PySME Documentation

Release 0.1

Jonathan Gross

June 08, 2016

Contents

Contents:

CHAPTER 1

Code documentation

1.1 system_builder

system_builder.**diffusion_op**(*dim*, *C_vector*, *triple_prods*, *basis_norms_sq*, *basis*, ***kwargs*)

Return a matrix D such that when ρ is vectorized the expression

$$\frac{d\rho}{dt} = \mathcal{D}[c]\rho = c\rho c^\dagger - \frac{1}{2}(c^\dagger c\rho + \rho c^\dagger c)$$

can be calculated by:

$$\frac{d\vec{\rho}}{dt} = D\vec{\rho}$$

Vectorization is done according to the order prescribed in *basis*, with the component proportional to identity in the last place.

Parameters

- **coupling_op** (*numpy.array*) – The operator c in matrix form
- **basis** (*list(numpy.array)*) – An almost complete (minus identity), Hermitian, traceless, orthogonal basis for the operators (does not need to be normalized).

Returns The matrix D operating on a vectorized density operator

Return type numpy.array

system_builder.**double_comm_op**(*dim*, *C_vector*, *triple_prods*, *M_sq*, *basis_norms_sq*, *basis*, ***kwargs*)

Return a matrix E such that when ρ is vectorized the expression

$$\frac{d\rho}{dt} = \left(\frac{M^*}{2}[c,[c,\rho]] + \frac{M}{2}[c^\dagger,[c^\dagger,\rho]]\right)$$

can be calculated by:

$$\frac{d\vec{\rho}}{dt} = E\vec{\rho}$$

Vectorization is done according to the order prescribed in *basis*, with the component proportional to identity in the last place.

Parameters

- **coupling_op** (*numpy.array*) – The operator c in matrix form

- **M_sq** (*complex*) – Complex squeezing parameter M defined by $\langle dB(t)dB(t)\rangle = Mdt$.
- **basis** (*list(numpy.array)*) – An almost complete (minus identity), Hermitian, traceless, orthogonal basis for the operators (does not need to be normalized).

Returns The matrix E operating on a vectorized density operator

Return type numpy.array

system_builder.**dualize**(*operator*, *basis*)

Take an operator to its dual vectorized form in a particular operator basis.

Designed to work in conjunction with `vectorize` so that, given an orthogonal basis $\{\Lambda^m\}$ where $\mathrm{Tr}[{\Lambda^m}^\dagger \Lambda^n] \propto \delta_{mn}$, the dual action of an operator A on another operator B interpreted as $\mathrm{Tr}[A^\dagger B]$ can be easily calculated by `sum([a*b for a, b in zip(dualize(A), vectorize(B))]` (in other words it becomes an ordinaty dot product in this particular representation).

Parameters

- **operator** (*numpy.array*) – The operator to vectorize
- **basis** (*list(numpy.array)*) – The basis to vectorize the operator in

Returns The vector components

Return type numpy.array

system_builder.**hamiltonian_op**(*dim*, *H_vector*, *double_prods*, *basis_norms_sq*, *basis*, ***kwargs*)

Return a matrix F such that when ρ is vectorized the expression

$$\frac{d\rho}{dt} = -i[H, \rho]$$

can be calculated by:

$$\frac{d\vec{\rho}}{dt} = F\vec{\rho}$$

Vectorization is done according to the order prescribed in *basis*, with the component proportional to identity in the last place.

Parameters

- **hamiltonian** (*numpy.array*) – The Hamiltonian H in matrix form
- **basis** (*list(numpy.array)*) – An almost complete (minus identity), Hermitian, traceless, orthogonal basis for the operators (does not need to be normalized).

Returns The matrix F operating on a vectorized density operator

Return type numpy.array

system_builder.**norm_squared**(*operator*)

Returns the square of the Frobenius norm of the operator.

Parameters **operator** (*numpy.array*) – The operator for which to calculate the squared norm

Returns The square of the norm of the operator

Return type Positive real

system_builder.**op_calc_setup**(*coupling_op*, *M_sq*, *N*, *H*, *partial_basis*)

Handle the repeated tasks performed every time a superoperator matrix is computed.

system_builder.**recur_dot**(*mats*)

Perform numpy.dot on a list in a right-associative manner.

system_builder.**vectorize**(*operator*, *basis*)
Vectorize an operator in a particular operator basis.

Parameters

- **operator** (*numpy.array*) – The operator to vectorize
- **basis** (*list(numpy.array)*) – The basis to vectorize the operator in

Returns The vector components

Return type numpy.array

system_builder.**weiner_op**(*dim*, *C_vector*, *double_prods*, *basis_norms_sq*, *basis*, ***kwargs*)
Return a the matrix-vector pair $(G, \vec{k})$ such that when ρ is vectorized the expression

$$d\rho = dW\ \left(c\rho + \rho c^\dagger - \rho\,\mathrm{Tr}[(c + c^\dagger)\rho]\right)$$

can be calculated by:

$$d\vec{\rho} = dW\,(G + \vec{k}\cdot\vec{\rho})\vec{\rho}$$

Vectorization is done according to the order prescribed in *basis*, with the component proportional to identity in the last place.

Parameters

- **coupling_op** (*numpy.array*) – The operator c in matrix form
- **basis** (*list(numpy.array)*) – An almost complete (minus identity), Hermitian, traceless, orthogonal basis for the operators (does not need to be normalized).

