Imperial College
London

Easy–to–use packages for estimating rank and spline parameters

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

Department of Primary Care and Public Health, Imperial College London

20th UK Stata Users' Group Meeting, 11–12 September, 2014
Downloadable from the conference website at
http://ideas.repec.org/s/boc/usug14.html

What are rank and spline parameters?

- So–called "non–parametric" methods are actually based on parameters.
- And these parameters can be sensible ones, which can be defined in words to non–mathematicians.
- **Rank parameters** (the "Kendall family") are defined in terms of ranks (or **ridits**).
- Sensible rank parameters include Kendall's τ_a, Somers' D, percentiles, and percentile slopes, differences and ratios, and are estimated using the package `somersd`[2][3].
- **Unrestricted spline parameters** (the "Schoenberg family") are defined in terms of **splines** (piecewise polynomials).
- Sensible spline parameters include the values of the spline at a list of reference points, or differences betwen these reference values, and are estimated using the package `bspline`[5].

What are rank and spline parameters?

- So–called "non–parametric" methods are actually based on parameters.
- And these parameters can be sensible ones, which can be defined in words to non–mathematicians.
- **Rank parameters** (the "Kendall family") are defined in terms of ranks (or **ridits**).
- Sensible rank parameters include Kendall's τ_a, Somers' D, percentiles, and percentile slopes, differences and ratios, and are estimated using the package `somersd`[2][3].
- **Unrestricted spline parameters** (the "Schoenberg family") are defined in terms of **splines** (piecewise polynomials).
- Sensible spline parameters include the values of the spline at a list of reference points, or differences betwen these reference values, and are estimated using the package `bspline`[5].

What are rank and spline parameters?

- So–called "non–parametric" methods are actually based on parameters.
- And these parameters can be sensible ones, which can be defined in words to non–mathematicians.
- **Rank parameters** (the "Kendall family") are defined in terms of ranks (or **ridits**).
- Sensible rank parameters include Kendall's τ_a, Somers' D, percentiles, and percentile slopes, differences and ratios, and are estimated using the package `somersd`[2][3].
- **Unrestricted spline parameters** (the "Schoenberg family") are defined in terms of **splines** (piecewise polynomials).
- Sensible spline parameters include the values of the spline at a list of reference points, or differences betwen these reference values, and are estimated using the package `bspline`[5].

What are rank and spline parameters?

- So–called "non–parametric" methods are actually based on parameters.
- And these parameters can be sensible ones, which can be defined in words to non–mathematicians.
- **Rank parameters** (the "Kendall family") are defined in terms of ranks (or **ridits**).
- Sensible rank parameters include Kendall's τ_a, Somers' D, percentiles, and percentile slopes, differences and ratios, and are estimated using the package `somersd`[2][3].
- **Unrestricted spline parameters** (the "Schoenberg family") are defined in terms of **splines** (piecewise polynomials).
- Sensible spline parameters include the values of the spline at a list of reference points, or differences betwen these reference values, and are estimated using the package `bspline`[5].

What are rank and spline parameters?

- So–called "non–parametric" methods are actually based on parameters.
- And these parameters can be sensible ones, which can be defined in words to non–mathematicians.
- **Rank parameters** (the "Kendall family") are defined in terms of ranks (or **ridits**).
- Sensible rank parameters include Kendall's τ_a, Somers' D, percentiles, and percentile slopes, differences and ratios, and are estimated using the package `somersd`[2][3].
- **Unrestricted spline parameters** (the "Schoenberg family") are defined in terms of **splines** (piecewise polynomials).
- Sensible spline parameters include the values of the spline at a list of reference points, or differences betwen these reference values, and are estimated using the package `bspline`[5].

What are rank and spline parameters?

- So–called "non–parametric" methods are actually based on parameters.
- And these parameters can be sensible ones, which can be defined in words to non–mathematicians.
- **Rank parameters** (the "Kendall family") are defined in terms of ranks (or **ridits**).
- Sensible rank parameters include Kendall's τ_a, Somers' D, percentiles, and percentile slopes, differences and ratios, and are estimated using the package `somersd`[2][3].
- **Unrestricted spline parameters** (the "Schoenberg family") are defined in terms of **splines** (piecewise polynomials).
- Sensible spline parameters include the values of the spline at a list of reference points, or differences betwen these reference values, and are estimated using the package `bspline`[5].

What are rank and spline parameters?

- So–called "non–parametric" methods are actually based on parameters.
- And these parameters can be sensible ones, which can be defined in words to non–mathematicians.
- **Rank parameters** (the "Kendall family") are defined in terms of ranks (or **ridits**).
- Sensible rank parameters include Kendall's τ_a, Somers' D, percentiles, and percentile slopes, differences and ratios, and are estimated using the package `somersd`[2][3].
- **Unrestricted spline parameters** (the "Schoenberg family") are defined in terms of **splines** (piecewise polynomials).
- Sensible spline parameters include the values of the spline at a list of reference points, or differences betwen these reference values, and are estimated using the package `bspline`[5].

Comprehensive solutions *versus* easy–to–use front–ends

- The packages `somersd` and `bspline` are "grand unified solutions" for estimating their whole respective families of parameters.
- Grand unified solutions have the advantage that the advanced user can learn (or even write) a single package, and can then estimate any parameter in the family.
- *For instance*, `somersd` can estimate Kendall's τ_a, percentile differences, and Theil–Sen percentile slopes, as well as the numerous aliases of Somers' D.
- *Similarly*, `bspline` can be used to estimate parameters for polynomials *and* splines with seasonal knots.
- *However*, most users, most of the time, want to do specific and basic tasks in a hurry.
- *So*, for these users, I have written `rcentile` as an easy–to–use front–end for `somersd`, and `polyspline` as an easy–to–use front–end for `bspline`.

Comprehensive solutions *versus* easy–to–use front–ends

- The packages `somersd` and `bspline` are "grand unified solutions" for estimating their whole respective families of parameters.
- Grand unified solutions have the advantage that the advanced user can learn (or even write) a single package, and can then estimate any parameter in the family.
- *For instance*, `somersd` can estimate Kendall's τ_a, percentile differences, and Theil–Sen percentile slopes, as well as the numerous aliases of Somers' D.
- *Similarly*, `bspline` can be used to estimate parameters for polynomials *and* splines with seasonal knots.
- *However*, most users, most of the time, want to do specific and basic tasks in a hurry.
- *So*, for these users, I have written `rcentile` as an easy–to–use front–end for `somersd`, and `polyspline` as an easy–to–use front–end for `bspline`.

Comprehensive solutions *versus* easy–to–use front–ends

- The packages `somersd` and `bspline` are "grand unified solutions" for estimating their whole respective families of parameters.
- Grand unified solutions have the advantage that the advanced user can learn (or even write) a single package, and can then estimate any parameter in the family.
- *For instance*, `somersd` can estimate Kendall's τ_a, percentile differences, and Theil–Sen percentile slopes, as well as the numerous aliases of Somers' D.
- *Similarly*, `bspline` can be used to estimate parameters for polynomials *and* splines with seasonal knots.
- *However*, most users, most of the time, want to do specific and basic tasks in a hurry.
- *So*, for these users, I have written `rcentile` as an easy–to–use front–end for `somersd`, and `polyspline` as an easy–to–use front–end for `bspline`.

Comprehensive solutions *versus* easy–to–use front–ends

- The packages `somersd` and `bspline` are "grand unified solutions" for estimating their whole respective families of parameters.
- Grand unified solutions have the advantage that the advanced user can learn (or even write) a single package, and can then estimate any parameter in the family.
- *For instance*, `somersd` can estimate Kendall's τ_a, percentile differences, and Theil–Sen percentile slopes, as well as the numerous aliases of Somers' D.
- *Similarly*, `bspline` can be used to estimate parameters for polynomials *and* splines with seasonal knots.
- *However*, most users, most of the time, want to do specific and basic tasks in a hurry.
- *So*, for these users, I have written `rcentile` as an easy–to–use front–end for `somersd`, and `polyspline` as an easy–to–use front–end for `bspline`.

Comprehensive solutions *versus* easy–to–use front–ends

- The packages `somersd` and `bspline` are "grand unified solutions" for estimating their whole respective families of parameters.
- Grand unified solutions have the advantage that the advanced user can learn (or even write) a single package, and can then estimate any parameter in the family.
- *For instance*, `somersd` can estimate Kendall's τ_a, percentile differences, and Theil–Sen percentile slopes, as well as the numerous aliases of Somers' D.
- *Similarly*, `bspline` can be used to estimate parameters for polynomials *and* splines with seasonal knots.
- *However*, most users, most of the time, want to do specific and basic tasks in a hurry.
- *So*, for these users, I have written `rcentile` as an easy–to–use front–end for `somersd`, and `polyspline` as an easy–to–use front–end for `bspline`.

Comprehensive solutions *versus* easy–to–use front–ends

- The packages `somersd` and `bspline` are "grand unified solutions" for estimating their whole respective families of parameters.
- Grand unified solutions have the advantage that the advanced user can learn (or even write) a single package, and can then estimate any parameter in the family.
- *For instance*, `somersd` can estimate Kendall's τ_a, percentile differences, and Theil–Sen percentile slopes, as well as the numerous aliases of Somers' D.
- *Similarly*, `bspline` can be used to estimate parameters for polynomials *and* splines with seasonal knots.
- *However*, most users, most of the time, want to do specific and basic tasks in a hurry.
- *So*, for these users, I have written `rcentile` as an easy–to–use front–end for `somersd`, and `polyspline` as an easy–to–use front–end for `bspline`.

Comprehensive solutions *versus* easy–to–use front–ends

- The packages `somersd` and `bspline` are "grand unified solutions" for estimating their whole respective families of parameters.
- Grand unified solutions have the advantage that the advanced user can learn (or even write) a single package, and can then estimate any parameter in the family.
- *For instance*, `somersd` can estimate Kendall's τ_a, percentile differences, and Theil–Sen percentile slopes, as well as the numerous aliases of Somers' D.
- *Similarly*, `bspline` can be used to estimate parameters for polynomials *and* splines with seasonal knots.
- *However*, most users, most of the time, want to do specific and basic tasks in a hurry.
- *So*, for these users, I have written `rcentile` as an easy–to–use front–end for `somersd`, and `polyspline` as an easy–to–use front–end for `bspline`.

`rcentile`: an easy–to–use front–end for `somersd`

- The package `rcentile` inputs a numeric variable and a list of percents, and outputs a matrix of confidence intervals for percentiles.
- These confidence intervals may be adjusted for clustered sampling and/or sampling–probability weights.
- They are calculated by inverting a confidence interval for a mean sign of pairwise differences, also known as a sign test statistic.
- This mean sign can be defined as a special case of Somers' D, closely related to the Gini inequality index.
- `rcentile` works by calling the module `sccendif` of the package `scsomersd`, which, in turn, works by calling the package `expgen`, and the module `cendif`[3] of the package `somersd`.

