# Contents

## I Introduction

### II Contents

<table>
<thead>
<tr>
<th>1 Installation Instructions</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Obtaining the Latest Software</td>
<td>7</td>
</tr>
<tr>
<td>1.2 Prerequisites</td>
<td>7</td>
</tr>
<tr>
<td>1.3 Building and Installation</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2 Overview of Time Encoding and Decoding</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 What is Time Encoding?</td>
<td>9</td>
</tr>
<tr>
<td>2.2 What is Time Decoding?</td>
<td>10</td>
</tr>
<tr>
<td>2.3 What are Time Encoding Machines?</td>
<td>11</td>
</tr>
<tr>
<td>2.4 What are Time Decoding Machines?</td>
<td>11</td>
</tr>
<tr>
<td>2.5 Why Should Neuroscientists Care About Time Encoding?</td>
<td>11</td>
</tr>
<tr>
<td>2.6 Why Should Engineers/Computer Scientists Care About Time Encoding?</td>
<td>12</td>
</tr>
<tr>
<td>2.7 Further Reading</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 Time Encoding and Decoding Machines</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Asynchronous Sigma-Delta Modulators</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Integrate-and-Fire Neurons</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4 Reference</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 TED Modules</td>
<td>17</td>
</tr>
<tr>
<td>4.2 Utility Modules</td>
<td>46</td>
</tr>
</tbody>
</table>

| 5 Publications | 63 |

| 6 Authors & Acknowledgments | 65 |

| 7 License | 67 |

<table>
<thead>
<tr>
<th>8 Change Log</th>
<th>69</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Release 0.7.1 - (Under Development)</td>
<td>69</td>
</tr>
<tr>
<td>8.2 Release 0.07 - (August 8, 2012)</td>
<td>69</td>
</tr>
<tr>
<td>8.3 Release 0.062 - (September 19, 2011)</td>
<td>69</td>
</tr>
<tr>
<td>8.4 Release 0.061 - (November 24, 2010)</td>
<td>69</td>
</tr>
<tr>
<td>8.5 Release 0.06 - (September 17, 2010)</td>
<td>70</td>
</tr>
<tr>
<td>8.6 Release 0.051 - (June 9, 2010)</td>
<td>70</td>
</tr>
<tr>
<td>Release</td>
<td>Date</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>8.7 0.05</td>
<td>June 1, 2010</td>
</tr>
<tr>
<td>8.8 0.04</td>
<td>April 19, 2010</td>
</tr>
<tr>
<td>8.9 0.03</td>
<td>November 12, 2009</td>
</tr>
<tr>
<td>8.10 0.021</td>
<td>October 23, 2009</td>
</tr>
<tr>
<td>8.11 0.02</td>
<td>June 26, 2009</td>
</tr>
<tr>
<td>8.12 0.011</td>
<td>May 08, 2009</td>
</tr>
<tr>
<td>8.13 0.01</td>
<td>May 08, 2009</td>
</tr>
</tbody>
</table>
Part I

Introduction
The Time Encoding and Decoding Toolkit (TED) is a library of Python routines, classes, and demo programs that implement time encoding and decoding algorithms. A MATLAB implementation of the toolbox is also available from the Bionet Group's code repository page.

Additional algorithms for encoding and decoding video signals implemented in Python are available in the Video Time Encoding and Decoding Toolkit.

To get started using the toolbox, go to the installation page and try running the demos. Check out the overview page for a high-level introduction to time encoding and decoding concepts.
Part II

Contents
CHAPTER 1

Installation Instructions

1.1 Obtaining the Latest Software

The latest version of the Time Encoding and Decoding Toolkit can be downloaded from the Bionet Group’s code repository page.

1.2 Prerequisites

The Python implementation of the Time Encoding and Decoding Toolkit requires that several software packages be present in order to be built and installed (older versions of these packages may work, but have not been tested):

- Cython 0.11.2 or later.
- Numpy 1.2.0 or later.
- Python 2.5 or later.
- Scipy 0.7.0 or later.

To run the CUDA-dependent implementations, you will also need:

- NVIDIA CUDA Toolkit 3.2 or later.
- PyCUDA 0.94.2 or later.
- scikit.cuda 0.04 or later.

To run the demo code and generate plots, the following package is also required:

- Matplotlib 0.98 or later.

Some of the utility functions may require the following packages:

- MEncoder 1.0 or later.
- PyTables 2.1.1 or later.
- OpenCV 2.1.0 or later.

To build the documentation, the following packages are also required:

- Docutils 0.5 or later.
- Jinja2 2.2 or later
- Pygments 0.8 or later
**1.3 Building and Installation**

To build and install the toolkit, download and unpack the source release and run:

```bash
python setup.py install
```

from within the main directory in the release. Sample code demonstrating how to use the toolkit is located in the `demos/` subdirectory. If you have `pip` installed, you can install the latest package code directly from Github as follows:

```bash
pip install git+git://github.com/bionet/ted.python.git
```

To rebuild the documentation, run:

```bash
make
```

from within the `docs/` subdirectory and follow the directions.
2.1 What is Time Encoding?

Consider a simple analog signal:

\[ u(t) \rightarrow \sum b \rightarrow \frac{1}{\tau} \int u(t) dt \rightarrow y(t) \rightarrow \delta \rightarrow (t_k)_{k \in \mathbb{Z}} \]
To illustrate the encoding process, the sample signal depicted above (with a bandwidth of 32 Hz and a duration of 100 ms) is encoded below using the Integrate-and-Fire neuron as a series of non-uniformly spaced spikes (represented by the red markers in the plot below):

The existence of such systems motivates the question as to whether signals that are encoded in such a manner can be faithfully recovered.

2.2 What is Time Decoding?

Under appropriate conditions \[1\], a signal that has been time encoded as a sequence of spikes can be faithfully recovered using a *time decoding machine*. The circuit below uses sinc kernels to reconstruct the encoded signal:

\[
(t_s)_{s \in \mathbb{Z}} \xrightarrow{c = G \cdot q} \sum_{k \in \mathbb{Z}} c_k \delta(t - s_k) \xrightarrow{\Omega} \hat{\omega}(t)
\]

Processing a time encoded signal with the above circuit results in the following reconstruction. Although the actual input signal is discretely represented with traditional computational models, the reconstruction can be made arbitrarily precise for bandlimited stimuli.
2.3 What are Time Encoding Machines?

A Time Encoding Machine, or TEM [2], represents analog information in the time domain. TEMs arise in single-input single-output (integrate-and-fire neurons, asynchronous sigma-delta modulators), single-input multi-output (population encoding models), and other classical circuit configurations:

![Diagram of Time Encoding Machine]

2.4 What are Time Decoding Machines?

A Time Decoding Machine, or TDM [2], recovers a signal encoded by a Time Encoding Machine according to a fidelity criterion:

![Diagram of Time Decoding Machine]

2.5 Why Should Neuroscientists Care About Time Encoding?

- Time Encoding Machines are *in silico* models of neural encoding. Under appropriate conditions, these encoders faithfully represent information about stimuli [2].
• The theory underlying time encoding applies to well-understood neuron models such as the (leaky) Integrate-and-Fire neuron [3], the Hodgkin-Huxley neuron, and feedback and feedforward configurations of these models [6].

• The theory of time encoding has been extended to address the representation of scalar-valued stimuli by populations of neurons [7] [8]. In particular, the time encoding and decoding of video signals has been successfully performed using ensembles of neurons [9] [13].

2.6 Why Should Engineers/Computer Scientists Care About Time Encoding?

• Signal representations in the time domain can exploit the increasing time resolution that scaling of integrated circuits technology [2] (Moore’s law) continues to offer. Traditional A/D conversion, by contrast, is adversely affected by the power density requirements that call for lower and lower voltages.

• Time encoding theory extends the application domain of well-understood concepts in information theory, communications theory, signal processing and machine learning to asynchronous information representation and computation [6].

• Time encoding theory has been extended to address the faithful representation of real-time (arbitrarily long) signals [10] and non-bandlimited signals [11].

2.7 Further Reading

See the publications page for the list of articles whose algorithms are implemented in the toolkit. More literature on time encoding and decoding is available on the Bionet publication server.
Time Encoding and Decoding Machines

The TED Toolkit provides implementations of a range of time encoding and decoding machines. Brief descriptions of these machines are provided in the following pages, along with links to the relevant articles that describe them in more detail and documentation for the Python functions and classes that implement them.

3.1 Asynchronous Sigma-Delta Modulators

3.1.1 Single-Input Single-Output Machines


\[
\begin{align*}
\sum_{k} \frac{1}{\delta} \int dt \quad y(t) \quad (t_k)_{k \in \mathbb{Z}}
\end{align*}
\]


\[
(t_k)_{k \in \mathbb{Z}} \rightarrow c = G^T q \rightarrow \sum_{k \in \mathbb{Z}} c_k \delta (t - s_k) \rightarrow \hat{u}(t)
\]


\[
(t_k)_{k \in \mathbb{Z}} \rightarrow d = \alpha T \cdot \text{SDP} \cdot q \rightarrow \frac{2}{\delta M} \sum_{s=-M}^{M} \tilde{d}s \cdot e^{j \omega s t / M} \rightarrow \hat{u}(t)
\]

**Time Decoding Machine - Real-Time Decoder** [10] Decodes a bandlimited, arbitrarily long signal encoded by an Asynchronous Sigma-Delta Modulator by stitching together blocks of data decoded by solving a Vandermonde system using the Björk-Pereyra Algorithm.

\[
(t_k)_{k \in \mathbb{Z}} \rightarrow V \cdot d = D \cdot q \rightarrow \sum_{s=-M}^{M} (u - 2\pi s N) \cdot e^{j \omega (s - 2\pi n) t / M} \quad \rightarrow \hat{u}(t)
\]

**Time Decoding Machine - Threshold-Insensitive Real-Time Decoder** [10] Decodes a bandlimited, arbitrarily long signal encoded by an Asynchronous Sigma-Delta Modulator by stitching
together blocks of data decoded by solving a Vandermonde system using the Björk-Pereyra Algorithm. This reconstruction method does not require the specification of an integrator threshold.

\[ (x_n) \rightarrow B \cdot \exp(-a_n x) \rightarrow \sum_{m=0}^{N} \left( -2m / N \right) \cdot \delta(2m / N) \rightarrow \delta(t) \]

**Time Decoding Machine - Threshold-Insensitive Method [2]** Reconstructs a bandlimited signal encoded with an Asynchronous Sigma-Delta Modulator using sinc kernels. This reconstruction method does not require the specification of an integrator threshold.

\[ (x_n) \rightarrow c = B \cdot \exp(-a_n x) \rightarrow \sum_{m=0}^{N} \delta(m / N) \rightarrow \delta(t) \]

### 3.1.2 Multi-Input Single-Output Machines

**Time Decoding Machine - MISO Decoder [7]** Decodes a bandlimited signal encoded by multiple Asynchronous Sigma-Delta Modulators using sinc kernels.

\[ (x_n) \rightarrow c = G \cdot q \rightarrow \sum_{m=0}^{N} \delta(m / N) \rightarrow \delta(t) \]

**Time Decoding Machine - Threshold-Insensitive MISO Decoder [2] [7]** Decodes a bandlimited signal encoded by multiple Asynchronous Sigma-Delta Modulators using sinc kernels. This reconstruction method does not require the specification of an integrator thresholds.

\[ (x_n) \rightarrow c = B \cdot \exp(-a_n x) \rightarrow \sum_{m=0}^{N} \delta(m / N) \rightarrow \delta(t) \]

