PGFMATH–xfp

define pgfmath functions using xfp

Jonathan P. Spratte[*]

2025-01-11 v1.0a

Abstract

PGFMATH–**xfp** provides a small wrapper to define pgfmath functions which use the floating point unit of expl3 (of which the document-level interface is called xfp).

Contents

1 Documentation

This package serves as a stopgap to allow the usage of xfp in pgfmath functions. It is only meant as a temporary fix to allow single functions using the expl3 fpu until a more sophisticated solution to allow broader support for it in PGF is available.

The defined functions should work correctly independent of the surrounding pgfmath context. This is achieved by first parsing the arguments via `\pgfmathsetmacro` with PGF settings applied locally to ensure that the resulting format is understandable by xfp's fpu.

Any function defined with PGFMATH–**xfp** will internally use the better precision and bigger value range of xfp for the individual steps of calculation. But the final result of the function will be given back to pgfmath and thus needs to fit into the surrounding

---

[*] jspratte@yahoo.de

pgfmath's number format (which depends on whether its fpu is installed or not). So it doesn't magically allow you to draw values bigger than `\maxdimen` for instance.

Though it has both pgfmath and xfp in its name, the package only loads pgfmath as a dependency, the access to xfp's fpu is done at the expl3 level.

It was created as a result of two questions on `https://tex.stackexchange.com`: expl3 cannot see declared functions and pgf: "Dimension too large" in a function which fits into a graph, /pgf/fpu=true does not help.

1.1 Document-Level Interface

`\pgfmxfpdeclarefunction` `\pgfmxfpdeclarefunction{⟨name⟩}{⟨arg-count⟩}[⟨process-args⟩]{⟨fp-expression⟩}`

Define a pgfmath function named ⟨*name*⟩ that takes ⟨*arg-count*⟩ arguments. The behaviour is different depending on whether the optional argument was used or not.

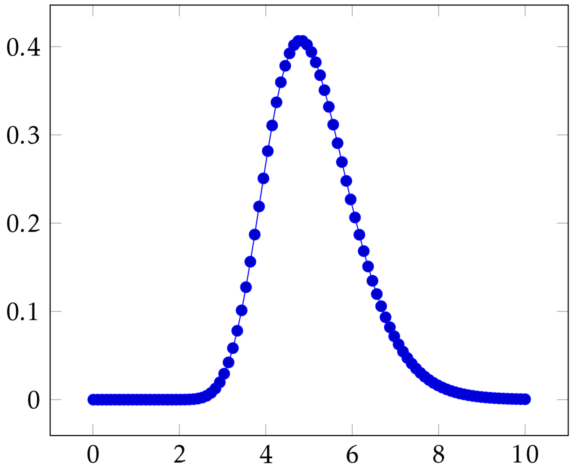
If it isn't the ⟨*fp-expression*⟩ can refer to the ⟨*arg-count*⟩ arguments using `#1`, *etc.*, and will get the arguments just like they are given to the function (translated to a format that xfp will understand by parsing them through pgfmath once).

If it is use ⟨*process-args*⟩ to specify any number of processed arguments in a comma separated list. Inside this list you can specify up to nine processed arguments using pgfmath functions in which you can refer to the arguments passed to your new function using `#1`, *etc.* You can refer to these processed arguments inside ⟨*fp-expression*⟩ using `#1`, *etc.* A result of this rule is that you have to explicitly use `#1` in ⟨*process-args*⟩ to forward it unaltered to the underlying xfp expression.

1.2 Examples

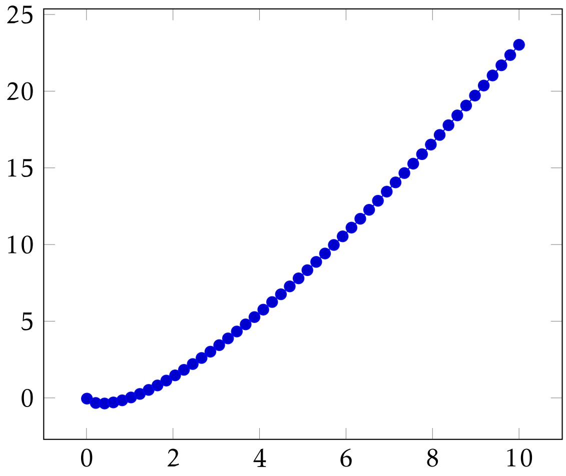
The following are examples taken from the two questions responsible for this package.

```
\pgfmxfpdeclarefunction{lognormal}{3}
  {exp(-((ln(#1) - #2)^2) / (2 * (#3)^2)) / (#1 * #3 * sqrt(2 * pi))}

\begin{tikzpicture}
\begin{axis}[ domain=0.01:10, samples=100 ]
  \addplot {lognormal(x,ln(5),0.2)};
\end{axis}
\end{tikzpicture}
```



