Δ Ι Ε Θ Ν Ε Σ Π Α Ν Ε Π Ι Σ Τ Η Μ Ι Ο Τ Η Σ Ε Λ Λ Α Δ Ο Σ
Σ Χ Ο Λ Η Μ Η Χ Α Ν Ι Κ Ω Ν
Τ Μ Η Μ Α Μ Η Χ Α Ν Ι Κ Ω Ν Π Λ Η Ρ Ο Φ Ο Ρ Ι Κ Η Σ ,
Υ Π Ο Λ Ο Γ Ι Σ Τ Ω Ν Κ Α Ι Τ Η Λ Ε Π Ι Κ Ο Ι Ν Ω Ν Ι Ω Ν

ΣΥΣΤΗΜΑΤΑ ΑΥΤΟΜΑΤΟΥ ΕΛΕΓΧΟΥ ΣΤΗΝ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΓΛΩΣΣΑ ΡΥΤΗΟΝ

Πτυχιακή Εργασία του

Βασίλη Μυρίσα Μαγκούνη (3277)

Επιβλέπων: Στ. Βολογιαννίδης, Επίκουρος Καθηγητής

ΣΕΡΡΕΣ, ΦΕΒΡΟΥΑΡΙΟΣ 2021

Υπεύθυνη Δήλωση *: Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης, βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών του Διεθνούς Πανεπιστημίου της Ελλάδας.*

Περίληψη

Το αντικείμενο της εργασίας είναι η ανάπτυξη και η μελέτη συστημάτων αυτομάτου ελέγχου με τη χρήση της γλώσσας προγραμματισμού Python. Για τη μελέτη των συστημάτων αυτομάτου ελέγχου χρησιμοποιήθηκε η βιβλιοθήκη της Python, Python Control Systems Library, όπως επίσης και οι βιβλιοθήκες NumPy, SciPy και Matplotlib για την εισαγωγή σημάτων, επίλυση μαθηματικών εξισώσεων και αναπαράσταση γραφικών παραστάσεων. Για την ανάπτυξη του κώδικα έγινε χρήση του ολοκληρωμένου περιβάλλοντος ανάπτυξης PyCharm.

Περιεχόμενα

Πίνακας εικόνων

Πίνακας πινάκων

Εισαγωγή

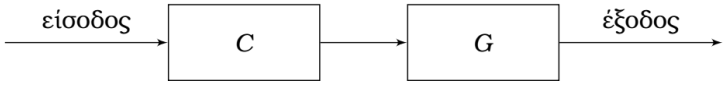
Για την κατανόηση των συστημάτων αυτομάτου ελέγχου, πρέπει να περιγράψουμε τι είναι ένα σύστημα. Σύστημα ονομάζουμε ένα σύνολο στοιχείων του φυσικού κόσμου τα οποία λειτουργούν μεταξύ τους για την επίτευξη κάποιου στόχου. Το περιβάλλον επιδρά σε ένα σύστημα με τη μορφή σημάτων εισόδου σε αυτό και εν συνεχεία το σύστημα παράγει σήματα εξόδου. Ένα σύστημα έχει προκαθορισμένη λειτουργία αναλόγως με τη χρήση του. Οι έξοδοι εξαρτώνται από τις αρχικές συνθήκες του συστήματος αλλά και τις προδιαγραφές του.[1]

Συστήματα Αυτομάτου Ελέγχου

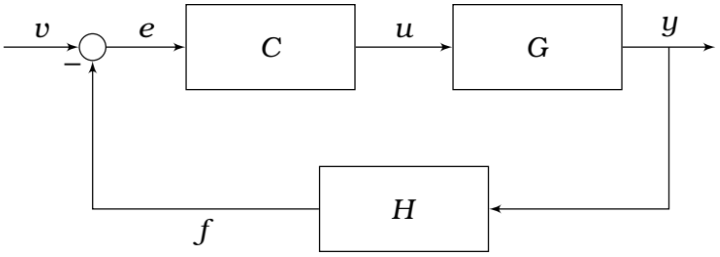
Συστήματα Αυτομάτου Ελέγχου (ΣΑΕ) ονομάζονται εκείνα τα συστήματα των οποίων η έξοδος ελέγχεται σύμφωνα με ορισμένα κριτήρια ή προδιαγραφές, από την είσοδο ελέγχου. Ο στόχος των ΣΑΕ είναι να μεταβάλουν τη συμπεριφορά ενός συστήματος με τέτοιον τρόπο ώστε να η έξοδός του να είναι η επιθυμητή. Οι βασικές κατηγορίες συστημάτων αυτομάτου ελέγχου είναι δύο. Τα συστήματα ανοιχτού βρόχου και τα συστήματα κλειστού βρόχου. Τα συστήματα ανοιχτού βρόχου έχουν μια προϋπολογισμένη είσοδο έτσι ώστε η έξοδος του συστήματος να είναι η επιθυμητή. Σε περίπτωση που το σήμα εισόδου δεχτεί αλλαγές από τυχαίους παράγοντες και καταλήξει να είναι διαφορετικό από αυτό που προϋπολογιζόταν τότε αντίστοιχα και η έξοδος του συστήματος θα διαφέρει από αυτήν που είναι επιθυμητή.
Τα συστήματα ανοιχτού βρόχου χαρακτηρίζονται από τον απλό σχεδιασμό τους, πράγμα που τα καθιστά εύχρηστα και πολύ φθηνά στην κατασκευή και στη συντήρησή τους. Μερικά παραδείγματα συστημάτων ανοιχτού βρόχου είναι η ηλεκτρική λυχνία, η βρύση, ο εκτυπωτής και το τηλεκοντρόλ.
Σε αντίθεση με τα συστήματα ανοιχτού βρόχου, τα συστήματα κλειστού βρόχου χρησιμοποιούν τη μέτρηση της εξόδου του συστήματος και τη συγκρίνουν με την επιθυμητή απόκριση εξόδου. Το σήμα που παράγεται κατά τη διαδικασία μέτρησης της εξόδου ονομάζεται σήμα ανάδρασης. Στα συστήματα στα οποία εφαρμόζεται η τεχνική της ανάδρασης η σχέση μεταξύ των μεταβλητών του συστήματος γίνεται σταθερή μέσω διαδικασιών σύγκρισης αυτών των μεταβλητών, με τη διαφορά τους να χρησιμοποιείται ως σήμα ελέγχου. Ένα σύστημα αυτομάτου ελέγχου κλειστού

[1] Πετρίδης Β., 2011. *Συστήματα Αυτομάτου Ελέγχου*, Εκδόσεις Ζήτη, (σελ 9-11)

βρόχου κάνει χρήση μιας προκαθορισμένης ποσότητας εξόδου όπως επίσης και ένα σήμα εισόδου αναφοράς για τον έλεγχο της διεργασίας. Η διαφορά που προκύπτει από το σήμα εξόδου και του συστήματος αναφοράς ενισχύεται και χρησιμοποιείται έτσι ώστε να ελαττώνεται διαρκώς. Τα συστήματα κλειστού βρόχου αποτελούν τη βάση για την ανάλυση και τη σχεδίαση συστημάτων αυτομάτου ελέγχου. Τα συστήματα αυτομάτου ελέγχου κλειστού βρόχου, αν και έχουν την ικανότητα να εντοπίζουν εξωτερικές επιδράσεις και να τις ελαχιστοποιούν μέσω της ανάδρασης, έχουν περίπλοκο σχεδιασμό για αυτό και η παραγωγή τους και ο σχεδιασμός τους απαιτεί υψηλό κόστος. Ορισμένα παραδείγματα συστημάτων κλειστού βρόχου είναι ο θερμοστάτης, ο κινητήρας αυτοκίνητων και τα κλιματίστικα τύπου inverter.[2]



Εικόνα 0.1 Σύστημα αυτομάτου ελέγχου ανοιχτού βρόχου



Εικόνα 0.2 Σύστημα αυτομάτου ελέγχου κλειστού βρόχου

Τα ΣΑΕ βρίσκουν εφαρμογή σε πολλούς τεχνολογικούς και επιστημονικούς τομείς όπως η ηλεκτρονική, η ρομποτική, οι τηλεπικοινωνίες και ο αυτοματισμός.[3] Παρά τις χρήσεις τους στη σημερινή εποχή, η ανάπτυξή τους προέρχεται από την αρχαιότητα με τους Βαβυλώνιους να αναπτύσσουν συστήματα αυτόματης άρδευσης. Ιδιαίτερη ανάπτυξη του τομέα παρατηρείται κατά τη διάρκεια του 19ου και του 20ου αιώνα. [4]

[2] Πετρίδης Β., 2011. *Συστήματα Αυτομάτου Ελέγχου*, Εκδόσεις Ζήτη, (σελ 17-18)
[3] Δρ. Βολογιαννίδης Σ., *Συστήματα Αυτομάτου Ελέγχου Θεωρία και Εφαρμογές, Διδακτικές Σημειώσεις Τμήματος Πληροφορικής και Επικοινωνιών*, (σελ 1)
[4] Πετρίδης Β., 2011. *Συστήματα Αυτομάτου Ελέγχου,* Εκδόσεις Ζήτη, (σελ 19-18)

Python

Η Python είναι μια διερμηνευόμενη, γενικού σκοπού, αντικειμενοστραφής γλώσσα προγραμματισμού. Ενσωματώνει μονάδες(modules), εξαιρέσεις(exceptions), δυναμικό προγραμματισμό(dynamic typing), υψηλού επιπέδου δυναμικούς τύπους δεδομένων και κλάσεις. Υποστηρίζει πολλαπλά προγραμματιστικά υποδείγματα εκτός του αντικειμενοστραφούς προγραμματισμού όπως ο διαδικαστικός (procedural programming) και ο συναρτησιακός προγραμματισμός(functional programming) .
Η ευκολία χρήσης και η αναγνωσιμότητα του κώδικα είναι τα κύρια στοιχεία της γλώσσας. Το συντακτικό της δίνει τη δυνατότητα στους προγραμματιστές να εκφράζουν έννοιες με λιγότερες γραμμές κώδικα σε σχέση με άλλες γλώσσες προγραμματισμού όπως η C++ ή η Java. Παρόλα αυτά, το γεγονός ότι είναι διερμηνευόμενη γλώσσα την καθιστά πιο αργή από τις μεταγλωττιζόμενες και για τον λόγο αυτό δεν είναι κατάλληλη για την ανάπτυξη λειτουργικών συστημάτων.
Η γλώσσα προγραμματισμού Python δημιουργήθηκε το 1989 από τον Ολλανδό Guido van Rossum και κυκλοφόρησε για πρώτη φορά το 1991. Από τότε η ραγδαία ανάπτυξή της και η δημιουργία νέων βιβλιοθηκών και πακέτων σε αυτήν, την καθιστούν πλέον μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού. Μερικές από τις χρήσεις της αφορούν τις διαδικτυακές εφαρμογές, ανάπτυξη λογισμικού, επιστημονική και αριθμητική πληροφορική, ανάπτυξη τεχνητής νοημοσύνης και μηχανική μάθηση.[5] [6]

Στόχος της εργασίας είναι η ανάπτυξη των αλγορίθμων ελέγχου με χρήση της προγραμματιστικής γλώσσας Python. Η κατανόηση των συστημάτων αυτομάτου ελέγχου και η εισαγωγή τους στο προγραμματιστικό περιβάλλον της python θα πραγματοποιηθεί χρησιμοποιώντας τα κατάλληλα εργαλεία, μεθοδολογίες και βιβλιοθήκες για τη σωστή μοντελοποίηση και επίλυσή τους μέσω παραδειγμάτων.

[5] Python, 2021, < https://el.wikipedia.org/wiki/Python >, [Πρόσβαση 28/2/2021]
[6] Python Software Foundation, 2001-2021. *General Python FAQ*
< https://docs.python.org/3/faq/general.html > [Πρόσβαση 28/2/2021]

1. Εισαγωγή στη γλώσσα Python

Όπως σε πολλές γλώσσες προγραμματισμού, έτσι και στην Python γίνεται χρήση της έννοιας της μεταβλητής. Μεταβλητή είναι ένα κομμάτι μνήμης στο οποίο αποθηκεύεται κάποια τιμή. Η γλώσσα Python έχει πολλά διαφορετικά είδη μεταβλητών για την εξυπηρέτηση των διάφορων αναγκών των προγραμματιστών. Ορισμένοι τύποι μεταβλητών είναι οι ακέραιοι αριθμοί (integers), οι αριθμοί κινητής υποδιαστολής (floats), μιγαδικοί αριθμοί (complex numbers), τα αλφαριθμητικά (strings), οι πίνακες (arrays), οι λίστες (lists), οι πλειάδες (tuples) και τα λεξικά (dictionaries). Οι τύποι μεταβλητών έχουν διαφορετικές ιδιότητες μεταξύ τους και η σωστή χρήση τους ανάλογα με τις ανάγκες που προκύπτουν θα συμβάλουν στη λύση των προβλημάτων.

1.1. Εισαγωγή Μεταβλητών

Για να ορίσουμε μεταβλητές στη γλώσσα Python ακολουθούμε κάποιους βασικούς κανόνες. Για αρχή, πρέπει να ορίσουμε ένα όνομα στη μεταβλητή και στη συνέχεια να αποδώσουμε κάποια τιμή σε αυτήν. Για παράδειγμα, για να ορίσουμε σε μια μεταβλητή με όνομα “m” την τιμή 7 θα ακολουθούσαμε την εξής διαδικασία:

```
m = 7
print(m)
```

Για να εμφανιστεί η τιμή της μεταβλητής στην κονσόλα χρησιμοποιείται η εντολή “print()” η οποία δέχεται σαν όρισμα το όνομα της μεταβλητής της οποίας θέλουμε να εμφανίσουμε. Τα ονόματα των μεταβλητών πρέπει να ξεκινούν πάντα με κάποιο γράμμα και ποτέ με αριθμό. Τα σύμβολα στα ονόματα των μεταβλητών πρέπει επίσης να αποφεύγονται. Επιπλέον, το όνομα μιας μεταβλητής δεν μπορεί να περιέχει κενά. Αντί αυτού χρησιμοποιείται είτε εναλλαγή μικρών και κεφαλαίων γραμμάτων (camelCaseExample) είτε κάτω παύλες (variable_name_example). Τέλος, η Python κάνει διάκριση πεζών-κεφαλαίων στα ονόματα μεταβλητών, δηλαδή μια μεταβλητή με όνομα “a” είναι διαφορετική από μια μεταβλητή με όνομα “A”.

Ο ορισμός πινάκων στην Python ακολουθεί παρόμοια μεθοδολογία. Για παράδειγμα για την εισαγωγή του πίνακα $array = \begin{bmatrix} 1,2,3 \\ 4,5,6 \\ 7,8,9 \end{bmatrix}$ θα ακολουθούσαμε την εξής διαδικασία:

```
import numpy as np
array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

Στην πρώτη σειρά γίνεται η εισαγωγή της βιβλιοθήκης numpy στην οποία ορίζουμε και το όνομα “np”. Έτσι, μέσω της εντολής “np.array()” της βιβλιοθήκης, ορίζουμε τον επιθυμητό πίνακα. Κάθε γραμμή του πίνακα διαχωρίζεται με αγκύλες και κόμμα ενώ κάθε στοιχείο του πίνακα πρέπει να διαχωρίζεται με κόμμα. Οι γραμμές περικλείονται και αυτές σε αγκύλες για να δηλώσουμε ότι είναι μέρος ενός πίνακα 2 διαστάσεων. Για να εμφανίσουμε το στοιχείο της δεύτερης γραμμής και δεύτερης στήλης γράφουμε την εντολή:

```
print(array[1,1])
```

Η αρίθμηση των στοιχείων σε έναν πίνακα ξεκινάει από το μηδέν, οπότε το πρώτο στοιχείο σε έναν πίνακα βρίσκεται στη θέση μηδέν, το δεύτερο στοιχείο στη θέση ένα, κτλ.

1.2. Ιδιότητες Πινάκων

Οι πίνακες αποτελούν ένα από τα σημαντικότερα εργαλεία στη μελέτη των ΣΑΕ για αυτόν το λόγο θα μελετήσουμε την υλοποίησή τους στην Python.

Για την εμφάνιση του μήκους ενός πίνακα γράφουμε την εντολή:

```
print(f"the length of the array is: {len(array)}")
```

Για την εμφάνιση των διαστάσεων ενός πίνακα σε μορφή πλειάδας:

```
print(f"Tuple of array dimensions: {array.shape}")
```

Για την εμφάνιση του αριθμού των διαστάσεων ενός πίνακα:

```
print(f"Number of array dimensions: {array.ndim}")
```

Για να εμφανίσουμε τον αριθμό των στοιχείων σε έναν πίνακα:

```
print(f"Number of elements in the array: {array.size}")
```

1.3. Δείκτες Στοιχείων Πίνακα

Όπως έχει προαναφερθεί, η αρίθμηση της θέσης των στοιχείων σε έναν πίνακα ξεκινάει από το μηδέν. Σε έναν πίνακα a = [x,y] η μεταβλητή x αναφέρεται στη γραμμή του πίνακα ενώ η μεταβλητή y στη στήλη. Έτσι, το στοιχείο της πρώτης γραμμής και δεύτερης στήλης του πίνακα "array" που ορίσαμε παραπάνω εμφανίζεται με την εντολή:

```
print(f"a[0,1] = {array[0, 1]}")
```

Σε περίπτωση που επιθυμούμε να εμφανίσουμε όλα τα στοιχεία της δεύτερης και της τρίτης γραμμής γράφουμε την εντολή:

```
print(f"a[1:3] = \n{array[1:3]}")
```

Επιπλέον, η εμφάνιση των στοιχείων της πρώτης στήλης και των δύο πρώτων γραμμών γίνεται με την εντολή:

```
print(f"a[0:2,0] = {array[0:2, 0]}")
```

Τέλος, για να ορίσουμε την τιμή "123" στη θέση [0,1] του πίνακα array γράφουμε:

```
array[0,1]=123
```

1.4. Εργαλεία Δημιουργίας Πινάκων

Η βιβλιοθήκη numpy παρέχει τους εξής τρόπους για δημιουργία πινάκων με ισαπέχοντα στοιχεία.

Η εντολή np.arrange(start, stop, step)

- start: Η τιμή από την οποία ξεκινάει ο πίνακας
- stop: Η τιμή στην οποία τελειώνει ο πίνακας, χωρίς να την περιλαμβάνει
- step: Το βήμα με το οποίο αυξάνονται οι τιμές του πίνακα

Η εντολή np.linspace(start, stop, num=50)

- start: Η τιμή από την οποία ξεκινάει ο πίνακας
- stop: Η τιμή στην οποία τελειώνει ο πίνακας
- num = 50: Το πλήθος των στοιχείων που θέλουμε να έχει ο πίνακας.
  Ο αριθμός 50 είναι η προκαθορισμένη τιμή

Για παράδειγμα, οι δημιουργία δύο πινάκων a1 = [3 4 5 6] και a2 = [1 2 3 4] μπορεί να γίνει ως εξής:

```
a1 = np.arange(3,7)
a2 = np.arange(1,5)
```

Η δημιουργία ενός πίνακα a ο οποίος προκύπτει από την ένωση του πίνακα a2 στο τέλος του πίνακα a1, σε μια νέα μεταβλητή με όνομα a ορίζεται:

```
a = np.append(a1, a2)
```

Αν θέλαμε να ενώσουμε τους πίνακες σε μια μεταβλητή b με τα στοιχεία του a2 στην πρώτη γραμμή και τα στοιχεία του a1 στη δεύτερη θα γράφαμε:

```
b = np.vstack((a2, a1))
```

1.5. Παραγωγή Ειδικών Πινάκων

Για τη δημιουργία τυχαίων αριθμών χρειάζεται η εισαγωγή του πακέτου random από τη βιβλιοθήκη numpy και να κατασκευάσουμε μια γεννήτρια τυχαίων αριθμών.

```
from numpy import random
rng = np.random.default_rng(12345)
```

Με τη χρήση του πακέτου random γίνεται δυνατή η δημιουργία πινάκων με τυχαία ορίσματα.

Για να ορίσουμε έναν πίνακα rng1 (2×4) τυχαίων πραγματικών αριθμών στο διάστημα [0,1) γράφουμε:

```
rng1 = rng.random((2, 4))
```

Η για έναν τετραγωνικό πίνακα rng2 (3×3) τυχαίων πραγματικών αριθμών στο διάστημα [0,1) δίνουμε την εντολή:

```
rng2 = rng.random((3, 3))
```

Ένας πίνακας rng3 (3×5) με τυχαίους ακέραιους αριθμούς στο διάστημα [50,100) οριζεται:

```
rng3 = random.randint(50,100, size=(3, 5))
```

Μεγάλη χρησιμότητα έχουν οι μοναδιαίοι και οι μηδενικοί πίνακες. Η δημιουργία τους ακολουθεί την εξής μεθοδολογία

Η δημιουργία ενός πίνακα zeros1 και ενός πίνακα ones1 (2×4) διαστάσεων γίνεται γράφοντας:

```
zeros1 = np.zeros((2, 4))
ones1 = np.ones((2, 4))
```

1.6. Πράξεις Πινάκων

Για να κάνουμε ορισμένες πράξεις πινάκων θα χρειαστεί η εισαγωγή των πακέτων matrix_power, inv, det για ύψωση πινάκων σε δύναμη, αντιστροφή πινάκων και υπολογισμό ορίζουσας πινάκων αντίστοιχα.

```
from numpy.linalg import matrix_power, inv, det
```

Για τα παραδείγματα που θα ακολουθήσουν θα ορίσουμε δύο νέους πίνακες, τον $x=\begin{bmatrix}1,2\\4,5\end{bmatrix}$ και τον $y=\begin{bmatrix}7,8\\9,10\end{bmatrix}$

```
x = np.array([[1, 2], [4, 5]])
y = np.array([[7, 8], [9, 10]])
```

Το άθροισμα πινάκων με κοινές διαστάσεις μπορεί να γίνει:

```
print(np.add(x,y))
```

Το μαθηματικό γινόμενο πινάκων AB ορίζεται:

```
print(np.dot(x, y))
```

Ενώ το γινόμενο πινάκων με κοινές διαστάσεις ανά στοιχείο γίνεται:

```
print(np.multiply(x, y))
```

Η διαίρεση πινάκων με κοινές διαστάσεις ανά στοιχείο γίνεται:

```
print(np.divide(x, y))
```

Το γινόμενο πινάκων $A(AA)$ υπολογίζεται ως εξής:

```
print(matrix_power(x, 3))
```

Πρέπει να επισημανθεί ότι η εντολή matrix_power() λειτουργεί τετραγωνικούς πίνακες

Ένας πίνακας με στοιχεία x_{ij}^{3} υπολογίζεται:

```
print(np.power(x, 3))
```

Ο υπολογισμός πίνακα όπου κάθε στοιχείο του είναι $x_{ij}^{y_{ij}}$ γίνεται:

```
print(np.power(x, y))
```

Για τον υπολογισμό αντίστροφου πίνακα $\left(A^{-1}\right)$:

```
print(inv(x))
```

Η ορίζουσα πίνακα $\left(|A|\right)$ γίνεται:

```
print(det(x))
```

Η απόλυτη τιμή ανά στοιχείο μπορεί να υπολογιστεί:

```
print(abs(x))
```

Με τις εντολές np.amax() και np.amin() μπορούμε να βρούμε τα ελάχιστα ή τα μέγιστα στοιχεία ανά γραμμή ή στήλη. Για παράδειγμα τα μέγιστα στοιχεία ανά γραμμή του πίνακα x υπολογίζονται:

```
print(np.amax(x, axis=0))
```

Ενώ τα ελάχιστα στοιχεία του πίνακα x ανά στήλη υπολογίζονται:

```
print(np.amin(x, axis=1))
```

Με την εντολή np.sum() μπορεί να βρεθεί το διάνυσμα με στοιχεία τα αθροίσματα ανά γραμμή ή στήλη γράφοντας:

```
sum_horizontal = np.sum(x, axis=0)
sum_vertical = np.sum(x, axis=1)
```

ενώ με την εντολή np.prod() μπορεί να βρεθεί το διάνυσμα με στοιχεία τα γινόμενα ανά γραμμή ή στήλη γράφοντας:

```
prod_horizontal = np.prod(x, axis=0)
prod_vertical = np.prod(ar1, axis=1)
```

Τέλος, για να βρεθεί το μέγιστο ή το ελάχιστο στοιχείο ενός πίνακα γίνεται χρήση των εντολών np.max() και np.min() αντίστοιχα:

```
print(np.max(x))
print(np.min(x))
```

Παράδειγμα, έστω οι πίνακες $A=\begin{pmatrix}3 & -7 & 5\\ 1 & 0 & 0\\ 8 & 3 & 1\end{pmatrix}$ και $B=\begin{pmatrix}4 & 7 & 6\\ 0 & -4 & 1\\ 5 & -8 & 3\end{pmatrix}$. Για να γίνει υπολογισμός του αθροίσματος, της αφαίρεσης του γινομένου, της διαίρεσης όπως επίσης και οι πράξεις AB^{-1}, $A^{-1}B$, $|A^{-1}|BA$, $|B|B^{-1}A$ θα ακολουθήσουμε τα εξής βήματα:

```
A = np.array([[3, -7, 5], [1, 0, 0], [8, 3, 1]])
B = np.array([[4, 7, 6], [0, -4, 1], [5, -8, 3]])
print(f"Πινακας A: \n{A}")
print(f"Πινακας B: \n{B}")
```

```
Πινακας A:
[[ 3 -7  5]
 [ 1  0  0]
 [ 8  3  1]]
Πινακας B:
[[ 4  7  6]
 [ 0 -4  1]
 [ 5 -8  3]]
```

```
print('A+B = \n', np.add(A, B), '\n')
print('A-B = \n', np.subtract(A, B), '\n')
print('A*B (element wise) = \n', np.multiply(A, B), '\n')
print('A/B (element wise) = \n', np.divide(A, B))
```

```
A+B =
[[ 7  0 11]
 [ 1 -4  1]
 [13 -5  4]]

A-B =
 [[ -1 -14  -1]
 [  1   4  -1]
 [  3  11  -2]]

A*B (element wise) =
 [[ 12 -49  30]
 [  0   0   0]
 [ 40 -24   3]]

A/B (element wise) =
 [[ 0.75       -1.          0.83333333]
 [       inf -0.          0.        ]
 [ 1.6        -0.375       0.33333333]]
```

Η μαθηματική πράξη AB^{-1}

```
C1 = np.dot(A, inv(B))
print(C1)
```

```
[[ 0.38129496  1.82733813  0.29496403]
 [-0.02877698 -0.49640288  0.22302158]
 [ 0.02158273 -3.87769784  1.58273381]]
```

Η μαθηματική πράξη $A^{-1}B$

```
C2 = np.dot(inv(A), B)
print(C2)
```

```
 [[ 1.73472348e-17 -4.00000000e+00  1.00000000e+00]
 [ 9.54545455e-01  4.59090909e+00 -1.27272727e+00]
 [ 2.13636364e+00  1.02272727e+01 -1.18181818e+00]]
```

Η μαθηματική πράξη $|A|BA$

```
C3a = np.dot(B,A)
C3 = np.dot(det(inv(A)), C3a)
print(C3)
```

```
[[ 3.04545455 -0.45454545  1.18181818]
 [ 0.18181818  0.13636364  0.04545455]
 [ 1.40909091 -1.18181818  1.27272727]]
```

Η μαθηματική πράξη $|B|B^{-1}A$

```
C4a = np.dot(inv(B), A)
C4 = np.dot(det(B), C4a)
print(C4)
```

```
[[ 167.  121.   11.]
 [ -35.  -47.   21.]
 [  -1. -188.   84.]]
```

1.7. Πολυώνυμα

Τα πολυώνυμα θα χρησιμοποιηθούν για την περιγραφή συστημάτων.

Για την εισαγωγή πολυωνύμων στην Python θα χρειαστεί να εισάγουμε το πακέτο polynomial από τη βιβλιοθήκη numpy.

```
from numpy.polynomial import Polynomial as P
```

Στη συνέχεια εισάγουμε δύο πολυώνυμα το $P_1(x)=2x3−x2+5x−6$ και το $P_2(x)=2x3+x$ και τα τυπώνουμε στην κονσόλα:

```
p1 = P([2, -1, 5, -6])
p2 = P([2, 0, 1, 0])
print(p1)
print(p2)
```

Για τις πράξεις μεταξύ πολυωνύμων είναι προτιμότερο να τα ορίζουμε με πίνακες και να κάνουμε χρήση των ειδικών πράξεων για πολυώνυμα που παρέχει η βιβλιοθήκη numpy. Για παράδειγμα ο ορισμός των παραπάνω πολυωνύμων μπορεί να γίνει

```
p1 = np.array([2, -1, 5, -6])
p2 = np.array([2, 0, 1, 0])
```

Για την πρόσθεση των δύο πολυωνύμων κάνουμε χρήση της εντολής np.polyadd()

```
print(np.polyadd(p1, p2))
```

Ο πολλαπλασιασμός πολυωνύμων γίνεται με την εντολη np.polymul() και γράφεται:

```
print(np.polymul(p1, p2))
```

Η διαίρεση πολυωνύμων γίνεται με την εντολή np.polydiv(). Η εντολή επιστρέφει το πηλίκο και το υπόλοιπο της διαίρεσης σε μορφή πινάκων.

```
print(np.polydiv(p1, p2))
```

Ο υπολογισμός των ριζών ενός πολυωνύμου μπορεί να υπολογιστεί με την εντολή np.roots(). Επιστρέφεται ένα διάνυσμα με πραγματικό και φανταστικό μέρος της λύσης. Για παράδειγμα, ορίζουμε μια μεταβλητή “r” με τις ρίζες του πολυώνυμου p1:

```
r = np.roots(p1)
print(r)
```

Είναι δυνατή η κατασκευή πολυωνύμου με τα στοιχεία του διανύσματος r με την εντολή np.poly().

