np: A Package for Nonparametric Kernel Smoothing with Mixed Datatypes

Jeffrey S. Racine   Tristen Hayfield

Department of Economics
and
Department of Mathematics & Statistics
McMaster University
Hamilton, ON Canada L8S 4M4

Friday, June 16, 2006

The np package

- The np package implements a variety of recently developed kernel methods that seamlessly handle the mix of continuous, unordered, and ordered factor datatypes often found in applied settings
- The package also allows the user to create their own routines using high-level function calls
- The underlying library is based on the N © library which is written in ANSI C
- The underlying code is MPI aware
- The design philosophy underlying np is simply to provide an intuitive, flexible, and extensible environment for applied kernel estimation

The np package

- The np package implements a variety of recently developed kernel methods that seamlessly handle the mix of continuous, unordered, and ordered factor datatypes often found in applied settings
- The package also allows the user to create their own routines using high-level function calls
- The underlying library is based on the N © library which is written in ANSI C
- The underlying code is MPI aware
- The design philosophy underlying np is simply to provide an intuitive, flexible, and extensible environment for applied kernel estimation

The np package

- The np package implements a variety of recently developed kernel methods that seamlessly handle the mix of continuous, unordered, and ordered factor datatypes often found in applied settings
- The package also allows the user to create their own routines using high-level function calls
- The underlying library is based on the N © library which is written in ANSI C
- The underlying code is MPI aware
- The design philosophy underlying np is simply to provide an intuitive, flexible, and extensible environment for applied kernel estimation

The np package

- The np package implements a variety of recently developed kernel methods that seamlessly handle the mix of continuous, unordered, and ordered factor datatypes often found in applied settings
- The package also allows the user to create their own routines using high-level function calls
- The underlying library is based on the N © library which is written in ANSI C
- The underlying code is MPI aware
- The design philosophy underlying np is simply to provide an intuitive, flexible, and extensible environment for applied kernel estimation

The np package

- The np package implements a variety of recently developed kernel methods that seamlessly handle the mix of continuous, unordered, and ordered factor datatypes often found in applied settings
- The package also allows the user to create their own routines using high-level function calls
- The underlying library is based on the N © library which is written in ANSI C
- The underlying code is MPI aware
- The design philosophy underlying np is simply to provide an intuitive, flexible, and extensible environment for applied kernel estimation

Workflow in np

- np handles different datatypes via the `data.frame()`, which preserves a variable's type once it has been cast (unlike `cbind()`)
- You create a data frame casting data according to type (continuous, `factor()`, `ordered()`), e.g.,
  - `data(Italy)`
  - `attach(Italy)`
  - `data <- data.frame(ordered(year),gdp)`
- Next, you typically proceed as follows:

Workflow in np

- np handles different datatypes via the `data.frame()`, which preserves a variable's type once it has been cast (unlike `cbind()`)
- You create a data frame casting data according to type (continuous, `factor()`, `ordered()`), e.g.,
  - `data(Italy)`
  - `attach(Italy)`
  - `data <- data.frame(ordered(year),gdp)`
- Next, you typically proceed as follows:

Workflow in np

- np handles different datatypes via the `data.frame()`, which preserves a variable's type once it has been cast (unlike `cbind()`)
- You create a data frame casting data according to type (continuous, `factor()`, `ordered()`), e.g.,
  - `data(Italy)`
  - `attach(Italy)`
  - `data <- data.frame(ordered(year),gdp)`
- Next, you typically proceed as follows:

Workflow in np

- np handles different datatypes via the `data.frame()`, which preserves a variable's type once it has been cast (unlike `cbind()`)
- You create a data frame casting data according to type (continuous, `factor()`, `ordered()`), e.g.,
  - `data(Italy)`
  - `attach(Italy)`
  - `data <- data.frame(ordered(year),gdp)`
- Next, you typically proceed as follows:

Workflow in np

- np handles different datatypes via the `data.frame()`, which preserves a variable's type once it has been cast (unlike `cbind()`)
- You create a data frame casting data according to type (continuous, `factor()`, `ordered()`), e.g.,
  - `data(Italy)`
  - `attach(Italy)`
  - `data <- data.frame(ordered(year),gdp)`
- Next, you typically proceed as follows:

Workflow in np

- np handles different datatypes via the `data.frame()`, which preserves a variable's type once it has been cast (unlike `cbind()`)
- You create a data frame casting data according to type (continuous, `factor()`, `ordered()`), e.g.,
  - `data(Italy)`
  - `attach(Italy)`
  - `data <- data.frame(ordered(year),gdp)`
- Next, you typically proceed as follows:
  - Compute appropriate bandwidths
  - Estimate an object
  - Alternately, plot the object via `np.plot()`

Workflow in np

- np handles different datatypes via the `data.frame()`, which preserves a variable's type once it has been cast (unlike `cbind()`)
- You create a data frame casting data according to type (continuous, `factor()`, `ordered()`), e.g.,
  - `data(Italy)`
  - `attach(Italy)`
  - `data <- data.frame(ordered(year),gdp)`
- Next, you typically proceed as follows:
  - Compute appropriate bandwidths
  - Estimate an object
  - Alternately, plot the object via `np.plot()`

np: A Package for Nonparametric Kernel Smoothing with Mixed Datatypes
└np: Kernel Smoothing with Mixed Datatypes

Workflow in np

- np handles different datatypes via the `data.frame()`, which preserves a variable's type once it has been cast (unlike `cbind()`)
- You create a data frame casting data according to type (continuous, `factor()`, `ordered()`), e.g.,
  - `data(Italy)`
  - `attach(Italy)`
  - `data <- data.frame(ordered(year),gdp)`
- Next, you typically proceed as follows:
  - Compute appropriate bandwidths
  - Estimate an object
  - Alternately, plot the object via `np.plot()`

np: A Package for Nonparametric Kernel Smoothing with Mixed Datatypes
└np: Kernel Smoothing with Mixed Datatypes

Workflow in np

- np handles different datatypes via the `data.frame()`, which preserves a variable's type once it has been cast (unlike `cbind()`)
- You create a data frame casting data according to type (continuous, `factor()`, `ordered()`), e.g.,
  - `data(Italy)`
  - `attach(Italy)`
  - `data <- data.frame(ordered(year),gdp)`
- Next, you typically proceed as follows:
  - Compute appropriate bandwidths
  - Estimate an object
  - Alternately, plot the object via `np.plot()`

np: A Package for Nonparametric Kernel Smoothing with Mixed Datatypes
└np: Kernel Smoothing with Mixed Datatypes

Workflow in np

- We have tried to make np sufficiently flexible to be of use to a wide range of users
- All options can be tweaked by the user (kernel function, kernel order, bandwidth type, estimator type and so forth)
- One function, `np.kernelsum()`, allows you to create your own estimators, tests, etc.
- The function `np.kernelsum()` is simply a call to highly optimized C code, so you get the benefits of compiled code with the flexibility of R

Workflow in np

- We have tried to make np sufficiently flexible to be of use to a wide range of users
- All options can be tweaked by the user (kernel function, kernel order, bandwidth type, estimator type and so forth)
- One function, `np.kernelsum()`, allows you to create your own estimators, tests, etc.
- The function `np.kernelsum()` is simply a call to highly optimized C code, so you get the benefits of compiled code with the flexibility of R

Workflow in np

- We have tried to make np sufficiently flexible to be of use to a wide range of users
- All options can be tweaked by the user (kernel function, kernel order, bandwidth type, estimator type and so forth)
- One function, `np.kernelsum()`, allows you to create your own estimators, tests, etc.
- The function `np.kernelsum()` is simply a call to highly optimized C code, so you get the benefits of compiled code with the flexibility of R

Workflow in np

- We have tried to make np sufficiently flexible to be of use to a wide range of users
- All options can be tweaked by the user (kernel function, kernel order, bandwidth type, estimator type and so forth)
- One function, `np.kernelsum()`, allows you to create your own estimators, tests, etc.
- The function `np.kernelsum()` is simply a call to highly optimized C code, so you get the benefits of compiled code with the flexibility of R

Non-smooth probability function estimation

- Consider the estimation of a probability function defined for unordered $X_i^d \in \mathcal{S} = \{0, 1, \ldots, c-1\}$, based upon n i.i.d. realizations from this process
- The "frequency" (`non-kernel`) estimator of $p(x^d)$ is given by

$$\tilde{p}(x^d) = \frac{\#X_i^d = x^d}{n} = \frac{1}{n}\sum_{i=1}^{n} I(X_i^d = x^d),$$

Non-smooth probability function estimation

- Consider the estimation of a probability function defined for unordered $X_i^d \in \mathcal{S} = \{0, 1, \ldots, c-1\}$, based upon n i.i.d. realizations from this process
- The "frequency" (`non-kernel`) estimator of $p(x^d)$ is given by

$$\tilde{p}(x^d) = \frac{\# X_i^d = x^d}{n} = \frac{1}{n}\sum_{i=1}^{n} I(X_i^d = x^d),$$

where $I(\cdot)$ is an indicator function defined by

$$I(X_i^d = x^d) = \begin{cases} 1 & \text{if } X_i^d = x^d \\ 0 & \text{otherwise.} \end{cases}$$

Non-smooth probability function estimation

- Consider the estimation of a probability function defined for unordered $X_i^d \in \mathcal{S} = \{0, 1, \ldots, c-1\}$, based upon n i.i.d. realizations from this process
- The "frequency" (`non-kernel`) estimator of $p(x^d)$ is given by

$$\tilde{p}(x^d) = \frac{\# X_i^d = x^d}{n} = \frac{1}{n}\sum_{i=1}^{n} I(X_i^d = x^d),$$

where $I(\cdot)$ is an indicator function defined by

$$I(X_i^d = x^d) = \begin{cases} 1 & \text{if } X_i^d = x^d \\ 0 & \text{otherwise.} \end{cases}$$

Smooth kernel estimation of a probability function

- Now, consider a `kernel` estimator of $p(x^d)$, defined as

$$\hat{p}(x^d) = \frac{1}{n}\sum_{i=1}^{n} L(X_i^d = x^d),$$

where $L(\cdot)$ is a kernel function defined by, say,

$$L(X_i^d = x^d) = \begin{cases} 1-\lambda & \text{if } X_i^d = x^d \\ \lambda/(c-1) & \text{otherwise,} \end{cases}$$

and where λ is a 'smoothing' parameter

Smooth kernel estimation of a probability function

- Now, consider a `kernel` estimator of $p(x^d)$, defined as

$$\hat{p}(x^d) = \frac{1}{n}\sum_{i=1}^{n} L(X_i^d = x^d),$$

where $L(\cdot)$ is a kernel function defined by, say,

$$L(X_i^d = x^d) = \begin{cases} 1-\lambda & \text{if } X_i^d = x^d \\ \lambda/(c-1) & \text{otherwise,} \end{cases}$$

and where λ is a 'smoothing' parameter

Smooth kernel estimation of a probability function

- Now, consider a `kernel` estimator of $p(x^d)$, defined as

$$\hat{p}(x^d) = \frac{1}{n} \sum_{i=1}^{n} L(X_i^d = x^d),$$

where $L(\cdot)$ is a kernel function defined by, say,

$$L(X_i^d = x^d) = \begin{cases} 1-\lambda & \text{if } X_i^d = x^d \\ \lambda/(c-1) & \text{otherwise,} \end{cases}$$

and where λ is a 'smoothing' parameter

Trivial example: smooth estimation of a probability function

- `x <- rbinom(100,1,0.5)`
- `plot(density(x))`
- `data <- data.frame(x=factor(x))`
- `bw <- np.density.bw(data)`
- `np.plot(data,bws=bw,ylim=c(0,1))`

Trivial example: smooth estimation of a probability function

- `x <- rbinom(100,1,0.5)`
- `plot(density(x))`
- `data <- data.frame(x=factor(x))`
- `bw <- np.density.bw(data)`
- `np.plot(data,bws=bw,ylim=c(0,1))`

Trivial example: smooth estimation of a probability function

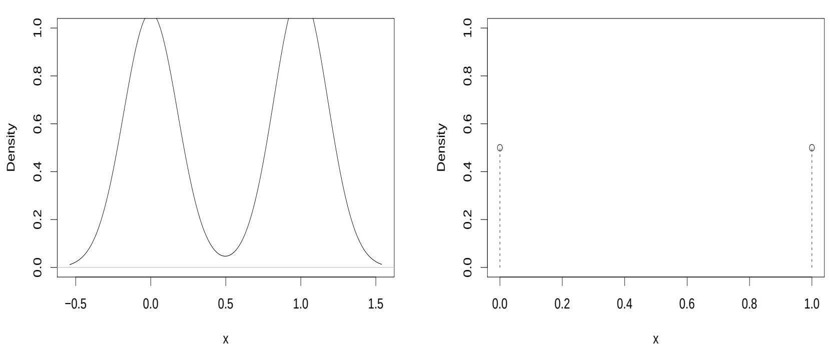
- `x <- rbinom(100,1,0.5)`
- `plot(density(x))`
- `data <- data.frame(x=factor(x))`
- `bw <- np.density.bw(data)`
- `np.plot(data,bws=bw,ylim=c(0,1))`

Trivial example: smooth estimation of a probability function

- `x <- rbinom(100,1,0.5)`
- `plot(density(x))`
- `data <- data.frame(x=factor(x))`
- `bw <- np.density.bw(data)`
- `np.plot(data,bws=bw,ylim=c(0,1))`

Trivial example: smooth estimation of a probability function

- `x <- rbinom(100,1,0.5)`
- `plot(density(x))`
- `data <- data.frame(x=factor(x))`
- `bw <- np.density.bw(data)`
- `np.plot(data,bws=bw,ylim=c(0,1))`



Smooth kernel estimation of mixed data probability functions

- Estimating a joint density function defined over mixed data follows naturally using generalized product kernels
- For example, for one discrete variable x^d and continuous variable x^c, our kernel estimator of the PDF would be

$$\hat{f}(x^d, x^c) = \frac{1}{nh_x}\sum_{i=1}^{n} L(X_i^d = x^d) W\left(\frac{X_i^c - x^c}{h_{x^c}}\right)$$

- $L(X_i^d = x^d)$ is a categorical data kernel function, while $W((X_i^c - x^c)/h_{x^c})$ is a continuous data kernel function (e.g., Epanechnikov or Gaussian)

Smooth kernel estimation of mixed data probability functions

- Estimating a joint density function defined over mixed data follows naturally using generalized product kernels
- For example, for one discrete variable x^d and continuous variable x^c, our kernel estimator of the PDF would be

$$\hat{f}(x^d, x^c) = \frac{1}{nh_x} \sum_{i=1}^{n} L(X_i^d = x^d) W\left(\frac{X_i^c - x^c}{h_{x^c}}\right)$$

- $L(X_i^d = x^d)$ is a categorical data kernel function, while $W((X_i^c - x^c)/h_{x^c})$ is a continuous data kernel function (e.g., Epanechnikov or Gaussian)

np: A Package for Nonparametric Kernel Smoothing with Mixed Datatypes
└Introduction to Kernel Smoothing with Mixed Data

Smooth kernel estimation of mixed data probability functions

- Estimating a joint density function defined over mixed data follows naturally using generalized product kernels
- For example, for one discrete variable x^d and continuous variable x^c, our kernel estimator of the PDF would be

$$\hat{f}(x^d, x^c) = \frac{1}{nh_x} \sum_{i=1}^{n} L(X_i^d = x^d) W\left(\frac{X_i^c - x^c}{h_{x^c}}\right)$$

- $L(X_i^d = x^d)$ is a categorical data kernel function, while $W((X_i^c - x^c)/h_{x^c})$ is a continuous data kernel function (e.g., Epanechnikov or Gaussian)

np: A Package for Nonparametric Kernel Smoothing with Mixed Datatypes
└Introduction to Kernel Smoothing with Mixed Data

Smooth kernel estimation of general statistical objects with mixed data

- Once we can consistently estimate a joint density function defined over mixed data, we can then proceed to estimate a range of statistical objects of interest to practitioners
- Some mainstays of applied data analysis include estimation of

np: A Package for Nonparametric Kernel Smoothing with Mixed Datatypes
└Introduction to Kernel Smoothing with Mixed Data

Smooth kernel estimation of general statistical objects with mixed data

- Once we can consistently estimate a joint density function defined over mixed data, we can then proceed to estimate a range of statistical objects of interest to practitioners
- Some mainstays of applied data analysis include estimation of
  - Regression functions and their derivatives
  - Conditional density functions and their quantiles
  - Conditional variance functions
  - Conditional mode functions (i.e., count data models, probability models etc.)

Smooth kernel estimation of general statistical objects with mixed data

- Once we can consistently estimate a joint density function defined over mixed data, we can then proceed to estimate a range of statistical objects of interest to practitioners
- Some mainstays of applied data analysis include estimation of
  - Regression functions and their derivatives
  - Conditional density functions and their quantiles
  - Conditional variance functions
  - Conditional mode functions (i.e., count data models, probability models etc.)

Smooth kernel estimation of general statistical objects with mixed data

- Once we can consistently estimate a joint density function defined over mixed data, we can then proceed to estimate a range of statistical objects of interest to practitioners
- Some mainstays of applied data analysis include estimation of
  - Regression functions and their derivatives
  - Conditional density functions and their quantiles
  - Conditional variance functions
  - Conditional mode functions (i.e., count data models, probability models etc.)

Smooth kernel estimation of general statistical objects with mixed data

- Once we can consistently estimate a joint density function defined over mixed data, we can then proceed to estimate a range of statistical objects of interest to practitioners
- Some mainstays of applied data analysis include estimation of
  - Regression functions and their derivatives
  - Conditional density functions and their quantiles
  - Conditional variance functions
  - Conditional mode functions (i.e., count data models, probability models etc.)

Smooth kernel estimation of general statistical objects with mixed data

- Once we can consistently estimate a joint density function defined over mixed data, we can then proceed to estimate a range of statistical objects of interest to practitioners
- Some mainstays of applied data analysis include estimation of
  - Regression functions and their derivatives
  - Conditional density functions and their quantiles
  - Conditional variance functions
  - Conditional mode functions (i.e., count data models, probability models etc.)

Nonparametric regression example

- data(oecd)
- attach(oecd)
- y <- growth
- X <- data.frame(factor(oecddummy), factor(year), initgdp, popgro, inv, humancap)
- bw <- np.regression.bw(xdat=X, ydat=y, regtype="ll")
- np.plot(xdat=X, ydat=y, bws=bw, plot.errors.method="bootstrap")

Nonparametric regression example

- data(oecd)
- attach(oecd)
- y <- growth
- X <- data.frame(factor(oecddummy), factor(year), initgdp, popgro, inv, humancap)
- bw <- np.regression.bw(xdat=X, ydat=y, regtype="ll")
- np.plot(xdat=X, ydat=y, bws=bw, plot.errors.method="bootstrap")

Nonparametric regression example

- data(oecd)
- attach(oecd)
- y <- growth
- X <- data.frame(factor(oecddummy), factor(year), initgdp, popgro, inv, humancap)
- bw <- np.regression.bw(xdat=X, ydat=y, regtype="ll")
- np.plot(xdat=X, ydat=y, bws=bw, plot.errors.method="bootstrap")

Nonparametric regression example

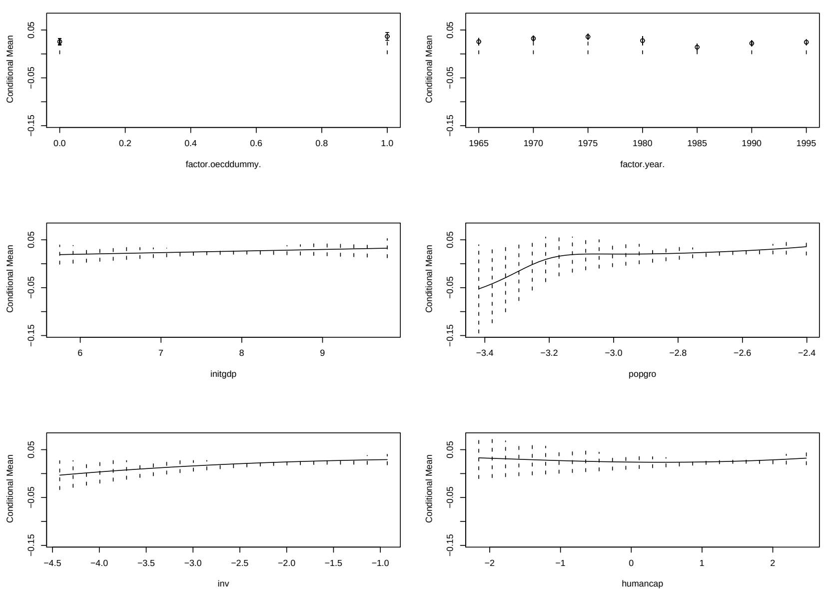
- data(oecd)
- attach(oecd)
- y <- growth
- X <- data.frame(factor(oecddummy), factor(year), initgdp, popgro, inv, humancap)
- bw <- np.regression.bw(xdat=X, ydat=y, regtype="ll")
- np.plot(xdat=X, ydat=y, bws=bw, plot.errors.method="bootstrap")

Nonparametric regression example

- `data(oecd)`
- `attach(oecd)`
- `y <- growth`
- `X <- data.frame(factor(oecddummy), factor(year), initgdp, popgro, inv, humancap)`
- `bw <- np.regression.bw(xdat=X, ydat=y, regtype="ll")`
- `np.plot(xdat=X, ydat=y, bws=bw, plot.errors.method="bootstrap")`

Nonparametric regression example

- `data(oecd)`
- `attach(oecd)`
- `y <- growth`
- `X <- data.frame(factor(oecddummy), factor(year), initgdp, popgro, inv, humancap)`
- `bw <- np.regression.bw(xdat=X, ydat=y, regtype="ll")`
- `np.plot(xdat=X, ydat=y, bws=bw, plot.errors.method="bootstrap")`



np: current capabilities

- Unconditional and conditional density estimation and bandwidth selection
- Conditional mean and gradient estimation (local constant and local polynomial)
- Conditional quantile and gradient estimation
- Model specification tests (regression, quantile, significance)
- Semiparametric regression (partially linear, index models, average derivative estimation)
- Index

np: current capabilities

- Unconditional and conditional density estimation and bandwidth selection
- Conditional mean and gradient estimation (local constant and local polynomial)
- Conditional quantile and gradient estimation
- Model specification tests (regression, quantile, significance)
- Semiparametric regression (partially linear, index models, average derivative estimation)
- Index

np: current capabilities

- Unconditional and conditional density estimation and bandwidth selection
- Conditional mean and gradient estimation (local constant and local polynomial)
- Conditional quantile and gradient estimation
- Model specification tests (regression, quantile, significance)
- Semiparametric regression (partially linear, index models, average derivative estimation)
- Index

np: current capabilities

- Unconditional and conditional density estimation and bandwidth selection
- Conditional mean and gradient estimation (local constant and local polynomial)
- Conditional quantile and gradient estimation
- Model specification tests (regression, quantile, significance)
- Semiparametric regression (partially linear, index models, average derivative estimation)
- Index

np: current capabilities

- Unconditional and conditional density estimation and bandwidth selection
- Conditional mean and gradient estimation (local constant and local polynomial)
- Conditional quantile and gradient estimation
- Model specification tests (regression, quantile, significance)
- Semiparametric regression (partially linear, index models, average derivative estimation)
- Index

np: current capabilities

- Unconditional and conditional density estimation and bandwidth selection
- Conditional mean and gradient estimation (local constant and local polynomial)
- Conditional quantile and gradient estimation
- Model specification tests (regression, quantile, significance)
- Semiparametric regression (partially linear, index models, average derivative estimation)
- Index