Manual for SDE Weak Approximation Library (version 1.0) [1]

Mariko Ninomiya[2]

December 7, 2011

[1]This research was partly supported by Center for Advanced Research in Finance at Graduate School of Economics, the University of Tokyo and by the Grant-in-Aid for Young Scientists(B) of Japan Society for the Promotion of Science, 23730297, 2011.

[2]Graduate School of Economics, the University of Tokyo

Contents

Chapter 1

About SDE Weak Approximation Library (version 1.0)

SDE Weak Approximation Library (SDE_WA) is a numerical library for researchers and practitioners who are interested in proceeding weak approximation of stochastic differential equations (SDEs) in their fields such as mathematical finance.

Two types of algorithms ([4][5]) for implementation of the approximation scheme which we call the Kusuoka scheme in this manual were successfuly constructed to attain second-order weak approximation. Improvement or extension of these algorithms have been recently researched in many fields. For mathematical explanation of the theory on which these algorithms are based, refer to [2], [3], and [4].

Three types of algorithms of weak approximation are available in SDE_WA : two algorithms of the Kusuoka approximation and the Euler–Maruyama scheme which is one of the most popular first-order approximation scheme. Any of the three algorithms can be applicable through SDE_WA if we give the definition of an SDE.

for more details.

You should have received a copy of the GNU Lesser General Public License version 2.1 along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, US

Chapter 2

Introduction

We consider weak approximation of SDEs, that is, calculation of $E\left[f\left(X(T,x)\right)\right]$ where $X(t,x) = \left(X^1(t,x), \ldots X^N(t,x)\right)$ is a diffution process denoted by

$$X^j(t,x) = x_j + \int_0^t V_0^j(X(s,x))\,\mathrm{d}s + \sum_{i=1}^{d}\int_0^t V_i^j(X(s,x)) \circ \mathrm{d}B(s), \tag{2.1}$$

for $j = 1, \ldots, N$. Here $V_i \in C_b^\infty(\mathbb{R}^N;\mathbb{R}^N)$, $B^0(t) = t$, $(B^1(t), \ldots, B^d(t))$ is a d-dimensional standard Brownian motion, and $\circ$ denotes Stratonovich integral.

SDE_WA deals with the Euler–Maruyama scheme and the Kusuoka scheme.

2.1 Algorithms

In this section, we introduce the three types of algorithms included in SDE_WA. One is the Euler–Maruyama scheme. The othre two called NV or NN in this manual are based on the Kusuoka approximation.

As seen in [1], if the process X is of the form as (2.1), then $X^j(t,x)$ can be rewritten with the Ito integral as follows:

$$X^j(t,x) = x_j + \int_0^t \tilde{V}_0^j(X(s,x))\,\mathrm{d}s + \sum_{i=1}^{d}\int_0^t V_i^j(X(s,x))\,\mathrm{d}B(s),$$

where

$$\tilde{V}_0^j(X(s,x)) = V_0^j(X(s,x)) + \frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{d} V_i^k(X(s,x))\frac{\partial V_i^j}{\partial x_k}(X(s,x)). \tag{2.2}$$

This relation held in the drifts of an Ito SDE and a Stratonovich SDE is to be important in defining SDE as to be mentioned later.

In the rest part of this manual, we let $[0,T]$ be partitioned into n intervals by $0 = t_0 < t_1 < \cdots < t_{n-1} < t_n = T$ with $\sum_{i=0}^{n-1}(t_{i+1} - t_i) = T$. For this partitioning, Δt_k denotes $t_k - t_{k-1}$.

2.1.1 Order of weak approximation

If there exists a family $\left\{X^\delta(\cdot,\cdot)\right\}_{\delta>0}$ of random variables which has constants K and δ_0 such that for an arbitrary $\delta \in (0,\delta_0)$

$$\left|E[(f(X(T,x)))] - E[f(X^\delta(T,x))]\right| \leq K\delta^p,$$

then the family $\left\{X^\delta(\cdot,\cdot)\right\}_{\delta>0}$ is said to be p-th order weak approximation of X.

2.1.2 Euler–Maruyama scheme

The Euler–Maruyama scheme (EM) is well-known as a first-order weak approximation ([1]) scheme implemented by a very simple algorithm represented by the following random variables: for $k = 1,\ldots,n$

$$\begin{aligned} X_0^{(\mathrm{EM}),n} &= x, \\ X_{t_k}^{(\mathrm{EM}),n} &= X_{t_{k-1}}^{(\mathrm{EM}),n} + \Delta t_k \tilde{V}_0\left(X_{t_{k-1}}^{(\mathrm{EM}),n}\right) + \sqrt{\Delta t_k}\sum_{i=1}^{d} V_i\left(X_{t_{k-1}}^{(\mathrm{EM}),n}\right) Z_k^i \end{aligned} \tag{2.3}$$

where $\tilde{V}_0$ denotes a drift in an Ito SDE. Here Z_k's are n independent d-dimensional random variables distributed as $N(0,1)$.

2.1.3 Kusuoka scheme

The Kusuoka scheme is one of the higher-order weak approximation schemes. Two kinds of algorithms for implementation of this scheme are considered in SDE_WA. They are developped by Ninomiya–Victoir (NV) [5] and Ninomiya–Ninomiya (NN) [4].

Notation 1. $\exp(V)x$ *denotes the solution at time 1 of the ODE*

$$\frac{\mathrm{d}z_t}{\mathrm{d}t} = V(z_t), \quad z_0 = x.$$

Now we introduce the algorithms of the Kusuoka scheme.

1. The NV alrorithm is defined by a family of random variables defined by

$$X_0^{(\mathrm{NV}),n} := x$$
$$X_{t_k}^{(\mathrm{NV}),n} :=$$
$$\begin{cases} \exp\left(\frac{\Delta t_k V_0}{2}\right)\exp\left(\sqrt{\Delta t_k}Z_k^1 V_1\right)\exp\left(\sqrt{\Delta t_k}Z_k^2 V_2\right)\cdots\exp\left(\sqrt{\Delta t_k}Z_k^d V_d\right)\exp\left(\frac{\Delta t_k V_0}{2}\right)X_{t_{k-1}}^{(\mathrm{NV}),n}, \\ \qquad\qquad \text{if } \Lambda_k = +1 \\ \exp\left(\frac{\Delta t_k V_0}{2}\right)\exp\left(\sqrt{\Delta t_k}Z_k^d V_d\right)\exp\left(\sqrt{\Delta t_k}Z_k^{d-1} V_{d-1}\right)\cdots\exp\left(\sqrt{\Delta t_k}Z_k^1 V_1\right)\exp\left(\frac{\Delta t_k V_0}{2}\right)X_{t_{k-1}}^{(\mathrm{NV}),n}, \\ \qquad\qquad \text{if } \Lambda_k = -1, \end{cases} \tag{2.4}$$

   for $k = 1, \ldots, n$ where (Λ_k, Z_k)'s are n-independent random variables such that Λ_k is a Bernoulli random variable independent of $Z_k \sim N_d(0,1)$

