The **easing** Library for PGF

Loh Ka-tsun

July 16, 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:
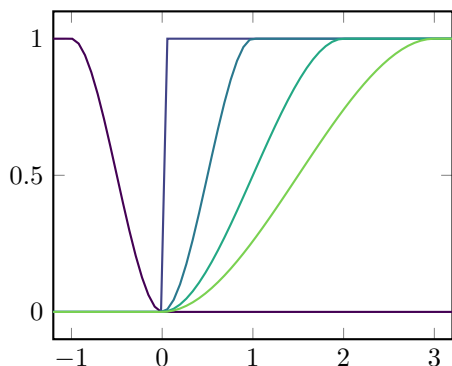

```
\input pgfplots
\usepgflibrary{easing}
\tikzpicture
\axis[
  domain=-1.2:3.2, samples=64,
  xmin=-1.2, xmax=3.2,
  cycle list={
    [samples of colormap=6 of viridis]},
  no marks, thick]
\pgfplotsinvokeforeach{-1,...,3}{
  \addplot{smoothstep(0,#1,x)};}
\endaxis
\endtikzpicture
\end
```

(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 $\langle shape \rangle$`easein(`a`,`b`,`x`)` has easing applied near the endpoint a.

- The *ease-out* form $\langle shape\rangle$`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 $\langle shape\rangle$`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.
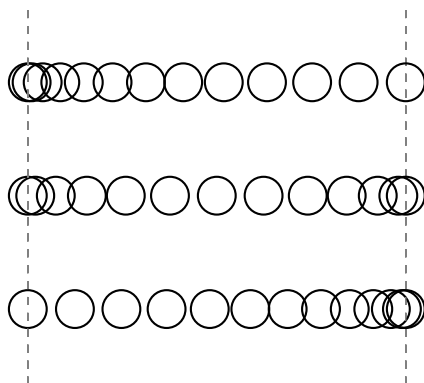
```
\input tikz
\usepgflibrary{easing}
\tikzpicture
\foreach\x in{0,...,12}{
  \draw[gray,dashed]
    (0,-1) -- (0,4) (5,-1) -- (5,4);
  \draw[thick]
    ({5*smootheasein(0,12,\x)},3)
    circle (0.25)
    ({5*smoothstep(0,12,\x)},1.5)
    circle (0.25)
    ({5*smootheaseout(0,12,\x)},0)
    circle (0.25);
}
\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 $\langle param\rangle$ for functions of shape $\langle shape\rangle$ is specified by setting the PGF key `/easing/`$\langle shape\rangle$`/`$\langle param\rangle$:
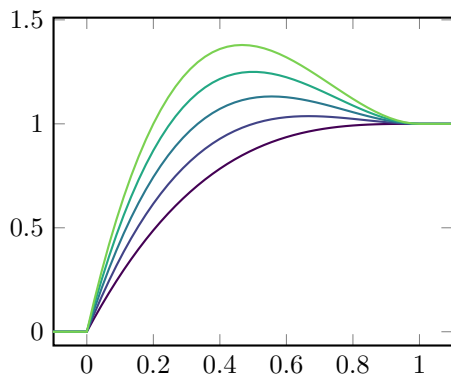
```
\input pgfplots
\usepgflibrary{easing}
\tikzpicture
\axis[
  domain=-0.2:1.2, samples=64,
  xmin=0, xmax=1, enlarge x limits,
  cycle list={
    [samples of colormap=6 of viridis]},
  no marks, thick]
\pgfplotsinvokeforeach{0,...,4}{
  \pgfkeys{easing,back/overshoot=#1}
  \addplot{backeaseout(0,1,x)};
}
\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 $0 \leq t \leq 1$.
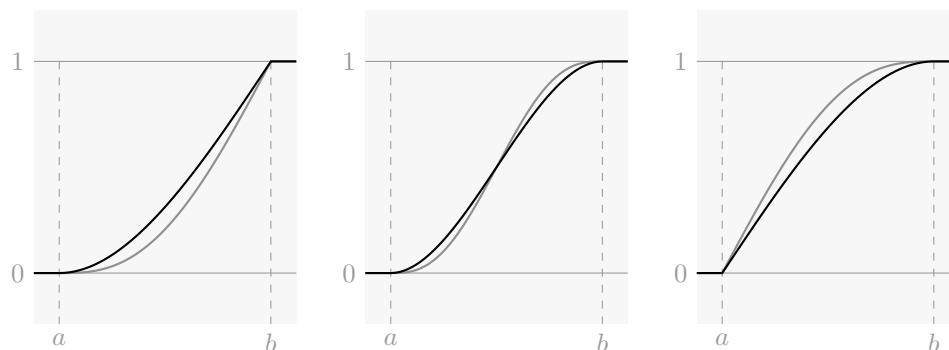

