MAPLE for Jump–Diffusion Stochastic Differential Equations in Finance

S. Cyganowski

Tipperary Institute, Cashel Road, Clonmel Co., Tipperary, Ireland

e-mail: scyganowski@tippinst.ie

L. Grüne and P. E. Kloeden

Fachbereich Mathematik, Johann Wolfgang Goethe–Universität D–60054 Frankfurt am Main, Germany

e-mail: gruene / kloeden@math.uni-frankfurt.de

February 5, 2002

Abstract

The occurrence of shocks in the financial market is well known and, since the 1976 paper of the Noble Prize laureate R.C. Merton, there have been numerous attempts to incorporated them into financial models. Such models often result in jump–diffusion stochastic differential equations. This chapter describes the use of MAPLE for such equations, in particular for the derivation of numerical schemes. It can be regarded as an addendum to the chapter in this book by Higham and Kloeden [5], which can be referred to for general background and additional literature on stochastic differential equations and MAPLE. All the MAPLE code in this paper as well as additional material can be obtained from the web site

`www.math.uni-frankfurt.de/~numerik/kloeden/maplesde/`

following the link related to this paper.

Contents

1 Introduction

A simple model which includes jumps in a financial model is described in the text book of Lamberton and Lapeyre [7], Chapter 7. Essentially, it consists of the usual Black–Scholes model described by the the scalar linear Ito stochastic differential equation (SDE)

$$dX_t = \mu X_t\, dt + \sigma X_t\, dW_t, \tag{1}$$

within a time interval $[\tau_n, \tau_{n+1})$ between jumps, with the jump in X_t at τ_n having magnitude

$$\Delta X_{\tau_n} = X_{\tau_n} - X_{\tau_n^-} = X_{\tau_n^-} U_n, \tag{2}$$

with $X_{\tau_n} = X_{\tau_n^-}(1 + U_n)$, where $X_{\tau_n^-} = \lim_{t \to \tau_n^-} X_t$ is the limit from the left, i.e., with $t < \tau_n$, and U_n is the relative magnitude of the jump.

The SDE (1) has the explicit solution

$$X_t = X_{t_0}\, e^{(\mu - \sigma^2/2)(t - t_0) + \sigma(W_t - W_{t_0})}.$$

on a time interval $[t_0, t]$ without jumps. In particular,

$$X_t = X_0\, e^{(\mu - \sigma^2/2)t + \sigma W_t}$$

for $t \in [0, \tau_1)$, so by continuity $X_{\tau_1^-} = X_0\, e^{(\mu - \sigma^2/2)\tau_1 + \sigma W_{\tau_1}}$,which gives

$$X_{\tau_1} = X_0\, e^{(\mu - \sigma^2/2)\tau_1 + \sigma W_{\tau_1}}\ (1 + U_1)$$

and consequently

$$X_t = X_{\tau_1}\, e^{(\mu - \sigma^2/2)(t - \tau_1) + \sigma(W_t - W_{\tau_1})} = X_0\, e^{(\mu - \sigma^2/2)t + \sigma W_t}\ (1 + U_1)$$

for $t \in [\tau_1, \tau_2)$. This procedure can be repeated to obtain an explicit solution after a finite number of jumps. Let N_t is the number of jumps that occur between time 0 and t, i.e.,

$$N_t = \begin{cases} 0 & \text{if} \quad 0 \le t < \tau_1, \\ \sum_{n \ge 1} n \mathbf{1}_{\{\tau_n \le t < \tau_{n+1}\}} & \text{if} \quad t \ge \tau_1, \end{cases}$$

Then the Black–Scholes SDE (1) with jumps at times τ_n of relative magnitudes U_n has the explicit solution

$$X_t = X_0 \, e^{(\mu - \sigma^2/2)t + \sigma W_t} \prod_{n=1}^{N_t} (1 + U_n) \,, \tag{3}$$

with the convention that $\prod_{n=1}^{0} = 1$. Further details can be found in [7], pages 143–144 and pages 167–168.

The explicit solution (3) contains three stochastic processes as inputs. The continuous time Wiener process W_t has already been discussed in the Higham and Kloeden chapter [5], where the reader can find its definition and properties. The others are the jump times τ_n (or alternatively, the continuous time stochastic process N_t indicating the number of jumps until time t) and the relative jump magnitudes U_n. These three processes are assumed to be independent of each other.

It is typically assumed that the jump times τ_n are independent of each other and are identically exponentially distributed with parameter $\lambda > 0$, i.e., with density $\mathbf{1}_{\{t>0\}} \, \lambda e^{-\lambda t}$. This means that the stochastic process N_t is a Poisson process, so

$$\mathbf{P}(N_t = n) = e^{-\lambda t} \frac{(\lambda t)^n}{n!}$$

with mean value $\mathbf{E}(N_t) = \lambda t$ and variance $\mathrm{Var}(N_t) = \lambda t$. Moreover, N_t is stationary and has independent increments. See e.g., see [3, 6].

