ellcurves

```
import time
```

```
def digit_to_char(digit):
    if (digit < 10):
        return str(digit)
    if (digit > 35):
        raise ValueError("Digit must not be greater than 35.")
    return chr(digit - 10 + ord('A'))
```

```
def char_to_digit(char):
    if ( (char < '0') or (char > 'Z') or ((char > '9') and (char <
'A')) ):
        raise ValueError("Char must be a digit from 0 to 9 or letter
from A to Z.")
    if (char <= '9'):
        return int(char)
    return int(ord(char) - ord('A') + 10)
```

```
def base_representation(x, w = 1, base = 2):
    signed = False
    b = base ** w
    if (x < 0):
        signed = True
        x = abs(x)
    res = x // b
    ret = str(digit_to_char(x % b))
    while (res != 0):
        ret = str(digit_to_char(res % b)) + ret
        res = res // b
    if (signed):
        ret = "-" + ret
    return ret
```

```
def LSB(x, w = 1, base = 2, bits = 1):
    return (x % (base ** (w*bits)))
```

```
def LSB_to_string(x, w = 1, base = 2, bits = 1):
    ret = base_representation(LSB(x, w, base, bits), w, base)
    if (len(ret) < bits):
        ret = "0"*(bits - len(ret)) + ret
    return ret
```

```
def get_bit(x, num, w = 1, base = 2):
    x //= base ** (w*num)
    return x % (base**w)
```

```
def wNAF_openssl(x, w = 1, base = 2):
    if (x == 0):
        return "0"

    sign = 1
    if (x < 0):
        sign = -1
        x = abs(x)

    bit = base**w
    next_bit = bit*base
    mask = next_bit - 1
    ret = ""
    i = 0
    len = x.nbits()

    window_val = LSB(x, w + 1, base)
    while (window_val != 0) or (i + w + 1 < len):
        digit = 0
        if (LSB(window_val) == 1):
            if (window_val >= bit):
                digit = window_val - next_bit
            else:
                digit = window_val
            window_val -= digit

        ret = base_representation(sign*digit, w, base) + ret
        i += 1
        window_val //= base
        window_val += get_bit(x, w + i) * bit
