SoftwareX 30 (2025) 102161

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Kneeliverse: A universal knee-detection library for performance curves

Mário Antunes [a,*], Tyler Estro [b], Pranav Bhandari [c], Anshul Gandhi [b], Geoff Kuenning [e], Yifei Liu [b], Carl Waldspurger [f], Avani Wildani [d,c], Erez Zadok [b]

[a] Instituto de Telecomunicacoes, Universidade de Aveiro, Campus Universitario de Santiago, Aveiro, 3810-193, Portugal
[b] Stony Brook University, Computer Science Building, Engineering Dr., Stony Brook, 11794, NY, USA
[c] Emory University, Mathematics & Science Center, Suite W401, 400 Dowman Drive, Atlanta, 30322, GA, USA
[d] Cloudflare, 101 Townshend Rd, San Francisco, 94107, CA, USA
[e] Harvey Mudd College, Department of Computer Science, 301 Platt Boulevard, Claremont, 91711, CA, USA
[f] Carl Waldspurger Consulting, Palo Alto, CA, USA

ARTICLE INFO

Keywords:
Knee estimation
Multi-knee estimation
Optimization
Python

ABSTRACT

Identifying knee and elbow points in performance curves is a critical task in various domains, including machine learning and system design. These points represent optimal trade-offs between cost and performance, facilitating efficient decision-making and resource allocation. However, accurately determining the knees and elbows in curves poses a significant challenge. To address this challenge, we introduce *Kneeliverse*, an open-source library dedicated to knee/elbow point detection. Kneeliverse incorporates a suite of well-established knee-detection algorithms, including Menger, L-method, Kneedle, and DFDT. Additionally, Kneeliverse extends these algorithms to detect multiple knees and elbows in complex curves, employing a recursive approach. Kneeliverse further includes Z-Method, a recently developed algorithm specifically designed for multi-knee detection.

Code metadata

Current code version	Version 1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00437
Permanent link to Reproducible Capsule	https://github.com/mariolpantunes/knee/releases/tag/1.0
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	python
Compilation requirements, operating environments & dependencies	OS-independent with only 2 dependencies: numpy and pyUTSAlgorithms
If available, link to developer documentation/manual	https://mariolpantunes.github.io/knee/kneeliverse.html
Support email for questions	mario.antunes@ua.pt

1. Motivation and significance

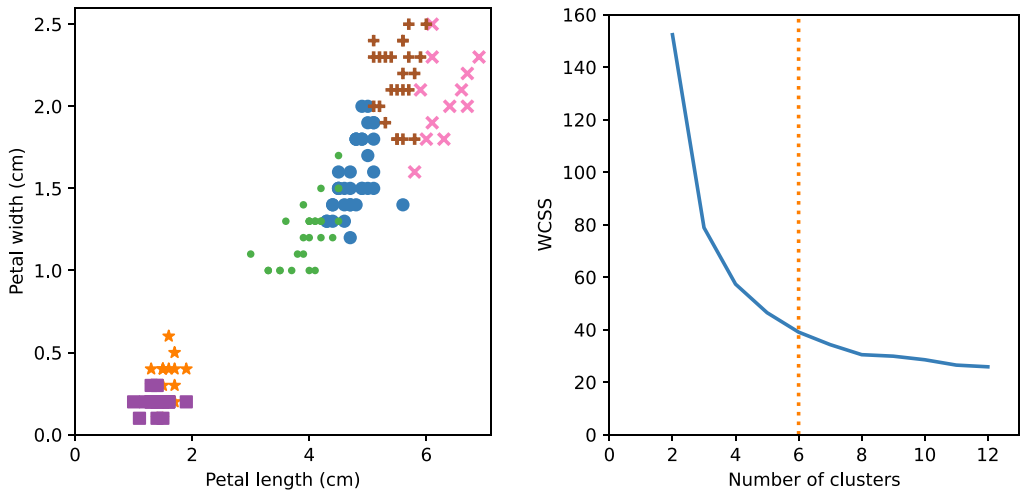
Identifying knee and elbow points in performance curves is critical for efficient decision making and resource allocation in various domains, including Machine Learning (ML) and systems design. For example, Yao et al. used knee detection to identify optimal controller parameters, maximizing performance while minimizing resource consumption [1]. In multi-objective optimization, targeting knee points on the Pareto front [2] improves efficiency and accuracy by focusing on the most optimal trade-offs.

Although the terms "knee" and "elbow" are sometimes used interchangeably to refer to inflection points, a distinction is often made. A knee signifies a downward bend in the curve (negative concavity), whereas an elbow indicates an upward bend (positive concavity). For simplicity, we use "knee" to refer to both types of points, since our library handles both equally well.

A knee point is typically defined as the point with maximum curvature. For continuous functions, curvature can be defined mathematically as in Eq. (1), where $f(x)$ represents a specific instance of a

* Corresponding author.
E-mail address: mario.antunes@av.it.pt (M. Antunes).

https://doi.org/10.1016/j.softx.2025.102161
Received 14 August 2024; Received in revised form 3 April 2025; Accepted 7 April 2025
Available online 14 May 2025



(a) 2D representation of the Iris dataset, plotting only petal width vs. petal length. Six clusters were chosen using the elbow method. Each color and marker represents a different cluster.

(b) Visual representation of the elbow method. The blue solid line represents the Within-Cluster Sum of Squares and the orange dotted line shows the knee as selected by the Kneedle algorithm.