```
poly = np.poly(r)
```

Ο υπολογισμός της παραγώγου ενός πολυωνύμου γίνεται με την εντολή np.polyder(). Για παράδειγμα η παράγωγος του p1 μπορεί να υπολογιστεί και να εκχωρηθεί σε μια μεταβλητή με όνομα “Dpoly” γράφοντας:

```
Dpoly = np.polyder(p1)
```

Έστω τα δύο πολυώνυμα $p(x) = 3x^4 + 5x^2 - x - 8$ και $q(x) = 2x^2 - x + 3$. Θέλουμε να βρεθούν οι ρίζες και οι παράγωγοι των πολυωνύμων όπως επίσης και οι ρίζες και οι παράγωγοι του αθροίσματος και του γινομένου τους.

Εισάγουμε τα πολυώνυμα

```
p = np.array([3, 0, 5, -1, -8])
q = np.array([2, -1, 3])
```

Υπολογίσουμε τις ρίζες τους

```
print('roots(p) = \n', np.roots(p), '\n')
print('roots(q) = \n', np.roots(q))
```

```
roots(p) =
 [-0.04540803+1.63419909j -0.04540803-1.63419909j
1.04531614+0.j
 -0.95450008+0.j        ]

roots(q) =
 [0.25+1.19895788j 0.25-1.19895788j]
```

Υπολογίζουμε τις παραγώγους των πολυωνύμων

```
print('D(p) = \n', np.polyder(p), '\n')
print('D(q) = \n', np.polyder(q))
```

```
D(p) =
 [12  0 10 -1]

D(q) =
 [ 4 -1]
```

Υπολογίζουμε τις ρίζες του αθροίσματος

```
sumpq = np.polyadd(p, q)
print('roots(p+q) = \n', np.roots(sumpq))
```

```
roots(p+q) =
 [-0.09511665+1.71112484j -0.09511665-1.71112484j
0.85440615+0.j
 -0.66417285+0.j
```

Η παράγωγος του αθροίσματος

```
sumpq = np.polyadd(p, q)
print('D(p+q) = \n', np.polyder(sumpq))
```

```
D(p+q) =
 [12  0 14 -2]
```

Οι ρίζες του γινομένου

```
prodpq = np.polymul(p, q)
print('roots(p*q) = \n', np.roots(prodpq))
```

```
roots(p*q) =
 [-0.04540803+1.63419909j -0.04540803-1.63419909j
1.04531614+0.j
  0.25      +1.19895788j  0.25      -1.19895788j -
0.95450008+0.j        ]
```

Η παράγωγος του γινομένου

```
print('D(p*q) = \n', np.polyder(prodpq))
```

```
D(p*q) =
 [ 36 -15  76 -21   0   5]
```

1.8. Γραφικές Παραστάσεις

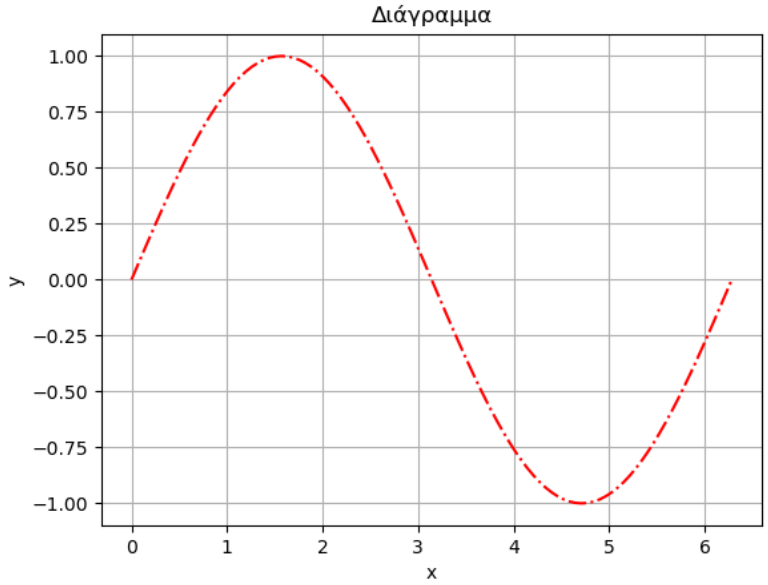
Οι γραφικές παραστάσεις αποτελούν σημαντικό εργαλείο στη μελέτη των συστημάτων αυτομάτου ελέγχου και θα χρησιμοποιούνται σε μεγάλο βαθμό για την κατανόηση και τη δημιουργία τους. Η αναπαράσταση γραφικών παραστάσεων στην Python γίνεται μέσω της βιβλιοθήκης matplotlib και του πακέτου της pyplot. Η εισαγωγή τους στο προγραμματιστικό περιβάλλον μας γίνεται ως εξής:

```
import matplotlib.pyplot as plt
```

Η εντολή **plot()** σχεδιάζει τη γραφική παράσταση μιας ή περισσότερων συναρτήσεων με τη βοήθεια πινάκων στους οποίους αποθηκεύουμε διακριτές τιμές. Στην μορφή αυτή της **plot()** σχεδιάζουμε το y συναρτήσει του x_i i=1..n, πάνω στους ίδιους άξονες. Το y_i πρέπει να είναι διάνυσμα ή πίνακας της ίδιας διάστασης με το x_i (οι y_i , x_i μπορεί να είναι απλές μεταβλητές, οπότε η γραφική παράσταση θα είναι ένα σημείο). Με τις επιλογές καθορίζουμε το χρώμα και το σύμβολο σχεδίασης της γραφικής παράστασης. Η μορφή των επιλογών είναι: 'cs' , όπου c το πρώτο γράμμα του χρώματος, π.χ. r για το red, g για το green, b για το blue, k για το black κ.τ.λ., και s για το σύμβολο σχεδίασης, π.χ. 'o', '+', 'x', '*', '-' κ.λ.π.

Παραδείγματος χάρη, η γραφική παράσταση του ημιτόνου γίνεται ως εξής:

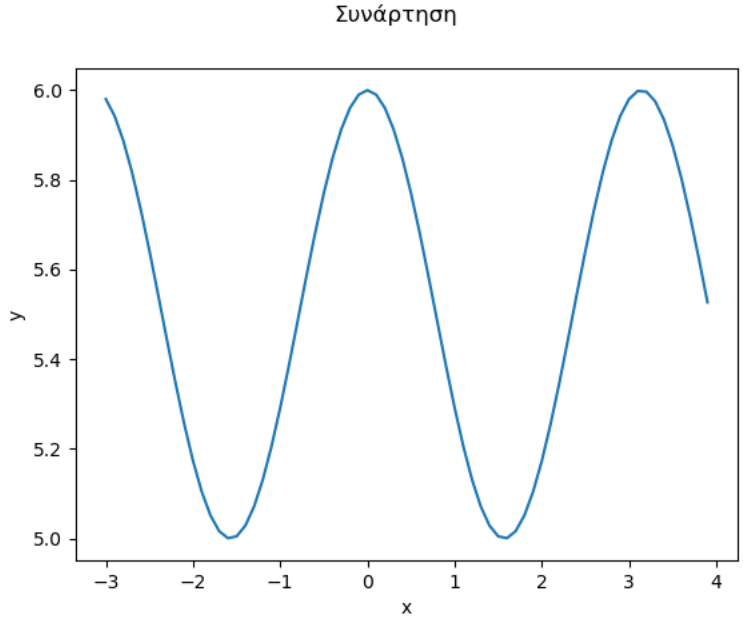
```
x = np.linspace(0, 2*np.pi, 101)
y = np.sin(x)
plt.plot(x, y, color='r', linestyle='-.')
plt.title('Διάγραμμα')   #Τίτλος του διαγράμματος
plt.ylabel('y')   #Τίτλος του άξονα y
plt.xlabel('x')   #Τίτλος του άξονα x
plt.grid(True)   #Ορισμός εμφάνισης πλέγματος
plt.show()   #Εντολή εμφάνισης του διαγράμματος
```



Εικόνα 1.0.1 Γραφική παράσταση ημιτόνου

Αν για παράδειγμα χρειαζόμασταν τη γραφική παράσταση της συνάρτησης $y = \cos(2x) + \sin^2(x) + 5$ στο διάστημα $D_f = (-3,4)$ θα γράφαμε:

```
x = np.arange(-3,4,0.1)
y = np.cos(2*x) + np.power(np.sin(x), 2) + 5

plt.plot(x,y)
plt.suptitle('Συνάρτηση')
plt.ylabel('y')
plt.xlabel('x')
plt.show()
```



Εικόνα 1.0.2 Γραφική παράσταση συνάρτησης y=f(x)

2. Σήματα και Συστήματα

Οι έννοιες του σήματος και του συστήματος συναντώνται σε πολλές τεχνολογικές επιστήμες και βρίσκουν εφαρμογή σε τομείς όπως οι τηλεπικοινωνίες, η επεξεργασία εικόνας, βίντεο και ήχου, η συμπίεση δεδομένων κλπ.[7]

Σήμα ορίζεται μια μαθηματική συνάρτηση μιας ή περισσότερων ανεξάρτητων μεταβλητών (μια από τις οποίες είναι υποχρεωτικά ο χρόνος) και τυπικά περιέχει πληροφορίες για τη χρονική εξέλιξη μιας ποσότητας η οποία περιγράφει ένα φαινόμενο ή διαδικασία.

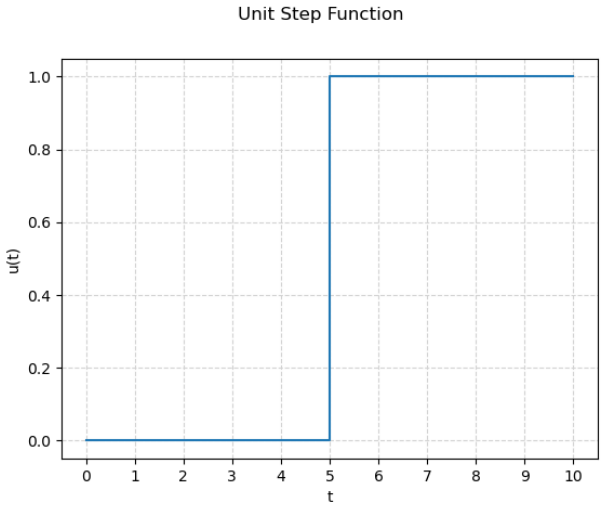
Όπως έχει προαναφερθεί, *σύστημα* ορίζεται ένα μέρος του φυσικού κόσμου το οποίο θεωρούμε ότι αποτελείται από ένα σύνολο στοιχείων τα οποία λειτουργούν συγχρόνως κατά προδιαγεγραμμένο τρόπο έτσι ώστε να επιτυγχάνεται κάποιος στόχος. Ένα σύστημα επικοινωνεί με το περιβάλλον μέσω σημάτων. Τα σήματα που δέχεται ένα σύστημα ονομάζονται διεγέρσεις ή είσοδοι και τα σήματα που παράγει ένα σύστημα λόγω των διεγέρσεων και των μη μηδενικών αρχικών συνθηκών ονομάζονται αποκρίσεις ή έξοδοι.

[7]Wikipedia, 2017. *Επεξεργασία σήματος* , < https://el.wikipedia.org/wiki/Επεξεργασία_σήματος > [Πρόσβαση 08/02/2021]

2.1. Βασικά Σήματα

2.1.1. Μοναδιαία συνάρτηση βαθμίδας (unit step function)

$$f(t)=\begin{cases}1 & \text{για} \quad t>t_0\\ 0 & \text{για} \quad t<t_0\end{cases}$$



Εικόνα 2.1 Διάγραμμα βηματικής συνάρτησης

1ος Τρόπος

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([0, 5, 10])
y = np.array([0, 0, 1])
plt.ylabel('u(t)')
plt.xlabel('t')
plt.suptitle('Unit Step Function')
plt.grid(True, linestyle='--', color='lightgrey')
step = plt.step(x, y)
plt.xticks(np.arange(0, 11, 1))
plt.show()
```

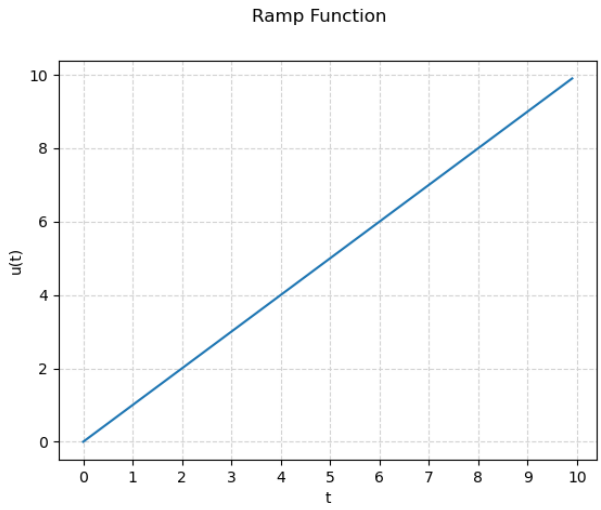
2ος Τρόπος

```
import numpy as np
import matplotlib.pyplot as plt

x = np.zeros(50)
y = np.ones(50)
step = np.append(x,y)
t = np.arange(0, 10, 0.1)
plt.ylabel('u(t)')
plt.xlabel('t')
plt.suptitle('Unit Step Function')
plt.grid(True, linestyle='--', color='lightgrey')
stepfig = plt.step(t,step)
```

```
plt.xticks(np.arange(0, 11, 1))
plt.show()
```

2.1.2. Συνάρτηση ράμπας (Ramp function)

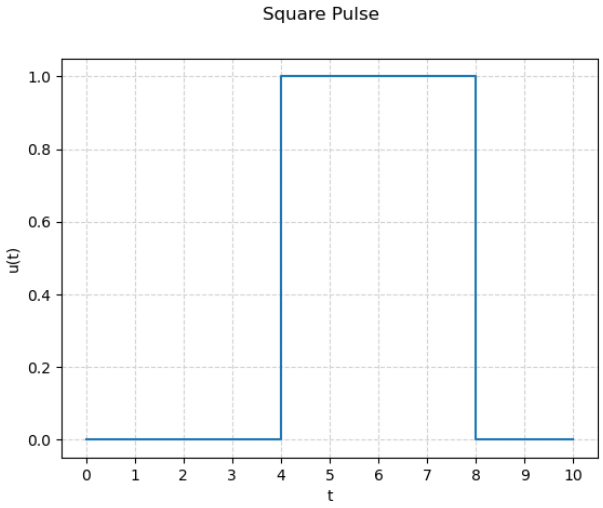
$$f(t) = t$$



Εικόνα 2.2 Διάγραμμα συνάρτησης ράμπας

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 10, 0.1)
y = np.ones(len(x))
ramp_function = np.multiply(x, y)
plt.plot(x, ramp_function)
plt.ylabel('u(t)')
plt.xlabel('t')
plt.suptitle('Ramp Function')
plt.grid(True, linestyle='--', color='lightgrey')
plt.xticks(np.arange(0, 11, 1))
plt.show()
```

2.1.3. Τετραγωνικός παλμός (Square Pulse)



Εικόνα 2.3 Διάγραμμα τετραγωνικού παλμού

1ος Τρόπος

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 4, 8, 10])
y = np.array([0, 0, 1, 0])

plt.ylabel('u(t)')
plt.xlabel('t')
plt.suptitle('Square Pulse')
plt.grid(True, linestyle='--', color='lightgrey')
square = plt.step(x, y)
plt.xticks(np.arange(0, 11, 1))
plt.show()
```

2ος Τρόπος

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)

l1 = len(np.linspace(0, 4, 40, endpoint=False))
l2 = len(np.linspace(4, 8, 40, endpoint=False))
l3 = len(np.linspace(8, 10, 20))
y1 = np.zeros(l1)
y2 = np.ones(l2)
y3 = np.zeros(l3)
y12 = np.append(y1, y2)
y = np.append(y12, y3)
plt.ylabel('u(t)')
plt.xlabel('t')
plt.suptitle('Square Pulse')
```

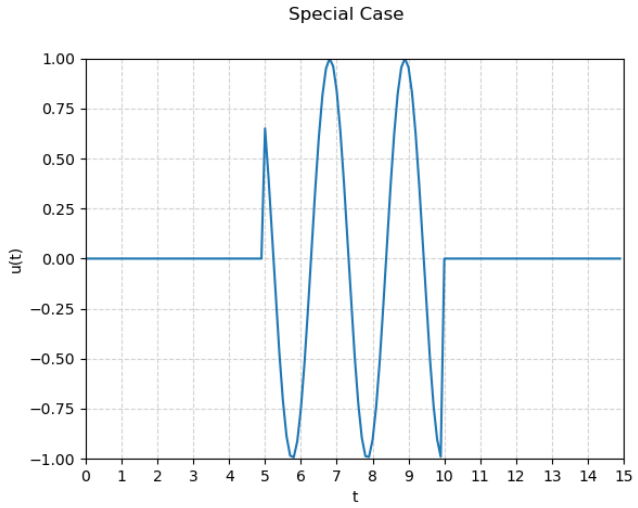
```
plt.grid(True, linestyle='--', color='lightgrey')
square = plt.step(x, y)
plt.xticks(np.arange(0, 11, 1))
plt.show()
```

2.2. Ειδικοί Τύποι Σημάτων

Για να απομονώσουμε μια συνάρτηση σε χρονικό διάστημα πχ.

$$f(t)=\begin{cases}\sin(t) & \text{για} \quad t\in[t_1,t_2]\\ 0 & \text{για} \quad t\notin[t_1,t_2]\end{cases}$$

ακολουθήσουμε ένα από τα παρακάτω σύνολα εντολών



Εικόνα 2.4 Διάγραμμα ειδικού σήματος

1ος Τρόπος

```
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0, 15, 0.1)
r = np.arange(5, 10, 0.1)
u1 = np.zeros(50)
u2 = np.sin(3 * r)
u3 = np.zeros(50)

u12 = np.append(u1, u2)
u = np.append(u12, u3)

plt.ylabel('u(t)')
```

```
plt.xlabel('t')
plt.suptitle('Special Case')
plt.grid(True, linestyle='--', color='lightgrey')
plt.axis([0, 15, -1, 1])
plt.xticks(np.arange(0, 16, 1))
plt.plot(t, u)
plt.show()
```

2ος Τρόπος

```
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0, 15, 0.1)
r = np.arange(5, 10, 0.1)

u1 = np.zeros(50)
u2 = np.ones(50)
u3 = np.zeros(50)

u12 = np.append(u1, u2)
u = np.append(u12, u3)

s = np.multiply(np.sin(3*t), u)

plt.ylabel('u(t)')
plt.xlabel('t')
plt.suptitle('Special Case')
plt.grid(True, linestyle='--', color='lightgrey')
plt.axis([0, 15, -1, 1])
plt.xticks(np.arange(0, 16, 1))
plt.plot(t, s)
plt.show()
```

2.3. Γεννήτριες Σημάτων

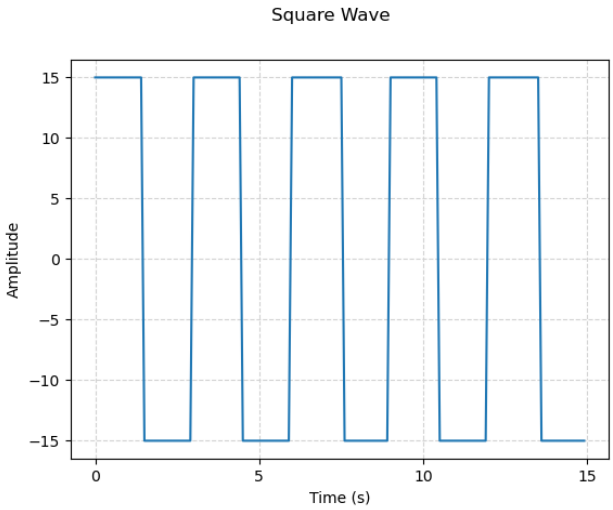
Για την παραγωγή ορισμένων ειδικών σημάτων θα χρειαστεί να εισάγουμε το πακέτο signal από τη βιβλιοθήκη scipy

```
from scipy import signal as sg
```

2.3.1. Τετραγωνικός παλμός (Square Pulse)

Η παραγωγή του παλμού γίνεται με τη χρήση της εντολής sg.square(t, duty=0.5)

Όπου t ορίζεται η είσοδος χρόνου και duty ορίζεται ο κύκλος καθήκοντος (duty cycle) ο οποίος έχει προκαθορισμένη τιμή το 0.5

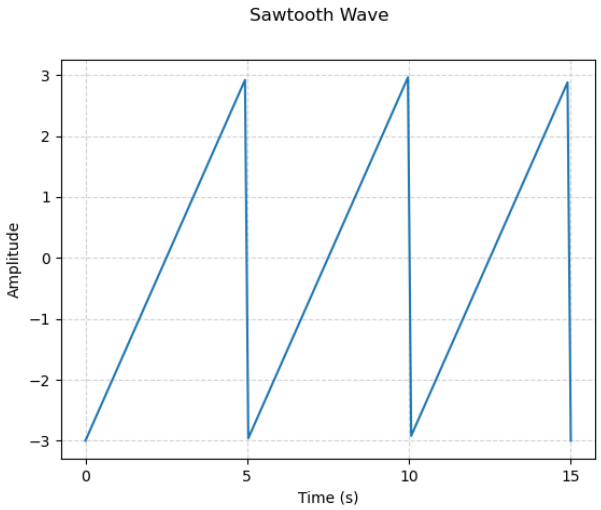


Εικόνα 2.5 Διάγραμμα τετραγωνικού παλμού

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal as sg

period = 3
frequency = 1 / period
amplitude = 15
time = np.arange(0, 15, 0.1)
signal_square = amplitude * sg.square(2 * np.pi * frequency *
time)
plt.plot(time, signal_square)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.suptitle('Square Wave')
plt.grid(True, linestyle='--', color='lightgrey')
plt.xticks(np.arange(0, 16, 5))
plt.show()
```

2.3.2. Οδοντωτός Παλμός (Sawtooth Wave)

Με αντιστοιχο τρόπο συντάσεται και η εντολή sg.sawtooth(t, width=1). Η διαφορά είναι στο δεύτερο όρισμα width, το οποίο είναι το πλάτος της αύξουσας ράμπας σαν ποσοστό του συνολικού κύκλου. Η προκαθορισμένη τιμή είναι 1 και παράγει μια αύξουσα ράμπα, ενώ για τιμή 0 παράγει μια φθίνουσα ράμπα.

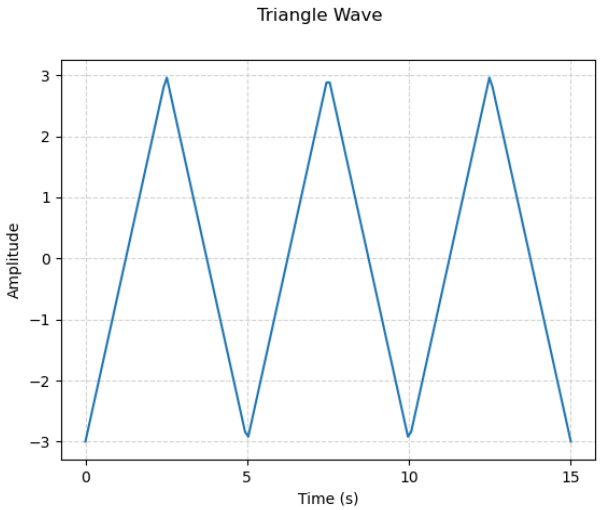


Εικόνα 2.6 Διάγραμμα οδοντωτού παλμού

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal as sg

period = 5
frequency = 1 / period
amplitude = 3
time = np.linspace(0, 15, 150)
signal_saw = amplitude * sg.sawtooth(2 * np.pi * frequency *
time)
plt.plot(time, signal_saw)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.suptitle('Sawtooth Wave')
plt.grid(True, linestyle='--', color='lightgrey')
plt.xticks(np.linspace(0, 15, 4))
plt.show()
```

2.3.3. Τριγωνικός παλμός (Triangle Wave)

Ο τριγωνικός παλμός χρησημοποιεί τη συνάρτηση sg.sawtooth() με όρισμα width=0.5



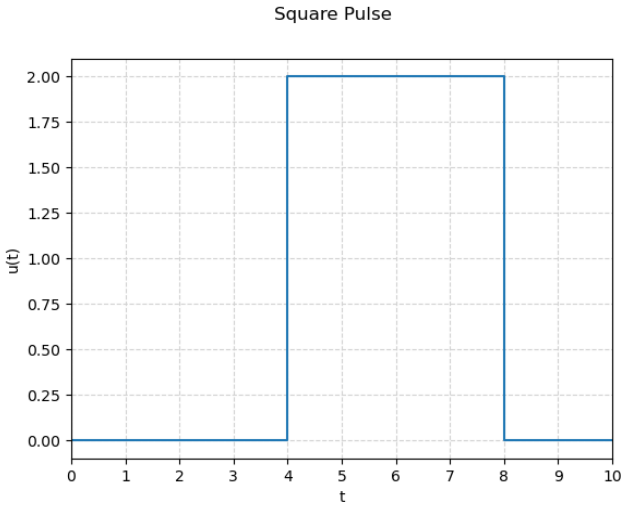
Εικόνα 2.7 Διάγραμμα τριγωνικού παλμού

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal as sg

period = 5
frequency = 1 / period
amplitude = 3
time = np.linspace(0, 15, 150)
signal_triangle = amplitude * sg.sawtooth(2 * np.pi * frequency *
time, width=0.5)
plt.plot(time, signal_triangle)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.suptitle('Triangle Wave')
plt.grid(True, linestyle='--', color='lightgrey')
plt.xticks(np.linspace(0, 15, 4))
plt.show()
```

2.4. Παραδείγματα Γραφικών Παραστάσεων

Παράδειγμα 1

Για να κατασκευάσουμε το παρακάτω τετραγωνικό παλμό θα γράψουμε

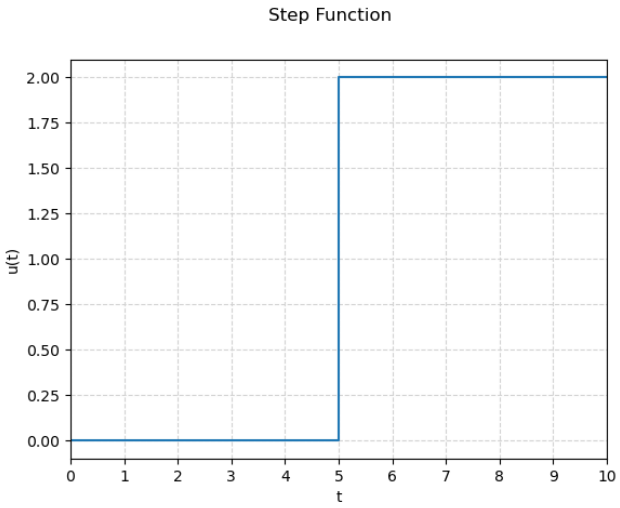


Εικόνα 2.8 Παράδειγμα 1

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 4, 8, 10])
y = np.array([0, 0, 2, 0])

plt.ylabel('u(t)')
plt.xlabel('t')
plt.suptitle('Square Pulse')
plt.grid(True, linestyle='--', color='lightgrey')
square = plt.step(x, y)
plt.xticks(np.arange(0, 11, 1))
plt.xlim(0, 10)
plt.show()
```

Παράδειγμα 2

Για να κατασκευάσουμε το παρακάτω διάγραμμα βηματικής συνάρτησης θα γράψουμε

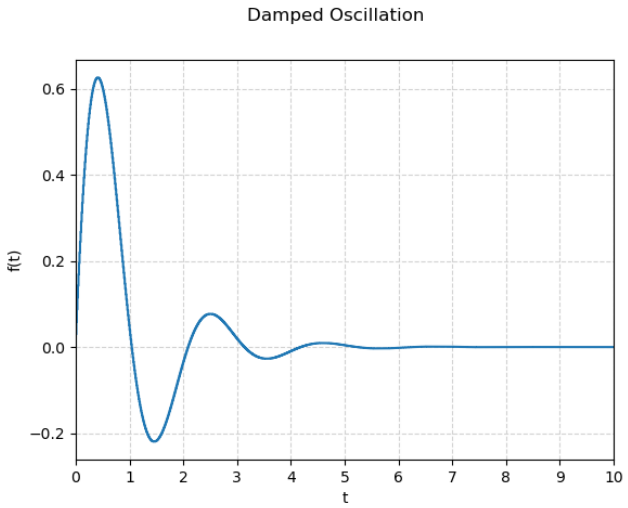


Εικόνα 2.9 Παραδειγμα 2

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([0, 5, 10])
y = np.array([0, 0, 2])
plt.ylabel('u(t)')
plt.xlabel('t')
plt.suptitle('Step Function')
plt.grid(True, linestyle='--', color='lightgrey')
step = plt.step(x, y)
plt.xticks(np.arange(0, 11, 1))
plt.xlim(0, 10)
plt.show()
```

Παράδειγμα 3

Για να κατασκευάσουμε το παρακάτω διάγραμμα φθίνουσας ταλάντωσης γράφουμε

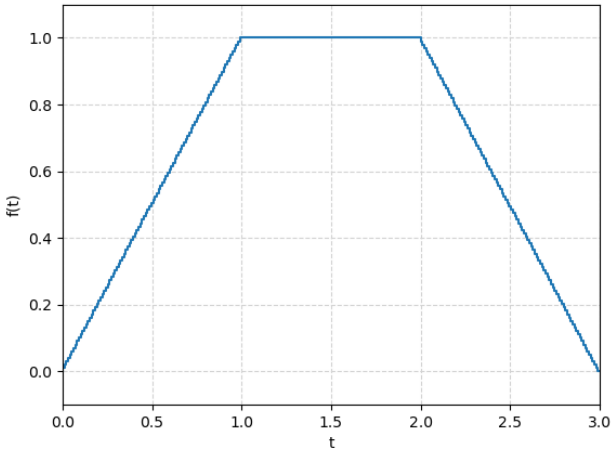


Εικόνα 2.10 Παραδειγμα 3

```
import matplotlib.pyplot as plt
import numpy as np

t = np.linspace(0, 10, 1000)
f = np.exp(-t) * np.sin(3 * t)

plt.ylabel('f(t)')
plt.xlabel('t')
plt.suptitle('Damped Oscillation')
plt.grid(True, linestyle='--', color='lightgrey')
step = plt.step(t, f)
plt.xticks(np.arange(0, 11, 1))
plt.xlim(0, 10)
plt.show()
```

Παράδειγμα 4

Για να κατασκευάσουμε το παρακάτω διάγραμμα γράφουμε

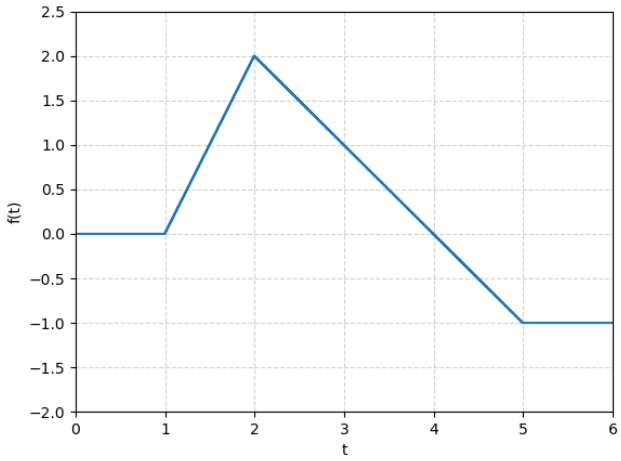


Εικόνα 2.11 Παραδειγμα 4

```
import matplotlib.pyplot as plt
import numpy as np

t1 = np.linspace(0, 1, 100)
t2 = np.linspace(1, 2, 100)
t3 = np.linspace(2, 3, 100)
y1 = t1
y2 = np.ones(len(t2))
y3 = 3-t3
y12 = np.append(y1, y2)
y = np.append(y12, y3)
t12 = np.append(t1, t2)
t = np.append(t12, t3)

plt.ylabel('f(t)')
plt.xlabel('t')
plt.grid(True, linestyle='--', color='lightgrey')
step = plt.step(t, y)
plt.xlim(0, 3)
plt.ylim(-0.1, 1.1)
plt.show()
```

Παράδειγμα 5

Για να κατασκευάσουμε το παρακάτω διάγραμμα γράφουμε

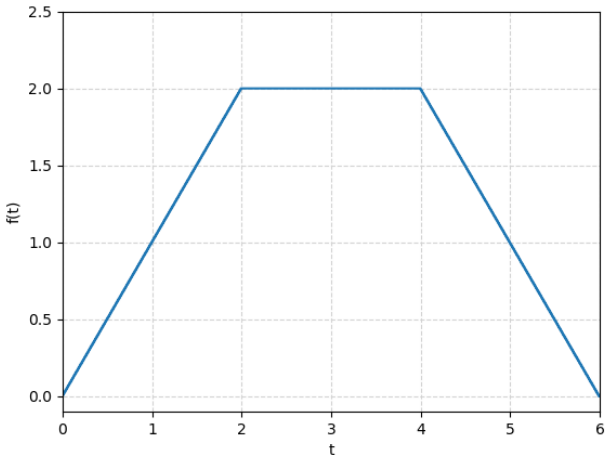


Εικόνα 2.12 Παραδειγμα 5

```
import matplotlib.pyplot as plt
import numpy as np

t1 = np.linspace(0, 1, 100)
t2 = np.linspace(1, 2, 100)
t3 = np.linspace(2, 5, 300)
t4 = np.linspace(5, 6, 100)

y1 = np.zeros(len(t1))
y2 = -2+2*t2
y3 = 4 - t3
y4 = np.ones(len(t4))* -1

t = np.append(np.append(t1, t2), np.append(t3, t4))
y = np.append(np.append(y1, y2), np.append(y3, y4))

plt.ylabel('f(t)')
plt.xlabel('t')
plt.grid(True, linestyle='--', color='lightgrey')
step = plt.step(t, y)
plt.xlim(0, 6)
plt.ylim(-2, 2.5)
plt.show()
```

Παράδειγμα 6

Για να κατασκευάσουμε το παρακάτω διάγραμμα γράφουμε

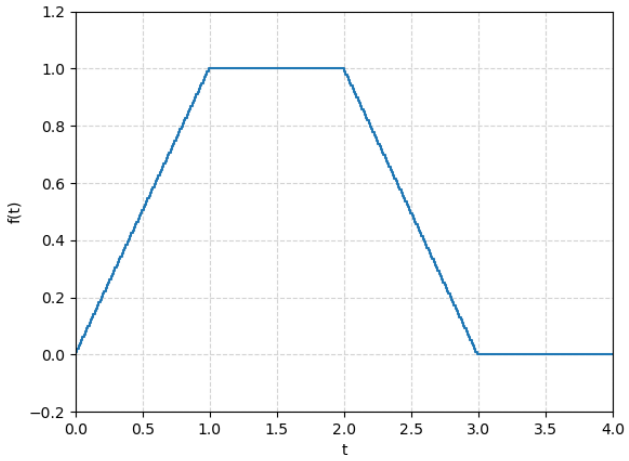


Εικόνα 2.13 Παραδειγμα 6

```
import matplotlib.pyplot as plt
import numpy as np

t1 = np.linspace(0, 2, 200)
t2 = np.linspace(2, 4, 200)
t3 = np.linspace(4, 6, 200)
y1 = t1
y2 = np.ones(len(t2)) * 2
y3 = 6-t3
y12 = np.append(y1, y2)
y = np.append(y12, y3)
t12 = np.append(t1, t2)
t = np.append(t12, t3)

plt.ylabel('f(t)')
plt.xlabel('t')
plt.grid(True, linestyle='--', color='lightgrey')
step = plt.step(t, y)
plt.xlim(0, 6)
plt.ylim(-0.1, 2.5)
plt.show()
```

