Python Montréal 29 nov 2010

by Franco Saliola and Sébastien Labbé

Two important (but minor) differences between Sage language and Python

Integer division in Python :

```
%python
2/3 + 4/5 + 1/7
```

```
0
```

in Sage:

```
2/3 + 4/5 + 1/7
```

```
169/105
```

Exponent (^) in Python :

```
%python
10^14  #exclusive OR
```

```
4
```

in Sage :

```
10^14
```

```
100000000000000
```

The preparser

```
preparse('2/3 + 2^3 + 3.0')
```

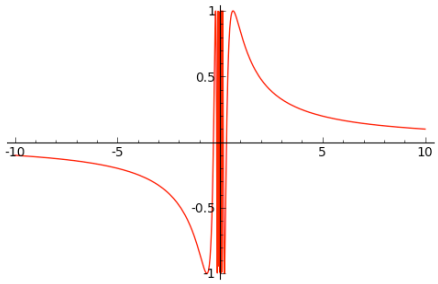
```
"Integer(2)/Integer(3) + Integer(2)**Integer(3) + RealNumber('3
```

```
preparse('2^3')
```

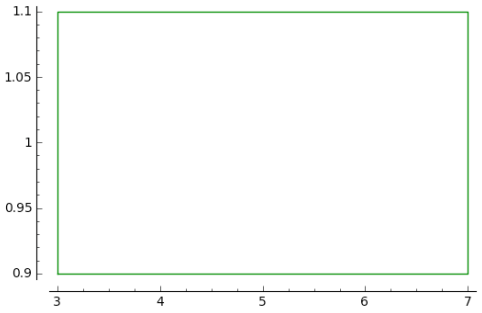
```
'Integer(2)**Integer(3)'
```


2D Plots

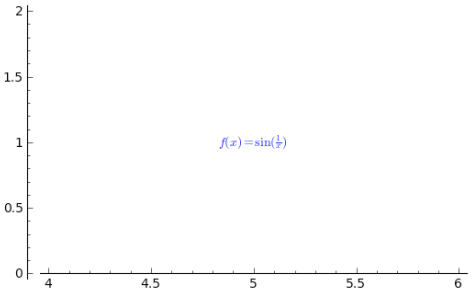
```
f = sin(1/x)
P = plot(f, -10, 10, color='red')
P
```



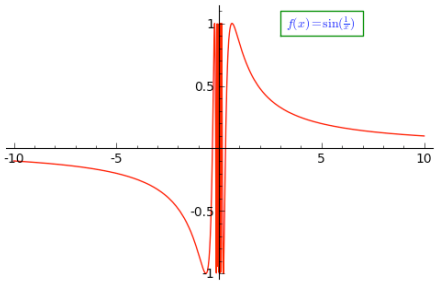

```
Q = line([(3,0.9), (7,0.9), (7,1.1), (3,1.1), (3,0.9)],
color='green')
Q
```




```
R = text('$f(x) = \\sin(\\frac{1}{x})$', (5,1))
R
```




```
Q + R + P
```



L'outil interact (exemples tirés du wiki de Sage : http://wiki.sagemath.org/)

