ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ
ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ VLSI

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Encapsulating measurement uncertainty *UPROP*: A Python library for Uncertainty Propagation

Στυλιανός Ε. Τσαγκαράκης

(Stylianos E. Tsagkarakis)

Επιβλέπων: Δημήτριος Ι. Σούντρης
Καθηγητής Ε.Μ.Π.

*

Αθήνα, Μάρτιος 2022

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ
ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ VLSI

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Encapsulating measurement uncertainty *UPROP*: A Python library for Uncertainty Propagation

Στυλιανός Ε. Τσαγκαράκης

(Stylianos E. Tsagkarakis)

Επιβλέπων: Δημήτριος Ι. Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 2022-03-04.

.....	.....	.....
Δημήτριος Σούντρης	Παναγιώτης Τσανάκας	Σωτήριος Ξύδης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Χ.Π.Α.

Αθήνα, Μάρτιος 2022

.....
Στυλιανός Ε. Τσαγκαράκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών
Ε.Μ.Π.

Περίληψη

Σε αυτή τη διπλωματική παρουσιάζουμε την βιβλιοθήκη *UPROP*. Η βιβλιοθήκη εισάγει τον τύπο `Uncertain` που αποτελεί ένα προγραμματιστικό μοντέλο για *περιγραφή, αποθήκευση και εκτέλεση πράξεων* πάνω σε αβέβαια δεδομένα. Υπερφορτώνουμε αριθμητικούς και συγκριτικούς τελεστές ούτως ώστε να πετύχουμε την μετάδοση και τη διάδοση του θορύβου εισόδου στην έξοδο. Παράλληλα αναπτύσουμε εφαρμογές, δοκιμάζοντας στην πράξη το προγραμματιστικό μοντέλο.

Ο κύριος στόχος μας είναι να παρέχουμε σε μη-ειδήμονες προγραμματιστές, μια φιλική στη χρήση διεπαφή με ένα ευρύ φάσμα δυνατοτήτων, που θα τους επιτρέπει να διαχειριστούν το θόρυβο χωρίς να τον αγνούν. Το κυριότερο πλεονέκτημα της βιβλιοθήκης *UPROP* πηγάζει από την ευλιξία και ευκολία στη χρήση της. Δοκιμάζουμε τη βιβλιοθήκη ανπτύσσοντας εφαρμογές που χρησιμοποιούν μετρήσεις αισθητήρων και μας επιτρέπουν να δείξουμε τα πλεονεκτήματα της χρήσης της βιλιοθήκης σε σχέση με τις συμβατικές μεθόδους. Η χρήση της οδηγεί σε βελτίωση στην ακρίβεια, σε σχέση με τις συμβατικές μεθόδους, που κυμαίνεται από $1.2\times$ έως $40\times$ με μέση τιμή $7\times$. Τέλος Συγκρίνουμε την υλοποίηση μας με βιβλιοθήκες και υλοποιήσεις υλικού τελευταίας τεχνολογίας και παραθέτουμε τις διαφορές τους.

Αν και αυτή η προσέγγιση διασφαλίζει ότι η πρόσβαση στο θόρυβο γίνεται με ένα ευχρηστό και μινιμαλιστικό τρόπο, υπάρχουν και μειονεκτήματά. Κατά τον υπολογισμό με τη βιβλιοθήκη *UPROP*, η επιλογή του μεγέθους αναπαράστασης των `Uncertain` αντικειμένων ενσωματώνει τον κλασικό συμβιβασμό μεταξύ ταχύτητας και ακρίβειας, πολύ μεγάλος και η χρήση της βιβλιοθήκης θα είναι πολύ αργή για πρακτική χρήση, πολύ μικρό και δε θα υπάρχει αρκετή ακρίβεια στους υπολογισμούς.

Λέξεις κλειδιά: θόρυβος, αβεβαιότητα, διάδοση θορύβου, Python, τυχαίες μεταβλητές, πιθανότητα

Abstract

In this work we present the *UPROP* Python library. A library for Uncertainty PROPagation, that allows developers to *describe*, *store*, *and execute* arithmetic operations on uncertain data. We overload arithmetic and comparison operators in order to achieve propagation of the input uncertainty to the output. To benchmark the library we develop real life applications with data derived from sensors.

The main goal of the *UPROP* Library is to provide non-expert programmers a basic, user-friendly interface with a wide range of operations, which allows them to reason about uncertainty without ignoring it. The main advantage of the `Uncertain` type originates from it's flexibility and minimalism. We develop GPS and sound recognition applications, using the *UPROP* library and uncertain sensor measurements. The usage of the library shows improved expressiveness and accuracy over the conventional usage of particle (one-sample) calculations. This approach leads to an average accuracy improvement, ranging from $1.2\times$ to $40\times$ with an average value of $7\times$. We further test the library using micro-benchmark comparisons with the state-of-the-art and list their differences.

While the implementation of the *UPROP* library ensures the accessibility and expressivity of the `Uncertain` type it also has the potential to make a practical implementation impossible. The representation size embodies the classic speed-accuracy trade-off. Too high and the `Uncertain` type will be too slow for practical use; too low and it will be too inaccurate to solve real problems.

Keywords: uncertainty, uncertainty propagation, random variables, python, probability

Ευχαριστίες

Πρωτίστως, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στον επιβλέποντα καθηγητή μου, τον καθηγητή Δημήτριο Σούντρη, Ε.Μ.Π., που μου παρείχε την δυνατότητα να συνεργαστώ με το εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων στο Εθνικό Μετσόβιο Πολυτεχνείο. Επιπλέον θα ήθελα να ευχαριστήσω τον καθηγητή Phillip Stanley-Marbell, στο Πανεπιστήμιο του Cambridge, που μου έδωσε την μου την ευκαιρία να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα. Ευχαριστώ επίσης τον μεταδιδάκτορα Βασίλειο Τσούτσουρα, στο Πανεπιστήμιο του Cambridge για την πολύτιμη βοήθεια και καθοδήγηση που μου παρείχε σε όλη τη διάρκεια εκπόνησης της εργασίας και που ήταν πάντα δίπλα μου για την επίλυση απορίων.

Ακόμα, θα ήθελα να ευχαριστήσω τους γονείς μου, Μάνο και Χριστίνα, που μου έδωσαν τις βάσεις για την απόκτηση γνώσεων και καλλιέργησαν μέσα μου την αγάπη για μάθηση, καθώς και τον αδερφό μου Μιχάλη που μαζί με τους γονείς μου με στηρίζουν τις επιλογές μου και μου δείχνουν εμπιστοσύνη σε ότι κάνω. Ανεκτίμητης σημασίας είναι και η στήριξη που έλαβα από τους φίλους μου στις δύσκολες και εύκολες στιγμές. Τέλος θα ήθελα να πω ένα τεράστιο ευχαριστήσω στην κοπέλα μου Χρύσα, που με υπομονή και αγάπη με υποστήριζε σε κάθε στιγμή των φοιτητικών μου χρόνων.

Acknowledgements

Firstly, I would like to express my gratitude to my supervisor, professor Dimitrios Soudris, N.T.U.A., who gave me the opportunity to work on this exciting project for my thesis with the Microprocessors and Digital Systems Laboratory in the National Technical University of Athens. I am forever grateful to professor Phillip Stanley-Marbell, University of Cambridge, for giving me the opportunity to work on such an interesting Thesis. Additionally, I would like to extend my deepest gratitude to post-doctoral researcher, Vasileios Tsoutsouras, University of Cambridge, who has been constantly supporting and guiding me on this journey, clearing my every doubt.

Furthermore, I would like to thank my parents, Christina and Manolis, for providing me the base for acquiring knowledge and assisted me in growing my love for learning. I would also like to thank my brother Michalis who, along with my parents support and trust my choices. Also to thank my friends who provided my with invaluable support throughout tough and easy times. Finally, a special thanks to my girl-friend Chrysa, who patiently and lovingly supported me in every moment throughout my university years.

Contents

List of Figures

List of Tables

Listings

Εκτεταμένη Περίληψη

1 Εισαγωγή

Αβεβαιότητα υπάρχει σε πολλές πτυχές της ζωής μας. Σε ένα μεγάλο βαθμό οι καθημερινές αποφάσεις και οποιοδήποτε ρίσκο που παίρνουμε συνοδεύεται από αυτήν. Βέβαια είμαστε τόσο συνηθισμένοι που συνήθως δεν την αντιλαμβανόμαστε, διότι έχουμε την ικανότητα να την επεξεργαζόμαστε υποσυνείδητα. Με αντίστοιχο τρόπο θα έπρεπε να λειτουργούν τα υπολογιστικά συστήματα. Αντιθέτως όμως, ενώ η ικανότητα των υπολογιστικών συστημάτων να αποθηκεύουν και να επεξεργάζονται δεδομένα βελτιώνεται συνεχώς, η ικανότητα τους να αντιμετώπιζουν την αβεβαιότητα και το θόρυβο στα δεδομένα έχει μείνει στάσιμη.

Στο παρελθόν σχεδιάζαμε τα συστήματα ελέγχου ως μεμονωμένα και κλειστά συστήματα υπό τον έλεγχο ενός κατασκευαστή ή σε κλειστά και προστατευόμενα περιβάλλοντα. Με την άνοδο της χρήσης του Διαδικτύου των πραγμάτων (IoT), των αυτόνομων αυτοκινήτων και του αυτοματισμού σε κάθε πτυχή της ζωής μας, εξαρτόμαστε όλο και περισσότερο από τις ακριβείς μετρήσεις αισθητήρων. Έτσι πρέπει και η ικανότητα των υπολογιστικών συστημάτων να συμβαδίσει με την αυξανόμενη ροή αβέβαιων μετρήσεων από αισθητήρες.

Όλες οι μετρήσεις έχουν έναν αναπόφευκτο βαθμό αβεβαιότητας (*θόρυβος μέτρησης*) και έτσι, ένα σύστημα που τις χρησιμοποιεί για να πάρει αποφάσεις θα πρέπει να λαμβάνει υπόψιν του και αυτήν την αλήθεια. Αυτό προκαλεί παραποίηση της αλήθειας σχετικά με την πραγματική τιμή της ποσότητας που θέλουμε να μετρήσουμε [1]. Επιπλέον, οι φυσικοί νόμοι που ενδεχομένως περιγράφουν την εξέλιξη κάποιου φυσικού συστήματος δεν ακολουθώνται πλήρως στην πραγματικότητα, λόγω της ύπαρξης *διεργασιακού θορύβου*. Γι΄ αυτούς τους λόγους έχουν αναπτυχθεί τεχνικές όπως η μετρίαση και το φιλτράρισμα, οι οποίες συνδυάζουν τις θορυβώθεις μετρήσεις από πολλαπλούς αισθητήρες μαζί με γνώση για τους φυσικούς νόμους που διέπουν ένα φυσικό σύστημα ώστε να καταφέρουν την εξομάλυνση του θορύβου.

Οι προγραμματιστές πρέπει να γνωρίζουν τα παραπάνω και να τα λαμβάνουν υπόψιν τους κατά τη διάρκεια ανάπτυξης του εκάστοτε συστήματος. Το να χρησιμοποιούν τα συγκεκριμένα δεδομένα *ως έχει* μπορεί να αποβεί μοιραίο και να καταλήξει σε εσφαλμένα αποτελέσματα με καταστροφικές συνέπειες.

Στην παρούσα διπλωματική παρουσιάζουμε την βιβλιοθήκη *UPROP* για τη γλώσσα προγραμματισμού Python. Η βιβλιοθήκη αυτή περιέχει τον τύπο δεδομένων: `Uncertain`. Ο τύπος `Uncertain` αποτελεί ένα προγραμματιστικό μοντέλο για *περιγραφή, αποθήκευση και εκτέλεση πράξεων* πάνω σε αβέβαια δεδομένα. Παράλληλα αναπτύσσουμε εφαρμογές, δοκιμάζοντας στην πράξη το παραπάνω προγραμματιστικό μοντέλο.

Όπως προαναφέραμε, οι αισθητήρες αποτελούν αναπόσπαστο κομμάτι των αυτοματοποιημένων διαδικασιών στη σύγχρονη τεχνολογία. Ένα χαρακτηριστικό παράδειγμα της σημασίας της αβεβαιότητας σε μετρήσεις αισθητήρων θα μπορούσε να είναι οι αισθητήτες μέτρησης απόστασης ενός αυτόνομου αυτοκινήτου. Στο τμήμα κώδικα 1 παρουσιάζουμε τον ψευδοκώδικα που χρησιμοποιούμε σε ανάλογα οχήματα προκειμένου να διατηρείται μια σταθερή απόσταση από το προπορευόμενο όχημα.

```
if distance < 10: # meters
    car.decrease_speed()
elif distance == 10:
    car.maintain_speed()
else:
    car.steady_speed()
```

Listing 1: Ψευδώς θετικά σε υποθέσεις.

Στο παραπάνω παράδειγμα ο προγραμματιστής αγνοεί την πιθανή αβεβαιότητα που κρύβουν τα δεδομένα. Έτσι υπάρχει η πιθανότητα ατυχήματος, λόγω καθυστερημένης πέδησης. Ακόμα και στην σπάνια περίπτωση που ο αισθητήρας δηλώσει απόσταση μεγαλύτερη των 10 μέτρων, τότε ο αυτόματος ελεγκτής θα επιταχύνει το όχημα και θα υπάρξει σύγκρουση. Αυτή η πληροφορία θα πρέπει να προτρέψει τους προγραμματιστές είτε να κάνουν στατιστική ανάλυση των δεδομένων εισόδου είτε να αντιμετωπίζουν τα δεδομένα αυτά ως αβέβαια.

Η βιβλιοθήκη που προτείνουμε στην παρούσα διπλωματική αποτελέι μια προσπάθεια να ενσωματώσουμε την αβεβαιότητα και τον θόρυβο σε μια πιθανοτική δομή δεδομένων. Έτσι θα μας δοθεί η δυνατότητα να προωθήσουμε τον ανάλογο θόρυβο στα αποτελέσματα πράξεων και κλήσεων συναρτήσεων πάνω στα εκάστοτε δεδομένα. Η σχεδίαση παρομοιων δομών αποτελέι μεγάλη πρόκληση αφού απαιτεί ισχυρά μαθηματικά θεμέλια καθώς και ε-

κτεταμένη βελτιστοποίηση ως προς τους χρόνους και τους πόρους που απαιτούνται για την εκτέλεση. Στο παρελθόν, η ανάπτυξή τους έχει απασχολήσει την ερευνητική κοινότητα τόσο σε μορφή γλωσσών προγραμματισμού όσο και σε επίπεδο υλικού.

Πιο συγκεκριμένα, σε επίπεδο υλικού, οι Τσούτσουρας κ.α. [2] παρουσίασαν την μικροαρχιτεκτονική Laplace η οποία χρησιμοποιεί κατανομές πιθανοτήτων σε αναπαράσταση μηχανής, ώστε να πραγματοποιούνται απρόσκοπτα πράξεις πάνω σε δεδομένα. Η μικροαρχιτεκτονική Laplace επεκτείνει το σύνολο εντολών RISC-V, ώστε να μπορεί ο χρήστης να εισάγει και να εξάγει πληροφορίες σχετικά με την κατανομή τυχαίων μεταβλητών. Η αναπαράσταση στη μνήμη αποτελείται από πλειάδες που χωρίζουν το πεδίο ορισμού σε N μέρη με ίση πιθανότητα και περιέχουν τη μέση τιμή, και την αντίστοιχη πιθανότητα που αναλογεί στο εκάστοτε μέρος.

Σε επίπεδο λογισμικού ο Bonrholt [3] υλοποίει τον αφαιρετικό τύπο δεδομένων *Uncertain*$<T>$. Ο Bornholt χρησιμοποιεί τις οριακές κατανομές τυχαίων μεταβλητών καθώς και συναρτήσεις δειγματοληψίας προκειμένου να αναπαραστήσει δεδομένα με θόρυβο. Σε συνδυασμό με ένα Bayesian Δίκτυο που σχηματίζεται από ένα κατευθυνόμενο ακυκλικό γράφο (DAG) μπορεί και εκτιμάει τις τιμές των μεταβλητών όταν χρειάζεται.

Η παρούσα εργασία συνεισφέρει στην επίλυση του προβλήματος του θορύβου στα δεδομένα, εμπλουτίζοντας την γλώσσα προγραμματισμού Python, παρέχοντας στους προγραμματιστές μια βιβλιοθήκη με την οποία μπορούν απρόσκοπτα, να υλοποιήσουν τις εφαρμογές τους.

2 Υλοποίηση

Οι υπάρχουσες αρχιτεκτονικές και γλώσσες προγραμματισμού δεν υποστηρίζουν εκ φύσεως την ύπαρξη θόρυβου σε δεδομένα που αποτελούνται από ένα και μόνο δείγμα. Οι υπολογιστές έχουν σχεδιαστεί με βάση την ιδέα ότι δεν υπάρχει αβεβαιότητα στις τιμές αυτές. Συνεπώς είτε δημιουργούνται εξατομικευμένες λύσεις από τους προγραμματιστές, είτε ο θόρυβος αυτός αγνοείται. Στην δεύτερη προσέγγιση είναι προφανές πώς ο προγραμματιστής βάζει τον χρήστη και σύστημά του σε κίνδυνο. Στην περίπτωση των εξατομικευμένων λύσεων, εάν ο προγραμματιστής δεν έχει διεπιστημονική γνώση ή δεν γίνει σωστός έλεγχος, τότε το σύστημα καθίσταται επικίνδυνο εν αγνοία των προγραμματιστών. Η άγνοια κινδύνου μπορεί να προκαλέσει σοβαρότερα προβλήματα.

Η κυριότερη μέθοδος που χρησιμοποιείται τόσο από την ερευνητική κοινότητα αλλά και από τη βιομηχανία για την προώθηση θορύβου στα αποτελέσματα είναι η μέθοδος Monte Carlo (Υποενότητα 4.1.1). Αποτελεί μια

αξιόπιστη μέθοδο, όμως είναι πολύ απαιτητική υπολογιστικά καθώς ο κάθε αλγόριθμος τρέχει επαναληπτικά πολλαπλάσιες φορές του μεγέθους της εισόδου.

Η **ιδέα** πίσω από την βιβλιοθήκη *UPROP* είναι να προσομοιώσουμε την μέθοδο Monte Carlo με μια ντετερμινιστική προσέγγιση, απαιτώντας λιγότερο χρόνο. Το σύστημα μας προωθεί τον θόρυβο γιατί θεωρούμε πως είναι ένας σημαντικός παράγοντας που μπορεί να παίξει καθοριστικό ρόλο στην αποτελεσματικότητα ενός συστήματος. Ο **στόχος** αυτής της βιβλιοθήκης είναι να επεκτείνει τη λειτουργικότητα της γλώσσας προγραμματισμού Python [4] ούτως ώστε να διευκολύνει τόσο την ανάπτυξη εφαρμογών που βασίζονται σε εμπειρικά δεδομένα (μετρήσεις από αισθητήρες), όσο και γνωστές παραμετρικές κατανομές.

Οι κύριοι στόχοι της βιβλιοθήκης *UPROP* είναι να αποτελεί ένα ① **μινιμαλιστικό** και ② **ευέλικτο** εργαλείο:

- Θα πρέπει να επιτρέπει στους προγραμματιστές Python να το υιοθετήσουν στον κώδικα τους, αντικαθιστώντας, ει δυνατόν τους συμβατικούς τύπους δεδομένων, όπως `int, double`.

- Ένα αντικείμενο τύπου `Uncertain` θα πρέπει να μπορεί να αναπαραστήσει μια τυχαία μεταβλητή.

- Η κλάση `Uncertain` θα πρέπει επίσης να **υπερφορτώνει, την πλειοψηφία των τελεστών της βασικής κλάσης της Python** ούτως ώστε οι προγραμματιστές να μπορούν απρόσκοπτα να χρησιμοποιήσουν τις ίδιες εκφράσεις που θα χρησιμοποιούσαν με τις τιμές ενός δείγματος, ενώ παράλληλα θα προωθείται ο θόρυβος μέσα από τελεστές και συναρτήσεις. Στο απλό παράδειγμα: `Z = A+B` όπου τα `A,B` αποτελούν δύο `Uncertain` αντικείμενα, θα καλείται ο `+` τελεστής ο οποίος θα επιστρέφει ένα νέο `Uncertain` αντικείμενο `Z`, συμπεριλαμβανομένου του συνδυασμού θορύβου που περιείχε τόσο το `A` όσο και το `B`.

Το κύριο συμπέρασμα από την υλοποίηση της *UPROP* βιβλιοθήκης, είναι ότι προκειμένου να χρησιμοποιηθεί ο τύπος `Uncertain` ως τύπος πρώτης τάξης, όπως οι τύποι `int, float`, etc πρέπει να υπάρχει κάποιος συμβιβασμός όσον αφορά την ταχύτητα εκτέλεσης και την χρήση μνήμης στο εκάστοτε υπολογιστικό σύστημα, με άμεσο κέρδος την ακρίβεια στους υπολογισμούς. Επιπλέον η πιθανοτική φύση των τυχαίων μεταβλητών περιορίζει το πλήθος των πράξεων και συναρτήσεων που μπορούν να εκτελεστούν με αυτές, καθώς πρέπει να γίνει η μετάβαση από το ντετερμινιστικό στο πιθανοτικό πεδίο ορισμού.

2.1 Αναπαράσταση κατανομών

Κατά γενική ομολογία, η πιο ακριβής αναπαράσταση μια κατανομής πιθανοτήτων στη μνήμη είναι να αποθηκεύονται όλα τα δείγματα. Όμως, πράττοντας έτσι ο υπολογιστικός φόρτος του συστήματος επιβαρύνεται σημαντικά. Συνεπώς η εύρεση ενός τρόπου αναπαράστασης που απαιτεί λιγότερες πράξεις και διατηρεί την ακρίβεια στους υπολογισμούς είναι μονόδρομος.

Οι Τσούτσουρας κ.α. [2] προτείνουν τον τρόπο αναπαράστασης τυχαίων μεταβλητών ως συνδυασμό συναρτήσεων Ντιράκ.

Definition 2.1 ***Dirac mixture representation****: Έστω $\delta(x)$ η συνάρτηση δέλτα (Dirac) -- ένας μοναδιαίος παλμός στη θέση x. Δεδομένης μιας τιμής $x_0 \in \mathbb{R}$, θεωρούμε ως την συνάρτηση: $\delta(x - x_0)$ ως την συνάρτηση μάζας πιθανότητας και την αποκαλούμε* ***τιμη ενός δείγματος****. Χρησιμοποιώντας αυτό τον ορισμό μπορούμε να μετασχηματίσουμε έναν πίνακα από δείγματα $x_1, x_2, ..., x_M$ σε μια συνάρτηση μάζας πιθανότητας χρησιμοποιώντας ένα σταθμισμένο άθροισμα.*

$$f_X(x) = \sum_{n=1}^{M} p_n \delta(x - x_n), \ \textit{όπου} \ p_n \in [0, 1], \ \sum_{n=1}^{M} p_n = 1 \tag{2.1}$$

Αν κάθε Ντιράκ στον συνδυασμό ισαπέχει με την επόμενη και την προηγούμενη, τότε αυτή η αναπαράσταση παίρνει τη μορφή των ιστογραμμάτων συχνοτήτων.

Είναι ξεκάθαρο πως το μέγεθος μνήμης που καταλαμβάνει ένα `Uncertain` αντικείμενο καθορίζεται από το πλήθος των Ντιράκ (ή στηλών στην περίπτωση του ιστογράμματος) που έχουμε χωρίσει το πεδίο ορισμού της εκάστοτε τυχαίας μετβλητής. Ένα σημαντικό συμπέρασμα που εξάγαμε κατά την πειραματική αξιολόγηση 3 είναι πως μπορεί να υπάρξει κέρδος από την μείωση του μεγέθους ενός αντικειμένου, χωρίς να υπάρξει απώλεια στην ακρίβεια υπολογισμών.

Αξιοσημείωτο είναι το γεγονός πως η χρησιμοποίηση ιστογραμμάτων σε σχέση με τις **τιμές ενός δείγματος** προσθέτει πολυπλοκότητα στις διαδικασίες με αντικείμενα τύπου `Uncertain`, σε αντίθεση με τη χρήση τιμών ενός δείγματος. Πιο συγκεκριμένα η μετατροπή των δειγμάτων σε ιστογράμματα απαιτεί ένα αλγόριθμο πολυπλοκότητας χρόνου, $\mathcal{O}(N \cdot k)$ για να ταξινομήσουμε τα N δείγματα σε k στήλες και να υπολογίσουμε τις συχνότητες. Η πολυπλοκότητα χώρου εξαρτάται από τον αριθμό των στηλών και είναι: $\mathcal{O}(k)$. Στην περίπτωση που επιλέγαμε να αποθηκεύσουμε τα N δείγματα ως έχει, η πολυπλοκότητα χώρου θα αυξανόταν σε $\mathcal{O}(N)$.

2.2 Μετάδοση θορύβου

Η βασική αρχή του τύπου `Uncertain` είναι να μπορεί να ενσωματώνει το θόρυβο -- ακόμα και τα πιο ακραία ενδεχόμενα, και να τον προωθεί/μεταδίδει μέσα από τελεστές και συναρτήσεις που χρησιμοποιούν `Uncertain` αντικείμενα. Έστω `Uncertain` αντικείμενα που αναπαριστούν τις τυχαίες μεταβλητές $X \sim U\,(0,6)$ και $Y \sim U\,(0,6)$.[1]Το γινόμενό τους $Z = X \cdot Y$ είναι επίσης `Uncertain` και αναπαριστά μια τυχαία μεταβλήτη. Το παράδειγμα αυτό φαίνεται στο τμήμα κώδικα 2.

```
X = Uncertain([i for i in range(0,100)])
X = Uncertain([i for i in range(100,200)])
Z = X + Y
```

Listing 2: Γινόμενο δύο `Uncertain` αντικειμένων.

Στο παραπάνω τμήμα κώδικα χρησιμοποιουμε κυκλική συνέλιξη των δύο τυχαίων μεταβλητών για να υπολογίσουμε όλες τις πιθανές τιμές της μεταβλητής Z. Η κυκλική συνέλιξη ορίζεται ως:

$$(f * g)(z) = \int_{-\infty}^{\infty} f(z-t)g(t)dt = \int_{-\infty}^{\infty} f(t)g(z-t)dt \tag{2.2}$$

Στο παρακάτω τμήμα κώδικα 3 δείχνουμε την υλοποίηση της κυκλικής συνέλιξης στην βιβλιοθήκη *UPROP*.

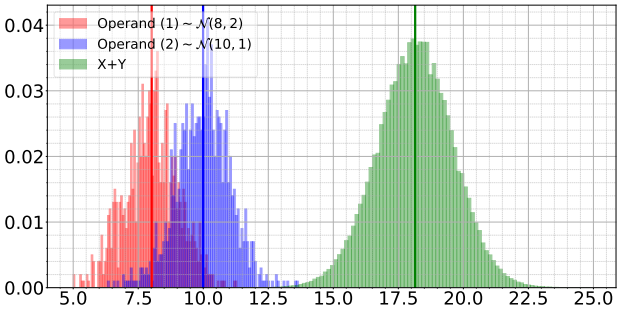
```
def addRandomVariables(X1, X2):
    """Calculate the circular convolution of X1 with X2."""
    dst_hist  = []
    dst_medians = []
    for i in range(0,len(X1.hist)):
        for j in range(0,len(X2.hist)):
            dst_hist.append(X1.hist[i]*X2.hist[j])
            dst_medians.append(X1.median[i] + X2.median[j])

    dstVar = createObjectFromOperation(dst_hist, dst_medians)
    return dstVar
```

Listing 3: Υλοποίηση της κυκλικής συνέλιξης μεταξύ δύο `Uncertain` αντικειμένων (τυχαίων μεταβλητών). Η παραγωγή του νέου `Uncertain` αντικειμένου γίνεται με βάση τον ορισμό 2.1 Στον πίνακα `dst_hist` αποθηκεύονται οι πιθανότητες p_n των τιμών, στην αντίστοιχη θέση, του πίνακα `dst_medians` στον οποίο αποθηκεύονται οι τιμές του πεδίου ορισμού $(x - x_n)$.

[1]Το U(a,b) αναπαριστά μια ομοιόμορφη κατανομή στο διάστημα $[a, b]$

2.3 Υπερφορτωμένοι αριθμητικοί τελεστές

Οι υπερφορτωμένοι αριθμητικοί τελεστές που υποστηρίζει η βιβλιοθήκη *UPROP* παρουσιάζονται στον πίνακα 1. Όπως προαναφέραμε χρησιμοποιούν την κυκλική συνέλιξη (ορισμός 2.2) προκειμένου να υπολογιστούν όλες οι πιθανές τιμές της παραγόμενης μεταβλητής. Παρουσιάζουμε ενα χαρακτηριστικό παράδειγμα στην Εικόνα 1 Ο αριθμός στηλών (bins) του παραγόμενου `Uncertain` αντικείμενου μετά από την κλήση ενός τελεστή είναι ο μεγαλύτερος αριθμός ανάμεσα στον αριθμό στηλών (bins) των αντικειμένων που το παρήγαγαν.



Σχήμα 1: Πρόσθεση δύο `Uncertain` μεταβλητών.

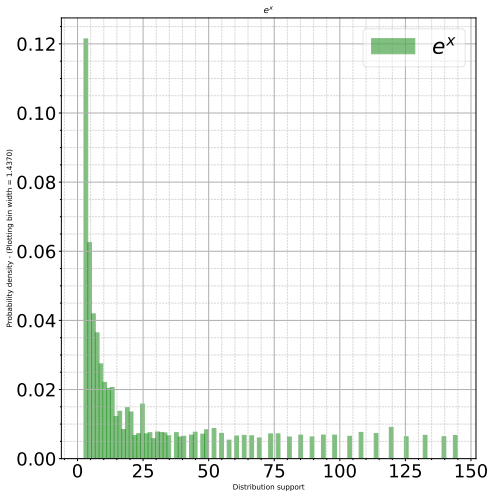
Μέθοδος	**Χρήση**	`type(a)`	`type(b)`
`__neg__`	`-a`	`Uncertain`	`-`
`__add__`	`a+b`	`Uncertain`	`Uncertain | Number`
`__sub__`	`a-b`	`Uncertain`	`Uncertain | Number`
`__mul__`	`a*b`	`Uncertain`	`Uncertain | Number`
`__truediv__`	`a/b`	`Uncertain`	`Uncertain | Number`
`__pow__`	`a**b`	`Uncertain`	`Uncertain | Number`
`__mod__`	`a%b`	`Uncertain`	`Uncertain | Number`
`__divmod__`	`divmod(a,b)`	`Uncertain`	`Uncertain | Number`

Πίνακας 1: Υπερφορτωμένοι αριθμητικοί τελεστές.

2.4 Χρήση της συνάρτησης `map` σε `Uncertain` αντικείμενα

Στη βιβλιοθήκη *UPROP* χρησιμοποιούμε αυτήν τη συνάρτηση προς όφελός μας ώστε να κάνουμε `map` τα `Uncertain` αντικείμενα στην εκάστοτε συνάρτηση εισόδου. Αυτή η λειουργία υλοποιείται στην συνάρτηση `u_map()`. Το αντικείμενο που επιστρέφουμε είναι τύπου `Uncertain`. Το τμήμα κώδικα 4 δείχνει την υλοποίηση της μεθόδου `u_map()`. Στην Εικόνα 2 παρουσιάζουμε το αποτέλεσμα της μεθόδου `u_map()` με την συνάρτηση e^x.

(α′) Το `Uncertain` αντικείμενο εισόδου: $X \sim U(0,5)$



(β′) Το `Uncertain` αντικείμενο εξόδου: e^x το οποίο παράχθηκε από τη μέθοδο `u_map`.

Σχήμα 2: Αποτελέσματα της `u_map()` μεθόδου με είσοδο $X \sim U(0,5)$ και συνάρτηση εισόδου την εκθετική συνάρτηση: e^x.

```
1  def u_map(obj, func: Callable, **kwargs) -> Uncertain:
2      """Map function <func> to the <obj> Uncertain object"""
3      new_medians = map(lambda x: func(x, **kwargs), obj._medians)
4      """
5          Since the ordering of the new mapped values has not
6          changed, compared with the original medians the histogram
7          values in the corresponding positions represent the
8          probablity of each new median.
9      """
10     dstVar = createObjectFromOperation(obj.hist, new_medians)
11     return dstVar
```

Listing 4: Υλοποίηση της συνάρτησης `u_map` η οποία αντιστοιχεί `Uncertain` αντικείμενα σε συναρτήσεις.

2.5 Εξαγωγή πληροφοριών θορύβου

Προκειμένου να εκμεταλλευτούμε το σύνολο των δυνατοτήτων του τύπου `Uncertain` πρέπει να δώσουμε πρόσβαση στο χρήστη σχετικά με την αβεβαιότητα και το θόρυβο του κάθε αντικείμενου. Πράττοντας έτσι, δίνουμε τη γνώση και την πληροφορία στο χρήστη προκειμένου να λάβει μια τεκμηριωμένη απόφαση με τη διάδοση της αβεβαιότητας από την είσοδο. Παρουσιάζουμε τις διαθέσιμες πληροφορίες της κατανομής ενός `Uncertain` αντικειμένου στον πίνακα 2.

Μέθοδος	**Σκοπός**	**Τύπος `val`**
`mean`	$\mu = \mathbb{E}[X]$	-
`variance`	$\mathbb{E}[(X-\mu)^2]$	-
`min()`	X_{min}	-
`max()`	X_{max}	-
`mode()`	$x_i : P(X = x_i) \geq P(X = x_j)\ i \neq j$	-
`NthMoment(N)`	$\sigma_N = \mathbb{E}[(X-\mu)^N]$	-
`prob(val)`	$P(X = \texttt{val})$	`Number`
`CDF(val)`	$F_X(x = \texttt{val})$	`Number`
`QF(val)`	$F_X^{-1}(x = \texttt{val})$	`Number`
`probabilityEQ(val)`	$P(X = \texttt{val})$	`Number`
`probabilityNE(val)`	$P(X \neq \texttt{val})$	`Number`
`probabilityLT(val)`	$P(X < \texttt{val})$	`Number`
`probabilityLE(val)`	$P(X \leq \texttt{val})$	`Number`
`probabilityGT(val)`	$P(X > \texttt{val})$	`Number`
`probabilityGE(val)`	$P(X \geq \texttt{val})$	`Number`

Πίνακας 2: Διαθέσιμες μέθοδοι των `Uncertain` αντικειμένων για την εξαγωγή πληροφορίας περί θορύβου και αβεβαιότητας.

