'monoProc'- Version 1.0-4
Strictly monotone and smooth regression estimation in R

Regine Scheder

October 6, 2005

1 Introduction

The R-package 'monoProc' is an implementation of the monotonization procedure introduced by Dette, Pilz, and Neumeyer (2005) and Dette, Scheder (2005). The goal of this software package is to provide a smooth monotonizing procedure which can be used for any smooth regression estimate with one or two independent variables. This goal is not easily to reach, but so far there are four classes of regression estimates which can be directly monotonized by applying 'monoproc'. Besides that, a somehow default-method to monotonize is available where the input is a 'list'. In the second section, the monotonizing procedure is briefly introduced within a nonparametric setting. The third section deals with some changes to S4-classes. The usage of the package 'monoProc' and the class 'monoproc' are presented in the fourth and fifth section, and last but not least some issues and problems of this implementation are discussed.
'monoProc' can be obtained via the WWW at http://homepage.ruhr-uni-bochum.de/Regine.Scheder/work.html.

2 The monotonizing procedure

The general setting of the R-package 'monoProc' can be described as the problem of estimating a regression function under monotonicty constraints in a nonparametric regression model, that is

$$Z = m(T) + \sigma(T)\varepsilon,$$

where m is a smooth monotone function, σ a smooth variance function without further constraints, ε denotes a random error, and T is the observed value of independent variables [more than one variable is possible]. The monotonizing procedure applies an unconstrained nonparametric estimate

of the regression function as preliminary estimate $\hat{m}$, e.g. the Nadaraya and Watson estimate or local polynomial estimates. In the following, the two steps of the monotonizing procedure are described.

- **Step 1** (isotonization)
  Define
  $$\hat{m}_I^{-1}(z) = \frac{1}{Nh_d} \sum_{i=1}^{N} \int_{-\infty}^{z} K_d \left(\frac{\hat{m}(\frac{i}{N}) - u}{h_d} \right) du, \tag{1}$$
  where h_d is a bandwidth with $h_d \to 0$ for a positive two times continuously differentiable, symmetric kernel K_d with compact support on [-1,1]. If N is sufficiently large, then $\hat{m}_I^{-1}(z)$ is strictly monotone increasing and a good approximation for
  $$\frac{1}{h_d} \int_0^1 \int_{-\infty}^{z} K_d \left(\frac{\hat{m}(x) - u}{h_d} \right) du dx.$$
  This last expression is a smooth version of
  $$\int_0^1 I\{\hat{m}(x) \leq z\} dx$$
  which is the fundamental motivation of this procedure, since this is for a monotone function $\hat{m}$ the inverse, i.e. $\hat{m}^{-1}$. To obtain a strictly monotone decreasing estimate, we define instead of (1)
  $$\hat{m}_A^{-1}(z) = \frac{1}{Nh_d} \sum_{i=1}^{N} \int_{z}^{\infty} K_d \left(\frac{\hat{m}(\frac{i}{N}) - u}{h_d} \right) du. \tag{2}$$

- **Step 2** (inversion)
  The inverse of $\hat{m}_I^{-1}(z)$ and $\hat{m}_A^{-1}(z)$, respectively, is calculated. This function is either strictly increasing or strictly decreasing.

It is worth mentioning that this procedure is not restricted to a nonparametric setting and can be applied in the same way in a parametric setup. Furthermore, this procedure can be used for any smooth function. An extension to multivariate regression setting is possible too.

3 S4-Classes for 'ksmooth' and 'locpoly'

To simplify the use of the function 'monoproc', two functions for nonparametric regression estimation are changed to functions with S4-Class values. Basically, both function, 'ksmooth' from the 'stats' package and 'locpoly' from 'KernSmooth' package, can be used in their original form, since their

values are of class 'list'. The new function 'ksmooth' has as value an object of class 'ksmooth' containing three components the x- and the y-values of the smoothed fit and the call of this object. The function and the class 'locpoly' are constructed correspondingly. In 'monoProc_1.0-4', the match.call function uses the original functions as argument that the variables can be accessed by their name. This is a first step to change these important standard functions into the S4-class system. But this means also that the existence of these two functions in the package 'monoProc' are considered as temporarily to allow things like changing the gridsize in 'monoproc' without fitting explicitly the unconstraint estimator again.

