**שיטת ניוטון-רפסון**

כאשר נגזרת של הפונקציה הנחקרת ניתנת לחישוב כמו הפונקציה עצמה אם משום שהיא ידועה אנליטית או משום שאפשר לקרב אותה, ניתן לנסות לפתור את הבעיה ע"י שיטת ניוטון – רפסון, שתתכנס לכל פונקציה מונוטונית ובדרך כלל תתכנס אם למשוואה שורש יחיד ואיננו קפריזי באופן מיוחד.

השיטה היא בעקרון די פשוטה, פונקציה f(x), פונקצית נגזרת f'(x) ובהנתן קירוב לשורש xn, הקירוב הבא לשורש יהיה

***xn+1 = xn – f(xn)/f'(xn)***

לדוגמא, אם אנחנו רוצים למצוא שורש ריבועי x0√ של מספר חיובי x0 ע"י פתירת המשוואה f(x) = x2 – x0 אזי הנוסחה המתקבלת היא

***xn+1 = xn – (xn2 – x0 (/2xn = xn – xn/2 + x0 /2 xn = (xn + x0/xn)/2***

כלומר

***xn+1 = (xn + x0/xn)/2***

אותה שיטה שחקרנו קודם בהקשר של נקודות שבת.

אינטואציה: צורה אחת להגיע לשיטת ניוטון הוא הלך המחשבה של "להביא" את הבעיה למצב שנוכל "לפתור" אותו ע"י מציאת שורש של קו ישר, כפי שעשינו בשיטת regula falsi. אם נתון קירוב xn לשורש, ואם אפשר לחשב את f(x) ו-f'(x) בכל נקודה, אז ניתן לחשב את המשיק ל-f(x) בנקודה xn :

***y - f(xn) = f'(xn)(x – xn)***

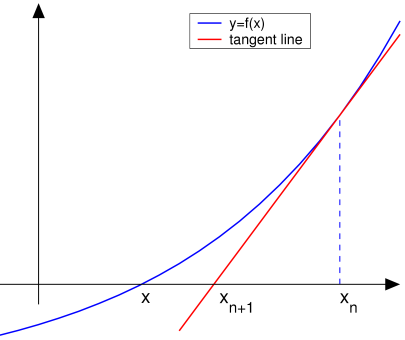
השורש של המשיק יהיה בנקודה y = 0 ומבחינתנו זו הנקודה שבה x = xn+1 לפיכך:

***-f(xn)= f'(xn)(xn+1 – xn)***

חילוק שני האגפים ב- f'(xn)והעברת xn אגף תיתן את הנוסחה

***xn+1 = xn – f(xn)/f'(xn)***

הציור הבא ממחיש את נקודת ההשקפה הזו:



נוסחת ניוטון תהיה איפוא מדויקת אם אנחנו מחפשים שורש של קו ישר - איטרציה אחת תיתן לנו את הפתרון. במקרים אחרים זה יהיה, במקרה הטוב, שיפור של הקירוב לשורש.

נקודת השקפה אחרת על שיטת ניוטון הוא להסתכל עליו מנקודת ראות של נוסחת טיילור סביב השורש:

***f(x\*) = f(x) + (x – x\*) f'(x) + (x – x\*)2 f''(ξ)/2***

אבל 0 = f(x\*) לכן

***(x\* – x) f'(x) + (x\* – x)2 f''(ξ)/2 = 0***

אם נניח ש-x קרוב למדי ל-x\*, הנגזרת השניה קיימת ו-0 ≠ f'(x\*) (כלומר x\* אינו שורש של f'(x) ) אז אולי אפשר להזניח את (x – x\*)2 f''(ξ)/2 ואז נחלץ את x\*

***f(x) + (x\* – x) f'(x) = 0***

***x\* – x = -f(x)/f'(x)***

***x\* = x - f(x)/f'(x)***

וזוהי נוסחת ניוטון.

מהניתוח שלעיל גם משתמע מה שנקרא "קצב ההתכנסות" של שיטת ניוטון. בתנאים מסוימים אם xn נמצא במרחק ε מהפתרון ( ε > | |x\* – xn ) אזי בתנאים הללו מרחק של xn+1 מ- x\* הוא O(ε2) ( 2εk > | |x\* – xn+1 עבור כלשהוא).

