# `gauss.sty` – A Package for Typesetting Matrix Operations

Manuel Kauers

October 11, 2002

**Abstract**

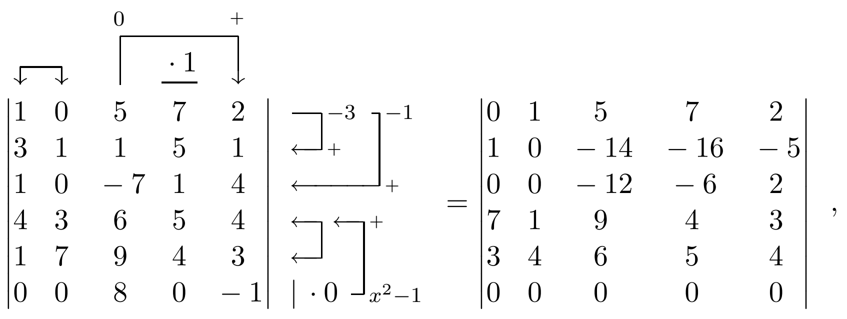
This package provides LaTeX-macros for typesetting operations on a matrix. By an "operation on a matrix" we understand a *row operation* or a *column operation*.

The user interface of the package is very straightforward and easy to understand while the results look quite pretty.

## Contents

## 1 Usage

If you find yourself in search of a package that enables you to easily typeset constructions like



then this package is what you need. It defines a new matrix environment which is extended by comprehensive macros for typesetting so-called “operations” on the matrix. An operation is either a row operation or a column operation, and may involve one or two lines. Examples of such operations arise in the context of Gaussian elimination for solving systems of linear equations in linear algebra: swaping rows, adding the multiple of one row to another, and multiply a row by a constant factor.

### 1.1 How to typeset matrix operations

`gmatrix` The package defines a new matrix environment `gmatrix` which behaves just like LaTeX’s and AMSLaTeX’s `matrix`. It takes an optional parameter ⟨*delimtype*⟩ to select the matrix delimiters. So, `gmatrix[p]` corresponds to `pmatrix`, `gmatrix[v]` to `vmatrix`, and so on.

The `gmatrix` environment can be used exaclty like its brothers and sisters defined by LaTeX and AMSLaTeX, for instance:

```
\begin{gmatrix}[p]
 a & b \\
 c & d
\end{gmatrix}
```

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

The content of the `gmatrix` environment consists of three parts: matrix, row operations, and column operations. The latter two are optional parts, and the ordering of them is arbitrary (i.e. row operations may be stated before column operations and vice versa). The matrix part is required, and it must be the first one.

`\rowops` `\colops` To skip to the next section, there are two comands `\rowops` which swiches to the row operation section, and `\colops` which switches to the column operation section.

`\mult` `\add` `\swap` Within the operation sections, you have to state the sequence of operations which are to be typeset. There are the three commands `\mult`, `\add`, and `\swap` to do this. These commands are specified as follows:

1. `\mult{i}{\cdot b}` typesets the operation "multiply the $i$th row (or column) by $b$",

2. `\swap[a][b]{i}{j}` typesets the operation "swap the $i$th and the $j$th row (or column)". $a$ and $b$ are labels to typeset at the end of the arrows, similar to the $\cdot b$ of the `\mult` command. The command does nothing if $i = j$.

3. `\add[a][b]{i}{j}` typesets the operation "add the $a$-fold of row (or column) $i$ to row (or column) $j$. $b$ is a label for the $j$th line. The command does nothing if $i = j$.

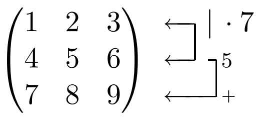
In the standard implementation, optional arguments of `\swap` and the second optional argument of `\add` are ignored. See Section 1.3 for how to enable them.

Rows are counted top-down, and columns are counted from left to right. The uppermost row and the leftmost column have the index 0. There is also the posibility to use `*` as index which causes the typesetting of several operations where `*` runs over all indices. For example, `\mult{*}{5}` in the `\rowops` section of a $n \times n$ matrix is equivalent to state `\mult{0}{5}`,…,`\mult{`$n-1$`}{5}`.

## 1.2 Examples

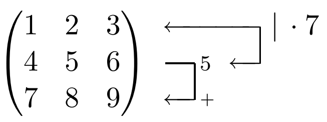
- A matrix with row operations

```
\begin{gmatrix}[p]
 1 & 2 & 3 \\
 4 & 5 & 6 \\
 7 & 8 & 9
\rowops
 \swap{0}{1}
 \mult{0}{\cdot 7}
 \add[5]{1}{2}
\end{gmatrix}
```



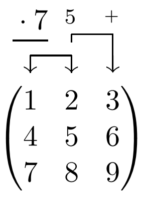
- The same operations in an other ordering

```
\begin{gmatrix}[p]
 1 & 2 & 3 \\
 4 & 5 & 6 \\
 7 & 8 & 9
\rowops
 \add[5]{1}{2}
 \swap{0}{1}
 \mult{0}{\cdot 7}
\end{gmatrix}
```



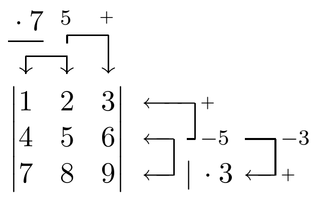
- A matrix with column operations

```
\begin{gmatrix}[p]
 1 & 2 & 3 \\
 4 & 5 & 6 \\
 7 & 8 & 9
\colops
 \swap{0}{1}
 \mult{0}{\cdot 7}
 \add[5]{1}{2}
\end{gmatrix}
```



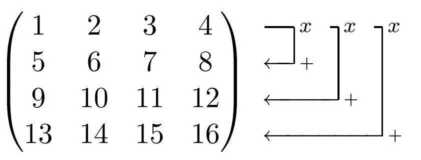
- A matrix with both row and column operations

```
\begin{gmatrix}[v]
 1 & 2 & 3 \\
 4 & 5 & 6 \\
 7 & 8 & 9
\rowops
 \swap{1}{2}
 \mult{2}{\cdot 3}
 \add[-5]{1}{0}
 \add[-3]{1}{2}
\colops
 \swap{0}{1}
 \mult{0}{\cdot 7}
 \add[5]{1}{2}
\end{gmatrix}
```



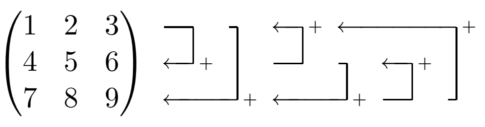
- Multiple operations using the `*` index

```
\begin{gmatrix}[p]
 1&2&3&4\\
 5&6&7&8\\
 9&10&11&12\\
 13&14&15&16
\rowops
 \add[x]{0}{*}
\end{gmatrix}
```

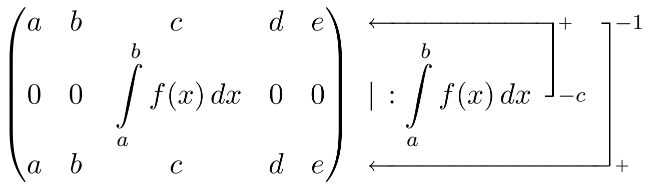


Note that the first row is not added to itself, because `\add[x]{0}{0}` has no effect. You can also use two stars:

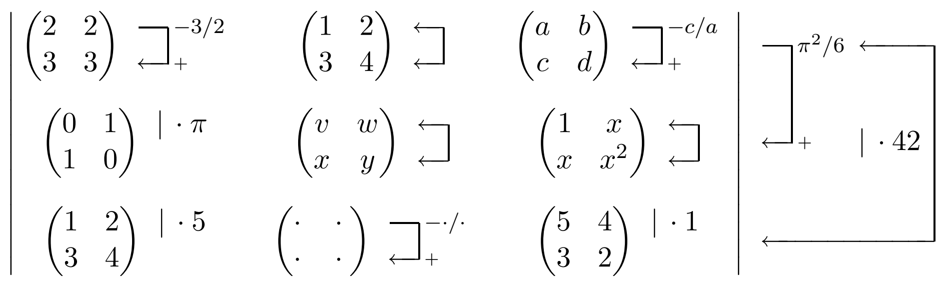
```
\begin{gmatrix}[p]
 1&2&3\\
 4&5&6\\
 7&8&9
\rowops
 \add{*}{*}
\end{gmatrix}
```



- The package clearly also handels a matrix with larger entries correctly:



Even nested `gmatrix`es are possible:

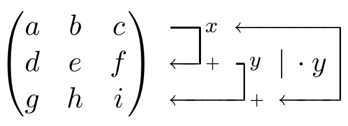


## 1.3 Adapting the package

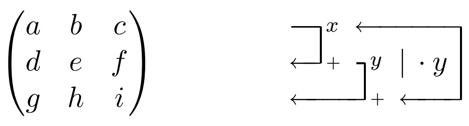
### 1.3.1 Distances and dimensions

The appearance of the operation lines and arrows depends strongly on the values of the dimension parameters described in this section.

`\rowarrowsep` `\colarrowsep` `\rowarrowsep` denotes the distance from the matrix to the operations. For example, `\rowarrowsep=10pt` yields

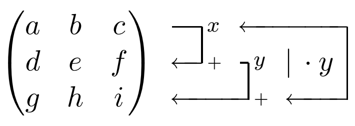


and `\rowarrowsep=50pt` yiels

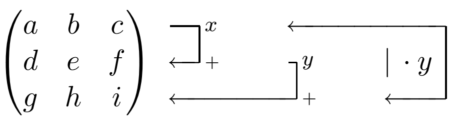


The corresponding dimension for column operations is `\colarrowsep`.

`\opskip` `\opskip` is the distance between two consecutive operations. For example, `\opskip=6pt` yields

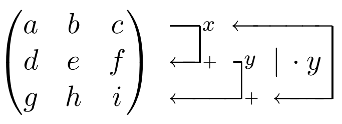


and `\opskip=30pt` yields

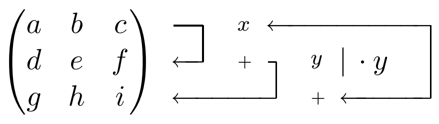


The `\opskip` length is responsible for both row and column operations.

`\labelskip` `\labelskip` is the distance between an operation arrow and its labels. For example, `\labelskip=3pt` yields

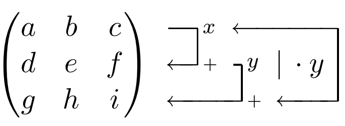


and `\labelskip=15pt` yields

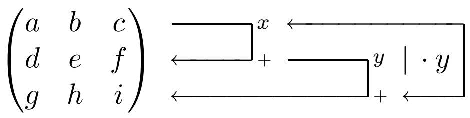


The `\labelskip` length is responsible for both row and column operations.

`\rowopminsize` `\colopminsize` The length `\rowopminsize` is the minimum amount of a horizontal operation segment to go to the right. For example, `\rowopminsize=3pt` yields



If the horizontal segment ends with an arrow tip and `\rowopminsize` is less than the width of `\leftarrow`, then the width of `\leftarrow` is taken. In the above example, this is the case in the `\add[x]{0}{1}` operation. An example for an exact use of a small value of `\rowopminsize` is the upper horizontal line of `\add[y]{1}{2}`. For comparation, `\rowopminsize=30pt` yields



The corresponding value for column operations is `\colopminsize`.

### 1.3.2 Labels

The typesetting of a label can be changed by redefining the macros which are responsible for label typesetting. Each label parameter of `\mult`, `\add`, and `\swap` is passed to special "fontifier" macros which take one argument and fontify it according to the semantical requirements. Here is a list of those fontifier macros and their default definitions:

`\rowmultlabel` `\rowmultlabel` is the label of a `\mult` operation in the `\rowops` section. Its default definition is `{|\,#1}`.

`\colmultlabel` `\colmultlabel` is the respective macro for the `\colops` section. It is defined to

```
\underline{\hbox to 1.2em{$\hss\mathstrut{}#1\hss$}}
```

by default.

`\rowswapfromlabel` `\rowswapfromlabel` is the label of a `\swap` operation in the `\rowops` section which is to place at the first of the two rows. It is defaultly defined to `{}`, i.e. the label parameter is ignored.

`\colswapfromlabel` `\colswapfromlabel` is the respective macro for the `\colops` section which is also empty by default.

`\rowswaptolabel` `\rowswaptolabel` is like `\rowswapfromlabel`, but for the other row. It is empy by default.

`\colswaptolabel` `\colswaptolabel` is `\rowswaptolabel`'s brother for the `\colops` section.

`\rowaddfromlabel` `\rowaddfromlabel` is the macro for the label of the from-line of an `\add` command. It is defined to `{\scriptstyle#1}` by default.

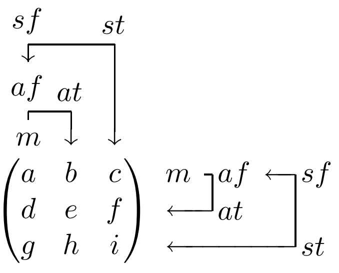
`\coladdfromlabel` `\coladdfromlabel` is respective macro for the column operations.

`\rowaddtolabel` `\rowaddtolabel` fontifies the label of the to-line of an `\add` command. This macro is defined to `{\scriptscriptstyle +}` by default, i.e. it ignores the parameter.

`\coladdtolabel` `\coladdtolabel` is the respective command for the column operation. It behaves likewise.

For the following example, all of the above labels were defined to `{#1}`, i.e. to identity.

```
\begin{gmatrix}[p]
 a & b & c \\
 d & e & f \\
 g & h & i
\colops
 \mult0{m}
 \add[af][at]01
 \swap[sf][st]02
\rowops
 \mult0{m}
 \add[af][at]01
 \swap[sf][st]02
\end{gmatrix}
```



### 1.3.3 Matrix delimiters

`\newmatrix` It is possible to define new delimiter specifiers to `gmatrix`, say `gmatrix[X]`, by defining a matrix environment `Xmatrix`. A definition of `Xmatrix` which fulfills the requirements needed for compatibility with `gmatrix` is provided automatically by the call of

`\newmatrix{`⟨*left-delim*⟩`}{`⟨*right-delim*⟩`}{X}`, which defines the environment `Xmatrix`. The arguments ⟨*left-delim*⟩ and ⟨*right-delim*⟩ need to be compatible to the `\left-\right` mechanism of TEX. As soon as `Xmatrix` exists, it is also possible to use `X` as optional argument to `gmatrix`.

By convention, the suffix is one single character. If you try to enter `g@` or the empty string as suffix, nothing is done, otherwise, the definition works as well.

## 1.4 Features

- You need not care about the width or height of some macro cells, operations are always aligned well, i.e. centered to the column or row.
- Operation elements will not intersect each other, unless you give some very huge labels.
- There is no restriction to the order of operation commands, so you can choose an arbitrary order to achive the best typographic result.
- If no operations are given, the result is exactly the result of the $\mathcal{AMS}$-TEX `matrix` environment.
- Unlike $\mathcal{AMS}$'s `matrix` environment, there is no limit to the matrix' size in our reimplementation `gmatrix`.
- Nested `gmatrix`'s are possible.

