**שיטות ישירות**

ישנם גם שיטות "ישירות" יותר, שאינן מבוססות על התאמת עקומות,

שאפשר להשתמש בהם רק בתנאי שהפונקציה ניתנת לחישוב בכל נקודה, ויעבדו גם כאשר הפונקציה אינה גזירה, או שהפונקציה אינה ידועה באופן אנליטי. אלו סכמות קטע כמו שיטת ההתאמה הריבועית, אך בניגוד אליו שיעור ההתכנסות היא דטרמיניסטית. הן אולי לא יתכנסו מהר במיוחד אבל גם לא לאט במיוחד במקרה הגרוע.

**שיטות פיבונצי (Fibonacci search) ויחס הזהב golden section search**

שיטות פיבונצי ויחס הזהב היא מעין גרסאות לאופטימיזציה של שיטת החצייה למציאת שורשים. אולם כאן הבעיה מורכבת מעט יותר, כי מציאת מינימום הוא עניין קצת יותר מורכב מאשר מציאת שורש. אם בשיטת החצייה היה לנו בעצם סכמה של 2 נקודות (קצוות הקטע של הפתרון) שאותו שיפרנו ע"י מציאת נקודה שלישית, כאן יש לנו סכמה של שלוש נקודות שאותו אנחנו משפרים ע"י חישוב נקודה רביעית. שתי הנקודות יהיו קצוות הקטע שבו המינימום צריך להימצא, והנקודה השלישית היא הערכה לנקודת המינימום. חישוב נקודה רביעית תאפשר לנו לבחור שלוש מתוך ארבע הנקודות שבו הקטע יותר קטן בפקטור מסוים.

אנחנו יוצאים מתוך הנחה שהפונקציה יורדת ממש מ-a עד שהיא מגיעה למינימום ומאותה רגע היא פונקציה עולה ממש. במושגים של פונקציות גזירות אנחנו בעצם מניחים שבקצה הראשון של הקטע הנגזרת שלילית, ושהיא פונקציה עולה ממש. בנקודת המינימום הנגזרת אפס בקצה השני הנגזרת חיובית. לפיכך אנחנו יכולים להניח שערכיו של הפונקציה f בקצוות הקטע גדולים מערכיו בכל נקודה פנימית.

שיטות פיבונצי ויחס הזהב הוא למעשה שיפור (במובן מסוים) של הרעיון היותר פשוט הבא, נקרא לו באופן לא רשמי שיטת השליש.

**שיטת השליש**:

בהינתן קטע (a,b) שבו ידוע שהמינימום של הפונקציה f(x) נמצא, נניח ש-f(a) ו-f(b) ידועים.

חשב

***x1= a+(b-a)/3***

***x2 = b-(b-a)/3***

חשב את f(x1) ו- .f(x2)

ישנם 2 תסריטים אפשריים, בהתאם ל-2 האפשרויות:

1. f(x1) > f(x2)
2. f(x1) < f(x2)

מקרה א):

| f(a)

| . . f(b) .

| . . .

| .f(x1) . .

| . . . . .

| . . . .

| . . f(x2) .

| . . .

\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.\_\_\_\_\_\_\_\_\_\_\_\_

a x1 x2 b

מקרה ב):

| f(a)

| . . f(b)

| . f(x2) .

| . . f(x1) . .

| . . . .

| . . .

| . . . .

| . .

\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a x1 x2 b

אם f(x1) > f(x2):

משמעות הדבר ש-f ממשיך לרדת בין x1 ל-x2. לפיכך המינימום יכול להיות בין x1 ל-x2 או בין x2 ל-b. מה שבטוח: הוא **לא** בין a ל-x1.

לפיכך אנחנו יכולים לקחת בתור קטע שבו נמצא המינימום את (x1,b).

אם f(x1) < f(x2):

המינימום יכול להיות בין a ל- x1 או בין x1 ל-x2. אבל הוא **לא** יכול להיות בין x2 ל-b. לפיכך אנחנו יכולים לקחת בתור קטע שבו נמצא המינימום את (a,x2).

