

Universidad de Buenos Aires  
Facultad de Ciencias Exactas y Naturales  
Departamento de Computación

# Métodos Numéricos

## Trabajo Práctico N°3

Más que Splines

Nombre	LU	Mail
Carla Livorno	424/08	carlalivorno@hotmail.com
Mariano De Sousa Bispo	389/08	marian_sabianaa@hotmail.com

### Abstract

El siguiente trabajo se propone la implementación de una curva paramétrica mediante *splines* naturales a partir de un conjunto de puntos.

Además, la curva provee la funcionalidad de seleccionar un punto cualquiera de la misma (generalmente este punto se encuentra meramente *cerca* de la curva) y moverlo a una nueva posición modificando la curva.

Spline cúbico natural      Curva paramétrica      Polinomios

# Índice

<b>1. Introducción teórica</b>	<b>2</b>
<b>2. Desarrollo</b>	<b>3</b>
2.1. Explicación . . . . .	3
2.2. Implementación . . . . .	4
2.3. Correctitud de <i>splines</i> . . . . .	8
<b>3. Resultados</b>	<b>9</b>
<b>4. Discusión</b>	<b>17</b>
<b>5. Conclusiones</b>	<b>19</b>
<b>6. Modo de compilación y uso</b>	<b>20</b>
<b>7. Apéndices</b>	<b>21</b>
7.1. Apéndice A: Enunciado . . . . .	21
<b>8. Referencias</b>	<b>24</b>

# 1. Introducción teórica

En este trabajo se utilizan splines cúbicos naturales como método de interpolación para generar una curva.

Un spline es una curva definida en porciones mediante polinomios (en este caso de grado 3). La idea central es que en vez de usar un único polinomio para interpolar todos los datos, se usan segmentos de polinomios entre puntos de control (pares coordenados) y se une cada uno de ellos adecuadamente para ajustar los datos. Los polinomios que definen la curva satisfacen ciertas condiciones específicas de continuidad en la frontera de cada intervalo para asegurar una transición suave.

Se utiliza a menudo la interpolación mediante splines porque da lugar a buenos resultados requiriendo solamente el uso de polinomios de bajo grado, evitando así las oscilaciones, indeseables en la mayoría de las aplicaciones, encontradas al interpolar mediante polinomios de grado elevado.

En este trabajo se utilizan curvas paramétricas en  $\mathbb{R}^2$  que dadas las coordenadas de los puntos de control definen la parametrización de la siguiente manera:

**Uniforme:** la variación del parámetro es igual entre cualquier par de puntos de control consecutivos;

***Chord-length:*** la variación del parámetro entre dos puntos de control consecutivos es proporcional a la distancia entre los mismos;

**Centrípeta:** la variación del parámetro es proporcional a la raíz cuadrada de la distancia entre los puntos de control.

## 2. Desarrollo

### 2.1. Explicación

Para construir la curva paramétrica se puede utilizar cualquiera de las *tres* parametrizaciones (uniforme, chord-length y centripeta) en el intervalo  $[0, 1]$ . La parametrización  $t$  se elige a partir de los puntos de control  $(x, y)$  recibidos en la entrada del programa. Una vez elegida la parametrización se generan dos splines, el primero  $S_x$  a partir del par  $(t, x)$  y el otro  $S_y$  a partir de  $(t, y)$ . De esta manera, queda definida una curva  $C \in \mathbb{R}^2$  donde  $C(t) = (S_x(t), S_y(t))$ .

Para mover un punto  $(x, y)$  cercano a la curva a una nueva posición  $(x^*, y^*)$ , calculamos primero el punto de la curva más próximo a  $(x, y)$  y luego construimos una nueva curva (manteniendo la parametrización original) de manera tal que ambas splines ( $S_x$  y  $S_y$ ) ahora pasen también por la nueva posición  $(x^*, y^*)$ .

Para calcular el punto de la curva más próximo a  $(x, y)$  minimizamos (derivamos y buscamos donde se anula distinguiendo entre máximos y mínimos) la función distancia<sup>1</sup> de la curva al punto  $(x, y)$  en cada intervalo  $[t_i, t_{i+1}] \in [0, 1]$  (polinomio que la conforma) y luego seleccionamos el mínimo en  $[0, 1]$ .

A continuación se detalla la búsqueda del punto más cercano a la curva dado  $(x, y)$ .

Sea  $n$  la cantidad de puntos de control y  $S_x^i, S_y^i$  el  $i$  - *esimo* polinomio de cada spline de la curva.

$$t / \min \sqrt{(S_x(t) - x)^2 + (S_y(t) - y)^2} \left\{ \begin{array}{l} t_1 / \min \sqrt{(S_x^{(1)}(t) - x)^2 + (S_y^{(1)}(t) - y)^2} \\ t_2 / \min \sqrt{(S_x^{(2)}(t) - x)^2 + (S_y^{(2)}(t) - y)^2} \\ \vdots \\ t_{n-1} / \min \sqrt{(S_x^{(n-1)}(t) - x)^2 + (S_y^{(n-1)}(t) - y)^2} \end{array} \right.$$

## 2.2. Implementación

La implementación esta dividida en módulos que realizan tareas específicas, a continuación detallaremos cada uno de ellos.

- Módulo Parametrización:

Este módulo implementa las *tres* parametrizaciones (uniforme, chord-length, centripeta) en el  $[0, 1]$  dado un conjunto de puntos de control.

