ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Πολυτεχνείο Κρήτης
Σχολή Μηχανικών Παραγωγής και Διοίκησης
Διπλωματική Εργασία

Τίτλος: Εφαρμογές Τεχνητών Νευρωνικών Δικτύων για την προσέγγιση ευθέων και αντιστρόφων προβλημάτων στη μηχανική μέσω της μεθόδου Physics Informed Neural Networks (PINNs)

Από τον Κέβιν Χότζα

Επιβλέπων: Σταυρουλάκης Γεώργιος, Καθηγητής
1ο μέλος εξεταστικής επιτροπής: Αλευράς Παναγιώτης, Επίκουρος Καθηγητής
2ο μέλος εξεταστικής επιτροπής: Μπακατσάκη Μαρία, ΕΔΙΠ

Χανιά, Μάρτιος 2024

Ευχαριστίες

Από την θέση αυτή, θα ήθελα να εκφράσω ένα μεγάλο ευχαριστώ και την ευγνωμοσύνη μου προς τον καθηγητή μου κ. Γεώργιο Ε. Σταυρουλάκη για την καθοδήγηση που μου προσέφερε και το χρόνο που διέθεσε δίνοντάς μου χρήσιμες συμβουλές και οδηγίες για την ολοκλήρωση της διπλωματικής μου εργασίας.

Στο ίδιο πλαίσιο ευγνωμοσύνης, θα ήθελα να ευχαριστήσω όλους τους καθηγητές του Τμήματος Μηχανικών Παραγωγής & Διοίκησης για τη συμβολή τους στην επιστημονική και τεχνολογική μου συγκρότηση στα χρόνια της φοίτησής μου στο Τμήμα.

Τέλος, οφείλω επίσης ένα μεγάλο ευχαριστώ στην οικογένεια μου και στους φίλους μου για την αμέριστη και ανιδιοτελή υποστήριξή τους καθόλη τη διάρκεια των σπουδών μου.

Περίληψη

Στη παρούσα διπλωματική εργασία εξετάζεται μια σχετικά πρόσφατη και ανερχόμενη τεχνική επίλυσης συνήθων και μερικών διαφορικών εξισώσεων αξιοποιώντας Τεχνητά Νευρωνικά Δίκτυα ΤΝΔ (Artificial Neural Networks ANN). Η υποκείμενη τεχνική αποκαλείται Physics Informed Neural Networks (PINNs), και ήταν απόρροια της σύνθεσης ποικίλων επιστημονικών κλάδων όπως η νευρολογία, η πληροφορική, η επιστήμη των υπολογιστών, τα εφαρμοσμένα μαθηματικά, η εφαρμοσμένη φυσική και η υπολογιστική μηχανική. Συνολικά, αποτελεί μια αριθμητική προσέγγιση για την επίλυση γραμμικών και μη γραμμικών προβλημάτων που συναντώνται σε ποικίλα επιστημονικά πεδία, όπως η μετάδοση θερμότητας, η ρευστομηχανική και η κλασσική μηχανική.

Η εφαρμογή αριθμητικών τεχνικών σε προβλήματα μεγάλης κλίμακας, που απαιτούν υψηλή ακρίβεια, απαιτεί την συνεχή βελτίωση της επιστήμης των υπολογιστών, τόσο σε επίπεδο hardware όσο και software. Αυτό αληθεύει και για την μελετηθείσα προσέγγιση, η οποία είχε απαρχές, τουλάχιστον σε θεωρητικό επίπεδο, πριν από περίπου δύο δεκαετίες, αλλά η επιτυχής υλοποίηση της με αξιοσημείωτα αποτελέσματα σε σύγχρονους χρονικούς ορίζοντες έλαβε χώρα μόλις πρόσφατα.

Η παρούσα εργασία έχει ως κύριο σκοπό την ανάπτυξη αλγορίθμων σε Python, με απώτερη πρόθεση την εφαρμογή της τεχνικής PINNs σε προβλήματα δυναμικής μηχανικής. Συγκεκριμένα, εξετάστηκαν προβλήματα όπως ο αρμονικός ταλαντωτής με απόσβεση και η κίνηση 2 μαζών συνδεδεμένων με 3 ελατήρια. Καθώς επίσης έγινε προσπάθεια ανάπτυξης ενός αλγορίθμου για ένα αντίστροφο μοντέλο PINNs το οποίο προβλέπει τη τιμή μιας άγνωστης παραμέτρου σε ένα σύστημα αρμονικού ταλαντωτή, αυτή του συντελεστή απόσβεσης.

Επιπλέον, παρουσιάζεται αναλυτικά η αρχή λειτουργίας των τεχνητών νευρωνικών δικτύων, τόσο από θεωρητικής όσο και μαθηματικής πλευράς. Τέλος, αναλύονται τμήματα του κώδικα που χρησιμοποιήθηκαν για την επίλυση των προβλημάτων της εργασίας.

Πίνακας περιεχομένων

Κεφάλαιο 1- Εισαγωγή

Τα τελευταία χρόνια, οι αξιοσημείωτες εξελίξεις στις τεχνικές μηχανικής μάθησης έχουν φέρει επανάσταση σε διάφορους τομείς, συμπεριλαμβανομένης της μηχανικής όρασης, της επεξεργασίας φυσικής γλώσσας, της ανάλυσης δεδομένων και της Θεωρίας Παιγνίων. Ένας τομέας που έχει κεντρίσει το ενδιαφέρον είναι η εφαρμογή των Νευρωνικών Δικτύων σε επιστημονικούς τομείς, επιτρέποντας στους ερευνητές να αντιμετωπίσουν πολύπλοκα προβλήματα που περιλαμβάνουν περίπλοκα φυσικά φαινόμενα. Μεταξύ αυτών των προσεγγίσεων, τα Νευρωνικά Δίκτυα με Πληροφόρηση Φυσικής (PINNs) έχουν αναδειχθεί ως ένα ισχυρό πλαίσιο που ενσωματώνει τις αρχές της φυσικής στα νευρωνικά δίκτυα, επιτρέποντας βελτιωμένη μοντελοποίηση, πρόβλεψη και ανάλυση φυσικών συστημάτων.

Η παραδοσιακή προσέγγιση για την κατανόηση και την προσομοίωση φυσικών συστημάτων βασίζεται σε μεγάλο βαθμό σε μαθηματικά μοντέλα και αριθμητικές μεθόδους. Αν και αυτές οι μέθοδοι έχουν αποδειχθεί αποτελεσματικές, συχνά υποφέρουν από περιορισμούς όπως η ανάγκη για ρητές εξισώσεις, το υψηλό υπολογιστικό κόστος και η υπόθεση της απλοποίησης των υποθέσεων. Αντίθετα, οι μέθοδοι μηχανικής μάθησης, ειδικά τα Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ), διαπρέπουν στην καταγραφή σύνθετων μοτίβων και σχέσεων σε μεγάλα σύνολα δεδομένων χωρίς να απαιτείται ρητή γνώση των υποκείμενων εξισώσεων. Αυτό έχει προκαλέσει σημαντικό ενδιαφέρον για το συνδυασμό των δυνατοτήτων της μοντελοποίησης με βάση τη φυσική και της μηχανικής μάθησης για την ανάπτυξη ενός νέου παραδείγματος για επιστημονική έρευνα.

Τα Νευρωνικά Δίκτυα με Πληροφόρηση Φυσικής αξιοποιούν τις εγγενείς δυνατότητες των νευρωνικών δικτύων ενώ ενσωματώνουν τους θεμελιώδεις νόμους της φυσικής ως προηγούμενη γνώση. Ενσωματώνοντας φυσικές αρχές, όπως νόμους διατήρησης, συμμετρίες και οριακές συνθήκες, στην αρχιτεκτονική του δικτύου, τα PINNs επιτρέπουν την εκπαίδευση μοντέλων που όχι μόνο μαθαίνουν από δεδομένα αλλά σέβονται και τους νόμους που διέπουν το υποκείμενο φυσικό σύστημα. Αυτή η ενοποίηση δίνει τη δυνατότητα στους ερευνητές να χειρίζονται πολύπλοκα, υψηλών διαστάσεων προβλήματα με περιορισμένα δεδομένα, αβέβαιες ή θορυβώδεις μετρήσεις και ελλιπείς πληροφορίες.

Η συγχώνευση της φυσικής και των νευρωνικών δικτύων έχει τεράστιες δυνατότητες σε διάφορους επιστημονικούς κλάδους. Από τη δυναμική των ρευστών και τη μηχανική των στερεών μέχρι την κβαντική φυσική και την κοσμολογία, τα PINNs έχουν αποδείξει την αποτελεσματικότητά τους σε μία ευρεία γκάμα εφαρμογών. Έχουν χρησιμοποιηθεί για την επίλυση προβλημάτων που περιλαμβάνουν προσομοιώσεις ροής ρευστού, αντίστροφα προβλήματα, σχεδιασμό υλικού, βελτιστοποίηση και εκτίμηση παραμέτρων, μεταξύ άλλων. Η ικανότητα των PINNs να καταγράφουν μη γραμμική δυναμική, να ανακαλύπτουν κρυφά μοτίβα και να γενικεύουν από περιορισμένα δεδομένα προσφέρει μια πολλά υποσχόμενη οδό για σημαντικές εξελίξεις στην επιστημονική ανακάλυψη, τον μηχανικό σχεδιασμό και τις διαδικασίες λήψης αποφάσεων.

Σε αυτή τη διπλωματική, στοχεύουμε να εισαχθούμε στον κόσμο των Νευρωνικών Δικτύων με Πληροφόρηση Φυσικής, διερευνώντας τις βασικές αρχές, τις μεθοδολογίες και τις εφαρμογές τους σε διαφορετικούς επιστημονικούς τομείς. Θα διερευνήσουμε την ενσωμάτωση περιορισμών που βασίζονται στη φυσική σε νευρωνικά δίκτυα, τους αλγόριθμους εκπαίδευσης που χρησιμοποιούνται και την αξιολόγηση των μοντέλων που προκύπτουν.

Με την αποσαφήνιση των βασικών εννοιών και μεθοδολογιών που περιβάλλουν τα νευρωνικά δίκτυα με πληροφόρηση φυσικής, αυτή η διπλωματική στοχεύει να παρέχει ένα εισαγωγικό κεφάλαιο σε αυτό το συναρπαστικό πεδίο.

Κεφάλαιο 2- Τεχνητά Νευρωνικά Δίκτυα

2.1 Ιστορική Αναδρομή

Ο τομέας των Τεχνητών Νευρωνικών Δικτύων ανθίζει σε παγκόσμιο επίπεδο στα τέλη του 20ου αιώνα και συγκεκριμένα μετά το 1980 όπου και επιτυγχάνει μεγάλη άνοδο. Η δυνατότητα θεμελίωσης του τομέα αυτού δόθηκε στους επιστήμονες χάρη στη ραγδαία καλυτέρευση του hardware των ηλεκτρονικών υπολογιστών καθώς και από τον εμπλουτισμό των αλγορίθμων εκπαίδευσης.

Το 1943 ήταν η χρονιά όπου έκανε την εμφάνιση του το πρώτο μοντέλο Νευρωνικού Δικτύου από τους McCullogh και Pitts [1], έναν νευροφυσιολόγο και έναν 18χρονο τότε πρωτοετή φοιτητή μαθηματικών. Οι νευρώνες αποτελούσαν τη βασική μονάδα του δικτύου. Ο τρόπος δράσης των νευρώνων με τις συνδέσεις τους παρομοιάζεται με τη λειτουργία του ηλεκτρισμού. Οι δύο ερευνητές προτείνουν η δομή ενός νευρωνικού δικτύου να είναι ένα μεγάλο σύνολο νευρώνων. Παράλληλα τεκμηριώνουν τη διαδικασία λειτουργίας των νευρώνων και των διασυνδέσεών τους. Οι McCulloch και Pitts [2], μέσω της συνεχούς ενασχόλησης τους με το τομέα κατορθώνουν το 1947 να φτάσουν στην αναγνώριση σχημάτων μέσω ενός νέου πιο προηγμένου προτύπου.

Στο καινούργιο πρότυπο, ο νευρώνας έχει δυαδική φύση. Ο κάθε νευρώνας δύναται να έχει πολλαπλές εισόδους, δίνοντας κατ'αποκλειστικό τρόπο μία μοναδική έξοδο, τα δεδομένα της οποίας απαγορεύεται να συνενωθούν με την έξοδο άλλου νευρώνα, αλλά οδηγούνται σε είσοδο διαφορετικού νευρώνα. Οι απολήξεις των νευρώνων, συμβαδίζουν με την κατάσταση τους και διαχωρίζονται σε διεγερτικές ή ανασταλτικές. Δηλαδή, ένας νευρώνας μπορεί είτε να πυροδοτεί, είτε να βρίσκεται σε καταστολή. Κατά τη φάση της διέγερσης ένας νευρώνας, στέλνει έναν παλμό με πληροφορίες, οι οποίες ελέγχονται μέσα στον νευρώνα, από τις πύλες. Οι πύλες παρομοίως με την έξοδο των νευρώνων μπορούν να είναι είτε διεγερτικές είτε ανασταλτικές. Στη δεύτερη εργασία τους σχετικά με τα Νευρωνικά Δίκτυα, οι δύο ερευνητές περιγράφουν ότι οι λειτουργίες αυτές λαμβάνουν χώρα σε διακριτό χρόνο, και επομένως, θεωρητικά, όλοι οι νευρώνες αποκρίνονται ταυτόχρονα. Άρα, το σύστημα έχει συγχρονισμένη δράση.

Τα δίκτυα McCulloch-Pitts προσδιορίζουν και επιχειρούν να αναλύσουν για πρώτη φορά, τη λειτουργικότητα του μηχανισμού μνήμης. Αυτός ο μηχανισμός βασίζεται την ύπαρξη κλειστών διαδρόμων του σήματος μέσα στο δίκτυο. Η ανυπαρξία κλειστής διαδρομής, έχει ως αποτέλεσμα τη παρατεταμένη παραμονή του δικτύου σε κατάσταση ηρεμίας. Πιο λεπτομερώς έχουμε τη δημιουργία ενός μηχανισμού ανάδρασης (feedback), ο οποίος ενώνει την απόληξη ενός κυττάρου με την είσοδο του ίδιου κυττάρου. Έχοντας σαν αποτέλεσμα την επανατροφοδότηση του νευρώνα με τις πληροφορίες που απεστάλησαν στις πύλες του κυττάρου. Αυτή η ροή πληροφορίας στο νευρώνα συγκροτεί το πρώτο μηχανισμό μνήμης.

Μελετώντας τους McColluch και Pitts, ο J. Von Neumann [3] λίγο αργότερα αξιοποίησε τις εργασίες τους για τη δημιουργία υπολογιστικών μηχανών, δίδοντας περαιτέρω ώθηση στην ανάπτυξη του τομέα των ΤΝΔ. Ο D. Hebb [4] επίσης έπαιξε σημαντικό ρόλο στην εξέλιξη των Νευρωνικών Δικτύων, καθώς το 1949, με το βιβλίο του "The Organization of Behavior," παρουσίασε τον κανόνα

μάθησης του Hebb. Σε αυτό το έργο, ο Hebb αποτύπωσε τις συνδέσεις μεταξύ των νευρώνων ως πυρήνα του συστήματος. Με βάση το κανόνα του Hebb, όποτε το δίκτυο χρησιμοποιεί τις νευρωνικές του συνδέσεις, αυτές ισχυροποιούνται, έχοντας ως αποτέλεσμα το δίκτυο να συγκλίνει προς την απόκτηση μάθησης του προτύπου ή της επιθυμητής διεργασίας.

Το 1958, το μοντέλο του αισθητήρα (perceptron) πρωτοπαρουσιάστηκε από τον F. Rosenblatt [5]. Αυτό το μοντέλο αποτελούνταν από μόνο δύο επίπεδα, ένα εισόδου και ένα εξόδου, όπου το σήμα κινούνταν μονόδρομα από την είσοδο στην έξοδο. Ήταν ένα απλό και υποσχόμενο μοντέλο που έλκυε το ενδιαφέρον της επιστημονικής κοινότητας. Οι Minsky και Papert [6], εν τούτοις το 1969, στο βιβλίο τους "Perceptrons", ανακάλυψαν τους περιορισμούς αυτού του μοντέλου μέσω εμπεριστατωμένης έρευνας και το αντικατέστησαν με πιο προηγμένες μεθόδους. Στο ίδιο βιβλίο, εξέφρασαν την ωφέλεια του μοντέλου του αισθητήρα, καθώς και το δεσμευτικό του χαρακτήρα.

Ταυτόχρονα με την εξέλιξη του προτύπου του Percepton, αναπτύχθηκαν δύο καινούργια μοντέλα το Adaline και το Madaline από δύο άλλους επιστήμονες, τους Widrow και Hoff [7]. Τα μοντέλα αυτά χρησιμοποιήθηκαν σε καθημερινά προβλήματα πρακτικής φύσης, όπως η αφαίρεση της ηχώς στις γραμμές τηλεφώνου. Για τις επόμενες δύο δεκαετίες οι κατασταλτικοί παράγοντες και τα προβλήματα που δημιουργούσαν αυτοί στην εξέλιξη του τομέα, αποθάρρυνε πολλούς ερευνητές να εργαστούν πάνω στα νευρωνικά δίκτυα. Όμως το 1982 ο J. Hopfield [8] με το έργο του, έδωσε την παρακίνηση που θα τελείωνε το ερευνητικό αδιέξοδο στο τομέα. Ουσιαστικά απέδειξε με απόλυτο μαθηματικό τρόπο τη χρήση ενός νευρωνικού δικτύου ως αποθηκευτικού μέσου (storage device), καθώς και τη δυνατότητα ενός δικτύου να επαναφέρει όλη τη πλατφόρμα ενός συστήματος έχοντας μόνο μερικά τμήματα διαθέσιμα από το σύστημα.

Μία ακόμη σημαντική βαθμίδα στην εξέλιξη των νευρωνικών δικτύων ήταν η βελτίωση που πραγματοποιήθηκε στη προσέγγιση της εκπαίδευσης των δικτύων με την επινόηση του κανόνα διόρθωσης του σφάλματος (error correction learning) [9]. Με βάση το κανόνα αυτό, κατά τη διάρκεια εκπαίδευσης ενός δικτύου, ανεξαρτήτως της καταστασής του, μια καθορισμένη στιγμή, αυτό που μετράει είναι η διαφορά που δίνει στην έξοδο του το δίκτυο από την προσδοκώμενη τιμή. Η απόκλιση που παράγεται από το δίκτυο τη καθορισμένη στιγμή, καλείται σφάλμα και θέτει σε λειτουργία έναν μηχανισμό ελέγχου, του οποίου σκοπός είναι να προκαλέσει μια αλληλουχία από διορθωτικές μεταβολές στα βάρη w των νευρώνων. Μέσω αυτών των αλλαγών το δίκτυο έρχεται ένα σκαλοπάτι πιο κοντά στο να νοείται ως εκπαιδευμένο.

Στο βιβλίο των McClelland και Rumelhart [10] με τίτλο “Parallel Distributed Processing”, το οποίο δημοσιεύθηκε το 1986, εκφράζεται κάτι το πρωτοπόρο, η ιδέα δηλαδή ότι ένα νευρωνικό δίκτυο μπορεί να νοηθεί και να λειτουργήσει ως παράλληλος επεξεργαστής. Στο συγκεκριμένο έργο το μοντέλο Perceptron εμπλουτίζεται καθώς δίνει την δυνατότητα και σε άλλα επίπεδα νευρώνων να υφίστανται πέρα από αυτού της εισόδου και της εξόδου, διαμορφώνοντας την εσωτερική δομή του δικτύου.

Ως αποτέλεσμα των παραπάνω προόδων στο τομέα, γίνεται εισαγωγή το 1986 μίας καινούργιας μεθόδου εκπαίδευσης, του back-propagation. Ο συγκεκριμένος αλγόριθμος συνιστά έως και στις μέρες μας τη πιο επωφελή τεχνική εκπαίδευσης των δικτύων. Εγγενώς από τότε και έπειτα το πεδίο των νευρωνικών δικτύων διαμορφώνεται στη τωρινή του κατάσταση.

Ένας ουσιαστικός συντελεστής που επέτρεψε στο πεδίο των νευρωνικών δικτύων να καρποφορήσει, ήταν η πρόταση ενός πιο αποδοτικού τρόπου υπολογισμού των αναγκαίων παραγώγων για τη πραγματοποίηση της διαδικασίας του back-propagation. Ο εν λόγω μηχανισμός ακούει στο όνομα Automatic Differentation και προτάθηκε από μηχανικό ηλεκτρονικών υπολογιστών Seppo Ilmari Linnainmaa [11]. Μέσω του αλγόριθμου αυτού ο υπολογισμός των απαραίτητων παραγώγων βελτιστοποιείται έχοντας ως αποτέλεσμα την ελαχιστοποίηση του υπολογιστικού κόστους.

Στην εποχή την οποία διανύουμε, οι Yoshua Bengio, Geoffrey Hinton και Yann LeCun θεωρούνται ως οι κατεξοχήν ερευνητές οι οποίοι συνείσφεραν στην εξέλιξη του αλγορίθμου back-propagation καθώς και γενικά στον εμπλουτισμό του τρόπου λειτουργίας των ΤΝΔ. Όπως είναι αναμενόμενο μεταξύ των γεγονότων που προαναφέρθηκαν παρενέβησαν και άλλοι ακόμα ερευνητές με πολύ σημαντικό έργο .

Ταυτόχρονα για να επακολουθήσει το ζενίθ που διαπιστώνουμε σήμερα στο πεδίο των ΤΝΔ ήταν αναγκαία και η σταδιακή δημιουργία hardware υπολογιστών με ισχυρότερη απόδοση και πιο οικονομική τιμή, καθώς η εφαρμογή της εκπαίδευσης δικτύων με πολλούς νευρώνες είναι μια διαδικασία έντονα επίμοχθη υπολογιστικά. Στο hardware περιλαμβάνονται ισχυρές κεντρικές μονάδες επεξεργασίας CPU, κάρτες γραφικών GPU ή TPU (Tensor Processing Unit), αποδεκτού μεγέθους και κόστους για την ευκολότερη επέκταση της έρευνας και των εφαρμογών των ΤΝΔ.

Ως γενική εικόνα του τομέα της τεχνητής νοημοσύνης, υπάρχει μια επικρατούσα κατανόηση. Πιστεύεται ότι υπάρχουν δύο κοινοί και παράλληλα αναπτυσσόμενοι κλάδοι, δηλαδή η τεχνητή νοημοσύνη ΤΝ (Artificial Intelligence AI) και η υπολογιστική νοημοσύνη ΥΝ (Computational Intelligence CI). Η υπολογιστική νοημοσύνη είναι ένα υποσύνολο της τεχνητής νοημοσύνης. Η ΤΝ είναι ένας τομέας που δημιουργεί μεθόδους με ισχυρό θεωρητικό υπόβαθρο και το ΥΝ διακρίνεται να είναι αφιερωμένο στην ανάπτυξη αλγορίθμων εμπνευσμένων από τη φύση, όπως εξελικτικούς αλγόριθμους που μιμούνται ουσιαστικά τη διαδικασία της φυσικής επιλογής ή την ασαφή λογική που προσομοιάζει την προφορική γλώσσα ή τα ΤΝΔ που προσομοιάζουν τις λειτουργίες του εγκεφάλου. Υποσύνολο της ΥΝ αποτελεί η μηχανική μάθηση (Machine Learning ML) της οποίας υποσύνολο αποτελεί η βαθιά μάθηση (Deep Learning DL) δηλαδή εκεί που ανήκουν τα Τεχνητά Νευρωνικά Δίκτυα [12,13].

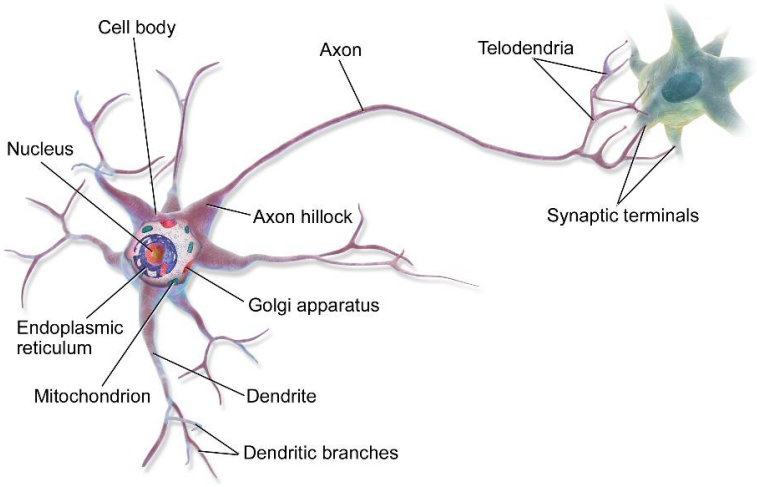
2.2 Βιολογικά Νευρωνικά Δίκτυα

Οι γώσεις που αποκτήθηκαν σχετικά με τη δομή και λειτουργία των Βιολογικών Νευρωνικών Δικτύων, και ειδικότερα τη λειτουργία του ανθρώπινου εγκεφάλου, αποτέλεσαν έναν σημαντικό πυλώνα στην ανάπτυξη των Τεχνητών Νευρωνικών Δικτύων (ΤΝΔ). Απλοϊκές διεργασίες που ακόμα και παιδιά μικρής ηλικίας δύναται να εκτελέσουν, ένας υπολογιστής, παρά τη τεράστια υπολογιστική ισχύ που διαθέτει σε σχέση με τον άνθρωπο, δυσχαιρένεται να πραγματοποιήσει. Κατά συνέπεια, η δομή και η λειτουργία των Βιολογικών Νευρωνικών Δικτύων αποτέλεσε την υπέρτατη πηγή έμπνευσης της εξέλιξης των ΤΝΔ. Κρίνεται αναγκαίο οπότε η κατανόηση ορισμένων κύριων σημείων στη δομή και τη λειτουργία των Βιολογικών Νευρωνικών Δικτύων, προτού κάνουμε αναφορά στα Τεχνητά Νευρωνικά Δίκτυα.

