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

Métodos Numéricos

Trabajo Práctico Nº3

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 mediantes *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

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 parámetrica se puede utilizar cualquiera de las *tres* parametrizaciones (uniforme, chord-length y centrípeta) 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[1] 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 busqueda 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.

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

2.2. Implementación

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

- Módulo Parametrización:
  Este módulo implementa las *tres* parametrizaciones (uniforme, chord-length, centrípeta) 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:

`Evaluar`	Evalua la spline (buscando previamente el polinomio correspondiente) en un valor recibido como parámetro.
`Polinomio`	Devuelve el polinomio requerido.

- Módulo Curva:
  El módulo `Curva` implementa una curva parámetrica con las siguientes operaciones:

`Punto cercano`	Dadas las coordenadas de un punto cercano a la curva, calcula el punto de la curva más próximo.
`Mover punto`	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.
`Muestreo`	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[1] del punto (x, y) a cada polinomio de la curva en el intervalo correspondiente ($[t_i, t_{i+1}]$) y se

[1] 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 $[t_i, t_{i+1}]$ ($i - esimo$ polinomio).

Como la función distancia ($d_i(t)$) es monótona creciente en $\mathbb{R}_{>0}$ podemos minimizar $d_i^2(t)$ ya que coinciden en los puntos críticos (mínimo y máximo).