`rcentile`: an easy–to–use front–end for `somersd`

- The package `rcentile` inputs a numeric variable and a list of percents, and outputs a matrix of confidence intervals for percentiles.
- These confidence intervals may be adjusted for clustered sampling and/or sampling–probability weights.
- They are calculated by inverting a confidence interval for a mean sign of pairwise differences, also known as a sign test statistic.
- This mean sign can be defined as a special case of Somers' D, closely related to the Gini inequality index.
- `rcentile` works by calling the module `sccendif` of the package `scsomersd`, which, in turn, works by calling the package `expgen`, and the module `cendif`[3] of the package `somersd`.

`rcentile`: an easy–to–use front–end for `somersd`

- The package `rcentile` inputs a numeric variable and a list of percents, and outputs a matrix of confidence intervals for percentiles.
- These confidence intervals may be adjusted for clustered sampling and/or sampling–probability weights.
- They are calculated by inverting a confidence interval for a mean sign of pairwise differences, also known as a sign test statistic.
- This mean sign can be defined as a special case of Somers' D, closely related to the Gini inequality index.
- `rcentile` works by calling the module `sccendif` of the package `scsomersd`, which, in turn, works by calling the package `expgen`, and the module `cendif`[3] of the package `somersd`.

`rcentile`: an easy–to–use front–end for `somersd`

- The package `rcentile` inputs a numeric variable and a list of percents, and outputs a matrix of confidence intervals for percentiles.
- These confidence intervals may be adjusted for clustered sampling and/or sampling–probability weights.
- They are calculated by inverting a confidence interval for a mean sign of pairwise differences, also known as a sign test statistic.
- This mean sign can be defined as a special case of Somers' D, closely related to the Gini inequality index.
- `rcentile` works by calling the module `sccendif` of the package `scsomersd`, which, in turn, works by calling the package `expgen`, and the module `cendif`[3] of the package `somersd`.

`rcentile`: an easy–to–use front–end for `somersd`

- The package `rcentile` inputs a numeric variable and a list of percents, and outputs a matrix of confidence intervals for percentiles.
- These confidence intervals may be adjusted for clustered sampling and/or sampling–probability weights.
- They are calculated by inverting a confidence interval for a mean sign of pairwise differences, also known as a sign test statistic.
- This mean sign can be defined as a special case of Somers' D, closely related to the Gini inequality index.
- `rcentile` works by calling the module `sccendif` of the package `scsomersd`, which, in turn, works by calling the package `expgen`, and the module `cendif`[3] of the package `somersd`.

`rcentile`: an easy–to–use front–end for `somersd`

- The package `rcentile` inputs a numeric variable and a list of percents, and outputs a matrix of confidence intervals for percentiles.
- These confidence intervals may be adjusted for clustered sampling and/or sampling–probability weights.
- They are calculated by inverting a confidence interval for a mean sign of pairwise differences, also known as a sign test statistic.
- This mean sign can be defined as a special case of Somers' D, closely related to the Gini inequality index.
- `rcentile` works by calling the module `sccendif` of the package `scsomersd`, which, in turn, works by calling the package `expgen`, and the module `cendif`[3] of the package `somersd`.

`rcentile` *versus* `centile` *versus* `qreg`

- Unlike `centile`, `rcentile` can produce confidence intervals adjusted for clustered sampling and sampling–probability weights.
- And, unlike `qreg`, `rcentile` does not produce symmetric confidence intervals, computed using a standard error for a percentile.
- Instead, `rcentile` produces asymmetric confidence intervals, using a standard error for a mean sign, or for the Fisher z–transformed or arcsine–transformed mean sign.
- The Central Limit Theorem usually works faster for mean signs (or other versions of Somers' D) than for percentiles[1].
- And, in the case of non–median percentiles (corresponding to non–null mean signs), the Central Limit Theorem might work even faster for z–transformed or arcsine–transformed mean signs.

`rcentile` *versus* `centile` *versus* `qreg`

- Unlike `centile`, `rcentile` can produce confidence intervals adjusted for clustered sampling and sampling–probability weights.
- And, unlike `qreg`, `rcentile` does not produce symmetric confidence intervals, computed using a standard error for a percentile.
- Instead, `rcentile` produces asymmetric confidence intervals, using a standard error for a mean sign, or for the Fisher z–transformed or arcsine–transformed mean sign.
- The Central Limit Theorem usually works faster for mean signs (or other versions of Somers' D) than for percentiles[1].
- And, in the case of non–median percentiles (corresponding to non–null mean signs), the Central Limit Theorem might work even faster for z–transformed or arcsine–transformed mean signs.

`rcentile` *versus* `centile` *versus* `qreg`

- Unlike `centile`, `rcentile` can produce confidence intervals adjusted for clustered sampling and sampling–probability weights.
- And, unlike `qreg`, `rcentile` does not produce symmetric confidence intervals, computed using a standard error for a percentile.
- Instead, `rcentile` produces asymmetric confidence intervals, using a standard error for a mean sign, or for the Fisher z–transformed or arcsine–transformed mean sign.
- The Central Limit Theorem usually works faster for mean signs (or other versions of Somers' D) than for percentiles[1].
- And, in the case of non–median percentiles (corresponding to non–null mean signs), the Central Limit Theorem might work even faster for z–transformed or arcsine–transformed mean signs.

`rcentile` *versus* `centile` *versus* `qreg`

- Unlike `centile`, `rcentile` can produce confidence intervals adjusted for clustered sampling and sampling–probability weights.
- And, unlike `qreg`, `rcentile` does not produce symmetric confidence intervals, computed using a standard error for a percentile.
- Instead, `rcentile` produces asymmetric confidence intervals, using a standard error for a mean sign, or for the Fisher z–transformed or arcsine–transformed mean sign.
- The Central Limit Theorem usually works faster for mean signs (or other versions of Somers' D) than for percentiles[1].
- And, in the case of non–median percentiles (corresponding to non–null mean signs), the Central Limit Theorem might work even faster for z–transformed or arcsine–transformed mean signs.

`rcentile` *versus* `centile` *versus* `qreg`

- Unlike `centile`, `rcentile` can produce confidence intervals adjusted for clustered sampling and sampling–probability weights.
- And, unlike `qreg`, `rcentile` does not produce symmetric confidence intervals, computed using a standard error for a percentile.
- Instead, `rcentile` produces asymmetric confidence intervals, using a standard error for a mean sign, or for the Fisher z–transformed or arcsine–transformed mean sign.
- The Central Limit Theorem usually works faster for mean signs (or other versions of Somers' D) than for percentiles[1].
- And, in the case of non–median percentiles (corresponding to non–null mean signs), the Central Limit Theorem might work even faster for z–transformed or arcsine–transformed mean signs.

`rcentile` *versus* `centile` *versus* `qreg`

- Unlike `centile`, `rcentile` can produce confidence intervals adjusted for clustered sampling and sampling–probability weights.
- And, unlike `qreg`, `rcentile` does not produce symmetric confidence intervals, computed using a standard error for a percentile.
- Instead, `rcentile` produces asymmetric confidence intervals, using a standard error for a mean sign, or for the Fisher z–transformed or arcsine–transformed mean sign.
- The Central Limit Theorem usually works faster for mean signs (or other versions of Somers' D) than for percentiles[1].
- And, in the case of non–median percentiles (corresponding to non–null mean signs), the Central Limit Theorem might work even faster for z–transformed or arcsine–transformed mean signs.

Example: Car weights in the `auto` data

In this example, we will estimate percentiles of car weights, assuming that we are sampling car firms from a population of car firms, instead of sampling car models from a population of car models:

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

. gene firm=word(make,1);

. lab var firm "Firm";

. describe firm;

              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
firm            str7    %9s                   Firm
```

The generated string variable `firm` specifies the clusters of car models that we have sampled.

Percentiles of car weights output by `rcentile`

We then use `rcentile` to report the percentiles, from percentile 12.5 to percentile 87.5, with 12.5% increments, assuming that our sample is the 23 car firms represented:

```
. rcentile weight, centile(12.5(12.5)87.5) cluster(firm) tdist;
Percentile(s) for variable: weight
Mean sign transformation: Fisher's z
Valid observations: 74
Number of clusters (firm) = 23
Degrees of freedom: 22
95% confidence interval(s) for percentile(s)
  Percent    Centile    Minimum    Maximum
     12.5       2050       1830       2280
       25       2240       2070       2750
     37.5       2670       2200       3260
       50       3190       2640       3400
     62.5       3350       2930       3670
       75       3600       3310       3880
     87.5       3900       3600       4130
```

Note that the confidence intervals are calculated using the default Fisher's z–transform for the mean sign. This is probably a good transformation for non–median percentiles, although the identity might be better for medians.

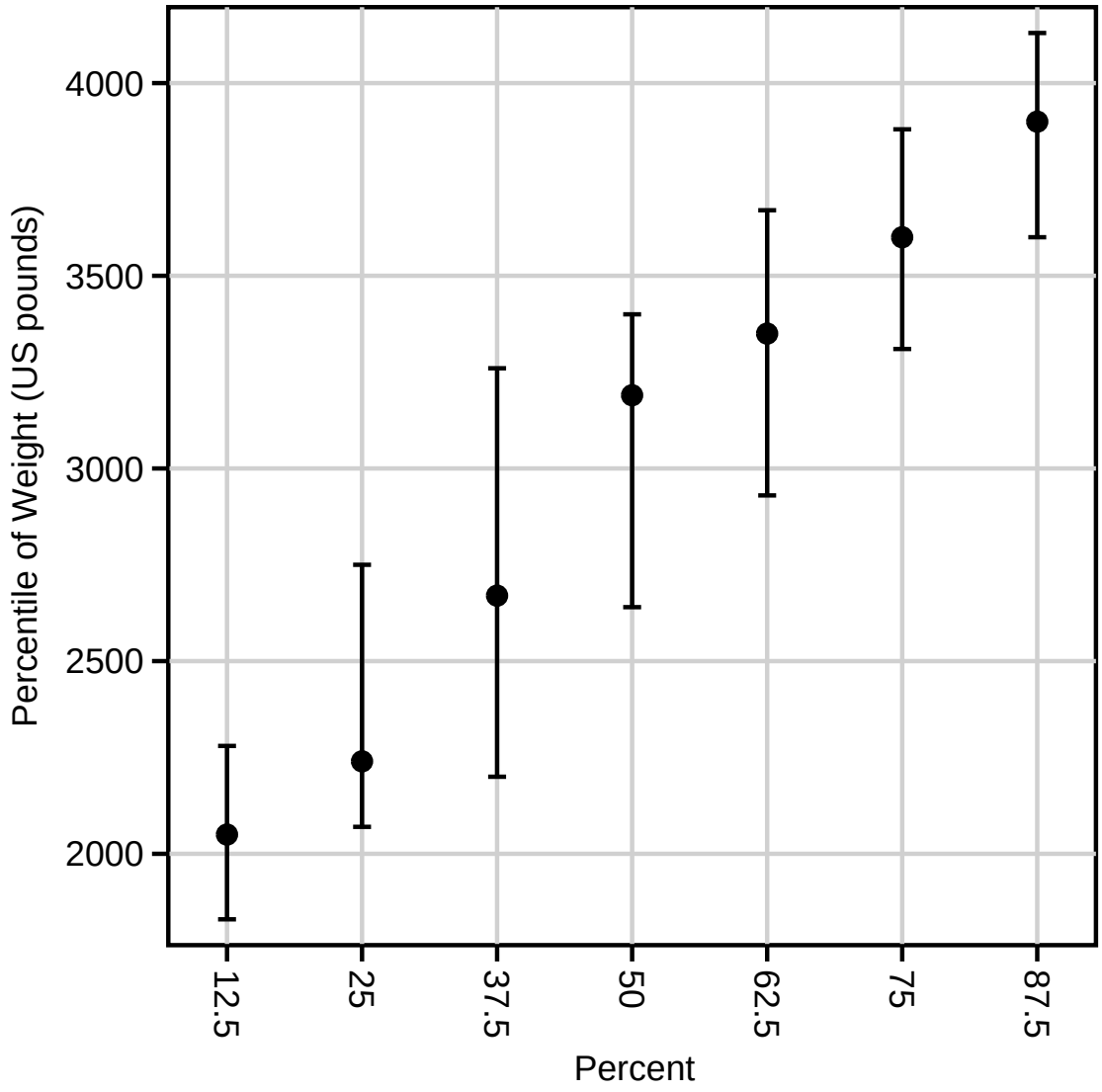
Saving and plotting the confidence limits

The confidence intervals are also saved in the matrix `r(cimat)`, which can be saved to a resultsset in memory using the `xsvmat` package, and plotted using `eclplot`:

```
. xsvmat, from(r(cimat)) name(col) norestore;

. describe;

Contains data
  obs:             7
 vars:             4
 size:           112
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
Percent         float   %9.0g
Centile         float   %9.0g
Minimum         float   %9.0g
Maximum         float   %9.0g
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved

. eclplot Centile Minimum Maximum Percent,
>   xlab(12.5(12.5)87.5)
>   ytitle("Percentile of Weight (US pounds)")
>   plotregion(margin(l=5 r=5))
>   xsize(5) ysize(5);
```

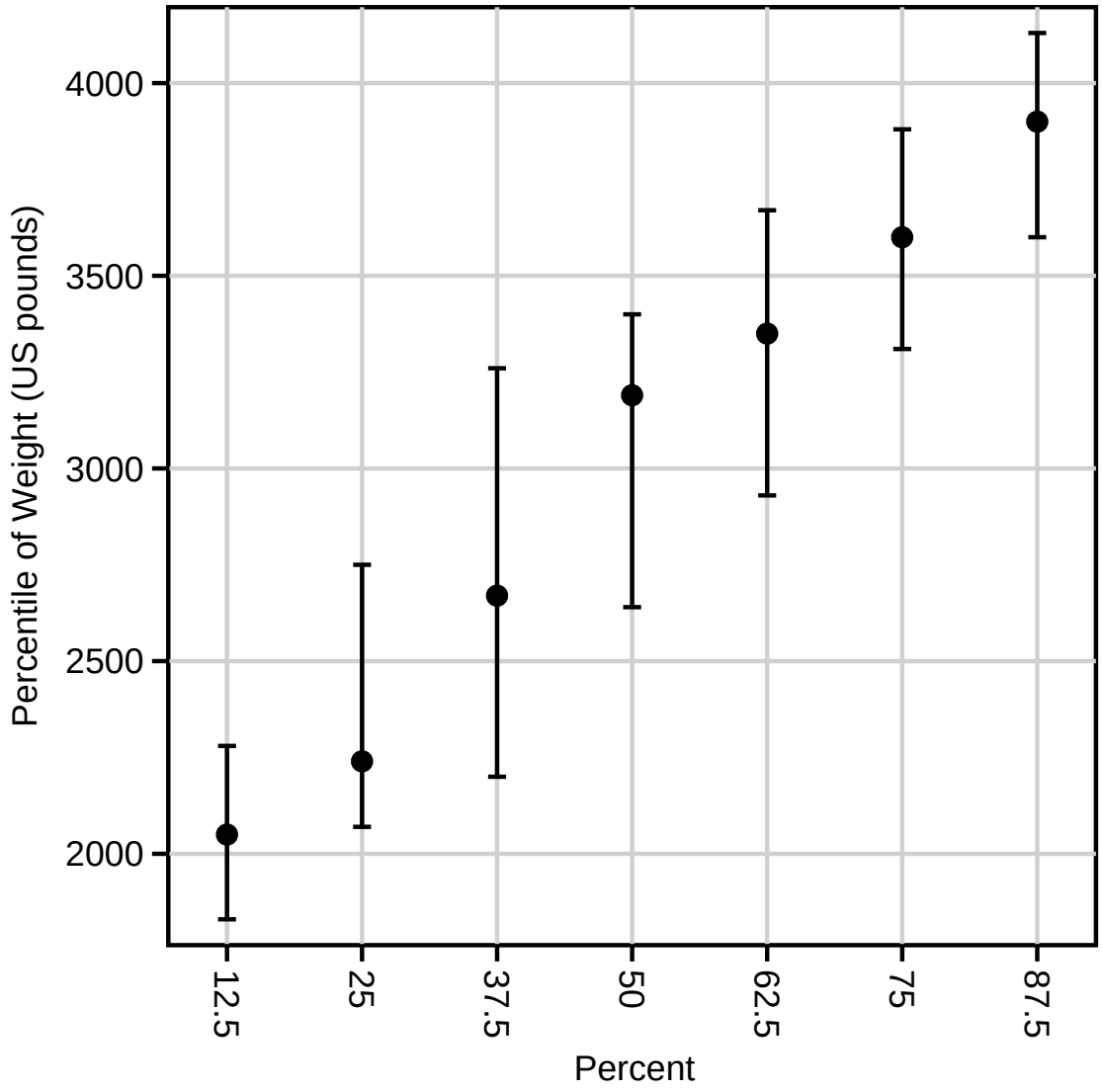
Robust confidence intervals for percentiles of car weight

- The percentiles are now plotted against the corresponding percentages, using `eclplot`.
- The confidence intervals are wider than the ones from `centile`, because of clustering.
- They are also asymmetric, unlike the ones from `qreg`.



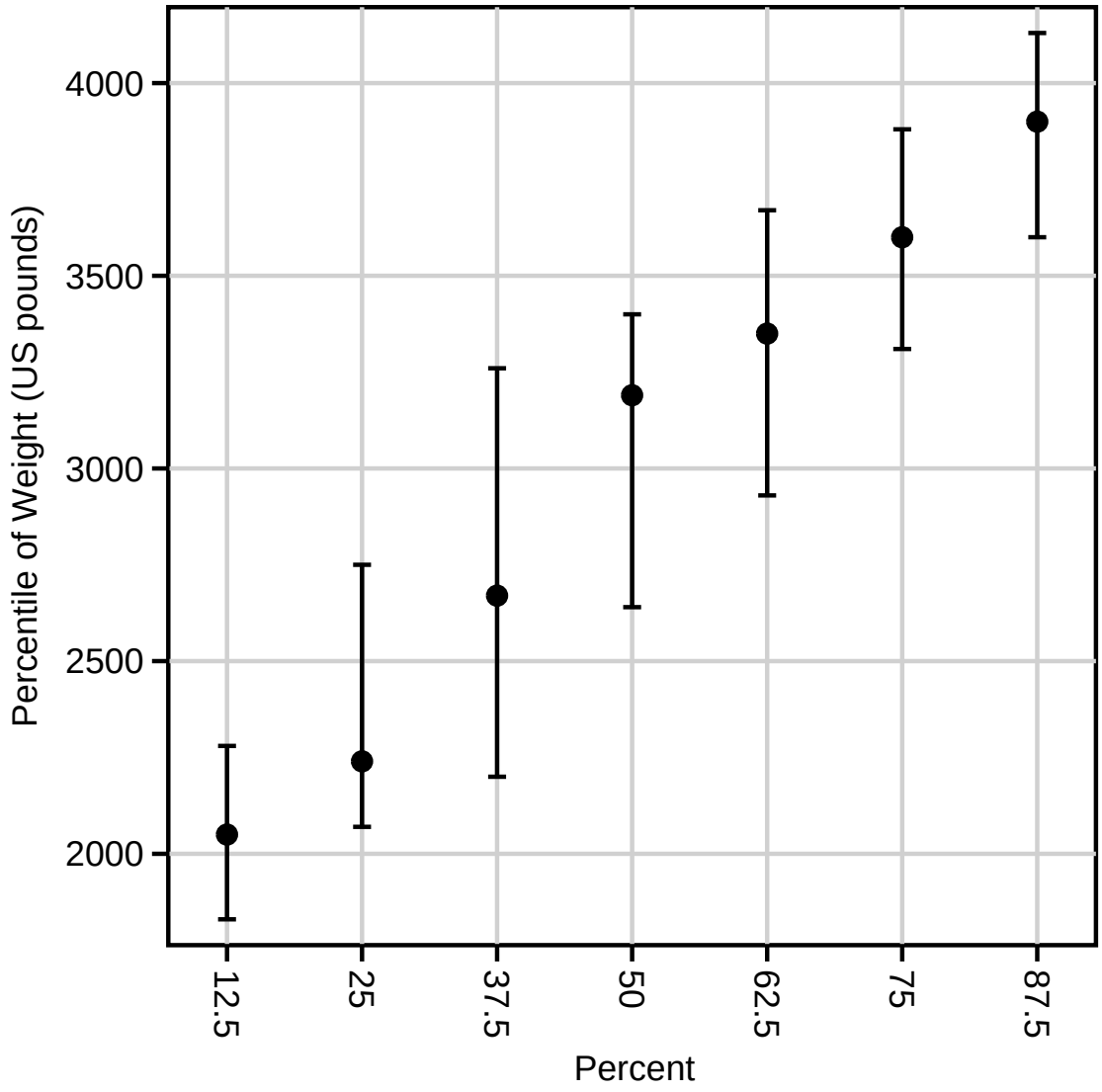
Robust confidence intervals for percentiles of car weight

- The percentiles are now plotted against the corresponding percentages, using `eclplot`.
- The confidence intervals are wider than the ones from `centile`, because of clustering.
- They are also asymmetric, unlike the ones from `qreg`.



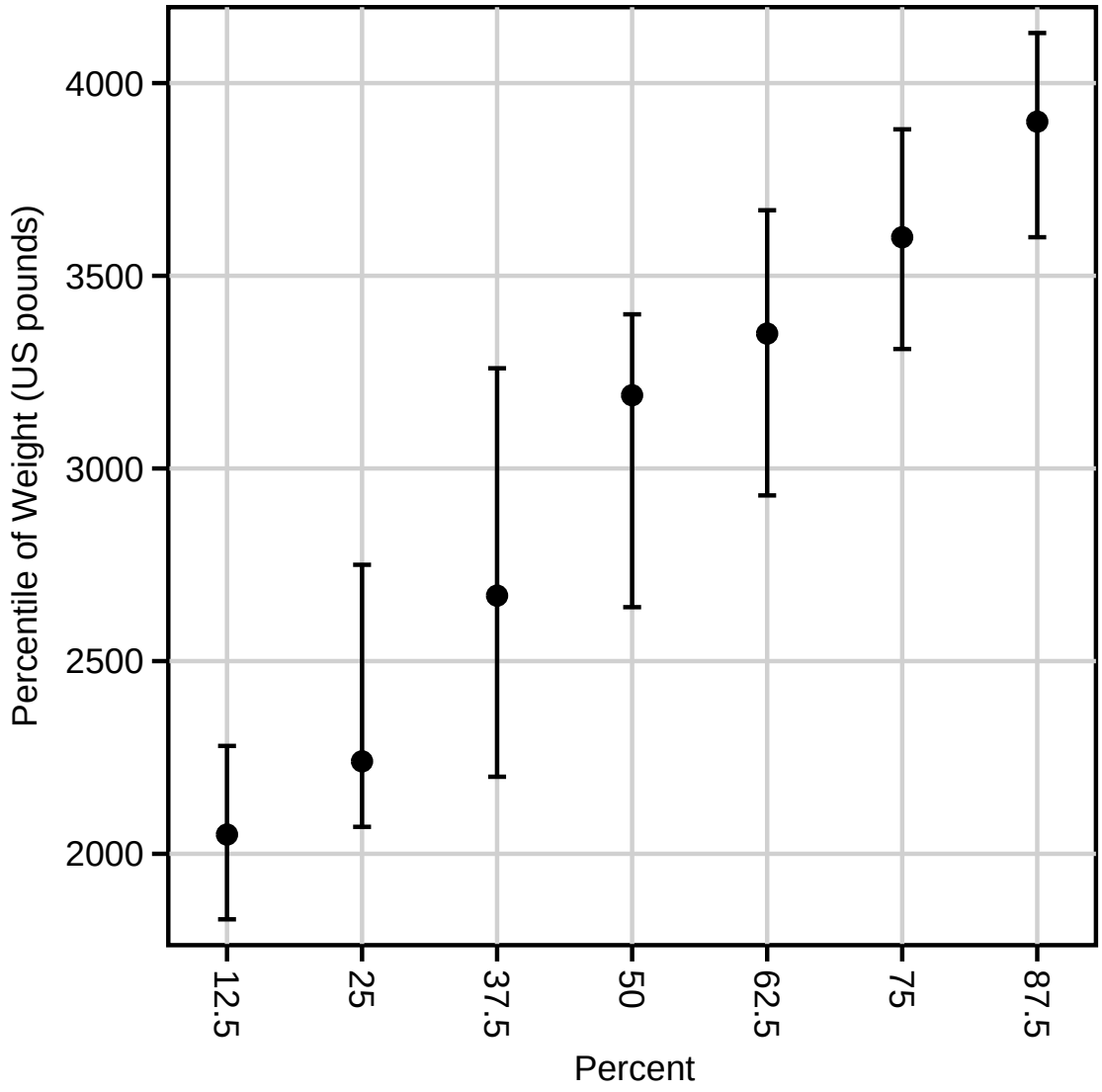
Robust confidence intervals for percentiles of car weight

- The percentiles are now plotted against the corresponding percentages, using `eclplot`.
- The confidence intervals are wider than the ones from `centile`, because of clustering.
- They are also asymmetric, unlike the ones from `qreg`.



Robust confidence intervals for percentiles of car weight

- The percentiles are now plotted against the corresponding percentages, using `eclplot`.
- The confidence intervals are wider than the ones from `centile`, because of clustering.
- They are also asymmetric, unlike the ones from `qreg`.



`polyspline`: an easy–to–use front–end for `bspline`

- The package `polyspline` inputs a numeric X–variable and a list of reference points on the X–axis, and outputs a list of generated **reference splines**, one per reference point.
- These reference splines can be included in the list of covariates for an estimation command.
- The corresponding parameters will then be polynomial (or other spline) levels (or effects).
- The levels will be values of the polynomial or spline at the reference points, and the effects will be differences between those values and the value at a base reference point.
- The reference splines therefore work for continuous factors as identifier (or dummy) variables work for discrete factors.
- `polyspline` works by calling the module `flexcurv`[4] of the package `bspline`, with sensible default options.

`polyspline`: an easy–to–use front–end for `bspline`

- The package `polyspline` inputs a numeric X–variable and a list of reference points on the X–axis, and outputs a list of generated **reference splines**, one per reference point.
- These reference splines can be included in the list of covariates for an estimation command.
- The corresponding parameters will then be polynomial (or other spline) levels (or effects).
- The levels will be values of the polynomial or spline at the reference points, and the effects will be differences between those values and the value at a base reference point.
- The reference splines therefore work for continuous factors as identifier (or dummy) variables work for discrete factors.
- `polyspline` works by calling the module `flexcurv`[4] of the package `bspline`, with sensible default options.

`polyspline`: an easy–to–use front–end for `bspline`

- The package `polyspline` inputs a numeric X–variable and a list of reference points on the X–axis, and outputs a list of generated **reference splines**, one per reference point.
- These reference splines can be included in the list of covariates for an estimation command.
- The corresponding parameters will then be polynomial (or other spline) levels (or effects).
- The levels will be values of the polynomial or spline at the reference points, and the effects will be differences between those values and the value at a base reference point.
- The reference splines therefore work for continuous factors as identifier (or dummy) variables work for discrete factors.
- `polyspline` works by calling the module `flexcurv`[4] of the package `bspline`, with sensible default options.

`polyspline`: an easy–to–use front–end for `bspline`

- The package `polyspline` inputs a numeric X–variable and a list of reference points on the X–axis, and outputs a list of generated **reference splines**, one per reference point.
- These reference splines can be included in the list of covariates for an estimation command.
- The corresponding parameters will then be polynomial (or other spline) levels (or effects).
- The levels will be values of the polynomial or spline at the reference points, and the effects will be differences between those values and the value at a base reference point.
- The reference splines therefore work for continuous factors as identifier (or dummy) variables work for discrete factors.
- `polyspline` works by calling the module `flexcurv`[4] of the package `bspline`, with sensible default options.

`polyspline`: an easy–to–use front–end for `bspline`

- The package `polyspline` inputs a numeric X–variable and a list of reference points on the X–axis, and outputs a list of generated **reference splines**, one per reference point.
- These reference splines can be included in the list of covariates for an estimation command.
- The corresponding parameters will then be polynomial (or other spline) levels (or effects).
- The levels will be values of the polynomial or spline at the reference points, and the effects will be differences between those values and the value at a base reference point.
- The reference splines therefore work for continuous factors as identifier (or dummy) variables work for discrete factors.
- `polyspline` works by calling the module `flexcurv`[4] of the package `bspline`, with sensible default options.

`polyspline`: an easy–to–use front–end for `bspline`

- The package `polyspline` inputs a numeric X–variable and a list of reference points on the X–axis, and outputs a list of generated **reference splines**, one per reference point.
- These reference splines can be included in the list of covariates for an estimation command.
- The corresponding parameters will then be polynomial (or other spline) levels (or effects).
- The levels will be values of the polynomial or spline at the reference points, and the effects will be differences between those values and the value at a base reference point.
- The reference splines therefore work for continuous factors as identifier (or dummy) variables work for discrete factors.
- `polyspline` works by calling the module `flexcurv`[4] of the package `bspline`, with sensible default options.

`polyspline`: an easy–to–use front–end for `bspline`

