Universidad Nacional de Colombia

Facultad de Ingeniería

Proyecto final

Métodos Numéricos
I Semestre de 2021

Autores:

Nicolás Darío Mejía Borda
nmejiab@unal.edu.co

Juan Sebastián Rodríguez Castellanos
juarodriguezc@unal.edu.co

Julio 27 de 2021

I. Objetivo

Con el fin de encontrar la ecuación de una función mostrada en una imagen, se realizará un software que utilizando visión artificial, encontrará los distintos puntos del gráfico. Luego, estos puntos serán transformados al sistema de coordenadas cartesianas de modo que puedan ser interpolados. Los tres métodos de interpolación serán: interpolación polinomial, interpolación de Lagrange e interpolación por splines cúbicos. El software contará también con la capacidad de graficar estas funciones y retornar el valor de estas mismas para que puedan ser graficadas en otras aplicaciones de software, tales como Geogebra.

II. Marco teórico

II-a Interpolación.

Dentro del contexto del análisis numérico, interpolación es un método por el cual basado en un conjunto de puntos, se obtienen nuevos siguiendo un patrón trazado por el conjunto anteriormente mencionado.

II-b Interpolación polinomial.

La interpolación polinomial consiste en determinar el polinomio único de grado n que se ajuste a $n+1$ puntos. Este polinomio, entonces, proporciona una fórmula para calcular valores intermedios.[1]
Teniendo en cuenta que la ecuación general de un polinomio de grado n es

$$y = a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n \tag{1}$$

Dados los puntos $p_0, p_1, p_2, ..., p_n$ descritos por las coordenadas cartesianas $(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, se obtendrá un polinomio de grado n, y pasará por cada uno de ellos, empezando por p_0, se sabe que cuando $x = x_0$ entonces $y = y_0$ por ende se puede sustituir x e y por x_0 e y_0 respectivamente , dejando así la ecuación del polinomio:

$$y_0 = a_0x_0^n + a_1x_0^{n-1} + a_2x_0^{n-2} + ... + a_{n-1}x_0 + a_n \tag{2}$$

si se hace esto para cada uno de los puntos se obtiene un sistema de $n+1$ ecuaciones con $n+1$ incógnitas de la forma:

$$\begin{aligned}
y_0 &= a_0x_0^n + a_1x_0^{n-1} + a_2x_0^{n-2} + ... + a_{n-1}x_0 + a_n \\
y_1 &= a_0x_1^n + a_1x_1^{n-1} + a_2x_1^{n-2} + ... + a_{n-1}x_1 + a_n \\
y_2 &= a_0x_2^n + a_1x_2^{n-1} + a_2x_2^{n-2} + ... + a_{n-1}x_2 + a_n \\
&\cdots \\
y_{n-1} &= a_0x_{n-1}^n + a_1x_{n-1}^{n-1} + a_2x_{n-1}^{n-2} + ... + a_{n-1}x_{n-1} + a_n \\
y_n &= a_0x_n^n + a_1x_n^{n-1} + a_2x_n^{n-2} + ... + a_{n-1}x_n + a_n
\end{aligned} \tag{3}$$

Este mismo puede ser representado en su matriz ampliada de forma:

$$\begin{pmatrix}
x_0^n & x_0^{n-1} & x_0^{n-2} & \cdots & x_0 & 1 \\
x_1^n & x_1^{n-1} & x_1^{n-2} & \cdots & x_1 & 1 \\
x_2^n & x_2^{n-1} & x_2^{n-2} & \cdots & x_2 & 1 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
x_{n-1}^n & x_{n-1}^{n-1} & x_{n-1}^{n-2} & \cdots & x_{n-1} & 1 \\
x_n^n & x_n^{n-1} & x_n^{n-2} & \cdots & x_n & 1
\end{pmatrix}
\begin{pmatrix}
y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n
\end{pmatrix} \tag{4}$$

Al ser resuelto este sistema,se obtienen los valores de $a_0, a_1, a_2, ..., a_{n-1}, a_n$ con los cuales se puede encontrar el polinomio interpolador.

II-c Interpolación de Lagrange.

La interpolación de Lagrange es un método numérico usado para interpolar que se basa en encontrar un polinomio que pase por todos los puntos $p_0, p_1, p_2, ..., p_n$ descritos por las coordenadas $(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$. Este polinomio se halla mediante la fórmula:

$$P(x) = \sum_{i=0}^{n} L_i \cdot f(x_i) \tag{5}$$

donde:

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \tag{6}$$

II-d Interpolación por splines cúbicos.

La interpolación por splines cúbicos es un tipo de interpolación que busca generar una función a trozos compuesta por n polinomios de orden 3 siendo $n+1$ la cantidad de puntos, cada uno de estos polinomios conecta dos puntos adyacentes; si se tienen $p_0, p_1, p_2, ..., p_n$ puntos se buscarán $f_1(x), f_2(x), ..., f_n(x)$ polinomios de forma que $f_i(x)$ conectará a p_{i-1} con p_i para i desde 1 hasta n.
Si $f_1(x)$ conectará a p_0 con p_1 entonces:

$$\begin{aligned} y_0 = f_1(x_0) = a_1 \cdot x_0^3 + b_1 \cdot x_0^2 + c_1 \cdot x_0 + d_1 \\ y_1 = f_1(x_1) = a_1 \cdot x_1^3 + b_1 \cdot x_1^2 + c_1 \cdot x_1 + d_1 \end{aligned} \tag{7}$$

Haciendo esto para cada una de las funciones obtendremos:

$$\begin{aligned} y_0 = f_1(x_0) = a_1 \cdot x_0^3 + b_1 \cdot x_0^2 + c_1 \cdot x_0 + d_1 \\ y_1 = f_1(x_1) = a_1 \cdot x_1^3 + b_1 \cdot x_1^2 + c_1 \cdot x_1 + d_1 \\ y_1 = f_2(x_1) = a_2 \cdot x_1^3 + b_2 \cdot x_1^2 + c_2 \cdot x_1 + d_2 \\ y_2 = f_2(x_2) = a_2 \cdot x_2^3 + b_2 \cdot x_2^2 + c_2 \cdot x_2 + d_2 \\ \dots \\ y_{n-1} = f_n(x_{n-1}) = a_n \cdot x_{n-1}^3 + b_n \cdot x_{n-1}^2 + c_n \cdot x_{n-1} + d_n \\ y_n = f_n(x_n) = a_n \cdot x_n^3 + b_n \cdot x_n^2 + c_n \cdot x_n + d_n \end{aligned} \tag{8}$$

De aquí se obtienen $2n$ ecuaciones. Ahora para cada par de funciones adyacentes la primera y segunda derivada deberán ser la misma en el punto que comparten:

$$\begin{aligned} f_1'(x_1) = f_2'(x_1), f_1''(x_1) = f_2''(x_1) \\ f_2'(x_2) = f_3'(x_2), f_2''(x_2) = f_3''(x_2) \\ f_3'(x_3) = f_4'(x_3), f_3''(x_3) = f_4''(x_3) \\ f_4'(x_4) = f_5'(x_4), f_4''(x_4) = f_5''(x_4) \\ \dots \\ f_{n-2}'(x_{n-2}) = f_{n-1}'(x_{n-2}), f_{n-2}''(x_{n-2}) = f_{n-1}''(x_{n-2}) \\ f_{n-1}'(x_{n-1}) = f_n'(x_{n-1}), f_{n-1}''(x_{n-1}) = f_n''(x_{n-1}) \end{aligned} \tag{9}$$