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

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

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 se encuentra '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
        pol_i = distDerivada(curva, (x, y), i)
        ptosCriticos = ceros(pol_i, i, i + 1)
        min_t = minimo(ptosCriticos)
        dist_min_global = dist(S_x(min_global), S_y(min_global), x, y)
        dist_min_t = dist(S_x(min_t), S_y(min_t), x, y)
        if dist_min_t < dist_min_global then
            dist_min_global = dist_min_t
    return (S_x(min_global), S_y(min_global))
```

Para mover un punto (x, y) calculamos el punto de la curva más próximo como se explicó 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.

  La siguiente figura ejemplifica este caso.

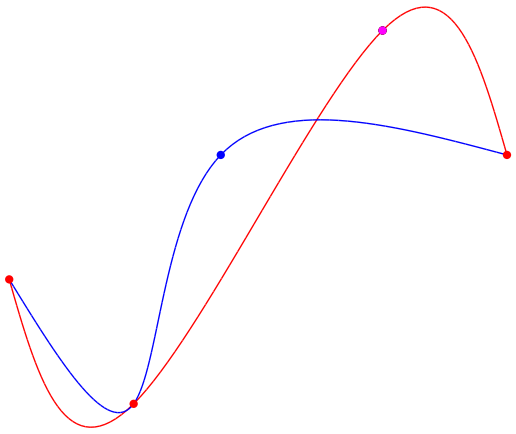

Figura 1: Punto más próximo a la curva corresponde a un punto de control de la misma

En este caso, la curva original (*roja*) se construyó a partir de *cuatro* puntos de control (los puntos *rojos* y el *rosa*), se seleccionó el punto *rosa* por lo que sí mismo es el más cercano y se lo desplazó como ya se explicó a la posición donde está el punto *azul*.

- 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 paramerización 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 $(x*, y*)$.

  La siguiente figura es un ejemplo de esto, donde el punto seleccionado no está sobre la curva y el punto más próximo a este no es un punto de control.

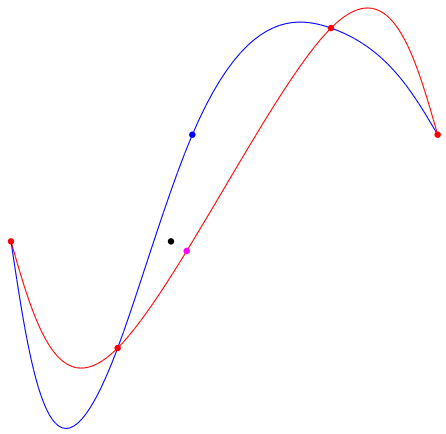

Figura 2: El punto seleccionado no pertenece a la curva y el punto más próximo no es un punto de control

La curva *roja* corresponde a la curva original y los puntos *rojos* a los puntos de control provistos para su construcción.

El punto *negro* es el punto seleccionado para ser movido, y el *rosa* el punto perteneciente a la curva más próximo a este.

La curva *azul* es la curva deformada de forma tal que el punto *rosa* toma la posición del punto *azul*.

Por último, la siguiente figura refleja el comportamiento cuando el punto seleccionado está sobre la curva.

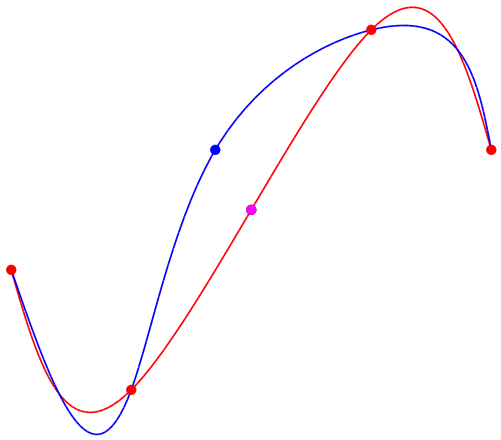

Figura 3: Punto seleccionado pertenece a la curva pero no es un punto de control

Como en las figuras anteriores la curva *roja* corresponde a la curva original. Los puntos de control son lo de color *rojo*.

El punto seleccionado se ve en color *rosa* porque el punto perteneciente a la curva más próximo es sí mismo. Finalmente, vemos en *azul* la curva deformada y la posición final del punto elegido.

