Vectorización de algoritmos generales de convergencia fuerte para la solución de Ecuaciones Diferenciales Estocásticas (SDE)

Por

Carlos Eduardo Quiñones Botero

Director
Jaime Londoño L.
Profesor de Matemáticas
Universidad Nacional, Bogotá

Maestría en Matemáticas Aplicadas
Universidad Eafit
Medellín

2007

ABSTRACT Este proyecto de tesis explica la librería SDEBLAS la cual consiste de rutinas construidas con base en los estándares de programación BLAS y LAPACK para la solución numérica de ecuaciones diferenciales estocásticas (SDE) . Estas rutinas corren métodos numéricos que no imponen restricciones sobre el tipo de SDE de manera que la librería tenga el uso más general posible.

Contents

Prefacio

La teoría de las Ecuaciones Diferenciales Estocásticas (SDE) ha sido desarrollada en el último medio siglo, pero no se ha creado una librería para la solución numérica de este tipo de ecuaciones. A pesar de que actualmente existen métodos numéricos de alta eficiencia para la solución de las SDE y de que existen aplicaciones académicas e industriales en donde se utilizan, no se ha desarrollado una librería de fuente abierta que implemente vectorizadamente los mejores algoritmos disponibles actualmente. Las implementaciones actuales tienen una o varias de las siguientes desventajas: No son de fuente abierta, solo implementan uno o dos algoritmos, no son vectorizadas o implementan algoritmos que no son de aplicación general. Esta situación limita seriamente las aplicaciones de este tipo de ecuaciones, porque para cada aplicación se requiere escribir un código que resuelva las ecuaciones involucradas.

El objetivo de esta tesis es doble. Por un lado presentar una librería para la solución de SDE pero también ofrecer, para cada método numérico considerado, una discusión general del método, un poco de matemáticas analíticas, una discusión de la eficiencia algorítmica de la solución y finalmente, la implementación de estos algoritmos. Para realizar la última tarea escogimos el lenguaje de programación C como herramienta principal. Históricamente el principal lenguaje de programación para la computación científica fue FORTRAN pero C tiene ventajas significativas sobre éste, notablemente su portabilidad de un computador a otro. Aunque durante largo tiempo a C le faltaron librerías numéricas de alta calidad debido a la sólida posición de FORTRAN esta deficiencia fue superada con el paso de los años. De hecho, y debido a su alta calidad, la librería modelo para

el desarrollo de SDEBLAS fue LAPACK, en cierto sentido cumpliendo el deseo de los programadores de esta última, que quisieron que su librería fuera modelo para otros desarrollos de software [1].

La forma en que esta escrito este trabajo permite leerlo con varios niveles de sofisticación. Para los lectores que buscan desarrollar sus aplicaciones basta leer los detalles sobre como implementar la librería. En tanto que los lectores con una preparación más profunda en matemática, tales como matemáticos o estudiantes de maestría en finanzas o matemáticas aplicadas pueden encontrar interesante la discusión más formal sobre los métodos.

Los profesores y los libros de matemáticas deben ser inspiradores. Cuando me inicie como profesor trate de recordar a todos mis mejores profesores, buscando cuales eran los detalles que los hacian memorables, y creo que esto me ha servido en mi propio ejercicio docente. Con los libros de matemáticas deberíamos hacer igual cosa. Pero en este caso encontrar los buenos ejemplos puede ser más difícil. Más aun, los libros de matemáticas inspiradores que recuerdo eran, casi siempre, poco convencionales. De todas formas, he tratado de incorporar un poco del estilo de cada uno de ellos en este libro. De *Numerical Recipes in C* he tratado de imitar sus inolvidables capítulos introductorios, que, sin terminología matemática, son capaces de explicar conceptos matemáticos muy complejos y sobre todo, de jerarquizar conceptos, explicando por ejemplo, cuando deberia usarse un metodo numérico y no otro. Las mejores revisiones de literatura que conozco aparecen en el *Cálculo* de Spivak y son una vez más, evaluaciones cualitativas sin terminología matemática acerca de las fortalezas de los libros que cita. Mucho más difícil fue encontrar el tipo de exposición requerido para las aplicaciones. Prácticamente todos los textos que presentan aplicaciones sobre SDE tienen la misma deficiencia: no se sabe para que se aplican sus aplicaciones. Quizás ésto se explica por los sólidos conocimientos de ingeniería que requieren estas aplicaciones y por cierto desdén que existe entre los matemáticos hacia las matemáticas aplicadas. Y como un pintor que mueve su caballete de un lado al otro tratando de encontrar la perspectiva adecuada para su cuadro, creo haber hallado una solución de compromiso en la excelente exposición de aplicaciones que aparece en *Mathematical Biology* de Murray.

Al escribir *Vectorización de algoritmos generales numéricos de convergencia fuerte para Ecuaciones Diferenciales Estocásticas* me beneficié del consejo y experiencia de muchos colegas, pero especialmente de Jaime Londoño de la Universidad Nacional de Bogotá y de Andrés Villegas de la Universidad Eafit de Medellín a quienes quiero expresar mis agradecimientos. El autor quiere expresar tambien sus agradecimientos a COLCIENCIAS, entidad que financió parte de este proyecto de tesis a través del proyecto de investigación “Una nueva perspectiva del consumo y la inversión de agentes en una economía"(Contrato CT-284-2003, Código 1216-05-13569) que tiene como investigador principal al profesor Jaime Londoño.

1
Descripción de la librería

1.1. Problemas que SDEBLAS puede resolver

SDEBLAS incorpora los métodos más generales para resolver SDE. Sin embargo estos métodos también tienen ciertas restricciones, que se comentan en la documentación de cada uno de ellos. Su generalidad implica que no están optimizados para ciertos problemas o incluso pueden producir respuestas incorrectas. Un ejemplo interesante lo encontramos en la modelación de grandes moléculas de ADN superenrolladas, que no pueden ser modeladas con métodos tales como el Euler-Maruyama debido a que no preserva las órbitas moleculares y requiere métodos especiales de descomposición que si preservan las estructura subyacente [49].

1.2. Computadores y sistemas operativos para los cuales SDEBLAS esta disponible

SDEBLAS se diseñó para obtener una alta eficiencia en computadores escalares o vectoriales que tomen ventaja de las mejoras en eficiencia de BLAS y LAPACK. Sin embargo, debe tenerse en cuenta que la librería fue escrita, depurada y optimizada bajo el sistema operativo Linux Suse 9.0. La alta portabilidad de C permitirá correr el código en otros sistemas operativos, si bien no se garantiza que lo hará con la misma eficiencia que en Linux.

A pesar de que actualmente el procesador de graficos (GPU) tiene más capacidad de computo y ancho de banda en la memoria que la CPU [36], SDEBLAS no fue escrita para correr sobre la GPU debido a las siguientes desventajas:

- Desarrollar software sobre la GPU es altamente complejo debido a un modelo de programación restrictivo, una falta de virtualización de los recursos del hardware y la necesidad de reescribir los algoritmos dentro de un entorno centrado en gráficos, aparte de que cada fabricante puede requerir ajustes en la programación.
- Muchos fabricantes de GPU no ofrecen precisión en punto flotante. Es posible que esta situación no cambie en los próximos años e incluso es probable que muchos fabricantes nunca lo ofrezcan.
- La cantidad de memoria disponible tiende a ser pequeña para las necesidades de la computación científica.

1.3. SDEBLAS comparado con otros programas para tareas similares

Sobre la implementación práctica de algoritmos existen varias referencias en la literatura, la mayor parte basadas en programas comerciales que limitan su aplicación.

- Los programas en FORTRAN desarrollados por Artemiev y otros para el sistema DYNAMICS & CONTROL [51].
- Los programas en FORTRAN desarrollados por Talay (1994) para el sistema PRESTO [51]. Este sistema escribe programas en FORTRAN para resolver SDE, haciendo primero las transformaciones de Ito o de Stratonovich que sean necesarias utilizando MAPLE.
- Los programas en TURBO – PASCAL incluidos en el libro de Kloeden, Platen y Schurz [30]. Estos programas son una fuente muy importante en cuanto a métodos numéricos para SDE debido a que implementan muchas rutinas diferentes. Sin embargo, fueron escritos más con propósitos pedagógicos que de eficiencia y por lo tanto muchos algoritmos no se desarrollaron en forma general, en cuanto al número de dimensiones de la ecuación o de las fuentes de aleatoriedad. Algunos de estos programas tienen un equivalente en C, como nos indicó Schurz en una comunicación privada (2005).
- Los programas en C desarrollados por B. Martensen para el sistema GNANS. Solo implementa el algoritmo de Euler – Maruyama en el

grupo de los algoritmos de aplicación general e implementa uno de los algoritmos asintóticamente eficientes propuestos por Newton [44], el cual no es un algoritmo general.

- El código de simulación OSCIL desarrollado en C por Schurz [51], que, como su nombre lo dice, se refiere problemas con oscilaciones lineales y no lineales y por lo tanto no es de aplicación general.
- Burrage [12], describe el proceso de vectorización de un algoritmo y muestra las ventajas del procedimiento.
- Las rutinas para convergencia fuerte de SDE escritas en MAPLE del paquete STOCHASTIC [16], las cuales usan métodos que exigen derivadas (con excepción de la rutina presentada para el Euler-Maruyama) o que imponen restricciones sobre los coeficientes de la SDE.
- Higham [27] presenta unas rutinas escritas en MATLAB donde ilustra conceptos importantes sobre SDE pero no son de utilidad práctica porque restringen las dimensiones a utilizar.

1.4. SDEBLAS y BLAS

SDEBLAS se desarrolló sobre los estándares de programación BLAS y LAPACK. BLAS es un estándar que define la interface para para ejecutar operaciones matriciales y vectoriales básicas. Las rutinas de SDEBLAS se escribieron de manera que hicieran el mayor uso posible de BLAS. Existen implementaciones de BLAS de alta eficiencia para muchos computadores modernos. SDEBLAS debería ser compilado con la respectiva versión de BLAS con el fin de alcanzar la máxima eficiencia. La implementación modelo de SDEBLAS no está diseñada para tener el rendimiento de la versión optimizada, pero permitirá que los usuarios que no tienen una implementación de BLAS disponible para su computador puedan correr SDEBLAS. Referencias sobre BLAS pueden encontrarse en Lawson [32], Dongarra [19]y Dongarra [22].

1.5. Problemas conocidos de SDEBLAS

SDEBLAS utiliza el método de Broyden [48] vectorizado para resolver los métodos implícitos utilizando para tal efecto una matriz Jacobiana. Si la matriz Jacobiana es singular o casi singular, situación que no aparece con frecuencia en la práctica, la implementación que utiliza la librería fallará . Los métodos desarrollados para tratar con este problema monitorean el número de condición de la matriz Jacobiana y la perturban si se detecta que

es singular o casi singular, pero este procedimiento no esta implementado en la rutina suministrada. Si el usuario tiene este problema, puede utilizar otra librería numérica que implemente otro método para hallar raices multidimensionales.

1.6. Software relacionado

Tal como se explicó previamente en el prefacio, SDEBLAS esta construido sobre otros paquetes, tal como se detalla a continuacion.

1.6.1. BLAS

BLAS es un estándar que define la interface para ejecutar operaciones matriciales y vectoriales básicas. El nivel 1 de BLAS ejecuta operaciones vectoriales. El nivel 2 de BLAS ejecuta operaciones entre vectores y matrices. Y el nivel 3 de BLAS ejecuta operaciones entre matrices y matrices [19] [20] [21] [22]. Debido a que BLAS es eficiente, portable y ampliamente disponible, se utiliza comunmente para desarrollar software de alta eficiencia.

1.6.2. LAPACK

Es una librería escrita en FORTRAN 77 desarrollada sobre BLAS para resolver los problemas de álgebra lineal numérica más comunes. Se diseñó para obtener una alta eficiencia en muchos computadores modernos de alto rendimiento [1].

1.6.3. CTSS

SDEBLAS utiliza la librería CTSS para generar números aleatorios [35] . La implementación modelo se construyó sobre los estándares de programación BLAS y LAPACK con el fin de mejorar su calidad [53].

1.7. Desempeño de SDEBLAS

Debido a que no se han implementado librerías con algoritmos generales para la solución de SDE, nuestra librería no se puede comparar contra otras implementaciones. Solo podemos hacer una comparación contra librerías que implementan SDE con procesos estocásticos unidimensionales o fuentes de aleatoriedad unidimensionales con excepción del algoritmo Wong-Zakai que se implementa en la librería CTSS [53]. Las SDE escogidas para

ejecutar esta tarea fueron algunos de los ejemplos y aplicaciones más representativos que aparecen en la literatura. Véase el capítulo *Aplicaciones y ejemplos* para más detalles.

1.8. Documentación y convenciones de programación

Las listas de argumentos de todas las rutinas de SDEBLAS siguen un único conjunto de convenciones para su diseño y documentación.

1.8.1. Orden de los argumentos

Los argumentos de una función de SDEBLAS aparecen en el siguiente orden (en donde se requieran):

- Funciones externas llamadas, en particular las que generan el vector de corrimiento y la matriz de volatilidad
- Dimensiones de la SDE o de los procesos de Wiener
- Numero de simulaciones
- Tiempos en los que se evalua la función
- Tamaño del incremento del tiempo que se utilizará para el método numérico
- Apuntadores a cadenas generadoras de numeros seudoaleatorios
- Vector de inicio
- Vector final
- Otros datos requeridos por la función

1.8.2. Descripción de los argumentos

La descripción de los argumentos incluye:

- Una clasificación del argumento como (entrada), (salida), (entrada/salida). Esta ultima clasificación significa que se utiliza el espacio de memoria del argumento y allí se devuelve información.
- El tipo del argumento.
- Las dimensiones, si el argumento es una matriz. Las dimensiones de las SDE y de los procesos de Wiener deben ser especificadas y no pueden tomar valores cero o negativos.

- Una especificación del valor que debe suministrarse al argumento.
- Cualquier restricción que los valores suministrados deban satisfacer, si el argumento es un escalar.

1.8.3. Manejo de errores

Todas las rutinas documentadas de SDEBLAS devuelven un argumento entero que indica el exito o fracaso de la computación como se especifica a continuación:

- 0 significa terminación exitosa
- Un valor mayor que 0 significa que se presentó un error durante el proceso de computo. Si las rutinas fi_asde, fi_bsde y fi_solution devuelven un valor mayor que cero se suspende la simulación en curso y se inicia una nueva simulación.

1.9. Precisión de la librería

SDEBLAS utiliza el tipo de dato ctss_real que está implementado en la librería CTSS y cuyo único objetivo es permitir que la precisión de los cálculos sea float o double. Su configuración por defecto es double, pero si se requiere más velocidad y menor precisión en los cálculos puede ser configurado como float. Esto puede ser efectuado para los métodos explícitos, pero no para los métodos implícitos porque el método de Broyden para encontrar raices multidimensionales requiere que la precisión sea double. Para cambiar la configuración por defecto abra el archivo ctss_types.h, siga las instrucciones que aparecen en el archivo y compile nuevamente la librería.