Παράδειγμα 7

Για να κατασκευάσουμε το παρακάτω διάγραμμα γράφουμε

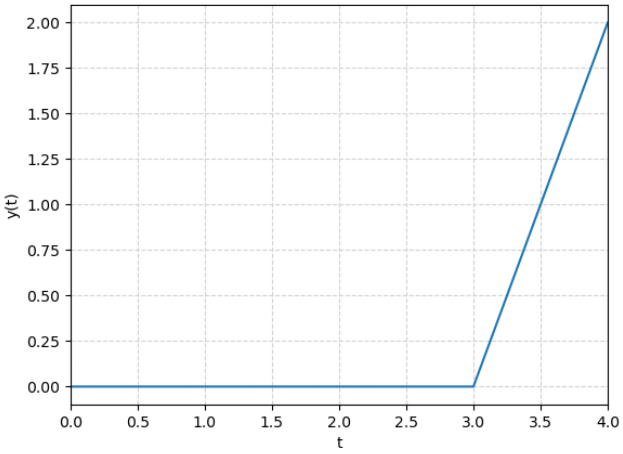


Εικόνα 2.14 Παραδειγμα 7

```
import matplotlib.pyplot as plt
import numpy as np

t1 = np.linspace(0, 1, 100)
t2 = np.linspace(1, 2, 100)
t3 = np.linspace(2, 3, 100)
t4 = np.linspace(3, 4, 100)

y1 = t1
y2 = np.ones(len(t2))
y3 = 3 - t3
y4 = np.zeros(len(t4))

y = np.append(np.append(y1, y2), np.append(y3, y4))
t = np.append(np.append(t1, t2), np.append(t3, t4))

plt.ylabel('f(t)')
plt.xlabel('t')
plt.grid(True, linestyle='--', color='lightgrey')
step = plt.step(t, y)
plt.xlim(0, 4)
plt.ylim(-0.2, 1.2)
plt.show()
```

Παράδειγμα 8

Για να κατασκευάσουμε το παρακάτω διάγραμμα γράφουμε

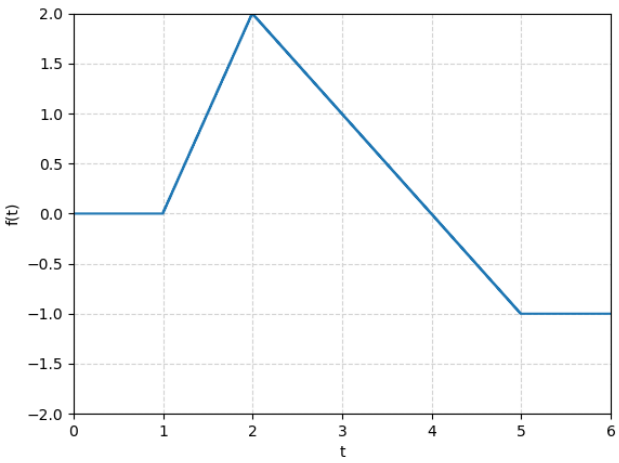


Εικόνα 2.15 Παραδειγμα 8

```
import matplotlib.pyplot as plt
import numpy as np

t1 = np.linspace(0, 3, 300)
t2 = np.linspace(3, 4, 100)

y1 = np.zeros(len(t1))
y2 = 2 *(t2-3)

t = np.append(t1, t2)
y = np.append(y1, y2)

plt.plot(t, y)
plt.ylabel('y(t)')
plt.xlabel('t')
plt.grid(True, linestyle='--', color='lightgrey')
plt.xlim(0, 4)
plt.show()
```

Παράδειγμα 9

Για να κατασκευάσουμε το παρακάτω διάγραμμα γράφουμε



Εικόνα 2.16 Παραδειγμα 9

```
import matplotlib.pyplot as plt
import numpy as np

t1 = np.linspace(0, 1, 100)
t2 = np.linspace(1, 2, 100)
t3 = np.linspace(2, 5, 300)
t4 = np.linspace(5, 6, 100)

y1 = np.zeros(len(t1))
y2 = -2+2*t2
y3 = 4 - t3
y4 = np.ones(len(t4))* -1

t = np.append(np.append(t1, t2), np.append(t3, t4))
y = np.append(np.append(y1, y2), np.append(y3, y4))

plt.ylabel('f(t)')
plt.xlabel('t')
plt.grid(True, linestyle='--', color='lightgrey')
step = plt.step(t, y)
plt.axis([0,6,-2,2])
plt.show()
```

3. Μαθηματική Περιγραφή Συστημάτων

Για την ανάλυση και τον σχεδιασμό συστημάτων ελέγχου γίνεται χρήση μαθηματικών μοντέλων σύνθετων φυσικών συστημάτων. Τα μαθηματικά μοντέλα, που προκύπτουν από τους φυσικούς νόμους, περιγράφονται γενικά από μη γραμμικές διαφορικές εξισώσεις. Πολλά φυσικά συστήματα συμπεριφέρονται γραμμικά γύρω από ένα σημείο λειτουργίας και για αυτόν τον λόγο είναι δυνατό να τα προσεγγίσουμε γραμμικά.[8]

Ορισμένα συστήματα στα οποία γίνεται χρήση τέτοιων μεθοδολογιών έχουν να κάνουν με την δυναμική μελέτη πλανητικών συστημάτων, την μελέτη σωματιδίων στο μικρόκοσμο, την συμπεριφορά ηλεκτρικών κυκλωμάτων, συστήματα που αφορούν πληθυσμούς κτλ.

Τέτοια συστήματα διαφορικών εξισώσεων είναι εξαιρετικά δύσκολο να λυθούν αναλυτικά. Με την πρόοδο της τεχνολογίας χρησιμοποιούνται όλο και περισσότερο αριθμητικές μέθοδοι επίλυσης προβλημάτων αρχικών και συνοριακών τιμών. Τέτοιες μέθοδοι είναι: μέθοδοι ενός βήματος (σειρές Taylor, Euler / Euler – Heun, Runge – Kutta κτλ), μέθοδοι πολλαπλών βημάτων (Adams, Milne κτλ), μέθοδοι πρόβλεψης διόρθωσης, μέθοδος Crank-Nicholson, μέθοδοι πεπερασμένων στοιχείων κτλ. Η μέθοδος που θα ακολουθήσουμε εμείς στηρίζεται στο μετασχηματισμό Laplace καθώς απλουστεύεται η περιγραφή των συστημάτων από αναλυτική σε αλγεβρική.

3.1. Διαφορική εξίσωση κατακόρυφου ελατηρίου

$$my''(t) + by'(t) + ky(t) = u(t)$$

όπου $y(t)$ η κατακόρυφη απομάκρυνση του ελατηρίου, $my''(t)$ το βάρος της μάζας m, $ky(t)$ η δύναμη επαναφοράς, $by'(t)$ η αντίσταση του αέρα και $u(t)$ η εξωτερική διέγερση η οποία είναι ρητή συνάρτηση του χρόνου (εξαναγκασμένη ταλάντωση).

[8] Δρ. Βολογιαννίδης Σ., Χατζηγεωργίου Κ., Γκουτζιαμάνης Π., *Συστήματα Αυτομάτου Ελέγχου - Σημειώσεις Εργαστηρίου*, ΤΕΙ Σερρών, (εργαστηριο 3, σελ 1)

Επίλυση Διαφορικής Εξίσωσης

Για την επίλυση της ΔΕ απαιτούνται τόσες αρχικές συνθήκες όση είναι και η τάξη της ΔΕ. Στην προκειμένη περίπτωση απαιτούνται 2. Η αρχική απομάκρυνση και η αρχική ταχύτητα.

$$Y_0 = \left[y(0), y'(0) \right]$$

Το πρόβλημα των ΔΕ είναι ότι δεν λύνονται πάντα αναλυτικά για αυτό καταφεύγουμε σε αριθμητικές μεθόδους. Μια πολύ χρήσιμη τεχνική είναι η μετατροπής μιας ΔΕ 2ης τάξης σε 2 ΔΕ 1ης τάξης και να ολοκληρώσουμε αριθμητικά το σύστημα. Έτσι η ΔΕ του ελατηρίου γράφεται σαν σύστημα πραγματοποιώντας την εξής αντικατάσταση

$$y' = z$$

$$z' = \frac{1}{m}\left(-bz - ky + u(t)\right)$$

Παράδειγμα επίλυσης ομογενούς ΔΕ ελατηρίου

Για $m = 10$, $k = 5$, $b = 2$, $u(t) = 0$ και αρχικές συνθήκες $Y_0 = \left[y(0), y'(0) \right] = [1,2]$

Για την αριθμητική ολοκλήρωση συστημάτων γίνεται χρήση της του πακέτου odeint τη βιβλιοθήκης scipy και υποπακέτου της integrate.

Η εντολή odeint(*func, y0, t*)

- *func*: Η συνάρτηση του συστήματος. Υπολογίζεται η παράγωγος του *y* ως προς *t*
- *y0*: Αρχικές συνθήκες του συστήματος (δέχεται σαν όρισμα και πίνακα).
- *t*: Μια συνέχεια χρονικών στιγμών για τα οποία θα λυθεί η *y*.

Η εντολή επιστρέφει έναν πίνακα με λύσεις των *y* και *y΄* για κάθε χρονική στιγμή *t*

Εισαγωγή των απαραίτητων βιβλιοθηκών για αριθμητικούς υπολογισμούς, γραφικές παραστάσεις και αριθμητικές ολοκλήρωσης συστημάτων ΔΕ με αρχικές συνθήκες

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint
```

Εισαγωγή του συστήματος ΔΕ 1ης τάξης σαν συνάρτηση

```
def system(u, t):
    y, z = u
    dudt = [z, (-2 * z - 5 * y) / 10]
    return dudt
```

Αρχικές συνθήκες και πεδίο ορισμού σαν διανύσματα

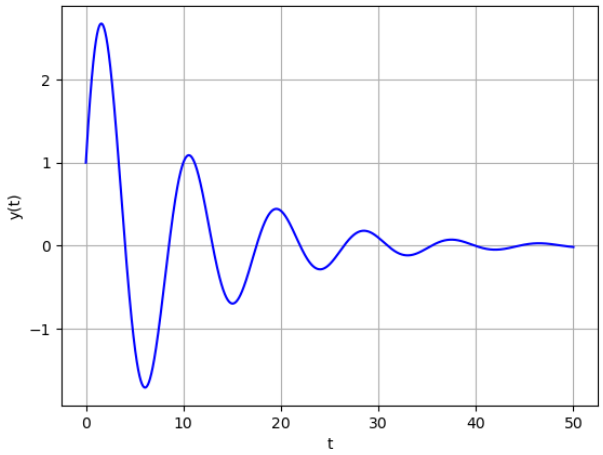
```
u0 = [1, 2]
t = np.linspace(0, 50, 10000)
```

Αριθμητική επίλυση του συστήματος

```
sol = odeint(system, u0, t)
```

Γραφική παράσταση απομάκρυνσης $y(t)$

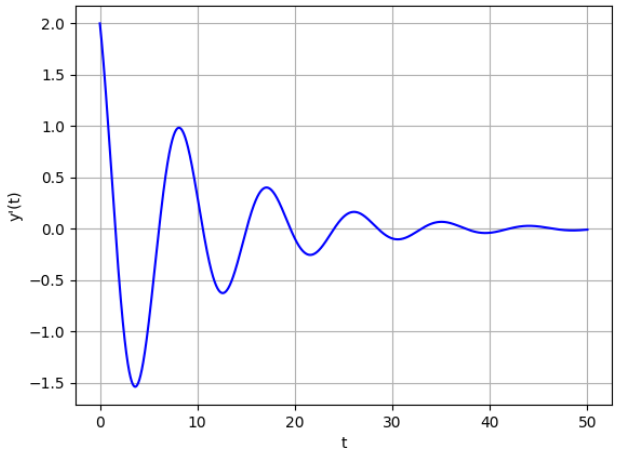
```
plt.figure(1)
plt.plot(t, sol[:, 0], 'b', label='y(t)') # plot y(t)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.grid()
```



Εικόνα 3.1 Διάγραμμα y(t)

Γραφική παράσταση ταχύτητας της απομάκρυνσης $z(t) = y'(t)$

```
plt.figure(2)
plt.plot(t, sol[:, 1], 'b', label='y(t)') # plot z(t)=y'(t)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.grid()
plt.show()
```



Εικόνα 3.2 Διάγραμμα y΄(t)

Παράδειγμα 1

Θα λύσουμε την ομογενή εξίσωση του ελατηρίου $u(t)=0$ για m=10, k=5, b=2 και αρχικές συνθήκες

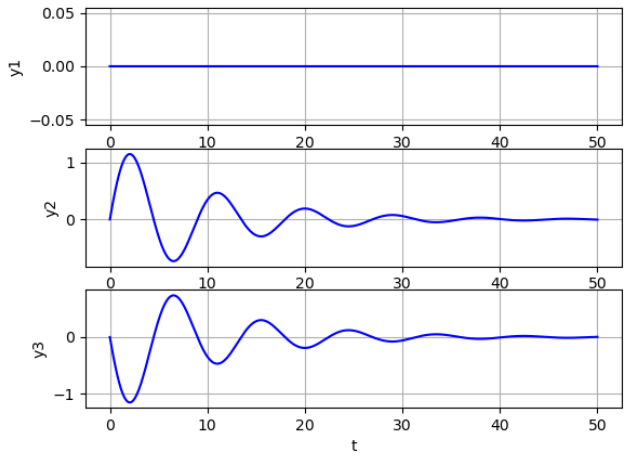
- $y(0)=0\ , y'(0)=0$
- $y(0)=0\ , y'(0)=1$
- $y(0)=0\ , y'(0)=-1$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

def pend(u, t):
    y, z = u
    dudt = [z, (-2*z - 5*y)/10]
    return dudt

u0a = [0, 0]
u0b = [0, 1]
u0c = [0, -1]

t = np.linspace(0,50,1000)

sola = odeint(pend, u0a, t)
solb = odeint(pend, u0b, t)
solc = odeint(pend, u0c, t)

print(f"max distance sola = {max(sola[:,0])}")
print(f"max distance solb = {max(solb[:,0])}")
print(f"max distance solc = {min(solc[:,0])}")

fig, (axs1, axs2, axs3) = plt.subplots(3, 1)
```

```
axs1.plot(t, sola[:, 0], 'b', label='y(t)')
axs2.plot(t, solb[:, 0], 'b', label='y(t)')
axs3.plot(t, solc[:, 0], 'b', label='y(t)')

plt.xlabel('t')

axs1.set_ylabel("y1")
axs2.set_ylabel("y2")
axs3.set_ylabel("y3")

axs1.grid()
axs2.grid()
axs3.grid()

plt.show()
```

Οι επιστρεφόμενες τιμές στην κονσόλα είναι μέγιστες αποστάσεις από σημείο ισορροπίας για το σύστημα με τις έκαστες αρχικές συνθήκες.

```
max distance sola = 0.0
max distance solb = 1.1530564285830702
max distance solc = -1.1530564285830702
```



Εικόνα 3.3 Διάγραμμα Παράδειγμα 1

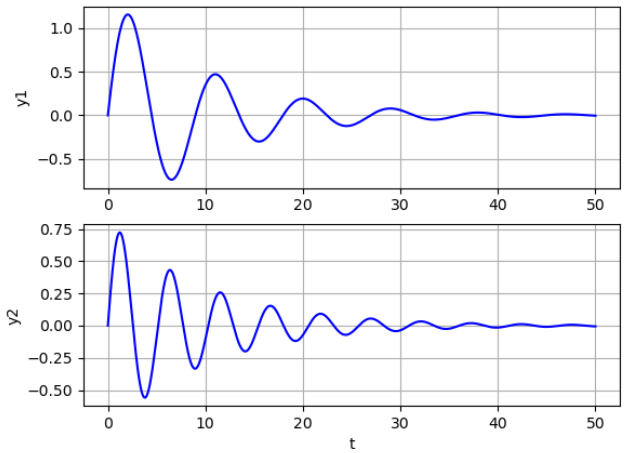
Παράδειγμα 2

Θα λύσουμε την ομογενή εξίσωση του ελατηρίου $u(t)=0$ για m=10, b=2 αρχικές συνθήκες $y(0)=0$, $y'(0)=1$ και

- k=5
- k=15

```
import matplotlib.pyplot as plt
import numpy as np
```

```
from scipy.integrate import odeint


def system(u, t, k):
    y, z = u
    dudt = [z, (-2 * z - k * y) / 10]
    return dudt


u0 = [0, 1]

t = np.linspace(0, 50, 10000)

k1 = 5
sol1 = odeint(system, u0, t, args=(k1,))
k2 = 15
sol2 = odeint(system, u0, t, args=(k2,))

fig, (axs1, axs2) = plt.subplots(2, 1)
axs1.plot(t, sol1[:, 0], 'b', label='y(t)')
axs2.plot(t, sol2[:, 0], 'b', label='y(t)')

plt.xlabel('t')

axs1.set_ylabel("y1")
axs2.set_ylabel("y2")

axs1.grid()
axs2.grid()

plt.show()
```



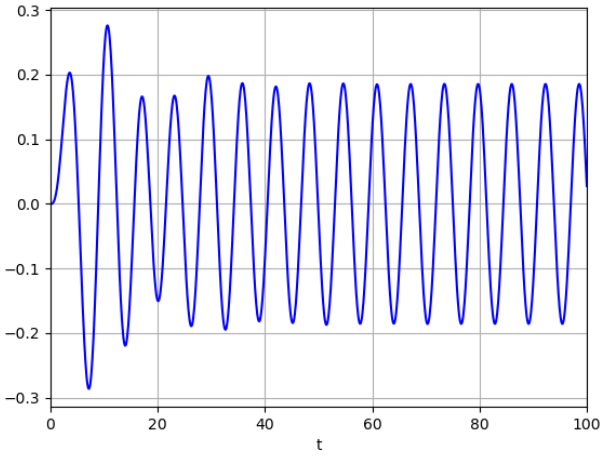
Εικόνα 3.4 Διάγραμμα Παράδειγμα 2

Παρατηρείται ότι για k=15 πραγματοποιούνται περισσότερες ταλαντώσεις στο σύστημα μέχρι να έρθει ισορροπία από ότι για k =5.

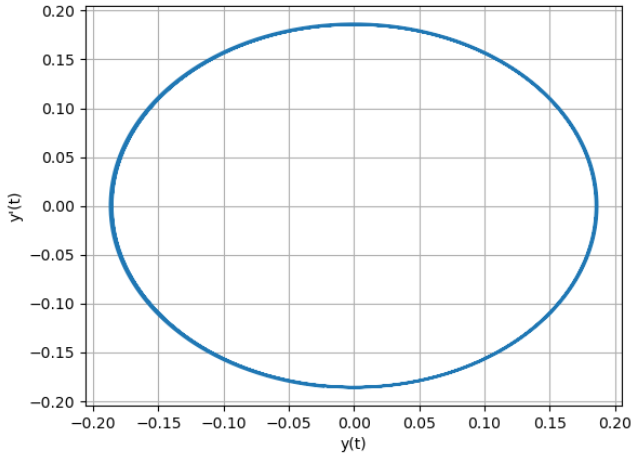
Παράδειγμα 3

Θα λύσουμε την μη ομογενή εξίσωση του ελατηρίου $u(t) = \sin(t)$ για m=10, k=5, b=2 και αρχικές συνθήκες $y(0) = 0$, $y'(0) = 0$.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint


def system(u, t):
    y, z = u
    dudt = [z, (-2 * z - 5 * y + np.sin(t)) / 10]
    return dudt

u0 = [0, 0]

t = np.linspace(0, 100, 10000)

sol = odeint(system, u0, t)

plt.figure(1)
plt.plot(t, sol[:, 0], 'b')
plt.title('Displacement plot')
plt.xlabel('t')
plt.ylabel('y')
plt.xticks(np.arange(0, 101, 20))
plt.xlim(0, 100)
plt.grid()

plt.figure(2)
plt.plot(sol[t>50, 0], sol[t>50, 1])
plt.title('Phase plot for t>50')
plt.xlabel("y(t)")
plt.ylabel("y'(t)")
plt.grid()

plt.show()
```



Εικόνα 3.5 Διάγραμμα y(t)



Εικόνα 3.6 Διάγραμμα φάσης

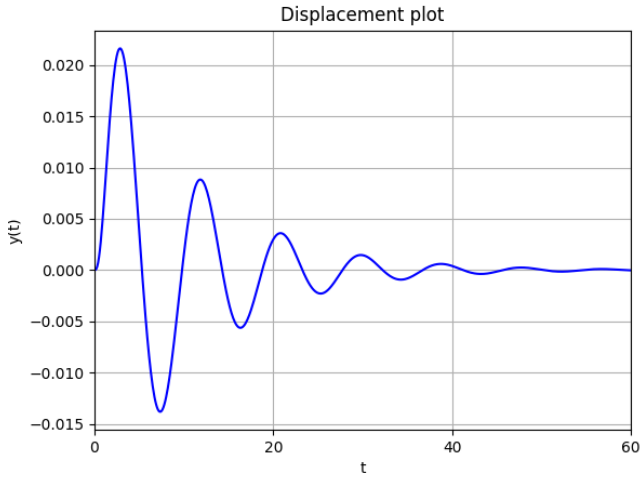
Παρατηρούμε πως το σύστημα εκτελεί ταλαντώσεις γύρω από το 0. Άρα το σύστημα είναι ευσταθές. Το διάγραμμα φάσης του συστήματος είναι κύκλος που σημαίνει ότι το σύστημα είναι ευσταθές

Παράδειγμα 4

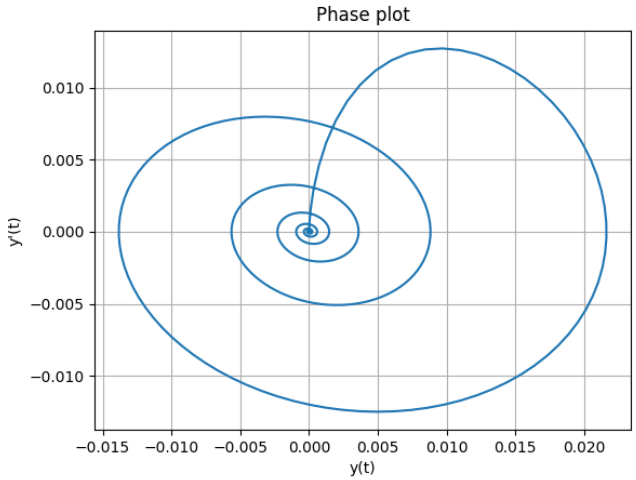
Θα λύσουμε την μη ομογενή εξίσωση του ελατηρίου $u(t)=\sin(t)e-2t$ για m=10, k=5, b=2 και αρχικές συνθήκες $y(0)=0$, $y'(0)=0$.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint


def system(u, t):
    y, z = u
```

```
    dudt = [z, (-2 * z - 5 * y + np.sin(t) * np.exp(-2 * t)) /
10]
    return dudt


u0 = [0, 0]

t = np.linspace(0, 100, 1000)

sol = odeint(system, u0, t)
plt.figure(1)
plt.title('Displacement plot')
plt.plot(t, sol[:, 0], 'b')
plt.xlabel('t')
plt.ylabel('y(t)')
plt.xticks(np.arange(0, 101, 20))
plt.xlim(0, 60)
plt.grid()

plt.figure(2)
plt.plot(sol[:, 0], sol[:, 1])
plt.title('Phase plot')
plt.xlabel("y(t)")
plt.ylabel("y'(t)")
plt.grid()
plt.show()
```



Εικόνα 3.7 Διάγραμμα y(t)



Εικόνα 3.8 Διάγραμμα φάσης

Από το διάγραμμα απομάκρυνσης του συστήματος παρατηρείται ότι το σύστημα έρχεται μετά από περίπου 50 δευτερόλεπτα σε ισορροπία. Το διάγραμμα φάσης είναι ευσταθής εστία.

4. Συνάρτηση Μεταφοράς

Έστω μια διαφορική εξίσωση της γενικής μορφής:

$$a_n y^{(n)}(t) + a_{n-1} y^{(n-1)}(t) + \dots + a_0 y^{(0)}(t) = b_m u^{(m)}(t) + b_{m-1} u^{(m-1)}(t) + \dots + b_0 u^{(0)}(t)$$

όπου $u(t)$ το σήμα εισόδου.

Έστω $f(t)$ πραγματική συνάρτηση του χρόνου $(t > 0)$.

4.1. Μετασχηματισμός Laplace

Ο μετασχηματισμός Laplace ορίζεται:

$$\mathcal{L}\{f(t)\} = F(s) = \int_0^\infty f(t) e^{-st} dt$$

όπου $s = \sigma + j\omega$ η μιγαδική συχνότητα.

Ο αντίστροφος μετασχηματισμός Laplace:

$$\mathcal{L}^{-1}\{F(s)\} = f(t) = \frac{1}{2\pi j} \int_{c-j\infty}^{c+j\infty} F(s) e^{st} ds$$

Ιδιότητες μετασχηματισμού Laplace

- **Γραμμικότητα $\mathcal{L}$ και $\mathcal{L}^{-1}$**

$$\mathcal{L}\{af(t) + bg(t)\} = \mathcal{L}\{af(t)\} + \mathcal{L}\{bf(t)\} = aF(s) + bG(s)$$

- **Παραγώγιση**

$$\mathcal{L}\left\{\frac{d^n}{dt^n} f(t)\right\} = s^n F(s) - s^{n-1} f(0) - s^{n-2} f'(0) - \cdots - f^{(n-1)}(0)$$

Αν το σύστημα έχει *μηδενικές αρχικές συνθήκες*, τότε η ΔΕ εξίσωση στο πεδίο του χρόνου μετατρέπεται σε αλγεβρική εξίσωση στο πεδίο των συχνοτήτων.

$$(a_n s^n + a_{n-1} s^{n-1} + \cdots + a_0) Y(s) = (b_m s^m + b_{m-1} s^{m-1} + \cdots + b_0) U(s)$$

Συνάρτηση μεταφοράς του συστήματος που αντιστοιχεί στην διαφορική εξίσωση ονομάζεται ο μετασχηματισμός Laplace της διαφορικής εξίσωσης που δίνεται από την ακόλουθη σχέση:

$$G(s) = \frac{b_m s^{(m)}(t) + b_{m-1} s^{(m-1)}(t) + \dots + b_0 s^{(0)}(t)}{a_n s^{(n)}(t) + a_{n-1} s^{(n-1)}(t) + \dots + a_0 s^{(0)}(t)} = \frac{Y(s)}{U(s)}$$

Η συνάρτηση μεταφοράς μας δίνει τον λόγο του μετασχηματισμού Laplace της εξόδου προς τον μετασχηματισμό Laplace της εισόδου και ορίζεται για μηδενικές αρχικές συνθήκες.

Συνάρτηση	$f(t)$	$L\{f(t)\}$
Μοναδιαία κρουστική συνάρτηση	$\delta(t)$	1
Μοναδιαία βηματική συνάρτηση	$1(t)$	$\frac{1}{s}$
Μοναδιαία συνάρτηση κλίσης	t	$\frac{1}{s^2}$
Πολυώνυμο	t^n	$\frac{n!}{s^{n+1}}$
Εκθετική	e^{-at}	$\frac{1}{s+a}$
Ημιτονοειδής	$\sin(\omega t)$	$\frac{\omega}{s^2+\omega^2}$
Συνημιτονοειδής	$\cos(\omega t)$	$\frac{s}{s^2+\omega^2}$
Αποσβενύμενη Ημιτονοειδής	$e^{-at}\sin(\omega t)$	$\frac{\omega}{(s+a)^2+\omega^2}$
Αποσβενύμενη Συνημιτονοειδής	$e^{-at}\cos(\omega t)$	$\frac{s+a}{(s+a)^2+\omega^2}$

Πίνακας 4.1 Μετασχηματισμοί Laplace γνωστών συναρτήσεων

4.2. Αποκρίσεις

Ελεύθερη απόκριση ενός συστήματος είναι η έξοδος του συστήματος όταν η είσοδος του είναι 0.

Δυναμική απόκριση ενός συστήματος είναι η έξοδος του συστήματος όταν οι αρχικές συνθήκες είναι μηδενικές.

Κρουστική απόκριση ενός συστήματος ονομάζεται η δυναμική απόκριση όταν η είσοδος του συστήματος είναι κρουστική συνάρτηση(συνάρτηση του Dirac).

Στην Python ο υπολογισμός της κρουστικής απόκρισης ενός συστήματος πραγματοποιείται με την εντολή: **T, yout = control.impulse_response(g, t)** όπου

- g: Η συνάρτηση μεταφοράς του συστήματος
- t: Το διάνυσμα του χρόνου

και επιστρέφει:

- T: Τιμές χρονικών στιγμών της εξόδου
- yout: Απόκριση του συστήματος

Βηματική απόκριση ενός συστήματος λέγεται η δυναμική απόκριση όταν η είσοδος του συστήματος είναι η βηματική συνάρτηση.

Στην Python υπολογισμός της βηματικής απόκρισης ενός συστήματος πραγματοποιείται με την εντολή: **T, yout = control.step_response(g, t)** η οποία λειτουργεί παρόμοια με την εντολή της κρουστικής απόκρισης.

Χαρακτηριστικά:

Μέγιστο πλάτος ταλάντωσης (Peak Amplitude): Η μέγιστη τιμή της βηματικής απόκρισης.

Χρόνος αποκατάστασης (Settling Time): Το χρονικό διάστημα στο οποίο η βηματική απόκριση φθάσει και θα παραμείνει σχεδόν σε κατάσταση ισορροπίας.

Υπερύψωση(Overshoot): Ποσοστό που δείχνει την ύπαρξη ή όχι έντονων ταλαντώσεων γύρο από το σημείο ισορροπίας.

Χρόνος ανόδου(Rise Time): Το χρονικό διάστημα στο οποίο η βηματική απόκριση φτάνει από το 10% στο 90% της τελικής τιμής.

Τελική τιμή(Steady time): Η τελική τιμή του συστήματος όταν φτάσει σε κατάσταση ισορροπίας.

Με την εντολή lsim() γίνεται δυνατή η προσομοίωση ενός γραμμικού συστήματος στην Python. Για τη χρήση της απαιτείται η εισαγωγή του πακέτου matlab από τη βιβλιοθήκη control.

```
from control import matlab
```

Η εντολή lsim() συντάσσεται ως εξής:

yout, T, xout = matlab.lsim(g, U, t)

- g: Η συνάρτηση μεταφοράς του συστήματος
- U: Είσοδος για κάθε t
- t: Το διάνυσμα του χρόνου

και επιστρέφει:

- T: Τιμές χρόνων της εξόδου
- yout: Απόκριση του συστήματος
- xout: Χρονική εξέλιξη του διανύσματος καταστάσεων

Έστω ότι ένα σύστημα περιγράφεται από την παρακάτω διαφορική εξίσωση:

$$5y''(t) + 3y'(t) + 7y(t) = 3u'(t) + 2u(t)$$

και επιθυμούμε να βρεθεί η συνάρτηση μεταφοράς και η εισαγωγή της στην Python.

Εισαγωγή πακέτου συστημάτων ελέγχου:

Για τη εισαγωγή της συνάρτησης μεταφοράς του συστήματος χρειάζεται η εισαγωγή της βιβλιοθήκης control.

```
import control as co
```

Εισαγωγή συνάρτησης μεταφοράς (1ος τρόπος)

```
g1 = co.tf([3, 2], [5, 3, 7])
print(f"g1 = {g1}")
```

```
g1 =

    3 s + 2

---------------

5 s^2 + 3 s + 7
```

Εισαγωγή συνάρτησης μεταφοράς (2ος τρόπος)

```
s = co.tf('s')
g2 = (3 * s + 2) / (5 * s ** 2 + 3 * s + 7)
print(f"g2 = {g2}")
```

```
g2 =
     3 s + 2
---------------
5 s^2 + 3 s + 7
```

Παράδειγμα 1

Έστω οι διαφορικές εξισώσεις των συστημάτων:

$$3y''(t)+2y'(t)+6y(t)=6u(t)$$

$$6y'''(t)-8y'(t)+4y(t)=2u'(t)-3u(t)$$

Και θέλουμε να εμφανίσουμε την κρουστική και τη βηματική απόκριση τους, όπως επίσης το μέγιστο πλάτος ταλάντωσης και ο χρόνος αποκατάστασης των αποκρίσεων.

