

Informe del taller de interpolación de la mano

Johan Daniel Ortégón Parra, Ricardo Riscanevo Cotrina

Abstract

Un polinomio es una expresión algebraica constituida por un número n de variables y constantes, con operaciones aritméticas y valores en los exponentes en la variable. Un polinomio da origen a una función polinómica al establecer una relación en la que cada valor de x tenga asignado un único valor como resultado, por lo cual, dependiendo del grado del polinomio, algunas funciones polinómicas reciben un respectivo nombre.

Una función lineal es una función polinómica de 1 grado que puede ser representada mediante 2 puntos como una línea recta con una pendiente. La función cuadrática es una función polinómica de 2 grado con un valor máximo o mínimo, la cual a través de 3 puntos puede ser representada como una parábola, y una función cúbica es una función polinómica de 3 grado que puede ser representada con 4 puntos en un plano mediante curvas cúbicas.

Al conocer solamente los valores que toma una función, sin conocer cuál es exactamente la expresión de la función, es posible aplicar un método de interpolación entre ellos interpolación polinómica de Lagrange. Este método permite obtener de manera sencilla un aproximado a una función que toma n valores dados, generando un polinomio único con un grado $n-1$ puntos, permitiendo hallar valores desconocidos para la función dada por los valores. Este método puede generar funciones polinómicas desde 1 grado en adelante, siendo el valor resultante irregular al ser mayor el grado de polinomio. Teniendo en cuenta las ventajas y desventajas de Lagrange al manejar el grado del polinomio resultante, es conveniente con el menor grado posible al momento de emplear el método, siendo la función lineal, cuadrática y cúbica las mejores opciones para solucionar esta problemática.

Por consiguiente en el presente documento se expondrán tres intentos para tratar de solucionar “El reto de la mano”, para cada método se usaron criterios de selección de puntos diferentes, interpolaciones diferentes y por ende se obtuvieron resultados diferentes, para cada método se realizará su análisis correspondiente y su comparación con los otros métodos.

Al final de cada método estarán resueltas las preguntas correspondientes al reto, **Solo para el método que se seleccionó se respondió con todo detalle.**

1. Interpolación basada en pendientes

1.1. Metodología para la selección de puntos

Para esta interpolación los puntos se seleccionaron tomando en cuenta únicamente las **Pendientes entre puntos**, se trabajó bajo el supuesto de que la mano podía graficarse únicamente con **Rectas y Curvas**, resumiendo los polinomios a usar a **lineales y cuadráticos**.

Proceso de secciones lineales

1. Ir comparando pendientes, punto tras punto

2. Comparar el valor de las pendientes contra un índice de tolerancia
3. En caso de que la tolerancia no se exceda continuar suponiendo que estamos en una función lineal
4. En caso de que la tolerancia sea excedida de manera consecutiva por más de 2 puntos se terminará ese intervalo y se pasa a trabajar con las condiciones de un segmento cuadrático

Ejemplo de un comportamiento lineal

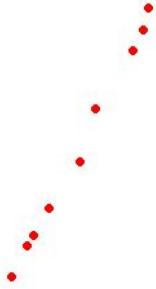


Figura 1: Comportamiento lineal

Fragmento del código encargado de la comparación

```
lineal <- function(d1,d2,pend,coord,x,y){

  if(d2 <=length(x))
  {
    if( x[d2] == x[d1]) { m = 0.0000001 }
    else
    {
      if( y[d2] == y[d1])
      {
        m = 0.000001
      }
      else
      {
        m = (y[d2]-y[d1])/(x[d2]-x[d1])
      }
    }
  }

  cat("resta",pend[length(pend)], "-",m,"=",abs(pend[length(pend)] - m),"\n")

  if( abs(pend[length(pend)] - m) < 0.5)
  {
    cat("1) d1:",d1," d2:",d2,"\n")
    lineal(d1,d2+1,pend,coord,x,y)
  }
  else
  {
    if( x[d2] == x[d2-1]) { m = 99999}
    else
    {
      if( y[d2] == y[d2-1])
      {
        m = 0.000001
      }
      else
      {
        m = (y[d2]-y[d2-1])/(x[d2]-x[d2-1])
      }
    }
    #m = (y[d2]-y[d2-1])/(x[d2]-x[d2-1])
    pend[length(pend)+1] = m
    cat("2) d1:",d2-1," d2:",d2,"\n")
    cat("m:",m,"\n")
    coord[length(coord)+1] = d2-1
    lineal(d2,d2+1,pend,coord,x,y)
  }
}
```

```

    }
}
else
{
    coord[length(coord)+1] = length(x)
    return(coord)
}
}

```

Donde la tolerancia de la pendiente se compara con el siguiente fragmento de código

```

if( abs(pend[length(pend)] - m) < 0.5) #Sinendo 0.5 la tolerancia
{
    cat("1) d1:",d1," d2:",d2,"\n")
    lineal(d1,d2+1,pend,coord,x,y)
}

```

Comportamiento con segmentos cuadráticos

1. Ir comparando pendientes, punto tras punto
2. verificar que las pendientes tengan un cambio continuo
3. Identificar fin de la curva
4. Tomar 3 puntos en la curva y expresar el polinomio de grado 2 correspondiente

