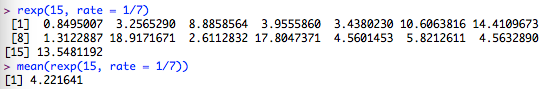
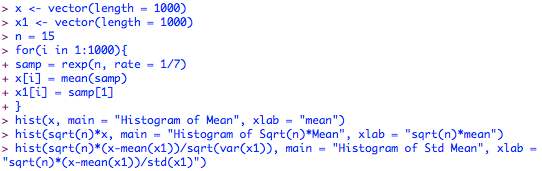
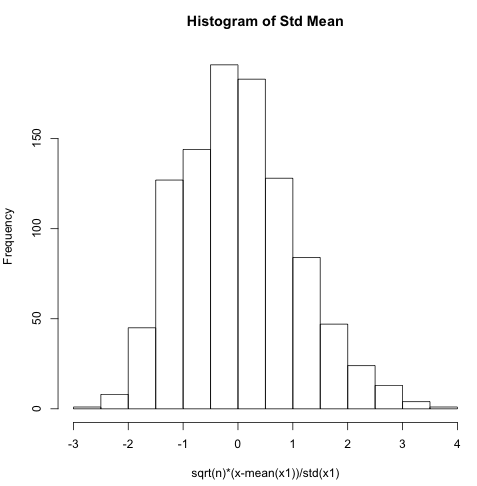
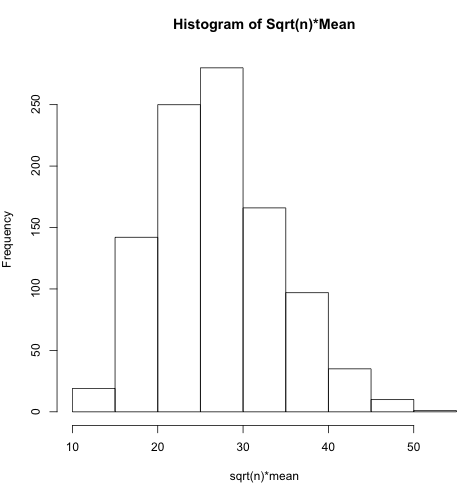
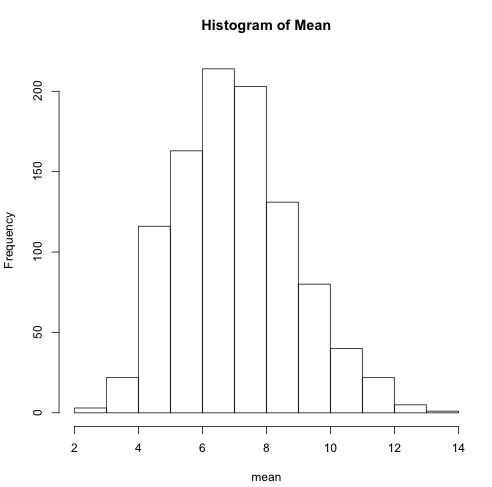
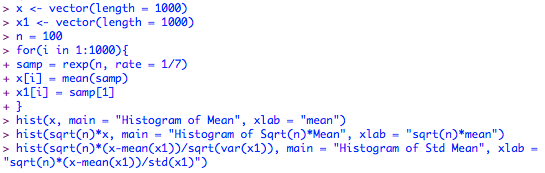
**Problem 1**