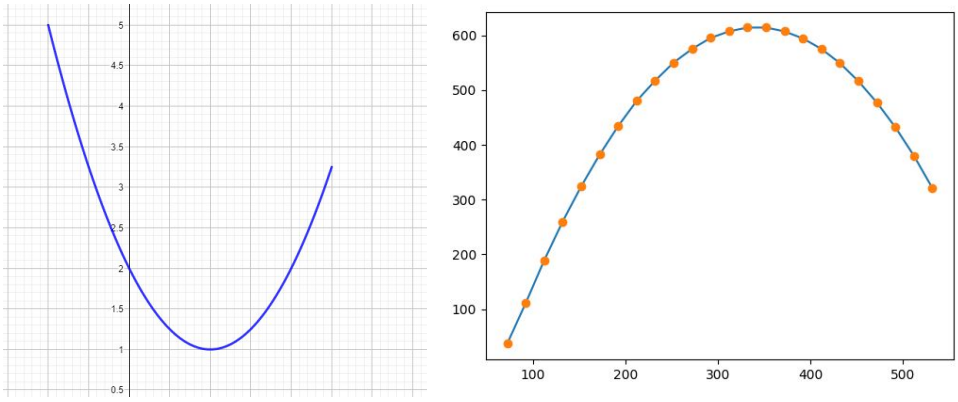
Estas últimas son $2n-2$ ecuaciones, en total hay $4n-2$ ecuaciones, las últimas dos ecuaciones son arbitrarias, pero se recomienda que estas sean:

$$\begin{aligned} f_1''(x_0) = 0 \\ f_n''(x_n) = 0 \end{aligned} \tag{10}$$

Con estas ya se obtiene un sistema de $4n$ ecuaciones para $4n$ incógnitas que al resolverlo retornará los valores de $a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2, a_3, b_3, c_3, d_3, ..., a_{n-1}, b_{n-1}, c_{n-1}, d_{n-1}, a_n, b_n, c_n, d_n$, con los cuales su pueden hallar las funciones que interpolan los puntos.

II-e OpenCV.

Es una librería software de visión artificial y machine learning. La cual permite identificar objetos, caras, clasificar acciones, identificar colores, entre muchas otras aplicaciones. En este caso se utiliza para identificar los puntos de la imagen y obtener un aproximado a los puntos de la función. Este reconocimiento se hace al seleccionar los puntos que estén dentro de un rango de color, separando los que se encuentren por fuera.
La imagen seleccionada es mostrada a continuación, junto al reconocimiento dado por la aplicación:



(a) (b)

Figura 1: Imagen y puntos reconocidos por openCV

Sin embargo, los puntos mostrados están en función de los pixeles de la imagen, por lo que es necesario transformarlos para que estén en una escala más pequeña y en la dirección correcta, para esto se aplica:

$$T\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x/100 \\ (-y+600)/100 \end{pmatrix} \tag{11}$$

De esta forma se tienen los puntos para interpolar, obteniendo el siguiente resultado:

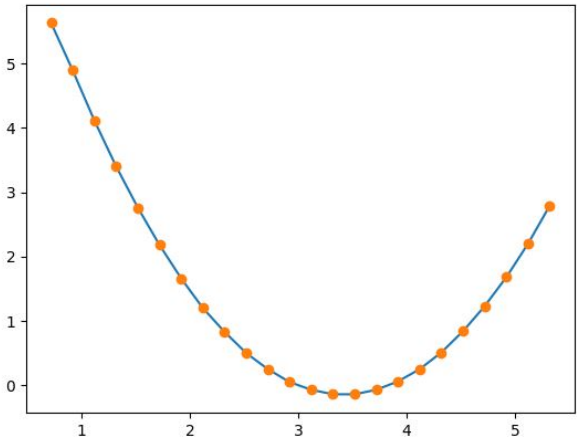


Figura 2: Puntos transformados para su interpolación

Pero, tal como se mencionó en interpolación polinomial e interpolación de Lagrange, al interpolar $n+1$ puntos se obtiene un polinomio de grado n, por lo que es necesario seleccionar una cantidad más pequeña de puntos, obteniendo una función más precisa.

II-f Ajuste de coordenadas.

Con openCV se obtiene una cantidad determinada de puntos los cuales se pueden interpolar, sin embargo, estos puntos no son precisos respecto a la función real a la que se plantea llegar, para llegar a una mejor aproximación es necesario tener por lo menos dos puntos de la función esperada. Por lo que teniendo (x_{t1}, y_{t1}) como mínimo absoluto y (x_{t2}, y_{t2}) como máximo absoluto de la función esperada, los demás puntos se pueden obtener de la siguiente manera:

$$\text{Sean } T\begin{pmatrix}x_1\\y_1\end{pmatrix} = \begin{pmatrix}x_{t1}\\y_{t1}\end{pmatrix} \qquad \text{y} \qquad T\begin{pmatrix}x_2\\y_2\end{pmatrix} = \begin{pmatrix}x_{t2}\\y_{t2}\end{pmatrix}$$

Entonces:

1. Ubicar los dos puntos de la función esperada en un gráfico.

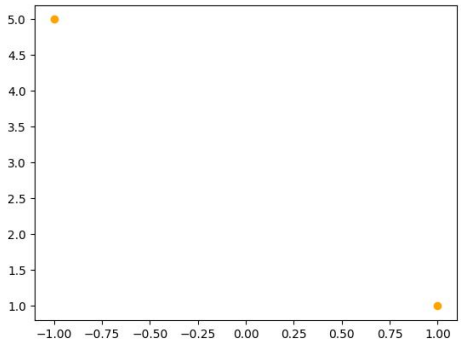


Figura 3: Ubicación de los puntos reales de la función

2. Calcular $minX$, $minY$, $maxX$, $maxY$ con la siguiente ecuación:

$$\begin{aligned} minX &= min\{x_{t1}, x_{t2}\} & maxX &= max\{x_{t1}, x_{t2}\} \\ minY &= min\{y_{t1}, y_{t2}\} & maxY &= max\{y_{t1}, y_{t2}\} \end{aligned}$$

3. Con los valores obtenidos calcular Δx y Δy.

$$\Delta x = \frac{maxX - minX}{|x_2 - x_1|} \qquad y \qquad \Delta y = \frac{maxY - minY}{|y_2 - y_1|}$$

   De esta forma se encuentra la equivalencia para transformar los puntos obtenidos de openCV a los puntos esperados.

4. Ahora es necesario transformar cada uno de los puntos obtenidos originalmente utilizando la siguiente fórmula:

$$x_{tn} = minX + \Delta x(x_n - min\{x_1, x_2\}) \qquad y \qquad y_{tn} = minY + \Delta y(y_n - min\{y_1, y_2\})$$

5. Una vez transformados todos los puntos, se procede hacer una gráfica para verificar el resultado.

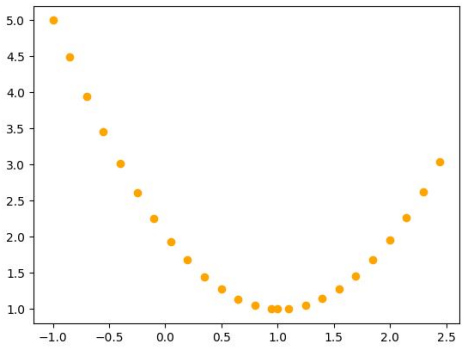


Figura 4: Ubicación de los puntos reales de la función

Una vez transformados los puntos, es posible encontrar una función más cercana a la función real.

II-g Error experimental.