La última de las operaciones de la clase *curva* (muestreo) 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:

`Evaluar`	Evalua el polinomio en un valor recibido por parámetro.
`Derivar`	Realiza la derivada primera del polinomio.
`Ceros`	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 algortimo 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 distribuídas, 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 subsección 'Ajuste de parámetros' en la sección de resultados así como discusión). 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 la tolerancia exigida (parámetro definido por nosotros, para más información ver nuevamente la subsección 'Ajuste de parámetros' en la sección de resultados asi como su análisis en discusión). La tolerancia que tomamos que le otorga a un valor real 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 la efectiva correctitud de los cálculos, 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 resultados de evaluar el polinomio a izquierda y derecha de un punto de control son iguales (en los casos bordes, es decir primer y último punto, se toma únicamente 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

3.1. Ajuste de parámetros

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.

La siguientes pruebas se realizaron para ajustar el parámetro (*tolerancia*) del programa que sirve para buscar los *ceros* de un polinomio y la máxima cantidad de iteraciones permitidas. El primer parámetro se utiliza para decidir si un valor es una raíz (más específicamente nos dice si estamos cerca del cero). La *tolerancia* es definida como la distancia entre un par de puntos, que en nuestro caso corresponde a las posibles raíces

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

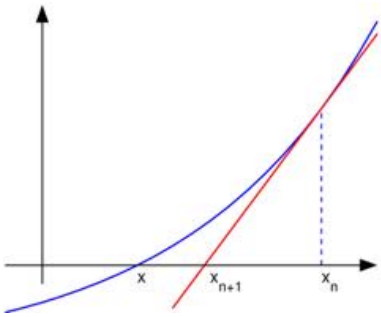

Figura 4: Método de Newton

En el gráfico anteriormente expuesto esta *tolerancia* se define como el valor absoluto entre la diferencia entre x_{n+1} y x_n. Si el algortimo 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* ($x_{n+1} - x_n < tolerancia$).

Como ya mencionamos, uno de los parámetros es para determinar la máxima cantidad de iteraciones que le permitimos al algoritmo buscar. Para ajustar este parámetro elegimos de manera aleatorea *siete* instancias (polinomios) y realizamos un gráfico de cantidad de iteraciones en función del error al cero teórico.

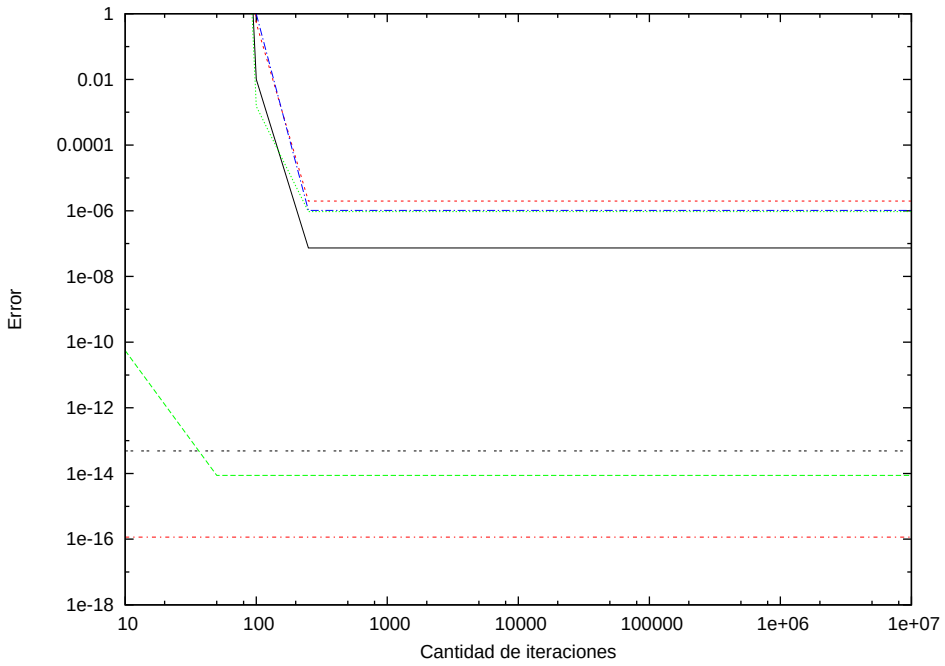

Figura 5: Cantidad de iteraciones en función del error al cero teórico

