LaTeX: A Guide for the Curious Mathematician

Nishtha Tikalal

Contents

Preface and Introduction: Why SageMath?

Mathematics is a vast and evolving field, and having the right tools to explore, compute, and visualize mathematical concepts can make a profound difference in learning and research. SageMath is a free, open-source mathematics software system designed to provide a comprehensive platform for all levels of mathematical computation.

What is SageMath?

SageMath (also called Sage) is a powerful software system that brings together many well-established open-source mathematics packages under a common interface. It provides capabilities spanning:

- Arithmetic: integers, rationals, real and complex numbers.
- Algebra: polynomial rings, factorization, group theory.
- Calculus: differentiation, integration, limits, series.
- Linear Algebra: matrices, vectors, eigenvalues, solution of systems.
- Number Theory, Combinatorics, Graph Theory, and more.
- Statistical and numerical computation.
- Visualization: 2D and 3D plotting.
- Programming: based on Python, with support for custom functions and algorithms.

SageMath aims to consolidate and unify these functions into a single system that is accessible, extensible, and suitable for both education and advanced research.

Main Features

- **Free and Open Source:** SageMath is available free of charge under the GPL license, fostering collaboration and continuous improvement.
- **Integrated Environment:** Combines dozens of mature mathematical libraries, including GAP, NumPy, SciPy, Maxima, matplotlib, and more.
- **Python-Based:** Uses the Python programming language, making it familiar and extensible for users with programming experience.
- **Versatile Interfaces:** Use SageMath via command-line, Jupyter notebooks, or the web-based CoCalc platform, among others.
- **Mathematical Typesetting:** Supports output in LaTeX format for professional-quality documents and presentations.
- **Visualization:** Create detailed 2D and 3D plots for functions, data, and geometric objects.
- **Extensible and Programmable:** Users can write custom algorithms and interfaces.

Comparison with Other Systems

SageMath was conceived as a free alternative to proprietary software such as Magma, Maple, Mathematica, and MATLAB. Unlike these commercial products, SageMath's open development model helps avoid vendor lock-in and promotes transparency.

Its strength lies in integrating existing open-source libraries under a cohesive, Pythonic interface rather than reinventing tools from scratch. For many users, SageMath offers comparable power and flexibility without licensing costs.

Who Should Use SageMath?

SageMath suits a wide spectrum of users:

- **Students** seeking a comprehensive tool for homework, projects, and exploration.
- **Educators** who want an accessible classroom tool with powerful capabilities.
- **Researchers** needing advanced algebraic, numerical, and visualization tools.
- **Developers** interested in extending mathematical software using Python.

About This Book

This book guides you through the essential features of SageMath, starting from installation to mastering mathematical computations, visualization, and integration with LaTeX. Whether you are a curious learner or a seasoned researcher, the goal is to empower you to use SageMath effectively for your mathematical needs.

Welcome to the world of SageMath!

Chapter 1

Getting Started with SageMath

This chapter guides you through installing SageMath on your computer, introduces the basic ways to interact with SageMath, and briefly explores the popular cloud-based SageMath platform, CoCalc.

1.1 Installation

SageMath supports major operating systems including Linux, macOS, and Windows (via Windows Subsystem for Linux—WSL). There are multiple ways to install SageMath:

- **Using Package Managers:** On Linux distributions like Debian, Ubuntu, and Arch Linux, SageMath is often available via the system package manager, making installation quick and easy.
- **Pre-built Binary Downloads:** For macOS and Windows, pre-built installers or binary tarballs can be downloaded and installed without compiling from source.
- **Conda-Forge:** SageMath can be installed into a Conda environment using the conda-forge channel, which helps manage dependencies automatically.
- **Building from Source:** Advanced users can download SageMath source code and compile it, which may take several hours but allows customization.
- **Cloud-based Options:** If installation is undesirable or impractical, SageMath is available through online platforms such as CoCalc (https://cocalc.com), providing instant access without local setup.

You will need at least 4 GB of free disk space, and it is recommended to have a working LaTeX installation to fully utilize SageMath's typesetting capabilities.

1.2 Starting SageMath: The Sage Console

Once installed, open a terminal or command prompt and type:

```
sage
```

This launches the SageMath interactive console with a prompt like:

```
sage:
```

You can enter Sage commands here directly. For example:

```
sage: 2 + 2
4


sage: factor(2005)
5 * 401
```

To exit, type `quit()` or press Ctrl+C twice.

1.3 Using Sage Notebook / Jupyter

SageMath can also be used in notebook interfaces:

- **Sage Notebook:** An older web-based notebook interface that can be launched using `sage -n notebook`.
- **Jupyter Notebooks:** Modern and widely used; start with `sage -n jupyter` to open a Jupyter notebook that supports SageMath kernels. This interface supports mixing executable code cells and rich text/LaTeX markup.

Jupyter notebooks are highly recommended for interactive exploration, documentation, and visualization.

1.4 CoCalc: SageMath in the Cloud

CoCalc (Collaborative Calculation) is an online service hosting SageMath, Jupyter notebooks, LaTeX documents, and more. It requires no installation and supports real-time col-

laboration.

Visit `https://cocalc.com` and create a free account to start a SageMath worksheet or Jupyter notebook immediately.

Chapter 2

Working with Numbers: Integers, Rationals, Real, and Complex Numbers

SageMath provides robust support for various classes of numbers commonly used in mathematics, including integers, rational numbers, real numbers, and complex numbers. This chapter introduces how to work with each number type, perform arithmetic, and understand their properties in SageMath.

2.1 Integers: The ZZ Ring

Integers in SageMath belong to the ring ZZ. You can create and manipulate integers simply by typing numbers:

```
sage: a = 42           # Integer assignment
sage: type(a)
<class 'sage.rings.integer.Integer'>

sage: b = a + 8        # Integer arithmetic
sage: b
50

sage: a.factor()
2 * 3 * 7
```

Integers support arithmetic operations, factorization, divisibility tests, gcd, and lcm computations.

2.2 Rational Numbers: The QQ Field

Rational numbers are fractions of integers represented as elements of the field QQ in SageMath:

```
sage: r = QQ(3/4)
sage: r
3/4
sage: s = QQ(5)
sage: r + s
23/4

sage: r * (1/2)
3/8

sage: r.numerator()
3

sage: r.denominator()
4
```

Rationals are automatically reduced to lowest terms, and all arithmetic operations preserve rational output where possible.

2.3 Real Numbers: The RR Field

Real numbers in SageMath have arbitrary precision and belong to the real field RR:

```
sage: x = RR(3.14159)   # Approximate real number
sage: y = RR('2.71828') # String input to maintain precision
sage: x + y
5.85987?
sage: x.n(digits=10)    # Print with 10-digit precision
3.141590000
```

SageMath supports symbolic manipulation of real numbers as well as numerical approximations at adjustable precision.

2.4 Complex Numbers: The CC Field

Complex numbers consist of real and imaginary components and are elements of the complex field CC in SageMath:

```
sage: z = CC(1 + 2*I)
sage: z
1.00000000000000 + 2.00000000000000*I

sage: z.real()
1.0

sage: z.imag()
2.0

sage: z.conjugate()
1.0 - 2.0*I

sage: abs(z)  # Magnitude
2.23606797749979
```

SageMath supports arithmetic, polar forms, and other complex analysis functions.

2.5 Converting Between Number Types

SageMath allows explicit conversion between types when meaningful:

```
sage: ZZ(5), QQ(5), RR(5), CC(5)
(5, 5, 5.00000000000000, 5.00000000000000 + 0.00000000000000*I)

sage: RR(QQ(2)/QQ(3))
0.666666666666667?
```

Conversion may raise errors if it is not mathematically valid, such as converting certain symbolic expressions to numbers.