The relative jump magnitudes U_n are also assumed to be independent and identically distributed. As indicated in Exercise 42 on page 159 of [7], there are several useful possible distributions for U_n, such as two-point or lognormal distributed on $[-1, \infty)$. In the first case the U_n take just two possible values a and b, with probabilities

$$\mathbf{P}(U_n = a) = p, \qquad \mathbf{P}(U_n = b) = 1 - p,$$

while in the second case the U_n have the same distribution as $e^{G_n}-1$ where G_n is Gaussian distributed with some mean μ_G and variance σ_G^2. For lognormal distributed relative jump magnitudes, $G_n = \ln(1+U_n)$ and thus the explicit solution (3) can be written alternatively as

$$X_t = X_0\, e^{(\mu-\sigma^2/2)t+\sigma W_t+\sum_{n=1}^{N_t} G_n}. \tag{4}$$

Section 4 contains an implementation of this formula in MAPLE.

2 Jump–diffusion SDEs

The simple example discussed in the Introduction contains the basic features of financial models which include jump effects that have been investigated since the pioneering work of Merton [11]. It is often more convenient, theoretically at least, to include the jump mechanism in the differential equation itself. For models like that in the Introduction this gives rise to a jump–diffusion SDE. In the scalar case, the general form of a jump-diffusion SDE reads

$$dX_t = a(t, X_t)\, dt + b(t, X_t)\, dW_t + c(t, X_{t^-})\, dN_t, \tag{5}$$

where $a(t,x)$ is the drift coefficient , $b(t,x)$ the diffusion coefficient and $c(t,x)$ the jump magnitude coefficient. As before W_t is Wiener process and N_t is an inhomogeneous Poisson counting process.

The jump–diffusion SDE (5) is interpreted as a stochastic integral equation

$$X_t = X_{t_0} + \int_{t_0}^{t} a\,(s, X_s)\; ds + \int_{t_0}^{t} b\,(s, X_s)\; dW_s(w) + \int_{t_0}^{t} c\,(s, X_{s^-})\; dN_s, \tag{6}$$

where the first integral is a deterministic Riemann integral, the second is a stochastic Ito integral and the third is a stochastic integral with respect to a Poisson counting process or, more generally, Poisson random measure [4]. The existence and pathwise uniqueness of a solution process X_t of (5) follows under the usual growth restriction, uniform Lipschitz, and smoothness conditions on the coefficient functions a, b and c, see [4].

Essentially, the jump–diffusion SDE (6) acts as a normal Ito SDE between jumps. Since the solutions are continuous from the left, one obtains

$$X_{t^-} = X_{t_0} + \int_{t_0}^{t} a\,(s, X_s)\; ds + \int_{t_0}^{t} b\,(s, X_s)\; dW_s(w),$$

and hence

$$X_t = X_{t^-} + c\,(t, X_{t^-})\ \Delta N_t$$

where ΔN_t is the integer jump in N_t at time t, if any, and $c\,(t, X_{t^-})$ is the magnitude of the jump.

As an example, consider the linear jump–diffusion SDE

$$dX_t = \mu X_t\,dt + \sigma X_t\,dW_t + \gamma X_{t^-}\,dN_t, \tag{7}$$

which, from the discussion above, has the explicit solution

$$X_t = X_0\,e^{(\mu-\sigma^2/2)t+\sigma W_t}\ (1+\gamma)^{N_t}\,, \tag{8}$$

Here γ is the nonrandom constant relative jump magnitude. Generalizations of the jump–diffusion (5) allow random jump coefficient $c(t, x)$, i.e., a random relative jump magnitude coefficient γ in (7) as occurs in the Black–Scholes model with jumps considered in the introduction.

Since explicit solutions jump-diffusion stochastic differential equations are rarely known, numerical schemes are required. These can be derived systematically as in the jump free case [5, 6] from stochastic Taylor expansions. Such expansions have been obtained for jump–diffusion SDEs by Mikulevicius and Platen [12], based on iterated applications of the Ito formula for jump–diffusion SDEs.

3 Numerical schemes for jump–diffusion SDEs

MAPLE procedures for various numerical schemes proposed by Maghsoodi [8] for jump–diffusion SDE of the type (5) will be presented in this section. Additional MAPLE procedures for other schemes including some of even higher order can be found in [2].

The schemes considered here will be for an $t \in [t_0, T]$ and a partition $t_0 < t_1 < \cdots < t_n < t_{n+1} < \cdots < t_{N_T} = T$ with stepsize $\Delta_n = t_{n+1} - t_n$ for the nth subinterval $[t_n, t_{n+1}]$. Let Y_n denote the approximation to the solution X_t at t_n and let ΔW_n and ΔN_n denote the increments of the Wiener process W_t and the Poisson counting process N_t, respectively, over the subinterval

$[t_n, t_{n+1}]$.

