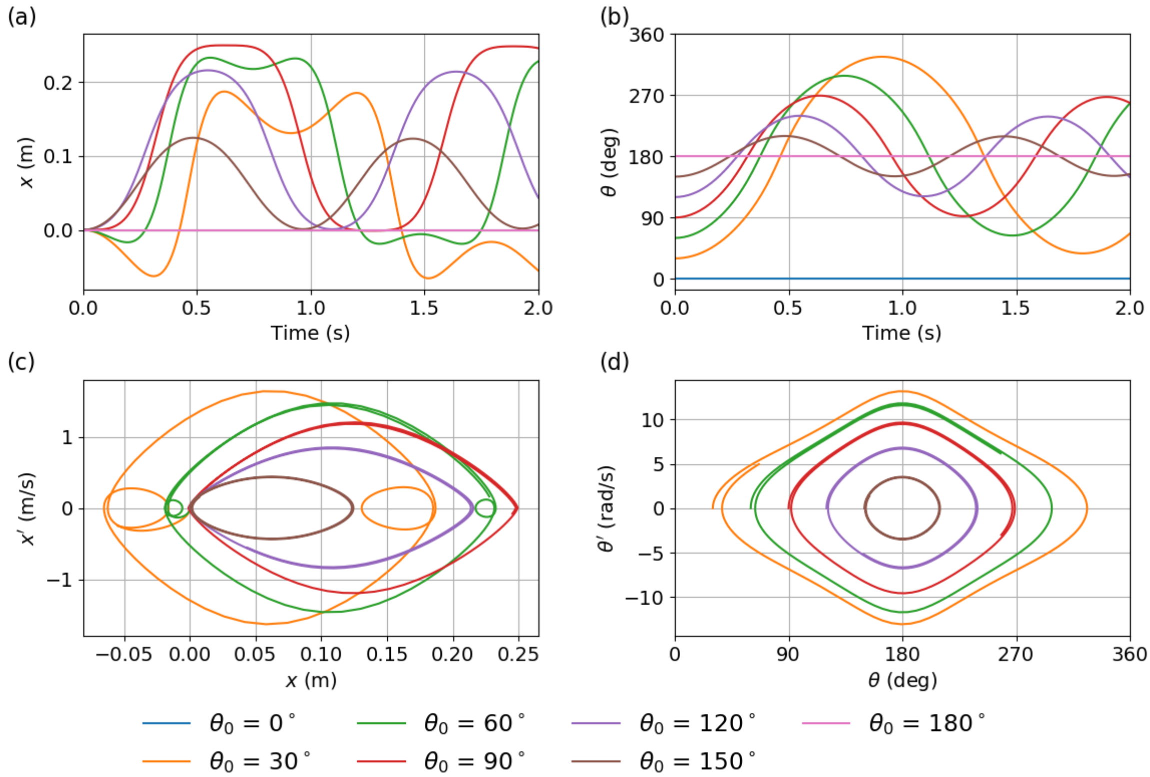


Figure 1: System dynamics for different initial positions of the pole: (a) Cart position x. (b) Pole position θ. (c) Cart phase plot (x' vs x). (d) Pole phase plot (θ' vs θ).

Figure 1 shows the dynamics when the system starts with the pole at various positions ($0 < \theta_o < \pi$) with no initial velocity ($\theta'_o = 0$) and the cart is stationary ($x_o = 0$, $x'_o = 0$). Symmetric dynamics are obtained when $-\pi < \theta_o < 0$ (data not shown). When $\theta_o = \pi$, the system is in its equilibrium position (see the $\theta_o = 180°$ lines in Figure 1). The same (lack of) dynamics is observed when $\theta_o = 0°$ as the system is in an unstable equilibrium. For any other initial angle, dynamics are observed. Angles in this section are remapped to an interval of 0 to 2π, to avoid discontinuities around the equilibrium position $\theta = \pi$ for presentation purposes.

Since there are no external forces applied to the system, if there were no frictional forces, the total momentum would be conserved and the centre of mass of the system would remain stationary. As the pole swings, its centre of mass moves to the left and therefore the cart needs to move to the right; and vice-versa when the pole swings back to the right (see the $\theta_o = 150°$ lines in Figure 1). In other words, the cart oscillates around an equilibrium position as the pole swings back and forth. As expected, the maximum oscillation of the cart takes place when the pole starts at $\theta_o = \pi/2$ (see Figure 1a). When the initial position of the pole is above the horizontal ($\theta_o < \pi/2$), its centre of mass moves to the right before it starts moving to the left as it swings towards the equilibrium position. This causes an additional oscillation in the cart motion, which can be recognised in the phase plot as small loops at the extreme values of x (see Figure 1c). If there were no energy losses in the system, the dynamics would be perfectly periodic, i.e. the trajectories in the phase plots would close themselves. However, it can be seen that due to the frictional losses (and to the finite time step and rounding errors), the trajectories do not close perfectly (see Figure 1c,d). Instead, they slowly drift towards the equilibrium position ($\theta = \pi, \theta' = 0$) as energy is dissipated in the system.

We know that for small perturbations around the equilibrium position (small angle approximation), the velocity of the cart and the pole are linearly related. This can be seen in the simulation results shown in Figure 2. As the perturbation increases, non-linear dynamics become increasingly dominant and the two velocities are no longer proportional to each other (see Figure 2).

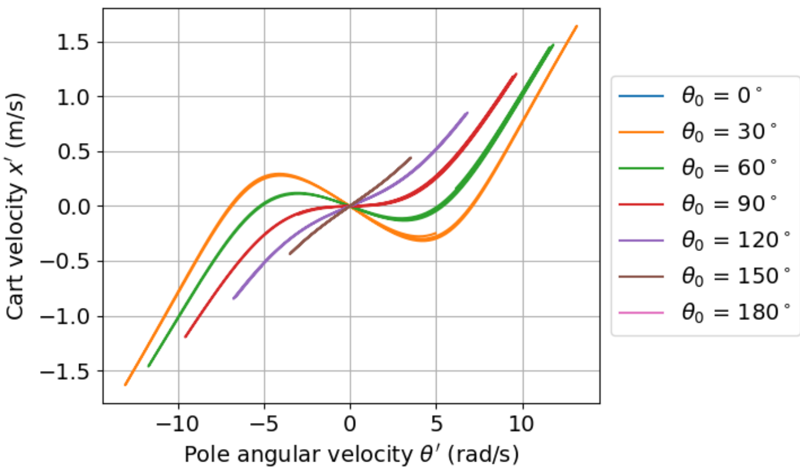


Figure 2: Phase plot showing the correlation between the cart and pole velocities for various initial pole angles. Initial conditions: $x_o = x'_o = \theta'_o = 0$.

Figure 3 shows similar plots for the system starting in its equilibrium position ($x_o = x'_o = 0$, $\theta_o = \pi$) but with the pole having different initial angular velocities. As in the previous case, since there are no external forces and the frictional forces are small, the total momentum of the system is almost conserved. In this case, as the system has horizontal momentum at $t = 0$ due to the velocity of the pole, the cart must move to the left as time goes on, which can be observed in Figure 3a. If the initial angular velocity of the pole is large enough, the pole can complete 360° turns, which can be observed for the initial angular velocity $\theta'_o = 15$rad/s case in Figure 3c. In this case, the angular velocity remains always positive (see Figure 3d).