Για αρχή, εισάγουμε τα απαραίτητα πακέτα στην Python:

```
import control as co
import matplotlib.pyplot as plt
import numpy as np
```

Εισάγουμε τις συναρτήσεις μεταφοράς και το διάνυσμα του χρόνου:

```
s = co.tf('s')
g1 = 6 / (3 * s ** 2 + 2 * s + 6)
g2 = (2 * s - 3) / (6 * s ** 3 - 8 * s + 4)

t = np.linspace(0, 200, 10000)
```

Υπολογισμός κρουστικής και βηματικής απόκρισης για κάθε συνάρτηση μεταφοράς:

```
t1, imp1 = co.impulse_response(g1, t)
t2, imp2 = co.impulse_response(g2, t)
t3, stp1 = co.step_response(g1, t)
t4, stp2 = co.step_response(g2, t)
```

Ορισμός συνάρτησης που επιστρέφει το μέγιστο πλάτος:

```
def peak_amplitude(signal, t):
    index_max = list(signal).index(max(signal))
    peak = [t[index_max], signal[index_max]]
    return peak
```

Ορισμός συνάρτησης που εκτυπώνει τα χαρακτηριστικά της βηματικής συνάρτησης (μέγιστο πλάτος, υπερύψωση, χρόνος ανόδου, χρόνος αποκατάστασης):

```
def step_info(t, yout):
    print(f"Max Amp: {max(yout)}")
    print("OS: %f%s" % ((yout.max() / yout[-1] - 1) * 100, '%'))
    print("Tr: %fs" % (t[next(i for i in range(0, len(yout) - 1)
if yout[i] > yout[-1] * .90)] - t[0]))
    print("Ts: %fs" % (t[next(len(yout) - i for i in range(2,
len(yout) - 1) if abs(yout[-i] / yout[-1]) > 1.02)] - t[0]))
```

Εκτύπωση μέγιστου χρόνου:

```
print(f"g1 impulse : time= {peak_amplitude(imp1,t1)[0]},
amplitude max= {peak_amplitude(imp1,t1)[1]}")
print(f"g2 impulse : time= {peak_amplitude(imp2,t2)[0]},
amplitude max= {peak_amplitude(imp2,t2)[1]}")
print(f"g1 step : time= {peak_amplitude(stp1,t3)[0]}, amplitude
max= {peak_amplitude(stp1,t3)[1]}")
print(f"g2 step : time= {peak_amplitude(stp2,t4)[0]}, amplitude
max= {peak_amplitude(stp2,t4)[1]}")
```

```
g1 impulse : time= 0.9600960096009601, amplitude max=
1.0234860045806817
g2 impulse : time= 197.23972397239726, amplitude max=
1.5189143752301397e+57
g1 step : time= 2.28022802280228, amplitude max=
1.4667414936186833
g2 step : time= 198.67986798679868, amplitude max=
5.73062695880044e+57
```

Χαρακτηριστικά της βηματικής:

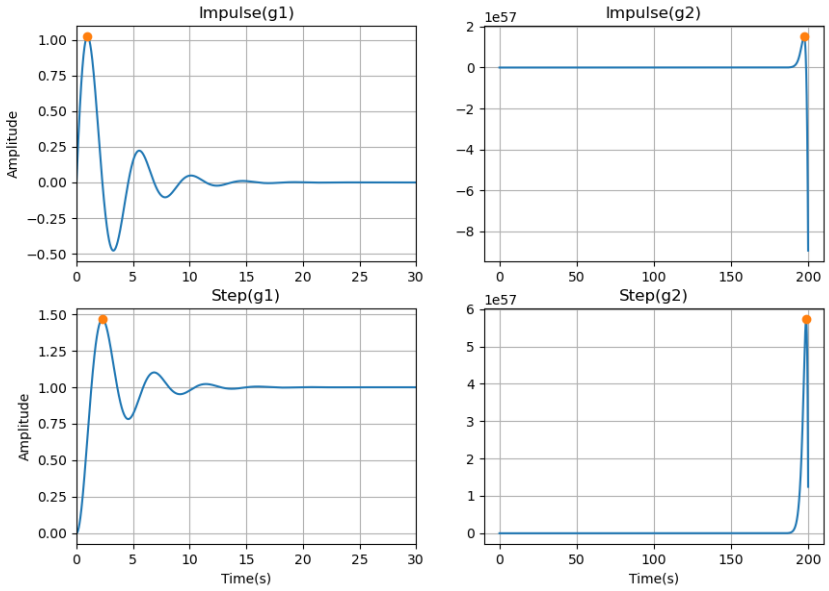
```
print(step_info(t3, stp1))
print(step_info(t1, imp1))
```

```
Max Amp: 1.4667414936186833
OS: 46.674149%
Tr: 1.220122s
Ts: 11.741174s
None
Max Amp: 1.0234860045806817
OS: -631209835267451558546339004416 0.000000%
Tr: 0.000000s
Ts: 199.899990s
```

None

Γραφικές παραστάσεις αποκρίσεων:

```
# ορισμός πλέγματος γραφικών παραστάσεων
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

# γραφικές παραστάσεις αποκρίσεων και μεγίστων χρόνων
axs[0, 0].plot(t1, imp1)
axs[0, 0].plot(peak_amplitude(imp1,t1)[0],
peak_amplitude(imp1,t1)[1], "o")
axs[0, 1].plot(t2, imp2)
axs[0, 1].plot(peak_amplitude(imp2,t2)[0],
peak_amplitude(imp2,t2)[1], "o")
axs[1, 0].plot(t3, stp1)
axs[1, 0].plot(peak_amplitude(stp1,t3)[0],
peak_amplitude(stp1,t3)[1], "o")
axs[1, 1].plot(t4, stp2)
axs[1, 1].plot(peak_amplitude(stp2,t4)[0],
peak_amplitude(stp2,t4)[1], "o")

# Όρια άξονα x γραφικής παράστασης
axs[0, 0].set_xlim([0, 30])
axs[1, 0].set_xlim([0, 30])

# πλέγματα
axs[0, 0].grid()
axs[0, 1].grid()
axs[1, 0].grid()
axs[1, 1].grid()

# Τίτλοι
axs[0, 0].set_title("Impulse(g1)")
axs[0, 1].set_title("Impulse(g2)")
axs[1, 0].set_title("Step(g1)")
axs[1, 1].set_title("Step(g2)")

plt.setp(axs[-1, :], xlabel='Time(s)')
plt.setp(axs[:, 0], ylabel='Amplitude')

plt.show()
```



Εικόνα 4.1 Διαγράμματα αποκρίσεων

Παράδειγμα 2

Έστω η συνάρτηση μεταφοράς:

$$G(s) = \frac{(4s^2 - 3)(6s + 8)}{(6s^3 - 2s^2 + 9)(s^2 + 8)}$$

Για την εισαγωγή της στην Python θα γράψουμε:

```
import control as co
import matplotlib.pyplot as plt
import numpy as np
from control import matlab

s = co.tf('s')
g1 = ((4 * s ** 2 - 3) * (6 * s + 8)) / ((6 * s ** 3 - 2 * s ** 2
+ 9) * (s ** 2 + 8))
print(g1)
```

```
    24 s^3 + 32 s^2 - 18 s - 24

-----------------------------------

6 s^5 - 2 s^4 + 48 s^3 - 7 s^2 + 72
```

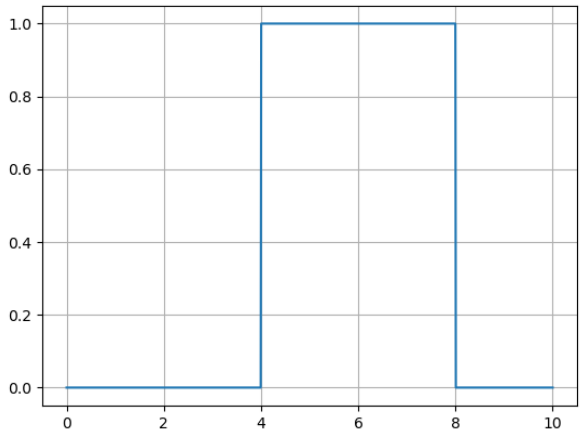
Η κρουστική και η βηματική απόκρισή της μαζί με το διάνυσμα χρόνου:

```
t = np.linspace(0, 200, 1000)
```

```
t1, imp = co.impulse_response(g1, t)
t2, stp = co.step_response(g1, t)
```

Έστω ο τετραγωνικός παλμός:

```
l1 = len(np.linspace(0, 4, 400, endpoint=False))
l2 = len(np.linspace(4, 8, 400, endpoint=False))
l3 = len(np.linspace(8, 10, 200))

sq = np.append(np.append(np.zeros(l1), np.ones(l2)),
np.zeros(l3))

plt.plot(tsq,sq)
plt.grid()
plt.show()
```



Εικόνα 4.2 Τετραγωνικός παλμός

Η απόκριση για είσοδο τετραγωνικό παλμό (Χρήση της lsim της control.matlab):

```
yout, T, xout = matlab.lsim(g1, sq, tsq)
```

Οι συναρτήσεις μέγιστου και ελάχιστου πλάτους:

```
def max_amplitude(signal, time):
    index_max = list(signal).index(max(signal))
    peak_max = [time[index_max], signal[index_max]]
    return peak_max
def min_amplitude(signal, time):
    index_min = list(signal).index(min(signal))
    peak_min = [time[index_min], signal[index_min]]
    return peak_min
```

Εκτύπωση $y_{\max}$ και $y_{\min}$:

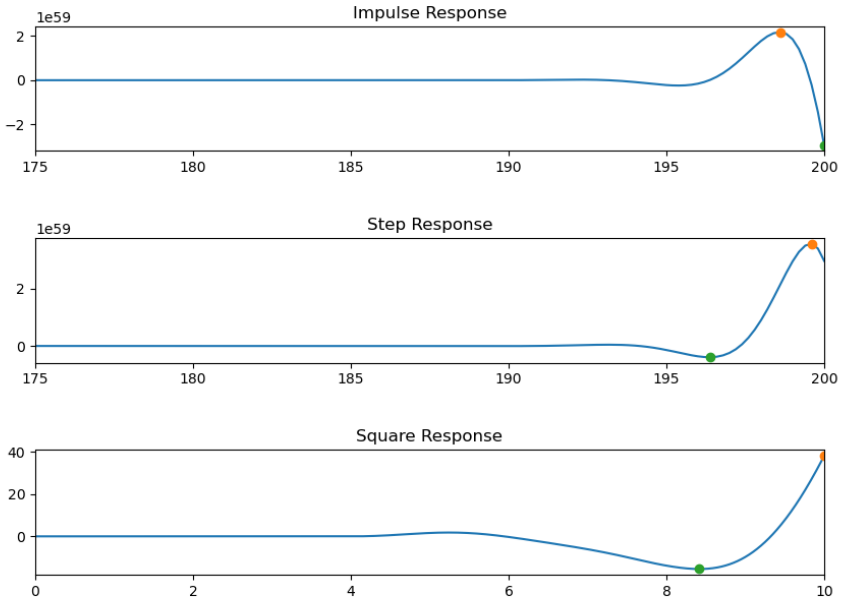
```
print(f"g1 impulse : time= {max_amplitude(imp, t1)[0]}, amplitude
max= {max_amplitude(imp, t1)[1]}")
print(f"g1 impulse : time= {min_amplitude(imp, t1)[0]}, amplitude
min= {min_amplitude(imp, t1)[1]}")
print(f"g1 step : time= {max_amplitude(stp, t2)[0]}, amplitude
max= {max_amplitude(stp, t2)[1]}")
print(f"g1 step : time= {min_amplitude(stp, t2)[0]}, amplitude
min= {min_amplitude(stp, t2)[1]}")
print(f"g1 square : time= {max_amplitude(yout, T)[0]}, amplitude
max= {max_amplitude(yout, T)[1]}")
print(f"g1 square : time= {min_amplitude(yout, T)[0]}, amplitude
min= {min_amplitude(yout, T)[1]}")
```

```
g1 impulse : time= 198.5985985985986, amplitude max=
2.1707643884822107e+59
g1 impulse : time= 200.0, amplitude min= -2.938916277094198e+59
g1 step : time= 199.5995995995996, amplitude max=
3.545233352225417e+59
g1 step : time= 196.3963963963964, amplitude min= -
3.912901344665081e+58
g1 square : time= 10.0, amplitude max= 38.21162393883365
g1 square : time= 8.418418418418419, amplitude min= -
15.484939024636743
```

Γραφικές παραστάσεις:

```
fig, axs = plt.subplots(3, 1, figsize=(10, 7))
plt.subplots_adjust(hspace=0.7)
axs[0].plot(t1, imp)
axs[0].plot(max_amplitude(imp, t1)[0], max_amplitude(imp, t1)[1],
"o")
axs[0].plot(min_amplitude(imp, t1)[0], min_amplitude(imp, t1)[1],
"o")

axs[1].plot(t2, stp)
axs[1].plot(max_amplitude(stp, t2)[0], max_amplitude(stp, t2)[1],
"o")
axs[1].plot(min_amplitude(stp, t2)[0], min_amplitude(stp, t2)[1],
"o")

axs[2].plot(T, yout)
axs[2].plot(max_amplitude(yout, T)[0], max_amplitude(yout, T)[1],
"o")
axs[2].plot(min_amplitude(yout, T)[0], min_amplitude(yout, T)[1],
"o")

axs[0].set_title("Impulse Response")
```

```
axs[1].set_title("Step Response")
axs[2].set_title("Square Response")

axs[0].set_xlim([175, 200])
axs[1].set_xlim([175, 200])
axs[2].set_xlim([0, 10])
plt.show()
```



Εικόνα 4.3 Γραφικές παραστάσεις του συστήματος με είσοδο α)κρουστική απόκριση, β)βηματική απόκριση, γ)τετραγωνικό παλμό

Παράδειγμα 3

Έστω η διαφορική εξίσωση του κατακόρυφου συστήματος (μάζας-ελατηρίου):

$$my''(t)+by'(t)+ky(t)=u(t)$$

Για $k=5$, $b=2$, οι συναρτήσεις μεταφοράς για $m_1=10$ και $m_2=80$ είναι:

```
import control as co
import matplotlib.pyplot as plt
import numpy as np


k = 5
b = 2
m1 = 10
m2 = 80
t = np.linspace(0, 1000, 10001)

s = co.tf('s')
```

```
g1 = 1 / (m1 * s ** 2 + b * s + k)
g2 = 1 / (m2 * s ** 2 + b * s + k)

print(g1, g2)
```

```
       1
---------------
10 s^2 + 2 s + 5


       1
---------------
80 s^2 + 2 s + 5
```

Η κρουστική και η βηματική απόκριση για κάθε περίπτωση:

```
t1, imp1 = co.impulse_response(g1, t)
t2, stp1 = co.step_response(g1, t)
t3, imp2 = co.impulse_response(g2, t)
t4, stp2 = co.step_response(g2, t)
```

Συναρτήσεις μεγίστου και ελαχίστου πλάτους:

```
def max_amplitude(signal, time):
    index_max = list(signal).index(max(signal))
    peak_max = [time[index_max], signal[index_max]]
    return peak_max


def min_amplitude(signal, time):
    index_min = list(signal).index(min(signal))
    peak_min = [time[index_min], signal[index_min]]
    return peak_min


def signal_info(signal, time):
    print(f"time= {max_amplitude(signal, time)[0]}, amplitude
max= {max_amplitude(signal, time)[1]}")
    print(f"time= {min_amplitude(signal, time)[0]}, amplitude
min= {min_amplitude(signal, time)[1]}")
```

Εκτύπωση μεγίστου και ελαχίστου πλάτους:

```
print("g1 impulse info")
signal_info(imp1, t1)
print("g1 step info")
signal_info(stp1, t2)
print("g2 impulse info")
signal_info(imp2, t3)
```

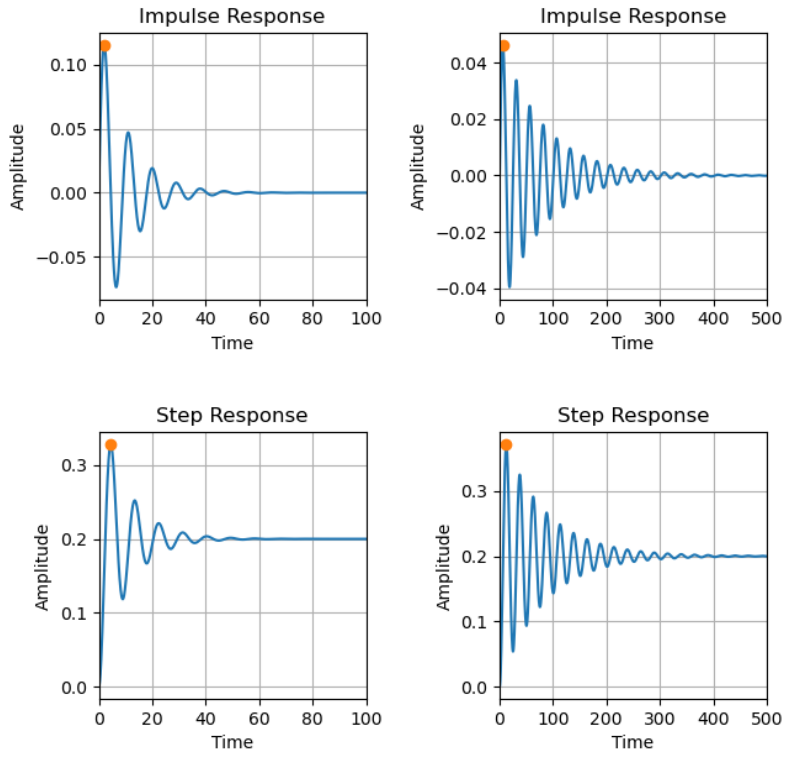
```
print("g2 step info")
signal_info(stp2, t4)
```

```
g1 impulse info
time= 2.0, amplitude max= 0.11525971422199222
time= 6.5, amplitude min= -0.07359680980773696
g1 step info
time= 4.5, amplitude max= 0.3276742861621773
time= 0.0, amplitude min= 0.0
g2 impulse info
time= 6.1000000000000005, amplitude max= 0.04633447636494252
time= 18.7, amplitude min= -0.03959051568146758
g2 step info
time= 12.600000000000001, amplitude max= 0.37089186928035905
time= 0.0, amplitude min= 0.0
```

Γραφικές παραστάσεις:

```
fig, axs = plt.subplots(2, 2, figsize=(7, 7))
plt.subplots_adjust(hspace=0.5, wspace=0.5)

axs[0, 0].plot(t1, imp1)
axs[0, 0].plot(max_amplitude(imp1, t1)[0], max_amplitude(imp1,
t1)[1], "o")

axs[0, 1].plot(t3, imp2)
axs[0, 1].plot(max_amplitude(imp2, t3)[0], max_amplitude(imp2,
t3)[1], "o")

axs[1, 0].plot(t2, stp1)
axs[1, 0].plot(max_amplitude(stp1, t2)[0], max_amplitude(stp1,
t2)[1], "o")

axs[1, 1].plot(t4, stp2)
axs[1, 1].plot(max_amplitude(stp2, t4)[0], max_amplitude(stp2,
t4)[1], "o")

axs[0, 0].set_title("Impulse Response")
axs[0, 1].set_title("Impulse Response")
axs[1, 0].set_title("Step Response")
axs[1, 1].set_title("Step Response")

axs[0, 0].set_xlabel("Time")
axs[0, 1].set_xlabel("Time")
axs[1, 0].set_xlabel("Time")
axs[1, 1].set_xlabel("Time")
```

```
axs[0, 0].set_ylabel("Amplitude")
axs[0, 1].set_ylabel("Amplitude")
axs[1, 0].set_ylabel("Amplitude")
axs[1, 1].set_ylabel("Amplitude")

axs[0, 0].set_xlim([0, 100])
axs[1, 0].set_xlim([0, 100])
axs[0, 1].set_xlim([0, 500])
axs[1, 1].set_xlim([0, 500])

axs[0, 0].grid()
axs[0, 1].grid()
axs[1, 0].grid()
axs[1, 1].grid()

plt.show()
```



Εικόνα 4.4 Διαγράμματα κρουστικής και βηματικής απόκρισης του συστήματος για m=10 (στήλη α), m=80 (στήλη β)

Ορισμός συνάρτησης υπολογισμού χαρακτηριστικών βηματικής απόκρισης:

```
def step_info(t, yout):
    print(f"Max Amp: {max(yout)}")
    print("OS: %f%s" % ((yout.max() / yout[-1] - 1) * 100, '%'))
    print("Tr: %fs" % (t[next(i for i in range(0, len(yout) - 1)
if yout[i] > yout[-1] * .90)] - t[0]))
```

```
    print("Ts: %fs" % (t[next(len(yout) - i for i in range(2,
len(yout) - 1) if abs(yout[-i] / yout[-1]) > 1.02)] - t[0]))
```

Τα χαρακτηριστικά των βηματικών αποκρίσεων για $m = 10$ και $m = 80$:

```
print("System 1 (m=10) step response info:")
step_info(t2, stp1)
print("-" * 50)
print("System 2 (m=80) step response info:")
step_info(t4, stp2)
```

```
System 1 (m=10) step response info:
Max Amp: 0.3276742861621773
OS: 63.837143%
Tr: 2.300000s
Ts: 33.000000s
--------------------------------------------------
System 2 (m=80) step response info:
Max Amp: 0.37089186928035905
OS: 85.445852%
Tr: 6.100000s
Ts: 292.300000s
```

Παρατηρούμε ότι από τα διαγράμματα και από τις step_info() των συστημάτων ότι το πρώτο σύστημα ($m = 10$) έρχεται πιο γρήγορα σε κατάσταση ισορροπίας, ενώ το μέγιστο πλάτος ταλάντωσης είναι πρακτικά το ίδιο και για δύο συστήματα.

5. Ανάλυση Συστημάτων

Όπως είπαμε, η συνάρτηση μεταφοράς ενός συστήματος είναι ο λόγος του μετασχηματισμού Laplace της εξόδου y(t) προς τον μετασχηματισμό Laplace της εισόδου u(t) με αρχικές τις συνθήκες του συστήματος να είναι 0.

$$G(s)=\frac{b_m s^{(m)}(t)+b_{m-1}s^{(m-1)}(t)+\cdots+b_0 s^{(0)}(t)}{a_n s^{(n)}(t)+a_{n-1}s^{(n-1)}(t)+\cdots+a_0 s^{(0)}(t)}=\frac{Y(s)}{U(s)}$$

Οι τιμές του s για τις οποίες ισχύει ότι $G(s)=0$ λέγονται **μηδενικά του συστήματος**, ενώ οι τιμές του s για τις οποίες η $G(s)$ απειρίζεται ονομάζονται **πόλοι του συστήματος**.[9] Δηλαδή, μηδενικά ενός συστήματος είναι οι τιμές με τις οποίες ο αριθμητής της $G(s)$ γίνεται 0 ενώ πόλους του συστήματος ονομάζουμε τις τιμές για τις οποίες ο παρονομαστής της $G(s)$ γίνεται 0.

Στην Python, για να υπολογιστούν οι πόλοι και τα μηδενικά συστημάτων εισάγουμε τη βιβλιοθήκη control και χρησιμοποιούμε τις εντολές pole() και zero() αντίστοιχα με όρισμα το σύστημα που μελετάμε. Επίσης, με την εντολή pzmap(), της βιβλιοθήκης control, είναι δυνατός ο σχεδιασμός του διαγράμματος πόλων-μηδενικών, με τα μηδενικά να συμβολίζονται με (o) ενώ οι πόλοι με (x).

Παράδειγμα 1

Έστω η συνάρτηση μεταφοράς $G(s)=\frac{s+4}{7s^2+3s-1}$

```
import control as co
import matplotlib.pyplot as plt

s = co.tf('s')
g = (s + 4) / (7 * s ** 2 + 3 * s - 1)
```

Οι πόλοι και τα μηδενικά του συστήματος

```
g_poles = co.pole(g)
g_zeros = co.zero(g)

print('poles')
for elem in g_poles:  # print poles as a column vector for
convenience
    print(f'{elem: .2f}')  # Two floating points per print
```

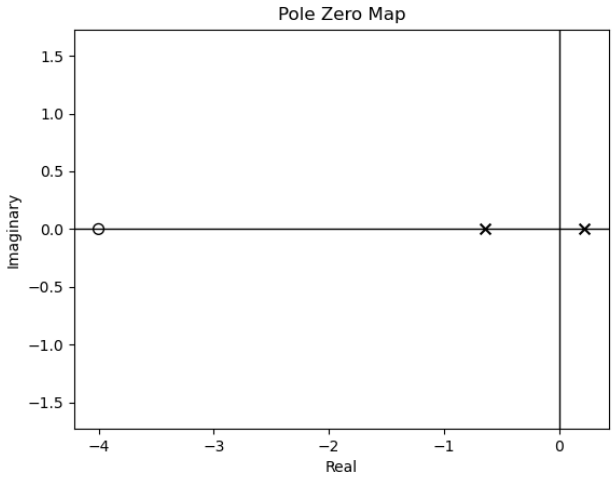
[9] Πετρίδης Β., 2011. *Συστήματα Αυτομάτου Ελέγχου,* εκδόσεις Ζήτη, (σελ 34-35)
Δρ. Βολογιαννίδης Σ., *Συστήματα Αυτομάτου Ελέγχου Θεωρία και Εφαρμογές, Διδακτικές Σημειώσεις Τμήματος Πληροφορικής και Επικοινωνιών*, (σελ 37)

```
print('zeros')
for elem in g_zeros:
    print(f'{elem: .2f}')
```

```
poles
-0.65
 0.22
zeros
-4.00
```

Το διάγραμμα πόλων-μηδενικών

```
g_map = co.pzmap(g, plot=True)
plt.show()
```



Εικόνα 5.1 Διάγραμμα πόλων-μηδενικών

5.1. Ευστάθεια Συστημάτων

Ορισμοί Ευστάθειας

- Ένα σύστημα είναι ευσταθές όταν η κρουστική συνάρτηση του τείνει στο 0 όταν ο χρόνος τείνει στο άπειρο.
- Ένα σύστημα είναι ευσταθές όταν για κάθε πεπερασμένου πλάτους είσοδο παράγεται πεπερασμένου πλάτους έξοδος.

Κριτήριο Ευστάθειας

Ένα σύστημα είναι ευσταθές όταν οι πόλοι του έχουν αυστηρά αρνητικό πραγματικό μέρος $\Re\left(s_p\right) < 0$.

Ένα σύστημα που δεν είναι ευσταθές, δηλαδή δεν ικανοποιεί το παραπάνω κριτήριο, θα λέγεται **ασταθές**.

Παράδειγμα 2

Έστω ένα σύστημα $G(s) = \frac{1}{(s+1)(s+2)}$

```
import control as co
import matplotlib.pyplot as plt
import numpy as np

s = co.tf('s')
g = 1 / ((s + 1) * (s + 2))
```

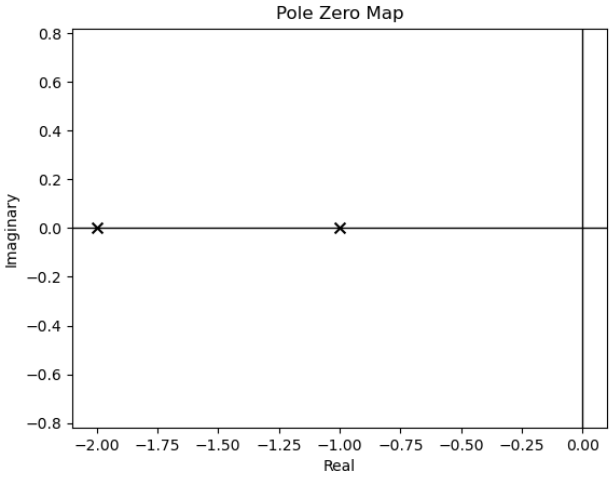
Οι πόλοι του συστήματος

```
print(co.pole(g))
```

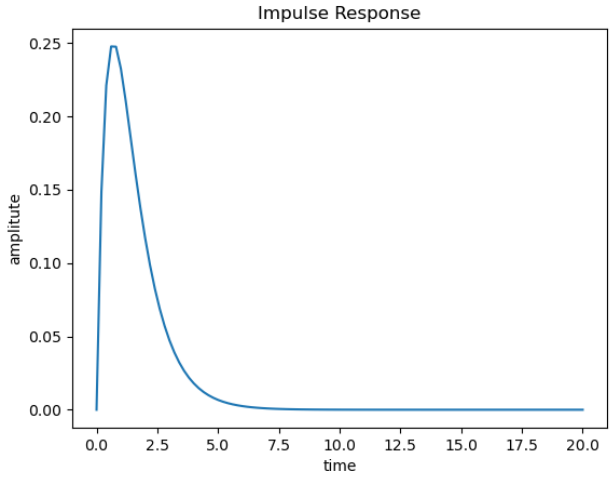
```
[-2. -1.]
```

Τα διαγράμματα πόλων-μηδενικών και η κρουστική απόκριση

```
# pole zero map
g_map = co.pzmap(g, plot=True)
plt.show()

# impulse response
ts = np.linspace(0, 20, 101)
t, y = co.impulse_response(g, ts)

# impulse response plot
plt.plot(t,y)
plt.title('Impulse Response')
plt.xlabel('time')
plt.ylabel('amplitute')
plt.show()
```



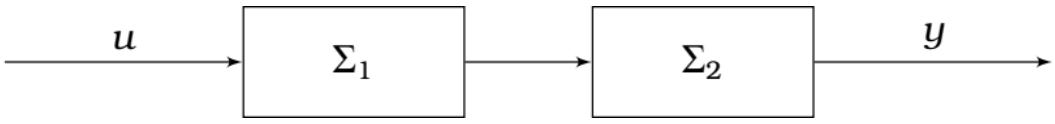
Εικόνα 5.2 Διάγραμμα πόλων-μηδενικών



Εικόνα 5.3 Διάγραμμα κρουστικής απόκρισης

5.2. Λειτουργικά διαγράμματα – διασυνδέσεις συστημάτων

Σύνδεση σε σειρά



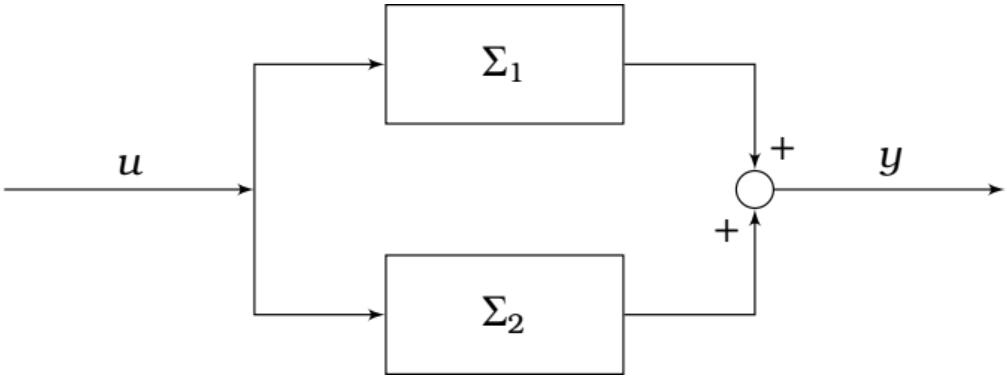
Εικόνα 5.4 Σύνδεση σε σειρά δύο συστημάτων

Η συνολική συνάρτηση μεταφοράς $G(s)$ δύο συστημάτων Σ_1 και Σ_2 σε σειρά με συναρτήσεις μεταφοράς G_1 και G_2 είναι:

$$G(s) = G_1(s) \cdot G_2(s)$$

Για τον υπολογισμό της συνολικής συνάρτησης μεταφοράς $G(s)$ δυο συστημάτων σε σειρά στην Python, γράφουμε την εντολή series(sys1, sys2) η οποία δέχεται σαν ορίσματα τα συστήματα που επιθυμούμε να συνδέσουμε σε σειρά. Για τη χρήση της απαιτείται η εισαγωγή της βιβλιοθήκης control.

Παράλληλη σύνδεση



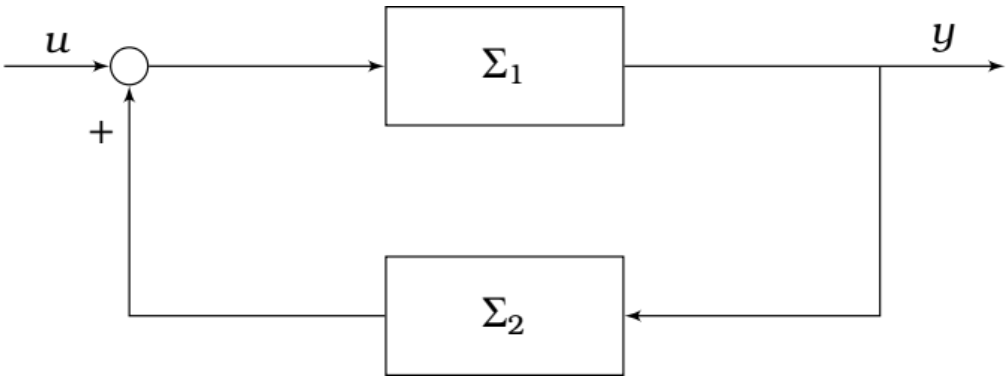
Εικόνα 5.5 Παράλληλη σύνδεση δύο συστημάτων

Η συνολική συνάρτηση μεταφοράς $G(s)$ δύο παράλληλων συστημάτων Σ_1 και Σ_2 με συναρτήσεις μεταφοράς G_1 και G_2 είναι:

$$G(s) = G_1(s) + G_2(s)$$

Στην Python ο υπολογισμός της συνολικής συνάρτησης μεταφοράς $G(s)$ δυο παράλληλα συνδεδεμένων συστημάτων γίνεται με την εντολή parallel(sys1, sys2).