Returns The matrix-vector pair $(G, \vec{k})$ operating on a vectorized density operator (k is returned as a row-vector)

Return type tuple(numpy.array)

1.2 gellmann

gellmann.**gellmann**(*j*, *k*, *d*)
Returns a generalized Gell-Mann matrix of dimension d. According to the convention in *Bloch Vectors for Qubits* by Bertlmann and Krammer (2008), returns Λ^j for $1 \leq j = k \leq d-1$, Λ_s^{kj} for $1 \leq k < j \leq d$, Λ_a^{jk} for $1 \leq j < k \leq d$, and I for $j = k = d$.

Parameters

- **j** (*positive integer*) – First index for generalized Gell-Mann matrix
- **k** (*positive integer*) – Second index for generalized Gell-Mann matrix
- **d** (*positive integer*) – Dimension of the generalized Gell-Mann matrix

Returns A genereralized Gell-Mann matrix.

Return type numpy.array

gellmann.**get_basis**(*d*)
Return a basis of orthogonal Hermitian operators on a Hilbert space of dimension d, with the identity element in the last place.

1.3 gramschmidt

gramschmidt.**orthonormalize**(*A*)

Return an orthonormal basis where A has support only on the first three elements. The first element is guaranteed to be proportional to the identity.

1.4 integrate

1.5 sde

sde.**euler**(*drift_fn*, *diffusion_fn*, *X0*, *ts*, *Us*)

Integrate a system of ordinary stochastic differential equations subject to scalar noise:

$$d\vec{X} = \vec{a}(\vec{X}, t)\, dt + \vec{b}(\vec{X}, t)\, dW_t$$

Uses the Euler method:

$$\vec{X}_{i+1} = \vec{X}_i + \vec{a}(\vec{X}_i, t_i)\Delta t_i + \vec{b}(\vec{X}_i, t_i)\Delta W_i$$

where $\Delta W_i = U_i\sqrt{\Delta t}$, U being a normally distributed random variable with mean 0 and variance 1.

Parameters

- **drift_fn** (*callable(X, t)*) – Computes the drift coefficient $\vec{a}(\vec{X}, t)$
- **diffusion_fn** (*callable(X, t)*) – Computes the diffusion coefficient $\vec{b}(\vec{X}, t)$
- **X0** (*array*) – Initial condition on X
- **ts** (*array*) – A sequence of time points for which to solve for X. The initial value point should be the first element of this sequence.
- **Us** (*array, shape=(len(t) - 1)*) – Normalized Weiner increments for each time step (i.e. samples from a Gaussian distribution with mean 0 and variance 1).

Returns Array containing the value of X for each desired time in t, with the initial value *X0* in the first row.

Return type numpy.array, shape=(len(ts), len(X0))

sde.**faulty_milstein**(*drift*, *diffusion*, *b_dx_b*, *X0*, *ts*, *Us*)

Integrate a system of ordinary stochastic differential equations subject to scalar noise:

$$d\vec{X} = \vec{a}(\vec{X}, t)\, dt + \vec{b}(\vec{X}, t)\, dW_t$$

Uses a faulty Milstein method (i.e. missing the factor of 1/2 in the term added to Euler integration):

$$\vec{X}_{i+1} = \vec{X}_i + \vec{a}(\vec{X}_i, t_i)\Delta t_i + \vec{b}(\vec{X}_i, t_i)\Delta W_i + \left(\vec{b}(\vec{X}_i, t_i) \cdot \vec{\nabla}_{\vec{X}}\right) \vec{b}(\vec{X}_i, t_i)\left((\Delta W_i)^2 - \Delta t_i\right)$$

where $\Delta W_i = U_i\sqrt{\Delta t}$, U being a normally distributed random variable with mean 0 and variance 1.

Parameters

- **drift** (*callable(X, t)*) – Computes the drift coefficient $\vec{a}(\vec{X}, t)$
- **diffusion** (*callable(X, t)*) – Computes the diffusion coefficient $\vec{b}(\vec{X}, t)$

- **b_dx_b** (*callable(X, t)*) – Computes the correction coefficient $\left(\vec{b}(\vec{X},t)\cdot\vec{\nabla}_{\vec{X}}\right)\vec{b}(\vec{X},t)$
- **X0** (*array*) – Initial condition on X
- **ts** (*array*) – A sequence of time points for which to solve for X. The initial value point should be the first element of this sequence.
- **Us** (*array, shape=(len(t) - 1)*) – Normalized Weiner increments for each time step (i.e. samples from a Gaussian distribution with mean 0 and variance 1).

Returns Array containing the value of X for each desired time in t, with the initial value *X0* in the first row.

Return type numpy.array, shape=(len(t), len(X0))

sde.**meas_euler**(*drift_fn*, *diffusion_fn*, *dW_fn*, *X0*, *ts*, *dMs*)

Integrate a system of ordinary stochastic differential equations conditioned on an incremental measurement record:

$$d\vec{X} = \vec{a}(\vec{X},t)\,dt + \vec{b}(\vec{X},t)\,dW_t$$

Uses the Euler method:

$$\vec{X}_{i+1} = \vec{X}_i + \vec{a}(\vec{X}_i,t_i)\Delta t_i + \vec{b}(\vec{X}_i,t_i)\Delta W_i$$

where $\Delta W_i = f(\Delta M_i, \vec{X}, t)$, ΔM_i being the incremental measurement record being used to drive the SDE.

