3.1 Using Perturbation Analysis in R

We have developed a package in R that makes perturbation-based sensitivity analysis simple to apply and to interpret. For most models this running a sensitivity analysis involves only two steps.

1. Specify the data, model, and model options for the unperturbed model, and optionally, the error functions for the perturbation.

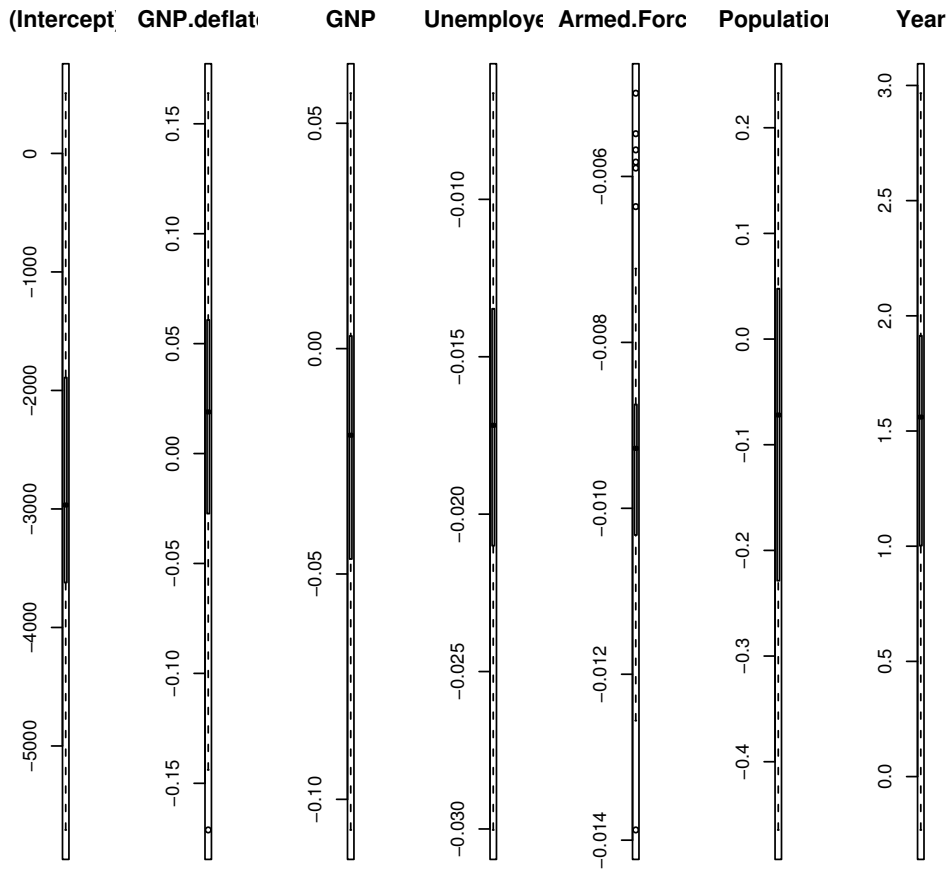
2. Use `summary()` or `plot(summary())` to see the sensitivity of the parameter estimates to perturbations.

Perturb works automatically almost with any **R** model, such as `lm`, `glm`, and `nls`, that accepts `data` as an argument to supply data and that returns estimated coefficients through `coef()`.

