APPLICATIONS OF MATHEMATICA TO THE STOCHASTIC CALCULUS

J. Michael Steele[1] & Robert A. Stine[2]
Robert A. Stine, Dept. of Statistics, The Wharton School, University of Pa, Phila, PA 19104-6302

Key Words: Black-Scholes, diffusion, symbolic computing.

Introduction

Symbolic computing has established its value in dealing with many of the more tedious aspects of differentiation and integration. Our goal here is to show that symbolic computing is also a powerful tool for the study of the stochastic calculus.

As an illustration of the power of this approach, we consider the classic option pricing problem of mathematical finance and use symbolic computations to obtain the Black-Scholes formula. Our objective is to derive this formula for pricing an option rather than manipulate the well-known final result.

After a brief review of diffusions, we describe a symbolic representation for a diffusion and illustrate how this representation is sufficient for simulating and plotting realizations of diffusions. We then consider transformations of diffusions via the Itô formula. The Itô formula is the basis of our symbolic algorithm that determines the diffusion defined by a transformation of another.

While this paper uses Mathematica, these same ideas could also be implemented in other symbolic environments, such as Macsyma. Our choice of Mathematica was based on its graphics, programmability, and large user base. If one has access to this system, the commands shown here can be replicated after installing the definitions of the functions.

A catalog of the details about the use of Mathematica appear in Wolfram (1991); the more gentle tutorial of Gray and Glynn (1991) provides a good introduction. Other applications of symbolic computing in statistics appear in Steele (1985) and Stine (1990).

Diffusions

A diffusion X_t is identified by two functions μ and σ and its initial value X_0. Assume that the functions $\mu, \sigma : (R \times R^+) \to R$ are members of L^2 and let W_t denote the standard Weiner process. A fundamental result of Itô's theory of stochastic integrals is that there exists a well-defined process X_t that can be written suggestively as

$$X_t = X_0 + \int_0^t \mu(X_s,s)\, dt + \int_0^t \sigma(X_s,s)\, dW_s \,. \qquad (1)$$

The vital intuition behind X_t and this representation is that the function μ determines the drift of X_t whereas σ controls its variability. For example, if X_t represents the price of a stock, then μ and σ could be interpreted as the associated rate of return and risk. Generally (1) is expressed in differential form as

$$dX_t = \mu(X_t,t)\, dt + \sigma(X_t,t)\, dW_t. \qquad (2)$$

Note that the process W_t is not of bounded variation so that the integral in (1) can not be interpreted naively. The interested reader can consult, for example, the book of Arnold (1974) or Duffie (1988) for further motivation and details.

Itô's Formula

Itô's formula is the key ingredient that underlies many of our symbolic manipulations of diffusions. Assume for the function f mapping $(R,R^+) \to R$ that the derivatives

$$f_x = \partial f(x,t)/\partial x, \quad f_{xx} = \partial^2 f(x,t)/\partial x^2,$$

and

$$f_t = \partial f(x,t)/\partial t$$

exist. If $Y_t = f(X_t,t)$, then Itô's Formula shows that Y_t is also a diffusion,

$$dY_t = \left(f_t(X_t,t) + \mu(X_t,t)\, f_x(X_t,t) + f_{xx}\sigma^2(X_t,t)/2\right) dt + \left(f_x(X_t,t)\, \sigma(X_t)\right) dW_t \qquad (3)$$

This expression is not, however, in the canonical form (2) since the drift and dispersion are functions of X_t rather than Y_t. If we

assume that f has an inverse g so that

$$y = f(x) \iff x = g(y),$$

we obtain the desired form by substituting $g(Y_t)$ for X_t in (3), obtaining

$$\begin{aligned} dY_t &= \{f_t(g(Y_t),t)+\mu(g(Y_t),t)f_x(X_t,t) + \\ &\qquad f_{xx}\,\sigma^2(g(Y_t),t)/2\}\,dt \\ &\quad + \{\,f_x(g(Y_t),t)\;\sigma(X_t,t)\,\}\,dW_t \\ &= \bar{\mu}(Y_t,t)\,dt + \bar{\sigma}(Y_t,t)\;dW_t\,. \end{aligned} \tag{4}$$

A key task for Mathematica is to manage the mapping $\{\,\mu(x,t),\ \sigma(x,t)\,\} \to \{\,\bar{\mu}(y,t),\ \bar{\sigma}(y,t)\,\}$.