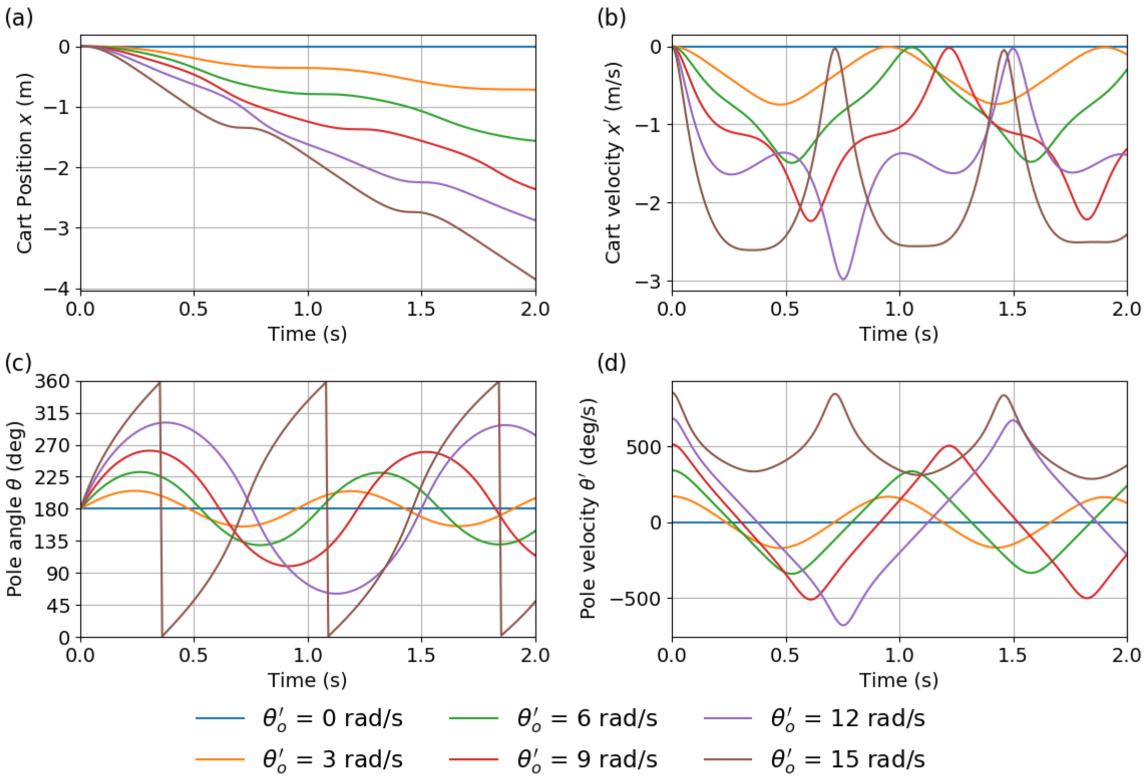


Figure 3: System dynamics for different initial angular velocity of the pole: (a) Cart position x. (b) Pole position θ. (c) Cart phase plot (x' vs x). (d) Pole phase plot (θ' vs θ).

Task 1.2 - Changes of State

The state of the system ($X = [x, x', \theta, \theta']$) after one time step depends on the previous state of the system, i.e. $X_{t+1} = f(X_t)$. In principle, this function f is complex, but provided the time step is small, the relationship is approximately linear. We can study the change in the state variables from a given state, i.e. $\Delta X = X_{t+1} - X_t$, based on observations of the system dynamics without any need for its governing equations.

Although not shown in the report, ΔX does not depend on the cart position x as the dynamics are exactly the same but with the whole system displaced in space. Figure 4 shows the change in state variables as a function of the initial pole position (θ_o) and angular velocity (θ'_o), assuming the cart is initially at rest ($x_o = x'_o = 0$). While the change in pole angle ($\Delta\theta$) is approximately independent of the pole angle (θ) and proportional to the pole velocity θ', i.e. $\Delta\theta \sim \theta' dt$ (see Figure 4c), the changes in cart position, cart velocity and pole velocity depend non-linearly on the initial angle and velocity of the pole (see Figure 4a,b,d). It is noted that the contour lines in Figure 4c are not horizontal and instead, they wobble due to the finite mass of the cart, i.e. the pole does not pivot around a stationary point. Although there are some numerical differences, the qualitative response is similar if the cart has some initial velocity, i.e. $x'_o \neq 0$.

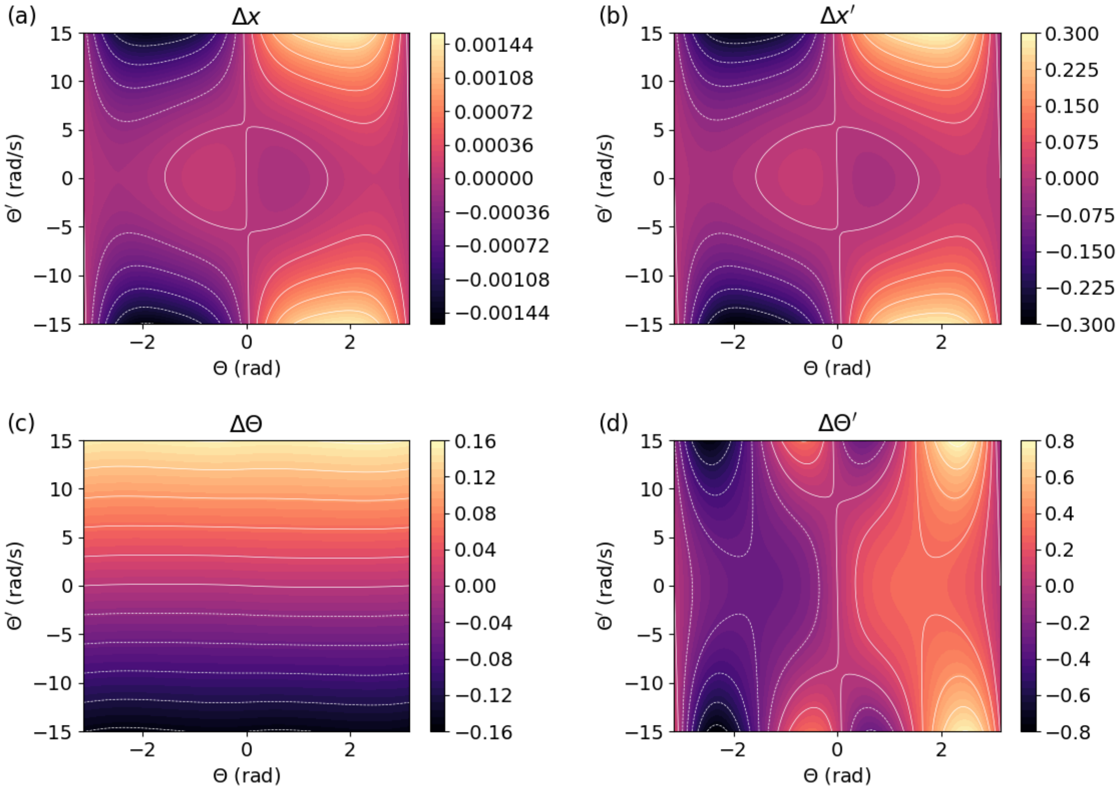


Figure 4: Change of state variables for $x = x' = 0$ and different θ and θ' initial conditions. (a) Δx, (b) $\Delta x'$, (c) $\Delta\theta$ and (d) $\Delta\theta'$.

Task 1.3 - Linear Model

The simplest model we can build of the system based on observations of its dynamics is a linear one in which the change in state variables is proportional to the state of the system:

$$Y = \Delta X = XC^T \tag{1}$$

where Y is the change in state X after one time step. To build the model, we observe ΔX for a number of random initial states and use that information to determine the 4×4 matrix C using standard linear regression (ordinary least squares).

Figure 5 compares the 'true' change in state variables (as predicted by the numerical integration of the equations of motion) after one time step with those predicted by the linear model. The data points correspond to 500 randomly sampled initial states within the following ranges: $-10 < x, x' < 10$, $-\pi < \theta < \pi$, and $-15 < \theta' < 15$ (blue points). In addition, a subset of initial states in which the pole is at a small angle ($|\theta| < 15°$) and angular velocity ($|\theta'| < 3$ rad/s) is also considered (orange points).