The example below shows how to conduct a sensitivity analysis of the classic analysis by Longley [1964] using `sensitivity()` and default noise functions.

```
> library(accuracy)
> data(longley)
> plongley = sensitivity(longley, lm, Employed ~ .)
> sp = summary(plongley)
> print(sp)

Sensitivity of coefficients to perturbations:
                      mean       stderr           min          2.5%
(Intercept)  -2.816858e+03 1.247001e+03 -5.708401e+03 -4.598391e+03
GNP.deflator  1.453037e-02 7.019360e-02 -1.712023e-01 -1.376388e-01
GNP          -1.902816e-02 3.590807e-02 -1.067855e-01 -7.660611e-02
Unemployed   -1.728276e-02 5.231649e-03 -3.003093e-02 -2.518958e-02
Armed.Forces -9.256278e-03 1.892578e-03 -1.387656e-02 -1.249442e-02
Population   -8.269447e-02 1.796583e-01 -4.645566e-01 -4.140411e-01
Year          1.486675e+00 6.417051e-01 -2.315547e-01  3.714535e-02
                     97.5%           max
(Intercept)  -21.445918706 508.139158286
GNP.deflator   0.125940789   0.163924128
GNP            0.051912542   0.056673221
Unemployed    -0.007318374  -0.006627787
Armed.Forces  -0.005528618  -0.004996486
Population     0.213444854   0.232697310
Year           2.405856671   2.966351120
```

```
Sensitivity of stderrs to perturbations:
                     mean       stderr          min         2.5%        97.5%
(Intercept)  9.639239e+02 1.825737e+02 5.055049e+02 6.088589e+02 1.245687e+03
GNP.deflator 9.875113e-02 2.650882e-02 4.876198e-02 5.579313e-02 1.572639e-01
GNP          3.242529e-02 7.092493e-03 1.612093e-02 1.717666e-02 4.676840e-02
Unemployed   4.825799e-03 1.002236e-03 2.252507e-03 2.844293e-03 6.992892e-03
Armed.Forces 2.676993e-03 5.265342e-04 1.417451e-03 1.654835e-03 3.769170e-03
Population   2.245257e-01 6.278467e-02 1.151731e-01 1.350575e-01 3.544378e-01
Year         4.962089e-01 9.307081e-02 2.606958e-01 3.157326e-01 6.401086e-01
                      max
(Intercept)  1.359574e+03
GNP.deflator 1.670187e-01
GNP          4.805988e-02
Unemployed   7.172210e-03
Armed.Forces 3.871694e-03
Population   4.305722e-01
Year         6.943244e-01

> plot(sp)
```



This is a rare example of a model that is very sensitive to noise. Even so, note that the small amounts of noise applied tremendously alter some of the estimated coefficients, but not others. In most practical cases, however, the substantive implications of your model will remain the same across the sensitivity analysis – in which case, you can publish them with greater confidence.

If you do not specify error functions, a default error function will be selected based on measurement type of the variable: continuous, ordered, or unordered. Continuous variables, by default are subject to a small amount of mean-zero component-wise uniformly distributed noise, which is typical of instrumentation-driven measurement error. Ordered factors are assigned a small probability of having observations reclassified to the neighboring classification, and unordered factors have a small probability of being reassigned to another legal value.

Alternatively, one can specify the error functions to use yourself, or use one of many supplied by *accuracy*. The *accuracy* package comes with a wide range of noise functions for continuous distributions, and random reclassification of factors. [2]

[2]The `perturb` module for collinearity diagnosis by Hendrickx, et. al [2004] (which was developed for `R` after the `accuracy` module) provides additional methods for randomly reclassifying factors that via its `reclassify()` function.

Your choice of error functions should be chosen to reflect measurement error model for the specific data you are using. In numerical analysis, uniform noise is often used since this is what would be expected from simple rounding error. Normal random noise is commonly used in statistics, under the assumption that measurement error is the sum of multiple independent error processes. In addition, when normal perturbations are used, the result can be interpreted, for many models, as equivalent to the results of running a slightly perturbed *model* on unperturbed data. In some cases, like discrete or ratio variables, other forms of noise are necessary to preserve the structure of the problem. [see for example, Altman, Gill, McDonald 2005]. Noise can. The magnitude of the noise is also under the control of the researcher. Most use a magnitude equivalent to the researchers estimate of the underlying measurement error in the data. Noise is usually adjusted to the size of each component, since this better preserves the structure of the problem, however in some cases the underlying measurement error model may imply norm-wise scaling of the noise. For more information on noise distributions and measurement error models see , e.g., Belsley 1991, Chaitin-Chatelin & Traviesas-Caasan [2004b], Caroll et. al [1995], Cheng & Van Ness [1999], Fuller [1987].

If multiple plausible measurement error models can be hypothesized for your data, we recommend that you run `perturb` multiple times with different noise specifications, However, in our experience with social science analyses, the choice of error model does not tend to effect, in practice, the substantive conclusions from the sensitivity analysis.

