Matematicas para Ingeniería II

UNIDAD 3

Alumnos:

Andrés García Leyva

Angel Manuel Guzmán Hoil

Juan Carlos López Can

Profesor:

Gustavo Adolfo Fajardo Pulido

IDYGS81

Investigación y solución por Software Metodos Numericos Ecuaciones diferenciales

Índice

Introducción

Los métodos numéricos se han convertido en una herramienta esencial para resolver un amplio abanico de problemas científicos y técnicos que involucran ecuaciones diferenciales y sistemas de ecuaciones no lineales.

En ingeniería es común enfrentar problemas modelados por ecuaciones diferenciales ordinarias (EDO) cuyas soluciones analíticas no son fáciles de obtener. Para estos casos, se emplean métodos numéricos de integración que aproximan la solución paso a paso. Tres métodos ampliamente utilizados son el método de Euler, el método de Euler mejorado (también conocido como método de Heun) y el método de Runge-Kutta de cuarto orden. Cada uno ofrece un equilibrio distinto entre simplicidad, precisión y estabilidad, por lo que es importante comprender sus fundamentos teóricos, su análisis de error y sus aplicaciones prácticas en diversas ramas de la ingeniería. A continuación, se presenta una investigación detallada de cada método, incluyendo su deducción matemática, comportamiento del error, criterios de selección y ejemplos de aplicación en contextos ingenieriles.

Método de Euler.

El **método de Euler** es el procedimiento más sencillo de integración numérica para resolver un problema de valor inicial en una EDO. Este método, propuesto por Leonhard Euler en el siglo XVIII [1], se basa en utilizar la derivada (pendiente) conocida en un punto para extrapolar linealmente el valor de la solución en el siguiente punto. En esencia, si se tiene una EDO de la forma y′(t)=f(t,y(t)) [2] con condición inicial y(t0)=y0, el método de Euler aproxima la solución dividendo el intervalo de integración en pasos pequeños de tamaño h. En cada paso, asume que la pendiente de la solución se mantiene constante (igual a la pendiente inicial) a lo largo del intervalo. Matemáticamente, la fórmula de recurrencia del método de Euler es:

$$y_{n+1} = y_n + h\, f(t_n, y_n)$$

donde y_n es la aproximación numérica de y(t_n,). Esta fórmula se deduce al truncar la serie de Taylor de y(t) después del término lineal. Partiendo de la expansión de Taylor:

$$y(t_n + h) = y(t_n) + y'(t_n)\, h + \frac{y''(\xi)}{2} h^2,$$

Para algún $\xi \in [t_n, t_n + 1]$, y recordando que $y'(t_n) = f(t_n, y(t_n))$, si se omiten los términos de orden h^2 o superiores se obtiene $y(t_n + h) \approx y(t_n) + f\big(t_n, y(t_n)\big)h$, al definir $y_{n+1} \approx y(t_n + h)$ y $y_n \approx y(t_n)$, se llega a la fórmula del método de Euler anterior. En términos geométricos, el método de Euler utiliza la pendiente de la curva solución en t_n para proyectar la solución hacia t_{n+1}, formando así un segmento lineal. Iterando este proceso, la solución numérica se construye como una *poligonal* que se espera siga aproximadamente a la curva verdadera (ver Ilustración **1**).

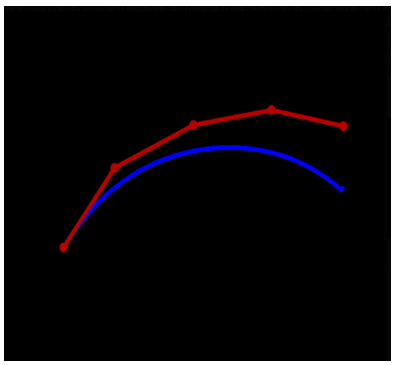


Ilustración 1

Ilustración del método de Euler, donde la curva exacta (azul) se aproxima mediante tramos lineales entre puntos calculados (rojo). Cada segmento lineal se construye usando la pendiente de la curva en el extremo izquierdo del tramo. A medida que se avanza en pasos h sucesivos, la poligonal resultante A_1, A_2, A_3,... se acerca a la solución real de la EDO. En general, reducir el tamaño del paso h mejora la fidelidad de esta aproximación, aunque

esto aumenta el costo computacional. El método de Euler, al ser explícito (no requiere resolver ecuaciones algebraicas en cada paso), es sencillo de implementar y sirve como base para métodos más avanzados

No obstante, su simplicidad conlleva limitaciones en estabilidad y exactitud, como se analiza a continuación.

Análisis del Estabilidad y Convergencia.

El método de Euler es un método consistente y convergente de primer orden, lo que significa que el error global de truncamiento decrece proporcionalmente a h [3] conforme $h \to 0$ En particular, el error local por paso es del orden $O(h^2)$ (proporcional a h^2) y el error global acumulado tras muchos pasos es del orden $O(h)$ esto implica que, si se reduce a la mitad el tamaño de paso, el error global aproximadamente se reduce a la mitad. Aunque la convergencia está garantizada para funciones suficientemente regulares (por el teorema de convergencia de métodos de un paso), la estabilidad del método de Euler depende fuertemente de h. En problemas donde la solución verdadera decae rápidamente (por ejemplo, EDOs *rígidas* con autovalores negativos grandes en valor absoluto), el método de Euler requiere pasos muy pequeños para mantenerse estable.

Análisis del Error: Precisión y Ventajas/Desventajas.

Debido a su naturaleza de primer orden, el método de Euler presenta **errores relativamente grandes** si se comparan con métodos de orden superior usando el mismo tamaño de paso. Su error global O(h) implica que para lograr alta precisión se necesitan pasos muy pequeños, con el consiguiente aumento en el número de iteraciones. La principal ventaja de Euler es su **simplicidad**: requiere una sola evaluación de la función f(t,y) por paso y la fórmula es fácil de implementar manualmente o en programación. Esto lo hace atractivo para cálculos rápidos, estimaciones iniciales o sistemas con recursos limitados (por ejemplo, para programación en microcontroladores donde se valora la sencillez). Otra ventaja es que, al ser explícito, no requiere resolver sistemas de ecuaciones en cada paso. Por otro lado, sus **desventajas** incluyen la acumulación rápida de error y posibles problemas de estabilidad para pasos grandes. En comparación con métodos más avanzados, Euler es menos preciso y puede divergir si no se escoge h adecuadamente. En ingeniería, el método de Euler se selecciona generalmente solo cuando la prioridad es la **facilidad de implementación** o cuando se desea un método simple para entender la dinámica cualitativa del sistema, sabiendo que habrá que compensar con un paso suficientemente pequeño para obtener precisión aceptable. Si el problema requiere mayor exactitud o si Euler presenta inestabilidad, se suele optar por métodos de orden superior (como Runge-Kutta) o métodos implícitos

Aplicaciones en Ingeniería.

A pesar de sus limitaciones, el método de Euler se ha utilizado históricamente en múltiples campos de la ingeniería para **simular sistemas dinámicos** en forma aproximada. Por ejemplo, en ingeniería eléctrica se puede aplicar Euler para estimar la respuesta temporal de un circuito RC o RL simple, discretizando la ecuación diferencial que rige la corriente o el voltaje en el circuito. En ingeniería mecánica, antes de la disponibilidad de computadores potentes, se utilizó Euler para obtener trayectorias aproximadas de movimiento bajo la segunda ley de Newton (por ejemplo, la caída libre con resistencia del aire, o la oscilación de un péndulo para pequeños intervalos de tiempo). En ingeniería civil, problemas de flujo de aguas subterráneas o difusión de calor de forma simplificada se abordaron con integraciones tipo Euler a paso pequeño. Sin embargo, en aplicaciones modernas el método de Euler suele reservarse para demostraciones conceptuales o casos donde la rapidez es primordial sobre la precisión. Muchos problemas de la física y la ingeniería (vibraciones estructurales, dinámica de sistemas, circuitos, etc.) conducen naturalmente a EDO, y Euler ofrece una primera aproximación rápida a sus soluciones. Aun cuando hoy en día es más común emplear métodos más precisos como Runge-Kutta en simulaciones, el método de Euler sigue siendo útil para **validar modelos de forma simple**, estimar cotas iniciales de resultados y enseñar los fundamentos de la integración numérica.

En la práctica, al comparar Euler con otros métodos numéricos, se encuentra que **casi siempre** es el menos preciso. Por ejemplo, frente a un método de **Euler mejorado** o un **Runge-Kutta de 4º orden**, Euler requerirá órdenes de magnitud más pasos para alcanzar la misma precisión [2] [4]. No obstante, su valor pedagógico y su facilidad de implementación mantienen al método vigente en entornos donde estas características son apreciadas.

Método de Euler Mejorado.

El **método de Euler mejorado**, también conocido como **método de Heun** o método del trapecio explícito, es una mejora sobre el Euler simple que busca aumentar la exactitud utilizando un esquema predictor-corrector en cada paso [5]. Al igual que Euler, parte de una EDO $y'=f(t,y)$ con condición inicial $y(t0)=y0$ y divide el dominio en pasos h. La idea central es: primero **predice** el valor en el extremo del intervalo usando un paso de Euler, y luego utiliza la pendiente en ese extremo predicho para **corregir** la estimación, tomando en promedio las pendientes al inicio y al final del intervalo. En términos geométricos, mientras el método de Euler usa solo la pendiente inicial (recta tangente en el punto izquierdo) para extrapolar, el Euler mejorado considera también la pendiente en el punto derecho del intervalo y toma un promedio, lo que equivale a aproximar

la solución en el intervalo por el segmento tangente medio (trapecio) en vez de un simple rectángulo.

