The easing Library for PGF

Loh Ka-tsun

July 17, 2021

1 Introduction

This library adds easing functions to the PGF mathematical engine.

2 Installation

The easing library is a PGF library; it works both with LaTeX and with plain TeX. Once the file `pgflibraryeasing.code.tex` is in a directory searched by TeX, the library can be loaded as follows:

with plain TeX

```
\input pgf
\usepgflibrary{easing}
```

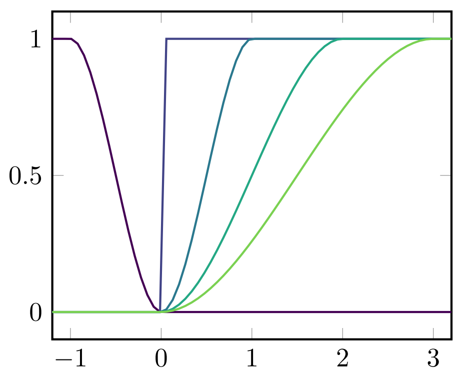
with LaTeX:

```
\usepackage{pgf}
\usepgflibrary{easing}
```

The easing library is compatible with, but does not depend on, the floating point unit library provided by PGF. To use both easing and the FPU, the FPU (or any packages/libraries which use the FPU, such as pgfplots) must be loaded before the easing library.

3 Usage

The routines implemented by the easing library are added to PGF's mathematical engine with `\pgfmathdeclarefunction`, so that they are recognised by by `\pgfmathparse` and can be used in any expression which is processed by the parser. As a first example, the following code produces plots of the function `smoothstep(`a`,`b`,`x`)` against the argument x, with one endpoint $a = 0$ and the other endpoint b ranging through the integers -1 to 3:




```
\input pgfplots
\usepgflibrary{easing}
\tikzpicture
\axis[
  domain=-1.2:3.2, samples=64,
  xmin=-1.2, xmax=3.2,
  cycle list={
    [samples of colormap=6 of viridis]},
  no marks, thick]
\pgfplotsinvokeforeach{-1,...,3}{
  \addplot{smoothstep(0,#1,x)};}
\endaxis
\endtikzpicture
\end
```

(This example also demonstrates the behaviour of the easing functions in some special cases: when the endpoints $b \leq a$, and in particular the degenerate case where $a = b$, in which the library chooses to consider the function that is 1 for all $x \geq 0$ and 0 otherwise.)

Like all functions declared in this way, the functions implemented by easing are also available as “public” macros, such as `\pgfmathsmoothstep`:

$S_1(0) = 0.0$
$S_1(0.25) = 0.15625$
$S_1(0.5) = 0.5$
$S_1(0.75) = 0.84375$
$S_1(1) = 1.0$

```
\input pgf
\usepgflibrary{easing}
\foreach\x in{0,0.25,...,1}{
  \pgfmathsmoothstep{0}{1}{\x}
  $S_1(\x)=\pgfmathresult$\par
}
\end
```

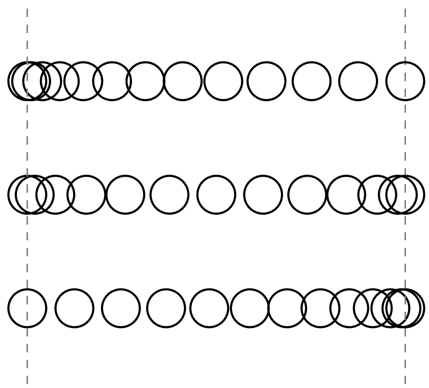
See Part VIII of the PGF manual for more details on the mathematical engine.

3.1 Naming conventions

For each shape, three functions are declared, all of which take three arguments a, b, and x. Where $a < b$, all of these function take value 0 whenever $x \leq a$ and 1 whenever $x \geq b$. The names of the functions adhere to the following pattern:

- The *ease-in* form ⟨*shape*⟩`easein(`a`,`b`,`x`)` has easing applied near the endpoint a.
- The *ease-out* form ⟨*shape*⟩`easeout(`a`,`b`,`x`)` has easing applied near the endpoint b. Its graph is that of the ease-in form reflected about both axes.