Note that in [2, 8, 9] the increments ΔW_n and ΔN_n are written ΔW_{n+1} and ΔN_{n+1}, i.e., with index $n+1$ instead of n. The change made here is for consistency with usual practice, in particular with the the notation used in the Chapter [5] of this book

3.1 Scalar jump–diffusion SDEs

Maghsoodi [8] generalized the Euler scheme, the simplest numerical scheme, to scalar jump–diffusion SDE (5), obtaining

$$Y_{n+1} = Y_n + a(t_n, Y_n)\,\Delta_n + b(t_n, Y_n)\,\Delta W_n + c(t_n, Y_n)\,\Delta N_n \qquad (9)$$

for n = 0,1, ...,$N_T - 1$. He showed that it is of first order in the mean square sense, i.e. $O(h)$ where $h = \max_{n=0,\ldots N_T-1} \Delta_n$, which is equivalent to the strong order $\gamma = \frac{1}{2}$ used in [5, 6].

A MAPLE procedure which returns a stochastic Euler scheme with constant time stepsize Δt the scalar jump–diffusion SDE (5) is given below. Here Δt, ΔW etc. are denoted by dt, dW etc.

```
Euler_jump:=proc(a::algebraic,b::algebraic,c::algebraic)
  local soln,h;
  soln:=Y[n+1]=Y[n]+a*dt+b*dW[n]+c*dN[n];
  soln:=subs(x=Y[n],soln)
end:
```

In this procedure the input functions a, b and c are required to be functions of a variable x. An example for the usage of this procedure can be found in Section 4.

Maghsoodi [8] also derived schemes of higher mean square order than the above Euler scheme (9). These are in principle based on appropriate stochastic Taylor expansions. He proposed several schemes of second mean square order, i.e., strong order $\gamma = 1$, which generalize the Milstein scheme to jump–diffusion SDEs. The first of these for the scalar jump–diffusion SDE (5) is

$$Y_{n+1} \;=\; Y_n + \left(a - \frac{1}{2}\,b\frac{\partial b}{\partial x}\right)\Delta_n + b\,\Delta W_n + \frac{1}{2}\,b\frac{\partial b}{\partial x}\,(\Delta W_n)^2$$

$$+\frac{1}{2}\,(3c - c_c)\,\Delta N_n + (b_c - b)\,\Delta W_n\,\Delta N_n \qquad (10)$$
$$+\frac{1}{2}\,(c_c - c)\,(\Delta N_n)^2 + \left(b\frac{\partial c}{\partial x} - b_c + b\right)\,\Delta Z_n,$$

where all functions are evaluated at the point (t_n, Y_n). Here $f_c(t,x)$ for a function f (either b or c) is defined by

$$f_c(t,x) = f(t, x + c(t,x))$$

and the random variable ΔZ_n as the mixed multiple stochastic integral

$$\Delta Z_n = \int_{t_n}^{t_{n+1}} \int_{t_n}^{s} dW_t\, dN_s = \int_{t_n}^{t_{n+1}} (W_s - W_{t_n})\, dN_s. \qquad (11)$$

See [10] for the simulation of such integrals.

A MAPLE procedure for the above Milstein–Maghsoodi scheme (10) with constant time step size Δt for scalar jump-diffusion SDE (5) is given below. Again in the MAPLE code we write "d" instead of "Δ".

```
Milstein_jump:=proc(a::algebraic,b::algebraic,c::algebraic)
  local soln;
  soln:=Y[n+1]=Y[n]+(a-(1/2)*b*diff(b,x))*dt+b*dW[n]
    +(1/2)*b*diff(b,x)*(dW[n])^2+(1/2)*(3*c-subs(x=Y[n]+c,c))*dN[n]
    +(subs(x=Y[n]+c,b)-b)*dW[n]*dN[n]
    +(1/2)*(subs(x=Y[n]+c,c)-c)*(dN[n])^2
    +(b*diff(c,x)-subs(x=Y[n]+c,b)+b)*dZ[n];
  subs(x=Y[n],soln):
end:
```

Section 4 includes an example for using this scheme.

Simpler schemes, which are also of second mean square order, can be derived by incorporating the jump times into the partition. In this case the new partition of $[t_0, T]$ is given by $t_0 = \tau_0 < \tau_1 < \cdots < \tau_{N_T} = T$ such that $\max_{n=0,\ldots,N_T-1}(\tau_{n+1} - \tau_n) \le \Delta t$, w.p.1. Note that not all of the partition times τ_n need be random here, but could be specified to ensure that the upper bound on stepsize length holds, e.g. a deterministic step of stepsize Δt is used unless a jump occurs within that time. A jump adapted version of the Milstein–Maghsoodi (10) is given by

$$\begin{aligned} Y_{n+1} &= Y_n + \left(a - \frac{1}{2}\,b\frac{\partial b}{\partial x}\right)\,\Delta_{\tau_n} + b\,\Delta W_{\tau_n} + \frac{1}{2}\,b\frac{\partial b}{\partial x}\,(\Delta W_{\tau_n})^2 \\ &\quad + c\Delta N_{\tau_n} + (b_c - b)\,\Delta W_{\tau_n}\Delta N_{\tau_n}. \end{aligned} \qquad (12)$$