Ejemplo de un comportamiento Cuadrático

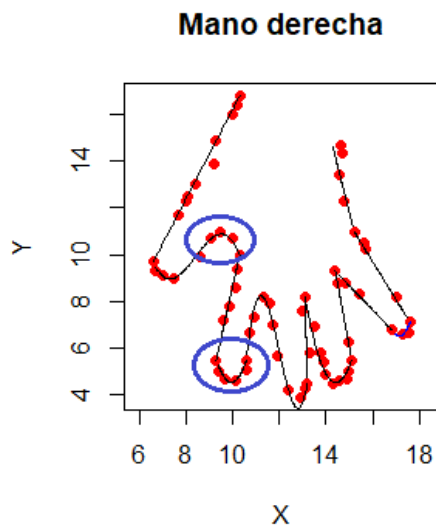


Figura 2: Comportamiento cuadrático

Fragmento del código encargado de la comparación

```
cuadratic <- function(d1,d2,pend,coorx)
{
  if(d2<=length(coord))
  {
    m = (y[coord[d2]]-y[coord[d2-1]])/(x[coord[d2]]-x[coord[d2-1]])
    m1 = pend[length(pend)]
    if( length(pend) < 0.5) { m0 = m1+1 }
    else { m0 = pend[length(pend)-1] }

    cat("m0: ",m0," m1: ",m1," m: ",m,"\n")
    if( m0 - m1 >0)
    {
      if( m1 > m)
      {
        pend[length(pend)+1] = m
        quadratic(d2,d2+1,pend,coorx)
      }
      else
      {
        pend[length(pend)+1] = m
        cat("1) d1:",coord[d2-1]," d2:",coord[d2],"\n")
        cat("m:",m,"\n")
        coorx[length(coorx)+1] = d2-1
        print(coorx)
        quadratic(d2,d2+1,pend,coorx)
      }
    }
  }
  else
  {
    if( m1 > m)
    {
      pend[length(pend)+1] = m
      quadratic(d2,d2+1,pend,coorx)
    }
    else
    {
      pend[length(pend)+1] = m
      cat("2) d1:",coord[d2-1]," d2:",coord[d2],"\n")
      cat("m:",m,"\n")
      coorx[length(coorx)+1] = d2-1
      print(coorx)
      quadratic(d2,d2+1,pend,coorx)
    }
  }
}
```

```

    }

}
else
{

    cat("return", coorx)
    return(coorx)
}
}

```

1.2. Algoritmo, requerimientos y codificación

Para que el algoritmo funcione correctamente se requiere - Principalmente es necesario que los **puntos estén en el orden en que se muestran en la grafica** - El algoritmo **No reconoce puntos con pendientes muy irregulares**, se tendrían que agregar muchas validaciones para que esté preparado en cada posible situación

1.2.1. Ejemplo de irregularidades

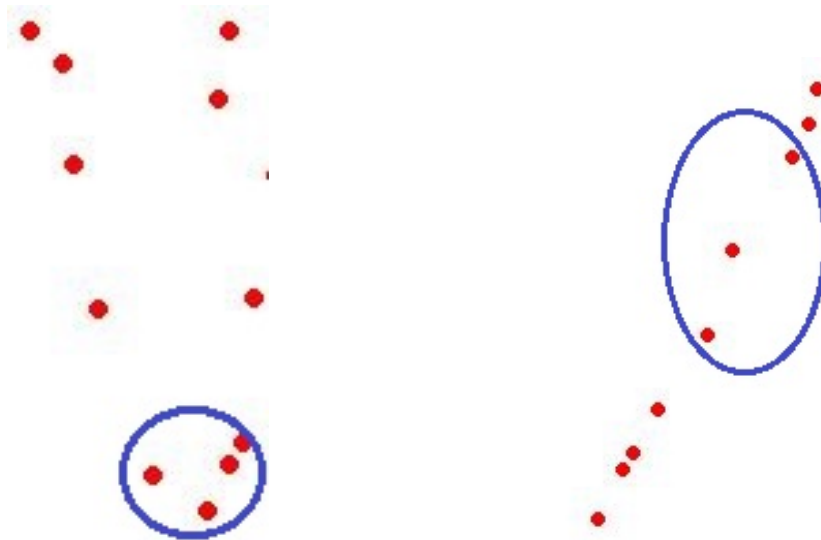
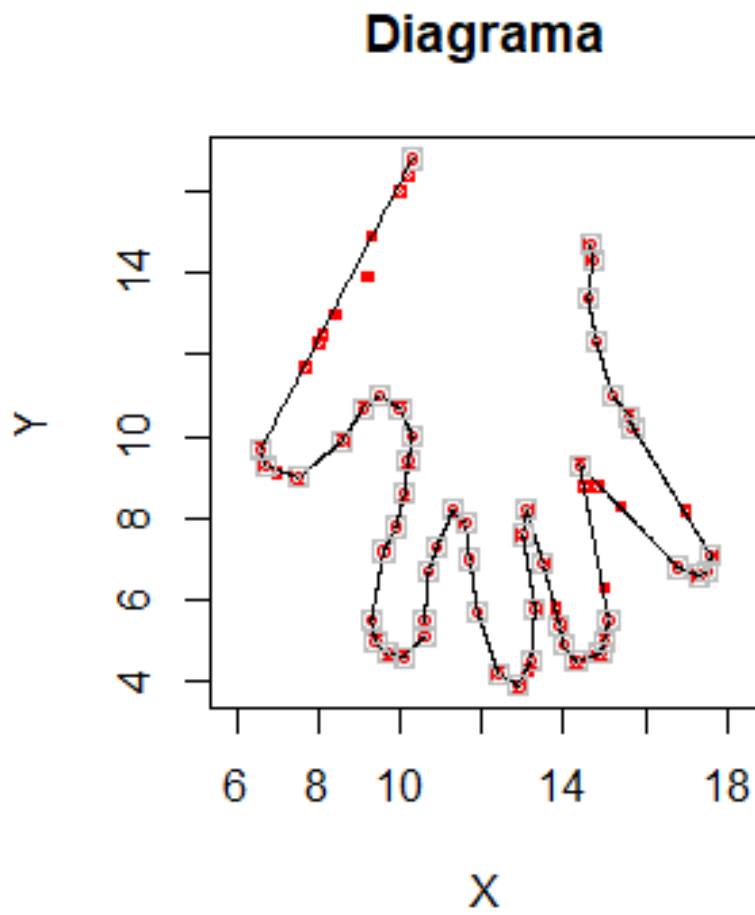


