

Some Examples of `sivipm` Use

J.P. Gauchi and A. Bouvier
INRA, UR1404
F78352 Jouy en Josas Cedex
France

April 13, 2015

Contents

1	Brief description of the <code>sivipm</code> package	2
2	Data	2
2.1	Example of a raw dataset: <code>cornell0</code>	2
2.2	Example of a transformed dataset: <code>XY180</code>	4
3	Polynomial	4
3.1	When the provided data are “raw” data	4
3.1.1	Polynomial described by a character vector	4
3.1.2	Polynomial of standard type	5
3.2	When the provided data are transformed data	5
3.2.1	Polynomial described by a character vector	6
3.2.2	Polynomial of a standard type	6
4	Polynomials handling	7
4.1	Binding polynomials	7
4.2	Removing monomials from a polynomial	8
5	Calculations	8
5.1	Individual and total sensitivity indices, significant components	8
5.2	Add an alea in the computation of the total sensitivity indices	11
5.3	Ask for more results	13
5.4	Computation Y by Y	14
5.5	When there are missing values	14
5.6	When there is no polynomial description	14
5.7	Confidence intervals for the total sensitivity indices by a bootstrap method	15
6	References	16

1 Brief description of the `sivipm` package

The `sivipm` R package computes total and individual sensitivity indices, significant components, and confidence intervals for the total sensitivity indices. The total and individual sensitivity indices are calculated using a method based on the VIP of the PLS regression, proposed by J.P. Gauchi ([Gauchi et al (2010)], [Gauchi (2012)], [Gauchi (2015)]). The significant components are calculated by the SIMCA software ([SIMCA Software]) rule and by the Lazraq & Cl  roux test ([Lazraq and Cl  roux (2001)]). The confidence intervals for the total sensitivity indices are determined by the bootstrap method.

Some examples of use are given here. Complete description of the syntax of the `sivipm` functions is available through the help functions of the package.

2 Data

The observed inputs and outputs must be stored in data-frames.

Input dataset may be,

- either, **raw** dataset. The number of columns is then the number of X-inputs,
- or **transformed** dataset. The number of columns is then the number of monomials in the polynomial. Each column is the value of a monomial calculated on non provided X-inputs.

Output dataset, or response variable, must be stored in a data-frame with one or several columns. The response can be uni or multivariate.

Missing values are accepted. However, if there are missing values and the response is multivariate, then it is not possible to compute the significant components.

Categorical variables must be transformed into indicator variables before processing analysis:

- if the variable has more than 2 categories, it must be split into as many 0/1 indicator variables as distinct categories;
- if it has two categories, set -1 on the lines corresponding to the first category, $+1$ on the lines corresponding to the second one (or vice versa);
- if it has one category only, the dataset is then not accepted.

The function `factorsplit` can be used to do this transformation: it creates a data-frame where the *factors* are split or changed into numeric variables.

2.1 Example of a raw dataset: `cornell0`

`cornell0` is an example of raw dataset. The data-frame contains seven input variables, followed by one response variable ([Kettaneh-Wold (1992)]).

Distillation	Reformat	NaphthaT	NaphthaC	Polymer	Alkylat	Gasoline	Y
0.00	0.23	0.00	0.00	0.00	0.74	0.03	98.7
0.00	0.10	0.00	0.00	0.12	0.74	0.04	97.8
0.00	0.00	0.00	0.10	0.12	0.74	0.04	96.6
0.00	0.49	0.00	0.00	0.12	0.37	0.02	92.0
0.00	0.00	0.00	0.62	0.12	0.18	0.08	86.6
0.00	0.62	0.00	0.00	0.00	0.37	0.01	91.2
0.17	0.27	0.10	0.38	0.00	0.00	0.08	81.9
0.17	0.19	0.10	0.38	0.02	0.06	0.08	83.1
0.17	0.21	0.10	0.38	0.00	0.06	0.08	82.4
0.17	0.15	0.10	0.38	0.02	0.10	0.08	83.2
0.21	0.36	0.12	0.25	0.00	0.00	0.06	81.4
0.00	0.00	0.00	0.55	0.00	0.37	0.08	88.1

