Plim - Partial Likelihood Infection Modelling

Barry Rowlingson, Peter Diggle

July 22, 2011

The Problem

Our data typically consist of reports of cases of an infectious disease. These cases are discretised in space by being reported within a known region, and discretised in time by a reporting date - usually a day or week for human infectious data.

Along with the data, we need the baseline population at risk for each region. We only consider situations where the number of cases is small relative to the population, and we do not reduce the population at risk as cases occur. In the current implementation the population must be constant in time, so we cannot currently adjust for large population changes during the time-span of the data.

Our aim is to model how the infectivity and susceptibility of regions depends on region-level covariates. Hence we need another dataset of covariate values for each region. This can in principle vary in time if there are known changes to the status of the regions. Each covariate gets a corresponding parameter for infectivity and susceptibility.

To account for spatial dependence in the model we need to specify the relative rate of transmission between pairs of regions. Typically this will be some kind of distance-decay function, but full flexibility is implemented. The transmission rate between two regions can also vary in time, perhaps to adjust for regions being quarantined as an epidemic progresses.

We also need to know the time period for which a case is infectious. In the current implementation this is a constant value.

With all these items prepared the model is fitted using maximum partial likelihood. The resulting ouput consists of parameter estimates and an empirical Hessian matrix. From this, approximate standard errors on the estimates can be computed. Parameters significantly non-zero can be interpreted as the corresponding covariate having a significant effect on the infectivity or susceptibility of the regions.

The Statistical Model

We have:

- m regions in our study area, with populations n_r for $r = 1 \ldots m$.

- N cases of times and regions (t_k, r_k) for $k = 1 \ldots N$.
- A vector of possibly time-varying explanatory variables for each region, $Z_r(t)$.
- A matrix of regional pair-wise explanatory variables $w_{ij}(t)$, for i, j in $1..N$ – this is also possibly time-varying.
- δ, the length of time that a case is infectious.

We then model the infectivity, A_i, of a case in region i and the susceptibility, B_j of an individual in region j as log-linear terms with the covariates and some parameters α, β:

$$\begin{aligned} A_i(t) &= \exp(Z_i(t)'\alpha) \\ B_j(t) &= \exp(Z_j(t)'\beta) \end{aligned}$$

The transmissivity, W_{ij} between an infected case in region i and a susceptible individual in region j is then some function $f()$ of the w_{ij} matrix and a vector of parameters γ:

$$W_{ij}(t) = f(w_{ij}(t); \gamma)$$

The form of $f()$ is chosen appropriately for the nature of the problem.

Then the total rate of transmission from an infectious individual in region i to a susceptible in region j is:

$$\lambda_{ij}(t) = \lambda_0(t) A_i(t) B_j(t) W_{ij}(t)$$

where $\lambda_0(t)$ models any change in transmission over the study period, including changes in case ascertainment for any reason, as long as such change is applied equally across the regions.

Using partial likelihood means we do not need to model or fit $\lambda_0(t)$, since this cancels out in the partial likelihood formula. We condition on the observed case times, and work out the probabilities of the cases occuring in the given regions as opposed to all the other regions.

By this model, a case at time t can only occur if there are infected cases at that time. Cases that occur when there are no infectious cases active add no information to the likelihood and so do not contribute, except as possibly active infections at the next time point.

If we define $M_r(t)$ as the total number of cases in region r from time t to $t - \delta$, then we can write the probability of a case at time t being in region r given there is a case somewhere at time t as:

$$P_r(t) = \frac{n_r \sum_j M_j(t)\lambda_{jr}(t)}{\sum_i \left(n_i \sum_j M_j(t)\lambda_{ji}(t) \right)}$$

In this formula the $\lambda_0(t)$ cancels out, leaving us just with the A, B, and W terms in $\lambda_{ij}(t)$.

To compute the partial log-likelihood we loop over all the times at which we have cases to get L:

$$L(\alpha, \beta, \gamma) = \sum_{k=1}^{N} \log(P_{r_k}(t_k))$$

and then maximise L over (α, β, γ) to get parameter estimates. Approximate standard errors can be returned by optimisation functions.