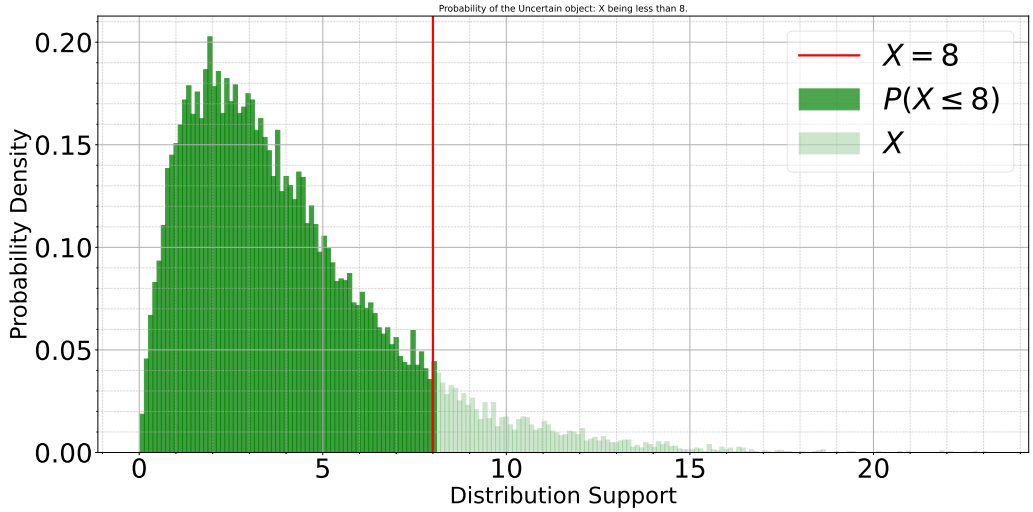
Αν και οι κεντρικές ροπές δεν ορίζουν πάντα μοναδικά κάποια κατανομή πιθανότητας [5], ο προγραμματιστής μπορει να αποκτήσει ένα σημαντικό όγκο πληροφοριών υπολογίζοντας μια σειρά από αυτές (π.χ. 1: μέση τιμή, 2: διακύμανση, 3: λοξότητα (ασυμμετρία), 4: κυρτότητα, κ.λπ.). Για παράδειγμα, η χρήση αυτών των πληροφοριών θα μπορούσε να παρέχει συνέπεια στις αποφάσεις για τη διατήρηση της σταθερότητας σε ένα αυτόματο σύστημα ελέγχου.

2.6 Τελεστές Σύγκρισης

Συμπληρωματικα με τη μέθοδο `u_map()`, η απαραίτητη λειτουργικότητα που οφείλει να έχει η βιβλιοθήκη ώστε να προσομοιώνει τους αριθμητικούς τύπους δεδομένων, είναι οι τελεστές σύγκρισης: $<, \leq, =, \neq, \geq, >$. Διακρίνουμε δύο κατηγορίες συγκρίσεων: (i) συγκριση ενός `Uncertain` αντικειμένου με ένα σταθερό αριθμό, και (ii) συγκριση δύο `Uncertain` αντικειμένων.

Όσον αφορά την πρώτη περίπτωση: αν συγκρίνουμε την μέση τιμή: $\mathbb{E}[X]$ --- ένα μόνο δείγμα που προέρχεται από την κατανομή, με μια αριθμητική σταθερά `value`: $\mathbb{E}[X] \lessgtr \texttt{value}$, το αποτέλεσμα μπορεί να είναι παραπλανητικό, που οδηγεί σε ψευδώς θετικά αποτελέσματα [3]. Αντίθετως, στην υλοποίηση μας συγκρίνουμε το τμήμα της πιθανότητας του `Uncertain` α-

ντικειμένου που βρίσκεται στα αριστερά ή στα δεξιά της τιμής `value`. Το σχήμα 3 οπτικοποιεί αυτή σύγκριση: $\text{Uncertain}(X) \leq 8 \Rightarrow P(X \leq 8)$.



Σχήμα 3: Η πιθανότητα ενός `Uncertain` αντικειμένου να είναι $X \leq$ `value` είναι το άθροισμα των πιθανοτήτων των στηλών των οποίων το μέσο είναι $\leq$ `value`. Σε αυτή την περίπτωση, η τιμή αναπαρίσταται με το σκούρο πράσινο χρώμα.

Όσον αφορά τη περίπτωση σύγκρισης δύο `Uncertain` αντικειμένων:

1. οι τελεστές $=$ και $\neq$ υλοποιούνται με τη χρήση του Kolmogorov–Smirnov ελέγχου, ενώ

2. για τους οι τελεστές $<, \leq, \geq, >$ προτείνουμε δύο μεθόδους:

   (α′) **Στοχαστική Κυριαρχία** και

   (β′) **Διαφορά Τυχαίων Μεταβλητών**.

Ο χρήστης μπορεί να διαλέξει τη μέθοδο που επιθυμεί είτε κατά την αρχικοποίηση των μεταβλητών είτε κατά την εκτέλεση.

Μέθοδος 1: Στοχαστική Κυριαρχία

Όπως αναφέρουμε στην Υποενότητα 2.3.1, η *Στοχαστική Κυριαρχία* συγκρίνει τις Αθροιστικές Συναρτήσεις Κατανομής των τυχαίων μεταβλητών. Στο τμήμα κώδικα παρουσιάζουμε τον ψευδοκώδικα υλοποίησης της:

```python
def stochastic_dominance_le(X,Y):
    for i in joint_distribution_support(X,Y):
        if X.CDF(i) >= Y.CDF(i):
            # partial ordering remains the same
            continue
        else:
            # found a value for which ordering reverses
            return False
    return True
```

Listing 5: Ψευδοκώδικας για την υλοποίηση της Στοχαστικής Κυριαρχίας για τον τελεστή $\leq$.

Μέθοδος 2: Διαφορά τυχαίων μεταβλητών

Η μέθοδος σύγκρισης *Διαφορά δύο μεταβλητών* είναι ισοδύναμη με τη σύγκριση ενός `Uncertain` αντικειμένου με το σταθερό αριθμό 0. Η ιδέα πίσω από την υλοποίηση πηγάζει από την αντιμεταθετική ιδιότητα των συγκριτικών τελεστών $X > Y \Rightarrow X - Y > 0$. Στο τμήμα κώδικα 6 παρουσιάζουμε ψευδοκώδικα για την υλοποίηση της.

```python
def difference_rv_gt(X: Uncertain, Y: Uncertain):
    Z : Uncertain = X-Y
    max_threshold = max(X.threshold, Y.threshold)
    return Z.probabilityGT(0) > max_threshold
```

Listing 6: Υλοποίηση της μεθόδου *Διαφορά τυχαίων μεταβλήτών*

3 Πειραματική αξιολόγηση

Στην ενότητα 2 παρουσιάσαμε την υλοποίηση της βιβλιοθήκης *UPROP*, ακολουθώντας συγκεκριμένες σχεδιαστικές αρχές. Σε αυτή την υποενότητα αξιολογούμε πειραματικά τη βιβλιοθήκη σε εφαρμογές. Αρχικά παρουσιάζουμε τις απαραίτητες πληροφορίες και μετρικές για την αξιολόγηση, που απαιτείται για την κατανόηση του υπόλοιπου κεφαλαίου. Στη συνέχεια πραγματοποιούμε πειράματα στους υπερφορτωμένους αριθμητικούς και συγκριτικούς τελεστές. Επεκτείνουμε τις δοκιμές μας με δύο εφαρμογές που χειρίζονται πειραματικές μετρήσεις: (i) εκτίμηση ταχύτητας από μετρήσεις συντεταγμένων GPS (με θόρυβο) και (ii) ένας αλγόριθμος ανίχνευσης ήχου από χειροκρότημα. Τέλος, συγκρίνουμε την βιβλιοθήκη *UPROP* με state-of-the-art πλατφόρμες και εργαλεία και βαθμολογούμε την απόδοσή της.

3.1 Κανονικοποιημένη απόσταση Wasserstein

Στην ενότητα 2.3 παρουσιάζουμε την απόσταση Wasserstein, η οποία είναι μια μέθοδος για την αξιολόγηση της απόκλισης μεταξύ δύο πολυδιάστατων κατανομών. Σε αυτή την υποενότητα, παρουσιάζουμε μια εναλλακτική χρήση της απόστασης Wasserstein για σύγκριση δύο εργαλείων (frameworks/platforms) που έχουν έξοδο κατανομές πιθανοτήτων. Όπως αναφέρουμε στην Υποενότητα 2.3.3, όταν υπολογίζουμε την απόσταση W_1 μεταξύ δύο κατανομών, υπολογίζουμε την περιοχή μεταξύ της καμπύλης των CDF. Έτσι, η υπολογισμένη τιμή συνδέεται άμεσα με το πεδίο ορισμού R_X κάθε κατανομής. Στην περίπτωσή μας, επιθυμούμε να έχουμε μια ενιαία μετρική για το σύνολο των δοκιμών μας. Για να γίνει αυτό, πρέπει να αποσυνδέσουμε την απόσταση από το πεδίο ορισμού. Επομένως, εισάγουμε την κανονικοποιημένη απόσταση Wasserstein:

Definition 3.1 *Η κανονικοποιημένη απόσταση Wasserstein είναι ο λόγος δύο* W_1 *αποστάσεων*

$$\mathrm{NWD} = \frac{W_1\left(F_a, F_{ref}\right)}{W_1\left(F_b, F_{ref}\right)} = \frac{\int_{\mathbb{R}} \left|F_a(x) - F_{ref}(x)\right| dx}{\int_{\mathbb{R}} \left|F_b(x) - F_{ref}(x)\right| dx} \tag{3.1}$$

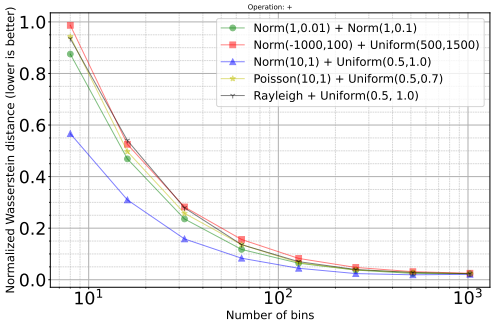
Η NWD παίρνει τιμές στην περιοχή: $[0, \infty)$ με τιμές πιο κοντά στο 0 να σημαίνει ότι η F_a έχει μικρότερη απόσταση Wasserstein με την F_{ref} από ότι η F_b έχει με την F_{ref}. Η NWD μας δίνει τη δυνατότητα να συγκρίνουμε δύο συστημάτα που έχουν έξοδο κατανομές πιθανοτήτων. Έτσι συγκρίνουμε την έξοδο της *UPROP* με τη τιμή της εξόδου στην περίπτωση της χρήσης ενός δείγματος ή με οποιοδήποτε άλλο σύστημα.

Σε αυτή τη διπλωματική, χρησιμοποιούμε την **αντίστροφη κανονικοποιημένη απόσταση Wasserstein** (NWD^{-1}) για να υποδηλώσουμε το κέρδος στην ακρίβεια των υπολογισμών της βιβλιοθήκης *UPROP*, έναντι των υπολογισμών με ένα δείγμα, χρησιμοποιώντας την προσομοίωση Monte Carlo ως την κατανομή αναφοράς. Επιλέξαμε την προσομοίωση Monte Carlo διότι αποτελεί την καθιερωμένη και κοινά αποδεκτή μέθοδο για τη προώθηση του θορύβου από την είσοδο στην έξοδο. Αυτό ισχύει τόσο στην ερευνητική κοινότητα όσο και στη βιομηχανία.

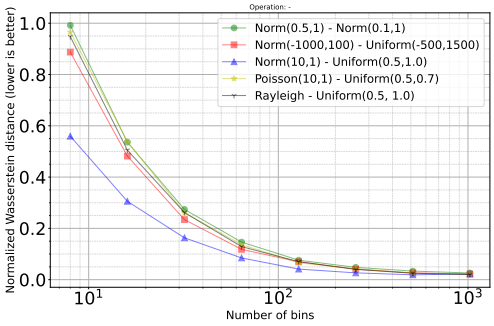
3.2 Αριθμητικοί Τελεστές

Για να ελέγξουμε τους αριθμητικούς τελεστές εκτελέσαμε μια σειρά απλών υπολογισμών μεταξύ Τυχαίων μεταβλητών που αντιπροσωπεύουν συγκεκριμένες κατανομές. Για παράδειγμα **προσθέτουμε, αφαιρούμε, πολλαπλασιάζουμε και διαιρούμε** τις δύο Τυχαίες μεταβλητές: $X_1 \sim \mathcal{N}(10, 1)$

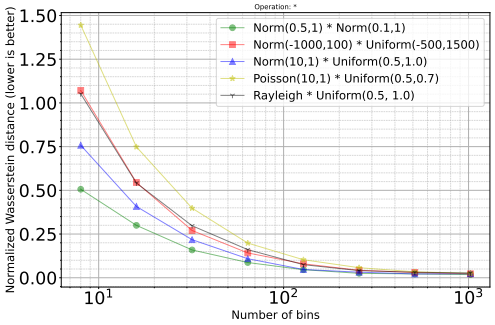
και $X_2 \sim \mathcal{U}(0.5, 1.0)$ χρησιμοποιώντας την βιβλιοθήκη *UPROP*. Στη συνέχεια εκτελούμε τους ίδιους υπολογισμούς με την προσομοίωση Monte Carlo και υπολογίζουμε την κανονικοποιημένη απόσταση Wasserstein μεταξύ τους.



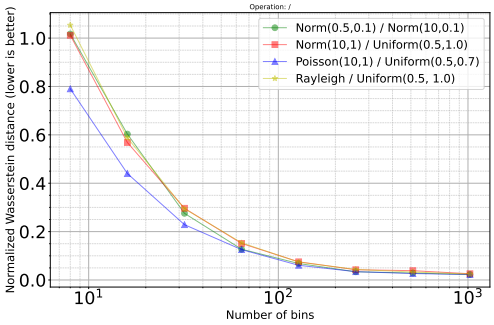
(α′) Κανονικοποιημένη απόσταση Wasserstein του τελεστή πρόσθεσης (+).



(β′) Κανονικοποιημένη απόσταση Wasserstein του τελεστή αφαίρεσης (-).



(γ′) Κανονικοποιημένη απόσταση Wasserstein του τελεστή πολλαπλασιασμού (*).



(δ′) Κανονικοποιημένη απόσταση Wasserstein του τελεστή διαίρεσης (/).

Σχήμα 4: Αυτό το σχήμα δείχνει την κανονικοποιημένη απόσταση Wasserstein (άξονας y) μεταξύ της κατανομής εξόδου της βιβλιοθήκης *UPROP* και του υπολογισμού με τη χρήση ενός δείγματος. Μετράμε αυτή την απόσταση από την προσομοίωση Monte Carlo. Χαμηλότερη τιμή σημαίνει καλύτερη απόδοση. Παρατηρούμε ότι όσο μεγαλύτερος είναι ο αριθμός των στηλών, τόσο μεγαλύτερη είναι η ακρίβεια υπολογισμών.

Η Εικόνα 4 εμφανίζει οπτικοποιημένα τα αποτελέσματα. Το **ελάχιστο κέρδος ακρίβειας** σε σχέση με τον υπολογισμό του δείγματος σωματιδίων, **είναι** $1,03\times$ για 8 στήλες (bins) και το **το μέγιστο είναι** $44,62\times$ για 1024 στήλες (bins).

3.3 Συγκριτικοί Τελεστές

Το δεύτερο σημαντικό ορόσημο που πρέπει να πετύχει η βιβλιοθήκη *UPROP* είναι να παρέχει στους προγραμματιστές τη δυνατότητα να εξάγουν πληροφορίες και να θέτουν ερωτήσεις σχετικά με τον θόρυβο `Uncertain` αντικει-

μένων. Αυτό επιτρέπει στο χρήστη να λάβει μια τεκμηριωμένη απόφαση με βάση το θόρυβο του συστήματος. Όπως αναφέρουμε στην ενότητα 2.6, υλοποιούμε τον τελεστή ισότητας(=) και ανισότητας ($\neq$ ή !=) χρησιμοποιώντας τη μέθοδο Kolmogorov–Smirnov [6]. Για τους υπόλοιπους τελεστές παρέχουμε στον προγραμματιστή δύο μεθόδους από τις οποίες μπορεί να επιλέξει:

① **Στοχαστική κυριαρχία** και

② ***Διαφορά τυχαίων μεταβλητών***.

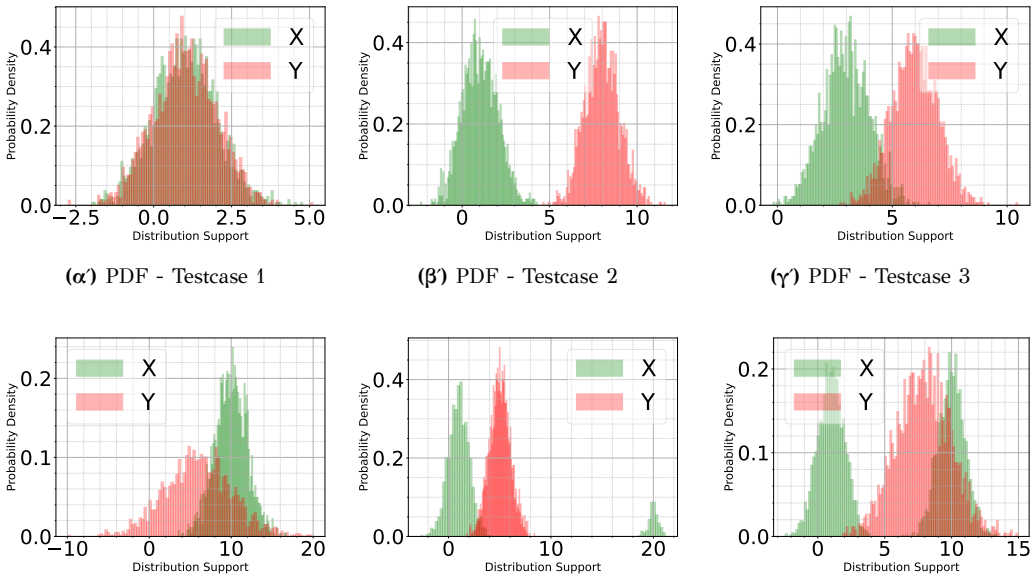
Για να δοκιμάσουμε εκτενώς κάθε μέθοδο, παρέχουμε ένα σύνολο από Τυχαίες Μεταβλητές που αντιπροσωπεύουν συγκεκριμένες κατανομές, τις οποίες συγκρίνουμε μεταξύ τους. Παρουσιάζουμε το γράφημα των συναρτήσεων στην Εικόνα 5.

	$X > Y$	$X \geq Y$	$X == Y$	$X \neq Y$	$X < Y$	$X \leq Y$
Testcase 1	False	***True***	True	False	False	***True***
Testcase 2	False	False	False	True	True	True
Testcase 3	False	False	False	True	True	True
Testcase 4	False	False	False	True	False	False
Testcase 5	False	False	False	True	***False***	***False***
Testcase 6	False	False	False	True	False	False

Πίνακας 3: Αποτελέσματα τελεστών υπό όρους για τα testcases 1-6, χρησιμοποιώντας τη μέθοδο *Διαφορά Τυχαίων Μεταβλητών*. Οι τιμές με τον αστερίσκο * σημειώνουν το διαφορετικό αποτέλεσμα με την μεθόδου *Στοχαστικής Κυριαρχίας*.

	$X > Y$	$X \geq Y$	$X == Y$	$X \neq Y$	$X < Y$	$X \leq Y$
Testcase 1	False	***False***	True	False	False	***False***
Testcase 2	False	False	False	True	True	True
Testcase 3	False	False	False	True	True	True
Testcase 4	False	False	False	True	False	False
Testcase 5	False	False	False	True	***True***	***True***
Testcase 6	False	False	False	True	False	False

Πίνακας 4: Αποτελέσματα τελεστών υπό όρους για τα testcases 1-6, χρησιμοποιώντας τη μέθοδο *Στοχαστική Κυριαρχία*. Οι τιμές με τον αστερίσκο * σημειώνουν το διαφορετικό αποτέλεσμα με την μεθόδου *Διαφορά Τυχαίων Μεταβλητών*.



(α′) PDF - Testcase 1 **(β′)** PDF - Testcase 2 **(γ′)** PDF - Testcase 3

(δ′) PDF - Testcase 4 **(ε′)** PDF - Testcase 5 **(ς′)** PDF - Testcase 6

Σχήμα 5: Probability Density Functions of two random variables used to test conditional operators and infer a partial order.

3.4 Υπολογισμός ταχύτητας από μετρήσεις GPS με θόρυβο

Για να δείξουμε τη δυνατότητα χρήσης του `Uncertain` τύπου για υπολογισμούς, υλοποιούμε μια εφαρμογή χρησιμοποιώντας το Παγκόσμιο Σύστημα Εντοπισμού Θέσης (GPS). Το παράδειγμα στο τμήμα κώδικα 1 μας παρακινεί να υλοποιήσουμε μια εφαρμογή που υπολογίζει την ταχύτητα χρησιμοποιώντας αβέβαιες μετρήσεις GPS από το smartphone του χρήστη. Εστιάζουμε στον τομέα του GPS επειδή οι εφαρμογές GPS είναι απλές στην κατανόηση, ευρέως χρησιμοποιούμενες στα κινητά και συνήθως αντιμετωπίζουν θόρυβο στις μετρήσεις. Για την λήψη των μετρήσεων χρησιμοποιήσαμε ένα κινητό τηλέφωνο και την εφαρμογή Phyphox [7] Για τον υπολογισμό της ταχύτητας χρησιμοποιούμε την μέθοδο Haversine που υπολογίζει την απόσταση δύο των συντεταγμένων GPS:

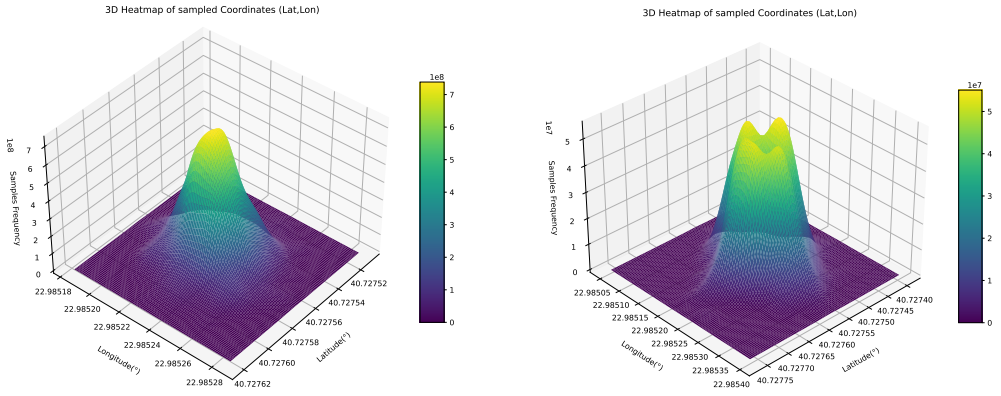
$$a = sin^2\left(\frac{\Delta\phi}{2}\right) + cos\phi_1 \cdot cos\phi_2 \cdot sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{3.2}$$

$$c = 2 \cdot \arctan2(\sqrt{a}, \sqrt{(1-a)}) \tag{3.3}$$

$$D = R \cdot c \tag{3.4}$$

Μετατρέποντας τις συντεταγμένες σε `Uncertain`

Για να υλοποιήσουμε την εφαρμογή υπολογισμού ταχύτητας πρέπει να μετατρέψουμε τα δεδομένα μας σε τύπο `Uncertain`. Το API των κινητών τηλεφώνων (Android και iPhone) παρέχει για κάθε μέτρηση την οριζόντια ακρίβεια της θέσης (σε μέτρα). Με βάση αυτό, δειγματολειπτούμε μια δισδιάστατη Γκαουσιανή κατανομή όπως δείχνουμε στην Εικόνα 6α′ και μια κατανομή Rayleigh όπως δείχνουμε στην Εικόνα 6β′.



(α′) Με βάση μια Γκαουσιανή κατανομή. **(β′)** Με βάση μια Rayleigh κατανομή.

Σχήμα 6: Δειγματοληψία γύρω από τις συντεταγμένες μέτρησης ώστε να προσθέσουμε τεχνητό θόρυβο στις μετρήσεις μας.

Performance Gain			
Test id	**Min**	**Mean**	**Max**
Test 01	2.07	**2.71**	3.91
Test 02	1.77	**2.34**	5.04
Test 03	17.01	**34.73**	52.79
Test 04	2.0	**7.79**	44.41
Test 05	4.24	**15.08**	79.06
Test 06	2.36	**5.57**	29.06

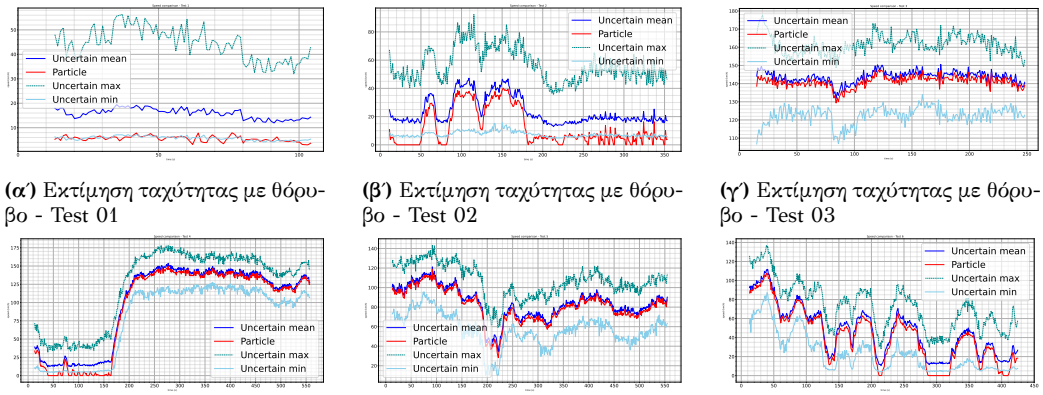
Πίνακας 5: Αυτός ο πίνακας δείχνει το κέρδος της βιβλιοθήκης *UPROP* όσον αφορά την ακρίβεια υπολογισμών σε σχέση με τον υπολογισμό μόνο με τις τιμές που μετρήσαμε.

Παρουσιάζουμε τα αποτελέσματα απόδοσης του μοντέλου μας στον Πίνακα 5. Κατά μέσο όρο έχουμε $10\times$ **κέρδος απόδοσης** σε σχέση με τον υπο-

λογισμό μόνο με τις τιμές των μετρήσεων. Αυτό το κέρδος απόδοσης **κυμαίνεται από** $2\times$ **έως** $34\times$ **σε συγκεκριμένες δοκιμές**. Η αξιολόγηση της απόδοσης του μοντέλου γίνεται με βάση την κανονικοποιημένη απόσταση Wasserstein.

Αυτοσυσχέτιση και αναπαράσταση μνήμης

Στην πλειοψηφία των υπολογισμών, η μέση τιμή των `Uncertain` αντικείμενων συγκλίνει με την τιμή των υπολογισμών με ένα δείγμα. Εκτός από την επιτυχημένη σύγκλιση, βλέπουμε ότι η βιβλιοθήκη UPROP καταφέρνει να προωθήσει το θόρυβο εισόδου στην εξόδο, κάτι που φαίνεται στην Εικόνα 7. Η επίδραση του θορύβου είναι εμφανής στο αποτέλεσμα. Η μέγιστη και ελάχιστη τιμή πολλές φορές αποκλίνουν κατά 50% από τη μέση τιμή. Η βιβλιοθήκη *UPROP* μπορεί να βοηθήσει τον προγραμματιστή να προβλέψει αυτές τις ακραίες τιμές και να πράξει αναλόγως.



(α΄) Εκτίμηση ταχύτητας με θόρυβο - Test 01

(β΄) Εκτίμηση ταχύτητας με θόρυβο - Test 02

(γ΄) Εκτίμηση ταχύτητας με θόρυβο - Test 03

(δ΄) Εκτίμηση ταχύτητας με θόρυβο - Test 04

(ε΄) Εκτίμηση ταχύτητας με θόρυβο - Test 05

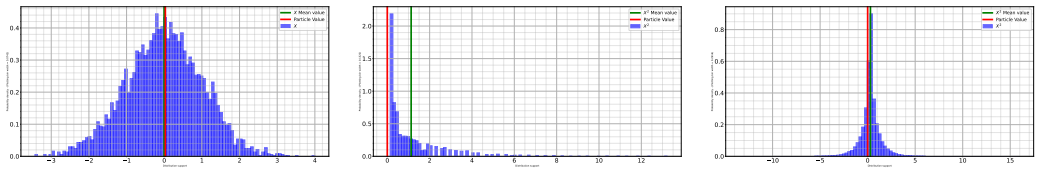
(ς΄) Εκτίμηση ταχύτητας με θόρυβο - Test 06

Σχήμα 7: Εκτίμηση Ταχύτητας - Σύγκριση μεταξύ της χρήσης `Uncertain` αντικειμένων και υπολογισμό με ένα δείγμα. Απεικονίζουμε την μέση τιμή, το ελάχιστο και το μέγιστο κάθε μεθόδου.

Σε μερικές περιπτώσεις, βλέπουμε πως το μοντέλο αποτυγχάνει να υπολογίσει την πραγματική μέση τιμή της ταχύτητας. Πιο συγκεκριμένα, παρατηρούμε πως **όσο πιο κοντά στο 0 είναι η υπολογισμένη τιμή, τόσο μεγαλύτερη είναι η ανακρίβεια του μοντέλου μας**. Η συμπεριφορά αυτή είναι ιδιαίτερα εμφανής στις Εικόνες 4.9a, 02 4.9b, 04 4.9d και 06 4.9f.

Αποδίδουμε αυτή τη συμπεριφορά στη πιθανοτική φύση των `Uncertain` αντικειμένων, και μόνο σε ορισμένες συνθήκες και μετασχηματισμούς/συναρτήσεις: ① το πεδίο ορισμού της `Uncertain` μεταβλητής πρέπει να βρίσκεται εκατέρωθεν του 0, ② η συνάρτηση μετασχηματισμού πρέπει να είναι

άρτια, π.χ. $f(x) = x^2$, και ③ πρέπει να ελέγχουμε για αυτοσυσχέτιση μεταβλητών. Ας υποθέσουμε ένα παράδειγμα όπου έχουμε το δείγμα $x = 0$ και το αντίστοιχο `Uncertain` αντικείμενο, μια κανονική κατανομή με αόριστη διασπορά, $\mu = 0$ και η συνάρτηση μετασχηματισμού είναι η $f(x) = x^2$. Εαν ικανοποιούνται οι παραπάνω συνθήκες, τότε το παραγόμενο `Uncertain` αντικείμενο προκύπτει με αρνητική λοξότητα ενώ η τιμή που αναλογεί στο ένα δείγμα είναι περίπου 0. Δείχνουμε αυτήν τη συμπεριφορά στην Εικόνα 8



(α΄) `Uncertain` μεταβλητή εισόδου $u_1 = \text{Uncertain}(X \sim \mathcal{N}(0, 1))$ και δείγμα εισόδου $p_1 = 0$.

(β΄) Αποτέλεσμα με **Αυτοσυσχέτιση ON**.

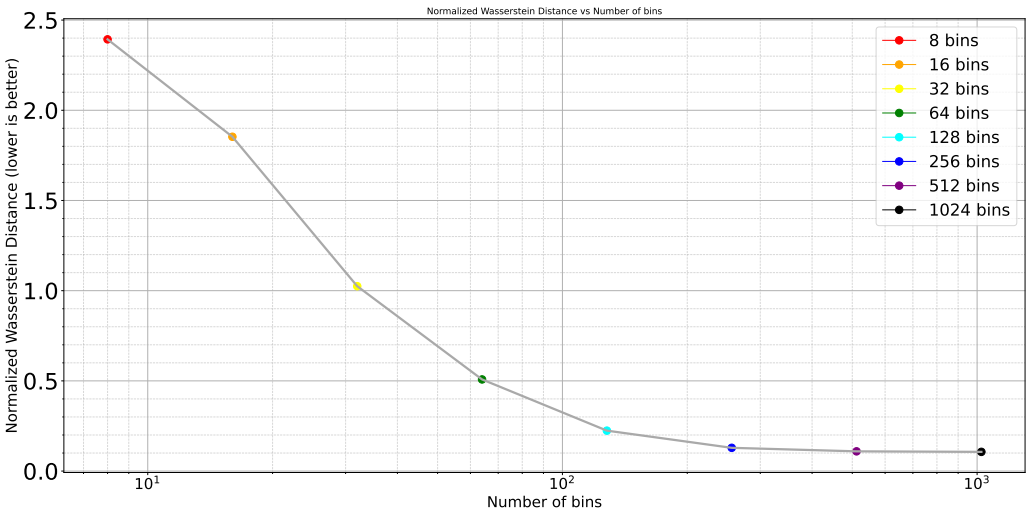
(γ΄) Αποτέλεσμα με **Αυτοσυσχέτιση OFF**.

Σχήμα 8: Ανάλυση κατανομής των `Uncertain` κατανομών με Αυτοσυσχέτιση ON και OFF. Με την αυτοσυσχέτιση ON: στην Εικόνα 8β΄, $u_2 \sim (\chi^2)$. Με την αυτοσυσχέτιση OFF: στην Εικόνα 8γ΄ $u_2 \sim \text{Laplace}(\mu = 0, b = 1)$. Ιδανικά θα έπρεπε να ήταν: $\text{mean}(u_2) \approx p_2$ σε κάθε περίπτωση.

