TME

Release 0.1.0

Zheng Zhao

Oct 08, 2022

CONTENTS:

CHAPTER

ONE

INSTALLATION

You can install `tme` by either `pip` or `setup.py`.

1.1 If you want to use JaX TME

Please note that `jax` is **not** included in the requirement list, because we would like the user themself to manually install *jax*. For the installation of *jax*, see, this link.

If you just want to use SymPy TME, you can ignore installing JaX.

1.2 By `setup.py`

1. Open a terminal and run `git clone git@github.com:zgbkdlm/tme.git` via SSH or `git clone https://github.com/zgbkdlm/tme.git` via HTTPS.
2. Navidate to the python folder by running `cd python`.
3. Run `python setup.py install` or `python setup.py develop`.

1.3 By pip

Open a terminal and run `pip install tme`.

1.4 Matlab

No installation required. Simply git clone the repository and navigate to the folder `matlab` and you are good to go.

CHAPTER

TWO

TME APIS

2.1 TME in SymPy

Taylor moment expansion (TME) in SymPy.

TME applies on stochastic differential equations (SDEs) of the form

$$dX(t) = a(X(t))dt + b(X(t))dW(t),$$

where $X\colon \mathbb{T} \to \mathbb{R}^d$, and $W\colon \mathbb{T} \to \mathrm{R}^w$ is a Wiener process having the unit spectral density. Functions $a\colon \mathbb{R}^d \to \mathbb{R}^d$ and $b\colon \mathbb{R}^d \to \mathbb{R}^{d\times w}$ stand for the drift and dispersion coefficients, respectively. $\mathbb{T} = [t_0, \infty)$ stands for a temporal domain.

For more detailed explanations of TME, see, Zhao (2021) or Zhao et al. (2020). Notations used in this doc-site follows from Zhao (2021).

2.1.1 Functions

mean_and_cov()
: TME approximation for mean and covariance. In case you just want to compute the mean, use function *expectation()* with argument `phi` fed by an identity function.

expectation()
: TME approximation for any expectation of the form $\mathbb{E}[\phi(X(t+\Delta t)) \mid X(t)]$.

generator()
: Infinitesimal generator.

generator_vec()
: Infinitesimal generator extended for vector-valued target function.

generator_mat()
: Infinitesimal generator extended for matrix-valued target function.

generator_power()
: Iterations/power of infinitesimal generators.

2.1.2 References

Zheng Zhao. State-space deep Gaussian processes. PhD thesis, Aalto University. 2021.

Zheng Zhao, Toni Karvonen, Roland Hostettler, and Simo Särkkä. Taylor Moment Expansion for Continuous-Discrete Gaussian Filtering. IEEE Transactions on Automatic Control, 66(9):4460-4467, 2021.

Didier Dacunha-Castelle and Danielle Florens-Zmirou. Estimation of the coefficients of a diffusion from discrete observations. Stochastics, 19(4):263–284, 1986.

Danielle Florens-Zmirou. Approximate discrete-time schemes for statistics of diffusion processes. Statistics, 20(4):547–557, 1989.

2.1.3 Notes

Currently, this symbolc implementation has a problem as the SymPy simplification function `sympy.simplify` is not working well. See details in the docstring of function *mean_and_cov()*.

2.1.4 Authors

Zheng Zhao, 2020, zz@zabemon.com, https://zz.zabemon.com

`tme.base_sympy.`**`expectation`**(*phi*, *x*, *drift*, *dispersion*, *dt=None*, *order=3*, *simp=True*)

TME approximation of expectation on target function ϕ.

Formally, this function approximates

$$\mathbb{E}[\phi(X(t+\Delta t)) \mid X(t)],$$

where $\phi\colon \mathbb{R}^d \to \mathbb{R}^{m\times n}$ is any smooth enough function of interest.

Parameters

phi
[Matrix] Target function.

x
[MatrixSymbol] Symbolic state vector.

drift
[Matrix] Symbolic drift coefficient.

dispersion
[Matrix] Symbolic dispersion coefficient.

dt
[Symbol, optional] Symbolic time interval. If this is not specified by the user, then the function will create one.

order
[int, default=3] Order of TME.

simp
[bool or Callable, default=True] Set True to simplify the results by calling the default `sympy.simplify` method. You can also give your own simplification method as a callable function input.

Returns

Matrix
TME approximation of $\mathbb{E}[\phi(X(t+\Delta t)) \mid X(t)]$.