Un error experimental es una desviación del valor medido de una magnitud física respecto al valor real de dicha magnitud. En general los errores experimentales son ineludibles y dependen básicamente del procedimiento elegido y la tecnología disponible para realizar la medición. Existen dos maneras de cuantificar el error de la medida: el error absoluto y el error relativo.[3]

Error absoluto. Es la diferencia entre el valor real f_r, y el valor medido f_m, este se expresa como:

$$e_{abs} = |f_r - f_m| \tag{12}$$

Error relativo. Es el cociente entre el error absoluto y el valor real, este se expresa como:

$$e_{rel} = \frac{e_{abs}}{|f_r|} = |\frac{f_r - f_m}{f_r}| \tag{13}$$

III. Resultados

A continuación se presentan los resultados obtenidos del proyecto, la cual es una aplicación funcional para interpolar funciones desde una fotografía, además de distintos ejemplos de uso para mostrar sus distintas funcionalidades.

III-a Desarrollo de la aplicación

La aplicación fue desarrollada en Python haciendo uso del entorno de desarrollo integrado Visual Studio Code. Para el desarrollo se definieron cinco etapas: **obtención de puntos a partir de una imagen, definición de coordenadas, definición de puntos a interpolar, interpolación y salida de datos**. Cada una de las etapas fue de vital importancia para el funcionamiento correcto de la aplicación, el objetivo principal de cada etapa fue:

Obtención de puntos: En esta etapa el enfoque principal fue identificar los puntos de una fotografía para transformarlos en puntos a interpolar, para esto se utilizó le librería OpenCV. Es necesario que el gráfico a interpolar esté dibujado en tonalidades azules y evitar una fotografía con ruido o colores que puedan afectar al resultado.

Definición de coordenadas. En esta etapa se usaron los puntos encontrados en la etapa anterior y se transformaron a coordenadas más adecuadas para su interpolación, estas coordenadas pueden ser arbitrarias o se pueden obtener respecto a dos puntos dados por el usuario.

Definición puntos a interpolar. Debido a que una gran cantidad de puntos dificulta su interpolación, se desarrolló una opción en la cual se puede decidir la cantidad de puntos a interpolar, lo cual mejora los resultados en distintas interpolaciones.

Interpolación. En esta etapa Se crearon tres funciones que reciben un vector con los puntos a interpolar, a conveniencia este vector recibió el nombre *Puntos* para las tres funciones definidas, posteriormente estas retornan el(los) polinomio(s) que definen la(s) función(es) que interpolan los puntos; Estas funciones son:

polinomial *polinomial* es una función ubicada dentro del fichero *interpolinomial*, esta función se encarga de retornar el polinomio resultante de interpolar por el método de interpolación polinomial.
Lo primero que hace esta función es crear dos matrices: coeficientes y solución, estas dos juntas describen la matriz ampliada descrita en la eq(4).

```
def polinomial(puntos):
    longitud = len(puntos)
    coeficientes = []
    solucion = []
```

Figura 5: Creación de matrices coeficientes y solución.

Posteriormente por medio de iteraciones se rellena la matriz de coeficientes respecto a lo descrito en el vector *Puntos*.

```
for i in range(longitud):
    coeficientes.append([])
    for j in range(longitud):
        coeficientes[i].append(puntos[i][0] ** ((longitud - j) - 1))
```

Figura 6: ciclo que rellena la matriz coeficientes

Luego de esto por medio de otro ciclo se rellena la matriz solución.

```
for i in range(longitud):
    solucion.append([])
    solucion[i].append(puntos[i][1])
```

Figura 7: ciclo que rellena la matriz solución

Por último con ayuda de la librería *numPy* se resuelve el sistema de ecuaciones y la solución es almacenada en el vector rarray para que una función *vtp* transforme el vector en un polinomio de la clase *symPy* y este sea retornado como salida de la función.

```
r = linalg.solve(coeficientes, solucion)
rarray = []

for i in range(len(r)):
    rarray.append(round(r[i][0], 5))
return(VTP.vtp(rarray))
```

Figura 8: Resolución de la matriz y retorno del polinomio interpolador

lagrange *lagrange* es una función ubicada dentro del fichero *interLagrange*, esta función se encarga de retornar el polinomio resultante de interpolar por el método de interpolación de Lagrange.
Teniendo en cuenta que el objetivo es recrear la fórmula descrita por la eq(5) y esta presenta un sumatorio se creó un ciclo que itera sobre los valores de i, pero también es sabido que dentro de este sumatorio se encuentra el productorio descrito en la eq(6), este productorio está descrito por un ciclo que itera sobre los valores de j. Este par de ciclos son.

```
for i in range(len(puntos)):#Sumatoria
    num = puntos[i][1]
    den = 1
    car = ""
    cont = 0
    for j in range(len(puntos)):#productorio
        if j != i:
            den = den * (puntos[i][0] - puntos[j][0])
            if cont == 0:
                car = car + "(" + "x - " + str(puntos[j][0]) + ")"
            else:
                car = car + "*" + "(" + "x - " + str(puntos[j][0]) + ")"
            cont = cont + 1
    coef = str(num) + "/" + str(den)

    if i == 0:
        par = par + coef + "*" + car
    else:
        par = par + " + " + coef + "*" + car
```

Figura 9: Ciclos for para productorio y sumatorio

Estos ciclos operan sobre diferentes variables de las cuales destaca *par*, que es una cadena de texto que almacena el polinomio, esta cadena luego es transformada en un polinomio de la clase *symPy* que es almacenado en *polinomiont*, este polinomio es desarrollado con ayuda de funciones de la misma clase y luego es retornado como valor de salida de la función.

```
polinomiont = parse_expr(par)
polinomiot = expand(simplify(polinomiont))
return polinomiot
```

Figura 10: Salida de la función lagrange

splines *splines* es una función ubicada dentro del fichero *intersplines*, esta función se encarga de retornar la función resultante de interpolar por el método de interpolación por Splines cúbicos.
esta función se divide en cuatro partes, la primera extrae las ecuaciones observadas eq(8).

```
for i in range(2 * (longitud - 1)):
    cf = []
    if i % 2 == 0:
        numero = puntos[int(i/2)][0]
    else:
        numero = puntos[int(i/2) + 1][0]

    for j in range(int(i/2) * 4):
        cf.append(0)

    cf.append(numero ** 3)
    cf.append(numero ** 2)
    cf.append(numero)
    cf.append(1)
    for j in range(((longitud - 1) * 4) - int(i/2) * 4 - 4):
        cf.append(0)
    coeficientes.append(cf)
```

Figura 11: Parte 1 interpolación por splines cúbicos.

La segunda parte extrae las ecuaciones referentes a las primeras derivadas descritas en eq(9).

```
for i in range(longitud - 2):
    cfd = []
    punto = puntos[i + 1][0]
    for j in range(i * 4):
        cfd.append(0)
    cfd.append(3 * (punto ** 2))
    cfd.append(2 * punto)
    cfd.append(1)
    cfd.append(0)
    cfd.append(-3 * (punto ** 2))
    cfd.append(-2 * (punto))
    cfd.append(-1)
    cfd.append(0)
    for j in range(4 * (longitud - 1) - (8 + i * 4)):
        cfd.append(0)
    coeficientes.append(cfd)
```

Figura 12: Parte 2 interpolación por splines cúbicos.

La tercera parte extrae las ecuaciones referentes a las segundas derivadas descritas en eq(9).

```
for i in range(longitud - 2):
    cfdd = []
    punto = puntos[i + 1][0]
    for j in range(i * 4):
        cfdd.append(0)
    cfdd.append(6 * punto)
    cfdd.append(2)
    cfdd.append(0)
    cfdd.append(0)
    cfdd.append(-6 * punto)
    cfdd.append(-2)
    cfdd.append(0)
    cfdd.append(0)
    for j in range(4 * (longitud - 1) - (8 + i * 4)):
        cfdd.append(0)
    coeficientes.append(cfdd)
```

Figura 13: Parte 3 interpolación por splines cúbicos.

