Biostat M280 HW2

Due 05/05

Julia Cheatsheet https://cheatsheets.quantecon.org/julia-cheatsheet.html (https://cheatsheets.quantecon.org/julia-cheatsheet.html)

Q1

1) Implement Kinship estimator.

First, we aim to optimize the computation of e_{ij}.

In [1]:
```
# variant 1: loops to check correctness (slow? yes)
function e1(X::Matrix{Float64})
    (n,m) = size(X)
    e = zeros(X)
    for i = 1:n, j = 1:n
        for k = 1:m
            e[i,j] += 0.25 * (X[i,k]*X[j,k]+(2-X[i,k])*(2-X[j,k]))
        end
    end
    return e
end

# variant 2: vectorized (fast? no, actually slow)
function e2(X::Matrix{Float64})
    (n,m) = size(X)
    e = zeros(X)
    X2 = (2 - X)
    for i = 1:n, j = 1:n
        xx = (X[i,:]' * X[j,:])[1]
        xx2 = (X2[i,:]' * X2[j,:])[1]
        e[i,j] = 0.25 .* (xx + xx2)
    end
    return e
end

# variant 3: matrix ops (very fast? yes)
function e3(X::Matrix{Float64})
    return 0.25 .* (X * X' + (2 - X) * (2 - X)')
end
```

Out[1]:
```
e3 (generic function with 1 method)
```

In [2]:
```
srand(1234)
X = rand(0.0:2.0, 10, 10)
e1(X)
string(sum(abs(e1(X) - e2(X)) .> 10e-5)) * " differences"
```

Out[2]: "0 differences"

In [3]:
```
string(sum(abs(e1(X) - e3(X)) .> 10e-5)) * " differences"
```

Out[3]: "0 differences"

In [4]:
```
using BenchmarkTools
srand(280)
X = rand(0.0:2.0, 100, 100)
@benchmark e1(X)
```

Out[4]:
```
BenchmarkTools.Trial:
  memory estimate:  78.20 KiB
  allocs estimate:  2
  --------------
  minimum time:     3.211 ms (0.00% GC)
  median time:      3.253 ms (0.00% GC)
  mean time:        3.291 ms (0.10% GC)
  maximum time:     5.895 ms (0.00% GC)
  --------------
  samples:          1513
  evals/sample:     1
```

In [5]:
```
using BenchmarkTools
srand(280)
X = rand(0.0:2.0, 100, 100)
@benchmark e2(X)
```

Out[5]:
```
BenchmarkTools.Trial:
  memory estimate:  36.47 MiB
  allocs estimate:  80004
  --------------
  minimum time:     12.080 ms (12.84% GC)
  median time:      15.555 ms (22.05% GC)
  mean time:        17.161 ms (21.02% GC)
  maximum time:     93.667 ms (85.58% GC)
  --------------
  samples:          291
  evals/sample:     1
```

In [6]:
```
using BenchmarkTools
srand(280)
X = rand(0.0:2.0, 100, 100)
@benchmark e3(X)
```

Out[6]:
```
BenchmarkTools.Trial:
  memory estimate:  469.22 KiB
  allocs estimate:  12
  --------------
  minimum time:     191.685 μs (0.00% GC)
  median time:      259.976 μs (0.00% GC)
  mean time:        301.669 μs (7.23% GC)
  maximum time:     5.129 ms (0.00% GC)
  --------------
  samples:          10000
  evals/sample:     1
```

Now we optimize the computation of $\hat{\Phi}$.

In [7]:
```
# variant 1: loops (slow? yes)
function kinship1(X::Matrix{Float64})
    (n,m) = size(X)
    e = e3(X)

    p = zeros(Float64, m)
    for k = 1:m
        p[k] = (1/(2*n))*sum(X[:,k])
    end

    sum_p = 0
    for k = 1:m
        sum_p += (p[k]^2 + (1- p[k])^2)
    end

    Phi = zeros(Float64, (n,n))
    for i = 1:n, j = 1:n
        Phi[i,j] = (e[i,j]-sum_p) / (m - sum_p)
    end

    return Phi
end
```

Out[7]:
```
kinship1 (generic function with 1 method)
```

In [8]:
```
# variant 2: avoid loops (faster? yes)
function kinship2(X::Matrix{Float64})
    (n,m) = size(X)
    e = e3(X)
    p = (1/(2*n)) * sum(X, 1)
    sum_p = m + 2*sum(p.^2 - p) # expand p^2 + (1 - p)^2 and simplify
    Phi = (e - sum_p) ./ (m - sum_p)
    return Phi
end
```

