SPTK4: An Open-Source Software Toolkit for Speech Signal Processing

Anonymous submission to SSW 2023

Abstract

The SPTK is an open-source suite of speech signal processing tools, which has been developed and maintained by the SPTK working group. It has been widely contributed to the speech signal processing community since 1998. Although the SPTK has reached over a hundred thousand downloads, the concepts as well as the features have not yet been widely disseminated. This paper gives an overview of the SPTK and some demonstrations to provide a better understanding of the toolkit. We have recently developed its differentiable PyTorch version, *diffsptk*, to move with the deep learning era. The details of the diffsptk are also presented in the paper. We hope that the toolkit will help developers and researchers working in the field of speech signal processing.

Index Terms: digital signal processing, open-source software, DDSP

1. Introduction

There are many types of speech applications such as speech synthesis, speech recognition, speaker recognition, and speech coding. In order to boost speech research and facilitate the development of speech products, it is beneficial to develop an open-source, general-purpose speech signal processing toolkit.

The Speech Signal Processing ToolKit, SPTK, was originally developed and used in the research group of Satoshi Imai and Takao Kobayashi at Tokyo Institute of Technology. The tools can be used via a command-line interface (CLI) on the UNIX environment. Some of the tools were repackaged by Keiichi Tokuda as the organizer in collaboration with Takashi Masuko and Kazuhito Koishida, and then were distributed as the SPTK version 1.0[1] in 1998. The source code of the distribution including data processing, graph drawing, sample rate conversion, Fourier transform, speech analysis, speech synthesis, and vector quantization was written in the traditional C language. In 2000, the SPTK version 2.0[2] was released with an additional 30 tools, bringing the total to about 100 tools. Note that the SPTK versions 1.0 and 2.0 were not approved for commercial use. Then the SPTK version 3.0[3] appeared in 2002 with the modified BSD license to be friendly to product development. The only difference from the SPTK version 2.0 was the license. The SPTK version 3.0 was then improved year by year. It was mainly maintained by the members of the research group of Keiichi Tokuda at Nagoya Institute of Technology, and matured into the version 3.11 in 2017. These versions have been openly maintained on the SourceForge platform and have been widely used in the speech signal processing community [1, 2, 3, 4] for a long time. However, the source code was not fairly readable, which made it difficult to understand the implemented algorithms and to modify the source code. In addition, the features were not sufficiently portable due to that the source code was written in the C language.

To address these issues, we rewrote the SPTK version 3.11 using the C++ language while retaining the core design of the SPTK, which is described in detail in the next section. Although it could be rewritten with the Python language, we selected the C++ one because of the importance of speed performance and the ability to be used in embedded platforms. We used the Python language only for graph drawing to generate modern images using sophisticated Python plotting libraries. The new SPTK was released as version 4.0[4] in 2021 with some additional tools. It continues to be maintained in the public GitHub repository. With the migration, the license was changed from the modified BSD license to the Apache License 2.0.

The new version of SPTK is readable and highly portable. However, the implemented features are not compatible with modern deep learning frameworks. The integration of deep learning with speech signal processing techniques is an interesting research area and the effectiveness has begun to be shown in various contexts [5, 6, 7, 8, 9]. We thus started to export the SPTK features to be compatible with one of the famous deep learning frameworks, PyTorch [10]. The exported SPTK library has been publicly distributed as *diffsptk*[5] since 2022. It includes some special signal processing modules such as mel-cepstral analysis [11] and mel-cepstral synthesis filtering [12], which are not implemented in other signal processing libraries [9, 13]. The details are described in Section 4.

The remaining problem with the SPTK is that the concepts and the features are not well explained, making it difficult for users to approach the SPTK. To solve the problem, we introduce the concepts of the SPTK as an open-source speech signal processing library and present the main features of the SPTK with some demonstrations to help users understand the tools. We also present the concepts of the differentiable version of the SPTK for differentiable digital signal processing (DDSP).

Related work Table 1 shows a summary of some signal processing libraries. Although there is some overlap in the table, the SPTK provides some unique features, especially in speech analysis and synthesis. The SPTK can work complementary with other libraries.

