Métodos de Ruge Kutta

Son un conjunto de métodos iterativos para aproximar ecuaciones diferenciales ordinarias basados en el problema de valor inicial (Cauchy) en el cual se predice una nueva pendiente proyectada a un paso de distancia (constante) y lo que diferencia los diversos métodos es el como se calcula la pendiente

Ecuación diferencial Ordinaria

$$\frac{dy}{dx} = f(x_n, y_n)$$

Método numérico

$$y_{n+1} = y_n + h\emptyset$$

Donde;

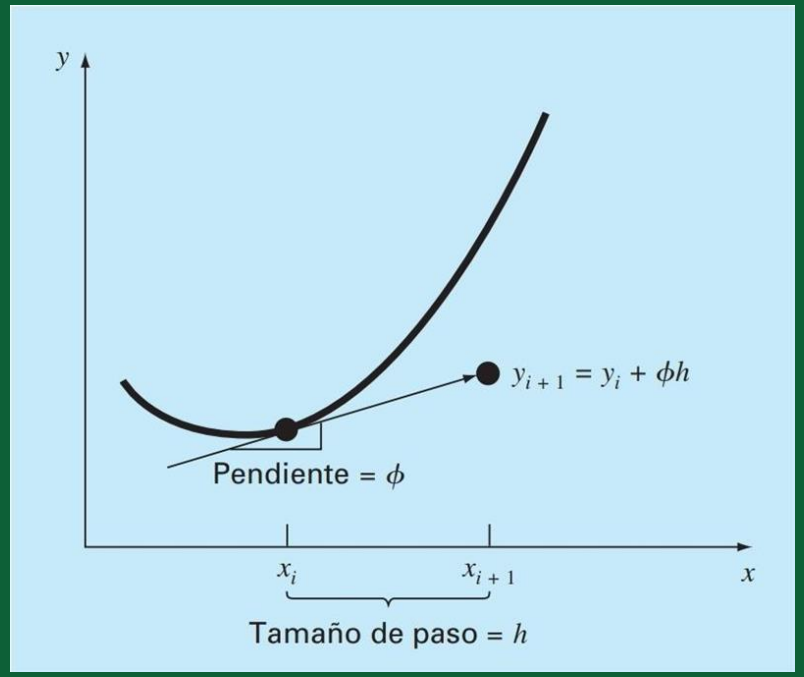
y_{n+1} = El valor de la función en el siguiente paso

y_n = El valor de la función en el paso actual

h = El tamaño del paso

$$y_{n+1} = y_n + hf(x_n, y_n)$$

$\emptyset = f(x_n, y_n)$ = Derivada de la función en el paso actual



Problema de valor inicial

El problema del valor inicial (Cauchy)

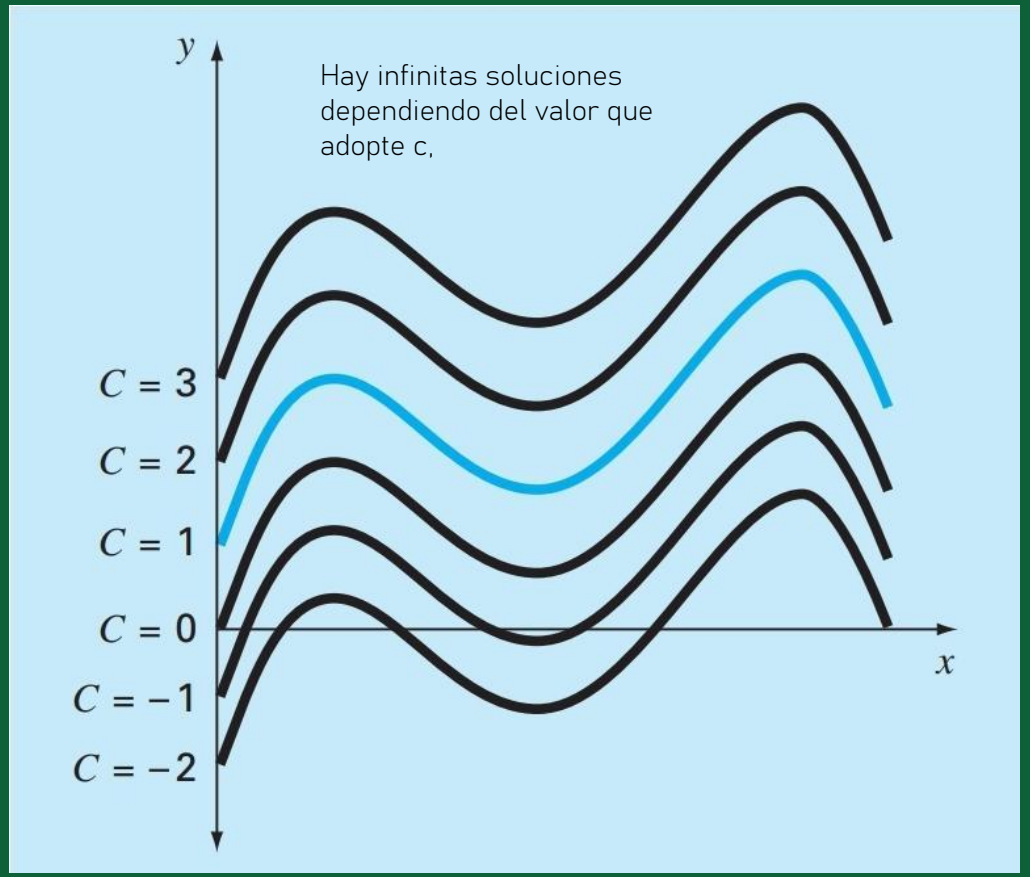
El problema de Cauchy es un concepto fundamental en ecuaciones diferenciales. Se trata de encontrar la solución de una ecuación diferencial que cumple con ciertas condiciones iniciales o de frontera

