**Seattle University -- April 23, 2008**

# Sage Demo For Univ of Puget Sound Talk: April 8, 2008

## Basic Arithmetic

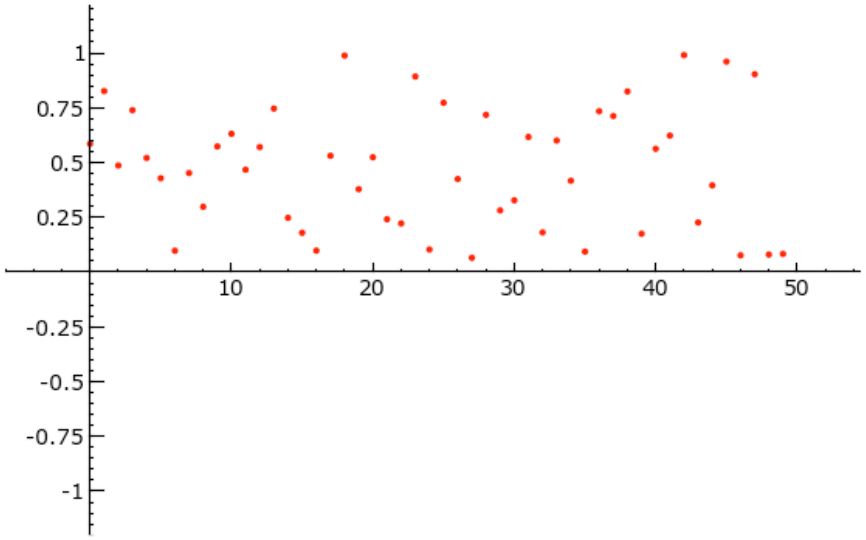
```
import scipy.stats

@interact
def example(n = (50..1000), clr=Color('red')):
   v = [random() for _ in range(n)]
   show(point(list(enumerate(v)), rgbcolor=clr.rgb()))
   print "n = ", n
   print "mean = ", scipy.stats.mean(v)
   print "std dev = ", scipy.stats.std(v)
```

n

clr #ff0000

```
n =  50
mean =  0.473305379818
std dev =  0.275798761039
```




```
2 + 2
```

```
4
```

```
expand((x+pi)^3)
```

```
x^3 + 3*pi*x^2 + 3*pi^2*x + pi^3
```

```
a = random_matrix(QQ, 10,20)
```

```
show(a)
```

