

The **easing** Library for PGF

Loh Ka-tsun

July 15, 2021

1 Introduction

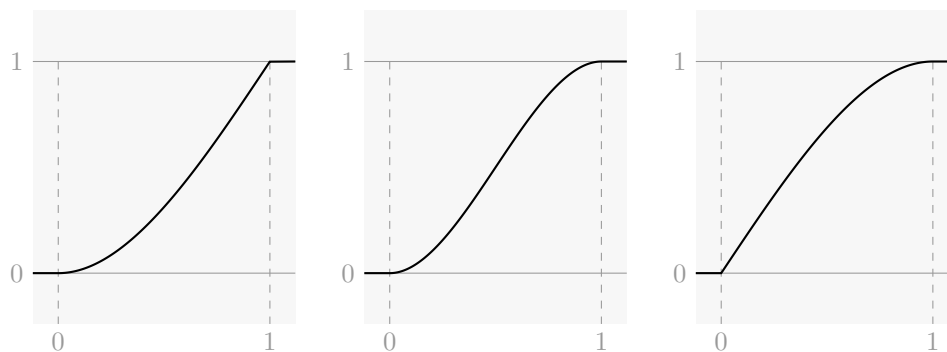
This library provides easing functions for the PGF mathematical engine.

2 Usage

3 List of easing function shapes

3.1 Polynomial and trigonometric

3.1.1 The smooth and smoother shapes



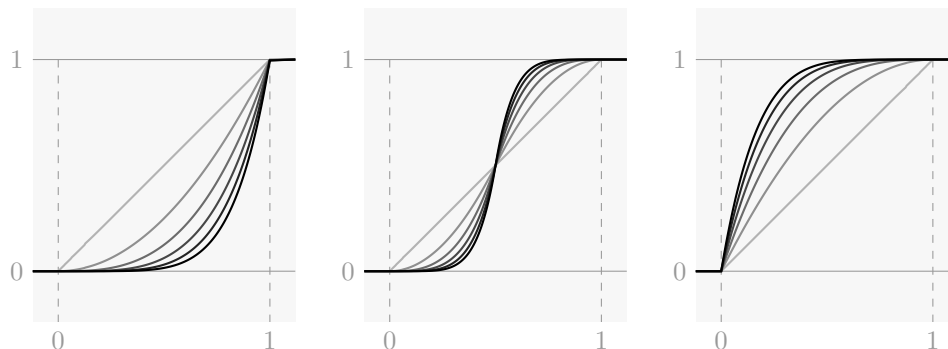
3.1.2 The pow shape and friends (linear, quad, cubic, quart, and quint)

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 -1 . The exponent defaults to 2.4.

When $n = 1$, the function is linear between 0 and 1.

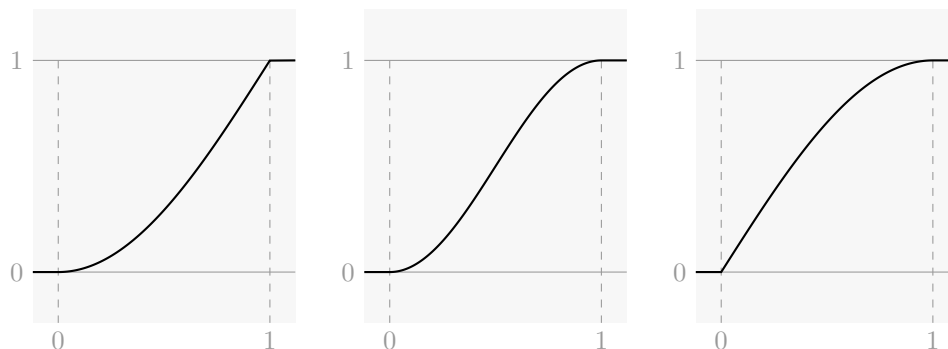
The shapes `linear`, `quad`, `cubic`, `quart`, and `quint` are the same functions as `pow` with $n = 1, \dots, 5$, respectively. Computations for these shapes are implemented

with \TeX registers, which is a little faster and more accurate than setting the argument then evaluating the equivalent `pow` function.



3.1.3 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 $0 \leq t \leq 1$.

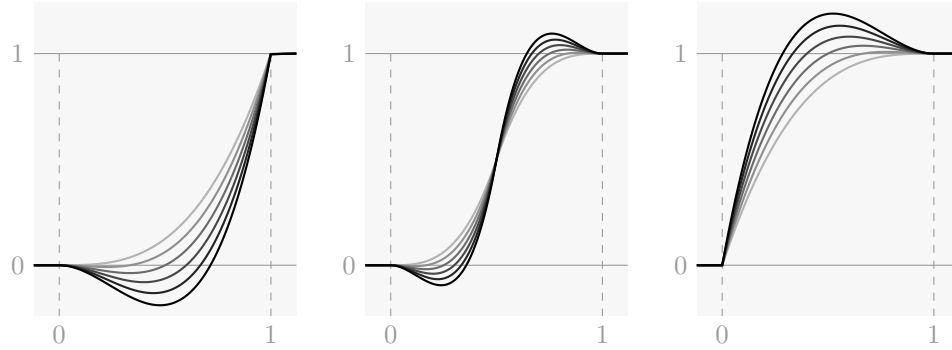


3.2 Other

3.2.1 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 by the PGF key `/easing/back/overshoot`. The parameter n defaults to 1.6.

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



4 Implementation

`\ifeasing@withfpu` This library uses \TeX registers and PGF's mathematical engine for computations.
`\easing@divide` 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` In absence of the FPU, the next section of code defines `\easing@linearstep`,
`\easing@linearstep@fixed` which expects as arguments plain numbers (i.e. things that can be assigned to
`\easing@linearstep@float` dimension registers). The net effect of `\easing@linearstep{#1}{#2}{#3}` is to
`\easing@linearstep` 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 \pgflibraryfpuiactive{%
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 identical 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 }%
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 The smooth shape.
\ easing@smootheasein@ne
\ easing@smootheaseout@ne 152 \def\easing@smoothstep@ne#1{%
153 \beginpgroup
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 \endpgroup
162 }%
163 \easing@derive@easein@nefromstep@ne{smooth}%
164 \easing@derive@easeout@nefromstep@ne{smooth}%
165 \easing@pgfmathinstall{smooth}%

```

`\easing@sinestep@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-`.

```

166 \def\easing@sineeasein@ne#1{%
167   \begingroup
168   \pgf@x#1pt
169   \multiply\pgf@x 90
170   \easing@cos{\pgfmath@tonumber\pgf@x}%
171   \pgf@x\pgfmathresult pt
172   \multiply\pgf@x -1
173   \advance\pgf@x 1pt
174   \pgfmathreturn\pgf@x
175   \endgroup
176 }%
177 \def\easing@sinestep@ne#1{%
178   \begingroup
179   \pgf@x#1pt
180   \multiply\pgf@x 180
181   \easing@cos{\pgfmath@tonumber\pgf@x}%
182   \pgf@x\pgfmathresult pt
183   \divide\pgf@x 2
184   \multiply\pgf@x -1
185   \advance\pgf@x 0.5pt

```

```

186 \pgfmathreturn\pgf@x
187 \endgroup
188 }%
189 \easing@derive@easeout@nefromeasein@ne{sine}%
190 \easing@pgfmathinstall{sine}%

\.easing@powstep@ne The pow shape.
\.easing@poweasein@ne Because of some wonkiness in the FPU, instead of invoking the pow function
\.easing@poweaseout@ne 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.)

191 \pgfkeys{/easing/.is family}%
192 \pgfkeys{easing,
193   pow/exponent/.estore in=\easing@param@pow@exponent,
194   pow/exponent/.default=2.4,
195   pow/exponent}%
196 \def\easing@poweasein@ne#1{%
197   \beginpgfgroup
198   \pgf@x#1pt
199   \ifdim\pgf@x=0pt
200     \edef\pgfmathresult{0}%
201   \else
202     \easing@ln{#1}%
203     \pgf@x\pgfmathresult pt
204     \pgf@x\easing@param@pow@exponent\pgf@x
205     \easing@exp{\pgfmath@tonumber\pgf@x}%
206     \fi
207     \pgfmathsmuggle\pgfmathresult
208   \endpgfgroup
209 }%
210 \easing@derive@easeout@nefromeasein@ne{pow}%
211 \easing@derive@step@nefromeasein@ne{pow}%
212 \easing@pgfmathinstall{pow}%

\.easing@quadstep@ne The quad-, cubic-, quart-, and quint- routines have explicit definitions.
\.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

213 \def\easing@quadeasein@ne#1{%
214   \beginpgfgroup
215   \pgf@x#1pt
216   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
217   \pgf@x\pgf@temp\pgf@x
218   \pgfmathreturn\pgf@x
219   \endpgfgroup
220 }%
221 \easing@derive@step@nefromeasein@ne{quad}%
222 \easing@derive@easeout@nefromeasein@ne{quad}%
223 \easing@pgfmathinstall{quad}%
224

```



```

225 \def\easing@cubiceasein@ne#1{%
226   \begingroup
227   \pgf@x#1pt
228   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
229   \pgf@x\pgf@temp\pgf@x
230   \pgf@x\pgf@temp\pgf@x
231   \pgfmathreturn\pgf@x
232   \endgroup
233 }%
234 \easing@derive@step@nefromeasein@ne{cubic}%
235 \easing@derive@easeout@nefromeasein@ne{cubic}%
236 \easing@pgfmathinstall{cubic}%
237
238 \def\easing@quarteasein@ne#1{%
239   \begingroup
240   \pgf@x#1pt
241   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
242   \pgf@x\pgf@temp\pgf@x
243   \pgf@x\pgf@temp\pgf@x
244   \pgf@x\pgf@temp\pgf@x
245   \pgfmathreturn\pgf@x
246   \endgroup
247 }%
248 \easing@derive@step@nefromeasein@ne{quart}%
249 \easing@derive@easeout@nefromeasein@ne{quart}%
250 \easing@pgfmathinstall{quart}%
251
252 \def\easing@quinteasein@ne#1{%
253   \begingroup
254   \pgf@x#1pt
255   \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
256   \pgf@x\pgf@temp\pgf@x
257   \pgf@x\pgf@temp\pgf@x
258   \pgf@x\pgf@temp\pgf@x
259   \pgf@x\pgf@temp\pgf@x
260   \pgfmathreturn\pgf@x
261   \endgroup
262 }%
263 \easing@derive@step@nefromeasein@ne{quint}%
264 \easing@derive@easeout@nefromeasein@ne{quint}%
265 \easing@pgfmathinstall{quint}%

```

```

\easing@backstep@ne The back shape.
\easing@backeasein@ne
\easing@backeaseout@ne 266 \pgfkeys{easing,
267   back/overshoot/.estore in=\easing@param@back@overshoot,
268   back/overshoot/.default=1.6,
269   back/overshoot}%
270 \def\easing@backeasein@ne#1{%
271   \begingroup

```

```

272 \pgf@x#1pt
273 \edef\pgf@temp{\pgfmath@tonumber\pgf@x}%
274 \advance\pgf@x -1pt
275 \pgf@x\easing@param@back@overshoot\pgf@x
276 \advance\pgf@x\pgf@temp pt
277 \pgf@x\pgf@temp\pgf@x
278 \pgf@x\pgf@temp\pgf@x
279 \pgfmathreturn\pgf@x
280 \endgroup
281 }%
282 \easing@derive@step@nfromeasein@ne{back}%
283 \easing@derive@easeout@nfromeasein@ne{back}%
284 \easing@pgfmathinstall{back}%

```