Post–`parmest` peripherals: `fvregen`, `invcise`, and `qqvalue`

With examples from the Avon Longitudinal Study of Parents and Children (ALSPAC) cohort study at Bristol University, UK
http://www.bristol.ac.uk/alspac/

Roger B. Newson
r.newson@imperial.ac.uk
http://www.imperial.ac.uk/nhli/r.newson/

National Heart and Lung Institute
Imperial College London

16th UK Stata Users' Group Meeting, 9–10 September, 2010
Downloadable from the conference website at
http://ideas.repec.org/s/boc/usug10.html

What is a post–`parmest` peripheral?

- The `parmest` package (Newson, 2008[5]) contains 4 modules: `parmest`, `parmby`, `metaparm`, and `parmcip`.
- These are used with Stata estimation commands to create output datasets (or resultssets), with 1 observation per estimated parameter, and data on parameter names, estimates, confidence limits, *P*–values, and other parameter attributes.
- These resultssets are then input to other Stata packages ("peripherals"), to produce tables, listings, plots, and secondary resultssets, containing observations corresponding to derived parameters.
- Stata packages that input resultssets include `eclplot` (which produces confidence interval plots) and `smileplot` (which executes multiple–test procedures).
- *Usually*, post–`parmest` peripherals can also be post–`statsby` peripherals.

What is a post–`parmest` peripheral?

- The `parmest` package (Newson, 2008[5]) contains 4 modules: `parmest`, `parmby`, `metaparm`, and `parmcip`.
- These are used with Stata estimation commands to create output datasets (or resultssets), with 1 observation per estimated parameter, and data on parameter names, estimates, confidence limits, *P*–values, and other parameter attributes.
- These resultssets are then input to other Stata packages ("peripherals"), to produce tables, listings, plots, and secondary resultssets, containing observations corresponding to derived parameters.
- Stata packages that input resultssets include `eclplot` (which produces confidence interval plots) and `smileplot` (which executes multiple–test procedures).
- *Usually*, post–`parmest` peripherals can also be post–`statsby` peripherals.

What is a post–`parmest` peripheral?

- The `parmest` package (Newson, 2008[5]) contains 4 modules: `parmest`, `parmby`, `metaparm`, and `parmcip`.
- These are used with Stata estimation commands to create output datasets (or resultssets), with 1 observation per estimated parameter, and data on parameter names, estimates, confidence limits, *P*–values, and other parameter attributes.
- These resultssets are then input to other Stata packages ("peripherals"), to produce tables, listings, plots, and secondary resultssets, containing observations corresponding to derived parameters.
- Stata packages that input resultssets include `eclplot` (which produces confidence interval plots) and `smileplot` (which executes multiple–test procedures).
- *Usually*, post–`parmest` peripherals can also be post–`statsby` peripherals.

What is a post–`parmest` peripheral?

- The `parmest` package (Newson, 2008[5]) contains 4 modules: `parmest`, `parmby`, `metaparm`, and `parmcip`.
- These are used with Stata estimation commands to create output datasets (or resultssets), with 1 observation per estimated parameter, and data on parameter names, estimates, confidence limits, *P*–values, and other parameter attributes.
- These resultssets are then input to other Stata packages ("peripherals"), to produce tables, listings, plots, and secondary resultssets, containing observations corresponding to derived parameters.
- Stata packages that input resultssets include `eclplot` (which produces confidence interval plots) and `smileplot` (which executes multiple–test procedures).
- *Usually*, post–`parmest` peripherals can also be post–`statsby` peripherals.

What is a post–`parmest` peripheral?

- The `parmest` package (Newson, 2008[5]) contains 4 modules: `parmest`, `parmby`, `metaparm`, and `parmcip`.
- These are used with Stata estimation commands to create output datasets (or resultssets), with 1 observation per estimated parameter, and data on parameter names, estimates, confidence limits, *P*–values, and other parameter attributes.
- These resultssets are then input to other Stata packages ("peripherals"), to produce tables, listings, plots, and secondary resultssets, containing observations corresponding to derived parameters.
- Stata packages that input resultssets include `eclplot` (which produces confidence interval plots) and `smileplot` (which executes multiple–test procedures).
- *Usually*, post–`parmest` peripherals can also be post–`statsby` peripherals.

What is a post–`parmest` peripheral?

- The `parmest` package (Newson, 2008[5]) contains 4 modules: `parmest`, `parmby`, `metaparm`, and `parmcip`.
- These are used with Stata estimation commands to create output datasets (or resultssets), with 1 observation per estimated parameter, and data on parameter names, estimates, confidence limits, *P*–values, and other parameter attributes.
- These resultssets are then input to other Stata packages ("peripherals"), to produce tables, listings, plots, and secondary resultssets, containing observations corresponding to derived parameters.
- Stata packages that input resultssets include `eclplot` (which produces confidence interval plots) and `smileplot` (which executes multiple–test procedures).
- *Usually*, post–`parmest` peripherals can also be post–`statsby` peripherals.

Three recent post–`parmest` peripheral packages

- Recent post–`parmest` peripheral packages (added to SSC in 2009 and 2010) include `fvregen`, `invcise`, and `qqvalue`.
- The `fvregen` package regenerates factor variables (introduced in Stata Version 11) in `parmest` resultssets.
- The `invcise` package generates standard errors "backwards" from confidence limits produced without standard errors (such as those for medians and median differences).
- The `qqvalue` package inputs multiple *P*–values and calculates the corresponding *q*–values (or "adjusted *P*–values"), by inverting multiple–test procedures.
- Each of these packages is the subject of a section of this presentation.

Three recent post–`parmest` peripheral packages

- Recent post–`parmest` peripheral packages (added to SSC in 2009 and 2010) include `fvregen`, `invcise`, and `qqvalue`.
- The `fvregen` package regenerates factor variables (introduced in Stata Version 11) in `parmest` resultssets.
- The `invcise` package generates standard errors "backwards" from confidence limits produced without standard errors (such as those for medians and median differences).
- The `qqvalue` package inputs multiple *P*–values and calculates the corresponding *q*–values (or "adjusted *P*–values"), by inverting multiple–test procedures.
- Each of these packages is the subject of a section of this presentation.

Three recent post–`parmest` peripheral packages

- Recent post–`parmest` peripheral packages (added to SSC in 2009 and 2010) include `fvregen`, `invcise`, and `qqvalue`.
- The `fvregen` package regenerates factor variables (introduced in Stata Version 11) in `parmest` resultssets.
- The `invcise` package generates standard errors "backwards" from confidence limits produced without standard errors (such as those for medians and median differences).
- The `qqvalue` package inputs multiple P–values and calculates the corresponding q–values (or "adjusted P–values"), by inverting multiple–test procedures.
- Each of these packages is the subject of a section of this presentation.

Three recent post–`parmest` peripheral packages

- Recent post–`parmest` peripheral packages (added to SSC in 2009 and 2010) include `fvregen`, `invcise`, and `qqvalue`.
- The `fvregen` package regenerates factor variables (introduced in Stata Version 11) in `parmest` resultssets.
- The `invcise` package generates standard errors "backwards" from confidence limits produced without standard errors (such as those for medians and median differences).
- The `qqvalue` package inputs multiple P–values and calculates the corresponding q–values (or "adjusted P–values"), by inverting multiple–test procedures.
- Each of these packages is the subject of a section of this presentation.

Three recent post–`parmest` peripheral packages

- Recent post–`parmest` peripheral packages (added to SSC in 2009 and 2010) include `fvregen`, `invcise`, and `qqvalue`.
- The `fvregen` package regenerates factor variables (introduced in Stata Version 11) in `parmest` resultssets.
- The `invcise` package generates standard errors "backwards" from confidence limits produced without standard errors (such as those for medians and median differences).
- The `qqvalue` package inputs multiple *P*–values and calculates the corresponding *q*–values (or "adjusted *P*–values"), by inverting multiple–test procedures.
- Each of these packages is the subject of a section of this presentation.

Three recent post–`parmest` peripheral packages

- Recent post–`parmest` peripheral packages (added to SSC in 2009 and 2010) include `fvregen`, `invcise`, and `qqvalue`.
- The `fvregen` package regenerates factor variables (introduced in Stata Version 11) in `parmest` resultssets.
- The `invcise` package generates standard errors "backwards" from confidence limits produced without standard errors (such as those for medians and median differences).
- The `qqvalue` package inputs multiple *P*–values and calculates the corresponding *q*–values (or "adjusted *P*–values"), by inverting multiple–test procedures.
- Each of these packages is the subject of a section of this presentation.

The `fvregen` package: Regenerating factors in `parmest` resultssets

- Factor variable lists were introduced in Version 11 of Stata, *mostly* for use with estimation commands.
- They can be expanded into lists of "virtual variables", which are equal to products involving indicators of group membership, and which may be included as *X*–variables in regression models.
- These virtual variables have alien–looking names, which are inherited by the corresponding parameters of the regression model, and stored in the variable `parm` of the `parmest` resultsset.
- `fvregen` reads the `parm` variable, and regenerates the factor variables, possibly with help from `descsave` (a Stata program which writes Stata programs).
- `fvregen` therefore supersedes the old `factext` package, just as factor variable lists supersede `xi`.

