Regression III: Advanced Methods

Bill Jacoby

Michigan State University

Goals of the lecture

- Introduce nonparametric regression:
- ***Bivariate local polynomial regression*** (loess)
  - bandwidths, degree of polynomials, weights
  - degrees of freedom and inference
- Introduce ***multiple nonparametric regression***, and discuss the limitations of multiple local polynomial regression
- Show how nonparametric regression can be used to test for nonlinearity

Why Nonparametric Regression?

- The linear model is desirable because it is simple to fit, results are easy to understand, and there is a wide variety of useful techniques for testing the assumptions involved
  - Nonetheless, there are cases when the linear model should not be applied because of an intrinsic nonlinearity in the data
- Nonparametric regression provides a means for modelling such data
  - Nonparametric regression can be used as a benchmark for linear models against which to test the linearity assumption
- Nonparametric regression also provides a useful way to enhance scatterplots to display underlying structure in the data—as we have already seen, it plays an important role in diagnostic plots

Introducing Nonparametric Regression

- Recall that linear regression traces the *conditional distribution* of a dependent variable *Y*, as a function of one or more explanatory variables *X's*

$$\begin{aligned} p(y|x_1, ..., x_k) &= f(x_1, ..., x_k) \\ &= \alpha + \beta_1 X_1 + \beta_2 X_2 \end{aligned}$$

- We have already discussed some measures to accommodate certain nonlinear relationships between a quantitative dependent variable and quantitative predictors within the linear framework (*e.g.,* transformations and polynomial regression)
- *Nonparametric regression* provides yet another way of modelling nonlinear relationship, by ***entirely relaxing the linearity assumption***

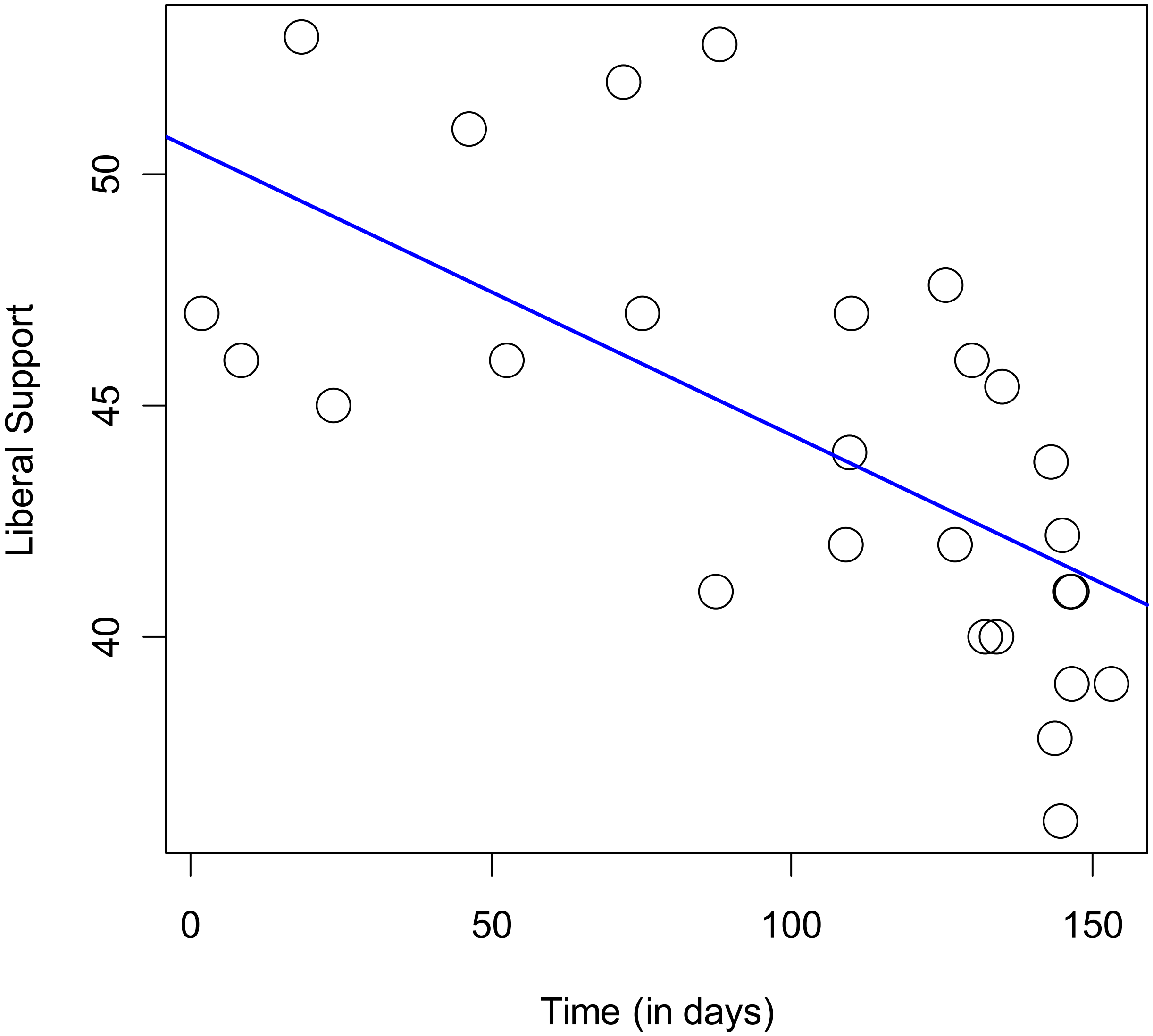
Types of nonparametric regression

- There are several types of nonparametric regression. The most commonly used is the ***lowess (or loess)*** procedure first developed by Cleveland (1979)
  - Lowess is an acronym for ***locally weighted scatterplot smoother***. Loess stands for ***local regression***. Both are essentially the same thing.
  - These models essentially fit local polynomial regressions and join them together
- Another important set of nonparametric models are ***smoothing splines***
  - These models partition the data and fit separate piecewise regressions to each section, smoothing them together where they join
- Loess can be generalized to several independent variables (but, an alternative approach is typically used instead)

Loess: Local polynomial regression How does it work?

- The following example explains the most common type of nonparametric regression—***loess***, which is essentially locally weighted polynomial regression
- The data are from 27 pre-election polls conducted before the 1997 Canadian election
  - ***Time***: Measured in number of days from January 1st, 1997
  - ***Liberal***: Percentage of respondents in the polls supporting the Liberal Party
- A scatterplot shows the possibility of a nonlinear relationship, so we'll try to explore this relationship using lowess
- In the following slides I show how a loess smooth is fit step-by-step