Parameters

- **drift_fn** (*callable(X, t)*) – Computes the drift coefficient $\vec{a}(\vec{X},t)$
- **diffusion_fn** (*callable(X, t)*) – Computes the diffusion coefficient $\vec{b}(\vec{X},t)$
- **dW_fn** (*callable(dM, dt, X, t)*) – The function that converts the incremental measurement and current state to the Wiener increment.
- **X0** (*array*) – Initial condition on X
- **ts** (*array*) – A sequence of time points for which to solve for X. The initial value point should be the first element of this sequence.
- **dMs** (*array, shape=(len(t) - 1)*) – Incremental measurement outcomes used to drive the SDE.

Returns Array containing the value of X for each desired time in t, with the initial value *X0* in the first row.

Return type numpy.array, shape=(len(ts), len(X0))

sde.**meas_milstein**(*drift_fn*, *diffusion_fn*, *b_dx_b_fn*, *dW_fn*, *X0*, *ts*, *dMs*)

Integrate a system of ordinary stochastic differential equations conditioned on an incremental measurement record:

$$d\vec{X} = \vec{a}(\vec{X},t)\,dt + \vec{b}(\vec{X},t)\,dW_t$$

Uses the Milstein method:

$$\vec{X}_{i+1} = \vec{X}_i + \vec{a}(\vec{X}_i,t_i)\Delta t_i + \vec{b}(\vec{X}_i,t_i)\Delta W_i + \frac{1}{2}\left(\vec{b}(\vec{X}_i,t_i)\cdot\vec{\nabla}_{\vec{X}}\right)\vec{b}(\vec{X}_i,t_i)\left((\Delta W_i)^2 - \Delta t_i\right)$$

where $\Delta W_i = f(\Delta M_i, \vec{X}, t)$, ΔM_i being the incremental measurement record being used to drive the SDE.

Parameters

- **drift_fn** (*callable(X, t)*) – Computes the drift coefficient $\vec{a}(\vec{X}, t)$
- **diffusion_fn** (*callable(X, t)*) – Computes the diffusion coefficient $\vec{b}(\vec{X}, t)$
- **b_dx_b_fn** (*callable(X, t)*) – Computes the correction coefficient $\left(\vec{b}(\vec{X}, t) \cdot \vec{\nabla}_{\vec{X}}\right) \vec{b}(\vec{X}, t)$
- **dW_fn** (*callable(dM, dt, X, t)*) – The function that converts the incremental measurement and current state to the Wiener increment.
- **X0** (*array*) – Initial condition on X
- **ts** (*array*) – A sequence of time points for which to solve for X. The initial value point should be the first element of this sequence.
- **dMs** (*array, shape=(len(t) - 1)*) – Incremental measurement outcomes used to drive the SDE.

Returns Array containing the value of X for each desired time in t, with the initial value *X0* in the first row.

Return type numpy.array, shape=(len(ts), len(X0))

sde.**milstein**(*drift*, *diffusion*, *b_dx_b*, *X0*, *ts*, *Us*)

Integrate a system of ordinary stochastic differential equations subject to scalar noise:

$$d\vec{X} = \vec{a}(\vec{X}, t)\, dt + \vec{b}(\vec{X}, t)\, dW_t$$

Uses the Milstein method:

$$\vec{X}_{i+1} = \vec{X}_i + \vec{a}(\vec{X}_i, t_i)\Delta t_i + \vec{b}(\vec{X}_i, t_i)\Delta W_i + \frac{1}{2}\left(\vec{b}(\vec{X}_i, t_i) \cdot \vec{\nabla}_{\vec{X}}\right) \vec{b}(\vec{X}_i, t_i)\left((\Delta W_i)^2 - \Delta t_i\right)$$

where $\Delta W_i = U_i\sqrt{\Delta t}$, U being a normally distributed random variable with mean 0 and variance 1.

Parameters

- **drift** (*callable(X, t)*) – Computes the drift coefficient $\vec{a}(\vec{X}, t)$
- **diffusion** (*callable(X, t)*) – Computes the diffusion coefficient $\vec{b}(\vec{X}, t)$
- **b_dx_b** (*callable(X, t)*) – Computes the correction coefficient $\left(\vec{b}(\vec{X}, t) \cdot \vec{\nabla}_{\vec{X}}\right) \vec{b}(\vec{X}, t)$
- **X0** (*array*) – Initial condition on X
- **ts** (*array*) – A sequence of time points for which to solve for X. The initial value point should be the first element of this sequence.
- **Us** (*array, shape=(len(t) - 1)*) – Normalized Weiner increments for each time step (i.e. samples from a Gaussian distribution with mean 0 and variance 1).

Returns Array containing the value of X for each desired time in t, with the initial value *X0* in the first row.