Formalmente, si y_n es la aproximación en t_n el método de Euler mejorado realiza dos etapas por paso:

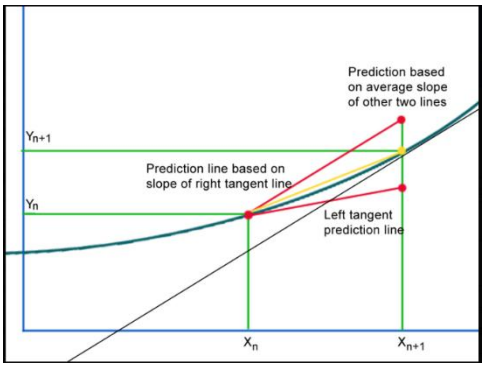
1. **Predicción (Paso de Euler):** Calcula un valor intermedio y_{n+1} usando Euler explícito estándar: $\tilde{y}_{n+1} = y_n + hf(t_n, y_n)$ Este es un estimado provisional de$y$ $(t_n + 1)$ usando la pendiente en t_n

2. **Corrección:** Calcula la pendiente en el extremo derecho usando $\tilde{y}_{n+1} = y_n + hf(t_n, y_n)$ Luego promedia la pendiente inicial $f(t_n, y_n)$ y la pendiente en el extremo:

$$y_{n+1} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, \tilde{y}_{n+1})\right).$$

   Esta fórmula es equivalente a aplicar la **regla del trapecio** para aproximar la integral de f en el intervalo $[t_n, t_{n+1}]$

En esencia, el método de Euler mejorado sigue considerando la solución y(t) aproximadamente lineal en cada subintervalo, pero utiliza una pendiente más refinada que tiene en cuenta información de ambos extremos, en lugar de solo el inicial. La Figura 2 esquematiza este proceso: primero se traza una línea tangente en t_n (pendiente roja izquierda) obteniendo una predicción, luego se evalúa la pendiente en t_{n+1} (pendiente roja derecha) y finalmente se corrige la predicción con la pendiente promedio (línea amarilla que conecta t_n y t_{n+1}). Este enfoque mitigará el error de Euler simple, especialmente si la pendiente cambia casi linealmente en el intervalo.



Esquema del método de Euler mejorado (Heun). Primero se realiza una predicción lineal usando la pendiente en t_n (recta tangente izquierda en rojo), luego otra usando la pendiente en t_{n+1} estimado (recta tangente derecha en rojo). Finalmente, la pendiente promedio de ambas (línea amarilla) define la aproximación corregida y_{n+1} (punto amarillo), logrando una mejor estimación de la verdadera curva (verde oscuro) en [t_n y t_{n+1}]. Al promediar las dos pendientes, el método obtiene una aproximación del valor futuro que suele estar más cerca de la solución real que la predicción de Euler simple. El método de Heun requiere por tanto **dos evaluaciones** de la función f por paso (una en el inicio y otra en el final del subintervalo), a diferencia de la única evaluación que necesita Euler. Esta inversión computacional adicional rinde frutos en la precisión, como veremos en

el análisis de error. Matemáticamente, el método de Euler mejorado pertenece a la familia de métodos de Runge-Kutta de segundo orden (RK2 explícito), ya que coincide con un método de orden 2 en la clasificación general de Runge-Kutta. Su deducción puede realizarse igualando coeficientes con la expansión en serie de Taylor hasta ($O(h^2)$) [2] [3], resultando en los coeficientes 1/2 para cada pendiente (inicio y fin). Aunque no profundizaremos en la derivación algebraica, es importante destacar que este método está diseñado para cancelar el término de error principal que quedaba en Euler simple, elevando así el orden de exactitud.

Análisis de Estabilidad y Convergencia.

El método de Euler mejorado es **convergente de orden 2**, lo que implica que su error global decrece aproximadamente con O(h^2) a medida que h disminuye. El error local por cada paso es del orden O(h^3) en contraste con el O(h^2) de Euler simple. En otras palabras, por cada paso, la corrección mediante el promedio de pendientes elimina el principal término de error que tenía el método de Euler. Esta mejora en orden de convergencia significa que, para un h pequeño, el error de Euler mejorado será mucho menor que el de Euler estándar con el mismo paso. Por ejemplo, si h se reduce a la mitad, el error global se reduce aproximadamente a una cuarta parte (ya que escala con h^2).

Análisis del Error: Precisión y Comparativa

El principal atractivo del método de Euler mejorado es su **mayor precisión** en comparación con Euler estándar sin un incremento desmesurado de complejidad. Al ser de segundo orden, para un mismo tamaño de paso xh, el error del Euler mejorado será mucho menor. Esto significa que en muchas aplicaciones podemos usar un paso relativamente mayor y aun así obtener resultados precisos, donde Euler simple habría requerido un paso quizá dos veces más pequeño (o más) para lograr igual exactitud. En cuanto a **ventajas**, Heun comparte la simplicidad conceptual de Euler (su formulación es entendible geométricamente como un promedio de pendientes) y sigue siendo relativamente fácil de implementar. Requiere el doble de evaluaciones de f(t,y) por paso, pero esto sigue siendo computacionalmente barato en muchos casos, especialmente comparado con métodos de orden aún más alto. También proporciona una forma sencilla de estimar el error: la diferencia entre la predicción $\tilde{y}_{n+1}$ y la corrección y_{n+1} nos da una indicación de la no linealidad en el intervalo, lo cual puede servir como estimador de error local (aunque no es un método de control de paso formal, da una idea).

Como **desventajas**, aunque mejora a Euler, el método sigue siendo explícito y de un solo paso, por lo que puede fallar en problemas muy rígidos a menos que h sea extremadamente pequeño. Además, con solo dos puntos de evaluación, puede no capturar comportamientos muy complejos dentro del intervalo si f(t,y) varía de forma no aproximadamente lineal. En tales casos, métodos de orden

más alto (como el clásico Runge-Kutta de 4º orden) brindarían aún mayor exactitud con un costo computacional mayor pero razonable. En resumen, el Euler mejorado ocupa un término medio: más preciso, pero ligeramente más costoso que Euler, menos preciso, pero más eficiente (menos evaluaciones) que un Runge-Kutta de orden superior. En ingeniería, un criterio de selección típico es usar Euler mejorado cuando se desea un aumento significativo de precisión sobre Euler, pero manteniendo el algoritmo sencillo y suficientemente rápido. Por ejemplo, en simulaciones en tiempo real donde Euler no es suficiente en exactitud, pero un RK4 completo sería demasiado lento, Heun puede ser una solución intermedia.

Aplicaciones en Ingeniería

El método de Euler mejorado encuentra aplicaciones en ingeniería en la modelación de fenómenos dinámicos donde se requiere cierto compromiso entre rapidez y precisión. Por ejemplo, en ingeniería eléctrica, se utiliza en simulaciones de circuitos RLC sencillos o sistemas de control discretizados, donde un paso de integración moderado puede ser viable. En ingeniería mecánica, Heun puede emplearse para simular la respuesta de un amortiguador y resorte (sistema masa-resorte simple) en situaciones donde se desea capturar con más exactitud la evolución que la que daría Euler: la fuerza variable del resorte a lo largo del desplazamiento se integra mejor con el promedio de pendientes. De hecho, cualquier sistema que responda a una ecuación diferencial de primer orden (o que se haya descompuesto en un sistema de primer orden) es candidato para simular con Euler mejorado si se busca más precisión. Algunos ejemplos incluyen: el cálculo de la velocidad de un proyectil con resistencia del aire moderada (donde la aceleración cambia con la velocidad), simulación básica de un motor eléctrico de corriente continua (donde la inductancia introduce una EDO para la corriente), o el modelado de la cinética de una reacción química simple. En estos casos, comparado con Euler, el método de Heun produce resultados más cercanos a la realidad con un esfuerzo computacional aún manejable. No obstante, en aplicaciones de alta fidelidad o donde se necesite una precisión muy alta (por ejemplo, simulaciones aeronáuticas, análisis transitorios muy detallados en electrónica, etc.), a menudo se prefiere ir directamente a métodos de orden 4 o adaptativos. Un aspecto interesante en la práctica es que muchos softwares de simulación y librerías numéricas implementan el método de Euler mejorado como parte de esquemas de integración adaptativos. Por ejemplo, un integrador podría usar Euler mejorado para obtener una estimación rápida y también calcular una estimación con otro método para comparar, ajustando el paso en base a la diferencia (esta es la base de métodos adaptativos tipo Runge-Kutta-Fehlberg, donde se combinan órdenes 2 y 3 o 4 y 5 para estimar error). En cuanto a la comparación con otros métodos, Euler mejorado se suele comparar directamente con el

método de punto medio (otro método RK2) y con Euler simple: todos son métodos de paso único explícitos. Entre ellos, Euler mejorado suele considerarse superior a Euler en exactitud y similar en estabilidad, mientras que frente al método de punto medio (que evalúa

f en el centro del intervalo) tiene comportamiento comparable en orden de error (ambos son orden 2). En la jerarquía de Runge-Kutta, Euler mejorado/Heun es un representante típico de segundo orden, preludio al método de Runge-Kutta de 4º orden que veremos después, el cual logra aún mayor precisión incorporando más evaluaciones dentro del paso.

Método de Rugen Kutta.

Los métodos de Runge-Kutta constituyen una familia de técnicas numéricas ampliamente utilizadas para resolver problemas de valor inicial en ecuaciones diferenciales ordinarias [6] (EDO). Entre ellos, el método de Runge-Kutta de cuarto orden (RK4) es particularmente famoso por su eficiencia y precisión. El objetivo principal de estos métodos es aproximar la solución de una EDO paso a paso, usando múltiples evaluaciones de la derivada en cada intervalo para capturar de manera más fiel la curvatura de la solución.

Dado un problema de valor inicial [7]:

$$y'(t) = f\big(t, y(t)\big), \quad y(t_0) = y_0, \quad t \in [t_0, T],$$

se desea determinar el valor de la función $y(t)$ en puntos discretos $t_n = t_0 + nh$, donde es el **tamaño de paso**. A grandes rasgos, los métodos de Runge-Kutta buscan encontrar una **aproximación** de la integral de la derivada f en cada intervalo $[t_{n,}, t_{n+1}]$ mediante evaluaciones intermedias de la función. Así, se logra una representación más precisa de la forma de $y(t)$ en ese subintervalo.