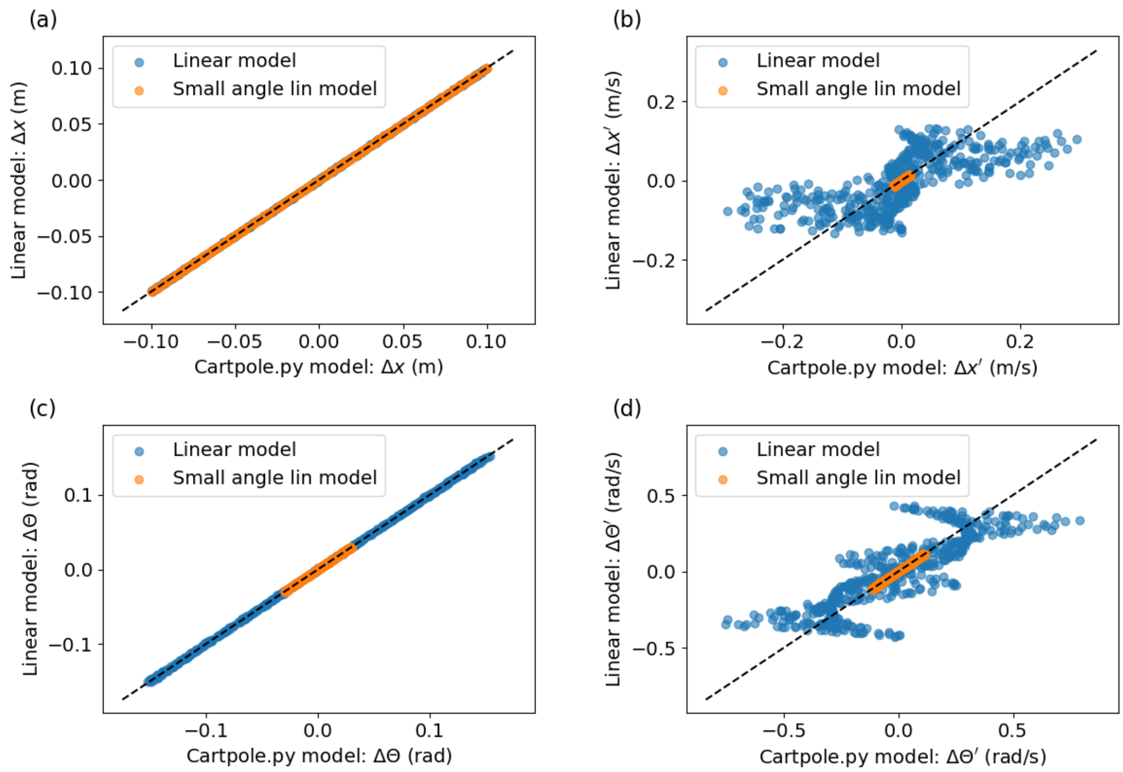


Figure 5: Comparison of state variable changes for 500 randomly selected initial states. $|x| < 10$, $|x'| < 10$, $|\theta| < \pi$, $|\theta'| < 15$. Small angle: $|\theta| < \pi/12$, $|\theta'| < 3$.

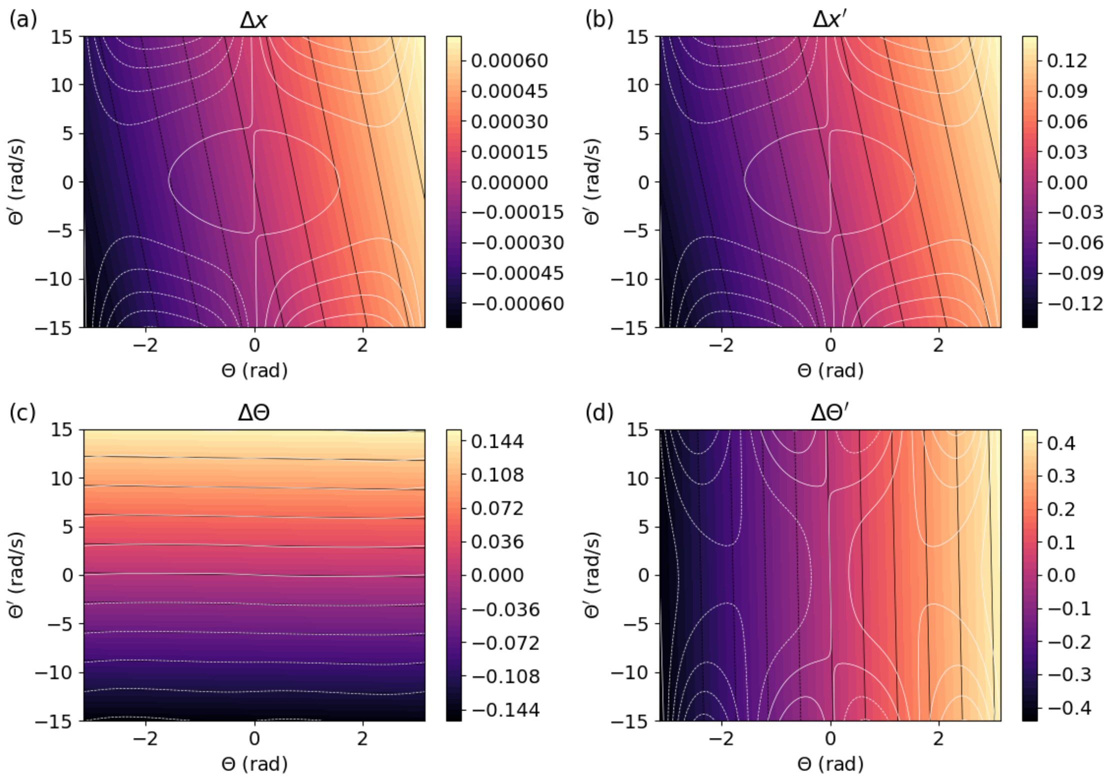


Figure 6: Change of state variables for $x = x' = 0$ and different θ and θ' initial conditions calculated using the linear model. (a) Δx, (b) $\Delta x'$, (c) $\Delta\theta$ and (d) $\Delta\theta'$. White overlaid lines correspond to the 'true' changes calculated by integrating the equations of motion.

The black dotted lines in Figure 5 represent the ideal case where the predicted and actual state changes are equal. Points lying close to this line represent state changes where the linear model agrees with the 'true' simulation, and therefore gives accurate predictions. As shown in Figure 5a,c, the change in cart position and pole angle are well predicted by the linear model. However, the changes in cart and pole velocities are poorly predicted (see Figure 5b,d). This highlights the limitation of the linear model. It is noted that a linear model built for small-angles around $\theta = 0$ (orange points) performs reasonably well and it is the larger angles and velocities that require non-linear components to be predicted accurately.

Figure 6 presents the predicted change in state variables using the linear model, equivalent to the 'true' changes in state variables shown in Figure 4. A few contour lines from Figure 4 have been overlaid in white on Figure 6 to aid the comparison. The prediction for $\Delta\theta$ closely matches the 'true' behaviour. However, for the other states, the model fails to reproduce the more complex, non-linear behaviour observed in the actual system, particularly at large θ and θ' values.

The linearised model approximates correctly the change in state variable x when varying x and x' but fails to do so for the rest of state variables. This is shown in Figure 7, which shows the changes predicted by the linear model (colour map and black contour lines) and compares it with the changes obtained by integrating the equations of motion (white lines).

