# pdiag.sty – A LaTeX $2_\varepsilon$ Package for Drawing Permutation Diagrams

Jason B. Hill

Febriary 25, 2009

**Instructions:** Please download `pdiag.tex` and `pdiag.sty`, save them to the same directory and attempt to compile. With any luck, this document should appear.

This document serves as both a testing platform and supporting documentation for the `pdiag` package. This package is designed considering several goals:

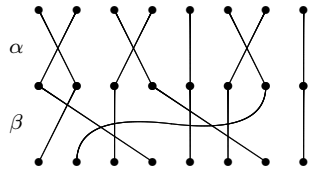
- The ability to typeset permutation diagrams quickly.
    1. The command structure within the package is intuitive and fast to write.
    2. 2-cycle permutation maps are drawn with a single command.
    3. Vertices unaffected by a given map are managed by the package.
    4. Diagrams can be scaled quickly and used in math environments.
- Results independent of `dvi` or `pdf` production paths.
    1. Thus far, results are consistent between `latex (dvi)`, `dvips (ps)` and `pdflatex (pdf)`. For systems where more than one of these environments is available, this package is an improvement over packages that require postscript processing to display graphics.
- The ability to modify lines between vertices with more control.
    1. By default, a straight line will be drawn.
    2. A bezier curve specified by 5 points can be drawn and an optional argument offers the ability to alter this bezier's path with a high amount of control.

# Contents



# 1 Notation

The following is a permutation diagram.




This diagram has two component maps, each on 8 vertices. The map label $\alpha$ applies to the first map and the label $\beta$ applies to the second map. All of the lines are straight in this diagram except for the bezier from vertex 7 to vertex 2 in $\beta$.

## 2 The pdiag Environment

A permutation diagram can be initiated by knowing the dimensions of the diagram (how many vertices and how many maps). A diagram may be started without knowing the full dimension of the diagram, but may not display or align property in the resulting typeset document until this dimension information is corrected. To initiate a diagram, use the `pdiag` environment. (All commands and environments, as well as included variables, in this package begin with `pd` to avoid any conflicts.)

The following code will generate a diagram with 5 vertices and 2 maps.

```
\[
    \begin{pdiag}{5}{2}
    \end{pdiag}
\]
```

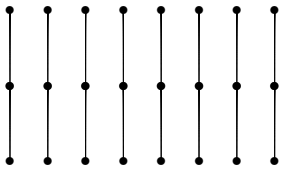
## 3 The pdendmap and pdendmapfill Commands

Before discussing how to create maps between vertices, we introduce the commands that terminate maps. The idea when entering map information is as follows: We will start with the top map in the diagram. When all information has been entered for that map, we will terminate that map and begin entering information for the next map downward in the diagram.

`pdendmap` – Terminate the current map, ignoring unmapped vertices.

`pdendmapfill` – Terminate the current map, and for any vertex that has not been explicitly permuted, draw a line down.

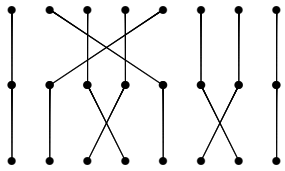
The identity map applied twice on 8 vertices is then as follows.



```
\[
\begin{pdiag}{8}{2}
    \pdendmapfill
    \pdendmapfill
\end{pdiag}
\]
```

# 4 The pdtrans Command for Transpositions

Mapping a single transposition is straightforward with the `pdtrans` command. This command takes 3 arguments (1 optional, discussed later). The required arguments are the numbers of the two vertices to permute.

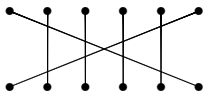


```
\[
\begin{pdiag}{8}{2}
    \pdtrans{2}{5}\pdendmapfill
    \pdtrans{3}{4}\pdtrans{6}{7}\pdendmapfill
\end{pdiag}
\]
```

# 5 The pdbraid Command for Transpositions

The `pdbraid` command is identical to the `pdtrans` command with the exception that it requires a single argument. A braid will create a permutation between two vertices numbered $i$ and $i+1$ where $i+1$ is considered modulo the number of vertices in the diagram.