Esperamos con este gráfico poder ver la cantidad de iteraciones promedio necesarias para que la solución puede considerarse correcta. Además esperamos poder dilucidar alguna relación entre la cantidad de iteraciones adecuadas y el valor de *toleracia* óptimo al cero teórico.

A continuación presentamos un nuevo gráfico utilizando ahora como coordenada x el costo en cantidad de operaciones del algoritmo y como y el error cometido, es decir la distancia a la raíz real. Se utilizaron para este *test* treinta y cinco (35) instancias aleatoreas, para distintos valores de *tolerancia*: $1e^{-}1$, $1e^{-}3$, $1e^{-}5$, $1e^{-}9$, $1e^{-}13$, $1e^{-}17$ y $1e^{-}24$ (se utilizó un generador de polinomios creado por nosotros, eligiendo de manera 'random' cada coeficiente).

El gráfico se presenta bajo escala logarítmica en y

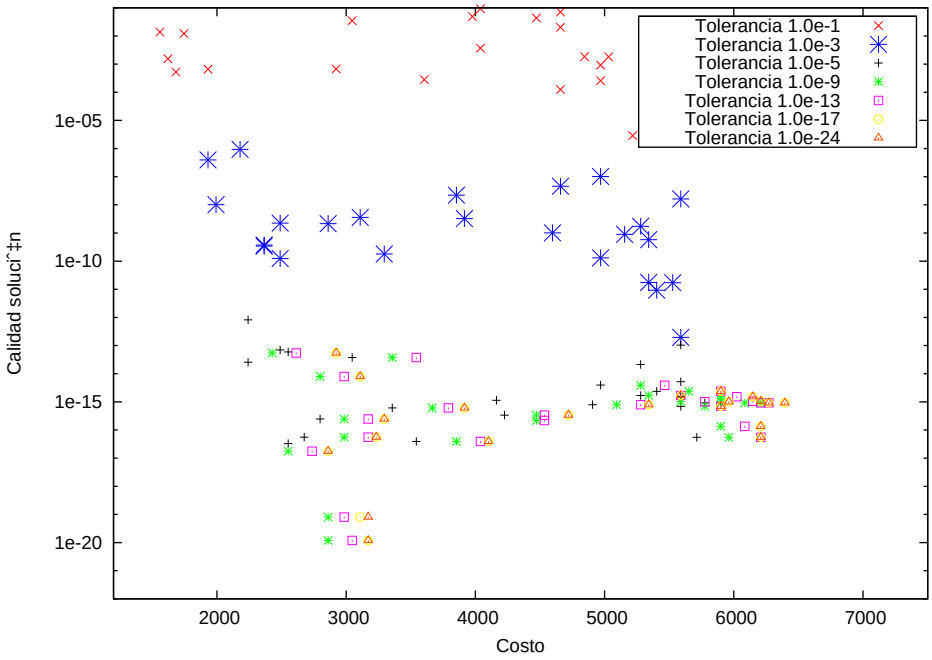

Figura 6: Costo del algoritmo de Newton en función de la calidad de la solución variando el parámetro tolerancia

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.

3.2. Análisis de curvas

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 polinomios (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 centrípeta).

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

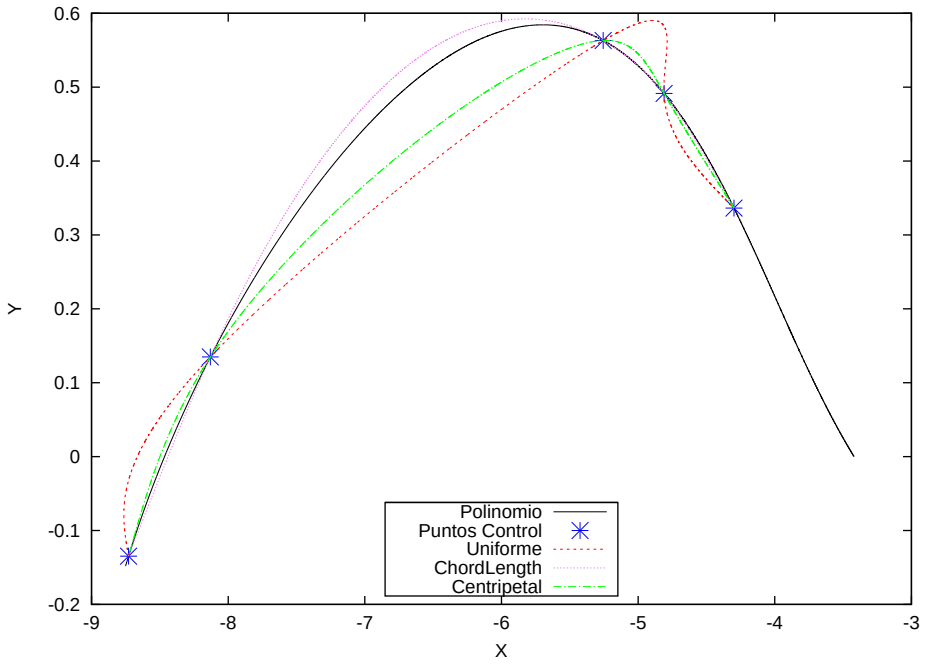

Figura 7: Aproximación por las distintas parametrizaciones utilizando cinco puntos de control aleatoreos

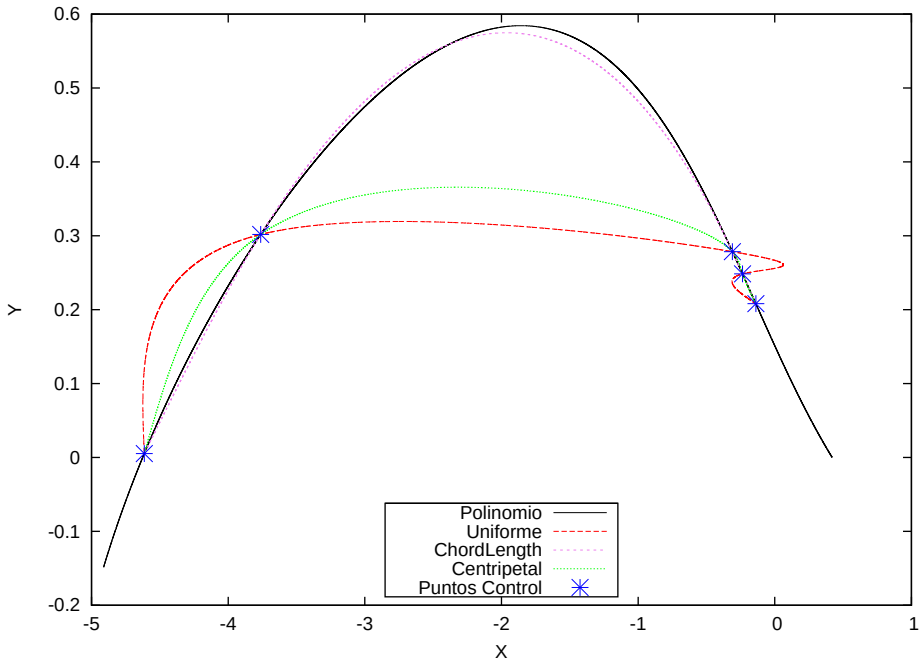

Figura 8: Aproximación por las distintas parametrizaciones utilizando cinco puntos de control aleatoreos

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

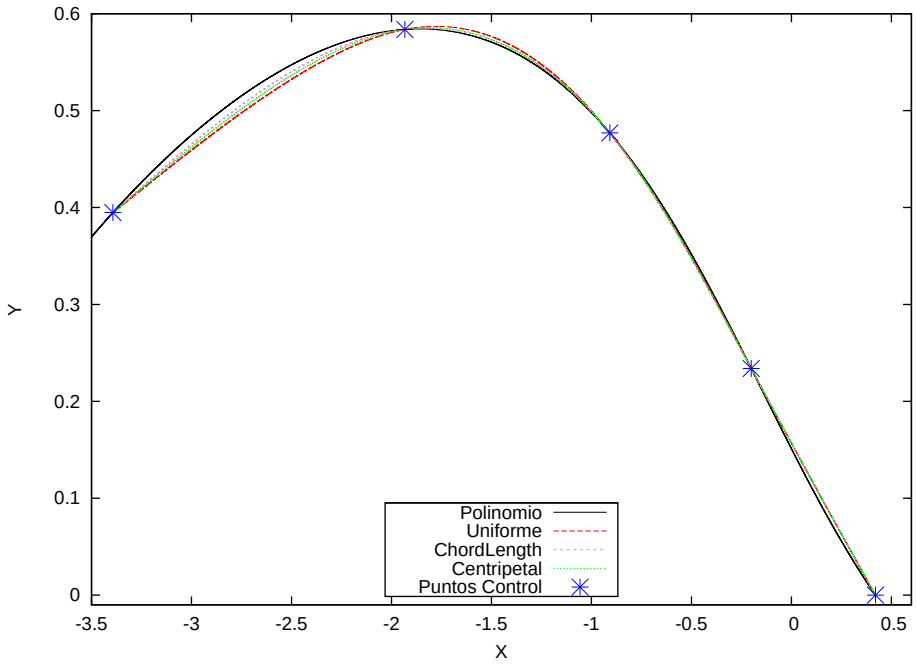

Figura 9: Aproximación por las distintas parametrizaciones utilizando cinco puntos distribuidos uniformemente

Los siguientes gráficos detallan cómo cambia la aproximación a una curva dada una parametrización a medida que se le provee más puntos de control. Los puntos fueron tomados aleatoreamente 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.

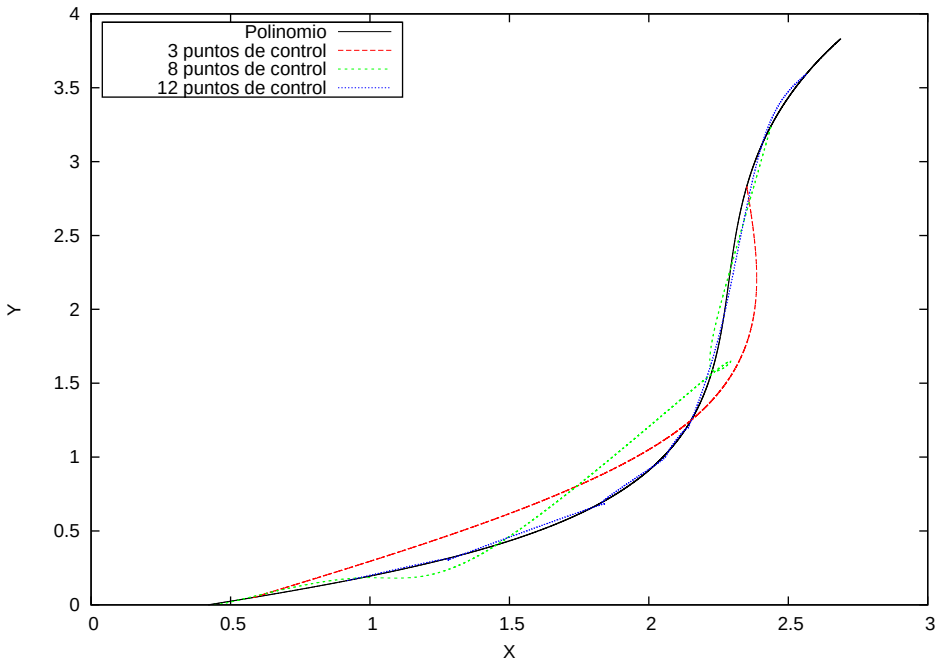

Figura 10: Aproximación utilizando parametrización uniforme

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

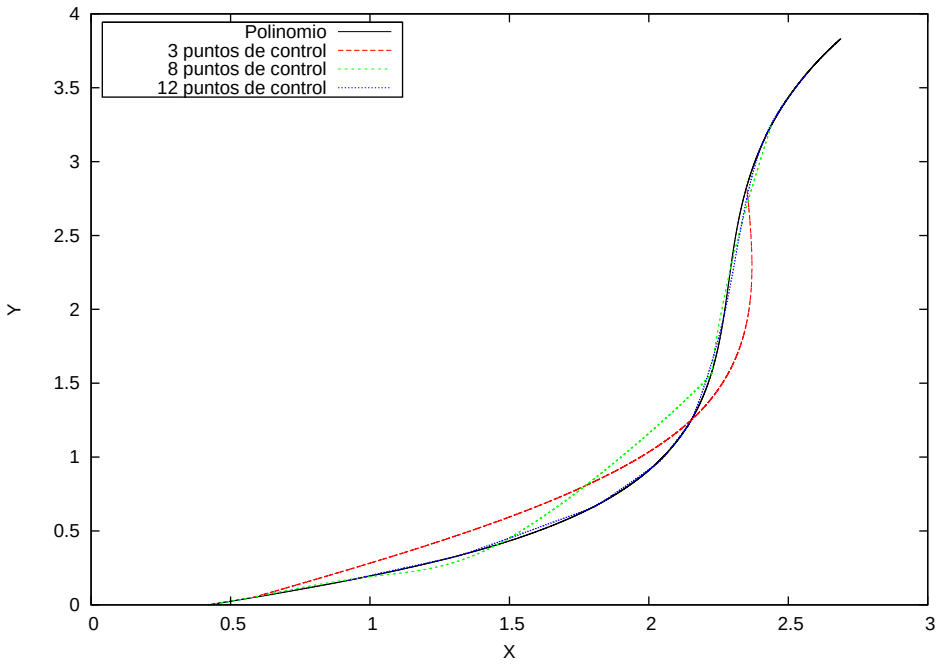

Figura 11: Aproximación utilizando parametrización centrípeta

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

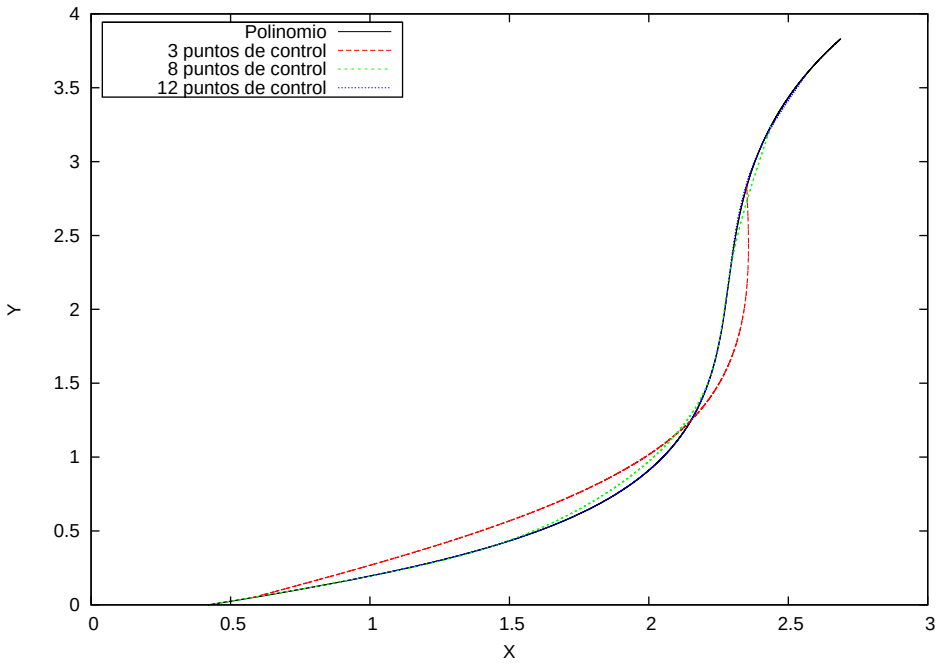

Figura 12: Aproximación utilizando parametrización por longitud de cuerda

4. Discusión

En esta sección, buscaremos conclusiones a la información suministrada por los gráficos de la sección anterior. Dividiremos la sección en dos subsecciones analizando en cada una de ella, cada una de las mismas en los resultados.