Fig. 1. Clustering the Iris dataset using the elbow method.

parameterized curve and x denotes the parameter along the curve.

$$K_{f(x)} = \frac{f''(x)}{(1 + f'(x)^2)^{\frac{3}{2}}} \tag{1}$$

However, in most real-world scenarios, we work with discrete sets of points instead of continuous functions, which prohibits direct application of the curvature equation [3]. As such, most knee-detection algorithms were developed to operate on discrete datasets. Several methods exist that utilize the curvature of a discrete sequence to identify knees: Menger curvature [3], L-method [4], Kneedle [5], and Dynamic First Derivative Thresholding (DFDT) [3].

Due to the importance and utility of knee detection we introduce Kneeliverse, a comprehensive, open-source library designed for universal knee detection. The library provides reference implementations of the most well-known knee-detection methods. In addition, Kneeliverse offers recent methods specifically designed for multi-knee detection.

In the concrete example of cluster analysis, finding the optimal number of clusters in distinct datasets poses a challenging, yet essential task. (*e.g.*, K-means [6]). A popular approach, typically called the *elbow method* [7], involves finding knees using within-cluster dispersion curves. According to this method, the number of clusters is found by plotting the explained variation as a function of the number of clusters, and then choosing the cluster count that matches the elbow in that curve. The same method can be used to choose various parameters in other data-driven models, such as the number of principal components needed to describe a dataset.

To illustrate this application, consider clustering the Iris dataset [8], which consists of 50 samples from each of three species of Iris plants (Setosa, Virginica, and Versicolor). Four features were measured for each sample: the lengths and widths, in centimeters, of the sepals and petals. This dataset has been widely used as an example dataset for classification and clustering problems.[1]

Fig. 1(a) plots the resulting clusters from the Iris dataset, where each color represents a different cluster. Six clusters were selected through the elbow method [7], as depicted in Fig. 1(b).

The quality of the clusters (computed with a centroid-based method, such as K-Means) can be measured with the Within-Cluster Sum of Squares (WCSS), defined as the sum of the squared distance between the average point (*i.e.*, the centroid) and each point of the cluster

$$WCSS = \sum_{i=1}^{k} \left(\sum_{x \in C_i} \left| x - \mu_i \right|^2 \right), \tag{2}$$

where k is the number of clusters, C_i is the ith cluster, x is a data point in cluster C_i, and μ_i is the centroid (mean) of cluster C_i. The blue curve in Fig. 1(b) represents the WCSS measure and the orange line represents the knee as selected by the Kneedle algorithm [5].

Conversely, multi-knee scenarios involve curves with several inflection points. The previous definition for a single knee point (see Eq. (1)) cannot be directly applied to detect these multi-knee points [9]. To address this, we propose a procedure to detect multi-knee points. Our procedure segments the functions after identifying the knee point, and recursively evaluates each new segment (see Definition 1).

Definition 1. An **admissible** algorithm for finding multi-knee points, acting on discrete function C, is a recursive procedure that takes as input a subset of C and a linearity threshold τ, and applies the following routine. The linearity threshold τ is a value between $[0.0, 1.0]$ that measures how close a subset of points is to a straight line.

Admissible recursive routine to find multi-knee points

```
function FindMultiKnee(discrete function C, left index x_i, right index x_j, linearity threshold τ)
    knee = max(K_{C(x_i,x_j)}) // Find knee k in range [x_i, x_j]
    if linearity(x_i, knee) ≤ τ then
        knees_left = FindMultiKnee(C, x_i, knee, τ)
    end if
    if linearity(knee, x_j) ≤ τ then
        knees_right = FindMultiKnee(C, knee, x_j, τ)
    end if
    return knees_left + knee + knees_right
end function
```

A concrete example of multi-knee detection is to identify "key points" in a Miss Ratio Curve (MRC) for cache simulation. A cache's miss ratio is an important predictor of its performance [10]. Recent methods, such as Z-Method [9], have been used to efficiently explore the design of multi-tier storage caches.

[1] https://scikit-learn.org/stable/datasets/toy_dataset.html

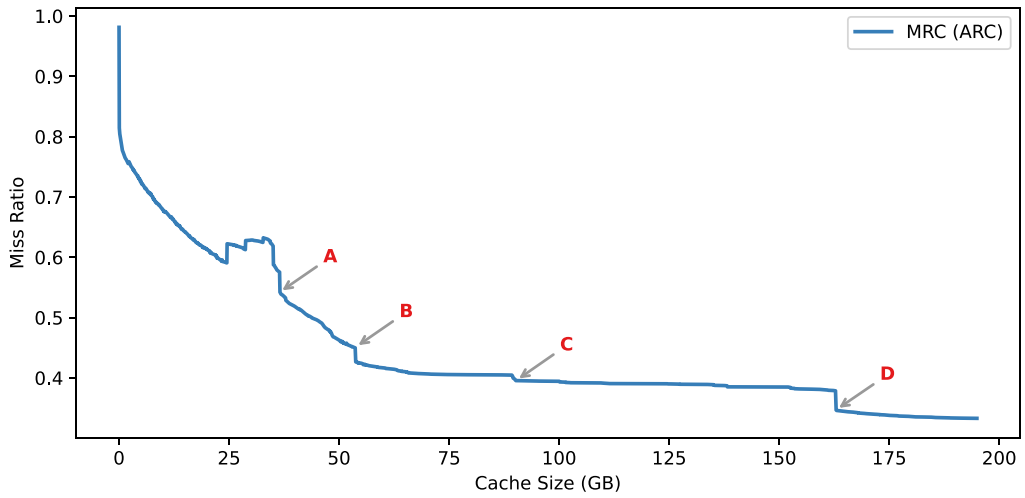


Fig. 2. Miss ratio curve (MRC) for trace `web0`, annotated to illustrate several key points: useful "knees" (points A and D), a useless "cliff" (B) and a non-relevant knee point (C).