4.1.2 The `pow` shape and friends (`linear`, `quad`, `cubic`, `quart`, and `quint`)

Polynomial easing. The ease-in form is defined as t^n for $0 \leq t \leq 1$, where the exponent n is set by 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.
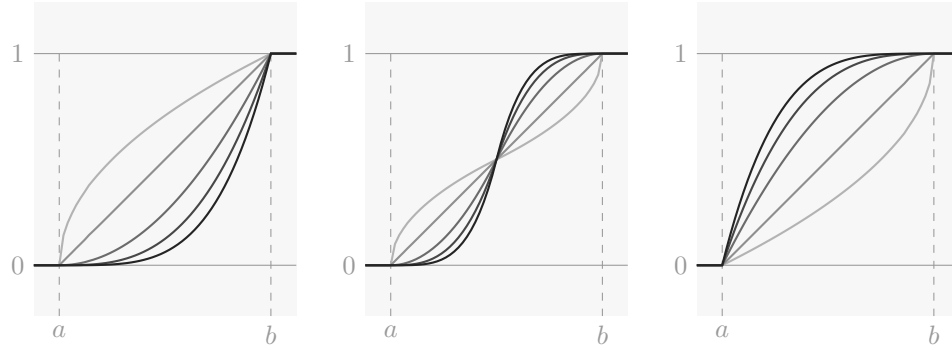
4.2 Trigonometric and exponential

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 $0 \leq t \leq 1$.
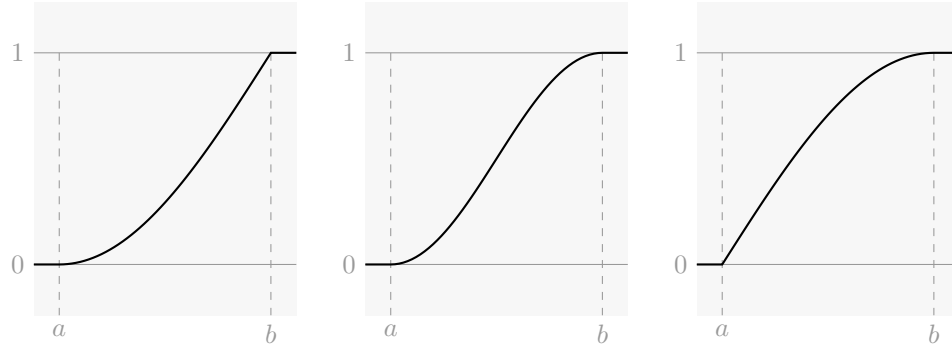
4.2.2 The `exp` shape

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

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 α, around $\alpha \leq 4$.
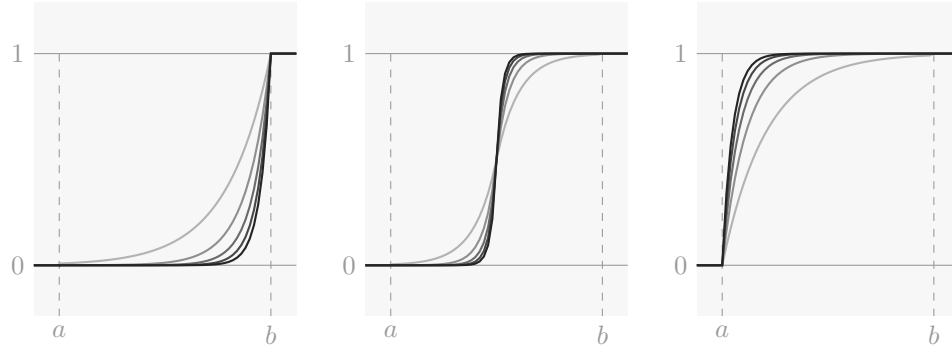
4.3 Other