Here Y_n is the approximation to X_{τ_n} and

$$\Delta_{\tau_n} = \tau_{n+1} - \tau_n, \qquad \Delta W_{\tau_n} = W_{\tau_{n+1}} - W_{\tau_n}, \quad \text{etc.}$$

The MAPLE procedure for the jump adapted Milstein–Maghsoodi (12) for a scalar jump-diffusion SDE (5) is given in figure 4.

```
Adaptive_jump:=proc(a::algebraic,b::algebraic,c::algebraic)
  local soln;
  soln:=Y[n+1]=Y[n]+(a-(1/2)*b*diff(b,x))*dt[n]
    +b*dW[n]+(1/2)*b*diff(b,x)*(dW[n])^2+c*dN[n]
    +(subs(x=Y[n]+c,b)-b)*dW[n]*dN[n];
  subs(x=Y[n],soln)
end:
```

In Section 4 we illustrate the usage of this procedure and also show how a suitable sequence τ_n can be computed.

3.2 Vector jump–diffusion SDEs

An N–dimensional jump–diffusion Ito SDE with an M–dimensional Wiener process W_t and a scalar inhomogeneous Poisson counting process N_t has the componentwise form

$$dX_t^i = a^i(t, X_t)\, dt + \sum_{j=1}^{M} b^{i,j}(t, X_t)\, dW_t^j + c^i(t, X_{t^-})\, dN_t, \tag{13}$$

for $i = 1, \ldots, N$. Note that superscripts are used for the indices of vectors and matrices. In particular, $X_t = (X_t^1, \ldots, X_t^N)^\top$ and $W_t = (W_t^1, \ldots, W_t^M)^\top$, where the components W_t^j of W_t are scalar Wiener processes which are pairwise independent. Moreover, as in [5], the coefficient $b^{i,j}$ is the (i,j)th component of the $N \times M$-matrix $B = [b^1 | \cdots | b^M]$ with b^j as its jth column vector.

The counterpart of the Euler scheme (9) for the vector jump–diffusion SDE (13) reads

$$Y_{n+1}^k = Y_n^k + a^k(t_n, Y_n)\, \Delta_n + \sum_{j=1}^{m} b^{k,j}(t_n, Y_n)\, \Delta W_n^j + c^k(t_n, Y_n) \Delta N_n, \tag{14}$$

where $\Delta W^j = W^j_{t_{n+1}} - W^j_{t_n}$ is the $N(0;\Delta_n)$ distributed increment of the jth component W^j_t of the M-dimensionalW_t on subinterval $[t_n, t_{n+1}]$ and ΔN_n is as in the scalar case. Note in particular that $\Delta W_n^{j_1}$ and $\Delta W_n^{j_2}$ are independent for $j_1 \neq j_2$.

Below we give a MAPLE procedure for the above Euler scheme (14) with a constant time stepsize for the vector jump-diffusion SDE (13).

```
Euler_jump_vector:=proc(a::array,b::array,c::array)
  local i,u,soln,h;
  for i to rowdim(a) do
    soln[i]:=Y.i[n+1]=Y.i[n]+a[i,1]*dt
             +sum('b[i,j]*dW.j[n]','j'=1..coldim(b))+c[i,1]*dN[n];
    for u to rowdim(a) do
      soln[i]:=subs(x[u]=Y.u[n],soln[i]) od
  od;
  RETURN(eval(soln)):
end:
```

The input variables a, b, and c in procedure 'Euler_jump_vector' must be matrices of appropriate order, i.e., a and c are considered as $N \times 1$ matrices and the diffusion matrix b is an $N \times M$ matrix. Thus, the MAPLE package 'linalg' must be initially read into the worksheet. Also, any variables present in the elements of the matrices must be given in the form $x[1]$, $x[2]$,... ,$x[N]$, where N is the dimension of the system. An example for the application of this scheme can be found on the web page indicated in the abstract.