To illustrate this, we plotted an MRC corresponding to the `web0` trace from Microsoft Research in the SNIA public dataset [11]. Intuitively, the most promising candidates are points where a little extra cache space produces a relatively large drop in the miss ratio; such points are often visible as "knees" in the MRC—*e.g.*, points A and D in Fig. 2. Note that although B has sharp curvature, it is not as useful since B is a "cliff". Finally, although the C point also represents a well-defined knee, its relevance in the overall trace is lower when compared with either A or D.

2. Software description

Kneeliverse [12] is an open-source library for single- and multi-knee detection in 2D performance curves. The code is distributed under the MIT license and is also available for easy installation via PyPI.[2]

The library provides reference implementations of the traditional single-knee-detection methods Menger, L-Method, Kneedle, and DFDT, which offer comparable performance to existing implementations. It also offers two recently developed multi-knee-detection methods: an extended version of Kneedle, and Z-Method. In addition, it includes a novel recursive algorithm capable of using any single-knee-detection algorithm for multi-knee detection.

The library also includes pre-processing routines that rely on an improved Ramer-Douglas-Peucker (RDP) curve-simplification algorithm and post-processing filters that select and rank the output of a multi-knee algorithm.

2.1. Architecture

Kneeliverse was written entirely in Python 3 with a minimal number of dependencies. It uses NumPy [13], which provides efficient numerical routines for large vectors, as well as pyUTSAlgorithms [14], a library that offers vector operations for unevenly spaced sequences of points.

A simplified version of Kneeliverse's pipeline is shown in Fig. 3. Depending on the complexity of the curve, one can choose whether to apply the pre-processing line simplification method (step A). The next step consists of applying either a single-knee or multi-knee detection method (step B). For multi-knee detection, one can use either of the two methods specially designed for that purpose (Kneedle or Z-Method), or alternatively apply the recursive approach that enables a single-knee algorithm to find multiple knees. Finally, we can apply the post-processing filter to reduce the number of knee candidates (step C).

2.1.1. Pre-processing

A curve can contain an arbitrary number of data points. However, knee-detection algorithms are prone to smoothing errors [9], as most were originally designed to work with small or partial data, such as for clustering analysis.

Kneeliverse provides pre-processing methods to reduce the number of points in a curve while preserving those that define its shape. This reduces the computational costs of subsequent steps while also improving knee-detection accuracy.

The RDP algorithm modifies a curve by finding a similar one with fewer points [15]. The main drawback of RDP is the need to define a threshold, which can be understood as the maximum allowed reconstruction error. We modified the original RDP algorithm to address this issue. Instead of a fixed threshold for perpendicular distance, we adopted a relevance-based cost metric that measures the difference between the fitted line and data points.

2.1.2. Knee-detection methods

Several methods exist for finding single knee points in a curve. Menger curvature [5,16] defines the curvature for a sequence of three points as the curvature of the circle circumscribed by those points. The *L*-method [4] fits two straight lines from the head of a curve to a candidate point, and from the candidate point to the curve's tail. The candidate that minimizes the Root Mean Square Error (RMSE) between the straight lines and the points of the curve is returned as the knee point. Similar to the *L*-method, DFDT [3,17] tries to identify the point where the function has a sharp angle. Instead of fitting two straight lines, this method relies on the first derivative of the curve. After computing that derivative, a thresholding algorithm is used to identify the threshold that separates the derivative values as "high" or "low." The knee is then the point with a derivative value that is closest to the previously computed threshold. Kneedle [5] uses the point on the curve that is furthest away from a line, defined by the head and tail points of the curve.

Except for Kneedle, the previously mentioned algorithms were not designed to detect multiple knees. Even the original Kneedle has some limitations [9]. Thus, we developed a recursive algorithm that can be used to adapt any single-knee-detection technique to handle multiple knees. The basic idea is to use a single-knee technique to select the best knee in a segment. We then split the current segment at that knee, and for each new segment check whether it is sufficiently linear. If not, we repeat the process recursively.

Finally, we also include our recently developed method Z-Method [9], which was inspired by the DFDT [17] and DSDT [3] knee-detection algorithms. In statistics, *z-score* (also known as *standard score*) is a

[2] https://pypi.org/project/kneeliverse/

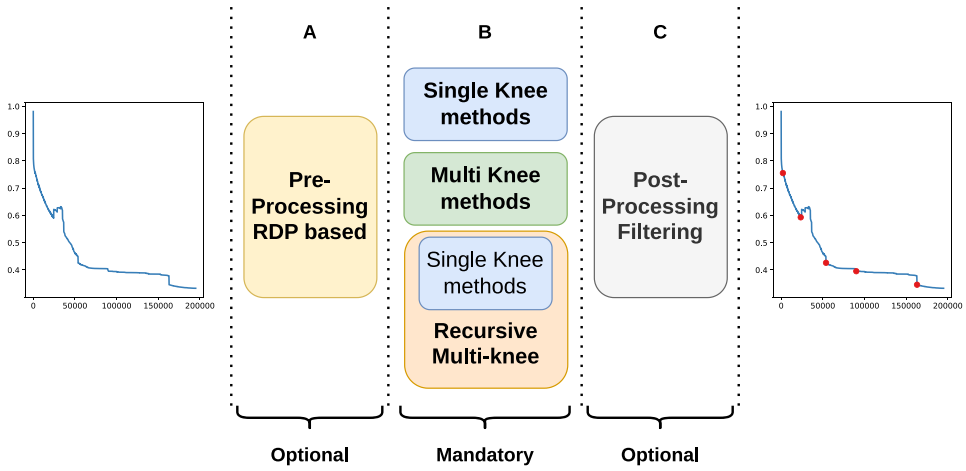
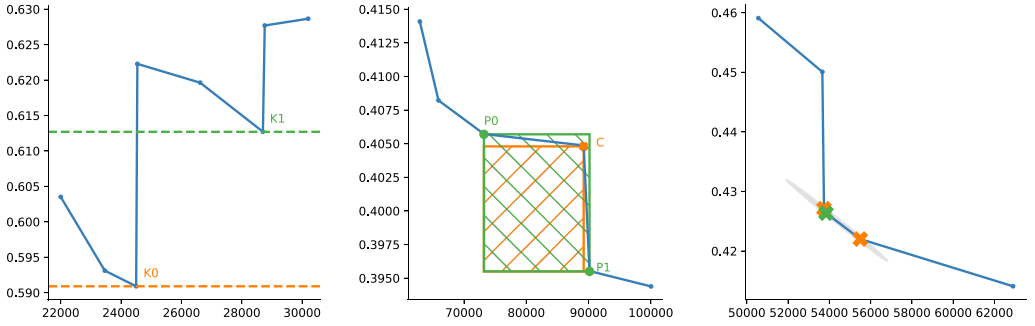