[1] https://sourceforge.net/projects/sp-tk/files/SPTK/SPTK-1.0/

[2] https://sourceforge.net/projects/sp-tk/files/SPTK/SPTK-2.0/

[3] https://sourceforge.net/projects/sp-tk/files/SPTK/SPTK-3.0/

[4] https://github.com/sp-nitech/SPTK/releases

[5] https://github.com/sp-nitech/diffsptk

2. Design

We design the SPTK based on the following policies:

- **Raw data format**: The data used in the SPTK do not have any headers or structures. Any data compression is not used. The raw data format enables users to read the contents of data files immediately via a binary file dump. This is very helpful to check the sanity of data in experiments. In addition, the data generated by the SPTK can be used in other software through simple binary reading. This policy is opposite to other well-known libraries such as Kaldi archive format (.ark), hierarchical data format (.hdf), and binary data format in NumPy (.npy). The data type used in the SPTK is basically little-endian 64-bit double (version 4.0 or higher), or 32-bit float (version 3.11 or lower).
- **Standard I/O-based**: The SPTK consists of over 100 commands. The commands basically receive input data from the standard input and send the processed data to the standard output. This means that users can perform complex data processing by combining the SPTK commands using the pipe command (|) in Unix-like computer operating systems. The SPTK commands of course can chain with basic UNIX commands such as *cat*, *less*, and *wc*. This policy makes the SPTK intuitive and easy to use and is a unique design not found in other tools [15, 16]. To prevent contamination of data, the error or warning messages from the SPTK commands are output to the standard error, not the standard output.
- **Non-interactive**: The SPTK commands do not require interactive user inputs. The parameters that control data processing, e.g., frame shift in speech analysis, must be set via command line options beforehand.
- **Minimum requirements**: The SPTK intentionally avoids the use of external libraries such as Eigen [17]. While importing more external libraries makes the development of the SPTK easier, some users or systems may not be able to install the libraries due to their machine environments. Furthermore, using multiple libraries makes licensing complicated and less user-friendly. To avoid the problems, we have implemented signal processing algorithms from scratch including the fast Fourier transform (FFT).
- **Thread-safe** (version 4.0 or higher): The SPTK ensures thread safety for parallel data processing. Basically, a C++ class in the SPTK has a *Run* function to operate some data processing. The *Run* function typically requires the reference of input data, the pointer of output data, and the pointer of buffers as the arguments. By using different buffers in different threads, users can perform some data processing in parallel without unintended data access.
- **No memory leaks** (version 4.0 or higher): The older versions of SPTK have a possibility of memory leaks. To avoid this problem, we use std::vector in C++ instead of the malloc function for dynamic memory allocation. In addition, a memory mismanagement detector is used to check for memory leaks.

3. Features

The section describes the main features in the current version of SPTK.

3.1. Data type conversion

One of the most frequently used SPTK commands is *x2x*. The command converts the data type of input to a specific data type. The following example increases all the values in the short-type example file (*data.short*) by two times.

```
$ x2x +sd data.short | sopr -m 2 |
    x2x +da | less
```

The first *x2x* converts from short to double type for processing the example data in the other SPTK commands. The last *x2x* converts from double type to ASCII to show the processed example data on screen.

3.2. Data rearrangement

The order of data in the SPTK is represented as the following vector:

$$\begin{bmatrix} \boldsymbol{x}_0^\mathsf{T} & \boldsymbol{x}_1^\mathsf{T} & \cdots & \boldsymbol{x}_{T-1}^\mathsf{T} \end{bmatrix}^\mathsf{T}, \tag{1}$$

where $\boldsymbol{x}_t$ is a D-dimensional data vector $[x_{t,1}, x_{t,2}, \ldots, x_{t,D}]^\mathsf{T}$ and T is the length of data sequence. The SPTK can rearrange the data on a CLI, e.g.,

$$\begin{bmatrix} \boldsymbol{x}_S^\mathsf{T} & \boldsymbol{x}_{S+1}^\mathsf{T} & \cdots & \boldsymbol{x}_E^\mathsf{T} \end{bmatrix}^\mathsf{T} \tag{2}$$

is obtained by *bcut* where $S \geq 0$ and $E < T$, and

$$\begin{bmatrix} \boldsymbol{y}_0^\mathsf{T} & \boldsymbol{y}_1^\mathsf{T} & \cdots & \boldsymbol{y}_{T-1}^\mathsf{T} \end{bmatrix}^\mathsf{T} \tag{3}$$

is obtained by *bcp* where $\boldsymbol{y}_t = [x_{t,s}, x_{t,s+1}, \ldots, x_{t,e}]^\mathsf{T}$ is a subvector of $\boldsymbol{x}_t$ ($s \geq 1$ and $e \leq D$). Concatenation (*merge*), reversing (*reverse*), delaying (*delay*), transpose (*transpose*) operations are also provided.