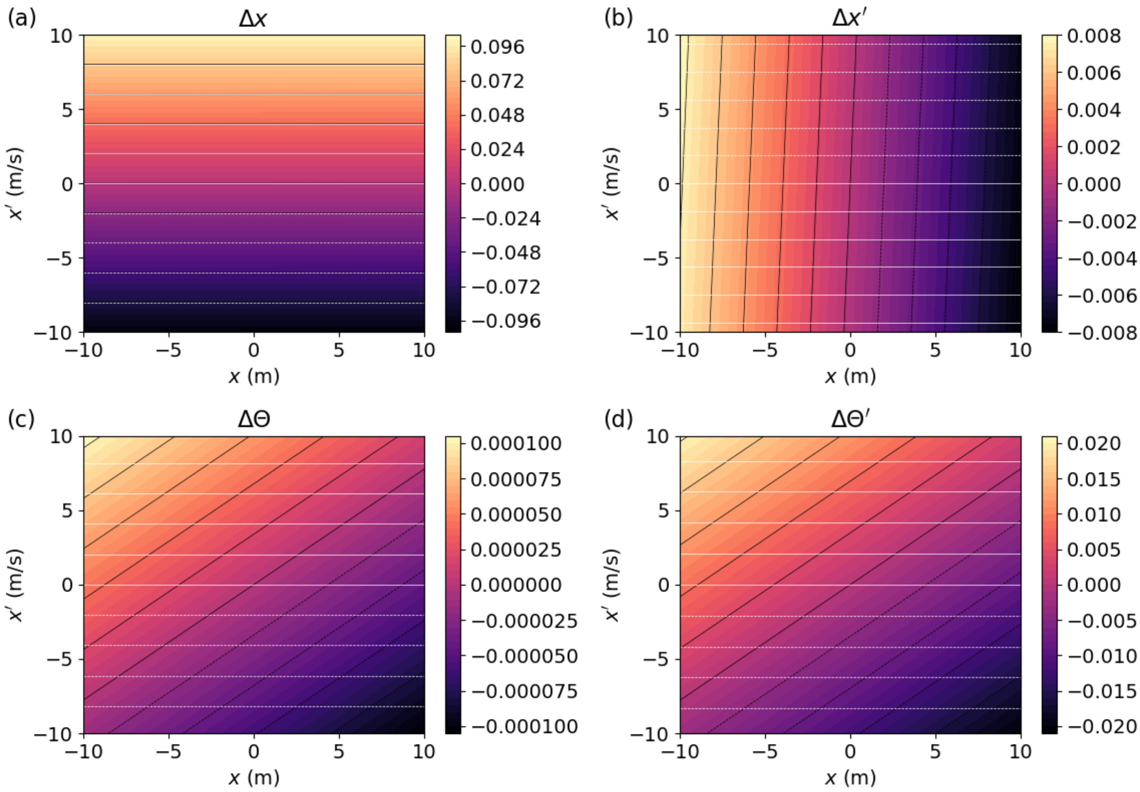


Figure 7: Change of state variables for $\theta = \theta' = 0$ and different x and x' initial conditions calculated using the linear model. (a) Δx, (b) $\Delta x'$, (c) $\Delta\theta$ and (d) $\Delta\theta'$. White overlaid lines correspond to the 'true' changes calculated by integrating the equations of motion.

Task 1.4 - System evolution

We can use the linear model developed in the previous section to simulate the system's time evolution and assess the linear model suitability by comparing the simulation results to those obtained from direct integration of the equations of motion (*CartPole.py* model).

Figure 8 shows the system dynamics when the cart and pole are initially stationary ($x_o = x'_o = \theta'_o = 0$) and the pole is let go at different angles. Although the state evolutions in both models are similar for the first few time steps, they quickly start to diverge. This is the case even for a starting angle of $\theta_o = \pi$, i.e. the system's stable equilibrium position (see purple lines for $\theta_o = 180°$ in Figure 8). This may seem surprising at first, but it is to be expected given the poor fit of the linear model at large angles $|\theta| >> 0$ (see Figure 6).

Although angles are remapped to an interval of 0° to 360° for presentation purposes, calculations are performed with the angle expressed in radians and remapped to $-pi$ to pi interval. This is necessary because the linear model cannot capture the periodic nature of angular motion and that is the pole angle range that was used for performing the linear regression.

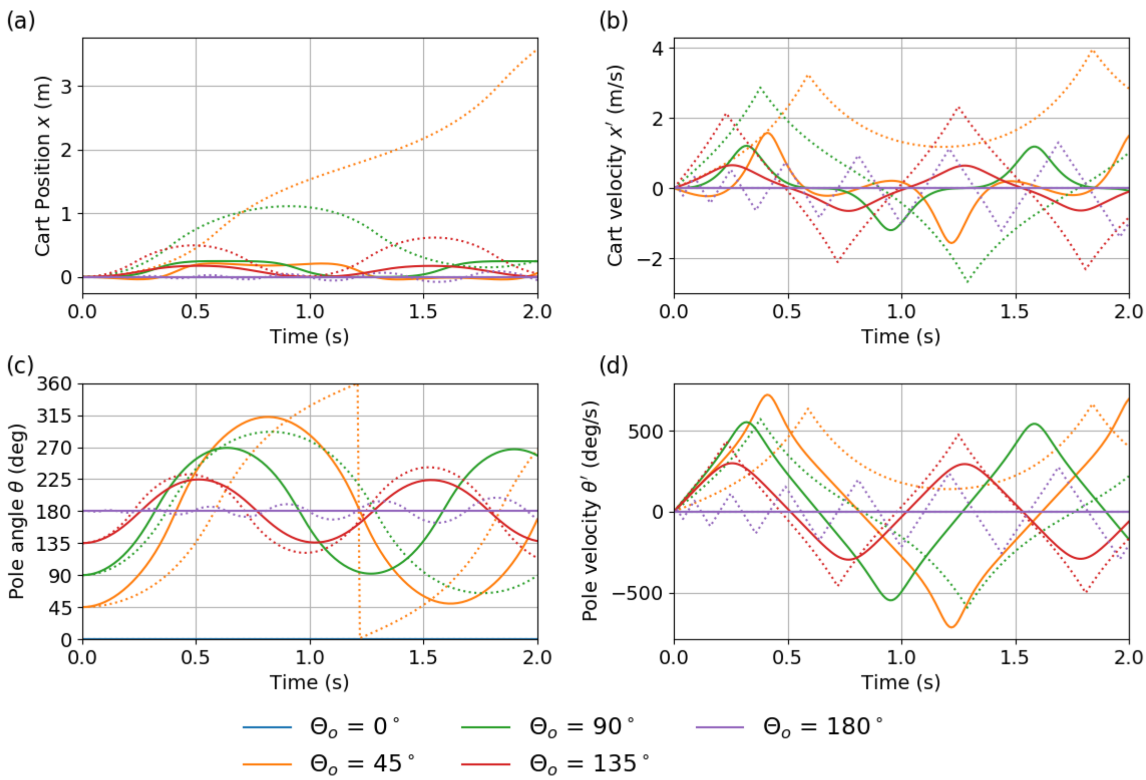


Figure 8: System dynamics predicted by the linear model (dotted lines) and the 'true' dynamics obtained by integrating the equations of motion (solid lines) for various initial positions of the pole: (a) Cart position x. (b) Cart velocity x' (c) Pole angle θ. (d) Pole velocity θ'.

The errors introduced by the linear model add energy into the system and as a result, the system becomes unstable. For example, letting the pole go at an initial angle of $\theta = 45°$ (orange lines in Figure 8), the linear model predicts that the pole swings past the stable equilibrium position ($\theta = \pi$) and rises to complete a full revolution instead of swinging back when reaching $\theta = 315°$. The cart dynamics are also incorrectly predicted, with the cart found to continuously travel in the +x direction instead of oscillating around an equilibrium position (see Figure 8a).

The only reasonable agreement between the two models occurs when the initial angle of the pole is $\theta_o = 0$. In this case, both models predict that the system remains stationary in the unstable equilibrium position, at least for the 2 seconds simulated. This is the case because the changes in state variables ΔX predicted by the two models agree when $\theta = \theta' = 0$ (see Figures 6, 7).

