The Journal of Open Source Software

Dimensionally Aligned Signal Projection Algorithms Library

Jacob Smith [1¶], **Nathan Martindale** [1], **Mark B. Adams** [1], **Scott L. Stewart** [1], **and Phillip Bingham** [1]

1 Oak Ridge National Laboratory ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

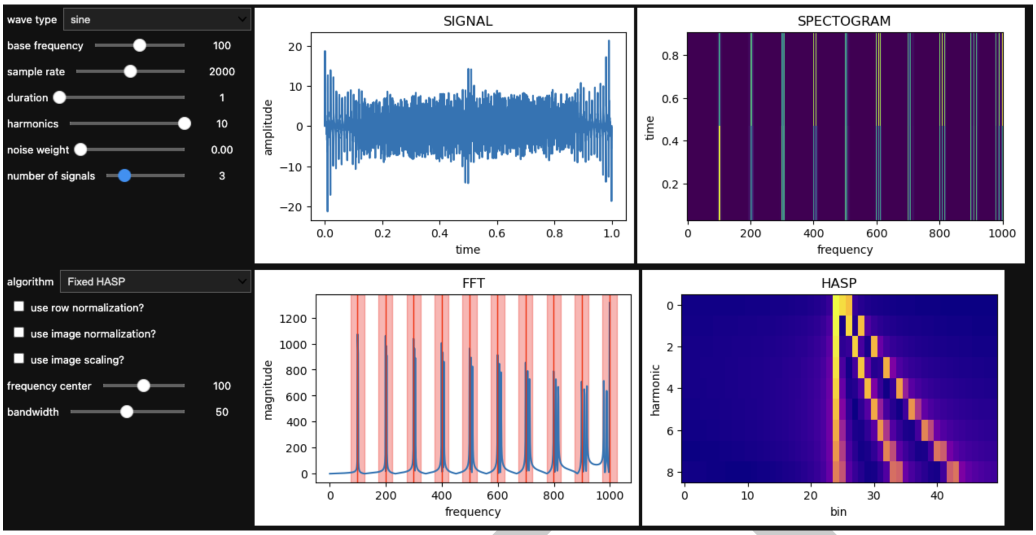
Submitted: 21 March 2024
Published: unpublished

Summary

Dimensionally aligned signal projection (DASP) algorithms are used to analyze fast Fourier transforms (FFTs) and generate visualizations that help focus on the harmonics for specific signals. At a high level, these algorithms extract the FFT segments around each harmonic frequency center, and then align them in equally sized arrays ordered by increasing distance from the base frequency. This allows for a focused view of the harmonic frequencies, which, among other use cases, can enable machine learning algorithms to more easily identify salient patterns. This work seeks to provide an effective open-source implementation of the DASP algorithms proposed in (Vann et al., 2018) as well as functionality to help explore and test how these algorithms work with an interactive dashboard and signal-generation tool.

The DASP library is implemented in Python and contains four types of algorithms for implementing these feature engineering techniques: fixed harmonically aligned signal projection (HASP), decimating HASP, interpolating HASP, and frequency aligned signal projection (FASP). Each algorithm returns a numerical array, which can be visualized as an image. The HASP algorithms are variations of the algorithms originally presented in (Vann et al., 2018). For consistency FASP, which is the terminology used for the short-time Fourier transform, has been implemented as part of the library to provide a similar interface to the STFT of the raw signal. Additionally, the library contains an algorithm to generate artificial signals with basic customizations such as the base frequency, sample rate, duration, number of harmonics, noise, and number of signals.

Finally, the library provides multiple interactive visualizations, each implemented using IPyWidgets and work in a Jupyter environment. A dashboard-style visualization is provided, which contains some common signal-processing visual components (signal, FFT, spectogram) updating in unison with the HASP functions (see Figure 1 below). Seperate from the dashboard, an independent visualization is provided for each of the DASP algorithms as well as the artifical signal generator. These visualizations are included in the library to aid in developing an intuitive understanding how the algorithms are affected by different input signals and parameter selections.



wave type: sine
base frequency: 100
sample rate: 2000
duration: 1
harmonics: 10
noise weight: 0.00
number of signals: 3

algorithm: Fixed HASP
□ use row normalization?
□ use image normalization?
□ use image scaling?
frequency center: 100
bandwidth: 50

Figure 1: Interactive dashboard displaying the custom sine wave (upper left), spectogram (upper right), FFT with added red bands around the bandwidth (bottom left), and Fixed HASP output (bottom right)

Statement of Need

Signals analysis in problem domains such as nonintrusive load monitoring often requires subject matter experts to create highly specific signal transforms for the target devices under test. The DASP algorithms were originally created as a feature-engineering technique to emphasize the harmonics present in a signal. This added emphasis has been shown to improve the performance of models that classify the type of downstream device(s) in noninstrusive load monitoring (NILM) applications (Vann et al., 2018).

