article graphicx amsmath

1 Tutorial: Running Monoids in Jupyter Notebook and SageMath End-Product Inhibition Cases

Step 1: Install a Jupyter Notebook Platform

Follow the steps below based on your operating system (Windows/macOS/Linux):

1. Visit: `https://www.anaconda.com/products/distribution`
2. Download the appropriate installer:
- Windows: `.exe`
- macOS: `.pkg`
- Linux: `.sh`
3. Run the installer and follow the setup instructions (default settings are recommended).
4. After installation:
- Open Anaconda Navigator and click "Launch" under Jupyter Notebook, OR
- Open a terminal (or Anaconda Prompt) and run: `jupyter notebook`. On Ubuntu: `jupyter-notebook`.

Step 2: Follow the Screenshots to Run Monoids for End-Product Inhibition Cases

Note: The script supports monoid generation for cones of dimension $n = 1$ up to $n = 9$. Use the correct filenames for each case. For example: `Cone3n_1.txt`, `Cone3n_2.txt`, ..., `Cone9n_10.txt`.

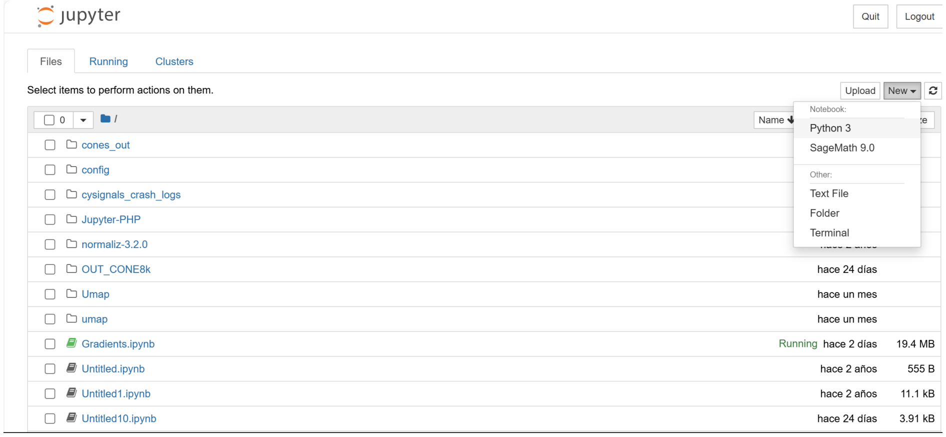
Note: To concatenate all the individual output files into one, use the following command format in Linux:

```
cat *_TF_DNA31.sage >> OUT31_TF_DNAsage.txt
```

Replace `31` with the corresponding number for your case. For $n = 4$, you would run:

```
cat *_TF_DNA41.sage >> OUT41_TF_DNAsage.txt
cat *_TF_DNA42.sage >> OUT42_TF_DNAsage.txt
cat *_TF_DNA43.sage >> OUT43_TF_DNAsage.txt
cat *_TF_DNA44.sage >> OUT44_TF_DNAsage.txt
cat *_TF_DNA45.sage >> OUT45_TF_DNAsage.txt
```

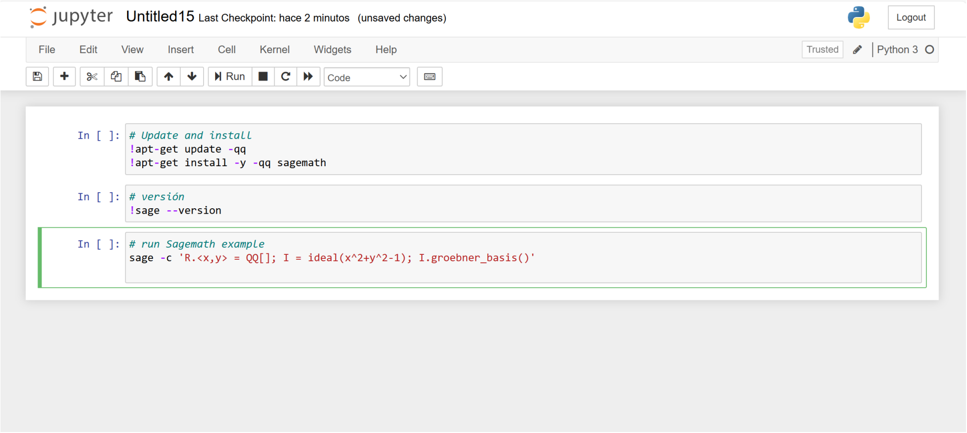
Case for the length of the pathway n=7. Allosteric system for end-product inhibition (lysine-aspartate).



Select items to perform actions on them.