### 3.2 Integrate-and-Fire Neurons

#### 3.2.1 Single-Input Single-Output Machines

**Time Encoding Machine [2]** Encodes a bandlimited signal using an Integrate-and-Fire neuron. Leaky and ideal neuron models are supported.
**Time Decoding Machine [1]** Decodes a bandlimited signal encoded by an Integrate-and-Fire neuron using sinc kernels.

\[
(\tau_k)_{k \in \mathbb{Z}} \quad c = G^{-1}q \quad \sum_{k \in \mathbb{Z}} c_k \delta(t - \tau_k) \quad \Omega \quad d(t)
\]


\[
(\tau_k)_{k \in \mathbb{Z}} \quad d = \alpha T S D P q \quad \sum_{m=-M}^{M} \alpha_d \, e^{i \Omega M t / M} \quad \Omega \quad d(t)
\]

**Time Decoding Machine – Real-Time Decoder [10]** Decodes a bandlimited, arbitrarily long signal encoded by an Integrate-and-Fire neuron by stitching together blocks of data decoded by solving a Vandermonde system using the Björk-Pereyra Algorithm.

\[
(\tau_k)_{k \in \mathbb{Z}} \quad \sum_{m=-M}^{M} \alpha_d \, e^{i \Omega M t / M} \quad \Omega \quad d(t)
\]


\[
(\tau_k)_{k \in \mathbb{Z}} \quad \begin{bmatrix} G & p & r \\ p & 0 & 0 \\ r & 0 & 0 \end{bmatrix} \begin{bmatrix} c \\ d_h \end{bmatrix} = \begin{bmatrix} q \\ 0 \end{bmatrix} \quad d_h + \sum_{k=1}^{n} c_k \psi_k(t) \quad d(t)
\]

### 3.2.2 Single-Input Multi-Output Machines

**Time Encoding Machine – SIMO Coupled IAF Encoder [12]** Encodes a finite energy signal encoded by with multiple coupled ON-OFF Integrate-and-Fire neurons.
3.2.3 Multi-Input Single-Output Machines


\[
\sum_{k \in \mathbb{Z}} c_k \delta(t-kT) = \sum_{k \in \mathbb{Z}} q_k \delta(t-kT) + \sum_{k \in \mathbb{Z}} \Omega_k \delta(t-kT)
\]


\[
(t^i_{k \in \mathbb{Z}}) \xrightarrow{G \rho r \theta \{d_i\} \{q\}} d_0 + d_1t + \sum_{k=1}^{N} \sum_{t=0}^{k-1} c_k \psi_k(t) \rightarrow \tilde{u}(t)
\]

3.2.4 Multi-Input Multi-Output Machines


\[
\sum_{i=1}^{M} \sum_{j=1}^{N} \mu_{ij} \delta(t-jT) \rightarrow \sum_{i=1}^{M} \sum_{j=1}^{N} b_{ij} \delta(t-jT) \rightarrow \sum_{i=1}^{M} \sum_{j=1}^{N} (t^i_{k \in \mathbb{Z}}) \rightarrow \tilde{u}(t)
\]


\[
(t^i_{k \in \mathbb{Z}}) \xrightarrow{G \rho r \theta \{d_i\} \{q\}} d_0 + d_1t + \sum_{k=1}^{N} \sum_{t=0}^{k-1} c_k \psi_k(t) \rightarrow \tilde{u}(t)
\]
4.1 TED Modules

4.1.1 Asynchronous Sigma-Delta Modulator Routines

Single-Input Single-Output Machines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asdm_decode</td>
<td>ASDM time decoding machine.</td>
</tr>
<tr>
<td>asdm_decode_fast</td>
<td>Fast ASDM time decoding machine.</td>
</tr>
<tr>
<td>asdm_decode_ins</td>
<td>Threshold-insensitive ASDM time decoding machine.</td>
</tr>
<tr>
<td>asdm_encode</td>
<td>ASDM time encoding machine.</td>
</tr>
</tbody>
</table>

*bionet.ted.asdm.asdm_decode*

*bionet.ted.asdm.asdm_decode*(*s*, *dur*, *dt*, *bw*, *b*, *d*, *k=1.0*, *sgn=-1)*

ASDM time decoding machine.

Decode a signal encoded with an Asynchronous Sigma-Delta Modulator.

**Parameters**

- **s** : ndarray of floats
  
  Encoded signal. The values represent the time between spikes (in s).

- **dur** : float
  
  Duration of signal (in s).

- **dt** : float
  
  Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

- **bw** : float
  
  Signal bandwidth (in rad/s).

- **b** : float
  
  Encoder bias.

- **d** : float
  
  Encoder threshold.

- **k** : float
Encoder integrator constant.

\texttt{sgn} : \{-1, 1\}

Sign of first spike.

\textbf{Returns} \texttt{u_rec} : ndarray of floats

Recovered signal.

\texttt{bionet.ted.asdm.asdm\_decode\_fast}

\texttt{bionet.ted.asdm.asdm\_decode\_fast} \((s, \text{dur}, dt, bw, M, b, d, k=1.0, sgn=-1)\)

Fast ASDM time decoding machine.

Decode a signal encoded by an Asynchronous Sigma-Delta Modulator using a fast recovery algorithm.

\textbf{Parameters} \texttt{s} : numpy array of floats

Encoded signal. The values represent the time between spikes (in s).

\texttt{dur} : float

Duration of signal (in s).

\texttt{dt} : float

Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

\texttt{bw} : float

Signal bandwidth (in rad/s).

\texttt{M} : int

Number of bins used by the fast algorithm.

\texttt{b} : float

Encoder bias.

\texttt{d} : float

Encoder threshold.

\texttt{k} : float

Encoder integrator constant.

\texttt{sgn} : \{-1, 1\}

Sign of first spike.

\textbf{Returns} \texttt{u_rec} : ndarray of floats

Recovered signal.

\texttt{bionet.ted.asdm.asdm\_decode\_ins}

\texttt{bionet.ted.asdm.asdm\_decode\_ins} \((s, \text{dur}, dt, bw, b, sgn=-1)\)

Threshold-insensitive ASDM time decoding machine.

Decode a signal encoded with an Asynchronous Sigma-Delta Modulator using a threshold-insensitive recovery algorithm.

\textbf{Parameters} \texttt{s} : ndarray of floats
Encoded signal. The values represent the time between spikes (in s).

**dur** : float
Duration of signal (in s).

**dt** : float
Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

**bw** : float
Signal bandwidth (in rad/s).

**b** : float
Encoder bias.

**sgn** : {-1, 1}
Sign of first spike.

**Returns** **u_rec** : ndarray of floats
Recovered signal.

```python
bionet.ted.asdm.asdm_encode(u, dt, b, d, k=1.0, dte=0.0, y=0.0, interval=0.0, sgn=1, quad_method='trapz', full_output=False)
```

ASDM time encoding machine.

Encode a finite length signal using an Asynchronous Sigma-Delta Modulator.

**Parameters**

**u** : array_like of floats
Signal to encode.

**dt** : float
Sampling resolution of input signal; the sampling frequency is 1/dt Hz.

**b** : float
Encoder bias.

**d** : float
Encoder threshold.

**k** : float
Encoder integration constant.

**dte** : float
Sampling resolution assumed by the encoder (s). This may not exceed dt.

**y** : float
Initial value of integrator.

**interval** : float
Time since last spike (in s).

**sgn** : {+1, -1}
Sign of integrator.
quad_method : {'rect', 'trapz'}

Quadrature method to use (rectangular or trapezoidal).

full_output : bool

If set, the function returns the encoded data block followed by the given parameters (with updated values for y, interval, and sgn). This is useful when the function is called repeatedly to encode a long signal.

Returns s : ndarray of floats

If full_output == False, returns the signal encoded as an array of time intervals between spikes.

s, dt, b, d, k, dte, y, interval, sgn, quad_method, full_output : tuple

If full_output == True, returns the encoded signal followed by updated encoder parameters.

Notes

When trapezoidal integration is used, the value of the integral will not be computed for the very last entry in u.

Multi-Input Single-Output Machines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asdm_decode_pop</td>
<td>Multi-input single-output ASDM time decoding machine.</td>
</tr>
<tr>
<td>asdm_decode_pop_ins</td>
<td>Threshold-insensitive multi-input single-output time decoding machine.</td>
</tr>
<tr>
<td>asdm_decode_vander</td>
<td>Asynchronous Sigma-Delta Modulator time decoding machine that uses BPA.</td>
</tr>
<tr>
<td>asdm_decode_vander_ins</td>
<td>Threshold-insensitive ASDM time decoding machine that uses BPA.</td>
</tr>
</tbody>
</table>

bionet.ted.asdm.asdm_decode_pop

bionet.ted.asdm.asdm_decode_pop(s_list, dur, dt, bw, b_list, d_list, k_list, sgn_list=[])

Multi-input single-output ASDM time decoding machine.

Decode a signal encoded by an ensemble of Asynchronous Sigma-Delta Modulators.

Parameters s_list : list of ndarrays of floats

Signal encoded by an ensemble of encoders. The values represent the time between spikes (in s). The number of arrays in the list corresponds to the number of encoders in the ensemble.

dur : float

Duration of signal (in s).

dt : float

Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

bw : float

Signal bandwidth (in rad/s).

b_list : list of floats

List of encoder biases.
**d_list** : list of floats

List of encoder thresholds.

**k_list** : list of floats

List of encoder integration constants.

**sgn_list** : list of integers {-1, 1}

List of signs of first spikes in trains.

**Returns u_rec** : ndarray of floats

Recovered signal.

**Notes**

The number of spikes contributed by each neuron may differ from the number contributed by other neurons.

---

**bionet.ted.asdm.asdm_decode_pop_ins**

```python
def bionet.ted.asdm.asdm_decode_pop_ins(s_list, dur, dt, bw, b_list, sgn_list=[]):
    # Threshold-insensitive multi-input single-output time decoding machine.
    # Decode a signal encoded by an ensemble of ASDM encoders using a threshold-insensitive recovery algorithm.

    Parameters
    **s_list** : list of ndarrays of floats
        Signal encoded by an ensemble of encoders. The values represent the time between spikes (in s). The number of arrays in the list corresponds to the number of encoders in the ensemble.

    **dur** : float
        Duration of signal (in s).

    **dt** : float
        Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

    **bw** : float
        Signal bandwidth (in rad/s).

    **b_list** : list of floats
        List of encoder biases.

    Returns
    **u_rec** : ndarray of floats
        Recovered signal.

    Notes

    The number of spikes contributed by each neuron may differ from the number contributed by other neurons.
```
bionet.ted.asdm.asdm_decode_vander

bionet.ted.asdm.asdm_decode_vander(s, dur, dt, bw, b, d, k, sgn=-1)
Asynchronous Sigma-Delta Modulator time decoding machine that uses BPA.

Decode a finite length signal encoded with an Asynchronous Sigma-Delta Modulator by efficiently solving a Vandermonde system using the Bjork-Pereyra Algorithm.

**Parameters**

- **s**: array_like of floats
  Encoded signal. The values represent the time between spikes (in s).
- **dur**: float
  Duration of signal (in s).
- **dt**: float
  Sampling resolution of original signal; the sampling frequency is 1/dt Hz.
- **bw**: float
  Signal bandwidth (in rad/s).
- **b**: float
  Encoder bias.
- **d**: float
  Encoder threshold.
- **k**: float
  Encoder integration constant.
- **sgn**: {-1, 1}
  Sign of first spike.