Return type numpy.array, shape=(len(ts), len(X0))

sde.**time_ind_taylor_1_5**(*drift*, *diffusion*, *b_dx_b*, *b_dx_a*, *a_dx_b*, *a_dx_a*, *b_dx_b_dx_b*, *b_b_dx_dx_b*, *b_b_dx_dx_a*, *X0*, *ts*, *U1s*, *U2s*)

Integrate a system of ordinary stochastic differential equations with time-independent coefficients subject to scalar noise:

$$d\vec{X} = \vec{a}(\vec{X})\, dt + \vec{b}(\vec{X})\, dW_t$$

Uses an order 1.5 Taylor method:

$$\rho_{i+1}^{\mu} = \rho_i^{\mu} + a_i^{\mu}\Delta t_i + b_i^{\mu}\Delta W_i + \frac{1}{2}b_i^{\nu}\partial_{\nu}b_i^{\mu}\left((\Delta W_i)^2 - \Delta t_i\right) + \tag{1.1}$$

$$b_i^{\nu}\partial_{\nu}a_i^{\mu}\Delta Z_i + \left(a_i^{\nu}\partial_{\nu} + \frac{1}{2}b_i^{\nu}b_i^{\sigma}\partial_{\nu}\partial_{\sigma}\right)b_i^{\mu}\left(\Delta W_i\Delta t_i - \Delta Z_i\right) \tag{1.2}$$

$$\frac{1}{2}\left(a_i^{\nu}\partial_{\nu} + \frac{1}{2}b_i^{\nu}b_i^{\sigma}\partial_{\nu}\partial_{\sigma}\right)a_i^{\mu}\Delta t_i^2 + \frac{1}{2}b_i^{\nu}\partial_{\nu}b_i^{\sigma}\partial_{\sigma}b_i^{\mu}\left(\frac{1}{3}(\Delta W_i)^2 - \Delta t_i\right) \tag{1.3}$$

Parameters

- **drift** (*callable(X)*) – Computes the drift coefficient $\vec{a}(\vec{X})$
- **diffusion** (*callable(X)*) – Computes the diffusion coefficient $\vec{b}(\vec{X})$
- **b_dx_b** (*callable(X)*) – Computes the coefficient $\left(\vec{b}(\vec{X})\cdot\vec{\nabla}_{\vec{X}}\right)\vec{b}(\vec{X})$
- **b_dx_a** (*callable(X)*) – Computes the coefficient $\left(\vec{b}(\vec{X})\cdot\vec{\nabla}_{\vec{X}}\right)\vec{a}(\vec{X})$
- **a_dx_b** (*callable(X)*) – Computes the coefficient $\left(\vec{a}(\vec{X})\cdot\vec{\nabla}_{\vec{X}}\right)\vec{b}(\vec{X})$
- **a_dx_a** (*callable(X)*) – Computes the coefficient $\left(\vec{a}(\vec{X})\cdot\vec{\nabla}_{\vec{X}}\right)\vec{a}(\vec{X})$
- **b_dx_b_dx_b** (*callable(X)*) – Computes the coefficient $\left(\vec{b}(\vec{X})\cdot\vec{\nabla}_{\vec{X}}\right)^2\vec{b}(\vec{X})$
- **b_b_dx_dx_b** – Computes $b^{\nu}b^{\sigma}\partial_{\nu}\partial_{\sigma}b^{\mu}\hat{e}_{\mu}$.
- **b_b_dx_dx_a** – Computes $b^{\nu}b^{\sigma}\partial_{\nu}\partial_{\sigma}a^{\mu}\hat{e}_{\mu}$.
- **X0** (*array*) – Initial condition on X
- **ts** (*array*) – A sequence of time points for which to solve for X. The initial value point should be the first element of this sequence.
- **U1s** (*array, shape=(len(t) - 1)*) – Normalized Weiner increments for each time step (i.e. samples from a Gaussian distribution with mean 0 and variance 1).
- **U2s** (*array, shape=(len(t) - 1)*) – Normalized Weiner increments for each time step (i.e. samples from a Gaussian distribution with mean 0 and variance 1).

Returns Array containing the value of X for each desired time in t, with the initial value *X0* in the first row.

Return type numpy.array, shape=(len(t), len(X0))

1.6 grid_conv

Functions for testing convergence rates using grid convergence

grid_conv.**calc_rate**(*integrator*, *rho_0*, *times*, *U1s=None*, *U2s=None*)

Calculate the convergence rate for some integrator.

Parameters

- **integrator** – An Integrator object.
- **rho_0** (*numpy.array*) – The initial state of the system
- **times** – Sequence of times (assumed to be evenly spaced, defining a number of increments divisible by 4).

- **U1s** (*numpy.array(len(times) - 1)*) – Samples from a standard-normal distribution used to construct Wiener increments ΔW for each time interval. If not provided will be generated by the function.
- **U2s** (*numpy.array(len(times) - 1)*) – Samples from a standard-normal distribution used to construct multiple-Ito increments ΔZ for each time interval. If not provided will be generated by the function.

Returns The convergence rate as a power of Δt.

Return type float

grid_conv.**double_increments**(*times*, *U1s*, *U2s=None*)

Take a list of times (assumed to be evenly spaced) and standard-normal random variables used to define the Ito integrals on the intervals and return the equivalent lists for doubled time intervals. The new standard-normal random variables are defined in terms of the old ones by

Parameters

- **times** (*numpy.array*) – List of evenly spaced times defining an even number of time intervals.
- **U1s** (*numpy.array(N, len(times) - 1)*) – Samples from a standard-normal distribution used to construct Wiener increments ΔW for each time interval. Multiple rows may be included for independent trajectories.
- **U2s** (*numpy.array(N, len(times) - 1)*) – Samples from a standard-normal distribution used to construct multiple-Ito increments ΔZ for each time interval. Multiple rows may be included for independent trajectories.