Método de Runge-Kutta de Cuarto Orden (RK4)

El **RK4** en su forma más habitual se enuncia a través de las siguientes

$$
\begin{aligned}
k_1 &= f\big(t_n,\, y_n\big),\\
k_2 &= f\Big(t_n + \frac{h}{2},\, y_n + \frac{h}{2}\,k_1\Big),\\
k_3 &= f\Big(t_n + \frac{h}{2},\, y_n + \frac{h}{2}\,k_2\Big),\\
k_4 &= f\Big(t_n + h,\, y_n + h\,k_3\Big),\\
y_{n+1} &= y_n + \frac{h}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big).
\end{aligned}
$$

ecuaciones:

1. **Evaluación inicial (k_1)**: Se calcula la pendiente en el punto (t_n, y_n) tal como en el método de Euler.
2. **Evaluaciones intermedias (k_2 y k_3)**: Se usan para refinar la estimación de la pendiente, evaluándola a mitad de paso $t_n + \frac{h}{2}$ con correcciones basadas en k_1 y k_2.
3. **Evaluación final (k_4)**: Se evalúa en $t_n + h$, usando la información acumulada.
4. **Combinación ponderada**: Finalmente, el valor y_{n+1} se obtiene agregando al valor anterior la suma ponderada de dichas pendientes k_1, k_2, k_3 y k_4. Esta fórmula logra un orden de convergencia 4.

Origen de los Coeficientes

Los coeficientes $\frac{1}{6}, \frac{2}{6}, \frac{2}{6}, \frac{1}{6}$ se derivan de comparar la **serie de Taylor** de $y(t)$ hasta términos de orden h^4 con la forma aproximada que se obtiene al combinar k_1, k_2, k_3 y k_4 [8]. La exactitud con polinomios hasta grado 4 garantiza un **error local de orden 5** y un **error global de orden 4**. Por ello, al disminuir el tamaño de paso h a la mitad, el error decrece aproximadamente por un factor de 16.

Análisis de Estabilidad y Convergencia.

Orden de Convergencia

El método RK4 tiene un **orden 4 de convergencia**, lo que significa que su error global es O(h^4). Comparado con métodos menos avanzados (por ejemplo, Euler de orden 1), esta ganancia en precisión es muy significativa y permite, en general, usar pasos de integración más grandes para un mismo nivel de error.

Estabilidad

Como la mayoría de los métodos de un paso **explícitos**, RK4 no es incondicionalmente estable. Sin embargo, su **región de estabilidad** en el plano complejo es mayor que la de métodos de orden inferior, permitiendo tolerar valores de h relativamente más grandes en muchos problemas. En ecuaciones diferenciales con términos que provocan decaimientos exponenciales rápidos (problemas con $ג < 0$ y grande en magnitud), todavía puede requerirse un paso pequeño para evitar la divergencia. Para problemas extremadamente rígidos, suelen preferirse métodos implícitos.

Aplicaciones en Ingeniería

El método RK4 se considera el **estándar** en multitud de problemas de simulación y análisis numérico por su **buena relación entre costo computacional y precisión** [9]. Ejemplos:

- **Ingeniería Aeroespacial:** Cálculo de trayectorias de naves y satélites, donde las fuerzas varían con la posición y la velocidad.
- **Ingeniería Mecánica:** Análisis de oscilaciones y sistemas masa-resorte, estimación de posiciones y velocidades con alta exactitud.
- **Ingeniería Eléctrica:** Resolución de transitorios en circuitos RLC, modelado de convertidores de potencia y controladores.
- **Ingeniería Civil:** Simulaciones de vibraciones de puentes y edificios sometidos a cargas dinámicas.

En problemas con requerimientos de alta precisión, RK4 suele ser la primera opción. Además, muchos **integradores adaptativos** (p. ej. Runge-Kutta-Fehlberg) se basan en la idea de RK4, combinándola con un método de quinto orden para estimar el error y variar el tamaño de paso según se necesite.

Ejemplos de ejercicios

Método de Euler

Comparación de resultados obtenidos usando el método de Euler con cinco tamaños de paso distintos, con los resultados exactos para $y' = 5y - 25x^2 + 2$, $y(0) = 2$

Código de Matlab:

```
% Parámetros
h = 0.1;
nstep = 10;

% Definición de la función
f = @(x, y) (5 * y - 25 * x^2 + 2);

% Valores iniciales
x = 0;
y = 2;

% Inicializar arrays para almacenar los valores
x_values = 0:h:(nstep*h);
y_values = zeros(1, nstep + 1);
exact_values = zeros(1, nstep + 1);
error_values = zeros(1, nstep + 1);

y_values(1) = y; % Valor inicial de y

% Método de Euler
for n = 1:nstep
    y = y + h * f(x, y);
    x = n * h;

    y_values(n + 1) = y; % Almacenar el valor de y en cada paso

    exact = 2 * exp(5 * x) + 5 * x^2 + 2 * x;
    exact_values(n + 1) = exact; % Almacenar el valor exacto

    error = abs(y - exact); % Calcular el error absoluto
    error_values(n + 1) = error; % Almacenar el error absoluto
end

% Calcular el margen de error promedio
margen_error_promedio = mean(error_values);
disp(['Margen de error promedio: ', num2str(margen_error_promedio)]);

% Crear tabla con los resultados
tabla = table(x_values', y_values', exact_values', error_values', ...
              'VariableNames', {'x', 'y_Euler', 'y_Exacto',
'Error_Absoluto'});
disp(tabla);

% Graficar los resultados
figure;
plot(x_values, y_values, 'bo-', 'DisplayName', 'Euler');
hold on;
plot(x_values, exact_values, 'r*-', 'DisplayName', 'Exacto');
```

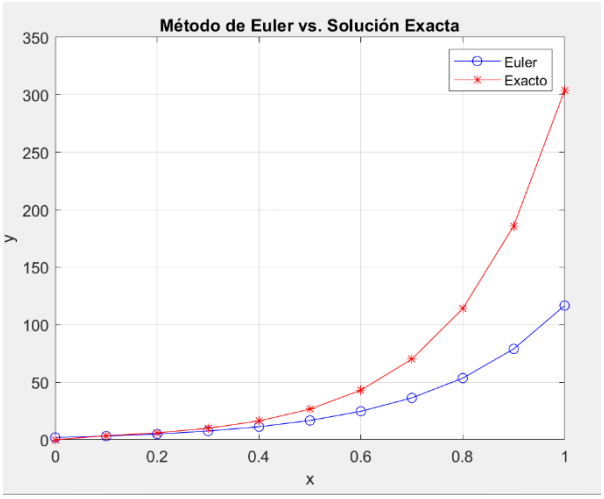
```
xlabel('x');
ylabel('y');
title('Método de Euler vs. Solución Exacta');
legend;
grid on;
hold off;
```

Tabla comparativa de euler y el exacto:

```
Margen de error promedio: 38.6644
```

x	y_Euler	y_Exacto	Error_Absoluto
0	2	0	0
0.1	3.2	3.5474	0.34744
0.2	4.975	6.0366	1.0616
0.3	7.5625	10.013	2.4509
0.4	11.319	16.378	5.0594
0.5	16.778	26.615	9.8369
0.6	24.742	43.171	18.429
0.7	36.413	70.081	33.668
0.8	53.595	114	60.401
0.9	78.992	185.88	106.89
1	116.66	303.83	187.16

Gráfica



Método de Euler mejorado

Usando el método de Euler mejorado con un tamaño de paso (h=0.1), determine la solución del problema de valor inicial

$$y' = 5y - 25x^2 + 2, \qquad y(0) = 2$$

en el intervalo $(0 \leq x \leq 1)$ Compare los resultados con los valores de solución exactos.

Codigo Matlab

```
h_values_1 = 0.1:0.1:1;
h_values_2 = 0.01:0.01:0.1;

results = [];

for h = h_values_1
    x = 0; y = 2;
    nstep = round(1/h);
    f = @(x,y) (5*y - 25*x^2 + 2);

    x_values = 0:h:(nstep*h);
    y_euler = zeros(1, nstep+1);
    y_heun = zeros(1, nstep+1);
    y_exact = zeros(1, nstep+1);
    error_euler = zeros(1, nstep+1);
    error_heun = zeros(1, nstep+1);

    y_euler(1) = y;
    y_heun(1) = y;
    y_exact(1) = 2*exp(5*x_values(1)) + 5*x_values(1)^2 + 2*x_values(1);

    for n = 1:nstep
        y_euler(n+1) = y_euler(n) + h*f(x_values(n), y_euler(n));
        y_pred = y_heun(n) + h*f(x_values(n), y_heun(n));
        y_heun(n+1) = y_heun(n) + h*(f(x_values(n), y_heun(n)) +
f(x_values(n+1), y_pred)) / 2;
        y_exact(n+1) = 2*exp(5*x_values(n+1)) + 5*x_values(n+1)^2 +
2*x_values(n+1);
        error_euler(n+1) = 100 * abs(y_exact(n+1) - y_euler(n+1)) /
y_exact(n+1);
        error_heun(n+1) = 100 * abs(y_exact(n+1) - y_heun(n+1)) /
y_exact(n+1);

        results = [results; h, x_values(n+1), y_euler(n+1), error_euler(n+1),
y_heun(n+1), error_heun(n+1), y_exact(n+1)];
    end
end

for h = h_values_2
    x = 0; y = 2;
    nstep = round(1/h);
    f = @(x,y) (5*y - 25*x^2 + 2);

    x_values = 0:h:(nstep*h);
    y_euler = zeros(1, nstep+1);
    y_heun = zeros(1, nstep+1);
```

```
    y_exact = zeros(1, nstep+1);
    error_euler = zeros(1, nstep+1);
    error_heun = zeros(1, nstep+1);

    y_euler(1) = y;
    y_heun(1) = y;
    y_exact(1) = 2*exp(5*x_values(1)) + 5*x_values(1)^2 + 2*x_values(1);

    for n = 1:nstep
        y_euler(n+1) = y_euler(n) + h*f(x_values(n), y_euler(n));
        y_pred = y_heun(n) + h*f(x_values(n), y_heun(n));
        y_heun(n+1) = y_heun(n) + h*(f(x_values(n), y_heun(n)) +
f(x_values(n+1), y_pred)) / 2;
        y_exact(n+1) = 2*exp(5*x_values(n+1)) + 5*x_values(n+1)^2 +
2*x_values(n+1);
        error_euler(n+1) = 100 * abs(y_exact(n+1) - y_euler(n+1)) /
y_exact(n+1);
        error_heun(n+1) = 100 * abs(y_exact(n+1) - y_heun(n+1)) /
y_exact(n+1);

        results = [results; h, x_values(n+1), y_euler(n+1), error_euler(n+1),
y_heun(n+1), error_heun(n+1), y_exact(n+1)];
    end
end

% Crear tabla de resultados
tabla_resultados = array2table(results, 'VariableNames', {'h', 'x', 'Euler',
'%Error_Euler', 'Euler_Mejorado', '%Error_Euler_Mejorado', 'Exacto'});

% Mostrar la tabla
disp(tabla_resultados);
```

Tabla comparativa de los métodos de Euler y Euler Mejorado

h	x	Euler	%Error_Euler	Euler_Mejorado	%Error_Euler_Mejorado	Exacto
0.1	0.1	3.2	9.7942	3.4875	1.6897	3.5474
0.1	0.2	4.975	17.586	5.8484	3.1164	6.0366
0.1	0.3	7.5625	24.476	9.5662	4.4657	10.013
0.1	0.4	11.319	30.891	15.426	5.8112	16.378
0.1	0.5	16.778	36.96	24.705	7.1752	26.615
0.1	0.6	24.742	42.688	39.477	8.556	43.171
0.1	0.7	36.413	48.041	63.113	9.9423	70.081
0.1	0.8	53.595	52.985	101.09	11.321	114
0.1	0.9	78.992	57.505	162.31	12.683	185.88
0.1	1	116.66	61.602	261.23	14.019	303.83
0.01	0.01	2.12	0.14329	2.123	0.0025762	2.123
0.01	0.02	2.246	0.28268	2.2522	0.005077	2.2523
0.01	0.03	2.3782	0.41851	2.388	0.0075096	2.3882
0.01	0.04	2.5169	0.55113	2.5306	0.0098804	2.5308
0.01	0.05	2.6623	0.68085	2.6802	0.012195	2.6806
0.01	0.06	2.8148	0.80795	2.8373	0.01446	2.8377
0.01	0.07	2.9746	0.93269	3.0021	0.016679	3.0026
0.01	0.08	3.1421	1.0553	3.1751	0.018858	3.1756
0.01	0.09	3.3176	1.176	3.3564	0.021	3.3571
0.01	0.1	3.5015	1.2951	3.5466	0.02311	3.5474

Método de Runge Kutta

Usando el método de Runge-Kutta clásico de cuarto orden con un tamaño de paso (h = 0.5), determine la solución del problema de valor inicial

$$y' = 5y - 25x^2 + 2, \qquad y(0) = 2$$

en el intervalo $(0 \leq x \leq 1)$ Compare los resultados con los valores de solución exactos.

Código de Matlab

```
% Parámetros

h = 0.1;

nstep = 10;


% Definición de la función

f = @(x, y) (5 * y - 25 * x^2 + 2);


% Valores iniciales

x = 0;

y = 2;


% Inicializar arrays para almacenar los valores

x_values = 0:h:(nstep*h);

y_values = zeros(1, nstep + 1);

exact_values = zeros(1, nstep + 1);

error_values = zeros(1, nstep + 1);


y_values(1) = y; % Valor inicial de y


% Método de Euler

for n = 1:nstep

  y = y + h * f(x, y);
```

```
    x = n * h;

    y_values(n + 1) = y; % Almacenar el valor de y en cada paso

    exact = 2 * exp(5 * x) + 5 * x^2 + 2 * x;
    exact_values(n + 1) = exact; % Almacenar el valor exacto

    error = abs(y - exact); % Calcular el error absoluto
    error_values(n + 1) = error; % Almacenar el error absoluto
end

% Calcular el margen de error promedio
margen_error_promedio = mean(error_values);
disp(['Margen de error promedio: ', num2str(margen_error_promedio)]);

% Crear tabla con los resultados
tabla = table(x_values', y_values', exact_values', error_values', ...
        'VariableNames', {'x', 'y_Euler', 'y_Exacto', 'Error_Absoluto'});
disp(tabla);

% Graficar los resultados
figure;
plot(x_values, y_values, 'bo-', 'DisplayName', 'Euler');
hold on;
plot(x_values, exact_values, 'r*-', 'DisplayName', 'Exacto');
xlabel('x');
ylabel('y');
title('Método de Euler vs. Solución Exacta');
```

legend;

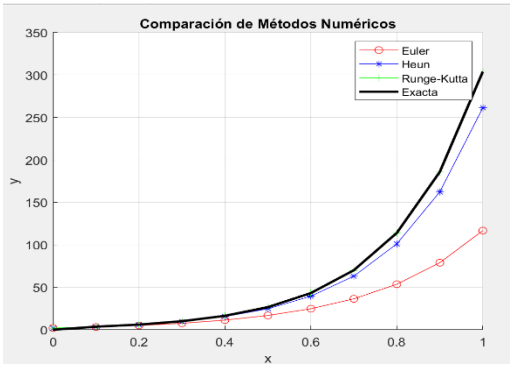
grid on;

hold off;

Tabla comparativa entre el método de Runge Kutta, Euler, Euler Mejorado

h	x	Euler	Error_Euler	Heun	Error_Heun	RK4	Error_RK4	Exacto
0.1	0.1	3.2	9.7942	3.4875	1.6897	3.5467	0.019669	3.5474
0.1	0.2	4.975	17.586	5.8484	3.1164	6.0343	0.036712	6.0366
0.1	0.3	7.5625	24.476	9.5662	4.4657	10.008	0.05319	10.013
0.1	0.4	11.319	30.891	15.426	5.8112	16.367	0.069932	16.378
0.1	0.5	16.778	36.96	24.705	7.1752	26.592	0.087184	26.615
0.1	0.6	24.742	42.688	39.477	8.556	43.126	0.10492	43.171
0.1	0.7	36.413	48.041	63.113	9.9423	69.995	0.12299	70.081
0.1	0.8	53.595	52.985	101.09	11.321	113.84	0.14124	114
0.1	0.9	78.992	57.505	162.31	12.683	185.59	0.15953	185.88
0.1	1	116.66	61.602	261.23	14.019	303.29	0.17775	303.83
1	1	14	95.392	31.5	89.632	124.73	58.947	303.83
0.01	0.01	2.12	0.14329	2.123	0.0025762	2.123	3.0871e-07	2.123
0.01	0.02	2.246	0.28268	2.2522	0.005077	2.2523	6.0886e-07	2.2523
0.01	0.03	2.3782	0.41851	2.388	0.0075096	2.3882	9.0124e-07	2.3882
0.01	0.04	2.5169	0.55113	2.5306	0.0098804	2.5308	1.1866e-06	2.5308
0.01	0.05	2.6623	0.68085	2.6802	0.012195	2.6806	1.4656e-06	2.6806
0.01	0.06	2.8148	0.80795	2.8373	0.01446	2.8377	1.739e-06	2.8377
0.01	0.07	2.9746	0.93269	3.0021	0.016679	3.0026	2.0072e-06	3.0026
0.01	0.08	3.1421	1.0553	3.1751	0.018858	3.1756	2.2709e-06	3.1756
0.01	0.09	3.3176	1.176	3.3564	0.021	3.3571	2.5304e-06	3.3571
0.01	0.1	3.5015	1.2951	3.5466	0.02311	3.5474	2.7863e-06	3.5474

Gráfica

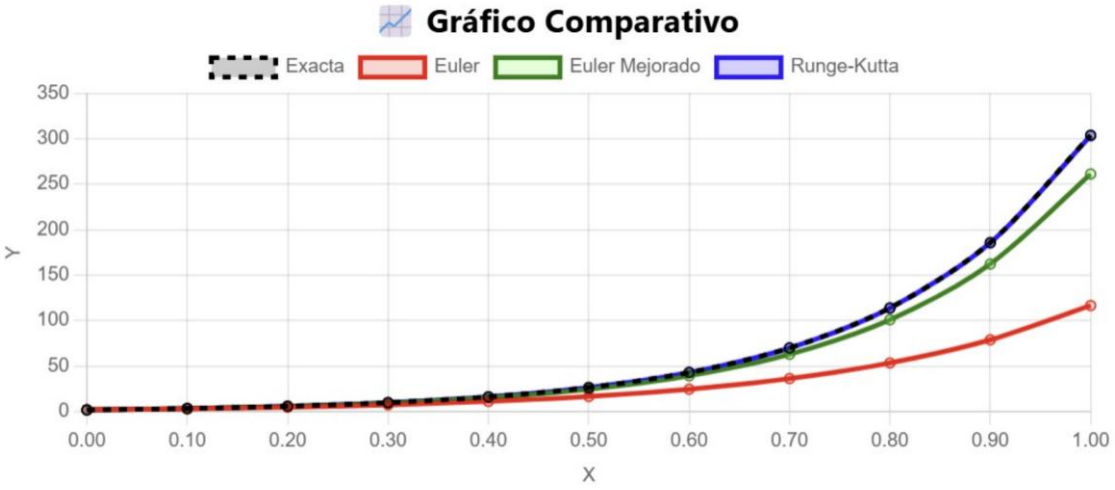


Comprobación con Software

Comparación de Métodos y Márgenes de Error

x	Exacta (Solución General)	Euler	Euler Mejorado	Runge-Kutta	Error Euler (%)	Error Euler Mejorado (%)	Error Runge-Kutta (%)
0.0000	2.0000	2.0000	2.0000000000	2.0000000000	0.0000000000	0.0000000000	0.0000000000
0.1000	3.5474	3.2000	3.4875000000	3.5467450000	9.7941696686	1.6897395997	0.0196632191
0.2000	6.0366	4.9750	5.8484370000	6.0343480000	17.5855622048	3.1164527968	0.0367039436
0.3000	10.0134	7.5625	9.5662110000	10.0080520000	24.4760370201	4.4656971343	0.0531902481
0.4000	16.3781	11.3187	15.4263430000	16.3666590000	30.8909973063	5.8112265099	0.0699299023
0.5000	26.6150	16.7781	24.7053070000	26.5917840000	36.9598624296	7.1752086721	0.0871836631
0.6000	43.1711	24.7422	39.4773740000	43.1257790000	42.6880413305	8.5559600846	0.1049194341
0.7000	70.0809	36.4133	63.1132330000	69.9947090000	48.0410797171	9.9423245533	0.1229934441
0.8000	113.9963	53.5949	101.0902530000	113.8352890000	52.9853846407	11.3214613622	0.1412423615
0.9000	185.8843	78.9924	162.3091610000	185.5877290000	57.5045343298	12.6826775280	0.1595259313
1.0000	303.8263	116.6636	261.2336360000	303.2862820000	61.6018863378	14.0187599471	0.1777450381

Gráfico Comparativo



Sistema de ecuaciones de 2 EDOs

Comparación de los resultados obtenidos usando el método de Euler y el método de Runge-Kutta de cuarto orden, con los resultados exactos para un sistema de problemas de valor inicial de primer orden y' = -y + 9z - 9, y(0) = 6 y z' = y - z + 1, z(0) = 1

Código Matlab:

```
clc; clear; close all;

% Definición de parámetros
x = 0;
y = 6;
z = 1;
h = 0.1;
nstep = 10;

% Definición de funciones
fnf = @(x, y, z) (-y + 9*z - 9);
fng = @(x, y, z) (y - z + 1);

% Inicialización de vectores de valores
y_values_euler = zeros(1, nstep+1);
z_values_euler = zeros(1, nstep+1);
y_values_rk4 = zeros(1, nstep+1);
z_values_rk4 = zeros(1, nstep+1);
y_exact_values = zeros(1, nstep+1);
z_exact_values = zeros(1, nstep+1);
x_values = 0:h:nstep*h;

% Asignar valores iniciales
y_values_euler(1) = y;
z_values_euler(1) = z;
y_values_rk4(1) = y;
z_values_rk4(1) = z;
y_exact_values(1) = 3*exp(2*x) + 3*exp(-4*x);
```

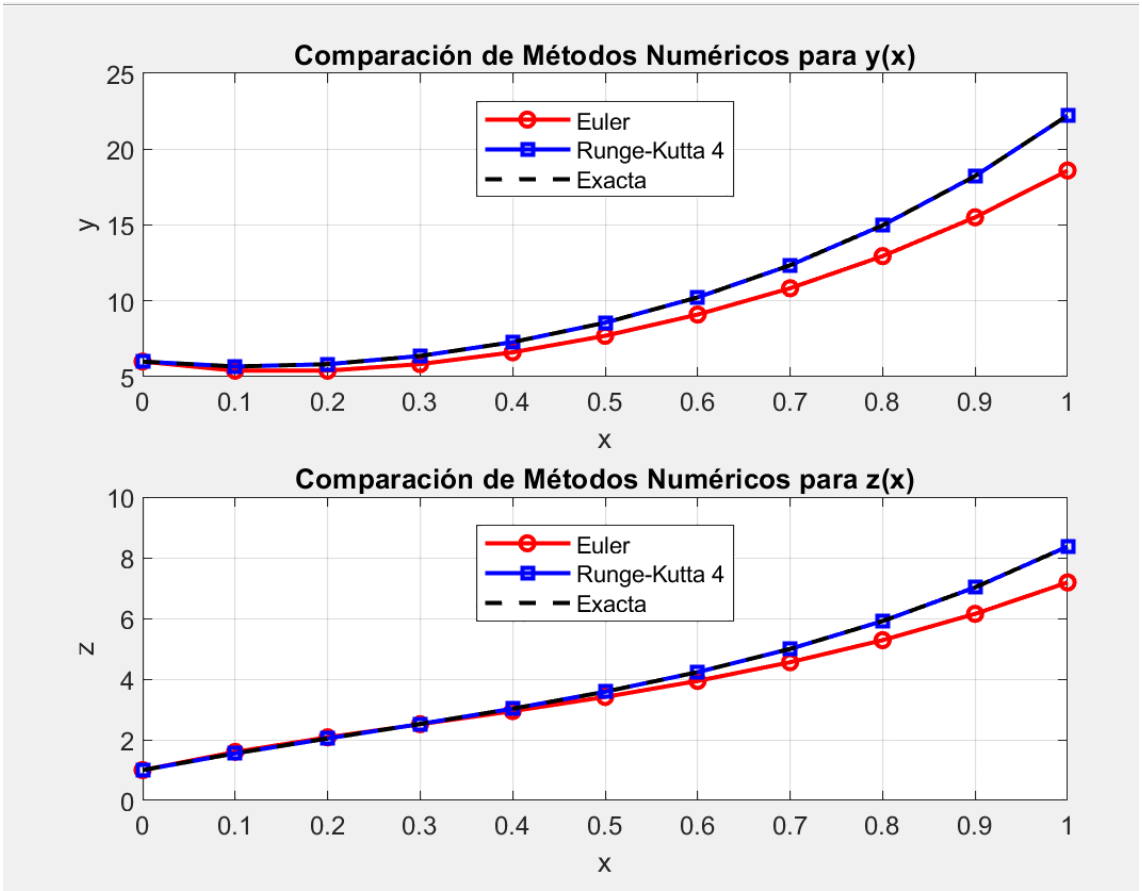
```
z_exact_values(1) = exp(2*x) - exp(-4*x) + 1;

% Método de Euler
y_euler = y;
z_euler = z;
for n = 1:nstep
    y_euler_new = y_euler + h * fnf(x_values(n), y_euler, z_euler);
    z_euler_new = z_euler + h * fng(x_values(n), y_euler, z_euler);

    y_values_euler(n+1) = y_euler_new;
    z_values_euler(n+1) = z_euler_new;

    y_euler = y_euler_new;
    z_euler = z_euler_new;
end

% Método de Runge-Kutta 4
y = 6;
z = 1;
for n = 1:nstep
    K1 = fnf(x_values(n), y, z);
    L1 = fng(x_values(n), y, z);

    K2 = fnf(x_values(n) + h/2, y + h*K1/2, z + h*L1/2);
    L2 = fng(x_values(n) + h/2, y + h*K1/2, z + h*L1/2);

    K3 = fnf(x_values(n) + h/2, y + h*K2/2, z + h*L2/2);
    L3 = fng(x_values(n) + h/2, y + h*K2/2, z + h*L2/2);

    K4 = fnf(x_values(n) + h, y + h*K3, z + h*L3);
    L4 = fng(x_values(n) + h, y + h*K3, z + h*L3);

    y = y + h*(K1 + 2*K2 + 2*K3 + K4) / 6;
    z = z + h*(L1 + 2*L2 + 2*L3 + L4) / 6;

    y_values_rk4(n+1) = y;
    z_values_rk4(n+1) = z;

    y_exact_values(n+1) = 3*exp(2*x_values(n+1)) + 3*exp(-4*x_values(n+1));
    z_exact_values(n+1) = exp(2*x_values(n+1)) - exp(-4*x_values(n+1)) + 1;
end

% Cálculo de los errores absolutos
error_y_euler = abs(y_values_euler - y_exact_values);
error_z_euler = abs(z_values_euler - z_exact_values);
error_y_rk4 = abs(y_values_rk4 - y_exact_values);
error_z_rk4 = abs(z_values_rk4 - z_exact_values);

% Mostrar resultados en una tabla
T = table(x_values', y_values_euler', z_values_euler', ...
          y_values_rk4', z_values_rk4', y_exact_values', z_exact_values', ...
          error_y_euler', error_z_euler', error_y_rk4', error_z_rk4', ...
          'VariableNames', {'x', 'Euler_y', 'Euler_z', 'RK4_y', 'RK4_z',
'Exact_y', 'Exact_z', ...
                            'Error_Euler_y', 'Error_Euler_z', 'Error_RK4_y',
'Error_RK4_z'});

disp(T);
```

```
% Graficar los resultados
figure;

% Gráfica de y
subplot(3,1,1);
plot(x_values, y_values_euler, 'o-r', 'LineWidth', 1.5, 'MarkerSize', 5);
hold on;
plot(x_values, y_values_rk4, 's-b', 'LineWidth', 1.5, 'MarkerSize', 5);
plot(x_values, y_exact_values, '--k', 'LineWidth', 1.5);
xlabel('x');
ylabel('y');
title('Comparación de Métodos Numéricos para y(x)');
legend('Euler', 'Runge-Kutta 4', 'Exacta', 'Location', 'Best');
grid on;

% Gráfica de z
subplot(3,1,2);
plot(x_values, z_values_euler, 'o-r', 'LineWidth', 1.5, 'MarkerSize', 5);
hold on;
plot(x_values, z_values_rk4, 's-b', 'LineWidth', 1.5, 'MarkerSize', 5);
plot(x_values, z_exact_values, '--k', 'LineWidth', 1.5);
xlabel('x');
ylabel('z');
title('Comparación de Métodos Numéricos para z(x)');
legend('Euler', 'Runge-Kutta 4', 'Exacta', 'Location', 'Best');
grid on;

% Gráfica del error absoluto
subplot(3,1,3);
plot(x_values, error_y_euler, 'r', 'LineWidth', 1.5);
hold on;
plot(x_values, error_z_euler, '--r', 'LineWidth', 1.5);
plot(x_values, error_y_rk4, 'b', 'LineWidth', 1.5);
plot(x_values, error_z_rk4, '--b', 'LineWidth', 1.5);
xlabel('x');
ylabel('Error absoluto');
title('Margen de error en y(x) y z(x)');
legend('Error Euler y', 'Error Euler z', 'Error RK4 y', 'Error RK4 z',
'Location', 'Best');
grid on;
```

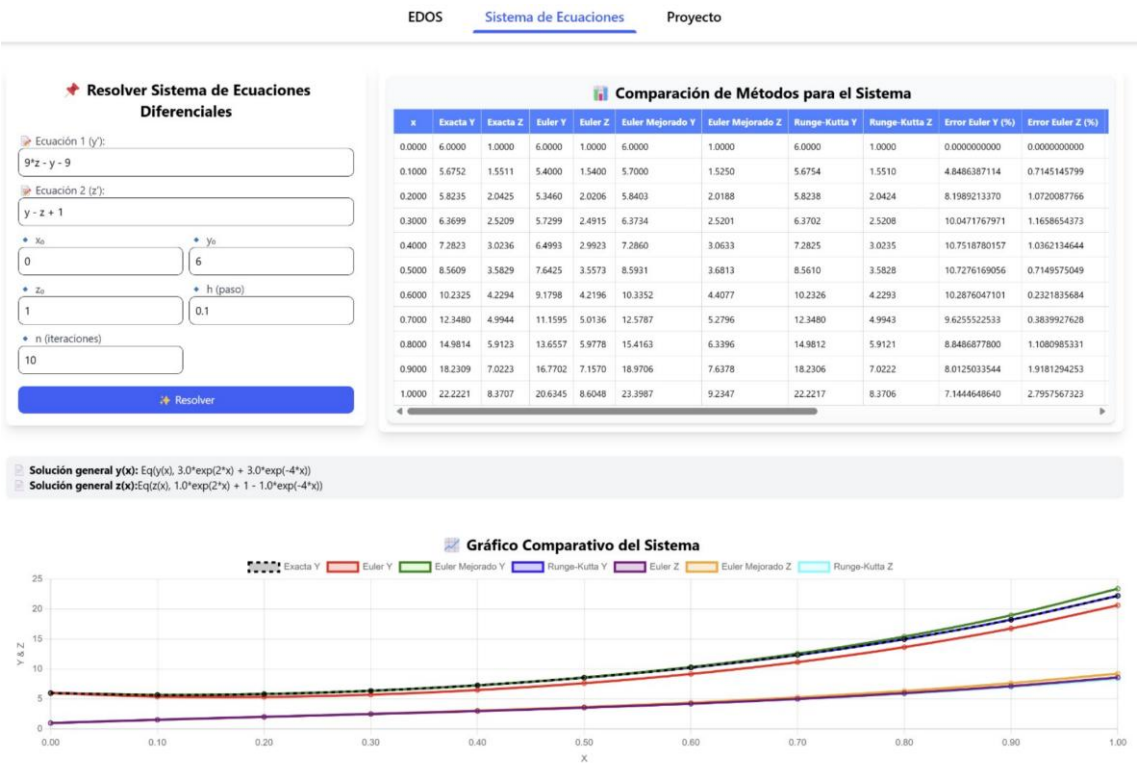
Tabla comparativa de los métodos Euler y Kutta 4

x	Euler_y	Euler_z	RK4_y	RK4_z	Exact_y	Exact_z	Error_Euler_y	Error_Euler_z	Error_RK4_y	Error_RK4_z
0	6	1	6	1	6	1	0	0	0	0
0.1	5.4	1.6	5.6754	1.551	5.6752	1.5511	0.27517	0.048917	0.00023159	8.2712e-05
0.2	5.4	2.08	5.8238	2.0424	5.8235	2.0425	0.42346	0.037504	0.00030137	0.00011393
0.3	5.832	2.512	6.3702	2.5208	6.3699	2.5209	0.53794	0.0089246	0.00028634	0.00012013
0.4	6.6096	2.944	7.2825	3.0235	7.2823	3.0236	0.67271	0.079644	0.00022872	0.00011645
0.5	7.6982	3.4106	8.561	3.5828	8.5609	3.5829	0.86261	0.17239	0.00015012	0.00011142
0.6	9.0979	3.9393	10.233	4.2293	10.233	4.2294	1.1346	0.29007	5.9875e-05	0.00010993
0.7	10.834	4.5552	12.348	4.9943	12.348	4.9944	1.5145	0.4392	3.9931e-05	0.00011489
0.8	12.95	5.283	14.981	5.9121	14.981	5.9123	2.0315	0.62925	0.0001517	0.00012839
0.9	15.51	6.1497	18.231	7.0222	18.231	7.0223	2.7213	0.87262	0.00028081	0.0001523
1	18.593	7.1857	22.222	8.3706	22.222	8.3707	3.6288	1.1851	0.000435	0.00018872

Gráfica



Comprobación con Software



Software desarrollado

Este proyecto consiste en un **Backend** construido con **Python + FastAPI** y la biblioteca **Sympy** para:

- Resolver Ecuaciones Diferenciales Ordinarias (EDO) mediante el Método de Ecuaciones Exactas.
- Resolver Sistemas de Ecuaciones lineales y no lineales.

El Frontend está desarrollado con Vue 3, utilizando Tailwind CSS para el diseño, Chart.js para las gráficas, Math.js para el manejo de expresiones matemáticas, TypeScript para el tipado estático, Axios para comunicarse con el Backend y Pinia para la gestión de estado global.

El objetivo es brindar una aplicación web sencilla que permita ingresar ecuaciones, resolverlas y visualizar los resultados, todo ello de forma dinámica.

Arquitectura del Proyecto.

La arquitectura se divide en dos grandes partes:

1) **Backend**
   a) **FastAPI** expone los endpoints para:
      i) Resolver ecuaciones diferenciales exactas.
      ii) Resolver sistemas de ecuaciones.
   b) Se utiliza **Sympy** para manipulación y resolución simbólica de ecuaciones.
   c) **Uvicorn** se encarga de levantar el servidor en desarrollo o producción.
2) **Frontend**

a) **Vue 3** como framework principal de JavaScript/TypeScript.

b) **Tailwind CSS** para estilos y maquetación rápida.

c) **Chart.js** para representar datos y/o soluciones gráficamente.

d) **Math.js** para evaluar expresiones matemáticas en el cliente.

e) **Pinia** para gestión de estado global de la aplicación.

f) **Axios** para comunicar el frontend con el backend mediante llamadas HTTP.

Portada principal del proyecto

- Presenta la aplicación, su objetivo y un resumen de las funcionalidades disponibles.
- Botones o enlaces que dirigen a las secciones de “EDOs” y “Sistemas de ecuaciones”.

Vista para resolver EDOS:

- Formulario para ingresar la ecuación diferencial (o el sistema, en caso de ser un sistema de EDOs).

- Presentación de la **solución simbólica** y, de ser necesario, la **solución numérica**.

Gráficas de EDOS:

- Gráficas comparativas (usando Chart.js) para visualizar la función solución versus aproximaciones.
- Tabla de comparación de resultados, errores.

Gráfico Comparativo

Grafica de errores:

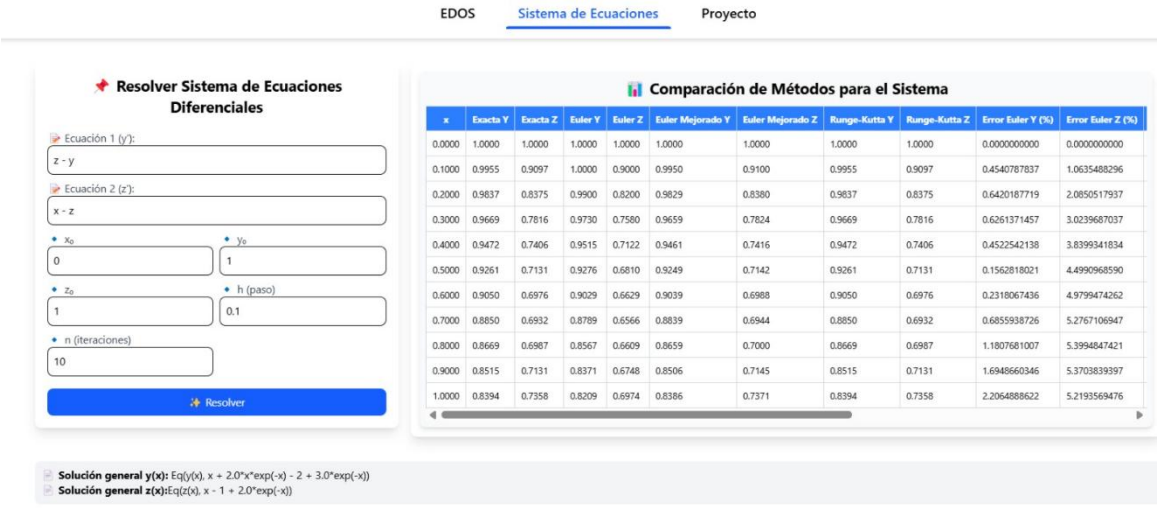
Tabla de comparación de errores

Comparación de Métodos y Márgenes de Error

x	Exacta (Solución General)	Euler	Euler Mejorado	Runge-Kutta	Error Euler (%)	Error Euler Mejorado (%)	Error Runge-Kutta (%)
0.0000	1.0000	1.0000	1.0000000000	1.0000000000	0.0000000000	0.0000000000	0.0000000000
0.1000	0.9052	0.9000	0.9055000000	0.9051630000	0.5703485834	0.0372770641	0.0000461835
0.2000	0.8213	0.8110	0.8219270000	0.8212690000	1.2504117207	0.0800898220	0.0000300659
0.3000	0.7492	0.7339	0.7501440000	0.7491820000	2.0397959134	0.1284362098	0.0000294564
0.4000	0.6897	0.6695	0.6909310000	0.6896800000	2.9245382367	0.1813951571	0.0000066749
0.5000	0.6435	0.6186	0.6449920000	0.6434700000	3.8712551986	0.2366328304	0.0001025243
0.6000	0.6112	0.5817	0.6129680000	0.6111890000	4.8242678767	0.2911763704	0.0001040750
0.7000	0.5934	0.5595	0.5954360000	0.5934150000	5.7096152868	0.3406224693	0.0000511938
0.8000	0.5907	0.5526	0.5929200000	0.5906720000	6.4487732712	0.3807473163	0.0001632241
0.9000	0.6034	0.5613	0.6058920000	0.6034310000	6.9781609326	0.4079443104	0.0001093317
1.0000	0.6321	0.5862	0.6347830000	0.6321220000	7.2662656177	0.4211919917	0.0002279900

Vista para resolver sistemas de ecuaciones:

- Permite ingresar múltiples ecuaciones y variables.



x	Exacta Y	Exacta Z	Euler Y	Euler Z	Euler Mejorado Y	Euler Mejorado Z	Runge-Kutta Y	Runge-Kutta Z	Error Euler Y (%)	Error Euler Z (%)
0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000000000	0.0000000000
0.1000	0.9955	0.9097	1.0000	0.9000	0.9950	0.9100	0.9955	0.9097	0.4540787837	1.0635488296
0.2000	0.9837	0.8375	0.9900	0.8200	0.9829	0.8380	0.9837	0.8375	0.6420187719	2.0850517937
0.3000	0.9669	0.7816	0.9730	0.7580	0.9659	0.7824	0.9669	0.7816	0.6261371457	3.0239687037
0.4000	0.9472	0.7406	0.9515	0.7122	0.9461	0.7416	0.9472	0.7406	0.4522542138	3.8399341834
0.5000	0.9261	0.7131	0.9276	0.6810	0.9249	0.7142	0.9261	0.7131	0.1562818021	4.4990968590
0.6000	0.9050	0.6976	0.9029	0.6629	0.9039	0.6988	0.9050	0.6976	0.2318067436	4.9799474262
0.7000	0.8850	0.6932	0.8789	0.6566	0.8839	0.6944	0.8850	0.6932	0.6855938726	5.2767106947
0.8000	0.8669	0.6987	0.8567	0.6609	0.8659	0.7000	0.8669	0.6987	1.1807681007	5.3994847421
0.9000	0.8515	0.7131	0.8371	0.6748	0.8506	0.7145	0.8515	0.7131	1.6948660346	5.3703839397
1.0000	0.8394	0.7358	0.8209	0.6974	0.8386	0.7371	0.8394	0.7358	2.2064888622	5.2193569476

Gráfico comparativo del sistema de ecuaciones:

Valores de Y:

Valores de Z:

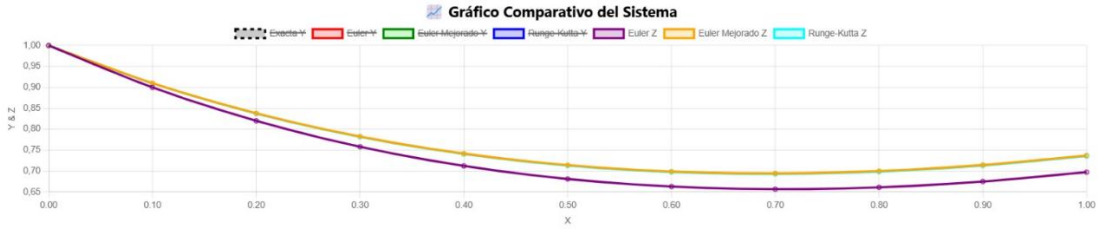
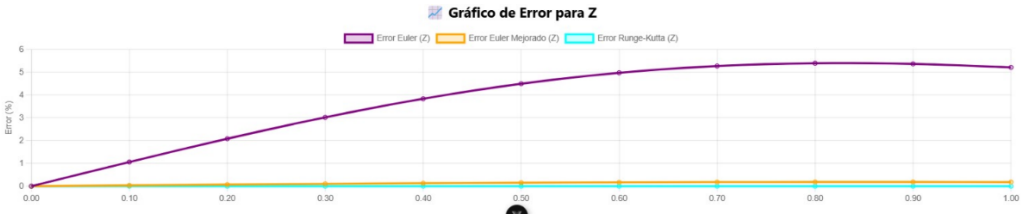


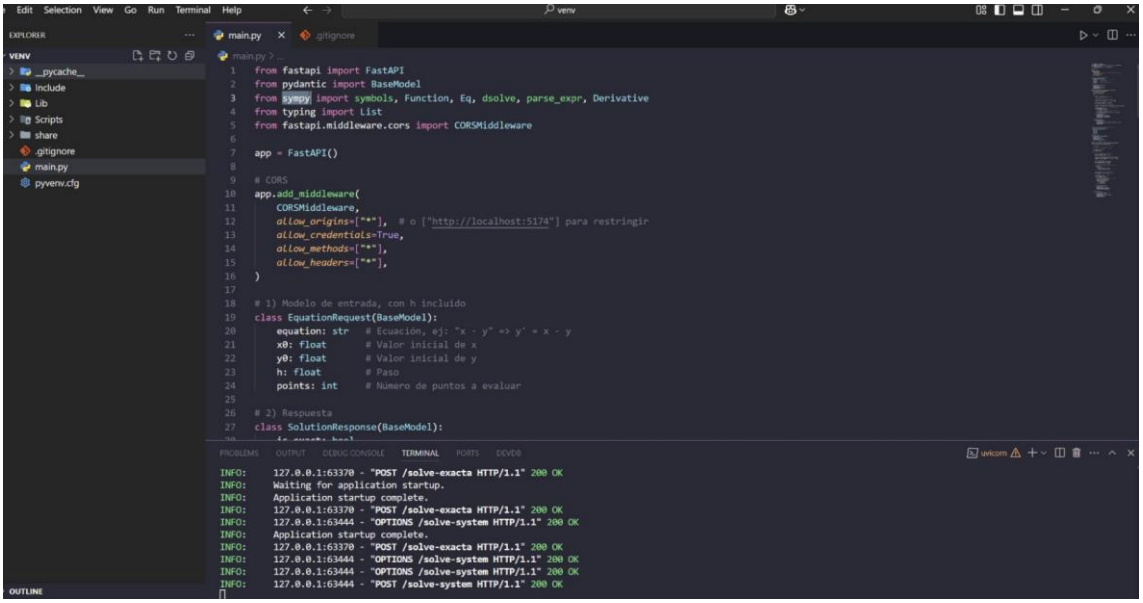
Gráfico de error para Y:

Gráfico de error para Z:



Evidencia en Código del Software

Backend en Python




```python
@app.post("/solve-exacta", response_model=SolutionResponse)
def solve_exacta(data: EquationRequest):
    x = symbols('x', real=True)
    # Definimos la función y(x)
    y_func = Function('y')(x)

    try:
        # Reemplazar '^' por '**'
        eq_str = data.equation.replace('^', '**')

        # Interpretar "y" como y(x):
        #  parse_expr(eq_str, {"x": x, "y": y_func})
        #  Si eq_str = "x - y", se convertirá en x - y_func
        eq_expr = parse_expr(eq_str, {"x": x, "y": y_func})

        # Formar la EDO: Derivative(y, x) = eq_expr
        ode = Eq(Derivative(y_func, x), eq_expr)

        # Resolver con la condición inicial y(x0) = y0
        solution = dsolve(ode, y_func, ics={y_func.subs(x, data.x0): data.y0})

        # Evaluar la solución en (points + 1) puntos, con paso h
        evaluated_points = []
        for i in range(data.points + 1):
            xi = data.x0 + i * data.h
            # Substituir x = xi en la parte derecha (solution.rhs)
            yi = solution.rhs.subs(x, xi)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  DEVDB
INFO:     127.0.0.1:63370 - "POST /solve-exacta HTTP/1.1" 200 OK
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:63370 - "POST /solve-exacta HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "OPTIONS /solve-system HTTP/1.1" 200 OK
INFO:     Application startup complete.
INFO:     127.0.0.1:63370 - "POST /solve-exacta HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "OPTIONS /solve-system HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "OPTIONS /solve-system HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "POST /solve-system HTTP/1.1" 200 OK
```

main.py .gitignore

main.py > ...

```python
def solve_exacta(data: EquationRequest):
        solution = dsolve(ode, y_func, ics={y_func.subs(x, data.x0): data.y0})

        # Evaluar la solución en (points + 1) puntos, con paso h
        evaluated_points = []
        for i in range(data.points + 1):
            xi = data.x0 + i * data.h
            # Substituir x = xi en la parte derecha (solution.rhs)
            yi = solution.rhs.subs(x, xi)
            evaluated_points.append({"x": float(xi), "y": float(yi)})

        return SolutionResponse(
            is_exact=True,
            general_solution=str(solution),
            evaluated_points=evaluated_points,
            equation_order=1  # Primer orden
        )

    except Exception as e:
        return SolutionResponse(
            is_exact=False,
            general_solution=f"Error al procesar la ecuación: {str(e)}",
            evaluated_points=[],
            equation_order=0
        )
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB

uvicorn

```
INFO:     127.0.0.1:63370 - "POST /solve-exacta HTTP/1.1" 200 OK
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:63370 - "POST /solve-exacta HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "OPTIONS /solve-system HTTP/1.1" 200 OK
INFO:     Application startup complete.
INFO:     127.0.0.1:63370 - "POST /solve-exacta HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "OPTIONS /solve-system HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "OPTIONS /solve-system HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "POST /solve-system HTTP/1.1" 200 OK
```

main.py .gitignore

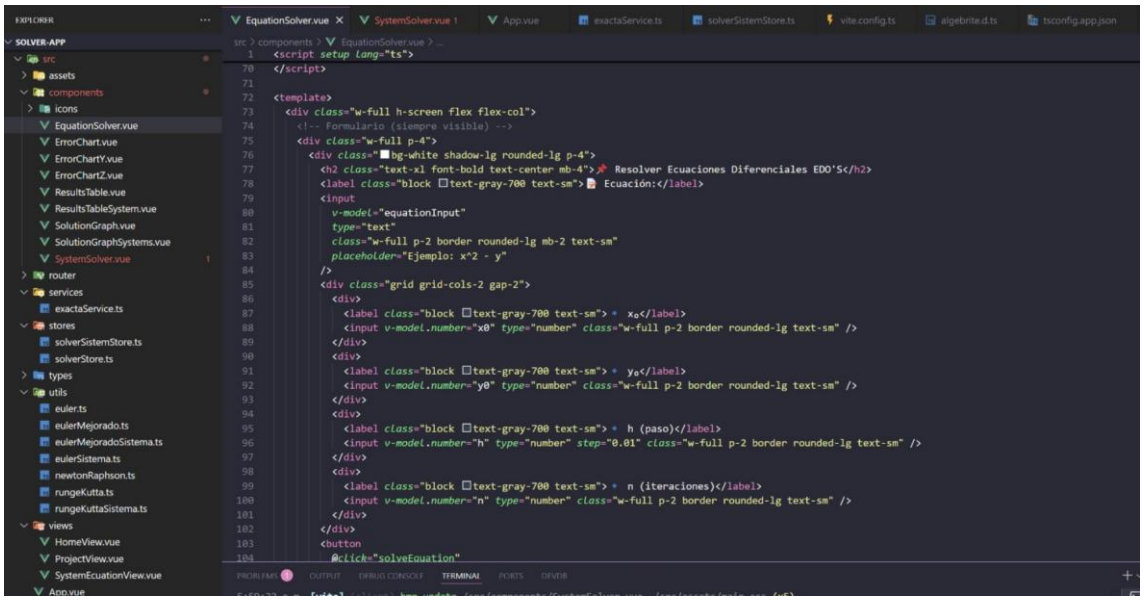
main.py > ...

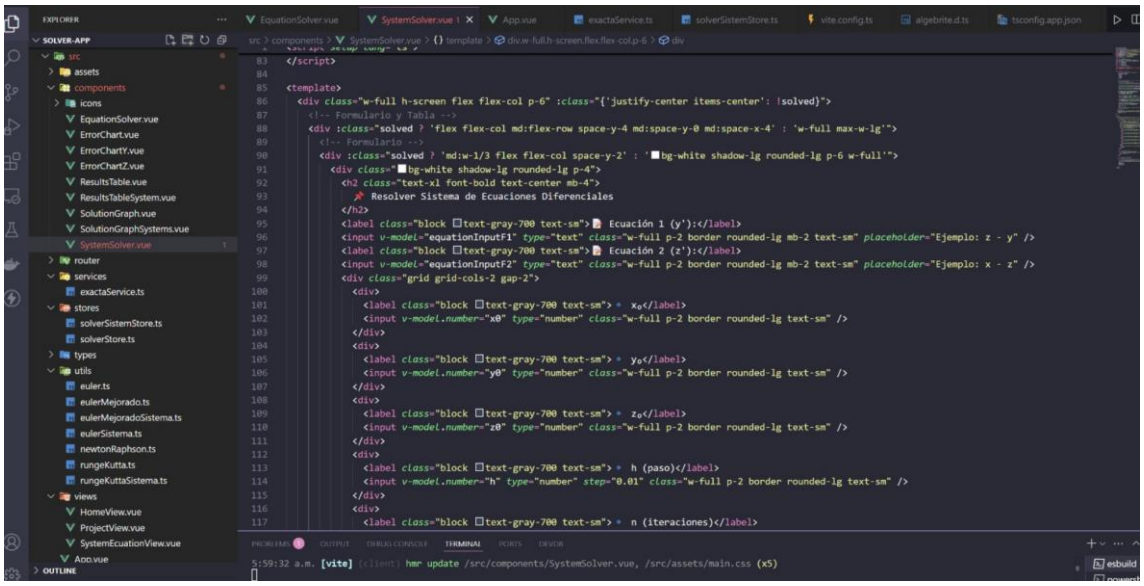
```python
# Modelo de entrada para el sistema y' = eqy, z' = eqz
class SystemEquationRequest(BaseModel):
    eqy: str  # Ecuación para y'
    eqz: str  # Ecuación para z'
    x0: float
    y0: float
    z0: float
    h: float
    points: int

# Modelo de salida
class SystemSolutionResponse(BaseModel):
    is_exact: bool
    general_solution_y: str
    general_solution_z: str
    evaluated_points: List[dict]

# -------------- ENDPOINT --------------

@app.post("/solve-system", response_model=SystemSolutionResponse)
def solve_system(data: SystemEquationRequest):
    """
    Resuelve un sistema de 2 ecuaciones de primer orden:
      y' = eqy
      z' = eqz
    con condiciones iniciales y(x0)=y0, z(x0)=z0.
    """
```

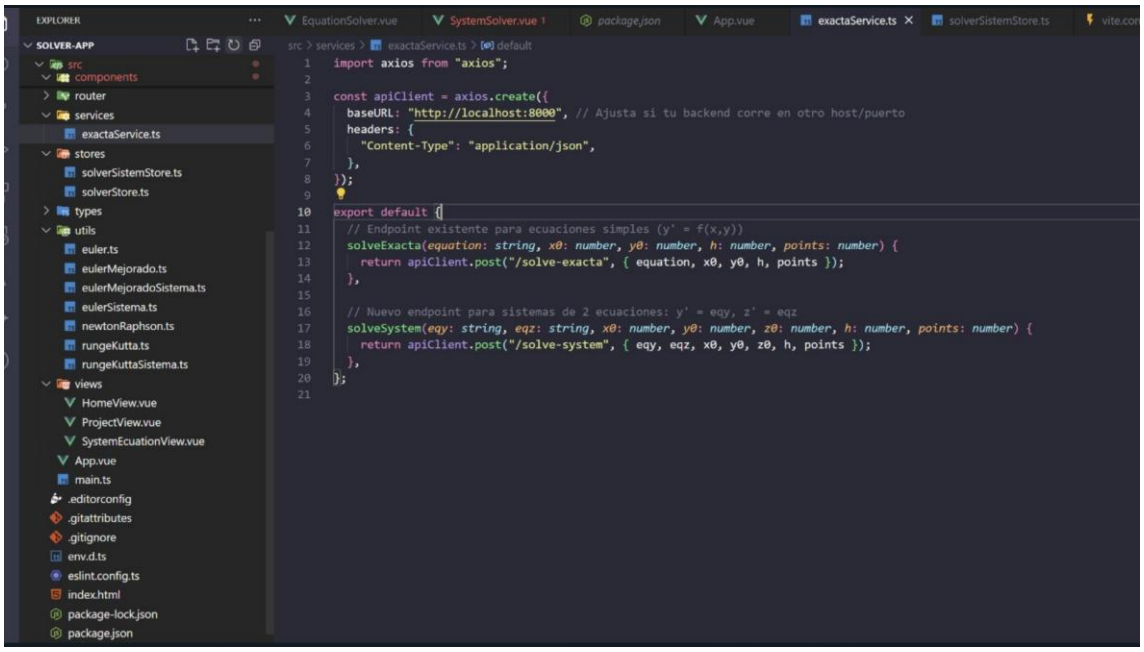
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB

```
INFO:     127.0.0.1:63370 - "POST /solve-exacta HTTP/1.1" 200 OK
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:63370 - "POST /solve-exacta HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "OPTIONS /solve-system HTTP/1.1" 200 OK
INFO:     Application startup complete.
INFO:     127.0.0.1:63370 - "POST /solve-exacta HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "OPTIONS /solve-system HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "OPTIONS /solve-system HTTP/1.1" 200 OK
INFO:     127.0.0.1:63444 - "POST /solve-system HTTP/1.1" 200 OK
```

Frontend en VueJS






EXPLORER
SOLVER-APP
src
components
router
services
exactaService.ts
stores
solverSistemStore.ts
solverStore.ts
types
utils
euler.ts
eulerMejorado.ts
eulerMejoradoSistema.ts
eulerSistema.ts
newtonRaphson.ts
rungeKutta.ts
rungeKuttaSistema.ts
views
HomeView.vue
ProjectView.vue
SystemEcuationView.vue
App.vue
main.ts
.editorconfig
.gitattributes
.gitignore
env.d.ts
eslint.config.ts
index.html
package-lock.json
package.json
EquationSolver.vue
SystemSolver.vue 1
package.json
App.vue
exactaService.ts
solverSistemStore.ts
src > services > exactaService.ts > default
import axios from "axios";
const apiClient = axios.create({
baseURL: "http://localhost:8000", // Ajusta si tu backend corre en otro host/puerto
headers: {
"Content-Type": "application/json",
},
});
export default {
// Endpoint existente para ecuaciones simples (y' = f(x,y))
solveExacta(equation: string, x0: number, y0: number, h: number, points: number) {
return apiClient.post("/solve-exacta", { equation, x0, y0, h, points });
},
// Nuevo endpoint para sistemas de 2 ecuaciones: y' = eqy, z' = eqz
solveSystem(eqy: string, eqz: string, x0: number, y0: number, z0: number, h: number, points: number) {
return apiClient.post("/solve-system", { eqy, eqz, x0, y0, z0, h, points });
},
};


EquationSolver.vue
SystemSolver.vue 1
App.vue
exactaService.ts
euler.ts
src > utils > euler.ts > euler
export function euler(
f: (x: number, y: number) => number,
x0: number,
y0: number,
h: number,
n: number
): Array<{ x: number; y: number }> {
let x = x0;
let y = y0;
const results = [{ x: parseFloat(x.toFixed(6)), y: parseFloat(y.toFixed(6)) }];
for (let i = 0; i < n; i++) {
y = y + h * f(x, y);
x = x + h;
results.push({ x: parseFloat(x.toFixed(6)), y: parseFloat(y.toFixed(6)) });
}
return results;
}

Referencias

[1] L. Euler, «Institutionum Calculi Integralis,» *Suiza: Apud Marci-Michaelis Bousquet & Socios,* vol. vol. 1. Lausana, 1768.

[2] S. C. C. y. R. P. Canale, Métodos Numéricos para Ingenieros, México D.F: McGraw-Hill, 2010.

[3] S. P. N. y. G. W. E. Hairer, Solving Ordinary Differential Equations I: Nonstiff Problems, Berlín, Alemania: Springer, 1993.

[4] R. L. Burden y J. D. Faires, Numerical Analysis, Boston, MA: Brooks/Cole, 2015.

[5] H. Heun, “Neue Methoden zur Approximation an die Lösungen von linearen Differentialgleichungen,”, Zeitschrift für angewandte Mathematik und Mechanik, 1928.

[6] J. C. Butcher, Numerical Methods for Ordinary Differential Equations, Chichester: UK: John Wiley & Sons, 2016.

[7] J. D. Lambert, Numerical Methods for Ordinary Differential Systems: The Initial Value Problem., Chichester: UK: John Wiley & Sons, 1991.

[8] C. F. G. y. P. O. Wheatley, Applied Numerical Analysis, 7th ed. Upper Saddle River, NJ: Pearson, 2004.

[9] S. A. T. W. T. V. y. B. P. F. W. H. Press, Numerical Recipes in C: The Art of Scientific Computing, Cambridge, Reino Unido: Cambridge University Press, 1992.