- The package `polyspline` inputs a numeric X–variable and a list of reference points on the X–axis, and outputs a list of generated **reference splines**, one per reference point.
- These reference splines can be included in the list of covariates for an estimation command.
- The corresponding parameters will then be polynomial (or other spline) levels (or effects).
- The levels will be values of the polynomial or spline at the reference points, and the effects will be differences between those values and the value at a base reference point.
- The reference splines therefore work for continuous factors as identifier (or dummy) variables work for discrete factors.
- `polyspline` works by calling the module `flexcurv`[4] of the package `bspline`, with sensible default options.

So what does `polyspline` add (or subtract)?

- `polyspline` only computes positive–degree reference splines (e.g. linear, quadratic, cubic, quartic or quintic).
- So it does not compute zero–degree splines (also known as right–continuous step functions), which are not even continuous, and which require increased documentation, causing confusion for casual users.
- In default, `polyspline` computes reference splines for a polynomial, with degree one less than the number of reference points.
- A polynomial is just a spline with no internal knots.
- *However*, if the user specifies a lower–degree spline (using the `power()` option), then the required internal knots are automatically interpolated between the reference points.
- This is done in such a way that the reference splines can still be computed, even if the reference points are irregularly spaced.

So what does `polyspline` add (or subtract)?

- `polyspline` only computes positive–degree reference splines (e.g. linear, quadratic, cubic, quartic or quintic).
- So it does not compute zero–degree splines (also known as right–continuous step functions), which are not even continuous, and which require increased documentation, causing confusion for casual users.
- In default, `polyspline` computes reference splines for a polynomial, with degree one less than the number of reference points.
- A polynomial is just a spline with no internal knots.
- *However*, if the user specifies a lower–degree spline (using the `power()` option), then the required internal knots are automatically interpolated between the reference points.
- This is done in such a way that the reference splines can still be computed, even if the reference points are irregularly spaced.

So what does `polyspline` add (or subtract)?

- `polyspline` only computes positive–degree reference splines (e.g. linear, quadratic, cubic, quartic or quintic).
- So it does not compute zero–degree splines (also known as right–continuous step functions), which are not even continuous, and which require increased documentation, causing confusion for casual users.
- In default, `polyspline` computes reference splines for a polynomial, with degree one less than the number of reference points.
- A polynomial is just a spline with no internal knots.
- *However*, if the user specifies a lower–degree spline (using the `power()` option), then the required internal knots are automatically interpolated between the reference points.
- This is done in such a way that the reference splines can still be computed, even if the reference points are irregularly spaced.

So what does `polyspline` add (or subtract)?

- `polyspline` only computes positive–degree reference splines (e.g. linear, quadratic, cubic, quartic or quintic).
- So it does not compute zero–degree splines (also known as right–continuous step functions), which are not even continuous, and which require increased documentation, causing confusion for casual users.
- In default, `polyspline` computes reference splines for a polynomial, with degree one less than the number of reference points.
- A polynomial is just a spline with no internal knots.
- *However*, if the user specifies a lower–degree spline (using the `power()` option), then the required internal knots are automatically interpolated between the reference points.
- This is done in such a way that the reference splines can still be computed, even if the reference points are irregularly spaced.

So what does `polyspline` add (or subtract)?

- `polyspline` only computes positive–degree reference splines (e.g. linear, quadratic, cubic, quartic or quintic).
- So it does not compute zero–degree splines (also known as right–continuous step functions), which are not even continuous, and which require increased documentation, causing confusion for casual users.
- In default, `polyspline` computes reference splines for a polynomial, with degree one less than the number of reference points.
- A polynomial is just a spline with no internal knots.
- *However*, if the user specifies a lower–degree spline (using the `power()` option), then the required internal knots are automatically interpolated between the reference points.
- This is done in such a way that the reference splines can still be computed, even if the reference points are irregularly spaced.

So what does `polyspline` add (or subtract)?

- `polyspline` only computes positive–degree reference splines (e.g. linear, quadratic, cubic, quartic or quintic).
- So it does not compute zero–degree splines (also known as right–continuous step functions), which are not even continuous, and which require increased documentation, causing confusion for casual users.
- In default, `polyspline` computes reference splines for a polynomial, with degree one less than the number of reference points.
- A polynomial is just a spline with no internal knots.
- *However*, if the user specifies a lower–degree spline (using the `power()` option), then the required internal knots are automatically interpolated between the reference points.
- This is done in such a way that the reference splines can still be computed, even if the reference points are irregularly spaced.

So what does `polyspline` add (or subtract)?

- `polyspline` only computes positive–degree reference splines (e.g. linear, quadratic, cubic, quartic or quintic).
- So it does not compute zero–degree splines (also known as right–continuous step functions), which are not even continuous, and which require increased documentation, causing confusion for casual users.
- In default, `polyspline` computes reference splines for a polynomial, with degree one less than the number of reference points.
- A polynomial is just a spline with no internal knots.
- *However*, if the user specifies a lower–degree spline (using the `power()` option), then the required internal knots are automatically interpolated between the reference points.
- This is done in such a way that the reference splines can still be computed, even if the reference points are irregularly spaced.

Example: Mileage and car weights in the `auto` data (again)

In the `auto` data, we use `polyspline` to generate a basis for a quadratic polynomial in `weight`:

```
. polyspline weight, refpts(2000 3000 4500) gene(qs_);
3 reference splines generated of degree: 2

. desc qs_*;

              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
qs_1            float   %8.4f                 Spline at 2,000
qs_2            float   %8.4f                 Spline at 3,000
qs_3            float   %8.4f                 Spline at 4,500
```

`polyspline` counts the reference points, and assumes that a quadratic is required, as the user has not specified otherwise. Note that the reference points are not regularly spaced.

Fitting a quadratic model with a reference–spline basis

We then fit a quadratic model of `mpg` with respect to `weight`, using `regress` with the `noconst` option:

```
. regress mpg qs_*, robust noconst;

Linear regression                                      Number of obs =      74
                                                       F(  3,    71) = 1204.74
                                                       Prob > F      =  0.0000
                                                       R-squared     =  0.9778
                                                       Root MSE      =  3.3587

------------------------------------------------------------------------------
             |               Robust
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        qs_1 |   28.16455   1.024978    27.48   0.000     26.12081     30.2083
        qs_2 |   20.62851   .4079148    50.57   0.000     19.81515    21.44187
        qs_3 |   14.29091   .9442261    15.14   0.000     12.40817    16.17364
------------------------------------------------------------------------------
```

The parameters are expected values of `mpg` at the 3 irregularly–spaced reference weights. These may be more informative than the usual parameters for a quadratic, expressed in miles per gallon per US pound, or in miles per gallon per squared US pound. *However*. . .

Listing the parameters of a quadratic model with a reference–spline basis

. . .as usual, the parameters can be listed more informatively, using the package `parmest`, with the `label`, `list()` and `format()` options:

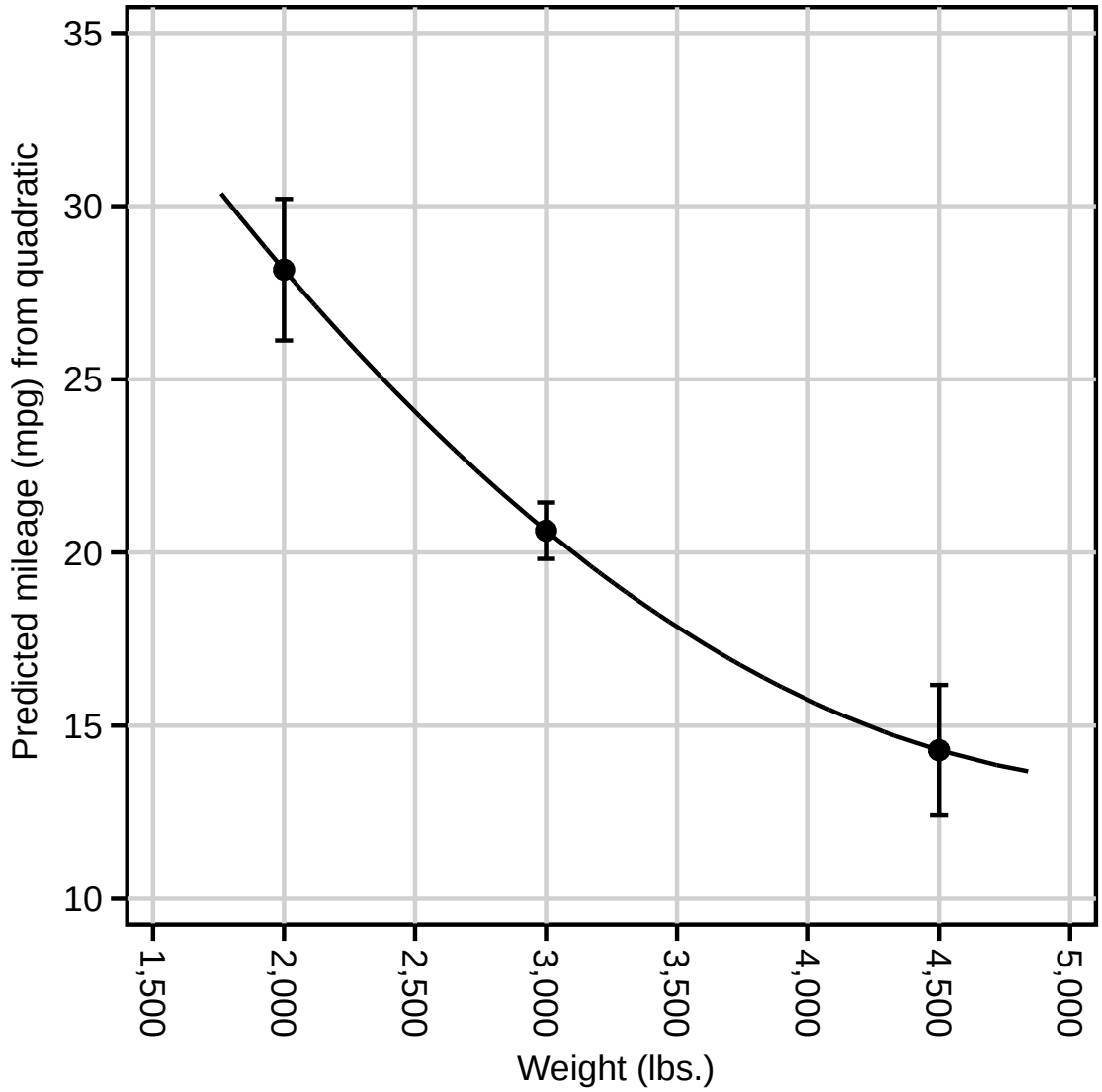
```
. parmest, label list(parm label estimate min* max*)
>   format(estimate min* max* %8.2f);
```