4 How to use 'monoProc'

Some important details about the use of this R-package are introduced in this section. So far, the 'monoProc'-Package contains the monotonizing function 'monoproc', a cross-validation function 'cv' (implementation only for monotonized 'locfit'-objects), several S4-classes, and two intrinsic functions 'mono.1d' and 'mono.2d', basically the default functions, which are called within 'monoproc'.
First of all, the value of the function 'monoproc' is an object of class 'monoproc' with some further specification depending on the original regression fit and the dimensionality of the problem. It is distinguished between an one-dimensional and a two-dimensional regression problem (i.e. the dimension of the independent variable). For an one-dimensional problem, the function 'monoproc' gives an object of class 'monoproc.1d' back and in the two-dimensional case an object of class 'monoproc.2d'. These two classes extend the class 'monoproc' and are implemented to provide different 'plot' and 'summary' methods. The classes 'monoproclocfit.1d' and 'monoproclocfit.2d' are 'monoproc'-objects which are monotonized 'locfit'-objects. This two additional classes are brought in because there is a cross validation function available for these two classes. The 'locfit' package provides a function to calculate the cross validation scores which is also used implicitly to evaluate the cross validation function for the monotone fit. In general, the intention is to have this cross validation function 'cv' for all monotone fits, but, currently, this is only existing for 'monoproclocfit.1d' and 'monoproclocfit.2d', respectively.
The basic formals of 'monoproc' are 'fit', 'bandwidth', 'xx', and 'dir' which define the signature of 'monoproc'. Some of them can be missing, e.g. 'dir' refers to a twodimensional problem and is therefore only needed in these cases (otherwise an error occurs). The variable 'xx' determines where the function 'monoproc' is evaluated, if missing the independent variables of 'fit' are used instead. There are three more variables, 'mono1', 'mono2', and 'gridsize'. Their default values are "increasing" for 'mono1' and 'mono2',

and 40 for the gridsize (but for lists the length of the response is used).
In order to illustrate the usage of 'monoProc', some detailed examples are discussed. The cars data in the datasets package is often used a standard example for regression estimates. Therefore, we will apply the monotonizing procedure to this dataset with several preliminary regression estimates. The aim is to demonstrate the features of 'monoProc' not to find the best regression fit for the cars data.

```
> library(monoProc)
> data(cars)
> speed <- cars$speed
> dist <- cars$dist
```

As a first fit for the cars data the Nadaraya-Watson estimate is calculated with the R-function 'ksmooth' and then 'monoproc' is applied. There exists another function in R which fits a monotone regression function, but this function computes a piecewise constant regression fit. These two monotone functions are compared with each other in Figure 1.

```
> fit1 <- ksmooth(speed, dist, "normal", bandwidth = 2.5)
> fit2 <- monoproc(fit1, bandwidth = 0.7)
> fit3 <- isoreg(speed, dist)
```

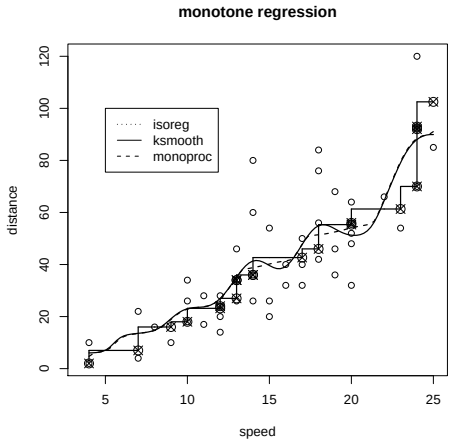

Figure 1: A comparison with the R-function isoreg.