The `fvregen` package: Regenerating factors in `parmest` resultssets

- Factor variable lists were introduced in Version 11 of Stata, *mostly* for use with estimation commands.
- They can be expanded into lists of "virtual variables", which are equal to products involving indicators of group membership, and which may be included as *X*–variables in regression models.
- These virtual variables have alien–looking names, which are inherited by the corresponding parameters of the regression model, and stored in the variable `parm` of the `parmest` resultsset.
- `fvregen` reads the `parm` variable, and regenerates the factor variables, possibly with help from `descsave` (a Stata program which writes Stata programs).
- `fvregen` therefore supersedes the old `factext` package, just as factor variable lists supersede `xi`.

The `fvregen` package: Regenerating factors in `parmest` resultssets

- Factor variable lists were introduced in Version 11 of Stata, *mostly* for use with estimation commands.
- They can be expanded into lists of "virtual variables", which are equal to products involving indicators of group membership, and which may be included as *X*–variables in regression models.
- These virtual variables have alien–looking names, which are inherited by the corresponding parameters of the regression model, and stored in the variable `parm` of the `parmest` resultsset.
- `fvregen` reads the `parm` variable, and regenerates the factor variables, possibly with help from `descsave` (a Stata program which writes Stata programs).
- `fvregen` therefore supersedes the old `factext` package, just as factor variable lists supersede `xi`.

The `fvregen` package: Regenerating factors in `parmest` resultssets

- Factor variable lists were introduced in Version 11 of Stata, *mostly* for use with estimation commands.
- They can be expanded into lists of "virtual variables", which are equal to products involving indicators of group membership, and which may be included as *X*–variables in regression models.
- These virtual variables have alien–looking names, which are inherited by the corresponding parameters of the regression model, and stored in the variable `parm` of the `parmest` resultsset.
- `fvregen` reads the `parm` variable, and regenerates the factor variables, possibly with help from `descsave` (a Stata program which writes Stata programs).
- `fvregen` therefore supersedes the old `factext` package, just as factor variable lists supersede `xi`.

The **`fvregen`** package: Regenerating factors in **`parmest`** resultssets

- Factor variable lists were introduced in Version 11 of Stata, *mostly* for use with estimation commands.
- They can be expanded into lists of "virtual variables", which are equal to products involving indicators of group membership, and which may be included as *X*–variables in regression models.
- These virtual variables have alien–looking names, which are inherited by the corresponding parameters of the regression model, and stored in the variable `parm` of the `parmest` resultsset.
- `fvregen` reads the `parm` variable, and regenerates the factor variables, possibly with help from `descsave` (a Stata program which writes Stata programs).
- `fvregen` therefore supersedes the old `factext` package, just as factor variable lists supersede `xi`.

The `fvregen` package: Regenerating factors in `parmest` resultssets

- Factor variable lists were introduced in Version 11 of Stata, *mostly* for use with estimation commands.
- They can be expanded into lists of "virtual variables", which are equal to products involving indicators of group membership, and which may be included as *X*–variables in regression models.
- These virtual variables have alien–looking names, which are inherited by the corresponding parameters of the regression model, and stored in the variable `parm` of the `parmest` resultsset.
- `fvregen` reads the `parm` variable, and regenerates the factor variables, possibly with help from `descsave` (a Stata program which writes Stata programs).
- `fvregen` therefore supersedes the old `factext` package, just as factor variable lists supersede `xi`.

Example: Mileage by repair record by car origin in the `auto` data

- In the `auto` data, we might want to know how car mileage in miles per gallon (`mpg`) is predicted by repair record (`rep78`).
- We might also suspect that the relationship is different in non–US models and in US models (indicated by the variable `foreign`).
- And we might also be willing to assume that the residual variation for mileages is the same in all value combinations of `foreign` and `rep78`.
- *So* we might use a homoskedastic regression model, with factor variables, to estimate separate effects of `rep78` in US models and in non–US models.
- We will plot these effects (and their confidence limits) against `rep78` for US models and for non–US models, using a 4-step programming process.

Example: Mileage by repair record by car origin in the `auto` data

- In the `auto` data, we might want to know how car mileage in miles per gallon (`mpg`) is predicted by repair record (`rep78`).
- We might also suspect that the relationship is different in non–US models and in US models (indicated by the variable `foreign`).
- And we might also be willing to assume that the residual variation for mileages is the same in all value combinations of `foreign` and `rep78`.
- *So* we might use a homoskedastic regression model, with factor variables, to estimate separate effects of `rep78` in US models and in non–US models.
- We will plot these effects (and their confidence limits) against `rep78` for US models and for non–US models, using a 4-step programming process.

Example: Mileage by repair record by car origin in the `auto` data

- In the `auto` data, we might want to know how car mileage in miles per gallon (`mpg`) is predicted by repair record (`rep78`).
- We might also suspect that the relationship is different in non–US models and in US models (indicated by the variable `foreign`).
- And we might also be willing to assume that the residual variation for mileages is the same in all value combinations of `foreign` and `rep78`.
- *So* we might use a homoskedastic regression model, with factor variables, to estimate separate effects of `rep78` in US models and in non–US models.
- We will plot these effects (and their confidence limits) against `rep78` for US models and for non–US models, using a 4-step programming process.

Example: Mileage by repair record by car origin in the `auto` data

- In the `auto` data, we might want to know how car mileage in miles per gallon (`mpg`) is predicted by repair record (`rep78`).
- We might also suspect that the relationship is different in non–US models and in US models (indicated by the variable `foreign`).
- And we might also be willing to assume that the residual variation for mileages is the same in all value combinations of `foreign` and `rep78`.
- *So* we might use a homoskedastic regression model, with factor variables, to estimate separate effects of `rep78` in US models and in non–US models.
- We will plot these effects (and their confidence limits) against `rep78` for US models and for non–US models, using a 4-step programming process.

Example: Mileage by repair record by car origin in the `auto` data

- In the `auto` data, we might want to know how car mileage in miles per gallon (`mpg`) is predicted by repair record (`rep78`).
- We might also suspect that the relationship is different in non–US models and in US models (indicated by the variable `foreign`).
- And we might also be willing to assume that the residual variation for mileages is the same in all value combinations of `foreign` and `rep78`.
- *So* we might use a homoskedastic regression model, with factor variables, to estimate separate effects of `rep78` in US models and in non–US models.
- We will plot these effects (and their confidence limits) against `rep78` for US models and for non–US models, using a 4-step programming process.

Example: Mileage by repair record by car origin in the `auto` data

- In the `auto` data, we might want to know how car mileage in miles per gallon (`mpg`) is predicted by repair record (`rep78`).
- We might also suspect that the relationship is different in non–US models and in US models (indicated by the variable `foreign`).
- And we might also be willing to assume that the residual variation for mileages is the same in all value combinations of `foreign` and `rep78`.
- *So* we might use a homoskedastic regression model, with factor variables, to estimate separate effects of `rep78` in US models and in non–US models.
- We will plot these effects (and their confidence limits) against `rep78` for US models and for non–US models, using a 4-step programming process.

Step 1: Save a description of the factors in a do–file

We start by inputting the `auto` data. We then run `descsave`, a "super" version of `describe`, on the factor variables:

```
. sysuse auto, clear;
(1978 Automobile Data)

. tempfile df0;

. descsave foreign rep78,
>   list(name type format vallab varlab, clean noobs)
>   do(`"`df0'"', replace);

    name      type   format   vallab   varlab
    foreign   byte   %8.0g    origin   Car type
    rep78     int    %8.0g             Repair Record 1978
```

`descsave` can list attributes of the variables being described, and/or save a temporary do–file, which can be used later to reconstruct these attributes for variables with the same names in another dataset.

Step 1: Save a description of the factors in a do–file

We start by inputting the `auto` data. We then run `descsave`, a "super" version of `describe`, on the factor variables:

```
. sysuse auto, clear;
(1978 Automobile Data)

. tempfile df0;

. descsave foreign rep78,
>   list(name type format vallab varlab, clean noobs)
>   do(`"`df0'"', replace);

    name      type   format   vallab   varlab
    foreign   byte   %8.0g    origin   Car type
    rep78     int    %8.0g             Repair Record 1978
```

`descsave` can list attributes of the variables being described, and/or save a temporary do–file, which can be used later to reconstruct these attributes for variables with the same names in another dataset.

Step 1: Save a description of the factors in a do–file

We start by inputting the `auto` data. We then run `descsave`, a "super" version of `describe`, on the factor variables:

```
. sysuse auto, clear;
(1978 Automobile Data)

. tempfile df0;

. descsave foreign rep78,
>   list(name type format vallab varlab, clean noobs)
>   do(`"`df0'"', replace);

    name       type   format   vallab   varlab
    foreign    byte   %8.0g    origin   Car type
    rep78      int    %8.0g             Repair Record 1978
```

`descsave` can list attributes of the variables being described, and/or save a temporary do–file, which can be used later to reconstruct these attributes for variables with the same names in another dataset.