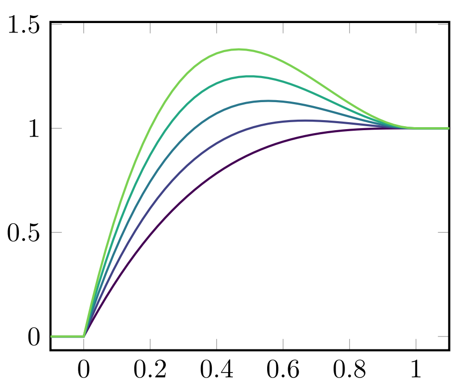
- The *step function* form ⟨*shape*⟩`step(`a`,`b`,`x`)` has easing applied near both endpoints. Its graph is that of the ease-in and ease-out forms concatenated then appropriately scaled.



```
\input tikz
\usepgflibrary{easing}
\tikzpicture
\foreach\x in{0,...,12}{
  \draw[gray,dashed]
    (0,-1) -- (0,4) (5,-1) -- (5,4);
  \draw[thick]
    ({5*smootheasein(0,12,\x)},3)
    circle (0.25)
    ({5*smoothstep(0,12,\x)},1.5)
    circle (0.25)
    ({5*smootheaseout(0,12,\x)},0)
    circle (0.25);
}
\endtikzpicture
\end
```

3.2 Specifying parameters

Some of these shapes can be modified by adjusting one or more parameters, which is done through `pgfkeys`: the parameter ⟨*param*⟩ for functions of shape ⟨*shape*⟩ is specified by setting the PGF key `/easing/`⟨*shape*⟩`/`⟨*param*⟩:




```
\input pgfplots
\usepgflibrary{easing}
\tikzpicture
\axis[
  domain=-0.2:1.2, samples=64,
  xmin=0, xmax=1, enlarge x limits,
  cycle list={
    [samples of colormap=6 of viridis]},
  no marks, thick]
\pgfplotsinvokeforeach{0,...,4}{
  \pgfkeys{easing,back/overshoot=#1}
  \addplot{backeaseout(0,1,x)};
}
\endaxis
\endtikzpicture
\end
```

For detailed descriptions of the parameters admitted by each shape, see the following section.

4 List of easing function shapes

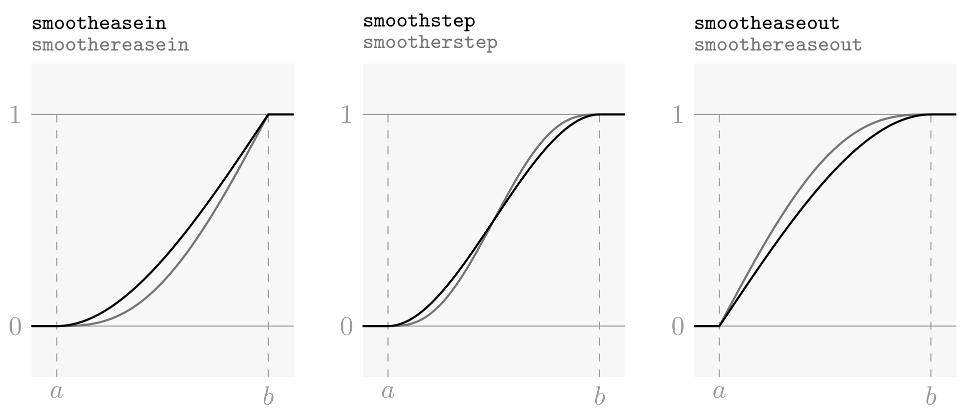
An exhaustive list follows of all the easing functions implemented by the `easing` library. For clarity, where mathematical expressions are given for functions, they are written in terms of a parameter t equal to $\frac{x}{b-a}$.

4.1 Polynomials

4.1.1 The `smooth` and `smoother` shapes

The step function form of the `smooth` shape is a third-order Hermite polynomial interpolation between 0 and 1, so that the first derivate at the endpoints are zero. It is defined $3t^2 - 2t^3$ for $0 \leq t \leq 1$.

The step function form of the `smoother` shape is a fifth-order Hermite polynomial interpolation between 0 and 1, so that the first and second derivates at the endpoints are zero. It is defined $10t^3 - 15t^4 + 6t^5$ for $0 \leq t \leq 1$.

