480 -- March 31, 2008

Sage Demo For Math 480
Lecture 1: March 31, 2008

Basic Arithmetic

```
2 + 3
```

```
5
```

```
expand((x+1)^5)
```

```
x^5 + 5*x^4 + 10*x^3 + 10*x^2 + 5*x + 1
```

```

```

Some Basic Programming

```
def f(n):
    """
    Compute the product of the even integers up to n.
    INPUT:
        n - a positive integer
    OUTPUT:
        a positive integer
    EXAMPLES:
        sage: f(4)
        8
    """
    # this algorithm isn't optimal; it's to illustrate using if
and for.
    j = 1
    for i in [1..n]:      # [..]'s is a Sage extension to python
        if i % 2 == 0:
            j *= i
    return j
```

```
f(4)
```

```
8
```

```
time f(80)
```

```
897108341211212142020325469195355364998152634499072000000000
CPU time: 0.00 s,  Wall time: 0.00 s
```

```
f?
```

```
f??
```

```

```

```

```

Interfaces

```
mathematica.eval('2 + 3')
```

```
5
```

```
isinstance(mathematica.eval('2 + 3'), str)
```

```
True
```

```
type(mathematica('2008'))
```

```
<class 'sage.interfaces.mathematica.MathematicaElement'>
```

```
a = mathematica('2008'); a
```

```
2008
```

```
a.FactorInteger()
```

```
{{2, 3}, {251, 1}}
```

```
mathematica('Integrate[Sin[x^2],x]')
```

```
Sqrt[Pi/2]*FresnelS[Sqrt[2/Pi]*x]
```

```
mathematica(sin(x^2)).Integrate(x)
```

```
Sqrt[Pi/2]*FresnelS[Sqrt[2/Pi]*x]
```

```
m = octave('rand(4)'); m
```

```
0.538285 0.687853 0.411032 0.279506
 0.884794 0.950939 0.584321 0.967065
 0.250411 0.470237 0.882247 0.825312
 0.840727 0.855436 0.326955 0.67196
```

```
m.inv()
```

```
-66.3407 -308.556 110.994 335.334
 65.0489 295.544 -107.175 -320.762
 -29.406 -145.226 53.2812 155.796
 14.5004 80.4728 -28.3571 -85.5287
```

```
m^10
```

```
3069.09 3555.82 2487.74 3214.89
 5366.08 6217.09 4349.63 5621.02
 3862.1 4474.59 3130.53 4045.58
 4269.86 4947.02 3461.06 4472.71
```

```

```

Cython

```
# Decide if an integer is a power of 2
def is2pow(n):
    while n != 0 and n%2 == 0:
        n = n >> 1
    return n == 1
```

```
time [n for n in range(10^5) if is2pow(n)]
```

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192,
16384, 32768, 65536]
CPU time: 1.71 s,  Wall time: 1.74 s
```

Now we implement the same function but in Cython, so we get to use C data types.

```
%cython
def is2pow(unsigned int n):
    while n != 0 and n%2 == 0:
        n = n >> 1
    return n == 1
```

__Users_wa...9_code_sage72_spyx.c __Users_wa...sage72_spyx.pyx.ht

```
time [n for n in range(10^5) if is2pow(n)]
```

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192,
16384, 32768, 65536]
CPU time: 0.03 s,  Wall time: 0.03 s
```

The result if **a lot faster**!

```
1.74/0.03
```

```
58.0000000000000
```

```

```

Interact

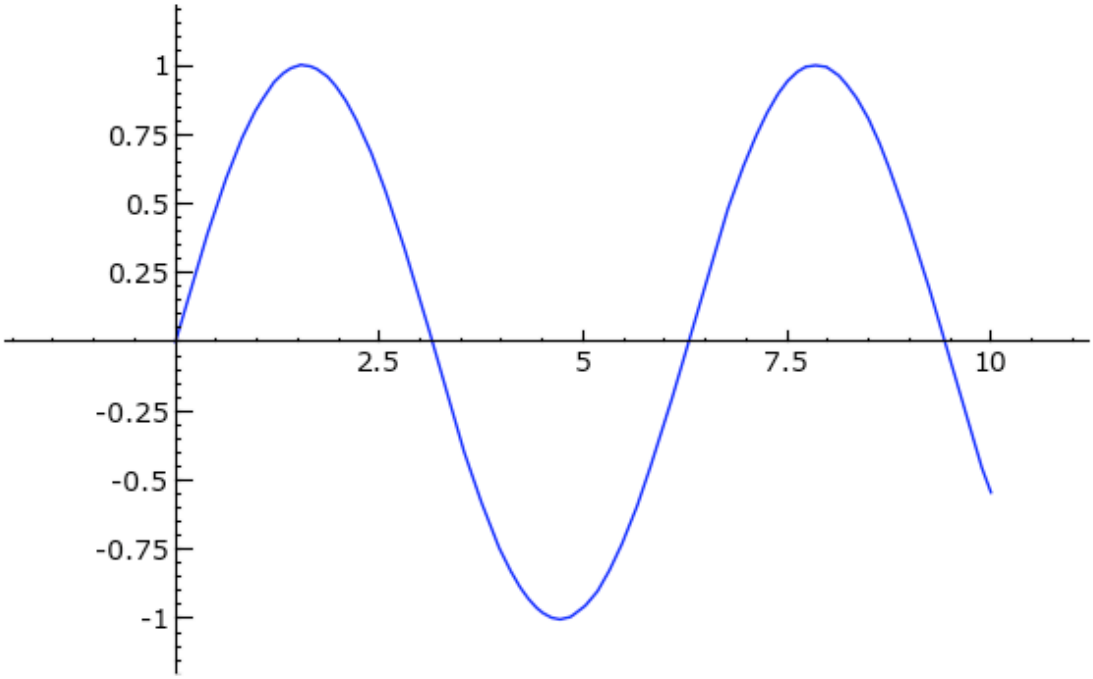
Many of my demos in class will user @interact, which is a simple facility in Sage for making interactive controls. Just put @interact on the line before a function definition, and the function becomes interactive.

```
@interact
def example(n = 15, f=sin(x)):
    print factor(n)
    show(plot(f, 0, 10))
```

n 15

f sin(x)

```
3 * 5
e300
```




```
%hide
```

example | Boy's Surface | Cross cap | Double Heart | Green bowtie | Heart | Maeder's Owl | Star of David | Two Interlinked Tori

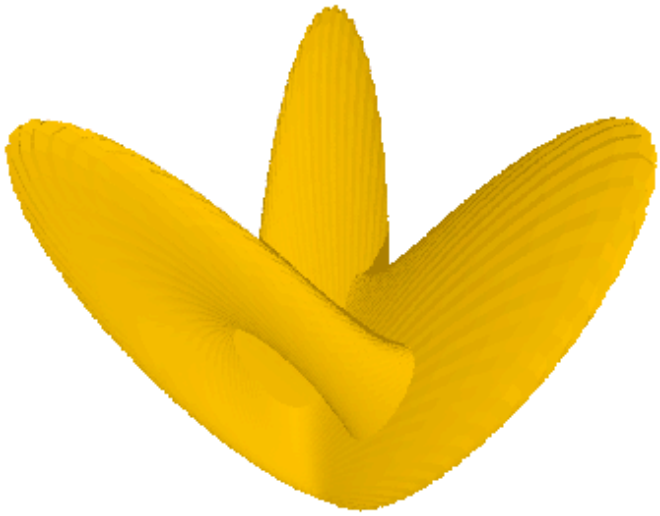
Raytrace ☑

Frame ☐

opacity

Boy's Surface

http://en.wikipedia.org/wiki/Boy's_surface



Algebraic Computing

Groups Rings and Fields

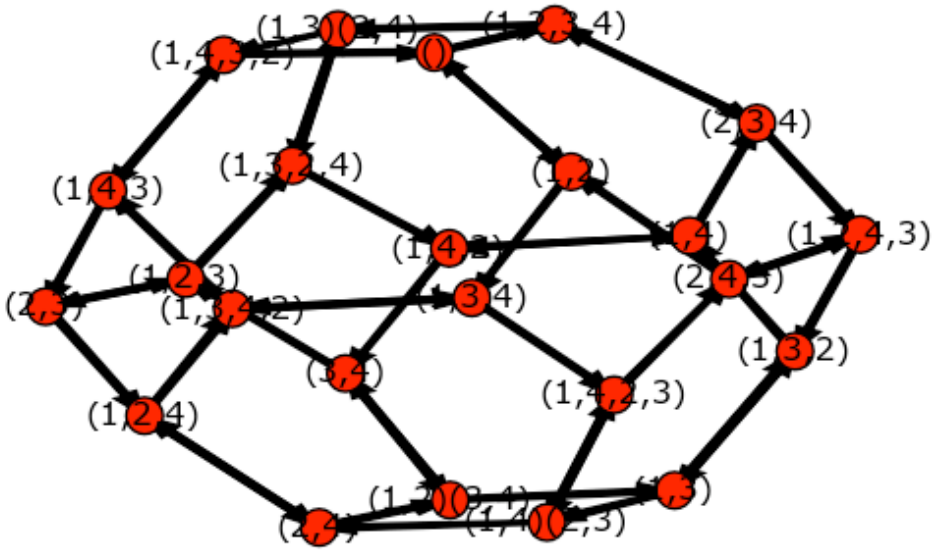
```
G = SymmetricGroup(4); G
```

```
Symmetric group of order 4! as a permutation group
```

```
for H in G.derived_series(): print H.order(), H
```

```
24 Permutation Group with generators [(1,2,3,4), (1,2)]
12 Permutation Group with generators [(1,3,2), (1,4,3)]
4 Permutation Group with generators [(1,4)(2,3), (1,2)(3,4)]
1 Permutation Group with generators [()]
```

```
H = G.cayley_graph(); H.show()
```



```
ZZ
```

Integer Ring

```
Integers(12)
```

Ring of integers modulo 12

```
QQ['x,y']
```

Multivariate Polynomial Ring in x, y over Rational Field

```
QQ
```

Rational Field

```
RDF
```

Real Double Field

```
CDF
```

Complex Double Field

```
RIF
```

Real Interval Field with 53 bits of precision

```
GF(9,'a')
```

Finite Field in a of size 3^2

```
QQbar
```

Algebraic Field

```
@interact
def _(q=9):
    if q > 1000   :
        print "q too big"; return
    K.<a> = GF(q)
    print K
    if not q.is_prime():
       print "a is a root of ", a.minpoly()
    print list(K)
```

q 9

```
Finite Field in a of size 3^2
a is a root of  x^2 + 2*x + 2
[0, 2*a, a + 1, a + 2, 2, a, 2*a + 2, 2*a + 1, 1]
```

Public Key Cryptography

%hide

New example

Number of bits of prime

8-Bit Diffie-Hellman Key Exchange

1. Alice and Bob agree to use the prime number p=179 and base g=2.
2. Alice chooses the secret integer a=162, then sends Bob (g^a mod p): 2^{162} mod 179 = 57.
3. Bob chooses the secret integer b=92, then sends Alice (g^b mod p): 2^{92} mod 179 = 171.
4. Alice computes $(g^b \text{ mod } p)^a$ mod p: 171^{162} mod 179 = **107**.
5. Bob computes $(g^a \text{ mod } p)^b$ mod p: 57^{92} mod 179 = **107**.

Linear Algebra

```
a = random_matrix(ZZ, 7); show(a); show(charpoly(a))
```

$$\begin{pmatrix} 37 & 1 & -2 & -1 & 0 & 1 & -5 \\ -1 & -9 & -1 & 0 & -7 & -498 & -19 \\ 1 & 0 & -1 & 1 & 1 & 0 & 1 \\ 0 & -10 & 1 & 42 & -8 & 0 & 32 \\ -3 & -1 & 3 & 1 & 0 & 0 & 0 \\ 0 & 2 & 16 & -2 & -1 & 1 & -4 \\ 4 & -2 & 1 & -4 & -2 & 0 & -1 \end{pmatrix}$$

$$x^7 - 69x^6 + 1877x^5 - 47017x^4 + 573758x^3 + 9491018x^2 - 47988595x + 72559345$$

```
a = random_matrix(ZZ, 500)
```

```
time a.determinant()
```

```
247894668947422403766522679583432853683597642568444477448390125964(
034864425364674272545401655518594187671428432094373898246829458119(
569312400199084152877982667644050888819854194456868240031819476597(
155261975820785865030036766491808054304784237583539962379352040377(
278600470274839682197357722642186362644973170481792801607260727867(
303961783185699940948732495239834708196468958857773981375770780584(
520743282974480123996293364034421668127395060683212138525642350044(
468193286520707215203802874470248149214175613081370534418088923075(
547671284338264646982642052566058008290559766026160236590620932320(
149367844912893852995099208720373130086034635355180671645490779372(
918177413395773999714872459335016008588143980155597511975617785534(
199304254520165770300792235904076234098190560323550997015628381781(
391865473066576084160510026699301273182069651764302970810746686689(
163171949858937442952745693327642607887529194959011688135497267795(
580439920155777887495542875842270414371304846784458872192198852536(
926590373609004848419470573017293928776193778691170753644773343291(
116574572017887916557270068498417730557049147823316417326955316558(
999867170878623807893182109562263727248284135991028139650829119252(
631495289149509794390490141261753862067621553035884005890115785442(
801637365733369756227530207056045312795236872179O
CPU time: 2.87 s,  Wall time: 2.84 s
```

```

```

```

```

Scientific Computing

Floating Point Numbers

```
2.0 + 3.0
```

```
5.00000000000000
```

```
html(r'<h1>Pi in various foating point fields</h1>')

@interact
def _(field = selector([(RDF, "Real Double Field"), (CDF,
"Complex Double Field"),
      (RealField, 'Multiprecision Real Field'),
      (ComplexField, 'Multiprecision Complex Field'),
      (RealIntervalField, 'Interval Arithmetic Field')]),
bits=53):
     if field == RealField: field = RealField(bits)
     if field == ComplexField: field = ComplexField(bits)
     if field == RealIntervalField: field =
RealIntervalField(bits)
     print field
     print "pi in this field: ", field(pi)
```

Pi in various foating point fields

field Real Double Field

bits 53

```
Real Double Field
pi in this field:  3.14159265359
pi^2 in this field:  9.86960440109
```

Numerical Linear Algebra

```
a = random_matrix(RDF, 3); a
```

```
[ 0.965919420107  0.222871093649 -0.177543827881]
[ 0.746396561966  0.690857102019  -0.64847199975]
[ 0.565781461804 -0.150165136508 -0.992689446543]
```

```
P, L, U = a.LU()
```

```
U
```

```
[ 0.965919420107  0.222871093649 -0.177543827881]
[            0.0  0.518637541465  -0.51127825463]
[            0.0             0.0  -1.16542156989]
```

```
a = random_matrix(RDF, 1000)
```

```
time P,L,U = a.LU()
```

```
CPU time: 1.66 s,  Wall time: 1.72 s
```

Symbolic Integration

```
integrate(sin(x)*cos(2*x), x)
```

```
cos(x)/2 - cos(3*x)/6
```

```
show(integrate(sin(x)*tan(x), x))
```

$$\frac{\log(\sin(x)+1)}{2} - \frac{\log(\sin(x)-1)}{2} - \sin(x)$$

```
import scipy.stats
```

```
v = [1..100]; v
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

```
scipy.stats.std(v)
```

```
29.011491975882016
```

```
scipy.stats.mean(v)
```

```
50.5
```

Statistical Computing

scipy.stats

```
scipy.stats.ttest_1samp([1..100], 50)
```

```
(0.17234549688642783, 0.86351775368618711)
```

The R Project for Statistical Computing

```
import rpy
```

```
rpy.mean([1..100])
```

```
50.5
```

```
rpy.std([1..100])
```

```
28.866070047722118
```

```

```