נספח א' להאקטון 2:

משתתפים :עומרי אלמלם, ניצן ראש, שירי אבודרם, יניב בן צבי, לירון גליקמן, בוריס לבייקין.

היישומים שבהם השתמשנו:

-**Simpson method**

**trapezoidal-**

**Lagrange interpolation-**

**gauss method-**

**linear approximation-**

**polynomial approximation-**

**Neville-**

**Spline cubic-**

**Jacobi method-**

**Sor method-**

**Bisection method-**

**Scant method-**

**Newton-**

**LU-**

**שיטת החצייה:**

בדרך כלל אין אפשרות לפתור משוואה של f(x)=0 בצורה אנליטית ולכן נשתמש בשיות איטרטיביות למציאת קירוב לפתרון.

הרעיון הוא להתחיל לקירוב מסוים (ניחוש) ובכל איטרציה לנסות לשפר את הקירוב עד שנגיע לקירוב מספיק טוב.

שיטת החצייה מתבססת על הרעיון של חיפוש בינארי:

אם יש קטע[a,b] וידוע שיש שורש בקטע הזה

אז נתבונן על אמצע הקטע [a,b] ונמצא תחום שקטן בחצי מהשורש שבתוכו.

האלגוריתם:

(1) התחלה עם קטע כלשהו [Xo,X1] כך ש f(Xo) \*f(X)<0

(2) נחשב X2=(X0+X1)/2

(3) אם f(X2)=0 אז סיימנו!

(4) אם f(Xo) \*f(X)<0 אז נמשיך עם הקטע [X0,X2]

(5) אחרת נמשיך עם הקטע [X1,X2]

תנאי עצירה אפשריים:

א. F(X2)< ε (ערך הפונקציה קרוב ל אפס עד כדי אפסילון)

ב. |Xo-X1|< δ (הגענו לקטע מאוד קטן שבתוכו נמצא השורש)

ג. K>N כאשר k= מספר האיטרציות(חזרנו על התהליך מס' רק של איטרציות)

הקוד נלקח מה Github

קישור לקוד:

<https://github.com/TheAlgorithms/Python/blob/master/arithmetic_analysis/bisection.py>

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

קלט : פונקציה, נקודת התחלה, נקודת סוף

פלט: הקירוב של השורש ,החיתוך של המשיק לפונקציה עם ציר ה X באיטרציה הnית

הסבר על הקוד:

תחילה נאתחל את נקודת ההתחלה ונקודת הסוף ונדפיס אותן

נבדוק האם נגזרת של הפונ' מתאפסת בנקודות אם מתאפסת השיטה לא תעבוד,

אחרת נבדוק האם הערך המוחלט של ערך הפונקציה בנקודה הנוכחית חלקי ערך הנגזרת באותה הנקודה גדול מגודל השגיאה .

נבדוק האם הערך של הפונקציה בנקודה הנוכחית שווה ל אפס אם כן מצאנו שורש ונחזיר אחרת נחשב את הערך החדש להיות ערך ההפרש בין הנקודה הנוכחית לערך הפונ' בנקודה הנוכחית חלקי ערך הנגזרת של הפונ' בנקודה.

נבדוק האם ערך הנגזרת בנקודה הנוכחית שווה ל אפס אם כן לא ניתן למצוא שורש אחרת נמשיך את הבדיקה עד שנגיע לקירוב שמתאים לגודל השגיאה או עד שנגיע למס' האיטרציות הדרושות.

בכל שלב נדפיס את הקירוב המשתנה ואת נקודות האמצע והתחלה החדשות

שיטת החצייה DFD להוכחת נכונות הקוד:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**שיטת המיתר: (method Secant)**

דרישות: שני ניחושים סX ו 1X

אלגוריתם: ניחוש (קירוב) של הנקודה השלישית 3X מגיעה כתוצאה מהאיטרציה, היא נקודת החיתוך של הישר עם ציר ה- X.