Το ανθρώπινο νευρικό σύστημα αποτελείται από νευρώνες, που αποτελούν τη βασική δομική μονάδα του. Οι νευρώνες είναι εξειδικευμένα κύτταρα που λειτουργούν ως οι πυρήνες των συστημάτων επεξεργασίας πληροφοριών στο κεντρικό νευρικό σύστημα του ανθρώπου. Αποτελούν θεμελιώδη συστατικά του εγκεφάλου, τόσο στον άνθρωπο όσο και στα ζώα. Εκτιμάται ότι ο εγκέφαλος ενός ενήλικα περιλαμβάνει περίπου 100 δισεκατομμύρια νευρώνες, και κάθε ένας από αυτούς συνδέεται με περίπου 10.000 άλλους νευρώνες μέσω συνάψεων. Οι συνάψεις αυτές αποτελούν τα σημεία επικοινωνίας μεταξύ των νευρώνων και ο αριθμός τους διαφέρει από κύτταρο σε κύτταρο.

Το σύνολο των νευρώνων και των συνάψεών τους αποτελεί ένα νευρωνικό δίκτυο, ενώ όλα τα νευρωνικά δίκτυα που υπάρχουν στον ανθρώπινο οργανισμό αποτελούν το Κεντρικό Νευρικό Σύστημα (ΚΝΣ). Οι νευρώνες, που είναι εξειδικευμένα κύτταρα, δεν αναπαράγονται ούτε πολλαπλασιάζονται, και έτσι ο υγιής ενήλικας χάνει καθημερινά νευρώνες, πράγμα που επηρεάζεται κυρίως από την ηλικία και τη κατανάλωση αλκοόλης. Σε αντίθεση με τους νευρώνες, οι συνάψεις δημιουργούνται και καταστρέφονται συνεχώς, βρίσκονται δηλαδή σε σταθερή ισορροπία. Ο εγκέφαλος, με τους νευρώνες ως δομικά στοιχεία, μεταφέρει πληροφορίες σε άλλα συστήματα του ανθρώπινου οργανισμού, όπως το πεπτικό σύστημα, και συμβάλλει στη συνεργασία και το συντονισμό της λειτουργίας τους.

Ο νευρώνας, ως η βασική μονάδα του νευρικού συστήματος, αποτελείται από διάφορα στοιχεία που επιτρέπουν την μετάδοση πληροφοριών. Η ανατομία του περιλαμβάνει το σώμα, τον πυρήνα, τους δενδρίτες, τον άξονα και τις συνάψεις. Συγκεκριμένα, το σώμα του νευρώνα αποτελεί το βασικό μέρος του, εντός του οποίου βρίσκεται ο πυρήνας του κυττάρου που περιέχει όλες τις γενετικές πληροφορίες. Οι δενδρίτες αποτελούν τα σημεία εισόδου των πληροφοριών, όπου ο κάθε νευρώνας λαμβάνει σήματα από τους προσκείμενους νευρώνες. Η πληροφορία ρέει στον νευρώνα μέσω των δενδριτών με μορφή ηλεκτρικών παλμών. Οι συνάψεις είναι τα σημεία όπου συνδέονται οι διακλαδώσεις του άξονα ενός νευρώνα με τους δενδρίτες άλλων νευρώνων. Η αμεσότητα μετάδοσης της ηλεκτρικής δραστηριότητας μεταξύ νευρώνων εξαρτάται από το πλάτος της σύναψης καθώς και από τη πυκνότητα του ηλεκτροχημικού υλικού. Η ποσότητα του ηλεκτρικού παλμού που διαβιβάζεται στους δενδρίτες του νευρώνα-παραλήπτη καλείται συναπτικό βάρος.



Εικόνα 1 (Δομή ενός βιολογικού νευρώνα Πηγή: https://www.biologyonline.com/dictionary/axon-terminal)

Οι νευρώνες υπάρχουν σε δύο δυνατές καταστάσεις: ενεργός και ανενεργός. Όταν ένας νευρώνας είναι ενεργός, πυροδοτεί ένα ηλεκτρικό σήμα, με το οποίο μεταφέρονται τα δεδομένα σε γειτονικούς νευρώνες. Αντίστοιχα, όταν ο νευρώνας παραμένει ανενεργός, δεν παράγει ηλεκτρικό σήμα. Η ευαισθησία του νευρώνα να αντιδράσει σε εξωτερικά ερεθίσματα, όπως ηλεκτρομαγνητικά, χημικά, θερμικά κ.ά., παράγει ηλεκτρικούς παλμούς που μεταφέρουν τις πληροφορίες. Ο κάθε νευρώνας συλλέγει όλο το ηλεκτρικό φορτίο από τις συνάψεις του στους δενδρίτες του. Το άθροισμα του φορτίου εισόδου επηρεάζει το αν είναι ενεργός ή ανενεργός. Εάν το συνολικό φορτίο είναι μεγαλύτερο από ένα κατώφλι, τότε ο νευρώνας παράγει ηλεκτρικούς παλμούς και θεωρείται ενεργός. Σε αντίθετη περίπτωση, ο νευρώνας είναι ανενεργός. Έτσι, μπορούμε να περιγράψουμε τον νευρώνα ως ένα δυαδικό στοιχείο [9].

2.3 Γενική Δομή Τεχνητών Νευρωνικών Δικτύων

Ένας μεμονωμένος τεχνητός νευρώνας είναι γνωστός ως perceptron. Η είσοδος ενός perceptron είναι ένα σύνολο τιμών $x \in \mathbb{R}^n$ και $y \in [-1,1]$ είναι η έξοδος που παράγεται, επίσης το y θεωρείται το μέτρο της διέγερσης του perceptron. Η λειτουργία του perceptron καθορίζεται από την εφαρμογή των 3 ακόλουθων πρακτικών στις εισόδους:

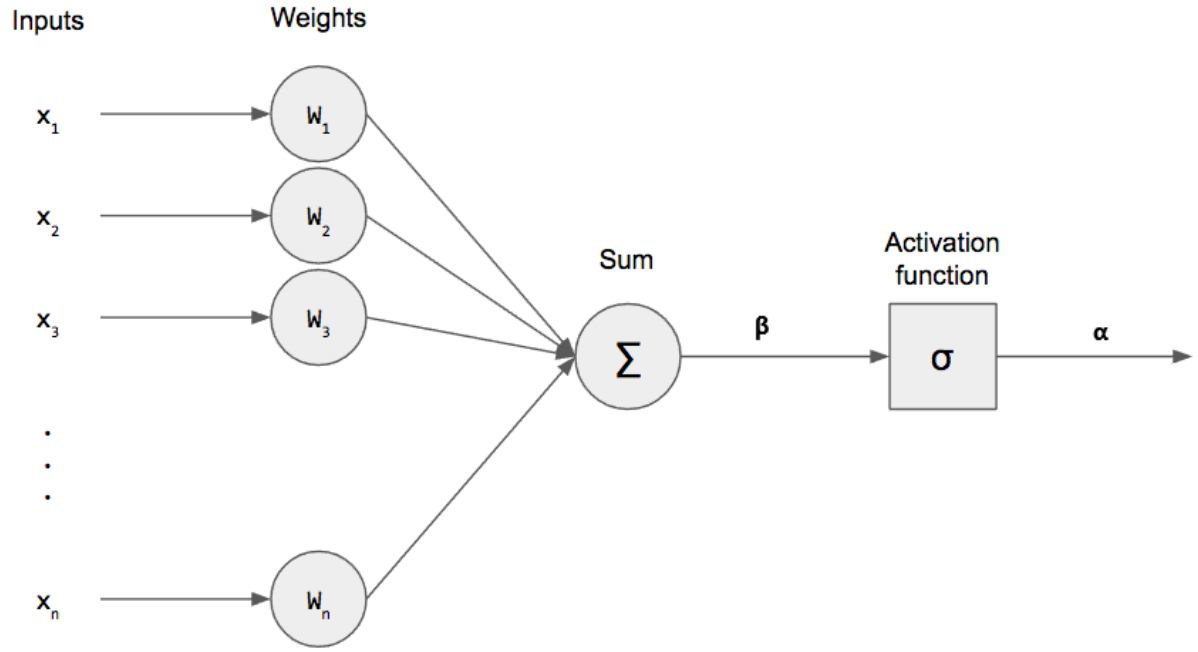
i. Το γινόμενο κάθε εισόδου $x_i \in x$ αθροίζεται με το αντίστοιχο βάρος $w_i \in w$. Καθε βάρος w_i μπορεί να ερμηνευθεί ως μέτρο ευαισθησίας του perceptron σε κάθε είσοδο x_i χωριστά. Για παράδειγμα, αν $w_i = 0$ τότε η είσοδος x_i δεν επιδρά καθόλου στην ενεργοποίηση του perceptron.

ii. Μια πόλωση β προστίθεται σε αυτό το άθροισμα. Η πόλωση ερμηνεύεται ως μέτρο του πόσο ενεργός ένας νευρώνας θα ήταν αν $x_i = 0$ για $i = 1, \ldots, n$.

iii. Γίνεται εκχώρηση της τιμής αυτής σε μια συνάρτηση ενεργοποίησης $\sigma : \mathbb{R} \to [-1,1]$, όπως η tanh.

Η ενεργοποίηση ενός perceptron α μπορεί επομένως να αναπαρασταθεί ως

$$\alpha = \sigma\left(\sum_{i=1}^{n} (x_i w_i) + \beta\right)$$



Εικόνα 2 (Perceptron: Το πρώτο Τεχνητό Νευρωνικό Δίκτυο Πηγή: https://medium.com/analytics-vidhya/perceptron-learning-algorithm-7ae7c4b90eb2)

Διαισθητικά λοιπόν, ένα ΤΝΔ F (που προσεγγίζει την f) είναι απλώς μια συλλογή από αυτά τα perceptons που τροφοδοτούνται μεταξύ τους, η ενεργοποίηση των προηγούμενων νευρώνων επηρεάζει την ενεργοποίηση των άλλων έτσι ώστε $F: x \rightarrow \hat{y}$. Κατά βάση, ένα ΤΝΔ είναι οργανωμένο σε $l=1,2,...,L$ στρώματα (layers), με $n^{[l]}$ αριθμό νευρώνων σε κάθε στρώμα. Οποιοδήποτε στρώμα μεταξύ του layer εισόδου $n^{[1]}$ και του layer εξόδου $n^{[L]}$ χαρακτηρίζεται ως κρυφό στρώμα (hidden layer) επειδή η κατάστασή του δεν είναι προσπελάσιμη από το χρήστη. Ο χρήστης έχει πρόσβαση μόνο στο layer εισόδου και το layer εξόδου.

Για $l=2,...,L-1$ η ενεργοποίηση $\alpha^{[l]}$ είναι επομένως:

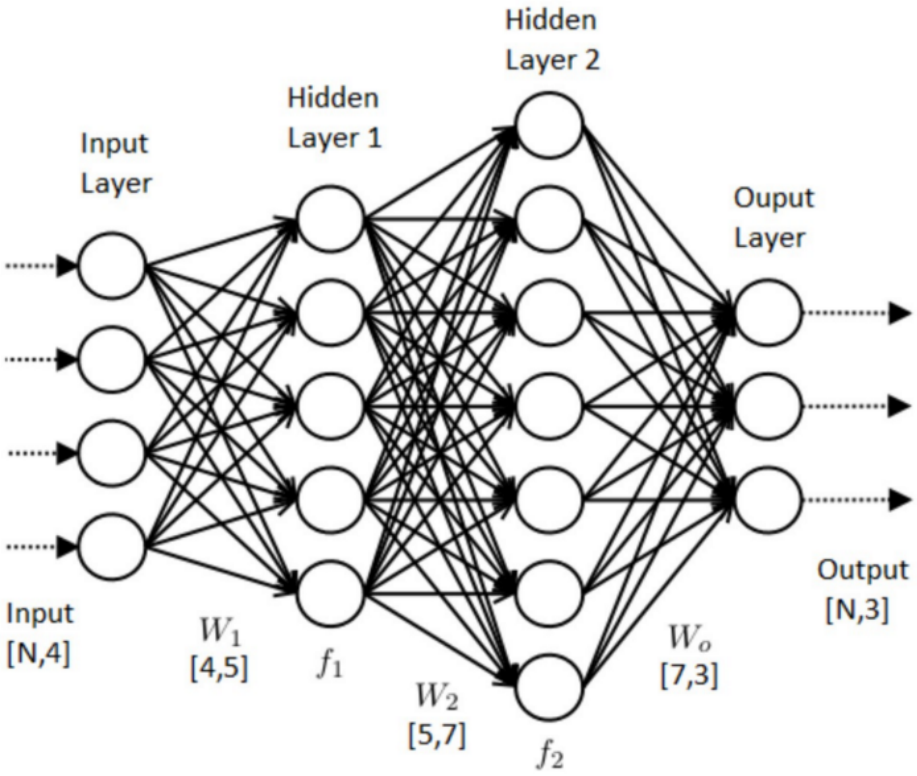
$$\begin{bmatrix} \alpha_1^{[l]} \\ \alpha_2^{[l]} \\ \vdots \\ \alpha_{n^{[l]}}^{[l]} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_1,1\, w_1,2 \ldots & w_{1,n^{[l-1]}} \\ w_2,1\, w_2,2 \ldots & w_{2,n^{[l-1]}} \\ \vdots \quad \vdots & \ddots \quad \vdots \\ w_{n^{[l]},1}\, w_{n^{[l]},2} \cdots & w_{n^{[l]},n^{[l-1]}} \end{bmatrix} \right) \begin{bmatrix} \alpha_1^{[l-1]} \\ \alpha_2^{[l-1]} \\ \vdots \\ \alpha_{n^{[l-1]}}^{[l-1]} \end{bmatrix} + \begin{bmatrix} \beta_1^{[l]} \\ \beta_2^{[l]} \\ \vdots \\ \beta_{n^{[l]}}^{[l]} \end{bmatrix}$$

που μπορεί να συμπυκνωθεί σε:

$$\alpha^{[l]} = \sigma\left(W^{[l]} \alpha^{[l-1]} + \beta^{[l]}\right)$$

όπου $\alpha^{[l]}, \beta^{[l]} \in \mathbb{R}^{n^{[l]}}$ και $W^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}$. Αυτός είναι ο μηχανισμός διάδοσης πληροφοριών μέσα στο σύστημα για τα ΤΝΔ ώστε η αντιστοίχιση της $f : x \rightarrow y$ να γίνει εφικτή.

Οι νευρώνες αναλόγως του τρόπου σύνδεσης τους, δημιουργούν ειδικούς τύπους δικτύων, όπως τα συνελικτικά δίκτυα (convolutional networks) ή τα επαναλαμβανόμενα δίκτυα (recurrent networks). Η Εικόνα 3 είναι ένα παράδειγμα ενός πλήρως συνδεδεμένου ΤΝΔ, όπου κάθε κόμβος στο επίπεδο $l-1$ συνδέεται με κάθε κόμβο στο l.



Εικόνα 3 (Ένα ΤΝΔ με 4 εισόδους, 2 κρυφά στρώματα και 3 εξόδους. Πηγή: https://www.researchgate.net/figure/Artificial-neural-network-of-multiple-layers-and-outputs-31_fig2_331097835)

Ένα σημαντικό αποτέλεσμα από τα ΤΝΔ είναι ότι ένα δίκτυο F με ένα μόνο κρυφό layer μπορεί να προσεγγίσει οποιαδήποτε συνεχή συνάρτηση f εντός μιας περιοχής I_n (η οποία έχει n διαστάσεις) με μία ακρίβεια $\varepsilon > 0$ τέτοια ώστε $|F(x) - f(x)| < \varepsilon \quad \forall \quad x \in I_n$ () υπό την προϋπόθεση ότι δεν τίθεται όριο στις τιμές των W και β (τα βάρη και τις πολώσεις του ΤΝΔ F). Επομένως η ακρίβεια καθορίζεται από την ποσότητα των νευρώνων. Αυτό είναι γνωστό ως καθολικό θεώρημα προσέγγισης [14,15].

Κεφάλαιο 3- Physics-Informed Neural Networks (PINNs)

3.1 Ιστορική Αναδρομή

Η πρώτη εργασία που υλοποίησε τη χρήση των ΤΝΔ ως μέσο επίλυσης διαφορικών εξισώσεων κατατίθεται από το τμήμα Επιστήμης Ηλεκτρονικών Υπολογιστών του Πανεπιστημίου Ιωαννίνων το 1997. Η εργασία αυτή είχε τίτλο “Artificial Neural Networks for Solving Ordinary and Partial Differential Equations” [16]. Ωστόσο οφείλουμε να αναφέρουμε ότι η αρχική ιδέα της προσπάθειας αυτής είχε προταθεί λίγα χρόνια νωρίτερα μέσα από εργασίες θεωρητικού υποβάθρου, για κάποιες από τις οποίες γίνεται παραπομπή στη βιβλιογραφία της παραπάνω. Κάποιες αντιπροσωπευτικές εργασίες από αυτές είναι η εργασία με τίτλο “Neural algorithms for solving differential equations” [17] και “The numerical solution of Linear Ordinary Differential Equations by Feedforward Neural networks” [18]. Οι ερευνητές των οποίων διαπιστώνουμε ότι προέρχονταν κατά βάση από το επιστημονικό πεδίο των εφαρμοσμένων μαθηματικών και της επιστήμης υπολογιστών.

Η εισαγωγή της μεθόδου στη Μηχανική και συγκεκριμένα στο πεδίο της ανάλυσης κατασκευών προήλθε μέσα από την εργασία “A neural network approach to the modelling, calculation and identification of semi-rigid connections in steel structures” [19]. Στα πρώτα δέκα χρόνια του 2000 δεν είχαμε κάποια αξιοσημείωτη αναφορά πάνω στο θέμα ούτε κάποια εφαρμογή. Η μέθοδος PINNs με τη σημερινή της μορφή παρουσιάζεται χαρακτηριστικά από την εργασία “Physics Informed Deep Learning (Part I): Data – driven Solutions of Nonlinear Partial Differential Equations” [20] στην οποία διατυπώνονται οι αρχές της μεθόδου, γίνεται ανάλυση της προγραμματιστικής υλοποίησης της και γίνεται εφαρμογή της στις Burger’s Equation και Schrödinger Equation, οι οποίες αποτελούν μερικές διαφορικές εξισώσεις. Η εν λόγω εργασία πρόκειται για ένα πολύ καλό εγχειρίδιο για την πρώτη επαφή κάποιου με το θέμα. Οι παραπάνω συγγραφείς έχουν δημοσιεύσει αρκετές εργασίες πάνω στη μέθοδο τα τελευταία έτη και με την συστηματική ενασχόληση τους υποδεικνύουν την αισιοδοξία τους για τη πορεία της μεθόδου και των δυνατοτήτων της. Μία ακόμα αντιπροσωπευτική εργασία με πιο αναλυτικές πληροφορίες για την εφαρμογή των PINNs έχει τίτλο “IDRLnet: A Physics-Informed Neural Network Library” [21].

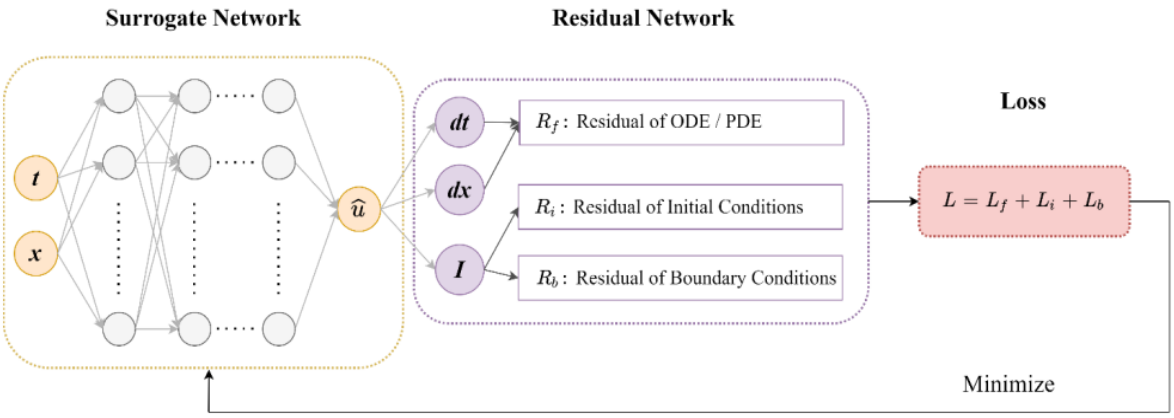
Η επιστημονική εξέταση της μεθόδου προβάλλει ιδιαίτερη άνοδο από το 2017 και μετά, διότι πέρα από τις προαναφερθείσες εργασίες, ταυτόχρονα έχουν δημοσιευθεί πολλές επιπρόσθετες από ερευνητές από όλα τα μήκη και πλάτη της Γης. Ως αντιπροσωπευτικές εργασίες του τελευταίου έτους αξίζει να αναφέρουμε την εργασία με τίτλο “Physics-informed machine learning” [22] και “Physics-Informed Neural Networks for elastic plate problems” [23] όπου στην εργασία αυτή, εφαρμόζεται η μέθοδος για την επίλυση της διαφορικής εξίσωσης που αφορά το πρόβλημα της πλάκας υπό κάμψη (Kirchhoff Plate Bending).

Όλες οι παραπάνω εργασίες και κάποιες επιπλέον αποτέλεσαν πηγή πληροφόρησης για την κατανόηση της μεθόδου, την εφαρμογή της και για την συγγραφή της παρούσας εργασίας.

3.2 Θεωρία των PINNs

Τα PINNs [24,25] είναι μια νέα κατηγορία δικτύων βαθιάς μάθησης που δύνανται να κωδικοποιήσουν διαφορικές εξισώσεις που διέπουν ένα σύνολο δεδομένων. Η διαφορά μεταξύ των PINNs και των παραδοσιακών λυτών διαφορικών εξισώσεων είναι ότι το PINN υπολογίζει διαφορικούς τελεστές χρησιμοποιώντας αυτόματη παραγώγιση. Η εφαρμογή των PINNs υφίσταται τόσο σε εποπτευόμενες όσο και σε μη εποπτευόμενες διαδικασίες μάθησης. Η διαδικασία εκπαίδευσης των PINNs απαιτεί ουσιαστικά λιγότερα δεδομένα από τις περισσότερες μεθόδους βαθιάς μάθησης, και τα δεδομένα δεν χρειάζονται επισήμανση. Οι προκλήσεις της εκπαίδευσης PINN αποτελούν την ικανοποίηση ανταγωνιστικών στόχων: εκμάθηση της λύσης της διαφορικής εξίσωσης εντός του πεδίου ορισμού, ενώ ικανοποιούνται οι αρχικές και οριακές συνθήκες (Initial Conditions και Boundary Conditions). Αυτό οδηγεί σε ασταθείς παραγώγους κατά τη διάρκεια της εκπαίδευσης του δικτύου μέσω gradient-based μεθόδων και συχνά προκαλεί τα PINNs να πασχίζουν στην ακριβή προσέγγιση της λύσης της διαφορικής εξίσωσης. Αυτή είναι μια γνωστή πρόκληση για μεθόδους που βασίζονται σε παραγώγους, όπου οι λύσεις που βρέθηκαν έχουν «κολλήσει» σε οριακούς κύκλους εάν υπάρχουν αρκετοί ανταγωνιστικοί στόχοι.

Τα PINNs λαμβάνουν ως είσοδο ένα σημείο στο υπολογιστικό πεδίο (που ονομάζεται collocation point) και ελαχιστοποιούν μια residual συνάρτηση (που ονομάζεται βήμα εκπαίδευσης). Η έξοδος τους είναι μια κατά προσέγγιση λύση της διαφορικής εξίσωσης. Η βασική πρόοδος του PINN είναι η ενσωμάτωση ενός residual δικτύου που περιέχει τις δίεπουσες εξισώσεις. Αυτό το δίκτυο, δεδομένης της εξόδου ενός δικτύου βαθιάς μάθησης (που ονομάζεται surrogate), υπολογίζει μια υπολειμματική τιμή (που ονομάζεται συνάρτηση απώλειας). Η Εικόνα 4 δείχνει ένα σχηματικό του PINN, όπου το υποκατάστατο μοντέλο αποτελείται από ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο με τις συντεταγμένες (t, x) ως εισόδους και χρησιμοποιείται για την κατασκευή $\hat{u}(t, x)$, που είναι μια προσέγγιση της αριθμητικής λύσης $u(t, x)$. Τα residual υπολογίζονται μετά για να ληφθεί η απώλεια L, η οποία χρησιμοποιείται για βελτιστοποίηση της προσέγγισης. Η υπολειπόμενη απώλεια αποτελείται από το residual της διαφορικής εξίσωσης L_f, το residual των αρχικών συνθηκών L_i, και το residual L_b των οριακών συνθηκών.



Εικόνα 4 (Σχηματική απεικόνιση ενός νευρωνικού δικτύου PINN. Πηγή: https://www.mdpi.com/1996-1073/15/20/7697)

Σε αυτή την εργασία, υλοποιούμε το υποκατάστατο δίκτυο του PINN χρησιμοποιώντας πλήρως συνδεδεμένα νευρωνικά δίκτυα (FNN). Γενικά, ποικίλα νευρωνικά δίκτυα έχουν αναπτυχθεί, μεταξύ των οποίων το νευρωνικό δίκτυο τροφοδοσίας (FNN), το συνελικτικό νευρωνικό δίκτυο (CNN) και το επαναλαμβανόμενο νευρωνικό δίκτυο (RNN). Κατά κανόνα, όλα τα δίκτυα που έχουν τουλάχιστον δύο στρώματα $(L \geq 2)$ ονομάζονται βαθιά. Το FNN (Fully Connected Neural Network) είναι ένα νευρωνικό δίκτυο επιπέδου L, με $(L-1)$ κρυφά στρώματα. Ένα FNN συμβολίζεται με $N^L(x)$: $\Re^{d_{in}} \rightarrow \Re^{d_{out}}$, όπου τα d_{in} και d_{out} αντιστοιχούν στις διαστάσεις της εισόδου και της εξόδου. Ο αριθμός των νευρώνων στο ℓ-στό στρώμα συμβολίζεται με N^ℓ. Ο αριθμός των νευρώνων στα στρώματα εισόδου και εξόδου συμβολίζεται με $N^0 = d_{\text{in}}$ και $N^L = d_{out}$, αντίστοιχα.

