**חישוב פונקציות**

 אחד התפקידים של האנליזה הנומרית היא לחשב ערכים מספריים של פונקציות ידועות.

**חישוב פולינום בנקודה לפי כלל הורנר**

בהינתן פולינום

***p(x, n) =a0 xn + a1 xn-1 + … +an-1 x + an***

חישוב הפולינום על פי ההגדרה באופן נאיבי עשוי להוביל לשיטות בזבזניות למדי בחישובים. שיטה חסכונית בחישובים לחישוב ערך של פולינום הוא חישוב לפי כלל הורנר:

***p(x,n) = ((…( a0\*x +a1)\*x +a2)\*x + …. )\*x + an***

המימוש בשפת C היא פשוטה יחסית. בהנחה שמקדמי הפולינום נתונים בצורה של מערך אזי המימוש יהיה:

float horner(float coeffs[], float x, int n)

{

 int i;

 float polyx;

 polyx = coeffs[0];

 for(i = 1; i <= n; i++)

 polyx = polyx \* x + coeffs[i];

 return polyx;

} /\* horner \*/

**חישוב פונקציות ע"י טורי (נוסחת) טיילור**

 **מוטיבציה**: הגם שפונקציות כמו ex, sin(x), arcsin(x) ידועות ברמה האנליטית, עדיין חישוב הערכים של הפונקציות הללו בדוק גבוה לכל נקודה רצויה איננו עניין של מה בכך. תורי טיילור הם שיטה כללית ומצוינת לביצוע המשימה הזו עבור פונקציות שידוע עליהם הפונקציות ה-n נגזרות שלהם. למען האמת, טורי טיילור הם קצת יקרים לחישוב, והם מהווים בדרך כלל שיטה לחישוב טבלאות ערכים ברמת ספרית תוכנות שמהם מחושבים ערכי הפונקציה ברמת האפליקציה. על שיטה זו, הנקראית דיסקרטיזציה, נרחיב בפרק על אינטרפולציה.

 טורי טיילור בגרסתם האינסופית אינם בעלי משמעות עבור האנליזה הנומרית אבל תהליכים אינסופיים מהסוג הזה בכל זאת יכולים להיות בסיס למימוש קירובים חישוביים כאשר קוטעים את התהליך האינסופי בנקודה סופית. השאלה תמיד, היא באיזה תנאים הדבר אפשרי, ומתי מבצעים את הקטיעה הזו.

**רקע מתמטי**: משפט קלאסי מחדו"א שכל פונקציה של משתנה ממשי אחד המקיימת תנאים מסוימים.

גירסת **ערך הביניים** לטורי טיילור (הכללה של נוסחת לגרנז') הקרויה **נוסחת טיילור** של ערך פונקציה כזו סביב נקודה a הינה:

**נוסחה 1:**

***f(x) = f(a) +(x-a) f'(a) +(x-a)2/2! f''(a)+ …***

 ***… + (x-a)n-1/(n-1)! f(n-1)(a) + (x-a)n/n! f(n)(a+ θ(x-a))***

 ***0 < θ < 1***

צורה נוספות של הנוסחה:

**נוסחה 2:**

***f(x+h) = f(x) +h f'(x) +h2/2! f''(x)+ …***

 ***… + hn-1/(n-1)! f(n-1)(x) + hn/n! f(n)(x+θh)***

 ***0 < θ < 1***

וכמובן נוסחת מקלורן:

**נוסחה 3:**

***f(x) = f(0) + x f'(0) +x2/2! f''(0)+ …***

 ***… + xn-1/(n-1)! f(n-1)(0) + xn/n! f(n)(θx)***

 ***0 < θ < 1***

המתואר לעיל נקרא נוסחת טיילור ולא טור טיילור משום שזו נוסחה סופית. צריך לשים לב הוא שהנוסחה הזו היא **מדויקת**, מדובר בשוויון ממש ולא בקירוב.

האיבר האחרון בנוסחאות נקרא **איבר השארית** וכידוע אין דרך כללית לחשב אותו, לא ידוע (ואולי אין) דרך לחשב את θ באופן כללי. אולם בדרך כלל ניתן לחסום את הערך שהוא יכול לקבל ועל סמך זה להעריך את רמת הדיוק של הסדרה הקטועה.

התרשים הבא ממחיש את היחס בין פולינומים המורכבים מהאיברים הראשונים של טור טיילור והפונקציה השווה לטור האינסופי, sin(x) במקרה הזה : הקו הישר זה y = x, העקום השני הינו y = x - x3/6, העקום השלישי הינו y = -x3/6+x5/120 העקום הרביעי הינו y = x -x3/6+x5/120 – x7/5040 וכו'. ככל שאנחנו מחשבים את הפולינום עם יותר איברים של התור, הפולינום מקרב את sin(x) לאורך קטע יותר גדול משני צידי הראשית.