The unstable nature of the dynamics predicted by the linear model can be better captured in the phase plot diagrams. Figure 9 shows the θ' vs θ pole phase diagram for a simulation of 500 time steps (5 seconds). It can be seen that the results obtained by integrating the equations of motion predict a periodic motion that slowly spirals towards the origin due to frictional losses in the system (see $\times$ and $\bullet$ symbols in Figure 9). On the other hand, the dynamics predicted by the linear model spiral out very rapidly (see $\times$ and $\blacksquare$ symbols).

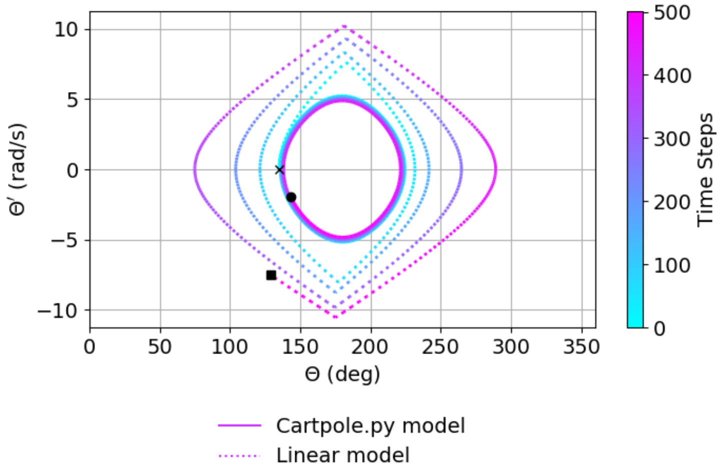


Figure 9: Evolution of the pole in phase plot θ' vs θ. Cart and pole initially at rest with the pole at an initial angle $\theta_o = 135°$. $\times$: Initial state, $\bullet$: Final state for 'true' model, $\blacksquare$: Final state for linear model.

Figure 9 also shows that the linearised model introduces an artifact at $\theta = \pi$. This stems from the poor velocity changes ($\Delta\theta'$) predictions the model makes at large $|\theta|$ values. As shown in Figure 6d, for $\theta \sim \pi$ the linear model underestimates the change in angular velocity and for $\theta \sim -\pi$ it overestimates it. Therefore, as the angle is remapped to a $[-\pi, \pi]$ interval, a discontinuity in angular acceleration is introduced.

Although it is beyond the scope of this study, it should be possible to use a different state variable ($\phi = \pi - \theta$) for the pole angle and develop a linear model to study small perturbations around the stable equilibrium position. That is not pursued here as we are interested in controlling the pole as an inverted pendulum. Given the unstable nature of the equilibirum at $\theta = 0$, it is not possible to capture the free dynamics of the system accurately with the linear model because without a controller the angle will always rapidly grow to $|\theta| = \pi$.

Appendix A - Code for Task 1.1: Dynamical simulation

```python
#Task 1.1
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.transforms import ScaledTranslation
from T00_CartPole import CartPole, remap_angle

def remap_angle_2pi(thetas):
    """Remap angles to the range [0, 2  )."""
    return thetas % (2 * np.pi) if np.isscalar(thetas) else np.array([theta % (2 * np.pi) for theta in thetas])

def simulate_rollout(initial_state, F, steps=200):
    """Simulate CartPole rollout given an initial state and force."""
    cartpole = CartPole(visual=False)
    cartpole.setState(initial_state)

    states = [initial_state]
    for _ in range(steps):
        cartpole.performAction(action=F)
        states.append(cartpole.getState().copy())

    time = np.arange(steps + 1) * cartpole.delta_time
    return time, np.array(states)

def add_legend(fig):
    """Clear and add a new legend to the figure."""
    for legend in fig.legends:
        legend.remove()
    fig.legends.clear()

    handles, labels = fig.axes[0].get_legend_handles_labels()
    if handles:
        fig.legend(handles, labels, loc='lower center', bbox_to_anchor=(0.5, 0.0),
                   ncol=int((len(labels)+1)/2), fontsize='large', frameon=False)

def plot_rollout(time, states_matrix, fig=None, style = "--", color = None, label=None):
    """Plot rollout trajectories: x, x',   ,   ' over time."""
    x, x_dot, theta, theta_dot = (
        states_matrix[:, 0],
        states_matrix[:, 1],
        remap_angle_2pi(states_matrix[:, 2]) * 180 / np.pi,
        states_matrix[:, 3] * 180 / np.pi
    )
    states = [x, x_dot, theta, theta_dot]
    ylabels = ['Cart Position $x$ (m)', 'Cart velocity $x\'$ (m/s)',
               'Pole angle $\\theta$ (deg)', "Pole velocity $\\theta'$ (deg/s)"]

    if fig is None:
        fig, _ = plt.subplots(2, 2, figsize=(12, 8))
        for i, ax in enumerate(fig.axes):
            ax.text(0.0, 1.0, f"({chr(ord('a') + i)})",
                    transform=ax.transAxes + ScaledTranslation(-55 / 72, +7 / 72, fig.dpi_scale_trans), fontsize=16)

    if color is None:
        color = next(fig.axes[0]._get_lines.prop_cycler)['color']

    for i, ax in enumerate(fig.axes):
        ax.plot(time, states[i], style, color=color, label=label)
        ax.set_xlabel('Time (s)')
        ax.set_ylabel(ylabels[i])
        ax.set_xlim(0, time[-1])
        ax.grid(True)
    fig.axes[2].set(ylim=(0, 360), yticks=np.arange(0, 361, 45))

    fig.subplots_adjust(left=0.1, right=0.97, top=0.95, bottom=0.2, hspace=0.35, wspace=0.3)
    add_legend(fig)
    return fig

def plot_phase_portraits(time, states_matrix, fig=None, style = "--", color=None, label=None):
    """Plot phase portraits for CartPole."""
    x, x_dot, theta, theta_dot = (
        states_matrix[:, 0],
        states_matrix[:, 1],
        remap_angle_2pi(states_matrix[:, 2]) * 180 / np.pi,
        states_matrix[:, 3]
    )

    if fig is None:
        fig, _ = plt.subplots(2, 2, figsize=(12, 8))

    if color is None:
        color = next(fig.axes[0]._get_lines.prop_cycler)['color']

    xlabels = ['$x$ (m)', r"$\theta$ (deg)", r"$\theta$ (deg)", r"$\theta'$ (rad/s)"]
    ylabels = [r"$x'$ (m/s)", r"$\theta'$ (rad/s)", r"$x$ (m)", r"$x'$ (m/s)"]
    data_pairs = [(x, x_dot), (theta, theta_dot), (theta, x), (theta_dot, x_dot)]
    for i, ax in enumerate(fig.axes):
        ax.plot(*data_pairs[i], style, color=color, label=label)
        ax.set(xlabel=xlabels[i], ylabel=ylabels[i])
        ax.grid(True)