Showing that a single pgfmath function argument can result in multiple arguments for the xfp expression. This example is suboptimal slow code, but could be educational.

```
\pgfmxfpdeclarefunction{fplog}{1}{ln(#1)}
\pgfmxfpdeclarefunction{nlogn}{1}[#1,fplog(#1)]{#1 * #2}

\begin{tikzpicture}
  \begin{axis}[ domain=0.01:10, samples=50 ]
    \addplot {nlogn(x)};
  \end{axis}
\end{tikzpicture}
```



1.3 expl3-Level Interface

`\pgfmxfp_declare:nnn` `\pgfmxfp_declare:nnn {⟨name⟩} {⟨arg-count⟩} {⟨fp-expression⟩}`

Defines a pgfmath function named ⟨*name*⟩, that takes {⟨*arg-count*⟩} arguments. The function will evaluate the ⟨`fp-expression`⟩ using the l3fp fpu and store the result for pgfmath. The arguments can be referred using `#1`, *etc*.

`\pgfmxfp_declare_processed_args:nnnn`

`\pgfmxfp_declare_processed_args:nnnn {⟨name⟩} {⟨arg-count⟩} {⟨processed-args⟩} {⟨fp-expression⟩}`

Defines a pgfmath function named ⟨`name`⟩, that takes {⟨*arg-count*⟩} arguments. The arguments will be evaluated through pgfmath according to the comma separated list of ⟨`processed-args`⟩ (in which you can refer to the arguments using `#1`, *etc.*) and the results of which will be the arguments for the ⟨`fp-expression`⟩ (in which you can refer to the ⟨`processed-args`⟩ using `#1`, *etc.*).

2 Implementation

2.1 General Package code

Some code for versioning support might not be available in older LaTeX 2$_\varepsilon$ releases.

```
\providecommand\DeclareRelease[3]{}
\providecommand\DeclareCurrentRelease[2]{}
```

Use these rollback functions to declare the current release.

```
\DeclareCurrentRelease{}{2025-01-11}
```

Make sure expl3 is available and load pgfmath and the PGF fpu.

```
\@ifundefined{ExplFileDate}
  {\RequirePackage{expl3}}
  {}
\RequirePackage{pgfmath}
\usepgflibrary{fpu}
```

`\pgfmxfpDate`
`\pgfmxfpVersion`

Store version and date in a macro.

```
\newcommand*\pgfmxfpDate{2025-01-11}
\newcommand*\pgfmxfpVersion{1.0a}
```

(*End of definition for* `\pgfmxfpDate` *and* `\pgfmxfpVersion`. *These functions are documented on page* **??**.)

Provide the package.

```
\ProvidesExplPackage
  {pgfmath-xfp}     {\pgfmxfpDate}
  {\pgfmxfpVersion} {define pgfmath functions using xfp}
```

2.2 Document-Level Interface

`\pgfmxfpdeclarefunction`

The document-level interface decides which of the two allocator functions are used.

```
\NewDocumentCommand \pgfmxfpdeclarefunction { m m o m }
  {
    \IfValueTF {#3}
      { \pgfmxfp_declare_processed_args:nnnn {#1} {#2} {#3} }
      { \pgfmxfp_declare:nnn {#1} {#2} }
      {#4}
  }
```

(*End of definition for* `\pgfmxfpdeclarefunction`. *This function is documented on page* 2.)

2.3 expl3-Level Interface

`\pgfmxfp_declare:nnn`

Start building the function body. First step is to initialize it with the common code. Then we add to the function body the input parsing step. For this we use a loop that will place `\pgfmathsetmacro` ⟨*tmp-cs$_i$*⟩ `{#`⟨*i*⟩`}` in the function body. Afterwards we do the real definitions. This strange construct is used to normalize the input. Depending on the context in which these functions are used, the arguments might be in the internal format of PGF's fpu-library or something else that l3fp will not understand. The `\pgfmathsetmacro` calls will be done in a local context in which PGF's fpu-library will be activated and set up to output in a format l3fp understands.

```
\cs_new_protected:Npn \pgfmxfp_declare:nnn #1#2#3
  {
    \@@_initialize_body:
    \int_step_inline:nn {#2}
      {
        \tl_put_right:Nx \l_@@_function_body_tl
          {
            \exp_not:n { \pgfmathsetmacro } \exp_not:c { @@_arg##1 }
              { \exp_not:n {####} ##1 }
          }
      }
    \@@_define_function:nnnn {#2} {#1} {#2} {#3}
  }
```