Out[8]:
```
kinship2 (generic function with 1 method)
```

In [9]:
```
using BenchmarkTools
srand(280)
X = rand(0.0:2.0, 1000, 10000)
@benchmark kinship1(X)
```

Out[9]:
```
BenchmarkTools.Trial:
  memory estimate:  360.48 MiB
  allocs estimate:  6049505
  --------------
  minimum time:     843.138 ms (17.51% GC)
  median time:      958.979 ms (16.49% GC)
  mean time:        1.002 s (23.20% GC)
  maximum time:     1.355 s (38.81% GC)
  --------------
  samples:          5
  evals/sample:     1
```

In [22]:
```
using BenchmarkTools
srand(280)
X = rand(0.0:2.0, 1000, 10000)
@benchmark kinship2(X)
```

Out[22]:
```
BenchmarkTools.Trial:
  memory estimate:  198.67 MiB
  allocs estimate:  27
  --------------
  minimum time:     368.366 ms (2.70% GC)
  median time:      506.146 ms (7.29% GC)
  mean time:        500.862 ms (13.87% GC)
  maximum time:     642.811 ms (27.00% GC)
  --------------
  samples:          10
  evals/sample:     1
```

2) Benchmark your implementation.

In [33]:
```
using BenchmarkTools
srand(280)
X = rand(0.0:2.0, 1000, 10000)
@benchmark kinship2(X)
```

Out[33]:
```
BenchmarkTools.Trial:
  memory estimate:  198.67 MiB
  allocs estimate:  27
  --------------
  minimum time:     334.318 ms (1.35% GC)
  median time:      413.373 ms (8.16% GC)
  mean time:        464.584 ms (14.96% GC)
  maximum time:     688.434 ms (26.77% GC)
  --------------
  samples:          11
  evals/sample:     1
```

We achieve a minimum time of 335ms using 200MiB memory. Note that this benchmark was executed on a juliabox.com instance (possibly shared by multiple virtual enviroments / users).

In [12]:
```
versioninfo()
```

```
Julia Version 0.5.1
Commit 6445c82 (2017-03-05 13:25 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: Intel(R) Xeon(R) CPU @ 2.60GHz
  WORD_SIZE: 64
  BLAS: libopenblas (USE64BITINT DYNAMIC_ARCH NO_AFFINITY Sandybridge)
  LAPACK: libopenblas64_
  LIBM: libopenlibm
  LLVM: libLLVM-3.7.1 (ORCJIT, sandybridge)
```

Q2

1) Sherman-Morrison

Show $L^{-1} = R \Leftrightarrow LR = RL = I$.

(1) $LR = I$.

$$\begin{aligned}
LR &= (A + uu^T)\left(A^{-1} - \frac{A^{-1}uu^TA^{-1}}{1 + u^TA^{-1}u}\right) \\
&= AA^{-1} + uv^TA^{-1} - \frac{AA^{-1}uv^TA^{-1} + uv^TA^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} \\
&= I + uv^TA^{-1} - \frac{uv^TA^{-1} + uv^TA^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} \\
&= I + uv^TA^{-1} - \frac{u(1 + v^TA^{-1}u)v^TA^{-1}}{1 + v^TA^{-1}u} \\
&= I + uv^TA^{-1} - uv^TA^{-1} \\
&= I.
\end{aligned}$$

(2) $RL = I.$

$$\begin{aligned}
RL &= \left(A^{-1} - \frac{A^{-1}uu^TA^{-1}}{1 + u^TA^{-1}u}\right)(A + uu^T) \\
&= I.
\end{aligned}$$

QED.

2) Woodbury

Apply binomial inverse theorem (3) with $B = I$. QED.

