Lecture 5: Simulation in R, with applications

Simulation: what and why?

- Simulation uses computers to mimic behaviour of stochastic processes and statistical models.
- Some uses:
  - Explore properties of a model if it is hard to calculate these properties analytically. E.g. agent-based modelling to study interactions between players in complex systems (investors, internet users, animals, disease transmission . . .) etc.
  - Numerical approximation of integrals, optimisation, solution of differential equations, . . .
  - Testing your code e.g. generate artificial datasets where you know what the answer should be.
  - Designing experiments / trials e.g. randomly assigning treatments to patients
  - Check finite-sample performance of 'large-sample' statistical approximations
  - Approximate calculation of confidence intervals, standard errors etc. when large-sample approximations don't work

Pseudo-random numbers

- Independent random variables are building-blocks of all stochastic / statistical models
- **Problem**: how to get a computer to generate independent random variables?
  - Computers can only do what is programmed $\Rightarrow$ output can't be random!

Pseudo-random numbers

- Independent random variables are building-blocks of all stochastic / statistical models
- **Problem**: how to get a computer to generate independent random variables?
  - Computers can only do what is programmed $\Rightarrow$ output can't be random!
- **Solution**: find way to generate pseudo-random numbers i.e. sequences $\{x_1, x_2, \ldots,\}$ with properties that are indistinguishable for all practical purposes from those of an i.i.d sequence of random variables $X_1, X_2, \ldots$

Pseudo-random numbers

- Independent random variables are building-blocks of all stochastic / statistical models
- **Problem**: how to get a computer to generate independent random variables?
  - Computers can only do what is programmed $\Rightarrow$ output can't be random!
- **Solution**: find way to generate pseudo-random numbers i.e. sequences $\{x_1, x_2, \ldots,\}$ with properties that are indistinguishable for all practical purposes from those of an i.i.d sequence of random variables $X_1, X_2, \ldots$
  - How to measure 'indistinguishable for all practical purposes'?
  - 'Standard' battery of tests for modern pseudo-random number generators: the Diehard tests (see https://en.wikipedia.org/wiki/Diehard_tests)

Pseudo-random number generators: some essential facts

- An iid sequence of Uniform$(0,1)$ random variables can be transformed into a sequence from any other distribution $\Rightarrow$ all pseudo-random number generators (PRNGs) aim to produce $U(0,1)$ sequences

Pseudo-random number generators: some essential facts

- An iid sequence of Uniform$(0, 1)$ random variables can be transformed into a sequence from any other distribution $\Rightarrow$ all pseudo-random number generators (PRNGs) aim to produce $U(0, 1)$ sequences
- For all PRNGs, next value in sequence is function of most recent value(s), so:
  - Almost all PRNGs will repeat themselves after finite 'period'
  - Successive values are not independent: challenge is to make them appear so

Pseudo-random number generators: some essential facts

- An iid sequence of Uniform$(0,1)$ random variables can be transformed into a sequence from any other distribution $\Rightarrow$ all pseudo-random number generators (PRNGs) aim to produce $U(0,1)$ sequences
- For all PRNGs, next value in sequence is function of most recent value(s), so:
  - Almost all PRNGs will repeat themselves after finite 'period'
  - Successive values are not independent: challenge is to make them appear so
- Default PRNG in R has period $2^{19937} - 1 \approx 10^{6000}$, and sequences of up to 623 successive values have joint distributions that are 'independent' — see `help(Random)`.

Pseudo-random number generators: some essential facts

- An iid sequence of Uniform$(0, 1)$ random variables can be transformed into a sequence from any other distribution $\Rightarrow$ all pseudo-random number generators (PRNGs) aim to produce $U(0, 1)$ sequences
- For all PRNGs, next value in sequence is function of most recent value(s), so:
  - Almost all PRNGs will repeat themselves after finite 'period'
  - Successive values are not independent: challenge is to make them appear so
- Default PRNG in R has period $2^{19937} - 1 \approx 10^{6000}$, and sequences of up to 623 successive values have joint distributions that are 'independent' — see `help(Random)`.
- Starting point in sequence determined by setting a seed:
  - `set.seed()` in R — uses system clock by default
  - Enables 'random' sequences to be reproduced exactly later on — useful for debugging, further work etc.
  - **NB** seed is *not* position in sequence! — used to *calculate* position

Given $U \sim U(0,1)$...