Step 1: Save a description of the factors in a do–file

We start by inputting the `auto` data. We then run `descsave`, a "super" version of `describe`, on the factor variables:

```
. sysuse auto, clear;
(1978 Automobile Data)

. tempfile df0;

. descsave foreign rep78,
>   list(name type format vallab varlab, clean noobs)
>   do(`"`df0'"', replace);

    name       type    format   vallab   varlab
    foreign    byte    %8.0g    origin   Car type
    rep78      int     %8.0g             Repair Record 1978
```

`descsave` can list attributes of the variables being described, and/or save a temporary do–file, which can be used later to reconstruct these attributes for variables with the same names in another dataset.

Step 1: Save a description of the factors in a do–file

We start by inputting the `auto` data. We then run `descsave`, a "super" version of `describe`, on the factor variables:

```
. sysuse auto, clear;
(1978 Automobile Data)

. tempfile df0;

. descsave foreign rep78,
>   list(name type format vallab varlab, clean noobs)
>   do(`"`df0'"', replace);

    name       type    format    vallab    varlab
    foreign    byte    %8.0g     origin    Car type
    rep78      int     %8.0g               Repair Record 1978
```

`descsave` can list attributes of the variables being described, and/or save a temporary do–file, which can be used later to reconstruct these attributes for variables with the same names in another dataset.

Step 1: Save a description of the factors in a do–file

We start by inputting the `auto` data. We then run `descsave`, a "super" version of `describe`, on the factor variables:

```
. sysuse auto, clear;
(1978 Automobile Data)

. tempfile df0;

. descsave foreign rep78,
>   list(name type format vallab varlab, clean noobs)
>   do(`"`df0'"', replace);

    name      type   format   vallab   varlab
    foreign   byte   %8.0g    origin   Car type
    rep78     int    %8.0g             Repair Record 1978
```

`descsave` can list attributes of the variables being described, and/or save a temporary do–file, which can be used later to reconstruct these attributes for variables with the same names in another dataset.

Step 1: Save a description of the factors in a do–file

We start by inputting the `auto` data. We then run `descsave`, a "super" version of `describe`, on the factor variables:

```
. sysuse auto, clear;
(1978 Automobile Data)

. tempfile df0;

. descsave foreign rep78,
>   list(name type format vallab varlab, clean noobs)
>   do(`"`df0'"', replace);

    name       type   format   vallab   varlab
    foreign    byte   %8.0g    origin   Car type
    rep78      int    %8.0g             Repair Record 1978
```

`descsave` can list attributes of the variables being described, and/or save a temporary do–file, which can be used later to reconstruct these attributes for variables with the same names in another dataset.

Step 2: Run the regression, creating a **parmby** resultsset

We now use the `parmby` module of `parmest` to run a regression model, with an intercept for each car origin type, and an effect for each repair record (compared to the baseline of 5). *In this case*, the results will be saved in the memory (overwriting the original data), and not to a file:

```
. parmby
>   "regress mpg ibn.foreign ibn.foreign#ib(last).rep78, noconst",
>   omit empty format(estimate min* max* %8.2f p %-8.2g) norestore;

Command: regress mpg ibn.foreign ibn.foreign#ib(last).rep78, noconst
note: 1.foreign#1.rep78 identifies no observations in the sample
note: 1.foreign#2.rep78 identifies no observations in the sample
```

`parmby` calls `regress`, which begins to produce its output. Note that not all values of `rep78` are represented in non–US models.

Step 2: Run the regression, creating a `parmby` resultsset

We now use the `parmby` module of `parmest` to run a regression model, with an intercept for each car origin type, and an effect for each repair record (compared to the baseline of 5). *In this case*, the results will be saved in the memory (overwriting the original data), and not to a file:

```
. parmby
>   "regress mpg ibn.foreign ibn.foreign#ib(last).rep78, noconst",
>   omit empty format(estimate min* max* %8.2f p %-8.2g) norestore;

Command: regress mpg ibn.foreign ibn.foreign#ib(last).rep78, noconst
note: 1.foreign#1.rep78 identifies no observations in the sample
note: 1.foreign#2.rep78 identifies no observations in the sample
```

`parmby` calls `regress`, which begins to produce its output. Note that not all values of `rep78` are represented in non–US models.

Step 2: Run the regression, creating a `parmby` resultsset

We now use the `parmby` module of `parmest` to run a regression model, with an intercept for each car origin type, and an effect for each repair record (compared to the baseline of 5). *In this case*, the results will be saved in the memory (overwriting the original data), and not to a file:

```
. parmby
>   "regress mpg ibn.foreign ibn.foreign#ib(last).rep78, noconst",
>   omit empty format(estimate min* max* %8.2f p %-8.2g) norestore;
```

```
Command: regress mpg ibn.foreign ibn.foreign#ib(last).rep78, noconst
note: 1.foreign#1.rep78 identifies no observations in the sample
note: 1.foreign#2.rep78 identifies no observations in the sample
```

`parmby` calls `regress`, which begins to produce its output. Note that not all values of `rep78` are represented in non–US models.

Step 2: Run the regression, creating a parmby resultsset

We now use the `parmby` module of `parmest` to run a regression model, with an intercept for each car origin type, and an effect for each repair record (compared to the baseline of 5). *In this case*, the results will be saved in the memory (overwriting the original data), and not to a file:

```
. parmby
>   "regress mpg ibn.foreign ibn.foreign#ib(last).rep78, noconst",
>   omit empty format(estimate min* max* %8.2f p %-8.2g) norestore;

Command: regress mpg ibn.foreign ibn.foreign#ib(last).rep78, noconst
note: 1.foreign#1.rep78 identifies no observations in the sample
note: 1.foreign#2.rep78 identifies no observations in the sample
```

`parmby` calls `regress`, which begins to produce its output. Note that not all values of `rep78` are represented in non–US models.

Step 2: Run the regression, creating a `parmby` resultsset

We now use the `parmby` module of `parmest` to run a regression model, with an intercept for each car origin type, and an effect for each repair record (compared to the baseline of 5). *In this case*, the results will be saved in the memory (overwriting the original data), and not to a file:

```
. parmby
>   "regress mpg ibn.foreign ibn.foreign#ib(last).rep78, noconst",
>   omit empty format(estimate min* max* %8.2f p %-8.2g) norestore;

Command: regress mpg ibn.foreign ibn.foreign#ib(last).rep78, noconst
note: 1.foreign#1.rep78 identifies no observations in the sample
note: 1.foreign#2.rep78 identifies no observations in the sample
```

`parmby` calls `regress`, which begins to produce its output. Note that not all values of `rep78` are represented in non–US models.

Step 2 (continued): Regression results

The Stata Version 11 `regress` produces its usual output:

```
      Source |       SS       df       MS              Number of obs =      69
-------------+------------------------------           F(  8,    61) =  163.18
       Model |  32114.3472     8  4014.2934           Prob > F      =  0.0000
    Residual |  1500.65278    61  24.6008652           R-squared     =  0.9554
-------------+------------------------------           Adj R-squared =  0.9495
       Total |       33615    69  487.173913           Root MSE      =  4.9599

------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     foreign |
          0  |         32   3.507197     9.12   0.000     24.98693    39.01307
          1  |   26.33333   1.653309    15.93   0.000     23.02734    29.63933
             |
    foreign#|
       rep78 |
        0 1  |        -11   4.959926    -2.22   0.030    -20.91798   -1.082015
        0 2  |    -12.875   3.921166    -3.28   0.002    -20.71586   -5.034145
        0 3  |        -13   3.634773    -3.58   0.001    -20.26818   -5.731822
        0 4  |  -13.55556   3.877352    -3.50   0.001     -21.3088    -5.80231
        1 1  |  (empty)
        1 2  |  (empty)
        1 3  |         -3   3.306617    -0.91   0.368     -9.61199     3.61199
        1 4  |  -1.444444   2.338132    -0.62   0.539    -6.119827    3.230938
------------------------------------------------------------------------------
```

There are 2 intercepts for US and non–US models, and effects on mileage for repair records other than 5 (the reference level).