**Returns**

- **u_rec**: ndarray of floats
  Recovered signal.

bionet.ted.asdm.asdm_decode_vander_ins

bionet.ted.asdm.asdm_decode_vander_ins(s, dur, dt, bw, b, sgn=-1)
Threshold-insensitive ASDM time decoding machine that uses BPA.

Decode a finite length signal encoded with an Asynchronous Sigma-Delta Modulator by efficiently solving a threshold-insensitive Vandermonde system using the Bjork-Pereyra Algorithm.

**Parameters**

- **s**: array_like of floats
  Encoded signal. The values represent the time between spikes (in s).
- **dur**: float
  Duration of signal (in s).
- **dt**: float
  Sampling resolution of original signal; the sampling frequency is 1/dt Hz.
- **bw**: float
  Signal bandwidth (in rad/s).
b: float
Encoder bias.
sgn: {-1, 1}
Sign of first spike.

Returns u_rec : ndarray of floats
Recovered signal.

Supporting Routines

asdm_recoverable ASDM time encoding parameter check.

bionet.ted.asdm.asdm_recoverable

bionet.ted.asdm.asdm_recoverable(u, bw, b, d, k)
ASDM time encoding parameter check.
Determine whether a signal encoded with an Asynchronous Sigma-Delta Modulator using the specified parameters can be perfectly recovered.

Parameters u: array_like of floats
Signal to test.

bw: float
Signal bandwidth (in rad/s).

b: float
Decoder bias.

d: float
Decoder threshold.

k: float
Decoder integration constant.

Returns rec : bool
True if the specified signal is recoverable.

Raises ValueError
When the signal cannot be perfectly recovered.

Notes
The bound assumed by this check is not as strict as that described in most of Prof. Lazar’s papers.

4.1.2 Integrate-and-Fire Neuron Routines (Sinc Kernel)

Single-Input Single-Output Machines
**iaf_decode**

IAF time decoding machine.

**iaf_decode_fast**

Fast IAF time decoding machine.

**iaf_decode_spline**

Spline interpolation IAF time decoding machine.

**iaf_decode_vander**

IAF time decoding machine that uses BPA.

**iaf_encode**

IAF time encoding machine.

---

**bionet.ted.iaf.iaf_decode**

```python
bionet.ted.iaf.iaf_decode(s, dur, dt, bw, b, d, R=inf, C=1.0)
```

IAF time decoding machine.

Decode a finite length signal encoded with an Integrate-and-Fire neuron.

**Parameters**

- **s**: ndarray of floats
  
  Encoded signal. The values represent the time between spikes (in s).

- **dur**: float
  
  Duration of signal (in s).

- **dt**: float
  
  Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

- **bw**: float
  
  Signal bandwidth (in rad/s).

- **b**: float
  
  Encoder bias.

- **d**: float
  
  Encoder threshold.

- **R**: float
  
  Neuron resistance.

- **C**: float
  
  Neuron capacitance.

**Returns** **u_rec**: ndarray of floats

Recovered signal.

---

**bionet.ted.iaf.iaf_decode_fast**

```python
bionet.ted.iaf.iaf_decode_fast(s, dur, dt, bw, M, b, d, R=inf, C=1.0)
```

Fast IAF time decoding machine.

Decode a signal encoded with an Integrate-and-Fire neuron using a fast recovery algorithm.

**Parameters**

- **s**: array_like of floats
  
  Encoded signal. The values represent the time between spikes (in s).

- **dur**: float
  
  Duration of signal (in s).
dt : float
    Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

bw : float
    Signal bandwidth (in rad/s).

M : int
    Number of bins used by the fast algorithm.

b : float
    Encoder bias.

d : float
    Encoder threshold.

R : float
    Neuron resistance.

C : float
    Neuron capacitance.

Returns u_rec : ndarray of floats
    Recovered signal.

bionet.ted.iaf.iaf_decode_spline

bionet.ted.iaf.iaf_decode_spline(s, dur, dt, b, d, R=inf, C=1.0)
    Spline interpolation IAF time decoding machine.
    Decode a signal encoded with an IAF neuron using spline interpolation.

Parameters s : array_like of floats
    Encoded signal. The values represent the time between spikes (in s).

dur : float
    Duration of signal (in s).

dt : float
    Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

b : float
    Encoder bias.

d : float
    Encoder threshold.

R : float
    Neuron resistance.

C : float
    Neuron capacitance.

Returns u_rec : ndarray of floats
Recovered signal.

**bionet.ted.iaf.iaf_decode_vander**

*bionet.ted.iaf.iaf_decode_vander*(s, dur, dt, bw, b, d, R, C)

IAF time decoding machine that uses BPA.

Decode a finite length signal encoded with an Integrate-and-Fire neuron by efficiently solving a Vandermonde system using the Bjork-Pereyra Algorithm.

**Parameters s**: array_like of floats

Encoded signal. The values represent the time between spikes (in s).

**dur**: float

Duration of signal (in s).

**dt**: float

Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

**bw**: float

Signal bandwidth (in rad/s).

**b**: float

Encoder bias.

**d**: float

Encoder threshold.

**R**: float

Neuron resistance.

**C**: float

Neuron capacitance.

**Returns u_rec**: ndarray of floats

Recovered signal.

**bionet.ted.iaf.iaf_encode**

*bionet.ted.iaf.iaf_encode*(u, dt, b, d, R=inf, C=1.0, dte=0, y=0.0, interval=0.0, quad_method='trapz', full_output=False)

IAF time encoding machine.

Encode a finite length signal with an Integrate-and-Fire neuron.

**Parameters u**: array_like of floats

Signal to encode.

**dt**: float

Sampling resolution of input signal; the sampling frequency is 1/dt Hz.

**b**: float

Encoder bias.
d : float
    Encoder threshold.
R : float
    Neuron resistance.
C : float
    Neuron capacitance.
dte : float
    Sampling resolution assumed by the encoder (s). This may not exceed dt.
y : float
    Initial value of integrator.
interval : float
    Time since last spike (in s).
quad_method : {'rect', 'trapz'}
    Quadrature method to use (rectangular or trapezoidal) when the neuron is ideal; exponential Euler integration is used when the neuron is leaky.
full_output : bool
    If set, the function returns the encoded data block followed by the given parameters (with updated values for y and interval). This is useful when the function is called repeatedly to encode a long signal.

Returns s : ndarray of floats
    If full_output == False, returns the signal encoded as an array of time intervals between spikes.
    [s, dt, b, d, R, C, dte, y, interval, quad_method, full_output] : list
        If full_output == True, returns the encoded signal followed by updated encoder parameters.

Notes

When trapezoidal integration is used, the value of the integral will not be computed for the very last entry in u.

Single-Input Multi-Output Machines

iaf_encode_coupled	Single-input multi-output coupled IAF time encoding machine.

bionet.ted.iaf.iaf_encode_coupled

bionet.ted.iaf.iaf_encode_coupled(u, dt, b_list, d_list, k_list, h_list, type_list)
    Single-input multi-output coupled IAF time encoding machine.
    Encode a signal with an ensemble of coupled ideal ON-OFF Integrate-and-Fire neurons.
Parameters $u$ : array_like of floats
   Signal to encode.

$dt$ : float
   Sampling resolution of input signal; the sampling frequency is 1/dt Hz.

$b\_list$ : list of floats
   List of encoder biases.

$d\_list$ : list of floats
   List of encoder thresholds.

$k\_list$ : list of floats
   List of encoder integration constants.

$h\_list$ : M x M array_like of functions
   Coupling functions. Function $h\_list[i][j]$ describes the coupling from the integrator output of neuron $i$ to the input of neuron $j$.

$type\_list$ : list of integers {-1, 1}
   Neuron types. A value of -1 indicates that a neuron is an OFF-type neuron, while a value of 1 indicates that it is an ON-type neuron.

Returns $s\_list$ : list of ndarrays of floats
   Encoded signal.

Multi-Input Single-Output Machines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iaf_decode_coupled</code></td>
<td>Multi-input single-output coupled IAF time decoding machine.</td>
</tr>
<tr>
<td><code>iaf_decode_pop</code></td>
<td>Multi-input single-output IAF time decoding machine.</td>
</tr>
<tr>
<td><code>iaf_decode_spline_pop</code></td>
<td>Multi-input single-output spline interpolation IAF time decoding machine.</td>
</tr>
</tbody>
</table>

**bionet.ted.iaf.iaf_decode_coupled**

Decode a signal encoded with an ensemble of coupled ON-OFF Integrate-and-Fire neurons.

Parameters $s\_list$ : list of ndarrays of floats
   Signal encoded by an ensemble of coupled encoders. The values represent the time between spikes (in s). The number of arrays in the list corresponds to the number of encoders in the ensemble.

$\text{dur}$ : float
   Duration of signal (in s).

$\text{dt}$ : float
   Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

$b\_list$ : list of floats
   List of encoder biases.
\textbf{d\_list} : list of floats

List of encoder thresholds.

\textbf{k\_list} : list of floats

List of encoder integration constants.

\textbf{h\_list} : M x M array\_like of functions

Coupling functions. Function \texttt{h\_list[i][j]} describes the coupling from the integrator output of neuron \textit{i} to the input of neuron \textit{j}.

\textbf{Returns u\_rec} : ndarray of floats

Recovered signal.

\textbf{Notes}

The number of spikes contributed by each neuron may differ from the number contributed by other neurons.

\texttt{bionet.ted.iaf.iaf\_decode\_pop}

\texttt{bionet.ted.iaf.iaf\_decode\_pop (s\_list, dur, dt, bw, b\_list, d\_list, R\_list, C\_list)}

Multi-input single-output IAF time decoding machine.

Decode a signal encoded with an ensemble of Integrate-and-Fire neurons.

\textbf{Parameters s\_list} : list of ndarrays of floats

Signal encoded by an ensemble of encoders. The values represent the time between spikes (in s). The number of arrays in the list corresponds to the number of encoders in the ensemble.

\textbf{dur} : float

Duration of signal (in s).

\textbf{dt} : float

Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

\textbf{bw} : float

Signal bandwidth (in rad/s).

\textbf{b\_list} : list of floats

List of encoder biases.

\textbf{d\_list} : list of floats

List of encoder thresholds.

\textbf{R\_list} : list of floats

List of encoder neuron resistances.

\textbf{C\_list} : list of floats

List of encoder neuron capacitances.

\textbf{Returns u\_rec} : ndarray of floats
Recovered signal.

Notes

The number of spikes contributed by each neuron may differ from the number contributed by other neurons.

**bionet.ted.iaf.iaf_decode_spline_pop**

```python
bionet.ted.iaf.iaf_decode_spline_pop(s_list, dur, dt, b_list, d_list, R_list, C_list)
```

Multi-input single-output spline interpolation IAF time decoding machine.

Decode a signal encoded with an ensemble of Integrate-and-Fire neurons using spline interpolation.

- **Parameters**
  - **s_list: list of ndarrays of floats**
    - Signal encoded by an ensemble of encoders. The values represent the time between spikes (in s). The number of arrays in the list corresponds to the number of encoders in the ensemble.
  - **dur: float**
    - Duration of signal (in s).
  - **dt: float**
    - Sampling resolution of original signal; the sampling frequency is 1/dt Hz.
  - **b_list: list of floats**
    - List of encoder biases.
  - **d_list: list of floats**
    - List of encoder thresholds.
  - **R_list: list of floats**
    - List of encoder neuron resistances.
  - **C_list: list of floats**

- **Returns**
  - **u_rec : ndarray of floats**
    - Recovered signal.

Notes

The number of spikes contributed by each neuron may differ from the number contributed by other neurons.

