Intro to Sage - SD 60 Chennai

Introduction to *Sage*:

Mathematics Software for All

Sage Days 60, IMSc, Chennai

Speaker: Karl-Dieter Crisman, Gordon College

Thank you to the organizers for the invitation to come and introduce you all to Sage.

Just the Basics

Let's get started. Hopefully by now you and I don't need help with the following computations:

```
2+2
```

```
4
```

```
integrate(x^3,x)
```

```
1/4*x^4
```

```
3*vector([1,2,3])
```

```
(3, 6, 9)
```

On the other hand, if I needed one of the following as an intermediate step, I might appreciate some assistance:

```
factorial(100)
```

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

```
matrix([[1,2,3],[4,5,7],[6,8,9]])^-1
```

```
[-11/7   6/7  -1/7]
[  6/7  -9/7   5/7]
[  2/7   4/7  -3/7]
```

What if someone sends me a possible counterexample to FLT?

```
27784^3 + 35385^3 - 40362^3
```

```
1
```

Good thing we checked. Now let's do an integral.

```
integral(3*x^2/sqrt(25*x^2-3), x)
```

```
3/50*sqrt(25*x^2 - 3)*x + 9/250*log(50*x + 10*sqrt(25*x^2 - 3))
```

Hmm, the antiderivative looks kind of ugly. Let me try to simplify it and typeset it nicely.

```
show(expand(integral(3*x^2/sqrt(25*x^2-3),x)))
```

$$\frac{3}{50}\sqrt{25\,x^2-3}x+\frac{9}{250}\log\left(50\,x+10\sqrt{25\,x^2-3}\right)$$

That's better. Well, maybe I should just type it in my notes, so I don't forget.

$$\frac{3}{50}\sqrt{25x^2-3x}+\frac{9}{250}\log\left(50x+10\sqrt{25x^2-3}\right)$$

Of course, maybe I needed a *definite* integral.

```
show(integral(3*x^2/sqrt(25*x^2-3),x,1,2))
```

$$\frac{3}{25}\sqrt{97}-\frac{3}{50}\sqrt{22}+\frac{9}{250}\log\left(10\sqrt{97}+100\right)-\frac{9}{250}\log\left(10\sqrt{22}+50\right)$$

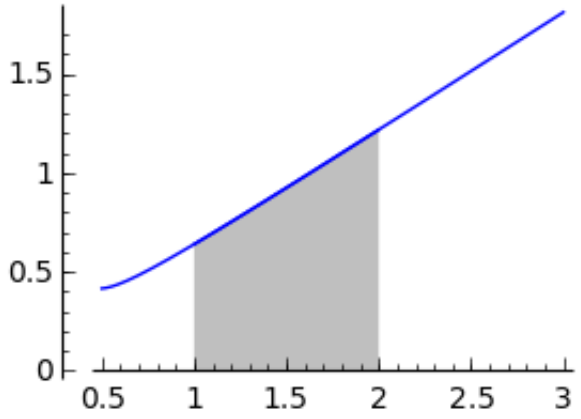
And maybe I needed it numerically approximated, with a built-in error bound computed.

```
numerical_integral(3*x^2/sqrt(25*x^2-3),1,2)
```

```
(0.9262503194124578, 1.0283444311781162e-14)
```

And maybe I need to visualize that as well...

```
show(plot(3*x^2/sqrt(25*x^2-3),
(x,1,2),fill=True)+plot(3*x^2/sqrt(25*x^2-3),(x,.5,3)),figsize=3)
html("This shows $\int_1^2 \\frac{3x^2}{\sqrt{25x^2-3}}\; dx$")
```



This shows $\int_1^2 \frac{3x^2}{\sqrt{25x^2-3}}\,dx$

Even if you haven't done these sorts of things yourself, you know you can do so with computer help, and possibly have something in a lab or on your laptop which can do them - or you use some web app or other. But creating something like this takes just a bit more time.

```
%hide
def Newton_Iterates(a_function, guess, iterations = 5):
    '''
```

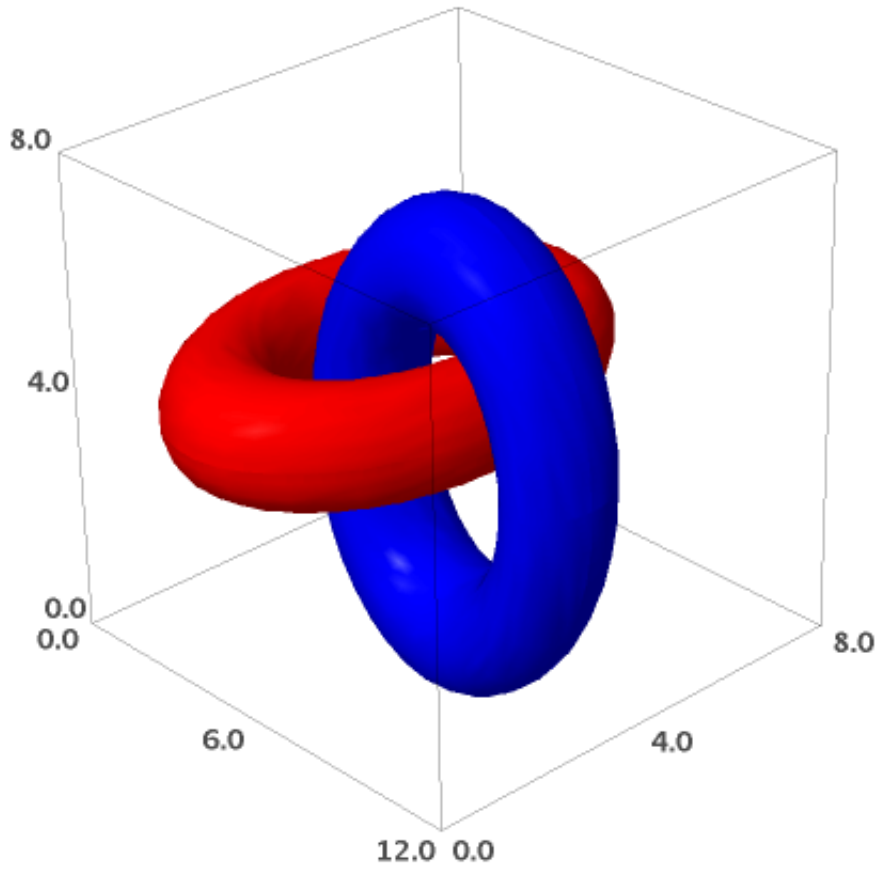
```
    Returns a list of the iterations of the Newton-Raphson method
approximations to a zero of a_function.

    INPUT:
        a_function: a function of one variable
        start: the starting value of the iteration
        iterations: (optional) the number of iterations to draw
    '''
    f(x)=a_function(x)
    deriv=f.derivative()
    approx(x)=x-f(x)/deriv(x)
    input = guess
    iter_list = [input]
    for i in range(iterations):
        input = approx(input).n()
        iter_list.append(input)
    return iter_list

def Newton_Graph(a_function, guess, iterations = 5):
    '''
    Returns a graphics object of a plot of a_function and Newton-
Raphson method tangent lines.

    INPUT:
        a_function: a function of one variable
        start: the starting value of the iteration
        iterations: (optional) the number of iterations to draw
        xmin: (optional) the lower end of the plotted interval
        xmax: (optional) the upper end of the plotted interval

    EXAMPLES:
        sage: f = lambda x: x^3-3*x^2+2*x
        sage: show(Newton_Graph(f,.5))

    Note: This is very slow with symbolic functions.
    '''
    iter_list = [[guess,0]]
    f(x)=a_function(x)
    deriv=f.derivative()
    approx(x)=x-f(x)/deriv(x)
    input = guess
    for i in range(iterations):
        iter_list.append([input,f(input)])
        input=approx(input).n()
        iter_list.append([input,0])
    approx_tangents = line(iter_list,rgbcolor=(1,0,0))
    xmin=min([point[0] for point in iter_list])-1
    xmax=max([point[0] for point in iter_list])+1
```

```
    ymin=min([point[1] for point in iter_list])-.5
    ymax=max([point[1] for point in iter_list])+.5
    basic_plot = plot(a_function, xmin, xmax, color='blue')
    P=basic_plot + approx_tangents
    P.show(ymin=ymin,ymax=ymax)

def cubic_approx(guess, intercept=0, iterations = 5):
    f(x)=x^3-3*x^2+2*x+intercept
    return Newton_Graph(f,guess,iterations=iterations)

def cubic_iterate(guess, intercept=0, iterations = 5):
    f(x)=x^3-3*x^2+2*x+intercept
    return Newton_Iterates(f,guess,iterations=iterations)

@interact
def _(guess=RR(.5),intercept=slider(-
2,2,.5,0),iterations=slider(1,20,1,5)):
    root =
cubic_iterate(guess,intercept=intercept,iterations=iterations)[-1]
    print 'start guess=', guess
    print 'intercept=', intercept
    print 'iterations =', iterations
    print 'root approximation =', root
    cubic_approx(guess,intercept=intercept,iterations=iterations)
```

Or maybe it was something that you really need to see from more than one viewpoint.

```
u, v = var('u,v')
f1 = (4+(3+cos(v))*sin(u), 4+(3+cos(v))*cos(u), 4+sin(v))
f2 = (8+(3+cos(v))*cos(u), 3+sin(v), 4+(3+cos(v))*sin(u))
p1 = parametric_plot3d(f1, (u,0,2*pi), (v,0,2*pi), texture="red")
p2 = parametric_plot3d(f2, (u,0,2*pi), (v,0,2*pi), texture="blue")
show(p1 + p2)
```

Sleeping... Make Interactive



So What?

It's so great to have this computational and visualization power. Unfortunately, with some such software, you may have run into one of the following three barriers to using it more widely.

- You used wonderful (free!) applets and demonstrations (e.g. for fractals, modular arithmetic), which handle **only one narrow piece of mathematics**.
- You were asked to use a powerful and impressive comprehensive software package, but it is **very expensive**, especially depending on how often you use it.
- You might have needed to use a computer lab whose **hours are inconvenient and might be distant** (certainly not useful at home).

Sage, the software I am currently using in this talk, is a **free, powerful option** which avoids all these issues. You can download it freely, *OR* you can run it off a server for interface **in your web browser**- just as I am doing right now! Indeed, there are four interfaces:

- Command line interface
- This notebook interface
- Cloud.sagemath.com, a next-gen web app
- Sage cell server

- (There are also apps for Android and iPhone/iPad which use the cell server technology.)

Let me demonstrate each of these for you briefly.

(Demo Interlude)

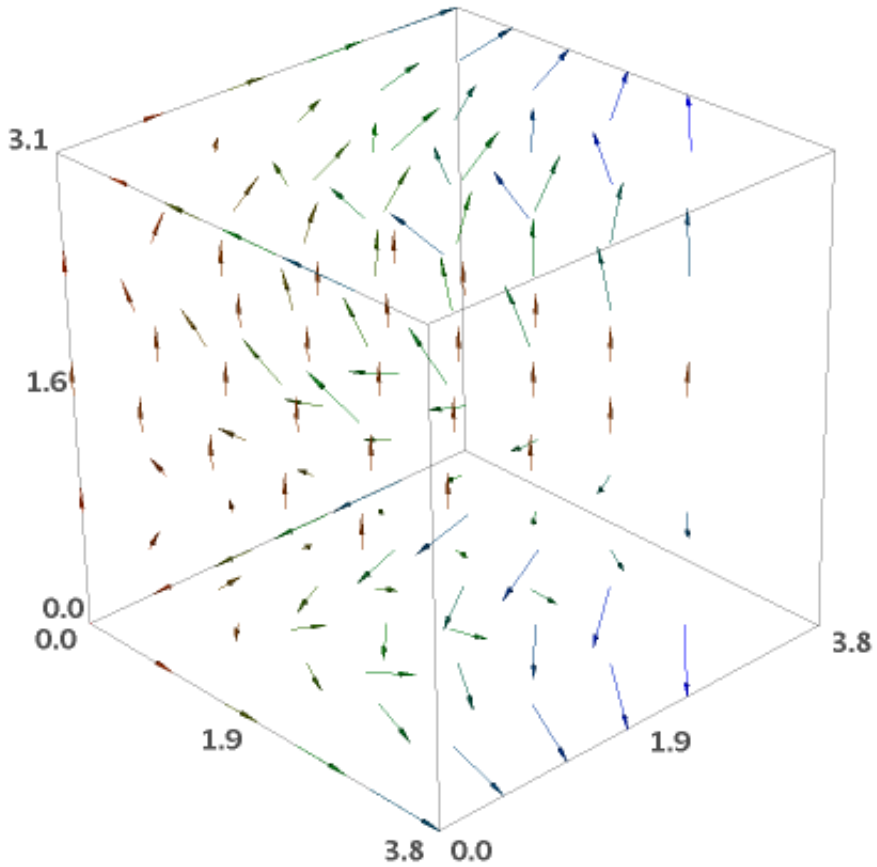
However, we'll stick with the notebook here.

More Basics

Sage handles other topics in undergraduate mathematics with aplomb, such as vector calculus, number theory, applied linear algebra, differential equations, statistics...

```
x,y,z=var('x y z')
plot_vector_field3d((x*cos(z),-y*cos(z),sin(z)), (x,0,pi), (y,0,pi),
(z,0,pi),colors=['red','green','blue'])
```

Sleeping... Make Interactive



This one is best viewed using 3D glasses, of course. The following one shows many theorems about numbers, color-encoded!

```
@interact
def power_table_plot(p=(7,prime_range(50))):
```

```
    P=matrix_plot(matrix(p-1,[mod(a,p)^b for a in range(1,p) for b in
srange(p)]),cmap='jet')
    show(P)
```

If you like numerical methods, these come built-in as well. Note the interactivity Sage can provide.

```
%hide
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'san_thome.png'), 2)
B_image = pylab.mean(pylab.imread(DATA + 'imsc_night.png'), 2)
@interact
def svd_image(i=(5,(1..50))):
    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))
    u1,s1,v1 = pylab.linalg.svd(B_image)
    B = sum(s1[j]*pylab.outer(u1[0:,j],v1[j,0:]) for j in range(i))
    g = graphics_array([[matrix_plot(A),matrix_plot(B)],
[matrix_plot(A_image),matrix_plot(B_image)]])
    show(g,axes=False, figsize=6)
    html('<h2>Compressed using %s eigenvalues</h2>'%i)
```

```
%hide
y = var('y')
@interact
def _(g=input_box(default = 1-y),a=2,b=2,auto_update=False):
     yfun = function('yfun',x)
     f = desolve(diff(yfun,x) - g(y=yfun), yfun, ics=[a,b])
     Q=plot(f,0,3)
     q = Q[0].get_minmax_data()
     P=plot_slope_field(g,(x,q['xmin'],q['xmax']),
(y,q['ymin'],q['ymax']))
     html('$f(x)=%s$'%(latex(f),))
     show(P+Q)
```

In the previous example, notice that it's possible to join several different types of computations together. In this example, we calculate a symbolic solution to a differential equation, but plot the slope field and solution numerically. Sage tries to be unified as much as possible.

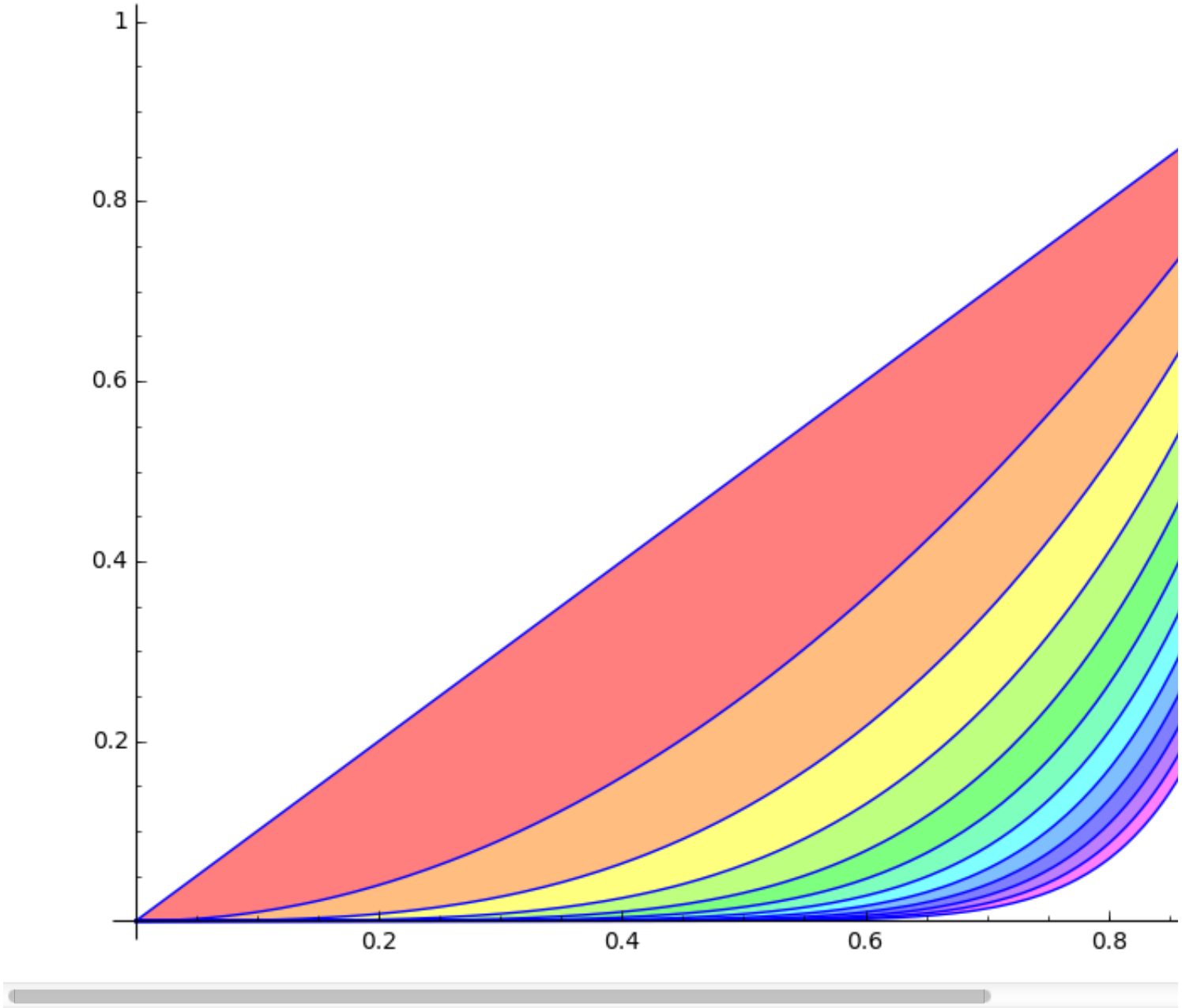
Statistics are possible via some very basic native Sage functionality, or via several very powerful components of Sage that are primarily statistical/numerical. By far the most powerful of these is R.

```
%r
library("MASS")
data(Cars93)
mean(Cars93$MPG.city)
xtabs( ~ Origin + MPG.city, data=Cars93)
```

```
[1] 22.36559
         MPG.city
Origin    15 16 17 18 19 20 21 22 23 24 25 26 28 29 30 31 32 33 39
42 46
  USA      2  3  4  5  8  3  3  4  7  2  2  0  1  2  0  2  0  0  0
0  0
  non-USA  0  0  4  7  2  5  3  3  1  3  4  2  1  4  1  0  1  1  1
1  1
```

Of course, one reason *I* like Sage is not all these 'useful' things, but producing eye candy:

```
plot([x^i for i in [1..12]],(x,0,1),fill = dict((i,[i+1]) for i in
[0..11]))
```



Can it help with Research?

But can you do research-level mathematics in Sage? In fact, it started as purely a research tool nearly a decade ago. Just because it quickly became the comprehensive software you see today doesn't mean it isn't full of cutting-edge mathematics. First, let's see a small example that doesn't use too much high-powered stuff. It's a tiny piece of what a student of mine used to disprove a conjecture of one of my colleagues - and to come up with another conjecture, which she proved. (It's since appeared in *Mathematical Social Sciences*.)

```
%hide
def get_correct_perms_generator(n):
    r"""
    Creates a generator to make every non-parametric data set
    with 3 candidates and n rows, in the normal form with each
    column ordered down and the top row ordered left to right.

    There are (3n!)/(n!^3*6) of these - found by using Sloane
    online encyclopedia because I was lazy, but this is obvious
    in hindsight.
    """
    from copy import copy
    firstcolumniter=Combinations(range(1,3*n),n-1)
    for combo1 in firstcolumniter:
        output1=[3*n]
        data1=range(1,3*n)
        combo1.sort()
        combo1.reverse()
        for item1 in combo1:
            output1.append(item1)
            data1.remove(item1)
        data1.sort()
        biggestleft=data1.pop()
        output1.append(biggestleft)
        secondcolumniter=Combinations(data1,n-1)
        for combo2 in secondcolumniter:
            output2=copy(output1)
            data2=copy(data1)
            combo2.sort()
            combo2.reverse()
            for item2 in combo2:
                output2.append(item2)
                data2.remove(item2)
            data2.sort()
            data2.reverse()
            for item3 in data2:
                output2.append(item3)
            yield output2

def get_profile(mylist,n):
    r"""
    Takes a non-parametric data set with 3 candidates
```

```
    and n rows and yields the voting profile associated
    to it.
    """
    import sage.combinat.permutation as permutation
    profile=[0,0,0,0,0,0]
    rawpref=[]
    standardrawpref=[]
    for i in mylist[0:n]:
        for j in mylist[n:2*n]:
            for k in mylist[2*n:]:
                rawpref=[i,j,k]
                standardrawpref=list(permutation.to_standard(rawpref))
                if standardrawpref==[3,2,1]:
                    profile[0]+=1
                elif standardrawpref==[3,1,2]:
                    profile[1]+=1
                elif standardrawpref==[2,1,3]:
                    profile[2]+=1
                elif standardrawpref==[1,2,3]:
                    profile[3]+=1
                elif standardrawpref==[1,3,2]:
                    profile[4]+=1
                elif standardrawpref==[2,3,1]:
                    profile[5]+=1
    return(profile)

def get_decomposition(myprofile):
    r"""
    Takes a voting profile with three candidates
    and yields the standard (Saari) decomposition
    into Basic (A and B), Reversal (A and B),
    Condorcet and Kernel components.
    """
    ProfileMatrix=(1/6)*matrix([[2,1,-1,-2,-1,1],[1,-1,-2,-1,1,2],
[0,1,-1,0,1,-1],[-1,1,0,-1,1,0],[1,-1,1,-1,1,-1],[1,1,1,1,1,1]])
    return (ProfileMatrix*matrix(6,myprofile)).transpose()
```

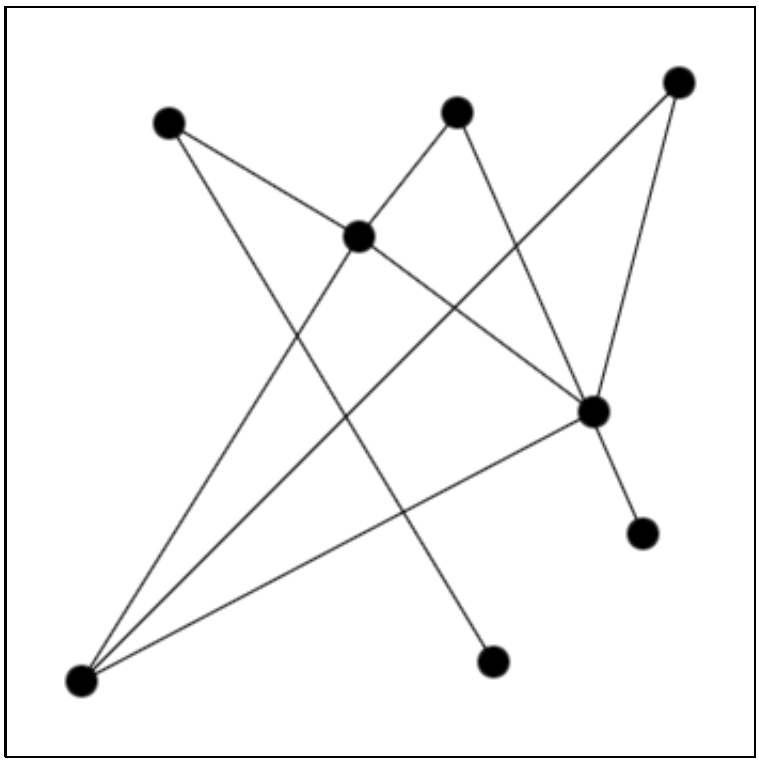
```
G = get_correct_perms_generator(2)
p = G.next()
p;get_profile(p,2);get_decomposition(get_profile(p,2))
```

```
    [6, 1, 5, 2, 4, 3]
    [2, 2, 0, 2, 2, 0]
    [  0   0 2/3   0   0 4/3]
```

Graphs, Groups, and More

Okay, let's look at some topics I have heard the attendees at this talk might be interested in. To start off, let's make a graph. Any old graph.

```
G = Graph({0:[1,2],1:[0,2,3],2:[0,1,3],3:[1,2,5,6],4:[5],5:[3,4],6:
[3,7],7:[6]}); G.set_pos({0:[315,315],1:[35,35],2:
[275,160.99999999999997],3:[165,243],4:[228,44],5:[76,296],6:
[211,301],7:[298,104]});  graph_editor(G);
```




```
G.chromatic_polynomial()
```

```
x^8 - 9*x^7 + 34*x^6 - 70*x^5 + 85*x^4 - 61*x^3 + 24*x^2 - 4*x
```

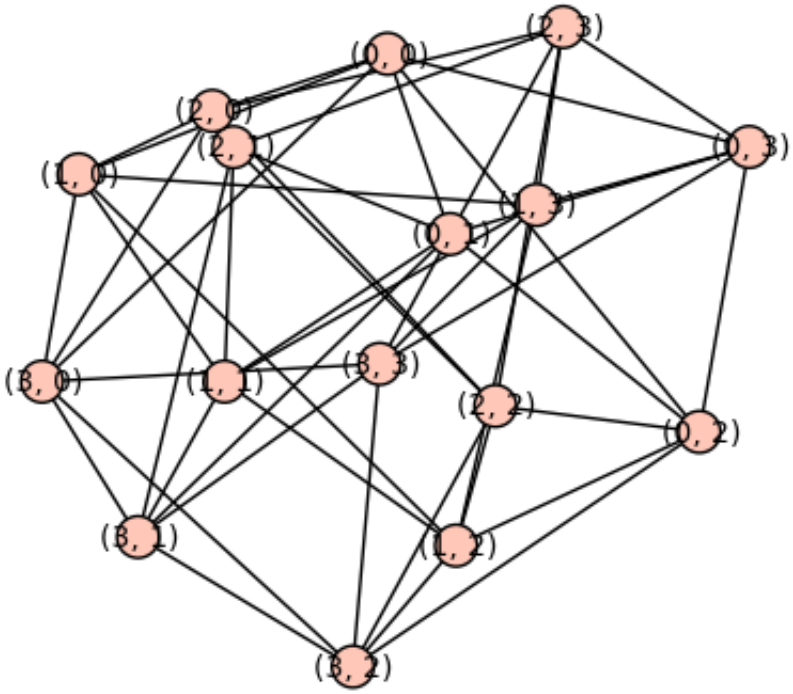
Note that this means any computation you want to do that Sage can do with graphs, you can in principle do by just "drawing" the graph.

Sage consists of many separate components, each of which is an open-source standard in its field. Let's see a nice example of all these things working together.

```
H = graphs.TetrahedralGraph() # Takes NetworkX graph
```

```
H = H.cartesian_product(H) # Uses Sage functionality
```

```
show(H) # uses matplotlib
```



```
Sym = H.automorphism_group() # Native Sage Code using C backend for graphs
```

```
len( Sym.list() )  # Needs GAP to calculate
```

```
1152
```

```
Sym._gap_().ComputedPCentralSeriess()  # Directly within GAP
```

```
[  ]
```

```
poly = H.chromatic_polynomial() # Uses compiled C code from Cython in Sage
```

```
deriv = derivative(poly,x) # Uses Ginac/Pynac for symbolics
```

```
deriv.subs(x=0)
```

```
-188580960
```

And yes, this graph is connected.

```
integral(poly,x) # Uses Maxima for integration, symbolic summation, etc.
```

```
1/17*x^17 - 3*x^16 + 1096/15*x^15 - 7902/7*x^14 + 161032/13*x^13 - 102358*x^12 + 7242454/11*x^11 - 16789056/5*x^10 + 123182360/9*x^9 - 89140779/2*x^8 + 114946568*x^7 - 230543668*x^6 + 1742815741/5*x^5 - 376891986*x^4 + 264255876*x^3 - 94290480*x^2
```

The book "Quo Vadis, Graph Theory?: A Source Book for Challenges and Directions" asks for

interpretations of such things.

Non-mathematical fun

There are many other great aspects of Sage I haven't mentioned yet. Let's see how each of these works.

- Tab-Completion for instant, contextual help
- It's *Open Source* - so fast bug fixes or enhancements happen as a matter of course - and *you* can contribute!
- It's a combination of research-level quality new code and well-regarded programs such as Maxima, GAP, PARI, R, ... so a **gigantic** range of functionality comes "out of the box"
- It's integrated - you can use the Python language, LaTeX, proprietary programs, command-line interface...

But the best thing to do is to try it out or download it!

- http://www.sagemath.org - the Sage website
- http://cloud.sagemath.com
- While on campus at SD 60, http://citron:8080/

More Math

Here is a gallery of other research-level things. But to really explore, one needs to peruse the gargantuan reference manual.

Analytic Number Theory

I like this example, as it shows that Riemann's exact formula for $\pi(x)$, the prime counting function, is completely different in nature than other approximations.

```
import mpmath
var('y')
L = lcalc.zeros_in_interval(10,50,0.1)
@interact
def _(n=(100,(60,10^3))):
   P = plot(prime_pi,n-50,n,color='black',legend_label='$\pi(x)$')
   P += plot(Li,n-50,n,color='green',legend_label='$Li(x)$')
   G = lambda x: sum([mpmath.li(x^(1/j))*moebius(j)/j for j in
[1..3]])
   P += plot(G,n-
50,n,color='red',legend_label='$\sum_{j=1}^{%s}\\frac{\mu(j)}
{j}Li(x^{1/j})$'%3)
   F = lambda x: sum([(mpmath.li(x^(1/j))-
log(2)+numerical_integral(1/(y*(y^2-1)*log(y)),x^(1/j),oo)
[0])*moebius(j)/j for j in [1..3]])-sum([(mpmath.ei(log(x)*
```

```
((0.5+l[0]*i)/j))+mpmath.ei(log(x)*((0.5-l[0]*i)/j))).real for l in L
for j in [1..3]])
   P += plot(F,n-50,n,color='blue',legend_label='Really good
estimate',plot_points=50)
   show(P)
```

Here's a result from the last couple years connected to work the founder of Sage, William Stein, is doing, as well as to Ramanujan (some of you will know the connection).

Theorem:

Let $r_{12}(n)$ denote the number of ways to write n as a sum of *twelve* squares (where order and sign both matter, so $(1,2)$ and $(2,1)$ and $(-2,1)$ are all different). As we let p go through the set of all prime numbers, the distribution of the fraction

$$\frac{r_{12}(p) - 8(p^5+1)}{32p^{5/2}}$$

is asymptotic to

$$\frac{2}{\pi}\sqrt{1-t^2}\,.$$

```
def dist(v, b, left=float(0), right=float(pi)):
    """
    We divide the interval between left (default: 0) and
    right (default: pi) up into b bins.

    For each number in v (which must left and right),
    we find which bin it lies in and add this to a counter.
    This function then returns the bins and the number of
    elements of v that lie in each one.

    ALGORITHM: To find the index of the bin that a given
    number x lies in, we multiply x by b/length and take the
    floor.
    """
    length = right - left
    normalize = float(b/length)
    vals = {}
    d = dict([(i,0) for i in range(b)])
    for x in v:
        n = int(normalize*(float(x)-left))
        d[n] += 1
    return d, len(v)
```

```
def graph(d, b, num=5000, left=float(0), right=float(pi)):
    s = Graphics()
    left = float(left); right = float(right)
    length = right - left
    w = length/b
    k = 0
    for i, n in d.iteritems():
        k += n
        # ith bin has n objects in it.
        s += polygon([(w*i+left,0), (w*(i+1)+left,0), \
                      (w*(i+1)+left, n/(num*w)), (w*i+left,
n/(num*w))],\
                      rgbcolor=(0,0,0.5))
    return s

def sqrt2():
    PI = float(pi)
    return plot(lambda x: (2/PI)*math.sqrt(1-x^2), -1,1,
plot_points=200,
                rgbcolor=(0.3,0.1,0.1), thickness=2)

delta = delta_qexp(10^5)

@interact
def delta_dist(b=(20,(10..150)), number = (500,1000,..,delta.prec())):
    D = delta[:number]
    w = [float(D[p])/(2*float(p)^(5.5)) for p in prime_range(number +
1)]
    d, total_number_of_points = dist(w,b,float(-1),float(1))
    show(graph(d, b, total_number_of_points,-1,1) + sqrt2(),
frame=True, gridlines=True)
```

Cython

The Sage-combinat crew has a huge number of good tutorials that are not (yet) in the official Sage documentation. While we're on the subject of prime numbers, here is one which shows how Cython can be used to speed up calculations of interest.

```
sage: def first_primes_python(m):
...       primes_list = []
...       n = 2
...       while len(primes_list) < m:
...           n_is_prime = True
...           for p in primes_list:
...               if n % p == 0:
...                   n_is_prime = False
```

```
...                     break
...             if n_is_prime == True:
...                 primes_list.append(n)
...             n = n + 1
...         return primes_list
```

```
%cython
def first_primes_cython_v1(m):
      primes_list = []
      n = 2
      while len(primes_list) < m:
          n_is_prime = True
          for p in primes_list:
              if n % p == 0:
                  n_is_prime = False
                  break
          if n_is_prime == True:
              primes_list.append(n)
          n = n + 1
      return primes_list
```

__Users_ka...2_code_sage73_spyx.c __Users_ka...ode_sage73_spyx.htm

```
time p = first_primes_python(5000)
```

```
Time: CPU 1.82 s, Wall: 1.82 s
```

```
time p = first_primes_cython_v1(5000)
```

```
Time: CPU 0.45 s, Wall: 0.45 s
```

Formal Power Series in Combinatorics

It turns out that one can do quite complicated things combining combinatorics and power series within Sage. This example is too long for me to include directly, so let's go to the link! (A world-readable link to a non-live version is here.)

As it says, "Therefore, instead of solving the equation, we look for the equation describing the object which is best suited to the problem we want to solve."

```

```

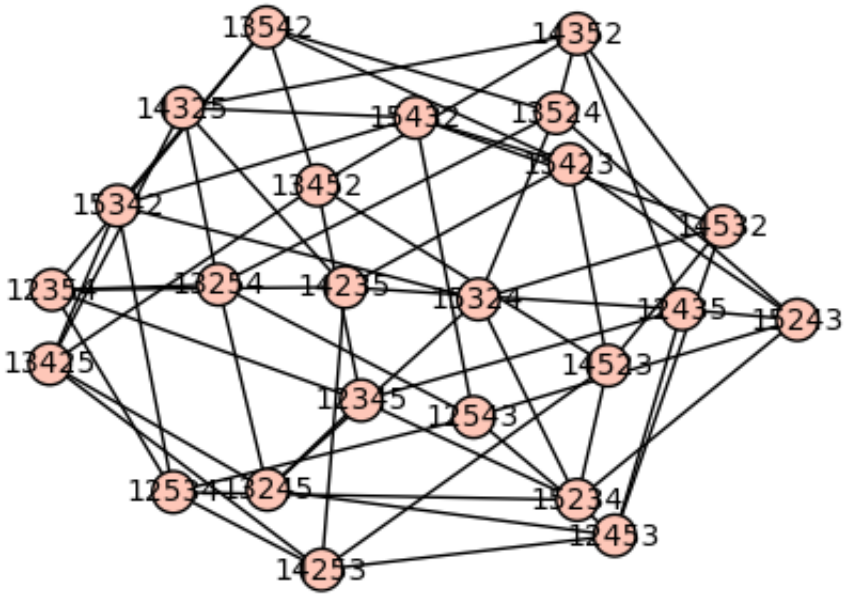
Random Self-Promotion

I don't have a heading for this, but it just shows that you can do a lot in Sage. This actually has applications in social choice theory, usually considered as a subbranch of economics.

```
d5 = {12345:[12354,12435,13245,15234,13452],12354:
[12534,13254,14235,13542],12435:[15243,12453,14352,14235],13245:
```

```
[15324,12453,13254,13425],15234:[14523,15243,12534,15324],13452:
[13425,13542,14523,14352],15432:[15423,15342,14532,14325,12543],12543:
[13254,15243,12534,12453],14325:[14352,14235,13254,13425],14532:
[15324,14352,12453,14523],15342:[13425,15324,12534,13542],15423:
[15243,14235,14523,13542],13524:[13542,15243,13254,14352,15324],14253:
[12534,12453,14523,14235,13425]}
G5 = Graph(d5)
```

```
plot(G5,figsize=4)
```




```
Gp5 = G5.automorphism_group()
print Gp5.order()
```

```
480
```

```
Gp5.character_table()
```

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

```
1                -1               -1
1                 1               -1
-1                  1]
[               1                 1
1                 1               -1
-1                -1                -1
-1                -1                 1
1                 1                 1
1                 1               -1
-1                  1]
[               2                 0
-2                  2                 0
0                 0                 0
0                 0                 2
-2                  0                 0
-2                  2                 0
0               -2]
[               4                 0
0               -1                -2
0                 1                 0
-1                  1                 1
1                 2               -1
-1                  0                 1
-4                  4]
[               4                 0
0               -1                -2
0                 1                 0
1               -1                 1
1               -2                 1
-1                  0                -1
4                 4]
[               4                 0
0               -1                 2
0               -1                 0
-1                  1                 1
1               -2                 1
-1                  0                 1
-4                  4]
[               4                 0
0               -1                 2
0               -1                 0
1               -1                 1
1                 2               -1
-1                  0                -1
4                 4]
[               5               -1
1                 0                -1
-1                -1                  1
1                 0                -1
-1                  1                  1
0                 1                 0
```

```
-5                                5]
[                               5                                -1
1                                0                                 1
1                                1                                -1
-1                               0                                -1
-1                               1                                 1
0                                1                                 0
5                                5]
[                               5                                 1
1                                0                                -1
1                               -1                                 1
-1                               0                                -1
-1                              -1                                -1
0                                1                                 0
5                                5]
[                               5                                 1
1                                0                                 1
-1                               1                                -1
1                                0                                -1
-1                              -1                                -1
0                                1                                 0
-5                               5]
[                               6                                 0
-2                               1                                   0
-2                               0                                   0
0                                1                                 0
0                                0                                 0
1                               -2                                 1
6                                6]
[                               6                                 0
-2                               1                                   0
2                                0                                 0
0                               -1                                 0
0                                0                                 0
1                               -2                                -1
-6                               6]
[                               6                                 0
2                                1                                 0
0                                0                                 0
0  2*zeta5^3 + 2*zeta5^2 + 1                                      0
0                                0                                 0
-1                              -2 -2*zeta5^3 - 2*zeta5^2 - 1
0                               -6]
[                               6                                 0
2                                1                                 0
0                                0                                 0
0 -2*zeta5^3 - 2*zeta5^2 - 1                                      0
0                                0                                 0
-1                              -2  2*zeta5^3 + 2*zeta5^2 + 1
0                               -6]
[                               8                                 0
```

```
0                -2                 0
0                 0                 0
0                 0                 2
-2                 0                 0
2                 0                 0
0                -8]
[               10                      0
-2                 0                 0
0                 0                 0
0                 0                -2
2                 0                 0
0                 2                 0
0               -10]
```

```
Chars5 = Gp5.irreducible_characters()
CCRs5 = Gp5.conjugacy_classes_representatives()
P5 = [sum([len(z) for z in x.cycle_tuples(True) if len(z)==1]) for x
in CCRs5]
for ch in Chars5:
    mysum = 0
    for i in range(19):
        mysum += P5[i]*ch.values()[i] * 480 /
Gp5.centralizer(CCRs5[i]).order()
    print mysum/480
```

```
1
0
0
1
0
0
0
0
0
0
1
1
0
0
1
1
0
0
```

To me, this is basically magic. I hope you will enjoy similar magical experiences with Sage!

```
```