תרגיל בית מס' 1

מרצה:

ד"ר ויקטור צ'רנוב

מגישים:

יהונתן כשאני – 307891572

ויליאם עביד – 322509621

תאריך הגשה:

09/12/2025 – י"ט כסלו, ה'תשפ"ו

תיאור הבעיה

בתרגיל זה נפתרת באופן נומרי פונקציה המבוססת על פונקציות בסל מן הסוג הראשון, הפותרות את המשוואה הדיפרנציאלית הבאה:

$$x^2\,\frac{d^2y}{dx^2}\,+\,x\,\frac{dy}{dx}\,+\,(x^2-\alpha^2)\;=\;0$$

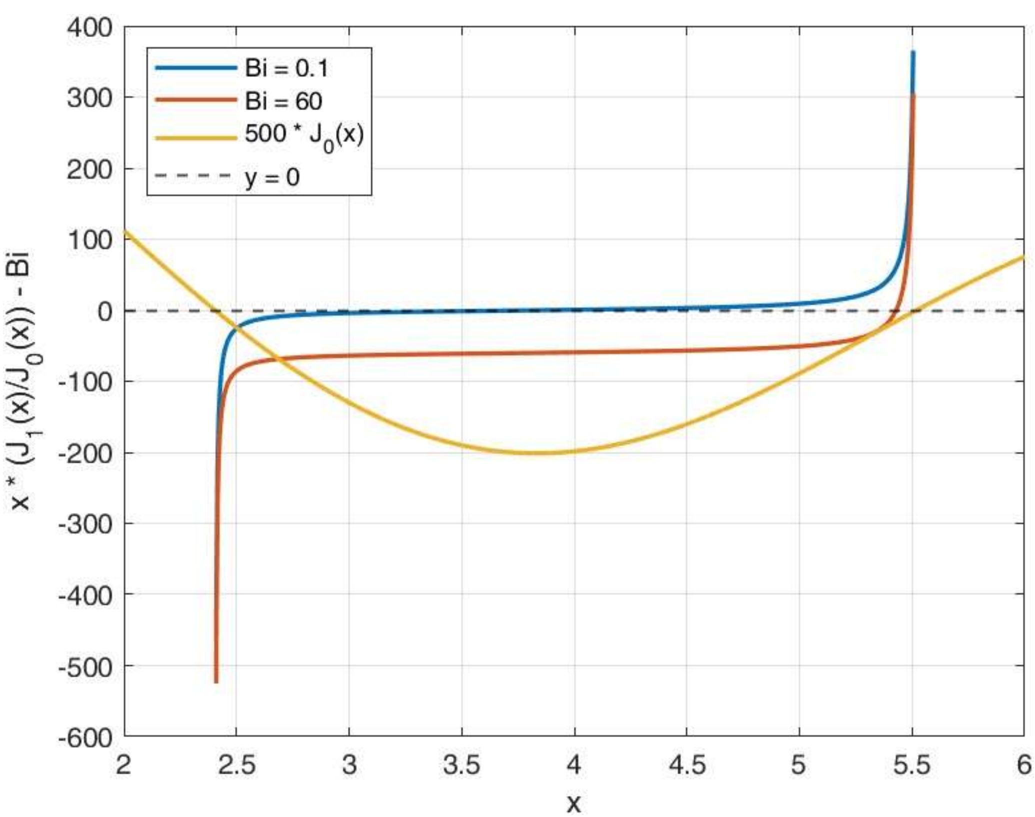
הפונקציה הנחקרת היא:

$$x\,\frac{J_1(x)}{J_0(x)}\;=\;Bi$$

והטווח הרלוונטי הוא $0.1 \le Bi \le 60$.

מטרת התוכנה היא לזהות פתרונות לפי מספרם הסידורי, עבור ערך נתון של Bi. התוכנה מקבלת כקלט את ערך ה-Bi ואת מספר הפתרון, ומחזירה את ערך ה-x המתאים.

בגרף המצורף ניתן לראות כי צורת הפונקציה כמעט ואינה משתנה עם שינוי ב-Bi, אלא עוברת הסטה קלה כלפי מעלה או כלפי מטה.