The Infinitesimal Generator

The infinitesimal generator simplifies the evaluation of the Itô formula. Given a diffusion X_t, the infinitesimal generator A measures the infinitesimal rate of change in the expected value of $f(X_t,t)$ given that X_t starts from x,

$$Af(x) = \lim \frac{E_x\, f(X_t,t) - f(x,0)}{t}$$

as $t \to 0$, where E_x denotes the expectation conditional on $X_0 = x$. Viewed as an operator, the infinitesimal associated with X_t is the sum

$$A = \frac{\partial}{\partial t} + \mu(x,t)\frac{\partial}{\partial x} + {}^1\!/_2\,\sigma(x,t)\frac{\partial^2}{\partial x^2}\,. \tag{5}$$

This expression is useful since the drift function in (3) is the result of applying the infinitesimal associated with X_t to the function f. Substituting (5) into (3), we obtain a much more manageable expression for Itô's formula (3),

$$\begin{aligned} dY_t &= df(X_t) \\ &= A[f]\,dt + f_x(X_t,t)\;\sigma(X_t,t)\;dW_t\,. \end{aligned}$$

Black-Scholes Option Pricing Model

All of these concepts are captured in the following simple variation of the options pricing problem. The problem is to determine the current market value of an option to purchase shares of a stock at some future time T for a set price P. Such options are known as European options. Let $V(S_t, t)$ denote the value of the option at time $t \le T$ where S_t denotes the price of the stock at time t.

The key feature of the Black-Scholes formulation of this problem is to introduce two diffusions. Assume that the market consists of two investments, stocks and bonds. The prices of shares of stock and bond S_t and B_t are treated as diffusions of known form,

$$\begin{aligned} dS_t &= \mu S_t\;dt + \sigma S_t\;dW_t \\ dB_t &= r\,B_t\,dt\,. \end{aligned}$$

The bond process represents a risk-free asset with guaranteed rate of return r, whereas the stock diffusion is a more risky asset offering higher potential returns with some chance of less. Here we ignore dividends and transaction costs and assume that the market does not permit arbitrage. This brief introduction is from Duffie (1988, §22) which contains further motivation and details.

The Black-Scholes approach duplicates the value of the option $V(S_t,t)$ with a portfolio consisting of a_t shares of stock and b_t shares of bond. We need to determine a_t and b_t. The portfolio representation for the value of the option implies that $V(S_t,t)$ is itself a diffusion, having the form

$$V(S_t,t) = a_t\,S_t + b_t\,B_t\,.$$

It follows that

$$\begin{aligned} dV(S_t,\,t) &= a_t\,dS_t + b_t\,dB_t \\ &= (a_t\,\mu\,S_t + b_t\,r\,B_t)\,dt + a_t\,\sigma S_t\,dW_t\,, \end{aligned} \tag{6}$$

so that the drift of $V(S_t,t)$ is $a_t\,\mu\,S_t + b_t\,r\,B_t$ and the dispersion is $a_t\,\sigma S_t$. On the other hand, the Itô formula also gives expressions for the mean and dispersion of $V(S_t,t)$,

$$dV(S_t\,,\,t) = A[V]\,dt + V_x(S_t,t)\;\sigma S_t\,dW_t\,, \tag{7}$$

where V_x denotes the derivative of V with respect to its first argument. Equating the drift and dispersion functions of (6) and (7) yields a system of equations which determine the coefficients a_t and b_t of the matching portfolio,

$$\begin{aligned} a_t &= V_x(S_t,t) \\ b_t &= \frac{Vxx(S_t,t)\sigma^2 S_t^2 + V_t(S_t,t)}{rB_t}\,, \end{aligned}$$

where V_{xx} denotes the second derivative of V with respect to its first argument, and V_t is the first derivative with respect to its second argument. Substituting these solutions into the portfolio representation (6) gives a partial differential equation for V,

$$\begin{aligned} &-rV(x,t) + V_t(x,t) + rxV_x(x,t) \\ &\qquad + \sigma^2 x^2 V_{xx}(x,t)/2 = 0. \end{aligned}$$

This PDE can be solved by various methods, including the Feynman-Kac theorem.

