User's guide for CECILE

Martin Choma, A. de Lataillade

November 4, 2002

Contents

1 Introduction **1**

2 Compiling and linking **4**
2.1 Decompressing package 4
2.2 Compiling and linking 5

3 The input files. **5**
3.1 Structure of input files 5

4 Output files **6**

5 Flowcharts **7**
5.1 Symbols description 7
5.2 Main file : *mcecile.f* 8
5.3 Subroutine genere_kg.f 11
5.3.1 Subroutine rass.f 12
5.3.2 subroutine gira.f 13
5.3.3 Subroutine chra.f 14
5.3.4 Subroutine unra.f 15
5.3.5 Subroutine urand.f 16
5.3.6 Subroutine cdss.f 17
5.3.7 Subroutine gicd.f 18
5.3.8 Subroutine stcd.f 19
5.3.9 Subroutine trss.f 20
5.3.10 Subroutine rags.f 21
5.4 Subroutine trajet.f 22

6 Subroutines description **24**
6.1 Main program mcecile.f 24
6.2 Subroutine description 24
6.2.1 Name: BLACK.F 24
6.2.2 Name:BLACK_DT 24
6.2.3 Name: CDSS.F 24

6.2.4 Name: CHRA.F . . . 25
6.2.5 Name: FICH.F . . . 25
6.2.6 Name: GENERE_B.F . . . 25
6.2.7 Name: GENERE_KG.F . . . 26
6.2.8 Name: GICD.F . . . 26
6.2.9 Name: GIRA.F . . . 26
6.2.10 Name:MODBGAZ.F . . . 26
6.2.11 Name: MODBGAZINTERP.F . . . 27
6.2.12 Name: MODBSUIE.F . . . 27
6.2.13 Name: PARAMBGAZ.F . . . 27
6.2.14 Name:PARAMIND.F . . . 27
6.2.15 Name: RAGS.F . . . 28
6.2.16 Name: RAND_UNIFORME.F . . . 28
6.2.17 Name: RASS.F . . . 28
6.2.18 Name: STCD.F . . . 28
6.2.19 Name: TIRB.F . . . 29
6.2.20 Name: TRAJET.F . . . 29
6.2.21 Name: TRAJET_FONC.F . . . 29
6.2.22 Name: TRAJET_SENS.F . . . 29
6.2.23 Name: TRSS.F . . . 30
6.2.24 Name: UNRA.F . . . 30
6.2.25 Name: URAND.F . . . 30
6.3 Include files description . . . 31
6.3.1 NAME: cecile.inc . . . 31
6.3.2 NAME: propradia.inc . . . 31
6.3.3 NAME: propradiabis.inc . . . 32
6.3.4 NAME: radiatif.inc . . . 32
6.3.5 NAME:entre.inc . . . 33

7 About bugs **33**

8 Monte Carlo method and random number generator **34**
8.1 Overview . . . 34
8.2 About random numbers . . . 34
8.3 Generation of Random Numbers . . . 35
8.3.1 Congruential Generators . . . 35
8.3.2 Shift-Register Generators . . . 35
8.3.3 Fibonacci generators . . . 36
8.3.4 Practical generators-Super Duper . . . 36
8.3.5 Unra generator in CECILE . . . 36
8.4 Testing random Number Sequences . . . 37
8.4.1 Testing of uniformity. . . . 38
8.4.2 Correlations in random numbers generators: Good's serial test . . . 40
8.4.3 Correlations in random number generators: Correlation coeficients . . . 44

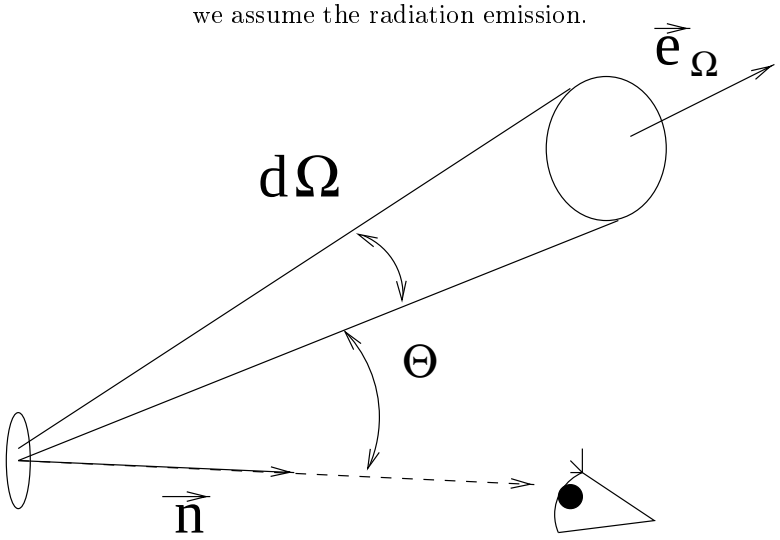
8.4.4 A composite test for correlations 46
8.4.5 A block test for correlations 48
8.4.6 Conclusions 50

9 References **50**

1 Introduction

Cecile is a software written in fortran for computing net exchange radiation between surfaces and volumes in one dimensional cylindric or plan system. Program uses Monte Carlo method with several thousands iterations for each volume. Concerning transfer of radiation, we define basic variable named intensity I.

There is an emission point, origin of radiation with elementary solid angle dΩ, and absorption surface. Observer has a viewing angle Θ throughout an elementary surface dS and there is an elementary spectrum dν through which we assume the radiation emission.



Observer receive a radiation quantity calculated as follows:

$$d\Phi = (I) * dS * d\Omega * cos\theta * d\nu$$

There is another, more interesting value named net exchange radiation, first time described in Hottel [*]. The net exchange radiation between two volumes is

$$\Psi_{I,J} = \int_{v=0}^{\infty} pdf_v dv \int_{V_i} pdf_{x_i} dV_i \int_{V_j} pdf_{x_j} dV_j \frac{(k_i \tau_{ij} k_j)}{||P_i P_j||^2} (I_{Bi} - I_{Bj})$$

which equals to

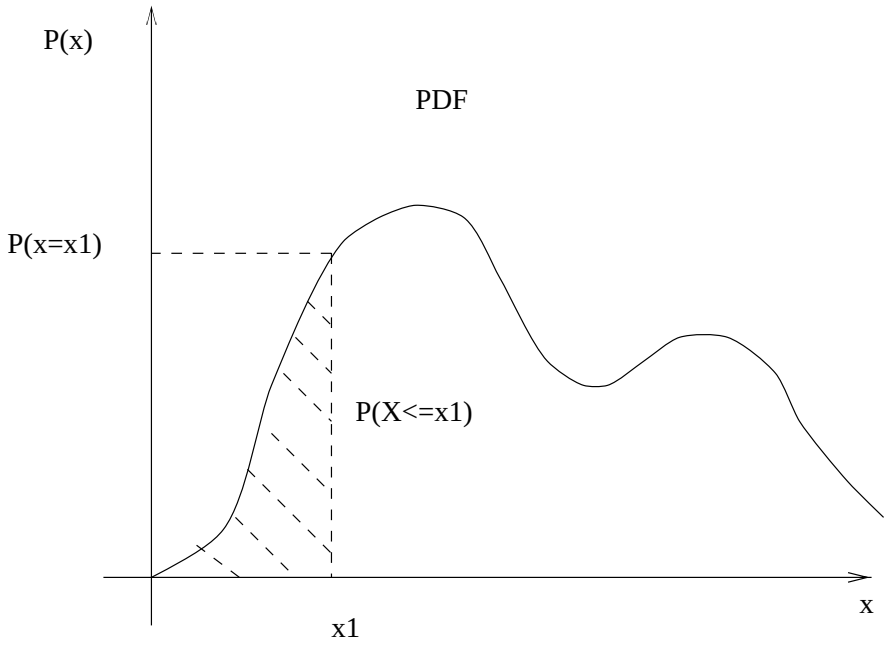
$$\Psi_{I,J} = \int_{v=0}^{\infty} pdf_v dv \int_{V_i} pdf_{x_i} dV_i \int_{0}^{4\pi} pdf_\Omega d\Omega \int_{S_j} pdf_{x_j} dS_j \frac{(k_i \tau_{ij} k_j)}{||P_i P_j||^2} (I_{Bi} - I_{Bj})$$

The discrete form of computing is $\Psi_{I,J} = \frac{\sum_{(v,P_i,P_j,\Omega)_{n=0}}^{10000} \Psi_{I,J}(v,P_i,P_j,\Omega)}{10000}$ with 10000 iterations

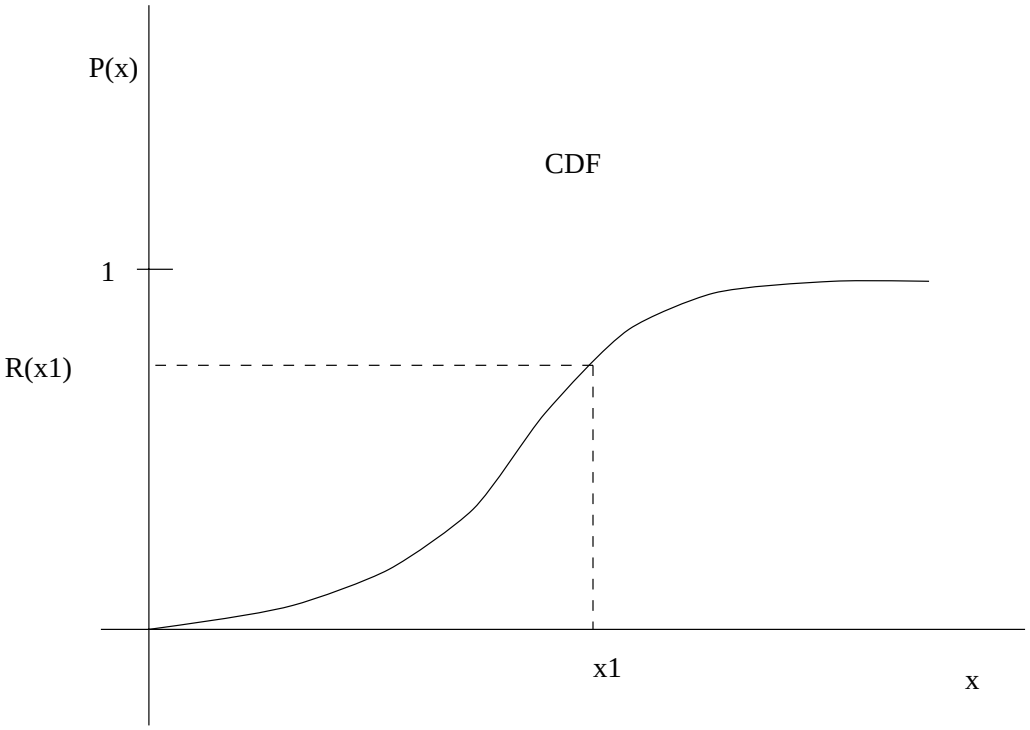
For each iteration, we have to random the multiplet (ν,Pi,Pj,Ω)

- $\Psi_{I,J}$- net exchange radiation between points I and J
- k$_i$- emission coefficient
- k$_j$- absorption coefficient
- I$_B$- black intensity
- τ_{ij}- transmitivity

In order to random all values, the probability density functions for the variables are used.



Because we can't determine the value of x from the probability of observing x, cumulative density functions are used instead. cdf: $P(x \leq x1) = R(x1) = \int_0^{x1} P(x)dx$



From the value of cumulative density function we can determine the value of x.

With computing e.g. monochromatic absorption coefficient, first the narrow band index in the pdf is determined, and each narrow band has its own pdf that is used for computing .

2 Compiling and linking

For compiling an executable file you'll need an FORTRAN 77 compiler. **Cecile** has been created and tested under Solaris 2.x environment. If you want to run **Cecile** under different platform, you will probably need to edit Makefile to set proper environment settings.

2.1 Decompressing package

1. Put the Cecile.tar.gz file into the directory you want to be installed

2. decompress file with the command :

   if you have GNU tar: *tar xzf cecile.tar.gz*

   if you don't have GNU tar : *gunzip cecile.tar.gz ; tar xvf cecile.tar*

2.2 Compiling and linking

Change to the cecile directory and for the default installation type

make cecile

This will create an executable file *cecile*. Note that all input, include and executable files should be in the same directory for proper execution of **CECILE** .

make clean will erase files not neccesary for running (such a *.o etc)

3 The input files.

In the **CECILE** directory you will find file *cecile.in* which is general input file and specifies other inputs/name of the input files.

3.1 Structure of input files

file *cecile.in*:

- first three lines : ignored - space for user comments - all lines not mentioned bellow are ignored
- 4th line: value of n - total number of volumes
- 7th line : file name where numbers of iterations -*ntir(0..n+1)* are stored (default - *n_tirage.in*)
- 10th line: file name where values of volumes - *r(0..n+1)* radiuses are stored (*rayons.in*) (the last two values are the same)
- 13th line : variable rsup_zero for cylindrical configuration - an important value to simulate parallel plan geometry
- 16th line :file name where value of pressure total- *ptot* is stored (*ptot.in*)
- 19th line: file name where values of temperature (in Kelvins)- *temp (0..n+1)* are stored (*temperat.in*)
- 22th line:file name where values of moll fractions of CO-*fm(1,1..n)* are stored (*fmco.in*)
- 25nd line:file name where values of moll fractions of CO2 -*fm(2,1..n)* are stored (*fmco2.in*)

- 28th line: file name where values of mollfractions of H20 - *fm(3,1..n)* are stored *(fmh20.in)*
- 31th line:file name where values of *fv (1..n)* are stored (*fv.in)*
- 34th line: logical value : true if spectral integration is used
- 37th and 40th line : needed if the variable in 34th line set to false
- 43th line:false if there are no interpolation profiles
- 46th line:true is value of psi are stored

file *SNBWN* :

- spectral values : *eta(ibande), delta_eta(ibande)*

file *SNBCO* (access by subroutine *parambgaz.f*)

- coefficient of absorption for CO : *kgb_piv(1,1..14,1..48), dinv(1,1..14,1..48)*

file *SNBCO2* (access by subroutine *parambgaz.f*)

- coefficient of absorption for CO2 : *kgb_piv(2,1..14,1..96), dinv(2,1..14,1..96)*

file *SNBH2O* (access by subroutine *parambgaz.f*)

- coefficient of absorption for H20: *kgb_piv(3,1..14,1..367), dinv(3,1..14,1..367)*

4 Output files

file *cecile.out:*

- describes names of the input files *ind.out, psi.out, dpsi.out*

file *ind.out*:

- imax=n jmax=n kmax=n

file *psi.out*:

- net exchange radiation: structure : *in, iin, r(in), r(iin), psi(in,iin), var_psi(in,iin)*

file *dpsi.out*:

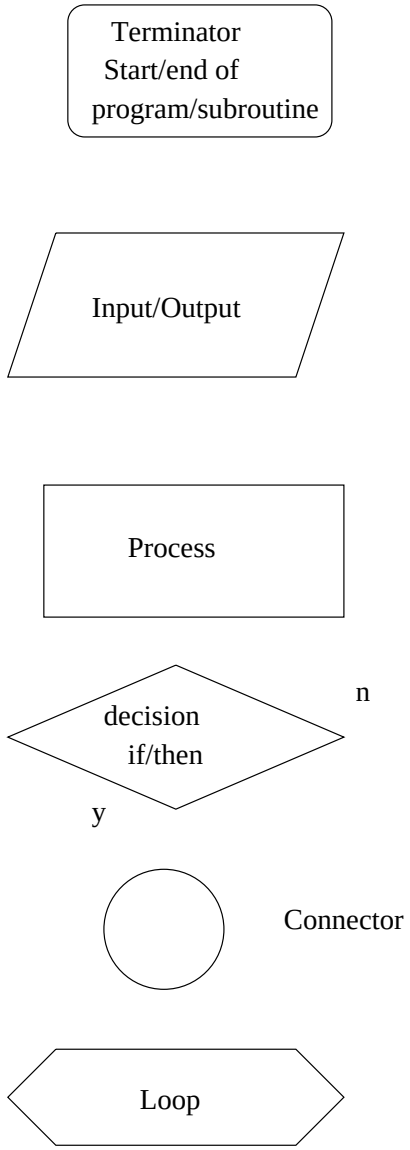
- relative net exchange radiation : structure : *in, iin,itmp1, dpsi6dfv(in,iin,itmp1), var_dpsi6fv(in,iin,itmp1)*

file *dpsidt.out*

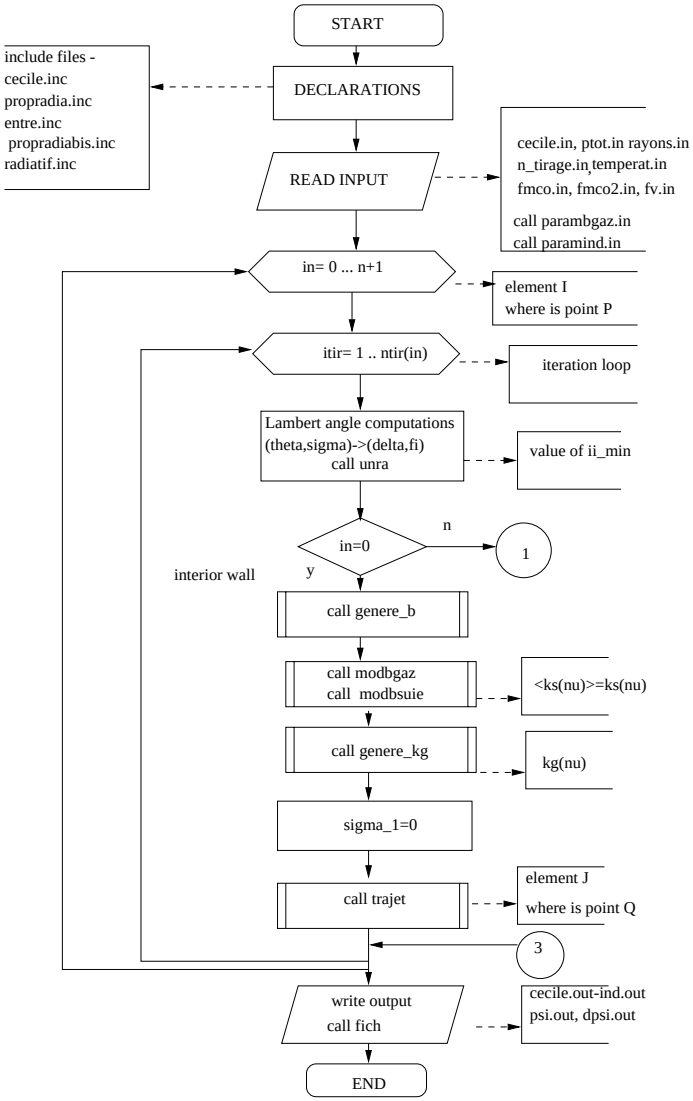
- relative net exchange radiation/temperature in, iin, itmp1, dpsi6dt(in,in,itmp1)

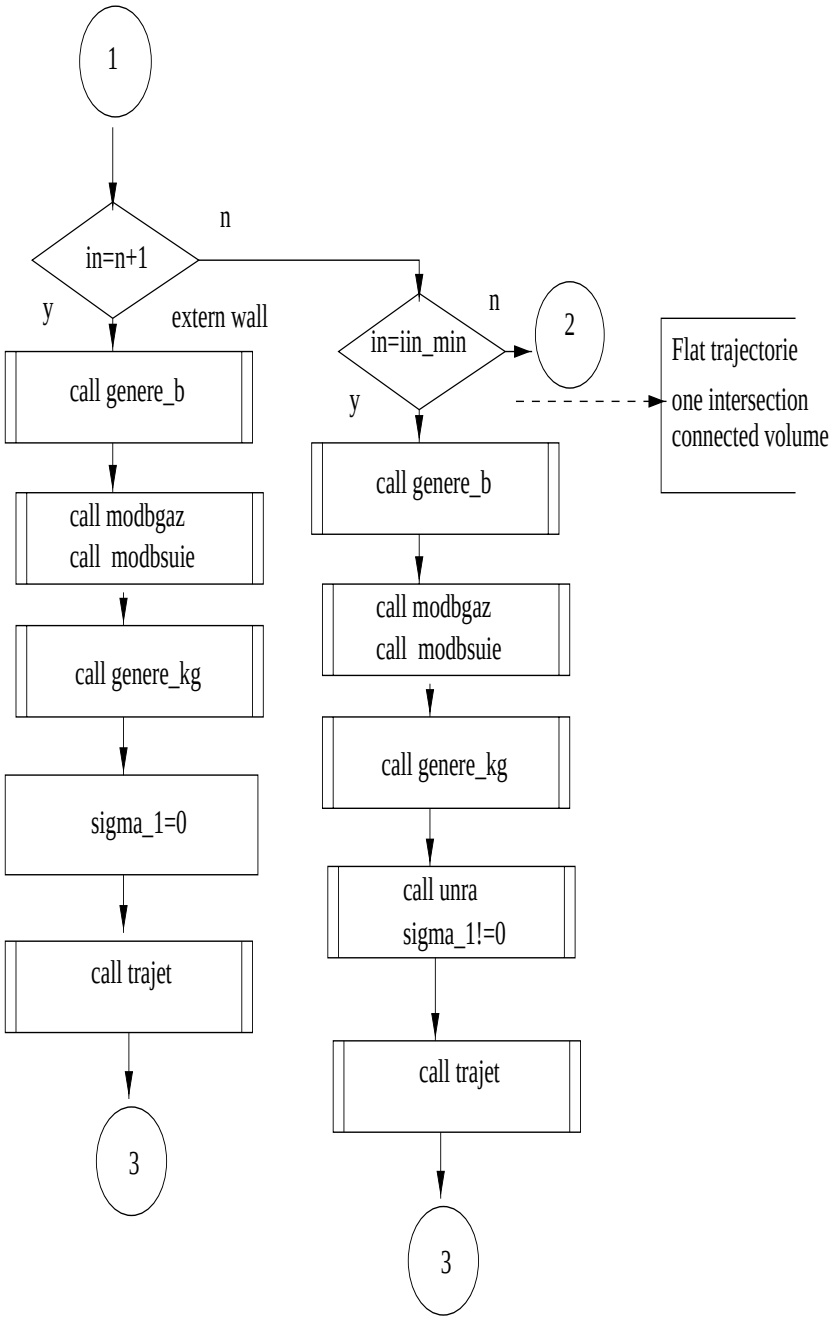
5 Flowcharts

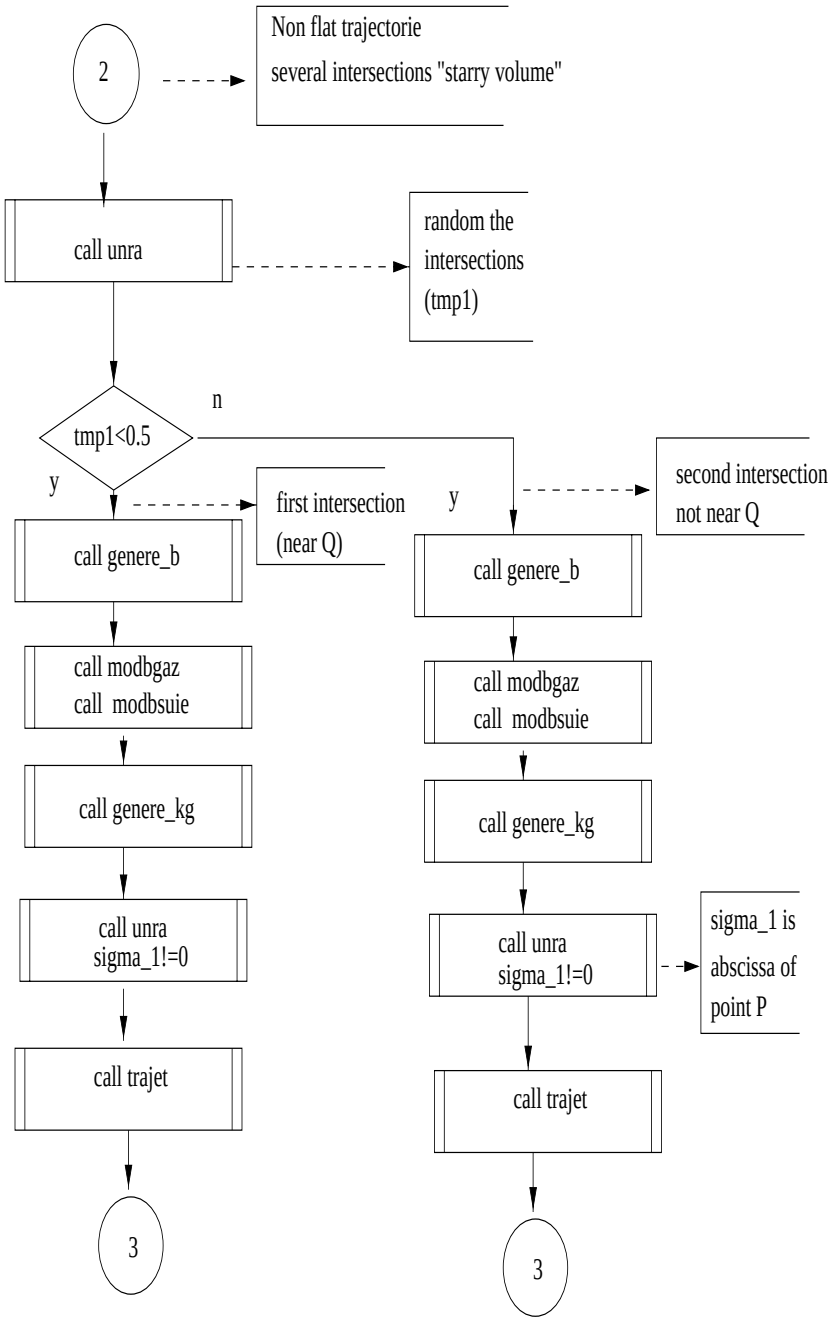
5.1 Symbols description



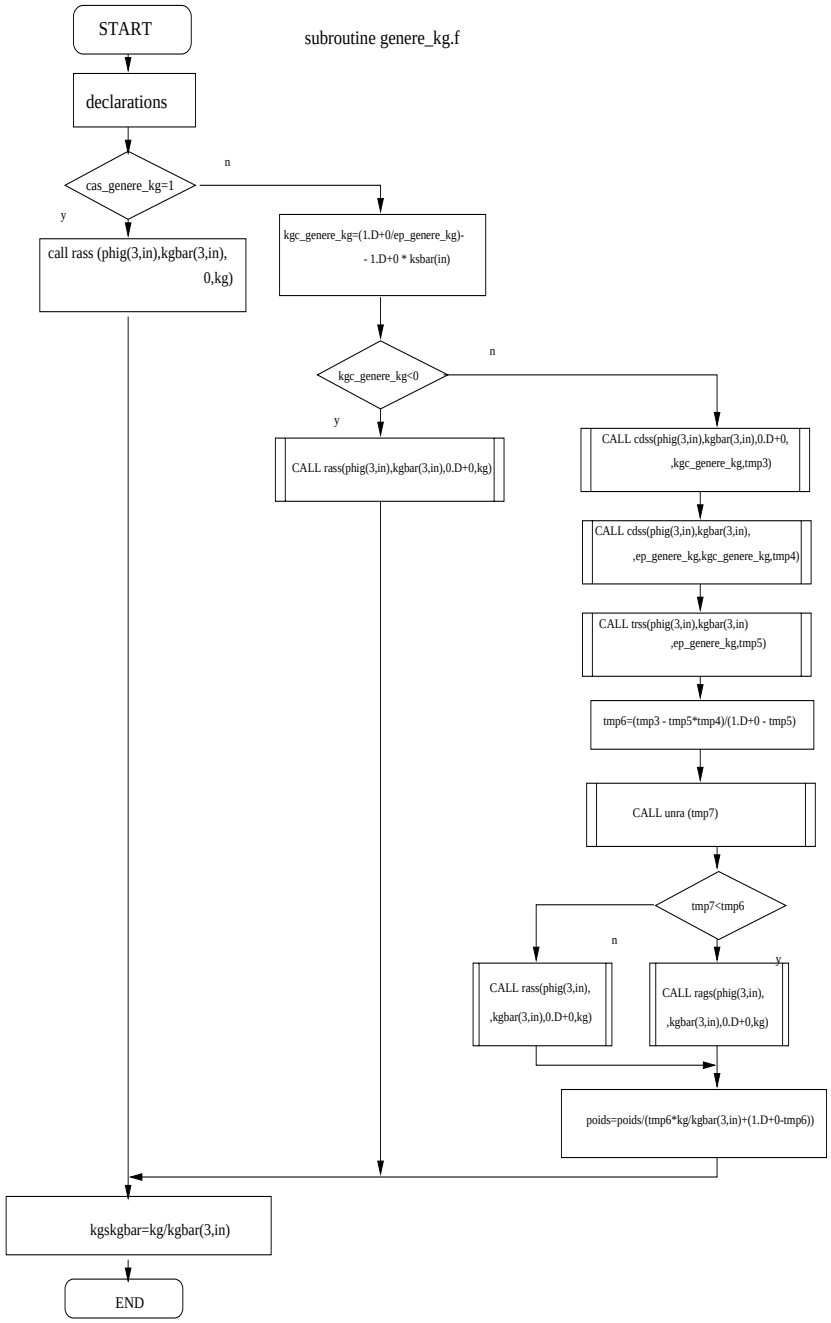
5.2 Main file : *mcecile.f*



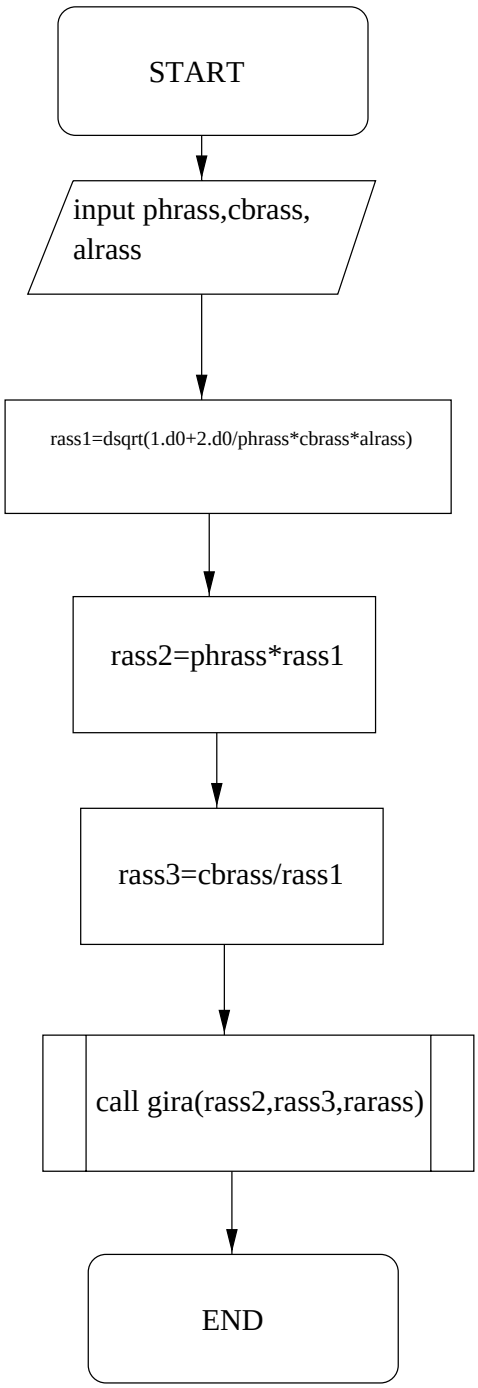

1
in=n+1
n
y
extern wall
call genere_b
call modbgaz
call modbsuie
call genere_kg
sigma_1=0
call trajet
3
in=iin_min
n
2
Flat trajectorie
one intersection
connected volume
y
call genere_b
call modbgaz
call modbsuie
call genere_kg
call unra
sigma_1!=0
call trajet
3


2
Non flat trajectorie
several intersections "starry volume"
call unra
random the
intersections
(tmp1)
tmp1<0.5
n
y
y
first intersection
(near Q)
second intersection
not near Q
call genere_b
call genere_b
call modbgaz
call modbsuie
call modbgaz
call modbsuie
call genere_kg
call genere_kg
call unra
sigma_1!=0
call unra
sigma_1!=0
sigma_1 is
abscissa of
point P
call trajet
call trajet
3
3

5.3 Subroutine genere_kg.f



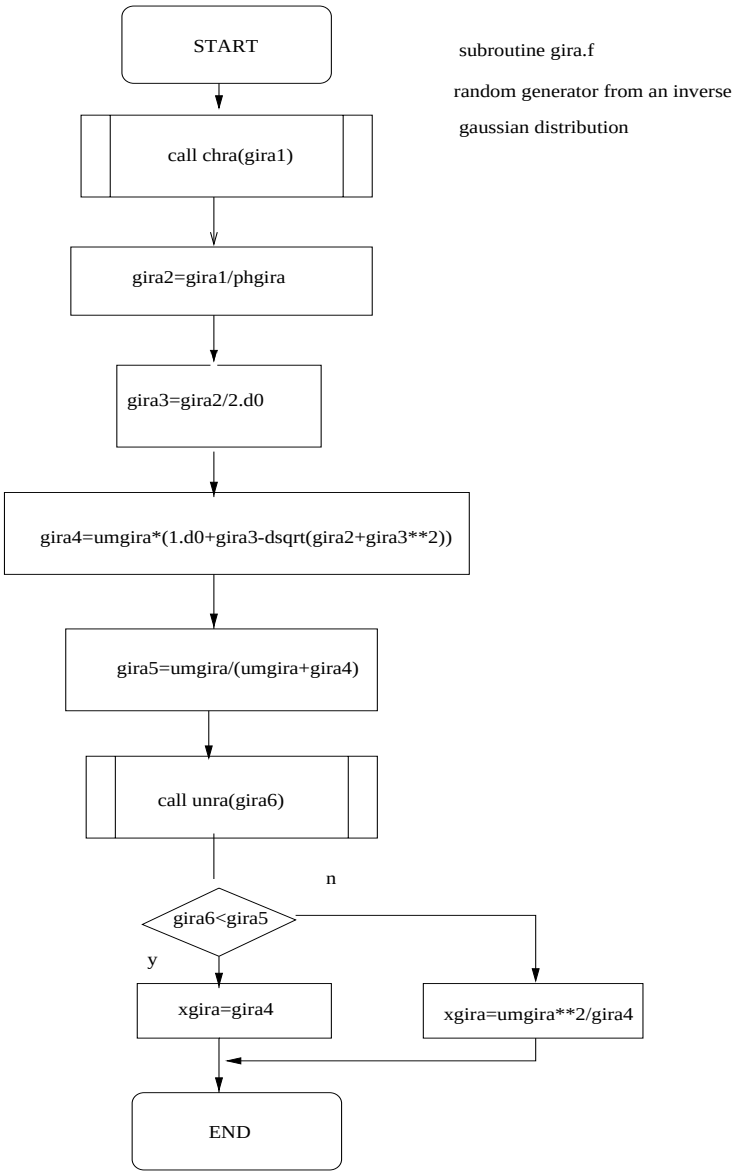
5.3.1 Subroutine rass.f



subroutine rass.f

k-sampling for surface-surface exchanges

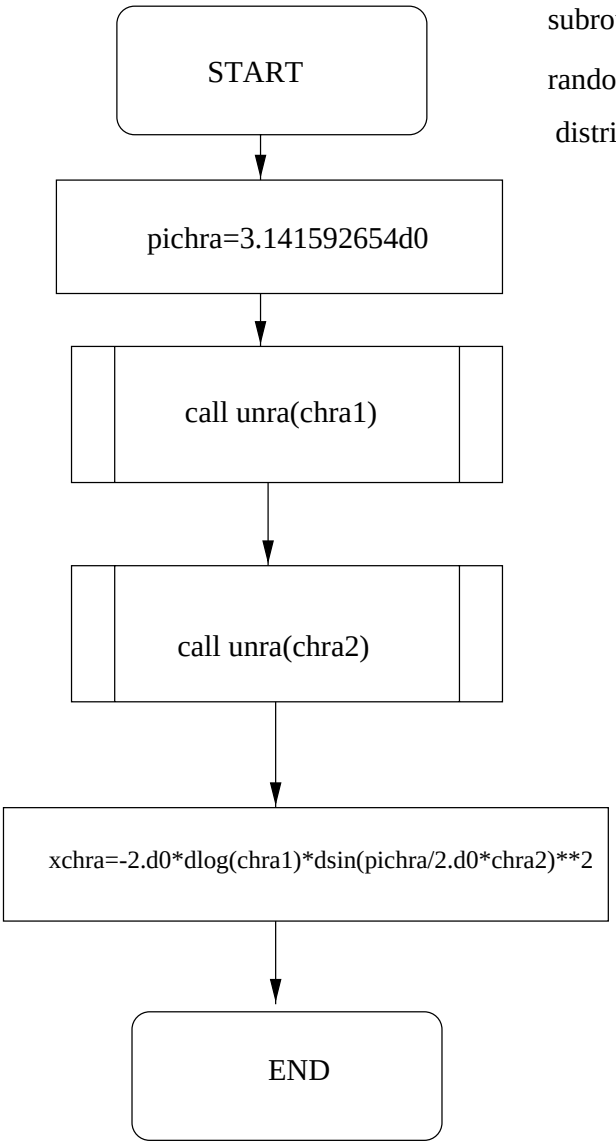
5.3.2 subroutine gira.f



subroutine gira.f

random generator from an inverse gaussian distribution

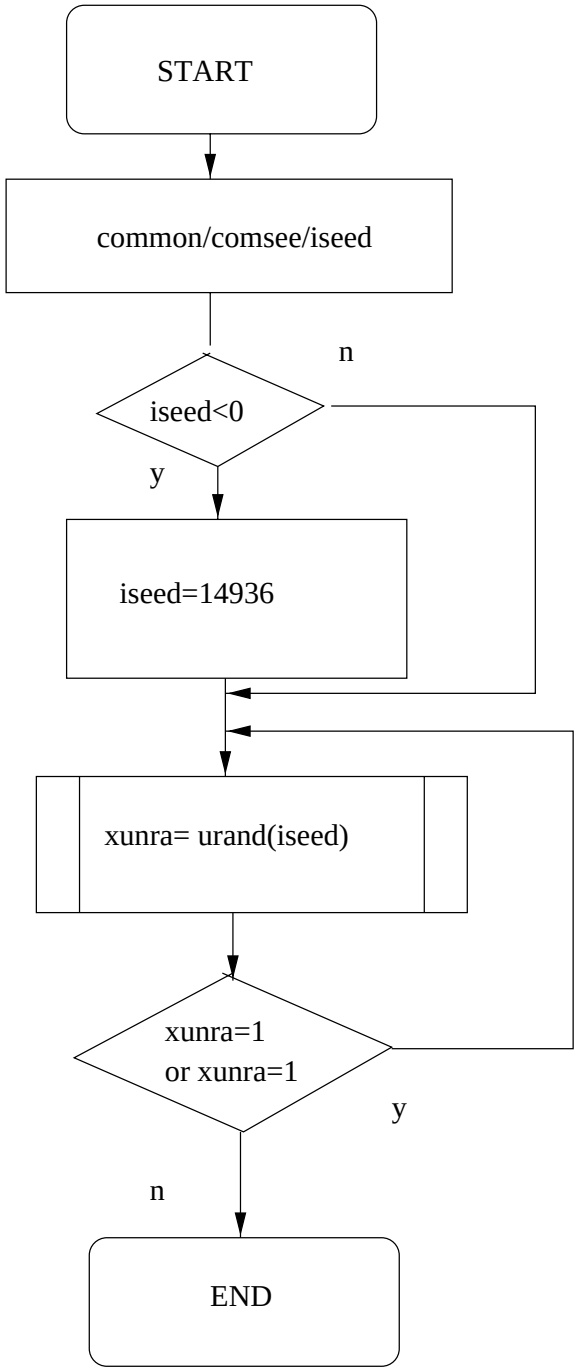
5.3.3 Subroutine chra.f



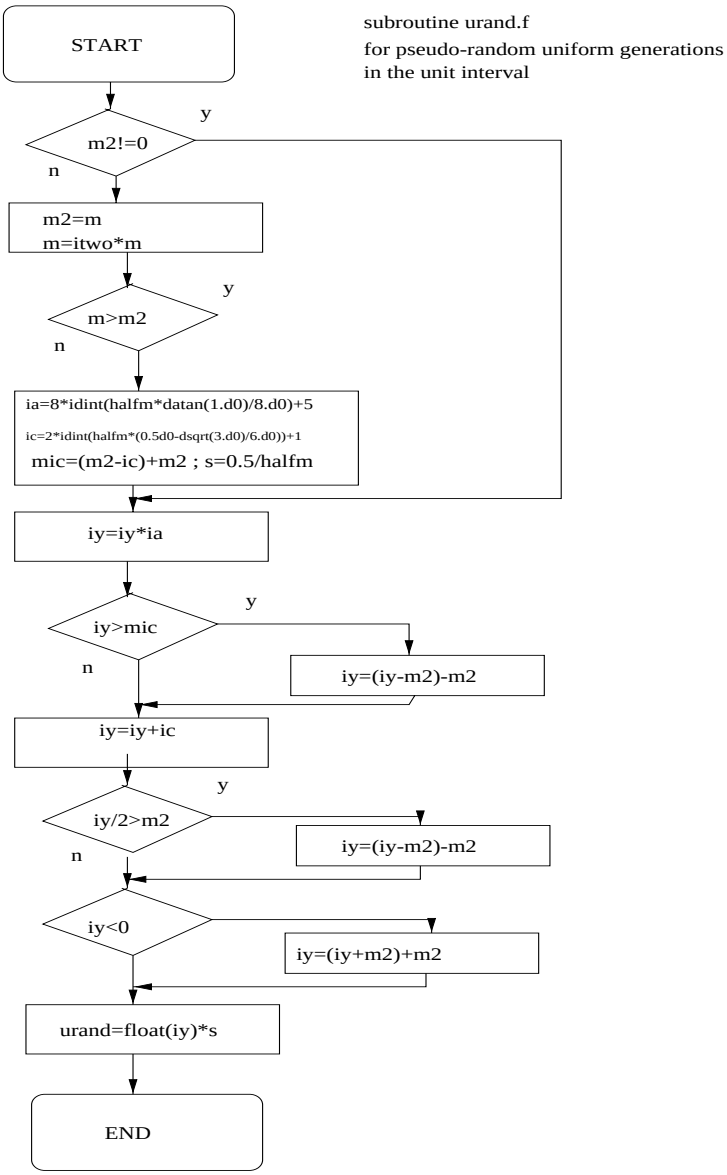
subroutine chra.f

random generator from a chi-square distribution

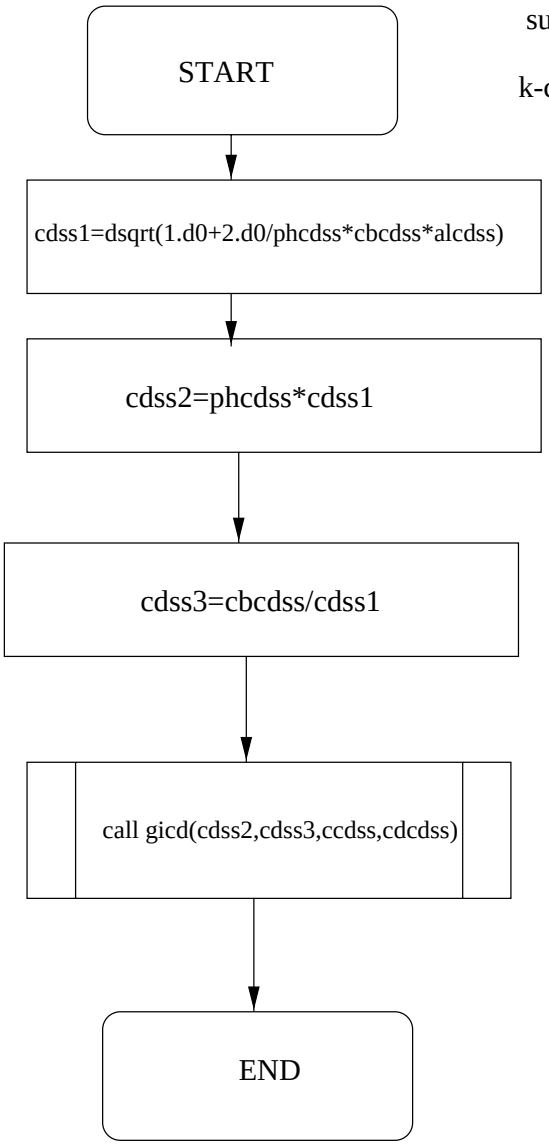
5.3.4 Subroutine unra.f



5.3.5 Subroutine urand.f



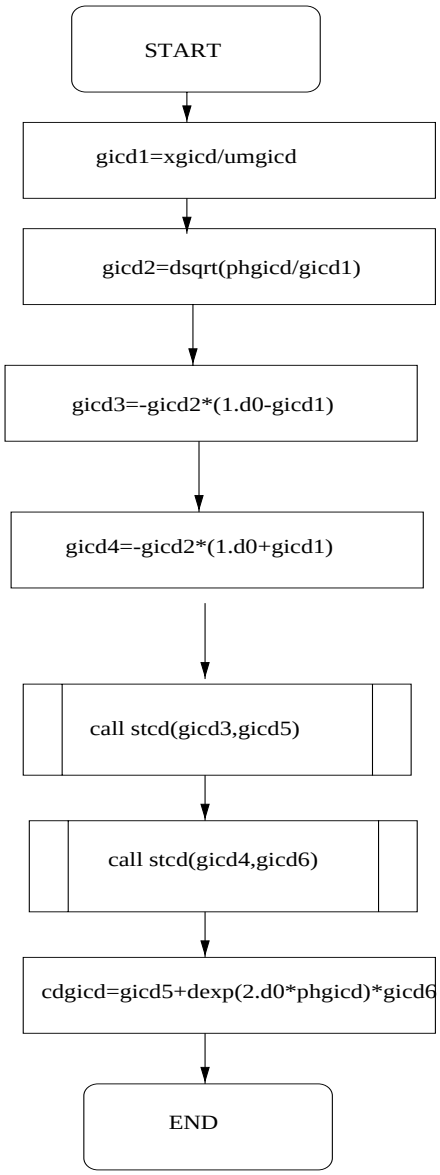
5.3.6 Subroutine cdss.f



subroutine cdss.f

k-cumulative for surface-surface exchanges

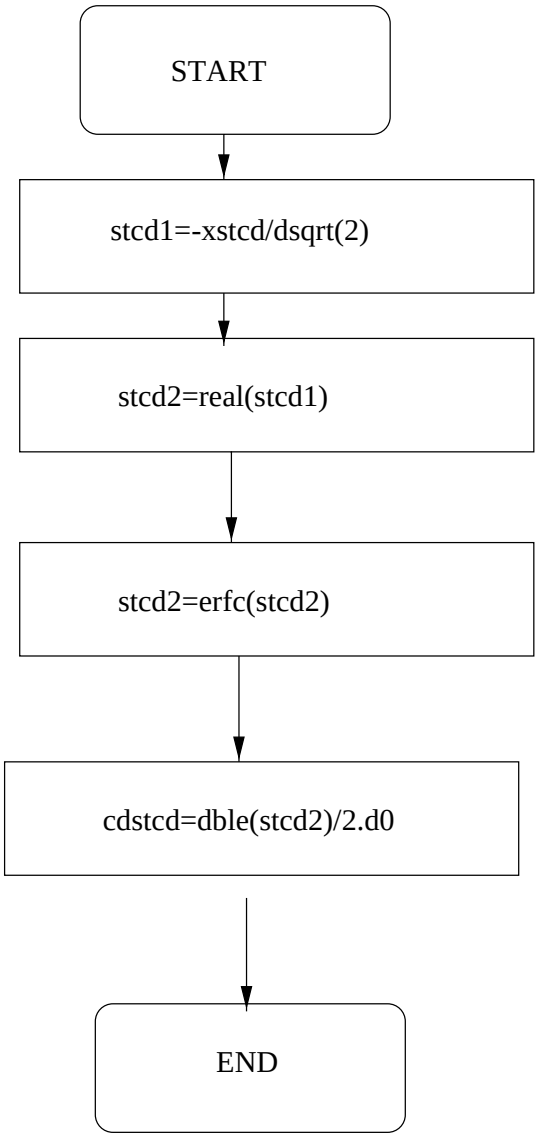
5.3.7 Subroutine gicd.f



subroutine gicd.f

inverse gaussian cumulative

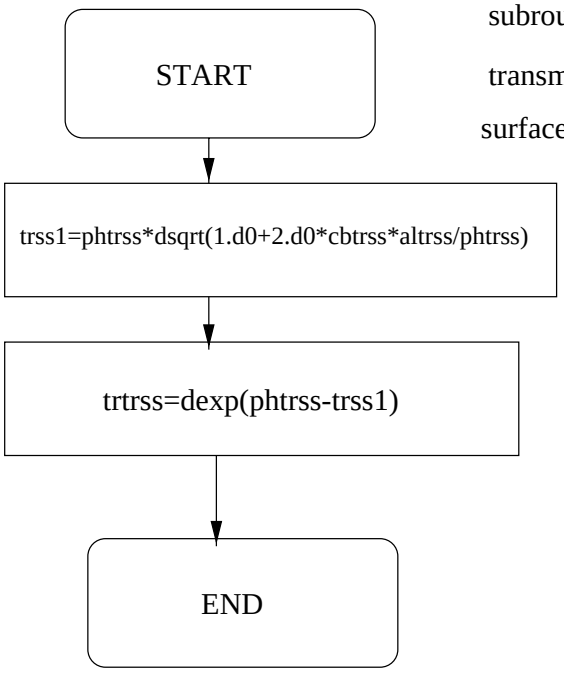
5.3.8 Subroutine stcd.f



subroutine stcd.f

cumulative of the standard normal

5.3.9 Subroutine trss.f

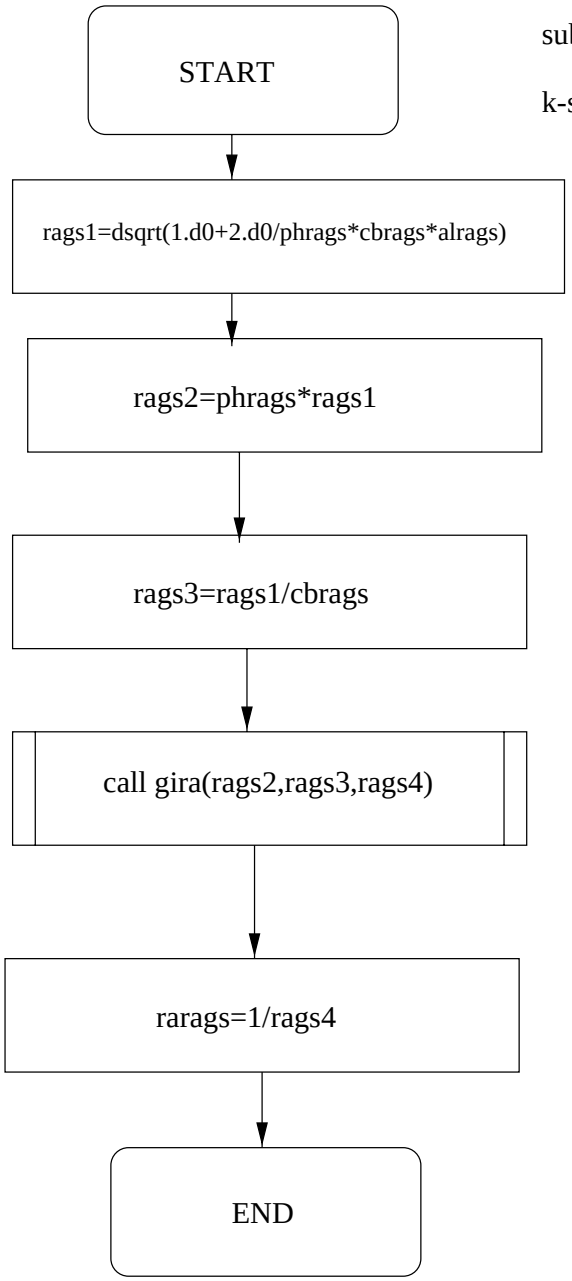


subroutine trss.f

transmission function for

surface-surface exchanges

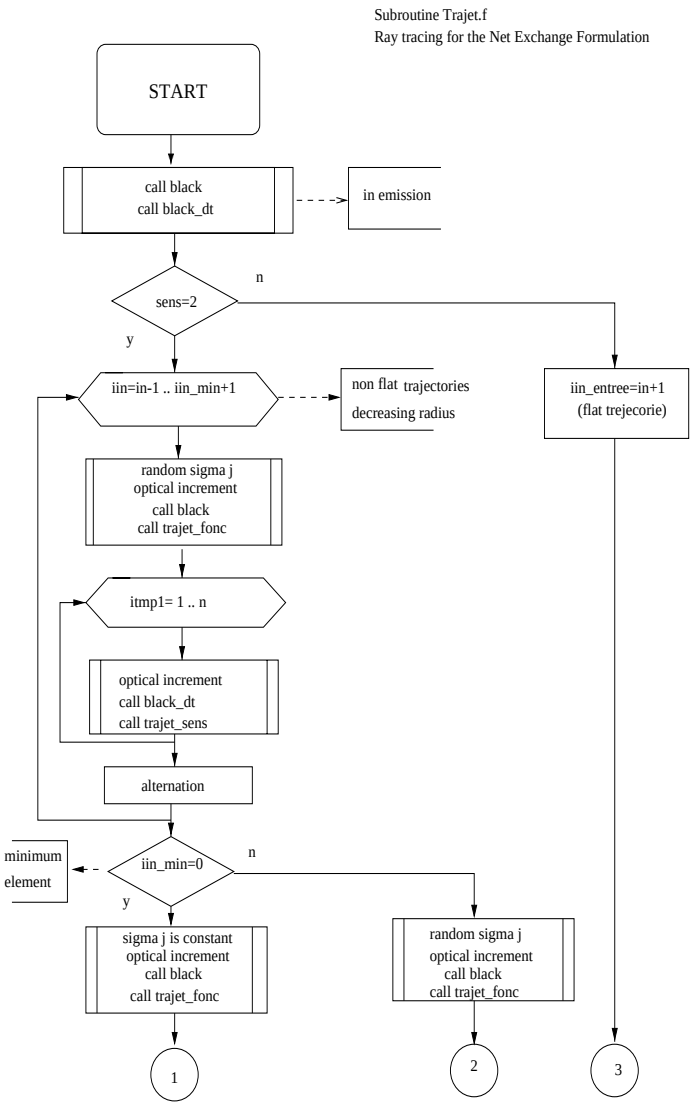
5.3.10 Subroutine rags.f

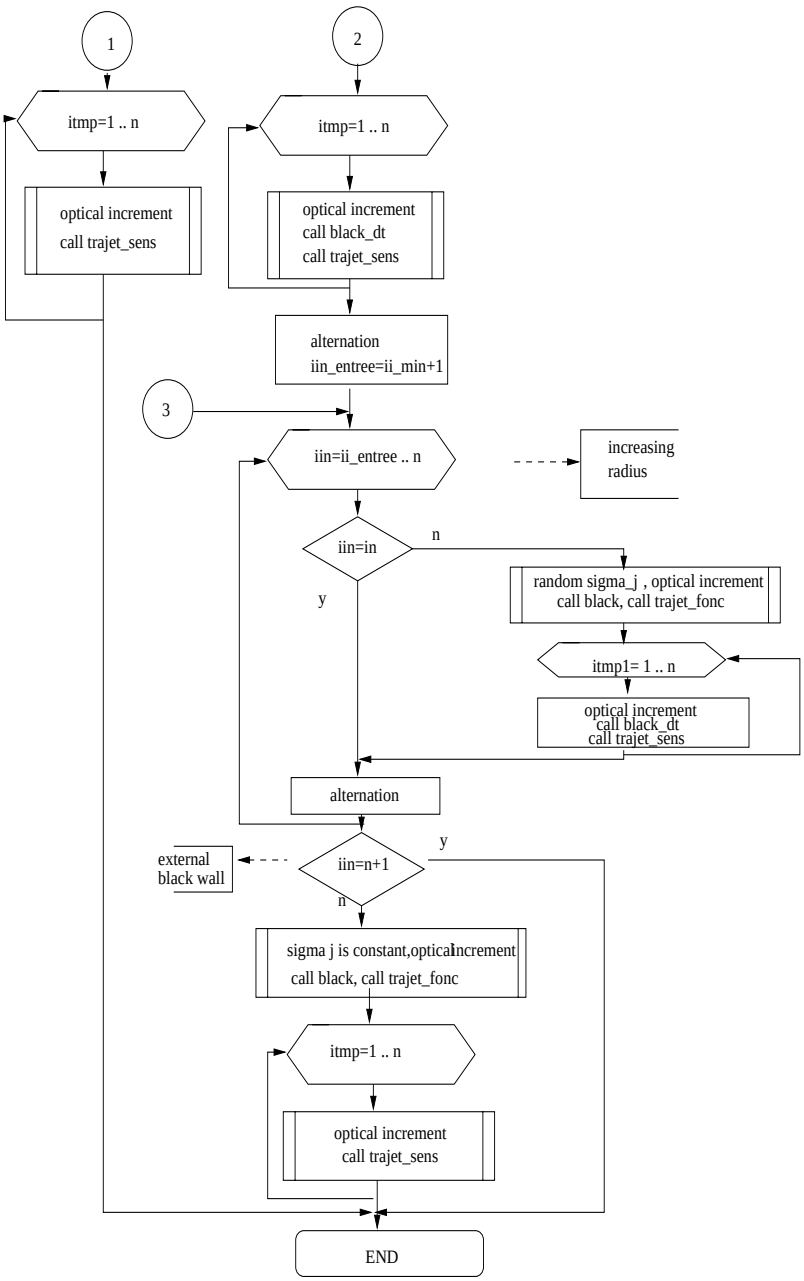


subroutine rags.f

k-sampling for gas-surface exchanges

5.4 Subroutine trajet.f




1
2
itmp=1 .. n
itmp=1 .. n
optical increment
call trajet_sens
optical increment
call black_dt
call trajet_sens
alternation
iin_entree=ii_min+1
3
iin=ii_entree .. n
increasing
radius
iin=in
n
y
random sigma_j , optical increment
call black, call trajet_fonc
itmp1= 1 .. n
optical increment
call black_dt
call trajet_sens
alternation
y
external
black wall
iin=n+1
n
sigma j is constant,optical increment
call black, call trajet_fonc
itmp=1 .. n
optical increment
call trajet_sens
END

6 Subroutines description

6.1 Main program mcecile.f

Description**:** main program, all other subroutines are called from mcecile.f

Variables:

Input variables : described in cecile.in

Output variables: described in cecile.out

Include files:cecile.inc, propradia.inc, propradiabis.inc, radiatif.inc, entre.inc

6.2 Subroutine description

6.2.1 Name: BLACK.F

Description**:** Monochromatic black intensity computed from wave frequency (blanu) and temperature (blat). Planck formula is used.

Variables:

Input variables : blanu,blat

Output variables: blae

Include files:none

6.2.2 Name:BLACK_DT

Description**:** Monochromatic black intensity sensitivity / temperature.

Variables:

Input variables : blanu,blat

Output variables: dblae

Include files:none

6.2.3 Name: CDSS.F

Description**:** k-cumulative for surface-surface exchanges

Variables:

Input variables : phcdss,alcdss,cbcdss, ccdss

Output variables: cdcdss

Include files:none

6.2.4 Name: CHRA.F

Description**:** random generator from chi-square distribution

Variables:

Input variables : none

Output variables: xchra

Include files:none

6.2.5 Name: FICH.F

Description**:** write results to output files

Variables:

Input variables :

Output variables:

Include files:cecile.inc

6.2.6 Name: GENERE_B.F

Description**:** Randomize the narrow band index (ibande)

Variables:

Input variables : in,

Output variables: ibande

Include files:cecile.inc,propradia.inc

6.2.7 Name: GENERE_KG.F

Description**:** Randomize the monochromatic absorption coefficient for H20,Co2 and C0

Variables:

Input variables : cas_genere_kg,ep_genere_kg, phig, kgbar

Output variables: kgskgbar, kg

Include files:propradia.inc, cecile.inc, radiatif.inc

6.2.8 Name: GICD.F

Description: inverse Gaussian cumulative

Variables:

Input variables : phgicd,umgicd,xgicd

Output variables: cdgicd

Include files:none

6.2.9 Name: GIRA.F

Description: random generator from inverse Gaussian distribution

Variables:

Input variables : phgira, umgira

Output variables: xgira

Include files:none

6.2.10 Name:MODBGAZ.F

Description: calculation of the transmissivity - narrow band model for gases

Variables:

Input variables : kgb_piv, dinv_piv, eta

Output variables: kgb6p, dinv, gamma, phi

Include files:cecile.inc, propradia.inc,propradiabis.inc

6.2.11 Name: MODBGAZINTERP.F

Description: calculates the interpolation coefficients

Variables:

Input variables : TM

Output variables: RT,IT

Include files:none

6.2.12 Name: MODBSUIE.F

Description: Narrow band model for soot

Variables:

Input variables : eta,fv

Output variables: ksbar

Include files:propradia.inc, cecile.inc, radiatif.inc, entre.inc

6.2.13 Name: PARAMBGAZ.F

Description:Subroutine reads the model parameters fo

Variables:

Input variables : none

Output variables: kgb_piv, dinv_piv

Include files:propradia.inc

6.2.14 Name:PARAMIND.F

Description: Subroutine searches the parameter indexes corresponding the wave number 'wvnb'.

Variables:

Input variables : none

Output variables: wvnb, ico2, ih2o,ico.lico.lico2,lih2o

Include files:propradia.inc

6.2.15 Name: RAGS.F

Description: k-sampling for gas-surface interface

Variables:

Input variables : phrags, cbrags, alrags

Output variables: raragS

Include files:none

6.2.16 Name: RAND_UNIFORME.F

Description: Random double precision numbers generator.

Variables:

Input variables : none

Output variables:xunra

Include files:none

6.2.17 Name: RASS.F

Description: k-sampling for surface-surface conditions

Variables:

Input variables : phrass, cbrass, alrass

Output variables: rarass

Include files:none

6.2.18 Name: STCD.F

Description: cumulative of the standart normal

Variables:

Input variables : xstcd

Output variables: cdstcd

Include files:none

6.2.19 Name: TIRB.F

Description: Random generator for the narrow band index (ibande)

Variables:

Input variables : p(nbande_mx)

Output variables: numbande

Include files:cecile.inc, propradia.inc

6.2.20 Name: TRAJET.F

Description: Ray tracing for the Net Exchange Formulation

Variables:

Input variables : poids,dpdksp,k

Output variables: psi, var-psi

Include files:cecile.inc, radiatf.inc, entre.inc

6.2.21 Name: TRAJET_FONC.F

Description: Calculation of psi

Variables:

Input variables : in, iin, w, bi , bj

Output variables: psi, var-psi

Include files:cecile.inc, propradia.inc

6.2.22 Name: TRAJET_SENS.F

Description: Calculation of derivation psi/soot volume fraction and temperature

Variables:

Input variables : in, iin,itmp1, w, bi , bj, w_dk, bi_dt, bj_dt

Output variables: dpsi6dfv, var_dpsi6dfv, dpsi6dt, var_dpsi6dt

Include files:cecile.inc, propradia.inc

6.2.23 Name: TRSS.F

Description: transmission functions, k-distributions, k-cumulatives, random k-generators

Variables:

Input variables : phtrss, cbtrss,altrss

Output variables: trtrss

Include files:none

6.2.24 Name: UNRA.F

Description: Random double precision numbers generator.

Variables:

Input variables :none

Output variables: xunra

Include files:none

6.2.25 Name: URAND.F

Description: pseudo-random uniform generations in the unit interval

Variables:

Input variables : iy

Output variables: (urand)

Include files:none

6.3 Include files description

6.3.1 NAME: cecile.inc

Description: general include file

Variables:

variable	dimension
n	1
in	1
iin	1
iin_min	1
test	1
n_mx	1
r	0:n+1
sin_teta	1
delta	1
f	1
long	0:n+1
sigma_1	1
sigma1_star	1
sigma_2	1
ntir	0:n+1
itir	1
prob1	1
prob2	1
poids	1
dpdksp	0:n+1
pi	1
bilan	0:n+1
var_bilan	0:n+1
psi	0:n+1,0:n+1
psir	0:n+1,0:n+1
psirm	0:n+1,0:n+1
var_psi	0:n+1,0:n+1
dpsi6dfv	0:n+1,0:n+1,1:n
var_dpsi6dfv	0:n+1,0:n+1,1:n
dpsidkgbar	0:n+1,0:n+1,1:n
dpsi6dt	0:n+1,0:n+1,1:n
vardpsi6dt	0:n+1,0:n+1,1:n

6.3.2 NAME: propradia.inc

Description: spectral values include file

Variables:

variable	dimension
ngaz_mx	1
ntemp_mx	1
ntemp	1
nbande_mx	1
nbande	1
ibande	1
lico	ibande
lico2	ibande
lih2o	ibande
ico	ibande
ico2	ibande
ih20	ibande
eta	ibande
delta_eta	ibande
kgb_piv	3,I,J
dinv_piv	3,I,J

6.3.3 NAME: propradiabis.inc

Description: Radiative properties not needed by EM2C programs

Variables:

variable	dimension
kgb6p	3,ibande
dinv	3,ibande
kgb6fv	3,ibande
gamma	3,ibande
phi	3,ibande

6.3.4 NAME: radiatif.inc

Description: Radiative transfer variables

Variables:

variable	dimension
k	iin
ks	n
ksbar	iin
kg	1
kgbar	3,n
phig	3,n
kgskgbar	1
emi	1
emj	1
demi	1
demj	1

6.3.5 NAME:entre.inc

Description: definition of input variables

Variables:

variable	dimension
ptot	1
temp	n
fm	3,n
fv	n

7 About bugs

In certain cases program can generate NaN (Not a Number) values as an output. It is caused by improper inputs in certain calculations, such as input value 0 in the cumulative computations. If you find a bug, please don't hesitate to contact us at email address *Amaury.Lataillade@enstimac.fr* and describe your experiences with the problem. We don't feel the program is bug free and any hints are appreciated.

Here are some examples of non proper behaviour of CECILE:

Surface:

- **genere_kg.f** - when kgbar = 0, division by zero occured, fixed by putting the condition not to call genere_kg when kgbar=0
- **rass.f** - in certain cases when variable phrass ~ 0 (very small number) the result of division was infinity, fixed by re-aranging formula by moving division to the end.

- **gira.f** - using a square of variable umgira produces infinity values in case of big value of umgira -> fixed by putting division first then multiplication.

Volume:

- **rass.f** - the same problem as for surface case
- **rags.f** - if the value of rags4 is equal to 0 the inverted value causes division by zero, because output value rarags is similar to cbrags (k), in such case we set rarags equal to cbrags (k)

8 Monte Carlo method and random number generator

8.1 Overview

Monte Carlo method solves a problem by generating suitable random numbers and observing that fraction of the numbers obeying some property or properties. The method is useful for obtaining numerical solutions to problems which are too complicated to solve analytically.

8.2 About random numbers

All random number generators are based upon specific mathematical algorithms, which are repeatable and sequential. As such, the numbers are just pseudorandom. Here, for simplicity, we shall term them just "random" numbers, subject to this realization. Formally,

- Truly random - is defined as exhibiting "true" randomness, such as the time between "tics" from a Geiger counter exposed to a radioactive element.
- Pseudorandom - is defined as having the appearance of randomness, but nevertheless exhibiting a specific, repeatable pattern.
- Quasi-random - is defined as filling the solution space sequentially (in fact, these sequences are not at all random - they are just comprehensive at a preset level of granularity). For example, consider the integer space [0, 100]. One quasi-random sequence which fills that space is 0, 1, 2,...,99, 100. Another is 100, 99, 98,...,2, 1, 0. Yet a third is 23, 24, 25,..., 99, 100, 0, 1,..., 21, 22. Pseudorandom sequences which would fill the space are pseudorandom permutations of this set (they contain the same numbers, but in a different, "random" order).

In cecile, we use 'pseudorandom' numbers based on linear congruential generators. Below you will find the test of that generators and some conclusions.

8.3 Generation of Random Numbers

8.3.1 Congruential Generators

Congruential generators are based on relation with m,n and ν non-negative integers,

$$\nu = n \bmod m$$

which defines ν as being congruent to n modulo m. It means that ν is the remainder when n is divided by m, or $\nu = n\text{-j}m$ where j is the largest integer consistent with ν being non-negative. In particular we note that $m = 2$ gives the bits 0 and 1. The only possible value that ν can assume are the integers in the range 0 to m-1. A sequence of integers $\{\nu_i\}$ can be generated from this relation by taking for n some function of the previous number ν_{i-1}, i.e.

$$\nu_i = \text{f}(\nu_{i-1}) \bmod m$$

Provided an initial seed $\nu_0$is supplied, a string of distinct values can be generated. The most commonly used generator for pseudorandom numbers is the Linear Congruential Generator

$$\nu_i = (a\nu_{i-1} + \text{c}) \bmod m$$

By taking $\xi_i = \nu_i/m$ we find a sequence of numbers distributed on [0,1].

In the linear congruential generator, only one previous random variable is used to derive the next one, we can use more of the earlier generated numbers, each associated with a specified multiplier - a compound generator. If the last k generated numbers are used, we can write $\nu_i = (a.n^{(i-1)} + c) \bmod \text{m}$

where $n^{(i-1)} = (\nu_{i-1}, \nu_{i-2}, ..., \nu_{i-k})$ and the set of multipliers is a= (a_1,a_2,..,a_k). By taking a value of just 2 for k and suitable multipliers $\text{a}_1$and a_2, a compound generator with acceptable result structure can be obtained, described next.

8.3.2 Shift-Register Generators

Compound generator of equation of above type, with m = 2, i.e generation of bits, known as shift-register, or Tauseworth, generator. We can write these bits as

$$\text{b}_i = (a_1 b_{i-1} + a_2 b_{i-2} + ... + a_{p-1} b_{i-p+1} + b_{i-p}) \bmod 2$$

where p indicates the earliest previous bit included in the sequence ($\text{a}_p = 1$), and the first p bits are provided; the constant c has been set equal to zero. There is a maximum of $2^p - 1$distinct possible iterations before sequence repeats itself.Interesting is the existence of sequences with only two non-zero coefficients, i.e. b_i is of the form $\text{b}_i = (\text{b}_{i-q} + b_{i-q}) \bmod 2 = \text{b}_{i-q} \oplus b_{i-p}$

where bs are only single bits, $\oplus$denotes the 'exclusive or' operation.

8.3.3 Fibonacci generators

The prescription described in the last section of constructing a lagged sequence can be done modulo m > 2 e.g. taking the p*th* and q*th* preceding numbers with, say, p>q, and combining them, mod m, according to some operation

$$\nu_i = (\nu_{i-q} \odot \nu_{i-p}) \text{mod m}$$

We can generate a sequence of numbers, given that the first p numbers are provided e.g. from a multiplicative congruential generator. The operation denoted by $\odot$ is conventionally the 'exclusive-or'. It may be however just simple addition or subtraction, and recent studies favour the latter operation, giving rise to a subtracted Fibonacci generator.And especially interesting development is the subtract-with-borrow generator of this type. The algorithm is $\nu_i = (\nu_{i-q} - \vec{\nu}_{i-p} - \beta) \text{ mod m}$

The 'carry' coefficient, β, which must be set to 0 or 1, arbitrarily, at initialization, is reassigned in each call as follows; if the quantity in brackets is negative, so that m must be added to it to carry out the modulo m operation, β is set to zero otherwise it assumes the value unity. If m, p, and q satisfy certain conditions, the period of this generator is m^p-m^q.

8.3.4 Practical generators-Super Duper

Most modern generators involve the combination of two or more of the elementary generators described above, an improvement both from the point of enhancing the statistical quality of the numbers and increasing the period of the generator. One such is called Super-Duper, originally written at McGill University, Canada. It combines the numbers produced by a linear congruential generator with one produced by a shift-register generator and has a period in excess of 10^{18}. The former uses equation ν_i=(aν_{i-1}+ c) mod m with m=2^{32}, a=69069 and c=0, with the seed ν_0, an odd integer to produce a 32 bit integer ν_i. The shift register employs equation b_i=($b_{i-q} + b_{i-q}$)mod 2 with p=32 and q=17 to produce bits, b_k, sequences of 32 of which are combined to form an integer μ_i,

$$\mu_i = \sum_{k=1}^{32} b_k 2^{k-1}$$

These two integers are combined bit-wise 'exclusive or' and scaled to give a variate on [0,1], as, $\xi = 2^{-32}(\nu_i \oplus \mu_i)$

8.3.5 Unra generator in CECILE

Unra is a uniform random number generator based on theory and suggestions given in d.e. Knuth (1969), vol 2. The integer iy (seed) should be initialized to an arbitrary integer prior to the first call to urand. The calling program should not alter the value of iy between subsequent calls to unra.Values of unra are returned in the interval (0,1). The recursive algorithm of unra is

ia=8*idint(k*datan(1)/8)+5
ic=2*idint(k*(0.5-$\sqrt{3}$/6))+1
iy=2*(ia*iy+ic)/k

where k=1.07374182E+09 and idint returns only integer part of number

8.4 Testing random Number Sequences

In this section we will test three random number generators with uniform outputs [0,1]. We will compare generator from cecile Unra with standart fortran Ran function, and the Super Duper described above.

In statistics, a **GOODNESS OF FIT** test used to compare two distributions. For nominal or "binned" measurements, a chi-square test is common. For ordinal or ordered measurements, a Kolmogorov-Smirnov test is appropriate.

We will use these tests for consistency of distribution

Chi-square test: the best known goodness of fit statistic. Chi-square is a way to numerically compare two sampled distributions:

- Some number of "bins" is selected, each typically covering a different but similar range of values.
- Some much larger number of independent observations are taken. Each is measured and classified in some bin, and a count for that bin is incremented.
- Each resulting bin count is compared to an expected value for that bin, based on the expected distribution. Each expectation is subtracted from the corresponding bin count, the difference is squared, then divided by the expectation:

$$\chi 2 = \sum \frac{(Observed_i - Expected_i)^2}{Expected_i^2}$$

The sum of all the squared normalized differences is the chi-square statistic, and the distribution depends upon the number of bins through the degrees of freedom or df. The df value is normally one less than the number of bins (though this will vary with different test structures). Ideally, we choose the number of bins and the number of samples to get at least ten counts in each bin. For distributions which trail off, it may be necessary to collect the counts (and the expectations) for some number of adjacent bins.

The chi-square c.d.f. tells us how often a particular value or lower would be seen when sampling the expected distribution. Ideally we expect to see chi-square values on the same order as the df value, but often we see huge values for which there really is little point in evaluating a precise probability.

Kolmogorov-Smirnov test: another goodness of fit test for comparing two distributions. Here the measurements need not be collected into "bins," but are instead re-arranged and placed in order of magnitude:

- N independent samples are collected and sorted in numerical order in array X as $x_1 \ldots x_N$, $x_{i+1} \geq x_i$
- $P(x_i)$ is the cumulative distribution of the sample: the fraction of the n observations which are less than or equal to x_i, $P(X \leq x_i) = i / N$
- F(X) is the reference cumulative distribution, the probability that a random value will be less than or equal to x. Here we want $F(x_i)$, the fraction of the distribution to the left of x_i which is a value from the array.
- we define the statistic $q = \sqrt{N} max|F(x_i) - P(x_i)|$
- If agreement of the two distributions will be acceptable subject to it being not less likely than f% (e.g. f=5, corresponding to less than 5% probability of the theoretical distribution, randomly sampled, giving rise to the recorded data), for the large value of N (in practice N≥30, f can be approximated by equation $f = 100e^{-2q^2}\{1 - 2q/3\sqrt{N} + O(N^{-1})\}$

Chi-Square Distribution: distribution tested are often compared to chi-square distribution, which is defined, with ν degrees of freedom as

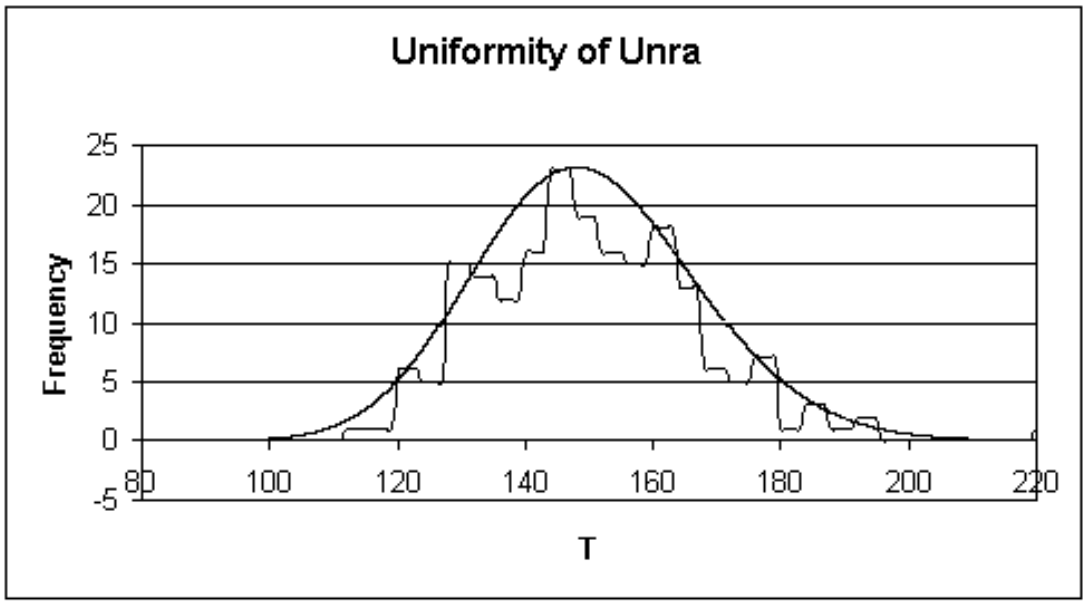
$$p_{\chi^2,\nu}(z) = (2^{\nu/2}\Gamma(\nu/2))^{-1} z^{(\nu/2)-1} e^{-z/2}$$

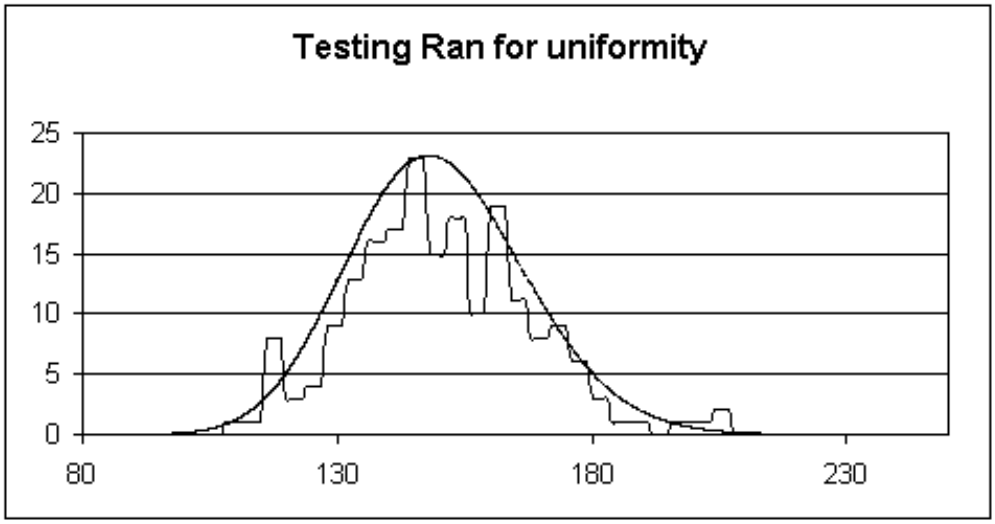
8.4.1 Testing of uniformity.

We use a string of N=10000 variates, numbers are histogrammed into k= 151 cells and the statistic T is calculated

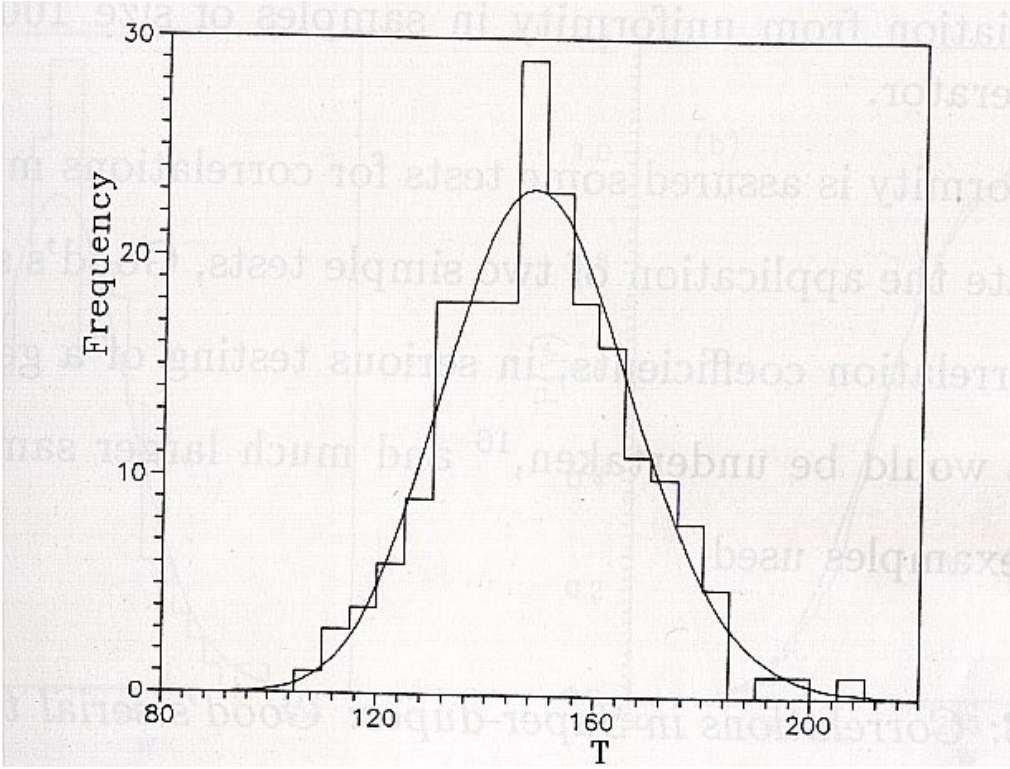
$$T = \overline{n}^{-1} \sum_{i=1}^{k} (n_i - n)^2$$

where $\overline{n} = N/k$, it should follow the χ^2distribution with ν=150 degrees of freedom. This was done for 200 samples, the histogram of these values is compared with $p_{\chi^2,\nu}(T)$ in following pictures.





Uniformity of Super-duper:



The cumulative frequency from the histogram was compared with $P_{\chi^2,\nu}(T)$ and the Kolmogorov-Smirnov Statistic q, defined earlier, formed.

- Unra q=0.486 -> f= 60.1 %.
- Ran q=0.5144 -> f=56.73 %
- Super Duper q=0.606, f=46 %

where f% means probility of obtaining larger value than q in such a comparision with the predicted χ^2distribution. Since this is the test of uniformity, we can conclude there is no significant evidence of deviation from uniformity in samples of size 10000 drawn from Super Duper and Ran generator. Unra, however has larger deviation and it can be considered not as uniform as other two generators.

8.4.2 Correlations in random numbers generators: Good's serial test

For each of 200 samples, each sample has 10000 variates, we take an overlapping pairs of numbers. The N-1 points (ξ_i, ξ_{i+1}), i=1,...,N-1 were histogrammed into k^2 sub squares with k=16, so for each from 16^2subsquares we have number $n_{i,j}$,i=1..16,j=1..16, which equals number of occurencies of pairs in those square. the following quantities were formed,

$$e_1 = (N-1)/k,\ e_2 = e1/k,\ c_i = \sum_{j=1}^{k} n_{ij}$$
$$t_1 = \sum^{k} (c_{ii=1} - e_1)^2/e_1$$

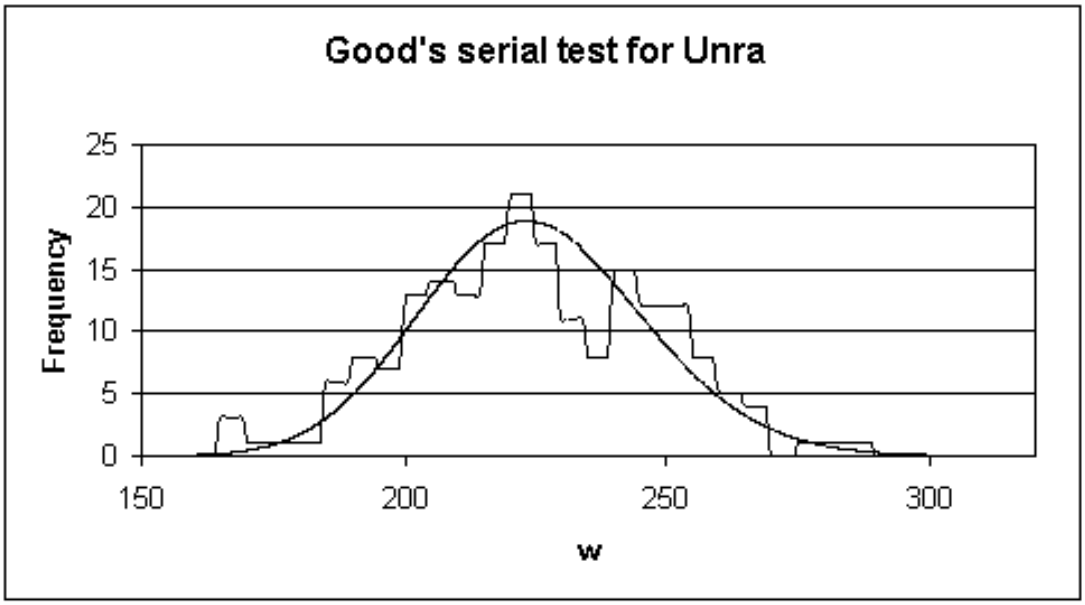
and

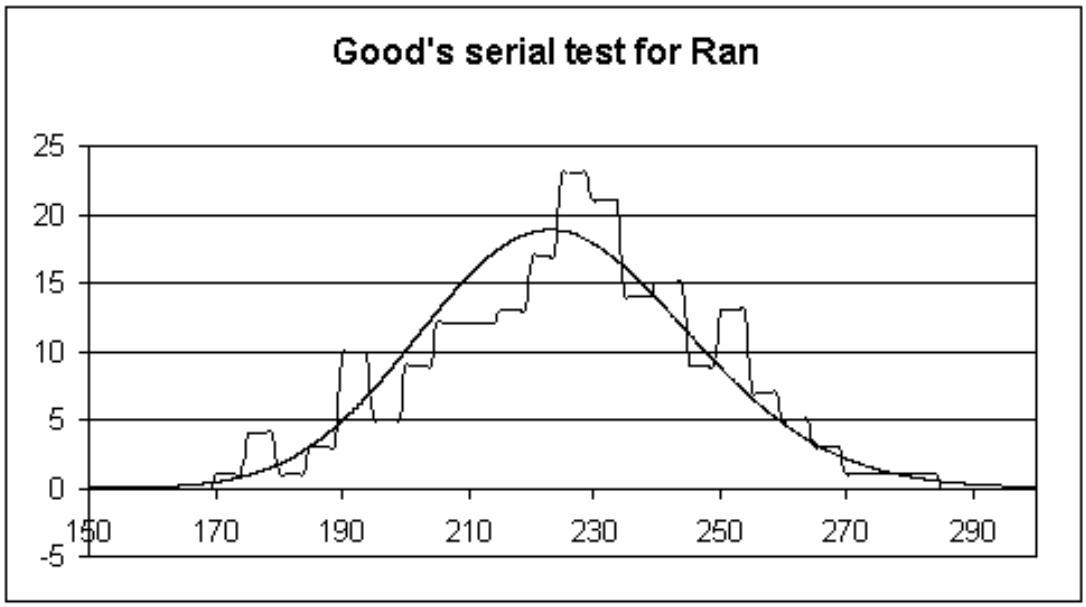
$$t_2 = \sum_{i=1}^{k} \sum_{j=1}^{k} (n_{ij} - e_2)^2/e_1$$