- **Bernoulli:** Set $X = 1$ if $U \leq p$, 0 otherwise; then $X \sim Ber(p)$

Today $U(0,1)$, tomorrow the Universe ...

Given $U \sim U(0,1)$...

- **Bernoulli:** Set $X = 1$ if $U \leq p$, 0 otherwise; then $X \sim Ber(p)$.
- **Binomial:** generate independent $Ber(p)$ values $X_1, X_2, \ldots, X_n$ and set $Y = \sum_{i=1}^{n} X_i$; then $Y \sim Bin(n,p)$.

Today $U(0,1)$, tomorrow the Universe ...

Given $U \sim U(0,1) \ldots$

- **Bernoulli:** Set $X = 1$ if $U \leq p$, 0 otherwise; then $X \sim Ber(p)$
- **Binomial:** generate independent $Ber(p)$ values $X_1, X_2, \ldots, X_n$ and set $Y = \sum_{i=1}^{n} X_i$; then $Y \sim Bin(n,p)$.
- **Distribution with CDF $F(\cdot)$:** set $X = F^{-1}(U)$, then X has distribution function $F(\cdot)$ ("inversion method")

UCL

Today $U(0,1)$, tomorrow the Universe ...

Given $U \sim U(0,1)\ldots$

- **Bernoulli:** Set $X = 1$ if $U \leq p$, 0 otherwise; then $X \sim Ber(p)$
- **Binomial:** generate independent $Ber(p)$ values $X_1, X_2, \ldots, X_n$ and set $Y = \sum_{i=1}^{n} X_i$; then $Y \sim Bin(n,p)$.
- **Distribution with CDF $F(\cdot)$:** set $X = F^{-1}(U)$, then X has distribution function $F(\cdot)$ ("inversion method")

Proof: $P(X < x) = P(F^{-1}(U) < x) = P(U < F(x)) = F(x)$, since $P(U < u) = u$ for $u \in [0,1]$.

Given $U \sim U(0,1)\ldots$

- **Bernoulli:** Set $X = 1$ if $U \leq p$, 0 otherwise; then $X \sim Ber(p)$
- **Binomial:** generate independent $Ber(p)$ values $X_1, X_2, \ldots, X_n$ and set $Y = \sum_{i=1}^{n} X_i$; then $Y \sim Bin(n,p)$.
- **Distribution with CDF $F(\cdot)$:** set $X = F^{-1}(U)$, then X has distribution function $F(\cdot)$ ("inversion method")

Proof: $P(X < x) = P(F^{-1}(U) < x) = P(U < F(x)) = F(x)$, since $P(U < u) = u$ for $u \in [0,1]$.

- **Exponential:** set $X = -\lambda^{-1} \log U$, then $X \sim Exp(\lambda)$ (application of inversion method)

Today $U(0,1)$, tomorrow the Universe ...

Given $U \sim U(0,1)\ldots$

- **Bernoulli:** Set $X = 1$ if $U \leq p$, 0 otherwise; then $X \sim Ber(p)$
- **Binomial:** generate independent $Ber(p)$ values $X_1, X_2, \ldots, X_n$ and set $Y = \sum_{i=1}^{n} X_i$; then $Y \sim Bin(n,p)$.
- **Distribution with CDF $F(\cdot)$:** set $X = F^{-1}(U)$, then X has distribution function $F(\cdot)$ ("inversion method")

Proof: $P(X < x) = P(F^{-1}(U) < x) = P(U < F(x)) = F(x)$, since $P(U < u) = u$ for $u \in [0,1]$.