- Módulo Spline:

Escribimos el módulo **Spline** que implementa un spline cúbico natural con las siguientes operaciones:

<b>Evaluar</b>	Evalua la spline (el polinomio correspondiente) en un valor recibido como parámetro
<b>Polinomio</b>	Devuelve el polinomio requerido.

- Módulo Curva:

El módulo **Curva** implementa una curva paramétrica con las siguientes operaciones:

<b>Punto cercano</b>	Dadas las coordenadas de un punto cercano a la curva, calcula el punto de la curva más próximo.
<b>Mover punto</b>	Dadas las coordenadas de un punto cercano a la curva, calcula el punto de la curva más próximo y construye una nueva spline resultante de modificar la spline original de manera que ahora pase por la nueva posición del punto seleccionado.
<b>Muestreo</b>	Devuelve un muestreo de la curva.

Esta clase cuenta con un método *private* (que se usa tanto para '**Punto cercano**' como para '**Mover Punto**') que busca el  $t \in [0, 1]$  tal que al evaluar la curva en  $t$  se obtiene el punto más cercano a la misma respecto de un punto dado  $(x, y)$ .

Se busca el  $t$  que minimiza la función distancia<sup>1</sup> del punto  $(x, y)$  a cada polinomio de la curva en el intervalo correspondiente  $([t_i, t_{i+1}])$  y se

---

<sup>1</sup>Distancia euclídea de  $(S_x(t), S_y(t))$  a  $(x, y)$ :  $\sqrt{(S_x(t) - x)^2 + (S_y(t) - y)^2}$

verifica si se puede actualizar el mínimo global, es decir, se selecciona el  $t$  que minimiza la distancia en  $[t_1, t_{i+1}]$ .

Llamamos  $d_i(t)$  a la distancia de  $(S_x^{(i)}(t), S_y^{(i)}(t))$  a  $(x, y)$ , es decir, la distancia a la curva en el intervalo  $[t_i, t_{i+1}]$  ( $i$  -ésimo polinomio).

Como la función distancia es BLABLABLA podemos minimizar  $d_i^2(t)$ .

$$d_i^2(t) = (S_x^{(i)} - x)^2 + (S_y^{(i)} - y)^2$$

Sea  $z = (t - t_i) \Rightarrow$

$$d_i^2(t) = (a_{x_i} + b_{x_i} * z + c_{x_i} * z^2 + d_{x_i} * z^3 - x)^2 + (a_{y_i} + b_{y_i} * z + c_{y_i} * z^2 + d_{y_i} * z^3 - y)^2$$

Para minimizar  $d_i^2(t)$  obtenemos la derivada:.

$$(d_i^2)'(t) = 2(S_x^{(i)}(t) - x)(S_x^{(i)}(t) - x)' + 2(S_y^{(i)}(t) - y)(S_y^{(i)}(t) - y)'$$

$$\text{Sea } E_{k_i} = 2(S_k^{(i)}(t) - k)(S_k^{(i)}(t) - k)'$$

$$E_{k_i} = 2b_{k_i}(a_{k_i} - k) + 2(b_{k_i}^2 + 2c_{k_i}(a_{k_i} - k)) * z + 6(c_{k_i}b_{k_i} + d_{k_i}(a_{k_i} - k)) * z^2 + 4(2d_{k_i}b_{k_i} + c_{k_i}^2) * z^3 + 10d_{k_i}c_{k_i} * z^4 + 6d_{k_i}^2 * z^5$$

$$\Rightarrow (d_i^2)'(t) = E_{x_i} + E_{y_i}$$

Nota: Esta expresión esta 'hardcodeada' en una función *private* que devuelve el polinomio 'distancia' derivado a partir de los coeficientes de  $S_x$ ,  $S_y$ .

Por último, buscamos los  $t$  donde  $(d_i^2)'(t)$  se anula. Estos  $t$  que son los puntos críticos de la función corresponden a máximos o a mínimos. Nos quedamos con el  $t$  tal minimiza  $d(t)$ .

A continuación exponemos el pseudocódigo de esta función con el objetivo de esclarecer su explicación:

```

PuntoMásProx(curva, (x, y))
  min_global = 0
  for i ← 1 to cantPtosControl − 1
    poli = distDerivada(curva, (x, y), i)
    ptosCriticos = ceros(poli, i, i + 1)
    min_t = minimo(ptosCriticos)
    dist_min_global = dist(Sx(min_global), Sy(min_global), x, y)
    dist_min_t = dist(Sx(min_t), Sy(min_t), x, y)
    if dist_min_t < dist_min_global then
      dist_min_global = dist_min_t
  return (Sx(min_global), Sy(min_global))

```

Para mover un punto  $(x, y)$  calculamos el punto de la curva más próximo como se explico previamente. Luego, separamos en los siguientes casos:

- Caso punto más cercano a  $(x, y)$  es un punto de control:

Se reemplaza el punto de control (aquel que es el punto más próximo a  $(x, y)$ ) por la posición final de  $(x, y)$  y se construye una nueva curva a partir de los nuevos puntos de control manteniendo la parametrización.

ACA PONER FIGURA!!!

- Caso punto más cercano a  $(x, y)$  no es un punto de control:

Se agrega como nuevo punto de control  $(x^*, y^*)$  (posición final de  $(x, y)$ ), además se agrega el  $t$  correspondiente a la parametrización (la parametrización para el resto de los puntos de control se mantiene). Para esto, se selecciona la posición que le corresponde a estos datos según un orden creciente de  $t$ . Luego, se construye una nueva curva que ahora también pase por  $(x^*, y^*)$ .

