6.2

```python
def calcular_c(err,x,y):

    #calculo w
    w = 1/np.square(err)

    #terminos suma
    a = np.sum(w*np.square(x))
    b = np.sum(w*y)
    c = np.sum(w*x)
    d = np.sum(w*x*y)

    #terminos delta
    e = np.sum(w)
    f = np.sum(w*np.square(x))
    g = np.square(np.sum(w*x))
    delta = f*e - g

    #calculo c
    c = (a*b - c*d)/delta

    return c
```

```python
def calcular_m(err,x,y):

    #calculo w
    w = 1/np.square(err)

    #terminos suma
    a = np.sum(w)
    b = np.sum(w*x*y)
    c = np.sum(w*x)
    d = np.sum(w*y)

    #terminos delta
    e = np.sum(w)
    f = np.sum(w*np.square(x))
    g = np.square(np.sum(w*x))
    delta = f*e - g

    #calculo m
    m = (a*b - c*d)/delta

    return m
```

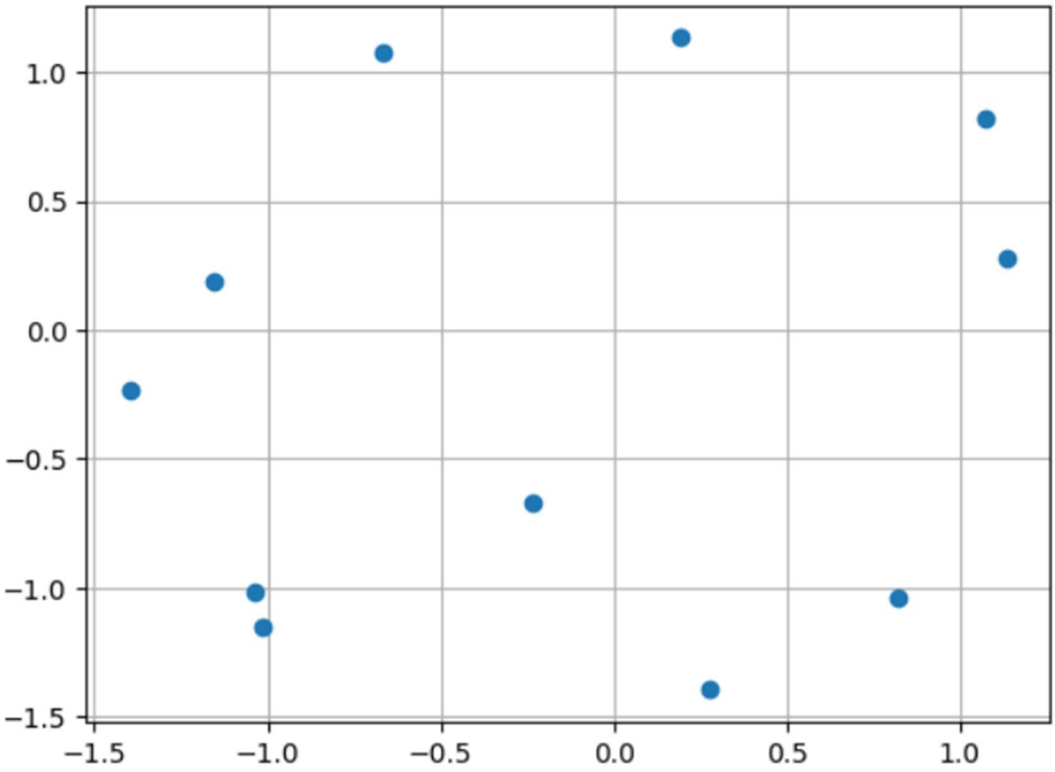
```python
def calcular_ac(err,x):

    #calculo w
    w = 1/np.square(err)

    #terminos suma
    a = np.sum(w*np.square(x))

    #terminos delta
    e = np.sum(w)
    f = np.sum(w*np.square(x))
    g = np.square(np.sum(w*x))
    delta = f*e - g

    #calculo ac
    ac = np.sqrt(a/delta)

    return ac
```

```python
def calcular_am(err,x):

        #calculo w
        w = 1/np.square(err)

        #terminos suma
        a = np.sum(w)

        #terminos delta
        e = np.sum(w)
        f = np.sum(w*np.square(x))
        g = np.square(np.sum(w*x))
        delta = f*e - g

        #calculo am
        am = np.sqrt(a/delta)

        return am
```

```
c
✓ 0.0s
-0.9474964662767381

m
✓ 0.0s
2.0284648216102745

ac
✓ 0.0s
3.386858673521736

am
✓ 0.0s
0.05197830827721802
```

```python
model = c + m*frecuencia
residuals = voltaje-model
normalized_residuals = residuals/error
lagged = np.roll(normalized_residuals, -1)
plt.plot(normalized_residuals, lagged, 'o')
plt.grid()
plt.show()
```

Lagged plot




```
def Durbin_Watson(residuals):
    return np.sum(np.square(normalized_residuals-lagged))/np.sum(normalized_residuals**2)
```

46] ✓ 0.0s

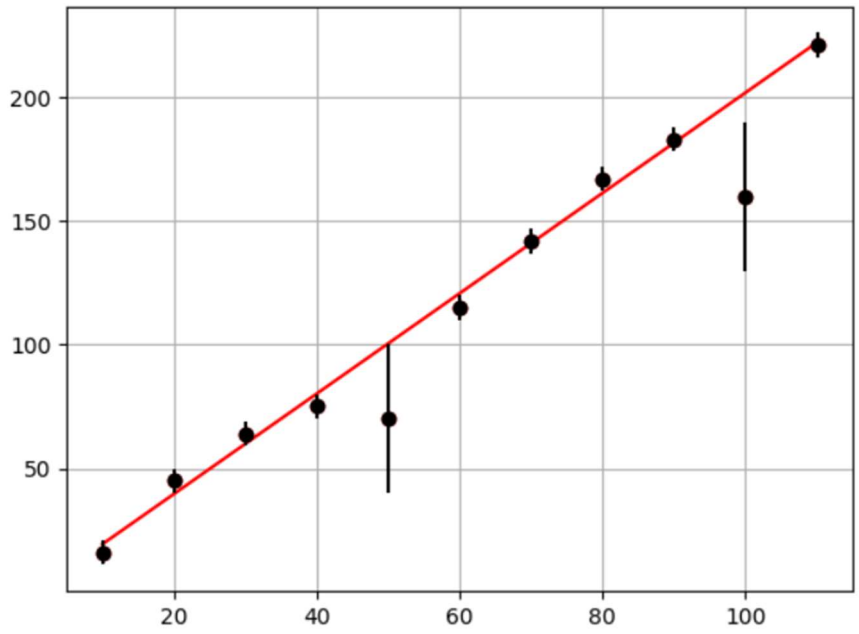
```
D = Durbin_Watson(residuals)
D
```

47] ✓ 0.0s

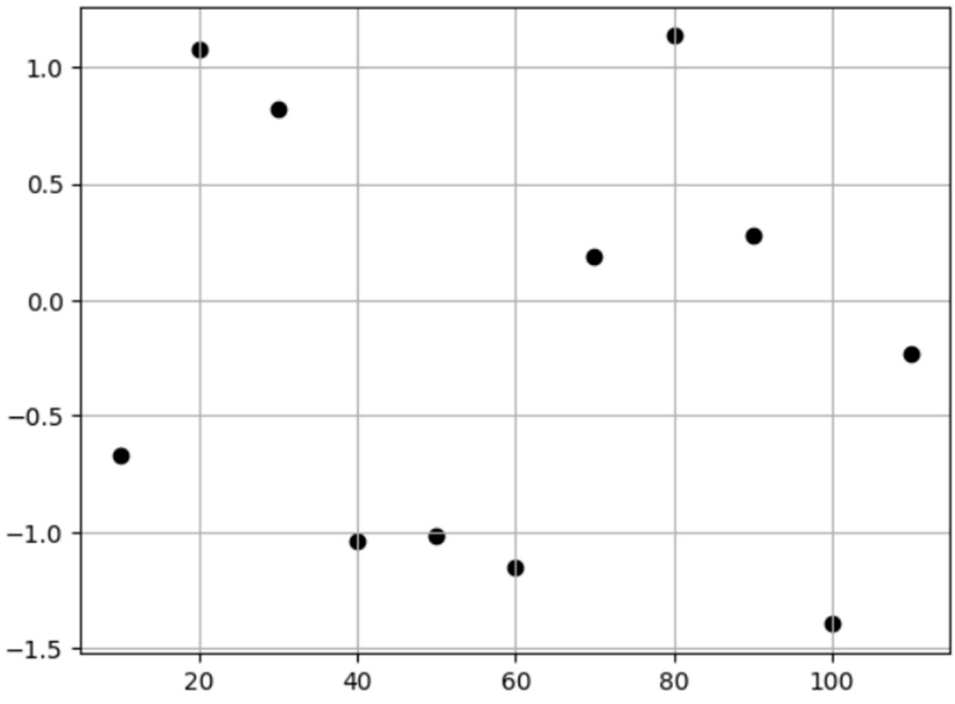
```
1.572140044519231
```

6.3

Datos con best-fit line



Normalized residuals



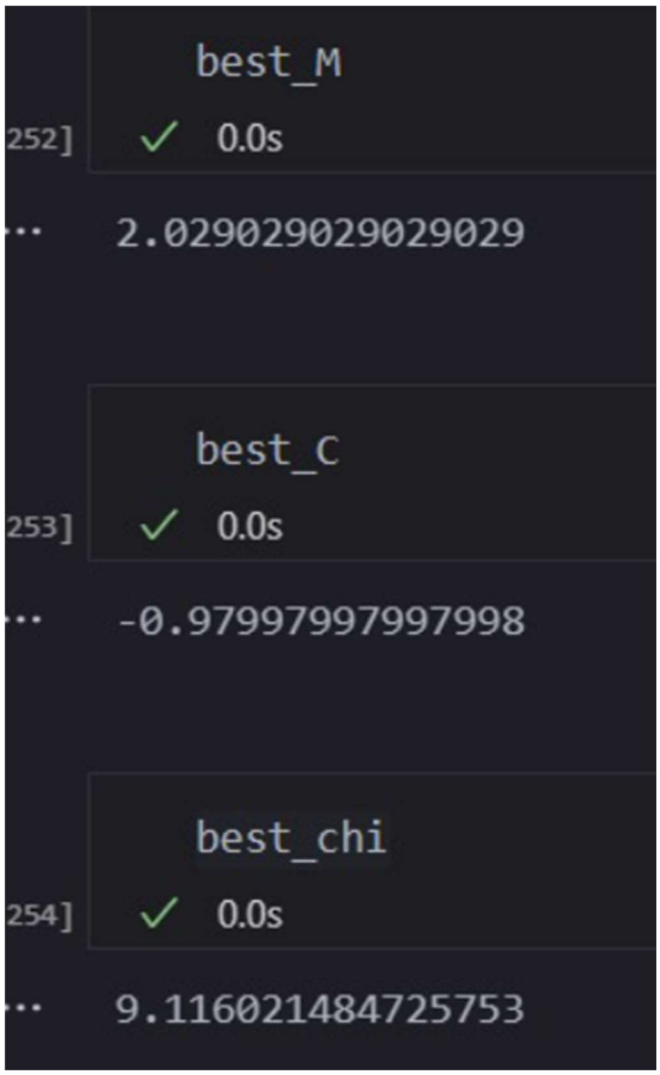
6.4

```python
def chi2(M, C):
    res = ((voltaje - (M * frecuencia + C)) / error) ** 2
    return np.sum(res)
```

✓ 0.0s

```python
M_values = np.linspace(1, 3, 1000)
C_values = np.linspace(-1, 1, 1000)
best_chi = np.inf
# Calculate chi-squared for each combination of parameters
chi2_grid = np.zeros((len(M_values), len(C_values)))
for i, M in enumerate(M_values):
    for j, C in enumerate(C_values):
        chi2_grid[i, j] = chi2(M, C)
        if chi2_grid[i, j] < best_chi:
            best_chi = chi2_grid[i, j]
            best_M = M
            best_C = C
```

✓ 8.0s




```python
def solve_Espacio_fase_c(m=m_min,c=c_min,x=x,y=y,contorno=1,alpha=dy,gmm=0.0001,infer=-1,tolerancia=1e-7):

    mjAnterior = m
    d = 1
    cj = c

    while d>tolerancia:
        #Loop que minimiza c

        cjAnterior = cj

        cj = nwt(f_c,cj+0.01*infer,args=(mjAnterior,x,y,alpha,contorno)) #Newton rapshon para minimizar c

        termino = 2*gmm*np.sum(x*(y-(mjAnterior)*x-cj)/(alpha)**2)

        mjAnterior += termino

        diff=np.abs(cj-cjAnterior)

        d = diff

    return cj
```

```python
def solve_Espacio_fase_m(m=m_min,c=c_min,x=x,y=y,contorno=1,alpha=dy,gmm=0.05,infer=-1,tolerancia=1e-7):

    cjAnterior = c
    d = 1
    mj = m

    while d>tolerancia:
        #loop que minimiza m

        mjAnterior = mj

        mj = nwt(f_m,mj+0.02*infer,args=(cjAnterior,x,y,alpha,contorno)) #Newton rapshon para minimizar m

        termino = 2*gmm*np.sum((y-(mj)*x-cjAnterior)/(alpha)**2)

        cjAnterior += termino

        diff=np.abs(mj-mjAnterior)

        d = diff

    return mj
```

Errores calculados de m y c:

Contorno 1:

```
Inferior:


m:
-0.017325659748360422


c:
-1.1289533645592529


Superior:


m:
0.017325677963016872


c:
1.1289523737901974
```

Contorno 4:

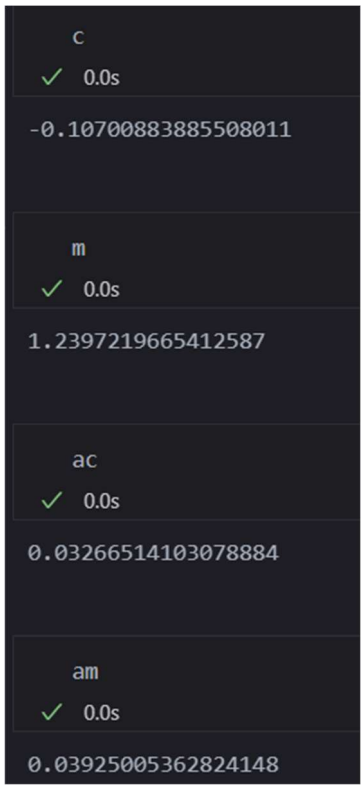
```
Inferior:


m:
-0.03465177823317589


c:
-2.2579062566317956


Superior:


m:
0.03465179644876434


c:
2.257905265862669
```

Contorno 9:

```
Inferior:


m:
-0.05197786414732053


c:
-3.3868591537285657


Superior:


m:
0.05197788236315951


c:
3.386858162959412
```

Esto se ajusta bastante bien a lo esperado.

6.5



M = 1.24 +- 0.04

C = -0.11 +- 0.03

Calculando la velocidad de la luz tenemos:
Vc = 304025095.50870967 +- 9807260.370254517

Ósea

Vc = 3.04*10^8 +- 0.10*10^8

El intercepto no es consistente, pero la velocidad de la luz da razonable.

6.6

i) Errores originales

ii) Errores ctes de 4

iii) errores 1 y final de 1, el resto de 8

A pesar de que no cambiamos los datos usados, tan solo cambiar los errores asociados a las mediciones impactó fuertemente el resultado. Esto, claramente se da porque en un cálculo con pesos de los mejores valores para la pendiente y corte, entre más preciso sea el dato, más contribuirá al resultado final. Por esto cambiar dichas precisiones afecta nuestros valores óptimos calculados.