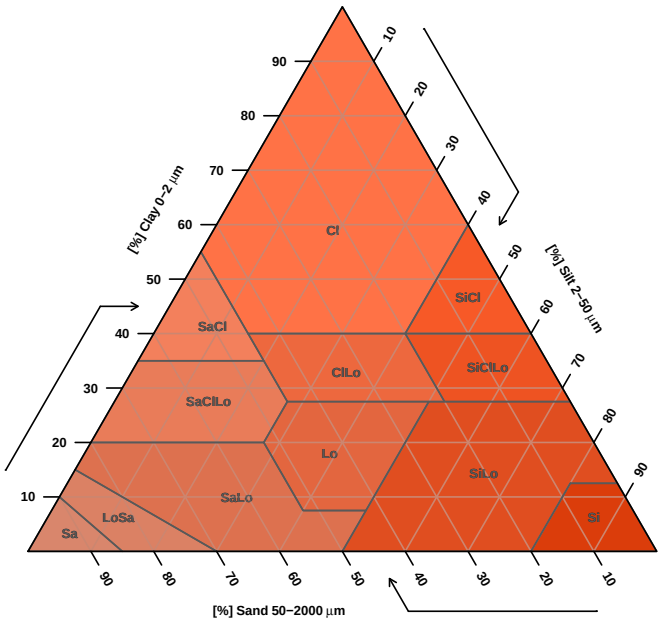
The soil texture wizard: R functions for plotting, classifying, transforming and exploring soil texture data

Julien Moeys

May 27, 2010

build number 48.



Contents

1 About this document

1.1 Why creating 'The soil texture wizard'?

Officially: *The Soil Texture Wizard* R functions are an attempt to provide a generic toolbox for soil texture data in R. These functions can (1) plot soil texture data (2) classify soil texture data, (3) transform soil texture data from and to different systems of particle size classes, and (4) provide some tools to 'explore' soil texture data (in the sense of a statistical visual analysis). All there

tools are designed to be inherently multi-triangles, multi-geometry and multi-particle sizes classification

Officiously: What was initially a slight reshape of R PLOTRIX package (by J. Lemon and B. Bolker), for my personal use[1], to add the French 'Aisne' soil texture triangle, gradually skidded and ended up in a totally reshaped and extended code (over a 3 year period). There is unfortunately no compatibility at all between the two codes.

1.2 About R

This document is about functions (and package project) written in R "language and environment for statistical computing" ([23]), and has been generated with R version 2.11.0 (2010-04-22).

R website: <http://www.R-project.org>

If you don't know about R, it is never too later to start...

1.3 About the author

I am an agriculture engineer, soil scientist and R programmer. See my website for more details (http://julienmoeys.free.fr/).

The R functions presented in this document may not always conform to the 'best R programming practices', they are nevertheless programmed carefully, well checked, and should work efficiently for most uses.

At this stage of development, some bugs should still be expected. The code has been written in 3 years, and tested quite extensively since then, but it has never been used by other people. If you find some bugs, please contact me at:

jules_m78-soiltexture@yahoo.fr

1.4 Credits and License

[1] It was also an excellent way to learn R.

  - **to Share** - to copy, distribute and transmit the work;
  - **to Remix** - to adapt the work.

- Under the following conditions:
  - **Attribution** - You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work);
  - **Share Alike** - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

'The soil texture wizard' R **functions** are licensed under a GNU General Public License Version 3.

Given the fact that a lot of the work presented here has been done on my free time, and given its highly permissive license, **this document is provided with NO responsibilities, guarantees or supports from the author or his employer** (Swedish University of Agricultural Sciences).

Please notice that the R software itself is licensed under a GNU General Public License Version 2, June 1991.

This tutorial has been created with the (great) **Sweave** tool, from Friedrich Leisch ([17]). Sweave allows the smooth integration of R code and R output (including figures) in a LaTeXdocument.

2 Introduction: About soil texture, texture triangles and texture classifications

2.1 What are soil granulometry and soil texture(s)?

Soil granulometry is the repartition of soil solid particles between (a range of) particle sizes. As the range of particle sizes is in fact continuous, they have been subdivided into different **particle size classes**.

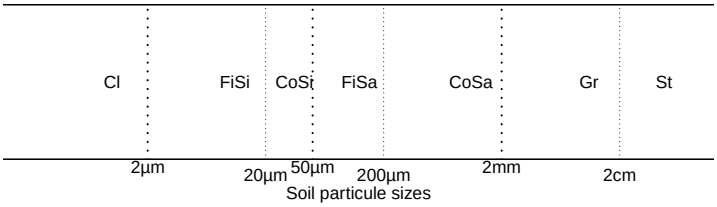
The most common subdivision of soil granulometry into classes is the **fine earth**, for particles ranging from **0 to 2mm (2000μm)**, and **coarse particles**, for particles bigger than **2mm**. Only the fine earth interests us in this document, although the study of soil granulometry can be extended to the coarse fraction (for stony soils).

Fine earth is generally (but not always; see below) divided into **3 particle size classes: clay (fine particles), silt (medium size particles) and sand (coarser particles in the fine earth)**. All soil scientists use the range **0-2μm** for **clay**. So silt lower limit is also always **2μm**. But the convention for **silt / sand** particle size limit **varies from country to country**. **Silt**

particle size range can be **2-20μm** (Atterberg system[20][24]; 'International system'; ISSS[2]; Australia[3][20]; Japan[24]), **2-50μm** (FAO[4]; USA; France[20][25]), **2-60μm** (UK and Sweden[24]) or **2-63μm** (Germany, Austria, Denmark and The Netherlands[24]). Logically, **sand** particle size range also varies accordingly to these systems: **20-2000μm**, **50-2000μm**, **60-2000μmeters** or **63-2000μmeters**.

Silt class is sometimes divided into **fine silts** and **coarse silts**, and **sand** class is sometimes divided into **fine sand** and **coarse sand**, but in this document / package, we only focus on clay / silt / sand classes.

Below is a scheme representing the different particle size classes used in France (with Cl for Clay, FiSi for Fine Silt, CoSi for Coarse Silt, FiSa for Fine Sand, CoSa for Coarse Sand, Gr for Gravels and St for Stones). The figure is adapted from Moeys 2007[21], and based on information from Baize & Jabiol 1995[5]. The particle size axis (abscissa) is log-scale:



Soil particles – and each soil particle size class – occupy a given volume in the soil, and have a given mass. They are nevertheless generally not expressed as 'absolute' volumetric quantities[5]. They are expressed as **'relative abundance'**, that is **kilograms of particles of a given class per kilograms of fine earth**. These measurements are also always made on dehydrated soil samples (dried slightly above 100°C), in order to be independent from soil water content (which varies a lot in time and space).

Soil texture is defined as the relative abundance of the 3 particle size classes: clay, silt and sand[6].

In summary, important information to know when talking about soil texture (and using these functions):

- Soil's fine earth is generally (but not always) divided into 3 soil texture classes:
  - Clay;
  - Silt;
  - Sand.
- The silt / sand limit varies:

[2]ISSS: International Society of Soil Science. Now IUSS, International Union of Soil Science
[3]Strangely, only a small number of countries have adopted the so called 'international system'
[4]Food and Agriculture Organization of the United Nations
[5]for instance kilograms of clay per liters of soil', or 'liters of clay per liter of soil'
[6]But some systems define for than 3 particle size classes for soil texture

  - 20μm; or
  - 50μm; or
  - 60μm; or
  - 63μm.

- Soil texture measurement do have a specific unit and a corresponding 'sum of the 3 texture classes', that is constant:
  - in % or $g.100g^{-1}$ (sum: 100); or
  - in fraction [−] or $kg.kg^{-1}$ (sum: 1); or
  - in $g.kg^{-1}$ (sum: 1000);

More than 3 particle size classes?

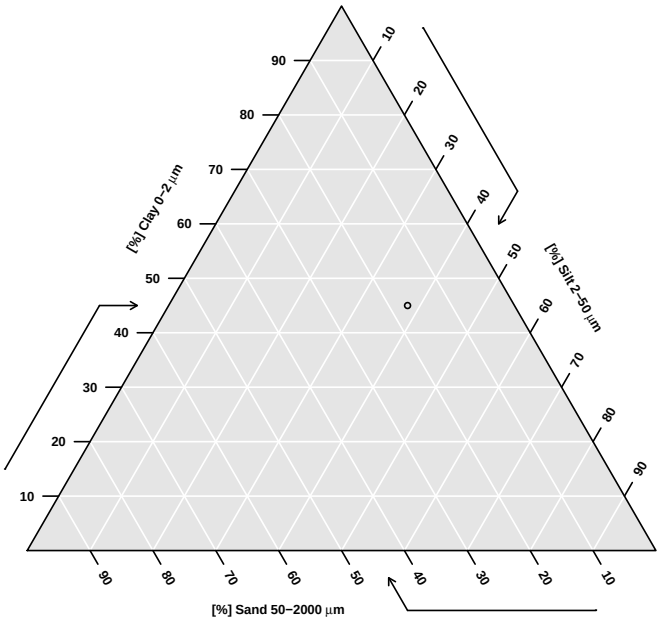
Some country have a particle size classes system that differ from the common 'clay silt sand' triplet. Sweden is using a system with 4 particle size classes: Ler [0-2μm], Mjäla [2-20μm], Mo [20-200μm] and Sand [200-2000μm] (See table 1 p.9 in Lidberg 2009[18]). Ler corresponds to clay. When considering the International or Australian particle size system (silt-sand limit 20μm), Mjäla is silt, and 'Mo + Sand' is sand. When considering other systems with a silt-sand limit at 50μm, 60μm or 63μm, Mjäla is fine-silt, Mo is 'coarse-silt + fine sand', and Sand is coarse-sand.

'The Soil Texture Wizard' has been made for systems with 3 particle size classes (clay, silt and sand), **because soil texture triangles have 3 sides, and thus can only represent texture data that are divided into 3 particle size classes**. There are methods to estimate 3 particle size classes when more classes are presented in the data (although the best is to measure texture so it also can fit a system with 3 particle size classes system).

2.2 What are soil texture triangle and classes

Soil texture triangles are also called **soil texture diagrams**.

Soil texture can be plotted on a **ternary plot** (also called triangle plot). In a ternary plot, 3D coordinates, which sum is constant, are projected in the 2D space, using simple trigonometry rules. The texture of a soil sample can be plotted inside a texture triangle, as shown in the example below for the texture 45% clay, 38% silt and 17% sand:



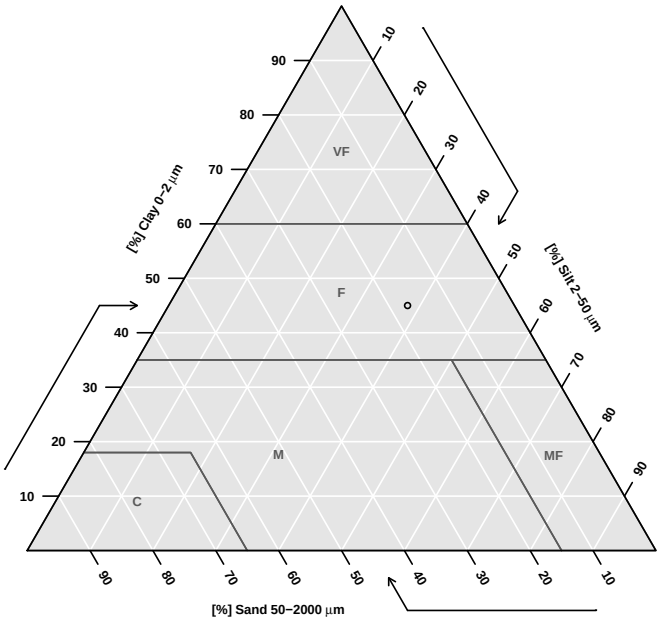
When mapping soil, field pedologists usually estimate texture by manipulating a moist (but not saturated) soil sample in their hand. Depending on the relative importance of clay silt and sand, the mechanical properties of the soil (plasticity, stickyness, roughness) varies. Pedologists have 'classified' clay silt and sand relative abundance as a function of what they could feel in the field: they have divided the 'soil texture space' into classes.

Soil particle size classes (clay, silt and sand) should not be confused with **soil texture classes**. While the first are ranges of particle sizes, the latter are defined by a 'range of clay, silt and sand' (see the graph below). Soil texture should not be confused with the concept of **soil structure**, that concerns the way these particles are arranged together (or not) into peds, clods and aggregates (etc.) of different size and shape[7]. This document does not deal with soil structure.

Soil texture classes are convenient to represent soil texture on soil maps[8], and there use is quite broad (soil description, soil classification, pedogenesis, soil functional properties, pedotransfer functions, etc.). One of these texture classification systems is the FAO system. Here is the representation of the same point as in the graph above, but with the FAO soil classification system on the background.

[7] In the same way bricks and cement (the texture) can be arranged into a house (the structure)

[8] It is more easy to represent 1 variable, soil texture class, than 3 variables: clay silt and sand



The soil texture class symbols are:

	abbr	name
1	VF	Very fine
2	F	Fine
3	M	Medium
4	MF	Medium fine
5	C	Coarse

Table 1: Texture classes of the FAO system / triangle

The main characteristics of the graph (texture triangle) are:

- 3 Axis, graduated from 0 to 100%, each of them carrying 1 particle size class.
  - Sand on the bottom axis;
  - Clay on the left axis;
  - Silt on the right axix.
- It is possible to permute clay, silt and sand axis, but this choice depend on the particle size classification used.
- Inside the triangle, the lines of equi-values for a given axis/particle size class are ALWAYS parallel to the (other) axis that intersect the axis of interest at 'zero' (minimum value).

- The 3 axis intersect each other in 3 submits, that are characterized by an **angle**. In the example above, all 3 angles are 60 degrees. But other angles are possible, depending on the soil texture classification used. It is for instance possible to have a 90 degrees angle on the left, and 45 degrees angles on the top and on the right (right-angled triangle).
- The 3 axis have a **direction** of increasing texture abundance. This direction is often referred as 'clock' or 'anticlock', but they can also be directed 'inside' the triangle in some cases. In the example above, all the axis are clockwise: texture increase when rotating in the opposite direction as a clock.
- **Labeled ticks** are placed at regular intervals (10%) on the texture triangle axes, apart if the axis is directed inside the triangle. Ticks can be placed at irregular intervals if they are placed at each value taken by the texture class polygons vertices (This is a smart representation, unfortunately not implemented here).
- An **broken arrow** is drawn 'parallel' to each axis. The first part indicate the direction of increasing value, and the second, broken, part indicates the direction of the equi-value for that axis/texture class.
- The **axis labels** indicates the texture class concerned, and should ideally remind the particle size limits, because these limits are of crucial importance when (re)using soil texture data (Silt and Sand does not exactly mean the same particle size limits everywhere).
- **Soil texture class boundaries** are drawn inside the triangle. They are 2D representation of 3D limits. They are generally **labeled** with soil texture class abbreviations (or full names).
- Inside the triangle frame, a grid can be represented, for each ticks and ticks label drawn outside the triangle.

3 Installing the package

3.1 Installing the package from r-forge

The Soil Texture Wizard is now available on r-forge, under the project name soiltexture. **If you have the latest R version** installed, the package can be intalled with the following commands:

```
install.packages(
    pkgs  = "soiltexture",
    repos = "http://R-Forge.R-project.org"
)   #
```

And it can be loaded with the following command:

```
require( soiltexture )
```

If you get bored of the package, you can unload it and uninstall it with the following commands:

```
detach( package:soiltexture )
remove.packages( "soiltexture" )
```

If you don't have the latest R version, please try to install the package from the binaries. In the next section, an example is given for R under MS Windows systems (Zip binaries).

3.2 Installing the package from Windows binaries (.zip)

To install and load the package directly from r-forge zip binaries, you can type the following command:

```
download.file(
    url =
"http://r-forge.r-project.org/bin/windows/contrib/2.10/soiltexture_1.0.zip",
    destfile = file.path( getwd(), "soiltexture_1.0.zip" )
)   #
#
install.packages(
    pkgs  = file.path( getwd(), "soiltexture_1.0.zip" ),
    repos = NULL
)   #
#
file.remove( "soiltexture_1.0.zip" )
```

Where 2.10 should be replaced by the latest stable R version and 1.0 by the latest package version on r-forge.

3.3 Load the latest package sources

If all the options above failed to intall the soiltexture package, you can still load the latest package sources in R by using the following command:

```
source(
    paste(
        sep = "",
        "http://r-forge.r-project.org/scm/viewvc.php/*checkout*",
        "/pkg/soiltexture/R/soiltexture.r?&root=soiltexture"
    )   #
)   #
```

The examples shown in this vignette are ran with these sources.

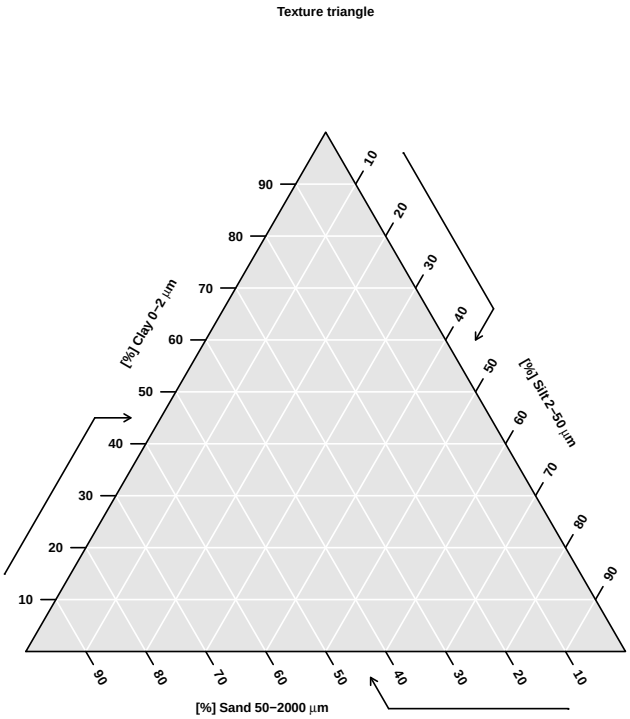
4 Plotting soil texture triangles and classification systems

The package comes with 8 predefined soil texture triangles. Empty (i.e. without soil textures data) soil texture triangles can be plotted, in order to obtain smart representation of the soil texture classification. Of course, it is also possible to plot 'classification free' texture triangles.

4.1 An empty soil texture triangle

Below is the code to display an empty triangle (without classification and without data):

```
TT.plot( class.sys = "none" )
```



The option `class.sys` (characters) determines the soil texture classification system used. If set to `'none'`, an empty soil texture triangle is plotted.

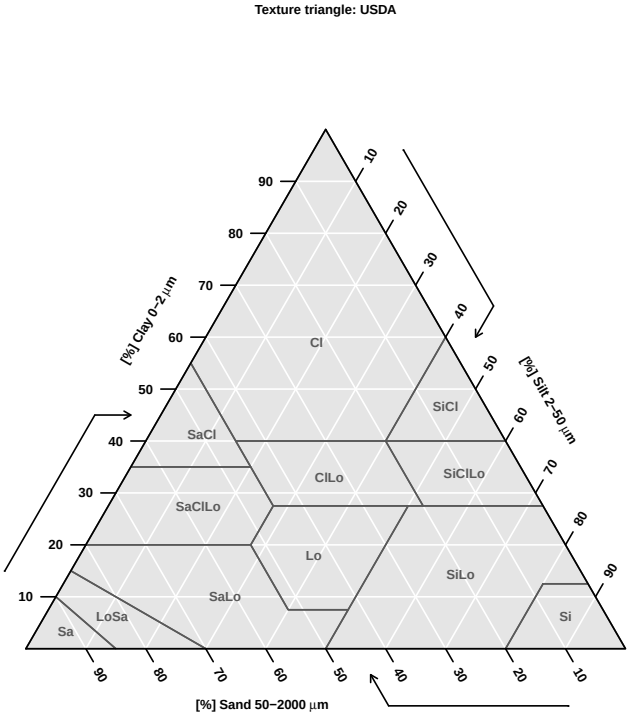
Without further options, the plotted default soil texture triangle has the same geometry as the FAO, USDA or French 'Aisne' soil texture triangles (i.e. all axis are clockwise, all angles are 60 degrees, sand is on the bottom axe, clay on the left and silt on the right).

The default unit is always percentage (0 to 100%). It is also equivalent to $g.100g^{-1}$.

