LSA methods comparison

Mathematica

Anton Antonov
PythonForPrediction at WordPress
SimplifiedMachineLearningWorkflows-book at GitHub
December 2021
February 2022

Get Python mandalas collection

Setup

In[32]:=
```
# "Standard" packages
import pandas
import numpy
import random
import io
import time

# Plotting packages
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid

# Image processing package(s)
from PIL import Image, ImageOps

# Random mandalas packages
from RandomMandala import random_mandala, figure_to_image
```

In[33]:=
```
pythonSession = First@Pick[ExternalSessions[], #["System"] == "Python" & /@ ExternalSessions[]]
```

Out[33]=
```
ExternalSessionObject[ System: Python Version: 3.10.2 Name: DefaultPythonSession ]
```

Mandala collection

In[51]:=
```
# A list to accumulate random mandala images
mandala_images = []

# Generation loop
random.seed(443)
tstart = time.time()
for i in range(512):
```

```
    # Generate one random mandala figure
    fig2 = random_mandala(n_rows=None,
                          n_columns=None,
                          radius=[8, 6, 3],
                          rotational_symmetry_order=6,
                          symmetric_seed=True,
                          connecting_function='bezier_fill',
                          face_color="0.",
                          alpha = 1.0)
    fig2.tight_layout()

    # Convert the figure into an image and add it to the list
    mandala_images = mandala_images + [figure_to_image(fig2)]

    # Close figure to save memoru
    plt.close(fig2)

# Invert image colors
mandala_images1 = [ImageOps.invert(img) for img in mandala_images]

# Binarize images
mandala_images2 = [im.convert('1') for im in mandala_images1]

# Resize images
width, height = mandala_images2[0].size
print([width, height])
ratio = height / width
new_width = 200
mandala_images3 = [img.resize((new_width, round(new_width * ratio)), Image.ANTIALIAS) for img in mandala_images2]
print("Process time: " + str(time.time()-tstart))
```

```
[640, 480]

Process time: 22.576107025146484
```

Get mandalas in Mathematica

In[52]:=
```
mandalas3 = ExternalEvaluate[pythonSession, "mandala_images3"];
Length[mandalas3]
```

Out[53]= 512

Process random mandalas collection

Show one of the random mandalas:

In[54]:=
```
mandalas3⟦12⟧
```

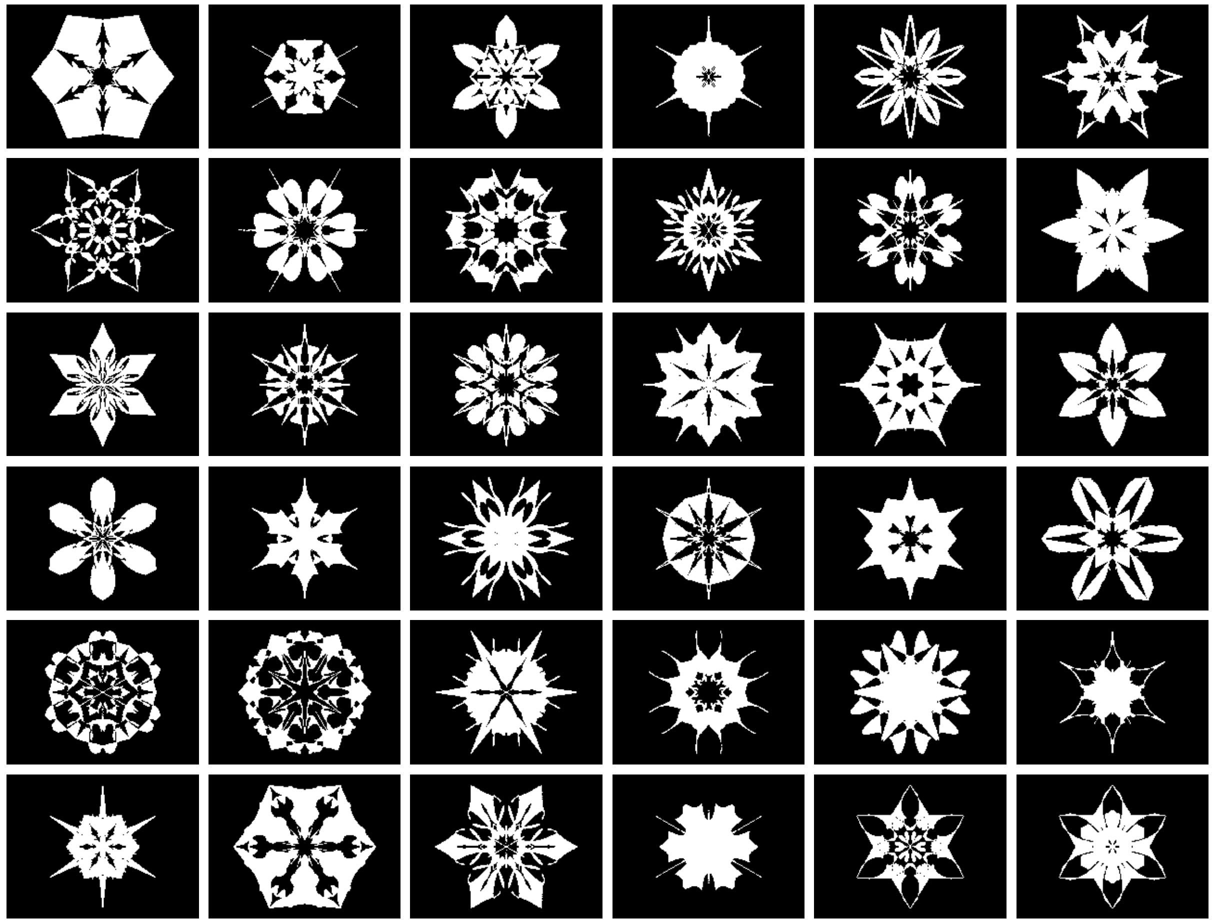
Out[54]=

In[55]:=
```
{imageSizeX, imageSizeY} = ImageDimensions[mandalas3⟦12⟧]
```

Out[55]= {200, 150}

Show an array of inverted random mandalas:

In[56]:=
```
Multicolumn[RandomSample[mandalas3, 36], 6]
```

Out[56]=


In[57]:=
```
(*ResourceFunction["RandomMandala"]["SymmetricSeed"→RandomChoice[{True,False}],"RotationalSymmetryOrder"→3,ImageSize→Large,opts]*)
```

In[58]:=
```
Tally[ImageDimensions /@ mandalas3]
```

Out[58]=
```
{{{200, 150}, 512}}
```

Convert each image into array and flatten that array:

In[59]:=
```
mandalaArrays = Flatten[ImageData[#]] & /@ mandalas3;
Dimensions[mandalaArrays]
```

Out[60]=
```
{512, 30000}
```

Make a matrix of flattened mandalas:

In[61]:=
```
mandalaMat = SparseArray[N@mandalaArrays]
```

Out[61]=
```
SparseArray[ Specified elements: 3436475
             Dimensions: {512, 30000} ]
Data not in notebook. Store now
```

Make the corresponding `SSparseMatrix` object:

In[62]:=
```
mandalaSMat = ToSSparseMatrix[mandalaMat, "RowNames" → Automatic, "ColumnNames" → Automatic]
```

Out[62]=
```
SparseArray[ Specified elements: 3 436 475
             Dimensions: {512, 30 000} ]
Data not in notebook. Store now
```

LSAMon object creation

Se the number of topics:

In[63]:=
```
numberOfTopics = 40;
```

Create the LSAMon object:

In[64]:=
```
AbsoluteTiming[
 lsaObj =
   LSAMonUnit[] ⟹
    LSAMonSetDocumentTermMatrix[mandalaSMat] ⟹
    LSAMonApplyTermWeightFunctions["GlobalWeightFunction" → "None", "LocalWeightFunction" → "None", "NormalizerFunction" → "None"];
]
```

Out[64]=
```
{0.065046, Null}
```

SVD

Here we extract image topics using Singular Value Decomposition (SVD):

In[65]:=
```
AbsoluteTiming[
 lsaSVDObj =
   lsaObj ⟹
    LSAMonExtractTopics["NumberOfTopics" → numberOfTopics, Method → "SVD", "MaxSteps" → 16, "MinNumberOfDocumentsPerTerm" → 0];
]
```

Out[65]=
```
{1.14347, Null}
```

In[66]:=
```
svdH = lsaSVDObj ⟹ LSAMonNormalizeMatrixProduct[Normalized → Right] ⟹ LSAMonTakeH
```

Out[66]=
```
SparseArray[ Specified elements: 591 400
             Dimensions: {40, 30 000} ]
Data not in notebook. Store now
```

NNMF

Here we extract image topics using Non-Negative Matrix Factorization (NNFM):

In[67]:=
```
AbsoluteTiming[
 lsaNNMFObj =
   lsaObj ⟹
    LSAMonExtractTopics["NumberOfTopics" → numberOfTopics, Method → "NNMF", "MaxSteps" → 16, "MinNumberOfDocumentsPerTerm" → 0];
]
```
Out[67]= {692.447, Null}