The Package

The package loads in the usual way, and checks for the presence of OpenMP functionality. When found, it will print how many threads the system will use.

```
> library(plim)

OpenMP available, will use  2  threads
```

Data

Swine flu data for California was searched for on the internet. Reports for incidence by county were found on the California Department of Public Health web pages. These took the form of weekly tables of cumulative total cases. The data was scraped from the web and converted to weekly incidence.

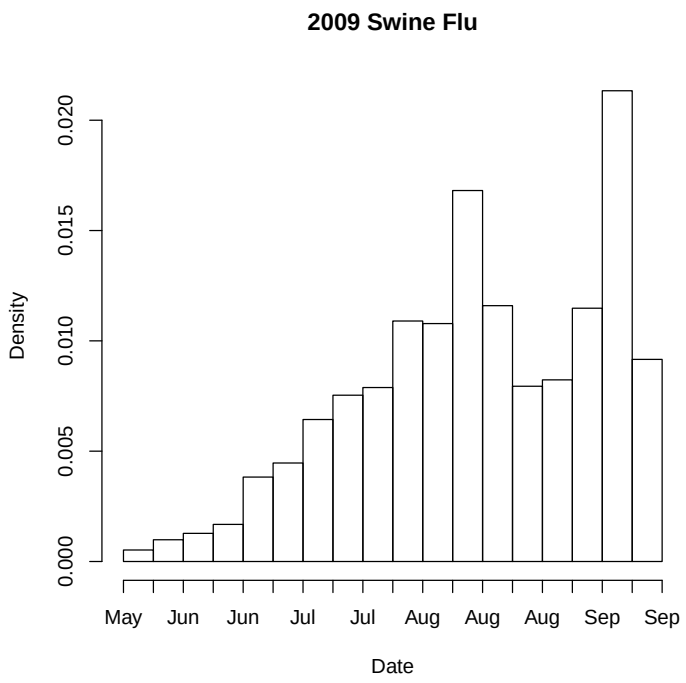
Problems with the data meant that some values needed to be imputed, such as when a cumulative count *decreased* in time. Cases were then randomly assigned to a day within the week. Missing values were linearly interpolated. The resulting dataset, distributed with this package, should not be treated as an accurate source of swine flu case counts for California, but merely used as an illustrative example. The structure mirrors the original inspiration for the development of the software which was an application to three areas in the UK during the early stage of the 2009 outbreak.

The case data is a two-column data frame, with 2464 cases. The first column is the standard US FIPS code for the county, and the second column is a vector of Date objects:

```
> summary(caFluCases)

   region               day
Length:2464        Min.   :2009-05-29
Class :character   1st Qu.:2009-07-29
Mode  :character   Median :2009-08-17
                   Mean   :2009-08-16
                   3rd Qu.:2009-09-12
                   Max.   :2009-09-26

> head(caFluCases)

  region        day
1 F06001 2009-06-06
2 F06001 2009-06-07
3 F06001 2009-06-09
4 F06001 2009-06-25
5 F06001 2009-06-20
6 F06001 2009-07-02
```

We can plot a histogram of the weekly total cases:

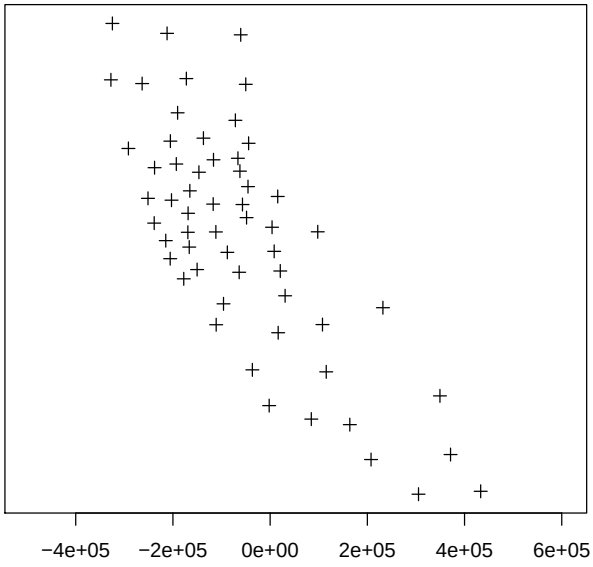
```
> hist(caFluCases$day,"week",xlab="Date",main="2009 Swine Flu")
```



The abrupt ending is due to the reporting changing from cases to hospitalisations. By then there is enough data for an example anyway.