4.2 The USDA soil texture classification

To display a USDA texture triangle, type:

```
TT.plot( class.sys = "USDA.TT" )
```



When the option `class.sys` is set to `"USDA.TT"`, a soil texture triangle with USDA classification system is used.

The USDA soil texture triangle has been built considering a silt - sand limit of 50μmeters.

See the table for soil texture classes symbols.

	abbr	name
1	Cl	clay
2	SiCl	silty clay
3	SaCl	sandy clay
4	ClLo	clay loam
5	SiClLo	silty clay loam
6	SaClLo	sandy clay loam
7	Lo	loam
8	SiLo	silty loam
9	SaLo	sandy loam
10	Si	silt
11	LoSa	loamy sand
12	Sa	sand

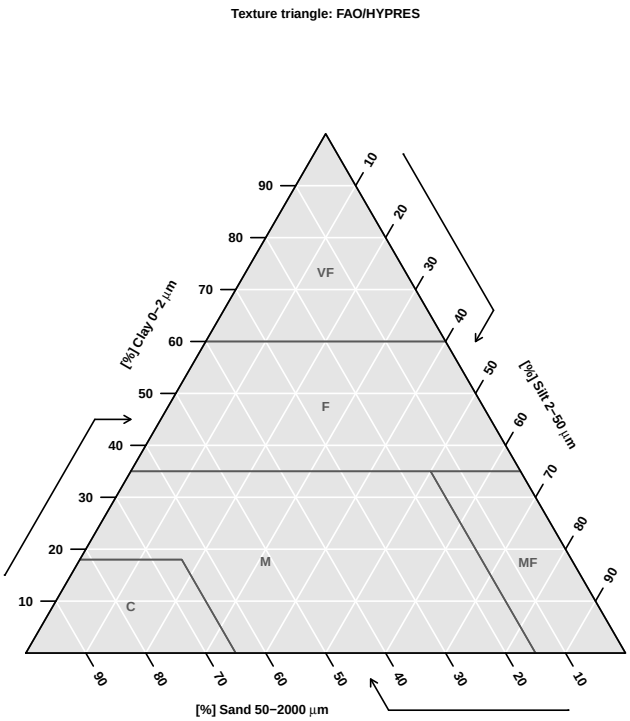
Table 2: Texture classes of the USDA system / triangle

The reference used to digitize this triangle is the Soil Survey Manual (Soil Survey Staff 1993[26]).

4.3 The FAO soil texture classification (also known as 'European Soil map', or 'HYPRES')

To display a FAO / HYPRES texture triangle, type:

```
TT.plot( class.sys = "FAO50.TT" )
```



De Forges et al. 2008[25] pointed out the fact that the silt-sand particle size limit that is officially related to the FAO soil texture triangle has changed over time, 50μm, then 63μm, and then again 50μm for some projects. We here consider that the FAO / EU Soil map / HYPRES soil texture triangle has a silt - sand limit of 50μm. As this choice is somehow arbitrary, we have named the 'FAO' option `"FAO50.TT"` in order to avoid any confusion. It will be explained later in the document how it is possible to add a custom texture triangle to the existing list, that could for instance be used to configure an FAO texture triangle with another silt - sand limit.

See the table for soil texture classes symbols.

The references used to digitize this triangle is the texture triangle provided by the HYPRES project web site ([10]). The The Canadian Soil Information System (CanSIS) also provides some details on this triangle ([3]).

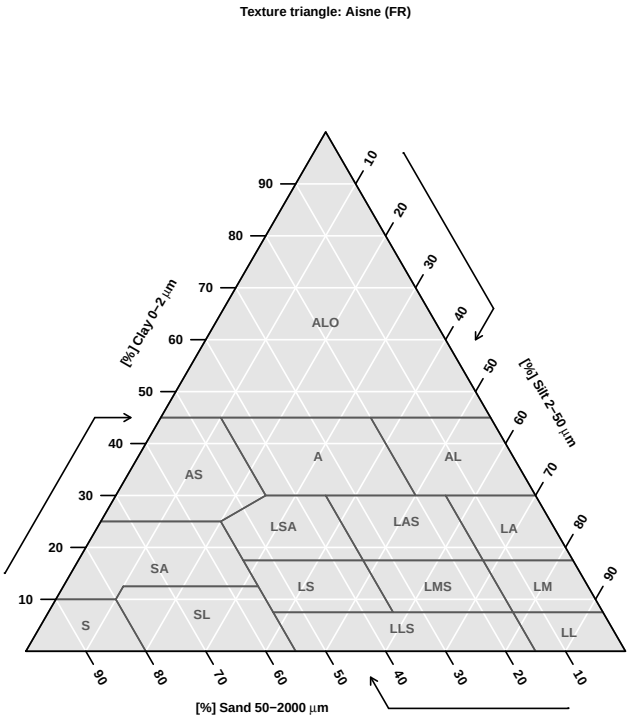
	abbr	name
1	VF	Very fine
2	F	Fine
3	M	Medium
4	MF	Medium fine
5	C	Coarse

Table 3: Texture classes of the FAO system / triangle

4.4 The French 'Aisne' soil texture classification

To display a French 'Aisne' texture triangle, type:

```
TT.plot( class.sys = "FR.AISNE.TT" )
```



The French Aisne soil texture triangle has been built considering a silt - sand limit of 50μmeters.

See the table for soil texture classes symbols[9].

The references used for digising this triangle is Baize and Jabiol 1995[5] and Jamagne 1967[15]. This triangle may be referred as the 'Triangle des textures de la Chambre d'Agriculture de l'Aisne' (en: texture triangle of the Aisne extension service).

[9] In classes 14 and 15, 'leger' should be replaced by 'léger'. R (and Sweave) can not display french accents easily, and I found no easy trics for displaying them.

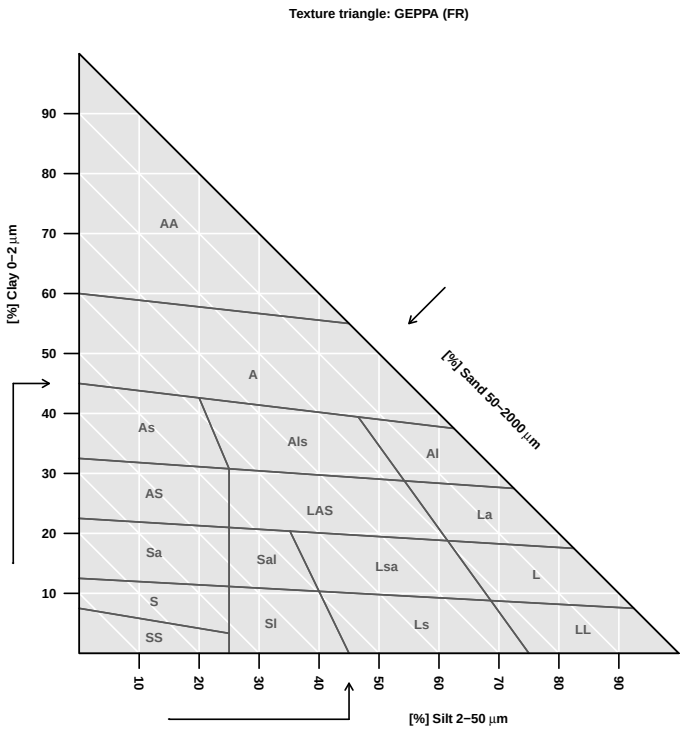
	abbr	name
1	ALO	Argile lourde
2	A	Argile
3	AL	Argile limoneuse
4	AS	Argile sableuse
5	LA	Limon argileux
6	LAS	Limon argilo-sableux
7	LSA	Limon sablo-argileux
8	SA	Sable argileux
9	LM	Limon moyen
10	LMS	Limon moyen sableux
11	LS	Limon sableux
12	SL	Sable limoneux
13	S	Sable
14	LL	Limon leger
15	LLS	Limon leger sableux

Table 4: Texture classes of the French 'Aisne' system / triangle

4.5 The French 'GEPPA' soil texture classification

To display a French 'GEPPA' texture triangle, type:

```
TT.plot( class.sys = "FR.GEPPA.TT" )
```



The French GEPPA soil texture triangle has been built considering a silt - sand limit of 50μmeters.

See the table for soil texture classes symbols.

	abbr	name
1	AA	Argile lourde
2	A	Argileux
3	As	Argile sableuse
4	Als	Argile limono-sableuse
5	Al	Argile limoneuse
6	AS	Argilo-sableux
7	LAS	Limon argilo-sableux
8	La	Limon argileux
9	Sa	Sable argileux
10	Sal	Sable argilo-limoneux
11	Lsa	Limon sablo-argileux
12	L	Limon
13	S	Sableux
14	SS	Sable
15	Sl	Sable limoneux
16	Ls	Limon sableux
17	LL	Limon pur

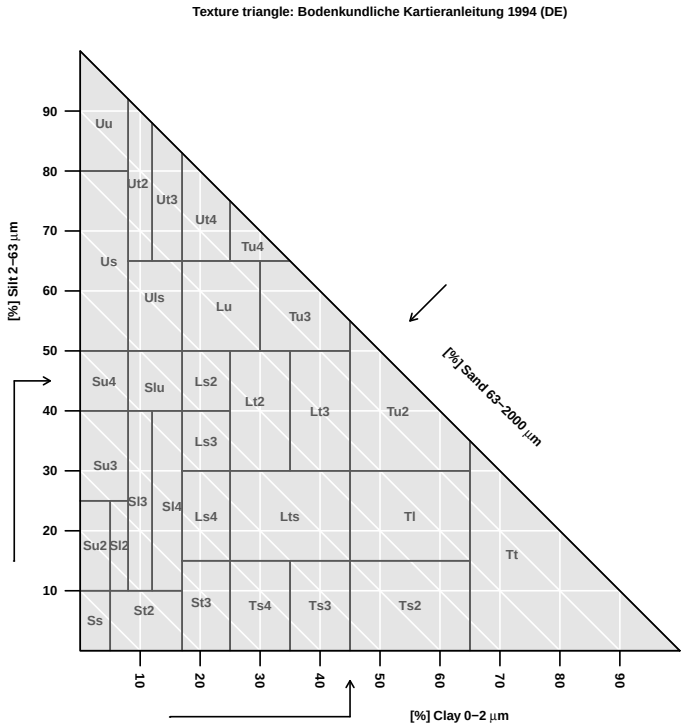
Table 5: Texture classes of the French 'GEPPA' system / triangle

This triangle has been digitized after `sols-de-bretagne.fr` 2009[27]. The website refers to an illustration from Baize and Jabiol 1995[5]. 'GEPPA' means 'Groupe d'Etude pour les Problèmes de Pédologie Appliquée' (en: Group for the study of applied pedology problems / questions).

4.6 The German Bodenartendiagramm (B.K. 1994) soil texture classification

To display a German Bodenartendiagramm (BK 1994) texture triangle, type:

```
TT.plot( class.sys = "DE.BK94.TT" )
```



The German Bodenartendiagramm (BK 1994) soil texture triangle has been built considering a silt - sand limit of 63μmeters.

See the table for soil texture classes symbols.

The references used to digitize this triangle and name the classes are `de.wikipedia.org` 2009 [4] and `nibis.ni.schule.de` 2009[2]. The triangle is also presented in GEOVLEX 2009[12] (Online lexicon from the Halle-Wittenberg University) and Bormann 2007[6] (for quadruple check). The triangle is referred as Bodenartendiagramm 'Korngrößendreieck' from Bodenkundliche Kartieranleitung 1994.

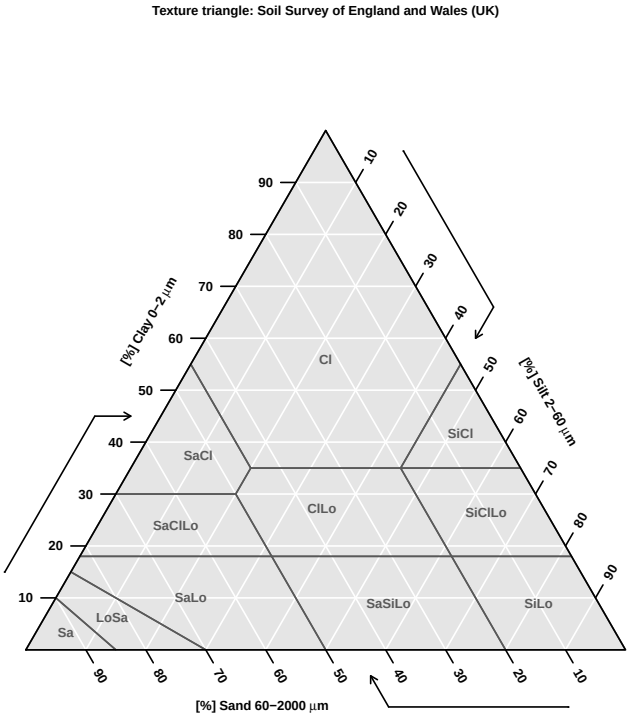
	abbr	name
1	Ss	reiner Sand
2	Su2	Schwach schluffiger Sand
3	Sl2	Schwach lehmiger Sand
4	Sl3	Mittel lehmiger Sand
5	St2	Schwach toniger Sand
6	Su3	Mittel schluffiger Sand
7	Su4	Stark schluffiger Sand
8	Slu	Schluffig-lehmiger Sand
9	Sl4	Stark lehmiger Sand
10	St3	Mittel toniger Sand
11	Ls2	Schwach sandiger Lehm
12	Ls3	Mittel sandiger Lehm
13	Ls4	Stark sandiger Lehm
14	Lt2	Schwach toniger Lehm
15	Lts	Sandig-toniger Lehm
16	Ts4	Stark sandiger Ton
17	Ts3	Mittel sandiger Ton
18	Uu	Reiner Schluff
19	Us	Sandiger Schluff
20	Ut2	Schwach toniger Schluff
21	Ut3	Mittel toniger Schluff
22	Uls	Sandig-lehmiger Schluff
23	Ut4	Stark toniger Schluff
24	Lu	Schluffiger Lehm
25	Lt3	Mittel toniger Lehm
26	Tu3	Mittel schluffiger Ton
27	Tu4	Stark schluffiger Ton
28	Ts2	Schwach sandiger Ton
29	Tl	Lehmiger Ton
30	Tu2	Schwach schluffiger Ton
31	Tt	Reiner Ton

Table 6: Texture classes of the German system / triangle

4.7 UK Soil Survey of England and Wales texture classification

To display a Soil Survey of England and Wales texture triangle (UK), type:

```
TT.plot( class.sys = "UK.SSEW.TT" )
```



UK Soil Survey of England and Wales texture triangle has been built considering a silt - sand limit of 60μmeters.

See the table for soil texture classes symbols.

The reference used to digitize this triangle is Defra – Rural Development Service – Technical Advice Unit 2006[9] (Technical Advice Note 52 – Soil texture).

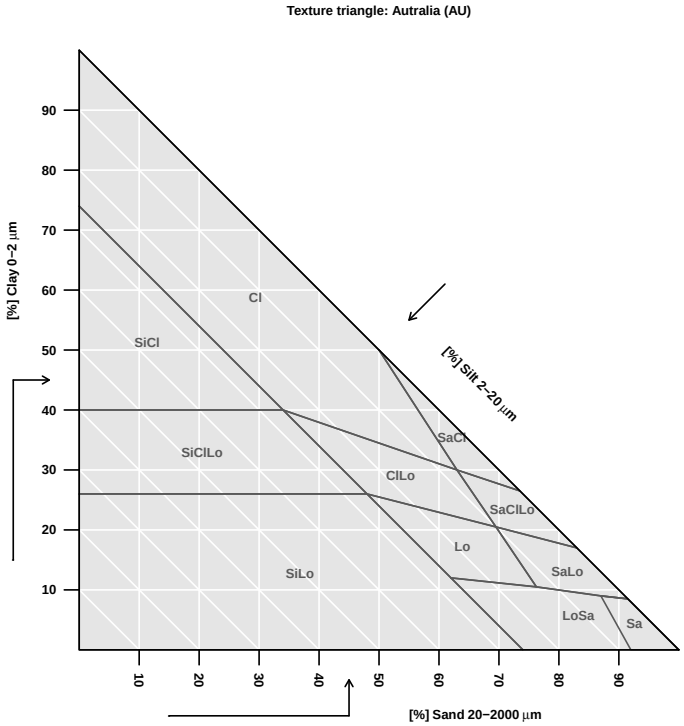
4.8 The Australian soil texture classification

To display an Autralian texture triangle, type:

```
TT.plot( class.sys = "AU.TT" )
```

	abbr	name
1	Cl	Clay
2	SaCl	Sandy clay
3	SiCl	Silty clay
4	ClLo	Clay loam
5	SiClLo	Silty clay loam
6	SaClLo	Sandy clay loam
7	SaLo	Sandy loam
8	SaSiLo	Sandy silt loam
9	SiLo	Silt loam
10	LoSa	Loamy sand
11	Sa	Sand

Table 7: Texture classes of the UK system / triangle



The Australian soil texture classification has been built considering a silt - sand limit of 20μmeters.

See the table for soil texture classes symbols.

There are probably small errors in the exact placement of some texture classes vertices (expected to be 1 or 2% of the 'exact value'), due to technical difficulties for reproducing precisely this triangle (reproduced after both Minasny and McBratney 2001[20], and Holbeche 2008[14] (brochure 'Soil Texture-Laboratory Method' from soilquality.org.au).

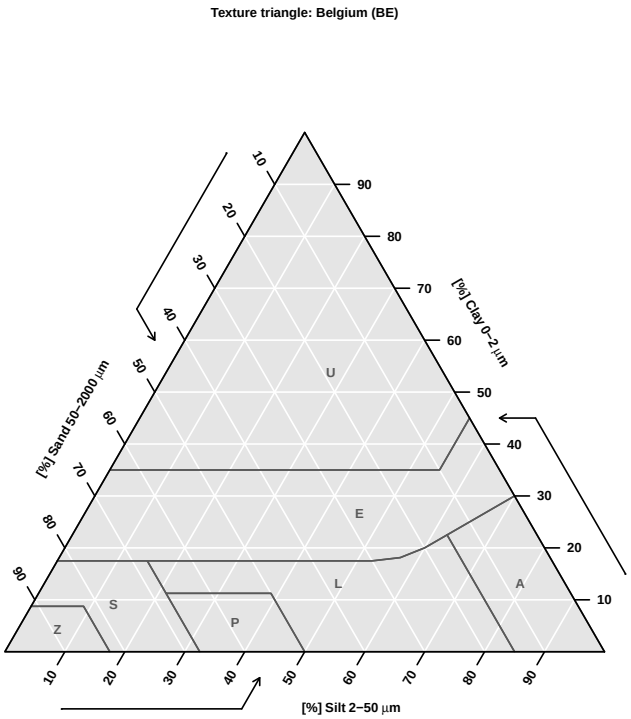
	abbr	name
1	Cl	Clay
2	SiCl	Silty clay
3	SiClLo	Silt clay loam
4	SiLo	Silty loam
5	ClLo	Clay loam
6	Lo	Loam
7	LoSa	Loamy sand
8	SaCl	Sandy clay
9	SaClLo	Sandy clay loam
10	SaLo	Sandy loam
11	Sa	Sand

Table 8: Texture classes of the Australian system / triangle

4.9 The Belgian soil texture classification

To display an Belgium texture triangle, type:

```
TT.plot( class.sys = "BE.TT" )
```



The Belgian soil texture classification has been built considering a silt - sand limit of 50μmeters.

See the table for soil texture classes symbols[10]. The class names are given in French and in Flemish.

	abbr	name
1	U	Argile lourde | Zware klei
2	E	Argile | Klei
3	A	Limon | Leem
4	L	Limon sableux | Zandleem
5	P	Limon sableux leger | Licht zandleem
6	S	Sable limoneux | Lemig zand
7	Z	Sable | Zand

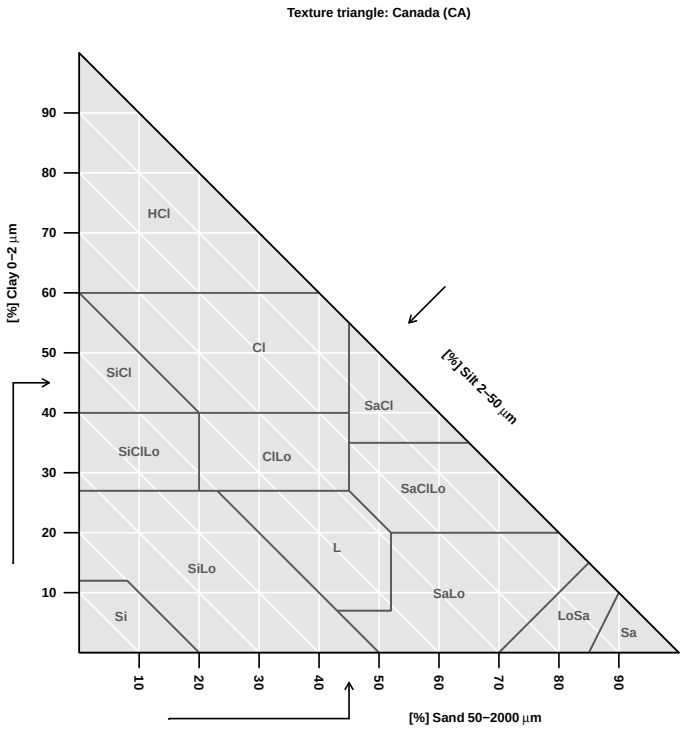
Table 9: Texture classes of the Belgian system / triangle

This texture triangle has been built after images from Defourny et al.[8] and Van Bossuyt[29].

