RAPPORT D’ÉTUDE 30/11/2006

N° 06DR114.doc

MCSim software validation

MCSim software validation - Validation du logiciel *MCSim*

DRC/DRCG – DRC/TOXI

Study participants - Liste des personnes ayant participé à l'étude : Frédéric Y. BOIS, Sandrine MICALLEF

PREAMBULE

Le présent rapport a été établi sur la base des informations fournies à l'INERIS, des données (scientifiques ou techniques) disponibles et objectives et de la réglementation en vigueur.

La responsabilité de l'INERIS ne pourra être engagée si les informations qui lui ont été communiquées sont incomplètes ou erronées.

Les avis, recommandations, préconisations ou équivalent qui seraient portés par l'INERIS dans le cadre des prestations qui lui sont confiées, peuvent aider à la prise de décision. Etant donné la mission qui incombe à l'INERIS de par son décret de création, l'INERIS n'intervient pas dans la prise de décision proprement dite. La responsabilité de l'INERIS ne peut donc se substituer à celle du décideur.

Le destinataire utilisera les résultats inclus dans le présent rapport intégralement ou sinon de manière objective. Son utilisation sous forme d'extraits ou de notes de synthèse sera faite sous la seule et entière responsabilité du destinataire. Il en est de même pour toute modification qui y serait apportée.

L'INERIS dégage toute responsabilité pour chaque utilisation du rapport en dehors de la destination de la prestation.

	Rédaction	**Vérification**	**Approbation**
NOM	F. BOIS	S. MICALLEF	P. HUBERT
Qualité	Délégué scientifique de la DRC	Ingénieur TOXI	Directeur de la DRC
Visa			

TABLE DES MATIÈRES

1. SUMMARY

Numerical simulations of the Lsodes integrator and pseudo-random generators of *MCSim* version 5.0.0 were performed to validate their outputs by comparison to a reference problem and reference software (*Matlab*). The Lsodes implementation gives the correct solution for a published stiff differential equation system.

Most of the pseudo-random generators of *MCSim* version 5.0.0 can be considered valid. However, the Half-Normal generator of *MCSim* version 5.0.0 should not be used and that generator should be fixed in the next version of the software.

2. GLOSSARY

ODE	Ordinary Differential Equation
QQplot	Quantile-quantile plot

3. INTRODUCTION

MCSim is a simulation package that allows to:

- Design statistical or simulation models (eventually dynamic, via ODEs),
- Perform Monte Carlo stochastic simulations,
- Bayesian inference through Markov Chain Monte Carlo simulations.

The aim of the present report is to validate some aspects of the software (integration routines and pseudo-random generators). These routines are called by most of the procedures performing the above tasks.

4. METHODS

4.1 INTEGRATION

The Lsodes integration routine provided by *MCSim* was tested using a published coupled oscillator problem (Skufca JD, 2004, Analysis still matters: a surprising instance of failure of Runge-Kutta-Felberg ODE solvers, *SIAM Review*, 46(4):729-737). The published paper gives an analysis of the behavior of the differential equation system studied, proves that it is stable, and examines the failure of some standard integration methods. The paper also notes the success of stiff implicit solvers, a class to which Lsodes, based on Gear's algorithm (Gear CW, 1971, Algorithm 407 - DIFSUB for solution of ordinary differential equations [D2]. *Communications of the ACM* 14:185-190) is based. The *MCSim* model code and input simulation file are given in Annex 1.

4.2 PSEUDO-RANDOM NUMBER GENERATION

Random simulation procedures of *MCSim* version 5.0.0 were validated on the basis of random generators available in *Matlab* version 7.0.0.19920 (R14).

The following distributions of *MCSim* were tested:

- Beta
- Binomial
- Chi2
- Exponential
- Gamma
- HalfNormal
- InvGamma
- LogNormal
- LogNormal_v
- LogUniform
- Normal
- Normal_v
- Piecewise
- Poisson

- TruncInvGamma
- TruncLogNormal
- TruncLogNormal_v
- TruncNormal
- TruncNormal_v
- Uniform

For each of the previous distributions, Monte Carlo simulations were run using *MCSim* to obtain a 10000-sample of random variables (Annex 2) These samples were compared to a 10000-sample obtained with Matlab (Annex 3). When a Matlab generator was available for the distribution studied (most of the time, in the statistical toolbox), we simply ran the proposed procedure. When the corresponding distribution was not implemented, procedures were coded in Matlab on the basis of other existing generators.