The function 'loess' from the 'stats'-package can be used as well as a preliminary estimate in 'monoproc'. To change the evaluated points in 'monoproc', use the variable 'xx'. In this example, the use of 'monoproc' does not make sense since the loess fit is already monotone but the 'goodness' of

the monotone approximation with a bandwdith 0.4 and a gridsize of 30 is demonstrated.

```
> cars.lo <- loess(dist ~ speed, cars, degree = 2,
+     control = loess.control(surface = "direct"))
> predict <- predict(cars.lo, data.frame(speed = seq(5,
+     25, 1)))
> monofit <- monoproc(cars.lo, bandwidth = 0.4, xx = seq(5,
+     25, 1), mono1 = "increasing", gridsize = 30)
```

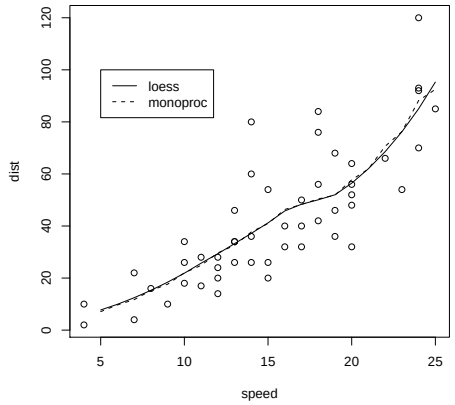

Figure 2: Plot of a loess object and a its corresponding monoproc fit.

Although 'monoproc' is not explicitly implemented for a parametric regression function, by transforming a polynomial regression fit into a 'list'-object, 'monoproc' can be used. It is recommended to use a equidistant design for the x-variables. In the following, it is shown how this works exemplified again through the cars data. First of all a polynomial regression of degree 4 is fitted, and a 'list' is created by using the function 'predict' with the equidistant vector d. The length of the vector d corresponds in this case to the gridsize N.

```
> d <- seq(0, 25, len = 200)
> pr <- lm(dist ~ poly(speed, 4))
> prlist <- list(speed = d, dist = predict(pr, data.frame(speed = d)))
> mpr <- monoproc(prlist, bandwidth = 0.05, mono1 = "increasing")
```

In the next example, the usage in a two-dimensional regression problem is presented. The Fat dataset in the package 'UsingR' contains two bodyfat measures obtained by underwater weighing and several body measurements (weight, height, and body circumfence measurements). Two outliers in this

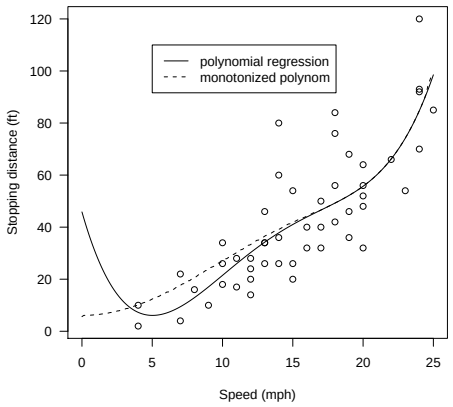

Figure 3: monoproc in a parametric setting.

dataset are removed for this analysis. Motivated by the Body Mass Index, we consider the simple regression model

$$Y = m(X_1, X_2) + \epsilon,$$

where

$$\begin{aligned} Y &= \text{body.fat.siri (\%)}, \\ X_1 &= \text{weight (lbs)}, \\ X_2 &= \text{height (inch)}, \end{aligned}$$

and m is a strictly monotone increasing in X_1 and strictly monotone decreasing in X_2. But first a local polynomial estimate without constraints is fitted using 'locfit.raw' from the 'locfit'-package.

```
> library(UsingR)
> library(locfit)
> data(fat)
> fat <- fat[-39, ]
> fat <- fat[-41, ]
> attach(fat)
> fit <- locfit.raw(cbind(weight, height), body.fat.siri,
+     alpha = 0.3, deg = 1, kern = "epan")
> fitmono <- monoproc(fit, bandwidth = 1, mono1 = "increasing",
+     mono2 = "decreasing", dir = "xy", gridsize = 30)
```

With the variable 'dir', the direction, the order of the monotonization, and the variables themselves (by choosing "x" or "y" only the first or rather the

second variable is monotonized) are determined. The gridsize refers to N in (1) and (2), respectively, that means for the actual calculation of $\hat{m}_I$, where both variables are montonized, N^2 points are used for the evaluation. It is therefore not recommended to use values for the gridsize bigger than 50.