Returns Times sampled at half the frequency and the modified standard-normal-random-variable samples for the new intervals. If `U2s=None`, only new U1s are returned.

Return type (numpy.array(len(times)//2 + 1), numpy.array(len(times)//2)[, numpy.array(len(times)//2]))

CHAPTER 2

Vectorization

In this module, integration of ordinary and stochastic master equations is performed on density operators parametrized by d^2 real numbers, where d is the dimension of the system Hilbert space. These are the components of the density operator as a vector in a basis that is Hermitian and, excepting the identity, traceless. Since the ordinary and stochastic master equations under consideration are trace preserving, one could neglect the basis element corresponding to the identity, but as the module currently stands it is included to simplify some expressions and provide a simple test to make sure calculations are proceeding as they ought to.

The preliminary basis having these properties that is used consists of the generalized Gell–Mann matrices:

$$\Lambda^{jk} = \begin{cases} |j\rangle\langle k| + |k\rangle\langle j|, & 1 \leq k < j \leq d \\ -i|j\rangle\langle k| + i|k\rangle\langle j|, & 1 \leq j < k \leq d \\ \sqrt{\frac{2}{k(k+1)}} \left(\sum_{l=1}^{k} |l\rangle\langle l| - |k\rangle\langle k| \right), & 1 \leq j = k < d \\ I, & j = k = d \end{cases}$$

I have toyed around with building a custom basis to make the coupling operator sparse by applying orthogonal transformations to the normalized version of this basis, but since that appears to have little effect I believe I will simply use this basis for the time being. This basis as I have written it is orthogonal, but not normalized:

$$\mathrm{Tr}[\Lambda^{jk}\Lambda^{mn}] = \delta_{jm}\delta_{kn}\big(2 + \delta_{jd}\delta_{kd}(d-2)\big)$$

The density operator and coupling operator are vectorized in the following manner:

$$\rho = \sum_{j,k} \rho_{jk}\Lambda^{jk}, \qquad \rho_{jk} \in \mathbb{R} \tag{2.1}$$

$$c = \sum_{j,k} c_{jk}\Lambda^{jk}, \qquad c_{jk} \in \mathbb{C} \tag{2.2}$$

2.1 Matrix representations of superoperators

We can write the unconditional vacuum master equation $d\rho/dt = c\rho c^\dagger - \frac{1}{2}(c^\dagger c\rho + \rho c^\dagger c)$ as a system of coupled first-order ordinary differential equations:

$$\mathrm{Tr}[\Lambda^{jk}\Lambda^{jk}]\frac{\mathrm{d}\rho_{jk}}{\mathrm{d}t} = \sum_{p,q}\rho_{pq}\left(\sum_{m,n}|c_{mn}|^2\,\mathrm{Tr}\left[\Lambda^{jk}\left(\Lambda^{mn}\Lambda^{pq}\Lambda^{mn} - \frac{1}{2}(\Lambda^{mn}\Lambda^{mn}\Lambda^{pq} + \Lambda^{pq}\Lambda^{mn}\Lambda^{mn})\right)\right] + \right. \tag{2.3}$$

$$\left.\sum_{dm+n<dr+s} 2\Re\left\{c_{mn}c_{rs}^*\,\mathrm{Tr}\left[\Lambda^{jk}\left(\Lambda^{mn}\Lambda^{pq}\Lambda^{rs} - \frac{1}{2}(\Lambda^{rs}\Lambda^{mn}\Lambda^{pq} + \Lambda^{pq}\Lambda^{rs}\Lambda^{mn})\right)\right]\right\}\right) \tag{2.4}$$

This means I can write the vectorized version of the equation, using single indices $w = dr+s$, $x = dj+k$, $y = dp+q$, and $z = dm+n$ for $\vec{\rho}$:

$$\frac{d\vec{\rho}}{dt} = D(\vec{c})\vec{\rho}$$

The matrix $D(\vec{c})$ has entries:

$$D_{xy}(\vec{c}) = (\mathrm{Tr}[\Lambda^x\Lambda^x])^{-1}\left(\sum_z |c_z|^2\,\mathrm{Tr}[\Lambda^x(\Lambda^z\Lambda^y\Lambda^z - \frac{1}{2}(\Lambda^z\Lambda^z\Lambda^y + \Lambda^y\Lambda^z\Lambda^z))] + \right. \tag{2.5}$$

$$\left.\sum_{z>w} 2\Re\left\{c_z c_w^*\,\mathrm{Tr}[\Lambda^x(\Lambda^z\Lambda^y\Lambda^w - \frac{1}{2}(\Lambda^w\Lambda^z\Lambda^y + \Lambda^y\Lambda^w\Lambda^z))]\right\}\right) \tag{2.6}$$

In a similar way we can calculate:

$$\rho' = \frac{M}{2}[c,[c,\rho]] + \frac{M^*}{2}[c^\dagger,[c^\dagger,\rho]]$$

in the vectorized form:

$$\vec{\rho}' = E(M,\vec{c})\vec{\rho}$$

where $E(M,\vec{c})$ has entries:

$$E_{xy}(M,\vec{c}) = 2(\mathrm{Tr}[\Lambda^x\Lambda^x])^{-1}\left(\sum_{w<z}\Re\{M^*c_wc_z\}\Re\{\mathrm{Tr}[\Lambda^x(\Lambda^w\Lambda^z\Lambda^y + \Lambda^y\Lambda^w\Lambda^z) - 2\Lambda^x\Lambda^w\Lambda^y\Lambda^z]\} + \right. \tag{2.7}$$

$$\left.\sum_w \Re\{M^*c_w^2\}(\Re\{\mathrm{Tr}[\Lambda^x\Lambda^w\Lambda^w\Lambda^y]\} - \mathrm{Tr}[\Lambda^x\Lambda^w\Lambda^y\Lambda^w])\right) \tag{2.8}$$