Liberal Support, January 1-June 1, 1997

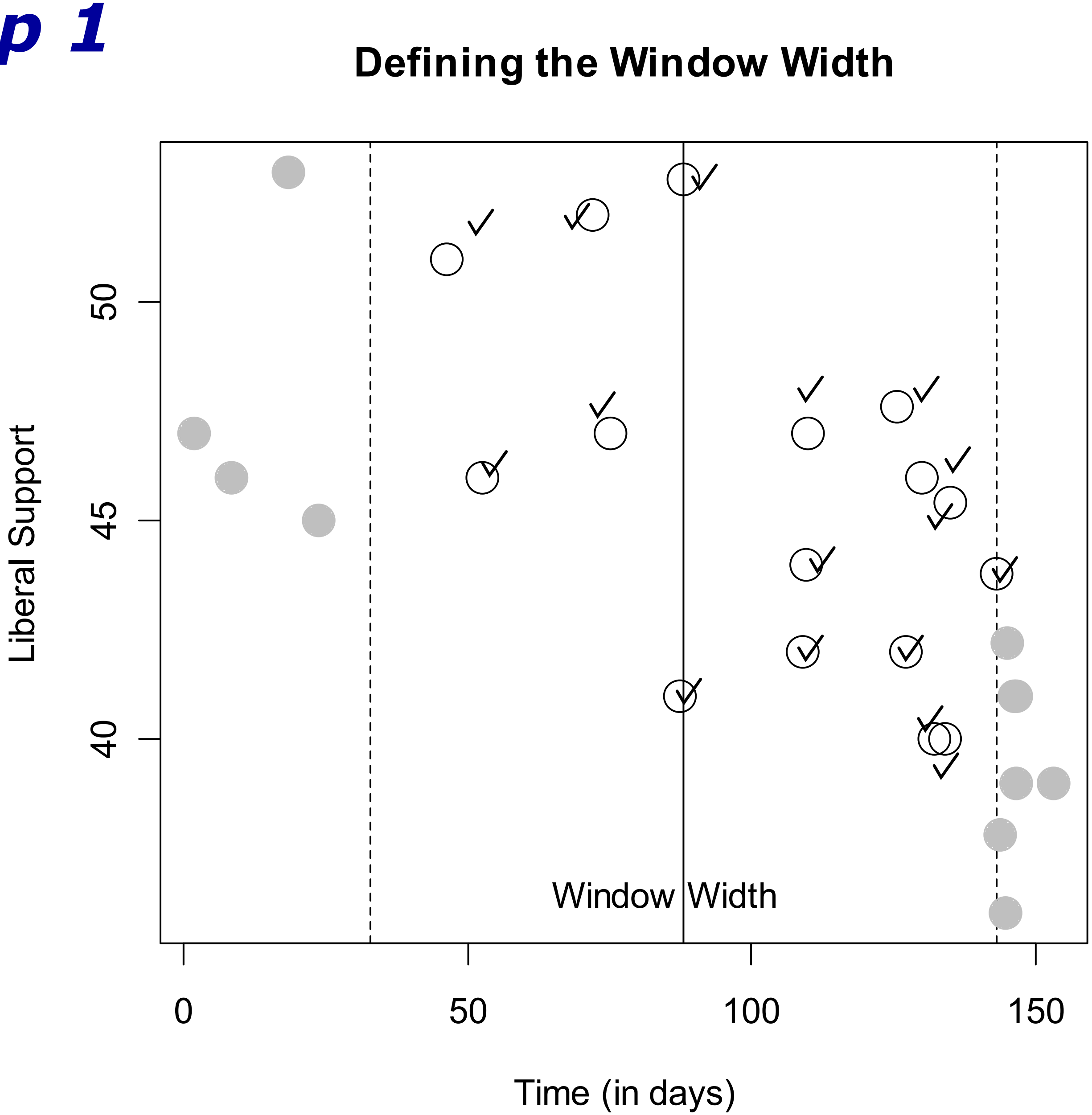


Lowess: Local Polynomial Regression
Step 1: Defining the window width

- The first step is to define the ***window width*** m, that encloses the closest neighbours to each data observation (the ***window half-width*** is labelled h)
  - For this example, we use m=16 (*i.e.*, for each data point we select the 16 nearest neighbours in terms of their *X*-value)
    - 16 was chosen to represent 60% of the data
    - The researcher typically chooses the window width by trial and error (more on this later)
  - The graph on the following page shows the 16 closest observations to $X_{(10)}$=88. Here we call $X_{(10)}$ our ***focal X***
- Although for this example I start at $X_{(10)}$, in the real case we would start with the first observation and move through the data, finding the 16 closest observations to each case

Local Polynomial Regression

Step 1



Local Polynomial Regression
Step 1 (R-script)

```
#Defining the window width
plot(TIME, LIBERAL, xlab="Time (in days)", ylab="Liberal Support",
    type=’n’, main="Defining the Window Width")
ord <- order(TIME)
time <- TIME[ord]
pre <- LIBERAL[ord]
x0 <- time[10]
diffs <- abs(time - x0)
which.diff <- sort(diffs)[16]
abline(v=c(x0-which.diff, x0+which.diff), lty=2)
abline(v=x0)
points(time[diffs > which.diff], Lib[diffs > which.diff], pch=16, cex=2, col=gray(.75))
points(time[diffs <= which.diff], Lib[diffs <= which.diff],cex=2)
x.n <- time[diffs <= which.diff]
y.n <- Lib[diffs <= which.diff]
text(locator(1), "Window Width")
```

Local Polynomial Regression
Step 2: Weighting the data

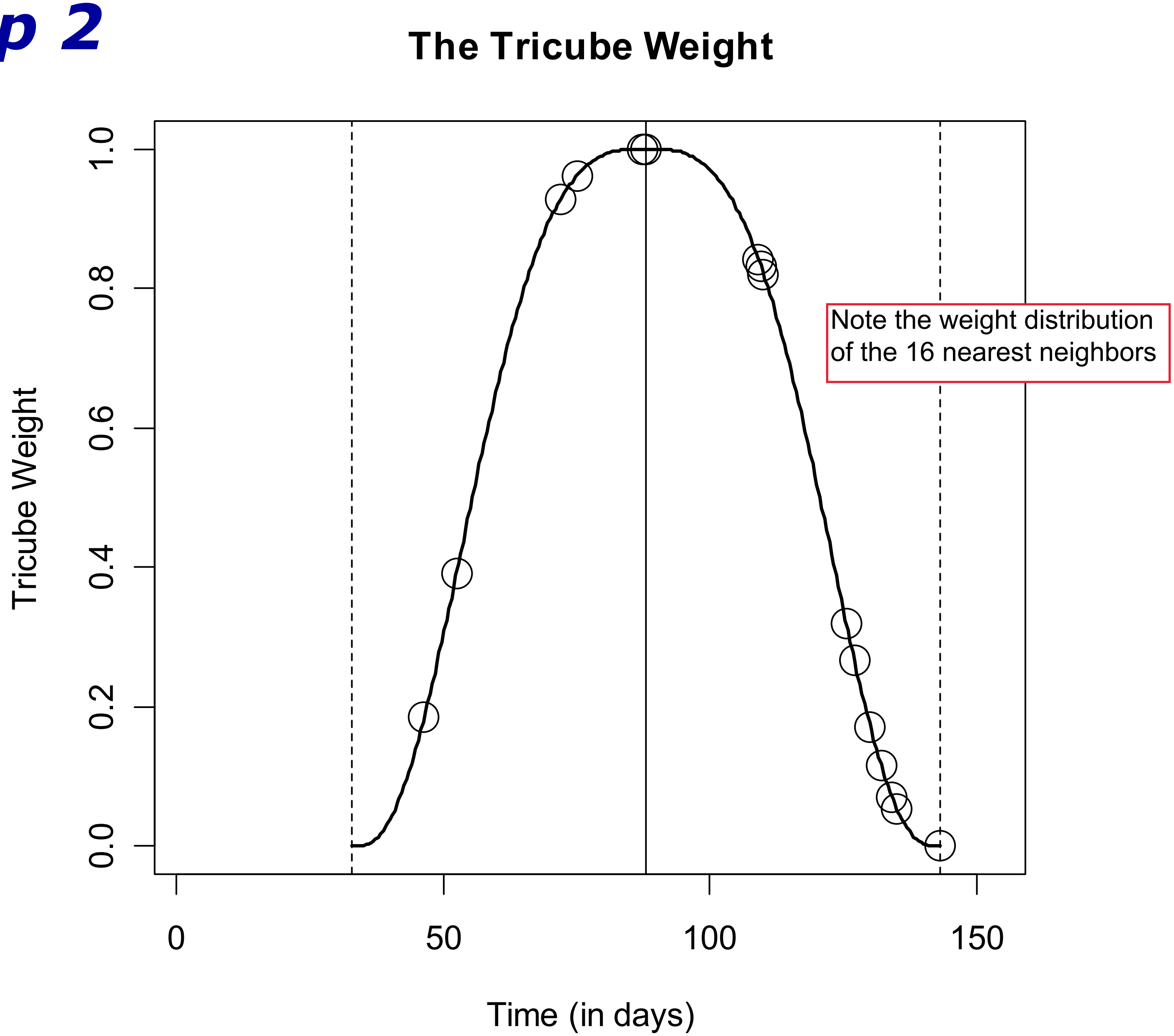
- We then choose a weight function to give greatest weight to observations that are closest to the focal *X* observation
  - In practice, the ***tricube weight*** function is usually used
- Let $z_i=(x_i-x_0)/h_i$, which is the scaled distance between the predictor value for the *i*th observation and the focal x

$$W_T(z) = \begin{cases} \left(1 - |z|^3\right)^3 & \text{for } |z| < 1 \\ 0 & \text{for } |z| \geq 1 \end{cases}$$

Here h_i is the half-width of the window centred on x_i

- Notice that observations more than *h* (the half-window or bandwidth of the local regression) away from the focal x receive a weight of 0