Next some census data was obtained from online sources. To avoid distributing county boundaries with the package, a Spatial Points Data Frame is given, using the centroid locations of each county.

```
> library(sp)
> plot(caCountyData)
> axis(1)
> box()
```

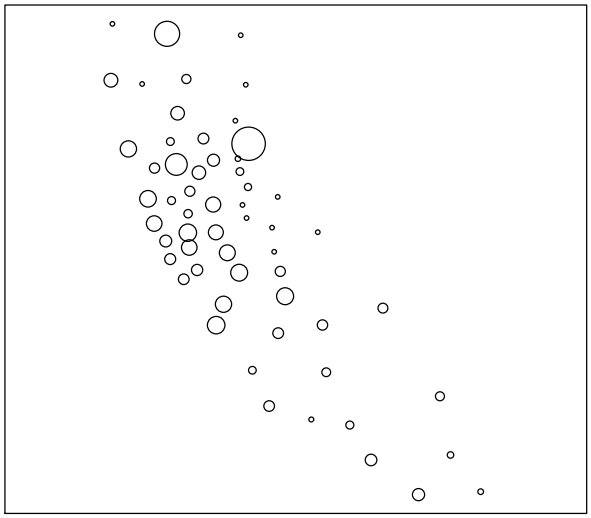


Each county has data relating to the distribution of ethnicity of its population and its economic state. The following table describes the columns in the data:

Column Name	Description
NAME	County Name
FIPS	FIPS county code
nWhite	white population count
nBlack	black population count
nNative	native population count
nAsian	asian population count
nPacific	hawaiian/other pacific islander count
nMixed	two or more races count
Population	total population
houseValueMed	median value of specified owner-occupied housing units
houseIncomeMed	median household income
pPeoplePoverty	people of all ages in poverty
fCivUnemp	civilian labor force unemployment

We can plot the total cases per population by county - first use `table` to get the total cases, and then plot using a scaled symbol. Notice how we use `factor` to both include the counties with zero cases in the data and ensure the totals are in the same order as the spatial data:

```
> totalCases = table(factor(caFluCases$region,
+   levels=caCountyData$FIPS))
> plot(caCountyData,cex=0.5+10000*(totalCases/caCountyData$Population),pch=1)
> box()
```



Fitting The Model

We set `delta` to 3 since flu cases are assumed to be infective for three days. Then for convenience we get the region codes and number of regions from the county data frame:

```
> delta = 3
> R = caCountyData$FIPS
> nRegions = length(R)
```

To prepare the data we need to specify which column contains the region IDs and which is the time - this is done by the `stCases` function which returns the same data frame but with an extra attribute so that `plim` knows where to get the region and time from. Although this seems pointless here, when data has more than two columns it becomes essential. [Later revisions of the package may use a formula-based method for this].

```
> caFluCases = stCases(caFluCases,"region","day")
```

Next we have to specify the covariate matrix. Our covariates do not change with time, so we use the `constantCovariates` function. We feed this a data frame of scaled covariate values, nicely named, and set the row names to the

FIPS values. Evaluating this function at any time value returns the scaled covariate matrix.

```
> Z1 = constantCovariates(
+    data.frame(
+       poverty = scale(caCountyData$pPeoplePoverty),
+       fWhite = scale(caCountyData$nWhite/caCountyData$Population),
+       row.names=R)
+     )
> Z1

Covariate generator function
58 regions
2 explanatory variables (poverty,fWhite)
Time-invariant

> # show the first few rows at time 0
> head(Z1(0))

          poverty     fWhite
F06001 -0.8376333 -2.9482490
F06003  0.3445911 -1.6322379
F06005 -0.9714700  0.6794125
F06007  1.4598972  0.4733668
F06009 -0.5922660  0.9088201
F06011 -0.1461436  0.8696211
```

Again this may seem overkill for a simple two-column covariate matrix, but it allows a more flexible framework to develop without breaking existing code.