3.3. Graph drawing

It is beneficial to understand data visually. The SPTK commands enable us to draw data (*fdrw*), waveform (*gwave*), discrete signals (*gseries*), log-spectrum (*glogsp*), running log-spectra (*grlogsp*), spectrogram (*gspecgram*), and pole-zero (*gpolezero*). These commands are implemented with the Python language using Plotly graphing library [18]. The examples in the paper use these commands but some options are omitted due to space limitation.

Table 1: *Summary of some open-source signal processing libraries*

	SPTK3	SPTK4	diffsptk	SciPy [14]	TorchAudio [13]
Language	C	C++	Python	Python	Python
UNIX-like commands	✓	✓			
Deep learning			✓		✓
Community	SourceForge	GitHub	GitHub	GitHub	GitHub
License	BSD 3-Clause	Apache 2.0	Apache 2.0	BSD 3-Clause	BSD 2-Clause

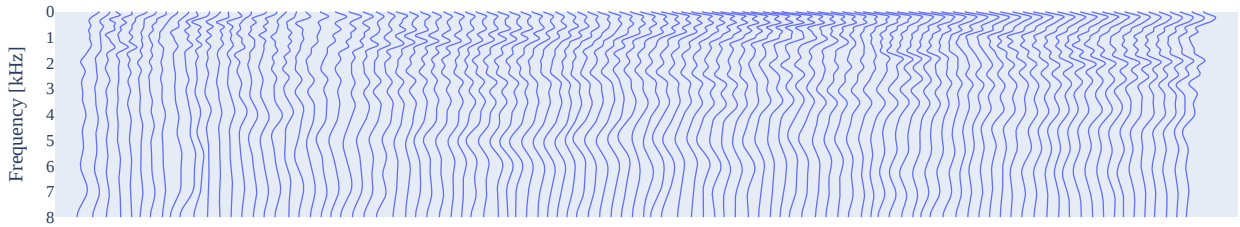

Figure 1: *Running spectra of the first 100 frames of the example data.*

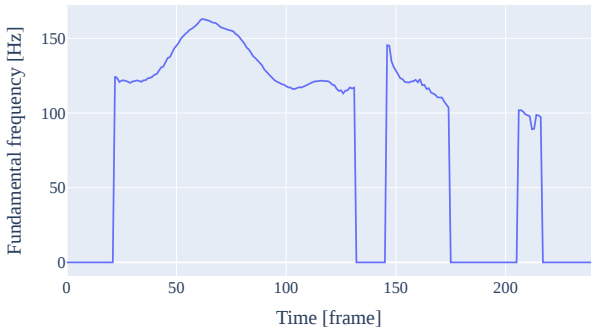

Figure 2: *A pitch contour of the example data.*

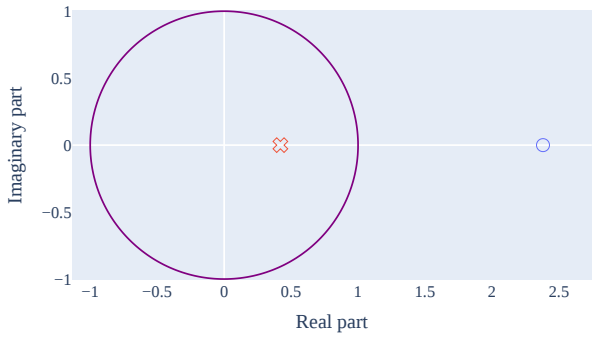

Figure 3: *Pole and zero of a first-order all-pass filter. Small circle and cross represent zero and pole, respectively.*

3.4. Spectral analysis

The example below computes spectra from a speech waveform using the short-term Fourier transform (STFT) with a 25-ms Blackman window and 5-ms frame shift.

```
$ x2x +sd data.short |
    frame -p 80 -l 400 |
    window -l 400 -L 512 |
    spec -l 512 > data.spec
```

