

The `randomwalk` package: customizable random walks*

Bruno Le Floch[†]

Released on 2015/09/20

Contents

1	How to use <code>randomwalk</code>	2
2	<code>randomwalk</code> implementation	3
2.1	Packages	3
2.2	Variables	4
2.3	User command and key-value list	5
2.4	Setup	5
2.5	Drawing	7
2.6	On random numbers and items	8

Abstract

The `randomwalk` package draws random walks. The following parameters can be customized:

- The number of steps, of course.
- The length of the steps, either a fixed length, or a length taken uniformly at random from a given list.
- The angle of each step, either taken uniformly at random from a given list, or uniformly distributed between 0 and 360 degrees.

*This file describes version v0.4, last revised 2015/09/20.

[†]E-mail blflatex@gmail.com

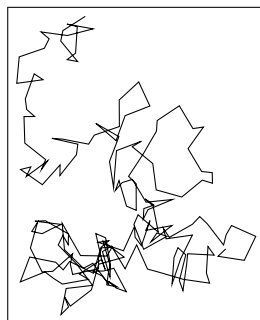


Figure 1: A 200 steps long walk, where each step has one of two lengths: `\RandomWalk {number = 200, length = {4pt, 10pt}}`

1 How to use `randomwalk`

`\RandomWalk`

The `randomwalk` package has a single user command: `\RandomWalk`, which takes a list of key-value pairs as its argument. A few examples are given in Figures 1, 2, and 3:

```
\RandomWalk {number = 200, length = {4pt, 10pt}}
\RandomWalk {number = 100, angles = {0,60,120,180,240,300}, degree}
\RandomWalk {number = 50, length = 1ex, angles = {0,24,48,-24,-48},
degree, angles-relative}
```

Here is a list of all the keys, and their meaning:

- **number**: the number of steps (default 10)
- **length**: the length of each step: either one dimension (*e.g.*, `1ex`), or a comma-separated list of dimensions (*e.g.*, `{2pt, 5pt}`), by default `10pt`. The length of each step is a (uniformly distributed) random element in this set of possible dimensions.
- **angles**: the polar angle for each step: a comma-separated list of angles, and each step takes a random angle in the list. If this is not specified, then the angle is uniformly distributed along the circle.
- **degree** or **degrees**: specify that the angles are given in degrees (by default, they are in radians).
- **angles-relative**: instead of being absolute, the angles are relative to the direction of the previous step.
- **revert-random** (boolean, false by default): revert the seed of the random number generator to its original value after the random walk.

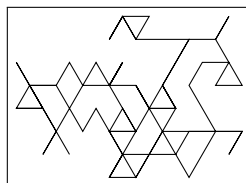


Figure 2: A walk with constrained angles: `\RandomWalk {number = 100, angles = {0,60,120,180,240,300}, degree}`

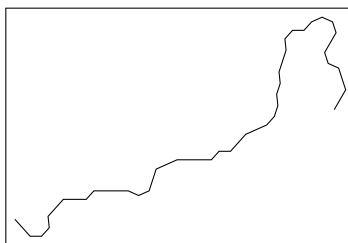


Figure 3: A last example, with small relative angles: `\RandomWalk {number = 50, length = 1ex, angles = {0,24,48,-24,-48}, degree, angles-relative}`

2 randomwalk implementation

2.1 Packages

The `expl3` bundle is loaded first.

```
<*package>
1 <@@=randomwalk>
2 \RequirePackage{expl3}[2015/09/11]
3 \ProvidesExplPackage
4   {randomwalk.sty} {2015/09/20} {0.4} {Customizable random walks}
5 \RequirePackage{xparse}[2015/09/11]
```

Load `pgfcore` for figures.

```
6 \RequirePackage{pgfcore}
```

Load `lcg` for random numbers. It needs to know the smallest and biggest random numbers that should be produced, which we take to be 0 and `\c__randomwalk_lcg_last_int = 231 - 2`. It will then store them in `\c@lcg@rand`: the `\c@` is there because of how $\text{\LaTeX} 2_{\epsilon}$ defines counters. To make it clear that `\c` has a very special meaning here, I do not follow $\text{\LaTeX} 3$ naming conventions. Also of note is that I use `\cr@nd` in `__randomwalk_walk:.`