The parameters of these test distributions were arbitrarily chosen. Parameter values are given in Annex 2. To compare both samples (from *MCSim* and *Matlab*), QQplot were used.

5. RESULTS

5.1 INTEGRATION

Figure 1 presents the results of simulations of the differential system in the conditions specified by the reference article of Skufca (2004). The numerical solution provided by MCSim is stable and corresponds exactly to the published results.

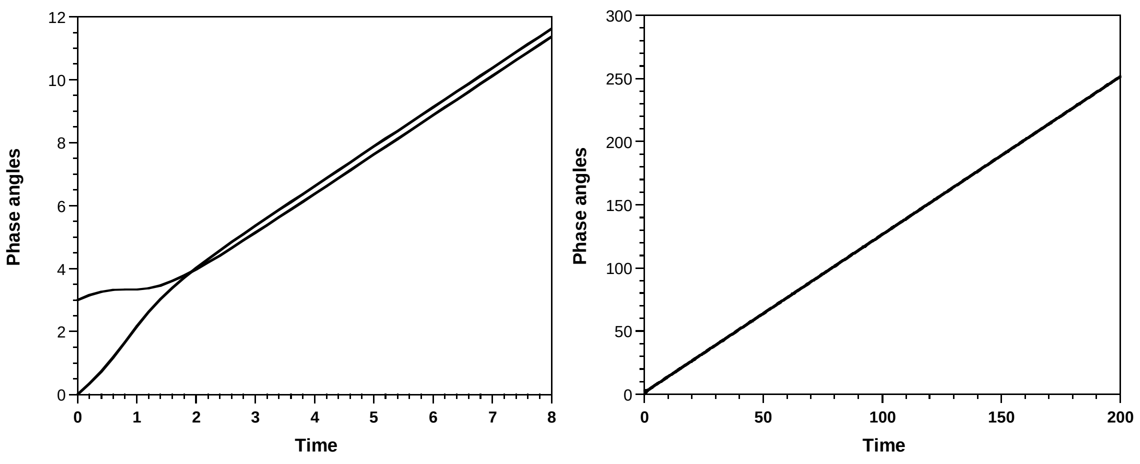

Figure 1: *MCSim*-computed trajectories of two coupled stiff differential equation oscillators. Only the time scale varies between the two plots. The trajectories are stable, as expected from a mathematical analysis of the system.

5.2 PSEUDO-RANDOM NUMBER GENERATION

QQplots from the simulated samples are displayed in Figure 2. In such plots, ordered values of both samples are plotted on each axis. If the two samples come from the same distribution, the plot is linear. Discrete distributions appear jagged. Superimposed on the plot is a line joining the first and third quartiles of each distribution (this is a robust linear fit of the order statistics of the two samples). This line is extrapolated to the ends of the sample to help evaluate the linearity of the plot.

