minicurso-ecmat

October 21, 2021

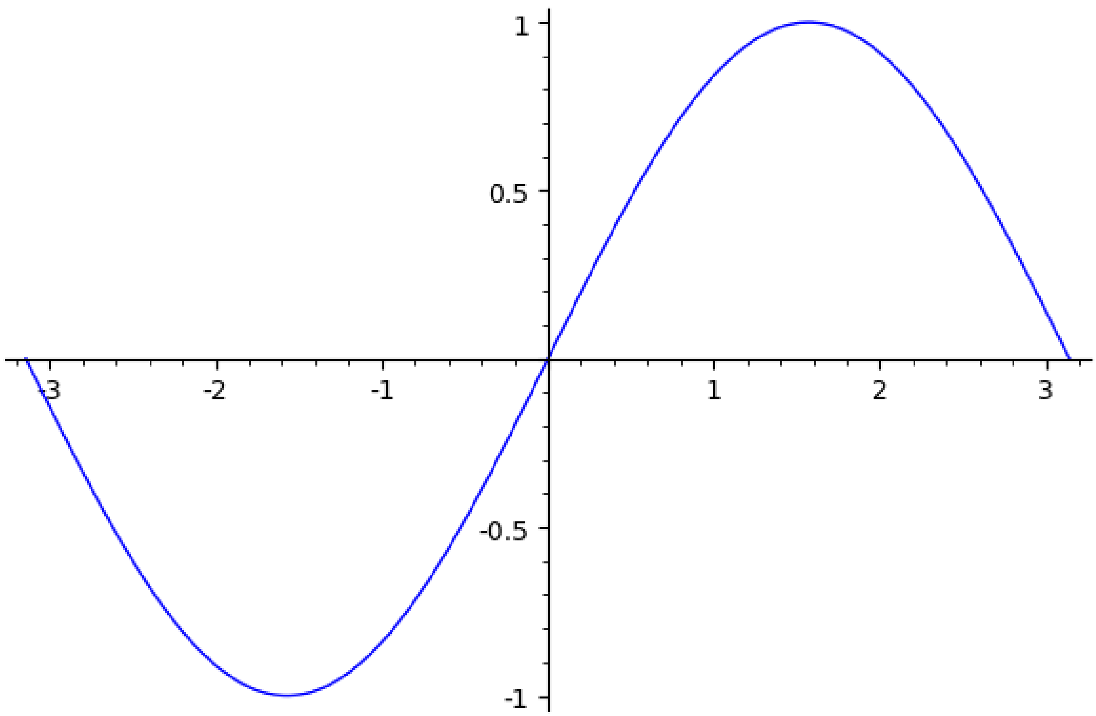
1 Aritmética com Sagemath

1.1 Exemplos Iniciais

1.1.1 Gráficos

[1]:
```
plot(sin(x),-pi,pi)
```

[1]:



[2]:
```
@interact
def _(n=slider([1..10])):
    show(plot(x^n, -1,1))
```

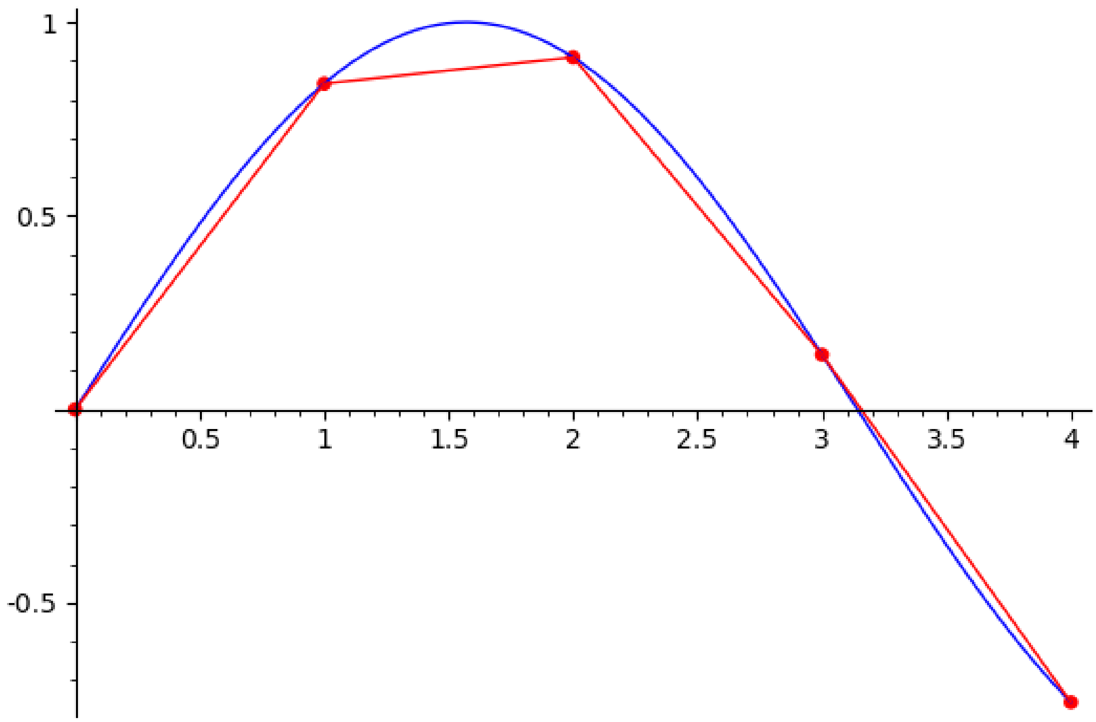
```
Widget Javascript not detected.  It may not be installed or enabled properly.
```