Local Polynomial Regression
Step 2



Note the weight distribution of the 16 nearest neighbors

Local Polynomial Regression
Step 2: R-script

```
#The Tricube Weight
tricube <- function(z) {
    ifelse (abs(z) < 1, (1 - (abs(z))^3)^3, 0)
    }
plot(range(TIME), c(0,1), xlab="Time (in days)",
           ylab="Tricube Weight",
           type=’n’, main="The Tricube Weight")
abline(v=c(x0-which.diff, x0+which.diff), lty=2)
abline(v=x0)
xwts <- seq(x0-which.diff, x0+which.diff, len=250)
lines(xwts, tricube((xwts-x0)/which.diff), lty=1, lwd=2)
points(x.n, tricube((x.n - x0)/which.diff), cex=2)
```

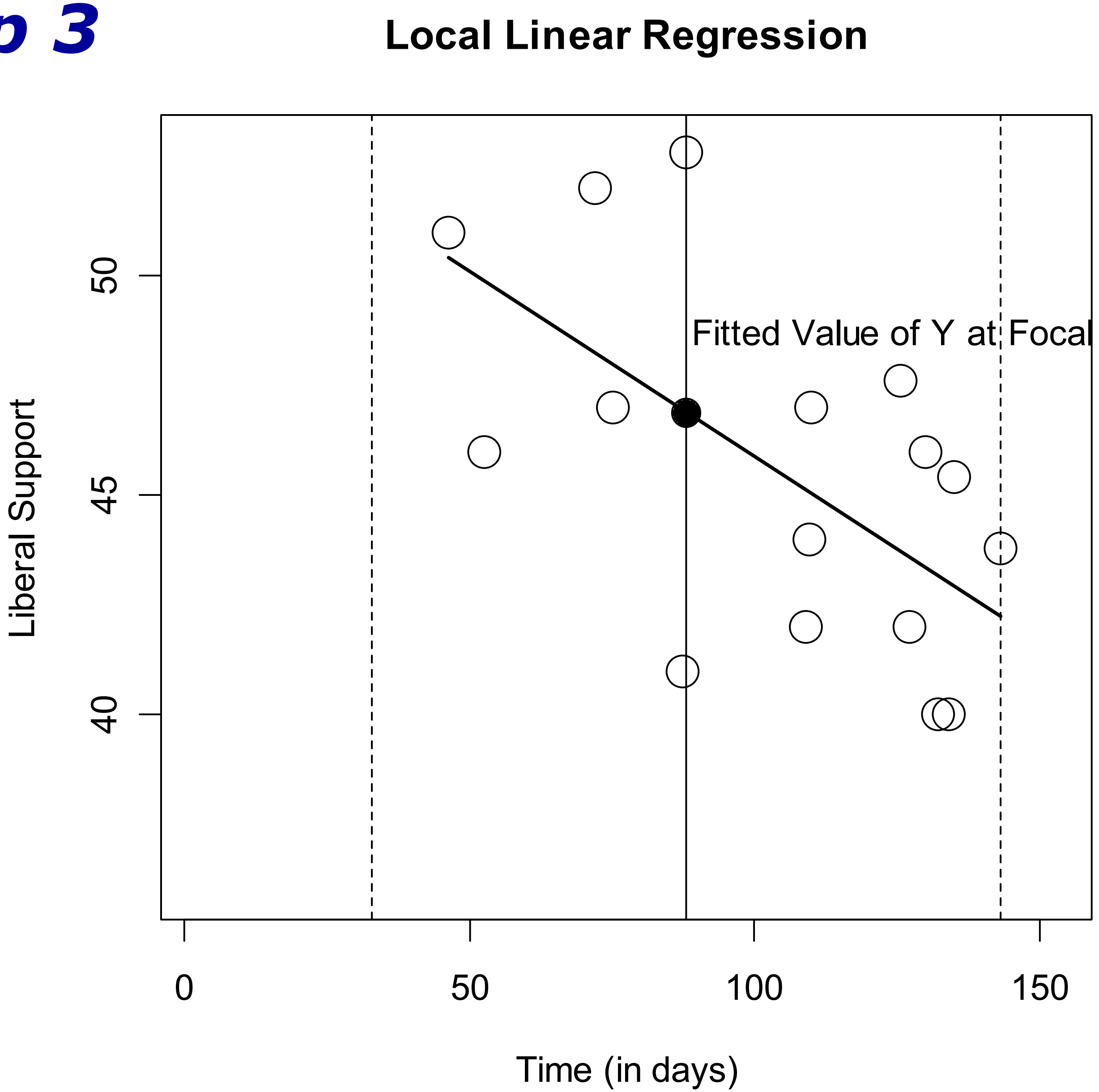
Local Polynomial Regression
Step 3: Locally weighted least squares

- A ***polynomial regression*** using ***weighted least squares*** (using the tricube weights) is then applied to the focal X observation, using only the nearest neighbour observations to ***minimize the weighted residual sum of squares***
  - Typically a local linear regression or a local quadratic regression is used, but higher order polynomials are also possible

$$Y_i = A + B_1(x_i - x_0) + B_2(x_i - x_0)^2 + \cdots + B_p(x_i - x_0)^p + E_i$$

- From this regression, we then calculate the ***fitted value*** for the focal *X* value and plot it on the scatterplot
  - The regression line within the window in the following graph shows the fitted value for the focal x_i from a local linear regression

Local Polynomial Regression
Step 3

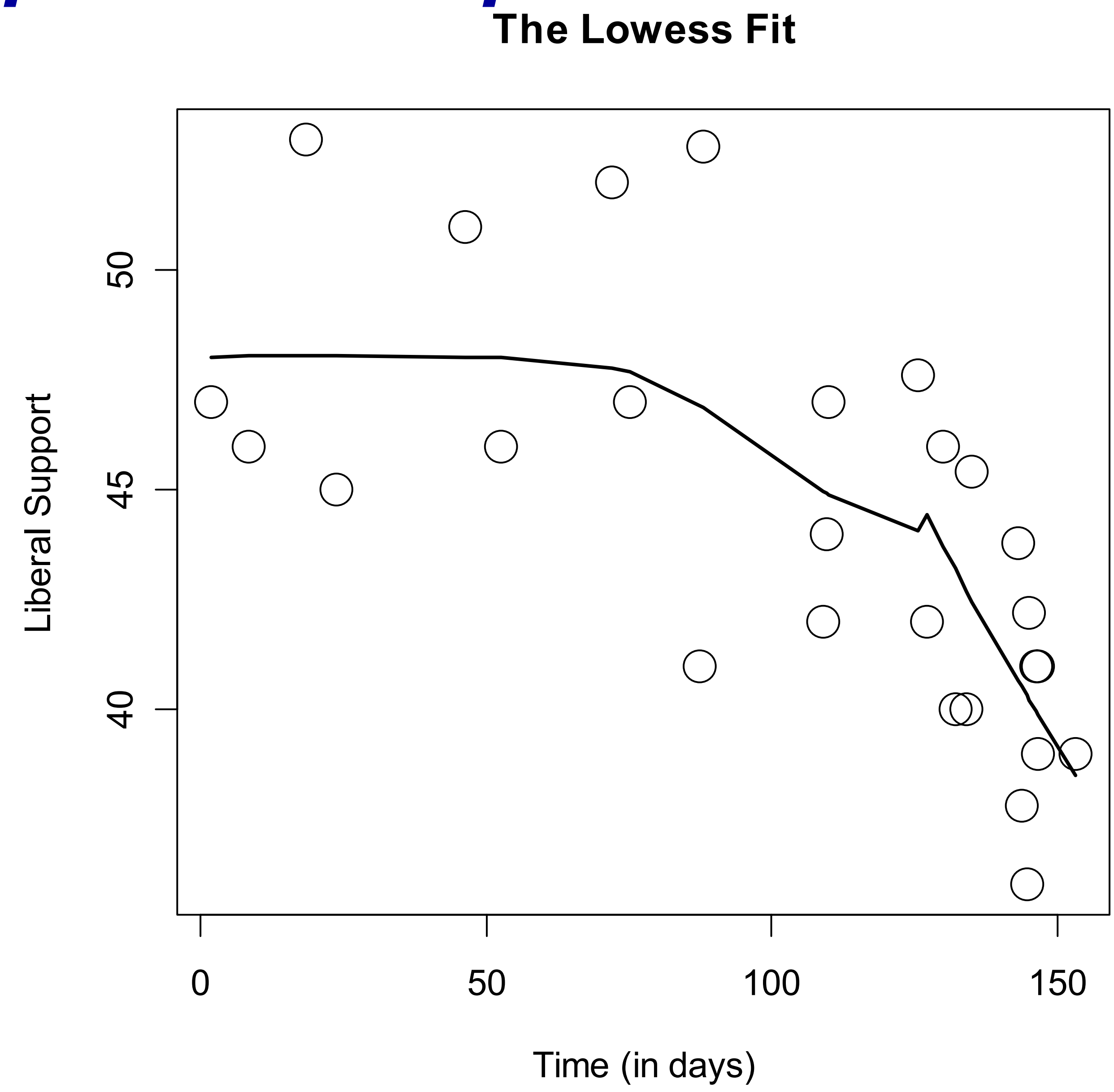


Locally Weighted Averaging
Step 4: The Nonparametric Curve

- Steps 1-3 are carried out for each observation in the data
  - There is a separate local regression for each value of X
  - A fitted value from these regressions for each focal X is plotted on the scatterplot
- The fitted values are connected, producing the local polynomial ***nonparametric regression curve***
  - As we shall see later, the larger the window width, the smoother the curve
  - Also, it is important to remember that there are no coefficient estimates—***The relationship between X and Y must be graphed***

```
plot(TIME, LIBERAL, xlab="Time (in days)", ylab="Liberal Support",
    main="The Lowess Fit", cex=2)
lines(lowess(TIME, LIBERAL, f=0.6, iter=0), lwd=2)
```