3.5 Μοντέλο εξάρθρωσης Brown-Ham

Σε αυτό το πείραμα υπολογίζουμε την τάση κοπής ενός κράμαατος μετάλλων χρησιμοποιώντας το μοντέλο εξάρθρωσης Brown-Ham [8, 9]. Οι Anderson et al. [8] παρέχουν εμπειρικά εύρη τιμών για τις εισόδους του μοντέλου μετατόπισης. Υποθέτουμε ότι οι είσοδοι του μοντέλου ακολουθούν μια ομοιόμορφη κατανομή σε αυτές τις περιοχές. Η εξίσωση 4.5.1 παρέχει την εξίσωση υπολογισμού της τάσης κοπής τους κράματος. Αρχικά, μετατρέπουμε όλες τις μεταβλητές που υπάρχουν στην εξίσωση 4.5.1 σε `Uncertain` αντικείμενα και στη συνέχεια εκτελούμε τον υπολογισμό. Χρησιμοποιούμε την προσομοίωση Monte Carlo ως μέθοδο αναφοράς.

Στην Εικόνα 9 παρατηρούμε ότι η απόσταση είναι αντιστρόφως ανάλογη με τον αριθμό των στηλών (όσο χαμηλότερο τόσο καλύτερο). Το κέρδος ακρίβειας της βιβλιοθήκης *UPROP* κυμαίνεται από $2\times$ -- με μέγεθος αναπαράστασης 64, έως $9,4\times$ -- με μέγεθος αναπαράστασης 1024, σε σύγκριση με τον υπολογισμό χρησιμοποιώνας ένα δείγμα. Εάν εκτελέσουμε τη συγκριτική αξιολόγηση με τα `Uncertain` αντικείμενα να έχουν μέγεθος αναπαρστασης από 8 έως 32, το μοντέλο μας έχει έως και $2\times$ χειρότερη ακρίβεια.



Σχήμα 9: Η Κανονικοποιημένη απόσταση Wasserstein μεταξύ της υλοποίησης με την βιβλιοθήκη *UPROP* και τον υπολογισμό ενός δείγματος, στο Πείραμα Brown-Ham. Παρατηρούμε πως η απόσταση είναι αντιστρόφως ανάλογη στον αριθμό των στηλών (bins) που χρησιμοποιούνται. (όσο χαμηλότερο τόσο καλύτερο).

4 Σύνοψη και μελλοντική έρευνα

4.1 Μελλοντική Έρευνα

Η βιβλιοθήκη *UPROP* αναλαμβάνει την πρόκληση να πλαισιώσει την αβεβαιότητα και το θόρυβο μετρήσεων χρησιμοποιώντας μια αναπαράσταση κατανομής στη μνήμης. Ο σχεδιασμός παρόμοιων συστημάτων, αποτελεί μια τεράστια πρόκληση τόσο από την άποψη μαθηματικών γνώσεων όσο και στη διαχείριση του υπολογιστικού κόστος εκτέλεσης. Προτείνουμε μερικές ενδιαφέρουσες δυνατότητες που μπορεί κανείς να ερευνήσει και η βιβλιοθήκη μας μπορεί να χρησιμεύσει ως βάση για αυτή την έρευνα, καθώς παρέχει τη δυνατότητα διαχείρησης και διάδοσης του θόρυβου εισόδου, στην έξοδο.

Αποτελεσματικότητα στους υπολογισμούς

Όπως συζητάμε εν συντομία στην Ενότητα 4.2, κατά τον υπολογισμό με τη βιβλιοθήκη *UPROP*, η επιλογή του αριθμού των στηλών (bins) που αντιπροσωπεύουν ένα αντικείμενο `Uncertain` ενσωματώνει τον κλασικό συμβιβασμό μεταξύ ταχύτητας και ακρίβειας. Η επιλογή του σωστού αριθμού bins είναι εξαιρετικά σημαντική: πολύ υψηλός και ο τύπος `Uncertain` θα είναι πολύ αργός για πρακτική χρήση. πολύ χαμηλός και θα είναι πολύ ανακρι-

βής για την επίλυση προβλημάτων. Ο τύπος `Uncertain` σίγουρα αυξάνει το υπολογιστικό κόστος σε σύγκριση υπολογισμούς ενός δείγματος. Η εύρεση τρόπων μείωσης ή εξάλειψης των περιττών υπολογισμών για τη βελτιστοποίηση χρόνου εκτέλεσης, θα ήταν το επόμενο σημαντικό βήμα στην χρήση της βιβλιοθήκης.

Εύρεση συσχέτισης μεταξύ μεταβλητών

Στην Υποενότητα 4.3.5, αναλύουμε μερικούς από τους λόγους για τους οποίους η εύρεση αυτοσυσχέτισης είναι σημαντική. Η αυτοσυσχέτιση αποτελεί ένα εργαλείο το οποίο μπορεί να επηρεάσει τη σχέση μεταξύ αιτίου και αιτιατού. Από τη άλλη η εύρεση της εισάγει υπολογιστικό κόστος. Μια τυπική λύση σε αυτό το πρόβλημα μπορεί να είναι η παρότρυνση του χρήστη να επαναπροσδιορίζει ρητά τις σχέσεις των `Uncertain` αντικείμενων ανά τακτά διαστήματα. Αυτό όμως θα μας παρέτρεπε από το σχεδιαστικό στόχο του μινιμαλισμού.

Σφάλμα μετατόπισης μετά από πράξεις

Τέλος, στο Section 4.7 παρουσιάζουμε το σφάλμα μετατόπισης που προκαλείται από τη χρήση των συνδυασμών συναρτήσεων Ντιράκ (ιστογράμματων) για τη μετάδοση του θορύβου στις πράξεις. Η εκτέλεση πράξεων με τις διάμεσες τιμές κάθε bin, έχει ως αποτέλεσμα μια "μετατόπιση προς τα δεξιά" των αποτελεσμάτων αφού κατά την παραγωγή του αποτελέσματος αγνοούμε ορισμένα δείγματα. Αυτή η συμπεριφορά ήταν αναμενόμενη καθώς είναι παράγωγο των σχεδιαστικών επιλογών μας. Επιλέξαμε ρητά να την κρατήσουμε στο σύστημά μας γιατί θεωρούμε ότι το κέρδος απλότητας των υπολογισμών αντισταθμίζει την απώλεια ακρίβειας, η οποία μπορεί να αντιμετωπιστεί με αύξηση των bins. Ωστόσο, θα ήταν ενδιαφέρον αν καταφέρουμε να καταπολεμήσουμε αυτό το σφάλμα μετατόπισης, αναλύοντας τα πλάτη των bins των τυχαίων μεταβλητών εισόδου και μετατοπίζοντας αριστερά το παραγόμενο αποτέλεσμα.

4.2 Σύνοψη

Παρέχοντας ανεπαρκή εργαλεία και δομές δεδομένων, οι γλώσσες προγραμματισμού ενθαρρύνουν τους προγραμματιστές να αγνοήσουν το θόρυβο στα δεδομένα και οι υπάρχουσες δομές είτε δεν είναι αρκετά εκφραστικές είτε απαιτούν σημαντικό χρόνο για έναν προγραμματιστή να τις χρησιμοποιήσει άνετα και αποτελεσματικά. Η βιβλιοθήκη *UPROP* συνεισφέρει στη

αντιμετώπιση αυτού του προβλήματος. Υποστηρίζει τη μετάδοση του θορύβου στους υπολογισμούς ενώ παράλληλα προσφέρει εύκολη και εύχρηστη διαχείριση του χωρίς υπερβολικές απαιτήσεις από τον χρήστη.

Αν και αυτή η προσέγγιση διασφαλίζει ότι η πρόσβαση στο θόρυβο γίνεται με ένα ευχρηστό και μινιμαλιστικό τρόπο, υπάρχουν και μειονεκτήματά. Κατά τον υπολογισμό με τη βιβλιοθήκη *UPROP*, η επιλογή του μεγέθους αναπαράστασης των `Uncertain` αντικειμένων ενσωματώνει τον κλασικό συμβιβασμό μεταξύ ταχύτητας και ακρίβειας. Η επιλογή του σωστού μεγέθους είναι εξαιρετικά σημαντική: πολύ υψηλό και η χρήση της βιβλιοθήκης θα είναι πολύ αργή για πρακτική χρήση, πολύ χαμηλή και δε θα υπάρχει αρκετή ακρίβεια στους υπολογισμούς.

Chapter 1

Introduction

The advance of technology is based on making it fit in so that you don't really even notice it, so it's part of everyday life.

– Bill Gates

Uncertainty and risk play a significant role in our everyday life. A degree of uncertainty accompanies everyday decisions and judgements followed by a risk assessment for the estimation of the probable causes. We are so accustomed to it that we normally do not even recognize it explicitly. The ability of computing systems to store and process data has continuously improved, but the ability of programs to cope with and reason about uncertainty has not.

In the past, we designed control systems as isolated and closed systems under the control of one manufacturer and/or closed and protected environments. With the rise of usage of the Internet of Things, self-driving cars and automation in every aspect of our lives, we are increasingly dependent on accurate sensor readings. Thus, the ability of computer systems has to keep up with the increasing flow of uncertain measurements.

Sensors affect each measurement by inducing noise to it and this produces uncertainty regarding the true value of the measurand (*quantity to measure*) [1]. The nature of physical measurements implies that there is always some uncertainty between the recorded result and measurand. Programmers have to take this uncertainty under consideration when developing applications. Treating uncertain data as exact might lead to incorrect results with catastrophic effects. Thus, the development of such applications that handle uncertainty effectively without adding any per-

formance overhead, is a major challenge. Programmers need to have interdisciplinary knowledge which is not something common. In this direction, robust ways and frameworks of tracking and quantifying uncertainty are essential, as the autonomy of computing systems increases. In the above example, of autonomous cars, the ability of the computing system to quantify how uncertain a decision might be is a key element to reducing hazards and avoiding accidents.

In this thesis, we investigate, propose, and develop a programming model for *describing*, *storing*, *executing* arithmetic operations on uncertain data, as well as using it to construct applications that handle them. We expand the popular programming language Python to allow applications to construct variables containing distributional information according to well-known parametric distributions (e.g., Gaussian) or empirical data (data sampled from sensors), advancing the state-of-the-art.

The *UPROP* library facilitates uncertainty propagation through arithmetic and conditional operators, and function calls. When initializing an `Uncertain` object containing distributional information, the developer has the option to perform statistical and probabilistic queries on this object, to more accurately predict and manage possible events.

1.1 Problem Statement

Data uncertainty measurement is a challenging and time-consuming task that may need advanced statistical and scientific approaches (classical or Bayesian) as well as human judgment to assess. Modern applications are increasingly confronted with the difficulty of computing in unpredictable environments. This ambiguity shows itself in the data that computers manage. In certain circumstances, the uncertainty originates by a sensor's low resolution or accuracy.

One type of *measurement uncertainty* is the spread across multiple measurements while the measurand is presumably constant. We reference the type of uncertainty caused by variance in values for a (nominally) fixed measurand as *aleatoric uncertainty*. When we retrieve measurements at the same time using different measurement tools, we may encounter aleatoric uncertainty. It is also possible to be uncertain about a measurand simply because we do not have enough information about it. We call this type of uncertainty as *epistemic uncertainty*. When training a neural network, for example, you begin with high values of epistemic uncertainty about the weights required to achieve a high prediction accuracy.

Suppose a cluster of sensors, that measure an autonomous car's dis-

tance from the leading car/object, when driving. These sensors are responsible for collision avoidance at any time. The computer of the car performs calculations based on the measurements provided by these sensors. Although sensor manufacturers report expected *aleatoric uncertainty* for their sensors, there might still be a high level of measurement uncertainty as Mohan et al. [10] report. In the possibility that some measurements are not available, we used output feedback control laws or state estimation algorithms [11]. These observations lead to the conclusion that, regardless of the hardware choice, developers still have to deal with uncertainty and this requires interdisciplinary knowledge. Code Listing 1.1 depicts the pseudo-code that an autonomous vehicle speed controller may use to maintain a constant distance from the vehicle in front of it, using sensor readings as input.

```
if distance < 10: # meters
    car.decrease_speed()
elif distance == 10:
    car.maintain_speed()
else:
    car.steady_speed()
```

Listing 1.1: False positives in conditionals.

In Listing 1.1, if the developer ignores the underlying uncertainty in measurement then the car might crash, due to late braking. Even if something implausible occurs, and the sensor reports a distance greater than 10 meters, the controller will instruct the car to accelerate, resulting in a collision. This information should urge the developer either (i) make some statistical analysis of the data, or (ii) treat measurements as uncertain before making critical decisions.

In most cases, rather than precise numbers, we should represent uncertain as probability distributions or approximated values with error boundaries. However, handling data as such would require developers to rewrite their code, perhaps resulting in optimization issues. This is why *many estimation techniques assume the available measurements are disturbance free* [10]. This means developers treat data as facts, rather than estimates. Bornholt [3] shows that this could lead to false positives and or false negatives.

1.2 Examples with Uncertainty

The most prominent examples of measurement uncertainty occur in our everyday interaction with technology: the smartphone. From blurry images captures to inaccurate GPS measurements, we subconsciously encounter uncertainty every time we use our smartphone. In this section we present some examples of uncertainty encounter that motivated us to develop the *UPROP* Python Library.

GPS Example

Modern and emerging applications compute over uncertain data from mobile sensors, search, vision, medical trials, benchmarking, chemical simulations, and human surveys. Characterizing uncertainty in these data sources requires domain expertise, or interdisciplinary knowledge. These data have become widely available and non-expert developers are increasingly consuming the results.

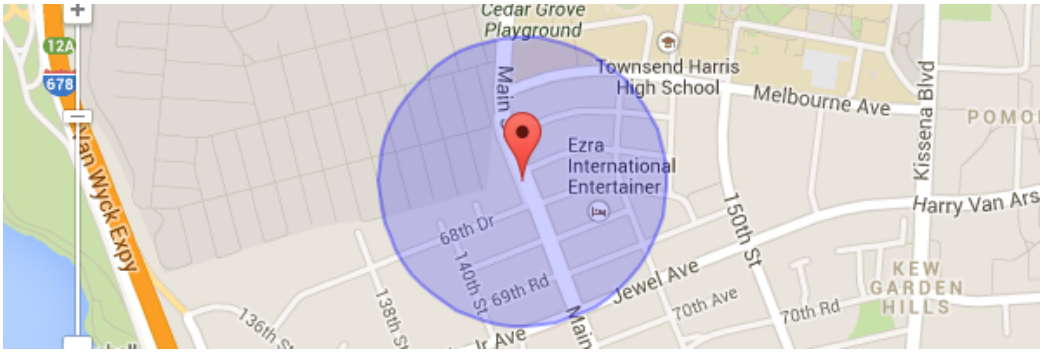


Figure 1.1: Google Maps representes uncertainty in GPS measurements with a circle of a specific radius, provided by the smartphone.

This section uses Global Positioning System (GPS) data to motivate a correct and accessible abstraction for uncertain data. On mobile devices, GPS sensors estimate location. APIs for GPS typically include a position and estimated error radius (a confidence interval for location). As we see in Figure 1.1 Google Maps depicts the horizontal accuracy as a radius around the measured coordinates.

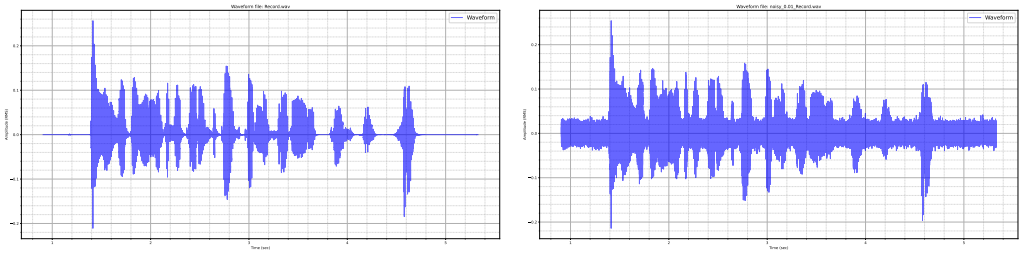
```
1 Latitude, Longitude, Horizontal_Accuracy = GPS_API()
```

Listing 1.2: GPS API usually returns a tuple of three elements: Latitude, Longitude and Horizontal accuracy.

Noisy sounds example

Another common example of noise and uncertainty in our everyday life is noisy sound signals. The online communication through video calls and conferences has seen an increased usage particularly after the initial outbreak of COVID-19 [12]. Communicating efficiently, without interferences due to bad signal strength, leading to ambiguous sound signals is not yet established. There has been extensive research in Speech and Language Processing (SLP) to counteract these obstacles. Povey et al. proposed the Kaldi speech recognition toolkit [13], which is intended for use by speech recognition researchers. Collobert et al. [14] propose a convolutional neural network for speech regocnition. Virtual personal assistants such as Google, Siri or Alexa [15] have seen an increased usage.

All these technologies strive to recognise speech using complex algorithms and Machine Learning methods. We propose the *UPROP* Python Library, a library that researchers can use to process sound signals and possibly manage uncertainty. We visualise a — clear and the equivalent noisy, speech signal in Figure 1.2, the first (1.2a) beeing the clear speech and the second (1.2b) with Added White Gaussian Noise (AWGN). Later in Chapter 4 we put the *UPROP* Python Library to the test, performing a sound recognition algorithm.



(a) Clear sounded speech.

(b) Equivalent Noisy speech.

Figure 1.2: Clear sounded and noisy speech are a common example in our everyday life when communicating via the internet.

1.3 Contributions of this work

In this work we present an uncertainty propagation library: the *UPROP* Python Library. This library wraps the `Uncertain` datatype, a datatype that encapsulates probability distributions — either empirical or well-known parametric distributions. Using the available computer architecture, we describe a memory representation to store probability

distributions and based on probability theory, we propagate Uncertainty through operations. We utilize Python's built in methods to our advantage and manage to propagate uncertainty even through user defined functions. The main advantage of the `Uncertain` type originates from it's flexibility and minimalism in usage.

We evaluate the proposed abstraction with micro-benchmarks and real life applications. The micro-benchmarks consist of tests on operators with well-known parametric distributions. The applications benchmarks test a combination of operators and functions mapping: ① using the Haversine Formula [16] to estimate user speed from uncertain GPS coordinates, ② a sound detection algorithm with noisy sound signals as input ③ and calculation of the cutting stress of an alloy precipitate using the Brown-Ham dislocation model [8, 9].

We compare the *UPROP* Python Library with the Monte Carlo Simulation and state-of-the-art platforms for uncertainty propagation. When using high number of bins the usage of the `Uncertain` type offers a significant advantage over the conventional calculation method, ranging from $1.2\times$ to $40\times$ more accurate. With regards to the comparison with other state-of-the-art frameworks, when using a small number of bins the *UPROP* library performs from $10\times$ to $2\times$ worse, while with higher numbers we observe an increase in accuracy. Overall the *UPROP* Python Library constitutes a flexible and minimal tool that the developer may use to ensure uncertainty propagation when designing a system.

1.4 Thesis Outline

Chapter 2 supports the thesis theoretical background by providing an overview of probability theory and random variables. In Chapter 3 we present some of the design principles that we use in the development of the *uncertain package*. We also show the implementation insights of the `Uncertain` type, information on memory representation of uncertain data and minimal examples of usage. Chapter 4 presents three case studies, specific unit tests and presents the results in comparison with state-of-the-art frameworks. Chapter 5 provides information about relevant literature around the field of programming with uncertainty. Finally, Chapter 6 concludes the thesis, summarizes the main points and points out possible future work.

Chapter 2

Notation & Theoretical Backgound

2.1 Probability Theory

In deterministic mathematics, a variable can take a single value at every single moment. Either $x = 5$ or $x = 4$ holds *True*. This is not true with *random variables* which variables often used to simulate the outcome of an *experiment*. It is possible for a *random variable* to take multiple values. These outcomes or possible values that a random variable can take must be mutually exclusive, which means they cannot happen simultaneously. To understand the concept of *random variables* we must first define and model an ”experiment” or ”random process”. Thus, we have to use the **probability space** or **probability triple** $(\Omega, \mathcal{F}, \mathcal{P})$. A *probability space* consists of three elements:

❶ A **sample space** Ω, which is the set often all possible outcomes.

❷ An **event space**, which is a set of events $\mathcal{F}$, an event being a set of outcomes in the *sample space*.

❸ A **probability function** $\mathcal{P}$, which assings each event in the event space a *probability*, which takes values between 0 and 1.

2.1.1 Probability

Thus, $\mathcal{P}$ or P is a function $P : \mathcal{F} \to [0, 1]$. To understand the notion of *probability function* we must first introduce the notion of the *probability* of an event. Consider an experiment: a single dice roll of a fair, six-sided die.

In terms of a random experiment, this is nothing but randomly selecting a sample of size 1 from a set of numbers that are mutually exclusive outcomes. In this particular experiment, the sample space is $\{1, 2, 3, 4, 5, 6\}$ from which we can form different events based on conditions, e.g., *'the observed outcome lies between 2 and 5'*.

Suppose that we repeat an experiment N times, keeping the conditions as similar as possible and that A is some event that may or may not occur on each repetition. After N trials the outcome A occurred $N(A)$ times. As N becomes larger, the ratio $N(A)/N$ converges to a constant limit. We refer to the probability of an event, as the proportion that the event occurs in the long run, that is if we repeat the experiment multiple times.

Definition 2.1.1 *Let A be an event in the Probability Space Ω [17]. Let $N(A)$ be the sample size of A and $N(\Omega)$ be the sample size of Ω. We notate the probability of the event A as $P(A)$, $\mathbb{P}(\mathcal{A})$ or $\Pr(A)$ and is given by the Equation*

$$P(A) = \lim_{N \to \infty} \frac{N(A)}{N(\Omega)} \tag{2.1.1}$$

Given the sample space Ω, of size $N(\Omega)$ and $N(A)$ the number of times in favor of A, we can also define probability as the ratio of the samples of A to the samples of Ω [17].

$$P(A) = \frac{N(A)}{N(\Omega)} \tag{2.1.2}$$

The probability $P(A)$ has the following properties:

1. $P(\Omega) = 1$
2. $P(\emptyset) = 0$
3. $0 \leq P(A) \leq 1, \forall A \subseteq \Omega$

with $\emptyset$ beeing the empty set: $\{\}$, $P(A) = 1$ only when $A = \Omega$, and $P(A) = 0$ only when $A = \emptyset$,

We say that two events A and B are **independent** if the occurrence of A does not provide any information for the occurrence of B and vice versa. If whether or not one event occurs does affect the probability that the other event will occur, then the two events are **dependent**. In terms of probability this means that

$$P(A \text{ and } B) = P(A) \cdot P(B) \tag{2.1.3}$$

2.1.2 Random Variables

We unify all the above ideas under the concept of a **Random Variable (RV)** which constitutes a numerical summary of random outcomes. The possible values we can assign to a random variable can be either discrete or continuous.

- **Discrete** random variables have discrete outcomes, e.g., the possible outcomes when rolling a die: $\{1, 2, 3, 4, 5, 6\}$.
- A **continuous** random variable may take on a continuum of possible values, countably or uncountably infinite, e.g., values in range: $[0, \pi]$

Random variables are usually notated with an uppercase letter, e.g., X.

Definition 2.1.2 *A random variable X is a measurable mapping from the sample space Ω associated with a random experiment into the set of real numbers $\mathbb{R}$ [17].*

$$X : \Omega \to \mathbb{R} \tag{2.1.4}$$

The sample space is the domain *over* on which we define a random variable and **not** the set of values it can take. We represent the probability that a random variable X over Ω takes the value $x \in \Omega$ with a *probability distribution*: $p : \Omega \to [0, \infty)$.

In the same way as Equation 2.1.3, we say two random variables are independent if the value of one has no bearing on the value of the other. Formally we say two random variables X and Y are *independent* if, for every possible $x \in \Omega_X$ and $y \in \Omega_Y$:

$$P(X = x \text{ and } Y = y) = P(X = x) \cdot P(Y = y) \tag{2.1.5}$$

We say that a *probability distribution* is a list of outcomes and their associated probabilities. For different type of Random Variables we define a different type of probability function.

❶ We represent a *discrete* Random Variable with a *probability mass function (PMF).*

❷ We represent a *continuous* Random Variable with a *probability density function (PDF).*

In Section 2.1.2 we discuss the two categories of a random variable. Figure 2.1 shows an example of a discrete random variable that takes

discrete values in the range: $(0, 6.5)$. Figure 2.2 shows a continuous random variable with distribution support in the range: $(0, 0.05)$

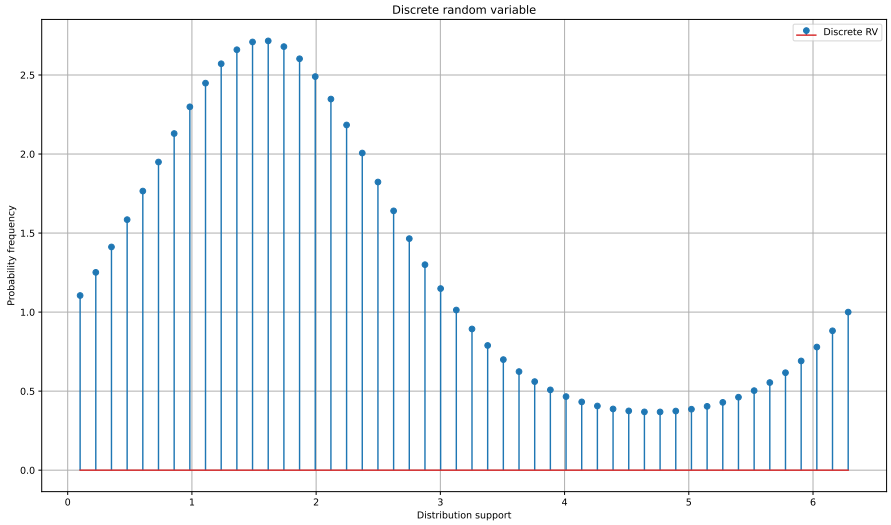


Figure 2.1: A discrete random variable.

Definition 2.1.3 *Let X be a discrete random variable. Let S be the sample space of X. We denote the probability mass function X as $p(x)_X$. The PMF assigns probabilities to the possible values of the random variable [17]. Specifically, if $x_1, x_2, \in S$ then the PMF is:*

$$p(x_i) = P(X = x_i) = P(\{s \in S \mid X(s) = x_i)\}) \tag{2.1.6}$$

where P is a probability measure and $p(x)_X$ can also simplified as $p(x)$. For the probability mass function it is true that:

$$\sum_x p_X(x) = 1 \tag{2.1.7}$$

Definition 2.1.4 *Let X be a discrete random variable. The cumulative distribution function of X is $F_X(x)$. $F_X(x)$ is a function that maps the values x on the real numbers domain [17].*

$$F_X(x) = P(X \leq x) = \sum_{x_i \leq x} P(X = x_i) = \sum_{x_i \leq x} p(x_i) \tag{2.1.8}$$

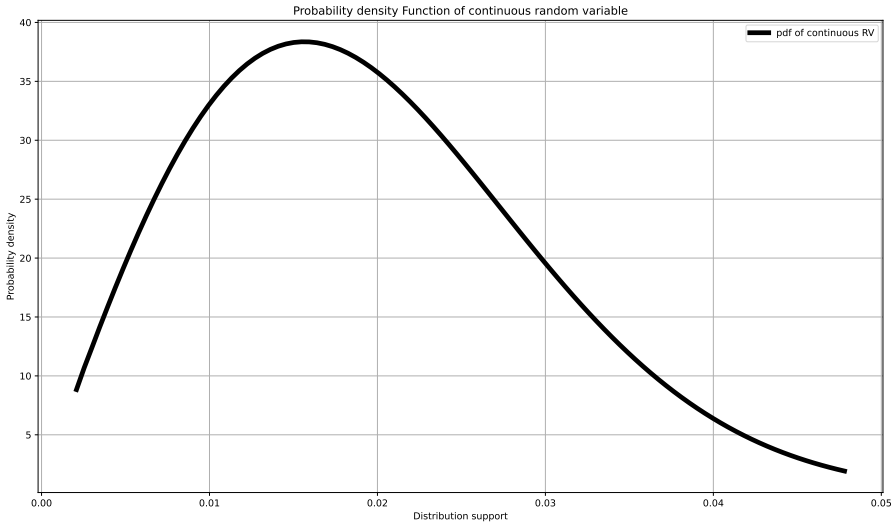


Figure 2.2: PDF of a continuous random variable.

To determine the distribution of a *discrete* random variable we can either provide its PMF or CDF. For *continuous* random variables, the CDF is well-defined so we can provide the CDF. However, the PMF does not work for continuous random variables, because $P(X = x) = 0$ for all $x \in \mathbb{R}$. Instead, we can use the probability density function (PDF). The concept is similar to mass density in physics: its unit is the probability per unit length.

Definition 2.1.5 *Let X be a continuous random variable. Let $\Delta > 0$. The probability density function of X is $f_X(x)$ [17].*

$$f_X(x) = \lim_{\Delta \to 0} \frac{P(x < X \leq x + \Delta)}{\Delta} \tag{2.1.9}$$

Using Equations 2.1.8 and 2.1.9 we conclude that:

$$f_X(x) = \lim_{\Delta \to 0} \frac{F_X(x + \Delta) - F_X(x)}{\Delta} = \frac{dF(x)}{dx} = F'(x) \tag{2.1.10}$$

We consider a continuous random variable X with an absolutely continuous CDF $F_X(x)$. Since the PDF is the derivative of the CDF, shown in Equation 2.1.10, we can obtain the CDF from PDF by integration.

Definition 2.1.6 *Let X be a continuous random variable. The Cumulative Density Function of X is $F_X(x)$ [17].*

$$F_X(x) = \int_{-\infty}^{x} f_X(u)du \tag{2.1.11}$$

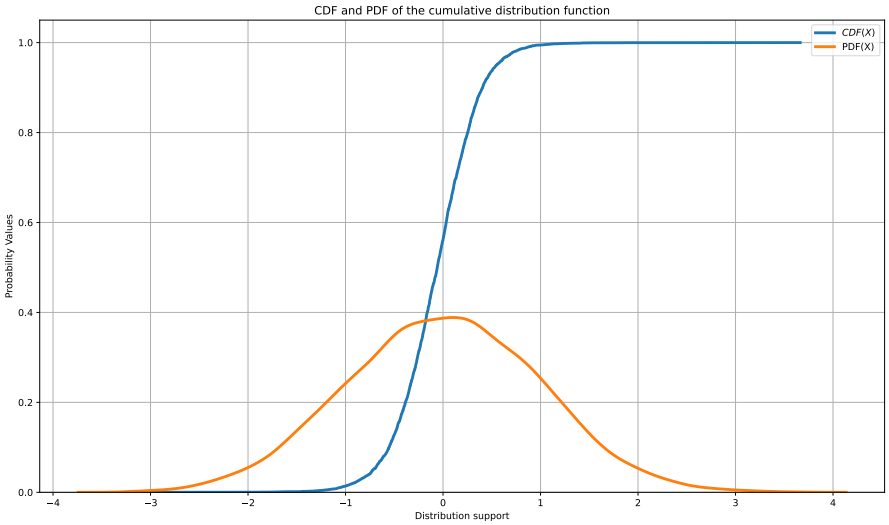


Figure 2.3: Cumulative Distribution Function and Probability Density Function of a continuous random variable $X \sim \mathcal{N}(0, 1)$[1]

Also, we have:

$$P(a < X \leq b) = F_X(b) - F_X(a) = \int_{a}^{b} f_X(u)du \tag{2.1.12}$$

In particular, the integral of the CDF over the entire range of the random variable (2.1.14) should be equal to 1 [17].

$$\int_{-\infty}^{\infty} f_X(u)du = 1 \tag{2.1.13}$$

The range of a random variable X is the set of possible values of the random variable. If X is a continuous random variable, we can define the range of X as the set of real numbers x for which the PDF is larger than zero, i.e.,

[1] $X \sim \mathcal{N}(\mu, \sigma^2)$ represents a Standard Normal Distribution with mean value: μ and variance: σ^2.

Definition 2.1.7 *Let X be a random variable. The range of X is R_X and is the range of all possible values that X can take. We refer to R_X as Distribution Support [17].*

$$R_X = \{x | f_X(x) > 0\} \tag{2.1.14}$$

Definition 2.1.8 *Let X be a random variable. The expected value $\mathbb{E}[X]$ of X is the weighted average of all possible values of the variable [17]. If X is discrete with probability mass function $p(x)$ then $\mathbb{E}[X] = \sum_{i=1}^{\infty} x_i p(x_i) dx$. If X is continuous with probability density function $f_X(x)$ then $\mathbb{E}[X] = \int_{\Omega} x f(x) dx$ [17].*

We also call $\mathbb{E}[X]$ the *mean or average of A* and we represent it with the greek letter μ.

Definition 2.1.9 *Let X be a random variable and k be a positive integer. The k^{th} moment and the the k^{th} centralized moment σ_k of X [17] respectively are:*

$$m_k = \mathbb{E}(X^k) \tag{2.1.15}$$

$$\sigma_k = \mathbb{E}[(X - \mu_1)^k] \tag{2.1.16}$$

Definition 2.1.10 *The variance: $Var(X)$ or σ^2, of a random variable X is the 2^{nd} centralized moment of X (σ_2). It is the expectation of the squared deviation of the random variable from its population mean. If X is discrete with PMF $p(x)$, then $Var(X) = \mathbb{E}[(X - \mu)^2] = \sum_{i=1}^{\infty} p(x_i)(x_i - \mu)^2$, while if X is continuous with PDF $f(x)$, then $Var(X) = \int_{\Omega} x^2 f(x) dx - \mu^2$ [17].*

Definition 2.1.11 *The mode of a random variable is the value at which the PDF (or the PMF) is at a maximum.*
If X discrete and has PMF $p(x)$:

$$mode = max\{x_i : p(x_i) \geq p(x_j),\ i \neq j\ \forall\{i, j\} \in \Omega\} \tag{2.1.17}$$

If X is continuous and has PDF $f(x)$:

$$mode = max\{x_i : f(x_i) \geq f(x_j),\ i \neq j\ \forall\{i, j\} \in \Omega\} \tag{2.1.18}$$

2.2 Arithmetic operations on random variables

Let X, Y be continuous random variables with probability density functions f_X and f_Y accordingly. Let the first two moments of X be μ_X, σ_X and of Y be μ_Y, σ_Y. If we tranform the random variable X through multiplication with b and addition of a, then we affect the mean and variance [17].

$$\begin{aligned} \mu_{aX+b} &= a\mu_X + b \\ \sigma^2_{aX+b} &= a^2\sigma^2_X \end{aligned} \tag{2.2.1}$$

The mean of the sum of difference of two random variables X and Y is the sum or difference of their means [17].

$$\begin{aligned} \mu_{X\pm Y} &= \mu_X \pm \mu_Y \\ \sigma^2_{X\pm Y} &= \sigma^2_X + \sigma^2_Y \end{aligned} \tag{2.2.2}$$

Theorem 2.2.1 *Let X, Y, two random variables. If X and Y have a joint density function, then $Z = X + Y$ has a density function $f_{X+Y}(Z)$ [17].*

$$f_{X+Y}(Z) = \int_{-\infty}^{\infty} f(x, z - x)dx \tag{2.2.3}$$

If X and Y are two independent random variables with probability density functions $f_X(x)$ and $f_Y(y)$ respectively then the probability density function of $Z = X + Y$ is $f_Z(z)$ and is the product of the circular convolution between f_X and f_Y. We notate the operation of *convolution* between X and Y as $f_{X+Y} = f(X) * g(Y)$ [17].

$$\begin{aligned} f(X) * g(Y) = f_{X+Y}(z) = \int_{-\infty}^{\infty} f_X(x) f_Y(z - x)dx = \\ \int_{-\infty}^{\infty} f_X(z - y) f_Y(y)dy \end{aligned} \tag{2.2.4}$$

The convolution of either two discrete or continuous random variables is shown in Equation 2.2.5 [17].

$$\begin{aligned} P(Z = z) &= \sum_{k=-\infty}^{\infty} P(X = k)P(Y = z - k) \\ (f * g)(z) &= \int_{-\infty}^{\infty} f(z - t)g(t)dt = \int_{-\infty}^{\infty} f(t)g(z - t)dt \end{aligned} \tag{2.2.5}$$

To conclude, following Equations 2.1.3, 2.2.5, when applying any of the basic operators between two random variables X and Y:

❶ We generate the new distribution support as a pointwise operation $(+, -, *, /)$, between the distribution support of X and Y.