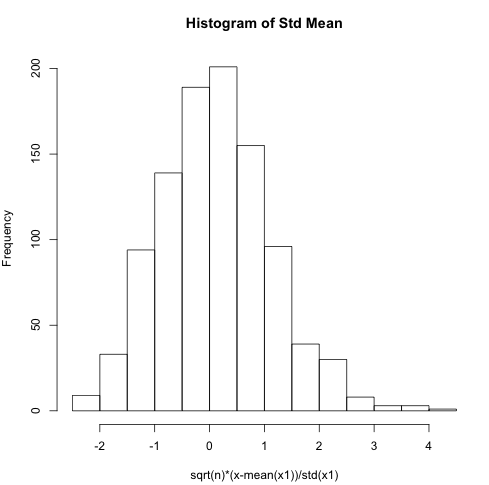
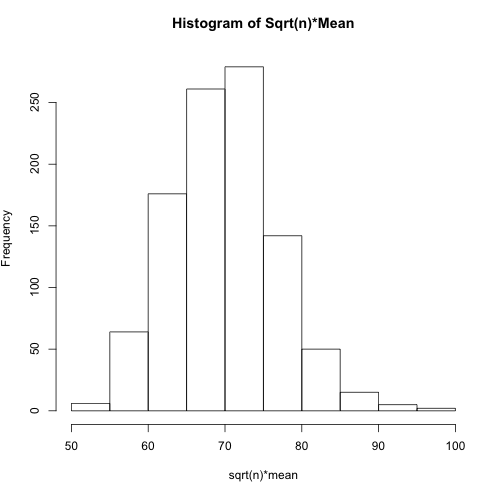
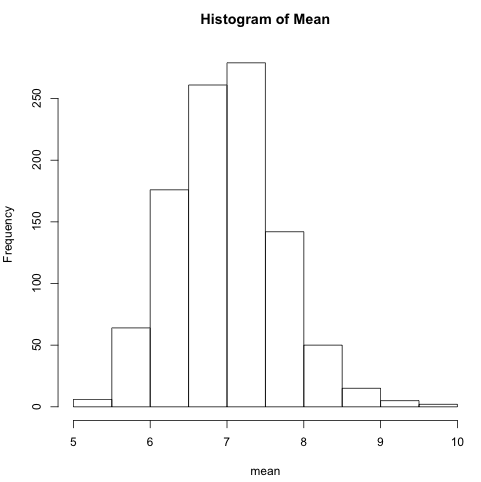
****

****

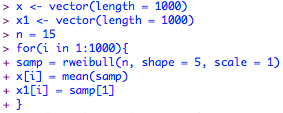


**Problem 2**

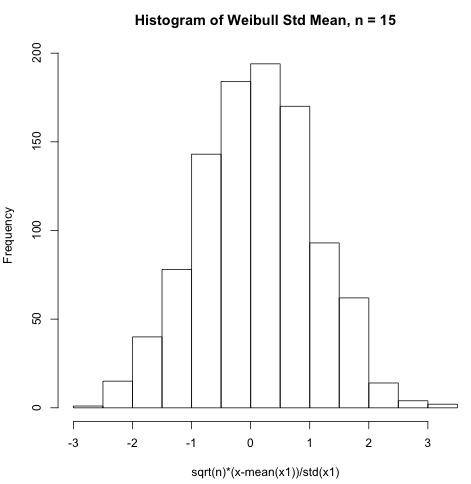
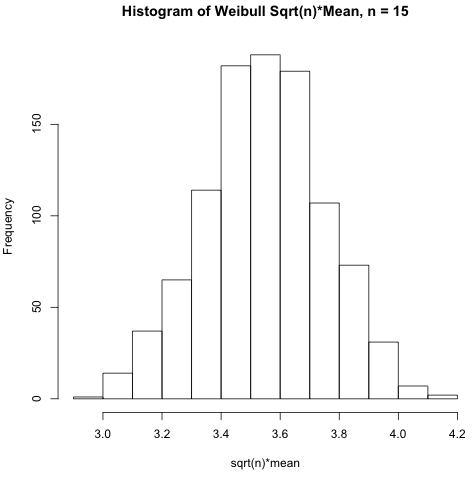
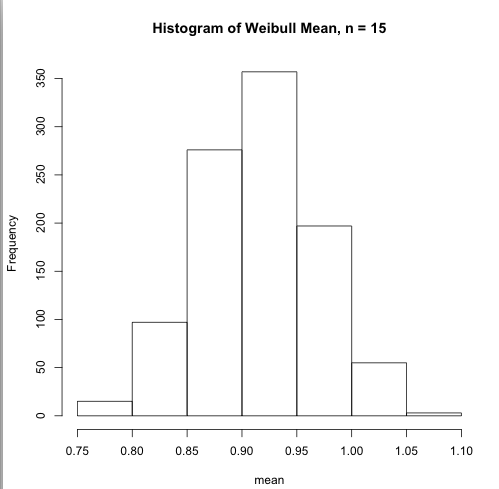