4.3.1 The `back` shape

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

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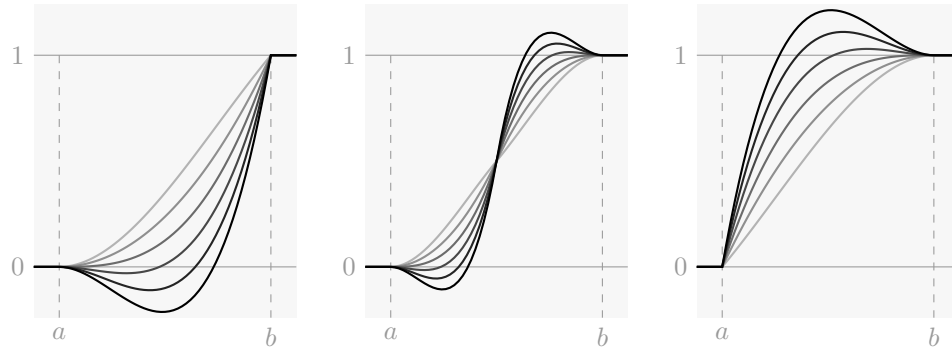
5   Implementation

`\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.

```
1 \newif\ifeasing@withfpu
2 \expandafter\ifx\csname pgflibraryfpuifactive\endcsname\relax
3 \easing@withfpufalse
4 \else
```

```
5 \easing@withfputrue
6 \fi
7 \ifeasing@withfpu
8 \let\easing@divide\pgfmath@basic@divide@
9 \let\easing@cos\pgfmath@basic@cos@
10 \let\easing@exp\pgfmath@basic@exp@
11 \let\easing@ln\pgfmath@basic@ln@
12 \else
13 \let\easing@divide\pgfmathdivide@
14 \let\easing@cos\pgfmathcos@
15 \let\easing@exp\pgfmathexp@
16 \let\easing@ln\pgfmathln@
17 \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.

```
18 \def\easing@linearstep@ne#1{%
19   \begingroup
20   \pgf@x#1pt
21   \ifdim1pt<\pgf@x\pgf@x 1pt\fi
22   \ifdim0pt>\pgf@x\pgf@x 0pt\fi
23   \pgfmathreturn\pgf@x
24   \endgroup
25 }%
26 \expandafter\def
27 \csname easing@linearstep\ifeasing@withfpu @fixed\fi\endcsname#1#2#3{%
28   \begingroup
29   \pgf@xa#3pt
30   \pgf@xb#2pt
31   \pgf@xc#1pt
32   \ifdim\pgf@xb=\pgf@xc
33   \edef\pgfmathresult{\ifdim\pgf@xa>\pgf@xb 1\else 0\fi}%
34   \else
35   \advance\pgf@xa-\pgf@xc
36   \advance\pgf@xb-\pgf@xc
37   \easing@divide{\pgfmath@tonumber\pgf@xa}{\pgfmath@tonumber\pgf@xb}%
38   \easing@linearstep@ne\pgfmathresult
39   \fi
40   \pgfmathsmuggle\pgfmathresult
```