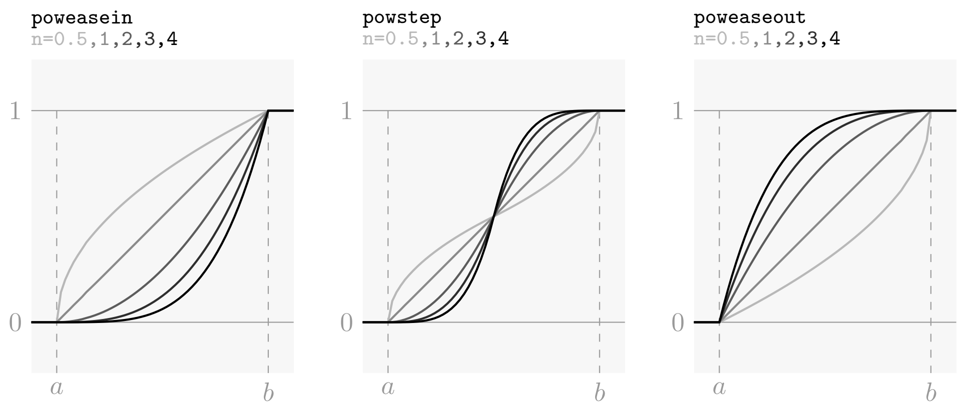


4.1.2 The `pow` shape and friends (`linear`, `quad`, `cubic`, `quart`, and `quint`)

Polynomial easing. The ease-in form is defined as t^n for $0 \leq t \leq 1$, where the exponent n is set with the PGF key `/easing/pow/exponent`, and should be greater than 0. The parameter defaults to $n = 2.4$.

When $n = 1$, the function is linear between 0 and 1. For $0 < n \leq 1$, the ease-in form has discontinuous derivative at 0.

The shapes `linear`, `quad`, `cubic`, `quart`, and `quint` are the same functions as `pow` with $n = 1, \ldots, 5$, respectively. Computations for these shapes are implemented with TEX registers, which is a little faster and more accurate than setting the argument then evaluating the equivalent `pow` function.

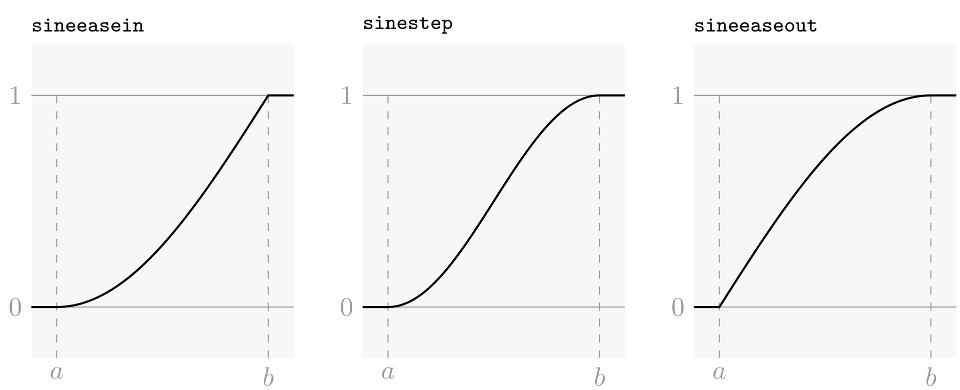


4.2 Trigonometric and exponential

4.2.1 The `sine` shape

An easing function that looks like a section of a sinusoid. The ease-out form is defined as $\sin(\frac{\pi}{2}t)$ for $0 \leq t \leq 1$.

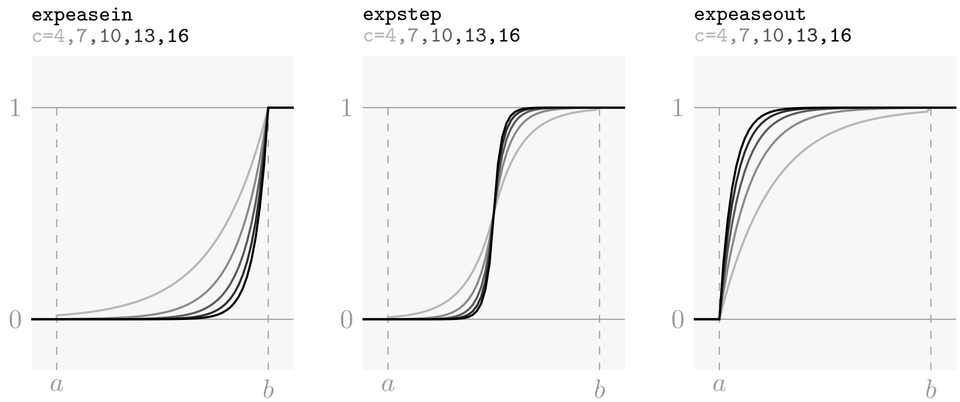
This shape admits no parameters.