ACA PONER LAS DOS FIGURAS!!!

La última de las operaciones selecciona  $m$  puntos de muestreo, donde

$m$  es recibido en la entrada. Estos puntos corresponden a un muestreo uniforme del rango del parámetro  $([0, 1])$  incluyendo los extremos.

■ Módulo Polinomio:

Escribimos el módulo `Polinomio` que implementa un polinomio de grado  $n$  con las siguientes operaciones:

<b>Evaluar</b>	Evalúa el polinomio en un valor recibido por parámetro.
<b>Derivar</b>	Realiza la derivada primera del polinomio.
<b>Ceros</b>	Busca una raíz del polinomio usando bisección y el método de Newton.

Para encontrar los ceros de un polinomio en el intervalo  $[a, b]$  implementamos un algoritmo heurístico que detallamos a continuación:

```
ceros(polinomio, a, b)  
    long_intervalo = (b - a) / grado(polinomio)  
    b = a + long_intervalo  
    for i ← 0 to grado(polinomio) - 1  
        raices[i] = BuscarRaiz(polinomio, a, b)  
        a = b  
        b = b + long_intervalo  
    return raices
```

Dado que el polinomio tiene a lo sumo tantas raíces como su grado, dividimos  $[a, b]$  en esa cantidad de intervalos, apostando a que las mismas se encuentran uniformemente distribuidas, si esto sucede encontramos una raíz en cada intervalo. Para buscar cada una de ellas ejecutamos el método de bisección mientras sea posible (cambie de signo en el intervalo) o hasta acercarnos lo suficiente (parámetro definido por nosotros, Ver Pruebas!!!!). En ambos casos el algoritmo aplica en última instancia el método de *Newton*.

En los intervalos donde no hay ninguna raíz el procedimiento es el mismo, el valor conseguido es producto de que el método de *Newton* utilizó todas las iteraciones permitidas, no alcanzando este valor la tol-



erancia (parámetro definido por nosotros, Ver Pruebas!!!!) que le otorga la condición de cero del polinomio.

**Observación:** Si tenemos intervalos sin raíces implica que no vamos a encontrarlas todas, ya que obtenemos una raíz por intervalo (vamos a encontrar tantas raíces como intervalos con raíces tengamos).

Como lo que buscamos es el mínimo global del polinomio en el intervalo  $[a, b]$  y los polinomios a los que les buscamos los puntos críticos son de grado 5  $((d_i^2)'(t))$  obtenemos 5 raíces como se explicó previamente. Luego elegimos aquella que al evaluar la curva nos da un valor menor a todas las demás, es por esto que podemos tolerar tener valores que no son considerados raíces ya que serán descartadas o no en el caso de no haber podido hallar la raíz que corresponde al mínimo global y el resto correspondan a máximos (ese es uno de los posibles casos).

### 2.3. Correctitud de *splines*

Con el fin de probar que efectivamente los cálculos sean correctos, se creó la clase *SplineTester* la cual exporta un constructor que recibe el *spline* a analizar y un método *isSuccessful* que devuelve un valor booleano correspondiente a si cumplió con todas las condiciones para ser un spline o no.

Utilizamos *splines* naturales.

Las siguientes son las condiciones que valida el objeto *SplineTester*:

- La derivada segunda en los puntos de control bordes es cero (natural).
- Los resultado de evaluar el polinomio a izquierda y derecha de un punto de control en si mismo son iguales (en los casos bordes, es decir primer y último punto, se toma el primer y último *spline* respectivamente).
- La primer derivada de cada par de polinomios contiguos evaluadas en el punto de control son iguales (valen las mismas consideraciones para el primer y último punto de control).
- La derivada segunda de cada par de polinomios evaluadas en el punto de control que las une son iguales (iguales consideraciones que en el item anterior).

### 3. Resultados

El algoritmo de *Newton* utilizado para buscar los ceros de un polinomio hace uso de dos parámetros, los cuales tuvimos que ajustar de manera de conseguir los mejores resultados posibles, mejores en el sentido de relación calidad de la solución y eficiencia en terminos de tiempo del algoritmo.

Uno de los parámetros es para determinar la máxima cantidad de iteraciones que le permitimos al algoritmo buscar. Para ajustar este parámetro BALABALABNAKABALKABKANSJBFDKBFKDBC.

La siguiente prueba se realizó para ajustar un parámetro (*tolerancia*) del programa que sirve buscar los *ceros* de un polinomio. Este parámetro se utiliza para decidir si un valor es una raíz (más específicamente nos dice si estamos cerca del cero).

Se muestra el siguiente gráfico para esclarecer el uso de este parámetro.

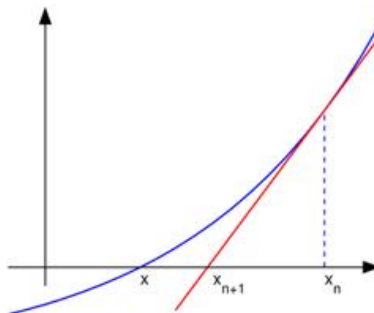


Figura 1: Método de Newton

Si el algoritmo de *Newton* termina porque consigue la *tolerancia* pedida (no se le terminan las iteraciones permitidas) devuelve  $x_{n+1}$  y la distancia entre  $x_{n+1}$  y  $x_n$  es menor a la *tolerancia* ( $x_{n+1} - x_n < \text{tolerancia}$ ).

