Tina's Random Number Generators Reference Manual

Generated by Doxygen 1.2.15

Wed Feb 19 02:00:02 2003

Contents

1 Tina's Random Number Generators' User Manual

Tina's Random Number Generators (TRNG) is a C^{++} parallel pseudo random number generators package.

1.0.1 Introduction

A lot of tasks in scientific computing can only be tackled by the power of parallel computers. Especially Monte Carlo simmulations are often well suited for parallel computers because of the inherent parallelism of these problems.

A straight forward way to generate pseudo random numbers in a parallel environment is to use the same serial pseudo random number generator on every processor but with (randomly chosen) different seeds or the same generator type but with different parameter sets. These methods are quiet arbitrary and it is not possible to enshure in every case that there are no correlations between the pseudo random number sequences.

To create a good pseudo random number generator for parallel applications we take an excellent sequential pseudo random number generator and distribute its numbers in a *well defined* way over the parallel jobs. There are two different approaches in distributing the numbers.

- **Sequence splitting** The sequential generator's series is split into different contiguous subsequences, which are distributed over the processors. The pseudo random number series r_i of a sequential generator is split into p non overlapping contiguous subsequences with k elements in each subsequence.

$$\begin{aligned} s_{0,i} &= r_i \\ s_{1,i} &= r_{i+k} \\ &\dots \\ s_{p-1,i} &= r_{i+(p-1)k} \end{aligned} \qquad i = 0, 1, \dots, k-1$$

  For a performand implementation it is necessary that r_{i+k} can be easy calculated from r_i even for large k.

- **Leapfrog method** With the leapfrog method the sequential generator's numbers r_i are alternated distributed over the different processors. For p processors we get the new sequences

$$\begin{aligned} t_{0,i} &= r_{pi} \\ t_{1,i} &= r_{pi+1} \\ &\dots \\ t_{p-1,i} &= r_{pi+(p-1)} \,. \end{aligned}$$

Figures 1 and 2 illustrate these different methods.

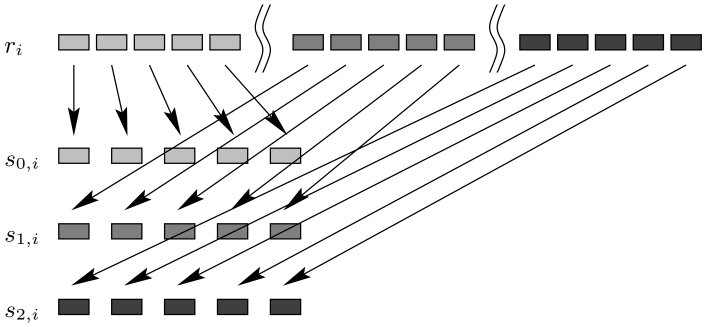


Figure 1: A sequential pseudo random number generator's parallelisation using sequence splitting.

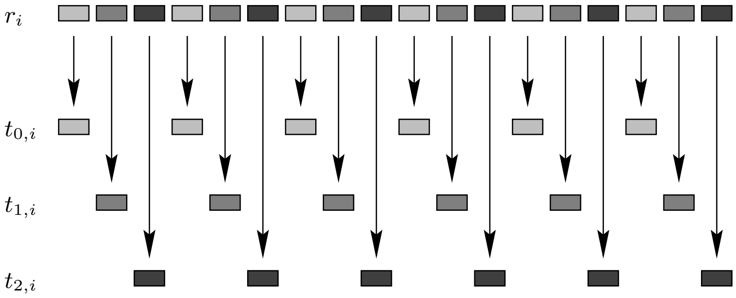


Figure 2: A sequential pseudo random number generator's parallelisation using leapfrog method.

A way to implement the leapfrog method could be to dedicate one processor to random number generation and spread these numbers via a network to the other processors. But this wold be a very inefficient way.

TRNG (Tina's Random Number Generators) is a library that gives you a powerfull tool for Monte Carlo simmulations on parallel computers. TRNG is a collection of random number generators specially designed for the needs of parallel Monte Carlo simmulations. With TRNG you can generate parallel streams of pseudo random numbers (with leapfrog or sequence splitting method) *without* any communication. Tina's Random Number Generators have a highly tuned implementation, have a long period and are empirically tested. Tina's Random Number Generators are easy to use and can be extendend by the user. It is a tool for your everyday work.

1.0.2 Software Distribution and Instalation

Tina's Random Number Generators are copyrighted by Heiko Bauke. You can contact the author of TRNG via electronic mail to `heiko.bauke@physik.uni-magdeburg.de` and get the software from `http://tina.nat.uni-magdeburg.de/TRNG`.

TRNG is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation. This program is distributed *without any* *warranty*; without even the implied warranty of *merchantability* or *fitness for a particular purpose*. See the GNU General Public License for more details.

Download the latest tar file from `http://tina.nat.uni-magdeburg.de/TRNG` and unpack and untar the file with `gzip -c -d trng-2.0.0.tar.gz | tar -xvf -`. Change into the directory `trng-2.0.0` and type `make`. After you have become root type `make install`. That's all!

If you don't have the GNU compiler but the SUN-Workshop compiler use the makefile `Makefile.SW` and type `make -f Makefile.SW` and `make -f Makefile.SW install`.

If you have installed TRNG successfully you can compile programs that use Tina's Random Number Generators, just include the header file `trng.h` and use the linker flag `-ltrng`.

TRNG comes with some example- and test-programs. You may compile these programs with `make examples` (or `make -f Makefile.SW examples`). The binaries are located in the directory `bin`. To compile the MPI-examples you need a MPI-Implementation with C^{++}-support like LAM (`http://www.lam-mpi.org`). Start these programs without commandline arguments to get a short description. The reader should investigate the following programs.

- `plausibility` This program does some tests for leapfrog and sequence splitting.
- `diehard_file` With this program you can generate files for applying the diehard test [7].
- `pi` This MPI program shows how to use TRNG. It computes π by a Monte Carlo method.
- `pi_advanced` This is a version of the program pi where the result is independent of the number processors.
- `exception` This is an example for exception handling in TRNG.
- `template_demo` TRNG is implemeted by using templates. This example shows how hande these templates.
- `copy_demo` is a demo programm that shows the difference between random number assigment and the copy function.

The author hopes that TRNG is a useful tool that fulfils your needs for parallel Monte Carlo simmulations. Please send feedback and bug reports to `heiko.bauke@physik.uni-magdeburg.de`.

1.0.3 An Example

In the following program we compute π by a Monte Carlo simmulation. The program uses MPI.

```
// ***************************************************************
//
// Monte-Carlo-pi-Calulation
//
// ***************************************************************


#include <cstdlib>
#include <cmath>
#include <iostream>
#include <trng.h>
#include <mpi.h>

using namespace TRNG;
using namespace std;

int main(int argc, char *argv[]) {

  // number of points in quare
  const long all_samples=10000001;

  // pseudo random number generator object
  LCG64 r;
  // MPI initialisation
  MPI::Init(argc, argv);

  // get rank and number of processes
  int size=MPI::COMM_WORLD.Get_size();
  int rank=MPI::COMM_WORLD.Get_rank();

  // split sequence of pseudo random numbers by leapfrog method
  r.split(size, rank);

  // no points in the quare
  long in=0l;

  // compute number of points per processor
  long num_samples=all_samples/size;
  // all_samples is not a multiple of size
  if (all_samples%size>rank)
    ++num_samples;

  for (long i=0l; i<num_samples; ++i) {
    double x=r.uniform();
    double y=r.uniform();
    // is point in square
    if (x*x+y*y<=1.0)
      // yes? increment in
      ++in;
  }

  // collect results and print pi
  long in_all;
  MPI::COMM_WORLD.Reduce(&in, &in_all, 1, MPI::LONG, MPI::SUM, 0);
  if (rank==0) {
    double pi=4.0*static_cast<double>(in_all)/static_cast<double>(all_samples);
    cout << "pi = " << pi << endl;
  }

  // quit MPI
  MPI::Finalize();

  return EXIT_SUCCESS;
}
```

The first version of our program gives *not* rusults independent of number of processors. The second version shows how to make the program independent of the number of used processors. The trick is to use different generators to calculate the x- and y-coordinates.

```
// ************************************************************
//
// Monte-Carlo-pi-Calulation
//
// ************************************************************


#include <cstdlib>
#include <cmath>
#include <iostream>
#include <trng.h>
#include <mpi.h>

using namespace TRNG;
using namespace std;

int main(int argc, char *argv[]) {

  // number of points in quare
  const long all_samples=1000000l;

  // pseudo random number generator object
  LCG64 rx;
  LCG64 ry;
  // jump 2^26 steps ahead; 2^26 >> all_samples
  ry.jump2(26);

  // MPI initialisation
  MPI::Init(argc, argv);

  // get rank and number of processes
  int size=MPI::COMM_WORLD.Get_size();
  int rank=MPI::COMM_WORLD.Get_rank();

  // split sequence of pseudo random numbers by leapfrog method
  rx.split(size, rank);
  ry.split(size, rank);

  // no points in the quare
  long in=0l;

  // compute number of points per processor
  long num_samples=all_samples/size;
  // all_samples is not a multiple of size
  if (all_samples%size>rank)
    ++num_samples;

  for (long i=0l; i<num_samples; ++i) {
    double x=rx.uniform();
    double y=ry.uniform();
    // is point in square
    if (x*x+y*y<=1.0)
      // yes? increment in
      ++in;
  }

  // collect results and print pi
  long in_all;
  MPI::COMM_WORLD.Reduce(&in, &in_all, 1, MPI::LONG, MPI::SUM, 0);
  if (rank==0) {
    double pi=4.0*static_cast<double>(in_all)/static_cast<double>(all_samples);
    cout << "pi = " << pi << endl;
  }

  // quit MPI
  MPI::Finalize();

  return EXIT_SUCCESS;
}
```

If a TRNG function is called with an invalid argument this function throws an exception **TRNG::error** (p. 38). The next example shows how to handle these exceptions.

```
#include <cstdlib>
#include <iostream>
#include <stdexcept>
#include <trng.h>

using namespace std;
using namespace TRNG;


int main(void) {
  ParkMiller R;

  cout << R.normal_dist() << endl;
  try {
    R.jump2(16l);
    cout << "jumped forward" << endl;
    // you can't jump backwards
    R.jump2(-16l);
    cout << "jumped backward" << endl;
  }
  catch (exception &e) {
    cerr << "oops!! " << e.what() << endl;
  }
  catch (...) {
    cerr << "something else went wrong" << endl;
  }
  return EXIT_SUCCESS;
}
```

Sometimes it is desired to use a random number generator object as a function argument. Tina's random number genrators are implemented by a template technique. The next program is an example with a function with a random number generator as an argument.

```
#include <cstdlib>
#include <iostream>
#include <trng.h>

using namespace std;
using namespace TRNG;

template<class RNG_type>
void foo(RNG_type &R) {
  cout << R.rand() << " is a random number from generator "
       << R.name() << endl;
}

int main(void) {
  ParkMiller R1;
  LCG64 R2;
  foo(R1);
  foo(R2);
  return EXIT_SUCCESS;
}
```