```
     +-------------------------------------------------+
     | parm             label   estimate   min95   max95 |
     |-------------------------------------------------|
  1. | qs_1   Spline at 2,000      28.16   26.12   30.21 |
  2. | qs_2   Spline at 3,000      20.63   19.82   21.44 |
  3. | qs_3   Spline at 4,500      14.29   12.41   16.17 |
     +-------------------------------------------------+
```

We can now see which expected value of `mpg` belongs to which of the 3 irregularly–spaced reference weights. And they are neatly formatted to 2 decimal places.

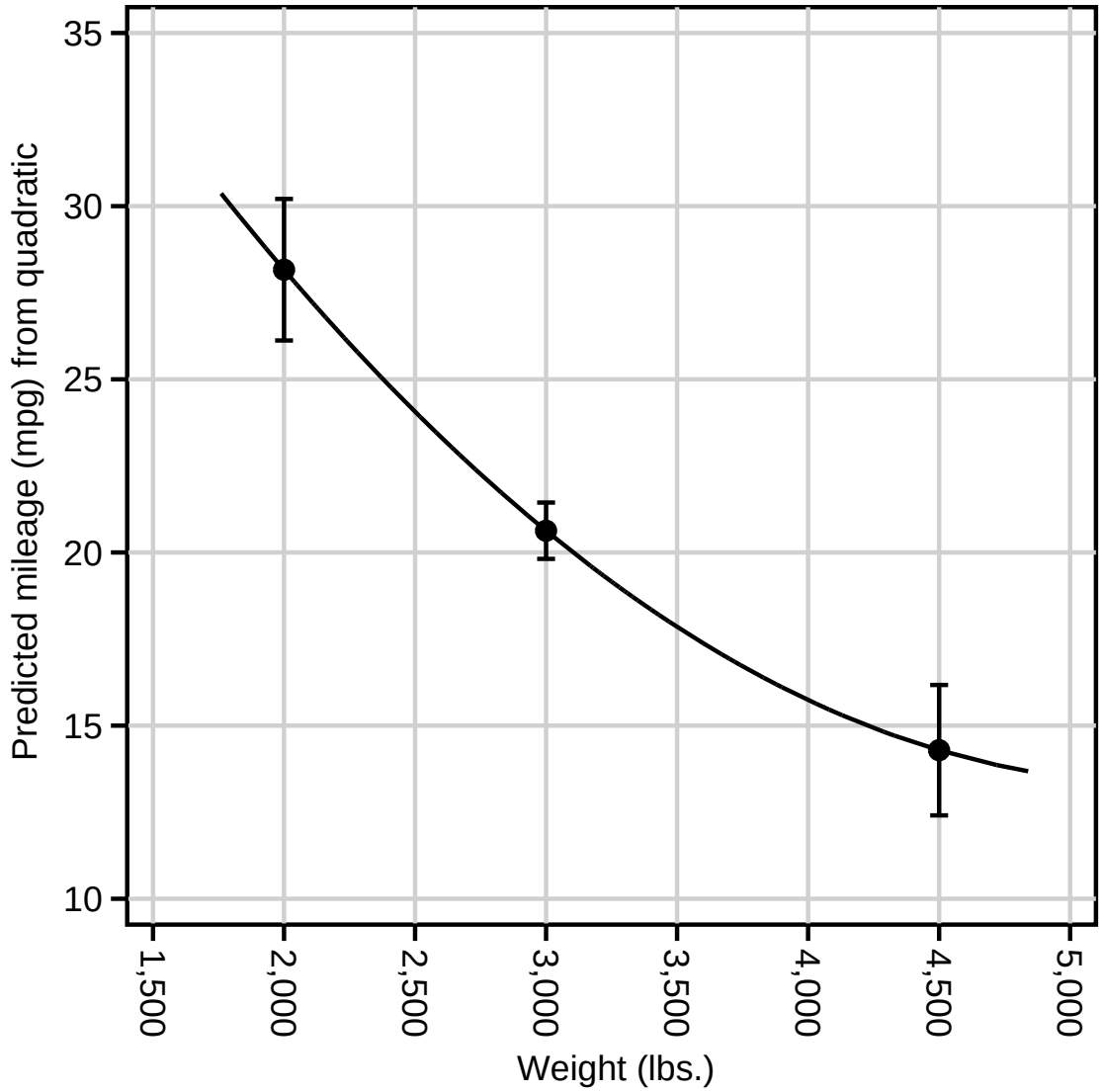
Plotting the predicted values with the reference–spline parmeters

- And it is even more informative to `append` the `parmest` resultsset to the main dataset, and plot the parameters and other predicted values.
- This is done using `eclplot`, with a `baddplot()` option to add the line.
- Unsurprisingly, the line passes through the parameter estimates.



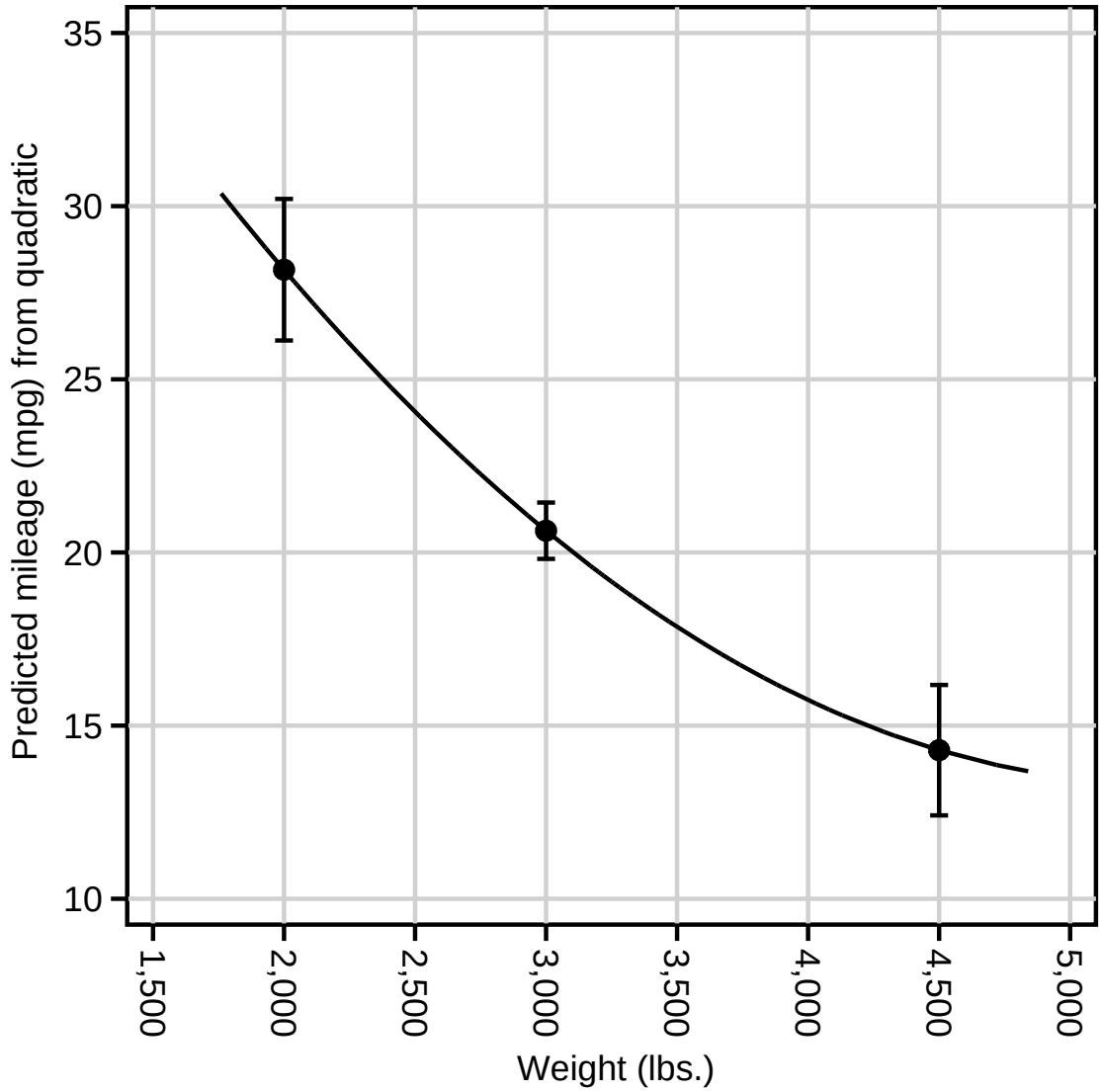
Plotting the predicted values with the reference–spline parmeters

- And it is even more informative to `append` the `parmest` resultsset to the main dataset, and plot the parameters and other predicted values.
- This is done using `eclplot`, with a `baddplot()` option to add the line.
- Unsurprisingly, the line passes through the parameter estimates.



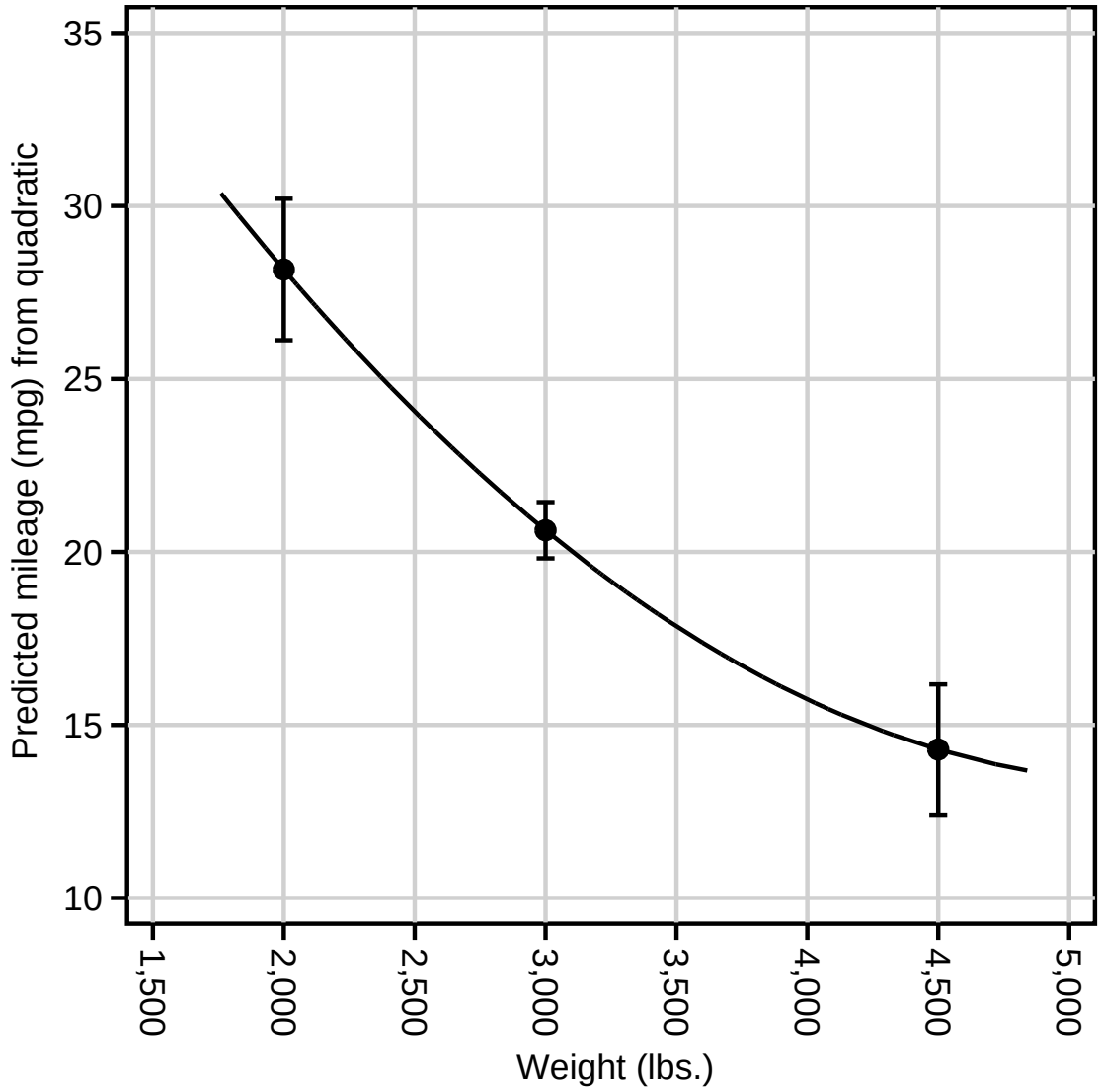
Plotting the predicted values with the reference–spline parmeters

- And it is even more informative to `append` the `parmest` resultsset to the main dataset, and plot the parameters and other predicted values.
- This is done using `eclplot`, with a `baddplot()` option to add the line.
- Unsurprisingly, the line passes through the parameter estimates.



Plotting the predicted values with the reference–spline parmeters

- And it is even more informative to `append` the `parmest` resultsset to the main dataset, and plot the parameters and other predicted values.
- This is done using `eclplot`, with a `baddplot()` option to add the line.
- Unsurprisingly, the line passes through the parameter estimates.



An alternative incomplete basis for the same quadratic model

Alternatively, we can use `polyspline` with a `base()` option to generate an alternative incomplete basis of reference splines for the same quadratic model, with a base weight of 2000 US pounds:

```
. polyspline weight, refpts(2000 3000 4500) base(2000) gene(bqs_);
3 reference splines generated of degree: 2

. desc bqs_*;