**Multi-Input Multi-Output Machines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>iaf_decode_delay</strong></td>
<td>Multi-input multi-output delayed IAF time decoding machine.</td>
</tr>
<tr>
<td><strong>iaf_encode_delay</strong></td>
<td>Multi-input multi-output delayed IAF time encoding machine.</td>
</tr>
<tr>
<td><strong>iaf_encode_pop</strong></td>
<td>Multi-input multi-output IAF time encoding machine.</td>
</tr>
</tbody>
</table>
bionet.ted.iaf.iaf_decode_delay

bionet.ted.iaf.iaf_decode_delay(s_list, T, dt, b_list, d_list, k_list, a_list, w_list)
Multi-input multi-output delayed IAF time decoding machine.

Decode several signals encoded with an ensemble of ideal Integrate-and-Fire neurons with delays.

Parameters s_list : list of ndarrays of floats
Signals encoded by an ensemble of encoders. The values represent the time between spikes (in s). The number of arrays in the list corresponds to the number of encoders in the ensemble.

T : float
Temporal support of signals (in s).

dt : float
Sampling resolution of input signals; the sampling frequency is 1/dt Hz.

b_list : list of floats
List of encoder biases.

d_list : list of floats
List of encoder thresholds.

k_list : list of floats
List of encoder integration constants.

a_list : N x M array_like of floats.
Delays (in s).

w_list : N x M array_like of floats.
Scaling factors.

Returns u_list : list of ndarrays of floats
Decoded signals.

Notes
The specified signal length max(map(sum, s_list)) must exceed the support T over which the signal is decoded by the length of the longest delay.

bionet.ted.iaf.iaf_encode_delay

bionet.ted.iaf.iaf_encode_delay(u_list, t_start, dt, b_list, d_list, k_list, a_list, w_list, y_list=None, interval_list=None, full_output=False)
Multi-input multi-output delayed IAF time encoding machine.

Encode several signals with an ensemble of ideal Integrate-and-Fire neurons with delays.

Parameters u_list : list of ndarrays of floats
Signals to encode. Each of the ndarrays must be of the same length.

t_start : float
Time at which to begin encoding (in s).

**dt** : float
Sampling resolution of input signals; the sampling frequency is 1/dt Hz.

**b_list** : list of floats
List of encoder biases.

**d_list** : list of floats
List of encoder thresholds.

**k_list** : list of floats
List of encoder integration constants.

**a_list** : N x M array_like of floats.
Neuron delays (in s).

**w_list** : N x M array_like of floats.
Scaling factors.

**y_list** : list of floats
Initial values of integrators.

**interval_list** : list of floats
Times since last spikes (in s).

**full_output** : bool
If set, the function returns the encoded data block followed by the given parameters (with updated values for y and interval). This is useful when the function is called repeatedly to encode a long signal.

Returns **s_list** : list of ndarrays of floats
If `full_output == False`, returns the signals encoded as a list of arrays of time intervals between spikes.

**[s_list, t_start, dt, b_list, d_list, k_list, a_list, w_list, y_list, interval_list, u_list_prev, full_output]** : list
If `full_output == True`, returns the encoded signals followed by updated encoder parameters.

**Notes**

`t_start` must exceed `max(a)`.

**bionet.ted.iaf.iaf_encode_pop**

**bionet.ted.iaf.iaf_encode_pop**(`u_list`, `dt`, `b_list`, `d_list`, `R_list`, `C_list`, `dte=0`, `y=None`, `interval=None`, `quad_method='trapz'`, `full_output=False`)

Multi-input multi-output IAF time encoding machine.

Encode several signals with an ensemble of Integrate-and-Fire neurons.

**Parameters** **u_list** : list of ndarrays
Signals to encode.

**dt** : float
Sampling resolution of input signal; the sampling frequency is 1/dt Hz.

**b_list** : list of floats
List of encoder biases.

**d_list** : list of floats
List of encoder thresholds.

**R_list** : list of floats
List of encoder resistances.

**C_list** : list of floats
List of encoder capacitances.

**dte** : float
Sampling resolution assumed by the encoders. This may not exceed dt.

**y** : ndarray of floats
Initial values of integrators.

**interval** : ndarray of float
Times since last spike (in s).

**quad_method** : {'rect', 'trapz'}
Quadrature method to use (rectangular or trapezoidal) when the neuron is ideal; exponential Euler integration is used when the neuron is leaky.

**full_output** : bool
If set, the function returns the encoded data block followed by the given parameters (with updated values for y and interval). This is useful when the function is called repeatedly to encode a long signal.

**Returns**

- **s_list** : ndarray of floats
  If full_output == False, returns the signal encoded as an array of time intervals between spikes.

  - [s_list, dt, b_list, d_list, R_list, C_list, dte, y, interval, quad_method, full_output] : list
    If full_output == True, returns the encoded signal followed by updated encoder parameters.

**Notes**

When trapezoidal integration is used, the value of the integral will not be computed for the very last entry in the arrays in u_list. Using this function to encode multiple signals is faster than than repeatedly invoking iaf_encode() when the number of signals is sufficiently high.
Supporting Routines
bionet.ted.iaf.iaf_recoverable

**bionet.ted.iaf.iaf_recoverable**(*u, bw, b, d, R, C*)

IAF time encoding parameter check.

Determine whether a signal encoded with an Integrate-and-Fire neuron with the specified parameters can be perfectly recovered.

**Parameters**

- **u**: array_like of floats
  - Signal to test.
- **bw**: float
  - Signal bandwidth (in rad/s).
- **b**: float
  - Decoder bias.
- **d**: float
  - Decoder threshold.
- **R**: float
  - Neuron resistance.
- **C**: float
  - Neuron capacitance.

**Returns** **rec**: bool

- True if the specified signal is recoverable.

**Raises** **ValueError**

- When the signal cannot be perfectly recovered.

### 4.1.3 Integrate-and-Fire Neuron Routines (Dirichlet Kernel)

**Single-Input Single-Output Machines**

**iaf_decode** IAF time decoding machine using trigonometric polynomials.

bionet.ted.iaf_trig.iaf_decode


IAF time decoding machine using trigonometric polynomials.

Decode a finite length signal encoded with an Integrate-and-Fire neuron assuming that the encoded signal is representable in terms of trigonometric polynomials.

**Parameters**

- **s**: ndarray of floats
  - Encoded signal. The values represent the time between spikes (in s).
dur : float
    Duration of signal (in s).

dt : float
    Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

bw : float
    Signal bandwidth (in rad/s).

b : float
    Encoder bias.

d : float
    Encoder threshold.

R : float
    Neuron resistance.

C : float
    Neuron capacitance.

M : int
    2*M+1 coefficients are used for reconstructing the signal.

smoothing : float
    Smoothing parameter.

Returns u_rec : ndarray of floats
    Recovered signal.

Multi-Input Single-Output Machines

    __iaf_decode_pop  Multi-input single-output IAF time decoding machine.

bionet.ted.iaf_trig.iaf_decode_pop

bionet.ted.iaf_trig.iaf_decode_pop(s_list, dur, dt, bw, b_list, d_list, R_list, C_list, M=5, smoothing=0.0)
    Multi-input single-output IAF time decoding machine.
    Decode a signal encoded with an ensemble of Integrate-and-Fire neurons assuming that the encoded signal is representable in terms of trigonometric polynomials.

    Parameters s_list : list of ndarrays of floats
        Signal encoded by an ensemble of encoders. The values represent the time between spikes (in s). The number of arrays in the list corresponds to the number of encoders in the ensemble.

    dur : float
        Duration of signal (in s).

dt : float
Sampling resolution of original signal; the sampling frequency is 1/dt Hz.

**bw** : float
Signal bandwidth (in rad/s).

**b_list** : list of floats
List of encoder biases.

**d_list** : list of floats
List of encoder thresholds.

**R_list** : list of floats
List of encoder neuron resistances.

**C_list** : list of floats.
List of encoder neuron capacitances.

**M** : int
2*M+1 coefficients are used for reconstructing the signal.

**smoothing** : float
Smoothing parameter.

**Returns**
- **u_rec** : ndarray of floats
  Recovered signal.

**Notes**

The number of spikes contributed by each neuron may differ from the number contributed by other neurons.

### 4.1.4 Integrate-and-Fire Neuron CUDA Routines (Sinc Kernel)

These routines make use of CUDA.

**Single-Input Single-Output Machines**

```python
iaf_decode
iaf_decode_pop
iaf_encode
iaf_encode_pop
```

### 4.1.5 Integrate-and-Fire Neuron CUDA Routines (Dirichlet Kernel)

These routines make use of CUDA.

**Single-Input Single-Output Machines**
4.1.6 Real-Time Time Encoder and Decoder Classes

Abstract Classes

- **SignalProcessor**: Abstract signal processor.
- **RealTimeDecoder**: Abstract real-time time decoding machine.
- **RealTimeEncoder**: Abstract real-time time encoding machine.

**bionet.ted.rt.SignalProcessor**

```python
class bionet.ted.rt.SignalProcessor(*args):
    Abstract signal processor.
```

This class describes a signal processor that retrieves blocks of signal data from a source, applies some processing algorithm to it, and saves the processed blocks.

**Notes**

The `process()` method must be extended in functional subclasses of this class in order.

**Methods**

- **process(get, put)**: Process data obtained from `get()` and write it using `put()`.
- **__init__(args)**: Initialize a signal processor with the specified parameters.

**bionet.ted.rt.RealTimeDecoder**

```python
class bionet.ted.rt.RealTimeDecoder(dt, bw, N, M, K):
    Abstract real-time time decoding machine.
```

This class implements a real-time time decoding machine. It must be subclassed to use a specific decoding algorithm.

**Parameters**

- **dt**: float
Sampling resolution of input signal; the sampling frequency is $1/dt$ Hz.

$\textbf{bw} : \text{float}$

Signal bandwidth (in rad/s).

$\textbf{N} : \text{int}$

Number of spikes to process in each block less 1.

$\textbf{M} : \text{int}$

Number of spikes between the starting time of each successive block.

$\textbf{K} : \text{int}$

Number of spikes in the overlap between successive blocks.

\section*{Methods}

\begin{tabular}{|l|}
\hline
\textbf{decode(data, ...) | } Decode a block of data using the additional parameters. \\
\hline
\textbf{process(get, put) | } Process data obtained from \textit{get()} and write it using \textit{put()}. \\
\hline
\textbf{\_\_init\_ (dt, bw, N, M, K)} \\
\hline
\end{tabular}

\section*{bionet.ted.rt.RealTimeEncoder}

\begin{ttbox}
\textbf{class bionet.ted.rt.RealTimeEncoder (*args)}
\end{ttbox}

Abstract real-time time encoding machine.

This class implements a real-time time encoding machine. It must be subclassed to use a specific encoding algorithm.

\section*{Notes}

The \textit{encode()} method must be extended to contain a time encoding algorithm implementation in functional subclasses of this class.

\section*{Methods}

\begin{tabular}{|l|}
\hline
\textbf{encode(data, ...) | } Encode a block of data using the additional parameters. \\
\hline
\textbf{process(get, put) | } Process data obtained from \textit{get()} and write it using \textit{put()}. \\
\hline
\textbf{\_\_init\_ (*args)} \\
\textbf{Initialize a real-time time encoder.} \\
\hline
\end{tabular}
Methods

```python
__init__(*args) Initialize a real-time time encoder.
encode(*args) Encode a block of data.
process(get, put) Encode data returned in blocks by function get() and write it to some destination using the function put().
```

Asynchronous Sigma-Delta Modulator Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASDMRealTimeDecoder</td>
<td>Real-time ASDM time decoding machine.</td>
</tr>
<tr>
<td>ASDMRealTimeDecoderIns</td>
<td>Real-time threshold-insensitive ASDM time decoding machine.</td>
</tr>
<tr>
<td>ASDMRealTimeEncoder</td>
<td>Real-time ASDM time encoding machine.</td>
</tr>
</tbody>
</table>

```python
class bionet.ted.rt.ASDMRealTimeDecoder(dt, bw, b, d, k, N, M, K)

Real-time ASDM time decoding machine.
```

This class implements a real-time time decoding machine that decodes data encoded using an Asynchronous Sigma-Delta Modulator.

