tkz-grapheur [en]

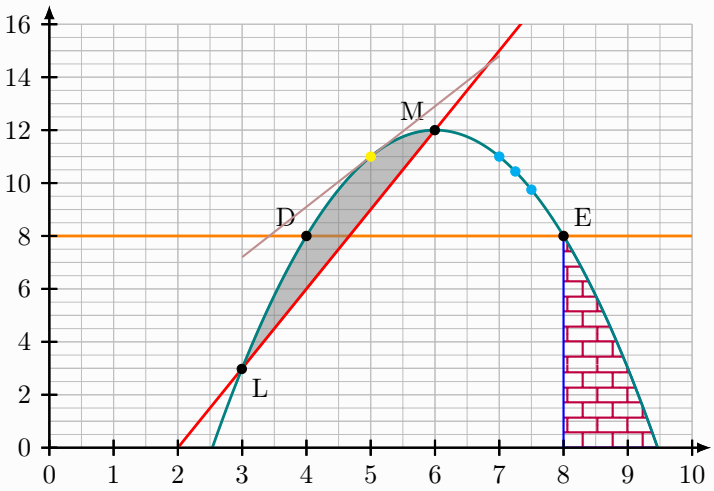
A grapher, based

on TikZ and xint.

Version 0.1.9 - 11/10/2024

Cédric Pierquet
c pierquet - at - outlook . fr
https://forge.apps.education.fr/pierquetcedric/package-latex-tkz-grapheur

---



To my Dad.

Contents

1 Introduction

1.1 Description and general ideas

With this modest package, far from the capabilities offered by the excellent packages `tkz-*`[1] (by Alain Matthes) or `tzplot`[2] (by In-Sung Cho), it is possible to work on function graphs, in Ti*k*Z language, in an *intuitive* and *explicit* way.
Concerning the overall operation:

- particular styles for the objects used have been defined, but they can be modified locally;
- the name of the commands is in *operational* form, so that the construction of the graphic elements has an almost *algorithmic* form.

1.2 Overall operation

To schematize, it *is enough*:

- to declare the parameters of the graphics window (**units in cm !**);
- to display grid/axes/graduations;
- to declare functions or interpolation curves;
- to possibly declare particular points;
- to place a point scatter.

It will then be possible:

- to draw curves;
- to graphically determine images or backgrounds;
- to add elements of derivation (tangents) or integration (domain);
- to draw a linear fit line or the curve of another fit.

1.3 Packages used, and package options

The package uses:

- `tikz`, with the libraries `calc,intersections,patterns,patterns.meta,bbox`;
- `simplekv`, `xintexpr`, `xstring`, `listofitems`;
- `xint-regression`[3] (for regressions, switchable via `[noxintreg]`).

The package also loads `siunitx` with the classic options, but it is possible not to load it using the `[nosiunitx]` option.
The package also loads the Ti*k*Z `babel` library, but it is possible not to load it using the `[notikzbabel]` option.
The different options are obviously cumulative.

```
<MINTED>
```

Also note that certain commands can use packages like `nicefrac`, which will therefore have to be loaded if necessary.
Concerning the *calculations* and *plots* part, the `xint` package takes care of it.

[1] for example tkz-base https://ctan.org/pkg/tkz-base and tkz- fct https://ctan.org/pkg/tkz-fct.
[2] CTAN: https://ctan.org/pkg/tzplot.
[3] CTAN: https://ctan.org/pkg/xint-regression.

1.4 Warnings

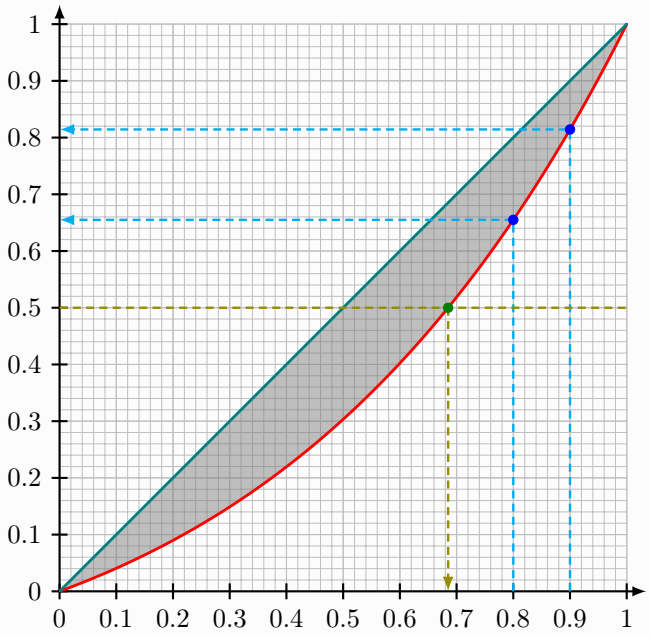
It is possible, due to the (multiple) calculations carried out internally, that the compilation time may be a little *long*.

The precision of the (determination) results seems to be around 10^{-4}, which should normally guarantee *satisfactory* plots and readings. It is still advisable to be cautious about the results obtained and those expected.

1.5 Introductory example

For example, we can start from the following example to *illustrate* the flow of the commands for this package. The commands and syntax will be detailed in the following sections!

```
<MINTED>
```



2 Basic Styles and Environment Creation

2.1 Basic Styles

The styles used for plots are given below.

For *simplicity* purposes, only the color of the elements can be configured, but if the user wishes, he can redefine the proposed styles.

```
<MINTED>
```

We therefore find:

- the origin of the mark (`Origx`/`Origy`);
- the extreme values of the axes (`Xmin`/`Xmax`/`Ymin`/`Ymax`);
- the parameters of the main and secondary grids (`Xgrid`/`Xgrids`/`Ygrid`/`Ygrids`).

Concerning the styles of *objects*, they are given below.

```
<MINTED>
```

The idea is therefore to be able to redefine styles globally or locally, and possibly add elements, using <MINTED>.

2.2 Creating the environment

The proposed environment is based on Ti*k*Z, so that any *classic* command linked to Ti*k*Z can be used alongside the package commands!

```
<MINTED>
```

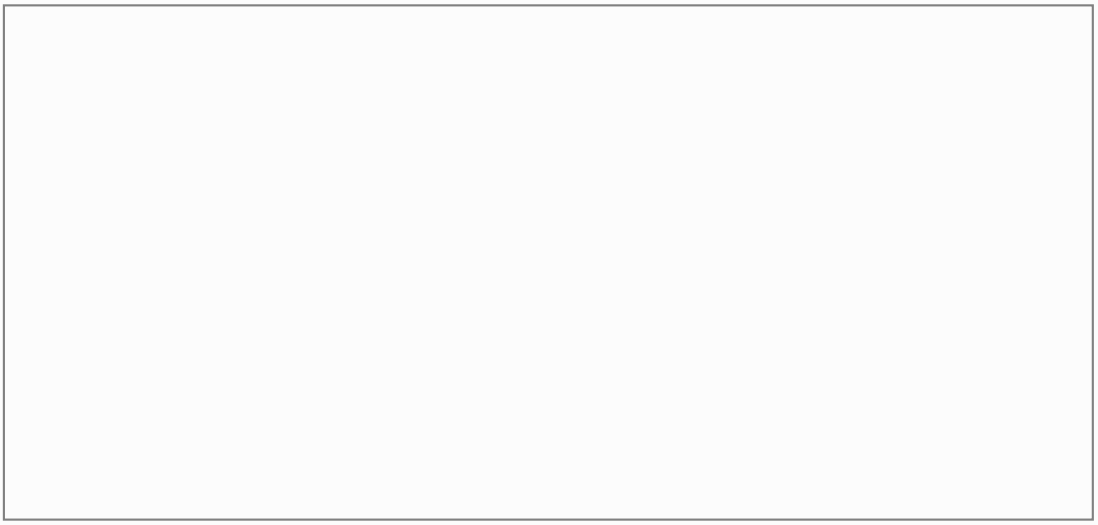
The `[tikz options]` are the *classic* options that can be passed to a Ti*k*Z environment, as well as the axes/grids/window keys presented previously.

The specific (and optional) `<keys>` are:

- `ThickGrad`: size of the axis graduations (`3pt` for 3pt *above* and 3pt *below*);
- `Frame`: boolean (`false` by default) to display a frame which delimits the graphic window (excluding possible graduations).

```
<MINTED>
```

<MINTED>



It will obviously be more meaningful with the added graphic elements!

2.3 Grids and axes

The first command *useful* will allow you to create the grids, axes and graduations.

```
<MINTED>
```

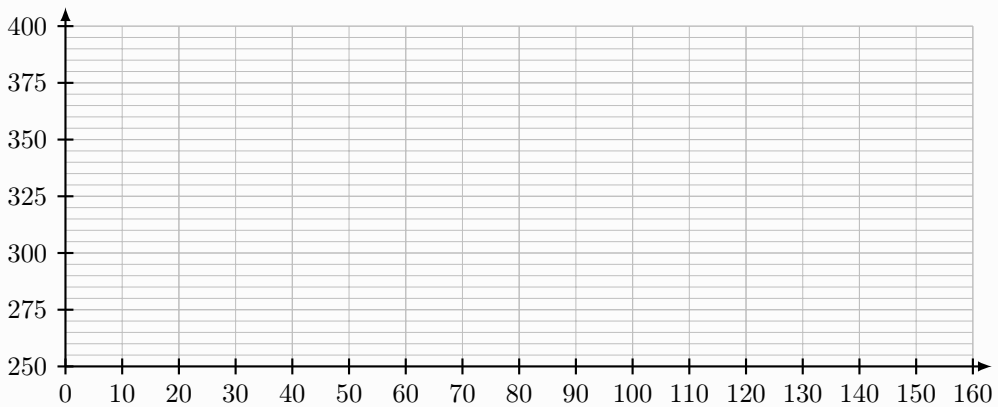
The optional `[keys]` available are:

- `Grid`: boolean (`true` by default) to display the grids (for a single grid, simply set the identical parameters for `Xgrid`/`Xgrids` or `Ygrid`/`Ygrids`);
- `Enlarge`: addition at the end of the axes (`0` by default);
- `Grads`: boolean (`true` by default) for graduations;
- `Font`: global font for graduations `empty` by default;
- `Format`: special formatting (see below) of the axis values.