    return ret
```

```
def wMNAF_openssl(x, w = 1, base = 2):
    if (x == 0):
        return "0"

    sign = 1
    if (x < 0):
        sign = -1
        x = abs(x)

    bit = base**w
```

```
    next_bit = bit*base
    mask = next_bit - 1
    ret = ""
    i = 0
    len = x.nbits()

    window_val = LSB(x, w + 1, base)
    while (window_val != 0) or (i + w + 1 < len):
        digit = 0
        if (LSB(window_val) == 1):
            if (window_val >= bit):
                digit = window_val - next_bit
                if (i + w + 1>= len):
                    digit = window_val & (mask // base)
            else:
                digit = window_val
            window_val -= digit

        ret = base_representation(sign*digit, w, base) + ret
        i += 1
        window_val //= base
        window_val += get_bit(x, w + i) * bit
    return ret
```

```
def wNAF(x, w = 1, base = 2):
    if (x == 0):
        return "0"

    sign = 1
    if (x < 0):
        sign = -1
        x = abs(x)

    ret = []

    while (x > 0):
        if (LSB(x, 1, base)):
            digit = LSB(x, w + 1, base)
            if (digit >= base**w):
                digit -= base**(w + 1)
            x -= digit
        else:
            digit = 0
        ret = [sign*digit] + ret
        x //= base

    return ret
```

```
def wMNAF(x, w = 1, base = 2):
    if (x == 0):
        return "0"

    sign = 1
    if (x < 0):
        sign = -1
        x = abs(x)

    mask = base**w - 1
    ret = []
    i = 0
    len = x.nbits()

    while (x > 0) or (i + w + 1 < len):
        if (LSB(x, 1, base)):
            digit = LSB(x, w + 1, base)
            if (digit >= base**w):
                digit -= base**(w + 1)
                if (i + w + 1 >= len):
                    digit &= mask
            x -= digit
        else:
            digit = 0
        ret = [sign*digit] + ret
        x //= base
        i += 1

    return ret
```

```
def print_table_of_values(w = 1):
    print("{0:>5}\t{1:>8}\t{2:>25}\t{3:>25}".format("e", "binary",
"NAF", "MNAF"))
    for k in range(1, 40):
        k = Integer(k)
        print("{0:>5}\t{1:>8}\t{2:>25}\t{3:>25}".format(k,
base_representation(k, w), wNAF(k, w), wMNAF(k, w)))
```

```
def precompute_values(P, w = 1, base = 2):
    G = P
    pos = [G]
    neg = [-G]
    for i in range(1, base**(w) - 1, base):
        G += 2*P
        pos.append(G)
        neg.append(-G)
    return pos,neg
```

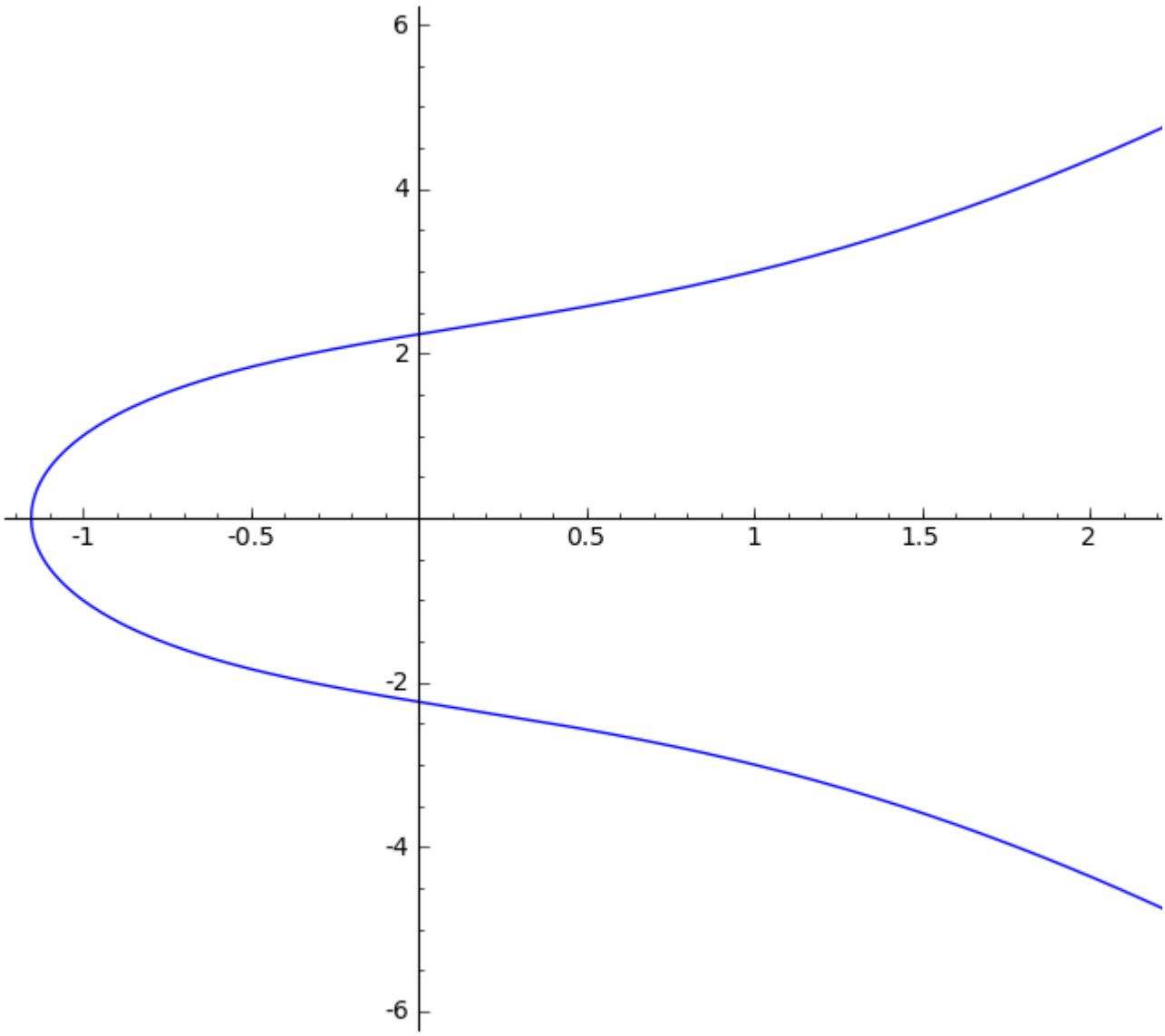
```
def scalar_multiply(n, P, w = 1, base = 2):
    pos, neg = precompute_values(P, w, base)
    wmnaf = wMNAF(n, w, base)
    Q = 0
    for w in wmnaf:
        Q = 2*Q
        if (w != 0):
            if (w > 0):
                Q += pos[(w - 1) // 2]
            else:
                Q += neg[(-w - 1) // 2]
    return Q
```

```
def generate_data(P, w, min_value = 10, max_value = 1024, step = 11,
printtimings = False):
    if (printtimings):
        print("{0:>10}\t\t{1:>25}\t\t{2:>25}\t\t{3:>5}".format("n",
"wMNAF time", "SAGE time", "Check"))

    graph = []
    for k in [1, 2, 3, 5, 7] + range(min_value, max_value, step):
        n = Integer(k)

        t0 = time.clock()
        wnaf = scalar_multiply(n, P, w)
        wnaf_time = time.clock() - t0

        t0 = time.clock()
        default = n*P
        sage_time = time.clock() - t0

        if (sage_time != 0) and (wnaf_time != 0):
            percentage = 100*wnaf_time/sage_time
            graph.append((k, percentage))

        if (printtimings):
            print("{0:>10}\t\t{1:>25}\t\t{2:>25}
\t\t{3:>5}".format(k, wnaf_time, sage_time, wnaf == default))

    return graph
```

```
#secp128r1
#p = int("FFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFF", 16)
#F = ZZ.quotient_ring(p*ZZ)
#E = EllipticCurve([int("FFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFC", 16),
int("E87579C11079F43DD824993C2CEE5ED3", 16)])
```

```
E = EllipticCurve([3,5])
w = 4
```

```
E.plot()
```




```
P = E.rational_points(bound = 15); P
```

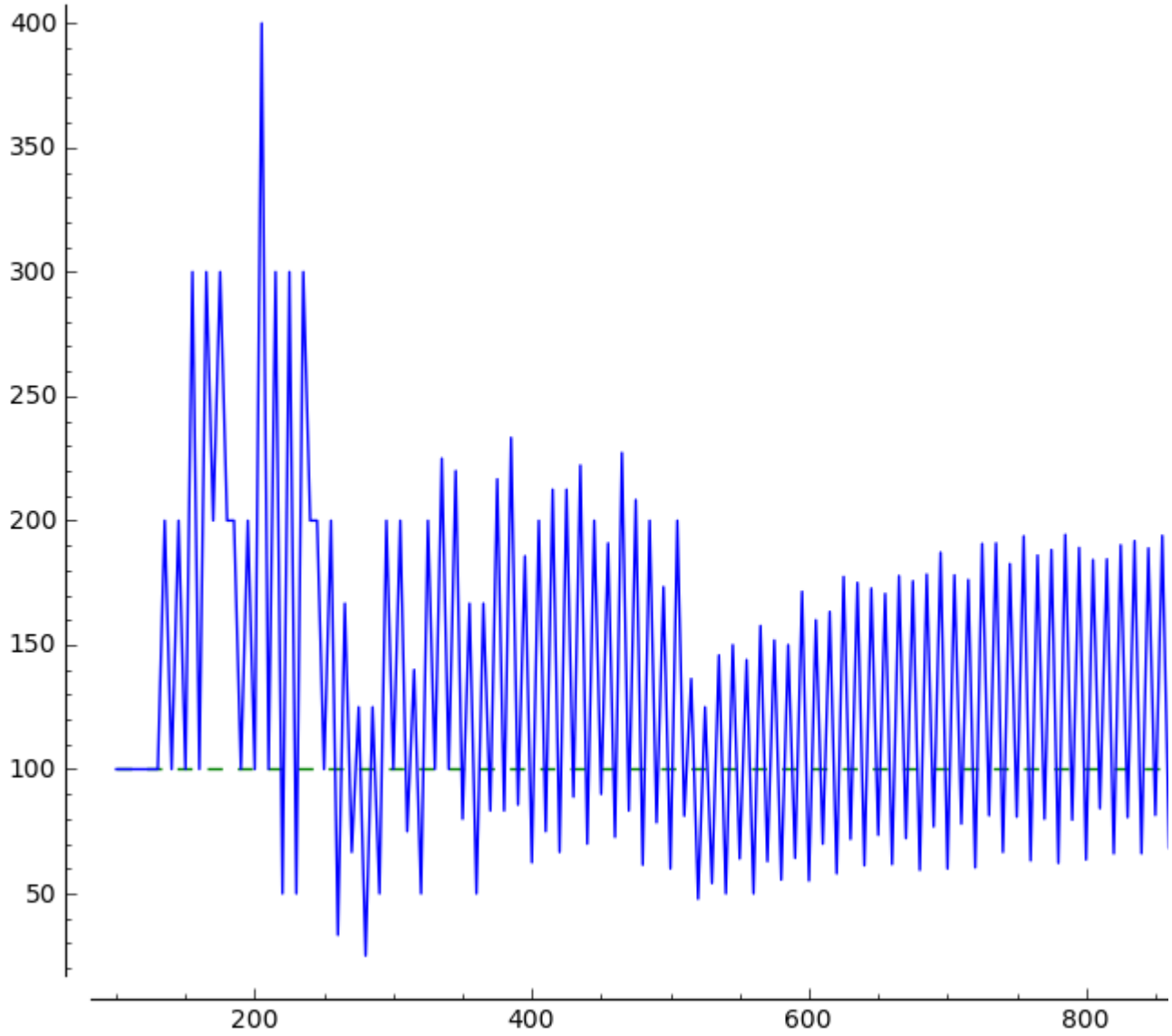
```
[(-1 : -1 : 1), (-1 : 1 : 1), (0 : 1 : 0), (1 : -3 : 1), (1 : 3 :
1), (4 : -9 : 1), (4 : 9 : 1)]
```

```
min_value = 100
max_value = 1000
step = 5
```

```
points = generate_data(P[0], w, min_value, max_value, step)
```

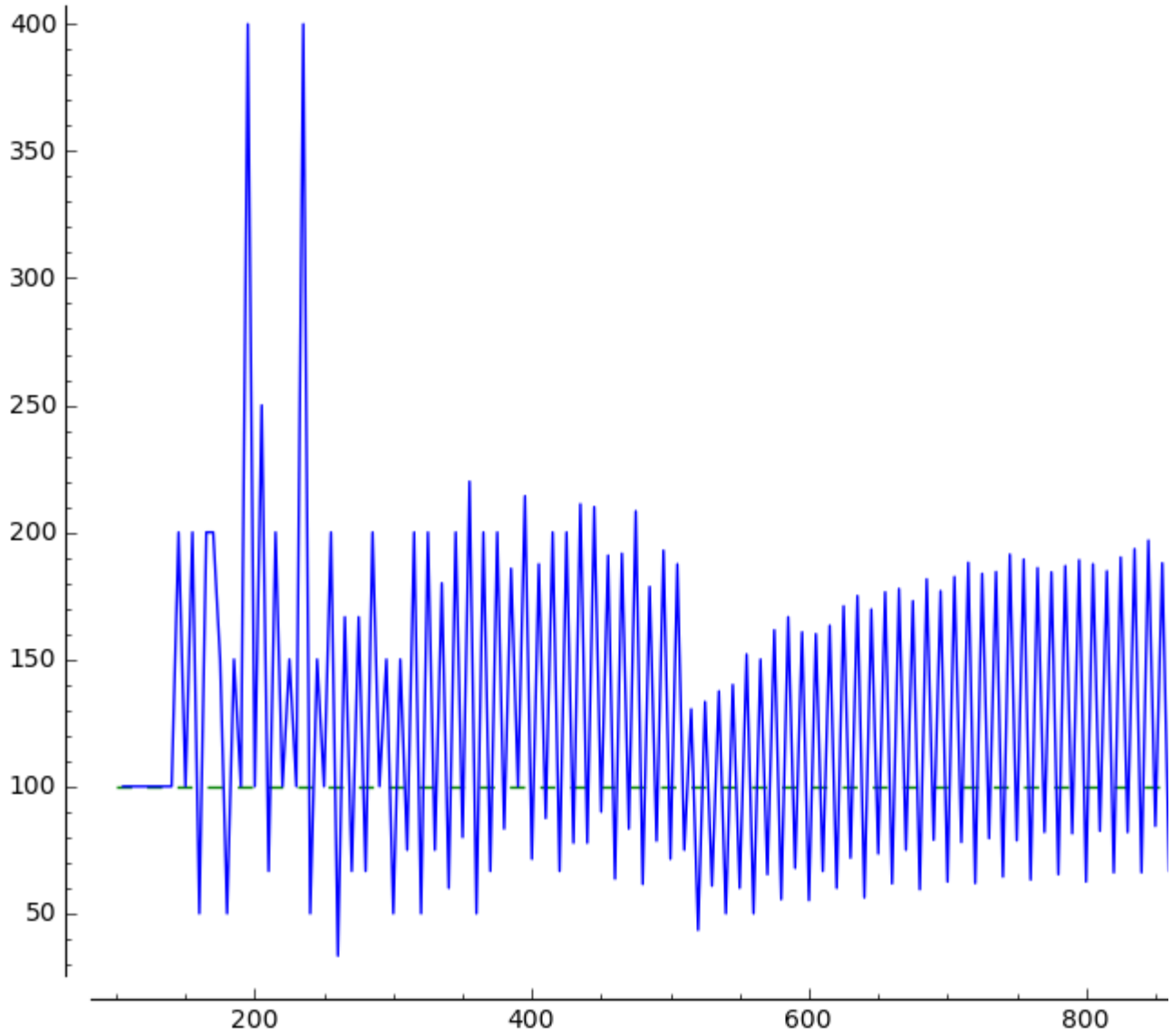
```
G = list_plot(points, plotjoined = True)
F = line([(min_value,100), (max_value,100)], rgbcolor = (0,0.5,0),
```

```
linestyle="--")
plot(F+G)
```



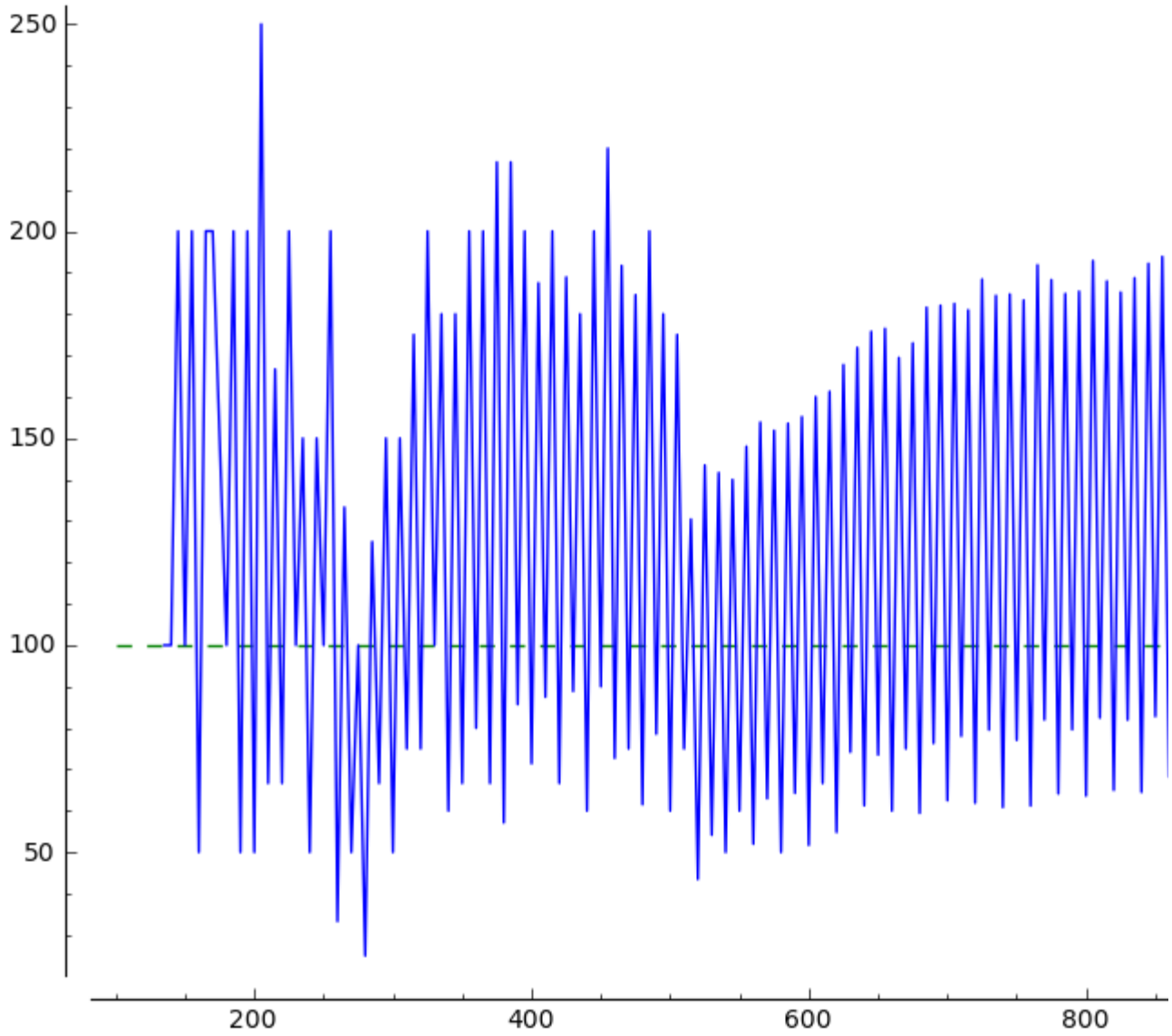

```
points = generate_data(P[0], 3, min_value, max_value, step)
```

```
G = list_plot(points, plotjoined = True)
F = line([(min_value,100), (max_value,100)], rgbcolor = (0,0.5,0),
linestyle="--")
plot(F+G)
```




```
points = generate_data(P[0], 2, min_value, max_value, step)
```

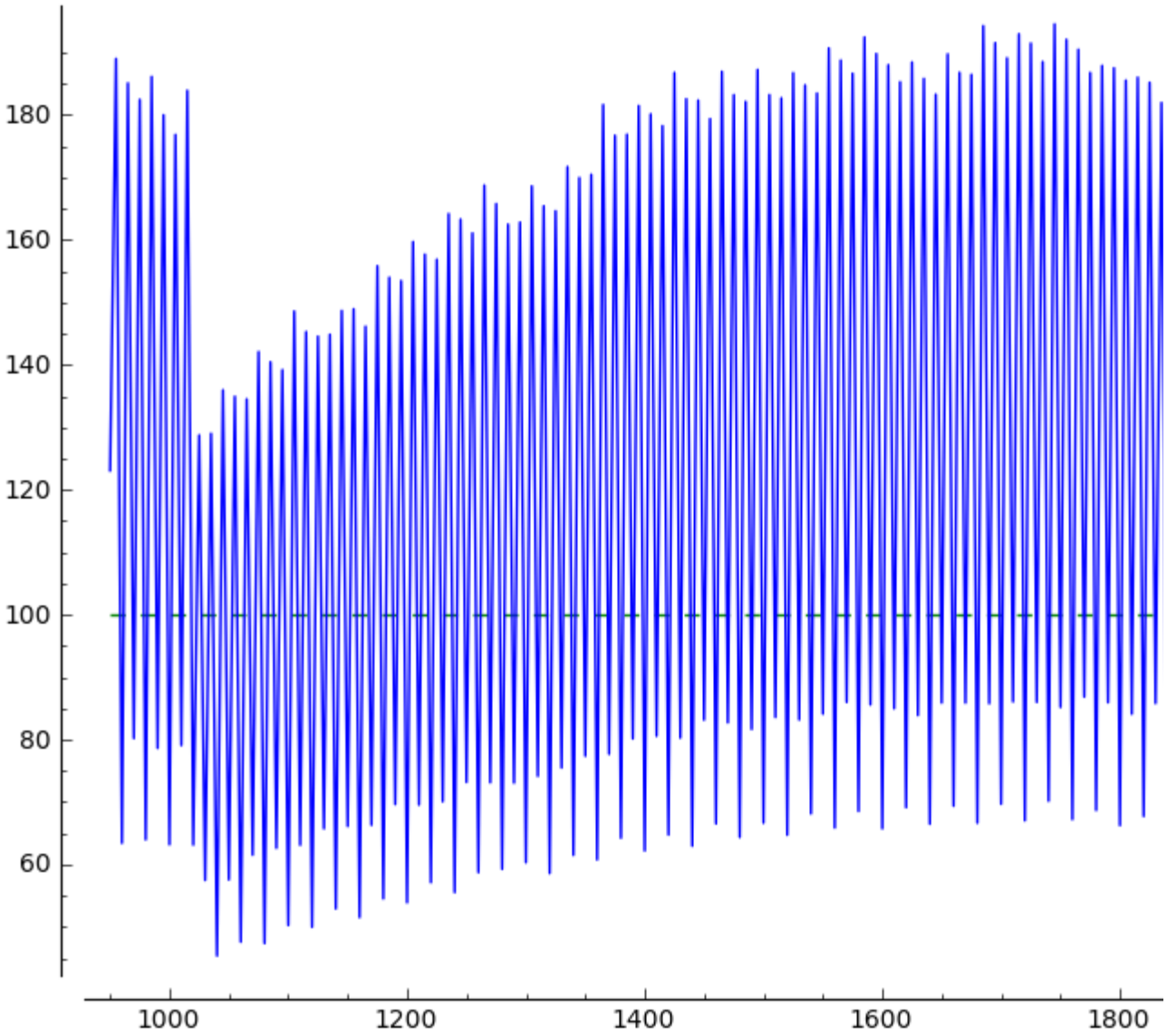
```
G = list_plot(points, plotjoined = True)
F = line([(min_value,100), (max_value,100)], rgbcolor = (0,0.5,0),
linestyle="--")
plot(F+G)
```




```
min_value = 950
max_value = 2000
step = 5
```

```
points = generate_data(P[0], 4, min_value, max_value, step)
```

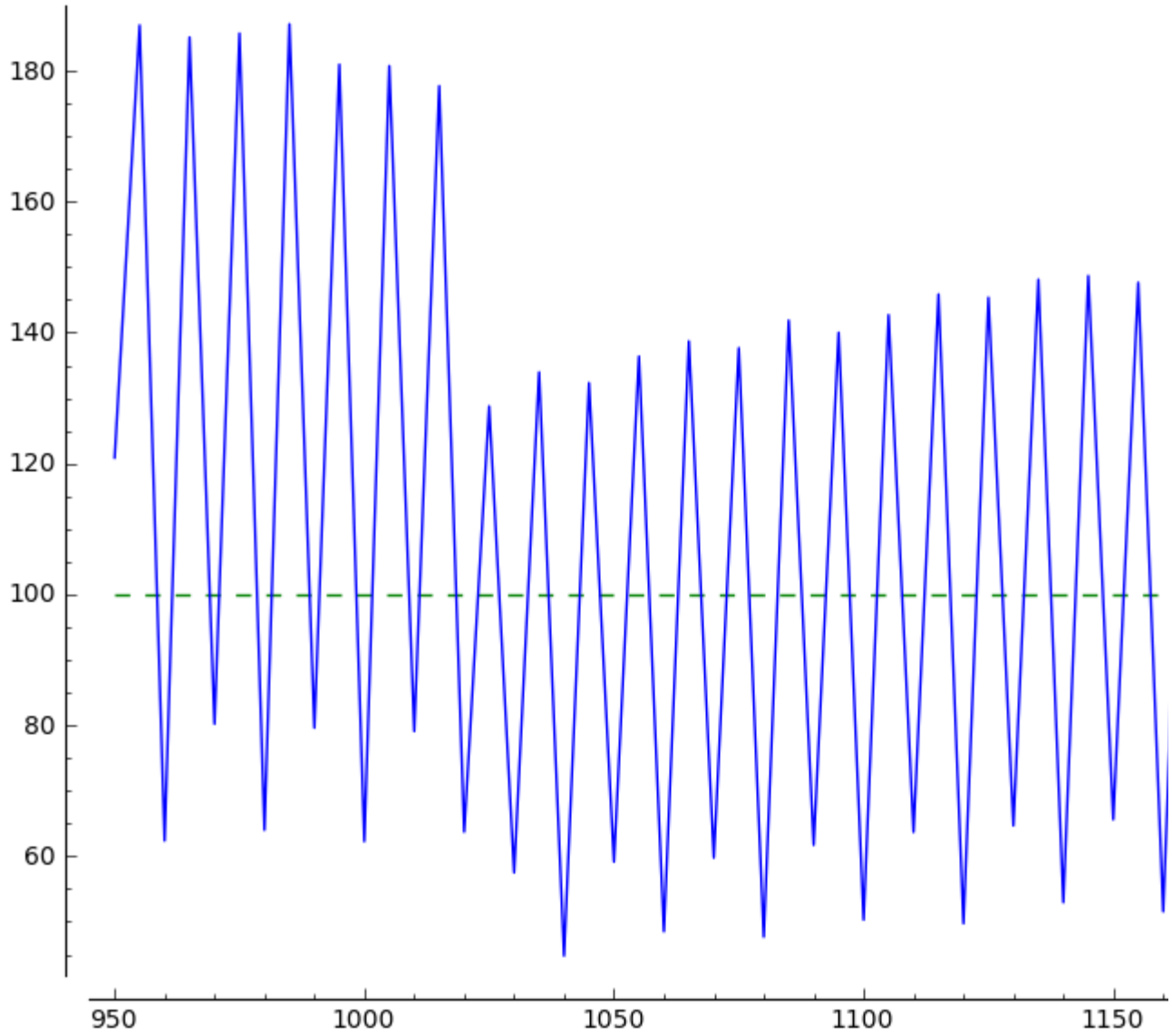
```
G = list_plot(points, plotjoined = True)
F = line([(min_value,100), (max_value,100)], rgbcolor = (0,0.5,0),
linestyle="--")
plot(F+G)
```




```
min_value = 950
max_value = 1200
step = 5
```

```
points = generate_data(P[0], 3, min_value, max_value, step)
```

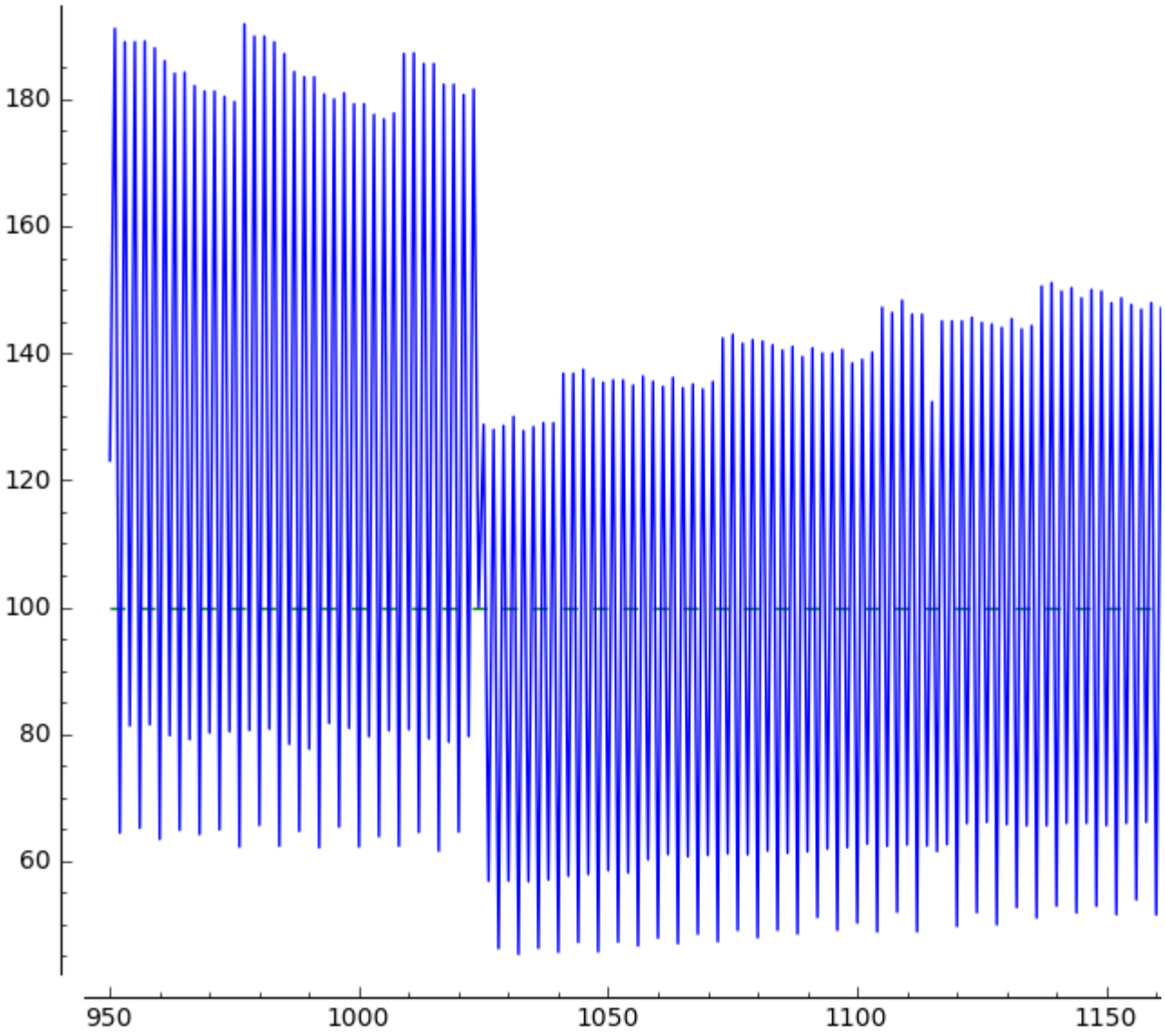
```
G = list_plot(points, plotjoined = True)
F = line([(min_value,100), (max_value,100)], rgbcolor = (0,0.5,0),
linestyle="--")
plot(F+G)
```




```
min_value = 950
max_value = 1200
step = 1
```

```
points = generate_data(P[0], 4, min_value, max_value, step)
```

```
G = list_plot(points, plotjoined = True)
F = line([(min_value,100), (max_value,100)], rgbcolor = (0,0.5,0),
linestyle="--")
plot(F+G)
```


180
160
140
120
100
80
60
950
1000
1050
1100
1150