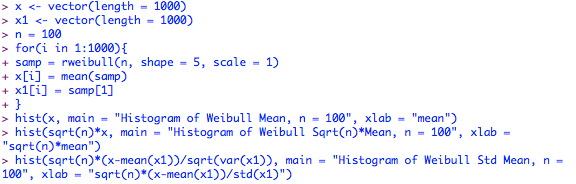
**Problem 3**

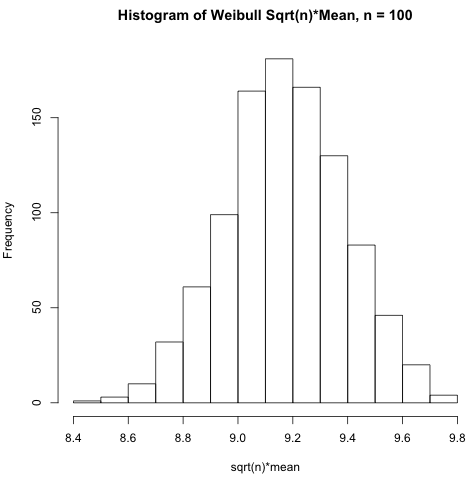
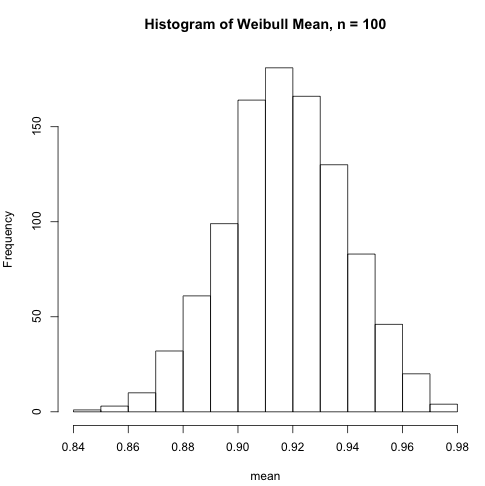


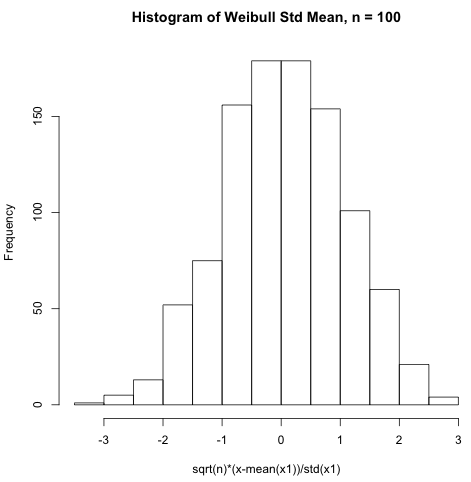
../../../../../../../../../Downloads/Screen%20Shot%202016-11-02%2

../../../../../../../../../Downloads/Screen%20Shot%202016-11-02%2

**Problem 4**







**Problem 5**

The example is illustrating how to use Gaussian Quadrature in R to approximate the mean of a Gamma Distribution with and .

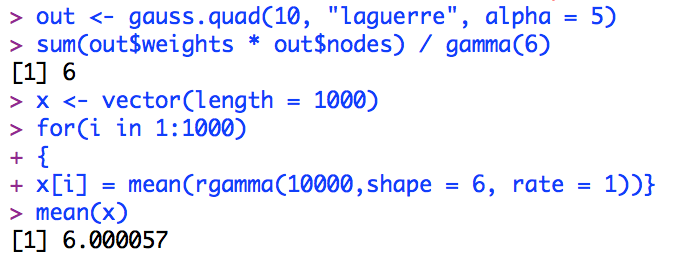
We know that to calculate exact mean of \Gamma Distribution, we have:

We also know that Gauss-Laguerre Quadrature approximate integrals with formula:

With , this becomes

where and are nodes and weights for Gaussian Quadrature, and therefore we can easily evaluate the mean of Gamma Distribution by dividing the sum of their products by .