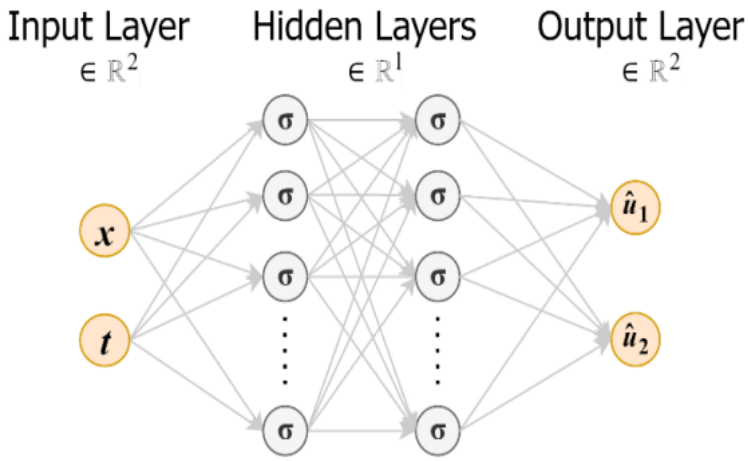
Ορίζουμε έναν πίνακα βάρους W^ℓ, μια πόλωση b^ℓ, και μια συνάρτηση ενεργοποίησης σ σε κάθε επίπεδο ℓ. Τα βάρη αυτού του δικτύου είναι εκπαιδεύσιμα. Τα βάρη και οι πολώσεις αποτελούν τις παραμέτρους θ του νευρωνικού δικτύου. Ένα DNN L επιπέδων ορίζεται ως:

Επίπεδο εισόδου: $N^0(x) = x \in \Re^{d_{in}}$ (1)

Κρυφό Επίπεδο: $N^\ell(x) = \sigma(W^\ell \cdot N^{\ell-1}(x) + b^\ell)$ (2)

Επίπεδο εξόδου: $N^L(x) = W^L \cdot N^{L-1}(x) + b^L \in \Re^{d_{out}}$ (3)

Ή έξοδος FNN είναι η προβλεπόμενη προσέγγιση της λύσης της διαφορικής εξίσωσης, $\hat{u}(t, x)$.



Εικόνα 5 (Αρχιτεκτονική FNN για την υλοποίηση υποκατάστατου μοντέλου PINN. Πηγή: https://www.mdpi.com/1996-1073/15/20/7697)

3.3 Αυτόματη Παραγώγιση

Η εκπαίδευση PINN [24,25] περιλαμβάνει την υλοποίηση ενός πλαισίου για τον υπολογισμό των παραγώγων. Σύνηθη γραφή των παραγώγων είναι με τη μορφή Ιακωβιανών ή Εσσιανών πινάκων. Κατά βάση, οι μέθοδοι υπολογισμού περιλαμβάνουν την αριθμητική παραγώγιση, τη συμβολική παραγώγιση και την αυτόματη παραγώγιση (Α.Π.: ονομάζεται επίσης αλγοριθμική παραγώγιση). Η αριθμητική πεπερασμένη διαφορά και η συμβολική παραγώγιση ενίοτε οδηγούν σε μειωμένη ακρίβεια για σύνθετες συναρτήσεις. Η Α.Π. υπολογίζει αυτόματα τις παραγώγους χρησιμοποιώντας τον κανόνα αλυσίδας για τη συσσώρευση τιμών, αντί να βασίζεται σε κανόνες παραγώγισης.

Δεδομένου ότι τα νευρωνικά δίκτυα αντιπροσωπεύουν μια συνθετική συνάρτηση, η Α.Π. βασίζεται στην επανειλημμένη εφαρμογή του κανόνα της αλυσίδας για τον υπολογισμό των παραγώγων. Επιπλέον, η Α.Π. χρησιμοποιεί την τεχνική του back-propagation, η οποία βοηθά στον ακριβή συντονισμό των βαρών ενός νευρωνικού δικτύου, χρησιμοποιώντας το σφάλμα που προκύπτει στην προηγούμενη επανάληψη. Η γενίκευση του μοντέλου καθώς και η μείωση του σφάλματος εξαρτάται από τη επιλογή των κατάλληλων βαρών. Η παράγωγος της αντικειμενικής συνάρτησης ως προς οποιοδήποτε βάρος ή πόλωση, το οποίο οπισθοδιαδίδεται στο δίκτυο, μας πληροφορεί λεπτομερώς για τον τρόπο με τον οποίο οι αλλαγές που υπόκεινται τα βάρη και οι πολώσεις επηρεάζουν τη συνολική συμπεριφορά του δικτύου. Η Α.Π. αξιολογεί τις παραγώγους σε δύο βήματα. Πρώτον, το μπροστινό πέρασμα (forward pass) υπολογίζει τις τιμές όλων των μεταβλητών. Δεύτερον, το backward pass υπολογίζει τις παραγώγους. Επομένως, η Α.Π. ανεξαρτήτως της διάστασης της εισόδου χρειάζεται μόνο ένα πέρασμα προς τα εμπρός και ένα πέρασμα προς τα πίσω για τον υπολογισμό όλων των παραγώγων.

3.4 Επιβολή αρχικών και οριακών συνθηκών

Στα PINNs υφίστανται δύο βήματα [24] στη διαδικασία κωδικοποίησης των Α.Σ και των Ο.Σ.

Πρώτον, κάθε Α.Σ και Ο.Σ μπορεί να έχει ανεξάρτητη διαχείριση, με το καθένα να εμφανίζεται στη συνάρτηση απώλειας ως ξεχωριστός όρος. Αυτός ο τύπος επιβολής είναι γνωστός ως ήπιος περιορισμός, όπου ο στόχος είναι η ελαχιστοποίηση κάθε όρου στη συνάρτηση απώλειας ταυτόχρονα. Τα PINNs συμβολίζονται ως $\hat{u}(x)=N(x;\theta)$. Η μέθοδος αυτή θεωρείται πιο απλοϊκή στην εφαρμογή και συνίσταται για διαφορικές εξισώσεις υψηλών διαστάσεων και σύνθετες γεωμετρίες.

Δεύτερον, υφίσταται να μετασχηματίσουμε την έξοδο του δικτύου σε κάποια συνάρτηση, έτσι ώστε οι Α.Σ και οι Ο.Σ να ικανοποιούνται εκ κατασκευής. Για παράδειγμα, τα PINNs μπορούν να χρησιμοποιήσουν τη σχέση $\hat{u}(x)=u_0+x\cdot N(x;\theta)$ ώστε να ικανοποιούν πάντα τη Α.Σ $u(x=0)=u_0$. Ως εκ τούτου, η συνάρτηση συνολικής απώλειας περιλαμβάνει μόνο τις απώλειες που προέρχονται από τις Δ.Ε. Αυτός ο τύπος επιβολής είναι γνωστός ως σκληρός περιορισμός, ο οποίος διασφαλίζει ότι οι Α.Σ ή οι Ο.Σ ικανοποιούνται πλήρως. Αυτό μειώνει την πολυπλοκότητα της εκπαίδευσης των PINNs.

3.5 Συνάρτηση απώλειας (Loss Function) και δείκτες για αξιολόγηση

Η ασυμφωνία μεταξύ του νευρωνικού δικτύου $\hat{u}$ και των περιορισμών που επιβάλλονται από τις Δ.Ε και τις αντίστοιχες Α.Σ ή/και Ο.Σ τους υπολογίζεται από τη συνάρτηση απώλειας [24]. Ως συνάρτηση απώλειας ορίζουμε τη σταθμισμένη άθροιση της L^2 νόρμας υπολειμμάτων για τη Δ.Ε και τις Α.Σ/Ο.Σ, με τη μορφή μέσου τετραγώνου σφάλματος (MSE), ως:

$$L(\theta;T)=w_f L_f(\theta:T_f)+w_i L_i(\theta:T_i)+w_b L_b(\theta:T_b)$$

Όπου w_f, w_i και w_b είναι τα βαθμωτά βάρη για τη Δ.Ε, Α.Σ και Ο.Σ, αντίστοιχα. Πριν από την προπόνηση καθορίζονται τα βαθμωτά βάρη, τα οποία χρησιμοποιούνται για την κατάταξη της βαρύτητας του κάθε όρου της απώλειας. Οι μεταβλητές T_f, T_i και T_b είναι τα σημεία εκπαίδευσης μέσα στο πεδίο ορισμού, στις Α.Σ και στις Ο.Σ. Αυτά τα σύνολα σημείων αναφέρονται συνήθως ως σύνολα υπολειπόμενων σημείων (residual points). Οι όροι L_f, L_i και L_b είναι το MSE των υπολοίπων για τη Δ.Ε, Α.Σ και Ο.Σ, αντίστοιχα.

Αυτά υπολογίζονται ως εξής:

$$L_f(\theta;T_f)=\frac{1}{|T_f|}\sum_{x\in T_f}\left\|f\left(x:\frac{d\hat{u}}{dx}\right)\right\|_2^2$$

$$L_i(\theta;T_i)=\frac{1}{|T_i|}\sum_{x\in T_i}\|I(\hat{u},x)\|_2^2$$

$$L_b(\theta;T_b)=\frac{1}{|T_b|}\sum_{x\in T_b}\|B(\hat{u},x)\|_2^2$$

Οι παράγωγοι στους όρους της συνάρτησης απωλειών υπολογίζονται με αυτόματη παραγώγιση. Για τον υπολογισμό των αρχικών και τελικών επιδόσεων του εκπαιδευμένου μοντέλου, χρησιμοποιούνται δύο μετρήσεις. Η πρώτη μέτρηση είναι η τιμή της συνάρτησης απώλειας στην $L(\theta;T)=w_f L_f(\theta:T_f)+w_i L_i(\theta:T_i)+w_b L_b(\theta:T_b)$. Η δεύτερη μέτρηση είναι το σχετικό σφάλμα L_2 σε σχέση με μια λύση αναφοράς $u(x)$. Η λύση αναφοράς $u(x)$ λαμβάνεται συνήθως από μια αριθμητική λύση υψηλής πιστότητας της ΔΕ. Το σχετικό σφάλμα L_2 ορίζεται ως:

$$L_2(u(x),\hat{u}(x))=\frac{\|u(x)-\hat{u}(x)\|_2}{\|u(x)\|_2}$$

Όπου $\|\ \ \|_2$ υποδηλώνει τον τυπικό κανόνα L_2 . Αξίζει να σημειωθεί ότι το σφάλμα στη συνάρτηση απώλειας L μετρά τον βαθμό στον οποίο ικανοποιείται η εξίσωση και όχι το σφάλμα της λύσης $\hat{u}$ σε σχέση με τη λύση αναφοράς u.

3.6 Συνάρτηση Ενεργοποίησης

Η συνάρτηση ενεργοποίησης [24,25,26] χρησιμοποιείται για την αντιστοίχιση της είσοδο x σε μια έξοδο y. Γενικά, σε διαφορετικά μέρη του FNN γίνεται χρήση διαφορετικών συναρτήσεων ενεργοποίησης. Η συνάρτηση ενεργοποίησης στο κρυφό επίπεδο ελέγχει πόσο καλά το μοντέλο δικτύου μαθαίνει το σύνολο των δεδομένων εκπαίδευσης, ενώ η συνάρτηση ενεργοποίησης στο επίπεδο εξόδου ορίζει τον τύπο των προβλέψεων που μπορεί να κάνει το μοντέλο. Μερικές από τις πιο διαδεδομένες συναρτήσεις ενεργοποίησης που χρησιμοποιούνται στη μηχανική μάθηση είναι η βηματική, η σιγμοειδής, η υπερβολική εφαπτομένη (tanh), και η Rectified Linear Unit (ReLU).

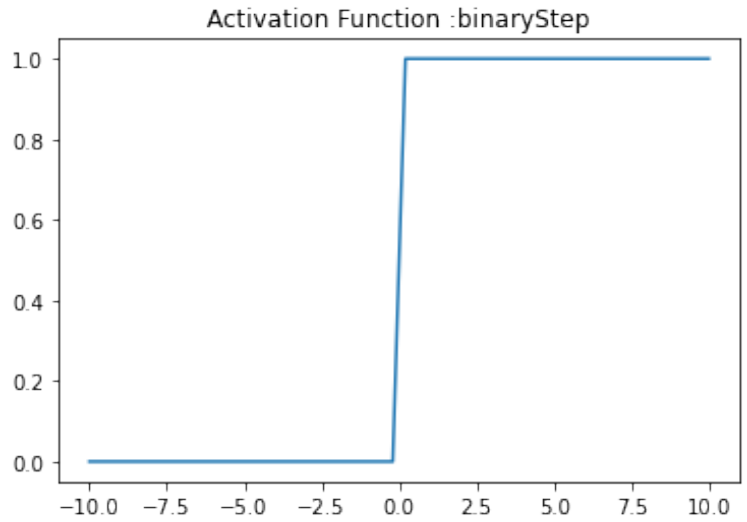
Η Βηματική συνάρτηση έχει ευρεία χρήση σε νευρωνικά δίκτυα με μόνο ένα νευρώνα και ορίζεται ως εξής:

$$f(net)=a\ \ \alpha v\ \ net<c\ \ \acute{\eta}\ \ b\ \ \alpha v\ \ net>c$$

Η ποσότητα net, παίρνει τις εξής 3 τιμές: a, b και $\frac{a+b}{2}$.

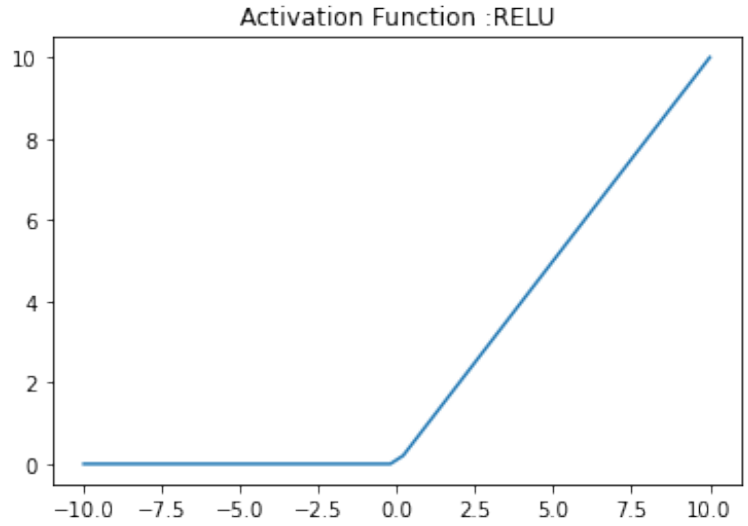
Οι πιο συνήθεις τιμές των a, b, c είναι: $a=0$, $b=1$, $c=0$ ή $a=-1$, $b=1$, $c=0$. Η δεύτερη περίπτωση Βηματικής συνάρτησης ονομάζεται και συνάρτηση Προσήμου (Signum Function). Η Βηματική συνάρτηση, παρουσιάζοντας μία πολύ απλουστευμένη λογική εφαρμόζεται σε απλά μοντέλα ενός νευρώνα. Πιο ειδικά, οι συναρτήσεις αυτές, βασίζονται στην ιδέα ύπαρξης ενός κατωφλιού c. Έτσι, όταν η τιμή της τελικής εξόδου του δικτύου είναι μικρότερη από την τιμή του κατωφλιού, τότε ο νευρώνας παραμένει ανενεργός. Αντιθέτως, όταν η τιμή της τελικής απόκρισης του δικτύου, είναι μεγαλύτερη του κατωφλιού, τότε ο νευρώνας είναι ενεργός και διαδίδει το σήμα του, στους προσκείμενους νευρώνες. Παρ'όλη την απλοϊκότητα της, η Βηματική συνάρτηση δεν

θεωρείται χρήσιμη στα ΤΝΔ, καθώς μειονεκτεί στο γεγονός ότι η παράγωγος της είναι μηδέν. Λαμβάνοντας υπόψιν ότι μάθηση στα ΤΝΔ είναι η μεταβολή των τιμών των βαρών και η μεταβολή σχετίζεται με την παράγωγο, η βηματική συνάρτηση δεν συνίσταται ως συνάρτηση ενεργοποίησης των νευρώνων στα ΤΝΔ.



Σχήμα 3.1: (Βηματική συνάρτηση. Πηγή: https://www.nbshare.io/notebook/751082217/Activation-Functions-In-Python/)

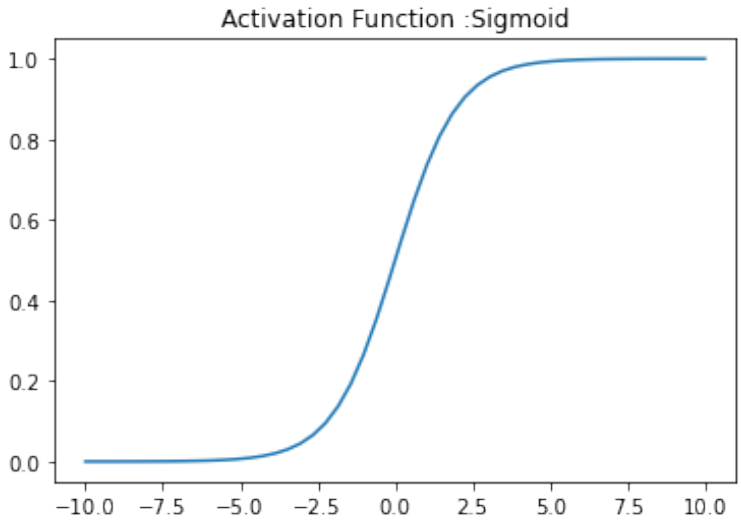
Η συνάρτηση ενεργοποίησης ReLU που ορίζεται ως $y = max\{0, x\}$ έχει το πλεονέκτημα σε σύγκριση με την σιγμοειδή ή την tanh να είναι υπολογιστικά ταχύτατη,. Η ReLU είναι λιγότερο επιρρεπής στο μηδενισμό παραγώγων που επηρεάζουν τη βαθιά εκπαίδευση μοντέλων. Εκτός από την ταχεία διαδικασία εκπαίδευσης, η ReLU αποφεύγει το κορεσμό σε μεγάλους θετικούς αριθμούς. Επιπλέον, η μη γραμμικότητα επιτρέπει τη διατήρηση και την εκμάθηση μοτίβων στα δεδομένα. Ωστόσο, σε εφαρμογές παλινδρόμησης, η συνάρτηση ReLU πάσχει από φθίνουσα ακρίβεια για παραγώγους δεύτερης και υψηλότερης τάξης.



Σχήμα 3.2: (Συνάρτηση ReLU. Πηγή: https://www.nbshare.io/notebook/751082217/Activation-Functions-In-Python/)

Η σιγμοειδής συνάρτηση ενεργοποίησης, που ονομάζεται επίσης λογιστικό σιγμοειδές, παίρνει οποιαδήποτε πραγματική τιμή ως είσοδο και παράγει τιμές εξόδου στην περιοχή από 0 έως 1. Η συνάρτηση ενεργοποίησης σιγμοειδούς ορίζεται ως $\sigma(x)=1/(1+e^{(-x)})$ και έχει τη μορφή σχήματος S.

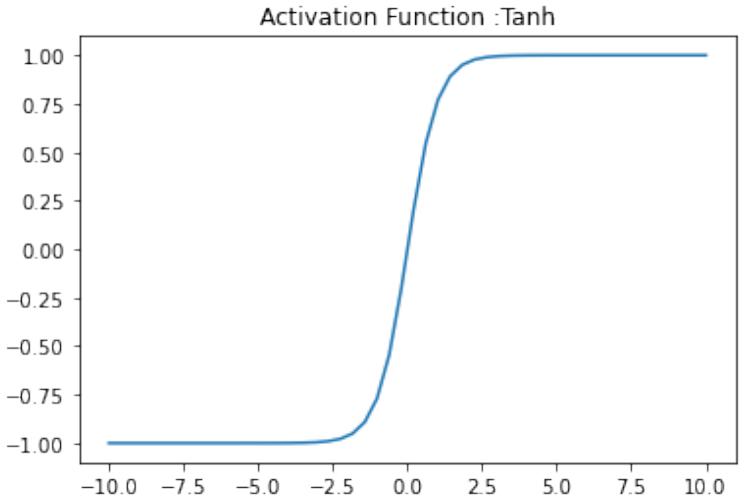
Όσο μεγαλύτερη είναι η είσοδος, τόσο πιο κοντά η τιμή εξόδου θα είναι στο 1, ενώ όσο μικρότερη είναι η είσοδος, τόσο πιο κοντά η έξοδος θα είναι στο 0. Επίσης, σε αντίθεση με τη συνάρτηση ενεργοποίησης ReLU, η συνάρτηση ενεργοποίησης σιγμοειδούς μπορεί να ξεπεράσει το πρόβλημα της φθίνουσας ακρίβειας σε παραγώγους υψηλότερης τάξης.



Σχήμα 3.3: (Συνάρτηση Sigmoid. Πηγή: https://www.nbshare.io/notebook/751082217/Activation-Functions-In-Python/)

Η συνάρτηση ενεργοποίησης υπερβολικής εφαπτομένης (tanh), είναι παρόμοια με την ενεργοποίηση σιγμοειδούς. Η συνάρτηση ενεργοποίησης tanh ορίζεται ως $\tanh = {(e^{x} - e^{-x})} \Big/ {(e^{x} + e^{-x})}$ και έχει σχήμα S.

Η Tanh παίρνει οποιαδήποτε πραγματική τιμή ως είσοδο και παράγει τιμές εξόδου μεταξύ του -1 και 1. Παρόμοια με τη συνάρτηση ενεργοποίησης σιγμοειδούς, το tanh μπορεί να ξεπεράσει τα προβλήματα που σχετίζονται με τη φθίνουσα ακρίβεια των παραγώγων υψηλότερης τάξης. Γενικά, και οι δύο συναρτήσεις ενεργοποίησης, σιγμοειδής και tanh χρησιμοποιούνται συνήθως για προβλήματα ταξινόμησης, λόγω της ευαισθησίας τους γύρω από ένα κεντρικό σημείο και της ικανότητας κατηγοριοποίησης στοιχείων σε διαφορετικές κατηγορίες.



Σχήμα 3.4: (Συνάρτηση Tanh. Πηγή: https://www.nbshare.io/notebook/751082217/Activation-Functions-In-Python/)

3.7 Βελτιστοποίηση

Ένας αλγόριθμος βελτιστοποίησης [24] επιταχύνει τη διαδικασία εκπαίδευσης μεταβάλλοντας σταδιακά τα βάρη του δικτύου ελαχιστοποιώντας τη συνάρτηση απώλειας. Αυτό συνήθως επιτυγχάνεται με τη χρήση βελτιστοποιητών gradient-based, οι οποίοι υπολογίζουν τις παραγώγους της συνάρτησης απώλειας σε σχέση με όλα τα βάρη και τις πολώσεις. Οι παράγωγοι υπολογίζονται ανάποδα από την έξοδο στην είσοδο, εφαρμόζοντας τον κανόνα αλυσίδας στρώμα προς στρώμα. Η ροή προς τα πίσω των παραγώγων είναι ο λόγος για την ονομασία της διαδικασίας ως backward pass ή back-propagation. Γενικά, υπάρχουν δύο βελτιστοποιητές που χρησιμοποιούνται πιο συχνά, ο Adam optimizer και ο περιορισμένης μνήμης βελτιστοποιητής Broyden-Fletcher-Goldfarb-Shanno (L-BFGS).

Ο αλγόριθμος βελτιστοποίησης Adam (ακρωνύμιο της «adaptive moment estimation») είναι μια επέκταση της stochastic gradient descent. Ο αλγόριθμος Adam ενημερώνει τα βάρη του δικτύου επανειλημμένα κατά τη διάρκεια της εκπαίδευσης εκτελώντας δύο λειτουργίες. Το πρώτο, που ονομάζεται ορμή, υπολογίζει τον εκθετικά σταθμισμένο μέσο όρο των παραγώγων. Το δεύτερο, που ονομάζεται μέση τετραγωνική ρίζα διάδοσης (RMSP), υπολογίζει τον κινητό εκθετικό μέσο όρο. Χρησιμοποιώντας μέσους όρους, ο αλγόριθμος συγκλίνει πιο γρήγορα προς τα ελάχιστα. Αυτό επιτυγχάνεται με την ελαχιστοποίηση των ταλαντεύσεων όταν πλησιάζει το ολικό ελάχιστο, ενώ κάνει αρκετά μεγάλα βήματα για να ξεφύγει από τα τοπικά ελάχιστα.

Ο αλγόριθμος βελτιστοποίησης L-BFGS είναι μια ψευδο-Νευτώνια μέθοδος που προσεγγίζει τον αλγόριθμο Broyden-Fletcher-Goldfarb-Shanno (BFGS) και χρησιμοποιεί λιγότερη μνήμη υπολογιστή. Ο L-BFGS χρησιμοποιεί μια εκτίμηση του αντίστροφου Εσσιανού πίνακα H^{-1} για να κατευθύνει την αναζήτησή του μέσω του χώρου των παραμέτρων και αποθηκεύει μόνο μερικά διανύσματα που αντιπροσωπεύουν έμμεσα την προσέγγιση. Ο αλγόριθμος L-BFGS είναι ιδιαίτερα κατάλληλος για προβλήματα βελτιστοποίησης με πολλές μεταβλητές λόγω της γραμμικής κλιμάκωσης της μνήμης.

3.8 Αρχικοποίηση

Οι αλγόριθμοι βελτιστοποίησης [24] χρειάζονται ένα αρχικό σύνολο βαρών με το οποίο θα ξεκινήσει η διαδικασία βελτιστοποίησης. Η επιλογή των αρχικών τιμών θα μπορούσε να επηρεάσει τη σύγκλιση του αλγορίθμου βελτιστοποίησης, με λανθασμένη επιλογή να οδηγεί πιθανώς σε πλήρη αποτυχία σύγκλισης.

Τα Βαθιά Νευρωνικά Δίκτυα χωρίς σωστή αρχικοποιήση βαρών μπορεί να υποφέρουν από κλίσεις που εξαφανίζονται ή εκρήγνυνται, το οποίο συμβαίνει όταν το σήμα που μεταδίδεται από στρώμα σε στρώμα σταματά να ρέει ή γίνεται κορεσμένο.

Για την αποφυγή της εμφάνισης τέτοιων προβλημάτων, η απόκλιση των εξόδων κάθε στρώματος θα πρέπει να είναι ίση με την απόκλιση των εισόδων του. Αντίστοιχα, η απόκλιση των παραγώγων θα πρέπει να είναι ίση πριν και μετά τη ροή μέσα από ένα στρώμα κατά τη διάρκεια της οπισθοδιάδοσης. Δύο από τις πιο συχνά χρησιμοποιούμενες τεχνικές αρχικοποίησης είναι η Glorot (επίσης γνωστή ως Xavier) και η Kaiming (επίσης γνωστή ως He) αρχικοποίηση.

