Bayesian Analysis of Tuberculosis Data

Introduction

Using the parameter values provided by James, we build a Bayesian model for tuberculosis spread among immigrants in Canada. Our goal is to estimate the percentage of new immigrants who have latent infections (i.e. early or late latent).

First, we define our fixed parameters.

```
# ----------------- All rates are on a time scale of years ----------------- #

# Background death rate for each compartment. The presence of alpha later makes this
const dS = 0.039
const dE = 0.039
const dL = 0.039
const dT = 0.039
const dR = 0.039

const alpha = 0.06  # Rate of additional deaths for those with active TB (homogeneous
const delta = 0.8   # Rate of recovery (homogeneous across compartments?)
const omega = 0.4   # Rate of progression out of early latent
const p = 0.05      # Fraction of people who progress directly from early latent to a
const nu = 0.0002   # Rate of progression out of late latent (to active)

const lambda = 223840   # Rate of immigration to Canada
const beta = 10^(-8)    # Transmission rate in Canada

r = 5   # Prior encourages q2 to be r times larger than q1
        # I.e. r times as many late latents immigrants as early latent immigrants
        # Not a constant because I would like to experiment with it
```

```
5
```

Next, we define priors for the missing parameters: q_1, the proportion of new immigrants who are early latent, and q_2, the proportion of new immigrants who are late latent. We assume that immigration screening catches all active cases of TB. Note that $q_1 + q_2$ cannot exceed 1 and, in fact, we expect $q_1 + q_2 \ll 1$ since otherwise we would see larger prevelance of TB in Canada. We do however assume that q_1 and q_2 are not both equal to zero as this would correspond to no TB importation, which is clearly false. This last assumption is not really binding since we will use continuous priors for q_1 and q_2, so the event $\{q_1 = q_2 = 0\}$ will have probability zero.

For now, we use the prior $\pi(q_1, q_2) \propto (r * q_1 + q_2)^{-1}$. The constant r determines the relative sizes of q_1 and q_2. I think that we should expect more late latents than early latents since people tend to spend less time in the early latent compartment. I'm not sure how much smaller I want q_1 to be (maybe ω/ν); I'm going to start with $r = 5$. In the future, we can try calling r a parameter and putting a hyperprior on it.

```
# Computes the prior density for q1 and q2.
function prior(q1, q2)
    prop_const = log(r)/(r-1)
    kernel = 1/(r*q1 + q2)
    return kernel / prop_const
end

function log_prior(q1, q2)
    return log(prior(q1, q2))
end
```

```
log_prior (generic function with 1 method)
```

Next, we need some data and a likelihood. This is where I'm going to need input from an ODE model. I think we've got historical data on numbers of cases in the population (i.e. across Canada), although it would be good to clarify exactly what this data looks like. I've found papers that connect data to a dynamical system by saying that data observed at time t follow a Poisson distribution with mean equal to whatever value is predicted by the dynamical system. This seems like a sensible place to start.

I would like you guys to write-up the dynamical system model that generates predictions at time points where we have data. Importantly, these predictions need to depend on the unspecified parameters q_1 and q_2. It is (theoretically) possible to call Matlab code in Julia, so however you want to build your function is fine. For now, I'm just going to generate random values so that I can get the code working.

```
using Distributions
using Random

Random.seed!(1)

N = 20  # Number of time points at which we have data

mu_ODE = 3  # Mean of ODE predictions


# Randomly generate some ODE predictions
ODE_dist = Exponential(mu_ODE)
ODE_vals = rand(ODE_dist, N)

# Randomly generate observed data centered on the ODE predictions
obs_vals = [rand(Poisson(ODE_vals[i])) for i in eachindex(ODE_vals)]
```

```
20-element Vector{Int64}:
  0
```

```
  3
  4
 10
  8
  1
  7
  4
  0
  1
  1
  4
  0
  0
  4
  1
  2
  0
  4
  1
```

It turns out that I do need a function to generate ODE predictions using q_1 and q_2. I'm just going to hack something together that should encourage reasonable values for the parameters. I'm also going to set the true values of these parameters here.

```
# True values of the disease importation parameters. Not constants so that I can test
q1_0 = 0.2
q2_0 = r*q1_0

# Returns an "ODE prediction". Rescaled up or down based on how q1 and q2 differ from
# I think there is an identifiability problem here, but I'm not worrying about it for
function ODE_pred(q1, q2)
    this_ODE_mean = mu_ODE * exp((q1 - q1_0) + (q2 - q2_0))
    return this_ODE_mean
end
```

```
ODE_pred (generic function with 1 method)
```

Now we do some analysis. Posterior is proportional to likelihood times prior, and we can get away without knowing the proportionality constant. First we need to code the likelihood function, then we can use it to compute the posterior. We assume that the data follow a Poisson distribution with mean equal to the prediction of the ODE system. It turns out to be more convenient to operate on log-scale because likelihoods tend to be quite small.

```
function log_likelihood(q1, q2, data)
    all_preds = [ODE_pred(q1, q2) for _ in eachindex(data)]
    all_lik_dists = Poisson.(all_preds)
    all_liks = map(pdf, all_lik_dists, data)    # Apply the pdf function elementwise t
    log_full_lik = sum(log.(all_liks))
    return log_full_lik
end
```

```
function log_posterior(q1, q2, data)
    A = log_likelihood(q1, q2, data)
    B = log_prior(q1, q2)

    return A+B
end

function posterior(q1, q2, data)
    log_post = log_posterior(q1, q2, data)
    return exp(log_post)
end
```

```
posterior (generic function with 1 method)
```

Now for some testing.

```
data = obs_vals
q1 = q1_0
q2 = q2_0

A = log_prior(q1_0, q2_0)
B = log_likelihood(q1, q2, data)

C = log_posterior(q1, q2, data)

C == A+B
```

```
true
```