איטרציה כללית:

יתרונות:

1. במידה והשיטה מתכנסת, התכנסותה מהירה יותר משיטות ליניאריות. ניתן לראות כי סדר

התכנסות של שיטה הוא 1/2 (1+√5)≈1.618

1. דרושים שני ניחושים התחלתיים אך אין דרישה לסימן שונה עבור ערכים של הפונקציה.

בכל איטרציה (למעט הראשונה) דרוש חישוב פונקציה אחד ויחיד.

חסרונות:

1. שיטה יכולה לא להתכנס
2. חילוק באפס - במידה ושיפוע של ישר משיק בנקודת האפס של פונקציה קרוב ל - 0.

הערות:

א. שיטה שתלויה ב-2 נקודות x\_(i+1)=G(x\_(i,) x\_(i-1) )

ב. \*אינטרפולציה ליניארית.

\*אינטרפולציה ליניארית - \*\*אינטרפולציה בעזרת 2 נקודות.

\*\* אינטרפולציה - היא שיטה המיועדת ליצירת נקודות חדשות של הנתונים בתחום קבוצת הנתונים הידועים המהווים קבוצת נקודות דיסקרטית.

הקוד נלקח מאתר ubc.

קישור לקוד:

<https://www.math.ubc.ca/~pwalls/math-python/roots-optimization/secant/>

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

קלט: פונקציה , ניחוש 1 , ניחוש 2 של נקודות , מספר האיטרציות

פלט: קירוב של השורש התלוי בטעות ובמספר האיטרציות המקסימליים.

הסבר של הקוד:

ראשית נבדוק האם עבור הערכים שניחשנו הפונ' הנתונה משנה סימן אם לא השיטה נכשלה

אם כן נגדיל את מספר האיטרציות ב 1

נעביר ישר בין ערכי הפונקציה ונחשב את נקודת החיתוך של הישר עם ציר ה X.

אם הנקודה שקיבלנו היא שורש המשוואה נחזיר אותה אם לא

נשנה את התחומים התחום העליון זה התחום העליון החדש והתחום הישן הינו הקירוב הנוכחי שקיבלנו

הדפסה של שינויים בערכים בכל איטרציה ומציאה קירוב לשורש המשוואה.

שיטת המיתר DFD לנכונות הקוד:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**שיטת ניוטון-רפסון:**

שיטת ניוטון-רפסון (או כלל ניוטון) היא אלגוריתם יעיל באנליזה נומרית, למציאת שורשים של פונקציה ממשית כלשהי, דהיינו נקודות בהן הפונקציה מתאפסת. השיטה פותחה באופן בלתי תלוי בידי אייזק ניוטון וג'וזף רפסון

השיטה מבוססת על הרעיון הבא:

בהינתן פונקציה שאת השורש שלה אנחנו מחפשים, ואנו מגבילים את עצמנו לתחום בו יש לפונקציה רק שורש אחד, אם נבחר נקודה קרובה לשורש, השורש של המשיק לפונקציה באותה נקודה יהיה קרוב יותר לשורש שאנו מחפשים. בכל איטרציה של הלולאה, יתקבל קירוב טוב יותר ויותר.

סדר הפעולות בשיטת ניוטון רפסון הוא:

1. בחירת נקודה קרובה לשורש המבוקש.
2. חישוב שיפוע המשיק לפונקציה בנקודה זו; זוהי הנגזרת של הפונקציה באותה נקודה.
3. חישב משוואת המשיק, באמצעות גאומטריה אנליטית.
4. מציאת שורש המשיק, כלומר הנקודה בה המשיק חותך את ציר ה-x.

אם בחירת הנקודה ההתחלתית הייתה טובה, הנקודה החדשה שהתקבלה קרובה יותר ממנה לשורש, ויש לחזור על התהליך עם הנקודה החדשה כנקודת ההתחלה. אם לא, הנקודה המתקבלת תחרוג מהתחום הנידון. תחת תנאים מסוימים ניתן להבטיח שהשיטה תעבוד היטב, גם עבור נקודות התחלתיות רחוקות מאוד מהשורש.

יתרונות של השיטה :

1. אין צורך לזכור 2 נקודות
2. ההתכנסות היא ריבועית P=2
3. אפשר להכליל להרבה ממדים.

