Python for Mathematical Visualization

A Four-Dimensional Case Study

David Dumas
University of Illinois at Chicago

May 20, 2017
Goal

Describe a mathematical visualization project that uses Python.

Highlight characteristics of Python that make it suitable for this application.
The PML Visualization Project

Joint work with François Guéritaud

http://dumas.io/PML
Steps

- Enumerate the objects
- Calculate coordinates
- Render an image
Topological enumeration
Topological enumeration
Topological enumeration
Topological enumeration
Topological enumeration
Topological enumeration
Topological enumeration
Topological enumeration

Not a simple path
Topological enumeration
Topological enumeration

Simple path space

START

FINISH
Simple loops on the five-holed sphere
Topological enumeration

Simple loops on the five-holed sphere
Simple loops on the five-holed sphere
Simple loops on the five-holed sphere
Topological enumeration

Simple loops on the five-holed sphere
Topological enumeration

Simple loops on the five-holed sphere
Topological enumeration

Simple loops on the five-holed sphere
Topological enumeration

Simple loops on the five-holed sphere
Simple loops on the five-holed sphere
Topological enumeration

Simple *loops* on the five-holed sphere
Simple loops on the five-holed sphere
Topological enumeration

Simple loops on the five-holed sphere
Nonperipheral simple closed curves on the five-holed sphere
Nonperipheral simple closed curves on the five-holed sphere (NSCC)
Topological enumeration

Nonperipheral simple closed curves on the five-holed sphere (NSCC)
Nonperipheral simple closed curves on the five-holed sphere (NSCC)
Nonperipheral simple closed curves on the five-holed sphere (NSCC)
Topological enumeration

Nonperipheral simple closed curves on the five-holed sphere (NSCC)
Topological enumeration

Nonperipheral simple closed curves on the five-holed sphere (NSCC)
Topological enumeration

Nonperipheral simple closed curves on the five-holed sphere (NSCC)
Nonperipheral simple closed curves on the five-holed sphere (NSCC)
Nonperipheral simple closed curves on the five-holed sphere (NSCC)
Nonperipheral simple closed curves on the five-holed sphere (NSCC)
Generators
Generators

\[ dumas.io/PML \]
Generators
Generators

da
b
c

dumasi.io/PML
Generators
Generators
Generators

da
b
c
d
e
Encoding curves in strings
Encoding curves in strings

ab
Encoding curves in strings

ab
Encoding curves in strings

ab
Encoding curves in strings

ab

dumas.io/PML
Encoding curves in strings

bc
Encoding curves in strings

ac
Encoding curves in strings

CA

dumas.io/PML
Encoding curves in strings
Encoding curves in strings

\[ \text{x} \]

\[ c \]

\[ b \]

\[ d \]

\[ e \]
Encoding curves in strings

abDcd
Naïve generator

All words in alphabet 'abcdeABCDE'.

```python
from itertools import product

N = 5  # Word length

for w in product('abcdeABCDE', repeat=N):
    print(''.join(w))
```
Naïve generator

Simple words in alphabet 'abcdeABCDEFGHIJKLMNOPQRSTUVWXYZ'.

```python
from itertools import product

N = 5  # Word length

for w in product('abcdeABCDEFGHIJKLMNOPQRSTUVWXYZ', repeat=N):
    if is_simple_curve(w):
        print(''.join(w))
```
Naïve generator

Simple words in alphabet 'abcdeABCDE'.

```python
from itertools import product

N = 5  # Word length

for w in product('abcdeABCDE', repeat=N):
    if is_simple_curve(w):
        print('',.join(w))

# TODO: Implement is_simple_curve()
```
Exponential haystack

aa, ab, ac, ad, ae, aB, aC, aD, aE, ba,
bb, bc, bd, be, bA, bC, bD, bE, ca, cb,
cC, cd, ce, cA, cB, cD, cE, da, db, dc,
dd, de, dA, dB, dC, dE, ea, eb, ec, ed,
ee, eA, eB, eC, eD, Ab, Ac, Ad, Ae, AA,
AB, AC, AD, AE, Ba, Bc, Bd, Be, BA, BB,
BC, BD, BE, Ca, Cb, Cd, Ce, CA, CB, CC,
CD, CE, Da, Db, Dc, De, DA, DB, DC, DD,
DE, Ea, Eb, Ec, Ed, EA, EB, EC, ED, EE
Exponential haystack
Twisting

cd

bc
Twisting

bc
cd

dumas.io/PML
Twisting

bc → ?

cd

dumas.io/PML
Twisting

bc → ?

dumias.io/PML
Twisting

bc → ?

cd
Twisting

bc → bd

$T_{cd}$
Twisting

$bc \rightarrow bd \rightarrow bDcd$

$T^2_{cd}$
Twisting is search & replace

$T_{cd}$ is equivalent to substituting:

\begin{align*}
  a & \rightarrow a \\
  b & \rightarrow b \\
  c & \rightarrow d \\
  d & \rightarrow Dcd \\
  e & \rightarrow e
\end{align*}
Twisting is search & replace

<table>
<thead>
<tr>
<th></th>
<th>$T_{ab}$</th>
<th>$T_{bc}$</th>
<th>$T_{cd}$</th>
<th>$T_{de}$</th>
<th>$T_{ea}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$\rightarrow$ b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>Aea</td>
</tr>
<tr>
<td>b</td>
<td>$\rightarrow$ Bab</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>$\rightarrow$ c</td>
<td>Cbc</td>
<td>d</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>$\rightarrow$ d</td>
<td>d</td>
<td>Dcd</td>
<td>e</td>
<td>d</td>
</tr>
<tr>
<td>e</td>
<td>$\rightarrow$ e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>Ede</td>
</tr>
</tbody>
</table>
Fact: Any nonperipheral simple closed curve can be transformed to any other one by applying twists $T_{ab}, T_{bc}, T_{cd}, T_{de}, T_{ea}$ and their inverses.
Twist-based generator

seed curves
Twist-based generator

twists $\Rightarrow$

seed curves
Twist-based generator

twists $\Rightarrow$

frontier
Twist-based generator

twists \implies \text{frontier}
Twist-based generator

twists \iff

frontier
Twist-based generator
Twist-based generator
Twist-based generator
Twist-based generator
Twist-based generator
Twist-based generator
Twist-based generator
Twist-based generator
Twist-based generator

Python’s `set` type stores collections of distinct elements and supports efficient boolean operations.

def depth = 5
twists = { Tab, Tab_inv, Tbc, Tbc_inv } # etc
curves = set()
frontier = {'ab', 'bc', 'cd', 'de', 'ea'}

for _ in range(depth):
    latest = \
        { T(x) for x in frontier for T in twists }
    frontier = latest.difference(curves)
curves.update(frontier)
How to visualize the entire set of NSCCs?
Assign coordinates.
How to visualize the entire set of NSCCs?
Assign coordinates.

In 1986, W. P. Thurston described a way to associate a 4-dimensional vector to each NSCC on the five-holed sphere.
Coordinates

How to visualize the entire set of NSCCs?
Assign coordinates.

In 1986, W. P. Thurston described a way to associate a 4-dimensional vector to each NSCC on the five-holed sphere.

The resulting cloud of points densely fills a 3-dimensional hypersurface in 4-space. This hypersurface is \( PML \).
Length & dlength

Curve $x \rightarrow$ matrix $\varphi(x) \rightarrow$ length $L(x)$

Tiling
Length & dlength

Curve $x$ $\rightarrow$ matrix $\varrho(x)$ $\rightarrow$ length $L(x)$

\[ \varrho(a) = \begin{pmatrix} 1 & 5.50 \\ 0 & 1 \end{pmatrix} \]

\[ \varrho(b) = \begin{pmatrix} 3.62 & 3.60 \\ -1.90 & -1.62 \end{pmatrix} \]

\[ \vdots \]

\[ \varrho(abcB) = \varrho(a)\varrho(b)\varrho(c)\varrho(b)^{-1} \]

\[ = \begin{pmatrix} p & q \\ r & s \end{pmatrix} \]

\[ L(abcB) = 2 \arccosh \left( \frac{1}{2}(p + s) \right) = 6.467 \]
Length & dlength

Curve $x \rightarrow$ matrix $\varphi(x) \rightarrow$ length $L(x)$

$$\varphi(a) = \begin{pmatrix} 1 & 5.50 \\ 0 & 1 \end{pmatrix} + \Delta \varphi(a)_1$$

$$\varphi(b) = \begin{pmatrix} 3.62 & 3.60 \\ -1.90 & -1.62 \end{pmatrix} + \Delta \varphi(b)_1$$

$$\vdots$$

$$\varphi(abcB) = \varphi(a)\varphi(b)\varphi(c)\varphi(b)^{-1}$$

$$= \begin{pmatrix} p & q \\ r & s \end{pmatrix}$$

$$L(abcB) = 2 \arccosh \left( \frac{1}{2} (p + s) \right) = 6.467 + \Delta L_1$$
Length & dlengt

Curve $x \rightarrow$ matrix $\varphi(x) \rightarrow$ length $L(x)$

$\varphi(a) = \begin{pmatrix} 1 & 5.50 \\ 0 & 1 \end{pmatrix} + \Delta\varphi(a)_2$

$\varphi(b) = \begin{pmatrix} 3.62 & 3.60 \\ -1.90 & -1.62 \end{pmatrix} + \Delta\varphi(b)_2$

$\vdots$

$\varphi(abcB) = \varphi(a) \varphi(b) \varphi(c) \varphi(b)^{-1}$

$= \begin{pmatrix} p & q \\ r & s \end{pmatrix}$

$L(abcB) = 2 \text{ arccosh} \left( \frac{1}{2}(p + s) \right) = 6.467 + \Delta L_2$
Computing matrices

Numpy’s matrix algebra + recursive splitting to compute $\rho(x)$:

```python
import numpy as np

def representation(gen_matrices):
    def _rho(x):
        if len(x) == 0:
            # identity matrix
            return np.eye(2)
        elif len(x) == 1:
            # generator matrix
            return gen_matrices[x]
        else:
            N = len(x)
            return np.dot(_rho(x[:N//2]),_rho(x[N//2:]))
    return _rho

rho = representation( {'a': np.array( [[0,1],[1,1]] )} ) # etc

print(rho('aaaaa')) # -> [[3 5], [5 8]]
```
Length and dlength

```python
eps = 0.0001
rho0 = representation( {'a': np.array( [[0,1],[1,1]] )} )
rho1 = representation( {'a': np.array( [[0,1],[1+eps,1]] )} )

L0 = 2.0*np.arccosh(0.5*np.trace(rho0('aaaaa')))
L1 = 2.0*np.arccosh(0.5*np.trace(rho1('aaaaa')))

dL = (L1 - L0) / eps
print(dL)
```

Four calculations like this give the four coordinates of the vector $dL$ associated to a curve.
Length and dlengh

dL = (L1 - L0) / eps
print(dL)

Four calculations like this give the four coordinates of the vector $dL$ associated to a curve.

Next problem: 4D space is difficult to visualize!
Stereographic projection

“Open up” the surface in 4-space and flatten to 3-space.
Stereographic projection

“Open up” the surface in 4-space and flatten to 3-space.

\((p_1, p_2, p_3, p_4) \rightarrow \left( \frac{p_1}{1-p_4}, \frac{p_2}{1-p_4}, \frac{p_3}{1-p_4} \right)\)
Stereographic projection

“Open up” the surface in 4-space and flatten to 3-space.

\[(p_1, p_2, p_3, p_4) \rightarrow \left( \frac{p_1}{1 - p_4}, \frac{p_2}{1 - p_4}, \frac{p_3}{1 - p_4} \right)\]
Stereographic projection

“Open up” the surface in 4-space and flatten to 3-space.

\[(p_1, p_2, p_3, p_4) \rightarrow \left( \frac{p_1}{1 - p_4}, \frac{p_2}{1 - p_4}, \frac{p_3}{1 - p_4} \right)\]
Stereographic projection

“Open up” the surface in 4-space and flatten to 3-space.

\[(p_1, p_2, p_3, p_4) \rightarrow \left( \frac{p_1}{1 - p_4}, \frac{p_2}{1 - p_4}, \frac{p_3}{1 - p_4} \right)\]
Stereographic projection

“Open up” the surface in 4-space and flatten to 3-space.

\[(p_1, p_2, p_3, p_4) \rightarrow \left( \frac{p_1}{1 - p_4}, \frac{p_2}{1 - p_4}, \frac{p_3}{1 - p_4} \right)\]
Projecting each $dL$ to a 3-vector $(x, y, z)$, we obtain data like

<table>
<thead>
<tr>
<th>word</th>
<th>L</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>aDBdbd</td>
<td>19.195</td>
<td>0.630</td>
<td>0.729</td>
<td>0.695</td>
</tr>
<tr>
<td>aDbcBd</td>
<td>16.790</td>
<td>0.596</td>
<td>0.470</td>
<td>1.133</td>
</tr>
<tr>
<td>bcbCBd</td>
<td>14.664</td>
<td>0.177</td>
<td>0.544</td>
<td>1.242</td>
</tr>
</tbody>
</table>

with $> 10^6$ rows.

To make short curves more prominent, but all curves visible, we draw a sphere centered at $(x, y, z)$ of radius $1/L$ for each curve.
Rendering

Projecting each $dL$ to a 3-vector $(x, y, z)$, we obtain data like

<table>
<thead>
<tr>
<th>word</th>
<th>L</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>aDBdbd</td>
<td>19.195</td>
<td>0.630</td>
<td>0.729</td>
<td>0.695</td>
</tr>
<tr>
<td>aDbcBd</td>
<td>16.790</td>
<td>0.596</td>
<td>0.470</td>
<td>1.133</td>
</tr>
<tr>
<td>bcbCBd</td>
<td>14.664</td>
<td>0.177</td>
<td>0.544</td>
<td>1.242</td>
</tr>
</tbody>
</table>

with $> 10^6$ rows.

To make short curves more prominent, but all curves visible, we draw a sphere centered at $(x, y, z)$ of radius $1/L$ for each curve.

Easy way to draw millions of spheres if real-time rendering is not essential?
Rendering

Party like it’s 1992
Rendering

Party like it’s 1992

with ray-tracing!
**POV-Ray**

*POV-Ray* is an open-source ray tracer that uses a scene description language (SDL) with syntax reminiscent of C.

We generate SDL sphere primitives from the *dl* data using:

```python
outfile.write('sphere { <%f, %f, %f>, %f }

(employee)  % (x,y,z,1.0/L))
```

Appending some camera and lighting setup boilerplate gives a full scene file for rendering with *POV-Ray*. 
Fractal dust
Rotating the pole (wide angle)
Rotating the pole (close-up)
Clifford flow
Rotating the pole
Rotating the pole II

pmls05-061
Twists
Twists
Twists
Twists
Rings poster

PML($S_{0.5}$) The projective measured lamination space of the five-punctured sphere
David Dumas and François Guéritaud