

```
import numpy as np
import matplotlib.pyplot as plt
```

✓ Datos de entrada

```
P = 950 * 745.7      # Potencia de HP a W = J / s
r = 1.2              # kg/m^3
CdA = 0.85           # Coeficiente de arrastre por área (frontal) en m^2
m = 940              # Masa en kg
mu = 0.015           # Coeficiente de rozamiento dinámico
g = 9.81             # Aceleración de la gravedad en Baku en m/s^2
```

✓ Función a evaluar

$$f(v) = \left(\frac{1}{2} \right) * r * Cd * A * v^3 + (mu * m * g) v - P = 0$$

```
def calcular_f(P, r, CdA, m, mu, g, v):
    return ( 0.5 * r * CdA) * (v**3) + (mu * m * g) * v - P

print(calcular_f(P, r, CdA, m, mu, g, 125))
```

```
304968.875
```

✓ Gráfico de la función para un intervalo coherente

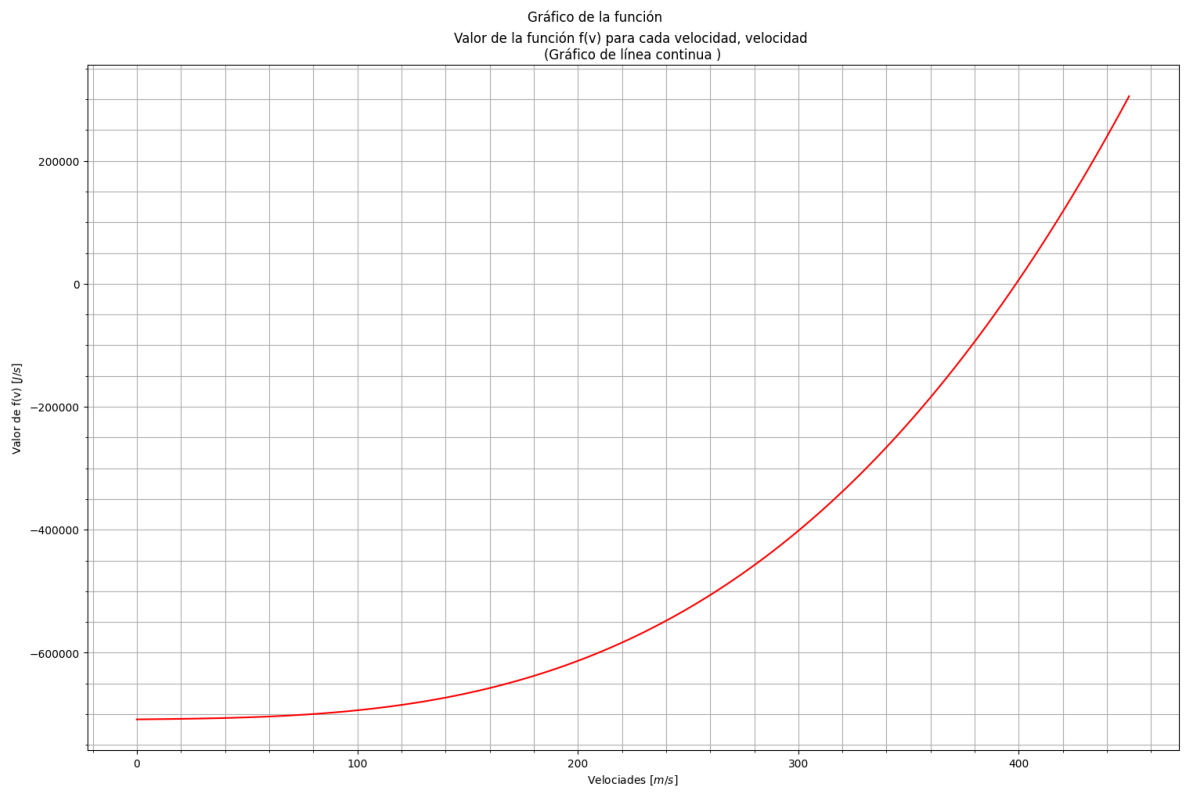
El intervalo inicial es 0 a 450 km/h que tiene sentido en un Formula 1

Podría graficar un intervalo mucho más amplio para ver todas las 3 raíces, pero podrían no verse porque hay raíces complejas o porque gráficamente están muy cercanas

```
velocidad_minima = 0          # [m/s]
velocidad_maxima = 125        # [m/s]          450 / 3.6 = 125 [m/s]
pasos = 1                     #                variación de 1 m/s

cantidad_de_puntos = int(1 + (velocidad_maxima - velocidad_minima)/pasos)
velocidades = np.linspace(velocidad_minima, velocidad_maxima, cantidad_de_punto
```

```
print(velocidades)
```

```
print(calcular_f(P, r, CdA, m, mu, g, 380/3.6))
print(calcular_f(P, r, CdA, m, mu, g, 400/3.6))
```

```
-94004.77921810688
6542.477366255131
```

Por el valor de la función para esas dos velocidades $F(v_1) \cdot F(v_2) < 0$ Y como $F(v)$ es continua para todo el dominio, por el teorema de Bolzano hay al menos una raíz en este intervalo v_1, v_2 (380 y 400 km/h)

```
calcular_f(P, r, CdA, m, mu, g, 398/3.6) * calcular_f(P, r, CdA, m, mu, g,
```

```
-5050267.211824421
```

Intervalo a, b

[398; 399] km/h aprox = [110; 111] m/s | este intervalo también cumple como el de 380 y 400 pero uso el otro porque es mejor para graficar

[380; 400] km/h aprox = [105; 112] m/s | a = 105 m/s ; b = 112 m/s

Teorema: Si $F'(X)$ es continua y $F'(X) \neq 0$ entonces:

$$|m_{k+1} - \alpha| \leq |m_{k+1} - m_k| = \Delta \alpha_{k+1}$$

$f'(v) = ((3/2) r * Cd * A) v^2 + (mu * m * g) > 0$ para el intervalo [a; b]

continua (polinómica)

Entonces el método es **incondicionalmente convergente**

› Métodos de Arranque

↳ 2 celdas ocultas

✓ Método de Punto Fijo

$$g(x) = x$$

Planteo equivalente $g(x) = x - q F(x)$ $q = \text{cte} \neq 0$

```
fig, ax = plt.subplots(
    nrows=1,
    ncols=1,
    figsize=(15, 10),
    constrained_layout=True
)

fig.suptitle("Gráfico de las funciones")

factor_de_escal_a_velocidad = 3600 / 1000

velocidades_kmh = velocidades * factor_de_escal_a_velocidad

mascara = (velocidades_kmh >= 380) & (velocidades_kmh <= 400)

v = velocidades[mascara]
v_kmh = velocidades_kmh[mascara]
f = fx[mascara]

gv = v - f

ax.plot(v_kmh, f, label="f(v)")
ax.plot(v_kmh, v, label="y = v")
ax.plot(v_kmh, gv, label="g(v) = v - f(v)")

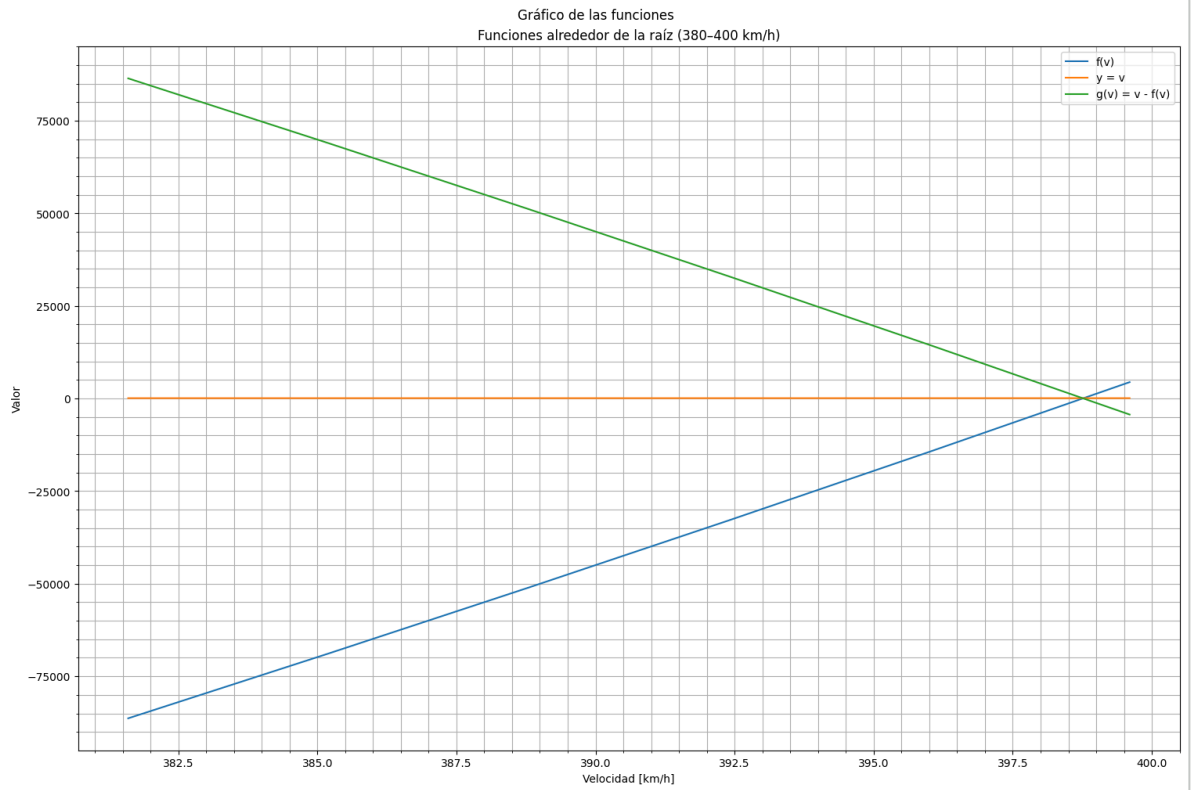
ax.set_title(
    "Funciones alrededor de la raíz (380-400 km/h)"
)

ax.set_xlabel("Velocidad [km/h]")
ax.set_ylabel("Valor")

ax.legend()

ax.minorticks_on()
ax.grid(True, which="both")

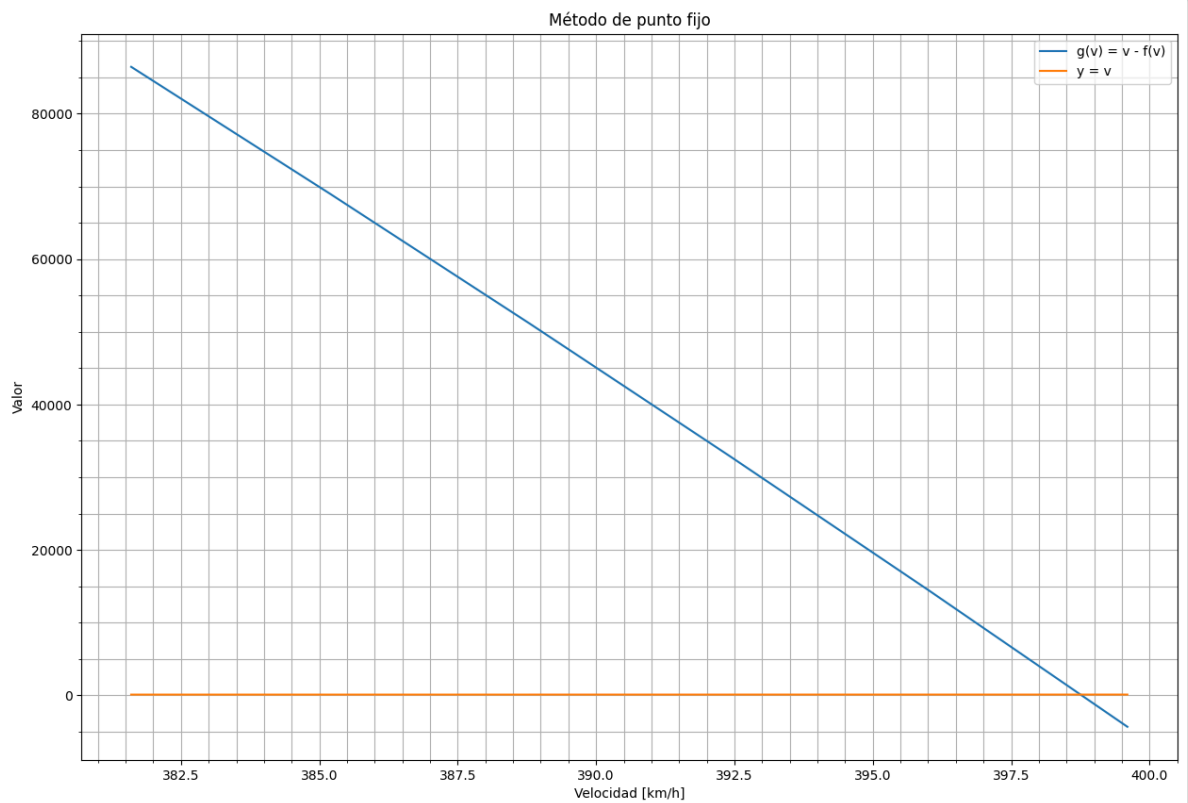
plt.show()
```



```
fx[mascara]
```

```
array([-86334.814, -68842.723, -51023.212, -32873.221, -14389.69 ,  
       4430.441])
```

```
fig, ax = plt.subplots(figsize=(15, 10))  
  
ax.plot(v_kmh, gv, label="g(v) = v - f(v)")  
ax.plot(v_kmh, v, label="y = v")  
  
ax.set_xlabel("Velocidad [km/h]")  
ax.set_ylabel("Valor")  
  
ax.set_title("Método de punto fijo")  
  
ax.legend()  
  
ax.minorticks_on()  
ax.grid(True, which="both")  
  
plt.show()
```



- Escoger una aproximación inicial $x_0 = 105$ semilla
- Generar la sucesión $x_{k+1} = g(x_k)$ $k \geq 0$ g función de iteración
- Hipótesis de convergencia $\lim_{k \rightarrow \infty} x_{k+1} = \beta$
- Seleccionar método óptimo

✓ Método Newton-Raphson

Uso directamente el método Newton-Raphson porque la derivada de la función es fácil de calcular y de operar

La función $f(v)$ es polinómica (grado 3), por lo tanto continua y derivable en cualquier punto del dominio (esto incluye al intervalo $[a, b]$)

$$f(v) = \left(\frac{1}{2} \right) r \cdot C_d \cdot A \cdot v^3 + (\mu \cdot m \cdot g) v - P$$

$$f(v) = A v^3 + B v - P$$

$$f'(v) = \left(\frac{3}{2} \right) r \cdot C_d \cdot A \cdot v^2 + (\mu \cdot m \cdot g) = 0$$

$$f'(v) = C v^2 + B$$

```
A = 0.5 * r * CdA
B = mu * m * g
C = 1.5 * r * CdA
```

```
print(A, B, C)
```

```
0.51 138.321 1.5299999999999998
```

```
def funcion(vel):
    return A * vel**3 + B * vel - P

def derivada(vel):
    return C * vel **2 + B
```

```
velocidades = np.linspace(105, 115, 1000)

gv = velocidades - funcion(velocidades)/ derivada(velocidades)

fig, ax = plt.subplots(
    figsize=(15, 10),
    constrained_layout=True
)

fig.suptitle("Método de Newton-Raphson")

ax.plot(
    velocidades,
    gv,
    label="g(v) = v - f(v)/f'(v)"
```

```
)

ax.plot(
    velocidades,
    velocidades,
    label="y = v"
)

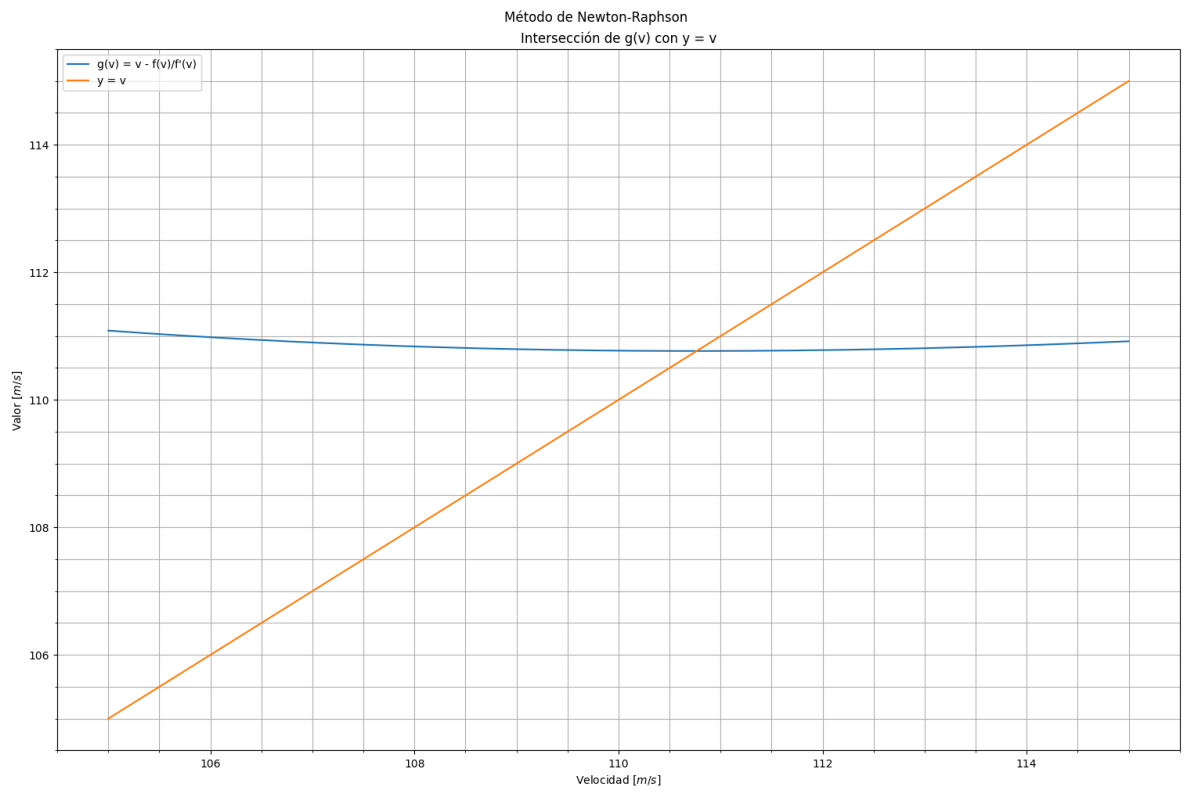
ax.set_title(
    "Intersección de g(v) con y = v"
)

ax.set_xlabel("Velocidad [ $\text{m/s}$ ]")
ax.set_ylabel("Valor [ $\text{m/s}$ ]")

ax.legend()

ax.minorticks_on()
ax.grid(True, which="both")

plt.show()
```



Acá si nos imaginamos un cuadrado que es el intervalo $[a;b] \times [a;b]$ ($a = 105 \text{ m/s}$ y $b = 112 \text{ m/s}$)

Se puede ver que la función $g(v)$ está completamente dentro del cuadrado para el intervalo $[a;b]$

Por lo tanto interseca al menos una vez con la recta $y=x$ y cumple la condición de existencia (hay al menos una intersección)

Aunque en el gráfico se ve como hay una sola intersección puede haber más y no ser gráficas (no para este ejemplo) Matemáticamente:

Una forma sería hallar las otras dos raíces de f , podría haber graficado un intervalo más grande que 0 a 450 km/h y haberlas visto

Otra es igualar la recta de 45° con $g(v)$ para ver que hay una sola intersección en $[a;b]$ (acá habría que hacer Ruffini)

Pero la más fácil parece ser que viendo que para el intervalo $[a;b]$ es estrictamente creciente:

$$f'(v) = 1.529 v^2 + 138.321$$

como

$f'(v) > 0$ para todo v (y por lo tanto para el intervalo)

Bolzano: hay una al menos una raíz

$g'(v) > 0$: hay como máximo una raíz

Entonces hay exactamente una sola raíz: se cumple unicidad

- Cumple condición de **existencia**
- Cumple condición de **unicidad**

Por lo tanto **converge** al **único punto fijo** v^* para cualquier v_0 perteneciente a $[a;b]$

Orden de Convergencia: P

$$P = \frac{\ln\left(\frac{\Delta_{k+1}}{\Delta_k}\right)}{\ln\left(\frac{\Delta_k}{\Delta_{k-1}}\right)}$$

Nota: las variables dk1, dk2 y dk3 representan:

dk1 = d k-1 # iteración 2 pasos anterior

dk2 = d k # iteración anterior

dk3 = d k+1 # iteración actual

```
# Defino una tolerancia
tolerancia = 1e-7
tol = 10e-8
print(tolerancia/tol)
#distintas formas de definir la tolerancia
```

1.0

```
vel = 105
iteracion = 0
condicion = True
dk1 = 0
dk2 = 0

print("Iteración    Velocidad                f(v)                f'(v)

while (iteracion < 20 and condicion):
    f = funcion(vel)
    df = derivada(vel)
    vel_nueva = vel - f / df      # g(x) = x - F(x)/F'(x)

    error = abs(vel_nueva - vel)

    if iteracion == 0:
        dk1 = error
        print(f"{iteracion}\t", f"{vel}                \t", f"{f}\t", f"{df}\t", f"{v

    elif iteracion == 1:
        dk2 = error
        print(f"{iteracion}\t", f"{vel}\t", f"{f}\t", f"{df}\t", f"{vel_nueva}\t
```

```

if iteracion >= 2:
    orden_de_convergencia_P = (np.log(error/dk2)) / (np.log(dk2/dk1))
    print(f"{iteracion}\t", f"{vel}\t", f"{f}\t", f"{df}\t", f"{vel_nueva}\t")

    dk1 = dk2
    dk2 = error

if error < tolerancia:
    condicion = False
    print("\nSe llegó a la tolerancia pedida")

vel = vel_nueva
iteracion += 1

if iteracion >= 20:
    print("\nSe llegó al máximo de iteraciones")

```

Iteración	Velocidad	$f(v)$	$f'(v)$
0	105	-103502.545000000004	17006.570999999996
1	111.08603256941097	6065.409600228537	19018.684146978718
2	110.76711410200522	17.270082707633264	18910.431956724566
3	110.76620084509247	0.00014134705998003483	18910.122411972705

Se llegó a la tolerancia pedida

nota: la función es un poco complicada porque añadir dk1 y dk2 de iteracio
así que podría ser más conveniente usar vectores para la próxima (de esta

```
print(vel_nueva)
```

110.76620083761779

```
print(calcular_f(P, r, CdA, m, mu, g, vel_nueva))
```

-1.1641532182693481e-10

```

print("V = "f"{vel_nueva:.7f}" " +- 0.0000001")
# escribí la tolerancia porque sino aparecía como 1e-7
# resultado truncado y sin redondeo

```

V = 110.7662008 +- 0.0000001

✓ Extra

Como extra voy a poner distintas semillas o no cortar cuando se cumple la tolerancia para ver más resultados

Acá todo igual que antes pero no corto para ver cómo sigue P

```

vel = 105
iteracion = 0
condicion = True
dk1 = 0
dk2 = 0

print("Iteración    Velocidad                f(v)                f'(v)

while (iteracion < 20):
    f = funcion(vel)
    df = derivada(vel)
    vel_nueva = vel - f / df      #  $g(x) = x - F(x)/F'(x)$ 

    error = abs(vel_nueva - vel)

    if iteracion == 0:
        dk1 = error
        print(f"{iteracion}\t", f"{vel}                \t", f"{f}\t", f"{df}\t", f"{v

    elif iteracion == 1:
        dk2 = error
        print(f"{iteracion}\t", f"{vel}\t", f"{f}\t", f"{df}\t", f"{vel_nueva}\t

    if iteracion >= 2:
        orden_de_convergencia_P = (np.log(error/dk2)) / (np.log(dk2/dk1))
        print(f"{iteracion}\t", f"{vel}\t", f"{f}\t", f"{df}\t", f"{vel_nueva}\t

        dk1 = dk2
        dk2 = error

    vel = vel_nueva
    iteracion += 1

if iteracion >= 20:
    print("\nSe llegó al máximo de iteraciones")

```

```

Iteración  Velocidad                f(v)                f'(v)
0      105                -103502.54500000004        17006.570999999996  111.086
1      111.08603256941097    6065.409600228537    19018.684146978718  110.7671141
2      110.76711410200522    17.270082707633264    18910.431956724566  110.7662008
3      110.76620084509247    0.00014134705998003483  18910.122411972705  110.766
4      110.76620083761779    -1.1641532182693481e-10  18910.122409439202  110.766
/tmp/ipykernel_1967/2202390329.py:25: RuntimeWarning: divide by zero encountered
orden_de_convergencia_P = (np.log(error/dk2)) / (np.log(dk2/dk1))

```

```

-----
ZeroDivisionError                                Traceback (most recent call last)
/tmp/ipykernel_1967/2202390329.py in <cell line: 0>()
    23
    24 if iteracion >= 2:
--> 25     orden_de_convergencia_P = (np.log(error/dk2)) /
(np.log(dk2/dk1))
    26     print(f"{iteracion}\t", f"{vel}\t", f"{f}\t", f"{df}\t", f"
{vel_nueva}\t", f"{error}\t", f"{orden_de_convergencia_P}")
    27

```

Pasos siguientes: [Explicar error](#)

```

Iteración  Velocidad                f(v)                f'(v)                vel_nueva                Error                P
0      105                -103502.54500000004        17006.570999999996  111.08603256941097    6.086032569410975    -
1      111.08603256941097    6065.409600228537    19018.684146978718  110.76711410200522    0.31891846740575147    -
2      110.76711410200522    17.270082707633264    18910.431956724566  110.76620084509247    0.0009132569127530132  1.9857709429742452
3      110.76620084509247    0.00014134705998003483  18910.122411972705  110.76620083761779    7.47468220652081e-09    2.00032513046631
4      110.76620083761779    -1.1641532182693481e-10  18910.122409439202  110.76620083761779    0.0                inf
/tmp/ipykernel_1967/2202390329.py:25: RuntimeWarning: divide by zero encountered in log
orden_de_convergencia_P = (np.log(error/dk2)) / (np.log(dk2/dk1))
-----
ZeroDivisionError                                Traceback (most recent call last)
/tmp/ipykernel_1967/2202390329.py in <cell line: 0>()
    23
    24 if iteracion >= 2:
--> 25     orden_de_convergencia_P = (np.log(error/dk2)) / (np.log(dk2/dk1))
    26     print(f"{iteracion}\t", f"{vel}\t", f"{f}\t", f"{df}\t", f"{vel_nueva}\t", f"{error}\t", f"{orden_de_convergencia_P}")
    27

ZeroDivisionError: float division by zero

```

Qué pasa acá?

Me dice de una división por cero. En el orden de convergencia. Porque el Error que es $| \text{velocidad en } k - \text{velocidad en } k - 1 |$ es 0.0

Es decir que para esa iteración el resultado ya es muy parecido y la resta de esos valores en el programa ya me da 0, no es que haya algo mal

Si la semilla es 0 m/s en vez de $a = 105$ m/s

```

vel = 0
iteracion = 0
condicion = True
dk1 = 0
dk2 = 0

```

```

print("Iteración    Velocidad                                f(v)                                f'(v)

while (iteracion < 20 and condicion):
    f = funcion(vel)
    df = derivada(vel)
    vel_nueva = vel - f / df      #  $g(x) = x - F(x)/F'(x)$ 

    error = abs(vel_nueva - vel)

    if iteracion == 0:
        dk1 = error
        print(f"{iteracion}\t", f"{vel}\t", f"{f}\t", f"{df}\t", f"{v

    elif iteracion == 1:
        dk2 = error
        print(f"{iteracion}\t", f"{vel}\t", f"{f}\t", f"{df}\t", f"{vel_nueva}\t

    if iteracion >= 2:
        orden_de_convergencia_P = (np.log(error/dk2)) / (np.log(dk2/dk1))
        print(f"{iteracion}\t", f"{vel}\t", f"{f}\t", f"{df}\t", f"{vel_nueva}\t

        dk1 = dk2
        dk2 = error

    if error < tolerancia:
        condicion = False
        print("\nSe llegó a la tolerancia pedida")

    vel = vel_nueva
    iteracion += 1

if iteracion >= 20:
    print("\nSe llegó al máximo de iteraciones")

```

Iteración	Velocidad	f(v)	f'(v)
0	0	-708415.0	138.321
1	5121.528907396563	68512381060.10208	40132127.59542691
2	3414.3584889533327	20299833571.218822	17836639.474364154
3	2276.2610575176122	6014616826.343423	7927625.856015934
4	1517.5702545479958	1781948362.7228844	3523758.121557969
5	1011.8748290245701	527816638.6282836	1566691.0455086592
6	674.9758340750477	156217008.9681343	697194.6571755187
7	450.91070769986567	46110436.8523215	311218.63446714234
8	302.7497952555637	13485589.735799985	140374.20194674714
9	206.6809339635442	3822856.8282600483	65495.343949985654
10	148.31255847750498	975910.6082955657	33793.1419532793
11	119.43360399482295	176965.14577811584	21962.831217684045
12	111.37612069387097	11596.818925354513	19117.42059904804
13	110.76951067212296	62.59123223600909	18911.244276648587
14	110.76620093579254	0.0018564964411780238	18910.122442715

Se llegó a la tolerancia pedida

Le toma 14 iteraciones en vez de 3 con esta semilla

Si ignoro la tolerancia:

```
vel = 0
iteracion = 0
dk1 = 0
dk2 = 0

print("Iteración    Velocidad                f(v)                f'(v)

while (iteracion < 20):
    f = funcion(vel)
    df = derivada(vel)
    vel_nueva = vel - f / df      #  $g(x) = x - F(x)/F'(x)$ 
```