Simulation of data using Julia

Rohan L. Fernando

May 2015

Julia Packages

- List of registered Julia packages (http://docs.julialang.org/en/release-0.1/packages/packagelist/#available-packages)
- Will use Distributions Package (http://distributionsjl.readthedocs.org/en) to simulate data.
- It can be added to your system with the command:

```
In [1]: Pkg.add("Distributions")

        INFO: Nothing to be done
        INFO: METADATA is out-of-date — you may not have the latest version of Distributions
        INFO: Use `Pkg.update()` to get the latest versions of your packages
```

- This needs to be done only once.
- But, to access the functions in the Distributions package the "using" command has to be invoked as:

```
In [2]: using Distributions
```

Simulate matrix of ``genotype" covariates

In [3]:
```
nRows = 10
nCols = 5
X = sample([0,1,2],(nRows,nCols))
```

Out[3]:
```
10x5 Array{Int64,2}:
 0  0  2  1  2
 1  1  1  1  0
 0  2  1  1  0
 1  2  1  0  2
 0  1  1  0  2
 2  1  0  0  2
 2  2  2  0  2
 1  2  2  2  1
 0  0  2  2  1
 2  2  0  2  0
```

Each element in $\mathbf{X}$ is sampled from the array [0,1,2].

Other methods of the function ``sample"

In [4]:
```
methods(sample)
```

Out[4]: 7 methods for generic function **sample**:

- sample(a::**AbstractArray{T,N}**) at /Users/rohan/.julia/v0.3/StatsBase/src/sampling.jl:277 (https://github.com/JuliaStats/StatsBase.jl/tree/23b36af460cfd6c147efef9074a76a4e8cf13dae/src/
- sample{T}(a::**AbstractArray{T,N}**,n::**Integer**) at /Users/rohan/.julia/v0.3/StatsBase/src/sampling.jl (https://github.com/JuliaStats/StatsBase.jl/tree/23b36af460cfd6c147efef9074a76a4e8cf13dae/src/
- sample{T}(a::**AbstractArray{T,N}**,dims::**(Int64...,)**) at /Users/rohan/.julia/v0.3/StatsBase/src/sampl (https://github.com/JuliaStats/StatsBase.jl/tree/23b36af460cfd6c147efef9074a76a4e8cf13dae/src/
- sample(wv::**WeightVec{W,Vec<:AbstractArray{T<:Real,1}}**) at /Users/rohan/.julia/v0.3/StatsBase/sr (https://github.com/JuliaStats/StatsBase.jl/tree/23b36af460cfd6c147efef9074a76a4e8cf13dae/src/
- sample(a::**AbstractArray{T,N}**,wv::**WeightVec{W,Vec<:AbstractArray{T<:Real,1}}**) at /Users/rohan/.julia/v0.3/StatsBase/src/sampling.jl:347 (https://github.com/JuliaStats/StatsBase.jl/tree/23b36af460cfd6c147efef9074a76a4e8cf13dae/src/
- sample{T}(a::**AbstractArray{T,N}**,wv::**WeightVec{W,Vec<:AbstractArray{T<:Real,1}}**,n::**Integer**) /Users/rohan/.julia/v0.3/StatsBase/src/sampling.jl:529 (https://github.com/JuliaStats/StatsBase.jl/tree/23b36af460cfd6c147efef9074a76a4e8cf13dae/src/
- sample{T}(a::**AbstractArray{T,N}**,wv::**WeightVec{W,Vec<:AbstractArray{T<:Real,1}}**,dims::**(Int6** /Users/rohan/.julia/v0.3/StatsBase/src/sampling.jl:532 (https://github.com/JuliaStats/StatsBase.jl/tree/23b36af460cfd6c147efef9074a76a4e8cf13dae/src/