חסרונות:

1. חישוב נגזרת עלול להיות קשה.
2. נגזרת עלולה להתאפס(לדוגמה בנק קיצון)

הקוד נלקח מהGithub

קישור לקוד:

<https://github.com/leesongithub/BisectionandNewtonMethods/blob/master/Newton's%20Method/MAD4401NewtonsMethod.py>

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

קלט: הפונקציה , הנגזרת של הפונקציה, גבול עליון , גבול תחתון , סטייה מקסימלית עם ערך דיפולטי של 0.1, ומספר איטרציות עם ערך דיפולטי של 10.

פלט : הדפסה של הקירוב של השורש שהוא בעצם החיתול של המשיק עם ציר הX במספר האיטרציות שהוכנס לו(או בערך הדיפולטי)

הסבר הקוד: הקוד ירוץ כל עוד הערך המוחלט של ערך הפונקציה בנקודה x הנוכחית חלקי ערך נגזרת הפונקציה באותו x גדול מגודל השגיאה והאם מספר האיטרציות הנוכחי לא גדול ממספר האיטרציות המקסימלי.בכל איטרציה הפונקציה תשמור את מספר האיטרציות שלה.

אם הגענו למספר האיטרציות או אם עברנו את הסטייה נעצור וה x שקיבלנו זהו השורש הרצוי

הקוד מחזיר רק נקודה אחת מהתחום שקיבל!

שיטת ניוטון-רפסון DFD לנכונות הקוד:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**גאוס זיידל:**

תמונה שמכילה חיה

התיאור נוצר באופן אוטומטי

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

קלט**:** מקבל מטריצה,וקטור, מקסימום שגיאה ,מקסימום איטרציות

פלט: מדפיס את וקטור X באיטרציה ה Nית שתלויה בשגיאה ובמקסימום איטרציות.

**שיטת SOR :**

שיטת SOR נועדה על מנת לזרז את התכנסותה של שיטת Gauss-Seidel..

**תמונה שמכילה אובייקט

התיאור נוצר באופן אוטומטי**

ω הוא פרמטר הרלקסציה אשר מקיים 0< ω  <2 על מנת להאיץ תהליך התכנסות איטי לוקחים ω>1

ואילו על מנת להבטיח התכנסות עבור שיטות מתבדרות, לוקחים ω<1

עבור ω=1 מקבלים חזרה את שיטת GS.

עבור מטריצה A נתונה, קיים ערך אופטימלי של ω אשר מביא להתכנסות המהירה ביותר

ניתן להוכיח שאם שיטת GS מתכנסת, אז שיטת SOR תתכנס מהר יותר.

דרך אחרת לפיתוח השיטה הוא באמצעות שימוש ייצוג המטריצה A באמצעות המכפלה DLU, כאשר D היא מטריצה אלכסונית

הקוד נלקח מהאינטרנט

קישור לקוד :