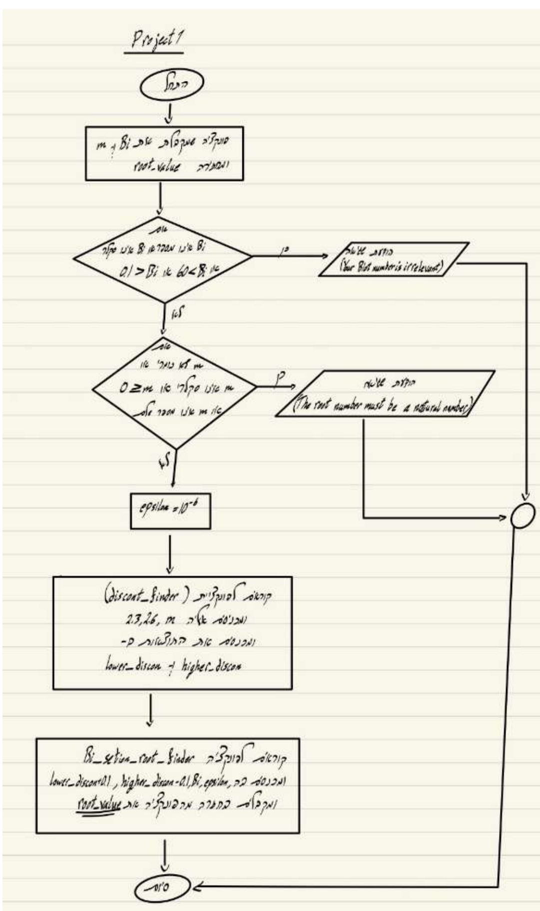


שיטת פתרון

היות שיש חשיבות מיוחדת לשמירה על הסדר של השורשים בפתרון, נבחר בשיטת החציה: לפונקציה יש נקודת אי-רציפות בה הסימן שלה עובר משלילי לחיובי: כל שורש של $J_0(x)$. כדי לא לזהות אותן בשוגג כנקודות איפוס (הרי שהסימן מצדדי נקודת אי-הרציפות שונה), נזהה אותן ונשתמש בהן כגבולות טווח החיפוש עבור כל שורש – השורש ה-n של הפונקציה שלנו יימצא בוודאות בין השורש ה-$n-1$ והשורש ה-n של $J_0(x)$.

נתחיל בפונקציה פשוטה שמיישמת את שיטת החצייה ל-$J_0(x)$, ונשתמש בשורשי הפונקציה שהיא החזירה כדי לתחום את אזור החיפוש שלנו אחרי השורש של הפונקציה הנחקרת. נשתמש במספרים הקרובים לגבולות התחום אך בתוכו כניחושים ראשוניים (ידוע לנו שאף אם נציב את מספר הביוט הגבוה ביותר המותר, שורש הפונקציה רחוק יותר מ-0.1 מנקודת אי-הרציפות), ונפעיל את שיטת החצייה למציאת שורש הפונקציה – נציב את ממוצע הניחושים בפונקציה, נבחן מה הסימן שלו ביחס לסימני ערכי הפונקציה בניחושים הראשוניים, ונחליף את הניחוש הראשוני בעל ערך הפונקציה באותו הסימן בממוצע הניחושים הראשוניים. נחזור על התהליך עד שנקבל ערך אקס אשר הצבתו בפונקציה הנחקרת ייתן ערך הרחוק כדי אפסילון מהאפס. ערך זה הפונקציה project1 תחזיר.

<u>תרשימי זרימה לפונקציית project1:</u>



שיטת ניוטון-רפסון:

השיטה בבסיסה פועלת כך – מנחשים x_0, מציבים בפונקציה ובנגזרת, ויוצרים משיק החותך את ציר האיקס:

$$< \mathrm{x}_0, f(x_0) >, < x_1, 0 >$$

הנגזרת בנקודה x_0 היא שיפוע הקו ולכן:

$$\frac{f(x_0)-0}{x_0-x_1} = f'(x_0)$$

ומכאן:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

נגדיר פונקציה חדשה: $f(x) = x\frac{J_1(x)}{J_0(x)} - Bi$

ידוע שגזירה של פונקציית בסל מסוג ראשון נראית כך: $\frac{dJ_0(x)}{dx} = -J_1(x)$ וכאשר $\alpha \geq 1$,

$$\frac{dJ_\alpha(x)}{dx} = \frac{J_{\alpha-1}(x) - J_{\alpha+1}(x)}{2}$$

ולפיכך:

$$f'(x) = \frac{J_1(x)}{J_0(x)} + x\frac{0.5J_0^2(x) - 0.5J_0(x)J_2(x) + J_1^2(x)}{J_0^2(x)}$$

$$f'(x) = \frac{J_1(x)}{J_0(x)} + x\frac{J_1^2(x)}{J_0^2(x)} + \frac{x}{2} - \frac{J_2(x)}{2J_0(x)}$$

$$\frac{f(x)}{f'(x)} = \frac{x\frac{J_1(x)}{J_0(x)} - Bi}{\frac{J_1(x)}{J_0(x)} + x\frac{J_1^2(x)}{J_0^2(x)} + \frac{x}{2} - \frac{J_2(x)}{2J_0(x)}}$$

ומכאן נסיק בקלות במקרה שלנו:

$$x_n = x_{n-1} - \frac{x\frac{J_1(x)}{J_0(x)} - Bi}{\frac{J_1(x)}{J_0(x)} + x\frac{J_1^2(x)}{J_0^2(x)} + \frac{x}{2} - \frac{J_2(x)}{2J_0(x)}}$$