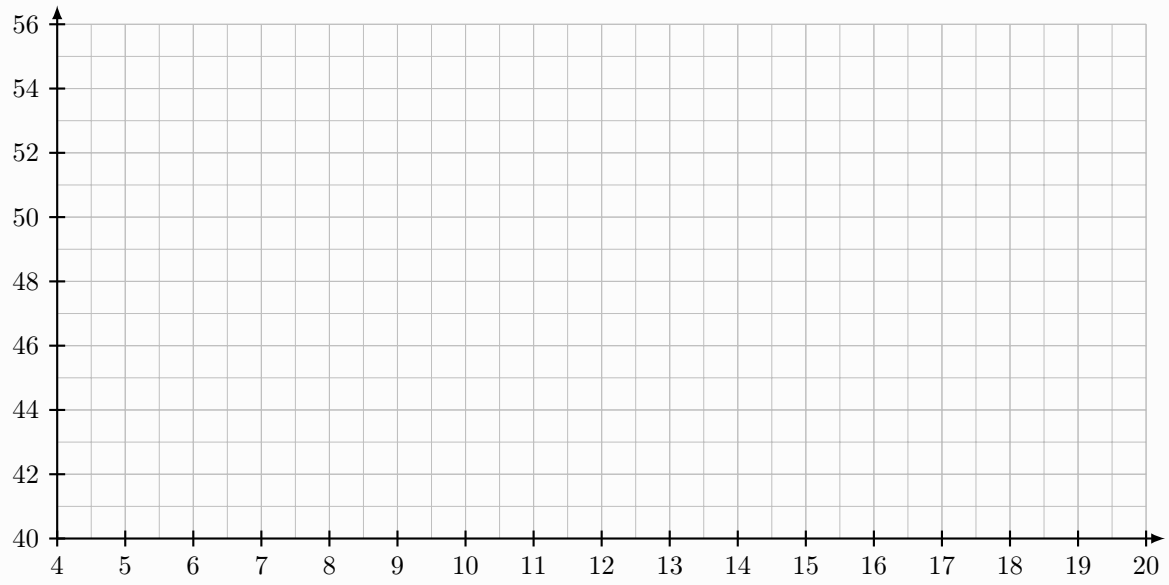
Concerning the `Format` key, it allows you to specify a specific setting for the axis values.
It can be given in the form `fmt` for combined formatting, or in the form `fmtX/fmtY` to differentiate the formatting.
The possible options are:

- `num`: format with siunitx;
- `year`: format in year;
- `frac`: format as fraction frac;
- `dfrac`: format as fraction dfrac;
- `nfrac`: format as fraction nicefrac; (to load!)
- `trig`: format in trig with frac;
- `dtrig`: format in trig with dfrac;
- `ntrig`: format in trig with nfrac;
- `sqrt`: format in root with frac;
- `dsqrt`: format in root with dfrac;
- `nsqrt`: format in root with nicefrac.

```
<MINTED>
```

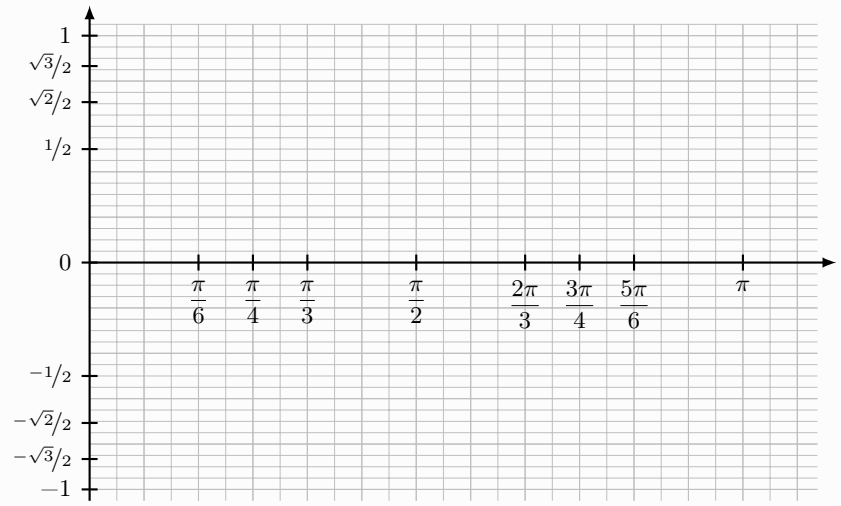


<MINTED>



Note that there are the Boolean keys `[Behind]` (without the graduations) and `[Above]` (without the grid) to display the axes in *under/over*-printing mode in the case of integrals for example.

<MINTED>



In the case where the formatting does not give satisfactory result(s), it is possible to use a generic command for placing the graduations.

2.4 Adding values manually

It is also possible to use a specific command to place values on the axes, independently of an *automated* formatting system.

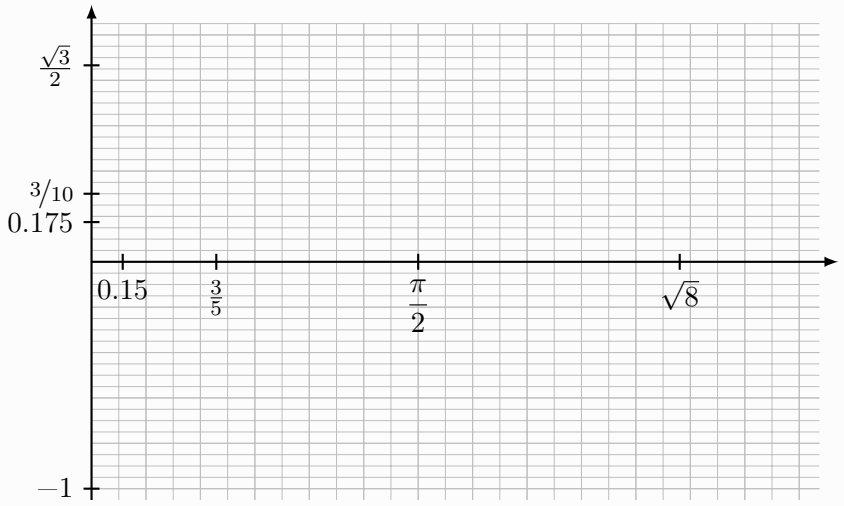
<MINTED>

The optional `[keys]` available are:

- `Font`: global font for graduations `empty` by default;
- `Lines`: boolean to add the tick marks `true` by default.

The mandatory arguments correspond to the x-coordinates (in Ti*k*Z language) and to the labels (in LaTeX language) of the graduations.

<MINTED>



3 Specific definition commands

3.1 Draw a line

The idea is to propose a command to draw a line (or an asymptote), from:

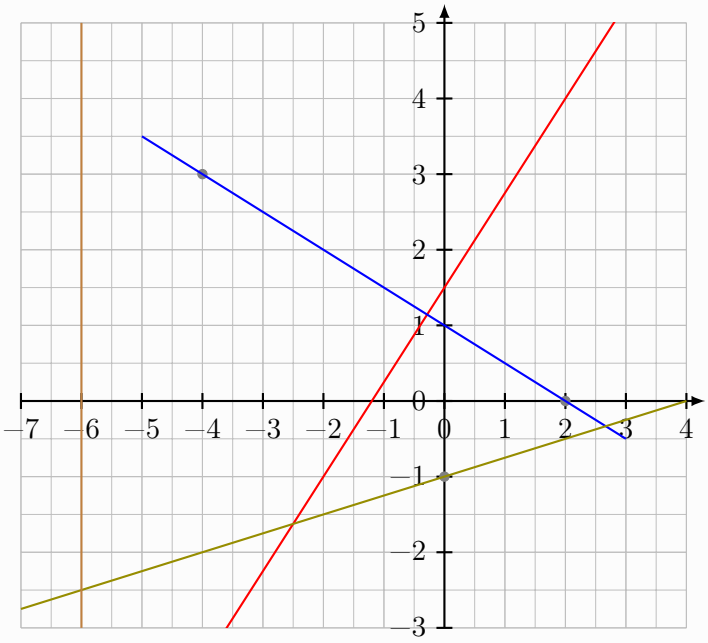
- of two points (or nodes);
- of a point (or node) and the slope.

```
<MINTED>
```

The optional `[keys]` available are:

- `Name`: possible name of the plot (for reuse);
- `Slope`: boolean to specify that the slope is used (`false` by default);
- `Start`: start of the plot (`\pflxmin` by default);
- `End`: end of the plot (`\pflxmax` by default);
- `Color`: color of the trace (`black` by default).

```
<MINTED>
```



3.2 Define a function, draw the curve of a function

The idea is to define a function, for later reuse. This command *creates* the function, without tracing it, because in certain cases elements will have to be traced beforehand.