Creation of the input data-frame, `XCornell10`, and the output data-frame, `YCornell10`:

```
> library(sivipm)
> XCornell10 <- cornell10[,1:7] # inputs
> YCornell10 <- as.data.frame( cornell10[,8]) # output
> dimnames(YCornell10)[[2]] <- "Y"
```

2.2 Example of a transformed dataset: XY180

XY180 is an example of transformed dataset¹. The data-frame contains 160 columns. The first ones are the values of 158 monomials calculated on 18 inputs and the two last are the response variables.

```
> X180 <- XY180[,1:158]
> Y180 <- XY180[, 159:160]
> dimnames(Y180)[[2]] = c("Y1", "Y2")
```

3 Polynomial

The polynomial can be described,

- either, by a character vector,
- or be chosen among a list of standard types.

Note. The first monomials must always be the X-input variables.

From the input data-frame and its polynomial description, an object of class `polyX`² has to be created before processing the calculations. The following sections illustrate the way of creating a `polyX` object.

3.1 When the provided data are “raw” data

3.1.1 Polynomial described by a character vector

The polynomial can be described by a character vector, whose each element codes a monomial. The variables are identified either by their names or by their numbers. The character “*” (asterisk) denotes interaction between variables.

For example, we create the following polynomial of maximal degree equal to 3:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_1*x_3 + x_2*x_2 + x_2*x_4 + x_3*x_4 + x_5*x_5 + x_6*x_6 + x_7*x_7*x_7$$

where x_i is the X-input number i .

It is described by the following character vector, where the variables are identified by their numbers:

¹This dataset is provided by S. Lefebvre (ONERA, Palaiseau, France) and described in [Gauchi (2015)].

²`polyX`: class which contains the polynomial description in a list structure and in an indicator matrix, and the transformed data, calculated on the raw data, if necessary.

```
> monomials <- c("1","2","3", "4","5", "6", "7",
+              "1*3", "2*2", "2*4", "3*4", "5*5",
+              "6*6", "7*7*7")
```

We could have identified the variables by their names as well:

```
> monomialsbis <- c("Distillation","Reformat","NaphthaT", "NaphthaC",
+                 "Polymer", "Alkylat", "Gasoline",
+                 "Distillation*NaphthaT", "Reformat*Reformat",
+                 "Reformat*NaphthaC", "NaphthaT*NaphthaC",
+                 "Polymer*Polymer", "Alkylat*Alkylat",
+                 "Gasoline*Gasoline*Gasoline")
```

The function `vect2polyX` creates the `polyX` object:

```
> PCornello <- vect2polyX (XCornello, monomials)
```

3.1.2 Polynomial of standard type

The function `crpolyX` can be used to generate three types of standard polynomials:

- **full**: complete polynomial with all the terms of degree less or equal to the maximal degree. Example with 2 variables and degree 3:
 $x_1 + x_2 + x_1^2 + x_2 * x_1 + x_2^2 + x_1^3 + x_2 * x_1^2 + x_2^2 * x_1 + x_2^3$
- **power**: only power terms of degree less or equal to the maximal degree. Example with 2 variables and degree 3: $x_1 + x_2 + x_1^2 + x_2^2 + x_1^3 + x_2^3$
- **interact**: only interactions of degree less or equal to the maximal degree. Example with 2 variables and degree 3: $x_1 + x_2 + x_2 * x_1 + x_2 * x_1^2 + x_2^2 * x_1$

Note. The first monomials are always the X-input variables.

For example, we create a polynomial composed with the power terms of degree less or equal to 2 on the 3 first variables of `XCornello`:

```
> PCornellobis <- crpolyX(XCornello[,1:3], 2, type="power")
> options(width=60) # set display width to avoid line truncation
> PCornellobis
```

Polynome description:

```
Distillation + Reformat + NaphthaT + Distillation*Distillation +
Reformat*Reformat + NaphthaT*NaphthaT
```

Polynome description using variable numbers:

```
1 + 2 + 3 + 1*1 + 2*2 + 3*3
```

Number of observations: 12

3.2 When the provided data are transformed data

The functions which create a `polyX` object from transformed data and, respectively from a character vector, and from a standard type of polynomial, are `vect2PolyXT`, and `crpolyXT`.