<https://trinket.io/python/aea71b62f6>

**תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי**

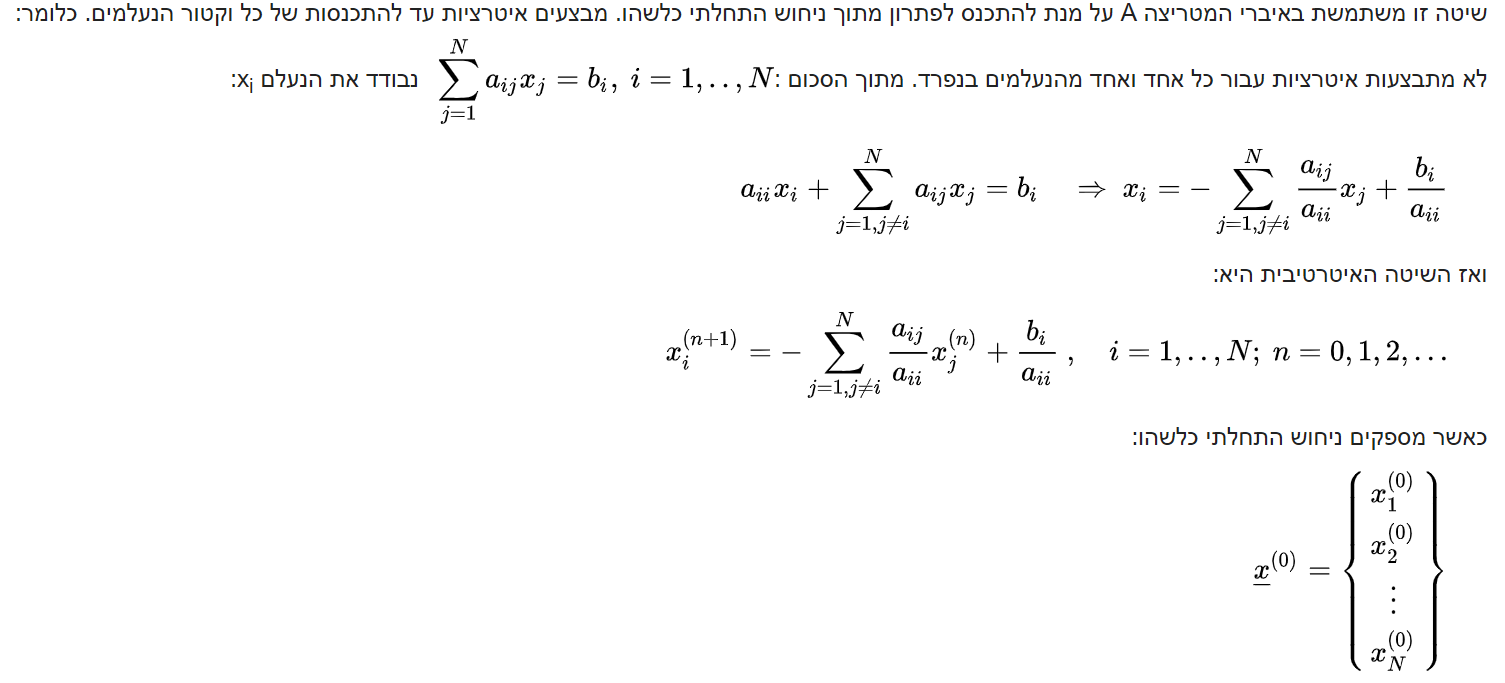
הסבר הקוד:

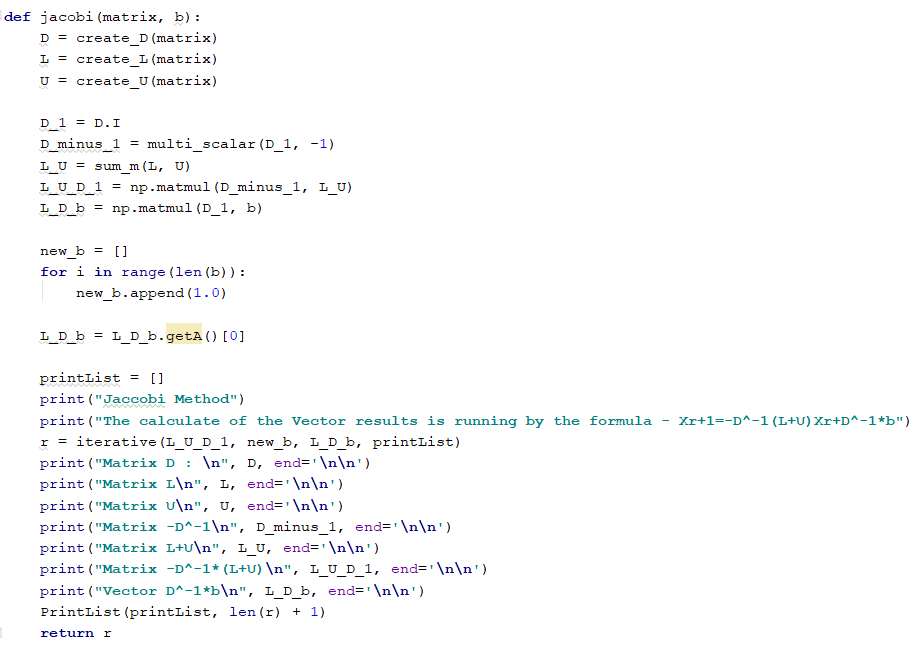
מדרגת את המטריצה עם אלכסון דומיננטי ומפעיל את גאוס זיידל