4.10 The Canadian soil texture classification

To display a Canadian texture triangle with English texture class abbreviations, type:

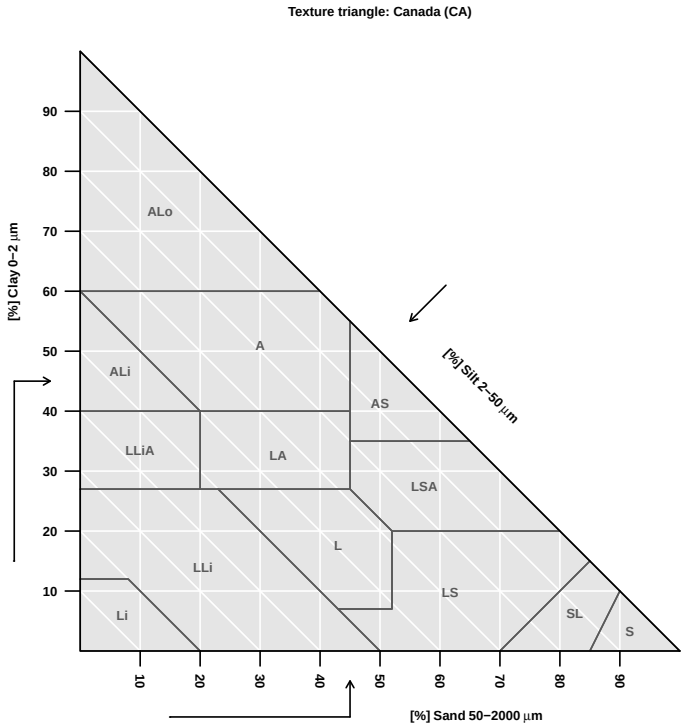
```
TT.plot( class.sys = "CA.EN.TT" )
```



[10]In classes 5, 'leger' should be replaced by 'léger'. R (and Sweave) can not display french accents easily, and I found no easy trics for displaying them.

For the same triangle with French texture class abbreviations type:

```
TT.plot( class.sys = "CA.FR.TT" )
```



The Canadian soil texture classification has been built considering a silt - sand limit of 50μmeters (reference[1]).

See the table for soil texture classes symbols, in English:

	abbr	name
1	HCl	Heavy clay
2	SiCl	Silty clay
3	Cl	Clay
4	SaCl	Sandy clay
5	SiClLo	Silty clay loam
6	ClLo	Clay loam
7	SaClLo	Sandy clay loam
8	SiLo	Silty loam
9	L	Loam
10	SaLo	Sandy loam
11	LoSa	Loamy sand
12	Si	Silt
13	Sa	Sand

Table 10: Texture classes of the Canadian (en) system / triangle

Or in French:

	abbr	name
1	ALo	Argile lourde
2	ALi	Argile limoneuse
3	A	Argile
4	AS	Argile sableuse
5	LLiA	Loam limono-argileux
6	LA	Loam argileux
7	LSA	Loam sablo-argileux
8	LLi	Loam limoneux
9	L	Loam
10	LS	Loam sableux
11	SL	Sable loameux
12	Li	Limon
13	S	Sable

Table 11: Texture classes of the Canadian (fr) system / triangle

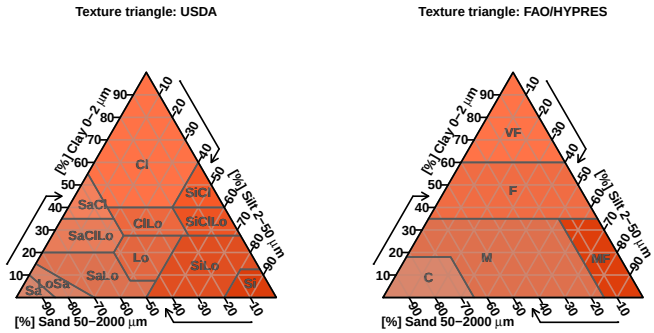
A reference image for this texture triangle can be found in this reference (not the one used for digitizing the triangle), and the boundaries have been checked using this reference[1].

4.11 Soil texture triangle with a texture classes color gradient

It is possible to have a nice color gradient (single hue, gradient of saturation and value) on the background, by setting the option `class.p.bg.col` (logical) to `TRUE`.

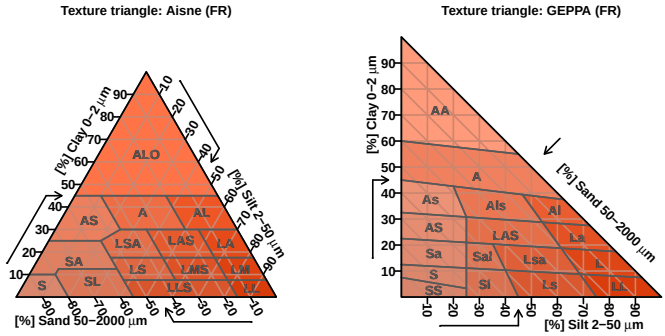
Example with the USDA and FAO soil texture triangles:

```
# Set a 2 by 2 plot matrix:
old.par <- par(no.readonly=T)
par("mfcol" = c(1,2),"mfrow"=c(1,2))
# Plot the triangles
TT.plot(
    class.sys       = "USDA.TT",
    class.p.bg.col  = TRUE
)   #
TT.plot(
    class.sys       = "FAO50.TT",
    class.p.bg.col  = TRUE
)   #
# Back to old parameters:
par(old.par)
```



Example with the French Aisne and French GEPPA soil texture triangles:

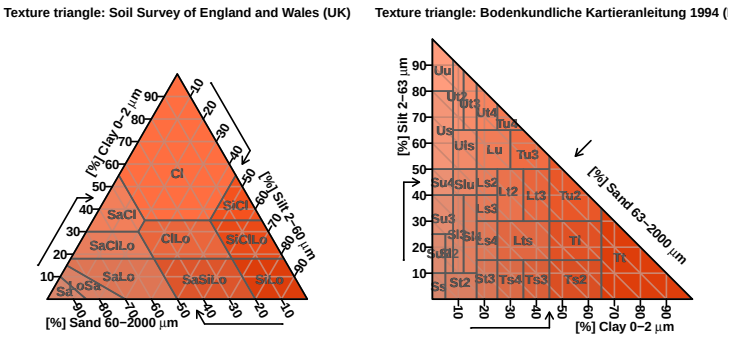
```
# Set a 2 by 2 plot matrix:
old.par <- par(no.readonly=T)
par("mfcol" = c(1,2),"mfrow"=c(1,2))
# Plot the triangles
TT.plot(
    class.sys       = "FR.AISNE.TT",
    class.p.bg.col  = TRUE
)   #
TT.plot(
    class.sys       = "FR.GEPPA.TT",
    class.p.bg.col  = TRUE
)   #
# Back to old parameters:
par(old.par)
```



Example with the UK (SSEW) and German (BK94) soil texture triangles:

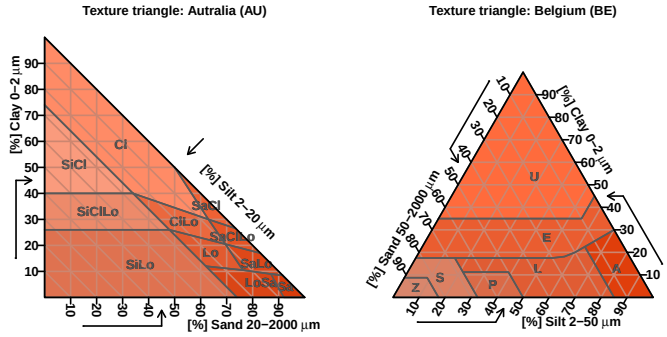
```
# Set a 2 by 2 plot matrix:
old.par <- par(no.readonly=T)
par("mfcol" = c(1,2),"mfrow"=c(1,2))
# Plot the triangles
TT.plot(
    class.sys       = "UK.SSEW.TT",
    class.p.bg.col  = TRUE
)   #
```

```
TT.plot(
    class.sys       = "DE.BK94.TT",
    class.p.bg.col  = TRUE
)   #
# Back to old parameters:
par(old.par)
```



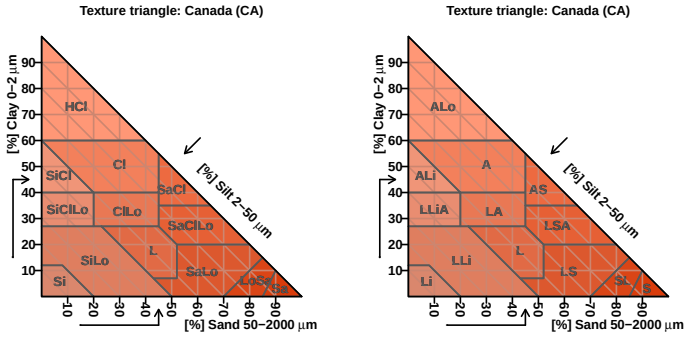
Example with the Australian and Belgian soil texture triangle:

```
# Set a 2 by 2 plot matrix:
old.par <- par(no.readonly=T)
par("mfcol" = c(1,2),"mfrow"=c(1,2))
# Plot the triangles
TT.plot(
    class.sys       = "AU.TT",
    class.p.bg.col  = TRUE
)   #
TT.plot(
    class.sys       = "BE.TT",
    class.p.bg.col  = TRUE
)   #
# Back to old parameters:
par(old.par)
```



And finally the Canadian texture triangle (with English and French abbreviations):

```
# Set a 2 by 2 plot matrix:
old.par <- par(no.readonly=T)
par("mfcol" = c(1,2),"mfrow"=c(1,2))
# Plot the triangles
TT.plot(
    class.sys       = "CA.EN.TT",
    class.p.bg.col  = TRUE
)   #
TT.plot(
    class.sys       = "CA.FR.TT",
    class.p.bg.col  = TRUE
)   #
# Back to old parameters:
par(old.par)
```



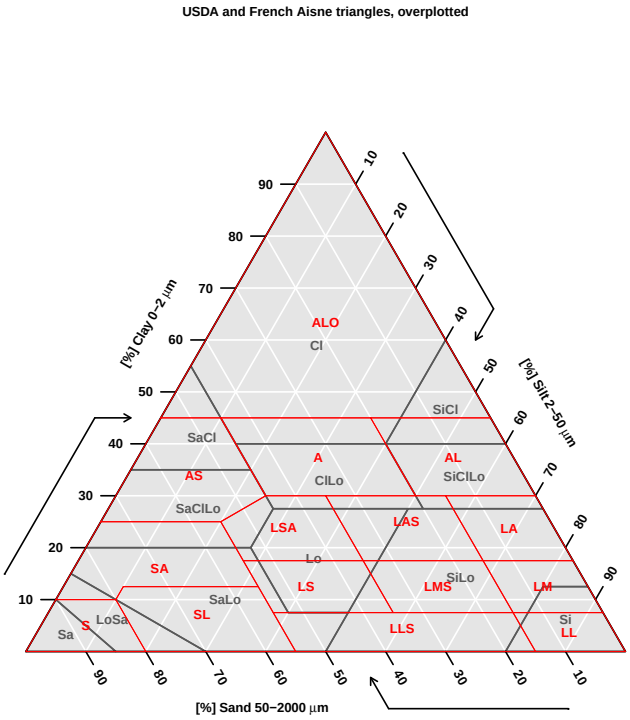
5 Overplotting two soil texture classification systems

5.1 Case 1: Overplotting two soil texture classification systems with the same geometry

Below is the code for plotting a French-Aisne texture triangle over a USDA texture triangle:

```
# First plot the USDA texture triangle, and retrieve its
#   geometrical features, silently outputted by TT.plot
geo <- TT.plot(
    class.sys   = "USDA.TT",
    main        = "USDA and French Aisne triangles, overplotted"
)   #
# Then overplot the French Aisne texture triangle,
#   and customise the colors so triangles are well distinct.
TT.classes(
    geo             = geo,
    class.sys       = "FR.AISNE.TT",
    # Additional "graphical" options
```

```
    class.line.col  = "red",
    class.lab.col   = "red",
    lwd.axis        = 2
)   #
```



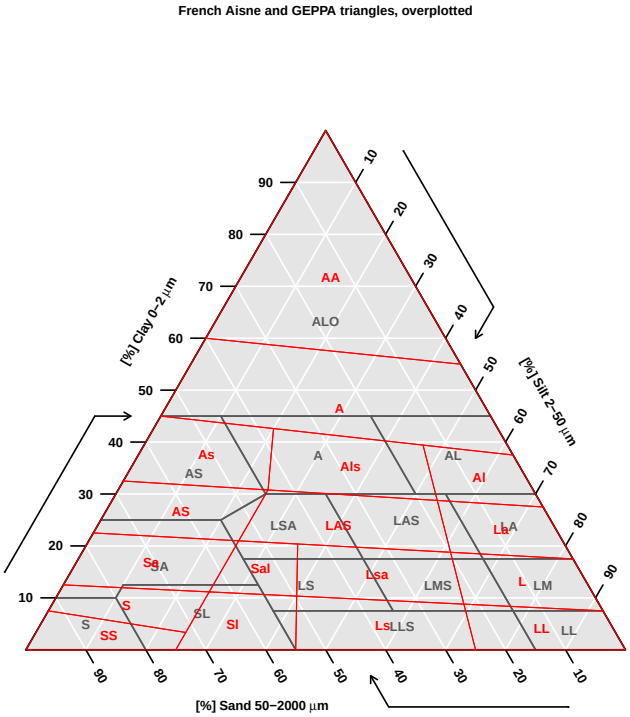
Beware that the result may not necessarily be very readable when printed, in black and white. Consider to change the line type as well (option `class.lty = 2` for `TT.classes`) is you want a more printer-friendly output.

5.2 Case 2: Overplotting two soil texture classification systems with different geometries

Below is the code to plot a French GEPPA texture triangle over a French Aisne texture triangle. The code is in fact almost identical to the previous case:

```
# First plot the USDA texture triangle, and retrieve its
#   geometrical features, silently outputted by TT.plot
geo <- TT.plot(
    class.sys   = "FR.AISNE.TT",
    main        = "French Aisne and GEPPA triangles, overplotted"
)   #
# Then overplot the French Aisne texture triangle,
#   and customise the colors so triangles are well distinct.
TT.classes(
    geo             = geo,
    class.sys       = "FR.GEPPA.TT",
    # Additional "graphical" options
```

```
    class.line.col  = "red",
    class.lab.col   = "red",
    lwd.axis        = 2
)   #
```



6 Plotting soil texture data

6.1 Simple plot of soil texture data

First, lets create a table containing (dummy) soil texture data, (in %), as well as dummy organic carbon content (in $g.kg^{-1}$, for later use):

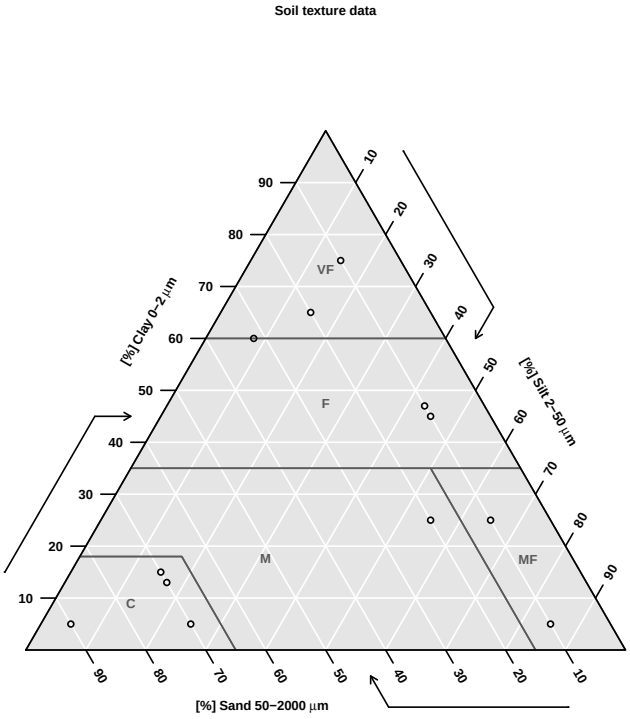
```
# Create a dummy data frame of soil textures:
my.text <- data.frame(
    "CLAY"  = c(05,60,15,05,25,05,25,45,65,75,13,47),
    "SILT"  = c(05,08,15,25,55,85,65,45,15,15,17,43),
    "SAND"  = c(90,32,70,70,20,10,10,10,20,10,70,10),
    "OC"    = c(20,14,15,05,12,15,07,21,25,30,05,28)
)   #
# Display the table:
my.text
```

```
  CLAY SILT SAND OC
1    5    5   90 20
2   60    8   32 14
3   15   15   70 15
```

```
4     5   25   70  5
5    25   55   20 12
6     5   85   10 15
7    25   65   10  7
8    45   45   10 21
9    65   15   20 25
10   75   15   10 30
11   13   17   70  5
12   47   43   10 28
```

The columns names include CLAY, SILT and SAND, so they are explicit for the `TT.plot` function. The code to display these soil texture data is:

```
TT.plot(
    class.sys   = "FAO50.TT",
    tri.data    = my.text,
    main        = "Soil texture data"
)   #
```

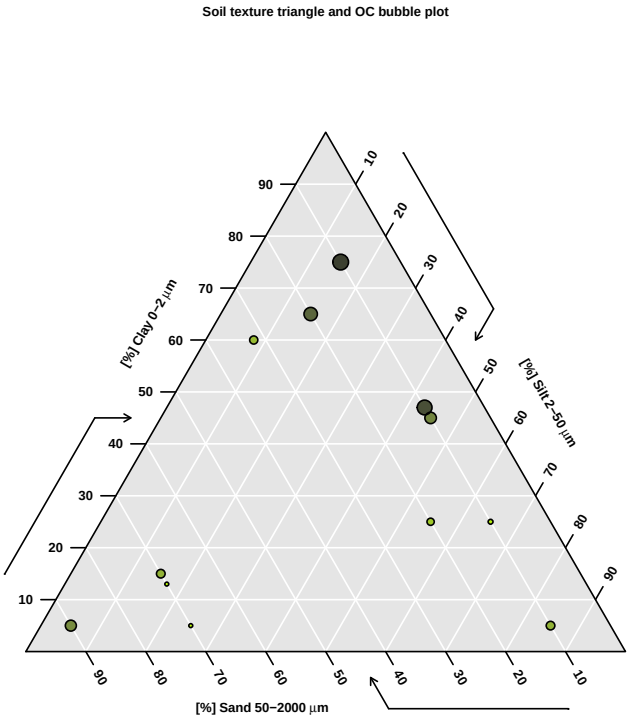


The option `tri.data` is a data frame containing numerical values. `colnames(tri.data)` must match with `blr.tex` option's values (default `c("CLAY","SILT","SAND")`). More columns can be provided, but are not used unless other options are chosen (see below).

6.2 Bubble plot of soil texture data and a 3rd variable

It could be interesting to plot the organic carbon content on top of the soil texture triangle. Bubble plots are good for this:

```
TT.plot(
    class.sys   = "none",
    tri.data    = my.text,
    z.name      = "OC",
    main        = "Soil texture triangle and OC bubble plot"
)   #
```



The option `z.name` is a character string, the name of the column in `tri.data` that contains a 3rd variable to be plotted.