3.2.1 Polynomial described by a character vector

The transformed data-frame `X180` (see Section 2.2) is the result of the calculation of 158 monomials on 18 X-inputs.

1. The polynomial is described by a character vector. Here, the variables are identified by their numbers:

```
> Pexp <- as.character(1:18)
> for (i in 1:13) {
+   Pexp <- c(Pexp, paste(i,"*",i, sep=""))
+ }
> for (i in 1:13) {
+   Pexp <- c(Pexp, paste(i,"*",i, "*",i, sep=""))
+ }
> for (i in 1:12) {
+   for (j in (i+1):13) {
+     Pexp <- c(Pexp, paste(i,"*",j, sep=""))
+   }
+ }
> for (i in 1:18) {
+   if (i != 15)
+     Pexp <- c(Pexp, paste("15*",i,sep=""))
+ }
> for (i in 16:18) {
+   for (j in 9:11) {
+     Pexp <- c(Pexp, paste(i, "*", j,sep=""))
+   }
+ }
> Pexp <- c(Pexp, c("13*16"), c("13*17"), c("13*18"),
+           c("14*16"), c("14*17"), c("14*18"),
+           c("14*9"), c("14*10"), c("12*14"), c("2*14"))
```

2. We create the `polyX` object, by using the `vect2polyXT` function. The names of the 18 inputs must be explicitly provided, because, unlike the raw data case, they cannot be deduced from the column names of the input data-frame.

```
> varnames180 <- c("ALTI", "MACH", "POWERS", "EAI", "CAP",
+                 "YAW", "ROLL", "PITCH", "VIS", "RH",
+                 "TA", "HBASE", "HOUR", "MODEL", "CLOUDS",
+                 "IHAZE1", "IHAZE2", "IHAZE3")
> PX180 <- vect2polyXT(varnames180, X180, Pexp)
```

3.2.2 Polynomial of a standard type

For illustrative purpose, we suppose that the data-frame `X180` is the result of the calculation of a polynomial of maximal degree 2 made up of the power terms. As this polynomial has 36 monomials only, the extra columns of `X180` are ignored.

```
> PX180b <- crpolyXT(varnames180, X180, 2, type="power")
> summary(PX180b)
```

```

Number of observations: 180
Polynome degree: 2
Number of monomials: 36
Number of variables: 18
Polynome description:
ALTI + MACH + POWERS + EAI + CAP + YAW + ROLL + PITCH + VIS +
RH + TA + HBASE + HOUR + MODEL + CLOUDS + IHAZE1 + IHAZE2 +
IHAZE3 + ALTI*ALTI + MACH*MACH + POWERS*POWERS + EAI*EAI + CAP*CAP +
YAW*YAW + ROLL*ROLL + PITCH*PITCH + VIS*VIS + RH*RH + TA*TA +
HBASE*HBASE + HOUR*HOUR + MODEL*MODEL + CLOUDS*CLOUDS + IHAZE1*IHAZE1 +
IHAZE2*IHAZE2 + IHAZE3*IHAZE3
Polynome description using variable numbers:
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 + 14 +
15 + 16 + 17 + 18 + 1*1 + 2*2 + 3*3 + 4*4 + 5*5 + 6*6 + 7*7 +
8*8 + 9*9 + 10*10 + 11*11 + 12*12 + 13*13 + 14*14 + 15*15 +
16*16 + 17*17 + 18*18

```

4 Polynomials handling

4.1 Binding polynomials

To put together the monomials of several polynomials calculated on the same dataset of inputs, use the `bind` method.

Example with the dataset `XCornell10`:

1. Creation of the first polynomial: a polynomial of degree 2, with the power terms only, calculated on the variables 3 and 4 of the data-frame `XCornell10`.

```

> P1 <- crpolyX(XCornell10[, 3:4], 2, type="power")
> print(P1)

```

```

Polynome description:
NaphthaT + NaphthaC + NaphthaT*NaphthaT + NaphthaC*NaphthaC
Polynome description using variable numbers:
1 + 2 + 1*1 + 2*2
Number of observations: 12

```

2. Creation of the second polynomial: interaction only.

```

> P2 <- vect2polyX(XCornell10[, 3:4], c("1", "2", "1*2"))
> print(P2)

```

```

Polynome description:
NaphthaT + NaphthaC + NaphthaT*NaphthaC
Polynome description using variable numbers:
1 + 2 + 1*2
Number of observations: 12

```

3. Put together the monomials of these two polynomials:

```
> P3 <- bind(P1, P2)
> print(P3)
```

```
Polynome description:
NaphthaT + NaphthaC + NaphthaT*NaphthaT + NaphthaC*NaphthaC +
NaphthaT*NaphthaC
Polynome description using variable numbers:
1 + 2 + 1*1 + 2*2 + 1*2
Number of observations: 12
```

Note. The duplicated monomials are removed.