1.0.4 How to extent TRNG

Youd cant find your favourite Tina's Random Number Generators? No problem! Yust extend TRNG to your needs. In the following example we implement an exclusive or lagged fibonacci generator. This example shows only how TRNG could be extented in principle. Sequence splitting and leapfrog method are implemented in a very stupid way. Unused values are just thrown away. Don't uses this generator in your applications. Doun't use exclusive or lagged fibonacci generator at all, see [2].

```
// ----------------------------------------------------------------------
// Time-stamp: <Tuesday, 10.12.2002, 13:29:06; edited by bauke>
//
// Tina's random number generators TRNG
//
// lagged Fibonacci generator
//
// Copyright (C) 2001, 2002 Heiko Bauke
//
// heiko.bauke@physik.uni-magdeburg.de
//
// TRNG is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation. This program
// is distributed WITHOUT ANY WARRANTY; without even the implied
// warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
// See the GNU General Public License for more details.
//
```

```
// ---------------------------------------------------------------------

#ifndef FIBONACCI_H
#define FIBONACCI_H

#include <trng.h>

// This is an example for extending Tina's random number generators.
// We implement a exclusive or lagged Fibonacci generator.
//
//   r_i=r_(i-q) xor r_(i-p); p>q
//
// Sequence splitting and leapfrog method are implemented in a very stupid
// way. Unused values are just thrown away. Don't use this generator in
// your applications.

// the template parametr determine the lags
template<long p1, long q1>
class Fibonacci : public TRNG::RNG<Fibonacci<p1, q1> > {
private:
  std::vector<long> r;
  long p, q, pointer, steps;
  TRNG::ParkMiller R_init;
public:
  // TRNG::user1_t for a user implemeneted rng
  static const TRNG::RNG_type type=TRNG::user1_t;

  const char * name(void) {
    return "Fibonacci";
  }

  void reset(void) {
    steps=1l;
    max_val=0x7fffffffl;
    max_val2=max_val/2l;
  }

  // this is the initialization procedure described in Fushimi and Tezuka,
  // Comm. ACM, 26 (1983) 516 which ensures that the resulting sequence is
  // P/32-distributed
  void seed(long s) {
    std::vector<unsigned long int> e(p1);
    std::vector<unsigned short int> a(2*p1);
    // construct P linearly independent basis vectors
    for (int i=0; i<p1; i++) {
      int k=31*i/p1;
      e[i]=static_cast<unsigned long int>(s) << k; // zeros to right of bit k
      e[i]=e[i] | ( 1l << k ); // and a one at bit k
    }
    // construct 2P-1 coefficient bits
    for (int i=0; i<p1; i++)
      a[i]=R_init.boolean() ? 1 : 0;
    for (int i=p1; i<2*p1; i++)
      a[i]=a[i-p1]^a[i-q1];
    // construct first P-1 entries (matrix seed) by
    // combining basis vectors according to coefficient bits
    for (int i=0; i<p1; i++) {
      r[i]=0;
      for (int j=0; j<p1; j++)
        if (a[i+j])
          r[i]=r[i]^e[j];
    }
    pointer=p-1l;
  }

  // calculate the next random number
  long rand(void) {
    long t;
    ++pointer;
    if (pointer==p)
```

```
      pointer=0l;
    r[pointer]=r[pointer]^r[(pointer+q)<p ? (pointer+q) : (pointer+q-p)];
    t=r[pointer];
    for (long i=1l; i<steps; ++i) {
      ++pointer;
      if (pointer==p)
        pointer=0l;
      r[pointer]=r[pointer]^r[(pointer+q)<p ? (pointer+q) : (pointer+q-p)];
    }
    return t;
  }

  void split(long s, long n) {
    if (s<1l || n>s || n<0l)
      throw TRNG::error("invalid arguments for Fibonacci::split");
    if (s>1l) {
      for (long i=0l; i<n; ++i)
        rand();
      steps*=s;
    }
  }

  void jump2(long s) {
    if (s<0l || s>63l)
      throw TRNG::error("invalid argument for Fibonacci::split");
    unsigned long long to=1ull<<s;
    for (unsigned long long i=0ull; i<to; ++i)
      rand();
  }

  void save_status(std::vector<long> &s) {
    s.resize(p+5);
    s[0]=type;
    s[1]=q;
    s[2]=p;
    s[3]=pointer;
    s[4]=steps;
    for (int i=0; i<p; ++i)
      s[i+5]=r[i];
  }

  void load_status(const std::vector<long> &s) {
    if (s[0]!=type)
      throw TRNG::error("Fibonacci::load_status wrong parameter");
    q=s[1];
    p=s[2];
    pointer=s[3];
    steps=s[4];
    r.resize(p);
      for (int i=0; i<p; ++i)
        r[i]=s[i+5];
  }

  Fibonacci & Fibonacci::operator=(TRNG::RNG<Fibonacci> &other) {
    if (this!=&other) {
      std::vector<long> s;
      other.save_status(s);
      load_status(s);
    }
    return *this;
  }

  Fibonacci(long seed_=0l) : r() {
    if (p1<0l || q1<0l || p1==q1)
      throw TRNG::error("bad arguments for Fibonacci::Fibonacci");
    if (p1>q1) {
      p=p1;
      q=q1;
    } else {
      p=q1;
      q=p1;
    }
    r.resize(p);
    reset();
    seed(seed_);
  }

  virtual ~Fibonacci() {};
};


#endif
```

2 Tina's Random Number Generators Namespace Index

2.1 Tina's Random Number Generators Namespace List

Here is a list of all documented namespaces with brief descriptions:

3 Tina's Random Number Generators Hierarchical Index

3.1 Tina's Random Number Generators Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

4 Tina's Random Number Generators Compound Index

4.1 Tina's Random Number Generators Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 Tina's Random Number Generators Namespace Documentation

5.1 TRNG Namespace Reference

Tina's random number generators namespace.

Compounds

- struct **vector2d_struct**

  two dimensional vector structure.

- struct **vector3d_struct**
  three dimensional vector structure.
- struct **vector4d_struct**
  four dimensional vector structure.
- class **RNG**
  pseudo random number generator template.
- class **generic_MLCG**
  linear congruential generator.
- class **ParkMiller**
  linear congruential generator.
- class **LCG32**
  linear congruential generator.
- class **LCG64**
  linear congruential generator.
- class **MRG2**
  multiple recursive generator.
- class **MRG3**
  multiple recursive generator.
- class **MRG4**
  multiple recursive generator.
- class **YARNLCG64**
  YARN – a modified linear recursive generator.
- class **YARN2**
  YARN – a modified multiple recursive generator.
- class **YARN3**
  YARN – a modified multiple recursive generator.
- class **YARN4**
  YARN – a modified multiple recursive generator.
- class **CLCG2**
  combined generator.
- class **CLCG3**
  combined generator.
- class **CLCG4**
  combined generator.
- class **EINV**

explicit inversive congruential generator.

- class **EINVLCG64**

  combined generator.

- class **trng_gsl**

  wrapper class for GSL random number generators.

- class **error**

  class for error handling.

- class **power**

  power (p. 45).

Typedefs

- typedef **vector2d_struct vector2d**

  two dimensional vector.

- typedef **vector3d_struct vector3d**

  three dimensional vector.

- typedef **vector4d_struct vector4d**

  four dimensional vector.

Enumerations

- enum **RNG_type** { **RNG_t**, **generic_MLCG_t**, **ParkMiller_t**, **LCG32_t**, **LCG64_t**, **MRG2_t**, **MRG3_t**, **MRG4_t**, **YARNLCG64_t**, **YARN2_t**, **YARN3_t**, **YARN4_t**, **CLCG2_t**, **CLCG3_t**, **CLCG4_t**, **EINV_t**, **EINVLCG64_t**, **trng_gsl_t**, **user1_t** = 1001, **user2_t** = 1002, **user3_t** = 1003, **user4_t** = 1004, **user5_t** = 1005 }

  pseudo random number generator types.

Functions

- const char ∗ **version** (void)

  TRNG version.

- long **modulo_invers** (long, long)

  modulo invers.

- void **gauss** (std::vector< long > &, std::vector< long > &, long)

  linear system solver in modular arithmetic.

- void **matrix_mult** (const std::vector< long > &, const std::vector< long > &, std::vector< long > &, long)

  matrix multiplication.

- void **matrix_vec_mult** (const std::vector< long > &, const std::vector< long > &, std::vector< long > &, long)

  matrix vector multiplication.

- double **Gamma** (double)

  Γ*-function.*

- double **ln_Gamma** (double)

  ln *of* Γ*-function.*

- double **Gamma_P** (double, double)

  incomplete Γ*-function.*

- double **Gamma_Q** (double, double)

  incomplete Γ*-function.*

- double **incomp_Gamma** (double, double)

  incomplete Γ*-function.*

- double **comp_incomp_Gamma** (double, double)

  incomplete Γ*-function.*

- double **Gamma_ser** (double, double)

  incomplete Γ*-function.*

- double **Gamma_cf** (double, double)

  incomplete Γ*-function.*

- double **ln_factorial** (long)

  logarithm of the factorial function.

- long **binomial_coeff** (long, long)

  binomial coefficient.

- double **errf** (double)

  error function.

- double **chi_square_test** (const std::vector< double > &, const std::vector< double > &)

  Chisquare test.

- double **chi_square_prob** (double, long)

  chisquare test.

- double **Stirling_num2** (long, long)

  Stirling number.

- double **Student_t** (double, long, bool=true)

  values for Student's t-distribution.

- long **find_interval** (const std::vector< double > &, const double)

  find interval.

- double **uniform_pdf** (double)

  probalility density.

- double **uniform_pdf** (double, double, double)

  probalility density.

- double **uniformco_pdf** (double)

  probalility density.

- double **uniformco_pdf** (double, double, double)

  probalility density.

- double **uniformcc_pdf** (double)

  probalility density.

- double **uniformcc_pdf** (double, double, double)

  probalility density.

- double **uniformoc_pdf** (double)

  probalility density.

- double **uniformoc_pdf** (double, double, double)

  probalility density.

- double **uniformoo_pdf** (double)

  probalility density.

- double **uniformoo_pdf** (double, double, double)

  probalility density.

- double **normal_dist_pdf** (double, double, double)

  probalility density.

- double **exp_dist_pdf** (double, double)

  probalility density.

- double **laplace_dist_pdf** (double, double)

  probalility density.

- double **tent_dist_pdf** (double, double)

  probalility density.

- double **Gamma_dist_pdf** (double, double, double)

  probalility density.

- double **Beta_dist_pdf** (double, double, double)

  probalility density.

- double **chi_square_dist_pdf** (double, double)

  probalility density.

- double **Student_t_dist_pdf** (double, double) *probalility density.*

- double **binomial_dist_pdf** (long, long, double)
  probalility density.
- double **poisson_dist_pdf** (long, double)
  probalility density.
- double **geometric_dist_pdf** (long, double)
  probalility density.

5.1.1 Detailed Description

All function and classes are encapsulated by the namespace TRNG.

5.1.2 Typedef Documentation

5.1.2.1 typedef struct vector2d_struct TRNG::vector2d

This structure is for storing two dimensional vectors. The method **TRNG::RNG::spherical2d**(void) (p. 59) returns this structure.

Definition at line 83 of file trng.h.

5.1.2.2 typedef struct vector3d_struct TRNG::vector3d

This structure is for storing three dimensional vectors. The method **TRNG::RNG::spherical3d**(void) (p. 59) returns this structure.

Definition at line 100 of file trng.h.

5.1.2.3 typedef struct vector4d_struct TRNG::vector4d

This structure is for storing four dimensional vectors. The method **TRNG::RNG::spherical4d**(void) (p. 59) returns this structure.

Definition at line 119 of file trng.h.

5.1.3 Enumeration Type Documentation

5.1.3.1 enum TRNG::RNG_type

Every pseudo random number generator's type can be identified by a class member type which has anny value from this enumeration type.

Enumeration values:

RNG_t	not specialized random number generator.
generic_MLCG_t	generic multiplicative linear congruential random number generator.
ParkMiller_t	random number generator class **ParkMiller** (p. 44).
LCG32_t	random number generator class **LCG32** (p. 39).
LCG64_t	random number generator class **LCG64** (p. 40).
MRG2_t	random number generator class **MRG2** (p. 41).
MRG3_t	random number generator class **MRG3** (p. 42).

MRG4_t random number generator class **MRG4** (p. 43).
YARNLCG64_t random number generator class **YARNLCG64** (p. 67).
YARN2_t random number generator class **YARN2** (p. 64).
YARN3_t random number generator class **YARN3** (p. 65).
YARN4_t random number generator class **YARN4** (p. 66).
CLCG2_t random number generator class **CLCG2** (p. 32).
CLCG3_t random number generator class **CLCG3** (p. 34).
CLCG4_t random number generator class **CLCG4** (p. 35).
EINV_t random number generator class **EINV** (p. 36).
EINVLCG64_t random number generator class **EINVLCG64** (p. 37).
trng_gsl_t random number generator class **trng_gsl** (p. 61).
user1_t user defined random number generator class nr 1.
user2_t user defined random number generator class nr 2.
user3_t user defined random number generator class nr 3.
user4_t user defined random number generator class nr 4.
user5_t user defined random number generator class nr 5.

Definition at line 44 of file trng.h.

5.1.4 Function Documentation

5.1.4.1 const char ∗ TRNG::version (void)

This function returns a pointer to a zero termiated string with the TRNG version.

Returns:
pointer to a zero termiated string

Definition at line 30 of file trnglib.cc.

5.1.4.2 long TRNG::modulo_invers (long *a*, long *m*)

Solves the equation $a \cdot x = 1 \mod m$.

Returns:
the inverse of a.

Parameters:
a a positive integer.
m a prime modulus.

Exceptions:
error (p. 38) if $a \leq 0$ or $m \leq 1$ or no inversive exists

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 46 of file trnglib.cc.

Referenced by gauss().

5.1.4.3 void TRNG::gauss (std::vector< long > & *a*, std::vector< long > & *b*, long *m*)

Solves a system of linear equations

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \mod m$$

in modular arithmetic using Gau"s elimination.

Parameters:

a reference to the coefficient matrix, content is destroyed after function call

b reference to the inhomogenous right side, contains the solution $(x_1, x_2, \dots, x_n)^T$ after function call

m prime modulus m

Exceptions:

error (p. 38) if coefficient matrix is singular or the matrices have invalid sizes

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 94 of file trnglib.cc.

References modulo_invers().

5.1.4.4 void TRNG::matrix_mult (const std::vector< long > & *a*, const std::vector< long > & *b*, std::vector< long > & *c*, long *m*)

Multiply two equal sized quadratic matrices a and b in modular arithmetic, $c = a \cdot b \mod m$.

Parameters:

a reference to matrix a

b reference to matrix b

c reference to matrix c

m modulus m

Exceptions:

error (p. 38) if the matrices are different sized

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 183 of file trnglib.cc.

5.1.4.5 void TRNG::matrix_vec_mult (const std::vector< long > & *a*, const std::vector< long > & *b*, std::vector< long > & *c*, long *m*)

Multiply a quadratic matrix a and \$a\$ vector b in modular arithmetic, $c = a \cdot b \mod m$.

Parameters:

a reference to matrix a

b reference to vector b

c reference to vector c

m modulus m

Exceptions:

error (p. 38) if the matrices are different sized

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 217 of file trnglib.cc.

5.1.4.6 double TRNG::Gamma (double *x*)

Computes the Γ-function $\Gamma(x) = \int\limits_0^\infty t^{x-1}\mathrm{e}^{-t}\,\mathrm{d}t$ for positive arguments.

Parameters:

x argument

Exceptions:

error (p. 38) if $x \le 0$

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 291 of file trnglib.cc.

5.1.4.7 double TRNG::ln_Gamma (double *x*)

Computes the Γ-function's logarithm $\ln\Gamma(x) = \ln\int\limits_0^\infty t^{x-1}\mathrm{e}^{-t}\,\mathrm{d}t$ for positive arguments. Method is discribed in [5].

Parameters:

x argument

Exceptions:

error (p. 38) if $x \le 0$

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 254 of file trnglib.cc.

5.1.4.8 double TRNG::Gamma_P (double *a*, double *x*)

Computes the incomplete Γ-function $P(a,x)=\frac{1}{\Gamma(a)}\int\limits_0^x t^{a-1}\mathrm{e}^{-t}\,\mathrm{d}t$ for positive arguments.

Parameters:

a argument a

x argument x

Exceptions:

error (p. 38) if $x<0$ or $a\leq 0$

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 353 of file trnglib.cc.

5.1.4.9 double TRNG::Gamma_Q (double *a*, double *x*)

Computes the incomplete Γ-function $Q(a,x)=\frac{1}{\Gamma(a)}\int\limits_x^\infty t^{a-1}\mathrm{e}^{-t}\,\mathrm{d}t$ for positive arguments.

Parameters:

a argument a

x argument x

Exceptions:

error (p. 38) if $x<0$ or $a\leq 0$

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 377 of file trnglib.cc.

5.1.4.10 double TRNG::incomp_Gamma (double *a*, double *x*)

Computes the incomplete Γ-function $\gamma(a,x)=\int\limits_0^x t^{a-1}\mathrm{e}^{-t}\,\mathrm{d}t$ for positive arguments.

Parameters:

a argument a

x argument x

Exceptions:
error (p. 38) if $x < 0$ or $a \leq 0$

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 401 of file trnglib.cc.

5.1.4.11 double TRNG::comp_incomp_Gamma (double *a*, double *x*)

Computes the complementary incomplete Γ-function $\Gamma(a,x)=\int\limits_x^\infty t^{a-1}\mathrm{e}^{-t}\,\mathrm{d}t$ for positive arguments.

Parameters:
a argument a

x argument x

Exceptions:
error (p. 38) if $x < 0$ or $a \leq 0$

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 423 of file trnglib.cc.

5.1.4.12 double TRNG::Gamma_ser (double *a*, double *x*)

Computes incomplete Gamma function's ($P(a,x)$) **power** (p. 45) series representation for $x \leq a+1$.

Parameters:
a argument a

x argument x

Exceptions:
error (p. 38) if $x < 0$ or $a \leq 0$ or at convergence problems

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 444 of file trnglib.cc.

5.1.4.13 double TRNG::Gamma_cf (double *a*, double *x*)

Computes incomplete Gamma function's ($Q(a, x)$) continued fraction representation for $x \geq a + 1$.

Parameters:

a argument a

x argument x

Exceptions:

error (p. 38) if $x < 0$ or $a \leq 0$ or at convergence problems

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 482 of file trnglib.cc.

5.1.4.14 double TRNG::ln_factorial (long *n*)

Returns:

$\ln(n!)$

Parameters:

n n

Exceptions:

error (p. 38) if $n < 0$

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 529 of file trnglib.cc.

5.1.4.15 long TRNG::binomial_coeff (long *n*, long *k*)

Returns:

binomial coefficient $\binom{n}{k}$, if $n < 0$ or $k < 0$ or $k > n$ 0 is returned

Parameters:

n n

k k

Author:

Heiko Bauke

Definition at line 555 of file trnglib.cc.

5.1.4.16 double TRNG::errf (double *x*)

Computes the error function

$$errf(x) = \frac{1}{\sqrt{2\pi}} \int\limits_{-\infty}^{x} \mathrm{e}^{-\frac{t^2}{2}} \, \mathrm{d}t \, .$$

Returns:
$errf(x)$

Author:
Heiko Bauke

Definition at line 575 of file trnglib.cc.

5.1.4.17 double TRNG::chi_square_test (const std::vector< double > & *prob*, const std::vector< double > & *observ*)

Applies a χ^2-test.

Returns:
χ^2-value

Parameters:
prob reference to a vector with some probabilities

observ reference to a vector with numbers of actual observations

Exceptions:
error (p. 38) if arguments are not equal sized or number of observations is less then five

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 594 of file trnglib.cc.

5.1.4.18 double TRNG::chi_square_prob (double *chi2*, long *df*)

Computes the probability corresponding to a χ^2-value. See [3] page 41 for details.

Parameters:
chi2 χ^2-value

df degrees of freedom

Exceptions:
error (p. 38) if degrees of freedom less than one

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 626 of file trnglib.cc.

5.1.4.19 double TRNG::Stirling_num2 (long *n*, long *m*)

Computes the Stirling number of the 2nd kind $\left\{ {n \atop m} \right\}$, see also [4] pp. 65ff.

Returns:

Stirling number of the 2nd kind $\left\{ {n \atop m} \right\}$

Parameters:

n 1st parameter

m 2nd parameter

Author:

Heiko Bauke

Definition at line 651 of file trnglib.cc.

5.1.4.20 double TRNG::Student_t (double *p*, long *nu*, bool *symmetric* = true)

Computes $t_{p,\nu}$ of the t-distribution. $t_{p,\nu}$ is defined in the symmetric case as

$$p = \frac{\Gamma((\nu+1)/2)}{\sqrt{\pi\nu}\,\Gamma(\nu/2)} \int_{-t_{p,\nu}}^{t_{p,\nu}} (1+x^2/\nu)^{\frac{\nu+1}{2}}\,\mathrm{d}x$$

and in the asymmetric case as

$$p = \frac{\Gamma((\nu+1)/2)}{\sqrt{\pi\nu}\,\Gamma(\nu/2)} \int_{-\infty}^{t_{p,\nu}} (1+x^2/\nu)^{\frac{\nu+1}{2}}\,\mathrm{d}x\,.$$

Parameters:

p probability p

nu degrees of freedom ν

symmetric is true for the symmetric case

Exceptions:

error (p. 38) if less than one degree of freedom or probability out of range $0 < p < 1$

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 691 of file trnglib.cc.

5.1.4.21 long TRNG::find_interval (const std::vector< double > & *borders*, const *double*)

This function searches for the interval which contains x using an $O(\ln n)$ algorithm. The intervals are

$$I_0 = (-\infty, b_0]\,, \quad I_1 = (-b_0, b_1]\,, \quad I_2 = (-b_1, b_2]\,, \quad \ldots \quad I_{n-1} = (-b_{n-2}, \infty)$$

if b is a sorted vector with elements b_i with $0 \leq i \leq n-2$.

Returns:

interval number

Parameters:

borders reference to a sorted vector b_i with interval edges

x value that determs the interval

Author:

Heiko Bauke

Definition at line 762 of file trnglib.cc.

Referenced by TRNG::RNG< LCG32 >::binomial_dist_tab().

5.1.4.22 double TRNG::uniform_pdf (double *x*)

Calculates the probalility density function for in $[0, 1)$ equidistributed random variate.

Returns:

1 if $x \in [0, 1)$ else 0

Author:

Heiko Bauke

Definition at line 789 of file trnglib.cc.

References uniformco_pdf().

5.1.4.23 double TRNG::uniform_pdf (double *x*, double *a*, double *b*)

Calculates the probalility density function for in $[a, b)$ equidistributed random variate.

Returns:

$\frac{1}{b-a}$ if $x \in [a, b)$ else 0

Author:

Heiko Bauke

Definition at line 800 of file trnglib.cc.

References uniformco_pdf().

5.1.4.24 double TRNG::uniformco_pdf (double *x*)

Calculates the probalility density function for in $[0, 1)$ equidistributed random variate.

Returns:

1 if $x \in [0, 1)$ else 0

Author:

Heiko Bauke

Definition at line 813 of file trnglib.cc.

Referenced by uniform_pdf().

5.1.4.25 double TRNG::uniformco_pdf (double x, double a, double b)

Calculates the probalility density function for in $[a, b)$ equidistributed random variate.

Returns:
$\frac{1}{b-a}$ if $x \in [a, b)$ else 0

Author:
Heiko Bauke

Definition at line 827 of file trnglib.cc.

5.1.4.26 double TRNG::uniformcc_pdf (double x)

Calculates the probalility density function for in $[0, 1]$ equidistributed random variate.

Returns:
1 if $x \in [0, 1]$ else 0

Author:
Heiko Bauke

Definition at line 843 of file trnglib.cc.

5.1.4.27 double TRNG::uniformcc_pdf (double x, double a, double b)

Calculates the probalility density function for in $[a, b]$ equidistributed random variate.

Returns:
$\frac{1}{b-a}$ if $x \in [a, b]$ else 0

Author:
Heiko Bauke

Definition at line 858 of file trnglib.cc.

5.1.4.28 double TRNG::uniformoc_pdf (double x)

Calculates the probalility density function for in $(0, 1]$ equidistributed random variate.

Returns:
1 if $x \in (0, 1]$ else 0

Author:
Heiko Bauke

Definition at line 874 of file trnglib.cc.

5.1.4.29 double TRNG::uniformoc_pdf (double x, double a, double b)

Calculates the probalility density function for in $(a, b]$ equidistributed random variate.

Returns:
$\frac{1}{b-a}$ if $x \in (a, b]$ else 0

Author:
Heiko Bauke

Definition at line 888 of file trnglib.cc.

5.1.4.30 double TRNG::uniformoo_pdf (double *x*)

Calculates the probalility density function for in $(0, 1)$ equidistributed random variate.

Returns:
1 if $x \in (0, 1)$ else 0

Author:
Heiko Bauke

Definition at line 904 of file trnglib.cc.

5.1.4.31 double TRNG::uniformoo_pdf (double *x*, double *a*, double *b*)

Calculates the probalility density function for in (a, b) equidistributed random variate.

Returns:
$\frac{1}{b-a}$ if $x \in (a, b)$ else 0

Author:
Heiko Bauke

Definition at line 918 of file trnglib.cc.

5.1.4.32 double TRNG::normal_dist_pdf (double *x*, double *sigma* = 1.0, double *mu* = 0.0)

This function calculates the probalility density for a random variate with normal distribution. This distribution is defined as

$$p(x) = \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

with mean μ and variance σ.

Returns:
$p(x)$

Parameters:

x x

sigma variance σ

mu mean μ

Exceptions:
error (p. 38) if $\sigma \leq 0$

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 943 of file trnglib.cc.

5.1.4.33 double TRNG::exp_dist_pdf (double *x*, double *mu* = 1.0)

This function calculates the probalility density for a random variate with exponential distribution. This distribution is defined as

$$p(x) = \begin{cases} \frac{1}{\mu}\mathrm{e}^{-\frac{x}{\mu}} & x \geq 0 \\ 0 & \text{else} \end{cases}$$

with $\mu > 0$.

Returns:

$p(x)$

Parameters:

x x

mu mean μ, $\mu > 0$

Exceptions:

error (p. 38) if $\mu \leq 0$

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 971 of file trnglib.cc.

5.1.4.34 double TRNG::laplace_dist_pdf (double *x*, double *a* = 1.0)

This function calculates the probalility density for a random variate with Laplace distribution. This distribution is defined as

$$p(x) = \frac{1}{2a}\mathrm{e}^{-|x|/a}$$

with $a > 0$.

Returns:

$p(x)$

Parameters:

x x

a parameter a, $a > 0$

Exceptions:

error (p. 38) if $a \leq 0$

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 997 of file trnglib.cc.

5.1.4.35 double TRNG::tent_dist_pdf (double *x*, double *a* = 1.0)

The tent shaped probability distribution is defined as

$$p(x) = \begin{cases} \dfrac{x+a}{a^2} & -a \le x \le 0 \\ \dfrac{a-x}{a^2} & 0 \le x \le a \\ 0 & \text{else} \end{cases}$$

with $a > 0$.

Returns:
$p(x)$

Parameters:

x x

a parameter a

Exceptions:
error (p. 38) if $a \le 0$

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 1024 of file trnglib.cc.

5.1.4.36 double TRNG::Gamma_dist_pdf (double *x*, double *a*, double *b*)

This function calculates the probalility density for a random variate with Γ-distribution. This distribution is defined as

$$p(x) = \begin{cases} \dfrac{1}{\Gamma(a)b^a} x^{a-1} \mathrm{e}^{-\frac{x}{b}} & x > 0 \\ 0 & \text{else} \end{cases} .$$

Returns:
$p(x)$

Parameters:

x x

a parameter a, $a > 0$

b parameter b, $b > 0$

Exceptions:
error (p. 38) if $a \le 0$ or $b \le 0$

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 1055 of file trnglib.cc.

5.1.4.37 double TRNG::Beta_dist_pdf (double *x*, double *a*, double *b*)

This function calculates the probalility density for a random variate with B-distribution. This distribution is defined as

$$p(x) = \begin{cases} \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1} & 0 \le x \le 1 \\ 0 & \text{else} \end{cases} .$$

Returns:

$p(x)$

Parameters:

x x

a parameter a, $a > 0$

b parameter b, $b > 0$

Exceptions:

error (p. 38) if $a \le 0$ or $b \le 0$

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 1092 of file trnglib.cc.

5.1.4.38 double TRNG::chi_square_dist_pdf (double *x*, double *nu*)

This function calculates the probalility density for a random variate with χ^2-distribution. This distribution is defined as

$$p(x) = \begin{cases} \frac{\left(\frac{x}{2}\right)^{\frac{\nu}{2}-1}}{2\Gamma(\frac{\nu}{2})} e^{-\frac{x}{2}} & x \ge 0 \\ 0 & \text{else} \end{cases} .$$

Returns:

$p(x)$

Parameters:

x x

nu parameter ν, $\nu \ge 1$

Exceptions:

error (p. 38) if $\nu < 1$

See also:

TRNG::error (p. 38)

Author:

Heiko Bauke

Definition at line 1124 of file trnglib.cc.

5.1.4.39 double TRNG::Student_t_dist_pdf (double *x*, double *nu*)

This function calculates the probalility density for a random variate with Student's t distribution. This distribution is defined as

$$p(x) = \frac{\Gamma\left(\frac{\nu-1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}} .$$

Returns:
$p(x)$

Parameters:

x x

nu parameter ν, $\nu \geq 1$

Exceptions:
error (p. 38) if $\nu \leq 0$

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 1151 of file trnglib.cc.

5.1.4.40 double TRNG::binomial_dist_pdf (long *k*, long *n*, double *p* = 0.5)

This function calculates the probalility density for a random variate with binomial distribution. This distribution is defined as

$$p(k) = \frac{n!}{k!(n-k)!}p^k(1-p)^{n-k}, \qquad 0 \leq k \leq n .$$

Returns:
$p(k)$

Parameters:

k parameter k

n number of trails n

p probability p in each trail

Exceptions:
error (p. 38) if $p \leq 0$ or $p > 1$ or $n \leq 0$

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 1177 of file trnglib.cc.

5.1.4.41 double TRNG::poisson_dist_pdf (long *k*, double *mu*)

This function calculates the probalility density for a random variate with poisson distribution. This distribution is defined as

$$p(k) = \begin{cases} \frac{\mu^k}{k!}\mathrm{e}^{-\mu} & k \geq 0 \\ 0 & \text{else} \end{cases}$$

with mean $\mu \geq 0$.

Returns:
$p(k)$

Parameters:
k parameter k

mu mean μ

Author:
Heiko Bauke

Definition at line 1208 of file trnglib.cc.

5.1.4.42 double TRNG::geometric_dist_pdf (long *k*, double *q*)

This function calculates the probalility density for a random variate with geometric distribution. This distribution is defined as

$$p(k) = q(1-q)^{k-1}\,, \qquad k \geq 1\,.$$

Returns:
$p(k)$

Parameters:
q probability q

k parameter k

Exceptions:
error (p. 38) if $p \leq 0$ or $p > 1$ or $n \leq 0$

See also:
TRNG::error (p. 38)

Author:
Heiko Bauke

Definition at line 1231 of file trnglib.cc.

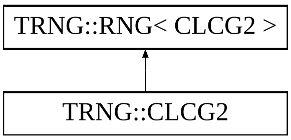
6 Tina's Random Number Generators Class Documentation

6.1 TRNG::CLCG2 Class Reference

combined generator.

`#include <trng.h>`

Inheritance diagram for TRNG::CLCG2::



Public Methods

- **CLCG2** (long a1=376555083l, long m1=2147482951l, long a2=1028879659l, long m2=2147482949l, long seed_=0l)

  constructor.

6.1.1 Detailed Description

This is a combined linear congruential random number generator with two generators.

$$\begin{aligned} q_{1,i} &= a_1 \cdot q_{1,i-1} \mod m_1 \\ q_{2,i} &= a_2 \cdot q_{2,i-1} \mod m_2 \\ r_i &= q_{1,i} + q_{2,i} \mod m_1 - 1 \end{aligned}$$

See also [6].

Author:

Heiko Bauke

Definition at line 1659 of file trng.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 TRNG::CLCG2::CLCG2 (long *a1* = 376555083l, long *m1* = 2147482951l, long *a2* = 1028879659l, long *m2* = 2147482949l, long *seed_* = 0l)

The constructor's default values implement a pseudo random number generator with

$$\begin{aligned} q_{1,i} &= 376\,555\,083 \cdot q_{1,i-1} \mod 2\,147\,482\,951 \\ q_{2,i} &= 1\,028\,879\,659 \cdot q_{2,i-1} \mod 2\,147\,482\,949 \\ r_i &= q_{1,i} + q_{2,i} \mod 2\,147\,482\,950\,. \end{aligned}$$

This generator has a period of $\frac{(2\,147\,482\,951-1)(2\,147\,482\,949-1)}{2} \approx 2^{61} \approx 2.31 \cdot 10^{18}$. The multipliers were found by an exhausive search applying the spectral test in up to eight dimensions.

Parameters:

a1 multiplier a_1

m1 prime modulus m_1

a2 multiplier a_2

m2 prime modulus m_2

seed_ default seed

The documentation for this class was generated from the following file:

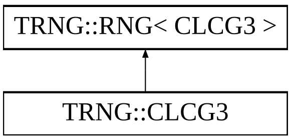
- **trng.h**

6.2 TRNG::CLCG3 Class Reference

combined generator.

`#include <trng.h>`

Inheritance diagram for TRNG::CLCG3::



Public Methods

- **CLCG3** (long a1=376555083l, long m1=2147482951l, long a2=1028879659l, long m2=2147482949l, long a3=225802979l, long m3=2147482943l, long seed_=0l)

  constructor.

6.2.1 Detailed Description

This is a combined linear congruential random number generator with three generators.

$$
\begin{aligned}
q_{1,i} &= a_1 \cdot q_{1,i-1} \mod m_1 \\
q_{2,i} &= a_2 \cdot q_{2,i-1} \mod m_2 \\
q_{3,i} &= a_3 \cdot q_{3,i-1} \mod m_3 \\
r_i &= q_{1,i} + q_{2,i} + q_{3,i} \mod m_1 - 1
\end{aligned}
$$

See also [6].

Author:
Heiko Bauke

Definition at line 1724 of file trng.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 TRNG::CLCG3::CLCG3 (long *a1* = 376555083l, long *m1* = 2147482951l, long *a2* = 1028879659l, long *m2* = 2147482949l, long *a3* = 225802979l, long *m3* = 2147482943l, long *seed_* = 0l)

The constructor's default values implement a pseudo random number generator with

$$
\begin{aligned}
q_{1,i} &= 376\,555\,083 \cdot q_{1,i-1} \mod 2\,147\,482\,951 \\
q_{2,i} &= 1\,028\,879\,659 \cdot q_{2,i-1} \mod 2\,147\,482\,949 \\
q_{3,i} &= 225\,802\,979 \cdot q_{3,i-1} \mod 2\,147\,482\,943 \\
r_i &= q_{1,i} + q_{2,i} + q_{3,i} \mod 2\,147\,482\,950\,.
\end{aligned}
$$

This generator has a period of $\frac{(2\,147\,482\,951-1)(2\,147\,482\,949-1)(2\,147\,482\,943-1)}{4} \approx 2^{91} \approx 2.48 \cdot 10^{27}$. The multipliers were found by an exhausive search applying the spectral test in up to eight dimensions.

Parameters:
a1 multiplier a_1

m1 prime modulus m_1

a2 multiplier a_2

m2 prime modulus m_2

a3 multiplier a_3

m3 prime modulus m_3

seed_ default seed

The documentation for this class was generated from the following file:

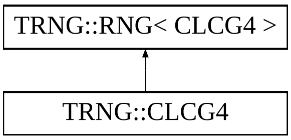
- **trng.h**

6.3 TRNG::CLCG4 Class Reference

combined generator.

`#include <trng.h>`

Inheritance diagram for TRNG::CLCG4::



Public Methods

- **CLCG4** (long a1=376555083l, long m1=2147482951l, long a2=1028879659l, long m2=2147482949l, long a3=225802979l, long m3=2147482943l, long a4=2028073966l, long m4=2147482859l, long seed_=0l)

  constructor.

6.3.1 Detailed Description

This is a combined linear congruential random number generator with four generators.

$$\begin{aligned} q_{1,i} &= a_1 \cdot q_{1,i-1} \mod m_1 \\ q_{2,i} &= a_2 \cdot q_{2,i-1} \mod m_2 \\ q_{3,i} &= a_3 \cdot q_{3,i-1} \mod m_3 \\ q_{4,i} &= a_3 \cdot q_{4,i-1} \mod m_4 \\ r_i &= q_{1,i} + q_{2,i} + q_{3,i} + q_{4,i} \mod m_1 - 1 \end{aligned}$$

See also [6].

Author:

Heiko Bauke

Definition at line 1797 of file trng.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 TRNG::CLCG4::CLCG4 (long *a1* = 376555083l, long *m1* = 2147482951l, long *a2* = 1028879659l, long *m2* = 2147482949l, long *a3* = 225802979l, long *m3* = 2147482943l, long *a4* = 2028073966l, long *m4* = 2147482859l, long *seed_* = 0l)

The constructor's default values implement a pseudo random number generator with

$$\begin{aligned} q_{1,i} &= 376\,555\,083 \cdot q_{1,i-1} \mod 2\,147\,482\,951 \\ q_{2,i} &= 1\,028\,879\,659 \cdot q_{2,i-1} \mod 2\,147\,482\,949 \\ q_{3,i} &= 225\,802\,979 \cdot q_{3,i-1} \mod 2\,147\,482\,943 \\ q_{4,i} &= 2\,028\,073\,966 \cdot q_{3,i-1} \mod 2\,147\,482\,859 \\ r_i &= q_{1,i} + q_{2,i} + q_{3,i} + q_{4,i} \mod 2\,147\,482\,950\,. \end{aligned}$$

This generator has a period of $\frac{(2\,147\,482\,951-1)(2\,147\,482\,949-1)(2\,147\,482\,943-1)(2\,147\,482\,859-1)}{8} \approx 2^{121} \approx 2.66 \cdot 10^{36}$. The multipliers were found by an exhaustive search applying the spectral test in up to eight dimensions.