The Milstein–Maghsoodi scheme for the vector jump-diffusion SDE (13) reads

$$\begin{aligned} Y^k_{n+1} &= Y^k_n + \left(a^k - \frac{1}{2}\sum_{j=1}^m \nabla_x b^j\, b^j\right)\,\Delta_n + \sum_{j=1}^m b^{k,j}\,\Delta W^j_n \\ &\quad + \frac{1}{2}\sum_{j=1}^m\sum_{l=1}^m \nabla_x b^j\, b^l\,\Delta W^j_n\,\Delta W^l_n + \frac{1}{2}\left(3c^k - (c^k)_c\right)\,\Delta N_n \qquad (15) \\ &\quad + \sum_{j=1}^m \left((b^{k,j})_c - b^{k,j}\right)\,\Delta W^j_n\,\Delta N_n + \frac{1}{2}\left((c^k)_c - c^k\right)\,(\Delta N_n)^2 \\ &\quad + \sum_{j=1}^m \left(\nabla_x c^k\, b^{k,j} - (b^{k,j})_c + b^{k,j}\right)\,\Delta Z^j_n, \end{aligned}$$

where $\nabla_x b^k$ is the matrix with (i,j)-th component given by $\frac{\partial b^{i,k}}{\partial x_j}$ and ΔZ_n^j is the mixed multiple stochastic integral

$$\Delta Z_n^j = \int_{t_n}^{t_{n+1}} \int_{t_n}^{s} dW_t^j \, dN_s = \int_{t_n}^{t_{n+1}} \left(W_s^j - W_{t_n}^j\right) \, dN_s.$$

A MAPLE procedure for this scheme, along with an example for its application, can be found on the web page given in the abstract.

4 Numerical Simulations

In this section we illustrate the schemes for scalar equations presented in this paper. An illustration of the schemes for vector valued SDEs can be obtained from the web page indicated in the abstract. The reader can also obtain all of the MAPLE code described below via this page.

We are going to illustrate the schemes for two jump–diffusion SDEs, one of type (1), (2), the other of type (5), where we restrict ourselves to linear coefficients because in this case we are able to compare the numerical results to the exact solution (3).

In order to perform simulations we first need MAPLE routines for simulating the stochastic processes involved in the solutions. In order to generate the needed random variables it is convenient to use MAPLE's "stats" package, for a description see the chapter in this book by Ombach and Jarnicka [13], as well as [3]. Furthermore, for plotting the results we will need the "plots" and "plottools" packages. All these packages should be loaded into the worksheet at the beginning of the session. In addition, it is convenient to read the "randomize" function, which enables us to initialize MAPLE's random number generator and thus allows us to do repeated simulations for the same path and jump times. All these preliminary operations are done by the following commands.

```
> with(stats): with(plots): with(plottools):
> readlib(randomize)():
```

The following routine taken from [3] generates a discrete approximation of a Wiener process on $[0,T]$ with n steps.

```
W_path := proc(T,n)
```

```
  local w, h, t, alist:
  w := 0:
  t := 0:
  h := T/n:
  alist := [0,w]:
  from 1 to n do
  t := t + h:
  w := w + random[normald[0,sqrt(h)]](1):
  alist := alist,[t,w]:
  od:
  [alist]:
end:
```

Note that for schemes with constant step size Δt (like Euler or Milstein–Maghsoodi) it is sufficient to generate paths of the Wiener process for $n = T/\Delta t$. Since we want to compare these schemes to the jump adapted scheme we will simulate a path for a finer discretization, allowing for the evaluation of the path for different sequences of discretization times. The following routine evaluates a path W generated by `W_path` with parameters T and n at arbitrary time instances t using interpolation.

```
Wt := proc(W,t,T,n)
  local i, dt:
  i  := floor(n*t/T):
  if (i=n) then W[n+1,2]
  else
    dt := t*n/T -i:
    dt*W[i+2,2] + (1-dt)*W[i+1,2]:
  fi:
end:
```

Next we describe the procedures used for generating a sequence of jump times and—for jump SDEs of type (1), (2)—jump magnitudes.

The following routine returns a (possibly empty) sequence of two dimensional arrays. The first components contains the jump times $0 \leq \tau_1 \leq \ldots \leq \tau_l \leq T$, where $\tau_{i+1} - \tau_i$ is exponentially distributed with parameter `lambda`> 0. If `sigma`> 0, then the second components of the arrays in the list contain a sequence U_i of jump magnitudes which are lognormally distributed on $[-1, \infty)$, i.e., $G_i = \ln(1 + U_i)$ is Gaussian distributed with mean value `mu` and variance `sigma`^{2}. These values are needed for the simulation of

(1), (2). If `sigma`= 0 then the second components of the arrays are all set to 1.

```
jumps:=proc(lambda::algebraic, T::algebraic,
            mu::algebraic, sigma::algebraic)
  local i, j, tau, t, again, U, Ulist;
  again:=true;
  t[0]:=0;
  for i from 0 while again=true do
    tau:=stats[random, exponential[1]](1):
      if (t[i]+tau<=T) then
      t[i+1]:=t[i]+tau:
    else
      again:=false:
    fi:
  od:
  for j from 1 to i-1 do
    if sigma=0 then U:=1:
    else U:=exp(stats[random, normald[mu,sigma]](1)-1): fi:
    if (j=1) then Ulist:=[t[j],U]:
    else Ulist:=Ulist, [t[j],U]:   fi:
  od;
  if (i=1) then []:
  else [Ulist]: fi:
end:
```