If A is the infinitesimal of diffusion X_t, the Feynman-Kac theorem states that the solution of the partial differential equation

$$A[V(x,t)] - r\,V(x,t) = 0 \quad (0 \le t \le T) \qquad (8)$$

with boundary condition $V(x,T) = g(x,T)$ is

$$V(x,t) = e^{-\rho(T-t)}\, E[g(X_T,T) \mid X_t = x]. \qquad (9)$$

For the Black-Scholes problem, the payoff function for the option determines the boundary condition. For example, the payoff function for a European option is the segmented function

$$g(x) = (x - P)^+ ,$$

where P is the exercise price of the option. Although (9) looks rather impressive, we later show how it simplifies in the case to a log-normal integration which is solved by standard methods.

A Data Structure for Diffusions

In order to use Mathematica when working on problems like deriving the Black-Scholes formula, we first need the ability to create and manipulate diffusions. Our choice here is simple, yet contains all of the needed information. Basically, the data structure is a list with four elements: the symbol which identifies the diffusion, the functions μ and σ, and finally the initial value. The functions μ and σ are entered as expressions of the identifying symbol and t. Rather than just put these items together in a list, however, we make the *head* of the expression the identifier *diffusion*. This subtle change permits some abstraction and provides the opportunity for type-checking in functions that operate on diffusions. In order to insure proper initialization, each diffusion is created by a call to the creator function *MakeDiffusion*.

To demonstrate the use of Mathematica, the following dialog shows boxed snapshots from a Mathematica session. Mathematica expressions are shown in Courier type and are numbered in italics. The dialog contains definitions for many of the functions that we use. Others, including those for making and printing diffusions, are imported from an external file. If one has access to Mathematica, these expressions can be entered directly into the program. Note that terminating a statement with a semi-colon suppresses printing the results of that statement.

We begin the session by importing the external definitions from the file Ito.m, and then create two diffusion objects.

```
In[1]:=
   <<Ito.m

In[2]:=
   bm = MakeDiffusion[W,0,1,0];
   exp= MakeDiffusion[X,r X,s X,v]

Out[2]:=
   diffusion[X,r X,s X, v]
```

The items bm and exp are Mathematica objects representing Brownian motion

$$dW_t = 0\,dt + 1\,dW_t$$

and the diffusion

$$X_t = r\,X_t\,dt + s\,X_t\,dW_t\,;\; X_0 = v. \qquad (10)$$

Notice that the name exp identifies the diffusion object, whereas the argument X in the second input is a symbol identifying the diffusion in the expressions for the drift and dispersion functions μ and σ.

While it is tempting to use the symbol X as the name of the diffusion object at line 2, we cannot do so here. Such statements lead to infinite recursion.

```
In[3]:=
   X=MakeDiffusion[X,r X,s X,v]

   General::recursion:
   Recursion depth of 256 exceeded.

Out[3]:=
   $Interrupted
```

The system repeatedly substitutes the entire expression for each X appearing in the right-hand side of In[3].

Automatic evaluation of arguments of functions can cause other problems. For example, if X had previously been assigned the value 2, as by the expression X = 2, then Mathematica would interpret the second line

of In[2] as

```
exp = MakeDiffusion[2,r 2,s 2,v],
```

and confusion would rein. To insure against this problem, the definition of the function `MakeDiffusion` begins

```
MakeDiffusion[s_Symbol,
              mu_, disp_, init_].
```

The leading argument `s_Symbol` provides type checking: the function is only called when the argument `s` is a symbol, and not, for example, a number.

To enhance this object-oriented approach we use a set of "methods" for manipulating diffusions. Although Mathematica does not provide a full set of object-oriented programming facilities, we can extend the abstraction by providing accessor methods. For example, the functions

```
symbol[d_diffusion]        :=d[[1]];
drift[d_diffusion]         :=d[[2]];
dispersion[d_diffusion]    :=d[[3]];
initialValue[d_diffusion]:=d[[4]];
```

extract the components of a diffusion by indexing into the data structure. As long as all other routines access the drift function μ via the function `drift` (rather than indexing directly), we can change the internal representation of a diffusion by changing these accessor functions rather than finding every place that we extract the drift component.