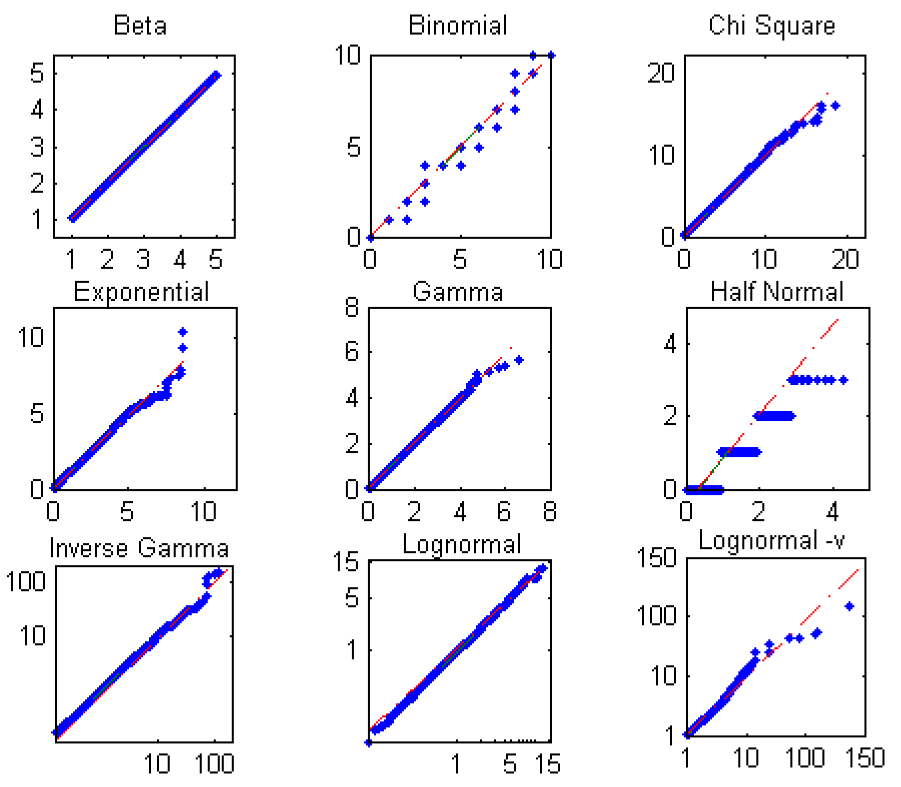

Figure 2a: QQplots of the samples generated from *Matlab* (x-axis) and from *MCSim* (y-axis). The red line joins the first and third quartiles of each distribution.

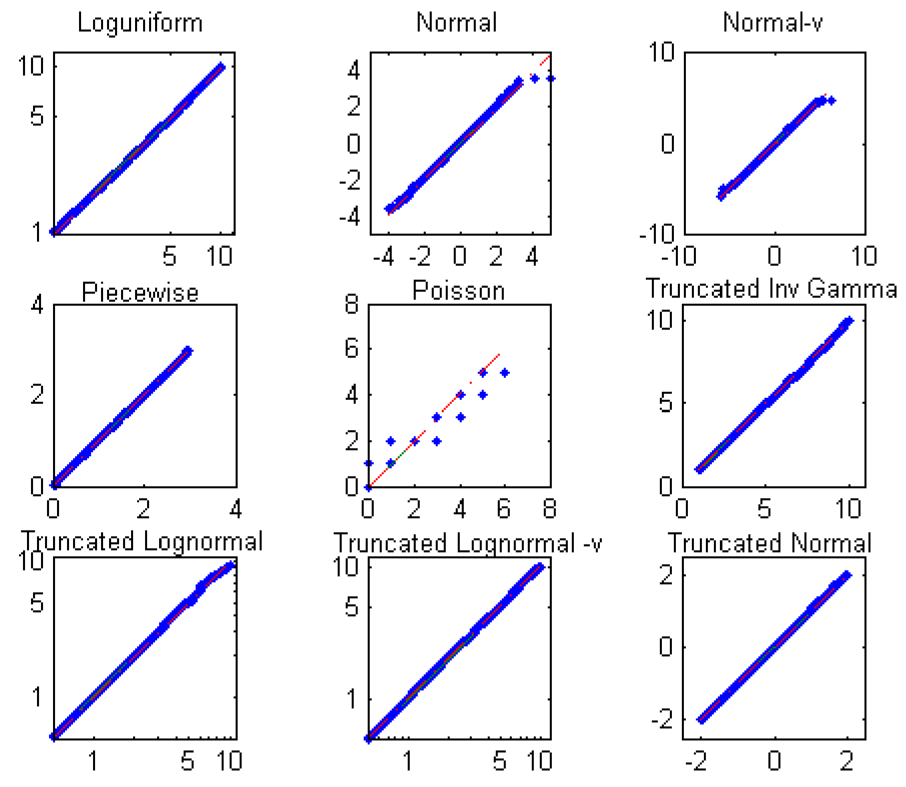

Figure 2b: QQplots of the samples generated from *Matlab* (x-axis) and from *MCSim* (y-axis). The red line joins the first and third quartiles of each distribution.