Figura 3: Irregularidades

1.3. Imágenes de resultado



1.4. Pruebas sobre el algoritmo y solución de preguntas

1.4.1. ¿El origen se puede modificar?

El origen puede modificarse, sin embargo, en algunos casos se pierde la calidad de la interpolación dependiendo el punto de origen, ya que la comparación entre pendientes cambia su orden y por ende el resultado, como se muestra a continuación

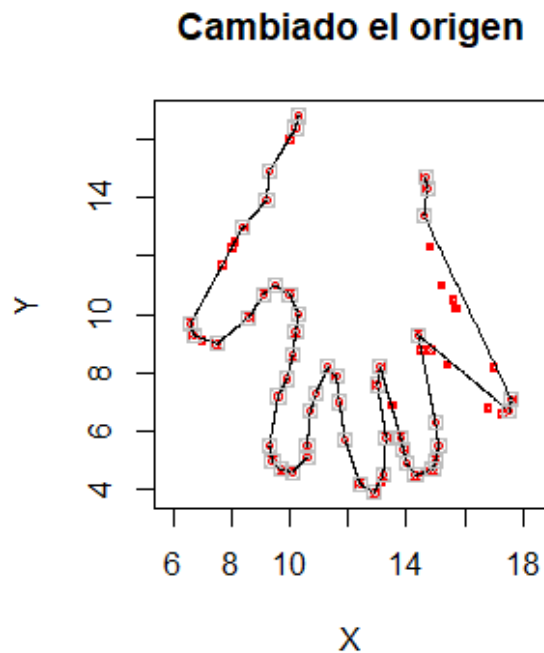


Figura 4: Mano con origen cambiado

1.4.2. ¿Si tenemos nueva información ósea nodos como podemos implementar esa información en el algoritmo de interpolación?

El algoritmo es capaz de manejar diversas cantidades de puntos, para ingresar o disminuir su cantidad lo único que se debe hacer es modificar los vectores $x2$ y $y2$ que aparecen en el código

1.4.3. ¿Su método es robusto, en el sentido que si se tienen más puntos la exactitud no disminuye?

Al haber nuevos puntos el algoritmo los interpola con los mismos criterios, por lo que la calidad no varía, siempre y cuando los puntos no se presenten de manera desordenada en comparación con su gráfico, **en la siguiente pagina**, un ejemplo producto de esta modificación.

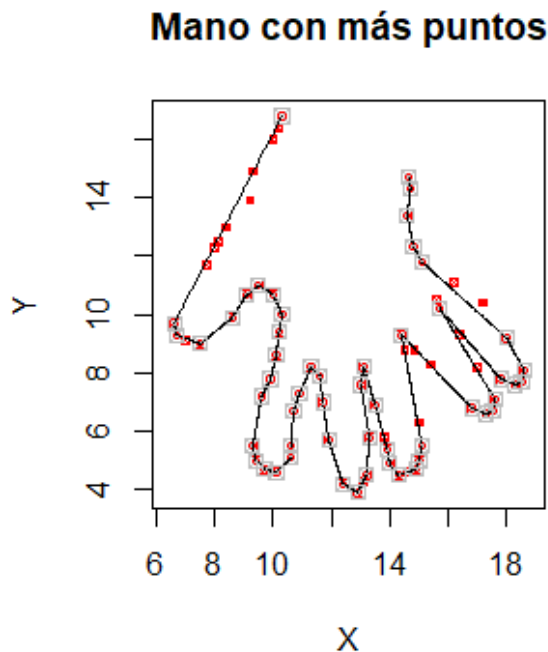


Figura 5: Mano con más puntos

1.5. Desventajas del Algoritmo

- Sensible a irregularidades
- Depende del orden de los puntos para funcionar correctamente
- La cantidad de puntos a tomar es mucho mayor en comparación a los demás algoritmos (50 puntos)