To illustrate the use of the diffusion data structure, consider how to display a diffusion. While the expression

```
diffusion[X,r X,s X,v]
```

in Out[2] is indicative of the mathematical notation in (10), it leaves a lot to be desired. Mathematica offers, however, some limited formatting capabilities that permit us to define the function `PrintDiffusion` which yields a more familiar rendering,

```
In[4]:=
   PrintDiffusion[exp]

Out[4]:=
   dX =(r X ) dt + (s X )dW ;X = v
     t      t             t    t  0
```

The data structure also contains sufficient information for producing simulated realizations. A discrete approximation to X_t suggests that we can "walk" the diffusion out from its starting point by incrementing time by small amounts and applying the functions μ and σ recursively. To do this easily in Mathematica, break the problem into two parts. First, build a function that generates the next value in the discrete approximation given the current value:

```
In[5]:=
   nextValue[{t_,current_}] := Block
   [  x1 = mu[current, t] dt;
      x2 = sig[current, t]nRand[dt];
      {t+dt,current+x1+x2} // N  ];

In[6]:=
   data=NestList[nextValue,{0,x0},100];
```

Here `nRand[x]` produces a zero-mean Gaussian random variable with variance `x`. Like `MakeDiffusion` and `PrintDiffusion`, `nRand` is defined in the external file Ito.m. The simulated realization is generated by applying `nextValue` recursively via the built-in function `NestList`. `NestList` returns the recursively defined list

```
{nextValue[{0,x0}],
 nextValue[nextValue[{0,x0}]],
   ...}.
```

The function `Realize` manages the associated variable and function initializations:

```
In[7]:=
Realize[d_diffusion, n_, dt_] :=
Block[{x,mu,sigma,sym=symbol[d],x0},
 mu[x_,t_] =(drift[d]      /.sym->x);
 sig[x_,t_]=(dispersion[d]/.sym->x);
 x0 = N[initialValue[d]];
 NestList[nextValue,{0,x0},n]  ];
```

`Realize` creates the functions `mu` and `sig` used in `nextValue`, with the symbol of the diffusion replaced by the generic identifier `x`. Had we instead attempted to define `mu` with the line

```
mu[sym_,t_] = drift[d];
```

Mathematica would look for the symbol `sym` in `drift`, rather than look for the symbol which is the value of `sym`.

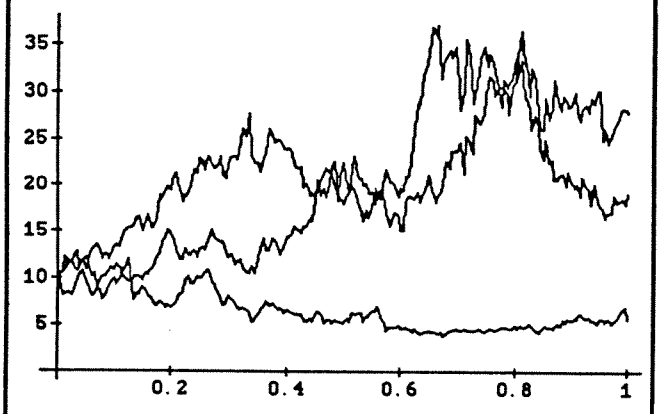
As an example, the following steps generate three realizations of the diffusion

$$dN_t = N_t\,dt + N_t\,dW_t \quad (N_0 = 10).$$

The mapping function (identified by /@) avoids looping, and Show plots the three realizations together on the same axes.

```
In[8]:=
   dif = MakeDiffusion[n,n,n,10];
   simDif  = Realize[#,200,.005]&
      /@ Table[dif,{3}];
   plts=ListPlot[#,PlotJoined->True]&
      /@ simDif;
   Show[plts]

Out[8]:=
```



Finally, we need to describe how to add, subtract, and multiply diffusions. These instructions are given as rules associated with the diffusion type. For example, the following rule describes how to add two diffusions:

```
diffusion /:
  diffusion[s1_,f1_,g1_,i1_] +
  diffusion[s2_,f2_,g2_,i2_] =
     diffusion[SUM, f1+f2, g1+g2,
                              i1+i2];
```

