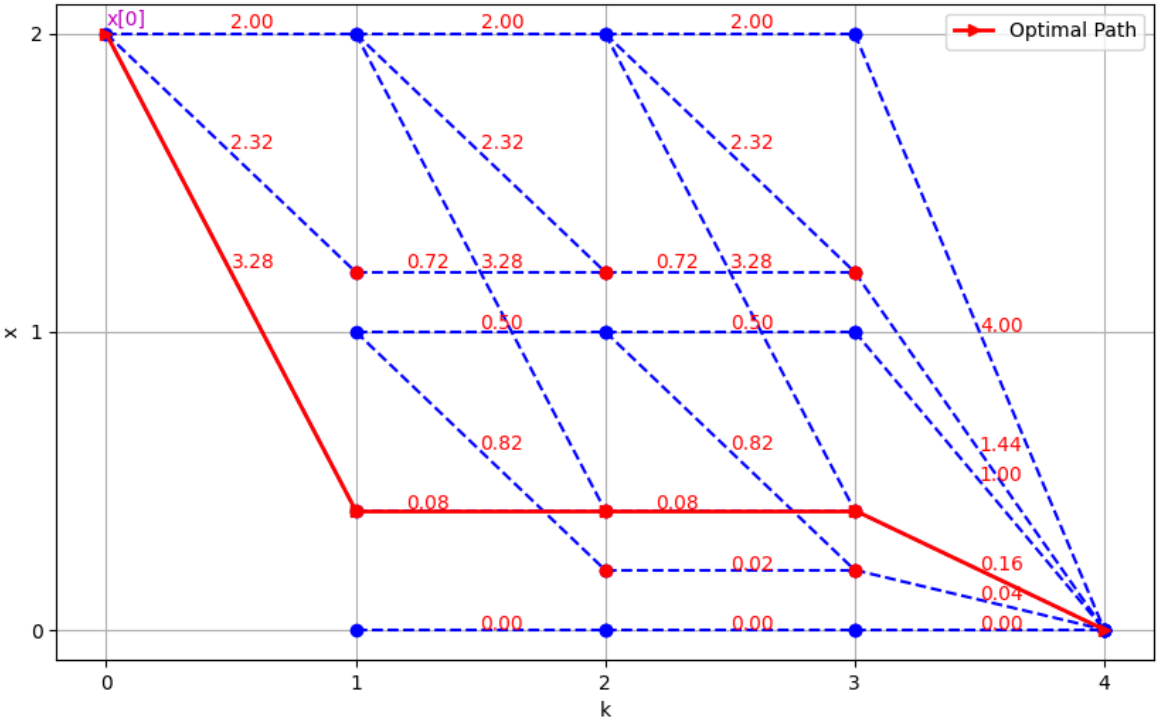
ΔΥΝΑΜΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΑΝΑΦΟΡΑ 2ης ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ

Σοφιανόπουλος Έκτορας 2017010016

https://colab.research.google.com/drive/1fz32vc3IKLRPNrhycfKobRxc2fF0XhG9?usp=sharing



Στα πλαίσια του του μαθήματος επιλύθηκε το ακόλουθο ΠΒΕ :

$$J = \frac{1}{2}\sum_{K=0}^{3}[x^2(k) + u^2(k)]$$

υπό τους περιορισμούς:

$$x(k+1) = x(k) + u(k)$$

$$x(0) = 2 \quad , \quad x(4) = 0$$

$$\boldsymbol{x}(k) = \{x(k) |\, 0 \le x(k) \le 2\}, \quad \boldsymbol{u}(k) = \{u(k) |\, -2 \le u(k) \le 0\}$$

Έστω $\Delta x(k) = 1$, για $k = 1,2,3$
$\Delta u(k) = 0.8$ για $k = 0,1,2$ και $\Delta u(k) = 0.2$ για $k = 3$.

Αρχικά κατασκευάστηκε το graph για τους δυνατούς συνδυασμούς k, x και u, φροντίζοντας για k=0 τα διάφορα transitions να έχουν ως έναρξη το x= 2, ενώ για k=3 να καταλήγουν στο x = 0 (της βαθμίδας k = 4). Αφού ολοκληρώθηκε αυτό το κομμάτι, έγινε προσθήκη συνάρτησης υπολογισμού του κόστους κάθε μετάβασης και οπτικοποίηση των τιμών στο γράφημα, με τέτοιο τρόπο ώστε να μην υπάρχει overlapping text.