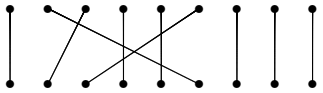
```
\[
\begin{pdiag}{6}{1}
    \pdbraid{2}\pdendmapfill
\end{pdiag}
\]
```



```
\[
\begin{pdiag}{6}{1}
    \pdbraid{6}\pdendmapfill
\end{pdiag}
\]
```

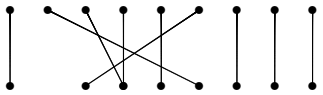
# 6 The pdmap Command for Permutations

The pdmap command has two arguments required, corresponding to the vertices of the domain and image of the current map.



```
\[
\begin{pdiag}{9}{1}
    \pdmap{2}{6}\pdmap{6}{3}\pdmap{3}{2}\pdendmapfill
\end{pdiag}
\]
```

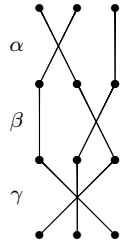
The variables stored to make the pdendmapfill command possible (automatic drawing of vertical lines for unmapped vertices) are only considered for vertices mapped from the domain. I.e., no consideration is placed on vertices that are or are not mapped to in the image. Thus, when using the pdmap command, one must keep this in mind so that situations like the following do not arise.



```
\[
\begin{pdiag}{9}{1}
    \pdmap{2}{6}\pdmap{6}{3}\pdmap{3}{4}\pdendmapfill
\end{pdiag}
\]
```

# 7 The pdname Command to Name a Map

The pdiag environment is enclosed in an array with a small amount of space on the left and right sides. By calling the pdname command, a label for the current map is placed to the left side of the map in this space.



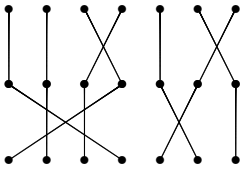

```
\[
\begin{pdiag}{3}{3}
```

```
    \pdname{\alpha}\pdbraid{1}\pdendmapfill
    \pdname{\beta}\pdbraid{2}\pdendmapfill
    \pdname{\gamma}\pdtrans{1}{3}\pdendmapfill
\end{pdiag}
\]
```

# 8 Optional Arguments

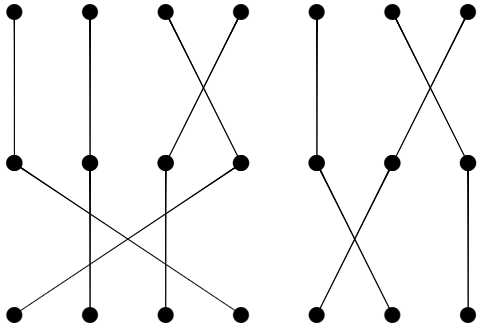
## 8.1 Scaling

An optional numerical argument in the `pdiag` environment allows for scaling of diagrams. The default value for this setting is 1, with 1 corresponding to a gap of 0.5 cm between vertices horizontally and 1.0 cm vertically. Notice the differences in the following diagrams, which are identical with the exception of their scale factors.



```
\[
\begin{pdiag}{7}{2}
    \pdbraid{3}\pdbraid{6}\pdendmapfill
    \pdtrans{1}{4}\pdbraid{5}\pdendmapfill
\end{pdiag}
\]
```

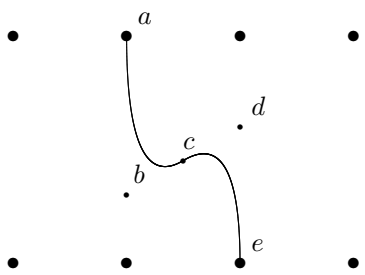
```
\[
\begin{pdiag}[0.5]{7}{2}
    \pdbraid{3}\pdbraid{6}\pdendmapfill
    \pdtrans{1}{4}\pdbraid{5}\pdendmapfill
\end{pdiag}
\]
```



```
\[
\begin{pdiag}[2]{7}{2}
    \pdbraid{3}\pdbraid{6}\pdendmapfill
    \pdtrans{1}{4}\pdbraid{5}\pdendmapfill
\end{pdiag}
\]
```

## 8.2 Bezier Curves

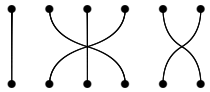
Any of the commands used to draw maps can take an optional argument to draw a bezier curve instead of a straight line. A bezier is drawn by considering 5 points in the plane between the domain vertex and image vertex. An example of the situation is shown here.




The points $a$ and $e$ are given as vertices, and so we know their coordinates. The point $c$ is the midpoint of points $b$ and $d$. Points $b$ and $d$ have the same $x$-coordinates as $a$ and $e$, respectively. Thus, the bezier is completely determined by the $y$-coordinates of $b$ and $d$. We therefore develop a standard for the required $y$-coordinates within a given map.