The leading "/:" associates this rule for addition with diffusion objects rather than the protected internal function "+". It is implicit in this rule that both diffusions are defined in terms of the same Weiner process. When the drift and dispersion functions do not depend on the diffusion symbols, as with Brownian motion, this simple rule works fine. Notice, however, that the symbolic name for the new diffusion, SUM, does not appear in the resulting drift and dispersion functions. Thus, this representation does not conform to the representation (2) since μ and σ are not functions of the leading symbol SUM. In many cases it is not possible to obtain this form. For example, consider

```
In[9]:=
 xDiff=MakeDiffusion[X,Sqrt[X],2,0];
 yDiff=MakeDiffusion[Y,Sqrt[Y],3,0];
 zDiff = xDiff + yDiff;

Out[9]:=
 diffusion[SUM,Sqrt[X]+Sqrt[Y],5,0]
```

In this case, it is not possible to express the drift $X^{1/2}+Y^{1/2}$ as a function of $Z = X+Y$ alone. Fortunately, in many cases such as the Black-Scholes illustration below, we can leave μ and σ in the form produced by this simple rule. Our expanded system notes that it cannot make this substitution and marks the diffusion object in such a way that other routines, such as that for finding the infinitesimal, are aware of the problem.

Implementating the Itô Formula

We begin with the infinitesimal generator. In keeping with the form of (5), we can implement the infinitesimal as a function which consists of derivative operators. The second two derivatives of (5) are captured by the function derivOp, which is the key element of evaluating the infinitesimal function. The #/& pairs in derivOp denote abbreviated function definitions, similar to the lambda functions of Lisp.

```
In[10]:=
  derivOp[var_Symbol]:=
   {D[#,var]&, D[#,var,var]&};

In[11]:=
A[d_diffusion][f_] :=
 Block[
  {dim,dft,disp,fd,sd,ggp,dt,Ops,
         driftOp, dispOp},
  dft=drift[d];disp=dispersion[d];
  ops = derivOp[ symbol[d] ];
  fd=ops[[1]][f]; sd=ops[[2]][f];
  dft = dft fd;
  disp = 1/2 disp disp sd;
  dt = D[f,t];
  Simplify[drift + disp + dt]
 ];
```

The function A uses an operator notation, with A[X] denoting the generator A for diffusion X_t. As an example, consider applying this operator to an arbitrary function $g(W_t)$ of Brownian motion W_t. Since the symbol associated with the Brownian motion bm is W, we denote this function g(W). The following output shows that $A[g] = g''/2$. Within this Mathematica session, g has not been defined and so is treated abstractly. To obtain a more explicit result, we use "%" to identify the result at Out[12] and substitute Log for g, giving us the infinitesimal for $Log[W_t]$.

```
In[12]:=
   A[bm][g[W]]

Out[12]:=
   g''[W]
   ------
     2

In[13]:=
   % /. g->Log

Out[13]:=
    -1
   ----
     2
   2 W
```

We have also experimented with vectors whose elements are diffusions. For example, vBM is a diffusion in R^d whose elements W[1], W[2],...,W[d] are independent scalar Brownian motions. With d=2, we find that the infinitesimal is the Laplacian

$$\frac{1}{2}\left\{\frac{\partial^2 g}{\partial x_1}+\frac{\partial^2 g}{\partial x_2}\right\}.$$

```
In[14]:=
   dim = 2;
   vBM = MakeDiffusion[Array[W,dim],
   Table[0,{i,dim}],
   IdentityMatrix[dim]];

In[15]:=
   A[vBM][g[W[1],W[2]]]
Out[15]:=
     (0,2)              (2,0)
   g    [W[1],W[2]]+g    [W[1],W[2]]
   ------------------------------
                2
```

As the number of dimensions increase, it becomes more and more easy to overwhelm the system with expressions that it is unable to simplify quickly.

With the infinitesimal in hand, the implementation of the Itô formula is straightforward, where we again use an operator notation.

```
In[16]:=
ito[d_diffusion][f_,opts___Rule]:=
Block[{drift,disp,oldSymbol,
   invRule, ns,invert},
   invert=itoInvert/.{opts}/.
            Options[ito];
   oldSymbol = symbol[d];
   drift = A[d][f];
   disp=D[f,oldSymbol].dispersion[d];
   Print["Prior to inversion..."];
   Print["d (",f,")(t) =
         (",drift,")dt (",disp,
         ") dW(t)"];
   If[Not[invert],
      ns = f,
      ns=itoSymbol/.{opts}/.
            Options[ito];
      invRules=First
        [Solve[ns==f,oldSymbol]];
      Print["Inversion rule... ",
      invRules];
   drift = Simplify[drift/.invRules];
   disp  = Simplify[disp/. invRules];
   MakeDiffusion[ns,drift,disp] ];
```