קלט: מקבל את מטריצה A , מקבל את b שזה ווקטור התוצאה של המשוואה , X0 בתור הניחוש הראשוני , השגיאה המקסימלית , מס' מקסימלי של איטרציות , והאם לבצע הדפסות או לא לבצע הדפסות לפי 1 ו 0 בהתאמה.

פלט: מחזיר לנו את ווקטור הפתרון ( X )

**יעקבי:**





הסבר הקוד:

שיטה זו משתמשת באיברי המטריצה A על מנת להתכנס לפתרון מתוך ניחוש התחלתי כלשהו. מבצעים איטרציות עד להתכנסות של כל וקטור הנעלמים.   
קלט: מקבל את מטריצה A , מקבל את b שזה ווקטור התוצאה של המשוואה ,   
פלט: מחזיר לנו את ווקטור הפתרון ( X )

**פתרון של מערכות משוואות ליניאריות:**

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

{\displaystyle \left\{{\begin{array}{rcrcccrcl}a\_{11}x\_{1}&+&a\_{12}x\_{2}&+&\cdots &+&a\_{1n}x\_{n}&=&b\_{1}\\a\_{21}x\_{1}&+&a\_{22}x\_{2}&+&\cdots &+&a\_{2n}x\_{n}&=&b\_{2}\\&&&\vdots &&&&&\vdots \\a\_{n1}x\_{1}&+&a\_{n2}x\_{2}&+&\cdots &+&a\_{nn}x\_{n}&=&b\_{n}\end{array}}\right.\quad \Rightarrow \quad \overbrace {\begin{bmatrix}a\_{11}&a\_{12}&\cdots &a\_{1n}\\a\_{21}&a\_{22}&\cdots &a\_{2n}\\\vdots &\vdots &\ddots &\vdots \\a\_{n1}&a\_{n2}&\cdots &a\_{nn}\end{bmatrix}} ^{\underline {A}}\overbrace {\begin{Bmatrix}x\_{1}\\x\_{2}\\\vdots \\x\_{n}\end{Bmatrix}} ^{\underline {x}}=\overbrace {\begin{Bmatrix}b\_{1}\\b\_{2}\\\vdots \\b\_{n}\end{Bmatrix}} ^{\underline {b}}}

בשיטות האנליטיות מבצעים פעולות שורה אלמנטריות על המטריצה לקבל הפתרון הרצוי, אשר תלוי בדיוק המחשב עליו מתבצע החישוב. נקודת התורפה המרכזית של השיטות האנליטיות היא גודל המערכת המרבי בו ניתן לטפל.

**קירוב ליניארי:**

קירוב ליניארי או קירוב מסדר ראשון הוא מושג במתמטיקה המתאר קירוב של פונקציה מתמטית כלשהי באמצעות פונקציה ליניארית (ליתר דיוק, פונקציה אפינית). לקירובים ליניארים יש שימוש נרחב במדעים ובמתמטיקה כדי לקבל קירוב לערך הפונקציה בסביבה של ערך קבוע מראש. היות שפונקציות ליניאריות הן קלות לחישוב ולפתרון, קירובים ליניארים מועדפים כמעט תמיד בניתוחים אנליטיים ונומריים אם הם מספקים את הדיוק הנדרש.

כאשר לפונקציה קיים קירוב ליניארי, נאמר שהפונקציה דיפרנציאבילית.