In[68]:=
```
nnmfH = lsaNNMFObj ⟹ LSAMonNormalizeMatrixProduct[Normalized → Right] ⟹ LSAMonTakeH;
```

ICA

Here we extract image topics using Independent Component Analysis (ICA):

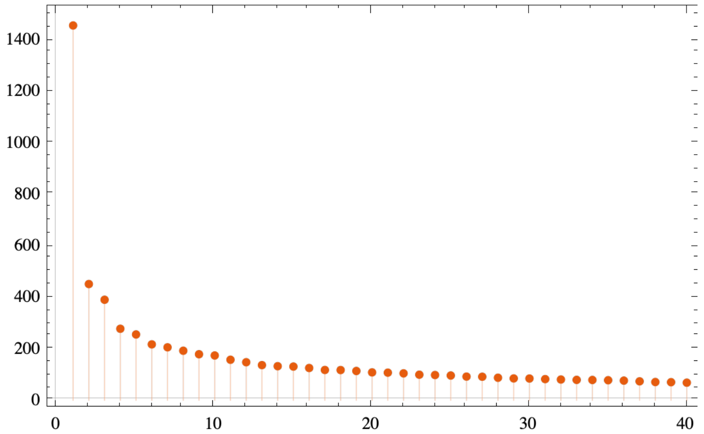
In[69]:=
```
AbsoluteTiming[
 lsaICAObj =
   lsaObj ⟹
    LSAMonExtractTopics["NumberOfTopics" → numberOfTopics, Method → "ICA", "MaxSteps" → 16, "MinNumberOfDocumentsPerTerm" → 0];
]
```
Out[69]= {1.48852, Null}

In[70]:=
```
icaH = lsaICAObj ⟹ LSAMonNormalizeMatrixProduct[Normalized → Right] ⟹ LSAMonTakeH;
```

Show topics interpretation

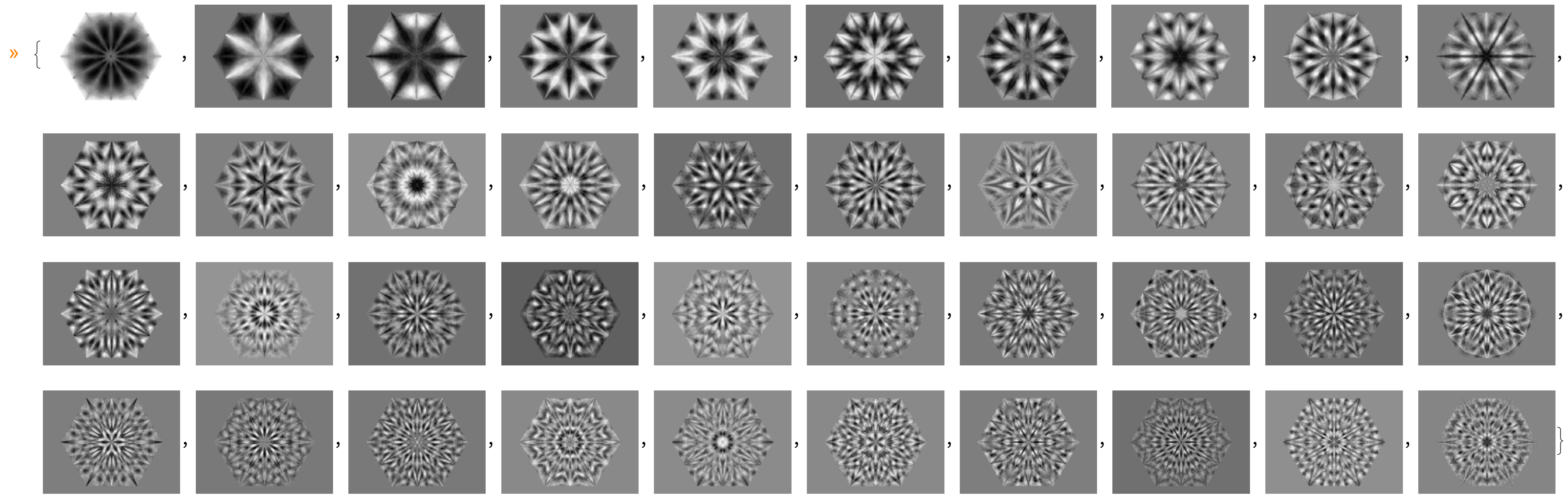
SVD

Show the importance coefficients of the topics (if SVD was used the plot would show the singular values):

