APTS Statistical Computing: Lab Problems

Here are some practical problems which aim to explore and reinforce some of the course material. They all use R. This is for convenience: many statistical numerical analysis tasks are best approached with a mixture of compiled code, using a language like C, and code in a high level language such as R or Matlab. But for exploratory purposes the high level languages are best. The problems here all use simulated data: when developing statistical modelling code, it is often best to start out with data where you know what the truth is (and can generate further replicates).

1. This question revisits the third example in section 1.1 of the notes. First simulate the data used in that example.
   (a) Write code to reproduce the good and bad fits produced by calls to `lm` in the example.
   (b) Examine the condition numbers of the model matrices in the two cases. Why was the second `lm` fit so bad?
   (c) Plot the second and third columns of the model matrix against each other for the two cases, and use `cor` to examine their correlation. Why are the condition numbers so high here?
   (d) One fix is simply to subtract the mean `x` value from the `x` vector before fitting. Try this and see what happens to the condition number and column correlations of the model matrix now.
   (e) If we want good condition numbers then the best thing would be have a model matrix made up of columns from an orthogonal matrix. *Orthogonal polynomials* provide a way of achieving this. Try fitting with `lm(y~poly(x,2))` to see these in action. Look at the correlation between the model matrix columns now, and the condition number.
   (f) The model matrix produced in the last part is not quite (column) orthogonal (meaning orthonormal), but we could rescale and use a model matrix `X <- cbind(n^-.5,poly(x,2))` where `n` is the number of data. What is the QR decomposition of this matrix? Show how the model matrix can be used directly to find the fitted values in this case, without any need for an `lm` call. Check that it works (efficiently please).

2. Consider the linear mixed model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{b} + \boldsymbol{\epsilon}, \quad \mathbf{b} \sim N(\mathbf{0}, \mathbf{I}\sigma_b^2), \quad \boldsymbol{\epsilon} \sim N(\mathbf{0}, \mathbf{I}\sigma^2)$$

   $\mathbf{X}$ and $\mathbf{Z}$ are model matrices, $\boldsymbol{\beta}$, σ_b^2 and σ^2 are parameters.

   (a) First simulate some data from a model of this sort, taking care to relate the code back to the mathematical statement of the model...

```
n <- 100;n.b <- 10;n.beta <- 5
X <- cbind(1,matrix(runif(n*n.beta-n),n,n.beta-1))
Z <- matrix(runif(n*n.b),n,n.b)
beta <- rep(1,n.beta)
b <- rnorm(n.b)
y <- X%*%beta + Z%*%b + rnorm(n)
```

   (b) Now find the (marginal) expectation, $\boldsymbol{\mu}$, and covariance matrix, $\mathbf{V}$, of $\mathbf{y}$. State the (marginal) distribution of $\mathbf{y}$.
   (c) Write an R function to evaluate the log likelihood of $\boldsymbol{\theta}^{\mathsf{T}} = (\boldsymbol{\beta}^{\mathsf{T}}, \sigma_b^2, \sigma^2)$ given data $\mathbf{y}$. Make sure that $\boldsymbol{\theta}$ is the first argument of the function.
   (d) To maximize the log likelihood of the model using unconstrained methods, it is better to use a parameterization that guarantees positive variances. Modify your function to accept a parameter vector $\boldsymbol{\theta}^{\mathsf{T}} = (\boldsymbol{\beta}^{\mathsf{T}}, \rho_b, \rho)$ where $\rho = \log(\sigma)$ and $\rho_b = \log(\sigma_b)$.

(e) Use `optim` to maximize your likelihood (note that `optim` *minimizes* by default).

(f) Actually, using general purpose optimization methods to find the optimizing $\boldsymbol{\beta}$ is a bit wasteful. Given the variance parameters, closed form expressions for the $\boldsymbol{\beta}$ maximizing the likelihood are available, and might as well be used. Then it is only necessary to use general methods for the variance parameters. The likelihood considered only as a function of the variance parameters, with the corresponding MLEs of $\boldsymbol{\beta}$ 'plugged in' is termed a 'profile likelihood'. Show that, given the variance parameters, the log-likelihood is maximized by the $\boldsymbol{\beta}$ minimizing

$$(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^{\mathsf{T}}\mathbf{V}^{-1}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}) = \|\mathbf{R}^{-\mathsf{T}}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})\|^2$$

where $\mathbf{R}^{\mathsf{T}}\mathbf{R} = \mathbf{V}$. ($\|\mathbf{x}\|^2 = \mathbf{x}^{\mathsf{T}}\mathbf{x}$ here.) Hence, produce a 'profile log likelihood' function equivalent to your previous log likelihood function. Your function should accept a vector of variance parameters as its first argument, and should return the corresponding profile log likelihood value. You might want to return the corresponding $\boldsymbol{\beta}$ values as an attribute of the return value. e.g.

```
.
.
  attr(ll, "beta") <- beta
  return(ll)
}
```

(g) Use optim to maximize your profiled log likelihood function, and confirm that you get near identical parameter estimates to those from part (e).

(h) Only if you have masses of time left over and would like a further challenge: Forming $\mathbf{V}$ explicitly is rather wasteful. Figure out how to evaluate the log likelihood without doing this, by making use of the QR decomposition of $\mathbf{Z}$.

3. Write your own code to optimize Rosenbrock's function by Newton's method.

4. `optim` will optionally return a finite difference approximation to the Hessian of its objective. Using this fact, try repeating the Laplace approximation and Laplace importance sampling parts of example in section 4.4.8. using `optim` to find the $\hat{\mathbf{b}}$ and the approximate Hessian.

5. Experiment with model fitting by maximizing the log likelihood from the previous question.