Random Number Generators

Geant4 School at IFIN-HH, Bucharest

Dennis Wright (SLAC)

using slides from talks by Marc Verderi and Makoto Asai

15 November 2016

Random Numbers

- The core of all Monte Carlo calculations is the ability to produce a long sequence of random numbers, uniformly distributed over the interval [0,1]
  - but digital computers, by design, are incapable of this

- A truly random sequence could be generated by coupling to an external device
  - radioactive decay
  - white noise
- Above approach is impractical
  - coupling to such a device would be cumbersome or dangerous
  - impossible to debug code if different random number is used every time

Pseudo-random Numbers

- An alternative to a truly random sequence would be one which uses a deterministic algorithm
  - given one number in the sequence, the next is efficiently calculated
- Many such pseudo-random generators have been developed
  - still an active field of research
  - example: Mersenne Twister (Matsumoto et al., 1998)

- Warning: many defective pseudo-random number generators have been used in research
  - check the history of the generator you use
  - don't change it unless you are confident of improvement

Using Pseudo-random Number Generators

- All generators discussed from now on are assumed to be pseudo-random; we simply drop the “pseudo”
  - sometimes called engines
- A generator uses a “seed” as a starting point
  - an integer or set of integers
  - from the same seed an engine always generates the same sequence of random numbers
- A generator also has a status
  - starting from a given seed, after N events, the engine is in some state which can be saved as a status
  - reloading this later on allows the engine to continue as if it had not been interrupted

Controlling Random Number Generators

- Setting the seeds
  - allows statistically independent jobs to be run in parallel
  - each of N parallel jobs will require N different seeds
  - this is a key need for simulation production

- Saving the status is useful when
  - debugging a crash → can use the last reported seed to go straight to the event in question
  - useful, but time-consuming to save seeds all the time

- The choice of engines depends on speed and accuracy considerations

Random Number Generators in Geant4

- Geant4 uses the HEPRandom module of the CLHEP library
  - documentation at geant4.org → User Support → Application Developers Guide → section 3.2.2
- HepRandomEngine is the abstract interface for random generators in CLHEP
  - all engines are of type HepRandomEngine
- A static instance exists, allowing the engine to be shared by all random number consumers
  - Geant4 uses this static instance which contains a random engine
  - engine can be changed by G4Random::setTheEngine(CLHEP::HEPRandomEngine*);
  - if you do nothing, defaults to HEPJamesRandom

Random Number Generators in Geant4

- Other available engines in CLHEP:
  - DRand48Engine, RandEngine, RanluxEngine, RanecuEngine
- Changing the engine is the only thing requiring C++ coding
  - all other generator control functions can be done interactively

- /random/setSeeds int [int [int [...]]]
  - number of ints depends on engine
- /random/setDirectoryName [dirName]
  - set or create directory in which to save engine status
- /random/setSavingFlag [value]
  - turn on (off) status change at beginning of each run or event
  - status then saved in currentRun.rndm and/or currentEvent.rndm

Random Number Generators in Geant4

- /random/saveThisRun (or saveThisEvent)
  - copy currentRun.rndm to runXXX.rndm or currentEvent.rndm to runXXXeventYYY.rndm
- /random/resetEngineFrom [fileName]
  - restore engine status from file where it was previously saved
  - its directory must have been previously set by /random/setDirectoryName

Random Number Generators in Geant4

- Previous commands use C++ methods which provide functionality. Some of these methods are listed here.
- Set the seeds
  - G4Random::setSeed(long seed, int);
  - G4Random::setSeeds(const long* seeds, int);
- Save engine status to file
  - G4Random::saveEngineStatus(const char filename[] =“Config.conf”);
- Restore engine status from a file
  - G4Random::restoreEngineStatus(const char filename[] = “Config.conf”);
- Display engine status
  - G4Random::showEngineStatus();

More Information on Generators

- Look in documentation and in base class header file:
  - geant4/source/externals/clhep/include/CLHEP/Random/RandomEngine.h

- Some of the above methods may be needed in the exercise
  - you may build file names with “std::ostringstream fileName;”
  - to get the “char*” do “fileName.str().c_str();”

Summary

- No such thing as a truly random number generator in simulation -> pseudo-random engines
  - many available, but take care

- In your first few uses of Geant4, no need to worry about random number generation
  - a default is supplied
- For more serious simulations, you will likely need to
  - set the seeds in order to run multiple, statistically independent jobs
  - choose your favorite random number engine
  - save and restore the random engine status
- Can be controlled with direct C++ coding or interactive commands