It seems that the `lcg` package has to be loaded after the document class, hence we do it `\AtBeginDocument`. Also worth noting is the call to `\rand`, which avoids some very odd bug.

```
7 \int_const:Nn \c__randomwalk_lcg_last_int { \c_max_int - \c_one }
```

```

8 \AtBeginDocument
9 {
10   \RequirePackage
11   [
12     first= \c_zero ,
13     last = \c__randomwalk_lcg_last_int ,
14     counter = lcg@rand
15   ]
16   { lcg }
17   \rand
18 }

```

2.2 Variables

`\l__randomwalk_internal_tl` Used for scratch assignments.

```

19 \tl_new:N \l__randomwalk_internal_tl
20 \int_new:N \l__randomwalk_internal_int

```

(End definition for \l__randomwalk_internal_tl and \l__randomwalk_internal_int.)

`\l__randomwalk_step_number_int` The number of steps requested by the caller.

```

21 \int_new:N \l__randomwalk_step_number_int

```

(End definition for \l__randomwalk_step_number_int.)

`\l__randomwalk_relative_angles_bool` `\l__randomwalk_degrees_bool` Booleans for whether angles are relative (keyval option), and whether they are in degrees.

```

22 \bool_new:N \l__randomwalk_relative_angles_bool
23 \bool_new:N \l__randomwalk_degrees_bool

```

(End definition for \l__randomwalk_relative_angles_bool and \l__randomwalk_degrees_bool.)

`\l__randomwalk_revert_random_bool` Booleans for whether to revert the random seed to its original value or keep the last value reached at the end of a random path.

```

24 \bool_new:N \l__randomwalk_revert_random_bool

```

(End definition for \l__randomwalk_revert_random_bool.)

`__randomwalk_next_angle:` `__randomwalk_next_length:` Set the `\l__randomwalk_angle_fp` and `\l__randomwalk_length_fp` of the next step, most often randomly.

```

25 \cs_new_protected_nopar:Npn \__randomwalk_next_angle: { }
26 \cs_new_protected_nopar:Npn \__randomwalk_next_length: { }

```

(End definition for __randomwalk_next_angle: and __randomwalk_next_length:.)

`\l__randomwalk_angle_fp` `\l__randomwalk_length_fp` Angle and length of the next step.

```

27 \fp_new:N \l__randomwalk_angle_fp
28 \fp_new:N \l__randomwalk_length_fp

```

(End definition for \l__randomwalk_angle_fp and \l__randomwalk_length_fp.)

`\l__randomwalk_x_dim` `\l__randomwalk_y_dim` Current coordinates: each `\pgfpathlineto` statement goes from the previous value of these to the next. See `__randomwalk_walk_step:`.

```
29 \dim_new:N \l__randomwalk_x_dim
30 \dim_new:N \l__randomwalk_y_dim

(End definition for \l__randomwalk_x_dim and \l__randomwalk_y_dim.)
```

`\l__randomwalk_angles_seq` `\l__randomwalk_lengths_seq` Sequences containing all allowed angles and lengths, as floating point numbers.

```
31 \seq_new:N \l__randomwalk_angles_seq
32 \seq_new:N \l__randomwalk_lengths_seq

(End definition for \l__randomwalk_angles_seq and \l__randomwalk_lengths_seq.)
```

2.3 User command and key-value list

`\RandomWalk` `\randomwalk:n` The user command `\RandomWalk` is based on the code-level command `\randomwalk:n`, which simply does the setup and calls the internal macro `__randomwalk_walk:`.

```
33 \DeclareDocumentCommand \RandomWalk { m }
34   { \randomwalk:n {#1} }
35 \cs_new_protected:Npn \randomwalk:n #1
36   {
37     \__randomwalk_setup_defaults:
38     \keys_set:nn { randomwalk } {#1}
39     \__randomwalk_walk:
40   }
```

(End definition for `\RandomWalk` and `\randomwalk:n`. These functions are documented on page 2.)