Locally Weighted Averaging
Step 4: The Nonparametric Curve



Robustness Weights
Adjusting for outliers (1)

- Since we are trying to determine an underlying structure in the data, ***we don't want unusual cases to have extraordinary influence on the curve***
- Following from the linear regression case, M-Estimation for robust regression can be adapted to ensure that the lowess smooth is not unduly affected by outliers

1. We start by calculating the residuals from the fitted values for the local regressions

$$E_i = Y_i - \widehat{Y}_i$$

2. Determine a measure of the scale of the residuals (*e.g.*, the *median absolute deviation* from the *median residual*):

$$\text{MAD} = \text{median}|E_i - \tilde{E}|$$

$$\text{where } \tilde{E} = \text{median}(E_i)$$

Robustness Weights
Adjusting for outliers (1)

3. Calculate resistance weights v_i for each observation using an appropriate weight function to determine the relative size of each residual. Here we use the ***Bisquare Weight Function***:

$$v_i \equiv w_B(z) = \begin{cases} \left(1 - z_i^2\right)^2 & \text{for } |z| < 1 \\ 0 & \text{for } |z_i| \geq 1 \end{cases}$$

$$\text{where } z = \frac{E_i}{t \times \text{MAD}}$$

and t is a tuning constant

- t=6 MADs corresponds approximately to 4 standard deviations. In other words, we exclude observations that have a probability of being observed of less than 0.0001.

Robustness Weights
Adjusting for outliers (3)

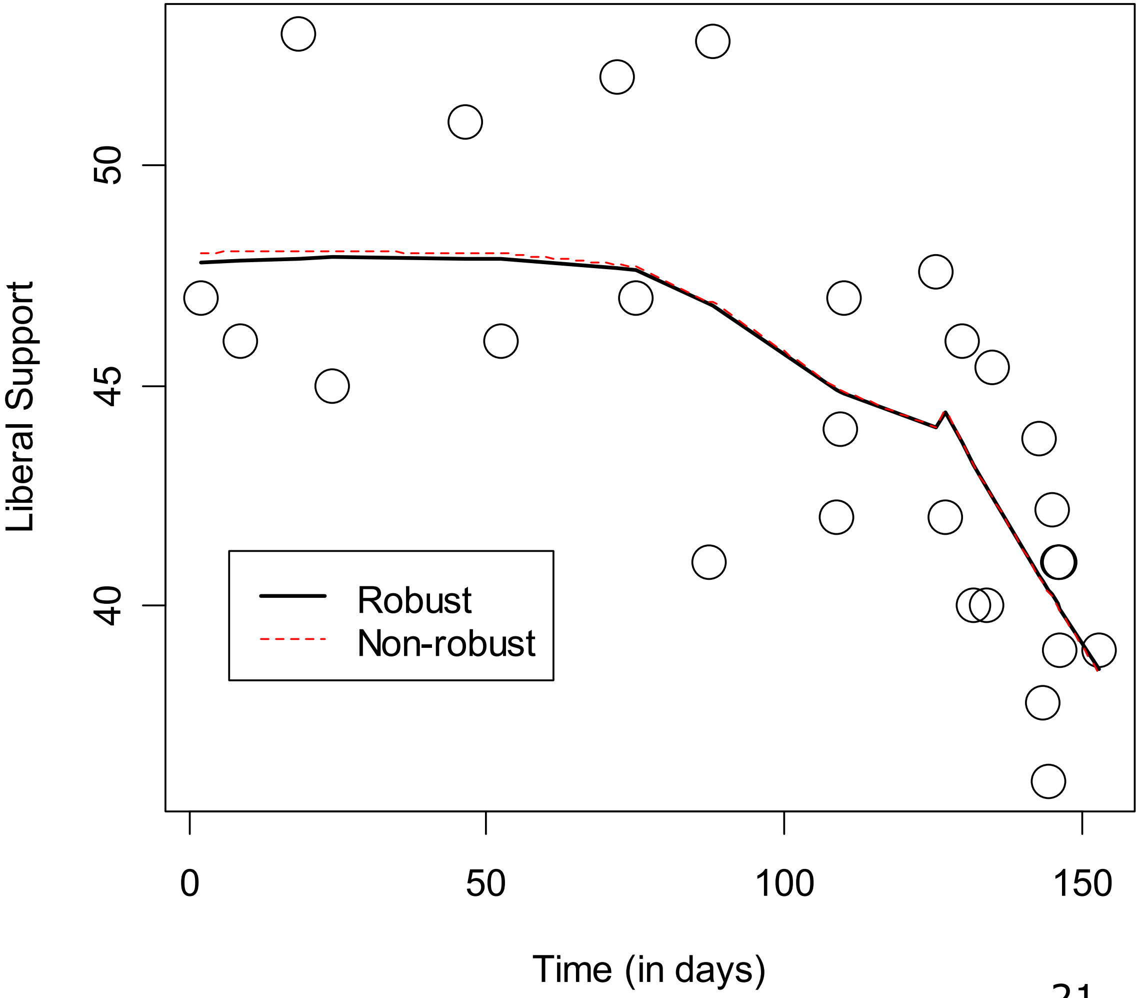
4. We then refit the local polynomial regressions using both local weights (w_i) and the resistance weights (v_i)
5. From these new regressions, we calculate new fitted values
6. Steps 1-4 are repeated (iterated) until the fitted values stabilize
7. Finally, a curve is drawn to ***connect the fitted values***, giving us the ***lowess smooth***

```
plot(TIME, LIBERAL, xlab="Time (in days)", ylab="Liberal Support",
    main="The Robust Lowess Fit", cex=2)
lines(lowess(TIME, LIBERAL, f=0.6), lwd=2)
lines(lowess(TIME, LIBERAL, f=0.6, iter=0),
     lty=2, col="red")
legend(locator(1), lty=c(1:2), lwd=c(2,1),
     col=c("black", "red"),
     legend=c(’Robust’, ’Non-robust’))
```

Robustness Weights
Adjusting for outliers (4)

- In this case the robust fit is nearly identical to the regular lowess fit, indicating that outliers are not problematic
- Nonetheless, most lowess procedures use the robust fit by default (`locfit` is an exception)

The Robust Lowess Fit



Interpreting the Local Regression Estimate

- In linear regression our interest is in the regression coefficients, in particular the slopes
  - Our interest, then, is in how well the estimated coefficients represent the true population coefficients
  - We focus on confidence intervals and t-test for individual coefficients
- In nonparametric regression we have ***no parameter estimates*** (hence the name “nonparametric”)
  - Our interest is on the fitted curve
  - We calculate estimates and confidence intervals (or envelopes) but they are with respect to the complete curve rather than a particular estimate
  - In other words, we focus on how well the estimated curve represents the population curve

Lowess in R (1)

- Local polynomial regression (or lowess smooths) can be fit several ways in **R**, using several different packages:
  - The base package of **R** contains the `lowess` function and the `loess` function (the latter is preferred and used to be in the `modreg` package)
  - The `locfit` package contain `locfit` and `locfit.robust` functions
  - The `sm` package contains the `sm.regression` function
- Each package has some comparative strengths and weakness
  - The unfortunate thing is that they all do not specify the models in the same way—*i.e.*, they have different model structure commands, and different ways of specifying the window width m (or window span S)