Next we need the transmission rate factor between pairs of locations. This is another possibly time-dependent matrix, so we use something that returns a function of time. For the null model, we have $f(w_{ij}(t);\gamma) = 1$ for all i, j at all times. We use the `constantTrans` function to return a function that evaluates to one at all time points. This transmissivity function has no γ parameters:

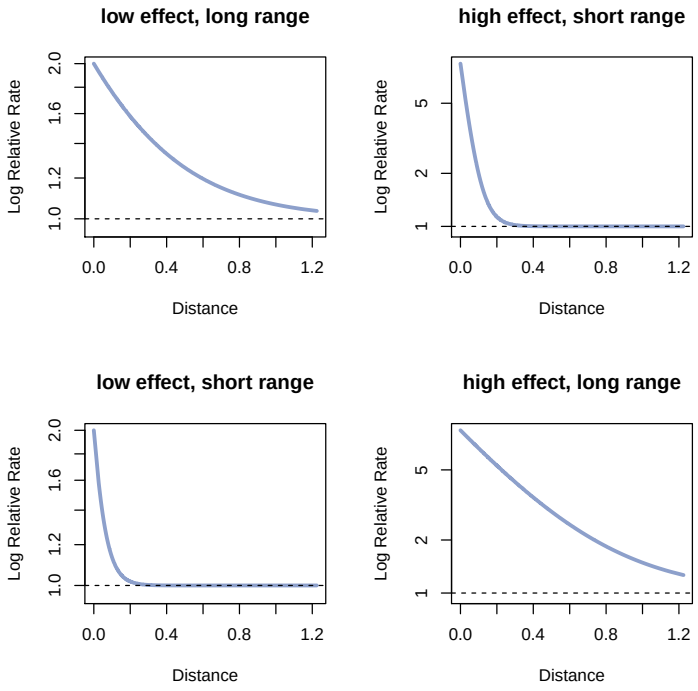
```
> W0 = constantTrans(matrix(1,ncol=nRegions,nrow=nRegions),R=R)
> W0

Transmission Matrix Generator
Fixed Transmission Model for 58 regions

> W0(0)[1:4,1:4]

       F06001 F06003 F06005 F06007
F06001      1      1      1      1
F06003      1      1      1      1
F06005      1      1      1      1
F06007      1      1      1      1
```

Not very exciting, but we can use `distDecayTrans` to implement a distance-decay effect between two regions. This function takes a distance matrix (the w_{ij} values) and returns a function of time and two γ parameters that control the magnitude and range of the distance decay. We scale the distance to thousands of kilometers (mega-metres?) to avoid large numbers.

```
> dMm = as.matrix(dist(coordinates(caCountyData)))/1e6
> W1 = distDecayTrans(dMm,R=R)
> W1

Transmission Matrix Generator
Distance Decay Model for 58 regions
   1+exp(gamma[1])*exp(-(dmat)/exp(gamma[2]))

> W1(0,c(0,0))[1:4,1:4]

         1        2        3        4
1 2.000000 1.810811 1.869155 1.797707
2 1.810811 2.000000 1.928331 1.823382
3 1.869155 1.928331 2.000000 1.853451
4 1.797707 1.823382 1.853451 2.000000
```

The plot method for `W1` can be used to show how the relative rate varies with distance under different parameter values:

```
> par(mfrow=c(2,2))
> plot(W1,c(0,-1),0);title("low effect, long range")
> plot(W1,c(2,-3),0);title("high effect, short range")
> plot(W1,c(0,-3),0);title("low effect, short range")
> plot(W1,c(2,-1),0);title("high effect, long range")
> par(mfrow=c(1,1))
```



Now we have everything ready to compute a null log-likelihood. Set the alpha and beta parameters for the covariates to zero, and compute using the constant transmission matrix:

```
> params=list(alpha=c(0,0),beta=c(0,0))
> nullModel = plimlik(params,caFluCases,delta,caCountyData$Population,Z1,W0)
> nullModel

[1] -7927.127
```