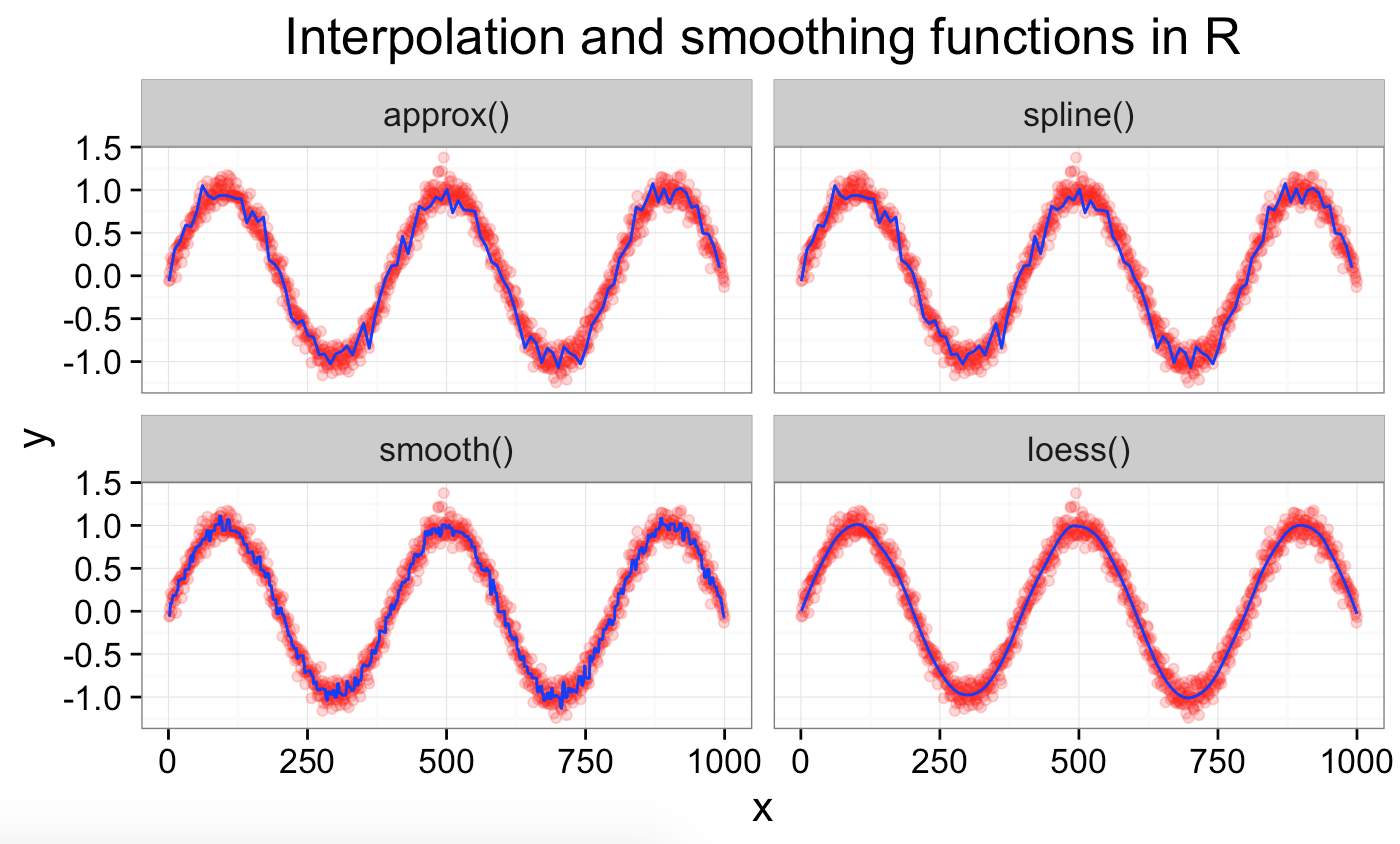
This particular choice of orthogonal polynomials is appropriate for this problem because as we learned in class, the quadrature rule is exact for all polynomials up to degree , while here has degree .