2.6 Visualizing Numbers with Plots

To help understand the different types of numbers discussed, the following plots provide visual insight.

2.6.1 Rational Approximations of π

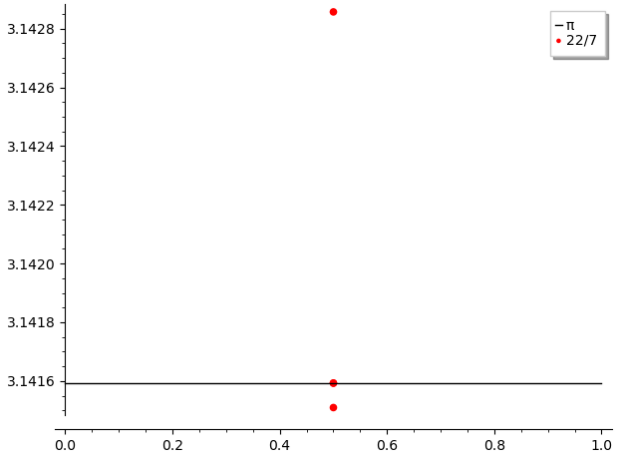

Figure 2.1: Rational approximations of π as points on the number line.

2.6.2 Plot of the Real-Valued Function $f(x) = \sin(x)$

2.6.3 Complex Numbers on the Complex Plane

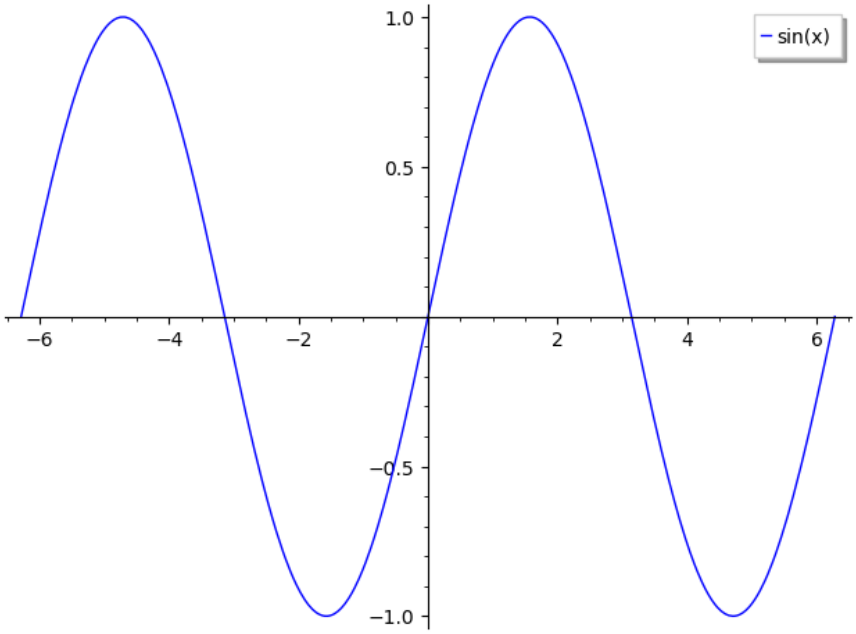

Figure 2.2: Plot of the function $f(x) = \sin(x)$ over $[-2\pi, 2\pi]$.

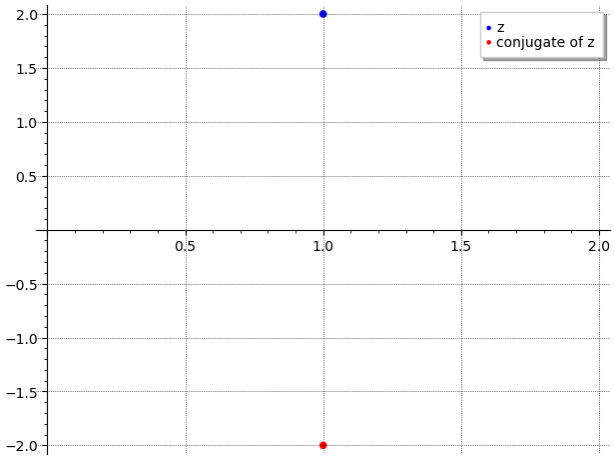

Figure 2.3: Points $z = 1 + 2i$ and its conjugate plotted on the complex plane.

Chapter 3

Variables and Expressions: Defining Variables and Algebraic Operations

SageMath allows symbolic representation and manipulation of variables and algebraic expressions, making it a powerful tool for exploring mathematical concepts.

3.1 Defining Variables

To work symbolically in SageMath, you first define variables using the `var()` function.

```
sage: x = var('x')
sage: y = var('y')
sage: z = var('z')
```

You can also define multiple variables at once:

```
sage: x, y, z = var('x y z')
```

Once defined, these variables can be used to build algebraic expressions.

3.2 Constructing and Manipulating Expressions

You can create algebraic expressions by combining variables with arithmetic operations:

```
sage: expr1 = x^2 + 2*x + 1
sage: expr2 = (x + 1)^2
```

SageMath understands that `expr1` and `expr2` represent the same polynomial:

```
sage: expr1 == expr2
True
```

3.3 Expression Simplification

SageMath provides various functions to simplify or manipulate expressions:

- `expand()`: Expands products and powers.
- `factor()`: Factors polynomials.
- `simplify()`: Simplifies expression algebraically.

Example:

```
sage: expr = (x + 1)^3
sage: expanded = expr.expand()
sage: factored = expanded.factor()
sage: simplified = (expr/(x + 1)).simplify()
```

3.4 Substitution in Expressions

You can substitute values or other expressions for variables using the `substitute()` method:

```
sage: expr = x^2 + y
sage: expr.substitute(x=2)
4 + y

sage: expr.substitute(x=y + 1)
(y + 1)^2 + y
```

3.5 Combining Expressions and Numeric Evaluation

Expressions can combine symbolic and numeric values. To evaluate numerically (e.g., at a decimal point), use `n()` or `numerical_approx()`:

```
sage: expr = sin(x) + 1/3
sage: expr.substitute(x=pi/4).n()
1.040573
```

3.6 Visualizing Expressions with Plots

To help visualize how variables and expressions behave, the following plots demonstrate some important concepts.

3.6.1 Plot of the Expression $(x+1)^3$

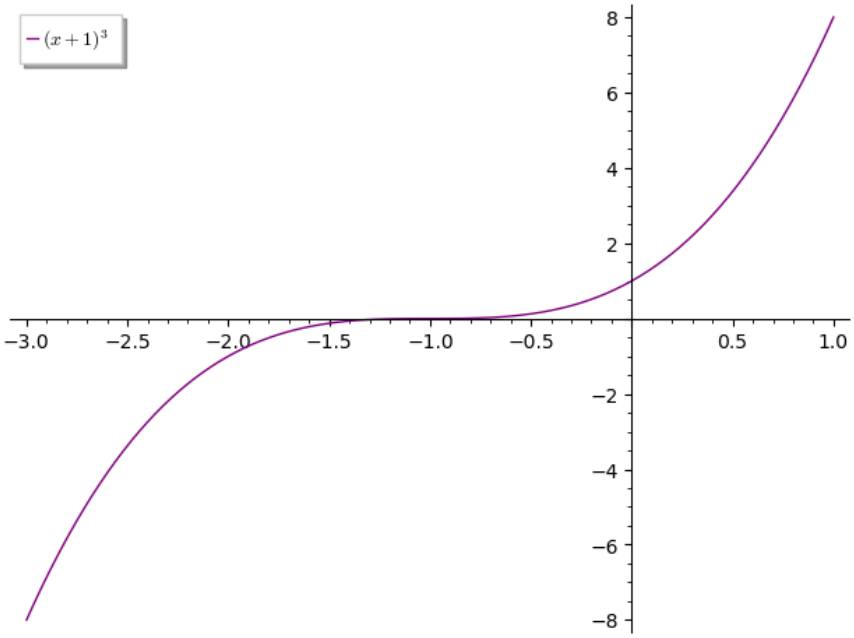

Figure 3.1: Plot of the cubic expression $(x+1)^3$ over the interval $[-3, 1]$.