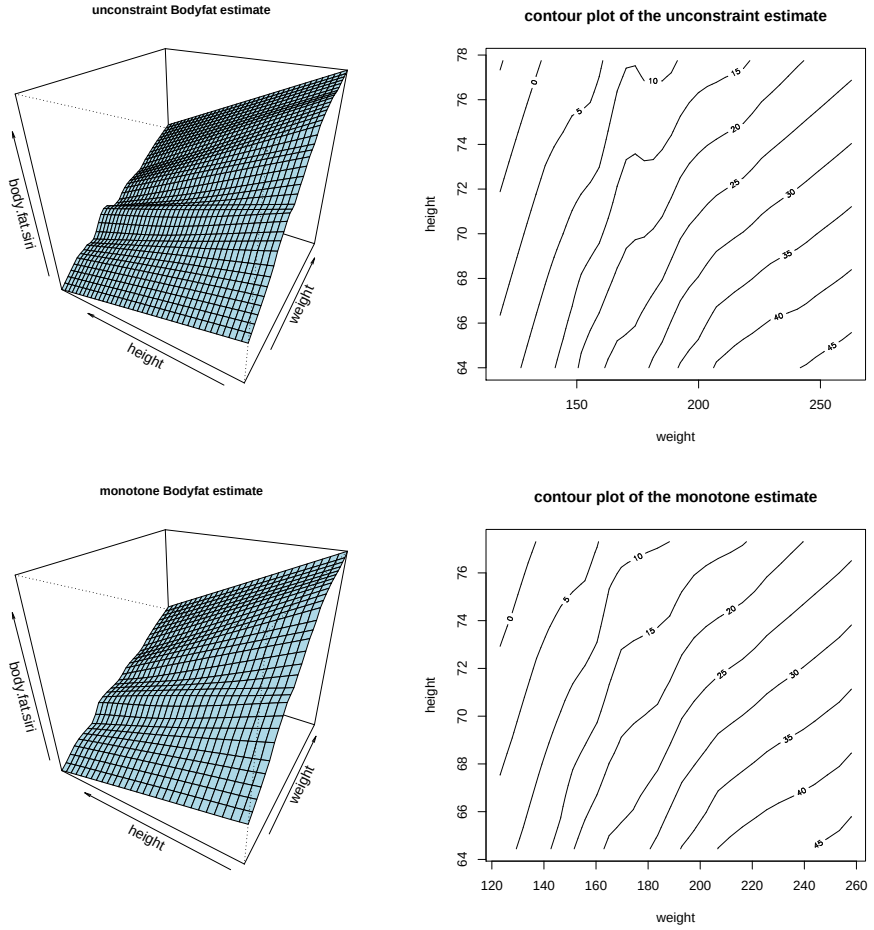

Figure 4: Bodyfat example: perspective and contour plots. Upper panel: the unconstraint regression estimate. Lower panel: the monotone estimate.

To compare the unconstraint and the monotized fit with each other, a cross validation function 'cv' for monotonized 'locfit'-objects is implemented. One-leave-out Cross Valdiation is used, i.e.

$$CV = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{m}_{I,i}(X_i))^2$$

where $\hat{m}_{I,i}(X_i)$ corresponds to the estimated value for Y_i after monotonization where the observation (X_i, Y_i) is left out. The value of the function 'cv'

is a matrix containing the values $\hat{m}_{I,i}(X_i)$ [the monotone estimate] in the first column and $\hat{m}(X_i)$ [the unconstrained estimate] in the second column. So applied to the Bodyfat-example, we obtain for the unconstraint estimator

```
> t <- cv(fitmono)
> sum((t[, 2] - body.fat.siri)^2)/250
```

```
[1] 31.45823
```

and for the monotone estimator

```
> sum((t[, 1] - body.fat.siri)^2)/250
```

```
[1] 31.6438
```

The Cross Validation function can also be used to choose the bandwidth of the monotonizing procedure.