Η στρατηγική αρχικοποίησης που ονομάστηκε από τον Xavier Glorot αρχικοποιεί τα βάρη του νευρωνικού δικτύου έτσι ώστε η απόκλιση των συναρτήσεων ενεργοποίησης να είναι ίδια σε κάθε επίπεδο. Σαν αποτέλεσμα, η σταθερή απόκλιση οδηγεί σε ομαλές ενημερώσεις κλίσης και ξεπερνά το πρόβλημα της έκρηξης ή της εξαφάνισης της κλίσης.

Οι πολώσεις αρχικοποιούνται με μηδέν, ενώ τα βάρη αρχικοποιούνται τυχαία σε κάθε επίπεδο ως:

$$W_{ij} = U[-1/\sqrt{n}, 1/\sqrt{n}]$$

όπου U είναι μια ομοιόμορφη κατανομή πιθανότητας και n το μέγεθος του προηγούμενου επιπέδου.

Η στρατηγική αρχικοποίησης Kaiming He είναι παρόμοια με την Glorot. Σε αυτή, τα βάρη σε κάθε στρώμα αρχικοποιούνται λαμβάνοντας υπόψη τις τιμές του προηγούμενου στρώματος, επιτυγχάνοντας έτσι το ολικό ελάχιστο της συνάρτησης κόστους ταχύτερα και πιο αποτελεσματικά. Τα βάρη είναι τυχαία, αλλά διαφέρουν ως προς το εύρος ανάλογα με τις τιμές του προηγούμενου στρώματος νευρώνων. Οι πολώσεις αρχικοποιούνται με μηδέν, ενώ τα βάρη αρχικοποιούνται τυχαία σε κάθε επίπεδο ως:

$$W_{ij} = G[0, \sqrt{2}/\sqrt{n}]$$

όπου G είναι μια κατανομή πιθανότητας Gauss και n είναι το μέγεθος του προηγούμενου στρώματος. Η Kaiming He Normal αρχικοποιήση χρησιμοποιείται συνήθως σε συνδυασμό με τη συνάρτηση ενεργοποίησης ReLU.

Κεφάλαιο 4- Ορισμός προβλημάτων προς επίλυση και προγραμματισμός

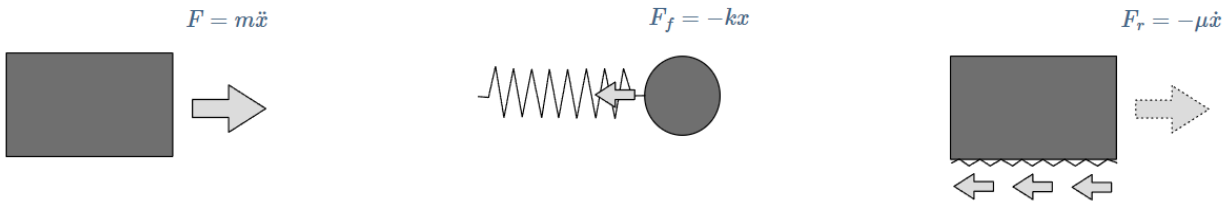
4.1 Πρόβλημα 1 : Πρόβλεψη θέσης στο χρόνο ενός αρμονικού ταλαντωτή με απόσβεση

Ένα από τα πιο κλασικά προβλήματα της μηχανικής αποτελεί ο αρμονικός ταλαντωτής με απόσβεση [27]. Κάνουμε λόγο για την αναπαράσταση της κίνησης ενός ταλαντωτή (π.χ. εκκρεμές ελατηρίου) το οποίο υπόκειται σε δύναμη επαναφοράς και δύναμη τριβής. Παρακάτω θα αναλύσουμε πως προκύπτει η εξίσωση της ταλάντωσης για το ταλαντωτή με απόσβεση.

Η εξίσωση της κίνησης

Η εξέλιξη ενός δυναμικού συστήματος που υπόκειται σε εξωτερικές δυνάμεις, στο χώρο και στο χρόνο περιγράφεται αναλυτικά από την εξίσωση κίνησης του. Η σύνηθης μορφή τους είναι διαφορικές εξισώσεις δεύτερης τάξης. Σε αυτή τη παράγραφο θα αναλύσουμε αρχικά τις δυνάμεις που διαμορφώνουν την ταλάντωση.

Οι δυνάμεις αυτές είναι τρείς: η αδράνεια, η δύναμη επαναφοράς του ελατηρίου και της τριβής. Η ισορροπία των δυνάμεων αυτών διαμορφώνει τη κίνηση του εκκρεμούς.



Εικόνα 6 (Αναπαράσταση των 3 δυνάμεων που επηρεάζουν τη κίνηση του εκκρεμούς. Πηγή: https://beltoforion.de/en/harmonic_oscillator/)

Σύμφωνα με το δεύτερο νόμο του Νεύτωνα οποιαδήποτε μεταβολή της κίνησης ενός σώματος είναι αντίστοιχη της δύναμης που του ασκείται καθώς επίσης και ότι αυτή η μεταβολή στη κίνηση πραγματοποιείται πάντοτε στην ίδια κατεύθυνση με αυτή της δραστικής δύναμης.

Η μάζα του εκκρεμούς εκτρέπεται προς τη θέση ισορροπίας της από τη δύναμη επαναφοράς F_f η οποία είναι ευθέως ανάλογη με τη εκτροπή του εκκρεμούς. Όμως κατευθύνεται αντίθετα από την εκτροπή του εκκρεμούς. Πολλαπλασιάζοντας την απόσταση εκτροπής με τη σταθερά του ελατηρίου k υπολογίζουμε το μέγεθος της F_f .

Όσον αφορά τη τριβή που θα αντιμετωπίσει το σύστημα, η δύναμη τριβής F_r εξαρτάται από τη ταχύτητα του εκκρεμούς και κατευθύνεται αντίθετα της φοράς κίνησης του σώματος. Ο συντελεστής τριβής συμβολίζεται με μ και η τιμή του καθορίζεται από το υλικό και το σχήμα του εκκρεμούς (βλ. επίσης νόμος Stokes).

Η δύναμη αδράνειας οπότε είναι αντίθετη των δύο αυτών δυνάμεων:

$$m\ddot{x} = -\mu\dot{x} - kx$$

το οποίο μπορεί να μετατραπεί σε:

$$\ddot{x} + \frac{\mu}{m}\dot{x} + \frac{k}{m}x = 0 \quad (1)$$

Πρόκειται για μια γραμμική, ομογενή διαφορική εξίσωση δεύτερης τάξης με συντελεστές σταθερής τιμής. Συνήθως για την επίλυση διαφορικών εξισώσεων τέτοιου τύπου επιλέγεται η εκθετική δοκιμαστική λύση.

Με βάση αυτό μορφοποιούμε την εξίσωση ως εξής:

$$x(t) = Ce^{\lambda t} \text{ (2)}$$

Με το σχηματισμό της πρώτης και δεύτερης παραγώγου της (2) αποκτούμε:

$$\dot{x}(t) = \lambda Ce^{\lambda t} \; \ddot{x}(t) = \lambda^2 Ce^{\lambda t} \text{ (3)}$$

Με την εισαγωγή των εξισώσεων (2) και (3) στη διαφορική εξίσωση (1) αποκτούμε:

$$\lambda^2 Ce^{\lambda t} + \frac{\mu}{m}\lambda Ce^{\lambda t} + \frac{k}{m}\lambda Ce^{\lambda t} = 0$$

Διαιρώντας με $Ce^{\lambda t}$ θα το απλοποιήσει σε:

$$\lambda^2 + \frac{\mu}{m}\lambda + \frac{k}{m} = 0 \text{ (4)}$$

Αυτή η εξίσωση ονομάζεται επίσης χαρακτηριστική εξίσωση. Είναι μια τετραγωνική εξίσωση σε κανονική μορφή. Επομένως υπάρχουν οι ακόλουθες δύο λύσεις για λ:

$$\lambda_{1,2}=-\frac{\mu}{2m}\pm\sqrt{\left(\frac{\mu}{2m}\right)^2-\frac{k}{m}} \text{ (5)}$$

Για περαιτέρω απλούστευση εισάγουμε τις νέες σταθερές δ και ω_0 :

$$\delta=\frac{\mu}{2m}, \quad \omega_0=\sqrt{\frac{k}{m}}$$

Η εξίσωση (5) τώρα μπορεί να γραφτεί ως:

$$\lambda_{1,2}=-\delta\pm\sqrt{\delta^2-\omega_0^2} \text{ (6)}$$

Η γενική λύση της ομογενούς διαφορικής εξίσωσης έχει τη μορφή:

$$x(t)=C_1x_1(t)+C_2x_2(t) \text{ (7)}$$

Οι συναρτήσεις $x_1(t)$ και $x_2(t)$ καθορίζονται από την τιμή της διακρίνουσας στην εξίσωση (6).

Υποκρίσιμη Απόσβεση

Για τη περίπτωση που εξετάζουμε εμείς (ταλάντωση με απόσβεση) ισχύει:

$$\delta<\omega_0$$

Σε αυτή την περίπτωση, η διακρίνουσα στην εξίσωση (6) είναι αρνητική. Επομένως το λ_1 και λ_2 είναι μιγαδικοί αριθμοί. Θα χρησιμοποιήσουμε την εκθετική υπόθεση $x(t)=Ce^{\lambda t}$ για την επίλυση της διαφορικής εξίσωσης.

$$x_1(t)=C_1e^{\lambda_1 t} \quad x_2(t)=C_2e^{\lambda_2 t}$$

Με την εισαγωγή της εκθετικής προσέγγισης στην εξίσωση (7) και αντικαθιστώντας το λ με την εξίσωση (6) αποκτούμε :

$$x(t)=e^{-\delta t}\left(C_1e^{\sqrt{\delta^2-\omega_0^2}t}+C_2e^{-\sqrt{\delta^2-\omega_0^2}t}\right) \text{ (8)}$$

Οι σταθερές C_1 και C_2 είναι μιγαδικές τιμές. Όταν αντιμετωπίζουμε μιγαδικούς αριθμούς χρησιμοποιούμε τον τύπο του Euler. Αυτή η εξίσωση δημιουργεί μια σύνδεση μεταξύ μιγαδικών εκθετικών συναρτήσεων και τριγωνομετρικών συναρτήσεων:

$$e^{i\varphi}=cos\varphi+isin\varphi$$

Είναι βοηθητικό να αναδιατάξουμε την εξίσωση (8) ώστε να απομονώσουμε το φανταστικό μέρος:

$$\sqrt{\delta^2-{\omega_0}^2}=\sqrt{-1*({\omega_0}^2-\delta)}=i\sqrt{{\omega_0}^2-\delta^2} \quad (9)$$

Για απλοποίηση ορίζουμε μια νέα σταθερά με το όνομα ω :

$$\omega=\sqrt{{\omega_0}^2-\delta^2} \quad (10)$$

Αυτή η σταθερά αντιπροσωπεύει τη φυσική συχνότητα του αρμονικού ταλαντωτή. Με τις σχέσεις απο τις εξισωσεις (9) και (10) η εξίσωση (8) μετατρέπεται σε :

$$x(t)=e^{-\delta t}\underbrace{(C_1e^{i\omega t}+C_2e^{-i\omega t})}_{\hat{x}(t)} \quad (11)$$

Από φυσικής άποψης, μόνο οι καθαρά πραγματικές λύσεις έχουν ενδιαφέρον. Για να τις βρούμε πρέπει να διαχωρίσουμε το πραγματικό και το φανταστικό μέρος. Όπως ήδη αναφέρθηκε το C_1 και C_2
είναι σταθερές μιγαδικής τιμής. Τώρα τα μεταμορφώνουμε στην πολική τους μορφή.

$$C_1=\hat{C}_1e^{i\varphi_1} \qquad C_2=\hat{C}_2e^{i\varphi_2} \quad (12)$$

Το πρώτο μέρος της εξίσωσης (11) είναι μια συνάρτηση εκθετικής απόσβεσης. Επειδή η σταθερά δ δεν είναι μιγαδική τιμή αυτό το μέρος της εξίσωσης δεν μπορεί να αποδώσει μιγαδικά αποτελέσματα. Εστιάζουμε λοιπόν στο δεύτερο μέρος που ονομάζουμε $\hat{x}(t)$. Αντικαθιστούμε τις σταθερές με αυτές που είναι γραμμένες σε πολική μορφή (12) και εφαρμόζουμε τον τύπο του Euler.

$$\hat{x}(t)=\hat{C}_1e^{i\varphi_1}e^{i\omega t}+\hat{C}_2e^{i\varphi_2}e^{-i\omega t}$$
$$\hat{x}(t)=\hat{C}_1e^{i(\varphi_1+\omega t)}+\hat{C}_2e^{i(\varphi_2-\omega t)}$$
$$\hat{x}(t)=\hat{C}_1\cos(\varphi_1+\omega t)+i\hat{C}_1\sin(\varphi_1+\omega t)+\hat{C}_2\cos(\varphi_2-\omega t)+i\hat{C}_2\sin(\varphi_2-\omega t)$$
$$\hat{x}(t)=\hat{C}_1\cos(\varphi_1+\omega t)+\hat{C}_2\cos(\varphi_2-\omega t)+i(\hat{C}_1\sin(\varphi_1+\omega t)+\hat{C}_2\sin(\varphi_2-\omega t)) \quad (13)$$

Στην εξίσωση (13) τα πραγματικά και τα φανταστικά μέρη της εξίσωσης διαχωρίζονται. Μας ενδιαφέρουν μόνο λύσεις για τις οποίες εξαφανίζεται το φανταστικό μέρος. Αυτό συμβαίνει όταν ισχύει το εξής:

$$\hat{C}_1\sin(\varphi_1+\omega t)+\hat{C}_2\sin(\varphi_2-\omega t)=0$$

$$\hat{C}_1 \sin(\varphi_1+\omega t)=-\hat{C}_2 \sin(\varphi_2-\omega t)$$
$$\hat{C}_1 \sin(\varphi_1+\omega t)=\hat{C}_2 \sin(-\varphi_2+\omega t) \quad (14)$$

Έτσι το φανταστικό μέρος της εξίσωσης (11) εξαφανίζεται εάν:

$$\hat{C}_1=\hat{C}_2, \varphi_1=-\varphi_2 \quad (15)$$

Με την εισαγωγή της εξίσωσης (15) στο υπόλοιπο πραγματικό μέρος της εξίσωσης (13) αποκτούμε το $\hat{x}(t)$:

$$\hat{x}(t)=\hat{C}_1 \cos(\varphi_1+\omega t)+\hat{C}_1 \cos(-\varphi_1-\omega t)$$
$$\hat{x}(t)=\hat{C}_1 \cos(\varphi_1+\omega t)+\hat{C}_1 \cos(\varphi_1+\omega t)$$
$$\hat{x}(t)=2\hat{C}_1 \cos(\varphi_1+\omega t) \quad (16)$$

Έτσι λαμβάνουμε την αναλυτική λύση της διαφορικής εξισώσεως για τη περίπτωση της υποκρίσιμης απόσβεσης :

$$x(t)=e^{-\delta t}(2A\cos(\varphi+\omega t)) \quad (17)$$

Όπου οι σταθερές φ_1 και $\hat{C}_1$ έχουν μετονομαστεί σε A και φ αντίστοιχα. Α είναι το πλάτος και φ η φάση της ταλάντωσης.

Οι σταθερές μπορούν να υπολογιστούν από τις αρχικές συνθήκες της ταλάντωσης :

$$x(0)=x_0, \dot{x}(0)=\upsilon_0 \quad (18)$$

Ο κώδικας που αναπτύχθηκε στηρίχτηκε στον τρόπο γραφής του προγράμματος που βρίσκεται στον σύνδεσμο https://github.com/okada39/pinn_burgers.

Ακολουθεί ο κώδικας Python που χρησιμοποιήθηκε:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.callbacks import Callback
from time import time


class PINN:

    def __init__(self, network, d, w0, m):
        self.network = network
```

```
        self.d = d
        self.w0 = w0
        self.m = m
        self.grads = GradientLayer(self.network)

    def build(self):
        # equation input
        tx_eqn = tf.keras.layers.Input(shape=(1,))
        # first condition input
        tx_1 = tf.keras.layers.Input(shape=(1,))
        # second condition input
        tx_2 = tf.keras.layers.Input(shape=(1,))

        # compute gradients
        x, dx_dt, d2x_dt2 = self.grads(tx_eqn)

        # equation output being zero

        u_eqn = self.m * d2x_dt2 + 2 * self.m * self.d * dx_dt + self.m * self.w0 ** 2
* x
        # condition 1 output
        u_1 = self.network(tx_1)
        # condition 2 output
        x, dx_dt, d2x_dt2 = self.grads(tx_2)
        u_2 = dx_dt


        # build the PINN model for equation
        return tf.keras.models.Model(
            inputs=[tx_eqn, tx_1, tx_2], outputs=[u_eqn, u_1, u_2])


class GradientLayer(tf.keras.layers.Layer):
    def __init__(self, model, **kwargs):
        self.model = model
        super().__init__(**kwargs)

    def call(self, t):
        with tf.GradientTape() as g:
            g.watch(t)
            with tf.GradientTape() as gg:
                gg.watch(t)
                x = self.model(t)
            dx_dt = gg.batch_jacobian(x, t)[..., 0]
        d2x_dt2 = g.batch_jacobian(dx_dt, t)[..., 0]
        return x, dx_dt, d2x_dt2


class Network:
```

```
    @classmethod
    def build(cls, num_inputs=1, layers=[15, 30, 15], activation='tanh',
num_outputs=1):
        # input layer
        inputs = tf.keras.layers.Input(shape=(num_inputs,))
        # hidden layers
        x = inputs
        initializer = tf.keras.initializers.glorot_normal
        for layer in layers:
            x = tf.keras.layers.Dense(layer, activation=activation,
                                      kernel_initializer=initializer)(x)
        # output layer
        outputs = tf.keras.layers.Dense(num_outputs, kernel_initializer=initializer)(x)

        return tf.keras.models.Model(inputs=inputs, outputs=outputs)


class stopAtLossValue(Callback):

    def on_batch_end(self, batch, logs={}):
        THR = 0.001
        if logs.get('loss') <= THR:
            self.model.stop_training = True


start = time()

# number of training samples
num_train_samples = 100
# number of test samples
num_test_samples = 100
# constants
d = 2
w0 = 10
m = 1
x_0 = 1
v_0 = 0

# build a core network model
network = Network.build()
network.summary()
# build a PINN model
pinn = PINN(network, d, w0, m).build()

# create training input
tx_eqn = np.linspace(0, 10, num_train_samples).reshape((num_train_samples, 1))

tx_1 = np.zeros((num_train_samples, 1))

tx_2 = np.zeros((num_train_samples, 1))
```

```
# create training output
u_eqn = np.zeros((num_train_samples, 1))  # u_eqn = 0
u_1 = x_0 * np.ones((num_train_samples, 1))  # u_1 = 1
u_2 = v_0 * np.ones((num_train_samples, 1))  # u_2 = 0


x_train = [tx_eqn, tx_1, tx_2]
y_train = [u_eqn, u_1, u_2]

callbacks = stopAtLossValue()
lr = 1e-3
pinn.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
             loss=tf.keras.losses.mse,
             metrics=[tf.keras.metrics.mse],)
model_history = pinn.fit(x_train, y_train, batch_size=16, epochs=10000,
                         callbacks=callbacks)
print(f"TOTAL TRAINING TIME = {round((time() - start), 2)} seconds")

t = np.linspace(0, 4, num_test_samples).reshape((num_test_samples, 1))

tx = np.concatenate([t], axis=-1)
u = network.predict(tx, batch_size=num_test_samples)

x = np.linspace(0, 4, num_test_samples)
w = np.sqrt(w0 ** 2 - d ** 2)
phi = np.arctan(-d / w)
A = 1 / (2 * np.cos([phi]))
cos = np.cos(phi + w * x)
sin = np.sin(phi + w * x)
exp = np.exp(-d * x)
u_exact = exp * 2 * A * cos