```

```python
    plt.tight_layout()
    plt.subplots_adjust(bottom=0.15)
    add_legend(fig)
    return fig

def plot_report(time, states_matrix, fig=None, style = "-", color=None, label=None):
    """Generate report-style plots with subfigure labels."""
    x, x_dot, theta, theta_dot = (
        states_matrix[:, 0],
        states_matrix[:, 1],
        remap_angle_2pi(states_matrix[:, 2]) * 180 / np.pi,
        states_matrix[:, 3]
    )

    if fig is None:
        fig, axes = plt.subplots(2, 2, figsize=(12, 8))
        for i, ax in enumerate(axes.flatten()):
            ax.text(0.0, 1.0, f"({chr(ord('a') + i)})",
                    transform=ax.transAxes + ScaledTranslation(-55 / 72, +7 / 72, fig.dpi_scale_trans),
                    fontsize=16)

    if color is None:
        color = next(fig.axes[0]._get_lines.prop_cycler)['color']

    xlabels = ['Time (s)', 'Time (s)', '$x$ (m)', r'$\theta$ (deg)']
    ylabels = ['$x$ (m)', r'$\theta$ (deg)', "$x'$ (m/s)", r"$\theta'$ (rad/s)"]
    data_pairs = [(time, x), (time, theta), (x, x_dot), (theta, theta_dot)]
    for i, ax in enumerate(fig.axes):
        ax.plot(*data_pairs[i], style, color=color, label=label)
        ax.set(xlabel=xlabels[i], ylabel=ylabels[i])
        ax.grid(True)
    fig.axes[0].set(xlim=(0,np.max(time)))
    fig.axes[1].set(xlim=(0,np.max(time)),yticks=np.arange(0, 361, 90))
    fig.axes[3].set(xlim=(0, 360),xticks=np.arange(0, 361, 90))

    fig.subplots_adjust(left=0.1, right=0.97, top=0.95, bottom=0.2, hspace=0.35, wspace=0.3)
    add_legend(fig)
    return fig

# ============================ MAIN ============================
if __name__ == "__main__":
    plt.rcParams.update({'font.size': 14})
    # Varying initial pole angles
    fig_rollout, fig_phase, fig_report = None, None, None
    for theta_deg in [0, 30, 60, 90, 120, 150, 180]:
        initial_state = [0.0, 0.0, np.radians(theta_deg), 0.0]
        time, states = simulate_rollout(initial_state, F=0.0)
        label = f"$\\theta_o$ = {theta_deg}$^\\circ$"
        fig_rollout = plot_rollout(time, states, fig=fig_rollout, label=label)
        fig_phase = plot_phase_portraits(time, states, fig=fig_phase, label=label)
        fig_report = plot_report(time, states, fig=fig_report, label=label)
    plt.show(block=False)

    # Varying initial cart velocities
    fig_rollout, fig_phase, fig_report = None, None, None
    for x_dot in np.linspace(-10, 10, 7):
        initial_state = [0.0, x_dot, np.pi, 0.0]
        time, states = simulate_rollout(initial_state, F=0.0)
        label = f"$x'$ = {x_dot:.2f} m/s"
        fig_rollout = plot_rollout(time, states, fig=fig_rollout, label=label)
        fig_phase = plot_phase_portraits(time, states, fig=fig_phase, label=label)
        fig_report = plot_report(time, states, fig=fig_report, label=label)
    plt.show(block=False)

    # Varying initial pole angular velocities
    fig_rollout, fig_phase, fig_report = None, None, None
    for theta_dot in np.linspace(0, 15, 6):
        initial_state = [0.0, 0.0, np.pi, theta_dot]
        time, states = simulate_rollout(initial_state, F=0.0)
        label = f"$\\theta_o'$ = {theta_dot:.0f} rad/s"
        fig_rollout = plot_rollout(time, states, fig=fig_rollout, label=label)
        fig_phase = plot_phase_portraits(time, states, fig=fig_phase, label=label)
        fig_report = plot_report(time, states, fig=fig_report, label=label)
    plt.show(block=False)

    # Phase plot theta_dot vs x_dot
    plt.figure(figsize=(8,5))
    plt.subplots_adjust(left=0.15, right=0.7, top=0.9, bottom=0.2)
    for theta_deg in [0, 30, 60, 90, 120, 150, 180]:
        initial_state = [0.0, 0.0, np.radians(theta_deg), 0.0]
        time, states = simulate_rollout(initial_state, F=0.0)
        label = f"$\\theta_0$ = {theta_deg}$^\\circ$"
        plt.plot(states[:, 3], states[:, 1], label=label)

    plt.xlabel(r"Pole angular velocity $\theta'$ (rad/s)")
    plt.ylabel(r"Cart velocity $x'$ (m/s)")
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.grid(True)
    plt.show()
```

Appendix B - Code for Task 1.2: Change state variables

```python
# Task 1.2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.transforms import ScaledTranslation
from T00_CartPole import remap_angle
from T11_dynamic_simulation import simulate_rollout, add_legend

plt.rcParams.update({'font.size': 14})

F = 0.0  # No force applied to cart

#%% Line plots - x and x_dot values
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
fig.subplots_adjust(left=0.1, right=0.97, top=0.95, bottom=0.2, hspace=0.35, wspace=0.3)

xs = np.linspace(-5, 5, 100)
x_dots = np.linspace(-10, 10, 7)
for x_dot in x_dots:
    dxs, dx_dots, dthetas, dtheta_dots = [], [], [], []
    for x in xs:
        X0 = [x, x_dot, 0.0, 0.0]
        _, X_t = simulate_rollout(X0, F, steps=1)
        dX = X_t[-1] - X0
        dxs.append(dX[0])
        dx_dots.append(dX[1])
        dthetas.append(remap_angle(dX[2]))
        dtheta_dots.append(dX[3])

    delta_states = [dxs, dx_dots, dthetas, dtheta_dots]
    for i,ax in enumerate(fig.axes):
        ax.plot(xs,delta_states[i], label=f"$x'$ = {x_dot:.1f} m/s")

ylabels = ["$\Delta x$", "$\Delta x'$", "$\Delta \\theta$", "$\Delta \\theta'$"]
for i, ax in enumerate(fig.axes):
    ax.set_xlim(xs[0], xs[-1])
    ax.set_xlabel("$x$ (m)")
    ax.set_ylabel(ylabels[i])
    ax.grid(True)
    ax.text(0.0, 1.0, f"({chr(ord('a') + i)})",
            transform=(ax.transAxes + ScaledTranslation(-35 / 72, 7 / 72, fig.dpi_scale_trans)),
                fontsize=16)

add_legend(fig)
plt.show(block=False)

#%% Line plots - theta and theta_dot values
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
fig.subplots_adjust(left=0.1, right=0.97, top=0.95, bottom=0.2, hspace=0.35, wspace=0.35)

thetas = np.linspace(-np.pi, np.pi, 100)
theta_dots = np.linspace(0, 15, 7)
for theta_dot in theta_dots:
    dxs, dx_dots, dthetas, dtheta_dots = [], [], [], []
    for theta in thetas:
        X0 = [0, 0.0, theta, theta_dot]
        _, X_t = simulate_rollout(X0, F, 1)
        dX=(X_t[-1] - X0)
        dxs.append(dX[0])
        dx_dots.append(dX[1])
        dthetas.append(remap_angle(dX[2]))
        dtheta_dots.append(dX[3])

    delta_states = [dxs, dx_dots, dthetas, dtheta_dots]
    for i,ax in enumerate(fig.axes):
        ax.plot(thetas, delta_states[i], label=f"$\Theta$'={theta_dot:.0f} rad/s")

ylabels = ["$\Delta x$", "$\Delta x'$", "$\Delta \\theta$", "$\Delta \\theta'$"]
for i, ax in enumerate(fig.axes):
    ax.set_xlim(-np.pi, np.pi)
    ax.set_xlabel("$\Theta$ (rad)")
    ax.set_ylabel(ylabels[i])
    ax.grid(True)
    ax.text(0.0, 1.0, f"({chr(ord('a') + i)})",
            transform=(ax.transAxes + ScaledTranslation(-35 / 72, 7 / 72, fig.dpi_scale_trans)),
                fontsize=16)

add_legend(fig)
plt.show(block=False)

#%% Contours of delta states - x and x_dot
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
fig.subplots_adjust(left=0.1, right=0.93, top=0.95, bottom=0.08, hspace=0.35, wspace=0.45)
x_vals = np.linspace(-5, 5, 100)
x_dot_vals = np.linspace(-10, 10, 100)
X, X_dot = np.meshgrid(x_vals, x_dot_vals)
zeros = np.zeros_like(X)
dx, dx_dot, dtheta, dtheta_dot = zeros.copy(), zeros.copy(), zeros.copy(), zeros.copy()
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        X0 = np.array([X[i, j], X_dot[i, j], 0, 0])
        _, X_t = simulate_rollout(X0, F, 1)
        dX = X_t[-1] - X0
```

```python
        dx[i, j] = dX[0]
        dx_dot[i, j] = dX[1]
        dtheta[i, j] = remap_angle(dX[2])
        dtheta_dot[i, j] = dX[3]
titles = [r"$\Delta x$", r"$\Delta x'$", r"$\Delta \Theta$", r"$\Delta \Theta'$"]
delta_states = [dx, dx_dot, dtheta, dtheta_dot]
for i, ax in enumerate(fig.axes):
    cs = ax.contourf(X, X_dot, delta_states[i], levels=50, cmap='magma')
    ax.contour(X, X_dot, delta_states[i], levels=10, colors='white', linewidths=0.5)
    fig.colorbar(cs, ax=ax)
    ax.set(title=titles[i], xlabel="x (m)", ylabel="x' (m/s)")
    ax.text(0, 1, f"({chr(97 + i)})", transform=ax.transAxes + ScaledTranslation(-55/72, 7/72, fig.dpi_scale_trans), fontsize=16)
plt.show(block=False)

#%% Contours of delta states - theta and theta_dot
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
fig.subplots_adjust(left=0.1, right=0.93, top=0.95, bottom=0.08, hspace=0.4, wspace=0.45)
theta_vals = np.linspace(-np.pi, np.pi, 100)
theta_dot_vals = np.linspace(-15, 15, 100)
Theta, Theta_dot = np.meshgrid(theta_vals, theta_dot_vals)
zeros = np.zeros_like(X)
dx, dx_dot, dtheta, dtheta_dot = zeros.copy(), zeros.copy(), zeros.copy(), zeros.copy()
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        X0 = np.array([0,0,Theta[i, j], Theta_dot[i, j]])
        _, X_t = simulate_rollout(X0, F, 1)
        dX = X_t[-1] - X0
        dx[i, j] = dX[0]
        dx_dot[i, j] = dX[1]
        dtheta[i, j] = remap_angle(dX[2])
        dtheta_dot[i, j] = dX[3]
titles = [r"$\Delta x$", r"$\Delta x'$", r"$\Delta \Theta$", r"$\Delta \Theta'$"]
delta_states = [dx, dx_dot, dtheta, dtheta_dot]
for i, ax in enumerate(fig.axes):
    cs = ax.contourf(Theta, Theta_dot, delta_states[i], levels=50, cmap='magma')
    ax.contour(Theta, Theta_dot, delta_states[i], levels=10, colors='white', linewidths=0.5)
    fig.colorbar(cs, ax=ax)
    ax.set(title=titles[i], xlabel=r"$\Theta$ (rad)", ylabel=r"$\Theta'$ (rad/s)")
    ax.text(0, 1, f"({chr(97 + i)})", transform=ax.transAxes + ScaledTranslation(-55/72, 7/72, fig.dpi_scale_trans), fontsize=16)
plt.show()
```