## 1.5 Trap doors and hints

- The last row *must not* end with an `\\`, but each other line should end with `\\`.
- The last row is used internally to measure the column's widths. Therefore, if you want to point to a column $i$, then the last row must have at least $i + 1$ entries.
- In row operations, the package considers the width of labels (that is, the width of factors in `\mult` and `\add`). But you have to take care that your labels are not higher than the corresponding line, otherwise they may intersect with other arrows or labels.
- analogously for column operations.

- The package should also run without the `amsmath` package, but if you use that package (which is assumed to be the usual situation), you have to load `gauss` after `amsmath`.

### 1.6 Bug parade

A list of submitted bugs and suggested work-arounds or fixes. If you face any bug that is not in the list below, feel free to contact me at `manuel@kauers.de`.

- Hans Frederik Nordhaug faced problems with versions of AMS-LaTeX that don't define `*matrix` environments as expected (e.g. version 2.13). The current version of `gauss` therefore redefines all those environments using our `\newmatrix` tool, and requires `amsmath` to be loaded prior to the `gauss` package.

- Morten Høgholm suggested the introduction of fontifying macros and the use of changeable lengths as discussed in Section 1.3. He also suggested some very fine typographic tunings.

- Herbert Voss found that a `\unitlength=1pt` was missing to make the behaviour of the package independent of redefinitions of `\unitlength` outside `gmatrix`.

## 2 Implementation

```
\ProvidesPackage{gauss}[2002/10/11]
\RequirePackage{amsmath}
\makeatletter
```

To avoid naming conflicts with other packages, our private control sequences all start with `\g@`. Permanently public are only the `gmatrix` environment, the fontifying macros (like `\rowaddfromlabel`), and the dimensions (like `\opskip`).

The `amsmath` package is not necessarily needed, but if used, it has to be loaded prior to the `gauss` package. This is forced by the `\RequirePackage` command.

### 2.1 Allocation of registers and definition of common macros

Boxes,...

```
\newbox\g@trash
\newbox\g@matrixbox
```

```
\newbox\g@eastbox
\newbox\g@northbox
\newbox\g@label
\newbox\g@b@tmp
\newbox\g@b@tmpa
\newbox\g@b@tmpb
```

. . . counters,. . .

```
\newcount\g@maxrow
\newcount\g@maxcol
\newcount\g@maxrow@old
\newcount\g@maxcol@old
\newcount\g@c@tmp
\newcount\g@c@tmpa
```

. . . and dimensions . . .

```
\newdimen\g@axisHeight
\newdimen\g@linethickness
\newdimen\g@tab
\newdimen\g@arrowht
\newdimen\g@arrowwd
\newdimen\g@d@tmp
\newdimen\g@d@tmpa
\newdimen\g@d@tmpb
\newdimen\g@d@tmpc
\newdimen\g@d@tmpd
\newdimen\g@d@tmpe
```

are allocated.

`\g@for` For frequent use, we define a special loop mechanism, which allowes to iterate over a given interval from a lower bound to a higher one, or reversely. The code to execute is given as the third argument of `\g@for`, using `#1` for the iteration variable that is substituted by `\the\loopCount` for each value in the given bounds.

Each of the bounds is also visited. Example: The following code prints out the numbers from 1 to 37, inclusively:

```
\g@for1\to37\do{#1 }
```

We first need some more control sequences: `\g@loopContent` is defined to the loop's body when the loop is entered. `\g@loopCount` is the variable to increment or decrement with each iteration. `\g@loopEnd` marks the value at which to stop the loop, and `\g@loopStep` contains the direction, i.e. `\g@loopStep` $= -1$ iff `\g@loopEnd` $< \langle \textit{start value} \rangle$.

```
\def\g@loopContent#1{}
```

```
\newcount\g@loopCount\g@loopCount=0
\newcount\g@loopEnd\g@loopEnd=1
\newcount\g@loopStep\g@loopStep=1
```

The `\g@loop` command executes the loop initialized by `\g@for`. Each iteration is executed in its own group to avoid side effects and expecially to provide nested loops.

```
\def\g@loop{%
 % base case?
 \ifnum\g@loopCount=\g@loopEnd\else
  % no: execute loop body
  {\expandafter\g@loopContent\expandafter{\the\g@loopCount}}%
  % increment or decrement the loop variable
  \advance\g@loopCount\g@loopStep
  % call \g@loop recursivly.
  \g@loop
 \fi
}
```

Finally, here is the definition of `\g@for`. Each value in the interval from `#1` to `#2`, including `#1` and `#2` is visited exactly one time.

```
\def\g@for#1\to#2\do#3{%
 \def\g@loopContent##1{#3}%
 \g@loopCount=#1
 \g@loopEnd=#2
 \ifnum\g@loopEnd>\g@loopCount%
  \g@loopStep=1
  \else\g@loopStep=-1
 \fi
 \advance\g@loopEnd\g@loopStep % inclusive upper bound
 \g@loop
}
```

`\g@checkBounds` The next tool is used by the generic operation commands to check whether or not a given index is valid. If `#2` $\leq$ `#3` $\leq$ `#4` does not hold, a package error is thrown that tells the user what happened.

Parameter `#1` contains 'r' or 'c' to denote "rows" or "columns", respectively. This piece of information is only used within the construction of the error message.

`\ifg@indexCorrect` The result of `\g@checkBounds` is returned via `\ifg@indexCorrect`.

```
\newif\ifg@indexCorrect
\def\g@checkBounds#1#2#3#4{%
 \g@indexCorrectfalse
 \ifnum#2>#3%
```

```
  \PackageError{gauss}{\g@shorterror{#1} #3<#2}{\g@longerror{#1}}
 \else
  \ifnum#3>#4%
   \PackageError{gauss}{\g@shorterror{#1} #3>#4}{\g@longerror{#1}}
  \else
   \g@indexCorrecttrue
  \fi
 \fi
}
```

We skip the definitions of `\g@shorterror` and `\g@longerror` which serve to produce error messages.

`\g@downarrow` For drawing horizontal arrows of arbitrary length, we use the construction

`\hbox to`⟨*width*⟩`{$\leftarrowfill$}`

which uses Plain-TEX's `\leftarrowfill`. Unfortunately, there is no vertical correspondence to that mechanism and thus, we have construct something like this by ourselves. We will do so by reimplementing a mechanism that is used by `\left` and `\right` to construct delimiters of arbitrary height.

```
\DeclareMathSymbol{\g@downarrowSymb}{\mathord}{largesymbols}{'\y}
\DeclareMathSymbol{\g@vertlineSymb}{\mathord}{largesymbols}{'\?}
\def\g@vertline{\hbox{$\g@vertlineSymb$}\kern-\lineskip}%
```

After allocating the basic symbols, we define `\g@downarrow` by a recursion which fills up a vbox with the necessary number of `\g@vertline`'s and a final `\g@downarrowSymb`.

