

# Documentation for ANISE<sup>©</sup>

(Algorithm for Numerical Integration of Stochastic Equations)

## Contents

<b>1</b>	<b>How ANISE works</b>	<b>2</b>
<b>2</b>	<b>Installing ANISE</b>	<b>3</b>
<b>3</b>	<b>Optional Installation of the GNU gcc/g++/g77 compilers</b>	<b>4</b>
<b>4</b>	<b>Getting a random number generator</b>	<b>4</b>
<b>5</b>	<b>The Box-Muller algorithm</b>	<b>4</b>
<b>6</b>	<b>Generating colored noise with ANISE</b>	<b>5</b>

Copyright © 2004 by Yin Mei Wong. ANISE cannot be distributed without the written permission of Innovative Stochastic Algorithms.

# 1 How ANISE works

ANISE is a numerical integration algorithm for solving systems of Itô stochastic differential equations (SDES) **with strong solutions**

$$dX_t^j = a^j(\mathbf{X}_t, t) dt + \sum_{k=1}^m b_k^j(\mathbf{X}_t, t) dW_t^k$$

where  $j = 1, \dots, n$  and

$$\mathbf{X}_t = (X_t^1, \dots, X_t^n).$$

Here  $dW_t^k$  are normally distributed differentials with mean 0 and variance  $dt$ . These differentials must be supplied by the user. They can be generated from uniformly distributed random numbers using the Box-Muller algorithm. The functions  $a^j(\mathbf{X}_t, t)$  and  $b_k^j(\mathbf{X}_t, t)$  must be differentiable to high order (greater than 8) in all variables, because this is a necessary condition for the existence of strong solutions, but are otherwise arbitrary.

Generally the program works much like a Runge-Kutta algorithm for ordinary differential equations. The user calls ANISE repeatedly over a sequence of relatively small time steps of length  $dt$  in order to obtain a dynamics over some larger time scale. At each time step the user provides stochastic differentials  $dW_t^k$  which are sampled  $N(0, dt)$ . ANISE asks the user to provide derivatives

$$\begin{aligned} \frac{\partial X_t^j}{\partial W_t^k} &= b_k^j(\mathbf{X}_t, t) \\ \frac{\partial X_t^j}{\partial t} &= -\frac{1}{2} \sum_{i=1}^n \sum_{k=1}^m \frac{\partial b_k^j(\mathbf{X}_t, t)}{\partial X_t^i} b_k^i(\mathbf{X}_t, t) + a^j(\mathbf{X}_t, t) \end{aligned}$$

at a number of intermediate points. This information is used to deduce the functional dependence of the solutions on time  $t$  and the Wiener processes  $W_t^k$ . Based on this data the solutions are advanced from the current time  $t$  to  $t+dt$ . The accuracy of the solution is maintained by comparing an internal estimate of the error to a user specified tolerance at all intermediate steps.

The fortran call sequence for ANISE is

```
call anise(n, m, t, x, dt, dw, tol)
```

while the C/C++ call sequence (type void) is

anise( $n, m, t, x, dt, dw, tol$ );

where all parameters are chosen lower case.

fortran	C/C++	description
integer*4 $n$	int $n$ ;	number of equations (input)
integer*4 $m$	int $m$ ;	number of Wiener processes (input)
real*8 $t$	double $t[1]$ ;	current time (input, set to $t + dt$ on return)
real*8 $x(n)$	double $x[n]$ ;	initial conditions on input, solutions on output
real*8 $dt$	double $dt$ ;	desired time step (input)
real*8 $dw(m)$	double $dw[m]$ ;	stochastic differentials (input sampled $N(0,dt)$ )
real*8 $tol$	double $tol$	desired tolerance (input)

The fortran user must supply a subroutine `stod` of the form

subroutine `stod`( $n,m,t,v,x,y$ )

while the C/C++ user must supply a type void function

`stod`( $n,m,t,v,x,y$ )

fortran	C/C++	description
real*8 $v(m)$	double $v[m]$ ;	supplied by <code>anise</code>
real*8 $y(n)$	double $y[n]$ ;	returned by <code>stod</code>

The quantities  $y(n)$  are defined via

$$y(i) = \frac{\partial X_t^i}{\partial t} + \sum_{j=1}^m v(j) \frac{\partial X_t^i}{\partial W_t^j}$$

and are linear combinations of the derivatives discussed above.