2. The NN algorithm is represented by the following random variables: for $k = 1, \ldots, n$

$$X_0^{(\mathrm{NN}),n} := x,$$
$$X_{t_k}^{(\mathrm{NN}),n} := \exp\left(\frac{\Delta t_k}{2}V_0 + \sum_{i=1}^{d}\sqrt{\Delta t_k}S_{1,k}^i V_i\right)\exp\left(\frac{\Delta t_k}{2}V_0 + \sum_{i=1}^{d}\sqrt{\Delta t_k}S_{2,k}^i V_i\right)X_{t_{k-1}}^{(\mathrm{NN}),n} \tag{2.5}$$

   where $\left(S_{j,k}^i\right)_{\substack{i\in\{1,\ldots,d\},j\in\{1,2\}\\ k\in\{1,\ldots,n\}}}$ are constructed by

$$\begin{pmatrix} S_{1,k}^i \\ S_{2,k}^i \end{pmatrix} = \begin{pmatrix} 1/2 & 1/\sqrt{2} \\ 1/2 & -1/\sqrt{2} \end{pmatrix}\begin{pmatrix} \eta_{1,k}^i \\ \eta_{2,k}^i \end{pmatrix}, \quad \text{where } \eta_{j,k}^i \overset{i.i.d.}{\sim} N(0,1). \tag{2.6}$$

Remark 1. *In the NV and the NN algorithms, Stratonovich SDEs are considered. Therefore, in use of these algorithms, an Ito SDE has to be converted to the corresponding Stratonovich SDE.*

This procedure can be done automatically through SDE_WA by giving some functions corresponding to $\frac{\partial V_i(y)}{\partial y}$*'s as you will see later.*

Remark 2. *For approximation of* $\exp(W)\cdot$ *in the NN algorithm where* $W = \frac{\Delta t_k}{2} V_0 + \sum_{i=1}^{d} \sqrt{\Delta t_k} S^i_{j,k} V_i$*,* $j = 1, 2$*, the Runge–Kutta method is applied. Also, in the case that there does not exist an explicit form of* $\exp(sV_i)\, y$ *in the NV algorithm, it should be approximated by the Runge–Kutta method, too. For more explanation, refer to the next section.*

These three types of implementation algorithms (EM, NV, and NN) of weak approximation of SDEs are included in SDE_WA.

2.2 Use of explicit forms or application of numerical integrator

SDE_WA has the following capabilities:

- numerical calculator of Ito–Stratonovich conversion
- 5-th or 7-th order "numerical integrator" for $\exp(W)y$ where $W \in C_b^\infty(\mathbb{R}^N; \mathbb{R}^N)$.

The Runge–Kutta method plays a role of "numerical integrator" in this lirary.

2.2.1 Ito–Stratonovich conversion

As we can see in (2.3), the EM algorithm is based on an Ito SDE. Hence, if the SDE is written in Stratonovich form, the drift term V_0 has to be converted to $\tilde{V}_0$ by (2.2). **This procedure is automatically done by SDE_WA as long as the the following information of the SDE is given by users:**

- the type of the SDE (Stratonovich)
- definitions of functions for $\frac{\partial V_1(y)}{\partial y}, \frac{\partial V_2(y)}{\partial y}, \ldots \frac{\partial V_d(y)}{\partial y}$.

If the considered SDE is written in Ito form, then the second information above ($\left(\frac{\partial V_1(y)}{\partial y}, \frac{\partial V_2(y)}{\partial y}, \ldots, \frac{\partial V_d(y)}{\partial y}\right)$) is not needed and would not be used even if it is given.

For the NN algorithm, simply contrary argument done for the EM algirthm holds, which means that for an Ito SDE, Ito–Stratonovich conversion should be automatically done by SDE_WA if the type of the SDE (Ito) and the definitions of $\frac{\partial V_1(y)}{\partial y}$, $\frac{\partial V_2(y)}{\partial y}$, $\ldots$ $\frac{\partial V_d(y)}{\partial y}$ are given. Also, if the SDE is defined in Stratonovich form, then the functions for $\left(\frac{\partial V_1(y)}{\partial y}, \frac{\partial V_2(y)}{\partial y}, \ldots, \frac{\partial V_d(y)}{\partial y}\right)$ are not needed and would not be used even if given.

There are more possibilities for the NV algorithm. Calculation of $\exp(sV_i)\, y$ is itterated for $i = 0, 1, \ldots, d$ in the NV algorithm. Each $\exp(sV_i)\, y$ denotes the solution at time s of the correspoinding ODE. In SDE_WA, users can choose the way of calculation of $\exp(sV_i)\, y$ from use of explicit forms or application of numerical integration. We give more explanation below about possible situations and choices.

From the simplicity of the form, it could be easy to find an explicit form of $\exp(sV_i)\, y$. The advantages of using explicit forms of $\exp(sV_i)\, y$ are:

- problems related to singularities caused by numerical methods (e.g. violation of the domain) can be avoided
- calculation can be speeded up.

If explicit forms do not exist or are not to be used for some reason, then numerical integration is to be applied to approximation of such $\exp(sV_i)\, y$'s as in the NN algorithm.

In SDE_WA, there are the following rules for approximation of $\exp(sV_i)$:

1. If the definition of the explicit form of $\exp(sV_i)\, y$ is given, then it is certainly used (i.e. numerical integration is not applied.)
2. Whichever the SDE is given in Ito or Stratonovich form, in giving the definition a function for $\exp(sV_i)\, y$, users have to obtain the explicit form of it by themselves. Hence, $\tilde{V}_0$ (Ito drift) has to be manually converted to V_0 (Stratonovich drift) by users in the process if the considered SDE is written in Ito form.
3. For $\exp(sV_i)\, y$ whose explicit form is not given, numerical integration is applied to approximation of it. In particular, if $i = 0$ and the

SDE is given in Ito form, then Ito–Stratonovich conversion would be automatically done. In this case, users must give the functions corresponding to $\frac{\partial V_1(y)}{\partial y}, \ldots, \frac{\partial V_d(y)}{\partial y}$.

2.2.2 Romberg extrapolation

Suppose that we have a p-th order scheme such that there exists a constant $K_f > 0$ satisfying that for a smooth function f

$$\left| E\left[f\left(X_T^{(\mathrm{ord}p),n}\right)\right] - E\left[f\left(X(T,x)\right)\right] - K_f \frac{1}{n^p} \right| \leq C_f \frac{1}{n^{p+1}}.$$

Then,

$$\frac{2^p}{2^p-1} E\left[f\left(X_T^{(\mathrm{ord}p),2n}\right)\right] - \frac{1}{2^p-1} E\left[f\left(X_T^{(\mathrm{ord}p),n}\right)\right]$$

attains $p+1$-st weak approximation.

Remark 3. *Relation between the order m of the Runge–Kutta method used in the NV and the NN algorithms and the order p of the weak approximation of SDEs is $m = 2p + 1$ because the order of approximation of ODEs becomes half in approximation of SDEs ([2]).*

Therefore $m = 5$ for the second-order approximation such as the naive NV and NN algorithms and $m = 7$ when the Romberg extrapolation is applied to these from $2(2 + 1) + 7 = 7$.

Let p be the order of approximation of SDEs and m the order of the Runge–Kutta method applied in the process of the NV or the NN algorithms.

$m = 2p + 1$ always has to hold if the NN algorithm is taken (Table 2.2).

For the EM algorithm, m does not even appear, that is, m can take any value (Table 2.1). (You will see later why we mention the case of EM here.)

In application of the NV algorithm, there are more possibilities than in the other two.

(i) If explicit forms of all $\exp(sV_i)\,y$'s and $\exp(sV_0)\,y$ are given, we do not need to care about m (Table 2.3) because of the rule we have seen in the previous section.

(ii) For such $\exp\left(\sqrt{\Delta t} Z^i V_i\right) y$ or $\exp\left(\frac{\Delta t V_0}{2}\right) y$ that its explicit form is not given, then it has to be approximated by the Runge–Kutta method of order $m = 2p + 1$ (Table 2.4).

	p	m
no Rom.	1	-
with Rom.	2	-

Table 2.1: EM

	p	m
no Rom.	2	5
with Rom.	3	7

Table 2.2: NN

	p	m
no Rom.	2	-
with Rom.	3	-

Table 2.3: NV:(i)

	p	m
no Rom.	2	5
with Rom.	3	7

Table 2.4: NV:(ii)

In SDE_WA, both the 5-th and 7-th order Runge–Kutta methods are equipped.

Chapter 3

Error handling

Chapter 4

Definitions of data types and functions

All data types and functions introduced here are declared in the header file <sde_wa.h>.

4.1 Defining SDE system and one-step calculator

The considered SDE and related information which we call an SDE system are defined using SDE_WA_SYSTEM data type.

–Data Type : SDE_WA_SYSTEM
This data type defines an SDE with arbitrary parameters by having the following members:

- enum SDE_type sde_type;
  enum SDE_type is defined by

  ```
  enum SDE_type {ITO=0, STR=1};
  ```

  to denote the form of the considered SDE.

- int (**V)(const double y[], double dy[], void *params);
  This is a pointer to an array of functions that store the vector-valued $V_i(y)$ in the vector dy, for arguments y(initial vector) and parameters

`params`.

- `int (**drift_corrector)(const double y[], double *dVdy[], void *params);`
  This is a pointer to an array of functions corresponding to $\frac{\partial V_0(y)}{\partial y}$, $\frac{\partial V_1(y)}{\partial y}$, ..., $\frac{\partial V_d(y)}{\partial y}$. Each `drift_correctorp[`$i$`]` stores $\frac{\partial V_i^{j+1}(y)}{\partial y_{k+1}}$ in `dVdy[`j`][`k`]` for $y =$`y` with parameters `params`.

  It should be remarked that this array has $(1 + d)$ function pointers whose first element (function)`drift_corrector[0]` is allocated for $\frac{\partial V_0(y)}{\partial y}$ which is not to be used. Hence, we can let `drift_corrector[0]=NULL`.

- `int (**exp_sV)(double s, const double y[], double exp_sVy[], void *params);`
  This is a pointer to an array of functions that store the vector of explicit forms of $\exp(sV_i)y$'s for $i = 0, 1, ..., d$ of the ODE

$$\frac{dz_t}{dt} = V_i(z_t), \quad z_0 = y$$

  in the vector `exp_sVy`, for arguments `s`, `y`, and parameters `params`. Remark that $\exp(sV_i)\, y$ should be obtained for a Stratonovich SDE.

  Any functions appearing above could be `NULL` when they do not exist or are not needed.

- `int dim_y;`
  This is the spacial dimension of the system of equations. `dim_y` corresponds to N of (2.1)

- `int dim_BM;`
  This is the dimension of the Browninan motion. `dim_BM` corresponds to d in (2.1)

- `void *params;`
  This is a pointer to arbitrary parameters of the system.

Memory for a one-step calculator should be allocated once and reused for itterative calculation in simulation. SDE_WA_SLTN data type describes a one-step calculator.

–Data Type:SDE_WA_SLTN
This data type describes a one-step calculator object determined by an algorithm, an SDE, parameters, and the required order of numerical integration (even if not used).

Though this data type includes various information as seen below, users only consider alg, mth_is, and sde in programming. However, users have to know data types of a sample point for EM, NV, and NN because these will be directly defined by users in programs.

- enum ALG alg;
  enum ALG is defined as follows :

  ```
  enum ALG {E_M=0, N_V=1, N_N=2};
  ```

- int mth_is;
  This integer denotes the order of numerical integration (the Runge–Kutta method) used in the NV or the NN algorithm. It can be 5 or 7 when the order of the scheme is 2 or 3 respectively. In the EM scheme, mth_is has to be set to any of 5 and 7 though it is never used.

- SDE_WA_SYSTEM *sde;
  This is a pointer to the considered SDE system.

- int (*one_step)(struct sde_wa_sltn *sl, double s);
  This is a function determined for alg. This function should store the result of one-step approximation with a time interval s.

- double *initv;
  This is a pointer to the array storing the initial vector for one-step approximation.

- double *destv;
  This is a pointer to the array to store the result of one-step approximation.

- `double *drift_step_interv;`
  This is a pointer to the array used in the process of converting the drift term or keeping intermidiate values in one-step approximation.

- `double **drift_corretor_matrix;`
  This is a pointer to the array used in the process of conversion between a Stratonovich-form SDE and an Ito-form SDE.

- ```
  union{
     double *em;
     RV_NV *nv;
     double *nn;
  }sample_pt;
  ```
  This is a pointer to a sample point determined by the considered `alg`. Here `RV_NV` is a data type defined for the NV algorithm as follows:

  ```
  enum Bernoulli_rv {T=0, H=1};
  typedef struct{
     enum Bernoulli_rv rv_nv_b;
     double *rv_nv_n;
  } RV_NV;
  ```

  For the EM shceme, `em[0]`, `em[1]`,..., `em[d]` get values of $Z_k^1, \ldots, Z_k^d$, respectively. For the NV algorithm, `nv->rv_nv_b` gets `H` or `T` which correspond to 1 or -1, respectively. Also, `nv->rv_nv_n[0]`, `nv->rv_nv_n[1]`,..., `nv->rv_nv_n[d-1]` get values of $Z_k^1, \ldots, Z_k^d$, respectively. For the NN algorithm, `nn[0]`,..., `nn[d-1]`, `nn[d]`,..., `nn[2d-1]` get values of $\eta_{1,k}^1, \eta_{1,k}^2, \ldots, \eta_{1,k}^d, \eta_{2,k}^1, \eta_{2,k}^2, \ldots, \eta_{2,k}^d$, respectively.(Correspondence can be found in Table 4.1.)

- `double *rk_step_interv;`
  This is a pointer to the array used in the process of the Runge–Kutta method in the NV or the NN algorithgm.

- `double *nn_sample_pt_interv;`
  This is a pointer to the array used in the process of conversion of a normally distributed sample point to a sample point satisfying (2.6). This process is conducted only in the NN algorithm.
  ```

4.2 Allocating/instantiating and freeing objects

A great number of calculations of (2.3), (2.4), or (2.5) is normally itterated in simulation. We give functions to allocate memory of `SDE_WA_SYSTEM` and `SDE_WA_SLTN` data types to be reused. The functions to free the allocated memory after itterative calculation are also given below.

–Function: `SDE_WA_SYSTEM *alloc_SDE_WA_SYSTEM(int N, int d, void *params);`
This function returns a pointer to a newly allocated memory of `SDE_WA_SYSTEM` data type for an `N`-dimensional SDE with a `d`-dimensional Brownian motion and parameters `params`.

–Function: `void free_SDE_WA_SYSTEM(SDE_WA_SYSTEM *sys);`
This function frees all the memory associated with an `SDE_WA_SYSTEM` data type object.

–Function: `SDE_WA_SLTN *alloc_SDE_WA_SLTN (enum ALG alg, int mth_is, SDE_WA_SYSTEM *sde);`
This function returns a pointer to a newly allocated memory of one-step calculator for the algorithm `alg` with `sde` and the (`mth_is`)-th-order Runge–Kutta method if the NV or the NN algorithm. The object is instantiated at the same time.
–Function: `void free_SDE_WA_SLTN(SDE_WA_SLTN *sltn);`
This function frees all the memory associated with an `SDE_WA_SLTN` data type object.

Remark 4. *Since* `free_SDE_WA_SLTN` *uses* `sltn->sde->dim_BM` *in freeing memory, we must* **NOT** *free* `SDE_WA_SYSTEM *sde` *until free* `SDE_WA_SLTN *sltn`.

4.3 One-step calculation

Here "one-step" means getting $X^{(*)}_{t_{(k+1)/n}}$ from $X^{(*)}_{t_{k/n}}$.

–Function: `int next_SDE_WA(SDE_WA_SLTN *X, double s, double y[], double dy[], void *rv);`
One-step calculation with initial vector `y`, a time interval `s`, and a sample

point `rv` should be done through this function. The result of the calculation is to be stored in `dy[0]`, `dy[1]`,... `dy[`$N-1$`]`.

`rv` has to be appropriately defined depending on each algorithm as explained in the previous chapter (also see Table 4.1).

`alg`	data type of `rv`	correspondence($i = 0, \ldots, d-1$)
`E_M`	`double *`	`rv[`i`]`$= Z_k^{i+1}$
`N_V`	`RV_NV *`	`rv.rv_nv_b`$= \Lambda_k$ `rv.rv_nv_n[`i`]`$= Z_k^{i+1}$
`N_N`	`double *`	`rv[`i`]`$= \eta_{1,k}^{i+1}$ `rv[`$d+i$`]`$= \eta_{2,k}^{i+1}$

Table 4.1: data types of `rv`

Chapter 5

Examples

5.1 Asian option under Heston SV model

The Heston model is well-known in finance as a two-factor stochastic volatility model written in Ito form by

$$\begin{aligned} X^1(t,x) =& x_1 + \int_0^t \mu X_1(s,x)\,\mathrm{d}s + \int_0^t X_1(s,x)\sqrt{X_2(s,x)}\,\mathrm{d}B^1(s), \\ X^2(t,x) =& x_2 + \int_0^t \alpha\left(\theta - X_2(s,x)\right)\,\mathrm{d}s \\ &+ \int_0^t \beta\sqrt{X_2(s,x)}\left(\rho\, dB^1(s) + \sqrt{1-\rho^2}\,\mathrm{d}B^2(s)\right) \end{aligned} \tag{5.1}$$

where $x = (x_1, x_2) \in (\mathbb{R}_{>0})^2$, $(B^1(t), B^2(t))$ is a two-dimensional standard Brownian motion, $-1 \le \rho \le 1$, and α, θ, μ are some positive coefficients such that $2\alpha\theta - \beta^2 > 0$ to ensure the existence and uniqueness of a solution to the SDE. We consider an Asian call option on this asset. Then the payoff is $\max\left(X_3(T,x)/T - K, 0\right)$ where

$$X_3(t,x) = \int_0^t X_1(s,x)\,ds. \tag{5.2}$$

Then, $\tilde{V}_0$, V_1, and V_2 become as follows:

$$\tilde{V}_0\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} \mu y_1 \\ \alpha(\theta - y_2) \\ y_1 \end{pmatrix},\ V_1\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_1\sqrt{y_2} \\ \beta\sqrt{y_2}\rho \\ 0 \end{pmatrix},\ V_2\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ \beta\sqrt{y_2(1-\rho^2)} \\ 0 \end{pmatrix}.$$

The arrays of partial derivatives of V_1 and V_2 become

$$\frac{\partial V_1}{\partial y}(y) = \begin{pmatrix} \sqrt{y_2} & y_1/(2\sqrt{y_2}) & 0 \\ 0 & \beta\rho/(2\sqrt{y_2}) & 0 \\ 0 & 0 & 0 \end{pmatrix}, \frac{\partial V_2}{\partial y}(y) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \beta\sqrt{1-\rho^2}/(2\sqrt{y_2}) & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Remark that $\frac{\partial V_i}{\partial y}(y) = \left(a_{jk}\right)_{j,k\in\{1,\dots,N\}}$ with $a_{jk} = \frac{\partial V_i^j}{\partial y_k}(y)$.

In the NV algorithm, explicit forms can be used if they exist. Before seeking them, the Ito SDE has to be converted to the correspoinding Stratonovich SDE as mentioned in Remark 1 and in 2.2.1.

In the case of this example, we obtain

$$V_0\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_1(\mu - y_2/2 - \rho\beta/4) \\ \alpha(\theta - y_2) - \beta^2/4 \\ y_1 \end{pmatrix}. \tag{5.3}$$

Then, there does not exist an explicit form of $\exp(sV_0)y$ while $\exp(sV_1)y$ and $\exp(sV_2)y$ have the following forms:

$$\exp(sV_1)\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_1 \exp\left(s\sqrt{y_2} + \rho\beta s^2/4\right) \\ \left(\rho\beta s/2 + \sqrt{y_2}\right)^2 \\ y_3 \end{pmatrix},$$

$$\exp(sV_2)\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ \left(\frac{s\beta\sqrt{1-\rho^2}}{2} + \sqrt{y_2}\right)^2 \\ y_3 \end{pmatrix}.$$

This is the procedure which should be done by those who take the NV algorithm and prefer to use explicit forms of $\exp(sV_i)\,y$'s for quicker calculation.

Remark 5. *There are two possibilities of improvements in terms of speed of calculation for the NV or the NN algorithm as follows:*

(i) We have obtained V_0 (5.3) from $\tilde{V}_0$. If the SDE system is defined as a Stratonovich SDE with V_0, then it is possible to avoid reiteration of Ito–Stratonovich conversion in the process of numerital integration.

(ii) Though $\exp(sV_0)\,y$ *does not an explicit form, it can be approximated as follows:*

$$\exp(sV_0)\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_1 \exp\left(\left(\mu - \frac{\rho\beta^2}{4} - \frac{J}{2}\right)s + \frac{y_2 - J}{2\alpha}\left(\mathrm{e}^{-\alpha s} - 1\right)\right) \\ J + (y_2 - J)\,\mathrm{e}^{-\alpha s} \\ y_3 + \frac{y_1\left(\mathrm{e}^{As} - 1\right)}{A} \end{pmatrix}, \quad (5.4)$$

where $J = \theta - \frac{\beta^2}{4\alpha}$ *and* $A = \mu - \frac{\rho\beta^2}{4} - \frac{y_2}{2}$.

Since use of explicit forms or closed-form approximation effectively speeds up the calculation for the NV algorithm, it is strongly recommended to pursue them as long as the NV algorithm is considered.

5.1.1 Example of source codes in C : definitions of functions for SDE

Since this model has five parameters α, β, μ, ρ and θ, we construct a data type `struct AH_params` for the set of these parameters as seen in `sde_wa_manual_ah_system.h`. Then we declare (`sde_wa_manual_ah_system.h`) and define (`sde_wa_manual_ah_system.c`) all functions which are to be used for instantiation of an SDE system object.

In this example, we define all functions for an SDE system object because it is often the case that some algorithms can be applied to one common SDE system for the purpose of comparison or some reasons. Such an SDE system should have all the definitions of functions which could be used in any algorithm. Since main source code `sde_wa_manual_ah_nn.c` to be given in 5.1.2 takes the NN algorithm, V_0 is also defined according to Remark5.

It should be noticed that the last function `asian_heston_call_payoff` is the definition of payoff of an Asian call option which is not to be used for instantiation of `SDE_WA_SYSTEM` data type object.

```
**************************************************************

/*
 * sde_wa_manual_program_ah_system.h
 */
/*
```

```
 * $Source: /Users/mariko/lib/source/sde_wa/sde_wa_manual/sde_wa_manual_progr
 * $Revision: 1.2 $
 * $Author: mariko $
 * $Date: 2011/11/29 05:46:46 $
 */

#ifndef _SDE_WA_MANUAL_PROGRAM_AH_SYSTEM_H_
#define _SDE_WA_MANUAL_PROGRAM_AH_SYSTEM_H_

#ifndef M_PIl
#define M_PIl (3.14159265358979323846264338327950l29)
#endif
/* parameters for an asian heston model*/
struct AH_params{
  double alpha;
  double beta;
  double mu;
  double rho;
  double theta;
};

int ah_V_0(const double y[], double dy[], void *params);
int ah_str_V_0(const double y[], double dy[], void *params);
int ah_V_1(const double y[], double dy[], void *params);
int ah_V_2(const double y[], double dy[], void *params);

int diff_ah_V_1(const double y[], double *dVdy[], void *params);
int diff_ah_V_2(const double y[], double *dVdy[], void *params);

int exp_ah_sVy_0(double s, const double y[], double dy[],
                 void *params);
int exp_ah_sVy_1(double s, const double y[], double dy[],
                 void *params);
int exp_ah_sVy_2(double s, const double y[], double dy[],
                 void *params);

/* declaration of f : not to be included in SDE_WA_SYSTEM */
double asian_heston_call_payoff(double x, double K);
```

```
#endif

*****************************************************************

*****************************************************************

/*
 * sde_wa_manual_program_ah_system.c
 */
/*
 * $Source: /Users/mariko/lib/source/sde_wa/sde_wa_manual/sde_wa_manual_progr
 * $Revision: 1.6 $
 * $Author: mariko $
 * $Date: 2011/11/29 06:25:53 $
 */

#include <math.h>
#include <sde_wa.h>
#include "sde_wa_manual_program_ah_system.h"

int ah_V_0(const double y[], double dy[], void *params){

  struct AH_params *pparams;
  pparams=params;

  dy[0]=pparams->mu*y[0];
  dy[1]=pparams->alpha*(pparams->theta-y[1]);
  dy[2]=y[0];

  return SDE_WA_SUCCESS;
}
int ah_str_V_0(const double y[], double dy[], void *params){

  struct AH_params *pparams;
  pparams=params;

  dy[0]=y[0]*(pparams->mu-y[1]/2-pparams->rho*pparams->beta/4);
```

```
  dy[1]=pparams->alpha*(pparams->theta-y[1])-pparams->beta*pparams->beta/4;
  dy[2]=y[0];

  return SDE_WA_SUCCESS;