Lowess in R (2)

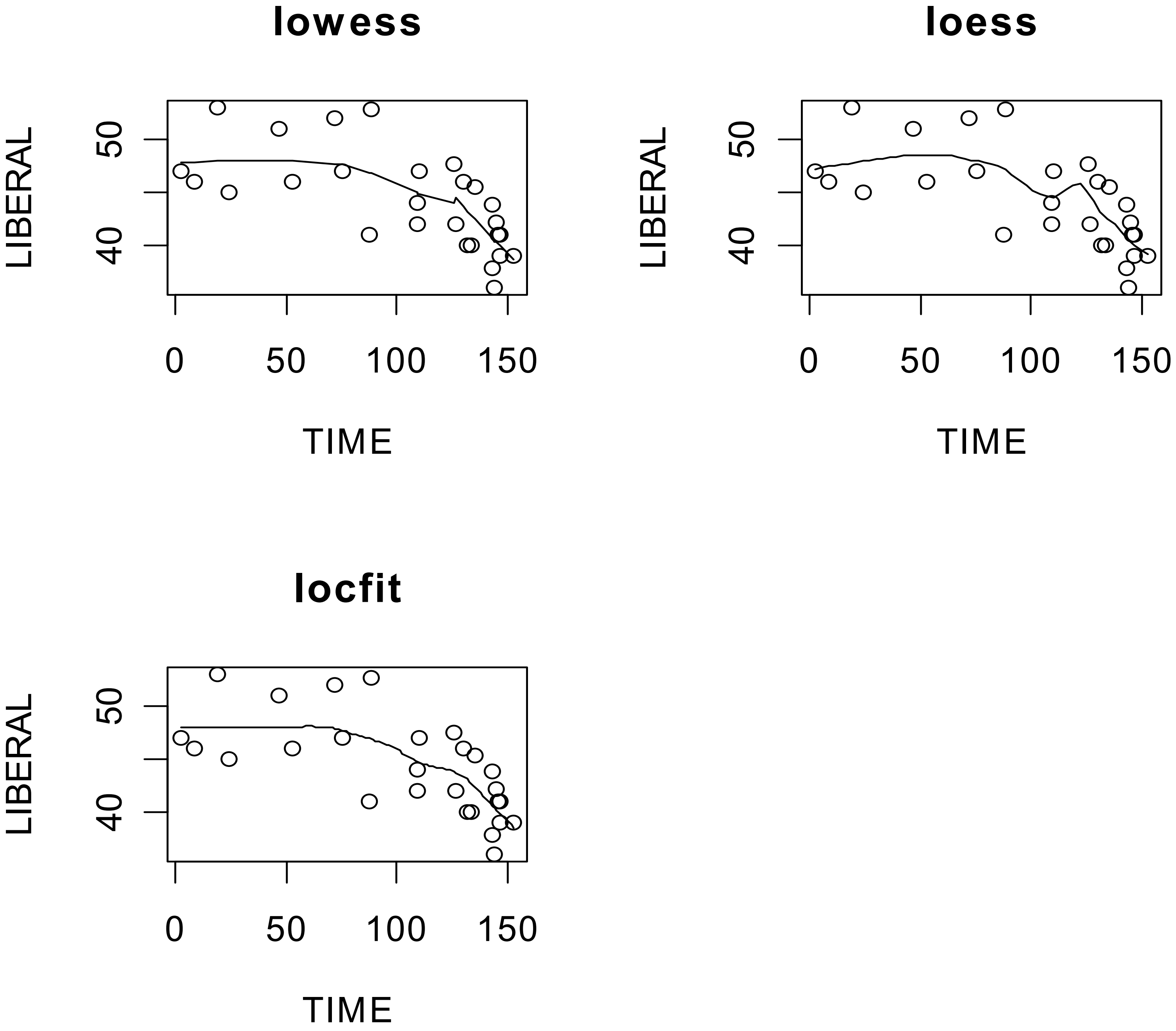
- Below I have specified the same model using the four functions. Notice the span (0.6—or 60% of the data. More about this later) is specified differently for each function. The model command also differs

```
#lowess in the base package:
model.lowess<-lowess(TIME,LIBERAL, f=0.6)
#lowess in modreg package:
library(modreg)
model.loess<-loess(LIBERAL~TIME, span=0.6, degree=1)
#lowess in locfit
library(locfit)
model.locfit<-locfit(LIBERAL~TIME, alpha=0.6, deg=1)
#nonparametric smooths in sm
library(sm)
model.sm<-sm.regression(TIME, LIBERAL, h=0.3)
```

Lowess in R (3)

- The separate functions also have different ways of graphing the loess smooth

```
split.screen(figs=c(2,2))
#lowess
screen(1)
plot(TIME, LIBERAL, main="lowess")
lines(model.lowess)
#loess
screen(2)
scatter.smooth(TIME, LIBERAL, span = .4, degree = 1, main="loess")
#locfit
screen(3)
plot(TIME,LIBERAL, main="locfit")
lines(model.locfit)
close.screen(all = TRUE)
```


lowess
LIBERAL
40
50
0
50
100
150
TIME
loess
LIBERAL
40
50
0
50
100
150
TIME
locfit
LIBERAL
40
50
0
50
100
150
TIME

Assumptions under the Loess Model

- The assumptions under the Loess model are much less restrictive than the assumptions for the linear model
  - Most importantly, no strong global assumptions are made about μ (the conditional mean of Y)
  - We assume, however, that locally around a point x, μ can be approximated by a small class of parametric functions (polynomial regression)
  - Still, the errors ε_i are assumed independent and randomly distributed with mean 0
- Finally, a number of choices—particularly window width, type of polynomial and type of weight function—affect the trade-off between bias and variance

Window Span (1)
Trade-off between bias and variance

- Recall that the window width m, is the ***number of cases*** in each local regression (the bandwidth, *h*, is half the window size)
  - It is more practical to to think in terms of ***percentage of cases*** across the range of X, which we call the span S
- The size of S has an important effect on the curve
- A ***span that is too small*** (meaning that insufficient data fall within the window) produces a curve characterised by a lot of noise
  - In other words, this results in a ***large variance***
- If the ***span is too large***, the regression will be over-smoothed and thus the local polynomial may not fit the data well
  - This may result in loss of important information, and thus the fit will have ***large bias***