We can compute a likelihood for the model with a distance-decay transmission function by using `W1` and including some gamma parameters:

```
> params = list(alpha=c(0,0),beta=c(0,0),gamma=c(2,-1))
> dModel = plimlik(params,caFluCases,delta,caCountyData$Population,Z1,W1)
> dModel

[1] -7872.713
```

Fitting a model is done with the `plim` function, and a set of initial values. The default Nelder-Mead method failed to find a good solution, so we use the BFGS option. The `plimtable` function shows a table of parameter values and standard error estimates based on an empirical estimate of the Hessian matrix.

```
> params = list(alpha=c(0,0),beta=c(0,0),gammas=c(2,-1))
> fit = plim(params,caFluCases,delta,caCountyData$Population,Z1,W1,
+   method="BFGS"
+ )
> fit.table = plimtable(fit)
> fit.table

                 Estimate Std. Error    Z value      Pr(>|Z|)
alpha.poverty -2.01432705 0.15746375 -12.792322  1.809840e-37
alpha.fWhite  -0.08222873 0.12331011  -0.666845  5.048712e-01
beta.poverty   0.29686612 0.03504676   8.470571  2.441920e-17
beta.fWhite    0.17742046 0.03195658   5.551923  2.825447e-08
gamma          2.67061242 0.13253962  20.149540  2.716255e-90
gamma         -2.78321959 0.11238172 -24.765768 2.096921e-135
Log-likelihood:  -7707.305
```

Firstly note that our log-likelihood has increased by 219 for six degrees of freedom, which is a highly significant change in deviance for the extra parameters included.

This table then tells four things about the data:

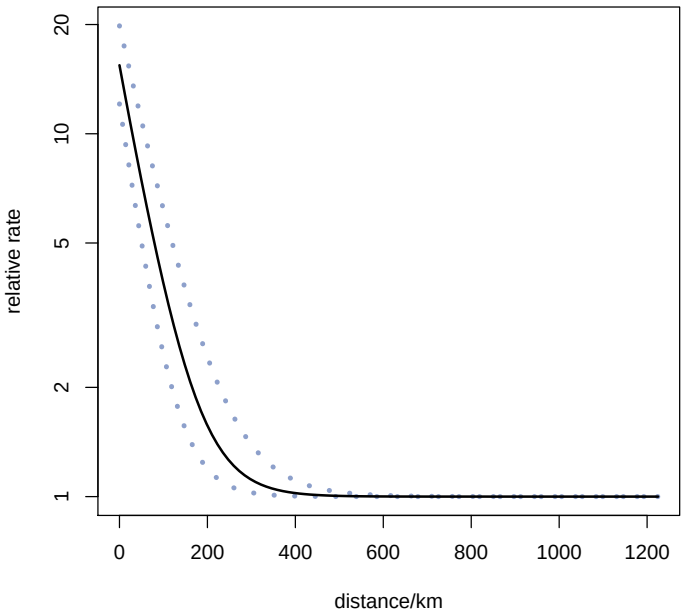
- Increasing infectivity with decreasing poverty
- No evidence of a link between infectivity and ethnicity
- Increasing susceptibility with increasing poverty
- Increasing susceptibility with increasing fraction of population white

Interpretation of these results are for the epidemiologists and social scientists. Perhaps affluent areas have more people travelling around, thus increasing the infectivity, but they also take more measures to prevent getting ill in the first place, thus decreasing their susceptibility?

Of course such debate is meaningless given the somewhat unauthoratitive nature of the data presented here, but still represents the sort of debate we have had with health providers in our motivating example of the UK swine-flu epidemic.

To get an idea of the spatial structure, we can plot the distance-decay function at the parameters of the maximum likelihood fit, together with the curve at plus and minus two standard errors of the fit to get some idea of the variability of the estimate:

```
> Wf = function(d,gamma){1+exp(gamma[1])*exp(-(d/1000)/exp(gamma[2]))}
> gfit = fit$par[5:6]
> gse = fit.table[5:6,2]
> s = seq(min(dMm),max(dMm),len=100)*1000
> Wv = cbind(Wf(s,gfit+2*gse),Wf(s,gfit-2*gse),Wf(s,gfit))
> matplot(s,Wv,type="l",xlab="distance/km",ylab="relative rate",
+  lty=c(3,3,1),col=c("#8DA0CB","#8DA0CB","black"),lwd=c(4,4,2),log="y")
```



This shows us that the relative transmission rate at zero distance is between 10 and 20 times the rate at large distance, and also that the effect drops off such that at 300km it is less than double.