4.2.2 The `exp` shape

An easing function that looks like an exponential function. The ease-in form is defined as $e^{c(t-1)}$ for $0 \leq t \leq 1$, where the parameter c is set with the PGF key `/easing/exp/speed`, and should be greater than 0. The parameter defaults to $c = 7.2$.

Because of the nature of the exponential function, this shape is only approximately continuous at the endpoints a and b. In practice, the discontinuity only becomes noticeable for small c, around $c \leq 4$.

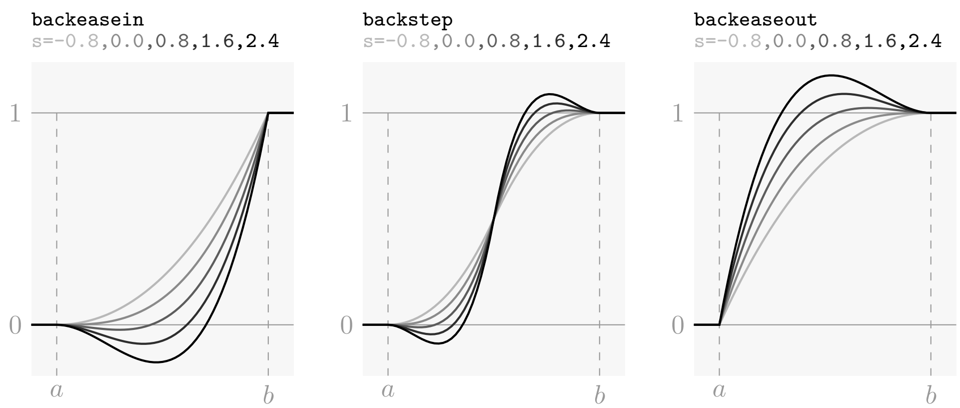


4.3 Other

4.3.1 The `back` shape

Anticipatory easing. The ease-in form is defined as $t^2(1-t)s + t^3$ for $0 \leq t \leq 1$, where the parameter s is set with the PGF key `/easing/back/overshoot`. The parameter defaults to $s = 1.6$.

When $s \leq 0$, there is no overshoot. When $s = 0$, the function is equivalent to `pow` with $n = 3$.

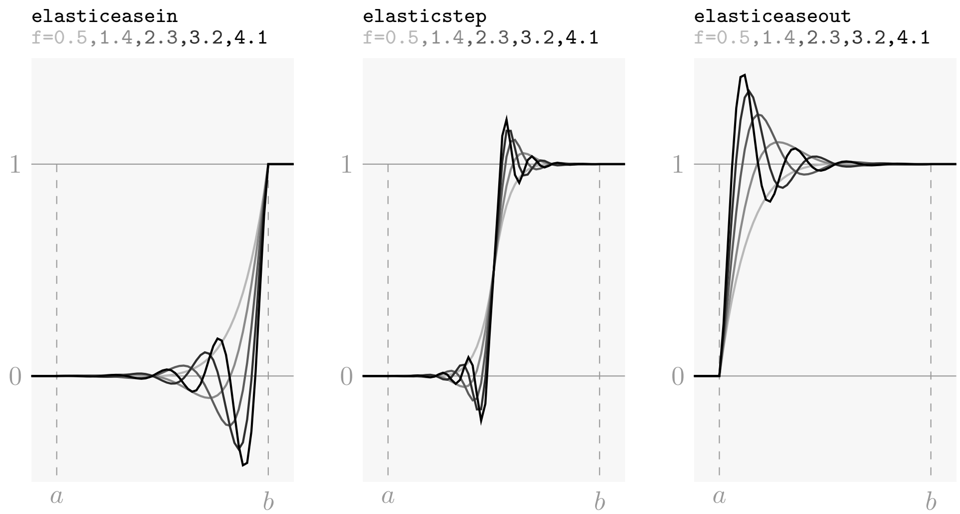


4.3.2 The `elastic` shape

Easing function that looks like a damped harmonic oscillator. The ease-out form is defined as $e^c(t-1)\cos(2\pi f(1-t))$. This shape admits two parameters:

- The frequency f is the number of oscillations between the endpoints. It is set with the PGF key `/easing/elastic/frequency`, and should be greater than 0. The frequency defaults to $f = 3$.
- The damping coefficient b affects the speed at which the amplitude decays. It is set with the PGF key `/easing/elastic/damping`, and should be greater than zero. The damping coefficient defaults to $b = 7.2$.

The function overshoots the range $[0, 1]$ when $f > 0.5$. For $0 < f \leq 1$, this function becomes a family of anticipatory easing curves that look slightly different from the `back` shape but are more expensive to compute.