}
int ah_V_1(const double y[], double dy[], void *params){

  struct AH_params *pparams;
  pparams=params;

  dy[0]=y[0]*sqrt(y[1]);
  dy[1]=(pparams->rho)*(pparams->beta)*sqrt(y[1]);
  dy[2]=0.0;

  return SDE_WA_SUCCESS;
}

int ah_V_2(const double y[], double dy[], void *params){

  struct AH_params *pparams;
  pparams=params;

  dy[0]=0.0;
  dy[1]=(pparams->beta)*sqrt((1.0-(pparams->rho)*(pparams->rho))*y[1]);
  dy[2]=0.0;

  return SDE_WA_SUCCESS;
}

int diff_ah_V_1(const double y[], double *dVdy[], void *params){

  struct AH_params *pparams;
  pparams=params;

  dVdy[0][0]=sqrt(y[1]);
  dVdy[0][1]=(y[0]/sqrt(y[1]))/2.0;
  dVdy[0][2]=0.0;
```

```
  dVdy[1][0]=0.0;
  dVdy[1][1]=pparams->rho*pparams->beta/(sqrt(y[1])*2.0);
  dVdy[1][2]=0.0;

  dVdy[2][0]=0.0;
  dVdy[2][1]=0.0;
  dVdy[2][2]=0.0;

  return SDE_WA_SUCCESS;
}

int diff_ah_V_2(const double y[], double *dVdy[], void *params){

  struct AH_params *pparams;
  pparams=params;

  dVdy[0][0]=0.0;
  dVdy[0][1]=0.0;
  dVdy[0][2]=0.0;

  dVdy[1][0]=0.0;
  dVdy[1][1]=pparams->beta*sqrt(1-pparams->rho*pparams->rho)
    /(sqrt(y[1])*2.0);
  dVdy[1][2]=0.0;

  dVdy[2][0]=0.0;
  dVdy[2][0]=0.0;
  dVdy[2][0]=0.0;

  return SDE_WA_SUCCESS;
}

int exp_ah_sVy_0(double s, const double y[], double dy[],
              void *params){

  double J;
  double A;
```

```
  struct AH_params *pparams;

  pparams=params;
  J=pparams->theta-pparams->beta*pparams->beta/(4.0*pparams->alpha);
  A=pparams->mu-pparams->rho*pparams->beta*pparams->beta/4.0-y[1]/2.0;

  dy[0]=y[0]*exp((pparams->mu-pparams->rho*pparams->beta*pparams->beta/4.0-J/
                 +(y[1]-J)*(exp(-pparams->alpha*s)-1.0)/(2.0*pparams->alpha))
  dy[1]=J+(y[1]-J)*exp(-pparams->alpha*s);
  dy[2]=y[2]+y[0]*(exp(A*s)-1.0)/A;
  return SDE_WA_SUCCESS;

}

int exp_ah_sVy_1(double s, const double y[], double dy[],
              void *params){

  struct AH_params *pparams;
  pparams=params;

  dy[0]=y[0]*exp(s*sqrt(y[1])+pparams->rho*pparams->beta*s*s/4.0);
  dy[1]=(pparams->rho*pparams->beta*s/2.0+sqrt(y[1]))*
    (pparams->rho*pparams->beta*s/2.0+sqrt(y[1]));
  dy[2]=y[2];
  return SDE_WA_SUCCESS;
}

int exp_ah_sVy_2(double s, const double y[], double dy[],
              void *params){


  struct AH_params *pparams;
  pparams=params;

  dy[0]=y[0];
  dy[1]=(s*pparams->beta*sqrt(1-pparams->rho*pparams->rho)/2.0
         +sqrt(y[1]))
    *(s*pparams->beta*sqrt(1-pparams->rho*pparams->rho)/2.0
```

```
      +sqrt(y[1]));
  dy[2]=y[2];
  return SDE_WA_SUCCESS;
}


double asian_heston_call_payoff(double x, double K){
  return (x - K > 0.0)? x - K: 0.0;
}
```

****

5.1.2 Example of a source code in C : main

In the following main source code, the NN algorighm is taken with quasi-Monte Carlo method using a Sobol sequence which is obtainable from the GNU Scientific Library. Also, the Romberg extrapolation is applied, which is indicated by `m_moment=7`. `m_moment` is given to `next_SDE_WA` as the order of the Runge–Kutta method.

Freeing procedure is done according to Remark 4.

****

```
/*
 * sde_wa_manual_program_ah_nn.c
 */
/*
 * $Source: /Users/mariko/lib/source/sde_wa/sde_wa_manual/sde_wa_manual_progr
 * $Revision: 1.7 $
 * $Author: mariko $
 * $Date: 2011/11/29 10:48:07 $
 */

#include <stdlib.h> /* for malloc */
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_qrng.h>
#include <sde_wa.h>
```

```
#include "sde_wa_manual_program_ah_system.h"

int main(void){

  enum ALG alg = N_N;
  int m_moment = 7; /* Romberg */
  int M = 10000;
  int n = 4;
  double x0=1.0;
  double x1=0.09;
  double K = 1.05;

  int neg_event_counter=0;

  struct AH_params ah_params;
  double alpha = 2.0;
  double beta = 0.1;  /** beta of the heston model **/
  double rho = 0.0;
  double theta = 0.09;      /** theta of the heston model **/
  double mu=0.05;


  ah_params.alpha=alpha;
  ah_params.beta=beta;
  ah_params.mu=mu;
  ah_params.rho=rho;
  ah_params.theta=theta;

  {

    SDE_WA_SYSTEM *sde;
    SDE_WA_SLTN *sl;
    double *sp;
    double **x;
    double **xR;
    double sum;
    double dt=1.0/(double)n;
    gsl_qrng *q;
```

```
double *u_seq1, *u_seq2, *n_seq1, *n_seq2;
double rom_weight1=4.0/3.0;
double rom_weight2=1.0/3.0;
int i,k;
double *tmp_pt;

sde=alloc_SDE_WA_SYSTEM(3, 2, &ah_params);
sde->sde_type=STR;

sde->V[0]=ah_str_V_0;
sde->V[1]=ah_V_1;
sde->V[2]=ah_V_2;

sde->drift_corrector[0]=NULL;
sde->drift_corrector[1]=diff_ah_V_1; /* for EM */
sde->drift_corrector[2]=diff_ah_V_2; /* for EM */

sde->exp_sV[0]=exp_ah_sVy_0; /* for NV */
sde->exp_sV[1]=exp_ah_sVy_1; /* for NV */
sde->exp_sV[2]=exp_ah_sVy_2; /* for NV */


sl=alloc_SDE_WA_SLTN(alg, m_moment, sde);
sp=(double *)malloc(sizeof(double)*sde->dim_BM*2);


u_seq1=(double *)malloc(sizeof(double)*(n+n/2)*sde->dim_BM*2);
u_seq2=u_seq1+(n+n/2)*sde->dim_BM;
n_seq1=(double *)malloc(sizeof(double)*(n+n/2)*sde->dim_BM*2);
n_seq2=n_seq1+(n+n/2)*sde->dim_BM;


x=(double **)malloc(sizeof(double *)*2);
for (i=0; i<2; i++) x[i]=(double *)malloc(sizeof(double)*sde->dim_y);

xR=(double **)malloc(sizeof(double *)*2);
for (i=0; i<2; i++) xR[i]=(double *)malloc(sizeof(double)*sde->dim_y);
```