אם f(x1) = f(x2):

המינימום הוא בין x1 ל-x2, ובודאי מוכל בכל אחד מהקטעים (x1,b) ו-(a,x2).

לפיכך יש לנו שיטה, שבהינתן קטע (a,b) שבו חייב להיות המינימום, אנחנו יכולים להקטין אותו לכדי 2/3 מאורכו ע"י חישוב הפונקציה ב-2 נקודות נוספות.

**קוד C**

long double quarter(long double (\*fp)(long double),

long double a, long double b, long double eps)

{

long double x1, fx1, x2, fx2;

long double factor = 4.0;

do {

x1 = a + (b-a)/factor;

fx1 = (\*fp)(x1);

x2 = b - (b-a)/factor;

fx2 = (\*fp)(x2);

if (fx1 > fx2)

a = x1;

else

b = x2;

} while ( (b - a) > eps);

return ( (a+b)/2);

} /\* quarter \*/

"שיטת השליש" היא שיטה פשוטה ואמינה אבל לא יעילה. היא מקטינה את הקטע בפקטור של 0.667 (מקטינה את גודלו ב-33%(. הסיבה שהיא לא שיטה רשמית מפני שהיא לא אופטימלית בצורה שהיא פוטרת את הבעיה: היא מחשבת את פונקצית המטרה f(x) **בשתי** נקודות לכל איטרציה, בעוד שקיימות דרכים להסתפק בחשוב **אחד** של פונקצית המטרה לכל איטרציה. כמו כן היא מקטינה את הקטע בפקטור 0.667 בעוד שקיימות שיטות להקטין אותו בפקטור של בערך 0.618 (הקטנת גודל הקטע ב-38%) לכל איטרציה , למעשה (√5-1)/2 זהו עוד מקרה של "יחס הזהב": הגבול של שני מספרים עוקבים של מספרי פיבונצי Fn-1/Fn limn כאשר n🡪∞ . בהנחה שחישוב של ערך פונקצית המטרה בנקודה הוא החלק היקר ביותר של החישוב, אלגוריתמים כאלו יהיו לפחות פי שניים יותר יעילים.

הדרך להסתפק בחישוב נקודה אחת לכל איטרציה הוא לחשב אותם כך שהנקודות יחשבו כך ששלוש מארבע הנקודות המשתתפות בכל איטרציה ישמשו גם את האיטרציה הבאה. מסתבר (אך מסובך להוכיח) שהשיטה היעילה ביותר היא שיטת חיפוש פיבונצי.

שיטת פיבונצי היא סכמה 4 נקודת (a, x1, x2, b) כמו שיטת השליש אלא ששלוש מהנקודות ישמשו בשלב הבא (ולא שניים כמו בשיטת השליש). ההחלטה איזה שלוש נקודות ישמשו באיטרציה הבאה תלויה בערכי הפונקציות בנקודות הפנימיות בדיוק כמו ב-quadratic fit. זאת למרות שכאן למען האמת המטרה היא מציאת קטע קטן שבו נמצא המינימום ולא הערכה אליו.

המאפיין העיקרי של שיטת פיבונצי הוא היכן למקם את נקודות החישוב של הפונקציה. בשיטה הזו מפעיל השיטה צריך **קודם כל להחליט כמה נקודות n**

הוא מתכוון לחשב את הפונקציה. אם, נניח, הקטנת הקטע היא המטרה אפשר לחשב חסם תחתון n למספר הנקודות שאנחנו צריכים:

***(b-a)φn < ε***

***nlog(φ) + log(b-a) < log(ε)***

***\_ \_***

***n = | log(ε)-log(ε)/log(φ)|***

ברגע ש-n ידוע אפשר לחשב את כל מספרי פיבונצי I = 0,1, …,n{Fi}.

הנקודות **הראשונות** יהיו במרחק Fn-1/Fn(b-a) ו- Fn-2/Fn (b-a) מהקצה הנמוך, ואשר לנקודות הבאות, הנקודה ה-i –ית תחושב במרחק Fi/Fnמאורך הקטע ה-i –יי מהקצה ה"נכון" כתלות בערכים היחסיים בין f(x1) ו- f(x2).