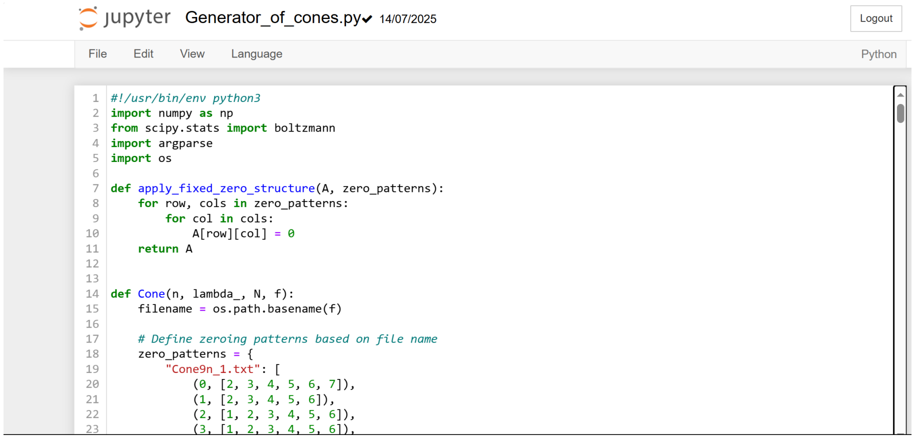
Name ↓			
cones_out			
config			
cysignals_crash_logs			
Jupyter-PHP			
normaliz-3.2.0			
OUT_CONE8k		hace 24 días	
Umap		hace un mes	
umap		hace un mes	
Gradients.ipynb	Running	hace 2 días	19.4 MB
Untitled.ipynb		hace 2 años	555 B
Untitled1.ipynb		hace 2 años	11.1 kB
Untitled10.ipynb		hace 24 días	3.91 kB

Figure 1: Open a new notebook using Python 3.



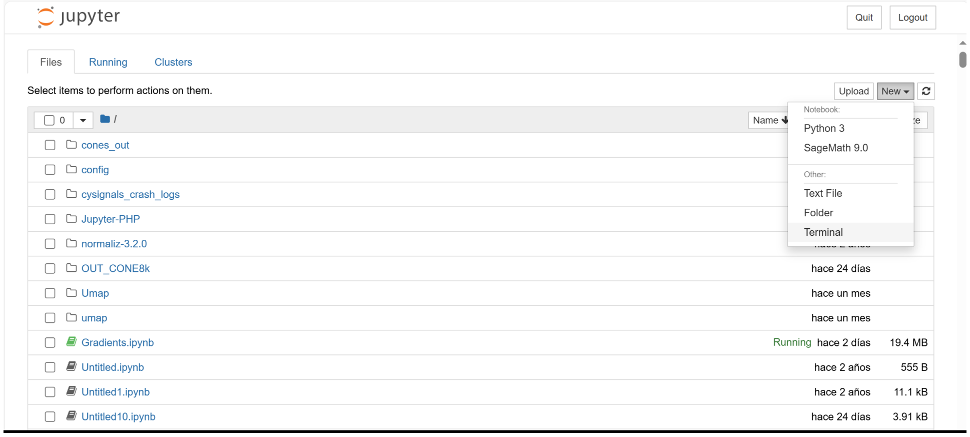
```
In [ ]: # Update and install
        !apt-get update -qq
        !apt-get install -y -qq sagemath

In [ ]: # versión
        !sage --version

In [ ]: # run Sagemath example
        sage -c 'R.<x,y> = QQ[]; I = ideal(x^2+y^2-1); I.groebner_basis()'
```

Figure 2: Install the platform needed to run algorithms in computational algebraic geometry. For computing monoid generators, copy and paste the code shown in this figure into your notebook.



```python
#!/usr/bin/env python3
import numpy as np
from scipy.stats import boltzmann
import argparse
import os

def apply_fixed_zero_structure(A, zero_patterns):
    for row, cols in zero_patterns:
        for col in cols:
            A[row][col] = 0
    return A


def Cone(n, lambda_, N, f):
    filename = os.path.basename(f)

    # Define zeroing patterns based on file name
    zero_patterns = {
        "Cone9n_1.txt": [
            (0, [2, 3, 4, 5, 6, 7]),
            (1, [2, 3, 4, 5, 6]),
            (2, [1, 2, 3, 4, 5, 6]),
            (3, [1, 2, 3, 4, 5, 6]),
```

Figure 3: Open the script file `Generator_of_cones.py`, which is available in the GitHub repository. Copy and paste it into a new Python notebook. You can wirte the Linux line: python3 Generator_of_cones.py -n 500 -f Cone3n$_1.txt$



Select items to perform actions on them.

cones_out			
config			
cysignals_crash_logs			
Jupyter-PHP			
normaliz-3.2.0			
OUT_CONE8k		hace 24 días	
Umap		hace un mes	
umap		hace un mes	
Gradients.ipynb	Running	hace 2 días	19.4 MB
Untitled.ipynb		hace 2 años	555 B
Untitled1.ipynb		hace 2 años	11.1 kB
Untitled10.ipynb		hace 24 días	3.91 kB

Figure 4: Once SageMath version 9.0 is installed, open a new notebook in SageMath.

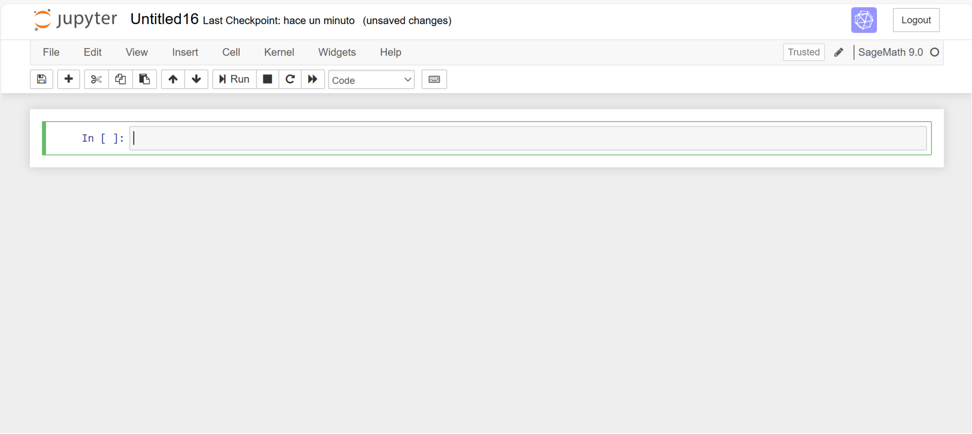
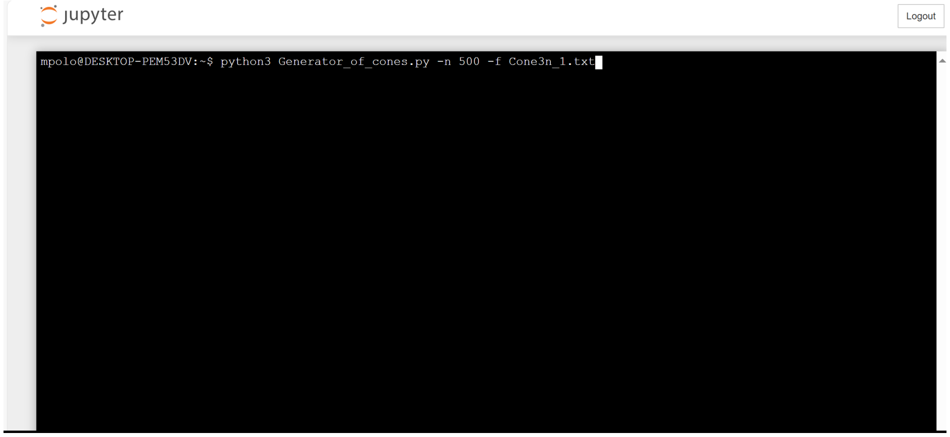
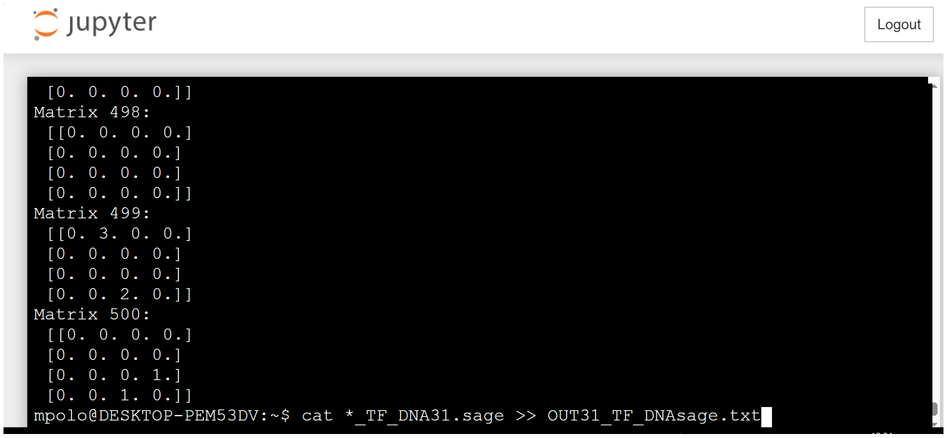


Figure 5: A new SageMath notebook will open. Leave it open; you'll run a script in this notebook shortly.



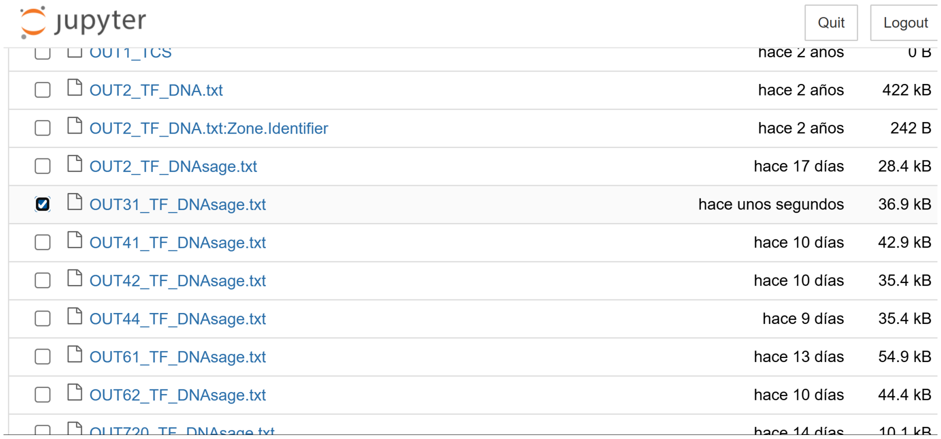
```
mpolo@DESKTOP-PEM53DV:~$ python3 Generator_of_cones.py -n 500 -f Cone3n_1.txt
```

Figure 6: Open a new terminal in Jupyter. Type the command shown in the figure. File names must follow the naming convention: `Cone3n_1.txt`, `Cone3n_2.txt`, ..., `Cone7n_8.txt`, depending on the length n of the pathways. The `-n 500` option generates 500 random matrices using the Boltzmann distribution.