3.6.2 Plot Showing Substitution Effects on $\sin(x) + \frac{1}{3}$

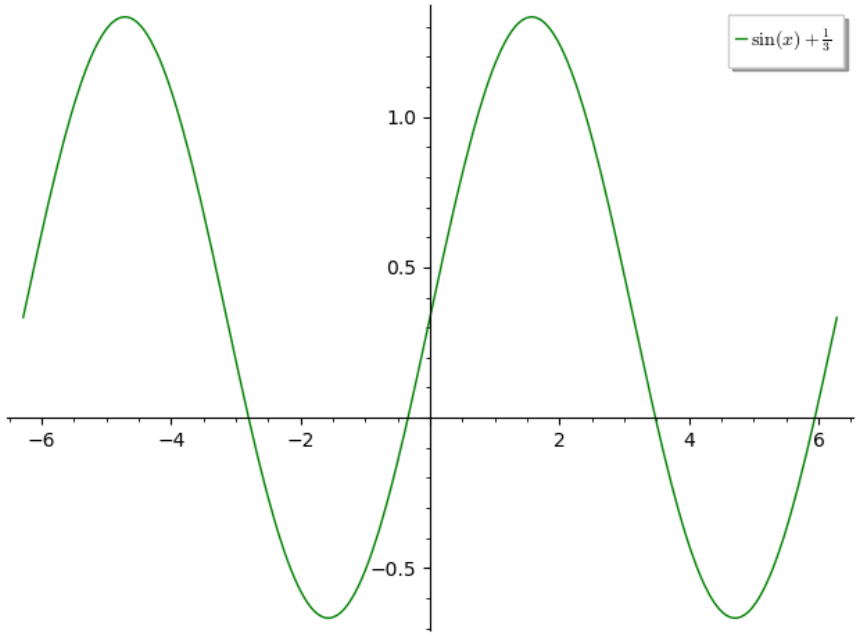

Figure 3.2: Plot of $\sin(x) + \frac{1}{3}$ over $[-2\pi, 2\pi]$, illustrating numeric shifts due to substitution.

3.7 Exercises

- Define variables a, b, c and construct the expression $a^2 + b^2 - c^2$.
- Expand and factor the expression $(x + 2)^3$.
- Substitute $x = 3$ and $y = 5$ into $x^2 + y^2$.
- Simplify the expression $\frac{(x^2-1)}{(x-1)}$ for $x \neq 1$.

Chapter 4

Functions and Calculus: Differentiation, Integration, and Function Plotting

Calculus is a fundamental branch of mathematics concerned with change and accumulation. SageMath allows symbolic and numeric calculus calculations with ease and provides powerful visualization tools.

4.1 Defining Functions

Functions in SageMath can be defined symbolically using variables:

```
sage: x = var('x')
sage: f = sin(x)^2 + cos(x)
```

Or using the `function()` construct, useful for defining dependent functions:

```
sage: f = function('f', x)
```

4.2 Differentiation

SageMath supports differentiation via the `diff()` command. For example, differentiating $\sin(x)$:

```
sage: diff(sin(x), x)
cos(x)
```

Higher-order derivatives:

```
sage: diff(sin(x^2), x, 4)
16*x^4*sin(x^2) - 48*x^2*cos(x^2) - 12*sin(x^2)
```

Partial derivatives for multivariable functions:

```
sage: x, y = var('x y')
sage: f = x^2 + 17*y^2
sage: f.diff(x)
2*x
sage: f.diff(y)
34*y
```

4.3 Integration

Indefinite integrals are computed with `integral()` or `integrate()`:

```
sage: integral(cos(x), x)
sin(x)
```

Definite integrals with limits:

```
sage: integral(cos(x), x, 0, pi/2)
1
```

SageMath handles symbolic and numerical integration depending on the integrand.

4.4 Function Plotting

Plotting functions helps visualize their behavior and is done using `plot()`:

```
sage: plot(sin(x), (x, -2*pi, 2*pi))
```

Add styles such as color, legend, and fill:

```
sage: plot(cos(x), (x, 0, pi/2), fill=True, color='red', legend_label='cos(x)')
```

Multiple plots can be combined by adding plot objects.

4.5 Example: Derivative and Plot of $f(x) = \sin(x)^2$

```
sage: f = sin(x)^2
sage: f_prime = diff(f, x)
sage: p1 = plot(f, (x, 0, 2*pi), color='blue', legend_label='sin(x)^2')
sage: p2 = plot(f_prime, (x, 0, 2*pi), color='green', legend_label="Derivative")
sage: combined_plot = p1 + p2
combined_plot.show()
```

4.6 Visualizing Calculus Concepts with Plots

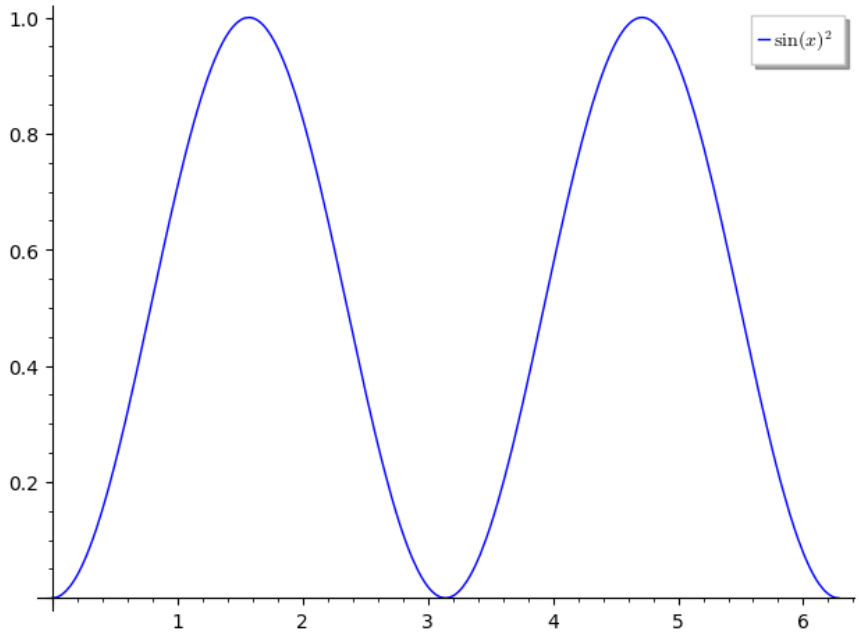

Figure 4.1: Plot of the function $\sin(x)^2$ over the interval $[0, 2\pi]$.

