The "pavemat" package

v0.2.0 MIT

Style matrices with custom paths, strokes and fills for appealing visualizations.

QuadnucYard

Pavemat provides a simple function `pavemat` for creating styled matrices with customizable paths, strokes, and fills. It allows users to define how paths should be drawn through the matrix, apply different strokes to these paths, and fill specific cells with various colors. This can be used for visualizing data structures and matrices, and creating custom grid layouts.

Table of Contents

Part I

Getting Started

I.1 Quick Start

To use Pavemat in your document, first import the package:

```
1 #import "@preview/pavemat:0.2.0": pavemat
```

Then create a basic matrix with paths:

```
#let matrix = $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9) $
#pavemat(
  matrix,
  pave: "DDSSAAW",
  fills: ("1-1": red.transparentize(80%))
)
```

$$\left(\begin{array}{cc|c} 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \end{array}\right)$$

I.2 Basic Examples

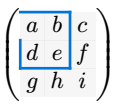
Highlighting Matrix Elements

```
#let matrix = $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9) $
#pavemat(
  matrix,
  fills: (
    "0-0": red.transparentize(80%),      // Connected region starting at (0,0)
    "[1-1]": blue.transparentize(80%),   // Only single cell (1,1)
    "[2-2]": green.transparentize(80%),  // Only single cell (2,2)
  )
)
```

$$\left(\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}\right)$$

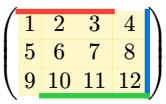
Drawing Paths Through Data

```
#let matrix = $ mat(a, b, c; d, e, f; g, h, i) $
#pavemat(
  matrix,
  pave: "DDSSAAW",
  stroke: blue + 1pt,
  fills: ("": gray.transparentize(90%))
)
```



Multiple Styled Paths

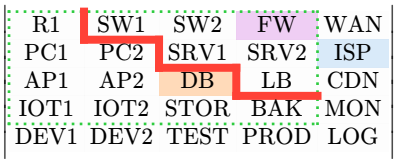
```
#let matrix = $ mat(1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12) $
#pavemat(
  matrix,
  pave: (
    (path: "DDD", stroke: red + 2pt),
    (path: "SSS", from: "top-right", stroke: blue + 2pt),
    (path: "AAA", from: "bottom-right", stroke: green + 2pt),
  ),
  fills: ("1-1": yellow.transparentize(80%))
)
```



I.3 Advanced Examples

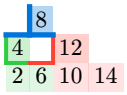
Complex Path Combinations

```
// Network topology visualization
#let network = $ mat(
  "R1", "SW1", "SW2", "FW", "WAN";
  "PC1", "PC2", "SRV1", "SRV2", "ISP";
  "AP1", "AP2", "DB", "LB", "CDN";
  "IOT1", "IOT2", "STOR", "BAK", "MON";
  "DEV1", "DEV2", "TEST", "PROD", "LOG"
) $

#pavemat(
  network,
  pave: (
    // Main data path (red backbone)
    (path: "DSDSDSDS", stroke: red + 3pt),
    // Backup path (blue failover)
    (path: "ADADADAD", from: (0, 4), stroke: (paint: blue, thickness: 2pt, dash: "dashed")),
    // Network perimeter (green security boundary)
    (path: "DDDDSSSSAAAAWWWW", stroke: (paint: green, thickness: 1pt, dash: "dotted"))
  ),
  fills: (
    "": gray.transparentize(98%),
    "[2-2]": orange.transparentize(70%),  // Critical database server
    "[0-3]": purple.transparentize(80%),  // Firewall
    "[1-4]": blue.transparentize(85%)     // Internet connection
  ),
  delim: "|"
)
```



Data Structure Visualization

```
// Binary search tree with node relationships
#let bst = $ mat(
  "", 8, "", "";
  4, "", 12, "";
  2, 6, 10, 14
) $

#pavemat(
  bst,
  pave: (
    // Root to left subtree
    (path: "SA", from: (0, 1), stroke: blue + 2pt),
    // Root to right subtree
    (path: "SD", from: (0, 1), stroke: blue + 2pt),
    // Left subtree internal connections
    (path: "S", from: (1, 0), stroke: green + 1.5pt),
    (path: "SD", from: (1, 0), stroke: green + 1.5pt),
    // Right subtree internal connections
    (path: "SA", from: (1, 2), stroke: red + 1.5pt),
    (path: "S", from: (1, 2), stroke: red + 1.5pt)
  ),
  fills: (
    "[0-1]": blue.transparentize(70%),    // Root node (8)
    "[1-0]": green.transparentize(75%),   // Left subtree root (4)
    "[1-2]": red.transparentize(75%),     // Right subtree root (12)
    "[2-0]": green.transparentize(85%),   // Leaf (2)
    "[2-1]": green.transparentize(85%),   // Leaf (6)
    "[2-2]": red.transparentize(85%),     // Leaf (10)
    "[2-3]": red.transparentize(85%)      // Leaf (14)
  ),
  delim: none,
)
```



