

UNIVERSIDAD DE GUANAJUATO

DIVISIÓN DE CIENCIAS E INGENIERÍAS

INGENIERÍA QUÍMICA SUSTENTABLE

MÉTODOS NUMÉRICOS

GRUPO: A-I

CLAVE: IILI06085

ALUMNA:

ANA ISABEL ESQUIVEL CASTRO

PROYECTO FINAL

**“SOLUCIÓN DE ECUACIONES DIFERENCIALES
ORDINARIAS”**

FECHA DE ENTREGA: 28/11/2025



Solución de ecuaciones diferenciales ordinarias

Pseudocódigo

Nota: Hasta debajo de la conclusión está el diagrama del proceso.

1. Iniciar programa en C
 - Incluir librerías: stdio.h, math.h
2. Definir constantes:
 - gravedad = -9.81
 - paso_tiempo = 0.1
 - iteraciones_max = 20
 - dimension_sistema = 6
3. Declarar variables:
 - estado_actual[6] (x, y, z, vx, vy, vz)
 - estado_temporal[6]
 - derivadas[6]
 - k1[6], k2[6], k3[6], k4[6]
 - t (tiempo actual)
 - h (tamaño de paso)
 - i, j (contadores)
4. Configurar condiciones iniciales:
 - t = 0.0
 - h = paso_tiempo
 - estado_actual = [0.0, 0.0, 10.0, 5.0, 3.0, 2.0]
5. Definir función derivadas_sistema:
 - Entradas: t, estado[6]
 - Salidas: deriv[6]
 - Proceso:
 - deriv[0] = estado[3] // $dx/dt = vx$
 - deriv[1] = estado[4] // $dy/dt = vy$
 - deriv[2] = estado[5] // $dz/dt = vz$
 - deriv[3] = 0 // $dvx/dt = 0$
 - deriv[4] = 0 // $dvy/dt = 0$
 - deriv[5] = gravedad // $dvz/dt = -9.81$

6. Seleccionar método numérico (euler, rk2, rk4)

7. Ejecutar simulación principal:
para i = 0 hasta iteraciones_max hacer:

según método_seleccionado hacer:

caso euler:

derivadas_sistema(t, estado_actual, derivadas)
para j = 0 hasta 5:
estado_actual[j] += h * derivadas[j]

caso rk2:

derivadas_sistema(t, estado_actual, k1)
para j = 0 hasta 5:
estado_temporal[j] = estado_actual[j] + h * k1[j]
derivadas_sistema(t + h, estado_temporal, k2)
para j = 0 hasta 5:
estado_actual[j] += h * 0.5 * (k1[j] + k2[j])

caso rk4:

derivadas_sistema(t, estado_actual, k1)
para j = 0 hasta 5:
estado_temporal[j] = estado_actual[j] + 0.5 * h * k1[j]
derivadas_sistema(t + h/2, estado_temporal, k2)
para j = 0 hasta 5:
estado_temporal[j] = estado_actual[j] + 0.5 * h * k2[j]
derivadas_sistema(t + h/2, estado_temporal, k3)
para j = 0 hasta 5:

```

    estado_temporal[j] =
estado_actual[j] + h * k3[j]
    derivadas_sistema(t + h,
estado_temporal, k4)
    para j = 0 hasta 5:
        estado_actual[j] += (h/6) * (k1[j]
+ 2*k2[j] + 2*k3[j] + k4[j])

```

actualizar tiempo: $t = t + h$

mostrar resultados:

```

imprimir "t=%.1f: Posición(%.2f,
%.2f, %.2f) Velocidad(%.2f, %.2f, %.2f)"

```

verificar condiciones físicas:

```

si estado_actual[2] <= 0 entonces:
    imprimir "¡La partícula tocó el
suelo!"

```

terminar simulación

8. Mostrar resumen final:

```

imprimir "Simulación completada"
imprimir "Tiempo final: t = %.2f"
imprimir "Posición final: (%.2f, %.2f,
%.2f)"
imprimir "Velocidad final: (%.2f, %.2f,
%.2f)"

```

9. Fin

Resultados

Para una partícula 3D bajo una fuerza como lo es la gravedad, necesitamos las siguientes ecuaciones de movimiento:

- ❖ $dx/dt = vx$ (velocidad en x)
- ❖ $dy/dt = vy$ (velocidad en y)
- ❖ $dz/dt = vz$ (velocidad en z)
- ❖ $dvx/dt = Fx/m$ (aceleración en x)
- ❖ $dvy/dt = Fy/m$ (aceleración en y)
- ❖ $dvz/dt = Fz/m$ (aceleración en z)

Podemos modificar la función de derivadas que se ha mostrado en la tarea y obtener:

```

void calcular_fuerzas(double t, double
pos[3], double vel[3], double fuerza[3]) {
    fuerza[0] = 0; // Sin fuerza en x
    fuerza[1] = 0; // Sin fuerza en y
    fuerza[2] = -9.81; // Gravedad en z (m/s²)
}

```

```

void derivadas_sistema(double t, double
estado[6], double deriv[6]) {
    // estado[0]=x, [1]=y, [2]=z, [3]=vx, [4]=vy,
[5]=vz
    // deriv[0]=dx/dt, [1]=dy/dt, [2]=dz/dt,
[3]=dvx/dt, [4]=dvy/dt, [5]=dvz/dt
}

```

```

double fuerza[3];
calcular_fuerzas(t, estado, &estado[3],
fuerza);

```

```

deriv[0] = estado[3]; // dx/dt = vx
deriv[1] = estado[4]; // dy/dt = vy
deriv[2] = estado[5]; // dz/dt = vz
deriv[3] = fuerza[0]; // dvx/dt = Fx/m (m=1)
deriv[4] = fuerza[1]; // dvy/dt = Fy/m
deriv[5] = fuerza[2]; // dvz/dt = Fz/m
}

```

En seguida, se puede implementar el método de Euler:

```
#include <stdio.h>
```

```

void derivadas_sistema(double t, double
estado[6], double deriv[6]) {
    double fuerza[3] = {0, 0, -9.81}; //
Gravedad solo en z
}

```

```

deriv[0] = estado[3]; // dx/dt = vx
deriv[1] = estado[4]; // dy/dt = vy
deriv[2] = estado[5]; // dz/dt = vz
deriv[3] = fuerza[0]; // dvx/dt = Fx
deriv[4] = fuerza[1]; // dvy/dt = Fy
deriv[5] = fuerza[2]; // dvz/dt = Fz
}

```

```
int main() {
double t = 0.0, h = 0.1; // Tiempo y paso
double estado[6] = {0, 0, 10, 5, 3, 2}; //
[x,y,z,vx,vy,vz]
double deriv[6]; // Derivadas

printf("Movimiento 3D - Método de
Euler\n");

for(int i = 0; i < 20; i++) {
derivadas_sistema(t, estado, deriv); //
Calcular derivadas

for(int j = 0; j < 6; j++) {
estado[j] = estado[j] + h * deriv[j]; // Euler
para cada variable
}
t = t + h; // Avanzar tiempo

printf("t=%.1f: x=%.2f, y=%.2f, z=%.2f\n",
t, estado[0], estado[1], estado[2]);

if(estado[2] <= 0) { // Si toca el suelo
printf("¡La partícula tocó el suelo!\n");
break;
}
}

return 0;
}
```

Y el resultado arroja:

t=0.10	x=1.1000	y=-0.1100
t=0.20	x=1.1979	y=-0.2420
t=0.30	x=1.2911	y=-0.3980
t=0.40	x=1.3764	y=-0.5798
t=0.50	x=1.4503	y=-0.7892
t=0.60	x=1.5085	y=-1.0276
t=0.70	x=1.5463	y=-1.2963
t=0.80	x=1.5583	y=-1.5960
t=0.90	x=1.5386	y=-1.9270
t=1.00	x=1.4805	y=-2.2890

Procedemos a continuar con Runge Kutta de 4° Orden:

```
#include <stdio.h>
```

```
void derivadas_sistema(double t, double
estado[6], double deriv[6]) {
double fuerza[3] = {0, 0, -9.81};
```

```
deriv[0] = estado[3];
deriv[1] = estado[4];
deriv[2] = estado[5];
deriv[3] = fuerza[0];
deriv[4] = fuerza[1];
deriv[5] = fuerza[2];
}
```

```
int main() {
double t = 0.0, h = 0.1;
double estado[6] = {0, 0, 10, 5, 3, 2}; //
Condiciones iniciales
double k1[6], k2[6], k3[6], k4[6];
double temp[6];
```

```
printf("Movimiento 3D - Runge-Kutta 4to
Orden\n");
```

```
for(int i = 0; i < 20; i++) {
// Paso 1: k1
derivadas_sistema(t, estado, k1);
```

```
// Paso 2: k2
for(int j = 0; j < 6; j++) {
    temp[j] = estado[j] + 0.5 * h * k1[j];
}
derivadas_sistema(t + h/2, temp, k2);

// Paso 3: k3
for(int j = 0; j < 6; j++) {
    temp[j] = estado[j] + 0.5 * h * k2[j];
}
derivadas_sistema(t + h/2, temp, k3);

// Paso 4: k4
for(int j = 0; j < 6; j++) {
    temp[j] = estado[j] + h * k3[j];
}
derivadas_sistema(t + h, temp, k4);

// Combinar resultados
for(int j = 0; j < 6; j++) {
    estado[j] += (h/6.0) * (k1[j] + 2*k2[j] + 2*k3[j] + k4[j]);
}
t += h;

printf("t=%.1f: x=%.2f, y=%.2f, z=%.2f\n",
t, estado[0], estado[1], estado[2]);

if(estado[2] <= 0) {
    printf("¡La partícula tocó el suelo!\n");
    break;
}

return 0;
}
```

Y se obtiene:

```
t=0.1: x=0.50, y=0.30, z=10.20
t=0.2: x=1.00, y=0.60, z=10.30
t=0.3: x=1.50, y=0.90, z=10.31
t=0.4: x=2.00, y=1.20, z=10.21
t=0.5: x=2.50, y=1.50, z=10.02
t=0.6: x=3.00, y=1.80, z=9.73
t=0.7: x=3.50, y=2.10, z=9.34
t=0.8: x=4.00, y=2.40, z=8.85
t=0.9: x=4.50, y=2.70, z=8.27
t=1.0: x=5.00, y=3.00, z=7.59
t=1.1: x=5.50, y=3.30, z=6.80
t=1.2: x=6.00, y=3.60, z=5.93
t=1.3: x=6.50, y=3.90, z=4.95
t=1.4: x=7.00, y=4.20, z=3.87
t=1.5: x=7.50, y=4.50, z=2.70
t=1.6: x=8.00, y=4.80, z=1.43
t=1.7: x=8.50, y=5.10, z=0.06
t=1.8: x=9.00, y=5.40, z=-1.41
```

Donde la partícula ya ha tocado el suelo.

Conclusión

En este proyecto se implementaron tres métodos numéricos para resolver ecuaciones diferenciales ordinarias: Euler, Runge-Kutta de segundo orden y Runge-Kutta de cuarto orden, los cuales mostraron ser efectivos para aproximar soluciones a sistemas de ecuaciones diferenciales, mostrando diferentes niveles de precisión y complejidad computacional. La extensión a sistemas 3D permitió modelar el movimiento de una partícula bajo fuerzas externas, específicamente la gravedad, haciendo que aplicar estos métodos sea viable en problemas físicos que son reales.

Simulación de mov. de Partícula ♥