**Parameters**
- `dt`: float
  - Sampling resolution of input signal; the sampling frequency is 1/dt Hz.
- `bw`: float
  - Signal bandwidth (in rad/s).
- `b`: float
  - Encoder bias.
- `d`: float
  - Decoder threshold.
- `k`: float
  - Decoder integration constant.
- `N`: int
  - Number of spikes to process in each block less 1.
- `M`: int
  - Number of spikes between the starting time of each successive block.
- `K`: int
  - Number of spikes in the overlap between successive blocks.

Methods

```python
decode(data, ...) Decode a block of data using the additional parameters.
process(get, put) Process data obtained from get() and write it using put().
__init__(dt, bw, b, d, k, N, M, K)
```
Methods

```python
__init__(dt, bw, b, N, M, K)
```
Decode a block of data that was encoded with an ASDM encoder.

```python
decode(data)
```
Decode a block of data that was encoded with an ASDM encoder.

```python
process(get, put)
```
Decode data returned in blocks by function `get()` and write it to some destination using the function `put()`.

```python
window(t, ll, lr, rl, rr)
```
Return a window defined over the vector of times `t` that forms a partition of unity over all time.

**bionet.ted.rt.ASDMRealTimeDecoderIns**

```python
class bionet.ted.rt.ASDMRealTimeDecoderIns(dt, bw, b, N, M, K)
```
Real-time threshold-insensitive ASDM time decoding machine.

This class implements a threshold-insensitive real-time time decoding machine that decodes data encoded using an Asynchronous Sigma-Delta Modulator.

**Parameters**

- `dt`: float
  Sampling resolution of input signal; the sampling frequency is $1/dt$ Hz.

- `bw`: float
  Signal bandwidth (in rad/s).

- `b`: float
  Encoder bias.

- `N`: int
  Number of spikes to process in each block less 1.

- `M`: int
  Number of spikes between the starting time of each successive block.

- `K`: int
  Number of spikes in the overlap between successive blocks.

**Methods**

```python
decode(data, ...)  # Decode a block of data using the additional parameters.
process(get, put)  # Process data obtained from `get()` and write it using `put()`.
__init__(dt, bw, b, N, M, K)
```

---

4.1. TED Modules
bionet.ted.rt.ASDMRealTimeEncoder

class bionet.ted.rt.ASDMRealTimeEncoder(dt, b, d, k=1.0, dte=0.0, quad_method='trapz')
    Real-time ASDM time encoding machine.

    This class implements a real-time time encoding machine that uses an Asynchronous Sigma-Delta Modulator to encode data.

    Parameters
    ----------
    dt : float
        Sampling resolution of input signal; the sampling frequency is 1/dt Hz.
    b : float
        Encoder bias.
    d : float
        Encoder threshold.
    k : float
        Encoder integration constant.
    dte : float
        Sampling resolution assumed by the encoder. This may not exceed dt.
    quad_method : {'rect', 'trapz'}
        Quadrature method to use (rectangular or trapezoidal).

    Methods
    -------
    encode(data, ...)  
        Encode a block of data using the additional parameters.
    process(get, put) 
        Process data obtained from get() and write it using put().

    __init__(dt, b, d, k=1.0, dte=0.0, quad_method='trapz')

    Methods
    -------
    __init__(dt, b, d[, k, dte, quad_method])
    encode(data)  
        Encode a block of data with an ASDM encoder.
    process(get, put) 
        Encode data returned in blocks by function get() and write it to some destination.

Integrate-and-Fire Neuron Classes and Routines

bionet.ted.rt.IAFRealTimeDecoder

class bionet.ted.rt.IAFRealTimeDecoder(dt, bw, b, d, R, C, N, M, K)
    Real-time IAF neuron time decoding machine.
This class implements a real-time time decoding machine that decodes data encoded using an Integrate-and-Fire neuron.

**Parameters**

- **dt**: float
  - Sampling resolution of input signal; the sampling frequency is $1/dt$ Hz.

- **bw**: float
  - Signal bandwidth (in rad/s).

- **b**: float
  - Encoder bias.

- **d**: float
  - Decoder threshold.

- **R**: float
  - Neuron resistance.

- **C**: float
  - Neuron capacitance.

- **N**: int
  - Number of spikes to process in each block less 1.

- **M**: int
  - Number of spikes between the starting time of each successive block.

- **K**: int
  - Number of spikes in the overlap between successive blocks.

**Methods**

- **decode(data, ...)**: Decode a block of data using the additional parameters.
- **process(get, put)**: Process data obtained from `get()` and write it using `put()`.

```python
__init__(dt, bw, b, d, R, C, N, M, K)
```

**Methods**

- **decode(data)**: Decode a block of data that was encoded with an IAF neuron.
- **process(get, put)**: Decode data returned in blocks by function `get()` and write it to some destination using the function `put()`.
- **window(t, ll, lr, rl, rr)**: Return a window defined over the vector of times `t` that forms a partition of unity over all time.

```python
bionet.ted.rt.IAFRealTimeEncoder
```

**class** `bionet.ted.rt.IAFRealTimeEncoder(dt, b, d, R=inf, C=1.0, dte=0.0, quad_method='trapz')`

Real-time IAF neuron time encoding machine.

This class implements a real-time time encoding machine that uses an Integrate-and-Fire neuron to encode data.
Parameters **dt**: float

Sampling resolution of input signal; the sampling frequency is $1/dt$ Hz.

**b**: float
Encoder bias.

**d**: float
Encoder threshold.

**R**: float
Neuron resistance.

**C**: float
Neuron capacitance.

**dte**: float
Sampling resolution assumed by the encoder. This may not exceed $dt$.

**quad_method**: {'rect', 'trapz'}

Quadrature method to use (rectangular or trapezoidal).

Methods

```
encode(data, ...)  # Encode a block of data using the additional parameters.
process(get, put)  # Process data obtained from get() and write it using put().
```

**__init__**(dt, b, d[, R, C, dte, quad_method])

__init__ function.

Methods

```
encode(data)  # Encode a block of data with an IAF neuron.
process(get, put)  # Encode data returned in blocks by function get() and write it to some destination.
```

`bionet.ted.rt.iaf_decode_delay`

`bionet.ted.rt.iaf_decode_delay(s_list, T_block, T_overlap, dt, b_list, d_list, k_list, a_list, w_list)`

Real-time multi-input multi-output delayed IAF time decoding machine.
Decode several signals encoded with an ensemble of ideal Integrate-and-Fire neurons with delays.

Parameters **s**: list of ndarrays of floats
Signals encoded by an ensemble of encoders. The values represent the time between spikes (in s). The number of arrays in the list corresponds to the number of encoders in the ensemble.

**T_block**: float
Length of block to decode during each iteration (in s).

**T_overlap**: float
Length of overlap between successive blocks (in s).

dt : float
    Sampling resolution of input signals; the sampling frequency is 1/dt Hz.

b_list : list
    List of encoder biases. Must be of length \( N \).

d_list : list
    List of encoder thresholds. Must be of length \( N \).

k_list : list
    List of encoder integration constants. Must be of length \( N \).

a_list : array_like
    Array of neuron delays (in s). Must be of shape \((N, M)\).

w_list : array_like
    Array of scaling factors. Must be of shape \((N, M)\).

Returns u_list : list
    Decoded signals.

bionet.ted.rt.iaf_encode_delay

bionet.ted.rt.iaf_encode_delay(u_list, T_block, t_begin, dt, b_list, d_list, k_list, a_list, w_list)
    Real-time multi-input multi-output delayed IAF time encoding machine.
    Encode several with an ensemble of ideal Integrate-and-Fire neurons with delays.

Parameters u_list : list
    Signals to encode. Must contain \( M \) arrays of equal length.

T_block : float
    Length of block to encode (in s) during each iteration.

t_begin : float
    Time at which to begin encoding (in s).

dt : float
    Sampling resolution of input signals; the sampling frequency is 1/dt Hz.

b_list : list
    List of encoder biases. Must be of length \( M \).

d_list : list
    List of encoder thresholds. Must be of length \( M \).

k_list : list
    List of encoder integration constants. Must be of length \( M \).

a_list : array_like
    Array of neuron delays (in s). Must have shape \((N, M)\).
\textbf{w_list} : array_like
Array of scaling factors. Must have shape \((N, M)\).

Returns \textbf{s_list} : list
List of arrays of interspike intervals.

\section*{4.1.7 Routines for Solving Vandermonde Systems}

\texttt{bpa} \hspace{1em} Solve a Vandermonde system using BPA.

\texttt{bionet.ted.bpa.bpa}

\texttt{bionet.ted.bpa.bpa(V, b)}
Solve a Vandermonde system using BPA.
Solve a Vandermonde linear system using the Bjork-Pereyra algorithm.

\textbf{Parameters} \texttt{V} : ndarray of floats, shape \((M, M)\)
A Vandermonde matrix.
\texttt{b} : ndarray of floats, shape \((M,)\)
The system solved by this routine is \(\text{dot}(V, d) == b\).

\textbf{Returns} \texttt{d} : ndarray of floats, shape \((M,)\)
System solution.

See also:
\texttt{numpy.linalg.solve}

Notes

The matrix is assumed to be oriented such that its second column contains the arguments that would need to be passed to the \texttt{vander()} function in order to construct the matrix.

\section*{4.2 Utility Modules}

\subsection*{4.2.1 Animation Routines}

\begin{center}
\begin{tabular}{ll}
\texttt{animate} & Animate sequence of frames. \\
\texttt{animate2} & Animate two sequence of frames simultaneously. \\
\texttt{animate_compare} & Animate two sequence of frames and their difference simultaneously. \\
\texttt{frame_compare} & Compare corresponding frames in two video sequences. \\
\end{tabular}
\end{center}

\texttt{bionet.utils.animate.animate}

\texttt{bionet.utils.animate.animate(data, step=1, delay=0)}
Animate sequence of frames.
Animate a sequence of \( Ny \times Nx \) bitmap frames stored in a \( M \times Ny \times Nx \) data array.

**Parameters**

- **data** : numpy.ndarray
  Sequence of \( M \) 2D bitmaps stored as an array with shape \( (M, Ny, Nx) \).

- **step** : int
  Skip \( step \) frames between each displayed frames.

- **delay** : float
  Wait \( delay \) seconds between each frame refresh.

---

**bionet.utils.animate.animate2**

\[
\text{bionet.utils.animate.} \text{animate2}(data_1, data_2, step=1, delay=0)
\]

Animate two sequence of frames simultaneously.

Animate two sequences of \( Ny \times Nx \) bitmap frames stored in two \( M \times Ny \times Nx \) data arrays.

**Parameters**

- **data_1** : numpy.ndarray
  Sequence of \( M \) 2D bitmaps stored as an array with shape \( (M, Ny, Nx) \).

- **data_2** : numpy.ndarray
  Sequence of \( M \) 2D bitmaps stored as an array with shape \( (M, Ny, Nx) \).

- **step** : int
  Skip \( step \) frames between each displayed frames.