❷ we generate a new random variable Z, with PDF $f_Z(z)$. $f_Z(z)$ is the results of the convolution of $f_X(x)$ and $f_Y(y)$.

2.3 Comparing distributions

Comparing two integers or any two particle values is well defined in modern mathematics. On the other hand, comparing random variables, thus comparing distributions is not limited to a single method. Some indicators showing the difference in two distributions, are: (i) the Kolmogorov-Smirnov test [6], (ii) the Kullback-Leibler divergence [18] (iii) or the Cramer-von Mises criterion [19]. In this thesis, we utilize the two criteria shown in Subsections 2.3.1 and 2.3.2.

2.3.1 Stochastic Dominance

We refer to a partial order between random variables as **stochastic dominance** [20]. It's a type of probabilistic ordering. Let A, B be two random variables that represents the outcomes a and b of an experiment, respectively. When we rank an outcome a of an experiment as superior to another outcome b then we can state that A is *stochastically dominant* over B. Stochastic dominance does not give a total order, but only a partial order. In terms of the cumulative distribution functions of the two random variables, A *dominating* B means that the CDF of A is less or equal to the CDF of B, with strict inequality at some x, as we show in Equation 2.3.1. In this thesis we only deal with first-order stochastic dominance (FSD). In Chapter 3 we present how we utilize Definition 2.3.1 to deduce partial ordering for `Uncertain` objects.

$$F_A(x) \leq F_B(x), \ \forall \ x \tag{2.3.1}$$

Definition 2.3.1 *Let A, B be two random variables. A has first-order stochastic dominance over B if for any outcome x, A gives at least as high a probability of receiving at least x as does B, and for some x, A gives a higher probability of receiving at least x [20].*

$$P[A \geq x] \geq P[B \geq x], \ \forall \ x \text{ and for some } x : P[A \geq x] > P[B \geq x] \tag{2.3.2}$$

2.3.2 Distribution Comparison Metric

While Equation 2.3.2 is an effective method for obtaining a boolean result, there should also be a metric to help understand the concept of distance between distributions.

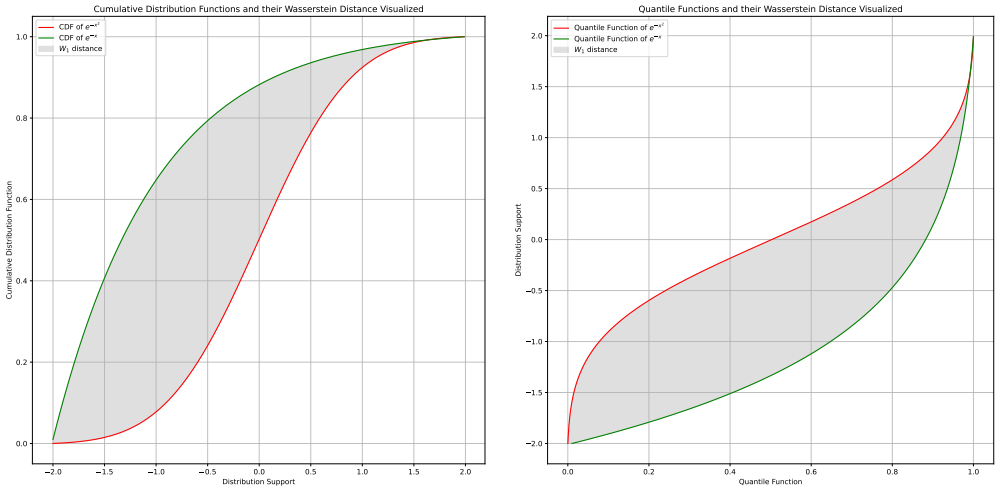
In this thesis, we propose a dissimilarity measure using the Wasserstein distance. [21, 22], also called the *Earth Mover's Distance (EMD)* [23]. The Wasserstein distance is typically used for image and audio processing as well as generative adversarial networks [24, 25]. The Wasserstein distance, thoroughly explained in Section 2.3.3, originates from the optimal transport problem, and is the distance between the distributions, and the calculation does not require any parameters. In the field of data science, a (dis)similarity between data corresponds to a distance between data. Therefore, we can obtain and quantify the dissimilarity between distributions by the Wasserstein distance. The optimal transport problem is the calculation of the minimum cost for transporting luggage from one point to another point, and thus the obtained Wasserstein distance in this thesis is the minimum deformation for changing a distribution to match another reference distribution. Hence, **a small value of the Wasserstein distance indicates that the distributions are similar, while a large value implies that they are dissimilar.**

In our evaluation, we measure the Wasserstein distance between the results of a single benchmark execution using the (i) `Uncertain` datatype and (ii) a state-of-the-art framework, using distributional inputs. We use it to compare the performance of the two frameworks.

2.3.3 Wasserstein - Earth Mover's Distance

The *Earth Mover's Distance (EMD)* or *Wasserstein Distance* is a method to evaluate divergence between two multi-dimensional distributions in some feature space where a given a distance measure between single features, called the *ground distance*. The EMD *"lifts"* this distance from measuring individual features to comparing full distributions.

Definition 2.3.2 *Let $(\mathcal{M}, d)$ be a metric space, and let $p \in [1, \infty)$. For any two probability measures μ, ν on $\mathcal{M}$, we define the Wasserstein distance of order*



(a) Calculation of the W_1 distance using the Cumulative Distribution Functions.

(b) Calculation of the W_1 distance using the Quantile Functions.

Figure 2.4: The grey area represents the W_1 distance between distribution e^{-x^2} and e^{-x}, The W_1 remains the same either using the CDF (Figure 2.4a) or the QF (Figure 2.4b) to calculate it.

p between μ and ν by the formula [21]:

$$\begin{aligned} W_p(\mu,\nu) &= \left(\inf_{\pi\in\Pi(\mu,\nu)}\int_{\mathcal{X}} d(x,y)^p d\pi(x,y)\right)^{1/p} \\ &= \inf\left\{[\mathbb{E}d(X,Y)^p]^{\frac{1}{p}}, \quad \text{law}(X)=\mu, \quad \text{law}(Y)=\nu\right\} \end{aligned} \tag{2.3.3}$$

We denote this distance as $W_p(\mu,\nu)$ or p^{th}-Wasserstein distance.

Definition 2.3.3 *We can also define W_p as:*

$$W_p(\mu,\nu)^p = \inf \mathbb{E}\left[d(X,Y)^p\right] \tag{2.3.4}$$

where d is the chosen metric and we take the infimum, over all the joint distributions of the random variables X and Y with marginals μ and ν, respectively.

To intuitively understand how the Wasserstein distance works, we provide an example. Given two distributions, the first beeing the mass of earth scattered in space and the second, a collection of holes in that same space; the Wasserstein distance measures the least amount of work needed to move the mass of earth into the holes. We call this amount of work the *ground distance*. This problem only makes sense if the mass scattered into space has the same mass as the holes to fill; therefore without loss of generality assume that μ and ν are probability distributions containing a total mass of 1. Since the *UPROP* Library supports only 1-Dimensional distributions the Wasserstein distance is, in this case, deducted to the Wasserstein-1 metric or W_1.

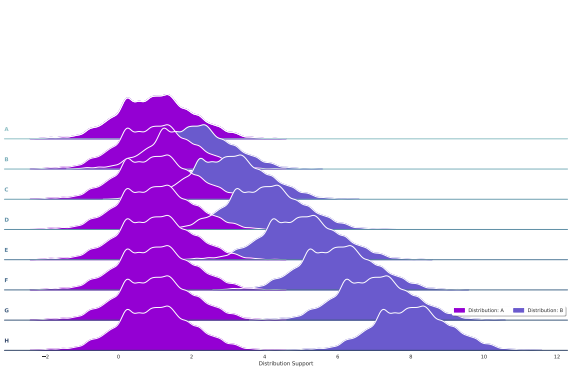
Definition 2.3.4 *Let X, Y be two random variables and their CDF F_1, F_2 respectively. We define the Wassersteind-1 metric between X and Y as:*

$$W_1(F_1, F_2) = \int_{\mathbb{R}} |F_1(x) - F_2(x)|\, dx \tag{2.3.5}$$

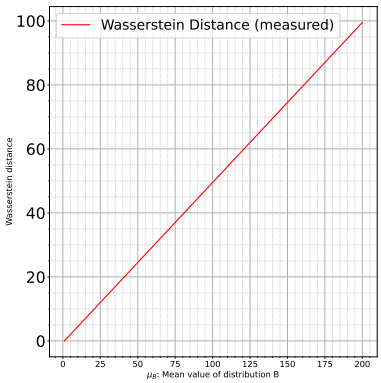
or alternatively as:

$$W_1(Q_1, Q_2) = \int_0^1 |Q_1(x) - Q_2(x)|\, dx \tag{2.3.6}$$

In Figure 2.4 we visualize the calculation of the W_1 distance. This is helpful to comprehend the calculation process. To understand the variation of the W_1 metric between two distributions we initialize two random variables A (purple) and B (blue), representing two gaussian distributions with arbitraty mean and variance. We iterate K times during which we transform the random variable B by (i) shifting it by the constant $k = 1$, (ii) scaling it by the constant $k = 1$ (iii) or shifting and scaling it by the constant $k = 1$. In Figures 2.5, 2.6 and 2.7 we present the results of these modifications, and we showcase the effect that transformations have on the Wasserstein distance. We observe that for all the examples, the Wasserstein distance follows the same transformation that we condition the random variable B.



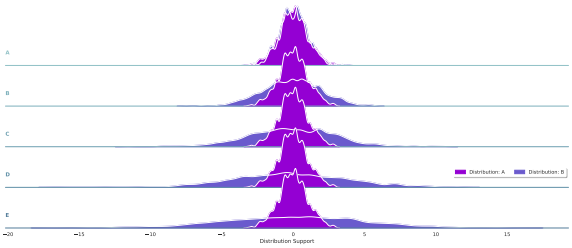
(a) Ridge plot of a distribution A and the transformed distribution B. Distribution B updates on each step. In the first step $A = B$ but in each step distribution B shifts right by a value of $k = 1$.



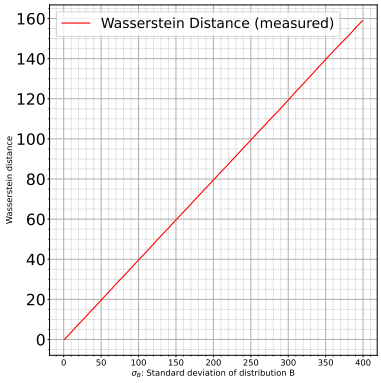
(b) Calculated Wasserstein distances (y-axis) for consecutive linear shifts of distribution B. The x-axis shows the variation of the mean value: μ_B.

Figure 2.5: Depiction of the variation of the Wasserstein Distance between distribution A and the same distribution shifted in every step: B. We observe that the variation in the Wasserstein metric follows the linear shift (by a value of $k = 1$) of the mean of distribution B.

We observe that the Wasserstein distance increase rate is faster for the combined shift and scale transformation, while the scaling example in Figure 2.6 has the slowest increase rate. We can attribute this to the fact that the two distributions share a portion of the distribution support.

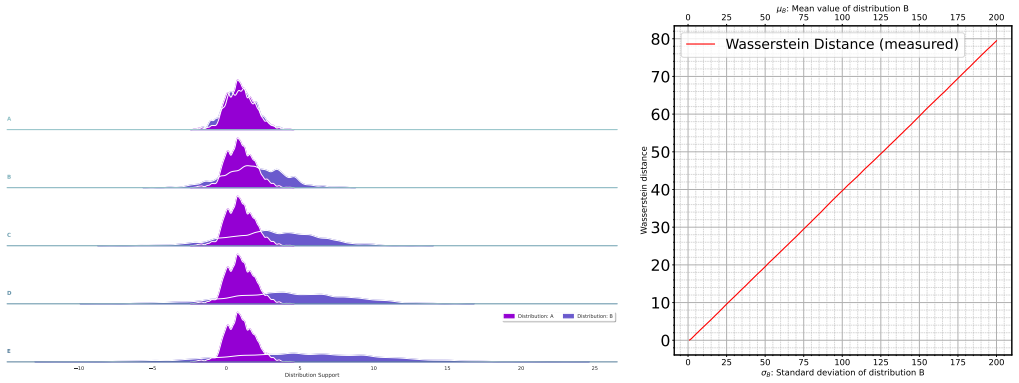


(a) Ridge plot of a distribution A and the transformed distribution B. We update distribution B on each step. In the first step $A = B$ but in each step distribution's we increase the standard deviation σ_B of B by a value of $k = 1$.



(b) Calculated Wasserstein distances (y-axis) for consecutive linear shifts of distribution B. The x-axis shows the variation of the standard deviation: σ_B.

Figure 2.6: Depiction of the variation of the Wasserstein distance between distribution A and B with the latter scaled by a factor of $k = 1$ in every step. We observe that the variation in the Wasserstein metric follows the linear shift of the standard deviation σ_B.



(a) Ridge plot of a distribution A and the transformed distribution B. We shift Distribution B in every step by 1 and we increase it's standard deviation by 1 as well.

(b) Calculated Wasserstein distances (y-axis) for consecutive linear shifts of distribution B. The x-axis shows the variation of the standard deviation: σ_B and the mean value: μ_B.

Figure 2.7: Depiction of the variation of the Wasserstei distances between distribution A and B with the latter shifted and scaled by a factor of $k = 1$ in every step. We observe that the variation in the Wasserstein metric follows the linear shift and scale of the standard deviation σ_B.

2.3.4 Kolmogorov-Smirnov Test

We use the Kolmogorov-Smirnov test (KS-test) to decide if a sample comes from a population with a specific distribution [6]. It is a non-parametric test of the equality of continuous 1-Dimensional probability distributions. With the KS-test we can obtain the Kolmogorov–Smirnov *statistic*. This *statistic* measures the distance between a sample's empirical CDF and the reference distribution's CDF, or between two samples' empirical CDFs. In particular, in this thesis, we use the two-sample Kolmogorov–Smirnov Test, to decide if two `Uncertain` objects are equal in distribution ($\stackrel{d}{=}$). We discuss this later in Subsection 3.4.1. We define the two-Sample Kolmogorov-Smirnov **test** as a set of two hypotheses:

① The null hypothesis, H_0: the two samples arose from the same distribution

② The alternative hypothesis, H_1: the two samples arose from different distributions

Suppose that the first sample has size m with an observed cumulative distribution function of $F_n(x)$ and that the second sample has size n with

an observed cumulative distribution function of $G_n(x)$. We define:

$$D_{m,n} = max_x|F_n(x) - G_m(x)| \tag{2.3.7}$$

We reject the null hypothesis (at significance level α) if $D_{m,n} > D_{m,n,\alpha}$ where $D_{m,n,\alpha}$ is the critical value. For m and n sufficiently large:

$$D_{m,n,\alpha} = c(\alpha)\sqrt{\frac{m+n}{mn}} \tag{2.3.8}$$

where $c(\alpha) =$ the inverse of the Kolmogorov distribution at α. Figure 2.8 represents the PDF of the Kolmogorov Distribution.

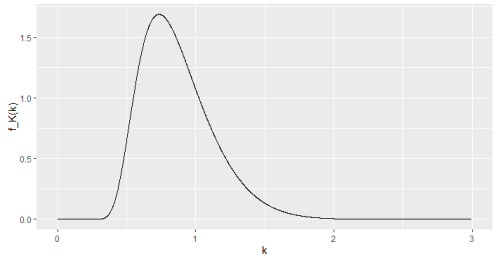


Figure 2.8: The Kolmogorov distribution's PDF [26].

In Chapter 3 we present how we utilize Equation 2.3.4 to implement equality comparisons with `Uncertain` objects.

Chapter 3

Implementation

In Chapter 2 we set the mathematical foundation needed for the *UPROP* Python Library. In this Chapter, we introduce and analyze the design principles and the implementation of the *U.PROP.* Library, (*Uncertainty Propagation*), wrapping the `Uncertain` type. We also state what is our main goal: To provide non-expert programmers, a basic, user-friendly interface with a wide range of operations, which allows them to think effectively about uncertainty and not ignore it. We present our efforts of creating a programming model that is efficient enough to apply in real-world scenarios.

In more detail: Section 3.1 presents our design principles for the *UPROP* Python Library. Section 3.2 describes `Uncertain` 's type memory representation of distributional information and what insights we gain from using it. In Section 3.3 we explain how uncertainty propagates through operations and function calls. Finally, in Section 3.4 we demonstrate with examples how users can query uncertainty from `Uncertain` objects and exploit it to their advantage.

3.1 Design Principles

3.1.1 The idea

Existing computer architectures and programming languages lack hardware support and abstractions to treat uncertainty the in a different way than they treat the arithmetic values i.e., with single particle values. Computers are build upon the assumption that no degree of uncertainty lies behing the entirety of the particle values. Most implementations by programmers or existing libraries usually represent uncertainty by using the

mean value of a set of samples or using statistical analysis. The de-facto method – of both the reseach community and the industry, is the Monte Carlo Simulation (Subsection 4.1.1). It constitutes a reliable but computationally expensive method of propagating uncertainty. The Monte Carlo Simulation generally yields more accurate results than other methods, accounting even for the extreme outcomes. The idea behind the *UPROP* Python Library approaches the Monte Carlo simulation but in a determenistic way, requiring less time and operations.

3.1.2 Goals

The *UPROP* Python Library introduces `Uncertain` , a generic data type that encapsulates a probability distribution over a memory representation. The goal of this library is to extend the Python Programming Language [4] in order to allow developers to seamlessly create variables with distributional information. Although the `Uncertain` datatype can support empirical data (e.g., data sampled from sensors) or data derived from well-known parametric distributions (e.g., Gaussian), the main focus is to provide developers mainly managing empirical data, a ① **minimal** and ② **flexible** tool.

- It should allow Python developers to adopt it in their code, **replacing**, when necessary, **the conventional numeric data types** such as `int`, `double` or `float`.
- An instance of an `Uncertain` variable should be able to **represent a Random Variable**.
- The `Uncertain` class should also **overload the majority of Python's base class operators** so that programmers may write the same expressions whilst propagating uncertainty through operations and functions.

The `Uncertain` data type incorporates and manipulates probability distributions of samples. Having samples as input, allows working along with conventional first-order datatypes. It also instigates programmers to account for data uncertainty by giving new semantics for conditional expressions, which propagate uncertainty through computations. `Uncertain` extends and differentiates from some earlier work in a way that focuses on providing a user-friendly interface for non-expert Python programmers to properly reason about uncertainty.

The key insight of the `Uncertain` type is that in order to represent uncertainty as a first-order datatype in memory there has to be some trade-off with regards to accuracy. Furthermore, the nature of a Random Variable limits the amount of operators accessible since we have to make some assumptions about the transition of regular operations to probabilistic operations.

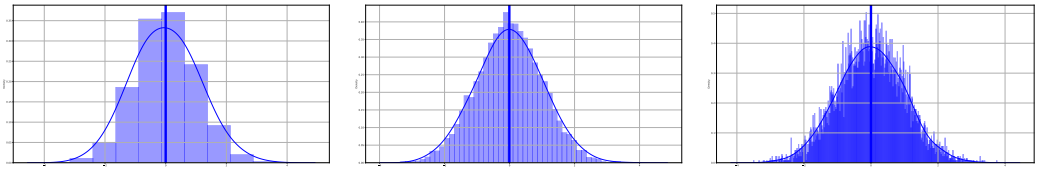
3.2 Representation of distributional information

As we mention in Subection 3.1.2, the focus of the *UPROP* Library is to provide a useful tool to developers mainly managing emprical data (i.e., measurements). Thus, the most accurate approach would be to directly store the samples. But this adds significant performance and memory overhead and quickly becomes impractical. In Subsection 2.1.2 we showcase that a random variable and hence a distribution can be explicitly defined by either providing its' PDF – if the random variable is continuous, its' PMF – if the random variable is discrete or even its' CDF. Thus, the *UPROP* Library represents the input distributions with an approximation of the PDF/PMF in memory by using a **Dirac Mixture Representation** as Tsoutsouras et al. [2] propose.

Definition 3.2.1 ***Dirac mixture representation:*** *Let $\delta(x)$ be the Dirac delta function — a unit impulse at position x. Given a value $x_0 \in \mathbb{R}$, we consider $\delta(x - x_0)$ as a probability mass function and we call it a particle value (or just particle). Using this definition, we transform an array of particle values $x_1, x_2..., x_M$ into a probability mass function using a weighted sum.*

$$f_X(x) = \sum_{n=1}^{M} p_n \delta(x - x_n), \text{ where } p_n \in [0, 1], \sum_{n=1}^{M} p_n = 1 \tag{3.2.1}$$

If the Dirac mixture is equally spaced in the x-axis, then this representation takes the form of a relative frequency histogram. It is clear that the memory size of an `Uncertain` object is determined by the number of Dirac (or bins). A key insight of using this representation is that there is a trade-off between the representation size of an `Uncertain` object and the accuracy of calculations.



(a) Histogram with 10 bins. **(b)** Histogram with 50 bins. **(c)** Histogram with 500 bins.

Figure 3.1: Three different histograms of the same random variable representing a Standard Normal Distribution with $\mu = 0,\ \sigma^2 = 1$. The blue line is the PDF of each distribution.

It's worth noting that employing histograms adds complexity to the process as compared to using just particle values. To transform the samples to histograms requires an algorithm of $\mathcal{O}(N \cdot k)$ time complexity, to sort the N samples into k bins and calculate their frequencies. The **space complexity** of this representation is $\mathcal{O}(k)$. If we opted for storing the samples directly, the corresponding **space complexity** would be $\mathcal{O}(N)$.

3.3 Uncertainty Propagation

The fundamental principle of the `Uncertain` type is capturing uncertainty – even the most extreme outcomes, and propagating it to any operation performed with `Uncertain` objects. Suppose two Random Variables and the corresponding `Uncertain` objects, $X \sim U(0, 6)$ and $Y \sim U(0, 6)$ [1], represented by the `Uncertain` type. The result from their addition $Z = X + Y$ is also an `Uncertain` object representing a Random Variable as we show in Listing 3.1.

```
1 X = Uncertain([i for i in range(0,100)])
2 X = Uncertain([i for i in range(100,200)])
3 Z = X + Y
```

Listing 3.1: Adding two `Uncertain` objects.

There are two takeaways from Listing 3.1:

❶ Data uncertainty is automatically propagated in the backend of the `Uncertain` type.

❷ There are not significant code alterations compared to an implementation with particle values (one-sample values)

[1] U(a,b) represents a uniform distribution in the interval $[a, b]$

We accomplish takeaway ❶ using Equation 2.2.5. This Equation defines the result of on operation between Random Variables which is a circular convolution of their distribution support positions and the corresponding probability masses. In Listing 3.2 we show our implementation of the circular convolution using the memory representation presented in Section 3.2.

```
def addRandomVariables(X1, X2):
    """Calculate the circular convolution of X1 with X2."""
    dst_hist  = []
    dst_medians = []
    for i in range(0,len(X1.hist)):
        for j in range(0,len(X2.hist)):
            dst_hist.append(X1.hist[i]*X2.hist[j])
            dst_medians.append(X1.median[i] + X2.median[j])

    dstVar = createObjectFromOperation(dst_hist, dst_medians)
    return dstVar
```

Listing 3.2: Implementation of the circular convolution between two Random Variables (`Uncertain` objects). The generation of the new `Uncertain` object comes from definition 3.2.1. The variable `dst_hist` holds the probability values p_n and `dst_medians` hold the $(x - x_n)$ distribution support points. We initialize the new object `dstVar` using the generated histogram.

3.3.1 Overloaded Magic or Dunder Methods

Magic methods in Python are the special methods that start and end with the double underscores. They are also called dunder methods. Magic methods are not meant to invoke directly from the user, but the invocation happens internally from the class on a certain action. For example, when the user add two numbers using the + operator, internally, we call the `__add__()` method. Listing 3.3 shows all the attributes and methods of the `int` class. The `Uncertain` class overloads most of the magic methods that the Python template class offers. We present there methods in Subsections 3.3.2, 3.4.1 and 3.4.3.

```
dir(int)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__',
 '__class__', '__delattr__', '__dir__', '__divmod__',
 '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__',
 '__int__', '__invert__', '__le__', '__lshift__', '__lt__',
 '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__',
 '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__',
 '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__',
 '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
 '__subclasshook__', '__truediv__', '__trunc__', '__xor__',
 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag',
 'numerator', 'real', 'to_bytes']
```

Listing 3.3: This Listing shows all the attributes and methods defined in the `int` class. We call the methods that start and end with the double underscores *magic* or *dunder* methods.

3.3.2 Overloaded Arithmetic Operators

We mention in Section 3.1 that the goal of the `Uncertain` type is to replace the conventional datatypes suck as `int`,`float`, ..., etc. In order to accomplish this, the functionality of `Uncertain` has to simulate the functionality of the aforementioned data types. We managed to achieve this by using Object-Oriented Programming (OOP) in Python.

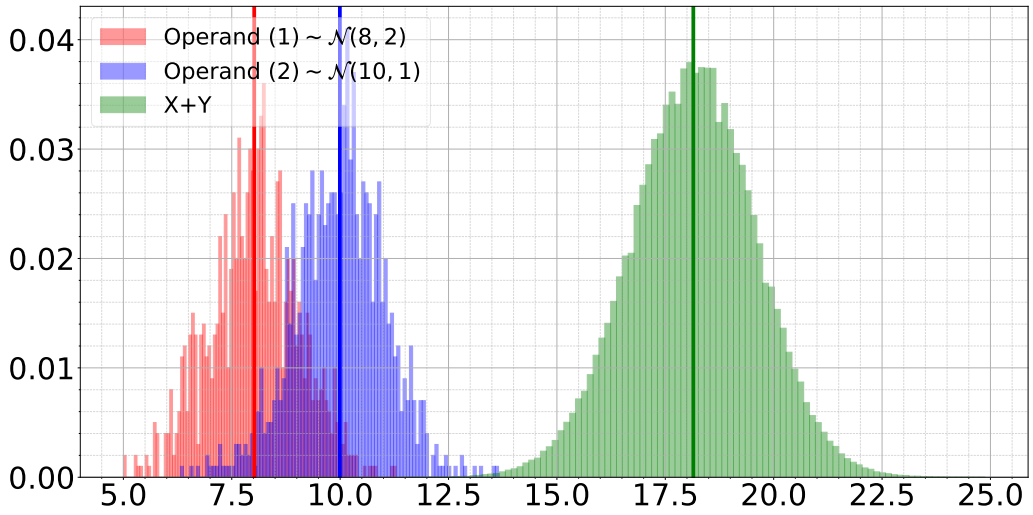


Figure 3.2: Adding two `Uncertain` variables. The 1^{st} operand represents a RV $X \sim \mathcal{N}(8, 1)$ and the 2^{nd} operand a RV $Y \sim \mathcal{N}(10, 1)$. Their addition generates a new distribution $Z \sim \mathcal{N}(18, 2)$, as expected, based on equation 2.2.2.

In Python3 the base class is the `object` class. All classes implicitly inherit from the built-in `object` base class. The `object` class provides some common methods, such as `__init__`, `__str__`, and `__new__`, that can be overridden by any child class. `Uncertain` utilizes this functionality and overloads a variety of operators.

The overloaded operators propagate uncertainty using a circular convolution between `Uncertain` objects as we show in Listing 3.2. In Figure 3.2 we showcase a simple example of the addition of two `Uncertain` objects. In Tables 3.1-3.3 we present the list of `Uncertain` 's overloaded operators. Columns `type(a)` and `type(b)` show the supported operand types when calling the corresponding operator. The return type of the overloaded operators is an `Uncertain` object.

Method	Usage	type(a)	type(b)
__neg__	-a	Uncertain	-
__add__	a+b	Uncertain	Uncertain | Number
__sub__	a-b	Uncertain	Uncertain | Number
__mul__	a*b	Uncertain	Uncertain | Number
__truediv__	a/b	Uncertain	Uncertain | Number
__pow__	a**b	Uncertain	Uncertain | Number
__mod__	a%b	Uncertain	Uncertain | Number
__divmod__	divmod(a,b)	Uncertain	Uncertain | Number

Table 3.1: Overloaded arithmetic operator methods.

Method	Usage	type(a)	type(b)
__iadd__	a += b	Uncertain	Uncertain | Number
__isub__	a -= b	Uncertain	Uncertain | Number
__imul__	a *= b	Uncertain	Uncertain | Number
__itruediv__	a /= b	Uncertain	Uncertain | Number

Table 3.2: Overloaded in place arithmetic operator methods.

The implementation of the circular convolution raises a dilemma, *aim for increased accuracy in expense of larger memory usage or aim for a trade-off between them?*. The chosen implementation of the convolution has space complexity of $\mathcal{O}(k * n)$, where k is the number of bins of the 1^{st} operand and n is the number of bins of the 2^{nd} operand. After a small number of operations and function calls, this quickly becomes impractical in terms of memory usage. Thus, to maintain a reasonable amount of

memory usage, we decided to keep the **same number of bins after each operation**. If $k > n$ then we opt to keep k bins.

Method	Usage	type(a)	type(b)
__rsub__	a-b	Number	Uncertain
__radd__	a+b	Number	Uncertain
__rmul__	a*b	Number	Uncertain
__rtruediv__	a/b	Number	Uncertain
__rpow__	a**b	Number	Uncertain
__rmod__	a%b	Number	Uncertain
__rdivmod__	divmod(a,b)	Number	Uncertain
__rfloordiv__	a//b	Number	Uncertain

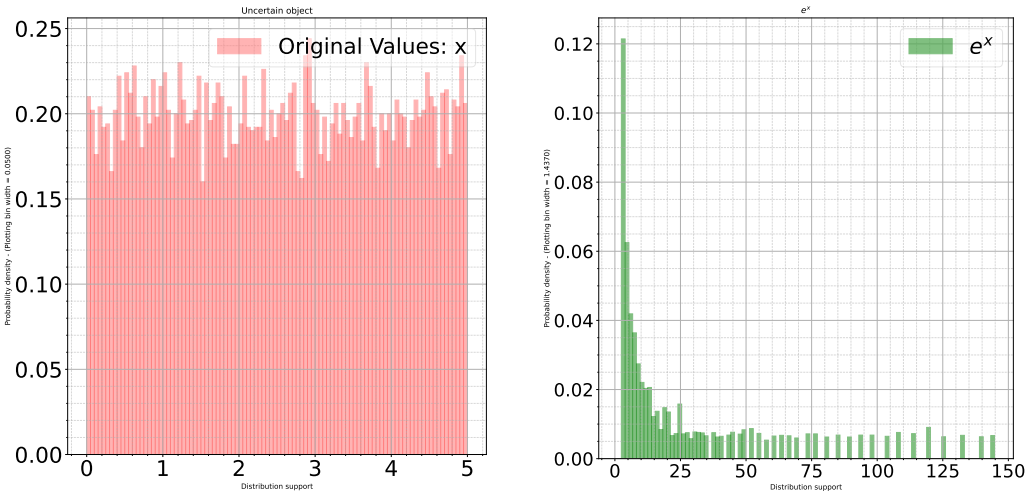
Table 3.3: Overloaded arithmetic operator methods with the preceding "r" e.g., __radd__ are only called if the left operand (a) does not support the corresponding operation and the operands are of different types.

3.3.3 Mapping Functions to `Uncertain` objects

A significant milestone for the `Uncertain` type to simulate numeric types is to be able to support all function calls using `Uncertain` objects. Python's `map()` is a built-in function that allows the processing and transformation of all the items in an iterable without using an explicit `for` loop. We use this function to our advantage and `map` the `Uncertain` 's distribution support points to a user-defined function. We implement this functionality in the `u_map()` function. The new mapped values create a new `Uncertain` object. Listing 3.5 shows our implementation of `u_map()` while Listing 3.4 presents an example using this function.

```
1 from UPROP import Uncertain, u_map
2 from math import exp
3
4 samples = [i for i in range(0,100)]
5 x = Uncertain(samples)
6 newX = u_map(x, exp)
```

Listing 3.4: Example of using the `u_map` function on an `Uncertain` object with function `math.exp`: e^x.



(a) The input `Uncertain` object: $X \sim U(0, 5)$

(b) The output `Uncertain` object: e^x generated by the `u_map` method.

Figure 3.3: Results of the example in Listing 3.4. Figure 3.3a represents input and Figure 3.3b is the output of the `u_map()` method.

```
1  def u_map(obj, func: Callable, **kwargs) -> Uncertain:
2      """Map function <func> to the <obj> Uncertain object"""
3      new_medians = map(lambda x: func(x, **kwargs), obj._medians)
4      """
5          Since the ordering of the new mapped values has not
6          changed, compared with the original medians the histogram
7          values in the corresponding positions represent the
8          probablity of each new median.
9      """
10     dstVar = createObjectFromOperation(obj.hist, new_medians)
11     return dstVar
```

Listing 3.5: `Uncertain` 's `u_map` implementation: mapping a function to an `Uncertain` object.