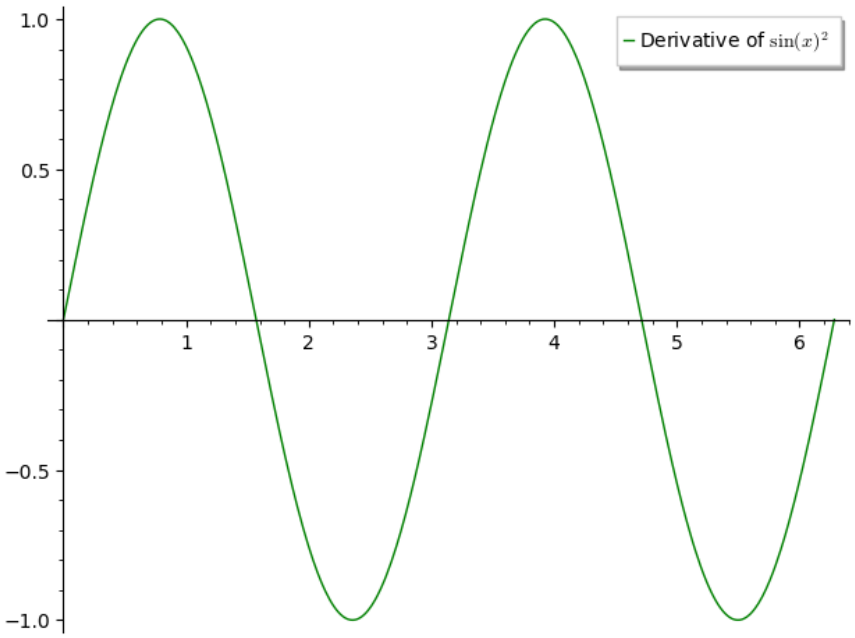

Figure 4.2: Plot of the derivative of $\sin(x)^2$ over $[0, 2\pi]$.

4.7 Exercises

- Compute and plot the first and second derivatives of e^{x^2}.
- Calculate the definite integral of $\cos(x)$ from 0 to π.
- Define $f(x, y) = x^2y + y^3$, compute $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$.
- Plot the function $\sin(x) + \cos(y)$ over $x \in [-\pi, \pi], y \in [-\pi, \pi]$.

Chapter 5

Linear Algebra: Matrices, Vectors, and Solving Systems

Linear algebra studies vectors, matrices, and the systems they form. SageMath provides comprehensive tools to create, manipulate, and solve problems involving these fundamental objects.

5.1 Matrices

Matrices in SageMath are created by passing a list of lists (rows):

```
sage: A = matrix([[1, 2, 3],
                  [3, 2, 1],
                  [1, 1, 1]])
```

You can specify the ring (number system) over which the matrix is defined:

```
sage: B = matrix(QQ, [[1, 2],
                      [3, 4],
                      [5, 6]])
```

5.2 Vectors

Vectors are one-dimensional arrays:

```
sage: v = vector([1, 1, -4])
```

Matrix and vector multiplication behaves as expected:

```
sage: A * v
(-9, 1, -2)
```

5.3 Solving Linear Systems

To solve $AX = Y$ for X, use:

```
sage: Y = vector([0, -4, -1])
sage: X = A.solve_right(Y)
sage: X
(-2, 1, 0)
```

You can check the solution:

```
sage: A * X == Y
True
```

5.4 Matrix Operations

SageMath supports common linear algebra operations such as:

- Reduced row echelon form: `A.echelon`$form()Kernel(nullspace)$: `A.kernel()`
- Transpose: `A.transpose()`
- Inverse (if invertible): `A.inverse()`
- Determinant: `A.determinant()`

5.5 Visualizing Matrices and Vectors

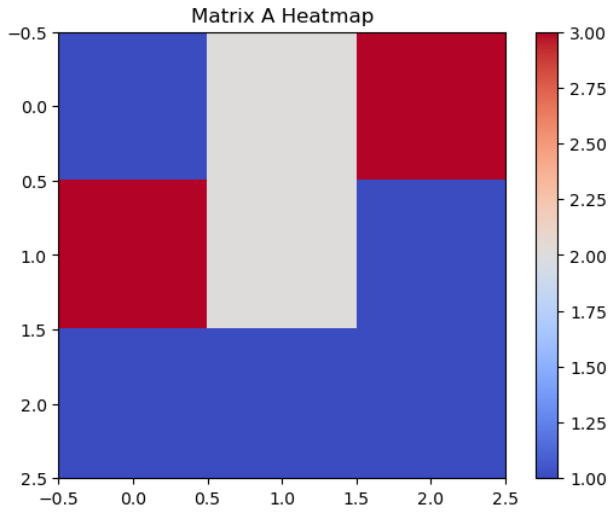

Figure 5.1: Heatmap visualization of matrix entries for matrix A.

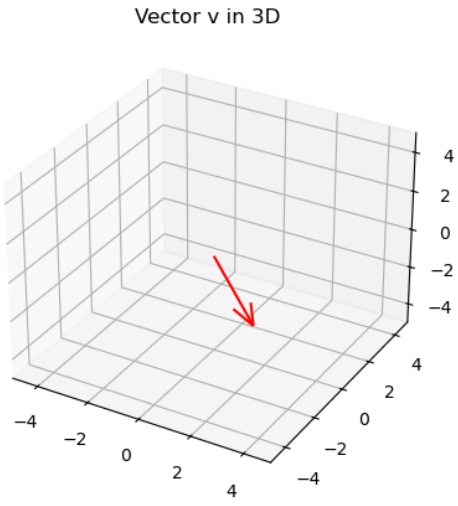

Figure 5.2: Plot of vector v in 3D space.

5.6 Exercises

- Create a 3×3 matrix A and compute its determinant and inverse (if it exists).
- Define a vector v with three elements, then compute the product Av.
- Solve the system $AX = Y$ for a given Y.

- Compute the kernel of A and interpret it.
- Compute the reduced row echelon form of A.

Chapter 6

Graphics in SageMath: 2D and 3D Plotting

Visualization is a powerful aid in understanding mathematical concepts. SageMath offers versatile plotting facilities, including two- and three-dimensional graphics, interactive plots, implicit plots for equations, visualization of Platonic solids, and contour plots.

6.1 2D Function Plotting

Basic 2D plotting uses the `plot()` function. For example:

```
sage: plot(sin(x), (x, -2\pi, 2\pi), color='blue')
```

Multiple functions can be combined by adding plots:

```
sage: p1 = plot(sin(x), (x, -2\pi, 2\pi), color='blue')
sage: p2 = plot(cos(x), (x, -2\pi, 2\pi), color='red')
sage: show(p1 + p2)
```

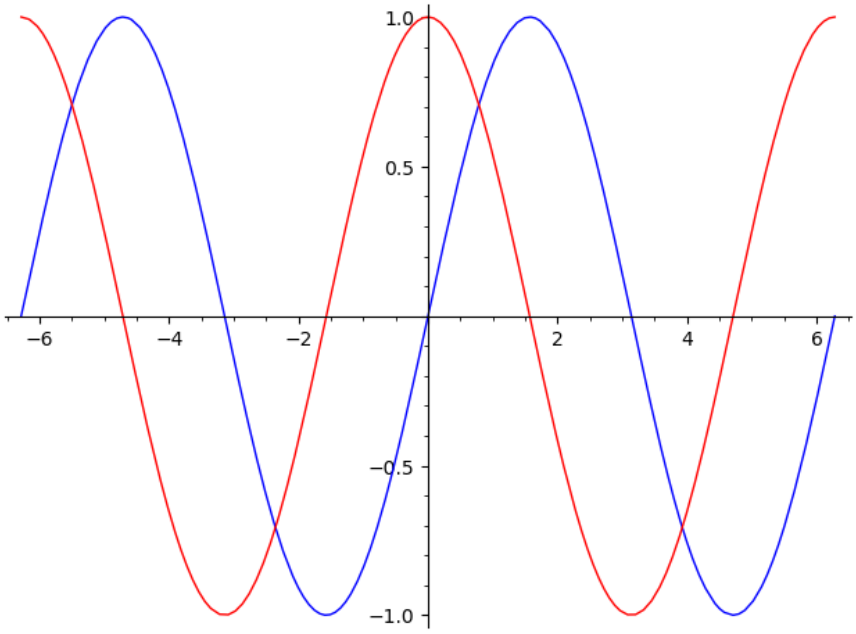

Figure 6.1: Combined plot of $\sin(x)$ and $\cos(x)$ in 2D.

6.2 Interactive Plots

You can create interactive plots with sliders using the `@interact` decorator:

```
@interact
def plot_function(f=input_box(x^2, label="Function f"),
                  a=slider(-5,0,1), b=slider(0,5,1)):
    show(plot(f, (x, a, b)))
```

6.3 Implicit Plots

SageMath supports plotting implicit curves defined by equations:

```
implicit_plot(x^2 + y^2 == 1, (x, -1.5, 1.5), (y, -1.5, 1.5))
```

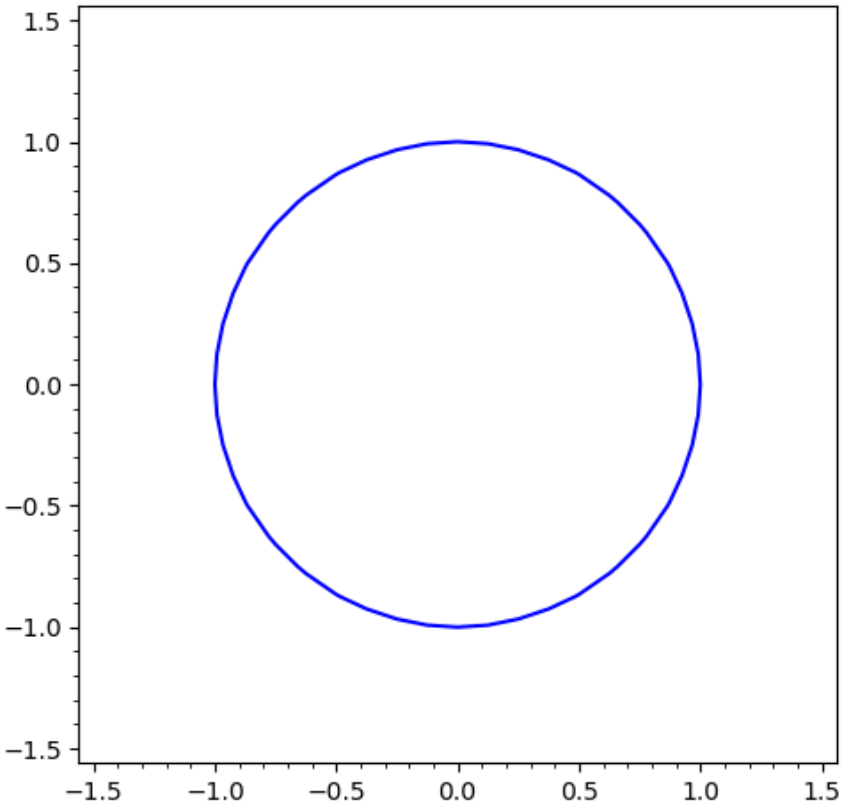

Figure 6.2: Implicit plot of the circle defined by $x^2 + y^2 = 1$.

6.4 3D Plotting

Plot surfaces with `plot3d`:

```
plot3d(sin(x*y), (x, -3, 3), (y, -3, 3))
```

And parametric 3D curves:

```
parametric_plot3d((cos(t), sin(t), t), (t, 0, 4\pi))
```

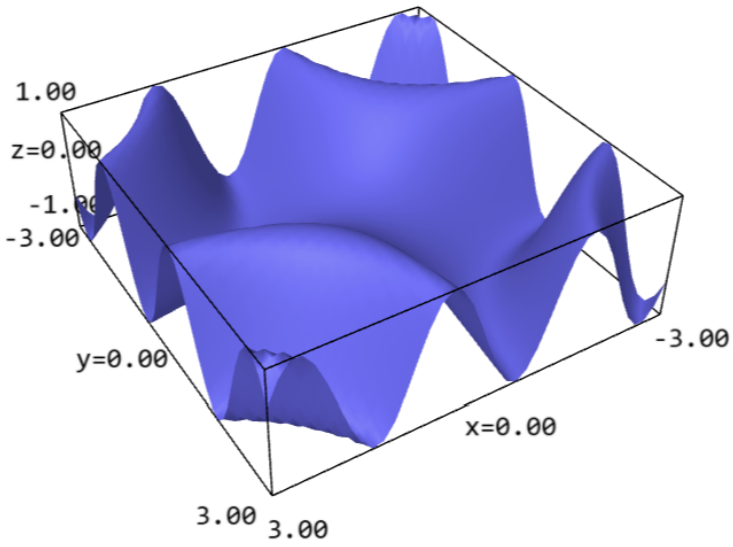

Figure 6.3: 3D surface plot of $\sin(xy)$.

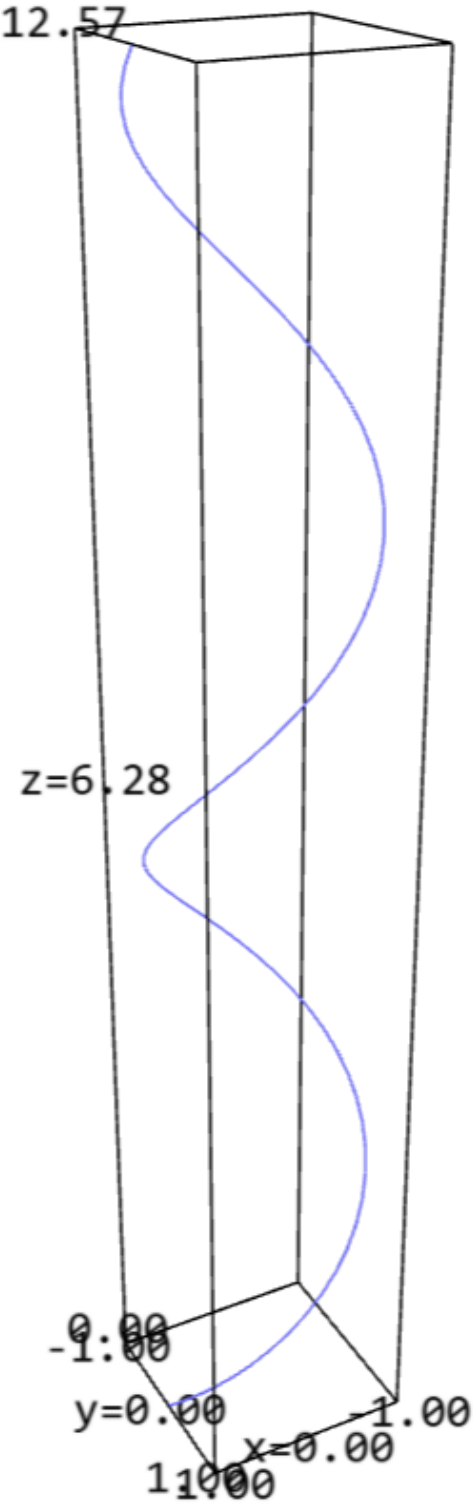

Figure 6.4: 3D parametric plot of a helix $(\cos(t), \sin(t), t)$.

6.5 Platonic Solids

Visualize Platonic solids easily:

```
tetrahedron().show()
cube().show()
dodecahedron().show()
```

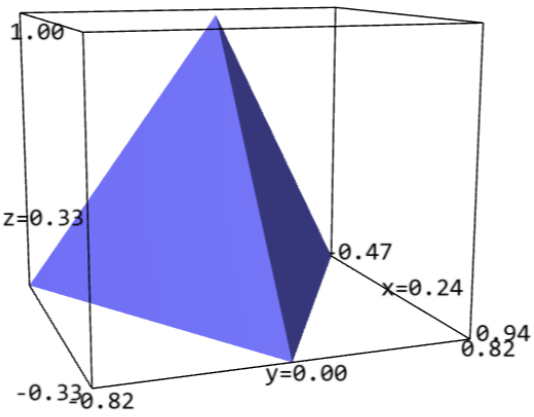

Figure 6.5: Visualization of the tetrahedron, a Platonic solid.

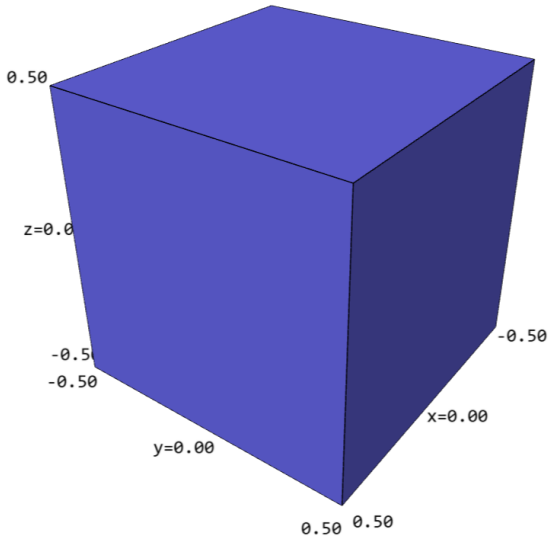

Figure 6.6: Visualization of the cube, a Platonic solid.

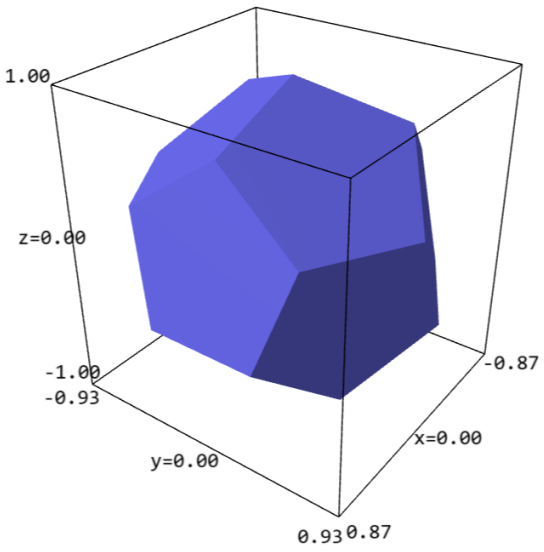

Figure 6.7: Visualization of the dodecahedron, a Platonic solid.

6.6 Contour Plots

Contour plots display level curves of functions of two variables:

```
contour_plot(sin(x)*cos(y), (x, -\pi, \pi), (y, -\pi, \pi))
```

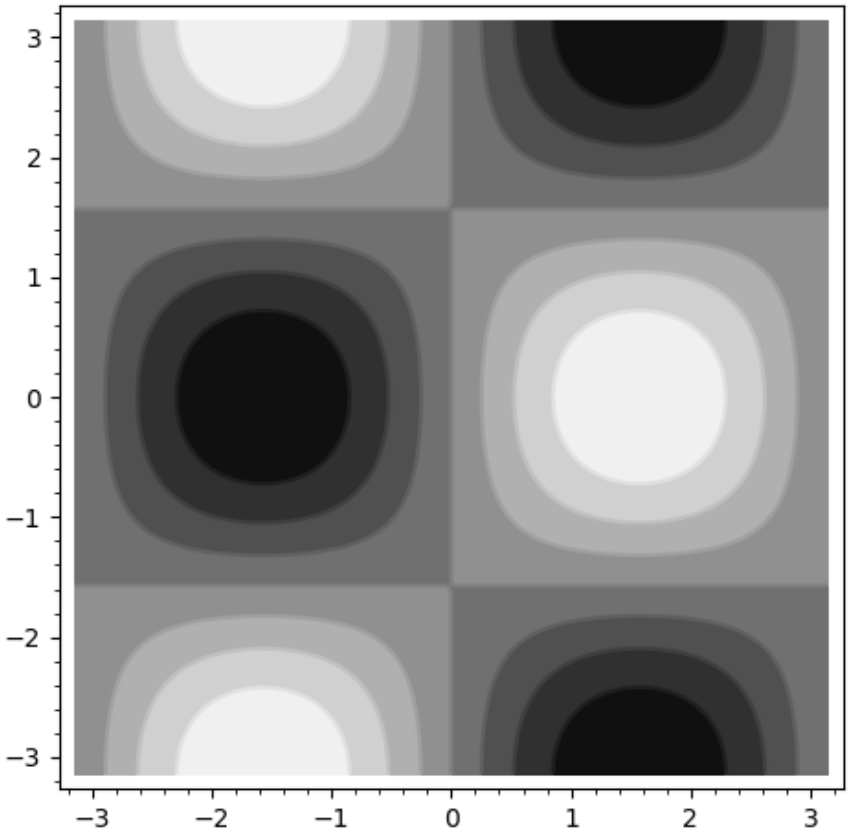

Figure 6.8: Contour plot of $\sin(x)\cos(y)$.

Chapter 7

Polynomials and Abstract Algebra

Polynomials are foundational objects in algebra, forming the basis for many algebraic structures. SageMath offers extensive support for defining, manipulating, and factoring polynomials as well as exploring rings and fields in abstract algebra.

7.1 Polynomial Rings and Definitions

Define a univariate polynomial ring over the rational numbers with an indeterminate x:

```
sage: R.<x> = PolynomialRing(QQ)
```

Construct a polynomial:

```
sage: f = x^3 - 2*x^2 + x - 1
```

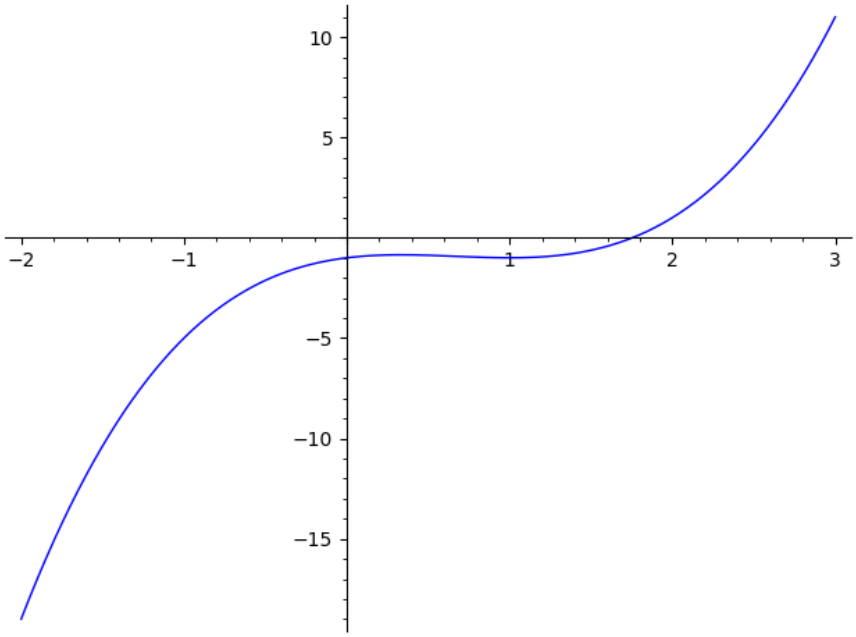

Figure 7.1: Graph of the polynomial $f(x) = x^3 - 2x^2 + x - 1$.

Multivariate polynomial rings can be defined similarly:

```
sage: S.<x,y,z> = PolynomialRing(QQ, 3)
sage: g = x*y + y^2 - z
```

7.2 Polynomial Arithmetic and Factorization

Perform arithmetic operations naturally:

```
sage: h = f * g
sage: h.expand()
```

Factor polynomials:

```
sage: f.factor()
```

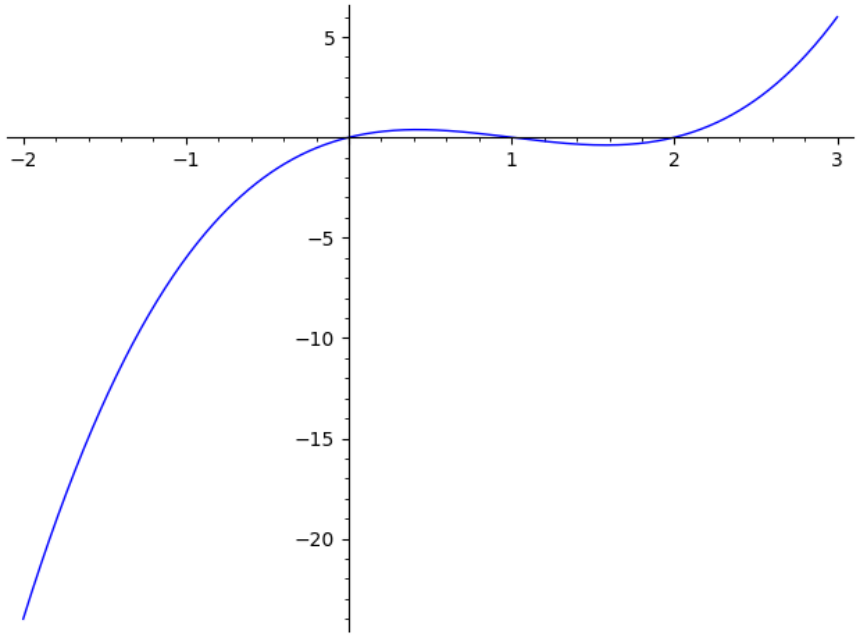

Figure 7.2: Factorization of $f(x) = x^3 - 2x^2 + x - 1$ into irreducible factors.

7.3 Roots of Polynomials

Find roots symbolically or numerically:

```
sage: f.roots(ring=CC)  # Complex roots numerically
sage: f.roots(ring=QQbar)  # Algebraic roots symbolically
```

Roots come with multiplicities and are returned as pairs.

7.4 3D Visualizations of Polynomials

Explore polynomial surfaces and parametric curves with 3D plotting:

```
var('x y')
g = x^3 - 2*x^2 + x - 1 + y^2
plot3d(g, (x, -2, 3), (y, -3, 3))
```

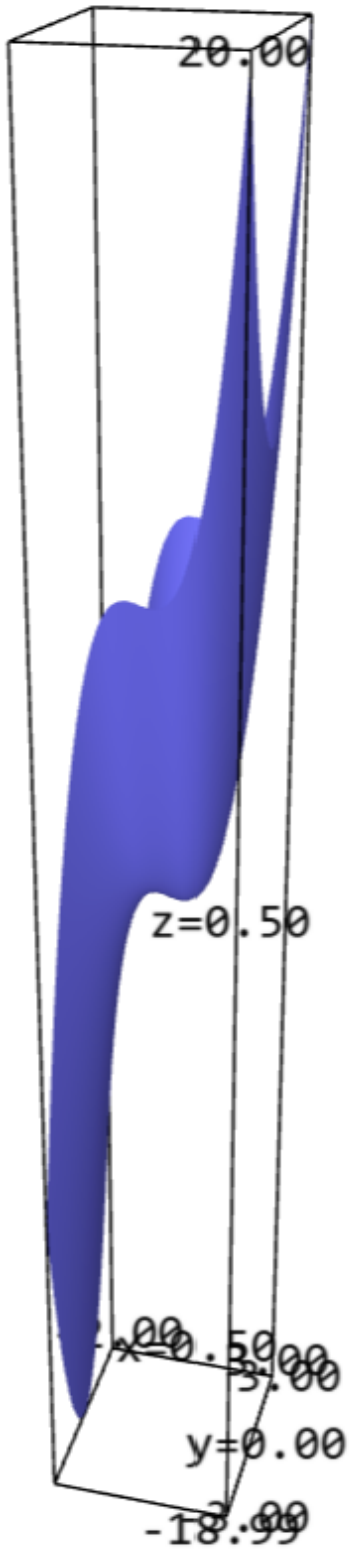

Figure 7.3: 3D surface plot of the polynomial $g(x, y) = x^3 - 2x^2 + x - 1 + y^2$.

Parametric 3D polynomial curves:

```
t = var('t')
parametric_plot3d((t, t^2, t^3), (t, -2, 2))
```

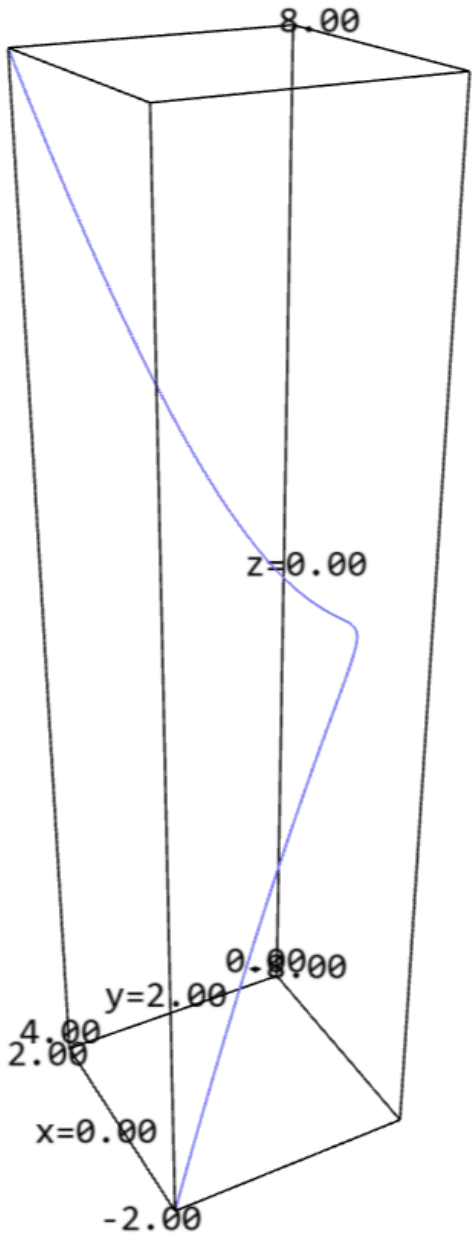

Figure 7.4: Parametric 3D plot of the twisted cubic curve (t, t^2, t^3).

7.5 Abstract Algebra: Rings and Fields

Define rings and fields explicitly:

```
sage: Zmod17 = Integers(17)  # Finite field of size 17
sage: F.<a> = GF(2^3, 'a')   # Finite field extension
```

Explore elements and polynomial rings over these:

```
sage: R = PolynomialRing(Zmod17, 'x')
```

7.6 Exercises

- Define the polynomial $f(x) = x^4 - 5x^3 + 7x - 3$ and factor it.
- Find the roots of $x^3 - 2x + 1$ in the complex field.
- Create the polynomial ring $\mathbb{Z}/5\mathbb{Z}[x, y]$ and define $f = x^2 + y^2 + 1$.
- Multiply two polynomials in $\mathbb{Q}[x]$ and expand the result.

Chapter 8

Solving Equations: Symbolic and Numerical Approaches

Solving equations is a fundamental task in mathematics and engineering. SageMath provides powerful tools to tackle both symbolic and numerical solutions of equations, systems of equations, and inequalities.

8.1 Symbolic Equation Solving

SageMath uses the `solve()` function for symbolic solutions. Declare variables first:

```
sage: var('x y p q')
(x, y, p, q)
```

Solve single or systems of equations:

```
sage: eq1 = p + q == 9
sage: eq2 = q*y + p*x == -6
sage: eq3 = q*y^2 + p*x^2 == 24
sage: solve([eq1, eq2, eq3, p == 1], p, q, x, y)
```

This returns exact symbolic solutions:

$$\begin{bmatrix} p = 1, \quad q = 8, \\ x = -\frac{4}{3}\sqrt{10} - \frac{2}{3}, \\ y = \frac{1}{6}\sqrt{10} - \frac{2}{3} \end{bmatrix}, \quad \begin{bmatrix} p = 1, \quad q = 8, \\ x = \frac{4}{3}\sqrt{10} - \frac{2}{3}, \\ y = -\frac{1}{6}\sqrt{10} - \frac{2}{3} \end{bmatrix}$$

8.2 Numerical Solutions

Use `n()` to get numerical approximations:

```
sage: solns = solve([eq1, eq2, eq3, p == 1], p, q, x, y, solution_dict=True)
sage: [[s[p].n(30), s[q].n(30), s[x].n(30), s[y].n(30)] for s in solns]
```

Output includes floating-point approximations to 30 bits precision.

8.3 Solving Inequalities

SageMath can solve inequalities and systems involving inequalities:

```
sage: var('x y')
sage: solve([x - y >= 2, x + y <= 3], x, y)
```

The solution describes the region satisfying constraints.

8.4 Differential Equations and Laplace Transform

Symbolic solutions extend to differential equations:

```
sage: t, s = var('t s')
sage: x = function('x')(t)
sage: y = function('y')(t)
sage: f = 2*x.diff(t, 2) + 6*x - 2*y
```

Taking Laplace transforms and solving algebraic equations:

```
sage: f.laplace(t,s)
```

Then solve in the Laplace domain and apply inverse Laplace transform to get time-domain solutions.

8.5 General Tips

- Use `solution_dict=True` in `solve()` for convenience.
- Large nonlinear systems may be slow; try numerical solvers.
- Use `find_root()` for scalar numerical roots in intervals.
- For polynomial roots, use `poly.roots()` with numeric fields.

8.6 Exercises

- Solve the system $x + y = 5$ and $2x - y = 3$.
- Find numerical roots for $x^3 - 2x + 1 = 0$ in the interval $[-3, 3]$.
- Solve the inequality system $x - y \geq 2$, $x + y \leq 3$.
- Solve the differential equation $y'' + y = 0$ with initial conditions $y(0) = 1$, $y'(0) = 0$.

Chapter 9

Programming Essentials in SageMath

SageMath integrates Python programming with extensive mathematical capabilities. This chapter introduces the core programming concepts necessary for effective SageMath usage, including variables, functions, conditional statements, loops, and data structures.

9.1 Variables and Data Types

Variables store values of different types: integers, floating points, symbolic expressions, lists, and more.

```
sage: a = 5
sage: b = 2.5
sage: c = var('c')
sage: expr = a + b*c
```

Sage automatically recognizes types, allowing fluid use of symbolic and numeric calculations.

9.2 Defining Functions

Functions encapsulate computation. Define functions using Python syntax:

```
sage: def square(x):
....:     return x^2
```

Functions can be symbolic or numeric.

9.3 Conditional Statements

Control flow with `if-elif-else`:

```
sage: def sign(x):
....:     if x > 0:
....:         return 1
....:     elif x < 0:
....:         return -1
....:     else:
....:         return 0
```

9.4 Loops

Iterate with `for` and `while` loops:

```
sage: for i in range(5):
....:     print(i)

sage: i = 0
sage: while i < 5:
....:     print(i)
....:     i += 1
```

9.5 Lists and Data Structures

Lists hold ordered collections:

```
sage: L = [1, 2, 3, 4]
sage: L.append(5)
```

Other structures include tuples, sets, and dictionaries.

9.6 Working with Matrices

Matrices are fundamental for linear algebra:

```
sage: M = Matrix([[1,2],[3,4]])
sage: M.det()
```

9.7 File Handling and Scripts

You can save Sage code in `.sage` or `.py` files and run them in SageMath:

```
sage: load('my_script.sage')
```

This allows modular and reusable programming.

9.8 Exercises

- Write a function that returns factorial(n).
- Create a list of squares for numbers 1 to 10 using a `for` loop.
- Define a matrix and compute its eigenvalues.
- Write a function to check if a number is prime using conditional statements.

Chapter 10

Exporting Work: Integration with LaTeX for Publication-Quality Output

SageMath seamlessly integrates with LaTeX, allowing users to embed computations, results, and plots directly into LaTeX documents for professional publication-quality output.

10.1 Using SageTeX

The `SageTeX` package enables embedding Sage computations inside LaTeX. It computes values during document compilation, producing dynamic content.

```
\documentclass{article}
\usepackage{sagetex}

\begin{document}

Number of partitions of 1269: $\sage{number_of_partitions(1269)}$.

\begin{sageblock}
f(x) = exp(x) * sin(2*x)
\end{sageblock}

Second derivative of $f$ is:

\[
```

```
\frac{d^2}{dx^2} \sage{f(x)} = \sage{diff(f,x,2)(x)}
\]

Here's a plot of $f$ from $-1$ to $1$:

\sageplot{plot(f, -1, 1)}

\end{document}
```

10.2 Preparing for SageTeX

Copy `sagetex.sty` into your document directory or ensure TeX can find it in its path. This file comes with Sage installation.

Compile your document with:

```
pdflatex yourfile.tex
sage yourfile.sagetex.sage
pdflatex yourfile.tex
```

This workflow runs Sage on the embedded code and incorporates outputs.

10.3 Generating LaTeX Code from Sage

Sage can produce LaTeX strings from mathematical expressions using the `latex()` function:

```
sage: var('z')
sage: latex(z^12)
'z^{12}'
```

You can copy these outputs into your LaTeX source.

10.4 Including Graphics

Generate plots in Sage and export as PDFs, PNGs, or TikZ code for LaTeX inclusion:

```
sage: p = plot(sin(x), (x,0,pi))
sage: p.save('sin_plot.pdf')
```

Then include in LaTeX:

```
\includegraphics{sin_plot.pdf}
```

or for TikZ plots:

```
sage: print(p.tikz_code())
```

10.5 Best Practices

- Use SageTeX for dynamic, computation-heavy documents. - For static documents, generate figures separately and include them. - Test compilation on small documents first. - Customize LaTeX preambles as needed for package dependencies.

10.6 Exercises

- Create a LaTeX document using SageTeX that computes a series sum.
- Generate a plot in Sage and include it as a TikZ picture in LaTeX.
- Convert Sage expressions to LaTeX and write a short mathematical report.