5 Implementation

`\ifeasing@withfpu`
`\easing@divide`

This library uses TEX registers and PGF's mathematical engine for computations.

It is possible that the user is loading this library together with the floating point unit library. We save the basic routines from `pgfmath`, so that when this happens, the FPU doesn't break everything when it does a switcharoo with the `pgfmath` macros.

```
\newif\ifeasing@withfpu
\expandafter\ifx\csname pgflibraryfpuifactive\endcsname\relax
\easing@withfpufalse
\else
\easing@withfputrue
\fi
\ifeasing@withfpu
\let\easing@divide\pgfmath@basic@divide@
\let\easing@cos\pgfmath@basic@cos@
\let\easing@exp\pgfmath@basic@exp@
\let\easing@ln\pgfmath@basic@ln@
\else
\let\easing@divide\pgfmathdivide@
\let\easing@cos\pgfmathcos@
\let\easing@exp\pgfmathexp@
\let\easing@ln\pgfmathln@
\fi
```

\easing@linearstep@ne
\easing@linearstep@fixed
\easing@linearstep@float
\easing@linearstep

In absence of the FPU, the next section of code defines `\easing@linearstep`, which expects as arguments plain numbers (i.e. things that can be assigned to dimension registers). The net effect of `\easing@linearstep{#1}{#2}{#3}` is to set `\pgfmathresult` to $\frac{\#3-\#1}{\#2-\#1}$, clamped to between 0 and 1.

If the FPU is loaded, `\easing@linearstep` is instead named `\easing@linearstep@fixed`, and we additionally define `\easing@linearstep@float`, which expects FPU-format floats as arguments. We do not format the output as a float since the FPU is smart enough to do that conversion quietly on its own.

The `\easing@linearstep` routine is the first step in the definition of all other routines that compute easing functions.

```
\def\easing@linearstep@ne#1{%
  \begingroup
  \pgf@x#1pt
  \ifdim1pt<\pgf@x\pgf@x 1pt\fi
  \ifdim0pt>\pgf@x\pgf@x 0pt\fi
  \pgfmathreturn\pgf@x
  \endgroup
}%
\expandafter\def
\csname easing@linearstep\ifeasing@withfpu @fixed\fi\endcsname#1#2#3{%
  \begingroup
  \pgf@xa#3pt
  \pgf@xb#2pt
  \pgf@xc#1pt
  \ifdim\pgf@xb=\pgf@xc
  \edef\pgfmathresult{\ifdim\pgf@xa>\pgf@xb 1\else 0\fi}%
  \else
  \advance\pgf@xa-\pgf@xc
  \advance\pgf@xb-\pgf@xc
  \easing@divide{\pgfmath@tonumber\pgf@xa}{\pgfmath@tonumber\pgf@xb}%
  \easing@linearstep@ne\pgfmathresult
  \fi
  \pgfmathsmuggle\pgfmathresult
  \endgroup
}%
\ifeasing@withfpu
\def\easing@linearstep@float#1#2#3{%
  \begingroup
  \pgfmathfloatsubtract{#3}{#1}%
  \edef\pgf@tempa{\pgfmathresult}%
  \pgfmathfloatsubtract{#2}{#1}%
  \edef\pgf@tempb{\pgfmathresult}%
  \pgfmathfloatifflags{\pgf@tempb}{0}{%
    \pgfmathfloatifflags{\pgf@tempa}{-}{%
      \edef\pgfmathresult{0}%
    }{%
      \edef\pgfmathresult{1}%
```

```
    }%
  }{%
    \pgfmathfloatdivide\pgf@tempa\pgf@tempb
    \pgfmathfloattofixed{\pgfmathresult}%
    \easing@linearstep@ne\pgfmathresult
  }%
  \pgfmathsmuggle\pgfmathresult
  \endgroup
}%
\def\easing@linearstep#1#2#3{%
  \pgflibraryfpuifactive{%
    \easing@linearstep@float{#1}{#2}{#3}}{%
    \easing@linearstep@fixed{#1}{#2}{#3}}%
}%
\fi
```

`\easing@linearstep@easein@ne`
`\easing@linearstep@easeout@ne`

The linear ease-in and ease-out functions are identitcal to the linear step function. We define the respective macros so as not to surprise the user with their absence.

```
\let\easing@lineareasein\easing@linearstep
\pgfmathdeclarefunction{lineareasein}{3}{%
  \easing@lineareasein{#1}{#2}{#3}}%
\let\easing@lineareaseout\easing@linearstep
\pgfmathdeclarefunction{lineareaseout}{3}{%
  \easing@lineareasein{#1}{#2}{#3}}%
```

`\easing@derive@easein@nefromstep@ne`
`\easing@derive@easeout@nefromstep@ne`
`\easing@derive@step@nefromeasein@ne`
`\easing@derive@easeout@nefromeasein@ne`

The pattern in general is that, for each shape, we define the one-parameter version of the step, ease-in, and ease-out routines interpolating between values 0 at 1 at the ends of the unit interval. Then by composing with `\easing@linearstep`, we obtain the three-parameter versions that allow the user to specify the begin and end points of the interpolation.

Most of the time it suffices to define just one of the three one-parameter versions of a shape to be able to infer the form of all three. This is done with the `\easing@derive–from–` macros.

```
\def\easing@derive@easein@nefromstep@ne#1{%
  \expandafter\def\csname easing@#1easein@ne\endcsname##1{%
    \begingroup
    \pgf@x##1 pt
    \divide\pgf@x 2
    \csname easing@#1step@ne\endcsname{\pgfmath@tonumber\pgf@x}%
    \pgf@x\pgfmathresult pt
    \multiply\pgf@x 2
    \pgfmathreturn\pgf@x
    \endgroup
  }%
}%
\def\easing@derive@easeout@nefromstep@ne#1{%
```

```
  \expandafter\def\csname easing@#1easeout@ne\endcsname##1{%
    \begingroup
    \pgf@x##1 pt
    \divide\pgf@x 2
    \advance\pgf@x 0.5pt
    \csname easing@#1step@ne\endcsname{\pgfmath@tonumber\pgf@x}%
    \pgf@x\pgfmathresult pt
    \multiply\pgf@x 2
    \advance\pgf@x -1pt
    \pgfmathreturn\pgf@x
    \endgroup
  }%
}%
\def\easing@derive@step@nefromeasein@ne#1{%
  \expandafter\def\csname easing@#1step@ne\endcsname##1{%
  \begingroup
    \pgf@x##1 pt
    \multiply\pgf@x 2
    \ifdim\pgf@x<1pt
    \csname easing@#1easein@ne\endcsname{\pgfmath@tonumber\pgf@x}%
    \pgf@x\pgfmathresult pt
    \divide\pgf@x 2
    \else
    \multiply\pgf@x -1
    \advance\pgf@x 2pt
    \csname easing@#1easein@ne\endcsname{\pgfmath@tonumber\pgf@x}%
    \pgf@x\pgfmathresult pt
    \divide\pgf@x 2
    \multiply\pgf@x -1
    \advance\pgf@x 1pt
    \fi
    \pgfmathreturn\pgf@x
    \endgroup
  }%
}%
\def\easing@derive@easeout@nefromeasein@ne#1{%
  \expandafter\def\csname easing@#1easeout@ne\endcsname##1{%
    \begingroup
    \pgf@x##1pt
    \multiply\pgf@x -1
    \advance\pgf@x 1pt
    \csname easing@#1easein@ne\endcsname{\pgfmath@tonumber\pgf@x}%
    \pgf@x\pgfmathresult pt
    \multiply\pgf@x -1
    \advance\pgf@x 1pt
    \pgfmathreturn\pgf@x
    \endgroup
  }%
}%
```

`\easing@pgfmathinstall`

The three-parameter versions of each routine is installed into the mathematical engine, so that they are available in `\pgfmathparse`.

```
\def\easing@pgfmathinstall#1{%
  \pgfmathdeclarefunction{#1step}{3}{%
    \easing@linearstep{##1}{##2}{##3}%
    \csname easing@#1step@ne\endcsname\pgfmathresult
  }%
  \pgfmathdeclarefunction{#1easein}{3}{%
    \easing@linearstep{##1}{##2}{##3}%
    \csname easing@#1easein@ne\endcsname\pgfmathresult
  }%
  \pgfmathdeclarefunction{#1easeout}{3}{%
    \easing@linearstep{##1}{##2}{##3}%
    \csname easing@#1easeout@ne\endcsname\pgfmathresult
  }%
}%
```

`\easing@smoothstep@ne`
`\easing@smootheasein@ne`
`\easing@smootheaseout@ne`

The `smooth` shape.

```
\def\easing@smoothstep@ne#1{%
  \begingroup
  \pgf@x#1pt
  \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
  \multiply\pgf@x-2
  \advance\pgf@x 3pt
  \pgf@x\pgf@temp\pgf@x
  \pgf@x\pgf@temp\pgf@x
  \pgfmathreturn\pgf@x
  \endgroup
}%
\easing@derive@easein@nefromstep@ne{smooth}%
\easing@derive@easeout@nefromstep@ne{smooth}%
\easing@pgfmathinstall{smooth}%
```