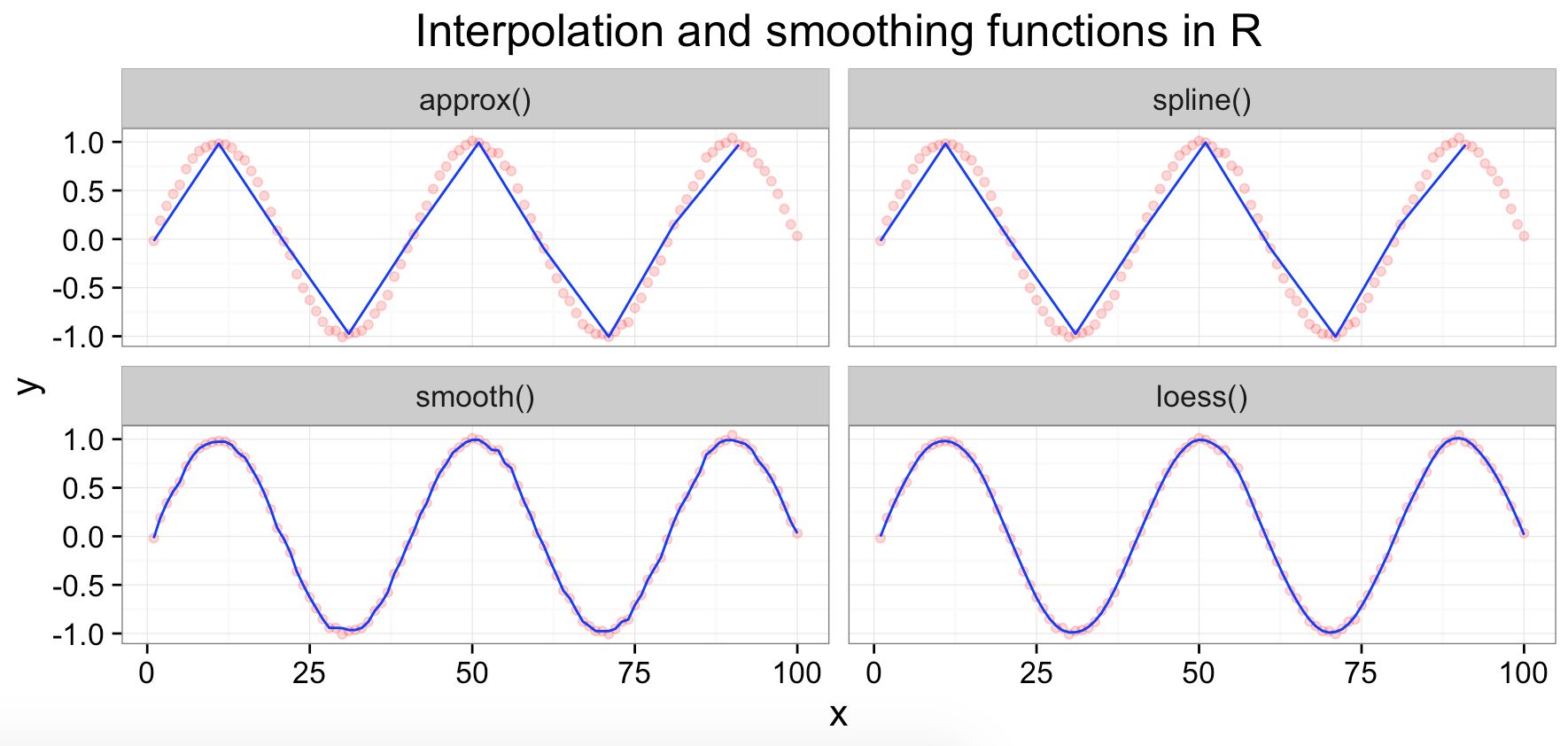
We can see that for mean of Gamma Distribution, the quadrature method gives a result of 6, as our sampling approximation gives a result extremely close to 6.

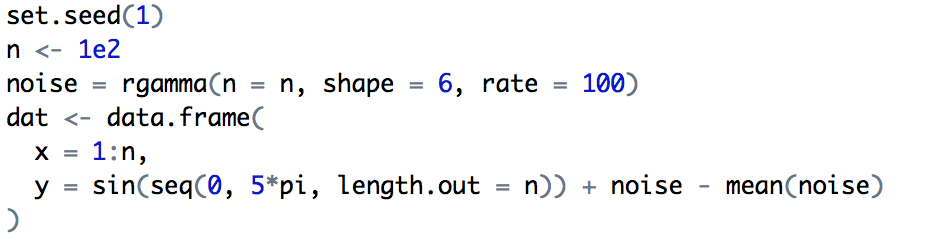
**Problem 6**

Here is the original plot:



1. Changing n to 1e2 and the random noise term to a Gamma Distribution with and so that variance is small, we then subtract the mean of this distribution to re-center noise around 0. This way, we have the plot:



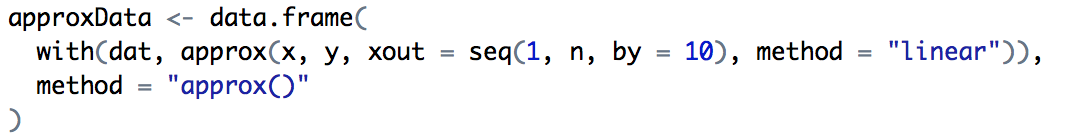


set.seed is used for possible desire for reproducible results of random variable generation.

Generate 100 random variables with Gamma Distribution with and as “noise.”

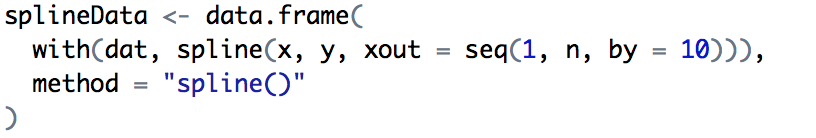
data.frame is used for creating a data frame, a list of variables of the same number of rows.