La cuarta parte extrae las ecuaciones descritas en eq(10).

```
ext0 = [6 * (puntos[0][0]), 2, 0, 0]
for i in range(4 * (longitud - 1) - 4):
    ext0.append(0)
coeficientes.append(ext0)
ext1 = []
for i in range(4 * (longitud - 1) - 4):
    ext1.append(0)
ext1.append(6 * puntos[longitud - 1][0])
ext1.append(2)
ext1.append(0)
ext1.append(0)
coeficientes.append(ext1)

for i in range(1, longitud - 1):
    solucion.append([puntos[i][1]])
    solucion.append([puntos[i][1]])
solucion.append([puntos[longitud - 1][1]])
for i in range(longitud - 2):
    solucion.append([0])
    solucion.append([0])

solucion.append([0])
solucion.append([0])
```

Figura 14: Parte 4 interpolación por splines cúbicos.

Por último se resuelve el sistema de ecuaciones y se retorna como valor de salida.

```
r = linalg.solve(coeficientes, solucion)
rarray = []
for i in range(len(r)):
    rarray.append(round(r[i][0], 5))

ecuaciones = []

for i in range(longitud - 1):
    y=np.linspace(puntos[i][0], puntos[i + 1][0])
    ecuaciones.append(VTP.vtp([rarray[i * 4], rarray[i * 4 + 1], rarray[i * 4 + 2], rarray[i * 4 + 3]]))

return ecuaciones
```

Figura 15: Salida de la función splines.

Salida de datos. Una vez calculados los polinomios de cada tipo de interpolación, se procede a realizar la respectiva gráfica, de modo que se puedan corroborar los resultados y analizar el comportamiento de la función obtenida. Además, existe la posibilidad de retornar las funciones de modo que sean compatibles con la calculadora gráfica Geogebra, de esta forma analizar más a fondo los resultados.

III-b Uso de la aplicación

La interfaz gráfica de la aplicación es la siguiente:

Figura 16: Interfaz gráfica de la aplicación desarrollada

Al hacer clic en el botón de cargar la imagen se brinda la opción de cargar archivos desde el computador:

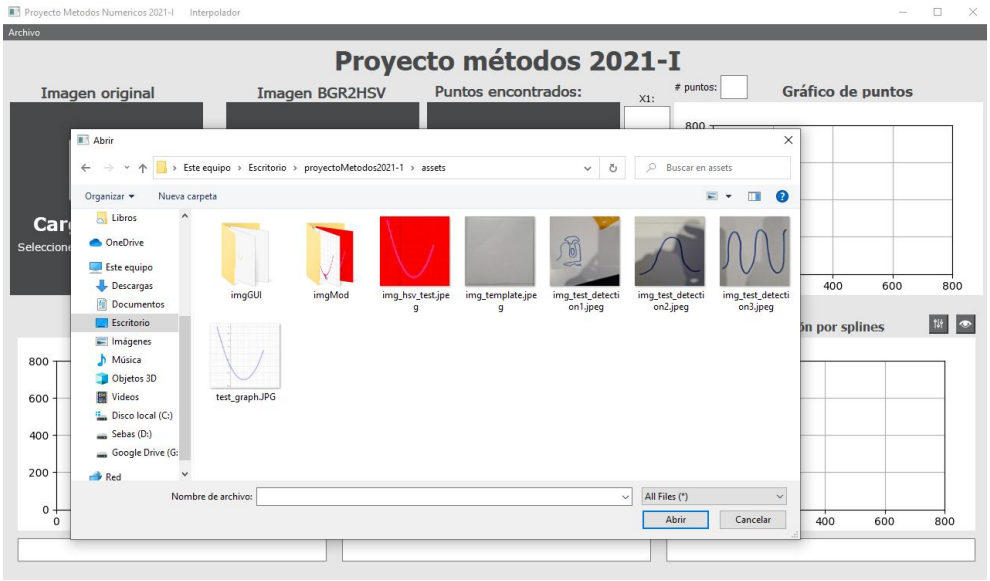


Figura 17: Interfaz gráfica cargar imagen

Una vez cargado el archivo, la aplicación calcula los distintos puntos a interpolar de la siguiente manera:

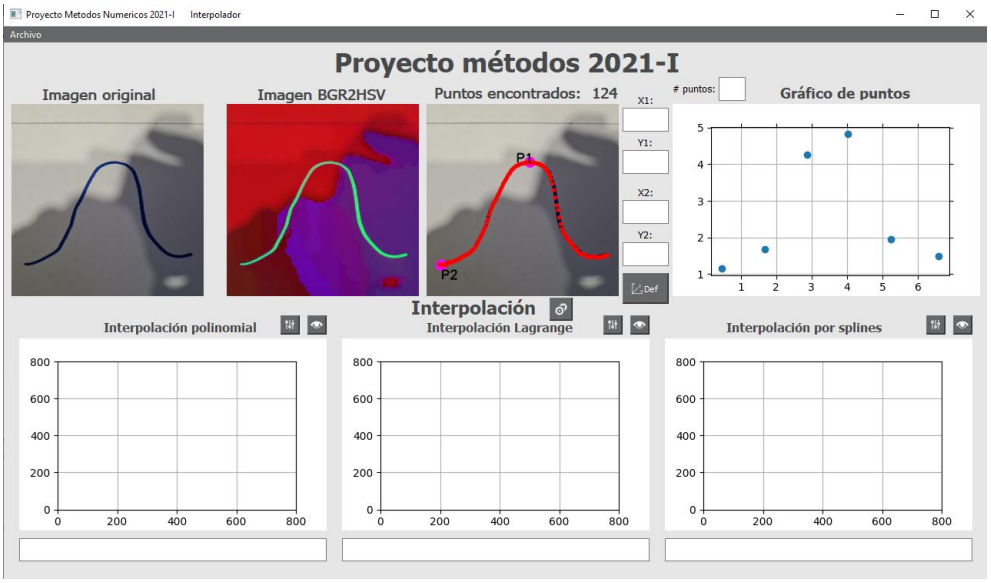


Figura 18: Interfaz gráfica imagen cargada

En caso de que el usuario tenga el máximo y mínimo absoluto de la gráfica, será posible ingresar el valor para mejorar los resultados. Además se brinda la opción de seleccionar el número de puntos a interpolar, si el usuario no selecciona un número, automáticamente se interpolaran 6 puntos. En caso de que se ingresen valores, el gráfico se ajusta:

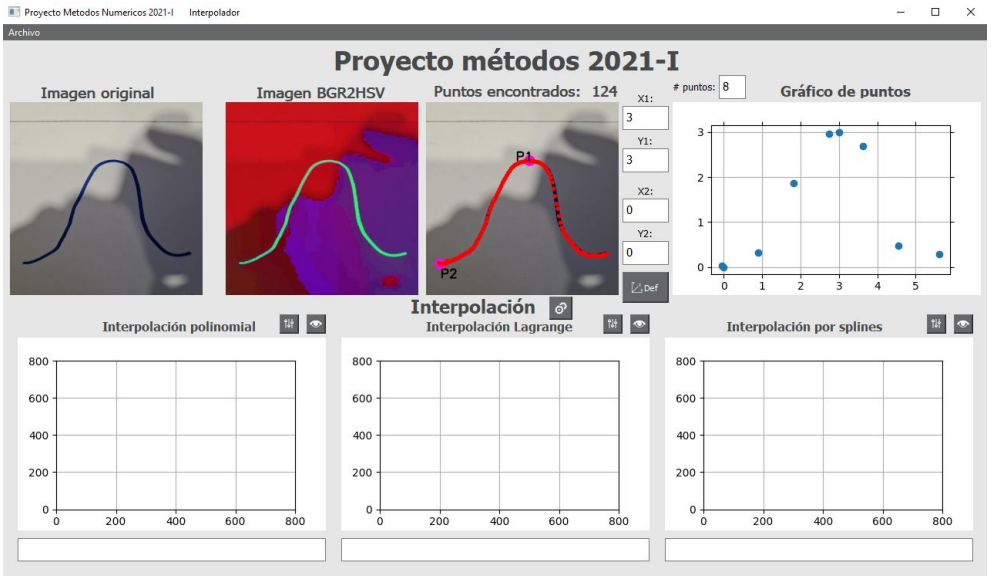


Figura 19: Interfaz gráfica puntos ajustados

A continuación es necesario presionar el botón de interpolar para iniciar los distintos algoritmos de interpolación, actualizando los tres gráficos inferiores:

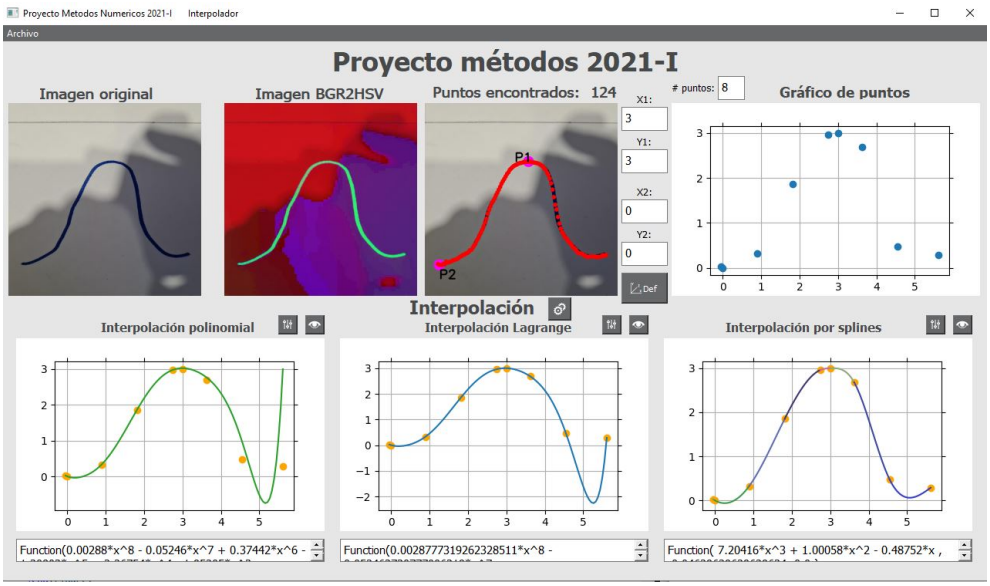


Figura 20: Interfaz gráfica gráficos de interpolación

Por último se brinda la opción de obtener un documento .txt con las funciones obtenidas al interpolar, para esto es necesario hacer clic en archivo / exportar funciones. El cual genera el siguiente archivo en la carpeta de la aplicación.

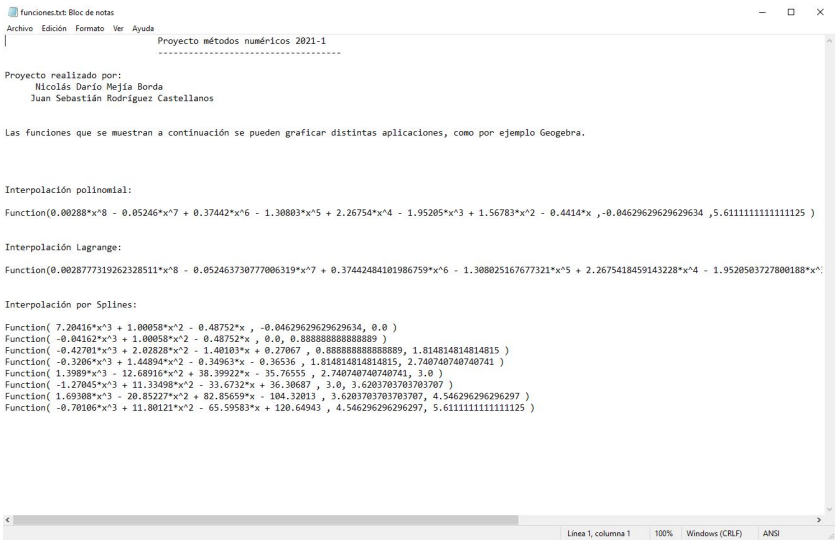


Figura 21: Resultados de funciones interpoladas

La aplicación se encuentra en un repositorio en GitHub y es posible descargarla y usarla. El link del repositorio es el siguiente: `https://github.com/juarodriguezc/proyectoMetodos2021-1.git` .

III-c Ejemplo con un gráfico generado por computador

Para mostrar los resultados obtenidos con la aplicación, se van a realizar distintas pruebas, la prueba actual consiste en utilizar un gráfico generado por Geogebra e interpolar usando la aplicación. Luego se

comparan los resultados obtenidos. La función generada fue:

$$f(x) = (x-1)^2 + 1$$

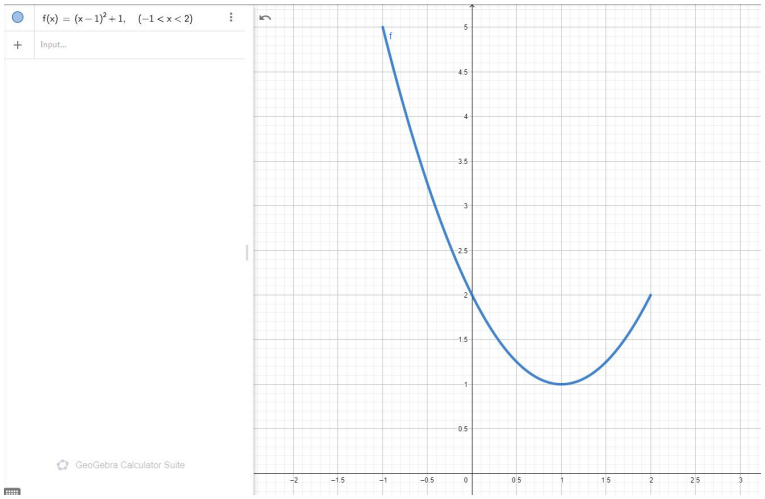


Figura 22: Gráfico original a interpolar

Se puede apreciar que es una función cuadrática, con dominio entre [-1, 2]. Por lo que ahora es necesario cargar la imagen a la aplicación. Una vez cargada la imagen el resultado que se muestra es el siguiente:

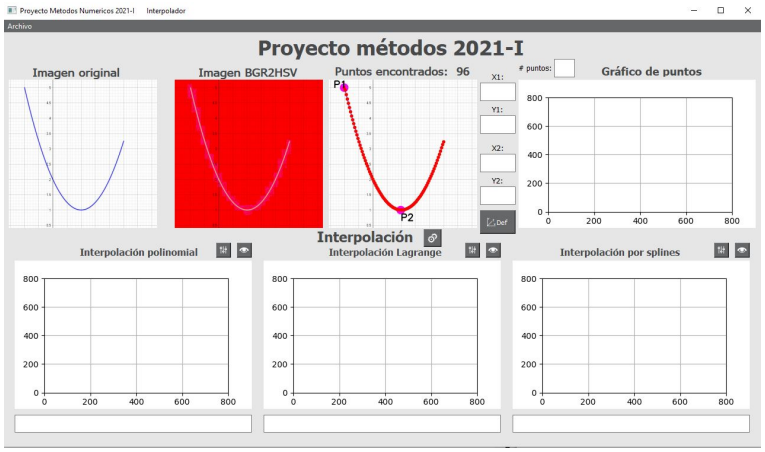


Figura 23: Aplicación con el gráfico cargado

Ahora es necesario definir el máximo y mínimo absoluto de la función, como se restringió el dominio del gráfico es una tarea sencilla, con $(x_1, y_1) = (-1, 5)$ y $(x_2, y_2) = (1, 1)$. Ahora existen dos alternativas, definir una cantidad baja de puntos a interpolar o definir una cantidad alta de puntos a interpolar.