Although the DASP algorithms have proven useful for signal-feature extraction, at the time of this writing few open-source libraries focus on harmonic-structure extraction and manipulation. NILMTK (Batra et al., 2014) provides an expansive library for preprocessing and analyzing NILM-specific data but has no functionality for harmonics. TorchSig (Boegner et al., 2022) is a machine learning toolkit for manipulating signals in PyTorch and has an expansive array of signal transforms and augmentations but also has nothing specific to harmonic structure. An implementation of an algorithm using average harmonic structure to extract notes of specific instruments from audio data (Duan et al., 2008) exists (cpvlordelo, 2021), but this algorithm focuses on learning harmonic models of specific instruments rather than as a general feature-engineering technique.

Algorithm Details

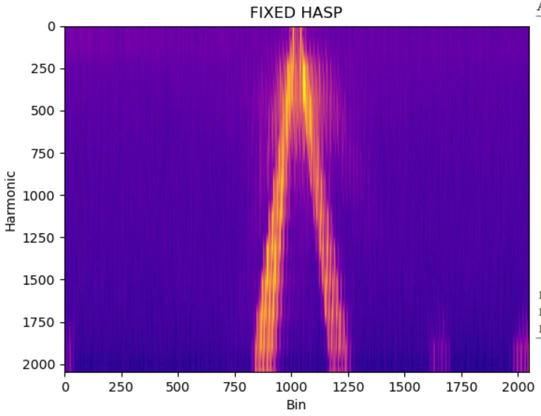
This section describes technical details of each algorithm included in the DASP library, shows an example figure of the output using a data sample which collected the electromagnetic spectrum around two fluorescent lightbulbs at a sample rate of 2 megasamples per second, and provides pseudo code for the general structure of the algorithm.

Harmonically Aligned Signal Projection

Each HASP algorithm follows the general structure of first calculating the appropriate step size to locate the index of each harmonic center, extracting the value at each index (harmonic center) along with an equal number of adjacent points from each side, and then unioning these equally sized segments before outputting them as one HASP array.

Fixed

Fixed HASP groups each harmonic center with its surrounding points using a fixed number of points. See Figure 2 below.



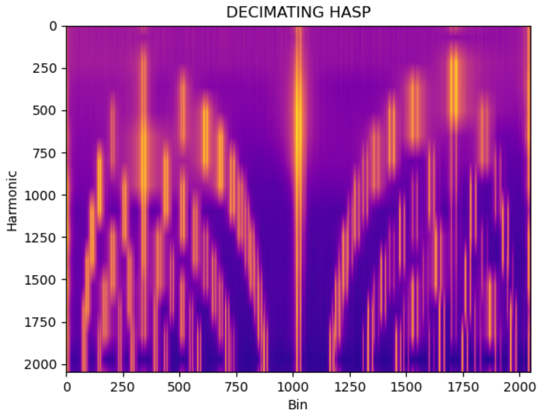
Algorithm 1 Fixed HASP

```
function FIXEDHASP($r, f_c, b, s, H_m$)
    ▷ Inputs:
        $r$ – Real Fast Fourier Transform (array)
        $f_c$ – Center Frequency (integer)
        $b$ – Bandwidth (integer)
        $s$ – Sample Rate (integer)
        $H_m$ – Max Desired Harmonics (integer)

    Let $r_s$ be the total number of points in $r$
    $f_{ci} = f_c * (r_s / (s / 2))$        ▷ rounded up to nearest integer
    $b_p = b * (r_s / (s / 2)$              ▷ rounded up to nearest integer
    Let $HASP_f$ be an empty array
    for $i = 1$ to $H_m$ do
        $b_{min} = (i * f_{ci}) - (b_p/2)$   ▷ rounded down to nearest integer
        $b_{max} = (i * f_{ci}) + (b_p/2)$   ▷ rounded down to nearest integer
        $r_{seg} = r[b_{min} \dots b_{max}]$
        append $r_{seg}$ to $HASP_f$
    end for
end function
```

Figure 2: (Left) Normalized and scaled fixed HASP image with a base frequency of 45,033 hertz. (Right) Fixed HASP pseudo code.

Decimating

Decimating HASP groups each harmonic center with its surrounding points, allowing the number of points to grow as the harmonics (and frequency) increase, and then decimates each harmonic (row) to be equal to the number of points in the row with the lowest frequency (and by extension the smallest number of points). See Figure 3 below.