Return type
`Matrix`

`tme.base_sympy.generator`(*phi*, *x*, *drift*, *dispersion*)

Infinitesimal generator for diffusion processes in Ito's SDE constructions.

$$(\mathcal{A}\phi)(x) = \sum_{i=1}^{d} a_i(x)\,\frac{\partial \phi}{\partial x_i}(x) + \frac{1}{2}\sum_{i,j=1}^{d} \Gamma_{ij}(x)\,\frac{\partial^2 \phi}{\partial x_i\,\partial x_j}(x),$$

where $\phi\colon \mathbb{R}^d \to \mathbb{R}$ must be sufficiently smooth function depending on the expansion order, and $\Gamma(x) = b(x)\,b(x)^\top$.

Parameters

phi
[Expr] Scalar-valued target function.

x
[MatrixSymbol] Symbolic state vector.

drift
[Matrix] Symbolic drift coefficient.

dispersion
[Matrix] Symbolic dispersion coefficient.

Returns

Expr
Symbolic $(\mathcal{A}\phi)(x)$.

Return type
`Expr`

`tme.base_sympy.generator_mat`(*phi_mat*, *x*, *drift*, *dispersion*)

Infinitesimal generator for matrix-valued $\phi\colon \mathbb{R}^d \to \mathbb{R}^{m\times n}$.

This function exactly corresponds to the operator $\overline{\mathcal{A}}$ in Zhao (2021).

Parameters

phi_mat
[Matrix] Matrix-valued target function.

x
[MatrixSymbol] Symbolic state vector.

drift
[Matrix] Symbolic drift coefficient.

dispersion
[Matrix] Symbolic dispersion coefficient.

Returns

Matrix
Symbolic $(\mathcal{A}\phi)(x)$.

Return type
`Matrix`

`tme.base_sympy.`**`generator_power`**(*phi*, *x*, *drift*, *dispersion*, *p*)

Iterations/power of infinitesimal generator for scalar/vector/matrix-valued ϕ.

$$(\mathcal{A}^p\phi)(x),$$

where p is the number of iterations.

Parameters

phi
[Matrix] Scalar/vector/Matrix-valued target function.

x
[MatrixSymbol] Symbolic state vector.

drift
[Matrix] Symbolic drift coefficient.

dispersion
[Matrix] Symbolic dispersion coefficient.

p
[int] Power/number of iterations.

Returns

List[Matrix]
A list of iterated generators $[(\mathcal{A}^0\phi)(x), (\mathcal{A}\phi)(x), \ldots, (\mathcal{A}^p\phi)(x)]$.

Return type
`List[Matrix]`

`tme.base_sympy.`**`generator_vec`**(*phi_vec*, *x*, *drift*, *dispersion*)

Infinitesimal generator for vector-valued $\phi\colon \mathbb{R}^d \to \mathbb{R}^m$.

Parameters

phi_vec
[Matrix] Vector-valued target function.

x
[MatrixSymbol] Symbolic state vector.

drift
[Matrix] Symbolic drift coefficient.

dispersion
[Matrix] Symbolic dispersion coefficient.

Returns

Matrix
Symbolic $(\mathcal{A}\phi)(x)$.

Notes

Since we are using `sympy.Matrix`, the function *generator_mat()* can exactly replace this function.

Return type
`Matrix`

`tme.base_sympy.`**mean_and_cov**(*x*, *drift*, *dispersion*, *dt*, *order=3*, *simp=True*)

TME approximation for mean and covariance.

Formally, this function approximates

$$\mathbb{E}[X(t+\Delta t) \mid X(t)],$$
$$\mathrm{Cov}[X(t+\Delta t) \mid X(t)].$$

See, Zhao (2021, Lemma 3.4) for details.

Parameters

x
[MatrixSymbol] Symbolic state vector.

drift
[Matrix] Symbolic drift coefficient.

dispersion
[Matrix] Symbolic dispersion coefficient.

dt
[Symbol] Symbolic time interval. You can create one, for example, by `dt=sympy.Symbol('dt', positive=True)`.

order
[int, default=3] Order of TME.

simp
[bool or Callable, default=True] Set `True` to simplify the results by calling the default `sympy.simplify` method. You can also use your own simplification method by feeding it a callable function.

> **Warning:** The method `sympy.simplify` can be unnecessarily slow when the system is complicated, see this page for details.

Returns

m
[Matrix] TME approximation of mean $\mathbb{E}[X(t+\Delta t) \mid X(t)]$.

cov
[Matrix] TME approximation of covariance $\mathrm{Cov}[X(t+\Delta t) \mid X(t)]$.

Return type
`Tuple[Matrix, Matrix]`

2.2 TME in TensorFlow

To be supported in future. You are very welcome to contribute!

2.3 TME in JaX

Taylor moment expansion (TME) in JaX.

For math details, please see the docstring of *tme.base_sympy*.

2.3.1 Functions

generator()
: Infinitesimal generator. This is a helper function around *generator_power()*.

generator_power()
: Iterations/power of infinitesimal generators.

mean_and_cov()
: TME approximation for mean and covariance. In case you just want to compute the mean, use function *expectation()* with argument `phi` fed by an identity function.

expectation()
: TME approximation for any expectation of the form $\mathbb{E}[\phi(X(t+\Delta t)) \mid X(t)]$.