4.1. Ajuste de parámetros

Ambos gráficos están relacionados entre sí, por lo tanto, no analizaremos cada uno por separado.

Podemos notar en primera instancia, que utilizar como cantidad de iteraciones permitidas mayores a 1000 en el algoritmo, no tiene sentido, ya que las soluciones no mejoran, es decir el error cometido al buscar la raíz no disminuye. Si bien no estamos contemplando todos los casos posibles, ya que existen infinitos polinomios, por ende infinitas soluciones, podemos reforzar esta conclusión viendo que en el segundo gráfico (Figura 6) sin importar el costo (donde costo es una medida proporcional a la cantidad de iteraciones) la calidad de la solución no mejora en la casi totalidad de los casos de $1{,}0e-17$, es decir no se aproxima más al cero ideal. Estos nos lleva a inducir que un parámetro para la cantidad de iteraciones máxima de valor 1000 es suficiente para que el algoritmo aproxime lo más posible a la raíz.

Por otro lado, el primer gráfico nos muestra que las soluciones no mejoran al error de $1{,}0e-17$. Al ser pocos polinomios los analizados, necesitamos más información para poder inferir una cota para la *tolerancia*. Recolectando información sobre el segundo gráfico, vemos que salvo escasos casos atípicos, el resto de las muestras, no mejoran la misma cota impuesta para el gráfico anterior.

Nuevamente no podemos afirmar que este comportamiento va a ser igual para cualquier instancia presentada. A pesar que el generador de polinomios al azar intenta ser lo mas uniforme posible, puede quizás existir un sesgo que no estamos tomando en cuenta y por lo tanto las soluciones no se pueden mejorar. Otra causa puede ser que el error intrínseco de las operaciones con punto flotante no nos dejen mejorar la solución.

Es arbitraria la decisión tomada, pero considerando el uso que le damos y viendo que en nuestras muestras el error no se hace más chicos que la cota previamente mencionada, tomamos como *tolerancia* al valor de $1{,}0e-17$.

4.2. Análisis de curvas

Los gráficos referenciados por (Figura:7 y Figura:8) 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 (basándonos en estas instancias) tiende a unir los puntos con lineas más rectas cuanto más separados estos se encuentran. Es decir si existe cambios abruptos en la dirección donde la curva debe moverse y además en el lugar donde se produce este cambio, existe varios puntos cercanos entre sí, se generan oscilaciones importantes.