**תנאים מספיקים** לחישוב ערכי פונקציה באמצעות נוסחת טיילור:

 תנאי ראשון הוא כמובן שהפונקציה שאותה אנחנו רוצים לחשב מקיימת את התנאים של טור / נוסחת טיילור שעיקרם שהפונקציה גזירה מספר רב של פעמים ושהטור שלה מתכנס. מנקודת ראות חישובית צריך את התנאי הנוסף שבו אנחנו צריכים להכיר את הנגזרות הללו מספיק טוב בכדי לחשב אותם בנקודה הרצויה. כיוון שכך, הנסיבות הטבעיות לשימוש בנוסחת טיילור הם עבור פונקציות ידועות היטב מבחינה מתמטית כמו שילוב של פונקציות אלמנטריות כמו sinx, cosx, ex, lnx וכו'.

עבור כל x0 נתון, הביטוי x0n/n! שואף לאפס. את זה אפשר לראות ממספר עובדות ידועות על n! למשל

 ***\_\_\_***

***n! ~ √2лn (n/e)n***

או בגרסתו החלשה יותר

***(n/3)n < n! < (n/2)n  n > 6 כאשר***

ולפיכך

***x0n/n! < 3x0n/nn = (3x0/n)n***

וברור ש- (3x0/n)n מתכנס לאפס כאשר x0 מוחזק קבוע ו-n שואף לאינסוף. אך זה אינו מבטיח התכנסות של נוסחת טיילור כי לפעמים הגורמים של n! מתבטלים על ידי הליך הגזירה החוזרת f(n). לדוגמא,

***arctan(x) = x – x3/3 + x5/5 – x7/7 + … = Σ1∞(-1)nx2n+1/(2n+1)***

המתכנס רק כאשר |x| < 1.

מצב רצוי ולעיתים הכרחי איפוא הוא להגיע בנוסחאות 1-3 לעיל שבו המספר המועלה בחזרה יהיה קטן בערכו המוחלט מ-1. כלומר שהערכים

(x-a) בנוסחה 1

h בנוסחה 2

x בנוסחה 3

 יהיו קטנים בערכם המוחלט מ-1. בהרבה מקרים לא קשה להביא את הבעיה לצורה כזו ונראה דוגמאות לכך.

ישנו מקרה נפוץ שבו קל יחסית להעריך את השגיאה.

ישנם הרבה טורי טיילור שכל שני גורמים עוקבים מחליפים סימן.

במצב כזה אפשר לנסות להסתמך המשפט הבא:

**משפט 1:**

תהי Σ∞(-1)i-1 ai סידרה אינסופית המקיימת < an+1 < an 0 עבור כל n, ו- an 🡪 0, כאשר n🡪∞, אזי הסדרה מתכנסת, ומקיימת את התנאי הבא:

יהיו:

***S = Σ∞(-1)i-1 ai***

***Sn = Σn(-1)i-1 ai***

אזי ***S – Sn < an+1***.

במילים אחרות בטור סדרות מחליפות סימן האיבר האחרון שחישבת חוסם את שארית הטור, בתנאי שהסדרה שואפת מונוטונית לאפס (לא משנה באיזה קצב).

**דוגמאות לשימושים של נוסחת טיילור:**

1. חישוב ex.

 נניח שאנחנו רוצים לחשב את ex בעזרת נוסחת טיילור, נניח בגלל מצב (לא נפוץ) שפונקצית הספרייה אינה מדויקת מספיק.

***ex = 1 + x + x2/2! + x3/3! + … + xn/n! + (θx)n+1/(n+1)!***

***ez+h = ez + hez + h2ez/2! + h3/3! + … + hnez/n! + (θh)n+1ez/(n+1)!***

הנוסחה הראשונה נוחה עבור ערכים 0 < x < 1 והנוסחה השניה נוחה עבור ערכים גדולים יותר.

 עבור 1 < x אפשר לומר שלא צריך את טור טיילור בכדי למצוא את המספר השלם k ו- ek שמקימים:

***ek < ex < ek+1***

k= |\_x\_| הוא פשוט הערך השלם התחתון של x, ו-ek ניתן לחשב מ-e על ידי k כפלים.