The resulting vbox has exactly the height given in `#1` (as TEX-length), and no depth. If `#1` is less than a minimum height, then it is set to that minimum height.

```
\def\g@downarrow#1{\vbox{%
 \vfill
 \baselineskip\z@\relax
 \g@d@tmpc=#1\relax
 \ifdim \g@d@tmpc<\g@arrowht
  \g@d@tmpc\g@arrowht\relax
 \fi
 \g@vlineRec
 \kern\g@d@tmpc
 \setbox\g@trash=\hbox{$\g@downarrowSymb$}%
 \hbox{\raise\dp\g@trash\box\g@trash}%
}}
\def\g@vlineRec{%
 \advance\g@d@tmpc-\g@arrowht
\ifdim \g@d@tmpc<\z@ \else
 \g@vertline
 \g@vlineRec
\fi
}
```

## 2.2 Converting floasts and lengths to each other

\g@defdim
\g@defdouble
\g@dim
\g@double

The typesetting of matrix operations is done by use of the `picture` environment of LaTeX. The macros of that environment require plain numbers, possibly containing a decimal point. Though it is not clearly correct, we will call that data format *float* or *double*.

`picture`'s macros do not work if you give them dimensions as input. And since the results of measuring a matrix are necessarily dimensions, we need a mechanism to convert dimensions to floats and vice versa.

This mechanism is the topic of the current section.

In fact, we almost provide our own data structure whose values can be shown as TeX dimensions or as floats. You can "construct a new instance" of that structure either by a dimension (using `\g@defdim`) or by a double (using `\g@defdouble`). In both cases, a macro is defined to be the corresponding double value.

Given an instance of our data structure, i.e. given a double, you can get its double representation using `\g@double` (this just typesets the double representation), and you can store its value into a TeX dimension using `\g@dim`.

Macros for manipulation on floats are defined in the following section.

We first need a macro that cuts away the "pt". This is rather tricky because the "pt" that arises in the result of some `\the`⟨*counter*⟩ has not the catcodes as expected. We can redefine them temporarily but we have to note that constructions like `\g@defdim{`⟨*identifier*⟩`}{12pt}` (i.e. giving the length directly) are no longer possible, since the "pt" of a directly given length has the "normal" catcodes.

```
\edef\redo#1{\catcode`p=#1\catcode`t=#1\relax}
\redo{12}
\def\g@del#1pt{#1}
\redo{11}
```

Defining a float by a dimension. The first argument expects an idetifier (identifiers are arbitrary strings), and the second argument expects a TeX dimension *register*, i.e. some control sequence `\cs` that evaluates to "... pt" if you say `\the\cs`.

It is not possible to specify a double by directly give a length. Use `\g@defdouble` below in that case.

```
\def\g@defdim#1#2{%
  \edef\g@defdim@arg{\the #2}%
  \edef\g@defdim@arg{\expandafter\g@del\g@defdim@arg}%
  \g@defdouble{#1}{\g@defdim@arg}%
}
```

And here is `\g@defdouble`. `#1` should be an identifier and `#2` should be the value to store in float `#1`. To avoid naming conflics with other macros, `#2` is stored into a macro based on `g@@` and the content of `#1`.

```
\def\g@defdouble#1#2{%
  \expandafter\expandafter\expandafter\global
  \expandafter\edef\csname g@@#1 \endcsname{#2}%
}
```

We now come to the macros for "reading" a float. These are `\g@dim` (to read the dimensional representation) and `\g@double` (for the double representation).

An error will occur if you try to read the value of a float that was not previously defined. ("Missing number, treated as zero.")

First `\g@dim`: Let `#1` be the identifier and `#2` the TEX dimension registern to store the value of `#1` in.

```
\def\g@dim#1#2{%
  \edef\g@dim@arg{\g@double{#1}}%
  #2=\g@dim@arg\p@\relax
}
```

And `\g@double` is even simpler:

```
\def\g@double#1{%
  \csname g@@#1 \endcsname
}
```

## 2.3 Macros for calculus on floats

We need some macros that provide simple arithmetic calculation on floats. Those are defined now.

`\g@advance` Given a float $f_1$, the following macro performs $f_1 := f_1 + f_2$ where $f_2$ may be either a TEX dimension or a float: If `\csname`$f_2$`\encsname` does not evaluate to some control sequence, it is assumed to denote a TEX dimension (e.g. `5pt`, or `\the\something`)

```
\def\g@advance#1#2{%
  \g@dim{#1}{\g@d@tmpa}% f_1 := #1
\expandafter\ifx\csname g@@#2 \endcsname\relax
 \g@d@tmpb=#2% f_2 := #2 (TeX dimension)
\else
 \g@dim{#2}{\g@d@tmpb}% f_2 := #2 (float)
\fi
\advance\g@d@tmpa\g@d@tmpb\relax% f_1 += f_2
\g@defdim{#1}{\g@d@tmpa}% #1 := f_1
}
```

`\g@min` `\g@minD` Given two floats $f_1, f_2$ and a TeX dimension $d_3$, the following macro performs $d_3 := \min\{f_1, f_2\}$.

```
\def\g@min#1#2#3{%
 \g@dim{#1}{\g@d@tmpa}% f_1 := #1
 \g@dim{#2}{\g@d@tmpb}% f_2 := #2
 \ifdim \g@d@tmpa<\g@d@tmpb
   #3=\g@d@tmpa
 \else
   #3=\g@d@tmpb
 \fi
 \relax
}
```

There is a so called *D*-version of the latter macro. By use of `\g@min`, this macro also calculates $\min\{f_1, f_2\}$, but its result is translated into a double representation which is then stored in control sequence `#3`.

```
\def\g@minD#1#2#3{%
 \g@min{#1}{#2}{\g@d@tmpc}%
 \edef\g@minD@arg{\the\g@d@tmpc}%
 \edef\g@minD@arg{\expandafter\g@del\g@minD@arg}%
 \edef#3{\g@minD@arg}%
}
```

`\g@max` `\g@maxD` And here is are the two opposite macros of the preceeding two.

```
\def\g@max#1#2#3{%
 \g@dim{#1}{\g@d@tmpa}%
 \g@dim{#2}{\g@d@tmpb}%
 \ifdim \g@d@tmpa<\g@d@tmpb
   #3=\g@d@tmpb
 \else
   #3=\g@d@tmpa
 \fi
 \relax
}
\def\g@maxD#1#2#3{%
 \g@max{#1}{#2}{\g@d@tmpc}%
\edef\g@maxD@arg{\the\g@d@tmpc}%
\edef\g@maxD@arg{\expandafter\g@del\g@maxD@arg}%
\edef#3{\g@maxD@arg}%
}
```

\g@dist \g@distD Given two floats $f_1, f_2$ and a TEX dimension $d_3$, the following macro performs $d_3 := f_1 - f_2$.

```
\def\g@dist#1#2#3{%
\g@dim{#1}{\g@d@tmpa}% f_1 := #1
\g@dim{#2}{\g@d@tmpb}% f_2 := #2
\ifdim \g@d@tmpa<\g@d@tmpb
  #3=\g@d@tmpb
  \advance#3 by-\g@d@tmpa
\else
  #3=\g@d@tmpa
  \advance#3 by-\g@d@tmpb
\fi
\relax
}
```

Again, we have a $D$-version, where the result is given in double representation.

```
\def\g@distD#1#2#3{%
\g@dist{#1}{#2}{\g@d@tmpc}%
\edef\g@distD@arg{\the\g@d@tmpc}%
\edef\g@distD@arg{\expandafter\g@del\g@distD@arg}%
\edef#3{\g@distD@arg}%
}
```

\g@updateArea \g@update While the macros that we have seen in this section so far are mainly used for elementary drawing purposes, we now define a slightly more sophisticated macro. It is needed to update the leftmost $x$-values of the so-far matrix operation set (in terms of row operations). It is invoked after adding a new operation to the set.

