Practical 3: Writing you own linear modelling functions

This practical is to be completed **individually**. What is submitted must be solely your own work. You can discuss the mathematical and statistical aspects of the practical, but code must not be shared with others. We have automatic systems available for checking for this, but in addition students tend to make distinctive errors in coding, or use distinctively convoluted solutions to problems: these tend to stand out even if you do the obvious things to try and hide the code sharing. You can use code from the lecture notes without citation, but if you use any other code that you did not write yourself you should cite where it came from.

In part this practical will be auto-marked. This means that it is essential that you stick exactly to the specification given here. Do not change specified function names, argument names or return names of elements of the returned objects. Also do not modify the format of something returned by an object where this is specified (e.g. if an object is supposed to be returned as a vector, do not return it as an object of class matrix - remember the function `drop`). This insistence on sticking to the specification is not just for marking convenience. When working on large collaborative projects working exactly to the agreed specification for inputs and outputs is essential, so it is a good habit to develop.

The aim of the project is to write your own functions for fitting linear models (using the QR decomposition approach), printing a simple summary of the fit, plotting a default residual plot and predicting the expected response, given new values of the predictors. The educational aim of the practical is to deepen your understanding of linear models and of classes and matrix computation in R. A good understanding of sections 9 to 11 of the notes is essential, as is familiarity with lists (5.2.3) and factor variables (5.2.4) in R. Obviously calling R's default modelling routines is not allowed for the purposes of this practical (you can use them for testing your functions, of course, but not within your functions). You should write the following functions:

- `linmod(formula,dat)` takes a model formula, `formula`, specifying a linear model, and a data frame containing the corresponding data, `dat`. Estimates the specified linear model using the QR decomposition of the model matrix approach. Returns an object of class `"linmod"`, a list containing the following elements:

  1. `beta` the vector of least squares parameter estimates for the model. The elements of `beta` should be named to identify the model components that they relate to (as in R's `lm` function). (Hint: `colnames`, `names`.)
  2. `V` the estimated covariance matrix of the least squares estimators.
  3. `mu` the vector of expected values of the response variable according to the estimated model (also known as 'fitted values').
  4. `y` the vector containing the response variable. `y <- model.frame(formula,dat)[[1]]` will obtain this (model frames are beyond the scope of this course, so you can treat this line of code as a 'black box').
  5. `yname` the name of the response variable, obtainable via `yname <- all.vars(formula)[1]`.
  6. `formula` the model formula.
  7. `flev` a named list. For each factor variable in `dat`, `flev` contains an item which is the vector of levels of the factor - the item's name should be the name of the factor variable. You need `flev` to predict properly with factors.
  8. `sigma` the estimated standard deviation, $\hat{\sigma}$, of the response (and residuals).

- `print.linmod(x,...)` — a print method function. `x` is an object of class `"linmod"`. The function should give the model formula defining the model and report the parameter estimates and their standard deviations, in a format like the one shown in this example (hint: `cbind`, `colnames`).

```
dist ~ speed + I(speed^2)

             Estimate        s.e.
(Intercept) 2.4701378 14.81716473
speed       0.9132876  2.03422044
I(speed^2)  0.0999593  0.06596821
```

- `plot.linmod` — a plot method function. `x` is an object of class `"linmod"`. The function plots the model residuals ($\hat{\boldsymbol{\epsilon}} = \mathbf{y} - \hat{\boldsymbol{\mu}}$) against the model fitted values ($\hat{\boldsymbol{\mu}} = \mathbf{X}\hat{\boldsymbol{\beta}}$). The y axis should be labelled "residuals" and the x axis "fitted values". A dashed horizontal line across the plot should indicate $\hat{\epsilon} = 0$.

- `predict.linmod(x,newdata)` — a predict method function. `x` is an object of class `"linmod"`, `newdata` is a data frame containing values of the predictor variables for which predictions of the response variable are required. The function should return a vector of predictions (one element for each row of `newdata`). You can of course use the `model.matrix` function with `x`'s model formula and `newdata` to produce the matrix mapping the model parameter estimates. Note however that some pre-processing of `newdata` is needed for correct functioning.

    1. `newdata` need not contain the response variable, which is not needed for prediction, but `x$formula` contains the response variable, which will cause `model.matrix` to complain if it is not supplied. You can deal with this by simply adding dummy response data to `newdata`. Note that if `foo` is a list, then `foo[["bar"]]` will result in the value `NULL` if `foo` does not have an element called `"bar"`. Function `is.null` is the usual way to test for `NULL`.
    2. It is not guaranteed that a user will supply factor variables with all the levels present that were present in fitting. Indeed they might only provide factor levels as character strings. Hence for any variable that was a factor in the original model fit, you should ensure that the corresponding variable in `newdata` is a factor with the same levels. (Hint: use `factors` and its `levels` argument.)

You should ensure that you have carefully tested your functions. You do not need to worry about careful handling of NAs for this project.

What to submit: One text file of carefully commented code containing the required functions. The file can contain other functions in addition to those listed, but should include no code outside functions. The file should be called P3xx.R where 'xx' (any number of digits) is replaced by your university number. The first line of the code comments should include your name and number. A github repo is not required for this project. However you might still want to use one to back up your work, as losing your work and hardware problems are not valid reasons for not submitting. However do not include a repo link in your submission, and do not invite me to your repo. The deadline is 16:00 Friday 5th November. It is strongly recommended that you submit well in advance of this, as last minute computer glitches will not be taken as an excuse for late submission[1].

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

1. is clearly commented, well laid out and well structured.
2. is reasonably clean concise and efficient, and avoids significant inefficiencies, such as using `solve` in this project, or very inefficient matrix computation.
3. passes a variety of automated model fitting and prediction checks. First class/distinction level marks *require* that prediction with factors works correctly.
4. sticks exactly to the specification.
5. is coded only using what is available in base R, without obvious 'cheats' that go against the spirit of the assignment (such as using `lm` within the required functions - using `lm` within a function to test your function is fine, of course).
6. has been produced individually - collaborating on code is not acceptable for this project.
7. avoids significant faults not covered by the above!

[1] This is for reasons of fairness - otherwise it becomes too easy to game the system for an extension.