3.4 Extracting uncertainty information

To unfold the full potential of the `Uncertain` type, we have to provide access and insights to the user about the *uncertainty* of any `Uncertain` object used. Following every initialization and operation, with `Uncertain` objects, we calculate and store the most significant distributional information related to them. This allows the user to make an informed decision

based on uncertainty propagation from the input. We present the available distributional information to query in table 3.4.

Method	Purpose	Type of `val`
`mean`	$\mu = \mathbb{E}[X]$	-
`variance`	$\mathbb{E}[(X-\mu)^2]$	-
`min()`	X_{min}	-
`max()`	X_{max}	-
`mode()`	$x_i : P(X = x_i) \geq P(X = x_j)\ i \neq j$	-
`NthMoment(N)`	$\sigma_N = \mathbb{E}[(X-\mu)^N]$	-
`prob(val)`	$P(X = \texttt{val})$	`Number`
`CDF(val)`	$F_X(x = \texttt{val})$	`Number`
`QF(val)`	$F_X^{-1}(x = \texttt{val})$	`Number`
`probabilityEQ(val)`	$P(X = \texttt{val})$	`Number`
`probabilityNE(val)`	$P(X \neq \texttt{val})$	`Number`
`probabilityLT(val)`	$P(X < \texttt{val})$	`Number`
`probabilityLE(val)`	$P(X \leq \texttt{val})$	`Number`
`probabilityGT(val)`	$P(X > \texttt{val})$	`Number`
`probabilityGE(val)`	$P(X \geq \texttt{val})$	`Number`

Table 3.4: `Uncertain` 's available methods to extract uncertainty.

Although centralized moments do not always uniquely identify probability distributions [5], the developer can have a significant amount of information by calculating a series of them (e.g., mean, variance, skewness, kurtosis, etc). For example, using this information could provide consistent decisions and maintain stability to a control system.

3.4.1 Conditional operators

Complementary to the `u_map()` method, the necessary functionality to achieve closure in simulating numeric types are the conditional operators $<, \leq, =, \neq, \geq$ and $>$. We present the implemented comparison operators in Table 3.5. We discern two subcategories: (i) comparing an `Uncertain` object with a constant number and (ii) comparing two `Uncertain` objects.

Let X be a random variable and its' expected value $\mathbb{E}[X]$. If we compare $\mathbb{E}[X]$ with a desired constant "`val`", the result can be misleading, which leads to false positives [3]. Instead, we allow the user set a probability threshold, and we compare that, with the portion of probability that is on the left or on the right of `value`. Figure 3.4 visualizes the *Probability of the* `Uncertain` *object* X *being* $\leq 8 \rightarrow P(X \leq 8)$.

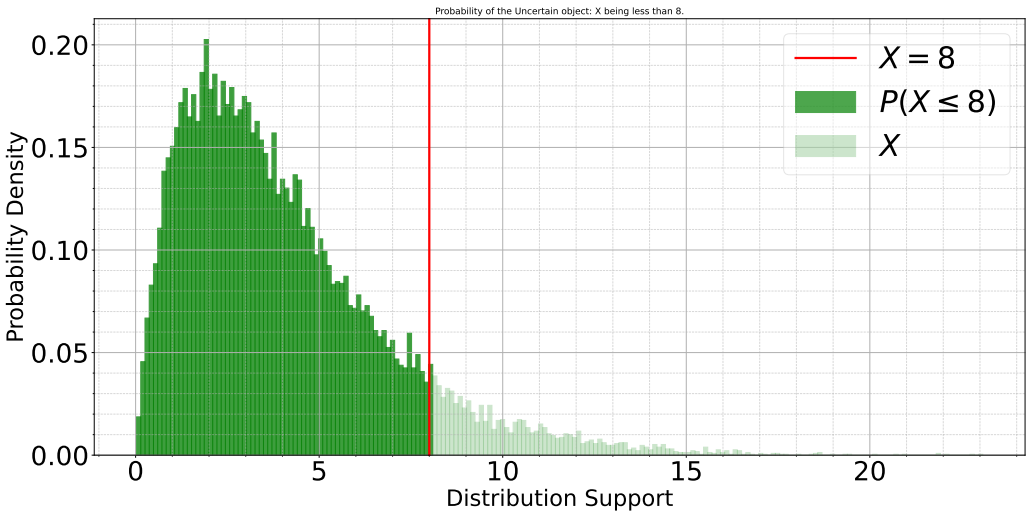


Figure 3.4: The probability of an `Uncertain` object being $X \leq$ `value` is the sum of the probabilities of the bins being $\leq$ `value`. In this case, it is the area on the left of the line, under the darker green bins.

Goodness-of-fit tests favour operators $\neq, =$ but not the rest. There are numerous goodness-of-fit tests, particularly tests for the one-, two- and K-sample problems. The principle behind the one-sample tests states that the testing a hypothesis that the sample observations have a hypothesised distribution, whereas the two-sample problem is concerned with testing the equality of the distributions of two independent samples. A statistical test is just a formal way to make a decision between two mutually exclusive hypotheses, the null hypothesis is usually well-defined and concerns the hypothesised distribution, whereas the alternative hypothesis is broad, and usually just constitutes the negation of the null hypothesis. Thus, when we reject the null hypothesis, the majority of tests do not give any information about what the true distribution might look like, or how the true distribution differs from the hypothesised. The same reasoning holds for the two-sample problem, when we reject the null hypothesis of equality of the two populations, there is often no information about how the two distributions disagree [27].

Under the aforementioned conditions and since for the equality ($=$) and negation ($\neq$) operators rejecting or accepting the null hypothesis suffices, **we implement these two using the Kolmogorov–Smirnov two-sample Test** [6]. For the rest, we provide two methods that the user can select during runtime: (i) **Stochastic Dominance** and (ii) ***Difference of Random Variables***. In order to select method (i) or method (ii), the developer has to define a member variable of the `Uncertain` object,

either during initialization of at runtime as we show in Listing 3.6.

Method	Usage	type(a)	type(b)
`__eq__`	`a == b`	`Uncertain`	`Uncertain | Number`
`__ne__`	`a != b`	`Uncertain`	`Uncertain | Number`
`__ge__`	`a >= b`	`Uncertain`	`Uncertain | Number`
`__gt__`	`a > b`	`Uncertain`	`Uncertain | Number`
`__le__`	`a <= b`	`Uncertain`	`Uncertain | Number`
`__lt__`	`a < b`	`Uncertain`	`Uncertain | Number`

Table 3.5: Overloaded `Uncertain` 's conditional operators.

```
X = Uncertain(samples1, comparison_method="difference")
X.set_comparison_method("dominance")

Y = Uncertain(samples2)

result: bool = X > Y
```

Listing 3.6: The developer can choose the desired comparison method by calling the `set_comparison_method()` method with either "dominance" or "difference" as argument. "dominance" is the default value during initialization.

Method 1: Stochastic Dominance

As we mention in Subsection 2.3.1, Stochastic Dominance comparing the CDF of Random Variables is a common method to obtain a partial order between them. We present the implementation in pseudocode in Listing 3.7.

```
def stochastic_dominance_le(X,Y):
    for i in joint_distribution_support(X,Y):
        if X.CDF(i) >= Y.CDF(i):
            # partial ordering remains the same
            continue
        else:
            # found a value for which ordering reverses
            return False
    return True
```

Listing 3.7: Pseudocode of the implementation of Stochastic Dominance for the $\leq$ operator. For each value in the distribution support range we compare the CDFs and infer the result.

Method 2: Difference of Random Variables

The *difference of Random Variables* method is equivalent to comparing an `Uncertain` object to a constant number. The principle behind the method originates from the transposition property of any comparison operator: $X > Y \Rightarrow X - Y > 0$. We present our implementation in Listing 3.8.

```
def difference_rv_gt(X: Uncertain, Y: Uncertain):
    Z : Uncertain = X-Y
    max_threshold = max(X.threshold, Y.threshold)
    return Z.probabilityGT(0) > max_threshold
```

Listing 3.8: The implementation of the *difference of random variables* method is equivalent to generating a new `Uncertain` object Z from the difference of two `Uncertain` objects X, Y and comparing the result with the user defined threshold.

For the case of comparing two `Uncertain` objects we present extensive testcase in Section 4.2.

3.4.2 Bounding uncertainty

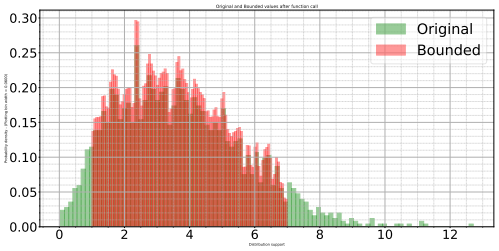
A characteristic of the `Uncertain` type is that it manages to propagate information about the less-likely outcomes of Random Variables (i.e., long-tailed distributions) through calculations. In extreme scenarios, this feature can turn out useful, but in other cases, this information may be redundant. This should be application-dependent. We provide the developers with the member function `bound()` that addresses this issue.

```
samples = rayleigh(scale=3,size=10000)
X = Uncertain(samples=samples,bins=100)
X1 = X.bound(lower=1, upper=7)
X2 = X.bound(lower=1, midlower=3, midupper=5, upper=7)
```

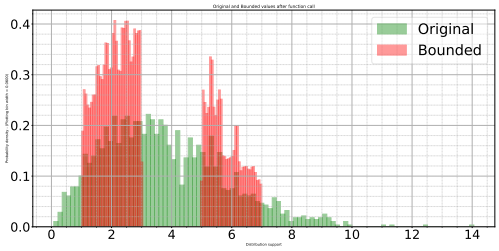
Listing 3.9: Using the public `bound()` method a user can define the lower and upper allowed limit for the variable. We show the results of this Listing in Figure 3.5.

With this method, the developer can bound an `Uncertain` object inside a range of values. Basically the `bound()` member function gives the developers the option to effectively **low-pass**, **high-pass or band-pass** filter the `Uncertain` object. In specific cases, the developer can utilize prior knowledge about any of the `Uncertain` objects and limit the values

accordingly. When bounding an `Uncertain` variable X with k number of bins, the returned object Y has the same number of bins. Although the `bound` method may result in increased calculation overhead, it limits data uncertainty. This can be valuable in decision making. In Listing 3.9 we present a usage example of the method `bound()` and it's results visualized in Figure 3.5.



(a) We can bound `Uncertain` variables inside the range [`lower, upper`]. In this example, we bound the `Uncertain` variable in the range $[1, 7]$



(b) We can also bound `Uncertain` variables inside the range [`lower, midlower`] $\cup$ [`midupper, upper`]. In this example we bound the `Uncertain` variable in the range $[1, 3] \cup [5, 7]$.

Figure 3.5: Bounding an `Uncertain` variable using the public `bound()` method. The original variable represents a Rayleigh distribution with $\sigma = 3$ and we generate the bounded after calling `bound()`. We show the implementation of this example in Listing 3.9.

3.4.3 Other overloaded magic methods

As we show in Subsection 3.3.1, the Python template class offers a variety of magic methods that the `Uncertain` type overloads. In Subsections 3.3.2 and 3.4.1 we showcase the arithmetic and conditional overloaded operators. In this Subsection, we list overloaded magic methods with general functionality that will ease the usage of the **UPROP** library for the developers.

```
>>> from UPROP import Uncertain
>>> bool(Uncertain([0,0,0]))  # __bool__()
>>> False
>>> bool(Uncertain([0,0,1]))  # __bool__()
>>> True
```

Listing 3.10: Using the `__bool__()` overloaded method. Following the behavior of `int, float`,etc, the `Uncertain __bool__()` method returns `False` only if the `mean` value is 0.

A significant magic function that Python offers is the `__bool__()` magic function. This function returns the boolean equivalent of any

object. For example `bool(1)` is `True` while `bool(0)` is `False`. In accordance with the default Python behavior we overload this function and the `Uncertain __bool__()` method returns `False` only if the `Uncertain` object's `mean` value is 0. In Table 3.6 we introduce some the implemented magic methods along with minor usage examples Listings 3.10, 3.11 and 3.12.

```
>>> from UPROP import Uncertain
>>> X = Uncertain([[0,1,2,3,4], bins=100)
>>> len(X)  # __len__()
>>> 100
>>>
>>> 2 in X  # __contains__()
>>> True
```

Listing 3.11: Using the `__len__()` overloaded method yields the number of bins that an `Uncertain` object has. Method `__contains__()` returns `True` if the argument is inside the R_X range.

```
>>> from UPROP import Uncertain
>>> X = Uncertain(samples=[1,2,3,4,5,6,7,8])
>>> X  # __repr__()
>>> Uncertain variable:
        Mean: 4.5
        Variance: 3.828125
        Bins: 4
        Bin Width: 1.75
>>> print(X)  # __str__()
>>> 4.5
```

Listing 3.12: Using the `__str__()` overloaded method prints the expected value of X: $\mathbb{E}[X]$. The method `__repr__()` is an unambiguous way to represent the characteristics of each Uncertain variable.

Method	Usage	type(a)	type(b)	Return type
__bool__	if (x)	Uncertain	-	bool
__contains__	b in a	Uncertain	Number	bool
__str__	str(a)	Uncertain	-	-
__format__	f"{a:b}"	Uncertain	str	str
__int__	int(a)	Uncertain	-	int
__float__	float(a)	Uncertain	-	float
__len__	len(a)	Uncertain	-	int
__repr__	print(a)	Uncertain	-	str

Table 3.6: Overloaded magic methods of the `Uncertain` type derived from Python's template class. In the column **Method** we list the method, followed by an example in column **Usage**, the **type** of operands, as well as the **Return type**.

3.4.4 Plotting `Uncertain` objects

To assist in extracting uncertainty from `Uncertain` objects we implement the `plot` method. We wrap the widely used `matplotlib.pyplot.hist()` with the `plot()` method to visualize any `Uncertain` variables. The user has the option to place the histogram on the current axis or provide a different one. The two axes x, y contain information for the distribution support R_X and probability density (or frequency) respectively. We generate the entirety of the diagrams and plots used in this thesis with the `plot()` method. We provide usage examples in Listings 3.13, 3.14 and the respective results in Figures 3.6 and 3.7. The parameters that of the `plot()` method are:

1. `ax`: <plt.Axes> which axis to plot on, The default is: `plt.gca()`.
2. `color`: <`str`> Fill color of the histogram.
3. `xlabel_flag`: <`bool`> Set the xlabel defined by `Uncertain` if set to `True` or leave empty if set to `False`. The label is: *"Distribution Support"*
4. `ylabel_flag`: <`bool`> Set the ylabel defined by `Uncertain` if set to `True` or leave empty if set to `False`. The label is: *"Probability Density - Plotting width = _"*
5. `label`: <`str`> Label of the data.
6. `mean`: <`bool`> Plot the mean value as a black vertical line if set to `True`.
7. `density`: <`bool`> Plot the `Uncertain` variable as a density histogram if set to `True` or as a Frequency histogram if set to `False`.
8. `kwargs`: Keyword arguments
   (a) `alpha`: <`float`> Alpha value of the histogram.
   (b) `edgecolor`: <`str`> The edgecolor of the histogram bars.
   (c) `fill`: <`bool`>: Fill the bars with color if set to `True` of leave empty set to `False`.
   (d) `hatch`: <`str`>: The fill pattern to fill the bars. Used only if `fill` is `True`.
   (e) `hist`: <`bool`> plot the histogram if set to `True` or plot the Kernel Density Estimation if set to `False`.

(f) `kde`: <`bool`>: Plot the Kernel Density Estimation (Estimated PDF), if set to `True`.

(g) `kwargs`: Any other `matplotlib.pyplot` Keyword arguments.

```
from UPROP import Uncertain
import numpy as np
import matplotlib.pyplot as plt

norm = np.random.default_rng().normal(10,5,5000)
X = Uncertain(norm, bins=50)
X.plot(color='b', alpha=.3,
       label="X", fill=True,
       hatch="*", edgecolor='b',
       ylabel_flag=True,
       xlabel_flag=True)

plt.grid()
plt.legend()
plt.show()
```

Listing 3.13: Using the `Uncertain` 's `plot()` method to visualize an `Uncertain` variable $X \sim \mathcal{N}(10, 5)$ with a star(*) hatch.

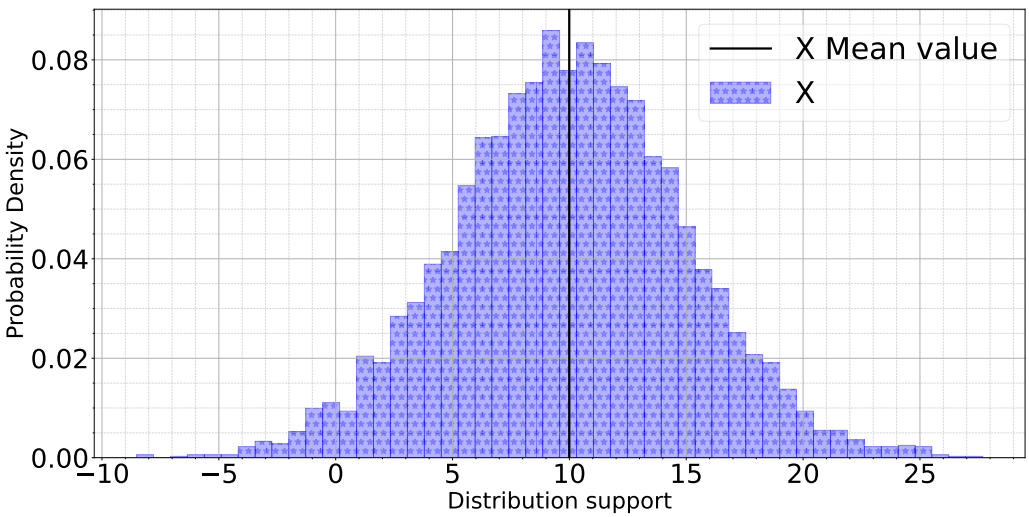


Figure 3.6: Result of Listing 3.13 using the `Uncertain` 's `plot()` method to visualize an `Uncertain` variable $X \sim \mathcal{N}(10, 5)$ with a star(*) hatch.

```python
from UPROP import Uncertain
import numpy as np
import matplotlib.pyplot as plt

norm = np.random.default_rng().normal(10,5,5000)
uni = np.random.default_rng().uniform(15,40,5000)

X = Uncertain(norm, bins=50)
X.plot(color='r', alpha=.5,
       label='X')
Y = Uncertain(uni, bins=50)
Y.plot(color='b', alpha=.5,
       label='Y', mean=False,
       ylabel_flag=True,
       xlabel_flag=True)

plt.grid()
plt.legend()
plt.show()
```

Listing 3.14: Using the `Uncertain` 's `plot()` method to visualize two `Uncertain` variables $X \sim \mathcal{N}(10, 5)$ and $Y \sim U(15, 40)$ using different parameters for each one.

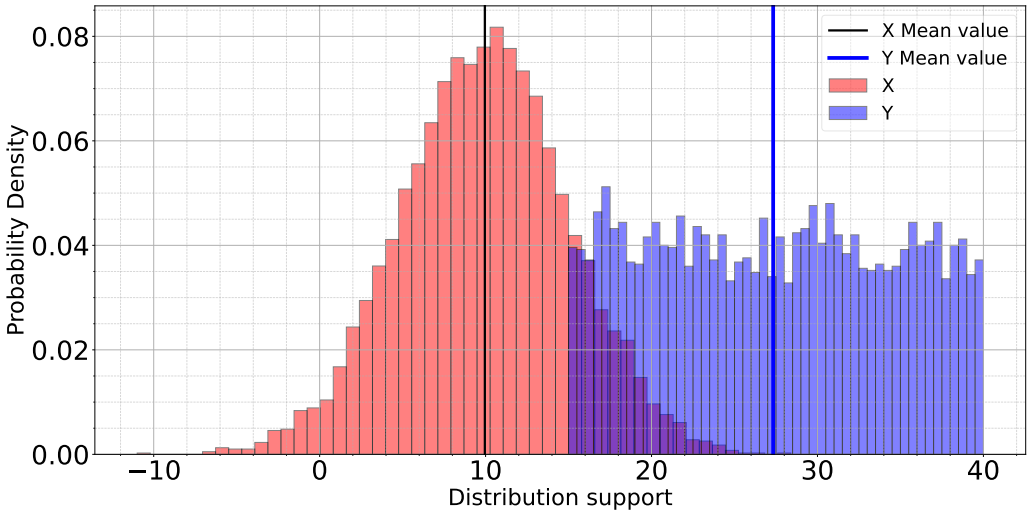


Figure 3.7: Result of Listing 3.14 using the `Uncertain` 's `plot()` method to visualize two `Uncertain` variables $X \sim \mathcal{N}(10, 5)$ and $Y \sim U(15, 40)$ using different parameters for each one.

Chapter 4

Evaluation

In Chapter 3 we discuss the implementation of the *UPROP* Library, following the specified design principles in Section 3.1. In this Section we experimentally evaluate the usage of the *UPROP* Library in real world applications. In Section 4.1 we present the necessary information and evaluation metrics needed to understand the rest chapter. We commence with micro benchmarks in Section 4.2 by evaluating the overloaded arithmetic and conditional operators. We extend our test suite with two applications manipulating experimental measurements: (i) Speed estimation from uncertain GPS coordinates and (ii) A clap detection algorithm from recorded sounds. We finally proceed to compare the *UPROP* Library with the state-of-the-art and rate it's performance.

4.1 Evaluation Introduction

4.1.1 Monte Carlo Simulation

The golden stantard method for uncertainty propagation in the research and industry community is the Monte Carlo Simulation (MCS). It is a mathematical technique, which we use to estimate the possible outcomes of an uncertain event. The Monte Carlo Simulation is basically a family of computational algorithms that rely on repeated random sampling to obtain certain numerical results, and we use it to solve problems that have a probabilistic interpretation [28].

Monte Carlo Simulations works by selecting a random value for each task, and then building models based on those values. This process is then repeated multiple times, with different values so in the end, the output is a distribution of outcomes. In Listing 4.1 we present the Monte

Carlo Simulation for sampling a Multivariate Gaussian distribution of the 2-dimensional random vector $\mathbf{X} = (X_1, X_2)^{\mathrm{T}}$ $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Figure 4.1 we plot the resulting Multivariate Gaussian.

```
MONTE_CARLO_MAX_SAMPLES = 5000
gaussian_2D = []
for _ in range(0, MONTE_CARLO_MAX_SAMPLES):
    sample_1 = random.normal(size=1)
    sample_2 = random.normal(size=1)
    gaussian_2D.append([sample_1, sample_2])
```

Listing 4.1: Sampling a Multivariate Gaussian Distribution using the Monte Carlo Simulation.

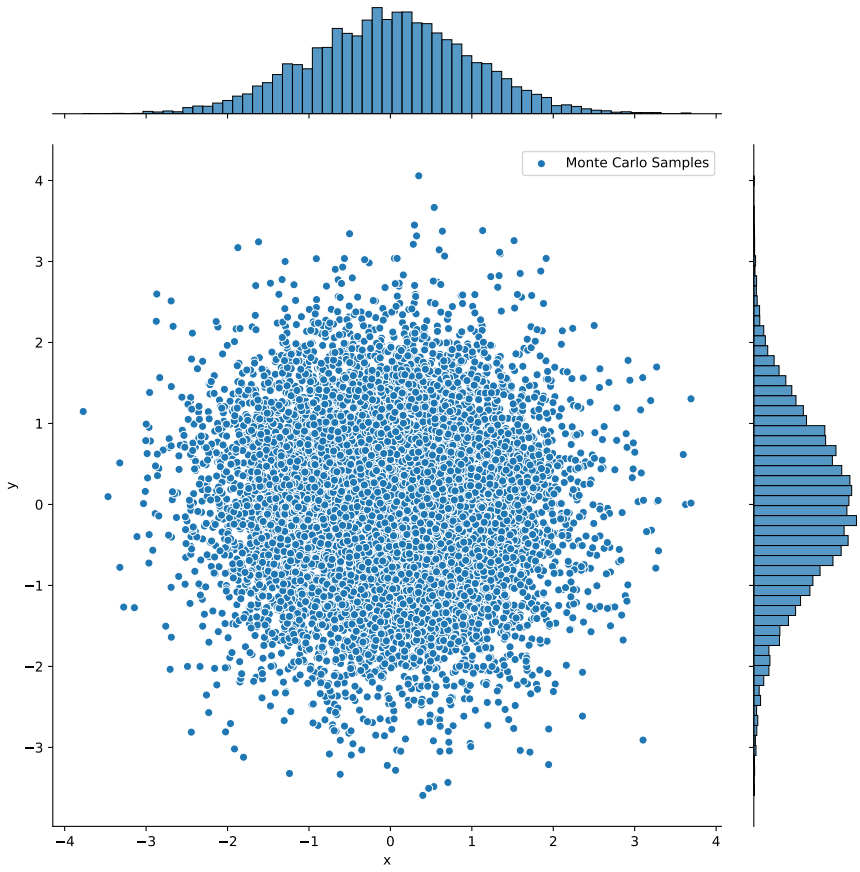


Figure 4.1: Sampling a Multivariate Gaussian with Monte Carlo Simulation.

4.1.2 Normalized Wasserstein distance

In Section 2.3 we present the Wasserstein Distance metric, which is a method to evaluate divergence between two multi-dimensional distributions. In this section we present an alternative usage of the Wasserstein distance to compare two frameworks with distributional output.

As we mention in Subsection 2.3.3, when we calculate the W_1 distance between two distributions, we calculate the area between the two distributions' CDF curves. Thus the calculated distance value is strongly bonded to the distribution support R_X of each distribution. In our case, we desire a uniform metric value for the entirety of our tests. To do so, we have to detach the distance from the distribution support. We therefore introduce the Normalized Wasserstein distance:

Definition 4.1.1 *Let F_a, F_b, F_{ref} be three cumulative distribution functions for three random variables. Let W_1 be the Wasserstein-1 distance. We define the Normalized Wasserstein Distance as the ratio of two W_1 distances:*

$$\mathrm{NWD} = \frac{W_1\left(F_a, F_{ref}\right)}{W_1\left(F_b, F_{ref}\right)} = \frac{\int_{\mathbb{R}} \left|F_a(x) - F_{ref}(x)\right| dx}{\int_{\mathbb{R}} \left|F_b(x) - F_{ref}(x)\right| dx} \tag{4.1.1}$$

The NWD takes values in the range: $[0, \infty)$ with values closer to 0 meaning that F_a has a smaller Wasserstein distance to F_{ref} than F_b has to F_{ref}. The NWD enables us to compare the distributional output of the *UPROP* Library with the particle value calculation method (conventional method) or any other framework.

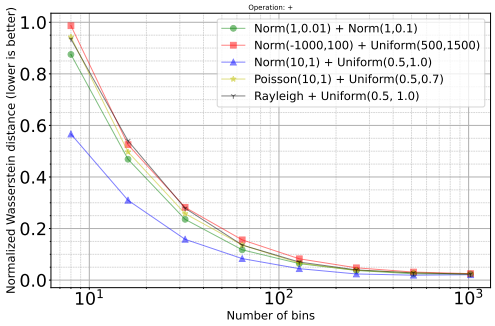
In this thesis we use the **inverse Normalized Wasserstein distance** (NWD^{-1}) to denote the accuracy gain of the *UPROP* Library, over the particle method calculation, using the Monte Carlo Simulation as the baseline. We consider the MSC as the *golden truth*. In Sections 4.2, 4.3, and 4.5 we calculate the Wasserstein distance of MSC (F_{ref}) for both the UPROP Python Library (F_a) and the particle method calculation (F_b) and proceed to calculate their inverse Normalized Wasserstein distance.

4.2 Operators evaluation

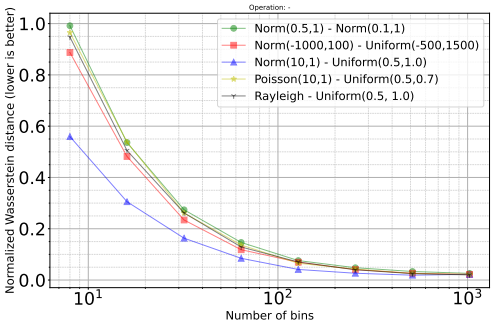
The fundamental goal of this research is to make it feasible for developers to use the *UPROP* Library instead of the conventional numeric types such as `int`, `float`, etc. As we mention earlier in Chapter 3 it is of utter importance that we fully implement and test the basic: **The arithmetic and comparison operators**.

4.2.1 Arithmetic Operators

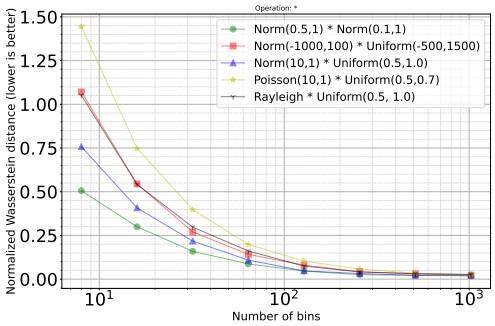
To test the arithmetic operators we performed a series of simple operations between Random Variables representing specific distributions. For example we **add**, **subtract**, **multiply and divide** two Random Variables: $X_1 \sim \mathcal{N}(10, 1)$ and $X_2 \sim \mathcal{U}(0.5, 1.0)$ using the *UPROP* Library. We then perform the same calculation with the Monte Carlo Simulation and measure their Normalized Wasserstein distance. Figure 4.2 shows the visualized results for each operation and Table 4.1 shows the mean values obtained for each number of bins across all benchmarks. The **minimum accuracy gain** over the particle sample calculation, is **is** $1.03\times$ for 8 bins and the **maximum is** $44.62\times$ for 1024 bins.



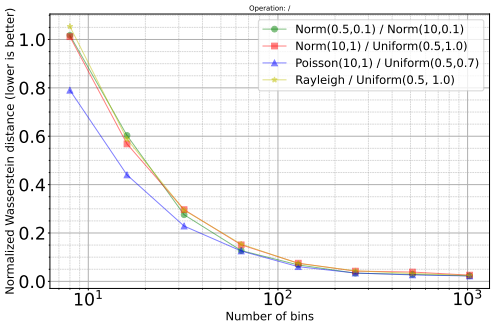
(a) Normalized Wasserstein distance of addition operator (+).



(b) Normalized Wasserstein distance of subtraction operator (-).



(c) Normalized Wasserstein distance of multiplication operator (*).



(d) Normalized Wasserstein distance of division operator (/).

Figure 4.2: This Figure shows the Normalized Wasserstein distance (y-axis) between the distributionl output of the *UPROP* Library and the particle sample calculation. We measure this distance from the Monte Carlo Simulation. Lower is better. We condition each operator to a series of tests, and we perform each test for a different number of bins (x-axis). We notice that the greater the number of bins, the higher the accuracy.

The selection of the representation size (number of bins) embodies the classic speed-accuracy trade-off. At small bin numbers, the calculation

is quick, but more inaccurate. At large bin numbers, the reverse is true: the calculation is slow but accurate. Choosing the correct bin number is therefore critically important: too high and the `Uncertain` type will be too slow for practical use; too low and it will be too inaccurate to solve real problems.

Bins\Operation	Add	Sub	Mul	Div
8	1.16	1.15	1.03	1.03
16	2.14	2.12	1.23	1.81
32	4.12	4.18	3.72	3.65
64	7.95	8.13	7.17	7.17
128	15.28	15.51	14.25	14.39
256	26.82	25.24	25.22	25.92
512	38.15	38.72	35.94	31.54
1024	**42.94**	**44.61**	**42.98**	**41.98**

Table 4.1: Mean inverse-Normalized Wasserstein distance between the Particle and the *UPROP* Library implementation for the arithmetic operators for each number of bins.

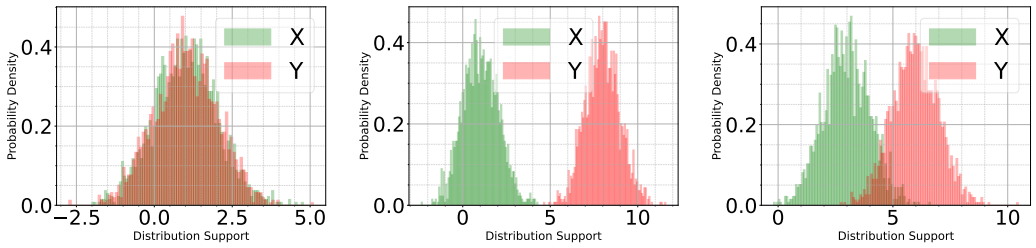
4.2.2 Conditional Operators

The second major milestone to achieve with the *UPROP* Library is to provide developers the ability to extract or query uncertainty from `Uncertain` objects. This allows the user to make an informed decision based on the propagated uncertainty. As we mention in Section 3.4.1, we implement the equality(= or `==`) and non-equality (≠ or `!=`) operator using the Kolmogorov Smirnov two-sample test [6]. For the rest operators we provide the developer with two methods which he can perform conditional queries on `Uncertain` objects:

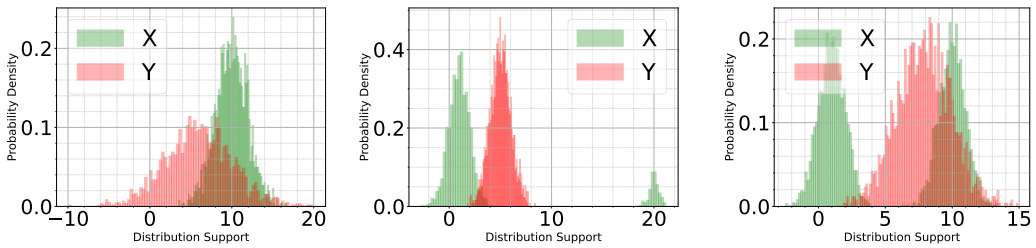
1. **Stochastic Dominance** and

2. ***Difference of Random Variables***.

To test each method extensively we provide a set of Random Variables representing specific distributions. We plot the PDF of these distributions in Figure 4.3 and the CDF in Figure 4.4. For each method we create a truth table (Table 4.2 and Table 4.3) to showcase the boolean results yielded from each conditional operator.

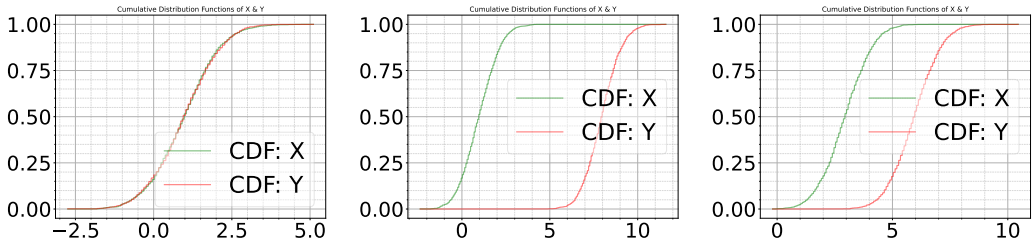


(a) PDF - Testcase 1 **(b)** PDF - Testcase 2 **(c)** PDF - Testcase 3

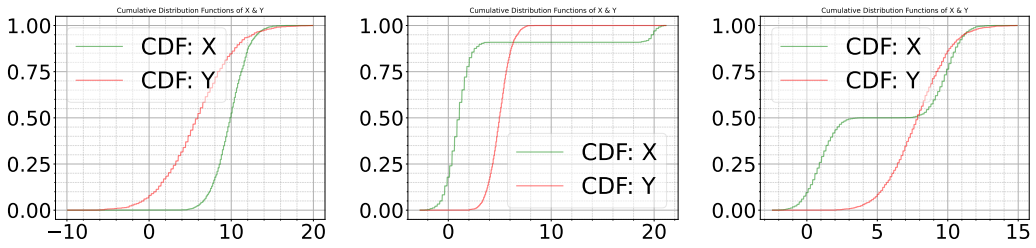


(d) PDF - Testcase 4 **(e)** PDF - Testcase 5 **(f)** PDF - Testcase 6

Figure 4.3: Probability Density Functions of two random variables used to test conditional operators and infer a partial order.



(a) CDF - Testcase 1 **(b)** CDF - Testcase 2 **(c)** CDF - Testcase 3



(d) CDF - Testcase 4 **(e)** CDF - Testcase 5 **(f)** CDF - Testcase 6

Figure 4.4: Cumulative Density Functions of two random variables used to test conditional operators and infer a partial order. As we show in equation 2.3.1 we compare the two CDF to reason about a partial ordering when a conditional operator is used.

	$X > Y$	$X \geq Y$	$X == Y$	$X \neq Y$	$X < Y$	$X \leq Y$
Testcase 1	False	***True***	True	False	False	***True***
Testcase 2	False	False	False	True	True	True
Testcase 3	False	False	False	True	True	True
Testcase 4	False	False	False	True	False	False
Testcase 5	False	False	False	True	***False***	***False***
Testcase 6	False	False	False	True	False	False

Table 4.2: Conditional operators results for Testcases 1-6, using *Stochastic Dominance* method. The values with the asterisk * note the different result between the *RV-Difference* method and *Stochastic Dominance* method.

We notate the differences between Tables 4.2 and 4.3 with a star $*$. The *Stochastic Dominance* methods basically requires the sign of I:

$$I = \text{CDF}(X = k) - \text{CDF}(Y = k), \forall k \in R_X \bigcup R_Y$$

to not change at any value k:

$$\text{sng}\, I = 1, \forall k \quad \text{or} \quad \text{sng}\, I = -1, \forall k$$

with sng beeing the sign function:

$$\text{sgn}\, x := \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$

	$X > Y$	$X \geq Y$	$X == Y$	$X \neq Y$	$X < Y$	$X \leq Y$
Testcase 1	False	***False***	True	False	False	***False***
Testcase 2	False	False	False	True	True	True
Testcase 3	False	False	False	True	True	True
Testcase 4	False	False	False	True	False	False
Testcase 5	False	False	False	True	***True***	***True***
Testcase 6	False	False	False	True	False	False

Table 4.3: Conditional operators results for Testcases 1-6, using *RV-Difference* method. The values with the asterisk * note the different result between the *Stochastic Dominance* method and *RV-Difference* method.

Of course, the results of Table 4.3 the user-defined comparison threshold we mention in Subsection 3.4.1 affects the result. **We find that by lowering the threshold we obtain more *True* results.**

4.3 Speed calculation from uncertain GPS coordinates

A common problem in our everyday life is inaccurate navigation in mobile applications that utilize the Global Positioning System (GPS) [29]. A satisfactory navigation service requires an accurate positioning technology. Most modern smartphones have GPS sensors to estimate the user's location, and an abundance of smartphone applications consume this location estimate through APIs. Even though the current smartphones have integrated multiple sensors, such as Global Navigation Satellite System receiver, gyroscope, accelerometer and magnetometer sensors, the performance of positioning in even in urban environments is still not accurate. The reasons of the errors include GNSS signals reflections, high dynamic of pedestrian activities and disturbance of the magnetic field in city environments.

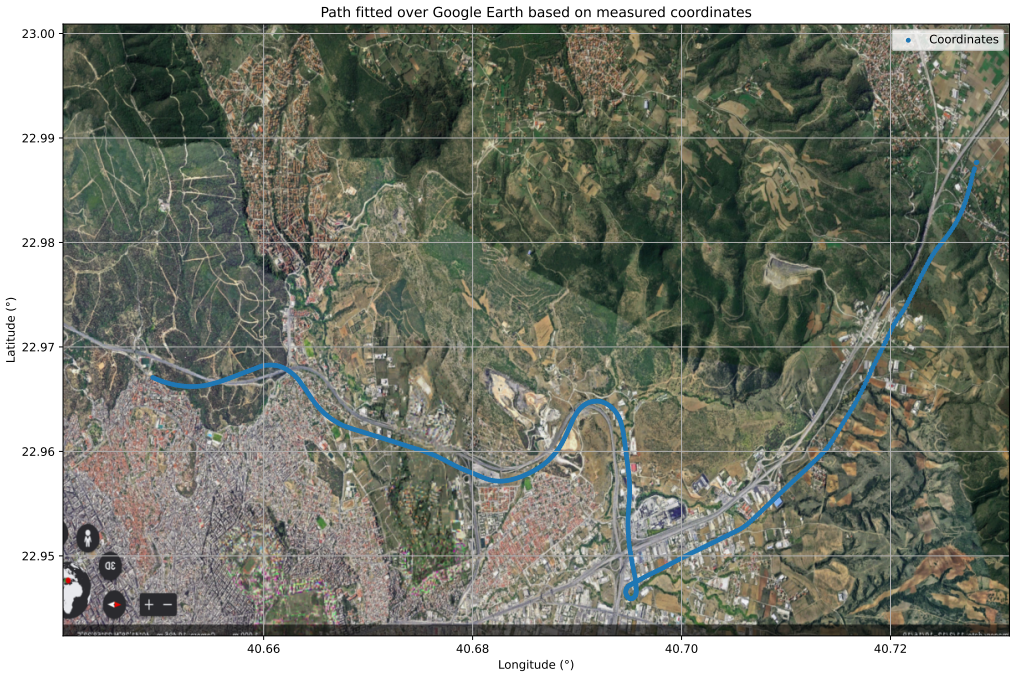


Figure 4.5: GPS Coordinates measured through the PhyPhox [7] mobile application, approximately fitted on top of Google Earth [30].

Bornholt [3] shows that using measurements, from sensors, as facts could lead to false positives and or false negatives. There have been attempts to improve the performance of GPS usage. Whang et al [31]

suggest the utilization of the camera sensor for improving the accuracy of the positioning. Ofstad et al [32] propose accelerometer augmented mobile phone localization (AAMPL), a system that uses accelerometer signatures to place mobile phones in the right context.

To demonstrate the feasibility of calculations using `Uncertain` objects we implement an application using the Global Positioning System (GPS). The example in Listing 1.1, replicating a situation where a autonomous car has to brake based on the leading cars distance, indicates the hazards of uncertainty in measurements. This particular example motivates us to implement an application that estimates speed using uncertain GPS measurements from the user's smartphone. We focus on the GPS domain because GPS applications are simple to understand, pervasive in the mobile computing landscape, and commonly experience uncertainty in measurements.

4.3.1 Calculating displacement from coordinates - The Haversine Formula

GPS satellites broadcast signals from space, and each GPS receiver uses these signals to calculate its three-dimensional location (latitude(°), longitude(°), and altitude(m)) relative to the Equator, the prime meridian and the sea level accordingly. Since the measured coordinates are in degrees(°) we have to use the haversine formula to determines the great-circle distance between these two points on a sphere.

We calculate the distance (D) of two coordinates using the Haversine Formula [16] shown in equations 4.3.1, 4.3.2 and 4.3.3. The haversine formula allows to compute the great-circle distance between two points – that is, the shortest distance over the earth's surface directly from the latitude (represented by ϕ) and longitude (represented by λ) of the two points. $R = 6,371,000$ represents the radius of the Earth in meters.

$$a = sin^2\left(\frac{\Delta\phi}{2}\right) + cos\phi_1 \cdot cos\phi_2 \cdot sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{4.3.1}$$

$$c = 2 \cdot \arctan2(\sqrt{a}, \sqrt{(1-a)}) \tag{4.3.2}$$

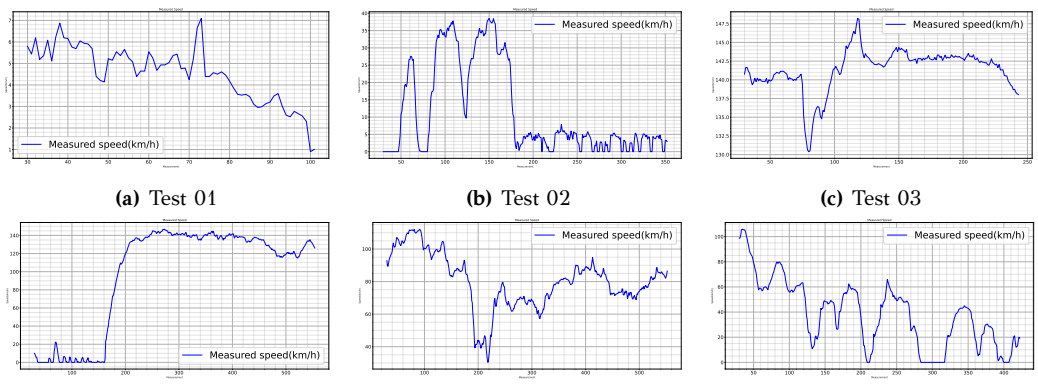
$$D = R \cdot c \tag{4.3.3}$$

4.3.2 Measuring GPS coordinates

Using the PhyPhox [7] mobile application we collected GPS Coordinates as we show in Table 4.4. The main file columns that interest us are: (i) Time, (ii) Latitude, (iii) Longitude, (iv) Altitude and (v) Speed. Figure 4.5 presents the measured coordinates approximately fitted on top of a screenshot of Google Earth [30].

File column	Measurement 1	Measurement 2	...
Time (s)	11.853037	12.842904	...
Latitude (°)	40.727566	40.727483	...
Longitude (°)	22.985232	22.984930	...
Altitude (m)	188.420768	190.421115	...
Altitude WGS84 (m)	234.0	236.0	...
Speed (m/s)	27.410000	27.269999	...
Direction (°)	250.300003	250.199997	...
Distance (km)	2.097027	2.124238	...
Horizontal Accuracy (m)	2.5	2.0	...
Vertical Accuracy (m)	100.0	100.0	...
Satellites	17	18	...

Table 4.4: Measurements collected by the PhyPhox [7] mobile application.



(a) Test 01 **(b)** Test 02 **(c)** Test 03

(d) Test 04 **(e)** Test 05 **(f)** Test 06

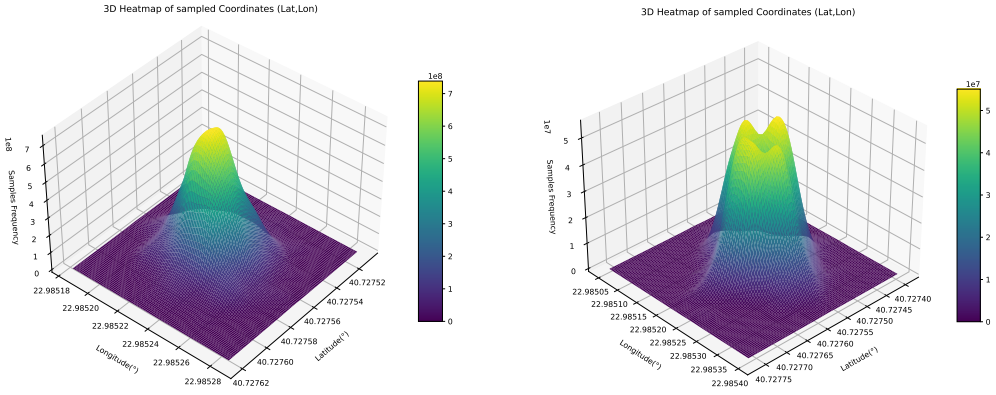
Figure 4.6: This Figure shows the user's speed as calculated by the PhyPhox [7] mobile application.

4.3.3 Converting measurements to `Uncertain` objects

As we mention in Chapter 3, we can initialize any `Uncertain` object using a set of samples. Thus we have to convert the obtained measurements to a distribution of samples and, consequently, perform the speed calculation. Different smartphone Operating Systems (OS) manufacturers provide a different definition of GPS accuracy:

- **Android's** [33] developers documentation suggests: *We define horizontal accuracy as* ***the radius of 68% confidence****. In other words, if you draw a circle centered at this location's latitude and longitude, and with a radius equal to the accuracy, then there is a 68% probability that the true location is inside the circle.*
- **iOS's** [34] developers documentation suggests: *HorizontalAccuracy:* ***The radius of uncertainty for the location, measured in meters****. The location's latitude and longitude identify the center of the circle, and this value indicates the radius of that circle. A negative value indicates that the latitude and longitude are invalid.*

We use an Android smartphone to obtain the measurements, so we consider the horizontal accuracy field provided by PhyPhox [7] as the radius of 68% confidence. In Figure 4.7 we present two different approaches to represent uncertainty in measurements. We sample a multivariate Gaussian distribution around the measured coordinates, with σ equal to the provided horizontal accuracy (Table 4.4) as we show in Figure 4.7a.

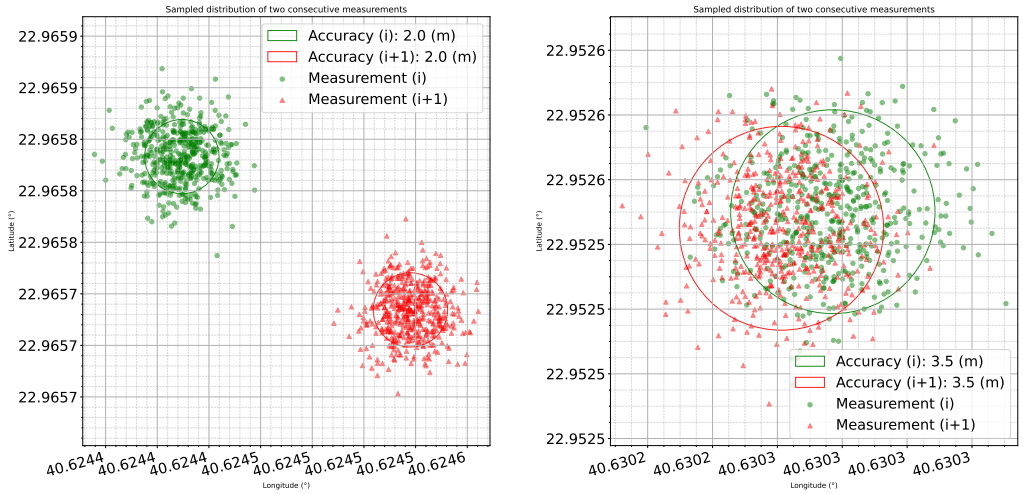


(a) Based on a Gaussian distribution.

(b) Based on a Rayleigh distribution.

Figure 4.7: Sampling around the measured coordinates to add artificial noise in measurements. We placed the reference point in the centre of the plot.

Bornholt [3] proposes that *it is high unlikely that the user's true location is exactly in the centre of the distribution*, thus sampling a Rayleigh distribution around it. We test this claim by adding a second sampling method as we show in Figure 4.7b sampling a mutlivariate Rayleigh distribution.



(a) Consecutive measurements with major displacement. (speed $\neq 0$)

(b) Consecutive measurements with minor displacement. (speed ≈ 0)

Figure 4.8: The sampling radius overlaps between moving and stationary states. Using a circle around the sampling point, we depict the corresponding horizontal accuracy of the measurement.

The function `init_uncertain_tuples()` in Line 4 of Listing 4.2 samples a two-dimensional Gaussian Distribution around particle measured tuples <lat,lon> and converts them to `Uncertain` objects. In Figure 4.8 we can distinctly observe the difference between stationary and non-stationary states when measuring the GPS Coordinates. There is a significant overlap between the two sampled distributions. Although the two sampling centers differ, this could lead to ambiguous results. For example, the user may be travelling forward, while the GPS measurement indicates that he is moving backward. This unveils the need to combine GPS Satellites measurements with other sensors's data to improve accuracy.

4.3.4 Speed estimation with `Uncertain` objects

Following the conversion of coordinates to `Uncertain` objects comes the speed estimation. This process includes using the Haversine Formula (4.3.1) to estimate the displacement and then calculate displacement over

time to obtain the desired value. Listing 4.2 shows the algorithm we use to perform these calculations. We also present the equivalent with particle values in Listing 4.3. The only differantiation between the two Listings is the definition of a complementary function to initialize the `Uncertain` objects and the usage of the `u_map()` method to assist calculations.

```
1  from UPROP import Uncertain
2  from numpy import radians, sin, cos, sqrt, arcsin
3
4  def init_uncertain_tuples(lat, lon, horiz_accuracy):
5      lat_dist, lon_dist = multivariate_normal(lat, lon, horiz_accuracy)
6      lat_unc = Uncertain(lat_dist)
7      lon_unc = Uncertain(lon_dist)
8      return (lat_unc, lon_unc)
9
10 def calculate_displacement(lat1, lon1, lat2, lon2):
11     # convert uncertain objects to radians through map
12     lat1 = lat1.u_map(radians) ; lon1 = lon1.u_map(radians)
13     lat2 = lat2.u_map(radians) ; lon2 = lon2.u_map(radians)
14
15     # calculate through haversine formula
16     dlat = (lat2-lat1)/2 ; dlon = (lon2-lon1)/2
17     a1 = dlat.u_map(sin)**2
18     a2 = lat1.u_map(cos)*lat2.u_map(cos)
19     a3 = dlon.u_map(sin)**2
20     a = (a1 + a2*a3).u_map(sqrt)
21
22     # multiply with 2*EARTH_RADIUS
23     return 2 * a.u_map(arcsin) * 6371000 # in meters
24
25 for i in range(len(coords)):
26     # measurement i
27     lat1, lon1 = init_uncertain_tuples(coords[i][0], coords[i][1])
28     # measurement i + 1
29     lat2, lon2 = init_uncertain_tuples(coords[i+1][0],
30                                        coords[i+1][1])
31     displ = calculate_displacement(lat1, lon1, lat2, lon2)
32     dt = time[i+1]-time[i]
33     # convert speed to kmh
34     speed = (displ/dt).u_map(kmh)
```

Listing 4.2: In this Listing we show the speed estimation from "noisy" or uncertain GPS measurements. In line 4 we initialize two `Uncertain` objects which we use to calculate the Haversine formula we present in Subsection 4.3.1. We later divide the estimated displacement with the elapsed time to calculate the speed. Throughout the code (Lines 12, 13, 17, 18, 19, 20, 34) we use the `u_map` method included in the **UPROP** Python Library. Listing 4.3 shows the equivalent calculations using particle measurements.

```python
from numpy import radians, sin, cos, sqrt, arcsin

def calculate_displacement(lat1, lon1, lat2, lon2):
    # convert degrees to radians
    lat1 = radians(lat1) ; lon1 = radians(lon1)
    lat2 = radians(lat2) ; lon2 = radians(lon2)

    # calculate through haversine formula
    dlat = (lat2-lat1)/2 ; dlon = (lon2-lon1)/2
    a1 = sin(dlat)**2
    a2 = cos(lat1)*cos(lat2)
    a3 = sin(dlon)**2
    a = sqrt((a1 + a2*a3))

    # multiply with 2*EARTH_RADIUS
    return 2 * arcsin(a) * 6371000 # in meters

for i in range(len(coordinates)):
    # measurement i
    lat1, lon1 = coordinates[i][1], coordinates[i][1]
    # measurement i + 1
    lat2, lon2 = coordinates[i+1][1], coordinates[i+1][1]
    displ = calculate_displacement(lat1, lon1, lat2, lon2)
    dt = time[i+1]-time[i]
    speed = kmh(displ/dt)
```

Listing 4.3: In this Listing we show the speed estimation from the particle measurements we collect. We use the Haversine formula (subsection 4.3.1) to calculate the displacement. We later divide it with the elapsed time between measurements to calculate speed. Listing 4.2 shows the equivalent calculations using `Uncertain` objects.

4.3.5 Speed Estimation Results

In the previous subsection (4.3.4) we present the methodology for estimating the user speed based on uncertain measurements. In this Subsection we present the results of those calculations, as well as comparison with the **conventional method of calculations**, **which is the particle sample calculation**. We consider the *Monte Carlo Simulation* (Ref: Subsection 4.1.1) to be the golden standard for uncertainty propagation, thus we compare our implementation and the particle sample calculation with the Monte Carlo Simulation distributional results.

We present the performance results of our model in Table 4.5. On average we have a $10\times$ **performance gain** over the particle sample calculation. This performance gain **ranges from** $2\times$ **to** $34\times$ **in specific tests**. We extract the performance evaluation of our model based on the

Normalized Wasserstein metric (Subsection 2.3.3).

Performance Gain			
Test id	**Min**	**Mean**	**Max**
Test 01	2.07	**2.71**	3.91
Test 02	1.77	**2.34**	5.04
Test 03	17.01	**34.73**	52.79
Test 04	2.0	**7.79**	44.41
Test 05	4.24	**15.08**	79.06
Test 06	2.36	**5.57**	29.06

Table 4.5: This table shows the *UPROP* Library accuracy gain over the particle sample calculations. Comparing these results after observing Figure 4.11 it becomes obvious that there is a strong connection of accuracy in calculations with accuracy in GPS measurements and thus the sampling radius. This is a generally expected behavior.

We plot our speed estimation results in Figure 4.9 using two methods:

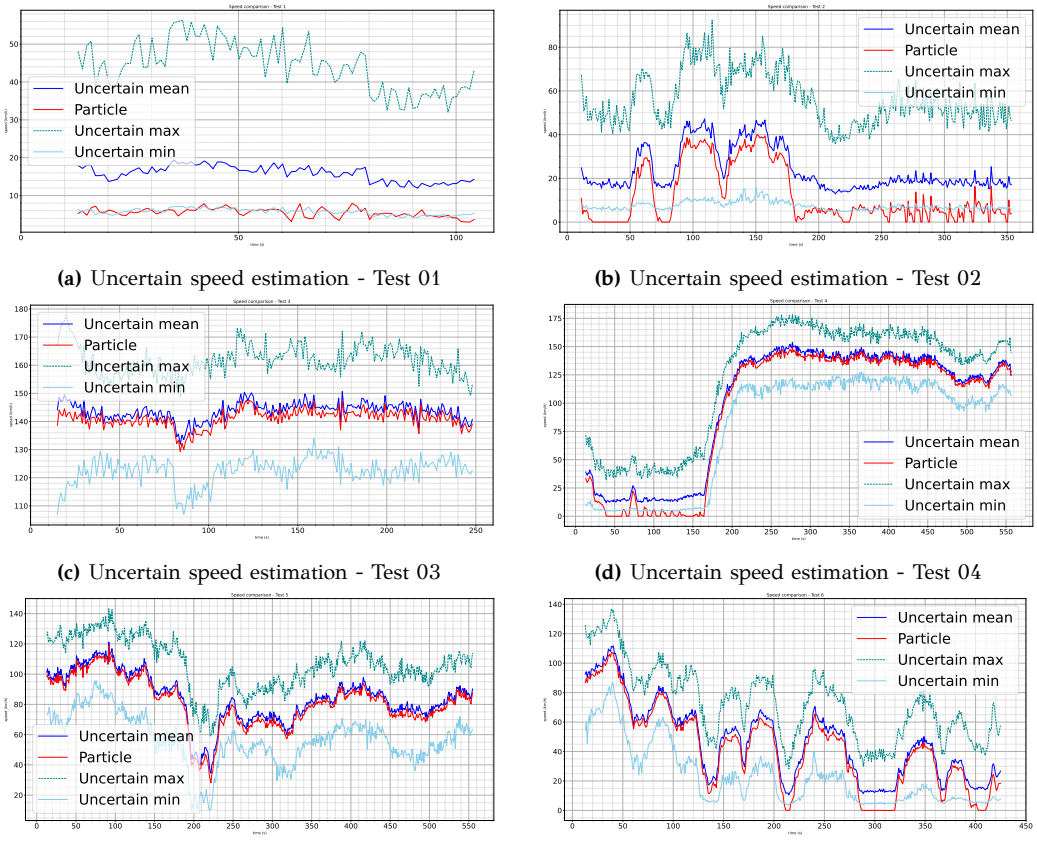
❶ `Uncertain` objects and

❷ Particle value calculations (calculations with only the measured coordinates), as we show in Listing 4.3.

We depict the `Uncertain` objects mean values with bold blue color, as well as it's minimum and maximum values with faint blue. The measured value (with bold green color) is the speed as estimated by the PhyPhox application. We show the particle calculated values (❷) with red color.

On average the `Uncertain` model's *mean value* is on-par with the particle calculated value. Besides this convergence, **On most occasions the *UPROP* Library manages to propagate the input uncertainty to the output**. The effect of this uncertainty is visible in the plotted results. The minimum and maximum value, in particular calculations, diverge more than 50% of the mean value from the mean value. The *UPROP* Library exposes this effect, thus assisting the developer to take more informed decisions.

In some calculations, the `Uncertain` model fails to calculate the true speed mean value: We conclude that: **The closer to 0 the calculated value, the higher the inaccuracy of our model**. This is markedly observable in tests: 01-(Figure 4.9a), 02-(Figure 4.9b), 04-(Figure 4.9d) and 06-(Figure 4.9f).



(a) Uncertain speed estimation - Test 01

(b) Uncertain speed estimation - Test 02

(c) Uncertain speed estimation - Test 03

(d) Uncertain speed estimation - Test 04

(e) Uncertain speed estimation - Test 05

(f) Uncertain speed estimation - Test 06

Figure 4.9: Speed estimation - Comparison between using the UPROP Python Library and the particle values. We depict the mean, min and max values of each method. The solid faint red and blue lines depict the minimum values and the faint dashed lines (in the top of each plot) the maximum values of each method accordingly.

The effect of the selected memory representation and of correlation.

We attribute this behavior to the distributional nature of the `Uncertain` objects. We found that this behavior only occurs in certain conditions and transformations:

❶ the `Uncertain` 's distribution support (R_X) has to be on either side of 0 (containing both negative and positive values),

❷ the transformation function should be even $f(x) = f(-x)$ e.g. $f(x) = x^2$ or $f(x) = |x|$, and

❸ we should keep track of autocorrelation.

Suppose we have an example where we have the particle value $x = 0$ and the corresponding `Uncertain` object is a normal distribution with undefined variance and $\mu = 0$. If we pass the object through an even function the generated `Uncertain` object becomes skewed (positively or negative depending on the function's sign) while the relevant particle value is approximately 0. We showcase this behavior in Figure 4.10 and we present the relevant pseudocode in Listing 4.4. In particular we set the particle value $p_1 = 0$ and the equivalent `Uncertain` value: $u_1 = \text{Uncertain}(X \sim \mathcal{N}(0, 1))$. We depict p_2 with the solid red line and the $\text{mean}(u_1)$ with the solid green line. For the transformation we choose the even function $f(x) = x^2$. Thus $p_2 = p_1^2 = 0$ and $u_2 = u_1^2$. More specifically, if:

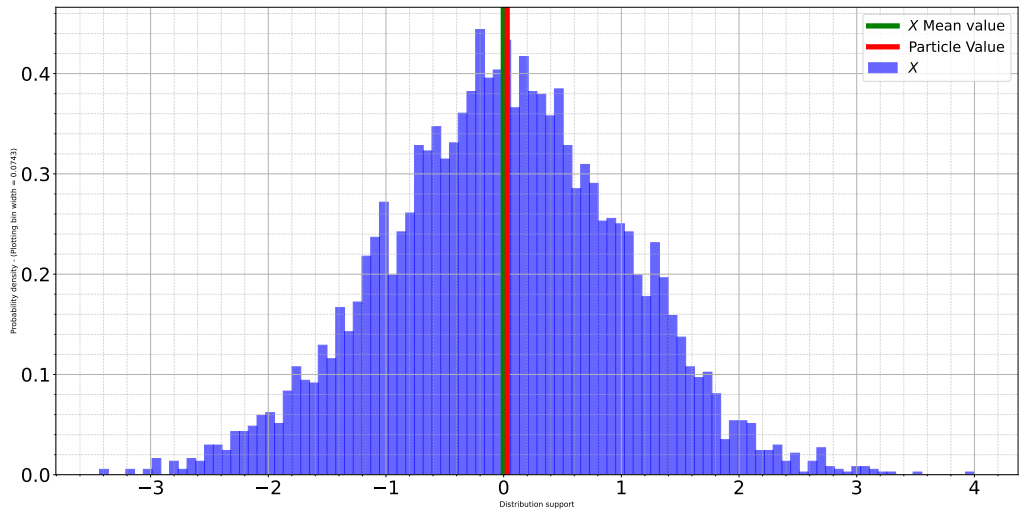
❶ autocorrelation **ON**: u_2 represents the **Laplace** distribution with $\mu = 0$ and $b = 1$ (Gaussian Mixture [35]).

❷ autocorrelation **OFF**: u_2 represents the **chi-squared** (χ^2) distribution with degree of freedom $k = 1$ [36].

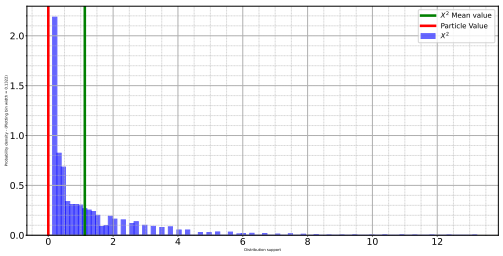
```
X = Uncertain(gaussian(0,1))

autocorrelation_ON()
Y = X*X  # chi-squared(k=1)

autocorrelation_OFF()
Y = X*X  # laplace(0,1)
```

Listing 4.4: This Listing shows the pseudocode needed to generate the distributions in Figure 4.10. By setting autocorrelation ON we generate the chi-squared (χ^2) distribution and by setting it OFF we generate the Laplace distribution.

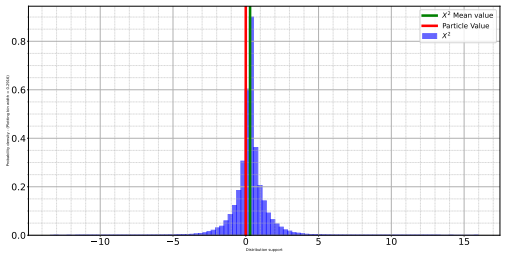
We see this behavior in the GPS application since *even* functions appear in the Haversine Formula. Specifically in equation 4.3.1 we have the function: $f(x) = \sin^2$. We test both cases of autocorrelation and we observe that if we choose the latter case: *autocorrelation OFF*, we have to normalize our data since we later map the squared value to the `sqrt()` function. As we know the `sqrt` takes only positive input values. This limits the accuracy of our model in values near 0 as we can observe in tests 01, 02, 04 and 06.



(a) Input `Uncertain` value $u_1 = \text{Uncertain}(X \sim \mathcal{N}(0,1))$ and input particle value $p_1 = 0$.



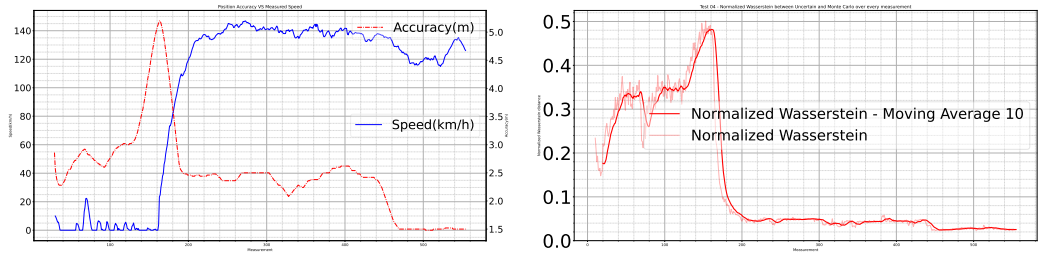
(b) Autocorrelation ON: `Uncertain` value $u_2 = u_1^2$ and particle value $p_2 = p_1^2 = 0$. There is a significant deviation between $\text{mean}(u_2)$ and p_2. The distribution of u_2 becomes positively skewed.



(c) Autocorrelation OFF: `Uncertain` value $u_2 = u_1^2$ and particle value $p_2 = p_1^2 = 0$. The mean value of the `Uncertain` object is equal to the particle value after the transformation ($\text{mean}(u_2) = p_2$).

Figure 4.10: Distributional analysis of `Uncertain` variables with autocorrelation ON and OFF. With autocorrelation ON: in Figure 4.10b, $u_2 \sim (\chi^2)$ (chi-squared distribution with $k = 1$). With autocorrelation OFF: in Figure 4.10c $u_2 \sim \text{Laplace}(\mu = 0, b = 1)$. Ideally it should be: $\text{mean}(u_2) \approx p_2$ for every situation.