$$\begin{pmatrix} 1 & 1 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & -1 & 2 & 0 & 0 & 1 & 0 & 0 & -2 & 1 & -2 & 0 \\ -1 & 1 & 1 & -2 & 0 & -1 & -1 & 0 & 1 & 0 & 1 & 2 & -2 & \frac{1}{2} & 0 & 0 & -1 & -1 & 1 & 1 \\ \frac{1}{2} & 0 & -2 & -2 & \frac{1}{2} & 0 & -2 & -1 & 1 & 1 & -2 & 1 & 0 & -1 & -1 & -\frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 1 & -\frac{1}{2} & -2 & 0 & -2 & 0 & 0 & -1 & -2 & 2 & 1 & 0 & -1 & 1 & 1 & 0 & -2 & 0 \\ 0 & 2 & 0 & -1 & -2 & -2 & 0 & -1 & \frac{1}{2} & -1 & 0 & 0 & 1 & 1 & 0 & 0 & -2 & 1 & -1 & 0 \\ -1 & -1 & -1 & -1 & 2 & -1 & -2 & 2 & -2 & 1 & -2 & -2 & -1 & -1 & 1 & 0 & 0 & 1 & -2 & 0 \\ -2 & -2 & 1 & 2 & \frac{1}{2} & 0 & -\frac{1}{2} & 1 & -2 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & 1 & -1 & -1 \\ -2 & 2 & 0 & \frac{1}{2} & 2 & 2 & 0 & 0 & -2 & -1 & 2 & 0 & 2 & 0 & -1 & -1 & -1 & -2 & -2 & 0 \\ 1 & -1 & 0 & 0 & 0 & -2 & -1 & 1 & 2 & -1 & -2 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & \frac{1}{2} & 0 \\ 2 & -2 & -2 & -2 & -1 & -\frac{1}{2} & 0 & 0 & -1 & 0 & 0 & -1 & \frac{1}{2} & 1 & 1 & -1 & 0 & -1 & 0 & 1 \end{pmatrix}$$

```
show(a.echelon_form())
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{94141}{666724} & -\frac{18305}{333362} & -\frac{68351}{166681} & -\frac{48901}{666724} & -\frac{316771}{666724} & \frac{36921}{333362} & -\frac{666641}{666724} & \frac{335269}{333362} & -\frac{2}{2} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{440577}{666724} & -\frac{146643}{333362} & \frac{53123}{166681} & -\frac{327755}{666724} & \frac{89951}{666724} & \frac{94103}{333362} & \frac{178373}{666724} & \frac{158183}{333362} & -\frac{1}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{952157}{666724} & \frac{391591}{333362} & -\frac{222736}{166681} & \frac{296477}{666724} & -\frac{294043}{666724} & \frac{8795}{333362} & -\frac{981665}{666724} & \frac{206353}{333362} & -\frac{8}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{278221}{333362} & -\frac{85499}{166681} & \frac{94488}{166681} & -\frac{136129}{333362} & -\frac{107117}{333362} & \frac{56771}{166681} & \frac{35973}{333362} & \frac{167821}{166681} & -\frac{9}{1} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \frac{946273}{666724} & \frac{84915}{333362} & -\frac{135119}{166681} & \frac{202079}{666724} & -\frac{544727}{666724} & -\frac{148053}{333362} & -\frac{1317989}{666724} & \frac{66003}{333362} & -\frac{7}{2} \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -\frac{169797}{333362} & \frac{47559}{166681} & \frac{96085}{166681} & -\frac{143699}{333362} & \frac{55455}{333362} & \frac{20391}{166681} & \frac{588217}{333362} & -\frac{193189}{166681} & \frac{6}{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \frac{704723}{666724} & -\frac{140215}{333362} & -\frac{35677}{166681} & \frac{300341}{666724} & \frac{455895}{666724} & -\frac{99903}{333362} & \frac{68929}{666724} & -\frac{191547}{333362} & \frac{3}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -\frac{658249}{333362} & -\frac{229650}{166681} & \frac{87593}{166681} & -\frac{138612}{166681} & \frac{410249}{333362} & \frac{127108}{166681} & \frac{969849}{333362} & -\frac{112343}{166681} & \frac{3}{7} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -\frac{249235}{333362} & \frac{8189}{166681} & \frac{17284}{166681} & -\frac{107917}{333362} & \frac{12829}{333362} & \frac{50993}{166681} & \frac{429591}{333362} & -\frac{134635}{166681} & \frac{18}{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -\frac{470807}{666724} & -\frac{14865}{333362} & -\frac{155806}{166681} & -\frac{432807}{666724} & \frac{454101}{666724} & \frac{85983}{333362} & \frac{391251}{666724} & \frac{378163}{333362} & \frac{4}{2} \end{pmatrix}$$

```
a = random_matrix(ZZ,100,x=-9,y=9)
```

```
a[0]
```

```
(-2, 6, -6, 1, 0, -9, -8, -9, -4, 7, 4, -8, -6, -8, -1, 4, -9, -9,
-6, -5, -4, -8, 7, 7, 0, 2, 3, -9, -9, 4, -8, 6, -8, 5, -2, -8, -2,
-6, -2, 1, -2, 1, 5, -5, -5, -7, 7, -3, 5, 8, -3, -8, -4, 2, -4, -1,
-3, -9, -2, 3, -1, 6, 7, 0, 2, -1, 8, -3, 7, -3, -4, 2, 2, 1, -3, 7,
1, 3, 6, -1, -2, -2, 2, 2, 8, -1, -4, 5, 1, -5, -1, -8, 8, -8, -7,
7, 8, 8, 5, 3)
```

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

```
19783673660813282675489468868865466483337196935834619733328845238330\
57389719552756670612726682503574400570335744840560261620726168701872\
08247179311960
CPU time: 0.04 s,  Wall time: 0.05 s
```

```
a = random_matrix(ZZ,500,x=-9,y=9)
```

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

```
97156110604562424757445380408416207503618472811347286206635544585121\
87003285113662379928217099977316482123188416298790928143927654636636\
79106574347146917776428646179477911181156946762799627800125520411930\
77631584072355606152732079517859492404664000546164560954194971169196\
95538725383048562230877049062997289477412528456340264906243056541048\
31815970687300832984695121732423878504288549662001628953230401341055\
22456768534421518506599213844853992154743099592294533575776271099660\
64988726142838872157669419165987397206970278318504656599043614531365\
95384221077904807803058969307424972344353799068888827488109101397812\
87435861687461273313269646689065903169762400584343701341313602519313\
```

```
491282875731428787642886068567649719152916416288405316613631845439091\
199272031349756887349609733207919080583916876976137766426239274341191\
285033591701629356672808067355784330220428413280657392137057928307961\
76406602765972314006244275711624391830400
CPU time: 2.28 s,  Wall time: 2.19 s
```

# 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

```
f = sin(x) + log(2*x) + pi; f
```

```
log(2*x) + sin(x) + pi
```

```
g = mathematica(f)
```

```
g
```
```
    Pi + Log[2*x] + Sin[x]
```
```
g.Integrate(x)
```
```
    -x + Pi*x - Cos[x] + x*Log[2*x]
```
```
type(g)
```
```
    <class 'sage.interfaces.mathematica.MathematicaElement'>
```
```
f._maple_init_()
```
```
    '((sin(x)) + (log((2) * (x)))) + (Pi)'
```
```
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
```
```
    0.942284 0.473816 0.745115 0.561523
     0.492998 0.623873 0.0561246 0.478341
     0.659267 0.513218 0.587767 0.796376
     0.372597 0.231948 0.907144 0.898596
```
```
m.inv()
```
```
    -0.471217 -0.778463 2.59322 0.0887005
     1.17364 4.56033 -2.21193 -3.02317
     -1.85425 -5.24752 2.08441 5.45862
     0.34253 -3.78197 -0.25153 2.8411
```
```
m^10
```
```
    1811.47 1292.63 1835.5 2077.54
     997.565 711.844 1010.8 1144.09
     1663.06 1186.73 1685.12 1907.33
     1619.88 1155.92 1641.38 1857.82
```
```

```

# 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: 1.68 s,  Wall time: 1.69 s
```

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

```
%cython
def is2pow(int 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: 0.02 s,  Wall time: 0.02 s
```

The result if **a lot faster**!

```
1.68/0.02
```

```
    84.0000000000000
```

```
is2pow(2^10)
```

```
    True
```

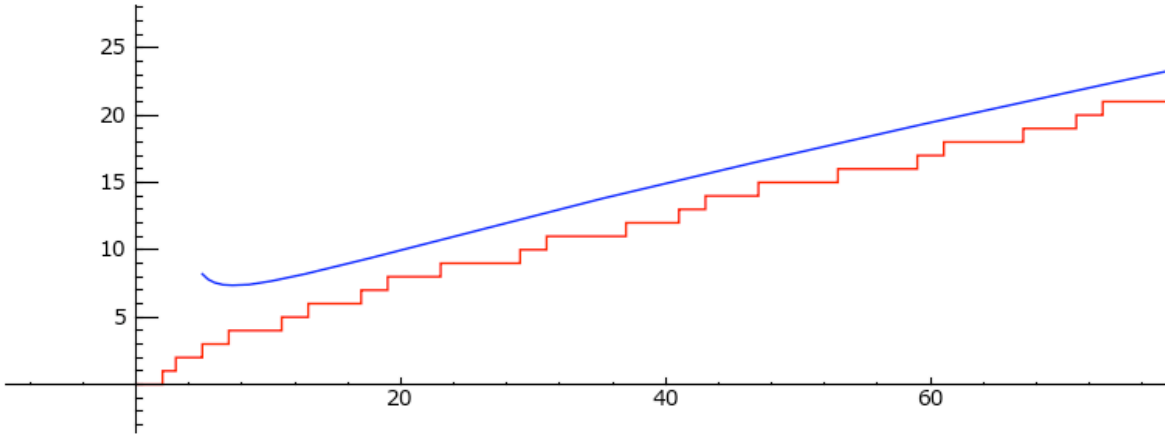
# Interact

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 _(N=(100,(2..4000))):
   html("<font color='red'>$\pi(x)$</font> and <font
color='blue'>$x/(\log(x)-1)$</font> for $x < %s$"%N)
   show(plot(prime_pi, 0, N, rgbcolor='red') + plot(x/(log(x)-1), 5, N,
rgbcolor='blue'), figsize=[10,3])
```

N

$\pi(x)$ and $x/(\log(x)-1)$ for $x < 88$




```
%hide
var('u,v')
plots = ['Two Interlinked Tori', 'Star of David', 'Double Heart',
```

```
        'Heart', 'Green bowtie', "Boy's Surface", "Maeder's Owl",
        'Cross cap']
plots.sort()

@interact
def _(example=selector(plots, buttons=True, nrows=2),
     tachyon=("Raytrace", True), frame = ('Frame', False),
     opacity=(1,(0.1,1))):
   url = ''
   if example == 'Two Interlinked Tori':
       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), color="red",
opacity=opacity)
       p2 = parametric_plot3d(f2, (u,0,2*pi), (v,0,2*pi),
color="blue",opacity=opacity)
       P = p1 + p2
   elif example == 'Star of David':
       f_x = cos(u)*cos(v)*(abs(cos(3*v/4))^500 + abs(sin(3*v/4))^500)^(-1/260)*
(abs(cos(4*u/4))^200 + abs(sin(4*u/4))^200)^(-1/200)
       f_y = cos(u)*sin(v)*(abs(cos(3*v/4))^500 + abs(sin(3*v/4))^500)^(-1/260)*
(abs(cos(4*u/4))^200 + abs(sin(4*u/4))^200)^(-1/200)
       f_z = sin(u)*(abs(cos(4*u/4))^200 + abs(sin(4*u/4))^200)^(-1/200)
       P = parametric_plot3d([f_x, f_y, f_z], (u, -pi, pi), (v, 0,
2*pi),opacity=opacity)
   elif example == 'Double Heart':
       f_x = ( abs(v) - abs(u) - abs(tanh((1/sqrt(2))*u)/(1/sqrt(2))) +
abs(tanh((1/sqrt(2))*v)/(1/sqrt(2))) )*sin(v)
       f_y = ( abs(v) - abs(u) - abs(tanh((1/sqrt(2))*u)/(1/sqrt(2))) -
abs(tanh((1/sqrt(2))*v)/(1/sqrt(2))) )*cos(v)
       f_z = sin(u)*(abs(cos(4*u/4))^1 + abs(sin(4*u/4))^1)^(-1/1)
       P = parametric_plot3d([f_x, f_y, f_z], (u, 0, pi), (v, -pi,
pi),opacity=opacity)
   elif example == 'Heart':
       f_x = cos(u)*(4*sqrt(1-v^2)*sin(abs(u))^abs(u))
       f_y = sin(u) *(4*sqrt(1-v^2)*sin(abs(u))^abs(u))
       f_z = v
       P = parametric_plot3d([f_x, f_y, f_z], (u, -pi, pi), (v, -1, 1), frame=False,
color="red",opacity=opacity)
   elif example == 'Green bowtie':
       f_x = sin(u) / (sqrt(2) + sin(v))
       f_y = sin(u) / (sqrt(2) + cos(v))
       f_z = cos(u) / (1 + sqrt(2))
       P = parametric_plot3d([f_x, f_y, f_z], (u, -pi, pi), (v, -pi, pi),
frame=False, color="green",opacity=opacity)
   elif example == "Boy's Surface":
       url = "http://en.wikipedia.org/wiki/Boy's_surface"
       fx = 2/3* (cos(u)* cos(2*v) + sqrt(2)* sin(u)* cos(v))* cos(u) / (sqrt(2) -
sin(2*u)* sin(3*v))
       fy = 2/3* (cos(u)* sin(2*v) - sqrt(2)* sin(u)* sin(v))* cos(u) / (sqrt(2) -
sin(2*u)* sin(3*v))
       fz = sqrt(2)* cos(u)* cos(u) / (sqrt(2) - sin(2*u)* sin(3*v))
       P = parametric_plot3d([fx, fy, fz], (u, -2*pi, 2*pi), (v, 0, pi), plot_points
= [90,90], frame=False, color="orange",opacity=opacity)
   elif example == "Maeder's Owl":
       fx = v *cos(u) - 0.5* v^2 * cos(2* u)
       fy = -v *sin(u) - 0.5* v^2 * sin(2* u)
       fz = 4 *v^1.5 * cos(3 *u / 2) / 3
       P = parametric_plot3d([fx, fy, fz], (u, -2*pi, 2*pi), (v, 0, 1),plot_points =
```

```
[90,90], frame=False, color="purple",opacity=opacity)
   elif example =='Cross cap':
       url = 'http://en.wikipedia.org/wiki/Cross-cap'
       fx = (1+cos(v))*cos(u)
       fy = (1+cos(v))*sin(u)
       fz = -tanh((2/3)*(u-pi))*sin(v)
       P = parametric_plot3d([fx, fy, fz], (u, 0, 2*pi), (v, 0, 2*pi), frame=False,
color="red",opacity=opacity)
   else:
       print "Bug selecting plot?"
       return


   html('<h2>%s</h2>'%example)
   if url:
       html('<h3><a target="_new" href="%s">%s</a></h3>'%(url,url))
   show(P, viewer='tachyon' if tachyon else 'jmol', frame=frame)
```

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

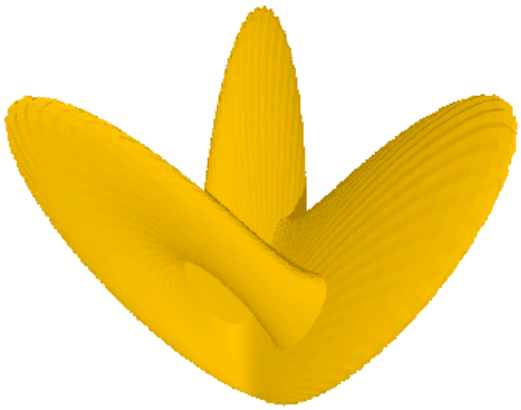
Raytrace [x]

Frame [ ]

opacity

**Boy's Surface**

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



# Public Key Cryptography

```
%hide

@interact
def diffie_hellman(button=selector(["New example"],label='',buttons=True),
   bits=("Number of bits of prime", (8,12,..512))):
   maxp = 2^bits
   p = random_prime(maxp)
   k = GF(p)
   if bits>100:
       g = k(2)
   else:
       g = k.multiplicative_generator()
   a = ZZ.random_element(10, maxp)
   b = ZZ.random_element(10, maxp)

   print """
<html>
<style>
.gamodp {
background:yellow
}
.gbmodp {
background:orange
}
.dhsame {
```

```
color:green;
font-weight:bold
}
</style>
<h2>%s-Bit Diffie-Hellman Key Exchange</h2>
<ol style="color:#000;font:18px Arial, Helvetica, sans-serif">
<li>Alice and Bob agree to use the prime number p=%s and base g=%s.</li>
<li>Alice chooses the secret integer a=%s, then sends Bob (<span
class="gamodp">g<sup>a</sup> mod p</span>):<br/>%s<sup>%s</sup> mod %s = <span
class="gamodp">%s</span>.</li>
<li>Bob chooses the secret integer b=%s, then sends Alice (<span
class="gbmodp">g<sup>b</sup> mod p</span>):<br/>%s<sup>%s</sup> mod %s = <span
class="gbmodp">%s</span>.</li>
<li>Alice computes (<span class="gbmodp">g<sup>b</sup> mod p</span>)<sup>a</sup> mod
p:<br/>%s<sup>%s</sup> mod %s = <span class="dhsame">%s</span>.</li>
<li>Bob computes (<span class="gamodp">g<sup>a</sup> mod p</span>)<sup>b</sup> mod
p:<br/>%s<sup>%s</sup> mod %s = <span class="dhsame">%s</span>.</li>
</ol></html>
   """ % (bits, p, g, a, g, a, p, (g^a), b, g, b, p, (g^b), (g^b), a, p,
      (g^ b)^a, g^a, b, p, (g^a)^b)
```

New example

Number of bits of prime

**8-Bit Diffie-Hellman Key Exchange**

1. Alice and Bob agree to use the prime number p=223 and base g=3.
2. Alice chooses the secret integer a=45, then sends Bob ($g^a$ mod p):
   $3^{45}$ mod 223 = 125.
3. Bob chooses the secret integer b=172, then sends Alice ($g^b$ mod p):
   $3^{172}$ mod 223 = 19.
4. Alice computes ($g^b$ mod p)$^a$ mod p:
   $19^{45}$ mod 223 = **64**.
5. Bob computes ($g^a$ mod p)$^b$ mod p:
   $125^{172}$ mod 223 = **64**.

# Scientific Computing

## Numerical Linear Algebra

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

```
[ -0.222249908591   0.477528335163  -0.210976537194]
[  0.311240578904 -0.0555639100136   0.206329748457]
[ -0.697319814849   0.789102843781   0.762639350439]
```

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

```
P
```

```
[0.0 0.0 1.0]
[0.0 1.0 0.0]
[1.0 0.0 0.0]
```

```
U
```

```
[-0.697319814849  0.789102843781  0.762639350439]
[             0.0   0.29664295502  0.546724941411]
[             0.0             0.0  -0.87061889312]
```

```
L
```

```
[             1.0             0.0             0.0]
[-0.446338354765             1.0             0.0]
[ 0.318720196756  0.761944006112             1.0]
```

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

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

```
CPU time: 1.73 s,  Wall time: 1.81 s
```

```

```

```

```

## Symbolic Computation

```
f = integrate(sin(x)*cos(2*x), x)
show(f)
```

$$\frac{\cos\left(x\right)}{2} - \frac{\cos\left(3x\right)}{6}$$

```
latex(f)
```

```
\frac{\cos \left( x \right)}{2} - \frac{\cos \left( {3 x}
\right)}{6}
```

```
f = integrate(sin(x^2))
f
```

```
sqrt(pi)*((sqrt(2)*I + sqrt(2))*erf((sqrt(2)*I + sqrt(2))*x/2) +
(sqrt(2)*I - sqrt(2))*erf((sqrt(2)*I - sqrt(2))*x/2))/8
```

```
show(f)
```

$$\frac{\sqrt{\pi}\left(\left(\sqrt{2}i+\sqrt{2}\right)\operatorname{erf}\left(\frac{\left(\sqrt{2}i+\sqrt{2}\right)x}{2}\right)+\left(\sqrt{2}i-\sqrt{2}\right)\operatorname{erf}\left(\frac{\left(\sqrt{2}i-\sqrt{2}\right)x}{2}\right)\right)}{8}$$

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

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

# Statistical Computing

## scipy.stats

```
import scipy.stats
```

```
scipy.stats
```

```
<module 'scipy.stats' from
'/Users/was/build/sage/local/lib/python2.5/site-packages/scipy/stats\
/__init__.pyc'>
```

```
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
```

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

```
(0.17234549688642783, 0.86351775368618711)
```

# The R Project for Statistical Computing

```
r = R() # interface to R
```

```
a = r([1..100])
```

```
a.mean()
```

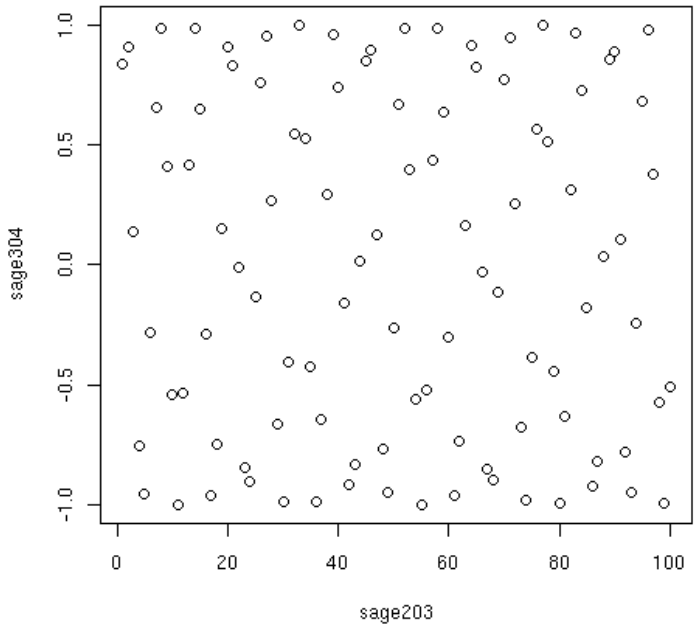
```
    [1] 50.5
```

```
a.sd()
```

```
    [1] 29.01149
```

```
_ = r.png(file='a.png')
x = r([1..100])
y = r([sin(n) for n in [1..100]])
print y.mean()
r.plot(x,y)
_ = r.dev_off()
```

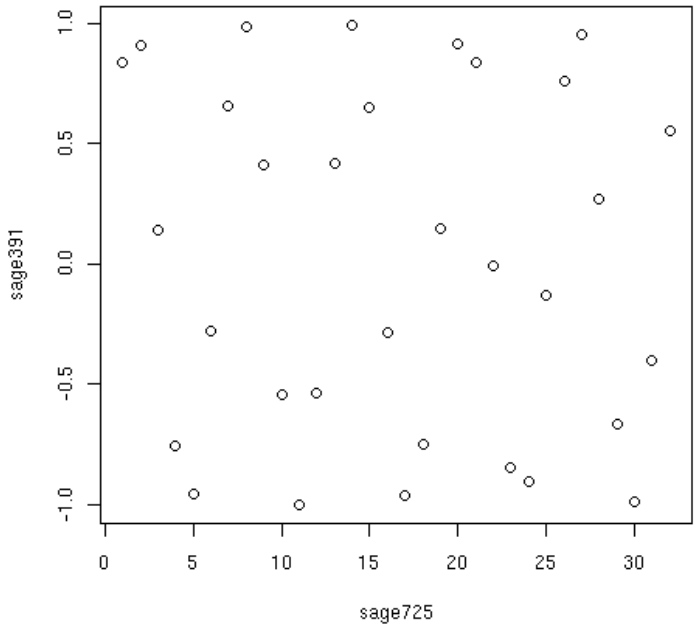
```
    [1] -0.001271710
```




```
@interact
def _(vals=(10..300)):
   _ = r.png(file='a.png')
   x = r([1..vals])
   y = r([sin(n) for n in [1..vals]])
   print y.mean()
   r.plot(x,y)
   _ = r.dev_off()
```

vals

```
[1] 0.01335748
```


sage391
1.0
0.5
0.0
-0.5
-1.0
0
5
10
15
20
25
30
sage725