```
/* sobol seq. => seq.dim.<=40*/
q=gsl_qrng_alloc(gsl_qrng_sobol, (n+n/2)*sde->dim_BM*2);


int neg_except_flag=0;
for (neg_except_flag=0, sum=0.0, i=0; i < M; i++){

  gsl_qrng_get(q, u_seq1);

  for (k=0; k<n+n/2; k++){
    n_seq1[k] = sqrt(-2.0*log(u_seq1[k]))*cos(2.0*M_PIl*u_seq2[k]);
    n_seq2[k] = sqrt(-2.0*log(u_seq1[k]))*sin(2.0*M_PIl*u_seq2[k]);
  }/* for k */

  dt=1.0/(double)n;

  for (x[0][0]=x0, x[0][1]=x1, x[0][2]=0.0, k=0; k< n; k++){

    sp[0]=n_seq1[k];
    sp[1]=n_seq1[n+n/2+k];
    sp[2]=n_seq2[k];
    sp[3]=n_seq2[n+n/2+k];

    next_SDE_WA(sl, dt, x[0], x[1], sp);

    if (x[1][1] < 0.0) {
        neg_except_flag=1; break;
    }

    tmp_pt=x[0];
    x[0]=x[1];
    x[1]=tmp_pt;

  }/* for k */

  if (neg_except_flag) {
    i-=1; neg_except_flag=0; neg_event_counter++;
    continue;
```

```
  }


  dt=2.0/(double)n;
  for (xR[0][0]=x0, xR[0][1]=x1, xR[0][2]=0.0, k=0; k< n/2; k++){

    sp[0]=n_seq1[n+k];
    sp[1]=n_seq1[n+n/2+k];
    sp[2]=n_seq2[n+k];
    sp[3]=n_seq2[n+n/2+k];


    next_SDE_WA(sl, dt, xR[0], xR[1], sp);


    if(xR[1][1] < 0.0) {
      neg_except_flag=1; break;
    }

    tmp_pt=xR[0];
    xR[0]=xR[1];
    xR[1]=tmp_pt;

  } /* for k */

  if (neg_except_flag) {
    i-=1; neg_except_flag=0; neg_event_counter++; continue;
  }

  sum+=rom_weight1*asian_heston_call_payoff(x[0][2], K)
    -rom_weight2*asian_heston_call_payoff(xR[0][2], K);


} /* for i */


free_SDE_WA_SLTN(sl);
free_SDE_WA_SYSTEM(sde);
```

```
    printf("\n%.12e\n", sum/(double)M);
  }
  return SDE_WA_SUCCESS;
}
```

****

5.2 Stochastic Area

Stochastic area is defined by

$$A(t) := \frac{1}{2}\left(\int_0^t B^2(s)\mathrm{d}B^1(s) - \int_0^t B^1(s)\mathrm{d}B^2(s)\right) \tag{5.5}$$

where $\left(B^1, B^2\right)$ is a two-dimensional standard Brownian motion.

Then, we consider the three-dimensional SDE written in the Stratonovich form by

$$\begin{aligned} X^1(t,x) &= \int_0^t \mathrm{d}B^1(s) \\ X^2(t,x) &= \int_0^t \mathrm{d}B^2(s) \\ X^3(t,x) &= \frac{1}{2}\int_0^t X^2(s)\mathrm{d}B^1(s) - \frac{1}{2}\int_0^t X^1(s)\mathrm{d}B^2(s). \end{aligned} \tag{5.6}$$

Then, V_0, V_1, and V_2 become as follows:

$$V_0\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix},\ V_1\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \frac{1}{2}y_2 \end{pmatrix},\ V_2\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ -\frac{1}{2}y_1 \end{pmatrix}$$

and the array of derivatives of V_1 and V_2 become

$$\frac{\partial V_1}{\partial y}(y) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1/2 & 0 \end{pmatrix}, \frac{\partial V_2}{\partial y}(y) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1/2 & 0 & 0 \end{pmatrix}.$$

Remark that $\frac{\partial V_i}{\partial y}(y) = \left(a_{jk}\right)_{j,k\in\{1,\ldots,N\}}$ with $a_{jk} = \frac{\partial V_i^j}{\partial y_k}(y)$.

The explicit forms for every V_i appearing here exist.

$$\exp(sV_0)\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}, \exp(sV_1)\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_1 + s \\ y_2 \\ y_3 + \frac{1}{2}y_2 s \end{pmatrix},$$
$$\exp(sV_2)\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 + s \\ y_3 - \frac{1}{2}y_1 s \end{pmatrix}.$$

This example can be characterized by no difference between the Ito SDE and the Stratonovich SDE.

5.2.1 Example of a source code in C : definitions of funtions for SDE

Since this model does not have parameters, we need not care about `params`. In the following file `sde_wa_manual_ah_system.c`, we give the definitions of all functions which are to be used for instatiation of an `SDE_WA_SYSTEM` data type object.

Remark that the last function `pf_f` is the definition of f in $E\left[f(X(T,x))\right]$. Here we let $f(x,K) = \max(x-K,0)$.

```
**************************************************************

/*
 * sde_wa_manual_program_sa_system.c
 */
/*
 * $Source: /Users/mariko/lib/source/sde_wa/sde_wa_manual/sde_wa_manual_progr
 * $Revision: 1.2 $
 * $Author: mariko $
```

```
 * $Date: 2011/11/24 03:34:10 $
 */

#include <sde_wa.h>
int sa_V_0(const double y[], double dy[], void *params){

  dy[0]=0.0;
  dy[1]=0.0;
  dy[2]=0.0;

  return SDE_WA_SUCCESS;