- **delay** : float
  Wait \( delay \) seconds between each frame refresh.

---

**bionet.utils.animate.animate_compare**

\[
\text{bionet.utils.animate.} \text{animate_compare}(data_1, data_2, step=1, delay=0)
\]

Animate two sequence of frames and their difference simultaneously.

Animate two sequences of \( Ny \times Nx \) bitmap frames stored in two \( M \times Ny \times Nx \) data arrays simultaneously with their difference.

**Parameters**

- **data_1** : numpy.ndarray
  Sequence of \( M \) 2D bitmaps stored as an array with shape \( (M, Ny, Nx) \).

- **data_2** : numpy.ndarray
  Sequence of \( M \) 2D bitmaps stored as an array with shape \( (M, Ny, Nx) \).

- **step** : int
  Skip \( step \) frames between each displayed frames.

- **delay** : float
  Wait \( delay \) seconds between each frame refresh.
**bionet.utils.animate.frame_compare**

*bionet.utils.animate.frame_compare*(data_1, data_2, i=0)

Compare corresponding frames in two video sequences.

Simultaneously display two corresponding frames from two video sequences of identical length.

**Parameters**

- **data_1**: numpy.ndarray
  Sequence of \( M \) 2D bitmaps stored as an array with shape \((M, Ny, Nx)\).

- **data_2**: numpy.ndarray
  Sequence of \( M \) 2D bitmaps stored as an array with shape \((M, Ny, Nx)\).

- **i**: int
  Index of frame to display.

---

**4.2.2 Band-Limited Function Routines**

---

**gen_band_limited** Generate a uniformly sampled, band-limited signal.

*bionet.utils.band_limited.gen_band_limited*

*bionet.utils.band_limited.gen_band_limited*(dur, dt, fmax, np=None, nc=3)

Generate a uniformly sampled, band-limited signal.

**Parameters**

- **dur**: float
  Duration of signal (s).

- **dt**: float
  Sampling resolution; the sampling frequency is \( 1/dt \) Hz.

- **fmax**: float
  Maximum frequency (Hz).

- **np**: float
  Noise power. If \( np \neq None \), Gaussian white noise is added to the generated signal before the latter is filtered.

- **nc**: int
  Number of discrete frequency components in generated signal.

**Returns**

- **u**: ndarray of floats
  Generated signal.

---

**4.2.3 Miscellaneous Routines and Classes**

---

**chunks** Chunk generator.

**func_timer** Time the execution of a function.

**SerialBuffer** Serial buffer class.
bionet.utils.misc.chunks

bionet.utils.misc.chunks(seq, n)

Chunk generator.

Return a generator whose next() method returns length n subsequences of the given sequence. If len(seq) % n != 0, the last subsequence returned will contain fewer than n entries.

Parameters seq : iterable
    Sequence to split into chunks.

    n : int
        Chunk size.

Returns g : generator
    Generator that will return length n chunks of seq.

bionet.utils.misc.func_timer

bionet.utils.misc.func_timer(f)

Time the execution of a function.

Parameters f : function
    Function to time.

class bionet.utils.misc.SerialBuffer

class bionet.utils.misc.SerialBuffer(get, n=1)

Serial buffer class.

This class implements a buffer that automatically replenishes its contents from a specified serial data source when it contains too little data to honor a read request.

Parameters get : function
    Data retrieval function. Must return an empty sequence or None when it can no longer retrieve any data.

    n : int
        Number of initial entries to load into buffer.

Methods

clear()
    Empty buffer.

read(n=1)
    Read n elements from buffer.

replenish(n=1)
    Replenish buffer to contain at least n elements.

__init__(get, n=1)
Table 4.29 – continued from previous page

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(get[, n])</td>
<td>Remove all elements from the buffer.</td>
</tr>
<tr>
<td>clear()</td>
<td>Read a block of data (default length = 1).</td>
</tr>
<tr>
<td>replenish([n])</td>
<td>Attempt to replenish the buffer such that it contains at least n entries (but do not throw any exceptions).</td>
</tr>
</tbody>
</table>

### 4.2.4 Extra Numpy Routines

- **crand**: Complex random values in a given shape.
- **hilb**: Construct a Hilbert matrix.
- **iceil**: Return the ceiling of the input, element-wise.
- **ifloor**: Return the floor of the input, element-wise.
- **iround**: Round an array to the nearest integer.
- **mdot**: Dot product of several arrays.
- **mpower**: Matrix power function.
- **rank**: Compute matrix rank.

---

**bionet.utils.numpy_extras.crand**

```python
bionet.utils.numpy_extras.crand(*args)
```

Complex random values in a given shape.

Create an array of the given shape whose entries are complex numbers with real and imaginary parts sampled from a uniform distribution over $[0, 1)$.

**Parameters**

- `d0, d1, ..., dn` : int
  - Shape of the output.

**Returns**

- `out` : numpy.ndarray
  - Complex random variables.

---

**bionet.utils.numpy_extras.hilb**

```python
bionet.utils.numpy_extras.hilb(n)
```

Construct a Hilbert matrix.

**Parameters**

- `n` : int
  - Number of rows and columns in matrix.

**Returns**

- `h` : numpy.ndarray
  - Generated Hilbert matrix of shape $(n, n)$.

---

**bionet.utils.numpy_extras.iceil**

```python
bionet.utils.numpy_extras.iceil(x)
```
Return the ceiling of the input, element-wise.

The ceil of the scalar \( x \) is the smallest integer \( i \), such that \( i \geq x \). It is often denoted as \( \lceil x \rceil \).

**Parameters**

- \( x \) : array_like
  - Input data.

**Returns**

- \( y \) : {numpy.ndarray, scalar}
  - The ceiling of each element in \( x \), with int dtype.

**bionet.utils.numpy_extras.ifloor**

Return the floor of the input, element-wise.

The floor of the scalar \( x \) is the largest integer \( i \), such that \( i \leq x \). It is often denoted as \( \lfloor x \rfloor \).

**Parameters**

- \( x \) : array_like
  - Input data.

**Returns**

- \( y \) : {numpy.ndarray, scalar}
  - The floor of each element in \( x \), with int dtype.

**bionet.utils.numpy_extras.iround**

Round an array to the nearest integer.

**Parameters**

- \( x \) : array_like
  - Input data.

**Returns**

- \( y \) : {numpy.ndarray, scalar}
  - The rounded elements in \( x \), with int dtype.

**bionet.utils.numpy_extras.mdot**

Dot product of several arrays.

Compute the dot product of several arrays in the order they are listed.

**bionet.utils.numpy_extras.mpower**

Matrix power function.

Compute \( x \) raised to the power \( y \) where \( x \) is a square matrix and \( y \) is a scalar.

4.2. Utility Modules 51
Notes

The matrix $x$ must be non-defective.

*bionet.utils.numpy_extras.rank*

*bionet.utils.numpy_extras.rank*(x, *args)

Compute matrix rank.

Estimate the number of linearly independent rows or columns of the matrix $x$.

**Parameters**

- **x**: array_like, shape $(M, N)$
  
  Matrix to analyze.

- **tol**: float
  
  Tolerance; the default is $\text{max(svd}(x)[1]) * \text{max}(\text{shape}(x)) * 1e-13$

**Returns**

- **r**: int
  
  Estimated rank of matrix.

4.2.5 Plotting Utilities

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contour</td>
<td>Create a 3D contour plot.</td>
</tr>
<tr>
<td>contourf</td>
<td>Create a 3D contourf plot.</td>
</tr>
<tr>
<td>plot_compare</td>
<td>Compare two signals and plot the difference between them.</td>
</tr>
<tr>
<td>plot_encoded</td>
<td>Plot a time-encoded signal.</td>
</tr>
<tr>
<td>plot_fourier</td>
<td>Plot the Discrete Fourier Transform of a signal.</td>
</tr>
<tr>
<td>plot_raster</td>
<td>Plot several time sequences as a raster.</td>
</tr>
<tr>
<td>plot_signal</td>
<td>Plot a signal.</td>
</tr>
<tr>
<td>surf</td>
<td>Create a surface plot.</td>
</tr>
<tr>
<td>wireframe</td>
<td>Plot a 3D wireframe.</td>
</tr>
</tbody>
</table>

*bionet.utils.plotting.contour*

*bionet.utils.plotting.contour*(X, Y, Z, *args)

Create a 3D contour plot.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y, Z</td>
<td>Data values as numpy.arrays</td>
</tr>
<tr>
<td>extend3d</td>
<td>Whether to extend contour in 3D (default: False)</td>
</tr>
<tr>
<td>stride</td>
<td>Stride (step size) for extending contour</td>
</tr>
<tr>
<td>zdir</td>
<td>The direction to use: x, y or z (default)</td>
</tr>
<tr>
<td>offset</td>
<td>If specified plot a projection of the contour lines on this position in plane normal to zdir</td>
</tr>
</tbody>
</table>

The positional and other keyword arguments are passed on to *contour*()

Returns a *contour*
bionet.utils.plotting.contourf

*bionet.utils.plotting.contourf(*\(X, Y, Z, *args*)*

Create a 3D contourf plot.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X, Y, Z)</td>
<td>Data values as <em>numpy.arrays</em></td>
</tr>
<tr>
<td>(zdir)</td>
<td>The direction to use: x, y or z (default)</td>
</tr>
<tr>
<td>(offset)</td>
<td>If specified plot a projection of the filled contour on this position in plane normal to (zdir)</td>
</tr>
</tbody>
</table>

The positional and keyword arguments are passed on to `contourf()`

Returns a `contourf`

Changed in version 1.1.0: The \(zdir\) and \(offset\) kwargs were added.

bionet.utils.plotting.plot_compare

*bionet.utils.plotting.plot_compare(*\(t, u, v, fig_title='', file_name='\)*)*

Compare two signals and plot the difference between them.

**Parameters**

\(t\): *ndarray of floats*

Times (s) at which the signal is defined.

\(u, v\): *ndarrays of floats*

Signal samples.

\(fig\_title\): *string*

Plot title.

\(file\_name\): *string*

File in which to save the plot.

bionet.utils.plotting.plot_encoded

*bionet.utils.plotting.plot_encoded(*\(t, u, s, fig\_title='', file\_name='\)*)*

Plot a time-encoded signal.

**Parameters**

\(t\): *ndarray of floats*

Times (in s) at which the original signal was sampled.

\(u\): *ndarray of floats*

Signal samples.

\(s\): *ndarray of floats*

Intervals between encoded signal spikes.

\(fig\_title\): *string*

Plot title.

\(file\_name\): *string*

File in which to save the plot.
Notes

The spike times (i.e., the cumulative sum of the interspike intervals) must all occur within the interval $t_{\text{min}}(t)$.

bionet.utils.plotting.plot_fourier

bionet.utils.plotting.plot_fourier($u$, $fs$, $fmin=0.0$, $fmax=None$, $style='line'$)

Plot the Discrete Fourier Transform of a signal.

Parameters

- **$u$** : ndarray of floats
  - Sampled signal.
- **$fs$** : float
  - Sampling rate (Hz).
- **$fmin$** : float
  - Minimum frequency to display (Hz).
- **$fmax$** : float:
  - Maximum frequency to display (Hz).
- **style** : {'line', 'semilogy', 'stem'}
  - Set plot style.

Notes

This function may take a long time to run if the frequency range is very large.

bionet.utils.plotting.plot_raster

bionet.utils.plotting.plot_raster($ts\text{ list}$, $plot\text{ stems}=True$, $plot\text{ axes}=True$, $marker='.'$, $markersize=5$, $fig\text{ title}='$', $file\text{ name}=''$

Plot several time sequences as a raster.