Curves of Pursuit

by Marshall Hampton.

```
%hide
npi = RDF(pi)
from math import cos,sin
def rot(t):
    return matrix([[cos(t),sin(t)],[-sin(t),cos(t)]])

def pursuit(n,x0,y0,lamb,steps = 100, threshold = .01):
    paths = [[[x0,y0]]]
    for i in range(1,n):
        rx,ry = list(rot(2*npi*i/n)*vector([x0,y0]))
        paths.append([[rx,ry]])
    oldpath = [x[-1] for x in paths]
    for q in range(steps):
        diffs = [[oldpath[(j+1)%n][0]-oldpath[j]
[0],oldpath[(j+1)%n][1]-oldpath[j][1]] for j in range(n)]
        npath = [[oldpath[j][0]+lamb*diffs[j][0],oldpath[j]
[1]+lamb*diffs[j][1]] for j in range(n)]
        for j in range(n):
            paths[j].append(npath[j])
        oldpath = npath
    return paths
html('<h3>Curves of Pursuit</h3>')
@interact
def curves_of_pursuit(n = slider([2..20],default = 5, label="#
of points"),steps = slider([floor(1.4^i) for i in
range(2,18)],default = 10, label="# of steps"), stepsize =
slider(srange(.01,1,.01),default = .2, label="stepsize"),
colorize = selector(['BW','Line color', 'Filled'],default =
'BW')):
    outpaths = pursuit(n,0,1,stepsize, steps = steps)
    mcolor = (0,0,0)
    outer = line([q[0] for q in outpaths]+[outpaths[0][0]],
rgbcolor = mcolor)
    polys = Graphics()
    if colorize=='Line color':
        colors = [hue(j/steps,1,1) for j in
```

```
range(len(outpaths[0]))]
    elif colorize == 'BW':
        colors = [(0,0,0) for j in range(len(outpaths[0]))]
    else:
        colors = [hue(j/steps,1,1) for j in
range(len(outpaths[0]))]
        polys = sum([polygon([outpaths[(i+1)%n]
[j+1],outpaths[(i+1)%n][j], outpaths[i][j+1]], rgbcolor =
colors[j]) for i in range(n) for j in range(len(outpaths[0])-1)])
        #polys = polys[0]
        colors = [(0,0,0) for j in range(len(outpaths[0]))]
    nested = sum([line([q[j] for q in outpaths]+
[outpaths[0][j]], rgbcolor = colors[j]) for j in
range(len(outpaths[0]))])
    lpaths = [line(x, rgbcolor = mcolor) for x in outpaths]
    show(sum(lpaths)+nested+polys, axes = False, figsize =
[5,5], xmin = -1, xmax = 1, ymin = -1, ymax =1)
```

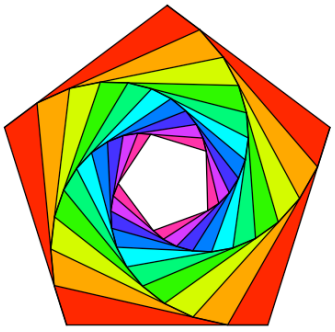

Curves of Pursuit

# of points 5

# of steps 10

stepsize 0.200000000000000

colorize BW



Factor Trees

by William Stein

```
%hide
import random
def ftree(rows, v, i, F):
    if len(v) > 0: # add a row to g at the ith level.
        rows.append(v)
```



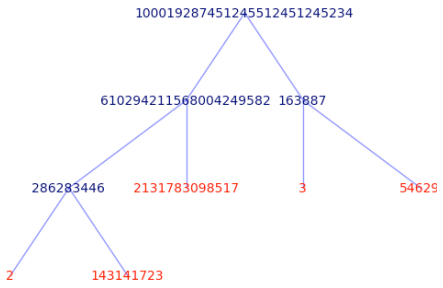
```
    w = []
    for i in range(len(v)):
        k, _, _ = v[i]
        if k is None or is_prime(k):
            w.append((None,None,None))
        else:
            d = random.choice(divisors(k)[1:-1])
            w.append((d,k,i))
            e = k//d
            if e == 1:
                w.append((None,None))
            else:
                w.append((e,k,i))
    if len(w) > len(v):
        ftree(rows, w, i+1, F)
def draw_ftree(rows,font):
    g = Graphics()
    for i in range(len(rows)):
        cur = rows[i]
        for j in range(len(cur)):
            e, f, k = cur[j]
            if not e is None:
                if is_prime(e):
                    c = (1,0,0)
                else:
                    c = (0,0,.4)
                g += text(str(e), (j*2-len(cur),-i),
fontsize=font, rgbcolor=c)
                if not k is None and not f is None:
                    g += line([(j*2-len(cur),-i),
((k*2)-len(rows[i-1]),-i+1)],
                    alpha=0.5)
    return g

@interact
def factor_tree(n=100, font=(10, (8..20)), redraw=['Redraw']):
    n = Integer(n)
    rows = []
    v = [(n,None,0)]
    ftree(rows, v, 0, factor(n))
    show(draw_ftree(rows, font), axes=False)
```