Using the Itô Formula

The intermediate printed output from this program is useful in solving stochastic integrals. For example, what is the value of

$$\int_0^t W_s\, dW_s\ ?$$

Since

$$\int_0^t x\, dx = t^2/2,$$

we might suspect that the stochastic integral is related to the function $x^2/2$. If we apply the function ito, we obtain

```
In[17]:=
ito[bm][1/2 W^2,itoSymbol->y]
Prior to inversion...
       2
      W          1
  d (--)(t) = (-) dt + (W) dW(t)
      2          2
Out[17]:=     1
  diffusion[y,-,Sqrt[2]Sqrt[y],0]
              2

In[18]:=
   ito[bm][Exp[W]]
Out[18]:=
                  Y
   diffusion[Y, -, Y, 1]
                  2
```

The intermediate result of In[17] implies that

$$\frac{W_t^2}{2} = \frac{t}{2} + \int_0^t W_s dW_s \ ,$$

so that the desired stochastic integral is

$$\int_0^t W_s dW_s = \frac{W_t^2 - t}{2} .$$

The option `itoSymbol->y` permits the user to specify what symbol to use for the new diffusion. The default symbol, as seen in In[18], is `Y`. An additional option `itoInvert` can be used to suppress the substitution of $g(Y_t)$ for X_t, leaving the new diffusion in the intermediate form (3). The second example shows that $exp(W_t)$ is the diffusion

$$dY_t = (Y_t/2)\, dt + Y_t\, dW_t ,$$

which is similar to the process simulated in the plot shown earlier.

Deriving the Black-Scholes Expression

Our implementation of diffusions permits us to derive the solution to the Black-Scholes problem symbolically by replicating the argument outlined in the introduction. The logic is essentially the same, only all of the calculations are performed symbolically by Mathematica.

Begin by defining the stock and bond diffusions, `stock` and `bond`. Then solve for the a_t and b_t portfolio sequences. As before, apply the Itô formula directly to the undefined function `v[S,t]`. Setting the option `itoInvert` to `False` prevents the system from performing the substitution $X_t = g(Y_t)$ leading from (3) to (4). As a result, the drift and dispersion of `vOfs` remain functions of the stock symbol `S` rather than introducing some new symbol. Next, at In[21], define the difference between the two representations of V given in (6) and (7).

```
In[19]:=
 stock= MakeDiffusion[S,mu S,sig S];
 bond = MakeDiffusion[B, r B, 0];

In[20]:=
 vOfs = ito[stock][v[S,t],
   itoInvert->False];

In[21]:=
 diff=vOfs - (at stock + bt bond);

In[22]:=
 roots = Solve[{drift[diff]==0,
   dispersion[diff]==0}, {at, bt}];
```

The resolution of In[21] requires the rules defined earlier for addition (and subtraction) of diffusions. Again, it is not necessary to resolve the drift and dispersion into functions of, say, the symbol `diff`. We want to retain the forms appearing in (6) as functions of S_t and B_t. Finally, to solve for the solution in terms of a_t and b_t, apply the built-in function `Solve` to the equations based on the drift and dispersion functions of the difference.

At this point we need to do some Mathematica house-keeping. The result of `Solve` is a list of rules identifying all solutions known to the system. For example, the roots of the equation $x^2 - 1 = 0$ shown with In[23] are returned as a list of lists of substitution rules. Thus, in order to use `roots`, we first use `roots` to define a value for a_t and then extract the result from the extraneous list. This result is achieved by the somewhat cryptic command at line 24. For b_t, we also simplify the resulting expression. The explicit simplification of the resulting expression deletes redundant terms and keeps the system from being overwhelmed later in the evaluation.

```
In[23]:=
   Solve[x^2 - 1 == 0]
Out[23]:=
   {{x -> 1}, {x -> -1}}

In[24]:=
   at = First[at/.roots]
Out[24]:=
     (1,0)
   v     [S, t]

In[25]:=
   bt=Simplify[First[bt/.roots]]
Out[25]:=
      (0,1)           2    2  (2,0)
  2 v     [S, t] + S sig  v     [S, t]
  ---------------------------------
                2 B r
```