Some researchers omit perturbations to outcome variables, since, in terms of statistical theory, mean-zero measurement error on outcome variables (as opposed to explanatory variables) contribute only to increased variance in estimates, not bias. While this attitude is well-justified in the context of statistical theory, it is not similarly justified in the computational realm. If the estimation of a model is computationally unstable, errors in the outcome variable may have large and unpredictable biases on the model estimate. Hence, the conservative default in our module is to subject all variables to perturbation, although you can change this.

Consider this example, which shows a sensitivity analysis of the anorexia analysis described in Venables and Ripley [2002]. In this case, we leave the dependent variable unperturbed, by assigning it the *identity* error function.

```
> data(anorexia, package = "MASS")
> panorexia = sensitivity(anorexia, glm, Postwt ~ Prewt + Treat +
+     offset(Prewt), family = gaussian, ptb.R = 100, ptb.ran.gen = c(PTBi,
+     PTBus, PTBus), ptb.s = c(1, 0.005, 0.005))
> summary(panorexia)
```

This function can be used in conjunction with `accuracy`. Hendrickx, et. al also provide a number of collinearity diagnostics, including one based on data perturbations.

```
Sensitivity of coefficients to perturbations:
                  mean      stderr        min       2.5%      97.5%        max
(Intercept) 49.7570055 0.400507716 48.5181534 49.0224197 50.3636033 50.4646899
Prewt       -0.5653872 0.004852886 -0.5743432 -0.5732861 -0.5563066 -0.5506232
TreatCont   -4.0975142 0.035348062 -4.2054884 -4.1730985 -4.0308849 -4.0128961
TreatFT      4.5641063 0.042065546  4.4577757  4.4870128  4.6527536  4.6774678


Sensitivity of stderrs to perturbations:
                  mean       stderr        min       2.5%      97.5%        max
(Intercept) 13.3889029 0.0518627561 13.2655519 13.2790441 13.4851746 13.5380041
Prewt        0.1611573 0.0006352277  0.1596833  0.1598193  0.1622761  0.1629719
TreatCont    1.8935009 0.0039662784  1.8842722  1.8864319  1.9019317  1.9045316
TreatFT      2.1333835 0.0044430846  2.1230710  2.1249585  2.1423483  2.1461762
```

Finally, if a model in **R** does not take a `data` argument or does not return coefficients through the `coef` method, it is usually only a matter of a few minutes to write a small wrapper that calls the original model with appropriate data, and that provides a `coef` method for retrieving the results. (Alternatively, you might to choose to run such models in **Zelig**, as described in the next section.)

For example, the `mle` function for maximum-likelihood estimation does not have an explicit `data` option. Instead, it normally receives data implicitly through the log-likelihood function, `ll`, passed into it. To adapt it for use in `perturb` we simply construct a another function that accepts data and a log-likelihood function separately, constructs a temporary log-likelihood function with the data passed in the environment, and then calls `mle` with the temporary function.

```
> mleD <- function(data, lld, ...) {
+     f = formals(lld)
+     f[1] = NULL
+     ll <- function() {
+         cl = as.list(match.call())
+         cl[1] = NULL
+         cl$data = as.name("data")
+         do.call(lld, cl)
+     }
+     formals(ll) = f
+     mle(ll, ...)
+ }
```

Finally, construct the log-likelihood function to accept data. As in this example, which is based on the documented example in the **Stats4** package:

```
> library(stats4)
> dat = as.data.frame(cbind(0:10, c(26, 17, 13, 12, 20, 5, 9, 8,
+     5, 4, 8)))
> llD <- function(data, ymax = 15, xhalf = 6) -sum(stats::dpois(data[[2]],
+     lambda = ymax/(1 + data[[1]]/xhalf), log = TRUE))
> summary(sensitivity(dat, mleD, llD))

Sensitivity of coefficients to perturbations:
           mean    stderr       min      2.5%     97.5%      max
ymax  24.939287 0.6009417 20.991162 24.676434 25.140677 26.22968
xhalf  3.072633 0.1608800  2.884105  2.988534  3.094082  4.16676
```

