```python
import sympy
from sympy.abc import x, y
from sympy import solve,expand,symbols
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.scale as mscale
from matplotlib.ticker import FixedLocator, NullFormatter

class Polynome:
    def __init__(self, liste_coefs=[0]):
        self.coefs = liste_coefs
        self.reduire()

    def __str__(self):
        s=""
        if (len(self.coefs)-1==0):
            return str(self.coefs[0])
        else:
            if self.coefs[0]!=0:
                s=str(self.coefs[0]) + " + "
            for i in range(1, len(self.coefs)):
                if self.coefs[i]!=0:
                    if self.coefs[i]!=1:
                        s+="{}.X^{} + ".format(self.coefs[i],i)
                    else: s+="X^{} + ".format(i)
            s = s[:-3].replace("+ -", "- ").replace("X^1 ","X ").replace("
1.X"," X")
            if s[-2:]=="^1": s = s[:-2]
            if s[:3]=="1.X": s = s[3:]
            return s

    def degre(self):
        return (len(self.coefs)-1)

    def __add__(self, other):
        if len(other.coefs) >len(self.coefs):
            liste_coefs = other.coefs[:]; n = len(self.coefs)
        else: liste_coefs = self.coefs[:]; n = len(other.coefs)
        for i in range(n):
            liste_coefs[i] = self.coefs[i] + other.coefs[i]
        return Polynome(liste_coefs)

    def __sub__(self, other):
        if len(other.coefs) >len(self.coefs):
            liste_coefs = other.coefs[:]; n = len(self.coefs)
        else: liste_coefs = self.coefs[:]; n = len(other.coefs)
        for i in range(n):
            liste_coefs[i] = self.coefs[i] - other.coefs[i]
        return Polynome(liste_coefs)

    def reduire(self):
        while round(self.coefs[-1],12) == 0 and len(self.coefs)>1:
            self.coefs.pop()
        for i in range(len(self.coefs)):
            self.coefs[i] = round(self.coefs[i],12)

    def __mul__(self, other):
        if isinstance(other,tuple):
            coef = other[0]; degre = other[1]
            liste_coefs = [0]*degre + self.coefs
            for i in range(degre, len(liste_coefs)):
                liste_coefs[i] = liste_coefs[i]*coef
        else:
            liste_coefs=[0]*(len(self.coefs)+len(other.coefs)-1)
            for i1 in range(len(self.coefs)):
                for i2 in range(len(other.coefs)):
                    liste_coefs[i1+i2] = liste_coefs[i1+i2] +
self.coefs[i1]*other.coefs[i2]
        poly = Polynome(liste_coefs)
        return poly

    def __pow__(self, n):
        poly = Polynome([1])
        for i in range(n):
            poly = poly*self
        return poly

    def __truediv__(self, other):
        if self.degre()<other.degre():
            return (Polynome([0]), self)
        r = Polynome(self.coefs)
        degre = r.degre() - other.degre()
        q = Polynome([0]*degre + [1])
        while (degre>=0) and (r.coefs[-1]!=0):
            coef = r.coefs[-1] / other.coefs[-1]
            q.coefs[degre] = coef
            p = other * (coef, degre)
            r = r - p
            degre = r.degre() - other.degre()
        return (q, r)

    def __eq__(poly1, other):
        return (poly1.coefs==other.coefs)

    def eval(self,x):
        valeur_polynome = self.coefs[0]
        power=1
        for coef in self.coefs[1:]:
            power = power*x
            valeur_polynome += coef*power
        return valeur_polynome

    def eval_horner(self,x):
        valeur_polynome = self.coefs[-1]
        for coef in reversed(self.coefs[:-1]):
            valeur_polynome = valeur_polynome*x + coef
        return valeur_polynome

def pgcdpoly(A, B):
    if (B == Polynome([0])):
        coef = 1/A.coefs[-1]
        A = A*(coef,0)
        return A
    else:
        Q, R = A / B
        return pgcdpoly(B, R)

def fctp(x): return((x-3)*(x-5)*(x-7)*(x-11)*(x-13)*(x-17)*(x-19))

def fctq(x): return((21-x)*(19-x)*(17-x)*(13-x)*(11-x)*(7-x)*(5-x))

def fctr(x): return(x**6-72*x**5+2085*x**4-30960*x**3+247539*x**2-
1005768*x+1616615)

fig, ax = plt.subplots(figsize=(10,5))
plt.yscale('symlog')
n = 24
x = symbols('x')
```

```
p = 1
q = 1
for k in [3,5,7,11,13,17,19]:
    p = p*(x-k)
    q = q*((n-x)-k)
print('n = ',n)
print('p = ',p,' \n qui se developpe en ',expand(p))
print('q = ',q,' \n qui se developpe en ',expand(q))
sp = solve(p, x, dict=True)
sq = solve(q, x, dict=True)
print('solution eq. gauche = ',sp)
print('solution eq. droite = ',sq)
a = np.linspace(0,n,10*n)
b = fctp(a)
c = fctq(a)
ax.plot(a,b,color='red')
ax.plot(a,c,color='blue')
ppourpgcd = Polynome([-4849845,4633919,-1748385,340419,-37215,2301,-75,1])
qpourpgcd = Polynome([33948915,-22737743,6204087,-897699,74745,-3597,93,-1])
r = pgcdpoly(ppourpgcd,qpourpgcd)
print('Le pgcd des deux polynomes est :',str(r))
rd = x**6-72*x**5+2085*x**4-30960*x**3+247539*x**2-1005768*x+1616615
print('rd = ',rd,' \n qui se developpe en ',expand(rd))
sr = solve(rd, x, dict=True)
print('solution eq. pgcd = ',sr)
print('valeur de r en 4 = ',int(rd.evalf(subs={x:4})))
print('valeur de r en n-4 = ',int(rd.evalf(subs={x:n-4})))
print('valeur de r en n/2 = ',int(rd.evalf(subs={x:n/2})))
d = fctr(a)
ax.plot(a,d,color='green')
plt.show()
```