Fig. 3. Kneeliverse pipeline. This pipeline offers multiple options for knee detection. It supports three steps: (A) Pre-processing the curve (optional); (B) Applying the selected knee algorithm (mandatory) and (C) Post-processing filtering (optional).



(a) Unwanted-knee removal (b) Corner detection (c) Clustering and ranking algorithms

Fig. 4. Post-processing methods applied to the `web0` trace (subsets of the trace). The X-axis represents cache size, and the Y-axis represents miss ratio (lower values indicate better performance). (a) Removal of unwanted knees: K_1 is removed since K_0 achieved better performance (lower on the y-axis) at a lower cache size (lower on the x-axis). (b) Overlapping rectangles used by the corner-detection algorithm. Cliff point C is removed due to the area of the overlapping rectangles created by its adjacent points P_0 and P_1 exceeding the threshold. (c) Clustering and ranking elements. The gray ellipse represents the cluster of knee candidates. The orange candidates are filtered out, while the green knee is selected as the best representative. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

transformation that normalizes a data value by quantifying how many standard deviations away it is from the mean; typically, a point whose z-score has an absolute value greater than three is considered an outlier [18]. For the purpose of detecting knees, such outliers in the second derivative indicate a significant change in the curve's slope. For more details, see [9].

2.1.3. Post-processing

We developed three filters to further reduce and rank the knee points. These filters were created due to (1) the differences between single- and multi-knee-detection and (2) the large number of points produced by our recursive strategy with some knee-detection algorithms.

The first filter, shown in Fig. 4(a), removes useless knees. When dealing with non-monotonic curves, a knee-detection algorithm can incorrectly choose a knee that is *above* a previously detected one. We remove such knees since they are sub-optimal and do not add useful information.

The second filter, shown in Fig. 4(b), removes cliff points located after a smooth, near-horizontal area that precedes a sharp descent. These points are found using a corner-detection algorithm that computes the overlapping area of two rectangles. The filter computes the percentage overlap between these two rectangles, and a knee candidate is removed if the overlap exceeds a threshold. Point B in Fig. 2 is a good example of a cliff.

The third and final filter, shown in Fig. 4(c), uses a hierarchical clustering algorithm to group knees by their distance along the x-axis, using a percentage of the x range as a threshold. After creating clusters, the knees within each cluster are ranked based on their relevance score, computed using two metrics: (i) the improvement given by the knee (*i.e.*, how much it decreases on the y-axis compared to the highest knee in the cluster) and (ii) the smoothness of the improvement, computed using the coefficient of determination (R^2). For each cluster, the knee with the highest value of S is selected as its representative.

2.2. Functionality

Next we describe Kneeliverse's main functionality. More complete documentation is available online.[3]

2.2.1. Pre-processing

The pre-processing functionality is in the module *knee.rdp*. There are two variants of the RDP algorithm, where the main difference is in how the reconstruction cost is computed (either by segment or on the whole curve).

The functions are the following:

[3] https://mariolpantunes.github.io/knee/kneeliverse.html

- *rdp*: Decimates a curve composed of line segments into a similar curve with fewer points. It uses different cost functions to decide when to keep or remove a line segment.
- *mapping*: Computes the reverse of the RDP method. It maps the indexes from the simplified curve (using the RDP algorithm) into the indexes of the original points, and returns the original 2D numpy vector.

2.2.2. Methods

Each knee-detection method is in a separate module with a similar interface. In any module, the functions are the following:

- *knee*: Single-knee detection. This function utilizes the chosen knee-detection algorithm to identify the optimal single knee point.
- *knees*: For methods that support native multi-knee detection, this function executes the method to identify the existing knees.
- *multi_knee*: This function is available for knee-detection methods that lack native multi-knee support. It employs our recursive approach to identify all existing knees.

2.2.3. Post-processing

The functionality for post-processing is contained within a module named *knee.postprocessing*, which provides a set of filters designed to improve the quality of the knee candidates. The functions are the following:

- *filter_worst_knees*: Filter the worst knee points. A worse knee is a knee that is higher on the y-axis than a previous one.
- *filter_corner_knees*: Filter the upper-left corner knee points. An upper-left knee corner does not provide a significant improvement to be considered.
- *filter_clusters*: Filter the knee points based on clustering. For each cluster, a single point is selected based on a ranking that is computed based on the slope and the improvement on the y-axis.

2.3. Performance comparison

We compared the Kneeliverse's single-knee detection methods on a small, synthetic dataset generated with *scikit-learn*. Multi-knee evaluation is detailed in [19,20]. We measured MAE (average difference between the expected knee values and the predicted knee value) and runtime (nanoseconds), summarized in Table 1.