2.3.2 References

See the docstring of *tme.base_sympy*.

2.3.3 Authors

Adrien Corenflos and Zheng Zhao, 2021

`tme.base_jax.`**`expectation`**(*phi*, *x*, *dt*, *drift*, *dispersion*, *order=3*)

TME approximation of expectation on any target function ϕ.

For math details, see the docstring of *tme.base_sympy.expectation()*.

Parameters

phi
: [Callable (d,) -> (...)] Target function (must be sufficiently smooth depending on the order).

x
: [jnp.ndarray (d,)] The state at which the generator is evaluated (i.e., the x in $\mathbb{E}[\phi(X(t+\Delta t)) \mid X(t) = x]$).

dt
: [float] Time interval.

drift
: [Callable (d,) -> (d,)] SDE drift coefficient.

dispersion
[Callable (d,) -> (d, w)] SDE dispersion coefficient, where *w* stands for the dimension of the Wiener process.

order
[int] Order of TME. Must be >=0. For the relationship between the expansion order and SDE coefficient smoothness, see, Zhao (2021).

Returns

jnp.ndarray (...)
TME approximation of $\mathbb{E}[\phi(X(t+\Delta t)) \mid X(t)]$. The output shape is consistence with the input shape of `phi`.

Return type
`ndarray`

`tme.base_jax.generator`(*phi*, *drift*, *dispersion*)

Infinitesimal generator for diffusion processes in Ito's SDE constructions.

$$(\mathcal{A}\phi)(x) = \sum_{i=1}^{d} a_i(x)\,\frac{\partial \phi}{\partial x_i}(x) + \frac{1}{2}\sum_{i,j=1}^{d} \Gamma_{ij}(x)\,\frac{\partial^2 \phi}{\partial x_i\,\partial x_j}(x),$$

where $\phi\colon \mathbb{R}^d \to \mathbb{R}$ must be sufficiently smooth function depending on the expansion order, and $\Gamma(x) = b(x)\,b(x)^\top$.

This is a helper function around *generator_power()*.

Parameters

phi
[Callable (d,) -> (...)] Target function.

drift
[Callable (d,) -> (d,)] SDE drift coefficient.

dispersion
[Callable (d,) -> (d, w)] SDE dispersion coefficient, where *w* stands for the dimension of the Wiener process.

Returns

Callable (...)
A callable function which carries out $x \mapsto \mathcal{A}\phi$. The output shape of this function is the same as `phi`.

Return type
`Callable`

`tme.base_jax.generator_power`(*phi*, *drift*, *dispersion*, *order=1*)

Iterations/power of infinitesimal generator.

For math details, see the docstring of *tme.base_sympy.generator_power()*.

Parameters

phi
[Callable (d,) -> (...)] Target function.

drift
[Callable (d,) -> (d,)] SDE drift coefficient.

dispersion
[Callable (d,) -> (d, w)] SDE dispersion coefficient, where *w* stands for the dimension of the Wiener process.

order
[int, optional] Number of generator iterations. Must be >=0. Default is 1, which corresponds to the standard infinitesimal generator.

Returns

List[Callable]
List of generator functions in ascending power order. Formally, this function returns $[\phi, \mathcal{A}\phi, \ldots, \mathcal{A}^p\phi]$, where `p` is the order. Each callable function in this list has exactly the same input-output shape signature as phi: (d,) -> (...).

Notes

The implementation is due to Adrien Corenflos. Thank you for contributing this.

You may also find a naive implementation of infinitesimal generators and their iterations in the test file `./test/test_tme_jax.py`.

Return type
`List[Callable]`

`tme.base_jax.mean_and_cov`(*x*, *dt*, *drift*, *dispersion*, *order=3*)

TME approximation for mean and covariance.

For math details, see the docstring of *tme.base_sympy.mean_and_cov()*.

Parameters

x
[jnp.ndarray (d,)] The state at which the generator is evaluated. (i.e., the x in $\mathbb{E}[X(t+\Delta t) \mid X(t) = x]$ and $\mathrm{Cov}[X(t+\Delta t) \mid X(t) = x]$).

dt
[float] Time interval.

drift
[Callable (d,) -> (d,)] SDE drift coefficient.

dispersion
[Callable (d,) -> (d, w)] SDE dispersion coefficient, where *w* stands for the dimension of the Wiener process.

order
[int, default=3] Order of TME. Must be >= 1.

Returns

m
[jnp.ndarray (d,)] TME approximation of mean $\mathbb{E}[X(t+\Delta t) \mid X(t) = x]$.

cov
[jnp.ndarray (d, d)] TME approximation of covariance $\mathrm{Cov}[X(t+\Delta t) \mid X(t) = x]$.