Column of ones for intercept

```
In [5]: X = [ones(nRows,1) X]
```

```
Out[5]: 10x6 Array{Float64,2}:
 1.0  0.0  0.0  2.0  1.0  2.0
 1.0  1.0  1.0  1.0  1.0  0.0
 1.0  0.0  2.0  1.0  1.0  0.0
 1.0  1.0  2.0  1.0  0.0  2.0
 1.0  0.0  1.0  1.0  0.0  2.0
 1.0  2.0  1.0  0.0  0.0  2.0
 1.0  2.0  2.0  2.0  0.0  2.0
 1.0  1.0  2.0  2.0  2.0  1.0
 1.0  0.0  0.0  2.0  2.0  1.0
 1.0  2.0  2.0  0.0  2.0  0.0
```

Simulate effects from normal distribution

```
In [6]: nRowsX, nColsX = size(X)
        mean = 0.0
        std  = 0.5
        b = rand(Normal(mean,std),nColsX)
```

```
Out[6]: 6-element Array{Float64,1}:
 -0.34724
  0.0406174
 -0.316707
  0.233593
  0.0933254
  0.277288
```

Simulate phenotypic values

```
In [7]: resStd = 1.0
        y = X*b + rand(Normal(0,resStd),nRowsX)
```

```
Out[7]: 10-element Array{Float64,1}:
         -0.0880872
         -1.17895
         -2.80082
          2.08141
          0.371737
         -0.358808
          0.0203133
         -1.26218
          0.317851
         -1.2807
```

Function to simulate data

```
In [8]: using Distributions
        function simDat(nObs,nLoci,bMean,bStd,resStd)
            X = [ones(nObs,1) sample([0,1,2],(nObs,nLoci))]
            b = rand(Normal(bMean,bStd),size(X,2))
            y = X*b + rand(Normal(0.0, resStd),nObs)
            return (y,X)
        end
        nObs    = 10
        nLoci   = 5
        bMean   = 0.0
        bStd    = 0.5
        resStd  = 1.0
        res = simDat(nObs,nLoci,bMean,bStd,resStd)
        y = res[1]
        X = res[2]
```

```
Out[8]: 10x6 Array{Float64,2}:
         1.0  1.0  0.0  2.0  2.0  1.0
         1.0  2.0  0.0  2.0  0.0  2.0
         1.0  2.0  1.0  0.0  0.0  0.0
         1.0  0.0  1.0  1.0  1.0  1.0
         1.0  2.0  1.0  1.0  2.0  1.0
         1.0  2.0  1.0  1.0  0.0  0.0
         1.0  1.0  1.0  0.0  0.0  0.0
         1.0  1.0  2.0  2.0  0.0  0.0
         1.0  0.0  0.0  2.0  0.0  0.0
         1.0  1.0  2.0  0.0  0.0  2.0
```

XSim: Genome sampler

- Simulate SNPs on chromosomes
- Random mating in finite population to generate LD
- Efficient algorithm for sampling sequence data

Install XSim

```
In [9]: # installing package
        # only needs to be done once
        Pkg.clone("https://github.com/reworkhow/XSim.jl.git")

        INFO: Cloning XSim from https://github.com/reworkhow/XSim.jl.git

        XSim already exists
        while loading In[9], in expression starting on line 3

         in error at error.jl:21
         in clone at pkg/entry.jl:148
         in clone at pkg/entry.jl:175
         in anonymous at pkg/dir.jl:28
         in cd at /Applications/Julia-0.3.7.app/Contents/Resources/julia/lib/julia/sys.dylib
         in __cd#228__ at /Applications/Julia-0.3.7.app/Contents/Resources/julia/lib/julia/sy
        s.dylib
         in clone at pkg.jl:30
```

Initialize sampler

In [10]:
```
using XSim
chrLength = 1.0
numChr    = 1
numLoci   = 2000
mutRate   = 0.0
locusInt  = chrLength/numLoci
mapPos    = [0:locusInt:(chrLength-0.0001)]
geneFreq = fill(0.5,numLoci)
XSim.init(numChr,numLoci,chrLength,geneFreq,mapPos,mutRate)
```