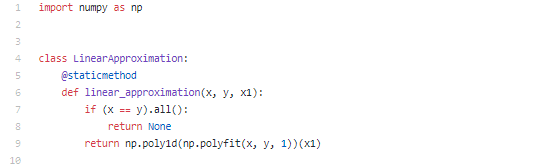
הבעיה היא שאנחנו לא יודעים כלום על רוב הפונקציות. ואנחנו יודעים די הרבה דברים על פונקציות לינאריות (אם יש נוסחה של פונקציה קווית - יודעים מה השיפוע, מה נקודות החיתוך על הצירים, ובעצם מהנוסחה יודעים איך לשרטט את הגרף)

אז מה שעושים זה מקרבים - מתקרבים לפונקציה שאנחנו לא מכירים בכלל בעזרת קירובים לינאריים

קוד נלקח מאתר scipy

קישור לקוד :

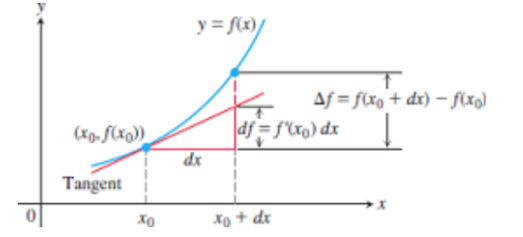
<https://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html>



קלט: מערך של X מערך של Y וסוג האינטרפולציה

פלט: גרף של פונקציה ע"פ הנקודות

קירוב ליניארי דיאגרמה לנכונות הקירוב:



**קירוב פולנומיאלי תוך שימוש במטריצת ון דר מונדה:**

קוד נלקח מהאינטרנט

קישור לקוד:

<https://github.com/yanivbenzvi/numerical-analysis1/blob/master/lib/polynomialAprox_method.py>

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

קלט :אוסף של נקודות X ו Y ופונקציה שעליה

פלט: הדפסה של וקטור המקדמים של קירוב הפולינום.

הסבר של הקוד:

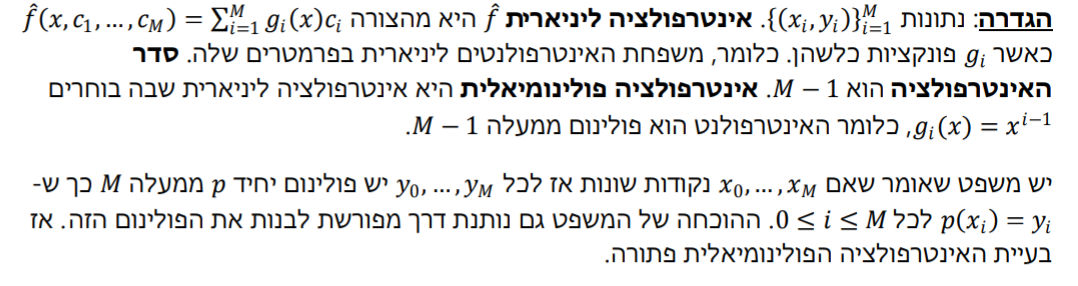
יצירת מטריצת מקדמים על פי ה איקסים שקיבלנו כקלט, יצירת מטריצת ון דר מונדה

מציאת המטריצה ההופכית

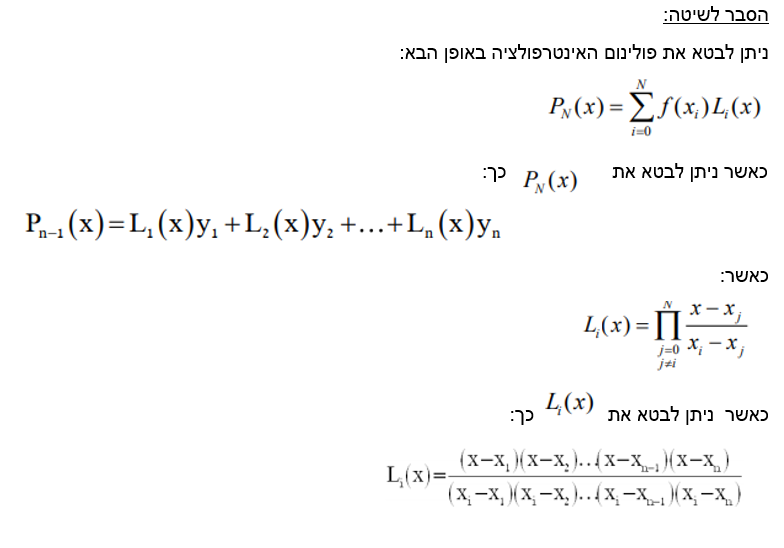
וקטור המקדמים של הפולינום שאותו אנחנו מחפשים הינו כפל של וקטור ערכי ה Y שקיבלנו