4.2 Removing monomials from a polynomial

The function **takeoff** removes monomials from a **polyX** object.

Example:

```
> X <- cornell0[,1:3]
> monos <- c("Distillation", "Reformat", "NaphthaT",
+           "Distillation*Reformat", "Reformat*Reformat")
> P <- vect2polyX(X, monos)
> # Remove the last two monomials:
> P2 <- takeoff(P, c(4,5))
> print(P2)
```

```
Polynome description:
Distillation + Reformat + NaphthaT
Polynome description using variable numbers:
1 + 2 + 3
Number of observations: 12
```

Same where the monomials to suppress are expressed in character strings:

```
> P2 <- takeoff(P, c("Distillation*Reformat", "Reformat*Reformat"))
```

Note. Taking off the monomials equal to the X-inputs, is not accepted.

5 Calculations

5.1 Individual and total sensitivity indices, significant components

The function **sivipm** is the main function of the package. It calculates the individual and total sensitivity indices and the significant components. Its arguments are the response data-frame and the **polyX** object. Its options are:

- **nc**, the number of components (2 by default);
- **options**, a vector to limit what is returned. Valid values are:
 - "fo.isivip", to return first order individual sensitivity indices,

- "tsivip", to return total sensitivity indices,
- "simca" and "lazraq", to return the significant components calculated by the SIMCA software rule ([SIMCA Software]) and by the Lazraq and Cl eroux test ([Lazraq and Cl eroux (2001)]), respectively. Threshold of these tests is 0.05.

(all is returned by default);

- **graph**, for drawing a graph of the total sensitivity indices (FALSE by default);
- **alea** to insert a random variable (FALSE by default) (see Section 5.2);
- **output** to return intermediate results (none by default) (see Section 5.3).

The returned value is a list whose components depend on what has been required:

- When the vector **options** includes "fo.isivip":
 - **fo.isivip**, first order individual sensitivity indices.
- When the vector **options** includes "tsivip":
 - **tsivip**, the total sensitivity indices,
 - **percentage**, the sorted percentages of total sensitivity indices.
- When the vector **options** includes "simca" or "lazraq":
 - **simca.signifcomponents** or **lazraq.signifcomponents**, boolean vectors whose values are TRUE for the significant components.
- When the option **alea** is set (see Section 5.2):
 - **monosignif**, a boolean vector whose values are TRUE for the significant monomials,
 - **correlalea**, the correlation between the outputs and the random variable.
- When the option **output** is set:
 - **output**, a list of intermediate results (see Section 5.3).

Example with the response dataset **YCornell10** (see Section 2.1) and the polyX object **PCornell10** (see Section 3.1.1):

```
> sivism(YCornell10, PCornell10, nc=10, graph= TRUE)
```

```
$fo.isivip
Distillation      Reformat      NaphthaT      NaphthaC
0.0935179735 0.0006690142 0.0936659204 0.0666229905
      Polymer      Alkylat      Gasoline
0.0325265346 0.1294409978 0.0732672626
```

```

$tsivip
Distillation      Reformat      NaphthaT      NaphthaC
  0.17366732    0.10664095    0.26149630    0.24027001
  Polymer      Alkylat      Gasoline
  0.07884904    0.25375385    0.14440597

```

```

$percentage
NaphthaT      Alkylat      NaphthaC      Distillation
  20.768783    20.153855    19.082930     13.793154
  Gasoline      Reformat      Polymer
  11.469134     8.469728     6.262416

```

```

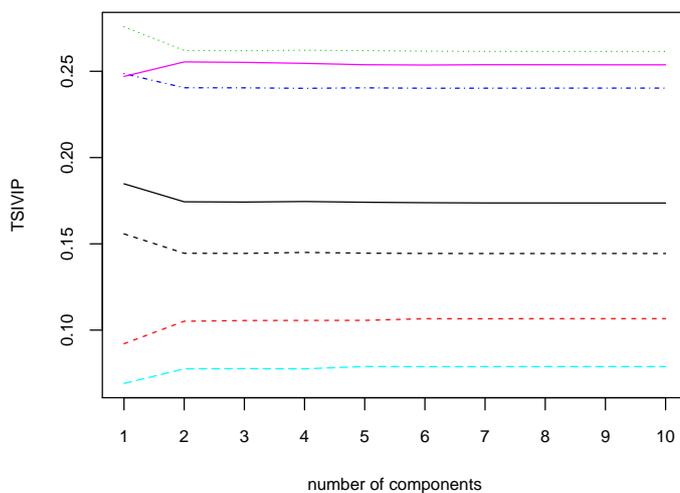
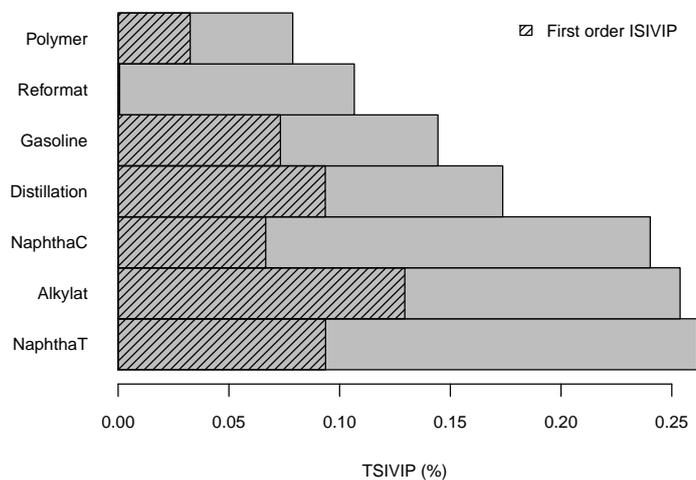
$simca.signifcomponents
  t1  t2  t3  t4  t5  t6  t7  t8  t9  t10
TRUE TRUE FALSE FALSE TRUE TRUE TRUE TRUE TRUE FALSE

```

```

$lazraq.signifcomponents
  t1  t2  t3  t4  t5  t6  t7  t8  t9  t10
TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

```



5.2 Add an alea in the computation of the total sensitivity indices

When the option **alea** is set, a uniform random variable is inserted into the computation of the total sensitivity indices. Comparison can then be made of its indices with the ones of the other variables. The non significant monomials, — those for which the individual sensitivity indices is less or equal than the one of the random variable — are excluded from the total sensitivity indices calculation. The correlation between the outputs and the random variable is returned.

Example with the response dataset YCornell10 (see Section 2.1) and the polyX object PCornell10 (see Section 3.1.1):

```
> set.seed(403)
> res <- sivism(YCornell10, PCornell10, nc=2, alea=TRUE,
+             graph=TRUE, options=c("fo.isivip", "tsivip"))
> res
```

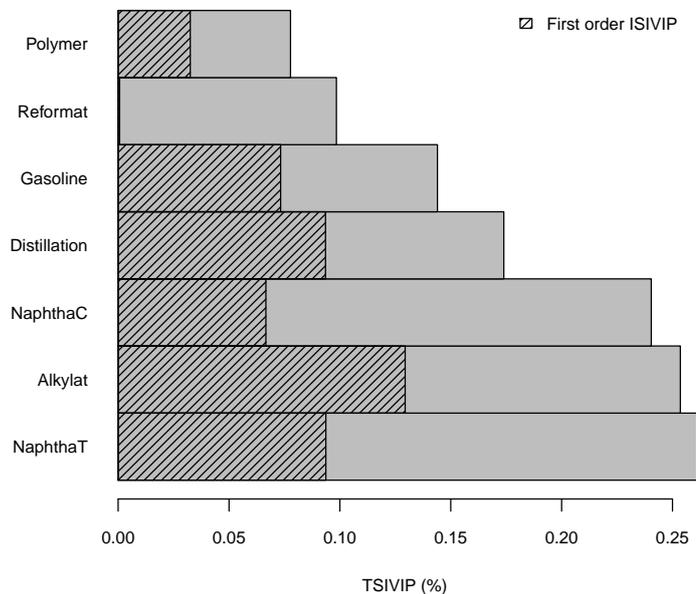
```
$fo.isivip
Distillation      Reformat      NaphthaT      NaphthaC
0.0935179735 0.0006690142 0.0936659204 0.0666229905
      Polymer      Alkylat      Gasoline
0.0325265346 0.1294409978 0.0732672626
```

```
$tsivip
Distillation      Reformat      NaphthaT      NaphthaC
0.17393473 0.09843718 0.26116685 0.24042730
      Polymer      Alkylat      Gasoline
0.07773183 0.25353931 0.14399843
```

```
$percentage
      NaphthaT      Alkylat      NaphthaC Distillation
20.906132 20.295555 19.245952 13.923293
      Gasoline      Reformat      Polymer
11.526923 7.879793 6.222351
```

```
$monosignif
      Distillation      Reformat
      TRUE      FALSE
      NaphthaT      NaphthaC
      TRUE      TRUE
      Polymer      Alkylat
      TRUE      TRUE
      Gasoline      Distillation*NaphthaT
      TRUE      TRUE
      Reformat*Reformat      Reformat*NaphthaC
      TRUE      TRUE
      NaphthaT*NaphthaC      Polymer*Polymer
      TRUE      TRUE
      Alkylat*Alkylat Gasoline*Gasoline*Gasoline
      TRUE      TRUE
```

```
$correlalea
      Y
[1,] -0.2123951
```



5.3 Ask for more results

The expert user can require additional results by setting the option `output`. Detailed explanations of these results can be found in [Gauchi et al (2010)], [Gauchi (2012)] and [Gauchi (2015)].

`output` is a character vector whose valid values are:

- `isivip`: to return all the individual sensitivity indices
- `betaNat`: to return `betaNat` (natural β) and `betaNat0` (β coefficient)
- `VIP`: to return `VIP` and `VIPind`
- `RSS`: to return `RSS`
- `PRESS`: to return `PRSS`
- `Q2`: to return `Q2` and `Q2cum`
- `PLS`: to return PLS results: `mweights`, `weights`, `x.scores`, `x.loadings`, `y.scores`, `y.loadings`, `cor.tx`, `cor.ty`, `expvar`, `X.hat`, `Y.hat`

Example with the response dataset `YCornell10` (see Section 2.1) and the `polyX` object `PCornell10` (see Section 3.1.1).

```
> res <- sivism(YCornell10, PCornell10, nc=2, output= c("betaNat", "PLS"))
> names(res)
```

```

[1] "fo.isivip"          "tsivip"
[3] "percentage"        "simca.signifcomponents"
[5] "lazraq.signifcomponents" "output"

> names(res$output)

[1] "betaNat" "betaNat0" "PLS"

```

5.4 Computation Y by Y

To compute the total sensitivity indices for each response variable successively, use the R function **apply**.

Example with the two-columns data-frame Y180:

```

> l180 <- apply(Y180, 2, sivipm, PX180, nc=2, options="tsivip")
> names(l180)

[1] "Y1" "Y2"

> names(l180$Y1)

[1] "tsivip" "percentage"

> names(l180$Y2)

[1] "tsivip" "percentage"

```

5.5 When there are missing values

When there are missing values, the option `simca` which rules the calculation of significant components by the SIMCA software rule ([\[SIMCA Software\]](#)) is ignored.

The option `lazraq` is active if the response is univariate only.

To illustrate this case, we created the data-frame `cornell11` from `cornell10` by introducing some missing values. We calculate the significant components:

```

> X <- cornell11[,1:7]
> Y <-as.data.frame( cornell11[,8])
> polyXm <- vect2polyX (X, monomials)
> sivipm(Y, polyXm, nc=8, options="lazraq")

$lazraq.signifcomponents
  t1  t2  t3  t4  t5  t6  t7  t8
TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE

```

5.6 When there is no polynomial description

The polynomial description is only required to calculate the total sensitivity indices. It can be omitted to compute the individual sensitivity indices of each column of the input dataset, and the significant components.