If I vectorize the plant Hamiltonian:

$$H = \sum_z h_z\Lambda^z, \qquad h_z \in \mathbb{R} \tag{2.9}$$

I can then calculate:

$$\frac{d\rho}{dt} = -i[H,\rho]$$

in the vectorized form:

$$\frac{d\vec{\rho}}{dt} = F(\vec{h})\vec{\rho}$$

where $F(\vec{h})$ has entries:

$$F_{x,y}(\vec{h}) = (\mathrm{Tr}[\Lambda^x\Lambda^x])^{-1}\sum_z h_z\,\Im\{\mathrm{Tr}[\Lambda^x(\Lambda^z\Lambda^y - \Lambda^y\Lambda^z)]\}$$

2.2 Nonlinear superoperator representation

The stochastic expression:

$$d\rho = dW\left(c\rho + \rho c^\dagger - \rho\,\mathrm{Tr}[(c+c^\dagger)\rho]\right)$$

can be calculated:

$$d\vec{\rho} = dW(G + \vec{k}\cdot\vec{\rho})\vec{\rho}$$

where we define:

$$G_{x,y} = 2\left(\mathrm{Tr}[\Lambda^x\Lambda^x]\right)^{-1}\sum_z \Re\left\{c_z\,\mathrm{Tr}[\Lambda^x\Lambda^z\Lambda^y]\right\} \tag{2.10}$$

$$k_x = -2\Re\{c_x\}\,\mathrm{Tr}[\Lambda^x\Lambda^f]$$

CHAPTER 3

SME integration

In order to integrate a stochastic equation:

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

if one wants to be more sophisticated than Euler integration, one can use Milstein integration:

$$X_{i+1} = X_i + a(X_i,t_i)\Delta t_i + b(X_i,t_i)\Delta W_i + \frac{1}{2}b(X_i,t_i)\frac{\partial}{\partial X}b(X_i,t_i)\left((\Delta W_i)^2 - \Delta t_i\right)$$

3.1 Vector Milstein

What if we are interested in a vector-valued equation:

$$d\vec{\rho} = \vec{a}(\vec{\rho},t)dt + \vec{b}(\vec{\rho},t)dW$$

The way to generalize the Milstein scheme (while still restricting ourselves to a scalar-valued Wiener process) is

$$\rho^\mu_{i+1} = \rho^\mu_i + a^\mu_i\Delta t_i + b^\mu_i\Delta W_i + \frac{1}{2}b^\nu_i\partial_\nu b^\mu_i\left((\Delta W_i)^2 - \Delta t_i\right),$$

where I have adopted an index convention for vectors such that

$$\vec{\rho} = \rho^\mu\hat{e}_\mu \tag{3.1}$$
$$a^\mu_i = a^\mu(\vec{\rho}(t_i),t_i) \tag{3.2}$$
$$\partial_\nu = \frac{\partial}{\partial\rho^\nu} \tag{3.3}$$

and indices that appear in both upper and lower positions in the same term are implicitly summer over.

For $b^\mu = G^\mu_\nu\rho^\nu + k_\nu\rho^\nu\rho^\mu$ as defined in Vectorization we can write:

$$b^\nu\partial_\nu b^\mu = \left(k_\nu G^\nu_\sigma\rho^\mu + G^\mu_\nu G^\nu_\sigma + 2k_\nu\rho^\nu(G^\mu_\sigma + k_\sigma\rho^\mu)\right)\rho^\sigma \tag{3.4}$$
$$b^\nu\partial_\nu b^\mu\hat{e}_\mu = \left(\left(\vec{k}^\mathsf{T}G\vec{\rho}\right) + G^2 + 2(\vec{k}\cdot\vec{\rho})\left(G + \vec{k}\cdot\vec{\rho}\right)\right)\vec{\rho} \tag{3.5}$$

3.2 Order 1.5 Taylor scheme

To get higher order convergence in time, we can use a more complicated update formula (restricting ourselves to $\vec{a}$ and $\vec{b}$ with no explicit time dependence, as we have in our problem):

$$\rho_{i+1}^{\mu} = \rho_i^{\mu} + a_i^{\mu}\Delta t_i + b_i^{\mu}\Delta W_i + \frac{1}{2}b_i^{\nu}\partial_{\nu}b_i^{\mu}\left((\Delta W_i)^2 - \Delta t_i\right) + \tag{3.6}$$

$$b_i^{\nu}\partial_{\nu}a_i^{\mu}\Delta Z_i + \left(a_i^{\nu}\partial_{\nu} + \frac{1}{2}b_i^{\nu}b_i^{\sigma}\partial_{\nu}\partial_{\sigma}\right)b_i^{\mu}\left(\Delta W_i\Delta t_i - \Delta Z_i\right) \tag{3.7}$$

$$\frac{1}{2}\left(a_i^{\nu}\partial_{\nu} + \frac{1}{2}b_i^{\nu}b_i^{\sigma}\partial_{\nu}\partial_{\sigma}\right)a_i^{\mu}\Delta t_i^2 + \frac{1}{2}b_i^{\nu}\partial_{\nu}b_i^{\sigma}\partial_{\sigma}b_i^{\mu}\left(\frac{1}{3}(\Delta W_i)^2 - \Delta t_i\right)\Delta W_i \tag{3.8}$$