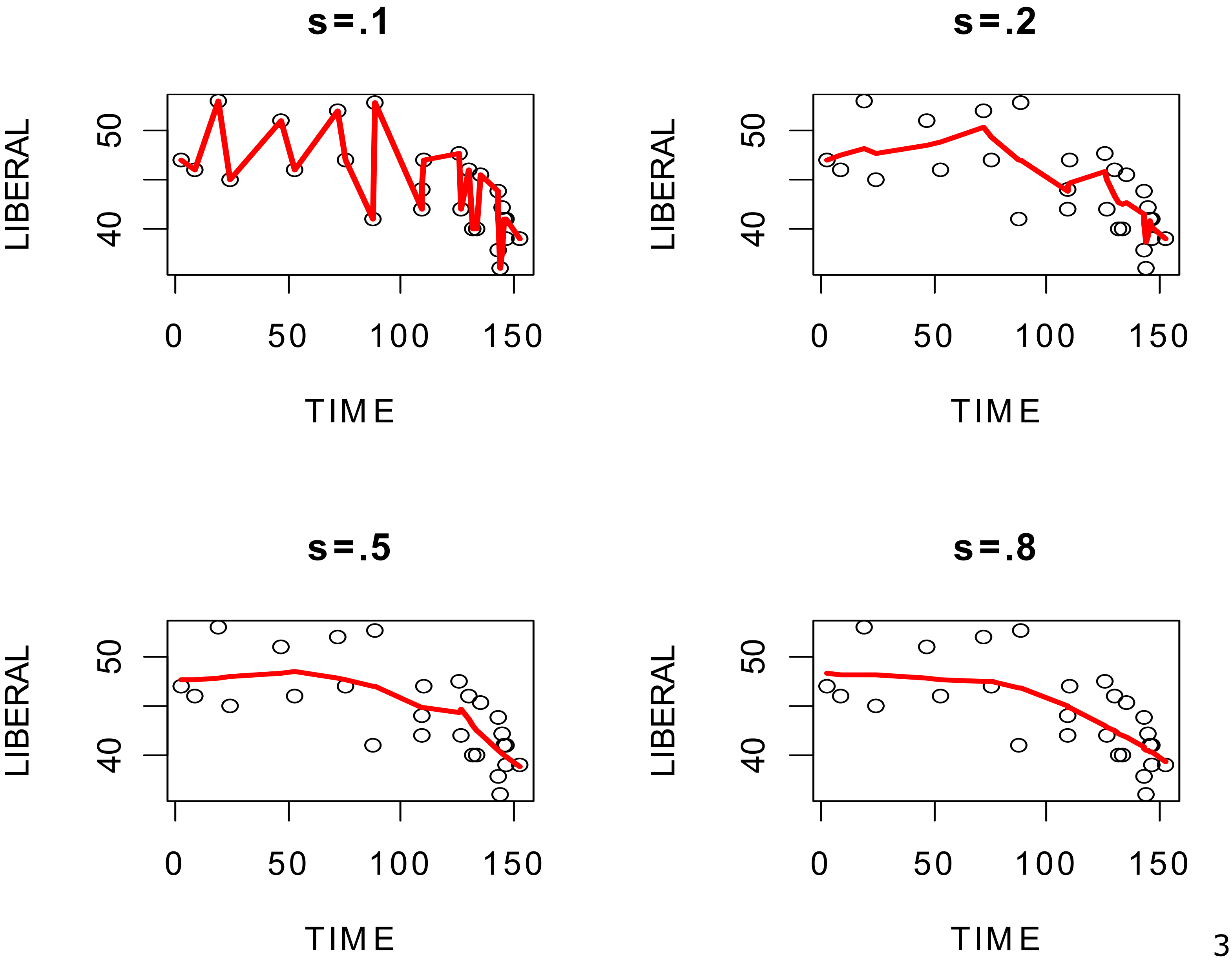
Type of Span (1)
Constant bandwidth

- $h(\mathrm{x})=h$ for all values of X
- In other words, ***a constant range of X*** is used to find the observations for the local regression for each X-value
- This works satisfactorily if the distribution of X is uniform and/or with large sample sizes
- It fails, however, if X has a non-uniform distribution
  - It can fail to capture the true trend simply because of data limitations—some local neighbourhoods may have none or too few cases
  - This is particularly problematic at the boundary regions or in more than one dimension
- Because of these limitations, lowess models typically avoid this type of bandwidth

Type of Span (2)
Nearest neighbor bandwidth

- The ***nearest neighbourhood method overcomes the sparse data problem***
- S is chosen so that each local neighbourhood around the focal x always contains a specified proportion of observations
  - Typically this is done visually by trial and error—*i.e.*, we fit the model and inspect the curve, changing the span until we have removed most of the roughness in the curve
  - Loess residuals can provide more systematic guidance
  - The span will usually fall between .3 and .8
- We want the ***smallest span that provides a smooth fit***
  - The default span for `loess` is S=.75; for `lowess` it is f=2/3

The effects of different Spans



s=.1

s=.2

s=.5

s=.8

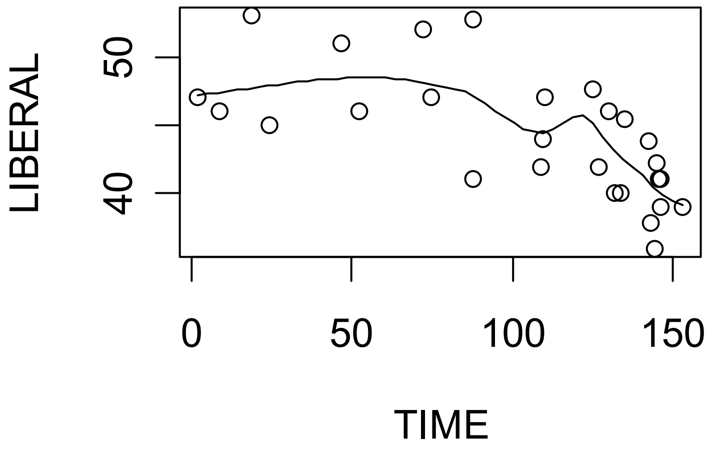
R-script for graphs showing Effects of Bandwidth

```
split.screen(figs=c(2,2))
screen(1)
plot(TIME, LIBERAL, main="s=.1")
lines(lowess(TIME, LIBERAL, f=.1), lwd=3, col="red")
screen(2)
plot(TIME, LIBERAL, main="s=.2")
lines(lowess(TIME, LIBERAL, f=.2, iter=0), lwd=3, col="red")
screen(3)
plot(TIME, LIBERAL, main="s=.5")
lines(lowess(TIME, LIBERAL, f=.5, iter=0), lwd=3,col="red")
screen(4)
plot(TIME, LIBERAL, main="s=.8")
lines(lowess(TIME, LIBERAL, f=.8, iter=0), lwd=3, col="red")
close.screen(all = TRUE)
```

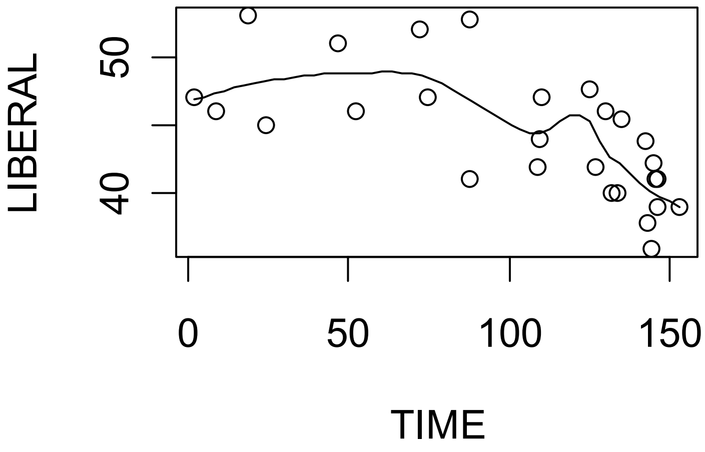
Local Polynomial Degree

- The degree of the polynomial also affects the bias-variance trade-off
  - A higher degree polynomial will provide a better approximation of the underlying mean than a lower polynomial degree—*i.e.*, ***a higher degree polynomial will have less bias***
  - Higher degree polynomials also have more coefficients to estimate, however, resulting in ***higher variability***
- It is usually most effective to ***choose a low degree polynomial and concentrate instead on choosing the best bandwidth***
  - The most commonly used polynomials are ***local linear*** and ***local quadratic***—the local linear has more bias, but has less variance, especially at the boundaries
  - Cubic and higher older polynomials tend not to improve the fit by much

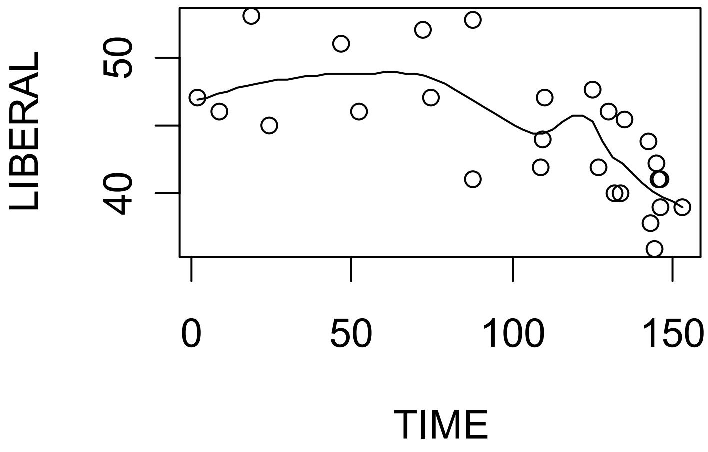
Local linear



Local quadratic



Local cubic



R-script for effect of Polynomial Degree

```
split.screen(figs=c(2,2))
screen(1)
library(modreg)
scatter.smooth(TIME, LIBERAL, span = .4,
      degree = 1, main="Local linear")
screen(2)
scatter.smooth(TIME, LIBERAL, span = .4,
      degree = 2, main="Local quadratic")
screen(3)
scatter.smooth(TIME, LIBERAL, span = .4,
      degree = 3, main="Local cubic")
close.screen(all = TRUE)
```

Weight Function

- The choice of the weight function has much less effect on the bias-variance trade-off than other elements of the nonparametric specification, but it can affect the visual quality of the fitted regression curve
- Although there is no restriction on the particular weight function that is used, it is desirable to use a smooth and continuous weight
- The most commonly used weight function for nonparametric regression models is the ***tricube weight function***
  - In fact, all of the loess functions in **R** use this weight function, and it cannot be altered easily. I see no reason why you would want to change this.