1.6. Ventajas del algoritmo

- Soporta la inserción de nuevos puntos
- Nunca supera el segundo grado en los polinomios que usa para sus interpolaciones

1.7. Notas de codificación y pruebas

El código completo de este algoritmo se encuentra en la misma carpeta del repositorio donde se extrajo este informe y tiene por nombre "ManoIntento1Pendientes", las pruebas que se hicieron con este algoritmo se encuentran en la carpeta de "Anexos" con sus respectivos códigos y el nombre de la prueba de cambio de orden tiene por nombre "ManoIntento1PendientesCambioOrigen" mientras que la prueba en la que se agregaron nuevos puntos tiene por nombre "ManoIntento1PendienteMasPuntos".

2. Interpolación por “Y” mínimos y máximos con cuadráticas

2.1. Metodología para la selección de puntos

Para este algoritmo los puntos se seleccionaron tomando en cuenta las variaciones del comportamiento de los puntos en el eje Y para la identificación de segmentos **Selección de “Puntos**

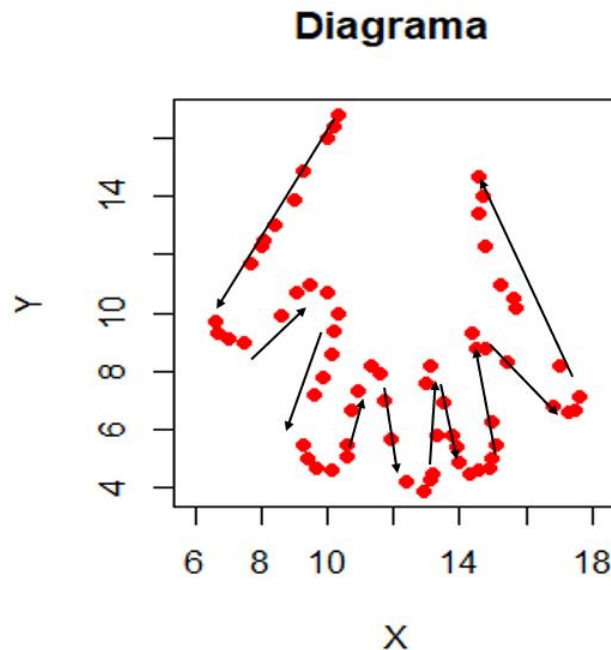


Figura 6: Comportamientos de Y

críticos de la mano”

Un punto critico de la mano sería el punto donde el comportamiento de la grafica cambia con respecto al eje Y

1. Se examina el comportamiento de la coordenada Y, punto por punto
2. Se van almacenando los puntos de un segmento hasta que el comportamiento de la Y cambie (De decreciente a creciente o inverso)
3. Terminada la validación se aplica interpolación a partir de 4 puntos para cada segmento encontrado

Fragmento del código encargado de la comparación

```
for(i in 1:fin_ciclo)
{
  resta = y2[i]-y2[i+1]
  multiplicacion = resta*resta_anterior

  if(multiplicacion < 0) # CAMBIO DE SENTIDO SI RESTA Y SU ANTERIOR TIENEN DIFERENTES SIGNOS
  {
    segmento_mano[cont_array_fragmento] = i
    cont_array_fragmento = cont_array_fragmento+1
```

```

contador = contador+1
cat(segmento_mano, "\n")

#EXTRACCION DE LOS DATOS QUE SE USARAN EN LA INTERPOLACION
punto_fin = length(segmento_mano)

datosx[1] = x2[segmento_mano[1]]#TOMAR EL PRIMER DATO
datosx[2] = x2[segmento_mano[2]]#TOMAR EL PRIMER DATO
datosx[3] = x2[segmento_mano[punto_fin]]#TOMAR EL ULTIMO DATO

datosy[1] = y2[segmento_mano[1]]#TOMAR EL PRIMER DATO
datosy[2] = y2[segmento_mano[2]]#TOMAR EL PRIMER DATO
datosy[3] = y2[segmento_mano[punto_fin]]#TOMAR EL ULTIMO DATO

cat("EL ultimo dato es: ",datosy[3])

if(length(segmento_mano) > 3)
{
  datosx[4] = x2[segmento_mano[punto_fin-1]]
  datosy[4] = y2[segmento_mano[punto_fin-1]]
  polinomioInterpolante = poly.calc(datosx,datosy)
  curve(polinomioInterpolante, add=T, from = datosx[1], to = datosx[3])
}
else
{
  polinomioInterpolante = poly.calc(datosx,datosy)
  curve(polinomioInterpolante, add=T, from = datosx[1], to = datosx[3])
}
segmento_mano = NULL
cont_array_fragmento = 1
}
else
{
  cat("otro:",y2[i], "\n")
  segmento_mano[cont_array_fragmento] = i
  cont_array_fragmento = cont_array_fragmento+1
}
resta_anterior = resta
}

```

2.2. Algoritmo, requerimientos y codificación

Para que este algoritmo funcione correctamente requiere:

- Principalmente es necesario que los puntos estén en el orden en que se muestran en la grafica.