(*End of definition for* `\pgfmxfp_declare:nnn`*. This function is documented on page 3.*)

`\pgfmxfp_declare_processed_args:nnnn`

This works mostly like `\pgfmxfp_declare:nnn`, but instead of using an `\int_step_inline:nn`-loop this uses `\clist_map_inline:nn` to map over the processed arguments. Those will be stored in the function body as `\pgfmathsetmacro` $\langle tmp\text{-}cs_i\rangle$ {$\langle expr_i\rangle$}.

```
\cs_new_protected:Npn \pgfmxfp_declare_processed_args:nnnn #1#2#3#4
  {
    \@@_initialize_body:
    \int_zero:N \l_@@_args_int
    \clist_map_inline:nn {#3}
      {
        \int_incr:N \l_@@_args_int
        \tl_put_right:Nx \l_@@_function_body_tl
          {
            \exp_not:n { \pgfmathsetmacro }
              \exp_not:c { @@_arg \int_use:N \l_@@_args_int }
              { \exp_not:n {##1} }
          }
      }
    \exp_args:NV \@@_define_function:nnnn \l_@@_args_int {#1} {#2} {#4}
  }
```

(*End of definition for* `\pgfmxfp_declare_processed_args:nnnn`*. This function is documented on page 4.*)

2.4 Internals

`\l_@@_function_body_tl`

This token list will be used to build the function’s top-level definition.

```
\tl_new:N \l_@@_function_body_tl
```

(*End of definition for* `\l_@@_function_body_tl`*. This variable is documented on page* **??**.)

`\l_@@_args_int`

In the case of `\pgfmxfp_declare_processed_args:nnnn` we’ll have to count how many arguments the auxiliary function will take.

```
\int_new:N \l_@@_args_int
```

(*End of definition for* `\l_@@_args_int`*. This variable is documented on page* **??**.)

`\@@_initialize_body:`

Each function will have the same start setting up pgf’s fpu.

```
\cs_new_protected:Npn \@@_initialize_body:
  {
    \tl_set:Nn \l_@@_function_body_tl
      {
```

```
    \group_begin:
    \pgfkeys{/pgf/fpu=true, /pgf/fpu/output~format=sci}%
   }
 }
```

(*End of definition for* `\@@_initialize_body:`. *This function is documented on page* **??**.)

`\@@_define_function:nnnn`
`\@@_define_function_aux:n`

First we define the internal function. Then add to the function body some code that'll use `\use:x` to expand the temporary macros that store the input arguments and forward the results to the internal function.

```
\cs_new_protected:Npn \@@_define_function:nnnn #1#2#3#4
 {
  \@@_define_internal_function:nnn {#1} {#2} {#4}
  \tl_put_right:Nx \l_@@_function_body_tl
   {
    \use:x
     {
      \exp_not:c { @@_function_ #2 _cmd }
      \int_step_function:nN {#1} \@@_define_function_aux:n
     }
   }
  \exp_args:Nnno
  \pgfmathdeclarefunction {#2} {#3} \l_@@_function_body_tl
 }
```

The auxiliary is just used to build the temporary macro names and prevent them from further expansion.

```
\cs_new:Npn \@@_define_function_aux:n #1 { { \exp_not:c { @@_arg#1 } } }
```

(*End of definition for* `\@@_define_function:nnnn` *and* `\@@_define_function_aux:n`. *These functions are documented on page* **??**.)

`\@@_define_internal_function:nnn`
`\@@_define_internal_function_aux:n`

The internal function is pretty straight forward, the only difficult part is building the parameter list. For that we use some simple loop, a slow but simplistic solution.

```
\cs_new_protected:Npn \@@_define_internal_function:nnn #1#2#3
 {
  \exp_last_unbraced:Nx
  \cs_set_protected:cpn
   {
    { @@_function_ #2 _cmd }
    \int_step_function:nN {#1} \@@_define_internal_function_aux:n
   }
   { \group_end: \exp_args:Nf \pgfmathparse { \fp_eval:n {#3} } }
 }
\cs_new:Npn \@@_define_internal_function_aux:n #1 { \exp_not:n {## #1} }
```

(*End of definition for* `\@@_define_internal_function:nnn` *and* `\@@_define_internal_function_aux:n`. *These functions are documented on page* **??**.)

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.