DFDT and Kneedle achieved the highest performance (lower MAE), with a slight advantage for Kneedle. L-method and Menger have similar performance in this synthetic dataset. When considering execution time, the reverse trend was observed.

3. Illustrative examples

This example illustrates the standard workflow employed with this library. To demonstrate this, consider the trace initially presented in Fig. 2. We developed an example that identifies significant knees in the trace. The following is the source code for this example:

```
#!/usr/bin/env python3

__author__ = 'Mario Antunes'
__version__ = '1.0'
__email__ = 'mario.antunes@ua.pt'
__status__ = 'Development'
__license__ = 'MIT'
__copyright__ = '''
Copyright (c) 2021-2024 Stony Brook University
Copyright (c) 2021-2024 The Research Foundation of SUNY
'''

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

import kneeliverse.rdp as rdp
import kneeliverse.kneedle as kneedle
import kneeliverse.postprocessing as pp
import kneeliverse.clustering as clustering
import kneeliverse.knee_ranking as knee_ranking


def main(args):
    # Open the input file
    points = np.genfromtxt(args.i, delimiter=',')

    # Apply the global RDP line simplification
    # algorithm as a pre-processing step (Step A) to
    # reduce the number of data points while preserving
    # the essential shape of the data.
    reduced, removed = rdp.mp_grdp(points,
      t=0.001, min_points=20)
    points_reduced = points[reduced]

    # Utilize the Kneedle algorithm to identify all
    # potential knee points (Step B).
    knees = kneedle.knees(points_reduced,
      p=kneedle.PeakDetection.All)

    # Filter the knee candidates using the previously
    # mentioned post-processing filters (Step C).
    knees = pp.filter_worst_knees(points_reduced, knees)
    knees = pp.filter_corner_knees(points_reduced,
      knees, t=0.33)
    knees = pp.filter_clusters(points_reduced, knees,
      clustering.average_linkage,
      0.05, knee_ranking.ClusterRanking.left)

    # Transform the knee points, represented in the
    # reduced space, back to their corresponding
    # locations in the original trace space.
    knees = rdp.mapping(knees, reduced, removed)

    # Plot the original trace, overlaying the locations
    # of the final knee points.
    plt.plot(x, y)
    plt.plot(x[knees], y[knees], 'o', markersize=7)
    plt.show()


if __name__ == '__main__':
    parser = argparse.ArgumentParser(
    description='Plot the Multi-Knees using kneedle')
    parser.add_argument('-i', type=str,
    required=True, help='input file')
    args = parser.parse_args()
    main(args)
```

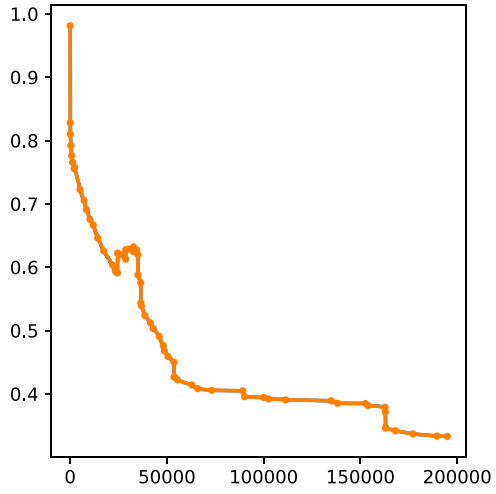
Code 1: Sample code to estimate the multi-knee points in a complex trace.

To gain a clearer understanding of the individual steps outlined in Code 1, we have included a sequence of figures depicting the results of each step (see Fig. 5).

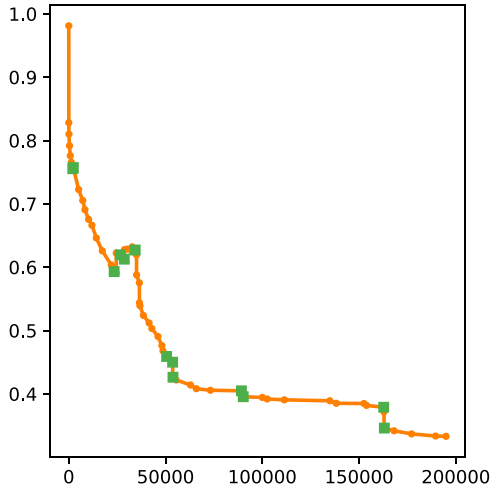
The source code loads a trace from a CSV file and utilizes the RDP algorithm to reduce the number of points in the trace. The original trace has approximately 250,000 points, whereas the reconstructed trace contains only 20 (see Fig. 5(a)). Following point reduction, the Kneedle algorithm is applied (Fig. 5(b)), and the identified knee candidates are filtered to remove irrelevant knees (Fig. 5(c)). Finally, the indices of the selected knee points are mapped back to the original trace (since they were calculated on the reduced trace) and visualized (Fig. 5(d)).

Table 1
Analysis of the knee detection methods. The performance was measured using MAE and the execution time as measured in nanoseconds.

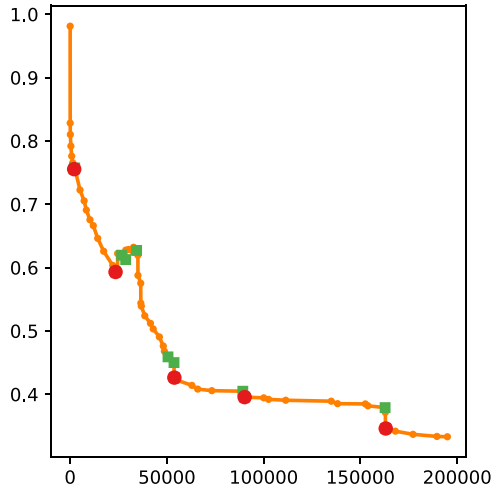
Method/	Menger		L-method		DFDT		Kneedle	
#Knees	Time	Error	Time	Error	Time	Error	Time	Error
5	4.23E4	10	4.23E4	3	2.24E5	3	1.20E5	1
6	3.90E4	9	3.90E4	4	1.19E5	4	7.26E4	2
7	3.46E4	8	3.46E4	4	1.16E5	5	9.09E4	3
8	3.16E4	4	3.16E4	5	1.23E5	6	6.92E4	4
9	3.18E4	6	3.18E4	5	1.23E5	7	6.91E4	4
	3.59E4	7.4	3.59E4	4.2	1.41E5	5	8.44E4	2.8



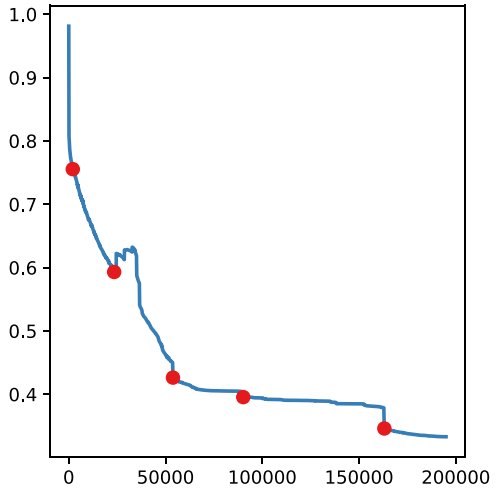
(a) Representation of the miss ratio curve of the original trace (250,000 points) in blue and the reduced version (with 20 points, computed by RDP) in orange.



(b) Knee candidates as selected by Kneedle (our improved version), in green.



(c) Result of applying the post-processing filters to remove unwanted knees. The green points are filtered out, while the red points are selected as final knees.



(d) Plotting the final knees back on the original trace.

Fig. 5. A visual representation of each step in a typical pipeline using Kneeliverse, in the `web0` trace.

4. Impact

Kneeliverse is an accessible and well-documented library targeting a wide range of applications. We initially developed it to simplify configuring and tuning multi-tier caching (MTC) systems, where exponential configuration growth demands efficient solutions. MTC is an important research topic that spans many areas, including VM management [21], heterogeneous networks [22], cloud storage [23], hardware design [24,25], and data centers [26,27].

In our previous work [9], our Kneeliverse-based framework reduced the number of points needed to accurately identify optimal two-tier

cache hierarchies for a diverse real-world dataset by an average factor of 5.5× for ARC and 7.7× for LRU eviction policies.

Knee detection is widely used in a variety of other domains to speed up computation or improve accuracy. Yeh et al. identified knees to find the ideal rank of k-dimensional motifs [28], reducing their solution search space. Jiang et al. proposed KT-DMOEA, a knee-based transfer learning method for solving dynamic multi-objective optimization problems [29]. Yue et al. developed a knee-based particle swarm optimization algorithm to solve sparse reconstruction problems [30]. In power system optimization, knees help determine the optimal number of capacitors to minimize losses and maximize economic efficiency [31].

In time series forecasting, finding knees in the objective space can guide the search and improve the prediction accuracy [32]. In drug combination therapy [33], knees are selected on Pareto fronts generated by optimization algorithms to identify treatments that optimally balance multiple objectives.

In lane detection, knees represent optimal edge detection thresholds for robust performance [34]. In rumination analysis, knee detection refines the identification of rumination periods in accelerometer data [35]. In symmetry detection, they help to determine the optimal level of human–machine agreement [36].

Knee detection also optimizes the number of labeling functions in data programming systems like Snorkel [37], improves decision-making in COVID-19 risk assessment [38], and enhances video streaming quality by identifying optimal trade-offs between video quality and user engagement [39]. In network anomaly detection [40], knees refine anomaly detection thresholds for improved accuracy and efficiency.

Finally, several other domains apply knee detection successfully: i) evaluating lithium-ion batteries [41–44], ii) network congestion control [5], iii) structural building modeling [45,46], iv) efficient drainage design and water management [47,48]

5. Conclusion

We have developed Kneeliverse, an open-source library that identifies knee points. Our library caters to a wide range of applications, spanning educational, academic, and industrial settings. One prominent use case lies in determining the optimal number of clusters for unsupervised learning tasks.

The growing challenge of identifying multiple knee points in highly intricate curves has driven the evolution of Kneeliverse. The current iteration provides a comprehensive pipeline capable of accurately detecting multiple knees in complex performance curves.

CRediT authorship contribution statement

Mário Antunes: Writing – review & editing, Writing – original draft, Validation, Software, Investigation, Data curation, Conceptualization. **Tyler Estro:** Writing – review & editing, Writing – original draft, Software, Investigation, Data curation, Conceptualization. **Pranav Bhandari:** Writing – original draft. **Anshul Gandhi:** Writing – review & editing, Writing – original draft, Validation. **Geoff Kuenning:** Writing – review & editing, Writing – original draft, Validation. **Yifei Liu:** Supervision, Data curation. **Carl Waldspurger:** Writing – review & editing, Writing – original draft, Validation. **Avani Wildani:** Writing – review & editing, Writing – original draft, Validation. **Erez Zadok:** Writing – review & editing, Writing – original draft, Validation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This work was made possible in part thanks to Dell-EMC, NetApp, Facebook, and IBM support; a SUNY/IBM Alliance award; and NSF awards CCF-1918225, CNS-1750109, CNS-1900589, CNS-1900706, CNS-1951880, CNS-2106263, CNS-2106434, and CNS-2214980. This work is also partially funded by FCT - Fundação para a Ciência e Tecnologia, I.P. by project reference UIDB/50008, and DOI identifier 10.54499/UIDB/50008.

References

[1] Yao Z, Yao J, Sun W. Adaptive RISE control of hydraulic systems with multilayer neural-networks. IEEE Trans Ind Electron 2019;66(11):8638–47. http://dx.doi.org/10.1109/tie.2018.2886773.

[2] Li W, Wang R, Zhang T, Ming M, Li K. Reinvestigation of evolutionary many-objective optimization: Focus on the Pareto knee front. Inform Sci 2020;522:193–213. http://dx.doi.org/10.1016/j.ins.2020.03.007.

[3] Antunes M, Ribeiro J, Gomes D, Aguiar RL. Knee/elbow point estimation through thresholding. In: 6th IEEE international conference on future internet of things and cloud. Barcelona, Spain: IEEE; 2018, p. 413–9.

[4] Salvador S, Chan P. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In: 16th IEEE international conference on tools with artificial intelligence. IEEE; 2004, p. 576–84.

[5] Satopaa V, Albrecht J, Irwin D, Raghavan B. Finding a "Kneedle" in a haystack: Detecting knee points in system behavior. In: 31st international conference on distributed computing systems workshops. Minneapolis, MN: IEEE; 2011, p. 166–71.

[6] Lloyd S. Least squares quantization in PCM. IEEE Trans Inform Theory 1982;28(2):129–37.

[7] Thorndike RL. Who belongs in the family? Psychometrika 1953;18(4):267–76. http://dx.doi.org/10.1007/bf02289263.

[8] Fisher RA. Iris. 1988, http://dx.doi.org/10.24432/C56C76, UCI Machine Learning Repository.

[9] Estro T, Antunes M, Bhandari P, Gandhi A, Kuenning G, Liu Y, et al. Accelerating multi-tier storage cache simulations using knee detection. Perform Eval 2024;164:102410. http://dx.doi.org/10.1016/j.peva.2024.102410, URL https://www.sciencedirect.com/science/article/pii/S0166531624000154.

[10] Estro T, Bhandari P, Wildani A, Zadok E. Desperately seeking ... Optimal multi-tier cache configurations. In: Proceedings of the 12th USeNIX workshop on hot topics in storage. Boston, MA: USENIX; 2020.

[11] Narayanan D, Donnelly A, Rowstron A. MSR Cambridge traces (SNIA IOTTA trace set 388). In: Kuenning G, editor. SNIA IOTTA trace repository. Storage Networking Industry Association; 2007, URL http://iotta.snia.org/traces/block-io?only=388.

[12] Antunes M, Estro T, Bhandari P, Gandhi A, Kuenning G, Liu Y, et al. (Multi)Knee/Elbow point detection library. 2023, http://dx.doi.org/10.5281/zenodo.10341887.

[13] Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.

[14] Antunes M, Liu Y. pyUTSAlgorithms (unevenly spaced time series algorithms). 2024, URL https://github.com/mariolpantunes/pyUTSAlgorithms.

[15] Ramer U. An iterative procedure for the polygonal approximation of plane curves. Comput Graph Image Process 1972;1(3):244–56.

[16] Tolsa X. Principal values for the Cauchy integral and rectifiability. Proc Amer Math Soc 2000;128(7):2111–9.

[17] Antunes M, Gomes D, Aguiar RL. Knee/Elbow estimation based on first derivative threshold. In: Fourth IEEE international conference on big data computing service and applications. Bamberg, Germany: IEEE; 2018, p. 237–40.

[18] Aggarwal CC. Outlier analysis. 3rd ed.. Springer Publishing Company; 2016, Incorporated.

[19] Estro T, Antunes M, Bhandari P, Gandhi A, Kuenning G, Liu Y, et al. Guiding simulations of multi-tier storage caches using knee detection. In: 31st international symposium on the modeling, analysis, and simulation of computer and telecommunication systems. 2023.

[20] Estro T, Antunes M, Bhandari P, Gandhi A, Kuenning G, Liu Y, et al. Guiding simulations of multi-tier storage caches using knee detection. Tech. Rep., (FSL-23-01). Computer Science Department, Stony Brook University; 2023.

[21] Rajasekaran S, Duan S, Zhang W, Wood T. Multi-cache: Dynamic, efficient partitioning for multi-tier caches in consolidated VM environments. In: IEEE international conference on cloud engineering. IEEE; 2016, p. 182–91. http://dx.doi.org/10.1109/IC2E.2016.10.

[22] Li X, Wang X, Li K, Han Z, Leung VC. Collaborative multi-tier caching in heterogeneous networks: Modeling, analysis, and design. IEEE Trans Wirel Commun 2017;16(10):6926–39.