Acoustic features can then be extracted from the spectra via autocorrelation analysis (*acorr, lpc*), adaptive mel-cepstral analysis (*amgcep*) [19, 20], or mel-generalized cepstral analysis (*mgcep*) [11, 21, 22]. The below example extracts 24-th order mel-cepstral coefficients from the obtained spectra.

```
$ mgcep -m 24 -l 512 -a 0.42 -q 0 \
    < data.spec > data.mc
```

Figure 1 shows the running spectra computed from the mel-cepstral coefficients of the first 100 frames of the example data. The figure is generated by the following command.

```
$ mgc2sp -m 24 -l 512 -a 0.42 data.mc |
    grlogsp -l 512 -t -x 16 -e 99 \
            -H 500 -W 2000 spec.pdf
```

3.5. Pitch analysis

Extraction of pitch contours of speech is an important signal processing. The SPTK provides a wrapper of sophisticated pitch extraction algorithms independently developed by third parties. The current implemented algorithms are RAPT [23], SWIPE' [24], REAPER [25], and DIO [26]. The example below extracts a pitch contour of the example data with 5-ms frame-shift by the RAPT algorithm.

```
$ x2x +sd data.short |
    pitch -a 0 -s 16 -p 80 > data.pit
```

Figure 2 shows the extracted F_0 contour obtained by the following command.

```
$ sopr -magic 0 -INV -m 16000 -MAGIC 0 \
    < data.pit | fdrw -g f0.pdf
```

In the command, *sopr* converts pitch [sec] to F_0 [Hz]. Note that the SPTK outputs an unvoiced symbol as 0 as a magic number. If $\log F_0$ as selected as the output format of pitch, the unvoiced symbol is represented as $-1e+10$. The SPTK also provides a command used for pitch mark (GCI) extraction (*pitch_mark*).

3.6. Speech synthesis (linear time-variant filtering)

The SPTK can reconstruct waveform from acoustic features given an excitation signal using a linear synthesis filter. The implemented synthesis filters are all-zero digital filter using impulse response (*zerodf*), all-pole digital filter using linear predictive coding (LPC) coefficients (*poledf*) [27], all-pole lattice digital filter using PARCOR coefficients (*ltcdf*), line spectral pairs (LSP) digital filter using LSP coefficients (*lspdf*) [28], and mel-log spectrum approximation (MLSA) digital filter using mel-cepstral coefficients (*mglsadf*) [19, 29]. In the following example, speech waveform is reconstructed from a simple excitation signal using the MLSA filter with the extracted mel-cepstral coefficients.

```
$ excite -p 80 data.pit |
    mglsadf -p 80 -m 24 -a 0.42 -P 7 \
    < data.mc | x2x +ds -r > syn.raw
```

The commands for checking the stability of these synthesis filters are provided (*lpccheck, lspcheck, mlsacheck*).

3.7. Linear time-invariant filtering

A signal can be processed using a finite/infinite impulse response digital filter. The below example applies a first-order all-pass filter to the example data:

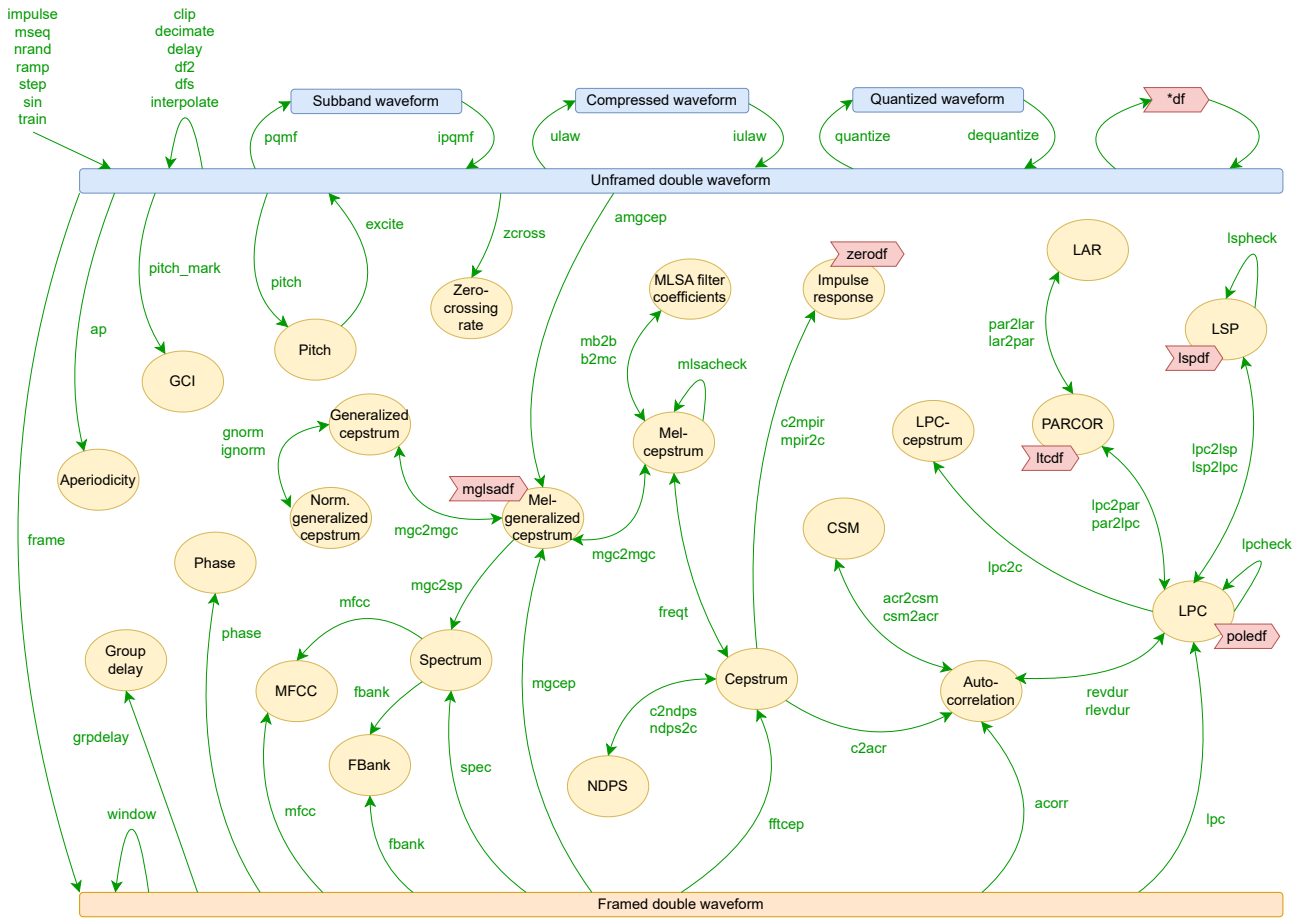

Figure 4: *Relationship between SPTK commands (strings in green) and data representations (circles in yellow). This block diagram is inspired by Mitch Bradley who kindly provided us the block diagram for the former versions of SPTK.*

```
$ x2x +sd data.short |
    dfs -a 1 -0.42 -b -0.42 1 > data.out
```

where the transfer function is

$$H(z) = \frac{-0.42 + z^{-1}}{1 - 0.42z^{-1}}. \tag{4}$$

The poles and zeros of a digital filter can be computed and plotted in the z-plane. Figure 3 plots the pole and the zero of the first-order all-pass filter $H(z)$. The figure is generated as the following manner.

```
$ echo 1 -0.42 | x2x +ad |
    root_pol -m 1 -q 0 > data.p
$ echo -0.42 1 | x2x +ad |
    root_pol -m 1 -q 0 > data.z
$ gpolezero -p data.p -z data.z pz.pdf
```

3.8. Mel-filter bank analysis

For speech recognition or speaker recognition, the SPTK supports mel-filter bank analysis (*fbank, mfcc*) based on the HTK [30]. The following example extracts 12-order mel-frequency cepstrum coefficients (MFCCs) with the first derivatives using a pre-emphasis filter.

```
$ x2x +sd data.short |
    frame -l 400 -p 160 -n 1 |
    dfs -b 1 -0.97 |
    window -l 400 -L 512 -w 1 -n 0 |
    mfcc -l 512 -n 40 -c 22 -m 12 |
        -L 64 -H 4000 -s 16 -q 4 -o 1 |
    delta -m 12 -d -0.5 0 0.5 > data.mfc
```

3.9. Parameter transformation

The relationship between the SPTK commands and data representation is illustrated in Fig. 4. As the figure shows, spectral parameters can be transformed into other representations including impulse response, autocorrelation, LPC coefficients, PARCOR coefficients, LSP coefficients [31], log area ratio (LAR), composite sinusoidal modeling (CSM) [32], cepstral coefficients, mel-cepstral coefficients, negative derivative of phase spectrum (NDPS) [33], etc.