2.3. Pruebas sobre el algoritmo y solución de preguntas

2.3.1. ¿El origen se puede modificar?

El origen se puede modificar, pero se pierde mucha calidad en las aproximaciones de la interpolación dada la sensibilidad que los polinomios cúbicos presentan frente a la selección de puntos, esto lo podemos ver a continuación.

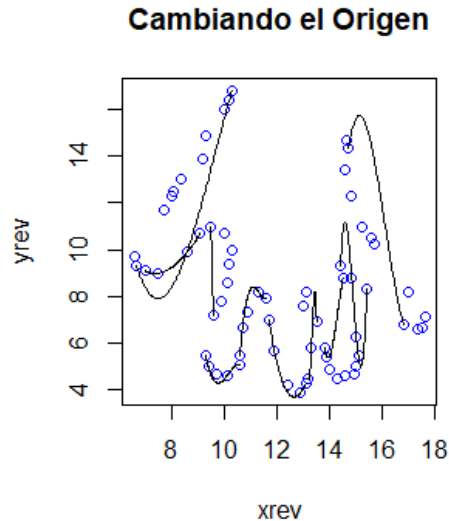


Figura 7: Cambiando origen

2.3.2. ¿Si tenemos nueva información ósea nodos como podemos implementar esa información en el algoritmo de interpolación?

El algoritmo es capaz de manejar diversas cantidades de puntos, para ingresar o disminuir su cantidad lo único que se debe hacer es modificar los vectores $x2$ y $y2$ que aparecen en el código.

2.3.3. ¿Su método es robusto, en el sentido que si se tienen más puntos la exactitud no disminuye?

El método no es robusto, puede reconocer nuevos puntos, pero su interpolación exitosa depende en gran parte de en qué parte del plano se ubican los nuevos puntos, como vemos a continuación, los puntos nuevos son ingresados, sin embargo, el algoritmo presenta dificultados para interpolar debido las coordenadas de los nuevos puntos en el eje y

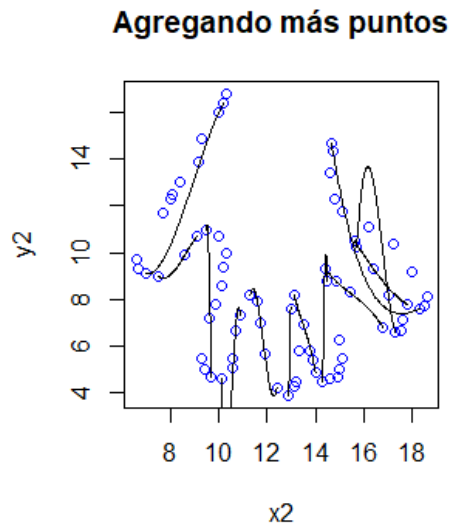


Figura 8: Agregando Puntos

2.4. *Desventajas del Algoritmo*

- Las cubicas son muy sensibles a irregularidades
- Depende del orden de los puntos para funcionar correctamente
- La cantidad de puntos de los que depende una cubica es poco eficiente (47 puntos)
- El ajuste es muy pobre en calidad

2.5. *Ventajas del algoritmo*

- Soporta la inserción de nuevos puntos
- Maneja una cantidad de puntos menor que el primer algoritmo
- Código menos complejo (el número de validaciones es mínima)

2.6. Imagen de resultado

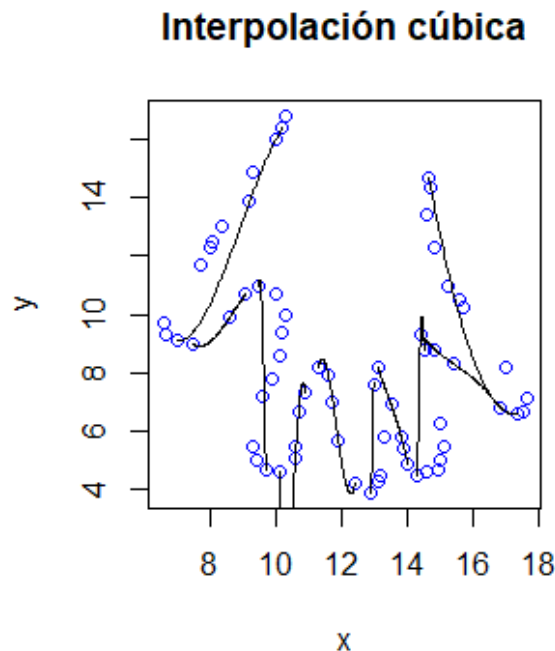


Figura 9: Interpolación con polinomios cúbicos

2.7. Notas de codificación y pruebas

El código completo de este algoritmo se encuentra en la misma carpeta del repositorio donde se extrajo este informe y tiene por nombre "ManoIntento2PolinomiosCubicos", las pruebas que se hicieron con este algoritmo se encuentran en la carpeta de "Anexos" con sus respectivos códigos, la prueba de cambio de origen tiene el nombre de "ManoIntento2PolinomiosCubicosCambiadoOrigen", y la de agregar más puntos es "ManoIntento2PolinomiosCubicosMasPuntos"

3. Interpolación de lineales, cuadráticas y cúbicas

3.1. Metodología para la selección de puntos

El criterio de selección de puntos para este algoritmo se basa en el **análisis manual**, donde se van identificando las tendencias de los puntos en comparación con el comportamiento expuesto por polinomios **lineales, cuadráticos y cúbicos**.