```
41    \endgroup
42 }%
43 \ifeasing@withfpu
44 \def\easing@linearstep@float#1#2#3{%
45    \begingroup
46    \pgfmathfloatsubtract{#3}{#1}%
47    \edef\pgf@tempa{\pgfmathresult}%
48    \pgfmathfloatsubtract{#2}{#1}%
49    \edef\pgf@tempb{\pgfmathresult}%
50    \pgfmathfloatifflags{\pgf@tempb}{0}{%
51      \pgfmathfloatifflags{\pgf@tempa}{-}{%
52        \edef\pgfmathresult{0}%
53      }{%
54        \edef\pgfmathresult{1}%
55      }%
56    }{%
57      \pgfmathfloatdivide\pgf@tempa\pgf@tempb
58      \pgfmathfloattofixed{\pgfmathresult}%
59      \easing@linearstep@ne\pgfmathresult
60    }%
61    \pgfmathsmuggle\pgfmathresult
62    \endgroup
63 }%
64 \def\easing@linearstep#1#2#3{%
65    \pgflibraryfpuifactive{%
66      \easing@linearstep@float{#1}{#2}{#3}}{%
67      \easing@linearstep@fixed{#1}{#2}{#3}}%
68 }%
69 \fi
```

\easing@linearstep@easein@ne   The linear ease-in and ease-out functions are identitcal to the linear step function.
\easing@linearstep@easeout@ne   We define the respective macros so as not to surprise the user with their absence.

```
70 \let\easing@lineareasein\easing@linearstep
71 \pgfmathdeclarefunction{lineareasein}{3}{%
72    \easing@lineareasein{#1}{#2}{#3}}%
73 \let\easing@lineareaseout\easing@linearstep
74 \pgfmathdeclarefunction{lineareaseout}{3}{%
75    \easing@lineareasein{#1}{#2}{#3}}%
```

\easing@derive@easein@nefromstep@ne   The pattern in general is that, for each shape, we define the one-parameter version
\easing@derive@easeout@nefromstep@ne   of the step, ease-in, and ease-out routines interpolating between values 0 at 1 at
\easing@derive@step@nefromeasein@ne   the ends of the unit interval. Then by composing with \easing@linearstep, we
\easing@derive@easeout@nefromeasein@ne 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.

```
76 \def\easing@derive@easein@nefromstep@ne#1{%
77   \expandafter\def\csname easing@#1easein@ne\endcsname##1{%
78     \begingroup
79     \pgf@x##1 pt
80     \divide\pgf@x 2
81     \csname easing@#1step@ne\endcsname{\pgfmath@tonumber\pgf@x}%
82     \pgf@x\pgfmathresult pt
83     \multiply\pgf@x 2
84     \pgfmathreturn\pgf@x
85     \endgroup
86   }%
87 }%
88 \def\easing@derive@easeout@nefromstep@ne#1{%
89   \expandafter\def\csname easing@#1easeout@ne\endcsname##1{%
90     \begingroup
91     \pgf@x##1 pt
92     \divide\pgf@x 2
93     \advance\pgf@x 0.5pt
94     \csname easing@#1step@ne\endcsname{\pgfmath@tonumber\pgf@x}%
95     \pgf@x\pgfmathresult pt
96     \multiply\pgf@x 2
97     \advance\pgf@x -1pt
98     \pgfmathreturn\pgf@x
99     \endgroup
100  }%
101 }%
102 \def\easing@derive@step@nefromeasein@ne#1{%
103   \expandafter\def\csname easing@#1step@ne\endcsname##1{%
104   \begingroup
105     \pgf@x##1 pt
106     \multiply\pgf@x 2
107     \ifdim\pgf@x<1pt
108     \csname easing@#1easein@ne\endcsname{\pgfmath@tonumber\pgf@x}%
109     \pgf@x\pgfmathresult pt
110     \divide\pgf@x 2
111     \else
112     \multiply\pgf@x -1
113     \advance\pgf@x 2pt
114     \csname easing@#1easein@ne\endcsname{\pgfmath@tonumber\pgf@x}%
115     \pgf@x\pgfmathresult pt
116     \divide\pgf@x 2
117     \multiply\pgf@x -1
118     \advance\pgf@x 1pt
119     \fi
120     \pgfmathreturn\pgf@x
121     \endgroup
122   }%
123 }%
```

9