To update a float $f_1$ with respect to $f_2$ is defined to execute $f_1 := \max\{f_1, f_2\}$. This is what the following macro does.

```
\def\g@update#1#2{%
\g@dim{#2}{\g@d@tmpe}
\g@dim{#1}{\g@d@tmpb}
\ifdim \g@d@tmpe>\g@d@tmpb
 \g@defdim{#1}{\g@d@tmpe}%
\fi
}
```

The matrix dimensions are stored in floats named ⟨*name*⟩ + ⟨*index*⟩ where ⟨*name*⟩ spcifies the dimension (e.g. “cy” for the current $y$ values of a *c*olumn) and ⟨*index*⟩ is the index of the row/column to which the float's value belongs.

Now, the following macro iterates over $i \in \{$`#3`$, \ldots,$ `#4`$\}$ and updates all the floats with name `#2` $+ i$ with respect to float `#1`.

```
\def\g@updateArea#1#2#3#4{\g@for#3\to#4\do{\g@update{#2##1}{#1}}}
```

## 2.4 Macros for measurements

The macros defined in this section are used to measure the dimensions of a given matrix and store the measured values into floats.

For each row $i$ of the matrix, the $y$-position of the center of row $i$ with respect to the bottom of the matrix is stored in a float named `ry`$+i$. Another float `rx` $+ i$ is initialized to 0. This latter value will always contain the leftmost position at which a new row operation can start without intersecting previous operations that crossed row $i$.

For each row $j$ of the matrix we similarly define the values `cx` $+ i$ and `cy` $+ i$. Note that `cx` $+ i$ corresponds to `ry` $+ i$ and `cy` $+ i$ corresponds to `rx`$+i$, since column operations grow vertically whereas row operations grow horizontally.

`\g@measureRows` We first consider row measuring. The following macro assumes that the current box is a `\vbox` that only contains a copy of the matrix, i.e. one `\hbox` per row including all the intermediate glues and kerns and whatever. You can initialize what we assume to have by saying

`\vbox{\halign{`...`}}` (typeset the matrix)
`\box0=\lastbox` (save the matrix)
`\vbox{\unhcopy0\g@measureRows}` (measure the row's heights)
`\box0` (restore the matrix)

Caution: The following macros will not work if the matrix was not constructed with an `\halign` because special knowledge about the structure of `\halign`'s result is used.

It is assumed that `\g@d@tmp` initially contains the $y$-position of the matrix's bottom. It is further assumed that `\g@maxrow` contains the total number of rows. These two counters will be modified during the recursion.

```
\def\g@measureRows{%
\setbox\g@trash\lastbox
\ifnum\g@maxrow<0% base case: this box is not part of the matrix
\else
```

```
  \ifdim\ht\g@trash=0pt%
   \advance\g@d@tmp\lastskip\unskip
   \advance\g@d@tmp\lastkern\unkern
   \unpenalty
  \else
   \advance\g@d@tmp\dp\g@trash
   \advance\g@d@tmp\g@axisHeight
   \g@defdim{ry\the\g@maxrow}{\g@d@tmp}%
   \g@defdim{rx\the\g@maxrow}{\z@}%
   \advance\g@d@tmp-\g@axisHeight
   \advance\g@d@tmp\ht\g@trash
   \advance\g@maxrow-1%
  \fi
  \g@measureRows
 \fi
}
```

`\g@measureCols` In fact, the row measurement is the easier case. The measurement of column widths is more complicated by two reasons: 1. The number of columns is unknown, and 2. we will meet the cells in reverse order.

This is why column measurement is implemented in two main steps. First the width of each cell and the distance between two preceeding cells is measured and stored into temporary floats `ct` + ⟨*index*⟩ (distance) and `cy` + ⟨*index*⟩ (width), where ⟨*index*⟩ is counted from back to front. By the way, we count the number of columns.

In the base case of the recursion we start a second recursion that will calculate the final results out of the intermediate results and that will arange the indexing properly.

What input do we expect? It is assumed that the current box is an `\hbox` whose first item is an `\hbox` of width 100cm (to detect the base case), followed by a copy of the last row of the matrix to measure. See the definition of `g@matrix` to see how such a situation can be constructed.

We further assume that `g@d@tmp` is initialized to 0.0pt.

```
\def\g@measureCols{%
 \setbox\g@trash\lastbox
 \ifdim \wd\g@trash=100cm%
  % base case. Invert the ordering and sum the dimensions.
  \g@defdouble{ct\the\g@maxcol}{0}%
  \g@defdouble{cy\the\g@maxcol}{0}%
  \global\g@maxcol\g@maxcol
  \g@c@tmp\g@maxcol
  \advance\g@c@tmp-1%
  \g@measureColsSucc
  \global\advance\g@maxcol-1%
 \else
  \ifdim \ht\g@trash=0pt%
   \advance\g@d@tmp\lastskip\unskip
   \advance\g@d@tmp\lastkern\unkern
   \unpenalty
  \else
   % use ct temporaryly to store the skip between
   % colnr + 1 and colnr.
   \g@defdim{ct\the\g@maxcol}{\g@d@tmp}%
   \g@d@tmp\z@
   % use cy temporaryly to store the cell's width.
   \g@defdim{cy\the\g@maxcol}{\wd\g@trash}%
   \advance\g@maxcol1%
  \fi
  \g@measureCols
 \fi
}
```

Now, the macro for the second step of the measurement algorithm is defined. This is easier, since we now already know the total number of columns that have been measured. Roughly speaking, we sum their width's from left to right to obtain the $x$-values of the horizontal center of each column. Furthermore, the $y$-values are now initialized to 0, and the order is inverted.

Knowledge about the implementation of `g@matrix` is used!

```
\def\g@measureColsSucc{%
 \ifnum \g@c@tmp<0\else
  \g@c@tmpa=\g@maxcol
  \advance\g@c@tmpa-\g@c@tmp
  \advance\g@c@tmpa-1
  \g@dim{cy\the\g@c@tmp}{\g@d@tmpa}% width of this cell
  \g@dim{ct\the\g@c@tmp}{\g@d@tmpb}% glue right to this cell
  \advance\g@d@tmp.5\g@d@tmpa%
  \g@defdouble{cy\the\g@c@tmp}{0}%
  \g@defdim{cx\the\g@c@tmpa}{\g@d@tmp}%
  \advance\g@d@tmp.5\g@d@tmpa
  \advance\g@d@tmp\g@d@tmpb
  \ifnum \g@c@tmpa=0%
   \advance\g@d@tmp.5\g@tab
  \fi
  \advance\g@c@tmp-1
  \g@measureColsSucc
 \fi
}
```

`\g@measureAxis` This is an easier macro. It measures and defines some common lengths, e.g. the distance between bottom line and math axis, and the dimensions of math arrows which are used for drawing arrows in operations.

```
\def\g@measureAxis{%
 % 1. Where is the math axis relative to the ground line?
 \setbox\g@trash=\hbox{$\vcenter{\hbox to 5pt{}}$}%
 \global\g@axisHeight=\ht\g@trash
 % 2. What is the minimum width of a horizontal arrow?
 \setbox\g@trash=\hbox{$\leftarrow$}%
 \global\g@arrowwd\wd\g@trash
 % 3. What is the minimum height of a vertical arrow?
 \setbox\g@trash=\vbox{\g@vertline}
 \global\g@arrowht=\ht\g@trash
 \global\advance\g@arrowht\dp\g@trash
 \global\advance\g@arrowht\lineskip
 % 4. What should be the thickness of ordinary lines?
 \global\g@linethickness=\fboxrule\relax
}
```

`\g@measureArea` The last marco of this subsection provides the measurement of a set of floats. (Therefore, it is rather a calculus macro.)

