The randomwalk package:
customizable random walks∗

Bruno Le Floch†

Released on 2015/09/20

Contents

1 How to use randomwalk 2

2 randomwalk 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
1 How to use 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.

\begin{verbatim}
\input{randomwalk başlayan}
\end{verbatim}

Load pgfcore for figures.

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 \_randomwalk_lcg_last_int = 2^{31} - 2. It will then store them in \_lcg@rand: the \_lcg@ is there because of how \LaTeX\ defines counters. To make it clear that \_lcg has a very special meaning here, I do not follow \LaTeX\ naming conventions. Also of note is that I use \_lcg@ 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.

\begin{verbatim}
\int_const:Nn \_randomwalk_lcg_last_int { \c_max_int - \c_one }
\end{verbatim}
2.2 Variables

\l__randomwalk_internal_tl
\l__randomwalk_internal_int
Used for scratch assignments.
\tl_new:N \l__randomwalk_internal_tl
\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.
\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.
\bool_new:N \l__randomwalk_relative_angles_bool
\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.
\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:
\cs_new_protected_nopar:Npn \__randomwalk_next_angle: { }
\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.
\fp_new:N \l__randomwalk_angle_fp
\fp_new:N \l__randomwalk_length_fp
(End definition for \l__randomwalk_angle_fp and \l__randomwalk_length_fp)
Current coordinates: each \texttt{\pgfpathlineto} statement goes from the previous value of these to the next. See \texttt{\_randomwalk\_walk\_step}.

\dim_new:N \l__randomwalk\_x\_dim
\dim_new:N \l__randomwalk\_y\_dim

(End definition for \l__randomwalk\_x\_dim and \l__randomwalk\_y\_dim.)

\l__randomwalk\_angles\_seq\n\l__randomwalk\_lengths\_seq
Sequences containing all allowed angles and lengths, as floating point numbers.

\seq_new:N \l__randomwalk\_angles\_seq
\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

The user command \texttt{\RandomWalk} is based on the code-level command \texttt{\randomwalk:n}, which simply does the setup and calls the internal macro \texttt{\_randomwalk\_walk}.

\DeclareDocumentCommand \RandomWalk { m } { \randomwalk:n {#1} }
\cs_new_protected:Npn \randomwalk:n #1
{ \__randomwalk\_setup\_defaults:
\keys_set:nn { randomwalk } {#1}
\__randomwalk\_walk:
}

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

We introduce the keys for the package.

\keys_define:nn { randomwalk }
{ number .value_required:n = true ,
length .value_required:n = true ,
angles .value_required:n = true ,
number .int_set:N = \l__randomwalk\_step\_number\_int ,
length .code:n = { \__randomwalk\_setup\_length:n {#1} } ,
angles .code:n = { \__randomwalk\_setup\_angles:n {#1} } ,
dergrees .bool_set:N = \l__randomwalk\_degrees\_bool ,
degrees .bool_set:N = \l__randomwalk\_degrees\_bool ,
angles-relative .bool_set:N = \l__randomwalk\_relative\_angles\_bool ,
revert-random .bool_set:N = \l__randomwalk\_revert\_random\_bool ,
}

### 2.4 Setup

The package treats the length of steps, and the angle, completely independently. The function \texttt{\_randomwalk\_next\_length} contains the action that decides the length of the next step, while the function \texttt{\_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.

\cs_new_protected_nopar:Npn \__randomwalk_setup_defaults:
{\int_set:Nn \l__randomwalk_step_number_int {10} \cs_gset_protected_nopar:Npn \__randomwalk_next_angle:
{ \__randomwalk_fp_set_rand:Nnn \l__randomwalk_angle_fp { 0 } { 360 } } \cs_gset_protected_nopar:Npn \__randomwalk_next_length:
{ \fp_set:Nn \l__randomwalk_length_fp {10} } \bool_set_false:N \l__randomwalk_revert_random_bool \bool_set_false:N \l__randomwalk_relative_angles_bool \fp_zero:N \l__randomwalk_angle_fp \fp_zero:N \l__randomwalk_length_fp \dim_zero:N \l__randomwalk_x_dim \dim_zero:N \l__randomwalk_y_dim }

(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.

\cs_new_protected:Npn \__randomwalk_setup_length:n #1
{\seq_set_split:Nnn \l__randomwalk_lengths_seq { , } {#1} \seq_set_map:NNn \l__randomwalk_lengths_seq \l__randomwalk_lengths_seq { \dim_to_fp:n {##1} } \cs_gset_protected_nopar:Npn \__randomwalk_next_length:
{ \__randomwalk_get_rand_seq_item:NN \l__randomwalk_lengths_seq \l__randomwalk_internal_tl \fp_set:Nn \l__randomwalk_length_fp { \l__randomwalk_internal_tl } }

(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 \texttt{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 \texttt{fp_add:Nn} to take the last angle into account.

\cs_new_protected:Npn \__randomwalk_setup_angles:n #1
{\seq_set_split:Nnn \l__randomwalk_angles_seq { , } {#1} \seq_set_map:NNn \l__randomwalk_angles_seq \l__randomwalk_angles_seq { \fp_to_tl:n {##1} } \cs_gset_protected_nopar:Npn \__randomwalk_next_angle:
{
\_randomwalk_get_rand_seq_item:NN
\_randomwalk_angles_seq \_randomwalk_internal_tl
bool_if:NF \_randomwalk_degrees_bool
{ \_l.put_right:NN \_randomwalk_internal_tl { rad } }
bool_if:NTF \_randomwalk_relative_angles_bool
{ \_fp.add:Nn } { \_fp.set:Nn }
\_randomwalk_angle_fp { \_randomwalk_internal_tl }
}
\_randomwalk_setup_angles:n

(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:.

\cs_new_protected_nopar:Npn \_randomwalk_walk:
{ \_randomwalk_walk_start:
\_randomwalk_walk_line:
\_randomwalk_walk_stop:
}
\_randomwalk_walk_start:
\begin{pgfpicture}
\pgfpathmoveto
{ \pgfpoint { \_randomwalk_x_dim } { \_randomwalk_y_dim } }
\end{pgfpicture}
\_randomwalk_walk_line:
{ \pgfpoint { \_randomwalk_x_dim } { \_randomwalk_y_dim } }
\_randomwalk_walk_stop:
{ \pgfusepath { stroke } }
(End definition for \_randomwalk_walk:)
\__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.

\__randomwalk_fp_set_rand:Nnn

We also need floating point random numbers, assigned to the variable \#1.

\__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.

2.6 On random numbers and items

For random numbers, the interface of lcg is not quite enough, so we provide our own \EFP\X3-y functions. Also, this will allow us to change quite easily our source of random numbers.
\begin{verbatim}
\{ \text{rand} \\tl_set:Nx \text{#2} \{ \seq_item:Nn \text{#1} \{ 1 + \\int_mod:nn \{ \text{\texttt{l@lcg@rand} } \} \{ \text{l\_randomwalk\_internal\_int} \} \}
\}
\end{verbatim}

(End definition for \texttt{l\_randomwalk\_get\_rand\_seq\_item:NN.)

\texttt{(/package)}