The "parameters" Class: Generalizing Common Components of Fitted Statistical Models

Matthew Owen

September 3, 2010

1 Functionality of the *parameters* Class

Several general features - sampling distribution, link function, systematic component, ancillary parameters, etc. - comprise statistical models. These features, while vastly differing between any two given specific models, share features that are easily classifiable, and usually necessary in the simulation of *quantities of interest*. That is, all statistical models have similar traits, and can be simulated using similar methods. Using this fact, the *parameters* class provides a set of functions and data-structures to aid in the planning and implementation of

2 Components of a *parameters* Object

simulations A vector or matrix of random draws taken from the model's distribution function. For example, a logit model will take random draws from a Multivariate Normal distribution.

alpha A vector specifying parameters to be passed into the distribution function. Values for this range from scaling factors to statistical means.

fam An optional parameter. *fam* must be an object of class "family". This allows for the implicit specification of the link and link-inverse function. It is recommend that the developer set either this, the link, or the linkinv parameter explicitly. Setting the family object implicitly defines *link* and *linkinv*.

link An optional parameter. *link* must be a function. Setting the link function explicitly is useful for defining arbitrary statistical models. *link* is used primarily to numerically approximate its inverse - a necessary step for simulating *quantities of interest*.

linkinv An optional parameter. *linkinv* must be a function. Setting the link's inverse explicitly allows for faster computations than a numerical approximation provides. If the inverse function is known, it is recommended that this function is explicitly defined.

3 Methods of the *parameters* Object

alpha Extracts the contents of alpha

coef Extracts the random samples from

link Extracts the link function from the parameter. This value exists as long as *fam* or *link* are explicitly set

linkinv Extracts the link-inverse function from the parameters. This value exists as long as *fam*, *link* or *linkinv* are explicitly set. If *linkinv* is not explicitly set, then a numerical approximation is used based on *fam* or *link*

4 Writing the *param* Method

The "param" function of an arbitrary Zelig model draws samples from the model, and describes the statistical model. In practice, this may be done in a variety of fashions, depending upon the complexity of the model.

4.1 Simplest Method: Returning a Vector/Matrix of Samples

Simply returning a vector of samples from the fitted model is by far the simplest approach to constructing a "param" function. While it is quick-and-easy to implement, it lacks the descriptiveness of other methods, and restricts the ability of the developer to make use of API functions. While this my be an insignificant loss to some, it typically will result in the developer writing harder-to-read code. It is recommended that the developer that the developer use the slightly more sophisticated methods when possible.

4.1.1 Example Code for Simple Vector/Matrix Method

```
param.default <- function(z, x, x1=NULL, num=num) {
  mvrnorm(n=num, mu=coef(z), Sigma=vcov(z))
}
```

4.1.2 Explanation

"mvrnorm" is the function that takes samples from a multivariate-normal distribution. In the above example, the *param.default* function returns a matrix with row size *num* and column size equivalent to that of the variance-covariance matrix of the fitted model. Zelig knows how to convert matrices of this type to a parameters object.

4.2 List Method: Returning an Indexed List of Parameters

While the simple method of returning a vector or matrix from a *param* function is extremely simple, it has no method for setting link or link-inverse functions for use within the actual simulation process. That is, it does not provide a clear, easy-to-read method for simulating *quantities of interest*. By returning an indexed list - or a parameters object - the developer can provide clearly labeled and stored link and link-inverse functions, as well as, ancillary parameters.

4.2.1 Example of Indexed List Method with *fam* Object Set

```
param.logit <- function(z, x, x1=NULL, num=num)
  list(
       coef  = mvrnorm(n=num, mu=coef(z), Sigma=vcov(z)),
       alpha = NULL,
       fam   = binomial(link="logit")
       )
```

4.2.2 Explanation of Indexed List with *fam* Object Set Example

The above example shows how link and link-inverse functions (for a "logit" model) can be set using a "family" object. Family objects exist for most statistical models - logit, probit, normal, Gaussian, et cetera - and come preset with values for link and link-inverses. This method does not differ immensely from the simple, vector-only method; however, it allows for the use of several API functions - *link*, *linkinv*, *coef*, *alpha* - that improve the readability and simplicity of the model's implementation.

The *param* function and the *parameters* class offer methods for automating and simplifying a large amount of repetitive and cumbersome code that may come with building the arbitrary statistical model. While both are in principle entirely optional - so long as the *qi* function is well-written - they serve as a means to quickly and elegantly implement Zelig models.

4.2.3 Example of Indexed List Method (with *link* Function) Set

```
param.poisson <- function(z, x, x1=NULL, num=num) {
```

```
  list(
       coef = mvrnorm(n=num, mu=coef(z), Sigma=vcov(z)),
       link = log,

       # because ``link'' is set,
       # the next line is purely optional
       linkinv = exp
       )
}
```

4.2.4 Explanation of Indexed List (with *link* Function) Example

The above example shows how a *parameters* object can be created with by explicitly setting the statistical model's link function. The *linkinv* parameter is purely optional, since Zelig will create a numerical inverse if it is undefined. However, the computation of the inverse is typically slower than non-iterative methods. As a result of this, if the link-inverse is known, it should be set, using the *linkinv* parameter.

The above example can also contain an *alpha* parameter, in order to store important ancillary parameters - mean, standard deviation, gamma-scale, etc. - that would be necessary in the computation of *quantities of interest*.

5 Using a *parameters* Object

Typically, a *parameters* object is used within a model's *qi* function. While the developer can typically omit the *param* function and the *parameters* object, it is not recommended. This is because making use of this function can vastly improve readability and functionality of a Zelig model. That is, *param* and *parameters* automate a large amount of repetitive, cumbersome code, and offer allow access to an easy-to-use API.

5.1 Example *param* Function

```
qi.logit <- function(z, x, x1=NULL, sim.param=NULL, num=1000) {
  coef <- coef(sim.param)
  inverse <- linkinv(sim.param)

  eta <- coef %*% t(x)
  theta <- link.inverse(eta)

  # et cetera...
}
```

5.2 Explanation of Above *qi* Code

The above is a portion of the actual code used to simulate *quantities of interest* for a "logit" model. By using the sim.par object, which is automatically passed into the function if a *param* function is written, *quantities of interest* can be computed extremely generically. The step-by-step process of the above function is as follows:

- Assign the simulations from *param.logit* to the variable "coef"
- Assign the link-inverse from *param.logit* to the variable "inverse"
- Compute η (eta) by matrix-multiplying our simulations with our explanatory results
- Conclude "simulating" the *quantities of interest* by applying the inverse of the link function. The result is a vector whose median is an approximate value of the *quantity of interest* and has a standard deviation that will define the confidence interval around this value

6 Future Improvements

In future releases of Zelig, *parameters* will have more API functions to facilitate common operations - sample drawing, matrix-multiplication, et cetera - so that the developer's focus can be exclusively on implementing important components of the model.