Step 2 (continued): The `parmby` resultsset

After `parmby`, we can list part of its resultsset:

```
. list parm omit empty estimate min* max* p, clean noobs;

                  parm   omit   empty   estimate    min95   max95   p
             0.foreign      0       0      32.00    24.99   39.01   5.3e-13
             1.foreign      0       0      26.33    23.03   29.64   2.1e-23
     0.foreign#1.rep78      0       0     -11.00   -20.92   -1.08   .03
     0.foreign#2.rep78      0       0     -12.88   -20.72   -5.03   .0017
     0.foreign#3.rep78      0       0     -13.00   -20.27   -5.73   .00069
     0.foreign#4.rep78      0       0     -13.56   -21.31   -5.80   .00089
   0o.foreign#5b.rep78      1       0       0.00     0.00    0.00   .
   1o.foreign#1o.rep78      1       1       0.00     0.00    0.00   .
   1o.foreign#2o.rep78      1       1       0.00     0.00    0.00   .
     1.foreign#3.rep78      0       0      -3.00    -9.61    3.61   .37
     1.foreign#4.rep78      0       0      -1.44    -6.12    3.23   .54
   1o.foreign#5b.rep78      1       0       0.00     0.00    0.00   .
```

The variable `parm` contains alien–looking parameter names, taken from their "virtual variables". The variables `omit` and `empty` indicate that the parameter is omitted, or corresponds to an empty group, respectively.

Step 3: Regenerate the factors using `fvregen`

To regenerate user–friendly factor variables from the alien–looking parameter names, we call `fvregen`, using the temporary do–file that we created earlier using `descsave`:

```
. fvregen, do(`"`df0'"');

Factor variables generated:
foreign rep78
```

Note that `fvregen` lists the names of the factor variables that it has regenerated.

Step 3: Regenerate the factors using **`fvregen`**

To regenerate user–friendly factor variables from the alien–looking parameter names, we call `fvregen`, using the temporary do–file that we created earlier using `descsave`:

```
. fvregen, do(`"`df0'"');

Factor variables generated:
foreign rep78
```

Note that `fvregen` lists the names of the factor variables that it has regenerated.

Step 3: Regenerate the factors using **`fvregen`**

To regenerate user–friendly factor variables from the alien–looking parameter names, we call `fvregen`, using the temporary do–file that we created earlier using `descsave`:

```
. fvregen, do(`"`df0'"');
```

```
Factor variables generated:
foreign rep78
```

Note that `fvregen` lists the names of the factor variables that it has regenerated.

Step 3: Regenerate the factors using **`fvregen`**

To regenerate user–friendly factor variables from the alien–looking parameter names, we call `fvregen`, using the temporary do–file that we created earlier using `descsave`:

```
. fvregen, do(`"`df0'"');

Factor variables generated:
foreign rep78
```

Note that `fvregen` lists the names of the factor variables that it has regenerated.

Step 3: Regenerate the factors using **fvregen**

To regenerate user–friendly factor variables from the alien–looking parameter names, we call `fvregen`, using the temporary do–file that we created earlier using `descsave`:

```
. fvregen, do(`"`df0'"');

Factor variables generated:
foreign rep78
```

Note that `fvregen` lists the names of the factor variables that it has regenerated.

Step 3 (continued): The regenerated resultsset

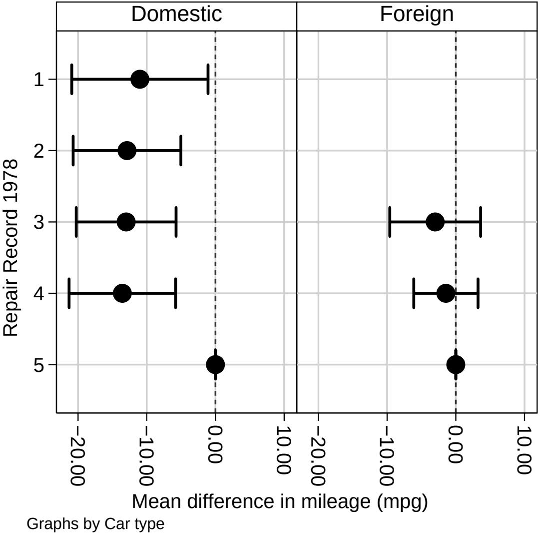
We can now list the resultsset, including the regenerated factors:

```
. list foreign rep78 omit empty estimate min* max* p,
>   noobs sepby(foreign);

  +------------------------------------------------------------------------+
  |  foreign   rep78   omit   empty   estimate    min95   max95   p        |
  |------------------------------------------------------------------------|
  | Domestic       .      0       0      32.00    24.99   39.01   5.3e-13 |
  |------------------------------------------------------------------------|
  |  Foreign       .      0       0      26.33    23.03   29.64   2.1e-23 |
  |------------------------------------------------------------------------|
  | Domestic       1      0       0     -11.00   -20.92   -1.08   .03      |
  | Domestic       2      0       0     -12.88   -20.72   -5.03   .0017    |
  | Domestic       3      0       0     -13.00   -20.27   -5.73   .00069   |
  | Domestic       4      0       0     -13.56   -21.31   -5.80   .00089   |
  | Domestic       5      1       0       0.00     0.00    0.00   .        |
  |------------------------------------------------------------------------|
  |  Foreign       1      1       1       0.00     0.00    0.00   .        |
  |  Foreign       2      1       1       0.00     0.00    0.00   .        |
  |  Foreign       3      0       0      -3.00    -9.61    3.61   .37      |
  |  Foreign       4      0       0      -1.44    -6.12    3.23   .54      |
  |  Foreign       5      1       0       0.00     0.00    0.00   .        |
  +------------------------------------------------------------------------+
```

In each origin group (US models and non–US models), there is an intercept, equal to the mean mileage for “reference” cars with `rep78==5`, and an effect (or mean difference) for each level of `rep78`, including zero effects for the reference and empty groups.

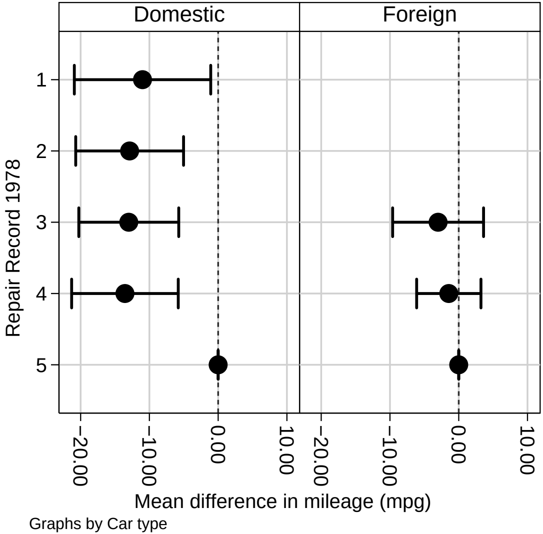
Step 4: Plot the confidence intervals using `eclplot`

- The plots correspond to the 2 origin groups (US and non–US models).
- The vertical axis is repair record (with a reference level of 5).
- The confidence intervals are for differences from the reference level in mean mileage, including reference groups but *not* empty groups.



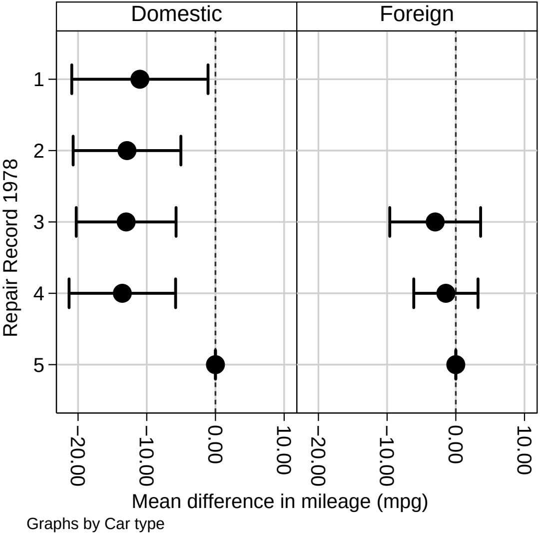
Step 4: Plot the confidence intervals using `eclplot`

- The plots correspond to the 2 origin groups (US and non–US models).
- The vertical axis is repair record (with a reference level of 5).
- The confidence intervals are for differences from the reference level in mean mileage, including reference groups but *not* empty groups.



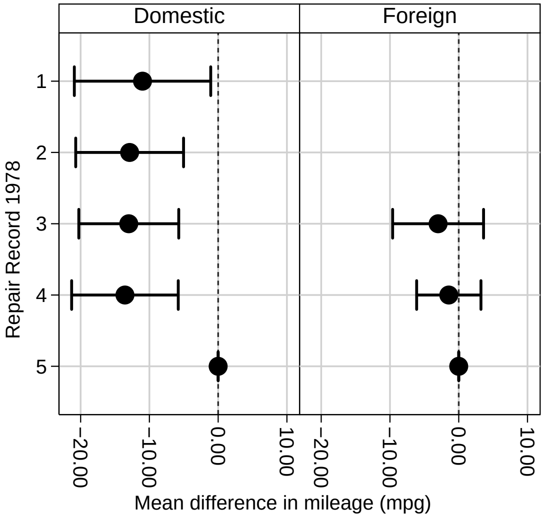
Step 4: Plot the confidence intervals using `eclplot`

- The plots correspond to the 2 origin groups (US and non–US models).
- The vertical axis is repair record (with a reference level of 5).
- The confidence intervals are for differences from the reference level in mean mileage, including reference groups but *not* empty groups.