# plot theory vs prediction
plt.plot(x, u_exact, label='theory', color='crimson')
plt.plot(x, u, label='pinn', color='royalblue', linestyle='dotted')
plt.xlabel('t')
plt.ylabel('x(t)')
plt.legend()
plt.show()
```

Ας αναλύσουμε τον κώδικα ενότητα ανά ενότητα:

1. Εισαγωγή Βιβλιοθηκών:
   - `numpy` (ως np): Μια βιβλιοθήκη για αριθμητικούς υπολογισμούς.
   - `matplotlib.pyplot` (ως plt): Μια βιβλιοθήκη για απεικόνιση δεδομένων.
   - `tensorflow` (ως tf): Ένα πλαίσιο μηχανικής μάθησης ανοιχτού κώδικα.
   - `keras.callbacks.Callback`: Μια κλάση από τη βιβλιοθήκη Keras για τη δημιουργία προσαρμοσμένων συναρτήσεων επανάκλησης.
   - `time`: Μια ενότητα για τη μέτρηση του χρόνου.

2. Ορισμοί κλάσης:
   - `PINN`: Μια κλάση που αντιπροσωπεύει το νευρωνικό δίκτυο με πληροφόρηση από τη φυσική. Διαθέτει μεθόδους για την προετοιμασία του PINN και τη δημιουργία του μοντέλου.
   - `GradientLayer`: Μια προσαρμοσμένη κλάση επιπέδου που υπολογίζει τις παραγώγους του μοντέλου σε σχέση με την είσοδο.

3. Network Class:
   - `build`: Μια μέθοδος κλάσης της κλάσης `Network` που δημιουργεί ένα μοντέλο νευρωνικού δικτύου με προσαρμόσιμη αρχιτεκτονική. Χρησιμοποιεί το λειτουργικό API Keras για να ορίσει τα επίπεδα και τις συνδέσεις τους.

4. Callback Class:
   - `stopAtLossValue`: Μια προσαρμοσμένη κλάση επανάκλησης που σταματά την εκπαίδευση όταν η τιμή απώλειας πέσει κάτω από ένα όριο (`THR`).

5. Αρχικοποίηση και διαμόρφωση:
   - Ρύθμιση των αρχικών τιμών για διάφορες παραμέτρους, όπως ο αριθμός των δειγμάτων εκπαίδευσης και δοκιμής, οι σταθερές (`d`, `w0`, `m`) και οι αρχικές συνθήκες (`x_0`, `v_0`).

6. Κατασκευή μοντέλου:
   - Δημιουργία μίας υπόστασης (instance) της κλάσης «Network» και κατασκευή του μοντέλου του βασικού δικτύου.
   - Εκτύπωση της αναφοράς του μοντέλου δικτύου.
   - Δημιουργία instance της κλάσης «PINN», διαβίβαση του μοντέλου δικτύου, των σταθερών και του επιπέδου κλίσης.
   - Δημιουργία του μοντέλου PINN καλώντας τη μέθοδο «build» της κλάσης «PINN».

7. Προετοιμασία Δεδομένων Εκπαίδευσης:
   - Δημιουργία εισόδου εκπαίδευσης (`tx_eqn`, `tx_1`, `tx_2`) χρησιμοποιώντας το `np.linspace` για τη δημιουργία τιμών σε ίσες αποστάσεις εντός ενός δεδομένου εύρους.
   - Εκκίνηση της εξόδου εκπαίδευσης (`u_eqn`, `u_1`, `u_2`) με κατάλληλες τιμές.

8. Σύνταξη και εκπαίδευση μοντέλου:
   - Σύνταξη του μοντέλου PINN με βελτιστοποιητή (Adam) και συνάρτηση απώλειας (μέσο τετραγωνικό σφάλμα).
   - Ρύθμιση της συνάρτησης επανάκλησης `stopAtLossValue` για διακοπή της προπόνησης όταν η απώλεια πέσει κάτω από ένα όριο.
   - Προσαρμογή του μοντέλου PINN στα δεδομένα εκπαίδευσης (`x_train`, `y_train`) για καθορισμένο αριθμό εποχών και μέγεθος παρτίδας.

9. Αξιολόγηση και Απεικόνιση:
   - Δημιουργία εισόδου δοκιμής (`t`) χρησιμοποιώντας το "np.linspace".
   - Συνένωση της εισόδου δοκιμής στο `tx` για πρόβλεψη.
   - Πρόβλεψη της εξόδου (`u`) χρησιμοποιώντας το μοντέλο δικτύου.
   - Υπολογισμός της ακριβούς λύσης (`u_exact`) με βάση τη δεδομένη εξίσωση και τις αρχικές συνθήκες.
   - Σχεδιάσμος της ακριβούς λύσης (`u_exact`) και της λύσης που έχει προβλεφθεί (`u`) χρησιμοποιώντας το "matplotlib.pyplot".

Το παρεχόμενο νευρωνικό δίκτυο με πληροφόρηση φυσικής (PINN) λειτουργεί ως εξής:

- Αρχικοποίηση:
  - Το PINN προετοιμάζεται με διάφορες παραμέτρους: «network» (αρχιτεκτονική νευρωνικού δικτύου), «d» (σταθερά απόσβεσης), «w0» (φυσική συχνότητα) και «m» (μάζα).

- Model Building (μέθοδος «build»):
  - Το μοντέλο PINN κατασκευάζεται χρησιμοποιώντας Keras. Καθορίζει τις εισόδους και τις εξόδους του μοντέλου PINN.
  - Ορίζονται τρεις είσοδοι:
    - `tx_eqn`: Αντιπροσωπεύει το χρόνο (t) για τη διαφορική εξίσωση.
    - `tx_1`: Αντιπροσωπεύει το χρόνο για την πρώτη αρχική συνθήκη.
    - `tx_2`: Αντιπροσωπεύει το χρόνο για τη δεύτερη αρχική συνθήκη.
  - Το μοντέλο υπολογίζει τις παραγώγους (πρώτη και δεύτερη) της λύσης σε σχέση με το χρόνο (`t`) χρησιμοποιώντας το "GradientLayer".

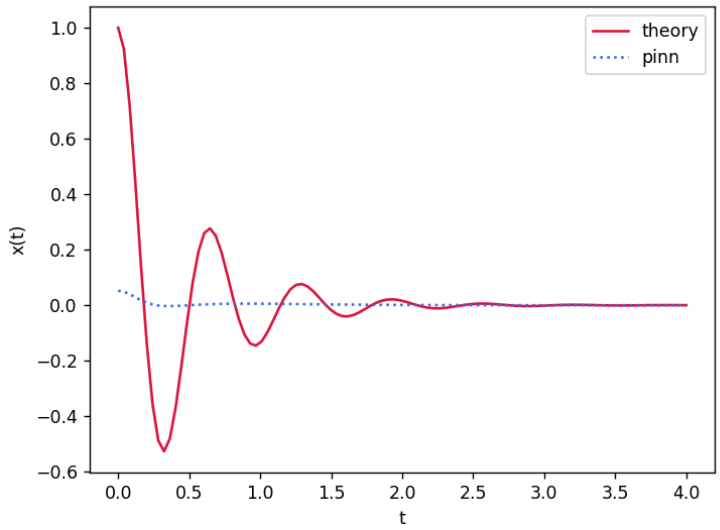
- Υπολογισμός παραγώγων (κατηγορία `GradientLayer`):
  - Το `GradientLayer` είναι ένα προσαρμοσμένο επίπεδο Keras που χρησιμοποιείται για τον υπολογισμό των παραγώγων των προβλέψεων του μοντέλου σε σχέση με το χρόνο (`t`). Αξιοποιεί τον μηχανισμό «GradientTape» του TensorFlow για να υπολογίσει αυτές τις παραγώγους. Υπολογίζει την πρώτη παράγωγο «dx_dt» και τη δεύτερη παράγωγο «d2x_dt2» της λύσης με βάση την έξοδο του μοντέλου.

- Ορισμός εξισώσεων και οριακών συνθηκών:
  - Το PINN ορίζει τρεις εξόδους:
    - `u_eqn`: Αντιπροσωπεύει την έξοδο της εξίσωσης, η οποία έχει οριστεί να ικανοποιεί τη συνήθη διαφορική εξίσωση δεύτερης τάξης.
    - `u_1`: Αντιπροσωπεύει την έξοδο της πρώτης αρχικής συνθήκης.
    - `u_2`: Αντιπροσωπεύει την έξοδο της δεύτερης αρχικής συνθήκης.
  - Το "u_eqn" ορίζεται με βάση τη ΣΔΕ με όρους που σχετίζονται με τα "d", "w0" και "m". Εξασφαλίζει ότι η ΣΔΕ είναι ικανοποιημένη. Τα «u_1» και «u_2» λαμβάνονται αξιολογώντας το νευρωνικό δίκτυο σε «tx_1» και «tx_2», αντίστοιχα.

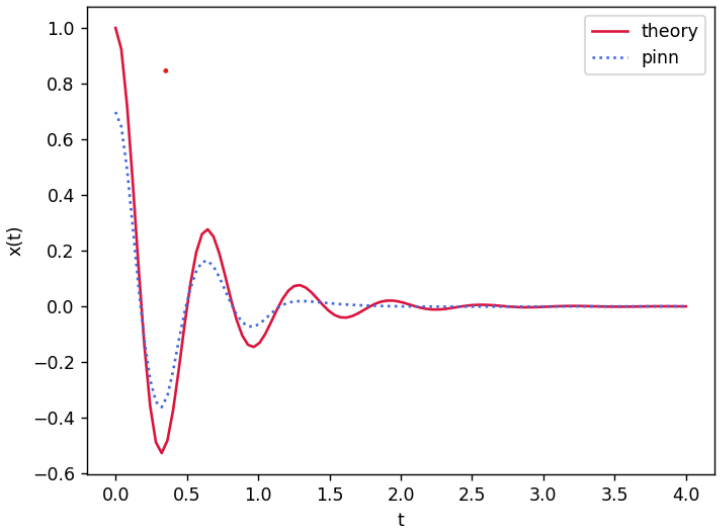
- Σύνταξη μοντέλου και εκπαίδευση:
  - Το μοντέλο PINN συντάσσεται με βελτιστοποιητή Adam, απώλεια μέσου τετραγωνικού σφάλματος (MSE) και ένα MSE metric. Εκπαιδεύεται χρησιμοποιώντας τα δεδομένα εκπαίδευσης («x_train» και «y_train») που δημιουργούνται με βάση τη ΣΔΕ και τις αρχικές συνθήκες. Η διαδικασία εκπαίδευσης στοχεύει στην ελαχιστοποίηση της ασυμφωνίας μεταξύ των προβλέψεων του PINN και των τιμών-στόχων που ορίζονται από τη ΣΔΕ και τις οριακές συνθήκες. Η εκπαίδευση χρησιμοποιεί μια προσαρμοσμένη επανάκληση `stopAtLossValue` για να σταματήσει όταν η απώλεια πέσει κάτω από ένα συγκεκριμένο όριο.

- Δοκιμή και αξιολόγηση:
  - Μετά την εκπαίδευση, το μοντέλο PINN χρησιμοποιείται για την πρόβλεψη της λύσης για ένα σύνολο χρονικών σημείων δοκιμής (`t`). Επιπλέον, η ακριβής λύση (`u_exact`) υπολογίζεται για σύγκριση. Στη συνέχεια, ο κώδικας σχεδιάζει την λύση πρόβλεψης (`u`) και την ακριβή λύση (`u_exact`) χρησιμοποιώντας το Matplotlib.

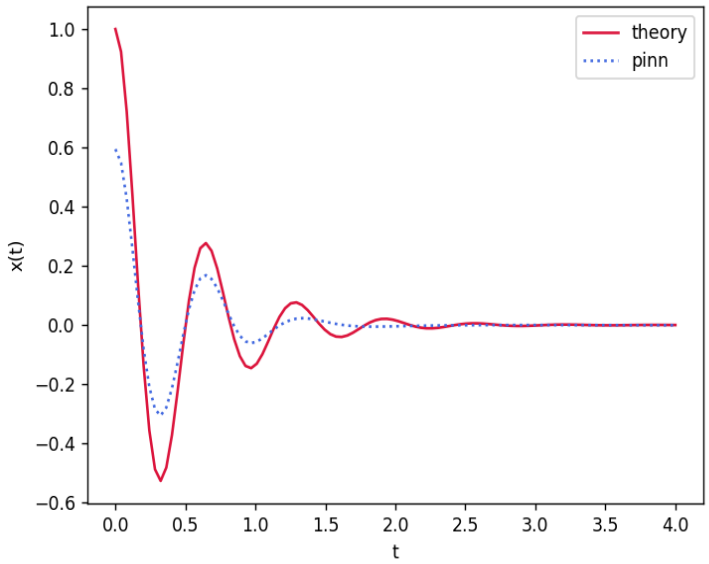
Η συμπεριφορά του συστήματος εξαρτάται από πολλές διαφορετικές παραμέτρους του προγράμματος όπως ο αριθμός των hidden layers και νευρώνων, η συνάρτηση ενεργοποίησης, το batch size, οι κύκλοι εκπαίδευσης (epochs) και το μέγεθος του δείγματος για εκπαίδευση. Για να μελετήσουμε τη συμπεριφορά του συστήματος επιλέξαμε να μεταβάλλουμε μόνο τον αριθμό των σημειών για εκπαίδευση (training sample) κρατώντας όλες τις υπόλοιπες παραμέτρους σταθερές. Σε όλες τις εκτελέσεις του προγράμματος που παρατίθονται παρακάτω χρησιμοποιήθηκε ως συνάρτηση ενεργοποίησης η tanh, η δομή του δικτύου ήταν η [15, 30, 15], το batch size ορίστηκε ίσο με 16 και οι κύκλοι εκπαίδευσης ήταν ίσοι με 10000. Επίσης αξίζει να σημειωθεί πως τα σημεία που επιλέχθηκαν για εκπαίδευση άνηκαν στο διάστημα [0,10] και είχαν ίση απόσταση μεταξύ τους.



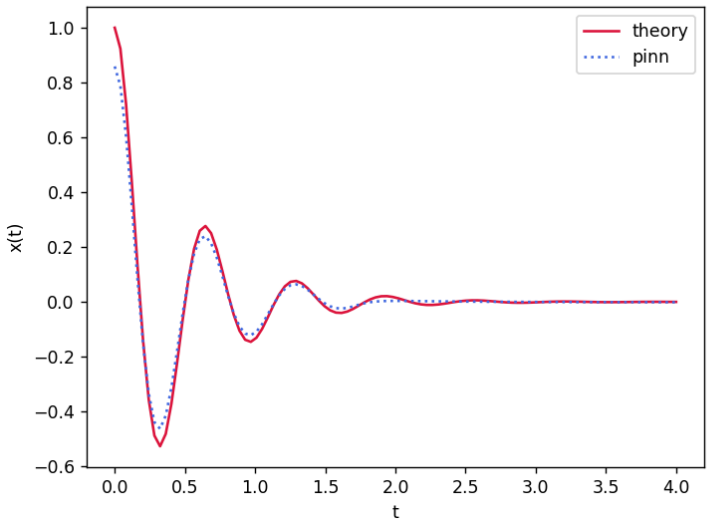
Εικόνα 7: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 50).



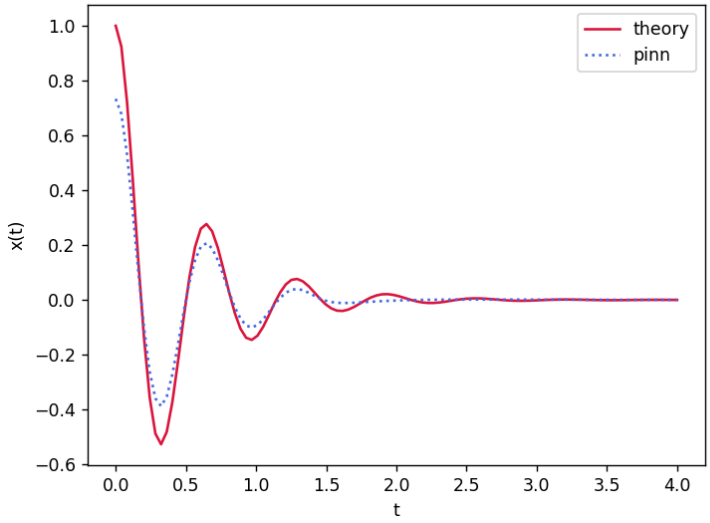
Εικόνα 8:Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 100).



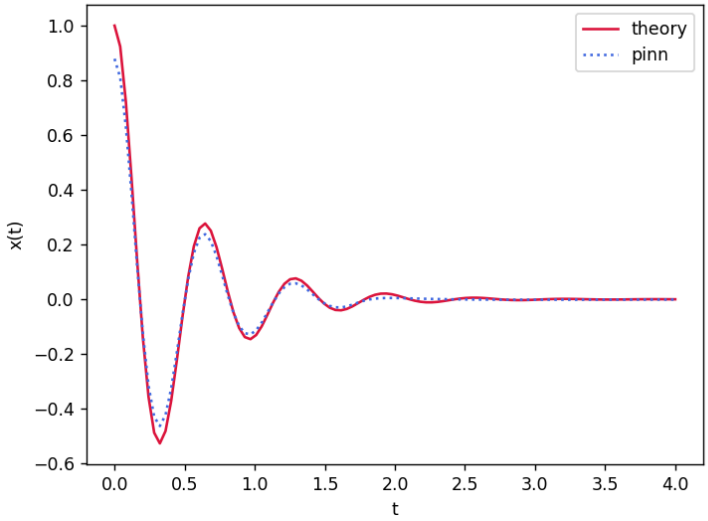
Εικόνα 9: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 150).



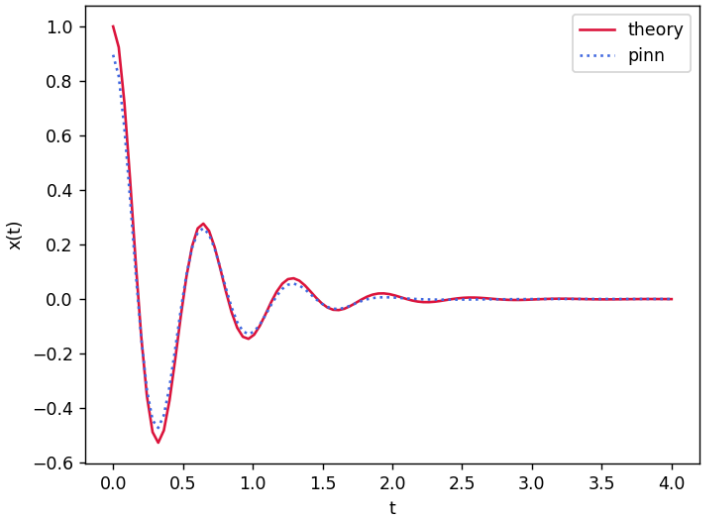
Εικόνα 10: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 200).



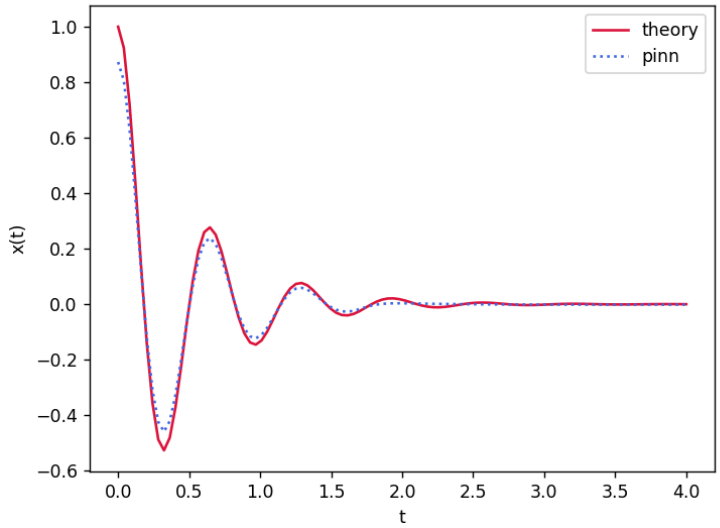
Εικόνα 11: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 250)



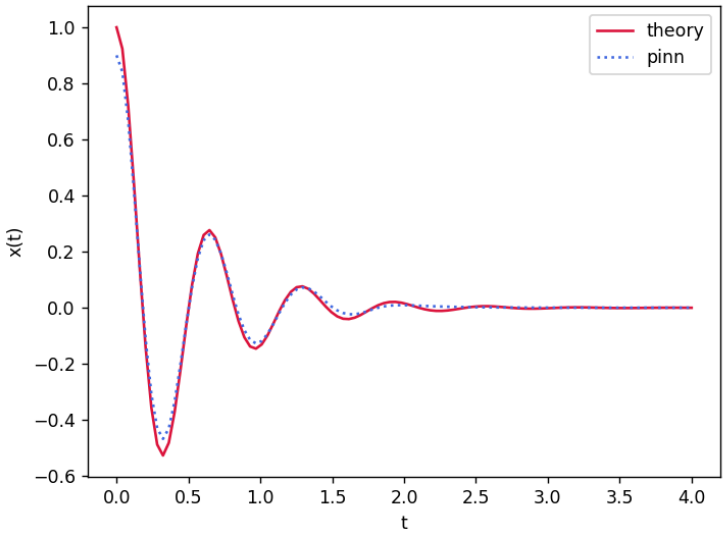
Εικόνα 12: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 300).



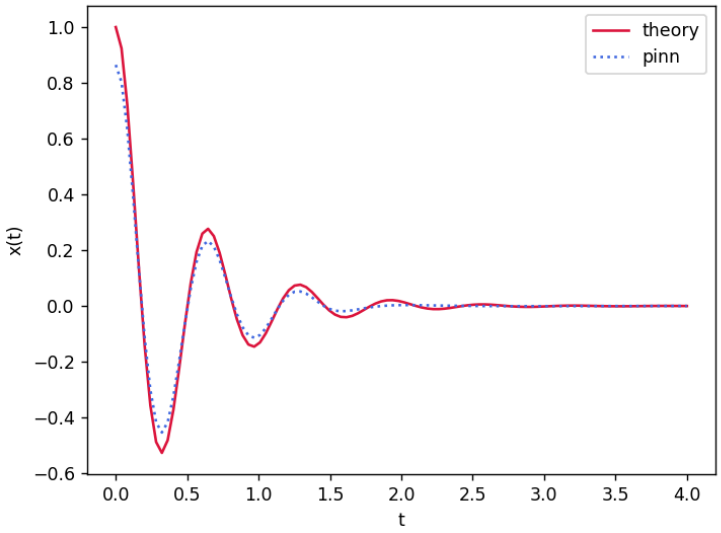
Εικόνα 13: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 350).



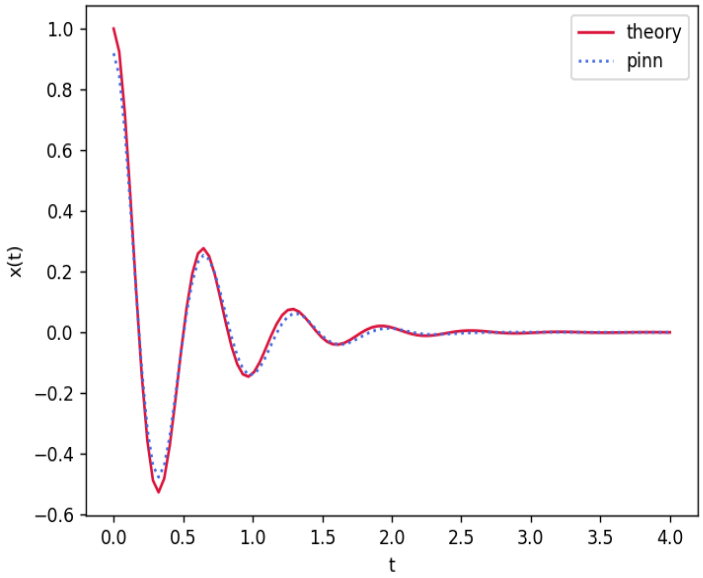
Εικόνα 14: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 400).



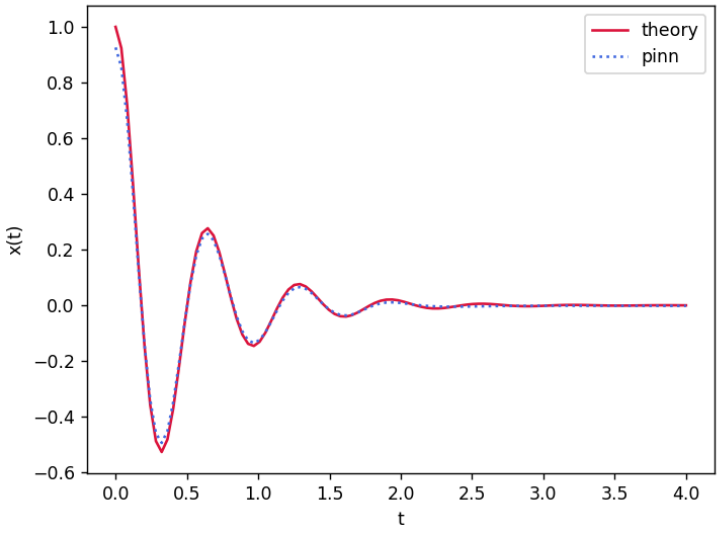
Εικόνα 15: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 450).



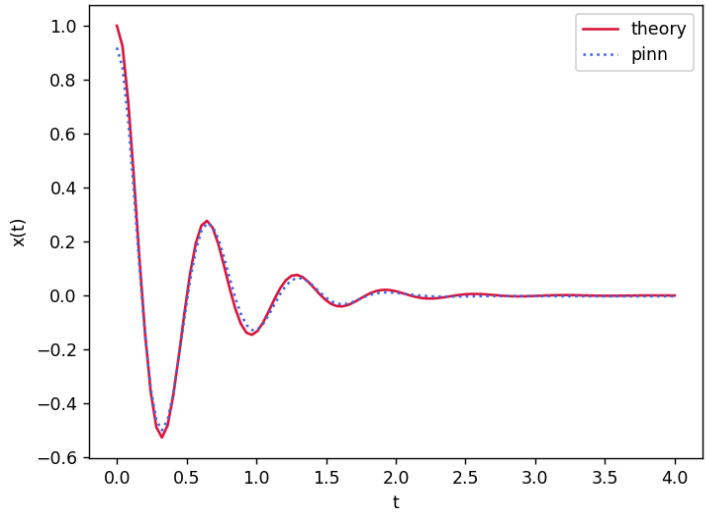
Εικόνα 16: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 500).



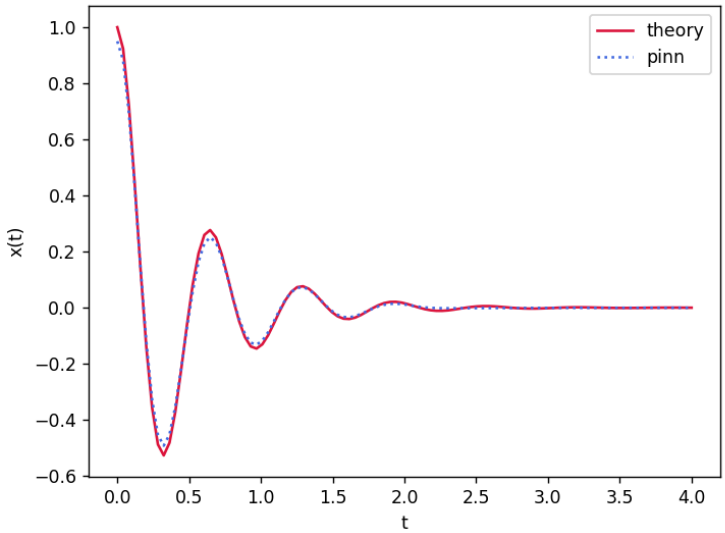
Εικόνα 17: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 600).



Εικόνα 18: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 700).



Εικόνα 19: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 1000).



Εικόνα 20: Γραφική αναπαράσταση της αναλυτικής λύσης και της πρόβλεψης του μοντέλου (training sample = 1200).

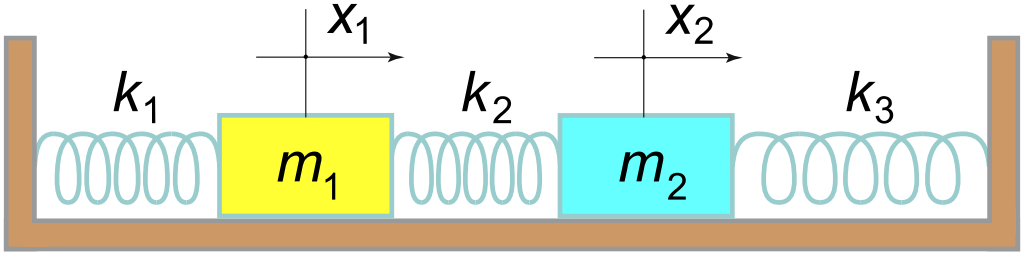
Πίνακας 4.1 (Πίνακας αποτελεσμάτων από την επίλυση του προβλήματος 1)

Εκτέλεση	**Layers**	**Batch size**	**Epochs**	**Training Sample**	**Loss mse**	**Time (sec)**
1	[15, 30, 15]	16	10000	50	0.9573	142.47
2	[15, 30, 15]	16	10000	100	0.2493	212.14
3	[15, 30, 15]	16	10000	150	0.2970	278.75
4	[15, 30, 15]	16	10000	200	0.2818	339.55
5	[15, 30, 15]	16	10000	250	0.1705	405.64
6	[15, 30, 15]	16	10000	300	0.0676	468.52
7	[15, 30, 15]	16	10000	350	0.1802	521.23
8	[15, 30, 15]	16	10000	400	0.0585	461.48
9	[15, 30, 15]	16	10000	450	0.5034	682.82
10	[15, 30, 15]	16	10000	500	0.3673	749.43
11	[15, 30, 15]	16	10000	550	0.1102	824.36
12	[15, 30, 15]	16	10000	600	0.5849	859.15
13	[15, 30, 15]	16	10000	650	0.1645	945.43
14	[15, 30, 15]	16	10000	700	0.0614	1040.97
15	[15, 30, 15]	16	10000	750	0.1134	1077.17
16	[15, 30, 15]	16	10000	800	0.0698	928.27
17	[15, 30, 15]	16	10000	1000	0.1987	1420.46
18	[15, 30, 15]	16	10000	1200	0.1045	1250.13

4.2 Πρόβλημα 2 : Πρόβλεψη θέσης στο χρόνο 2 μαζών συνδεδεμένων με 3 ελατήρια

Ένα σύστημα που αποτελείται από δύο μάζες και τρία ελατήρια έχει δύο βαθμούς ελευθερίας [28]. Αυτό σημαίνει ότι η διαμόρφωσή του μπορεί να περιγραφεί από δύο γενικευμένες συντεταγμένες, οι οποίες μπορούν να επιλεγούν ως οι μετατοπίσεις της πρώτης και της δεύτερης μάζας από τη θέση ισορροπίας.

Η κίνηση των συνδεδεμένων μαζών περιγράφεται με δύο διαφορικές εξισώσεις δεύτερης τάξης. Στην απλούστερη περίπτωση μπορούμε να αγνοήσουμε τις δυνάμεις της τριβής και της αντίστασης του αέρα και να εξετάσουμε μόνο την ελαστική δύναμη που υπακούει στο νόμο του Hooke. Αποδεικνύεται ότι ακόμη και ένα τόσο απλοποιημένο σύστημα έχει μη τετριμμένες δυναμικές ιδιότητες. Γενικά, ο χαρακτήρας της κίνησης καθορίζεται από δύο ιδιοσυχνότητες που εξαρτώνται από τις παραμέτρους του συστήματος (δηλαδή τη μάζα των σωμάτων και τις σταθερές του ελατηρίου). Επιπλέον, η κίνηση των μαζών εξαρτάται σε μεγάλο βαθμό από τις αρχικές συνθήκες.



Εικόνα 21 (Σύστημα που αποτελείται από 2 μάζες και 3 ελατήρια. Πηγή: https://mathlake.com/mass-spring-system)

Το σύστημα αποτελείται από δύο μάζες m_1 και m_2 και τρία ελατήρια με συντελεστές ακαμψίας k_1 , k_2 , k_3. Η μετατόπιση των μαζών από τις θέσεις ισορροπίας τους καθορίζεται από τις συντεταγμένες
x_1 και x_2.
Ας υπολογίσουμε τις εξισώσεις κίνησης.

Γράφουμε τις εκφράσεις για την κινητική και τη δυναμική ενέργεια. Να σημειωθεί ότι η συνολική ενέργεια σε αυτό το ιδανικό σύστημα διατηρείται.

$$T=\frac{m_1\dot{x}_1^2}{2}+\frac{m_2\dot{x}_2^2}{2},\quad V=\frac{k_1x_1^2}{2}+\frac{k_2(x_2-x_1)^2}{2}+\frac{k_3x_2^2}{2}.$$

Η Λαγκρανζιανή (L)του συστήματος γράφεται ως εξής:

$$L = T - V = \frac{1}{2}\left[m_1 \dot{x_1}^2 + m_2 \dot{x_2}^2 - k_1 x_1^2 - k_2 (x_2 - x_1)^2 - k_3 x_2^2\right].$$

Συνθέτουμε τις διαφορικές εξισώσεις Lagrange:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{x}_i} = \frac{\partial L}{\partial x_i} \quad \text{ή} \quad \frac{d}{dt}\frac{\partial L}{\partial \dot{x}_1} = \frac{\partial L}{\partial x_1} \ , \ \frac{d}{dt}\frac{\partial L}{\partial \dot{x}_2} = \frac{\partial L}{\partial x_2}.$$

Βρίσκουμε τις μερικές παραγώγους:

$$\frac{\partial L}{\partial \dot{x}_1} = \frac{1}{2}\cdot 2 m_1 \dot{x}_1 = m_1 \dot{x}_1 \ , \ \frac{\partial L}{\partial x_1} = \frac{1}{2}\left[-2 k_1 x_1 + 2 k_2 (x_2 - x_1)\right] = -k_1 x_1 + k_2 (x_2 - x_1),$$

$$\frac{\partial L}{\partial \dot{x}_2} = \frac{1}{2}\cdot 2 m_2 \dot{x}_2 = m_2 \dot{x}_2, \frac{\partial L}{\partial x_2} = \frac{1}{2}\left[-2 k_2 (x_2 - x_1) - 2 k_2 x_2\right] = -k_2 (x_2 - x_1) - k_3 x_2.$$

Ως αποτέλεσμα, λαμβάνουμε το ακόλουθο σύστημα εξισώσεων που περιγράφουν την κίνηση των μαζών:

$$m_1 \ddot{x}_1 = -k_1 x_1 + k_2 (x_2 - x_1) \Rightarrow \ddot{x}_1 = -\frac{k_1}{m_1} x_1 + \frac{k_2}{m_1} x_2 - \frac{k_2}{m_1} x_1 \Rightarrow \ddot{x}_1 = -\frac{k_1 + k_2}{m_1} x_1 + \frac{k_2}{m_1} x_2$$

$$m_2 \ddot{x}_2 = -k_2 (x_2 - x_1) - k_3 x_2 \Rightarrow \ddot{x}_2 = -\frac{k_2}{m_2} x_2 + \frac{k_2}{m_2} x_1 - \frac{k_3}{m_2} x_2 \Rightarrow \ddot{x}_2 = \frac{k_2}{m_2} x_1 - \frac{k_2 + k_3}{m_2} x_2$$

Αυτό το σύστημα μπορεί να γραφτεί σε μορφή πίνακα:

$$\ddot{\boldsymbol{X}} = K\,\boldsymbol{X} \ \text{όπου} \ \boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \ \ddot{\boldsymbol{X}} = \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix}, \ K = \begin{bmatrix} -\frac{k_1 + k_2}{m_1} & \frac{k_2}{m_1} \\ \frac{k_2}{m_2} & -\frac{k_2 + k_3}{m_2} \end{bmatrix}$$

Θα αναζητήσουμε τη λύση $\boldsymbol{X}(t)$ με τη μορφή αρμονικών ταλαντώσεων, δηλαδή όπως

$$\boldsymbol{X}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} e^{i\omega t},$$

Όπου B_1, B_2 είναι τα πλάτη των ταλαντώσεων των μαζών, ω είναι οι ιδιοσυχνότητες που πρέπει να προσδιοριστούν.

Αντικαθιστώντας τις δοκιμαστικές συναρτήσεις $x_1(t), x_2(t)$ στην εξίσωση του πίνακα, λαμβάνουμε τη βοηθητική εξίσωση που καθορίζει τις ιδιοσυχνότητες:

$$det\left(K+\omega^2 I\right)=0\Rightarrow\begin{vmatrix}-\frac{k_1+k_2}{m_1}+\omega^2 & \frac{k_2}{m_1}\\ \frac{k_2}{m_2} & -\frac{k_2+k_3}{m_2}+\omega^2\end{vmatrix}=0\Rightarrow\left(\omega^2-\frac{k_1+k_2}{m_1}\right)\left(\omega^2-\frac{k_2+k_3}{m_2}\right)-\frac{k_2^2}{m_1 m_2}=0$$

$$\Rightarrow\omega^4-\frac{k_1+k_2}{m_1}\omega^2-\frac{k_2+k_3}{m_2}\omega^2+\frac{(k_1+k_2)(k_2+k_3)}{m_1 m_2}-\frac{k_2^2}{m_1 m_2}=0\Rightarrow$$
$$\omega^4-\left(\frac{k_1+k_2}{m_1}+\frac{k_2+k_3}{m_2}\right)\omega^2+\frac{(k_1+k_2)(k_2+k_3)}{m_1 m_2}-\frac{k_2^2}{m_1 m_2}=0$$

Λύνοντας αυτή την εξίσωση, βρίσκουμε τις ιδιοσυχνότητες. Ας υπολογίσουμε πρώτα τη διακρίνουσα:

$$D=\left(\frac{k_1+k_2}{m_1}+\frac{k_2+k_3}{m_2}\right)^2-4\left[\frac{(k_1+k_2)(k_2+k_3)}{m_1 m_2}-\frac{k_2^2}{m_1 m_2}\right]$$
$$\Rightarrow D=\left(\frac{k_1+k_2}{m_1}\right)^2+\left(\frac{k_2+k_3}{m_2}\right)^2+\frac{2(k_1+k_2)(k_2+k_3)}{m_1 m_2}-\frac{4(k_1+k_2)(k_2+k_3)}{m_1 m_2}+\frac{4k_2^2}{m_1 m_2}$$
$$\Rightarrow D=\left(\frac{k_1+k_2}{m_1}-\frac{k_2+k_3}{m_2}\right)^2+\frac{4k_2^2}{m_1 m_2}$$

Στη συνέχεια, το τετράγωνο των ιδιοσυχνοτήτων θα περιγραφεί από τον τύπο

$$\omega^2=\frac{1}{2}\left\{\left(\frac{k_1+k_2}{m_1}+\frac{k_2+k_3}{m_2}\right)\pm\left[\left(\frac{k_1+k_2}{m_1}-\frac{k_2+k_3}{m_2}\right)^2+\frac{4k_2^2}{m_1 m_2}\right]^{\frac{1}{2}}\right\}$$

Στη συνέχεια, για να αποφύγουμε τους πολύπλοκους τύπους, εξετάζουμε την απλούστερη περίπτωση όπου η ακαμψία όλων των ελατηρίων είναι η ίδια: $k_1=k_2=k_3=k$.

Επιπλέον, εισάγουμε την αναλογία μάζας: $\mu=\frac{m_2}{m_1}$.Τότε ο τύπος για το τετράγωνο των συχνοτήτων των ταλαντώσεων παίρνει τη μορφή:

$$\omega^2=\frac{1}{2}\left[\left(\frac{2k}{m_1}+\frac{2k}{m_2}\right)\pm\sqrt{\left(\frac{2k}{m_1}-\frac{2k}{m_2}\right)^2+\frac{4k^2}{m_1 m_2}}\right]=k\left[\left(\frac{1}{m_1}+\frac{1}{m_2}\right)\pm\sqrt{\left(\frac{1}{m_1}-\frac{1}{m_2}\right)^2+\frac{1}{m_1 m_2}}\right]=\frac{k}{m_2}\left[\mu+1\pm\sqrt{(\mu-1)^2+\mu}\right]$$