Código de error	Descripción
0	(SDEBLAS_SUCCESS) SDEBLAS se ejecutó exitosamente
1	(SDEBLAS_FAILURE) SDEBLAS falló por una razón no especificada
2	(SDEBLAS_ILL_INPUT) Un parámetro de entrada tiene un valor no permitido
3	(SDEBLAS_MATH_ERROR) Se presentó una operación matemática no valida
4	(SDEBLAS_CONV_ERROR) El método no converge
5	(SDEBLAS_UNDERFLOW) Se produjo un valor más pequeño que lo permitido por la computadora
6	(SDEBLAS_OVERFLOW) Se produjo un valor mayor que lo permitido por la computadora
7	(SDEBLAS_JACOBIAN_SINGULAR_BROYDEN) No se puede encontrar la raiz multidimensional en el método implícito porque la matriz Jacobiana es singular
8	(SDEBLAS_MAX_ITER_BROYDEN) No se puede encontrar la raiz multidimensional en el método implícito porque se excedieron las iteraciones
9	(SDEBLAS_R_SINGULAR_BROYDEN) No se puede encontrar la raiz multidimensional en el método implícito porque la matriz R de la factorización QR es singular
10	(SDEBLAS_ROUNDOFF_PROBLEM_LINE_SEARCH) No se puede encontrar la raiz multidimensional en el método implícito porque ocurrió un error de redondeo
11	(SDEBLAS_JACOBIAN_NOTSTART_BROYDEN) No se puede encontrar la raiz multidimensional en el método implícito aunque se trató de reiniciar
12	(SDEBLAS_QRUPDATE_FAILURE) No se puede encontrar la raiz multidimensional en el método implícito porque la actualización de la factorización QR falló

Cuadro 1.1: Códigos de error

2

Herramientas para la solución numérica de SDE

2.1. Herramientas para la solución numérica de SDE

2.1.1. Procesos de Wiener

Un proceso de Wiener estándar $W = \{W(t), t > 0\}$ es un proceso estocástico sobre $[0, T]$ compuesto por variables aleatorias $W(t)$ con t en el intervalo $[0, T]$ y que además satisface las siguientes condiciones:

- $W(0) = 0$ $w.p$,1.
- Si $0 \leq s < t \leq T$ la variable aleatoria dada por $W(t) - W(s)$ se distribuye normalmente con media cero y varianza $t - s$.
- Si $0 \leq s < t < u < v \leq T$ los incrementos $W(t) - W(s)$ y $W(v) - W(u)$ son independientes.

Los procesos de Wiener se llaman algunas veces movimentos brownianos, el cual es un término que se utiliza para describir el comportamiento errático de las partículas en un líquido. El término se refiere al descubrimiento [5] del botánico inglés Robert Brown quien al examinar partículas de polen y esporas observó que se movían, lanzando la hipótesis, equivocada a la luz de los conocimientos modernos, de que los granos de polen estaban vivos. Este modelo matemático se utiliza para describir muchos otros fenómenos con los que no tiene similitud aparente. Ejemplos interesantes son las fluc-

tuaciones en el mercado de acciones y la evolución de las características físicas en el registro fósil [54].

2.1.2. Puente Browniano

En los métodos de paso variable frecuentemente se debe construir un proceso de Wiener cuyas trayectorias pasen por el mismo punto inicial x, no necesariamente 0, y un punto dado y en el tiempo $t = T$. A este proceso $B_{0,x}^{T,y}$ se le pueden definir trayectorias en $0 \leq t \leq T$ como

$$B_{0,x}^{T,y}(t,\omega) = x + W(t,\omega) - \frac{t}{T}\left\{W(T,\omega) - y + x\right\}$$

las cuales son llamadas un puente browniano. Se trata de un proceso gaussiano que satisface $B_{0,x}^{T,y}(0,\omega) = x$ y $B_{0,x}^{T,y}(T,\omega) = y$.

2.1.3. Convergencia en el sentido de la media de cuadrados

Si tenemos una secuencia infinita de variables aleatorias $X_1, X_2, ...$ con $E(X_n^2) < \infty$ para $n = 1, 2, ...$ $E(X^2) < \infty$ y

$$\lim_{n\to\infty} E(|X_n - X|^2) = 0$$

2.1.4. Integración estocástica

Generalidades

Podemos escribir una ODE en forma diferencial simbólica de la siguiente manera

$$dx = a(t,x)\,dt$$

e integrando en ambos lados, como una ecuación integral

$$x(t) = x_0 + \int_{t_0}^{t} a(s, x(s))\,ds$$

donde $x(t) = x(t; x_0, t_0)$ es una solución que satisface la condición inicial $x(t_0) = x_0$. Una condición suficiente para la existencia y unicidad de la solución $x(t) = x(t; x_0, t_0)$ es que la función $a(t,x)$ cumpla la condición de Lipschitz con respecto a la variable espacial [29], es decir

$$|a(t,x) - a(t,y)| \leq k\,|x - y| \tag{2.1}$$

para todo $x, y \in \mathbb{R}$ y $0 \leq t \leq T$ donde k es una constante positiva. Para garantizar la existencia global de una solución, esto es, su existencia para todo $t > 0$ se requiere acotar el crecimiento de la función $a(t,x)$

$$|a(t,x)|^2 \leq L\left(1+|x|^2\right)$$

para todo $x \in \mathbb{R}$ y $0 \leq t \leq T$ donde L es una constante positiva

Estas soluciones se pueden relacionar mediante la ecuación

$$x(t;x_0,t_0) = x(t;x(s;x_0,t_0),s) \tag{2.2}$$

para todo $t_0 \leq s \leq t$. Esta propiedad corresponde a un proceso de Markov.

Para capturar la dinámica de un movimiento browniano no bastan las ODE determínisticas anteriores. Requerimos SDE que tomen la forma

$$dX_t = a(t,X_t)\,dt + b(t,X_t)\,dW(t) \tag{2.3}$$

en donde $a(t,X_t)$ es un término determinístico de variación lenta llamado coeficiente de corrimiento, $b(t,X_t)$ es de variación rápida llamado coeficiente de difusión utilizado para modelar el ruido y $dW(t)$ es un proceso de Wiener.

Esta diferencial simbólica puede interpretarse como la ecuación

$$X(t) = X(0) + \int_{t_0}^{t} a(s,X(s))\,ds + \int_{t_0}^{t} b(s,X(s))dW(s)\,. \tag{2.4}$$

La primera integral es una integral de Riemann-Stieltjes convencional. Pero como en la segunda integral el proceso de Wiener W_t no es diferenciable no existe como una integral convencional. De manera que debemos desarrollar técnicas especiales para evaluar la segunda integral de 2.4. Existen dos interpretaciones para llevar ésto a cabo, la forma de Itô o la forma de Stratonovich. La forma de Stratonovich sigue las reglas usuales del cálculo de Riemann-Stieltjes y por eso se prefiere en muchas aplicaciones.

Debe notarse sin embargo que la ecuación 2.3 no tiene significado por si sola. Es simplemente una versión conveniente y apropiada de la ecuación 2.4. Sin embargo, desde el punto intuitivo es preferible porque captura la dinámica infinitesimal del proceso X.

Integrales de Itô y Stratonovich

De acuerdo con las técnicas para resolver integrales de Riemann-Stieltjes, si en la segunda integral de 2.4 tenemos que $b(t,x) \equiv b$ esperariamos que fuera igual a $b\{W_t(\omega) - W_{t_0}(\omega)\}$. Teniendo en cuenta este resultado, podemos proceder a definir la integral de Itô $I(f)(\omega)$ de una función f sobre el intervalo de tiempo $0 \leq t \leq 1$ como

$$I(f)(\omega) = \int_0^1 f(s,\omega)\,dW_s(\omega).$$

Si consideramos una función escalonada no aleatoria $f(t,\omega) \equiv f_j$ sobre t con $t_j \leq t < t_{j+1}$ para $j = 1, 2, ..., n$ donde $0 = t_1 < t_2 < ... < t_{n+1} = 1$ tenemos

$$I(f)(w) = \sum_{j=1}^{n} f_j\{W_{t_{j+1}}(\omega) - W_{t_j}(\omega)\}.$$

Esta integral se define análogamente, bajo condiciones apropiadas de medibilidad, para funciones escalonadas f_j aleatorias.

Para un integrando general $f : [0,1] \times \Omega \to \mathbb{R}$ se define la integral de Itô $I(f)$ como el límite de integrales $I(f^{(n)})$ de funciones escalonadas aleatorias $f^{(n)}$ que convergen a f.en probabilidad. Para que esto pueda ser posible, se debe determinar un modo de convergencia en el cual tal secuencia de funciones escalonadas y el límite existan. Se puede probar que esta condición es $E(\left|f^{(n)} - f\right|^2) \to 0$ cuando $n \to \infty$. En este caso, el límite en el sentido de la media de los cuadrados $I(f^{(n)})$ existe, es único y es llamado la Integral de Itô de f en $0 \leq t < 1$.

Análogamente se puede definir sobre cualquier intervalo acotado $[t_0, t]$ dando como resultado la variable aleatoria

$$X_t(\omega) = \int_{t_0}^{t} f(s,\omega)\, dW_x(\omega)$$

integrable en el sentido de la media de los cuadrados y con las propiedades $E(X_t) = 0$ y $E(X_t^2) = \int_{t_0}^{t} E(f(s,\cdot)^2)\, ds$.

En la Integral de Itô el integrando f se evalua en el extremo izquierdo del intervalo de discretización. Si el integrando se evalua en el punto medio del intervalo de discretización, obtenemos lo que se conoce como Integral de Stratonovich que se denota como $f \circ dW$. La integral de Stratonovich sigue las reglas de la integral de Riemann como por ejemplo

$$\int_0^t W_s(\omega) \circ dW_s(\omega) = \frac{1}{2} W_t^2(\omega).$$

Se observa que el anterior cálculo es muy similar a la respectiva propiedad del cálculo de Riemann-Stieltjes. Podría entonces pensarse que es mejor utilizar la Integral de Stratonovich en lugar de la Integral de Itô. Sin embargo, la Integral de Stratonovich no tiene una propiedad crucial para las demostraciones de convergencia, conocida como propiedad de martingala, y que puede expresarse como $E(W_t - W_s \mid A_s) = 0$ donde $A_t = \sigma\{W_s, 0 \leq s \leq t\}$, la σ-álgebra de los valores del proceso de Wiener hasta el tiempo t.y que en lenguaje corriente equivale a decir que la mejor predicción del futuro es el presente.

2.1.5. Ecuaciones diferenciales estocásticas

Al incluir procesos aleatorios dentro de ecuaciones diferenciales se obtienen dos tipos de ecuaciones diferentes. Cada uno de ellos tiene trayectorias muestrales diferenciables y no diferenciables respectivamente y requiren métodos de análisis completamente distintos. En el primer caso, tenemos las *ecuaciones diferenciales aleatorias*, las cuales corresponden a una ecuación diferencial ordinaria con coeficientes aleatorios o un valor inicial tambien aleatorio y que se resuelven trayectoria por trayectoria como ecuaciones diferenciales ordinarias con soluciones diferenciables. En el segundo caso uno de los términos corresponde a un proceso irregular aleatorio, se escribe como diferenciales estocásticas pero se interpretan como ecuaciones integrales con integrales de Itô o Stratonovich y son llamadas *ecuaciones diferenciales estocásticas* (SDE). Sus soluciones son no diferenciables, debido a que las trayectorias muestrales que provienen de los procesos de Wiener tampoco lo son. Un ejemplo clásico es el movimiento browniano, en el cual una partícula de polvo sobre la superficie del agua sufre un bombardeo molecular aleatorio que no depende del estado de las variables, por ejemplo la velocidad o la posición de la partícula de polvo, y que fue descrito por Langevin [29] de la siguiente manera:

$$\frac{dX_t}{dt} = -aX_t + b\,\xi_t$$

en el cual X_t es uno de los compontes de la velocidad de la partícula, el primer término de la derecha es una fuerza de fricción que depende de la velocidad y el segundo término representa las fuerzas moleculares por medio de un proceso de ruido blanco ξ con intensidad b. Ahora podemos interpretar la ecuación de Langevin como una ecuación diferencial estocástica

$$dX_t = -aX_t\,dt + b\,dW_t$$

o como una ecuación integral estocástica

$$X_t = X_{t_0} - \int_{t_0}^{t} a\,X_s\,ds + \int_{t_0}^{t} b\,dW_s \tag{2.5}$$

donde la segunda integral corresponde a una integral estocástica de Itô.

Muchas aplicaciones requieren que la dinámica subyacente sea modelada vectorialmente. En este caso consideramos un proceso de Wiener $m-$dimensional $W = \{W_t, t \geq 0\}$ con componentes $W_t^1, W_t^2, ..., W_t^n$ los cuales son procesos de Wiener escalares independientes y dW_n^j corresponde a la entrada $j - ésima$ del vector con m dimensiones correspondiente al proceso de Wiener. $a : [t_0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ y el coeficiente de difusión $b : [t_0, T] \times \mathbb{R}^d \to \mathbb{R}^{d\times m}$ son funciones reales suficientemente suaves que satisfacen un acotamiento de crecimiento lineal. a^k corresponde a la entrada $j - ésima$ del vector a y $b^{k,j}$ corresponde a la a la fila $k - ésima$

y la columna $j-ésima$ de la matriz b. Con estos vectores y matrices construimos la SDE $d-$dimensional

$$dX_t = a(t, X_t)\, dt + b(t, X_t)\, dW_t \tag{2.6}$$

que puede interpretarse como una *ecuación integral estocástica* con valor inicial $X_{t_0} \in \mathbb{R}^d$ así

$$X_t = X_{t_0} + \int_{t_0}^{t} a(s, X_s)\, ds + \int_{t_0}^{t} b(s, X_s)\, dW_s. \tag{2.7}$$

La primera integral es una integral de Lebesgue y la segunda una integral de Itô. Se calculan componente por componente de manera que el $i-ésimo$ componente es

$$X_t^i = X_{t_0}^i + \int_{t_0}^{t} a^i(s, X_s)\, ds + \sum_{j=1}^{m} \int_{t_0}^{t} b^{i,j}(s, X_s)\, dW_s^j \tag{2.8}$$

Se puede demostrar que la ecuación 2.7 tiene una solución fuerte única $X = \{X_t, t \in [t_0, T]\}$ sobre $[t_0, T]$ con

$$\sup_{t_0 \le t \le T} E(X_t^2) < \infty$$

siempre en cuando X_{t_0} sea independiente de $W = \{W_t, t \in [t_0, T]\}$ con $E(X_{t_0})^2 < \infty$ y los coeficientes a y b satisfacen las condiciones de Lipschitz.

De acuerdo con las ecuaciones 2.6 y 2.7 una SDE $d-$dimensional

$$dX_t = a(t, X_t)\, dt + \sum_{j=1}^{m} b^j(t, X_t)\, dW_t^j \tag{2.9}$$

para $t \in [t_0, T]$ con valor inicial $X_{t_0} \in \mathbb{R}^d$ se puede interpretar como una ecuación integral estocástica vectorial

$$X_t = X_{t_0} + \int_{t_0}^{t} a\,(s, X_s)\; ds + \sum_{j=1}^{m} \int_{t_0}^{t} b^j(s, X_s)\, dW_s^j. \tag{2.10}$$

En lo anterior hemos considerado la integral estocástica como una integral de Itô, pero con diferentes tipos de integrales estocásticas podemos obtener distintas soluciones de la ecuación 2.10 De entre los diferentes tipos de integrales estocásticas propuestos se prefiere con frecuencia la integral de Stratonovich porque sigue las reglas del cálculo determinístico. Para aclarar que estamos ante una SDE de Stratonovich utilizamos una notación diferente

$$dX_t = \underline{a}\,(t, X_t)\; dt + \sum_{j=1}^{m} b^j(t, X_t) \circ dW_t^j \tag{2.11}$$

o la respectiva ecuación integral estocástica vectorial

$$X_t = X_{t_0} + \int_{t_0}^{t} \underline{a}(s, X_s)\, ds + \sum_{j=1}^{m} \int_{t_0}^{t} b^j(s, X_s) \circ dW_s^j. \tag{2.12}$$

Si queremos que las ecuaciones 2.9 y 2.11 tengan la misma solución debemos corregir el coeficiente de corrimiento original $a(t, X_t)$ de la siguiente manera

$$\underline{a}(t, X_t) = a(t, x) - \frac{1}{2} \sum_{j=1}^{d} \sum_{k=1}^{m} b^{j,k}(t, x) \frac{\partial b^k}{\partial x_j}(t, x). \tag{2.13}$$

Si el coeficiente de corrimiento y el coeficiente de difusión no dependen del tiempo, decimos que la SDE es autónoma y toma la forma

$$dX_t = a(X_t)\, dt + b(X_t)\, dW_t.$$

Toda SDE no autónoma se puede escribir como una SDE vectorial autónoma con una dimensión adicional haciendo el primer componente del coeficiente de corrimiento igual a 1.

2.1.6. Expansiones de Taylor estocásticas

Expansión de Taylor determinística

Deduzcamos a continuación la serie de Taylor determinística con una nueva notación que permita presentar con más claridad la serie de Taylor estocástica. Consideremos la ecuación diferencial ordinaria unidimensional

$$\frac{d}{dt} X_t = a(X_t)$$

que tiene como solución $X = \{X_t, t \in [t_0, T]\}$ con valor inicial X_{t_0} donde $0 \leq t_0 \leq T$, y que puede ser escrita como una ecuación integral

$$X_t = X_{t_0} + \int_{t_0}^{t} a(X_S)\, ds.$$

Si la función a es suficientemente suave y además cumple la condición de la ecuación 2.1, podemos escribir, mediante la regla de la cadena

$$\frac{d}{dt} f(X_t) = a(X_t) f'(X_t)$$

Si denotamos L el operador definido como

$$Lf = a\, f'$$

la anterior ecuación escrita en forma integral se convierte en

$$f(X_t) = f(X_{t_0}) + \int_{t_0}^{t} Lf(X_s)\, ds \tag{2.14}$$

para todo $t \in [t_0, T]$. Cuando $f(x) = x$ tenemos

$$X_t = X_{t_0} + \int_{t_0}^{t} a(X_s)\, ds \tag{2.15}$$

si aplicamos la relación 2.14 a la función $f(x) = a$ obtenemos

$$\begin{aligned} X_t &= X_{t_0} + \int_{t_0}^{t} \left(a\left(X_{t_0}\right) + \int La\left(X_z\right)\, dz \right) ds \\ &= X_{t_0} + a(X_{t_0}) \int_{t_0}^{t} ds + \int_{t_0}^{t} \int_{t_0}^{s} La(X_z)\, dz\, ds \end{aligned} \tag{2.16}$$

que es la expansión de Taylor más simple. Si en la doble integral de 2.16 reemplazamos la relación $f = La$ obtenemos

$$X_t = X_{t_0} + a(X_{t_0}) \int_{t_0}^{t} ds + La(X_{t_0}) \int_{t_0}^{t} \int_{t_0}^{s} dz\, ds + R_3$$

en donde R^3 toma el valor

$$R^3 = \int_{t_0}^{t} \int_{t_0}^{s} \int_{t_0}^{z} L^2 a(X_u)\, du\, dz\, ds.$$

Al repetir este proceso $r+1$ veces con una función f que es $r+1$ diferenciable obtenemos la fórmula de Taylor general en forma integral

$$\begin{aligned} f(X_t) &= f(X_{t_0}) + \sum_{l=1}^{r} \frac{(t-t_0)^l}{l!} L^l f\left(X_{t_0}\right) \\ &\quad + \int_{t_0}^{t} ... \int_{t_0}^{s_2} L^{r+1} f(X_{s_1}) ds_1 ... d_{r+1} \end{aligned}$$

para $r = 1, 2, 3...$

Expansión de Stratonovich-Taylor

Asuma que X_t es el proceso estocástico que es la solución de la SDE de Stratonovich

$$X_t = X_{t_0} + \int_{t_0}^{t} \underline{a}(X_s)\, ds + \int_{t_0}^{t} b(X_s) \circ dW_s \tag{2.17}$$

Si f es una función dos veces diferenciable se sigue por la formula de Ito

$$f(X_t) = f(X_{t_0}) + \int_{t_0}^{t} \underline{L}^0 f(X_s)\, ds + \int_{t_0}^{t} \underline{L}^1 f(X_s) \circ dW_s \tag{2.18}$$

para $t \in [t_0, T]$ en donde $\underline{L}^0$ y $\underline{L}^1$ son

$$\underline{L}^0 f = a f'$$

y

$$\underline{L}^1 f = b f'$$

si hacemos $f(x) = x$ tenemos que $\underline{L}^0 f = \underline{a}$ y $\underline{L}^1 f = b$ y 2.18 se reduce a

$$X_t = X_{t_0} + \int_{t_0}^{t} \underline{a}(X_s)\, ds + \int_{t_0}^{t} b(X_s) \circ\, dW_s \tag{2.19}$$

aplicando 2.18 a las funciones $f = \underline{a}$ y $f = b$ del integrando obtenemos

$$X_t = X_{t_0} + \underline{a}(X_{t_0}) \int_{t_0}^{t} ds + b(X_{t_0}) \int_{t_0}^{t} \circ\, dW_s + R$$

en donde

$$\begin{aligned} R \;=\; & \int_{t_0}^{t} \int_{t_0}^{s} \underline{La}\,(X_z)\; dz\, ds + \int_{t_0}^{t} \int_{t_0}^{s} \underline{L}^1 \underline{a}\,(X_z) \circ dW_z\, ds \\ & + \int_{t_0}^{t} \int_{t_0}^{s} L^0 b\,(X_z)\; dz \circ dW_s + \int_{t_0}^{t} \int_{t_0}^{s} L^1 b\,(X_z) \circ dW_z \circ dW_s. \end{aligned}$$

La anterior explansión se conoce como expansión de Stratonovich-Taylor y es la expansión no trivial más sencilla. Analogamente se pueden obtener expansiones de Stratonovich-Taylor más complejas. La expansión de Stratonovich-Taylor se prefiere en análisis estocástico numérico sobre la expansión Itô-Taylor debido a que tiene una estructura más sencilla.

Expansión de Kahunen-Loeve

Un proceso de Wiener estándar $W = \{W_t, t \geq 0\}$ consiste de una cantidad no contable de variables aleatorias. Sin embargo es posible representarlo sobre cualquier intervalo acotado $0 \leq t \leq T$ en términos de una cantidad contable de variables aleatorias gaussianas independientes. Esta representación, llamada expansión de Kahunen-Loeve [29], es muy similar a las series de Fourier pero en ella los coeficientes son las variables aleatorias. En el sentido de la convergencia de la media de los cuadrados W_t puede calcularse como

$$W_t\,(\omega) = \sum_{n=0}^{\infty} Z_n\,(\omega)\, \phi_n\,(t)$$

para $0 \leq t \leq T$ y donde $Z_0, Z_1, Z_2...$ son variables aleatorias gaussianas estándar e independientes y $\phi_0, \phi_1, \phi_2...$ son las funciones no aleatorias

$$\phi_n(t) = \frac{2\sqrt{2T}}{(2n+1)\pi} \sin\left(\frac{(2n+1)\pi t}{2T}\right)$$

para $n = 0, 1, 2....$ Estas funciones de tiempo son ortogonales con respecto al producto interior definido como

$$\left\langle \phi_i, \phi_j \right\rangle = \int_0^T \phi_i(t)\,\phi_j(t)\,dt = 0$$

si $i \neq j$ para $i, j = 0, 1, 2...$ y además satisfacen

$$\sum_{n=0}^{\infty} \phi_n(s)\,\phi_n(t) = \text{mín}\{s, t\}$$

siendo esta última la función de covarianza del proceso W_t. Por otra parte, $Z_n(\omega)$ puede ser recobrado de

$$Z_n(\omega) = \frac{2}{T}\left(\frac{(2n+1)\pi}{2\sqrt{T}}\right)^2 \int_0^T W_t(\omega)\,\phi_n(t)\,dt$$

para $n = 0, 1, 2...$

Aproximación de integrales de Stratonovich múltiples

La expansión de Stratonovich-Taylor desarrollada en la ecuación 2.19 es el punto de partida para la construcción de soluciones para SDE. A partir de ella se pueden construir esquemas numéricos de alto orden para aproximar las soluciones de las SDE. Sin embargo, las integrales de Stratonovich múltiples no se pueden expresar a partir de integrales estocásticas más simples, especialmente cuando el proceso de Wiener es multidimensional. Sin embargo es posible obtener una aproximación de tales integrales a partir de la expansión de Kahunen-Loeve. No explicaremos todos los detalles para llegar a este resultado, pero el lector interesado puede encontrarlos en Kloeden [31] [30], Milstein [39] y Liske [34].

Si se tiene un puente browniano B_x generado a partir de un proceso de Wiener de $m-$dimensional $W_t = \left(W_t^1, W_t^2, ..., W_t^m\right)$ sobre el intervalo $[0, \Delta]$ y definido como

$$\left\{W_t - \frac{t}{\Delta} W_\Delta, 0 \leq t \leq \Delta\right\}$$

tenemos que la expansión de Fourier que converge en el sentido de la media de los cuadrados, para el componente $j - ésimo$ del puente browniano es

$$W_t^j - \frac{t}{\Delta} W_\Delta^j = \frac{1}{2} a_{j,0} + \sum_{r=1}^{\infty} \left(a_{j,r} \cos\left(\frac{2r\pi t}{\Delta}\right) + b_{j,r} \sin\left(\frac{2r\pi t}{\Delta}\right)\right)$$

en donde

$$a_{j,r} = \frac{2}{\Delta}\int_0^{\Delta}\left(W_s^j - \frac{s}{j}W_{\Delta}^j\right)\cos\left(\frac{2r\pi s}{\Delta}\right)\,ds$$

$$b_{j,r} = \frac{2}{\Delta}\int_0^{\Delta}\left(W_s^j - \frac{s}{j}W_{\Delta}^j\right)\sin\left(\frac{2r\pi s}{\Delta}\right)\,ds$$

son variables gaussianas para $j = 1, .., m$ y $r = 0, 1, 2...$

Por medio de un truncamiento podemos obtener una aproximación del puente browniano

$$W_t^{j,p} = \frac{t}{\Delta}W_{\Delta}^j + \frac{1}{2}a_{j,0} + \sum_{r=1}^{p}\left(a_{j,r}\cos\left(\frac{2r\pi t}{\Delta}\right) + b_{j,r}\sin\left(\frac{2r\pi t}{\Delta}\right)\right). \quad (2.20)$$

Denotemos por $J_{(j_1,j_2,...,j_l),t}$ la integral múltiple de Stratonovich definida como

$$J_{(j_1,j_2,...,j_l),t} = \int_0^t\int_0^{s_l}...\int_0^{s_2} dW_{s_1}^{j_1}\circ...\circ dW_{s_l-1}^{j_l-1}\circ dW_{s_l}^{j_l} \quad (2.21)$$

para $\alpha = (j_1, ...j_l) \in \{0, 1...m\}^l$ en donde $W = (W^1, ..., W^m)$ es un proceso de Wiener $m-$dimensional. Adoptamos la convención $W_t^0 = t$ con el fin de que podamos establecer adicionalmente la integración con respecto a t.

Para las integrales de Riemann-Stieltjes de las funciones lisas definidas en 2.20 escribamos

$$J_{(j_1,j_2,...,j_l),t}^P = \int_0^t\int_0^{s_l}...\int_0^{s_2} dW_{s_1}^{j_1,p}dW_{s_2}^{j_2,p}...dW_{s_l}^{j_1,p}.$$

Para cada $j = 1, ..., m$ y $r = 1, ..., p$ con $p = 1, 2...$ definimos las siguientes variables gaussianas independientes estandarizadas:

$$\xi_j = \frac{1}{\sqrt{\Delta}}W_{\Delta}^j$$

$$\zeta_{j,r} = \sqrt{\frac{2}{\Delta}}\pi r a_{j,r}$$

$$\eta_{j,r} = \sqrt{\frac{2}{\Delta}}\pi r b_{j,r}$$

$$\mu_{j,p} = \frac{1}{\sqrt{\Delta\rho_\rho}}\sum_{r=p+1}^{\infty} a_{j,r}$$

$$\phi_{j,p} = \frac{1}{\sqrt{\Delta\alpha_p}} \sum_{r=p+1}^{\infty} \frac{1}{r} b_{j,r}$$

en donde $\mu_{j,p}$ y $\phi_{j,p}$ convergen en el sentido de la media de los cuadrados y teniendo

$$\rho_\rho = \frac{1}{12} - \frac{1}{2\pi^2} \sum_{r=1}^{p} \frac{1}{r^2}$$

$$\alpha_p = \frac{\pi^2}{180} - \frac{1}{2\pi^2} \sum_{r=1}^{p} \frac{1}{r^4}.$$

Se puede deducir que podemos aproximar las integrales múltiples de Stratonovich $J_{(j_1,\ldots,j_l),\Delta}$ en el sentido de tener las mismas distribuciones de probabilidad por medio de su integral equivalente de Riemann-Stieltjes $J^p_{(j_1,\ldots,j_l),\Delta}$ para $p = 1, 2...$(omitiendo Δ de J^p_α) y asumiendo que $j, j_1, j_2, j_3 \in \{1, ..., m\}$.

Para integrales de Stratonovich de multiplicidad uno

$$J^p_{(0)} = \Delta$$

$$J^p_{(j)} = \sqrt{\Delta}\,\xi_j.$$

Para integrales de Stratonovich de multiplicidad dos

$$J^p_{(0,0)} = \frac{1}{2}\Delta^2$$

$$J^p_{(j,0)} = \frac{1}{2}\Delta(\sqrt{\Delta}\xi_j + a_{j,0})$$

$$J^p_{(0,j)} = \frac{1}{2}\Delta(\sqrt{\Delta}\xi_j - a_{j,0})$$

$$J^p_{(j_1,j_2)} = \frac{1}{2}\Delta\xi_{j_1}\xi_{j_2} - \frac{1}{2}\sqrt{\Delta}(a_{j_2,0}\xi_{j_1} - a_{j_1,0}\xi_{j_2}) + \Delta A^p_{j_1,j_2}$$

en donde

$$a_{j,0} = -\frac{1}{\pi}\sqrt{2\Delta} \sum_{r=1}^{p} \frac{1}{r}\zeta_{j,r} - 2\sqrt{\Delta\rho_\rho}\mu_{j,p}$$

$$A^p_{j_1,j_2} = \frac{1}{2\pi} \sum_{r=1}^{p} \frac{1}{r} \left(\zeta_{j_1,r}\eta_{j_2,r} - \eta_{j_1,r}\zeta_{j_2,r}\right).$$

Para integrales de Stratonovich de multiplicidad tres

$$J^p_{(0,0,0)} = \frac{1}{3!}\Delta^3$$

$$J^p_{(0,j,0)} = \frac{1}{3!}\Delta^{5/2}\xi_j - \frac{1}{\pi}\Delta^2 b_j$$

$$J^p_{(j,0,0)} = \frac{1}{3!}\Delta^{5/2}\xi_j + \frac{1}{4}\Delta^2 a_{j,0} + \frac{1}{2\pi}\Delta^2 b_j$$

$$J^p_{(0,0,j)} = \frac{1}{3!}\Delta^{5/2}\xi_j - \frac{1}{4}\Delta^2 a_{j,0} + \frac{1}{2\pi}\Delta^2 b_j$$

$$\begin{aligned} J^p_{(j_1,0,j_2)} &= \frac{1}{3!}\Delta^2\xi_{j_1}\xi_{j_2} + \frac{1}{2}a_{j_1,0}J^p_{(0,j_2)} + \frac{1}{2\pi}\Delta^{3/2}\xi_{j_2}b_{j_1} \\ &\quad -\Delta^2 B^p_{j_1,j_2} - \frac{1}{4}\Delta^{3/2}a_{j_2,0}\xi_{j_1} + \frac{1}{2\pi}\Delta^{3/2}\xi_{j_1}b_{j_2} \end{aligned}$$

$$\begin{aligned} J^p_{(0,j_1,j_2)} &= \frac{1}{3!}\Delta^2\xi_{j_1}\xi_{j_2} - \frac{1}{\pi}\Delta^{3/2}\xi_{j_2}b_{j_1} + \Delta^2 B^p_{j_1,j_2} \\ &\quad -\frac{1}{4}\Delta^{3/2}a_{j_2,0}\xi_{j_1} + \frac{1}{2\pi}\Delta^{3/2}\xi_{j_1}b_{j_2} + \Delta^2 C^p_{j_1,j_2} + \frac{1}{2}\Delta^2 A^p_{j_1,j_2} \end{aligned}$$

$$\begin{aligned} J^p_{(j_1,j_2,0)} &= \frac{1}{2}\Delta^2\xi_{j_1}\xi_{j_2} - \frac{1}{2}\Delta^{3/2}\left(a_{j_2,0}\xi_{j_1} - a_{j_1,0}\xi_{j_2}\right) \\ &\quad +\Delta^2 A^p_{j_1,j_2} - J^p_{(j_1,0,j_2)} - J^p_{(0,j_1,j_2)} \end{aligned}$$

$$\begin{aligned} J^p_{(j_1,j_2,j_3)} &= \frac{1}{\sqrt{\Delta}}\xi_{j_1}J^p_{(0,j_2,j_3)} + \frac{1}{2}a_{j_1,0}J^p_{(j_2,j_3)} + \frac{1}{2\pi}\Delta b_{j_1}\xi_{j_2}\xi_{j_3} \\ &\quad -\Delta^{3/2}\xi_{j_2}B^p_{j_1,j_3} + \Delta^{3/2}\xi_{j_3}\left(\frac{1}{2}A^p_{j_1,j_2} - C^p_{j_2,j_1}\right) + \Delta^{3/2}D^p_{j_1,j_2,j_3} \end{aligned}$$

en donde

$$b_j = \sqrt{\frac{\Delta}{2}}\sum_{r=1}^{p}\frac{1}{r^2}\eta_{j,r} + \sqrt{\Delta\alpha_p}\phi_{j,p}$$

$$B^p_{j_1,j_2} = \frac{1}{4\pi^2}\sum_{r=1}^{p}\frac{1}{r^2}(\zeta_{j_1,r}\zeta_{j_2,r} + \eta_{j_1,r}\eta_{j_2,r})$$

$$C^p_{j_1,j_2} = -\frac{1}{2\pi^2}\sum_{\substack{r,l=1 \\ r\neq l}}^{p}\frac{r}{r^2-l^2}\left(\frac{1}{l}\zeta_{j_1,r}\zeta_{j_2,l} - \frac{l}{r}\eta_{j_1,r}\eta_{j_2,l}\right)$$

$$
\begin{aligned}
D^{p}_{j_1,j_2,j_3} &= -\frac{1}{\pi^2 2^{5/2}}\sum_{r,l=1}^{p}\frac{1}{l(l+r)}[\zeta_{j_2,l}\left(\zeta_{j_3,l+r}\eta_{j_1,r}-\zeta_{j_1,r}\eta_{j_1,l+r}\right)\\
&\qquad +\eta_{j_2,l}\left(\zeta_{j_1,r}\zeta_{j_3,l+r}+\eta_{j_1,r}\eta_{j_3,l+r}\right)]\\
&\quad +\frac{1}{\pi^2 2^{5/2}}\sum_{l=1}^{p}\sum_{r=1}^{l-1}\frac{1}{r(l-r)}[\zeta_{j_2,l}\left(\zeta_{j_1,r}\eta_{j_3,l-r}+\zeta_{j_3,l-r}\eta_{j_1,r}\right)\\
&\qquad -\eta_{j_2,l}\left(\zeta_{j_1,r}\zeta j_{3,l-r}-\eta_{j_1,r}\eta_{j_3,l-r}\right)]\\
&\quad +\frac{1}{\pi^2 2^{5/2}}\sum_{l=1}^{p}\sum_{r=l+1}^{2p}\frac{1}{r(r-l)}[\zeta_{j_2,l}\left(\zeta_{j_3,r}-l\eta_{j_1,r}-\zeta_{j_1,r}\eta_{j_3,r-l}\right)\\
&\qquad +\eta_{j_2,l}\left(\zeta_{j_1,r}\zeta_{3,l-r}+\eta_{j_1,r}\eta_{j_3,r-l}\right)]
\end{aligned}
$$

donde para $r > p$ hacemos $\zeta_{j,r} = 0$ y $\eta_{j,r} = 0$ para todo $j = 1, ..., m$

2.1.7. Conceptos de convergencia para la solución de SDE

Con el fin de juzgar la calidad de una aproximación en tiempo discreto debemos establecer un criterio que refleje el objetivo de la simulación. A diferencia de la modelación determinística, existen en los modelos estocásticos muchos tipos diferentes de convergencia que tienen sentido sea desde el punto de vista teórico o práctico. De manera que al diseñar algoritmos para encontrar la solución de la SDE se deben especificar los tipos de problemas donde se aplicarán con el fin de optimizar su eficiencia con respecto a un criterio de convergencia determinado [47]. En términos generales, existen dos tipos de convergencia: La primera, denominada convergencia fuerte en donde se aproximan las trayectorias de la solución de la SDE; y la segunda, denominada convergencia debil en donde se aproximan los momentos o la distribución de probabilidad de la SDE. A continuación presentamos el criterio de convergencia fuerte que es el que nos interesa en nuestro trabajo:

Definición 2.1 *Un proceso estocástico Y^{δ} que es una aproximación en tiempo discreto Y^{δ} de la solución exacta X de una SDE con tamaño máximo de subintervalo δ converge en el sentido fuerte con orden $\gamma \in (0, \infty)$ en el tiempo t si existe una constante $K < \infty$ que no depende de δ, y un $\delta_0 > 0$ tal que*

$$E\left|X_t - Y^{\delta}(t)\right| \leq K\,\delta^{\gamma} \tag{2.22}$$

para cada $\delta \in (0, \delta_0)$.

En nuestras pruebas numéricas nos enfocaremos en el error en el punto final $t = T$ de manera que el error $e^{strong}_{\Delta t}$ en el punto extremo es

$$e_{\Delta t}^{strong} = E\left|X_T - Y^{\delta}(T)\right| \tag{2.23}$$

y también calcularemos la tasa de convergencia fuerte $R(\Delta t)$

$$R(\Delta t) = \frac{e_{\Delta t}^{strong}}{\Delta t}.$$

Como el acotamiento especificado en 2.22 se debe cumplir para cualquier punto de $[0, T]$ se debe cumplir también para el punto final, por lo tanto

$$e_{\Delta t}^{strong} = K\delta^{\gamma}. \tag{2.24}$$

Existen situaciones donde se requiere una aproximación a la trayectoria de la solución de la SDE, tales como la generación del escenario del precio de una acción en el mercado, la computación del estimado de un filtro de alguna variable no observada o las pruebas de un estimador estadístico para los parámetros de la solución de una SDE. En otras situaciones solo interesa estimar los momentos o la distribución de probabilidad de la solución de la SDE. En el primer caso se requeriría un algoritmo con convergencia fuerte y en el segundo un algoritmo con convergencia débil.

2.1.8. Intervalos de confianza para el error en la convergencia fuerte

Utilizando el teorema del límite central se puede demostrar que $e_{\Delta t}^{strong}$ se comporta asintóticamente como una variable aleatoria gaussiana cuando aumenta el número de simulaciones N y converge en distribución a la esperanza no aleatoria del error de la convergencia fuerte cuando $N \to \infty$. Es claro que no se puede generar un número infinito de trayectorias pero podemos estimar la varianza σ_{ϵ}^2 de $\hat{e}_{\Delta t}^{strong}$ y de esta manera construir un intervalo de confianza para $e_{\Delta t}^{strong}$. Esto se puede llevar a cabo efectuando M grupos de N simulaciones cada uno y calculando la media de la media de cada grupo

$$\hat{\epsilon} = \frac{1}{NM}\sum_{j=1}^{M}\sum_{k=1}^{N}\left|X_{T,k,j} - Y_{T,k,j}\right|$$

y por lo tanto la varianza es

$$\hat{\sigma}_{\epsilon}^2 = \frac{1}{M-1}\sum_{j=1}^{M}\left(\hat{\epsilon}_j - \hat{\epsilon}\right)^2$$

Se ha observado empíricamente que las medias de los grupos se pueden asimilar como gaussianas si el numero de simulaciones de cada bachada es mayor que 15. Y como podemos utilizar una distribución t-Student para

construir los intervalos de confianza de la suma de distribuciones gaussianas independientes, tenemos que una distribución t_student con $M-1$ grados de libertad para un intervalo de confianza ϵ de $100\,(1-\alpha)$ % aplicada a nuestro caso tiene la forma

$$(\hat{\epsilon} - \Delta\hat{\epsilon},\, \hat{\epsilon} + \Delta\hat{\epsilon})$$

en donde

$$\Delta\epsilon = t_{1-\alpha,M-1}\sqrt{\frac{\hat{\sigma}_{\epsilon}^{2}}{M}}$$

con $t_{1-\alpha,M-1}$ se determina a partir de la distribución t-Student con $M-1$ grados de libertad. En nuestros resultados utilizaremos $M = 20$ y $\alpha = 0{,}1$ que corresponde a $t_{1-\alpha,M-1} \approx 1{,}73$. Este valor permite obtener un error absoluto ϵ colocado en el intervalo de confianza con probabilidad $1-\alpha = 0{,}9$.

2.2. Métodos para encontrar raices multidimensionales

En cada paso de la simulación de un método implícito se debe encontrar una raiz multidimensional. Para solucionar este problema existen varios métodos, pero los más conocidos son el método de Newton y el método de Broyden. El método de Newton no fue implementado en SDEBLAS porque necesita derivadas analíticas y el objetivo de SDEBLAS es implementar métodos lo más generales posibles. El método de Broyden no tiene este inconveniente pero en ciertas situaciones puede aparecer una matriz singular o casi singular en las operaciones y la solución a este problema no ha sido determinada complemente [48]. Afortunadamente esta situación es rara en la práctica y si se presenta, SDEBLAS abortará la simulación en curso y reiniciará una nueva simulación. SDEBLAS no utiliza ninguna librería externa que implemente el método de Broyden debido principalmente a que estas librerías al ser llamadas solicitan memoria dentro de sus rutinas internas para alrededor de diez matrices auxiliares y al final liberan esta memoria [48]. Este proceso consume tiempo y no es problema si se utiliza para encontrar una sola raíz multidimensional como es el objetivo usual de estas librerías. Pero en nuestro caso debemos encontrar repetidamente miles de raices multidimensionales y esas solicitudes de memoria repetidas tomarían mucho tiempo.

3
Métodos numéricos

Los métodos sin derivadas se obtienen a partir de los correspondientes esquemas de Taylor fuertes reemplazando las derivadas por diferencias finitas utilizando valores de soporte apropiados. Enfatizamos que no son simples adaptaciones heurísticas de los métodos determinísticos. Los métodos numéricos tratan de resolver la SDE $d-$dimensional

$$dX_t = a(t, X_t)\, dt + b(t, X_t)\, dW_t$$

en donde el coeficiente de corrimiento $a : [t_0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ y el coeficiente de difusión $b : [t_0, T] \times \mathbb{R}^d \to \mathbb{R}^{d\times m}$ son funciones reales suficientemente suaves que satisfacen un acotamiento de crecimiento lineal. a^k corresponde a la entrada $j-ésima$ del vector a y $b^{k,j}$ corresponde a la a la fila $k-ésima$ y la columna $j-ésima$ de la matriz b.

Por dW_t entendemos los incrementos de un proceso de Wiener $m-$ dimensional que discretizados tienen como entrada $j-ésima$

$$\Delta W_n^j = \int_{\tau_n}^{\tau_{n+1}} dW_t^j = W_{\tau_{n+1}}^j - W_{\tau_n}^j$$

Existen diferentes métodos para aproximar la solución de SDE de Itô y de Stratonovich, de los cuales solo nos interesan los que no requieren derivadas debido a que la diferenciación numérica no es estable [6], y que relacionamos a continuación.

3.1. Método de Euler Maruyama explícito

Para el caso general multidimensional, en $d, m = 1, 2, ...$ el $k - ésimo$ componente del método de Euler-Maruyama tiene la forma

$$Y_{n+1}^k = Y_n^k + a^k \Delta_n + \sum_{j=1}^{m} b^{k,j} \Delta W_n^j. \tag{3.1}$$

en donde $Y_0^k \in R^d$. Por Δ_n entendemos la discretización de dt correspondiente a la longitud del subintervalo del tiempo $[\tau_n, \tau_{n+1}]$ y que se calcula como

$$\Delta_n = \int_{\tau_n}^{\tau_{n+1}} dt = \tau_{n+1} - \tau_n.$$

3.2. Método de Euler Maruyama implícito

Se trata del método más sencillo en la familia de los métodos implícitos y tiene orden de convergencia fuerte de $\gamma = 0{,}5$. Con el fin de garantizar la existencia de la solución únicamente el término que corresponde al coeficiente de corrimiento es implícito [30].

Para el caso multidimensional general $d, m = 1, 2, ...$ la familia de métodos de Euler-Maruyama implícitos tiene como $k - ésimo$ componente

$$Y_{n+1}^k = Y_n^k + \{\alpha a^k(\tau_{n+1}, Y_{n+1}) + (1-\alpha)a^k\}\Delta_n + \sum_{j=1}^{m} b^{k,j} \Delta W_n^j \tag{3.2}$$

en donde $Y_0^k \in R^d$ y $\alpha \in [0, 1]$ genera el grado en que el método es implícito. Con $\alpha = 0$ se obtiene el método de Euler-Maruyama explícito y con $\alpha = 1$ se tiene un método completamente implícito.

3.3. Método de Milstein-Platen explícito

Debido a las diferencias entre el cálculo estocástico y el cálculo determínistico, no es posible hacer generalizaciones para las SDE de los esquemas numéricos determinísticos más conocidos, tales como el Runge-Kutta. En la literatura se encuentran contra-ejemplos en donde se demuestra que esquemas tales como el método de Heun no convergen [30].

A partir de un método propuesto por Milstein [38] que requería la evaluación de derivadas, Platen [46] desarrolló otro método que no requeria esa evaluación. En el caso general multidimensional $d, m = 1, 2, 3, ..$ el $k-ésimo$ componente se aproxima a la solución de la SDE de Stratonovich 2.11 como

$$Y_{n+1}^k = Y_n^k + \underline{a}^k \Delta_n + \sum_{j=1}^{m} b^{k,j} \Delta W_n^j + \frac{1}{\sqrt{\Delta_n}} \sum_{j_1=1}^{m} \sum_{J_2=1}^{m} \{b^{k,j_2}(\tau_n, \bar{\Psi}_n^{j_1}) - b^{k,j_2}\} J_{(j_1,j_2)} \tag{3.3}$$

en donde $Y_0^k \in R^d$ y

$$\bar{\Psi}_n^j = Y_n + \underline{a}\Delta_n + b^j \sqrt{\Delta_n}$$

Se aproxima a la solución de la SDE de Itô como

$$Y_{n+1}^k = Y_n^k + a^k \Delta_n + \sum_{j=1}^{m} b^{k,j} \Delta W_n^j + \frac{1}{\sqrt{\Delta_n}} \sum_{j_1=1}^{m} \sum_{J_2=1}^{m} \{b^{k,j_2}(\tau_n, \bar{\Psi}_n^{j_1}) - b^{k,j_2}\} I_{(j_1,j_2)} \tag{3.4}$$

en donde $Y_0^k \in R^d$ y

$$\bar{\Psi}_n^j = Y_n + a\Delta_n + b^j \sqrt{\Delta_n}$$

3.4. Métodos de tipo Runge-Kutta implícito

Interpolando entre el esquema completamente implícito y el correspondiente esquema explícito, podemos formar una familia de métodos implícitos de convergencia fuerte de tipo Runge-Kutta con orden de convergencia fuerte de $\gamma = 1$. En el caso multidimensional general $d, m = 1, 2, ...$ su $k-ésimo$ componente se aproxima a la solución de la SDE de Itô 2.9 como

$$Y_{n+1}^k = Y_n^k + \{\alpha a(\tau_{n+1}, Y_{n+1}^k) + (1-\alpha)a^k\}\Delta_n + \sum_{j=1}^{m} b^{k,j} \Delta W_n^j + \frac{1}{\sqrt{\Delta_n}} \sum_{j_1,j_2}^{m} \{b^{k,j_2}(\tau_n, \bar{\Psi}_n^{j_1}) - b^{k,j_2}\} I_{(j_1,j_2)} \tag{3.5}$$

en donde $Y_0^k \in R^d$ y

$$\bar{\Psi}_n^j = Y_n + a\Delta_n + b^j \sqrt{\Delta_n}$$

para $j = 1, ..., m$ y parámetro $\alpha \in [0, 1]$ [30].

4
Aplicaciones y ejemplos

En la librería SDEBLAS se desarrollaron algoritmos correspondientes a los métodos de Euler-Maruyama, Euler-Maruyama implícito, Milstein-Platen y Runge-Kutta implícito. Estos algoritmos se corrieron con 20 grupos de 100 simulaciones cada uno y se compararon para la solución de los problemas que describimos abajo. Si las aplicaciones y ejemplos escogidos tenían soluciones explícitas se calcularon las medias de los errores con un nivel de confianza del 90 % para comparar el nivel de rendimiento. La media de los errores está dentro del rango de valores reportados en la literatura como por ejemplo [9].

Estas aplicaciones y ejemplos se corrieron en un computador con un procesador Celeron de 800 Mhz bajo el sistema operativo Linux Suse 9.0. Los tiempos que tomaron las distintas pruebas variaron ampliamente dependiendo, en orden descendiente de importancia, de la cantidad de pasos requeridos, del algoritmo utilizado y de la dimensión del problema. La cantidad de pasos requeridos aumenta el tiempo de procesamiento en forma aproximadamente lineal. El algoritmo más rápido fue el Euler-Maruyama explícito y el más lento el Runge-Kutta implícito, que es entre 12 y 13 veces más lento que el primero, lo cual es consistente con el mayor numero de cálculos requeridos y con la raíz multidimensional implícita que se debe calcular en cada paso. Al aumentar la dimensión del espacio de estados del proceso de uno a dos, el tiempo de procesamiento se incrementa 30 %. Bajo estas consideraciones los tiempos de procesamiento variaron entre 0,45 s y 56,9 s. Estos valores cambiarán desde luego en forma importante dependiendo del hardware, del sistema operativo y de los procesos concurrentes que este corriendo cada computador.

4.1. Ejemplo 1: Exponencial unidimensional

La solución de la ecuación [3]

$$dX_t = \alpha X_t\, dt + \sigma X_t\, dW_t \qquad X(0) = 1{,}0 \qquad t \in [0,\, 1]$$

esta dada por

$$X(t) = x_0 \exp\left\{\left(\alpha - \frac{1}{2}\sigma^2\right) t + \sigma W\,(t)\right\}$$

Algoritmo	$\hat{\epsilon}$	$\Delta\epsilon$
Euler-Maruyama explícito vectorizado SDEBLAS	1.50553e-1	4.40700e-3
Euler-Maruyama implícito vectorizado SDEBLAS	1.53515e-1	4.55600e-3
Milstein-Platen explícito vectorizado SDEBLAS	3.19020e-2	1.20200e-3
Runge-Kutta implícito vectorizado SDEBLAS	2.10700e-2	1.17100e-3

Cuadro 4.1: Comparación del error y tasa de convergencia de los algoritmos para resolver la SDE de la exponencial unidimensional con $\alpha = 1$, $\sigma = 1$, $h = 0{,}01$ y $t \in [0,\, 1]$.

En el cuadro 4.1 se muestra que los todos los métodos convergen a la solución.

La rutina emstrong.m escrita en Matlab y publicada por Higham [27] que implementa el método de Euler-Maruyama no es exactamente comparable porque evalua una sola bachada de 1000 simulaciones, calcula diferentes pasos y al final genera una gráfica. Pero puede ser modificada fácilmente para evaluar una sola bachada de 100 simulaciones con un paso de $h = 0{,}01$ y no generar ninguna gráfica, de manera que podamos compararla con el desempeño de SDEBLAS. Los resultados muestran que SDEBLAS es aproximadamente 22 veces más rápido que emstrong.m.

4.2. Ejemplo 2: Exponencial bidimensional

Se puede demostrar que la solución de la SDE

$$dX_t = AX_t\, dt + BX_t\, dW_t \qquad X(0) = [1{,}0\ 0{,}5]^T \qquad t \in [0,\, 1]$$

con

$$A = \begin{bmatrix} -a & a \\ a & -a \end{bmatrix} \ y\ B = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix}$$

se puede escribir como

$$X_t = P \begin{bmatrix} \exp\left(\rho^+\left(t\right)\right) & 0 \\ 0 & \exp\left(\rho^-\left(t\right)\right) \end{bmatrix} P^{-1} X_0$$

en donde

$$\rho^{\pm}\left(t\right) = \left(-a - \frac{1}{2}b^2 \pm a\right) t + b\, W_t$$

y

$$P = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{con} \quad P^{-1} = P$$

Algoritmo	$\hat{\epsilon}$	$\Delta\epsilon$
Euler-Maruyama explícito vectorizado SDEBLAS	8.02540e-2	4.69400e-3
Euler-Maruyama implícito vectorizado SDEBLAS	8.02540e-2	4.69400e-3
Milstein-Platen explícito vectorizado SDEBLAS	4.46400e-3	2.93000e-4
Runge-Kutta implícito vectorizado SDEBLAS	4.46400e-3	2.93000e-4

Cuadro 4.2: Comparación del error y tasa de convergencia de los algoritmos para resolver la SDE de la exponencial bidimensional con $a = 1$, $b = 1$, $h = 0{,}01$ y $t \in [0,\, 1]$.

En el cuadro 4.2 se muestra que todos los métodos convergen a la solución exacta.

4.3. Ejemplo 3: Ecuación estocástica de Duffing

Las soluciones fuertes de SDE son importantes al modelar sistemas físicos que cruzan una transición de fase desde un estado espacialmente uniforme a un estado de menor simetría [7] Un parámetro de bifurcación tiene un valor crítico en el cual se produce un intercambio entre ambos estados, pero el ruido tambien puede cambiar la simetría inicial. Un ejemplo aparece al estudiar la dinámica de la SDE de Duffing

$$dX_t = \left(\mu t X_t - X_t^3\right) dt + b\, dW_t \qquad X(0) = 0{,}0 \qquad t \in [0,\, 200]\,.$$

En el caso en que $b \gg \sqrt{\mu}$ las trayectorias oscilan entre $\sqrt{\mu t}$ y $-\sqrt{\mu t}$ antes de seguir una sola de las ramas.

Los cuatro métodos convergen a la solución en forma muy similar a como se muestra en la figura 4.1.

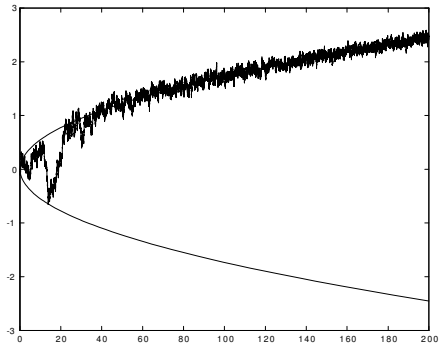

Figura 4.1: Gráfica de y vs el problema estocástico Duffing utilizando el método de Milstein-Platen con $\mu = 0{,}03$, $b = 0{,}25$, $h = 0{,}01$ y $t \in [0,\ 200]$.

4.4. Ejemplo 4: Modelo de tasa de interés de Cox-Ingersoll-Ross (CIR)

Se trata de un modelo para tasas de interes estocásticas [42] [43] que también se utiliza como una alternativa al movimiento browniano geométrico que aparece en el modelo de Black-Scholes. En este caso el proceso de tasa de interés $X(t)$ es la solución de la ecuación

$$dX(t) = [a + bX(t)]\,dt + \sigma\sqrt{X(t)}dW(t) \qquad X(0) = 1{,}0 \quad t \in [0,\ 1]$$

con parámetros reales $a \geq 0$, $b \in \mathbb{R}$ y $\sigma > 0$. Es bien conocido que la solución fuerte es única y preserva la no negatividad de los datos iniciales, lo cual puede que no suceda con cada una de las trayectorias calculadas por ciertos métodos numéricos [42]. Dependiendo de los valores específicos de los parámetros a, b y σ se pueden presentar diferentes comportamientos en la frontera :

- Si $a \geq \frac{\sigma^2}{2}$ entonces $X(t) > 0$ si $x_0 > 0$ porque no se puede alcanzar el valor en la frontera $X(t) = 0$

- Si $a < \frac{\sigma^2}{2}$ existen infinitos valores de $t > 0$ para los cuales $X(t) = 0$. Cuando una trayectoria alcanza el 0 regresa de inmediato a ese valor.

  Al implementar este método descartamos las trayectorias que generan $X(t)$ negativos porque no se podría en este caso calcular la solución exacta.

Los resultados por cualquiera de los cuatro métodos muestran que la solución numérica conserva las propiedades cualitativas de la solución exacta.

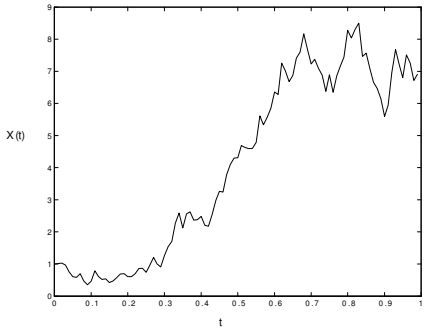

Figura 4.2: Gráfica de y vs el modelo de tasa de interés de Cox utilizando el método de Euler-Maruyama con $a = 1{,}0$, $b = 1{,}0$, $h = 0{,}01$ y $t \in [0, 1]$.

4.5. Ejemplo 5: SDE solucionable en forma explícita

Kloeden y Platen [29] presenta la solución explícita de la SDE

$$dX_t = 1\,dt + 2\sqrt{X_t}dW_t \qquad X(0) = 1{,}0 \qquad t \in [0, 0{,}5]$$

como

$$X_t = \left(W_t + \sqrt{X_0}\right)^2$$

Algoritmo	$\hat{\epsilon}$	$\Delta\epsilon$
Euler-Maruyama explícito vectorizado SDEBLAS	7.72650e-2	3.35100e-3
Euler-Maruyama implícito vectorizado SDEBLAS	7.72650e-2	3.35100e-3
Milstein-Platen explícito vectorizado SDEBLAS	2.39560e-2	4.26900e-3
Runge-Kutta implícito vectorizado SDEBLAS	2.39560e-2	4.26900e-3

Cuadro 4.3: Comparación del error y tasa de convergencia de los algoritmos para resolver la SDE solucionable en forma explícita con $h = 0{,}01$ y $t \in [0, 0{,}5]$

En el cuadro 4.3 se muestra se muestra que todos los métodos convergen a la solución exacta.

4.6. Ejemplo 6: SDE resuelta en dos etapas

Burrage presenta un metodo tipo Runge-Kutta de dos etapas para SDE y lo ilustra con la siguiente SDE no lineal expresada en la forma de Stratonovich

$$dX = -\alpha\left(1-X^2\right)\,dt + \beta\left(1-X^2\right)\circ dW\left(t\right), \qquad X(0) = 0{,}5 \qquad t \in [0,1]\,.$$

La solución exacta de esta ecuación es

$$X\left(t\right) = \frac{\left(1+X_0\right)\exp\left(-2\alpha t + 2\beta W\left(t\right)\right) + X_0 - 1}{\left(1+X_0\right)\exp\left(-2\alpha t + 2\beta W\left(t\right)\right) - X_0 + 1}$$

Algoritmo	$\hat{\epsilon}$	$\Delta\epsilon$
Euler-Maruyama explícito vectorizado SDEBLAS	5.31295e-1	3.22020e-2
Euler-Maruyama implícito vectorizado SDEBLAS	5.28827e-1	3.20890e-2
Milstein-Platen explícito vectorizado SDEBLAS	5.35559e-1	3.01870e-2
Runge-Kutta implícito vectorizado SDEBLAS	5.32596e-1	3.02980e-2

Cuadro 4.4: Comparación del error y tasa de los algoritmos para resolver la SDE resuelta mediante el método de Runge-Kutta de dos etapas con $\alpha = 5{,}0$, $\beta = -5{,}0$, $h = 0{,}001$ y $t \in [0,\ 1]$.

En el cuadro 4.4 se muestra que todos los métodos convergen a la solución exacta si bien se requiere un paso de $h = 0{,}001$ para alcanzar los resultados medianamente aceptables que se presentan. Se puede mejorar la precisión disminuyendo aun más el paso pero esto solo será factible hacerlo si hay capacidad de computo disponible.

5
Notas sobre la literatura

No existe, a juicio al autor, una buena referencia desde el punto didáctico que proporcione un tratamiento introductorio sobre las ecuaciones diferenciales estocásticas. La única obra de este género que merece mención es *Numerical solution of SDE through computer experiments* de Kloeden y Platen, 2003. Este libro se puede complementar con *Numerical Solution of Stochastic Differential Equations*, Kloeden y Platen, 1995, que sigue siendo la obra más completa de la materia aunque no trata en profundidad ciertos tópicos importantes. Una referencia más matemática puede encontrarse en *Stochastic Differential Equations* de Oksendal, 1998, la cual es a juicio del autor la mejor referencia sobre aplicaciones en ingeniería y finanzas. Sobre aplicaciones a la biología *Numerical methods for strong solutions of stochastic differential equations: an overview*, Burrage et al, 2004, contiene una buena perspectiva sobre el estado actual de investigación en este campo.

Dos artículos importantes sobre el estado del arte en métodos numéricos de solución de SDE son *An introduction to numerical methods for stochastic differential equations*, Platen, 1999 y *Numerical methods for strong solutions of stochastic differential equations: an overview*, Burrage et al, 2004. Debido a la diversidad de problemas numéricos que aparecen en el estudio de las SDE, es importante traer la amplia experiencia que existe en la solución de ODE a la solución de SDE. Para una importante revisión del estado del arte en ODE véase Gupta [26]. Una revisión sobre el cálculo de integrales múltiples de Stratonovich e Ito puede encontrarse en el breve pero admirable libro de Milstein, *Numerical Integration of Stochastic Differential Equations*, 1995.

Referencias para BLAS pueden encontrarse en Lawson [32], Dongarra [19]y Dongarra [22], así como en el anexo del libro de Andersen [1] y en *www.netlib.org* .

El lector interesado en la implementación de las librerías de vectorización puede consultar Andersen (1995).

6
Conclusiones

- BLAS y LAPACK permiten programar librerías científicas de alta confiabilidad porque simplifican y estandarizan el código al no tener que reescribir nuevamente muchas rutinas del álgebra lineal. Al mismo tiempo se alcanza alta eficiencia porque estas librerías incorporan muchos de los avances en algoritmos que se han desarrollado en las últimas décadas. En la prueba explicada en el ejemplo 1 para la exponencial unidimensional, SDEBLAS fue aproximadamente 22 veces más rápido que otra rutina publicada [27] para el método de Euler-Maruyama.

- Los cuatros métodos implementados convergen en una variedad de ejemplos y aplicaciones, si bien algunas veces pueden converger más lentamente.

- Observando el desempeño de los métodos en los ejemplos analizados en esta tesis, podemos concluir que, en general, el método de Milstein Platen y el método de Runge-Kutta implícito tienen mejor precisión que los métodos de Euler Maruyama explícito y Euler Maruyama implícito.

- Al comparar los resultados del método Londoño-Villegas [52] utilizando los mismos ejemplos explicados anteriormente para SDEBLAS, encontramos que los valores $\hat{\epsilon}$ y $\Delta\epsilon$ del método Londoño-Villegas [52] son menores por varios ordenes de magnitud que los correspondientes valores de los métodos implementados en SDEBLAS. En consecuencia, si se desea resolver una SDE se recomienda utilizar el método

Londoño-Villegas en primer lugar y utilizar a continuación los métodos implementados en SDEBLAS para confirmar que el primer método converge a la solución.

7

Notación

Ω	espacio muestral
ω	evento
$a \in A$	a es un elemento del conjunto A
P	medida de probabilidad
A	$\sigma-$ álgebra
$\varnothing$	conjunto vacio
$A \cap B$	intersección de los conjuntos A y B
$A \cup B$	unión de los conjuntos A y B
$a.s.$	casi seguramente
$w.p$,1.	con probabilidad 1
SDE	ecuación diferencial estocástica
ODE	ecuación diferencial ordinaria
$P(. \mid .)$	probabilidad condicional
$\mathbb{R}^d$	espacio euclidiano de dimensión d
F_x	función de distribución de la variable aleatoria X
$[a, b]$	intervalo cerrado desde a hasta b
(a, b)	intervalo abierto desde a hasta b
$U(a, b)$	uniformemente distribuido en $[a, b]$
exp(.)	función exponencial
e	número de Euler
mod c	módulo c
F^{-1}	inversa de la función F
ínf	ínfimo
sup	supremo

$\log_a$	logaritmo en base a
$\ln(\cdot)$	logaritmo en base e
$\mid \cdot \mid$	norma euclideana
$E(X)$	esperanza de X
$Var(X)$	varianza de x
$\hat{\mu}_N$	media muestral
$\hat{\sigma}^2_N$	varianza muestral
$N(\mu, \sigma^2)$	distribuido gaussianamente con media μ y varianza σ^2
$E(X \mid A)$	esperanza condicional de X dado A
$cov(X_1, X_2)$	covarianza de X_1y X_2
∞	infinito
$Lim_{n\to\infty}$	límite cuando tiende a infinito
$100\alpha\,\%$	nivel de significación
$W(t), W_t$	proceso de Wiener en el tiempo t
W^j_t	$j-simo$ componente de un proceso de Wiener W en el tiempo t
W^0_t	primer componente de un proceso de Wiener W en el tiempo t
$\int ...dt$	integral (Lebesgue, Riemman)
$\int ...dW_t$	integral de Ito
$\int ... \circ\, dW_t$	integral de Stratonovich
Δ, δ, Δ_n	dimensión del paso del tiempo
$a(t, X_t)$	coeficiente de deriva de una ecuación diferencial estocástica
$a^i(\cdot)$	$i-simo$ componente de la deriva
$b^j(t, X_t)$	coeficiente de difusión correspondiente a W^j
$b^{i,j}(\cdot)$	$i-simo$ componente del coeficiente de difusión $b^j(\cdot)$
T	tiempo de terminación
$e^{strong}_{\Delta t}$	error de la convergencia fuerte en el punto final
X^T	transpuesta de la matriz X
$\underline{a}(\cdot)$	coeficiente de deriva corregido de Stratonovich
$\frac{\partial}{\partial x}a$	derivada parcial de a con respecto a x
$\frac{\partial^2}{\partial x_i \partial x_j}a$	derivada parcial de segundo orden de a
$l!$	factorial de l
L^0, L^1, L^j	operadores diferenciales
$\underline{L}^0$, $\underline{L}^1$, $\underline{L}^j$	operadores diferenciales
$J_{(j_1,j_2,...,j_t),t}$	integral multiple de Stratonovich
máx$\{\cdot,\cdot\}$	máximo
$\Delta W_n, \Delta W^j_n$	incremento del proceso de Wiener
$\Delta Z_n, \Delta Z^j_n$	integral múltiple estocástica
$I_{(j_1,j_2,...j_t),t}$	integral múltiple de Ito
τ_n	tiempo de discretización
Y_n	aproximación Y en el tiempo de discretización τ_n
η_t	entero más grande para el cual $\tau_n \leq t$
ϵ	error absoluto
$\hat{\epsilon}$	estimado para el error absoluto ϵ
μ	error medio

$\hat{\mu}$	estimado para el error medio μ
$a \ll b$	b es mucho más grande que a
i	raíz cuadrada de -1
$\mathrm{Re}(\lambda)$	parte real del número complejo λ
$\mathrm{Im}(\lambda)$	parte imaginaria del número complejo λ
d	dimensión del proceso de Ito X
m	dimensión del proceso de Wiener W
$a, b, ...$	abreviaciones de $a(Y_n), b(Y_n)$
I	matriz identidad
V_{j_1,j_2}	variable aleatoria para integrales de Wiener dobles
$\prod_{i=1}^{N} a_i$	producto
$\sum_{i=1}^{N} a_i$	suma
(a, b)	producto escalar
$\det A$	determinante de la matriz A
A^{-1}	inversa de la matriz A
∇f	vector gradiente de la función f
$\#A$	cardinalidad del conjunto A
a^+	parte positiva de a
$\equiv$	identicamente igual a
$\approx$	aproximadamente igual a

8
Anexos

8.1. Indice de rutinas de SDEBLAS

Rutina	**Descripción**
fi_block_01	Rutina auxiliar para calcular una secuencia de pasos que se usan en todos los métodos
fi_block_02	Rutina auxiliar para calcular una secuencia de pasos de los métodos de Euler-Maruyama implícito y Runge-Kutta implícito
fi_block_03	Rutina auxiliar para calcular una secuencia de pasos de los métodos Milstein-Platen y Runge-Kutta implícito
fi_broyden	Rutina auxiliar para calcular raices multidimensionales con el método de Broyden
fi_jacobian	Rutina auxiliar para aproximar el Jacobiano en el método de Broyden
fi_euler_maruyama	Rutina principal para aproximar una SDE multidimensional con el método de Euler-Maruyama explícito de orden 1.0
fi_euler_maruyama_implicit	Rutina principal para aproximar una SDE multidimensional con el metodo de Euler-Maruyama implícito de orden 1.0
fi_evaluate_implicit	Rutina auxiliar para evaluar implícitamente una SDE
fi_ijito	Rutina auxiliar para calcular integrales de Ito dobles
f_implicit_data_free	Rutina auxiliar para liberar memoria del tipo de dato utilizado en los métodos implícitos
f_implicit_data_malloc	Rutina auxiliar para asignar memoria a un tipo de dato utilizado en los métodos implícitos
fi_ijstratonovich	Rutina auxiliar para calcular integrales de Stratonovich dobles
fi_jito	Rutina auxiliar para calcular integrales de Ito simples
fi_jstratonovich	Rutina auxiliar para calcular integrales de Stratonovich simples
fi_line_search	Rutina auxiliar para calcular raices multidimensionales con el metodo de Broyden
fi_milstein_platen	Rutina principal para aproximar una SDE multidimensional con el método de Milstein-Platen explícito de orden 1.0
fi_qrupdate	Rutina auxiliar para calcular raices multidimensionales con el método de Broyden
fi_rotate	Rutina auxiliar para calcular raices multidimensionales con el método de Broyden
fi_runge_kutta_implicit	Rutina principal para aproximar una SDE multidimensional con el metodo de Runge-Kutta implícito de orden 1.0
fi_wiener	Rutina auxiliar para construir un proceso de Wiener

8.2. Descripción de las rutinas de SDEBLAS

8.2.1. fi_euler_maruyama

Argumento	**Descripción**
fi_a fi_asde	(Entrada: fi_a) Referencia a la función que genera la matriz de corrimiento a de $n \times 1$
fi_b fi_bsde	(Entrada: fi_b) Referencia a la función que genera la matriz de difusión b de $m \times n$
fi_alfa fi_alfasde	(Entrada: fi_alfa) Referencia a la funcion α en la ecuación $dW = \alpha dt + dB_t$ que permite generar ruidos de diferente tipo
fi_exact fi_solution	Referencia a la función que genera la solución exacta utilizando la misma realización del proceso de Wiener
int i_dimension	(Entrada: entero) Dimensión n de la SDE
int i_wienerdimension	(Entrada: entero) Dimensión m del proceso de Wiener
int i_simulation	(Entrada: entero) Numero de simulaciones
ctss_real r_tstart	(Entrada: ctss_real) Tiempo en que inicia la simulación, usualmente 0.0
ctss_real r_tfinish	(Entrada: ctss_real) Tiempo en que termina la simulación, usualmente 1.0
ctss_real r_stepsize	(Entrada: ctss_real) Longitud del paso
ctss_stream_state_ptr s_stream	(Entrada: ctss_stream_state_ptr) Abstracción de una secuencia aleatoria, que contiene un puntero a void con el estado actual del generador aleatorio que se usará para generar el proceso de Wiener
ctss_real *ar_x0	(Entrada: ctss_real) Matriz de $n \times 1$ que contiene el vector inicial de la simulación
ctss_real *ar_xfexpected	(Entrada/Salida: ctss_real) Matriz de $n \times 1$ que contiene el vector final esperado de la simulación
void *v_adata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_asde
void *v_bdata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_bsde
void *v_exactdata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_solution

8.2.2. fi_milstein_platen

Argumento	**Descripción**
fi_a fi_asde	(Entrada: fi_a) Referencia a la función que genera la matriz de corrimiento a de $n \times 1$
fi_b fi_bsde	(Entrada: fi_b) Referencia a la función que genera la matriz de difusión b de $m \times n$
fi_alfa fi_alfasde	(Entrada: f_alfa) Referencia a la funcion α en la ecuación $dW = \alpha dt + dB_t$ que permite generar ruidos de diferente tipo
fi_exact fi_solution	Referencia a la función que genera la solución exacta utilizando la misma realización del proceso de Wiener
int i_dimension	(Entrada: entero) Dimensión n de la SDE
int i_wienerdimension	(Entrada: entero) Dimensión m del proceso de Wiener
int i_simulation	(Entrada: entero) Número de simulaciones
int i_p	(Entrada: entero) Truncamiento que se utilizará para aproximar las integrales de Stratonovich. Utilice un valor menor o igual a 8.
int i_straito_option	(Entrada: entero) 0 si se desea correr una SDE en forma de Ito, 1 en forma de Stratonovich
ctss_real r_tstart	(Entrada: ctss_real) Tiempo en que inicia la simulación, usualmente 0.0
ctss_real r_tfinish	(Entrada: ctss_real) Tiempo en que termina la simulación, usualmente 1.0
ctss_real r_stepsize	(Entrada: ctss_real) Longitud del paso
ctss_stream_state_ptr s_stream	(Entrada: ctss_stream_state_ptr) Abstracción de una secuencia aleatoria, que contiene un puntero a void con el estado actual del generador aleatorio que se usará para generar el proceso de Wiener
ctss_stream_state_ptr s_stream_integral	(Entrada: ctss_stream_state_ptr) Abstracción de una secuencia aleatoria, que contiene un puntero a void con el estado actual del generador aleatorio que se usará para generar las integrales estocásticas
ctss_real *ar_x0	(Entrada: ctss_real) Matriz de $n \times 1$ que contiene el vector inicial de la simulación
ctss_real *ar_xfexpected	(Entrada/Salida: ctss_real) Matriz de $n \times 1$ que contiene el vector final esperado de la simulación
void *v_adata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_asde
void *v_bdata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_bsde
void *v_exactdata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_solution

8.2.3. fi_euler_maruyama_implicit

Argumento	**Descripción**
fi_a fi_asde	(Entrada: fi_a) Referencia a la función que genera la matriz de corrimiento a de $n \times 1$
fi_b fi_bsde	(Entrada: fi_b) Referencia a la función que genera la matriz de difusión b de $m \times n$
fi_alfa fi_alfasde	(Entrada: fi_alfa) Referencia a la funcion α en la ecuación $dW = \alpha dt + dB_t$ que permite generar ruidos de diferente tipo
fi_exact fi_solution	Referencia a la función que genera la solución exacta utilizando la misma realización del proceso de Wiener
int i_dimension	(Entrada: entero) Dimensión n de la SDE
int i_wienerdimension	(Entrada: entero) Dimensión m del proceso de Wiener
int i_simulation	(Entrada: entero) Número de simulaciones
ctss_real r_tstart	(Entrada: ctss_real) Tiempo en que inicia la simulación, usualmente 0.0
ctss_real r_tfinish	(Entrada: ctss_real) Tiempo en que termina la simulación, usualmente 1.0
ctss_real r_stepsize	(Entrada: ctss_real) Longitud del paso
ctss_stream_state_ptr s_stream	(Entrada: ctss_stream_state_ptr) Abstracción de una secuencia aleatoria, que contiene un puntero a void con el estado actual del generador aleatorio que se usará para generar el proceso de Wiener
ctss_stream_state_ptr s_stream_integral	(Entrada: ctss_stream_state_ptr) Abstracción de una secuencia aleatoria, que contiene un puntero a void con el estado actual del generador aleatorio que se usará para generar las integrales estocásticas
ctss_real *ar_x0	(Entrada: ctss_real) Matriz de $n \times 1$ que contiene el vector inicial de la simulación
ctss_real *ar_xfexpected	(Entrada/Salida: ctss_real) Matriz de $n \times 1$ que contiene el vector final esperado de la simulación
ctss_real *ar_implicit	(Entrada: ctss_real) Matriz diagonal de $n \times n$ en donde $0 \leq \alpha_{kk} \leq 1$ indica que tan implícito es cada componente k de un método implícito
void *v_adata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_asde
void *v_bdata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_bsde
void *v_exactdata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_solution

8.2.4. fi_runge_kutta_implícito

Argumento	**Descripción**
fi_a fi_asde	(Entrada: fi_a) Referencia a la función que genera la matriz de corrimiento a de $n \times 1$
fi_b fi_bsde	(Entrada: fi_b) Referencia a la función que genera la matriz de difusión b de $m \times n$
fi_alfa fi_alfasde	(Entrada: f_alfa) Referencia a la funcion α en la ecuación $dW = \alpha dt + dB_t$ que permite generar ruidos de diferente tipo
fi_exact fi_solution	Referencia a la función que genera la solución exacta utilizando la misma realización del proceso de Wiener
int i_dimension	(Entrada: entero) Dimensión n de la SDE
int i_wienerdimension	(Entrada: entero) Dimensión m del proceso de Wiener
int i_simulation	(Entrada: entero) Número de simulaciones
int i_p	(Entrada: entero) Truncamiento que se utilizará para aproximar las integrales de Stratonovich. Utilice un valor menor o igual a 8.
ctss_real r_tstart	(Entrada: ctss_real) Tiempo en que inicia la simulación, usualmente 0.0
ctss_real r_tfinish	(Entrada: ctss_real) Tiempo en que termina la simulación, usualmente 1.0
ctss_real r_stepsize	(Entrada: ctss_real) Longitud del paso
ctss_stream_state_ptr s_stream	(Entrada: ctss_stream_state_ptr) Abstracción de una secuencia aleatoria, que contiene un puntero a void con el estado actual del generador aleatorio que se usará para generar el proceso de Wiener
ctss_stream_state_ptr s_stream_integral	(Entrada: ctss_stream_state_ptr) Abstracción de una secuencia aleatoria, que contiene un puntero a void con el estado actual del generador aleatorio que se usará para generar las integrales estocásticas
ctss_real *ar_x0	(Entrada: ctss_real) Matriz de $n \times 1$ que contiene el vector inicial de la simulación
ctss_real *ar_xfexpected	(Entrada/Salida: ctss_real) Matriz de $n \times 1$ que contiene el vector final esperado de la simulación
ctss_real *ar_implicit	(Entrada: ctss_real) Matriz diagonal de $n \times n$ en donde $0 \leq \alpha_{kk} \leq 1$ indica que tan implícito es cada componente k de un método implícito
void *v_adata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_asde
void *v_bdata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_bsde
void *v_exactdata	(Entrada: puntero a void) Apuntador a los datos adicionales que se requieran para evaluar fi_solution

8.2.5. fi_statistic

Argumento	**Descripción**
int i_batch	Número de bachadas
ctss_real *ar_xferror	Matriz de i_batch entradas que contiene la norma del vector de error de cada bachada
ctss_real *ar_statistic	Matriz que devuelve en la entrada 0 la media del error, en la entrada 1 la varianza del error y en la entrada el delta del error

8.3. Guía de instalación rápida

Para instalar SDEBLAS en un computador que corra el sistema operativo Linux, ejecute los siguientes pasos:

1. Instalación de BLAS: Puede correr SDEBLAS con una versión optimizada de BLAS para su computador o puede tambien correr SDEBLAS con la versión genérica de BLAS que instala LAPACK por defecto. En el primer caso debe descargar BLAS en la dirección *http://www.netlib.org* (verificada en enero de 2007) y seguir las instrucciones que aparecen en el archivo descomprimido. En el segundo caso, pase al siguiente numeral.