Algorithm 2 Decimating HASP

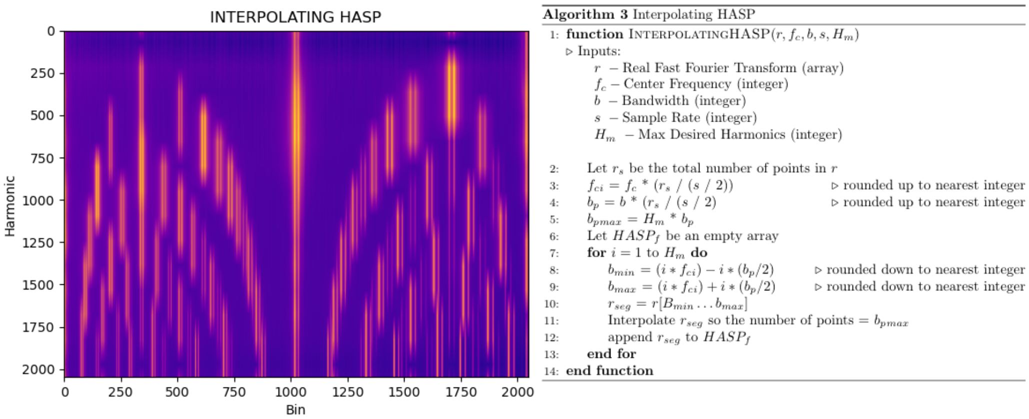
```
function DECIMATINGHASP($r, f_c, b, s, H_m$)
    ▷ Inputs:
        $r$ – Real Fast Fourier Transform (array)
        $f_c$ – Center Frequency (integer)
        $b$ – Bandwidth (integer)
        $s$ – Sample Rate (integer)
        $H_m$ – Max Desired Harmonics (integer)

    Let $r_s$ be the total number of points in $r$
    $f_{ci} = f_c * (r_s / (s / 2))$            ▷ rounded up to nearest integer
    $b_p = b * (r_s / (s / 2)$                  ▷ rounded up to nearest integer
    Let $HASP_f$ be an empty array
    for $i = 1$ to $H_m$ do
        $b_{min} = (i * f_{ci}) - i * (b_p/2)$   ▷ rounded down to nearest integer
        $b_{max} = (i * f_{ci}) + i * (b_p/2)$   ▷ rounded down to nearest integer
        $r_{seg} = r[B_{min} \dots b_{max}]$
        Decimate $r_{seg}$ so the number of points in each row = $b_p$
        append $r_{seg}$ to $HASP_f$
    end for
end function
```

Figure 3: (Left) Normalized and scaled decimating HASP image with a base frequency of 45,033 hertz. (Right) Decimating HASP pseudo code.

This version of the algorithm increases resolution at lower harmonics.

Interpolating

Interpolating HASP groups each harmonic center with its surrounding points, allowing the number of points to grow as the harmonics (and frequency) increase, and then interpolates each harmonic (row) to be equal to the number of points in the highest frequency (and by extension the largest row). See Figure 4 below.



Algorithm 3 Interpolating HASP

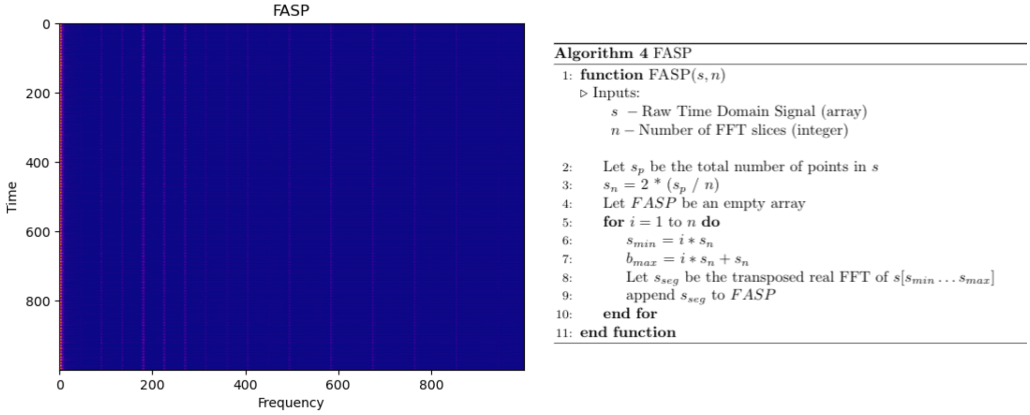
```
function INTERPOLATINGHASP($r, f_c, b, s, H_m$)
  ▷ Inputs:
      $r$ – Real Fast Fourier Transform (array)
      $f_c$ – Center Frequency (integer)
      $b$ – Bandwidth (integer)
      $s$ – Sample Rate (integer)
      $H_m$ – Max Desired Harmonics (integer)

    Let $r_s$ be the total number of points in $r$
    $f_{ci} = f_c * (r_s / (s / 2))$            ▷ rounded up to nearest integer
    $b_p = b * (r_s / (s / 2)$                  ▷ rounded up to nearest integer
    $b_{pmax} = H_m * b_p$
    Let $HASP_f$ be an empty array
    for $i = 1$ to $H_m$ do
        $b_{min} = (i * f_{ci}) - i * (b_p/2)$    ▷ rounded down to nearest integer
        $b_{max} = (i * f_{ci}) + i * (b_p/2)$    ▷ rounded down to nearest integer
        $r_{seg} = r[B_{min} \dots b_{max}]$
        Interpolate $r_{seg}$ so the number of points = $b_{pmax}$
        append $r_{seg}$ to $HASP_f$
    end for
end function
```