Assuming that `#4` is a float identifier and for all $i \in I := \{\texttt{\#2}, \ldots, \texttt{\#3}\}$ `#1` $+\, i$ is a float identifier, the macro does

$$\texttt{\#4} := \max_{i \in I}\{\texttt{\#1} + i\}$$

```
\def\g@measureArea#1#2#3#4{%
 \g@defdim{#4}{\z@}%
 \g@for#2\to#3\do{%
  \g@max{#1##1}{#4}{\g@d@tmpe}%
  \g@defdim{#4}{\g@d@tmpe}%
 }%
}
```

## 2.5 Macros for drawing purposes

This Section defines low level macros for drawing purposes within a `picture` environment by use of floats.

`\g@vline` Let $f_1, f_2$ and $f_3$ be floats. Then,

`\g@vline{`$f_1$`}{`$f_2$`}{`$f_3$`}`

draws a vertical line from $(f_1, f_2)$ to $(f_2, f_3)$.

```
\def\g@vline#1#2#3{%
  \g@minD{#2}{#3}{\min}
  \g@distD{#2}{#3}{\dist}
  \put(\g@double{#1},\min){\line(0,1){\dist}}
}
```

`\g@vvline` Let $f_1, f_2$ and $f_3$ be floats. Then,

`\g@vvline{`$f_1$`}{`$f_2$`}{`$f_3$`}`

draws a vertical line of length `f_3`, starting at $(f_1, f_2)$, i.e. a line from $(f_1, f_2)$ to $(f_1, f_2 + f_3)$.

```
\def\g@vvline#1#2#3{%
  \put(\g@double{#1},\g@double{#2}){\line(0,1){\g@double{#3}}}
}
```

`\g@varrow` Let $f_1, f_2$ and $f_3$ be floats. Then,

`\g@varrow{`$f_1$`}{`$f_2$`}{`$f_3$`}`

draws an arrow from $(f_1, \max\{f_2, f_3\})$ to $(f_1, \min\{f_2, f_3\})$.

```
\def\g@varrow#1#2#3{%
  \g@dim{#2}{\g@d@tmpa}%
  \g@dim{#3}{\g@d@tmpb}%
  \advance\g@d@tmpb-\g@d@tmpa
  \g@cbox{#1}{#2}{\g@downarrow{\g@d@tmpb}}%
}
```

`\g@hline` Let $f_1, f_2$ and $f_3$ be floats. Then,

`\g@hline{`$f_1$`}{`$f_2$`}{`$f_3$`}`

draws a horizontal line from $(f_1, f_2)$ to $(f_3, f_2)$.

```
\def\g@hline#1#2#3{%
  \g@minD{#1}{#3}{\min}%
  \g@distD{#1}{#3}{\dist}%
  \put(\min,\g@double{#2}){\line(1,0){\dist}}%
}
```

`\g@hhline` Let $f_1, f_2$ and $f_3$ be floats. Then,

`\g@hhline{`$f_1$`}{`$f_2$`}{`$f_3$`}`

draws a horizontal line of length `f_3`, starting at $(f_1, f_2)$, i.e. a line from $(f_1, f_2)$ to $(f_1 + f_3, f_2)$.

```
\def\g@hhline#1#2#3{%
 \put(\g@double{#1},\g@double{#2}){\line(1,0){\g@double{#3}}}%
}
```

`\g@harrow` Let $f_1, f_2$ and $f_3$ be floats. Then,

`\g@harrow{`$f_1$`}{`$f_2$`}{`$f_3$`}`

draws an arrow from $(\max\{f_1, f_3\}, f_2)$ to $(\min\{f_1, f_3\}, f_2)$.

```
\def\g@harrow#1#2#3{%
 \g@dim{#1}{\g@d@tmpa}%
 \g@dim{#3}{\g@d@tmpb}%
 \advance\g@d@tmpb-\g@d@tmpa
 \advance\g@d@tmpb2\p@
 \g@rbox{#1}{#2}{\hbox to\g@d@tmpb{\leftarrowfill}}%
}
```

The remaining two macros allow to put arbitrary math material to a specified position. Those are used for typesetting so called labels within matrix operations, for example, the factor at an `\add` arrow.

`\g@rbox` The first one is intended to use for row operations. Assuming that `#1`, `#2` are float identifiers and `#3` is math material, we put `#3` into an `\hbox` and put that box to point (`#1`, `#2`).

The box will be vertically aligned to `#2` (i.e., the math axis of `#3` will be at height `#2`), and horizontally start at `#1`.

The macro puts the math material of `#3` into `\g@label` and just copies its content when using, so you can reuse `\g@label` (e.g. for measuring purposes).

```
\def\g@rbox#1#2#3{%
 \setbox\g@label=\hbox{$\relax#3\relax$}%
 \ht\g@label\z@\dp\g@label\z@
 \setbox\g@label=\hbox{$\mathstrut\box\g@label$}%
 \put(\g@double{#1},\g@double{#2})%
 {\makebox(0,0)[l]{\kern-\p@\copy\g@label}}%
}
```

`\g@cbox` The last macro of this section does the corresponding job for columns.

Here, `#3` will be centered horizontally to `#1`, whereas `#2` denotes the height of the label's bottom.

Again, you can reuse the constructed box, it remains in register `\g@label`.

```
\def\g@cbox#1#2#3{%
 \setbox\g@label=\hbox{$\relax#3\relax$}%
 \setbox\g@label=\hbox{\raise\dp\g@label\box\g@label}%
 \put(\g@double{#1},\g@double{#2})%
  {\makebox(0,0)[b]{\copy\g@label}}%
}
```

## 2.6 Generic operation commands

Before `\halign` begins, the matrix construction macro defines, what to do if the matrix is finished. This is defined in `\g@endregion` (see the next section for further information).

The `\rowops` and `\colops` commands are temporarily set to `\g@east` or `\g@north`, respectively. Thus, when entering an operation part, the first thing to do is to invoke `\g@endregion` to do the things that have to be done when the matrix input finishes. After that, `\g@endregion` has to be redefined to avoid doing the same process two times. Fortunately, `\g@north` and `\g@east` can easily reuse `\g@endregion` and store there those things that have to be done at the end of a region.

Hence, each switching to another part of the matrix input consists of three parts:

1. Invoke `\g@endregion` to finish the current input part.
2. Redefine `\g@endregion` to do the stuff that has to be done at the end of the region that is now starting. The result of the region is stored into a special box register which is used in the `gmatrix` environment.
3. Initialize the new region.

You can imagine that it is easy to define further regions (e.g. for operations to the right or below the matrix).

The two predefined regions `\rowops` and `\colops` are very similar, so we will show just one of them in this documentation.