בדומה, אם נשתמש בשיטת המיתר (אני אוותר על הפיתוח הפעם), ננחש x_0, x_1 ונחשב $f(x_0), f(x_1)$:

$$x_2 = x_1 - f(x_1)\frac{(x_1 - x_0)}{f(x_1) - f(x_0)}$$

אם נציב אן ואת ערך הפונקציה נקבל:

$$x_{n+1} = x_n - \left(x_n\frac{J_1(x_n)}{J_0(x_n)} - Bi\right)\frac{(x_n - x_{n-1})}{\left(x_n\frac{J_1(x_n)}{J_0(x_n)} - Bi\right) - \left(x_{n-1}\frac{J_1(x_{n-1})}{J_0(x_{n-1})} - Bi\right)}$$

$$x_{n+1} = x_n - \left(x_n\frac{J_1(x_n)}{J_0(x_n)} - Bi\right)\frac{(x_n - x_{n-1})}{\left(x_n\frac{J_1(x_n)}{J_0(x_n)}\right) - \left(x_{n-1}\frac{J_1(x_{n-1})}{J_0(x_{n-1})}\right)}$$

$$x_{n+1} = \frac{x_n\left(x_n\frac{J_1(x_n)}{J_0(x_n)}\right) - \left(x_{n-1}\frac{J_1(x_{n-1})}{J_0(x_{n-1})}\right) - \left(\left(x_n\frac{J_1(x_n)}{J_0(x_n)} - Bi\right)x_n - \left(x_n\frac{J_1(x_n)}{J_0(x_n)} - Bi\right)x_{n-1}\right)}{\left(x_n\frac{J_1(x_n)}{J_0(x_n)}\right) - \left(x_{n-1}\frac{J_1(x_{n-1})}{J_0(x_{n-1})}\right)}$$

$$x_{n+1} = \frac{((-Bi)x_n - (-Bi)x_{n-1})}{\left(x_n\frac{J_1(x_n)}{J_0(x_n)}\right) - \left(x_{n-1}\frac{J_1(x_{n-1})}{J_0(x_{n-1})}\right)} = Bi\frac{x_{n-1} - x_n}{\left(x_n\frac{J_1(x_n)}{J_0(x_n)}\right) - \left(x_{n-1}\frac{J_1(x_{n-1})}{J_0(x_{n-1})}\right)}$$

ואחרונה, שיטת החציה:

$$x_{low}, x_{high} => x_{new} = \frac{x_{high} + x_{low}}{2} => f(x_{new}) * f(x_{low}) < 0$$

במידה ואי-השוויון מתקיים, מחליפים את האיקס הגבוה באיקס החדש, וחוזרים על התהליך.

קל לראות שבעוד ששיטת החצייה משאירה את ערכי האיקס בטווח, שיטת נ"ר ושיטת המיתר מאפשרות איקסים מחוץ לטווח, לפעמים באופן משמעותי, מה שמגביל את השימוש בהן, ומאלץ להשתמש בבדיקות משניות, מלבד הבדיקה הראשית שמתרחשת בכל איטרציה בכל שיטה (קרבת ערך הפונקציה באיקס החדש לאפס $|f(x_n)| < \varepsilon$).

<u>השיטה הנומרית:</u>

בתוך התוכנות יושמו שלושת השיטות שלמדנו לטובת מציאת שורשים: שיטת החצייה, ניוטון-רפסון והמיתר. בקוד שלנו יישמנו את שיטת החצייה פעמיים, פעם אחת כדי לזהות את שורשי $J_0(x)$ (בשל טבעה האוסילטורי של הפונקציה, השתמשנו בצעד בגודל בערך פאי כדי לזהות את השורש הבא) ברמת דיוק נמוכה ופעם שנייה כדי לקבל את השורש של הפונקציה הנחקרת ברמת דיוק מספקת.

התחלנו בניחושים ראשוניים שהתבססו על גרף הפונקציה, וידאנו סימנים מנוגדים, וכיון שפונקציית בסל רציפה, משפט ערך הביניים מתקיים וניתן להסיק שקיים שורש ביניהם. אחרי כל שורש שמצאנו דילגנו פאי וקצת (בחרנו 0.1, היות שזה הספיק לרווח בין השורשים הראשונים, וההפרש בין שורשי-פונקציית בסל שואף לפאי) קדימה כניחוש גבוה, ושתי אפסילון קדימה כניחוש נמוך (המרחק המקסימלי מהשורש הקרוב הוא אפסילון, אז שני אפסילון בוודאות יעבירו אותנו לצידו הימני), וידאנו סימנים מנוגדים והתחלנו מחדש. כשהגענו לשורש ה-m, ידענו כי שתי השורשים של פונקציית הבסל שיש בידינו הם אי-הרציפויות המקיפות את השורש ה-m של הפונקציה הנחקרת, צמצמנו את הטווח באופן מינימלי (השתמשנו בערכי הקצה של מספרי הביוט כדי להחליט מה הערך הקרוב ביותר לאי-הרציפות בו הפונקציה משנה סימן) והתחלנו לחפש את שורש הפונקציה הנחקרת בטווח – מצאנו איקס חדש ע"י מיצוע חשבוני של ערכי האיקס הקיימים (התחלנו בגבולות הטווח וצמצמנו), הצבנו אותו בפונקציה וצמצמנו את הטווח כך שהוא יהיה אחד הגבולות, כאשר ערך הפונקציה בגבול השני הפוך ממנו בסימנו; חזרנו על הפעולות האחרונות עד שקיבלנו איקס שערך הפונקציה הנחקרת בו קרוב כדי אפסילון לאפס. היות ששיטת החצייה מגבילה באופן אינהרנטי פיזור של ניחושים (כל ממוצע חשבוני בין נקודות יניב נקודה ביניהן), השתמשנו בתנאי-עצירה יחיד, והוא קרבה אפסילונית של ערך הפונקציה בנקודה לאפס.

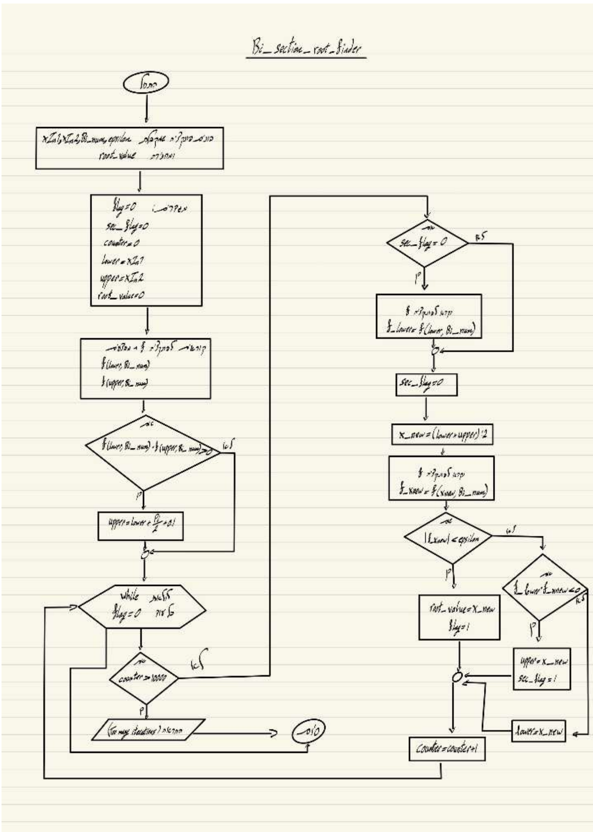
הפונקציה של ה-AI הפעילה שיטה שונה למדי: את הפונקציה המקורית היא המירה לפונקציה עם אותם שורשים אבל בלי אי-רציפויות:

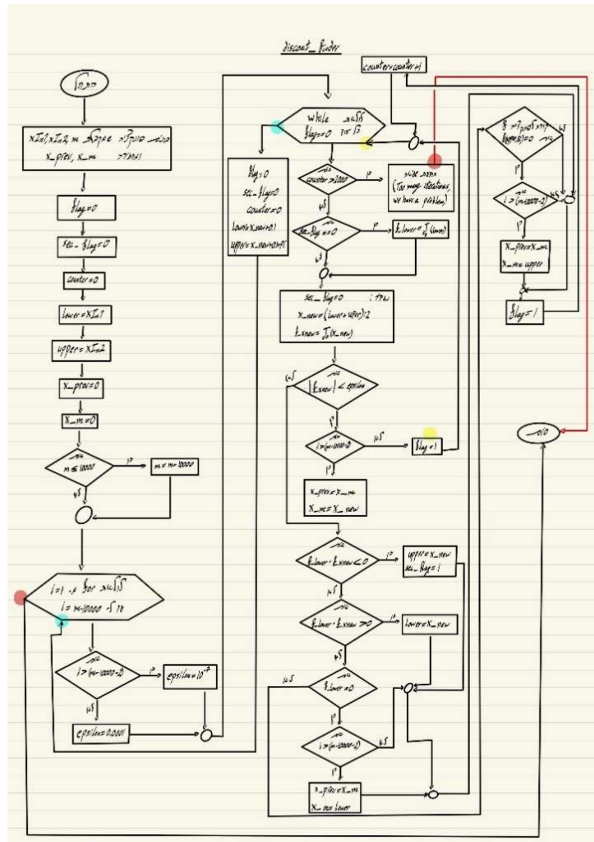
$$g(x) = xJ_1(x) - Bi * J_0(x)$$

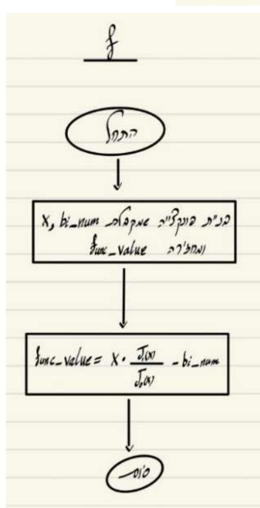
ועברה שיטתית על כל האיקסים (בצעדים של $\frac{\pi}{20}$) כשהיא מזהה כל שינוי סימן בערך הפונקציה בנקודה. כל שינוי סימן כזה הפעיל פונקציה של שיטת המיתר – שני האיקסים שהיו מעורבים בהחלפת הסימן (לפני ואחרי השורש, בעצם) משמשים כנקודות העוגן, ומיתר מועבר ביניהם, ושומרים את השורש של אותו המיתר כניחוש הראשון של שיטת ניוטון-רפסון (בהתחשב בכך שערך הפונקציה משנה סימן ביניהן, השורש תמיד יתקבל בין הנקודות). בשיטת ניוטון-רפסון, משתמשים בנגזרת הפונקציה בנקודה כדי לקבל משיק, ומשתמשים בשורש המשיק בתור

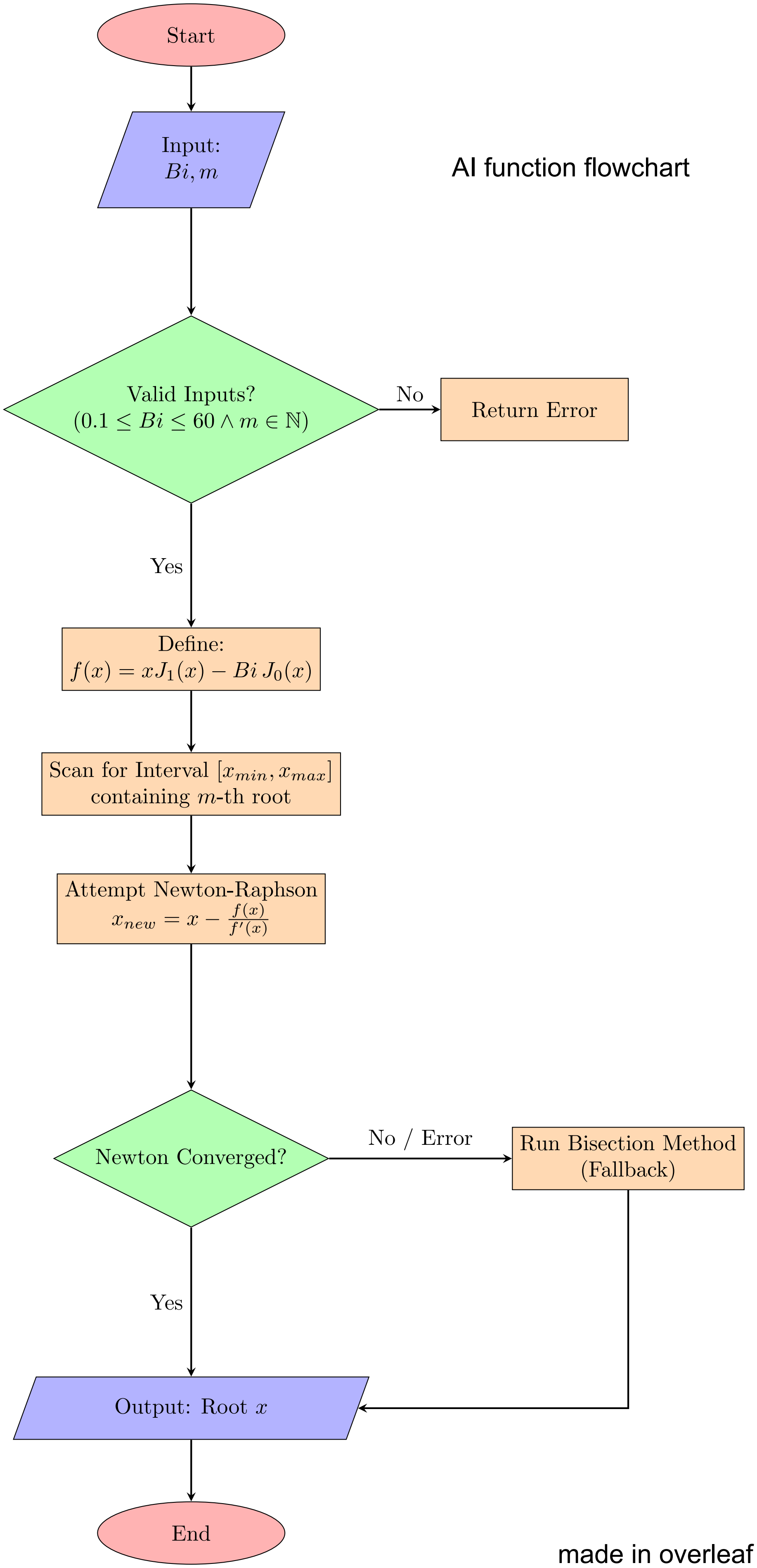
האיקס הבא. בשלב זה הפונקציה של ה-AI משתמשת בגידור כדי למנוע מהניחוש הבא לצאת מהטווח המקורי (של $\frac{\pi}{20}$), ובוחנת כל ערך פונקציה בנקודה איקס חדשה לקרבה מספקת לאפס. במידה והניחוש הבא של ניוטון-רפסון חורג מהטווח או שערך הנגזרת קטן מדי (ולכן הניחוש הבא רחוק מדי), הפונקציה זורקת שגיאה ועוברת להשתמש בשיטת החציה ברציונל דומה לזה של הפונקציה שלנו.