The statistic w is defined by

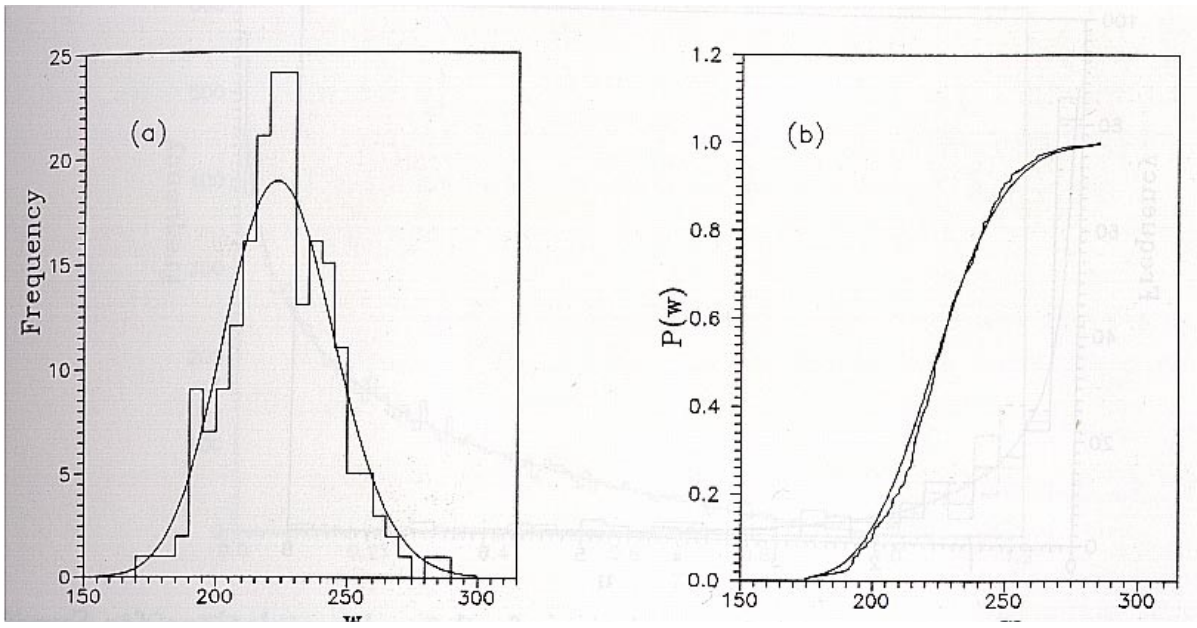
$$w = t_2 - t_1$$

The distribution of this walues are shown in following pictures with comparision with $p_{\chi^2,\nu}(w)$ for ν=k(k-2)+1= 225 degrees of freedom.

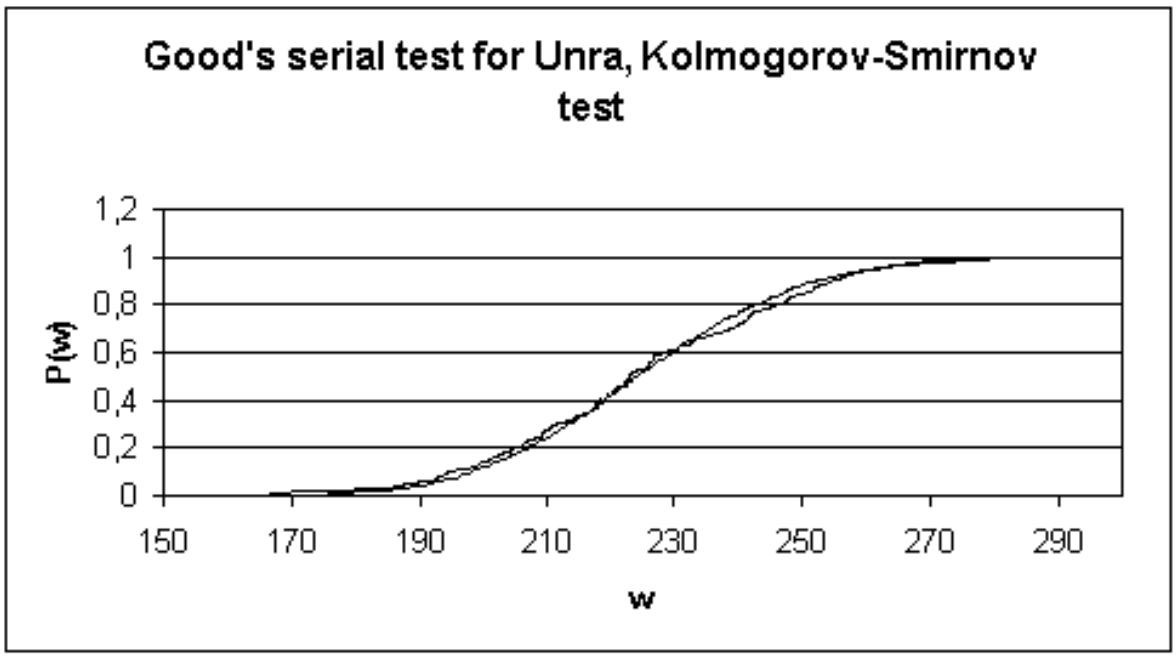


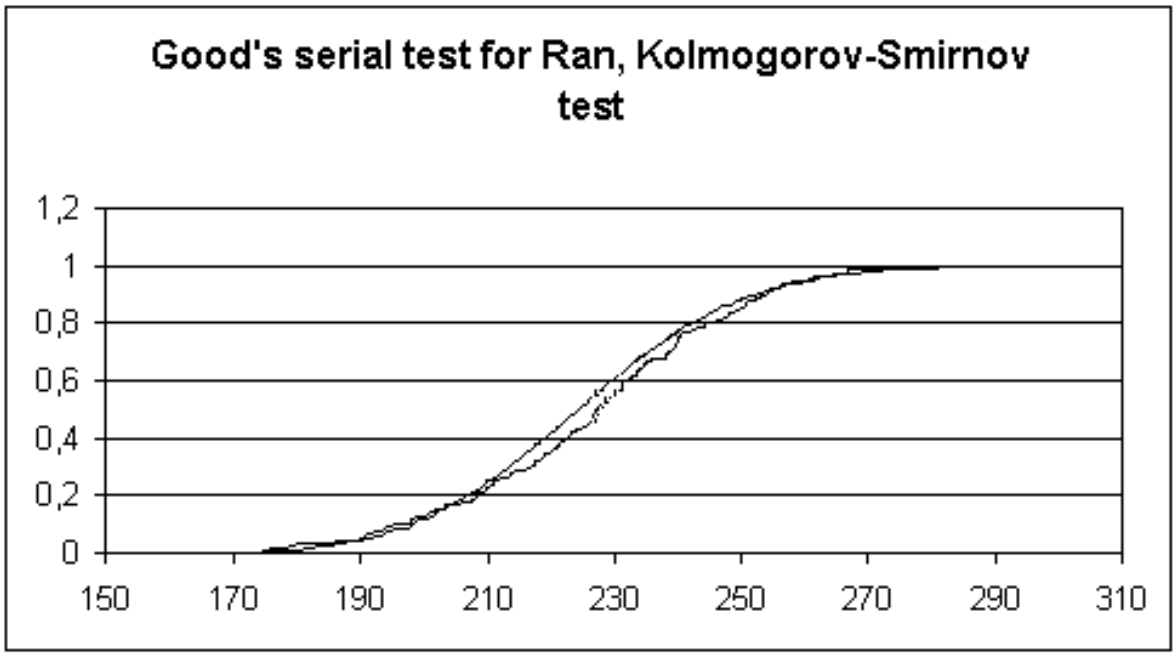


Super duper : normal and cumulative distributions



We formed also the corresponding cumulative distributions and apply Kolmogorov-Smirnov test.





Kolmogorov-Smirnov statistics:

- unra: q=0.814 -> f=25.16%
- ran: q=1.227 -> f=3.50 %
- super duper: q=0.957 f=15.3 %

The f is probability of getting a value larger than q in a sample of size 200 in comparison of cumulative distributions.

8.4.3 Correlations in random number generators: Correlation coeficients

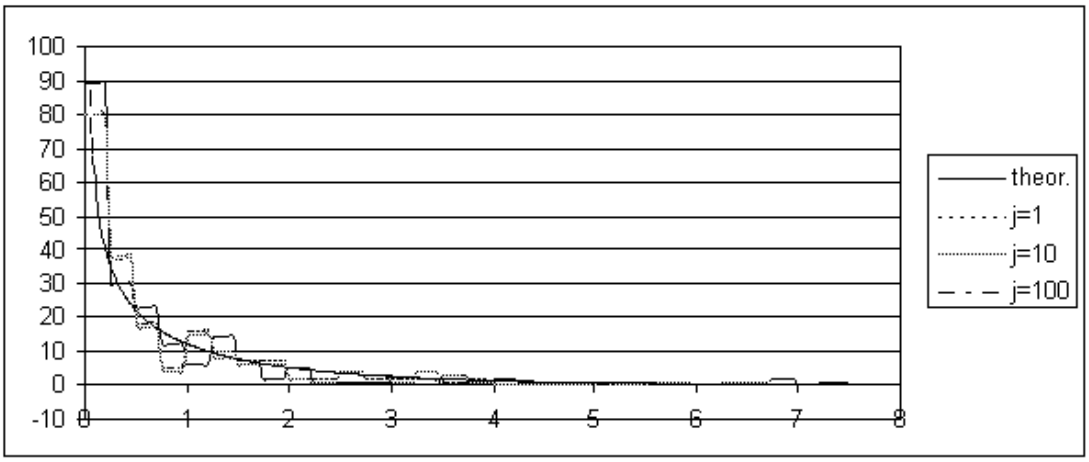
In this test we choose the j separation correlations test for samples of 10000 variations. We take the sample of 10000 random numbers and the estimation of the correlation coeeficients are

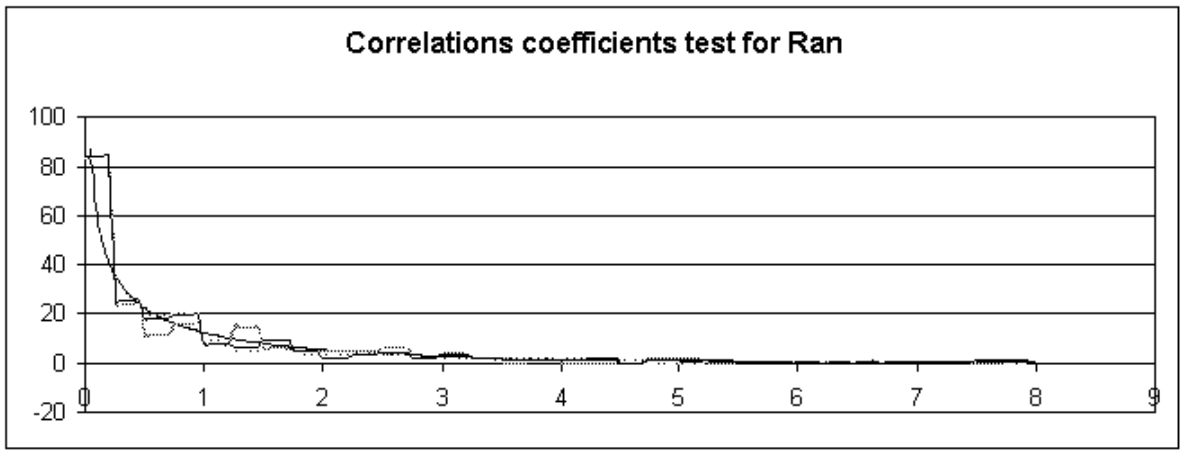
$$r_j = 12(<\xi_i\xi_{i+j} > -1/4) = 12(< (\xi_i - 1/2)(\xi_{i+j} - 1/2) >)$$

for j=1,10 and 100

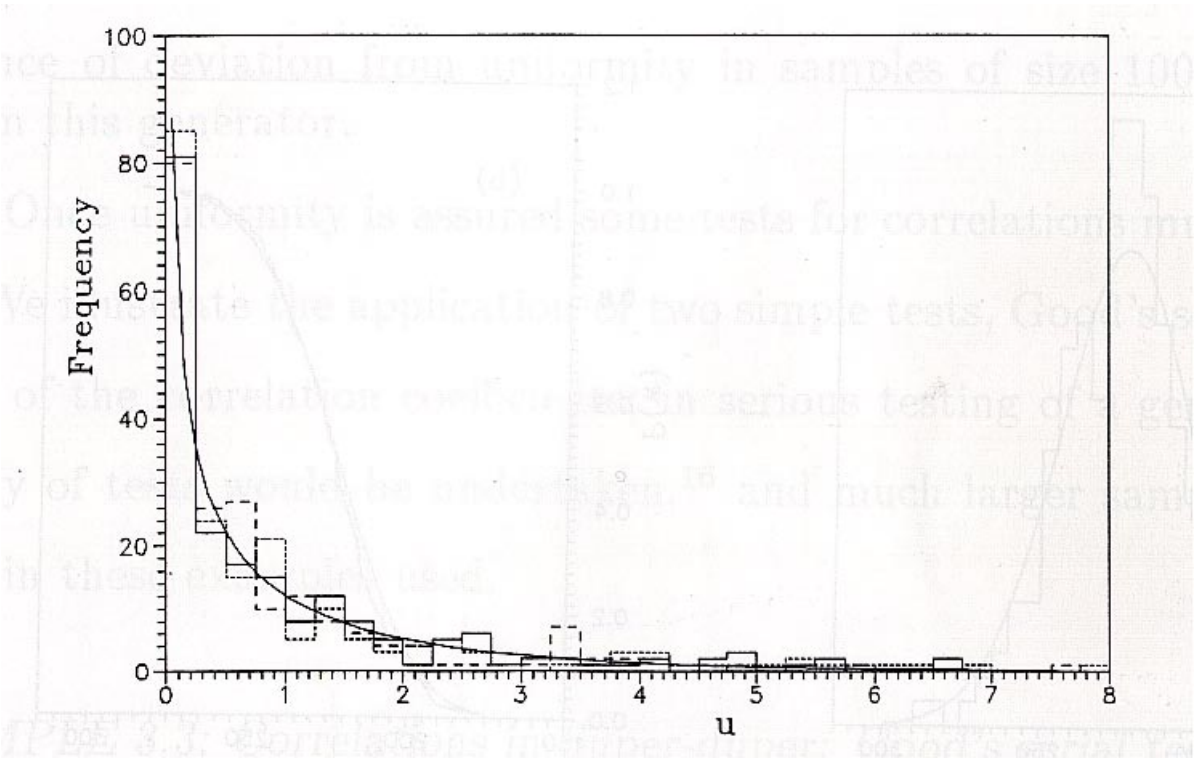
The relevant statistics is u=Nr_j^2 which, independent of j, can be shown to approximately have the chi-square distribution with one degree of freedom. For each of the three values of j, 200 such coefficients were determined.Their distribution is shown in following figures.