3.2. Selección de puntos manuales

Los puntos se eligieron de acuerdo con: - En los diferentes segmentos de la gráfica se iban probando las diferentes combinaciones entre puntos y tipos de polinomios mientras se rectificaba que

los puntos seleccionados pudieran **producir un polinomio valido.**

Fragmento del codigo encargado de la selecci3n de puntos

```
DatosX = NULL; DatosY = NULL;
DatosX[1] = x[3];
DatosX[2] = x[5];
DatosY[1] = y[3];
DatosY[2] = y[5];
PolinomioLagrange = poly.calc(DatosX,DatosY)
PolinomioLagrange

DatosX1 = NULL; DatosY1 = NULL;
DatosX1[1] = x[5];
DatosX1[2] = x[9];
DatosY1[1] = y[5];
DatosY1[2] = y[9];
PolinomioLagrange1 = poly.calc(DatosX1,DatosY1)
PolinomioLagrange1

DatosX2 = NULL; DatosY2 = NULL;
DatosX2[1] = x[9];
DatosX2[2] = x[11];
DatosX2[3] = x[12];
DatosY2[1] = y[9];
DatosY2[2] = y[11];
DatosY2[3] = y[12];
PolinomioLagrange2 = poly.calc(DatosX2,DatosY2)
PolinomioLagrange2

DatosX3 = NULL; DatosY3 = NULL;
DatosX3[1] = x[12];
DatosX3[2] = x[15];
DatosY3[1] = y[12];
DatosY3[2] = y[15];
PolinomioLagrange3 = poly.calc(DatosX3,DatosY3)
PolinomioLagrange3

DatosX4 = NULL; DatosY4 = NULL;
DatosX4[1] = x[15];
DatosX4[2] = x[18];
DatosY4[1] = y[15];
DatosY4[2] = y[18];
PolinomioLagrange4 = poly.calc(DatosX4,DatosY4)
```

PolinomioLagrange4

3.3. Algoritmo, requerimientos y codificación

Para que el algoritmo funcione necesita:

- Que el usuario conozca los datos X, Y de los puntos
- Que el usuario conozca el comportamiento de la figura para determinar los posibles polinomios a usar

3.4. Imagen de resultado

Mano Interpolación cúbica, cuadrática y lineal

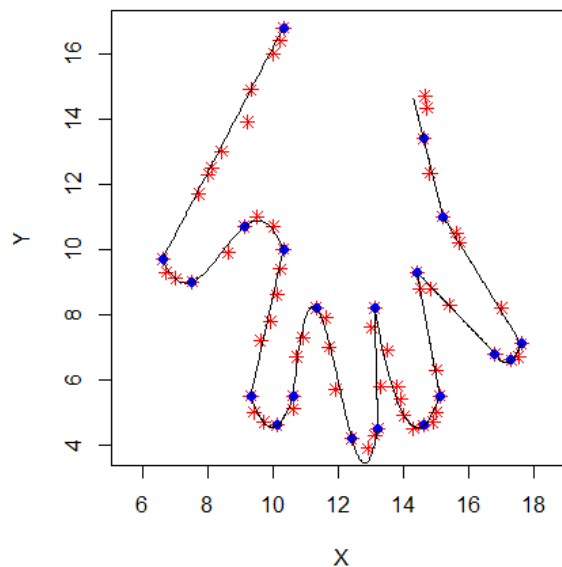


Figura 10: Interpolación con polinomios cúbicos, cuadráticos y lineales

3.5. Desventajas del Algoritmo

- No es autónomo, requiere de un usuario capaz de reconocer los puntos y comportamientos de los polinomios
- Soporta la nueva inserción de datos (solo si se modifica el código)
- La exactitud depende del usuario

3.6. Ventajas del algoritmo

- No depende del orden de los puntos para funcionar
- Maneja una cantidad de puntos menor que los dos algoritmos anteriores (20 puntos)
- Código menos complejo, no existen validaciones, solo entradas de datos, la construcción de los polinomios y su gráfica.

3.7. Notas de codificación y pruebas

Este es el algoritmo seleccionado para resolver el reto. El código completo de este algoritmo se encuentra en la misma carpeta del repositorio donde se extrajo este informe y tiene por nombre "ManoIntento3CubicasCuadraticasLineales", las pruebas que se hicieron con este algoritmo se encuentran en la carpeta de "Anexos" con sus respectivos códigos, para la prueba de cambiar el origen el código es "ManoIntento3CambiarOrigen" para la de agregar nuevos puntos el código es "ManoIntento3MasPuntos"

