**קירוב פונקציות על סמך נקודות נתונות**

**התאמת עקומים Curve fitting**

**התאמת עקומים** הינה סידרה של שימושים שונים של הרעיון של קירוב פונקציות כאשר מה שנתון עליה היא סידרה של נקודות ערכים וערך הפונקציה בערכים הללו

***{(xi,f(xi))} i = 1, …,n, x1 < x2 < … < xn***

ישנם שלוש שימושים או גישות בסיסיות לרעיון:

* **אינטרפולציה** **interpolation** שבו הקירוב עובר **בדיוק** בנקודות הנתונות והקירוב **מוגדר רק עבור x בתחום x1 < x < xn**.
* **אקסטרפולציה** **extrapolation** שבו הקירוב עובר **בדיוק** בנקודות הנתונות אבל המטרה היא לתת הערכה לערכי הפונקציה **מעבר** לתחום x1 < x < xn.
* **רגרסיה regression**  ניסיון למצוא פונקציה המקרבת הכי טוב שאפשר מדידות רבות שאינן מדויקות.

**אינטרפולציה interpolation**

בגישה של **אינטרפולציה** הרעיון הוא איך לקרב טוב ככל האפשר את ערכי הפונקציה בערכים x **שבין** הנקודות. באינטרפולציה, טווח פונקצית הקירוב היא xn > x0 < x. ההנחה כאן היא שהנקודות הללו הן מעין מדידות **מדויקות** כלומר רמת הדיוק של כל (xi,f(xi)) היא גבוהה והקירוב צריך לעבור דרך הנקודות הללו. מבין שלושת הגישות המצוינות לעיל, לגישה הזו יש את התנאים הטובים ביותר לספק נתונים בדיוק גבוה: היא מקבלת את ההנחה שנתונים מדויקים כמו באקסטרפולציה ולא מצפים ממנה לחזות ערכי פונקציה מחוץ לטווח הנקודות כמו באקסטרפולציה ורגרסיה.

 הצורך באינטרפולציה יכול להתעורר בכל מיני הקשרים מדעיים או הנדסיים, אפילו במדעי החברה. הנקודות הנתונות יכולות להיות תוצאות של מדידות פיסיקליות יחסית לזמן, לחץ או טמפרטורה. למשל התחממות מנוע יחסית למאמץ או זמן. הם יכולים להיות נתונים של סקרים שנעשו בהפרש זמנים כזה או אחר כמו גודל אוכלוסיה בשנתונים.

שימוש אחד של אינטרפולציה היא קירוב פונקציות יקרות לחישוב. אם נניח נוסחת החישוב של פונקציה בדיוק רצוי, ע"י, נניח, נוסחת טיילור היא יקרה מאד, ניתן להוזיל את העלות על ידי חישוב מספר מסוים של נקודות, לשמור אותם ולקרב הפונקציה בערכי הביניים ע"י אינטרפולציה. שיטה זו נקראית

**דיסקרטיזציה** ורבים מפונקציות הספרייה (כמו sin, cos, exp) של מעבדים או קומפילרים מיושמים בצורה הזו.

**שיטות אינטרפולציה בסיסיות**

**אינטרפולציה לינארית**

 השיטה הפשוטה ביותר ואחת הנפוצות ביותר בשימוש היא האינטרפולציה הלינארית. בהנתן נקודת בינייים x מחשבים את f(x) כערך על הקו הישר המחבר בין שני ערכי הנקודות הסמוכות כלומר אם x < xi+1 > xi אזי

***f(x) = f(xi) + ((f(xi+1) – f(xi))/(xi+1 – xi)))(x – xi)***

גרפית זה נראה כך:

אם אלו הן הנקודות



אזי הקירוב יהיה:



אינטרפולציה לינארית היא פשוטה וכללית במובן הזה שהיא נותנת פתרון זהה באיכותו לכל פונקצית מקור יהיו מאפייניו אשר יהיו. הקירוב יהיה מדויק אם פונקצית המקור הוא בפועל קו ישר. אולם במקרים אחרים הקירוב אינו נותן הערכות בדיוק גבוה לפונקצית המקור אלא אם הנקודות הנתונות צפופות ביותר. בעיה אחרת היא שפונקצית הקירוב אינו גזיר גם אם פונקצית המקור כן, וזה יכול להפריע בישום שיטות שונות על הקירוב, כמו שיטת ניוטון.

**קוד C**

typedef struct point

{

 long double x;

 long double fx;

} POINT, \*POINT\_PTR;

typedef struct li\_rec

{

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

 POINT\_PTR point\_arr;

} LI\_REC, \*LI\_REC\_PTR;

int bin\_search(LI\_REC\_PTR li\_r, long double x)

{

 int low, high, mid;

 low = 0;

 high = li\_r->n-1;

while(1)

{

 mid = (low+high)/2;

 if (li\_r->point\_arr[mid].x == x)

 return mid;

 if (li\_r->point\_arr[mid+1].x == x)

 return mid+1;

 if ( (li\_r->point\_arr[mid].x < x) &&

 (li\_r->point\_arr[mid+1].x > x))

 return mid;

 if( li\_r->point\_arr[mid+1].x < x )

 low = mid+1;

 else /\* li\_r->point\_arr[mid].x > x \*/

 high = mid-1;

} /\* while \*/

} /\* bin\_search \*/

long double interpolate(LI\_REC\_PTR li\_rec, long double x)

{

 int i;

 long double t;

 i = bin\_search(li\_rec, x);

 t = (li\_rec->point\_arr[i+1].fx - li\_rec->point\_arr[i].fx);

 t = t/(li\_rec->point\_arr[i+1].x - li\_rec->point\_arr[i].x);

 t = t\*(x - li\_rec->point\_arr[i].x);

 t = li\_rec->point\_arr[i].fx + t;

 return t;

} /\* interpolate \*/

**אינטרפולציה פולינומיאלית**

נסיון למצוא עקומת אינטרפולציה מתחכמת יותר הוא להעביר פולינום דרך הנקודות הנתונות. בהינתן n נקודות {(xi,f(xi))} i=1,...,n ישנו פולינום אחד ויחיד מדרגה n-1 שעובר דרך הנקודות הללו. אילו היו שניים כאלו Pn(x) ו-Qn(x) אזי היתה לנו מצב שלפולינום ההפרש Rn(x) = Pn(x) - Qn(x) היו n שורשים וזה לפי משפט אפשרי רק אם Rn(x) הוא פולינום האפס.

 בשלב זה אנחנו מדברים על פולינום **יחיד** שיממש את **כל** הקירוב על פני **כל** טווח ההגדרה. זו לא הגישה היחידה ועוד נראה זאת.

באופן גרפי זה נראה ככה:



כך נראה הפונקציה y=tan(x) ופולינום האינטרפולציה שלו מדרגה 4:



העקום שגם יורד קצת ושיש לו מינימום בערך ב-0.5 הוא פולינום האינטרפולציה. הקו השני פונקצית ה-tan(x).

**אינטואציה**: אינטרפולציה פולינומיאלית מקרבת את פונקצית המקור ע"י עקום רציף, חלק ואפילו גזיר. אם פונקצית המקור היתה במקרה או שלא במקרה פולינום מדרגה n-1 או פחות, הקירוב הזה יהיה **מדויק**, כלומר חישוב נקודת ביניים x ע"י אינטרפולציה תיתן לנו **בדיוק** את f(x). מעבר לכך, עבור פונקציות f(x) גזירות אינסוף פעמים, לפי נוסחת טיילור/מקלורן הם למעשה שווים לפולינומים אינסופיים. אם בטווח מסוים n המקדמים הראשונים הם **דומיננטיים**, כפי שזה קורה למשל באינטרוולים קטנים, אזי הקירוב המתקבל מאינטרפולציה פולינומיאלית תהיה קירוב טוב למדי. יחד עם זאת, קיימות דוגמאות של פונקציות אלמנטריות, כמו פונקציה רונגה runge שהיא

***f(x) = 1/(1+25x2),***

 שעל פני אינטרוול לא קטן מספיק הקירוב יהיה גרוע גם כאשר n 🡪 ∞.

**מימוש אינטרפולציה**

 כאשר אנחנו משתמשים באינטרפולציה פולינומיאלית המקדמים של הפלינום הם לא העיקר, אלא היכולת לחשב את ערכי הפולינום בנקודות רצויות. למעשה קבוצת המקדמים אינה יחידה – יש דרכים שונות ליצג מקדמים של פולינום לפי גישות שונות של חישוב ערך של פולינום.

 הגישה הסטנדרטית לייצוג פולינום הוא



את ערך הפולינום הזה ניתן לחשב ע"י כלל הורנר ב-n כפלים וסכומים בתנאי שיש לך את המקדמים. את המקדמים ניתן לקבל ע"י פתרון מערכת המשוואות.



הנקרא מטריצת ונדרמוד. חישוב נקודת אינטרפולציה אחת מנקודת ההתחלה לפי גישת ונדרמונד יחייב חישוב המטריצה, מהלך שלוקח n כפלים לחשב כל שורה, O(n2) פעולות בסך הכל, ופתרון המערכת תיקח עוד O(n3) פעולות. הסה"כ הפעולות יהיה O(n3). גישת ונדרמונד משתלמת רק אם מחשבים את המקדמים פעם אחת לחישוב מספר גדול של נקודות אינטרפולציה. ישנן גישות חסכוניות יותר.

הגישות הבאות יפתרו את הבעיה ב-O(n2).

**נוסחת אינטרפולציה של לגרנז**

 גישת לגרנז מחשבת ערך של הפולינום בנקודה מבלי למען האמת לחשב את המקדמים של הפולינום.

בהינתן סידרה {(xi,f(xi)} i=0, …,n וערך x שבו צריך לחשב את הקירוב,

***L(x) = Σ f(xi) li(x)***

כאשר ה- ***l***i(x) הם פולינומי לגרנז המוגדרים כ-

 ***n***

***li(x) = ∏ (x- xj)/(xi – xj)***

 ***j=0***

 ***j ≠ i***

פולינומי לגרנז נבנו כך שיהיה להם את התכונה

***li(xi) = 1***

***li(xj) = 0 i ≠ j***

לפיכך L(xi) = f(xi) כפי שנדרש באינטרפולציה. L(x) הוא סכום של n פולינומים מדרגה 1-n ולכן הוא בעצמו פולינום מדרגה n-1 ומאחר והוא מזדהה עם f(xi) בנקודות xi  הוא מזדהה לפולינום האינטרפולציה היחיד שיושג בכל גישה אחרת.

בהנתן x חישוב כל ***l***i(x) לוקח O(n) חישובים ומאחר שיש O(n) כאלו הסיבוכיות סה"כ היא O(n2). אם אנחנו ממשים איזה שהיא אינטרפולציה קבועה על מספר רב שהיה יותר יעיל להשתמש בגישה אחרת, למשל לחשב מקדמים.

**קוד C**

typedef struct point

{

 long double x;

 long double fx;

} POINT, \*POINT\_PTR;

typedef struct li\_rec

{

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

 POINT\_PTR point\_arr;

} LI\_REC, \*LI\_REC\_PTR;

long double interpolate(LI\_REC\_PTR li\_rec, long double x)

{

 int i, j, n;

 long double t1, t2;

 n = li\_rec->n;

 t1 = 0.0;

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

 {

 t2 = li\_rec->point\_arr[i].fx;

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

 if(i != j)

 {

 t2 = t2\*(x - li\_rec->point\_arr[j].x);

 t2 = t2/(li\_rec->point\_arr[i].x - li\_rec->point\_arr[j].x);

 } /\* if \*/

 t1 = t1 + t2;

 } /\* for \*/

 return t1;

} /\* interpolate \*/