3.2 Perturbation Analysis Using Zelig

Zelig [King, et. al 2005] is an easy-to-use R package that can estimate and help interpret the results of a large range of statistical models. Zelig provides a uniform interface to these models, that the **Accuracy** package can utilize to enable sensitivity analyses. Another advantage of Zelig is that it provides uniform tools for computing and visualizing quantities of real interest (emulating the popular **Clarify** package for **Stata** [Tomz, Wittenberg and King 2003]), such as predicted values, expected values, first differences, and risk ratios. Using **Accuracy** and **Zelig** the researcher can easily compute the sensitivity of such predicted values (etc.) to measurement error. [3]

To illustrate, we replicate the sensitivity analysis of Longley's model (above), using Zelig. Instead of specifying the data and model in `perturb` we specify it in **Zelig**, and we use the convenience function `sensitivityZelig()` to run the sensitivity analysis

```
> library(Zelig)

Loading required package: MASS

Attaching package: 'MASS'


        The following object(s) are masked _by_ .GlobalEnv :
```

[3] **Zelig** also integrates nonparametric matching methods as an optional preprocessing step. Thus **Accuracy** supports sensitivity analysis of models subject to such pre-processing as well.

```
         anorexia

Loading required package: boot
##
##  Zelig (Version 2.4-5, built: 2005-10-18)
##  Please refer to http://gking.harvard.edu/zelig for full documentation
##  or help.zelig() for help with commands and models supported by Zelig.
##
```

```
> zelig.out = zelig(Employed ~ ., "ls", longley)
> perturb.zelig.out = sensitivityZelig(zelig.out)
```

Just as above, `summary()` and `plot(summary())` summarize the sensitivity of the model coefficients. In addition, we can use the **Zelig** methods `setx` and `sim` to simulate various quantities of interest, which we display by using `summary()` and `plot()` on the output of `sim`.

In more detail, this code generates sensitivity estimates of the parameter coefficients:

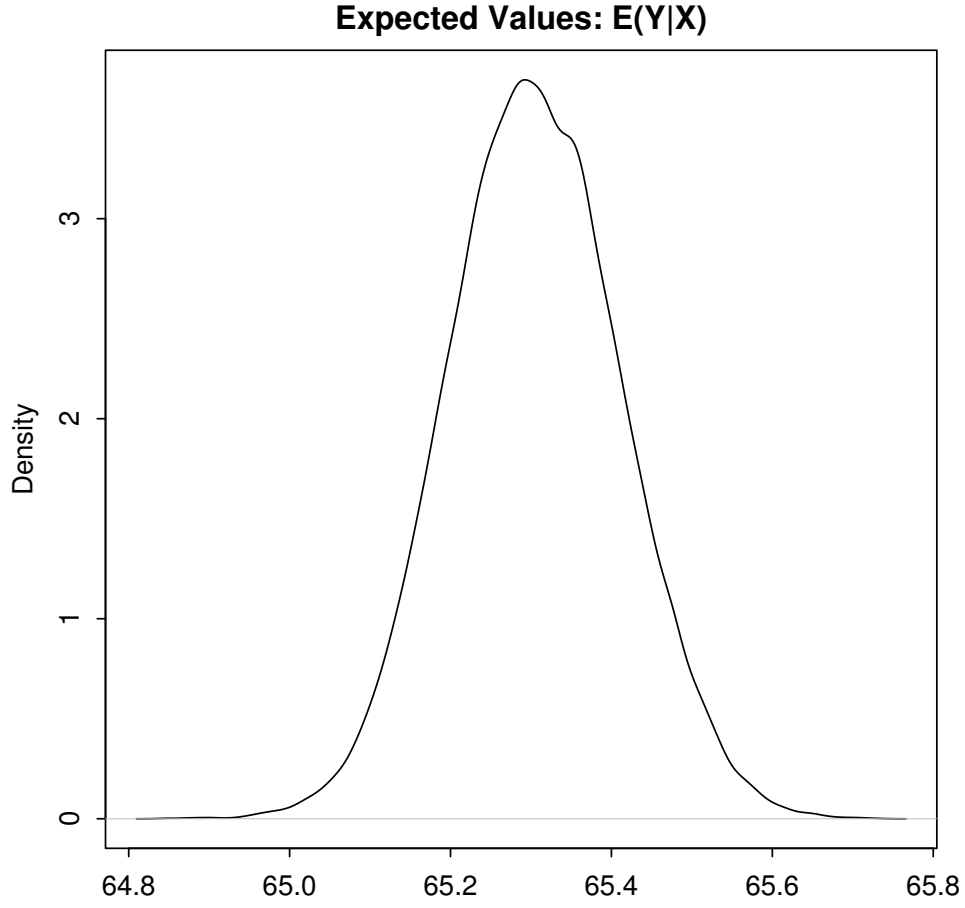
```
> summary(perturb.zelig.out)

Sensitivity of coefficients to perturbations:
                   mean    stderr        min       2.5%      97.5%        max
(Intercept)  -3.597e+03 8.639e+02 -4.883e+03 -4.741e+03 -1.792e+03 -1.557e+03
GNP.deflator  2.321e-02 5.969e-02 -7.371e-02 -5.720e-02  1.203e-01  1.236e-01
GNP          -4.367e-02 2.801e-02 -9.428e-02 -9.113e-02  6.424e-03  2.158e-02
Unemployed   -2.145e-02 4.118e-03 -2.920e-02 -2.826e-02 -1.430e-02 -1.200e-02
Armed.Forces -1.052e-02 1.436e-03 -1.289e-02 -1.258e-02 -7.923e-03 -7.055e-03
Population    2.134e-02 1.813e-01 -3.041e-01 -2.252e-01  3.923e-01  4.251e-01
Year          1.885e+00 4.435e-01  8.171e-01  9.613e-01  2.480e+00  2.536e+00


Sensitivity of stderrs to perturbations:
                  mean    stderr       min      2.5%     97.5%       max
(Intercept)  9.146e+02 1.909e+02 4.862e+02 5.635e+02 1.234e+03 1.277e+03
GNP.deflator 8.415e-02 1.476e-02 5.227e-02 6.027e-02 1.080e-01 1.086e-01
GNP          3.022e-02 7.205e-03 1.791e-02 1.985e-02 4.523e-02 5.435e-02
Unemployed   4.566e-03 1.013e-03 2.618e-03 3.021e-03 6.676e-03 7.625e-03
Armed.Forces 2.394e-03 4.363e-04 1.479e-03 1.611e-03 3.110e-03 3.228e-03
Population   1.989e-01 5.507e-02 1.230e-01 1.317e-01 3.226e-01 3.850e-01
Year         4.702e-01 9.754e-02 2.507e-01 2.912e-01 6.323e-01 6.492e-01
```

While this code generates predictions of the distribution of the explanatory variable, 'Employed', around the point where 'Year' equals 1955, and the other variables are at their means:

```
> setx.out = setx(perturb.zelig.out, Year = 1955)
> sim.perturb.zelig.out = psim(perturb.zelig.out, setx.out)
> summary(sim.perturb.zelig.out)

**** 30  COMBINED perturbation simulations

  Model: ls
  Number of simulations: 1000

Values of X
     (Intercept) GNP.deflator   GNP Unemployed Armed.Forces Population Year
1947           1        101.7 387.7      319.3        260.7      117.4 1954

Expected Values: E(Y|X)
     mean     sd 2.5% 97.5%
1947 65.3 0.1064 65.1 65.51
```

This creates a profile plot of the predicted distribution of the explanatory variable:

```
> plot(sim.perturb.zelig.out)

**** 30  COMBINED perturbation simulations
```



Zelig currently has several dozen models already available. For example we could easily have run the same analysis with a 'normal bayesian' regression model instead of OLS as follows, just by substituting `zelig(Employed ~.,"normal.bayes",longley)` for `zelig(Employed ~.,"ls",longley)`. Finally it is to add new models to Zelig, by providing a few simple methods and interface function. Although the effort to do so is slightly greater than the approach above, writing a **Zelig** interface has the advantage of enabling a variety of additional quantities of interest to be easily computed.

4 Using Universal Numeric Fingerprints (UNF's) to Ensure Data Integrity

In the course of past research [see, e.g., Altman, Gill, McDonald 2003], we performed numerical benchmark tests on over twenty different software packages and versions of packages. An unexpected lesson from this testing is that loading data into a statistical software package from text or binary files–a seemingly straightforward operation–may produce unexpected results. We discovered many cases where a data file that was saved without error in one package was interpreted in unintended