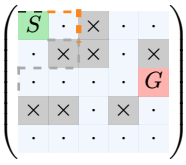
Mathematical Annotations

```
// Gaussian elimination steps for system solving
#let system = $ mat(
  2, -1, 3, 7;
  1, 2, -1, 4;
  3, 1, 2, 10
) $

#pavemat(
  system,
  pave: (
    // Vertical separator before last column
    (path: "SSS", from: (0, 3), stroke: black + 0.5pt)
  ),
  fills: (
    "0-0": yellow.transparentize(90%),    // Elimination row (base layer)
    "[0-0]": red.transparentize(80%),     // Pivot element (override)
    "[0-3]": blue.transparentize(80%),    // Target value
    "[1-3]": blue.transparentize(80%),    // Target value
    "[2-3]": blue.transparentize(80%)     // Target value
  ),
  delim: "["
)
```

$$\left[\begin{array}{ccc|c} 2 & -1 & 3 & 7 \\ 1 & 2 & -1 & 4 \\ 3 & 1 & 2 & 10 \end{array}\right]$$

Algorithm Visualization

```
// Pathfinding algorithm trace (A* search)
#let pathgrid = $ mat(
  S, ·, ×, ·, ·;
  ·, ×, ×, ·, ×;
  ·, ·, ·, ·, G;
  ×, ×, ·, ×, ·;
  ·, ·, ·, ·, ·
) $

#pavemat(
  pathgrid,
  pave: (
    // Optimal path with varying confidence levels
    (path: "D(thickness: 1pt, paint: orange)D(thickness: 2pt, paint: orange)S(thickness: 3pt,
paint: green)S(thickness: 3pt, paint: green)"),
    // Explored nodes (backtracking paths)
    (path: "DSA", from: (1, 1), stroke: (paint: gray, thickness: 1pt, dash: "dashed")),
    (path: "AS", from: (2, 1), stroke: (paint: gray, thickness: 1pt, dash: "dashed"))
  ),
  fills: (
    "": blue.transparentize(95%),          // Open space (base layer)
    "[0-2]": gray.transparentize(40%),    // Obstacles (override base)
    "[1-1]": gray.transparentize(40%),
    "[1-2]": gray.transparentize(40%),
    "[1-4]": gray.transparentize(40%),
    "[3-0]": gray.transparentize(40%),
    "[3-1]": gray.transparentize(40%),
    "[3-3]": gray.transparentize(40%),
    "[0-0]": green.transparentize(60%),    // Start node (top priority)
    "[2-4]": red.transparentize(60%)       // Goal node (top priority)
  )
)
```



Part II
Syntax

II.1 Overview

Pavemat uses two main concepts to style matrices:

- **Pave strings**: Define paths through the matrix using directional characters
- **Position strings**: Specify which cells to fill or target

II.2 Pave string

A pave string defines paths through the matrix. This string consists of directional characters that guide the path from one cell to another. It can also include styled segments and custom control characters for more advanced configurations.

II.2.1 Basic Directional Characters

A pave string primarily uses the following basic directional characters to move through the matrix:

- `W`, `w`: Move up
- `S`, `s`: Move down
- `A`, `a`: Move left
- `D`, `d`: Move right

If a *lowercase* letter is used, this segment will *not* be displayed (hidden path).

Basic Examples:

```
// Matrix border with corner highlights
#pavemat(
  $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9) $,
  pave: "DDSSAAWW",  // Complete border
)
```

$$\begin{pmatrix} \overline{1} & \overline{2} & 3 \\ 4 & 5 & 6 \\ \underline{7} & \underline{8} & 9 \end{pmatrix}$$

```
// Data flow visualization
#pavemat(
  $ mat(a, b, c, d; e, f, g, h; i, j, k, l) $,
  pave: "DDDSSAAA",  // Flow from top-left to bottom-right
  stroke: (paint: green, thickness: 2pt),
)
```

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}$$

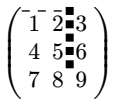
```
// Hidden segments for complex routing
#pavemat(
  $ mat(α, β, γ; δ, ε, ζ; η, θ, ι) $,
  pave: "DdSs",  // Diagonal pattern with hidden segments
  stroke: purple + 2pt,
  debug: true,  // Shows hidden segments in gray
)
```

