Proyecto 2.
CO3211.
Luis Diaz: 15-10420;
Sept-Dic 2018.

Introducción:

Para el presente proyecto se requirió un programa que fuera capaz de aproximar la longitud de la línea de costa de la península de Paraguaná. Se solicitó que esto se hiciera mediante la aproximación de una curva a la forma del mapa, y luego calcular la longitud de dicha curva. Esta se debía calcular utilizando spline cúbico, y tomando puntos de un mapa real de la región. El procedimiento fue el siguiente:

1) Escoger una serie de puntos en el mapa.

2) Conseguir una curva que se aproxime apropiadamente a la forma de la región.

3) Calcular la longitud de la curva.

A continuación se explicará como se hizo cada una de estas cosas.

Puntos en el mapa:

Para empezar, se escogieron varios juegos de puntos en lugar de hacer un solo conjunto de puntos que delimitaran el mapa completo por sí solos. Esto es, cada conjunto de puntos representaba un segmento de la curva que estamos buscando. En total se escogieron 14 conjuntos de puntos, y estos son los criterios que se utilizaron para escogerlos:

1) Se intentó escoger los puntos de tal forma que siempre hubiese un punto en un valle o cresta de la figura, y otros dos a cada lado de la concavidad de este. Esto se hizo para conservar información relevante sobre las pendientes correspondientes a la primera y segunda derivada.

2) Para cada punto P_i del conjunto de k puntos “ $paraguana_j$ ” se cumple que P_i aparece antes en la figura de la región que P_{i+1} . Esto es, para x_i de P_i , se cumple que $x_i<x_{i+1}$ para $1 \leq i<k$ (En otras palabras, se buscó evitar el “*zig-zag*” incoherente en la gráfica del segmento).

Cada uno de estos conjuntos de puntos se usó para crear un spline diferente. El segundo criterio tiene la finalidad de poder encontrar una función que se adapte al conjunto de puntos, pues si este criterio no se cumple, se tendrá que una preimagen x podría tener varias imágenes (lo cual no tendría sentido y provocaría incoherencias en la función expresada en el programa) y además también es necesario para mantener la figura correcta de la zona geográfica escoger siempre puntos contiguos. Todos los spline juntos forman la figura requerida.

Encontrar la curva:

Para encontrar la se utilizó una función que calcula el spline cúbico. En particular, se usó un spline cúbico libre para no tener que buscar la pendiente en los extremos de cada segmento de costa. Dado que se utilizaron 14 segmentos para representar la costa completa, serían necesarias 28 pendientes. Así, una solución más simple y apropiada para el problema fue utilizar un spline libre.

La función en cuestión está en el archivo “splineCubico.m” y fue implementada siguiendo el procedimiento explicado en las láminas del curso. Se creó un spline por cada conjunto de puntos, de tal manera que cada spline definiera un segmento diferente de la línea de costa. Para evaluar el spline se usó la función de “HornerSpline.m”. La función splineCubico es capaz de crear dos tipos de spline, amarrado o libre dependiendo de la cantidad de argumentos. Si, además del conjunto de preimágenes e imágenes, se le pasa como argumento la derivada a ambos extremos el intervalo de preimágenes, entonces la función

devuelve un spline amarrado construido con esos parámetros. Cualquier otra cantidad de argumentos devuelve un error.

Longitud de curva:

Para calcular la longitud de curva se utilizaron dos funciones: splineLength.m y LongitudArco.m. La primera es una función auxiliar que calcula la longitud de curva de un spline en su correspondiente intervalo de convergencia (el intervalo donde están contenidas sus preimágenes) mediante la segunda función. Así, con la splineLength se calculan todas las longitudes de los spline que ussamos y luego se suman todas las longitudes. Internamente, splineLength usa LongitudArco para calcular la longitud de cada polinomio que forma el spline en su correspondiente intervalo.

Por su parte, LongitudArco usa la siguiente fórmula para calcular la longitud de un polinomio de grado 3 en el intervalo [a,b]:

$$L=\int_a^b \sqrt{1+\left(P_3{}'(x)\right)^2}\,dx$$

Además, dado que la documentación de matlab sugería usar la función “integral( )” en lugar de usar “quad( )”, se usó la primera.

Finalmente, usando la escala de 10 kilómetros por 66 pixeles, se hizo la regla de 3 para obtener el resultado real. El resultado que se obtuvo fue: 253.09km.

Descripción de las funciones:

Las funciones utilizadas en el proyecto son las siguientes:

-HornerSpline.m
-LongitudArco.m
-splineCubico.m

-Horner.m
-plotHelper.m
-splineLength.m
-ContornoCosta.m
-LongitudLineaCosta

HornerSpline: Dado un Spline S, un vector x, y un vector de preimágenes "Xprev" utilizadas para construir S, esta función devuelve un vector "y" tal que $y_i = S(x_i)$. Esto es, la evaluación de x_i en el spline S .

Para esto, primero se busca linealmente el intervalo $[Xprev_j, Xprev_{j+1}]$ al que pertenece x_i , y una ves encontrado se evalua el polinomio S_j en el valor x_i mediante la función Horner que hemos utilizado en laboratorios anteriores para evaluar polinomios. La llamada de la función Horner se hace sobre el valor $(x_i - Xprev_j)$.

LongitudArco: Dado un coeficiente de constantes que representan los coeficientes de un polinomio S_j de grado 3 de un spline S , el valor inicial "init", y el final "final" que definen el intervalo $[init, final]$ para el que el polinomio S_j está definido en S , esta función devuelve un valor real " L " que representa la longitud de la línea de curva del polinomio S_j en el intervalo $[init, final]$. El cálculo de esta longitud se hace mediante la ecuación:

$$L = \int_a^b \sqrt{1 + (P_3'(x))^2}\, dx$$

splineCubico: Dados dos vectores X, Y tales que existe f tal que $Y_i = f(X_i)$, $[X_0, X_n]$ contienen a todos los valores X_i con $0 \le i \le n$ y $X_i < X_{i+1}$; también posiblemente los valores a,b tales que $a = f'(X_0)$, $b = f'(X_n)$, la función splineCubico devuelve una matriz S_{nx4} que define al spline que mejor aproxima a la función f . Si se pasan los valores a,b, el spline será de tipo amarrado, y de lo contrario será un spline libre.

El procedimiento seguido para calcular este spline es el mismo que el de las láminas del curso. Sea S_j un polinomio del spline representado por S tal que $S_j = a + bx + cx^2 + dx^3$, se cumple que $a = S(j,1), b = S(j,2) c = S(j,3), d = S(j,4)$.

Horner: Dado un polinomio P, un vector x, devuelve un vector y tal que $y_i = P(x_i)$. El método para evaluar $P(x_i)$ es el método de Horner.

plotHelper: Dado un spline S, y el conjunto de preimágenes X utilizadas en la construcción de S, plotHelper devuelve dos vectores x,y tales que $x = X_1 : 0.1 : X_n$ (en la notación de matlab) y $y_i = S(x_i)$. Esta función es utilizada para graficar, pues los vectores x,y son las preimágenes e imágenes que aparecen graficadas en el lienzo.

SplineLength: Dado un spline S y el conjunto X de preimágenes utilizadas en la construcción de S, esta función devuelve un float l que es la longitud de arco del Spline S. Para calcular l, este programa hace llamadas suscesivas a LongitudArco sobre cada uno de los polinomios del spline S. Esta es una función auxiliar para calcular la suma de las longitudes de todos los polinomios de un spline.

ContornoCosta: Dado un valor booleano b, dibuja una aproximación al mapa solicitado de la península de Paraguaná. Si $b \equiv 1$, entonces muestra el dibujo de la aproximación con los puntos originales también en el lienzo; de lo contrario, muestra solo la aproximación.

Esta función primero crea los 14 spline correspondiente a las 14 imágenes .bmp donde están seleccionados los puntos y luego grafica todos los spline con ayuda de plotHelper. Los datos de los puntos (x,y) sobre el mapa son tomados del archivo data.m. La finalidad de esta decisión es reducir el tiempo de corrida del algoritmo final, pues la captura de puntos toma mucho tiempo. En cualquier caso, en el archivo test.m se encuentra el código que capturó los

puntos, una selección distinta de puntos solo amerita volver a ejecutar ese archivo.

LongitudLineaCosta: Devuelve la longitud “L” de la línea de la costa de la península de paraguaná. Para calcularla primero construye los 14 splines que se usaron para aproximar su línea de costa y luego, con ayuda de splineLength, suma las longitudes de todos estos splines. Esto es:

$$L=\sum_{i=1}^{14} splineLength(S^i, X^i)$$

Con S^i el i-ésimo spline, y X^i el vector de preimágenes usado para calcular S^i .

Resultados:

La longitud de costa hayada fue de 253.09km. Este valor puede cambiar dependiendo de la selección de puntos en los mapas, así que esta es una aproximación.

Archivos:

Código:

-proyecto2.m
-captura_puntos.m
-ContornoCosta.m
-Horner.m
-HornerSpline.m
-LongitudArco.m
-LongitudLineaCosta.m
-plotHelper.m
-splineCubico.m
-splineLength.m

-test.m

Resultados:

-informe.pdf
-data.mat
-paraguana_origin.bmp
-Aproximacion.ofig
-AproximacionVsPuntos.ofig
-paraguana_origin.bmp
-paraguana1.bmp
-paraguana2.bmp
-paraguana3.bmp
-paraguana4.bmp
-paraguana5.bmp
-paraguana6.bmp
-paraguana7.bmp
-paraguana8.bmp
-paraguana9.bmp
-paraguana10.bmp
-paraguana11.bmp
-paraguana12.bmp
-paraguana13.bmp
-paraguana14.bmp