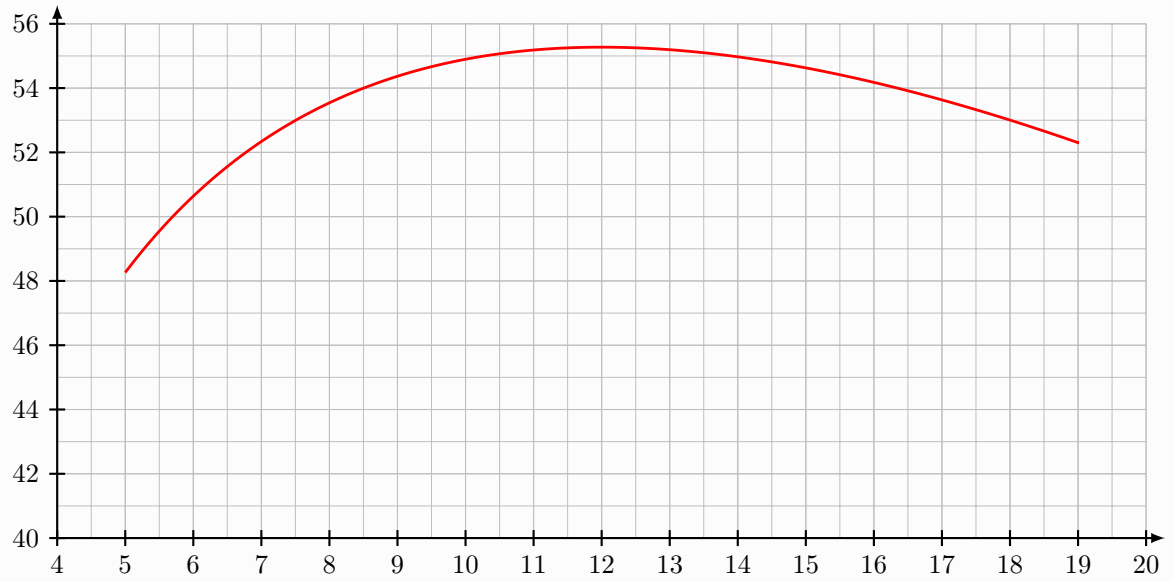
There is also a command to plot the curve of a previously defined function.

```
<MINTED>
```

The optional `[keys]` for definition or tracing are:

- `Start`: lower bound of the definition set (`\pflxmin` by default);
- `End`: lower bound of the definition set (`\pflxmax` by default);
- `Name`: name of the curve (important for the rest!);
- `Color`: color of the trace (`black` by default);
- `Step`: plot step (it is determined *automatically* at the start but can be modified);
- `Trace`: boolean to also trace the curve (`false` by default).

```
<MINTED>
```



3.3 Define/draw an interpolation curve (simple)

It is also possible to define a curve via support points, therefore a simple interpolation curve.

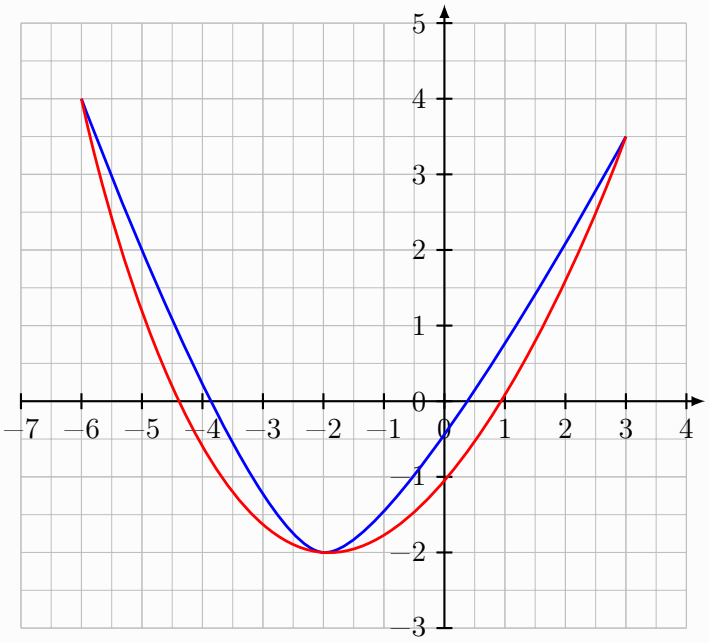
```
<MINTED>
```

The optional `[keys]` for definition or tracing are:

- `Name`: name of the interpolation curve (important for the rest!);
- `Color`: color of the trace (`black` by default);
- `Tension`: setting the *tension* of the interpolation plot (`0.5` by default);
- `Trace`: boolean to also trace the curve (`false` by default).

The mandatory argument allows you to specify the list of support points in the form `(x1,y1)(x2,y2)...`.

```
<MINTED>
```



3.4 Define/draw an interpolation curve (Hermite)

It is also possible to define a curve via support points, therefore an interpolation curve with derivative control.

Some operations require different techniques depending on the type of function used, a Boolean key `Spline` will allow the code to adapt its calculations depending on the object used.

```
<MINTED>
```

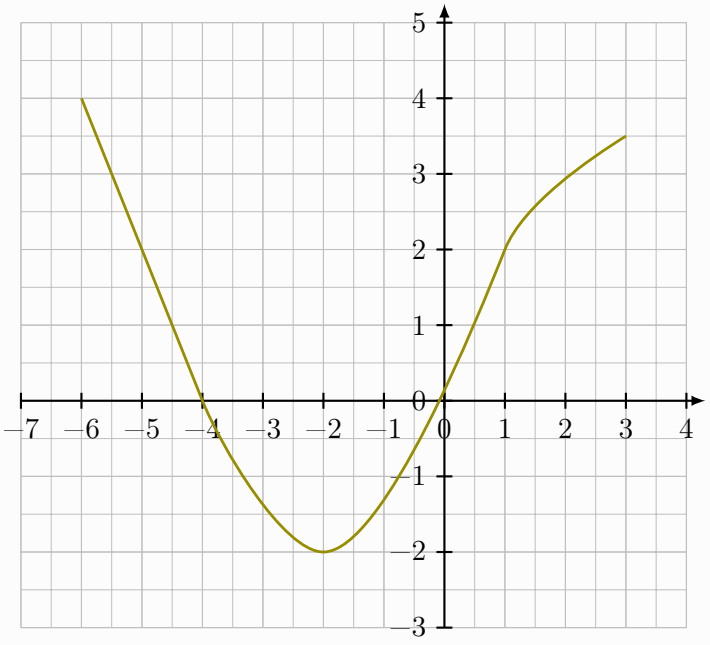
The optional `[keys]` for definition or tracing are:

- `Name`: name of the interpolation curve (important for the rest!);
- `Coeffs`: modify (see the ProfLycee[4] the *coefficients* of the spline;
- `Color`: color of the trace (`black` by default);
- `Trace`: boolean to also trace the curve (`false` by default).

The mandatory argument allows you to specify the list of support points in the form `x1/y1/f'1§x2/y2/f'2§...` with:

- `xi/yi` the coordinates of the point;
- `f'i` the derivative at the support point.

```
<MINTED>
```



[4] CTAN documentation: https://ctan.org/pkg/proflycee

3.5 Define points as nodes

The second idea is to work with Ti*k*Znodes, which could be useful for tangent plots, representations of integrals...

It is also possible to define nodes for *image* points.

Certain commands (explained later) allow you to determine particular points of curves in the form of nodes, so it seems interesting to be able to define them directly.

<MINTED>

The optional `[keys]` available are:

- `Mark`: boolean to mark points (`false` by default);
- `Color`: color of the points, if `Mark=true` (`black` by default).

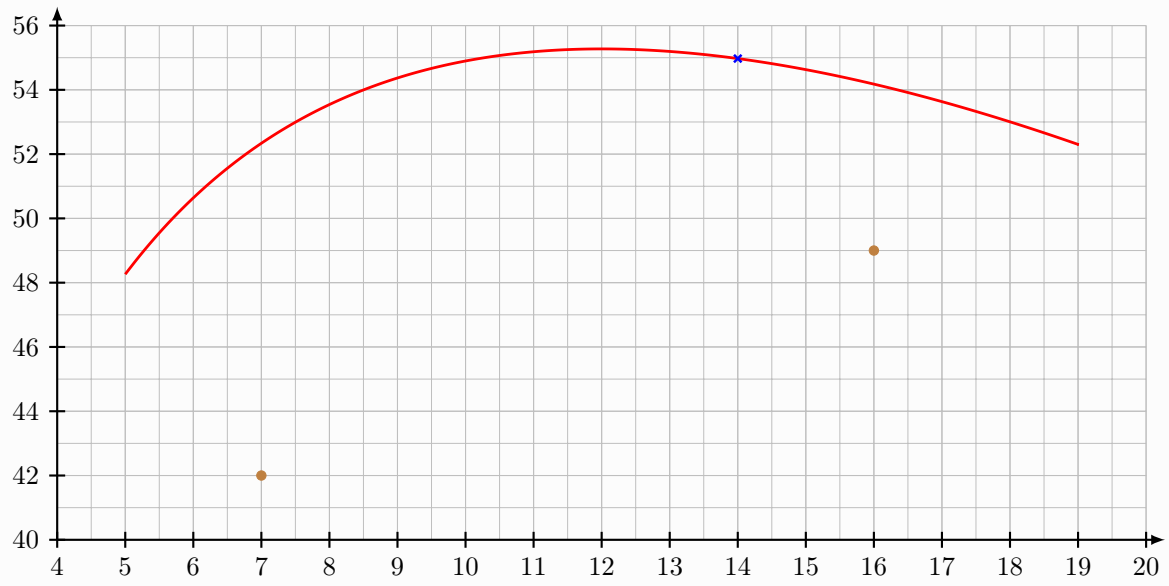
<MINTED>

The optional `[keys]` available are:

- `Name`: node name (`empty` by default);
- `Spline`: boolean to specify that a spline is used (`false` by default).