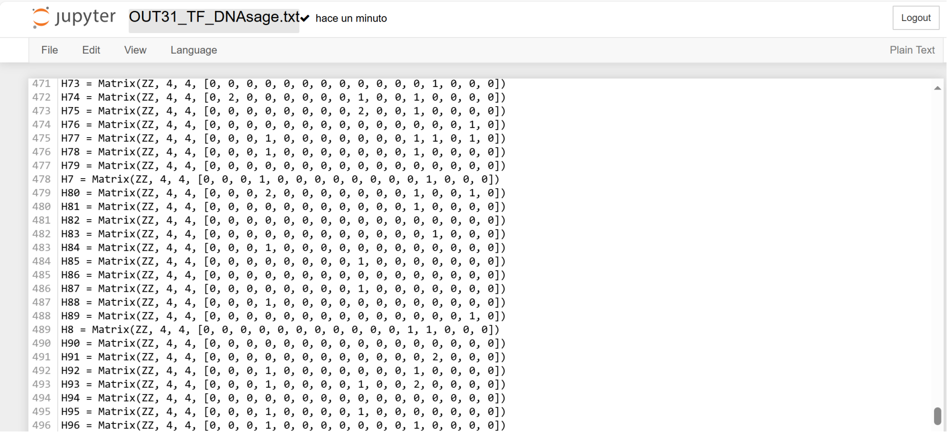
```
 [0. 0. 0. 0.]]
Matrix 498:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
Matrix 499:
 [[0. 3. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 2. 0.]]
Matrix 500:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]]
mpolo@DESKTOP-PEM53DV:~$ cat *_TF_DNA31.sage >> OUT31_TF_DNAsage.txt
```

Figure 7: After generating 500 random matrices, save them in a single file using the command shown. The output file in this case is `OUT31_TF_DNAsage.txt`.



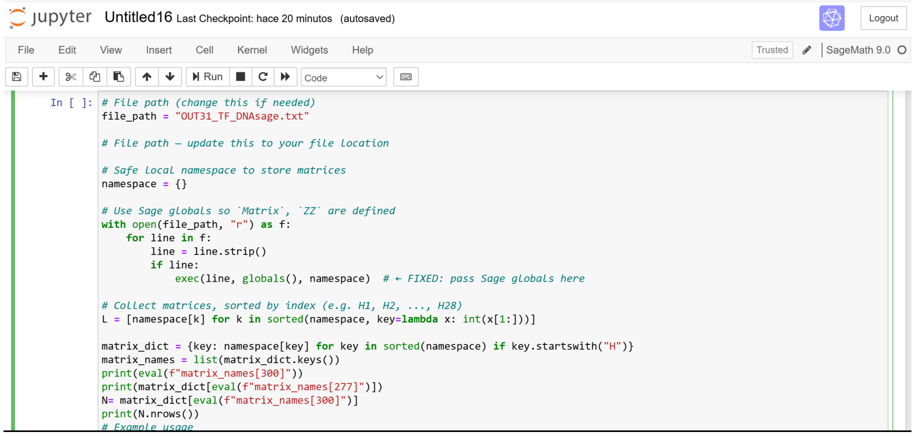
□	OUT1_TCS	hace 2 años	0 B
□	OUT2_TF_DNA.txt	hace 2 años	422 kB
□	OUT2_TF_DNA.txt:Zone.Identifier	hace 2 años	242 B
□	OUT2_TF_DNAsage.txt	hace 17 días	28.4 kB
☑	OUT31_TF_DNAsage.txt	hace unos segundos	36.9 kB
□	OUT41_TF_DNAsage.txt	hace 10 días	42.9 kB
□	OUT42_TF_DNAsage.txt	hace 10 días	35.4 kB
□	OUT44_TF_DNAsage.txt	hace 9 días	35.4 kB
□	OUT61_TF_DNAsage.txt	hace 13 días	54.9 kB
□	OUT62_TF_DNAsage.txt	hace 10 días	44.4 kB
□	OUT720_TF_DNAsage.txt	hace 14 días	10.1 kB

Figure 8: You can verify the saved output file in the Jupyter file tree as shown.



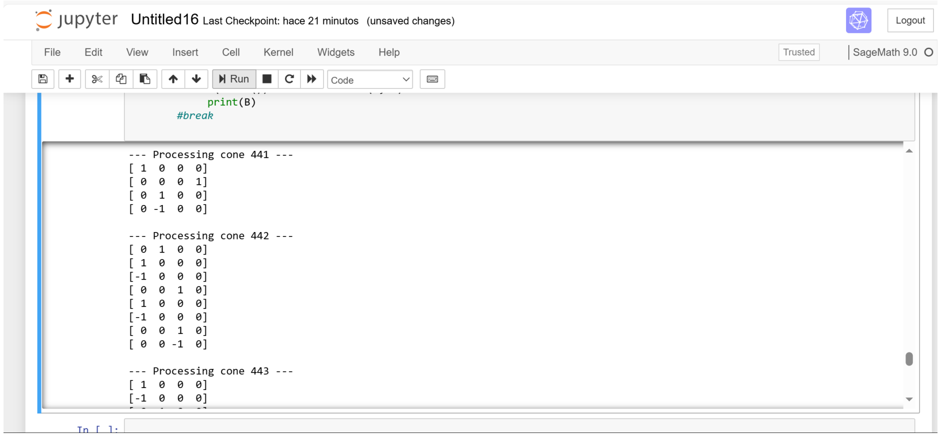
```
H73 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0])
H74 = Matrix(ZZ, 4, 4, [0, 2, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0])
H75 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0])
H76 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
H77 = Matrix(ZZ, 4, 4, [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0])
H78 = Matrix(ZZ, 4, 4, [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0])
H79 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
H7 = Matrix(ZZ, 4, 4, [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0])
H80 = Matrix(ZZ, 4, 4, [0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0])
H81 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0])
H82 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
H83 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0])
H84 = Matrix(ZZ, 4, 4, [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
H85 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0])
H86 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
H87 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0])
H88 = Matrix(ZZ, 4, 4, [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
H89 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
H8 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0])
H90 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
H91 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0])
H92 = Matrix(ZZ, 4, 4, [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0])
H93 = Matrix(ZZ, 4, 4, [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0])
H94 = Matrix(ZZ, 4, 4, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
H95 = Matrix(ZZ, 4, 4, [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0])
H96 = Matrix(ZZ, 4, 4, [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0])
```

Figure 9: The content of the output file should appear as shown. These files are formatted for execution in SageMath.



```
# File path (change this if needed)
file_path = "OUT31_TF_DNAsage.txt"

# File path — update this to your file location

# Safe local namespace to store matrices
namespace = {}

# Use Sage globals so `Matrix`, `ZZ` are defined
with open(file_path, "r") as f:
    for line in f:
        line = line.strip()
        if line:
            exec(line, globals(), namespace)  # ← FIXED: pass Sage globals here

# Collect matrices, sorted by index (e.g. H1, H2, ..., H28)
L = [namespace[k] for k in sorted(namespace, key=lambda x: int(x[1:]))]

matrix_dict = {key: namespace[key] for key in sorted(namespace) if key.startswith("H")}
matrix_names = list(matrix_dict.keys())
print(eval(f"matrix_names[300]"))
print(matrix_dict[eval(f"matrix_names[277]")])
N= matrix_dict[eval(f"matrix_names[300]")]
print(N.nrows())
# Example usage
```

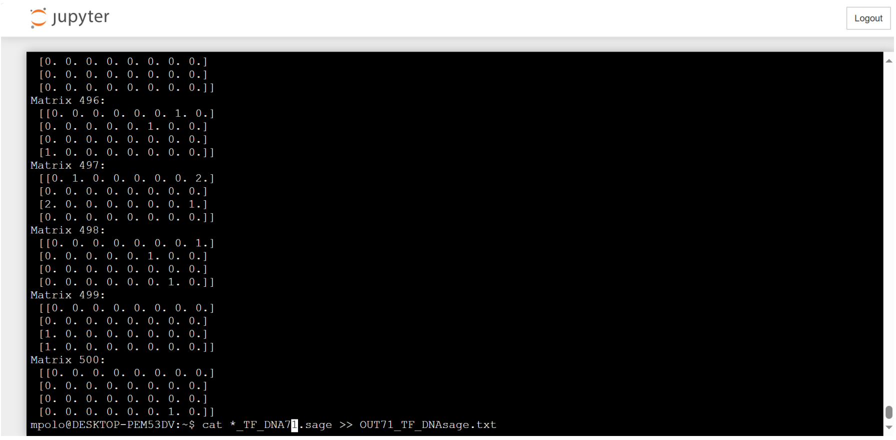
Figure 10: Return to the SageMath notebook left open in Figure 5. Copy and paste the script from the file `Sagemathscrpt.txt` (available in the GitHub repository). Make sure to set the filename at the beginning of the script (e.g., `OUT31_TF_DNAsage.txt`).



```
            print(B)
        #break
```

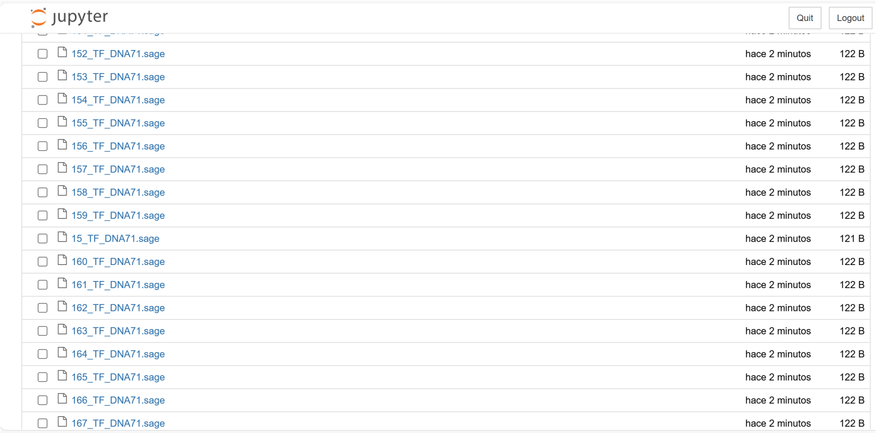
```
--- Processing cone 441 ---
[ 1  0  0  0]
[ 0  0  0  1]
[ 0  1  0  0]
[ 0 -1  0  0]

--- Processing cone 442 ---
[ 0  1  0  0]
[ 1  0  0  0]
[-1  0  0  0]
[ 0  0  1  0]
[ 1  0  0  0]
[-1  0  0  0]
[ 0  0  1  0]
[ 0  0 -1  0]

--- Processing cone 443 ---
[ 1  0  0  0]
[-1  0  0  0]
```

Figure 11: After pasting the script into the notebook, run it. The output will be 500 matrix representations of monoids for the selected pathway length n.



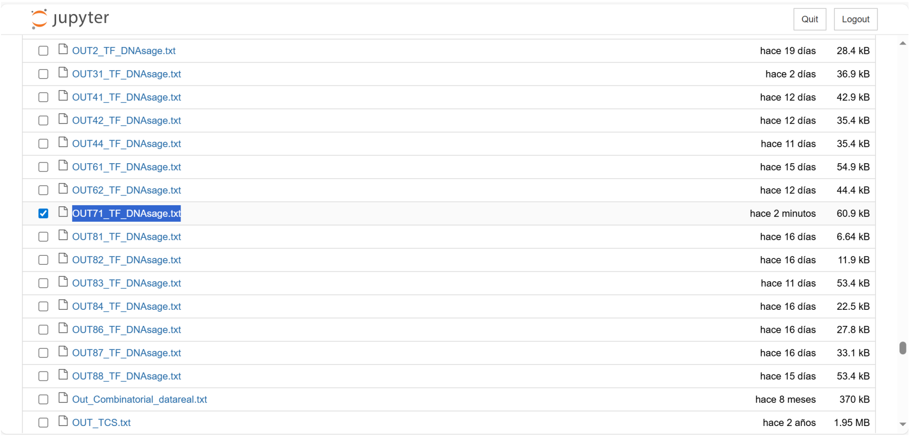
```
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
Matrix 496:
 [[0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]]
Matrix 497:
 [[0. 1. 0. 0. 0. 0. 0. 2.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [2. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
Matrix 498:
 [[0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]]
Matrix 499:
 [[0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]]
Matrix 500:
 [[0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]]
mpolo@DESKTOP-PEM53DV:~$ cat *_TF_DNA71.sage >> OUT71_TF_DNAsage.txt
```

Figure 12: You only write in new terminal on you Jupyter-Notebook the line in linux: python3 Generator_of_cones.py -n 500 -f Cone7n_1.txt



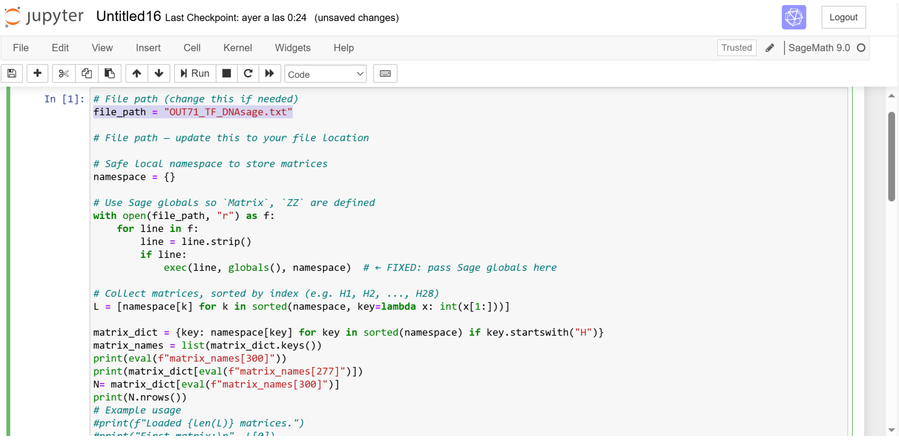
☐	152_TF_DNA71.sage	hace 2 minutos	122 B
☐	153_TF_DNA71.sage	hace 2 minutos	122 B
☐	154_TF_DNA71.sage	hace 2 minutos	122 B
☐	155_TF_DNA71.sage	hace 2 minutos	122 B
☐	156_TF_DNA71.sage	hace 2 minutos	122 B
☐	157_TF_DNA71.sage	hace 2 minutos	122 B
☐	158_TF_DNA71.sage	hace 2 minutos	122 B
☐	159_TF_DNA71.sage	hace 2 minutos	122 B
☐	15_TF_DNA71.sage	hace 2 minutos	121 B
☐	160_TF_DNA71.sage	hace 2 minutos	122 B
☐	161_TF_DNA71.sage	hace 2 minutos	122 B
☐	162_TF_DNA71.sage	hace 2 minutos	122 B
☐	163_TF_DNA71.sage	hace 2 minutos	122 B
☐	164_TF_DNA71.sage	hace 2 minutos	122 B
☐	165_TF_DNA71.sage	hace 2 minutos	122 B
☐	166_TF_DNA71.sage	hace 2 minutos	122 B
☐	167_TF_DNA71.sage	hace 2 minutos	122 B

Figure 13: After write the following liux line to get your files in an only file: cat *_TF_DNA71.sage ¿¿ OUT71_TF_DNAsage.txt



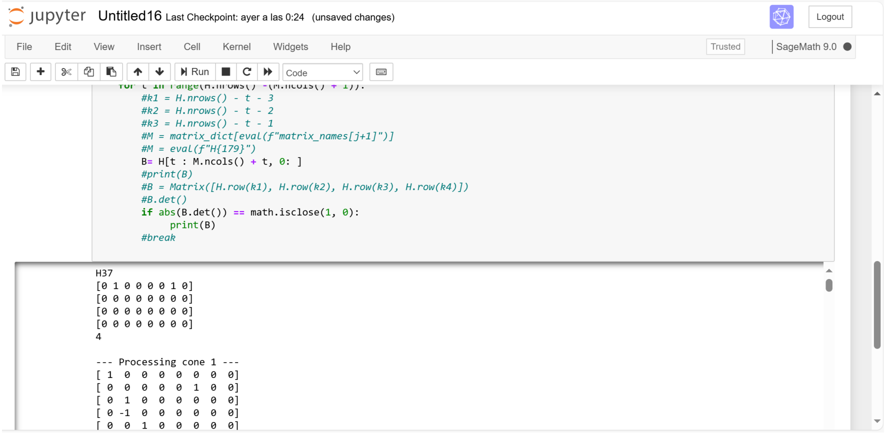
☐	OUT2_TF_DNAsage.txt	hace 19 días	28.4 kB
☐	OUT31_TF_DNAsage.txt	hace 2 días	36.9 kB
☐	OUT41_TF_DNAsage.txt	hace 12 días	42.9 kB
☐	OUT42_TF_DNAsage.txt	hace 12 días	35.4 kB
☐	OUT44_TF_DNAsage.txt	hace 11 días	35.4 kB
☐	OUT61_TF_DNAsage.txt	hace 15 días	54.9 kB
☐	OUT62_TF_DNAsage.txt	hace 12 días	44.4 kB
☑	OUT71_TF_DNAsage.txt	hace 2 minutos	60.9 kB
☐	OUT81_TF_DNAsage.txt	hace 16 días	6.64 kB
☐	OUT82_TF_DNAsage.txt	hace 16 días	11.9 kB
☐	OUT83_TF_DNAsage.txt	hace 11 días	53.4 kB
☐	OUT84_TF_DNAsage.txt	hace 16 días	22.5 kB
☐	OUT86_TF_DNAsage.txt	hace 16 días	27.8 kB
☐	OUT87_TF_DNAsage.txt	hace 16 días	33.1 kB
☐	OUT88_TF_DNAsage.txt	hace 15 días	53.4 kB
☐	Out_Combinatorial_datareal.txt	hace 8 meses	370 kB
☐	OUT_TCS.txt	hace 2 años	1.95 MB

Figure 14: You can visualize this file in your environment path on Jupyter platform.



```
# File path (change this if needed)
file_path = "OUT71_TF_DNAsage.txt"

# File path — update this to your file location

# Safe local namespace to store matrices
namespace = {}

# Use Sage globals so `Matrix`, `ZZ` are defined
with open(file_path, "r") as f:
    for line in f:
        line = line.strip()
        if line:
            exec(line, globals(), namespace)  # ← FIXED: pass Sage globals here

# Collect matrices, sorted by index (e.g. H1, H2, ..., H28)
L = [namespace[k] for k in sorted(namespace, key=lambda x: int(x[1:]))]

matrix_dict = {key: namespace[key] for key in sorted(namespace) if key.startswith("H")}
matrix_names = list(matrix_dict.keys())
print(eval(f"matrix_names[300]"))
print(matrix_dict[eval(f"matrix_names[277]")])
N= matrix_dict[eval(f"matrix_names[300]")]
print(N.nrows())
# Example usage
#print(f"Loaded {len(L)} matrices.")
#print("First matrix:\n", L[0])
```

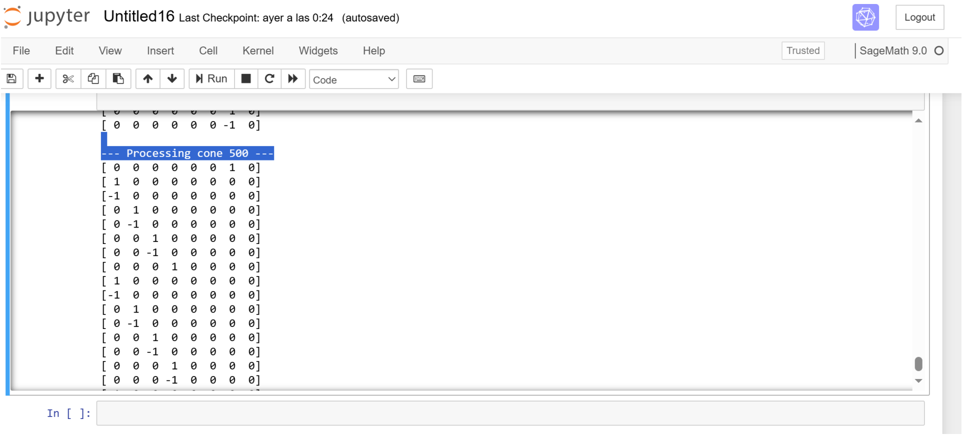
Figure 15: Now, to execute the script for monoid on Sagemath platform. Write correctly the name of you out file, in this case: OUT71_TF_DNAsage.txt



```
    for t in range(H.nrows() -(M.ncols() + 1)):
        #k1 = H.nrows() - t - 3
        #k2 = H.nrows() - t - 2
        #k3 = H.nrows() - t - 1
        #M = matrix_dict[eval(f"matrix_names[j+1]")]
        #M = eval(f"H{179}")
        B= H[t : M.ncols() + t, 0: ]
        #print(B)
        #B = Matrix([H.row(k1), H.row(k2), H.row(k3), H.row(k4)])
        #B.det()
        if abs(B.det()) == math.isclose(1, 0):
            print(B)
        #break
```

```
H37
[0 1 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
4

--- Processing cone 1 ---
[ 1  0  0  0  0  0  0  0]
[ 0  0  0  0  0  1  0  0]
[ 0  1  0  0  0  0  0  0]
[ 0 -1  0  0  0  0  0  0]
[ 0  0  1  0  0  0  0  0]
```

Figure 16: Run the script and the processing of the first matrices will be visualized on your screen.



```
[ 0  0  0  0  0  0  1  0]
[ 0  0  0  0  0  0 -1  0]

--- Processing cone 500 ---
[ 0  0  0  0  0  0  1  0]
[ 1  0  0  0  0  0  0  0]
[-1  0  0  0  0  0  0  0]
[ 0  1  0  0  0  0  0  0]
[ 0 -1  0  0  0  0  0  0]
[ 0  0  1  0  0  0  0  0]
[ 0  0 -1  0  0  0  0  0]
[ 0  0  0  1  0  0  0  0]
[ 1  0  0  0  0  0  0  0]
[-1  0  0  0  0  0  0  0]
[ 0  1  0  0  0  0  0  0]
[ 0 -1  0  0  0  0  0  0]
[ 0  0  1  0  0  0  0  0]
[ 0  0 -1  0  0  0  0  0]
[ 0  0  0  1  0  0  0  0]
[ 0  0  0 -1  0  0  0  0]
```

```
In [ ]:
```

Figure 17: Finally, if you script run correctly, it will be visualized the last matrix, in this for $n = 500$. You can change this number of simulations. Only consider the RAM on you device.

2 Technical considerations about this pipeline.

Important considerations of this pipeline: You can run the complete pipeline for genotype arithmetic via the Jupyter Notebooks provided in the following [GitHub repository link]. Here, there is a tutorial to run examples to a different pathway length with $n = 3, 4, 5, 6, 7, 8, 9$. An important remark is that the families of orthogonal cones used in the simulations have row lattice-vectors that are mutually orthogonal. You can always transform a face of any of these cones through a linear transformation. Therefore, from the 500 simulated matrices, you can select a subset of lattice-vectors that form an identity matrix. This identity matrix can then be permuted freely, since matrix permutations are also linear transformations. These facts are supported by two key theorems used in our methodology: Birkhoff's Theorem and the theorem related to magic squares (i.e., matrices whose rows and columns all sum to the same value). These theorems state that every doubly stochastic matrix can be expressed as a convex combination of permutation matrices; see [1], [2]. Based on these results, we justify the matrices presented in our application example in Table 1. Using these principles, the permuted identity matrices shown in Table 1 can be derived.

Note: The 5 step of the pipline is calculated by hand in the last row in Table 1. The constranits are calculated in agreement the formula given in (2).

References

[1] Cox D, Little J, O'Shea D. Ideals, varieties, and algorithms: an introduction to computational geometry and commutative algebra. 2nd ed. Springer; 2004.

[2] Brualdi RA. Combinatorial matrix classes. Cambridge University Press; 2010. Available from: https://doi.org/10.1017/CBO9780511721182.

[3] Kluyver, T., Ragan-Kelley, B., Fernando P'erez, Granger, B., Bussonnier, M., Frederic, J., … Willing, C. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides B. Schmidt (Eds.), Positioning and Power in Academic Publishing: Players, Agents and Agendas (pp. 87–90).

[4] William Stein and David Joyner. SAGE: *System for Algebra and Geometry Experimentation.* ACM SIGSAM Bulletin, volume 39, number 2, pages 61–64, 2005.