# Quick start guide for the **grpreg** package

Patrick Breheny

April 7, 2018

This guide is intended to briefly demonstrate the basic usage of **grpreg**. For more details, see the other vignettes, documentation for individual functions, and the references.

**grpreg** comes with an example data set, **Birthwt**. The outcome, **Birthwt$bwt**, records the birth weights (in kg) of 189 babies. The following predictors are available:

```
> data(Birthwt)
> head(Birthwt$X, n=3)

            age1         age2        age3        lwt1        lwt2
[1,] -0.05833434  0.011046300  0.02956182  0.12446282 -0.02133871
[2,]  0.13436561  0.055245529 -0.09690705  0.06006722 -0.06922831
[3,] -0.04457006 -0.009415469  0.04508877 -0.05918388  0.03746349
             lwt3 white black smoke ptl1 ptl2m ht ui ftv1 ftv2 ftv3m
[1,] -0.130731102     0     1     0    0     0  0  1    0    0     0
[2,] -0.033348413     0     0     0    0     0  0  0    0    0     1
[3,]  0.004618178     1     0     1    0     0  0  0    1    0     0
```

This is a design matrix derived from the original data set, in which several terms have been expanded. For example, there are multiple indicator functions for race ("other" being the reference group) and several continuous factors such as age have been expanded using polynomial contrasts (splines would give a similar structure). Hence, the columns of the design matrix are *grouped*; this is what grpreg is designed for. The grouping information is encoded as follows:

```
> Birthwt$group

 [1] age   age   age   lwt   lwt   lwt   race  race  smoke ptl   ptl
[12] ht    ui    ftv   ftv   ftv
Levels: age lwt race smoke ptl ht ui ftv
```

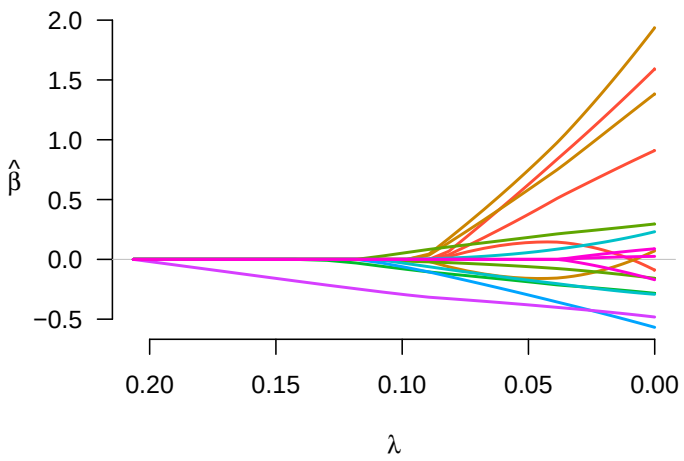
Here, groups are given as a factor; unique integer codes (which are essentially unlabeled factors) and character vectors are also allowed[1]. To fit a group lasso model to this data:

```
> X <- Birthwt$X
> y <- Birthwt$bwt
> group <- Birthwt$group
> fit <- grpreg(X, y, group, penalty="grLasso")
```

We can then plot the coefficient paths with

```
> plot(fit)
```

[1] Character vectors are not an ideal choice, however, as the order of the groups is left unspecified, which can lead to ambiguity if you also try to set the **group.multiplier** option.



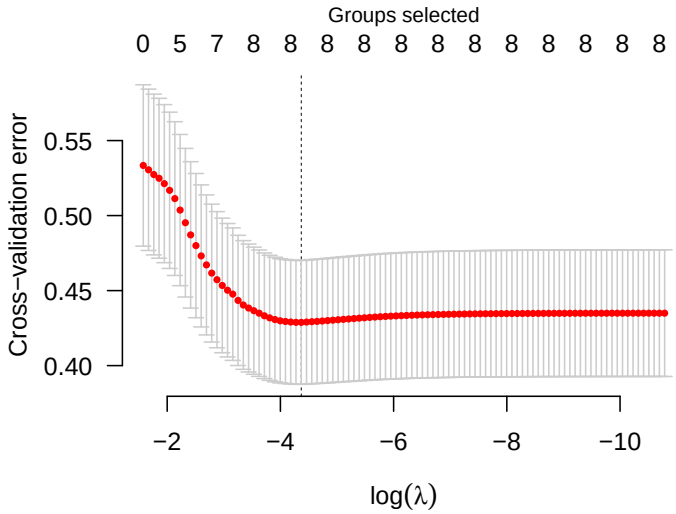

Notice that when a group enters the model (e.g., the green group), all of its coefficients become nonzero; this is what happens with group lasso models. To see what the coefficients are, we could use the `coef` function:

```
> coef(fit, lambda=0.05)

(Intercept)        age1        age2        age3        lwt1        lwt2
 3.02892181  0.14045229  0.62608119  0.37683684  0.74715315 -0.15825582
       lwt3       white       black       smoke        ptl1       ptl2m
 0.58290856  0.18344777 -0.06107624 -0.18778377 -0.17422515  0.05710668
         ht          ui        ftv1        ftv2       ftv3m
-0.29776948 -0.38050822  0.00000000  0.00000000  0.00000000
```

Note that the number of physician's visits (`ftv`) is not included in the model at $\lambda = 0.05$. Typically, one would carry out cross-validation for the purposes of carrying out inference on the predictive accuracy of the model at various values of $\lambda$.

```
> cvfit <- cv.grpreg(X, y, group, penalty="grLasso")
> plot(cvfit)
```




The coefficients corresponding to the value of $\lambda$ that minimizes the cross-validation error can be obtained via `coef`:

```
> coef(cvfit)

(Intercept)        age1        age2        age3        lwt1        lwt2
 3.04213663  0.02666724  1.33468687  0.78302016  1.59845996 -0.03813407
       lwt3       white       black       smoke        ptl1       ptl2m
 1.17444357  0.26743587 -0.12754742 -0.25835460 -0.26807555  0.17274876
         ht          ui        ftv1        ftv2       ftv3m
-0.49583076 -0.45267804  0.06311539  0.02029020 -0.10242061
```

Predicted values can be obtained via `predict`, which has a number of options:

```
> predict(cvfit, X=head(X))

[1] 2.544473 3.000644 3.045276 2.562359 2.576351 3.084683

> predict(cvfit, type="ngroups")

[1] 8
```

Note that the original fit (to the full data set) is returned as `cvfit$fit`; it is not necessary to call both `grpreg` and `cv.grpreg` to analyze a data set. Several other penalties are available, as are methods for logistic regression and Cox proportional hazards regression.