Η αντιστοίχιση των states x με τους ελέγχους u(x) έγινε σε ένα dictionary έτσι ώστε να εφαρμόζονται οι κατάλληλοι έλεγχοι ανάλογα με το x της κάθε επανάληψης.

Δεύτερο κομμάτι της επίλυσης αποτέλεσε ο υπολογισμός του “cost to go” κάθε κόμβου. Για την αποθήκευση των τιμών, αρχικοποιήθηκε ένα nested dictionary.

```
V = {k: {x: np.inf for x in states} for k in range(1, Khorizon)}
V[0] = {x_0: np.inf}
V[1] = {x: np.inf for x in states if x != 0.2}
V[Khorizon] = {0: 0}
```

Εξωτερικό dictionary : key = k, value = εσωτερικό dictionary : key = x, value = cost to go

Το cost to go κάθε κόμβου αρχικοποιήθηκε στο άπειρο, το κόστος του τελευταίου κόμβου και τελικού προορισμού στο 0, ενώ για τα timestamps με λιγότερα states δεσμεύτηκαν οι αντίστοιχες θέσεις στο dictionary.

Ο κύριος βρόγχος του κώδικα αναπτύσσεται ως εξής :

Για κάθε timestamp k, ξεκινώντας από το k = 3 και συνεχίζοντας αναδρομικά, γίνεται αντιστοίχιση του k με τα κατάλληλα states, το κάθε ένα από τα οποία αντιστοιχίζεται με τα κατάλληλα control inputs.
Για κάθε u(x), υπολογίζεται το σωστό επόμενο state μέσω της συνάρτησης dynamics() καθώς και το transition cost. Για k = 3, το επόμενο state γνωρίζουμε ότι είναι το 0.

Για states με ακέραια τιμή x, το cost to go υπολογίζεται προσθέτοντας το cost to go του x_next με το transition cost :

```
if x in [0, 1, 2]: V[k][x] = V[k+1][x_next] + cost
```

Για τα ενδιάμεσα states χρησιμοποιήθηκε ο τύπος γραμμικής παρεμβολής :
$y = y0 +(x1−x0)(y1−y0) × (x−x0)$, όπου y = V[k][x]

```
elif x in [0.2, 0.4]: V[k][x] = V[k][0] + (V[k][1]-V[k][0]) * x

elif x in [1.2]: V[k][x] = V[k][1] + (V[k][2] - V[k][1]) * (x -1)
```

Το αποτέλεσμα αποθηκεύεται στην θέση V[k][x] του dictionary εφόσον είναι μικρότερο από την τιμή που έχει ήδη. Έτσι το cost to go που αντιστοιχεί σε κάθε κατάσταση είναι πάντα το μικρότερο δυνατό.

Ο δεύτερος βρόγχος του κώδικα υπολογίζει και οπτικοποιεί την βέλτιστη διαδρομή. Σε κάθε timestamp ξεκινώντας από το 0, αρχικά γεμίζει το dictionary feasible_transitions = {} με τα δυνατά transitions από την τωρινή κατάσταση (key = feasible next state , value = cost to go). Ύστερα επιλέγει το state με το μικρότερο cost to go από το dictionary και το προσθέτει στη λίστα optimal_path. Το next_state γίνεται current_state και ο βρόγχος συνεχίζει μέχρι να μην υπάρχει το επόμενο state στο dictionary V[k].

Η βέλτιστη διαδρομή σχηματίζεται στο γράφημα στο τέλος κάθε επανάληψης με ένα κόκκινο βέλος ξεκινώντας από το k και το δεύτερο από το τέλος στοιχείο της λίστας optimal_path και καταλήγοντας στο k + 1 και στο τελευταίο στοιχείο της λίστας.

Για την αποφυγή error στο πρόγραμμα χρησιμοποιήθηκε η συνάρτηση :

```
def nearest_state(x, states):

  return min(states, key=lambda state: abs(state - x))
```

Η οποία μετατρέπει κάθε νέο state x που υπολογίζεται στην κοντινότερη τιμή η οποία ανήκει στη λίστα με τα states του προβλήματος.