Computing Discreetly with Sage

A Tour

Robert A. Beezer
University of Puget Sound

January 25, 2018

1 What is Sage?

- An open source system for advanced mathematics.
- An open source mathematics distribution (like Linux) with *Python* as the glue.
- A tool for learning and teaching mathematics.
- A tool for mathematics research.

Mission Statement Create a viable free open source alternative to Magma, Maple, Mathematica, and Matlab.

- Created in 2005 by William Stein.
- Free and open, GPL license.
- Includes about 100 open source packages.
- Installed: 53,230 files, 15,278,715 lines of code (v 8.1).
- Now has around 540,000 lines of new code, by several hundred mathematician-programmers.

Some of the 100 packages included:

- Groups, Algorithms, Programming (GAP) - group theory
- PARI - rings, finite fields, field extensions
- Singular - commutative algebra
- SciPy/NumPy - scientific computing, numerical linear algebra
- Integer Matrix Library (IML) - integer, rational matrices
- CVXOPT - linear programming, optimization
- NetworkX - graph theory
- Pynac - symbolic manipulation
- Maxima - calculus, differential equations

2 Basic Combinatorial Numbers

2.1 Binomial Coefficients

The number of 3-sets chosen from a 10-set, $\binom{10}{3}$.

```
binomial(10, 3)
```

The coefficients of an expansion of $(a+b)^n$.

```
var('a, b')
expr = (a+b)^10
expr.expand()
```

All of the coefficients.

```
bc = binomial_coefficients(10)
bc
```

A Python dictionary, indexed by powers of the two variables in the expansion.

```
bc[(3, 7)]
```

Tote up all of these binomial coefficients, to get 2^{10}. (Size of the power set, or the result of setting $a = 1$ and $b = 1$).

```
sum(bc.values())
```

Actual subsets of size 3 from a 10-set; one way to understand a binomial coefficient.

```
S = list("AIMS-WOMEN")
S
```

```
sub = Subsets(S, 3)
sub
```

`sub` is a "generator". We can list the possibilities.

```
sub.list()
```

We can iterate over `sub`.

```
for three in sub:
    print three
```

2.2 Catalan Numbers

```
catalan_number(8)
```

$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!}$

```
(1/9)*binomial(16, 8)
```

2.3 Bell Numbers

In honor of Eric Temple Bell.

```
bell_number(6)
```

Number of partitions of a set into disjoint non-empty sets.

```
S = list('MATHS')
part = SetPartitions(S)
part
```

```
part.list()
```

That's hard to read.

```
part[34]
```

```
len(part)
```

A double-check.

```
part.cardinality()
```

2.4 Stirling Numbers

Stirling numbers come in two flavors, "first" and "second", or "cycle" and "subset". We'll demonstrate the first.

```
stirling_number1(6, 3)
```

The number of permutations on n symbols (in cycle notation) having exactly k cycles, $\left\{ n \atop k \right\}$.

```
perm = Permutations(6)
a = perm[134]
a
```

In cycle notation.

```
a.cycle_string()
```

Now we get the trivial cycles. List length is what we want.

```
a.cycle_tuples()
```

Collect all permutations with 3 cycles.

```
three = [p for p in perm if len(p.cycle_tuples())==3]
three
```

How many?

```
len(three)
```

3 (Some) Areas of Discrete Mathematics

3.1 Graph Theory

Create graphs in a natural way:

```
harary = Graph([(0,1), (1,2), (2,3), (3,0), (1,3)])
harary
```

```
harary.plot()
```

```
harary.num_verts(), harary.num_edges()
```

```
harary.is_planar()
```

```
H = harary.hamiltonian_cycle()
H.plot()
```

```
harary.degree_sequence()
```

```
sorted(harary.degree_sequence())
```

There are many pre-defined graphs (digraphs, too):

```
graphs.
```

Constant time generation of free trees, by B. Richmond, A. Odlyzko, B.D. McKay

```
trees_iterator = graphs.trees(8)
T8 = list(trees_iterator)
T8
```