Appendix C - Code for Task 1.3: Linear model

```python
# Task 1.3
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.transforms import ScaledTranslation
from T11_dynamic_simulation import simulate_rollout

def X_Y_dataset_step(N, limits):
    X = np.zeros((N, 4))
    Y = np.zeros((N, 4))
    for i in range(N):
        X0 = np.array([np.random.uniform(*limit) for limit in limits])
        _, X_t = simulate_rollout(X0, F=0.0, steps=1)
        X[i] = X0
        Y[i] = X_t[-1] - X0
    return X, Y

def least_squares_solution(X,Y):
    # least squares solution to Y = X  C_T
    C_T, *_ = np.linalg.lstsq(X, Y, rcond=None) #C = C_T.T  # shape (4, 4)
    return(C_T)

#==============================================
if __name__ == "__main__":
    plt.rcParams.update({'font.size': 14})
    N = 500 # Number of random initial states

    #%% Scatter plot
    #==================
    # Any initial state
    limits = [(-10,10),(-10,10),(-np.pi, np.pi),(-15,15)]
    X,Y= X_Y_dataset_step(N,limits)
    X_true = X + Y
    C_T = least_squares_solution(X,Y)
    Y_pred = X @ C_T  #Linear model
    X_pred = X  + Y_pred #Linear model

    fig, axes = plt.subplots(2, 2, figsize=(12, 8))
    fig.subplots_adjust(left=0.1, right=0.97, top=0.95, bottom=0.08, hspace=0.35, wspace=0.35)
    for i,ax in enumerate(fig.axes):
        ax.scatter(Y[:, i], Y_pred[:, i], alpha=0.6,label='Linear model')

    # Initial state limited to small angles
    limits = [(-10,10),(-10,10),(-np.pi/12, np.pi/12),(-3,3)]
    X,Y= X_Y_dataset_step(N,limits)
    X_true = X + Y
    C_T_2 = least_squares_solution(X,Y)
    Y_pred = X @ C_T_2  #Linear model
    X_pred = X  + Y_pred #Linear model

    for i,ax in enumerate(fig.axes):
        ax.scatter(Y[:, i], Y_pred[:, i], alpha=0.6,label='Small angle lin model')

    # Plot ideal line y=x, label axes and add legend
    state_labels = ['$\Delta x$ (m)', "$\Delta x'$ (m/s)", '$\Delta \Theta$ (rad)', "$\Delta \Theta'$ (rad/s)"]
    for i,ax in enumerate(fig.axes):
        ax.plot(ax.get_xlim(), ax.get_xlim(), ls="--", color='black')
        ax.set_xlabel(f'Cartpole.py model: {state_labels[i]}')
        ax.set_ylabel(f'Linear model: {state_labels[i]}')
        ax.text(0.0, 1.0, "("+chr(ord('a')+i)+")", transform=(ax.transAxes + ScaledTranslation(-55/72, +7/72, fig.dpi_scale_trans)), fontsize=16)
        ax.legend()

    #%% Contour plots - theta and theta_dot
    #==================================
    thetas = np.linspace(-np.pi, np.pi, 100)
    theta_dots = np.linspace(-15, 15, 100)
    Theta, Theta_dot = np.meshgrid(thetas, theta_dots)

    X = np.zeros((len(Theta.ravel()), 4))
    X[:, 2] = Theta.ravel()
    X[:, 3] = Theta_dot.ravel()
    Y_pred = X @ C_T  # Predicted deltas using linear model
    Y = np.zeros((len(Theta.ravel()), 4))
    for i, X0 in enumerate(X):
        _, X_t = simulate_rollout(X0, F=0.0, steps=1)
        Y[i] = X_t[-1] - X0

    fig, axes = plt.subplots(2, 2, figsize=(12, 8))
    fig.subplots_adjust(left=0.1, right=0.95, top=0.95, bottom=0.08, hspace=0.35, wspace=0.35)
    titles = [r"$\Delta x$", r"$\Delta x'$", r"$\Delta \Theta$", r"$\Delta \Theta'$"]
    for i, ax in enumerate(fig.axes):
        Z = Y_pred[:, i].reshape(Theta.shape)
        cs = ax.contourf(Theta, Theta_dot, Z, levels=50, cmap='magma')
        ax.contour(Theta, Theta_dot, Z, levels=10, colors='black', linewidths=0.5)
        Z = Y[:, i].reshape(Theta.shape)
        ax.contour(Theta, Theta_dot, Z, levels=10, colors='white', linewidths=0.5)
        fig.colorbar(cs, ax=ax)
        ax.set(title=titles[i], xlabel=r"$\Theta$ (rad)", ylabel=r"$\Theta'$ (rad/s)")
        ax.text(0, 1, f"({chr(97 + i)})", transform=ax.transAxes + ScaledTranslation(-55/72, 7/72, fig.dpi_scale_trans), fontsize=16)
    plt.show(block=False)

    #%% Contour plots - x and x_dot
    #================================
```

```python
    xs = np.linspace(-10, 10, 100)
    x_dots = np.linspace(-10, 10, 100)
    X, X_dot = np.meshgrid(xs, x_dots)

    States = np.zeros((len(X.ravel()), 4))
    States[:, 0] = X.ravel()
    States[:, 1] = X_dot.ravel()
    Y_pred = States @ C_T  # Predicted deltas using linear model
    Y = np.zeros((len(X.ravel()), 4))
    for i, X0 in enumerate(States):
        _, X_t = simulate_rollout(X0, F=0.0, steps=1)
        Y[i] = X_t[-1] - X0

    fig, axes = plt.subplots(2, 2, figsize=(12, 8))
    fig.subplots_adjust(left=0.1, right=0.95, top=0.95, bottom=0.08, hspace=0.35, wspace=0.4)
    titles = [r"$\Delta x$", r"$\Delta x'$", r"$\Delta \Theta$", r"$\Delta \Theta'$"]
    for i, ax in enumerate(fig.axes):
        Z = Y_pred[:, i].reshape(X.shape)
        cs = ax.contourf(X, X_dot, Z, levels=50, cmap='magma')
        ax.contour(X, X_dot, Z, levels=10, colors='black', linewidths=0.5)
        Z = Y[:, i].reshape(X.shape)
        ax.contour(X, X_dot, Z, levels=10, colors='white', linewidths=0.5)
        fig.colorbar(cs, ax=ax)
        ax.set(title=titles[i], xlabel=r"$x$ (m)", ylabel=r"$x'$ (m/s)")
        ax.text(0, 1, f"({chr(97 + i)})", transform=ax.transAxes + ScaledTranslation(-55/72, 7/72, fig.dpi_scale_trans), fontsize=16)

    plt.show()
```