Unra:





Super-duper:



We can see there is no evident deviance from theoretical distribution in any of renadom numbers generators

8.4.4 A composite test for correlations

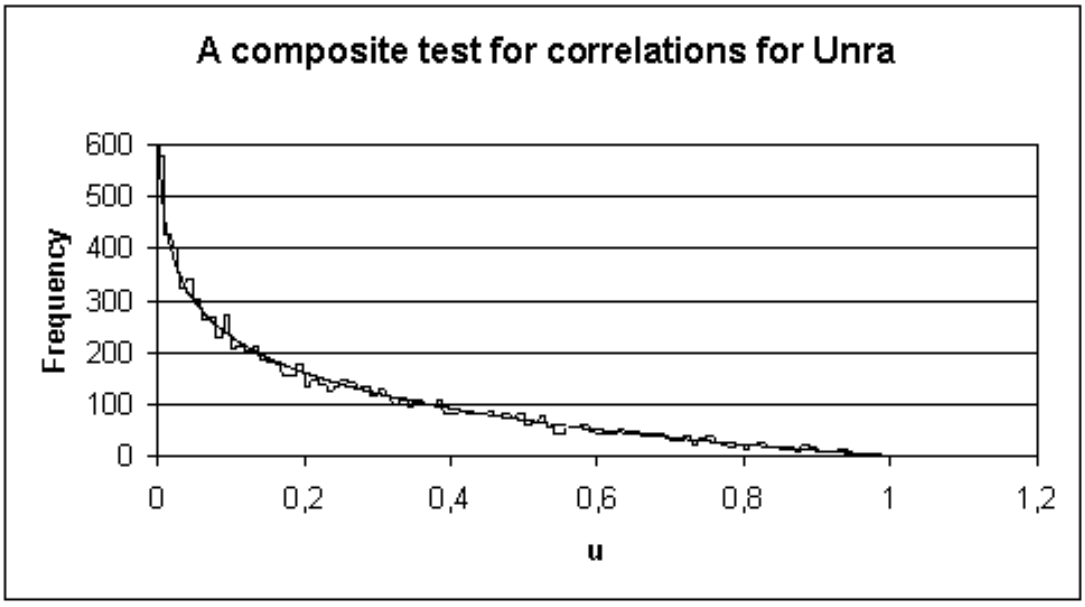
An alternative to a calculation of the correlations of lagged pairs is to combine such pairs or other multiplets, to form variate whose distribution, based on independent variates, is theoretically known. For example, we find that the product of two independent uniform random numbers

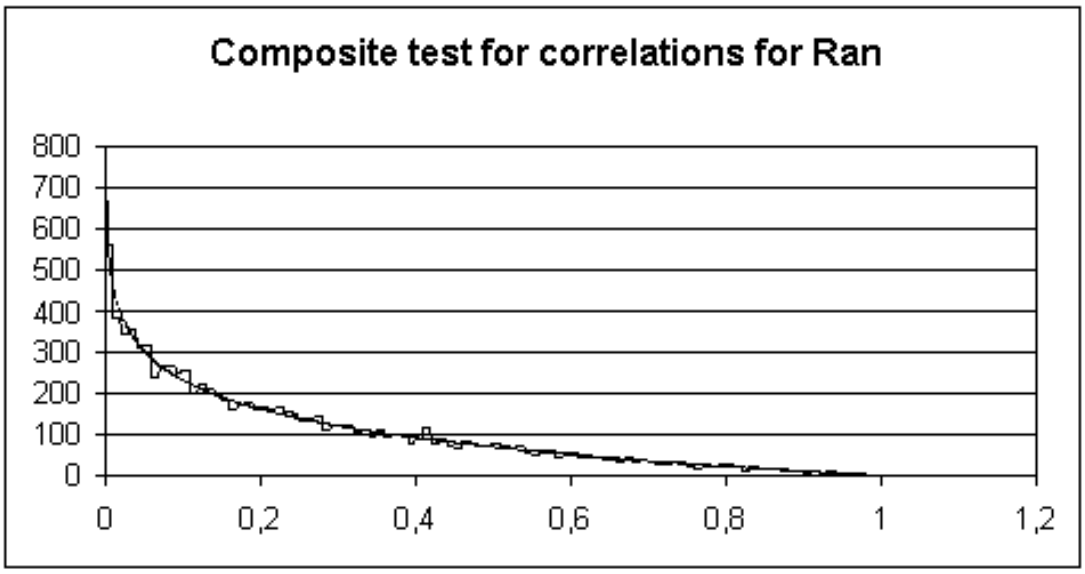
- u=$\xi_i \xi_{i+j}$

should be distributed as

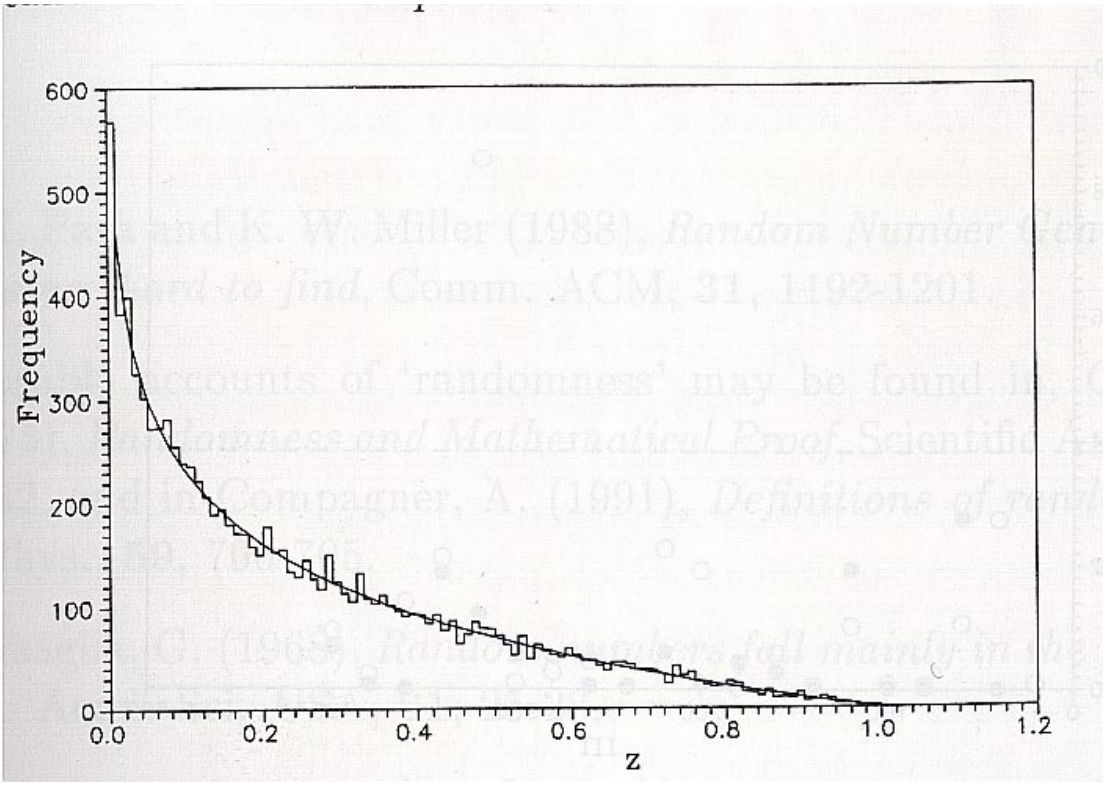
- p(u)=-ln(u) for 0< u $\leq$1.

Taking j=1,10000 non-overlapping pairs from generators were formed, and histogram of 100 bins was formed. Their distribution is compared with its theoretical distribution in following figures




Composite test for correlations for Ran
800
700
600
500
400
300
200
100
0
0
0,2
0,4
0,6
0,8
1
1,2

Super duper:


600
500
400
300
200
100
0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
1.2
z

There is no large deviance observed, however, the chi-square T with 99 degrees of freedom for the

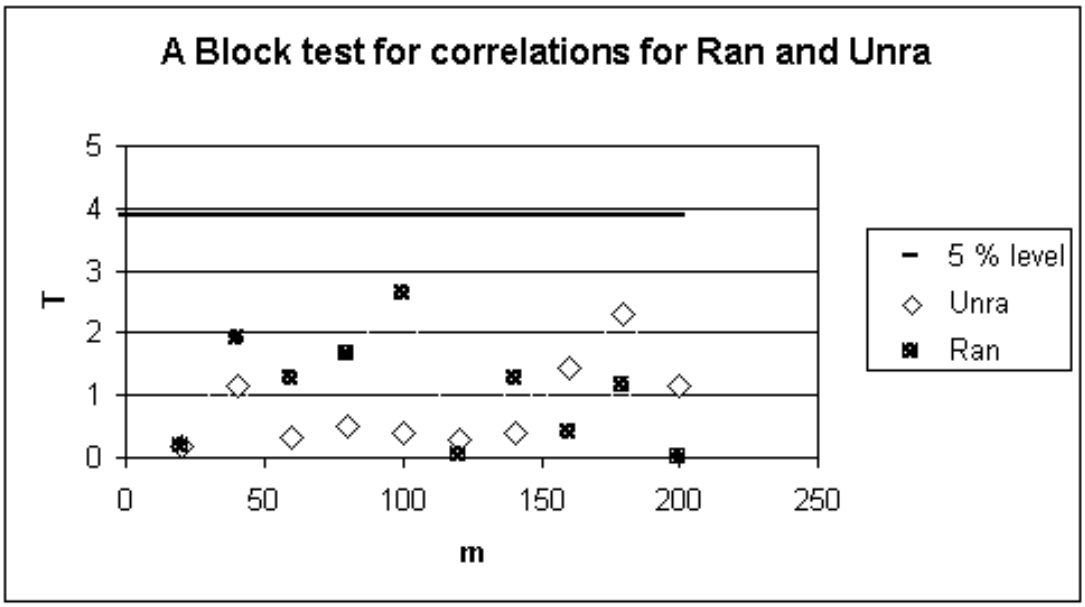
- unra has value T of 103.12 thus the probability that this value or larger would be seen is 36.63 %
- ran : T=84.53 -> 84.97 % probability
- super-duper:T=72.7 -> 98 % confidence level of agreement between distributions.

8.4.5 A block test for correlations

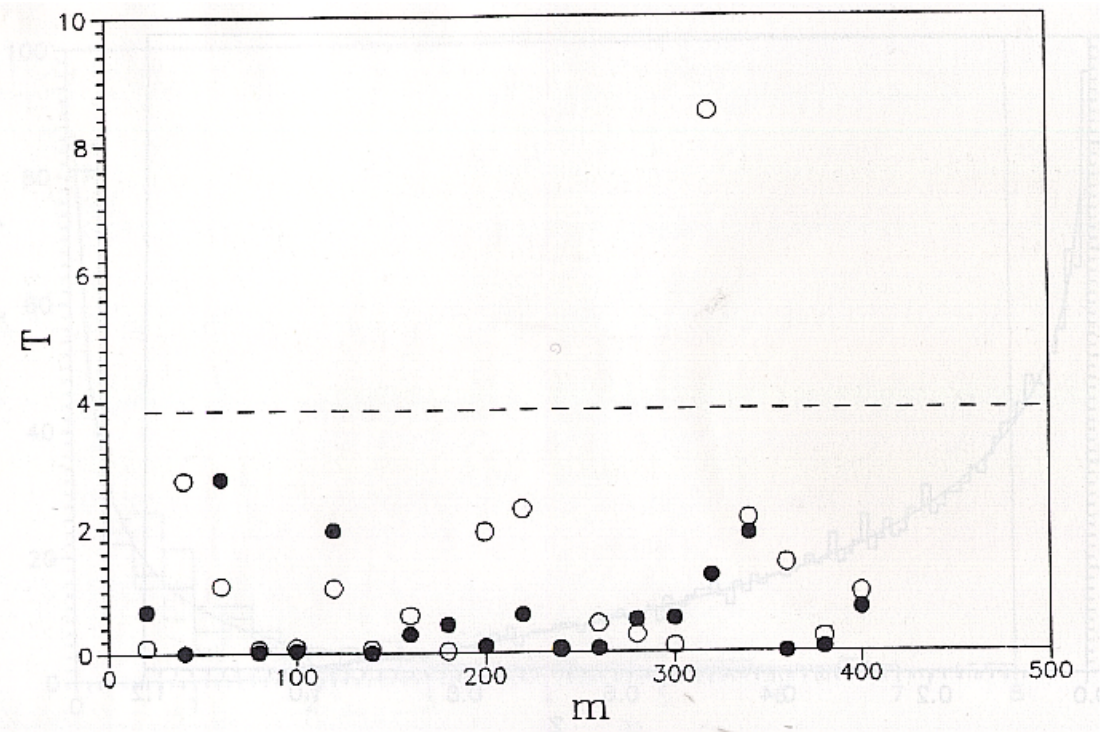
An important source of information on the quality of random number generators is the result obtained when using numbers to simulate a problem, whose solution is known.These algorithms are sensitive to non-randomness, in particular non-uniformity, in successive sequences of a given length. A large number,N, of non-overlapping sequences of the length m, $(\xi_1, \xi_2, ...\xi_m)$, are generated and in each case the mean is found, The number of cases where the mean is less than a half N_-, and the greater than a half, N_+, is tested against expectations using the chi-square statistics, i.e.

$$T=\sum_{i=\mp} \frac{(N_i - N/2)^2}{N/2} = \frac{(2N_+ - N)^2}{N}$$

This quantity should follow the chi-square distribution with one degree of freedom. In following figures see the value obtained for the N=10000samples of sequences of different length in the range m=20 to 200 (for unra and ran) resp. m=20 to 400 (super duperand ranecu used in [*]MacKeown). the horizontal line at 3.841 represents the 5% level for the chi-square test with one degree of freedom. No systematic divergence from expectations is seen in either case, and we can be fairly confident of the quality of the variates produced by these generators.



Super Duper and Ranecu:



8.4.6 Conclusions

There are numerous ways how to test random number generators, we performed only few of them. We let the final judgment on the reader of this chapter, since the author of the tests have no deep knowledge of statistics. However we feel that from these tests all of the generators proved their quality, and it's on the user, which of them will want to use in his/her work. The tests of uniformity showed that Unra generator may have problems with uniform distribution, anyway it seems to be sufficient generator of random number sequences used in computations in Cecile software.

9 References

- Dufresne, J.L, et al "Inverse Gaussian distribution"
- Press, W.H., Teukolsky, S.A. 1992, "Numerical Recipes in Fortran", Cambridge University Press
- MacKeown, P.K. 1997, "Stochastic Simulation in Physics", Springer
- Page, C.G 1995, "Proffesional Programmer's Guide to Fortran 77", Internet Distribution
- NAG Fortran Library Manual Mark 15
- Hottel, H.C., Sarofim, A.F. 1967, "Radiative transfer", McGraw-Hill
- Knuth, D.E.1981, The Art of Computer Programming, Vol.2, Menlo Park