כפול המטריצה ההופכית והדפסת התוצאה

אינטרפולציה:



**אינטרפולצית לגראנז':**

**תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי**

קוד נלקח מאתר trinket

קישור לקוד :

<https://trinket.io/python/bccab55640>

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

קלט: מערך של X ו Y ומספר האיטרציות U, X[i] הוא הערך של X בנקודה ה iית

Y[i] הערך של הפונקציה ב X[i]

פלט: ערך משוערך וערך מדוייק של הפולינום

הסבר הקוד:

נאתחל משתנים ועבור כל נקודה נגדיר פולינום בסיס בלולאה פנימית ניצור את המכנה ואת המונה ונכפול בערך שהתקבל מהמונה והמכנה בפולינום של השלב הקודם

ונבצע חיבור של האינטרפולצית לגראנז'

יודפס הערך המדוייק והערך המשוערך(המקורב)

**שיטת נוויל:**

בד"כ לא רצוי לחבר מספרים מסדרי גודל שונים כי הדיוק של החישוב נפגע מאוד בגלל אופן הייצוג של המספרים במחשב. בפרט, הבעיה חמורה מאוד כאשר סדר האינטרפולציה גבוה ואז עלול להיות הבדל מאוד גדול בין 

הרעיון הוא לבנות את הפולינום בשלבים ובכל פעם לעשות חישובים על מספרים בסדרי גודל דומים.

תמונה שמכילה טקסט, מפה

התיאור נוצר באופן אוטומטי

תמונה שמכילה אובייקט, צילום מסך

התיאור נוצר באופן אוטומטי

קוד נלקח מאוניברסיטת קולומביה שבאוהיו נכתב ע"י Ningchuan Xiao

קישור לקוד:

<https://github.com/yanivbenzvi/numerical-analysis1/blob/master/lib/Neville.py>

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

קלט: ערך של X לאינטרפולציה , מערך של Xים ומערך של Yים

פלט: פולינום ממעלה N

הסבר: מעבר בצורה רקורסיבית על איברים ב שתי לולאות מקוננות עוברים על איברים בפעם הראשונה מאתחלים את מערך הפולינומים בערכי ה Y שקיבלנו

בשאר האיטרציות הכנסנו ערכים לפי נוסחאת נוויל

ובסוף קראנו לפונקציה שנשתמש בה להדפסת איטרציות ותוצאות.

**אינטרפולציות פולינומיאליות למקוטעין – Splines**

תמונה שמכילה שולחן, מקורה

התיאור נוצר באופן אוטומטי

* בין כל שני פולינומים (מקטעים) נדרוש:

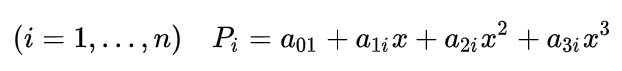
- רציפות בנקודת החיבור. בין כל שני פולינומים (מקטעים) נדרוש:

- רציפות בנקודת החיבור.

- רציפות הנגזרת בנקודת החיבור.

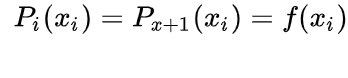
- נגזרת שניה רציפה.

עבור spline קובי: נתונות n+1 נקודות.

- הפולינום: 

- יש לנו 4n נעלמים (מספר הפולינום) ולכן דורשים 4n משוואות.

- המשוואות: נדרוש שהפולינום יעברו דרך הנקודות הנתונות ונדרוש רציפות:



תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

קלט: מערך Xים ו Yים אשר מציג את ערכי Xi ושל Yi בנקודה i

פלט: ערך הפולינומים שקיבלנו בין 2 נקודות ומחזיר איחוד של כל הפונקציות

הסבר לקוד:

3 פונקציות

1. generateSplines שמקבלת X,Y ומחזירה פולינום ממעלה 3
2. f(x) שמקבלת x מסויים ומחזירה קירוב
3. createPoly מקבלת מקדמים ומחזירה אובייקט של פונקציה

**שיטת הטרפז:**

שיטת הטרפז היא שיטת חישוב הנותנת קירוב לערכו של שטח כלוא או לערכו של אינטגרל מסוים על פונקציה בין שני ערכים ללא ביצוע פעולת האינטגרל כלל.

