Practical 3: Smooth deconvolution

The file `engcov.txt` contains data on deaths from Covid-19 in English hospitals (variable `nhs`) against day of the year 2020 (variable `julian`). It is useful to use these data to infer the trajectory of new infections each day that eventually resulted in these deaths. This is possible given that there is reasonably good data implying that if d is the interval from infection to death then $\log(d) \sim N(3.151, 0.469^2)$.

The following statistical model is one way to do this. Let y_i be the deaths on day of the year t_i, then

$$y_i \sim \text{Poi}(\mu_i) \text{ where } \mu_i = \sum_{j=1}^{\min(29+i,80)} f(t_i - j)\pi(j) \;\; i = 1, \ldots, n$$

where $\pi(j)$ is the probability function for days from infection to death and $f(t)$ is the number of new infections occurring on day t, which we assume to be a smooth function of time. The slightly fussy limits on the summation are to avoid looking too far back for the origins of the infections causing the first deaths, since it is inevitable that we see short durations before long durations at the epidemic start: this enhances statistical stability.

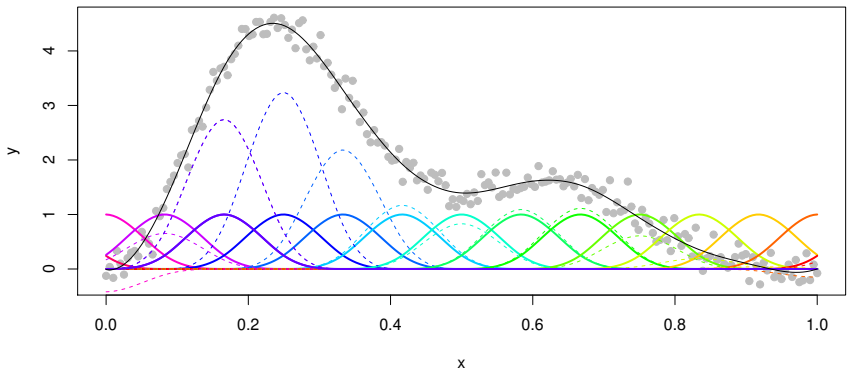
We can model $f(t)$ using the sort of linear model covered in the notes. If $\mathbf{f} = [f(t_1-30), f(t_1-29), \ldots, f(t_n)]^T$ we can write $\mathbf{f} = \tilde{\mathbf{X}}\boldsymbol{\beta}$ (e.g. we could model f as a polynomial function of time, although a better option is given later). Substituting this into the model, and writing the kth row of the model matrix for f as $\tilde{\mathbf{X}}_k$, we get

$$\mu_i = \sum_{j=1}^{\min(29+i,80)} \tilde{\mathbf{X}}_{30+i-j}\pi(j)\boldsymbol{\beta} = \mathbf{X}\boldsymbol{\beta}$$

by definition of $\mathbf{X}$ (the model matrix for the deaths). How exactly should $f(t)$ be represented? Polynomials with more than a few terms behave poorly, and a better option is to use B-splines. That is we write

$$f(t) = \sum_{k=1}^{K} b_k(t)\beta_k$$

where the β_k are unknown parameters to be estimated and the $b_k(t)$ are simple known functions - which in the B-spline case are just translations of each other. The following illustrates the idea.



Each of the solid coloured curves is a B-spline basis function, b_k. The dashed versions are after multiplication by β_ks chosen so that $\sum_k b_k(t)\beta_k$ (shown in black) closely approximates the grey points. Each of the b_k is made up of 4 sections of cubic polynomial, joined continuously at some evenly spaced x values, known as the 'knots' of the spline. In the picture there is a knot corresponding to the peak of each b_k plus some extra ones beyond the plotting range to complete the definitions. Splines are a good way to closely approximate a wide variety of unknown functions. So the model matrix for f will be defined by $\tilde{X}_{ij} = b_j(t_i)$.