II.2.2 Styled Segments

To apply styles to specific segments of the path, you can enclose the style specifications in parentheses. These styles, **which affect only the subsequent segment until the closing parenthesis or the end of the segment**, can include changes to stroke properties such as dash pattern, color, and thickness. The style is evaluated as dictionary and then passed to `grid.hline.stroke` and `grid.vline.stroke`. **Styles can be overlaid.**

Basic Style Examples:

```
// Thick dotted line
#pavemat(
  $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9) $,
  pave: "DD(thickness: 3pt, dash: 'dotted')SS"
)
```



Advanced Style Examples:

- Solid Stroke: `"SS(dash: 'solid')DDD"`
- Colored Stroke: `"SS(paint: red)DDD"`
- Thickness and Dash: `` `SS(thickness: 2pt, dash: "dotted")DDD`.text ``

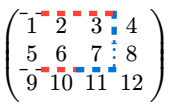
Single quotes (`'`) in the string will be simply replaced by double quotes (`"`), so there is no need to escape them.

In the example `"SS(dash: 'solid')DDD"`, the path first moves down twice and then uses a solid stroke for the next segment that moves right three times.

The style has its scope, ended by a right bracket `]`. You can optionally add one `[` after style parenthesis `)`, which is just ignored.

Complex Style Scoping Example:

```
#pavemat(
  $ mat(1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12) $,
  pave: "D(paint: red, thickness: 2pt)DD(paint: blue)S(thickness: 1pt, dash: 'dotted')S]A]A]A"
)
```



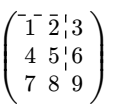
Explanations:

- `D`:
  - Moves right once (`D`) from starting position (0,0) to (0,1).
- `(paint: red, thickness: 2pt)DD`:
  - The stroke is red and 2pt thick.
  - Moves right twice (`DD`) from (0,1) to (0,3).
- `(paint: blue)S`:
  - The stroke changes to blue, while thickness is still 2pt.
  - Moves down (`S`) from (0,3) to (1,3).
- `(thickness: 1pt, dash: 'dotted')S]`:
  - The stroke changes to 1pt thick with a dotted pattern.
  - Moves down (`S`) from (1,3) to (2,3).
  - When encountering `]`, the style resets to the previous state (blue stroke).
- `A]A]A`:
  - Moves left (`A`) with blue stroke from (2,3) to (2,2).
  - Style resets again with `]`, moves left (`A`) from (2,2) to (2,1).
  - Style resets again with `]`, moves left (`A`) from (2,1) to (2,0).

II.2.3 Custom Direction Characters

If you prefer `UDLR` to `WASD`, you can customize control characters through arguments:

```
#pavemat(
  $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9) $,
  pave: "RRDDUULL",
  dir-chars: (up: "U", down: "D", left: "L", right: "R")
)
```



II.2.4 Starting Positions

By default, paths start from the top-left corner, which is `(0, 0)`. You can specify different starting positions:

Important: The `from` parameter uses **grid coordinates** (intersection points), not cell coordinates. For an $n \times m$ matrix:

- Cell coordinates range from $(0, 0)$ to $(n - 1, m - 1)$
- Grid coordinates range from $(0, 0)$ to (n, m)

```
#pavemat(
  $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9) $,
  pave: (
    (path: "DD", from: "top-left"),     // Grid point (0,0)
    (path: "SS", from: "top-right"),    // Grid point (0,3)
    (path: "AA", from: "bottom-right"), // Grid point (3,3)
    (path: "WW", from: "bottom-left")   // Grid point (3,0)
  ),
  stroke: blue + 1pt
)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

You can also use numeric grid coordinates:

```
#pavemat(
  $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9) $,
  pave: (path: "WASSDDS", from: (1, 1)),  // Start from grid point (1,1)
  stroke: red + 2pt
)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

II.3 Fill Position

A position string is formatted as `"{x}-{y}"` or `"[{x}-{y}]"`. Examples include `"0-1"`, `"top-left"`, `"bottom-2"`, and `"[2-right]"`.

- `{x}`: Can be an integer or the literal `top` or `bottom`.
- `{y}`: Can be an integer or the literal `left` or `right`.

In the `fills` parameter, position strings determine which cells to fill with the specified color.

Key Difference:
- **Without brackets** `"x-y"`: Uses flood fill algorithm to fill the entire connected region starting from cell (x, y). Cells are considered connected if there's no path line blocking them.
- **With brackets** `"[x-y]"`: Fills only the single, specific cell at position (x,y).

Named positions like `"top-left"`, `"bottom-right"` work the same as coordinate positions and also use flood fill unless enclosed in brackets.