Generate “dat” via seq, a function used for generating sequence by specifying from = 0, to = 5\*pi, length.out as the desired length of sequence. We add in noise but subtract its mean to make it center around 0.

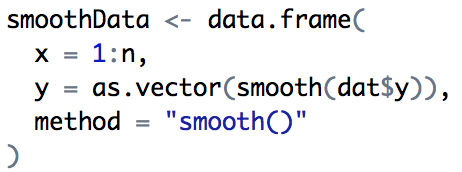


with() applies an expression to a dataset, similar to DATA= in SAS.

approx() return a list of points that linearly interpolate given data points; *xout* is the set of numeric values specifying where interpolation happens (here is by every 10 unit); *method* specifies which interpolation method to use, which is “linear” here; *rule* describes how interpolation is to take place outside the interval [min(x), max(x)], which is by default as 1 here that means returning NAS for such points; *f* indicates a compromise between left- and right- continuous step functions, which is by default as 0 here; *ties* handles the tied x values, which by default return the mean here.

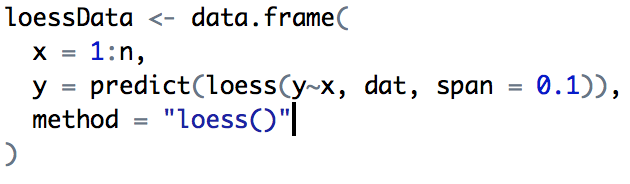


spline() uses cubic spline interpolation. By default, method = “fmm”, and the spline used is that of Forsythe, Malcolm and Moler (an exact cubic is fitted through the four points at each end of the data, and this is used to determine the end conditions). Other options are “natural” and “periodic.”



as.vector is a generic attempt to coerce it argument into a vector of *mode*, whose default is “any,” which means to coerce to whichever vector mode is most convenient.

smooth() utilizes Tukey’s Smoothing method which allows the predicative line to wiggle at points. *kind* is a character string indicating kind of smoother required, which by default is “3RS3R”, a concatenation of 3R, S and 3R, 3RSS similarly. *twiceit* indicates if the result should be “twiced”, which by default is FALSE. endrule is a character string indicating the rule for smoothing at boundary, which is by default “Tukey.” *do.ends* indicates if the 3-splitting of ties should also happen at boundaries, which by default is FALSE, and is only used for *kind* = “S”.



predict() is a generic function for predictions from the results of various model fitting functions.

loess(formula, data, weights, subset, na.action, model = FALSE,

span = 0.75, enp.target, degree = 2,

parametric = FALSE, drop.square = FALSE, normalize = TRUE,

family = c("gaussian", "symmetric"),

method = c("loess", "model.frame"),

control = loess.control(...), ...)

fits a polynomial surface determined by one or more numerical predictors, using local fitting.

*span* is the parameter which controls degree of smoothing.

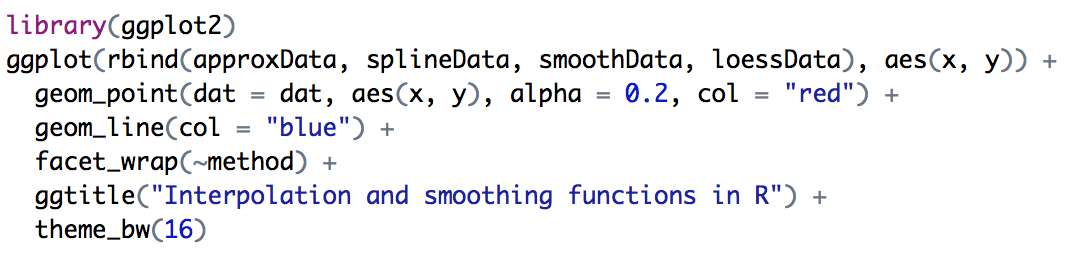
*degree* is the degree of polynomials to be used, which by default is 2.

*parametric*means should any terms be fitted globally rather than locally, which by default is FALSE.

*drop.square* for fits with more than one predictor and degree = 2, should the quadratic term be dropped for particular predictors?

*normalize* means should the predictors be normalized to a common scale if there is more than one, which by default is TRUE.

*family* determines if "gaussian" fitting is by least-squares, and if "symmetric" a re-descending M estimator is used with Tukey's biweight function.



Call package ggplot2 in library and plot the 4 graphs with different data sets. Mostly formatting functions here.