Simulate random mating in finite population

In [14]:
```
pop     = startPop()
nGen    = 10
popSize = 500
pop.popSample(nGen,popSize)
```

```
Sampling 500 animals into base population.
Sampling 500 animals into generation: 1
Sampling 500 animals into generation: 2
Sampling 500 animals into generation: 3
Sampling 500 animals into generation: 4
Sampling 500 animals into generation: 5
Sampling 500 animals into generation: 6
Sampling 500 animals into generation: 7
Sampling 500 animals into generation: 8
Sampling 500 animals into generation: 9
```

Julia & IJulia Cheat-sheet (for 18.xxx at MIT)

Basics:

`julialang.org` documentation

`github.com/stevengj/julia-mit` installation & tutorial

`ipython notebook --profile-julia` start IJulia browser

shift-return execute input cell in IJulia

Defining/changing variables:

`x = 3` define variable x to be 3

`x = [1,2,3]` array/"column"-vector (1,2,3)

`y = [1 2 3]` 1×3 row-vector (1,2,3)

```
A = [1 2 3 4; 5 6 7 8; 9 10 11 12]
```

—set A to 3×4 matrix with rows 1,2,3,4 etc.

`x[2] = 7` change x from (1,2,3) to (1,7,3)

`A[2,1] = 0` change $A_{2,1}$ from 5 to 0

`u, v = (15.03, 1.2e-27)` set u=15.03, v=1.2×10^{-27}

`f(x) = 3x` define a function f(x)

`x -> 3x` an "anonymous" function

Constructing a few simple matrices:

`rand(12), rand(12,4)` random length-12 vector or 12×4 matrix with uniform random numbers in [0,1)

`randn(12)` Gaussian random numbers (mean 0, std. dev. 1)

`eye(5)` 5×5 identity matrix I

`linspace(1.2,4.7,100)` 100 equally spaced points from 1.2 to 4.7

`diagm(x)` matrix whose diagonal is the entries of x

Portions of matrices and vectors:

`x[2:12]` the 2nd to 12th elements of x

`x[2:end]` the 2nd to the last elements of x

`A[5,1:3]` row vector of 1st 3 elements in 5th row of A

`A[5,:]` row vector of 5th row of A

`diag(A)` vector of diagonals of A

Arithmetic and functions of numbers:

`3*4, 7+4, 2-6, 8/3` mult., add, sub., divide numbers

`3^7, 3^(8+2im)` compute 3^7 or 3^{8+2i} power

`sqrt(-5+0im)` $\sqrt{-5}$ as a complex number

`exp(12)` e^{12}

`log(3), log10(100)` natural log (ln), base-10 log ($\log_{10}$)

`abs(-5), abs(2+3im)` absolute value |–5| or |2+3i|

`sin(5pi/3)` compute sin(5π/3)

`besselj(2,6)` compute Bessel function $J_2(6)$

Arithmetic and functions of vectors and matrices:

`x * 3, x + 3` multiply/add every element of x by 3

`x + y` element-wise addition of two vectors x and y

`A*y, A*B` product of matrix A and vector y or matrix B

`x * y` not defined for two vectors!

`x .* y` element-wise product of vectors x and y

`x .^ 3` every element of x is cubed

`cos(x), cos(A)` cosine of every element of x or A

`exp(A), expm(A)` exp of each element of A, matrix exp e^A

`x', A'` conjugate-transpose of vector or matrix

`x'*y, dot(x,y), sum(conj(x).*y)` three ways to compute $x \cdot y$

`A \ b, inv(A)` return solution to Ax=b, or the matrix A–1

`λ, V = eig(A)` eigenvals λ and eigenvectors (columns of V) of A

Plotting (type `using PyPlot` first)

`plot(y), plot(x,y)` plot y vs. 0,1,2,3,... or versus x

`loglog(x,y), semilogx(x,y), semilogy(x,y)` log-scale plots

`title("A title"), xlabel("x-axis"), ylabel("foo")` set labels

`legend(["curve 1", "curve 2"], "northwest")` legend at upper-left

`grid(), axis("equal")` add grid lines, use equal x and y scaling

`title(L"the curve $e^\sqrt{x}$")` title with LaTeX equation

`savefig("fig.png"), savefig("fig.eps")` save as PNG or EPS image

The objective of this laboratory session is to gain familiarity with the mixed linear models that we will be using in the Bayesian analyses later in the course.