n 100

font 10

redraw Redraw



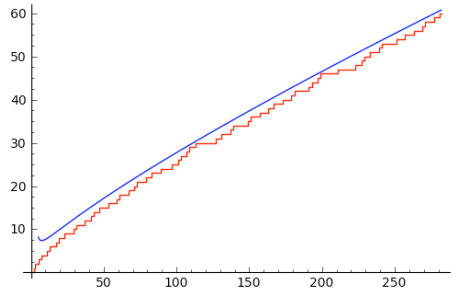
Illustrating the prime number theorem

by William Stein

```
@interact
def _(N=(100,(2..2000))):
    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'))
```

N 98

$\pi(x)$ and $x/(\log(x)-1)$ for $x < 282$



Stock Market data, fetched from Yahoo and Google

by William Stein

```
%hide

import urllib

class Day:
    def __init__(self, date, open, high, low, close, volume):
        self.date = date
        self.open=float(open); self.high=float(high);
self.low=float(low); self.close=float(close)
        self.volume=int(volume)
    def __repr__(self):
        return '%10s %4.2f %4.2f %4.2f %4.2f %10d'%(self.date,
self.open, self.high,
                    self.low, self.close, self.volume)

class Stock:
```



```
    def __init__(self, symbol):
        self.symbol = symbol.upper()

    def __repr__(self):
        return "%s (%s)"%(self.symbol, self.yahoo()['price'])

    def yahoo(self):
        url = 'http://finance.yahoo.com/d/quotes.csv?s=%s&f=%s'
% (self.symbol, 'l1c1va2xj1b4j4dyekjm3m4rr5p5p6s7')
        values =
urllib.urlopen(url).read().strip().strip('"').split(',')
        data = {}
        data['price'] = values[0]
        data['change'] = values[1]
        data['volume'] = values[2]
        data['avg_daily_volume'] = values[3]
        data['stock_exchange'] = values[4]
        data['market_cap'] = values[5]
        data['book_value'] = values[6]
        data['ebitda'] = values[7]
        data['dividend_per_share'] = values[8]
        data['dividend_yield'] = values[9]
        data['earnings_per_share'] = values[10]
        data['52_week_high'] = values[11]
        data['52_week_low'] = values[12]
        data['50day_moving_avg'] = values[13]
        data['200day_moving_avg'] = values[14]
        data['price_earnings_ratio'] = values[15]
        data['price_earnings_growth_ratio'] = values[16]
        data['price_sales_ratio'] = values[17]
        data['price_book_ratio'] = values[18]
        data['short_ratio'] = values[19]
        return data

    def historical(self):
        try:
            return self.__historical
        except AttributeError:
            pass
        symbol = self.symbol
        def get_data(exchange):
            name = get_remote_file('http://finance.google.com
/finance/historical?q=%s:%s&output=csv'%(exchange,
symbol.upper()),
                           verbose=False)
```



```
            return open(name).read()
        R = get_data('NASDAQ')
        if "Bad Request" in R:
            R = get_data("NYSE")
        R = R.splitlines()
        headings = R[0].split(',')
        self.__historical = []
        try:
            for x in reversed(R[1:]):
                date, opn, high, low, close, volume =
x.split(',')
                self.__historical.append(Day(date,
opn,high,low,close,volume))
        except ValueError:
            pass
        self.__historical =
Sequence(self.__historical,cr=True,universe=lambda x:x)
        return self.__historical

    def plot_average(self, spline_samples=10):
        d = self.historical()
        if len(d) == 0:
            return text('no historical data at Google Finance
about %s'%self.symbol, (0,3))
        avg = list(enumerate([(z.high+z.low)/2 for z in d]))
        P = line(avg) + points(avg, rgbcolor='black',
pointsize=4) + \
                text(self.symbol, (len(d)*1.05, d[-1].low),
horizontal_alignment='right', rgbcolor='black')
        if spline_samples > 0:
            k = 250//spline_samples
            spl = spline([avg[i*k] for i in range(len(d)//k)] +
[avg[-1]])
            P += plot(spl, (0,len(d)+30), color=(0.7,0.7,0.7))
        P.xmax(260)
        return P

    def plot_diff(self):
        d = self.historical()
        if len(d) == 0:
            return text('no historical data at Google Finance
about %s'%self.symbol, (0,3))
        diff = []
        for i in range(1, len(d)):
            z1 = d[i]; z0 = d[i-1]
```



```
            diff.append((i, (z1.high+z1.low)/2 - (z0.high +
z0.low)/2))
        P = line(diff,thickness=0.5) + points(diff,
rgbcolor='black', pointsize=4) + \
                text(self.symbol, (len(d)*1.05, 0),
horizontal_alignment='right', rgbcolor='black')
        P.xmax(260)
        return P

symbols = ['bsc', 'vmw', 'sbux', 'aapl', 'amzn', 'goog', 'wfmi',
'msft', 'yhoo', 'ebay', 'java', 'rht', ]; symbols.sort()
stocks = dict([(s,Stock(s)) for s in symbols])

@interact
def data(symbol = symbols, other_symbol='', spline_samples=
(8,[0..15])):
    if other_symbol != '':
        symbol = other_symbol
    S = Stock(symbol)
    html('<h1 align=center><font color="darkred">%s</font>
</h1>'%S)
    S.plot_average(spline_samples).save('avg.png', figsize=
[10,2])
    S.plot_diff().save('diff.png', figsize=[10,2])

    Y = S.yahoo()
    k = Y.keys(); k.sort()
    html('Price during last 52 weeks:<br>Grey line is a spline
through %s points (do not take seriously!):<br> <img
src="cell://avg.png">'%spline_samples)
    html('Difference from previous day:<br> <img
src="cell://diff.png">')
    html('<table align=center>' + '\n'.join('<tr><td>%s</td>
<td>%s</td></tr>'%(k[i], Y[k[i]]) for i in range(len(k))) +
'</table>')
```

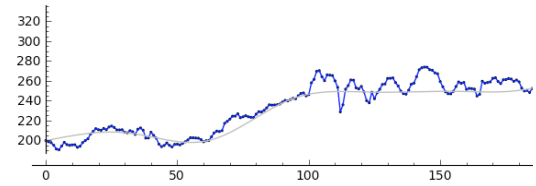
symbol aapl

other_symbol

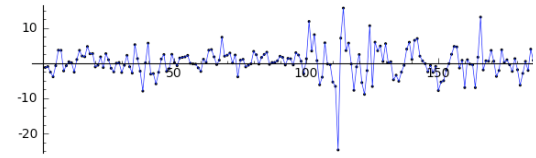
spline_samples 8

AAPL (312.06)

Price during last 52 weeks:
Grey line is a spline through 8 points (do not take seriously!):



Difference from previous day:



200day_moving_avg	272.838
50day_moving_avg	309.407
52_week_high	321.30
52_week_low	188.68
avg_daily_volume	18913800
book_value	52.175
change	-4.81
dividend_per_share	0.00
dividend_yield	N/A
earnings_per_share	15.154
ebitda	19.364B
market_cap	286.3B
price	312.06

Cryptography

The Diffie-Hellman Key Exchange Protocol

by Timothy Clemans and William Stein

```
@interact
def diffie_hellman(bits=slider(8, 513, 4, 8, 'Number of bits',
False),
    button=selector(["Show new example"],label='',buttons=True)):
    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, .gbmodp {
color:#000;
padding:5px
}
.gamodp {
background:#846FD8
}
.gbmodp {
background:#FFFC73
}
.dhsame {
color:#000;
font-weight:bold
}
</style>
<h2 style="color:#000;font-family:Arial, Helvetica,
sans-serif">%s-Bit Diffie-Hellman Key Exchange</h2>
<ol style="color:#000;font-family: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)
```


Number of bits

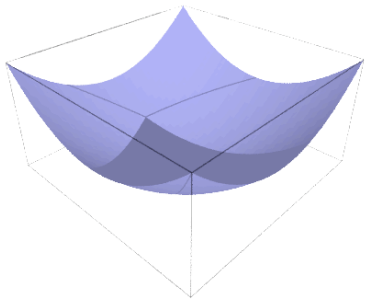
Show new example

8-Bit Diffie-Hellman Key Exchange

1. Alice and Bob agree to use the prime number p = 23 and base g = 5.
2. Alice chooses the secret integer a = 223, then sends Bob (g^a mod p):
   5^{223} mod 23 = 10 .
3. Bob chooses the secret integer b=220, then sends Alice (g^b mod p):
   5^{220} mod 23 = 1 .
4. Alice computes (g^b mod p)^a mod p:
   1^{223} mod 23 = **1**.
5. Bob computes (g^a mod p)^b mod p:
   10^{220} mod 23 = **1**.

Dessiner une fonction $\mathbb{R}^2 \mapsto \mathbb{R}$: la commande plot3d

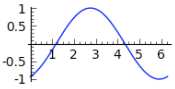
```
def f(x, y):
    return x^2 + y^2
plot3d(f, (-10,10), (-10,10), viewer='tachyon')
```



Animations

```
a = animate([sin(x + float(k)) for k in srange(0,2*pi,0.3)],
xmin=0, xmax=2*pi, figsize=[2,1])
```

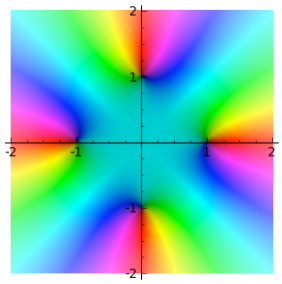
```
a.show()
```



La commande complex_plot pour les fonctions $\mathbb{C} \mapsto \mathbb{C}$

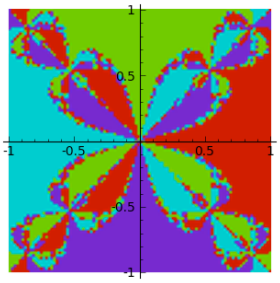
```
f(x) = x^4 - 1
```

```
complex_plot(f, (-2,2), (-2,2))
```




```
def newton(f, z, precision=0.001) :
    while abs(f(x=z)) >= precision:
        z = z - f(x=z) / diff(f)(x=z)
    return z
```

```
complex_plot(lambda z : newton(f, z), (-1,1), (-1,1))
```

Utilisation du Notebook : Écriture, édition et évaluation d'une saisie

Pour **évaluer une saisie** dans le *Notebook de Sage*, tapez la saisie dans une cellule et faites **shift-entrée** ou cliquer le lien evaluate. Essayez-le maintenant avec une expression simple (e.g., **2 + 2**). La première évaluation d'une cellule prend plus de temps que les fois suivantes, car un processus commence.

```
2+3
```

```
5
```

```
4+5
```

```
9
```

Créez de nouvelles **cellules de saisie** en cliquant sur la ligne bleue qui apparaît entre les cellules lorsque vous déplacez la souris. Essayez-le maintenant.

Vous pouvez **rééditer** n'importe quelle cellule en cliquant dessus (ou en utilisant les flèches du clavier). Retournez plus haut et changez votre 2 + 2 en un 3 + 3 et réévaluez la cellule.

Vous pouvez aussi **éditer ce texte-ci** en double cliquant dessus ce qui fera apparaître un éditeur de texte TinyMCE Javascript. Vous pouvez même ajouter des expressions mathématiques telles que $\sin(x) - y^3$ comme avec LaTeX.

$$\int e^x dx = e^x + c$$

Comment consulter l'aide contextuelle et obtenir de la documentation

Vous trouvez la **liste des fonctions** que vous pouvez appelez sur un objet X en tappant **X.<touche de tabulation>**.

```
X = 2009
```

Écrivez `X.` et appuyez sur la touche de tabulation.

```
X.factor()
```

```
7^2 * 41
```

Une fois que vous avez sélectionné une fonction, disons **factor,** tappez **X.factor(<touche de tabulation>** ou **X.factor?<touche de tabulation>** pour ***obtenir de l'aide et des exemples*** d'utilisation de cette fonction. Essayez-le maintenant avec **X.factor**.

```
4+5
```

```
9
```

Pour obtenir l'aide complète et un tutoriel plus exhaustif, cliquez sur le lien Help en haut à droite de cette page, et cliquer ensuite sur Fast Static Versions of the Documentation.

Résolution d'équations polynomiales

```
a,b,c,d,X = var('a,b,c,d,X')
```

```
s = solve(a*X^2 + b*X + c == 0, X)
show(s)
```

$$\left[X = -\frac{b+\sqrt{-4\,ac+b^2}}{2\,a}, X = -\frac{b-\sqrt{-4\,ac+b^2}}{2\,a}\right]$$

```
s = solve(a*X^3 + b*X^2 + c*X + d == 0, X)
show(s[0])
```

$$X = -\frac{1}{2}\left(i\sqrt{3}+1\right)\left(\frac{\sqrt{27\,a^2d^2+4\,ac^3-b^2c^2-2\left(9\,abc-2\,b^3\right)d\sqrt{3}}}{18\,a^2}-\frac{27\,a^2d}{18\,a^2}\right.$$

Algèbre linéaire

```
A = matrix(3, [9,4,2,4,6,1,6,4,3,2,3,4,2,7,8,6,5,3]); A
```

```
[9 4 2 4 6 1]
[6 4 3 2 3 4]
[2 7 8 6 5 3]
```

```
show(A)
```


$$\begin{pmatrix} 9 & 4 & 2 & 4 & 6 & 1 \\ 6 & 4 & 3 & 2 & 3 & 4 \\ 2 & 7 & 8 & 6 & 5 & 3 \end{pmatrix}$$

```
latex(A)
```

```
\left(\begin{array}{rrrrrr}
9 & 4 & 2 & 4 & 6 & 1 \\
6 & 4 & 3 & 2 & 3 & 4 \\
2 & 7 & 8 & 6 & 5 & 3
\end{array}\right)
```

```
r = random_matrix(ZZ, 200)
r[0]
```

```
(6, 1, -4, 1, 3, 2, 0, 4, 1, 2, 1, -2, 0, 3, 1, 5, 0, 0, 3, -4,
4, -1, -29, 2, 0, 1, 2, 4, -1, 1, 0, 1, 0, -22, 0, -2, 0, -1, -
-3, -1, 0, 1, 1, 1, -32, 1, -1, -1, 0, 5, -1, -13, 0, 2, -1, -5
-1, 0, 16, 1, 1, -5, 0, -5, -3, -1, 1, 0, 1, -6, 0, -1, 1, 1, 0
0, -2, 1, 3, 0, 2, 5, -5, 3, 0, -9, 3, -1, 5, 0, -1, -1, 3, 0,
1, 0, 3, -1, 0, 0, 1, 0, -1, 0, 0, -7, 1, 0, 0, -3, -1, 12, 1,
-74, 1, 1, 0, 1, 1164, 21, -109, -5, -2, 1, 1, 3, -30, 17, -28,
1, 161, -4, 1, 10, 2, -1, -1, 4, -6, 0, 17, 0, 25, -1, -1, -1,
-2, -1, -1, -1, 1, -6, -1, -2, 1, 2, -1, 0, -6, 1, -3, -1, 6, 0
0, -4, -1, 1, 1, 12, -7, -1, 1, -1, -1, 1, 2, 2, -25, -2, -1, 0
2, 3, 1, -3, 12, -10, 1, 0)
```

```
time r.determinant()
```

```
-152983472555375793876315950202554859056791125466280319677059860
678498643950537364353970517653742451011978074893930576631303800
320367108343096737279292961922986751212672768426559125041480745
537349595918795304320650017756944297650514839135909212675679278
268065203061006918276079882798699436138525602103991441803985640
108445365995538743928854242975889677111801200822167214010176841
025967919280593528387375527729346129462119334016134776715537155
Time: CPU 0.45 s, Wall: 0.73 s
```

```
r.determinant?
```


File: /Users/slabbe/Applications/sage-4.6/devel/sage/sage/matrix/matrix_integer_dense.pyx

Type: <type 'builtin_function_or_method'>

Definition: r.determinant(algorithm='default', proof=None, stabilize=2)

Docstring:

Return the determinant of this matrix.

INPUT:

- `algorithm`
  - `'default'` – use 'pari' when number of rows less than 50;
    otherwise, use 'padic'
  - `'padic'` - uses a p-adic / multimodular algorithm that relies on code in IML anc
  - `'linbox'` - calls linbox det (you *must* set proof=False to use this!)
  - `'ntl'` - calls NTL's det function
  - `'pari'` - uses PARI
- `proof` - bool or None; if None use proof.linear_algebra(); only relevant for the padic al

  Note

  It would be *VERY VERY* hard for det to fail even with proof=False.
- `stabilize` - if proof is False, require det to be the same for this many CRT primes in

ALGORITHM: The p-adic algorithm works by first finding a random vector v, then solving A*x divisor of the determinant. Then we compute $\det(A)/d$ using a multimodular algorithm and th

TIMINGS: This is perhaps the fastest implementation of determinants in the world. E.g., for a core2 duo 2.6Ghz running OS X, Sage takes 4.12 seconds, whereas Magma takes 62.87 sec same problem Sage takes 5.73 seconds. For another example, a 200x200 random matrix witl Sage with proof True, 0.11 in Sage with proof False, and 0.21 seconds in Magma with proof T

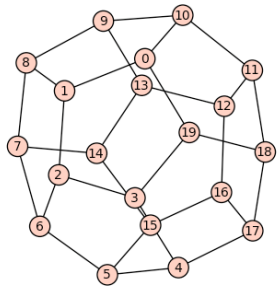
EXAMPLES:

```
sage: A = matrix(ZZ,8,8,[3..66])
sage: A.determinant()
0
```

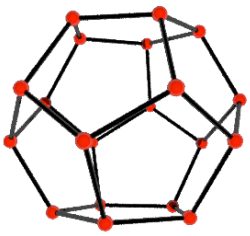
```
sage: A = random_matrix(ZZ,20,20)
sage: D1 = A.determinant()
sage: A._clear_cache()
sage: D2 = A.determinant(algorithm='ntl')
sage: D1 == D2
True
```


Théorie des graphes

```
D = graphs.DodecahedralGraph()
D.show()
```




```
D.show3d(viewer='tachyon')
```



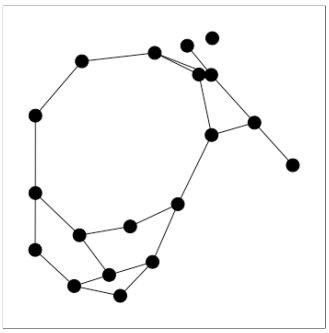
```
D.chromatic_polynomial()
```

```
x^20 - 30*x^19 + 435*x^18 - 4060*x^17 + 27393*x^16 - 142194*x^1
589875*x^14 - 2004600*x^13 + 5673571*x^12 - 13518806*x^11 +
27292965*x^10 - 46805540*x^9 + 68090965*x^8 - 83530946*x^7 +
85371335*x^6 - 71159652*x^5 + 46655060*x^4 - 22594964*x^3 +
7171160*x^2 - 1111968*x
```

```

```

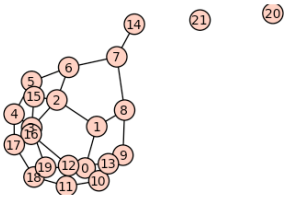
```
graph_editor(D);
```

live:
variable name: D
strength:
length:
help

Save Close

```
D.show()
```



Recherche dans l'encyclopédie de séquences en ligne de Sloane

```
sloane_find([1,5,29,169],1)
```



```
    Searching Sloane's online database...
    []
```

```
sloane_find([1,2,3,4,5,6],1)
```

```
    Searching Sloane's online database...
    []
```

```

```

```

```

```

```

Cython

The Sage notebook allows transparently editing and compiling Cython code simply by typing **%cython** at the top of a cell and evaluate it. Variables and functions defined in a Cython cell are imported into the running session.

Example 1, pure Python

Here is some simple Python code to *numerically* integrate the function $f(x) = x^2$.

```

```

```
from math import sin

def f(x):
    return sin(x**2)

def integrate_f_py(a, b, N):
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

```
timeit('integrate_f_py(0, 1, 1000)', number=50)
```



```
    50 loops, best of 3: 18.5 ms per loop
```

```

```

```

```

Example 1, compiled with Cython (no other changes)

Simply compiling this in Cython gives a speedup.

```
%cython
from math import sin

def f(x):
    return sin(x**2)

def integrate_f_cy0(a, b, N):
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

```
    __Users_sl...0_code_sage32_spyx.c    __Users_sl...ode_sage32_spy
```

```
timeit('integrate_f_cy0(0, 1, 1000)', number=50)
```

```
    50 loops, best of 3: 16.7 ms per loop
```

```

```

```

```

Example 1, typed and compiled with Cython

Adding some static type declarations makes a much greater difference.

```
%cython
from math import sin

def f(double x):
    return sin(x**2)

def integrate_f_cy(double a, double b, int N):
```

```
    cdef int i
    cdef double s, dx
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

__Users_sl...0_code_sage35_spyx.c __Users_sl...ode_sage35_spy

```
timeit('integrate_f_cy(0, 1, 1000)')
```

```
625 loops, best of 3: 489 µs per loop
```

```
18500 /489.0
```

```
37.8323108384458
```

Example 2, pure Python

Here is a Python function that computes the sum of the first n positive integers.

```
def mysum_py(n):
    s = 0
    for k in range(n):
        s += k
    return s
```

```
time mysum_py(10^6)
```

```
499999500000
Time: CPU 2.09 s, Wall: 2.16 s
```

Example 2, just compiled with Cython

Simply compiling this function with Cython provides a speedup.

```
%cython
def mysum_cy0(n):
    s = 0
    for k in range(n):
```



```
        s += k
    return s
```

__Users_sl...0_code_sage41_spyx.c __Users_sl...ode_sage41_spy

```
time mysum_cy0(10^6)
```

```
499999500000L
Time: CPU 0.25 s, Wall: 0.27 s
```

```
2.09 / 0.25
```

```
8.36000000000000
```

Example 2, typed and compiled with Cython

Adding some static type declarations makes a much greater difference.

```
%cython
def mysum_cy1(n):
    cdef int k
    cdef long long s

    s = 0
    for k in range(n):
        s += k
    return s
```

__Users_sl...0_code_sage45_spyx.c __Users_sl...ode_sage45_spy

```
time mysum_cy1(10^6)
```

```
499999500000L
Time: CPU 0.00 s, Wall: 0.00 s
```

```
2.09 / 0.00
```

```
+infinity
```

```
timeit('mysum_cy1(10^6)')
```

```
125 loops, best of 3: 1.57 ms per loop
```

```
2.09/0.00157
```

```
1331.21019108280
```