APTS Statistical Computing: Assessment 2015/16

The work provided here is intended to take up to half a week to complete. Students should talk to their supervisors to find out whether or not their department requires this work as part of any formal accreditation process. It is anticipated that departments will decide on the appropriate *level* of assessment locally, and may choose to drop some (or indeed all) of the parts, accordingly. So make sure that your supervisor or local organizer of APTS assessment has looked at the assignment before you start, and has told you which parts of it to do. In order to avoid undermining institutions' local assessment procedures the module lecturer will not respond to enquiries from students about this assignment.

Supplementary functions `sparse.vectorise` and `generate.Q` are here:
`http://people.bath.ac.uk/fl353/apts/functions.R`

A number of statistical problems require estimation of dependence structures, such as covariance matrices, precision matrices, or conditional dependence model parameters. Sometimes it is helpful to restrict the allowed conditional dependence structure by imposing a Markov condition. In a Gaussian model that is Markov with respect to a specified conectivity graph, the most flexible alternative is to look for a positive definite *precision matrix* $\boldsymbol{Q}$ such that its non-zero pattern matches the graph structure.

One way of parameterising such an $n \times n$ matrix $\boldsymbol{Q}$ is via its Cholesky decomposition $\boldsymbol{Q} = \boldsymbol{R}^\top \boldsymbol{R}$. For a given order of the nodes in the model, the Cholesky factor can also be sparse (but usually with additional non-zero *infill* nodes compared with $\boldsymbol{Q}$), and we will parameterise such matrices in the following way:

$$\mathbf{R} = \begin{pmatrix} \mathrm{e}^{\theta_1} & 0 & \theta_{n+1} & \theta_{n+3} & \cdots & 0 \\ 0 & \mathrm{e}^{\theta_2} & \theta_{n+2} & 0 & \cdots & \theta_{n+4} \\ 0 & 0 & \mathrm{e}^{\theta_3} & 0 & \cdots & \theta_{n+5} \\ 0 & 0 & 0 & \mathrm{e}^{\theta_4} & \cdots & \theta_{n+6} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \mathrm{e}^{\theta_n} \end{pmatrix}$$

where $\boldsymbol{\theta}$ is a parameter vector, and $\mathbf{Q} = \mathbf{R}^\mathsf{T}\mathbf{R}$. The total number of parameters is N, which in the example above is equal to $n + 6$.

Let $\mathcal{X} = \{\boldsymbol{x}^{[1]}, \ldots, \boldsymbol{x}^{[M]}\}$ be a set of M independent observational vectors from $\boldsymbol{x} \sim \mathrm{N}(\mathbf{0}, \boldsymbol{Q}^{-1})$ (we will store it as an $n \times M$ matrix `X`). We can seek $\hat{\boldsymbol{\theta}}$ to minimise the negated log-likelihood function

$$\begin{aligned} \ell(\boldsymbol{\theta}; \mathcal{X}) &= \frac{nM}{2}\log(2\pi) - \frac{M}{2}\log\det(\boldsymbol{R}^\top\boldsymbol{R}) + \frac{1}{2}\sum_{m=1}^{M} {\boldsymbol{x}^{[m]}}^\top \boldsymbol{R}^\top \boldsymbol{R}\boldsymbol{x}^{[m]} \\ &= \frac{nM}{2}\log(2\pi) - M\sum_{i=1}^{n}\theta_i + \frac{1}{2}\sum_{m=1}^{M} {\boldsymbol{x}^{[m]}}^\top \boldsymbol{R}^\top \boldsymbol{R}\boldsymbol{x}^{[m]}. \end{aligned} \tag{1}$$

If we let $\boldsymbol{E}_i$ be a matrix with zeros everywhere except for a 1 in the element corresponding to the i^{th} parameter, the derivatives of $\ell(\boldsymbol{\theta}; \mathcal{X})$ with respect to each θ_i can be obtained through

$$\frac{\partial \boldsymbol{R}}{\partial \theta_i} = \begin{cases} \mathrm{e}^{\theta_i} \boldsymbol{E}_i, & 1 \leq i \leq n, \\ \boldsymbol{E}_i, & n < i \leq N, \end{cases}$$

$$\frac{\partial \ell(\boldsymbol{\theta}; \mathcal{X})}{\partial \theta_i} = -M \mathbb{I}[i \leq n] + \sum_{m=1}^{M} \boldsymbol{x}^{[m]\top} \boldsymbol{R}^\top \frac{\partial \boldsymbol{R}}{\partial \theta_i} \boldsymbol{x}^{[m]}.$$

Answer the following questions.

0. First, generate a test case with `generate.Q` from `functions.R`:

```
source("functions.R")
set.seed(1)
Q <- generate.Q(100)
image(Q)
R <- chol(Q)
pattern <- (R != 0)
image(R)
M <- 10
X <- solve(R, Matrix(rnorm(nrow(Q) * M), nrow(Q), M))
qqnorm(diag(1 / diag(solve(Q))^0.5) %*% X)
```

   The last line can be used to check if something in the $\mathcal{X}$ sampling is obviously incorrect.