Statistical Inference and Degrees of Freedom

- The concept of degrees of freedom for nonparametric models is not as intuitive as for linear models since there are no parameters estimated
- Nonetheless, the degrees of freedom for a nonparametric model are a ***generalization of the number of parameters in a parametric model***
  - The analogy to the linear model is not perfect, but approximate
- Using the *approximate degrees of freedom*, we can carry out F-tests to compare different estimates applied to the same dataset:
  - Compare different levels of polynomial fits; Compare the smoothed model to a linear model *etc.*
- Determining the degrees of freedom is also necessary for constructing confidence envelopes around the fitted curve

Statistical Inference and Degrees of Freedom (2)

- The degrees of freedom in parametric regression, can be defined in several ways—the first is that the number of predictors, k (including the constant)
- Recall that the hat matrix $\mathbf{H}$ transforms Y into Y-hat. From this matrix we can also determine the degrees of freedom, where they are equal to:
  - the rank and trace (i.e., the sum of the diagonals) of the hat matrix $\mathbf{H}$
  - the trace of $\mathbf{HH}$▫
  - the trace of 2$\mathbf{H}$-$\mathbf{HH}$▫
- These alternative expressions follow from the fact that H is symmetric and idempotent—*i.e.*, $\mathbf{H}$=$\mathbf{H}$▫

Statistical Inference and Degrees of Freedom (3)

- Analogous degrees of freedom for nonparametric models are obtained by substituting the smoother matrix **S**, which plays a similar role to the hat matrix **H**—*i.e.*, it transforms Y into Y-hat
- We can then see the *approximate degrees of freedom* as defined in several equivalent ways:
  - $\mathrm{df}_{\mathrm{MOD}}=\mathrm{trace}(\mathbf{S})$
  - $\mathrm{df}_{\mathrm{MOD}}=\mathrm{trace}(\mathbf{SS}$▫$)$
  - $\mathrm{df}_{\mathrm{MOD}}=\mathrm{trace}(2\mathbf{S}\text{-} \mathbf{SS}$▫$)$
- The residual degrees of freedom is then $df_{\mathrm{RES}}=n\text{-}df_{\mathrm{MOD}}$, and the ***estimated error variance*** is $S^2=\sum e^2{}_i/df_{\mathrm{RES}}$
- Unlike in linear model, the degrees of freedom for nonparametric regression are not necessarily whole numbers

Confidence Envelope for the Regression Curve

- The estimate variance of the fitted value Y-hat at X=x_i is

$$\widehat{V}(\widehat{y}_i) = S^2 \sum_{j=1}^{n} w_{ij}^2$$

where S^2 is the estimated error variance and w_{ij} are the weights applied to each x

- Assuming normally distributed errors, an approximate 95% confidence interval the population regression $\mu|x_i$ is

$$\widehat{y}_i \pm 2\sqrt{\widehat{V}(\widehat{y}_i)}$$

- We then simply join the confidence intervals for each of the X-values together to plot the ***95% confidence band or envelope*** for the regression function
- Alternatively, bootstrap standard errors can be used

Hypothesis Tests

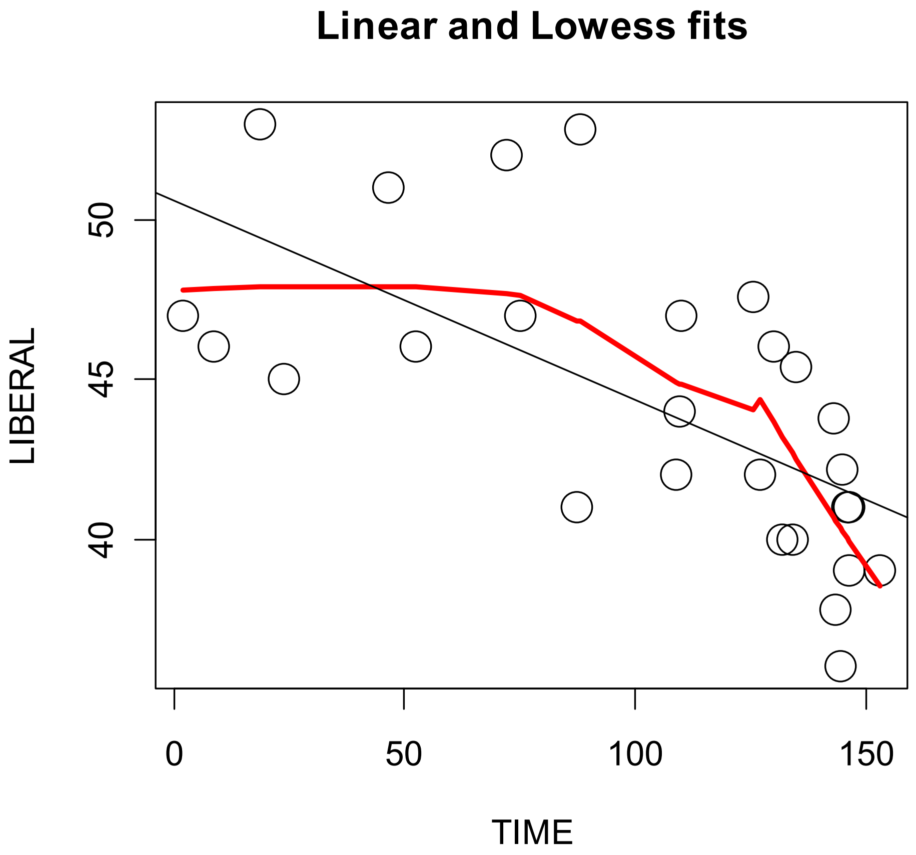
- F-tests comparing the residual sums of squares for alternative nested models can be carried out in exactly the same way as for linear models—these tests are only approximate because of the approximation of the *df*
- Perhaps the most useful aspect of nonparametric regression is that it allows us to test for nonlinearity by contrasting it with a linear regression model
  - These models are nested because a linear model is a special case of a general, nonlinear, relationship
  - The F-test takes the usual form:

$$F_0 = \frac{\frac{\text{RSS}_0 - \text{RSS}_1}{\text{trace}(\text{S}) - 2}}{\frac{\text{RSS}_1}{n - \text{trace}(\text{S})}}$$

Where RSS_0 is the residual sum of squares from the linear model; RSS_1 is from the nonparametric model; trace(S) is the *df* for the nonparametric model

Comparing the OLS and Lowess fits

- The red line is the OLS fit; the black the lowess smooth from a local linear regression with a span=.6
- There is a clear departure from linearity in these data—the linear linear does not seem to fit well
- An incremental F-test allows us to test for nonlinearity

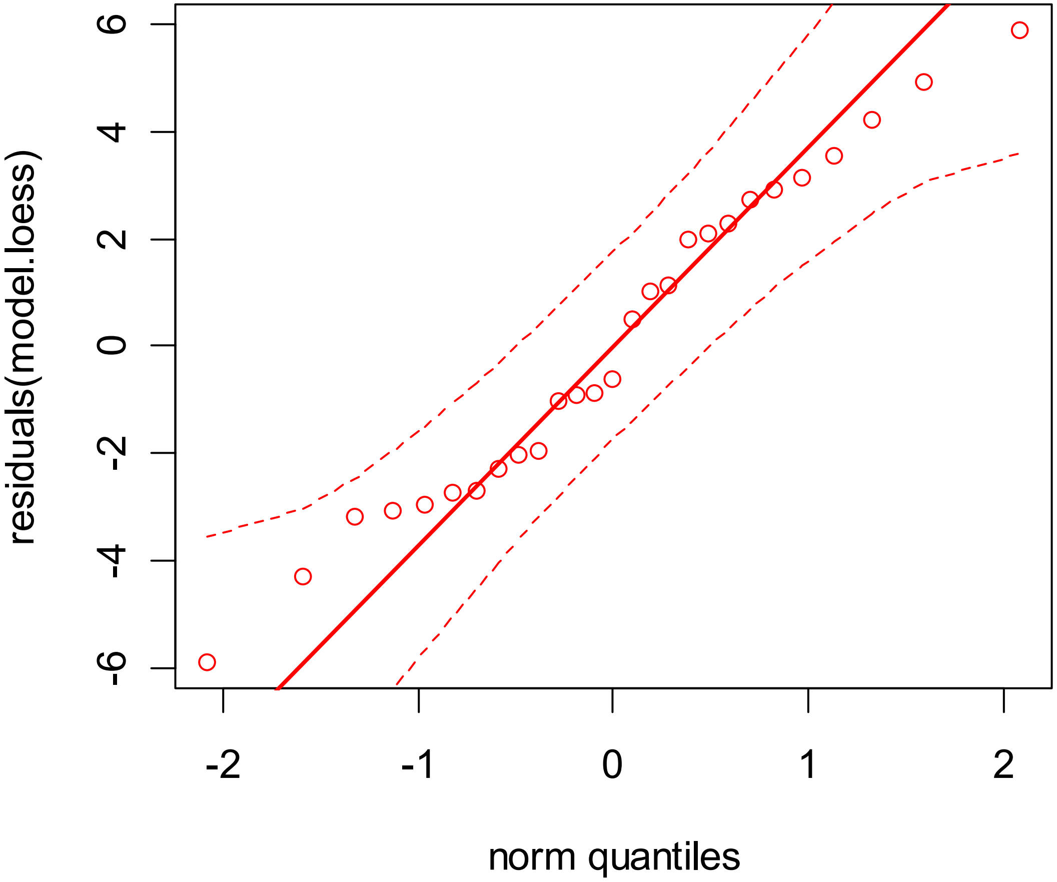


```
plot(TIME,LIBERAL, main="Linear and Lowess fits", cex=2)
lines(lowess(TIME, LIBERAL, f=.6), lwd=3, col="red")
abline(lm(LIBERAL~TIME))
```