```
124 \def\easing@derive@easeout@nefromeasein@ne#1{%
125   \expandafter\def\csname easing@#1easeout@ne\endcsname##1{%
126     \begingroup
127     \pgf@x##1pt
128     \multiply\pgf@x -1
129     \advance\pgf@x 1pt
130     \csname easing@#1easein@ne\endcsname{\pgfmath@tonumber\pgf@x}%
131     \pgf@x\pgfmathresult pt
132     \multiply\pgf@x -1
133     \advance\pgf@x 1pt
134     \pgfmathreturn\pgf@x
135     \endgroup
136   }%
137 }%
```

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

```
138 \def\easing@pgfmathinstall#1{%
139   \pgfmathdeclarefunction{#1step}{3}{%
140     \easing@linearstep{##1}{##2}{##3}%
141     \csname easing@#1step@ne\endcsname\pgfmathresult
142   }%
143   \pgfmathdeclarefunction{#1easein}{3}{%
144     \easing@linearstep{##1}{##2}{##3}%
145     \csname easing@#1easein@ne\endcsname\pgfmathresult
146   }%
147   \pgfmathdeclarefunction{#1easeout}{3}{%
148     \easing@linearstep{##1}{##2}{##3}%
149     \csname easing@#1easeout@ne\endcsname\pgfmathresult
150   }%
151 }%
```

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

The smooth shape.

```
152 \def\easing@smoothstep@ne#1{%
153   \begingroup
154   \pgf@x#1pt
155   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
156   \multiply\pgf@x-2
157   \advance\pgf@x 3pt
158   \pgf@x\pgf@temp\pgf@x
159   \pgf@x\pgf@temp\pgf@x
160   \pgfmathreturn\pgf@x
161   \endgroup
162 }%
163 \easing@derive@easein@nefromstep@ne{smooth}%
164 \easing@derive@easeout@nefromstep@ne{smooth}%
165 \easing@pgfmathinstall{smooth}%
```

<easing@smootherstep@ne
<easing@smoothereasein@ne
<easing@smoothereaseout@ne   The smoother shape.

```
166 \def\easing@smootherstep@ne#1{%
167   \begingroup
168   \pgf@x#1pt
169   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
170   \multiply\pgf@x 6
171   \advance\pgf@x -15pt
172   \pgf@x\pgf@temp\pgf@x
173   \advance\pgf@x 10pt
174   \pgf@x\pgf@temp\pgf@x
175   \pgf@x\pgf@temp\pgf@x
176   \pgf@x\pgf@temp\pgf@x
177   \pgfmathreturn\pgf@x
178   \endgroup
179 }%
180 \easing@derive@easein@nefromstep@ne{smoother}%
181 \easing@derive@easeout@nefromstep@ne{smoother}%
182 \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–.

```
183 \def\easing@sineeasein@ne#1{%
184   \begingroup
185   \pgf@x#1pt
186   \multiply\pgf@x 90
187   \easing@cos{\pgfmath@tonumber\pgf@x}%
188   \pgf@x\pgfmathresult pt
189   \multiply\pgf@x -1
190   \advance\pgf@x 1pt
191   \pgfmathreturn\pgf@x
192   \endgroup
193 }%
194 \def\easing@sinestep@ne#1{%
195   \begingroup
196   \pgf@x#1pt
197   \multiply\pgf@x 180
198   \easing@cos{\pgfmath@tonumber\pgf@x}%
199   \pgf@x\pgfmathresult pt
200   \divide\pgf@x 2
201   \multiply\pgf@x -1
202   \advance\pgf@x 0.5pt
203   \pgfmathreturn\pgf@x
204   \endgroup
205 }%
206 \easing@derive@easeout@nefromeasein@ne{sine}%
207 \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.)