For this model we will want to make K, the number of basis functions, fairly large (say 80), to avoid underfitting the data. But the risk is that we then overfit. A good way to deal with this is to impose a smoothing penalty during fitting that allows us to easily control the smoothness of f. The simplest reasonable one is $P = \lambda\sum_{k=2}^{K-1}(\beta_{k-1} - 2\beta_k + \beta_{k+1})^2/2$, which penalizes overly rapid variation in the β_k and hence in $f(t)$

itself. The penalty is added to the negative log likelihood during fitting, with *smoothing parameter*, λ, being increased/decreased to obtain smoother/wigglier fits. It is easy to show that $P = \lambda\boldsymbol{\beta}^T\mathbf{S}\boldsymbol{\beta}/2$ for a readily computed matrix $\mathbf{S}$ (`S <- crossprod(diff(diag(k),diff=2))`).

So far the model has the structure of a Poisson GLM, but with an identity link and an extra penalty on the likelihood. We can't change the identity link without changing the model, but that means that $f(t)$ could be estimated as negative, which makes no sense. That needs to be fixed, and the easiest way to do so is to re-express the model in terms of new parameters γ_k, such that $\beta_k = \exp(\gamma_k)$. This ensures that the β_k are positive. Because the B-spline basis functions, b_k, are non-negative, this means that $f(t)$ must be positive.

The basic task for this practical is to write code to fit the deconvolution model to the Covid death data, given a value for λ and then find the value of λ minimizing the model BIC criterion. The fitting will be done using the BFGS method built into `optim` to minimize the penalized negative log likelihood, using exact derivatives. The log likelihood for the Poisson is simply

$$\sum_{i=1}^{n} l_i \text{ where } l_i = y_i \log(\mu_i) - \mu_i - \log(y_i!)^1$$

To get the gradient of the penalized negative log likelihood w.r.t. $\boldsymbol{\gamma}$ requires $\partial l_i/\partial\gamma_j = F_{ij}$ where $\mathbf{F} = \text{diag}(y_i/\mu_i - 1)\mathbf{X}\text{diag}(\boldsymbol{\beta})$ and the fact that $\partial P/\partial\boldsymbol{\gamma} = \text{diag}(\boldsymbol{\beta})\mathbf{S}\boldsymbol{\beta}$.

1. Write a function to evaluate $\tilde{\mathbf{X}}$, $\mathbf{X}$ and $\mathbf{S}$. For $\tilde{\mathbf{X}}$ use `splineDesign` from the `splines` library in R. You will need to supply a sequence of $K+4$ evenly spaced 'knots' defining the B-splines, where the middle $K-2$ knots cover the interval over which $f(t)$ is to be evaluated. The following code evaluates the model probability of each fatal disease duration from 1 day up to the maximum of 80 days considered here.

   ```
   d <- 1:80;edur <- 3.151;sdur <- .469
   pd <- dlnorm(d,edur,sdur);
   pd <- pd/sum(pd)
   ```

2. Write functions, suitable for use with `optim`, evaluating the penalized negative log likelihood and its derivative vector w.r.t $\boldsymbol{\gamma}$. Don't forget to test the derivative function by finite differencing.

3. As a preliminary sanity check before proceeding, and as part of finding sane starting values for $\boldsymbol{\gamma}$, fit the model using $\lambda = 5 \times 10^{-5}$, and plot the actual and fitted deaths against time, along with the fitted daily infection curve, f, taking care to get the relative timing right.