              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
bqs_1           byte    %8.4f                 Spline at 2,000
bqs_2           float   %8.4f                 Spline at 3,000
bqs_3           float   %8.4f                 Spline at 4,500
```

Again, `polyspline` counts the reference points, and assumes that a quadratic is required. *However*, the spline at the base weight of 2000 US pounds has been set to zero, and compressed to a byte variable to save space.

Fitting the same quadratic model with the alternative incomplete basis

Again, we fit the quadratic model using `regress`, this time without the `noconst` option:

```
. regress mpg bqs_*, robust;
note: bqs_1 omitted because of collinearity

Linear regression                                      Number of obs =      74
                                                       F(  2,    71) =   57.15
                                                       Prob > F      =  0.0000
                                                       R-squared     =  0.6722
                                                       Root MSE      =  3.3587

------------------------------------------------------------------------------
             |               Robust
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       bqs_1 |           0  (omitted)
       bqs_2 |  -7.536052   1.011027    -7.45   0.000    -9.551982   -5.520122
       bqs_3 |  -13.87364   1.319137   -10.52   0.000    -16.50392   -11.24336
       _cons |   28.16456   1.024978    27.48   0.000     26.12081    30.20831
------------------------------------------------------------------------------
```

This time, there is a parameter `_cons`, the first reference–spline parameter has been omitted and set to zero, and the other reference–spline parameters are negative, representing differences in mileage, compared to the base mileage.

Listing the parameters for the alternative incomplete basis

And, again, these parameters are more informative when listed using `parmest`:

```
. parmest, label list(parm label estimate min* max* p)
>   format(estimate min* max* %8.2f p %-8.2g);
```

	parm	label	estimate	min95	max95	p
1.	o.bqs_1	Spline at 2,000	0.00	0.00	0.00	.
2.	bqs_2	Spline at 3,000	-7.54	-9.55	-5.52	1.7e-10
3.	bqs_3	Spline at 4,500	-13.87	-16.50	-11.24	4.0e-16
4.	_cons	Constant	28.16	26.12	30.21	1.4e-39

We see that the omitted parameter belongs to the base reference weight of 2000 US pounds, whose expected mileage is the constant term. The other parameters are the differences in mileage, predicted by the quadratic model, for reference weights of 3000 and 4500 US pounds, respectively. Their confidence limits and *P*–values show these quadratic effects to be significantly negative.

A linear spline basis for the same 3 reference weights

`polyspline` can also fit non–polynomial splines, with lower degree than the default. For these 3 reference weights, we can use the option `power(1)` to generate a basis of 3 linear reference splines:

```
. polyspline weight, refpts(2000 3000 4500) power(1) gene(ls_);
3 reference splines generated of degree: 1

. desc ls_*;

              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
ls_1            float   %8.4f                 Spline at 2,000
ls_2            float   %8.4f                 Spline at 3,000
ls_3            float   %8.4f                 Spline at 4,500
```

This time, no base reference weight has been specified. *So*. . .

Fitting a linear spline model for the same 3 reference weights

. . .when we fit the linear spline model, using `regress` with the `noconst` option, the parameters are the expected values of mileage, under the linear spline model, at the same 3 reference weights:

```
. regress mpg ls_*, robust noconst;

Linear regression                                      Number of obs =      74
                                                       F(  3,    71) = 1192.80
                                                       Prob > F      =  0.0000
                                                       R-squared     =  0.9778
                                                       Root MSE      =  3.3545

------------------------------------------------------------------------------
             |               Robust
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        ls_1 |   28.24616   1.063046    26.57   0.000     26.12651    30.36581
        ls_2 |   19.94222   .6206224    32.13   0.000     18.70473     21.1797
        ls_3 |   14.08828   .8485617    16.60   0.000     12.39629    15.78026
------------------------------------------------------------------------------
```

Unsurprisingly, the 3 reference mileages are similar to (but not exactly the same as) the ones estimated using the quadratic model. . .

Listing the linear–spline parameters for the same 3 reference weights

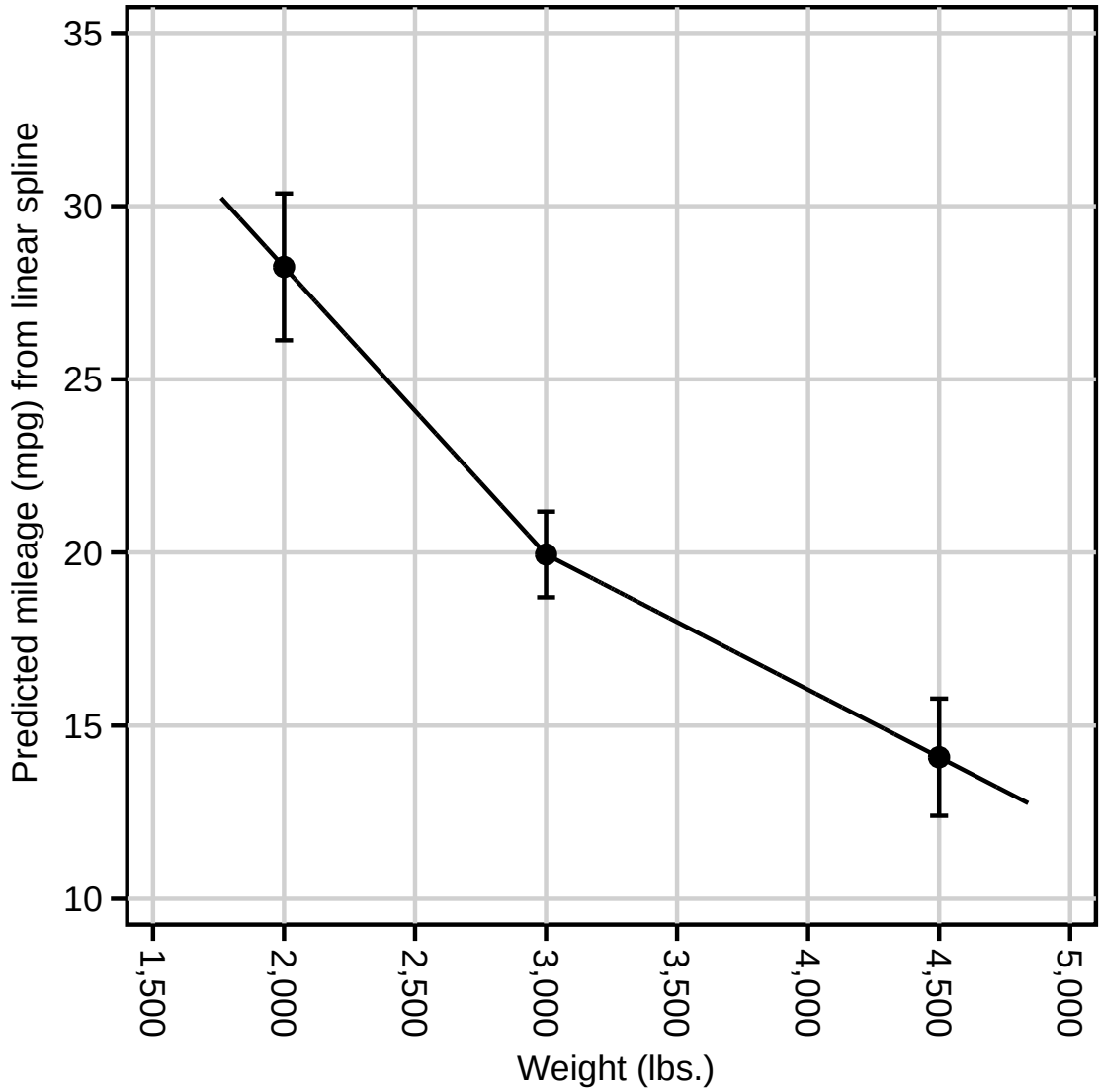
. . .and this is made clearer when these reference mileages are listed using `parmest`:

```
. parmest, label list(parm label estimate min* max*)
>   format(estimate min* max* %8.2f);