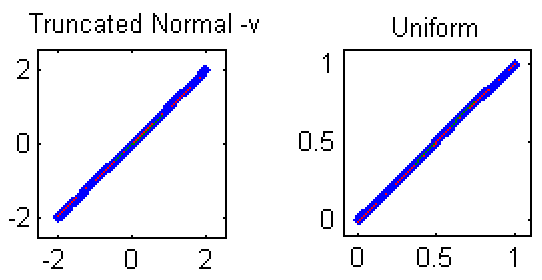

Figure 2c: QQplots of the samples generated from *Matlab* (x-axis) and from *MCSim* (y-axis). The red line joins the first and third quartiles of each distribution.

Figure 2a-c shows that the *MCSim* and *Matlab* samples are quite similar. We can then consider that these samples are drawn from the same distribution, except for the Half-Normal distribution. Indeed, in this version of *MCSim*, the generator produces integers, that is inconsistent with generating real numbers drawn from a Normal distribution which parameters are *m* and *s* and truncated to the interval [*m*, $+\Re$[.

6. CONCLUSIONS

The Lsodes integrator appears to be correctly implemented and gives the correct solution to a difficult stiff differential equation system, published in the scientific literature. Other simulations (data not shown) of equilibrium solutions for typical stiff compartmental differential systems have shown us that the numerical integration solution do not deviate by more than 1 in 10000 from the algebraic solution. In any case the reproducibility of the solutions obtained should be checked by varying the relative and absolute tolerance factors of the routine.

Most of the pseudo-random generators of *MCSim* version 5.0.0 can be considered valid, by comparison with a reference simulation software (Matlab version 7.0.0.19920, R14).

However, the Half-Normal generator of *MCSim* version 5.0.0 should NOT be used and that generator should imperatively be fixed in the next version of the software.

7. ANNEXES LIST

Repère	**Désignation**	**Nombre de pages**
Annex 1	*MCSim* model code and input file used for the simulation of a stiff coupled oscillators differential problem	1 A4
Annex 2	*MCSim* code used for the simulation of pseudo-random samples	1 A4
Annex 3	Matlab code used for the simulation of pseudo-random samples	4 A4

ANNEX 1

MCSim model code and input file used for the simulation of a stiff coupled oscillators differential problem

Model File

```
#------------------------------------------------------------------------------
# oscillators.model
#
# This is a simple test of the software based on the model studied in
# Skufca JD, 2004, Analysis still matters: a surprising instance of
# failure of Runge-Kutta-Felberg ODE solvers
# SIAM Review, 46(4):729-737.
#
# F Bois, copyright (c) 2005. All rights reserved.
#------------------------------------------------------------------------------

States = {theta_1, theta_2};

# Parameters

omega_1 = 1;
omega_2 = 1.5; # there is an notation error in eq 1 of the paper; it should
               # say omega_2 instead of omega_1...
k_1 = 1;
k_2 = 1;

Dynamics {
  dt(theta_1) = omega_1 + k_1 * sin(theta_2 - theta_1);
  dt(theta_2) = omega_2 + k_2 * sin(theta_1 - theta_2);
}
```

Simulation input file

```
#------------------------------------------------------------------------------
# oscillators.in
#
# This is a simple test of the software based on the model studied in
# Skufca JD, 2004, Analysis still matters: a surprising instance of
# failure of Runge-Kutta-Felberg ODE solvers
# SIAM Review, 46(4):729-737.
#
# F Bois, copyright (c) 2005. All rights reserved.
#------------------------------------------------------------------------------
OutputFile ("oscillators.out");

Simulation {
  theta_1 = 3;
  theta_2 = 0;

  PrintStep (theta_1, 0, 200, 0.2);
  PrintStep (theta_2, 0, 200, 0.2);
}
```