Recall from Vectorization that:

$$\vec{a}(\vec{\rho}) = Q\vec{\rho} \tag{3.9}$$

$$Q := (N+1)D(\vec{c}) + ND(\vec{c}^*) + E(M,\vec{c}) + E(M,\vec{c}^*) \tag{3.10}$$

ΔZ is a new random variable related to ΔW:

$$\Delta W_i = U_{1,i}\sqrt{\Delta t_i} \tag{3.11}$$

$$\Delta Z_i = \frac{1}{2}\left(U_{1,i} + \frac{1}{\sqrt{3}}U_{2,i}\right)\Delta t_i^{3/2} \tag{3.12}$$

where U_1, U_2 are normally distributed random variables with mean 0 and variance 1.

The new terms in the higher-order update formula are given below:

$$b^{\nu}\partial_{\nu}a^{\mu}\hat{e}_{\mu} = QG\vec{\rho} + (\vec{k}\cdot\vec{\rho})Q\vec{\rho} \tag{3.13}$$

$$a^{\nu}\partial_{\nu}b^{\mu}\hat{e}_{\mu} = GQ\vec{\rho} + (\vec{k}\cdot\vec{\rho})Q\vec{\rho} + \left(\vec{k}^{\mathsf{T}}Q\vec{\rho}\right)\vec{\rho} \tag{3.14}$$

$$a^{\nu}\partial_{\nu}a^{\mu}\hat{e}_{\mu} = Q^2\vec{\rho} \tag{3.15}$$

$$b^{\nu}\partial_{\nu}b^{\sigma}\partial_{\sigma}b^{\mu}\hat{e}_{\mu} = G^3\vec{\rho} + 3(\vec{k}\cdot\vec{\rho})G^2\vec{\rho} + 3\left(\vec{k}^{\mathsf{T}}G\vec{\rho} + 2(\vec{k}\cdot\vec{\rho})^2\right)G\vec{\rho} \tag{3.16}$$

$$\left(\vec{k}^{\mathsf{T}}G^2\vec{\rho} + 6(\vec{k}\cdot\vec{\rho})\vec{k}^{\mathsf{T}}G\vec{\rho} + 6(\vec{k}\cdot\vec{\rho})^3\right)\vec{\rho} \tag{3.17}$$

$$b^{\nu}b^{\sigma}\partial_{\nu}\partial_{\sigma}b^{\mu}\hat{e}_{\mu} = 2\left(\vec{k}^{\mathsf{T}}G\vec{\rho} + (\vec{k}\cdot\vec{\rho})^2\right)\left(G\vec{\rho} + (\vec{k}\cdot\vec{\rho})\vec{\rho}\right) \tag{3.18}$$

$$b^{\nu}b^{\sigma}\partial_{\nu}\partial_{\sigma}a^{\mu}\hat{e}_{\mu} \tag{3.19}$$

We explore testing the convergence rates in Testing.

CHAPTER 4

Testing

To test the stochastic integrators, I am taking my cue from Ian Hawke and employing *grid convergence*. There are two points that make using grid convergence checks on stochastic integration slightly less trivial than for ordinary integration.

The first point is that the stochastic integration methods have convergence rates that are given as *expectation values* of the convergence rates for each trajectory. For strong approximation techniques (which are supposed to converge in trajectory), this means that I'll need to calculate the convergence rates for an ensemble of trajectories and take the average in order to compare to the expected convergence rate.

The second point is that I have to use consistent random increments ΔW and ΔZ for each trajectory. I will do this by calculating all my increments for the smallest timestep, and then using those values to construct the corresponding increments for larger timesteps. My integrators also deal in standard-normal random variable U_1 and U_2, so the thing I actually need to construct are the corresponding standard-normal random variables $\tilde{U}_1$ and $\tilde{U}_2$ for the larger time increments.

4.1 Longer stochastic increments

Let's write down what we want first:

$$\tilde{\Delta} = 2\Delta \tag{4.1}$$

$$\Delta W = U_1\sqrt{\Delta} \tag{4.2}$$

$$\Delta \tilde{W} = \tilde{U}_1\sqrt{\tilde{\Delta}} \tag{4.3}$$

$$\Delta Z = \frac{1}{2}\Delta^{3/2}\left(U_1 + \frac{1}{\sqrt{3}}U_2\right) \tag{4.4}$$

$$\Delta \tilde{Z} = \frac{1}{2}\tilde{\Delta}^{3/2}\left(\tilde{U}_1 + \frac{1}{\sqrt{3}}\tilde{U}_2\right) \tag{4.5}$$

Now we will write down how the increments are defined and work out what our new standard-normal random variables are.