3.10. Vector quantization

For speech coding, a codebook for vector quantization can be generated using the Linde-Buzo-Gray algorithm [34]. Vector quantization can then be performed with the generated codebook. The example below designs a codebook of the extracted mel-cepstral coefficients and reconstructs them from the encoded vector indices.

```
$ lbg -m 24 -e 32 data.mc > mc.cb
$ msvq -m 24 -s mc.cb < data.mc |
    imsvq -m 24 -s mc.cb > data.mc.dec
```

Multi-stage vector quantization is also supported.

3.11. Subband decomposition

Subband analysis and synthesis using pseudo-quadrature mirror filters (PQMFs) [35, 36] is supported. The filter coefficients will be designed to have a desired stopband attenuation. The example below decomposes the example data to two-channel signals and reconstructs them from the decomposed signals.

```
$ x2x +sd data.short |
    pqmf -k 2 -m 20 |
    decimate -l 2 -p 2 |
    interpolate -l 2 -p 2 |
    sopr -m 2 | ipqmf -k 2 -m 20 |
    x2x +ds -r > data.raw
```

3.12. Voice conversion

The SPTK provides the commands for Gaussian mixture model (GMM)-based voice conversion [37, 38]. The alignment between the feature vector sequence of a source speaker and a target speaker can be obtained by dynamic time warping (*dtw*). The joint feature vector consisting of the feature vectors of the source and target speakers can then be modeled by GMMs (*gmm*). Finally, a feature vector sequence of the target speaker can be predicted from the trained GMMs and a given feature vector sequence of the source speaker (*vc*). It is easy to append dynamic features to the feature vector sequences (*delta*) so that smoothed feature vector sequence of a target speaker can be obtained.

3.13. Distance calculation

It is important to evaluate experimental results based on some objective metrics. The SPTK can compute signal-to-noise ratio (SNR), root-mean-square error (RMSE), and cepstral distance [39] for the metrics. These commands accepts two inputs as follows.

```
$ cdist -m 24 -o 0 data.mc data.mc.dec |
    x2x +da
```

This example computes mel-cepstral distance in decibels and shows it on screen.

3.14. Statistics calculation

The SPTK can easily compute statistics of data by using a single command, e.g, average (*average*), summation (*vsum*), mean, covariance (*vstats*), median (*median*), minimum, and maximum (*minmax*). The example below shows the mean vector of the extracted mel-cepstral coefficients on screen.

```
$ vstat -m 24 -o 0 data.mc | x2x +da
```

4. PyTorch version

Incorporating digital signal processing techniques with deep neural networks is an interesting topic. Although some signal processing libraries for deep learning such as TorchAudio [13] have already been distributed, they have not implemented the core features of the SPTK. We have thus re-implemented some of the SPTK features based on a deep learning framework and provided them as a supplemental DDSP library. The name of the library is diffsptk since it is a differentiable version of the SPTK. We selected the PyTorch [10] as a deep learning framework because it is widely used by the deep learning community and easy to use.

We design the diffsptk based on the following polices:

- **Non-recursive**: The SPTK originally written in the C/C++ language involves recursive algorithms in the implementation within frequency warping [40], parameter transformations [22, 41], etc. This is suitable for non-parallel computation but is definitely unsuitable for DDSP using GPU parallel computing. To avoid slow training/inference, we have replaced the recursive implementation with a non-recursive one using some mathematical techniques such as matrix multiplication and FFT as much as possible.
- **Dimension-last**: A neural network module in PyToch accepts *tensors* as input and output. In diffsptk, the shape of the tensors is basically assumed as (B, T, D) rather than (B, D, T), where B is the mini-batch size, T is the length of data, and D is the dimension of data. This is more intuitive because the shape (B, T, D) is compatible with the C version of SPTK described in Eq. (1).
- **Precomputed**: The parameters corresponding to the command line options in the SPTK commands must be set via the constructor of a PyTorch module, not the forward function used at runtime. This is consistent with the C++ class in the SPTK4. The policy allows us to reduce computation time at runtime by performing calculations in advance that depend only on the parameters.
- **Consistent**: To ensure that the the new PyTorch implementation matches the original C++ implementation, unit tests for checking the consistency must be implemented and passed on each signal processing module.