ANNEX 2

MCSim code used for the simulation of pseudo-random samples

```
#---------------------------------------------------------------
# random.in
#
# Input file for testing random number generators and associated
# functions.
#
# F Bois, copyright (c) 1998. All rights reserved.
#---------------------------------------------------------------

MonteCarlo ("random.out", 10000, 45776.43324);

Distrib(x_beta,          Beta, 2, 2, 1, 5);
Distrib(x_binomial,      Binomial, 0.5, 10);
Distrib(x_chi2,          Chi2, 2);
Distrib(x_exp,           Exponential, 1);
Distrib(x_gamma,         Gamma, 2, 2);
Distrib(x_halfnorm,      HalfNormal, 1);
Distrib(x_invgamma,      InvGamma, 2, 2);
Distrib(x_lognorm,       LogNormal, 1, 2);
Distrib(x_lognormv,      LogNormal_v, 1, 2);
Distrib(x_logunif,       LogUniform, 1, 10);
Distrib(x_norm,          Normal, 0, 1);
Distrib(x_normv,         Normal_v, 0, 2);
Distrib(x_piecewise,     Piecewise, 0, 1, 2, 3);
Distrib(x_poisson,       Poisson, 1);
Distrib(x_truncinvgamma, TruncInvGamma, 2, 2, 1, 10);
Distrib(x_trunclognorm,  TruncLogNormal, 1, 2, 0.5, 10);
Distrib(x_trunclognormv, TruncLogNormal_v, 1, 2, 0.5, 10);
Distrib(x_truncnorm,     TruncNormal, 0, 2, -2, 2);
Distrib(x_truncnormv,    TruncNormal_v, 0, 2, -2, 2);
Distrib(x_unif,          Uniform, 0, 1);

Simulation {
 Print (dummy, 1);
}

End.
```

ANNEX 3

Matlab code used for the simulation of pseudo-random samples

Main MATLAB program

```
% Checking MCSIM

close all
load random.out
Y = random;

X1 =  betarnd(2,2,1,10000)* (5-1) + ones(1,10000); %2 2 1 5 =
betarnd(2,2,1,10000)* (length_interval=max-min) + min
X2 =  binornd(10,0.5, 1,10000);
X3 =  chi2rnd(2, 1,10000);
X4 =  exprnd(1, 1,10000);
X5 =  gamrnd(2,1/2, 1,10000);
X6 =  HalfNormal( 1, 10000);
X7 =  ones(1,10000)./gamrnd(2,1/2, 1,10000);
X8 = lognrnd(log(1),log(2),1,10000);
X9 = lognrnd(log(1),2^0.5,1,10000); %X9 = lognrnd(log(1),log(2^0.5),1,10000);
ln_X10 = unifrnd(log(1),log(10), 1,10000);
X10 = exp(ln_X10);
X11 = normrnd(0,1,1,10000);
X12 = normrnd(0,2^0.5,1,10000);
X13 = piecewisernd(0, 1, 2, 3,10000);
X14 = poissrnd(1, 1,10000);
X15 = truncinvgammarnd(2,2,1,10, 10000);
X16 = trunclognormalrnd( 1, 2, 0.5, 10, 10000);
X17 = trunclognormalvrnd( 1, 2^0.5, 0.5, 10, 10000);
X18 = truncnormalrnd( 0, 2, -2, 2, 10000);
X19 = truncnormalrnd( 0, 2^0.5, -2, 2, 10000);
X20 = unifrnd(0,1,1,10000);

Y1 = Y(:,2);
Y2 = Y(:,3);
Y3 = Y(:,4);
Y4 = Y(:,5);
Y5 = Y(:,6);
Y6 = Y(:,7);
Y7 = Y(:,8);
Y8 = Y(:,9);
Y9 = Y(:,10);
Y10 = Y(:,11);
Y11 = Y(:,12);
Y12 = Y(:,13);
Y13 = Y(:,14);
Y14 = Y(:,15);
Y15 = Y(:,16);
Y16 = Y(:,17);
Y17 = Y(:,18);
Y18 = Y(:,19);
Y19 = Y(:,20);
Y20 = Y(:,21);

figure(1)
subplot(3,3,1,'align')
qqplot(X1,Y1)
axis([0.5 5.5 0.5 5.5])
axis square
subplot(3,3,2,'align')
qqplot(X2,Y2)
axis([0 10 0 10])
axis square
subplot(3,3,3,'align')
qqplot(X3,Y3)
```

```
axis([0 22 0 22])
axis square
subplot(3,3,4,'align')
qqplot(X4,Y4)
axis([0 12 0 12 ])
axis square
subplot(3,3,5,'align')
qqplot(X5,Y5)
axis([0 8 0 8])
axis square
subplot(3,3,6,'align')
qqplot(X6,Y6)
axis([0 5 0 5])
axis square
subplot(3,3,7,'align')
qqplot(X7,Y7)
axis([0 200 0 200])
axis square
subplot(3,3,8,'align')
qqplot(X8,Y8)
axis([0 20 0 20])
axis square
subplot(3,3,9)
qqplot(X9,Y9)
axis([0 150 0 150])
axis square

figure(2)
subplot(3,3,1,'align')
qqplot(X10,Y10)
axis([0 12 0 12])
axis square
subplot(3,3,2,'align')
qqplot(X11,Y11)
axis([-5 5 -5 5])
axis square
subplot(3,3,3,'align')
qqplot(X12,Y12)
axis([-10 10 -10 10])
axis square
subplot(3,3,4,'align')
qqplot(X13,Y13)
axis([0 4 0 4 ])
axis square
subplot(3,3,5,'align')
qqplot(X14,Y14)
axis([0 8 0 8])
axis square
subplot(3,3,6,'align')
qqplot(X15,Y15)
axis([0 11 0 11])
axis square
subplot(3,3,7,'align')
qqplot(X16,Y16)
axis([0 11 0 11])
axis square
subplot(3,3,8,'align')
qqplot(X17,Y17)
axis([0 15 0 15])
axis square
subplot(3,3,9,'align')
qqplot(X18,Y18)
axis([-2.5 2.5 -2.5 2.5])
axis square
%axis([-3 3 -3 3])

figure(3)
subplot(1,2,1,'align')
qqplot(X19,Y19)
axis([0 2.2 0 2.2])
axis square
```

```
subplot(1,2,2,'align')
qqplot(X20,Y20)
axis([0 1 0 1])
axis square

% Kolmogorov Smirnov
z = 0:0.01:20;
zz = chi2cdf(z,2);
Z = [z; zz];
[h,p,k,cv]=kstest(X3,Z')
```