בהנחה שאנחנו נמצאים עם רביעייה (a, x1, x2, b)

אם f(x2) < :f(x1)

תחושב נקודה חדשה עבור x\*2=b - (b-a)Fi/Fn

הרביעייה החדשה תהיה (x1, x2, x\*2, b)

אחרת (f(x2) > f(x1) (

תחושב נקודה חדשה עבור x\*1=a + (b-a)Fi/Fn

הרביעייה החדשה תהיה (a, x\*1, x1, x2)

ההשוואה **האחרונה** תהיה סמוך מאד לנקודה האחרונה שחושבה ותשמש להכריע מי מבין שני הקטעים הסמוכים לנקודה הזו תהיה הקטע הסופי.

האיטרציה ה-i-יי, 2<i תקטין את קטע ההערכה למינימום בפקטור של

***Fn-i/Fn-i+1***

והפקטור הזה יתכנס ל-φ כאשר n🡪∞.

לדוגמא, נניח האלגוריתם מופעל על פונקציה f(x) על קטע (a,b) שבו עבור f(x) המינימום קרוב מאד ל-a (נניח הפונקציה f(x) יורדת במורד תלול מימין ל-a ולאחר מכן עולה בהדרגה). במקרה כזה תמיד יהיה ש- f(x2) > (f(x1 . נניח שהוחלט ש-n=5, הנקודות שהפונקציה תחושב בהם יהיו:

(b-a)5/8, (b-a)3/8, (b-a)2/8, (b-a)1/8, (b-a)1/8+ε.

**ההיגיון מאחורי שיטת פיבונצי**

צריך לזכור את המטרה. שיטת פיבונצי (כמו בשיטת השליש) מטרתה לקצץ מרווח מאחד מקצוות הקטע הכולל את המינימום, בכדי שקטע האי הידיעה בדיוק היכן המינימום יהיה קטן יותר. אלא שבניגוד לשיטת השליש, אנחנו רוצים ששלוש מארבעת הנקודות ישמשו לאיטרציה הבאה, בניגוד לשתיים בשיטת השליש. בצורה כזו נוכל באיטרציה הבאה לחשב את פונקצית המטרה בנקודה אחת ולא שתיים כפי שזה בשיטת השליש.

על מנת שאחת מנקודות הביניים תוכל לשמש לאיטרציה הבאה, היא צריכה לקיים את אחד התנאים על הקטע המוקטן שהנקודות הקיימות מקיימות על הקטע הנוכחי.

הנקודות של הקטע הנוכחי מקיימות את התנאים הבאים:

נגדיר U= (b-a)/Fn כאשר Fn הוא מספר פיבונצי n. לצורך הניתוח, U ישמש כ"איבר היחידה" של מרחק עבורנו.

FnU

🡨----------------------------------🡪

a x1 x2 b

🡨-------🡪🡨---------------------🡪

Fn-2U Fn-1U

🡨-----------------🡪🡨---------🡪

Fn-1U Fn-2U

בעקבות ההשוואה אנחנו מגלים שאחד משתי המרווחים (a,x1) או (x2,b) אינו מכיל את המינימום ולפיכך אפשר להוציא אותו מהחיפוש למינימום.

לשם הפשטות נניח שגילינו שאנחנו לא צריכים את (x2,b) יותר. המקרה השני הוא סימטרי.

בהנחה הזו בשלב הבא הקטע (a,x2) ישמש (a',b') כלומר

Fn-1U

🡨--------------------🡪

a x1 x2 b

🡨-------🡪🡨-------🡪

Fn-2U Fn-3U

a' x2' b'

מנקודת ראות יחסית, x1 הקודם נמצא בדיוק היכן ש-x2' צריך להיות עכשיו:

אם נסמן m=n-1 אזי הקטע שלנו עכשיו באורך FmU ו- x1 נמצא במרחק Fm-1U מהקצה a' ו-Fm-2U מהקצה b'. לפיכך, אם יש לנו את הערך של f(x1) מהאיטרציה הקודמת הוא יכול לשמש f(x2') של האיטרציה הבאה. בכדי לבצע את האיטרציה הבאה, אנחנו צריכים רק לחשב את x1' ו-f(x1') ולבצע את ההשוואה והעדכון.