`\g@north` The `\g@north` macro is the generic version of `\colops`, its corresponding part is `\g@east`.

```
\def\g@north{%
```

1. Finish the current region

```
 \g@endregion
```

2. Redefine `\g@endregion` and prevent `\colops` from being called again.

```
\gdef\colops{\PackageError{gauss}
  {Two sets of column operations are specified in %
   just one matrix. This is not allowed.}}%
\gdef\g@endregion{%
  \end{picture}\egroup
  \g@measureArea{cy}{0}{\the\g@maxcol}{sum}%
  \g@dim{sum}{\ht\g@northbox}%
  \global\setbox\g@northbox=\hbox{%
   \raise\colarrowsep\box\g@northbox}%
}%
```

3. Initialization of the `\colops` region: Define the operation macros to be the corresponding private versions of this region (see below), set `sum` := 0 and start the `picture` environment where the operations are painted in.

```
\def\swap{\g@north@arrow11\colswapfromlabel\colswaptolabel}%
\def\add{\g@north@arrow01\coladdfromlabel\coladdtolabel}%
\let\mult\g@north@mult
\g@defdim{sum}{\z@}%
\global\setbox\g@northbox=\hbox\bgroup
  \begin{picture}(\g@double{w},0)(0,0)
    \linethickness{\g@linethickness}%
}
```

`\g@north@mult` The multiplication macro is the simplest one because it affects only one single column.

```
\def\g@north@mult#1#2{%
 \ifx *#1
```

Reduce `*` to a set of numbers.

```
   \g@for0\to\g@maxcol\do{\g@north@mult{##1}{#2}}%
 \else
```

Now `#1` is a number. Is it an index?

```
   \g@checkBounds{c}{0}{#1}{\the\g@maxcol}%
   \ifg@indexCorrect
```

Yes, it is. Typeset the operation.

```
    \g@cbox{cx#1}{cy#1}{\colmultlabel{#2}}%
    \g@dim{cy#1}{\g@d@tmpc}%
    \advance\g@d@tmpc\ht\g@label
    \g@defdim{cy#1}{\g@d@tmpc}%
    \g@advance{cy#1}{\the\opskip}%
    \g@update{sum}{cx#1}%
   \fi
 \fi
```

```
}
```

`\g@north@arrow` The `\g@north@arrow` macro is a generalisation of `\swap` and `\add`. It takes the following eight parameters:

- `#1`: 0 to make the 'from' line non-arrowed, 1 to get an arrow tip
- `#2`: 0 to make the 'to' line non-arrowed, 1 to get an arrow tip
- `#3`: macro for 'from' label rendering
- `#4`: macro for 'to' label rendering
- `#5`: [opt] label of the 'from' row
- `#6`: [opt] label of the 'to' row
- `#7`: index of the 'from' row
- `#8`: index of the 'to' row

If only one of the two optional arguments is given, then it is taken as `#5` and `#6` is taken empty. If both are missing, both are taken empty.

In `\g@north` above, `\add` is defined to

`\g@north@arrow01\coladdfromlabel\coladdtolabel`

and `\swap` is defined as

`\g@north@arrow11\colswapfromlabel\colswaptolabel`

```
\def\g@north@arrow#1#2#3#4{%
 \@ifnextchar[%
 {\g@north@arrow@a{#1}{#2}{#3}{#4}}%
 {\g@north@arrow@b{#1}{#2}{#3}{#4}{}[]}%
}
\def\g@north@arrow@a#1#2#3#4[#5]{%
 \@ifnextchar[%
 {\g@north@arrow@b{#1}{#2}{#3}{#4}{#5}}%
 {\g@north@arrow@b{#1}{#2}{#3}{#4}{#5}[]}%
}
\def\g@north@arrow@b#1#2#3#4#5[#6]#7#8{%
 \ifx *#7
```

Reduce star indices to loops of number indices. `**` needs a special handling.

```
  \ifx *#8
   \g@for0\to\g@maxcol\do{%
    \g@north@arrow@b{#1}{#2}{#3}{#4}{#5}[#6]{##1}{*}}%
  \else
```

Two loops rather than one because going from #8 down to 0 looks better than going from 0 to #8

```
   \g@for#8\to0\do{%
    \g@north@arrow@b{#1}{#2}{#3}{#4}{#5}[#6]{##1}{#8}}%
   \g@for#8\to\g@maxcol\do{%
    \g@north@arrow@b{#1}{#2}{#3}{#4}{#5}[#6]{##1}{#8}}%
  \fi
\else
  \ifx *#8
```

Reduce star indices to loops of number indices.

```
   \g@for#7\to0\do{%
    \g@north@arrow@b{#1}{#2}{#3}{#4}{#5}[#6]{#7}{##1}}%
   \g@for#7\to\g@maxcol\do{%
    \g@north@arrow@b{#1}{#2}{#3}{#4}{#5}[#6]{#7}{##1}}%
  \else
```

Now, #7 and #8 are numbers.

```
   \ifnum #7=#8\else
    \g@checkBounds{c}{0}{#7}{\the\g@maxcol}%
    \ifg@indexCorrect
     \g@checkBounds{c}{0}{#8}{\the\g@maxcol}%
     \ifg@indexCorrect
```

Now, #7 and #8 are different from each other, and both of them are legal indices. Store the current hights of the operations tower above column #7 and #8 into tmp1 and tmp2, respectively.

```
      \g@defdouble{tmp1}{\g@double{cy#7}}%
      \g@defdouble{tmp2}{\g@double{cy#8}}%
```

Find out the height of the horizontal connection line. First increment #7 and #8 by the minimum amounts.

```
      \ifx0#1
       \g@advance{cy#7}{\the\colopminsize}%
      \else
       \g@advance{cy#7}{\the\g@arrowht}%
      \fi
      \ifx0#2
       \g@advance{cy#8}{\the\colopminsize}%
      \else
       \g@advance{cy#8}{\the\g@arrowht}%
      \fi
```

Incorporate the columns between #7 and #8 into the height. Then sum denotes the level of the horizontal line.

```
      \g@measureArea{cy}{#7}{#8}{sum}%
```

Draw arrows and/or vertical lines from #7's and #8's height up to sum.

```
\ifx0#1
 \g@vline{cx#7}{tmp1}{sum}%
\else
 \g@varrow{cx#7}{tmp1}{sum}%
\fi
\ifx0#2
 \g@vline{cx#8}{tmp2}{sum}%
\else
 \g@varrow{cx#8}{tmp2}{sum}%
\fi
```

Draw the horizontal line.

```
\g@hline{cx#7}{sum}{cx#8}%
```

Insert space between the horizontal line and the labels if at least one of the labels #5 and #6 is not empty. Typeset the labels into boxes and measure them.

```
\setbox\g@b@tmpa=\hbox{$#3{#5}$}%
\setbox\g@b@tmpb=\hbox{$#4{#6}$}%
\ifdim\ht\g@b@tmpa>\z@
 \g@advance{sum}{\the\labelskip}%
\else
 \ifdim\ht\g@b@tmpb>\z@
  \g@advance{sum}{\the\labelskip}%
 \fi
\fi
```

Draw the 'from' label if there is one

```
\g@d@tmpc\z@
\ifdim\ht\g@b@tmpa>\z@
 \g@cbox{cx#7}{sum}{\kern-\p@\vcenter{\box\g@b@tmpa}}%
 \g@d@tmpc=\ht\g@label
\fi
```

Draw the 'to' label if there is one

```
\ifdim\ht\g@b@tmpb>\z@
 \g@cbox{cx#8}{sum}{\kern-\p@\vcenter{\box\g@b@tmpb}}%
 \ifdim \ht\g@label>\g@d@tmpc
  \g@d@tmpc=\ht\g@label
 \fi
\fi
```

Advance the sum by the maximum height of the two labels and the desired space between two consecutive operations

```
\g@advance{sum}{\the\g@d@tmpc}%
\g@advance{sum}{\the\opskip}%
```

Update all column tower heights between #7 and #8 to sum.

```
      \g@updateArea{sum}{cy}{#7}{#8}%
```

That's it.

```
     \fi
    \fi
   \fi
  \fi
 \fi
}
```

\g@east
\g@east@mult

The corresponding eastern macros are very similar to the above defined northern versions. Maybe there is a way to define generic operation commands once for all regions, but this would at least lead to less comprehesive definitions.