Por lo que podemos ver, la parametrización uniforme no modela bien las curvas en los casos donde entre un par de puntos de control la curva original no los une mediante algo que se asemeja a una recta.

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

La parametrización centrípeta si bien es mejor que la uniforme (en estos casos), 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:9) 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 (centrípeta 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 mientas que el resto de las parametrizaciones (centrípeta y longitud de cuerda) lo son en menor medida, siendo longitud de cuerda la menos dependiente de este suceso.

En los gráficos (Figura:10, 11y 12) podemos ver que a medida que la cantidad de puntos de control aumenta, la aproximación se acerca cada vez

más a la curva real. De todas formas, la velocidad de acercamiento varía dependiendo de la parametrización, pudiendo notar (basándonos 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 centrípeta y por último la parametrización por longitud de cuerda.

En el caso de la parametrización por longitud de cuerda, observamos que dada la presición 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 precision, no sería necesario aumentar la cantidad de puntos de control.

Nota: Es importante destacar que existen infinitos polinomios en $\mathbb{R}^n$ (en particular en $\mathbb{R}^n$), por lo tanto no podemos considerar como generales las conclusiones tomadas en este análisis, ya que pueden existir casos donde la mejor aproximación a la curva no es utilizando la parametrización por longitud de cuerda. Tanto los gráficos como el análisis nos da una idea general de como estas se comportan.

5. Preguntas

A continuación pasamos a responder algunas de las preguntas enunciadas en el trabajo práctico.

- ¿Depende la forma de la curva de la elección de la parametrización?

  Como vimos en los gráficos de la sección resultados, la elección de la parametrización está completamente ligada a la forma de la curva. Como ejemplo de esto podemos hacer referencia a los gráficos 7, 8 y 9. Si bien todas las parametrizaciones comparten ciertos puntos (por ejemplo los mismos puntos de control), la forma en que aproximan a la curva es distinta.

- ¿Cambia la forma de la curva si en lugar de deformar la curva conservando la parametrización el programa la recalcula al mover el punto?

  Para poder responder esta pregunta deformamos la curva manteniendo la parametrización y recalculándola (el código implementa la primer opción). Graficamos los resultados, donde en cada uno de ellos se utilizó sólo una parametrización. La curva original es igual para los tres gráficos. Se presentan los mismos en el siguiente orden, utilizando la parametrización uniforme, la centrípeta y por último por longitud de cuerda.