$$\Delta_n := \int_{\tau_n}^{\tau_{n+1}} ds \tag{4.6}$$

$$\tilde{\Delta}_n := \int_{\tau_n}^{\tau_{n+2}} ds \tag{4.7}$$

$$= \int_{\tau_n}^{\tau_{n+1}} ds + \int_{\tau_{n+1}}^{\tau_{n+2}} ds \tag{4.8}$$

$$= \Delta_n + \Delta_{n+1} \tag{4.9}$$

$$\Delta W_n := \int_{\tau_n}^{\tau_{n+1}} dW_s \tag{4.10}$$

$$\Delta \tilde{W}_n := \int_{\tau_n}^{\tau_{n+2}} dW_s \tag{4.11}$$

$$= \int_{\tau_n}^{\tau_{n+1}} dW_s + \int_{\tau_{n+1}}^{\tau_{n+2}} dW_s \tag{4.12}$$

$$= \Delta W_n + \Delta W_{n+1} \tag{4.13}$$

$$\Delta Z_n := \int_{\tau_n}^{\tau_{n+1}} \int_{\tau_n}^{s_2} dW_{s_1} ds_2 \tag{4.14}$$

$$\Delta \tilde{Z}_n := \int_{\tau_n}^{\tau_{n+2}} \int_{\tau_n}^{s_2} dW_{s_1} ds_2 \tag{4.15}$$

$$= \int_{\tau_n}^{\tau_{n+1}} \int_{\tau_n}^{s_2} dW_{s_1} ds_2 + \int_{\tau_{n+1}}^{\tau_{n+2}} \int_{\tau_n}^{s_2} dW_{s_1} ds_2 \tag{4.16}$$

$$= \Delta Z_n + \int_{\tau_{n+1}}^{\tau_{n+2}} \int_{\tau_n}^{\tau_{n+1}} dW_{s_1} ds_2 + \int_{\tau_{n+1}}^{\tau_{n+2}} \int_{\tau_{n+1}}^{s_2} dW_{s_1} ds_2 \tag{4.17}$$

$$= \Delta Z_n + \Delta_{n+1} \Delta W_n + \Delta Z_{n+1} \tag{4.18}$$

We will assume equal time intervals, so $\Delta_n = \Delta$. We start by assuming we are simulating ΔW_n and ΔZ_n by the independent standard-normal random variables $U_{1,n}$ and $U_{2,n}$ using the expressions

$$\Delta W_n = U_{1,n} \sqrt{\Delta} \tag{4.19}$$

$$\Delta Z_n = \frac{1}{2} \Delta^{3/2} \left(U_{1,n} + \frac{1}{\sqrt{3}} U_{2,n} \right) \tag{4.20}$$

Now we want to derive expressions for the new independent standard-normal random variables $\tilde{U}_{1,n}$ and $\tilde{U}_{2,n}$. Start by looking at $\Delta \tilde{W}_n$:

$$\Delta \tilde{W}_n = \Delta W_n + \Delta W_{n+1} \tag{4.21}$$

$$= (U_{1,n} + U_{1,n+1}) \sqrt{\Delta} \tag{4.22}$$

$$= \frac{U_{1,n} + U_{1,n+1}}{\sqrt{2}} \sqrt{\tilde{\Delta}} \tag{4.23}$$

This tells us

$$\tilde{U}_{1,n} = \frac{U_{1,n} + U_{1,n+1}}{\sqrt{2}} \,. \tag{4.24}$$

It is easy to verify this is a standard-normal random variable.

Now look at $\Delta\tilde{Z}_n$:

$$\Delta\tilde{Z}_n = \Delta Z_n + \Delta\Delta W_n + \Delta Z_{n+1} \tag{4.25}$$

$$= \frac{1}{2}\Delta^{3/2}\left(U_{1,n} + \frac{1}{\sqrt{3}}U_{2,n}\right) + \Delta U_{1,n}\sqrt{\Delta} + \frac{1}{2}\Delta^{3/2}\left(U_{1,n+1} + \frac{1}{\sqrt{3}}U_{2,n+1}\right) \tag{4.26}$$

$$= \frac{1}{2}\Delta^{3/2}\left(3U_{1,n} + U_{1,n+1} + \frac{1}{\sqrt{3}}\left(U_{2,n} + U_{2,n+1}\right)\right) \tag{4.27}$$

$$= \frac{1}{2}\tilde{\Delta}^{3/2}\frac{1}{2\sqrt{2}}\left(2(U_{1,n} + U_{1,n+1}) + U_{1,n} - U_{1,n+1} + \frac{1}{\sqrt{3}}(U_{2,n} + U_{2,n+1})\right) \tag{4.28}$$

$$= \frac{1}{2}\tilde{\Delta}^{3/2}\left(\frac{U_{1,n} + U_{1,n+1}}{\sqrt{2}} + \frac{1}{\sqrt{3}}\left(\frac{\sqrt{3}}{2}\frac{U_{1,n} - U_{1,n+1}}{\sqrt{2}} + \frac{1}{2}\frac{U_{2,n} + U_{2,n+1}}{\sqrt{2}}\right)\right) \tag{4.29}$$

This tells us

$$\tilde{U}_{2,n} = \frac{\sqrt{3}}{2}\frac{U_{1,n} - U_{1,n+1}}{\sqrt{2}} + \frac{1}{2}\frac{U_{2,n} + U_{2,n+1}}{\sqrt{2}}\,. \tag{4.30}$$

Again, it is relatively straightforward to verify that this is another standard-normal random variable and independent of $\tilde{U}_{1,n}$.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

g

s