4. We need some way of choosing an appropriate λ. One approach is to choose λ to minimise the BIC criterion, $-2l(\hat{\boldsymbol{\beta}}) + \log(n)\text{EDF}$, where EDF is the effective degrees of freedom of the model. This can be defined as $\text{EDF} = \text{trace}(\mathbf{H}_\lambda^{-1}\mathbf{H}_0)$ where $\mathbf{H}_\lambda$ is the Hessian (second derivative matrix) w.r.t. $\boldsymbol{\beta}$ (not $\boldsymbol{\gamma}$!) of the negative log-likelihood at $\hat{\boldsymbol{\beta}}$ plus $\lambda\mathbf{S}$. That is $\mathbf{H}_\lambda = \mathbf{X}^T\mathbf{W}\mathbf{X} + \lambda\mathbf{S}$ where $\mathbf{W} = \text{diag}(y_i/\hat{\mu}_i^2)$. Note that $\hat{\mu}_i$ is computed from the $\hat{\boldsymbol{\beta}}$ corresponding to λ, even for $\mathbf{H}_0$. Write a loop to grid search for the λ minimizing BIC. Search over $\log\lambda$ values generated by `seq(-13,-7,length=50)`.

5. What is the uncertainty associated with $\hat{f}$? One way to assess this is non-parametric bootstrapping[2]. If n is the original sample size, we resample n day-death data pairs from the original data with replacement and refit the model to this resampled dataset. Repeating this multiple times builds up an estimate of the uncertainty in $\hat{\boldsymbol{\beta}}$ and hence anything derived from $\hat{\boldsymbol{\beta}}$. The easiest way to do this is to recognise that the bootstrap resampling is exactly equivalent to re-weighting the terms in the log likelihood, so that it becomes $\sum_i w_i l_i$, where the w_i take values 0, 1, 2, ... according to how many times the corresponding day-death pair was resampled. This means that we do not have to actually resample the data or recompute $\mathbf{X}$ etc for each sample. All we need to do is generate bootstrap weights and rework the negative log likelihood and derivative functions to use them. Generating the weights is easy:
   `wb <- tabulate(sample(n,replace=TRUE),n) ## non-para bootstrap weights`
   So, assuming λ is fixed at the values you selected, write code to obtain 200 bootstrap replicates of $\hat{f}$.

[1] log factorials can be evaluated using the `lgamma` function in R, given that the $\Gamma(y+1) = y!$, but you could just well drop the factorial term as it is independent of the parameters

[2] as covered in the exercise in section 6.1.1 of the notes and in section 16.1.

6. Finally produce a plot showing the daily death data against day of the year, with the model fit to these data overlaid, and the daily new infection rate f also shown with 95% confidence limits. Make sure that the axes have sensible limits and labels.

One piece of work - the text file containing your commented R code - is to be submitted for each group of 3 on Learn by 12:00 (noon) Friday 14th November 2025. You should supply a link to your github repo at the beginning of your submitted code. The repo should be made public shortly after the submission deadline. No extensions are available on this course, because of the frequency with which work has to be submitted. So late work will automatically attract a penalty (of 100% after work has been marked and returned). Technology failures will not be accepted as a reason for lateness (unless it is provably the case that Learn was unavailable for an extended period), so aim to submit ahead of time.

Marking Scheme: Full marks will be obtained for code that:

1. does what it is supposed to do, and has been coded in R approximately as indicated using base R (that is marks will be lost for simply finding a package or online code that simplifies the task for you).
2. is carefully commented, so that someone reviewing the code can easily tell exactly what it is for, what it is doing and how it is doing it without having read this sheet, or knowing anything else about the code. Note that *easily tell* implies that the comments must also be as clear and *concise* as possible. You should assume that the reader knows basic R, but not that they know exactly what every function in R does.
3. is well structured and laid out, so that the code itself, and its underlying logic, are easy to follow.
4. is reasonably efficient. As a rough guide the whole code should take at most a few 10s of seconds to run - much longer than that and something is probably wrong.
5. was prepared collaboratively using git and github in a group of 3, with the development process traceable on github (simply uploading fully or nearly completed work at or near the end does not do this).
6. contains no evidence of having been copied, in whole or in part, from anyone else (AI counts as anyone else for this purpose), other students on this course, students at other universities (there are now tools to help detect this, including internationally), online sources, ChatGPT etc.
7. includes a comment stating team member contributions.

Individual marks may be adjusted within groups if contributions are widely different.