ברגע מצאנו את אלה, נוכל לחשב את ex לפי

***ex = ek+h =***

***ek + hek + h2ek/2! + h3/3! + … + hnek/n! + (θh)n+1ek/(n+1)! =***

***ek(1 + h + h2/2! + h3/3! + … + hn/n!) + (θh)n+1ek/(n+1)!***

***כאשר 0 < θ <1***

ולפיכך השגיאה היחסית

1. ***(1 + h + h2/2! + h3/3! + … + hn/n!)/ex  < hn+1/(n+1)!***

**חישוב e-x**

 עבור חזקות שליליות של e, דרך אמינה היא לחשב את e|x| ולאחר מכן לחשב את 1.0/e|x|. זה בטוח נכון כאשר 1 << |x|. שימוש בנוסחת טיילור כפשוטו במצב הזה איננה כדאית משום שהתוצאה היא מאד קטנה והיא תוצאה של ערכים גדולים **המבטלים זה את זה**. בדיוק רגיל, למשל, התוצאה עשויה להיות חסר משמעות לחלוטין.

**קוד C**

חישוב e x לפי נוסחת מקלורן:

long double my\_exp(long double x, long double eps)

{

 long double n, nf, xp, result, term;

 result = 1;

 xp = x;

 nf = 1.0;

 n = 1.0;

 do {

 term = xp/nf;

 result += term;

 xp = xp\*x;

 n = n + 1.0;

 nf = nf \* n;

 } while (term > eps);

 return result;

} /\* my\_exp \*/

מימוש ex לפי טיילור:

long double compute\_e()

{

 long double n, nf, result, term, eps;

 result = 1;

 nf = 1.0;

 n = 1.0;

 eps = 0.000000000000000001;

 do {

 term = 1.0/nf;

 result += term;

 n = n + 1.0;

 nf = nf \* n;

 } while (term > eps);

 return result;

} /\* compute\_e \*/

long double my\_exp(long double x, long double eps)

{

 long double n, nf, xp, result, term, ek, e;

unsigned int k;

 e = compute\_e();

 k = x;

 x = x - k;

 ek = 1;

 for (; k > 0; k--)

 ek = ek \* e;

 result = 1.0;

 xp = x;

 nf = 1.0;

 n = 1.0;

 do {

 term = xp/nf;

 result += term;

 xp = xp\*x;

 n = n + 1.0;

 nf = nf \* n;

 } while (term > eps);

 result = ek\* result;

 return result;

} /\* my\_exp \*/

**חישוב ln(x)**

נוסחת טיילור עבור פונקצית ln יכולה להיכתב

***ln(1+x) = x - x2/2 + x3/3 – x4/4 + … +(-1)n xn+1/(n+1)***

בתנאי ש- ***|x| < 1***.

מאחר ומדובר בסדרה של איברים המחליפה סימן כאשר 0 < x, האיבר האחרון המחושב חוסם את השארית. אם x < 0 אפשר לחשב את ln(z) – כאשר z = 1/(1+x).

 קושי ברור מאליו הוא שבתנאים הללו אנחנו יכולים לחשב רק ערכים של ln בין 1 ל-2. בכדי להביא את החישוב לצורה כזו ניתן לעשות אותו לפי האלגוריתם הבא:

בכדי לחשב את ln(z) כאשר 2 < z:

חשב בדיוק גבוה את ln(2)

מצא את ה-k השלם המקים:

***2k < z < 2k+1***

לפיכך

אם  ***y = z/2k*** אזי:

***1 < y < 2***

***ln(y) = ln(z) – kln(2)***

***ln(z) = ln(y) +kln(2)***

את ln(y) ניתן לחשב לפי נוסחת טיילור כאשר

***x = y -1***

**קוד C**

התוכנית הבאה מחשבת את ln(x) עבור x > 1 כללי:

long double my\_lnx(long double z, long double eps)

{

 long double n, k, x, xp, result, term, ln2, y, sign;

 if ( z > 2.0)

 {

 ln2 = 0.6931471805599453094;

 k = 1.0;

 y = z/2.0;;

 while( y > 2.0)

 {

 y = y / 2.0;

 k = k + 1.0;

 } /\* while \*/

 } /\* if \*/

 x = y - 1;

 result = 0;

 xp = x;

 n = 1.0;

 sign = 1.0;

do {

 term = xp/n;

 result = result + sign\*term;

 sign = 0 - sign;

 xp = xp\*x;

 n = n + 1.0;

 } while (term > eps);

 result = result + k\*ln2;

 return result;

} /\* my\_lnx \*/

**חישוב sin(x)**

הפיתוח נוסחת מקלורן של sin(x) (x ברדיאנים) הינו:

***sin(x) = x – x3/3! + x5/5! + … + (-1)n x2n+1/(2n+1)!***

הטור מתכנס לכל x, אבל כדאי לחשב אותו על הערכים הקטנים ביותר שניתן, אחרת הוא יתכנס לאט יותר ושוב יהיה לנו מקרה של חשוב של ערכים גדולים המבטלים זה את זה ומכנסים שגיאות עיגול בסדר גודל של התוצאה.