Appendix: Writing your own transmission rate functions

The package enables you to formulate new ways of specifying the transmission rate between pairs of regions by the construction of new transmission matrix generating functions. These are subclasses of standard R functions with some extra attributes.

Consider the case where we have daily commuting flows between pairs of regions. We would expect the transmission rate to be increasing with increasing flow (unlike a distance-decay model where transmission rate decreases with increasing distance). We may wish to model this with a simple linear relationship, constrained to have positive slope and a positive intercept.

First we'll set up a simple matrix of hypothetical commuter flows, with some zeroes for regions where nobody works or lives:

```
> nR = 4
> M=matrix(as.integer(20*runif(nR*nR)*(runif(nR*nR)>.6)),ncol=nR)
> rownames(M)=letters[1:nR]
```

```
> colnames(M)=letters[1:nR]
> M

  a  b  c  d
a 0  0 13 19
b 8  0  0 13
c 0  0  0  0
d 0 16  0  0
```

Now we need a function of time and gamma parameters which returns the values of the linear function of the matrix. Since the optimiser works with unconstrained parameter values, we exponentiate `gamma` so the effective values are always positive:

```
> c1 = function(t,gamma){
+   return(exp(gamma[1]) + exp(gamma[2]) * M)
+     }
> c1(0,c(0.5,0.2))

          a         b         c         d
a  1.648721  1.648721 17.526957 24.855374
b 11.419943  1.648721  1.648721 17.526957
c  1.648721  1.648721  1.648721  1.648721
d  1.648721 21.191165  1.648721  1.648721
```

The problem with this is that it relies on the existence of matrix `M` outside of itself. If you change `M` then the function will return other values. We could perhaps have a function that took `M` as a parameter value, but that would mean that when the likelihood function tries to evaluate the transmission rates it would have to know how to call it.

To solve all this we write a function that returns a function. This ensures that:

- The *interface* for all transmission rate functions is the same: a function of the time and a vector of parameters
- Any data needed by the function is encapsulated statically within the function.

There are some further requirements that the `plim` routines need to know about transmission rate functions:

- How many gamma parameters does it take?
- Is it constant over time?
- What are the region names?
- How many regions are there?

This information is attached to the transmission function by attribute values. A sample transmission rate function constructor then looks as follows:

```
> LinearTrans = function (mat, R = row.names(mat))
+ {
+     force(mat) #otherwise 'mat' isn't visible in function ct
+     ct = function(t, gamma) {
+         return(exp(gamma[1]) + exp(gamma[2]) * mat)
+     }
+     attr(ct, "ngamma") = 2
+     attr(ct, "regions") = R
+     attr(ct, "tconst") = TRUE
+     attr(ct, "size") = nrow(mat)
+     class(ct) <- c("plimtrans", "incLinearTrans", "function")
+     return(ct)
+ }
```

Typical usage would then be:

```
> L = LinearTrans(M)
> L(0,c(0.5,0.2))

          a         b         c         d
a  1.648721  1.648721 17.526957 24.855374
b 11.419943  1.648721  1.648721 17.526957
c  1.648721  1.648721  1.648721  1.648721
d  1.648721 21.191165  1.648721  1.648721
```

This object can then be passed into `plim` and the likelihood will be maximised over the two gamma parameters.

A number of transmission generator functions are included in the package:

- `constantTrans` returns the input matrix as its output. Useful for null models or known fixed offsets.
- `distDecayTrans` the two-parameter distance decay effect used in the California flu example
- `incLinearTrans` a version of `LinearTrans`, but with more error checking
- `pieceLinearTrans` a piecewise-linear version of `incLinearTrans`.

The source code for each of these is readable and serves as further illustration.