Step 4: Plot the confidence intervals using `eclplot`

- The plots correspond to the 2 origin groups (US and non–US models).
- The vertical axis is repair record (with a reference level of 5).
- The confidence intervals are for differences from the reference level in mean mileage, including reference groups but *not* empty groups.



Graphs by Car type

The `invcise` package: Standard errors from confidence limits

- `invcise` inputs confidence limits and (optionally) degrees of freedom, and generates standard errors "backwards", by inverting the usual confidence interval formulas.
- It is used when confidence limits have been generated by an *unusual* formula, as with confidence limits for medians and Hodges–Lehmann median differences.
- The generated standard errors, with their estimates, can be input to the `metaparm` module of `parmest` (or to `metan`), to generate interval estimates of linear combinations of *independent* parameters, as proposed by Bonett and Price (2002)[1].
- Applications include means of differences ("meta–analysis") and differences between differences ("interactions").
- Note that a Hodges–Lehmann median difference is *not* a difference between medians, although `invcise` and `metaparm` can handle either of these.

The `invcise` package: Standard errors from confidence limits

- `invcise` inputs confidence limits and (optionally) degrees of freedom, and generates standard errors “backwards”, by inverting the usual confidence interval formulas.
- It is used when confidence limits have been generated by an *unusual* formula, as with confidence limits for medians and Hodges–Lehmann median differences.
- The generated standard errors, with their estimates, can be input to the `metaparm` module of `parmest` (or to `metan`), to generate interval estimates of linear combinations of *independent* parameters, as proposed by Bonett and Price (2002)[1].
- Applications include means of differences (“meta–analysis”) and differences between differences (“interactions”).
- Note that a Hodges–Lehmann median difference is *not* a difference between medians, although `invcise` and `metaparm` can handle either of these.

The `invcise` package: Standard errors from confidence limits

- `invcise` inputs confidence limits and (optionally) degrees of freedom, and generates standard errors “backwards”, by inverting the usual confidence interval formulas.
- It is used when confidence limits have been generated by an *unusual* formula, as with confidence limits for medians and Hodges–Lehmann median differences.
- The generated standard errors, with their estimates, can be input to the `metaparm` module of `parmest` (or to `metan`), to generate interval estimates of linear combinations of *independent* parameters, as proposed by Bonett and Price (2002)[1].
- Applications include means of differences (“meta–analysis”) and differences between differences (“interactions”).
- Note that a Hodges–Lehmann median difference is *not* a difference between medians, although `invcise` and `metaparm` can handle either of these.

The `invcise` package: Standard errors from confidence limits

- `invcise` inputs confidence limits and (optionally) degrees of freedom, and generates standard errors "backwards", by inverting the usual confidence interval formulas.
- It is used when confidence limits have been generated by an *unusual* formula, as with confidence limits for medians and Hodges–Lehmann median differences.
- The generated standard errors, with their estimates, can be input to the `metaparm` module of `parmest` (or to `metan`), to generate interval estimates of linear combinations of *independent* parameters, as proposed by Bonett and Price (2002)[1].
- Applications include means of differences ("meta–analysis") and differences between differences ("interactions").
- Note that a Hodges–Lehmann median difference is *not* a difference between medians, although `invcise` and `metaparm` can handle either of these.

The `invcise` package: Standard errors from confidence limits

- `invcise` inputs confidence limits and (optionally) degrees of freedom, and generates standard errors “backwards”, by inverting the usual confidence interval formulas.
- It is used when confidence limits have been generated by an *unusual* formula, as with confidence limits for medians and Hodges–Lehmann median differences.
- The generated standard errors, with their estimates, can be input to the `metaparm` module of `parmest` (or to `metan`), to generate interval estimates of linear combinations of *independent* parameters, as proposed by Bonett and Price (2002)[1].
- Applications include means of differences (“meta–analysis”) and differences between differences (“interactions”).
- Note that a Hodges–Lehmann median difference is *not* a difference between medians, although `invcise` and `metaparm` can handle either of these.

The `invcise` package: Standard errors from confidence limits

- `invcise` inputs confidence limits and (optionally) degrees of freedom, and generates standard errors "backwards", by inverting the usual confidence interval formulas.
- It is used when confidence limits have been generated by an *unusual* formula, as with confidence limits for medians and Hodges–Lehmann median differences.
- The generated standard errors, with their estimates, can be input to the `metaparm` module of `parmest` (or to `metan`), to generate interval estimates of linear combinations of *independent* parameters, as proposed by Bonett and Price (2002)[1].
- Applications include means of differences ("meta–analysis") and differences between differences ("interactions").
- Note that a Hodges–Lehmann median difference is *not* a difference between medians, although `invcise` and `metaparm` can handle either of these.

Example: Rank interactions in the `auto` data

- In the `auto` data, we might generate a new variable `odd=mod(_n,2)`, indicating that a model is odd–numbered (instead of even–numbered).
- We might then estimate 2 Hodges–Lehmann median differences in weight (pounds) between US and non–US cars, one for even car models and one for odd car models.
- We might then estimate a difference between these 2 median differences, to measure “interaction” between oddness and car model origin.
- This can be done using `invcise` to calculate standard errors for the 2 median differences, and then using `metaparm` to calculate a confidence interval for the difference between differences.
- Note that this example is similar to a previous one with *mean* weight differences in Newson (2008)[5].

Example: Rank interactions in the `auto` data

- In the `auto` data, we might generate a new variable `odd=mod(_n,2)`, indicating that a model is odd–numbered (instead of even–numbered).
- We might then estimate 2 Hodges–Lehmann median differences in weight (pounds) between US and non–US cars, one for even car models and one for odd car models.
- We might then estimate a difference between these 2 median differences, to measure "interaction" between oddness and car model origin.
- This can be done using `invcise` to calculate standard errors for the 2 median differences, and then using `metaparm` to calculate a confidence interval for the difference between differences.
- Note that this example is similar to a previous one with *mean* weight differences in Newson (2008)[5].

Example: Rank interactions in the `auto` data

- In the `auto` data, we might generate a new variable `odd=mod(_n,2)`, indicating that a model is odd–numbered (instead of even–numbered).
- We might then estimate 2 Hodges–Lehmann median differences in weight (pounds) between US and non–US cars, one for even car models and one for odd car models.
- We might then estimate a difference between these 2 median differences, to measure "interaction" between oddness and car model origin.
- This can be done using `invcise` to calculate standard errors for the 2 median differences, and then using `metaparm` to calculate a confidence interval for the difference between differences.
- Note that this example is similar to a previous one with *mean* weight differences in Newson (2008)[5].

Example: Rank interactions in the `auto` data

- In the `auto` data, we might generate a new variable `odd=mod(_n,2)`, indicating that a model is odd–numbered (instead of even–numbered).
- We might then estimate 2 Hodges–Lehmann median differences in weight (pounds) between US and non–US cars, one for even car models and one for odd car models.
- We might then estimate a difference between these 2 median differences, to measure “interaction” between oddness and car model origin.
- This can be done using `invcise` to calculate standard errors for the 2 median differences, and then using `metaparm` to calculate a confidence interval for the difference between differences.
- Note that this example is similar to a previous one with *mean* weight differences in Newson (2008)[5].

Example: Rank interactions in the `auto` data

- In the `auto` data, we might generate a new variable `odd=mod(_n,2)`, indicating that a model is odd–numbered (instead of even–numbered).
- We might then estimate 2 Hodges–Lehmann median differences in weight (pounds) between US and non–US cars, one for even car models and one for odd car models.
- We might then estimate a difference between these 2 median differences, to measure "interaction" between oddness and car model origin.
- This can be done using `invcise` to calculate standard errors for the 2 median differences, and then using `metaparm` to calculate a confidence interval for the difference between differences.
- Note that this example is similar to a previous one with *mean* weight differences in Newson (2008)[5].

Example: Rank interactions in the `auto` data

- In the `auto` data, we might generate a new variable `odd=mod(_n,2)`, indicating that a model is odd–numbered (instead of even–numbered).
- We might then estimate 2 Hodges–Lehmann median differences in weight (pounds) between US and non–US cars, one for even car models and one for odd car models.
- We might then estimate a difference between these 2 median differences, to measure "interaction" between oddness and car model origin.
- This can be done using `invcise` to calculate standard errors for the 2 median differences, and then using `metaparm` to calculate a confidence interval for the difference between differences.
- Note that this example is similar to a previous one with *mean* weight differences in Newson (2008)[5].

Step 1: Create a `statsby` resultsset

We begin by loading the `auto` data, and adding the `odd` variable as in Newson (2008)[5]. We then use `statsby` to create a resultsset with 1 observation per value of `odd`, and data on results from the `cendif` module of the `somersd` package (Newson, 2006b[4]), containing unequal–variance confidence intervals for Hodges–Lehmann median differences between US and non–US models:

```
. statsby N=(r(N))
>   estimate=(el(r(cimat),1,2)) dof=(r(df_r))
>   min95=(el(r(cimat),1,3)) max95=(el(r(cimat),1,4)),
>   by(odd) clear:
>   cendif weight, by(foreign) transf(iden) tdist;
```

Note that, although `statsby` is used, the variable names in the resultsset are "`parmest`–like".