In[71]:=
```
ListPlot[Norm /@ SparseArray[lsaSVDObj ⟹ LSAMonNormalizeMatrixProduct[Normalized → Left] ⟹ LSAMonTakeH], Filling → Axis, PlotRange → All, PlotTheme → "Scientific"]
```
Out[71]=


Show the interpretation of the extracted image topics:

In[72]:=
```
lsaSVDObj ⟹
  LSAMonNormalizeMatrixProduct[Normalized → Right] ⟹ LSAMonEchoFunctionContext[ImageAdjust[Image[Partition[#, ImageDimensions[mandalas3〚1〛]〚1〛]]] & /@ SparseArray[#H] &];
```

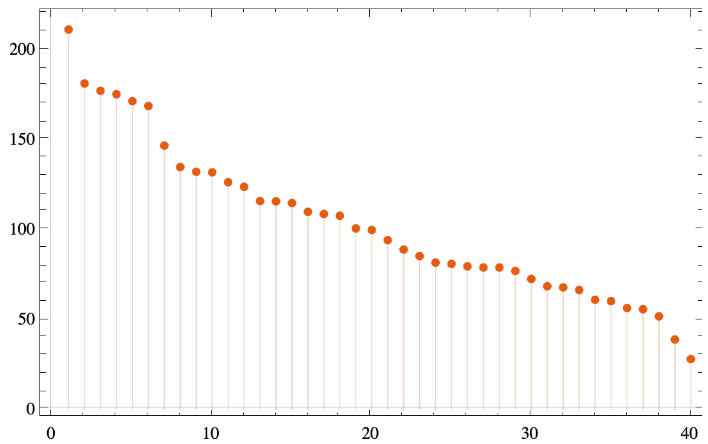


NNMF

Show the importance coefficients of the topics :

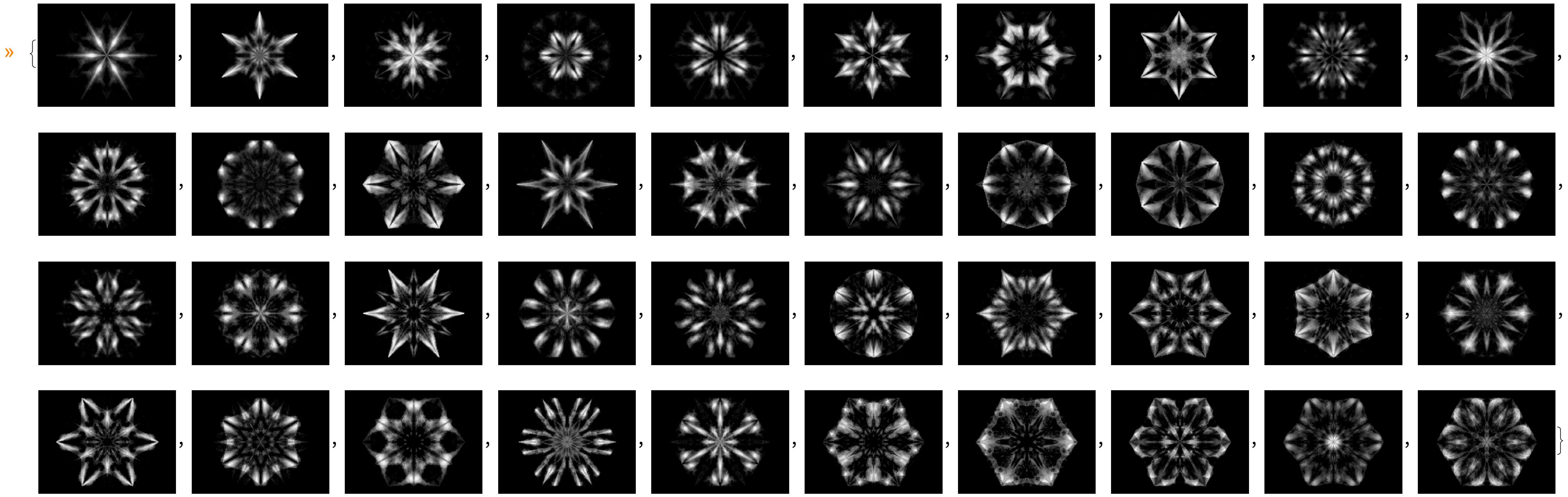
In[73]:=
```
ListPlot[Norm /@ SparseArray[lsaNNMFObj ⟹ LSAMonNormalizeMatrixProduct[Normalized → Left] ⟹ LSAMonTakeH], Filling → Axis, PlotRange → All, PlotTheme → "Scientific"]
```

Out[73]=



Show the interpretation of the extracted image topics:

In[74]:=
```
lsaNNMFObj ⟹
  LSAMonNormalizeMatrixProduct[Normalized → Right] ⟹ LSAMonEchoFunctionContext[ImageAdjust[Image[Partition[#, ImageDimensions[mandalas3〚1〛]〚1〛]]] & /@ SparseArray[#H] &];
```

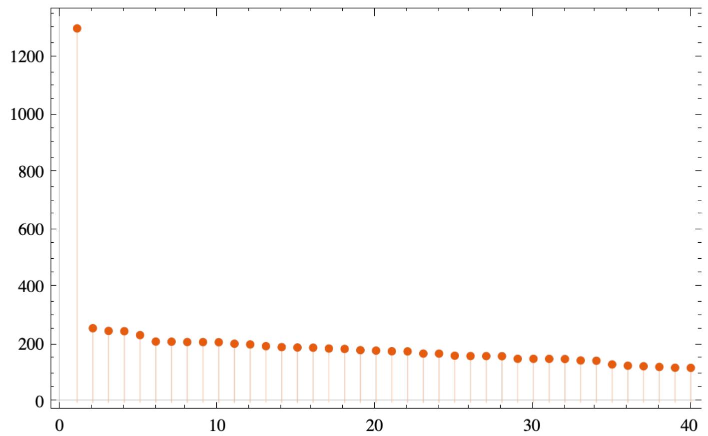


ICA

Show the importance coefficients of the topics :

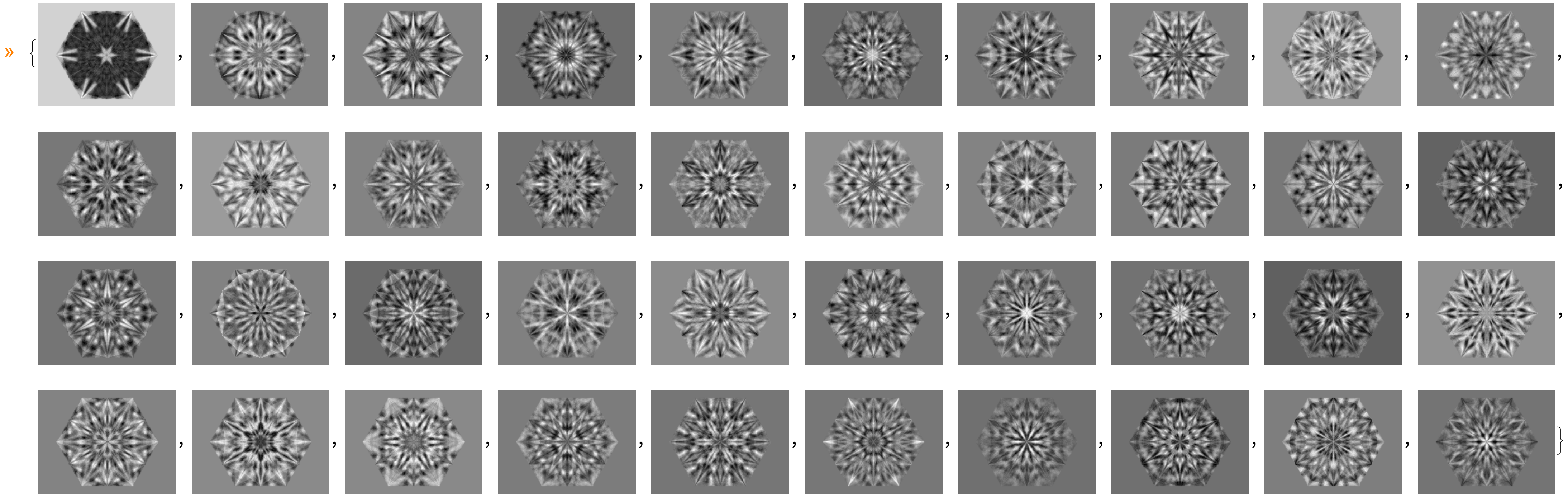
In[75]:=
```
ListPlot[Norm /@ SparseArray[lsaICAObj ⟹ LSAMonNormalizeMatrixProduct[Normalized → Left] ⟹ LSAMonTakeH], Filling → Axis, PlotRange → All, PlotTheme → "Scientific"]
```

Out[75]=



Show the interpretation of the extracted image topics:

In[76]:=
```
lsaICAObj⟹
  LSAMonNormalizeMatrixProduct[Normalized → Right]⟹LSAMonEchoFunctionContext[ImageAdjust[Image[Partition[#, ImageDimensions[mandalas3⟦1⟧]⟦1⟧]]] & /@ SparseArray[#H] &];
```



Approximation

Get a new, unseen random mandala:

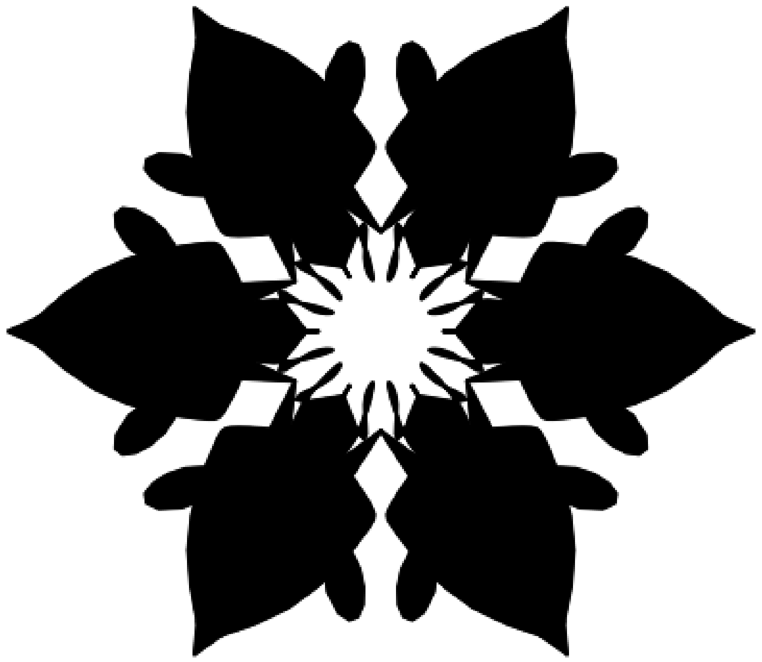
In[229]:=
```
(*opts={"Radius"→10,
  "RotationalSymmetryOrder"→6,
  "ConnectingFunction"→FilledCurve@*BezierCurve,
  ImageSize→imageSizeX};
testMandala=ResourceFunction["RandomMandala"]["SymmetricSeed"→True,"RotationalSymmetryOrder"→6,opts]*)
```

In[230]:=
```
testMandala = ExternalEvaluate[pythonSession,
  "
# random.seed(332)
random.seed(89)

fig = random_mandala(radius=[8, 6, 3],
                     rotational_symmetry_order=6,
                     symmetric_seed=True,
                     connecting_function='bezier_fill',
                     face_color='0.',
                     alpha = 1.0)

figure_to_image(fig)
"]
```

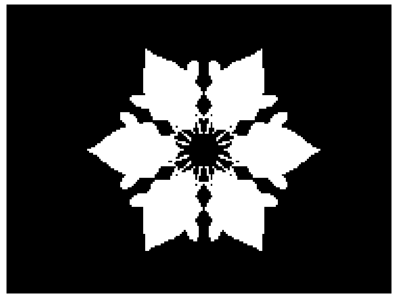
Out[230]=



Get the generated mandala image in Mathematica:

In[231]:=
```
testMandala01 = Binarize[ColorNegate[ColorConvert[ImageResize[testMandala, ImageDimensions[mandalas3[[1]]]], "Grayscale"]]]
```

Out[231]=



Get the corresponding vector:

In[232]:=
```
testMandalaMat = SparseArray[{Flatten[ImageData[testMandala01]]}];
testMandalaSMat = ToSSparseMatrix[testMandalaMat, "RowNames" → Automatic, "ColumnNames" → Automatic]
```

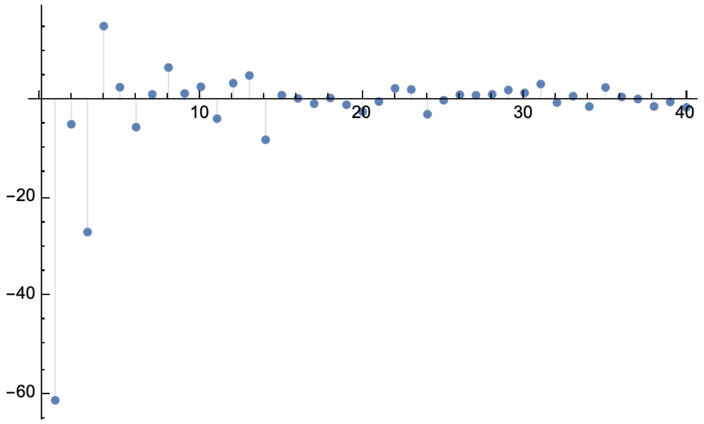
Out[233]= SparseArray[Specified elements: 6013 Dimensions: {1, 30 000}]

SVD

Represent by topics :

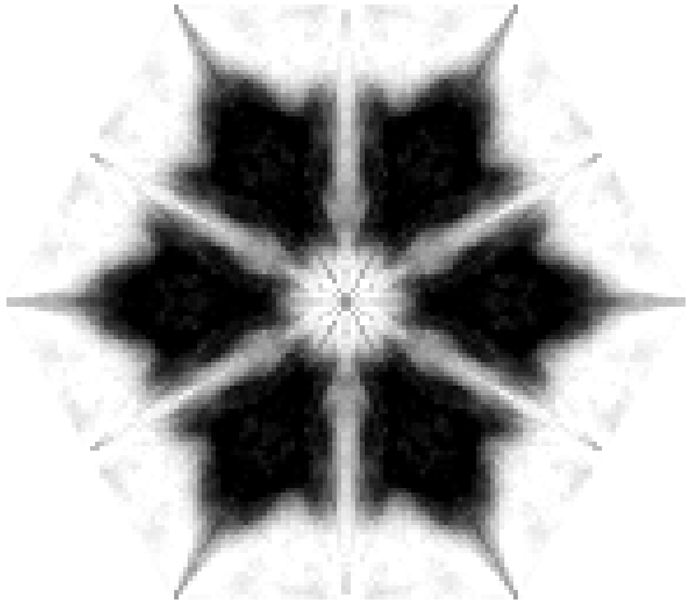
In[234]:=
```
matRepresentation =
  lsaSVDObj ⟹
   LSAMonNormalizeMatrixProduct[Normalized → Right] ⟹
   LSAMonRepresentByTopics[testMandalaSMat] ⟹
   LSAMonTakeValue;
lsCoeff = Normal@SparseArray[matRepresentation[[1, All]]];
ListPlot[lsCoeff, Filling → Axis, PlotRange → All]
```

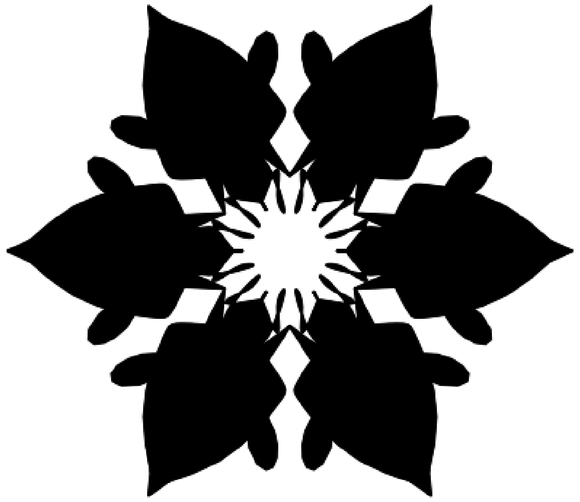
Out[236]=



In[237]:=
```
vecRepresentation = lsCoeff.SparseArray[svdH];
GraphicsGrid[{{ColorNegate@Image[Rescale[Partition[vecRepresentation, ImageDimensions[testMandala01][[1]]], {0, 1}]], testMandala}}, ImageSize → 1000]
```

Out[238]=

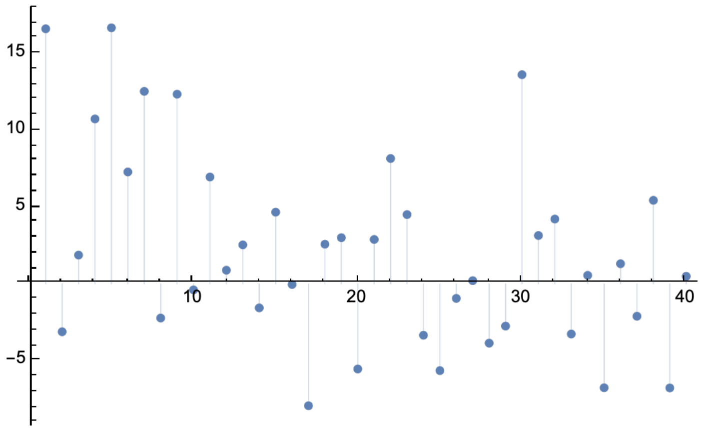




NNMF

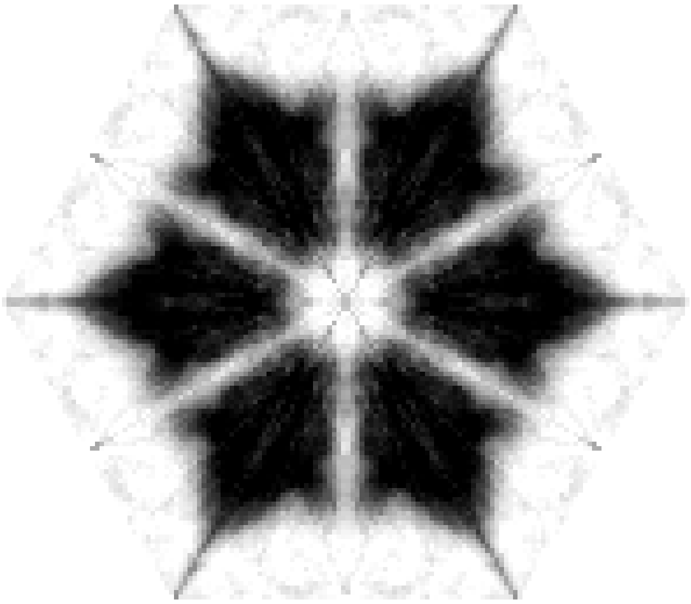
In[239]:=
```
matRepresentation =
  lsaNNMFObj ⟹
   LSAMonNormalizeMatrixProduct[Normalized → Right] ⟹
   LSAMonRepresentByTopics[testMandalaSMat] ⟹
   LSAMonTakeValue;
lsCoeff = Normal@SparseArray[matRepresentation[[1, All]]];
ListPlot[lsCoeff, Filling → Axis, PlotRange → All]
```

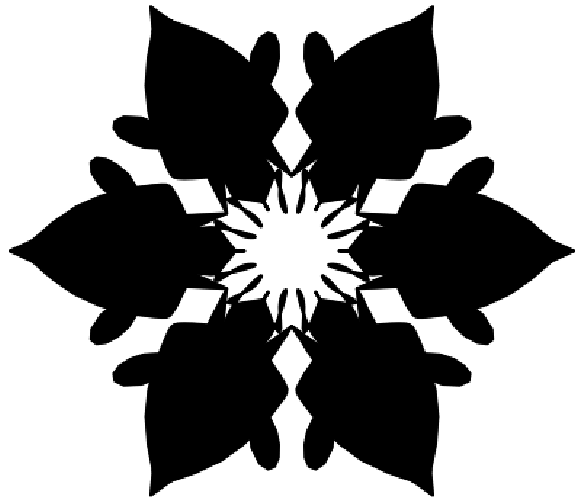
Out[241]=



In[242]:=
```
vecRepresentation = lsCoeff.SparseArray[nnmfH];
GraphicsGrid[{{ColorNegate@Image[Rescale[Partition[vecRepresentation, ImageDimensions[testMandala01][[1]]], {0, 1}]], testMandala}}, ImageSize → 1000]
```

Out[243]=

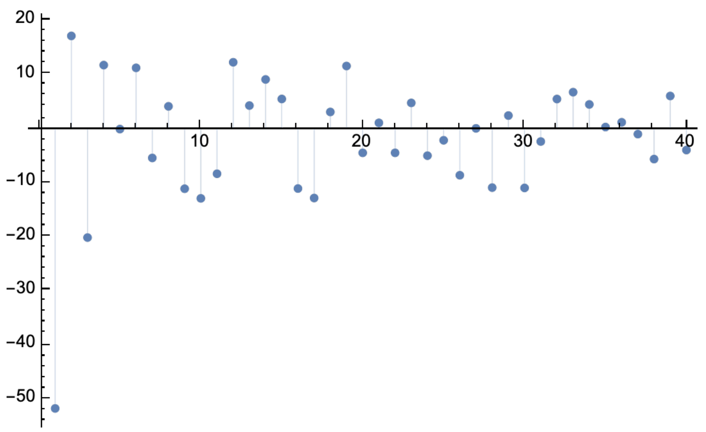




ICA

In[244]:=
```
matRepresentation =
  lsaICAObj ⟹
   LSAMonNormalizeMatrixProduct[Normalized → Right] ⟹
   LSAMonRepresentByTopics[testMandalaSMat] ⟹
   LSAMonTakeValue;
lsCoeff = Normal@SparseArray[matRepresentation[[1, All]]];
ListPlot[lsCoeff, Filling → Axis, PlotRange → All]
```

Out[246]=



In[247]:=
```
vecRepresentation = lsCoeff.SparseArray[icaH];
GraphicsGrid[{{ColorNegate@Image[Rescale[Partition[vecRepresentation, ImageDimensions[testMandala01][[1]]], {0, 1}]], testMandala}}, ImageSize → 1000]
```

Out[248]=