Esperamos ver que cuanto menor es la *tolerancia* mejor es la aproximación al cero teórico. Recordar que el algoritmo que busca los ceros de un polinomio tiene dos criterios de parada (la tolerancia y la cantidad de iteraciones permitidas), creemos que la aproximación va a mejorar al disminuir la tolerancia ya que si termina por el primer motivo el valor conseguido va a ser igual o más refinado (más cercano a 0) y si lo hace por el segundo va a mejorar la aproximación durante más iteraciones. A pesar de esto, debemos buscar un equilibrio entre la calidad de la solución y el costo de la misma. Realizamos el siguiente gráfico con el fin de conseguir un valor adecuado para

dicho parámetro, se grafica costo del algoritmo en función de la calidad de la solución encontrada (evaluar el polinomio en la raíz obtenida por el algoritmo) para distintos valores de *tolerancia*:  $1e^{-1}$ ,  $1e^{-3}$ ,  $1e^{-5}$ ,  $1e^{-9}$ ,  $1e^{-13}$ ,  $1e^{-17}$  y  $1e^{-24}$ . Corrimos la prueba para 35 instancias generadas aleatoriamente (hicimos un generador de polinomios, eligiendo de manera 'random' cada coeficiente).

El gráfico se presenta bajo escala logarítmica en  $y$

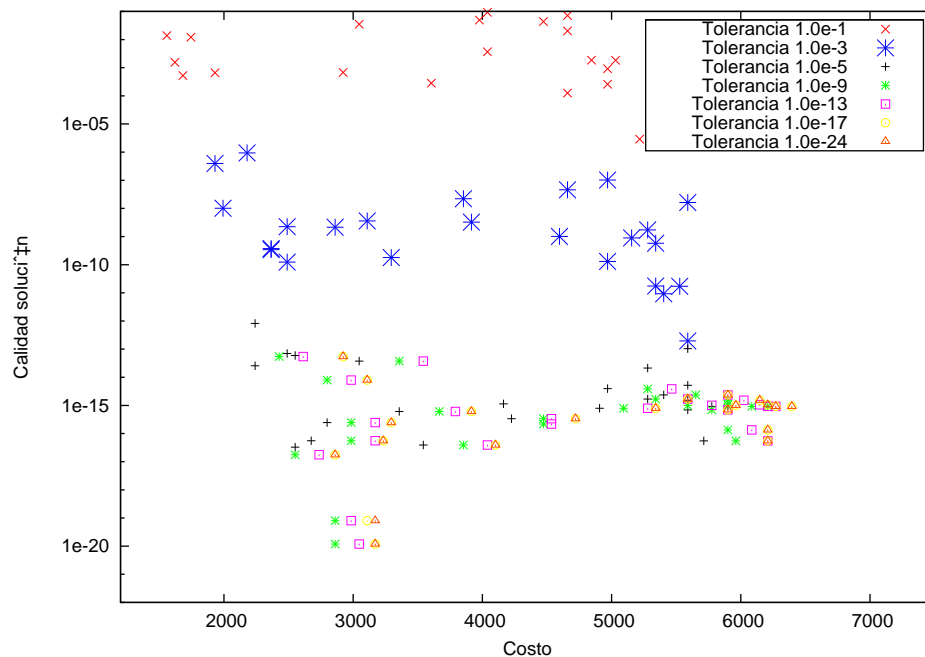


Figura 2: Costo del algoritmo de Newton en función de la calidad de la solución variando el parámetro tolerancia

Para poder analizar si existen diferencias en el uso de distintas parametrizaciones se creó un generador de puntos de control a partir de un par de poli-

nomios (uno para la coordenada  $x$  y otra para  $y$  que fueron elegidos de manera arbitraria) los cuales forman la curva. Los puntos de control pueden ser elegidos de forma aleatoria o uniforme. Graficamos entonces la curva con los puntos de control seleccionados de alguna de las dos formas anteriormente mencionadas, y las curvas utilizando las distintas de parametrizaciones (uniforme, por longitud de cuerda y centripeta).

Los siguientes gráficos utilizan un conjunto de cinco puntos de control aleatorios dada la misma curva.

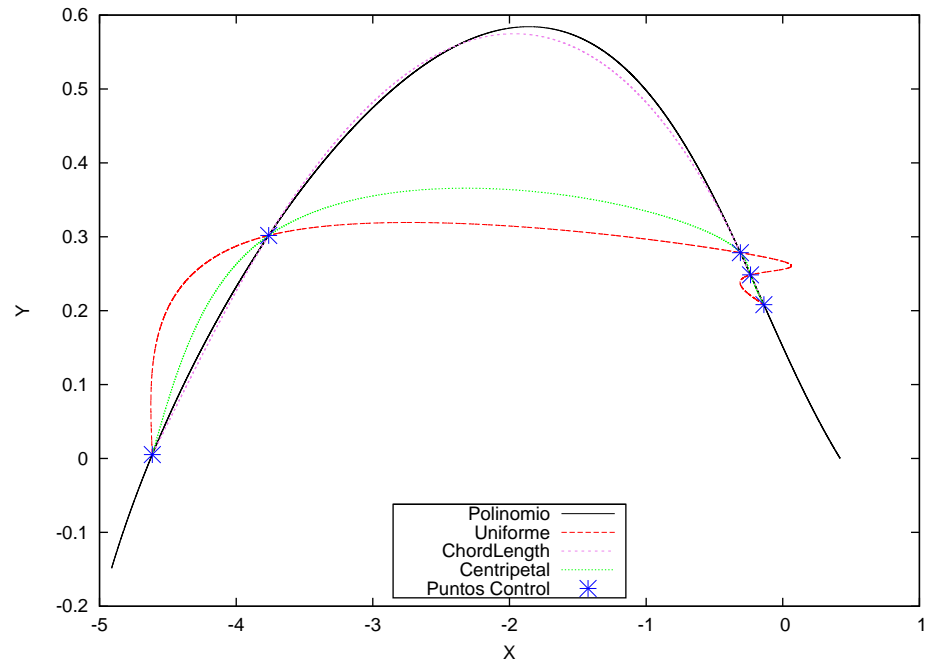


Figura 3: Aproximación por las distintas parametrizaciones utilizando cinco puntos de control aleatorios

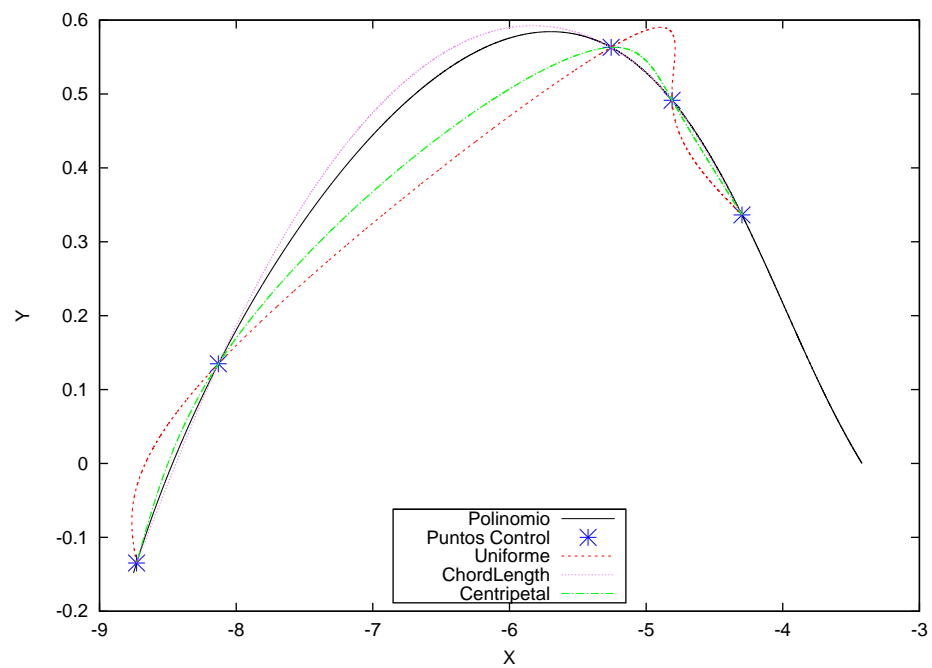


Figura 4: Aproximación por las distintas parametrizaciones utilizando cinco puntos de control aleatorios

A continuación, se utiliza el mismo polinomio que el gráfico anterior pero tomando los puntos de control uniformemente.

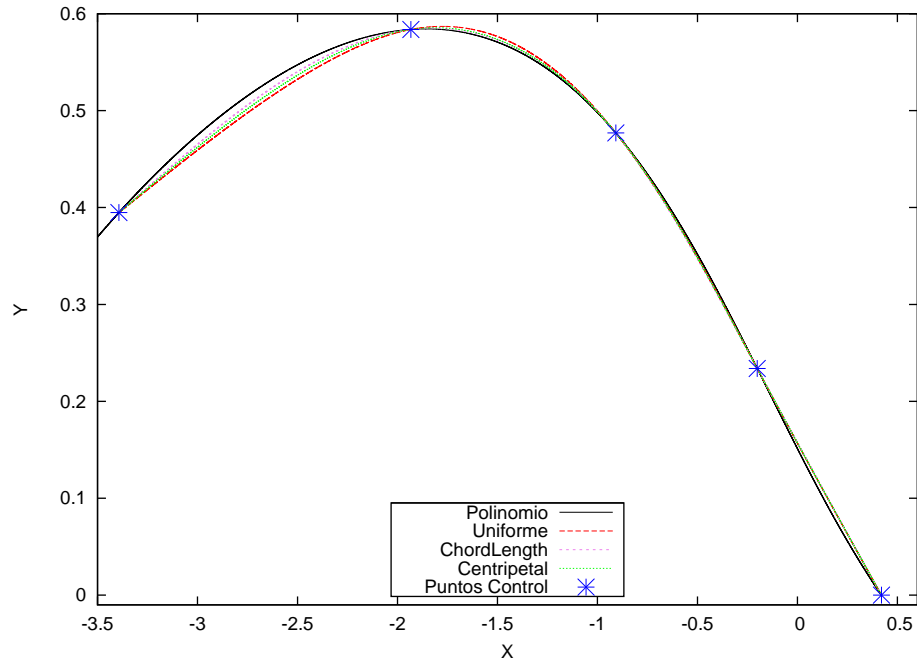


Figura 5: Aproximación por las distintas parametrizaciones utilizando cinco puntos distribuidos uniformemente

Los siguientes gráficos detallan cómo cambia la aproximación a una curva dada una parametrización a medida que se le proveen más puntos de control. Los puntos fueron tomados aleatoriamente y se utilizó la misma curva para las tres parametrizaciones para una comparación y análisis más claro.

El primero de los gráficos muestra la variación en la aproximación para la parametrización uniforme.

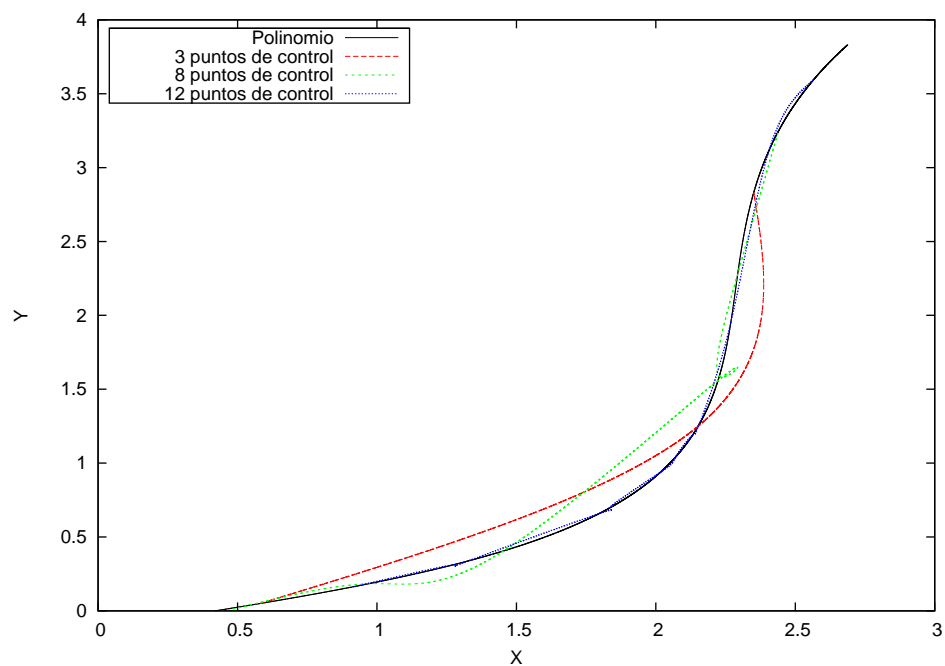


Figura 6: Aproximación utilizando parametrización uniforme

El siguiente gráficos muestra la variación en la aproximación para la parametrización centripeta.

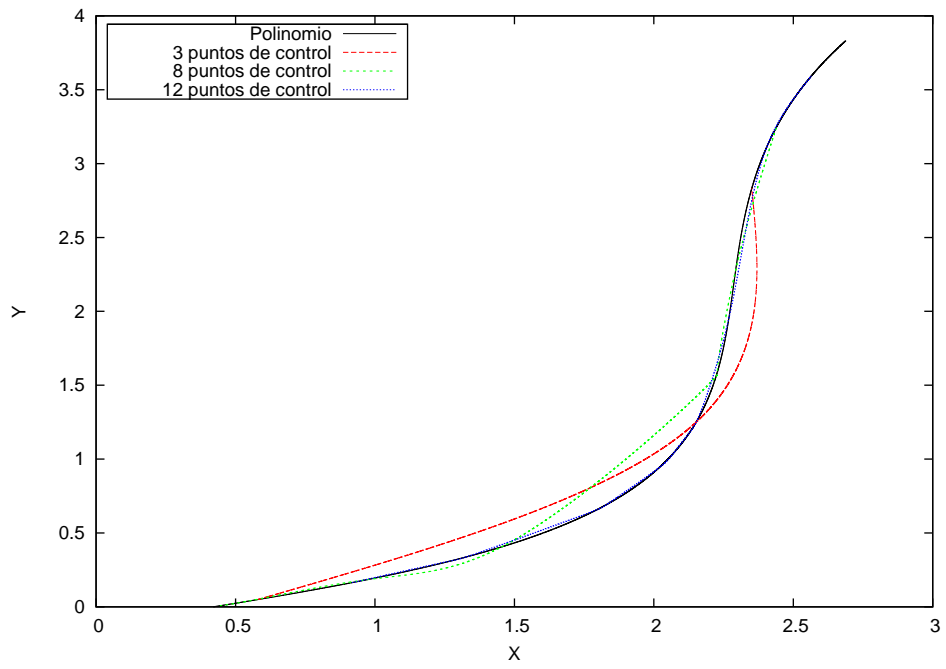


Figura 7: Aproximación utilizando parametrización centrípeta

Por ultimo, se muestra la variación utilizando parametrización por longitud de cuerda.