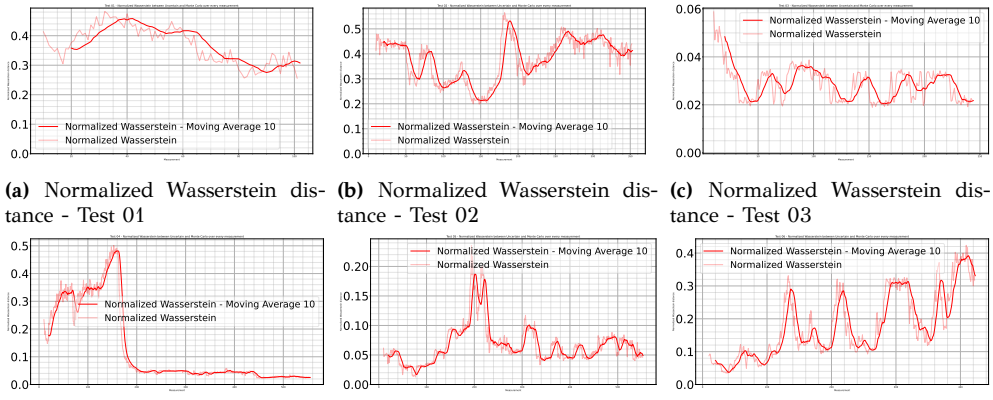
We also observe a highly dependent relationship of the distribution support range R_X with the accuracy of measurements. We can notice that, by comparing Figure 4.11a and 4.11b. **The higher the meaurement accuracy the lower the Normalized Wasserstein distance** Which means that the smaller the radius around the sampling point, the higher accuracy we obtain. We certainly expect this behavior since the deviation of the sampled distribution decreases and thus we propagate less uncertainty through the model.



(a) Measured speed vs measurement accuracy - Test 04.

(b) Normalized Wasserstein distance - Test 04.

Figure 4.11: This Figure shows that tncertainty in measurements is highly dependent to uncertainty in calculations. When we obtain high



(a) Normalized Wasserstein distance - Test 01

(b) Normalized Wasserstein distance - Test 02

(c) Normalized Wasserstein distance - Test 03

(d) Normalized Wasserstein distance - Test 04

(e) Normalized Wasserstein distance - Test 05

(f) Normalized Wasserstein distance - Test 06

Figure 4.12: Normalized Wasserstein distance for each test. We generate each result between using the *UPROP* Library and the Monte Carlo Simulation. Table 4.5 presents the min, mean and max values of this Figure.

Figure 4.13 presents the speed values calculated by using the *UPROP* Library and the Monte Carlo Simulation. Based on it we can safely state the *UPROP* Library usage yields narrower distribution support values than the Monte Carlo implementation. This explains the results in Table 4.5 and why the `Uncertain` model has at least $2\times$ accuracy for each test. We should also note that both the `Uncertain` model and the Monte Carlo Simulation fail to capture the values near 0 due to the reasons we explain earlier.

We also notice a significant **deviation between the particle and measured values**. In Figure 4.14 we plot the ratio $\frac{\text{particle}}{\text{measured}}$. Ideally this ratio should be equal to 1. In *Test 02* (orange line) for particular particle values are $10\times$ the measured value. This raises questions for both the quality

of measurements and the back-end calculations of the PhyPhox [7] application.

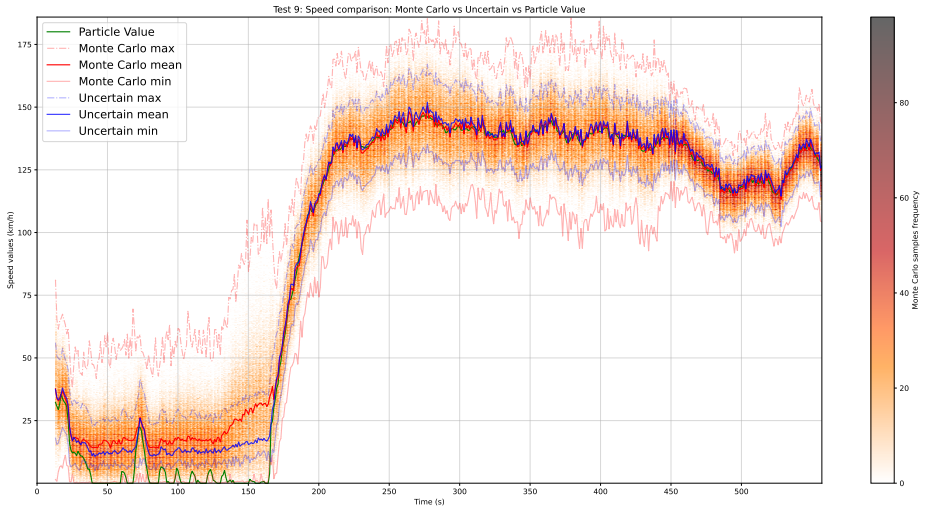


Figure 4.13: Speed estimation - Test 04: Comparison between using the *UPROP* Library vs the Monte Carlo Simulation and the particle values. We depict the mean, min and max values of each method. For the Monte Carlo Simulation we also plot the heatmap of the calculated values.

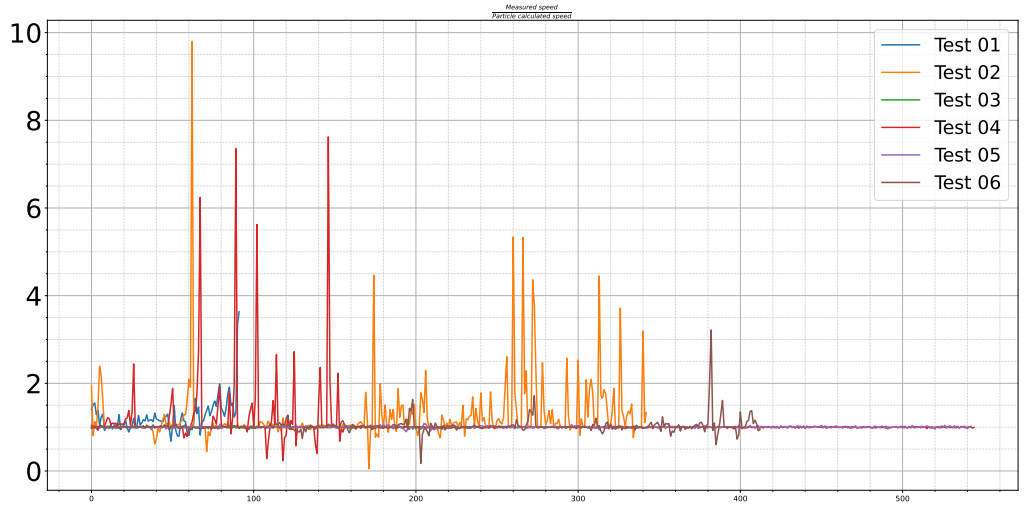
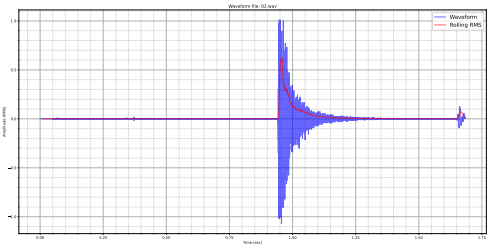


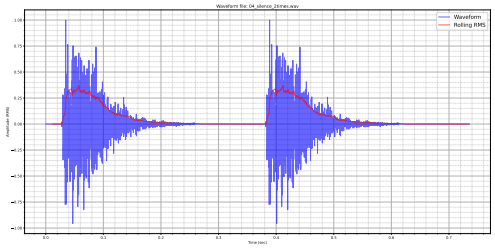
Figure 4.14: This Figure shows the Particle vs Measured values ratio: $\frac{\text{particle}}{\text{measured}}$. Ideally the ratio should be equal to 1. Specific particle values in *Test 02* are almost $10\times$ the measured value.

4.4 Detecting claps from noisy sound signals

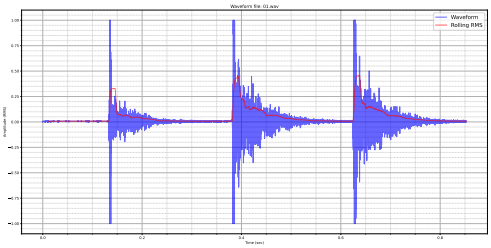
Following the GPS application we now face a challenge with the UPROP Python Library. A major step in programming with it, is asking questions with conditional operators. Speech recognition is one of the most emerging fields of research, as speech is the natural way of communication [37]. Researchers develop neural networks for speech recognition even in smartphones [38, 39]. There is also an emerging research interest in sound quality [40, 41] in the telecommunications sector. The rising research interest lead us to develop an application with sound recognition using the UPROP Python Library:



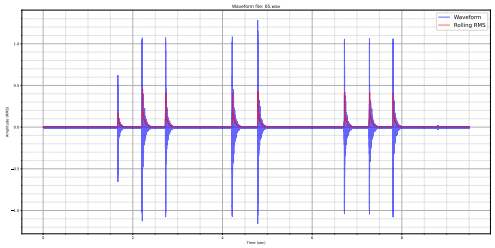
(a) Input sound signal consisting of 1 distinct clap.



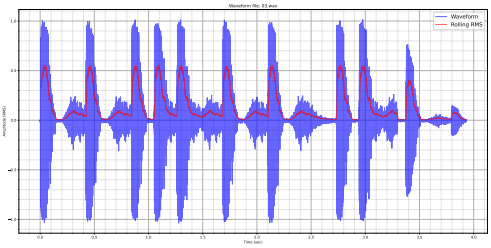
(b) Input sound signal consisting of 2 distinct claps.



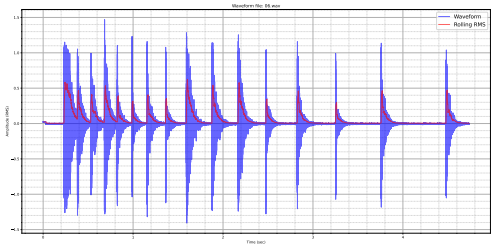
(c) Input sound signal consisting of 3 distinct claps.



(d) Input sound signal consisting of a 8 distinct (high SNR) claps



(e) Input sound signal consisting of 10 hard to distinguish (low SNR) claps.



(f) Input sound signal consisting of 16 distinct claps.

Figure 4.15: This figure shows the recorded sound signals that we use to test the clap detection algorithm we present in Subsection 4.4.3 using the **UPROP** Python Library through the algorithm. The red line depicts the Root Mean Square Amplitude (RMS) of the signal, which represents its' energy at the given time.

Given the popularity of speech recognition and the inseparable relation of sound and noise signals we decided to implement a naive **clap detection algorithm** to showcase the usage of the *UPROP* Library. We utilized as input recorded sound signals and we later enhanced the signal with Additive White Gaussian Noise ($AWGN$). We select this application because it can showcase the usage of (i) **calculations** and (ii) **conditional operators** using `Uncertain` objects, to arrive at a desired result.

Clap detectors have been around for some time, with their functionality ranging from switching lights on and off, to perform any user defined action. There is a lot of research conducted on the area of sound recognition and there are different suggestions on how to detect claps [42]. Some algorithms include estimating the absolute energy of the clap sound, its source azimuth (estimated from stereo microphones), and its range as conveyed by the direct-to-reverberant energy balance [43].

4.4.1 Problem Description

The problem is to implement a simple robust method to detect single claps in sound using the *UPROP* Library. The algorithm should be able to detect the exact number of claps present in the input sound signal and output this clap count. We recorded a series of sound signals, each with a different amount of claps and interval between them. Figure 4.15 shows 4 of the recorded sounds.

Two characteristics of a clapping sound:

1. there is a sudden increase in amplitude when the clap occurs [44, 42].
2. it is short in duration

4.4.2 Adding noise to a signal

Similarly with GPS speed calculation in Section 4.3 we add have to define the noise of the inputs signals in order to use the *UPROP* Library. We recorded the input signals using an XLR microphone, thus resulting in lower Signal to Noise Ratio (SNR) [45]. The audio encoding of the input signal is 16-bits. We propose two methods of adding noise to a signal:

1. Adding constant Additive White Gaussian Noise [46]
2. Adding Noise based on Signal to Noise Ratio [47]

Adding constant AWGN

We selected adding constant Additive White Gaussian Noise ($AWGN$) because it is the most widely used noise model in research to mimic the effect of multiple random processes that occur in nature. It is the equivalent of having background noise in the recording environment. Listing 4.5 presents the code used to generate and add $AWGN$ in the input signal. Figure 4.16 shows the $AWGN$ (4.16a) and the reference signal combined with $AWGN$ (4.16b).

```
from UPROP import Uncertain
import soundfile

# read signal
signal = soundfile.read(file)
uncertain_signal = []
for measurement in signal:
    # define a gaussian distribution around the measurement
    noisy_measurement = numpy.random.normal(measurement,
                                            stdev,
                                            len(signal))
    uncertain_measurement = Uncertain(noisy_measurement)

    # to ensure measurements lie in [-1,1]
    uncertain_measurement = uncertain_measurement.bound(lower=-1,
                                                        upper=1)
    uncertain_signal.append(uncertain_measurement)
```

Listing 4.5: We convert measurements to `Uncertain` object by creating a distribution $\mathcal{N} \sim (\mu, \sigma)$ aroung constant noise. σ is user-defined and μ is the measured amplitude. We bound the `Uncertain` variable to ensure the measurements follow the original 16-bit audio encoding.

Adding noise based on SNR

Signal-to-Noise Ratio (SNR) is a measure that compares the level of a signal to the level of background noise. We define SNR as the ratio of signal power to the noise power:

$$SNR = \frac{P_{signal}}{P_{noise}} \tag{4.4.1}$$

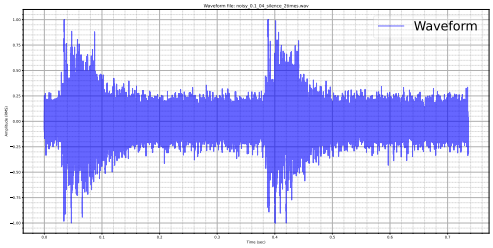
where P is the average power. A ratio higher than 1 indicates that the signal is more prominent than the noise. The Signal-to-Noise Ratio

is often expressed in decibels (dB).

$$SNR_{dB} = 10 \log_{10} \frac{P_{signal}}{P_{noise}} \tag{4.4.2}$$

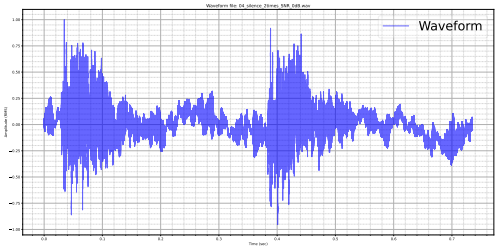
The wide usage of SNR in the telecommunications sector as a characterization for the quality of a wireless connection inspired us to adding noise to the reference signals based on a Signal-to-Noise Ratio. Rahul et al [47] suggest that there is a relative SNR wall after which even the most robust detectors fail to detect the received clean signal.

(a) The $AWGN$ signal has an almost constant RMS Amplitude of 0.2.

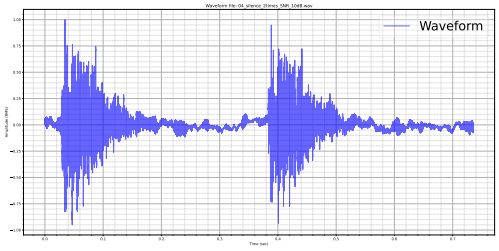


(b) The generated signal after adding $AWGN$ of distribution: $\mathcal{N} \sim (0, 0.1)$. The addition of noise adds energy to the waveform.

Figure 4.16: This figure shows the modified sound signal which is the result of: the reference plus Additive White Gaussian Noise ($AWGN$).



(a) We modify the input sound signal with by adding noise with $SNR = 0\ dB$.



(b) We modify the input sound signal with by adding noise with $SNR = 10\ dB$.

Figure 4.17: This figure shows the modified sound signal which is the result of: the reference plus Additive White Gaussian Noise (AWGN) based on a specific Signal to Noise Ratio (SNR). Higher SNR means the reference signal is more prominent.

We selected the SNR values based on relative research from Parrish et al [48] who propose that a signal with an SNR value of 20 dB or more is the recommended for data networks, where as an SNR value of 25 dB or more is the recommended for networks that use voice applications. We opt for multiple values of SNR ranging between 0 dB (1:1 signal-to-noise

power ratio) to 25 dB (316:1 signal-to-noise power ratio). Listing 4.6 shows the pseudocode method for adding noise based on an SNR value. Figure 4.17 shows the reference waveform with added noise based on a specific user-defined SNR.

```
from UPROP import Uncertain
import soundfile

# read signal
signal = soundfile.read(ref_file)
noise = soundfile.read(noise_file)

uncertain_signal = []
ref_singal_rms = rms(signal)
noise_signal_rms = rms(noise)

for measurement in signal:
    # find the noise amplitude based on SNR and add it
    # to the measurement
    snr = convert_to_ratio(snr_db)
    noise_amplitude = measurement / snr
    mixed_amplitude = measurement + noise_amplitude

    # to convert to uncertain generate a normal distribution
    # around the measurement with standard deviation equal
    # to noise_amplitude
    noisy_signal = normal_distibution(mu = measurement,
                                      stdev =abs(noise_amplitude),
                                      size = 500)
    uncertain_measurement = Uncertain(noisy_signal)

    # to ensure measurements lie in [-1,1] to fit encoding
    uncertain_measurement = uncertain_measurement.bound(lower=-1,
                                                        upper=1)
    uncertain_signal.append(uncertain_measurement)
```

Listing 4.6: Pseudocode for adding noise based on a specific SNR. We convert measurements to `Uncertain` objects by creating a distribution $\mathcal{N} \sim (\mu, \sigma)$ aroung a measurement. We define the variance of noise (σ) based on the SNR and μ is the measured amplitude. We bound the `Uncertain` variable to ensure the measurements follow the original 16-bit audio encoding.

4.4.3 Algorithm explanation

The main goal of the algorithm is to analyze a sound signal, capture these two characteristics and infer if the sound is a clapping sound. To

recognise the loudness characteristic of a clapping sound we compare the amplitude of a short term average (consisting of the previous m samples), effectively low pass filtering the signal with a threshold value. We define the threshold using a constant over which we consider a clapping sound might suprass, plus the average of a long term window. The long term window consists of $n \gg m$ samples, thus taking the noise floor into account. We adapt this threshold value continuously.

```
from UPROP import Uncertain

for uncertain_sample in samples:
    short_term_average : Uncertain = mean_of_last_samples(K)
    long_term_average : Uncertain = mean_of_last_samples(M)

    # to avoid added noise amplitude floor increase the threshold
    uncertain_threshold = const_threshold + long_term_average
    max_val = 0
    clap_duration = 0

    # as long we have a high enough amplitude
    while (short_term_average > uncertain_threshold):
        max_val = max(short_term_average.max(), max_val)
        clap_duration += 1
        # if we exceed the max duration break
        if clap_duration > max_allowed_duration:
            clap_duration = 0
            max_val = 0
            break
        # update values
        next_samples()
        short_term_average : Uncertain = mean_of_last_samples(K)
        long_term_average : Uncertain = mean_of_last_samples(M)
        uncertain_threshold = const_threshold + long_term_average

    # if the max value is above the const
    # universal threshold count 1 clap
    if 0 < clap_duration < max_allowed_duration:
        if max_val > clap_likeness_threshold:
            clap_count += 1

    clap_duration = 0
    max_val = 0
    next_samples()
```

Listing 4.7: Pseudocode of the clap detection algorithm.

To capture the second characteristic of a clapping sound, which is

the shortness of duration we count the duration of registered events. If this average of the short term window is above the defined threshold we start counting the duration of this event. As long as `short_term_average > long_term_average` we increase the duration. Finally if the duration does not exceed the defined duration threshold we register a clap event. Listing 4.7 presents the pseudocode for the algorithm.

4.4.4 Emphasizing on the usage of `Uncertain` objects

In Listing 4.7 we initialize two `Uncertain` objects in lines 4 and 5 and consequently, we use the overloaded arithmetic operators, in line 8 to define a third. We use these `Uncertain` objects throughout the code to decide if the short term average surpasses the long term average. In Subsection 3.4.1 we propose two methods of comparing Random Variables or rather, `Uncertain` objects, which we further analyze and test in Section 4.2.2. We use this functionality in line 13 where we compare two `Uncertain` objects and whether to register or not a clapping sound.

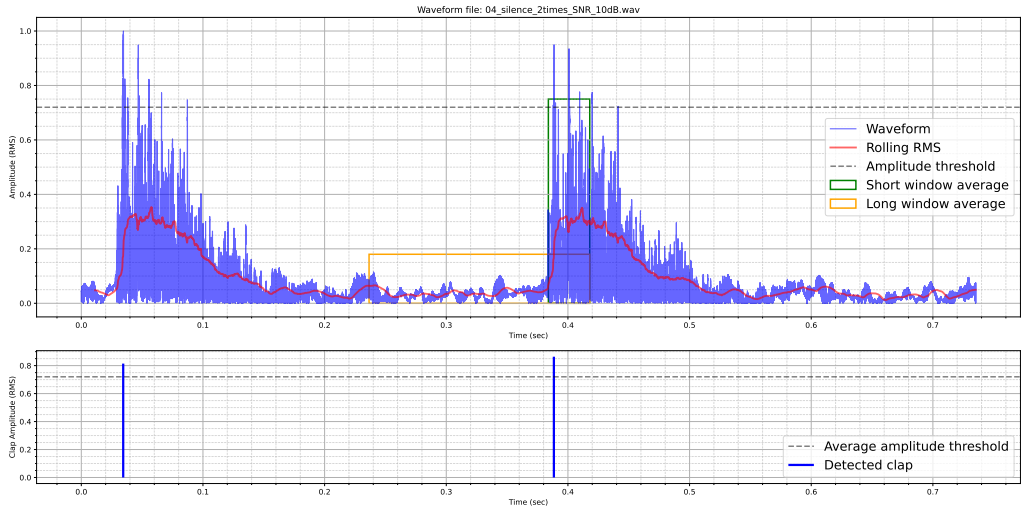


Figure 4.18: This Figure visualizes the algorithm we present in Subsection 4.4.3. We take into account the absolute value of the reference signal calculating the short term window average and later compare it with the long term window average. If the short term window average surpasses the amplitude threshold then and the duration of the event is less than the duration threshold we register a clap event. In this example, we register two clap events.

The primary goal of the *UPROP* Library is to seamlessly merge with existing code so that it does not act as an extra burden for programmers.

Specifically in Listing 4.7 the only section we observe a differentiation with using conventional numeric types is in line 14. **Thus we consider that the `Uncertain` object has achieved a goal we set: Minimality.**

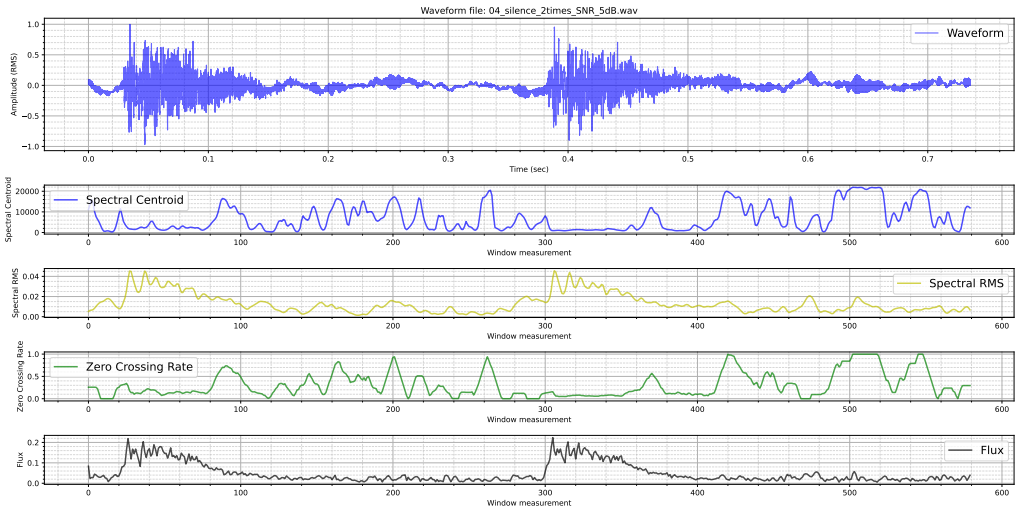


Figure 4.19: This Figure shows 4 selected sound descriptors of the input sound signal. The Spectral Centroid is the barycenter of the signals' spectrum. The Spectral RMS or *Total Energy* estimates the signal power of a frame around a given time. The zero-crossing rate is a measure of the number of times a signal values crosses the zero axis. **Spectral flux** is a measure of the variability of the spectrum over time [49].

4.4.5 Validating Recognitions

In order to validate the method we present in Subsection 4.4.1 using the **UPROP** Python Library, we perform the same clap sound recognition algorithm using the *Spectral Flux* instantaneous spectral sound descriptor.

Sound descriptors are specific extracted audio features from sound signals that assist in sound classification and characterization. Geoffroy Peeters [50] reports a large set of audio features for sound description. We select a small subset of low-level descriptors and visualize them along the input signal in Figure 4.19. This gives us the initial insight of the descriptor to use for distringushing the clap sounds. The insight from Figure 4.19 is that the Spectral Flux seems to be the appropriate descriptor to use to validate our results.

Spectral flux ($flux(t)$), is a measure of the variability of the spectrum over time [49], calculated by comparing the power spectrum for one frame against the power spectrum from the previous frame:

Definition 4.4.1

$$flux(t) = \left(\sum_{k=b_1}^{b_2} |s_k(t) - s_k(t-1)|^p \right)^{\frac{1}{p}} \tag{4.4.3}$$

Where s_k is the spectral value at bin k. The magnitude spectrum and power spectrum are both commonly used. b_1 and b_2 are the window frame edges, between which to calculate the spectral flux and p is the norm type.

Calculated this way, the spectral flux is not dependent upon overall power, since we normalize the spectra, nor on phase considerations, since we only compare the magnitudes. We decided to use spectral flux because of it's popularity in onset detection [51] and audio segmentation [52]. Figure 4.20 depicts a waveform with multiple clapping sounds and we observe that the spectral flux can distringuish the moments when a clap occurs.

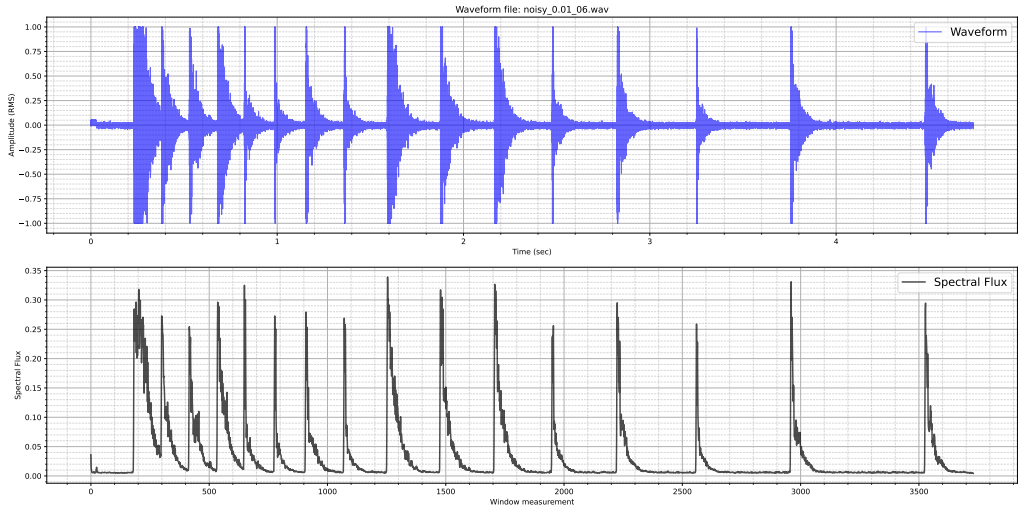


Figure 4.20: This figure depicts a waveform with multiple clapping sounds and we observe that the spectral flux can distringuish the moments when a clap occurs.

The validation algorithm we use is a variation of the algorithm presented in Subsection 4.4.1. Firstly, we use the original particle measurements. Furthermore , instead of averaging the amplitude of the measurements with the added noise, we average the values of *Spectral Flux* descriptors. We redefine the *spectral flux threshold* accordingly to fit our data. Figure 4.20 shows the short and long term window and the resulting clap detections. We observe that the algorithm using the *UPROP*

Library recognizes the same amount of claps, at the same time compared with the algorithm using *spectral flux*.

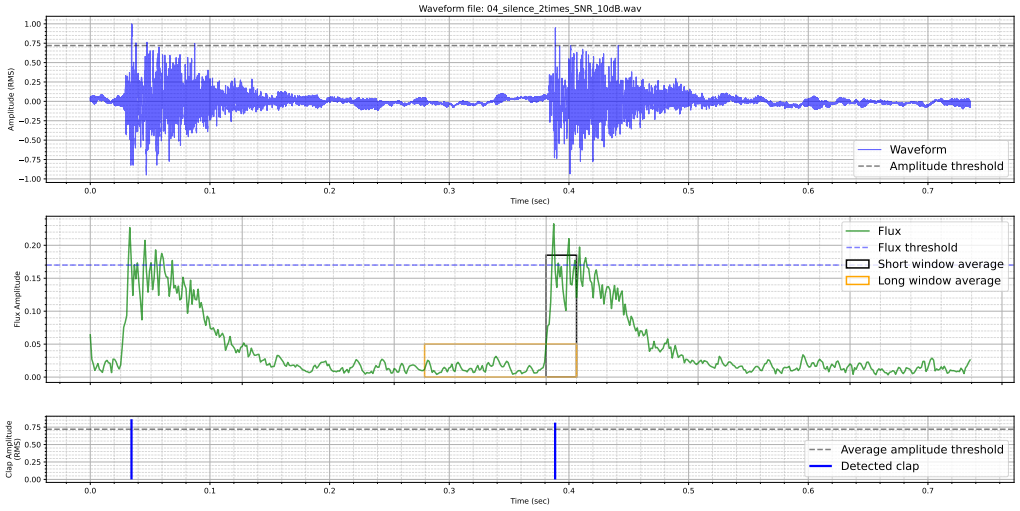


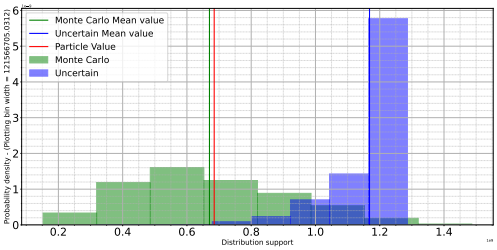
Figure 4.21: Detected clapping sounds using spectral flux descriptor. The input waveform is on the top subfigure, In the middle subfigure we show the instantaneous spectral flux descriptor shows. We low pass filter the descriptor by computing a short term average on the *short term window*. This short term average has to exceed a threshold in order to trigger further evaluation of the microphone input. The evaluation includes measuring the duration of how long the low pass filtered descriptor exceeds the constant threshold. We observe that the recognitions are accurate, resulting in two claps.

4.5 Brown Ham dislocation model

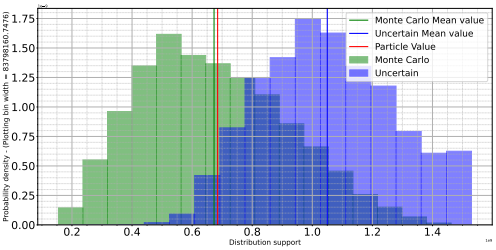
This benchmark calculates the cutting stress of an alloy precipitate using the Brown-Ham dislocation model [8, 9]. Anderson et al. [8] provide empirical value ranges for the inputs of the dislocation model. We assume that the inputs of the dislocation model follow a uniform distribution across these ranges. Equation 4.5.1 defines the calculation of the cutting stress of an alloy precipitate.

$$\sigma_C = \frac{M\gamma}{2b} \cdot \left(\sqrt{\frac{8R_S\gamma\phi}{\pi G b^2}}\right) \tag{4.5.1}$$

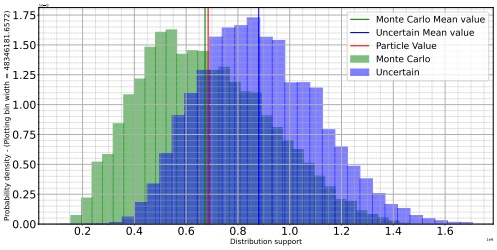
We calculate the Normalized Wasserstein distance between the implementation using the *UPROP* Library, and the conventional method: using particle sampled values. We use the Monte Carlo Simulation as reference. We plot the results in Figure 4.23.



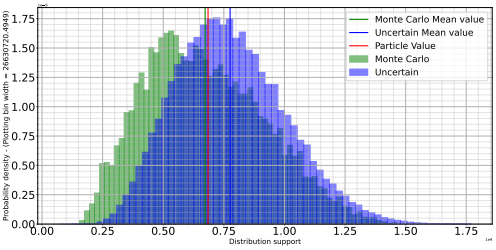
(a) Brown-Ham coefficient calculation with 8 bins.



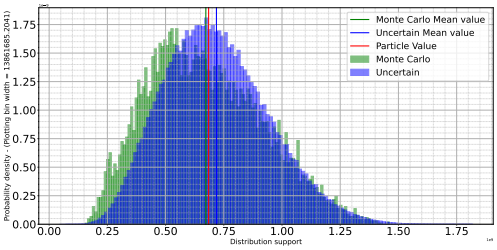
(b) Brown-Ham coefficient calculation with 16 bins.



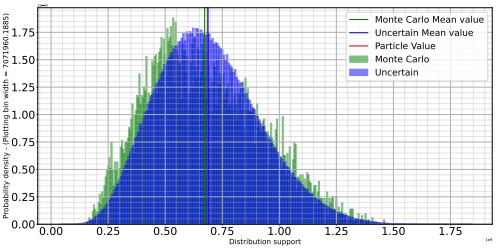
(c) Brown-Ham coefficient calculation with 32 bins.



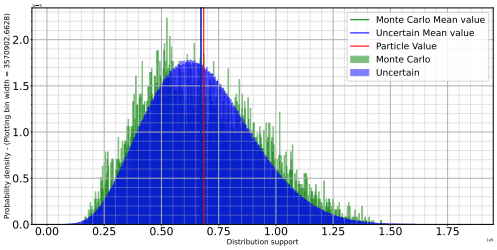
(d) Brown-Ham coefficient calculation with 64 bins.



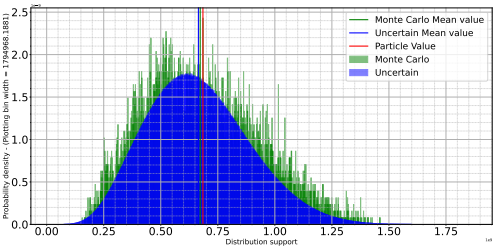
(e) Brown-Ham coefficient calculation with 128 bins.