We introduce the keys for the package.

```
41 \keys_define:nn { randomwalk }
42   {
43     number .value_required:n = true ,
44     length .value_required:n = true ,
45     angles .value_required:n = true ,
46     number .int_set:N = \l__randomwalk_step_number_int ,
47     length .code:n = { \__randomwalk_setup_length:n {#1} } ,
48     angles .code:n = { \__randomwalk_setup_angles:n {#1} } ,
49     degree .bool_set:N = \l__randomwalk_degrees_bool ,
50     degrees .bool_set:N = \l__randomwalk_degrees_bool ,
51     angles-relative .bool_set:N = \l__randomwalk_relative_angles_bool ,
52     revert-random .bool_set:N = \l__randomwalk_revert_random_bool ,
53   }
```

2.4 Setup

`__randomwalk_setup_defaults:` The package treats the length of steps, and the angle, completely independently. The function `__randomwalk_next_length:` contains the action that decides the length of the next step, while the function `__randomwalk_next_angle:` pertains to the angle.

`__randomwalk_setup_defaults:` sets the default values before processing the user's key-value input. This also sets initial values of variables that currently cannot

be altered through keys, because it might be good to provide keys for their initial values too later on.

```

54 \cs_new_protected_nopar:Npn \__randomwalk_setup_defaults:
55 {
56   \int_set:Nn \l__randomwalk_step_number_int {10}
57   \cs_gset_protected_nopar:Npn \__randomwalk_next_angle:
58     { \__randomwalk_fp_set_rand:Nnn \l__randomwalk_angle_fp { 0 } { 360 } }
59   \cs_gset_protected_nopar:Npn \__randomwalk_next_length:
60     { \fp_set:Nn \l__randomwalk_length_fp {10} }
61   \bool_set_false:N \l__randomwalk_revert_random_bool
62   \bool_set_false:N \l__randomwalk_relative_angles_bool
63   \fp_zero:N \l__randomwalk_angle_fp
64   \fp_zero:N \l__randomwalk_length_fp
65   \dim_zero:N \l__randomwalk_x_dim
66   \dim_zero:N \l__randomwalk_y_dim
67 }

```

(End definition for __randomwalk_setup_defaults:.)

`__randomwalk_setup_length:n` Convert each item in the comma list into a floating point, then define `__randomwalk_next_length:` to set `\l__randomwalk_length_fp` to a random floating point in the list.

```

68 \cs_new_protected:Npn \__randomwalk_setup_length:n #1
69 {
70   \seq_set_split:Nnn \l__randomwalk_lengths_seq { , } {#1}
71   \seq_set_map:NNn \l__randomwalk_lengths_seq
72     \l__randomwalk_lengths_seq { \dim_to_fp:n {##1} }
73   \cs_gset_protected_nopar:Npn \__randomwalk_next_length:
74     {
75       \__randomwalk_get_rand_seq_item:NN
76         \l__randomwalk_lengths_seq \l__randomwalk_internal_tl
77       \fp_set:Nn \l__randomwalk_length_fp { \l__randomwalk_internal_tl }
78     }
79 }

```

(End definition for __randomwalk_setup_length:n.)

`__randomwalk_setup_angles:n` Two complications compared to `__randomwalk_setup_length:n`. First, the angle can be given in radians rather than degrees: then add `rad` after the randomly chosen value (in principle it would be better to convert angles once and for all at the beginning, but that interacts in a complicated way with the fact that keys can be given in any order). Second, angles can be relative, in which case we use `\fp_add:Nn` to take the last angle into account.

```

80 \cs_new_protected:Npn \__randomwalk_setup_angles:n #1
81 {
82   \seq_set_split:Nnn \l__randomwalk_angles_seq { , } {#1}
83   \seq_set_map:NNn \l__randomwalk_angles_seq
84     \l__randomwalk_angles_seq { \fp_to_tl:n {##1} }
85   \cs_gset_protected_nopar:Npn \__randomwalk_next_angle:
86     {

```

```

87     \l__randomwalk_get_rand_seq_item:NN
88     \l__randomwalk_angles_seq \l__randomwalk_internal_tl
89     \bool_if:NF \l__randomwalk_degrees_bool
90     { \tl_put_right:Nn \l__randomwalk_internal_tl { rad } }
91     \bool_if:NTF \l__randomwalk_relative_angles_bool
92     { \fp_add:Nn } { \fp_set:Nn }
93     \l__randomwalk_angle_fp { \l__randomwalk_internal_tl }
94   }
95 }

```

(End definition for `__randomwalk_setup_angles:n`.)