יש מצבים ששיטת ניוטון אינה מתכנסת או מתכנסת לאט. זה בדרך כלל קורה כאשר x\*הוא שורש של הנגזרת או שהנגזרת השנייה לא קיימת בנקודה הזו. דוגמא לכך היא הפונקציה

***f(x) = x + x2sin(2/x)***

**סיכום**

שיטת ניוטון לא תמיד מתכנסת, היא עשויה לא להתכנס גם כאשר נקודת ההתחלה רחוקה מדי. לכן שיטת ניוטון היא במקרה הכללי שיטה לחדד במהירות קירוב טוב לפתרון, ואת הקירוב הראשוני צריך בשיטה אחרת.

שיטת ניוטון מחיבת יכולת לחשב את הנגזרת בכל נקודה. פונקצית הנגזרת לא תמיד ידועה ולעיתים החישוב (נניח ע"י סכמה הגבול) הוא יקר ולא משתלם להשקיע בשביל הדיוק, שבן מדובר בסך הכל בסכמה לא מדויקת ולא תמיד יש צורך לחשב בדיוק מירבי את החישוב הלא מדויק ממילא.

**קוד C**

long double newton(long double (\*fun)(long double),

long double (\*fd)(long double),

long double x0, long double eps)

{

long double fdv, f0;

int i;

do {

f0 = (\*fun)(x0);

fdv = (\*fd)(x0);

x0 = x0 - f0/fdv;

} while (fabsl(f0) > eps);

return x0;

} /\* newton \*/

**שיטת המיתר**

שיטת המיתר הוא מעין שילוב של regula falsi ושיטת ניוטון. יתרונה על פני ניוטון היא שהיא אינה דורשת את היכולת לחשב את הנגזרת של הפונקציה שאת השורש שלה אנחנו רוצים לחשב. בכדי להימנע מכך, הסכמה של חישוב xn+1 מסתמך על 2 נקודות קודמות כלומר הסכמה היא מצורה 2 או

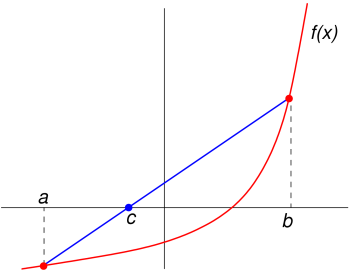
***xn+1 = F(xn, xn-1)***

המחיר הוא שצריך לספק לאלגוריתם 2 קירובים ראשוניים שונים לפתרון.

השיטה היא די פשוטה: בהינתן שני קירובים לשורש (xn-1,f(xn-1))

ו-(xn,f(xn)) הערך xn+1 יהיה פשוט השורש של הקו הישר המחבר את שתי הנקודות.

באופן גרפי זה נראה כך:



נוסחת החישוב של xn+1 הינה:

***xn+1 = xn  - (xn - xn-1)f(xn)/(f(xn) – f(xn-1))***

זה כאילו לוקחים את שיטת ניוטון ולוקחים את

***f'(xn) ≈ (f(xn) – f(xn-1))/(xn – xn-1)***

כקירוב של הנגזרת.

כעת נוכיח xn+1 אכן השורש של המיתר:

משוואת הקו הישר הינו:

***y – f(xn) = (f(xn) –f(xn-1))(x– xn)/(xn - xn-1)***

הערך שאנחנו מחפשים הוא הערך של x שבה y = 0, כלומר

***-f(xn) = (f(xn) –f(xn))(x– xn)/( xn - xn-1)***

***-f(xn)( xn - xn-1)/ (f(xn) –f(xn-1)) = x - xn***

***x = xn -f(xn)( xn - xn-1)/ (f(xn) –f(xn-1))***

או

***xn+1 = xn - (xn-xn-1) f(xn)/ (f(xn) –f(xn-1))***

קצב ההתכנסות של שיטת המיתר נמוך מזה של שיטת ניוטון – 1.618 לעומת 2.0. עדיין משתלם לפעמים להשתמש בשיטה הזו אם חישוב הנגזרת הוא דבר יקר, ובודאי אם הנגזרת לא ידועה בדיוק.