The first mandatory argument is the *object* considered (name of the curve for the spline, function otherwise); the second is the abscissa of the point considered.

<MINTED>



3.6 Mark Points

The idea is to offer something to score points with a particular style.

```
<MINTED>
```

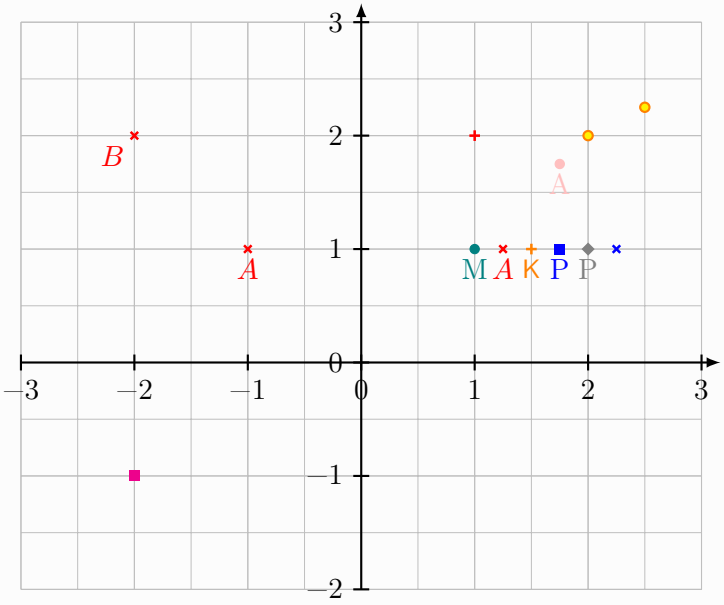
The *starred* version scores the points without the *names*, while the *unstarred* version displays them:

- in the case of the *starred* version, the list should be given in the form `(ptA),(ptB),...`;
- otherwise, the list should be given in the form `(ptA)/poslabelA/labelA,...`.

The optional `[keys]` available are:

- `Color`: color (`black` by default);
- `Style`: style of marks (`o` by default).

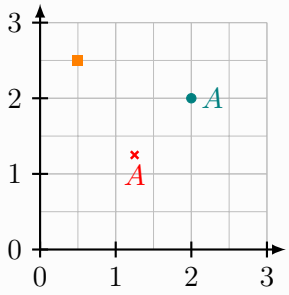
```
<MINTED>
```



Note that it is also possible to modify the size of the `o/x/+/c` marks via the `[keys]`:

- `Sizex=...` (`2pt` by default) for points *cross*;
- `Sizeo=...` (`1.75pt` by default) for the points *circle*;
- `Sizec=...` (`2pt` by default) for the *square* points.

```
<MINTED>
```



3.7 Retrieve node coordinates

It is also possible, with a view to reusing coordinates, to recover the coordinates of a node (defined or determined).
The calculations are carried out by floating according to the (re)calculated units, the values are therefore approximated !

```
<MINTED>
```

3.8 Place text

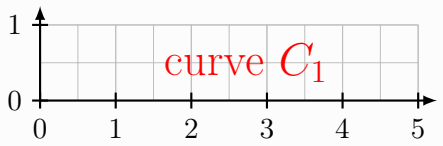
Note that a text placement command is available.

```
<MINTED>
```

The available `[keys]` are:

- `Font=...` (`\normalsize\normalfont` by default) for the font;
- `Color=...` (`black` by default) for the color;
- `Position=...` (`empty` by default) for the position of the text relative to the coordinates.

```
<MINTED>
```



4 Specific commands for using curves

4.1 Image placement

It is possible to place points (images) on a curve, with possible construction lines.
The function/curve used must have been declared previously for this command to work.

```
<MINTED>
```

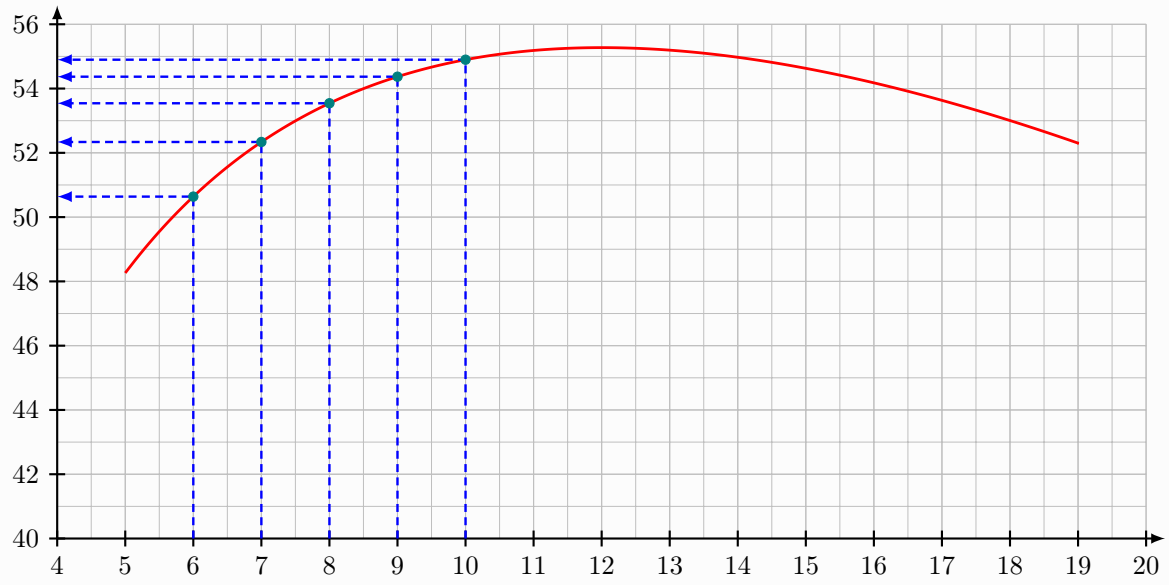
The optional `[keys]` available are:

- `Lines`: boolean to display construction traits (`false` by default);
- `Colors`: color of the points/lines, in the form `Couleurs` or `ColPoint/ColLines`;
- `Spline`: boolean to specify that the curve used is defined as a spline (`false` by default).

The first mandatory argument allows you to specify:

- the name of the curve in the case `Spline=true`;
- the name of the function otherwise.

<MINTED>



4.2 Antecedent determination

It is possible to graphically determine the antecedents of a given reality.
The function/curve used must have been declared previously for this command to work.

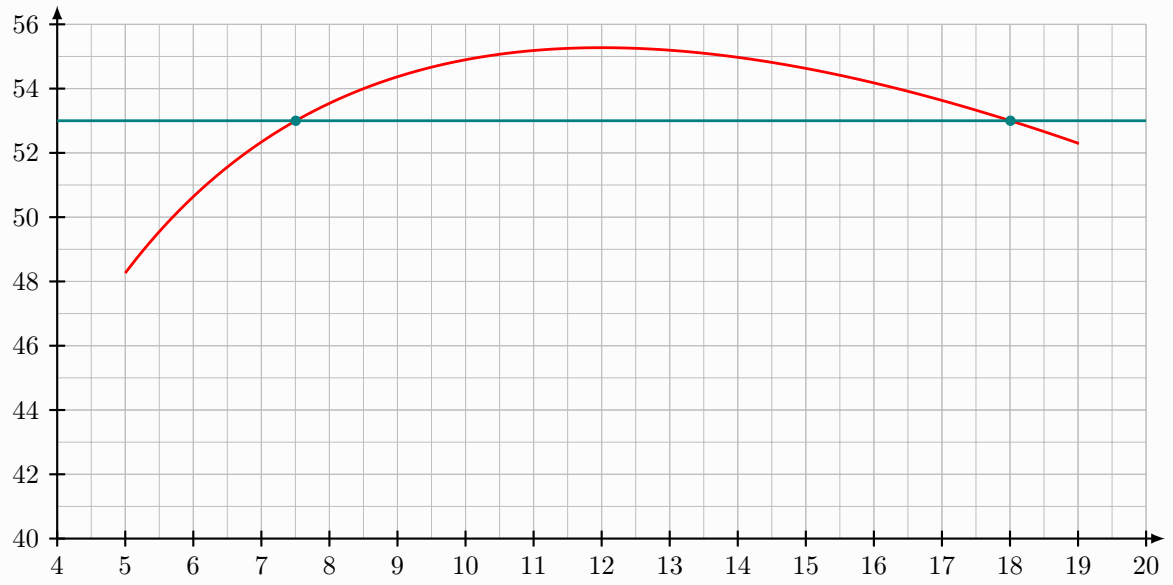
```
<MINTED>
```

The optional `[keys]` available are:

- `Name`: base of the name of the **nodes** intersection (`S` by default, which will give S-1, S-2, etc);
- `Disp`: boolean to display the points (`true` by default);
- `Color`: color of the points (`black` by default);
- `DispLine`: boolean to display the horizontal line (`false` by default).