Figure 4: (Left) Normalized and scaled interpolating HASP image with a base frequency of 45,033 hertz. (Right) Interpolating HASP pseudo code.

This version of the algorithm increases resolution at higher harmonics.

Frequency Aligned Signal Projection

FASP simply creates a spectogram of the input signal. See Figure 5 below.



Algorithm 4 FASP

```
function FASP($s, n$)
  ▷ Inputs:
      $s$ – Raw Time Domain Signal (array)
      $n$ – Number of FFT slices (integer)

    Let $s_p$ be the total number of points in $s$
    $s_n = 2 * (s_p / n)$
    Let $FASP$ be an empty array
    for $i = 1$ to $n$ do
        $s_{min} = i * s_n$
        $b_{max} = i * s_n + s_n$
        Let $s_{seg}$ be the transposed real FFT of $s[s_{min} \dots s_{max}]$
        append $s_{seg}$ to $FASP$
    end for
end function
```

Figure 5: (Left) Normalized FASP image with normalized rows. (Right) FASP pseudo code.

Audience

The target audience for a DASP algorithms library is researchers, data scientists, and engineers in the field of signal processing who want to use machine learning as a tool for identifying patterns in signals.

This open-source software is licensed under a BSD-3 clause license, is registered on DOE Code, and is available on GitHub. The package is also pip installable with `pip install dasp-stacker`. Finally, linting for this project is performed using black (Python Software Foundation,

2022) and flake8 (Python Code Quality Authority, 2023) as well as other pre-commit hooks with pre-commit (pre-commit Organization, 2023).

Acknowledgements

The authors would like to acknowledge the US Department of Energy, National Nuclear Security Administration's Office of Defense Nuclear Nonproliferation Research and Development (NA-22) for supporting this work.

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

References

Batra, N., Kelly, J., Parson, O., Dutta, H., Knottenbelt, W., Rogers, A., Singh, A., & Srivastava, M. (2014). NILMTK: An open source toolkit for non-intrusive load monitoring. *Proceedings of the 5th International Conference on Future Energy Systems*, 265–276.

Boegner, L., Gulati, M., Vanhoy, G., Vallance, P., Comar, B., Kokalj-Filipovic, S., Lennon, C., & Miller, R. D. (2022). *Large scale radio frequency signal classification*. https://arxiv.org/abs/2207.09918

cpvlordelo. (2021). Source-separation-AHS. In *GitHub repository*. https://github.com/cpvlordelo/source-separation-AHS; GitHub.

Duan, Z., Zhang, Y., Zhang, C., & Shi, Z. (2008). Unsupervised single-channel music source separation by average harmonic structure modeling. *IEEE Transactions on Audio, Speech, and Language Processing*, *16*(4), 766–778. https://doi.org/10.1109/TASL.2008.919073

pre-commit Organization. (2023). *Pre-commit - a framework for managing and maintaining multi-language pre-commit hooks*. https://pre-commit.com/.

Python Code Quality Authority. (2023). *Flake8 - your tool for style guide enforcement*. https://flake8.pycqa.org/en/latest/.

Python Software Foundation. (2022). *Black - the uncompromising code formatter*. https://github.com/psf/black.

Vann, J. M., Karnowski, T. P., Kerekes, R., Cooke, C. D., & Anderson, A. L. (2018). A dimensionally aligned signal projection for classification of unintended radiated emissions. *IEEE Transactions on Electromagnetic Compatibility*, *60*(1), 122–131. https://doi.org/10.1109/TEMC.2017.2692962