Step 2: Add standard errors to the resultsset using `invcise`

The `statsby` resultsset has estimates and confidence limits (and degrees of freedom) for the median differences, but no standard errors. We use `invcise` to add these standard errors in a new variable `stderr`:

```
. list, abbr(32) clean noobs;

     odd    N   estimate   dof   min95   max95
    Even   37       1050    36     430    1350
     Odd   37       1225    36     690    1600

. invcise min95 max95 dof, stderr(stderr);
Confidence level assumed: 95%

. list odd N estimate stderr dof min95 max95, abbr(32) clean noobs;

     odd    N   estimate      stderr   dof   min95   max95
    Even   37       1050   226.81394    36     430    1350
     Odd   37       1225   224.34858    36     690    1600
```

Note that these standard errors are calculated using a *t*–distribution.

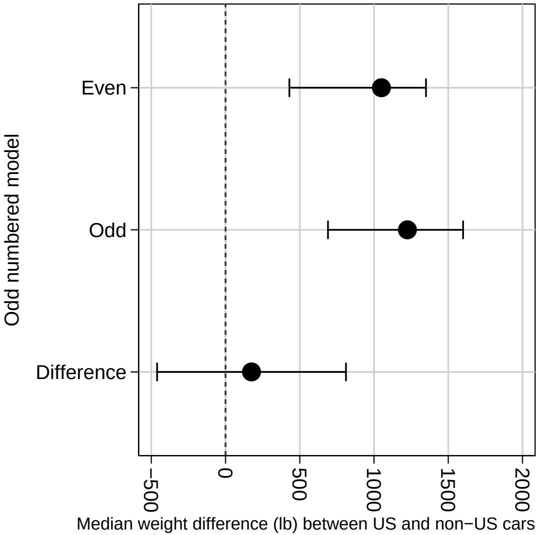
Step 3: Add difference between differences using **metaparm**

We then use the `metaparm` module of `parmest` to create a new resultsset, with 1 observation in a temporary file, and data on the odd–even difference between median differences. This is `append`ed to the main resultsset, and assigned a new value of the variable `odd`. The new resultsset is then listed:

```
. tempfile tf1;

. metaparm [iweight=(odd==1)-(odd==0)], sumvar(N) saving(`"`tf1'"', replace);
(note: file C:\DOCUME~1\rnewson\LOCALS~1\Temp\ST_000000bt.tmp not found)
file C:\DOCUME~1\rnewson\LOCALS~1\Temp\ST_000000bt.tmp saved

. append using `"`tf1'"', gene(interact);
dof was float now double
min95 was float now double
max95 was float now double

. lab def odd 2 "Difference", add;

. replace odd=2 if interact;
(1 real change made)

. list odd N estimate stderr dof min95 max95, abbr(32) clean noobs;

           odd    N   estimate      stderr       dof       min95       max95
          Even   37       1050   226.81394        36         430        1350
           Odd   37       1225   224.34858        36         690        1600
    Difference   74        175   319.02484   71.9914   -460.9657    810.9657
```

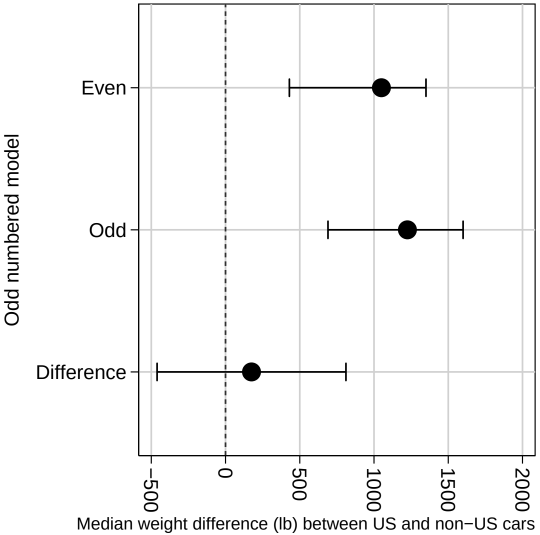
Step 4: Plot of median differences and their odd–even difference

- This plot was produced using `eclplot` in the final resultsset.
- Unsurprisingly, US cars are typically heavier than non–US cars, whether they are even– or odd–numbered.
- *However*, the population odd–even difference between the 2 Hodges–Lehmann median differences may be zero.



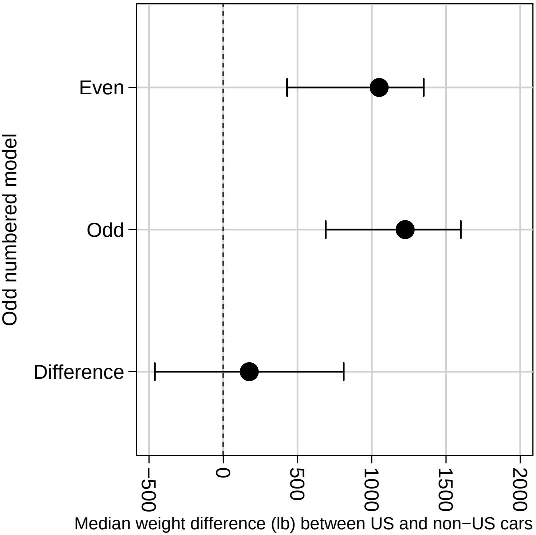
Step 4: Plot of median differences and their odd–even difference

- This plot was produced using `eclplot` in the final resultsset.
- Unsurprisingly, US cars are typically heavier than non–US cars, whether they are even– or odd–numbered.
- *However*, the population odd–even difference between the 2 Hodges–Lehmann median differences may be zero.



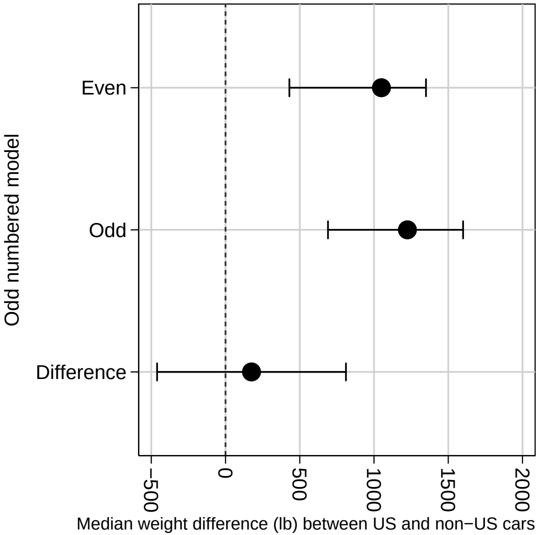
Step 4: Plot of median differences and their odd–even difference

- This plot was produced using `eclplot` in the final resultsset.
- Unsurprisingly, US cars are typically heavier than non–US cars, whether they are even– or odd–numbered.
- *However*, the population odd–even difference between the 2 Hodges–Lehmann median differences may be zero.



Step 4: Plot of median differences and their odd–even difference

- This plot was produced using `eclplot` in the final resultsset.
- Unsurprisingly, US cars are typically heavier than non–US cars, whether they are even– or odd–numbered.
- *However*, the population odd–even difference between the 2 Hodges–Lehmann median differences may be zero.



The `qqvalue` package: Frequentist q–values from P–values

- The `qqvalue` package is similar to the R package `p.adjust`.
- It inputs a variable containing multiple P–values, and outputs a variable containing the corresponding q–values (or "adjusted P–values"), defined by inverting a user–specified multiple–test procedure that aims to control the familywise error rate (FWER) or the false discovery rate (FDR).
- For each P–value, the frequentist q–value (or "quasi–q–value") is the lowest FWER or FDR for which that P–value would be in the discovery set, if the specified multiple–test procedure was used on the whole set of P–values.
- The range of multiple–test procedures available using `qqvalue` is a subset of 7 of the 11 available using the `multproc` module of the SSC package `smileplot` (Newson *et al.*, 2003)[2].

The `qqvalue` package: Frequentist q–values from P–values

- The `qqvalue` package is similar to the R package `p.adjust`.
- It inputs a variable containing multiple P–values, and outputs a variable containing the corresponding q–values (or "adjusted P–values"), defined by inverting a user–specified multiple–test procedure that aims to control the familywise error rate (FWER) or the false discovery rate (FDR).
- For each P–value, the frequentist q–value (or "quasi–q–value") is the lowest FWER or FDR for which that P–value would be in the discovery set, if the specified multiple–test procedure was used on the whole set of P–values.
- The range of multiple–test procedures available using `qqvalue` is a subset of 7 of the 11 available using the `multproc` module of the SSC package `smileplot` (Newson *et al.*, 2003)[2].