5. Conclusions

This paper summarizes the core design, features, history, and current progress of SPTK. The SPTK provides useful UNIX-like signal processing commands and a library to support product developments and research experiments. The differential version of SPTK has begun to be distributed to follow the deep learning paradigm.

6. Acknowledgements

We thank all the contributors to SPTK. The names of the main contributors are listed at `https://github.com/sp-nitech/SPTK/blob/master/README.md`.

7. References

[1] H. Zen, K. Oura, T. Nose, J. Yamagishi, S. Sako, T. Toda, T. Masuko, A. W. Black, and K. Tokuda, "Recent development of the HMM-based speech synthesis system (HTS)," in *The 2009 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, 2009, pp. 121–130.

[2] K. Oura, A. Mase, T. Yamada, S. Muto, Y. Nankaku, and K. Tokuda, "Recent development of the HMM-based singing voice synthesis system - Sinsy," in *7th ISCA Workshop on Speech Synthesis (SSW 7)*, 2010, pp. 211–216.

[3] Z. Wu, O. Watts, and S. King, "Merlin: An open source neural network speech synthesis system," in *9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, 2016, pp. 202–207.

[4] Z. Wu, J. Yamagishi, T. Kinnunen, C. Hanilçi, M. Sahidullah, A. Sizov, N. Evans, M. Todisco, and H. Delgado, "ASVspoof: The automatic speaker verification spoofing and countermeasures

challenge," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 4, pp. 588–604, 2017.

[5] M. Ravanelli and Y. Bengio, "Speaker recognition from raw waveform with SincNet," in *2018 IEEE Spoken Language Technology Workshop (SLT)*, 2018, pp. 1021–1028.

[6] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, p. 6309–6318.

[7] J. Valin and J. Skoglund, "LPCNet: Improving neural speech synthesis through linear prediction," in *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 5891–5895.

[8] G. Yang, S. Yang, K. Liu, P. Fang, W. Chen, and L. Xie, "Multi-band MelGAN: Faster waveform generation for high-quality text-to-speech," in *2021 IEEE Spoken Language Technology Workshop (SLT)*, 2021, pp. 492–498.

[9] J. Engel, L. H. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable digital signal processing," in *International Conference on Learning Representations*, 2020.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[11] K. Tokuda, T. Kobayashi, T. Chiba, and S. Imai, "Spectral estimation of speech by mel-generalized cepstral analysis," *Electronics and Communications in Japan (Part 3)*, vol. 76, no. 2, pp. 30–43, 1993.

[12] T. Yoshimura, S. Takaki, K. Nakamura, K. Oura, Y. Hono, K. Hashimoto, Y. Nankaku, and K. Tokuda, "Embedding a differentiable mel-cepstral synthesis filter to a neural speech synthesis system," in *2023 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2023 (forthcoming).

[13] Y.-Y. Yang, M. Hira, Z. Ni, A. Chourdia, A. Astafurov, C. Chen, C.-F. Yeh, C. Puhrsch, D. Pollack, D. Genzel, D. Greenberg, E. Z. Yang, J. Lian, J. Mahadeokar, J. Hwang, J. Chen, P. Goldsborough, P. Roy, S. Narenthiran, S. Watanabe, S. Chintala, V. Quenneville-Bélair, and Y. Shi, "TorchAudio: Building blocks for audio and speech processing," *arXiv preprint arXiv:2110.15018*, 2021.

[14] P. Virtanen *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[15] F. Eyben, M. Wöllmer, and B. Schuller, "openSMILE: The Munich versatile and fast open-source audio feature extractor," in *Proceedings of the 18th ACM International Conference on Multimedia*, 2010, pp. 1459–1462.

[16] P. Boersma and D. Weenink, "Praat, a system for doing phonetics by computer," *Glot international*, vol. 5, pp. 341–345, 01 2001.

[17] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[18] Plotly Technologies Inc. (2015) Collaborative data science. Montreal, QC.