[3]:
```
var('y')
plot3d(x^2 - y^2, (x,-1,1),(y,-1,1))
```

[3]: Graphics3d Object

[4]:
```
P1 = plot(sum([point((n,sin(n)),color='red',size=30) for n in [0..4]]))
P2 = plot(sin(x),0,4)
P3 = line([(n,sin(n)) for n in [0..4]],color='red')
P1+P2+P3
```

[4]:



1.1.2 Cálculo Diferencial e Integral

[37]:
```
f(x) = (3*x^2 - x + 2)/(5*x - 4)
show(lim(f,x=1))
```

```
x |--> 4
```

[38]:
```
show(f.derivative())
```

```
x |--> (6*x - 1)/(5*x - 4) - 5*(3*x^2 - x + 2)/(5*x - 4)^2
```

[7]:
```
show(f.integral(x))
```

```
x |--> 3/10*x^2 + 7/25*x + 78/125*log(5*x - 4)
```

1.1.3 Matrizes e Álgebra linear

[8]:
```
A = matrix([[1,2,4],
            [2,3,1],
            [3,1,5]])
show(A)
```

```
[1 2 4]
[2 3 1]
[3 1 5]
```

[9]:
```
show(det(A))
```

```
-28
```

[10]:
```
show("A^(-1) = ", A.inverse(), " ; A na forma escalonada: ", A.echelon_form())
```

```
'A^(-1) = ' [ -1/2  3/14  5/14]
[  1/4   1/4  -1/4]
[  1/4 -5/28  1/28] ' ; A na forma escalonada: ' [ 1  0 18]
[ 0  1  7]
[ 0  0 28]
```

[11]:
```
show("Polinômio Car.: ", A.charpoly())
show("Polinômio Min.: ", A.minpoly())
```

```
'Polinômio Car.: ' x^3 - 9*x^2 + 6*x + 28
```

```
'Polinômio Min.: ' x^3 - 9*x^2 + 6*x + 28
```

[12]:
```
A.eigenvalues()
```

[12]:
```
[-1.378695206755170?, 2.616358832559789?, 7.762336374195381?]
```

[13]:
```
show(A.LU())
```

```
(
[0 0 1]  [  1   0   0]  [   3    1    5]
[0 1 0]  [2/3   1   0]  [   0  7/3 -7/3]
[1 0 0], [1/3 5/7   1], [   0    0    4]
)
```

1.1.4 Álgebra - Cálculo Simbólico

[14]:
```
var('a,b')
show(expand((a+b)^5))
```

```
a^5 + 5*a^4*b + 10*a^3*b^2 + 10*a^2*b^3 + 5*a*b^4 + b^5
```

[15]:
```
show(factor(a^6 - b^6))
```

```
(a^2 + a*b + b^2)*(a^2 - a*b + b^2)*(a + b)*(a - b)
```

[16]:
```
show(solve(a^2 + a - 1,a))
```

```
[a == -1/2*sqrt(5) - 1/2, a == 1/2*sqrt(5) - 1/2]
```