For example, we calculate the first order individual sensitivity indices of all the columns of the transformed dataset XY180, without having to describe the polynomial:

```

> b <- new("polyX", dataX.exp=X180)
> res <- siviipm(Y180, b, nc=2, options="fo.isivip")
> names(res)

```

```
[1] "fo.isivip"
```

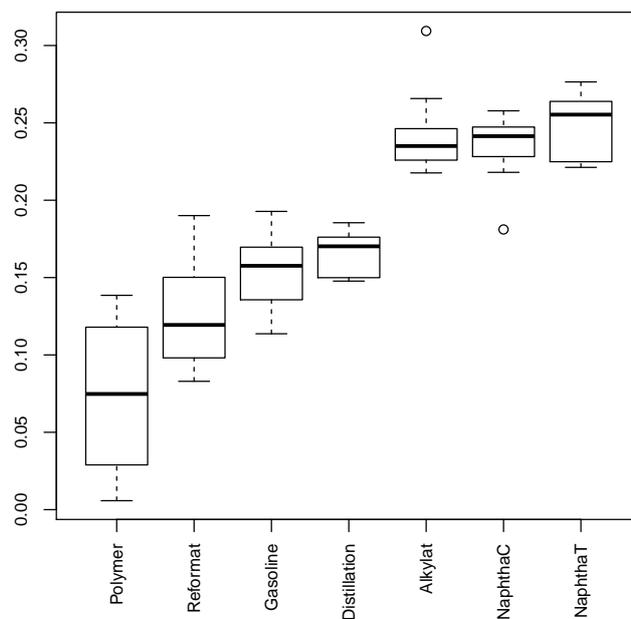
5.7 Confidence intervals for the total sensitivity indices by a bootstrap method

The function `sivipboot` calculates confidence intervals for the total sensitivity indices. Its arguments are the response data-frame, the `polyX` object, and the number of bootstrap loops. Its options are `nc`, the required number of components (2 by default), `alpha`, the test threshold (0.05 by default), and `graph` for a boxplot of the results.

In the following example, confidence intervals are calculated in ten bootstrap loops:

```
> sivipboot(YCornello, PCornello, B=10 , nc=4, alpha=0.05, graph=TRUE)
```

	IC.inf	IC.sup
Distillation	0.147709256	0.1854368
Reformat	0.082951362	0.1900557
NaphthaT	0.221228476	0.2764461
NaphthaC	0.181112141	0.2577898
Polymer	0.005790846	0.1385232
Alkylat	0.217641106	0.3093869
Gasoline	0.113681952	0.1927613



6 References

- [Gauchi et al (2010)] Gauchi, J.-P. and Lehuta, S. and Mahã©vas, S. 2010. Optimal sensitivity analysis under constraints: Application to fisheries. In *Procedia - Social and Behavioral Sciences*, vol. 2. Elsevier, pp. 7658-7659. Sixth International Conference on Sensitivity Analysis of Model Output (Milan, Italy), 19-22 July; co-organized by the ELEUSI research center of Bocconi University and by the Joint Research Center of the European Commission.
- [Gauchi (2012)] Gauchi, J.-P. 2012. Global Sensitivity Analysis: The SIVIP method (SAS/IML language). Rapport technique 2012-3. INRA, UR1404, F-78350 Jouy-en-Josas, France.
- [Gauchi (2015)] Gauchi, J.-P. 2015. A practical method of global sensitivity analysis under constraints. Rapport technique 2015-1. INRA, UR1404, F-78350 Jouy-en-Josas, France.
- [Kettaneh-Wold (1992)] Kettaneh-Wold, N. 1992. Analysis of mixture data with partial least squares. *Chemometrics and Intelligent Laboratory Systems*. Vol. 14, pp. 57-69.
- [Lazraq and Clérout (2001)] Lazraq, A. and Clérout, R. 2001. The PLS multivariate regression model: testing the significance of successive PLS components. *Journal of Chemometrics*. Vol. 15(6), pp 523-536.
- [SIMCA Software] SIMCA Software. <http://www.umetrics.com/products/simca>.
- [Tenenhaus (1998)] Tenenhaus M. 1998. *La régression PLS. Théorie et pratique*. Ed. TECHNIP. Paris.