**קוד C**

long double secant(long double (\*fun)(long double),

long double x1,

long double x0, long double eps)

{

long double f0, f1, oldx1;

int i;

do {

f0 = (\*fun)(x0);

f1 = (\*fun)(x1);

oldx1 = x1;

x1 = x1 - ((x1-x0)/(f1-f0))\*f1;

x0 = oldx1;

} while (fabsl(f1) > eps);

return x1;

} /\* secant \*/

**השיטה של Steffensen**

השיטה של Steffensen הוא מעין שיטת ביניים בין שיטת המיתר ושיטת ניוטון. שיטת ניוטון מחייב ידיעת הנגזרת ומימוש שני פונקציות, זו הנחקרת והנגזרת שלה. שיטת המיתר מחייבת ידיעת שני ניחושים התחלתיים. לעומת זאת שיטת Steffensen מחייבת מימוש פונקציה אחת (זו הנחקרת) ונקודה התחלתית אחת כל זה במחיר חישוב הפונקציה בשני נקודות פר איטרציה. זאת לעומת שיטת המיתר שמחשבת ערך אחד של פונקציה פר איטרציה.

השיטה של Steffensen דומה לשיטת ניוטון בכך שהיא שעבור פולינום נתון השיטה מתכנסת בסדר ריבועי קרוב לפתרון מסדר ראשון (לינארי כאשר מדובר בפתרון מסדר שניים או יותר). כמוהו ההתכנסות תלויה מאד בניחוש טוב עבור הערך ההתחלתי.

תאור השיטה:

בהינתן x0 שיטת Steffensen מחשבת סידרת נקודות לפי הנוסחה הבאה:

xn+1 = xn – f(xn)/g(xn)

כאשר

g(xn) = { f(xn + f(xn)) – f(xn)}/f(xn)

אינטואיטיבית אם הפונקציה הנחקרת היא מונוטונית עולה וחוצה את ציר ה-X באזור השורש משליי לחיובי, חישוב הנוסחה של Steffensen תמפה נקודה לפני השורש קדימה ונקודה אחרי השורש אחורה.

קוד C:

long double steffensen(long double (\*fun)(long double),

long double x0, long double eps)

{

long double f0, f1, g;

int i;

f0 = (\*fun)(x0);

while (fabsl(f0) > eps)

{

f1 = (\*fun)(x0 + (\*fun)(x0));

g = (f1 - f0)/f0;

x0 = x0 - f0/g ;

f0 = (\*fun)(x0);

} // while

return x0;

} /\* steffensen \*/

**שיטת ה-Inverse quadratic interpolation**

**(שיטת הפרבולה ההפוכה)**

גם בשיטה זו לא צריך לדעת לחשב את הנגזרת של פונקצית המטרה. היא מחייבת שלוש נקודות קירוב התחלתיות לפתרון.

הרעיון של השיטה הזו היא לנסות לקרב את ההשתנות של x כפונקציה של y = f(x) ולחשב את הערך של x בנקודה y=0. לשון אחר היא לומר שאנחנו רוצים לקרב את x(y) ולחשב באמצעות הקירוב הזה קירוב לערך x(0).

בכדי להמחיש את הרעיון, נניח שאנחנו רוצים למצוא את השורש של הקו הישר

***y = f(x) = 3x+6***

***אזי***

***x(y) = f-1(y) = (1/3)y -2***

***והשורש הוא***

***x(0) = f-1(0) = (1/3)0 -2 = -2***

כמובן שאנליטית השיטה הזו תעבוד לעיתים רחוקות ביותר. הפונקציה ההפוכה לא תמיד קיימת בכלל (הפונקציה f לא חייבת להיות בעלת ערך יחודי בכל נקודה).

מה שאפשר לעשות באנליזה נומרית הוא להיות מעין אופטימיים נצחיים, כלומר לנסות לקרב את הפונקציה ההפוכה על סמך נקודות שהיא עוברת בהם. למשל, אם נתונים שתי נקודות

***(y0, x0) = (f(x0), x0) = (f0,x0), (y1, x1) = (f(x1), x1) = (f1,x1)***

אפשר לקרב את הפונקציה x(y) בעזרת הקו הישר

***x(y) = x0 + {(x1 – x0)/(y1-y0)}(y – y0) =***

***x0 + {(x1 – x0)/(f1-f0)}(y – f0)***

ואז

***x(0) = x0 - {(x1 – x0)/(f1-f0)}f0***

או

***xn = xn-2 - {(xn-1 – xn-2)/(fn-1-fn-2)}fn-2***

זהו פשוט שיטת המיתר (secant) שאנחנו כבר מכירים. ניסיון לקרב את הפונקציה ההפוכה בעזרת קו ישר לא תרם לנו הרבה אולם, אם ננסה לקרב אותה על ידי פרבולה נצליח יותר.