We skip the definitions of \g@east, \g@east@mult and \g@east@arrow in this documentation.

## 2.7 The gmatrix environment

gmatrix calls #1matrix where matrix is redefined to g@matrix. g@matrix typesets the matrix using \halign and stores the operations into box registers \g@northbox and \g@eastbox, respectively. The matrix itself is stored into \g@matrixbox.

The "real" typesetting is done at the end of gmatrix.

gmatrix

. . . and here is gmatrix:

```
\newenvironment{gmatrix}[1][]
{\unitlength=1pt\def\g@environment{#1matrix}%
 \begin{g@matrix}%
}{%
 \end{g@matrix}%
```

Delete temporarily the definition of matrix.

```
 \let\matrix\@empty
 \let\endmatrix\@empty
```

Find out sizes of the matrix. Set \g@d@tmp to the height of the matrix.

```
 \g@d@tmpa\ht\g@matrixbox \advance\g@d@tmpa\p@
 \g@d@tmpb\dp\g@matrixbox \advance\g@d@tmpb\p@
 \g@d@tmp\ht\g@northbox \ht\g@northbox\z@
 \dp\g@northbox\z@
 \ifdim \g@d@tmp>\z@
  \advance\g@d@tmp-\opskip
 \fi
 \advance\g@d@tmp.5\ht\g@matrixbox
 \advance\g@d@tmp.5\dp\g@matrixbox
```

Start the matrix environment to get the left delimiter.

```
 \begin{\g@environment}%
```

Typeset the column operations to the north of the matrix, and the matrix itself.

```
  \vcenter{\hbox{%
   \rlap{\raise\ht\g@matrixbox\box\g@northbox}% north
   % 1 additional pt above and below the matrix
   \rule\z@\g@d@tmpa\lower\g@d@tmpb\null
   \box\g@matrixbox% the matrix itself
  }}%
```

Close the matrix environment to get now the right delimiter.

```
 \end{\g@environment}%
```

Finally typeset the eastern operations. Insert vertical space of `\g@d@tmp` (the height of the matrix) and horizontal space of `\rowarrowsep` before.

```
 \rule\rowarrowsep\z@
 \rule\z@\g@d@tmp
 \g@dim{d}{\g@d@tmpa}%
 \vcenter{\hbox{\lower\g@d@tmpa\box\g@eastbox}}%
}
```

Here is the definition of `\g@endmatrix`. This is the initial `\g@endregion` which is defined within `\begin{gmatrix}` to finish the matrix input.

```
\def\g@endmatrix{%
  \mathstrut\crcr
 \egroup % end of \halign
 \egroup % end of \vbox, this contains the matrix
```

Save the matrix into matrixbox.

```
 \global\setbox\g@matrixbox\lastbox
```

Measure the matrix' dimensions.

```
 \g@measureAxis
 \setbox\g@trash=\vbox{%
  \unvcopy\g@matrixbox
```

Copy the last row of the matrix into `\g@eastbox` and reinsert it to the vbox.

```
  \global\setbox\g@eastbox=\lastbox
  \copy\g@eastbox
  \g@d@tmp\z@ {\g@measureRows}% measure rows
 }%
 \setbox\g@trash=\hbox{%
```

Insert a box of width 100cm to recognize the beginning of the hbox within the measurement recursion.

```
 \hbox to 100cm{.\hfill.}%
 \unhbox\g@eastbox
 \g@d@tmp\z@ {\g@measureCols}% measure columns
}%
```

Determine global dimensions of the matrix (total height, etc.).

```
\g@d@tmpa=\ht\g@matrixbox\advance\g@d@tmpa\dp\g@matrixbox
\g@defdim{h}{\g@d@tmpa}%
\g@defdim{w}{\wd\g@matrixbox}%
\g@defdim{d}{\dp\g@matrixbox}%
}%
```

g@matrix Finally, we have the following definition of g@matrix:

```
\edef\g@prae{\hfil$\relax\noexpand\mathstrut}
\edef\g@post{\relax$\hfil}
\newenvironment{g@matrix}
{\setbox\g@trash=\hbox\bgroup
  \global\g@maxrow@old\g@maxrow
  \global\g@maxcol@old\g@maxcol
  \global\g@maxrow0%
  \global\g@maxcol0%
  \let\rowops\g@east
  \let\colops\g@north
  \vbox\bgroup
   % count rows while typesetting
   \def\\{\mathstrut\cr\global\advance\g@maxrow1\relax}%
   \global\let\g@endregion\g@endmatrix
   \global\g@tab=2\arraycolsep
   \ialign\bgroup\g@prae##\g@post&&\kern\g@tab\g@prae##\g@post\cr
}{%
  \g@endregion
 \egroup % end of \hbox
 % enable nested gmatrixes (for DniQ :-)
 \global\g@maxrow\g@maxrow@old
 \global\g@maxcol\g@maxcol@old
 \global\let\g@endregion\g@endmatrix
 \global\let\rowops\g@east
 \global\let\colops\g@north
}
```

## 2.8 Public tools

`\newmatrix`

The `\newmatrix` command allows to define new matrix environments with special delimiters as described in Section 1.

```
\def\newmatrix#1#2#3{%
 \ifx g#3 \else
  \ifx {#3}{g@} \else
   \expandafter\ifx\csname#3matrix\endcsname\relax
    \newenvironment{#3matrix}%
     {\left#1\begin{matrix}}{\end{matrix}\right#2}%
   \else
    \renewenvironment{#3matrix}%
     {\left#1\begin{matrix}}{\end{matrix}\right#2}%
   \fi
  \fi
 \fi
}
```

For compatibility reasons, we redefine predefined matrix environments such as `pmatrix`. This is necessary to avoid problems that arise when dealing with earlier $\mathcal{AMS}$TEX versions.

```
\newmatrix()p
\newmatrix[]b
\newmatrix\lbrace\rbrace B
\newmatrix\lvert\rvert v
\newmatrix\lVert\rVert V
```

`\rowmultlabel` `\colmultlabel` `\rowaddfromlabel` `\coladdfromlabel` `\rowaddtolabel` `\coladdtolabel` `\rowswapfromlabel` `\colswapfromlabel` `\rowswaptolabel` `\colswaptolabel`

Labels of operations are typeset using the so-called fontifying macros described in Section 1.3. All of them take exaclty one argument, and they are called within math mode. The user may redefine them to adjust the appearence of operations according to his needs. The following is the standard definition:

```
\def\rowmultlabel#1{|\,#1}
\def\rowswapfromlabel#1{}
\def\rowswaptolabel#1{}
\def\rowaddfromlabel#1{\scriptstyle #1}
\def\rowaddtolabel#1{\scriptscriptstyle +}
\def\colmultlabel#1{%
 \underline{\hbox to 1.2em{$\hss\mathstrut{}#1\hss$}}%
}
\def\colswapfromlabel#1{}
\def\colswaptolabel#1{}
\def\coladdfromlabel#1{\scriptstyle #1}
\def\coladdtolabel#1{\scriptscriptstyle +}
```

`\colarrowsep` `\rowarrowsep` `\labelskip` `\opskip` `\colopminsize` `\rowopminsize`

Finally, we define the public lengths of Section 1.3:

```
\newdimen\colarrowsep\colarrowsep=.5em
\newdimen\rowarrowsep\rowarrowsep=.5em
\newdimen\opskip\opskip=5pt
\newdimen\labelskip\labelskip=4pt
\newdimen\colopminsize\colopminsize=3pt
\newdimen\rowopminsize\rowopminsize=3pt
```

And that's all.

```
\makeatother
```