Θετική ανάδραση



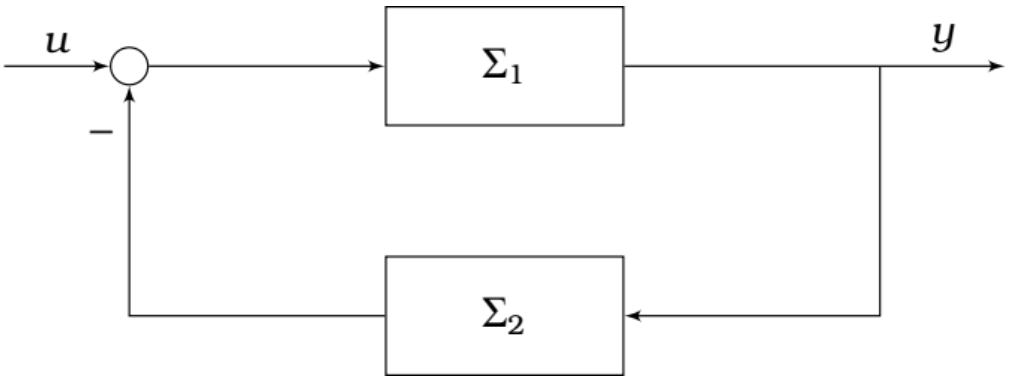
Εικόνα 5.6 Θετική ανάδραση δύο συστημάτων

Η συνολική συνάρτηση μεταφοράς $G(s)$ δύο συστημάτων Σ_1 και ,Σ_2 συνδεδεμένα σε θετική ανάδραση, με συναρτήσεις μεταφοράς G_1 και G_2 είναι:

$$G(s) = \frac{G_1(s)}{1 - G_1(s)G_2(s)}$$

Στην Python ο υπολογισμός της συνολικής συνάρτησης μεταφοράς $G(s)$ δυο συνδεδεμένων συστημάτων σε θετική ανάδραση γίνεται με την εντολή feedback(sys1, sys2, sign=1).

Αρνητική ανάδραση



Εικόνα 5.7 Αρνητική ανάδραση δύο συστημάτων

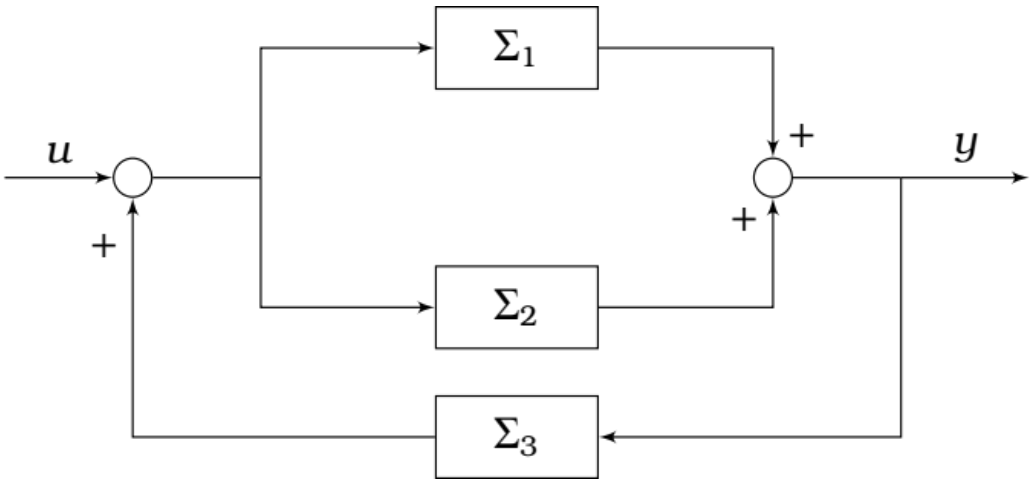
Η συνολική συνάρτηση μεταφοράς $G(s)$ δύο συστημάτων Σ_1 και $,\Sigma_2$ συνδεδεμένα σε αρνητική ανάδραση, με συναρτήσεις μεταφοράς G_1 και G_2 είναι:

$$G(s) = \frac{G_1(s)}{1 + G_1(s)G_2(s)}$$

Στην Python ο υπολογισμός της συνολικής συνάρτησης μεταφοράς $G(s)$ δυο συνδεδεμένων συστημάτων σε αρνητική ανάδραση γίνεται με την εντολή feedback(sys1, sys2, sign=-1).

Παράδειγμα 3

Έστω το παρακάτω σύστημα



$$G_1(s)=\frac{1}{(s+1)(s-1)}\ ,\ G_2(s)=\frac{1}{(s+2)}\ ,\ G_3(s)=\frac{0.5s}{3s^2+5s+1}$$

Ορισμός συναρτήσεων μεταφοράς συστημάτων

```
import control as co

s = co.tf('s')
g1 = 1 / ((s + 1) * (s - 1))
g2 = 1 / (s + 2)
g3 = (0.5 * s) / (3 * s ** 2 + 5 * s + 1)
```

Υπολογισμός ολικής συνάρτησης μεταφοράς

```
g12 = co.parallel(g1, g2)
g123 = co.feedback(g12, g3, 1)
gtot = co.series(g123, g3)
print(gtot)
```

```
             1.5 s^5 + 4 s^4 + 4.5 s^3 + 3 s^2 + 0.5 s

-------------------------------------------------------------------

9 s^7 + 48 s^6 + 80.5 s^5 + 20 s^4 - 74.5 s^3 - 73 s^2 - 21.5 s - 2
```

Παράδειγμα 4

Έστω οι συναρτήσεις μεταφοράς:

$$G_1(s)=\frac{s+2}{5s^3+7s-3}\ ,\ G_2(s)=\frac{1}{(s^2-3s+4)(s+8)}$$

```
import control as co
import matplotlib.pyplot as plt
```

```
s = co.tf('s')
g1 = (s + 2) / (5 * s ** 3 + 7 * s - 3)
g2 = 1 / ((s ** 2 - 3 * s + 4) * (s + 8))
```

Ορίζουμε τρεις τρόπους συνδεσιμότητας (σε σειρά, παράλληλα και με αρνητική ανάδραση)

```
g12s = co.series(g1, g2)
g12p = co.parallel(g1, g2)
g12f = co.feedback(g1, g2, -1)
print(g12s, g12p, g12f)
```

```
                          s + 2

-----------------------------------------------------

5 s^6 + 25 s^5 - 93 s^4 + 192 s^3 - 155 s^2 + 284 s - 96


             s^4 + 12 s^3 - 10 s^2 - s + 61

-----------------------------------------------------

5 s^6 + 25 s^5 - 93 s^4 + 192 s^3 - 155 s^2 + 284 s - 96


            s^4 + 7 s^3 - 10 s^2 - 8 s + 64

-----------------------------------------------------

5 s^6 + 25 s^5 - 93 s^4 + 192 s^3 - 155 s^2 + 285 s - 94
```

Οι πόλοι και τα μηδενικά των $G_1(s)$ και $G_2(s)$

```
# G1(s)
g1_poles = co.pole(g1)
print("g1 poles")
for elem in g1_poles:
    print(f'{elem: .2f}')

g1_zeros = co.zero(g1)
print("g1 zero")
for elem in g1_zeros:
    print(f'{elem: .2f}')

# G2(s)
g2_poles = co.pole(g2)
print("g2 poles")
for elem in g2_poles:
    print(f'{elem: .2f}')

g2_zeros = co.zero(g2)
print("g2 zero")
for elem in g2_zeros:
    print(f'{elem: .2f}')
```

```
g1 poles
-0.19+1.23j
-0.19-1.23j
 0.39+0.00j
g1 zero
-2.00
g2 poles
-8.00+0.00j
 1.50+1.32j
 1.50-1.32j
g2 zero
```

Οι πόλοι και τα μηδενικά των $G_1(s)$ και $G_2(s)$ σε σειρά

```
# G1(s) * G2(s)
g12s_poles = co.pole(g12s)
print("g12s poles")
for elem in g12s_poles:
    print(f'{elem: .2f}')

g12s_zeros = co.zero(g12s)
print("g12s zeros")
for elem in g12s_zeros:
    print(f'{elem: .2f}')
```

```
g12s poles
-8.00+0.00j
 1.50+1.32j
 1.50-1.32j
-0.19+1.23j
-0.19-1.23j
 0.39+0.00j
g12s zeros
-2.00
```

Οι πόλοι και τα μηδενικά των $G_1(s)$ και $G_2(s)$ παράλληλα

```
# G1(s) + G2(s)
g12p_poles = co.pole(g12p)
print("g12p poles")
for elem in g12p_poles:
    print(f'{elem: .2f}')
```

```
g12p_zeros = co.zero(g12p)
print("g12p zeros")
for elem in g12p_zeros:
    print(f'{elem: .2f}')
```

```
g12p poles
-8.00+0.00j
 1.50+1.32j
 1.50-1.32j
-0.19+1.23j
-0.19-1.23j
 0.39+0.00j
g12p zeros
-12.75+0.00j
 1.15+1.33j
 1.15-1.33j
-1.55+0.00j
```

Οι πόλοι και τα μηδενικά των $G_1(s)$ και $G_2(s)$ σε αρνητική ανάδραση

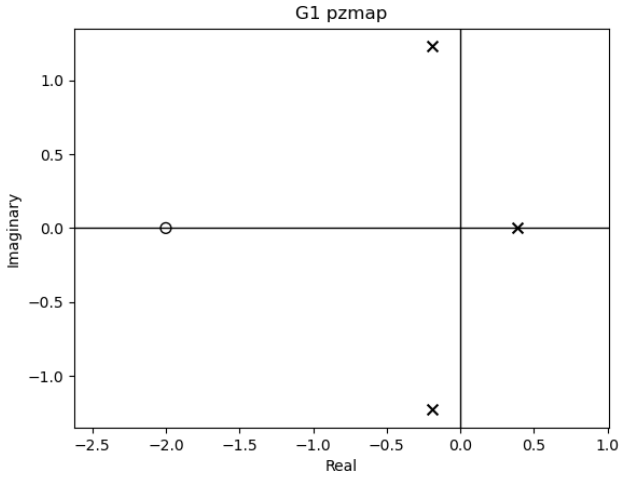
```
# feedback(G1, G2, -1)
g12f_poles = co.pole(g12f)
print("g12f poles")
for elem in g12f_poles:
    print(f'{elem: .2f}')

g12f_zeros = co.zero(g12f)
print("g12f zeros")
for elem in g12f_zeros:
    print(f'{elem: .2f}')
```

```
g12f poles
-8.00+0.00j
 1.50+1.32j
 1.50-1.32j
-0.19+1.23j
-0.19-1.23j
 0.38+0.00j
g12f zeros
-8.00+0.00j
-2.00+0.00j
 1.50+1.32j
```

```
1.50-1.32j
```

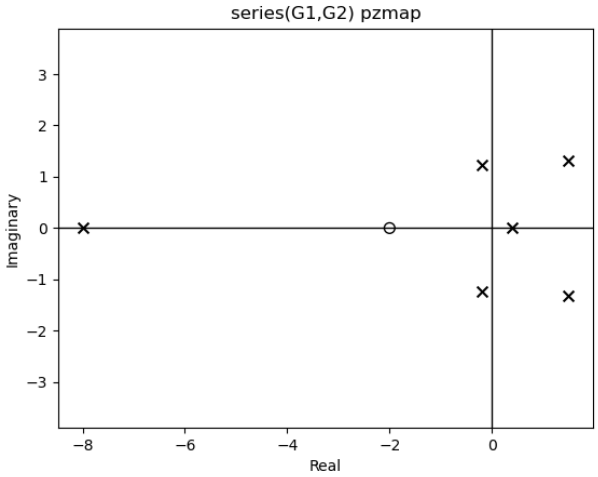
Οι γραφικές παραστάσεις πόλων-μηδενικών για $G_1(s)$, $G_2(s)$ και για τις συνδέσεις τους σε σειρά, παράλληλα και σε αρνητική ανάδραση

```
plt.figure(1)
g1_map = co.pzmap(g1, title='G1 pzmap')

plt.figure(2)
g2_map = co.pzmap(g2, title='G2 pzmap')

plt.figure(3)
g12s_map = co.pzmap(g12s, title='series(G1,G2) pzmap')

plt.figure(4)
g12p_map = co.pzmap(g12p, title='parallel(G1,G2) pzmap')

plt.figure(5)
g12f_map = co.pzmap(g12f, title='feedback(G1,G2,-1) pzmap')

plt.show()
```

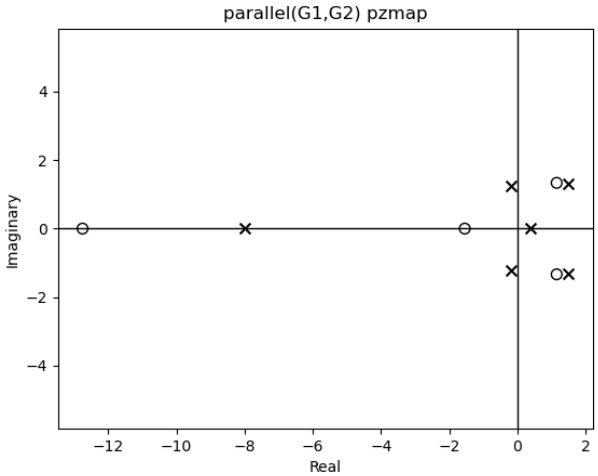


Εικόνα 5.8 Διάγραμμα πόλων-μηδενικών για G1

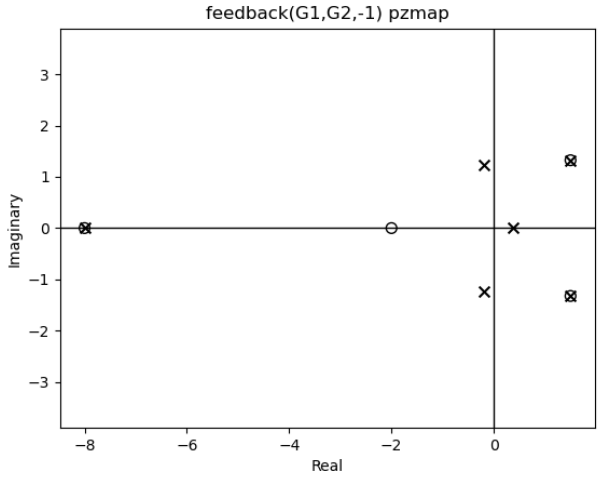
Εικόνα 5.9 Διάγραμμα πόλων-μηδενικών για G2



Εικόνα 5.10 Διάγραμμα πόλων-μηδενικών για G1,G2 σε σειρά



Εικόνα 5.11 Διάγραμμα πόλων-μηδενικών για G1, G2 παράλληλα

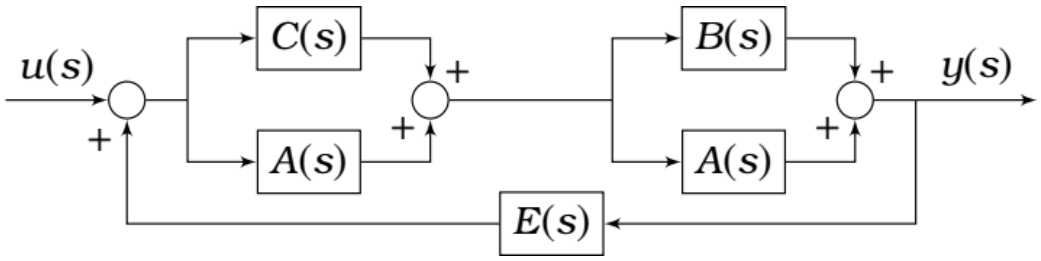


Εικόνα 5.12 Διάγραμμα πόλων-μηδενικών για G1, G2 σε αρνητική ανάδραση

Από τα διαγράμματα των πόλων-μηδενικών συμπεραίνουμε ότι η συνάρτηση μεταφοράς της σειριακής σύνδεσης έχει τους πόλους και τα μηδενικά και των δύο αρχικών συναρτήσεων. Επίσης, παρατηρούμε ότι η συνάρτηση μεταφοράς της παράλληλης σύνδεσης δημιουργεί τρία επιπλέον μηδενικά στα τριών από τους πόλους. Τέλος, η συνάρτηση μεταφοράς της σύνδεσης με αρνητική ανάδραση είναι παρόμοια με την προηγούμενη μόνο που τα επιπλέον μηδενικά είναι μετατοπισμένα προς τα δεξιά, ακριβώς πάνω στους πόλους.

Παράδειγμα 5

Έστω το σύστημα



$$A(s)=\frac{1}{s-3}\ ,\ B(s)=\frac{s+2}{s-2}\ ,\ C(s)=\frac{s+3}{s+4}\ ,\ E(s)=\frac{1}{s(s^2+1)}$$

Οι συναρτήσεις μεταφοράς

```
import control as co
import matplotlib.pyplot as plt
import numpy as np

s = co.tf('s')
```

```
a = 1 / (s - 3)
b = (s + 2) / (s - 2)
c = (s + 3) / (s + 4)
e = 1 / (s * ((s ** 2) + 1))
```

Η ολική συνάρτηση μεταφοράς του συστήματος

```
acp = co.parallel(c, a)
bap = co.parallel(b, a)
acbas = co.series(acp, bap)
acbaef = co.feedback(acbas, e, 1)

print(acbaef)
```

```
s^7 + s^6 - 12 s^5 - 7 s^4 + 27 s^3 - 8 s^2 + 40 s
-------------------------------------------------------------
s^7 - 4 s^6 - 10 s^5 + 61 s^4 - 84 s^3 + 79 s^2 - 64 s – 40
```

Οι πόλοι και τα μηδενικά του συστήματος

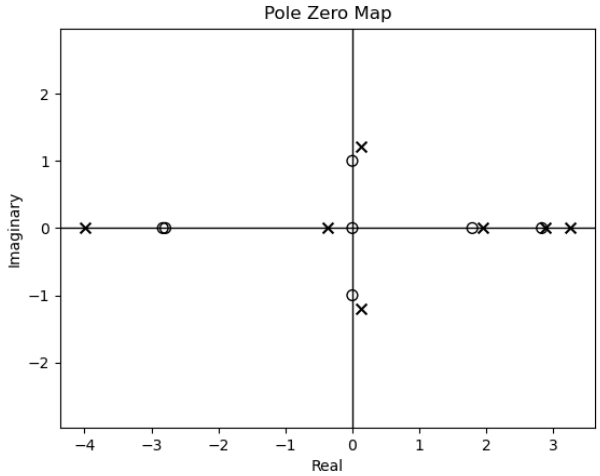
```
g_poles = co.pole(acbaef)
print("G poles")
for elem in g_poles:
    print(f'{elem: .2f}')

g_zeros = co.zero(acbaef)
print("G zeros")
for elem in g_zeros:
    print(f'{elem: .2f}')
```

```
G poles
-4.00+0.00j
 3.26+0.00j
 2.89+0.00j
 1.95+0.00j
 0.14+1.21j
 0.14-1.21j
-0.37+0.00j
G zeros
-2.83+0.00j
-2.79+0.00j
 2.83+0.00j
 1.79+0.00j
```

```
 0.00+1.00j
 0.00-1.00j
 0.00+0.00j
```

Το διάγραμμα πόλων-μηδενικών

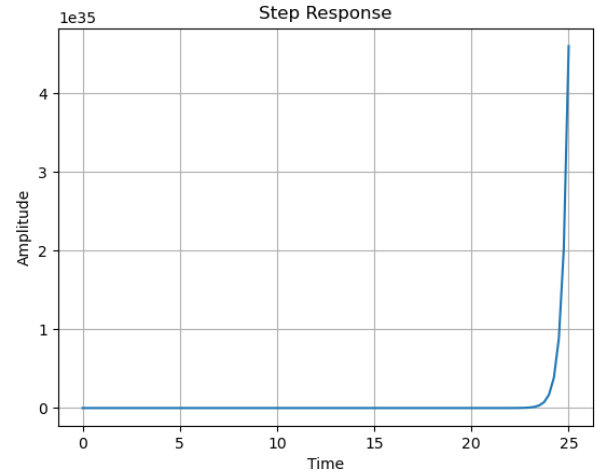
```
g1_map = co.pzmap(acbaef, plot=True)
```



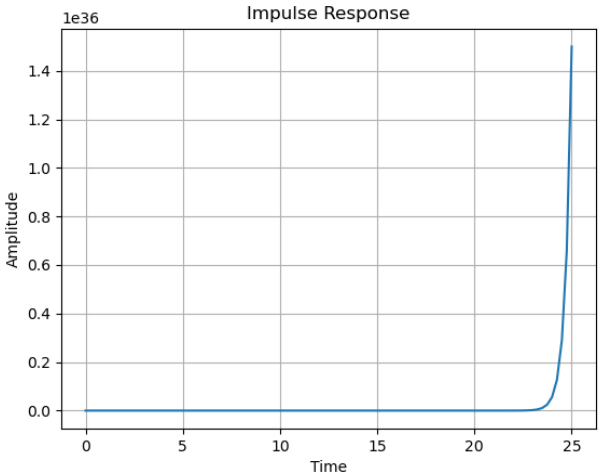
Εικόνα 5.13 Διάγραμμα πόλων-μηδενικών

Τα διαγράμματα της βηματικής και της κρουστικής απόκρισης

```
t1, stp = co.step_response(acbaef, t)
t2, imp = co.impulse_response(acbaef, t)

plt.figure(2)
plt.plot(t1, stp)
plt.title("Step Response")
plt.ylabel("Amplitude")
plt.xlabel("Time")
plt.grid()

plt.figure(3)
plt.plot(t2, imp)
plt.title("Impulse Response")
plt.ylabel("Amplitude")
plt.xlabel("Time")
plt.grid()
plt.show()
```



Εικόνα 5.14 Διάγραμμα βηματικής απόκρισης

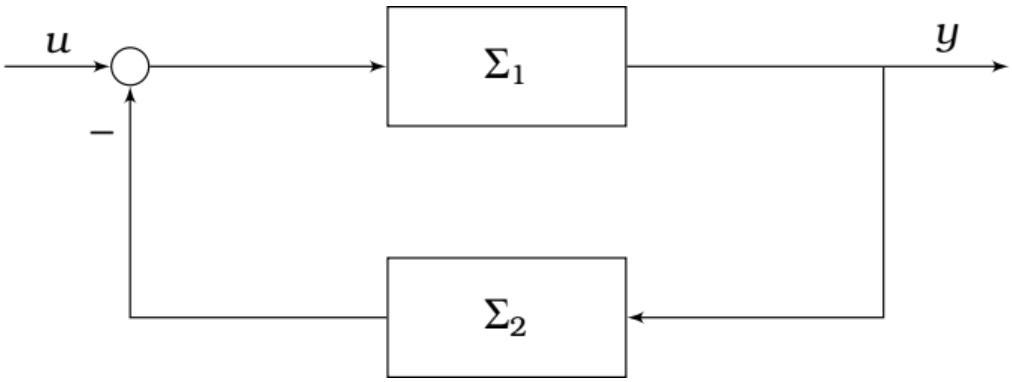


Εικόνα 5.15 Διάγραμμα κρουστικής απόκρισης

Το σύστημα συμπεραίνουμε ότι δεν είναι ευσταθές και αυτό φαίνεται και από τα τρία διαγράμματα καθώς, στο διάγραμμα πόλων-μηδενικών παρατηρούμε ότι η συνάρτηση μεταφοράς του συστήματος παρουσιάζει πόλους στο δεξί μέρος του μιγαδικού επιπέδου αλλά και στα διαγράμματα της βηματικής και κρουστικής απόκρισης παρατηρούμε ότι το πλάτος απειρίζεται με το πέρασμα του χρόνου.

Παράδειγμα 6

Έστω το σύστημα



Όπου $\Sigma_1 : y''(t) - 3y'(t) + 2y(t) = u'(t) + u(t)$ και $\Sigma_2 : y(t) = ku(t),\ k \in \square$

Οι συναρτήσεις μεταφοράς του συστήματος ($k = 2$, $k = 3$, $k = 5$)

```
import control as co
import matplotlib.pyplot as plt
import numpy as np

s = co.tf('s')
g1 = (s + 1) / ((s ** 2) - (3 * s) + 2)

h1 = co.feedback(g1, 2, -1)
h2 = co.feedback(g1, 3, -1)
h3 = co.feedback(g1, 5, -1)
```

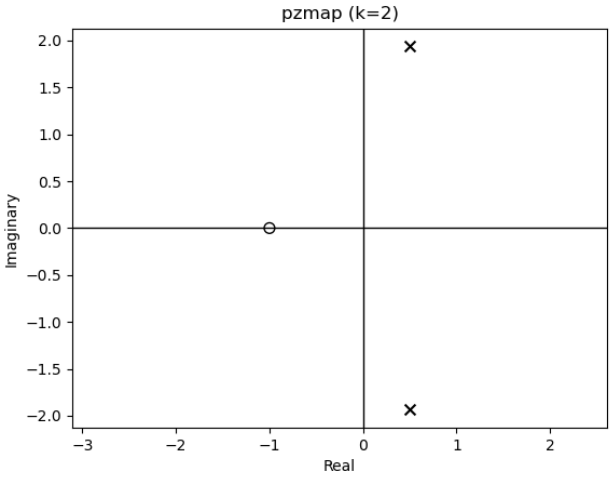
Οι πόλοι του συστήματος για κάθε περίπτωση

```
print("poles (k=2)")
for elem in co.pole(h1):
    print(f'{elem: .2f}')
print("poles (k=3)")
for elem in co.pole(h2):
    print(f'{elem: .2f}')
print("poles (k=5)")
for elem in co.pole(h3):
    print(f'{elem: .2f}')
```

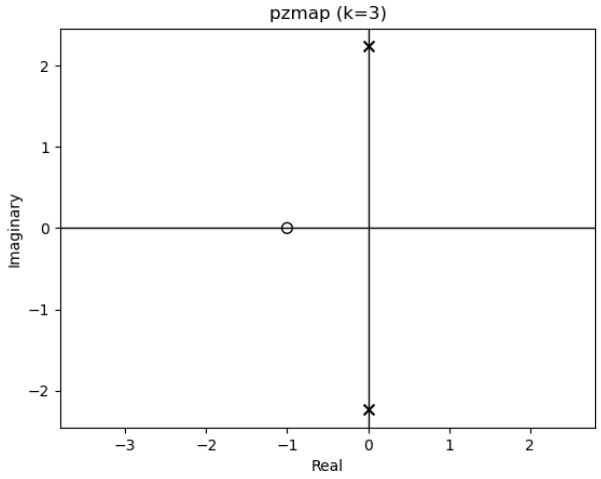
```
poles (k=2)
 0.50+1.94j
 0.50-1.94j
poles (k=3)
-0.00+2.24j
 0.00-2.24j
poles (k=5)
-1.00+2.45j
-1.00-2.45j
```

Τα διαγράμματα πόλων-μηδενικών των συστημάτων

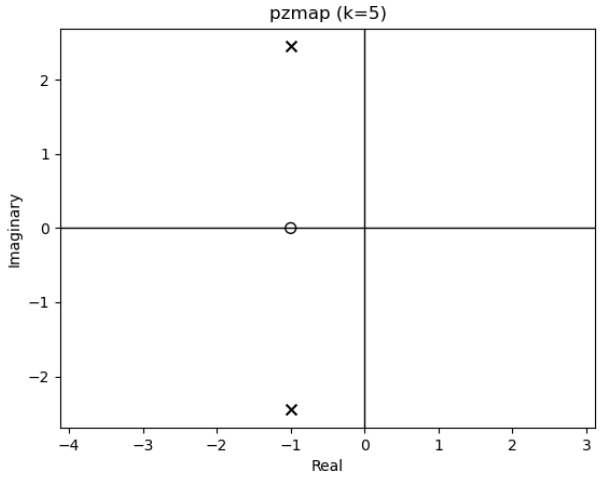
```
plt.figure(1)
h1_map = co.pzmap(h1, plot=True, title='pzmap (k=2)')
plt.figure(2)
h2_map = co.pzmap(h2, plot=True, title='pzmap (k=3)')
plt.figure(3)
h3_map = co.pzmap(h3, plot=True, title='pzmap (k=5)')
```



Εικόνα 5.16 Διάγραμμα πόλων-μηδενικών για k=2



Εικόνα 5.17 Διάγραμμα πόλων-μηδενικών για k=3



Εικόνα 5.18 Διάγραμμα πόλων-μηδενικών για k=5

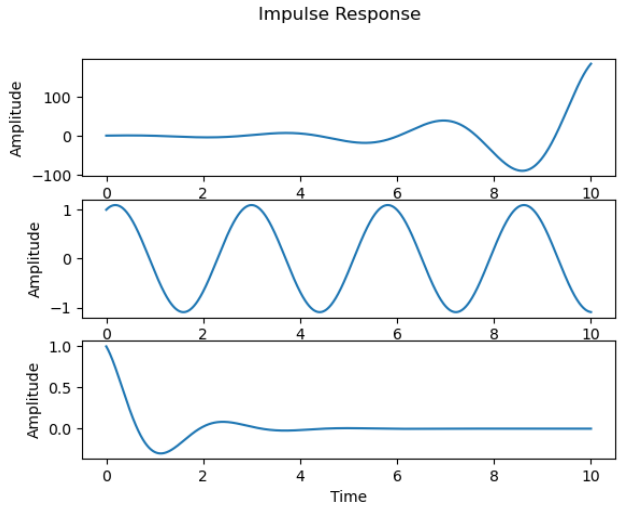
Οι χρονικές αποκρίσεις των συστημάτων

```
t = np.linspace(0, 10, 1000)

t1, imp1 = co.impulse_response(h1, t)
t2, imp2 = co.impulse_response(h2, t)
t3, imp3 = co.impulse_response(h3, t)

t4, stp1 = co.step_response(h1, t)
t5, stp2 = co.step_response(h2, t)
t6, stp3 = co.step_response(h3, t)
```

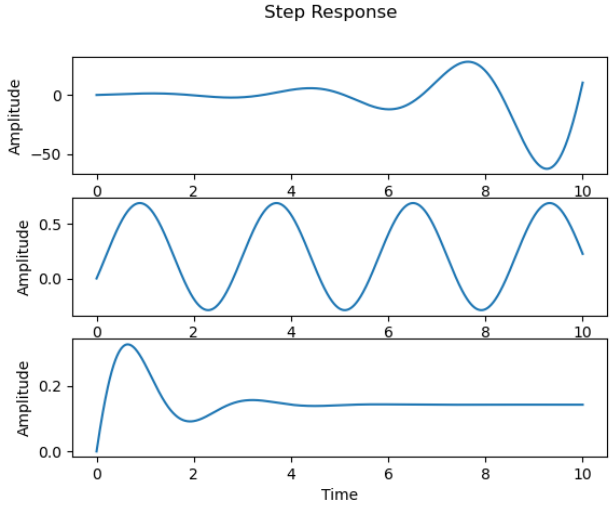
Οι γραφικές παραστάσεις των αποκρίσεων

```
fig1, axs = plt.subplots(3, 1)
fig1.suptitle("Impulse Response")
axs[0].plot(t1, imp1)
axs[1].plot(t2, imp2)
axs[2].plot(t3, imp3)

axs[0].set_ylabel("Amplitude")
axs[1].set_ylabel("Amplitude")
axs[2].set_ylabel("Amplitude")

axs[2].set_xlabel("Time")

fig2, axs = plt.subplots(3, 1)
fig2.suptitle("Step Response")
axs[0].plot(t4, stp1)
axs[1].plot(t5, stp2)
axs[2].plot(t6, stp3)

axs[0].set_ylabel("Amplitude")
axs[1].set_ylabel("Amplitude")
```

```
axs[2].set_ylabel("Amplitude")

axs[2].set_xlabel("Time")

plt.show()
```



Εικόνα 5.19 Διαγράμματα κρουστικής απόκρισης για k=2, k=3, k=5



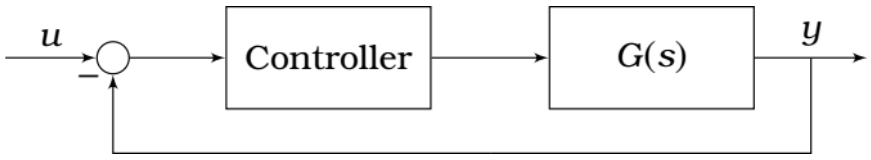
Εικόνα 5.20 Διαγράμματα βηματικής απόκρισης για k=2, k=3, k=5

Από τα διαγράμματα συμπεραίνουμε ότι για $k=2$ το σύστημα είναι ασταθές καθώς στο διάγραμμα πόλων-μηδενικών παρατηρούμε δύο πόλους στο δεξί μέρος του μιγαδικού επιπέδου όπως επίσης και στις γραφικές παραστάσεις της βηματικής και κρουστικής απόκρισης παρατηρούμε ότι η καμπύλες έχουν την τάση να αυξάνουν το πλάτος τους στο πέρασμα του χρόνου. Για $k=3$, παρατηρούμε πως στο διάγραμμα πόλων-μηδενικών, οι πόλοι του συστήματος βρίσκονται επάνω στον άξονα των φανταστικών τιμών, δηλαδή το πραγματικό μέρος των πόλων είναι μηδέν, ενώ στα διαγράμματα της κρουστικής και βηματικής του απόκρισης παρατηρούμε ότι το σήμα

ταλαντώνεται γύρο από ένα σημείο ισορροπίας με σταθερό πλάτος. Τέλος, για $k = 5$ στο διάγραμμα πόλων-μηδενικών βλέπουμε τους πόλους του συστήματος να έχουν μετατοπιστεί στο αριστερό μέρος του μιγαδικού επιπέδου ενώ στα διαγράμματα της κρουστικής και βηματικής απόκρισης βλέπουμε το πλάτος του σήματος της εξόδου τείνει στο 0. Έτσι, μπορούμε να πούμε ότι το σύστημα για $k = 3$ και για $k = 5$ είναι ευσταθές.

6. Γεωμετρικός τόπος ριζών

Έστω ένα σύστημα το οποίο περιγράφεται από το παρακάτω σχήμα



Όπως είναι φανερό, το σύστημα έχει ανάδραση. Τα συστήματα αυτά ονομάζονται κλειστά. Τα συστήματα τα οποία δεν έχουν ανάδραση ονομάζονται ανοιχτά συστήματα. Το σήμα εισόδου του συστήματος $G(s)$ πολλαπλασιάζεται από έναν ελεγκτή. Έτσι αν το σύστημα έχει συνάρτηση μεταφοράς $G(s)=\frac{Y(s)}{U(s)}$ τότε η συνάρτηση μεταφοράς του ολικού συστήματος λόγο του ελεγκτή και της ανάδρασης είναι $T(s)=\frac{G}{1+GK}=\frac{KY(s)}{U(s)+Y(s)K}$. Ο ελεγκτής αποτελεί ένα σύστημα, με τη χρήση του οποίου, ορίζουμε στο κλειστό σύστημα τις ιδιότητες που επιθυμούμε. Όταν ο ελεγκτής είναι πραγματικός αριθμός τότε ονομάζεται συντελεστής κέρδους (K). Οι ρίζες της εξίσωσης $U(s)+Y(s)K=0$ ονομάζονται πόλοι του κλειστού συστήματος και η θέση τους στο μιγαδικό επίπεδο μεταβάλλεται όσο το K αλλάζει τιμές. Το διάγραμμα που δείχνει τον τρόπο με τον οποίο μεταβάλλονται οι πόλοι του κλειστού συστήματος στο μιγαδικό επίπεδο ανάλογα με τις τιμές του K λέγεται γεωμετρικός τόπος ριζών.[10]

6.1. Κανόνες κατασκευής γεωμετρικού τόπου ριζών

Ορισμένοι κανόνες για την κατασκευή γεωμετρικού τόπου ριζών (γ.τ.ρ) [11]

- Οι κλάδοι του γ.τ.ρ έχουν πλήθος ίσο με $\max\{n_p, n_z\}$ και αρχίζουν από τους πόλου το ανοιχτού συστήματος για K κοντά στο 0 και καταλήγουν είτε στα μηδενικά του συστήματος είτε στο άπειρο.
- Ο γ.τ.ρ είναι συμμετρικός ως προς τον πραγματικό άξονα .

[10] Δρ. Βολογιαννίδης Σταύρος, *Συστήματα Αυτομάτου Ελέγχου Θεωρία και Εφαρμογές, Διδακτικές Σημειώσεις Τμήματος Πληροφορικής και Επικοινωνιών*, (σελ 57-58)
[11] Δρ. Βολογιαννίδης Σταύρος, *Συστήματα Αυτομάτου Ελέγχου Θεωρία και Εφαρμογές, Διδακτικές Σημειώσεις Τμήματος Πληροφορικής και Επικοινωνιών*, (σελ 58-59)

- Ένα τμήμα του πραγματικού άξονα μπορεί να είναι μέρος του γ.τ.ρ αν ο αριθμός των πραγματικών πόλων και μηδενικών του συστήματος που βρίσκονται δεξιά του τμήματος είναι περιττός.
- Σε περίπτωση που δύο πραγματικοί πόλοι ή δύο πραγματικά μηδενικά του ανοιχτού συστήματος είναι τοποθετημένα το ένα δίπλα στο άλλο, στον άξονα των πραγματικών αριθμών και το διάστημα μεταξύ τους είναι μέρος του γ.τ.ρ τότε υπάρχει σημείο μεταξύ των πόλων ή των ριζών από το οποίο ο κλάδος φεύγει ή έρχεται αντίστοιχα.

Ο γεωμετρικός τόπος ριζών στην Python παράγεται με την εντολή της βιβλιοθήκης control root_locus(system) η οποία δέχεται ως όρισμα τη συνάρτηση ενός συστήματος. Στο διάγραμμα που δημιουργείται μπορούμε να μετακινήσουμε τους πόλους στο και να δούμε την τιμή του συντελεστή κέρδους που πρέπει να ορίσουμε στο σύστημα ώστε να μεταβάλουμε τον πόλο στη συγκεκριμένη τιμή.

Παράδειγμα 1

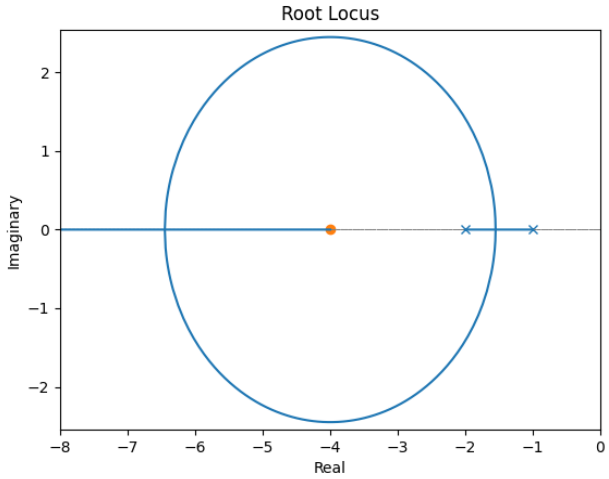
Έστω οι συναρτήσεις

$$G_1(s)=\frac{s+4}{(s+2)(s+1)}\ ,\ G_2(s)=\frac{s^2+3s+4}{(s+2)(s^2+1)}\ ,\ G_3(s)=\frac{2s+1}{2s^2+4s+8}$$

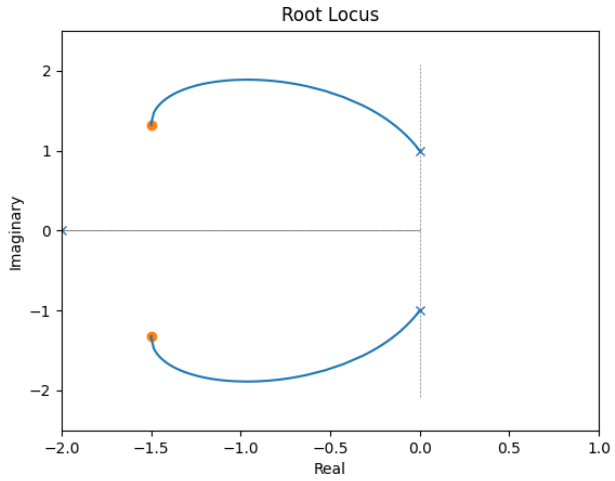
Οι γεωμετρικοί τόποι ριζών για κάθε συνάρτηση θα είναι

```
import control as co
import matplotlib.pyplot as plt

s = co.tf('s')
g1 = (s + 4) / ((s + 2) * (s + 1))
g2 = (s ** 2 + 3 * s + 4) / ((s + 2) * (s ** 2 + 1))
g3 = (2 * s + 1) / (s ** 2 + 4 * s + 8)

plt.figure(1)
g1_rlocus = co.rlocus(g1)
plt.xlim([-8, 0])

plt.figure(2)
g2_rlocus = co.rlocus(g2)
plt.xlim([-2, 1])

plt.figure(3)
g3_rlocus = co.rlocus(g3)
plt.xlim([-4, 0])
```

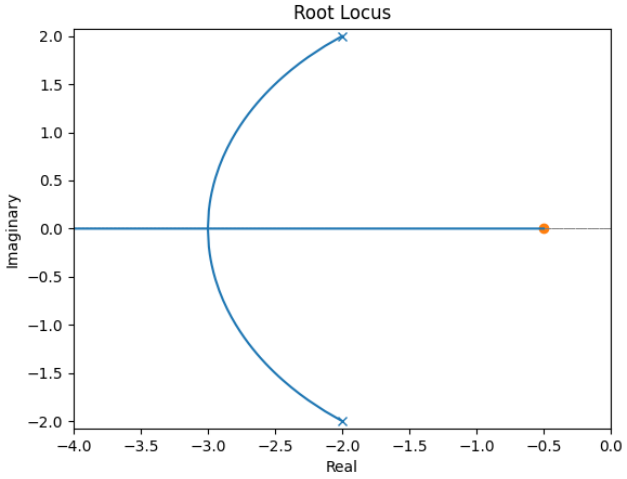
```
plt.show()
```



Εικόνα 6.1 Γεωμετρικός τόπος ριζών $G_1(s)$



Εικόνα 6.2 Γεωμετρικός τόπος ριζών $G_2(s)$



Εικόνα 6.3 Γεωμετρικός τόπος ριζών $G_3(s)$

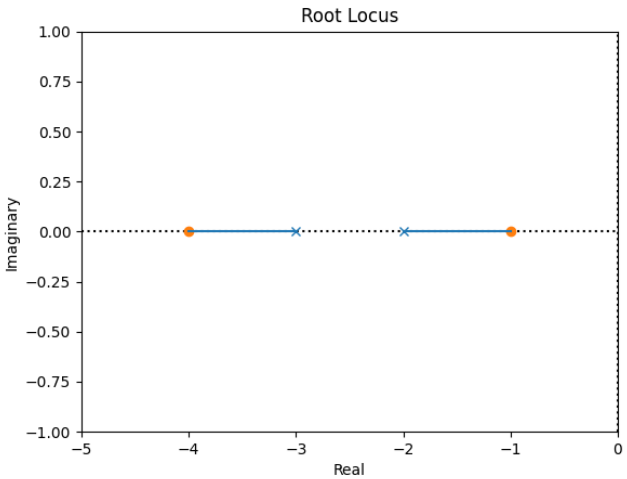
Παράδειγμα 2

Έστω οι συναρτήσεις

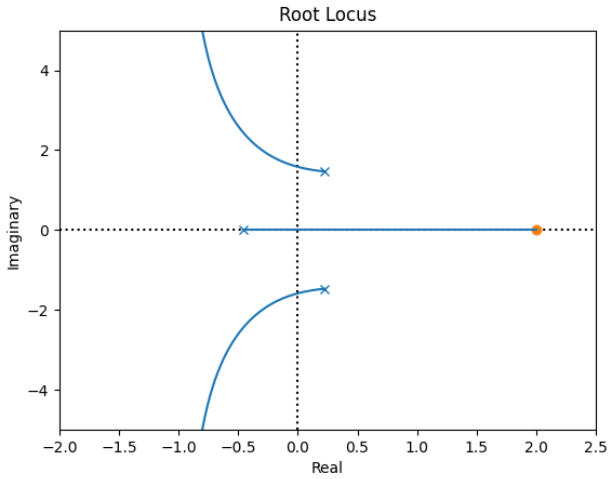
$$G_1(s)=\frac{(s+1)(s+4)}{s^2+5s+6},\ G_2(s)=\frac{s-2}{s^3+2s+1}\ ,\ G_3(s)=\frac{(s-2)(s^2+2)}{(s-3)(s+1)}$$

Οι γεωμετρικοί τόποι των ριζών για κάθε συνάρτηση είναι

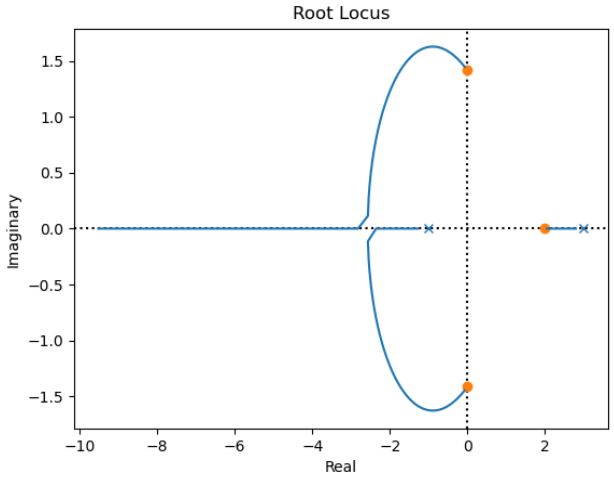
```
import control as co
import matplotlib.pyplot as plt

s = co.tf('s')

g1 = ((s + 1) * (s + 4)) / (s ** 2 + 5 * s + 6)
plt.figure(1)
g1_rlist, g1_klist = co.root_locus(g1, grid=False)
plt.xlim([-5, 0])

g2= (s-2)/(s**3+2*s+1)
plt.figure(2)
g2_rlist =co.root_locus(g2, grid=False)
plt.xlim([-2, 2.5])
plt.ylim([-5, 5])

plt.figure(3)
g3 = (s-2)*((s**2)+2)/((s-3)*(s+1))
rlist, klist = co.root_locus(g3,kvect=np.linspace(0.1,10,5001),
grid=False)


plt.show()
```



Εικόνα 6.4 Γεωμετρικός τόπος ριζών G1(s)



Εικόνα 6.5 Γεωμετρικός τόπος ριζών $G_2(s)$



Εικόνα 6.6 Γεωμετρικός τόπος ριζών $G_3(s)$

Παράδειγμα 3

Έστω ένα σύστημα με συνάρτηση μεταφοράς $G(s) = \dfrac{s+1}{s^2 - 5s + 6}$

Ορίζουμε τη συνάρτηση μεταφοράς

```
import control as co
import matplotlib.pyplot as plt
import numpy as np


s = co.tf('s')
g = (s + 1) / (s ** 2 - 5 * s + 6)
```

Υπολογίζουμε τους πόλους του ανοιχτου συστήματος

```
g_poles = co.pole(g)
```

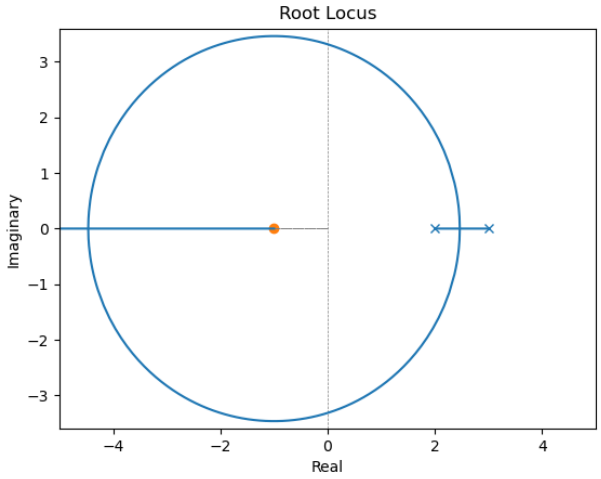
```
[3. 2.]
```

Η συνάρτηση μεταφοράς και οι πόλοι του κλειστού συστήματος

```
gf = co.feedback(g, sign=-1)
print(co.pole(gf))
[2.+1.73205081j 2.-1.73205081j]
```

Ο γεωμετρικός τόπος των ριζών

```
plt.figure(1)
grtl = co.root_locus(g, xlim=(-5, 5))
plt.show()
```



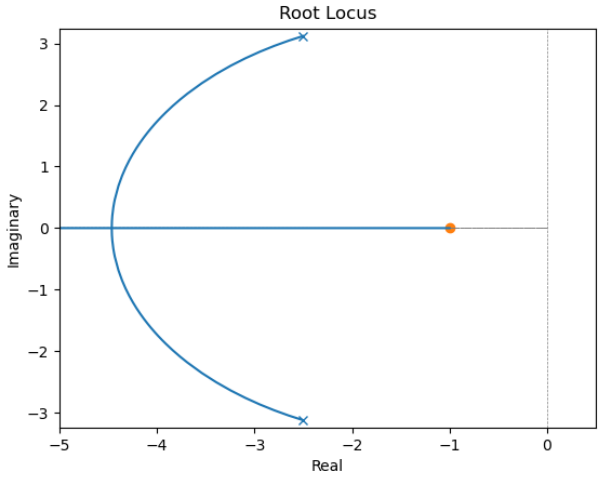
Εικόνα 6.7 Γεωμετρικός τόπος ριζών του ανοιχτού συστήματος

Για k=10 το σύστημα είναι ευσταθές και εχει την εξής συνάρτηση μεταφοράς

```
g_gained = co.feedback(10 * g, sign=-1)
print(g_gained)
print(co.pole(g_gained))
[-2.5+3.122499j -2.5-3.122499j]


  10 s + 10

--------------

s^2 + 5 s + 16
```

Ο γεωμετρικός τόπος ριζών του κλειστού συστήματος με k=10

```
gg_rlc = co.root_locus(g_gained, xlim=(-5, 0.5))
```



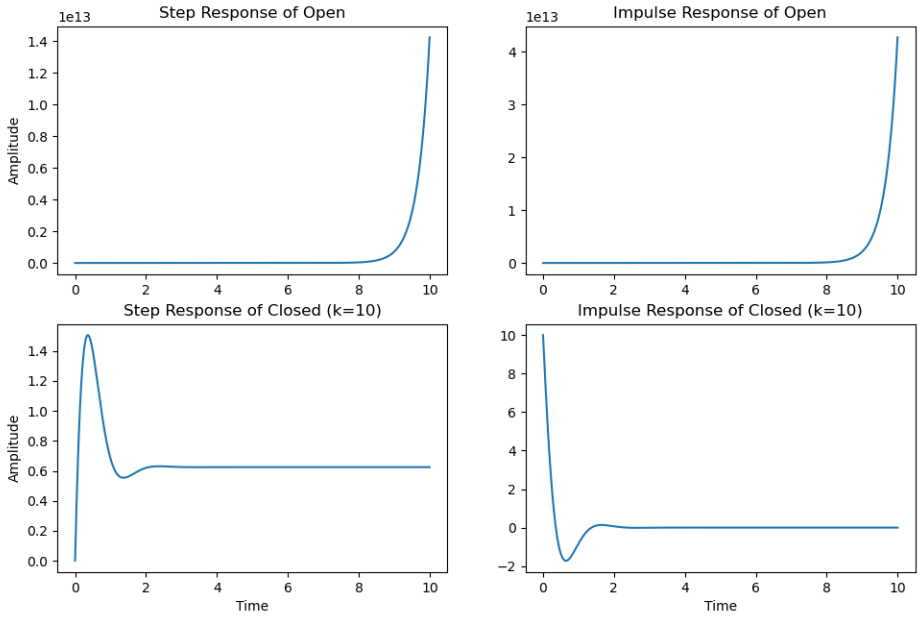
Εικόνα 6.8 Γεωμετρικός τόπος ριζών του κλειστού συστήματος με k=10

Οι αποκρίσεις του ανοιχτού και του κλειστού συστήματος

```
t = np.linspace(0, 10, 1000)

t1, gstp = co.step_response(g, t)  # step response of open system
t2, gimp = co.impulse_response(g, t)  # impulse response of open
system
t3, ggstp = co.step_response(g_gained, t)  # step response of
closed system k=10
t4, ggimp = co.impulse_response(g_gained, t)  # impulse response
of closed system k=10
```

οι γραφικές παραστάσεις των αποκρίσεων

```
fig3, axs = plt.subplots(2, 2)
axs[0, 0].plot(t1, gstp)
axs[0, 1].plot(t2, gimp)
axs[1, 0].plot(t3, ggstp)
axs[1, 1].plot(t4, ggimp)

axs[0, 0].title.set_text('Step Response of Open')
axs[0, 1].title.set_text('Impulse Response of Open')
axs[1, 0].title.set_text('Step Response of Closed (k=10)')
axs[1, 1].title.set_text('Impulse Response of Closed (k=10)')

axs[0, 0].set_ylabel('Amplitude')
axs[1, 0].set_ylabel('Amplitude')
axs[1, 0].set_xlabel('Time')
axs[1, 1].set_xlabel('Time')
plt.show()
```



Εικόνα 6.9 Κρουστικές και βηματικές αποκρίσεις για το ανοιχτό και το κλειστό σύστημα

Τα χαρακτηριστικά της βηματικής απόκρισης του κλειστού συστήματος

```
def step_info(t, yout):
    print(f"Max Amp: {max(yout)}")
    print("OS: %f%s" % ((yout.max() / yout[-1] - 1) * 100, '%'))
    print("Tr: %fs" % (t[next(i for i in range(0, len(yout) - 1)
if yout[i] > yout[-1] * .90)] - t[0]))
    print("Ts: %fs" % (t[next(len(yout) - i for i in range(2,
len(yout) - 1) if abs(yout[-i] / yout[-1]) > 1.02)] - t[0]))

step_info(t3, ggstp)
```

```
Max Amp: 1.5060462140945594

OS: 140.967394%

Tr: 0.070070s

Ts: 1.051051s
```

Παράδειγμα 3

Έστω ένα σύστημα που περιγράφεται από την διαφορική εξίσωση: $y''(t)-5y'(t)-24y(t)=u''(t)+5u'(t)+4u(t)$

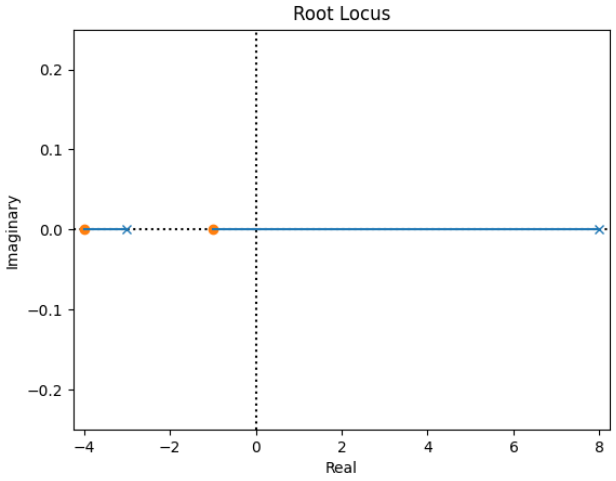
Εισάγουμε τη συνάρτηση μεταφοράς

```
import control as co
import matplotlib.pyplot as plt
```

```
import numpy as np

s = co.tf('s')
g = (s**2+5*s+4)/(s**2-5*s-24)
```

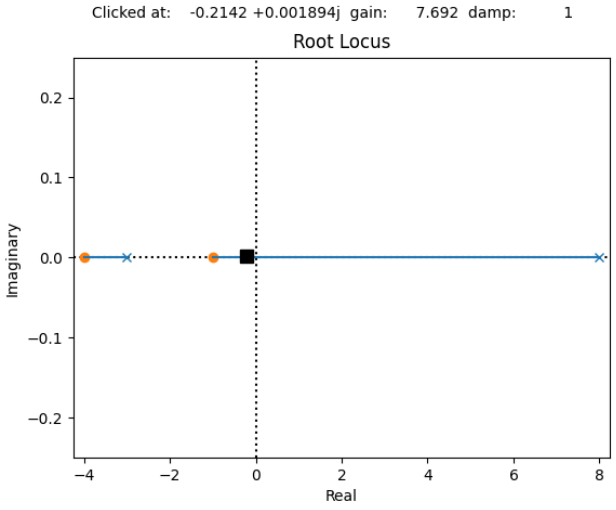
Ο γεωμετρικός τόπος ριζών του συστήματος και οι πόλοι του

```
print(co.pole(g))
co.root_locus(g, ylim=(-0.25, 0.25), grid=False)
[ 8. -3.]
```



Εικόνα 6.10 Γεωμετρικός τόπος ριζών του ανοιχτού συστήματος

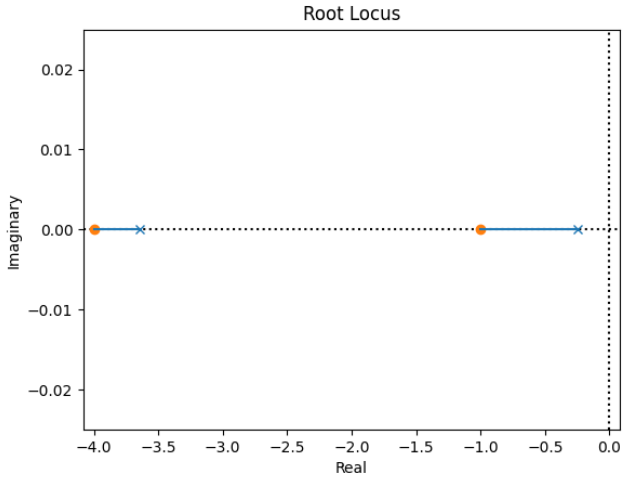
Το σύστημα έχει πόλους στα σημεία 8 και -3, που σημαίνει ότι το σύστημα είναι ασταθές. Αυτό μπορούμε να το δούμε και από το διάγραμμα γεωμετρικών τόπων ριζών του συστήματος καθώς ο πόλος με τιμή 8 βρήσκεται στο δεξί μέρος του μιγαδικού επιπέδου. Επίσης, παρατηρούμε ότι για κάποια τιμή k το σύστημα μπορεί να γίνει ευσταθές, καθώς ο πόλος που βρίσκεται με τιμή 8 παίρνει αρνητική τιμή.



Εικόνα 6.11 Γεωμετρικός τόπος ριζών με μελέτη για κ

Άρα για k > 7 περίπου το σύστημα γίνεται ευσταθές. Για k=8 ο γεωμετρικός τόπος ριζών του κλειστού συστήματος είναι:

```
g_gained = co.feedback(8 * g, sign=-1)
plt.figure(2)
co.root_locus(g_gained, ylim=(-0.025, 0.025), grid=False)
```

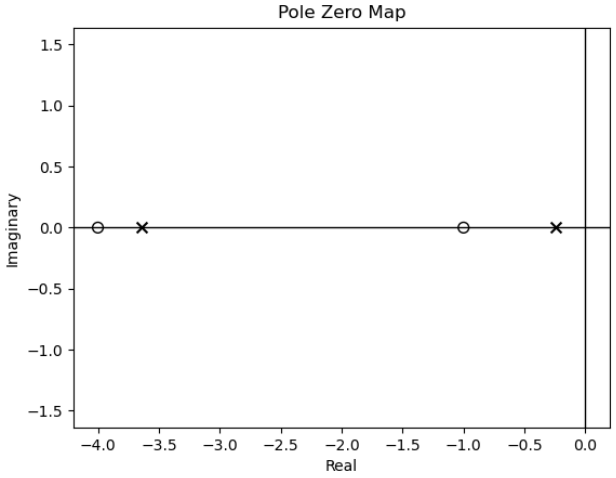


Εικόνα 6.12 Γεωμετρικός τόπος ριζών του κλεστού συστήματος με k=8

Η συνάρτηση του κλειστού συστήματος και το διάγραμμα πόλων-μηδενικών

```
print(g_gained)
plt.figure(3)
co.pzmap(g_gained)
plt.show()
```

```
8 s^2 + 40 s + 32

-----------------

9 s^2 + 35 s + 8
```



Εικόνα 6.13 Διάγραμμα πόλων-μηδενικών του κλειστού συτήματος με k=8

6.2. Το εργαλείο SISOTOOL

Το SISOTOOL είναι ένα χρήσιμο εργαλείο για τη σχεδίαση ελεγκτών με τη μέθοδο του γεωμετρικού τόπου ριζών. Για να χρησιμοποιήσουμε το περιβάλλον του sisotool εισάγουμε τη συνάρτηση μεταφοράς του συστήματος και χρησιμοποιούμε την εντολή co.sisotool της βιβλιοθήκης control. Το sisotool ανοίγει ένα νέο διαδραστικό παράθυρο το οποίο έχει 4 διαγράμματα που περιγράφουν το κλειστό σύστημα (με προκαθορισμένη συνάρτηση ανάδρασης $H(s)=1$ και αρνητική ανάδραση). Αριστερά έχει τα διαγράμματα Bode για την συχνοτική ανάλυση του συστήματος, δεξιά και πάνω έχει τον γεωμετρικό τόπο των ριζών (root locus) και δεξιά κάτω την βηματική απόκριση του συστήματος. Στο διάγραμμα root locus επιλέγουμε με αριστερό κλικ τους πόλους πάνω στον γεωμετρικό τόπο και τα υπόλοιπα διαγράμματα επανασχεδιάζονται κατάλληλα. Σύροντας τους πόλους του συστήματος στο αριστερό μιγαδικό επίπεδο του γεωμετρικού ριζών, εξασφαλίζουμε την ευστάθεια του συστήματος για διάφορες τιμές του συντελεστή κέρδους C. Στην περίπτωση που το σύστημα δεν είναι σταθεροποιήσιμο αλλάζοντας το συντελεστή κέρδους C θα πρέπει να προσθέσουμε μηδενικά ή πόλους αλλάζοντας ουσιαστικά τη δυναμική συμπεριφορά του αρχικού συστήματος προς όφελός μας.

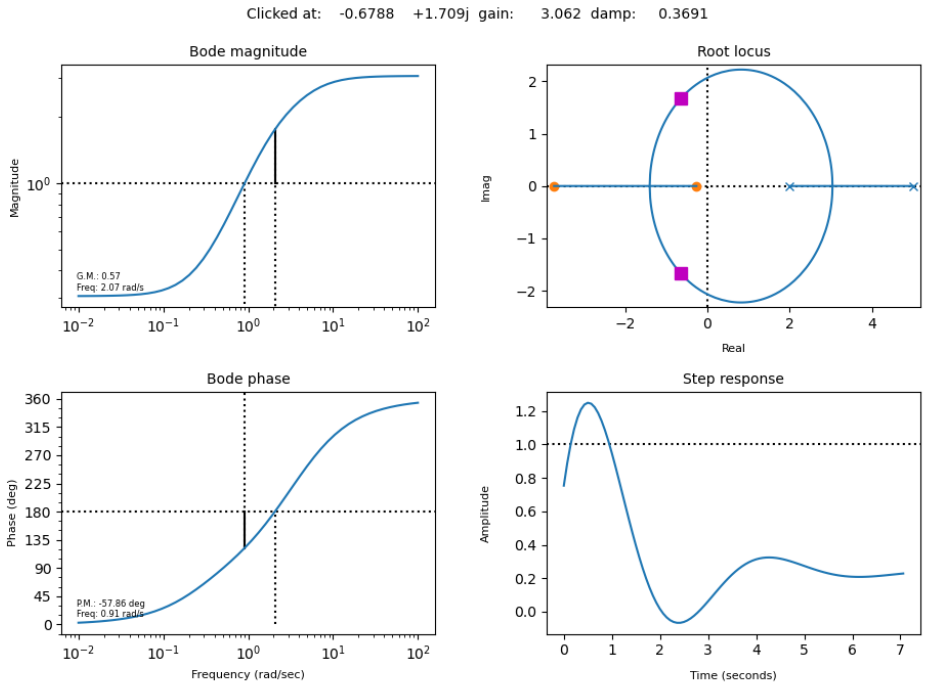
Παράδειγμα 4

Έστω ένα σύστημα με συνάρτηση μεταφοράς $G(s)=\dfrac{s^2+4s+1}{s^2-7s+10}$

```
import control as co
import matplotlib.pyplot as plt
import numpy as np

s = co.tf('s')
g = (s ** 2 + 4 * s + 1) / (s ** 2 - 7 * s + 10)
```

Η δυναμική συμπεριφορά του συστήματος με τη βοήθεια του sisotool.

```
siso = co.sisotool(g)
```



Εικόνα 6.14 Διαγράμματα Sisotool του συστήματος G(s)

Από το sisotool διαπιστώνουμε ότι για K=3 οι πόλοι βρίσκονται στο αριστερό μιγαδικό επίπεδο. Επίσης, παρατηρούμε στο διάγραμμα της κρουστικής απόκρισης ότι το σήμα εξόδου του κλειστού συστήματος έρχεται σε ισορροπία σε περίπου 5sec. Άρα το σύστημα είναι ευσταθές.