<u>תרשימי זרימה של הפונקציות ששימשו בפונקציה project1:</u>








Start
Input:
Bi, m
AI function flowchart
Valid Inputs?
$(0.1 \leq Bi \leq 60 \wedge m \in \mathbb{N})$
No
Return Error
Yes
Define:
$f(x) = xJ_1(x) - Bi\,J_0(x)$
Scan for Interval $[x_{min}, x_{max}]$
containing m-th root
Attempt Newton-Raphson
$x_{new} = x - \frac{f(x)}{f'(x)}$
Newton Converged?
No / Error
Run Bisection Method
(Fallback)
Yes
Output: Root x
End
made in overleaf

<u>תוצאות:</u>

תוצאות נוספות ימצאו בשני קבצי ה-txt המצורפים, עם נתוני המהירות של שתי הפונקציות בהרצות אקראיות כמפורט.

Bi	m	x
24.8	1	2.3101
24.8	2	5.3052
22.7	3	8.2961
23	4	11.3253
0.5	5	13.3611
10	6	17.0099
12	7	20.1594
50	8	23.9026
58.7	10	30.1566
34	20	60.98
15	30	92.0492
7	40	123.3612
9	50	154.7791
11.3	60	186.198
36	70	217.7175
59.7	80	249.205

<u>דיון ומסקנות:</u>

למרות הניסיון המוקדם שלנו להשתמש בשלושת השיטות, שני פקטורים הטו את הכף לטובתה של שיטת-החצייה: דרישת הסף של התוכנה הייתה ספירה מדויקת של המספר השורשים הקטן מזה שאנו מחפשים, ודבר זה דורש שליטה מהודקת יותר באיקס ומכאן שהדבר יעלה בזמן-ריצה במידה וננסה להשתמש בשיטה בה השליטה באיקס לא אינהרנטית; הפקטור השני הוא ששתי השיטות המבוססות על שיפועי-מיתרים רגישות ל"מישורים" – טווח איקסים בו השיפוע מינימלי, ולכן כל שימוש בשיטת ניוטון-רפסון לדוגמא יגרור הפרשים מאד גדולים בין האיקס הראשוני לחדש (מה שהתבטא במהלך הניסיונות המוקדמים שלנו באיקסים שליליים וגדולים בשניים-שלושה סדרי-גודל מהרצוי) או ידרוש שימוש במגבילים חזקים שיסכלו כל תקווה להאיץ את תהליך ההכנסות של הפונקציה ביחס לשימוש פשטני בהרבה שיטת החצייה. למרות שנראה שהקוד יעיל מספיק כשהוא מנצח את ה-AI [הערה קטנה – מבחני הביצועים של הקוד היו מעט לא הוגנים, היות שהטולרנס של הקודים שונה בשני סדרי גודל. כעת פשוט מאוחר מכדי להריץ את כולם שוב], סביר שניתן לשפר את יעילות הקוד עם שליטה מהודקת יותר בפרמטרים (גודל צעד מדויק יותר), לעשות אותו רובוסטי יותר עם יותר מבחני-כשל ופעולה בהתאם במקום כישלון שנזרק למשתמש ולהבין את המתמטיקה של הפונקציה טוב יותר כדי להיות ודאיים בנוגע לאוסילציות. כמו כן, סביר שניתן לשפר את הקוד של ה-AI מעבר למה שעשינו, ולהתאים אותו הרבה יותר לשורשים גדולים מ-1, השורש היחידי בו הוא באמת מהיר.