The first mandatory argument allows you to specify the **name** of the curve.
The second mandatory argument allows you to specify the value to reach.

```
<MINTED>
```



4.3 Antecedent construction

It is possible to graphically construct the antecedents.
The function/curve used must have been declared previously for this command to work.

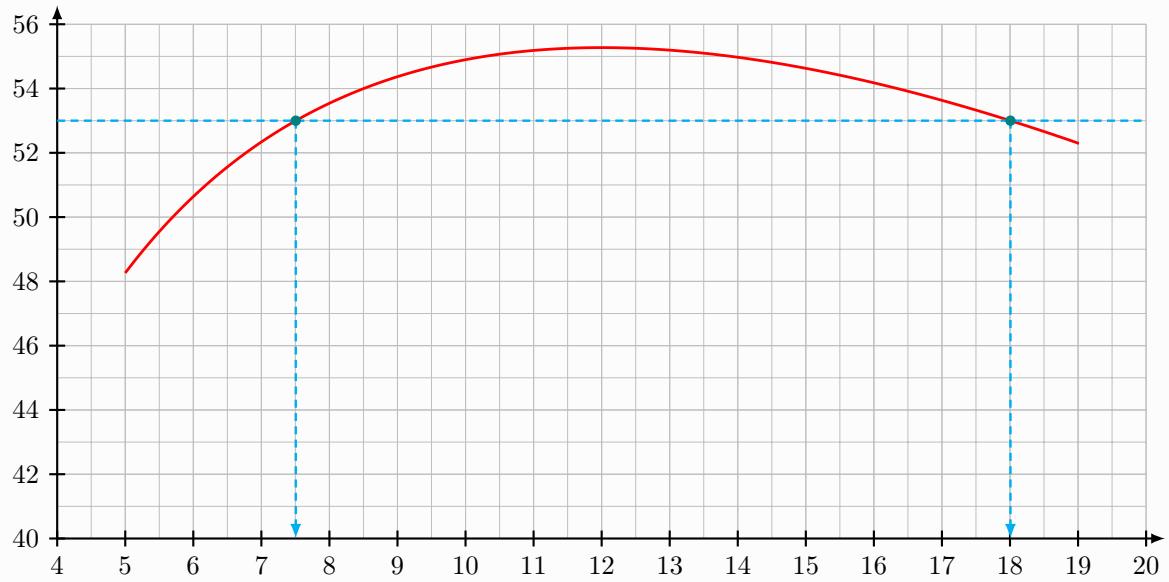
```
<MINTED>
```

The optional `[keys]` available are:

- `Colors`: color of the points/lines, in the form `Color` or `ColPoint/ColLines`;
- `Name`: name *possible* for the intersection points linked to the antecedents (`empty` by default);
- `Lines`: boolean to display construction traits (`false` by default).

The first mandatory argument allows you to specify the **name** of the curve.
The second mandatory argument allows you to specify the value to reach.

```
<MINTED>
```



Graphically, the antecedents of 53 are (approximately):

- 7.505 389 008 644 637
- 18.007 022 378 275 07

4.4 Intersections of two curves

It is also possible to determine (in the form of nodes) the possible points of intersection of two previously defined curves.

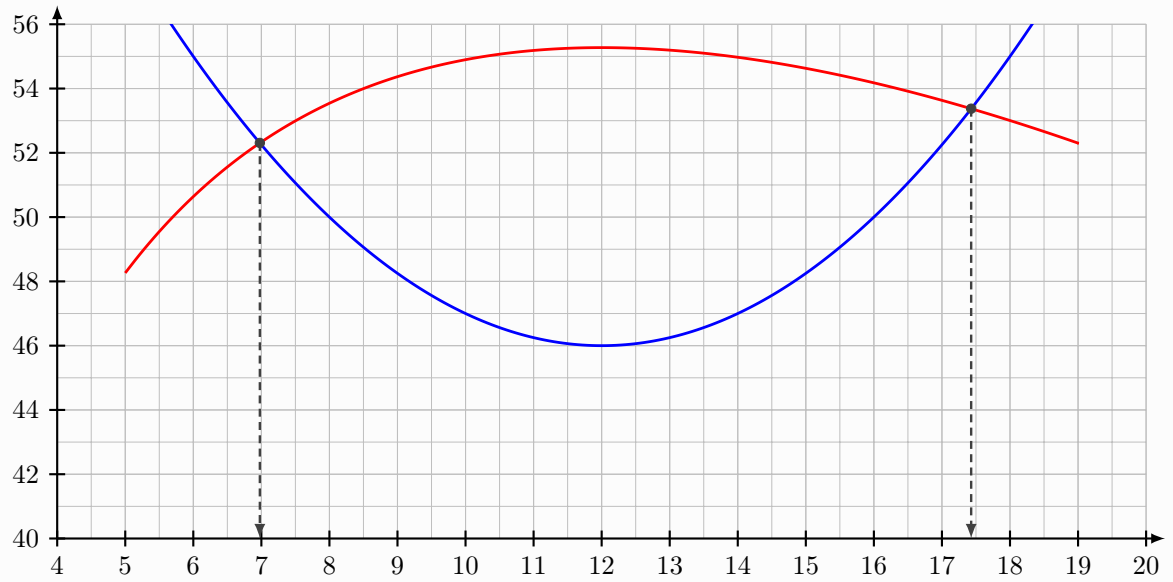
```
<MINTED>
```

The optional `[keys]` available are:

- `Name`: base of the name of the **nodes** intersection (`S` by default, which will give S-1, S-2, etc);
- `Disp`: boolean to display the points (`true` by default);
- `Color`: color of the points (`black` by default).

The first mandatory argument allows you to specify the **name** of the first curve.
The first mandatory argument allows you to specify the **name** of the second curve.

```
<MINTED>
```



The solutions of $f(x) = g(x)$ are $\alpha \approx 6.977\,766\,172\,581\,613$ and $\beta \approx 17.429\,687\,326\,385\,03$.
The points of intersection of the curves of f and g are therefore $(6.98; 52.31)$ and $(17.43; 53.37)$.

4.5 Extrema

The idea (still *experimental*) is to offer commands to extract the extrema of a curve defined by the package.
The command creates the corresponding node, and it is therefore possible to retrieve its coordinates for later use.

It is possible, by specifying it, to work on the different curves managed by the package (function, interpolation, spline).
For singular curves, it is possible that the results are not quite those expected...

💣 For the moment, the *limitations* are:

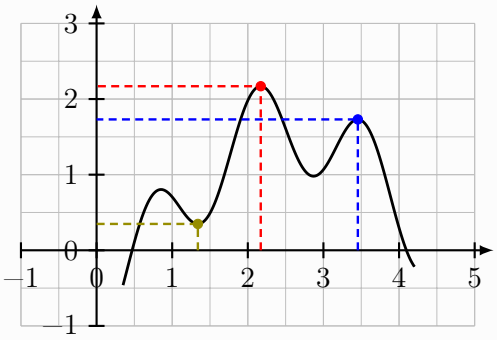
- no management of multiple extrema (only the first will be processed)...
- no management of extrema at the boundaries of the route...
- no automatic recovery of curve definition parameters...
- compilation time may be longer...

```
<MINTED>
```

The optional `[keys]` available are:

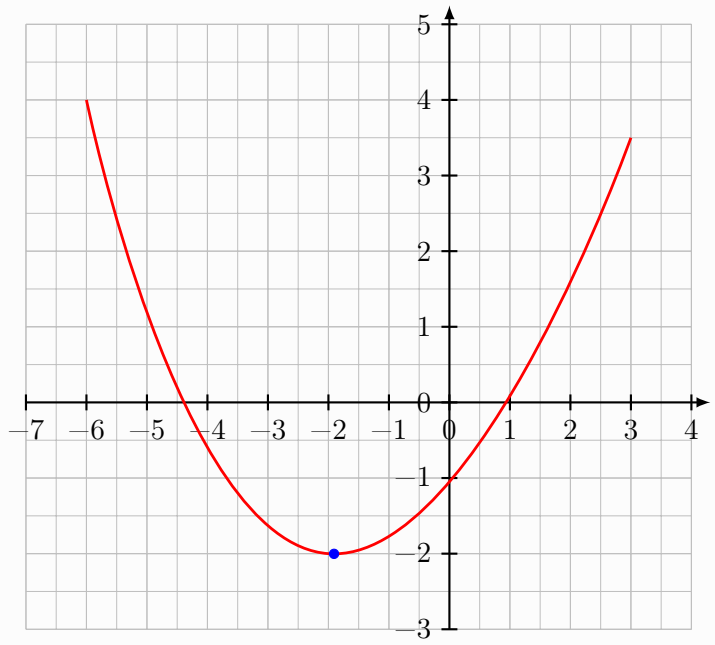
- `Method`: method, among `function/interpo/spline` for calculations (`function` by default);
- `Start`: start of the plot (`\pflxmin` by default);
- `End`: end of the plot (`\pflxmax` by default);
- `Step`: not in the plot if `function` (it is determined *automatically* at the start but can be modified);
- `Coeffs`: modify the *coefficients* of the spline if `spline`;
- `Tension`: setting the *tension* of the interpolation plot if `interpo`(`0.5` by default).

```
<MINTED>
```