We consider the vertical midpoint between $a$ and $e$ to be the origin, with $a$ at -1 and $e$ at 1. (This may seem somewhat backwards at first, but consider the bezier as traveling from $a$ to $e$, from -1 to 1, and it makes sense.) We set the default value for the $y$-coordinate of $b$ to -0.3, and the default value for the $y$-coordinate of $e$ to 0.3.

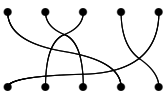
To initiate a bezier curve, place the optional argument [1] inside any of the mapping commands. This will draw a bezier with default $y$-coordinates calculated for points $b$ and $d$ in the above diagram.



```
\[
\begin{pdiag}{6}{1}
    \pdtrans[1]{2}{4}
    \pdbraid[1]{5}
    \pdendmapfill
\end{pdiag}
\]
```

### 8.2.1 Modifying a Bezier with the `pdmbez` Command

The `pdmbez` command takes two arguments, corresponding to the $y$-coordinates of the points $b$ and $d$ discussed above. The first argument corresponds to the $y$-coordinate of $b$, while the second argument corresponds to the $y$-coordinate of $d$. The values do not need to be between -1 and 1, but values outside this range can force the bezier outside of the current map's area. The default bezier $y$-coordinate values are reset upon completion of the next mapping command. When the `pdmbez` command is used to modify the bezier of the `pdbraid` command, the two resulting lines are mirror images of each other. Always remember to supply the optional [1] argument to the mapping command being used, or a straight line (not a bezier) will be drawn.

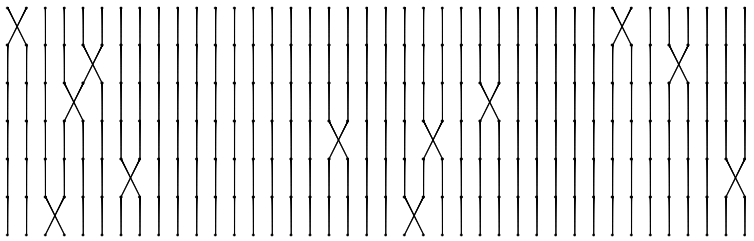


```
\[
\begin{pdiag}{5}{1}
    \pdmbez{-0.2}{0.3}\pdmap[1]{1}{4}
    \pdmbez{-0.1}{0.6}\pdmap[1]{4}{5}
    \pdmbez{-0.6}{-0.2}\pdtrans[1]{2}{3}
    \pdmbez{0.6}{0.7}\pdmap[1]{5}{1}
\end{pdiag}
\]
```

Experimenting with the `pdmbez` command is probably the best way to learn it.

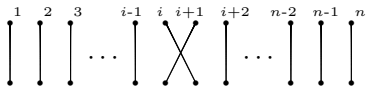
# 9 Examples

The environment is limited to 50 vertices per map at present. The following example may require some processing time (floating point calculations required to calculate points for the diagram are highly inefficient in LaTeX $2_\varepsilon$).



```
\[
\begin{pdiag}[0.5]{40}{6}
\pdbraid{1}\pdbraid{33}\pdendmapfill
\pdbraid{5}\pdbraid{36}\pdendmapfill
\pdbraid{4}\pdbraid{26}\pdendmapfill
\pdbraid{18}\pdbraid{23}\pdendmapfill
\pdbraid{7}\pdbraid{39}\pdendmapfill
\pdbraid{3}\pdbraid{22}\pdendmapfill
\end{pdiag}
\]
```

We could use diagrams to define the generators of the symmetric groups:

$\sigma_i :=$ 


```
\[
\sigma_i:=\begin{pdiag}[0.8]{3}{1}
    \put(0.6,2.2){{\tiny 1}}\put(1.6,2.2){{\tiny 2}}\put(2.6,2.2){{\tiny 3}}
    \pdendmapfill
\end{pdiag}\hspace{-0.5pc}\cdots\hspace{-0.5pc}\begin{pdiag}[0.8]{4}{1}
    \put(0.0,2.2){{\tiny $i$-1}}\put(1.2,2.2){{\tiny $i$}}
    \put(1.8,2.2){{\tiny $i$+1}}\put(3.3,2.2){{\tiny $i$+2}}
    \pdbraid{2}
    \pdendmapfill
\end{pdiag}\hspace{-0.5pc}\cdots\hspace{-0.5pc}\begin{pdiag}[0.8]{3}{1}
    \put(-0.2,2.2){{\tiny $n$-2}}\put(1.2,2.2){{\tiny $n$-1}}\put(2.6,2.2){{\tiny $n$}}
    \pdendmapfill
\end{pdiag}
\]
```