## 2 Installing ANISE

ANISE is distributed as a pre-compiled subroutine and thus requires no installation. To make the routine available to multiple users in a linux environment create a library via the command

**ar r libanise.a anise.o**

and put the library in the `/usr/local/lib` directory. Users can then link to the library when compiling in the usual way (e.g.

`g77 -g myprogram.f -o myprogram -lanise` or `gcc -g myprogram.c -o myprogram -lanise`  
or `g++ -g myprogram.c++ -o myprogram -lanise` ).

### 3 Optional Installation of the GNU gcc/g++/g77 compilers

For users who do not at present have access to a compiler, the GNU compilers gcc/g++/g77 are freely available and easy to install (see <http://gcc.gnu.org>). Windows users must choose between installing minGW (minimalist GNU for Windows [www.mingw.org](http://www.mingw.org)) and cygwin (GNU+Cygnus+Windows [www.cygwin.com](http://www.cygwin.com)). Both are very simple to install. minGW includes the gcc/g77 compilers but a separate gcc/g77 package must be installed with cygwin.

### 4 Getting a random number generator

Generation of random numbers with good statistical properties is an essential prerequisite for successful use of ANISE. An excellent well tested algorithm for generating uniform random numbers, `ran2.f` or `ran2.c/ran2.c++`, is included with Numerical Recipes Software package ([www.nr.com](http://www.nr.com)). An algorithm `gasdev.f` or `gasdev.c/gasdev.c++` is also available for generating normally distributed random numbers. By default this calls a function called `ran1.f` or `ran1.c/ran1.c++`. We recommend that this be replaced with a call to `ran2`. Alternatively a number of generators of uniform and normally distributed random numbers are available for free download from the GAMS (Guide to Available Mathematical Software) website [gams.nist.gov](http://gams.nist.gov). The algorithm `drnor.f`, for example, has been tested and can be downloaded from the GAMS website at [gams.nist.gov/serve.cgi/Module/NMS/DRNOR/11308/](http://gams.nist.gov/serve.cgi/Module/NMS/DRNOR/11308/).

### 5 The Box-Muller algorithm

The Box-Muller algorithm is used to generate normally distributed stochastic numbers with zero mean and unit variance from a pair of uniformly distributed numbers on  $[0, 1]$ . Specifically, if  $x_1$  and  $x_2$  are independent and uniformly distributed then  $y = \sqrt{-2 \ln x_1} \cos 2\pi x_2$  is normally distributed with zero mean and unit variance.

## 6 Generating colored noise with ANISE

Colored noises with given statistical properties can be easily generated using the ANISE program. Suppose for example that a noise  $V_t$  with correlation function

$$M[V_t^*V_s] = \alpha(t-s)$$

is desired. In many cases it is possible to expand  $\alpha(t)$  as a sum of the form

$$\alpha(t) = \sum_j A_j e^{-i\omega_j t} e^{-\gamma_j t}.$$

This expansion can for example be generated using a least squares method (see 'other programs' at the ANISE home page) or the free Harminv program (<http://ab-initio.mit.edu/harminv/>). Provided that the coefficients  $A_j$  are positive then the noise  $V_t$  can be generated from Wiener processes  $W_t^j$  via

$$\begin{aligned} V_t &= \sum_j \mu_t^j \\ \mu_t^j &= \sum_j \sqrt{2\gamma_j A_j} \int_{-\infty}^t e^{i\omega_j(t-u)} e^{-\gamma_j(t-u)} dW_u^j. \end{aligned}$$

Now the colored noise  $V_t$  can be generated using ANISE by integrating the stochastic differential equations

$$d\mu_t^j = (i\omega_j - \gamma_j)\mu_t^j dt + \sqrt{2\gamma_j A_j} dW_t^j$$

for all  $j$ .