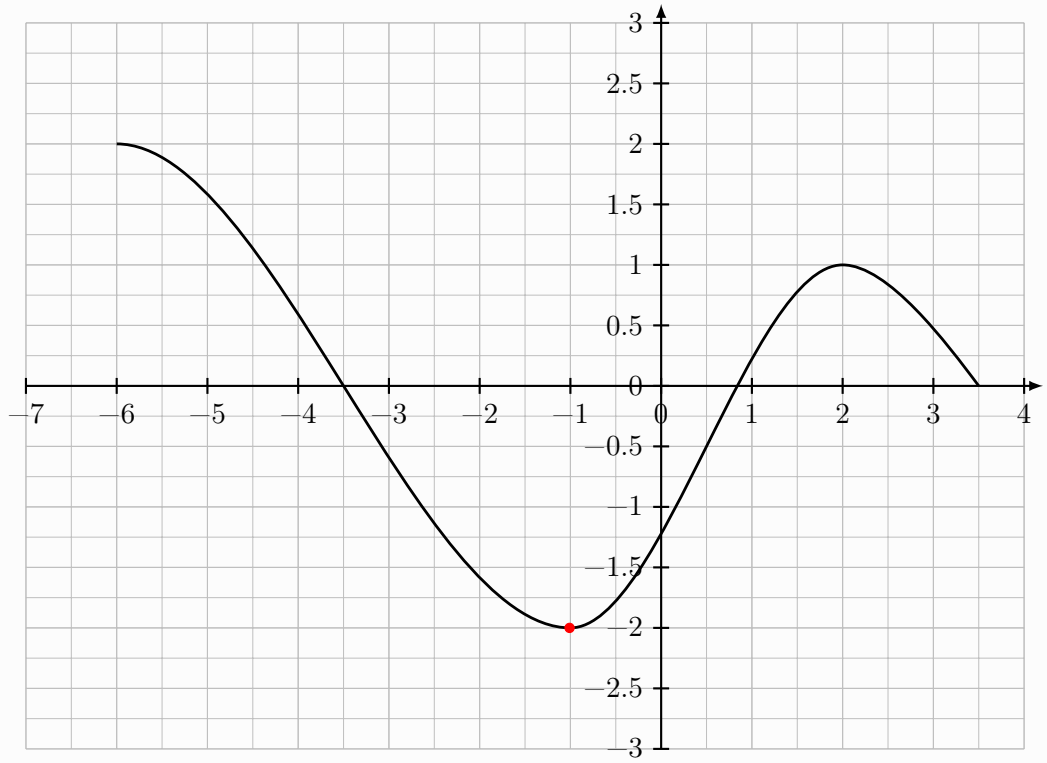
The maximum is $M \approx 2.17$, reached in $x \approx 2.17$

<MINTED>



The minimum is $M \approx -2.003$, reached at $x \approx -1.908$

<MINTED>



4.6 Integrals (improved version)

We can also work with integrals.
In this case it is preferable to highlight the domain **before** the plots, to avoid overprinting in relation to the curves/points.
It is possible to :

- represent an integral **under** a defined curve;
- represent an integral **between** two curves;
- the integration limits can be x-coordinates and/or nodes.

💣 Given the differences in processing between formula curves, simple interpolation curves or cubic interpolation curves, the arguments and keys may differ depending on the configuration!

```
<MINTED>
```

The optional `[keys]` for definition or tracing are:

- `Colors` =: colors of the filling, in the form `Col` or `ColBorder/ColBg` (`gray` by default);
- `Style`: type of filling, among `fill`/`hatch` (`fill` by default);
- `Opacity`: opacity (`0.5` by default) of the filling;
- `Hatch`: style (`north west lines` by default) of the hatch filling;
- `Type`: type of integral among
  - `fct` (default) for an integral under a curve defined by a formula;
  - `spl` for an integral under a curve defined by a cubic spline;
  - `itp` for an integral under a curve defined by interpolation ;
  - `fct/fct` for an integral between two curves defined by a formula;
  - `fct/spl` for an integral between a curve (above) defined by a formula and a curve (below) defined by a spline cubic;
  - etc.
- `Step`: steps (calculated by default otherwise) for the plot;
- `Junction`: junction of segments (`bevel` by default);
- `Bounds`: type of terminals among:
  - `abs` for the limits given by the abscissa;
  - `nodes` for the limits given by the nodes;
  - `abs/node` for the limits given by abscissa and node;
  - `node/abs` for the limits given by node and abscissa;
- `Border`: boolean (`true` by default) to display the side lines,
- `SplineName`: macro (important!) of the spline generated previously for a higher version spline;
- `SplineNameB`: macro (important!) of the spline generated previously for a lower version spline;
- `InterpoName`: name (important!) of the interpolation curve generated previously, in higher version;

- `InterpoBName`: name (important!) of the interpolation curve generated previously, in lower version;
- `Tension`: Tension for the interpolation curve generated previously, in higher version;
- `TensionB`: Tension of the interpolation curve generated previously, in lower version.

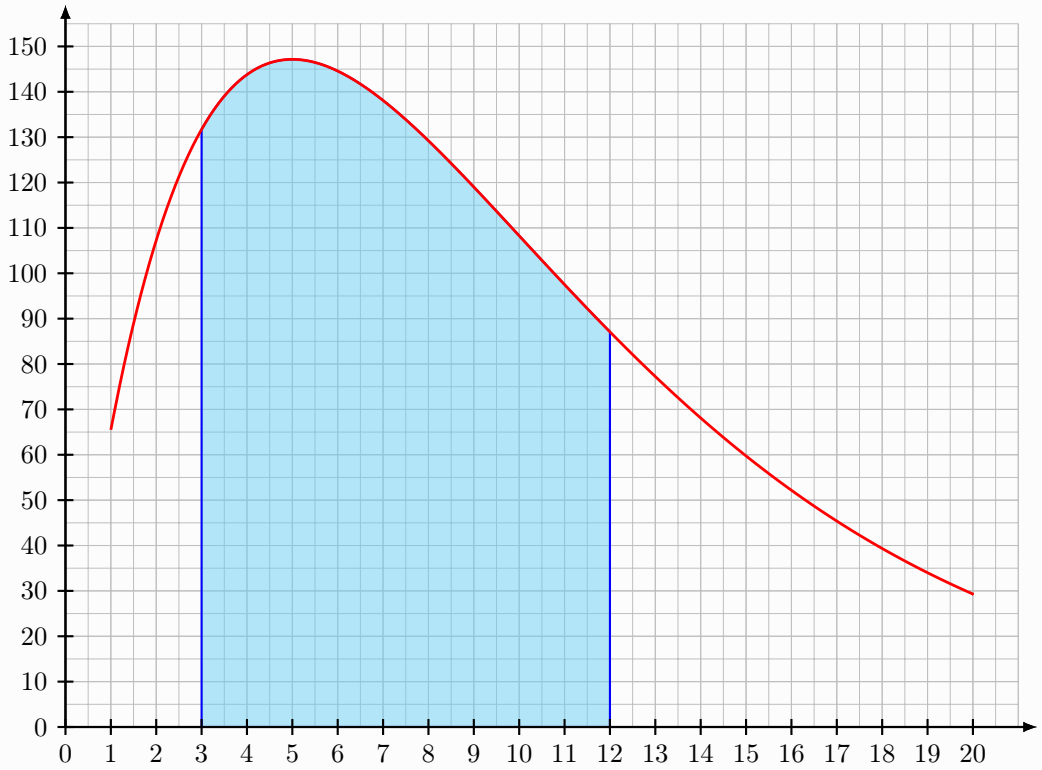
The first required argument is the spline function or curve or list of interpolation points.

The next optional argument is the spline function or curve or list of interpolation points.

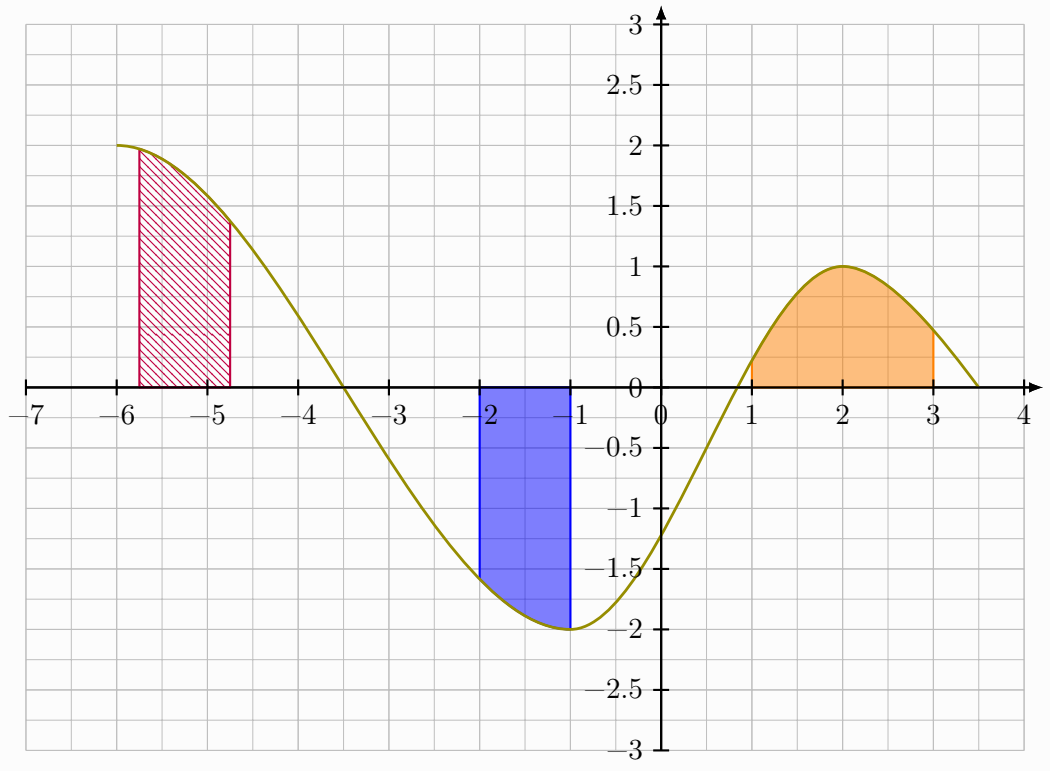
The last two mandatory arguments are the limits of the integral, given in a form consistent with the key `Bounds`.