`\easing@smootherstep@ne`
`\easing@smoothereasein@ne`
`\easing@smoothereaseout@ne`

The `smoother` shape.

```
\def\easing@smootherstep@ne#1{%
  \begingroup
  \pgf@x#1pt
  \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
  \multiply\pgf@x 6
  \advance\pgf@x -15pt
  \pgf@x\pgf@temp\pgf@x
  \advance\pgf@x 10pt
  \pgf@x\pgf@temp\pgf@x
  \pgf@x\pgf@temp\pgf@x
  \pgf@x\pgf@temp\pgf@x
  \pgfmathreturn\pgf@x
  \endgroup
```

```
}%
\easing@derive@easein@nefromstep@ne{smoother}%
\easing@derive@easeout@nefromstep@ne{smoother}%
\easing@pgfmathinstall{smoother}%
```

\easing@sinestep@ne
\easing@sineeasein@ne
\easing@sineeaseout@ne

The `sine` shape.

We write down both the `easein` and `step` forms of this, since they are simple compared to what would have been obtained by `\easing@derive`–.

```
\def\easing@sineeasein@ne#1{%
  \begingroup
  \pgf@x#1pt
  \multiply\pgf@x 90
  \easing@cos{\pgfmath@tonumber\pgf@x}%
  \pgf@x\pgfmathresult pt
  \multiply\pgf@x -1
  \advance\pgf@x 1pt
  \pgfmathreturn\pgf@x
  \endgroup
}%
\def\easing@sinestep@ne#1{%
  \begingroup
  \pgf@x#1pt
  \multiply\pgf@x 180
  \easing@cos{\pgfmath@tonumber\pgf@x}%
  \pgf@x\pgfmathresult pt
  \divide\pgf@x 2
  \multiply\pgf@x -1
  \advance\pgf@x 0.5pt
  \pgfmathreturn\pgf@x
  \endgroup
}%
\easing@derive@easeout@nefromeasein@ne{sine}%
\easing@pgfmathinstall{sine}%
```

\easing@powstep@ne
\easing@poweasein@ne
\easing@poweaseout@ne

The `pow` shape.

Because of some wonkiness in the FPU, instead of invoking the `pow` function from `pgfmath`, we compute t^n approximately by computing $e^{n \ln t}$ using `ln` and `exp` instead (which is what `pgfmath` does anyway when the exponent is not an integer.)

```
\pgfkeys{/easing/.is family}%
\pgfkeys{easing,
  pow/exponent/.estore in=\easing@param@pow@exponent,
  pow/exponent/.default=2.4,
  pow/exponent}%
\def\easing@poweasein@ne#1{%
  \begingroup
```

```
  \pgf@x#1pt
  \ifdim\pgf@x=0pt
  \edef\pgfmathresult{0}%
  \else
  \easing@ln{#1}%
  \pgf@x\pgfmathresult pt
  \pgf@x\easing@param@pow@exponent\pgf@x
  \easing@exp{\pgfmath@tonumber\pgf@x}%
  \fi
  \pgfmathsmuggle\pgfmathresult
  \endgroup
}%
\easing@derive@easeout@nefromeasein@ne{pow}%
\easing@derive@step@nefromeasein@ne{pow}%
\easing@pgfmathinstall{pow}%
```

\easing@quadstep@ne
\easing@quadeasein@ne
\easing@quadeaseout@ne
\easing@cubicstep@ne
\easing@cubiceasein@ne
\easing@cubiceaseout@ne
\easing@quartstep@ne
\easing@quarteasein@ne
\easing@quarteaseout@ne
\easing@quintstep@ne
\easing@quinteasein@ne
\easing@quinteaseout@ne