Exercise 1

The lecture notes introduced the equations for generalized least squares (GLS). The GLS equation(s) for the model we discussed in the lecture are

$$\hat{\mathbf{b}}^0 = \left(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X}\right)^{-}\left(\mathbf{X}'\mathbf{V}^{-1}\mathbf{y}\right), \text{ for } \mathbf{V} = \mathbf{ZGZ}' + \mathbf{R}.$$

These equations are useful as **V** is typically full rank, hut are not practical in many situations where **V** is large. In this example with just the mean fitted as the only fixed effect, the GLS equation will be a scalar form.

In order to form **V**, you will need to know **G** and **R**.

Create a small Hendersonian data set hy constructing a vector **y** of phenotypic observations (no more than 6 observations). Create a corresponding **X** matrix to represent the incidence matrix for the fixed effects. This matrix will have as many rows as there are observations in **y**, and as many columns as there are fixed effects in **b**. Use the minimum configuration for **X** which is a vector of 1's that would correspond to a model that included an overall mean. Other alternatives for **X** might he to include a vector of covariates (eg age of the animal at measurement) or a class variable such as a fixed effect for the sex of the measured animal, or covariates and classes.

Construct a **G** matrix that will be square and have order equal to the number of animals in the pedigree file. For ease of viewing, the order of **G** should not exceed 6. The **G** matrix is the variance-covariance matrix of the fitted random effects, such as the breeding values. In that case, **G** will be the product of the numerator relationship or **A** matrix, and the scale additive genetic variance. You could form **A** for some simple pedigree and assume a value of the additive genetic variance, or create a small pedigree, and use Julia to form A^{-1} directly and invert it to inspect **A**. Note that the pedigree might contain some animals that do not have observed phenotypes, so the length of **y** may be less than the order of **G**.

Construct an incidence matrix **Z**, that relates the observations in **y** to the corresponding breeding value in **u**. The matrix **Z** may be an identity matrix if all animals in the pedigree have a phenotypic record. More typically, **Z** has as many rows as there are records in **y**, and as many columns as there are animals in **u** (and therefore the **G** matrix).

Construct **R**, the variance-covariance matrix for the residual effects, which for independent and identically distributed residual effects will be an identity matrix of order equal to the length of **y**, multiplied by the scalar residual variance. Recall that

the heritability is the ratio of the genetic variance over the phenotypic variance, and the phenotypic variance in this model is the sum of the additive genetic and residual variances, so the values you assume will imply a particular heritability.

Lastly, construct **y** using MvNormal to produce the vectors u and e. Remember these vectors may be different lengths if some animals in the pedigree do not have observations.

Given defined values for all these vectors, matrices and constants, calculate the phenotypic variance-covariance matrix **V**, and then solve the GLS equations to obtain best linear unbiased estimates (BLUEs) of the fixed effects. Use the BLUEs to adjust the phenotypic records and form deviations, that you can then use to compute the best linear unbiased predictions (BLUP) of the random effects as a linear function of these deviations, as described below. Note that this form of obtaining BLUP works with a singular **G** matrix.

The equations to obtain BLUP estimates are

$$\hat{\mathbf{u}} = \mathbf{G}\mathbf{Z}'\mathbf{V}^{-1}\left(\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}^{0}\right).$$

Be sure to save all your steps so you can immediately repeat your calculations with a modified dataset or different parameters. Print out and inspect the results of all your calculations.

Exercise 2

Repeat the same exercise as above, but this time estimate the BLUEs and predict the BLUPs by setting up and solving the mixed model equations. The answers should be identical to those you obtained using GLS. The mixed model equations are shown below.

$$\begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z}+\mathbf{G}^{-1} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{b}}^{0} \\ \hat{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{y} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{y} \end{bmatrix}$$

Exercise 3

Obtain the variance of the estimated BLUP effects, and the prediction error variance. These values require elements of the inverse of the mixed model coefficient matrix. We will use the following notation

$$\begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z}+\mathbf{G}^{-1} \end{bmatrix} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix}$$