- **Baja cantidad de puntos n = 6:**
  Al realizar la interpolación con una baja cantidad de puntos, los resultados fueron los siguientes:

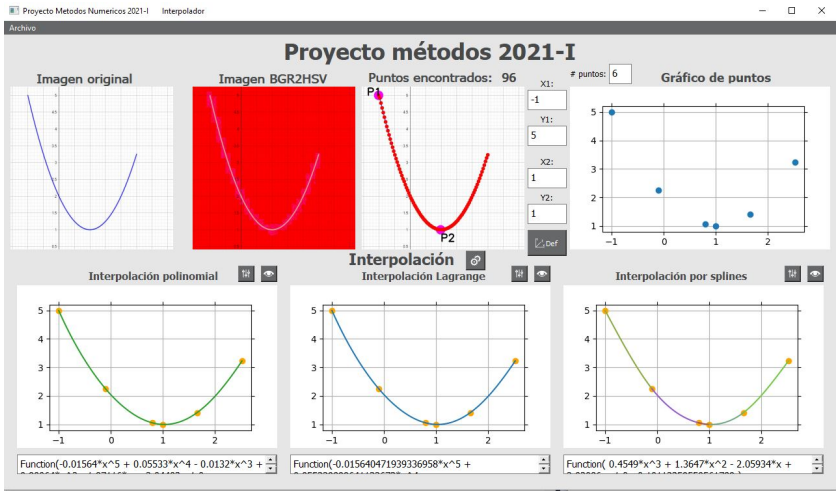


Figura 24: Interpolación con 6 puntos obtenidos.

Se puede apreciar como los tres algoritmos de interpolación obtuvieron resultados similares, logrando gráficos muy parecidos al de la función original.

- **Alta cantidad de puntos n = 18**
  Al realizar la interpolación con una alta cantidad de puntos, los resultados fueron los siguientes:

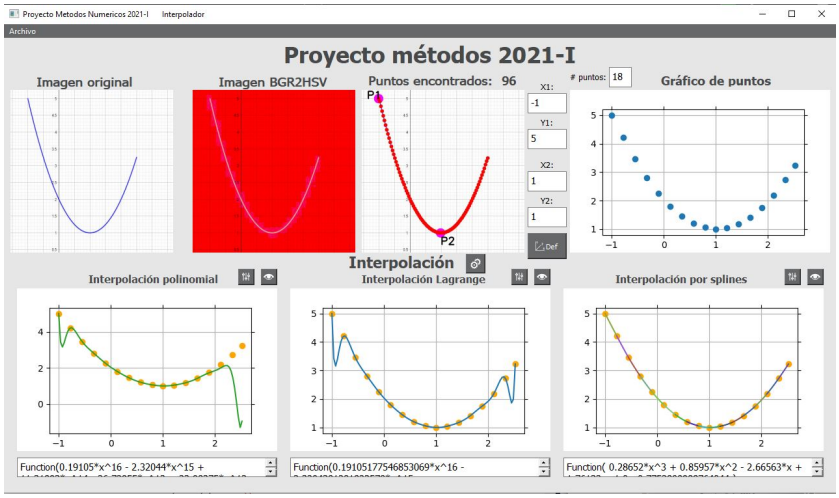


Figura 25: Interpolación con 18 puntos obtenidos.

Se puede apreciar como los resultados varían bastante a los obtenidos anteriormente, en este caso el gráfico de interpolación polinomial se aleja después de cierto punto. El gráfico de la función obtenida por interpolación de Lagrange cubre todos los puntos, pero empieza a presentar fluctuaciones. Mientras que la interpolación por splines sigue manteniendo su forma parabólica.

III-d Ejemplo utilizando un gráfico a mano

Además de la capacidad de interpolar gráficos generados por computador, la aplicación puede identificar los gráficos dibujados a mano e interpolar de la mejor manera posible. Por esta razón, a continuación se van a mostrar los resultados obtenidos al interpolar un gráfico hecho a mano.
La imagen a interpolar es la siguiente:

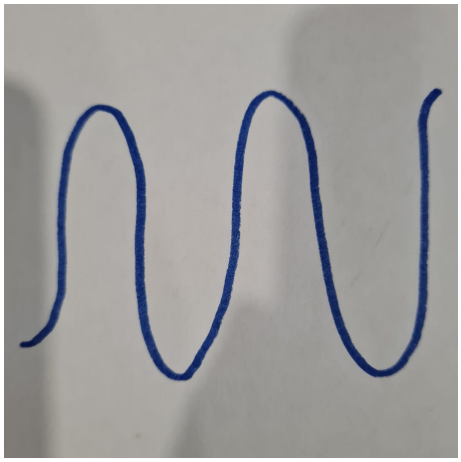


Figura 26: Gráfico hecho a mano a interpolar.

Por lo que al cargar la imagen en la aplicación se obtiene el siguiente resultado:

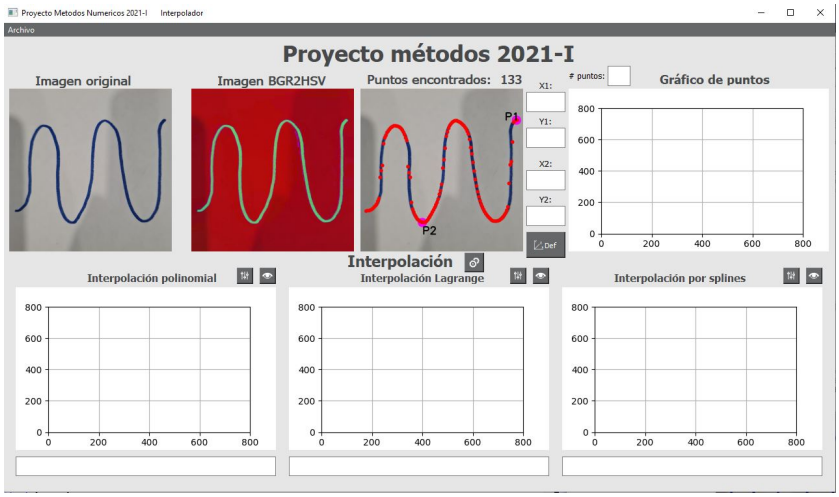


Figura 27: Gráfico hecho a mano cargado en la aplicación.