The **qqvalue** package: Frequentist q–values from P–values

- The `qqvalue` package is similar to the R package `p.adjust`.
- It inputs a variable containing multiple P–values, and outputs a variable containing the corresponding q–values (or "adjusted P–values"), defined by inverting a user–specified multiple–test procedure that aims to control the familywise error rate (FWER) or the false discovery rate (FDR).
- For each P–value, the frequentist q–value (or "quasi–q–value") is the lowest FWER or FDR for which that P–value would be in the discovery set, if the specified multiple–test procedure was used on the whole set of P–values.
- The range of multiple–test procedures available using `qqvalue` is a subset of 7 of the 11 available using the `multproc` module of the SSC package `smileplot` (Newson *et al.*, 2003)[2].

The `qqvalue` package: Frequentist *q*–values from *P*–values

- The `qqvalue` package is similar to the R package `p.adjust`.
- It inputs a variable containing multiple *P*–values, and outputs a variable containing the corresponding *q*–values (or "adjusted *P*–values"), defined by inverting a user–specified multiple–test procedure that aims to control the familywise error rate (FWER) or the false discovery rate (FDR).
- For each *P*–value, the frequentist *q*–value (or "quasi–*q*–value") is the lowest FWER or FDR for which that *P*–value would be in the discovery set, if the specified multiple–test procedure was used on the whole set of *P*–values.
- The range of multiple–test procedures available using `qqvalue` is a subset of 7 of the 11 available using the `multproc` module of the SSC package `smileplot` (Newson *et al.*, 2003)[2].

The `qqvalue` package: Frequentist *q*–values from *P*–values

- The `qqvalue` package is similar to the R package `p.adjust`.
- It inputs a variable containing multiple *P*–values, and outputs a variable containing the corresponding *q*–values (or "adjusted *P*–values"), defined by inverting a user–specified multiple–test procedure that aims to control the familywise error rate (FWER) or the false discovery rate (FDR).
- For each *P*–value, the frequentist *q*–value (or "quasi–*q*–value") is the lowest FWER or FDR for which that *P*–value would be in the discovery set, if the specified multiple–test procedure was used on the whole set of *P*–values.
- The range of multiple–test procedures available using `qqvalue` is a subset of 7 of the 11 available using the `multproc` module of the SSC package `smileplot` (Newson *et al.*, 2003)[2].

Multiple–test procedures: `qqvalue` *versus* `multproc`

- The `multproc` module of the `smileplot` package (Newson *et al.*, 2003)[2] inputs a variable, containing multiple P–values for multiple comparisons, and a single uncorrected critical P–value, assumed to represent the allowable familywise error rate (FWER) or false discovery rate (FDR).
- It outputs a single *corrected* critical P–value, and identifies the discovery set as the set of input P–values at or below this corrected critical P–value.
- The user can therefore call `multproc` multiple times, with descending uncorrected critical P–values (such as 0.25, 0.05, and 0.01), and identify a list of nested discovery sets, associated with ascending levels of confidence in their "significance".
- *However*, the q–values provided by `qqvalue` are (arguably) more informative, as they can be compared with *any* uncorrected critical P–value, and not just a short list of uncorrected critical P–values.

Multiple–test procedures: `qqvalue` *versus* `multproc`

- The `multproc` module of the `smileplot` package (Newson *et al.*, 2003)[2] inputs a variable, containing multiple *P*–values for multiple comparisons, and a single uncorrected critical *P*–value, assumed to represent the allowable familywise error rate (FWER) or false discovery rate (FDR).
- It outputs a single *corrected* critical *P*–value, and identifies the discovery set as the set of input *P*–values at or below this corrected critical *P*–value.
- The user can therefore call `multproc` multiple times, with descending uncorrected critical *P*–values (such as 0.25, 0.05, and 0.01), and identify a list of nested discovery sets, associated with ascending levels of confidence in their "significance".
- *However*, the *q*–values provided by `qqvalue` are (arguably) more informative, as they can be compared with *any* uncorrected critical *P*–value, and not just a short list of uncorrected critical *P*–values.

Multiple–test procedures: `qqvalue` *versus* `multproc`

- The `multproc` module of the `smileplot` package (Newson *et al.*, 2003)[2] inputs a variable, containing multiple *P*–values for multiple comparisons, and a single uncorrected critical *P*–value, assumed to represent the allowable familywise error rate (FWER) or false discovery rate (FDR).
- It outputs a single *corrected* critical *P*–value, and identifies the discovery set as the set of input *P*–values at or below this corrected critical *P*–value.
- The user can therefore call `multproc` multiple times, with descending uncorrected critical *P*–values (such as 0.25, 0.05, and 0.01), and identify a list of nested discovery sets, associated with ascending levels of confidence in their "significance".
- *However*, the *q*–values provided by `qqvalue` are (arguably) more informative, as they can be compared with *any* uncorrected critical *P*–value, and not just a short list of uncorrected critical *P*–values.

Multiple–test procedures: `qqvalue` *versus* `multproc`

- The `multproc` module of the `smileplot` package (Newson *et al.*, 2003)[2] inputs a variable, containing multiple *P*–values for multiple comparisons, and a single uncorrected critical *P*–value, assumed to represent the allowable familywise error rate (FWER) or false discovery rate (FDR).
- It outputs a single *corrected* critical *P*–value, and identifies the discovery set as the set of input *P*–values at or below this corrected critical *P*–value.
- The user can therefore call `multproc` multiple times, with descending uncorrected critical *P*–values (such as 0.25, 0.05, and 0.01), and identify a list of nested discovery sets, associated with ascending levels of confidence in their "significance".
- *However*, the *q*–values provided by `qqvalue` are (arguably) more informative, as they can be compared with *any* uncorrected critical *P*–value, and not just a short list of uncorrected critical *P*–values.

Multiple–test procedures: `qqvalue` *versus* `multproc`

- The `multproc` module of the `smileplot` package (Newson *et al.*, 2003)[2] inputs a variable, containing multiple *P*–values for multiple comparisons, and a single uncorrected critical *P*–value, assumed to represent the allowable familywise error rate (FWER) or false discovery rate (FDR).
- It outputs a single *corrected* critical *P*–value, and identifies the discovery set as the set of input *P*–values at or below this corrected critical *P*–value.
- The user can therefore call `multproc` multiple times, with descending uncorrected critical *P*–values (such as 0.25, 0.05, and 0.01), and identify a list of nested discovery sets, associated with ascending levels of confidence in their "significance".
- *However*, the *q*–values provided by `qqvalue` are (arguably) more informative, as they can be compared with *any* uncorrected critical *P*–value, and not just a short list of uncorrected critical *P*–values.

Example: Methylation assays in the ALSPAC birth cohort study

- In the ALSPAC birth cohort study in Bristol, there were 14060 subjects of known gender.
- In a nested pilot study, the cord blood DNA of 174 subjects (69 girls and 105 boys) was subjected to methylation assays, measuring DNA methylation (percent) at 1505 methylation sites in the human genome.
- A methylation site is a position in the genome where a single DNA base can either be methylated (typically implying that a gene is switched off), or unmethylated (typically implying that a gene is switched on).
- Each of the 1505 assays measured the percent of genomes in the blood sample in which a particular site was methylated.
- The methylation data were considered to be useful at 1495 of these sites.

Example: Methylation assays in the ALSPAC birth cohort study

- In the ALSPAC birth cohort study in Bristol, there were 14060 subjects of known gender.
- In a nested pilot study, the cord blood DNA of 174 subjects (69 girls and 105 boys) was subjected to methylation assays, measuring DNA methylation (percent) at 1505 methylation sites in the human genome.
- A methylation site is a position in the genome where a single DNA base can either be methylated (typically implying that a gene is switched off), or unmethylated (typically implying that a gene is switched on).
- Each of the 1505 assays measured the percent of genomes in the blood sample in which a particular site was methylated.
- The methylation data were considered to be useful at 1495 of these sites.

Example: Methylation assays in the ALSPAC birth cohort study

- In the ALSPAC birth cohort study in Bristol, there were 14060 subjects of known gender.
- In a nested pilot study, the cord blood DNA of 174 subjects (69 girls and 105 boys) was subjected to methylation assays, measuring DNA methylation (percent) at 1505 methylation sites in the human genome.
- A methylation site is a position in the genome where a single DNA base can either be methylated (typically implying that a gene is switched off), or unmethylated (typically implying that a gene is switched on).
- Each of the 1505 assays measured the percent of genomes in the blood sample in which a particular site was methylated.
- The methylation data were considered to be useful at 1495 of these sites.

Example: Methylation assays in the ALSPAC birth cohort study

- In the ALSPAC birth cohort study in Bristol, there were 14060 subjects of known gender.
- In a nested pilot study, the cord blood DNA of 174 subjects (69 girls and 105 boys) was subjected to methylation assays, measuring DNA methylation (percent) at 1505 methylation sites in the human genome.
- A methylation site is a position in the genome where a single DNA base can either be methylated (typically implying that a gene is switched off), or unmethylated (typically implying that a gene is switched on).
- Each of the 1505 assays measured the percent of genomes in the blood sample in which a particular site was methylated.
- The methylation data were considered to be useful at 1495 of these sites.

