Working Paper 92-17
April 1992

División de Economía
Universidad Carlos III de Madrid
Calle Madrid, 126
28903 Getafe (Spain)
Fax (341) 624-9875

COMPUTING NONPARAMETRIC FUNCTIONAL ESTIMATES IN SEMIPARAMETRIC PROBLEMS

Miguel A. Delgado*

Abstract

We offer a set of FORTRAN routines which compute nonparametric estimates of a number of functionals. The routines are primarily intended to be used in the estimation of semiparametric models. Therefore, the outputs are vectors containing the estimates evaluated at each data point. The routines permit the estimation of conditional expectations, robust conditional location functionals, conditional quantiles and densities. The user may also obtain estimates of other functionals, applied in semiparametric estimation, by defining the input functions appropriately. We also review a number of semiparametric models and discuss their estimation using our routines with the help of standard econometric software.

Key words:
Nonparametric functional estimation, semiparametric models, Fortran routines.

* Departamento de Estadística y Econometría, Universidad Carlos III de Madrid.

COMPUTING NONPARAMETRIC FUNCTIONAL ESTIMATES IN SEMIPARAMETRIC PROBLEMS

Miguel A. Delgado

Department of Economics, Indiana University, Bloomington IN 47405.

Key Words and Phrases: *nonparametric functional estimation; semiparametric models; Fortran routines.*

ABSTRACT

We offer a set of FORTRAN routines which compute nonparametric estimates of a number of functionals. The routines are primarily intended to be used in the estimation of semiparametric models. Therefore, the outputs are vectors containing the estimates evaluated at each data point. The routines permit the estimation of conditional expectations, robust conditional location functionals, conditional quantiles and densities. The user may also obtain estimates of other functionals, applied in semiparametric estimation, by defining the input functions appropriately. We also review a number of semiparametric models and discuss their estimation using our routines with the help of standard econometric software.

1. INTRODUCTION.

Nonparametric functional estimation has been shown to be useful in econometric data analysis and model specification (see e.g. Robinson 1986). Currently, there exist a few packages which perform smooth nonparametric estimation with good graphical facilities (e.g. XploRe of Broich et. al. 1990 or N-Kernel of McQueen 1990). These packages, however, are mainly intended for exploratory data analysis.

Many if not most econometrics models are semiparametric. A parametric structure explaining some basic economic phenomena (e.g. utility or cost functions) is usually known and one is interested in the estimation of these parameters and in making inferences on the assumed structure from the data. However, many features of the data generating process are of unknown form; i.e. the functional form cannot be justified from economic theory and is not of specific economic interest. In the recent econometric literature, the estimation of a number of semiparametric models requires nonparametric estimation of certain functionals in the first step. In these models, it is explicitly recognized that certain features of the underlying distribution of the data are unknown while others follow a known parametric model. The goal is to obtain estimates for the parametric part that are asymptotically equivalent to those obtained when the nonparametric part of the model is perfectly known. A survey of the recent semiparametric econometric literature is in Robinson (1988b). Therefore, once the nonparametric estimates are computed, standard econometric software can be used in order to compute the semiparametric parameter estimates.

Our objective is to provide the user with a battery of routines which produce nonparametric estimates of different functionals frequently employed in semiparametric estimation. This is why the output of our routines is always a vector containing the nonparametric estimates evaluated at each data point. The output can then be read by the user-favored econometric program, e.g. TSP, SAS, LIMDEP etc.

The routines are written in standard FORTRAN-77. They were tested on a VAX/VMS Version V5.3-1. We only report single precision versions. For double precision versions, the usual changes must be made in the code. The routines communicate possible errors by means of PAUSE statements instead of the typical IFAIL parameters.

This document is organized as follows. Section 2 contains a description of the routines. In particular, section 2.1 discusses the functionals to be estimated and the general estimation methods. In sections 2.2-2.5 we discuss the specific estimation procedures, i.e. k nearest neighbors, kernels, kernel nearest neighbors and resampling methods. For each of these methods we explain

the particular routine, the algorithm used, its formal structure and its efficiency. In section 3 we review some semiparametric models and we discuss their estimation using our routines and some popular packages.

2. NONPARAMETRIC ESTIMATION OF REGRESSION CURVES

2.1. INTRODUCTION

Suppose we observe a random sample $\{(Y_i, X_i),\ 1\le i\le n\}$ from the $\mathbb{R}^q\times\mathbb{R}^r$ multivariate random variable (Y, X). We consider the nonparametric regression model,

$$E(Y|X=x)= m(x), \tag{2.1}$$

where m(.) is an unknown function. The purpose of all routines is to obtain estimates of $\{m(X_i),\ i=1,\ldots n\}$. It may be estimated by smooth nonparametric regression techniques. For a survey on smooth nonparametric estimation of regression functionals, the reader may wish to consult Prakasa Rao (1986) or Härdle (1990). Given a sequence of nonparametric weights $\{\omega_i(x),\ 1\le i\le n\}$, $m(x)$ is estimated by,

$$\hat{m}(x)= \sum_j Y_j\ \omega_j(x), \tag{2.2}$$

where, henceforth, the summation run from 1 to n. The estimate in (2.2) has an unbounded influence function. We offer the possibility of estimating $r(x)$ defined as the solution to,

$$E\left[\psi(Y - r(X))|X=x\right]= 0,$$

by

$$\sum_j \psi(Y_j- \hat{r}(x))\ \omega_j(x)= 0, \tag{2.3}$$

where $\psi(.)$ is a function provided by the user, which bounds the influence of the errors (see e.g. Andrews et al 1972 for examples). The statistical literature dealing with smooth nonparametric estimation of robust conditional location functionals is quite extensive; see e.g. Tsybakov (1982), Robinson (1984), Härdle (1984), Härdle and Tsybakov (1988), Boente and Fraiman (1989 and 1990). Examples of ψ-functions are:

$$\text{Huber: } \psi(u)= u\ \max\{1,\ c/|u|\}, \tag{2.4}$$

$$\text{Andrews: } \psi(u)= \sin(u)\ 1(|u|\le c\ \pi), \tag{2.5}$$

$$\text{Bisquare: } \psi(u)= u\ [u^2- c^2]^2\ 1(|u|\le c). \tag{2.6}$$

The constant c is defined inside the functions as a parameter and it can be changed by the user as desired.

We also offer the option of computing conditional L-estimates. In particular, the routines offer the possibility of estimating the α conditional

quantile of Y. The α conditional quantile of Y evaluated at $X=x$ is defined as:

$$q_\alpha(Y|X=x) = q_\alpha(x) = \{q_\alpha^L(x) + q_\alpha^U(x)\}/2,$$

where,

$$q_\alpha^L(x) = \sup\left\{y \mid \Pr\{Y \le y \mid X=x\} \le \alpha\right\},$$

$$q_\alpha^U(x) = \inf\left\{y \mid \Pr\{Y \le y \mid X=x\} \ge \alpha\right\}.$$

Then $q_\alpha(x)$ is estimated from the estimated conditional distribution of Y as suggested by Stone (1977), i.e. by

$$\hat{q}_\alpha(Y|X=x) = \hat{q}_\alpha(x) = \{\hat{q}_\alpha^L(x) + \hat{q}_\alpha^U(x)\}/2, \tag{2.7}$$

where,

$$\hat{q}_\alpha^L(x) = \max_{1\le i\le n}\left\{Y_i \mid \textstyle\sum_j 1(Y_j \le Y_i)\ \omega_j(x) \le \alpha\right\},$$

$$\hat{q}_\alpha^U(x) = \min_{1\le i\le n}\left\{Y_i \mid \textstyle\sum_j 1(Y_j \le Y_i)\ \omega_j(x) \ge \alpha\right\}.$$

For instance, $\hat{q}_{.5}(x)$ is the estimate of the conditional median. L-estimates consisting of combinations of conditional quantiles may be obtained by calling the main routines several times. However, it is possible to just call the routines once, computing the nonparametric weights once, by modifying the main routine.

Note that, unlike $\hat{q}_{.5}(x)$, the robust estimates defined in (2.3) are not scale invariant. Robust estimates of the conditional scale can be computed using the procedure employed for the estimation of conditional quantiles. For instance, a scale measure is

$$s(x) = \text{med}(|Y - q_{.5}(X)| \,|X=x). \tag{2.8}$$

Then $s(X_i)$ is estimated by $\hat{s}(x)$, where the conditional medians in (2.8) are estimated according to (2.7). Then, scale invariant estimates can be computed as the solution to,

$$\textstyle\sum_j \psi(Y_j - \hat{r}(x)/\hat{s}(x))\ \omega_j(x) = 0, \tag{2.9}$$

which can be computed using our software by passing the scale estimate to the robust function using a common statement.

The input in all these routines is the observed sample $\{(Y_i, X_i),\ 1\le i\le n\}$.

In some semiparametric applications, the unknown functionals to be estimated are the conditional expectations of a known parametric function; i.e. one is interested in getting estimates of

$$m(x,\ \theta^0) = E\{g(\theta^0,\ Y,\ X) | X = x\},$$

where $g(.)$ is of known form and θ^0 is a px1 vector of unknown parameters. Given some preliminary root-n-consistent estimate of θ^0, $\hat{\theta}_n$ say, $m(x,\ \theta^0)$ can be estimated by

$$\hat{m}(x) = \sum_j g(\hat{\theta}_n,\ Y_j,\ X_j)\ \omega_j(x).$$

The input here is $\{g(\hat{\theta}_n,\ Y_j,\ X_j),\ i=1,\ldots n\}$.

In some econometric applications, it is possible to express the endogenous variable Y in terms of the exogenous X, a vector of parameters and an unobservable variable V, which is a function of a vector of parameters, Y and X. That is,

$$Y = R(\gamma^0,\ V,\ X).$$

This expression may be obtained by analytical or numerical methods, where $V = h(\delta^0,\ Y,\ X)$ is of known form and γ^0 and δ^0 are vectors of parameters. After a suitable reparameterization we have that,

$$g(\theta,\ Y,\ X) = t(\xi^0,\ V,\ X) \text{ a.s.}$$

where $\xi^0 = (\theta^0,\ \gamma^0,\ \delta^0)$. When V is statistically independent of X,

$$E\{g(\theta^0,\ Y,\ X) | X=x\} \equiv E\{t(\xi^0,\ V,\ x)\}.$$

Then, if some estimate of ξ^0 is available, one can estimate $m(x)$ by

$$\tilde{m}(x) = n^{-1} \sum_j t(\hat{\xi}_n,\ \hat{V}_j,\ x),$$

where $\hat{\xi}_n$ and $\hat{V}_j$ are estimates of ξ^0 and V_j respectively. This method has been used in the estimation of different functionals in econometric nonlinear models (see e.g. Aguirre-Torres and Gallant 1983, Duan 1983 and Brown and Mariano 1984). An interesting application of these techniques is to the estimation of optimal instruments in semiparametric instrumental variable estimation of nonlinear models. In particular, Robinson (1990) proved that the optimal instruments can be estimated using a sample (not necessarily random) without replacement from the empirical distribution of $\hat{V}_i$. The sample size has to increase with n but at an arbitrary rate. This method is computationally very competitive compared to analogues which are based on smooth nonparametric regression, when an explicit formula for $t(.)$ is available. Furthermore, it avoids the problematic choice of a smoothing parameter. Alternatively, Kelejian (1974) proposed to estimate the optimal instruments by bootstrapping (sampling with replacement) from the empirical distribution of $\hat{V}_i$. These estimation methods will be discussed in more detail in sections 2.3

and 3.2.

2.2. NONPARAMETRIC K NEAREST NEIGHBORS REGRESSION

The k nearest neighbors (k-nn) weights presented in this section were introduced by Stone (1977) and have been used in a number of semiparametric applications. In k-nn regression, nonparametric weights $\omega_i(x)= \omega_i(x, k_n)$, are used, where $k_n < n$ is a positive integer provided by the user and

$$\omega_i(x, k_n)= c_{p_i(x)}(k_n)\ 1(p_i(x)\leq k_n). \tag{2.10}$$

In equation (2.10), $1(.)$ is the indicator function, $c_i(.)$ is a function provided by the user such that $\sum_{i=1}^{k_n} c_i(k_n)= 1$ and $c_i(k_n)> 0$ for any $i\leq k_n$, and

$$p_i(x)= 1+ \sum_{j=1}^{n} 1\left\{\rho(X_i, x)\leq \rho(X_j, x)\right\},$$

where $\rho(.,.)$ is a distance function. The user can choose between the following $c_i(.)$ functions (proposed by Stone 1977):

$$\text{Uniform: } c_i(k)= k^{-1} \text{ for } 1\leq i\leq n, \tag{2.11}$$

$$\text{Triangular: } c_i(k)= (k- i+ 1)/b_k \text{ for } 1\leq i\leq n \text{ where } b_k= k(k+ 1)/2, \tag{2.12}$$

$$\text{Quadratic: } c_i(k)= (k- (i-1)^2)/b_k \text{ for } 1\leq i\leq n \text{ where } b_k= k(k+1)(4k-1)/6. \tag{2.13}$$

The number of nearest neighbors to a given point may be located according to the following distance functions,

$$\text{Euclidean: } \rho(X_i, X_j)= \sum_{m=1}^{r} (X^*_{mi}- X^*_{mj})^2, \tag{2.14}$$

$$\text{Least absolute: } \rho(X_i, X_j)= \sum_{m=1}^{r} |X^*_{mi}- X^*_{mj}|, \tag{2.15}$$

$$\text{Maximum: } \rho(X_i, X_j)= \max_m |X^*_{mi}- X^*_{mj}|. \tag{2.16}$$

where $X_i'= (X_{1i},\ldots,X_{ni})'$ and $X^*_{mi}= X_{mi}/s_m$ where $s_m^2= \{\sum_{i=1}^{n}(X_{mi}- \bar{X}_m)^2/(n-1)\}$ and $\bar{X}_m= n^{-1} \sum_i X_{mi}$. The asymptotic properties of k-nn nonparametric estimates have only been studied using the Euclidean distance.

The integer k_n has to be chosen by the user. A common rule of thumb of setting $k_n= [n^{1/2}]$ may give good results. It is also possible to choose k_n using a least squares cross-validation criterion, as suggested by Li (1984). However, it is important to note that semiparametric estimation is not necessarily going to improve by using nonparametric estimates obtained from cross-validation. In semiparametric estimation a precise cross-validation function should be constructed for the particular semiparametric model under consideration. There are few cross-validation results in this case.

The following weights have been used in a number of semiparametric estimation problems and should be used when a cross-validation function is used,

$$\omega_i(X_j,\ k)= 1(i\neq j)\ c_{p_i(X_j)}(k), \tag{2.18}$$

and

$$p_i(X_j)= 1+ \sum_{j=1}^{n} 1\left\{\rho(X_i,\ x)\leq \rho(X_j,\ x)\right\}1(i\neq j).$$

2.2.1. FORTRAN ROUTINES

2.2.1.1. Nearest Neighbors Estimation.

The routine applies the algorithm of Friedman et. al. (1975) for searching for the k_n nearest neighbors to a given observation. The user should provide the data set $\{(Y_i,\ X_i),\ 1\leq i\leq n\}$, the choice of k_n, the value of the three indicator parameters in order to choose the type of estimates desired and external routines to establish the type of weight function ($c_i(.)$) and the robust ψ-function (when it is required).

When the indicator parameter IO1= 0, the own observation is not used; that is, (2.10) is applied. When IO1= 1, the own observation is used; that is, (2.18) is applied. When the indicator parameter AIO2=0, the output is regression estimates as defined in (2.2). When AIO2< 0, robust regression is performed as in (1.3). In this case the user should provide the robust ψ-function by means of an external function. When AIO2> 0, the output is the vector of conditional AIO2 quantiles evaluated at each data point. When IO3=0, the Euclidean distance is applied, when IO3<0, the least absolute distance is applied and when IO3>0, the maximum distance is applied.

Method: The searching algorithm is based on a data preprocessing and a basic procedure. Let $X_i= (X_{i1},\ldots,X_{ir})'$ and $X_{(1)m},\ X_{(2)m},\ldots,\ X_{(n)m}$ the sorted observations with respect to the regressor m. In the preprocessing, the regressors are sorted with respect to each coordinate and then the dispersion of the data around the i-th observation, in the original data set is estimated according to the formula,

$$s_{mi}= \left|X_{(p_{mi}-\ \bar{t}/2)m} - X_{(p_{mi}+\ \bar{t}/2)m}\right|,$$

where $X_{(p_{mi})m}= X_{im}$, p_{mi} is the rank of the observation i-th with respect to the regressor m, and $\bar{t}$ is the estimated maximum number of observations that the algorithm needs to examine before finding the k-th nearest neighbor of X_i. Friedman et. al. (1975) recommended setting $\bar{t}= [t]$, where t is the maximum number of observations necessary to examine when X is uniformly distributed in the r-dimensional unit hypercube. They formally justified that the uniform case is the "less favorable" because the dispersion of the data is expected to

be smaller than when the X density has an infinite support. They found that,

$$t= \pi^{-1/2} \ (k\ r\ \Gamma(r/2))^{1/r}\ (2n)^{1-1/r}. \qquad (2.19)$$

This is why our routine needs a call to a function which computes the log $\Gamma()$. Then the basic procedure for searching the k nearest neighbors of X_i is implemented on the coordinate m(i), where

$$s_{m(i)i}= \max_m s_{mi}.$$

Let $\gamma_m(X_i, X_j)$ denote the distance between X_i and X_j on the coordinate m and $\rho(X_i, X_j)$ the distance on the full dimensionality. The search on the coordinate d= m(i) is performed according to the following algorithm:

0.- Find the k+1 nearest neighbors to X_i according to the coordinate d, using the sorting performed in the preprocessing. Let ${}_dX^{(i)}_{(1)},\ldots,{}_dX^{(i)}_{(k+1)}$ the sorted k+1 nearest neighbors according to the coordinate d., i.e.

$$\gamma_d(X_i, {}_dX^{(i)}_{(1)})\le \gamma_d(X_i, {}_dX^{(i)}_{(2)})\le\ldots\le \gamma_d(X_i, {}_dX^{(i)}_{(k+1)}).$$

Also let $X^{(i)}_{(1)},\ldots,X^{(i)}_{(k+1)}$ be the sorted vector ${}_dX^{(i)}_{(1)},\ldots,{}_dX^{(i)}_{(k+1)}$ according to the full dimensionality, i.e.

$$\rho(X_i, X^{(i)}_{(1)})\le \rho(X_i, X^{(i)}_{(2)})\le\ldots\le \rho(X_i, X^{(i)}_{(k+1)}).$$

Then if

$$\gamma_d(X_i, {}_dX^{(i)}_{(k+1)})> \rho(X_i, X^{(i)}_{(k)}),$$

$X^{(i)}_{(1)},\ldots,X^{(i)}_{(k)}$ are the sorted k nearest neighbors of X_i. Otherwise, set s= 1 and,

1.- Find the k+s+1 nearest neighbor on the coordinate d ${}_dX^{(i)}_{(k+s+1)}$. If,

$$\gamma_d(X_i, {}_dX^{(i)}_{(k+s+1)})> \rho(X_i, X^{(i)}_{(k+1)}),$$

$X^{(i)}_{(1)},\ldots,X^{(i)}_{(k)}$ are the sorted k nearest neighbors of X_i. Otherwise, the distance $\rho(X_i, {}_dX^{(i)}_{(k+s+1)})$ in the full dimensionality is computed in order to sort ${}_dX^{(i)}_{(k+s+1)}$ into $X^{(i)}_{(1)},\ldots,X^{(i)}_{(k)}$ by straight insertion, obtaining the sorted vector $X^{(i)}_{(1)},\ldots,X^{(i)}_{(k+1)}$. Then set s=s+1 and go to 1.

Structure:

SUBROUTINE KNNRE(X,Y,NOBS,NVAR,KNN,IO1,AIO2,IO3,SE,SCL,IW,IRK,WS,NS,
TOL,MAXIT,ROBF,WF)

Formal Parameters:

X	Real Array (NOBS,NVAR)	Input: Matrix of regressors.
Y	Real Array	Input: Dependent variable.

	(NOBS)		
NOBS	Integer	Input:	Number of observations.
NVAR	Integer	Input:	Number of regressors.
KNN	Integer	Input:	Number of nearest neighbors.
IO1	Integer	Input:	Determines whether (2.10) or (2.18) should be applied. IO1=0: Weights computed according to (2.10) IO1=1: Weights computed according to (2.18)
AIO2	Real	Input:	Determines the type of nonparametric estimation performed. AIO2=0: k-nn regression. AIO2<0: Robust k-nn regression. AIO2>0: k-nn AIO2-th conditional quantile estimate.
IO3	Integer	Input:	Determines the distance function used. IO3=0 Euclidean distance. IO3<0 Least absolute distance. IO3>0 Maximum distance.
SE	Real Array (NOBS)	Output:	Nonparametric estimates evaluated at each observation.
SCL	Real Array (NVAR)	Workspace:	
IW	Integer Array (NOBS, NVAR)	Workspace:	
IRK	Integer Array (NOBS, NVAR)	Workspace:	
WS	Real Array (KNN)	Workspace:	
NS	Integer Array (KNN)	Workspace:	
TOL	Real	Input:	Tolerance for robust estimation. It only needs to be defined when AIO2< 0.
MAXIT	Integer	Input:	Maximum number of iterations when computing the robust estimates. It only needs be defined when AIO2< 0.
ROBF	REAL FUNCTION ROBF(A):		External function provided by the user only when AIO2<0. Otherwise, it need not be defined. It is the robust ψ-function where the real A is the argument.
WF	REAL FUNCTION WF(I,KNN):		External function always provided by the user. It is the k-nn weight function; i.e. WF(I,KNN)= C_I(KNN) with $C_I(.)$ defined in (2.10).

Auxiliary Routines: The log of the gamma function is computed using the routine GAMMLN in Press et. al. (1986). The gamma function is an internal function found in some FORTRAN compilers. The user can use any function which computes the gamma function or its logarithm. The subroutines SORT1 and SORT2 are based on the Williams (1964) algorithm, Heapsort, as implemented by Press et. al. (1986). The routine SORT3 perform sorting by straight insertion. Note that the speed of our routine heavily depends on the sorting routine used. Heapsort takes on average $n\log_2 n$ comparisons and it is about 10% more inefficient than average in the worst possible case. The fastest algorithm is Quicksort (Hoare 1962) which takes on average n comparisons but in the worst possible case it takes n^2. The user may change SORT1 and SORT2 by other sorting routines. A discussion of several competing sorting algorithms is in

Knuth (1973). The subroutines MIDD, RIGHT and LEFT perform the basic procedure and the functions DIST1 and DIST2 calculate the distances in one dimension and in the full dimensionality respectively.

Remark: The user can save the space used in IW and IRK by avoiding the preprocessing step in the algorithm. In this case, one starts the search at an arbitrary coordinate; e.g. the first one.

Time: The CPUTIME, reported in seconds, has been computed using the FORTRAN-77 Library Subroutines LIB$INT_TIMER and LIB$STAT_TIMER. The Tables 1-4 refer to the following data. NOBS data points were generated, the X-data is uniformly distributed over the interval (0, 3), the Y-data with density

$$.9\ \phi(Y-\ m(X))+\ .1\ \phi((Y-\ m(X))/9)$$

where $\phi(.)$ denotes the standard normal density. These data were also used by Härdle (1987) in his timing calculations. Table 1 gives timings for different calling modes setting IO1=0, AIO2=0., and IO3= 0, i.e. applying the Euclidean distance. For comparison, the time consumption using the "brute force" method (i.e. for each observation NOBS distances are computed and then sorted in order to find the k nearest neighbors) are reported. The "brute force" timings refer only to the calculation of the distances and sorting of the NOBS vectors (estimates are not computed). The other timings for different KNN and NVAR refer to callings to KNNRE (i.e. not only the k nearest neighbors are located but the estimates are computed).

According to Table 1, with NVAR fixed, the CPUTIME increases quite parsimoniously with KNN; however, with KNN fixed, it increases at a very fast rate with NVAR. Note that according to Friedman et. al. (1975), the case when the X's are uniformly distributed is the least cooperative for this algorithm. We only offer results for uniform weights. Quadratic and Triangular weights are obviously more expensive.

Table 2 reports timings for different values of AIO2 (IO1=0 and IO3=0). When AIO2<0, we have used - here and in other tables- the Huber ψ-function. We have also set AIO2= .5, which corresponds to the conditional median estimates of Y. The cost of the M-estimate is reasonable, but the L-estimate is too expensive compared with the others.

With respect to IO3, obviously IO3< 0 is the cheapest mode and IO3> 0 the most expensive.

TABLE 1

Timing comparisons for different KNN, NVAR and NOBS values (IO1=0 AIO2=0, IO3=0)

NOBS= 500 NOBS= 1000

	NVAR= 1	NVAR= 3	NVAR= 5	NVAR= 1	NVAR= 3	NVAR= 5
BF	9.83	10.78	11.86	42.92	46.58	51.01
KNNRE						
KNN= 3	0.12	1.83	5.04	.21	5.33	17.14
KNN= 6	0.18	2.49	6.05	.37	8.11	20.37
KNN= 12	0.33	3.36	7.53	.57	9.84	26.26
KNN= 24	0.72	4.66	8.33	.94	13.56	28.96
KNN= 48	1.49	6.80	10.39	1.96	19.51	35.51

TABLE 2

Timing comparisons for different KNN, NVAR, NOBS and AIO2 values (IO1=0, IO3=0)

	NOBS= 500			NOBS= 1000		
		NVAR= 1			NVAR= 1	
	AIO2< 0	AIO2= 0	AIO2= .5	AIO2< 0	AIO2= 0	AIO2= .5
KNN= 6	.55	.18	4.98	1.10	.37	19.40
KNN= 24	2.27	.72	20.70	4.14	.99	71.43
		NVAR= 5			NVAR= 5	
KNN= 6	6.33	6.05	10.93	21.39	20.37	40.00
KNN= 24	9.82	8.33	27.29	31.69	28.96	100.68

Accuracy: In Table 3, we report the average square error (ASE) and maximum absolute deviation (MAD) through the NOBS data points for different values of AIO2. The M-estimates and the conditional median estimates perform very similarly. Both robust estimates perform better than the non-robust estimate.

TABLE 3

MAD and SE for different KNN, NVAR, NOBS and AIO2 values (IO1=0, IO3=0)

	NOBS= 500					
		NVAR= 1			NVAR= 5	
	AIO2< 0	AIO2= 0	AIO2= .5	AIO2< 0	AIO2= 0	AIO2= .5
KNN= 6						
MAD	4.842	4.730	4.900	6.888	6.888	5.662
ASE	.366	1.386	.346	.566	1.758	.508
KNN= 24						

MAD	.608	1.426	.812	.720	2.790	.887
ASE	.050	.220	.082	.067	.474	.087

NOBS= 1000

	NVAR= 1			NVAR= 5		
	AIO2< 0	AIO2= 0	AIO2= .5	AIO2< 0	AIO2= 0	AIO2= .5
KNN= 6						
MAD	1.287	2.567	1.514	1.404	4.344	1.393
ASE	.132	.687	.176	.134	.959	.161
KNN= 24						
MAD	.503	1.172	.646	.503	1.172	.646
ASE	.036	.194	.056	.036	.194	.056

2.3. NONPARAMETRIC KERNEL REGRESSION

The Kernel regression method was introduced by Nadaraya (1964) and Watson (1964). Applications of this method in semiparametric estimation are in abundant supply (see section 3).

In kernel regression, nonparametric weights $\omega_i(x) = \omega_i(x, h_n)$ are used where,

$$\omega_i(x, h_n) = K_i(x)/\hat{f}(x), \tag{2.19}$$

and

$$\hat{f}(x) = \sum_i K_i(x), \tag{2.20}$$

$$K_i(x) = (n\, h_n^r \det(\Sigma_n)^{1/2})^{-1} K\left(h_n^{-1}\, \Sigma_n^{-1/2}(x - X_i)\right),$$

$\hat{f}(x)$ is the multivariate density estimate of X evaluated at x while h_n is a smoothing parameter provided by the user. Σ_n is a nonsingular matrix which may be provided by the user. This matrix can be used to scale the X's. In particular we offer the option of using the sample covariance matrix of X. The user may decide to used a nondiagonal matrix of bandwidths defining Σ_n appropriately. K(.) is a kernel function which is provided by the user. The most popular kernel is the gaussian, i.e.

$$K(x) = (2\pi)^{-r/2} \exp\{-x'x/2\}.$$

A computationally attractive alternative is the Epanechnikov kernel (see Silverman 1986); i.e.

$$K(x) = .5\, c_r\, (r+2)\, (1 - x'x)\, 1(x'x < 1),$$

where c_r is the volume of the r-dimensional sphere: $c_1 = 2$, $c_2 = \pi$, $c_3 = 4\pi/3$ etc. In some semiparametric applications, kernels have been used of the form,

$$K(x) = \prod_{j=1}^{r} k(x_j),$$

where $x_j = (x_1, \ldots, x_r)'$ and $k: \mathbb{R} \longrightarrow \mathbb{R}$ is an even function. In this case one set Σ_n to the unit matrix. Robinson (1988) introduced Barlett's (1963) "high order" kernels for bias-reduction, to the estimation of semiparametric models. Robinson (1988) defined a high order kernel of order ℓ as that satisfying

$$\int_{\mathbb{R}} u^i \, k(u) \, du = \delta_{0i}, \quad i = 0, \ldots, \ell-1,$$

$$k(u) = O\left((1 + |u|^{\ell+1+\varepsilon})\right) \text{ some } \varepsilon > 0,$$

where δ_{ij} is the Kroneker's δ.

Note that the estimate of $m(x)$ is,

$$\hat{m}(x) = \hat{d}_n(x) / \hat{f}_n(x) \tag{2.21}$$

where,

$$\hat{d}_n(x) = \sum_i K_i(x) \, Y_i \tag{2.22}$$

The output of the kernel routine is $\{(\hat{d}_n(X_i), \hat{f}_n(X_i)), i = 1, \ldots, n\}$. The user may compute $\hat{m}(X_i)$ easily from (2.21). Both, $\hat{d}_n(X_i)$ and $\hat{f}_n(X_i)$, have been used in a number of semiparametric estimation and testing problems (see section 3).

Once Σ_n has been set, the choice of h_n may be done by a process of trial and error. It may be a tedious task. In order to minimize trouble, the user may follow the recommendations of Silverman (1986) for the estimation of multivariate densities. Another possibility is to select h_n automatically by optimizing some cross-validation criterion function, as we have discussed for k-nn (see e.g. Härdle and Marron 1990).

2.3.1. FORTRAN ROUTINES

2.3.1.1. Kernel estimation with symmetric kernels

The output of this routine are kernel estimates of $\hat{f}_n(X_i)$ and $\hat{d}_n(X_i)$, $i=1,\ldots,n$. The kernel functions satisfy the condition $K_i(x) = K_i(-x)$. Using this condition, the routine evaluates the kernel function $n(n-1)/2$ times. Beside the storage space required to keep the output and Σ, one only needs an array of dimension NVAR as workspace (this is used in order to evaluate the kernel function). The indicator parameter IO1 establishes when Σ is provided by the user and IO2 when Σ is diagonal or not. When IO1=0 and IO2=0, Σ is a diagonal matrix with diagonal components the sample variances of the regressors. When IO1=0 and IO2≠0, Σ is the sample covariance of the regressors. When IO1≠0 and IO2=0, Σ is provided by the user as a vector containing the diagonal components. When IO1≠0 and IO2≠0, Σ is a symmetric matrix provided by the user as a vector containing the lower triangular

components of Σ.

Method: The algorithm is efficient with respect to storage and requires n(n-1)/2 calls to the kernel function. Improvements on CPUTIME depend on the particular kernel function used. For instance, the gaussian kernel function should be set to zero in single precision when $\sum_i x'\Sigma^{-1}x \geq 30$ because exp(-15)≈ 30E-08 which is negligible in single precision. As Silverman (1986) noted, some kernels are multiplied by a given constant (e.g. the gaussian kernel is multiplied by $(2\pi)^{-r/2}$). These n(n-1)/2 multiplications should be saved. If the user just wants conditional expectation estimates, the results are fine without performing the multiplications. When estimates $\hat{f}_n(X_i)$ and $\hat{d}_n(X_i)$ are desired, the output can be multiplied by the given constant after the call to the routine.

Recently Silverman (1982) and Härdle (1987) have proposed kernel estimates based on the fast Fourier transform when X_i is an scalar. In particular Silverman (1982) noted that the Fourier transform of $\hat{f}_n(x)$ is given by,

$$\tilde{f}_n(\omega) = \tilde{K}(h\omega)\ u(\omega)$$

where $\tilde{K}()$ is the Fourier transform of the Kernel function and

$$u(\omega) = n^{-1} \sum_{j=1}^{n} \exp\{i\omega X_j\}$$

$u(\omega)$ is found by constructing a histogram of 2^k cells and then the fast Fourier transform is applied. Next estimates of f(.) are found by inverse Fourier transform of $\tilde{f}_n(\omega)$. In this way one can obtain 2^k estimates of $\hat{f}_n(x_i)$, where $x_i = X_{(1)} + (i-1/2)(X_{(n)} - X_{(1)})/2^k$, where $X_{(1)}$ and $X_{(n)}$ are, respectively, the lower and upper limit of the interval on which the estimate is calculated. It seems difficult to implement this method in order to get estimates evaluated at precise data points. On the other hand, this method has only be implemented in one dimension. The implementation in higher dimensions seems cumbersome.

Structure:

SUBROUTINE KEREGSY(X,Y,NOBS,NVAR,HB,IO1,IO2,B,SE,FK,W,FKER)

Formal Parameters:

X	Real Array (NOBS,NVAR)	Input: Matrix of regressors.
Y	Real Array (NOBS)	Input: Dependent variable.
NOBS	Integer	Input: Number of observations.
NVAR	Integer	Input: Number of regressors.
HB:	Real	Input: Bandwidth parameter.
IO1	Integer	Input: Determines whether or not Σ is provided by the user. IO1=0: Σ is set to the sample covariance

matrix.
IO1≠0: Σ is provided by the user as a lower triangle stored in B.

IO2 Integer Input: Determines when Σ is diagonal or not.
IO2=0: Σ is diagonal.
IO2≠0: Σ is not diagonal.

B Real Array
(NVAR) when IO2=0
(NVAR×(NVAR+1)/2) when IO2≠0
Input/Workspace: When IO1≠0 and IO2=0, B should contain the diagonal elements of Σ. When IO1≠0 and IO2≠0, B should contain Σ as lower triangle, e.g. suppose NVAR=3, Σ is stored into B as σ_{11}, σ_{21}, σ_{22}, σ_{31}, σ_{32}, σ_{33}. Otherwise, B is used as workspace.

SE Real Array Output: Contains $\hat{d}_n(X_i)$, i= 1,...,n.
(NOBS)

FK Real Array Output: Contains $\hat{f}_n(X_i)$, i=1,...,n.
(NOBS) when IO2=0
max{NOBS, (NVAR×(NVAR+1)/2)} when IO2≠0

W Real Array Workspace:
(NVAR)

FKER REAL FUNCTION FKER(W, NVAR): External function defined by the user. It is the multivariate kernel function.

Auxiliary routines: The routine uses the RSS algorithms A6 and A7 (Healey 1968 a and b) for Cholensky decomposition and inversion of a symmetric matrix.

Time: Using the same data as in KNNRE and a gaussian kernel, Table 4 gives CPUTIMEfor different calling modes where diag(Σ)=(1,1,...,1)' and HB=.5 in all cases. It is interesting to note that k-nn regression is not necessarily more expensive than kernel regression.

TABLE 4

Timing for different NVAR and NOBS values. B is the unit matrix and HB= .5

	NVAR= 1	NVAR= 3	NVAR= 5
NOBS= 500	2.84	3.86	4.80
NOBS= 1000	12.52	16.07	19.48

2.2.1.2. Kernel estimation with possibly non-symmetric kernels:

The storage requirements for this routine are the same as in KEREGSY but now n(n+1) calls to the kernel function are needed because symmetry can not be exploited. Robust conditional expectation estimates and conditional quantile estimates can be also computed. The output in these routines is the same as in KNNRE.

Structure:

SUBROUTINE KEREGG(Y,X,NOBS,NVAR,HB,IO1,IO2, AIO2,B,SE,FK,W,FKER,

ROBF, TOL, MAXIT)

Formal Parameters: All the parameters are the same as in KNNRE and KREGSY.

Note that:

IO1: The same as in KEREGSY.
IO2: The same as in KEREGSY.
SE: The same as in KNNRE.
FK: The same as in KEREGSY, but on exit it does not contain relevant information. It is just workspace.

Time: The routine is twice as expensive as KREGSY.

2.4. NONPARAMETRIC KERNEL NEAREST NEIGHBORS.

This method was introduced by Collomb (1980). It is similar to the kernel method but now the bandwidth changes at each point where the regression function is estimated. The weights are the same as in (2.19) except that:

$$K_i(x) = (n\, H(k_n, x)^r)^{-1}\, K\Big(H(k_n, x)(x - X_i)\Big),$$

where $H(k_n, x)$ is the distance between x and the nearest neighbor of x. Note that when $K(u) = 1(\|u\|_r \leq 1)$, the weights in (2.19) are just the uniform k-nn weights.

2.4.1. FORTRAN ROUTINE

The routine is a combination of KNNRE and KEREGG. The distances to the k-th nearest neighbor are computed using the algorithm in KNNREG and then the kernel estimates are computed by using the "brute force" method in KEREGG. Note than even with symmetric kernels $\omega_i(X_j) \neq \omega_j(X_i)$ because the "bandwidth" change from observation to observation. This routine is about as expensive as KNNRE and KEREGG together.

Structure:

SUBROUTINE KERNN(X, Y, NOBS, NVAR, KNN, AIO2, IO3, SE, FK, SCL, IW, IRK, WS, NS, W, FKER, ROBF, MAXIT, TOL)

Formal Parameters: All the parameters obey the definitions in KEREGG and KNNRE.

2.5. NONPARAMETRIC ESTIMATES BASED ON RESAMPLING.

In this case, one is interested in estimating $m(x) = E\{g(\theta^0, Y, X) | X=x\}$ where $g(.)$ is a known function and θ^0 is a vector of unknown parameters. It is known that,

$$g(\theta^0, Y, X) = t\Big(\xi^0, h(\delta^0, Y, X), X\Big) \text{ a.s.},$$

where $t(.)$ may be obtained by numerical or analytical methods, ξ^0 and δ^0 are unknown parameters, $h(.)$ is a known function and $V = h(\delta^0, Y, X)$ is

independent of X. Consequently, $m(\alpha)$ is estimated by,

$$\tilde{m}(\alpha)= M^{-1} \sum_{j\in M(\alpha)} \hat{t}_j(\alpha), \tag{2.24}$$

where $M(\alpha)$ is a sample of size M of the integers $\{1,2,\ldots,n\}$ and

$$\hat{t}_j(\alpha)= t\left(\hat{\xi}_n,\ \hat{V}_j,\ \alpha\right), \tag{2.25}$$

where $\hat{\xi}_n$ and $\hat{\delta}_n$ are estimates of ξ^0 and δ^0, respectively, and $\hat{V}_j= h(\hat{\delta}_n,\ Y_j,\ X_j)$.

The estimates defined in (2.24) have been used for optimal instrumental variables estimation in nonlinear simultaneous equations models. The routine discussed below permits to choose $M(\alpha)= M$ fixed for all α (the same $\hat{V}_j$'s are used in computing all the $\tilde{m}(X_i)$ for i=1,...,n), $M(\alpha)$ being a random sample without replacement and $M(\alpha)$ being a random sample with replacement. We do not consider the possibility of using different sample sizes for each nonparametric estimate. This feature would require an extra array of n elements. The user can easily add this feature. The input in this routine will be $\{(Y_i,\ X_i,\ \hat{V}_i),\ i= 1,\ldots,n\}$.

Robinson (1990) considered the following linear transformation model:

$$V = h(\theta^0,\ Y,\ X) = \operatorname{arcsinh}(\lambda^0 Y)/\lambda^0 - \alpha^0 - \beta^{0'} X,$$

where $\theta= (\lambda,\ \alpha,\ \beta')'$ and $E(V) = E(V|X= \alpha) = 0$. The optimal instruments in a nonlinear three stages least squares procedure is the vector:

$$H_i'= \left(-1,\ -X',\ E[g(\theta^0,\ Y,\ X)|X=\alpha]\right), \tag{2.26}$$

where,

$$g(\theta^0,\ Y,\ X)= \partial V(\theta^0,\ Y,\ X)/\partial\theta= \tanh[\lambda^0(\alpha^0+ \beta^{0'} X+ V)]/\lambda^{0^2} - (\alpha^0- \beta^{0'} X)/\lambda^0$$

Obviously,

$$E\{\tanh[\lambda^0(\alpha^0+ \beta^{0'} X+ V)]|X=\alpha\} = E\{\tanh[\lambda^0(\alpha^0+ \beta^{0'} X+ V)]\}.$$

Then, $\xi^0= \theta^0$ and $t(\xi^0,\ V,\ X)= \tanh[\lambda^0(\alpha^0+ \beta^{0'} X+ V)]/\lambda^{0^2} - (\alpha^0- \beta^{0'} X)/\lambda^0$.

2.5.1 FORTRAN ROUTINE

This routine computes estimates as defined in (2.24). A function which specifies $t(\xi^0,\ V,\ X)$ is required. Y_i, X_i, $\hat{V}_i$ or other vectors needed for the computation of t(.) should pass to the function through common statements. An example is given below.

Method: When IO=0, the conditional expectation estimates are just an arithmetic mean. When IO< 0, a random sample without replacement is used. The

speed of the program will depend on the algorithm used for random sampling. Efficient algorithms for random sampling are abundant (see Knuth 1969 and Devroy 1986). As usual, there is a trade off between efficiency and complexity of the program, between speed and space. In order to perform the random sampling, we of course require a random generator for uniforms (0, 1). We have used the Knuth portable generator as implemented by Press et. al. (1986). It seems preferable to the RAN internal function provided in the FORTRAN compilers. The user, however, can use any other generator, e.g. the NAG or the IMSL. The generator is initialized by the number of seconds elapsed from midnight (SECNDS function). It may alternatively be initialized by the nearest integer to the first observation of the first regressors or by whatever procedure the user finds convenient. Among the many algorithms available for random sampling, we have chosen the sequential sampling using the spacing method as implemented by Devroy and Yuen's (1981) (see also Devroy 1986 Chapter XII). This algorithm takes expected time proportional to MX. The method is more complicated than standard sequential sampling (Knuth's 1969 algorithm S) which takes expected time proportional to NOBS. The random sampling algorithm based on 'suffling' or 'swapping' also takes expected time proportional to MX. However, in this case, it is necessary to swap the original X matrix or use an extra integer array of dimension NOBS. For an excellent presentation of random sampling algorithms, the reader may consult Devroy (1986). For random sampling with replacement (IO1>0), we have used the fact that if U is uniformly distributed in the interval (0,1), $Z=\lfloor 1+ U*NOBS \rfloor$ is a random integer uniformly distributed in the interval (1,...,NOBS) (see Knuth 1969). Consequently, we can do the bootstrapping using the random uniform generator.

Structure:

SUBROUTINE RESA(NOBS,NVAR,MX,THETA,NPAR,SE,IO,TFUN)

Formal Parameters:

NOBS	Integer	Input:	Number of observations.
NVAR	Integer	Input:	Number of regressors in the exogenous variable.
MX	Integer	Input:	Sample size used to compute the estimates.
THETA	Real Array (NPAR)	Input:	Preliminary parameter estimates.
NPAR	Integer	Input:	Number of parameters.
IO	Integer	Input:	Determines the type of resampling performed. IO= 0: $M(X_i)$= {1,2,...,MX} i=1,..,n. IO< 0: $M(X_i)$ is a sample without replacement of size MX from the integers {1,...,n}. IO> 0: $M(X_i)$ is a sample with replacement of size MX from the integers {1,...,n}.
SE	Real Array (NOBS)	Output:	Conditional expectation estimates evaluated at each data point.
TFUN	REAL FUNCTION TFUN(NOBS,NVAR,THETA,NPAR,I,J):		

External function provided by the user which should compute $\hat{t}_J(X_I)$ as defined in (2.25).

Example: Consider the arcsinh model discussed above. Suppose the preliminary estimate of λ^0 is stored in theta(1), the estimate of α^0 is stored in theta(2) and theta(j), j=3,..,npar contains the estimates of β^0. Suppose NOBS=100 and NVAR=4. Then the TFUN would be:

```
      function tfun(nobs,nvar,theta,npar,i,j)
      real theta(*)
      common dat/x(100,4),y(100)/
      c=0.
      do 1,k=3,npar
1     c=c+x(j,k-2)*theta(k)
      c=c+theta(2)
      y1=y(j)*theta(1)
      y2=y1*y1
      v=log(y1+sqrt(y2+1))/theta(1)-c
      c=0
      do 2,k=3,npar
2     c=c+x(i,k-2)*theta(k)
      c=theta(2)+c
      v=(v+c)*theta(1)
      v=tanh(v)/(theta(1)*theta(1))
      c=c/theta(1)
      tfun=v-c
      return
      end
```

The reader has surely realized that it is not an efficient function. The function is called by the mean routine M*NOBS times. Hence, it is more efficient to perform some calculations in the main program i.e.

```
      do 1,i=1,nobs
      c=0.
      do 2,k=3,npar
2     c=c+x(i,k-2)*theta(k)
      c=theta(2)+c
      x1(i)=c*theta(1)
      x2(i)=c/theta(1)
      d=y(i)*theta(1)
      dd=(d*d)+1
      d=log(d+sqrt(dd))/theta(1)-c
1     v(i)=d*theta(1)
      tt=theta(1)*theta(1)
```

The arrays x1, x2 and v and tt are passed to TFUN through a common statement, i.e.

```
      function tfun(nobs,nvar,theta,npar,i,j)
      real theta(*)
      common dat/x1(100),x2(100),v(100),tt/
      c=v(j)+x1(i)
      c=tanh(c)/tt
      tfun=c-x2(i)
      return
      end
```

3. ESTIMATION OF SEMIPARAMETRIC MODELS

3.1. ASYMPTOTICALLY EFFICIENT ESTIMATION IN THE PRESENCE OF HETEROSKEDASTICITY OF UNKNOWN FORM

Consider the model,

$$E[Y|X=x]= g(\theta^0, x),$$
$$Var(Y|X=x)= \Omega(x),$$

where $g(.,.)$ is a qx1 vector of functions of known form, θ^0 is px1 vector of unknown parameters and $\Omega(.)$ is a qxq matrix of unknown functions.

Given a preliminary estimate of θ^0, say $\tilde{\theta}_n$, the $\alpha\beta$-th unknown component of $\Omega(X_i)$, $\sigma_{\alpha\beta}(X_i)$, is estimated by

$$\hat{\sigma}_{\alpha\beta}(X_i)= \sum_j U_{\alpha j}(\tilde{\theta}_n)U_{\beta j}(\tilde{\theta}_n)\ \omega_j(X_i), \tag{3.1}$$

where $U_{\alpha j}(\theta)$ is the α-th component of $U_j(\theta)= Y_j- g(\theta^0, x)$. Alternatively, $\sigma_{\alpha\beta}(X_i)$ may be estimated by,

$$\hat{\sigma}_{\alpha\beta}(X_i)= \sum_j Y_\alpha Y_{\beta j}\ \omega_j(X_i)- \sum_j Y_\alpha\ \omega_j(X_i)\sum_j Y_{\beta j}\ \omega_j(X_i). \tag{3.2}$$

Some components of $\Omega(X_i)$ may be known or their functional form may be known in which case they can be estimated by parametric methods. So, we can construct the minimum distance estimate,

$$\bar{\theta}_n= \text{argmin}_\theta \sum_i U_i(\theta)'\ \hat{\Omega}_i^{-1}\ U_i(\theta). \tag{3.3}$$

It has been proved to be first order efficient, under certain regularity conditions, by Carroll (1982), Robinson (1987), Newey (1987), Delgado (1989a and b). That is,

$$\hat{V}_n^{-1/2}\ n^{1/2}(\bar{\theta}_n- \theta^0)\ \longrightarrow_d N(0,\ I_q),$$

where $\hat{V}_n= n^{-1}\sum_i \dot{g}(\bar{\theta}_n,\ X_i)'\ \hat{\Omega}_i^{-1}\ \dot{g}(\bar{\theta}_n,\ X_i)$ and $\dot{g}(\theta,\ X_i)= \partial g(\theta,\ X_i)/\partial\theta$.

The estimates defined in (3.3) have an unbounded influence function in residuals and leverage. Delgado (1990) proposed to use the estimates defined in (3.1) and (3.2) in order to correct for heteroskedasticity in the linear regression model using GM-estimates. It is also possible to scale by a robust estimate of the conditional scale. In particular, θ^0 can be estimated as the solution to

$$\sum_i \Psi\left[(Y_i- X_i'\dot{\theta}_n)/\hat{s}(X_i),\ X_i\right]\ X_i/\hat{s}(X_i)= 0, \tag{3.4}$$

where $\hat{s}(X_i)$ is some robust estimate of the conditional scale.

In the single equation case, the estimate defined in (3.3) can be easily computed by using any standard econometric package. The NAG and IMSL libraries offer a number of routines for minimization of sum of squares. The

Levenberg-Marquardt algorithm is also implemented in Press et. al. (1988). These FORTRAN routines can be used after computing the conditional variance estimates and modifying the independent variable and the nonlinear function appropriately. In TSP, one can use the LSQ command. For instance, suppose our data set consist of a dependent variable Y and 2 regressors (X), which are stored in the file 'DATA.DAT'. The model to be estimated is $E[Y|X=x]= \exp(\alpha^0+ \beta_1^0 X_{1i}+ \beta_2^0 X_{2i})$. Suppose, further that, we have estimated $\{Var(Y|X=X_i), i=1,\ldots,n\}$ according to (3.1) or (3.2) and the estimates are stored in the file 'WEIGHT.DAT'. The following TSP program computes the estimates of (3.3) and the corresponding standard errors by

```
read(file='data.dat')y x1 x2;
read(file='weight.dat')w;
w=sqrt(w);
y=y/w;
frml tot y=exp(b0+b1*x1+b2*x2)/w;
param b0-b2;
lsq tot;
stop;
```

The resulting standard errors and t-ratios should be divided by the 'standard deviation of the regression'.

In the multivariate case, TSP can still be used but the TSP program will be more cumbersome. Suppose we have the following equations:

$$E(Y_1|X=X_i)= \exp\{\alpha_1^0+ \beta_{11}^0 X_{1i}+ \beta_{12}^0 X_{2i}\},$$

$$E(Y_2|X=X_i)= \exp\{\alpha_1^0+ \beta_{21}^0 X_{1i}+ \beta_{22}^0 X_{2i}\}.$$

The data is stored in the file 'DATA.DAT' and we have estimated $\{Var((Y_1,Y_2)'|X=X_i), i=1,\ldots,n\}$ by $\{\hat{\Sigma}_i, i=1,\ldots,n\}$ according to (3.1) or (3.2). So, we can compute $\{\hat{\Sigma}_i^{-1/2}, i=1,\ldots,n\}$ (using e.g. Healey 1968a and b), which is stored in the file 'WEIGHT.DAT' with i-th raw component $(\hat{\sigma}_i^{11},\hat{\sigma}_i^{12},\hat{\sigma}_i^{22},\hat{\sigma}_i^{13},\hat{\sigma}_i^{23},\hat{\sigma}_i^{33},\ldots.)$, where $\hat{\sigma}_i^{\alpha\beta}$ is the $\alpha\beta$-th component of $\hat{\Sigma}_i^{-1/2}$(it is the output of CHOL in Healey 1968). Then the following TSP program will compute the estimates in (3.3)

```
read(file='data.dat')y1,y2,x1,x2;
read(file='weight.dat')s11,s12,s22;
y11=y1*s11+y2*s12;
y22=y2*s22;
frml tot1 y11=eq1*s11+eq2*s12;
frml tot2 y22=eq2*s22;
frml eq1 exp(a1+b11*x1+b12*x2);
frml eq2 exp(a2+b21*x1+b22*x2);
param a1,a2,b11,b12,b21,b22;
eqsub tot1 eq1 eq2;
eqsub tot2 eq2;
lsq(maxit=0) tot1 tot2;
```

When the equations are linear, it may be easier to use any FORTRAN program for

solving linear equation systems; e.g. using GAUSSJ routine in Press et. al. (1986). On the other hand one can also use the trick suggested by Gallant (1975), transforming the multivariate problem to an univariate one; i.e.

$$\hat{\theta}_n = \text{argmin}_\theta \sum_{s=1}^{nq} (Y_s^* - f_s^*(\theta))^2,$$

where $s = q(i-1)+\alpha$, $Y_s^* = (\hat{\sigma}_i^\alpha)'\, Y_i$ and $f_s^*(\theta) = (\hat{\sigma}_i^\alpha)'\, f(\theta, X_i)$, $\hat{\sigma}_i^\alpha = (\hat{\sigma}_i^{\alpha 1}, \hat{\sigma}_i^{\alpha 2}, \ldots, \hat{\sigma}_i^{\alpha q})$ for $i=1,\ldots,n$ and $\alpha=1,\ldots,q$. Then any routine for univariate nonlinear regression can be used.

The robust semiparametric estimates (3.5) can be computed using the NAG-13 libraries G02HDF and G02FF. One can also use the ML command in TSP-4.1 or the anologue routine in GAUSS.

3.2. OPTIMAL SEMIPARAMETRIC INSTRUMENTAL VARIABLE ESTIMATORS

Suppose it is given that

$$E\left[U(\theta^0,\ Y,\ X)\,|\,X\right] = 0, \tag{3.5}$$

where $U(.)$ is a qx1 vector of known functions, θ^0 is an unknown px1 vector of parameters, X is a rx1 random variable and Y is a sx1 random variable. It is also known that,

$$\text{Var}\left[U(\theta^0,\ Y,\ X)\,|\,X=x\right] = \Omega(x),$$

where $\Omega(x)$ is positive definite for all X where some components are unknown functions. Let H_i be a matrix of instruments, such that

$$H_i = R(X_i,\ \theta^0)\ \Omega(X_i)^{-1},$$

where,

$$R(X_i,\ \theta^0) = E\left[\partial U(\theta^0,\ Y_i,\ X_i)/\partial\theta\,|\,X_i\right],$$

and some components of $R(X_i,\ \theta^0)$ are of unknown functional form. Amemiya (1977) defined the optimal nonlinear least squares (NL3S1S) as

$$\hat{\theta}_n = \text{argmin}_\theta \sum_i U_i(\theta)'\, H_i'\, \left(\sum_i H_i\ \Omega(X_i)\ H_i'\right)^{-1} \sum_i H_i U_i(\theta).$$

Since some components of $\Omega(X_i)$ and $R_i(X_i,\ \theta^0)$ are of unknown functional form, a feasible estimate may be constructed by estimating the unknown components of $\Omega(X_i)$ and $R_i(X_i,\ \theta^0)$ by nonparametric regression. Let $\hat{\Omega}_i$ and $\hat{R}_i$ nonparametric estimates of $\Omega(X_i)$ and $R_i(X_i,\ \theta^0)$, respectively. The optimal instruments can be estimated by $\hat{H}_i = \hat{R}_i\hat{\Omega}_i^{-1}$ and θ^0 would be estimated by,

$$\tilde{\theta}_n = \text{argmin}_\theta \sum_i U_i(\theta)' \hat{H}_i' \left(\sum_i \hat{H}_i \hat{\Omega}_i \hat{H}_i'\right)^{-1} \sum_i \hat{H}_i U_i(\theta). \qquad (3.6)$$

Similarly, when the $\alpha\beta$-th component of $R(\theta^0, X_i)$, $r_{\alpha\beta}(\theta^0, X_i)$, is of unknown functional form, it is estimated by,

$$\hat{r}_{\alpha\beta}(X_i) = \sum_j c_{\alpha\beta}(Y_j, X_j, \tilde{\theta}_n)\, \omega_j(X_i),$$

where $c_{\alpha\beta}(Y_j, X_j, \theta)$ is the $\alpha\beta$-th component of the matrix $\partial U(\theta, Y_i, X_i)/\partial\theta$. . When $r_{\alpha\beta}(\theta, X_i)$ is of known functional form, it is estimated by $\hat{r}_{\alpha\beta}(X_i) = r_{\alpha\beta}(\tilde{\theta}_n, X_i)$. Then, $\hat{R}_i$ is the matrix with components $\hat{r}_{\alpha\beta}(X_i)$.

The feasible optimal NL3SLS (3.3) has been proposed by Newey (1987). In the case when $\Omega(X_i)$ is constant for all i, the optimal instruments are estimated by $\tilde{H}_i = \hat{R}_i \tilde{\Omega}^{-1}$; where

$$\tilde{\Omega} = n^{-1} \sum_j U_j(\tilde{\theta}_n) U_j(\tilde{\theta}_n)'.$$

Newey (1990) has proved, under regularity conditions, that the corresponding NL3SLS is first order efficient. As noted by Newey (1987, 1990), a first order asymptotically equivalent estimate is the solution to,

$$\sum_i \hat{H}_i\, U_i(\mathring{\theta}_n) = 0. \qquad (3.7)$$

This estimate may be easier to compute, with standard software, than (3.3).

For certain econometric models, Robinson (1990) has proposed to estimate $R(\theta^0, X_i)$ by estimates as defined in section 2.5.. In particular, if $c_{\alpha\beta}(Y_i, X_i, \theta) = t_{\alpha\beta}\left(\xi(\theta), h(\theta, Y_i, X_i), X_i\right)$, where $t_{\alpha\beta}(.)$, and $h(.)$ are known functions , $V_i = h(\theta^0, Y_i, X_i)$ is independent of X_i, and $r_{\alpha\beta}(\theta, X_i)$ is of unknown form, we can estimate $r_{\alpha\beta}(\theta^0, X_i)$ using RESA. Using a "superlanguage" (e.g. GAUSS or APL), (3.6) can be computed straightforwardly. Standard packages can also be used. In the one equation case one can use the TSP command LSQ. For instance, consider the arcsinh tranformation model discussed in section 2.5 with only one regressor. First preliminary estimates of θ^0 are needed in order to compute estimates of the optimal instruments. We can use as preliminary estimates the intrumental variables one with instruments

$$W_i = (-1, -X_i, -X_i^2)'.$$

and the following TSP program (the data is in 'DATA.DAT')

```
read data(file='data.dat')y,x;
w1=-1;
w2=-x;
w3=-x*x;
frml tot zero=(log(yl+ya)/lamda)-a-b*x;
frml yl y*lamda;
frml ya sqrt((yl*yl)+1);
```

```
zero=0;
equsub tot yl ya;
param lamda,1,a,1,b,1;
lsq(hiter=d,inst=(w1-w3)) tot;
```

Once the estimate of θ^0, $\tilde{\theta}_n = (\tilde{\lambda}_n, \tilde{\alpha}_n, \tilde{\beta}_n')'$, has been obtained the optimal instrument are computed as

$$\hat{H}_i = \left(-1, -X_i, \hat{r}(X_i)\right),$$

where $\hat{r}(X_i) = \hat{m}(X_i)/\tilde{\lambda}_n^2 - \tilde{\beta}_n X'/\tilde{\lambda}_n$ and $\hat{m}(X_i)$ are nonparametric estimates of $m(X_i) = E\{\tanh(\lambda^0(\beta^{0'} X + V)|X\}$. Note that when an intercept is included, it can not be used in the nonparametric regression. Alternatively, an estimate of $m(X_i)$ can be obtained using (2.25). Suppose that $\hat{m}(X_i)$ was computed using any routine and was stored in the file 'INSTRU.DAT'. The following TSP code should be added to the above TSP code in order to get the semiparametric NL3SLS estimates:

```
read(file='instru.dat')w3;
lsq(hiter=d,maxit=0,inst=(w1-w3)) tot;
```

Another example, considered by Newey (1990), is the following linear regression model,

$$U_i(\theta^0) = Y_i - X_i'\beta^0 - \delta^0 D_i,$$

where D_i is dummy variable such that $E\{U_i(\theta^0)D_i|X_i, Z_i\} \neq 0$ and Z_i and X_i are non-nested. In this case, optimal instrumental variables estimates can be obtained by using the command INST in TSP. Suppose that $X_i = (1, X_{2i})'$ and Z_i is a scalar random variable. The data (Y_i, X_{1i}, D_i) are in three columns on the file 'DATA.DAT'. Estimates of $E(D_i|Z_i)$ can be obtained using some of the routines in Section 2. Suppose that these estimates are kept in the file INSTRU.DAT. The following TSP program obtains the semiparametric estimates:

```
read(file='data.dat')y x d;
read(file='instru.dat')w;
inst y c x d invr c x w;
```

3.3 SEMIPARAMETRIC PARTIALLY LINEAR MODELS

Robinson (1988) considered the following regression model:

$$E(Y|X = x, Z = z) = \alpha^0 + x'\beta^0 + \gamma(z),$$

where $\gamma(.)$ is of unknown functional form. Noting that:

$$E(Y|Z = z) = \alpha^0 + E(X|Z = z)'\beta^0 + \gamma(z),$$

Robinson (1988) proposed to estimate β^0 by

$$\hat{\beta}_n = \left\{\sum_i X_i^* X_i^{*'} \hat{I}_i\right\}^{-1} \sum_i X_i^* Y_i^* \hat{I}_i, \qquad (3.8)$$

where $X_i^* = X_i - \hat{m}_X(Z_i)$ and $Y_i^* = Y_i - \hat{m}_Y(Z_i)$, $\hat{m}_X(Z_i)$ and $\hat{m}_Y(Z_i)$ are higher order kernel estimates of $E(X|Z=Z_i)$ and $E(Y|Z=Z_i)$, respectively, and $\hat{I}_i = 1(\hat{f}(Z_i) > b)$ where $\hat{f}(Z_i)$ is the corresponding density estimate of Z evaluated at Z_i, and b is a small number. Under regularity conditions,

$$\left\{\sum_i X_i^* X_i^{*\prime} \hat{I}_i\right\}^{-1/2} n^{1/2}(\hat{\beta}_n - \beta^0) \longrightarrow_d N(0, I_r).$$

The nonparametric estimates of the regression functions and the density functions can be computed with KEREGSY. Then any package may be used to compute $\hat{\beta}_n$ and its standard errors. For instance, suppose we have only one X and we have computed $\hat{m}_X(Z_i)$, $\hat{m}_Y(Z_i)$ and $\hat{f}(Z_i)$ using KEREGSY, storing the estimates in the file 'NONP.DAT'. The data, as usual, is in the file 'DATA.DAT'. Setting b= .0001, the following TSP program computes the semiparametric estimates,

```
read(file='data.dat')y,x;
read(file='nonp.dat')mx,my,fz;
select fz> .0001;
y=y-my;
x=x-mx;
olsq y x;
```

3.4. SEMIPARAMETRIC ESTIMATION AND INFERENCE BASED ON AVERAGED ESTIMATED.

Some semiparametric estimators use nonparametric estimates of the functional,

$$\mathring{f}(x) = \partial f(x)/\partial x, \tag{3.9}$$

where $f(x)$ is the density of X evaluated at x. It may be consistently estimated by

$$\hat{\mathring{f}}(x) = \partial \hat{f}(x)/\partial x, \tag{3.10}$$

where $\hat{f}(x)$ is the kernel estimate of $f(x)$ as defined in (2.20); i.e.

$$\hat{\mathring{f}}(x) = \sum_i K_i'(x),$$

where $K_i'(x) = \partial K_i(x)/\partial x$. Therefore, we can obtain estimates $\{(\hat{\mathring{f}}(X_i), \hat{\mathring{d}}_Y(X_i)),\ i=1,\ldots n\}$, where $\hat{\mathring{d}}_Y(x) = \sum_i Y_i K_i'(x)$, by using KEREGSY. Note that setting Σ_n to the unit matrix,

$$K_i(x) = (n\, h_n^{r+1})^{-1} K'(h_n^{-1}(x - X_i)),$$

where K'(.) is the derivative of the kernel function. Then, all what we have to do is to define the function FKER in KEREGSY as K'() and on exit we have to divide SE and FK by h_n.

Powell et. al. (1989) proposed an estimator of

$$\delta = E\{f(X)\ \partial g(X)/\partial X\},$$

where g(.) is an unknown function. It is interesting in a number of

econometric applications to limited dependent variable models where

$$E(Y|X=x)= G(x'\beta^0),$$

where G() is a function of unknown form and β^0 a rx1 vector of unknown parameters. In this case $\delta= c\beta^0$ where c is an unknown constant. Powell et. al. noted that, under mild regularity conditions,

$$\delta= -2\ E\{Y\ \mathring{f}(X)\}.$$

Then, they estimated δ by

$$\hat{\delta}_n= -2\ n^{-1}\ \sum_i Y_i\ \hat{\mathring{f}}(X_i). \tag{3.11}$$

Here $\hat{\mathring{f}}(X_i)$ does not use the own observation; i.e.

$$\hat{\mathring{f}}(X_i)= \sum_{j\neq i}\ K'_j(X_i).$$

This feature can be incorporated in KEREGSY by an slight modification in the main code. Powell et. al. proved, under regularity conditions, that

$$\hat{\Omega}_\delta^{-1/2}\ n^{1/2}\ (\hat{\delta}_n- E(\hat{\delta}_n)) \longrightarrow_d N(0,\ I_r),$$

where

$$\hat{\Omega}_\delta= 4\ n^{-1}\ \sum_i\ \hat{r}_i\hat{r}_i'\ -\ 4\ \hat{\delta}_n\hat{\delta}_n',$$

and $\hat{r}_i= n(n-1)^{-1}\ (\hat{\mathring{d}}_Y(X_i)- Y_i\hat{\mathring{f}}(X_i))$. Then we can compute everything with KREGSY. Powell et. al. also considered higher order kernels in order to avoid the asymptotic bias. They also proposed an instrumental variable estimate of $\delta^*= \delta/E(f(X))$. They noted that

$$\delta^*= \left\{E(\mathring{f}(X)\ X')\right\}^{-1}\ E(\mathring{f}(X)\ Y).$$

Then δ^* is estimated by

$$\hat{D}_n= \hat{\delta}_{Xn}^{-1}\ \hat{\delta}_n, \tag{3.12}$$

where $\hat{\delta}_{Xn}= \sum_i \hat{\mathring{f}}(X_i)X_i'$. Under regularity conditions,

$$\hat{\Omega}_D^{-1/2}\ n^{1/2}\ (\hat{\delta}_n- \delta^*) \longrightarrow_d N(0,\ I_r),$$

where $\hat{\Omega}_D= 4\ n^{-1}\ \sum_i\ \hat{r}_{Di}\hat{r}_{Di}'$ and $\hat{r}_{Di}= n(n-1)^{-1}\ (\hat{\mathring{d}}_U(X_i)- \hat{U}_i\hat{\mathring{f}}(X_i))$ where $\hat{U}_i= Y_i- X_i'\hat{D}_n$. Then $\hat{D}_n$ can be computed using the command INST in TSP, defining as instruments $\{\hat{\mathring{f}}(X_i),\ i=1,..,n\}$.

Robinson (1989) considered the following statistic, for testing conditional moment restrictions in nonparametric and semiparametric models for economic time series,

$$\hat{\lambda}_n = \hat{\tau}_n' \hat{\Phi}^{-1} \hat{\tau}_n, \qquad (3.13)$$

where

$$\hat{\tau}_n = n^{-1} \sum_i \begin{pmatrix} \hat{D}_{G^o}(X_i) \\ \hat{\mathring{d}}_G(X_i) \end{pmatrix},$$

where $\hat{D}_{G^o}(X_i) = \sum_j G^0(Y_j, X_j; Y_i, X_i)\, K_j(X_i)$ and $G()$ is a known rx1 function. We can compute $\hat{\mathring{d}}_G(X_i)$ directly with KEREGSY. However, the computation of $\hat{D}_{G^o}(X_i)$ will require a modification in the dimensions of the input array. We can of course run KEREGSY n times but it will be computationally expensive. On the other hand $\hat{\Phi}$ is the estimate of the asymptotic covariance matrix of $\hat{\tau}_n$. It may be computed by bootstrapping or by nonparametric methods. In particular, a possibility is,

$$\hat{\Phi} = (2\pi)^{-1} \sum_{j=-m}^{m} \ell(j/m)\, \hat{\Gamma}_j,$$

where $\ell(.)$ is a real 'lag window' function (e.g. the modified Barlett $\ell(u)=1-|u|$) and $\hat{\Gamma}_j = n^{-1} \sum_i c_i\, c_{i+j}'$ where $c_i = \sum_j (q_{ij} + q_{ji})$ and

$$q_{ij} = \begin{pmatrix} G^0(Y_j, X_j; Y_i, X_i)\, K_j(X_i) \\ G(Y_j, X_j)\, K_j'(X_i) \end{pmatrix}.$$

3.5. ASYMPTOTICALLY EFFICIENT ESTIMATION IN THE PRESENCE OF DISTURBANCE DISTRIBUTION OF UNKNOWN FORM

Bickel (1982) considered the linear regression model $E(Y|X=x) = x'\beta^0$ by the one step estimate,

$$\hat{\beta}_n = \tilde{\beta}_n - \left\{ \sum_i X_i X_i' \hat{\mathring{f}}(\tilde{U}_i)^2 \hat{f}(\tilde{U}_i)^{-2} \right\}^{-1} \sum_i X_i \hat{\mathring{f}}(\tilde{U}_i) \hat{f}(\tilde{U}_i)^{-1}. \qquad (3.15)$$

where $\tilde{\beta}_n$ is a preliminary root-n-consistent estimate, $\tilde{U}_i$ are the residuals computed from this estimate and $\hat{f}(\tilde{U}_i)$ and $\hat{\mathring{f}}(\tilde{U}_i)$ are kernel estimates of $f(U_i)$ and $\mathring{f}(U_i)$, the density of $U = Y - X'\beta^0$ and its derivative evaluated at U_i. Bickel (1982) proved, under regularity conditions, that

$$n^{1/2}(\hat{\beta}_n - \beta^0) \longrightarrow_d N\left(0, [E(X X' \mathring{f}(U)^2 f(U)^{-2})]^{-1}\right).$$

This result was proved before by Stone (1975) for the case where X= 1. Manski (1982) has extended this estimation method to nonlinear and simultaneous equations models and Lee (1990) to sample selection models.

The estimates $\{(\hat{f}(\tilde{U}_i)$ and $\hat{\mathring{f}}(\tilde{U}_i)),\ i=1,..,n\}$ can be obtained by KEREGSY and the second term in (3.14) by performing a regression of $e=(1,\ldots,1)'$

against $X_i \dot{\hat{f}}(\tilde{U}_i)\hat{f}(\tilde{U}_i)^{-1}$ using any econometric package.

3.6. LINEAR REGRESSION PARAMETER ESTIMATION CONSTRUCTED BY NONPARAMETRIC ESTIMATION

Faraldo Roca and González Manteiga (1985), Cristóbal Cristóbal et. al. (1989) and Stute and González Manteiga (1990) considered the estimation of the linear regression model $Y= X'\beta^0+ \varepsilon$ where $E(\varepsilon)= 0$, by the general class of estimators

$$\hat{\beta}_n= \text{argmin}_\beta \int (\hat{m}_Y(x) - x'\beta)^2 \, d\, \Omega_n(x), \tag{3.16}$$

where $\Omega_n(x)$ is a weighting function and $\hat{m}_Y(x)$ is a nonparametric estimate of $E(Y|X= x)$. They propose to use the weighting function,

$$\Omega_n(x)= \int_{-\infty}^{x} \hat{f}(t) \, d\, t,$$

where $\hat{f}(x)$ is the nonparametric estimate of the density of X evaluated at x. Then,

$$\hat{\beta}_n= \left\{\sum_i X_i X_i' \hat{f}(X_i)\right\}^{-1} \sum_i X_i \hat{d}_Y(X_i).$$

Faraldo Roca and González Manteiga (1985) and Cristóbal Cristóbal et. al. (1989) proved that $\hat{\beta}_n$ is to first order as efficient as ordinary least squares. They shown good performance of the biased $\hat{\beta}_n$ with respect to ordinary least squares when mean squared error (MSE) is used for comparison. It obviously requires a 'judicious choice' of the smoothing parameter. Faraldo Roca and González Manteiga (1985) calculated the optimal bandwidth using kernels which minimize the MSE of $\hat{\beta}_n$ in the one regressor case. For this choice of bandwidth the MSE of $\hat{\beta}_n$ is smaller than the variance of the ordinary least squares.

Note that once the nonparametric estimates have been computed using KEREGSY, one can use any econometric package in order to compute $\hat{\beta}_n$ by using any routine solving linear equation systems or regressing $\hat{m}_Y(X_i)\hat{f}(X_i)^{1/2}$ against $X_i\hat{f}(X_i)^{1/2}$.

3.6 ASYMPTOTICALLY EFFICIENT ESTIMATION IN THE PRESENCE OF AUTOCORRELATION OF UNKNOWN FORM

Hidalgo (1990) considered the linear regression model $Y_i= X_i'\beta^0+ U_i$ where $U_i= \rho(U_{i-1})+ \varepsilon_i$, $i=1,\ldots,n$ where $\rho(.)$ is an unknown function and ε_i are white noise. He proposed the estimate

$$\hat{\beta}_n= \text{argmin}_\beta \sum_{i=2}^{n} \left(Y_i- X_i'\beta- \hat{\rho}(Y_{i-1}- X_{i-1}'\beta)\right)^2 \hat{I}_{i-1}, \tag{3.16}$$

where

$$\hat{\rho}(U_{i-1}(\beta)) = (2nh)^{-1} \sum_{\substack{j=2 \\ j\neq i}} U_j \left\{K\left((U_{i-1}(\beta) - U_{j-1}(\beta))h^{-1}\right) - K\left((U_{i-1}(\beta) + U_{j-1}(\beta))h^{-1}\right)\right\} \hat{f}\left(U_{i-1}(\beta)\right)^{-1},$$

and $\hat{I}_{i-1} = 1\left(\hat{f}\left(U_{i-1}(\beta)\right) > b\right)$, where b is a small number chosen by the user and

$$\hat{f}\left(U_{i-1}(\beta)\right) = (2nh)^{-1} \sum_{\substack{j=2 \\ j\neq i}} U_j \left\{K\left((U_{i-1}(\beta) - U_{j-1}(\beta))h^{-1}\right) + K\left((U_{i-1}(\beta) + U_{j-1}(\beta))h^{-1}\right)\right\}$$

Hidalgo (1990) proved that, under regularity conditions,

$$n^{1/2}(\hat{\beta}_n - \beta^0) \longrightarrow_d N(0, V_o),$$

where $V_o = Var(\varepsilon)\ [\text{plim}\ n^{-1} \sum_{i=2}(X_i - X_{i-1}\rho'(U_{i-1}))(X_i - X_{i-1}\rho'(U_{i-1}))']^{-1}$ and $\rho'(.)$ is the derivative of $\rho(.)$.

The estimator in (3.1.6) seems difficult to compute. However, a linearized version of this estimate would permit to use our routines. In particular, Hidalgo (1990) suggested to use a Gauss-Newton one step estimate from the ordinary least squares, $\bar{\beta}_n$; i.e.

(3.18)

$$\hat{\beta}_n^{(1)} = \bar{\beta}_n - \left\{\sum_{i=2}(X_i - X_{i-1}\hat{\rho}'(\bar{U}_{i-1}))(X_i - X_{i-1}\hat{\rho}'(\bar{U}_{i-1}))\bar{I}_{i-1}\right\}^{-1} \times \sum_{i=2}(X_i - X_{i-1}\hat{\rho}'(\bar{U}_{i-1}))(\bar{U}_i - \hat{\rho}(\bar{U}_{i-1})\bar{I}_{i-1}$$

where $\bar{U}_i = Y_i - \bar{\beta}_n' X_i$, $\hat{\rho}'(\bar{U}_{i-1}) = \partial\hat{\rho}(\bar{U}_{i-1})/\partial U_{i-1}$ and $\bar{I}_{i-1} = 1\left(\hat{f}(\bar{U}_{i-1}) > b\right)$ where b is a small number.

Then $\hat{\rho}(\bar{U}_{i-1})$ and $\hat{\rho}'(\bar{U}_{i-1})$ can be computed using KEREGSY and the second term of (3.18) can be easily computed using TSP as indicated in section 3.3. A full iterated estimated can be obtained using any routine for nonlinear least squares.

ACKNOWLEDGEMENTS

I thank Peter Robinson for comments on a previous version of this manuscript. Financial support from the Economic and Social Research Council (ESRC), reference number: R00023411 is gratefully acknowledged. Soft copies of the routines are available from the author.

REFERENCES

Aguirre-Torres, V. and A.R. Gallant (1983), "The null and non-null distribution of the Cox test for multivariate nonlinear regression", Journal of Econometrics 21, 5-33.

Andrews, A.C., P.J. Bickel, F.R. Hampel, P.J. Huber, W.H. Rogers and W.J. Tukey (1972): Robust Estimates of Location: Surveys and Advances, Princeton University Press: Princeton.

Barlett, M.S. (1963): "Statistical estimation of density functions", Sankhya A 25, 145-154.

Bickel, P. (1982): "On adaptive estimation", Annals of Statistics 447-671.

Boente, G. and R. Fraiman (1989): "Robust nonparametric estimation for dependent observations", Annals of Statistics 17, 1242-1256.

Boente, G. and R.Fraiman (1990): "Asymptotic distribution of robust estimators for nonparametric models from mixing processes", Annals of Statistics 18, 891-906.

Broich, T., W. Härdle and A. Krause (1990): "XploRe- a computing environment for eXploratory Regression and Analysis", Springer-Verlag (forthcoming).

Brown, B.W. and R. Mariano (1984): "Residual-based procedures for prediction and estimation in a nonlinear simultaneous equation system", Econometrica 52, 321-343.

Carroll, R.J. (1982): "Adapting for heteroscedasticity in linear models", Annals of Statistics 10, 1224-1233.

Collomb, G. (1980): "Estimation de la régression par la métode des k points les plus proches avec noyau: quelques propiétés de convergence ponctuelle", Lecture Notes in Mathematics 831, 159-175.

Cristóbal Cristóbal, J.A., P. Faraldo Roca and W. González Manteiga (1987): "A class of linear regression parameter estimators constructed by nonparametric estimation", Annals of Statistics 15, 603-609.

Delgado, M.A. (1989a): "Asymptotically efficient fully iterative nonlinear weighted least squares in the presence of heteroskedasticity of unknown form", manuscript.

Delgado, M.A. (1989b): "Semiparametric generalized least squares estimation in the nultivariate nonlinear regression model", manuscript.

Delgado, M.A. (1990): "Bounded influence regression in the presence of heteroskedasticity of unknown form", manuscript.

Devroye, L. (1986): Non-Uniform Random Variate Generation, Springer-Verlag: New York.

Devroy, L. and C. Yuen (1981), "Inversion-with-correction for the computer generation of discrete random variables", manuscript.

Duan, N. (1983): "Smearing estimate: a nonparametric retransformation method", Journal of the American Statistical Association 78, 605-610.

Faraldo Roca, P. and W. González Manteiga (1985): "On efficiency of a new class of linear regression estimates obtained by preliminary non-parametric regression" in New Perspectives in Theoretical and

Applied Statistics, (M. Puri et. al. editors), Wiley: New York.

Friedman, J.H., F. Baskett and L.J. Shustek (1975), "An algorithm for finding nearest neighbors", IEE Transactions on Computers C-24, 1149-1158.

Gallant, A.R. (1975), "Seemingly unrelated nonlinear regression", Journal of Econometrics 3, 35-50.

Härdle, W. (1984): "Robust regression function estimation", Journal of Multivariate Analysis 14, 169-180.

Härdle, W. (1987): "Resistant smoothing using the fast Fourier transform", Statistical Algorithm 222, Applied Statistics 36, 104-111.

Härdle, W. (1990): Applied Nonparametric Regression, Cambridge University Press, Econometric Society Monographs.

Härdle, W. and A.B. Tsybakov (1990): "Robust nonparametric regression with simultaneous scale curve estimation", Annals of Statistics 16, 120-135.

Härdle, W. and J.S. Marron (1985): "Optimal bandwidth selection in nonparametric function estimation", Annals of Statistics 13, 1465-1481.

Healy, M.J.R. (1968a): "Triangular decomposition of a symmetric matrix", Statistical Algorithm 6, Applied Statistics 17, 195-197.

Healy, M.J.R. (1968b): "Inversion of a positive-semidefinite symmetric matrix", Statistical Algorithm 7, Applied Statistics 17, 198-199.

Hidalgo, F.J. (1990): "Adaptive semiparametric estimation in the presence of autocorrelation of unknown form", Journal of Time Series Analysis (forthcoming).

Hoare, C.A.R. (1962): "Quicksort", Computer Journal 5, 10-15.

Keleijan, H.H. (1974): "Instrumental variable estimation of nonlinear econometric models", manuscript.

Knuth, D.E. (1969): The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Addison-Wesley, Reading,Mass.

Knuth, D.E. (1973): The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading,Mass.

Lee L.F. (1990): Efficient semiparametric scoring estimation of sample selection models", manuscript.

Li, K.C. (1984): "Consistency of nearest neighbor estimates in non-parametric regression", Annals of Statistics 12, 230-240.

Manski, C.F. (1982), "Adaptive estimation of non-linear regression models", Econometric Reviews 3, 145-194.

McQueen, J.B. (1990): N-Kernel, Non-standard Statistical Software, Santa Monica.

Nadaraya, E.A. (1964), "On estimating regression", Theory of Probability and its Applications 9, 141-142.

Newey, W. (1987): "Efficient estimation of models with conditional moment restrictions", manuscript.

Newey, W. (1990): "Efficient instrumental variable estimation of nonlinear models", Econometrica 58, 809-837.

Powell J.L., J.H. Stock and T.M. Stoker (1989): "Semiparametric estimation of index coefficients", Econometrica 57, 1403-1430.

Prakasa Rao, B.L.S. (1983): Nonparametric Functional Estimation, Orlando: Academic Press.

Press, W., B.P. Flannery, S.A. Tenkolsky and W.T. Vetterling (1985): Numerical Recipes, Cambridge University Press: London.

Robinson, P.M. (1984): "Robust nonparametric autoregression", Lecture Notes in Statistics 26, 247-255.

Robinson, P.M. (1986): "Nonparametric methods in specification", The Economic Journal, Supplement 96, 134-141.

Robinson, P.M. (1987): "Asymptotically efficient estimation in the presence of heteroskedasticity of unknown form", Econometrica 55, 531-548.

Robinson, P.M. (1988a): "Root-n-consistent semiparametric regression", Econometrica 56, 931-954.

Robinson, P.M. (1988b): "Semiparametric econometrics: a survey", Journal of Applied Econometrics 3, 35-51.

Robinson, P.M. (1989): "Hypothesis testing in semiparametric and nonparametric models for econometric time series", Review of Economic Studies 56, 511-534.

Robinson, P.M. (1990): "Best nonlinear three-stage least squares of certain econometric models", Econometrica (forthcoming).

Silverman, B.W. (1982): "Kernel density estimation using the fast Fourier transform", Statistical Algorithm 175, Applied Statistics 31, 93-97.

Silverman, B.W. (1986): Density Estimation for Statistics and Data Analysis, Chapman and Hall: New York

Stone, C.J. (1975): "Adaptive maximum likelihood estimation of a location parameter", Annals of Statistics 3, 267-284.

Stone, C.J. (1977): "Consistent nonparametric regression" (with discussion), Annals of Statistics 5, 595-645.

Stute, W. and W. González Manteiga (1990) "Nearest neighbor smoothing in linear regression", Journal of Multivariate Analysis 34, 61-74.

Tsybakov, A.B. (1982): "Robust estimates of a function", Problems, Information and Transmission 18, 190-201.

Watson, G.S. (1964): "Smooth regression analysis", Sankhya A 26, 359-372.

Williams, J.W.J. (1964): "Heapsort", Algorithm 232, Communications of the ACM 7, 347-348.

```
c
c     The function GAMMLN is in Numerical Recipes pp. 157 (Press et. al 1986)
c     Any other function computing the log of the gamma function can be used.
c
      subroutine knnre(x,y,nobs,nvar,knn,io1,aio2,io3,se,scl,iw,irk,
     *  ws,ns,tol,maxit,robf,wf)
      integer iw(nobs,*),irk(nobs,*),ns(*)
      real x(nobs,*),scl(nvar),ws(*),y(*),se(*)
      data pi/.56418958350d0/
      if(knn.gt.nobs)pause 'knn>nobs'
      if(nvar.gt.nobs)pause 'nvar>nobs'
      obs=real(nobs)
      var=real(nvar)
      obs1=obs-1.
      do 2,j=1,nvar
      q=0.
      qq=0.
      do 3,i=1,nobs
3     q=x(i,j)+q
      q=q/obs
      do 4,i=1,nobs
      c=(x(i,j)-q)
4     qq=(c*c)+qq
      if(qq.eq.0.)pause 'one x is a constant'
2     scl(j)=sqrt(qq/obs1)
      ok=float(knn)
      var1=var/2.
      var2=1./var
      var3=(2.*obs)**(1.-var2)
      cmax=pi*var3*((ok*var*exp(gammln(var1)))**var2)
      icmax=nint(cmax/2.)
      if(icmax.ge.nobs/2)icmax=(nobs/2)-2
      call sort1(nobs,nvar,x,iw)
      do 40,i=1,nobs
      do 40,j=1,nvar
40    irk(iw(i,j),j)=i
      do 12,i=1,nobs
      q=0.
      do 20,j=1,nvar
      l=irk(i,j)
      if(l.le.icmax)then
      k=iw(l+icmax,j)
      c=abs(x(k,j)-x(iw(1,j),j))
      else if(l.ge.nobs-icmax)then
      k=iw(l-icmax,j)
      c=abs(x(k,j)-x(iw(nobs-icmax,j),j))
      else
      k=iw(l-icmax,j)
      n=iw(l+icmax,j)
      c=abs(x(k,j)-x(n,j))
      endif
      if(c.gt.q)then
      q=c
      is=j
      endif
20    continue
c     is=1
      l=irk(i,is)
      if(l.lt.nobs)then
      if(l.gt.1)then
```

```
        call midd(nobs,nvar,x,iw,l,is,knn,scl,ws,ns,io3)
        else
        l=iw(2,is)
        ik=1
        ik0=1
        i0=1
        zer=0.0d0
        call right(nobs,nvar,x,iw,i0,is,knn,scl,ws,ns,ik,ik0,zer,l,io3)
        endif
        else
        k=iw(nobs-1,is)
        ik=1
        ik0=1
        zer=0.
        call left(nobs,nvar,x,iw,nobs,is,knn,scl,ws,ns,ik,ik0,zer,k,io3)
        endif
        cc=0.
        do 466,j=1,knn-io1
        l=ns(j)
        k=j+io1
466     cc=cc+y(l)*wf(k,knn)
        se(i)=cc
        if(aio2.lt.0.)then
        cm=se(i)
        ii=0
100     cc=0.
        bb=0.
        do 10,j=1,knn-io1
        l=ns(j)
        k=io1+j
        st=y(l)-cm
        if(st.eq.0.)st=10e-07
        ct=robf(st)/st
        cc=cc+ct*wf(k,knn)
10      bb=bb+(y(l)*ct*wf(k,knn))
        ss=0.
        if(cc.gt.0.)ss=bb/cc
        if(abs(ss-cm).gt.tol.and.ii.le.maxit)then
        cm=ss
        ii=ii+1
        goto 100
        else
        se(i)=ss
        endif
        else if(aio2.gt.0.)then
        dd=-10.0e+13
        ee=10.e+13
        do 101,m=1,nobs
        cc=0.
        aa=0.
        do 111,j=1,knn-io1
        l=ns(j)
        k=j+io1
111     if(y(l).le.y(m))cc=cc+wf(k,knn)
        if(y(m).gt.dd.and.sngl(aio2).ge.sngl(cc))dd=y(m)
        if(y(m).lt.ee.and.sngl(cc).ge.sngl(aio2))ee=y(m)
101     continue
        se(i)=(dd+ee)/2.
        endif
12      continue
        return
        end
```

```
c
      subroutine midd(nobs,nvar,x,iw,i,is,knn,scl,ws,ns,io3)
      integer iw(nobs,nvar),ns(knn)
      real scl(nvar),ws(knn),x(nobs,nvar)
      ik1=1
      ik2=1
      l1=iw(i+1,is)
      l2=iw(i-1,is)
      k=iw(i,is)
      q=x(k,is)
      a=scl(is)
      r1=dist1(q,x(l1,is),a,io3)
      r2=dist1(q,x(l2,is),a,io3)
      do 10,ik=1,knn
      if(r1.lt.r2)then
      ws(ik)=dist2(x,nobs,nvar,k,l1,scl,io3)
      ns(ik)=l1
      ik1=ik1+1
      if(i+ik1.gt.nobs)then
      ik0=ik+1
      call left(nobs,nvar,x,iw,i,is,knn,scl,ws,ns,ik0,ik2,r2,l2,io3)
      return
      else
      l1=iw(i+ik1,is)
      r1=dist1(q,x(l1,is),a,io3)
      endif
      else
      ws(ik)=dist2(x,nobs,nvar,k,l2,scl,io3)
      ns(ik)=l2
      ik2=ik2+1
      if(ik2.eq.i)then
      ik0=ik+1
      call right(nobs,nvar,x,iw,i,is,knn,scl,ws,ns,ik0,ik1,r1,l1,io3)
      return
      else
      l2=iw(i-ik2,is)
      r2=dist1(q,x(l2,is),a,io3)
      endif
      endif
10    continue
      call sort2(ws,ns,knn)
      do 11,ik=knn+1,nobs
      if(r1.lt.r2)then
      if(ws(knn).lt.r1)return
      qq=dist2(x,nobs,nvar,k,l1,scl,io3)
      if(qq.lt.ws(knn))then
      ws(knn)=qq
      ns(knn)=l1
      call sort3(ws,ns,knn)
      endif
      ik1=ik1+1
      if(i+ik1.gt.nobs)goto 400
      l1=iw(i+ik1,is)
      r1=dist1(q,x(l1,is),a,io3)
      goto 11
400   call left(nobs,nvar,x,iw,i,is,knn,scl,ws,ns,ik,ik2,r2,l2,io3)
      return
      else
      if(ws(knn).lt.r2)return
      qq=dist2(x,nobs,nvar,k,l2,scl,io3)
      if(qq.lt.ws(knn))then
      ws(knn)=qq
```

```
      ns(knn)=l2
      call sort3(ws,ns,knn)
      endif
      ik2=ik2+1
      if(i.eq.ik2)goto 200
      l2=iw(i-ik2,is)
      r2=dist1(q,x(l2,is),a,io3)
      goto 11
200   call right(nobs,nvar,x,iw,i,is,knn,scl,ws,ns,ik,ik1,r1,l1,io3)
      return
      endif
11    continue
      return
      end
c
      subroutine left(nobs,nvar,x,iw,i,is,knn,scl,ws,ns,ik0,ik,r1,
     * l,io3)
      integer iw(nobs,nvar),ik,is,i,knn,ns(knn)
      real scl(nvar),ws(knn),x(nobs,nvar)
      k=iw(i,is)
      a=scl(is)
      q=x(k,is)
      if(ik0.gt.knn)goto 11
      if(ik0.eq.knn)goto 12
10    ws(ik0)=dist2(x,nobs,nvar,k,l,scl,io3)
      ns(ik0)=l
      ik=ik+1
      l=iw(i-ik,is)
      ik0=ik0+1
      if(ik0.le.knn)goto 10
      r1=dist1(q,x(l,is),a,io3)
12    call sort2(ws,ns,knn)
      if(i.le.ik)return
11    if(ws(knn).lt.r1)return
      qq=dist2(x,nobs,nvar,k,l,scl,io3)
      if(qq.lt.ws(knn))then
      ws(knn)=qq
      ns(knn)=l
      call sort3(ws,ns,knn)
      endif
      ik=ik+1
      if(i.le.ik)return
      l=iw(i-ik,is)
      r1=dist1(q,x(l,is),a,io3)
      if(ik.lt.i)goto 11
      return
      end
c
      subroutine right(nobs,nvar,x,iw,i,is,knn,scl,ws,ns,ik0,ik,r1,
     * l,io3)
      integer iw(nobs,nvar),ns(knn)
      real scl(nvar),ws(knn),x(nobs,nvar)
      a=scl(is)
      k=iw(i,is)
      q=x(k,is)
      if(ik0.gt.knn)goto 11
      if(ik0.eq.knn)goto 12
10    ns(ik0)=l
      ws(ik0)=dist2(x,nobs,nvar,k,l,scl,io3)
      ik=ik+1
      l=iw(ik+i,is)
      ik0=ik0+1
```

```
        if(ik0.le.knn)goto 10
        r1=dist1(q,x(l,is),a,io3)
12      call sort2(ws,ns,knn)
        if(i+ik.gt.nobs)return
11      if(ws(knn).lt.r1)return
        qq=dist2(x,nobs,nvar,k,l,scl,io3)
        if(qq.lt.ws(knn))then
        ws(knn)=qq
        ns(knn)=l
        call sort3(ws,ns,knn)
        endif
        ik=ik+1
        if(i+ik.gt.nobs)return
        l=iw(i+ik,is)
        r1=dist1(q,x(l,is),a,io3)
        goto 11
        end
c
        subroutine sort1(n,m,arrin,indx)
c
c    sort the columns of arrin and keep the sorting in an index
c
        integer n,m,indx(n,m)
        real arrin(n,m)
        if(n.eq.1)return
        do 1,k=1,m
        do 11,j=1,n
11      indx(j,k)=j
        l=n/2+1
        ir=n
10      continue
        if(l.gt.1)then
        l=l-1
        indxt=indx(l,k)
        q=arrin(indxt,k)
        else
        indxt=indx(ir,k)
        q=arrin(indxt,k)
        indx(ir,k)=indx(1,k)
        ir=ir-1
        if(ir.eq.1)then
        indx(1,k)=indxt
        go to 1
        endif
        endif
        i=l
        j=l+l
20      if(j.le.ir)then
        if(j.lt.ir)then
        if(arrin(indx(j,k),k).lt.arrin(indx(j+1,k),k))j=j+1
        endif
        if(q.lt.arrin(indx(j,k),k))then
        indx(i,k)=indx(j,k)
        i=j
        j=j+j
        else
        j=ir+1
        endif
        go to 20
        endif
        indx(i,k)=indxt
        go to 10
```

```
1       continue
        return
        end
c
        subroutine sort2(x,m,n)
c
c        sort a vector x and keep the index
c
        integer m(n)
        real x(n)
        if(n.eq.1)return
        l=n/2+1
        ir=n
10      continue
        if(l.gt.1)then
        l=l-1
        ra=x(l)
        ib=m(l)
        else
        ra=x(ir)
        ib=m(ir)
        x(ir)=x(1)
        m(ir)=m(1)
        ir=ir-1
        if(ir.eq.1)then
        x(1)=ra
        m(1)=ib
        return
        endif
        endif
        i=l
        j=l+l
20      if(j.le.ir)then
        if(j.lt.ir)then
        if(x(j).lt.x(j+1))j=j+1
        endif
        if(ra.lt.x(j))then
        x(i)=x(j)
        m(i)=m(j)
        i=j
        j=j+j
        else
        j=ir+1
        endif
        goto 20
        endif
        x(i)=ra
        m(i)=ib
        goto 10
        end
c
        subroutine sort3(x,m,n)
c
c        sort by straight insertion vectors x and m, with respect to x
c        where the last n-1 numbers are already sorted and the first not.
c
        integer m(n)
        real x(n)
        if(n.eq.1)return
        a=x(n)
        j=m(n)
        do 1,i=n-1,1,-1
```

```
        if(x(i).le.a)goto 10
        m(i+1)=m(i)
1       x(i+1)=x(i)
        i=0
10      x(i+1)=a
        m(i+1)=j
        return
        end
c
        function dist1(x,y,a,io3)
c
c        calculate the scaled distance between points x and y with scale a
c
        dist1=(x-y)/a
        if(io3.eq.0)then
        dist1=dist1*dist1
        else
        dist1=abs(dist1)
        endif
        return
        end
c
        function dist2(x,nobs,nvar,k,l,scl,io3)
c
c        calculate the full distance between the points k and l
c
        real x(nobs,*),scl(*)
        if(io3.eq.0)then
        c=0.
        do 1,i=1,nvar
        cc=(x(k,i)-x(l,i))/scl(i)
1       c=(cc*cc)+c
        dist2=c
        else if(io3.lt.0)then
        c=0.
        do 2,i=1,nvar
        cc=(x(k,i)-x(l,i))/scl(i)
2       c=abs(cc)+c
        dist2=c
        else
        c=0.
        do 3,i=1,nvar
        cc=(x(k,i)-x(l,i))/scl(i)
        cc=abs(cc)
        if(cc.ge.c)c=cc
3       dist2=c
        endif
        return
        end
```

SUBROUTINE KREGSY

```
c
c     The subroutines CHOL and SYMINV are the Hoaley (1968a and b) algorithms
c     AS6 and AS7 of the Royal Statistical Society.
c     Define the dimensions in CHOL and SYMINV as implicit (*).
c
      subroutine keregsy(y,x,nobs,nvar,hb,io1,io2,b,se,fk,w,fker)
      real x(nobs,*),y(*),se(*),fk(*),b(*),w(*)
      r1=float(nobs)
      r2=r1-1.
      if(io1.eq.0)then
```

```
      do 1,j=1,nvar
      c=0.
      do 2,i=1,nobs
2     c=c+x(i,j)
1     w(j)=c/r1
      if(io2.eq.0)then
      do 3,j=1,nvar
      c=0.
      do 4,i=1,nobs
4     c=c+x(i,j)*x(i,j)
      c=c/r2
3     b(j)=c-(r1*w(j)*w(j)/r2)
      else
      k=0
      do 5,j=1,nvar
      do 5,l=1,j
      c=0.
      k=k+1
      do 6,i=1,nobs
6     c=c+x(i,j)*x(i,l)
      c=c/r2
5     b(k)=c-(r1*w(j)*w(l)/r2)
      endif
      endif
      if(io2.eq.0)then
      do 7,i=1,nvar
7     b(i)=sqrt(b(i))*hb
      r2=1.
      do 8,i=1,nvar
8     r2=r2*b(i)
      r1=r1*r2
      else
      call syminv(b,nvar,b,w,nul,ifa)
      if(ifa.eq.2)pause 'singular weighting matrix'
      call chol(b,nvar,fk,nul,ifa,det)
      r1=r1*(hb**nvar)/sqrt(det)
      mm=nvar*(nvar+1)/2
      l=0
      do 20,i=1,nvar
      do 20,j=i-1,nvar-1
      l=l+1
      k=i+(j*(j+1)/2)
20    b(l)=fk(k)
      do 9,i=1,mm
9     b(i)=b(i)/hb
      endif
      do 11,i=1,nobs
      se(i)=0.
11    fk(i)=0.
      if(io2.eq.0)then
      do 12,i=1,nobs-1
      do 12,j=i+1,nobs
      do 13,k=1,nvar
13    w(k)=(x(i,k)-x(j,k))/b(k)
      r2=fker(w,nvar)
      d1=r2*y(j)
      d2=r2*y(i)
      fk(i)=fk(i)+r2
      se(i)=se(i)+d1
      fk(j)=fk(j)+r2
      se(j)=se(j)+d2
12    continue
```

```
      else
      do 14,i=1,nobs-1
      do 14,j=i,nobs
      m=0
      do 15,l=1,nvar
      r2=0.
      do 16,k=1,nvar
      m=m+1
16    r2=r2+(x(i,k)-x(j,k))*b(m)
15    w(l)=r2
      r2=fker(w,nvar)
      d1=r2*y(j)
      d2=r2*y(i)
      fk(i)=fk(i)+r2
      se(i)=se(i)+d1
      fk(j)=fk(j)+r2
14    se(j)=se(j)+d2
      endif
      do 17,i=1,nvar
17      w(i)=0.
      rk=fker(w,nvar)
c
c     if the own observation is not considered set rk=0.
c
      do 18,i=1,nobs
      se(i)=se(i)+y(i)*rk
      fk(i)=fk(i)+rk
      se(i)=se(i)/r1
18    fk(i)=fk(i)/r1
      return
      end
```

SUBROUTINE KEREGG

```
      subroutine keregg(y,x,nobs,nvar,hb,io1,io2,aio2,b,se,fk,w,
     ;  fker,robf,tol,maxit)
      real x(nobs,*),y(*),se(*),fk(nobs),b(*),w(*)
      r1=float(nobs)
      r2=r1-1.
      if(io1.eq.0)then
      do 1,j=1,nvar
      c=0.
      do 2,i=1,nobs
2     c=c+x(i,j)
1     w(j)=c/r1
      if(io2.eq.0)then
      do 3,j=1,nvar
      c=0.
      do 4,i=1,nobs
4     c=c+x(i,j)*x(i,j)
      c=c/r2
3     b(j)=c-(r1*w(j)*w(j)/r2)
      else
      k=0
      do 5,j=1,nvar
      do 5,l=1,j
      c=0.
      k=k+1
      do 6,i=1,nobs
6     c=c+x(i,j)*x(i,l)
      c=c/r2
5     b(k)=c-(r1*w(j)*w(l)/r2)
```

```
        endif
        endif
        if(io2.eq.0)then
        do 7,i=1,nvar
 7      b(i)=sqrt(b(i))*hb
        r2=1.
        do 8,i=1,nvar
 8      r2=r2*b(i)
        r1=r1*r2
        else
        call syminv(b,nvar,b,w,nul,ifa)
        if(ifa.eq.2)pause 'singular weighting matrix'
        call chol(b,nvar,se,nul,ifa,det)
        r1=r1*(hb**nvar)/sqrt(det)
        mm=nvar*(nvar+1)/2
        l=0
        do 20,i=1,nvar
        do 20,j=i-1,nvar-1
        l=l+1
        k=i+(j*(j+1)/2)
20      b(l)=se(k)
        do 9,i=1,mm
9       b(i)=b(i)/hb
        endif
        do 17,i=1,nobs
        if(io2.eq.0)then
        do 12,j=1,nobs
        do 13,k=1,nvar
13      w(k)=(x(i,k)-x(j,k))/b(k)
12      fk(j)=fker(w,nvar)
        else
        do 14,j=1,nobs
        m=0
        do 15,l=1,nvar
        r2=0.
        do 16,k=1,nvar
        m=m+1
16      r2=r2+(x(i,k)-x(j,k))*b(m)
15      w(l)=r2
14      fk(j)=fker(w,nvar)
        endif
        cc=0.
        do 11,j=1,nobs
11      cc=cc+fk(j)
        do 21,j=1,nobs
21      fk(j)=fk(j)/cc
        cc=0.
        do 22,j=1,nobs
22      cc=cc+fk(j)*y(j)
        se(i)=cc
        if(aio2.lt.0.)then
        cm=se(i)
        ii=0
100     cc=0.
        bb=0.
        do 10,j=1,nobs
        st=y(j)-cm
        if(st.eq.0.)st=10e-07
        ct=robf(st)/st
        cc=cc+ct*fk(j)
10      bb=bb+(y(j)*ct*fk(j))
        ss=0.
```

```
      if(cc.gt.0.)ss=bb/cc
      if(abs(ss-cm).gt.tol.and.ii.le.maxit)then
      cm=ss
      ii=ii+1
      goto 100
      else
      se(i)=ss
      endif
      else if(aio2.gt.0.)then
      dd=-10.0e+13
      ee=10.e+13
      do 101,m=1,nobs
      cc=0.
      do 111,j=1,nobs
111   if(y(j).le.y(m))cc=cc+fk(j)
      if(y(m).gt.dd.and.sngl(aio2).ge.sngl(cc))dd=y(m)
      if(y(m).lt.ee.and.sngl(cc).ge.sngl(aio2))ee=y(m)
101   continue
      se(i)=(dd+ee)/2.
      endif
17    continue
      return
      end
```

SUBROUTINE KERNN

```
c
c     This subroutine calls the same subroutines  and  functions as KNNRE
c
      subroutine kernn(x,y,nobs,nvar,knn,aio2,io3,se,fk,scl,
     *  iw,irk,ws,ns,w,fker,robf,maxit,tol)
      integer iw(nobs,*),irk(nobs,*),ns(*)
      real x(nobs,*),scl(*),ws(*),y(*),se(*),fk(*),w(*)
      data pi/.5641895835/
      obs=real(nobs)
      var=real(nvar)
      obs1=obs-1.
      do 2,j=1,nvar
      q=0.
      qq=0.
      do 3,i=1,nobs
3     q=x(i,j)+q
      q=q/obs
      do 4,i=1,nobs
      c=(x(i,j)-q)
4     qq=(c*c)+qq
2     scl(j)=sqrt(qq/obs1)
      ok=float(knn)
      var1=var/2.
      var2=1./var
      var3=(2.*obs)**(1.-var2)
      cmax=pi*var3*((ok*var*exp(gammln(var1)))**var2)
      icmax=nint(cmax/2.)
      if(icmax.ge.nobs/2)icmax=(nobs/2)-2
      call sort1(nobs,nvar,x,iw)
      do 40,i=1,nobs
      do 40,j=1,nvar
40    irk(iw(i,j),j)=i
      do 12,i=1,nobs
      q=0.
      do 20,j=1,nvar
      l=irk(i,j)
      if(l.le.icmax)then
```

```
      k=iw(l+icmax,j)
      c=abs(x(k,j)-x(iw(1,j),j))
      else if(l.ge.nobs-icmax)then
      k=iw(l-icmax,j)
      c=abs(x(k,j)-x(iw(nobs-icmax,j),j))
      else
      k=iw(l-icmax,j)
      n=iw(l+icmax,j)
      c=abs(x(k,j)-x(n,j))
      endif
      if(c.gt.q)then
      q=c
      is=j
      endif
20    continue
c     is=1
      l=irk(i,is)
      if(l.lt.nobs)then
      if(l.gt.1)then
      call midd(nobs,nvar,x,iw,l,is,knn,scl,ws,ns,io3)
      else
      l=iw(2,is)
      ik=1
      ik0=1
      i0=1
      zer=0.0d0
      call right(nobs,nvar,x,iw,i0,is,knn,scl,ws,ns,ik,ik0,zer,l,io3)
      endif
      else
      k=iw(nobs-1,is)
      ik=1
      ik0=1
      zer=0.0d0
      call left(nobs,nvar,x,iw,nobs,is,knn,scl,ws,ns,ik,ik0,zer,k,io3)
      endif
      do 17,j=1,nobs
      do 13,k=1,nvar
13    w(k)=(x(i,k)-x(j,k))/ws(knn)
17    fk(j)=fker(w,nvar)
      cc=0.
      do 11,j=1,nobs
11    cc=cc+fk(j)
      do 21,j=1,nobs
21    fk(j)=fk(j)/cc
      cc=0.
      do 22,j=1,nobs
22    cc=cc+fk(j)*y(j)
      se(i)=cc
      if(aio2.lt.0.)then
      cm=se(i)
      ii=0
100   cc=0.
      bb=0.
      do 101,j=1,nobs
      st=y(j)-cm
      if(st.eq.0.)st=10e-07
      ct=robf(st)/st
      cc=cc+ct*fk(j)
101   bb=bb+(y(j)*ct*fk(j))
      ss=0.
      if(cc.gt.0.)ss=bb/cc
      if(abs(ss-cm).gt.tol.and.ii.le.maxit)then
```

```
        cm=ss
        ii=ii+1
        goto 100
        else
        se(i)=ss
        endif
        else if(aio2.gt.0.)then
        dd=-10.0e+13
        ee=10.e+13
        do 102,m=1,nobs
        cc=0.
        do 111,j=1,nobs
111     if(y(j).le.y(m))cc=cc+fk(j)
        if(y(m).gt.dd.and.sngl(aio2).ge.sngl(cc))dd=y(m)
102     if(y(m).lt.ee.and.sngl(cc).ge.sngl(aio2))ee=y(m)
        se(i)=(dd+ee)/2.
        endif
12      continue
        return
        end
```

SUBROUTINE RESA

```
c
c       The funcion RAN3 is the portable random generator in Numerical Recipes
c       pp. 199. Any other generator for Uniforms (0,1) can be used.
c       The function GAMMLN is in Numerical Recipes pp. 156. Any other function
c       computing the log of the gamma function can be used.
c
        subroutine resa(nobs,nvar,mx,theta,npar,se,io,tfun)
        real theta(*),se(*)
        ii=-int(secnds(0.))-1
        if(io.lt.0)then
        do 20,i=1,nobs
        n=nobs
        k=mx
        it=ii+i
        iy=0
        ix=0
        c=0.
200     if(k.lt.n)then
        u=ran3(it)
        a=float(k)
        x=-log(1.-u)
        x=tanh(x/(2.*a))
        x=x/(1.+x)
        b=float(n+1)
        a=b-a
        x=x*a
        ix=int(x)+1
        t=-gammln(b)
        x=float(ix)
        b=b-x
        t=t+gammln(b)
        t=t+gammln(a)
        x=a-x
        t=exp(t-gammln(x))
        goto 25
100     t=t*float(n-k-ix)/float(n-ix)
        ix=ix+1
25      if(1-u.lt.t)goto 100
        else
```

```
      ix=1
      endif
      k=k-1
      n=n-ix
      iy=ix+iy
      c=c+tfun(nobs,nvar,theta,npar,i,iy)
      if(k.gt.0)goto 200
20    se(i)=c/float(mx)
      else if(io.gt.0)then
      ii=int(secnds(0.))+1
      on=float(nobs)
      do 21,i=1,nobs
      it=ii+i
      c=0.
      do 500,l=1,mx
      u=ran3(it)
      iy=int(on*u)+1
500   c=c+tfun(nobs,nvar,theta,npar,i,iy)
21    se(i)=c/float(mx)
      else
      do 22,i=1,nobs
      c=0.
      do 23,iy=1,mx
23    c=c+tfun(nobs,nvar,theta,npar,i,iy)
22    se(i)=c/float(mx)
      endif
      return
      end
```