Both for the explicit solution and for the evaluation of the numerical schemes we need procedures which compute the necessary information from the sequence of jumps generated by `jumps`. For the explicit solution we need to evaluate the sum $\sum_{n=1}^{N_t} G_n$ in (4). The following routine works simultaneously for equations of type (1), (2) and for equations of type (5) with $c(t, X) = \gamma X$. In the first case `gamm` has to be set to 1 and `U` has to be generated by `jumps` with `sigma`> 0, while in the second case one has to set `gamm`$= \gamma$ and `U` has to be generated by `jumps` with `sigma`$= 0$. In both cases, the parameter t is the time for which the sum in (4) is evaluated.

```
jumpsum:=proc(t::algebraic, U::list, gamm::algebraic)
  local i, j, nj, sum, again;
  sum:=0: j:=0:
  nj:=nops(U):
```

```
  if (nj>0) then
    again:=true:
    for i from 1 while (again) do
      if (t>=U[i,1]) then j:=i:
      else again:=false: fi:
      if (i=nj) then again:=false: fi:
    od:
  fi:
  for i from 1 to j do
    sum:=sum+ln(1+gamm*U[i,2]):
  od:
  sum:
end:
```

For the evaluation of the numerical schemes we need to compute the increments $\Delta N_n = N_{n\Delta t} - N_{(n-1)\Delta t}$ and $\Delta N_{\tau_n} = N_{\tau_n} - N_{\tau_{n-1}}$ (written as `dN[n]` and `dN[tau[n]]`, respectively, in our MAPLE notation) of the Poisson process N_t related to a sequence of jumps. This is done by the following routine, which computes $N_{t2} - N_{t1}$ from the sequence `U` generated by the routine `jumps`, above, for arbitrary times $t2 > t1 \geq 0$.

```
djump := proc(t1::algebraic, t2::algebraic, U::list)
  local i, sum:
  sum:=0:
  for i from 1 to nops(U) do
    if (U[i,1]>=t1) and (U[i,1]<t2) then
      sum:=sum + U[i,2]:
    fi:
  od:
  sum:
end:
```

Now we have all the necessary routines in order to start our simulations. The first set of examples is for the SDE of type (1), (2) with $\mu = 2$, $\sigma = 1$ and initial value $X_0 = 1$. The jumps are constructed according to the description in the Introduction with $\lambda = 1$, $\mu_G = 0$ and $\sigma_G = 2$. We compute the solution on the interval $[0, 1]$ with $n = 100$ discretization steps, i.e., $\Delta t = 1/100$.

We first generate a discrete path of a Wiener process on $[0, 1]$ with $n = 1000$ steps by

```
> rand1:=50: T:=1: W_steps:=1000: randomize(rand1):
  W:=W_path(T,W_steps):
```

and then specify the desired variables as

```
> X0:=1:   steps:=100: mu:=2:  sigma:=1:
  gamm:=1: lambda:=1:  muG:=0: sigmaG:=2:
```

Now we generate a jump sequence by

```
> rand2:=1: randomize(rand2): U:=jumps(lambda,T,muG,sigmaG):
  lines:=[seq(line([U[i,1],0], [U[i,1],gamm*U[i,2]],
                  color=black, linestyle=1), i=1..nops(U))]:
```

The "randomize" commands here initialize MAPLE's random number generator. With this construction each call of the above commands produces the same path W and the same jump sequence U for the same values of `rand1` and `rand2`, respectively. The MAPLE list `lines` contains graphical information for plotting the jumps, which is used below.

The following MAPLE code now computes the exact solution, which is plotted to the variable `a` using a black solid line, the Euler solution (plotted to `b` gray solid), the Milstein–Maghsoodi solution (plotted to `c` black dashed) and the jump adapted solution (plotted to `d` gray dashed), which are finally plotted onto the screen into one diagram by the `display` command.

For the jump adapted scheme we have to construct a suitable jump adapted sequence of time steps τ_n, which here is accomplished by adding the jump times from `U` to an equidistributed sequence of time–steps with step size Δt.