- **Exponential:** set $X = -\lambda^{-1} \log U$, then $X \sim Exp(\lambda)$ (application of inversion method)
- **Normal:** generate $R \sim Exp(1/2)$ and $\theta = 2\pi U$ independently, then $X = \sqrt{R}\sin(\theta)$ and $Y = \sqrt{R}\cos(\theta)$ are independent $N(0,1)$ random variables (Box-Muller algorithm)

Given $U \sim U(0,1)$…

- **Bernoulli:** Set $X = 1$ if $U \leq p$, 0 otherwise; then $X \sim Ber(p)$
- **Binomial:** generate independent $Ber(p)$ values $X_1, X_2, \ldots, X_n$ and set $Y = \sum_{i=1}^{n} X_i$; then $Y \sim Bin(n,p)$.
- **Distribution with CDF $F(\cdot)$:** set $X = F^{-1}(U)$, then X has distribution function $F(\cdot)$ ("inversion method")

Proof: $P(X < x) = P(F^{-1}(U) < x) = P(U < F(x)) = F(x)$, since $P(U < u) = u$ for $u \in [0,1]$.

- **Exponential:** set $X = -\lambda^{-1} \log U$, then $X \sim Exp(\lambda)$ (application of inversion method)
- **Normal:** generate $R \sim Exp(1/2)$ and $\theta = 2\pi U$ independently, then $X = \sqrt{R}\sin(\theta)$ and $Y = \sqrt{R}\cos(\theta)$ are independent $N(0,1)$ random variables (Box-Muller algorithm)

Other methods in workshop …

UCL

- Functions are `runif()`, `rnorm()`, `rbinom()`, `rpois()`, `rgamma()`, `rgeom()`, ... (very long list!)
- **Example:** 10 pseudo-random numbers from $U(0,1)$

```
> runif(10)
 [1] 0.61600894 0.06056044 0.32564408 0.64546838 0.11969141
 [6] 0.29347111 0.91849457 0.44402218 0.84813316 0.49687587
```

- For different parameters use, e.g., `runif(10,min=0,max=100)`

Pseudo-random number generation in R

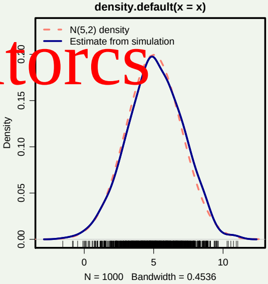
- Functions are `runif()`, `rnorm()`, `rbinom()`, `rpois()`, `rgamma()`, `rgeom()`, ... (very long list!)
- **Example:** 10 pseudo-random numbers from $U(0,1)$

```
> runif(10)
 [1] 0.61600894 0.06056044 0.32564408 0.64546838 0.11969141
 [6] 0.29347111 0.91849457 0.44402218 0.84813316 0.49687587
```

- For different parameters use, e.g., `runif(10,min=0,max=100)`

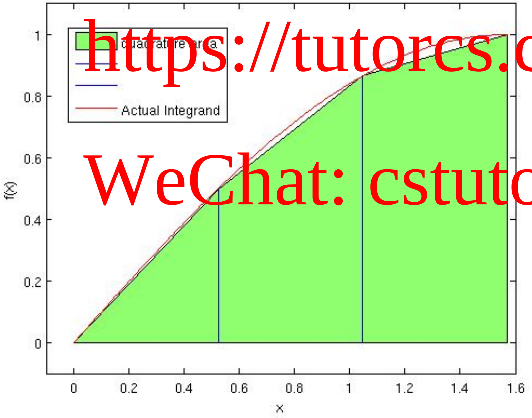
Another example: simulating from $N(5, 2^2)$

```
> x <- rnorm(1000,mean=5,sd=2)
> plot(density(x),col="darkblue")
> rug(x)
> fx <- function(x) dnorm(x,5,2)
> curve(fx, -10, 15, lty=2,
+           add=TRUE, col="salmon")
```



- Recall trapezium rule for quadrature, from Workshop 3:

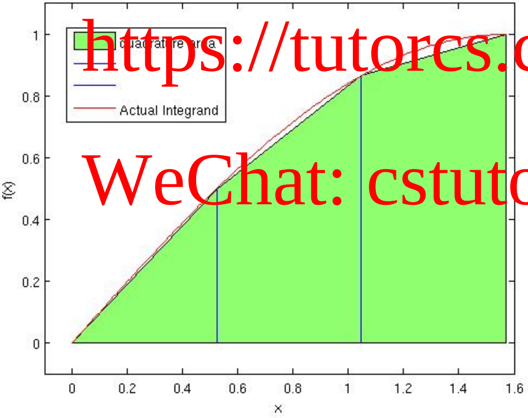
$$\int_{x_0}^{x_n} f(x)dx \approx \frac{h}{2}\left[f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)\right]$$



- Can show that doubling the number of evaluation points reduces approximation error by factor of four $\Rightarrow$ error is of order $O(n^{-2})$.