Στην περίπτωση ίσων μαζών $(\mu=1)$, οι ιδιοσυχνότητες περιγράφονται από τους ακόλουθους τύπους:

$$\omega_1=\sqrt{\frac{3k}{m}}\ ,\ \omega_2=\sqrt{\frac{k}{m}}$$

Να σημειωθεί ότι οι συχνότητες ω_1,ω_2 είναι πάντα πραγματικοί αριθμοί. Αυτό προκύπτει από γενικές φυσικές εκτιμήσεις. Πράγματι, στην περίπτωση της φανταστικής συχνότητας, θα υπήρχε διαρροή ενέργειας, η οποία έρχεται σε αντίθεση με την προϋπόθεση διατήρησης της ενέργειας στο σύστημα. Το γεγονός αυτό, όμως, μπορεί να αποδειχθεί καθαρά μαθηματικά. Στην πραγματικότητα, το ερώτημα τίθεται μόνο για τη συχνότητα ω_2.Η συνθήκη μη αρνητικότητας για ω_2^2 δίνεται από

$$\omega_2^2>0,\ \mu+1-\sqrt{(\mu-1)^2+\mu}>0,\ \Rightarrow\mu+1>\sqrt{(\mu-1)^2+\mu}$$

Η αριστερή πλευρά της ανισότητας και η έκφραση κάτω από την τετραγωνική ρίζα στη δεξιά πλευρά είναι πάντα θετικές. Αφού τετραγωνίσουμε και τις δύο πλευρές, παίρνουμε

$$(\mu+1)^2>(\mu-1)^2+\mu,\Rightarrow\mu^2+2\mu+1>\mu^2-2\mu+1+\mu,\Rightarrow3\mu>0,\Rightarrow\mu>0$$

το οποίο ισχύει πάντα.
Τώρα βρίσκουμε το ιδιοδιάνυσμα $H_1=(H_{11},H_{21})^T$ που αντιστοιχεί στη συχνότητα ω_1.
Αυτό καθορίζεται από την εξίσωση

$$(K+\omega_1^2 I)\boldsymbol{H_1}=\mathbf{0}$$

Συνεπώς

$$\begin{Bmatrix}\left(-\frac{2}{m_1}+\frac{1}{m_2}\left[\mu+1+\sqrt{(\mu-1)^2+\mu}\right]\right)H_{11}+\frac{1}{m_1}H_{21}=0\\ \frac{1}{m_2}H_{11}+\left(-\frac{2}{m_1}+\frac{1}{m_2}\left[\mu+1+\sqrt{(\mu-1)^2+\mu}\right]\right)H_{21}=0\end{Bmatrix}$$

Στο τελευταίο σύστημα, οι δύο εξισώσεις εξαρτώνται γραμμικά (καθώς η διακρίνουσα του K είναι μηδέν για $\omega^2=\omega_1^2$). Επομένως, οι συντεταγμένες του ιδιοδιανύσματος H_1 μπορεί να εκφραστεί, για παράδειγμα, από την πρώτη εξίσωση. Για $H_{11}=1$. Τότε

$$H_{21}=m_1\left(\frac{2}{m_1}-\frac{1}{m_2}\left[\mu+1+\sqrt{(\mu-1)^2+\mu}\right]\right)\Rightarrow$$

$$\Rightarrow 2-\frac{1}{\mu}\left[\mu+1+\sqrt{(\mu-1)^2+\mu}\right]=1-\frac{1+\sqrt{(\mu-1)^2+\mu}}{\mu}=\frac{\mu-1-\sqrt{(\mu-1)^2+\mu}}{\mu}$$

Έτσι, το διάνυσμα H_1 έχει τις ακόλουθες συντεταγμένες:

$$\boldsymbol{H_1}=\begin{bmatrix}H_{11}\\H_{21}\end{bmatrix}=\begin{bmatrix}1\\\frac{\mu-1-\sqrt{(\mu-1)^2+\mu}}{\mu}\end{bmatrix}$$

Ομοίως, μπορούμε να προσδιορίσουμε το ιδιοδιάνυσμα $H_2=(H_{12},H_{22})^T$ που αντιστοιχεί στη συχνότητα ω_2. Σε αυτή την περίπτωση, έχουμε την ακόλουθη εξίσωση για H_2 :

$$(K+\omega_2^2 I)\boldsymbol{H_2}=\boldsymbol{0}$$

Στη ανεπτυγμένη μορφή γράφεται ως

$$\begin{Bmatrix}\left(-\frac{2}{m_1}+\frac{1}{m_2}\left[\mu+1-\sqrt{(\mu-1)^2+\mu}\right]\right)H_{12}+\frac{1}{m_1}H_{22}=0\\\frac{1}{m_2}H_{12}+\left(-\frac{2}{m_1}+\frac{1}{m_2}\left[\mu+1-\sqrt{(\mu-1)^2+\mu}\right]\right)H_{22}=0\end{Bmatrix}$$

Θεωρώντας $H_{12}=1$, βρίσκουμε τις συντεταγμένες H_{22} από τη πρώτη εξίσωση :

$$H_{22}=m_1\left(\frac{2}{m_1}-\frac{1}{m_2}\left[\mu+1-\sqrt{(\mu-1)^2+\mu}\right]\right)=2-\frac{1}{\mu}\left[\mu+1-\sqrt{(\mu-1)^2+\mu}\right]=\frac{\mu-1+\sqrt{(\mu-1)^2+\mu}}{\mu}$$

Ως εκ τούτου,

$$\boldsymbol{H_2}=\begin{bmatrix}H_{12}\\H_{22}\end{bmatrix}=\begin{bmatrix}1\\\frac{\mu-1+\sqrt{(\mu-1)^2+\mu}}{\mu}\end{bmatrix}$$

Μόλις βρούμε τις ιδιοσυχνότητες ω_1,ω_2 και τα ιδιοδιανύσματα H_1,H_2, μπορούμε να γράψουμε τη γενική λύση του συστήματος. Λάβετε υπόψη ότι καθένα από τα ιδιοδιανύσματα αντιστοιχεί στο τετράγωνο της ιδιοσυχνότητας, δηλαδή δύο τιμές συχνότητας με αντίθετα πρόσημα. Το διάνυσμα H_1
συνδέεται με δύο συχνότητες $\pm\omega_1$ και το διάνυσμα H_2 σχετίζεται με τις συχνότητες $\pm\omega_2$. Ως αποτέλεσμα, η γενική μιγαδική λύση παριστάνεται ως το άθροισμα τεσσάρων όρων:

$$\boldsymbol{X}(t)=\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}=C_1 e^{i\omega_1 t}\boldsymbol{H_1}+C_2 e^{-i\omega_1 t}\boldsymbol{H_1}+C_3 e^{i\omega_2 t}\boldsymbol{H_2}+C_4 e^{-i\omega_2 t}\boldsymbol{H_2}$$

όπου $C_1,\ldots,C_4$ είναι σταθεροί (σε αυτή την περίπτωση μιγαδικοί) αριθμοί και γράφονται ως εξής:

$C_1=\alpha_1+i\beta_1,\ C_2=\alpha_1+i\beta_2,\ C_3=\alpha_3+i\beta_3,\ C_4=\alpha_4+i\beta_4.$

Για να διατηρήσουμε τις αξίες των $x_1(t), x_2(t)$ πραγματικές σε οποιαδήποτε t είναι απαραίτητο να ικανοποιηθούν οι ακόλουθες σχέσεις:

$C_1=\bar{C}_2,\ \Rightarrow \alpha_1+i\beta_1=\alpha_2-i\beta_2,\ \Rightarrow\ \alpha_1=\alpha_2\ \text{και}\ \beta_1=-\beta_2$

$C_3=\bar{C}_4,\ \Rightarrow \alpha_3+i\beta_3=\alpha_4-i\beta_4,\ \Rightarrow\ \alpha_3=\alpha_4\ \text{και}\ \beta_4=-\beta_4$

Τότε οι φανταστικοί όροι της γενικής λύσης θα ακυρωθούν.

$$\boldsymbol{X}(t)=(\alpha_1+i\beta_1)e^{i\omega_1 t}\boldsymbol{H_1}+(\alpha_2+i\beta_2)e^{-i\omega_1 t}\boldsymbol{H_1}+(\alpha_3+i\beta_3)e^{i\omega_2 t}\boldsymbol{H_2}+(\alpha_4+i\beta_4)e^{-i\omega_2 t}\boldsymbol{H_2}$$
$$\Rightarrow \boldsymbol{X}(t)=\alpha_1(e^{i\omega_1 t}+e^{-i\omega_1 t})\boldsymbol{H_1}+i\beta_1(e^{i\omega_1 t}-e^{-i\omega_1 t})\boldsymbol{H_1}+\alpha_3(e^{i\omega_2 t}+e^{-i\omega_2 t})\boldsymbol{H_2}+i\beta_3(e^{i\omega_2 t}-e^{-i\omega_2 t})\boldsymbol{H_2}$$

Οι εκφράσεις στις αγκύλες μπορούν να απλοποιηθούν με τον τύπο του Euler:

$e^{i\omega t}+e^{-i\omega t}=2\cos(\omega t)\ \ \text{και}\ \ e^{i\omega t}-e^{-i\omega t}=2i\sin(\omega t)$

Όποτε,

$$\boldsymbol{X}(t)=2[\alpha_1\cos(\omega_1 t)-\beta_1\sin(\omega_1 t)]\boldsymbol{H_1}+2[\alpha_3\cos(\omega_2 t)-\beta_3\sin(\omega_2 t)]\boldsymbol{H_2}$$

Στη συνέχεια, είναι βολικό να εισαχθούν οι γωνίες φάσης φ_1,φ_2 και να χρησιμοποιηθεί η τριγωνομετρική ταυτότητα

$\cos(\omega t+\varphi)=cos\omega t cos\varphi-sin\omega t sin\varphi$

Ως αποτέλεσμα, η γενική λύση θα γραφτεί με την ακόλουθη μορφή:

$$\boldsymbol{X}(t)=\begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}=A_1\cos(\omega_1 t+\varphi_1)\boldsymbol{H_1}+A_2\cos(\omega_2 t+\varphi_2)\boldsymbol{H_2}$$

όπου οι πραγματικές σταθερές $A_1, A_2, \varphi_1, \varphi_2$ εξαρτώνται από τις αρχικές μετατοπίσεις και ταχύτητες των μαζών και οι ιδιοσυχνότητες ω_1, ω_2 και τα ιδιοδιανύσματα H_1, H_2 δίνονται από τις σχέσεις:

$$\omega_{1,2}=\sqrt{\frac{k}{m_2}}\left[\mu+1\pm\sqrt{(\mu-1)^2+\mu}\right]^{\frac{1}{2}}$$

$$\boldsymbol{H_1}=\begin{bmatrix}H_{11}\\H_{21}\end{bmatrix}=\begin{bmatrix}1\\\dfrac{\mu-1-\sqrt{(\mu-1)^2+\mu}}{\mu}\end{bmatrix}$$

$$\boldsymbol{H_2}=\begin{bmatrix}H_{12}\\H_{22}\end{bmatrix}=\begin{bmatrix}1\\\dfrac{\mu-1+\sqrt{(\mu-1)^2+\mu}}{\mu}\end{bmatrix}$$

Η μεταβολή των ταχυτήτων των μαζών μπορεί να βρεθεί παραγωγίζοντας τη γενική λύση:

$$\dot{\boldsymbol{X}}(t)=\begin{bmatrix}\dot{x_1}(t)\\\dot{x_2}(t)\end{bmatrix}=-A_1\omega_1\sin(\omega_1 t+\varphi_1)\boldsymbol{H_1}-A_2\omega_2\sin(\omega_2 t+\varphi_2)\boldsymbol{H_2}$$

Από εδώ προκύπτει ότι αν οι ταχύτητες είναι μηδενικές στον αρχικό χρόνο $t=0$, οι γωνίες φάσης φ_1,φ_2
είναι επίσης μηδέν. Στη συνέχεια, εξετάζουμε μόνο αυτή την περίπτωση. Η γενική λύση είναι το άθροισμα δύο αρμονικών με συχνότητες ω_1,ω_2:

$$\boldsymbol{X}(t)=\begin{bmatrix}x_1(t)\\x_2(t)\end{bmatrix}=A_1\cos(\omega_1 t)\boldsymbol{H_1}+A_2\cos(\omega_2 t)\boldsymbol{H_2}$$

Υπολογίζουμε τις σταθερές A_1,A_2 ανάλογα με τις αρχικές μετατοπίσεις. Θεωρούμε πως

$$\boldsymbol{X}(0)=\begin{bmatrix}x_1(0)\\x_2(0)\end{bmatrix}=\begin{bmatrix}x_{10}\\x_{20}\end{bmatrix}$$

Συνεπώς,

$$\boldsymbol{X}(0)=\begin{bmatrix}x_1(0)\\x_2(0)\end{bmatrix}=A_1\boldsymbol{H_1}+A_2\boldsymbol{H_2}\Rightarrow A_1H_{11}+A_2H_{12}=x_{10}\quad\text{και}\quad A_1H_{21}+A_2H_{22}=x_{20}$$

Αυτό το αλγεβρικό σύστημα μπορεί να λυθεί με τον κανόνα του Cramer:

$$\Delta=\begin{vmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{vmatrix}=H_{11}H_{22}-H_{12}H_{21}$$

$$\Delta_1=\begin{vmatrix} x_{10} & H_{12} \\ x_{20} & H_{22} \end{vmatrix}=x_{10}H_{22}-x_{20}H_{12}$$

$$\Delta_2=\begin{vmatrix} H_{11} & x_{10} \\ H_{21} & x_{20} \end{vmatrix}=x_{20}H_{11}-x_{10}H_{21}$$

$$\Rightarrow A_1=\frac{\Delta_1}{\Delta}=\frac{x_{10}H_{22}-x_{20}H_{12}}{H_{11}H_{22}-H_{12}H_{21}}$$

$$\Rightarrow A_2=\frac{\Delta_2}{\Delta}=\frac{x_{20}H_{11}-x_{10}H_{21}}{H_{11}H_{22}-H_{12}H_{21}}$$

Επομένως στις αρχικές συνθήκες ισχύει :

$$\boldsymbol{X}(0)=\begin{bmatrix} x_{10} \\ x_{20} \end{bmatrix},\ \ \dot{\boldsymbol{X}}(0)=\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Καταλήγουμε στον ακόλουθο τύπο για τη γενική λύση:

$$\boldsymbol{X}(t)=\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}=A_1\cos(\omega_1 t)\,\boldsymbol{H_1}+A_2\cos(\omega_2 t)\,\boldsymbol{H_2}$$

όπου οι σταθερές A_1, A_2 δίδονται από τους τύπους :

$$A_1=\frac{x_{10}H_{22}-x_{20}H_{12}}{H_{11}H_{22}-H_{12}H_{21}}$$

$$A_2=\frac{x_{20}H_{11}-x_{10}H_{21}}{H_{11}H_{22}-H_{12}H_{21}}$$

Τα ιδιοδιανύσματα και οι ιδιοσυχνότητες εκφράζονται σε σχέση με τον λόγο μάζας μ τη μάζα του δεύτερου σώματος m_2 και τη σταθερά του ελατηρίου k με τους παραπάνω τύπους.

Το σύστημα απλοποιείται σημαντικά όταν και οι δύο μάζες είναι ίσες. Βάζοντας $\mu=1$, λαμβάνουμε τον ακόλουθο τύπο (για τις ίδιες αρχικές συνθήκες):

$$\boldsymbol{X}(t)=\begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}=A_1\cos(\omega_1 t)\,\boldsymbol{H_1}+A_2\cos(\omega_2 t)\,\boldsymbol{H_2}$$

$$\omega_1=\sqrt{\frac{3k}{m}},\ \omega_2=\sqrt{\frac{k}{m}},\ \boldsymbol{H_1}=\begin{bmatrix}H_{11}\\H_{12}\end{bmatrix}=\begin{bmatrix}1\\-1\end{bmatrix},\ \boldsymbol{H_2}=\begin{bmatrix}H_{12}\\H_{22}\end{bmatrix}=\begin{bmatrix}1\\1\end{bmatrix},\ A_1=\frac{x_{10}-x_{20}}{2},\ A_2=\frac{x_{10}+x_{20}}{2}$$

Επομένως, στην περίπτωση ίσων μαζών και ίσων συντελεστών ακαμψίας, η κίνηση των μαζών δίνεται από τους εξής τύπους :

$$x_1(t)=\frac{x_{10}+x_{20}}{2}\cos\left(\sqrt{\frac{k}{m}}t\right)+\frac{x_{10}-x_{20}}{2}\cos\left(\sqrt{\frac{3k}{m}}t\right),$$
$$x_2(t)=\frac{x_{10}+x_{20}}{2}\cos\left(\sqrt{\frac{k}{m}}t\right)-\frac{x_{10}-x_{20}}{2}\cos\left(\sqrt{\frac{3k}{m}}t\right)$$

Ακολουθεί ο κώδικας Python που χρησιμοποιήθηκε:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.callbacks import Callback
from time import time
class PINN:

    def __init__(self, network, m1, m2, k):
        self.network = network
        self.m1 = m1
        self.m2 = m2
        self.k = k
        self.grads = GradientLayer(self.network)

    def build(self):
        # equation input
        eq = tf.keras.layers.Input(shape=(1,))
        eq_m_1 = tf.keras.layers.Input(shape=(1,))
        eq_m_2 = tf.keras.layers.Input(shape=(1,))

        x, dx_dt, d2x_dt2 = self.grads(eq)
        u_eq = dx_dt[..., 0]*(self.m1 * d2x_dt2[..., 0] + 2 * self.k * x[..., 0] -
self.k * x[..., 1]) + dx_dt[..., 1]*(self.m2 * d2x_dt2[..., 1] + 2 * self.k * x[..., 1]
- self.k * x[..., 0])

        u_eq_m1 = self.m1 * d2x_dt2[..., 0] + 2 * self.k * x[..., 0] - self.k * x[...,
1]
        u_eq_m2 = self.m2 * d2x_dt2[..., 1] + 2 * self.k * x[..., 1] - self.k * x[...,
0]
        # condition 1 output
```

```
        u_eq_m1_1 = self.network(eq_m_1)[..., 0]
        u_eq_m2_1 = self.network(eq_m_1)[..., 1]
        # condition 2 output
        x, dx_dt, d2x_dt2 = self.grads(eq_m_2)
        u_eq_m1_2 = dx_dt[..., 0]
        u_eq_m2_2 = dx_dt[..., 1]

        # build the PINN model for equation
        return tf.keras.models.Model(
            inputs=[eq, eq_m_1, eq_m_2],
            outputs=[u_eq, u_eq_m1_1, u_eq_m2_1, u_eq_m1_2, u_eq_m2_2, u_eq_m1,
u_eq_m2])


class GradientLayer(tf.keras.layers.Layer):
    def __init__(self, model, **kwargs):
        self.model = model
        super().__init__(**kwargs)

    def call(self, t ):
        with tf.GradientTape() as g:
            g.watch(t)
            with tf.GradientTape() as gg:
                gg.watch(t)
                x = self.model(t)
            dx_dt = gg.batch_jacobian(x, t)[..., 0]
        d2x_dt2 = g.batch_jacobian(dx_dt, t)[..., 0]
        return x, dx_dt, d2x_dt2

class Network:

    @classmethod
    def build(cls, num_inputs=1, layers=[15, 30, 30, 15], activation='tanh',
num_outputs=2):
        # input layer
        inputs = tf.keras.layers.Input(shape=(num_inputs,))
        # hidden layers
        x = inputs
        initializer = tf.keras.initializers.HeNormal()

        for layer in layers:
            x = tf.keras.layers.Dense(layer, activation=activation,
                                      kernel_initializer=initializer)(x)
        # output layer
        outputs = tf.keras.layers.Dense(num_outputs,
                                        kernel_initializer=initializer)(x)

        return tf.keras.models.Model(inputs=inputs, outputs=outputs)
class stopAtLossValue(Callback):
```

```
    def on_batch_end(self, batch, logs={}):
        THR = 0.001
        if logs.get('loss') <= THR:
            self.model.stop_training = True

start = time()
# number of training samples
num_train_samples = 50
# number of test samples
num_test_samples = 100
# constants
m1 = 4
m2 = 4
k = 2
x1_0 = 0
x2_0 = 1
v1_0 = 0
v2_0 = 0

# build a core network model
network = Network.build()
network.summary()
# build a PINN model
pinn = PINN(network, m1, m2, k).build()

# create training input

eq = np.random.rand(num_train_samples, 1)
eq[..., 0] = np.linspace(0, 20, num_train_samples)

eq_m_1 = np.zeros((num_train_samples, 1))

eq_m_2 = np.zeros((num_train_samples, 1))


# create training output
u_eq = np.zeros((num_train_samples, 1))
u_eq_m1 = np.zeros((num_train_samples, 1))
u_eq_m2 = np.zeros((num_train_samples, 1))
u_eq_m1_1 = x1_0 * np.ones((num_train_samples, 1))
u_eq_m2_1 = x2_0 * np.ones((num_train_samples, 1))
u_eq_m1_2 = v1_0 * np.ones((num_train_samples, 1))
u_eq_m2_2 = v2_0 * np.ones((num_train_samples, 1))


x_train = [eq, eq_m_1, eq_m_2]
y_train = [u_eq, u_eq_m1_1, u_eq_m2_1, u_eq_m1_2, u_eq_m2_2, u_eq_m1, u_eq_m2]

callbacks = stopAtLossValue()
lr = 1e-3
```

```
pinn.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
             loss=tf.keras.losses.mse,
             metrics=[tf.keras.metrics.mse],)
model_history = pinn.fit(x_train, y_train, batch_size=16, epochs=10000,
                         callbacks=callbacks)
print(f"TOTAL TRAINING TIME = {round((time() - start), 2)} seconds")

t = np.linspace(0, 10, num_test_samples).reshape((num_test_samples, 1))
tx = np.concatenate([t], axis=-1)
u = network.predict(tx, batch_size=num_test_samples)
u_m1 = u[..., 0].reshape(t.shape)
u_m2 = u[..., 1].reshape(t.shape)



u_exact_1 = ((x1_0 + x2_0) / 2) * np.cos(np.sqrt((k / m1)) * t) + ((x1_0 - x2_0) / 2) *
np.cos(np.sqrt(((3 * k) / m1)) * t)

u_exact_2 = ((x1_0 + x2_0) / 2) * np.cos(np.sqrt((k / m2)) * t) - ((x1_0 - x2_0) / 2) *
np.cos(np.sqrt(((3 * k) / m2)) * t)

u_exact = u_exact_1 + u_exact_2

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

# plot theory vs prediction

plt.plot(x, u_exact_1, label='theory_1', color='royalblue')
plt.plot(x, u_exact_2, label='theory_2', color='black')

plt.plot(x, u_m1, label='pinn_m1', color='royalblue', linestyle='dotted')
plt.plot(x, u_m2, label='pinn_m2', color='black', linestyle='dotted')
plt.xlabel('t')
plt.ylabel('u(t)')
plt.legend()
plt.show()

# Plot loss values
plt.figure(2)
plt.plot(model_history.history['loss'])
plt.xlabel("Epoch")
plt.title("MSE loss")
plt.show()
```