(f) Brown-Ham coefficient calculation with 256 bins.



(g) Brown-Ham coefficient calculation with 512 bins.



(h) Brown-Ham coefficient calculation with 1024 bins.

Figure 4.22: This figure shows the distributions generated for the Brown-Ham [9] coefficient calculation with varying number of bins, using the Monte Carlo Simulation, `Uncertain` objects through the **UPROP** Python Library, and just particle values. As the number of bins increases the distributions' mean appears to converge to the particle value.

We observe that this distance is inversely proportional to the number of bins (Lower is better). The *UPROP* Library performs from $2\times$ more accurate, with 64 bins, up to $9.4\times$ with 1024 bins, compared to the particle sample calculation. If we perform the benchmark with the `Uncertain`

objects having from 8 to 32 bins, our model performs up to $2\times$ worse, with regards to accuracy. We also visualize the resuts for each number of bins in Figure 4.22. The two distributions indeed converge as the number of bins increases.

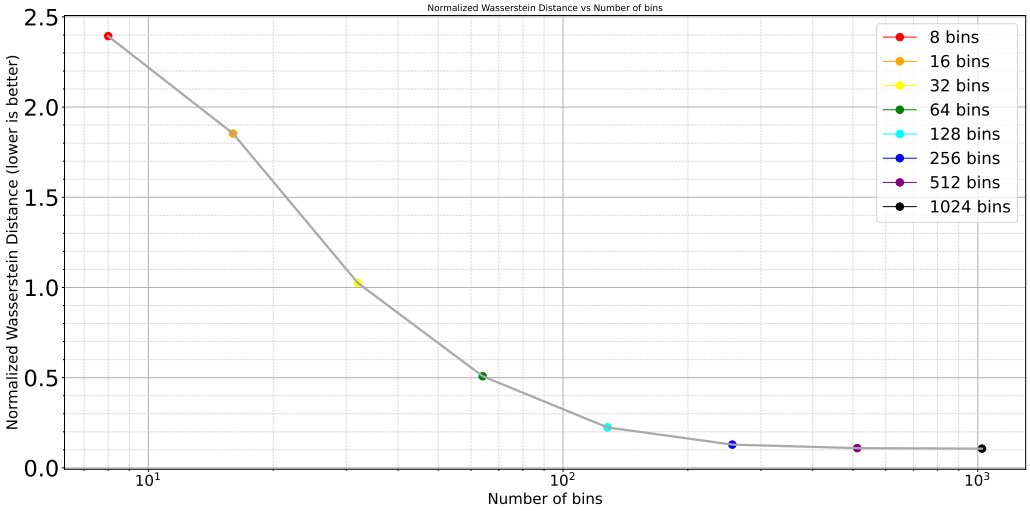


Figure 4.23: The Normalized Wasserstein distance between the `Uncertain` and Monte Carlo distributions is inversely proportional to the number of bins. (Lower is better).

4.6 NIST Uncertainty Machine - Comparison

The NIST Uncertainty Machine [53] is a web-based software application to evaluate the measurement uncertainty associated with a scalar output quantity that is a known and explicit function of a set of input quantities for which estimates and evaluations of measurement uncertainty is available. Random variables model the input and output quantities and they use their probability distributions to characterize measurement uncertainty. It supports both known and empirical probability distributions, but has limited capabilities due to its web-based nature and lack of features.

We select a benchmark to compare the *UPROP* Library with the NIST Uncertainty Machine. We designed this benchmark specifically to test the ability of each framework to track correlation between variables. We show the benchmark implementation using the UPROP Python Library in Listing ??. We recursively add a variable to a sum. We observe major differences in the PDF function of the output variable X.

The `Uncertain` object fails to capture the correlation between the variable X_{sum} and X_{ref}. This is because we only support autocorrelation. We check this by comparing the memory address of each object at the time we invoke each operation / function.

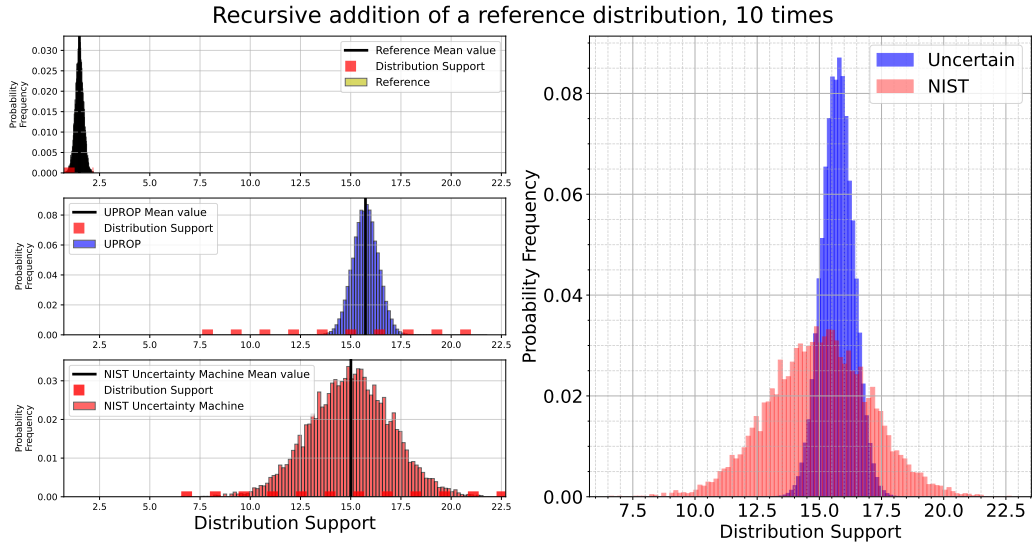


Figure 4.24: Comparison between the *UPROP* Library and the NIST Uncertainty Machine. We initialize a Random Variable `X` $\sim \mathcal{N}(0, 1)$ and we add this variable to a sum: `sum += X` for 10 times.

```
X_ref = Uncertain(samples)
X_sum = 0

for i in range(0,10):
    X_sum += X_ref
```

Listing 4.8: Adding a reference distribution to a sum 10 times. We initialize a Random Variable `X` $\sim \mathcal{N}(0,1)$ and we add this variable to the sum: `sum += X` for 10 times.

4.7 Pyro - Comparison

Pyro [54] is a programming library written in Python and supported by PyTorch on the backend, which is a widely used platform for deep learning. With this said, Pyro is mainly focused on Machine Learning (ML) tasks and enables the user to utilize flexible and expressive deep probabilistic modelling for any computable distribution, even empirical while unifying modern deep learning and Bayesian modelling.

We compare the *UPROP* Library with Pyro's **experimental feature** of Random Variables using a simple benchmark of two linear transformations on two Random Variables. We present the version compatible with the *UPROP* Library in Listing 4.9 and the version compatible with Pyro in Listing 4.10. Theoretically they should both yield as a result, a Random Variable Z with $\mu_Z = 35$ and $\sigma_Z = \sqrt{18}$. Indeed, as we see in Figure 4.25 both the *UPROP* Library and Pyro [54] return the expected result.

```
from UPROP import Uncertain
import random

X = Uncertain(random.normal(5, 1))
Y = Uncertain(random.normal(20, 3))
Z = 3*X+Y
```

Listing 4.9: Adding a reference distribution to a sum 10 times using the *UPROP* Library. We initialize a Random Variable $X \sim \mathcal{N}(0,1)$ and we add this variable to the sum: `sum += X` for 10 times.

```
from pyro.distributions.torch import Normal

X = Normal(5,  1).rv
Y = Normal(20, 3).rv
Z = X.mul(3)+Y
```

Listing 4.10: Adding a reference distribution to a sum 10 times using the *UPROP* Library. We initialize a Random Variable $X \sim \mathcal{N}(0,1)$ and we add this variable to the sum: `sum += X` for 10 times.

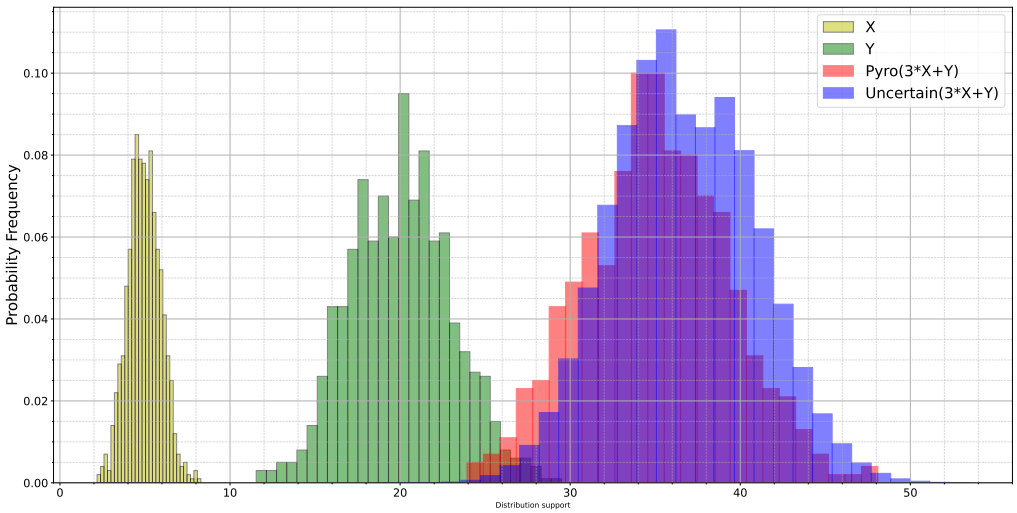
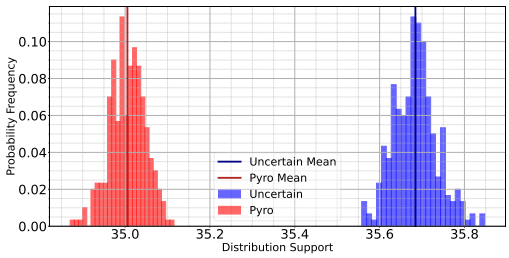
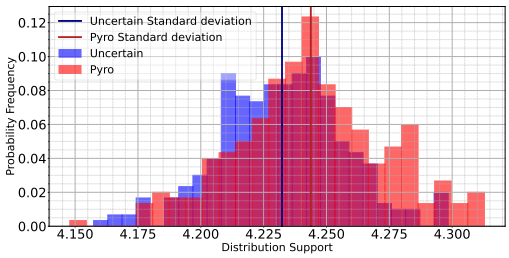


Figure 4.25: Applying linear operations on Random Variables. Comparison between the *uncertain package* and Pyro [54]. The result is stored in variable $Z = 3 * X + Y$. The Random Variables are $X \sim \mathcal{N}(5, 1)$ and $Y \sim \mathcal{N}(20, 3)$.



(a) Distribution of **mean values** for the `Uncertain` object and the Pyro object.



(b) Distribution of **standard deviation values** for the `Uncertain` object and the Pyro object.

Figure 4.26: Based on equations 2.2.1 and 2.2.2 the result should have a mean $\mu_{3*X+Y} = 3*\mu_X + \mu_Y = 35$ and a standard deviation of $\sigma = \sqrt{3^2\sigma_X^2 + \sigma_Y^2} = \sqrt{18} \approx 4.24$. We observe an offset of 0.8 from the expected mean value, with regards to the mean value of *uncertain package*, while Pyro yields the expected results. The standard deviation σ, remains almost identical.

To apprehend in depth the distributional output of this benchmark repeat the process k times and keep the mean and standard deviation of Random Variable Z. We plot the distribution of these values in Figure 4.26. While **the standard deviation remains almost the same** for both frameworks, **the mean value diverges** — in this example by approximately 0.7.

4.7.1 Compounding error

In Figure 4.26 where we compare the distribution of the first two centralized moments of the results of each framework we observe that the `Uncertain` object's mean deviates by a margin of approximately 0.7 from the expected value of 35. This happens due to the memory representation chosen for `Uncertain` objects. We should note that we observe this *"shift-to-the-right"* behavior, in every benchmark. As we explain in Section 3.2 the memory representation of any `Uncertain` object is a **Dirac mixture representation** (Definition 3.2.1). We propagate uncertainty through a circular-convolution, which we showcase as pseudocode in Listing 3.2.

As we show throughout this section, the accuracy of our model increases when we increase the number of bins representing an `Uncertain` variable. The reason behind this lack of accuracy with less bins is not the actual number of bins, but in fact the bin width. Less bins means less groups of samples with higher probability. Thus, since we only represent each bin with 2 numbers: its probability and its median, we end up partially distorting the distributional information obtained from the samples.

This issue is further enhanced by performing operations with only the median values. By doing so we generate new samples and we bin them accordingly such as the smallest generated sample becomes the first bin edge and the largest becomes the last bin edge. All this while keeping the probability of each value. One simple observation, is that we ignore the first sample value that originally formed the first `Uncertain` object. This results in a right shift of the minimum possible value and explains the issue we notice in this benchmark.

Chapter 5

Related Research

The *UPROP* Python Library attempts to encapsulate measurement uncertainty by using a distributional memory representation to propagate uncertainty through operations and functions. Designing frameworks like this, poses a massive challenge for both the mathematical foundations and in terms of efficiency. There have been multiple attempts to face this challenge which we lists and compare with the *UPROP* Python Library below. In Table 5.1 we present a brief comparison

System	**Hardware/ Software**	**Uncertainty Propagation Method**	**Programming Language**	**Execution**
UPROP	Software	Circular Convolution	Python	Locally
Signaloid.io	Software & Hardware	-	C	Online
NIST Uncertainty Machine	Software	Monte Carlo Simulation	R	Online
Pyro	Software	-	Python	Locally
Uncertain<T>	Software	Sampling Functions	C#	Locally

Table 5.1: Mean inverse-Normalized Wasserstein distance between the Particle and the *UPROP* Library implementation for the arithmetic operators for each number of bins.

Stan [55] is a domain-specific language designed for describing a restricted class of probabilistic programs and performing high-quality automated inference in those models. Church [56], a probabilistic dialect of Scheme, was an

early universal probabilistic programming language, capable of representing any computable probability distribution. Venture [57] is a universal language with a focus on expressiveness and flexibility and a custom syntax and virtual machine. Anglican [58] and webPPL [59] are lightweight successors to Church, embedded as syntactic subsets [60] in the general-purpose programming languages Clojure and JavaScript. Edward [61] is a PPL built on static TensorFlow graphs that features composable representations for models and inference. ProbTorch [62] is a PPL built on PyTorch with a focus on deep generative models and developing new objectives for variational inference. Turing [63] is a PPL embedded in Julia featuring composable MCMC algorithms. Pyro [54] is a probabilistic programming language (PPL) written in Python and supported by PyTorch on the backend, which is a widely used platform for deep learning. With this said, Pyro is mainly focused on Machine Learning (ML) tasks. Pyro enables flexible and expressive deep probabilistic modelling for any computable distribution, even empirical while unifying modern deep learning and Bayesian modelling.

Laplace Microarchitecture

Tsoutsouras et al. [2] present *Laplace Microarchitecture* for tracking machine representations of probability distributions paired with architectural state. Laplace uses the RISC-V ISA with two suggested extensions to induce distributional information to the microarchitecture and enable users to query and extract information about the distribution of the random variables. It represents the variables using either Telescoping torques representation (TTR) or Regularly-quantized representation (RHQR) which simulates histograms. TTR using the definition of *mean value*, that splits the support of the distribution in two halves, uniformly on each side of the mean, represents the distribution in a Dirac Mixture of $\log(N)$ Dirac delta functions $\delta(x)$ — a unit impulse at position x. They generate these Dirac deltas by recursively splitting each half of the distribution support, using the mean value for each half. The probability of each possible variable value defines the height of each Dirac delta. Laplace also supports correlation-tracking between variables by tracking ancestors on a hardware level.

Bornholt's Uncertain Type

Bornholt et al. [3] introduce *Uncertain<T>*, a programming language abstraction to represent random variables. Bornholt et al. use user-specified sampling functions to represent the variables, combined with a directed acyclic graph (DAG), constructed on runtime, to represent the order of arithmetic operations. This DAG forms a Bayesian Network and evaluated it whenever necessary. As seen in Figure 5.1, Bornholt corrects the calculation of the posterior by using prior knowledge. They achieve this by fitting the likelihood distribution

calculated by the framework, with prior knowledge. The result of the fitting constitutes the final posterior.

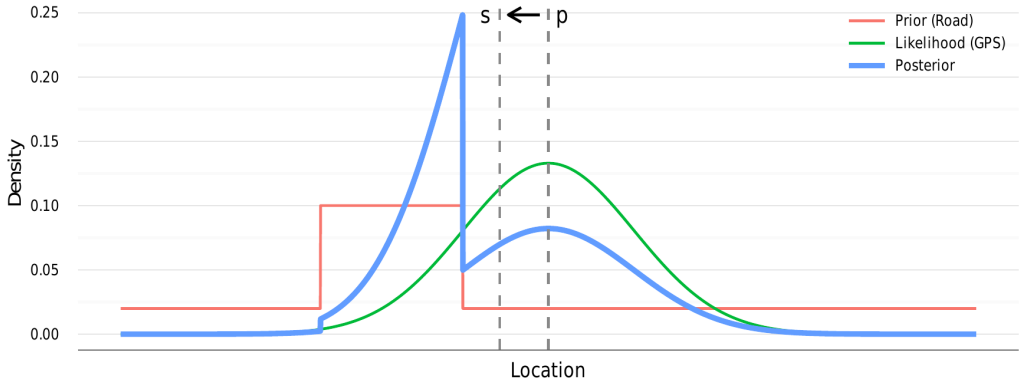


Figure 5.1: Bornholt fits calculated estimates generated by the Uncertain Type with prior knowledge to produce the posterior distribution of the data.

Uncertain-UP

Manousakis et al. [64] represent random variables using a mean, variance-covariance matrix, and when necessary the entire empirical distribution. Their work applies to DAG-based data processing systems, such as Apache or Hadoop. They use first-order Differential Analysis to propagate uncertainty through DAG nodes under the condition that they represent continuous and differentiable functions. In case this condition is not met, they use Monte Carlo simulation as a fallback method.

NIST Uncertainty Machine

The NIST Uncertainty Machine [53] is a web-based software application to evaluate the measurement uncertainty associated with a scalar output quantity that is a known and explicit function of a set of input quantities for which estimates and evaluations of measurement uncertainty is available. Random variables model the input and output quantities and they use their probability distributions to characterize measurement uncertainty. It supports both known and empirical probability distributions. The memory representation of the random variables consists of the first two centralized moments of the distribution (mean, variance). The NIST Uncertainty Machine implements the approximate method of uncertainty evaluation described in the GUM [1], and the Monte Carlo method of the GUM Supplements 1 and 2.

Chapter 6

Conclusion

6.1 Future Work

The *UPROP* Python Library undertakes the challenge to encapsulate measurement uncertainty by using a distributional memory representation. Designing frameworks like this, poses a massive challenge for both the mathematical foundations and in terms of efficiency. Below we propose some interesting capabilities one can explore now that we have the ability to manage and propagate measurement uncertainty. Our implementation can serve as a basis for this exploration.

6.1.1 Efficiency in Computations

As we briefly discuss in Section 4.2, when computing with the *UPROP* Python Library the selection of the number of bins that represent an `Uncertain` object embodies the classic speed-accuracy trade-off. At small bin numbers, the calculation is quick, but more inaccurate. At large bin numbers, the reverse is true: the calculation is slow but accurate. Choosing the correct bin number is therefore critically important: too high and the `Uncertain` type will be too slow for practical use; too low and it will be too inaccurate to solve real problems.

The `Uncertain` type certainly introduces a performance overhead when compared to particle (one-sample) calculations. Finding ways to either reduce or eliminate unnecessary computations to optimize this performance overhead and ultimately reduce execution time, would be the next major step in calculating with the *UPROP* Python Library.

6.1.2 Correlation tracking between variables

In Subsection 4.3.5, we analyze some of the reasons why tracking correlation becomes both a blessing and a curse in some situations. Correlation may not

imply causation, but causation does certainly imply correlation. On the other hand, this may introduce significant inaccuracy to our model as we show in the GPS benchmark. A specific condition where this is prevalent, is mapping even functions to `Uncertain` objects that have distribution support close to 0.

A typical solution to this problem might be to urge the user to explicitly redefine objects after specific operations that meet some conditions. But this would alter the design goal of minimality.

6.1.3 Compounding error after operations

Finally, in Section 4.7 we showcase the compounding error caused by using Dirac Mixtures (histograms) to propagate uncertainty in operations. Performing operations with the median values of each bin, results in a "right-shift" of the results since we ignore some samples. We expected this design driven behavior and we explicitly kept it because we consider it worth for the simplicity-to-accuracy trade-off.

However it would be interesting if we manage to combat this compounding error caused by operations by analyzing the input Random Variables's bin widths and counter-shifting the any new `Uncertain` objects.

6.2 Conclusion

The issues that software engineers are confronting are getting to be more complex, and equivocal, requiring them to inference information from a handful of sensor measurements to PetaBytes of Big Data. These information sources and the programs that utilize them, have one thing in common: they introduce uncertainty. By providing insufficient tools and abstractions, programming languages encourage programmers to waive their responsibilities in the face of uncertainty. Existing abstractions, either are not expressive enough or demand a significant amount of time for a developer to comfortably and effectively use them.

The *UPROP* Python Library aims to eliminate this, by addressing uncertainty propagation and management without excess action from the user side. The *UPROP* Python Library is a principled abstraction for computing with uncertain data. The idea of the abstraction we propose originates from uncertainty in computations and the fact that developers choose to ignore it. Reports show that we have acquired the know-how to identify *measurement uncertainty* but lack in tools that propagate it through computations. Our abstraction starts by defining a memory representation of distributions and uses mathematical knowledge to set the foundation of uncertainty propagation.

While this principled approach is a virtue and ensures the `Uncertain` type is accessible and expressive, it also has the potential to make a practical im-

plementation impossible. The selection of representation bins in an `Uncertain` object embodies the classic speed-accuracy trade-off. At small bin numbers, the calculation is quick, but more inaccurate. At high bin numbers, the reverse is true: the calculation is slow but accurate. Choosing the correct bin number is therefore critically important: too high and the `Uncertain` type will be too slow for practical use; too low and it will be too inaccurate to solve real problems.

Bibliography

[1] "Guide to the expression of uncertainty in measurement," Joint Committee for Guides in Metrology, Tech. Rep., Mar. 2008.

[2] V. Tsoutsouras, O. Kaparounakis, B. Bilgin, C. Samarakoon, J. Meech, J. Heck, and P. Stanley-Marbell, "The laplace microarchitecture for tracking data uncertainty and its implementation in a RISC-v processor." ACM, Oct. 2021.

[3] J. Bornholt, T. Mytkowicz, and K. S. McKinley, "Uncertain<T>: A first-order type for uncertain data," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. Association for Computing Machinery, 2014, p. 51–66.

[4] "Python programming language." [Online]. Available: WelcometoPython.org.(n. d.).Python.Org.RetrievedDecember9,2021,fromhttps://www.python.org/

[5] C. Berg, "Indeterminate moment problems and the theory of entire functions," *Journal of Computational and Applied Mathematics*, vol. 65, no. 1-3, pp. 27–55, 1995.

[6] M. Karson, "Handbook of Methods of Applied Statistics. Volume I: Techniques of Computation Descriptive Methods, and Statistical Inference. Volume II: Planning of Surveys and Experiments. I. M. Chakravarti, R. G. Laha, and J. Roy, New York, John Wiley; 1967,," *Journal of the American Statistical Association*, vol. 63, no. 323, pp. 392–394, Sep. 1968. [Online]. Available: https://doi.org/10.1080/01621459.1968.11009335

[7] S. Kuhlen, C. Stampfer, T. Wilhelm, and J. Kuhn, "Phyphox bringt das Smartphone ins Rollen," *Physik in unserer Zeit*, vol. 48, no. 3, pp. 148–149, 2017.

[8] M. Anderson, F. Schulz, Y. Lu, H. Kitaguchi, P. Bowen, C. Argyrakis, and H. Basoalto, "On the modelling of precipitation kinetics in a turbine disc nickel based superalloy," *Acta Materialia*, vol. 191, pp. 81–100, 2020.

[9] L. Brown and R. Ham, "Dislocation-particle interactions," *Strengthening methods in crystals*, pp. 9–135, 1971.

[10] M. Mohan Rayguru, M. Rajesh Elara, B. Ramalingam, J. Muthugala, M. Viraj, P. Samarakoon, and S. Bhagya, "A path tracking strategy for car like robots with sensor unpredictability and measurement errors," *Sensors*, vol. 20, no. 11, p. 3077, 2020.

[11] O. Kaparounakis, V. Tsoutsouras, D. Soudris, and P. Stanley-Marbell, "Automated physics-derived code generation for sensor fusion and state estimation," 2020.

[12] A. Wong, S. Ho, O. Olusanya, M. V. Antonini, and D. Lyness, "The use of social media and online communications in times of pandemic covid-19," *Journal of the Intensive Care Society*, vol. 22, no. 3, pp. 255–260, 2021.

[13] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*, no. CONF. IEEE Signal Processing Society, 2011.

[14] R. Collobert, C. Puhrsch, and G. Synnaeve, "Wav2letter: an end-to-end convnet-based speech recognition system," *arXiv preprint arXiv:1609.03193*, 2016.

[15] G. López, L. Quesada, and L. A. Guerrero, "Alexa vs. siri vs. cortana vs. google assistant: a comparison of speech-based natural user interfaces," in *International Conference on Applied Human Factors and Ergonomics*. Springer, 2017, pp. 241–250.

[16] C. C. Robusto, "The cosine-haversine formula," *The American Mathematical Monthly*, vol. 64, no. 1, pp. 38–40, 1957.

[17] O. Kallenberg and O. Kallenberg, *Foundations of modern probability*. Springer, 1997, vol. 2.

[18] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, Mar. 1951, publisher: Institute of Mathematical Statistics. [Online]. Available: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-1/On-Information-and-Sufficiency/10.1214/aoms/1177729694.full

[19] T. W. Anderson, "On the distribution of the two-sample cramer-von mises criterion," *The Annals of Mathematical Statistics*, pp. 1148–1159, 1962.

[20] H. Levy, "Stochastic dominance and expected utility: Survey and analysis," *Management science*, vol. 38, no. 4, pp. 555–593, 1992.

[21] C. Villani, "The Wasserstein distances," in *Optimal Transport: Old and New*, ser. Grundlehren der mathematischen Wissenschaften, C. Villani, Ed. Berlin, Heidelberg: Springer, 2009, pp. 93–111. [Online]. Available: https://doi.org/10.1007/978-3-540-71050-9_6

[22] A. Ramdas, N. G. Trillos, and M. Cuturi, "On Wasserstein Two-Sample Testing and Related Families of Nonparametric Tests," *Entropy*, vol. 19, no. 2, p. 47, Feb. 2017. [Online]. Available: https://www.mdpi.com/1099-4300/19/2/47

[23] "EMD." [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RUBNER/emd.htm

[24] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, vol. 40, no. 2, pp. 99–121, 2000.

[25] K. Ni, X. Bresson, T. Chan, and S. Esedoglu, "Local histogram based segmentation using the wasserstein distance," *International journal of computer vision*, vol. 84, no. 1, pp. 97–111, 2009.

[26] "Kolmogorov–Smirnov test," Oct. 2021, page Version ID: 1051865496. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Kolmogorov%E2%80%93Smirnov_test&oldid=1051865496

[27] O. Thas, *Comparing distributions*. Springer, 2010, vol. 233.

[28] C. Z. Mooney, *Monte carlo simulation*. Sage, 1997, no. 116.

[29] D. Wells, N. Beck, A. Kleusberg, E. J. Krakiwsky, G. Lachapelle, R. B. Langley, K.-p. Schwarz, J. M. Tranquilla, P. Vanicek, and D. Delikaraoglou, "Guide to gps positioning," in *Canadian GPS Assoc*. Citeseer, 1987.

[30] "Google earth." [Online]. Available: htts://earth.google.com

[31] H. Wang, Y. Gu, and S. Kamijo, "Pedestrian positioning in urban city with the aid of Google maps street view," in *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, May 2017, pp. 456–459.

[32] A. Ofstad, E. Nicholas, R. Szcodronski, and R. R. Choudhury, "AAMPL: accelerometer augmented mobile phone localization," in *Proceedings of the first ACM international workshop on Mobile entity localization and tracking in GPS-less environments*, ser. MELT '08. New York, NY, USA: Association for Computing Machinery, Sep. 2008, pp. 13–18. [Online]. Available: https://doi.org/10.1145/1410012.1410016

[33] "Android location accuracy - api level 1." [Online]. Available: https://developer.android.com/reference/android/location/Location

[34] "horizontalAccuracy | Apple Developer Documentation." [Online]. Available: https://developer.apple.com/documentation/corelocation/cllocation/1423599-horizontalaccuracy

[35] P. S. Laplace, "Memoir on the Probability of the Causes of Events," *Statistical Science*, vol. 1, no. 3, pp. 364–378, 1986, publisher: Institute of Mathematical Statistics. [Online]. Available: https://www.jstor.org/stable/2245476

[36] M. K. Simon, *Probability Distributions Involving Gaussian Random Variables*. New York, Ny: Springer, Nov. 2006. [Online]. Available: https://www.biblio.com/book/probability-distributions-involving-gaussian-random-variables/d/501173421

[37] S. Bhatt, A. Jain, and A. Dev, "Continuous speech recognition technologies—a review," *Recent Developments in Acoustics*, pp. 85–94, 2021.

[38] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 4960–4964.

[39] A. Chern, Y.-H. Lai, Y.-P. Chang, Y. Tsao, R. Y. Chang, and H.-W. Chang, "A smartphone-based multi-functional hearing assistive system to facilitate speech recognition in the classroom," *IEEE Access*, vol. 5, pp. 10 339–10 351, 2017.

[40] A. P. Markopoulou, F. A. Tobagi, and M. J. Karam, "Assessing the quality of voice communications over internet backbones," *IEEE/ACM transactions on networking*, vol. 11, no. 5, pp. 747–760, 2003.

[41] H.-W. Gierlich and F. Kettler, "Advanced speech quality testing of modern telecommunication equipment: An overview," *Signal processing*, vol. 86, no. 6, pp. 1327–1340, 2006.

[42] N. Lesser and D. Ellis, "Clap detection and discrimination for rhythm therapy," in *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 3, 2005, pp. iii/37–iii/40 Vol. 3.

[43] D. H. Mershon and J. N. Bowers, "Absolute and relative cues for the auditory perception of egocentric distance," *Perception*, vol. 8, no. 3, pp. 311–322, 1979.

[44] L. Peltola, C. Erkut, P. R. Cook, and V. Valimaki, "Synthesis of hand clapping sounds," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 3, pp. 1021–1029, 2007.

[45] D. H. Johnson, "Signal-to-noise ratio," *Scholarpedia*, vol. 1, no. 12, p. 2088, 2006.

[46] P. Bergmans, "A simple converse for broadcast channels with additive white gaussian noise (corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 279–280, 1974.

[47] R. Tandra and A. Sahai, "Snr walls for signal detection," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 1, pp. 4–17, 2008.

[48] T. B. Parrish, D. R. Gitelman, K. S. LaBar, and M.-M. Mesulam, "Impact of signal-to-noise on functional mri," *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 44, no. 6, pp. 925–932, 2000.

[49] E. Scheirer and M. Slaney, "Construction and evaluation of a robust multifeature speech/music discriminator," in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, 1997, pp. 1331–1334 vol.2.

[50] G. Peeters, "A large set of audio features for sound description (similarity and classification) in the cuidado project," *CUIDADO Ist Project Report*, vol. 54, no. 0, pp. 1–25, 2004.

[51] G. Tzanetakis and P. Cook, "Multifeature audio segmentation for browsing and annotation," in *Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. WASPAA'99 (Cat. No. 99TH8452).* IEEE, 1999, pp. 103–106.

[52] S. Dixon, "Onset detection revisited," in *Proceedings of the 9th International Conference on Digital Audio Effects*, vol. 120. Citeseer, 2006, pp. 133–137.

[53] T. Lafarge and A. Possolo, "The NIST Uncertainty Machine," *NCSLI Measure*, vol. 10, no. 3, pp. 20–27, 2015.

[54] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep universal probabilistic programming," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 973–978, 2019.

[55] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," *Journal of statistical software*, vol. 76, no. 1, pp. 1–32, 2017.

[56] N. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, "Church: a language for generative models," *arXiv:1206.3255 [cs]*, Jul. 2014, arXiv: 1206.3255. [Online]. Available: http://arxiv.org/abs/1206.3255

[57] V. Mansinghka, D. Selsam, and Y. Perov, "Venture: a higher-order probabilistic programming platform with programmable inference," *arXiv:1404.0099 [cs, stat]*, Mar. 2014, arXiv: 1404.0099. [Online]. Available: http://arxiv.org/abs/1404.0099

[58] D. Tolpin, J.-W. van de Meent, H. Yang, and F. Wood, "Design and implementation of probabilistic programming language anglican," in *Proceedings of the 28th Symposium on the Implementation and Application of Functional programming Languages*, 2016, pp. 1–12.

[59] N. D. Goodman and A. Stuhlmüller, "The Design and Implementation of Probabilistic Programming Languages," http://dippl.org, 2014, accessed: 2021-11-13.

[60] D. Wingate, A. Stuhlmueller, and N. Goodman, "Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, Jun. 2011, pp. 770–778, iSSN: 1938-7228. [Online]. Available: https://proceedings.mlr.press/v15/wingate11a.html

[61] D. Tran, A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, and D. M. Blei, "Edward: A library for probabilistic modeling, inference, and criticism," 2017.

[62] N. Siddharth, B. Paige, J.-W. van de Meent, A. Desmaison, N. D. Goodman, P. Kohli, F. Wood, and P. Torr, "Learning disentangled representations with semi-supervised deep generative models," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5927–5937.

[63] H. Ge, K. Xu, and Z. Ghahramani, "Turing: A language for flexible probabilistic inference," in *International conference on artificial intelligence and statistics*. PMLR, 2018, pp. 1682–1690.

[64] I. Manousakis, I. n. Goiri, R. Bianchini, S. Rigo, and T. D. Nguyen, "Uncertainty propagation in data processing systems," in *Proceedings of the ACM Symposium on Cloud Computing*. Association for Computing Machinery, 2018, p. 95–106.