Random numbers in AD Model Builder

AD Model Builder foundation

anders@nielsensweb.org

Random numbers are useful for

- Simulating systems too complex to handle analytically
- Solving stochastic differential equations
- Numerical integration (dim $>$ n)
- Markov Chain Monte Carlo methods
- Encryption
- Bootstrap methods
- Design of experiments
- ...

How do we get random numbers

1. Use a natural phenomena: Geiger counter, dice, coin, or a deck of cards
2. Table: You can fit 125 Million random numbers on a CD
3. Computer algorithm: Easy, but the algorithm should be carefully chosen
   - Should be fast
   - Should be easy to use
   - Possible to reproduce a sequence
   - Numbers should be uncorrelated and follower the desired distribution
   - The period should be long

A high quality random number generator is build into AD Model Builder.

How to use

```
DATA_SECTION
 LOC_CALCS
  random_number_generator rng(123456);
  dvector sample(1,5);

  sample.fill_randu(rng);
  cout<<"Uniform(0,1): "<<sample<<endl;

  sample.fill_randn(rng);
  cout<<"Normal(0,1): "<<sample<<endl;

  sample.fill_randpoisson(1.5,rng);
  cout<<"pois(1.5): "<<sample<<endl;

  sample.fill_randnegbinomial(1.5,2.0,rng);
  cout<<"neg.bin(1.5,2): "<<sample<<endl;

  sample.fill_randcau(rng);
  cout<<"Cauchy: "<<sample<<endl;

  sample.fill_randbi(0.8,rng);
  cout<<"binomial(n=1,p=0.8): "<<sample<<endl;

  dvector p("{.01,.495,.495}");
  sample.fill_multinomial(rng,p);
  cout<<"multinomial(n=1,p=(.01,.495,.495)): "
      <<sample<<endl;

  ad_exit(0);
 END_CALCS
PARAMETER_SECTION
  objective_function_value nll;

PROCEDURE_SECTION
```

```
Uniform(0,1):  0.779837 0.229835 0.0126429
               0.714228 0.654815

Normal(0,1):  -0.325127 1.03682 0.567672
              -0.670345 2.89024

pois(1.5):  2 2 0 1 2

neg.bin(1.5,2):  0 3 1 0 4

Cauchy:  -0.188267 -1.30511 -9.11156
         29.8652 2.83259

binomial(n=1,p=0.8):  1 1 0 0 1

multinomial(n=1,p=(.01,.495,.495)):  3 2 3 2 3
```

Testing a model

- Verifying an implementation
  1. Choose some realistic parameters
  2. Generate a data set from the true model
  3. Run the model
  4. Evaluate if the estimates are close enough
  5. Repeat 1–4 a few times with different seeds
- AD Model Builder makes these steps very simple.
- To do an actual simulation study (hundreds of data sets) gets a little more tricky if we want to use only AD Model Builder — but we will show how.

The model and the real data

- The following non-linear model is assumed to describe the relation between density D within pot and yield Y per plant:

$$\log(Y_i) = -\log(\alpha + \beta D_i) + \varepsilon_i, \quad \text{where } \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

```
DATA_SECTION
  init_int N
  init_vector density(1,N)
  init_vector yield(1,N)
  vector logYield(1,N)
  !! logYield=log(yield);
PARAMETER_SECTION
  init_number logA
  init_number logB
  init_number logSigma
  objective_function_value nll
  sdreport_number a
  sdreport_number b
  sdreport_number sigma
  vector pred(1,N)
  number ss
PROCEDURE_SECTION
  b=exp(logB);
  a=exp(logA)-b*min(density);
  sigma=exp(logSigma);
  ss=square(sigma);
  pred=-log(a+b*density);
  nll=0.5*(N*log(2*M_PI*ss)+
   sum(square(logYield-pred))/ss);
```

```
#N
10
#density
5 7 10 15 25 34 51 77 115 173
#yield
6.97 5.569 2.814 2.401 1.89 1.124 0.623 0.592 0.382 0.204
```

```
index   name      value       std dev
    1   logA     -1.9044e+00 1.0966e-01
    2   logB     -3.6793e+00 6.7797e-02
    3   logSigma -1.9246e+00 2.2361e-01
    4   a         2.2708e-02 2.1107e-02
    5   b         2.5241e-02 1.7113e-03
    6   sigma     1.4594e-01 3.2633e-02
```

Testing implementation with simulated data

```
DATA_SECTION
  init_int N
  init_vector density(1,N)
  init_vector yield(1,N)
  vector logYield(1,N)

  // Replace with simulated data
  !! random_number_generator rng(123456);
  !! logYield.fill_randn(rng);
  !! logYield*= 0.15; // assumed sd
  !! logYield+= -log(.022+.025*density);

PARAMETER_SECTION
  init_number logA
  init_number logB
  init_number logSigma
  objective_function_value nll
  sdreport_number a
  sdreport_number b
  sdreport_number sigma
  vector pred(1,N)
  number ss
PROCEDURE_SECTION
  b=exp(logB);
  a=exp(logA)-b*min(density);
  sigma=exp(logSigma);
  ss=square(sigma);
  pred=-log(a+b*density);
  nll=0.5*(N*log(2*M_PI*ss)+
         sum(square(logYield-pred))/ss);
```

```
index   name       value       std dev
    1   logA     -1.8823e+00 8.1064e-02
    2   logB     -3.6725e+00 5.0495e-02
    3   logSigma -2.2184e+00 2.2361e-01
    4   a         2.5178e-02 1.5895e-02
    5   b         2.5412e-02 1.2832e-03
    6   sigma     1.0878e-01 2.4324e-02
```

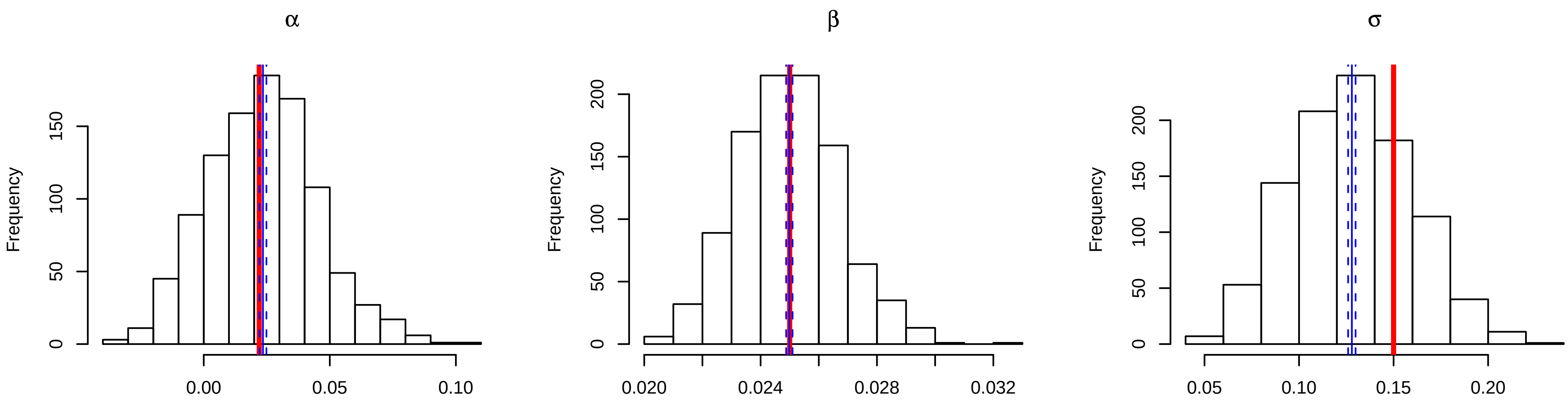
Setting up a full simulation study

```
DATA_SECTION
  init_ivector sim(1,3);
  int orgseed;
 LOC_CALCS
  random_number_generator rng(sim(1));
  orgseed=sim(1);
  for(int i=sim(2); i<=sim(3); ++i){
    int rv=system("./nlsim");
    sim(2)++;
    sim(1)=(int)round(1.0e9*randu(rng));
    ofstream datout("sim.dat");
    datout<<sim<<endl;
  }
  cout<<"All done"<<endl;
  sim(1)=orgseed;
  sim(2)=1;
  ofstream datout("sim.dat");
  datout<<sim<<endl;
  ad_exit(0);
 END_CALCS

PARAMETER_SECTION
  objective_function_value nll;

PROCEDURE_SECTION
```

```
123456 1 1000
```

```
DATA_SECTION
  init_int N
  init_vector density(1,N)
  init_vector yield(1,N)
  vector logYield(1,N)

  // Replace with simulated data
  !! ad_comm::change_datafile_name("sim.dat");
  init_ivector sim(1,3);
  !! random_number_generator rng(sim(1));
  !! logYield.fill_randn(rng);
  !! logYield*= 0.15; // assumed sd
  !! logYield+= -log(.022+.025*density);

PARAMETER_SECTION

  ... this section is identical to previous slide

PROCEDURE_SECTION

  ... this section is identical to previous slide

REPORT_SECTION
  ofstream simout;
  if(sim(2)==1){
    simout.open("sim.out");
  }else{
    simout.open("sim.out", ios::app);
  }
  simout<<a<<" "<<b<<" "<<sigma<<endl;
```

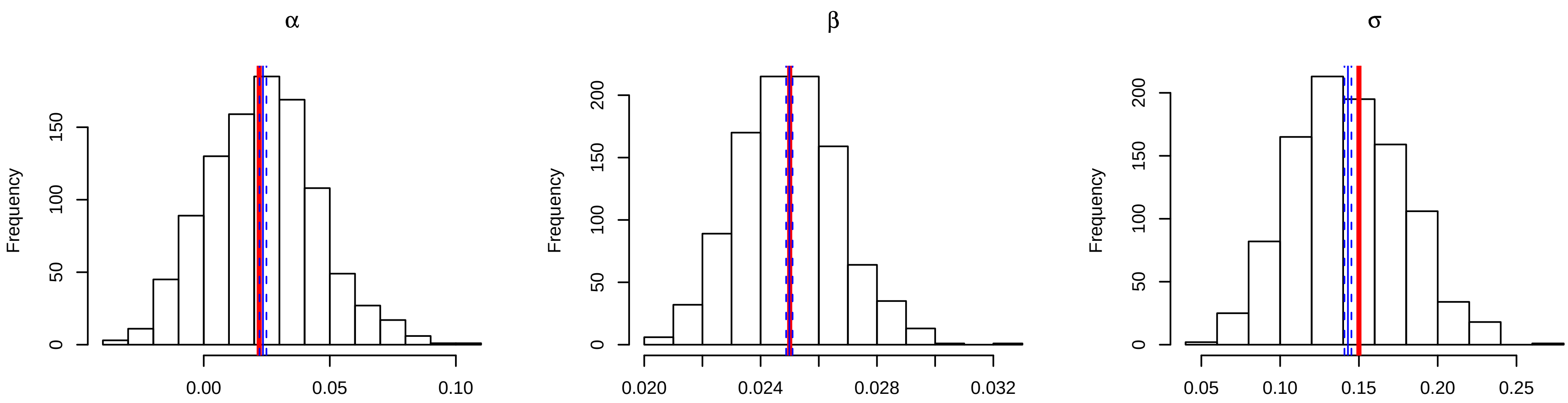
Results of simulations



Parameter	True value	low	high
α	0.022	0.022	0.025
β	0.025	0.0249	0.0251
σ	0.150	0.126	0.13

Why is this an expected result? Can we fix it?

Results with random effects (flat prior) on α and β



Parameter	True value	low	high
α	0.022	0.022	0.025
β	0.025	0.0249	0.0251
σ	0.150	0.141	0.145

Almost!

Exercises

Exercise 1: Pick a model from one of the previous modules and test it via simulation.