Ας αναλύσουμε τον κώδικα ενότητα ανά ενότητα:

1. Ο κώδικας ξεκινάει εισάγοντας τις απαραίτητες βιβλιοθήκες: «numpy» για αριθμητικούς υπολογισμούς, «matplotlib.pyplot» για σχεδίαση, «tensorflow» ως backend για λειτουργίες μηχανικής μάθησης και «keras.callbacks» για τον ορισμό προσαρμοσμένων επιστροφών κλήσης κατά τη διάρκεια της εκπαίδευσης.

2. Ο κώδικας ορίζει μια κλάση που ονομάζεται «PINN». Αυτή η κλάση αντιπροσωπεύει το νευρωνικό δίκτυο με πληροφόρηση από τη φυσική και περιέχει μεθόδους για την κατασκευή του μοντέλου. Η μέθοδος `__init__` αρχικοποιεί το PINN με το παρεχόμενο δίκτυο,τις τιμές μάζας (`m1` και `m2`) και τη σταθερά ελατηρίου (`k`). Η μέθοδος «build» κατασκευάζει το μοντέλο PINN χρησιμοποιώντας την κλάση «Model» του TensorFlow.

3. Ο κώδικας ορίζει μια προσαρμοσμένη κλάση «GradientLayer» που είναι μια υποκατηγορία του «tf.keras.layers.Layer». Αυτό το επίπεδο υπολογίζει τις πρώτες και τις δεύτερες παραγώγους της εισόδου σε σχέση με το χρόνο χρησιμοποιώντας τη λειτουργία «GradientTape» του TensorFlow.

4. Ο κώδικας ορίζει μια κλάση που ονομάζεται «Network», η οποία είναι υπεύθυνη για τη δημιουργία του μοντέλου του πυρήνα του νευρωνικού δικτύου. Η μέθοδος «build» λαμβάνει παραμέτρους όπως τον αριθμό των εισόδων, τον αριθμό των κρυφών επιπέδων και μονάδων, τη συνάρτηση ενεργοποίησης και τον αριθμό των εξόδων, και κατασκευάζει ένα νευρωνικό δίκτυο τροφοδοσίας χρησιμοποιώντας τις παρεχόμενες προδιαγραφές.

5. Ο κώδικας ορίζει μια προσαρμοσμένη κλάση επανάκλησης που ονομάζεται `stopAtLossValue`. Αυτή η επανάκληση διακόπτει τη διαδικασία εκπαίδευσης εάν η τιμή απώλειας πέσει κάτω από ένα συγκεκριμένο όριο («THR»).

6. Ο κώδικας προετοιμάζει τις μεταβλητές και τις σταθερές που απαιτούνται για το μοντέλο PINN, όπως τον αριθμό των δειγμάτων εκπαίδευσης και δοκιμής («num_train_samples» και «num_test_samples»), τις τιμές μάζας («m1» και «m2»), τη σταθερά ελατηρίου («k»), τις αρχικές θέσεις (`x1_0` και `x2_0`) και τις αρχικές ταχύτητες (`v1_0` και `v2_0`).

7. Ο κώδικας δημιουργεί το μοντέλο καλώντας τη μέθοδο «build» της κλάσης «Network». Εκτυπώνει μια αναφορά της αρχιτεκτονικής του μοντέλου.

8. Ο κώδικας κατασκευάζει το μοντέλο PINN καλώντας τη μέθοδο «build» της κλάσης «PINN» και διαβιβάζει το μοντέλο του βασικού δικτύου και άλλες παραμέτρους.

9. Ο κώδικας δημιουργεί εισόδους εκπαίδευσης (`eq`, `eq_m_1`, και `eq_m_2`) και εξόδους εκπαίδευσης (`u_eq`, `u_eq_m1_1`, "u_eq_m2_1`, "u_eq_m1_2`, "u_eq_m2_e_, «u_eq_m2») για το μοντέλο PINN.

10. Ο κώδικας συντάσσει το μοντέλο PINN με τον καθορισμένο βελτιστοποιητή (Adam), τη συνάρτηση απώλειας (μέσο τετραγωνικό σφάλμα) και τα κριτήρια (μέσο τετραγωνικό σφάλμα).

11. Ο κώδικας εκπαιδεύει το μοντέλο PINN καλώντας τη μέθοδο «fit» με τις εισόδους και τις εξόδους εκπαίδευσης, το μέγεθος παρτίδας, τον αριθμό των εποχών και την επανάκληση «stopAtLossValue».

12. Ο κώδικας μετρά και εκτυπώνει τον συνολικό χρόνο εκπαίδευσης.

13. Ο κώδικας δημιουργεί δοκιμαστικές εισόδους (`t` και `tx`) και προβλέπει τις εξόδους (`u`) χρησιμοποιώντας το μοντέλο του βασικού δικτύου.

14. Ο κώδικας υπολογίζει την ακριβή λύση ("u_exact_1" και "u_exact_2") για το δεδομένο πρόβλημα της φυσικής.

15. Ο κώδικας σχεδιάζει τις θεωρητικές λύσεις («u_exact_1» και «u_exact_2») καθώς και τις λύσεις πρόβλεψης («u_m1» και «u_m2») χρησιμοποιώντας matplotlib.

16. Ο κώδικας απεικονίζει τις τιμές απώλειας σε εποχές κατά τη διάρκεια της εκπαίδευσης.

Ο παρεχόμενος κώδικας υλοποιεί ένα νευρωνικό δίκτυο με πληροφόρηση φυσικής (PINN) για την επίλυση ενός φυσικού συστήματος που διέπεται από τρείς συνήθεις διαφορικές εξισώσεις δεύτερης τάξης με ένα μοντέλο νευρωνικού δικτύου. Ακολουθεί μια ανάλυση του τρόπου λειτουργίας του παρεχόμενου PINN:

- Κλάση PINN (`PINN`):
  - Αυτή η κλάση ξεκινά με ένα νευρωνικό δίκτυο («network») και τις παραμέτρους του φυσικού συστήματος «m1», «m2» και «k». Το «network» είναι ένα ξεχωριστό νευρωνικό δίκτυο που θα χρησιμοποιηθεί για την προσέγγιση της λύσης στο φυσικό σύστημα. Η μέθοδος «build» δημιουργεί το μοντέλο PINN, το οποίο αποτελείται από: τους όρους απώλειας που βασίζονται στη φυσική και τις αρχικές συνθήκες.

- Κλάση GradientLayer (`GradientLayer`):
  - Αυτό το προσαρμοσμένο επίπεδο υπολογίζει τις παραγώγους της εξόδου του νευρωνικού δικτύου σε σχέση με το χρόνο (`t`) χρησιμοποιώντας την gradient tape του TensorFlow. Η μέθοδος «call» υπολογίζει την πρώτη και τη δεύτερη παράγωγο της εξόδου του νευρωνικού δικτύου σε σχέση με το χρόνο. Αυτές οι πληροφορίες χρησιμοποιούνται για την επιβολή των εξισώσεων της φυσικής.

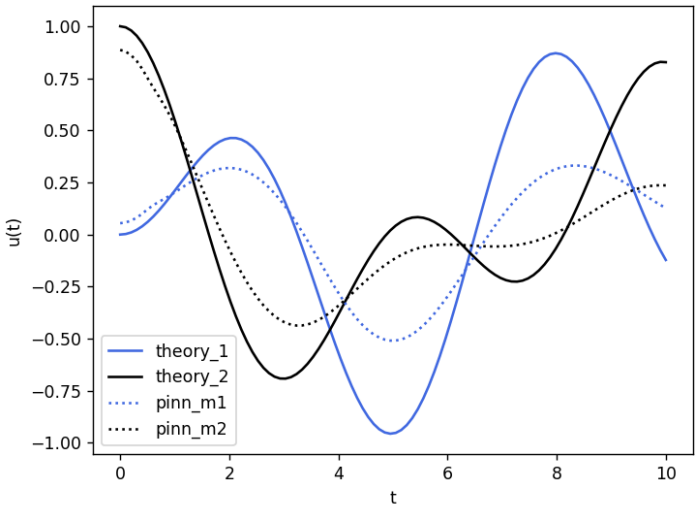
- Κλάση Network (`Network`):
  - Αυτή η κλάση χρησιμοποιείται για την κατασκευή του νευρωνικού δικτύου που προσεγγίζει τη λύση στο φυσικό σύστημα. Καθορίζει την αρχιτεκτονική (αριθμός επιπέδων, νευρώνες ανά επίπεδο, συνάρτηση ενεργοποίησης) του νευρωνικού δικτύου με τροφοδότηση.

- Κλάση StopAtLossValue (`stopAtLossValue` Callback):
  - Αυτή η επανάκληση χρησιμοποιείται κατά τη διάρκεια της εκπαίδευσης για τη διακοπή της όταν η απώλεια πέσει κάτω από ένα συγκεκριμένο όριο («THR»).

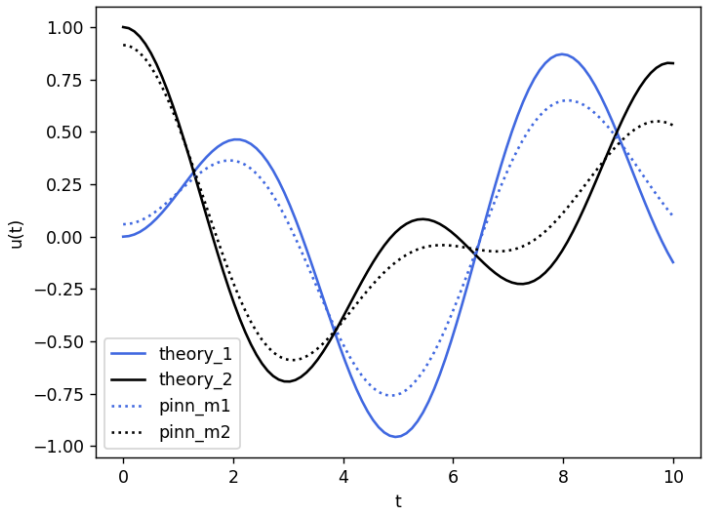
- Εκπαίδευση:
  - Ο κώδικας δημιουργεί δεδομένα εκπαίδευσης (`x_train`, `y_train`) με βάση τις ΣΔΕ και τις αρχικές συνθήκες. Το μοντέλο PINN έχει γίνει compile με τον βελτιστοποιητή Adam και την απώλεια μέσου τετραγωνικού σφάλματος (MSE). Η προπόνηση εκτελείται χρησιμοποιώντας τη μέθοδο «fit» με προσαρμοσμένη επανάκληση για διακοπή της εκπαίδευσης όταν η απώλεια είναι κάτω από το όριο.

- Δοκιμή και αξιολόγηση:
  - Μετά την εκπαίδευση, το μοντέλο PINN χρησιμοποιείται για την πρόβλεψη των λύσεων «u_m1» και «u_m2» σε ένα δοκιμαστικό εύρος χρόνου. Αυτές οι προβλέψεις βασίζονται στην έξοδο του νευρωνικού δικτύου. Υπολογίζονται επίσης αναλυτικές λύσεις ("u_exact_1" και "u_exact_2") για το ίδιο φυσικό σύστημα. Στη συνέχεια, ο κώδικας σχεδιάζει τις αναλυτικές και τις λύσεις πρόβλεψης για να απεικονίσει πόσο καλά το PINN προσεγγίζει το φυσικό σύστημα.

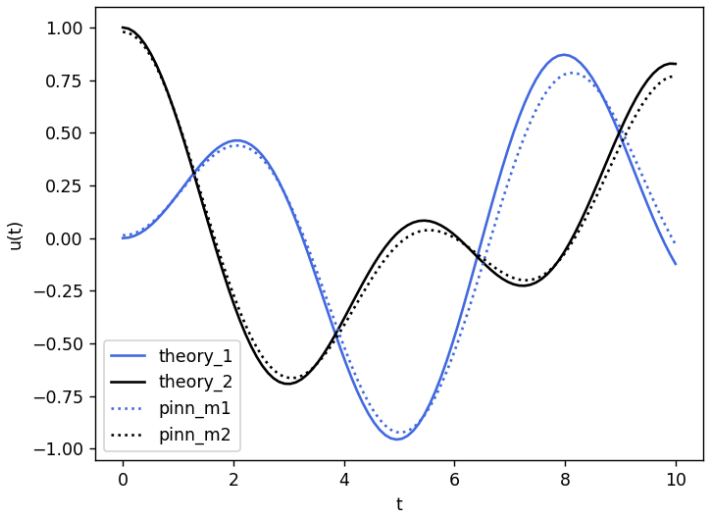
Σε συμφωνία με το πρόβλημα 1, η συμπεριφορά του κώδικα του προβλήματος 2 μελετήθηκε έχοντας σταθερές τις εξής παραμέτρους: τη δομή του δικτύου που ορίστηκε ως [15, 30, 30, 15], η tanh χρησιμοποιήθηκε ως συνάρτηση ενεργοποίησης, το batch size ορίστηκε ίσο με 16 και οι κύκλοι εκπαίδευσης ήταν ίσοι με 10000. Σε κάθε εκτέλεση του κώδικα ουσιαστικά μεταβάλλαμε μόνο τον αριθμό των σημείων για εκπαίδευση (training sample). Παρατίθενται παρακάτω τα γραφήματα για κάθε διαφορετική εκτέλεση του κώδικα.



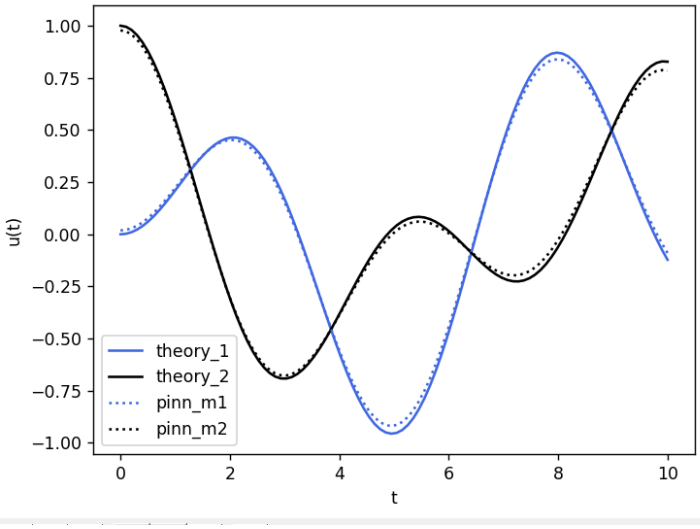
Εικόνα 22: Γραφική αναπαράσταση αναλυτικής λύσης για κάθε μάζα και η πρόβλεψη του μοντέλου (training sample = 50).



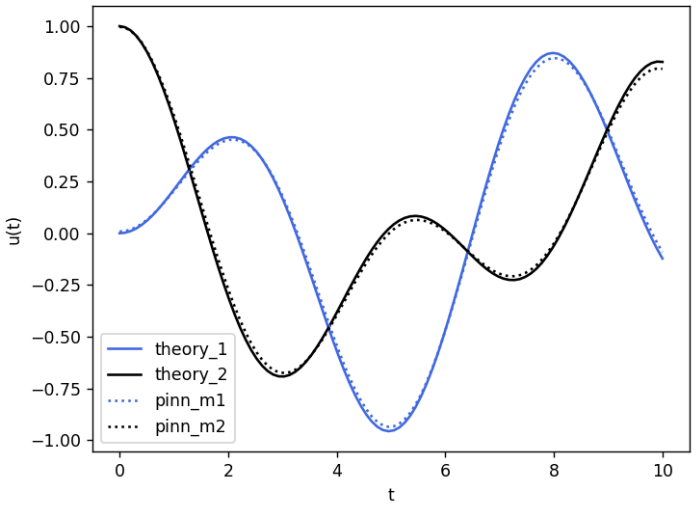
Εικόνα 23: Γραφική αναπαράσταση αναλυτικής λύσης για κάθε μάζα και η πρόβλεψη του μοντέλου (training sample = 100).



Εικόνα 24: Γραφική αναπαράσταση αναλυτικής λύσης για κάθε μάζα και η πρόβλεψη του μοντέλου (training sample = 150).



Εικόνα 25: Γραφική αναπαράσταση αναλυτικής λύσης για κάθε μάζα και η πρόβλεψη του μοντέλου (training sample = 200).



Εικόνα 26: Γραφική αναπαράσταση αναλυτικής λύσης για κάθε μάζα και η πρόβλεψη του μοντέλου (training sample = 250).

Πίνακας 4.2 (Πίνακας αποτελεσμάτων από την επίλυση του προβλήματος 2)

Εκτέλεση	**Layers**	**Batch size**	**Epochs**	**Training Sample**	**Loss mse**	**Time (sec)**
1	[15, 30, 30, 15]	16	10000	50	0.1079	233.42
2	[15, 30, 30, 15]	16	10000	100	0.0541	311.75
3	[15, 30, 30, 15]	16	10000	150	0.0226	404.75
4	[15, 30, 30, 15]	16	10000	200	0.0043	502.11
5	[15, 30, 30, 15]	16	6987	250	0.0009	418.03

4.3 Πρόβλημα 3: Πρόβλεψη συντελεστή απόσβεσης στην εξίσωση κίνησης αρμονικού ταλαντωτή

Σε αυτή τη παράγραφο παραθέτω το κώδικα Python που χρησιμοποιήθηκε για την κατασκευή ενός αντίστροφου μοντέλου PINN για την εξίσωση ενός αρμονικού ταλαντωτή στην οποία ο συντελεστής απόσβεσης ήταν η άγνωστη παράμετρος. Ο κώδικας που αναπτύχθηκε στηρίχθηκε στο τρόπο γραφής των προγραμμάτων που βρίσκονται στους συνδέσμους https://github.com/alexpapados/Physics-Informed-Deep-Learning-Solid-and-Fluid-Mechanics, https://github.com/benmoseley/harmonic-oscillator-pinn/blob/main/Harmonic%20oscillator%20PINN.ipynb και https://github.com/jmorrow1000/PINN-iPINN/tree/main.

Ο κώδικας Python που χρησιμοποιήθηκε είναι ο εξής :

```
import tensorflow as tf
import numpy as np
import time
import matplotlib.pyplot as plt

# Generate exact solution dataset
def calculate_x_exact_at_time_t(d, w0, t):
    w = np.sqrt(w0 ** 2 - d ** 2)
    phi = np.arctan(-d / w)
    A = 1 / (2 * np.cos(phi))
    cos = np.cos(phi + w * t)
    exp = np.exp(-d * t)
    x = exp * 2 * A * cos
    return x


# Deep Neural Network
class Network:

    @classmethod
    def build(cls, num_inputs=1, layers=[200, 200, 200], activation='gelu',
num_outputs=1):
        # input layer
        inputs = tf.keras.layers.Input(shape=(num_inputs,))
        # hidden layers
        x = inputs
        initializer = tf.keras.initializers.glorot_normal
        for layer in layers:
            x = tf.keras.layers.Dense(layer, activation=activation,
                                      kernel_initializer=initializer)(x)

        # output layer
        outputs = tf.keras.layers.Dense(num_outputs, kernel_initializer=initializer)(x)

        return tf.keras.models.Model(inputs=inputs, outputs=outputs)
```

```
# Physics Informed Neural Network
class PINNs:
    def __init__(self, t_train, t_obs, x_obs, m, w0, network):
        self.t_train = tf.convert_to_tensor(t_train, dtype=tf.float32)
        self.t_obs = tf.convert_to_tensor(t_obs, dtype=tf.float32)
        self.x_obs = tf.convert_to_tensor(x_obs, dtype=tf.float32)
        self.t_init = tf.Variable([[0.]], dtype=tf.float32)
        self.m = m
        self.d = tf.Variable(initial_value=0.1, dtype=tf.float32, trainable=True,
name='var_d')
        self.w0 = w0
        self.dnn = network
        self.optimizer = tf.optimizers.Adam()
        self.iter = 0

    def net_x(self, t):
        x = self.dnn(t)
        return x

    def loss_func(self):
        with tf.GradientTape(persistent=True) as tape:
            tape.watch(self.t_train)
            tape.watch(self.t_obs)
            tape.watch(self.t_init)

            x_pred = self.net_x(self.t_train)

            # Equation of motion constraint
            dx_dt = tape.gradient(x_pred, self.t_train)
            d2x_dt2 = tape.gradient(dx_dt, self.t_train)
            residual = self.m * d2x_dt2 + 2 * self.m * self.d * dx_dt + self.m *
self.w0 ** 2 * x_pred
            eq_loss = tf.reduce_mean(tf.square(residual))

            # Initial conditions constraint
            x0_pred = self.net_x(self.t_init)
            ic_1_loss = tf.reduce_mean(tf.square(x0_pred - 1))
            v0_pred = tape.gradient(x0_pred, self.t_init)
            ic_2_loss = tf.reduce_mean(tf.square(v0_pred - 0))

            # Discrepancy loss
            x_pred_obs = self.net_x(self.t_obs)
            disc_loss = tf.reduce_mean(tf.square(self.x_obs - x_pred_obs))

            loss = disc_loss*10000 + eq_loss + ic_1_loss + ic_2_loss

        trainable = self.dnn.trainable_variables
        trainable.append(self.d)
```

```
        self.optimizer.minimize(loss, trainable, tape=tape)
        self.iter += 1
        if self.iter % 1 == 0:
            print('Loss:', loss.numpy(), 'd_exact:', d_exact, 'd_PINNs:',
self.d.numpy(), 'iteration:', self.iter)
        return loss

    def train(self, nIter):
        for _ in range(nIter):
            self.loss_func()


# Generate training dataset
d_exact = 2.0
w0_exact = 20.0
m_exact = 1.0

t_obs = tf.random.uniform((50, 1), maxval=1)
x_obs = calculate_x_exact_at_time_t(d_exact, w0_exact, t_obs)
t_train = np.linspace(0, 1, 100).flatten()[:, None]
# Define PINNs Model
network = Network.build()
model = PINNs(t_train, t_obs, x_obs, m_exact, w0_exact, network)

# Train PINNs
tic = time.time()
model.train(30000)
toc = time.time()
print('Total training time:', toc - tic)
# Get the learned damping coefficient
d_learned = model.d.numpy()
print('Learned damping coefficient:', d_learned)

# plot theory vs prediction
t_test = np.linspace(0, 2, 100).flatten()[:, None]
x_exact = calculate_x_exact_at_time_t(d_exact,w0_exact,t_test)
u = network.predict(t_test)
plt.figure(figsize=(6, 2.5))
plt.scatter(t_obs[:, 0], x_obs[:, 0], label="Observations", alpha=0.6)
plt.plot(t_test[:, 0], u[:, 0], label="PINN solution", color="tab:green")
plt.plot(t_test[:, 0], x_exact[:, 0], label="Exact solution", color="tab:red",
linestyle='dotted')
plt.legend()
plt.show()
```

Ακολουθεί μια ανάλυση του κώδικα:

1. Εισαγωγή των απαραίτητων βιβλιοθηκών: TensorFlow, NumPy, Matplotlib και time.