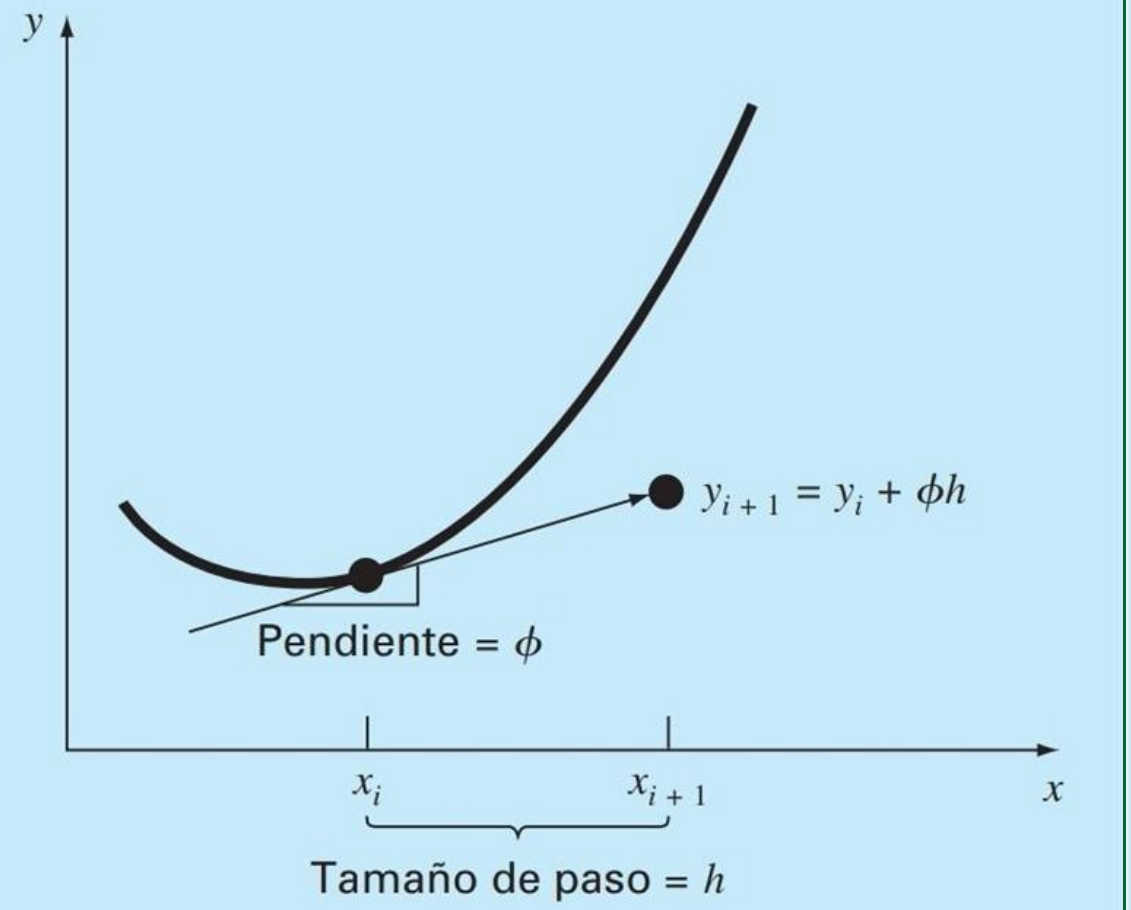
$$\frac{dy}{dx} = f(x_n, y_n)$$

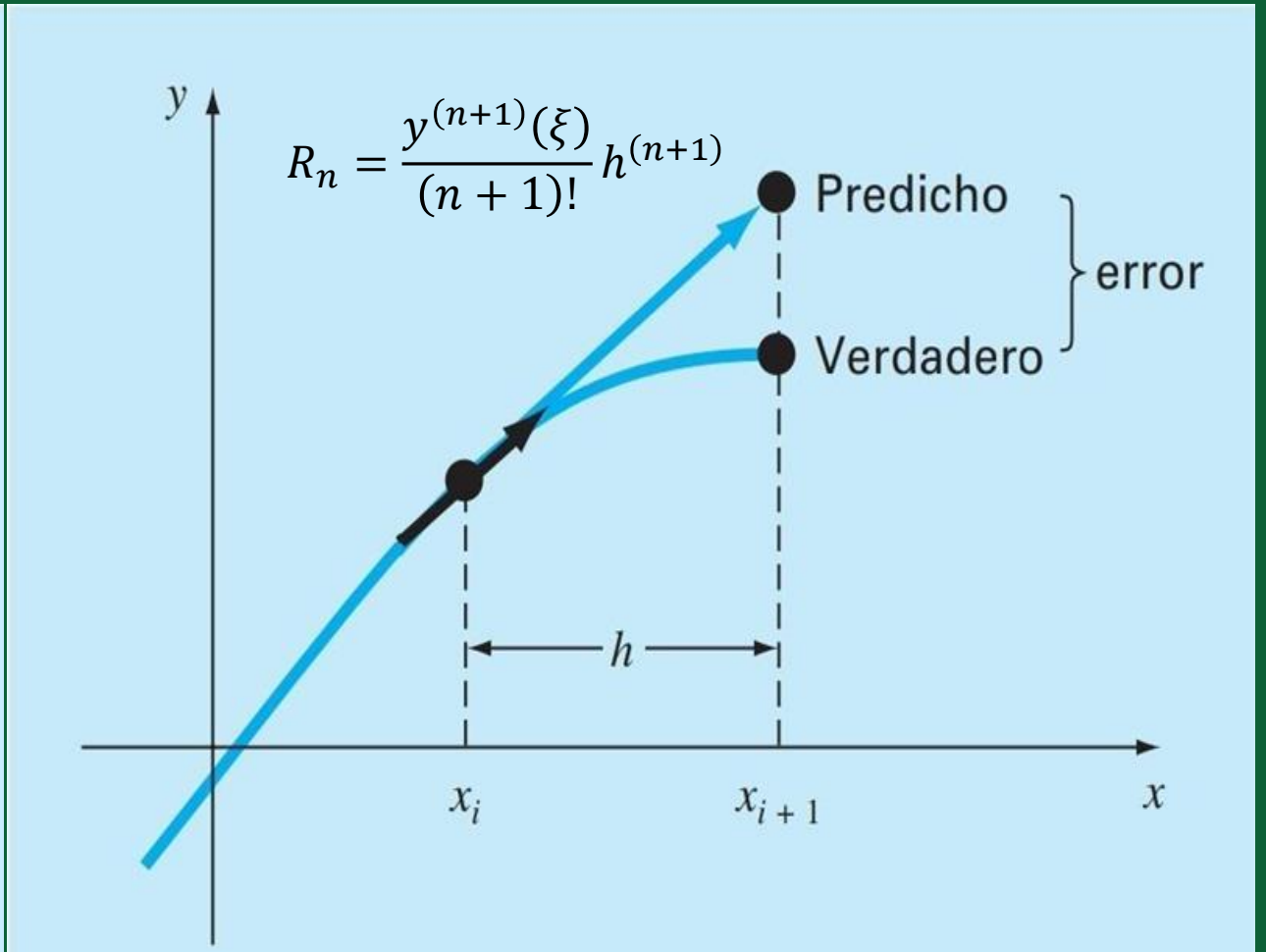
Ecuación diferencial Ordinaria

Caso polinómico por ejemplo

$$y = F(x) + c$$



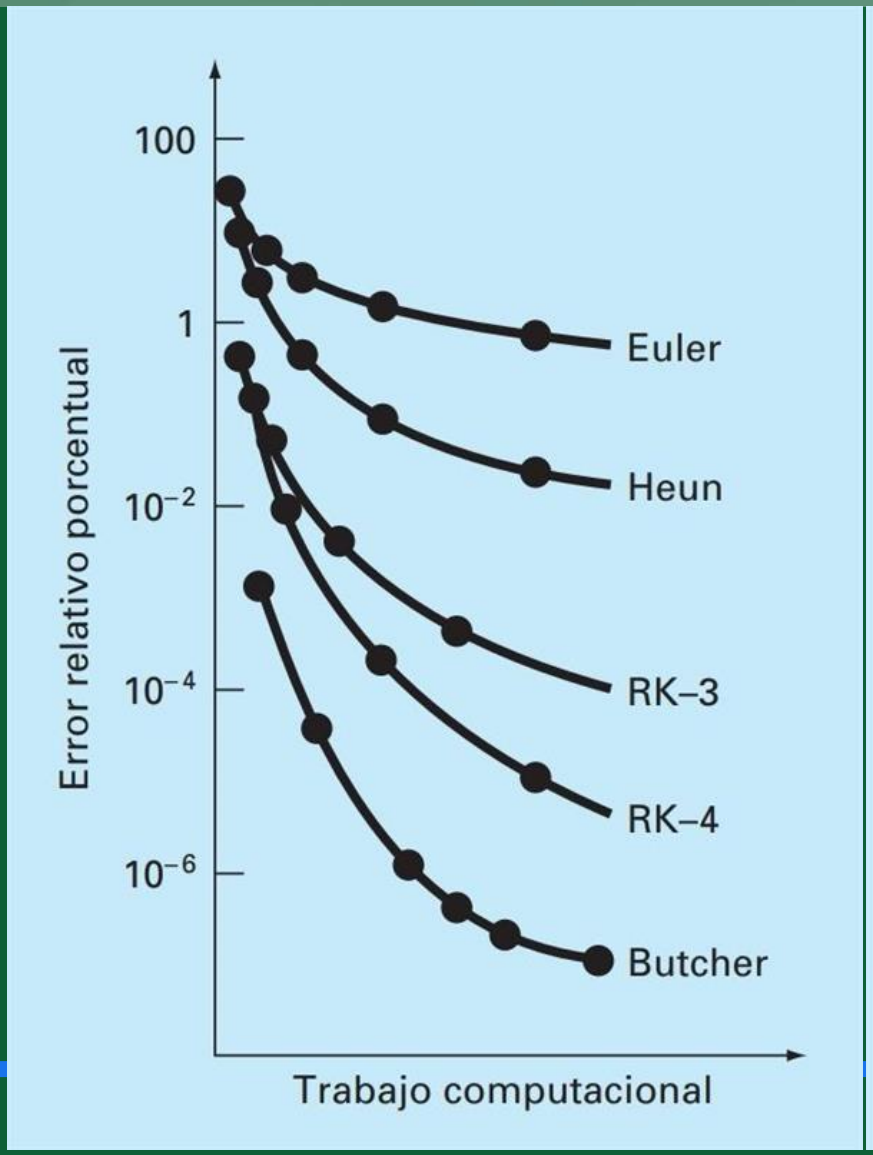

y
$y_{i+1} = y_i + \phi h$
Pendiente = ϕ
x_i
x_{i+1}
x
Tamaño de paso = h

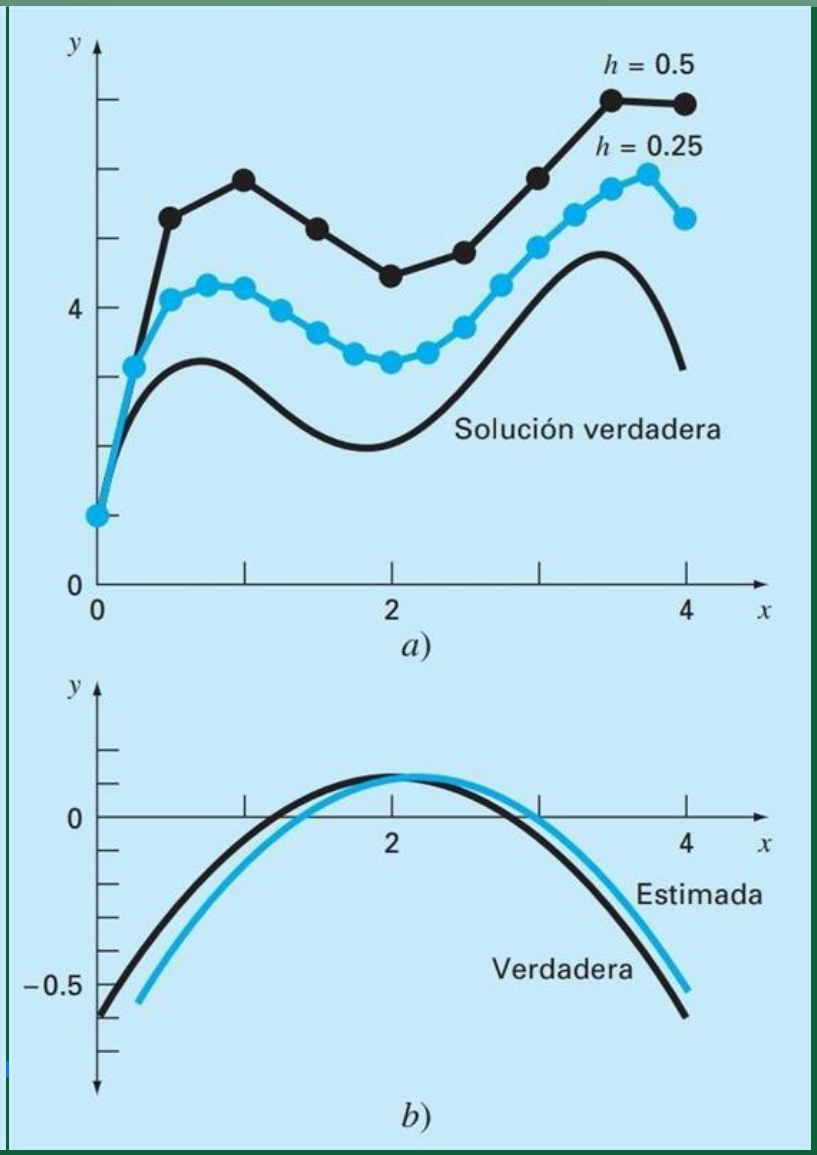

$R_n = \frac{y^{(n+1)}(\xi)}{(n+1)!} h^{(n+1)}$
y
Predicho
Verdadero
error
h
x_i
x_{i+1}
x

CLASE 6

Ing. Omar Cáceres

El efecto del tamaño del paso, así como también como se calcule la pendiente según el método escogido determinará la precisión de la aproximación así como el error es como se muestra en la grafica siguiente





Método de Euler

Es un procedimiento numérico para encontrar aproximaciones de soluciones de ecuaciones diferenciales ordinarias. Especialmente útil cuando no se puede encontrar la solución exacta.

Formula del Método de Euler

$$y_{n+1} = y_n + hf(x_n, y_n)$$

Donde;

y_{n+1} = El valor de la función en el siguiente paso

y_n = El valor de la función en el paso actual

h = El tamaño del paso

$f(x_n, y_n)$ = Derivada de la función en el paso actual

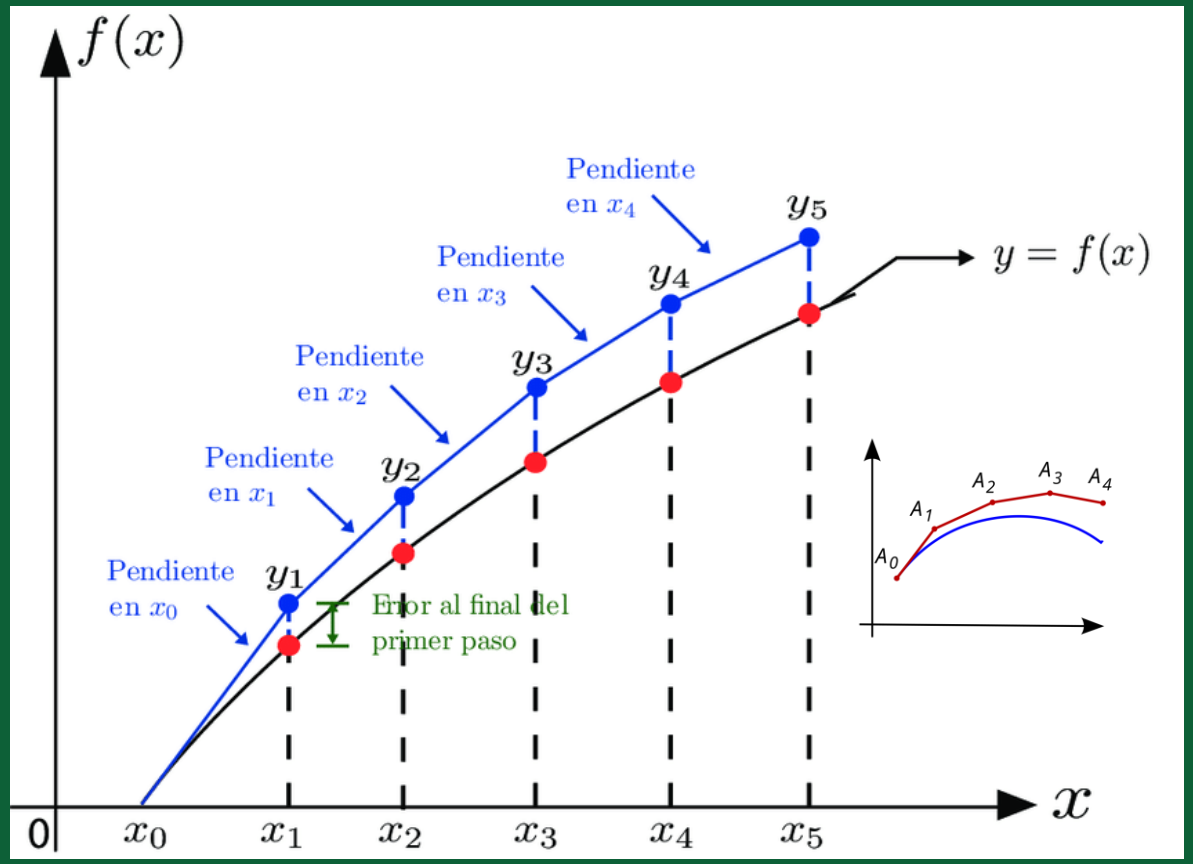
Se utiliza la pendiente de la función en el punto inicial para hacer una predicción del siguiente valor. Esta pendiente se calcula como: $f(x_n, y_n)$ donde f es la derivada de la función.

Segmento de línea recta

1. Se utiliza la pendiente de la función en el punto inicial para hacer una predicción del siguiente valor. Esta pendiente se calcula como: $f(x_n, y_n)$ donde f es la derivada de la función.

2. Segmento de línea recta: La predicción se realiza moviéndose a lo largo de un segmento de línea recta que sigue la pendiente inicial. Este segmento se extiende desde el punto inicial (x_n, y_n) hasta el punto predicho (x_{n+1}, y_{n+1}).

3. El nuevo punto: (x_{n+1}, y_{n+1}) se calcula utilizando la formula de Euler $y_{n+1} = y_n + hf(x_n, y_n)$, donde h es el tamaño del paso



Código para Euler

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

def euler(f, y0, t0, tf, h):
    t_values = np.arange(t0, tf + h, h)
    y_values = np.zeros(len(t_values))
    y_values[0] = y0
    for i in range(1, len(t_values)):
        y_values[i] = y_values[i - 1] + h * f(t_values[i - 1], y_values[i - 1])
    return t_values, y_values

 def f(t, y):
    return t+y

def solucion_particular(t):
    return -t-1+2*np.exp(t)

# Condiciones iniciales y parámetros
y0, t0, tf, h = 1, 0, 1, 0.1

# Soluciones
t_euler, y_euler = euler(f, y0, t0, tf, h)
y_real = solucion_particular(t_euler)

# Crear DataFrame
resultados = pd.DataFrame({
    'Tiempo': t_euler,
    'Euler': y_euler,
        'Solución Real': y_real
})

print(resultados)

# Graficar resultados
plt.plot(t_euler, y_euler, label='Euler')

plt.plot(t_euler, y_real, label='Solución Real')
plt.xlabel('Tiempo')
plt.ylabel('Valor de y')
plt.legend()
plt.title('Métodos de Euler, Euler Mejorado y Solución Real')
plt.show()
```

Método de Euler mejorado (Heun)

Se basa en la idea de mejorar la aproximación inicial obtenida con el método de Euler. Geométricamente, esto se puede entender como una corrección de la pendiente inicial.

1. Predicción inicial: En el método de Euler, se utiliza la pendiente del punto inicial para hacer una predicción de la siguiente posición. Esto se representa como una línea recta que sigue la pendiente inicial.
2. Corrección: En el método de Heun, se calcula una segunda pendiente en el punto predicho. Luego, se toma el promedio de la pendiente inicial y la pendiente predicha para obtener una mejor aproximación de la pendiente real en el intervalo.

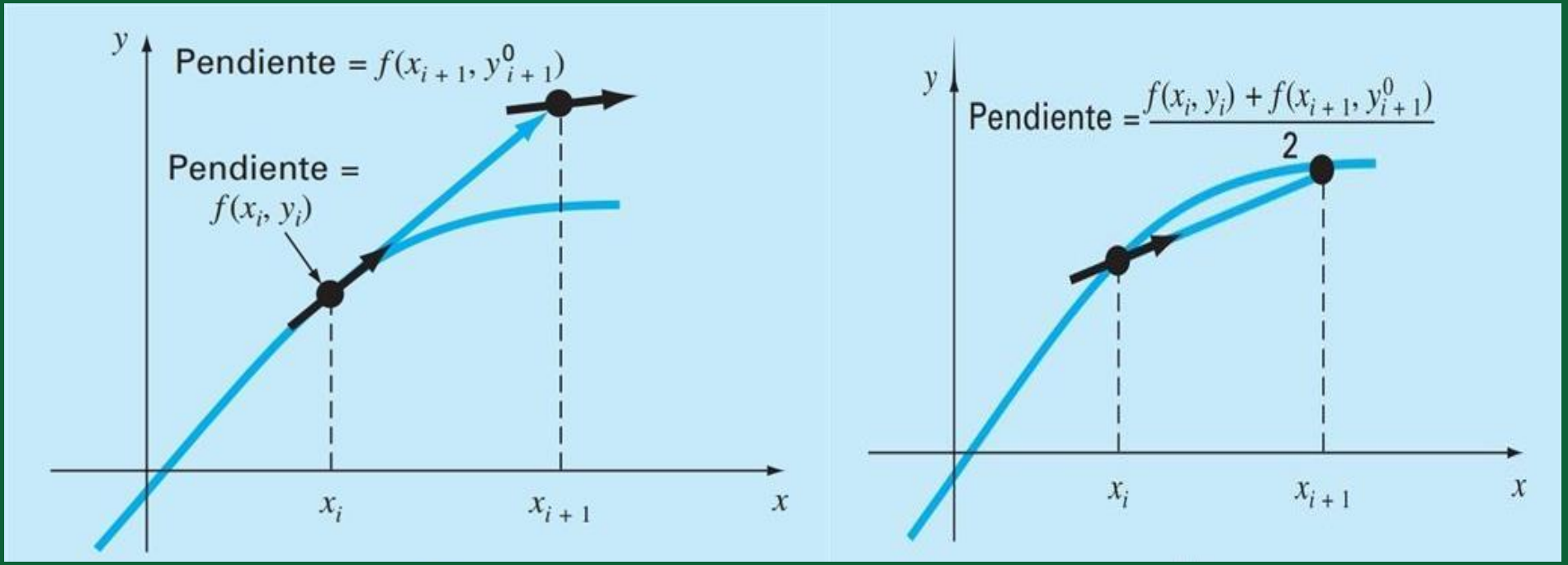
$$y_{n+1} = y_n + \frac{h}{2}\left(f(x_n, y_n) + f(x_{n+1}, y^*)\right)$$

y^* = Es el valor predicho usando Euler → $\boxed{y^* = y_n + hf(x_n, y_n)}$

CLASE 6

Ing. Omar Cáceres

Método de Euler mejorado (Heun)
Interpretación geométrica.



Código para Heun

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

def euler_mejorado(f, y0, t0, tf, h):
    """ Método de Euler mejorado (Heun). """
    t = np.arange(t0, tf + h, h)
    y = np.zeros_like(t)
    y[0] = y0

    for i in range(len(t) - 1):
        y_pred = y[i] + h * f(t[i], y[i])  # Predicción con Euler
        y[i + 1] = y[i] + h * (f(t[i], y[i]) + f(t[i] + h, y_pred)) / 2
  # Corrección

    return t, y

# Ecuación diferencial: dy/dt = y*sin(t)
def f(t, y):
    return 0.4 * t * y

# Solución exacta
def sol_exacta(y0, t):
    return (1 / np.exp(0.2)) * np.exp(0.2 * t**2)

# Parámetros
y0 , t0, tf, h= 1, 1, 2, 0.1     # Condiciones iniciales

# Cálculo de soluciones
t, y_eul_mej = euler_mejorado(f, y0, t0, tf, h)
y_real = sol_exacta(y0, t)

# Crear tabla resumen
tabla = pd.DataFrame({
    'Tiempo': t,
    'Exacta': y_real,
    'Euler Mejorado': y_eul_mej
})

print(tabla)

# Graficar resultados
plt.figure(figsize=(8, 5))
plt.plot(t, y_real, label="Solución Exacta", color="blue", linestyle="-", linewidth=2)
plt.plot(t, y_eul_mej, label="Euler Mejorado", color="green", linestyle="dashdot",
linewidth=2, marker="d")

plt.xlabel("Tiempo")
plt.ylabel("Solución y")
plt.title("Comparación de Métodos Numéricos para EDO")
plt.legend()
plt.grid()
plt.show()
```

CLASE 6

Ing. Omar Cáceres

Método de Runge Kutta 4

La interpretación geométrica del método de Runge-Kutta de cuarto orden (RK4) se basa en la idea de aproximar la solución de una ecuación diferencial mediante una serie de estimaciones de la pendiente en distintos puntos. En términos visuales, el método busca mejorar la precisión de la trayectoria de la solución al considerar varias pendientes intermedias en cada paso.
Imagina que quieres seguir una curva en un gráfico, pero solo puedes avanzar en pequeños pasos. En cada paso:

Algunos pasos son:

1_ Pendiente inicial k1: Se calcula la pendiente en el punto actual.
2. Primera corrección k2: Se estima la pendiente en un punto intermedio, avanzando la mitad del paso.
3. Segunda corrección k3: Se vuelve a estimar la pendiente en otro punto intermedio.
4. Corrección final k4: Se calcula la pendiente en el punto final del paso.

MODELADO Y SIMULACIÓN

Ecuaciones Diferenciales

Método de Runge Kutta 4

CLASE 6

Ing. Omar Cáceres

Cálculos de las pendientes

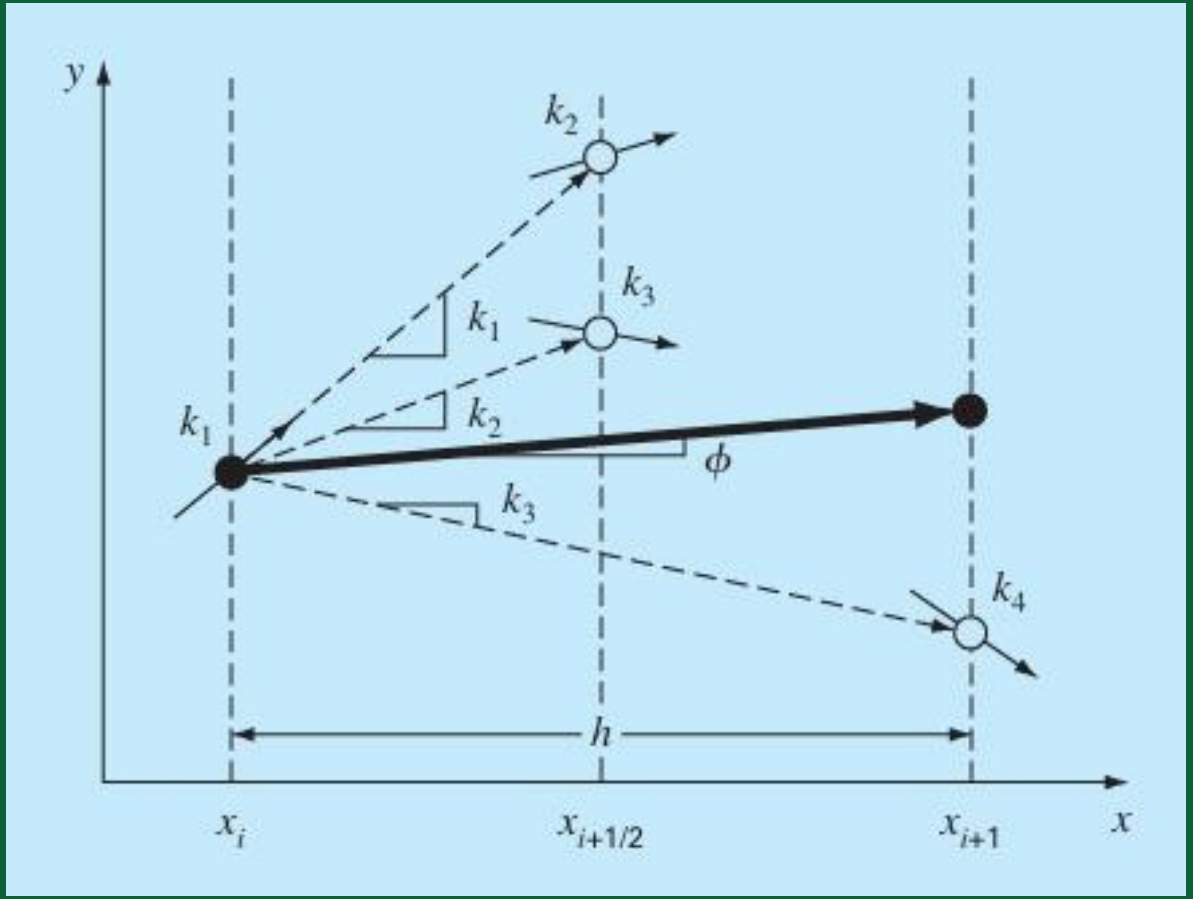
$$k_1 = f(x_n, y_n)$$

$$k_2 = f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1 h\right)$$

$$k_3 = f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2 h\right)$$

$$k_4 = f(x_n + h,\ y_n + k_3 h)$$

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$



MODELADO Y SIMULACIÓN
Ecuaciones Diferenciales
Método de Runge Kutta 4

CLASE 6

Ing. Omar Cáceres

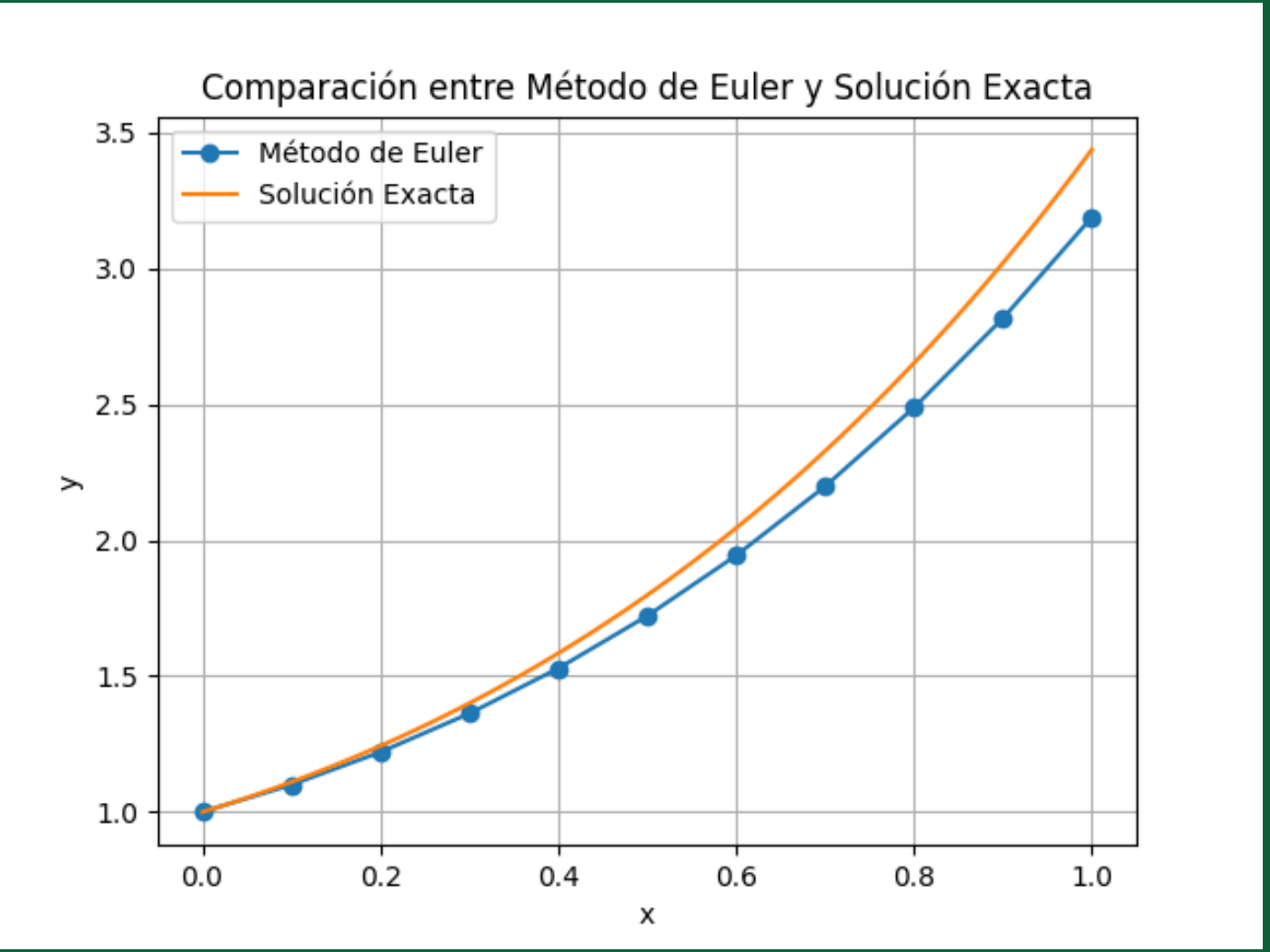
Código Runge Kutta 4

```
import numpy as np
from tabulate import tabulate

# Definimos la EDO: dy/dx = x + y
def f(x, y):
    return x + y

# Solución exacta para dy/dx = x + y, y(0)=1
def solucion_exacta(x):
    return 2*np.exp(x) - x - 1

# RK4 Modificado (h/2 dentro)
def rk4_modificado(f, x0, y0, h, pasos):
    x, y = x0, y0
   solucion = [(x, y)]
    for _ in range(pasos):
        k1 = f(x, y)
        k2 = f(x + h/2, y + (h/2)*k1)
        k3 = f(x + h/2, y + (h/2)*k2)
        k4 = f(x + h, y + h*k3)
        y += h * (k1 + 2*k2 + 2*k3 + k4) / 6
        x += h
        solucion.append((x, y))
    return solucion
 # Parámetros
x0, y0, h, pasos = 0.0, 1.0, 0.1, 10
```

```
# Calculamos todas las soluciones
sol_rk4_mod = rk4_modificado(f, x0, y0, h, pasos)
sol_exacta = [(x, solucion_exacta(x)) for x in np.arange(x0, x0 + (pasos+1)*h, h)]

# Preparamos datos para la tabla comparativa
tabla_comparativa = []
headers = ["x", "Exacta", "RK4 Modificado"]

for i in range(pasos + 1):
    x = x0 + i*h
    y_rk4m = sol_rk4_mod[i][1]
    y_exacta = sol_exacta[i][1]

    tabla_comparativa.append([
        f"{x:.1f}",

        f"{y_rk4m:.6f}",
        f"{y_exacta:.6f}"

    ])

# Imprimimos tabla
print(tabulate(tabla_comparativa, headers=headers, tablefmt="grid"))
```

dy/dx=x+y

n	x	Euler	Exacta	Error
0	0.0000	1.0000	1.0000	0.0000
1	0.1000	1.1000	1.1103	0.0103
2	0.2000	1.2200	1.2428	0.0228
3	0.3000	1.3620	1.3997	0.0377
4	0.4000	1.5282	1.5836	0.0554
5	0.5000	1.7210	1.7974	0.0764
6	0.6000	1.9431	2.0442	0.1011
7	0.7000	2.1974	2.3275	0.1301
8	0.8000	2.4872	2.6511	0.1639
9	0.9000	2.8159	3.0192	0.2033
10	1.0000	3.1875	3.4366	0.2491

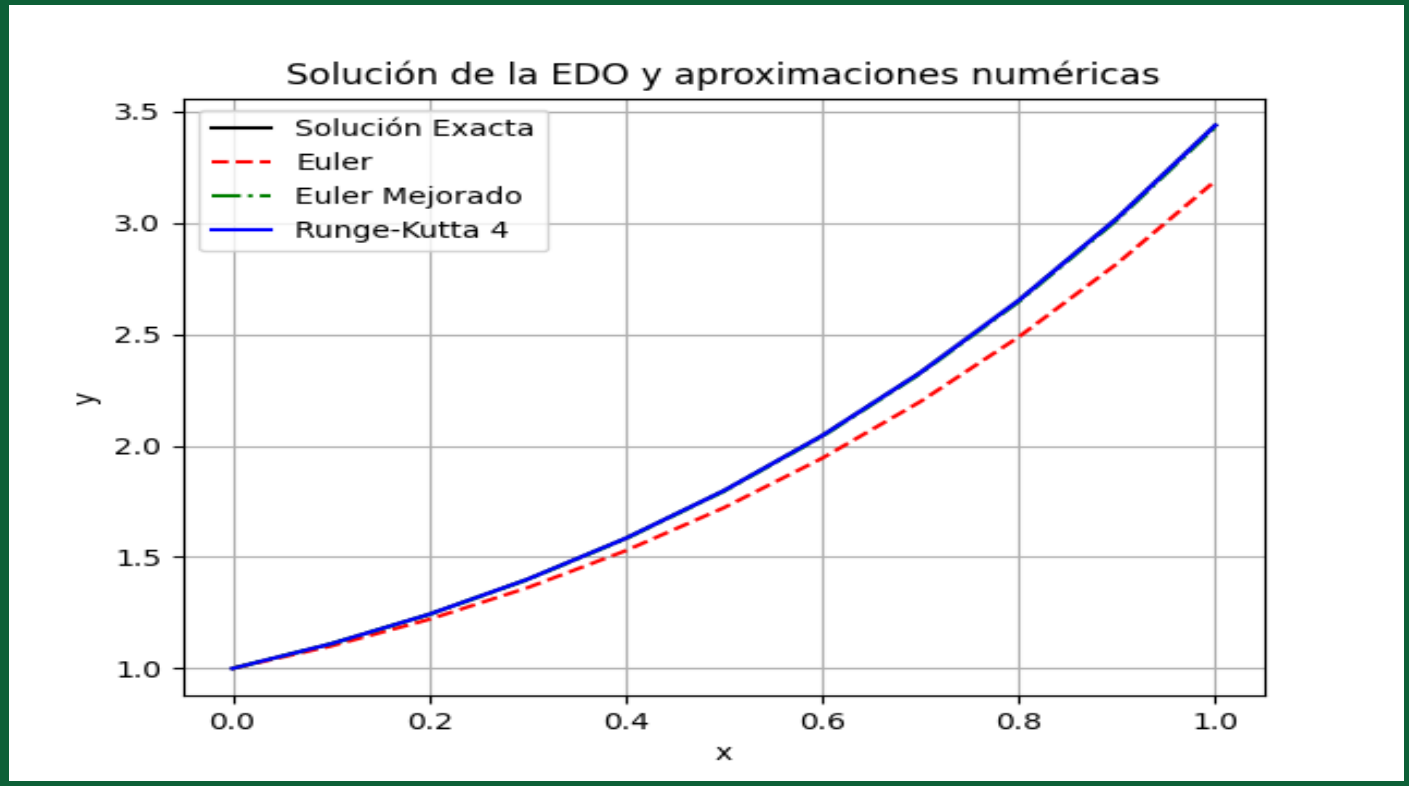


MODELADO Y SIMULACIÓN
Ecuaciones Diferenciales

CLASE 6

Ing. Omar Cáceres

n	x	Sol Exacta	Euler	Euler M	Runge-K4
0	0.0	1.000000	1.000000	1.000000	1.000000
1	0.1	1.110342	1.100000	1.110000	1.110342
2	0.2	1.242806	1.220000	1.242050	1.242805
3	0.3	1.399718	1.362000	1.398465	1.399717
4	0.4	1.583649	1.528200	1.581804	1.583648
5	0.5	1.797443	1.721020	1.794894	1.797441
6	0.6	2.044238	1.943122	2.040857	2.044236
7	0.7	2.327505	2.197434	2.323147	2.327503
8	0.8	2.651082	2.487178	2.645578	2.651079
9	0.9	3.019206	2.815895	3.012364	3.019203
10	1.0	3.436564	3.187485	3.428162	3.436559



MODELADO Y SIMULACIÓN
Ecuaciones Diferenciales

CLASE 6

Ing. Omar Cáceres

t	Exacta	Euler	Euler M	Runge-Kutta
0	1.000000	1	1	1
1	3.436564	2	3	3
2	11.77	5	9	10
3	36.17	12	26	31
4	104.19	27	70	89
5	290.82	58	181	248
6	799.85	121	460	680
7	2185.26	248	1159	1852
8	5952.91	503	2908	5028
9	16196.16	1014	7282	13631
10	44041.93	2037	18219	36933