Example: Methylation assays in the ALSPAC birth cohort study

- In the ALSPAC birth cohort study in Bristol, there were 14060 subjects of known gender.
- In a nested pilot study, the cord blood DNA of 174 subjects (69 girls and 105 boys) was subjected to methylation assays, measuring DNA methylation (percent) at 1505 methylation sites in the human genome.
- A methylation site is a position in the genome where a single DNA base can either be methylated (typically implying that a gene is switched off), or unmethylated (typically implying that a gene is switched on).
- Each of the 1505 assays measured the percent of genomes in the blood sample in which a particular site was methylated.
- The methylation data were considered to be useful at 1495 of these sites.

Example: Methylation assays in the ALSPAC birth cohort study

- In the ALSPAC birth cohort study in Bristol, there were 14060 subjects of known gender.
- In a nested pilot study, the cord blood DNA of 174 subjects (69 girls and 105 boys) was subjected to methylation assays, measuring DNA methylation (percent) at 1505 methylation sites in the human genome.
- A methylation site is a position in the genome where a single DNA base can either be methylated (typically implying that a gene is switched off), or unmethylated (typically implying that a gene is switched on).
- Each of the 1505 assays measured the percent of genomes in the blood sample in which a particular site was methylated.
- The methylation data were considered to be useful at 1495 of these sites.

Comparing methylation levels in boys and girls

- As a preliminary analysis, we compared methylation levels between the 105 boys and the 69 girls.
- The methylation levels at the 1495 sites were non–Normally distributed, in ways that varied greatly from site to site (sometimes negatively skewed, sometimes positively skewed, sometimes bimodal, sometimes semi–discrete).
- This seemed to suggest that it would be a good idea to measure differences by estimating rank parameters, namely Somers' D (Newson, 2006a)[3] and the Hodges–Lehmann median difference (Newson, 2006b)[4], with robust confidence limits.
- Both of these parameters were restricted to comparisons within laboratory batches, to remove batch effects.
- We therefore calculated 1495 confidence intervals and P–values for Somers' D of site–specific percentage methylation with respect to male gender, and the corresponding q–values, using the Simes–Benjamini–Hochberg procedure.

Comparing methylation levels in boys and girls

- As a preliminary analysis, we compared methylation levels between the 105 boys and the 69 girls.
- The methylation levels at the 1495 sites were non–Normally distributed, in ways that varied greatly from site to site (sometimes negatively skewed, sometimes positively skewed, sometimes bimodal, sometimes semi–discrete).
- This seemed to suggest that it would be a good idea to measure differences by estimating rank parameters, namely Somers' *D* (Newson, 2006a)[3] and the Hodges–Lehmann median difference (Newson, 2006b)[4], with robust confidence limits.
- Both of these parameters were restricted to comparisons within laboratory batches, to remove batch effects.
- We therefore calculated 1495 confidence intervals and *P*–values for Somers' *D* of site–specific percentage methylation with respect to male gender, and the corresponding *q*–values, using the Simes–Benjamini–Hochberg procedure.

Comparing methylation levels in boys and girls

- As a preliminary analysis, we compared methylation levels between the 105 boys and the 69 girls.
- The methylation levels at the 1495 sites were non–Normally distributed, in ways that varied greatly from site to site (sometimes negatively skewed, sometimes positively skewed, sometimes bimodal, sometimes semi–discrete).
- This seemed to suggest that it would be a good idea to measure differences by estimating rank parameters, namely Somers' *D* (Newson, 2006a)[3] and the Hodges–Lehmann median difference (Newson, 2006b)[4], with robust confidence limits.
- Both of these parameters were restricted to comparisons within laboratory batches, to remove batch effects.
- We therefore calculated 1495 confidence intervals and *P*–values for Somers' *D* of site–specific percentage methylation with respect to male gender, and the corresponding *q*–values, using the Simes–Benjamini–Hochberg procedure.

Comparing methylation levels in boys and girls

- As a preliminary analysis, we compared methylation levels between the 105 boys and the 69 girls.
- The methylation levels at the 1495 sites were non–Normally distributed, in ways that varied greatly from site to site (sometimes negatively skewed, sometimes positively skewed, sometimes bimodal, sometimes semi–discrete).
- This seemed to suggest that it would be a good idea to measure differences by estimating rank parameters, namely Somers' *D* (Newson, 2006a)[3] and the Hodges–Lehmann median difference (Newson, 2006b)[4], with robust confidence limits.
- Both of these parameters were restricted to comparisons within laboratory batches, to remove batch effects.
- We therefore calculated 1495 confidence intervals and *P*–values for Somers' *D* of site–specific percentage methylation with respect to male gender, and the corresponding *q*–values, using the Simes–Benjamini–Hochberg procedure.

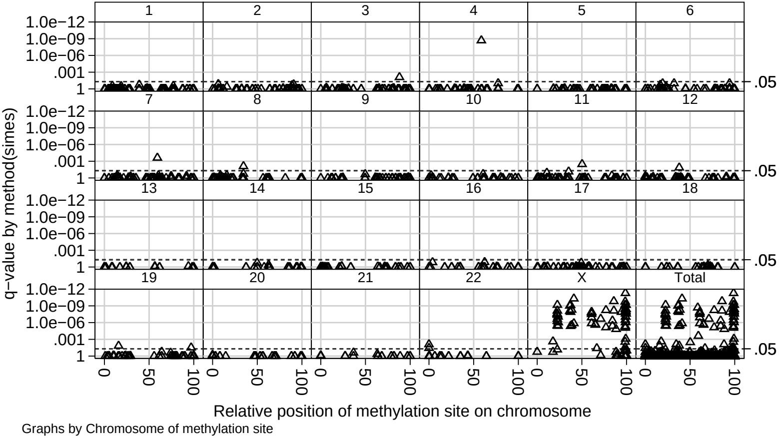
Comparing methylation levels in boys and girls

- As a preliminary analysis, we compared methylation levels between the 105 boys and the 69 girls.
- The methylation levels at the 1495 sites were non–Normally distributed, in ways that varied greatly from site to site (sometimes negatively skewed, sometimes positively skewed, sometimes bimodal, sometimes semi–discrete).
- This seemed to suggest that it would be a good idea to measure differences by estimating rank parameters, namely Somers' D (Newson, 2006a)[3] and the Hodges–Lehmann median difference (Newson, 2006b)[4], with robust confidence limits.
- Both of these parameters were restricted to comparisons within laboratory batches, to remove batch effects.
- We therefore calculated 1495 confidence intervals and P–values for Somers' D of site–specific percentage methylation with respect to male gender, and the corresponding q–values, using the Simes–Benjamini–Hochberg procedure.

Comparing methylation levels in boys and girls

- As a preliminary analysis, we compared methylation levels between the 105 boys and the 69 girls.
- The methylation levels at the 1495 sites were non–Normally distributed, in ways that varied greatly from site to site (sometimes negatively skewed, sometimes positively skewed, sometimes bimodal, sometimes semi–discrete).
- This seemed to suggest that it would be a good idea to measure differences by estimating rank parameters, namely Somers' *D* (Newson, 2006a)[3] and the Hodges–Lehmann median difference (Newson, 2006b)[4], with robust confidence limits.
- Both of these parameters were restricted to comparisons within laboratory batches, to remove batch effects.
- We therefore calculated 1495 confidence intervals and *P*–values for Somers' *D* of site–specific percentage methylation with respect to male gender, and the corresponding *q*–values, using the Simes–Benjamini–Hochberg procedure.

1495 q–values for Somers' D of methylation with respect to gender



In spite of the large number of sites, a lot of gender differences are still "significant", *mostly* for sites in the X–chromosome. (Girls have 2 X–chromosomes per cell, one of which is inactivated by methylation.)

References

[1] Bonett, D. G. and Price, R. M. 2002. Statistical inference for a linear function of medians: Confidence intervals, hypothesis testing, and sample size requirements. *Psychological Methods* **7(3)**: 370–383.

[2] Newson, R. 2003. Multiple–test procedures and smile plots. *The Stata Journal* **3(2)**: 109–132.

[3] Newson, R. 2006a. Confidence intervals for rank statistics: Somers' *D* and extensions. *The Stata Journal* **6(3)**: 309–334.

[4] Newson, R. 2006b. Confidence intervals for rank statistics: Percentile slopes, differences, and ratios. *The Stata Journal* **6(4)**: 497–520.

[5] Newson, R. B. 2008. `parmest` and extensions. Presented at the *14th UK Stata Users' Group Meeting*, 8–9 September, 2008. Downloadable from the conference website at *http://ideas.repec.org/s/boc/usug08.html*

This presentation can be downloaded from the conference website at *http://ideas.repec.org/s/boc/usug10.html*

The `parmest`, `descsave`, `fvregen`, `invcise`, `qqvalue`, `eclplot`, and `smileplot` packages, mentioned in this presentation, can be downloaded from SSC, using the `ssc` command.