De manera similar al problema anterior, se da la opción de ajustar los puntos dependiendo del máximo y mínimo absoluto del gráfico, sin embargo, como se trata de una función hecha a mano, esos campos se van a dejar vacíos. Respecto al número de puntos a interpolar, se realizará de manera similar al ejemplo anterior:

- **Baja cantidad de puntos n = 6:**
  Al realizar la interpolación con una baja cantidad de puntos, los resultados fueron los siguientes:

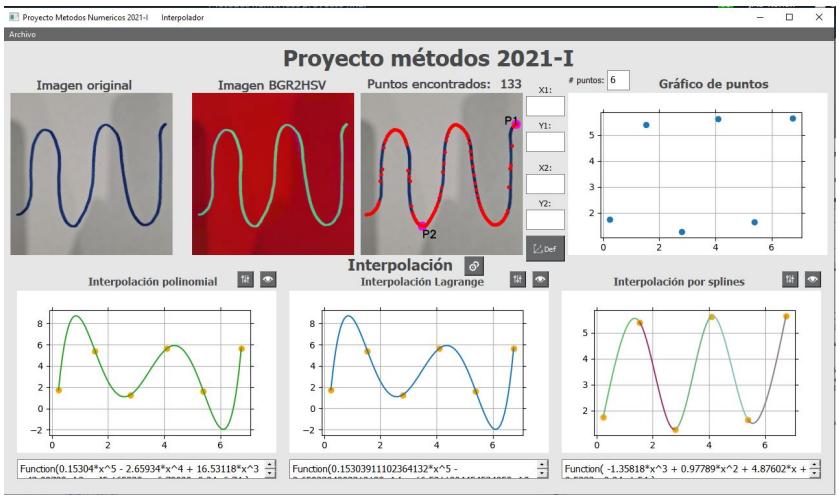


Figura 28: Interpolación con 6 puntos obtenidos.

Se puede apreciar como los algoritmos de interpolación polinomial y Lagrange obtuvieron resultados similares, mientras que los resultados obtenidos por splines son bastante diferentes.

- **Alta cantidad de puntos n = 18**
  Al realizar la interpolación con una cantidad de puntos, los resultados fueron los siguientes:

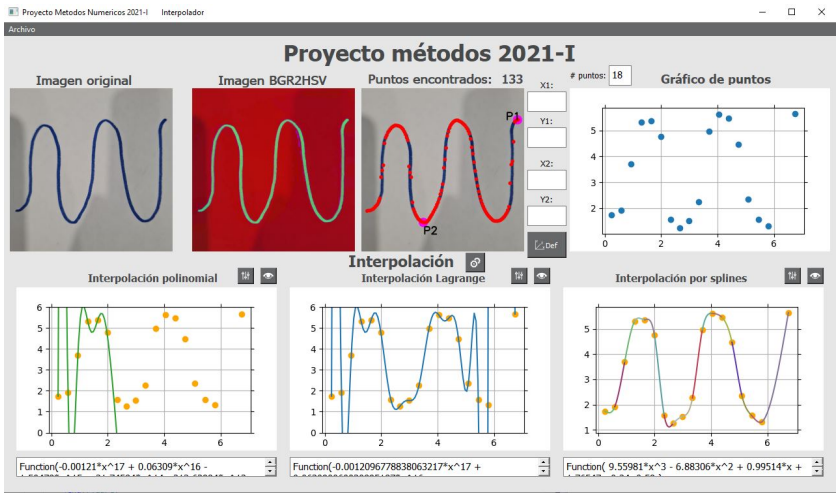


Figura 29: Interpolación con 18 puntos obtenidos.

Los resultados ahora son muy diferentes, se puede apreciar como la interpolación polinomial deja de tocar los puntos cuando $x > 2$, un comportamiento similar pareciera que ocurre en la interpolación de Lagrange, sin embargo, el polinomio obtenido pasa por todos los puntos establecidos, aunque fluctúa bastante. Ahora, respecto a la interpolación por splines, los resultados son bastante cercanos al dibujo realizado, pasa por todos los puntos interpolados y no fluctúa como los anteriores algoritmos.

IV. Análisis de resultados

De los resultados obtenidos al interpolar una función generada por computador se obtuvieron distintas funciones que requieren de un análisis más profundo. En el anexo 1 se incluyen distintas gráficas comparativas con los valores de las funciones y sus respectivos errores. Esta parte del documento se divide en tres categorías

IV-a Análisis de los resultados obtenidos con 6 puntos

Para empezar con el análisis de los resultados se utiliza la funcionalidad de la aplicación que retorna las respectivas funciones al interpolar con los puntos. Cuando se interpola con 6 puntos las funciones obtenidas son:

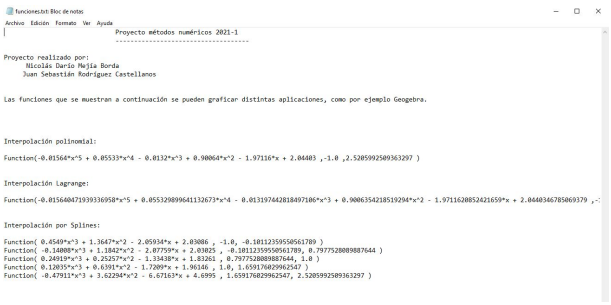


Figura 30: Documento con todas las funciones al interpolar

Una vez obtenidas las funciones, se utiliza la herramienta Excel para obtener más información de los resultados obtenidos. A continuación se muestra un gráfico comparativo de los errores obtenidos al interpolar **polinomial - gris oscuro, Lagrange - naranja, Splines - morado**.

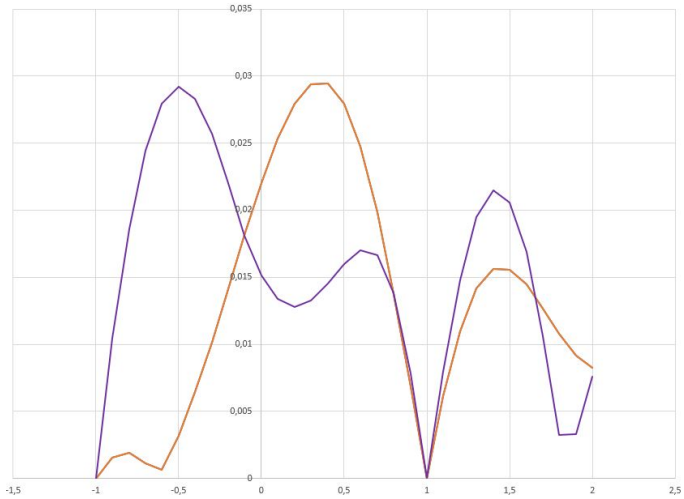


Figura 31: Comparación errores relativos interpolación

A simple vista no es posible ver el error obtenido por interpolación polinomial, esto se debe a que los valores obtenidos de error son muy similares a los obtenidos por interpolación de Lagrange. Además, se puede evidenciar que con 6 puntos, los tres algoritmos de interpolación poseen errores muy similares.

IV-b Análisis de los resultados obtenidos con 18 puntos

Una vez obtenidos los resultados con 6 puntos, se realizará un análisis similar con 18 puntos. De manera similar al análisis anterior, se utiliza la funcionalidad de la aplicación para obtener las distintas funciones, las cuales son:

Figura 32: Documento con todas las funciones al interpolar

Obtenidas las funciones al interpolar, se utiliza Excel para analizar los resultados. A continuación se muestra un gráfico comparativo con los errores obtenidos al interpolar **polinomial - gris oscuro, Lagrange - naranja, Splines - morado**.