כל זה תופס עד לאיטרציה האחרונה. באיטרציה האחרונה אין שלב בא לעבור אליו ולכן ההיגיון שם הוא קצת אחר.

עבור המקרה השני ((a',x1',b') = (x1,x2,b)) הדיאגרמה האחרונה תהיה:

Fn-1U

🡨---------------------🡪

a x1 x2 b

🡨----🡪🡨---------🡪

Fn-3U Fn-2U

העקרון הוא כזה: על מנת שאחד הנקודות הפנימיות x1 ו-x2 תוכל לשמש באיטרציה הבאה (ובכך לחסוך חישוב פונקצית המטרה באחת הנקודות), הן צריכות להיבחר כך שהן יקיימו תנאים עבור הקטע הנוכחי שהן יוכלו לקיים גם עבור הקטע הבא, שיהיה קצר יותר. מבחינה מעשית פירוש הדבר ש-x1 יוכל להיות ה-x2' של האליטרציה הבאה, ו-x2 יוכל להיות x1' של האיטרציה הבאה.

**קוד C**

void compute\_fibo\_numbers(long double fibo[], long double n)

{

unsigned int i;

fibo[0] = fibo[1] = 1.0;

for (i=2; i <= n; i++)

fibo[i] = fibo[i-1] + fibo[i-2];

}/\* compute\_fibo\_numbers \*/

void fibo\_search(long double (\*fp)(long double),

long double a, long double b, unsigned int n,

long double \*aptr, long double \*bptr)

{

long double x1, fx1, x2, fx2, fxs;

long double last, delta;

long double \*fibo\_n;

long double fn;

long double diff;

int i;

fibo\_n = (long double\*)malloc((n+1)\*sizeof(long double));

compute\_fibo\_numbers(fibo\_n, n);

fn = fibo\_n[n];

diff = b-a;

x1 = a+diff\*fibo\_n[n-2]/fn;

x2 = a+diff\*fibo\_n[n-1]/fn;

fx1 = (\*fp)(x1);

fx2 = (\*fp)(x2);

last = x1;

for(i = n-3; i > 0; i--)

{

if (fx1 > fx2)

{

a = x1;

x1 = x2;

fx1 = fx2;

x2 = b - diff\*fibo\_n[i]/fn;

fx2 = (\*fp)(x2);

last = x2;

} /\* if \*/

else

{

b = x2;

x2 = x1;

fx2 = fx1;

x1 = a + diff\*fibo\_n[i]/fn;

fx1 = (\*fp)(x1);

last = x1;

}/\* else \*/

} /\* for \*/

/\* last comparison \*/

delta = (b-a)/64.0;

if (last == x1)

{

fxs = (\*fp)(x1+delta);

if (fxs < fx1)

{

\*aptr = x1;

\*bptr = b;

} /\* if \*/

else

{

\*aptr = a;

\*bptr = x1;

} /\* else \*/

} /\* if \*/

else /\* last == x2 \*/

{

fxs = (\*fp)(x2-delta);

if (fxs < fx2)

{

\*aptr = a;

\*bptr = x2;

} /\* if \*/

else

{

\*aptr = x2;

\*bptr = b;

} /\* else \*/

} /\* else \*/

} /\* fibo\_search \*/

שיטת פיבונצי הוא אולי אופטימלית, אבל היא מסובכת שלא צורך ולא טבעי להגדיר אלגוריתם במספר הצעדים שהוא אמור לעשות, וצריך גם לחשב ןלשמור את מספרי פיבונצי.

**חיפוש יחס הזהב golden section**

אם נותנים ל-n לשאוף לאינסוף מקבלים את שיטת יחס הזהב. שוב זו סכמה של ארבע נקודות בשיטה הזו בכל איטרציה הרביעייה (a, x1, x2, b) :

***x1 = a+ (1-φ)(b-a)***

***x2 = a + φ(b-a)***

***φ= (√5 -1)/2 ≈ 0.618***

ברור ששיטה זו הרבה יותר פשוטה לתכנות.