  En las figuras visualizamos en color *rojo* la curva original y los puntos de control. En *negro* el punto seleccionado y en *rosa* el punto en la curva más proximo a este. Vemos también en *azul* la curva modificada manteniendo la parametrización y en *verde* la curva deformada recalculándola.

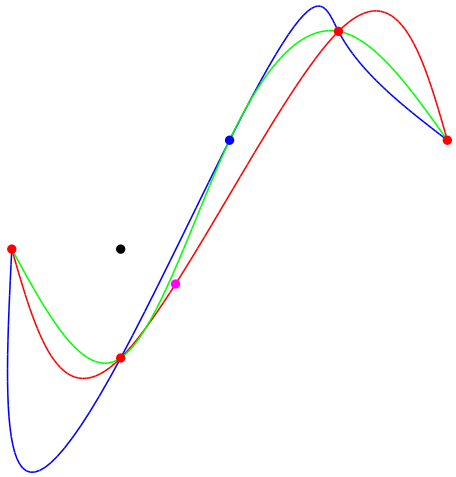

Figura 13: Cambio de parametrización (Uniforme)

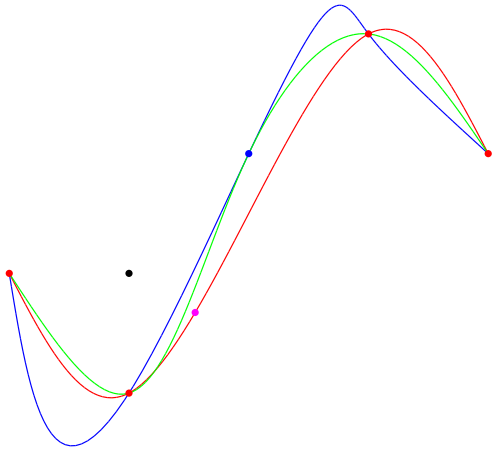

Figura 14: Cambio de parametrización (Centrípeta)

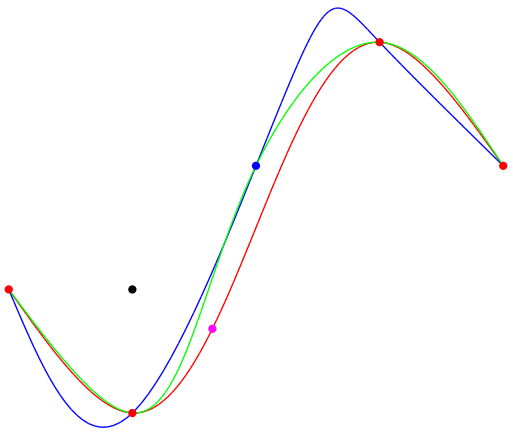

Figura 15: Cambio de parametrización (Longitud de cuerda)

Efectivamente la forma de la curva cambia al recalcular la parametrización respecto a no hacerlo.

Las curvas donde se recalcula la parametrización (curva *verde*) en las *tres* figuras (usando cualquiera de las parametrizaciones) tienen una forma más similar a la curva original en los puntos alejados al punto movido en contraste con la curva que mantiene la parametrización (curva *azul*).

6. Conclusiones

La utilización de splines para aproximación de curvas resulta ser muy efectiva ya que posee una implementación sencilla, y a su vez posee tiempo de procesamiento aceptables, consiguiendo buenas aproximaciones necesitando poca información sobre la curva. La distribución uniforme o semi uniforme de los puntos de control mejora la aproximación significativamente.

La selección de la parametrización a utilizar depende de la curva a aproximar en particular, por lo tanto para poder elegir la más apta, es necesario tener alguna información extra sobre la misma, la cual pueda ser relacionada con esta elección.

El método heurístico utilizado para conseguir el punto en la curva más cercano al *clickeado* resultó ser altamente efectivo en los polinomios de grado cinco (5) utilizados. La relación costo-presición, satisfizo nuestras necesidades.

La dificultad más grande que tuvimos fue en el análisis de los datos obtenidos y en la realización de las pruebas. Mencionándolo una vez más, existen infinitos polinomios, y no es posible englobar sus comportamientos en una familia finita de casos. Todas nuestras conclusiones se basan en instancias elegidas al azar, suponiendo que el generador no condiciona los resultados obtenidos (son pseudo-aleatoreos con distribución uniforme). Hemos de resaltar que nuestras conclusiones posiblemente no apliquen para todos los casos.

La elección de los parámetros tuvo el mismo problema, fueron elegidos basados en pocas instancias. La realización de gráficos entendibles y a la vez que aporten suficiente información es complicada ya que si por ejemplo utilizaramos cientos o miles de curvas, no podríamos entender cuál es el comportamiento. Pero a su vez si la entrada es poca y ahora sí entendible, al ser un problema tan grande, como por ejemplo acotar la cantidad de iteraciones necesarias para conseguir una buena precisión en la búsqueda de raíces para polinomios de grado cinco (5), las conclusiones y por ende las desiciones tomadas, son poco generales.

Con respecto a la implementación, la idea original consistía en implementar todo de la forma más genérica posible para luego por ejemplo, poder utilizar *splines* de grado distinto a tres (3), y probar si había diferencias si existían diferencias en la forma de la curva según la condición de borde usada (natural, sujeto, etc). Como primero debíamos implementar lo básico requerido, y dado que mantener la generalidad complicaba la implementación

se tomaron decisiones que hicieron imposible realizar pruebas como estas.

Nos hubiera gustado poder implementar una interfaz gráfica para la visualización de la curva con posibilidad de moverla arrastrándola con el *mouse*. Lamentablemente por falta de tiempo esto no fue posible, pero de así haber sido, nos hubiera ayudado a responder la pregunta opcional número cuatro (4), ya que hubieramos tenido que pensar la mejor forma de realizar todos los cálculos.

7. 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 parametrization_mode` (donde el archivo input cumple las condiciones del enunciado).

El tercer parámetro corresponde a los distintos tipos de parametrizaciones, acepta tres valores posibles, *u* que corresponde a parametrización uniforme, *cl* que corresponde a *chord length* y *cp* que corresponde a centrípeta. En caso de equivocarse con este parámetro e ingresar otro tomará por defecto la parametrización uniforme.

8. Apéndices

8.1. 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[2] 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 descripto 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;

[2]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[3].

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 contínuamente 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:

[3]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.

9. Referencias

Burden y J.D.Faires, Análisis numérico, International Thomson Editors, 1998 http://en.cppreference.com/w/cpp