השוואה בין הפתרון שלנו לפתרון ה-AI:

השתמשנו בעיקר ב-Gemini 3 Pro Thinking (לטובת הכתיבה עצמה של הקוד) ומעט ב-ChatGPT במודל GPT 5.1 (בעיקר לצורך בחינה של הקוד ושיפור שלו).

https://gemini.google.com/share/56b4fda1cefc

https://chatgpt.com/share/69360c94-d63c-8004-9617-6796e9e021db

https://gemini.google.com/share/d7255ff2f13d

ביצועי הריצה של הפונקציה שקיבלנו מה-AI היו טובים יותר ב-m נמוכים, וגרועים יותר ככל שה-m עלה (מצורף קובץ TXT עם 4 תוצאות מבחני-ריצה), אבל הפתרונות נותרו זהים (עד כדי אפסילון).

<u>רשימת קבצים:</u>

<u>החלק העצמאי בפרויקט:</u>

project1.m – קובץ הפרויקט, מקבל מספר ביוט ומספר טבעי *n*, ומחזיר את השורש ה-*n* של הפונקציה.

f.m – יישום הפונקציה בתוכנה. מסוגלת לקבל וקטורים, לטובת בדיקות במהלך העבודה. מקבל מספר ביוט ואקס, ומחזירה את ערך הפונקציה בנקודה.

Discont_finder.m – פונקציה המיישמת את שיטת החציה למציאת שורשים של $J_0(x)$, ומחזירה נקודות הקרובות כדי דלתא של אפסילון לנקודת אי-הרציפות של הפונקציה הנחקרת. מקבלת שני ניחושים ראשוניים ומספר טבעי *n*, ומחזירה את השורשים ה-*n-1* וה-*n* של $J_0(x)$.

Bi_section_root_finder.m – פונקציה המיישמת את שיטת-החציה למציאת שורשי-הפונקציה הנחקרת, ומחזירה נקודה הקרובה כדי דלתא של אפסילון לשורש הפונקציה הנחקרת. מקבלת שני ניחושים ראשוניים, מספר ביוט ואפסילון.

speed_check.m – סקריפט פשוט לבדיקת מהירות הפונקציה הראשית, מייצר שני וקטורים בגודל 1000 של מספרים רנדומליים היכולים לשמש כפרמטרים של הפונקציה הראשית, ואז מזינים אותם לפונקציה כך שתרוץ 1000 פעמים על מספרי ביוט רנדומליים ותחפש עד השורש ה-10000 של הפונקציה הנחקרת. תוצאות 5 הרצות שונות של הסקריפט מופיעות בקובץ *TXT* מצורף.

AI_testing_script.m – נועד לבדיקת התוכנה באופן עקבי עם הבדיקות של פונקצית ה-*AI* – מריץ בדיקות עם מספרי ביוט ומוני-שורש אקראיים, בודק מה המרחק בין ערך הפונקציה בנקודה לאפס ואת הקרבה למכפלה של פאי (מחזור הפונקציה).

<u>החלק בפרויקט שנכתב ע"י *AI*:</u>

SolveBesselAI – הפונקציה העיקרית, ותחת קובץ זה יש את שאר הפונקציות המשמשות את הפונקציה העיקרית (מלבד אחת). הפונקציה מקבלת מספר ביוט ומספר מונה *m*, ומחזירה את השורש ה-*m* של הפונקציה עם מספר הביוט הנתון. הפונקציה משתמשת בצעד קבוע כדי לעבור על האיקסים ולזהות שינוי סימן, אחרי *m* שינויי סימן, משתמשת בשיטת המיתר כדי לנחש פתרון ראשוני (משתמשת באיקסים שהתקבלו משינוי הסימן האחרון), ומפעילה מאותו ניחוש ראשוני את שיטת ניוטון-רפסון, עם גיבוי למקרה של התבדרות או חריגה מהטווח המקורי בצורת פונקציה של שיטת החציה

bessel_func_product – מקבלת מספר ביוט ואיקס, ומחזירה את הערך בנקודה וערך הנגזרת בנקודה.

run_newton – מקבלת ניחוש ראשוני, טולרנס, "הנדל" של הפונקציה הנחקרת, גבולות טווח ומספר איטרציות מקסימלי (בפונקצית האם הוא נגזר מגודל ה-*m*), ומחזירה איקס שערך הפונקציה בנקודה שלו הוא במרחק אפסילון ("טולרנס") מהאפס, ומספר איטרציות. יישום שגרתי של ניוטון-רפסון, מפורט בשלב השיטה הנומרית.

run_bisection – מקבלת גבולות טווח לחיפוש, טולרנס, מספר איטרציות מקסימלי ו"הנדל" של הפונקציה הנחקרת, ומחזירה איקס שערך הפונקציה בנקודה שלו הוא במרחק אפסילון ("טולרנס") מהאפס, ומספר איטרציות. יישום שגרתי של שיטת החציה, מפורט בשלב השיטה הנומרית.

comp_script – הוכחת היכולת של הקוד. סקריפט שמפעיל את הקוד של ה-*AI* ולצידו את הקוד שלנו על 6 מקרים שונים ומציג את ההבדלים בתוצאות (סדר גודל של 10^{-8}), ואז מריץ את הקודים 10000 על פרמטרים אקראיים ומשווה מהירות ריצה. מייצר את הקובץ *comparison_results.txt*

speed_test_AI – סקריפט בדיקת מהירות זהה ל-*speed_test*, פשוט מקומי לתיקייה אחרת.