Due to the fact that the equation is linear, the coefficients in front of the term ΔZ_n in the Milstein–Maghsoodi scheme vanish and thus we do not need to simulate this term. A MAPLE routine `zjump` for its evaluation, which is analogous to `djump`, above, using a formula from [10] is contained in the worksheets related to this paper.

```
> X:=[0,X0]:
  for n from 1 to steps do
    t:=n*T/steps:
    X:=X, [t, X0*exp((mu-sigma^2/2)*t
              +sigma*Wt(W,t,T,W_steps)+jumpsum(t,U,gamm))]:
  od:
  a:=plot([X],color=black,linestyle=1):
```

```
> i:='i':
  n:='n':
  X:=[0,X0]:
  X1:=X0:
  scheme:=rhs(Euler_jump(mu*x,sigma*x,gamm*x)):
  h:=T/steps:
  for i from 1 to steps do
    t:=i*T/steps:
    jump:=djump(t-h,t,U):
    wiener:=Wt(W,t,T,W_steps)-Wt(W,t-h,T,W_steps):
    X1:=subs(Y[n]=X1,dt=h,dW[n]=wiener,dN[n]=jump,scheme):
    X:=X,[t,X1]:
  od:
  b:=plot([X],color=gray,linestyle=1):

> i:='i':
  n:='n':
  X:=[0,X0]:
  X1:=X0:
  scheme:=rhs(Milstein_jump(mu*x,sigma*x,gamm*x)):
  h:=T/steps:
  for i from 1 to steps do
    t:=i*T/steps:
    jump:=djump(t-h,t,U):
    wiener:=Wt(W,t,T,W_steps)-Wt(W,t-h,T,W_steps):
    X1:=subs(Y[n]=X1,dt=h,dW[n]=wiener,dN[n]=jump,scheme):
    X:=X,[t,X1]:
  od:
  c:=plot([X],color=black,linestyle=2):

> i:='i':
  n:='n':
  X:=[0,X0]:
  X1:=X0:
  scheme:=rhs(Adaptive_jump(mu*x,sigma*x,gamm*x)):
  h:=T/steps:
  adaptsteps:=steps+nops(U):
  j:=1: k:=1:
  tau[0]:=0:
```

```
  for i from 1 to adaptsteps do
    if (j<=nops(U)) and (U[j,1]<k*T/steps) then
      tau[i]:=U[j,1]: j:=j+1:
    else
      tau[i]:=k*T/steps: k:=k+1:
    fi:
  od:
  for i from 1 to adaptsteps do
    t:=tau[i];
    h:=tau[i]-tau[i-1];
    jump:=djump(tau[i-1],tau[i],U):
    wiener:=Wt(W,tau[i],T,W_steps)-Wt(W,tau[i-1],T,W_steps):
    X1:=subs(Y[n]=X1,dt[n]=h,dW[n]=wiener,dN[n]=jump,scheme):
    X:=X,[t,X1]:
  od:
  d:=plot([X],color=gray,linestyle=2):

> display(a,b,c,d,lines);
```

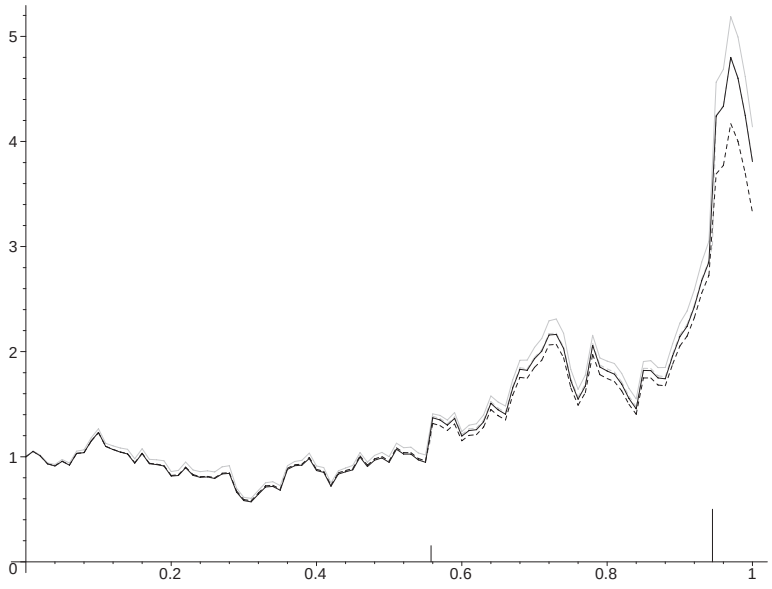

Figure 1: MAPLE output for SDE (1), (2) (solid = exact, gray solid = Euler, dashed = Milstein–Maghsoodi, gray dashed = adapted)

Figure 1 shows the graphical output from these routines. In addition, the jumps are indicated by vertical lines. The solution for the jump adapted scheme is not visible because it almost coincides with the exact solution. We would like to emphasize again that for these type of jump–diffusion SDEs with a random jump coefficient $c(t, X)$ a rigorous convergence analysis for all of the numerical schemes under consideration does not exist. Nevertheless, experiments with different time steps revealed that the Euler and the jump adapted scheme show very promising results while the Milstein–Maghsoodi scheme performs worse than expected.

In our second example, we apply the same routines as above to the SDE (5) with $a(t, X) = \mu X$, $b(t, X) = \sigma X$ and $c(t, X) = \gamma X$ with $\mu = 2$, $\sigma = 1$ and $\gamma = 5$. The initial value $X_0 = 1$, the time interval $[0, T]$ and the number of discretization steps $n = 100$ were chosen as in our first example, above.