```
208 \pgfkeys{/easing/.is family}%
209 \pgfkeys{easing,
210   pow/exponent/.estore in=\easing@param@pow@exponent,
211   pow/exponent/.default=2.4,
212   pow/exponent}%
213 \def\easing@poweasein@ne#1{%
214   \begingroup
215   \pgf@x#1pt
216   \ifdim\pgf@x=0pt
217   \edef\pgfmathresult{0}%
218   \else
219   \easing@ln{#1}%
220   \pgf@x\pgfmathresult pt
221   \pgf@x\easing@param@pow@exponent\pgf@x
222   \easing@exp{\pgfmath@tonumber\pgf@x}%
223   \fi
224   \pgfmathsmuggle\pgfmathresult
225   \endgroup
226 }%
227 \easing@derive@easeout@nefromeasein@ne{pow}%
228 \easing@derive@step@nefromeasein@ne{pow}%
229 \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.

```
230 \def\easing@quadeasein@ne#1{%
231   \begingroup
232   \pgf@x#1pt
233   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
234   \pgf@x\pgf@temp\pgf@x
235   \pgfmathreturn\pgf@x
236   \endgroup
237 }%
238 \easing@derive@step@nefromeasein@ne{quad}%
239 \easing@derive@easeout@nefromeasein@ne{quad}%
240 \easing@pgfmathinstall{quad}%
241
242 \def\easing@cubiceasein@ne#1{%
243   \begingroup
244   \pgf@x#1pt
245   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
246   \pgf@x\pgf@temp\pgf@x
247   \pgf@x\pgf@temp\pgf@x
```

```
248   \pgfmathreturn\pgf@x
249   \endgroup
250 }%
251 \easing@derive@step@nefromeasein@ne{cubic}%
252 \easing@derive@easeout@nefromeasein@ne{cubic}%
253 \easing@pgfmathinstall{cubic}%
254
255 \def\easing@quarteasein@ne#1{%
256   \begingroup
257   \pgf@x#1pt
258   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
259   \pgf@x\pgf@temp\pgf@x
260   \pgf@x\pgf@temp\pgf@x
261   \pgf@x\pgf@temp\pgf@x
262   \pgfmathreturn\pgf@x
263   \endgroup
264 }%
265 \easing@derive@step@nefromeasein@ne{quart}%
266 \easing@derive@easeout@nefromeasein@ne{quart}%
267 \easing@pgfmathinstall{quart}%
268
269 \def\easing@quinteasein@ne#1{%
270   \begingroup
271   \pgf@x#1pt
272   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
273   \pgf@x\pgf@temp\pgf@x
274   \pgf@x\pgf@temp\pgf@x
275   \pgf@x\pgf@temp\pgf@x
276   \pgf@x\pgf@temp\pgf@x
277   \pgfmathreturn\pgf@x
278   \endgroup
279 }%
280 \easing@derive@step@nefromeasein@ne{quint}%
281 \easing@derive@easeout@nefromeasein@ne{quint}%
282 \easing@pgfmathinstall{quint}%
```

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

```
283 \pgfkeys{easing,
284   exp/speed/.estore in=\easing@param@exponent@speed,
285   exp/speed/.default=7.2,
286   exp/speed}%
287 \def\easing@expeasein@ne#1{%
288   \begingroup
289   \pgf@x#1pt
290   \advance\pgf@x -1pt
291   \pgf@x\easing@param@exponent@speed\pgf@x
292   \easing@exp{\pgfmath@tonumber\pgf@x}%
293   \pgfmathsmuggle\pgfmathresult
294   \endgroup
```

```
295 }%
296 \easing@derive@step@nefromeasein@ne{exp}%
297 \easing@derive@easeout@nefromeasein@ne{exp}%
298 \easing@pgfmathinstall{exp}%
```

\easing@backstep@ne
\easing@backeasein@ne
\easing@backeaseout@ne   The back shape.

```
299 \pgfkeys{easing,
300   back/overshoot/.estore in=\easing@param@back@overshoot,
301   back/overshoot/.default=1.6,
302   back/overshoot}%
303 \def\easing@backeasein@ne#1{%
304   \begingroup
305   \pgf@x#1pt
306   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
307   \advance\pgf@x -1pt
308   \pgf@x\easing@param@back@overshoot\pgf@x
309   \advance\pgf@x\pgf@temp pt
310   \pgf@x\pgf@temp\pgf@x
311   \pgf@x\pgf@temp\pgf@x
312   \pgfmathreturn\pgf@x
313   \endgroup
314 }%
315 \easing@derive@step@nefromeasein@ne{back}%
316 \easing@derive@easeout@nefromeasein@ne{back}%
317 \easing@pgfmathinstall{back}%
```