Parameters

- **$ts\text{ list}$** : list of ndarrays
  - Time sequences to plot.
- **plot\text{ stems}** : bool
  - Show stems for all events.
- **plot\text{ axes}** : bool
  - Show horizontal axes for all sequences.
- **marker** : char
  - Marker symbol.
- **markersize** : int
  - Marker symbol size.
- **fig\text{ title}** : string
Plot title.

```
file_name : string
```

File in which to save the plot.

**bionet.utils.plotting.plot_signal**

```
bionet.utils.plotting.plot_signal(t, u, fig_title='', file_name='')
```

Plot a signal.

**Parameters**
- `t`: ndarray of floats
  Times (in s) at which the signal is defined.
- `u`: ndarray of floats
  Signal samples.
- `fig_title`: string
  Plot title.
- `file_name`: string
  File in which to save the plot.

**bionet.utils.plotting.surf**

```
bionet.utils.plotting.surf(X, Y, Z, *args)
```

Create a surface plot.

By default it will be colored in shades of a solid color, but it also supports color mapping by supplying the `cmap` argument.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>X</code>, <code>Y</code>, <code>Z</code></td>
<td>Data values as 2D arrays</td>
</tr>
<tr>
<td><code>rstride</code></td>
<td>Array row stride (step size)</td>
</tr>
<tr>
<td><code>cstride</code></td>
<td>Array column stride (step size)</td>
</tr>
<tr>
<td><code>color</code></td>
<td>Color of the surface patches</td>
</tr>
<tr>
<td><code>cmap</code></td>
<td>A colormap for the surface patches.</td>
</tr>
<tr>
<td><code>facecolors</code></td>
<td>Face colors for the individual patches</td>
</tr>
<tr>
<td><code>norm</code></td>
<td>An instance of Normalize to map values to colors</td>
</tr>
<tr>
<td><code>vmin</code></td>
<td>Minimum value to map</td>
</tr>
<tr>
<td><code>vmax</code></td>
<td>Maximum value to map</td>
</tr>
<tr>
<td><code>shade</code></td>
<td>Whether to shade the facecolors</td>
</tr>
</tbody>
</table>

Other arguments are passed on to `Poly3DCollection`

**bionet.utils.plotting.wireframe**

```
bionet.utils.plotting.wireframe(X, Y, Z, *args)
```

Plot a 3D wireframe.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>X</code>, <code>Y</code>, <code>Z</code></td>
<td>Data values as 2D arrays</td>
</tr>
<tr>
<td><code>rstride</code></td>
<td>Array row stride (step size)</td>
</tr>
<tr>
<td><code>cstride</code></td>
<td>Array column stride (step size)</td>
</tr>
</tbody>
</table>
Keyword arguments are passed on to LineCollection.
Returns a Line3DCollection

4.2.6 Extra Pylab Routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>eps</strong></td>
<td>Compute the spacing of floating point numbers.</td>
</tr>
<tr>
<td><strong>minmax</strong></td>
<td>Return the range of the given array.</td>
</tr>
<tr>
<td><strong>realmax</strong></td>
<td>Return the largest positive floating point number representable with the specified precision on this computer.</td>
</tr>
<tr>
<td><strong>realmin</strong></td>
<td>Return the smallest positive floating point number representable with the specified precision on this computer.</td>
</tr>
</tbody>
</table>

bionet.utils.pylab_extras.eps

bionet.utils.pylab_extras.eps(x)

Compute the spacing of floating point numbers.

bionet.utils.pylab_extras.minmax

bionet.utils.pylab_extras.minmax(x)

Return the range of the given array. If the array has 2 dimensions, return an array containing the minima and maxima of each of the rows.

bionet.utils.pylab_extras.realmax

bionet.utils.pylab_extras.realmax(t=<type 'numpy.float64'>)

Return the largest positive floating point number representable with the specified precision on this computer. Double precision is assumed if no floating point type is specified.

bionet.utils.pylab_extras.realmin

bionet.utils.pylab_extras.realmin(t=<type 'numpy.float64'>)

Return the smallest positive floating point number representable with the specified precision on this computer. Double precision is assumed if no floating point type is specified.

4.2.7 Extra Scipy Routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ci(z)</strong></td>
<td>Cosine integral of a complex value.</td>
</tr>
<tr>
<td><strong>chi(z)</strong></td>
<td>Hyperbolic cosine integral of a complex value.</td>
</tr>
<tr>
<td><strong>ei(z)</strong></td>
<td>Exponential integral of a complex value.</td>
</tr>
<tr>
<td><strong>si(z)</strong></td>
<td>Sine integral of a complex value.</td>
</tr>
<tr>
<td><strong>shi(z)</strong></td>
<td>Hyperbolic sine integral of a complex value.</td>
</tr>
</tbody>
</table>

bionet.utils.scipy_extras.ci

bionet.utils.scipy_extras.ci(z)

Cosine integral of a complex value.
**4.2.8 Extra Signal Processing Routines**

**Error Analysis Routines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db</td>
<td>Convert the specified power value to decibels assuming a reference value of 1.</td>
</tr>
<tr>
<td>rms</td>
<td>Compute the root mean squared value of the specified array x.</td>
</tr>
<tr>
<td>snr</td>
<td>Compute the signal-to-noise ratio (in dB) of a signal given its reconstruction.</td>
</tr>
</tbody>
</table>

**4.2. Utility Modules**

bionet.utils.signal_extras.db

bionet.utils.signal_extras.db(x)

Convert the specified power value to decibels assuming a reference value of 1.

bionet.utils.signal_extras.rms

bionet.utils.signal_extras.rms(x)

Compute the root mean squared value of the specified array x.

bionet.utils.signal_extras.snr

bionet.utils.signal_extras.snr(u, u_rec, k_min=0, k_max=None)

Compute the signal-to-noise ratio (in dB) of a signal given its reconstruction.

**Parameters**

- **u**: numpy array of floats
  - Original signal.
- **u_rec**: numpy array of floats
Reconstructed signal.

\[ \text{k\_min} : \text{int} \]
Lower index into the signal over which to compute the SNR.

\[ \text{k\_max} : \text{int} \]
Upper index into the signal over which to compute the SNR.

**Filtering Routines**

- **downsample**
  Downsample a vector \( x \) by returning every \( n \)th entry. An optional offset may be specified.

- **fftfilt**
  Filter the signal \( x \) with the FIR filter described by the coefficients in \( b \) using the overlap-add method. If the FFT length \( n \) is not specified, it and the overlap-add block length are selected so as to minimize the computational cost of the filtering operation.

- **remezord**
  Calculate the parameters required by the Remez exchange algorithm to construct a finite impulse response (FIR) filter that approximately meets the specified design.

  **Parameters**
  - freqs : array_like of floats
    A monotonic sequence of band edges specified in Hertz. All elements must be non-negative and less than 1/2 the sampling frequency as given by the Hz parameter.
  - amps : array_like of floats
    A sequence containing the amplitudes of the signal to be filtered over the various bands.
  - rips : array_like of floats
    A sequence specifying the maximum ripples of each band.
  - alg : \{'herrmann', 'kaiser', 'ichige'\}
    Filter length approximation algorithm.

  **Returns**
  - numtaps : int
    Desired number of filter taps.
**bands**: ndarray of floats
A monotonic sequence containing the band edges.

**amps**: ndarray of floats
Desired gain for each band region.

**weights**: ndarray of floats
Filter design weights.

See also:
scipy.signal.remez

bionet.utils.signal_extras.upsample

bionet.utils.signal_extras.upsample(x, n, offset=0)
Upsample a vector x by inserting n-1 zeros between every entry. An optional offset may be specified.

**Miscellaneous Routines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextpow2</td>
<td>Return the first integer N such that $2^N \geq \text{abs}(x)$</td>
</tr>
<tr>
<td>oddceil</td>
<td>Return the smallest odd integer no less than x.</td>
</tr>
<tr>
<td>oddround</td>
<td>Return the nearest odd integer from x.</td>
</tr>
</tbody>
</table>

bionet.utils.signal_extras.nextpow2

bionet.utils.signal_extras.nextpow2(x)
Return the first integer N such that $2^N \geq \text{abs}(x)$

bionet.utils.signal_extras.oddceil

bionet.utils.signal_extras.oddceil(x)
Return the smallest odd integer no less than x.

bionet.utils.signal_extras.oddround

bionet.utils.signal_extras.oddround(x)
Return the nearest odd integer from x.

**4.2.9 Trigonometric Polynomial Routines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>em</td>
<td>Trigonometric polynomial basis function.</td>
</tr>
<tr>
<td>gen_dirichlet_coeffs</td>
<td>Generate random Dirichlet coefficients for a real signal.</td>
</tr>
<tr>
<td>gen_trig_poly</td>
<td>Construct a trigonometric polynomial with specified Dirichlet coefficients.</td>
</tr>
<tr>
<td>gen_trig_poly_2d</td>
<td>Construct a 2D trigonometric polynomial.</td>
</tr>
<tr>
<td>get_dirichlet_coeffs</td>
<td>Compute the Dirichlet coefficients of a trigonometric polynomial.</td>
</tr>
<tr>
<td>scale_down_coeffs</td>
<td>Scale down Dirichlet coefficients.</td>
</tr>
</tbody>
</table>
**bionet.utils.trig_poly.em**

*bionet.utils.trig_poly.em*(\(m, t, Omega, M\))  
Trigonometric polynomial basis function.

**Parameters**
- **\(m\)** : int  
  Order of basis function.
- **t** : numpy.ndarray  
  Times over which to compute the function.
- **Omega** : float  
  Bandwidth (in rad/s).
- **M** : int  
  Trigonometric polynomial order.

**Returns**
- **u** : numpy.ndarray  
  Basis function \(\exp(1j*m*Omega*t/M)/sqrt(T)\), where \(T == 2*pi*M/Omega\).

**bionet.utils.trig_poly.gen_dirichlet_coeffs**

*bionet.utils.trig_poly.gen_dirichlet_coeffs*(\(M\))  
Generate random Dirichlet coefficients for a real signal.

**Parameters**
- **M** : int  
  Trigonometric polynomial order.

**Returns**
- **am** : numpy.ndarray  
  Array of Dirichlet coefficients with shape \((2*M+1)\). The coefficients are ordered such that \(am[0]\) contains the coefficient for \(m == -M\).

**bionet.utils.trig_poly.gen_trig_poly**

*bionet.utils.trig_poly.gen_trig_poly*(\(T, dt, am, method='fft', scale_down=False\))  
Construct a trigonometric polynomial with specified Dirichlet coefficients.

**Parameters**
- **T** : float  
  Period (i.e., duration) of the trigonometric polynomial.
- **dt** : float  
  Time resolution.
- **am** : int or numpy.ndarray  
  Trigonometric polynomial order or array of Dirichlet coefficients. If the latter, the length of the array must be odd.
- **method** : {'fft', 'inner'}  
  Method to use when computing coefficients. The FFT method is generally faster than using inner products.
- **scale_down** : bool
If true, linearly scale down all coefficients such that the magnitude of the high-frequency coefficients are reduced the most.

**Returns**  
\(u\) : numpy.ndarray  
Generated signal.

### bionet.utils.trig_poly.gen_trig_poly_2d

**bionet.utils.trig_poly.gen_trig_poly_2d(Sx, Sy, dx, dy, c)**  
Construct a 2D trigonometric polynomial.

**Parameters**  
\(Sx\) : float  
Period of signal along the X-axis.  
\(Sy\) : float  
Period of signal along the Y-axis.  
\(dx\) : float  
Resolution along the X-axis.  
\(dy\) : float  
Resolution along the Y-axis.  
\(c\) : tuple or numpy.ndarray  
X-axis and Y-axis trigonometric polynomial orders or an array of Dirichlet coefficients with shape \((2^My+1, 2^Mx+1)\).