In order to simulate this equation we can use the same code as above when we change the parameters as follows.

```
> X0:=1:   steps:=100: mu:=2:  sigma:=1:
  gamm:=5: lambda:=1:  muG:=0: sigmaG:=0:
```

With these parameters, the above code produces the graphical output shown in Figure 2.

Here the Milstein–Maghsoodi scheme (black dashed line) produces a much better result than for equation (1), (2). Again, however, the adapted scheme gives the best solution, which in the diagram is almost indistinguishable from the exact solution. We should, however, note that Maghsoodi [8] reports that jump adapted schemes can be computationally inefficient for nonlinear equations.

References

[1] S. Cyganowski, L. Grüne and P.E. Kloeden, MAPLE for Stochastic Differential Equations, in *Theory and Numerics of Differential Equations*, J.F. Blowey, J.P. Coleman, A.W. Craig, eds., Springer Verlag (2001), 127-178.

[2] S.Cyganowski and P.E. Kloeden, MAPLE schemes for jump-diffusion stochastic differential equations, in in *Proc. 16th IMACS World Congress,* Lausanne 2000, M. Deville, R. Owens, eds., Dept. of Computer Science, Rutgers University, (2000)., 2000, paper 216-9 on CD.

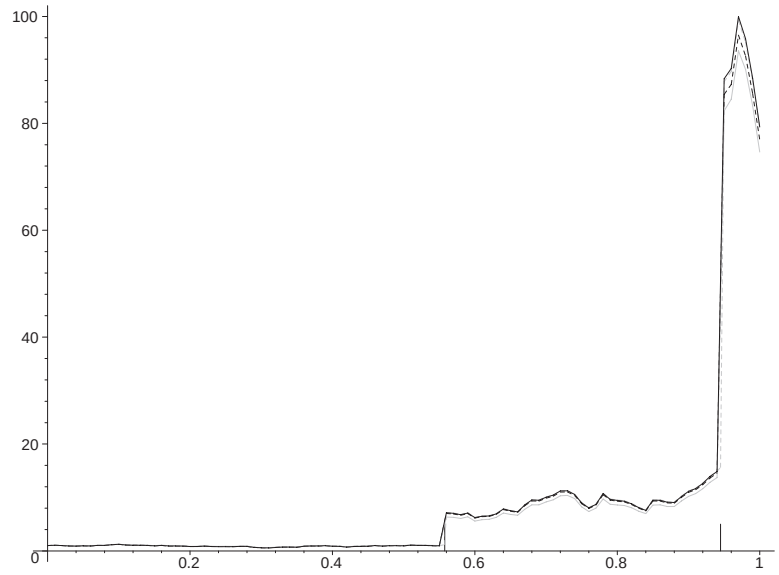

Figure 2: MAPLE output for SDE (5) (solid = exact, gray solid = Euler, dashed = Milstein–Maghsoodi, gray dashed = adapted)

[3] S. Cyganowski, P.E. Kloeden and J. Ombach, *From Elementary Probability to Stochastic Differential Equations with* MAPLE, Springer–Verlag, Heidelberg, 2002.

[4] I.I. Gikhman and A.V. Skorokhod, *Stochastic Differential Equations*, Springer-Verlag, Berlin, 1972.

[5] D.J. Higham and P.E. Kloeden, MAPLE and MATLAB for Stochastic Differential Equations in Finance, in *Programming Languages and Systems in Computational Economics and Finance,* S.S. Nielsen, ed., Kluwer Academic Publishers, Amsterdam (2002).

[6] P.E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations*, Springer–Verlag, Heidelberg, 1992; second revised printing 1999.

[7] D. Lamberton and B, Lapeyre, *Stochastic Calculus Applied to Finance*, Chapman & Hall, London, 1996.

[8] Y. Maghsoodi, Mean square efficient numerical solution of jump–diffusion stochastic differential equations, *Indian J. Statistics*, **58** (1996), pp. 25-47.

[9] Y. Maghsoodi, Exact solutions and doubly efficient approximation and simulation of jump–diffusion Ito equations, *Stoch. Anal. Applns.,* **16** (1998), pp. 1049–1072.

[10] Y. Maghsoodi and C.J. Harris, In-probability approximation and simulation of non-linear jump-diffusion stochastic differential equations, *IMA Journal of Mathematical Control and Information,* **4** (1987), pp. 65-92.

[11] R.C. Merton, Option pricing when underlying stock return rates are discontinuous, *J. Financial Econ.*, **3** (1976), pp. 141–183.

[12] R. Mikulevicius and E. Platen, Time discrete Taylor approximations for Ito processes with jump component, *Math. Nachr.*, **138** (1988), pp. 93–104.

[13] J. Ombach and J. Jarnicka, Statistics and Simulations with MAPLE, in *Programming Languages and Systems in Computational Economics and Finance,* S.S. Nielsen, ed., Kluwer Academic Publishers, Amsterdam (2002).