Parameters:

a1 multiplier a_1

m1 prime modulus m_1

a2 multiplier a_2

m2 prime modulus m_2

a3 multiplier a_3

m3 prime modulus m_3

a4 multiplier a_4

m4 prime modulus m_4

seed_ default seed

The documentation for this class was generated from the following file:

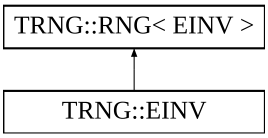
- **trng.h**

6.4 TRNG::EINV Class Reference

explicit inversive congruential generator.

`#include <trng.h>`

Inheritance diagram for TRNG::EINV::



Public Methods

- **EINV** (long a_=1073741831l, long seed_=0l)
  constructor.

6.4.1 Detailed Description

This is an explicit inversive congruential generator.

$$r_i = \overline{a \cdot i} \mod (2^{30} + 2^{28} + 3)$$

This generator has a period of $2^{30} + 2^{28} + 3 \approx 2^{30} \approx 1.34 \cdot 10^9$. This type has excelent statistical properties but its period is too short for large applications. You may combine this genrator with an other one. See also [1].

Author:
Heiko Bauke

Definition at line 1875 of file trng.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 TRNG::EINV::EINV (long *a_* = 1073741831l, long *seed_* = 0l)

The default multiplier is 1 073 741 831.

Parameters:

a_ multiplier a

seed_ seed

The documentation for this class was generated from the following file:

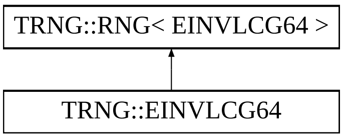
- **trng.h**

6.5 TRNG::EINVLCG64 Class Reference

combined generator.

`#include <trng.h>`

Inheritance diagram for TRNG::EINVLCG64::



Public Methods

- **EINVLCG64** (long seed_=0l)

  constructor.

6.5.1 Detailed Description

This is a combined generator.

$$
\begin{aligned}
q_{1,i} &= \overline{1073741831 \cdot i} \mod (2^{30} + 2^{28} + 3) \\
q_{2,i} &= \left\lfloor \frac{q_{3,i}}{2^{33}} \right\rfloor \\
q_{3,i} &= 18\,145\,460\,002\,477\,866\,997 \cdot q_{3,i-1} + 1 \mod 2^{64} \\
r_i &= q_{1,i} + q_{2,i} \mod 2^{31}
\end{aligned}
$$

Author:
Heiko Bauke

Definition at line 1934 of file trng.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 TRNG::EINVLCG64::EINVLCG64 (long *seed_* = 0l)

Parameters:
seed_ seed

The documentation for this class was generated from the following file:

- **trng.h**

6.6 TRNG::error Class Reference

class for error handling.

`#include <trnglib.h>`

6.6.1 Detailed Description

This class is thrown, if an error occurs.

Definition at line 35 of file trnglib.h.

The documentation for this class was generated from the following file:

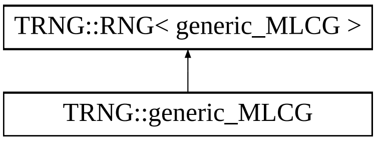
- **trnglib.h**

6.7 TRNG::generic_MLCG Class Reference

linear congruential generator.

`#include <trng.h>`

Inheritance diagram for TRNG::generic_MLCG::



6.7.1 Detailed Description

This class implements a simple linear congruential pseudo random number generator with a prime modulus in the form

$$r_i = a \cdot r_{i-1} \mod m \,.$$

r_i is the actual pseudo random number. The modulus m has to be a prime smaller than 2^{31} and a a generating element of the multiplicative group modulo m to generate a maxmal length period. The behaviour of method **rand**() (p. 49) is different from other generators, it returns never 0. For this reason this generator is for internal use only.

Author:
Heiko Bauke

Definition at line 1050 of file trng.h.

The documentation for this class was generated from the following file:

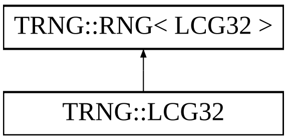
- **trng.h**

6.8 TRNG::LCG32 Class Reference

linear congruential generator.

`#include <trng.h>`

Inheritance diagram for TRNG::LCG32::



Public Methods

- **LCG32** (unsigned long a_=69069ul, unsigned long b_=1ul, long seed_=0l)
  constructor.

6.8.1 Detailed Description

This class implements a simple linear congruential pseudo random number generator with a **power** (p. 45) of two modulus in the form

$$q_i = a \cdot q_{i-1} + b \mod 2^{32}$$
$$r_i = \lfloor q_i/2 \rfloor \,.$$

r_i is the actual pseudo random number. To get a full period of 2^{32} a and b have to be choosen that $a \equiv 1 \mod 4$ and b is odd.

Author:
Heiko Bauke

Definition at line 1145 of file trng.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 TRNG::LCG32::LCG32 (unsigned long *a_* = 69069ul, unsigned long *b_* = 1ul, long *seed_* = 0l)

The constructor's default values implement a pseudo random number generator used on VAX. This generator has a period of $2^{32} \approx 4.29 \cdot 10^9$. This generator is just a toy. Its period is too short and it has some bad statistical properties.

Parameters:

a_	multiplier a
b_	additive constant
seed_	default seed

The documentation for this class was generated from the following file:

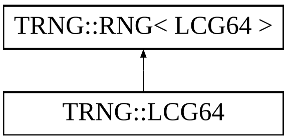
- **trng.h**

6.9 TRNG::LCG64 Class Reference

linear congruential generator.

`#include <trng.h>`

Inheritance diagram for TRNG::LCG64::



Public Methods

- **LCG64** (unsigned long long a_=18145460002477866997ull, unsigned long long b_=1ul, long seed_=0ul)

  constructor.

6.9.1 Detailed Description

This class implements a simple linear congruential pseudo random number generator with a **power** (p. 45) of two modulus in the form

$$\begin{aligned} q_i &= a \cdot q_{i-1} + b \mod 2^{64} \\ r_i &= \lfloor q_i / 2^{33} \rfloor \, . \end{aligned}$$

r_i is the actual pseudo random number. To get a full period of 2^{64} a and b have to be choosen that $a \equiv 1 \mod 4$ and b is odd.

Author:

Heiko Bauke

Definition at line 1201 of file trng.h.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 TRNG::LCG64::LCG64 (unsigned long long *a_* = 18145460002477866997ull, unsigned long long *b_* = 1ul, long *seed_* = 0ul)

The constructor's default values implement a pseudo random number generator with $a = 18\,145\,460\,002\,477\,866\,997$ and $b = 1$. This generator has a period of $2^{64} \approx 1.84 \cdot 10^{19}$. **LCG64** (p. 40) is the quick and dirty generator in **TRNG** (p. 11).

Parameters:

a_ multiplier a

b_ additive constant

seed_ default seed

The documentation for this class was generated from the following file:

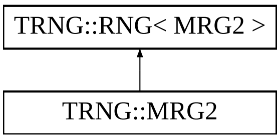
- **trng.h**

6.10 TRNG::MRG2 Class Reference

multiple recursive generator.

`#include <trng.h>`

Inheritance diagram for TRNG::MRG2::



Public Methods

- **MRG2** (long a0_=1498809829l, long a1_=1160990996l, long modulus_=2147483647l, long seed_=0l)

  constructor.

6.10.1 Detailed Description

This multiple recursive generator uses a linear recurrence of order two with a prime modulus.

$$r_i = a_1 \cdot r_{i-1} + a_2 \cdot r_{i-2} \mod m$$

Author:

Heiko Bauke

Definition at line 1249 of file trng.h.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 TRNG::MRG2::MRG2 (long *a0_* = 1498809829l, long *a1_* = 1160990996l, long *modulus_* = 2147483647l, long *seed_* = 0l)

The constructor's default values implement a pseudo random number generator with $a_1 = 1\,498\,809\,829$, $a_2 = 1\,160\,990\,996$ and $m = 2^{31} - 1$ as proposed in [9]. This generator has a period of $2^{2\cdot 31} - 1 \approx 2^{62} \approx 4.61 \cdot 10^{18}$.

Parameters:

a0_ multiplier a_1

a1_ multiplier a_2

modulus_ prime modulus

seed_ default seed

The documentation for this class was generated from the following file:

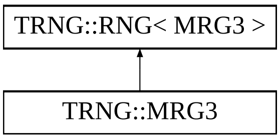
- **trng.h**

6.11 TRNG::MRG3 Class Reference

multiple recursive generator.

`#include <trng.h>`

Inheritance diagram for TRNG::MRG3::



Public Methods

- **MRG3** (long a0_=2021422057l, long a1_=1826992351l, long a2_=1977753457l, long modulus_=2147483647l, long seed_=0l)

  constructor.

6.11.1 Detailed Description

This multiple recursive generator uses a linear recurrence of order three with a prime modulus.

$$r_i = a_1 \cdot r_{i-1} + a_2 \cdot r_{i-2} + a_3 \cdot r_{i-3} \mod m$$

Author:

Heiko Bauke

Definition at line 1303 of file trng.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 TRNG::MRG3::MRG3 (long *a0_* = 2021422057l, long *a1_* = 1826992351l, long *a2_* = 1977753457l, long *modulus_* = 2147483647l, long *seed_* = 0l)

The constructor's default values implement a pseudo random number generator with $a_1 = 2\,021\,422\,057$, $a_2 = 1\,826\,992\,351$, $a_3 = 1\,977\,753\,457$ and $m = 2^{31} - 1$ as proposed in [9]. This generator has a period of $2^{3\cdot 31} - 1 \approx 2^{93} \approx 9.90 \cdot 10^{27}$.

Parameters:

a0_ multiplier a_1

a1_ multiplier a_2

a2_ multiplier a_3

modulus_ prime modulus

seed_ default seed

The documentation for this class was generated from the following file:

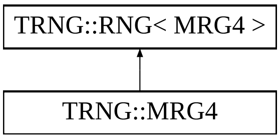
- **trng.h**

6.12 TRNG::MRG4 Class Reference

multiple recursive generator.

`#include <trng.h>`

Inheritance diagram for TRNG::MRG4::



Public Methods

- **MRG4** (long a0_=2001982722l, long a1_=1412284257l, long a2_=1155380217l, long a3_=1668339922l, long modulus_=2147483647l, long seed_=0l)

  constructor.

6.12.1 Detailed Description

This multiple recursive generator uses a linear recurrence of order four with a prime modulus.

$$r_i = a_1 \cdot r_{i-1} + a_2 \cdot r_{i-2} + a_3 \cdot r_{i-3} + a_4 \cdot r_{i-4} \mod m$$

Author:

Heiko Bauke

Definition at line 1362 of file trng.h.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 TRNG::MRG4::MRG4 (long *a0_* = 2001982722l, long *a1_* = 1412284257l, long *a2_* = 1155380217l, long *a3_* = 1668339922l, long *modulus_* = 2147483647l, long *seed_* = 0l)

The constructor's default values implement a pseudo random number generator with $a_1 = 2\,001\,982\,722$, $a_2 = 1\,412\,284\,257$, $a_3 = 1\,155\,380\,217$, $a_4 = 1\,668\,339\,922$ and $m = 2^{31} - 1$ as proposed by [9]. This generator has a period of $2^{4\cdot 31} - 1 \approx 2^{124} \approx 2.13 \cdot 10^{37}$.

Parameters:

a0_ multiplier a_1

a1_ multiplier a_2

a2_ multiplier a_3

a3_ multiplier a_4

modulus_ prime modulus

seed_ default seed

The documentation for this class was generated from the following file:

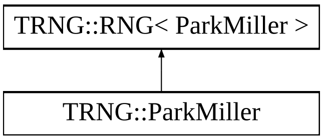
- **trng.h**

6.13 TRNG::ParkMiller Class Reference

linear congruential generator.

`#include <trng.h>`

Inheritance diagram for TRNG::ParkMiller::



Public Methods

- **ParkMiller** (long a_=16807l, long modulus_=2147483647l, long seed_=0l)
  constructor.