[17]:
```
# interseção do círculo unitário com a parábola y = x^2
var('x,y')
sol = solve([
    x^2 + y^2 == 1,
    y == x^2,
],x,y)
show(sol)
```

```
[[x == -sqrt(1/2*sqrt(5) - 1/2), y == 1/2*sqrt(5) - 1/2], [x == sqrt(1/2*sqrt(5) - 1/2), y == 
```

[18]:
```
f(x) = a*x^2 + b*1/x^2
show(f(x))
show(f(f(f(x))))
show(f(x).subs(a=3,b=2))
```

```
a*x^2 + b/x^2
```

```
((a*x^2 + b/x^2)^2*a + b/(a*x^2 + b/x^2)^2)^2*a + b/((a*x^2 + b/x^2)^2*a + b/(a*x^2 + b/x^2)^2
```

```
3*x^2 + 2/x^2
```

1.1.5 Álgebra Abstrata

[19]:
```
G = SymmetricGroup(3)
G.is_abelian()
```

[19]:
```
False
```

[20]:
```
G.multiplication_table(names='elements')
```

[20]:
```
      *       ()   (2,3)   (1,2) (1,2,3) (1,3,2)   (1,3)
       +------------------------------------------------
     ()|      ()   (2,3)   (1,2) (1,2,3) (1,3,2)   (1,3)
  (2,3)|   (2,3)      () (1,2,3)   (1,2)   (1,3) (1,3,2)
  (1,2)|   (1,2) (1,3,2)      ()   (1,3)   (2,3) (1,2,3)
(1,2,3)| (1,2,3)   (1,3)   (2,3) (1,3,2)      ()   (1,2)
(1,3,2)| (1,3,2)   (1,2)   (1,3)      () (1,2,3)   (2,3)
  (1,3)|   (1,3) (1,2,3) (1,3,2)   (2,3)   (1,2)      ()
```

[21]:
```
g = G("(1,2,3)")
g.inverse()
```

[21]: (1,3,2)

1.2 Conceitos básicos

1.2.1 Tipos de Dados | Conjuntos numéricos

[22]:
```
parent(2)
```

[22]: Integer Ring

[23]:
```
parent(2/1)
```

[23]: Rational Field

[24]:
```
parent(2.0)
```

[24]: Real Field with 53 bits of precision

[25]:
```
parent(2+I)
# parent(CC(2+I))
```

[25]: Symbolic Ring

1.2.2 Operações

[26]:
```
2 + 2
```

[26]: 4

[27]:
```
3 * 2
```

[27]: 6

[28]:
```
3^2
```

[28]: 9

Algumas operações são feitas simbolicamente. Para exibir uma aproximação usamos N (aproximação Numérica)

[29]:
```
show((2 + sqrt(3))*(2-sqrt(5)))
```

```
-(sqrt(5) - 2)*(sqrt(3) + 2)
```

[30]:
```
show(N((2 + sqrt(3))*(2-sqrt(5))))
```

```
-0.881017686069242
```

Cuidado! O resultado da operação depende dos tipos de objetos!

[31]:
```
parent(2+2.0)
```

[31]: Real Field with 53 bits of precision

A divisão de inteiros é racional!

[32]:
```
parent(3/7)
```

[32]: Rational Field

[33]:
```
1/5 + 3/7
```

[33]: 22/35

É possível converter números de tipo para outro… quando faz sentido. * ZZ Inteiros * QQ Racionais * RR Reais * CC Complexos

[34]:
```
RR(2)
```

[34]: 2.00000000000000

[35]:
```
QQ(1.5123)
```

[35]: 15123/10000

[ ]:
```
ZZ(pi)
```

1.2.3 Atribuição e igualdade

O símbolo = tem um sentido diferente do que estamos habituados em matemática: serve como atribuição, veja

[39]:
```
m = 3
show(m^2)
```

9

As vezes usamos $\leftarrow$ para indicar uma atribuição, a primeira linha do código acima poderia ser denotada por $m \leftarrow 3$, se lendo: m **recebe o valor** 3

Se desejamos verificar **se** uma igualdade é verdadeira usamos o operador ==

[40]:
```
2 + 2 == 5
```

[40]: False

[42]:
```
2 == 3 - 1
```

[42]: True

Outras operações booleanas comuns:

[43]:
```
3 > 2
```

[43]: True

[44]:
```
3 >= 2
```

[44]: True

[45]:
```
4 != 3 # != : diferente de
```

[45]: True

1.2.4 Funções comuns

[46]:
```
sqrt(4)
```

[46]: 2

[47]:
```
sqrt(5)
```

[47]: sqrt(5)

[48]:
```
log(1)
```

[48]: 0

[49]:
```
conjugate(1+i)
```

[49]: -I + 1

[50]:
```
cos(pi/3)
```

[50]: 1/2

1.2.5 Listas

[51]:
```
[x^2 for x in [1..5]]
```

[51]: [1, 4, 9, 16, 25]

[52]:
```
quadrados = [x^2 for x in [1..10]]
show(quadrados)
```

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

[53]:
```
quadrados[3]
```

[53]: 16

[54]:
```
quadrados[4:8]
```

[54]: [25, 36, 49, 64]

[55]:
```
# Adicionando elementos (no final da lista)
quadrados.append(11^2)
show(quadrados)
```

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]

[56]:
```
# Verificando se um objeto está na lista
81 in quadrados
```

[56]: True

1.3 Divisão Euclidiana

[57]:
```
show(10.quo_rem(3))
show((10 // 3, 10 % 3))
```

```
(3, 1)
```

```
(3, 1)
```

Podemos guardar o quociente e o resto da seguinte forma:

[58]:
```
(q,r) = 10.quo_rem(3)
10 == 3 * q + r
```

[58]:
```
True
```

Atenção: Se o b for negativo o resto será negativo! A identidade $a = bq + r$ ainda será válida mas r não estará no intervalo $\{0, 1, \ldots, b - 1\}$.

[59]:
```
show(10.quo_rem(-3))
```

```
(-4, -2)
```

A função divmod também pode ser utilizada.

[60]:
```
divmod(10,3)
```

[60]:
```
(3, 1)
```

1.3.1 Divisores

[61]:
```
a = 5
b = 3
b.divides(a)
```

[61]:
```
False
```

Criando uma lista com divisores de um dado a.

[62]:
```
a = 32
divisores = []
for b in [1..a]:
    if b.divides(a):
        divisores.append(b)
show(divisores)
```

```
[1, 2, 4, 8, 16, 32]
```

Matemáticamente estamos criando o conjunto $\{b \ : \ b \in \{1, \ldots, n\}$ se $b \mid a\}$

[63]:
```
a = 32
[b for b in [1..a] if b.divides(a)]
```

[63]: [1, 2, 4, 8, 16, 32]

Obviamente o sage possui uma função que faz isso automaticamente: a função divisors

[64]:
```
divisors(32)
```

[64]: [1, 2, 4, 8, 16, 32]

Desafio: Encontre mais um número perfeito.

[65]:
```
#
```

Desafio Extra: Um número é dito *abundante* se a soma de seus divisores próprios é maior que si mesmo e é dito *deficiente* se a soma de seus divisores próprios é menor que si mesmo. Crie um código que receba um número n e decida se ele é deficiente, perfeito ou abundante.

1.4 Primos

Algoritmo para decidir se um dado $n > 2$ é primo: * resultado $\leftarrow$ ``Primo'' * Para $i = 2, \ldots, n-1$:. * * Teste se $i \mid n$ * * Caso positivo, resultado $\leftarrow$ ``Composto'' * Exiba resultado

[66]:
```
n = 2^10+1
resultado = "Primo"
for i in [2..n-1]:
    if i.divides(n):
        resultado = "Composto"
print(resultado)
```

Composto

Formas alternativas equivalentes:

[67]:
```
n = 31
show(divisors(n))
show(len(divisors(31)))
```

[1, 31]

2

[68]:
```
n = 13
all([not i.divides(n) for i in [2..n-1]])
```

[68]: True

[69]:
```
31.is_prime()
```

[69]: True

1.4.1 Outras funções úteis sobre primos

[70]:
```
# Primos menores que um $N$
list(primes(10))
```

[70]: [2, 3, 5, 7]

[71]:
```
# Primeiros $N$ primos
primes_first_n(10)
```

[71]: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

[72]:
```
# Primos em um intervalo
prime_range(50,100)
```

[72]: [53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

[73]:
```
# Divisores primos (como divisors mas só primos)
show(divisors(30))
show(prime_divisors(30))
```

[1, 2, 3, 5, 6, 10, 15, 30]

[2, 3, 5]

A função de contagem de primos é definida para um número real positivo x por

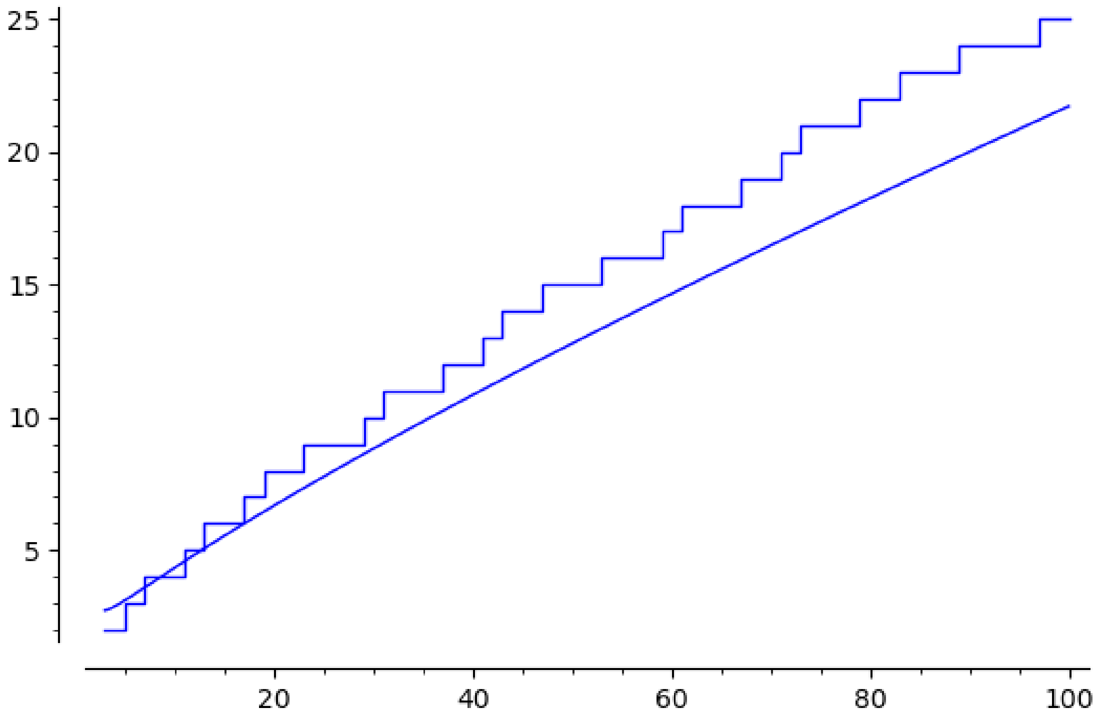
$$\pi(x) = \#\{n \in \mathbb{N} \mid n \leq x \text{ e } n \text{ é primo}\}$$

A função π é bastante importante na Teoria Analítica dos Números. Um famoso teorema diz que $pi(x) \sim n/\log(n)$.

[74]:
```
show(prime_pi(10))
P1 = plot(prime_pi,3,100)
P2 = plot(lambda x: x/log(x), 3, 100)
P1+P2
```

4

[74]:



Desafio Extra: Pesquise sobre o Teorema dos Números primos e exiba dados numéricos que indiquem sua validade.

1.5 MDC

Denote por $D(n)$ o conjunto dos divisores positivos de n.

$$\mathrm{mdc}(a,b) = \max\{k \in D(a) \mid k \in D(b)\}$$

[75]:
```
a = 1001
b = 109
max([k for k in divisors(a) if k in divisors(b)])

# Da = set(divisors(a))
# Db = set(divisors(b))
# show(Da.intersection(Db))
```

[75]: 1

Algoritmo de Euclides Dados $a, b \in \mathbb{N}$ * Enquanto $b \neq 0$ repita os passos a seguir: * * $(q, r) \leftarrow$ Quociente e resto da divisão de a por b * * $a \leftarrow b$ * * $b \leftarrow r$ * Resultado: a

[76]:
```
a = 1001
b = 109
while b != 0:
    print(a, '\t', b)
    (q,r) = a.quo_rem(b)
    (a,b) = (b,r)
```

```
1001     109
109      20
20       9
9        2
2        1
```

Agora, com atribuições simultâneas:

[77]:
```
a = 12313810
b = 5684196841680
while b != 0:
    (a,b) = (b, a.quo_rem(b)[1])
show(a)
```

```
10
```

No sage, o mdc pode ser calculado usando a função gcd

[78]:
```
gcd(a,b)
```

[78]: 10

Algoritmo de Euclides estendido

[79]:
```
a = 12
b = 5
M = matrix(ZZ, [
    [1,0],
    [0,1],
    [a,b]
])
show(M)
```

```
[ 1  0]
[ 0  1]
[12  5]
```

A função piso $\lfloor\cdot\rfloor$ é denotada por floor no sage. Calculemos $q = \lfloor m_{31}/m_{32} \rfloor$ **lembrando que os índices começam do** 0.

[80]:
```
while M[2,1] != 0:
    q = floor(M[2,0]/M[2,1])
    M = M*matrix([[0,1],[1,-q]])
```

[81]:
```
show(M)
show(M[0,0]*a+M[1,0]*b == gcd(a,b))
```

```
[ -2   5]
[  5 -12]
[  1   0]
```

```
True
```

No sage a função xgcd retorna o mdc e a solução (x, y) para a equação no Teorema de Bézout.

[82]:
```
xgcd(a,b)
```

[82]:
```
(1, -2, 5)
```

1.6 Fatoração

A partir do menor primo $p = 2$, testamos se $p \mid n$, caso positivo adicionamos tal p na fatoração de n e substituimos n por $\frac{n}{p}$. Caso contrário passamos para o próximo primo. o algoritmo pára quando $n/p = 1$.

[83]:
```
n = 360
p = 2
while n != 1:
    if p.divides(n):
        print(n,'\t|\t',p)
        n = n/p
    else:
        p = p.next_prime()
```

```
360     |       2
180     |       2
90      |       2
45      |       3
15      |       3
5       |       5
```

Vamos alterar o código acima para guardar os primos na fatoração de n em uma lista.

[84]:
```
n = 360
p = 2
fatoracao = []
```

```
while n != 1:
    if p.divides(n):
        fatoracao.append(p)
        n = n/p
    else:
        p = p.next_prime()
show(fatoracao)
show(prod(fatoracao))
```

[2, 2, 2, 3, 3, 5]

360

[85]:
```
factor(360)
```

[85]: 2^3 * 3^2 * 5

A função factor retorna um objeto especifico da classe Factorization. Ela pode ser transformada em uma lista usando a função list, resultando uma lista com pares (p, e) onde p é um fator primo e e é o expoente com que p aparece na fatoração do número em questão.

[86]:
```
list(factor(360))
```

[86]: [(2, 3), (3, 2), (5, 1)]

Extra: factor funciona em outros objetos

[87]:
```
factor(x^2 - 1)
```

[87]: (x + 1)*(x - 1)

2 Congruências

[88]:
```
3 % 2
```

[88]: 1

[89]:
```
mod(3,2)
```

[89]: 1

[90]:
```
parent(3 % 2)
```

[90]: Integer Ring

[91]:
```
parent(mod(3,2))
```

[91]: Ring of integers modulo 2

[92]:
```
a =  3001 % 13
show(a)
```

11

[ ]:
```
a^8418418418413814
```

[94]:
```
a = mod(3001,13)
show(a)
```

11

[95]:
```
a^8418418418413814
```

[95]: 4

[96]:
```
Z13 = Zmod(13)
Z13
```

[96]: Ring of integers modulo 13

[97]:
```
show(list(Z13))
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

Listamos os possíveis quadrados em $\mathbb{Z}_{13}$

[98]:
```
pretty_print("n \t n^2 mod 13")
for n in Z13:
    print("{} \t {}".format(n,n^2))
#     pretty_print(html(str(n)+"$^2\equiv " + str(n^2) + "\pmod{13}$"))
```

'n \t n^2 mod 13'

```
0        0
1        1
2        4
3        9
4        3
5        12
6        10
7        10
```

```
8         12
9         3
10        9
11        4
12        1
```

[99]:
```python
Z13.addition_table(names='elements')
```

[99]:
```
 +   0  1  2  3  4  5  6  7  8  9 10 11 12
  +---------------------------------------
 0|  0  1  2  3  4  5  6  7  8  9 10 11 12
 1|  1  2  3  4  5  6  7  8  9 10 11 12  0
 2|  2  3  4  5  6  7  8  9 10 11 12  0  1
 3|  3  4  5  6  7  8  9 10 11 12  0  1  2
 4|  4  5  6  7  8  9 10 11 12  0  1  2  3
 5|  5  6  7  8  9 10 11 12  0  1  2  3  4
 6|  6  7  8  9 10 11 12  0  1  2  3  4  5
 7|  7  8  9 10 11 12  0  1  2  3  4  5  6
 8|  8  9 10 11 12  0  1  2  3  4  5  6  7
 9|  9 10 11 12  0  1  2  3  4  5  6  7  8
10| 10 11 12  0  1  2  3  4  5  6  7  8  9
11| 11 12  0  1  2  3  4  5  6  7  8  9 10
12| 12  0  1  2  3  4  5  6  7  8  9 10 11
```

[100]:
```python
Z13.multiplication_table(names='elements')
```

[100]:
```
 *   0  1  2  3  4  5  6  7  8  9 10 11 12
  +---------------------------------------
 0|  0  0  0  0  0  0  0  0  0  0  0  0  0
 1|  0  1  2  3  4  5  6  7  8  9 10 11 12
 2|  0  2  4  6  8 10 12  1  3  5  7  9 11
 3|  0  3  6  9 12  2  5  8 11  1  4  7 10
 4|  0  4  8 12  3  7 11  2  6 10  1  5  9
 5|  0  5 10  2  7 12  4  9  1  6 11  3  8
 6|  0  6 12  5 11  4 10  3  9  2  8  1  7
 7|  0  7  1  8  2  9  3 10  4 11  5 12  6
 8|  0  8  3 11  6  1  9  4 12  7  2 10  5
 9|  0  9  5  1 10  6  2 11  7  3 12  8  4
10|  0 10  7  4  1 11  8  5  2 12  9  6  3
11|  0 11  9  7  5  3  1 12 10  8  6  4  2
12|  0 12 11 10  9  8  7  6  5  4  3  2  1
```

[101]:
```python
Z13.is_field()
```

[101]: True

Desafio Extra: Que teorema é esse?

[102]:
```
prod(list(Zmod(13))[1:])
```

[102]:
```
12
```

Quando existe, podemos encontrar a inversa modular da forma natural ^-1.

[103]:
```
a = Z13(3)
a^-1
```

[103]:
```
9
```

[ ]:
```
Z12 = Zmod(12)
a = Z12(6)
a^-1
```

Podemos listar os elementos invertíveis…

[105]:
```
Z12.list_of_elements_of_multiplicative_group()
```

[105]:
```
[1, 5, 7, 11]
```

…mas é retornada uma lista de inteiros. O grupo *abstrato* dos invertíveis pode ser obtido da seguinte forma:

[106]:
```
G = Z12.unit_group()
print(G)
print(G.cayley_table())
```

```
Multiplicative Abelian group isomorphic to C2 x C2
*  a b c d
 +--------
a| a b c d
b| b a d c
c| c d a b
d| d c b a
```

Voltando aos invertíveis módulo n.

[107]:
```
print("Invertíveis módulo 13")
show(Z13.list_of_elements_of_multiplicative_group())
print("Invertíveis módulo 12")
show(Z12.list_of_elements_of_multiplicative_group())
```

```
Invertíveis módulo 13
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Invertíveis módulo 12

[1, 5, 7, 11]

Afirmação. Os invertíveis módulo m são exatamente os coprimos módulo m. A quantidade de invertíveis é, portanto, o número de coprimos com m em $\{0, 1, \ldots, m-1\}$, ou seja, a quantidade de elementos no conjunto

$$\{n \mid 0 \leq n \leq m-1 \text{ e } \mathrm{mdc}(n, m) = 1\}$$

[108]:
```
m = 15
len([n for n in [0..m-1] if gcd(n,m) == 1])
```

[108]: 8

Denotamos esse número por $\varphi(m)$. φ define uma função aritmética $\mathbb{N} \to \mathbb{N}$ chamada *função totiente de Euler*. Essa função está implementada no sagemath com o nome euler_phi. Calculemos φ para outros valores.

[109]:
```
euler_phi(13)
```

[109]: 12

Observação importante *(para mais tarde)*: A função φ é uma função multiplicativa. Isso implica que $\varphi(n)$ pode ser calculada facilmente se soubermos a fatoração em primos de n. *Em particular*, $\varphi(n) = n - 1$ para n primo e, se $n = pq$, com $p \neq q$ primos **muito** grandes, pode-se calcular facilmente

$$\varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$$

2.0.1 Teorema Chinês dos restos

Resolvemos o sistema

$$\begin{cases} x \equiv 7 \pmod{12} \\ x \equiv 9 \pmod{13} \end{cases}$$

[110]:
```
# x = ?
```

[111]:
```
crt(7,9,12,13)
```

[111]: 139