     +-------------------------------------------------+
     | parm            label   estimate   min95   max95 |
     |-------------------------------------------------|
  1. | ls_1   Spline at 2,000      28.25   26.13   30.37 |
  2. | ls_2   Spline at 3,000      19.94   18.70   21.18 |
  3. | ls_3   Spline at 4,500      14.09   12.40   15.78 |
     +-------------------------------------------------+
```

Again, we can now see which reference mileage belongs to which reference weight. *However*, it might be even more informative to see the predicted mileages at non–reference weights, too.

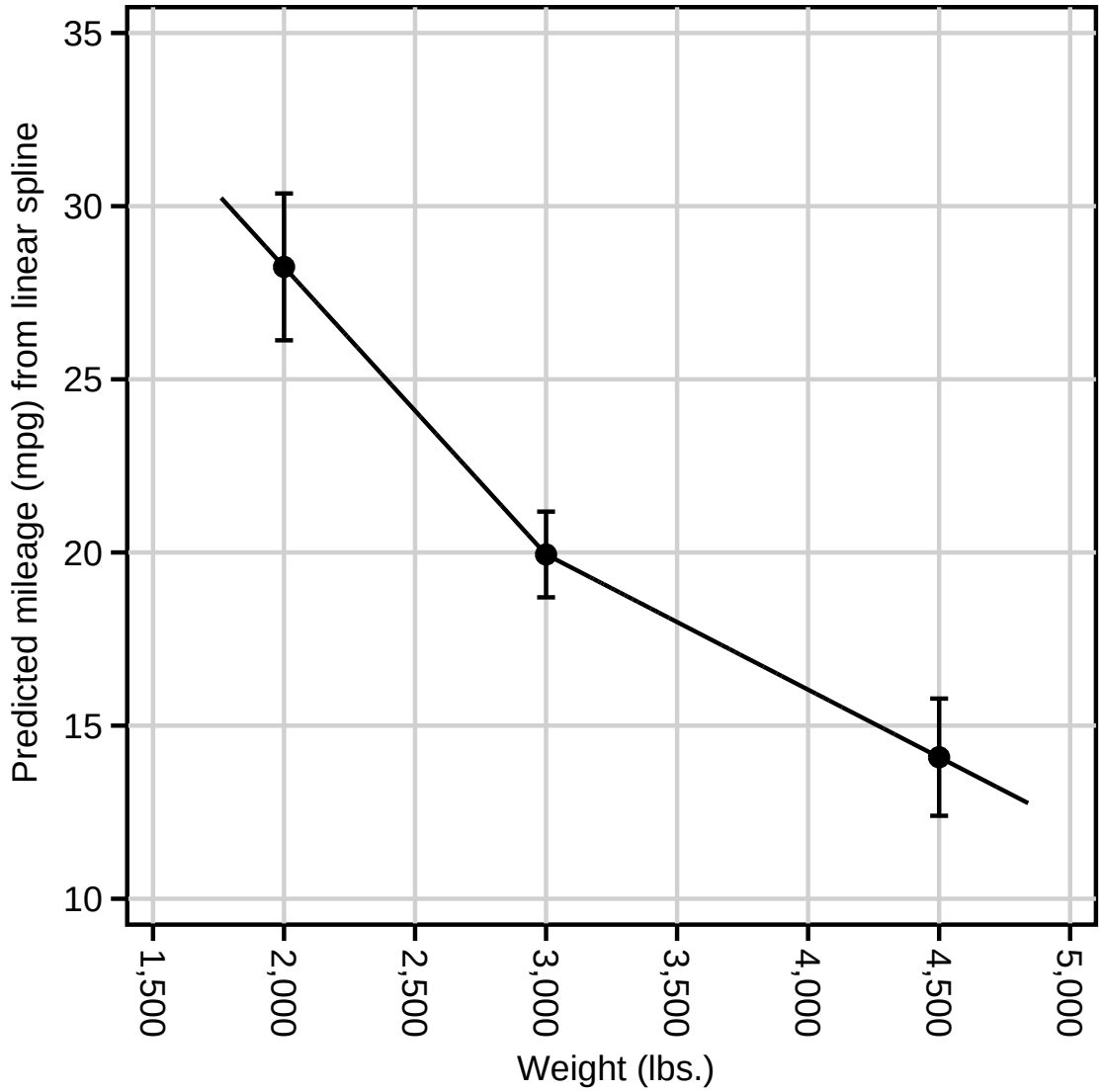
Plotting the predicted values with the reference–spline parameters

- And here they are, thanks once again to the `baddplot()` option of `eclplot`.
- We see that the spline is *mostly* linear, except for a single knot.
- `polyspline` has placed this knot at a "sensible" position, in this case at the middle one of the 3 irregularly–spaced reference weights.



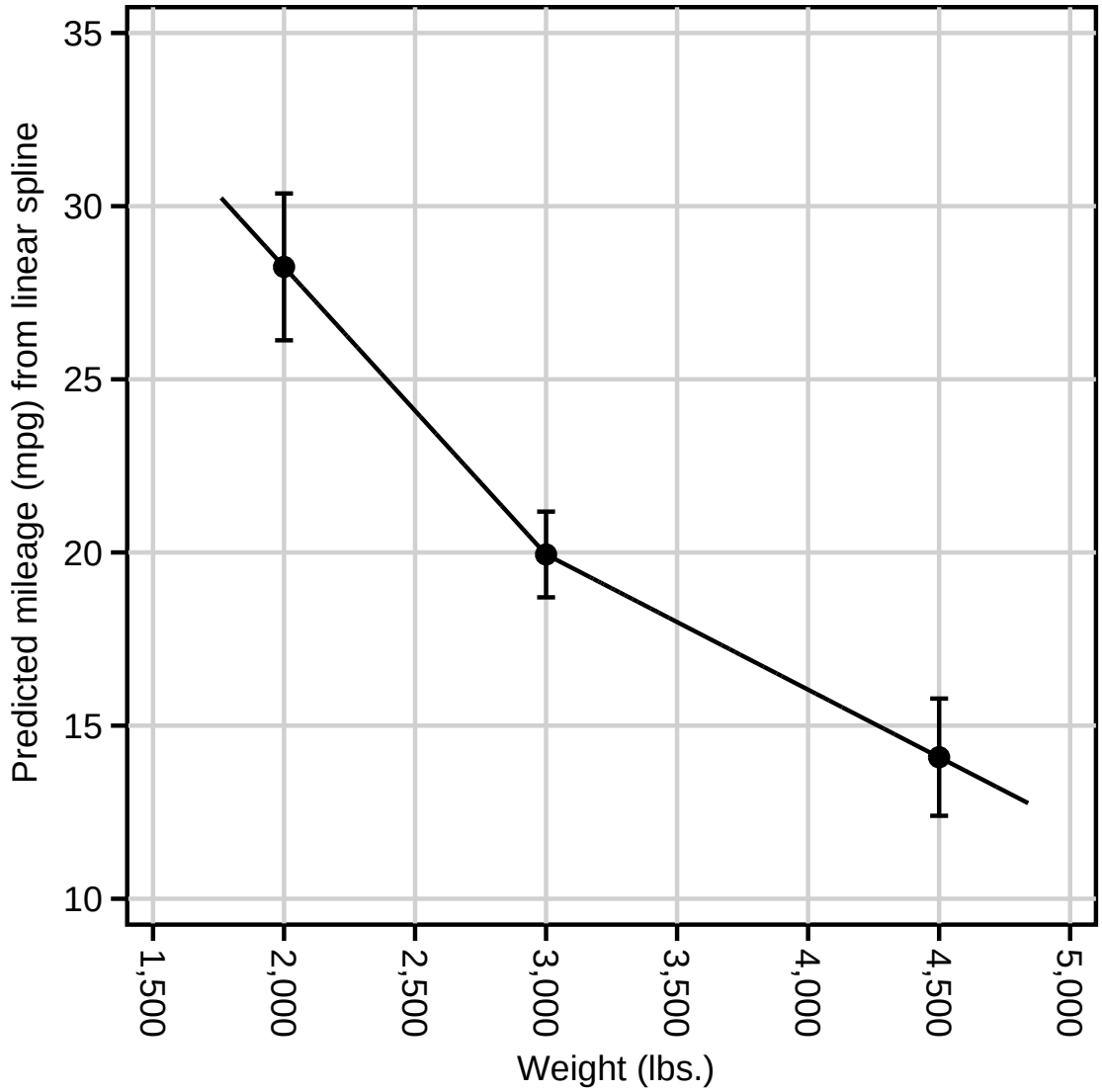
Plotting the predicted values with the reference–spline parameters

- And here they are, thanks once again to the `baddplot()` option of `eclplot`.
- We see that the spline is *mostly* linear, except for a single knot.
- `polyspline` has placed this knot at a "sensible" position, in this case at the middle one of the 3 irregularly–spaced reference weights.



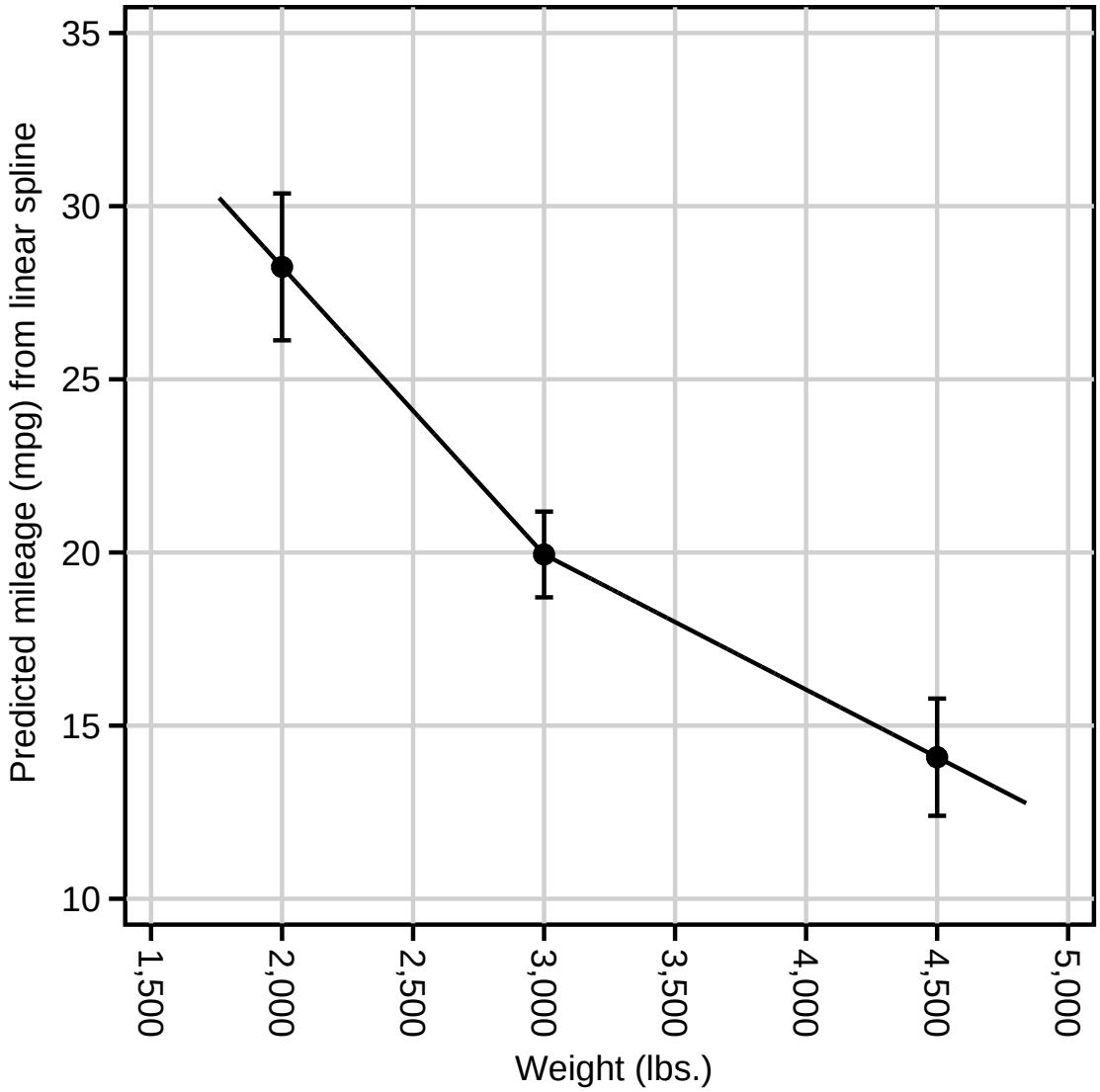
Plotting the predicted values with the reference–spline parameters

- And here they are, thanks once again to the `baddplot()` option of `eclplot`.
- We see that the spline is *mostly* linear, except for a single knot.
- `polyspline` has placed this knot at a "sensible" position, in this case at the middle one of the 3 irregularly–spaced reference weights.



Plotting the predicted values with the reference–spline parameters

- And here they are, thanks once again to the `baddplot()` option of `eclplot`.
- We see that the spline is *mostly* linear, except for a single knot.
- `polyspline` has placed this knot at a "sensible" position, in this case at the middle one of the 3 irregularly–spaced reference weights.



References

[1] Hampel, F. R., E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel. 1986. *Robust statistics. The approach based on influence functions.* New York, NY: Wiley.

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

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

[4] Newson, R. B. 2011. Sensible parameters for polynomials and other splines. Presented at the *17th UK Stata User Meeting*, 15–16 September, 2011. Downloadable from the conference website at *http://ideas.repec.org/p/boc/usug11/01.html*

[5] Newson, R. B. 2012. Sensible parameters for univariate and multivariate splines. *The Stata Journal* **12(3)**: 479–504.

This presentation, and the do–file producing the examples in the `auto` data, can be downloaded from the conference website at *http://ideas.repec.org/s/boc/usug14.html*

The packages used in this presentation can be downloaded from SSC, using the `ssc` command.