6.13.1 Detailed Description

This class implements a simple linear congruential pseudo random number generator with a prime modulus in the form

$$\begin{aligned} q_i &= a \cdot q_{i-1} \mod m \\ r_i &= q_i - 1\,. \end{aligned}$$

r_i is the actual pseudo random number. The modulus m has to be a prime smaller than 2^{31} and a a generating element of the multiplicative group modulo m to generate a maxmal length period.

Author:

Heiko Bauke

Definition at line 1096 of file trng.h.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 TRNG::ParkMiller::ParkMiller (long *a_* = 16807l, long *modulus_* = 2147483647l, long *seed_* = 0l)

The constructor's default values implement a modified random number generator proposed by Park and Miller. This generator has a period of $2^{31} - 2 \approx 2^{31} \approx 2.15 \cdot 10^9$. See also [8].

Parameters:

a_ multiplier a

modulus_ prime modulus m

seed_ default seed

The documentation for this class was generated from the following file:

- **trng.h**

6.14 TRNG::power Class Reference

power (p. 45).

`#include <trnglib.h>`

6.14.1 Detailed Description

This data structure is used in the YARN random number generator family. It allows the fast calculation of all powers of a given base b modulo m.

Definition at line 51 of file trnglib.h.

The documentation for this class was generated from the following file:

- **trnglib.h**

6.15 TRNG::RNG< RNG_type > Class Template Reference

pseudo random number generator template.

`#include <trng.h>`

Public Methods

- const char ∗ **name** (void)

  pseudo random number generator's name.

- void **reset** (void)

  reset the pseudo random number generator.

- void **seed** (long s=0l)

  seed the pseudo random number generator.

- long **rand** (void)

  next pseudo random number.

- long **max** (void)

  maximal pseudo random number.

- bool **boolean** (void)

  pseudo random boolean.

- bool **boolean** (const double p)

  pseudo random boolean.

- double **uniform** (void)

  pseudo random number.

- double **uniform** (const double a, const double b)

  pseudo random number.

- double **uniformco** (void)

  pseudo random number.

- double **uniformco** (const double a, const double b)

  pseudo random number.

- double **uniformcc** (void)

  pseudo random number.

- double **uniformcc** (const double a, const double b)

  pseudo random number.

- double **uniformoc** (void)

  pseudo random number.

- double **uniformoc** (const double a, const double b)

  pseudo random number.

- double **uniformoo** (void)

  pseudo random number.

- double **uniformoo** (const double a, const double b)

  pseudo random number.

- long **uniforml** (const long b)

  pseudo random number.

- long **uniforml** (const long a, const long b)

  pseudo random number.

- double **normal_dist** (const double sigma=1.0, const double mu=0.0)

  pseudo random number.

- double **exp_dist** (const double mu=1.0)

  pseudo random number.

- double **laplace_dist** (const double a=1.0) *pseudorandom number.*

- double **tent_dist** (const double a=1.0)
  pseudorandom number.
- double **Gamma_dist** (const double a, const double b)
  pseudo random number.
- double **Beta_dist** (const double a, const double b)
  pseudo random number.
- double **chi_square_dist** (const double nu)
  pseudo random number.
- long **binomial_dist** (long n, double p=0.5)
  pseudo random number.
- long **binomial_dist_tab** (long n, double p=0.5)
  pseudo random number.
- double **Student_t_dist** (const double nu)
  pseudo random number.
- long **poisson_dist** (double mu=1.0)
  pseudo random number.
- long **geometric_dist** (double q)
  pseudo random number.
- template<class t_function> double **rejection** (t_function p, double a1, double a2, double p_max)
  pseudo random number.
- long **discrete_dist** (const std::vector< double > p)
  pseudo random number.
- **vector2d spherical2d** (void)
  pseudo random vector.
- **vector3d spherical3d** (void)
  pseudo random vector.
- **vector4d spherical4d** (void)
  pseudo random vector.
- void **split** (long s, long n)
  sequence splitting.
- void **jump** (long long s)
  sequence splitting.
- void **jump** (long long s, long n)
  sequence splitting.

- void **jump2** (long s)
  sequence splitting.

- void **jump2** (long s, long n)
  sequence splitting.

- void **save_status** (std::vector< long > &s)
  status saving.

- void **load_status** (const std::vector< long > &s)
  status restoring.

Static Public Attributes

- const **TRNG::RNG_type type** = RNG_t
  pseudo random number generator type.

Protected Attributes

- long **max_val**
  maximum value that the random number gernerator's method **rand**() (p. 49) *can return.*

- long **max_val2**
  the half of the maximum value that the random number gernerator's method **rand**() (p. 49) *can return.*

6.15.1 Detailed Description

template<class RNG_type> class TRNG::RNG< RNG_type >

All the pseudo random number generators are derived form this base class. Routines for generation of uniform and nonuniform variartes are implemented in this base class. Generator specific tasks like sequence splitting or leapfrog method are implementetd by the derived classes. The method **rand**() (p. 49), the generator's core method, has of course also to be reimplemented for every new generator.

Author:
Heiko Bauke

Definition at line 133 of file trng.h.

6.15.2 Member Function Documentation

6.15.2.1 template<class RNG_type> const char* TRNG::RNG< RNG_type >::name (void) `[inline]`

Returns a pointer to a zero terminated string containing the pseudo random number generator's name.

Returns:
pointer to a zero termiated string

Definition at line 157 of file trng.h.

6.15.2.2 template<class RNG_type> void TRNG::RNG< RNG_type >::reset (void) `[inline]`

The pseudo random number generator's parameters are set to some default values.

Definition at line 166 of file trng.h.

6.15.2.3 template<class RNG_type> void TRNG::RNG< RNG_type >::seed (long s = 0l) `[inline]`

The pseudo random number generator's state is set. Calling **reset**() (p. 49) and **seed**() (p. 49) with the seed value used by the initialisation will set the generator in the same state as after initialisation.

Parameters:

s new seed

Exceptions:

error (p. 38) if $s < 0$

See also:

TRNG::error (p. 38)

Definition at line 179 of file trng.h.

6.15.2.4 template<class RNG_type> long TRNG::RNG< RNG_type >::rand (void) `[inline]`

This is the generator's core method. It calculates the next pseudo random number. It's an integer number r with $0 \leq r \leq \max$. You can determine the upper bound max by calling the method **max**() (p. 49).

Returns:

next integer pseudo random number

Definition at line 191 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::boolean(), TRNG::RNG< LCG32 >::uniformcc(), TRNG::RNG< LCG32 >::uniformco(), TRNG::RNG< LCG32 >::uniformoc(), and TRNG::RNG< LCG32 >::uniformoo().

6.15.2.5 template<class RNG_type> long TRNG::RNG< RNG_type >::max (void) `[inline]`

Returns:

maximal pseudo random number returned by **rand**() (p. 49)

Definition at line 199 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::boolean(), TRNG::RNG< LCG32 >::uniformcc(), TRNG::RNG< LCG32 >::uniformco(), TRNG::RNG< LCG32 >::uniformoc(), and TRNG::RNG< LCG32 >::uniformoo().

6.15.2.6 template<class RNG_type> bool TRNG::RNG< RNG_type >::boolean (void) `[inline]`

Returns:

pseudo random boolean with probability $\frac{1}{2}$ for return value true

Definition at line 208 of file trng.h.

6.15.2.7 template<class RNG_type> bool TRNG::RNG< RNG_type >::boolean (const double *p*) `[inline]`

Returns:
pseudo random boolean with probability p for return value true

Definition at line 217 of file trng.h.

6.15.2.8 template<class RNG_type> double TRNG::RNG< RNG_type >::uniform (void) `[inline]`

Returns:
a in $[0, 1)$ uniform distributed random number

Definition at line 225 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::uniforml().

6.15.2.9 template<class RNG_type> double TRNG::RNG< RNG_type >::uniform (const double *a*, const double *b*) `[inline]`

Returns:
a in $[a, b)$ uniform distributed random number

Parameters:
- ***a*** lower bound
- ***b*** upper bound

Definition at line 235 of file trng.h.

6.15.2.10 template<class RNG_type> double TRNG::RNG< RNG_type >::uniformco (void) `[inline]`

Returns:
a in $[0, 1)$ uniform distributed random number

Definition at line 243 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::binomial_dist(), TRNG::RNG< LCG32 >::binomial_dist_tab(), TRNG::RNG< LCG32 >::discrete_dist(), TRNG::RNG< LCG32 >::Gamma_dist(), TRNG::RNG< LCG32 >::poisson_dist(), TRNG::RNG< LCG32 >::rejection(), TRNG::RNG< LCG32 >::spherical2d(), TRNG::RNG< LCG32 >::spherical3d(), TRNG::RNG< LCG32 >::spherical4d(), and TRNG::RNG< LCG32 >::uniform().

6.15.2.11 template<class RNG_type> double TRNG::RNG< RNG_type >::uniformco (const double *a*, const double *b*) `[inline]`

Returns:
a in $[a, b)$ uniform distributed random number

Parameters:
- ***a*** lower bound
- ***b*** upper bound

Definition at line 254 of file trng.h.

6.15.2.12 template<class RNG_type> double TRNG::RNG< RNG_type >::uniformcc (void) `[inline]`

Returns:
a in $[0, 1]$ uniform distributed random number

Definition at line 263 of file trng.h.

6.15.2.13 template<class RNG_type> double TRNG::RNG< RNG_type >::uniformcc (const double *a*, const double *b*) `[inline]`

Returns:
a in $[a, b]$ uniform distributed random number

Parameters:

a lower bound

b upper bound

Definition at line 274 of file trng.h.

6.15.2.14 template<class RNG_type> double TRNG::RNG< RNG_type >::uniformoc (void) `[inline]`

Returns:
a in $(0, 1]$ uniform distributed random number

Definition at line 283 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::exp_dist().

6.15.2.15 template<class RNG_type> double TRNG::RNG< RNG_type >::uniformoc (const double *a*, const double *b*) `[inline]`

Returns:
a in $(a, b]$ uniform distributed random number

Parameters:

a lower bound

b upper bound

Definition at line 294 of file trng.h.

6.15.2.16 template<class RNG_type> double TRNG::RNG< RNG_type >::uniformoo (void) `[inline]`

Returns:
a in $(0, 1)$ uniform distributed random number

Definition at line 303 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::Gamma_dist(), TRNG::RNG< LCG32 >::geometric_dist(), TRNG::RNG< LCG32 >::laplace_dist(), TRNG::RNG< LCG32 >::normal_dist(), and TRNG::RNG< LCG32 >::tent_dist().

6.15.2.17 template<class RNG_type> double TRNG::RNG< RNG_type >::uniformoo (const double *a*, const double *b*) `[inline]`

Returns:

a in (a, b) uniform distributed random number

Parameters:

a lower bound

b upper bound

Definition at line 314 of file trng.h.

6.15.2.18 template<class RNG_type> long TRNG::RNG< RNG_type >::uniforml (const long *b*) `[inline]`

Returns:

If $b > 0$ a in $[0, b)$ uniform distributed natural random number is returned, if $b < 0$ return value is in $(b, 0]$.

Parameters:

b upper bound

Definition at line 324 of file trng.h.

6.15.2.19 template<class RNG_type> long TRNG::RNG< RNG_type >::uniforml (const long *a*, const long *b*) `[inline]`

Returns:

a in $[a, b)$ uniform distributed natural random number

Parameters:

a lower bound

b upper bound

Definition at line 334 of file trng.h.

6.15.2.20 template<class RNG_type> double TRNG::RNG< RNG_type >::normal_dist (const double *sigma* = 1.0, const double *mu* = 0.0) `[inline]`

A normal distributed random variate has a probability density

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{x^2}{2\sigma^2}} .$$

This method uses the polar (Box-Mueller) method, see [3] for details.

Returns:

a normal distributed random number with mean μ and variance σ

Parameters:

sigma variance σ

mu mean μ

Exceptions:

error (p. 38) if $\sigma \leq 0$

See also:

TRNG::error (p. 38)

Definition at line 354 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::Student_t_dist().

6.15.2.21 template<class RNG_type> double TRNG::RNG< RNG_type >::exp_dist (const double *mu* = 1.0) `[inline]`

A exponential distributed random variate has a probability density

$$p(x) = \begin{cases} \frac{1}{\mu}\mathrm{e}^{-\frac{x}{\mu}} & x \geq 0\,, \qquad \mu > 0 \\ 0 & \text{else} \end{cases} .$$

The transformation method is discribed in [3]. One random number is used to get one random number with exponential distribution.