בקירוב של הפונקציה תהיה הבאה: בהינתן **שלוש** נקודות שבהן עוברת הפונקציה x(y):

(f(x0), x0), (f(x1), x1), (f(x2), x2)

נעביר פרבולה p(y) שעוברת בשלוש הנקודות הללו:

נסמן:

***f(x0) = f0, f(x1) = f1, f(x2) = f2,***

***p(y) = {(y-f1)(y-f2)x0})/{(f0-f1)(f0-f2)} +***

***{(y-f0)(y-f2)x1})/{(f1-f0)(f2-f1)} +***

***{(y-f0)(y-f1)x2})/{(f2-f0)(f2-f1)}***

הנוסחה של p(y) הוא מקרה פרטי של **נוסחת האינטרפולציה של לגרנג'**  שנדון בקורס בפרק אחר. אבל כאן מספיק לשים לב לשתי עובדות:

1. p(y) הוא פרבולה ב-y, היתר קבועים והחזקה של y בכל אחד משלושת המחוברים הו 2.
2. p(f0) = x0, p(f1) = x1, p(f2) = x2. בכל השמה של fi מתאפסים שניים מן הגורמים והאיבר הנותר xi. לדוגמא אם נחשב p(f1)

***p(f1) = {(f1-f1)(f1-f2)x0})/{(f0-f1)(f0-f2)} +***

***{(f1-f0)(f1-f2)x1})/{(f1-f0)(f2-f1)} +***

***{(f1-f0)(f1-f1)x2})/{(f2-f0)(f2-f1)} =***

***0 + 1x1 + 0***

המקדמים של x0 ו-x2 הם אפס כי כל אחד מהם מכיל את הגורם f1-f1. המקדם של x1 הוא אחד משום שהמונה והמכנה של המקדם של x1 הינם ((f1-f0)(f1-f2 והם מצמצמים זה את זה.

אילו p(y) היתה פונקציה שמזדהה עם x(y) אזי p(0) היה מחשב לנו x\* השורש של f(x) או f(x\*) = 0. מאחר ו-p(y) הוא במקרה הטוב רק קירוב ל-x(y) הערך של p(0) משמש כקירוב משופר לשורש x3 לשורש. כלומר

***x3 = p(0) = {(-f1)(-f2)x0})/{(f0-f1)(f0-f2)} +***

***{(-f0)(-f2)x1})/{(f1-f0)(f2-f1)} +***

***{(-f0)(-f1)x2})/{(f2-f0)(f2-f1)}***

***x3 = p(0) = f1 f2 x0/{(f0-f1)(f0-f2)} +***

***f0 f2 x1/{(f1-f0)(f2-f1)} +***

***f0 f1 x2/{(f2-f0)(f2-f1)}***

או באופן כללי

***xn+1 = p(0) = fn-1 fn x0/{(fn-2-fn-1)(fn-2-fn)} +***

***fn-2 fn xn-1/{(fn-1-fn-2)(fn-fn-1)} +***

***fn-2 fn-1 xn/{(fn-fn-2)(fn-fn-1)}***

p(y) הוא קירוב ל-x(y) במובן הזה שהוא מזהה איתו בשלוש נקודות. שום דבר לא מבטיח ש-p(y) הוא קירוב מעולה ל-x(y) אבל אם x0, x1, x2 הם בחזקת ניחוש טוב האלגוריתם יכנס במהירות לפתרון. אם הם רחוקים מהפתרון האלגוריתם עשוי להתכנס לאט ואפילו בכלל לא. האלגוריתם הזה כמו רבים מקודמיו בדרך כלל אינו בשימוש בפני עצמו אלא כאלגוריתם לשיפור קירובים טובים, בדרך כלל כאלגוריתם קצה של האלגוריתמים שיטת החצייה והמיתר.

תוכנית :C

long double invquad(long double (\*fun)(long double),

long double x0, long double x1, long double x2, long double eps)

{

long double f0, f1, f2, c0, c1, c2, temp;

int i;

f0 = (\*fun)(x0);

f1 = (\*fun)(x1);

f2 = (\*fun)(x2);

while (fabsl(f2) > eps)

{

c0 = f2\*f1/((f0-f1)\*(f0-f2));

c1 = f0\*f2/((f1-f0)\*(f1-f2));

c2 = f0\*f1/((f2-f0)\*(f2-f1));

temp = c0\*x0 + c1\*x1 + c2\*x2;

x0 = x1;

x1 = x2;

x2 = temp;

f0 = f1;

f1 = f2;

f2 = (\*fun)(x2);

} // while

return x2;

} /\*invquad \*/