Η συνάρτηση μεταφοράς του κλειστού συστήματος με C=3

```
fg = co.feedback(3 * g, sign=-1)
```

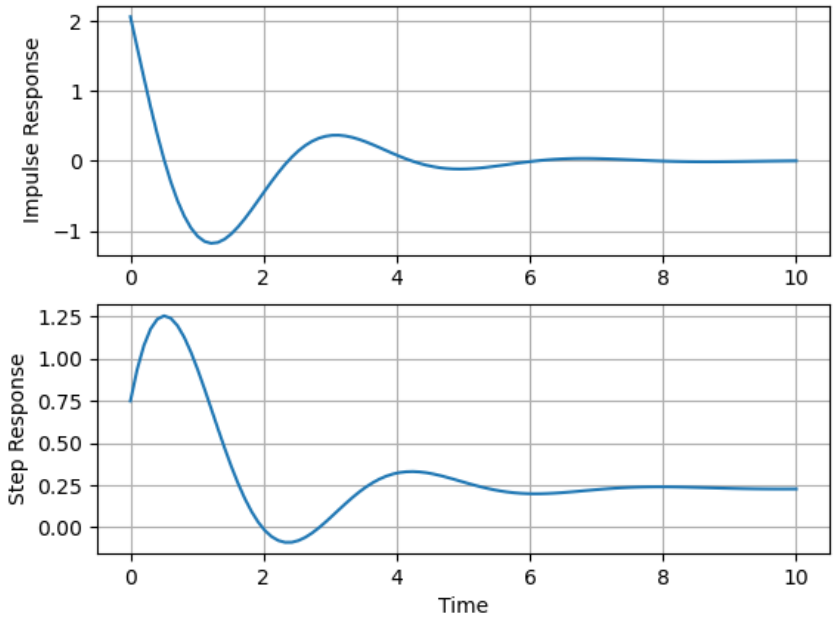
Η κρουστική και η βηματική απόκριση του κλειστού συστήματος με C=3

```
t = np.linspace(0, 10, 100)

t1, impg = co.impulse_response(fg, t)
t2, stpg = co.step_response(fg, t)
```

Οι γραφικές παραστάσεις των αποκρίσεων

```
fig, ax = plt.subplots(2,1)
ax[0].plot(t1, impg)
ax[1].plot(t2, stpg)

ax[0].set_ylabel("Impulse Response")
ax[1].set_ylabel("Step Response")
ax[1].set_xlabel("Time")
ax[0].grid()
```

```
ax[1].grid()
plt.show()
```



Εικόνα 6.15 Κρουστική και βηματική απόκριση του συστήματος

Τα χαρακτηριστικά της βηματικής απόκρισης

```
def step_info(t, yout):
    print(f"Max Amp: {max(yout)}")
    print("OS: %f%s" % ((yout.max() / yout[-1] - 1) * 100, '%'))
    print("Tr: %fs" % (t[next(i for i in range(0, len(yout) - 1)
if yout[i] > yout[-1] * .90)] - t[0]))
    print("Ts: %fs" % (t[next(len(yout) - i for i in range(2,
len(yout) - 1) if abs(yout[-i] / yout[-1]) > 1.02)] - t[0]))

step_info(t2, stpg)
```

```
Max Amp: 1.2557270647475887

OS: 451.081247%

Tr: 0.000000s

Ts: 8.888889s
```

Παράδειγμα 5

Έστω σύστημα με συνάρτηση μεταφοράς $G(s) = \frac{s+1}{s^2+6s-5}$

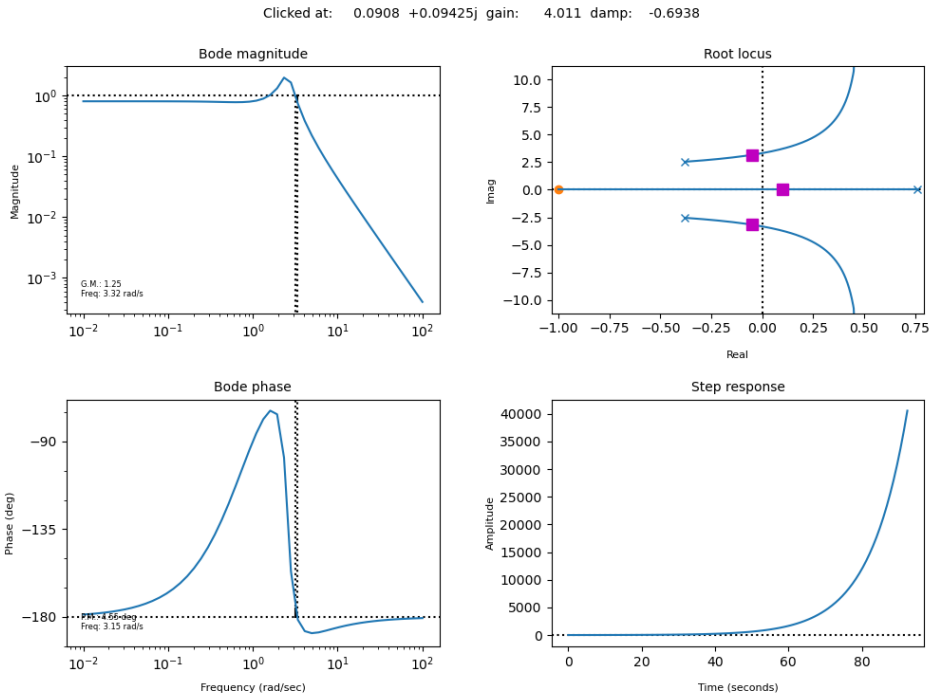
Εισάγουμε τη συνάρτηση μεταφοράς στην Python

```
import control as co
import matplotlib.pyplot as plt
```

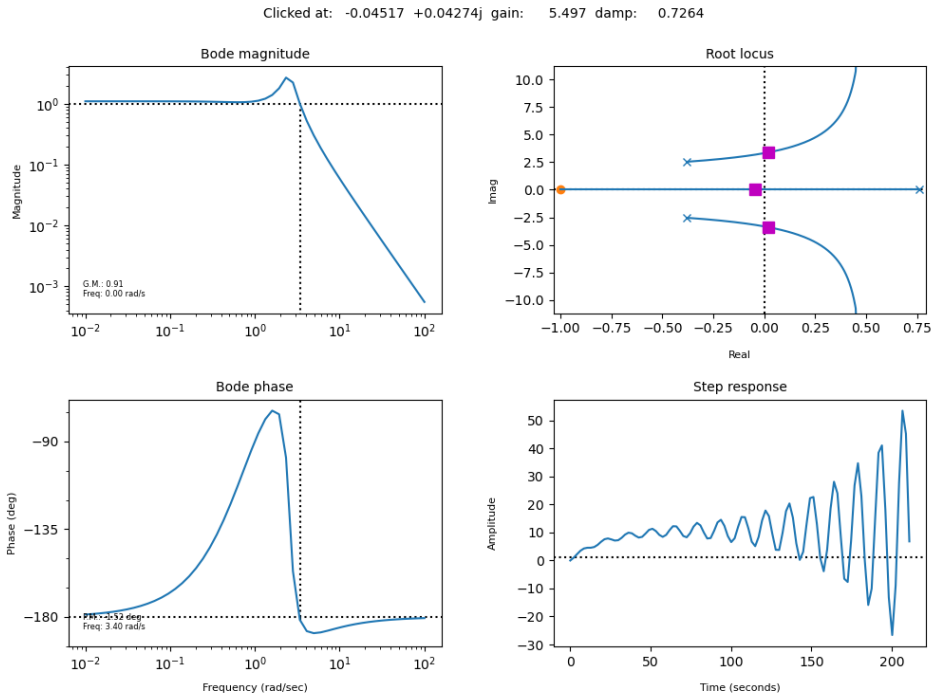
```
s = co.tf('s')
g = (s + 1) / (s ** 3 + 6 * s - 5)
```

Εισάγουμε τη συνάρτηση μεταφοράς στο sisotool

```
plt.figure(1)
siso = co.sisotool(g)
```



Εικόνα 6.16 Διαγράμματα sisotool περίπτωση 1



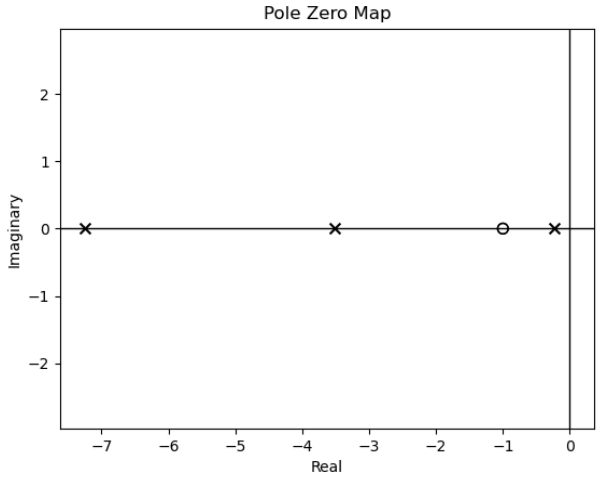
Εικόνα 6.17 Διαγράμματα sisotool περίπτωση 2

Απ' το sisotool διαπιστώνουμε ότι δεν υπάρχει συντελεστής κέρδους ώστε το σύστημα να είναι σταθεροποιήσιμο (Για $K \approx 4.8$ οι πόλοι ανταλλάσουν θέσεις στο μιγαδικό επίπεδο). Για αυτό θα επέμβουμε στο σύστημα εισάγοντας μηδενικά. Δοκιμάζοντας τιμές, προσθέτουμε ένα πραγματικό μηδενικό στη θέση σ=-1. Έτσι εισάγουμε την πολυωνυμική συνάρτηση ελέγχου $C(s) = 11(s+1)$.

Η συνάρτηση μεταφοράς του κλειστού συστήματος και το διάγραμμα πόλων-μηδενικών

```
# control function
k = 11
z = -1
cs = k * (s - z)

# the controlled system
g_feedback = co.feedback(co.series(g, cs), 1, sign=-1)
print(g_feedback)

# pole zero plot of the closed controlled system
plt.figure(3)
co.pzmap(g_feedback)
plt.show()
```

```
  11 s^2 + 22 s + 11
----------------------
s^3 + 11 s^2 + 28 s + 6
```



Εικόνα 6.18 Διάγραμμα πόλων-μηδενικών του κλειστού συστήματος

Παράδειγμα 6

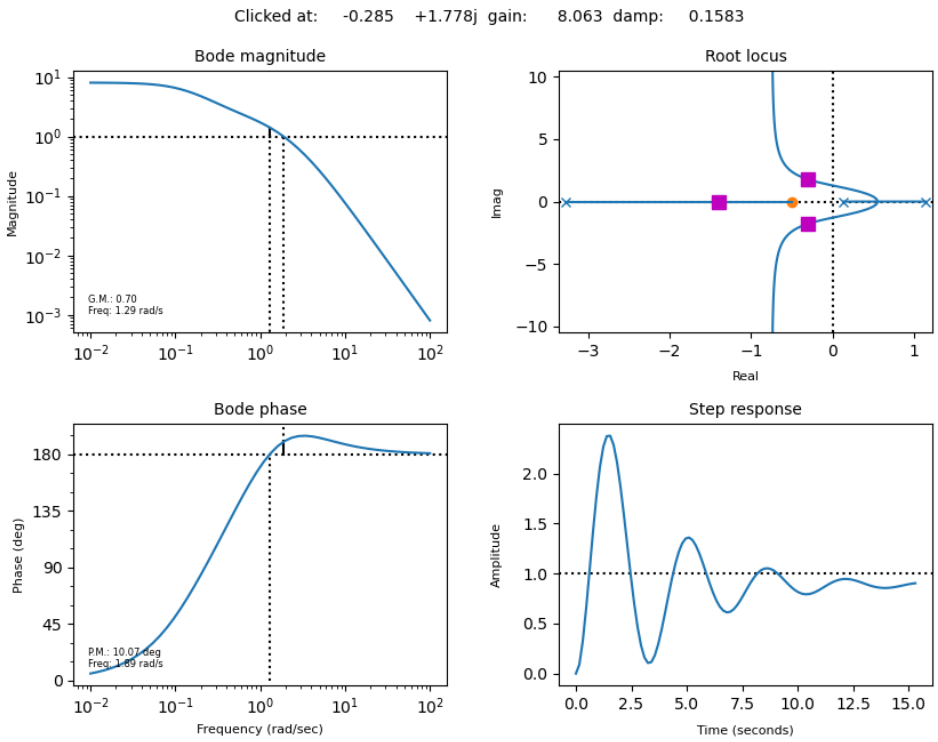
Έστω σύστημα με συνάρτηση μεταφοράς $G(s) = \dfrac{2s+1}{2s^3+4s^2-8s+1}$

Εισάγουμε τη συνάρτηση μεταφοράς στην Python

```
import control as co
import matplotlib.pyplot as plt
import numpy as np

s = co.tf('s')
g = (2*s+1)/(2*s**3+4*s**2-8*s+1)
```

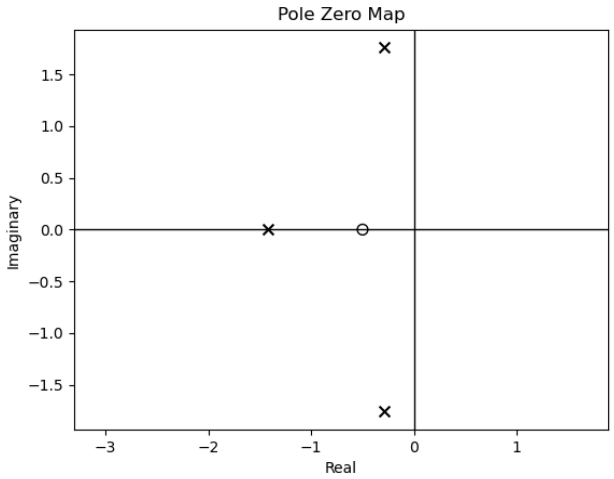
Εισάγουμε τη συνάρτηση μεταφοράς στο sisotool

```
plt.figure(1)
siso = co.sisotool(g)
```



Εικόνα 6.19 Διαγράμματα sisotool του συστήματος

Το σύστημα είναι σταθεροποιήσιμο για K > 5.5



Εικόνα 6.20 Διάγραμμα πόλων-μηδενικών του συστήματος για K=8

Θέλουμε το κλειστό σύστημα να είναι ευσταθές με χρόνο αποκατάστασης $Ts < 7$ και υπερύψωση $OS < 20\%$. Για αυτό θα σχεδιάσουμε τις ευθείες των περιορισμών στο διάγραμμα των γεωμετρικών τόπων (root locus).

Γνωρίσουμε ότι $Ts \approx \frac{-4}{\zeta\omega_n} = \frac{-4}{\sigma}$ και $OS = e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}}$ όπου σ το σημείο τομής με τον πραγματικό άξονα και $\theta = \cos^{-1}(\zeta)$ η γωνία με τον πραγματικό άξονα.

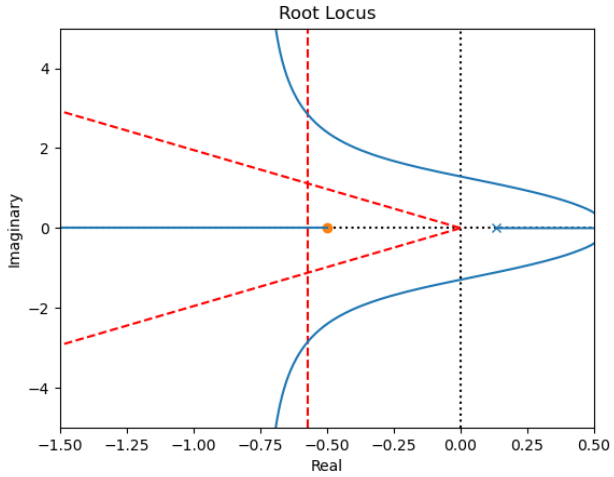
Ο γεωμετρικός τόπος του περιορισμού για τον χρόνο αποκατάστασης είναι ευθεία κατακόρυφη που διέρχεται από το σημείο σ ενώ ο γεωμετρικός τόπος του περιορισμού για την υπερύψωση είναι οι ευθείες με γωνίες θ και $-\theta$.

Ορίζουμε τις συναρτήσεις που σχεδιάζουν τις ευθείες για δοσμένη υπερύψωση.

```
def os_cosines(x):
    log2 = (np.log(x)) ** 2
    pi2 = (np.pi) ** 2
    numrator = np.log(x)
    denominator = np.sqrt(log2+pi2)
    return numrator / denominator

def plot_line(slope):
    axes = plt.gca()
    x_vals = np.array(axes.get_xlim())
    y_vals = slope*x_vals
    plt.plot(x_vals,y_vals, 'r--')
```

Ο γεωμετρικός τόπος του αρχικού κλειστού συστήματος με τους περιορισμούς

```
# root locus plot
plt.figure(3)
co.root_locus(g, grid=False)

# settling time plot
TS = 7
plt.axvline(-4/TS, color='r', linestyle='--')

# overshoot plots
slope1 = np.tan(np.arccos(os_cosines(0.2)))
plot_line(slope1)
plot_line(-slope1)

plt.xlim([-1.5, 0.5])
plt.ylim([-5,5])
```

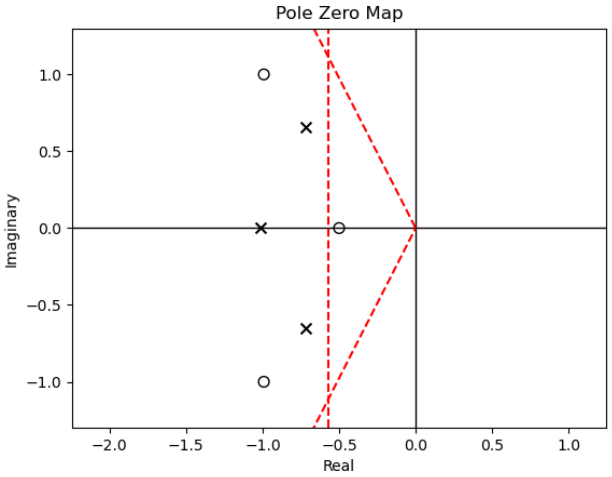


Εικόνα 6.21 Διάγραμμα γεωμετρικού τόπου ριζών του συστήματος με περιορισμούς

Παρατηρούμε ότι το αρχικό κλειστό σύστημα ΔΕΝ είναι σταθεροποιήσιμο κάτω από τους δοσμένους περιορισμούς ($Ts < 7, OS < 20\%$) καθώς δεν υπάρχουν πόλοι που να βρίσκονται στο αριστερό ημιεπίπεδο που ορίζουν οι ευθείες των περιορισμών.

Βάζουμε δύο φανταστικούς πόλους $s_p = -1 \pm j$. Η συνάρτηση ελέγχου θα έχει τη μορφή $C(s) = 21.535(1 + s + 0.71s^2)$

```
k = 21.535
cs = k * (1 + s + (0.71 * s) ** 2)
g_feedback = co.feedback(co.series(g, cs), 1, sign=-1)

plt.figure(4)
co.pzmap(g_feedback)

TS = 7
slope1 = np.tan(np.arccos(os_cosines(0.2)))
plt.axvline(-4 / TS, color='r', linestyle='--')
plot_line(slope1)
plot_line(-slope1)

plt.xlim([-1.5, 0.5])
plt.ylim([-1.3, 1.3])

plt.show()
```



Εικόνα 6.22 Διάγραμμα πόλων-μηδενικών με περιορισμούς

7. Είδη ελεγκτών

Μέχρι τώρα μελετήσαμε τον έλεγχο σταθεροποιήσιμων συστημάτων με συντελεστή κέρδους και με προσθήκη πολυωνυμικών ελεγκτών. Τρία, επιπλέον είδη πολυωνυμικών ελεγκτών είναι οι ελεγκτές προήγησης (lead compensators), οι ελεγκτές υστέρησης (lag compensators) και οι PID ελεγκτές.[12]

Ελεγκτής προήγησης (Lead compensator)

$$C(s)=\frac{s-z_0}{s-p_0}K\text{, όπου}\left|z_0\right|<\left|p_0\right|,\ z_0,\ p_0\in R^-,\ K\in R$$

Ο ελεγκτής προήγησης προκύπτει αν προσθέσουμε ένα πραγματικό μηδενικό στο αριστερό μιγαδικό επίπεδο του γ.τ.ρ. με τον πόλο να βρίσκεται αριστερά από το μηδενικό.

Ελεγκτής υστέρησης (Lag compensator)

$$C(s)=\frac{s-z_0}{s-p_0}K\text{, όπου }\left|z_0\right|>\left|p_0\right|,\ z_0,\ p_0\in R^-,\ K\in R$$

Ο ελεγκτής υστέρησης προκύπτει αν προσθέσουμε έναν πραγματικό πόλο και ένα πραγματικό μηδενικό στο αριστερό μιγαδικό επίπεδο του γ.τ.ρ. με τον πόλο να βρίσκεται δεξιά από το μηδενικό.

Ελεγκτής PID

$$C(s)=K_p+K_I\frac{1}{s}+K_{D^s}\quad K_D,K_P,K_I\in\square$$

Ο ελεγκτής PID προκύπτει αν προσθέσουμε έναν πραγματικό πόλο στο 0 και δύο μηδενικά στα αριστερό μιγαδικό επίπεδο του γ.τ.ρ.
Αντίστοιχα,

Ελεγκτής PI

$$C(s)=K_p+K_I\frac{1}{s}$$

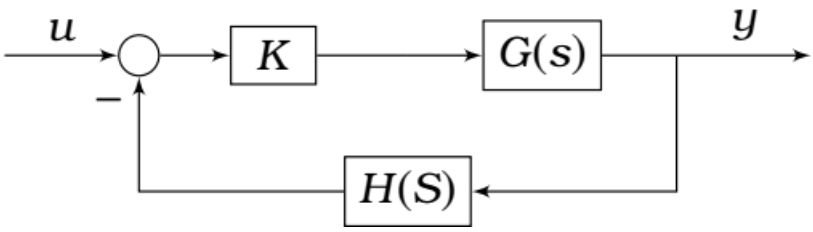
[12] Δρ. Βολογιαννίδης Σ., Χατζηγεωργίου Κ., Γκουτζιαμάνης Π., *Συστήματα Αυτομάτου Ελέγχου - Σημειώσεις Εργαστηρίου*, ΤΕΙ Σερρών, (εργαστηριο 8, σελ 1)

Ελεγκτής PD

$C(s) = K_p + K_D s$

Παράδειγμα 1

Έστω το σύστημα $G(s)\frac{s+1}{(s-1)(s-4)}$, $H(s)=1$ και η σύνδεση του σχήματος.

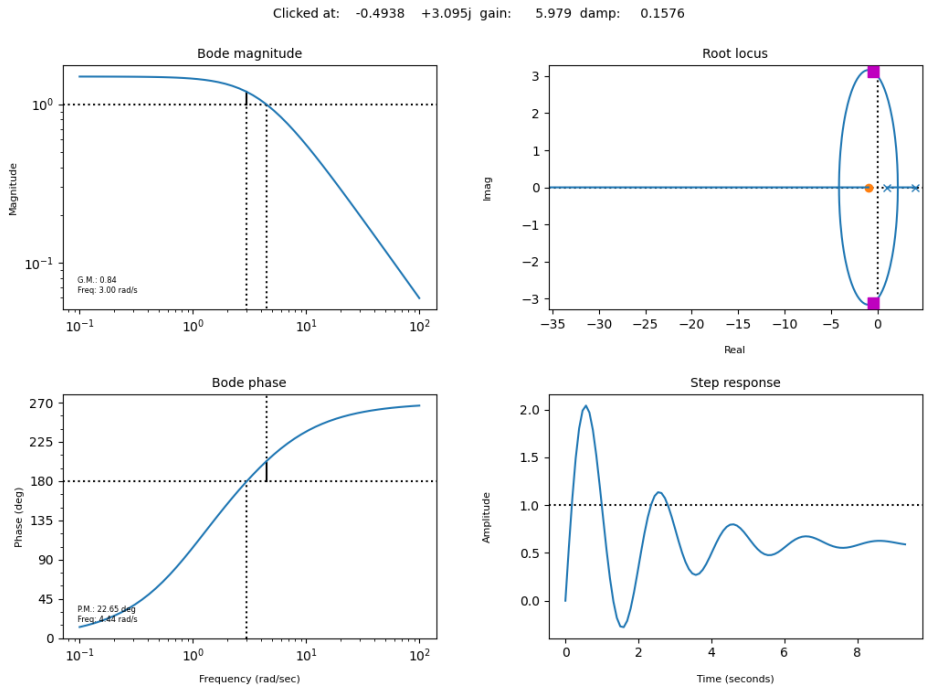


Εισάγουμε τη συνάρτηση μεταφοράς του συστήματος

```
import matplotlib.pyplot as plt
import control as co
import numpy as np


s = co.tf('s')
g = (s+1)/((s-1)*(s-4))
h1 = 1
```

από το sisotool, για $K > 5$ οι πόλοι έρχονται στο μιγαδικό αρνητικό μέρος

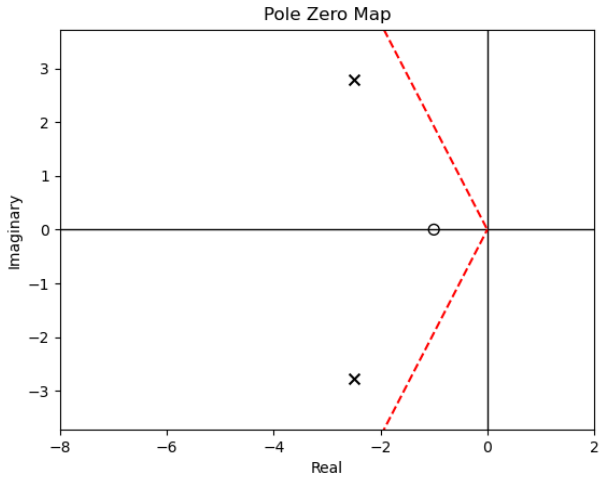
```
plt.figure(1)
co.sisotool(g)

plt.show()
```



Στη συνέχεια επιλέγουμε συντελεστή κέρδους $K = 10$ ώστε να πετύχουμε υπερύψωση $OS < 30\%$

```
def slope_Os(x):
    log2 = (np.log(x)) ** 2
    pi2 = (np.pi) ** 2
    enumrator = log2 + np.sqrt((1 + 4 * pi2) * log2)
    denominator = 2 * pi2
    return enumrator / denominator


def plot_Os(x):
    slope = np.tan(np.arccos(slope_Os(x)))  # overshoot 30%
    axes = plt.gca()
    x1_vals = np.array(axes.get_xlim())
    x_vals = np.array((x1_vals[0], 0))
    y1_vals = slope * x_vals
    y2_vals = -slope * x_vals
    plt.plot(x_vals, y1_vals, 'r--', x)
    plt.plot(x_vals, y2_vals, 'r--')


s = co.tf('s')
g = (s + 1) / ((s - 1) * (s - 4))
h1 = 1
k = 10

gf = co.feedback(k * g, h1, -1)
co.pzmap(gf, grid=False)
plot_Os(0.3)
plt.xlim([-8, 2])
```

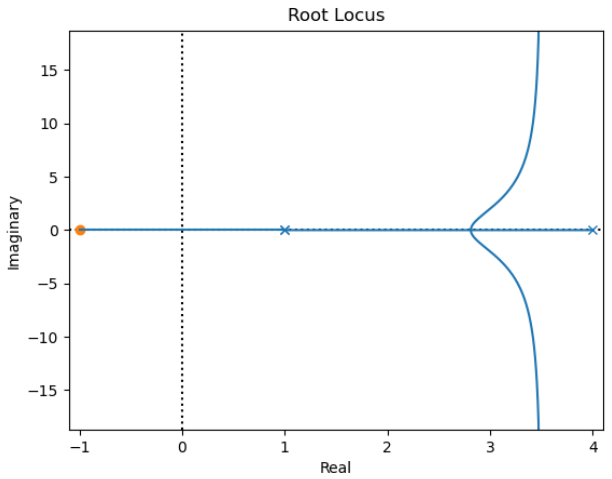
```
plt.ylim([-3, 3])

plt.show()
```



Εικόνα 7.1 Διάγραμμα πόλων-μηδενικών με K=10 και OS<30%

Έστω $H_2(s) = \frac{1}{s-1}$. Ελέγχουμε αν υπάρχει κατάλληλος ελεγκτής που να κάνει το σύστημα ευσταθές.

```
s = co.tf('s')

g = (s + 1) / ((s - 1) * (s - 4))
h2 = 1 / (s - 1)
gf = co.feedback(g, h2, -1)

plt.figure(1)
co.root_locus(g * h2, grid=False)

plt.show()
```



Εικόνα 7.2 Διάγραμμα γ.τ.ρ. για $H_2(s)$

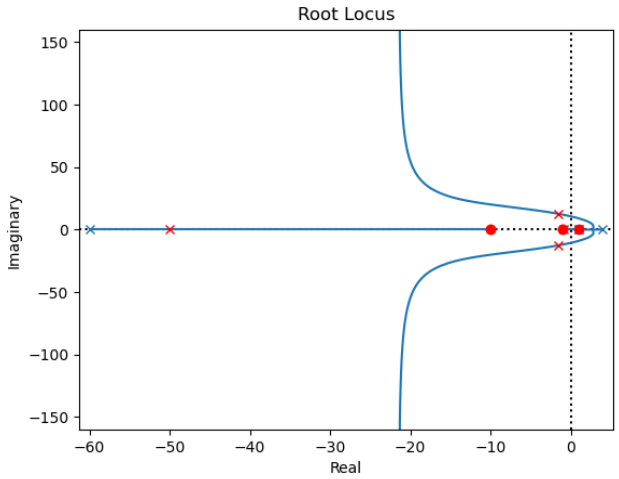
Από τον γ.τ.ρ. παρατηρούμε ότι οι πόλοι είναι πραγματικοί θετικοί αριθμοί, και δεν μπορεί το σύστημα να γίνει ευσταθές για κανένα K καθώς οι κλάδοι απειρίζονται στο δεξί ημιεπίπεδο των μιγαδικών αριθμών.

Στη συνέχεια γίνεται μελέτη για να ελέγξουμε αν υπάρχει ελεγκτής προήγησης που να κάνει το σύστημα ευσταθές.

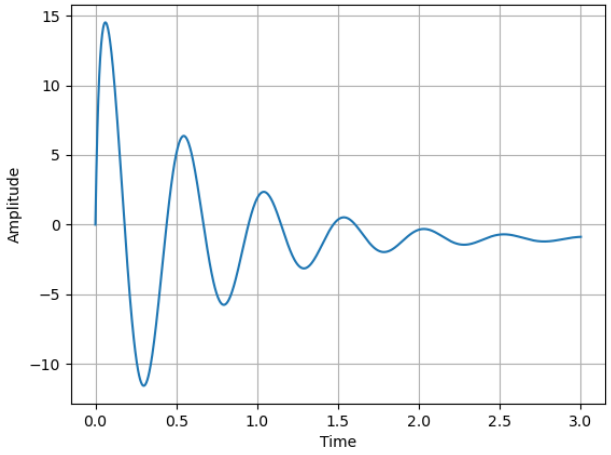
Για αρχή κάνουμε μια συνάρτηση κατασκευής ελεγκτή προήγησης με είσοδο τις θέσεις ενός πραγματικού μηδενικού και ενός πραγματικού πόλου, με τα οποία κατασκευάζει τα πολυώνυμα μετά την προσθήκη των ο,χ

```
def lead_comp(spole, szero):
    cz = 1 + (s / (-szero))
    cp = 1 / (1 + (s / (-spole)))
    return cz * cp


k = 120
gf = co.feedback(k * lead_comp(-60, -10) * g, h2, -1)  #
υπολογίζω τους πόλου και τα μηδενικά του κλειστού συστήματος με K
= 120 και feedback=h2
poles = co.pole(gf)
zeros = co.zero(gf)
print(f"poles: {poles}")
print(f"zeros: {zeros}")

plt.figure(1)
co.root_locus(lead_comp(-60, -10) * g * h2, grid=False)  #
σχεδιάζω την γραφική παράστασή μου με lead_compensator
plt.plot(poles.real, poles.imag, 'rx')
plt.plot(zeros.real, zeros.imag, 'ro')

plt.figure(2)
t = np.linspace(0, 3, 1000)
t1, yout = co.step_response(gf, t)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.grid()
plt.plot(t1, yout)
plt.show()
```

```
poles: [-49.9458761  +0.j          -1.60166149+12.69734304j
  -1.60166149-12.69734304j  -0.85080092 +0.j        ]
zeros: [-10.   1.  -1.]
```



Εικόνα 7.3 Διάγραμμα γ.τ.ρ. με K=120 και ελεγκτή προήγησης



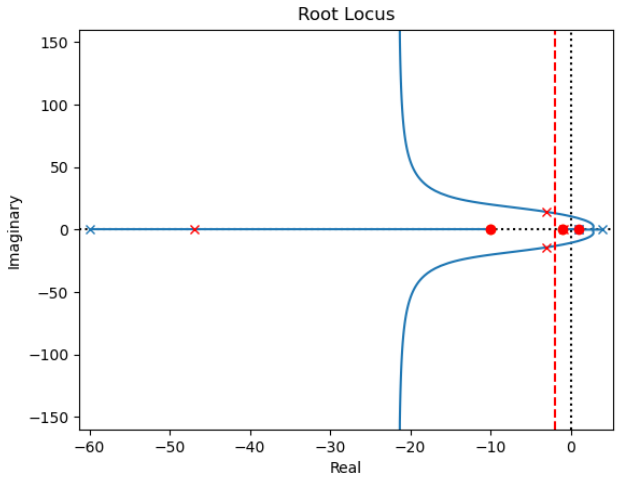
Εικόνα 7.4 Διάγραμμα βηματικής απόκρισης του κλειστού συστήματος

Αν ορίσουμε συντελεστή κέρδους $K = 150$ μπορούμε να πετύχουμε ευστάθεια στο σύστημα με χρόνο αποκατάστασης $Ts < 2$.