The 'monoproc'-method for 'list'-objects is a kind of default-method. This method gives the flexibility to apply 'monoproc' to regular functions. In the following, the function

$$f(x,y) = \frac{1}{2}(y + \frac{sin(6\pi y)}{3\pi})(1 + (2x-1)^3)$$

is used as an example. This function is strictly monotone increasing in x but not in y. To monotonize this function with respect to y, a 'list'-object is created. In order to monotonize this function in y, the variable 'dir' is set to "y" and 'mono2' to "increasing". It is to remark that the x- and y-variable in list do not have to be of the same length, but z has to correspond to the values of x and y.

```
> x <- seq(1:50)/51
> y <- seq(1:70)/71
> z <- matrix(0, nrow = 50, ncol = 70)
> for (i in 1:70) {
+     for (j in 1:50) {
+         z[j, i] <- 0.5 * (y[i] + 1/(3 * pi) * sin(6 *
+             pi * y[i])) * (1 + (2 * x[j] - 1)^3)
+     }
+ }
> list <- list(x = x, y = y, z = z)
> mono <- monoproc(list, bandwidth = 9e-04, dir = "y",
+     mono2 = "increasing")
```

In Figure 6, two-dimensional plots are presented to show how the monotonizing procedure works. The commands for Figure 6 might be interesting to see how the fitted values can be accessed. Therefore all commands for the given figures can be found in the Appendix.

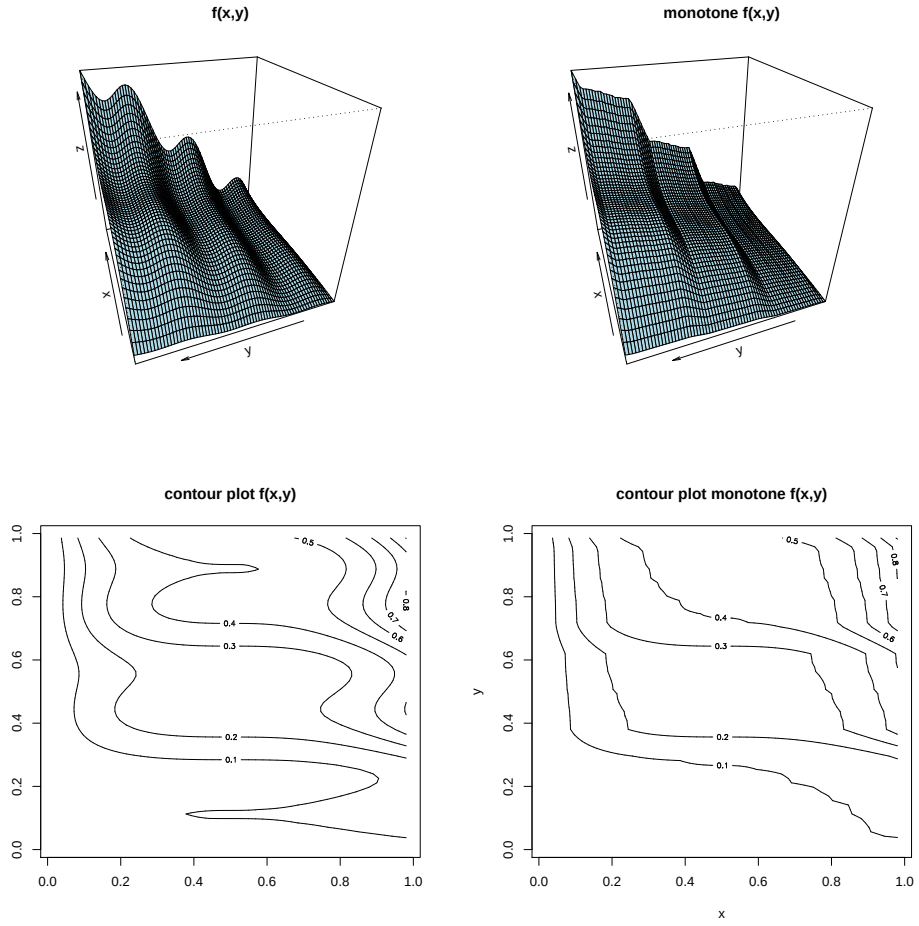

Figure 5: Monotonizing with respect to a single variable in a twodimensional problem.

5 The class 'monoproc'

To allow users more flexibility in building new methods for the class 'monoproc', we will discuss briefly some features of this class. First of all, the dependency between the 'monoproc'-classes can be illustrated by the following graph.

$$\text{monoproc} \Rightarrow \text{monoproc.1d}$$
$$\text{monoproc} \Rightarrow \text{monoproc.2d}$$

```
> slotNames("monoproc")

[1] "fit"       "fitold"    "gridsize"  "bandwidth" "kernel"
```

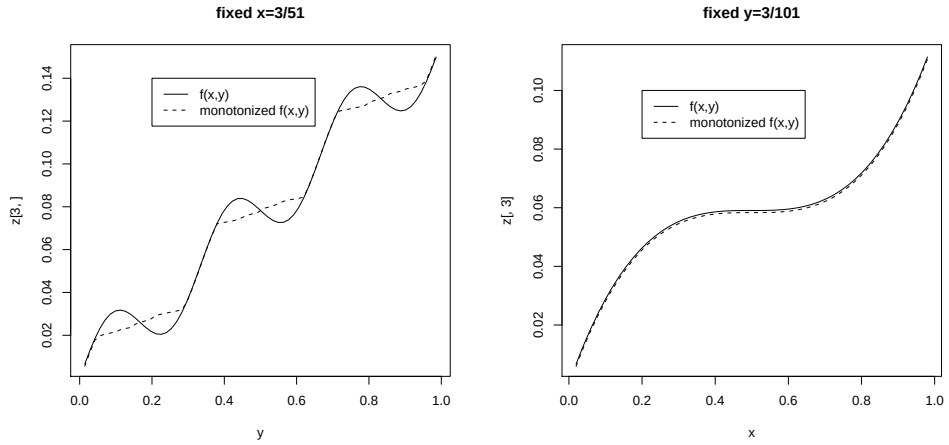

Figure 6: The function f and its monotonization at fixed $x = \frac{3}{51}$ (left) and at fixed $y = \frac{3}{101}$ (right).

```
[6] "mono"      "name"      "call"

> slotNames("monoproc.1d")

[1] "fit"       "fitold"    "gridsize"  "bandwidth" "kernel"
[6] "mono"      "name"      "call"

> slotNames("monoproc.2d")

[1] "dir"       "fit"       "fitold"    "gridsize"  "bandwidth"
[6] "kernel"    "mono"      "name"      "call"
```

The classes 'monoproc' and 'monoproc.1d' are basically the same. In the class 'monoproc.2d', the slot 'dir' is attached. The names for the slots are chosen self-explanatory and the slot 'call' can be easily used for further manipulations.

6 Problems to solve in the future

At the moment, especially the Cross-Validation function 'cv' might be computationally not efficient since for a two-dimensional problem this function calls another R function in which, furthermore, a C-Programm is called. But the general issue of this Package or of the function 'monoproc' is related to the fact that it should be able to be used with any smooth function coming from anywhere (i.e. 'locfit', 'ksmooth', or 'locpoly'). This is a real problem or challenge since all this 'objects' have very different properties, e.g. the slots for the fitted values have different name or are only available