3) Binomial inversion

$$\begin{aligned}
&(A + UBV^T)(A^{-1} - A^{-1}U(B^{-1} + V^TA^{-1}U)^{-1}V^TA^{-1}) \\
&= (A + UBV^T)(A^{-1} - A^{-1}UB(B + V^TA^{-1}UB)^{-1}BV^TA^{-1}) \\
&= (A + UBV^T)(A^{-1}) - ((A + UBV^T)(A^{-1})(UB(B + V^TA^{-1}UB)^{-1}BV^TA^{-1}) \\
&= (A + UBV^T)(A^{-1}) - ((A + UBV^T)(A^{-1}UB)(B + V^TA^{-1}UB)^{-1}BV^TA^{-1}) \\
&= (A + UBV^T)(A^{-1}) - (U(B + BV^TA^{-1}UB))(B + V^TA^{-1}UB)^{-1}BV^TA^{-1}) \\
&= I + UBV^TA^{-1} - U(B + BV^TA^{-1}UB)(B + BV^TA^{-1}UB)^{-1}BV^TA^{-1} \\
&= I + UBV^TA^{-1} - UBV^TA^{-1} \\
&= I.
\end{aligned}$$

QED.

4) Recall Sylvester's determinant theorem

$$\det(I_m + AB) = \det(I_n + BA).$$

Observe

$$\det(X + AB) = \det(X)\det(I_n + BX^{-1}A).$$

Apply

$$\begin{aligned}\det(A + UV^T) &= \det A \det(I_n + A^{-1}UV^T) \\ &= \det A \det\begin{pmatrix} I & V^T \\ -A^{-1} & I \end{pmatrix} \\ &= \det A \det(I_m + V^T A^{-1} U).\end{aligned}$$

QED.

Q3

We know

$$y_i = x_i^T\beta + z_i^T\gamma + \epsilon_i$$

where $\epsilon \sim N(0, \sigma_0^2)$ and $\gamma \sim N(0_q, \sigma_1^2 I_q)$.

1) Show

$$y \sim N(X\beta, \sigma_0^2 \cdot I + \sigma_1^2 ZZ^T).$$

Note that the sum of normally distributed random variables is normally distributed. Observe

$$\begin{aligned}E(y) &= E(X\beta + Z\gamma + \epsilon) \\ &= X\beta + ZE(\gamma) + E(\epsilon)\ . \\ &= X\beta\end{aligned}$$

And

$$\begin{aligned}Var(y) &= Var(X\beta + Z\gamma + \epsilon) \\ &= Var(X\beta) + ZVar(\gamma)Z^T + Var(\epsilon) + 2Cov(X\beta, Z\gamma) + 2Cov(X\beta, \epsilon) + 2Cov(Z\gamma, \epsilon) \\ &= \sigma_1^2 ZZ^T + \sigma_0^2 I.\end{aligned}$$

QED.

2) Implement function logpdf.

In [13]:
```
function logpdf_mvn(y::Vector, Z::Matrix, sigma0, sigma1)
    sigma = sigma0^2 * I + sigma1^2 * Z * Z'
    n = length(y)
    sigmachol = cholfact(sigma)
    - (n//2) * log(2π) - (1//2) * logdet(sigmachol) - (1//2) * sumabs2(s
igmachol[:L] \ y)
end
```

Out[13]:
```
logpdf_mvn (generic function with 1 method)
```

3) Benchmark your result.

In [14]:
```
using BenchmarkTools, Distributions

srand(280)

n, q = 2000, 10
Z = randn(n, q)
sigma0, sigma1 = 0.5, 2.0

sigma = sigma1^2 * Z * Z' + sigma0^2 * I
mvn = MvNormal(sigma)
y   = rand(mvn)

# check you answer matches that from Distributions.jl
@show logpdf_mvn(y, Z, sigma0, sigma1)
@show logpdf(mvn, y)

# benchmark
@benchmark logpdf_mvn(y, Z, sigma0, sigma1)
```

```
logpdf_mvn(y,Z,sigma0,sigma1) = -1571.5736734654183
logpdf(mvn,y) = -1571.5736734654186
```

Out[14]:
```
BenchmarkTools.Trial:
  memory estimate:  122.39 MiB
  allocs estimate:  25
  --------------
  minimum time:     171.113 ms (23.83% GC)
  median time:      217.344 ms (33.17% GC)
  mean time:        229.807 ms (33.25% GC)
  maximum time:     343.878 ms (38.90% GC)
  --------------
  samples:          22
  evals/sample:     1
```

In [15]:
```
@benchmark logpdf(mvn, y)
```

Out[15]:
```
BenchmarkTools.Trial:
  memory estimate:  15.78 KiB
  allocs estimate:  3
  --------------
  minimum time:     9.624 ms (0.00% GC)
  median time:      13.829 ms (0.00% GC)
  mean time:        14.273 ms (0.00% GC)
  maximum time:     40.516 ms (0.00% GC)
  --------------
  samples:          349
  evals/sample:     1
```