<u>קבצי טקסט:</u>

PERFORMANCE SPEED TEST.TXT – תוצאות של 5 הרצות שונות של סקריפט ההשוואה, המראות איך השינוי בגודל ה-*m* משפיע על מהירות הפונקציה לרעה בהרבה יותר במקרה של פונקציית ה-*AI* לעומת הפונקציה שלנו

comparison_results.TXT - הדפסה ישירה של ההפעלה האחרונה של סקריפט *comp_script*.

The next 3 pages are the txt files with the results that were reffered to in the legend, in reverese order

```
==============================================================
               PROJECT 1: SOLVER COMPARISON
==============================================================

--- PART 1: ACCURACY & ROOT LOCATION ---
Bi     | m     | x (Manual)    | x (AI)        | Diff
--------------------------------------------------------------
  24.8 |     1 |     2.310059 |      2.310059 |   2.21e-09
  24.8 |     2 |     5.305160 |      5.305160 |   3.50e-09
  24.8 |    57 |   176.851841 |    176.851841 |   5.32e-09
   0.5 |    17 |    51.053330 |     51.053329 |   1.65e-08
  60.0 |    87 |   271.178817 |    271.178817 |   1.67e-09
  10.0 |   149 |   465.761773 |    465.761773 |   3.34e-10


--- PART 2: PERFORMANCE SPEED TEST (10,000 Runs) ---
Benchmarking for Bi=22.6, m=78...
Manual Code Total Time:    8.4477 seconds
AI Code Total Time:       37.6921 seconds
Speedup Factor:          0.22x faster
```

```
--- PART 2: PERFORMANCE SPEED TEST (100,000 Runs) ---
Benchmarking for Bi=60.0, m=1...
Manual Code Total Time:    9.2955 seconds
AI Code Total Time:        5.0621 seconds
Speedup Factor:          1.84x faster

--- PART 2: PERFORMANCE SPEED TEST (100,000 Runs) ---
Benchmarking for Bi=58.7, m=2...
Manual Code Total Time:   15.8030 seconds
AI Code Total Time:       14.4358 seconds
Speedup Factor:          1.09x faster

--- PART 2: PERFORMANCE SPEED TEST (100,000 Runs) ---
Benchmarking for Bi=0.4, m=3...
Manual Code Total Time:   16.7616 seconds
AI Code Total Time:       17.0403 seconds
Speedup Factor:          0.98x faster

--- PART 2: PERFORMANCE SPEED TEST (100,000 Runs) ---
Benchmarking for Bi=43.8, m=8...
Manual Code Total Time:   24.3076 seconds
AI Code Total Time:       55.9455 seconds
Speedup Factor:          0.43x faster


--- PART 2: PERFORMANCE SPEED TEST (100,000 Runs) ---
Benchmarking for Bi=9.4, m=47...
Manual Code Total Time:   68.4326 seconds
AI Code Total Time:      234.5463 seconds
Speedup Factor:          0.29x faster
```

```
--------------------speed_check.m-------------------

x = randi([1 600], 1, 1000)/10;
y = randi(10000, 1, 1000);

tic;
for i = 1:1000
    project1(x(i), y(i));
end
toc;
```

```
>> speed_check
Elapsed time is 25.049200 seconds.
>> speed_check
Elapsed time is 28.300633 seconds.
>> speed_check
```

```
Elapsed time is 25.938098 seconds.
>> speed_check
Elapsed time is 24.118749 seconds.
>> speed_check
Elapsed time is 26.874450 seconds.
```

--------------------speed_test_AI.m--------------------

```
x = randi([1 600], 1, 1000)/10;
y = randi(10000, 1, 1000);

tic;
for i = 1:1000
    SolveBesselAI(x(i), y(i));
end
toc;
```

```
>> speed_test_AI
Elapsed time is 177.906670 seconds.
>> speed_test_AI
Elapsed time is 196.428480 seconds.
>> speed_test_AI
Elapsed time is 188.661306 seconds.
>> speed_test_AI
Elapsed time is 189.041462 seconds.
>> speed_test_AI
Elapsed time is 191.873669 seconds.
```