In the case of curves defined by *points*, it is necessary to work on intervals on which the first curve is **above** the second.
It will undoubtedly be interesting to work with *intersections* in this case.

<MINTED>



<MINTED>



4.7 Tangents

The idea of this command is to draw the tangent to a previously defined curve, specifying:

- the point (abscissa or node) at which we wish to work;
- possibly the direction (in the case of a discontinuity or a terminal);
- possibly the step (h) of the calculation;
- the *lateral spacings* to draw the tangent.

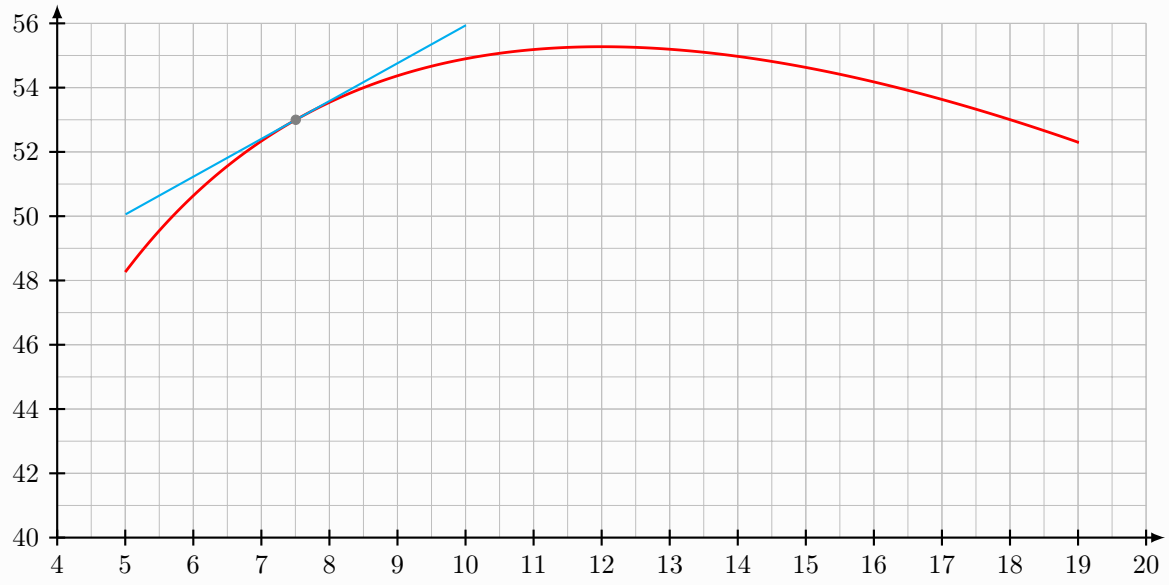
```
<MINTED>
```

The optional `[keys]` for definition or tracing are:

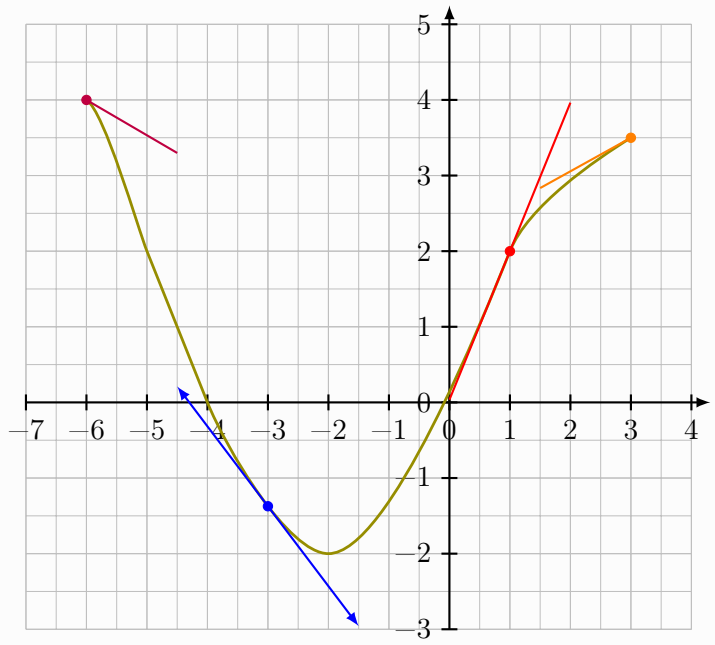
- `Colors` =: colors of the plots, in the form `Col` or `ColLine/ColPoint` (`black` by default);
- `OffsetL` =: left horizontal spacing to start the trace (`1` by default);
- `OffsetR` =: left horizontal spacing to start the trace (`1` by default);
- `DispPt`: boolean to display the support point (`false` by default);
- `Spline`: boolean to specify that a spline is used (`false` by default);
- `h`: delta h used for calculations (`0.01` by default);
- `Direction`: allows you to specify the *direction* of the tangent, among `lr`/`l`/`r` (`lr` by default);
- `Node`: boolean to specify that a node is used (`false` by default).

The first required argument is the spline function or curve (if applicable).
The last mandatory argument is the work point (abscissa version or node following the key `Node`).

```
<MINTED>
```



<MINTED>



5 Commands specific to two-variable statistics

5.1 Limitations

Given the specific features of Ti*k*Z, we advise you not to use values that are too *large* at axis level (this can cause problems with years, for example), or else you'll have to *transform* axis and/or data values so that everything is displayed as it should be (also beware of regressions, calculations, etc.).

5.2 The point scatter

In addition to commands linked to functions, it is also possible to represent double statistical series.
The following paragraph shows that adding a key allows you to add the linear adjustment line.

```
<MINTED>
```

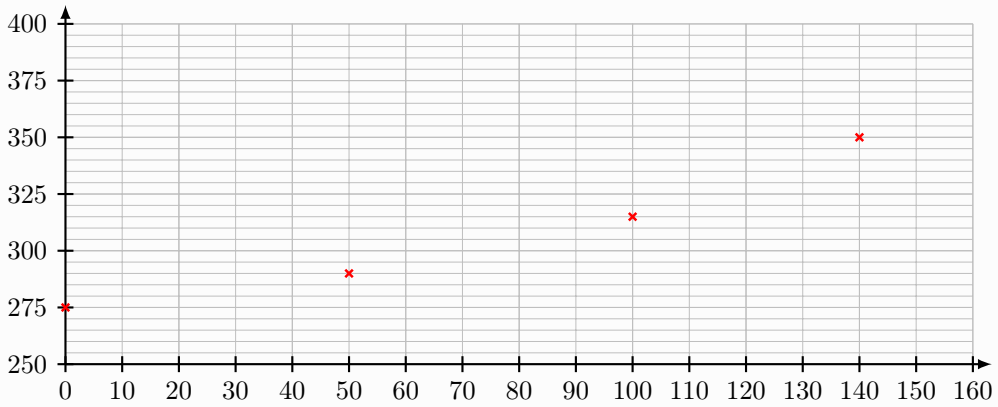
The optional `[key]` is:

- `ColorScatter`: color of the cloud points (`black` by default).

The mandatory arguments allow you to specify:

- the list of x;
- the list of y.

```
<MINTED>
```

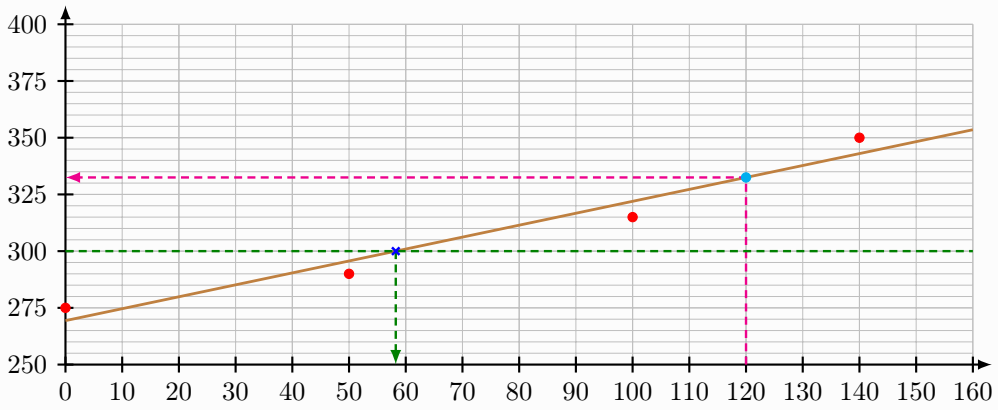


5.3 The regression line

The linear regression line (obtained by the least squares method) can easily be added, using the key `DrawLine`.
In this case, new keys are available:

- `ColorLine`: color of the line (`black` by default);
- `Rounds`: precision of coefficients (`empty` by default);
- `Start`: initial abscissa of the plot (`\pflxmin` by default);
- `End`: terminal abscissa of the plot (`\pflxmax` by default);
- `Name`: name of the line, for later use (`reglin` by default).