Half Normal function

```
function z = HalfNormal( sd, size);

for i = 1:size
    x = normrnd(0,sd,1,1);
    while (x<0 )
        x = normrnd(0,sd,1,1);
    end
    z(i)=x;
end
```

Empirical cumulative density function of the Piecewise distribution

```
function z = piecewisecdf(x, Min,A,B,Max);

for i = 1 : length(x)
    zz(i) = piecewisepdf(x(i), Min, A, B, Max);
end
I = sum(zz);       %constante de normalisation
z = cumsum(zz)/I;
plot(x, z)
```

Probability density function of the Piecewise distribution

```
function z = piecewisepdf(x, Min,A,B,Max);

if x <= Min
    z = 0;
elseif (x > Min) & (x <= A)
    z = 1/(A-Min)*(x-Min);
elseif (x > A)   & (x <= B)
    z = 1;
elseif  (x > B) & (x <= Max)
    z = -1/(Max-B)*(x-B) + 1;
else
    z = 0;
end
```

Random generator of the Piecewise distribution function

```
function z = piecewisernd(Min,A,B,Max, n);

u = rand(1,n);

x_min = Min -0.001;
x_max = Max +0.001;

x_cdf  = x_min:0.001:x_max;
y_cdf = piecewisecdf(x_cdf,Min,A,B,Max);
for i = 1:n
     delta = y_cdf - ones(size(y_cdf))*u(i) ;
     delta_abs = abs(delta);
     [delt, I]   = min(delta_abs);
     z(i) = x_cdf(I);
end
```

Random generator of the truncated Inverse Gamma distribution function

```
function z = truncinvgammarnd(alpha,beta,Min,Max,size);

for i = 1:size
    x =  1/gamrnd(alpha,1/beta, 1,1);
    while (x<Min | x>Max )
        x =  1/gamrnd(alpha,1/beta, 1,1);
    end
    z(i)=x;
end
```

Random generator of the truncated lognormal distribution function

```
function z = trunclognormalrnd(Moy,Std,Min,Max,size);

for i = 1:size
    ln_x = normrnd(log(Moy), log(Std), 1,1);
    x = exp(ln_x);
    while (x<Min | x>Max )
       ln_x = normrnd(log(Moy), log(Std), 1,1);
       x = exp(ln_x);
    end
    z(i)=x;
end
```

Random generator of the truncated lognormal_v distribution function

```
function z = trunclognormalvrnd(Moy,Std,Min,Max,size);

for i = 1:size
    x = lognrnd(log(Moy),Std,1,1);
    while (x<Min | x>Max )
        x = lognrnd(log(Moy),Std,1,1);
    end
    z(i)=x;
end
```