[23] Spillane RP, Shetty PJ, Zadok E, Archak S, Dixit S. An efficient multi-tier tablet server storage architecture. In: Proceedings of the 2nd ACM symposium on cloud computing. Cascais, Portugal; 2011.
[24] Nori AV, Gaur J, Rai S, Subramoney S, Wang H. Criticality aware tiered cache hierarchy: A fundamental relook at multi-level cache hierarchies. In: 45th ACM/IEEE annual international symposium on computer architecture. 2018, p. 96–109. http://dx.doi.org/10.1109/ISCA.2018.00019.
[25] Srikantaiah S, Kultursay E, Zhang T, Kandemir M, Irwin MJ, Xie Y. MorphCache: A reconfigurable adaptive multi-level cache hierarchy. In: 2011 IEEE 17th international symposium on high performance computer architecture. IEEE; 2011, p. 231–42.
[26] Yang Z, Hoseinzadeh M, Andrews A, Mayers C, Evans D, Bolt R, et al. AutoTiering: Automatic data placement manager in multi-tier all-flash datacenter. In: 2017 IEEE 36th international performance computing and communications conference. 2017, p. 1–8. http://dx.doi.org/10.1109/PCCC.2017.8280433.
[27] Liu Z, Lee HW, Xiang Y, Grunwald D, Ha S. eMRC: Efficient miss rate approximation for multi-tier caching. In: 19th USeNIX conference on file and storage technologies. USENIX Association; 2021, URL https://www.usenix.org/conference/fast21/presentation/liu.
[28] Yeh C-CM, Kavantzas N, Keogh E. Matrix profile VI: Meaningful multidimensional motif discovery. In: 2017 IEEE international conference on data mining. 2017, p. 565–74. http://dx.doi.org/10.1109/ICDM.2017.66.
[29] Jiang M, Wang Z, Hong H, Yen GG. Knee point-based imbalanced transfer learning for dynamic multiobjective optimization. IEEE Trans Evol Comput 2021;25(1):117–29. http://dx.doi.org/10.1109/TEVC.2020.3004027.
[30] Yue C, Liang J, Qu B, Song H, Li G, Han Y. A knee point driven particle swarm optimization algorithm for sparse reconstruction. In: Simulated evolution and learning. Cham: Springer International Publishing; 2017, p. 911–9.
[31] Duong MQ, Lam LH, Tu BTM, Huy GQ, Hieu NH. A combination of K-mean clustering and elbow technique in mitigating losses of distribution network. 2019, URL https://api.semanticscholar.org/CorpusID:203634225.
[32] Du W, Leung SYS, Kwong CK. Time series forecasting by neural networks: A knee point-based multiobjective evolutionary algorithm approach. Expert Syst Appl 2014;41(18):8049–61. http://dx.doi.org/10.1016/j.eswa.2014.06.041.
[33] Spolaor S, Papetti DM, Cazzaniga P, Besozzi D, Nobile MS. A comparison of multi-objective optimization algorithms to identify drug target combinations. In: 2021 IEEE conference on computational intelligence in bioinformatics and computational biology. IEEE; 2021, p. 1–8. http://dx.doi.org/10.1109/cibcb49929.2021.9562773.
[34] Suddamalla U, Kundu S, Farkade S, Das A. A novel algorithm of lane detection addressing varied scenarios of curved and dashed lanemarks. In: 2015 international conference on image processing theory, tools and applications. IEEE; 2015, p. 87–92.
[35] Hamilton AW, Davison C, Tachtatzis C, Andonovic I, Michie C, Ferguson HJ, et al. Identification of the rumination in cattle using support vector machines with motion-sensitive bolus sensors. Sensors 2019;19(5):1165.
[36] Funk C, Liu Y. Symmetry reCAPTCHA. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, p. 5165–74.
[37] Ratner A, Bach SH, Ehrenberg H, Fries J, Wu S, Ré C. Snorkel: Rapid training data creation with weak supervision. Proc VLDB Endow Int Conf Very Large Data Bases 2017;11:269.
[38] Cuevas E. An agent-based model to evaluate the COVID-19 transmission risks in facilities. Comput Biol Med 2020;121:103827.
[39] Ahmed A, Shafiq Z, Bedi H, Khakpour A. Suffering from buffering? Detecting QoE impairments in live video streams. In: 2017 IEEE 25th international conference on network protocols. IEEE; 2017, p. 1–10.
[40] Dromard J, Roudière G, Owezarski P. Online and scalable unsupervised network anomaly detection method. IEEE Trans Netw Serv Manag 2016;14(1):34–47.
[41] Fermín-Cueto P, McTurk E, Allerhand M, Medina-Lopez E, Anjos MF, Sylvester J, dos Reis G. Identification and machine learning prediction of knee-point and knee-onset in capacity degradation curves of lithium-ion cells. Energy AI 2020;1:100006. http://dx.doi.org/10.1016/j.egyai.2020.100006.
[42] Strange C, Li S, Gilchrist R, dos Reis G. Elbows of internal resistance rise curves in Li-Ion cells. Energies 2021;14(4):1206. http://dx.doi.org/10.3390/en14041206.
[43] Attia PM, Bills A, Brosa Planella F, Dechent P, dos Reis G, Dubarry M, et al. Review—"Knees" in Lithium-Ion battery aging trajectories. J Electrochem Soc 2022;169(6):060517. http://dx.doi.org/10.1149/1945-7111/ac6d13.
[44] Costa N, Anseán D, Dubarry M, Sánchez L. ICFormer: A deep learning model for informed lithium-ion battery diagnosis and early knee detection. J Power Sources 2024;592:233910. http://dx.doi.org/10.1016/j.jpowsour.2023.233910.
[45] Gong W, Tien YM, Juang CH, Martin JR, Zhang J. Calibration of empirical models considering model fidelity and model robustness — Focusing on predictions of liquefaction-induced settlements. Eng Geol 2016;203:168–77. http://dx.doi.org/10.1016/j.enggeo.2015.11.003.
[46] Gong W, Tang H, Wang H, Wang X, Juang CH. Probabilistic analysis and design of stabilizing piles in slope considering stratigraphic uncertainty. Eng Geol 2019;259:105162. http://dx.doi.org/10.1016/j.enggeo.2019.105162.
[47] Yu Y, Shen M, Sun H, Shang Y. Robust design of siphon drainage method for stabilizing rainfall-induced landslides. Eng Geol 2019;249:186–97. http://dx.doi.org/10.1016/j.enggeo.2019.01.001.
[48] Null SE, Olivares MA, Cordera F, Lund JR. Pareto optimality and compromise for environmental water management. Water Resour Res 2021;57(10). http://dx.doi.org/10.1029/2020wr028296.