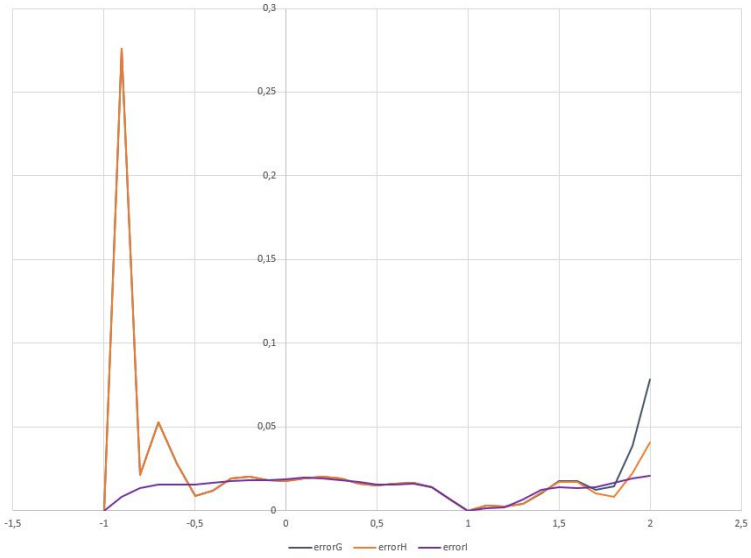


Figura 33: Comparación errores relativos interpolación

En este gráfico se puede evidenciar bastantes diferencias entre los errores relativos de los diferentes algoritmos de interpolación. Tanto el algoritmo de interpolación polinomial, como el de Lagrange tienden a tener un error bastante alto entre $-1 \leq x \leq -0{,}5$. Sin embargo, este error luego se mantiene bajo en el resto de la interpolación. Por otro lado, la interpolación por splines mantiene un error bastante bajo en todo momento.

IV-c Comparativa de los resultados entre 6 y 18 puntos

Para terminar con el análisis de los resultados, se va a hacer una comparativa de cada algoritmo cuando se varía la cantidad de puntos a interpolar. Para esto se utilizan los datos mostrados en Excel, por lo que se obtiene:

Interpolación polinomial:
Al comparar los errores de la interpolación polinomial cuando se interpolan 6 y 18 puntos, se obtiene el siguiente gráfico: **(azul : 6 puntos , naranja : 18 puntos)**

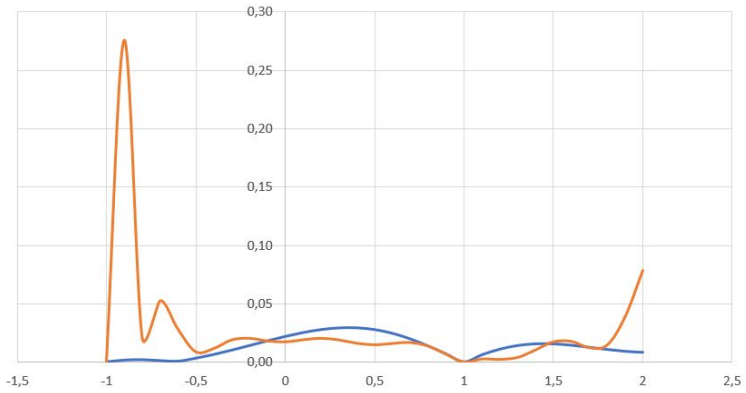


Figura 34: Comparación errores relativos interpolación polinomial

Donde se puede evidenciar que el error aumenta en gran medida cuando la cantidad de puntos aumenta, el error en naranja muestra más picos y errores mayores, mientras que el error en azul se mantiene más estable.

Interpolación de Lagrange:
De manera similar, cuando se interpolan 6 y 18 puntos usando el algoritmo de Lagrange, se obtiene el siguiente gráfico: **(azul : 6 puntos , naranja : 18 puntos)**

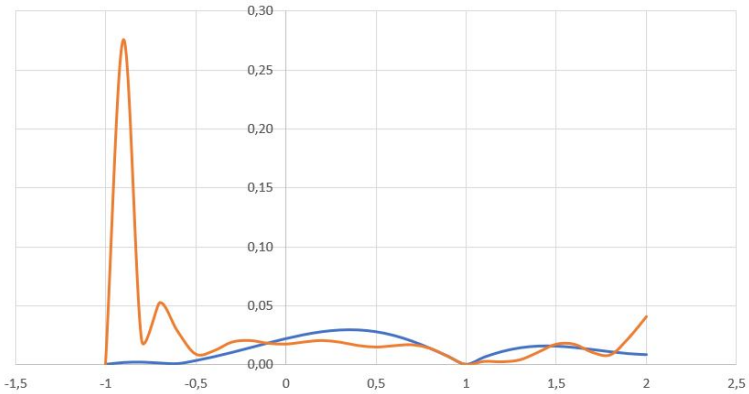


Figura 35: Comparación errores relativos interpolación de Lagrange

Donde se puede evidenciar que el error aumenta cuando la cantidad de puntos aumenta, es un comportamiento bastante similar que el obtenido con interpolación polinomial.

Interpolación por splines:
Al comparar los errores de la interpolación por splines cuando se interpolan 6 y 18 puntos, se obtiene el siguiente gráfico: **(azul : 6 puntos , naranja : 18 puntos)**

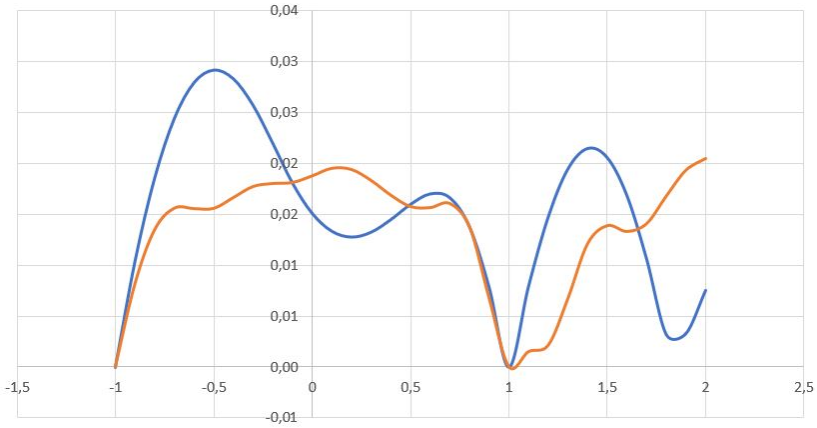


Figura 36: Comparación error relativo interpolación por Splines

Donde se puede evidenciar que al contrario de los dos algoritmos anteriores, el algoritmo de interpolación por Splines tiene un error ligeramente menor cuando la cantidad de puntos a interpolar aumenta.

V. Conclusiones

- Las interpolaciones polinomial y de Lagrange llegan a ser buenas generando funciones que pasen por los puntos deseados, pero para buscar puntos intermedios no son tan buenos cuando se trata de una cantidad de puntos muy grande.
- La interpolación por splines cúbicos es buena para acercarse a curvas suaves, y para hallar valores intermedios, aunque con gran cantidad de puntos genera muchas funciones con las cuales el trato puede llegar a ser muy engorroso.
- Al momento de tratar polinomios con un software es preferible utilizar una gran cantidad de polinomios de bajo grado que pocos de muy alto grado ya que los errores generados por almacenamiento y tratado de datos llega a ser muy grande.
- El error obtenido dentro del software puede deberse en gran parte al error que se genera cuando se obtienen los puntos por medio de openCV ya que hay un cierto desfase al utilizar esta librería.
- Al momento de interpolar con métodos como interpolación polinomial o interpolación de Lagrange es preferible usar una menor cantidad de puntos para obtener curvas con mejor resultado.

VI. Bibliografía

[1] S. Chapra and R. Canale, Métodos numéricos para ingenieros (5a. ed.), 5th ed. Distrito Federal: McGraw-Hill Interamericana, 2007, p. 503.

[2] García Quesada, Tutorial de Analisis Numérico Interpolación : Splines cúbicos, 1st ed. Gran Canaria: ULPGC, 2000.

[3] .ᴱʳror experimental - Wikipedia, la enciclopedia libre", Es.wikipedia.org, 2021. [Online]. Available: https://es.wikipedia.org/wiki/Error_experimental. [Accessed: 27- Jul- 2021].