Returns:

a exponential distributed random number with mean μ

Parameters:

mu mean μ

Exceptions:

error (p. 38) if $\mu \leq 0$

See also:

TRNG::error (p. 38)

Definition at line 395 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::Student_t_dist().

6.15.2.22 template<class RNG_type> double TRNG::RNG< RNG_type >::laplace_dist (const double *a* = 1.0) `[inline]`

The two-sided exponential probability distribution is

$$p(x) = \frac{1}{2a}\mathrm{e}^{-|x|/a}\,, \qquad a > 0\,.$$

It is also known as the Laplace distribution. The is implementation adopted from the GNU scientific library and uses a simple transformation method. One random number is used to get one random number with Laplace distribution.

Returns:

a pseudo random number with probability density $p(x)$

Parameters:

a parameter a, $a > 0$

Exceptions:

error (p. 38) if $a \leq 0$

See also:

TRNG::error (p. 38)

Definition at line 417 of file trng.h.

6.15.2.23 template<class RNG_type> double TRNG::RNG< RNG_type >::tent_dist (const double *a* = 1.0) `[inline]`

This method generates a pseudo randeom number with a tent shaped probability distribution.

$$p(x) = \begin{cases} \dfrac{x+a}{a^2} & -a \leq x \leq 0 \\ \dfrac{a-x}{a^2} & 0 \leq x \leq a \\ 0 & \text{else} \end{cases}, \qquad \text{with } a > 0$$

The implementation uses a simple transformation method. One random number is used to get one random number with a tent shaped distribution.

Returns:
a pseudo random number with probability density $p(x)$

Parameters:
a parameter a, $a > 0$

Exceptions:
error (p. 38) if $a \leq 0$

See also:
TRNG::error (p. 38)

Definition at line 448 of file trng.h.

6.15.2.24 template<class RNG_type> double TRNG::RNG< RNG_type >::Gamma_dist (const double *a*, const double *b*) `[inline]`

A gamma distributed random number has a probability density

$$p(x) = \begin{cases} \dfrac{1}{\Gamma(a) b^a} x^{a-1} \mathrm{e}^{-\frac{x}{b}} & x > 0 \\ 0 & \text{else} \end{cases}.$$

The is implemetation adopted from the GNU scientific library. See also [3].

Returns:
a gamma distributed random number

Parameters:
a parameter a

b parameter b

Exceptions:
error (p. 38) if $a \leq 0$ or $b \leq 0$

See also:
TRNG::error (p. 38)

Definition at line 478 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::Beta_dist(), TRNG::RNG< LCG32 >::chi_square_dist(), TRNG::RNG< LCG32 >::Gamma_dist(), and TRNG::RNG< LCG32 >::poisson_dist().

6.15.2.25 template<class RNG_type> double TRNG::RNG< RNG_type >::Beta_dist (const double *a*, const double *b*) `[inline]`

A Beta distributed random number has a probability density

$$p(x) = \begin{cases} \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1} & 0 \le x \le 1 \\ 0 & \text{else} \end{cases}$$

The method is discribed in [3] pp. 129-130.

Returns:

a Beta distributed random number

Parameters:

a parameter a, $a > 0$

b parameter b, $b > 0$

Exceptions:

error (p. 38) if $a \le 0$ or $b \le 0$

See also:

TRNG::error (p. 38)

Definition at line 556 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::binomial_dist().

6.15.2.26 template<class RNG_type> double TRNG::RNG< RNG_type >::chi_square_dist (const double *nu*) `[inline]`

The χ^2-distribution is just a special case of the Γ-distribution with $a = \nu/2$ and $b = 1$.

$$p(x) = \frac{\left(\frac{x}{2}\right)^{\frac{\nu}{2}-1}}{2\Gamma\left(\frac{\nu}{2}\right)} \mathrm{e}^{-\frac{x}{2}}\,, \qquad x \le 0$$

The method is discribed in [3] p. 130.

Returns:

a chi square distributed random number

Parameters:

nu degrees of freedom ν

Exceptions:

error (p. 38) if $\nu < 1$

See also:

TRNG::error (p. 38)

Definition at line 581 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::Student_t_dist().

6.15.2.27 template<class RNG_type> long TRNG::RNG< RNG_type >::binomial_dist (long *n*, double *p* = 0.5) `[inline]`

The binomial distribution is a discrete distribution with

$$p(k) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}\,, \qquad 0 \le k \le n\,.$$

The method is discribed in [3] p. 131. The implemetation is adopted from the GNU scientific library.

Returns:
a binomial distributed pseudo random number

Parameters:

- ***n*** number of trails n
- ***p*** probability p in each trail

Exceptions:
error (p. 38) if $p \le 0$ or $p > 1$ or $n \le 0$

See also:
TRNG::error (p. 38)

Definition at line 603 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::poisson_dist().

6.15.2.28 template<class RNG_type> long TRNG::RNG< RNG_type >::binomial_dist_tab (long *n*, double *p* = 0.5) `[inline]`

The binomial distribution is a discrete distribution with

$$p(k) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}\,, \qquad 0 \le k \le n\,.$$

This method's implementation uses a lookup table and is very fast if this method is called often with the same parameter set.

Returns:
a binomial distributed pseudo random number

Parameters:

- ***n*** number of trails n
- ***p*** probability p in each trail

Exceptions:
error (p. 38) if $p \le 0$ or $p > 1$ or $n \le 0$

See also:
TRNG::error (p. 38)

Definition at line 643 of file trng.h.

6.15.2.29 template<class RNG_type> double TRNG::RNG< RNG_type >::Student_t_dist (const double *nu*) `[inline]`

Student's t-distribution with ν degrees of freedom is defined as

$$p(x) = \frac{\Gamma\left(\frac{\nu-1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}\,.$$

This method's implementation is adopted from the GNU scientific library, see also [3].

Returns:
a random number with Student's-t distribution with ν degrees of freedom

Parameters:
nu degrees of freedom, $\nu > 0$

Exceptions:
error (p. 38) if $\nu \leq 0$

See also:
TRNG::error (p. 38)

Definition at line 689 of file trng.h.

6.15.2.30 template<class RNG_type> long TRNG::RNG< RNG_type >::poisson_dist (double *mu* = 1.0) `[inline]`

The probability distribution for Poisson variates is

$$p(k) = \frac{\mu^k}{k!}\mathrm{e}^{-\mu}\,, \qquad k \geq 0\,.$$

This method's implementation is adopted from the GNU scientific library, see also [3].

Returns:
a poisson distributed pseudo random number

Parameters:
mu mean μ

Exceptions:
error (p. 38) if $\mu \leq 0$

See also:
TRNG::error (p. 38)

Definition at line 721 of file trng.h.

6.15.2.31 template<class RNG_type> long TRNG::RNG< RNG_type >::geometric_dist (double *q*) `[inline]`

The geometric probability distribution is

$$p(k) = q(1-q)^{k-1}\,, \qquad k \geq 1\,.$$

This method's implementation is adopted from the GNU scientific library, see also [3].

Returns:

a geometric distributed pseudo random number

Parameters:

q probability q

Exceptions:

error (p. 38) if $q \leq 0$ or $q > 1$

See also:

TRNG::error (p. 38)

Definition at line 759 of file trng.h.

6.15.2.32 template<class RNG_type> template<class t_function> double TRNG::RNG< RNG_type >::rejection (t_function *p*, double *a1*, double *a2*, double *p_max*) `[inline]`

Returns a random number calculated by the rejection method. Assume your desired probability distribution is $p(x) = \frac{3}{4}(1 - x^2)$ for $x \in [-1, 1]$. Write a class that calculates this probability distribution

```
class p {
public:
  double operator()(double x) {
    return 0.75*(1.0-x*x);
  }
};
```

and use

```
my_rng.rejection(p(), -1.0, 1.0, 0.75)
```

to generate a pseudo random number with probability distribution $p(x)$.

Returns:

random number

Parameters:

p function object, a function describing the probability density in the range between a_1 and a_2

a1 lower bound a_1

a2 upper bound a_2

p_max maximum of the probability function $p(x)$ for $x \in [a_1, a_2]$

Definition at line 796 of file trng.h.

6.15.2.33 template<class RNG_type> long TRNG::RNG< RNG_type >::discrete_dist (const std::vector< double > *p*) `[inline]`

This method can be used to generate discrete random variates with an arbitrary probability distribution. The algorithm is $O(\ln n)$. If you need a faster algorithm see [10].

Returns:

random number k, $0 \leq k \leq n - 1$

Parameters:

p n dimensional commulative probability vector p_0, $p_1, \ldots p_{n-1}$, with $p_i < p_{i+1}$ and $p_{n-1} = 1$

Exceptions:

error (p. 38) if vector empty

See also:

TRNG::error (p. 38)

Definition at line 818 of file trng.h.

6.15.2.34 template<class RNG_type> vector2d TRNG::RNG< RNG_type >::spherical2d (void) `[inline]`

This method calculates a unit vector with a uniform distributed direction in two dimensions. For a vector uniform distributed inside the unit citcle multiply the vector with $\sqrt{u}$ where u is uniforum distributed in $[0, 1)$. The vector is stored in a structure **vector2d** (p. 16).

Returns:

a two dimensional unit vector

Definition at line 845 of file trng.h.

6.15.2.35 template<class RNG_type> vector3d TRNG::RNG< RNG_type >::spherical3d (void) `[inline]`

This method calculates a unit vector with a uniform distributed direction in three dimensions. For a vector uniform distributed inside the unit shere multiply the vector with $\sqrt[3]{u}$ where u is uniforum distributed in $[0, 1)$. The vector is stored in a structure **vector3d** (p. 16).

Returns:

a three dimensional unit vector

Definition at line 869 of file trng.h.

6.15.2.36 template<class RNG_type> vector4d TRNG::RNG< RNG_type >::spherical4d (void) `[inline]`

This method calculates a unit vector with a uniform distributed direction in four dimensions. For a vector uniform distributed inside the unit hyper-shere multiply the vector with $\sqrt[4]{u}$ where u is uniforum distributed in $[0, 1)$. The vector is stored in a structure **vector4d** (p. 16).

Returns:

a four dimensional unit vector

Definition at line 893 of file trng.h.

6.15.2.37 template<class RNG_type> void TRNG::RNG< RNG_type >::split (long *s*, long *n*) `[inline]`

The pseudo random number generator's sequence is splitted into s sequences using the leapfrog method. Sequence number n is selected. $0 \leq n < s$

Parameters:

s number of sequences

n selected sequence

Exceptions:

error (p. 38) if $s < 1$ or $n \geq s$ or $n < 0$

See also:
TRNG::error (p. 38)

Definition at line 924 of file trng.h.

6.15.2.38 template<class RNG_type> void TRNG::RNG< RNG_type >::jump (long long *s*) `[inline]`

The pseudo random number generator jumps s steps ahead.

Parameters:
s determines the jump size

Exceptions:
error (p. 38) if $s < 0$

See also:
TRNG::error (p. 38)

Definition at line 935 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::jump().

6.15.2.39 template<class RNG_type> void TRNG::RNG< RNG_type >::jump (long long *s*, long *n*) `[inline]`

The pseudo random number generator jumps $n \cdot s$ steps ahead.

Parameters:
s determines the jump size

Exceptions:
error (p. 38) if $s < 0$ or $n < 0$

Parameters:
s paremeter s
n paremeter n

See also:
TRNG::error (p. 38)

Definition at line 956 of file trng.h.

6.15.2.40 template<class RNG_type> void TRNG::RNG< RNG_type >::jump2 (long *s*) `[inline]`

The pseudo random number generator jumps 2^s steps ahead.

Parameters:
s determines the jump size

Exceptions:
error (p. 38) if $s < 0$

See also:
TRNG::error (p. 38)

Definition at line 972 of file trng.h.

Referenced by TRNG::RNG< LCG32 >::jump(), and TRNG::RNG< LCG32 >::jump2().

6.15.2.41 template<class RNG_type> void TRNG::RNG< RNG_type >::jump2 (long *s*, long *n*) `[inline]`

The pseudo random number generator jumps $n \cdot 2^s$ steps ahead.

Parameters:

s determines the jump size

n determines the jump size

Exceptions:

if $s < 0$ or $n < 0$

See also:

TRNG::error (p. 38)

Definition at line 984 of file trng.h.

6.15.2.42 template<class RNG_type> void TRNG::RNG< RNG_type >::save_status (std::vector< long > & *s*) `[inline]`

The status of the pseudo random number generator is saved into a vector.