בכדי לראות איך מגיעים לכך נעשה את החישוב הבא:

בכל שלב נבחרות 4 נקודות :a, x1, x2, b

| f(a)

| f(b)

| f(x2)

| f(x1)

|

| f(x2)

|

|

\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a x1 x2 b

🡨 u 🡪🡨 ------------ v -------------🡪

🡨----- w ----🡪

אנחנו רוצים לבחור את x1 ו-x2 כך ששלוש מתוך ארבע הנקודות "יעברו" לשלב הבא. לשם המחשה נניח שהתחלנו עם (a, x1, b), x1 קרוב יתר ל-a מאשר b ולפיכך ולפיהם בחרנו את x2 שמבחינת יעיליות עדיף שיהיה ב"קטע הגדול" שבין x1 ל-b.

השיטה שבו "נבחר" את x2 תהיה בדיעבד השיטה שבה נבחרה x1.

השלישיה "שתמשיך" לשלב הבא תהיה:

(I)אם f(x1) > f(x2) זו תהיה (x1, x2,b) = (a',x1',b') . כלומר במקרה הזה x2 ישמש כנקודה **הקרובה לקצה השמאלי** של הקטע (ה-x1 של השלב הבא).

(II) אחרת ( f(x1) < f(x2)) זו תהיה (a, x1,x2) = (a',x2',b') . כלומר במקרה הזה x1 ישמש **כנקודה הקרובה לקצה הימני** של הקטע (ה-x2 של השלב הבא).

בכדי ששתי האפשרויות של שלוש הנקודות אכן יוכלו לשמש אותנו עם בדיוק אותו אפקט בשלב הבא היחסים בין הנקודות בשלב הנוכחי צריכים להשתמר בשלב הבא כלומר:

עבור מקרה (I) x2 אכן צריך להיות ה- x1'של השלב הבא, כלומר:

***w/(v-w) = u/v (\*)***

עבור מקרה (II) x1 אכן צריך להיות ה- x2'של השלב הבא, כלומר:

***w/u = u/v (\*\*)***

אפשר לחשוב על זה במושגים של "קטע קצר" ו-"קטע ארוך".

בשלישיה המקורית ה"קטע הקצר" הוא u ו"הקטע הארוך" הוא v.

עבור השלישיה החדשה:

במקרה (I) "הקטע הקצר" הוא w ו"הקטע הארוך" הוא v-w.

במקרה (II) "הקטע הקצר" הוא w ו"הקטע הארוך" הוא u.

אם נחלץ את w מ-(\*\*) ונציב ב-(\*):

***w = u2/v***

***u2/(v(v - u2/v)) = u/v***

***u/(v - u2/v) = 1***

***v - u2/v = u***

***(u/v)2 + (u/v) -1 =0***

***u/v = (-1 + √5)/2***

***u/v = (-1 - √5)/2***

כלומר

***u/v = (√5 -1)/2 ≈ 0.618***

**קוד C**

void golden(long double (\*fp)(long double), long double a, long double b,

long double eps, long double \*aptr, long double \*bptr)

{

long double x1, fx1, fb, x2, fx2;

long double phi = 0.61803398874989484820;

long double phi1 = 1.0 - phi;

x1 = a + (b-a)\*phi1;

fx1 = (\*fp)(x1);

x2 = a + (b-a)\*phi;

fx2 = (\*fp)(x2);

do {

if (fx1 > fx2)

{

a = x1;

x1 = x2;

fx1 = fx2;

x2 = a + (b-a)\*phi;

fx2 = (\*fp)(x2);

}

else

{

b = x2;

x2 = x1;

fx2 = fx1;

x1 = a + (b-a)\*phi1;

fx1 = (\*fp)(x1);

} /\* else \*/

} while ( (b - a) > eps);

\*aptr = a;

\*bptr = b;

} /\* golden \*/