Position Examples:

```
// Demonstrating flood fill vs single cell fill
#pavemat(
  $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9) $,
  pave: "DDSS",  // Creates barriers that separate regions
  fills: (
    "0-0": red.transparentize(80%),     // Flood fill: fills entire connected region
    "[2-2]": blue.transparentize(80%),  // Single cell: fills only this one cell
  )
)
```

```
// Fill connected regions
#pavemat(
  $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9) $,
  pave: "DDSSAA",  // Creates barriers that separate regions
  fills: (
    "0-0": red.transparentize(80%),     // Connected region starting at (0,0)
    "[1-1]": blue.transparentize(80%),  // Only cell (1,1)
    "bottom-right": green.transparentize(80%)  // Connected region at bottom-right
  )
)
```

```
// Global fill with specific overrides (order matters!)
#pavemat(
  $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9) $,
  pave: "DDSSAAW",
  fills: (
    "": gray.transparentize(90%),       // Global fill (base layer)
    "[0-0]": red.transparentize(80%),   // Override single cell
    "1-1": blue.transparentize(80%)     // Override connected region (applied last)
  )
)
```

Important: Fill order matters! Later fills override earlier ones, so place base fills first and specific overrides last.

Part III

API Reference

III.1 pavemat

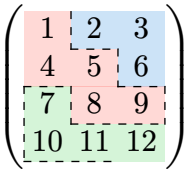
```
#pavemat(
  ⟨eq⟩,
  ⟨pave⟩: (),
  ⟨stroke⟩: (dash: "dashed", thickness: 0.5pt),
  ⟨fills⟩: (:),
  ⟨dir-chars⟩: (:),
  ⟨delim⟩: auto,
  ⟨block⟩: auto,
  ⟨display-style⟩: true,
  ⟨debug⟩: false
) → content
```

Create a *pavemat* from [math.mat] or [math.equation].

For details on the syntax of pave strings and position specifications, please refer to the manual.

Example:

```
#pavemat(
  $ mat(1, 2, 3; 4, 5, 6; 7, 8, 9; 10, 11, 12) $,
  pave: "dSDSDSLLAAWASSDD",
  fills: (
    "1-1": red.transparentize(80%),
    "1-2": blue.transparentize(80%),
    "3-0": green.transparentize(80%),
  ),
)
```



Argument

⟨eq⟩ — `equation` | `mat` | `array`

haha The input matrix expression to be styled. It can be a mathematical equation or a matrix. Specifically,

- A math.equation. It should contain only a math.mat as its body. Example: $mat(1, 2; 3, 4)$. math.display in the equation is not properly supported yet.

- A `math.mat`. Example: `math.mat((1, 2), (3, 4))`.
- A nested array. Example: `((1, 2), (3, 4))`.

If a matrix type is given, pavemat will use its style, including `row-gap`, `column-gap` and `delim`.

Argument

⟨pave⟩: `()` `str` | `dictionary` | `array`

Describes the pavement lines. It accepts the following formats:
- A path string like `"WASD"`.
- A dictionary with fields: `path`, `from` (optional, default: `"top-left"`), `stroke` (optional, default: empty). The path is a pave string.
- An array whose item type is either string or dictionary described above.

Argument

⟨stroke⟩: `(dash: "dashed", thickness: 0.5pt)` `stroke`

The global stroke style applying to all segments. This argument will be passed to `cell.stroke`. Accepts anything can be used as stroke. Examples: `blue + 1pt`, `(dash: "dashed", thickness: 0.5pt)`.

Argument

⟨fills⟩: `(:)` `dictionary`

Specifies the fill colors for specific cells. The key represents a position, and the value is the color passed to `cell.fill`. An empty key `""` is used for global fill.

Argument

⟨dir-chars⟩: `(:)` `dictionary`

Controls whether the output is in display style. Its fields will override the default `(up: "W", down: "S", left: "A", right: "D")`. Example: `(up: "U", down: "D", left: "L", right: "R")`

Argument

⟨delim⟩: `auto` `auto` | `any`

The delimiter of the matrix. If set to `auto`, it uses the delimiter of the input matrix.

Argument

⟨block⟩: `auto` `auto` | `bool`

Controls whether the output is block-style. If set to `auto`, it uses the block setting of the input equation.

Argument

⟨display-style⟩: `true` `bool`

Controls whether the output is in display style. If set to `true`, the result will be granted a `math.display(...)`.

Argument

`(debug): false` `bool`

Controls whether the output is block-style. If set to `auto`, it uses the block setting of the input equation.

Part IV

Index

P