Parameters:

s reference to a vector of long

Definition at line 1001 of file trng.h.

6.15.2.43 template<class RNG_type> void TRNG::RNG< RNG_type >::load_status (const std::vector< long > & *s*) `[inline]`

The status of the pseudo random number generator is restored from a vector.

Parameters:

s reference to a vector of long

Definition at line 1011 of file trng.h.

6.15.3 Member Data Documentation

6.15.3.1 template<class RNG_type> const TRNG::RNG_type TRNG::RNG< RNG_type >::type = RNG_t `[static]`

This numerical value determins the random number generator type.

Definition at line 149 of file trng.h.

The documentation for this class was generated from the following file:

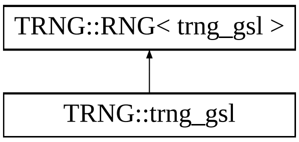
- **trng.h**

6.16 TRNG::trng_gsl Class Reference

wrapper class for GSL random number generators.

`#include <trng_gsl.h>`

Inheritance diagram for TRNG::trng_gsl::



Public Methods

- **trng_gsl** (const gsl_rng_type *T_=gsl_rng_mt19937, long seed_=0l)
  constructor.

6.16.1 Detailed Description

This class implements a simple wrapper for the random number generators in the GNU Scientific Library. Don't use tese generators in parallel applications. Leapfrog and jumping ahead are implemented by throwing some numbers away.

See also:
`http://sources.redhat.com/gsl/ref/gsl-ref_toc.html`

Author:
Heiko Bauke

Definition at line 38 of file trng_gsl.h.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 TRNG::trng_gsl::trng_gsl (const gsl_rng_type * *T_* = gsl_rng_mt19937, long *seed_* = 0l) `[inline]`

The constructor takes two agruments.

Parameters:
T_ random number generator type, see also GNU Scientific Library Reference Manual.
seed_ default seed

See also:
`http://sources.redhat.com/gsl/ref/gsl-ref_17.html`

Definition at line 161 of file trng_gsl.h.

The documentation for this class was generated from the following file:

- **trng_gsl.h**

6.17 TRNG::vector2d_struct Struct Reference

two dimensional vector structure.

`#include <trng.h>`

Public Attributes

- double **x1**

  1st element.

- double **x2**

  2nd element.

The documentation for this struct was generated from the following file:

- **trng.h**

6.18 TRNG::vector3d_struct Struct Reference

three dimensional vector structure.

`#include <trng.h>`

Public Attributes

- double **x1**

  1st element.

- double **x2**

  2nd element.

- double **x3**

  3rd element.

The documentation for this struct was generated from the following file:

- **trng.h**

6.19 TRNG::vector4d_struct Struct Reference

four dimensional vector structure.

`#include <trng.h>`

Public Attributes

- double **x1**

  1st element.

- double **x2**

  2nd element.

- double **x3**

  3rd element.

- double **x4**

  4th element.

The documentation for this struct was generated from the following file:

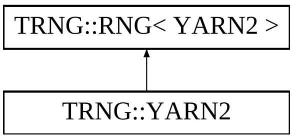
- **trng.h**

6.20 TRNG::YARN2 Class Reference

YARN – a modified multiple recursive generator.

`#include <trng.h>`

Inheritance diagram for TRNG::YARN2::



Public Methods

- **YARN2** (long a0_=1498809829l, long a1_=1160990996l, long modulus_=2147483647l, long seed_=0l, long b_=1235678931l)

  constructor.

6.20.1 Detailed Description

This multiple recursive generator uses a linear recurrence of order two with a prime modulus.

$$q_i = a_1 \cdot q_{i-1} + a_2 \cdot q_{i-2} \mod m$$

Linear structures are destroyed by a nonlinear bijective mapping from $r_i = \begin{cases} b^{q_i} \mod m & \text{if } q_i \neq 0 \\ 0 & \text{if } q_i = 0 \end{cases}$. The parameter b has to be a generating element of the multiplicative group modulo m.

Author:
Heiko Bauke

Definition at line 1485 of file trng.h.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 TRNG::YARN2::YARN2 (long *a0_* = 1498809829l, long *a1_* = 1160990996l, long *modulus_* = 2147483647l, long *seed_* = 0l, long *b_* = 1235678931l)

The constructor's default values implement a pseudo random number generator with $a_1 = 1\,498\,809\,829$, $a_2 = 1\,160\,990\,996$ and $m = 2^{31} - 1$ as proposed in [9]. This generator has a period of $2^{2 \cdot 31} - 1 \approx 2^{62} \approx 4.61 \cdot 10^{18}$. The parameter b is chosen to be $1\,235\,678\,93l$.

Parameters:

a0_ multiplier a_1

a1_ multiplier a_2

modulus_ prime modulus

seed_ default seed

b_ parameter b

The documentation for this class was generated from the following file:

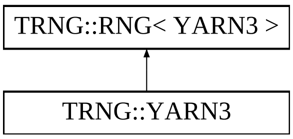
- **trng.h**

6.21 TRNG::YARN3 Class Reference

YARN – a modified multiple recursive generator.

`#include <trng.h>`

Inheritance diagram for TRNG::YARN3::



Public Methods

- **YARN3** (long a0_=2021422057l, long a1_=1826992351l, long a2_=1977753457l, long modulus_=2147483647l, long seed_=0l, long b_=123567893l)
  constructor.

6.21.1 Detailed Description

This multiple recursive generator uses a linear recurrence of order three with a prime modulus.

$$q_i = a_1 \cdot q_{i-1} + a_2 \cdot q_{i-2} + a_3 \cdot q_{i-3} \mod m$$

Linear structures are destroyed by a nonlinear bijective mapping from $r_i = \begin{cases} b^{q_i} \mod m & \text{if } q_i \neq 0 \\ 0 & \text{if } q_i = 0 \end{cases}$. The parameter b has to be a generating element of the multiplicative group modulo m.

Author:

Heiko Bauke

Definition at line 1541 of file trng.h.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 TRNG::YARN3::YARN3 (long *a0_* = 2021422057l, long *a1_* = 1826992351l, long *a2_* = 1977753457l, long *modulus_* = 2147483647l, long *seed_* = 0l, long *b_* = 123567893l)

The constructor's default values implement a pseudo random number generator with $a_1 = 2\,021\,422\,057$, $a_2 = 1\,826\,992\,351$, $a_3 = 1\,977\,753\,457$ and $m = 2^{31} - 1$ as proposed in [9]. This generator has a period of $2^{3\cdot31} - 1 \approx 2^{93} \approx 9.90 \cdot 10^{27}$. The parameter b is chosen to be $1\,235\,678\,93l$.

Parameters:

a0_ multiplier a_1

a1_ multiplier a_2

a2_ multiplier a_3

modulus_ prime modulus

seed_ default seed

b_ parameter b

The documentation for this class was generated from the following file:

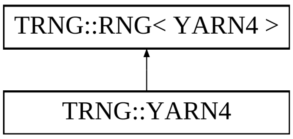
- **trng.h**

6.22 TRNG::YARN4 Class Reference

YARN – a modified multiple recursive generator.

`#include <trng.h>`

Inheritance diagram for TRNG::YARN4::



Public Methods

- **YARN4** (long a0_=2001982722l, long a1_=1412284257l, long a2_=1155380217l, long a3_=1668339922l, long modulus_=2147483647l, long seed_=0l, long b_=123567893l)

  constructor.

6.22.1 Detailed Description

This multiple recursive generator uses a linear recurrence of order four with a prime modulus.

$$q_i = a_1 \cdot q_{i-1} + a_2 \cdot q_{i-2} + a_3 \cdot q_{i-3} + a_4 \cdot q_{i-4} \mod m$$

Linear structures are destroyed by a nonlinear bijective mapping from $r_i = \begin{cases} b^{q_i} \mod m & \text{if } q_i \neq 0 \\ 0 & \text{if } q_i = 0 \end{cases}$. The parameter b has to be a generating element of the multiplicative group modulo m.

Author:

Heiko Bauke

Definition at line 1599 of file trng.h.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 TRNG::YARN4::YARN4 (long *a0_* = 2001982722l, long *a1_* = 1412284257l, long *a2_* = 1155380217l, long *a3_* = 1668339922l, long *modulus_* = 2147483647l, long *seed_* = 0l, long *b_* = 123567893l)

The constructor's default values implement a pseudo random number generator with $a_1 = 2\,001\,982\,722$, $a_2 = 1\,412\,284\,257$, $a_3 = 1\,155\,380\,217$, $a_4 = 1\,668\,339\,922$ and $m = 2^{31} - 1$ as proposed by [9]. This generator has a period of $2^{4\cdot 31} - 1 \approx 2^{124} \approx 2.13 \cdot 10^{37}$. The parameter b is chosen to be $1\,235\,678\,93l$.

Parameters:

a0_ multiplier a_1

a1_ multiplier a_2

a2_ multiplier a_3

a3_ multiplier a_4

modulus_ prime modulus

seed_ default seed

b_ parameter b

The documentation for this class was generated from the following file:

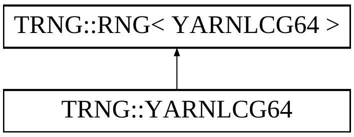
- **trng.h**

6.23 TRNG::YARNLCG64 Class Reference

YARN – a modified linear recursive generator.

`#include <trng.h>`

Inheritance diagram for TRNG::YARNLCG64::



Public Methods

- **YARNLCG64** (unsigned long long a_=18145460002477866997ull, unsigned long long b_=1ul, long seed_=0ul, long c_=123567893l)

  constructor.

6.23.1 Detailed Description

This class implements a simple linear congruential pseudo random number generator with a **power** (p. 45) of two modulus in the form

$$
\begin{aligned}
q_i &= a \cdot q_{i-1} + b \mod 2^{64} \\
r_i &= \lfloor q_i / 2^{33} \rfloor \,.
\end{aligned}
$$

r_i is the actual pseudo random number. To get a full period of 2^{64} a and b have to be choosen that $a \equiv 1 \mod 4$ and b is odd. Linear structures are destroyed by a nonlinear bijective mapping from $r_i = \begin{cases} b^{q_i} \mod 2^{31}-1 & \text{if } q_i \neq 0 \wedge q_i \neq 2^{31}-1 \\ 0 & \text{else} \end{cases}$. The parameter b has to be a generating element of the multiplicative group modulo $2^{31}-1$.

Author:

Heiko Bauke

Definition at line 1434 of file trng.h.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 TRNG::YARNLCG64::YARNLCG64 (unsigned long long *a_* = 18145460002477866997ull, unsigned long long *b_* = 1ul, long *seed_* = 0ul, long *c_* = 123567893l)

The constructor's default values implement a pseudo random number generator with $a = 18\,145\,460\,002\,477\,866\,997$ and $b = 1$. This generator has a period of $2^{64} \approx 1.84 \cdot 10^{19}$.

Parameters:

a_ multiplier a

b_ additive constant

seed_ default seed

c_ parameter c

The documentation for this class was generated from the following file:

- **trng.h**

References

[1] J. Eichenauer-Hermann. Statistical independence of a new class of inversive congruential pseudorandom numbers. *Mathematics of Computation*, 60:375–384, 1993.

[2] Alan M. Ferrenberg and D. P. Landau. Monte carlo simulations: Hidden errors from "good" random number generators. *Physical Review Letters*, 69(23):3382–3384, 1992.

[3] Donald E. Knuth. *The art of computer programming*, volume 2. Addison-Wesley, Reading, Messachusetts, 2nd edition, 1981.

[4] Donald E. Knuth. *The art of computer programming*, volume 1. Addison-Wesley, Reading, Messachusetts, 2nd edition, 1981.

[5] C. Lanczos. *SIAM Journal on Numerical Analysis, ser. B*, 1:86–96, 1964.

[6] Pierre L'Ecuyer. Efficient and protable random number generators. *Communications of the ACM*, 31(6):742–749, 774, Juni 1988.

[7] George Marsaglia. `http://stat.fsu.edu/pub/diehard/cdrom/`; `http://stat.fsu.edu/~geo/`, 1995.

[8] Stephen K. Park and Keith W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, Oktober 1988.

[9] Raymond Couture Pierre L'Ecuyer, François Blouin. A search for good multiple recursive random number generators. *ACM Transactions on Modeling and Computer Simulation*, 3(2):87–98, April 1993.

[10] Alastair J. Walker. An efficient method for generating discrete random variables with general distributions,. *ACM Trans on Mathematical Software*, 3:253–256, 1977.

Index