With the expressions for a_t and b_t in hand, we can solve the resulting partial differential equation via the Feynman-Kac theorem. In order to use the Feynman-Kac theorem, we need to expand the infinitesimal $A[V(x,t)]$ in (8) and identify a diffusion X_t such that (8) holds. We first set up the PDE assuming that the payoff function is g, and then use pattern matching rules to apply the Feynman-Kac theorem.

```
In[26]:=
   pde = Expand[at S + bt B - v[S,t]];

In[27]:=
   soln = FeynmanKac[pde, g]
Out[27]:=
                                    g[Z,T-t]
Ave[diffusion[Z,r Z,z sig,x]][-------]
                                     r(T-t)
                                    E

In[28]:=
  xt=MakeDiffusion[X,r-(s^2)/2,s,start]
  ito[xt][Exp[X]]
Out[28]:=
  diffusion[Y, Y r, Y s,Exp[start] ]
```

In the result Out[27], `Ave[d_diffusion]` denotes the expected value with respect to the diffusion d. Translated into more standard notation, the solution at Out[27] is

$$V(x,t) = e^{-r\,(T-t)}\, E\,[\, g(Z_{T-t})\,] \qquad (11)$$

where $dZ_t = r\,Z_t\,dt + s\,Z_t\,dW_t$, $Z_0 = x$.

In order to compute the expectation in (11), notice that the calculation at line 28 implies that the diffusion Z_t in (11) is

$$Z_t = Exp[\,(r-\sigma^2/2)\,dt + \sigma\,dW_t\,].$$

That is, Z_t is the exponential of a Gaussian random variable with mean $(r-\sigma^2/2)$ and variance σ^2. That is, Z_t is lognormal. If we assume the payoff function $g(x) = (x-P)^+$, then a standard log-normal integration reduces (11) to

$$V(x,t) = x\,\Phi\{\frac{log(x/P) + (r+\sigma^2/2)(T-t)}{\sigma(T-t)^{1/2}}\} \qquad (12)$$
$$- P\,e^{-r(T-t)}\,\Phi\{\frac{log(x/P) + (r-\sigma^2/2)(T-t)}{\sigma(T-t)^{1/2}}\}$$

where $\Phi(x)$ is the standard cumulative normal distribution. The expression (12) for $V(x,t)$ is well-known and has been explored by Miller (1990) using Mathematica.

Directions

With this machinery in place, our plans include exploring the breadth of input diffusions that can be handled in this manner. For example, what other stock and bond diffusions are amenable to these operations? We also plan to add other methods which use diffusions to solve partial differential equations, such as the Girsanov transformation. At some point, we would like to integrate these techniques into the PDE solving capabilities of version 2 of Mathematica. We also need better methods for solving expectations with respect to diffusions as seen in (11). The solution from the Feynman-Kac theorem is convenient, but still requires considerable work and insight to reach the useful expression in (12).

We gave a single example of the use of vector diffusions. Vector models increase the complexity of the symbol management, and we intend to expand our functions in this direction.

References

Arnold, L.(1974). *Stochastic Differential Equations: Theory and Applications.* Wiley, New York.

Duffie, D. (1988). *Security Markets.* Academic Press, New York.

Gray, T.W. and J. Glynn (1991). *Exploring Mathematics with Mathematica.* Addison-Wesley, Redwood City, CA.

Miller, R. (1990). Computer-aided financial analysis: an implementation of the Black-Scholes model. *Mathematica Journal*, 1, 75-79.

Steele, M. (1985). MACSYMA as a tool for statisticians. *American Statistical Association Proceedings of the Statistical Computing Section*, 1-4. American Statistical Association, Washington.

Stine, R. A. (1990). Mathematica in time series analysis. *American Statistical Association Proceedings of the Statistical Computing Section.* American Statistical Association, Washington.

Wolfram, S. (1991). *Mathematica: A System for Doing Mathematics by Computer (2nd Edition).* Addison-Wesley, Redwood City, CA.

[1] Supported by NSF DMS-8812868, ARO DAALO3-89-G-0092, AFOSR-89-0301 and NSA-MDA-904-89-2034.

[2] Equipment and software provided by a grant from Merck, Sharp and Dohme.