From a path to a star:

```
[tree.diameter() for tree in T8]
```

Visually:

```
graphs_list.show_graphs(T8)
```

3.2 Group Theory

Prototypical use of Sage: permutation groups. Built from the mature open source package GAP (Groups, Algorithms, Programming).

```
G = DihedralGroup(8)
G
```

```
G.list()
```

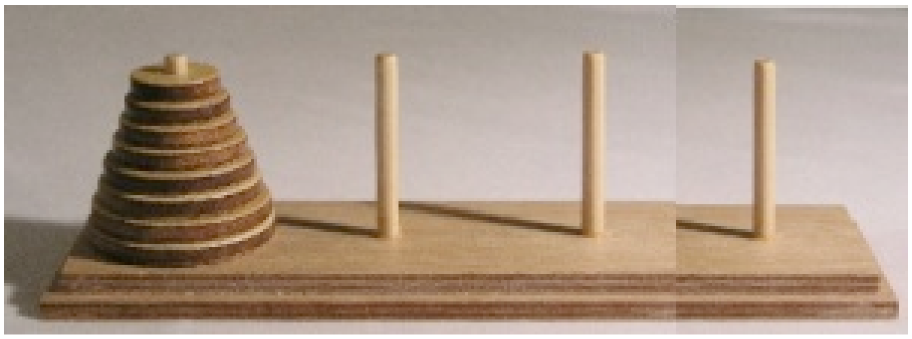
```
G.is_abelian()
```

```
sg = G.subgroups()
[H.order() for H in sg]
```

```
H = sg[14]
H.list()
```

```
H.is_normal(G)
```

3.3 Put Them Together: Tower of Hanoi



- `graphs.HanoiTowerGraph(n, d)`
- Generalize to n pegs and d disks
- State graph: intermediate configurations, edges are "one move"
- Labels: d-tuple, large disk to small disk; entries are peg numbers
- Example: $n = 3$, $d = 4$: $(2, 0, 2, 1)$

```
T = graphs.HanoiTowerGraph(3, 4, positions=True)
T.show(figsize=12)
```

A solution is path between states "all the disks on one peg" and "all the disks on another peg."

```
solution=T.shortest_path((0,0,0,0), (1,1,1,1))
solution
```

Minimum number of moves:

```
len(solution) - 1
```

```
T.diameter()
```

More general:

```
T = graphs.HanoiTowerGraph(4, 3, positions=True)
T.show(figsize=12)
```

```
T = graphs.HanoiTowerGraph(4, 4, labels=False,
    positions=True)
T.show(figsize=12)
```

Forget about graphics, work with graph itself.

```
T = graphs.HanoiTowerGraph(4, 8, labels=False,
    positions=False)
T.num_verts()
```

Code vertices to integers: d-tuples, base n. All disks on peg 0, move to all disks on peg 3.

```
solution = T.shortest_path(0, 4^8-1)
solution
```

```
len(solution)-1
```

Theorem: automorphisms of the state graph are just the obvious ones (renumber pegs)

```
T = graphs.HanoiTowerGraph(4, 6, labels=False,
    positions=False)
A = T.automorphism_group()
A.order()
```

```
S4 = SymmetricGroup(4)
S4.is_isomorphic(A)
```

Automorphisms are computed via Brendan McKay's `nauty` algorithm, once re-implemented as `NICE`.

3.4 Linear Algebra

3.4.1 Exact Linear Algebra

Many possible fields and rings: finite fields, field extensions, algebraic numbers. Over the integers and rationals powered by Integer Matrix Library (IML).

```
A = matrix(QQ,
[[1, -2, 3, 2, -1, -4, -3, 4],
[3, -2, 2, 5, 0, 6, -5, -5],
[0, -1, 2, 1, -2, -4, -1, 4],
[-3, 2, -1, -1, -6, -3, 5, 3],
[3, -4, 4, 0, 7, -7, -7, 6]])
A
```

```
A.rref()
```

```
b = vector(QQ, [2, -1, 3, 4, -3])
A.solve_right(b)
```

And it is fast. 1000×1000 matrix with single digit integer entries.

```
A = random_matrix(ZZ, 1000, 1000, x=-9, y=9)
%time A.determinant()
```

We can combine linear algebra with graph theory (aka "algebraic graph theory").

```
K = graphs.KneserGraph(8,3)
K.plot()
```

```
adj = K.adjacency_matrix()
adj
```

```
K.spectrum()
```

A small “singular graph.” (I. Sciriha, 2007)

```
S = graphs.CycleGraph(4)
S.add_vertices([4, 5, 6])
S.add_edges([(2,4), (2,5), (2,6)])
S.add_edges([(3,4), (3,5), (3,6)])
S.plot()
```

```
adj = S.adjacency_matrix()
ker = adj.kernel()
ker
```

Notice this is the kernel over the integers, and is computed as a module. It is easy to upgrade to the rationals.

```
adjQ = adj.change_ring(QQ)
kerQ = adjQ.kernel()
kerQ
```

A matrix kernel (null space) is a vector space, and has all the attendant properties.

```
kerQ.dimension()
```

```
kerQ.basis()
```

3.4.2 Numerical Linear Algebra

Numerical linear algebra is supplied by SciPy, through to LAPACK, ATLAS, BLAS.

A matrix of double-floating point real numbers (RDF).

```
B = matrix(RDF,
[[0.4706, 0.3436, 0.7156, 0.1706, 0.3863, 0.222, -0.9673],
[0.9433, -0.7333, -0.2906, -0.5203, 0.3548, 0.7577, 0.3936],
[-0.8998, 0.9269, -0.9646, -0.2294, -0.8171, 0.4568, 0.5949],
[0.8814, 0.89, -0.2059, 0.7434, -0.1642, 0.6918, 0.7113],
[-0.0034, -0.9842, 0.7213, -0.7196, -0.7422, 0.3335, 0.5829],
[-0.5676, 0.6433, -0.2296, 0.2681, 0.2992, 0.6988, 0.3332],
[0.0366, -0.5788, 0.5882, 0.1559, -0.6434, 0.871, -0.6518]])
B
```

And the QR decomposition of `B`.

```
Q, R = B.QR()
Q
```

```
(Q.conjugate_transpose()*Q).round(4)
```

```
R.round(4)
```

```
(Q*R-B).round(4)
```

3.4.3 Image Compression

```
import pylab
A_image = pylab.mean(pylab.imread('images/mystery.png'), 2)
@interact
def svd_image(i=(1,(1..194)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in
        range(i))
    # g =
        graphics_array([matrix_plot(A),matrix_plot(A_image)])
    show(matrix_plot(A), axes=display_axes, figsize=(6,8))
    html('<h2>Compressed␣using␣%s␣singular␣values</h2>'%i)
```

3.5 Simple Number Theory

Sage was born of necessity to do number theory.

```
p = next_prime(10^25)
q = next_prime(10^25+5*10^24+10^12)
m = p*q
print p, '␣x␣', q, '='
print m
```

Factor 50-digit number ($\underset{\sim}{}$166 bits).

```
m.factor()
```

Euler ϕ function. ("totient" function.)

```
euler_phi(100)
```

Integers less than 100 and relatively prime to 100. (Note the `srange` function to generate Sage integers.)

```
relp = [x for x in srange(100) if gcd(x, 100) == 1]
relp
```

```
len(relp)
```

Fact: $\sum_{d|n} \phi(d) = n$

Proof: Group the fractions, $\frac{i}{n}$, $0 \leq i \leq n-1$, by denominators once written in reduced terms.

```
n = 100
sum([euler_phi(d) for d in divisors(n)]) == n
```

3.6 Linear Recurrence Relations

Numbers of certain objects can sometimes be counted by recurrence relations. We would like closed-form expressions for terms of sequences defined this way.

3.6.1 Perrin's Sequence

Perrin Sequence:

$p(0) = 3$; $p(1) = 0$; $p(2) = 2$

$p(n) = p(n-2) + p(n-3)$

Looks like the Fibonacci sequence, but "skips back" two terms, not one.

Compute by hand: $3, 0, 2, 3, 2, 5, 5, 7, 10, 12, 17, \ldots$

This is in Sloane's Online Encyclopedia of Integer Sequences as sequence number A001608.

A brute-force approach with a Python function. Impractical above about $n = 60$.

```
def perrin(n):
    if n == 0:
        return 3
    elif n == 1:
        return 0
    elif n == 2:
        return 2
    else:
        return perrin(n-2) + perrin(n-3)
```

```
perrin(10)
```

```
perrin(20)
```

```
perrin(23).factor()
```

Fact: If q is prime, then q divides $p(q)$.

(First composite number that behaves this way is 521^2.)

Generating function: $f(x) = \sum_{i=0}^{\infty} p_i \, x^i$

Theory gives easy computation for Perrin sequence, denominator comes from recurrence relation, numerator is simple polynomial multiplication.

$$f(x) = \frac{3 - x^2}{1 - x^2 - x^3}.$$

We can expand f as a Taylor series.

```
var('x')
f=(3-x^2)/(1-x^2-x^3)
f
```

```
series = f.taylor(x, 0, 20)
series
```

```
[series.coefficient(x, i) for i in range(20)]
```

3.6.2 Decompose with Partial Fractions

Partial Fractions can simplify a rational generating function. New, three-term recurrence.

$a(0) = 7$; $a(1) = 41$; $a(2) = 204$

$a(n) = 7a(n-1) - 12a(n-2) + 10a(n-3)$

Generating function—the rational function:

```
h = (x^2 - 8*x + 7)/(1 - 7*x + 12*x^2 -10*x^3)
h
```

Check $a(3)$, first new term of the sequence:

```
7*204 -12*41 + 10*7
```

```
h.taylor(x, 0, 3)
```

Create partial fraction decomposition and examine the pieces:

```
h.partial_fraction()
```

```
denom1 = 1/(2*x^2 - 2*x + 1)
denom1.taylor(x, 0, 30)
```

```
denom2 = 1/(5*x - 1)
denom2.taylor(x, 0, 8)
```

3.6.3 Using SymPy

SymPy is a pure Python package *included* in Sage, but not *integrated* with Sage.

docs.sympy.org/dev/modules/solvers/solvers.html#recurrence-equtions (sic)

Import pieces of the SymPy library.

```
from sympy import Function, rsolve
from sympy.abc import n
y = Function('y')
```

Define the recurrence as an expression in $y(\cdot)$ that equals zero.

```
k = y(n+3)- 7*y(n+2) + 12*y(n+1) - 10*y(n)
```

And solve:

```
rsolve(k, y(n))
```

```
rsolve(k, y(n), {y(0):7, y(1):41, y(2):204})
```

4 Sage Environs

Sage ships with a Jupyter notebook server, which is a web application that provides a convenient interface to Sage commands, components and features.

Much of this section will only behave properly within a Jupyter Notebook server using a Sage kernel. However some portions are transferable to command-line use or via the Sage Cell Server.

4.1 (Some) Indiscreet Mathematics

A symbolic derivative (from Maxima).

```
f(x) = x^3*e^-x
df = f.derivative()
df
```

Derivative of a function is again a function, and can be evaluated.

```
slope = df(4)
slope
```

Arbitrary precision numerical values on request (from MPmath).

```
N(slope, digits=20)
```

Can display plots in the notebook (via matplotlib).

```
plot(df, 0, 10, color='red', thickness=5)
```

Study the multivariate integral $\int_{-4}^{4}\int_{0}^{x^2} y^2 - 10x^2\,dy\,dx$.

```
var('x␣y␣z')
integral(integral(y^2-10*x^2, (y, 0, x^2)), (x, -4, 4))
```

3-D plots are especially intriguing.

```
surface = plot3d(y^2-10*x^2, (x, -4, 4), (y, 0, 16))
show(surface)
```

Implicit plots allow for more general surfaces.

```
region = implicit_plot3d(y-x^2, (x, -4, 4), (y, 0, 16), (z,
    0, 98), color='red', opacity=0.20)
show(surface+region)
```

4.2 Interactive Explorations

Interactive demonstrations are easy to create with the "interact" decorator and modified function arguments.

```
@interact
def plotter(maxdegree=range(2,40)):
    import sage.plot.colors
    colors = sage.plot.colors.rainbow(maxdegree+1)
    var('x')
    wholeplot = plot(x^1, (x, 0, 1), color=colors[1])
    for i in range(2, maxdegree+1):
        newplot = plot(x^i, (x, 0, 1), color=colors[i])
        wholeplot = wholeplot + newplot
    show(wholeplot)
```

```
@interact
def taylor(order=slider(1, 12, 1, default=Integer(2),
    label="Degree")):
  var('x')
  x0  = 0
  f   = sin(x)*e^(-x)
  p   = plot(f, -1, 5, thickness=2)
  dot = point((x0,f(x=x0)), pointsize=80, rgbcolor=(1,0,0))
  ft  = f.taylor(x, x0 ,order)
  pt  = plot(ft, -1, 5, color='green', thickness=2)
  show(dot + p + pt, ymin = -0.5, ymax = 1)
```

4.3 LaTeX Integration

...is superb.

```
latex(integrate(sec(x), x))
```

```
A = random_matrix(QQ, 6, num_bound=9, den_bound=9)
latex(A)
```

Now switch display mode to LaTeX.

```
%display latex
```

```
A = random_matrix(QQ, 6, num_bound=9, den_bound=9)
A
```

And back to plain text.

```
%display plain
```

```
P = graphs.PetersenGraph()
P.set_latex_options(vertex_shape='diamond',
    vertex_color='red', vertex_label_color='gold',
    edge_color='blue')
```

Cut and paste a LaTeX representation into your research article.

```
latex(P)
```

New **Markdown** cells also allow HTML and LaTeX.We can add text to our notebooks using TeX syntax and dollar signs. Previous multivariate integral: `\int_0^4\int_0^{x^2}y^2-10x^2\,dy\,dx`

```
```

Can embed images this way also.

4.4 Help, Doctests, Source Code

A huge number of examples are provided for (a) learning to use Sage commands, and (b) to test Sage commands. We call these "doctests."

```
M = matrix(QQ, [[1, -2, 2], [-4, 5, 6], [1, 2, 4]])
M
```

Illustrate tab-completion (rational form), help (doctests, `zig-zag form`), source code.

```
M.
```

4.5 Cython

A Sage-inspired project to convert Python to C, then compile.

Factorial, Python-style.

```
def py_fact(n):
    fact = 1
    for i in range(n):
        fact = fact*(i+1)
    return fact
```

```
py_fact(12)
```

```
timeit('py_fact(12)')
```

Cython-style. (cdef, long in header)

```
%%cython
def cy_fact(n):
    cdef:
        long fact, i
    fact = 1
    for i in range(n):
        fact = fact*(i+1)
    return fact
```

```
cy_fact(12)
```

```
timeit('cy_fact(12)')
```

4.6 Sage Single Cell Server

See HTML version of this presentation.

5 Symbolic Manipulation and Plotting Discretely

You need to declare symbolic variables (except `x` comes pre-defined). That done, summations simplify as expected.

```
var('i,␣n')
expr = sum(i^2, i, 0, n)
expr
```

We'll recognize this result if we factor.

```
expr.factor()
```

Arbitrarily complicated polynomials as summands can be simplified.

```
var('i,␣n')
expr = sum(2*i^5 - 6*i^4 +7*i^2 - 8, i, 0, n)
expr
```

We can convert this *symbolic expression* to a *callable function*.

```
var('t')
g(t) = expr.subs(n=t)
g
```

And call it—thus making n concrete.

```
g(10)
```

Straightforward to plot a discrete function, we will plot using an expression.

```
var('i,␣n')
expr = sum(2*i^3 - 12*i^2, i, 0, n)
points = [(k, expr.subs(n=k)) for k in range(10)]
list_plot(points)
```

Again, but with options.

```
list_plot(points, size=200, color='purple')
```

```
list_plot(points, color='red', plotjoined=True)
```

6 Hacking on Sage

DEMONSTRATION: Modifying Sage source code.

- Location of `catalan_number()` (bottom of source with ?? query)
- Edit: `SAGE_ROOT/src/sage/combinat/combinat.py`
- Change: add `print "Hello, AIMS!"` to `def catalan_number():`
- Rebuild: `./sage -b` at `SAGE_ROOT`
- Run: `./sage` at `SAGE_ROOT`
- Test: `./sage -t SAGE_ROOT/src/sage/combinat/combinat.py`

This worksheet available at: `buzzard.ups.edu/talks.html`