**אינטרפולצית Spline**

 המוסג אינטרפולצית Spline או אינטרפולצית הקיסם הינו בעצם הכללה של האינטרפולציה הלינארית. ב- Spline אנחנו מחשבים את הקירוב בנקודות הביניים ע"י פולינום ולאו דווקא על ידי קו ישר, אבל בניגוד לאינטרפולציה פולינומיאלית שבו פולינום **יחיד** מחשב את **כל** ערכי הקירוב, . ב- Spline הערכים **בכל קטע** נחשבים ע"י פולינום **אחר**. מטבע הדברים ה-Spline יוצר קירוב פחות מאולץ מהאינטרפולציה הפולינומיאלית ובדרך כלל יותר מדויקת. לעיתים spline משמשת מימוש קירוב בדרגת פולינום נמוכה גם בהינתן נקודות רבות, שכן הודות להסרת האילוץ של פולינום יחיד, דרגת הפולינום לא חייב לעלות עם מספר הנקודות.

המילה spline קיסם מרמז שזוהי טכניקה המדמה טכניקה של בעלי מלאכה שמציירים / ממשים עקומות חלקות ע"י קביעת קיסמים.

לפיכך בהינתן n+1 נקודות {(xi, f(xi)} i=0,1,…n פונקצית spline הינה

 ***S0(x) x ε [x0,x1]***

 ***S1(x) x ε [x1,x2]***

 ***.***

***S(x) = .***

 ***.***

 ***Sn-1(x) x ε [xn-1,xn]***

 המאפיינים של פונקצית ה-spline נקבעים ע"י הנקודות הנתונות **והדרגה של הפולינום שנבחר** ל-spline.

 פונקציות spline מדרגה n המחשבות ערכים בקטעים בעלי קצה משותף xk, יזדהו בנקודת הקצה הזו בערך הפונקציה f(xk) ובערך n-1 הנגזרות הראשונות f(i)(xk),i=1, …,n-1 .

למעט spline לינארי המשוואות הללו לא מספיקות לקבוע חד ערכית את המקדמים ויש דרגות חופש, שמנוצלות ליישום תכונות אופי מסוימות על הפתרון.

**Spline לינארי**

Spline לינארי מזדהה עם האינטרפולציה לינארית, שלמען האמת מבחינת ההגדרות הוא spline ולא אינטרפולציה פולינומיאלית.

**Spline ריבועי**

 אם המטרה היא לקרב את הפונקציה בעזרת n פולינומים מדרגה 2, ישלנו איפוא 3n נעלמים. המשוואות

***Si(xi) = f(xi) i=0,1, …,n משוואות n+1***

***Si-1(xi) = Si(xi) i=1,2, …n-1 משוואות n-1***

נותנים 2n תנאים.

נוסף לכך יש את התנאים על הנגזרות בנקודות המשותפות

S'i-1(xi) = S'i(xi) i=1,2, …n-1 משוואות n-1

סה"כ יש 3n-1 תנאים, יש לפיכך דרגת חופש אחת. אם ידוע ערך הנגזרת באחת הקצוות, למשל

***S'0(x0) = z0***

זה ישלים את התמונה.

גראפית זה נראה כך:



נניח שזהו המצב,

הפתרון יראה כך:

במקרה הזה נוח לייצג את הפולינומים של ה-spline בצורה

***Si(x) = ai(x-xi)2 + bi(x-xi) + f(xi)***

***i = 0, …,n-1***

בפועל אנחנו נקבל שבו נוח לבטא את ai ו- biבמושגים של וקטור zi

***bi = zi  (I)***

***ai = (zi+1 – zi)/(2(xi+1 –xi)) (II)***

כאשר

***zi+1 = -zi  + (f(xi+1) –f(xi))/(2(xi+1 –xi)) (III)***

כלומר שהנוסחה של Si(x) היא בעצם

***Si(x) = (zi+1 – zi)/(2(xi+1 –xi)) (x-xi)2 + zi(x-xi) + f(xi)***

**הוכחה:**

נסמן

***bi = zi***

עבור S0(x):

***S'0(x0) = 2 ai(x0-x0) + b0 = z0***

לפיכך

***b0 = z0***

מהמשוואה

***S'0 (x1) = S'1(x1)***

נובע

***2 a0(x1-x0) + b0 = b1***

לפיכך

***2 a0(x1-x0) + z0 = z1***

***a0 = (z1 – z0)/(2(x1-x0))***

שמתאים לנוסחה.

לפיכך אנחנו נמצאים במצב שאפשר להוכיח את הנוסחאות לעיל באינדוקציה.

למשל עבור i= 1,

מתקיים

***S0(x1) = S1(x1)***

כלומר

***a0(x1-x0)2 + b0(x1-x0) + f(x0) = f(x1)***

***a0(x1-x0)2 = f(x1) - f(x0) – z0(x1-x0)***

***a0 = (f(x1) - f(x0))/ (x1-x0)2 – z0/(x1-x0)***

אבל מצד שני

***a0 = (z1 – z0)/(2(x1-x0))***

לפיכך

***(z1 – z0)/(2(x1-x0)) = (f(x1) - f(x0))/ (x1-x0)2 – z0/(x1-x0)***

לפיכך

***z1 – z0 = 2(f(x1) - f(x0))/ (x1-x0) – 2z0***

***z1 = – z0 = 2(f(x1) - f(x0))/ (x1-x0)***

זה מוכיח ש-z1 מקים את נוסחה (III). למעשה האלגברה הזו משלימה הוכחה באינדוקציה.

הוכח שנוסחאות (I) ו-(II) נכונות עבור i=0. נניח נכונות עבור i. עבור i+1 מתקיים:

מההגדרה

***bi = zi***

מהמשוואה

***S'i(xi+1) = S'i+1(xi+1)***

מתקיים

***2 ai(xi+1-xi) + bi = bi+1***

לפיכך

***ai = (zi+1 – zi)/(2(xi+1-xi))***

מצד שני

**ai(xi+1-xi)2 + bi(xi+1-xi) + f(xi) = f(xi+1)**

**ai = (f(xi+1) - f(xi))/ (xi+1-xi)2 – zi/(xi+1-xi)**

**(zi+1 – zi)/(2(xi+1-xi)) = (f(xi+1) - f(xi))/ (xi+i-xi)2 – zi/(xi+1-xi)**

**zi+1 = -zi  + 2(f(xi+1) –f(xi))/(xi+1 –xi)**

נשאר רק להוכיח ש

***ai+1 = (zi+2 – zi+1)/(2(xi+2-xi+1))***

אנחנו יכולים להניח את נוסחה (III) עבור n-1 i+1<

***zi+2 – zi+1 = -2zi+1  + 2(f(xi+2) –f(xi+1))/(xi+2 –xi+1)***

***(zi+2 – zi+1) )/(2(xi+2-xi+1)) =***

 ***-zi+1/(xi+2-xi+1) + (f(xi+2) –f(xi+1))/(xi+2 –xi+1)2***

מצד שני i+1 < n-1 לכן:

***ai+1(xi+2-xi+1)2 + bi+1(xi+2-xi+1) + f(xi+1) = f(xi+2)***

לפיכך

***ai+1 =***

 ***-bi+1/(xi+2-xi+1) + (f(xi+2) –f(xi+1))/(xi+2 –xi+1)2***

 ***=***

 ***-zi+1/(xi+2-xi+1) + (f(xi+2) –f(xi+1))/(xi+2 –xi+1)2***

מ.ש.ל

**קוד C**

typedef struct point

{

 long double x;

 long double fx;

} POINT, \*POINT\_PTR;

typedef struct si\_rec

{

 int n; /\* no\_of\_points \*/

 POINT\_PTR point\_arr;

 long double \*z;

} SI\_REC, \*SI\_REC\_PTR;

void compute\_z(SI\_REC\_PTR si\_r, long double z0)

{

 long double zi, t;

 int i;

 si\_r->z[0] = z0;

 zi = z0;

 for(i=1; i <= si\_r->n; i++)

 {

 t = 2\*(si\_r->point\_arr[i].fx - si\_r->point\_arr[i-1].fx);

 t = t/(si\_r->point\_arr[i].x -si\_r->point\_arr[i-1].x);

 zi = -zi + t;

 si\_r->z[i] = zi;

 }

} /\* compute\_z \*/

long double interpolate(SI\_REC\_PTR si\_r, long double x)

{

 int i, j, n;

 long double t1, t2, a, b, c;

 n = si\_r->n;

 for(i=0; i < n; i++)

 if (x >= si\_r->point\_arr[i].x && x < si\_r->point\_arr[i+1].x)

 break;

 b = si\_r->z[i];

 c = si\_r->point\_arr[i].fx;

 a = si\_r->z[i+1] - si\_r->z[i];

 a = a/(2\*(si\_r->point\_arr[i+1].x - si\_r->point\_arr[i].x));

 t1 = x - si\_r->point\_arr[i].x;

 t2 = c + b\*t1 + a\*t1\*t1;

 return t2;

} /\* interpolate \*/