- Recall trapezium rule for quadrature, from Workshop 3:

$$\int_{x_0}^{x_n} f(x)dx \approx \frac{h}{2}\left[f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)\right]$$



- Can show that doubling the number of evaluation points reduces approximation error by factor of four $\Rightarrow$ error is of order $\mathcal{O}(n^{-2})$.
- What about higher-dimensional integrals? Perhaps use lattice of points in place of grid $x_0, x_1, \ldots, x_n$.

Multidimensional integrals

Suppose we want to evaluate $\int_{a_1}^{b_1} \ldots \int_{a_p}^{b_p} f(x_1, \ldots, x_p)\, dx_1 \ldots dx_p$ using the trapezium rule or similar

The curse of dimensionality

Multidimensional integrals

Suppose we want to evaluate $\int_{a_1}^{b_1} \dots \int_{a_p}^{b_p} f(x_1, \dots, x_p)\, dx_1 \dots dx_p$ using the trapezium rule or similar

- $p = 1$: for a 1-D interval $\int_a^b$, use grid of n evaluation points.
- $p = 2$: For a 2-D rectangle, e.g. $\int_{a_1}^{b_1} \int_{a_2}^{b_2}$: n points for the first variable and n points for the second $\Rightarrow$ n^2 evaluation points.
- $p = 3$: For a 3-D box: $\int_{a_1}^{b_1} \int_{a_2}^{b_2} \int_{a_3}^{b_3}$ needs n^3 evaluation points

The curse of dimensionality

Multidimensional integrals

Suppose we want to evaluate $\int_{a_1}^{b_1} \dots \int_{a_p}^{b_p} f(x_1, \dots, x_p)\, dx_1 \dots dx_p$ using the trapezium rule or similar

- $p = 1$: for a 1-D interval $\int_a^b$, use grid of n evaluation points.
- $p = 2$: For a 2-D rectangle, e.g. $\int_{a_1}^{b_1} \int_{a_2}^{b_2}$: n points for the first variable and n points for the second $\Rightarrow$ n^2 evaluation points.
- $p = 3$: For a 3-D box: $\int_{a_1}^{b_1} \int_{a_2}^{b_2} \int_{a_3}^{b_3}$ needs n^3 evaluation points

Multidimensional integrals

Quadrature methods for p-dimensional integrals require on the order of n^p evaluations: not usually feasible for p bigger than 5 or 6 (100^6 is 10^{12} evaluations ...)

- Suppose we want to evaluate an integral where integrand can be written as $\phi(x)g(x)$, where $g(x)$ is a pdf from which we can simulate

Monte Carlo Integration

- Suppose we want to evaluate an integral where integrand can be written as $\phi(x)g(x)$, where $g(x)$ is a pdf from which we can simulate
- Then we want $\int_{\mathbb{R}} \phi(x)g(x)dx$ which is $\mathbb{E}[\phi(X)] = \theta$ say, where X is a random variable with pdf $g(\cdot)$.

- Suppose we want to evaluate an integral where integrand can be written as $\phi(x)g(x)$, where $g(x)$ is a pdf from which we can simulate
- Then we want $\int_{\mathbb{R}} \phi(x)g(x)dx$ which is $\mathbb{E}[\phi(X)] = \theta$ say, where X is a random variable with pdf $g(\cdot)$.

- Estimate by drawing iid samples from X and using sample mean: $\widehat{\theta} = N^{-1} \sum_{i=1}^{N} \phi(X_i)$

Monte Carlo Integration

- Suppose we want to evaluate an integral where integrand can be written as $\phi(x)g(x)$, where $g(x)$ is a pdf from which we can simulate
- Then we want $\int_{\mathbb{R}} \phi(x)g(x)dx$ which is $\mathbb{E}[\phi(X)] = \theta$ say, where X is a random variable with pdf $g(\cdot)$.
- Estimate by drawing iid samples from X and using sample mean: $\widehat{\theta} = N^{-1}\sum_{i=1}^{N} \phi(X_i)$
- Approximation error quantified by standard error of $\widehat{\theta}$ as estimator of θ
  - Can be more accurate than quadrature methods in high dimensions, for same computational cost
  - Clever sampling strategies can be used to reduce standard errors and improve precision (see workshop)

A nasty integral?

$$\int_0^\infty \int_0^\infty \int_0^\infty \int_0^\infty \cos(x_1 - 2x_2 + 3x_3 - 4x_4)\exp\left[-\sum_{i=1}^{4} x_i\right] dx_1 \dots dx_4$$

Can use integration by parts ...

Monte Carlo integration: example

A nasty integral?

$$\int_0^\infty \int_0^\infty \int_0^\infty \int_0^\infty \cos(x_1 - 2x_2 + 3x_3 - 4x_4) \exp\left[-\sum_{i=1}^{4} x_i\right] dx_1 \dots dx_4$$

Can use integration by parts ... but life is short!

- **Note** that $g(x) = \exp\left[-\sum_{i=1}^{4} x_i\right]$ $(x_i \in \mathbb{R}^+ : i = 1, \dots, 4)$ is joint density of four iid $Exp(1)$ random variables: $X_1, \dots, X_4$, say.
- So integral is
  $\mathbb{E}\left[\cos\left(X_1 - 2X_2 + 3X_3 - 4X_4\right)\right] = \mathbb{E}\left[\phi(X_1, X_2, X_3, X_4)\right]$, say.

Example continued: the integral in half a second

Reminder:

Integral is $\mathbb{E}\left[\cos\left(X_1 - 2X_2 + 3X_3 - 4X_4\right)\right]$ where $\{X_i\}$ are iid $Exp(1)$


```
n <- 10^6      # Sample size
#
# Simulate Xs and store in matrix: column 1 is X1 etc. Then
#   use matrix arithmetic for linear combination - it's fast
#
X.matrix <- matrix(rexp(4*n, rate=1), ncol=4)
phi <- cos(X.matrix %*% c(1,-2,3,-4))
mean(phi)
[1] 0.02157164
sd(phi) / sqrt(n)      #   Standard error of the mean
[1] 0.0007074219
```


NB 10^6 evaluation points here — but 10^8 needed for quadrature in 4 dimensions with 100 points in each dimension

Simulating system behaviour

- **Another use for simulation**: study complex systems when analytical (mathematical) treatment is too hard (or boring)

Example:

- S_t is the price of a stock at time t. It develops in time as a geometric Brownian Motion with parameters μ and σ^2:

$$S_t = \exp\left((\mu - \frac{1}{2}\sigma^2)t + \sigma B_t \right)$$

$$B_t = \sum_{j=1}^{t} \varepsilon_j \quad \text{where} \quad \varepsilon_j \sim \mathcal{N}(0,1) \quad \Rightarrow \quad B_t \sim \mathcal{N}(0,t)$$

- Some stocks are bought at time 0, and sold again at a random time T where $T \sim \Gamma(2,3)$. What is the expected stock price at the time of sale?

Example: estimating stock price using simulation

- Suppose $\mu = 0.5$, $\sigma^2 = 0.01$


```
> nsims <- 10000                 # Number of simulations
> mu <- 0.5; sigsq <- 0.01       # Define parameter values
> T <- rgamma(nsims,2,3)         # Sale times
> Bt <- rnorm(nsims,mean=0,sd=sqrt(T)) # Values of B[T]
> SalePrice <- exp( (mu-0.5*sigsq)*T+(sqrt(sigsq)*Bt) )
> mean(SalePrice)            # Estimate of expected price
[1] 1.447391
> sd(SalePrice) / sqrt(nsims)  # Standard error
[1] 0.004663135
```


Caveat:

It is often better to solve this kind of problem analytically if possible, because it provides more insight (e.g. formulae relating sale price to μ and σ^2). Option pricing etc. is covered in STAT0013 *Stochastic Methods in Finance 1*.

And now what you've all been waiting for . . .

In-course assessment 1

In-course assessment information is on the IN-COURSE ASSESSMENT 1 tab on the STAT0023 Moodle page