after calling another function to evaluate or to predict. This makes the function 'monoproc' a bit messy. For this reason, the functions 'mono.1d' and 'mono.2d' are used to do the actual monotonization. Both functions have basically a list containing the independent variables (maximal two independent variables) and the response (same length as the variables) as argument. The length of the elements of this list determines the integral approximation described in section 2 where N corresponds to the length of the response. With this functions and also with the method 'monoproc' for 'list', it is easily possible to extend this monotonizing procedure to other objects. Further work might be therefore focused on the extension of 'monoproc' to other classes as arguments. Another emphasis is to make the cross validation function 'cv' available for all 'monoproc'-objects.
In 'monoProc_1.0-4', the signature of the function 'monoproc' contains the variables 'fit', 'bandwidth', 'xx', and 'dir'. This allows to match the method 'monoproc' in a more direct way (avoids some if-statements). Missing values for the bandwidth of the 'monoproc' function are not accepted. In this case, 'monoproc' gives the message:

```
> monoproc(fit)

the bandwidth for the monotonizing has to be specified!
```

Acknowledgements The author would like to thank Duncan Temple Lang for helpful comments and suggestions for the implementation of this package.

References

H. Dette, K.F. Pilz, N. Neumeyer (2005). A simple nonparametric estimator of a monotone regression function. To appear in: Bernoulli.

H. Dette, R. Scheder (2005). Strictly monotone and smooth nonparametric regression for two or more variables. Technical report, Department of Mathematics.
http://www.ruhr-uni-bochum.de/mathematik3/preprint.htm

Appendix

This appendix gives the commands for the figures in this paper.

Figure 1

```
> plot(fit3, plot.type = "single", main = "monotone regression",
+     xlab = "speed", ylab = "distance", par.fit = list(col = "black",
```

```
+         cex = 1.5, pch = 13, lwd = 1.5, lty = 3))
> lines(fit1, lty = 1, lwd = 1.5)
> lines(fit2, lty = 2, lwd = 1.5)
> legend(5, 100, c("isoreg", "ksmooth", "monoproc"),
+     col = c(1, 1, 1), lty = c(3, 1, 2))
```

Figure 2

```
> plot(cars.lo, xlab = "speed", ylab = "dist")
> lines(seq(5, 25, 1), predict)
> lines(monofit, lty = 2)
> legend(5, 100, c("loess", "monoproc"), lty = c(1,
+     2))
```

Figure 3

```
> plot(cars, xlab = "Speed (mph)", ylab = "Stopping distance (ft)",
+     las = 1, xlim = c(0, 25))
> lines(d, predict(pr, data.frame(speed = d)))
> lines(mpr, lty = 2)
> legend(5, 110, c("polynomial regression", "monotonized polynom"),
+     lty = c(1, 2))
```

Figure 4

```
> plot(fit, type = "persp", theta = 300, phi = 30,
+     col = "lightblue", cex = 1.3, main = "unconstraint Bodyfat estimate")
> plot(fit, main = "contour plot of the unconstraint estimate")
> plot(fitmono, theta = 300, phi = 30, col = "lightblue",
+     cex = 1.3, main = "monotone Bodyfat estimate")
> plot(fitmono, type = "contour", main = "contour plot of the monotone estimate")
```

Figure 5

```
> persp(x, y, z, phi = 30, theta = 250, col = "lightblue",
+     main = "f(x,y)")
> plot(mono, phi = 30, theta = 250, col = "lightblue",
+     main = "monotone f(x,y)")
> contour(x, y, z, main = "contour plot f(x,y)")
> plot(mono, type = "contour", main = "contour plot monotone f(x,y)")
```

Figure 6

```
> plot(y, z[3, ], type = "l", main = "fixed x=3/51")
> lines(mono@fit@y, mono@fit@z[3, ], lty = 2)
> legend(0.2, 0.14, c("f(x,y)", "monotonized f(x,y)"),
+     lty = c(1, 2))
> plot(x, z[, 3], type = "l", main = "fixed y=3/101")
```

```
> lines(mono@fit@x, mono@fit@z[, 3], lty = 2)
> legend(0.2, 0.1, c("f(x,y)", "monotonized f(x,y)"),
+     lty = c(1, 2))
```