Diagnostics

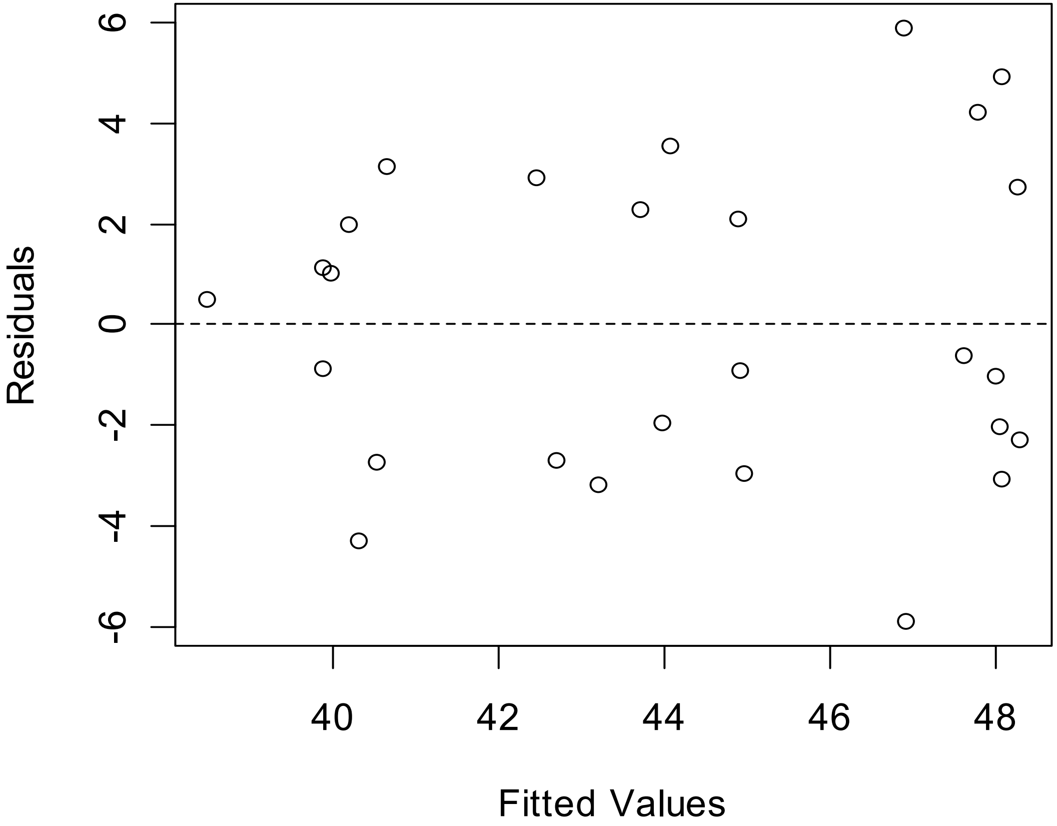
- Best fit in nonparametric regression depends on not only variable selection but also the choice of ***smoothing parameter***
- As with the linear model, the most important diagnostic component is the residuals
- Several plots can be useful:
  - ***Residuals vs predictor values*** (useful for detecting lack of fit, such as trimmed peaks, and nonconstant error variance)
  - ***Quantile comparison plots of the residuals*** (to detect departures from normality in the residual distribution)
  - ***Absolute residuals vs predictors*** (to detect dependence of the residuals on predictors)
- In **R**, these plots do not have a named function, but they are easily constructed

Quantile Comparison Plot of Residuals: Looking for outliers



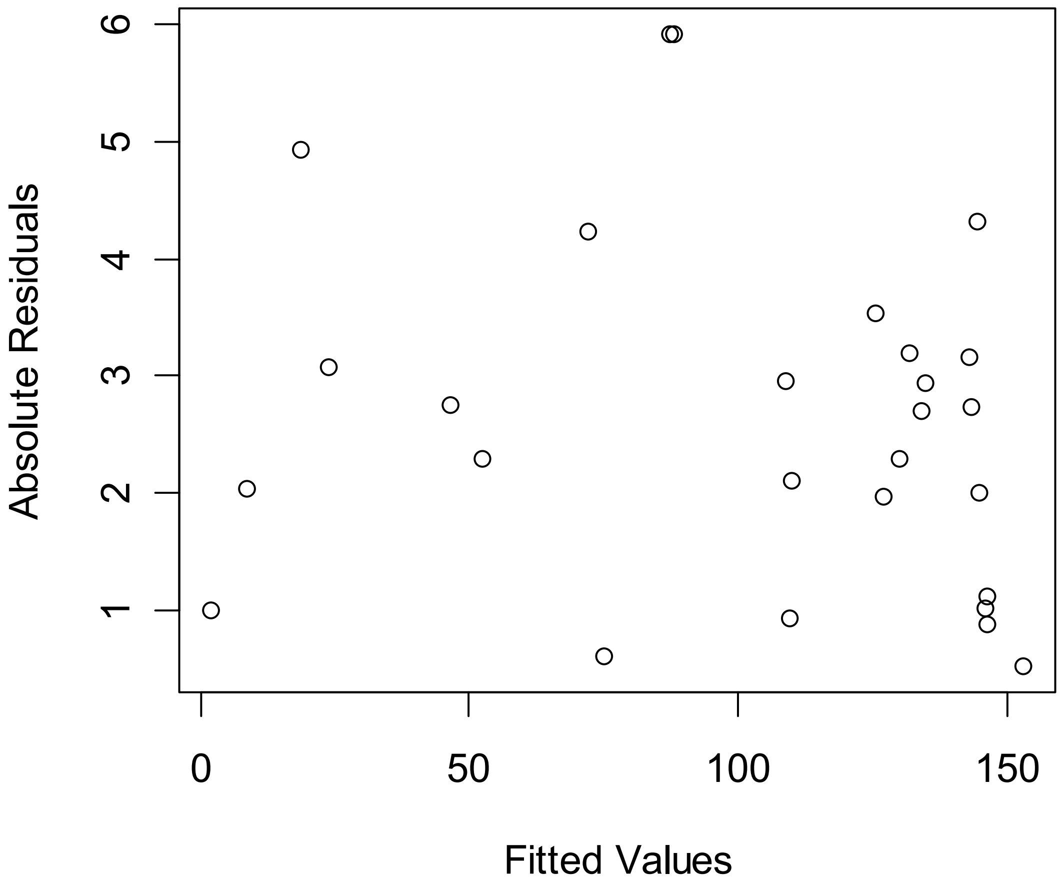
```
> library(car)
> qq.plot(residuals(model.loess))
```

Residual Plot: Looking for nonconstant error variance and poor fit



```
> plot(model.loess$fit, residuals(model.loess),
    xlab="Fitted Values", ylab="Residuals")
> abline(h=0,lty=2)
```

Absolute Residuals versus Predictors



```
plot(model.loess$x, abs(residuals(model.loess)),
    xlab="Fitted Values", ylab="Absolute Residuals")
```