Notes

When *order = 1*, the TME mean and cov approximations are exactly the same with Euler–Maruyama.

Return type
`Tuple`[`ndarray`, `ndarray`]

2.4 TME in Matlab

Docing Matlab codes takes a bit of efforts. We refer the user to the docstrings in .mat files for details.

In folder `matlab`, you can find

1. folder +`models` containing some example models;
2. folder +`tools` containing the implementations of TME as well as some tool functions.

The file `TME.m` is the main file of TME in Matlab.

CHAPTER

THREE

EXAMPLES

This section shows runnable TME examples in Python and Matlab. In particular, we use the following two example models.

1. Model 1,

$$dX(t) = \tanh(X(t))dt + dW(t).$$

2. Model 2,

$$\begin{aligned} dX_1(t) &= X_2(t), \\ dX_2(t) &= (X_1(t)\,(\kappa - (X_1(t))^2))dt + X_1(t)dW(t). \end{aligned}$$

We want to compute their mean, covariance/variance, or more generally $\mathbb{E}[\phi(X(t+\Delta t)) \mid X(t)]$ for any function of interest ϕ.

3.1 In Python

See, Jupyter Notebook (SymPy) for Model 1.

See, Jupyter Notebook (JaX) for Model 1.

See, Jupyter Notebook (JaX) for Model 2.

See, Jupyter Notebook (JaX) for a stochastic Lorenz model.

3.2 In Matlab

See, Matalb codes for Models 1 and 2

CHAPTER

FOUR

WHAT IS TME?

To be added.

CHAPTER

FIVE

TME FILTERS AND SMOOTHERS

Since TME can be used to approximate any expectation of the form $\mathbb{E}[\phi(X(t)) \mid X(s)]$, it is natural to apply TME for Gaussian filtering and smoothing (Zhao, 2021).

See, a Python (Jax) implementation of TME Gaussian filters and smoothers in https://github.com/zgbkdlm/tmefs.

CHAPTER

SIX

BIBLIOGRAPHY

The TME method is due to Dacunha-Castelle and Florens-Zmirou (1986); Florens-Zmirou (1989), where they use it to estimate diffusion processes coefficients.

It is also shown by Zhao et al. (2020) and Zhao (2021) that TME can be used in Gaussian filtering and smoothering, as well as deep GP regression problems.

- Didier Dacunha-Castelle and Danielle Florens-Zmirou. Estimation of the coefficients of a diffusion from discrete observations. Stochastics, 19(4):263–284, 1986.
- Danielle Florens-Zmirou. Approximate discrete-time schemes for statistics of diffusion processes. Statistics, 20(4):547–557, 1989.
- Mathieu Kessler. Estimation of an ergodic diffusion from discrete observations. Scandinavian Journal of Statistics, 24(2):211-229, 1997.
- Zheng Zhao. State-space deep Gaussian processes. PhD thesis, Aalto University, 2021. [PDF]
- Zheng Zhao and Simo Särkkä. Non-linear Gaussian smoothing with Taylor moment expansion. IEEE Signal Processing Letters, 2021.
- Zheng Zhao, Toni Karvonen, Roland Hostettler, and Simo Särkkä. Taylor moment expansion for continuous-discrete Gaussian filtering. IEEE Transactions on Automatic Control, 66(9):4460-4467, 2021. [DOI]

Taylor moment expansion (TME) is a method for approximating propogations (in expectations) of stochastic differential equations (SDEs) solutions through (non-linear) functions. More formally, consider X be the solution of an SDE, then the TME method is able to approximate $\mathbb{E}[\phi(X(t)) \mid X(s)]$ for any target function ϕ of interest and times $t < s$. This method is originally due to Dacunha-Castelle and Florens-Zmirou (1986) and Florens-Zmirou (1989). A more modern interpretation of the method is found in Zhao (2021).

This site consists of the documentation of the implementations of TME in Python and Matlab, (while we focus on Python a bit more). In addition, the Matlab implementation features a few examples of TME-based Gaussian filters and smoothers.

The source codes are availabe at the git repository https://github.com/zgbkdlm/tme.

The implementations and math in the docstrings are based on Zhao (2021).

To begin, you can start with the examples in *Examples*.

CHAPTER

SEVEN

JAX VS SYMPY

For some reasons we implemented TME in both SymPy and JaX. However, we would suggest to use JaX as much as you can, due to two reasons: 1) the JaX implementaion is usually more efficient. 2) the simplication method *sympy.simplify()* in SymPy is sometimes inefficient.

However, the SymPy codes are written in an exactly-as-per-equation fashion, so the SymPy version can be a good reference implementaion.

CHAPTER

EIGHT

LICENSE

GPL v3 or later.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX