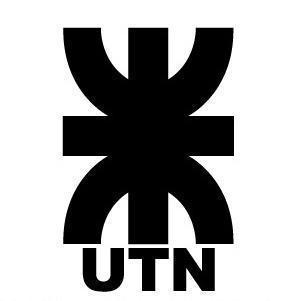
**UNIVERSIDAD TECNOLÓGICA NACIONAL**

**FACULTAD REGIONAL DE SANTA FE**

**TRABAJO PRÁCTICO N° 4**

**Integración numérica, Diferenciación numérica y Resolución de EDOS**

**Cátedra:** Matemática Superior

**Carrera:** Ingeniería en Sistemas de Información

**Comisión:** 3A

**Año:** 2021 - 2° Cuatrimestre

**Integrantes:** Albani, Juan - juanalbani48@gmail.com

Musso, Emilio - emilio.musso2898@gmail.com

Nuñez, Nahuel - nahuelnunezz14@gmail.com

Pagliaruzza, Manuel - manupa4@live.com.ar

Peiretti, Tomás - tomaspeiretti@gmail.com

Riquelme, Nahuel - nahuelriquelme00@gmail.com

**ÍNDICE**

[1 - DESARROLLO](#_afd6d2qsyxmj) 3

[1.1 - CONSTRUCCIÓN DE MÉTODOS DE APROXIMACIÓN INTEGRAL](#_etmwgp124wsq) 3

[1.1.1 - FÓRMULA SEMIABIERTA](#_o9teqr1eoh7k) 3

[1.1.1.1 - ANALISIS DE LA EXPANSION DE TAYLOR](#_jsvnwomhewiz) 3

[1.1.1.2 - ANALISIS DE LA UTILIDAD Y DEL ORDEN DE ERROR](#_jerwkc20n895) 4

[1.1.2 - FÓRMULA SUPRACERRADA](#_zgfb6fq9ie1w) 5

[1.1.2.1 - ANALISIS DE LA EXPANSION DE TAYLOR](#_ow1fgnsz4j7l) 5

[1.1.2.2 - ANALISIS LA UTILIDAD Y DEL ORDEN DE ERROR](#_wmpkar22dop9) 7

[1.2 - MÉTODO PARA EDOS](#_a871nfqbtu1c) 8

[1.3 - APLICACIÓN A ECG](#_skct06wlrjzt) 9

[1.3.1 - OBTENCIÓN DE LA APROXIMACIÓN Y DIAGRAMA V-W](#_5pt5a2ttn7ez) 9

[1.3.2 - ESTIMADOR DE RICHARDSON PARA EL MÉTODO](#_wu7kxxbu010t) 13

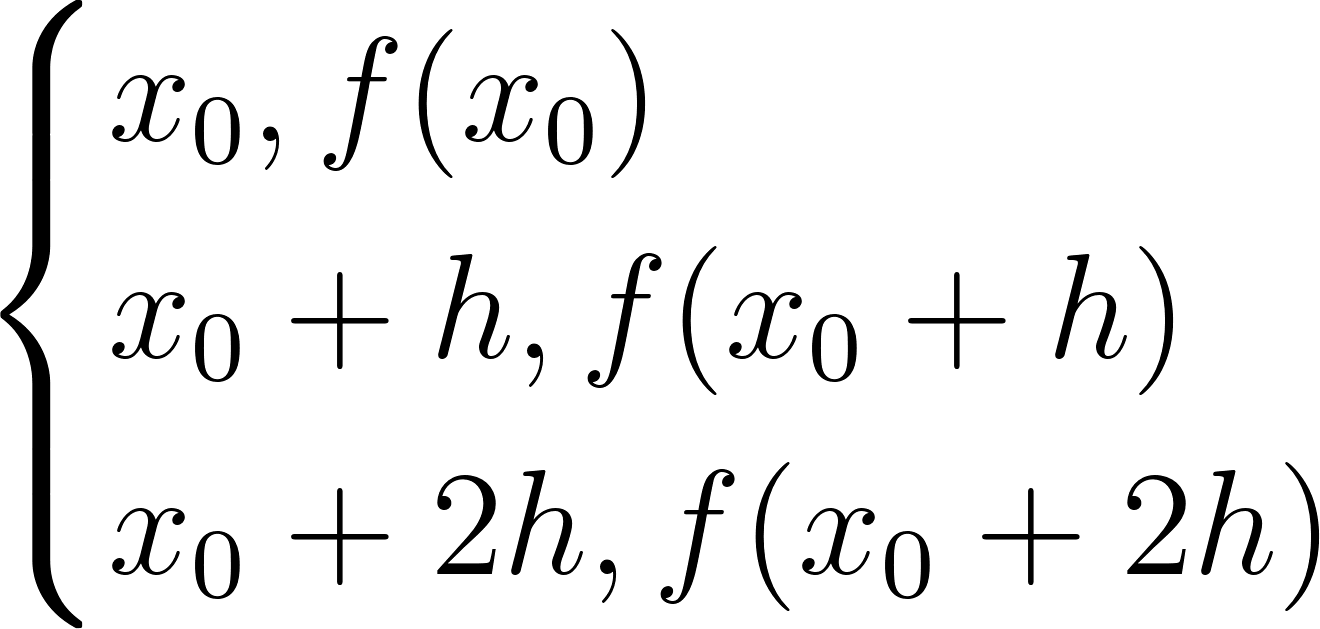
[1.3.3 - OBSERVACIONES Y CONCLUSIONES](#_tf8xppwiuck6) 13

[1.4 - MODELO CARDIACO 2D](#_i7nqzeh72z60) 15

# 1 - DESARROLLO

## 1.1 - CONSTRUCCIÓN DE MÉTODOS DE APROXIMACIÓN INTEGRAL

Se debe tener en cuenta que los puntos utilizados para formular los métodos de aproximación son los siguientes:

[](https://www.codecogs.com/eqnedit.php?latex=%5Cbegin%7Bcases%7D%20x_%7B0%7D%2C%20f(x_%7B0%7D)%20%5C%5C%5C%5C%20%20x_%7B0%7D%20%2B%20h%2C%20f(x_%7B0%7D%20%2B%20h)%20%5C%5C%5C%5C%20%20x_%7B0%7D%20%2B%202h%2C%20f(x_%7B0%7D%20%2B%202h)%20%5Cend%7Bcases%7D#0)

### 1.1.1 - FÓRMULA SEMIABIERTA

#### 1.1.1.1 - ANALISIS DE LA EXPANSION DE TAYLOR

Se comenzó planteando la fórmula de la integral a calcular utilizando el teorema fundamental del cálculo:

Siendo F la función primitiva de f, es decir:

Para este caso, la fórmula de la integral resultó:

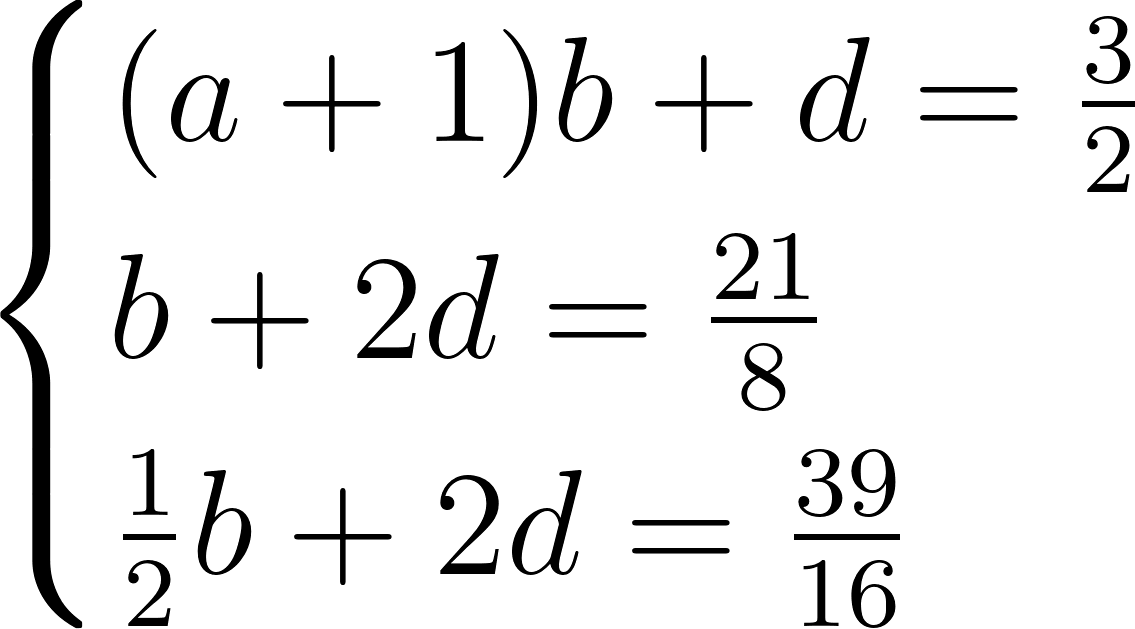
A continuación, se realizaron las expansiones de taylor de las funciones presentes en la integral y se calculó su diferencia:

Se sabe que , entonces la **expansión de taylor exacta** resulta:

Luego se desarrollaron los puntos dados:

Se plantea la siguiente combinación lineal para encontrar una aproximación a la integral propuesta, que denominaremos **expansión de taylor aproximada**, que surge de operar los puntos conocidos:

Luego, para conseguir que los primeros coeficientes de la **expansión de taylor exacta** y la **expansión de taylor aproximada** sean iguales, planteamos el siguiente sistema:

[](https://www.codecogs.com/eqnedit.php?latex=%5Cbegin%7Bcases%7D%20(a%2B1)b%20%2B%20d%20%3D%20%5Cfrac%7B3%7D%7B2%7D%20%5C%5C%20b%20%2B%202d%20%3D%20%5Cfrac%7B21%7D%7B8%7D%20%5C%5C%20%20%5Cfrac%7B1%7D%7B2%7Db%20%2B%202d%20%3D%20%5Cfrac%7B39%7D%7B16%7D%20%5Cend%7Bcases%7D#0)

Resolviendo el sistema y reemplazando en los parámetros correspondientes se pudo obtener que la siguiente ecuación permite aproximar la integral entre x0+h y x0+2.5h:

#### 

#### 1.1.1.2 - ANALISIS DE LA UTILIDAD Y DEL ORDEN DE ERROR

A continuación, se muestran los primeros 10 coeficientes de la expansión de la integral exacta y de la expansión de la aproximación:

* **Coeficientes Expansión exacta:**

[**1.50000000000000 2.62500000000000 2.43750000000000** 1.58593750000000 0.805468750000000 0.337695312500000 0.120903087797619 0.0378194173177083 0.0105095272972470 0.00262779518410012]

* **Coeficientes Expansión aproximada:**

[**1.50000000000000 2.62500000000000 2.43750000000000** 1.56250000000000 0.765625000000000 0.303125000000000 0.100520833333333 0.0286458333333333 0.00715215773809524 0.00158833498677249]

Como se observa, la expansión de la fórmula obtenida en el punto anterior coincide con la expansión exacta hasta el término que contiene a la segunda derivada de f. Si solo se toman los tres primeros términos de la expansión, se puede concluir que el error que se introducirá en el resultado respetara la fórmula:

Siendo el máximo absoluto de la tercera derivada de f en el intervalo de integración y el valor que resulta de la resta de los primeros coeficientes no coincidentes (exacta - aproximada).

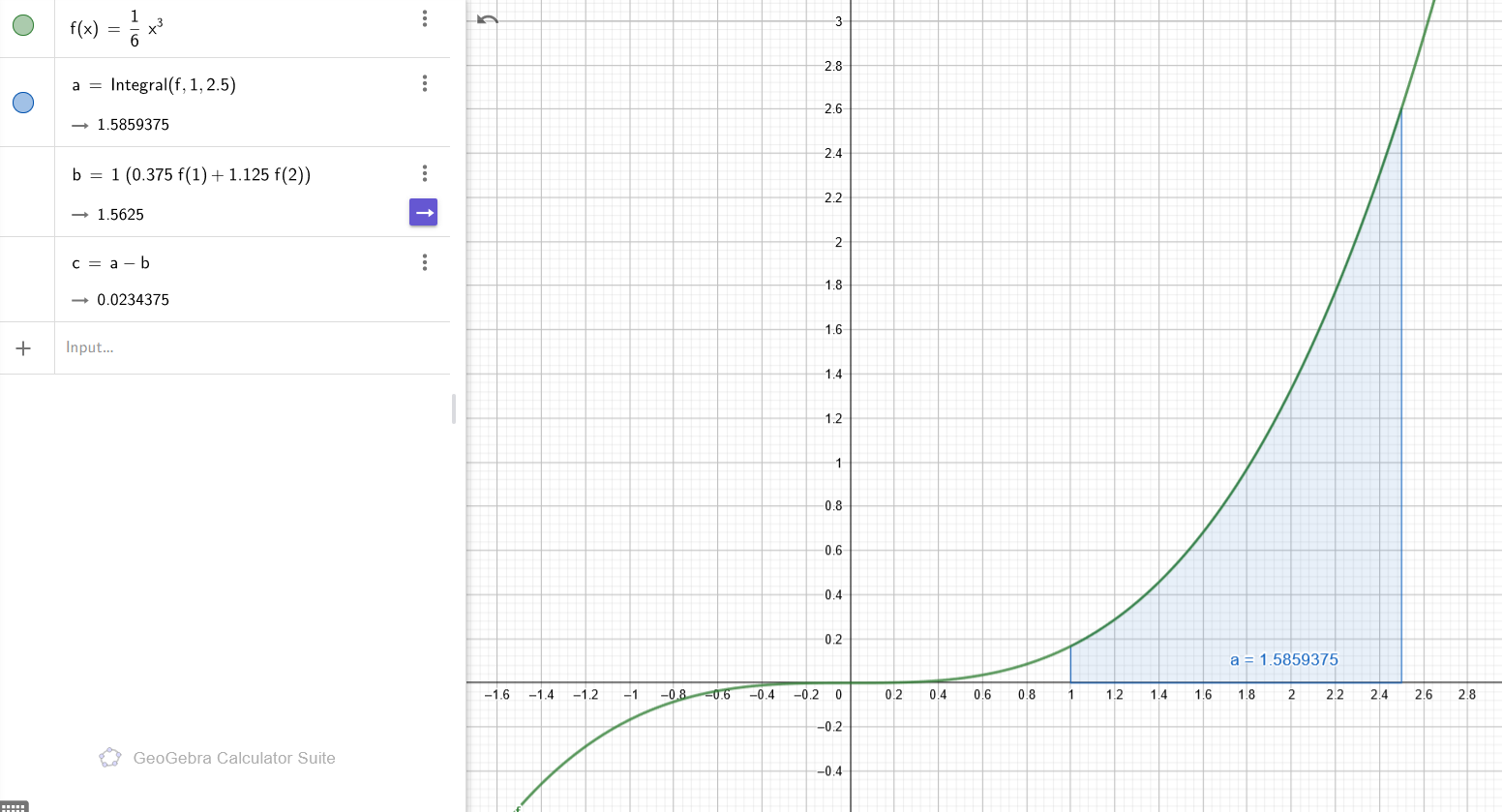


Imagen 1 - Comprobación del coeficiente 3/128 del error aplicando **h = 1** y **x0 = 0** sobre una función con derivada tercera constante **f’’’(x) = 1**

Finalmente de los cálculos realizados se puede concluir:

* Si f(x) es una función polinómica de orden menor que 3, el valor de la integral obtenido a partir de la fórmula estipulada será exacto.
* Como el error depende de , si se reduce el paso a la mitad entonces el error se verá reducido 16 veces.
* Esta fórmula puede ser utilizada como parte de un método para EDOS para predecir el valor de una función en un instante de tiempo futuro, a partir de utilizar dos puntos ya conocidos.
* Comparando esta fórmula con la fórmula abierta que aplica el método de Milne, se cree que la diferencia en el error puede estar dándose porque los puntos que participan en la fórmula semiabierta no se encuentran equiespaciados respecto del extremo superior de la integral (a diferencia de Milne que todos los puntos sí lo están).  
   Otra de las hipótesis es que la diferencia en el error se debe a una cuestión de simetría de las funciones utilizadas para aproximar la integral. La fórmula abierta de Milne utiliza un polinomio de interpolación de orden 2.

### 1.1.2 - FÓRMULA SUPRACERRADA

#### 1.1.2.1 - ANALISIS DE LA EXPANSION DE TAYLOR

Luego de haber desarrollado el análisis para la fórmula semiabierta, se descubrió que era posible generalizar el desarrollo cambiando los valores que multiplican a **h** en los extremos de la integral por parámetros:

Con esto, se implementó un código que dado los valores para **m** y **M**, se puede obtener la ecuación que permite aproximar la integral, utilizando la función getFormulaAproximacion(m,M). Este código realiza el procedimiento desarrollado en 1.1.1.1

#m\*h

#m: multiplicador de h

def generarCoeficientesExpansion(cant, m):

expansion = []

for i in range(0,cant,1):

denominador = sym.factorial(i)

numerador = m\*\*i

expansion.append(numerador/denominador)

return np.array(expansion)

def getDiferenciaExpansionExacta(cant, m\_limit\_inf, m\_limit\_sup):

expansion\_exacta\_lim\_sup = generarCoeficientesExpansion(cant, m\_limit\_sup)

expansion\_exacta\_lim\_inf = generarCoeficientesExpansion(cant, m\_limit\_inf)

expansion\_exacta\_diferencia = expansion\_exacta\_lim\_sup - expansion\_exacta\_lim\_inf

expansion\_exacta\_diferencia = np.delete(expansion\_exacta\_diferencia,0)

return expansion\_exacta\_diferencia

def getFormulaAproximacion(m\_limit\_inf, m\_limit\_sup):

exp\_h = generarCoeficientesExpansion(10, 1)

exp\_2h = generarCoeficientesExpansion(10, 2)

exp\_exacta = getDiferenciaExpansionExacta(11,m\_limit\_inf,m\_limit\_sup)

'''

v1 = [i1 i2 ...]

v2 = [e1 e2 ...]

diff = [f1 f2 ...]

(i1+a)\*b + e1\*d = f1

i2\*b + e2\*d = f2

i3\*b + e3\*d = f3

'''

A =np.array([ [float(exp\_h[1]), float(exp\_2h[1])],

[float(exp\_h[2]), float(exp\_2h[2])] ])

b = np.array([ float(exp\_exacta[1]), float(exp\_exacta[2]) ])

x = np.linalg.solve(A,b)

a = ((exp\_exacta[0] - x[1]\*exp\_2h[0]) / x[0]) - exp\_h[0]

exp\_h[0]+=a

exp\_h \*= x[0]

exp\_2h \*=x[1]

print("Coeficientes Expansion exacta:",exp\_exacta)

print("Coeficientes Expansion aproximada:",exp\_h+exp\_2h)

if a != 0:

print("h \* ( "+str(x[0])+"\*(f(x0+h) + "+str(a)+"\*f(x0)) + "+str(x[1])+"\*f(x0+2h) )\n")

else:

print("h \* ( "+str(x[0])+"\*f(x0+h) + " +str(x[1])+"\*f(x0+2h) )")

Para este caso, la ecuación que permite aproximar la integral entre x0+0.5h y x0+1.5h resulta:

#### 1.1.2.2 - ANALISIS LA UTILIDAD Y DEL ORDEN DE ERROR

* **Coeficientes Expansión exacta:**

[**1.00000000000000 1.00000000000000 0.541666666666667 0.208333333333333** 0.0630208333333333 0.0157986111111111 0.00338851686507937 0.000635540674603175 0.000105934210345018 1.58906697806437e-5]

* **Coeficientes Expansión aproximada:**

[**1.00000000000000 1.00000000000000 0.541666666666667 0.208333333333333** 0.0659722222222222 0.0187500000000000 0.00497685185185185 0.00124007936507936 0.000287285052910053 6.13150352733686e-5]

Como se observa, la expansión de la fórmula obtenida en el punto anterior coincide con la expansión exacta hasta el término que contiene a la tercera derivada de f. Si solo se toman los cuatro primeros términos de la expansión, se puede concluir que el error que se introducirá en el resultado respetara la fórmula:

Siendo el máximo absoluto de la cuarta derivada de f en el intervalo de integración y el valor que resulta de la resta de los primeros coeficientes no coincidentes (exacta - aproximada).

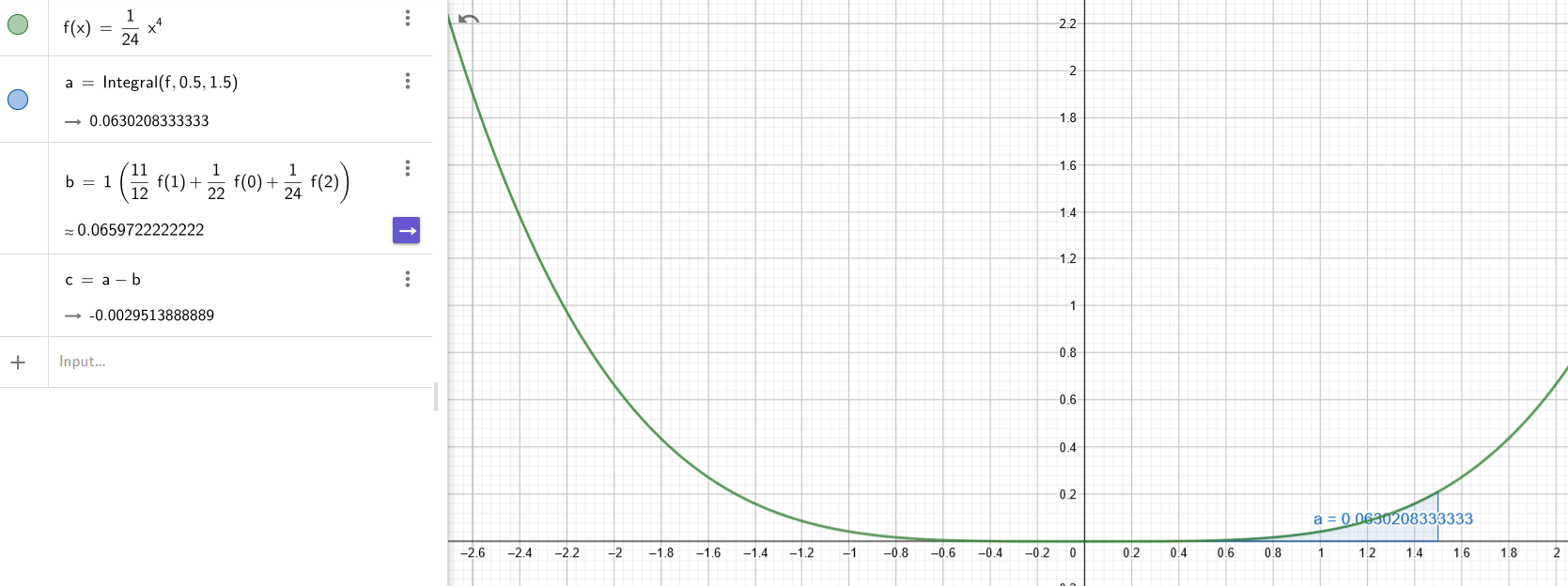


Imagen 2 - Comprobación del coeficiente -17/5760 del error aplicando **h = 1** y **x0 = 0** sobre una función con derivada cuarta constante **f’’’’(x) = 1**

Finalmente de los cálculos realizados se puede concluir:

* Si f(x) es una función polinómica de orden menor que 4, el valor de la integral obtenido a partir de la fórmula estipulada será exacto.
* Como el error depende de , si se reduce el paso a la mitad entonces el error se verá reducido 32 veces, el doble que en el caso de la fórmula semiabierta.
* Esta fórmula puede ser utilizada como parte de un método para EDOS para hallar el valor de una función en un instante de tiempo intermedio entre dos puntos ya conocidos.
* Como el tramo a integrar en este método queda contenido entre los puntos conocidos, se considera razonable que el error sea menor respecto a la fórmula semiabierta. Esto puede deberse a que en este caso resulta mucho más “sencillo” asegurar la convergencia del método.

## 1.2 - MÉTODO PARA EDOS

A continuación, se detallan los pasos del método de resolución para problemas de valor inicial y el orden del error que cada uno paso introduce en la solución:

1. Para comenzar las iteraciones del método PCS (Predictor-Corrector-Supracorrector) se necesitan cuatro puntos iniciales equiespaciados. Este inconveniente puede ser resuelto realizando 3 iteraciones de un Método Runge-Kutta de orden 4, ya que los procedimientos de este tipo son de autoarranque. Un método R-K de orden 4 introduce un error del orden por paso, con lo cual el orden acumulado de las 3 iteraciones termina resultando del orden
2. Luego, se utiliza la fórmula semiabierta desarrollada en el punto 1.1.1 para predecir el valor del siguiente punto en la iteración. Este paso introduce un error del orden de .
3. Una vez hallado se utiliza la fórmula cerrada del método de Milne (N-C de orden 3) para corregir el valor hallado anteriormente y mejorar la aproximación. Este paso introduce un error del orden de .
4. Por último, se utiliza este valor corregido para supracorregir la aproximación de la iteración previa aplicando la fórmula supracerrada desarrollada en el punto 1.1.2. Este paso introduce un error del orden de .



Imagen 3 - Iteración del método PCS

## 1.3 - APLICACIÓN A ECG

### 1.3.1 - OBTENCIÓN DE LA APROXIMACIÓN Y DIAGRAMA V-W

Para este inciso, como paso inicial se realizó una aproximación del sistema utilizando únicamente el método Runge-Kutta de orden 4. Esto se hizo para tener un punto de referencia con el que comparar y verificar el método PCS.

#aproxima el siguiente paso

def runge\_kutta4(m, paso):

v = m[-1][1]

w = m[-1][2]

k1v = 3\*(v+ w - (v\*\*3)/3)

k1w = -1/3\*(w - 0.7 + 0.8\*v)

k2v = 3\*( (v+k1v\*paso/2) + (w+k1w\*paso/2) - ((v+k1v\*paso/2)\*\*3)/3)

k2w = -1/3\*((w+k1w\*paso/2) - 0.7 + 0.8\*(v+k1v\*paso/2))

k3v = 3\*( (v+k2v\*paso/2) + (w+k2w\*paso/2) - ((v+k2v\*paso/2)\*\*3)/3)

k3w = -1/3\*((w+k2w\*paso/2) - 0.7 + 0.8\*(v+k2v\*paso/2))

k4v = 3\*( (v+k3v\*paso) + (w+k3w\*paso) - ((v+k3v\*paso)\*\*3)/3)

k4w = -1/3\*((w+k3w\*paso) - 0.7 + 0.8\*(v+k3v\*paso))

v\_sig = v + 1/6\*paso\*(k1v + 2\*k2v + 2\*k3v + k4v)

w\_sig = w + 1/6\*paso\*(k1w + 2\*k2w + 2\*k3w + k4w)

return np.append(m,[[m[-1][0]+paso, v\_sig, w\_sig, 3\*(v\_sig+ w\_sig - (v\_sig\*\*3)/3), -1/3\*(w\_sig - 0.7 + 0.8\*v\_sig)]],axis=0)

Luego se aplicó el método PCS para el cálculo. Para su implementación, se decidió utilizar una matriz donde cada fila contiene los valores de **V,** **W** y sus derivadas para un instante **t**. Además de esto, fue necesario generalizar este método de manera equivalente a la vista en clases para el método Runge-Kutta de orden 4, teniendo que realizar los cálculos para cada ecuación diferencial del sistema.

m = [

[t0 v0 w0 v'0 w'0]

[t1 v1 w1 v'1 w'1]

[t2 v2 w2 v'2 w'2]

[t3 v3 w3 v'3 w'3]

......................

]

def metodo\_pcs(m,paso):

t = m[-1][0]

#predictor

vp = m[-3][1] + paso \* (0.375\*m[-3][3] + 1.125\*m[-1][3])

wp = m[-3][2] + paso \* (0.375\*m[-3][4] + 1.125\*m[-1][4])

dvp = 3\*(vp+ wp - (vp\*\*3)/3)

dwp = -1/3\*(wp - 0.7 + 0.8\*vp)

#corrector

vc = m[-2][1] + 0.5\*(paso/3)\*(m[-2][3] + 4\*m[-1][3] + dvp)

wc = m[-2][2] + 0.5\*(paso/3)\*(m[-2][4] + 4\*m[-1][4] + dwp)

dvc = 3\*(vc+ wc - (vc\*\*3)/3)

dwc = -1/3\*(wc - 0.7 + 0.8\*vc)

m = np.append(m, [[m[-1][0]+paso\*0.5, vc, wc, dvc, dwc]], axis=0)

#supracorrector

vsp = m[-4][1] + paso\*( (11/12)\*(m[-3][3]+(1/22)\*m[-5][3]) + (1/24)\*dvc)

wsp = m[-4][2] + paso\*( (11/12)\*(m[-3][4]+(1/22)\*m[-5][4]) + (1/24)\*dwc)

dvsp = 3\*(vsp+ wsp - (vsp\*\*3)/3)

dwsp = -1/3\*(wsp - 0.7 + 0.8\*vsp)

m[-2][1] = vsp

m[-2][2] = wsp

m[-2][3] = dvsp

m[-2][4] = dwsp

return m, vc-vsp, wc-wsp

paso = 0.002

matriz = np.array([[0,0,0,0,7/30]])

v = []

w = []

t = []

error\_v = []

error\_w = []

# 3 iteraciones de RK

for i in range(3):

v.append(matriz[-1][1])

w.append(matriz[-1][2])

t.append(matriz[-1][0])

matriz = runge\_kutta4(matriz,0.5\*paso)

# metodo PCS

for i in range(10000):

v.append(matriz[-1][1])

w.append(matriz[-1][2])

t.append(matriz[-1][0])

matriz,ev,ew = metodo\_pcs(matriz,paso)

error\_v.append(ev)

error\_w.append(ew)

Al obtener la aproximación y graficar V y W en función del tiempo, se logra observar en la imagen 4 que el sistema se estabiliza con V tendiendo a un valor de 1.44 y W tendiendo a un valor de -0.45 aproximadamente. Esta estabilización del sistema produce en el diagrama de estado, a medida que avanza el tiempo, que los puntos se concentren en la misma zona de la gráfica haciendo parecer que ésta finaliza en ese punto.

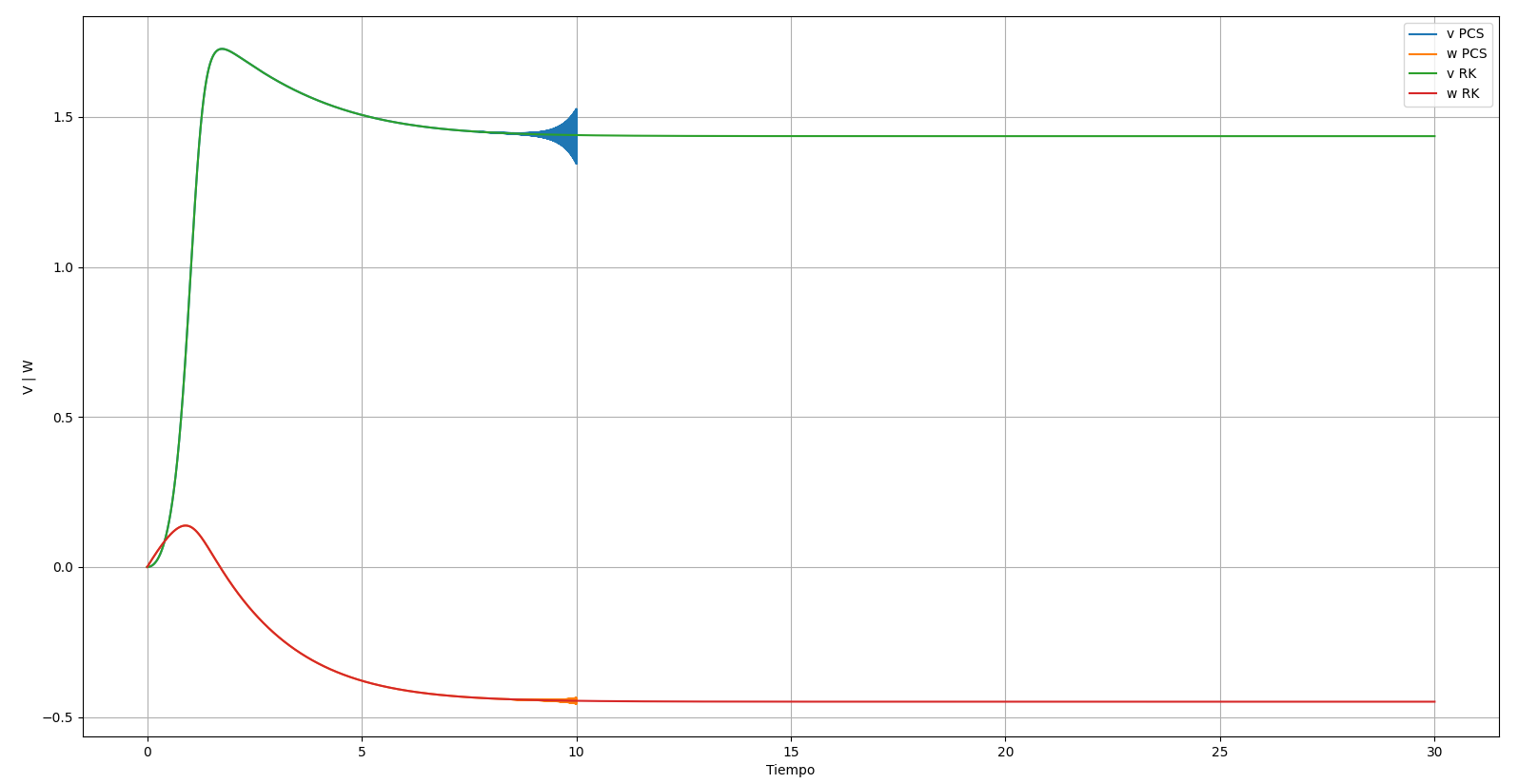


Imagen 4 - V y W en función del tiempo.

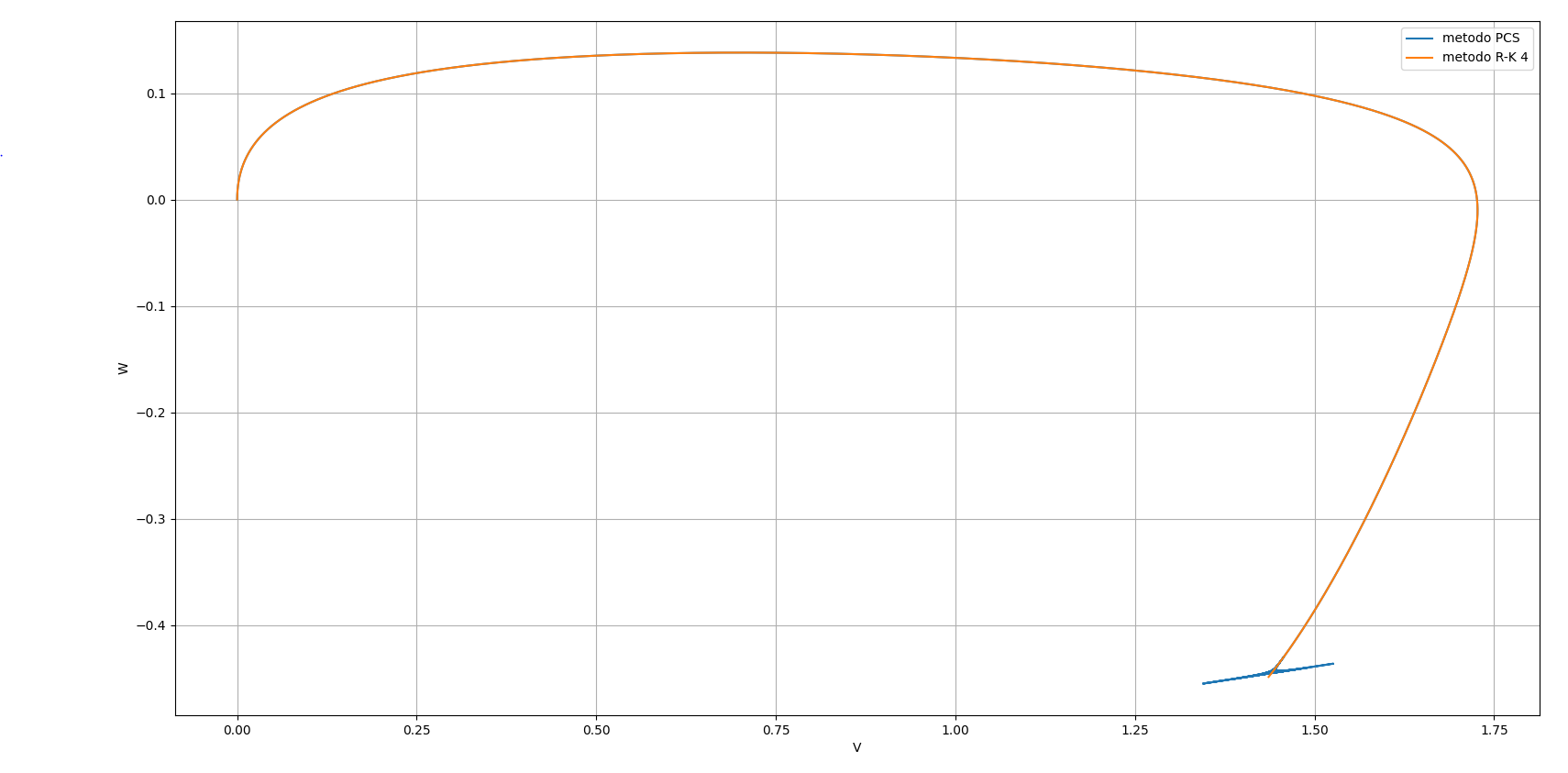


Imagen 5 - Diagrama de estado V-W.

De esto, se puede observar lo siguiente:

* Se observa que el método PCS avanza más rápido en las iteraciones que el R-K. Esto es debido a que el primero utiliza valores guardados en memoria para calcular el siguiente paso, mientras que el R-K necesita evaluar 4 veces la derivada para calcular el nuevo punto.
  + Se realizó un análisis de los métodos para tratar de mostrar la diferencia en los tiempos de ejecución, pero no resultó muy apreciable ya que los tiempos obtenidos son muy similares. Se cree que esto puede deberse a que las ecuaciones diferenciales de este sistema son fáciles de evaluar. Probablemente para sistemas con ecuaciones complejas, el método PCS tardaría un tiempo considerablemente menor en comparación al R-K.
* Debido a que el método PCS utiliza puntos precalculados en anteriores iteraciones, este resulta más propenso a diverger y/o oscilar que el método Runge-Kutta de orden 4. Esto puede ser debido a que el error insertado en los puntos se va acumulando. Esto se puede visualizar en la imagen 4 a partir de t = 8 aproximadamente.
* Un cambio que podría hacerse al programa para aprovechar los beneficios de velocidad de avance ofrecidos por el PCS y reduciendo sus problemas de divergencia y/o oscilaciones, es utilizar el metodo PCS en las primeras iteraciones hasta que comience a diverger y a partir de ese punto pasar a utilizar un metodo R-K.

### 1.3.2 - ESTIMADOR DE RICHARDSON PARA EL MÉTODO

Para tener una idea del error, se decidió generar un estimador de Richardson haciendo uso de la fórmula correctora y la supracorrectora de la misma manera que se había desarrollado el estimador para el método de milne en la clase de práctica.

Suponiendo que y restando miembro a miembro ambas ecuaciones se obtiene que:

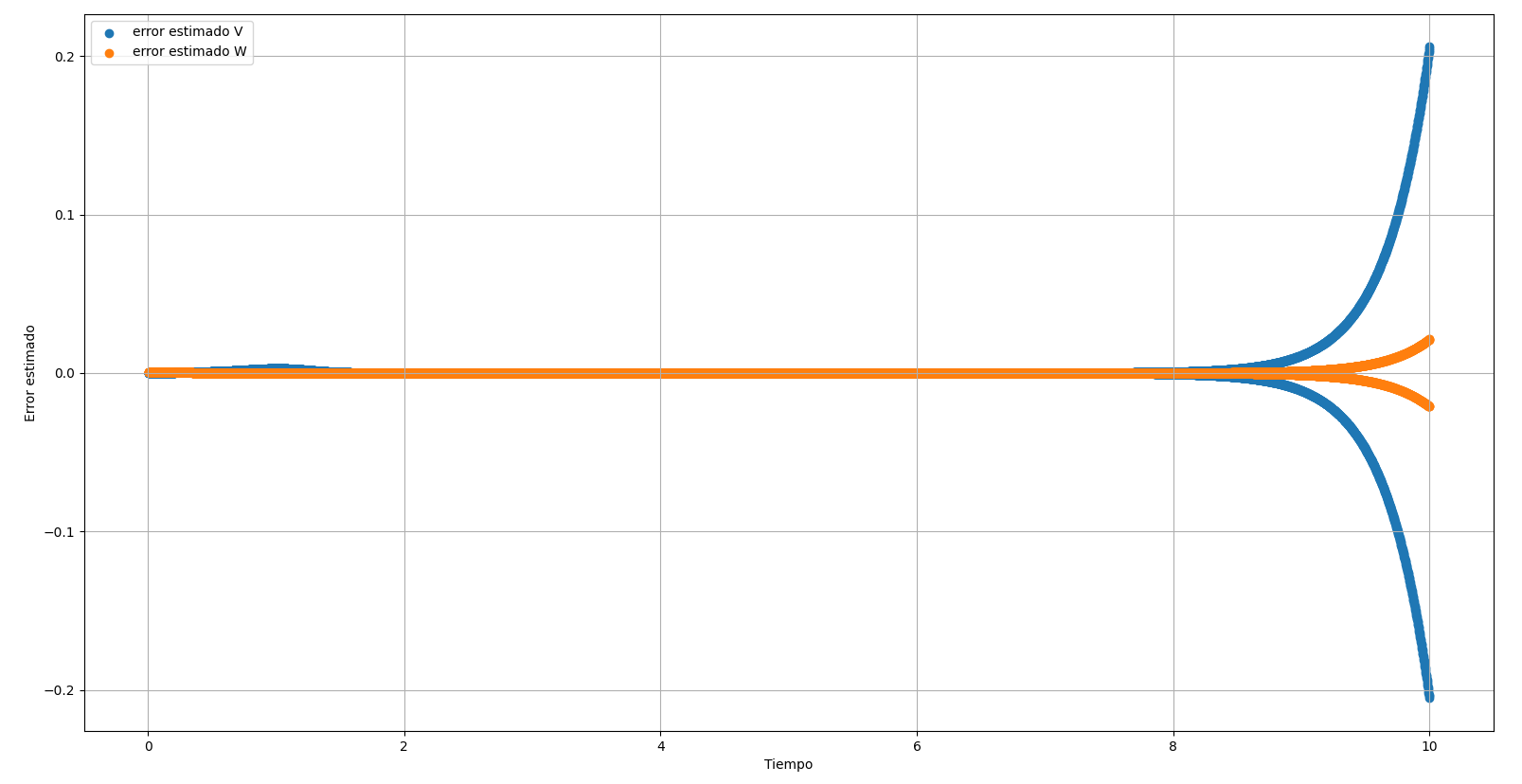
Luego, despejando :

Este estimador permite tener una noción del orden del error que se está introduciendo en cada paso del método PCS y poder visualizar el signo del mismo.

### 1.3.3 - OBSERVACIONES Y CONCLUSIONES

En la siguiente imagen se puede ver la oscilación del método a medida que transcurre el tiempo, ilustrada por los valores del estimador de Richardson. Este estimador podría usarse para agregar una condición de parada al método, donde la idea es establecer una tolerancia máxima para este parámetro en valor absoluto, de modo tal, que si se supera dicho valor se detenga el método para evitar que diverja.

Durante las primeras 8 unidades de tiempo el método PCS logra obtener una aproximación muy parecida a la obtenida con el método RK-4, donde el error estimado es pequeño. Pasadas las 8 unidades de tiempo, como se mencionó previamente, el método comienza a oscilar entre los valores obtenidos de la fórmula corregida y la fórmula supracorregida, produciendo que, al pasar el tiempo, las oscilaciones sean cada vez mayores (y consecuentemente que el método diverja).



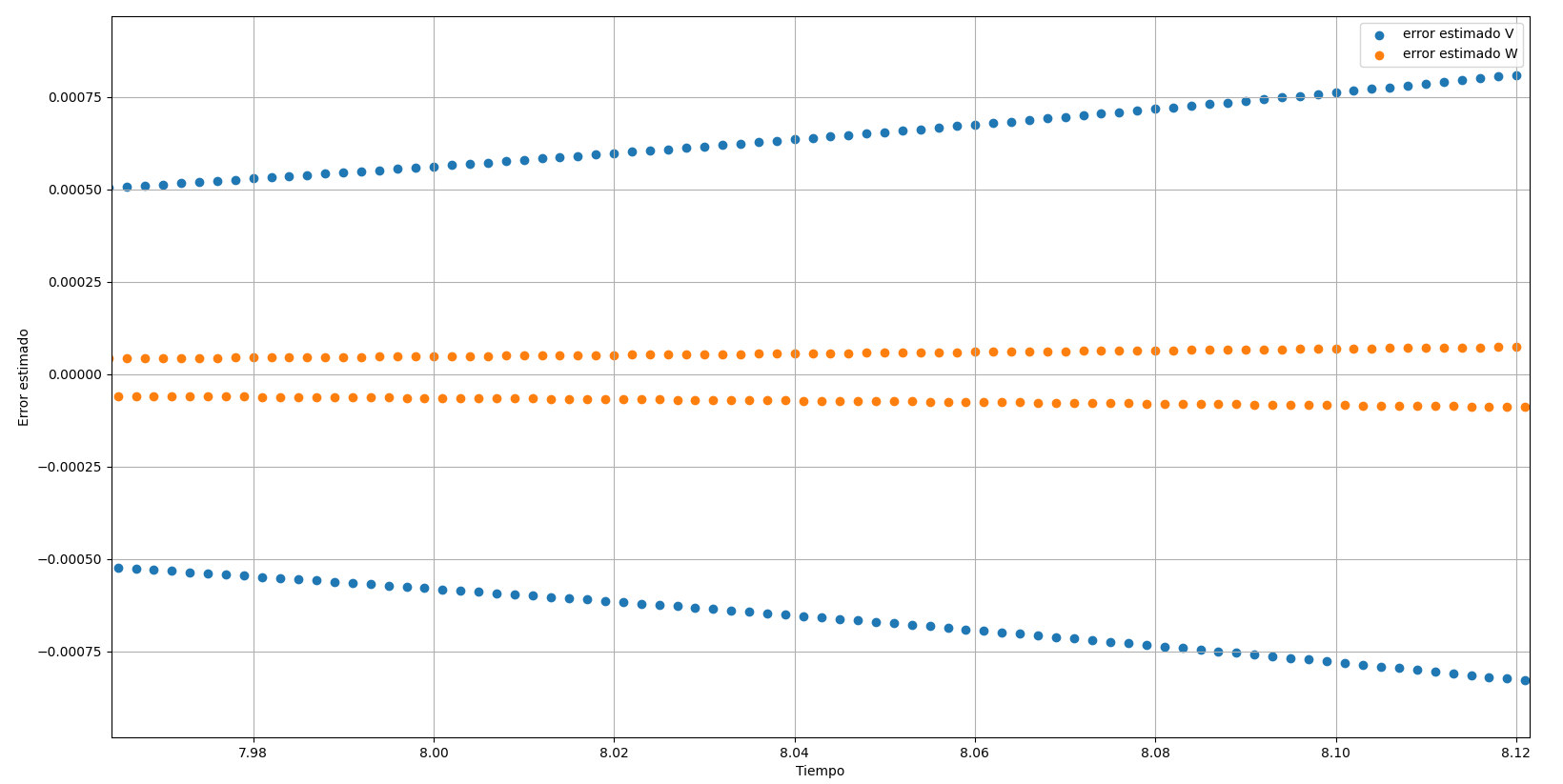


Imagen 6 - Estimador de Richardson para el método PCS.

En conclusión, no existe método que sea mejor que otro para aproximar sistemas de ecuaciones diferenciales. La mejor opción sería combinar diferentes métodos para aprovechar las ventajas de cada uno. Como en este caso, se podría aplicar el método PCS hasta el tiempo 8 aprovechando la velocidad que brinda, y luego seguir con RK-4 que a pesar de ser más lento, no diverge.

Por otro lado, utilizar el estimador de Richardson desarrollado en el inciso anterior como condición de parada del método introduciría las siguientes ventajas:

* No se necesitan calcular las derivadas de la función para hallar su valor.
* Debido a cómo funcionan los pasos en el método, ya se tienen calculados los valores de las imágenes corregidas y supra corregidas en memoria. Por ende, no es necesario recalcular los puntos para hallar el valor del estimador.
* Además de utilizar el valor de los estimadores como condición de parada, se podría restar el valor obtenido por el estimador de **V** y **W** a las aproximaciones calculadas en cada iteración para disminuir el error introducido por aplicar el método.

## 1.4 - MODELO CARDIACO 2D

Para automatizar el proceso de creación de la malla, se decidió leer la imagen dada en el enunciado pixel por pixel analizando los colores y, en base a esto, crear una matriz (un caracter indica la inicial del color) en donde cada celda indica el color de cada nodo de la malla.

Para poder ir variando la cantidad de nodos de la malla, se subdivide la imagen en sub-regiones según un paso en píxeles. Es decir, si el paso es h = 1, la malla tendrá la misma cantidad de nodos que la cantidad de píxeles que tiene la imagen. Si el paso es h = 10, se subdivide la imagen en cuadrantes de 10x10 pixeles y se coloca como color predominante en la nueva matriz aquel que se repita una mayor cantidad de veces en la sub-region.

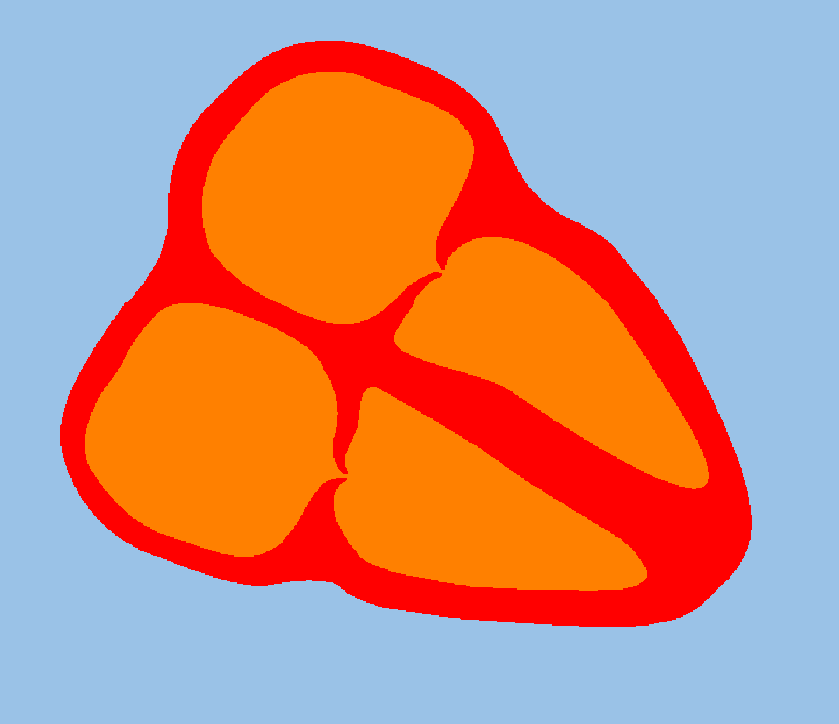


Imagen 7 - Malla con paso h=1 (izquierda), malla con paso h=10 (centro), malla con paso h=30 (derecha)

Luego, se crea la matriz inicial de temperaturas, en donde todos los nodos pertenecientes al corazón se encuentran a 37 °C y el resto a -150 °C. Para el cálculo de las aproximaciones en el tiempo siguiente, se decidió optar por el uso del método que aplica diferencias adelantadas por su simplicidad de implementación. Para los puntos fuera de los bordes, se considera que se encuentran a -150 °C y que su temperatura es constante durante toda la simulación.

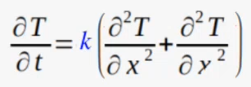
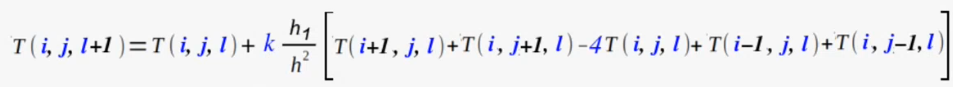
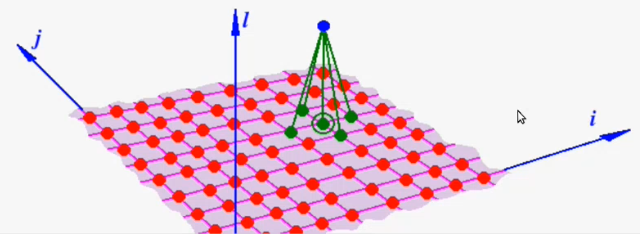


Imagen 8 - Método de diferencias finitas adelantadas

Donde el valor de **k** estará dado en función del color que se encuentre indicado en la matriz de colores del nodo que se esté calculando. Esta decisión fue tomada para facilitar la implementación de la fórmula y teniendo en cuenta que se está utilizando en una discretización del problema real, donde la continuidad del sistema asigna un **k** único a cada punto.

def calcularTemperaturaSiguienteTiempo(malla, temperaturas, pasoEspacial, pasoTemporal):

ans = []

enfriado = True

for i in range(len(malla)):

aux = []

for j in range(len(malla[0])):

coef\_k = 1

if malla[i][j] == "naranja":

coef\_k = 0.8

elif malla[i][j] == "rojo":

coef\_k=0.5

temp = temperaturas[i][j]

coef = coef\_k\*pasoTemporal/(pasoEspacial\*\*2)

#centro

temp += coef\*-4\*temperaturas[i][j]

#arriba

if i==0:

temp += coef\*-150

else:

temp += coef\*temperaturas[i-1][j]

#abajo

if i==len(malla)-1:

temp += coef\*-150

else:

temp += coef\*temperaturas[i+1][j]

#izq

if j==0:

temp += coef\*-150

else:

temp += coef\*temperaturas[i][j-1]

#der

if j==len(malla[0])-1:

temp += coef\*-150

else:

temp += coef\*temperaturas[i][j+1]

if temp >= 0:

enfriado = False

aux.append(temp)

ans.append(aux)

return ans, enfriado

Como la fórmula requiere del paso espacial (h en la fórmula) en metros, se obtuvo un conversor de píxeles a metros para realizar la aproximación del modelo en función del tamaño de un corazón real. Este conversor posee un valor de .

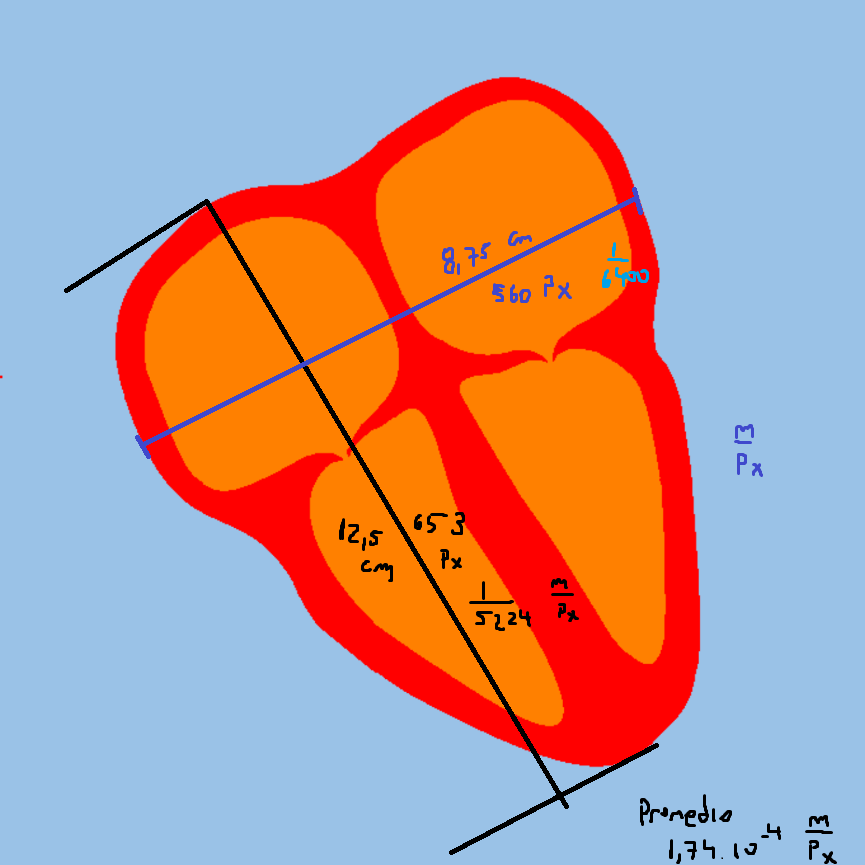


Imagen 9 - Cálculo del conversor de píxeles a metros.

Se observó que debido a que el paso espacial resulta muy pequeño en metros, el valor del coeficiente que multiplica a k se dispara rápidamente causando que la simulación falle por overflow. Por esta razón, para poder resolver este inconveniente es necesario disminuir el valor del espacio temporal o aumentar el paso espacial.

Luego de varias simulaciones, se obtuvo la siguiente tabla se muestran los tiempos de ejecución requeridos en función del paso espacial y temporal:

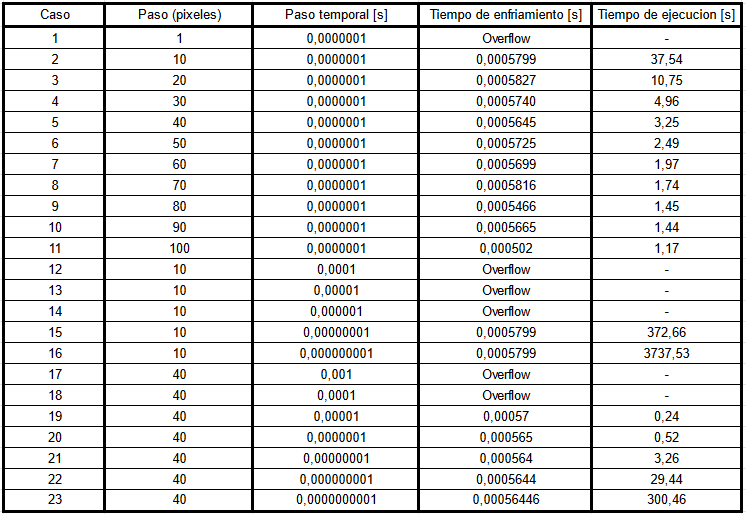


Tabla 1 - tiempos de ejecución requeridos en función del paso espacial y temporal

El paso en pixeles fue elegido “grande” para la mayoría de las simulaciones ya que utilizando un paso menor que 10, el método diverge (se produce overflow). Esto podría haberse solucionado reduciendo el paso temporal, pero los tiempos de ejecución se volvían extremadamente largos.

El tiempo requerido para que todo el modelo del corazón se enfríe por debajo de 0 °C ronda entre 0,0005 y 0,0006 segundos aproximadamente (0,6 milisegundos). Se cree que el tiempo resulta muy pequeño por la baja temperatura a la que se encuentra el líquido al sumergir el corazón (-150 °C). Además, se cree que si el modelo fuera en tres dimensiones en vez de dos, el tiempo sería considerablemente mayor.

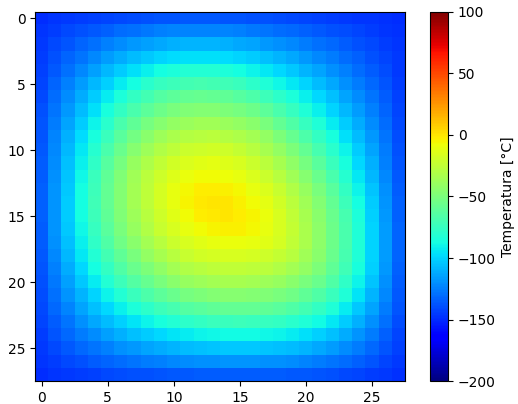
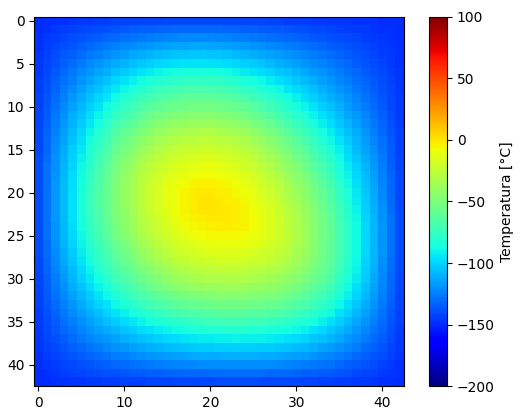
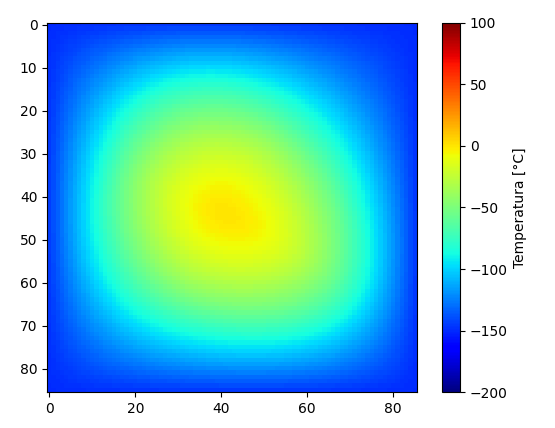


Imagen 10 - Estado de la malla luego de enfriar el corazón bajo 0 °C, con paso h=1 (izquierda), con paso h=10 (centro), con paso h=30 (derecha)

La diferencia entre los resultados obtenidos con los diferentes pasos en pixeles no resulta evidente (dejando de lado los tiempos de ejecución), ya que no se logra obtener una relación entre el paso y el valor del tiempo de enfriamiento del modelo. Se cree que esto se debe por la forma en que son seleccionados los nodos en la malla. Al subdividir la imagen y seleccionar por cada región el color predominante, se está realizando un estilo de aproximación de la matriz real, obteniendo quizás un mejor resultado que si se tomarían los nodos cada cierto valor (ignorando los nodos que se encuentren entremedio).