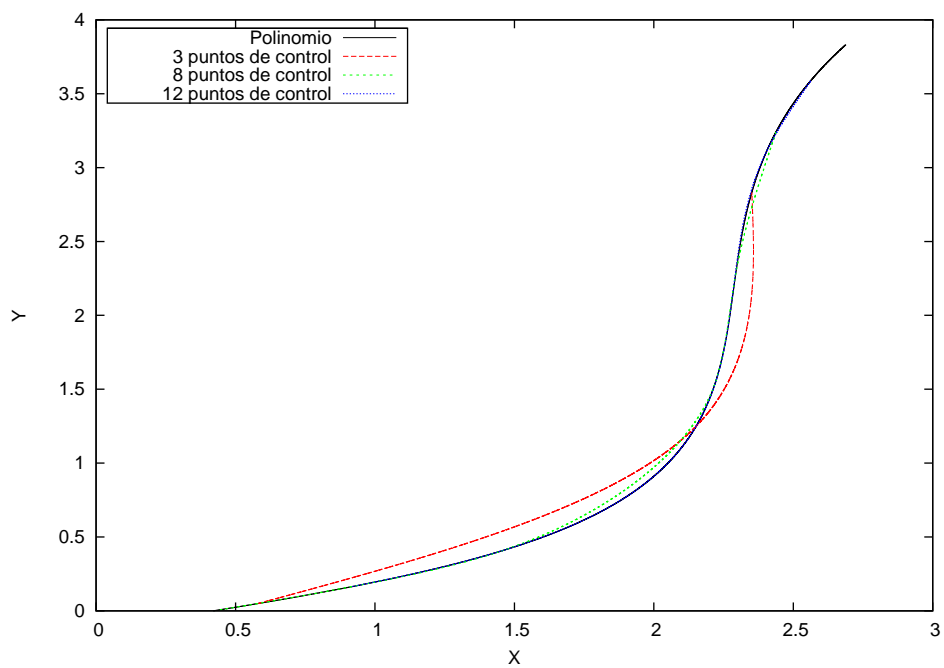


Figura 8: Aproximación utilizando parametrización por longitud de cuerda

## 4. Discusión

En esta sección, buscaremos conclusiones a la información suministrada por los gráficos de la sección anterior.

CHAMUZCO GRAFICOS ANTERIORES CHAMUZCO GRAFICOS AN-  
TERIORES CHAMUZCO GRAFICOS ANTERIORES CHAMUZCO GRAFI-  
COS ANTERIORES CHAMUZCO GRAFICOS ANTERIORES CHAMUZ-  
CO GRAFICOS ANTERIORES CHAMUZCO GRAFICOS ANTERIORES  
CHAMUZCO GRAFICOS ANTERIORES CHAMUZCO GRAFICOS AN-  
TERIORES CHAMUZCO GRAFICOS ANTERIORES CHAMUZCO GRAFI-  
COS ANTERIORES CHAMUZCO GRAFICOS ANTERIORES CHAMUZ-  
CO GRAFICOS ANTERIORES CHAMUZCO GRAFICOS ANTERIORES

En el tercer y cuarto gráfico (Figura:3 y Figura:4) vemos que la elección de la parametrización define la forma en la que se aproxima la curva. Si bien las tres parametrizaciones cumple todas las condiciones exigidas por los splines, la que utiliza la parametrización por longitud de cuerda es la que mejor aproxima a la curva original mientras que la parametrización uniforme consigue la peor aproximación.

Observamos que la parametrización uniforme (basandonos en estas instancias) tiende a unir los puntos con líneas más rectas cuanto más separados se encuentran estos. Por otra parte, cuando tenemos dos puntos cercanos posteriores a un par de punto lejanos la línea sigue su trayectoria y luego retoma para unirse con el punto correspondiente.

Inferimos además que, para la parametrización por longitud de cuerda, no es necesario tener puntos de control muy cercanos entre si para obtener una buena aproximación, (ver segundo y tercer punto de control).

La parametrización centrípeta si bien es mejor que la uniforme (en estos caso), evitando por ejemplo la oscilación entre los tres puntos de control cercanos entre sí, dista considerablemente de la curva original y de la aproximación a la curva bajo la parametrización por longitud de cuerda.

En el otro gráfico donde los puntos de control fueron seleccionados de manera uniforme (Figura:5) observamos que a pesar de la proximidad de las *cuatro* curvas la parametrización por longitud de cuerda sigue siendo la que mejor aproxima y uniforme la que peor lo hace.

Concluimos finalmente que la parametrización por longitud de cuerda tiene trayectorias suaves en contraste con las demás (centripeta y uniforme),

siendo centripeta a su vez más suave que uniforme.

Además, la parametrización uniforme es totalmente dependiente de la distribución de los puntos de control mientras que el resto de las parametrizaciones (centripeta y longitud de cuerda) lo son en menor medida, siendo longitud de cuerda la menos dependiente de este suceso.

En los gráficos (Figura:6, 7y 8) podemos ver que a medida que la cantidad de puntos de control aumenta, la aproximación se acerca cada vez mas a la curva real. De todas formas, la velocidad de acercamiento varía dependiendo de la parametrización, pudiendo notar (basandonos en la curva analizada), que la parametrización que mayor cantidad de puntos de control necesita para aproximar la curva es la uniforme, seguido de la centripeta y por último la parametrización por longitud de cuerda.

En el caso de la parametrización por longitud de cuerda, observamos que dada la precisión del gráfico, este no aporta información sobre la existencia de diferencias con respecto a la curva real. Por lo tanto, y suponiendo que quisieramos visualizar y/o analizar datos con esa precisión, no sería necesario aumentar la cantidad de puntos de control.

## 5. Conclusiones

## 6. Modo de compilación y uso

Para compilar se hace uso de la herramienta Makefile.

Abrir una terminal dentro la carpeta *code* entregada y escribir el comando "make".

Para ejecutar el programa: `./tp3 input output` (donde el archivo input cumple las condiciones del enunciado).