}

int sa_V_1(const double y[], double dy[], void *params){

  dy[0]=1.0;
  dy[1]=0.0;
  dy[2]=0.5*y[1];

  return SDE_WA_SUCCESS;
}

int sa_V_2(const double y[], double dy[], void *params){

  dy[0]=0.0;
  dy[1]=1.0;
  dy[2]=-0.5*y[0];

  return SDE_WA_SUCCESS;
}

int diff_sa_V_1(const double y[], double *dVdy[], void *params){
  dVdy[0][0]=0.0;
  dVdy[0][1]=0.0;
  dVdy[0][2]=0.0;

  dVdy[1][0]=0.0;
```

```
  dVdy[1][1]=0.0;
  dVdy[1][2]=0.0;

  dVdy[2][0]=0.0;
  dVdy[2][1]=0.5;
  dVdy[2][2]=0.0;

  return SDE_WA_SUCCESS;
}

int diff_sa_V_2(const double y[], double *dVdy[], void *params){
  dVdy[0][0]=0.0;
  dVdy[0][1]=0.0;
  dVdy[0][2]=0.0;

  dVdy[1][0]=0.0;
  dVdy[1][1]=0.0;
  dVdy[1][2]=0.0;

  dVdy[2][0]=-0.5;
  dVdy[2][1]=0.0;
  dVdy[2][2]=0.0;

  return SDE_WA_SUCCESS;
}

int exp_sa_sVy_0(double s, const double y[], double dy[],
               void *params){
  dy[0]=y[0];
  dy[1]=y[1];
  dy[2]=y[2];

  return SDE_WA_SUCCESS;
}

int exp_sa_sVy_1(double s, const double y[], double dy[],
               void *params){
  dy[0]=y[0]+s;
```

```
  dy[1]=y[1];
  dy[2]=y[2]+0.5*y[1]*s;

  return SDE_WA_SUCCESS;
}

int exp_sa_sVy_2(double s, const double y[], double dy[],
                 void *params){
  dy[0]=y[0];
  dy[1]=y[1]+s;
  dy[2]=y[2]-0.5*y[0]*s;

  return SDE_WA_SUCCESS;
}


/* def. of f */
double pf_f(double x, double K){
  return (x > K)? x-K : 0.0;
}

*****************************************************************
```

5.2.2 Example of a source code in C : main

In the main source code given below, we take the naive NV algorithm with quasi-Monte Carlo method as in the previous example. `sp` in this code stands for a "sample point". Since the NV algorithm uses a special data type for a sample point as we have seen in Chapter 4, we should pay a little attention in constructing it.

Freeing procedure is done according to Remark 4.

```
/*
 * sde_wa_manual_program_sa_nv.c
 */
/*
 * $Source: /Users/mariko/lib/source/sde_wa/sde_wa_manual/sde_wa_manual_progr
 * $Revision: 1.5 $
```

```
 * $Author: mariko $
 * $Date: 2011/11/24 03:34:26 $
 */

#include <stdlib.h> /* for malloc */
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_qrng.h>
#include <sde_wa.h>
#include "sde_wa_manual_program_sa_system.h"


int main (void){

  enum ALG alg = N_V;
  int m_moment = 5;
  int M = 100000;
  int n = 4;
  double x0=0.0;
  double x1=0.0;
  double K=0.5;

  {
    SDE_WA_SYSTEM *sde;
    SDE_WA_SLTN *sl;
    RV_NV sp;
    double **x;
    double sum;
    double dt=1.0/(double)n;
    gsl_qrng *q;
    double *u_seq1, *u_seq2, *u_seqB;
    double *n_seq1, *n_seq2;
    int i, k;
    double *tmp_pt;

    sde=alloc_SDE_WA_SYSTEM(3, 2, NULL);
    sde->sde_type=STR;
    sde->V[0]=sa_V_0;
```

```
sde->V[1]=sa_V_1;
sde->V[2]=sa_V_2;
sde->drift_corrector[0]=NULL;
sde->drift_corrector[1]=NULL;
sde->drift_corrector[2]=NULL;
sde->exp_sV[0]=exp_sa_sVy_0;
sde->exp_sV[1]=exp_sa_sVy_1;
sde->exp_sV[2]=exp_sa_sVy_2;

sl=alloc_SDE_WA_SLTN(alg, m_moment, sde);
sp.rv_nv_n=(double *)malloc(sizeof(double)*sde->dim_BM);

u_seq1=(double *)malloc(sizeof(double)*n*(sde->dim_BM+1));
u_seq2=u_seq1+n;
u_seqB=u_seq1+2*n;
n_seq1=(double *)malloc(sizeof(double)*n*(sde->dim_BM+1));
n_seq2=n_seq1+n;

x=(double **)malloc(sizeof(double *)*2);
for (i=0; i<2; i++) x[i]=(double *)malloc(sizeof(double)*sde->dim_y);

/* sobol seq. => seq.dim.<=40*/
q=gsl_qrng_alloc(gsl_qrng_sobol, (sde->dim_BM+1)*n);

for (sum=0.0, i=0; i< M; i++){

  gsl_qrng_get(q, u_seq1);

  for (k=0; k<n; k++){
    n_seq1[k] = sqrt(-2.0*log(u_seq1[k]))*cos(2.0*M_PIl*u_seq2[k]);
    n_seq2[k] = sqrt(-2.0*log(u_seq1[k]))*sin(2.0*M_PIl*u_seq2[k]);
  }/* for k */

  for (x[0][0]=x0, x[0][1]=x1, x[0][2]=0.0, k=0; k<n; k++){
    if(u_seqB[k]>=0.5) sp.rv_nv_b=T;
    else sp.rv_nv_b=H;
    sp.rv_nv_n[0]=n_seq1[k];
    sp.rv_nv_n[1]=n_seq2[k];
```

```
        next_SDE_WA(sl, dt, x[0], x[1], &sp);

        tmp_pt=x[0];
        x[0]=x[1];
        x[1]=tmp_pt;

      }/* for k */

      sum +=pf_f(x[0][2], K);
    }/* for i */


    free_SDE_WA_SLTN(sl);
    free_SDE_WA_SYSTEM(sde);

    printf("\n%.12e\n", sum/(double)M);

  }

  return SDE_WA_SUCCESS;

}
```

Index

Bibliography

[1] Peter E. Kloeden and Eckhard Platen, *Numerical Solution of Stochastic Differential Equations*, Springer Verlag, Berlin, 1999.

[2] Shigeo Kusuoka, *Approximation of Expectation of Diffusion Process and Mathematical Finance*, Advanced Studies in Pure Mathematics, Proceedings of Final Taniguchi Symposium, Nara 1998 (T. Sunada, ed.), vol. 31, 2001, pp. 147–165.

[3] Terry Lyons and Nicolas Victoir, *Cubature on Wiener Space*, Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences **460** (2004), 169–198.

[4] Mariko Ninomiya and Syoiti Ninomiya, *A new weak approximation scheme of stochastic differential equations by using the Runge–Kutta method*, Finance and Stochastics **13** (2009), no. 3, 415–443.

[5] Syoiti Ninomiya and Nicolas Victoir, *Weak Approximation of Stochastic Differential Equations and Application to Derivative Pricing*, Applied Mathematical Finance **15** (2008), 107–121.