Appendix D - Code for Task 1.4: System evolution

```python
#Task 1.4
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.transforms import ScaledTranslation
from matplotlib.collections import LineCollection
from T00_CartPole import CartPole, remap_angle
from T11_dynamic_simulation import simulate_rollout, add_legend, remap_angle_2pi, plot_rollout, plot_phase_portraits, plot_report
from T13_linear_model import X_Y_dataset_step, least_squares_solution

def linear_simulate_rollout(C_T, initial_state, steps=200):
    # linear model rollout using C_T
    state = initial_state
    rollout = [state]
    for _ in range(steps):
        delta = state @ C_T
        state = state + delta
        state[2] = remap_angle(state[2])  # to avoid divergence
        rollout.append(state.copy())
    return np.array(rollout)

#################
def plot_gradient_line(ax, x, y, cmap='cool'):
    segments = np.array([x, y]).T.reshape(-1, 1, 2)
    segments = np.concatenate([segments[:-1], segments[1:]], axis=1)
    lc = LineCollection(segments, cmap=cmap, norm=plt.Normalize(0, len(x)))
    lc.set_array(np.arange(len(x)))
    lc.set_linewidth(2)
    return ax.add_collection(lc)

#######################
if __name__ == "__main__":
    plt.rcParams.update({'font.size': 14})
    # Linear model
    N = 500
    limits = [(-10,10),(-10,10),(-np.pi, np.pi),(-15,15)]
    F = 0.0
    tsteps = 200
    X,Y = X_Y_dataset_step(N, limits)
    C_T = least_squares_solution(X, Y)

    #%% Varying initial cart velocity - x_dot
    fig_rollout, fig_phase, fig_report = None, None, None
    for x_dot in np.linspace(-10,10,7):
        initial_state = [0.0, x_dot, np.pi, 0.0] # start hanging down, with velocity

        label = f"$x^\prime$ = {x_dot:.2f}$m/s$"
        time, true_traj = simulate_rollout(initial_state, F, steps=tsteps)
        predicted_traj = linear_simulate_rollout(C_T, initial_state, steps=tsteps)

        # True model
        fig_rollout = plot_rollout(time,true_traj,fig=fig_rollout,label=label)
        # fig_phase = plot_phase_portraits(time,true_traj,fig=fig_phase,label=label)
        # fig_report = plot_report(time,true_traj,fig=fig_report,label=label)

        # Linear model: Dotted line
        lastcolor = fig_rollout.axes[0].lines[-1].get_color()
        fig_rollout = plot_rollout(time, predicted_traj, fig=fig_rollout, style=":", color=lastcolor, label=None)
        # fig_phase = plot_phase_portraits(time,predicted_traj, fig=fig_phase, style=":", color=lastcolor, label=None)
        # fig_report = plot_report(time,predicted_traj, fig=fig_report,style=":",color=lastcolor, label=None)

    plt.show(block=False)

    #%% Varying initial cart angle - theta
    fig_rollout, fig_phase, fig_report = None, None, None
    for theta_deg in [0,45,90,135,180]:
        initial_state = [0.0, 0.0, theta_deg/180*np.pi, 0.0]

        label = f"$\Theta_o$ = {theta_deg}$^\circ$"
        time, true_traj = simulate_rollout(initial_state, F, steps=tsteps)
        predicted_traj = linear_simulate_rollout(C_T, initial_state, steps=tsteps)

        # True model
        fig_rollout = plot_rollout(time,true_traj,fig=fig_rollout,label=label)
        # fig_phase = plot_phase_portraits(time,true_traj,fig=fig_phase,label=label)
        # fig_report = plot_report(time,true_traj,fig=fig_report,label=label)

        # Linear model: Dotted line
        lastcolor = fig_rollout.axes[0].lines[-1].get_color()
        fig_rollout = plot_rollout(time, predicted_traj, fig=fig_rollout, style=":", color=lastcolor, label=None)
        # fig_phase = plot_phase_portraits(time,predicted_traj, fig=fig_phase, style=":", color=lastcolor, label=None)
        # fig_report = plot_report(time,predicted_traj, fig=fig_report,style=":",color=lastcolor, label=None)

    plt.show(block=False)

    #%% Phase plot - theta vs theta_dot
    fig_report = None
    for theta_deg in [135]:
        initial_state = [0.0, 0.0, theta_deg/180*np.pi, 0.0]
```

```python
        label = f"$\\Theta_o$ = {theta_deg}$^\\circ$"
        time, true_traj = simulate_rollout(initial_state, F, steps=500)
        predicted_traj = linear_simulate_rollout(C_T, initial_state, steps=500)

        fig = plt.figure(figsize=(8, 5))
        ax = fig.add_subplot(1, 1, 1)
        line=plot_gradient_line(ax, remap_angle_2pi(true_traj[:,2])*180/np.pi, true_traj[:,3])
        ax.plot([], [], color=plt.get_cmap('cool')(0.8), label="Cartpole.py model")
        line2=plot_gradient_line(ax, remap_angle_2pi(predicted_traj[:,2])*180/np.pi, predicted_traj[:,3])
        ax.plot([], [], ':', color=plt.get_cmap('cool')(0.8), label="Linear model")
        line2.set_linestyle(':')
        plt.plot(remap_angle_2pi(true_traj[0,2])*180/np.pi, true_traj[0,3], 'x',color='black')
        plt.plot(remap_angle_2pi(true_traj[-1,2])*180/np.pi, true_traj[-1,3], 'o',color='black')
        plt.plot(remap_angle_2pi(predicted_traj[-1,2])*180/np.pi, predicted_traj[-1,3], 's',color='black')
        cbar = plt.colorbar(line, ax=ax)
        cbar.set_label('Time Steps')
        plt.xlim(0, 360)
        ymin, ymax = plt.ylim()
        plt.ylim(-ymax, ymax)
        plt.xlabel(r'$\Theta$ (deg)')
        plt.ylabel(r'$\Theta^\prime$ (rad/s)')
        plt.subplots_adjust(left=0.15, right=0.9, top=0.9, bottom=0.3)
        plt.legend(loc='lower center', bbox_to_anchor=(0.5, -0.5), ncol=1, frameon=False)
        plt.grid(True)

    plt.show()
```