## 7. Apéndices

### 7.1. Apéndice A: Enunciado

#### Laboratorio de Métodos Numéricos - Primer cuatrimestre de 2011 Trabajo Práctico Número 3: CAD - Más que splines

---

Los programas de diseño asistido por computadora (CAD) son herramientas fundamentales para ingenieros, arquitectos, diseñadores, artistas y animadores. Sus interfaces gráficas esconden un sinnúmero de complicadas operaciones. Un ejemplo básico de tales operaciones es la tarea de seleccionar un punto cualquiera de una curva y moverlo a una nueva posición deformando la curva. El punto seleccionado se ingresa usualmente mediante un dispositivo apuntador (*mouse* o tableta digitalizadora) interactuando con la interfaz, y en general el punto ingresado está meramente *cerca* de la curva.

En este trabajo práctico se deberán diseñar algoritmos e implementar un programa que, dadas las coordenadas  $(\bar{x}_i, \bar{y}_i) \in \mathbb{R}^2$  de una serie de puntos de control ( $i = 1..n$ ), construya una spline natural<sup>2</sup> paramétrica que pase por los puntos en el orden dado. Además, dadas las coordenadas  $(x^*, y^*) \in \mathbb{R}^2$  de un punto cercano a la curva, calcule el punto de la curva más próximo y construya una nueva spline natural resultante de modificar la spline original de forma que ahora además pase por la nueva posición  $(\bar{x}^*, \bar{y}^*) \in \mathbb{R}^2$  del punto seleccionado. El procedimiento descrito se muestra en la figura con la spline original dibujada en línea de trazos y la spline deformada en línea continua.

El programa deberá trabajar las splines como curvas paramétricas en  $\mathbb{R}^2$ . Dadas las coordenadas de los puntos de control existen varias estrategias para definir la parametrización. Algunas de las parametrizaciones comúnmente utilizadas son:

**Uniforme:** la variación del parámetro es igual entre cualquier par de puntos de control consecutivos;

---

<sup>2</sup>Esto es suficiente para nuestro TP, pero en realidad los sistemas de CAD utilizan más frecuentemente otros mecanismos para obtener, describir y manipular curvas y superficies.

**Chord-length:** la variación del parámetro entre dos puntos de control consecutivos es proporcional a la distancia entre los mismos;

**Centrípeta:** la variación del parámetro es proporcional a la raíz cuadrada de la distancia entre los puntos de control<sup>3</sup>.

En este trabajo práctico deberán utilizar alguna de estas parametrizaciones. Opcionalmente podrán implementar las restantes y comparar los resultados obtenidos con las tres variantes.

Además, el programa deberá conservar el valor del parámetro que le corresponde a cada punto de control y al punto seleccionado, antes y después de moverlo.

### Preguntas:

1. ¿Depende la forma de la curva de la elección de la parametrización?
2. ¿Cambia la forma de la curva si en lugar de deformar la curva conservando la parametrización el programa la recalcula al mover el punto?
3. (Opcional) ¿Cómo cambia la forma de la curva según la condición de borde usada (natural, sujeto, *not-a-knot*, etc.)?
4. (Opcional) Si se quiere redibujar continuamente la curva mientras el usuario mueve el punto seleccionado, ¿cómo se puede calcular esto más eficientemente?
5. (Opcional) Luego de mover el punto seleccionado, ¿cambia toda la curva (*control global*) o solamente una parte (*control local*)? ¿Qué consecuencias puede tener esto?
6. (Opcional) Si el intervalo del parámetro se muestrea uniformemente, ¿los puntos resultantes quedan espaciados uniformemente? ¿Qué otras alternativas de muestreo serían apropiadas?
7. (Opcional) Si se necesitara que las longitudes de curva entre puntos consecutivos sean todas iguales, ¿cómo debería muestrearse?

### Archivos de entrada / salida

La entrada de datos se realizará mediante un archivo de texto con el siguiente formato:

---

<sup>3</sup>Este método fue propuesto por Eugene Lee en *Choosing nodes in parametric curve interpolation*, Computer-Aided Design 21, 1989.

- En la primera línea figurará el número  $n$  de puntos de control utilizados para definir la spline y, separado por espacio, el número  $m$  de puntos de muestreo de la spline.
- En las siguientes  $n$  líneas figurarán las coordenadas  $\bar{x}$  e  $\bar{y}$  de cada punto de control separadas por espacio.
- Una línea en blanco
- Una línea con las coordenadas  $x^*$  e  $y^*$  del punto próximo a la curva, separadas por espacio.
- Una línea en blanco
- Una línea con las coordenadas  $\bar{x}^*$  e  $\bar{y}^*$  de la nueva posición del punto, separadas por espacio.

La salida de datos estará dada por un archivo de texto con el siguiente formato:

- En la primera línea figurará el número  $m$  de puntos muestreados.
- En las siguientes  $m$  líneas figurarán las coordenadas  $x$  e  $y$  de cada punto muestreado en la spline original, separadas por espacio. Estos puntos corresponderán a un muestreo uniforme del rango del parámetro e incluirán los extremos. De esta forma, probablemente este conjunto de puntos no incluya los puntos de control originales.
- Una línea en blanco
- Una línea con las coordenadas del punto en la curva original más próximo al punto ingresado, separadas por espacio.
- Una línea en blanco
- En las siguientes  $m$  líneas figurarán las coordenadas  $x$  e  $y$  de cada punto muestreado en la spline deformada, separadas por espacio.



## 8. Referencias