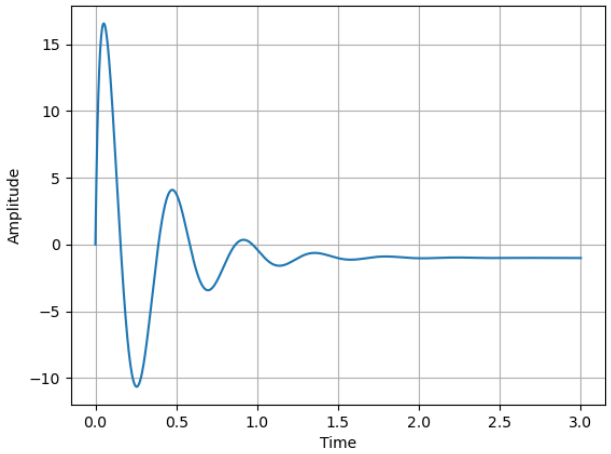
```
def step_info(t, yout):
    print(f"Max Amp: {max(yout)}")
    print("OS: %f%s" % ((yout.max() / yout[-1] - 1) * 100, '%'))
    print("Tr: %fs" % (t[next(i for i in range(0, len(yout) - 1)
if yout[i] > yout[-1] * .90)] - t[0]))
    print("Ts: %fs" % (t[next(len(yout) - i for i in range(2,
len(yout) - 1) if abs(yout[-i] / yout[-1]) > 1.02)] - t[0]))

def plot_ts(TS):
    plt.axvline(-4 / TS, color='r', linestyle='--')
```

```
k = 150
gf = co.feedback(k * lead_comp(-60, -10) * g, h2, -1)  #
υπολογίζω τους πόλου και τα μηδενικά του κλειστού συστήματος με K
= 150 και feedback=h2
poles = co.pole(gf)
zeros = co.zero(gf)

plt.figure(1)
co.root_locus(lead_comp(-60, -10) * g * h2, grid=False)  #
σχεδιάζω την γραφική παράστασή μου με lead_compensator
plot_ts(2)
plt.plot(poles.real, poles.imag, 'rx')
plt.plot(zeros.real, zeros.imag, 'ro')

plt.figure(2)
t = np.linspace(0, 3, 1000)
t1, yout = co.step_response(gf, t)

step_info(t1, yout)
plt.plot(t1, yout)

plt.show()
```

```
poles: [-46.95680156 +0.j          -3.08340943+14.26050633j
  -3.08340943-14.26050633j  -0.87637958 +0.j        ]
zeros: [-10.   1.  -1.]
Max Amp: 16.547138212868937
OS: -1735.480719%
Tr: 0.000000s
Ts: 1.666667s
```



Εικόνα 7.5 Διάγραμμα γ.τ.ρ. με K=150, ελεγκτή προήγησης και περιορισμό Ts<2

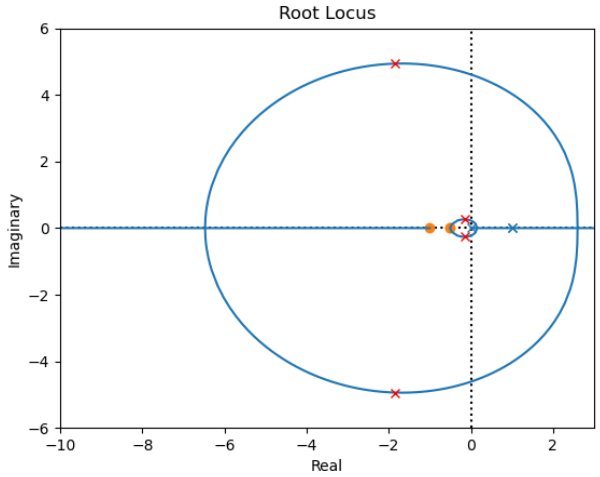


Εικόνα 7.6 Διάγραμμα βηματικής απόκρισης του κλειστού συστήματος με περιορισμό Ts<2

Σε περίπτωση που έχουμε $H_2(s)=\frac{1}{s-1}$ και θέλουμε να δούμε αν υπάρχει PID ελεγκτής που κάνει το σύστημα ευσταθές κάνουμε

```
def pid_comp(szero1, szero2):
    cz1 = 1 + (s / (-szero1))
    cz2 = 1 + (s / (-szero2))
    return cz1 * cz2 * (1 / s)

k = 2.5
gf = co.feedback(k * pid_comp(-0.5, -0.5) * g, h2, -1)
# υπολογίζω τους πόλους και τα μηδενικά του κλειστού συστήματος
με K = 2.5 και feedback=h2
poles = co.pole(gf)
zeros = co.zero(gf)
print(f"poles: {poles}")
print(f"zeros: {zeros}")

plt.figure(1)
# σχεδιάζω τους πόλους με PID compensator και K=2.5
co.root_locus(pid_comp(-0.5, -0.5) * g * h2, grid=False)
plt.xlim(-10, 3)
plt.ylim(-6, 6)
plt.plot(poles.real, poles.imag, 'rx')

plt.show()
```

```
poles:    [-1.85322931+4.93838897j    -1.85322931-4.93838897j    -
0.14677069+0.26137109j  -0.14677069-0.26137109j]
zeros:    [    1.    +0.00000000e+00j    -1.    +0.00000000e+00j    -
0.5+1.24720983e-08j  -0.5-1.24720983e-08j]
```



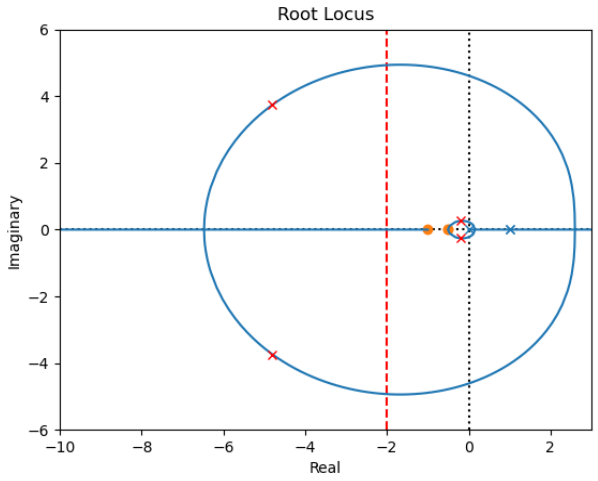
Εικόνα 7.7 Διάγραμμα γ.τ.ρ. με K=2.5 και ελεγκτή PID

Για $K = 4$ παρατηρούμε ότι το σύστημα είναι ευσταθές με $Ts < 2$ (δύο πόλοι τουλάχιστον βρίσκονται αριστερά από την ευθεία του περιορισμού)

```
k = 4
gf = co.feedback(k*pid_comp(-0.5, -0.5)*g, h2, -1) # υπολογίζω
τους πόλου και τα μηδενικά του κλειστού συστήματος με K = 4 και
feedback=h2
poles = co.pole(gf)
zeros = co.zero(gf)
print(f"poles: {poles}")
print(f"zeros: {zeros}")

plt.figure(1)
co.root_locus(pid_comp(-0.5, -0.5)*g*h2, grid=False) # σχεδιάζω
την γραφική παράστασή μου με pid_compensator
plot_ts(2)
plt.xlim(-10,3)
plt.ylim(-6,6)
plt.plot(poles.real, poles.imag, 'rx')

t = np.linspace(0,3,1000)
t1, yout = co.step_response(gf, t)

step_info(t1, yout)


plt.show()
```

```
poles:    [-4.79786695+3.74074482j    -4.79786695-3.74074482j    -
0.20213305+0.25925518j  -0.20213305-0.25925518j]
zeros:    [    1.    +0.00000000e+00j    -1.    +0.00000000e+00j    -
0.5+1.24720983e-08j  -0.5-1.24720983e-08j]
```

```
Max Amp: 16.0
OS: -1691.786281%
Tr: 0.000000s
Ts: 0.795796s
```

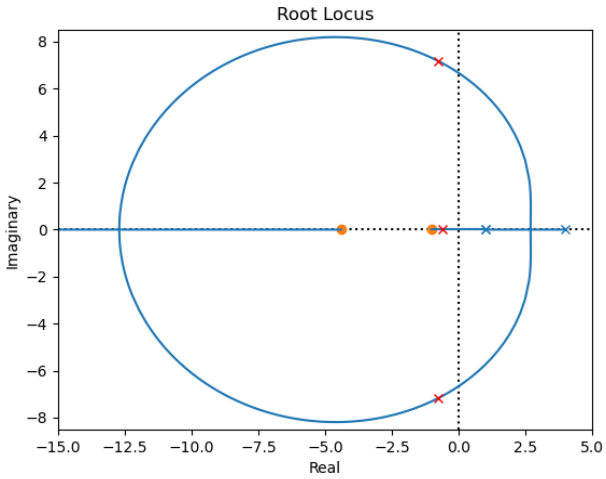


Εικόνα 7.8 Διάγραμμα γ.τ.ρ. με K=4, ελεγκτή PID και περιορισμό Ts<2

Τέλος, μελετάμε αν για $H_2(s)=\frac{1}{s-1}$ υπάρχει PD ελεγκτής που κάνει το σύστημα ευσταθές.

```
def pd_comp(szero):
    cz1 = 1 + (s / (-szero))
    return cz1


k = 36
gf = co.feedback(k * pd_comp(-4.4) * g, h2, -1)  # υπολογίζω τους πόλου και τα μηδενικά του κλειστού συστήματος με K = 36 και feedback=h2
poles = co.pole(gf)
zeros = co.zero(gf)
print(f"poles: {poles}")
print(f"zeros: {zeros}")

plt.figure(1)
co.root_locus(pd_comp(-4.4) * g * h2, grid=False)  # σχεδιάζω την γραφική παράστασή μου με pd_compensator
plt.xlim(-15, 5)

plt.plot(poles.real, poles.imag, 'rx')

plt.show()
poles: [-0.78451504+7.18365307j -0.78451504-7.18365307j
```

```
-0.6127881 +0.j]
zeros: [-4.4  1.  -1. ]
```

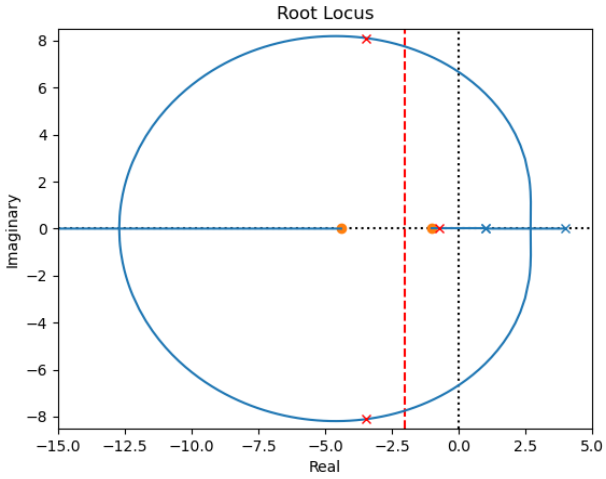


Εικόνα 7.9 Διάγραμμα γ.τ.ρ. με K=36 και ελεγκτή PD

Για $K = 60$ παρατηρούμε ότι το σύστημα είναι ευσταθές με $Ts < 2$ (δύο πόλοι τουλάχιστον βρίσκονται αριστερά από την ευθεία του περιορισμού)

```
k = 60
gf = co.feedback(k*pd_comp(-4.4)*g, h2, -1) # υπολογίζω τους
πόλου και τα μηδενικά του κλειστού συστήματος με K = 60 και
feedback=h2
poles = co.pole(gf)
zeros = co.zero(gf)
print(f"poles: {poles}")
print(f"zeros: {zeros}")

plt.figure(1)
co.root_locus(pd_comp(-4.4)*g*h2, grid=False) # σχεδιάζω την
γραφική παράστασή μου με pd_compensator
plot_ts(2)
plt.xlim(-15,5)

plt.plot(poles.real, poles.imag, 'rx')

t = np.linspace(0,3,1000)
t1, yout = co.step_response(gf, t)
step_info(t1, yout)

plt.show()
```

```
poles: [-3.45758568+8.10520183j -3.45758568-8.10520183j
-0.72119228+0.j]
zeros: [-4.4  1.  -1. ]
Max Amp: 13.636363636363635
OS: -1438.572729%
Tr: 0.000000s
Ts: 1.246246s
```

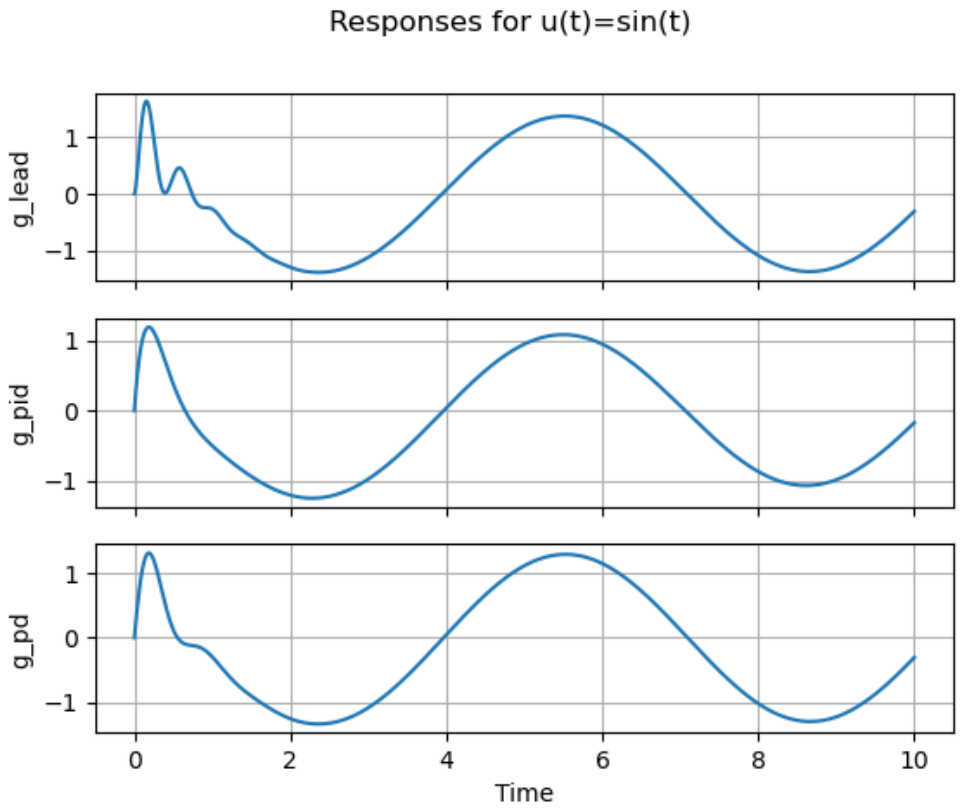


Εικόνα 7.10 Διάγραμμα γ.τ.ρ. με K=60, ελεγκτή PD και περιορισμό Ts<2

Οι αποκρίσεις των συστημάτων με $u(t) = \sin(t)$ και με ελεγκτές προήγησης, PID και PD αντίστοιχα.

```
import control as co
import matplotlib.pyplot as plt
import numpy as np
from control import matlab


def lead_comp(spole, szero):
    cz = 1 + (s / (-szero))
    cp = 1 / (1 + (s / (-spole)))
    return cz * cp


def pid_comp(szero1, szero2):
    cz1 = 1 + (s / (-szero1))
    cz2 = 1 + (s / (-szero2))
    return cz1 * cz2 * (1 / s)


def pd_comp(szero):
    cz1 = 1 + (s / (-szero))
    return cz1
```

```
s = co.tf('s')
t = np.linspace(0, 10, 1000)
g = (s + 1) / ((s - 1) * (s - 4))
h2 = 1 / (s - 1)
ut = np.sin(t)

k_lead = 150
g_lead = co.feedback(k_lead * lead_comp(-60, -10) * g, h2, -1)
ylead, T1, u1out = matlab.lsim(g_lead, ut, t)

k_pid = 4
g_pid = co.feedback(k_pid * pid_comp(-0.5, -0.5) * g, h2, -1)
ypid, T2, u2out = matlab.lsim(g_pid, ut, t)

k_pd = 60
g_pd = co.feedback(k_pd * pd_comp(-4.4) * g, h2, -1)
ypd, T3, u3out = matlab.lsim(g_pd, ut, t)

fig, axs = plt.subplots(3, 1)
fig.suptitle('Responses for u(t)=sin(t)')
axs[0].plot(T1, ylead)
axs[1].plot(T2, ypid)
axs[2].plot(T3, ypd)

axs[0].set_ylabel('g_lead')
axs[1].set_ylabel('g_pid')
axs[2].set_ylabel('g_pd')
axs[2].set_xlabel('Time')
axs[0].grid()
axs[1].grid()
axs[2].grid()

for axs in fig.get_axes():
    axs.label_outer()
plt.show()
```



Εικόνα 7.11 Διάγραμμα αποκρίσεων των συστημάτων με ελεγκτή προήγησης, PID και PD

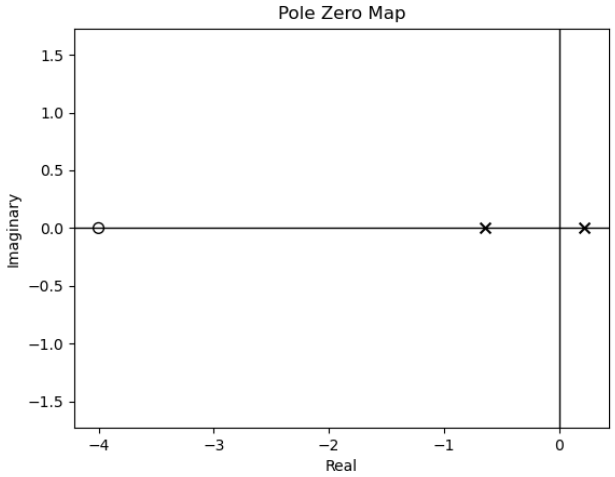
8. Μονάδα συμβατότητας με το MATLAB

Η μονάδα συμβατότητας με το MATLAB (MATLAB compatibility module) περιέχει ένα πλήθος από συναρτήσεις οι οποίες προσομοιώνουν μέρος της λειτουργικότητας του MATLAB. Ο σκοπός αυτών των συναρτήσεων είναι η παροχή μιας απλής διασύνδεσης στη βιβλιοθήκη των συστημάτων ελέγχου της Python σε άτομα που είναι οικεία με την εργαλειοθήκη συστημάτων ελέγχου του MATLAB (MATLAB Control Systems Toolbox).

Παράδειγμα 1

Έστω το σύστημα που περιγράφεται από την συνάρτηση μεταφοράς $G(s) = \frac{s+4}{7s^2+3s-1}$ και θέλουμε να βρούμε τους πόλους και τα μηδενικά του όπως επίσης να κατασκευάσουμε το διάγραμμα πόλων-μηδενικών, με τη χρήση της μονάδας control.matlab.

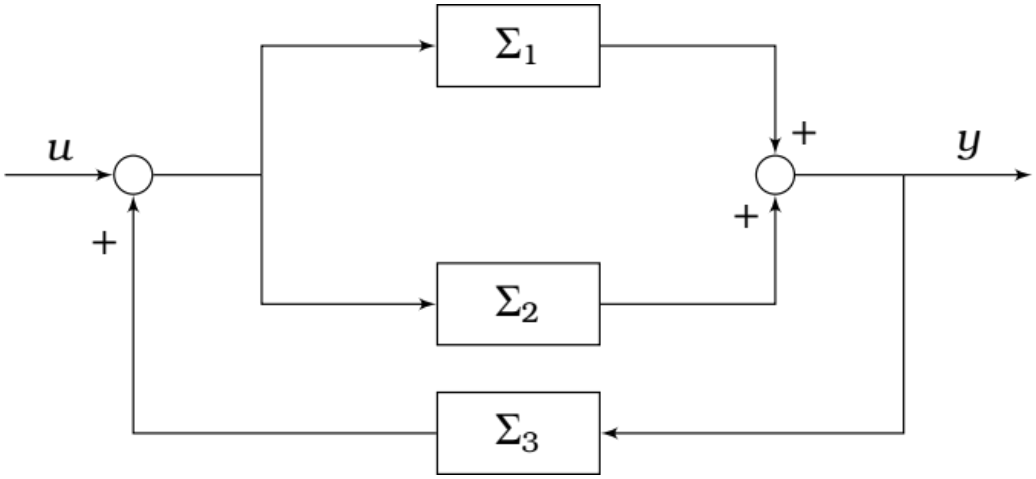
```
import control as co
import control.matlab
import matplotlib.pyplot as plt

s = co.matlab.tf("s")
g = (s + 4) / (7 * s ** 2 + 3 * s - 1)
g_poles = co.matlab.pole(g)
g_zeros = co.matlab.zero(g)

print('poles')
for elem in g_poles:  # print poles as a column vector for
convenience
    print(f'{elem: .2f}')  # Two floating points per print

print('zeros')
for elem in g_zeros:
    print(f'{elem: .2f}')

g_map = co.matlab.pzmap(g, plot=True)
plt.show()
poles
-0.65
 0.22
zeros
-4.00
```



Εικόνα 8.1 Διάγραμμα πόλων-μηδενικών του συστήματος

Παράδειγμα 2

Έστω το παρακάτω σύστημα



$$G_1(s)=\frac{1}{(s+1)(s-1)}\ ,\ G_2(s)=\frac{1}{(s+2)}\ ,\ G_3(s)=\frac{0.5s}{3s^2+5s+1}$$

Και θέλουμε να βρούμε την συνολική συνάρτηση μεταφοράς του συστήματος.

```
import control.matlab
import control as co

s = co.matlab.tf('s')
g1 = 1/((s+1)*(s-1))
g2 = 1/(s+2)
g3 = (0.5*s)/(3*s**2+5*s+1)


g12 = co.matlab.parallel(g1,g2)
```

```
g123 = co.matlab.feedback(g12,g3,-1)
g = co.matlab.series(g123,g3)
gtot =
co.matlab.series(co.matlab.feedback(co.matlab.parallel(g1,g2),g3,
-1),g3)
print(g, gtot)
             1.5 s^5 + 4 s^4 + 4.5 s^3 + 3 s^2 + 0.5 s
-----------------------------------------------------------------
9 s^7 + 48 s^6 + 83.5 s^5 + 28 s^4 - 65.5 s^3 - 67 s^2 - 20.5 s - 2


             1.5 s^5 + 4 s^4 + 4.5 s^3 + 3 s^2 + 0.5 s
-----------------------------------------------------------------
9 s^7 + 48 s^6 + 83.5 s^5 + 28 s^4 - 65.5 s^3 - 67 s^2 - 20.5 s - 2
```

Για να υπολογίσουμε την κρουστική και την βηματική απόκριση με το MATLAB compatibility module χρησιμοποιούμε τις εντολές impulse() και step() και τις συντάσσουμε ως εξής:

yout, T = matlab.impulse(sys, t)

- sys: Η συνάρτηση μεταφοράς του συστήματος
- t: Το διάνυσμα του χρόνου

και επιστρέφει:

- yout: Απόκριση του συστήματος
- T: Τιμές χρόνων της εξόδου

Με αντίστοιχο τρόπο συντάσσεται και η εντολή step().

Παράδειγμα 3

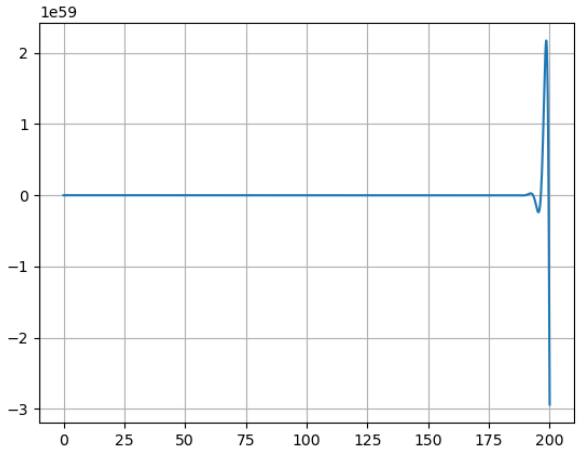
Έστω η συνάρτηση μεταφοράς:

$$G(s) = \frac{(4s^2 - 3)(6s + 8)}{(6s^3 - 2s^2 + 9)(s^2 + 8)}$$

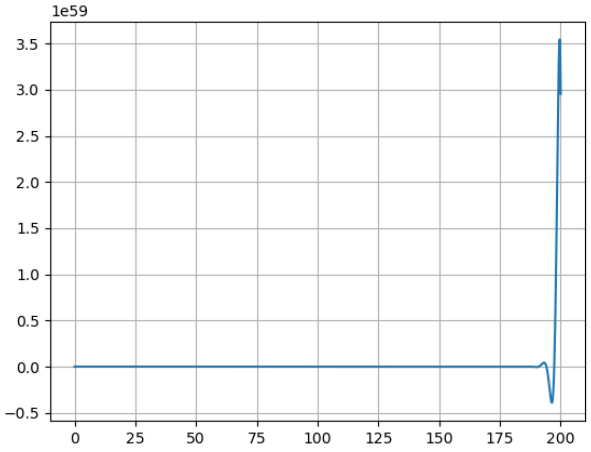
Για να κατασκευάσουμε τις γραφικές παραστάσεις της βηματικής και της κρουστικής απόκρισης με τη μονάδα του MATLAB γράφουμε

```
import control as co
import matplotlib.pyplot as plt
import numpy as np
from control import matlab
```

```
s = co.tf('s')
g1 = ((4 * s ** 2 - 3) * (6 * s + 8)) / ((6 * s ** 3 - 2 * s ** 2
+ 9) * (s ** 2 + 8))
print(g1)
t = np.linspace(0, 200, 1000)

imp, t1  = co.matlab.impulse(g1, t)
stp, t2  = co.matlab.step(g1, t)
plt.grid()
plt.figure(1)
plt.plot(t1,imp)

plt.figure(2)
plt.grid()
plt.plot(t2, stp)
plt.show()
```



Εικόνα 8.2 Διάγραμμα κρουστικής απόκρισης



Εικόνα 8.3 Διάγραμμα βηματικής απόκρισης

Τέλος με το MATLAB compatibility module είναι δυνατή η κατασκευή του γεωμετρικού τόπου ριζών με την εντολή control.matlab.rlocus(sys) όπως επίσης και η χρήση του εργαλείου sisotool με την εντολή control.matlab.sisotool(sys). Ο τρόπος σύνταξης τους και τα αποτελέσματά των εντολών είναι ίδια με τα αποτελέσματα των αντίστοιχων εντολών της βιβλιοθήκης control.

9. Παρατηρήσεις και Συμπεράσματα

Στην εργασία αυτή μελετήσαμε τον τρόπο με τον οποίο μπορούμε να χρησημοποιήσουμε τη γλώσσα προγραμματισμού Python και να αναπτύξουμε σε αυτήν κώδικα για τη δημιουργία και τη μελέτη συστημάτων αυτομάτου ελέγχου. Πρέπει να σημειωθεί ότι για τη σωστή μελέτη των συστημάτων χρειάζεται καλή γνώση της γλώσσας Python όπως επίσης και των επιμέρους βιβλιοθηκών που χρησιμοποιήσαμε.

Κατά τη διάρκεια της εγασίας και χρησιμοποιώντας τα πακέτα που παρέχει η Python χρειάστηκε να αναπτύξουμε δικούς μας κώδικες, που δε συμπεριελάμβανε η βιβλιοθήκη Control, προκειμένου να εμβαθύνουμε στη μελέτη των συστημάτων. Για αυτό το λόγο η μελέτη των ΣΑΕ με τη χρήση της Python αποτελεί έναν ισχυρό εργαλείο για την μελέτη αλλά και για την εκμάθηση τους, που όμως επιδέχεται βελτιώσεων.

Συνεπώς, θα μπορούσε μεσοπρόθεσμα να αναπτυχθεί μια ολοκληρομένη εφαρμογή που να εκμεταλεύεται όλες τις δυνατότητες που παρέχει η Python, βελτιωτικούς κώδικες, μερικούς από τους οποίους αναπτύξαμε στην εργασία και ένα κατάλληλο γραφικό περιβάλλον. Ο σκοπός αυτού του λογισμικού θα είναι η μελέτη των συστημάτων από οποιονδήποτε χρήστη χωρίς την ανάγκη συγγραφής κώδικα. Έτσι, θα μπορούσε να αποτελέσει ένα αρχικό στάδιο μιας εμπορικής εφαρμογής.

Βιβλιογραφία

[1] Δρ. Βολογιαννίδης Σταύρος, Χατζηγεωργίου Κατερίνα, Γκουτζιαμάνης Παύλος, *Συστήματα Αυτομάτου Ελέγχου - Σημειώσεις Εργαστηρίου*, ΤΕΙ Σερρών.

[2] Δρ. Βολογιαννίδης Σταύρος, *Συστήματα Αυτομάτου Ελέγχου Θεωρία και Εφαρμογές, Διδακτικές Σημειώσεις Τμήματος Πληροφορικής και Επικοινωνιών*.

[3] Richard C. Dorf, Robert H. Bishop, 2000. «*Modern Control Systems*» *9th Edition*.

[4] Πετρίδης Β., 2011. «*Συστήματα Αυτομάτου Ελέγχου*», εκδόσεις Ζήτη.

[5] Βουγιατζής Γ., Μελετλίδου Ε. 2015. «*Εισαγωγή στα μη Γραμμικά Δυναμικά Συστήματα*», ΣΕΑΒ.

[6] Till Tantau, 2007 .«TikZ and PGF Manual v.1.18», http://sourceforge.net/projects/pgf.

[7] MIT Open Courseware, «Systems, Modeling, and Control II», < https://ocw.mit.edu/courses/mechanical-engineering/2-004-systems-modeling-and-control-ii-fall-2007/lecture-notes/ > [Πρόσβαση 28/02/2021]

[8] Ψούνης Δ., «*Σειρά μαθημάτων της γλώσσας προγραμματισμού Python*», < https://www.youtube.com/watch?v=OrlntZ2suBM&list=PLLMmbOLFy25Eohpgb_V3GWKdf8sL0Upvt > [Πρόσβαση 28/02/2021]

[9] Python Software Foundation, 2001- 2021. «*Python Documentation*», < https://www.python.org/doc/versions/ > [Πρόσβαση 28/02/2021]

[10] Python Control Systems Library, 2020 «*Documentation*», < https://python-control.readthedocs.io/en/latest/intro.html > [Πρόσβαση 28/02/2021]

[11] Matplotlib, 2021. «*Documentation*», < https://matplotlib.org/stable/contents.html > [Πρόσβαση 28/02/2021]

[12] SciPy Organisation, «*Documentation*», < https://www.scipy.org/index.html > [Πρόσβαση 28/02/2021]

[13] NumPy, «*Documentation*», < https://numpy.org/doc/stable/contents.html > [Πρόσβαση 28/02/2021]

[14] Python, 2021. < https://el.wikipedia.org/wiki/Python > [Πρόσβαση 28/02/2021]

[15] General Python FAQ, 2021. < https://docs.python.org/3/faq/general.html > [Πρόσβαση 28/02/2021]

[16] Επεξεργασία σήματος, 2017. < https://el.wikipedia.org/wiki/Επεξεργασία_σήματος > [Πρόσβαση 28/02/2021]