Introduction to SageMath

For Python and SymPy Users

Ádám Gyenge

March 17, 2025

Outline

What is SageMath?

- Open-source mathematics software system.
- Combines many existing open-source packages (Maxima, GAP, R, etc.).
- Provides a unified interface and additional functionality.
- Built using Python; enhances Python with advanced math capabilities.

History of SageMath

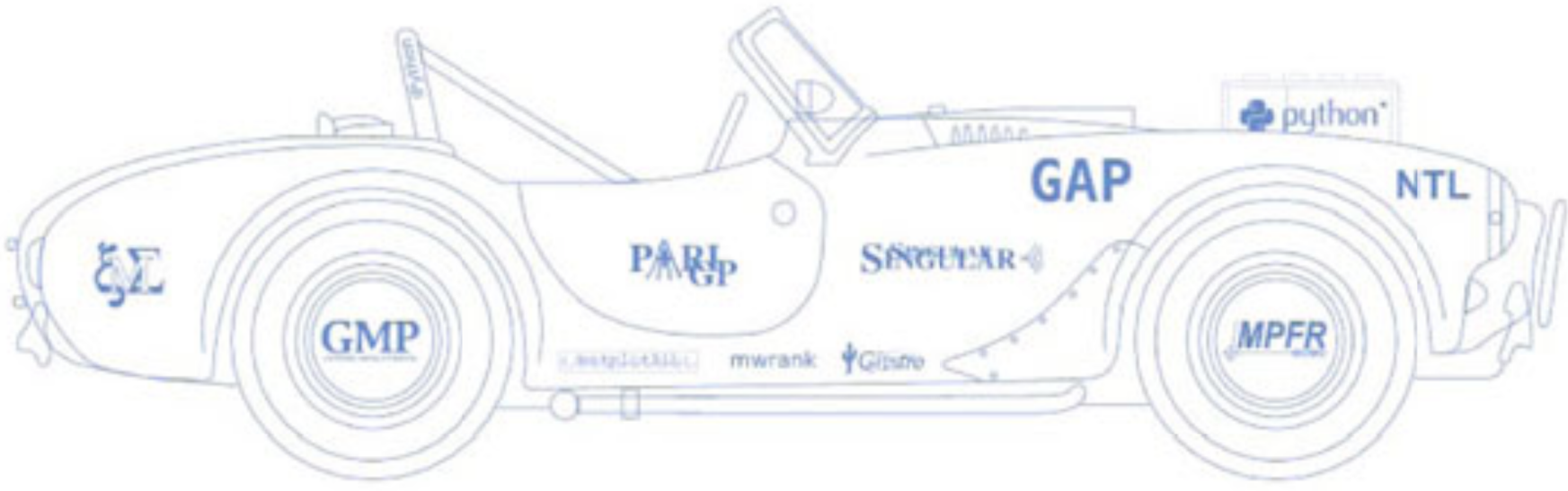
- **2004:** SageMath (originally called SAGE, "Software for Algebra and Geometry Experimentation") was created by **William Stein**, a professor at the University of Washington.
- **Mission:** To create a free, open-source alternative to proprietary mathematical software like Mathematica, MATLAB, and Maple.
- **Core Philosophy:**
  - Built on top of existing open-source libraries (e.g., NumPy, SciPy, Maxima, and R).
  - Emphasis on transparency, extensibility, and collaboration.
- **2005:** First public release (Sage 1.0).
- **2008:** Development of **SageMathCloud** (now CoCalc), expanding Sage's usability in the cloud.
- **Today:** SageMath is widely used in education, research, and industry, with a vibrant community contributing to its development.

Key Software Packages in SageMath

- **Algebra:** GAP, Maxima
- **Number Theory:** PARI/GP, FLINT
- **Geometry:** Singular, Polymake
- **Statistics:** R
- **Graph Theory:** NetworkX
- **Linear Algebra:** NumPy, SciPy

Motto of Sage

Building the car instead of reinventing the wheel.



What is CoCalc?

An All-in-One Collaborative Mathematics Platform

- **CoCalc** (formerly SageMathCloud) is an online platform designed to support **SageMath** and other computational tools.
- Enables users to perform mathematical computations, write documents, and collaborate in real time.
- Key features:
  - Access **SageMath** seamlessly in the cloud.
  - Collaborative editing of Jupyter notebooks, LaTeX documents, and more.
  - Integrated support for Python, R, Julia, and other programming languages.
  - Version control for tracking changes.
- **No Installation Needed:** Work directly in your web browser with computational resources provided by CoCalc servers.

Why Use CoCalc for SageMath?

- **Ease of Use:**
  - Quickly access SageMath without worrying about local setup or dependencies.
  - Intuitive interface tailored for computational mathematics.
- **Collaboration Tools:**
  - Real-time document sharing for joint problem-solving and research.
  - Integrated chat and commenting system for effective teamwork.
- **Rich Ecosystem:**
  - Access LaTeX for creating mathematical documents.
  - Explore Jupyter notebooks for interactive computations.
  - Use Sage Worksheets for a native SageMath experience.
- **Flexible Resources:**
  - Scale computational resources based on project needs (free and paid plans available).
  - Ideal for educational settings with students needing easy access to SageMath.

Sage from the Python programmer's viewpoint

- SageMath enhances Python syntax with mathematical constructs.
- In the background, When Sage loads a file, it converts it to Python, which is then executed by the Python interpreter.
- File extension: `.sage` instead of `.py`
- Sage can also be used in Jupyter.
- SymPy: 10 Mb
- Sage: 1 Gb

Syntax differences

- Power:
  Python: `2**3`
  Sage: `2^3`
- Symbolic variables are defined using `var()`.
  Python: `x = sympy.symbols('x')`
  Sage: `var('x')`
- Mathematical operations are more intuitive:
  Python: `sympy.sin(x)`
  Sage: `sin(x)`
- When used command-line, the prompt starts with
  Python: `>>>`
  Sage: `sage:`
- A bunch of packages are loaded automatically.

Data types

Sage adds many types to the Python built-in types. E.g. vector spaces:

```
sage: V = VectorSpace(QQ, 1000000); V
Vector space of dimension 1000000 over Rational Field

sage: type(V)
<class 'sage.modules.free_module.
   FreeModule_ambient_field_with_category'>
```

Simplifying Expressions

Python (SymPy):

```
from sympy import symbols, simplify
x = symbols('x')
expr = x**2 - 2*x + 1
simplify(expr)
```

SageMath:

```
var('x')
expr = x^2 - 2*x + 1
simplify(expr)
```

Solving Equations

Python (SymPy):

```
from sympy import Eq, solve
solve(Eq(x**2 - 2, 0), x)
```

SageMath:

```
solve(x^2 - 2 == 0, x)
```

Calculus

Python (SymPy):

```
from sympy import diff, integrate, sin, exp
diff(sin(x), x)
integrate(exp(x), x)
```

SageMath:

```
diff(sin(x), x)
integrate(e^x, x)
```

Limits

Python (SymPy):

```
from sympy import limit
limit(sin(x)/x, x, 0)
```

SageMath:

```
limit(sin(x)/x, x=0)
```

Linear Algebra

```
# Define a matrix
A = Matrix([[1, 2], [3, 4]])
# Compute determinant
det = A.determinant()
# Eigenvalues
eigenvals = A.eigenvalues()
print(det, eigenvals)
```

Output:

- Determinant: -2
- Eigenvalues: $\{5.37, -0.37\}$ (approx.)

Polynomials

```
R.<x> = PolynomialRing(QQ)
f = x^3 - 3*x^2 + 4*x - 2
factors = f.factor()
print(factors)
```

Output:

- Factors: $(x-1)^2(x+2)$

Solving Equations

Solving Equations

- Solve algebraic equations symbolically or numerically.
- Example: Solving a quadratic equation

Code Example:

```
var('x')
eq = x^2 - 4*x + 3 == 0
solutions = solve(eq, x)
print(solutions)
```

Output:

- Solutions: $x = 1, x = 3$

Rings

- Sage supports abstract algebraic structures, including rings and modules.
- Example: Creating a ring and checking properties

Code Example:

```
R = Integers(12)  # Ring of integers modulo 12
print(R.is_commutative())  # Check if commutative
print(R(5) * R(7))         # Perform multiplication
```

Output:

- Commutative: True
- Multiplication: 11 (mod 12)

Plotting in SageMath

- The default plotting method in uses the **matplotlib** package
- 2D Plot: `plot(sin(x), (x, -2*pi, 2*pi))`
- 3D Plot: `plot3d(x^2 + y^2, (x, -2, 2), (y, -2, 2))`
- Combine plots: `plot(f) + plot(g)`
- Other plotting libraries can also be used, if installed.
- Example: **gnuplot**

```
sage: maxima.plot2d('sin(x)','[x,-5,5]')
```

Rings

- Sage supports ring structures: `ZZ`, `QQ`, `RR`, `CC`.
- Operations: `ZZ(3) + ZZ(5)`.
- Modular arithmetic: `Mod(7, 3)`.

Polynomials

- Define a polynomial ring: `R.<x> = PolynomialRing(QQ)`.
- Operations: Addition, multiplication, division.
- GCD: `gcd(p1, p2)`.

Example: Factorize $x^2 - 2$.

Group Theory

Invokes the software package **GAP** in the backcground.

- Access group libraries: `groups.permutation`.
- Create groups: `AbelianGroup()`, `SymmetricGroup()`.
- Group operations: Multiplication, identity, inverses.

Abelian Groups and Direct Sums

Definition

- A group G is *abelian* if $gh = hg$ for all $g, h \in G$.
- The *direct sum* $G \oplus H$ of two groups G and H is the Cartesian product

$$G \times H = \{(g, h) : g \in G, h \in H\}$$

equipped with the group operation

$$(g_1, h_1) \times (g_2, h_2) = (g_1 g_2, h_1 h_2)$$

and unit

$$(1_G, 1_H).$$

Example with SageMath

```
sage: G = AbelianGroup([2, 3])  # Z/2Z x Z/3Z
sage: print(G)
Multiplicative Abelian group isomorphic to C2 x C3

sage: print(G.is_commutative())  # Check if G is abelian
True

# Elements of G
sage: for g in G: print(g)
1
f1
f1^2
f0
f0*f1
f0*f1^2
```

Normal Subgroups and Quotient Groups

Definition

- A subgroup $N \leq G$ is *normal* if $gNg^{-1} = N$ for all $g \in G$.
- The *quotient group* G/N is formed by the cosets of N in G:

$$G/N = \{gN : g \in G\}$$

with

$$g_1N \cdot g_2N = g_1g_2N.$$

Example with SageMath:

```
G = SymmetricGroup(3)  # S_3
N = G.subgroup([(1, 2)])  # A subgroup of S_3
print(N.is_normal(G))  # Check if N is normal

# Quotient group
Q = G.quotient(N)
print(Q)
```

Permutation Groups

Definition

A *permutation group* acts on a set by permuting its elements.

Example with SageMath:

```
# Define a permutation group generated by cycles
G = PermutationGroup([(1, 2, 3), (4, 5)])
print(G)

# Properties of G
print("Order:", G.order())
print("Generators:", G.gens())

# Permute an element
sigma = G.gens()[0]
print(sigma(2))
```

Introduction to Graphs in SageMath

- SageMath provides built-in support for graphs and graph algorithms.
- Graphs can be defined using adjacency lists, matrices, or predefined structures.
- Example: Creating a simple graph in SageMath:

```
G = Graph({1: [2, 3], 2: [3, 4], 3: [4], 4: []})
G.show()
```

Graph Properties and Visualization

- SageMath allows computation of various graph properties:

```
G.is_connected()   # Check if the graph is connected
G.degree(1)        # Get the degree of vertex 1
G.diameter()       # Compute the graph's diameter
```

- Graphs can be visualized using different layouts:

```
G.show(layout='circular')
```

Algorithms on Graphs

- SageMath includes many graph algorithms, such as shortest paths and spanning trees.
- Example: Computing shortest path using Dijkstra's algorithm:

```
G.shortest_path(1, 4)
```

- Finding a minimum spanning tree:

```
T = G.minimum_spanning_tree()
T.show()
```

Graph Coloring and Applications

- Graph coloring is useful in scheduling, register allocation, and networking.
- SageMath can compute chromatic numbers and colorings:

```
G.chromatic_number()
G.coloring()
```

- Example: Coloring a graph:

```
G.plot(vertex_colors=G.coloring())
```

Advanced Features

- Parallel computing: `parallelize`.
- Custom algorithms: Write Python functions in Sage.
- Interfaces to external software (e.g., R, Maxima).

Using SageMath in Python

All functionalities of Sage can be accessed directly from Python.

- Install Sage library: `pip install sagemath`.
- Import Sage: `from sage.all import *`.
- Use Sage objects and methods in Python scripts.

Demo: Solve $x^2 - 2 = 0$ in a Python script using Sage.

Conclusion

- SageMath extends Python for advanced math tasks.
- Unified access to multiple software packages.
- Easy integration with Python workflows.

Next Steps: Install SageMath and try out examples!