2.5 Drawing

`__randomwalk_walk:` We are ready to define `__randomwalk_walk:`, which draws a `pgf` picture of a random walk with the parameters set up by the `keys`. We reset coordinates to zero originally. Then draw the relevant `pgf` picture by repeatedly calling `__randomwalk_walk_step:`.

```

96 \cs_new_protected_nopar:Npn \__randomwalk_walk:
97   {
98     \__randomwalk_walk_start:
99     \prg_replicate:nn { \l__randomwalk_step_number_int }
100     { \__randomwalk_walk_step: }
101     \bool_if:NF \l__randomwalk_revert_random_bool
102     { \int_gset_eq:NN \cr@nd \cr@nd }
103     \__randomwalk_walk_stop:
104   }

```

`\cr@nd` is internal to the `lcg` package.

(End definition for `__randomwalk_walk:.`)

`__randomwalk_walk_start:` These functions encapsulate all of the `pgf`-related code. The `start` function begins the `pgfpicture` environment and starts a path at position (x,y) . The `line` function adds to the path a line from the previous position to the new (x,y) . The `stop` function draws the path constructed by `__randomwalk_walk_step:` and ends the `pgfpicture` environment.

```

105 \cs_new_protected_nopar:Npn \__randomwalk_walk_start:
106   {
107     \begin{pgfpicture}
108       \pgfpathmoveto
109       { \pgfpoint { \l__randomwalk_x_dim } { \l__randomwalk_y_dim } }
110     }
111 \cs_new_protected_nopar:Npn \__randomwalk_walk_line:
112   {
113     \pgfpathlineto
114     { \pgfpoint { \l__randomwalk_x_dim } { \l__randomwalk_y_dim } }
115   }
116 \cs_new_protected_nopar:Npn \__randomwalk_walk_stop:
117   {
118     \pgfusepath { stroke }
119     \end{pgfpicture}
120   }

```

(End definition for `_randomwalk_walk_start:`, `_randomwalk_walk_line:`, and `_randomwalk_walk_stop:`.)

`_randomwalk_walk_step:` `_randomwalk_walk_step:` calls `_randomwalk_next_length:` and `_randomwalk_next_angle:` to determine the length and angle of the new step. This is then converted to cartesian coordinates and added to the previous end-point. Finally, we call `pgf's \pgfpathlineto` to produce a line to the new point.

```

121 \cs_new_protected_nopar:Npn \_randomwalk_walk_step:
122   {
123     \_randomwalk_next_length:
124     \_randomwalk_next_angle:
125     \dim_add:Nn \l__randomwalk_x_dim
126       {
127         \fp_to_dim:n
128         { \l__randomwalk_length_fp * cosd ( \l__randomwalk_angle_fp ) }
129       }
130     \dim_add:Nn \l__randomwalk_y_dim
131       {
132         \fp_to_dim:n
133         { \l__randomwalk_length_fp * sind ( \l__randomwalk_angle_fp ) }
134       }
135     \_randomwalk_walk_line:
136   }

```

(End definition for `_randomwalk_walk_step:`.)

2.6 On random numbers and items

For random numbers, the interface of `lcg` is not quite enough, so we provide our own $\text{\LaTeX}3$ -y functions. Also, this will allow us to change quite easily our source of random numbers.

`_randomwalk_fp_set_rand:Nnn` We also need floating point random numbers, assigned to the variable `#1`.

```

137 \cs_new_protected:Npn \_randomwalk_fp_set_rand:Nnn #1#2#3
138   {
139     \rand
140     \fp_set:Nn #1
141     { #2 + (#3 - (#2)) * \c@lcg@rand / \c__randomwalk_lcg_last_int }
142   }

```

(End definition for `_randomwalk_fp_set_rand:Nnn.`)

`_randomwalk_get_rand_seq_item:NN` We can now pick an element at random from a sequence. If the sequence has a single element, no need for randomness.

```

143 \cs_new_protected:Npn \_randomwalk_get_rand_seq_item:NN #1#2
144   {
145     \int_set:Nn \l__randomwalk_internal_int { \seq_count:N #1 }
146     \int_compare:nTF { \l__randomwalk_internal_int = 1 }
147     { \tl_set:Nx #2 { \seq_item:Nn #1 { 1 } } }

```



```

148     {
149       \rand
150       \tl_set:Nx #2
151       {
152         \seq_item:Nn #1
153         {
154           1 +
155           \int_mod:nn { \c@lcg@rand } { \l__randomwalk_internal_int }
156         }
157       }
158     }
159   }

```

(End definition for __randomwalk_get_rand_seq_item:NN.)

```

160 </package>

```