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 c 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 Nc 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.,
  ```r
  data(Italy)
  attach(Italy)
  data <- data.frame(ordered(year), gdp)
  ```
- Next, you typically proceed as follows:
  ```r
  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()`
**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()`

- 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.
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.

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

$$\hat{p}(x^d) = \frac{\#X^d_i = x^d}{n} = \frac{1}{n} \sum_{i=1}^{n} I(X^d_i = x^d),$$

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

$$I(\cdot) = \begin{cases} 1 & \text{if } \cdot \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$
Non-smooth probability function estimation

- Consider the estimation of a probability function defined for unordered \( X^d_i \in S = \{0, 1, \ldots, c - 1\} \), based upon \( n \) i.i.d. realizations from this process.

- The "frequency" (non-kernel) estimator of \( p(x^d) \) is given by:

\[
\hat{p}(x^d) = \frac{\# X^d_i = x^d}{n} = \frac{1}{n} \sum_{i=1}^{n} I(X^d_i = x^d),
\]

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

\[
I(X^d_i = x^d) = \begin{cases} 
1 & \text{if } X^d_i = x^d \\
0 & \text{otherwise}.
\end{cases}
\]

Smooth kernel estimation of a probability function

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

\[
\hat{p}(x^d) = \frac{1}{n} \sum_{i=1}^{n} L(X^d_i = x^d),
\]

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

\[
L(X^d_i = x^d) = \begin{cases} 
1 - \lambda & \text{if } X^d_i = x^d \\
\lambda/(c - 1) & \text{otherwise},
\end{cases}
\]

and where \( \lambda \) is a 'smoothing' parameter.

Non-smooth probability function estimation

- Consider the estimation of a probability function defined for unordered \( X^d_i \in S = \{0, 1, \ldots, c - 1\} \), based upon \( n \) i.i.d. realizations from this process.

- The "frequency" (non-kernel) estimator of \( p(x^d) \) is given by:

\[
\hat{p}(x^d) = \frac{\# X^d_i = x^d}{n} = \frac{1}{n} \sum_{i=1}^{n} I(X^d_i = x^d),
\]

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

\[
I(X^d_i = x^d) = \begin{cases} 
1 & \text{if } X^d_i = x^d \\
0 & \text{otherwise}.
\end{cases}
\]

Smooth kernel estimation of a probability function

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

\[
\hat{p}(x^d) = \frac{1}{n} \sum_{i=1}^{n} L(X^d_i = x^d),
\]

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

\[
L(X^d_i = x^d) = \begin{cases} 
1 - \lambda & \text{if } X^d_i = x^d \\
\lambda/(c - 1) & \text{otherwise},
\end{cases}
\]

and where \( \lambda \) 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,

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

and where $\lambda$ is a ‘smoothing’ parameter.

Trivial example: smooth estimation of a probability function

```r
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

\[ x \leftarrow \text{rbinom}(100,1,0.5) \]
\[ \text{plot(density(x))} \]
\[ \text{data} \leftarrow \text{data.frame}(x=\text{factor}(x)) \]
\[ \text{bw} \leftarrow \text{np.density.bw(data)} \]
\[ \text{np.plot(data,bws=bw,ylim=c(0,1))} \]

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
    \[ \hat{f}(x^d, x^c) = \frac{1}{nh_x} \sum_{i=1}^{n} L(X^d_i = x^d) W \left( \frac{X^c_i - x^c}{h_{x^c}} \right) \]
  - \( L(X^d_i = x^d) \) is a categorical data kernel function, while \( W((X^c_i - x^c)/h_{x^c}) \) 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)

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

```r
# Load data
data(oecd)
attach(oecd)

# Define variables
y <- growth
X <- data.frame(factor(oecddummy), factor(year), initgdp, popgro, inv, humancap)

# Calculate bandwidth
bw <- np.regression.bw(xdat=X, ydat=y, regtype="ll")

# Plot results
np.plot(xdat=X, ydat=y, bws=bw, plot.errors.method="bootstrap")
```

Nonparametric regression example

```r
# Load data
data(oecd)
attach(oecd)

# Define variables
y <- growth
X <- data.frame(factor(oecddummy), factor(year), initgdp, popgro, inv, humancap)

# Calculate bandwidth
bw <- np.regression.bw(xdat=X, ydat=y, regtype="ll")

# Plot results
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
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