**Returns**  
\(S\) : numpy.ndarray  
Generated signal.

**Notes**

This function uses the FFT to generate the output signal.

See [http://fourier.eng.hmc.edu/e101/lectures/Image_Processing/node6.html](http://fourier.eng.hmc.edu/e101/lectures/Image_Processing/node6.html) for an example of a DFT of a real 2D discrete signal.

### bionet.utils.trig_poly.get_dirichlet_coeffs

**bionet.utils.trig_poly.get_dirichlet_coeffs(u, dt, M, method=’fft’)**  
Compute the Dirichlet coefficients of a trigonometric polynomial.

**Parameters**  
\(u\) : numpy.ndarray  
Input signal.  
\(dt\) : float  
Time resolution (s).  
\(M\) : int  
Trigonometric polynomial order.

**method** : {‘fft’, ‘inner’}
Method to use when computing coefficients. The FFT method is generally faster than using inner products.

Returns am : numpy.ndarray

Array of $2^M+1$ Dirichlet coefficients.

Notes

Assumes that $u$ is defined over times $dt\cdot\text{arange}(0, \text{len}(u))$ and that $dt\cdot\text{len}(u)$ is equal to the period of the trigonometric polynomial.

bionet.utils.trig_poly.scale_down_coeffs

bionet.utils.trig_poly.scale_down_coeffs(am)

Scale down Dirichlet coefficients.

Parameters am : numpy.ndarray

Array of Dirichlet coefficients of a real trigonometric polynomial. Must be of odd length.

Returns am_new : numpy.ndarray

Array of scaled coefficients; the highest frequency coefficients are scaled down the most.

4.2.10 Video I/O Classes and Routines

ReadVideo
WriteFigureVideo
WriteVideo
video_capture
The Time Encoding and Decoding Toolkit implements algorithms from the following publications:


Authors & Acknowledgments

This software was written and packaged by Lev Givon, currently at the Bionet Group ¹ at Columbia University.

Some parts of the code are based (in part) on earlier implementations written by the following alumni of the Bionet Group under the supervision of Prof. Aurel A. Lazar ²:

- Eftychios A. Pnevmatikakis
- Yevgeniy B. Slutskiy
- Robert J. Turetsky

Special thanks are due to the following Bionet Group members and alumni for reviewing the code and providing useful suggestions:

- Robert J. Turetsky
- Yiyin Zhou

¹ http://www.bionet.ee.columbia.edu/
² http://www.ee.columbia.edu/~aurel/
License

Copyright (c) 2009-2014, Lev Givon. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of Lev Givon nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
8.1 Release 0.7.1 - (Under Development)

- Add pip installation information.
- Replace distribute_setup with ez_setup.
- Change name to Time Encoding and Decoding Toolkit.
- Fix various demos.
- Change to PEP 440 version numbering.
- Use ReadTheDocs theme in HTML docs.

8.2 Release 0.07 - (August 8, 2012)

- Merge with utils toolkit.
- Add docs for merged utilities.
- Improve finding of includes/libraries needed to build bpa extension.
- Download distribute for package installation if it isn’t available.

8.3 Release 0.062 - (September 19, 2011)

- Improve precision of IAF encoders.
- Fix signs in delayed IAF example.
- Add decoders that use the trigonometric polynomial approximation.
- Add more PyCUDA decoder implementations.

8.4 Release 0.061 - (November 24, 2010)

- Add docs for PyCUDA-based functions.
- Update installation information.
• Update copyright information.

8.5 Release 0.06 - (September 17, 2010)

• Add SISO PyCUDA-based IAF TEM/TDM implementations.
  • Fix encoder resampling.
  • Update docs.

8.6 Release 0.051 - (June 9, 2010)

• Eliminate various unnecessary loops and summations.

8.7 Release 0.05 - (June 1, 2010)

• Add documentation.
  • Add more efficient concurrent IAF encoder.
  • Fix bug in MIMO spline interpolation decoders.

8.8 Release 0.04 - (April 19, 2010)

• Improve docstrings, use numpy docstring conventions.
  • Add IAF interpolation spline decoding algorithms.
  • Add coupled IAF encoding and decoding algorithms.
  • Add MIMO delayed IAF encoding and decoding algorithms.

8.9 Release 0.03 - (November 12, 2009)

• Accelerate IAF decoder implementations.
  • Add IAF population decoding algorithm.
  • Add IAF real-time decoding algorithm.
  • Clean up demos.
  • Facilitate installation on x86_64 Linux.

8.10 Release 0.021 - (October 23, 2009)

• Clean up real-time algorithm implementations.
  • Make time array construction consistently use a fixed time step.
8.11 Release 0.02 - (June 26, 2009)

- Add real-time encoding and decoding algorithms.
- Add ASDM population decoding algorithms.
- Make code more readable.

8.12 Release 0.011 - (May 08, 2009)

- Change package name to Time Encoding and Decoding Toolbox.

8.13 Release 0.01 - (May 08, 2009)

- First public release.
Part III

Index
- genindex
- search
### Symbols

<table>
<thead>
<tr>
<th>Method</th>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>() (bionet.ted.rt.ASDMRealTimeDecoder method)</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td><strong>init</strong>() (bionet.ted.rt.ASDMRealTimeDecoderIns method)</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td><strong>init</strong>() (bionet.ted.rt.ASDMRealTimeEncoder method)</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td><strong>init</strong>() (bionet.ted.rt.IAFRealTimeDecoder method)</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td><strong>init</strong>() (bionet.ted.rt.IAFRealTimeEncoder method)</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td><strong>init</strong>() (bionet.ted.rt.RealTimeDecoder method)</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td><strong>init</strong>() (bionet.ted.rt.RealTimeEncoder method)</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td><strong>init</strong>() (bionet.ted.rt.SignalProcessor method)</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td><strong>init</strong>() (bionet.utils.misc.SerialBuffer method)</td>
<td>49</td>
<td></td>
</tr>
</tbody>
</table>

---

**A**

- animate() (in module bionet.utils.animate), 46
- animate2() (in module bionet.utils.animate), 47
- animate_compare() (in module bionet.utils.animate), 47
- asdm_decode() (in module bionet.ted.asdm), 17
- asdm_decode_fast() (in module bionet.ted.asdm), 18
- asdm_decode_ins() (in module bionet.ted.asdm), 18
- asdm_decode_pop() (in module bionet.ted.asdm), 20
- asdm_decode_pop_ins() (in module bionet.ted.asdm), 21
- asdm_decode_vander() (in module bionet.ted.asdm), 22
- asdm_decode_vander_ins() (in module bionet.ted.asdm), 22
- asdm_encode() (in module bionet.ted.asdm), 19
- asdm_recoverable() (in module bionet.ted.asdm), 23
- ASDMRealTimeDecoder (class in bionet.ted.rt), 40
- ASDMRealTimeDecoderIns (class in bionet.ted.rt), 41

---

**B**

- bpa() (in module bionet.ted.bpa), 46

---

**C**

- chi() (in module bionet.utils.scipy_extras), 57
- chunks() (in module bionet.utils.misc), 49
- ci() (in module bionet.utils.scipy_extras), 56
- contour() (in module bionet.utils.plotting), 52
- contourf() (in module bionet.utils.plotting), 53
- crand() (in module bionet.utils.numpy_extras), 50

---

**D**

- db() (in module bionet.utils.signal_extras), 57
- downsample() (in module bionet.utils.signal_extras), 58

---

**E**

- ei() (in module bionet.utils.scipy_extras), 57
- em() (in module bionet.utils.trig_poly), 60
- eps() (in module bionet.utils.pylab_extras), 56

---

**F**

- fftfilt() (in module bionet.utils.signal_extras), 58
- frame_compare() (in module bionet.utils.animate), 48
- func_timer() (in module bionet.utils.misc), 49

---

**G**

- gen_band_limited() (in module bionet.utils.band_limited), 48
- gen_dirichlet_coeffs() (in module bionet.utils.trig_poly), 60
- gen_trig_poly() (in module bionet.utils.trig_poly), 60
- gen_trig_poly_2d() (in module bionet.utils.trig_poly), 61
- get_dirichlet_coeffs() (in module bionet.utils.trig_poly), 61
H
hilb() (in module bionet.utils.numpy_extras), 50

I
iaf_decode() (in module bionet.ted.iaf), 24
iaf_decode() (in module bionet.ted.iaf_trig), 35
iaf_decode_coupled() (in module bionet.ted.iaf), 28
iaf_decode_delay() (in module bionet.ted.iaf), 31
iaf_decode_delay() (in module bionet.ted.rt), 44
iaf_decode_fast() (in module bionet.ted.iaf), 24
iaf_decode_pop() (in module bionet.ted.iaf), 29
iaf_decode_pop() (in module bionet.ted.iaf_trig), 36
iaf_decode_spline() (in module bionet.ted.iaf), 25
iaf_decode_spline_pop() (in module bionet.ted.iaf), 30
iaf_decode_vander() (in module bionet.ted.iaf), 26
iaf_encode() (in module bionet.ted.iaf), 26
iaf_encode_coupled() (in module bionet.ted.iaf), 27
iaf_encode_delay() (in module bionet.ted.iaf), 31
iaf_encode_delay() (in module bionet.ted.rt), 45
iaf_encode_pop() (in module bionet.ted.iaf), 32
iaf_recoverable() (in module bionet.ted.iaf), 35
IAFRealTimeDecoder (class in bionet.ted.rt), 42
IAFRealTimeEncoder (class in bionet.ted.rt), 43
iceil() (in module bionet.utils.numpy_extras), 50
ifloor() (in module bionet.utils.numpy_extras), 51
iround() (in module bionet.utils.numpy_extras), 51

M
mdot() (in module bionet.utils.numpy_extras), 51
minmax() (in module bionet.utils.pylab_extras), 56
mpower() (in module bionet.utils.numpy_extras), 51

N
nextpow2() (in module bionet.utils.signal_extras), 59

O
oddceil() (in module bionet.utils.signal_extras), 59
oddround() (in module bionet.utils.signal_extras), 59

P
plot_compare() (in module bionet.utils.plotting), 53
plot_encoded() (in module bionet.utils.plotting), 53
plot_fourier() (in module bionet.utils.plotting), 54
plot_raster() (in module bionet.utils.plotting), 54
plot_signal() (in module bionet.utils.plotting), 55

R
rank() (in module bionet.utils.numpy_extras), 52
realmax() (in module bionet.utils.pylab_extras), 56
realmin() (in module bionet.utils.pylab_extras), 56
RealTimeDecoder (class in bionet.ted.rt), 38
RealTimeEncoder (class in bionet.ted.rt), 39
remezord() (in module bionet.utils.signal_extras), 58
rms() (in module bionet.utils.signal_extras), 57
S
scale_down_coeffs() (in module bionet.utils.trig_poly), 62
SerialBuffer (class in bionet.utils.misc), 49
shi() (in module bionet.utils.scipy_extras), 57
si() (in module bionet.utils.scipy_extras), 57
SignalProcessor (class in bionet.ted.rt), 38
snr() (in module bionet.utils.signal_extras), 57
surf() (in module bionet.utils.plotting), 55
U
upsample() (in module bionet.utils.signal_extras), 59
W
wireframe() (in module bionet.utils.plotting), 55