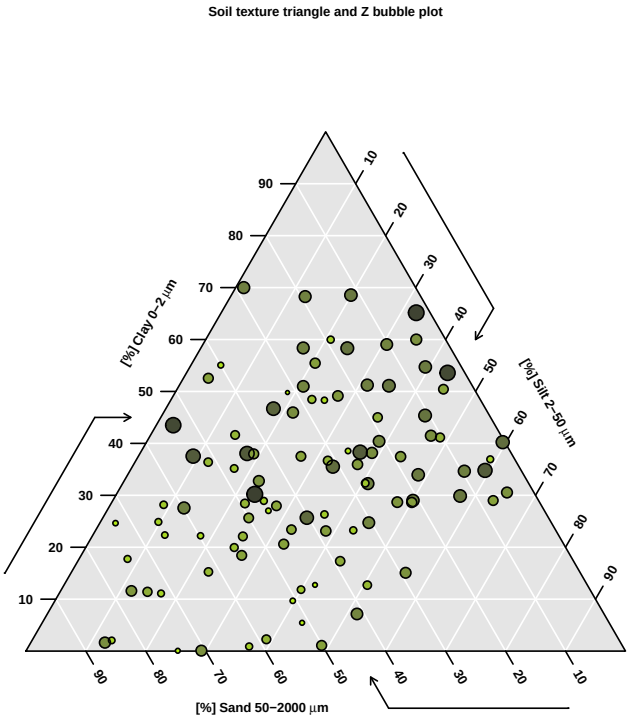
The 3rd variable is plotted with an 'expansion' factor proportional to `z.name` value. Low values have a small diameter and high values have a big diameter. To re-enforce the visual effect, a single hue color gradient is added to the point background, with hight saturation and high color's value (bright) for low `z.name`'s values, and low saturation and low color's value (dark) for high `z.name`'s values.

The function keeps good visual effect, even with a lot of values. Below is a test using `TT.dataset()` function, that generate a (quick and dirty) dummy soil texture datasets, with a 4th z variable (named 'Z'), correlated to the texture data.

```
rand.text        <- TT.dataset(n=100,seed.val=1980042401)
```

```
TT.plot(
    class.sys   = "none",
    tri.data    = rand.text,
```

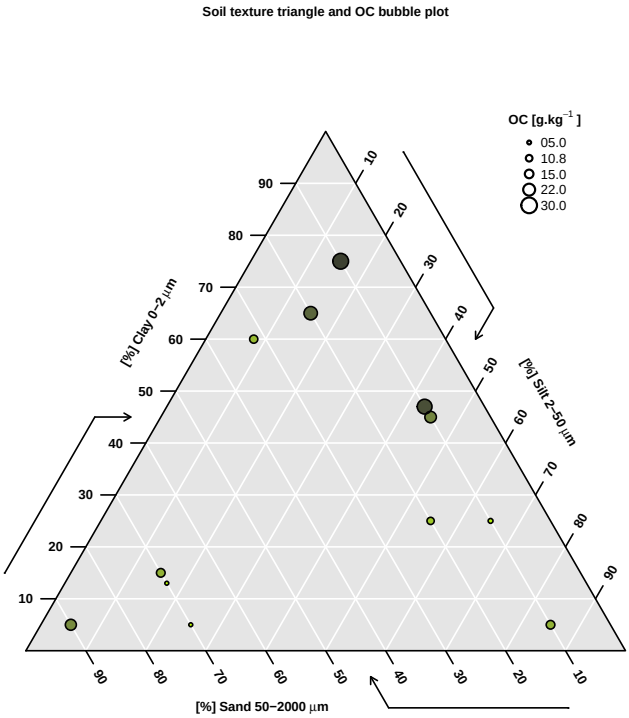
```
    z.name      = "Z",
    main        = "Soil texture triangle and Z bubble plot"
)   #
```



This function is primarily intended for exploratory data analysis or for rather qualitative analysis, as it is difficult for the reader to know the real `z.name` value of a point. It is nevertheless possible to add 'manually' a legend, as in the example below:

```
TT.plot(
    class.sys   = "none",
    tri.data    = my.text,
    z.name      = "OC",
    main        = "Soil texture triangle and OC bubble plot"
)   #
# Recompute some internal values:
z.cex.range <- TT.get("z.cex.range")
def.pch     <- par("pch")
def.col     <- par("col")
def.cex     <- TT.get("cex")
oc.str      <- TT.str(
    my.text[,"OC"],
    z.cex.range[1],
    z.cex.range[2]
)   #
# The legend:
legend(
```

```
    x           = 80,
    y           = 90,
    title       =
        expression( bold('OC [g.kg'^-1 ~ ']') ),
    legend      = formatC(
        c(
            min( my.text[,"OC"] ),
            quantile(my.text[,"OC"] ,probs=c(25,50,75)/100),
            max( my.text[,"OC"] )
        ),
        format  = "f",
        digits  = 1,
        width   = 4,
        flag    = "0"
    ),  #
    pt.lwd      = 4,
    col         = def.col,
    pt.cex      = c(
            min( oc.str ),
            quantile(oc.str ,probs=c(25,50,75)/100),
            max( oc.str )
    ),  #,
    pch         = def.pch,
    bty         = "o",
    bg          = NA,
    box.col     = NA,
    text.col    = "black",
    cex         = def.cex
)   #
```



This code is obviously complicated, but it produces a smart legend. It is not possible (or easy) to add an automatic legend to a plot, because the optimal number of decimals may change from dataset to dataset, as well as the quantiles displayed.

6.3 Heatmap and / or contour plot of soil texture data and a 4th variable

Another way to explore a 4th variable is heatmap. The heatmap represent a local average value (by inverse distance interpolation) of the 4th variable in the form of a colored map.

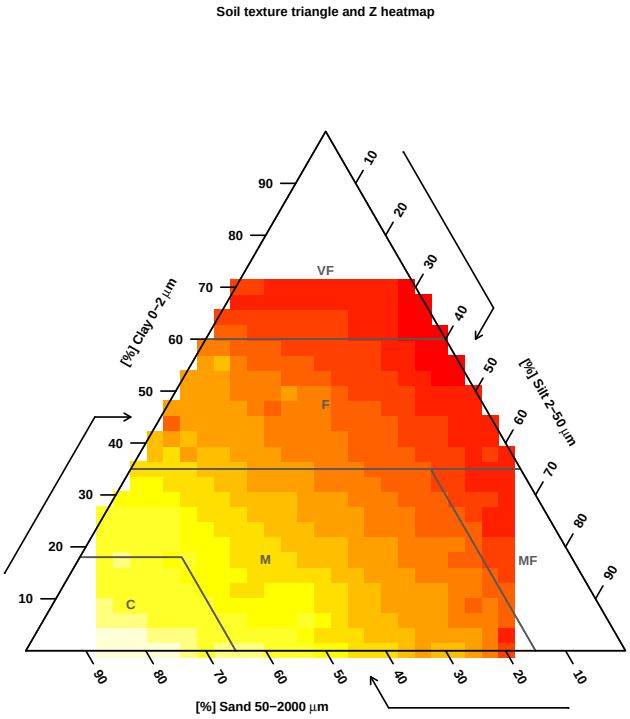
Plotting a heatmap now follows 4 steps, that somehow works as 'sandwich' plots:

- (1) Retrieve the geometrical parameters of the future plot with `TT.geo.get()` function. It doesn't plot anything, but returns geometrical parameters that will be used to determine the x-y grid on which calculating the inverse distance. A call to `geo <- TT.plot()` would also work.
- (2) Calculate inverse weighted distances of the 4th variable (here 'Z') on a regular x-y grid, using `TT.iwd()` function. It returns a grid with interpolated values.
- (3) Plot this grid with the function `TT.image()` (or with `TT.contour()`). This function is a wrapper for the `image()` (or `TT.contour()`) function, adapted to triangle plots. The grid format is compatible with `image()` or

`TT.contour()`. `TT.image()` can have an option `add = TRUE` to plot the image on top of an existing triangle plot.

- (4) Add a standard triangle plot on to of the heatmap, using the standard `TT.plot()` function (with `add = TRUE`). If plot has been called in step 1, step 4 is not necessary, and the heatmap is plotted on top of the existing triangle.

```
geo <- TT.geo.get()
#
iwd.res <- TT.iwd(
    geo         = geo,
    tri.data    = rand.text,
    z.name      = "Z",
)   #
#
TT.image(
    x       = iwd.res,
    geo     = geo,
    main    = "Soil texture triangle and Z heatmap"
)   #
#
TT.plot(
    geo         = geo,
    grid.show   = FALSE,
    add         = TRUE  #  <<-- important
)   #
```



`TT.iwd()` has 3 important parameters:

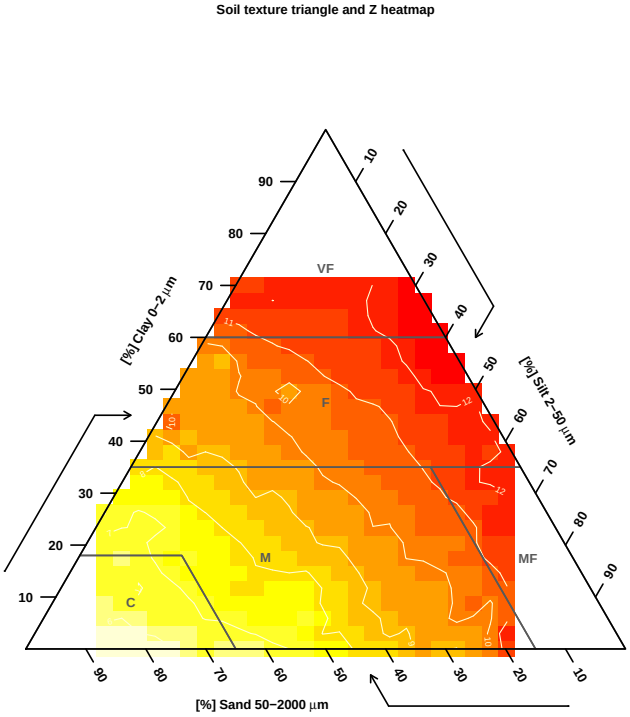
- (1) `pow` (default value 0.5) is the power used for the inverse weighted distance interpolation. Low values means strong smoothing, and vice versa;
- (2) `q.max.dist` (default value 0.5) is used to determines the maximum (Euclidian) distance of the points used to calculate interpolated values. Data points located further than that distance are not used. `q.max.dist` is the quantile of the Euclidian distance, so 0.5 means that points located further that the 50% quantile of all Euclidian distances will not be used to calculate a given grid value (notice that this is very experimental!). The higher the value, the more points used to calculate the interpolated values (and the stronger the smoothing);
- (3) `n` is the number of x and y values used to calculate the interpolation grid. The number of nodes in the grid is n^2.

`TT.image()` accepts most of the options existing in `image()`.

The is no 'heatmap legend', but it is possible to add a contour plot to the existing plot, in order to replace the color legend:

```
TT.image(
    x       = iwd.res,
    geo     = geo,
    main    = "Soil texture triangle and Z heatmap"
)   #
#
```

```
TT.contour(
    x       = iwd.res,
    geo     = geo,
    add     = TRUE, #  <<-- important
    lwd     = 2
)   #
#
TT.plot(
    geo         = geo,
    grid.show   = FALSE,
    add         = TRUE  #  <<-- important
)   #
```



`TT.contour()` accepts most of the options existing in `contour()`.

Inverse Weighted Distance interpolation is not really 'state of the art' statistics, but rather a visual way of exploring the data. Interpolation is NOT done on a clay / silt / sand mesh, but rather on a x-y grid in the triangle. So data density is not equal between clay, silt and sand. Moreover, the interpolator might not be the most relevant one.

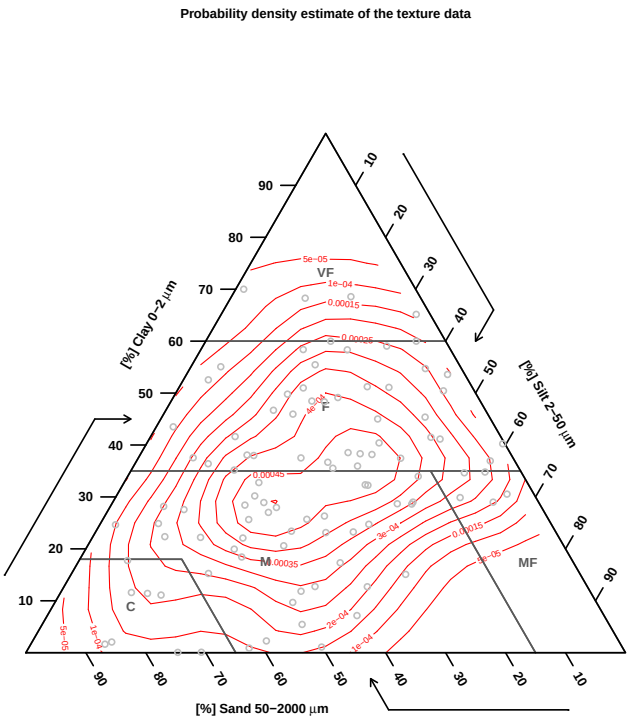
This function is only provided as 'experimental' and it is susceptible to be modified significantly in the future.

6.4 Two-dimensional kernel (probability) density estimation for texture data

The `kde2d()` function of the MASS package by W. N. Venables and B. D. Ripley[30], for 2D kernel probability density estimate has been 'wrapped' into the function `TT.kde2d()` so it becomes usable with texture data and texture triangle. It returns an x-y-z list / grid object that can be plotted with `TT.contour()` or `TT.image()`.

The same 'sandwich' plot structure as for inverse weight distance estimate of a 4th variable is also valid for contour plot of probability density estimates:

```
geo <- TT.geo.get()
#
kde.res <- TT.kde2d(
    geo         = geo,
    tri.data    = rand.text
)   #
#
TT.contour(
    x       = kde.res,
    geo     = geo,
    main    = "Probability density estimate of the texture data",
    lwd     = 2,
    col     = "red"
)   #
#
TT.plot(
    tri.data    = rand.text,
    geo         = geo,
    grid.show   = FALSE,
    add         = TRUE, #  <<-- important
    col         = "gray"
)   #
```



Using `TT.image()` would also work here.

As `kde2d()`, `TT.kde2d()` accepts a 'n' option that determines the number of values in the x and y axes (The total number of nodes is n^2). The parameter 'h' from `kde2d()` has NOT been implemented into `TT.kde2d()`, and the default calculation method is used.

Please note that the probability density is estimated on the x-y grid of the plot, and NOT on the clay / silt / sand coordinates system. So a different plot geometry may give a slightly different probability density estimate...

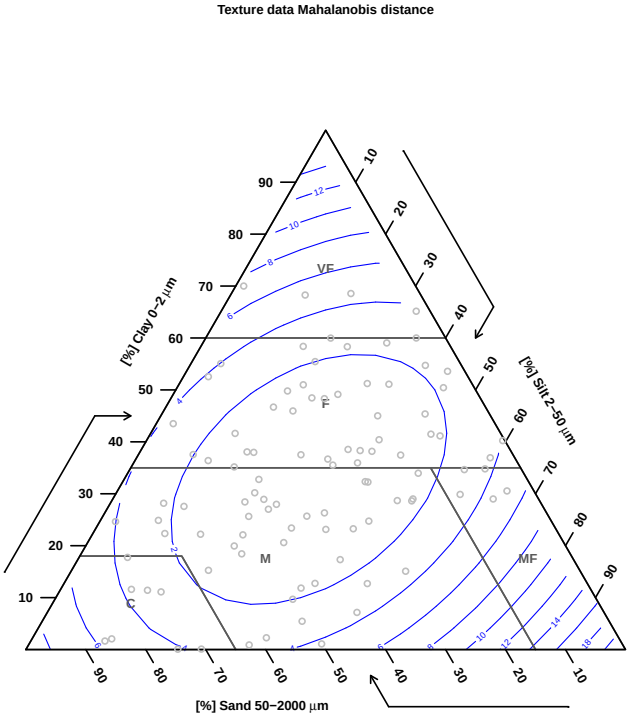
6.5 Contour plot of texture data Mahalanobis distance

The `mahalanobis()` function (part of the default R functions) has been 'wrapped' into the function `TT.mahalanobis()` so it becomes usable with texture data and texture triangle. It returns an x-y-z list / grid object that can be plotted with `TT.contour()` or `TT.image()`.

Some authors[16] have recommended that the Mahalanobis distance should be computed on the additive log-ratio transform of soil texture data in order to take into account the fact the 3 texture classes are not independent random variables (but rather compositional data). For this reason an option has been added that transform the texture data by an additive log-ratio prior to the computation of the Mahalanobis distance (the default is no transformation of the data). The log-ratio transformation code used here has been taken from the 'chemometrics' package[11] by Filzmoser and Varmuza (function `alr()`).

The same 'sandwich' plot structure as for inverse weight distance estimate of a 4th variable is also valid for contour plot of probability density estimates. Below is a first example without texture transformation:

```
geo <- TT.geo.get()
#
maha <- TT.mahalanobis(
    geo         = geo,
    tri.data    = rand.text
)   #
#
TT.contour(
    x       = maha,
    geo     = geo,
    main    = "Texture data Mahalanobis distance",
    lwd     = 2,
    col     = "blue"
)   #
#
TT.plot(
    tri.data    = rand.text,
    geo         = geo,
    grid.show   = FALSE,
    add         = TRUE, #  <<-- important
    col         = "gray"
)   #
```



All the options of `mahalanobis()` are also available in `TT.mahalanobis()`. The option `divisorvar` is an integer that determines which texture classes (number 1, 2 or 3 in 'css.names') is NOT used to calculate the Mahalanobis distance (using the 3 texture classes crashes the `mahalanobis()` function).

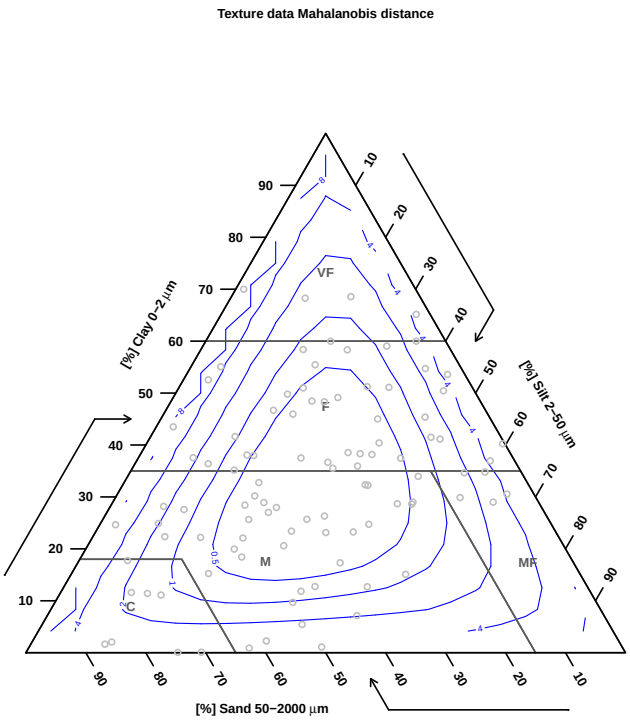
If you want to compute the Mahalanobis distance on texture data transformed with an additive log-ratio, you can set the option `alr = TRUE` (default = FALSE). The option `divisorvar` is then an integer used in the log-ratio transformation of texture data,

$$alr(textureClass_i) = log_{10}(textureClass_i / textureClass_{divisorvar})$$

where `i` and `divisorvar` are index of `css.names`. The Mahalanobis distance is then computed on 2 of the 3 log-ratio transformed texture classes (`css.names[-divisorvar]`).

Below is an example of Mahalanobis distance plot with log-ratio transformation:

```
geo <- TT.geo.get()
#
maha <- TT.mahalanobis(
    geo         = geo,
    tri.data    = rand.text,
    alr         = TRUE  #  <<-- important
)   #
#
TT.contour(
    x       = maha,
    geo     = geo,
    main    = "Texture data Mahalanobis distance",
    lwd     = 2,
    col     = "blue",
    levels  = c(0.5,1,2,4,8)  #  <<-- manually set. Otherwise
)   #                                 ugly plot
#
TT.plot(
    tri.data    = rand.text,
    geo         = geo,
    grid.show   = FALSE,
    add         = TRUE,  #  <<-- important
    col         = "gray"
)   #
```



The Mahalanobis distances computed on a regular x-y grid have an extremely skewed distribution, with a few very high values near the borders of the triangle. For this reason the automatic levels of the contour function fails to show anything relevant, and it is recommended that the user manually set the levels, as in the example.

Please notice that the `TT.mahalanobis()` has not been tested extensively for practical and theoretical validity.

Using `TT.image()` would also work here.