<MINTED>



5.4 Other regressions

In partnership with the `xint-regression` package, loaded by the package (but *can be deactivated* via the `[noxintreg]` option), it is possible to work on other types of regression:

- linear $ax+b$;
- quadratic ax^2+bx+c;
- cubic ax^3+bx^2+cx+d;
- power ax^b;
- exponential ab^x or e^{ax+b} or be^{ax} or $C+be^{ax}$;
- logarithmic $a+b\ln(x)$;
- hyperbolic $a+\dfrac{b}{x}$.

The command, similar to that of defining a curve, is:

<MINTED>

The `[keys]` available are, classically:

- `Start`: lower bound of the definition set (`\pflxmin` by default);
- `End`: lower bound of the definition set (`\pflxmax` by default);
- `Name`: name of the curve (important for the rest!);
- `Color`: color of the trace (`black` by default);
- `Step`: plot step (it is determined *automatically* at the start but can be modified).

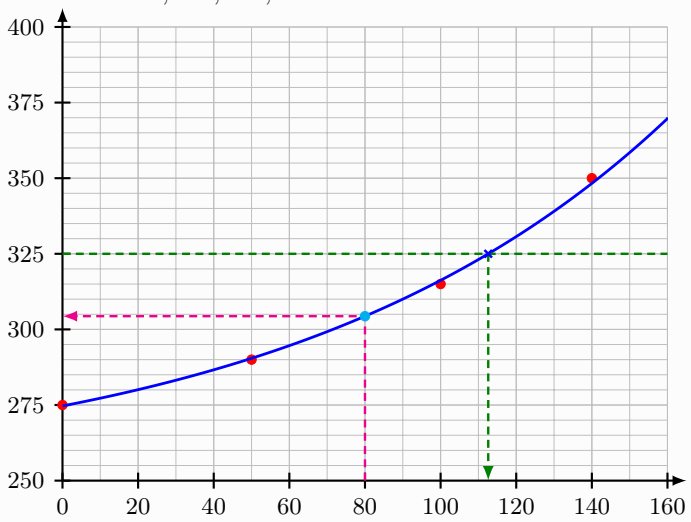
The second argument, optional and between `<...>`, allows you to name the regression function.
The third argument, mandatory and between `{...}` allows you to choose the type of regression, among:

- `lin`: linear $ax+b$;
- `quad`: quadratic ax^2+bx+c;
- `cub`: cubic ax^3+bx^2+cx+d;
- `pow`: power ax^b;
- `expab`: exponential ab^x
- `hyp`: hyperbolic $a+\dfrac{b}{x}$;
- `log`: logarithmic $a+b\ln(x)$;
- `exp`: exponential e^{ax+b};
- `expalt`: exponential be^{ax};
- `expoff=C`: exponential $C+be^{ax}$.

The fourth argument, optional and between `<...>`, allows you to specify the rounding(s) for the coefficients of the regression function.
The last two arguments are the lists of values of X and Y.

```
<MINTED>
```

ListX := 0,50,100,140
ListY := 275,290,315,350



We obtain $y = 250 + 24.7\mathrm{e}^{0.01x}$

6 Auxiliary commands

6.1 Intro

In addition to purely *graphic* commands, some auxiliary commands are available:

- a to format a number with a given precision;
- one for working on random numbers, with constraints.

6.2 Formatted rounding

The `\RoundNb` command allows you to format, using the `siunitx` package, a number (or a calculation), with a given precision. This can be *useful* for formatting results obtained using coordinate retrieval commands, for example.

```
<MINTED>
```

```
<MINTED>
```

```
0.33
16.1
2.303
```

6.3 Random number under constraints

The idea of this second command is to be able to determine a random number:

- integer or decimal;
- under constraints (between two fixed values).

This can allow, for example, to work on curves with *random* points, but respecting certain constraints.

```
<MINTED>
```

The star version takes the constraints in strict form (lower bound $<$ macro $<$ upper bound) while the normal version takes the constraints in broad form (lower bound) $\leq$ macro $\leq$ upper bound).

Note that the *terminals* can be existing *macros*!

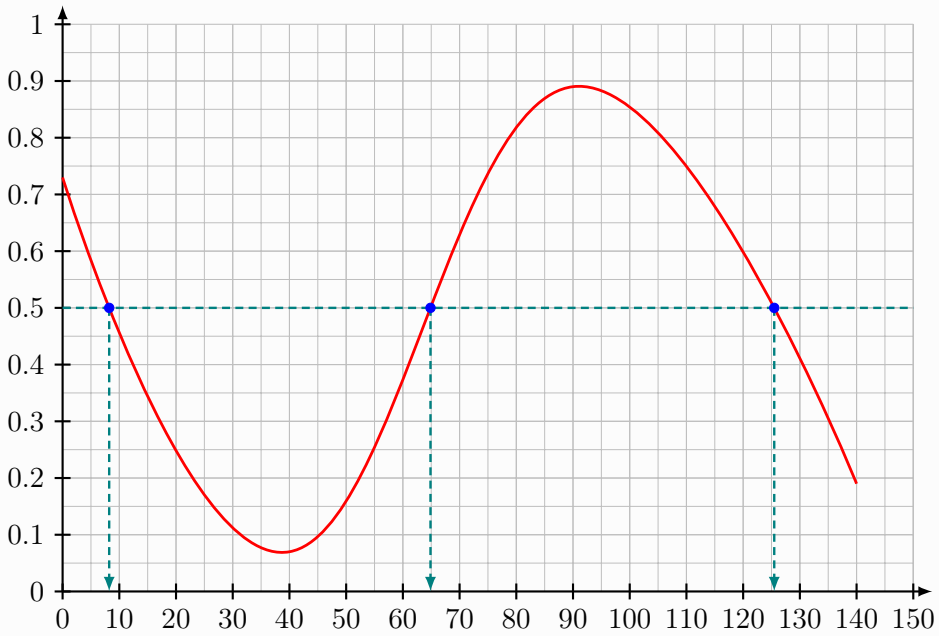
```
<MINTED>
```

```
0.95 & 0.07 & 0.91 & 0.12
```

```
<MINTED>
```

```
0.92 & 0.1 & 0.59 & 0.19
```

<MINTED>

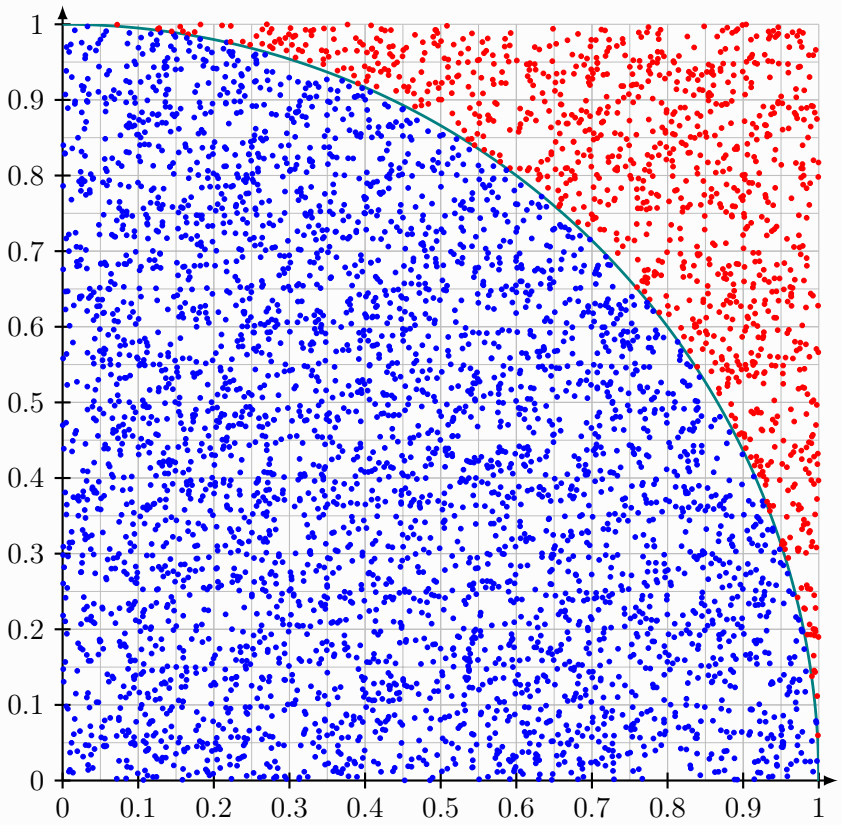


The solutions of $f(x) = 0.5$ are, by graphic reading: $\begin{cases} \alpha \approx 8 \\ \beta \approx 65 \\ \gamma \approx 125 \end{cases}$.

6.4 Monte-Carle method

<MINTED>

<MINTED>



There is 3956 blue points, there is 1044 red points.
And $\frac{3956}{5000} \approx 0.7912$ et $\frac{\pi}{4} \approx 0.7854$.

7 History

0.1.9 : Bugfix
0.1.8 : New commands [in french doc] (binomial, cabweb,\ldots)
0.1.6 : Vertical asymptote + [in french doc] commands for integrals
0.1.5 : Initial version [en]