2. Instalación de LAPACK: Descárguelo en la dirección *http://www.netlib.org* (verificada en enero de 2007) y descomprimalo. Si esta utilizando una versión optimizada de BLAS debe configurar LAPACK como lo indica la guía de instalación de este último. Si no utiliza una versión optimizada de BLAS y planea usar la versión genérica que tiene incluida LAPACK el proceso de instalación se limita usualmente a descomprimir LAPACK e ingresar la instrucción Make.

3. Instalación de CTSS: CTSS se instala por defecto al instalar SDEBLAS. La versión genérica de SDEBLAS utiliza una versión de CTSS que genera los números aleatorios sin ayuda de la Mathematical Kernel Library (MKL) de Intel. Si desea utilizar los generadores de alta calidad incluidos en MKL, debera configurar a CTSS para que enlace esta librería.

4. Instalación de SDEBLAS: Descomprima el archivo e ingrese la orden Make . Esto compilará la librería SDEBLAS (libvsde.a) y la instalará en el directorio lib. Para correr los ejemplos, debera ingresar en el directorio example y abrir el archivo Make. Este archivo contiene un encabezado donde se enlazan las librerías LAPACK (lapack_LINUX.a) y BLAS (blas_LINUX.a). Deberá ajustar las rutas donde aparecen estos archivos de acuerdo con la localización donde instaló LAPACK y BLAS. Tenga en cuenta que los nombres lapack_LINUX.a y blas_LINUX.a son nombres estándar que se utilizan al instalar LAPACK pero pueden ser cambiados al efectuar esta instalación. Primero compile la librería abriendo el directorio ./software e ingresando la instrucción make en la terminal. Luego abra el directorio ./software/example e ingrese la instrucción make para compilar el ejemplo y enlazar la librería.

5. Pruebe la instalación: La instalación de LAPACK ya fue probada al instalar este paquete. Para probar la instalación de SDEBLAS abra la terminal de Linux, coloquese en el directorio example e ingrese

la instrucción ./run_sdeblas. Puede cambiar el ejemplo ejecutado copiando otro ejemplo desde el archivo example_all.c en el archivo example.c y cambiando los parámetros del ejemplo y el método en el archivo run_sdeblas.c.

8.4. Manual del usuario

8.4.1. Directorios que se deben utilizar

Lo más recomendable para correr el programa es mantener la estructura original de los archivos. Esto significa tener los siguientes directorios:

./software: En este directorio se incluirán los demás directorios y el archivo makefile que compila la libreria

./example: En este directorio aparecen los archivos run_sdeblas.c que corre el programa, el archivo example.c que contiene la información sobre como construir la SDE y los archivos data_euler_maruyama.txt, data_milstein_platen.txt, data_euler_maruyama_implicit.txt y data_runge_kutta_implicit.txt que contienen los datos de la última simulación que se corrió con cada método. También aparece el archivo makefile que compila el ejemplo enlazando la librería.

./source: Este directorio contiene el código fuente de SDEBLAS que se utiliza para compilar la librería. Salvo situaciones especiales, por ejemplo cambiar el método para calcular raices multidimensionales, este directorio no debería ser modificado por el usuario

./include: Este directorio contiene los archivos de encabezados de SDEBLAS que se utilizan para compilar la librería. Al igual que con el directorio ./source, este directorio no debería ser modificado por el usuario.

./lib: Contiene la librería compilada. Este directorio nunca debe ser modificado por el usuario.

8.4.2. Como construir los archivos que se utilizan para correr a SDEBLAS

A continuación aparece una descripción detallada sobre como construir los dos archivos necesarios para correr a SDEBLAS. Presentaremos el ejemplo de la SDE unidimensional

$$dX_t = \alpha X_t\,dt + \sigma X_t\,dW_t \qquad X(0) = 1{,}0 \qquad t \in [0,\,1]$$

cuya solución esta dada por

$$X(t) = x_0 \exp\left\{\left(\alpha - \frac{1}{2}\sigma^2\right)t + \sigma W(t)\right\}$$

utilizando los parámetros $\alpha = 1$, $\sigma = 1$, $h = 0{,}01$ y $t \in [0,\,1]$.

example.c

En primer lugar se deben incluir todos los siguientes archivos de encabezados los cuales permiten buscar los correspondientes archivos de SDEBLAS:

```
#include "sdeblas_types.h"
#include "sdeblas_error.h"
#include çtss_types.h"
#include çtss_error.h"
#include çtss_rng.h"
#include .auxiliar.h"
#include .example.h"
#include çtss_example.h"
#include çtss_rng.h"
```

A continuación aparecen algunas macros que se han encontrado útiles, pero que no necesariamente se requieren en el código:

```
#define Ith(A,m,n,i,j) A[m*j+i]
#define ALFA RCONST(1.9)
#define SIGMA RCONST(0.1)
#define ONE RCONST(1.0)
#define TWO RCONST(2.0)
#define SQUARE(x) ((x)*(x))
#define PI RCONST(3.14159265)
```

En el archivo example.c siempre deben aparecer cuatro funciones: fi_asde que construye el coeficiente de corrimiento, fi_bsde que construye el coeficiente de difusión, fi_asde que se utiliza para construir ciertos tipos de ruidos especiales y fi_solution que construye la solución explícita en caso de que exista.

Para calcular el siguiente incremento dX_t observamos que en la SDE utilizamos un coeficiente de corrimiento igual a αX_t, pero como en nuestro ejemplo $\alpha = 1$, tenemos que el coeficiente de corrimiento es X_t. Este valor ya lo conocemos y corresponde al valor de la trayectoria en el paso anterior. Por eso el código $ar\_yout[0] = ar\_xin[0]$ indica que el coeficiente de corrimiento $ar\_yout[0]$ toma el valor de la trayectoria en el paso anterior. La función fi_asde que aparece a continuación tiene otros parametros de entrada, tales como la dimensión de la SDE i_dimension y el paso que estamos calculando r_t, que no se están utilizando, pero que en otros ejemplos se pueden necesitar. El parámetro v_adata se utiliza para pasar otro tipo de información diferente a la dimensión de la SDE y el tamaño del paso. De manera que como en nuestro ejemplo no estamos utilizando estos tres parámetros, los podríamos haber suprimido. Pero casi siempre debemos tener en cuenta el parametro ar_xin y siempre debemos tener en cuenta el parámetro ar_yout que contendrá el coeficiente de corrimiento. Obsérvese que los parámetros ar_xin, ar_yout y v_adata se pasan como punteros, lo cual es un detalle muy importante porque implica que solo se debe pedir memoria una vez para estos vectores. Si esta condición no se cumpliera el código perdería velocidad significativamente.

```
int fi_asde      (
                              int i_dimension,
                              ctss_real r_t,
                              ctss_real *ar_xin,
                              ctss_real *ar_yout,
                              void *v_adata
                            )
{
     ar_yout[0] = ar_xin[0];
     return SDEBLAS_SUCCESS;
}
```

El código para el coeficiente de difusión fi_bsde se construye de manera muy similar al código para el coeficiente de corrimiento, solo que se incluye, aunque no se utiliza, el parámetro i_wienerdimension correspondiente a la dimensión del proceso de Wiener

```
int fi_bsde      (
                              int i_dimension,
                              int i_wienerdimension,
                              ctss_real r_t,
                              ctss_real *ar_xin,
                              ctss_real *ar_yout,
                              void *v_bdata
                            )
{
     ar_yout[0] = ar_xin[0];

     return SDEBLAS_SUCCESS;
}
```

SDEBLAS permite escoger el tipo de proceso de Wiener de la SDE, siguiendo la ecuación $dW_t = \alpha_t dt + d\beta_t$ donde β_t es un movimiento browniano (proceso de Wiener) y α_s es independiente de $\beta_t - \beta_s$ para todo $0 \leq s \leq t \leq T$. Si $\alpha_t = 0$ hablamos de ruido blanco y si $\alpha_t = -kW_t$ hablamos de ruido gaussiano coloreado. En este caso estamos generando ruido blanco:

```
int fi_alfasde      (
                              int i_wienerdimension,
                              ctss_real r_tstart,
                              ctss_real r_tfinish,
                              ctss_real r_stepsize,
                              ctss_real *ar_yout,
                              void *v_alfadata
                            )
```

```
{
    /*esta SDE tiene un proceso de Wiener unidimensional*/
    int i_i, i_j;
    int i_stepnumber;

    /*calculamos el numero de pasos*/
    i_stepnumber = fi_stepnum(r_tstart, r_tfinish, r_stepsize);

    for(i_i = 0; i_i < i_stepnumber ; i_i++)
    {
        for(i_j = 0; i_j < i_wienerdimension ; i_j++)
        {
            ar_yout[i_i*i_wienerdimension + i_j] = 0.0;
        }
    }

    return SDEBLAS_SUCCESS;
}
```

Para encontrar el error del algoritmo de convergencia fuerte, generamos la solución explícita. Si una SDE no tiene solución explícita de todas formas se debería escribir la función fi_solution y devolver un vector nulo, teniendo en cuenta que en este caso el error reportado por SDEBLAS no tendría sentido.

```
int fi_solution     (
                int i_dimension,
                int i_wienerdimension,
                int i_step,
                ctss_real r_t,
                ctss_real *ar_wienertotal,
                ctss_real *ar_xin,
                ctss_real *ar_yout,
                void *v_exactdata
            )
{
    ctss_real r_alfa, r_sigma, x0;

    r_alfa = RCONST(1.0);
    r_sigma = RCONST(1.0);
    x0 = RCONST(1.0);

    ar_yout[0] = x0 * exp((r_alfa - r_sigma*r_sigma/2)*r_t
+ r_sigma*ar_wienertotal[0]);
    return SDEBLAS_SUCCESS;
```

}

run_sdeblas.c

run_sdeblas.c es la rutina desde donde se correrá todo el codigo. En la rutina run_sdeblas.c aparece como correr todos los ejemplos, pero en la explicacion que sigue solo mostraremos como correr la exponencial unidimensional.

Para construir run_sdeblas.c debemos, al igual que en example.c, incluir siempre los siguientes archivos de encabezados que permitirán enlazar a SDEBLAS:

```
#include ."example.h"
#include run_sdeblas.h"
#include "sdeblas_error.h"
#include çtss_types.h"
#include çtss_error.h"
#include çtss_rng.h"
#include "wiener.h"
#include "blas.h"
#include "lapack.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

El siguiente archivo de encabezado solo se necesita incluir si de desea calcular la velocidad del código

```
#include <time.h>
```

La siguiente macro no es obligatoria pero usualmente es útil

```
#define PI RCONST(3.14159265)
```

Abrimos la rutina principal que debe tener todo programa en c y definimos algunas variables

```
main()
{

    /*variables generales*/
    int i_batch;
    int i_count;
    int i_i, i_j;
```

La siguiente variable contendrá los tamaños de las matrices con el fin de solicitarle memoria al sistema operativo y debe ser de tipo long con el fin de crear matrices de tamaños muy grandes

long l_memory;

Las siguientes variables, con el tipo especial clock solo se incluyen si se desea calcular la velocidad del código:
clock_t cl_start, cl_finish;

Creamos una variable especial que contendrá la información del generador aleatorio de CTSS para el proceso de Wiener:
ctss_stream_state_ptr s_stream;

Si el método numérico requiere el calculo de integrales de Stratonovich dobles necesitamos otra variable aleatoria:
ctss_stream_state_ptr s_stream_integral;

Para construir las semillas de las dos variables anteriores creamos otras variables que contienen un vector de tres o cuatro enteros positivos, los cuales pueden ser escogidos al azar:
unsigned int aui_seed_stream[] = {15,5,485,1};
unsigned int aui_seed_stream_integral[] = {27,9,100};

Para ingresar la información sobre la dimensión de la SDE, la dimensión del proceso de Wiener, el número de pasos y si se desea correr un método en su forma de Ito o Stratonovich, se utilizan las siguientes variables:

```
int i_dimension;
int i_wienerdimension;
int i_stepnumber, i_straito_option;
```

Las siguientes variables contendrán el tiempo en que se iniciarán y terminarán las simulaciones y el tamaño de cada paso. Son de tipo ctss_real, que son un tipo especial de variable definida en CTSS que permite cambiar la precisión entre float y double. SDEBLAS se ha configurado por defecto como double y en general esto no se deberia cambiar. Si se desea configurar el programa para trabajar como float se deben abrir los encabezados de CTSS, pero esto solo se puede hacer si el método es explícito
ctss_real r_tstart;
ctss_real r_tfinish;
ctss_real r_stepsize;

A continuación procedemos a declarar las variables que contendrán los vectores. Obsérvese que debido al manejo exigido por BLAS, nunca se declaran matrices. Todas las matrices se declaran como vectores colocando cada fila despues de la anterior. El vector que contiene el punto inicial es

ctss_real *ar_x0;

El valor esperado de cada bachada es
ctss_real *ar_xfexpected;

El valor esperado de todas las bachadas se incluye en el siguiente vector
ctss_real *ar_xfbatchexpected;

En cada corrida de SDEBLAS debe existir una solución explícita. Como esta solución no siempre existe, se debería hacer la solución explícita igual al vector nulo, lo cual supone que el error calculado entre una simulación y la solución explícita no tendría sentido. Esta información, tenga o no tenga sentido, se devuelve como la media de errores sobre una bachada en la siguiente variable:
ctss_real *ar_xerror;

Las siguientes variables no se utilizan frecuentemente y su objetivo es pasar cierta información necesaria para construir los coeficientes de corrimiento y difusion y la solución explícita, respectivamente:
ctss_real *ar_adata;
ctss_real *ar_bdata;
ctss_real *ar_exactdata;

La manera en que se ingresa la información sobre que tan implícito es un método se hace con la siguiente matriz diagonal
ctss_real *ar_implicit;

Los datos de error para cada bachadas aparecen en la siguiente variable
ctss_real *ar_xferror;

En el siguiente vector de tres entradas aparecerán los datos de la media del error, varianza del error y delta del error para todas las bachadas
ctss_real *ar_statistic;

Para ingresar el número de simulaciones para cada bachada utilizamos la siguiente variable
int i_simulation;

Para calcular las integrales dobles de Stratonovich se utiliza una variable que determina la precisión del cálculo
int i_p;

En algunas circunstancias los cálculos de raices multidimensionales que utilizan los métodos implícitos implementados en SDEBLAS pueden fallar. Para saber si ésto ha ocurrido, se debe usar la siguiente variable
int i_error;

Las siguientes variables se utilizan para algunos cálculos finales con BLAS. Pero si el usuario no conoce BLAS, puede suprimirlas y escribir un código sin BLAS

```
int i_n;
int i_incx, i_incy;
ctss_real r_alfa;
```

A continuación damos valor a algunas variables. El método de Milstein-Platen puede utilizarse en forma de Ito con la siguiente variable en 0, o en forma de Stratonovich, con la siguiente variable en 1. Debe tenerse en cuenta que el coeficiente de corrimiento construido en fi_asde debe tener la correspondiente forma de Ito o Stratonovich.