6.6 Plotting text in a texture triangle

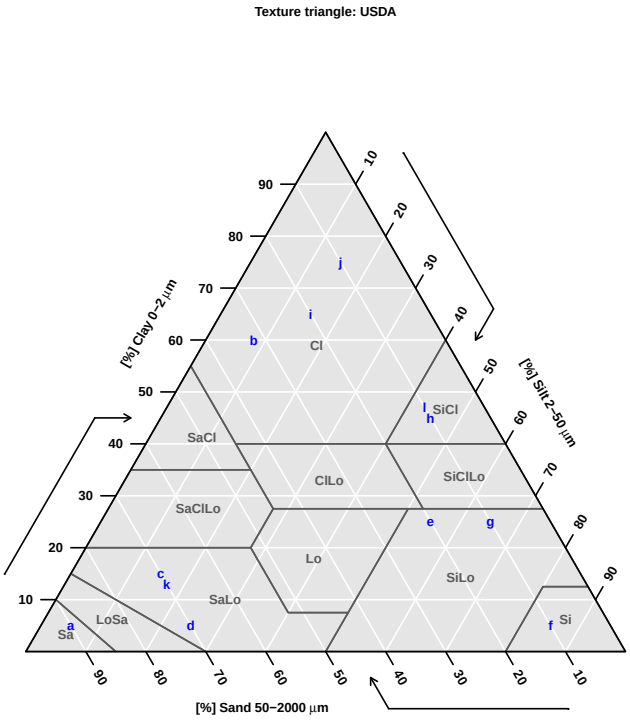
As the `text()` function of R standard plot functions, `TT.plot()` is completed by a `TT.text()` function that displays text into an existing texture triangle plot. Its use is similar to `TT.points()`, apart that it has a `labels`, and a `font` option, as the `text()` function. Below is a simple example:

```
# Display the USDA texture triangle:
geo     <- TT.plot(class.sys="USDA.TT")
# Create some custom labels:
labelz  <- letters[1:dim(my.text)[1]]
labelz
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

```
# Display the text
TT.text(
```

```
    tri.data    = my.text,
    geo         = geo,
    labels      = labelz,
    font        = 2,
    col         = "blue"
)   #
```



As for the `text()` function, it is also possible to set `adj`, `pos` and / or `offset` parameters (not shown here).

7 Control of soil texture data in The Soil Texture Wizard

Several controls are done (internally) on soil texture data prior to soil texture plots or soil texture classification:

- Clay, silt and sand column names must correspond to the names given in the option `css.names` (default to CLAY, SILT and SAND);
- There should not be any **negative** values in clay, silt and sand (i.e. values that lies outside the triangle). This control can be relaxed by setting the option `tri.pos.tst` to FALSE;
- All the row sums of the 3 texture classes must be equal to **text.sum** (generally 100, for 100%). In fact, (absolute) differences lower than **text.sum**

*** text.tol** are allowed (with **text.tol** option default to 1/1000, so textures sum must be between 99.9 and 100.1). This text can be relaxed by setting `tri.sum.tst` to FALSE;

- No missing values are allowed in the texture data (NA).

A test of the data can be conducted externally, using `TT.data.test`. An error occur if the data don't pass the tests:

```
TT.data.test( tri.data = rand.text )
```

This function accepts options `css.names`, `text.sum`, `text.tol`, `tri.sum.tst` and `tri.pos.tst`.

7.1 Normalizing soil texture data (sum of the 3 texture classes)

If you have a texture data table with some rows where the sum of the 3 texture classes is not 100%, but you know this is not due to errors in the data, you may want to normalize the sum of the 3 texture classes to 100%. The function `TT.normalise.sum` do that for you, and return a data table with normalised clay, silt and sand values. The option `residuals` can be set to TRUE if you want the residuals to be returned (initial row sum - final row sum):

```
res <- TT.normalise.sum( tri.data = rand.text )
#
# With output of the residuals:
res <- TT.normalise.sum(
    tri.data    = rand.text,
    residuals   = TRUE  #  <<-- default = FALSE
)   #
#
colnames( rand.text )
```

```
[1] "CLAY" "SILT" "SAND" "Z"
```

```
colnames( res )  #  "Z" has been dropped
```

```
[1] "CLAY"      "SILT"      "SAND"      "residuals"
```

```
max( res[ , "residuals" ] )
```

```
[1] 2.842171e-14
```

7.2 Normalizing soil texture data (sum of X texture classes)

[function and section to be written]

8 Classify soil texture data: TT.points.in.classes()

The function `TT.points.in.classes()` classify a table of soil texture data (`tri.data`) and returns a table where each row is one soil texture sample, and each column a soil texture class (given the system `class.sys`). Values are 0

when the point is 'out' of the class, 1 when 'in', 2 when 'on a polygon side' and 3 when 'on the polygon corner(s) (vertex / vertices)' (As in the underying function `point.in.polygon()` from the 'sp' package). In the examples below I will only show the results for the 5 first row of the dummy soil texture data created above, with the FAO classification:

```
TT.points.in.classes(
    tri.data    = my.text[1:5,],
    class.sys   = "FAO50.TT"
)   #
```

```
     VF F M MF C
[1,]  0 0 0  0 1
[2,]  2 2 0  0 0
[3,]  0 0 0  0 1
[4,]  0 0 0  0 1
[5,]  0 0 1  0 0
```

A major interest of the function resides in the fact that it is possible to use another classication very easily, USDA in the xample below:

```
TT.points.in.classes(
    tri.data    = my.text[1:5,],
    class.sys   = "USDA.TT"
)   #
```

```
     Cl SiCl SaCl ClLo SiClLo SaClLo Lo SiLo SaLo Si LoSa Sa
[1,]  0    0    0    0      0      0  0    0    0  0    0  1
[2,]  1    0    0    0      0      0  0    0    0  0    0  0
[3,]  0    0    0    0      0      0  0    0    1  0    0  0
[4,]  0    0    0    0      0      0  0    0    1  0    0  0
[5,]  0    0    0    0      0      0  0    1    0  0    0  0
```

The result can also be returned in a logical form with the option `PiC.type = "l"` (for 'logical'. default is "n" as numeric). Value is TRUE if the sample belong to the class, and FALSE if it is outside the class. In case of a point located at the border of two or more texture classes, several texture classes (columns) are marked TRUE.

```
TT.points.in.classes(
    tri.data    = my.text[1:5,],
    class.sys   = "FAO50.TT",
    PiC.type    = "l"
)   #
```

```
       VF     F     M    MF     C
[1,] FALSE FALSE FALSE FALSE  TRUE
[2,]  TRUE  TRUE FALSE FALSE FALSE
[3,] FALSE FALSE FALSE FALSE  TRUE
[4,] FALSE FALSE FALSE FALSE  TRUE
[5,] FALSE FALSE  TRUE FALSE FALSE
```

And finally, the results can be a vector of character, of the same length as the number of soil samples, and containing the abbreviation of the texture class(es) to which the sample belongs. In case of a sample lying on the border of two classes, the classes abbreviation are concatenated (separated by a comma).

```
TT.points.in.classes(
    tri.data    = my.text[1:5,],
    class.sys   = "FAO50.TT",
    PiC.type    = "t"
)   #
```

```
[1] "C"     "VF, F" "C"     "C"     "M"
```

Notice that the second value lies between two classes, and that they are outputted separated by a comma.

The comma separator can be replaced by any character string, as in the function `paste()`, with the option `collapse`:

```
TT.points.in.classes(
    tri.data    = my.text[1:5,],
    class.sys   = "FAO50.TT",
    PiC.type    = "t",
    collapse    = ";"
)   #
```

```
[1] "C"    "VF;F" "C"    "C"    "M"
```

9 Converting soil texture data and systems with different silt-sand particle size limit

'The Soil Texture Wizard' comes with functions to transform soil textures data from 1 particle sizes system (limits between the clay, silt and sand particles) to another particle size system, with a log-linear transformation. For instance, it is possible to convert a textures data table measured in a system that have a silt / sand limit is 60μm into a system that has a silt / sand limit is 50μm.

It is important to keep in mind several limitations when transforming soil texture data:

- Transforming soil texture with a 'log-linear interpolation' consider that the cumulated particle size (mass) distribution is linear between two consecutive particle size classes limits, when plotted against a log transform of the particle size;
- Because of this, **transforming soil texture is at best an approximation of what would be obtained with laboratory measurements**;
- The bigger the difference between two particle size limit used to interpolate a new particle size limit, the more uncertain the estimation (= the bigger the errors);

- Because of this, the more particle size classes you have in the initial soil texture data (i.e. the smaller the differences between 2 successive particle size classes limits), the more precise the transformation.
- Transforming soil texture data using a log-linear interpolation is not the most precise method (especially if you have more than 3 particle size classes). On the other hand, it is certainly the most simple method. See Nemes et al. 1999 [22] for a comparison of different methods for soil texture data transformation.

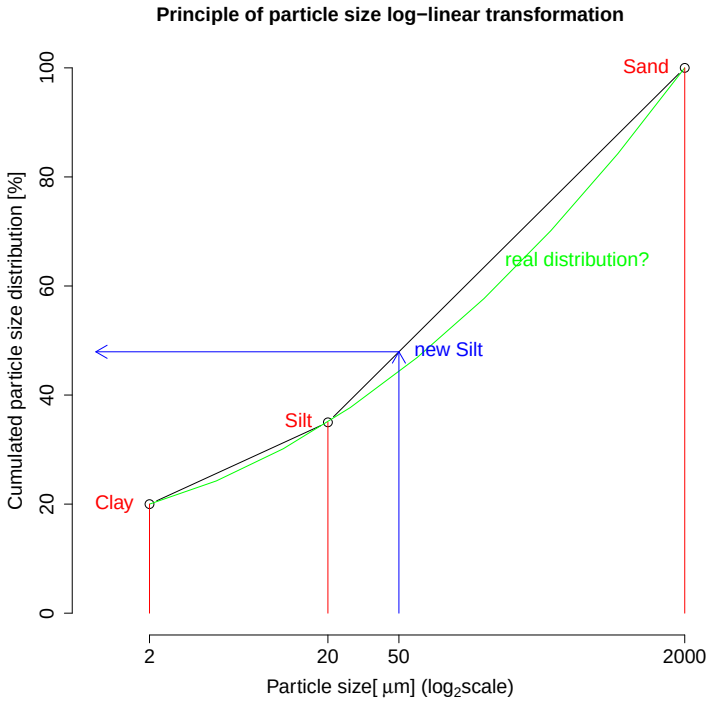
This package comes with 2 functions for texture transformations:

- `TT.text.transf()`, that only works with 3 particle size classes, clay, silt and sand. It can be used independently, for transforming a table of soil texture data, but it is also 'embedded' into `TT.plot()` and `TT.points.in.classes()` to allow transparent, on the fly transformation of soil texture data or soil texture triangles / classification.
- `TT.text.transf.X()`, that works with 3 or more particle size classes. The number of particle size classes in the input data do not need to be equal to the number of particle size classes in the final system (output). It is not embedded and not embeddable into `TT.plot()` and `TT.points.in.classes()`, and it is not doing as many data consistency tests as `TT.text.transf()`.

Of course, it is also possible to define your own texture transformation function. If this function works on clay, silt and sand, and if it has the same options as `TT.text.transf()`, it can also be embedded in `TT.plot()` and `TT.points.in.classes()` by changing a simple option.

If your data have more than 3 particle size classes, you should probably use `TT.text.transf.X()` instead of `TT.text.transf()`.

The figure below / above illustrate how the log-linear interpolation works, and why it is sometimes / often inaccurate.



9.1 Transforming soil texture data (from 3 particle size classes)

Here are the non transformed data (reminder):

```
my.text[1:5,]
```

```
  CLAY SILT SAND OC
1    5    5   90 20
2   60    8   32 14
3   15   15   70 15
4    5   25   70  5
5   25   55   20 12
```

Now the (dummy) data will be transformed, assuming that they have been measured with a 63μmeters silt-sand particle size limit, and that we want them to be with a 50μmeters silt-sand limit. Please don't forget this is not an 'exact' transformation, but rather an estimation:

```
TT.text.transf(
        tri.data        = my.text[1:5,],
        base.css.ps.lim = c(0,2,50,2000),
        dat.css.ps.lim  = c(0,2,63,2000)
)   #
```

```
  CLAY      SILT     SAND OC
1    5  4.665054 90.33495 20
2   60  7.464087 32.53591 14
```

```
3   15 13.995163 71.00484 15
4    5 23.325271 71.67473  5
5   25 51.315597 23.68440 12
```

Lets create a copy of the dummy data table, with new French columns names.

```
# Copy the data.frame
my.text.fr  <- my.text
# Curent columns names:
colnames(my.text.fr)
```

```
[1] "CLAY" "SILT" "SAND" "OC"
```

```
# New columns names:
colnames(my.text.fr) <- c("ARGILE","LIMON","SABLE","CO")
```

It is also possible to transform it:

```
TT.text.transf(
    tri.data        = my.text.fr[1:5,],
    base.css.ps.lim = c(0,2,50,2000),
    dat.css.ps.lim  = c(0,2,63,2000),
    css.names       = c("ARGILE","LIMON","SABLE")
)   #
```

```
  ARGILE     LIMON    SABLE CO
1      5  4.665054 90.33495 20
2     60  7.464087 32.53591 14
3     15 13.995163 71.00484 15
4      5 23.325271 71.67473  5
5     25 51.315597 23.68440 12
```

As you can see, OC values are kept and returned untransformed in the outputted data.frame.

9.2 Transforming soil texture data (from 3 or more particle size classes)

When more than 3 particle size classes are present in the dataset, it is sometimes necessary (and anyway recommended) to use `TT.text.transf.X()` instead of `TT.text.transf()`. But `TT.text.transf.X()` is not performing as much 'data consistency' checks as `TT.text.transf()`. In particular:

- The option `tri.data` should be a data.frame with only soil texture data (no additional extra columns should be present). It will not necessary bug or warn if more data are provided (although there are good chances that it bugs because the sum of textures is not 100%);
- The columns in the `tri.data` data.frame should be **in ascending order of particle size**. If the columns are provided in another order, it will not bug or warn;
- The length of the particle size classes limits should be equal to the number of columns in `tri.data` + 1 (from the lower limit of the 1st class to the upper limit of the upper class).

We need first to create a dummy dataset with more than 3 particle size classes:

```
# Create a random fraction between 0 and 1
r.frac <- runif(n=dim(my.text)[1])
#
my.text4 <- cbind(
    "CLAY"          = my.text[,"CLAY"],
    "FINE_SILT"     = my.text[,"SILT"] * r.frac,
    "COARSE_SILT"   = my.text[,"SILT"] * (1-r.frac),
    "SAND"          = my.text[,"SAND"]
)   #
#
my.text4[1:5,]
```

```
     CLAY FINE_SILT COARSE_SILT SAND
[1,]    5  1.993923  3.00607682   90
[2,]   60  7.983799  0.01620122   32
[3,]   15  2.907617 12.09238343   70
[4,]    5 10.414043 14.58595668   70
[5,]   25 34.866125 20.13387494   20
```

Transform this data frame, from a system where the silt - sand limit is at 63μm to a system where the silt - sand limit is at 50μm:

```
TT.text.transf.X(
    tri.data        = my.text4[1:5,],
    base.ps.lim = c(0,2,20,50,2000),
    dat.ps.lim  = c(0,2,20,63,2000)
)   #
```

```
  C1        C2          C3       C4
1  5  1.993923  2.40058780 90.60549
2 60  7.983799  0.01293795 32.00326
3 15  2.907617  9.65671534 72.43567
4  5 10.414043 11.64802889 72.93793
5 25 34.866125 16.07847618 24.05540
```

Notice the differences in options name as compared to `TT.text.transf()`. Notice also that the columns names in the input data are not preserved in the output data (There is not systematically a correspondence between input and output).

As said before, the number of particle size classes in the input data do not need to be identical to the number of particle size classes in the output:

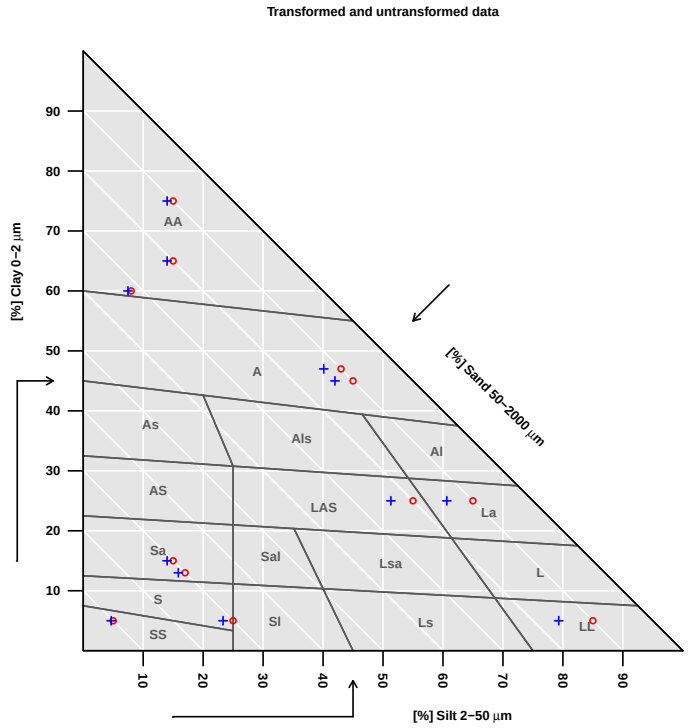
```
TT.text.transf.X(
    tri.data        = my.text4[1:5,],
    base.ps.lim = c(0,2,50,2000),
    dat.ps.lim  = c(0,2,20,63,2000)
)   #
```

```
  C1        C2       C3
1  5  4.394511 90.60549
2 60  7.996737 32.00326
3 15 12.564332 72.43567
4  5 22.062072 72.93793
5 25 50.944601 24.05540
```

9.3 Plotting and transforming 'on the fly' soil texture data

It is possible to plot data on a triangle and transform them 'on the fly', if they have been measured in another particle size system (classes sizes) as the triangle:

```
# First, plot the data without transformation:
geo <- TT.plot(
    class.sys   = "FR.GEPPA.TT",
    tri.data    = my.text,
    col         = "red",
    main        = "Transformed and untransformed data"
)   #
# Then, re-plot them with transformation:
TT.points(
    tri.data        = my.text,
    geo             = geo,
    dat.css.ps.lim  = c(0,2,63,2000),
    css.transf      = TRUE,
    col             = "blue",
    pch             = 3
)   #
```



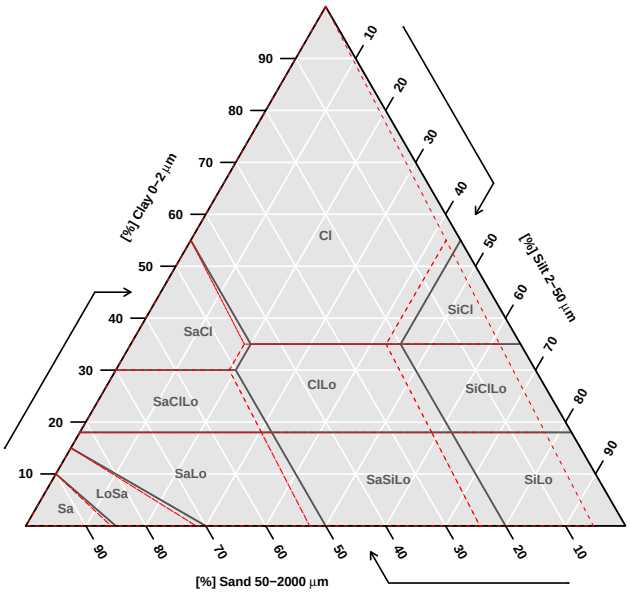
Notice that the `dat.css.ps.lim` and `css.transf` options could be used directly with `TT.plot()` (not shown here).

9.4 Plotting and transforming 'on the fly' soil texture triangles / classification

The example below shows how it is possible to project the UK soil texture triangle in a particle size system that has 0, 2, 50 and 2000μm as reference. The background triangle (gray) is in fact NOT transformed, and plotted 'as it is', without taking care of the real particle size limits (**because the default value of the `css.transf` option is `FALSE`**). The second plot (red) is transformed as it should have been (because the UK triangle has a 60μm silt sand limit).