The `quad`–, `cubic`–, `quart`–, and `quint`– routines have explicit definitions.

```
\def\easing@quadeasein@ne#1{%
  \begingroup
  \pgf@x#1pt
  \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
  \pgf@x\pgf@temp\pgf@x
  \pgfmathreturn\pgf@x
  \endgroup
}%
\easing@derive@step@nefromeasein@ne{quad}%
\easing@derive@easeout@nefromeasein@ne{quad}%
\easing@pgfmathinstall{quad}%

\def\easing@cubiceasein@ne#1{%
  \begingroup
  \pgf@x#1pt
  \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
  \pgf@x\pgf@temp\pgf@x
  \pgf@x\pgf@temp\pgf@x
  \pgfmathreturn\pgf@x
  \endgroup
}%
\easing@derive@step@nefromeasein@ne{cubic}%
\easing@derive@easeout@nefromeasein@ne{cubic}%
\easing@pgfmathinstall{cubic}%

\def\easing@quarteasein@ne#1{%
  \begingroup
  \pgf@x#1pt
  \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
  \pgf@x\pgf@temp\pgf@x
  \pgf@x\pgf@temp\pgf@x
  \pgf@x\pgf@temp\pgf@x
```

```
  \pgfmathreturn\pgf@x
  \endgroup
}%
\easing@derive@step@nefromeasein@ne{quart}%
\easing@derive@easeout@nefromeasein@ne{quart}%
\easing@pgfmathinstall{quart}%

\def\easing@quinteasein@ne#1{%
  \begingroup
  \pgf@x#1pt
  \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
  \pgf@x\pgf@temp\pgf@x
  \pgf@x\pgf@temp\pgf@x
  \pgf@x\pgf@temp\pgf@x
  \pgf@x\pgf@temp\pgf@x
  \pgfmathreturn\pgf@x
  \endgroup
}%
\easing@derive@step@nefromeasein@ne{quint}%
\easing@derive@easeout@nefromeasein@ne{quint}%
\easing@pgfmathinstall{quint}%
```

\easing@expstep@ne
\easing@expeasein@ne
\easing@expeaseout@ne

The `exp` shape.

```
\pgfkeys{easing,
  exp/speed/.estore in=\easing@param@exponent@speed,
  exp/speed/.default=7.2,
  exp/speed}%
\def\easing@expeasein@ne#1{%
  \begingroup
  \pgf@x#1pt
  \advance\pgf@x -1pt
  \pgf@x\easing@param@exponent@speed\pgf@x
  \easing@exp{\pgfmath@tonumber\pgf@x}%
  \pgfmathsmuggle\pgfmathresult
  \endgroup
}%
\easing@derive@step@nefromeasein@ne{exp}%
\easing@derive@easeout@nefromeasein@ne{exp}%
\easing@pgfmathinstall{exp}%
```

\easing@backstep@ne
\easing@backeasein@ne
\easing@backeaseout@ne

The `back` shape.

```
\pgfkeys{easing,
  back/overshoot/.estore in=\easing@param@back@overshoot,
  back/overshoot/.default=1.6,
  back/overshoot}%
\def\easing@backeasein@ne#1{%
  \begingroup
  \pgf@x#1pt
```

```
  \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
  \advance\pgf@x -1pt
  \pgf@x\easing@param@back@overshoot\pgf@x
  \advance\pgf@x\pgf@temp pt
  \pgf@x\pgf@temp\pgf@x
  \pgf@x\pgf@temp\pgf@x
  \pgfmathreturn\pgf@x
  \endgroup
}%
\easing@derive@step@nefromeasein@ne{back}%
\easing@derive@easeout@nefromeasein@ne{back}%
\easing@pgfmathinstall{back}%
```

`\easing@elasticstep@ne`
`\easing@elasticeasein@ne`
`\easing@elasticeaseout@ne`

The `elastic` shape.

```
\pgfkeys{easing,
  elastic/frequency/.estore in=\easing@param@elastic@frequency,
  elastic/damping/.estore in=\easing@param@elastic@damping,
  elastic/frequency/.default=3,
  elastic/damping/.default=7.2,
  elastic/frequency, elastic/damping}%
\def\easing@elasticeasein@ne#1{%
  \begingroup
  \pgf@xa#1pt
  \advance\pgf@xa -1pt
  \pgf@xb-\pgf@xa
  \pgf@xa\easing@param@elastic@damping\pgf@xa
  \easing@exp{\pgfmath@tonumber\pgf@xa}%
  \pgf@xa\pgfmathresult pt
  \pgf@xb 360\pgf@xb
  \pgf@xb\easing@param@elastic@frequency\pgf@xb
  \easing@cos{\pgfmath@tonumber\pgf@xb}%
  \pgf@xa\pgfmathresult\pgf@xa
  \pgfmathreturn\pgf@xa
  \endgroup
}%
\easing@derive@step@nefromeasein@ne{elastic}%
\easing@derive@easeout@nefromeasein@ne{elastic}%
\easing@pgfmathinstall{elastic}%
```