1. Explain why the representation of $\mathbf{Q}$ is a sensible way of parameterizing a (sparse) positive definite matrix.

2. Write an R routine `sparse.to.theta` to take a sparse Cholesky factor $\mathbf{R}$ and a `pattern` and turn it into a parameter vector $\boldsymbol{\theta}$ (`theta`). *Hint*: use `sparse.vectorise`

3. By reading the help text for `Matrix::sparseMatrix` and the information in `functions.R`, figure out how to do the reverse of `sparse.vectorise`, i.e. how to reconstruct a matrix `R` from the output of `sparse.vectorise(R, pattern)`.
   Hint: the output matches some of the needed input to `sparseMatrix`.

4. Write an R routine `sparse.from.theta` to take a parameter vector $\boldsymbol{\theta}$ (`theta`) and a pattern (`pattern`) and convert it into the equivalent sparse Cholesky factor $\mathbf{R}$.
   Hint: `sparse.vectorise(pattern, pattern)` and `sparseMatrix` may be useful.

   Check that, with `theta <- sparse.to.theta(R, pattern)`,
   `sparse.to.theta(sparse.from.theta(theta, pattern), pattern)`
   is equal to `theta`, and that
   `sparse.from.theta(sparse.to.theta(R, pattern), pattern)`
   is the same matrix as `R`, and also that `sparse.from.theta(theta, pattern)` is the Cholesky factor of `Q`, all up to small numerical deviations.

5. Write an R function `negloglike` which takes arguments `theta` ($\boldsymbol{\theta}$, vector of length N), $\mathcal{X}$, and a pattern (`X`, $n \times M$ matrix, and `pattern`, a sparse $n \times n$ matrix), and evaluates $\ell(\boldsymbol{\theta}; \mathcal{X})$. See the comment at the end of `functions.R` for a suggestion for the first few lines of the function. The output of `negloglike(theta, X, pattern)` for the earlier test case should be approximately 208.65.

   In the case of a dense $\boldsymbol{R}$ and a simple naive left-to-right implementation, the third term in (1) would have operator count $\sim 3Mn^2$. Your implementation should have operator count at most $\sim M(n^2 + 2n)$, and explicit loops (in particular, no loop over m!).

6. Write an R function (same arguments as previous part) to evaluate the vector of $\partial\ell/\partial\theta_i$ values. For a simple implementation it may be helpful to first write a function `sparse.E(i,pattern)` very similar to `sparse.from.theta` that constructs an $\boldsymbol{E}_i$ matrix. Do however take care to not evaluate any matrix products of the type $\boldsymbol{R}^\top\boldsymbol{R}$, $\boldsymbol{R}^\top\boldsymbol{E}_i$, or $\boldsymbol{E}_j^\top\boldsymbol{E}_i$, but instead using sparse matrix-vector and/or element wise products wherever suitable.

   Again, you should be able to not loop over $m = 1, \ldots, M$, but you are allowed to loop over the indices into θ, $i = 1, \ldots, N$. On the lecturer's system, the output of
   `system.time({ print(norm(as.matrix(negloglike.grad(theta, X, pattern)))) })`
   is

```
[1] 902.7741
   user  system elapsed
  2.056   0.000   2.052
```

7. Write basic R code to test your routine from part 6 by finite differencing. If there is a problem, go back to 6 and fix it!

8. Now we would like to use `optim` with our new functions to find the Maximum Likelihood estimate of $\boldsymbol{\theta}$. The output of an R session trying to do that is copied on the next page. Explain why the convergence in the test case appears to be very slow. *Hint*: consider the values of n, N, and M!

```
set.seed(1)
Q <- generate.Q(100)
image(Q)
R <- chol(Q)
pattern <- (R != 0)
image(R)
M <- 10
X <-solve(R, Matrix(rnorm(nrow(Q) * M), nrow(Q), M))
qqnorm(diag(1 / diag(solve(Q))^0.5) %*% X)
> ## True theta:
theta <- sparse.to.theta(R, pattern)
> ## Try to find the ML estimate, for simplicity starting from the true
> ## parameter vector:
> theta.opt <- optim(theta, fn=negloglike, gr=negloglike.grad,
+                    X, pattern,
+                    method="BFGS",
+                    control=list(trace=1, maxit=200))
initial  value 208.649069
iter  10 value 61.595031
iter  20 value -54.461304
iter  30 value -131.668431
iter  40 value -170.194944
iter  50 value -207.353905
iter  60 value -232.249357
iter  70 value -250.212042
iter  80 value -268.056399
iter  90 value -285.275605
iter 100 value -305.095776
iter 110 value -327.209242
iter 120 value -346.325421
iter 130 value -371.769576
iter 140 value -402.803958
iter 150 value -431.437734
iter 160 value -452.370010
iter 170 value -473.123122
iter 180 value -491.680796
iter 190 value -512.041399
iter 200 value -527.022856
final  value -527.022856
stopped after 200 iterations
```