```
# Not transformed
geo <- TT.plot(
    class.sys   = "UK.SSEW.TT",
    base.css.ps.lim = c(0,2,50,2000),
    main        =
        "Dummy transformation of the UK texture triangle"
)   #
# Transformed
TT.classes(
    geo             = geo,
    class.sys       = "UK.SSEW.TT",
    css.transf      = TRUE,
    # Additional "graphical" options
    class.line.col  = "red",
```

```
    class.lab.col   = "red",
    lwd.axis        = 2,
    class.lab.show  = "none",
    class.lty       = 2
)   #
```

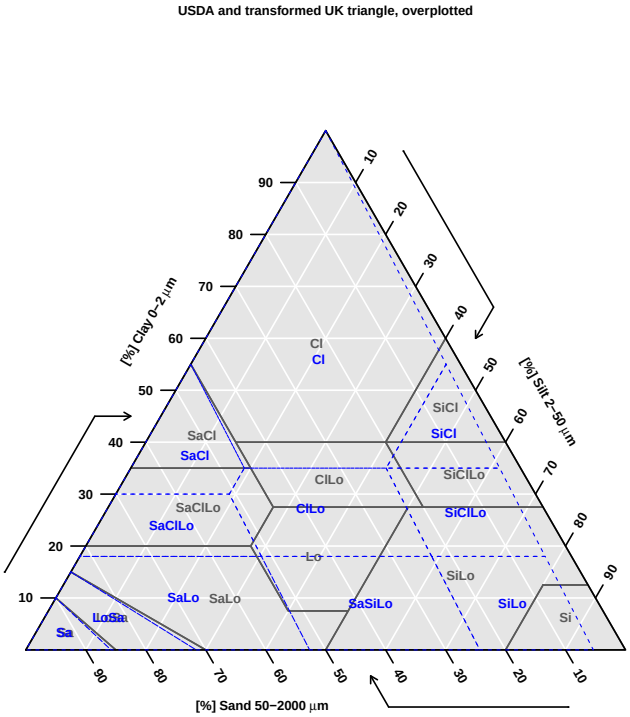


We clearly see that clay content is unaffected, but there is a 'gap' created by the fact that 100% silt become 'less than 100% silt and some Sand (Particles that were considered as silt in the original UK system become sand in the 50μm system)'! So the texture triangle is compacted toward the sand side.

In a second example we can show how to compare a USDA soil texture triangle, and a UK soil texture triangle re-projected in the same particle size system, 50μm (while the UK triangle is based on a 60μm limit):

```
# No transformation needed or stated
geo <- TT.plot(
    class.sys   = "USDA.TT",
    main        =
        "USDA and transformed UK triangle, overplotted"
)   #
# Transformed
TT.classes(
    geo             = geo,
    class.sys       = "UK.SSEW.TT",
    css.transf      = TRUE,  #  <<-- important
    # Additional "graphical" options
```

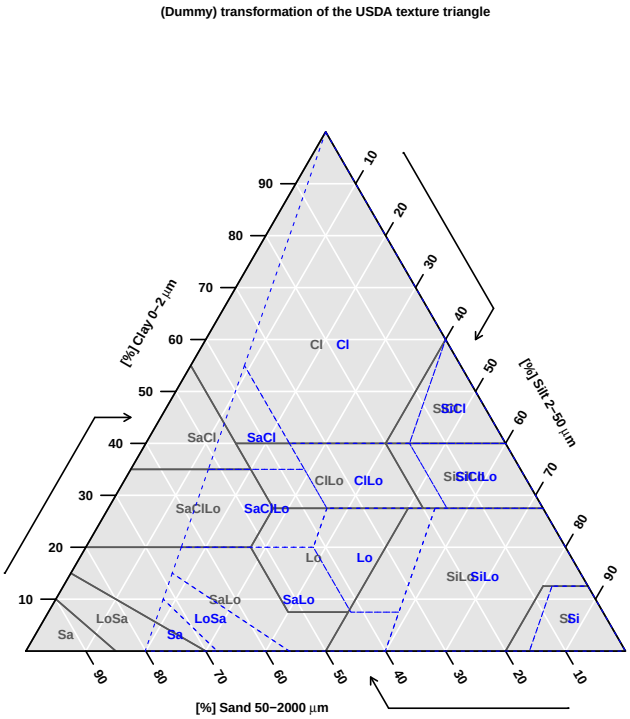
```
    class.line.col  = "blue",
    class.lab.col   = "blue",
    lwd.axis        = 2,
    class.lty       = 2
)   #
```



Now another test where the background silt sand limit is 50μmeters and the 2nd triangle is 'said to have' 20μmeters limit. Two USDA texture triangles. The 1st is not transformed (because it is not needed), while the 2nd is transformed:

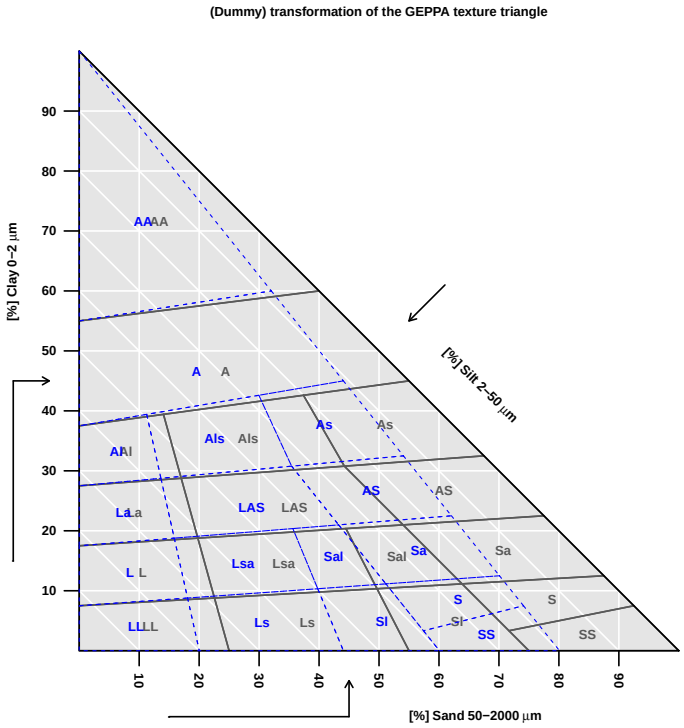
```
# Untransformed
geo <- TT.plot(
    class.sys   = "USDA.TT",
    main        =
        "(Dummy) transformation of the USDA texture triangle"
)   #
# Transformed
TT.classes(
    geo             = geo,
    class.sys       = "USDA.TT",
    tri.css.ps.lim  = c(0,2,20,2000),
    css.transf      = TRUE,  #  <<-- important
    # Additional "graphical" options
    class.line.col  = "blue",
    class.lab.col   = "blue",
    lwd.axis        = 2,
    class.lty       = 2
```

```
)   #
```



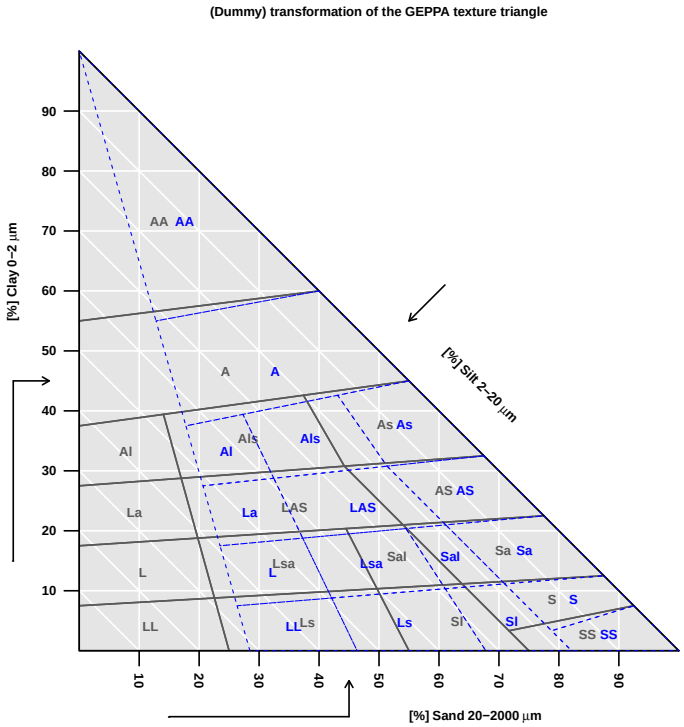
Now another test with a right-angled triangle (the French GEPPA). The background silt sand limit is 50μmeters and the 2nd triangle is 'said to have' 20μmeters limit. The 1st is not transformed (because it is not needed), while the 2nd is transformed:

```
geo <- TT.plot(
    class.sys   = "FR.GEPPA.TT",
    blr.tx      = c("SAND","CLAY","SILT"),
    main        =
        "(Dummy) transformation of the GEPPA texture triangle"
)   #
TT.classes(
    geo             = geo,
    class.sys       = "FR.GEPPA.TT",
    tri.css.ps.lim  = c(0,2,20,2000),
    css.transf      = TRUE,  #  <<-- important
    # Additional "graphical" options
    class.line.col  = "blue",
    class.lab.col   = "blue",
    lwd.axis        = 2,
    class.lty       = 2
)   #
```



Now another test with the same triangle. The background silt sand limit is fixed at 20μmeters, while the GEPPA triangle has a 50μm silt sand limit. The fist triangle is not transformed, while the second is transformed:

```
# Not transformed
geo <- TT.plot(
    class.sys       = "FR.GEPPA.TT",
    blr.tx          = c("SAND","CLAY","SILT"),
    base.css.ps.lim  = c(0,2,20,2000),
    main        =
        "(Dummy) transformation of the GEPPA texture triangle"
)   #
# Transformed
TT.classes(
    geo             = geo,
    class.sys       = "FR.GEPPA.TT",
    css.transf      = TRUE,  #  <<-- important
    # Additional "graphical" options
    class.line.col  = "blue",
    class.lab.col   = "blue",
    lwd.axis        = 2,
    class.lty       = 2
)   #
```



9.5 Classifying and transforming 'on the fly' soil texture data

It is possible to transform soil texture data when classifying them according to a given soil texture classification[11].

```
TT.points.in.classes(
    tri.data        = my.text[1:5,],
    class.sys       = "USDA.TT",
    dat.css.ps.lim  = c(0,2,20,2000),
    css.transf      = TRUE   #  <<-- important
)   #
```

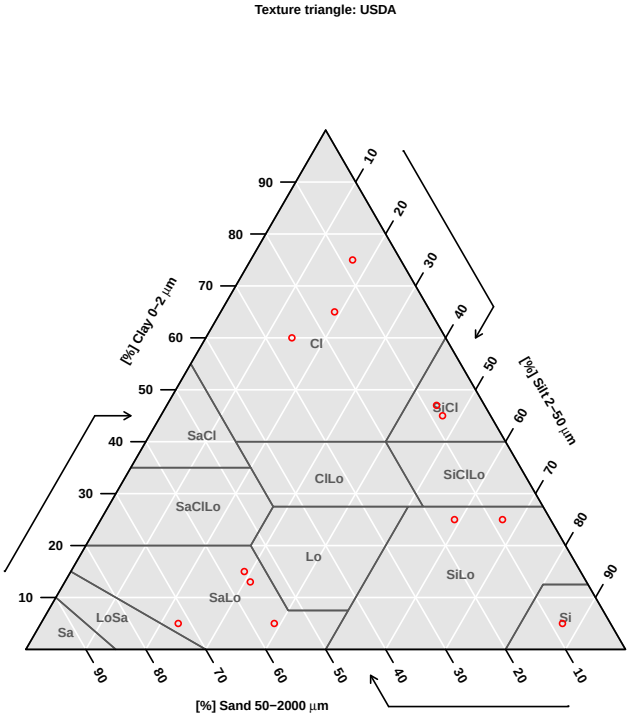
```
     Cl SiCl SaCl ClLo SiClLo SaClLo Lo SiLo SaLo Si LoSa Sa
[1,]  0    0    0    0      0      0  0    0    1  0    0  0
[2,]  1    0    0    0      0      0  0    0    0  0    0  0
[3,]  0    0    0    0      0      0  0    0    1  0    0  0
[4,]  0    0    0    0      0      0  0    0    1  0    0  0
[5,]  0    0    0    0      0      0  0    1    0  0    0  0
```

Visualize the result (all data):

```
TT.plot(
    class.sys       = "USDA.TT",
    tri.data        = my.text,
```

[11] And also the other way round: classifying soil texture data according to a transformed soil texture triangle / classification, but this is NOT recommended, for results consistency

```
    dat.css.ps.lim  = c(0,2,20,2000),
    css.transf      = TRUE,  #  <<-- important
    col             = "red"
)   #
```



But don't do (the texture triangle is transformed, not the data, and that is not good at all):

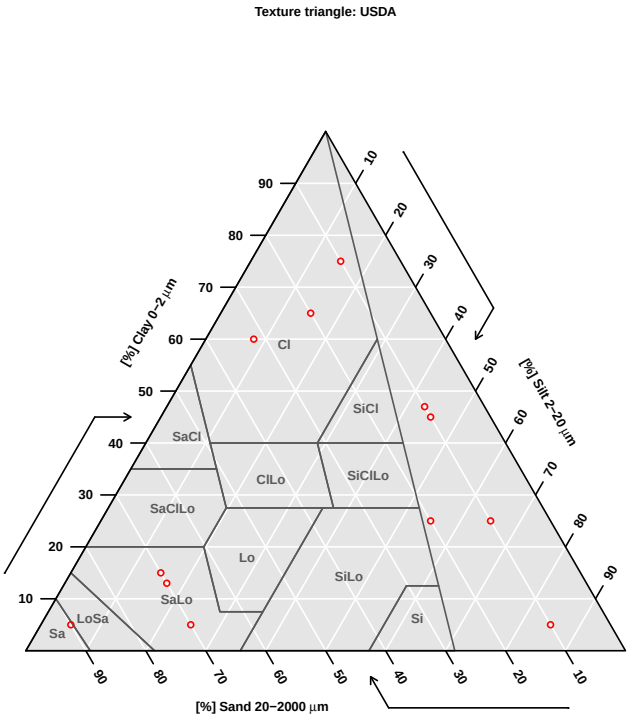
```
TT.points.in.classes(
    tri.data        = my.text[1:5,],
    class.sys       = "USDA.TT",
    dat.css.ps.lim  = c(0,2,20,2000),
    base.css.ps.lim = c(0,2,20,2000),
    css.transf      = TRUE
)   #
```

```
     Cl SiCl SaCl ClLo SiClLo SaClLo Lo SiLo SaLo Si LoSa Sa
[1,]  0    0    0    0      0      0  0    0    0  0    0  1
[2,]  1    0    0    0      0      0  0    0    0  0    0  0
[3,]  0    0    0    0      0      0  0    0    1  0    0  0
[4,]  0    0    0    0      0      0  0    0    1  0    0  0
[5,]  0    0    0    0      0      0  0    0    0  0    0  0
```

Visualize the difference (all data) – and you will understant why doing this causes some problems:

```
TT.plot(
    class.sys       = "USDA.TT",
```

```
    tri.data        = my.text,
    dat.css.ps.lim  = c(0,2,20,2000),
    base.css.ps.lim = c(0,2,20,2000),
    css.transf      = TRUE,
    col             = "red"
)   #
```



Some points lie outside the transformed soil texture classification, so they apparently don't belong to any class... Reason why it is better to transform soil texture data rather than soil texture classification / triangle.

9.6 Using your own custom transformation function when plotting or classifying soil texture data

The `TT.text.transf()` function has been introduced a little earlier in this document. The function is transforming a data.frame containing soil texture data into another data.frame, containing soil texture data estimated in another particle size system.

This function is also the underlying function used by `TT.plot()` and `TT.points.in.classes()` when transforming texture data 'on the fly'.

You may well create an alternative function to `TT.text.transf()`, that fits better your requirements, and willing to use it also when plotting or classifying data with `TT.plot()` or `TT.points.in.classes()`. It is possible by changing the option `text.transf.fun` of `TT.plot()` or `TT.points.in.classes()`. This option is a character string naming the function to use

when transforming data 'on the fly'. The function must accept the same arguments / options as `TT.text.transf()` (to see them, type `formals(TT.text.transf-)`), even if some of them are in fact not used by your own function (it is just there for compatibility). On the other hand, `TT.text.transf()` has 2 unused 'options slots', `trsf.add.opt1` and `trsf.add.opt2` that you may use in your own function if needed (as these 'options slots' are also ready for use in `TT.plot()` and `TT.points.in.classes()`).

Here is a simple example:

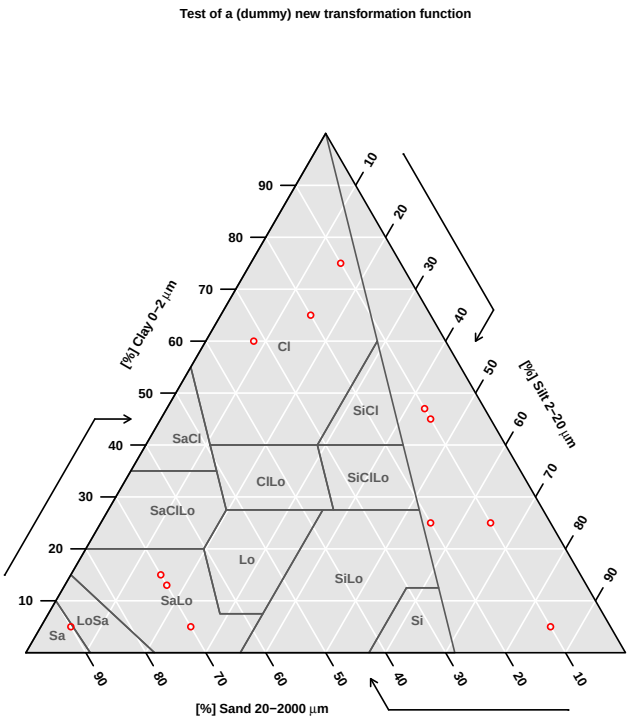
```
# Create a new function, in fact the copy of TT.text.transf()
TT.text.transf2 <- TT.text.transf
# Imagine some changes in TT.text.transf2...

# Use your new function (will give identical results)
TT.points.in.classes(
    tri.data        = my.text[1:5,],
    class.sys       = "USDA.TT",
    dat.css.ps.lim  = c(0,2,20,2000),
    base.css.ps.lim = c(0,2,20,2000),
    css.transf      = TRUE,
    text.transf.fun = "TT.text.transf2"  #  <<-- important
)   #
```

```
     Cl SiCl SaCl ClLo SiClLo SaClLo Lo SiLo SaLo Si LoSa Sa
[1,]  0    0    0    0      0      0  0    0    0  0    0  1
[2,]  1    0    0    0      0      0  0    0    0  0    0  0
[3,]  0    0    0    0      0      0  0    0    1  0    0  0
[4,]  0    0    0    0      0      0  0    0    1  0    0  0
[5,]  0    0    0    0      0      0  0    0    0  0    0  0
```

Of course it works with `TT.plot()`:

```
TT.plot(
    class.sys       = "USDA.TT",
    tri.data        = my.text,
    dat.css.ps.lim  = c(0,2,20,2000),
    base.css.ps.lim = c(0,2,20,2000),
    css.transf      = TRUE,
    col             = "red",
    text.transf.fun = "TT.text.transf2", #  <<-- important
    main            =
        "Test of a (dummy) new transformation function"
)   #
```



10 Customize soil texture triangle's geometry

Behind the ability of 'The soil texture wizard' package to plot seemlessly any soil texture classification system lies a system that disconnect **soil texture class boundaries** (expressed as 3D volumes, whose submit coordinates are expressed in clay / silt / sand proportion) and **soil texture triangle geometry**, where soil texture values are projected in a plane (as points, or 2D polygons).

'The soil texture wizard' package allows to change:

- Clay, Silt and Sand locations in the bottom, left and right axis;
- Triangle vertices angles;
- Direction of the axis (Clockwise, Anticlockwise or neutral).

All angles combinations are allowed, provided they sum to 180 degrees. Nevertheless only 60 / 60 / 60 degrees or one 90 degrees together with two 45 degrees are recommended for obtaining smart plots (and correct anti-allysing).

Only 4 combinations of axis directions are allowed (and, as far as I know, geometrically possible):

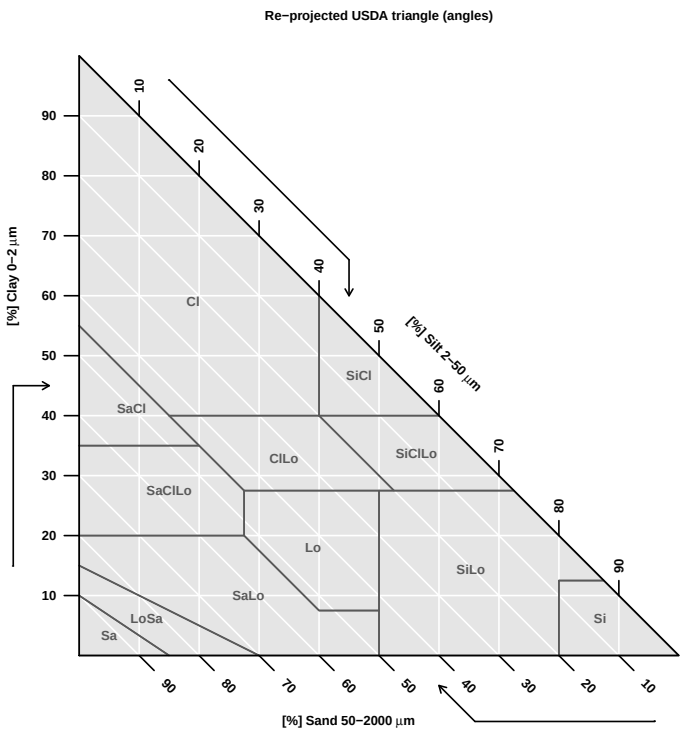
- Full anticlockwise directions;
- Full clockwise direction;

- Bottom anticlockwise, left clockwise, and right neutral (inside). This is the best presentation for triangles with a 90 degrees angle on the left, but other angles are allowed.
- Bottom clockwise, left neutral (inside), and right anticlockwise. This is the best presentation for triangles with a 90 degrees angle on the right, but other angles are allowed.

10.1 Customise angles

Below is a code example to project the USDA soil texture triangle using 90 and 45 / 45 angles:

```
TT.plot(
    class.sys   = "USDA.TT",
    tlr.an      = c(45,90,45),
    main        = "Re-projected USDA triangle (angles)"
)   #
```



the option `tlr.an` accept a vector of numerical, for the Top, Left and Right ANgles respectively.

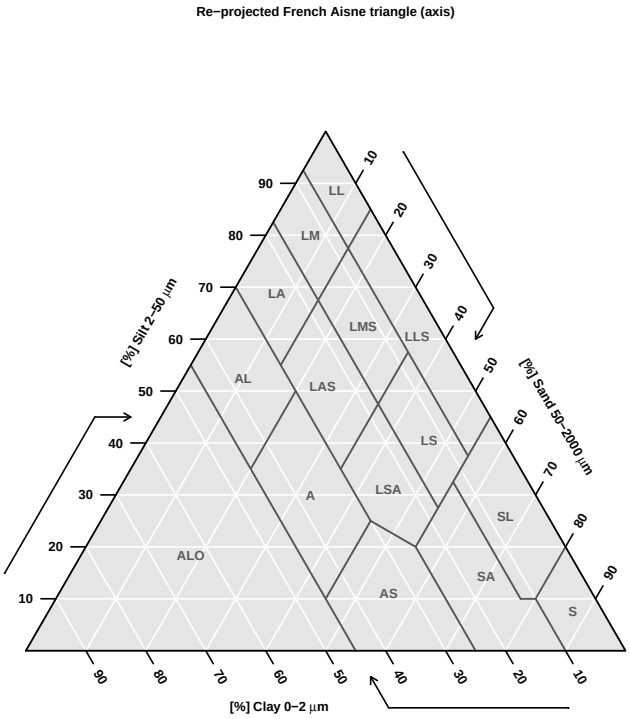
the option `main` accept charcter string, for the plot title. If set to `NA`, no title is plotted, and the graph is enlarged a bit.

Notice that this triangle has NOT the same geometry as the german system, although it has the same angles (the axis directions are different, and clay silt and sand are 'carried' by different axis).

10.2 Customize texture class axis

Below is a code example to project the French 'Aisne' soil texture triangle, with clay at the bottom, silt on the left and sand on the right:

```
TT.plot(
    class.sys   = "FR.AISNE.TT",
    blr.tx      = c("CLAY","SILT","SAND"),
    main        = "Re-projected French Aisne triangle (axis)"
)   #
```



The option `blr.tx` accept a vector of character strings. The **values** of the vector indicates which texture class should be drawn for the Bottom, Left and Right TEXtures; respectively.

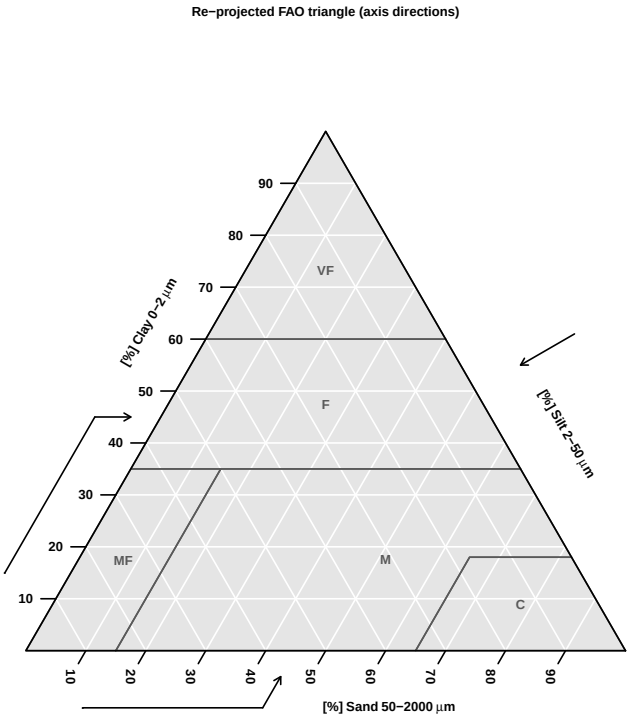
The option `blr.tx` should not be confused with the option `css.names`, presented below ('Internationalization'), that defines the (columns) names taken by clay, silt and sand (respectively) in the data.frame passed to `tri.data`.

The option `blr.tx` should not be confused with the option `css.lab`, presented below ('Internationalization'), that defines the names (or expressions) displayed for clay, silt and sand axis labels / titles.

10.3 Customise axis direction

Below is a code example to project the FAO soil texture triangle, with the bottom axis anticlockwise, the left axis clockwise and the right axis 'neutral' (inside).

```
TT.plot(
    class.sys   = "FAO50.TT",
    blr.clock   = c(FALSE,TRUE,NA),
    main        = "Re-projected FAO triangle (axis directions)"
)   #
```

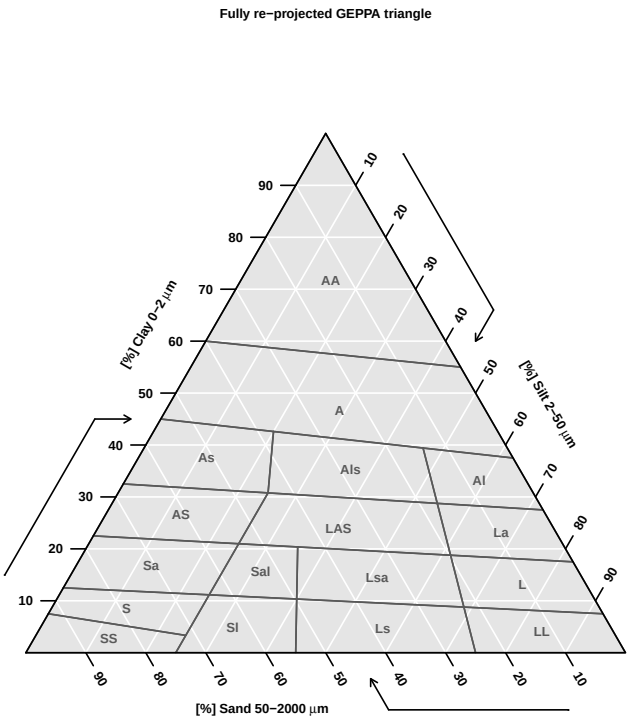


the option `blr.clock` accept a vector of boolean (TRUE / FALSE) as argument for bottom direction (here clock = FALSE), left (clock = FALSE) and right (clock = NA). The `NA` value is used for the neutral case.

10.4 Customise everything: plot The French GEPPA classification in the French Aisne triangle

Below is a French GEPPA soil texture triangle plotted as a USDA or FAO or French Aisne soil texture triangle, that is with custom angles, axis directions and clay silt sand positions on the axis:

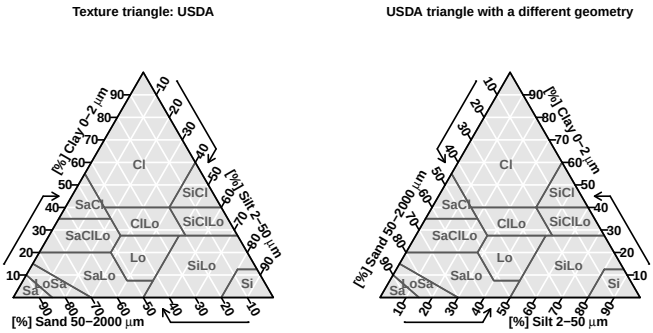
```
TT.plot(
    class.sys   = "FR.GEPPA.TT",
    tlr.an      = c(60,60,60),
    blr.tx      = c("SAND","CLAY","SILT"),
    blr.clock   = c(TRUE,TRUE,TRUE),
    main        = "Fully re-projected GEPPA triangle"
)   #
```



10.5 Miscellaneous: Different triangle geometry, but same projected classes

I once found an illustration on a website showing a USDA and a French Aisne soil texture triangle displayed together, but with a different triangle geometry than the usual one. The surprise was that although the triangle geometry was different, the texture classes shapes (i.e. inside the triangle) looked exactly the same as the 'standard' display. Here is a practical example (not an explanation of this strange phenomenon!):

```
# Set a 2 by 2 plot matrix:
old.par <- par(no.readonly=T)
par("mfcol" = c(1,2),"mfrow"=c(1,2))
# Plot the triangles with different geometries:
TT.plot( class.sys = "USDA.TT" )
TT.plot(
    class.sys   = "USDA.TT",
    blr.tx      = c("SILT","SAND","CLAY"),
    blr.clock   = c(FALSE,FALSE,FALSE),
    main        = "USDA triangle with a different geometry"
)   #
# Back to old parameters:
par(old.par)
```



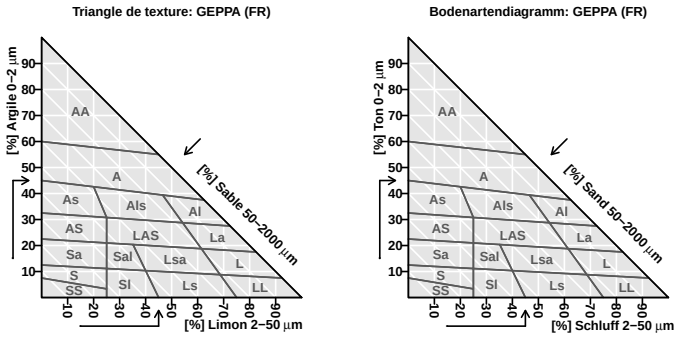
11 Internationalization: title, labels and data names in different languages

11.1 Choose the language of texture triangle axis and title

The `TT.plot()` function comes with a `lang` option ("lang" for language) that allows to plot texture triangles with a title and axis labels in other languages than English (the default).

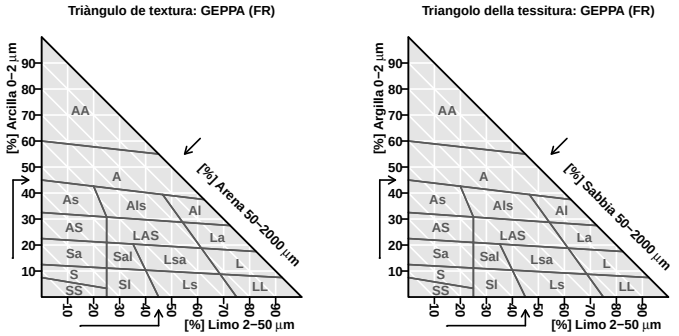
Here is a first example with French (`lang = "fr"`) and German (`lang = "de"`):

```
# Set a 2 by 2 plot matrix:
old.par <- par(no.readonly=T)
par("mfcol" = c(1,2),"mfrow"=c(1,2))
# Plot the triangles with different languages:
TT.plot(
    class.sys   = "FR.GEPPA.TT",
    lang        = "fr"
)   #
TT.plot(
    class.sys   = "FR.GEPPA.TT",
    lang        = "de"
)   #
# Back to old parameters:
par(old.par)
```



A second example with Spanish (`lang = "es"`), and Italian (`lang = "it"`):

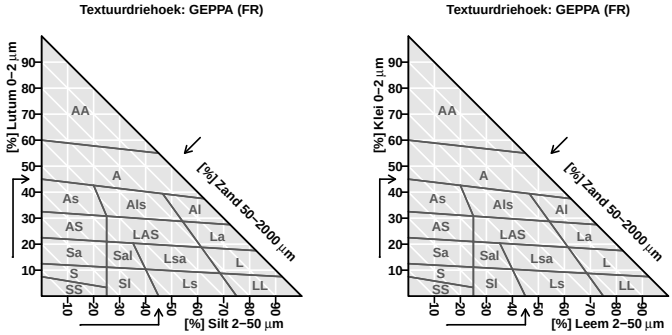
```
# Set a 2 by 2 plot matrix:
old.par <- par(no.readonly=T)
par("mfcol" = c(1,2),"mfrow"=c(1,2))
# Plot the triangles with different languages:
TT.plot(
    class.sys   = "FR.GEPPA.TT",
    lang        = "es"
)   #
TT.plot(
    class.sys   = "FR.GEPPA.TT",
    lang        = "it"
)   #
# Back to old parameters:
par(old.par)
```



And a third example with Dutch (`lang = "nl"`) and Flemish (`lang = "nl"`) or, to be more exact, with the terms used on the Dutch texture triangle and on the Flemish version of the Belgian texture triangle.

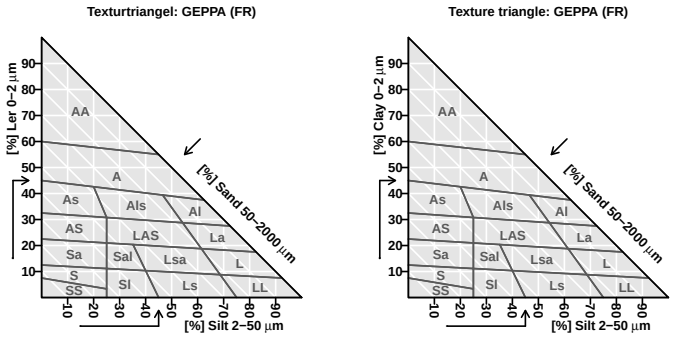
```
# Set a 2 by 2 plot matrix:
old.par <- par(no.readonly=T)
par("mfcol" = c(1,2),"mfrow"=c(1,2))
# Plot the triangles with different languages:
TT.plot(
    class.sys   = "FR.GEPPA.TT",
```

```
    lang        = "nl"
)   #
TT.plot(
    class.sys   = "FR.GEPPA.TT",
    lang        = "fl"
)   #
# Back to old parameters:
par(old.par)
```



And finally in Swedish (`lang = "se"`) and in English, the default language (`lang = "en"`):

```
# Set a 2 by 2 plot matrix (for size):
old.par <- par(no.readonly=T)
par("mfcol" = c(1,2),"mfrow"=c(1,2))
# Plot the triangles with different languages:
TT.plot(
    class.sys   = "FR.GEPPA.TT",
    lang        = "se"
)   #
# Plot the triangles with different languages:
TT.plot(
    class.sys   = "FR.GEPPA.TT",
    lang        = "en"
)   #
# Back to old parameters:
par(old.par)
```

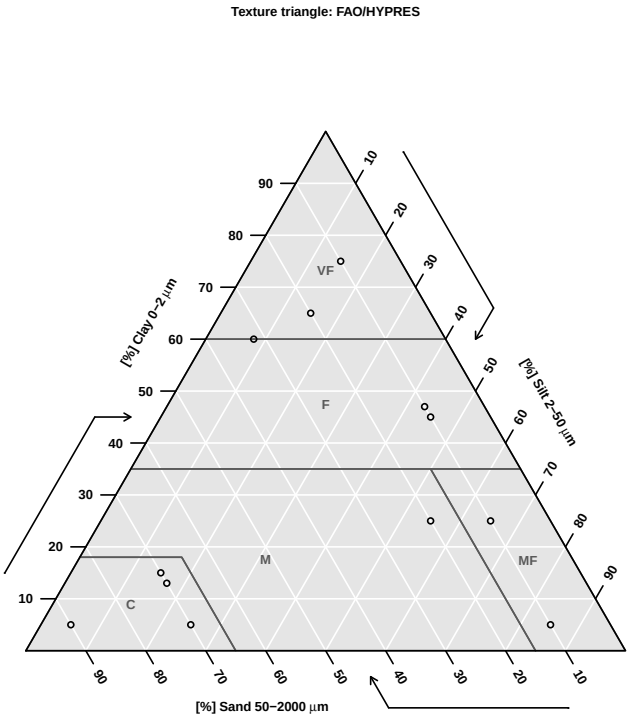


Please report any mistakes in these translations. Please don't hesitate to send me new translations in other languages. This option is easily extensible.

11.2 Use custom (columns) names for soil texture data

The dummy data table with french names can be displayed, but the option `css.names` (for Clay, Silt, Sand NAMES) must be specified, as a vector of character string, corresponding to the columns names of `tri.data`. **Notice that the order of css.names is NOT NECESSARILY the order of `tri.data` columns names**, so `css.names == colnames(my.text.fr)` would NOT always be true (although it is true here).

```
TT.plot(
    tri.data    = my.text.fr,
    class.sys   = "FAO50.TT",
    css.names   = c("ARGILE","LIMON","SABLE")
)   #
```

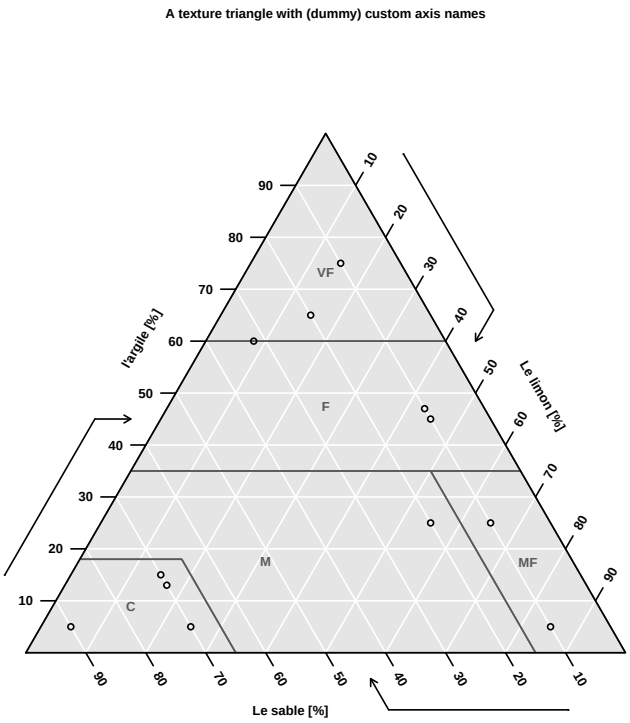


You can see that the column names ARGILE, LIMON and SABLE do not appear on the texture triangle plot. There is always a clear separation between the characteristics of the texture classification, the characteristics of the soil texture data provided, and the characteristics of the soil texture triangle plot/presentation. This is necessary for making different systems compatible.

11.3 Use custom labels for the axis

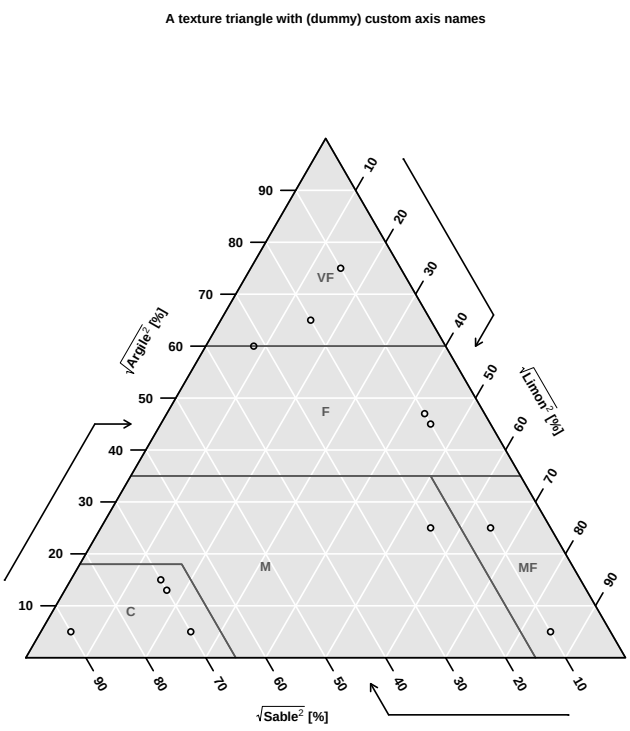
It is also possible to use custom labels for the triangle axis. The option `css.lab` accept a vector (3) of character string, or a vector of expressions, for Clay, Silt and Sand LABels respectively (in that order, no matter the texture triangle system used. The function automatically places the good label on the good axis).

```
TT.plot(
    tri.data    = my.text.fr,
    class.sys   = "FAO50.TT",
    css.names   = c("ARGILE","LIMON","SABLE"),
    css.lab     = c("l'argile [%]","Le limon [%]","Le sable [%]"),
    main        =
        "A texture triangle with (dummy) custom axis names"
)   #
```



The use of expressions allows a finer customization of the axis labels, as for any R plot:

```
TT.plot(
    tri.data    = my.text.fr,
    class.sys   = "FAO50.TT",
    css.names   = c("ARGILE","LIMON","SABLE"),
    css.lab     = expression(
        bold(sqrt('Argile'^2)~'[%]'),
        bold(sqrt('Limon'^2)~'[%]'),
        bold(sqrt('Sable'^2)~'[%]')
    ),  #
    main        =
        "A texture triangle with (dummy) custom axis names"
)   #
```



12 Checking the geometry and classes boundaries of soil texture classifications

'The Soil Texture Wizard' comes with several functions that helps to retrieve information about the geometry and classes boundaries of soil texture classifications, and thus to check by yourself that they are 'correct'.

12.1 Checking the geometry of soil texture classifications

Below is a simple example of the way to retrieve the geometry of a given texture triangle (as it is implemented in the 'The Soil Texture Wizard'):

```
# Fisrt, retrieve all the data about
#   the USDA texture triangle
tmp <- TT.get("USDA.TT")
# It is not displayed here because it is to big
#   The list names are:
names(tmp)
```

```
 [1] "main"            "tt.points"       "tt.polygons"
 [4] "blr.clock"       "tlr.an"          "blr.tx"
 [7] "base.css.ps.lim" "tri.css.ps.lim"  "unit.ps"
[10] "unit.tx"         "text.sum"
```

```
# If we drop "tt.points" and "tt.polygons", that will be
#   presented later, the list size is more reasonable
tmp[ !names(tmp) %in% c("tt.points","tt.polygons") ]
```

```
$main
[1] "USDA"

$blr.clock
[1] TRUE TRUE TRUE

$tlr.an
[1] 60 60 60

$blr.tx
[1] "SAND" "CLAY" "SILT"

$base.css.ps.lim
[1]    0    2   50 2000

$tri.css.ps.lim
[1]    0    2   50 2000

$unit.ps
bold(mu) * bold("m")

$unit.tx
bold("%")

$text.sum
[1] 100
```

Most of the list's names correspond to `TT.plot()`'s options that have been presented earlier in the document.

12.2 Checking classes names and boundaries of soil texture classifications

The function `TT.classes.tbl()` returns a table with information about each soil texture class of a given soil texture classification / triangle:

```
# Retrieve and save the table:
tmp2 <- TT.classes.tbl( class.sys = "FAO50.TT" )
# Display the first part:
tmp2[,1:2]
```

```
     abbr name
[1,] "VF" "Very fine"
[2,] "F"  "Fine"
[3,] "M"  "Medium"
[4,] "MF" "Medium fine"
[5,] "C"  "Coarse"
```

```
# Then display the last column (and the 1st again):
tmp2[,c(1,3)]
```

```
     abbr points
[1,] "VF" "2, 1, 3"
[2,] "F"  "4, 2, 3, 6"
[3,] "M"  "7, 4, 5, 11, 10, 8"
[4,] "MF" "11, 5, 6, 12"
[5,] "C"  "9, 7, 8, 10"
```

The 1st column is the classes abbreviation, the 2nd column is the classes full name (in the original language), and the 3rd class contains a list of vertices numbers, separated by a comma. These vertices are those who compose the class 'polygon'.

But the vertices numbers are of course not self explicit. For this reason, another function, `TT.vertices.tbl()`, extracts the vertices coordinates (as clay silt and sand content) from the triangle definition:

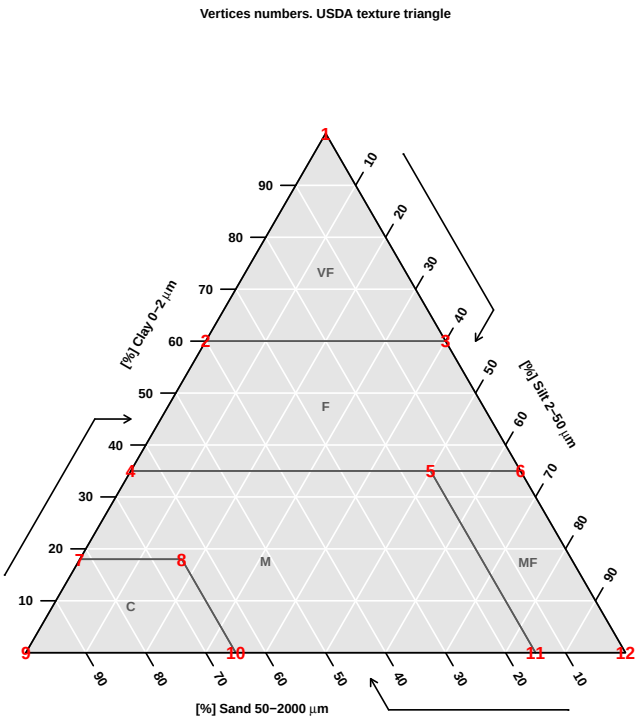
```
TT.vertices.tbl( class.sys = "FAO50.TT" )
```

```
   points CLAY SILT SAND
1       1 1.00 0.00 0.00
2       2 0.60 0.00 0.40
3       3 0.60 0.40 0.00
4       4 0.35 0.00 0.65
5       5 0.35 0.50 0.15
6       6 0.35 0.65 0.00
7       7 0.18 0.00 0.82
8       8 0.18 0.17 0.65
9       9 0.00 0.00 1.00
10     10 0.00 0.35 0.65
11     11 0.00 0.85 0.15
12     12 0.00 1.00 0.00
```

The 'points' number outputted by `TT.classes.tbl()` logically correspond to those outputted by `TT.vertices.tbl()`.

Finally, as it is difficult to visualize the position of each vertex with just a table, the function `TT.vertices.plot()` completes the two presented above, by plotting the vertices position and numbers on top of a texture triangle (preferable the same as the one you are actually checking!):

```
geo <- TT.plot(
    class.sys   = "FAO50.TT",
    main        = "Vertices numbers. USDA texture triangle"
)   #
TT.vertices.plot(
    geo         = geo,
    class.sys   = "FAO50.TT",
    col         = "red",
    cex         = 2,
    font        = 2
)   #
```



Many `TT.vertices.plot()` options are shared with `TT.text()`, which an the underlying function of the first.

13 Adding your own, custom, texture triangle(s)

As said in the introduction, the idea behind 'The Soil Texture Wizard' is to be able to display 'any soil texture triangle / classification' in any 'triangle geometry'.

As there are much more texture classification systems than those implemented in 'The Soil Texture Wizard', the function `TT.add()` is there to 'add' any texture triangle to the existing list (your home made texture triangles).

In the example below, we will (1) extract and save the definition of the FAO50 texture triangle, (2) change the silt sand limit of the saved triangle definition from 50μm to 63μm, and (3) load the 'new' texture triangle into the existing list of triangle. There, it will be saved and usable as any texture triangle (at least until R is shut down).

```
# Step 1
FAO63 <- TT.get("FAO50.TT")
#
# Visualize the data that will be modified
FAO63[[ "base.css.ps.lim" ]]
```

```
[1]    0    2   50 2000
```

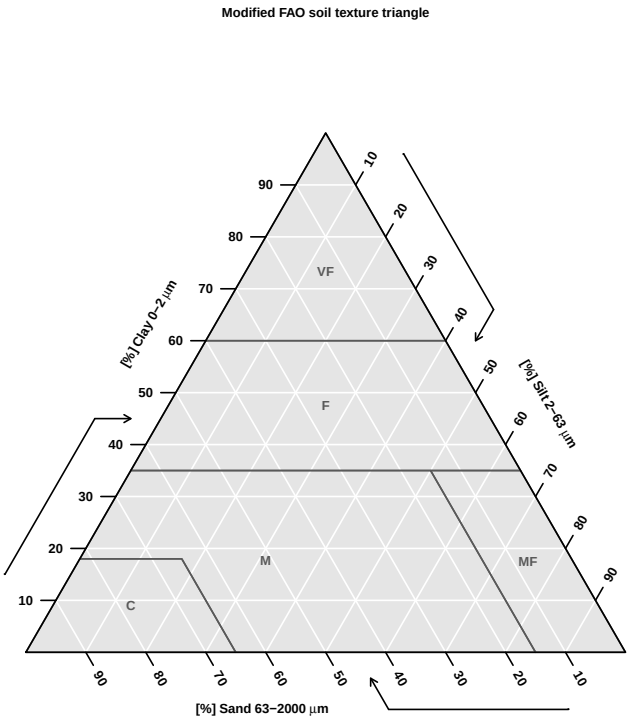
```
FAO63[[ "tri.css.ps.lim" ]]
```

```
[1]    0    2   50 2000
```

```
#
# Step 2
FAO63[[ "base.css.ps.lim" ]][3] <- 63
FAO63[[ "tri.css.ps.lim" ]][3]  <- 63
#
# Step 3: Load the new texture triangle
TT.add( "FAO63.TT" = FAO63 )
```

This (not so) new texture triangle is now usable as any other in the initial list:

```
TT.plot(
    class.sys   = "FAO63.TT",
    main        = "Modified FAO soil texture triangle"
)   #
```



If you consider creating a completely new triangle, you will need to copy / imitate the structure of existing texture triangle definitions.

This definition is a list of parameters:

```
# Get the definition of the FAO50 texture triangle
FAO50 <- TT.get( "FAO50.TT" )
#
# Check its class (list)
class( FAO50 )
```

```
[1] "list"
```

```
#
# Check its parameters names
names( FAO50 )
```

```
 [1] "main"            "tt.points"       "tt.polygons"
 [4] "blr.clock"       "tlr.an"          "blr.tx"
 [7] "base.css.ps.lim" "tri.css.ps.lim"  "unit.ps"
[10] "unit.tx"         "text.sum"
```

```
#
# Check its parameters class
for( i in 1:length(FAO50) )
{   #
    print(
        paste(
            names( FAO50 )[i],
            class( FAO50[[i]] ),
            sep = ": "
        )   #
    )   #
}   #
```

```
[1] "main: character"
[1] "tt.points: data.frame"
[1] "tt.polygons: list"
[1] "blr.clock: logical"
[1] "tlr.an: numeric"
[1] "blr.tx: character"
[1] "base.css.ps.lim: numeric"
[1] "tri.css.ps.lim: numeric"
[1] "unit.ps: call"
[1] "unit.tx: call"
[1] "text.sum: numeric"
```

All elements are (probably) self explicit, or have been presented earlier in the document (as options of `TT.plot()` for instance), except for `tt.points` and `tt.polygons`.

`tt.points` is a 3 column data.frame, with columns names CALY SILT and SAND, that gives the clay, silt and sand 'coordinates' of all the vertices in the triangles. Rows numbers corresponds to the 'official' vertex number. Vertices are unique, they can not be repeated, even if they are 'used' in several classes definition / polygon.

`tt.polygons` is a list. Each element of the list corresponds to one texture class. The name of each element of the list is the texture class abbreviation. Then, each element of the list (i.e. each class) itself contains a list, with two elements, named `"name"` and `"points"`. `"name"` is a single character string, corresponding to the texture class 'full name' in the triangle's native language (but special character should be avoided). `"points"` is an ordered vector of integers, corresponding to the numbers of the vertices numbers that compose

each class's polygon. The numbers in fact corresponds to the row names of `tt.points`. They should be ordered, in the sense that each successive pair of vertices will be joined by a polygon edge (+ one between the 1st and the last points).

If you want to have a more precise idea of the code's shape for a new texture triangle, you can have a look inside 'The Soil Texture Wizard' code (FUNCTION_TEXTURE_WIZARD.R): search the name of a texture triangle (e.g. "FAO50.TT") in the code and look the structure of the list corresponding to it.

If you implement a texture triangle that you think can be used by many soil scientists, please consider sending its definition's code to me, I will implement it into 'The Soil Texture Wizard' (copy and paste)[12]. If doing this, it is als good to send an image of the 'reference' texture triangle that was used to obtain the triangle's definition (so it can be checked!).

14 Further readings

Readers wishing to go further in the study of soil texture triangles and related topics may like the following articles: Richer de Forges et al. 2001 a (in French, [25]) and b (Poster in English, [24]; Nemes et al. 1999 (about texture transformation, [22]); Gerakis and Baer (program for USDA texture classification, [13]); Liebens 2001 (Excel MACRO for USDA texture classification, [19]); Minasny and McBratney 2001 (about texture transformation, [20]); Teh and Rashid 2003 (program for texture classification in a lot of systems, [28]) and Christopher (Teh) and Mokhtaruddin 1996[7].

References

[1] Agriculture and Agri-Food Canada. Glossary – canadian soil information system. Internet, april 2010. http://sis.agr.gc.ca/cansis/intro.html.

[2] Anomymous. Soil – korngrößendreieck – particle size diagram. Web page, 2009. Last access 2009/08/24. http://nibis.ni.schule.de/~trianet/soil/boden5.htm.

[3] Anonymous. The canadian soil information system (cansis) – national soil database (nsdb) – land potential database (lpdb) – fao soil texture. Website, 08 2009. http://sis.agr.gc.ca/cansis/nsdb/lpdb/faotext.html.

[4] Anonymous. Fingerprobe (boden). Website (Wikipedia.de), 2009. Last access 2009/08/24.http://de.wikipedia.org/wiki/Fingerprobe_%28Boden%29.

[5] D. Baize and B. Jabiol. *Guide pour la description des sols*. col. Techniques et pratiques. INRA, 1995. 375 p.

[12] Under the same open-source licence terms as all the function set, but with reference to you work and your work's 'model'

[6] H. Bormann. Analysis of the suitability of the german soil texture classification for the regional scale application of physical based hydrological model. *Advances in Geosciences*, 11:7–13, 2007. www.adv-geosci.net/11/7/2007/.

[7] T.B.S. Christopher and A.M. Mokhtaruddin. A computer program to determine the soil textural class in 1-2-3 for windows and excel. *Communications in Soil Science and Plant Analysis*, 27(9 and 10):2315–2319, 1996. http://www.informaworld.com/smpp/content~db=all~content=a905339412~tab=linking.

[8] Defourny, Delvaux, Ducarme, and Radoux. Travaux pratiques de cartographie du sol. Website, last checked 2009/10/14 2009. http://www.icampus.ucl.ac.be/courses/MILA2230/document/texture.html.

[9] Defra – Rural Development Service – Technical Advice Unit. Technical advice note 52 & 53 – soil texture. Technical report, Rural Development Service, august 2006. 6 p. http://www.defra.gov.uk/environment/land/soil/information/publications.htm.

[10] European Soil Bureau working group "HYdraulic PRoperties of European Soils" (HYPRES). Texture classes. HYPRES Website, 08 2009. http://www.macaulay.ac.uk/hypres/hypressoil.html.

[11] P. Filzmoser and K. Varmuza. *chemometrics: Multivariate Statistical Analysis in Chemometrics*, 2008. R package version 0.4. http://cran.r-project.org/web/packages/chemometrics/index.html.

[12] GEOVLEX, MLU Halle-Wittenberg. Glossardatenbank – bodenart. Website, 2009. Last access 2009/08/24. http://mars.geographie.uni-halle.de/mlucampus/geoglossar/terme_datenblatt.php?terme=Bodenart.

[13] A. Gerakis and B. Baer. A computer program for soil textural classification. *Soil Science Society of America Journal*, 63:807–808, 1999. http://nowlin.css.msu.edu/software/triangle_form.html.

[14] Georgina Holbeche. Soilquality.org.au – physics – measuring soil texture in the laboratory. Online pdf document., Department of Agriculture and Food, The State of Western Australia – The University of Western Australia. Healthy Soils for Sustainable Farms programme., 2008. 2p. http://www.soilquality.org.au/documents/28/original/Phys_-_Measuring_Soil_Texture_in_the_Lab_web.pdf.

[15] M. Jamagne. Bases et techniques d'une cartographie des sols. *Annales Agronomiques*, 18 (hors série):142, 1967.

[16] R. M. Lark and T. F. A. Bishop. Cokriging particle size fractions of the soil. *European Journal of Soil Science*, 58:763–774, June 2007. http://www3.interscience.wiley.com/journal/118000728/abstract.

[17] F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In Wolfgang Härdle and Bernd Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, Germany, 2002. ISBN 3-7908-1517-9. http://www.stat.uni-muenchen.de/~leisch/Sweave.

[18] Linnéa Lidberg. Texturbestämning genom fält-, pipett- och hydrometermetoder. Examensarbete (msc soil science and environmental studies), SLU – Institutionen för skoglig marklära (Department of Forest Soils), Uppsala, Sweden, 2009. vol 20. 31 pp. http://ex-epsilon.slu.se/archive/00003079/.

[19] J. Liebens. Spreadsheet macro to determine usda soil textural subclasses. *Communications in Soil Science and Plant Analysis*, 32(1 and 2):255–265, 2001. http://www.informaworld.com/openurl?genre=article&issn=0010%2d3624&volume=32&issue=1&spage=255.

[20] B. Minasny and A.B. McBratney. The australian soil texture boomerang: a comparison of the australian and usda/fao soil particle-size classification systems. *Australian Journal of Soil Research*, 39:1443–1451, 2001. http://www.publish.csiro.au/nid/84/paper/SR00065.htm.

[21] Julien Moeys. *Variabilité spatiale et déterminismes agroŰpédologiques du devenir dŠun herbicide dans lŠhorizon de surface – Application au cas de l'isoproturon dans un secteur agricole de Beauce chartraine*. Soil science, AgroParisTech, Paris, France, December 2007. 273 pp. + annexes. http://pastel.paristech.org/4448/.

[22] A. Nemes, J.H.M. Wösten, A. Lilly, and J.H. Oude Voshaar. Evaluation of different procedures to interpolate particle-size distributions to achieve compatibility within soil databases. *Geoderma*, 90:187–202, 1999. http://dx.doi.org/10.1016/S0016-7061(99)00014-2.

[23] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0. http://www.R-project.org.

[24] A. Richer de Forges, C. Feller, M. Jamagne, and D. Arrouays. Lost in the triangular diagrams of soil texture. Poster, 2008. http://www.prodinra.inra.fr/prodinra/pinra/data/2008/08/PROD2008387413ba_20080819120904968.pdf.

[25] A. Richer de Forges, C. Feller, M. Jamagne, and D. Arrouays. Perdus dans le triangle des textures. *Etudes et Gestion des Sols*, 15(2):97–111, 2008. (en: Lost in the textures triangle).

[26] Soil Survey Division Staff. *Soil survey manual*, volume Handbook 18, chapter 3. Soil Conservation Service. U.S. Department of Agriculture, 1993. http://soils.usda.gov/technical/manual/print_version/chapter3.html.

[27] Sols de Bretagne. Photothèque – etude d'un sol – triangle des textures geppa. Website, 2009. Last access 2009/08/24. http://www.sols-de-bretagne.fr/index.php/component/option,com_datsogallery/Itemid,79/func,detail/catid,2/id,66/.

[28] C.B.S. Teh and M.A. Rashid. Object-oriented code to lookup soil texture classes for any soil classification scheme. *Communications in Soil Science and Plant Analysis*, 34(1):1–11, 2003. http://www.informaworld.com/smpp/content~db=all~content=a713624019~tab=linking.

[29] Bart Van Bossuyt. De belgische textuurdriehoek. website, last checked 2009/10/14 2009. http://www.begeleidzelfstandigleren.com/aardrijkskunde/losse_animaties/textuurdriehoek_bodems.html.

[30] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. ISBN 0-387-95457-0. http://www.stats.ox.ac.uk/pub/MASS4.