3.8. Polinomios utilizados

Cuadro 1: Add caption

Rango	polinomios
[3;5]	$71.8 - 4*x$
[5;9]	$35.7 - 1.625*x$
[9;11;12]	$764.34 - 88.49167*x + 2.583333*x^2$
[12;15]	$24.3 - 1.041667*x$
[15;18]	$87.47143 - 5.428571*x$
[18;21;27]	$441.286 - 60.57*x + 2.1*x^2$
[27;30]	$492.9 - 37*x$
[30;33;37;40]	$-4118.286 + 1039.407*x - 86.9356*x^2 + 2.413194*x^3$
[40;42;45]	$227.305 - 44.775*x + 2.25*x^2$
[45;50]	$-36.35 + 4.5*x$
[50;53;55;58]	$212.4478 - 74.7797*x + 9.037377*x^2 - 0.3578132*x^3$
[58;67]	$-2.964865 + 1.918919*x$

3.9. Tabla de los n-k puntos no seleccionados

Cuadro 2: Add caption

Posición	No seleccionados	
	x	y
1	14,65	14,71
2	14,71	14,33
4	14,8	12,33
6	15,6	10,5
7	15,7	10,22
8	17	8,2
10	17,52	6,7
13	15,4	8,3
14	14,83	8,8
16	14,5	8,8
17	15	6,3
19	15	5
20	14,9	4,7
22	14,3	4,5
23	14	4,9
24	13,9	5,4
25	13,8	5,8
26	13,5	6,9
28	13	7,6
29	13,3	5,8
31	13,1	4,3
32	12,9	3,9
34	11,9	5,7
35	11,7	7
36	11,6	7,9
38	10,9	7,3
39	10,7	6,7
41	10,6	5,1
43	9,7	4,7
44	9,4	5
46	9,6	7,2
47	9,9	7,8
48	10,1	8,6
49	10,2	9,4
51	10	10,7
52	9,5	11
54	8,6	9,9
56	7	9,1
57	6,7	9,3
59	7,7	11,7
60	8	12,3
61	8,1	12,5
62	8,4	13
63	9,2	13,91
64	9,3	14,9
65	10	16
66	10,2	16,4
67	10,3	16,8

4. Solución de las 5 preguntas

4.1. ¿El origen se puede modificar?

El origen se puede modificar, siendo necesario en el código tomar los puntos de manera inversa a lo realizado normalmente, debido a que se modifican los valores de los segmentos de la mano.

Esta información está sustentada con las pruebas que proporciona el algoritmo “ManoIntento3CambiarOrigen”, en este informe se presenta el siguiente resultado:

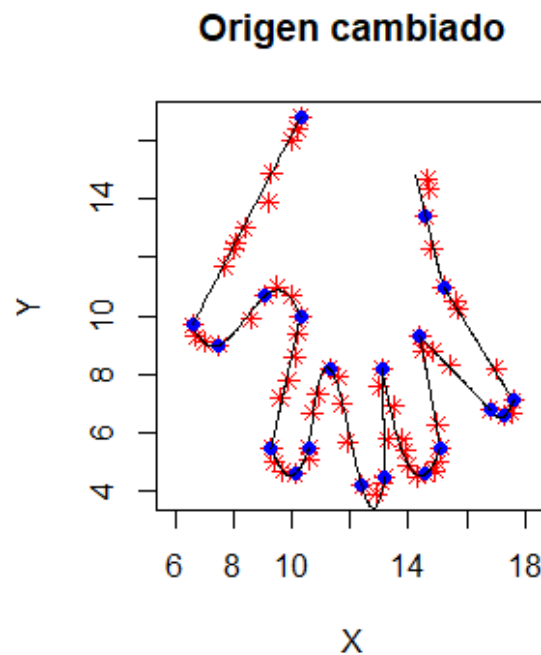


Figura 11: Pruebas del cambio de origen

4.2. ¿Si tenemos nueva información ósea nodos como podemos implementar esa información en el algoritmo de interpolación?

Debido a que es necesario escoger un rango específico de puntos para graficar la mano, agregar nuevos puntos que modifiquen significativamente el contorno de la mano dará como resultado que estos nuevos puntos afecten los segmentos ya seleccionados, siendo necesario actualizar y modificar los intervalos de puntos seleccionados a graficar en cada polinomio.

4.3. ¿Su método es robusto, en el sentido que si se tienen más puntos la exactitud no disminuye?

Al tomar mas puntos la mano no ve afectada su precisión de manera negativa, pero esto permite que se pueda tomar algunos de estos puntos nuevos y dibujar con una mayor suavidad y precisión el contorno de la mano

Esta información está sustentada con las pruebas que proporciona el algoritmo “ManoIntento3MasPuntos”, en este informe se presenta el siguiente resultado:

Mano agregando puntos

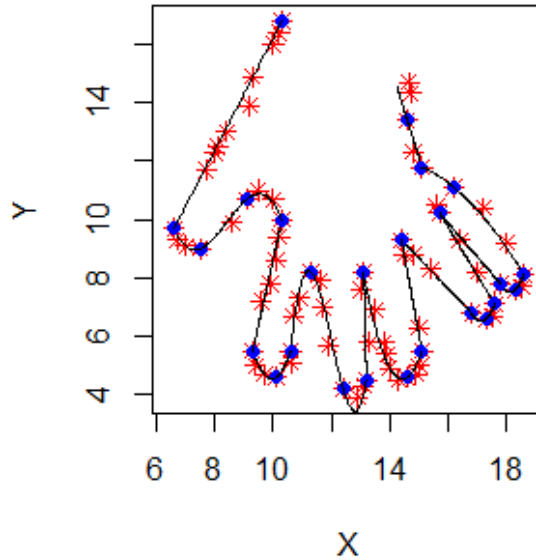


Figura 12: Pruebas del cambio de origen

4.4. ¿Si la información adicional o suponga tiene la información de otra mano con más cifras significativas cómo se comporta su algoritmo? la exactitud decae?