2. Ορισμός συνάρτησης «calculate_x_exact_at_time_t»:
   - Αυτή η συνάρτηση υπολογίζει την ακριβή λύση για ένα σύνολο δεδομένων παραμέτρων («d», «w0», «t») χρησιμοποιώντας μαθηματικές εξισώσεις.

3. Ορισμός της κλάσης «Network»:
   - Αυτή η κλάση κατασκευάζει ένα νευρωνικό δίκτυο πρόσθιας τροφοδότησης. Το δίκτυο αρχικοποιείται με έναν καθορισμένο αριθμό εισόδων, τα μεγέθη των επιπέδων, τη συνάρτηση ενεργοποίησης και τον αριθμό των εξόδων.

4. Ορισμός της κλάσης «PINNs»:
   - Αυτή η κλάση αντιπροσωπεύει το πυρήνα του κώδικα, το Physics Informed Neural Network. Αρχικοποιείται με τα δεδομένα εκπαίδευσης και παρατήρησης, τις γνωστές παραμέτρους(m, w0) του συστήματος και το νευρωνικό δίκτυο. Η μέθοδος `net_x` προβλέπει τις θέσεις χρησιμοποιώντας το νευρωνικό δίκτυο. Η μέθοδος `loss_func` υπολογίζει την απώλεια, η οποία είναι ένας συνδυασμός της απώλειας ασυμφωνίας (διαφορά μεταξύ της πρόβλεψης και των ακριβών λύσεων), του περιορισμού της εξίσωσης κίνησης και των περιορισμών των αρχικών συνθηκών. Η μέθοδος `train` βελτιστοποιεί τις παραμέτρους του μοντέλου χρησιμοποιώντας τον βελτιστοποιητή Adam.

5. Δημιουργία συνόλου δεδομένων εκπαίδευσης:
   - Ορίζονται οι ακριβείς τιμές για τον συντελεστή απόσβεσης (`d_exact`), τη γωνιακή συχνότητα (`w0_exact`) και τη μάζα (`m_exact`). Δημιουργούνται τυχαίοι χρόνοι παρατήρησης (`t_obs`) και αντίστοιχες παρατηρούμενες θέσεις (`x_obs`) καθώς και ο χρόνος εκπαίδευσης (`t_train`).

6. Αρχικοποίηση μοντέλου PINNs:
   - Δημιουργείται ένα instance της κλάσης `PINNs`, αρχικοποιώντας το μοντέλο με τα δεδομένα εκπαίδευσης, τις ακριβείς παραμέτρους και το νευρωνικό δίκτυο.

7. Εκπαίδευση του μοντέλου PINNs (`model.train(30000)`):
   - Η μέθοδος `train` της κλάσης `PINNs` καλείται για να εκπαιδεύσει το μοντέλο για 30000 επαναλήψεις. Κατά τη διάρκεια κάθε επανάληψης, το μοντέλο βελτιστοποιεί τις παραμέτρους του για την ελαχιστοποίηση της καθορισμένης συνάρτησης απωλειών, η οποία περιλαμβάνει τόσο περιορισμούς φυσικής όσο και προσαρμογή δεδομένων.

8. Εκτύπωση της πρόβλεψης του συντελεστή απόσβεσης και του χρόνου εκπαίδευσης:
   - Μετά την εκπαίδευση, ο κώδικας εκτυπώνει τον συντελεστή απόσβεσης που έχει μάθει (`d_learned`), ο οποίος αντιπροσωπεύει πόσο καλά το μοντέλο αντιλήφθηκε τον συντελεστή απόσβεσης από τα δεδομένα, καθώς και το συνολικό χρόνο εκπαίδευσης.

9. Απεικόνιση αποτελεσμάτων:
   - Δημιουργείται ένα εύρος χρόνου δοκιμής (`t_test`) για την αξιολόγηση της απόδοσης του μοντέλου σε ένα ευρύτερο χρονικό διάστημα. Η ακριβής λύση, οι προβλέψεις του PINN και τα παρατηρούμενα δεδομένα απεικονίζονται με τη χρήση του Matplotlib για σύγκριση.

Ο παρεχόμενος κώδικας υλοποιεί ένα νευρωνικό δίκτυο με πληροφόρηση φυσικής (PINN) για την επίλυση μιας διαφορικής εξίσωσης στο πλαίσιο της εύρεσης του συντελεστή απόσβεσης σε ένα δυναμικό σύστημα.

Ας δούμε λίγο πιο αναλυτικά τη κλάση `PINNs` το πυρήνα δηλαδή του κώδικα.

Κλάση `**PINNs**` :

Μέθοδος `__init__`: Αρχικοποίηση του αντικειμένου με τις απαραίτητες παραμέτρους και μεταβλητές.

- `t_train`: Χρόνοι εκπαίδευσης για το μοντέλο PINNs.
- `t_obs`: Χρόνοι παρατήρησης για τα παρατηρούμενα δεδομένα.
- `x_obs`: Παρατηρούμενες θέσεις που αντιστοιχούν στο `t_obs`.
- `m`: Μάζα του συστήματος.
- `w0`: Γωνιακή συχνότητα του ταλαντωτή.
- `network`: Το νευρωνικό δίκτυο που χρησιμοποιείται για τις προβλέψεις.
- `self.t_train`, `self.t_obs`, `self.x_obs`, `self.m`, `self.w0`, `self.dnn`: Μεταβλητές κλάσης για την αποθήκευση των παρεχόμενων δεδομένων.
- `self.d`: Μια μεταβλητή με δυνατότητα εκπαίδευσης που αντιπροσωπεύει τον συντελεστή απόσβεσης.
- `self.optimizer`: Ο βελτιστοποιητής Adam που χρησιμοποιείται για την εκπαίδευση.
- `self.iter`: Ένας μετρητής για την παρακολούθηση του αριθμού των επαναλήψεων εκπαίδευσης.

Μέθοδος `net_x`: Αυτή η μέθοδος προβλέπει τις θέσεις χρησιμοποιώντας το νευρωνικό δίκτυο για δεδομένο χρόνο εισόδου `t`.

- Καλεί το νευρωνικό δίκτυο (`self.dnn`) με τον χρόνο εισόδου`t` για να προβλέψει τις θέσεις.

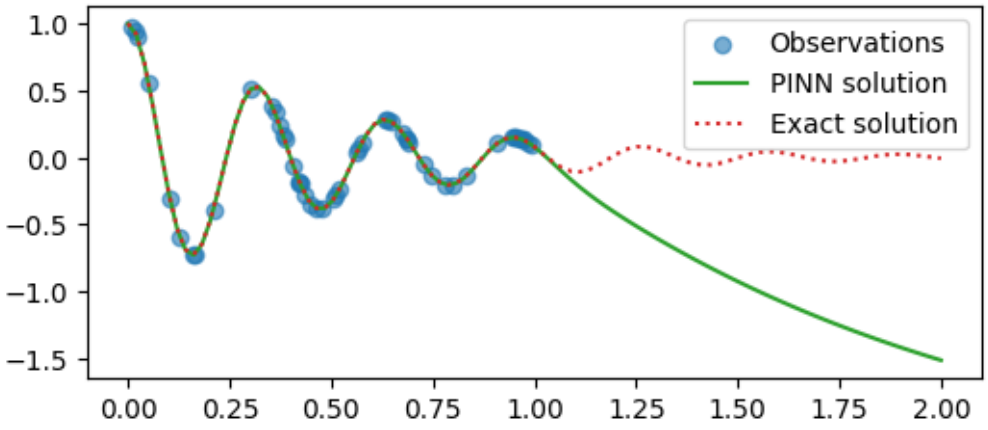
Μέθοδος `loss_func`: Αυτή η μέθοδος υπολογίζει τη συνάρτηση απωλειών, η οποία περιλαμβάνει την απώλεια ασυμφωνίας, τον περιορισμό της εξίσωσης κίνησης και τους περιορισμούς των αρχικών συνθηκών.

- Χρησιμοποιεί το GradientTape της TensorFlow για τον υπολογισμό των κλίσεων σε σχέση με τους χρόνους εκπαίδευσης και παρατήρησης.
- Υπολογίζει τις προβλεπόμενες θέσεις, ταχύτητες και επιταχύνσεις.
- Διαμορφώνει την απώλεια ασυμφωνίας, την εξίσωση του περιορισμού κίνησης και τους περιορισμούς αρχικών συνθηκών.
- Συνδυάζει τις επιμέρους απώλειες και επιστρέφει τη συνολική απώλεια.

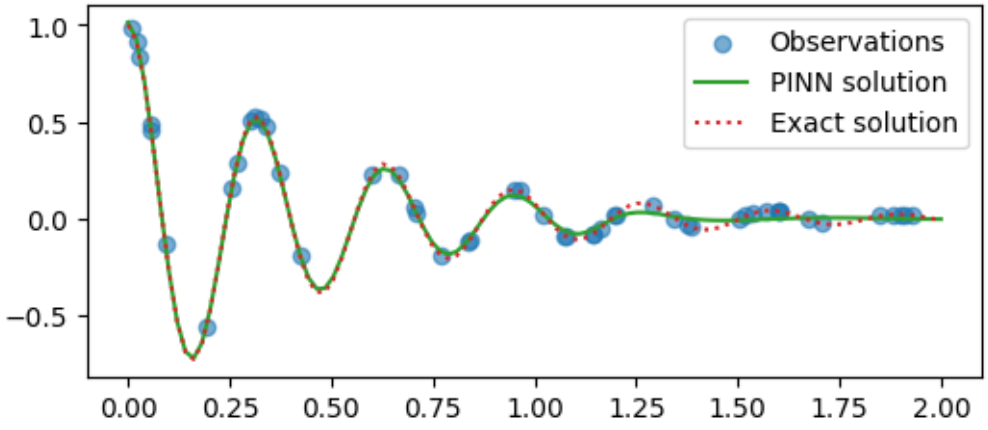
Μέθοδος `train`: Αυτή η μέθοδος εκπαιδεύει το μοντέλο PINNs για έναν καθορισμένο αριθμό επαναλήψεων.

- Καλεί επαναληπτικά τη μέθοδο `loss_func` και χρησιμοποιεί τον βελτιστοποιητή Adam για την ελαχιστοποίηση της συνολικής απώλειας.
- Εκτυπώνει περιοδικά την απώλεια, τον μαθημένο συντελεστή απόσβεσης και τις πληροφορίες της επανάληψης.

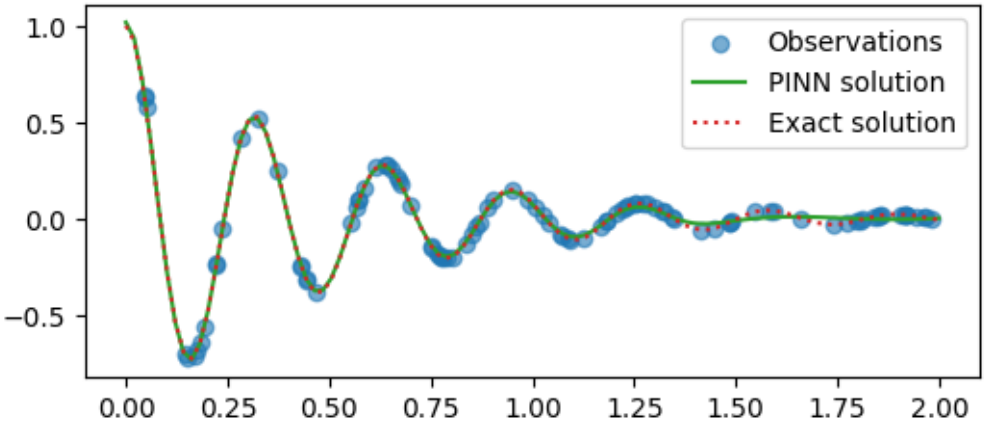
Για να μελετήσουμε τη συμπεριφορά του κώδικα του προβλήματος 3 παραμετροποιήσαμε σε κάθε εκτέλεση τη δομή του δικτύου, τον αριθμό των σημείων για εκπαίδευση, τον αριθμό των δεδομένων παρατήρησης καθώς και το διάστημα μέσα από το οποίο προέρχονται τα σημεία αυτά. Οι κύκλοι εκπαίδευσης ήταν σταθεροί σε κάθε εκτέλεση και ίσοι με 30000. Παραθέτουμε παρακάτω τα αποτελέσματα από κάθε εκτέλεση.



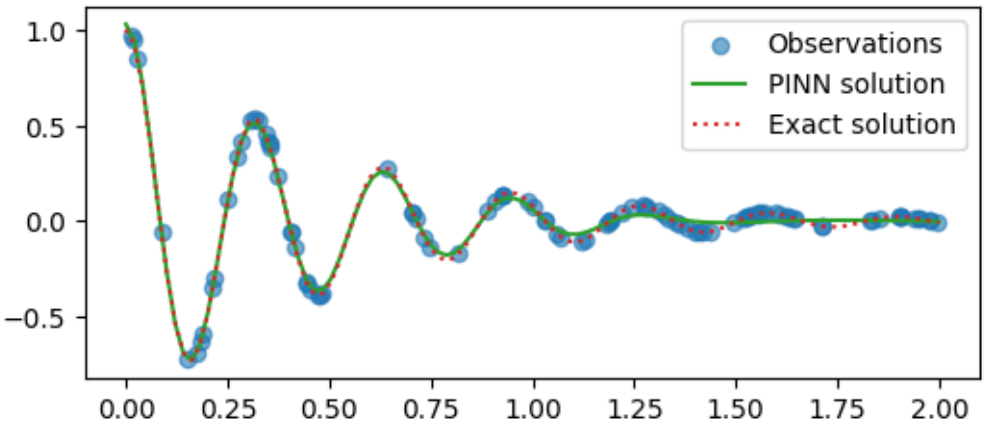
Εικόνα 27: Γραφική αναπαράσταση της αναλυτικής λύσης, της πρόβλεψης του μοντέλου και των δεδομένων παρατήρησης (t_obs=50 in [0,1], t_train=100 in [0,1], layers=[1,20, 20, 20,1], d_exact= 2.0, d_PINNs= 2.0035133)



Εικόνα 28: Γραφική αναπαράσταση της αναλυτικής λύσης, της πρόβλεψης του μοντέλου και των δεδομένων παρατήρησης (t_obs=50 in [0,2], t_train=100 in [0,2], layers=[1,20, 20, 20,1], d_exact= 2.0, d_PINNs= 2.1616294)



Εικόνα 29: Γραφική αναπαράσταση της αναλυτικής λύσης, της πρόβλεψης του μοντέλου και των δεδομένων παρατήρησης (t_obs=100 in [0,2], t_train=100 in [0,2], layers=[1,20, 20, 20,1], d_exact= 2.0, d_PINNs= 2.0870335)



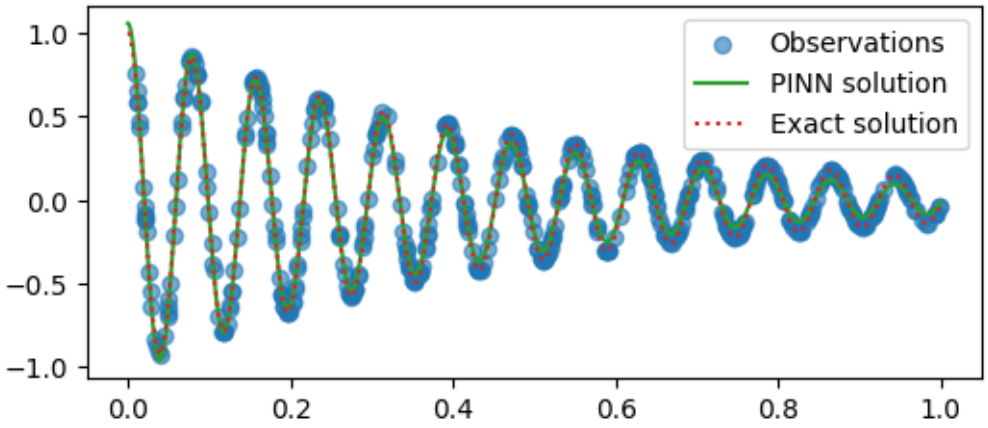
Εικόνα 30: Γραφική αναπαράσταση της αναλυτικής λύσης, της πρόβλεψης του μοντέλου και των δεδομένων παρατήρησης (t_obs=100 in [0,2], t_train=200 in [0,2], layers=[1,20, 20, 20,1], d_exact= 2.0, d_PINNs= 2.190139)

Στη προσπάθεια μας να διερευνήσουμε το πόσο καλά το PINN προσαρμόζεται σε ταλαντώσεις υψηλότερης συχνότητας αυξήσαμε την τιμή του ω_0 από 20 σε 80. Εκτελώντας το κώδικα αρκετές φορές, μεταβάλλοντας κάθε φορά διαφορετικές παραμέτρους του (συνάρτηση ενεργοποίησης, αρχικοποίηση βαρών, αριθμός σημείων εκπαίδευσης κτλ), παρατηρήσαμε ότι το PINN δυσκολεύεται να συγκλίνει.

Η αύξηση του μεγέθους της συχνότητας καθιστά το πρόβλημα πιο δύσκολο ως προς την επίλυση του λόγω του Spectral Bias των νευρωνικών δικτύων. [29] Το Spectral Bias πρόκειται για ένα φαινόμενο που παρατηρείται κατά την εκπαίδευση των νευρωνικών δικτύων και δηλώνει πως τα νευρωνικά δίκτυα δρουν μεροληπτικά προς την εκμάθηση λιγότερο πολύπλοκων συναρτήσεων.

Για να παρακάμψουμε αυτή τη δυσκολία επιλέξαμε να χρησιμοποιήσουμε τη μέθοδο του Transfer Learning. [30] Σύμφωνα με τη μέθοδο αυτή ένα μοντέλο που αναπτύχθηκε για μια εργασία επαναχρησιμοποιείται ως αφετηρία για ένα μοντέλο σε μια δεύτερη εργασία.

Με αυτό το τρόπο λοιπόν εκπαιδεύσαμε το μοντέλο μας για $\omega_0 = 10$ και χρησιμοποιήσαμε το εκπαιδευμένο αυτό μοντέλο για να προσεγγίσουμε τη λύση για $\omega_0 = 20$. Αυτή η διαδικασία επαναλήφθηκε έως ότου να προσεγγιστεί το πρόβλημα με συχνότητα $\omega_0 = 80$.



Εικόνα 31: Γραφική αναπαράσταση της αναλυτικής λύσης, της πρόβλεψης του μοντέλου και των δεδομένων παρατήρησης για $\omega_0 = 80$

Βιβλιογραφία

[1] McCullogh W. S. & Pitts W., (1943) A logical calculus of the ideas immanent in nervous activitity, The Bulletin of Mathematical Biophysics, Vol. 5(4), 115-133. Available at: https://link.springer.com/article/10.1007/BF02478259

[2] McCullogh W. S. & Pitts W., (1947) How we know universals: the perception of auditory and visuals forms, The Bulletin of Mathematical Biophysics, Vol. 9, 127-147. Available at: https://link.springer.com/article/10.1007/BF02478291

[3] Neumann J., (1958) The computer and the brain, Yale University Press, New Haven. Available at: https://complexityexplorer.s3.amazonaws.com/supplemental_materials/5.6+Artificial+Life/The+Computer+and+The+Brain_text.pdf

[4] Hebb D., (1949) The organisation of behavior. Available at: https://pure.mpg.de/rest/items/item_2346268_3/component/file_2346267/content

[5] Rosenblatt, F., (1958) The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408. Available at: https://doi.org/10.1037/h0042519

[6] Minsky M. & Papert S., (1969) Perceptrons, MIT Press, expanded edition, 1988. Available at: https://mitpress.mit.edu/9780262630221/perceptrons/

[7] Widrow B. & Hoff M. E., (1960) Adaptive Switching Circuits, 1960 WESCON Convention, Record Part 4, pp. 96-104. Available at: https://www-isl.stanford.edu/~widrow/papers/c1960adaptiveswitching.pdf

[8] Hopfield J. J., (1982) Neural networks and physical systems with emergent collective computational abilities, Proceedings of the National Academy of Sciences. Available at: https://www.pnas.org/doi/epdf/10.1073/pnas.79.8.2554

[9] Argyrakis, P., (2001). Νευρωνικά Δίκτυα και Εφαρμογές. Ελληνικό Ανοικτό Πανεπιστήμιο, Σχολή θετικών Επιστημών και Τεχνολογίας. ISBN, 960-538. Available at: Νευρωνικά Δίκτυα (βιβλίο) Π_Αργυράκης.pdf

[10] Rumelhart, D. E., McClelland, J. L., & the PDP Research Group. (1986). Parallel distributed processing: Explorations in the microstructure of cognition (Vol. 1). MIT Press. Available at: https://doi.org/10.7551/mitpress/5236.001.0001

[11] Linnainmaa, S., (1976) Taylor expansion of the accumulated rounding error. BIT Numerical Mathematics, 16(2):146–160. Available at: https://link.springer.com/article/10.1007/BF01931367

[12] Katsikis, D., (2021) Solving direct problems in mechanics using physics informed neural networks (PINNs), Thesis, Chania: Technical University of Crete. Available at: https://doi.org/10.26233/heallink.tuc.90771

[13] Mumford, C. L., & Jain, L. C. (2009). Synergy in Computational Intelligence. Berlin. Available at: https://link.springer.com/chapter/10.1007/978-3-642-01799-5_1

[14] Small, E., (2021). An Analysis of Physics-Informed Neural-Networks. Available at: https://arxiv.org/pdf/2303.02890.pdf

[15] Hornik, K., Stinchcombe, M. & White H. (1989). Multilayer feedforward networks are universal approximators. Neural Networks. Vol.2. Pergamon Press. pp.359–366. Available at: https://doi.org/10.1016/0893-6080(89)90020-8

[16] Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1997). Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. IEEE Transactions on Neural Networks, 9(5). Available at: https://arxiv.org/pdf/physics/9705023.pdf

[17] Lee, H., & Kang, I. (1990). Neural algorithms for solving differential equations. Journal of Computational Physics(91). Available at: https://www.sciencedirect.com/science/article/pii/002199919090007N

[18] Meade Jr, A. J., & Fernadez, A. A. (1994). The numerical solution of Linear Ordinary Differential Equations by Feedforward Neural Networks. Math. Comput. Modelling, 19(12). Available at: https://www.sciencedirect.com/science/article/pii/0895717794900957

[19] Stavroulakis, G. E., Avdelas, A., Abdalla, K. M., & Panagiotopoulos, P. D. (1997). A neural network approach to the modelling, calculation and identification of semi-rigid connections in steel structures. Journal of Constructional Steel Research, 44(1-2). Available at: https://www.sciencedirect.com/science/article/abs/pii/S0143974X97000394

[20] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Physics Informed Deep Learning (Part I)Data-driven Solutions of Nonlinear Partial Differential Equations. Available at: https://arxiv.org/pdf/1711.10561.pdf

[21] Peng, W., Zhang, J., Zhou, W., Zhao, X., Yao, W., & Chen, X. (2021). IDRLnet: A Physics-Informed Neural Network Library. Available at: https://arxiv.org/pdf/2107.04320.pdf

[22] Karniadakis, G. E., Kevrekidis, Y., Lu, L., & Perdikaris, P. (2021). Physics-informed machine learning. Nature Reviews Physics volume(3). Available at: https://www.nature.com/articles/s42254-021-00314-5

[23] Muradova, A. D., & Stavroulakis, G. E. (2022). Physics-Informed Neural Networks for elastic plate problems. Available at: https://www.researchgate.net/publication/359095092_PHYSICS-INFORMED_NEURAL_NETWORKS_FOR_ELASTIC_PLATE_PROBLEMS

[24] Prantikos, K., (2022) Physics-Informed Neural Network Solution of Point Kinetics Equations for PUR-1 Digital Twin, Thesis, West Lafayette, Indiana: School of Nuclear Engineering. Available at: Physics-Informed Neural Network Solution of Point Kinetics Equations for a Nuclear Reactor Digital Twin (mdpi.com)

[25] Subramanian, N., K., (2021) Physics Informed Neural Networks in Fluid Dynamics, Master Thesis, Technical University of Munich. Available at: https://mediatum.ub.tum.de/doc/1632867/1632867.pdf

[26] Dimitriou, P., (2018) Artificial Neural Networks and Implementation, Thesis, Karlovasi: Aegean University: Department of Mathematics. Available at: https://hellanicus.lib.aegean.gr/bitstream/handle/11610/19590/

[27] Berg, I., (2023) Damped Harmonic Oscillator [online]. Available at: https://beltoforion.de/en/harmonic_oscillator/

[28] Svirin, A., (2023) Mass-Spring System [online]. Available at: https://math24.net/mass-spring-system.html

[29] Cao, Y., Fang, Z., Wu, Y., Zhou, D., & Gu, Q. (2019). Towards Understanding the Spectral Bias of Deep Learning. International Joint Conference on Artificial Intelligence. Available at : https://arxiv.org/abs/1912.01198

[30] Mustajab, A.H., Lyu, H., Rizvi, Z.H., & Wuttke, F. (2024). Physics-Informed Neural Networks for High-Frequency and Multi-Scale Problems using Transfer Learning. Available at : https://arxiv.org/abs/2401.02810