```
i_straito_option = 0;
```

El número de bachadas usualmente es 20

```
i_batch = 20;
```

Si queremos calcular la velocidad del programa, iniciamos el reloj

```
cl_start = clock();
```

Damos valores especificos para la SDE exponencial unidimensional que estamos calculando. Es importante no cometer errores con estos valores, porque se generará un error en tiempo de ejecución que puede ser dificil de depurar

```
i_dimension = 1;
i_wienerdimension = 1;
```

Escogemos el tiempo de inicio, de terminación y paso del método. La macro RCONST está definida en CTSS y tiene como objetivo transformar la constante en float o double, según como se haya escogido. De manera que todas las constantes que se definan en SDEBLAS deberían estar precedidas de RCONST.

```
r_tstart = RCONST(0.0);
r_tfinish = RCONST(1.00);
r_stepsize = RCONST(0.01);
```

El número de simulaciones por cada bachada es usualmente 100

```
i_simulation = 100;
```

Se recomienda el valor de 7 para esta variable porque permite calcular la integral doble de Stratonovich con precision aceptable

```
i_p = 7;
```

Solicitamos memoria para los vectores del punto de inicio y la matriz de implicitud. Este código es estándar y no requiere cambiarse si antes se definió i_dimension

```
l_memory = i_dimension;
ar_x0 = (ctss_real *) malloc ( l_memory * sizeof (ctss_real));

l_memory = i_dimension*i_dimension;
ar_implicit = (ctss_real *) malloc ( l_memory * sizeof (ctss_real));


l_memory = 1;
ar_adata = (ctss_real *) malloc ( l_memory * sizeof (ctss_real));
```

Solicitamos memoria para los vectores que pasan información especial para construir los coeficientes de corrimiento y de difusión y la solución explícita. Los tamaños dependeran de la cantidad de información que se deba pasar. Verifique cuidadosamente estos tamaños para evitar desbordamientos de memoria

```
l_memory = 1;
ar_bdata = (ctss_real *) malloc ( l_memory * sizeof (ctss_real));

l_memory = 1;
ar_exactdata = (ctss_real *) malloc ( l_memory * sizeof (ctss_real));
```

Definimos el punto de inicio de la simulación

```
ar_x0[0] = RCONST(1.0);
```

Si colocamos un valor de 1 el método será completamente implícito y si colocamos 0 no será implicito

```
ar_implicit[0] = RCONST(1.0);
```

De la siguiente manera se pasarian datos a las funciones fi_asde, fi_bsde y fi_solution

```
ar_adata[0] = RCONST(7.0);
ar_bdata[0] = RCONST(1.0);
ar_exactdata[0] = RCONST(0.0);
```

Ejecutamos el método de Runge-Kutta implícito el número de bachadas determinado

```
for(i_j = 0; i_j < i_batch ; i_j++)
{
i_error = fi_runge_kutta_implicit      (
                                        fi_asde,
                                        fi_bsde,
```

```
        fi_alfasde,
        fi_solution,
        i_dimension,
        i_wienerdimension,
        i_simulation,
        i_p,
        r_tstart,
        r_tfinish,
        r_stepsize,
        s_stream,
        s_stream_integral,
        ar_x0,
        ar_implicit,
        ar_xfexpected,
        ar_xerror,
        ar_adata,
        ar_bdata,
        ar_exactdata
);
```

Si no se presentó error, procedemos a calcular las estadísticas correspondientes

```
if (i_error == SDEBLAS_SUCCESS)
{
```

Obtenemos el error del punto final de la bachada que se corrió:

```
ar_xferror[i_j] = ar_xerror[i_stepnumber-1];
```

Sumamos los puntos finales de cada bachada para más adelante obtener el valor esperado del punto final. Si el usuario no conoce BLAS, este fragmento de código también puede escribirse sin BLAS. Simplemente estamos sumando el vector ar_xfexpected en ar_xfbatchexpected. Obsérvese lo que mencionamos anteriormente, no existen matrices en SDEBLAS, solo vectores. ar_xfexpected es una matriz de i_dimension * (i_stepnumber+1), pero la orden AXPY utiliza un vector con numero de filas igual a i_dimension * (i_stepnumber+1)

```
i_n = i_dimension * (i_stepnumber+1);
r_alfa = RCONST(1.0);
i_incx = 1;
i_incy = 1;
AXPY(&i_n, &r_alfa, ar_xfexpected, &i_incx, ar_xfbatchexpected,
&i_incy);
}
```

En otro caso tratamos de correr el método sin que se devuelva un error, si esto no es posible reportamos el problema

```
else
{
      i_j = i_j - 1;
      i_count = i_count + 1;
      if(i_count == 100)
      {
            printf(.Error matematico\n");
            return SDEBLAS_MATH_ERROR;
}
}
}
```

Obtenemos el valor esperado de la trayectoria para todas las bachadas. Si se desea escribir este código sin BLAS basta dividir todo el vector ar_xfbatchexpected entre i_batch

```
i_n = i_dimension * (i_stepnumber+1);
r_alfa = (RCONST(1.0))/i_batch;
i_incx = 1;
SCAL(&i_n, &r_alfa, ar_xfbatchexpected, &i_incx);
```

Calculamos las estadisticas e imprimimos los resultados

```
fi_statistic      (
                        i_batch,
                        ar_xferror,
                        ar_statistic
                  );
printf("Media error = %f\n", ar_statistic[0]);
printf("Varianza error = %f\n", ar_statistic[1]);
printf("Delta media error = %f\n", ar_statistic[2]);
```

Detenemos el reloj e imprimimos el tiempo

```
cl_finish =clock();
printf("tiempo = %f s\n", ((double) (cl_finish-cl_start))/CLOCKS_PER_SEC);
```

Liberamos la memoria de los vectores de estados de los generadores aleatorios

```
ctss_rng_delete_stream (s_stream);
ctss_rng_delete_stream (s_stream_integral);
```

Liberamos la memoria y salimos del programa

```
free(ar_implicit);
free(ar_adata);
```

```
free(ar_bdata);
free(ar_exactdata);
free(ar_x0);
free(ar_xfexpected);
free(ar_xfbatchexpected);
free(ar_xerror);
free(ar_xferror);
free(ar_statistic);
return;
}
```

References

[1] ANDERSON, E et al. *LAPACK: Users' Guide*. Philadelphia: Society for industrial and applied mathematics, 1995. 325 p.

[2] BAXENDALE, P. y HARRIS, T. *Isotropic stochastic flows*. Annals Probab, Vol 14, (1986), p 1155-1179.

[3] BJORK, T. ARBITRAGE THEORY IN CONTINUOUS TIME. 1 ed. New York: Oxford University Press, 1998. 312 p

[4] *BLAS Frequently Asked Questions (FAQ).* Disponible en: http://www.netlib.org/blas/faq.html

[5] BROWN, Robert. *A brief account of microscopical observations made in the months of june, july and august, 1827, on the particle contained in the pollen of plants; and the general existence of active molecules in organic and inorganic bodies*. Phil. Mag, Vol 4, (1828), p. 161-173.

[6] BURDEN, Richard y FAIRES, Douglas. *Análisis numérico*. México: Grupo Editorial Iberoaméricana S.A de C.V. 1985. 715 p.

[7] BURRAGE, K, BURRAGE, P y TIAN, T. *Numerical methods for strong solutions of stochastic differential equations: an overview*. Proc. R. Soc. Lond., Vol 460, (2004), p 373-402

[8] BURRAGE, Kevin y TIAN, Tianhai. *Predictor-Corrector methods of Runge Kutta-Type for stochastic differential equations*. SIAM J. Numer. Anal, Vol 40, (2002), p 1516-1537.

[9] BURRAGE, Kevin y TIAN, T. *Two-stage stochastic Runge-Kutta methods for stochastic differential equations*. BIT, Vol 42, N 3, (2002), p 625-643.

[10] BURRAGE, Kevin y TIAN, T. Implicit stochastic Runge-Kutta methods for stochastic differential equations. BIT, Vol 44, (2004), p 2139.

[11] BURRAGE, K y BURRAGE, P. *Order conditions of stochastic Runge-Kutta methods by B-Series*. SIAM J. Numer. Anal, Vol 38, (2000), p 1626-1640.

[12] BURRAGE, P. *Vectorized simulations of stochastic differential equations*. ANZIAM, Vol 45, (2004), p 350-363.

[13] BURRAGE, P. *Runge-Kutta methods for stochastic differential equations*. PhD Thesis. University of Queensland, 1999.

[14] BURRAGE, P. y BURRAGE, K. *A variable stepsize implementation for stochastic differential equations*. SIAM J. Sci. Comput., Vol 24, (2002), p 484-864.

[15] CAVERHILL, A., CHAPPEL, M. y ELWORTHY, K. *Characteristic exponents for stochastic flows*. Springer Lecture Notes in Mathematics Berlin: Springer, Vol 1158,1986.

[16] CYGANOWSKY S., GRUNE, L y KLOEDEN, P. *MAPLE for Stochastic Differential Equations*, Theory and Numerics of Differential Equations (Editors J.F. Blowey, J.P. Coleman, A.W. Craig). Berlin: Springer Verlag, 2001. p 127-178.

[17] CASTRO, R. El universo LATEX. 1 ed. Bogotá: Editorial Unibiblos, 2001. 402 p.

[18] COX, J. *Notes on option pricing I: Constant elasticity of variance difusions*. Journal of Portfolio Management Vol 22, (1996), p 15-17.

[19] DONGARRA, J., DU CROZ, J., HAMMARLING, S. y HANSON, R. *An extended set of FORTRAN Basic Linear Algebra Subprograms*. ACM Trans. Math. Soft, Vol 14, (1988), p 1-17.

[20] DONGARRA, J., DU CROZ, J., HAMMARLING, S. y HANSON, R. *Algorithm 656: An extended set of FORTRAN Basic Linear Algebra Subprograms*. ACM Trans. Math. Soft, Vol 14, (1988), p 18-32.

[21] DONGARRA, J., DU CROZ, J., HAMMARLING, S. y DUFF, S. *A set of Level 3 Basic Linear Algebra Subprograms*. ACM Trans. Math. Soft, Vol 16, (1990), p 1-17.

[22] DONGARRA, J., DU CROZ, J., HAMMARLING, S. y DUFF, S. *Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms.* ACM Trans. Math. Soft, Vol 16, (1990), p 18-28.

[23] GAINES, J. and LYONS T. *Variable step size control in the numerical solution of stochastic differential equations.* SIAM, J. Appl. Math, Vol 57, (1997), p 1455-1484.

[24] GOLDBERG, David. *What every computer scientist should know about floating-point arithmetic.* Computing Surveys, 1991. p 171-264.

[25] GOLUB, G. y VAN LOAN, C. *Matrix Computations.* 3 ed. Baltimore: The John Hopkins University Press, 1996. 694 p.

[26] GUPTA, G, SACKS-DAVIS, R. Y TISCHER, P. *A review of recent developments in solvings ODEs. Computing Surveys*, Vol 17, (1985), p 1-47.

[27] HIGHAM, Desmond. *An algorithmic introduction to numerical simulation of stochastic differential equations.* SIAM Review, Vol 43, (2001), p 525-546.

[28] JIMENEZ, J. *Simulation of stochastic differential equations through the local linearization method. A comparative study.* Journal of Statistical Physics, Vol. 94, Nos 3/4, (1999).

[29] KLOEDEN, Peter y PLATEN, Eckhard. *Numerical Solution of Stochastic Differential Equations.* 2 ed. Berlin: Springer-Verlag, 1995. 632 p.

[30] KLOEDEN, Peter, PLATEN, Eckhard y SCHURZ, Henri. *Numerical solution of SDE through computer experiments.* 3 ed. Berlin: Springer-Verlag, 2003. 292 p.

[31] KLOEDEN, P., PLATEN, E. y WRIGHT, I. *The approximation of multiple stochastic integrals.* J. Stoch. Anal. Appl, Vol 10, (1992), p 431-441.

[32] LAWSON, C., HANSON, R., KINKAID, D. y KROGH, F. *Basic Linear Algebra Subprograms for FORTRAN usage.* ACM Trans. Math. Soft, Vol 5, (1979), p 308-323.

[33] L'ECUYER, Pierre y SIMARD, Richard. *An object-oriented random number package with many long streams and substreams.* Operation Research, Vol 50, (2002).

[34] LISKE, H., PLATEN, E y WAGNER, W. *About mixed multiple Wiener integrals.* Preprint P-Math-23/82. IMath Akad. der Wiss. der DDR, Berlin, (1982).

[35] LONDONO, J. y VILLEGAS, M. A Wong-Zakai type approximation for solutions of stochastic differential equations. 2006. En preparación.

[36] MCCORMICK, Patrick. *Supernova collapse simulated on a GPU.* Disponible en: http://www.eetasia.com/ART_8800367816_480100_f578f99a_no.htm

[37] MATSUMOTO, M y NISHIMURA, T. *Mersenne-Twister: A 623-Dimensionally equidistributed uniform pseudo-random number generator.* ACM Trans. Model. Comput. Simul, Vol 8, (1998), p 3-30.

[38] MILSTEIN, G. *Approximate integration of stochastic differential equations.* Theory Probab. Appl., Vol 19, (1974), p 557-562.

[39] MILSTEIN, G. *Numerical integration of stochastic differential equations.* Netherlands: Kluwer Academic Publishers, 1995. 170 p.

[40] MISLTEIN, G, PLATEN, E y SCHURZ, H. *Balanced implicit methods for stiff stochastic systems.* SIAM, J. Numer. Anal, Vol 35, (1998), p 1010-1019.

[41] MILSTEIN, G, REPIN, Yu y TRETYAKOV, M. *Numerical methods for stochastic systems preserving symplectic structure.* SIAM, J Numer. Anal, Vol 40, (2002), p 1583-1604.

[42] MORO, Esteban y SCHURZ, Henri. *Non-negativity preserving numerical algorithms for SDE.* arXiv:math.NA/0509724, Vol 1, (2005), p 1-23.

[43] COX, J, INGERSOLL, J y ROSS, A. *A Theory of term structure of interest rates.* Econometrica. Vol 53 (1985), p. 385-407.

[44] NEWTON, Nigel. *Asymptotically efficient Runge-Kutta methods for a class of Ito and Stratonovich equations.* SIAM, J. Apl. Math, Vol 51, (1991), p 542-567.

[45] OKSENDAL, Bernt. *Stochastic differential equations.* 5 ed. Berlin: Springer-Verlag. 1998. 324 p.

[46] PLATEN, E. *Zur zeitdiskreten Approximation von Itoprozesse.* IMath Akad. der Wiss. der DDR, Berlin, (1984).

[47] PLATEN, Eckhard. *An introduction to numerical methods for stochastic differential equations. Acta Numerica*, (1999), p 197-246.

[48] PRESS, William et al. *Numerical recipes in C.* 2 ed. New York: Cambridge University Press. 1997. 994 p.

[49] SCHLICK, T. *Molecular modelling and simulation: an interdisciplinary guide*. Springer. 2002.

[50] SCHUSS, Z. *Theory and applications of stochastic differential equations*. Wiley. 1980.

[51] SCHURZ, Henri. *A brief introduction to numerical analysis of (ordinary) stochastic differential equations without tears*. Minneapolis: Institute for mathematics and its applications, 1999. 161 p.

[52] VILLEGAS, Mauricio. *CTSS Library documentation*. 2005.

[53] VILLEGAS, M y LONDONO, J. *Numerical performance of some type of Wong-Zakai type approximations for stochastic differential equations*. 2006. En preparación.

[54] www.wikipedia.com. Bajo la entrada Brownian motion.