El algoritmo se ve afectado mínimamente, presentando en la gráfica de la mano un pequeño cambio en el contorno, pero ningún cambio significativo que afecte su resultado al dibujar el contorno.

5. índice de Jaccard

El código "ManoIntento3" contiene la siguiente función:

```
indiceJaccard <- function(x)
{
  j = (20/67)
  return (j)
}
```

$$J = 20/67 = 0,4029850746268 \quad (1)$$

6. Eficiencia del método

La eficiencia del método se mide en base a la cantidad de polinomios que se tuvieron que generar para graficar la mano, para el código actual **se generan 12 polinomios**, esta será su medida de eficiencia

6.1. *Calculos del error*

El cálculo del error se hizo teniendo presente que la mano se interpoló por segmentos, donde cada segmento tenía su propio polinomio, teniendo esto en cuenta el error para cada segmento se calculo evaluando los valores dados por el polinomio en los puntos “X” y comparándolo mediante una resta con los valores que debieron dar en “Y”.

6.1.1. *Error por punto*

Para cada punto se tomo como error la distancia de este con respecto a la gráfica dada por el polinomio, esta distancia se calculo evaluando el polinomio y comparando con su valor esperado en “Y”, la comparación solo toba valores absolutos para facilitar la medición de este error Esta información se encuentra en las tablas “Errores por cada punto de la gráfica” y “Parte dos: Errores por cada punto de la gráfica” expuestas en las siguientes paginas

6.1.2. *Error Total*

El Error total se calculo con la suma de los errores individuales de cada punto, dando como resultado: **397.2071**

6.1.3. *Error Promedio*

El Error promedio se calculo como el Error total dividido entre el numero de puntos de la gráfica dando el siguiente resultado: **5.92842**

Cuadro 3: Errores por cada punto de la gráfica

Dato	X	Y	Error
1	14.65	14.7	1,5
2	14.71	14.33	1,37
3	14.6	13.4	1,95E-08
4	14.8	12.33	0,27
5	15.2	11.0	1,42E-08
6	15.6	10.5	2,05
7	15.7	10.22	1.566.667
8	17.0	8.20	1.635
9	17.6	7.10	1,84E-06
10	17.52	6.70	0,65
11	17.3	6.60	0,3208333
12	16.8	6.80	2,66E-09
13	15.4	8.30	4.428.571
14	14.83	8.80	1.834.286
15	14.4	9.30	1,78E-08
16	14.5	8.80	4.254
17	15.0	6.30	1.064
18	15.1	5.50	2,84E-07
19	15.0	5.00	67,1
20	14.9	4.70	63,1
21	14.6	4.60	51,9
22	14.3	4.50	40,7
23	14.0	4.90	30
24	13.9	5.40	26,8
25	13.8	5.80	23,5
26	13.5	6.90	13,5
27	13.1	8.20	1,03E-07
28	13.0	7.60	3.920.641
29	13.3	5.80	0,6381132
30	13.2	4.50	7,28E-06
31	13.1	4.30	3,3
32	12.9	3.90	2,9
33	12.4	4.20	3,2
34	11.9	5.70	4,7
35	11.7	7.00	6
36	11.6	7.90	6,9
37	11.3	8.20	7,2
38	10.9	7.30	6,3
39	10.7	6.70	5,7
40	10.6	5.50	4,5

Cuadro 4: Parte dos: Errores por cada punto de la gráfica

Dato	X	Y	Error
41	10.6	5.10	0,4
42	10.1	4.60	1,08E-07
43	9.7	4.7	0,01
44	9.4	5.0	0,23
45	9.3	5.5	8,53E-08
46	9.6	7.2	0,35
47	9.9	7.8	0,4
48	10.1	8.60	0,5
49	10.2	9.40	0,15
50	10.3	10.0	7,11E-09
51	10.0	10.7	0,1247708
52	9.50	11.0	0,1162377
53	9.10	10.7	9,49E-07
54	8.6	9.9	0,2574834
55	7.5	9.0	6,54E-07
56	7.0	9.1	0,00861487
57	6.7	9.3	0,1946332
58	6.6	9.7	5,22E-07
59	7.70	11.7	0,1108108
60	8.00	12.3	0,08648649
61	8.10	12.5	0,07837838
62	8.40	13.0	0,1540541
63	9.20	13.91	0,7791892
64	9.30	14.9	0,01891892
65	10	16	0,2243243
66	10.2	16.4	0,2081081
67	10.3	16.8	3,55E-09

6.1.4. Notas del error

El código que tiene las instrucciones para calcular el error esta en la carpeta “Anexos” con el nombre “Calculo del error”