בשיטה זו אנו נחלק את הפונקציה שלנו למספר אינטרוולים כך שבכל אינטרוול אנו נחשב שטח של טרפז, אשר נותן קירוב לשטח האמתי המיוצג ע"י האינטגרל של אותו אינטרוול.

ובעצם כך יחושב השטח ע"י הנוסחה הבאה:

תמונה שמכילה אובייקט, שעון

התיאור נוצר באופן אוטומטי

קוד נלקח מGithub

קישור לקוד :

<https://github.com/minwoobae/python_numerical_method/blob/master/pynumerical.py>

תמונה שמכילה טקסט, צילום מסך

התיאור נוצר באופן אוטומטי

קלט : הפונקציה , טווח שעליו אנחנו רוצים לבצע את החישוב , ומספר האינטרוולים.

פלט: מאיזה שיעור האינטגרל של הפונקציה

הסבר הקוד: ראשית הפונקציה תבדוק אם מספר האיטרוולים שקיבלה גדול מ-0 לאחר מכן תבדוק אם מספר האיטרוולים זוגי במידה ולא לא נוכל להמשיך והפונקציה תעצר.

נקבל שרוחבו של כל טרפז הוא h = (b-a)/n כאשר a,b הם התחומים ו-n זהו מספר האינטרוולים.

הפונקציה תרוץ על פי הנוסחא ותעצור לאחר מספר האנטרוולים שקיבלה.

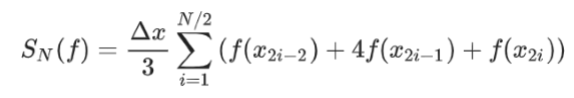
**שיטת סימפסון:**

הגדרה:

חוק סימפסון משתמש בפולינום ריבועי על כל תת ערך של הפונקציה על מנת להעריך את הפונקציה ולחשב את האינטגרל המוחלט.

זהו שיפור של שיטת הטרפז אשר מעריכה את הפונקציה על ידי קו ישר על כך תת ערך של הפונקציה.

הנוסחה של שיטת סימפסון היא:



N מספר זוגי של תת ערכים [a,b] כך ש:



הקוד נלקח מאתר Mathematical Python

קישור לקוד :

[**https://www.math.ubc.ca/~pwalls/math-python/integration/simpsons-rule/**](https://www.math.ubc.ca/~pwalls/math-python/integration/simpsons-rule/)

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

קלט: פונקציה , 2 אינטרוולים לאינטגרציה , מספר של תת אינטרוולים

פלט: חישוב השטח לפי נוסחאת סימפסון

הסבר על הקוד:

קירוב האינטגרל של הפונקציה בדיקה ש התת אינטרוולים הינם זוגיים לפי חוק סימפסון

הדפסת ערך Dx

קירובים וחישוב הקירוב של האינטגרל עם הערכים שהתקבלו לפי נוסחאת סימפסון

**שיטת רונגה קוטה:**

נעבוד עם המקטעים

i ∈ N , xi= h i + x0

נתרכז במציאת פתרון לבעיה y ′ ( x ) = f ( x , y )

עם תנאי ההתחלה y ( x0 ) = y0

שיטת אוילר היא שיטה מספרית לפתרון משוואה דיפרנציאלית ראשונה של סדר ראשון עם ערך התחלתי נתון. זה הכי בסיסי בשיטה מפורשת עבור אינטגרציה נומרית משוואות דיפרנציאליות רגילות ו הוא פשוטה שיטת ראנגה-קוטה

הקוד נלקח מהאינטרנט

קישור לקוד :

<http://github.com/minwoobae/python_numerical_method/blob/master/pynumerical.py>

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי