APTS Stat Comp Assessment 2010

APTS does not conduct assessments. The material here is intended to provide the basis for assessment by student's home departments, but it is assumed that those departments will decide locally how much of the assessment it is appropriate to use, as well as on the *level* expected (in particular, the last part, 9., is only for those requiring something quite challenging). So, **before** starting the assessment, discuss which parts of the assessment you should complete, with your advisor/supevisor. The work should take up to half a week.

The data frame loaded into R by the command:

```
alg <- read.table("http://www.maths.bath.ac.uk/~sw283/MA40198/algae.dat")
```

gives the population of cells in a laboratory Chemostat (`cell.pop`) at a series of times (`hours`) from the start of the experiment. One possible model for the counts is that the population, N_t, is growing according to the dynamic model:

$$N_{t+1} = N_t \exp\{\Delta r(1 - N_t/K) + e_t\} \tag{1}$$

where the 'process errors', e_t, are i.i.d. $N(0, \sigma_e^2)$; N_0, r, K and σ_e^2 are unknown parameters; Δ is a timestep. Assume that Δ is 2 hours, and that the index t counts the number of timesteps taken. The `cell.pop` data (y, say) is observed with error, and not at regular intervals, so the model is completed by the further assumption

$$\mathtt{y}(\tau) = n(\tau) + \epsilon_\tau$$

where the ϵ_τ are i.i.d. $N(0, \sigma^2)$, τ is continuous time and $n(\tau)$ is the piecewise linear interpolant of the N_t. This introduces a further parameter σ^2.

The aim of this assignment is to estimate the 5 parameters of this model, using some of the methods covered in the statistical computing course. Initial values of $r = .07$, $K = 1000$, $N_0 = 5$, $\sigma = 20$ and $\sigma_e = 0.05$ can easily be found from the raw data.

1. The R routine, `srick`, given below, iterates (1), for given r, K, N_0 and e_t. It then computes (half) the sum of squares, S, of differences between the data in `y` and the model predicted population at the observation times in `t.y`. In addition, the routine evaluates the first derivative vector, and Hessian matrix, of S, w.r.t. the elements of `e`.

   Derive the derivative expressions used in `srick`.

2. Test the gradient and Hessian produced by `srick`, numerically. Fix any errors you find.

3. Write down the joint p.d.f. of the cell population data vector, $\mathbf{y}$, and the vector of 'process errors', $\mathbf{e}$. Write an R routine to evaluate the log of this joint p.d.f. and its gradient vector and Hessian matrix w.r.t. the elements of $\mathbf{e}$, via a call to `srick`.

4. Write an R routine to evaluate the log likelihood of the model parameter vector

   $$\boldsymbol{\theta}^\mathsf{T} = (\log r, \log K, \log N_0, \log \sigma, \log \sigma_e),$$

   using a Laplace approximation (why is a log parameterization appropriate here?). Within this function, you should write your own Newton iteration loop to find the process error values, $\hat{\mathbf{e}}_y$, which maximize the joint p.d.f. given the data. Make sure that this loop deals with indefinite Hessians appropriately, and includes a step halving loop, in case the initial Newton direction fails.

5. Check that the initial values, given above, are sensible, both graphically, and using your log-likelihood function.

6. Use `optim` with Nelder - Mead as the optimizer, to estimate the model parameters.

7. Does the model fit appear sensible?

8. Estimate approximate 95% confidence intervals for the parameters, if this appears to be sensible.

9. Extra part: Replace the Laplace approximate log likelihood with an importance sampling estimate of the log likelihood, using a 'Laplace proposal'. Re-estimate the parameters using the new approximation. To get this to work, you will need to use the same set of underlying standard normal random deviates each time the importance sampling estimator is used with new parameter values.

```
srick <- function(r,K,n0,e,y,t.y,dt=2)
{ n.e <- length(e)
  N <- rep(n0,n.e+1)
  dss.de <- dN.de <- rep(0,n.e)
  dss.dee <- dN.dee <- matrix(0,n.e,n.e)
  n.y <- length(y)
  mu <- y*0
  if (dt*n.e < t.y[n.y]) stop("data longer than simulation")
  ss <- t1 <- 0
  k <- 1 ## counter for data
  for (t in 1:n.e) {
    t0 <- t1; t1 <- t1 + dt
    dN.dee0 <- dN.dee;dN.de0 <- dN.de ## derivs at interval start
    exp.rNKe <- exp(dt*r*(1-N[t]/K)+e[t])
    N[t+1] <- N[t]* exp.rNKe
    ## start second derivatives...
    dN.dee <- dN.dee * exp.rNKe*(1-N[t]*r*dt/K) +
      dN.de%*%t(dN.de) * exp.rNKe *(N[t]*r*dt/K-2)*r*dt/K
    ## first derivatives...
    dN.de <- dN.de * exp.rNKe * (1 - N[t]*r*dt/K)
    dN.de[t] <- N[t+1]

    ## Complete second derivatives...
    dN.dee[,t] <- dN.dee[t,] <- dN.de
    dN.dee[t,t] <- N[t+1]

    ## Now get rss and derivs...
    while (t.y[k]>t0&&t.y[k]<=t1) {
      b <- (t.y[k]-t0)/dt
      mu[k] <- N[t]*(1-b) + N[t+1]*b ## E(y) by linear interp
      ss <- ss + .5 * (y[k] - mu[k])^2
      dmu.de <- dN.de0*(1-b) + dN.de*b
      dss.de <- dss.de - (y[k]-mu[k])*dmu.de
      dmu.dee <- dN.dee0*(1-b) + dN.dee*b
      dss.dee <- dss.dee - (y[k] - mu[k]) * dmu.dee +
                                      dmu.de%*%t(dmu.de)
      k <- k + 1
      if (k>n.y) break;
    }
    if (k>n.y) break;
  }
  list(dss.dee=dss.dee,dss.de=dss.de,N=N,ss = ss,mu=mu)
}
```