[19] K. Tokuda, T. Kobayashi, and S. Imai, "Adaptive cepstral analysis of speech," *IEEE Transactions on Speech and Audio Processing*, vol. 3, no. 6, pp. 481–489, 1995.

[20] T. Fukada, K. Tokuda, T. Kobayashi, and S. Imai, "An adaptive algorithm for mel-cepstral analysis of speech," in *1992 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1, 1992, pp. 137–140.

[21] K. Tokuda, T. Kobayashi, and S. Imai, "Generalized cepstral analysis of speech - unified approach to LPC and cepstral method," in *First International Conference on Spoken Language Processing (ICSLP)*, 1990, pp. 37–40.

[22] K. Tokuda, T. Kobayashi, T. Masuko, and S. Imai, "Mel-generalized cepstral analysis - a unified approach to speech spectral estimation," in *3rd International Conference on Spoken Language Processing (ICSLP)*, 1994, pp. 1043–1046.

[23] D. Talkin, "A robust algorithm for pitch tracking (RAPT)," in *Speech coding and synthesis*, 2005, pp. 495–518.

[24] A. Camacho and J. Harris, "A sawtooth waveform inspired pitch estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 124, pp. 1638–52, 2008.

[25] D. Talkin, "REAPER: Robust epoch and pitch estimator," https://github.com/google/REAPER, 2015.

[26] M. Morise, H. Kawahara, and H. Katayose, "Fast and reliable F0 estimation method based on the period extraction of vocal fold vibration of singing voice and speech," in *AES 35th International Conference*, 2009.

[27] B. S. Atal and S. L. Hanauer, "Speech analysis and synthesis by linear prediction of the speech wave," *The Journal of the Acoustical Society of America*, vol. 50, no. 2B, pp. 637–655, 08 1971.

[28] F. Itakura, "Line spectrum representation of linear predictor coefficients of speech signals," *Acoustical Society of America Journal*, vol. 57, no. S1, p. S35, 1975.

[29] S. Imai, K. Sumita, and C. Furuichi, "Mel log spectrum approximation (MLSA) filter for speech synthesis," *Electronics and Communications in Japan Part I-communications*, vol. 66, pp. 10–18, 1983.

[30] S. J. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, *The HTK Book Version 3.4*. Cambridge University Press, 2006.

[31] P. Kabal and R. Ramachandran, "The computation of line spectral frequencies using Chebyshev polynomials," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 6, pp. 1419–1426, 1986.

[32] S. Sagayama and F. Itakura, "Duality theory of composite sinusoidal modeling and linear prediction," in *1986 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 11, 1986, pp. 1261–1264.

[33] B. Yegnanarayana, "Speech analysis by pole-zero decomposition of short-time spectra," *Signal Processing*, vol. 3, no. 1, pp. 5–17, 1981.

[34] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. 28, no. 1, pp. 84–95, 1980.

[35] Y.-P. Lin and P. Vaidyanathan, "A Kaiser window approach for the design of prototype filters of cosine modulated filterbanks," *IEEE Signal Processing Letters*, vol. 5, no. 6, pp. 132–134, 1998.

[36] F. Cruz-Roldan, P. Amo-Lopez, S. Maldonado-Bascon, and S. Lawson, "An efficient and simple method for designing prototype filters for cosine-modulated pseudo-QMF banks," *IEEE Signal Processing Letters*, vol. 9, no. 1, pp. 29–31, 2002.

[37] T. Toda, A. W. Black, and K. Tokuda, "Voice conversion based on maximum-likelihood estimation of spectral parameter trajectory," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 8, pp. 2222–2235, 2007.

[38] J.-L. Gauvain and C.-H. Lee, "Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains," *IEEE Transactions on Speech and Audio Processing*, vol. 2, no. 2, pp. 291–298, 1994.

[39] R. Kubichek, "Mel-cepstral distance measure for objective speech quality assessment," in *Proceedings of IEEE Pacific Rim Conference on Communications Computers and Signal Processing*, vol. 1, no. 1, 1993, pp. 125–128.

[40] A. V. Oppenheim and D. H. Johnson, "Discrete representation of signals," *Proceedings of the IEEE*, vol. 60, no. 6, pp. 681–691, 1972.

[41] A. V. Oppenheim and R. W. Schafer, *Discrete-time signal processing*, 3rd ed., ser. Prentice-Hall Signal Processing Series. Pearson, 2009.