אנחנו יודעים כמובן שאם 0<x<л/2

***sin(x) = sin(x+2л)***

***sin(x) = sin(л –x)***

***sin(x+л) = - sin(x)***

לפיכך בהינתן 2л x > , ניתן לחשב את sin(y) כאשר y = x - |\_x/2л\_|2л

אם л < y, ניתן לחשב את sin(z)- כאשר л -z = y. אם л/2 < z ניתן לחשב את sin(u)- כאשר u = л – z.

אם y < л > 2/л ניתן לחשב את לחשב את sin(u) כאשר u = л – y.

**קוד C**

 להלן קוד C המחשב את sin(x) לפי העקרונות הללו:

long double my\_sin(long double x, long double eps)

{

 long double sinx, xx, nf, R, xp, n, sign, flag;

 unsigned long int k;

 flag = 1.0;

 if (x < 0)

 {

 flag = -1.0;

 x = 0 - x;

 } /\* if \*/

if ( x> 2\*pi)

 {

 k = x/(2\*pi);

 x = x - k\*2\*pi;

 if (x > pi)

 {

 x = x - pi;

 flag = -1.0 \*flag;

 } /\* if \*/

 if (x > pi/2.0)

 x = pi - x;

 } /\* if \*/

 xx = x\*x;

 nf = 1.0;

 sinx = 0.0;

 xp = x;

 n = 1.0;

 sign = 1.0;

 do

 {

 R = my\_abs(xp / nf) ;

 sinx = sinx + sign \* R;

 sign = - sign;

 xp = xp \* xx;

 nf = nf \* ((n+1.0)\*(n+2.0));

 n = n + 2.0;

 } while( R >= eps );

 return flag \* sinx;

} /\* my\_sin \*/

**שימוש לאינטגרציה**

אחד השימושים של נוסחת טיילור הוא להתגבר על העובדה שאינטגרלים של פונקציות אפילו אלמנטאריות לא ניתן לבטא אנליטית או בצורה מדויקת יותר: האינטגרל של צירוף או הרכבה של פונקציות אלמנטאריות לא בהכרח ניתן לביטוי כצירוף או הרכבה של פונקציות אלמנטאריות. זאת משום שהצירוף המקורי לא חייבת להיות נגזרת של פונקציה אחרת או בלשון אחר הפונקציה האנטי נגזרת לא תמיד קיימת. דוגמא פשוטה לכך הוא הפונקציה

***sinI(x)= ∫0x(sin(t)/t)dt***

בשלב מאוחר יותר בקורס אנחנו נראה שניתן באופן כללי להתמודד עם הבעיה במה שנקרא אינטגרציה נומרית. אבל במקרים מסוימים ניתן להתמודד עם הבעה בעזרת נוסחת טיילור אם ידוע נוסחת טיילור של פונקצית המקור והוא מתכנס.

עבור sinI(x) יש לנו את נוסחת טייור של sin(t):

***sin(t) =t – t3/3! + t5/5! + … + (-1)nt2n+1/(2n+1)!***

לפיכך את sin(t)/t ניתן לבטא:

***sin(t)/t =1 – t2/3! + t4/5! + … + (-1)nt2n/(2n+1)!***

לפיכך את ∫(sin(t)/t)dt ניתן לבטא

***∫(sin(t)/t)dt =t – t3/(3 3!) + t5/(5 5!) + … +***

 ***(-1)nt2n+1/((2n+1)(2n+1)!)***

בהנחה ש-sinI(0) = 0 אנחנו מקבלים

***sinI(x)= x – x3/(3 3!) + x5/(5 5!) + … +***

 ***(-1)nx2n+1/((2n+1)(2n+1)!)***

תוכנית C:

float my\_abs(float x)

{

 if ( x >= 0)

 return x;

 else

 return -x;

} /\* my\_abs \*/

float pi;

float my\_Isinx\_x(float x, float eps)

{

 float sinx\_xx, xx, nf, R, xp, n, sign;

 xx = x\*x;

 nf = 1.0;

 sinx\_xx = 0.0;

 xp = x;

 n = 1.0;

 sign = 1.0;

 do

 {

 R = xp / ((2\*n+1)\*nf) ;

 sinx\_xx = sinx\_xx + sign \* R;

 sign = - sign;

 xp = xp \* xx;

 nf = nf \* ((2\*n)\*(2\*n+1.0));

 n = n + 1.0;

 R = my\_abs(R);

 } while( R >= eps );

 return sinx\_xx;

} /\* my\_sinx\_x \*/