סימולציה נומרית

תנועת רקטות

מדע חישובי - פרויקט יא׳

מחבר: רועי דביר

מוסד לימוד: חמד״ע

מנחה: דריה דוברובין

תמצית

רקטות, בדומה לטילים, הם גופים הנעים באמצעות דחף אותו יוצר המנוע. אולם בשונה מטילים הם חסרי מערכת ניהוג ועל כן מהווים מודל פשוט לבחינת תנועה בהשפעת כוחות אווירודינמיים. גם מטוסים, כל עוד אין מתייחסים ליכולת הניהוג שלהם, מתנהגים בדיוק כמו רקטות. למרות פשטותו של המודל, המשוואות המתארות אותו קשות לפתרון אנליטי, מלבד במקרי קצה פשוטים, ולכן יש צורך בסימולציה נומרית שתיתן את מהירות ומיקום הרקטה כתלות בפרמטרים השונים. בעבודה זו אציג סימולציה נומרית המבוססת על אלגוריתם אוילר-קרומר לפתרון המודל וכן תוצאות שונות שהתקבלו באמצעותה. באמצעות סימולציה זו נחקור את מהירות ההמראה של מטוס כתלות במסתו, מקדם הגרר, מקדם העילוי והדחף הנוצר על ידי המנועים.

תוכן

1. מבוא (גם חלק מהרקע התיאורטי)

רקטה היא גוף הנע כתוצאה מדחף שמייצר המנוע אולם בניגוד לטיל, לרקטה אין מערכת ניווט והנחיה ולכן מסלול התנועה שלה יכול להיות מחושב מראש (תנועה בליסטית). לרוב, צורת הרקטה היא גוף גלילי ארוך והיא משוגרת מתוך מתקן שיגור ייעודי [1]. את הרקטות הראשונות המציאו הסינים במהלך המאה ה-13 והן היו מונעות באבקת שריפה [2]. הניתוח המתמטי הראשון לתנועת בליסטית של רקטות בוצע על ידי ויליאם מור הבריטי בשנת 1813. בשנת 1903 פרסם מען הטילים הרוסי קונסטנטין ציאולקובסקי משוואה המתארת את מהירותו הסופית של טיל כפונקציה של גורמים שונים. משוואה זו מוכרת כיום כנוסחת ציאולקובסקי והיא מהווה את הבסיס לעבודות מאוחרות יותר בנושא הנעת טילים רב שלבית [3]. נוסחת ציאולקובסקי מתארת את מהירותה הסופית של הרקטה כתלות במהירות ובמסה ההתחלתיות, המסה הסופית של הרקטה (כתוצאה משריפת הדלק) ומהירות פליטת הגזים. הנוסחה אינה מביאה בחשבון את הגרר הנוצר כתוצאה מהחיכוך עם האוויר ואת כוח העילוי. במהירויות נמוכות (למשל הליכה) אין לחיכוך עם האוויר משמעות רבה אולם ככל שהמהירות עולה לחיכוך יש השפעה רבה יותר ויותר על תנועת הגוף. דבר זה נובע מכך שהחיכוך עם האוויר מפעיל כוח שגודלו ריבועי ביחס למהירות [4]. בנוסף, ישנה משמעות רבה גם לכוח העילוי וגם זאת מאותה סיבה של תלות ריבועית במהירות [5]. מכיוון שמערכת המשוואות המתארות את תנועת הרקטה תחת השפעת הכוחות האוירודינמיים היא לא ליניארית (כי כוח הגרר קשור למהירות בריבוע) קשה ואפילו בלתי אפשרי לפתור אותן באופן אנליטי. לכן, יש צורך בסימולציה נומרית כדי לקבל את מסלול הרקטה כתלות בפרמטרים השונים [6]. סימולציות נומריות אלו נבדלות אחת מן השנייה בפרמטרים ובמודלים אותן הן מביאות בחשבון וכן באלגוריתם באמצעותו מחושב הפתרון. מאמרם של קמפבל ואוקוצו [6] העוסק בפרוייקטים בטילאות לתלמידי אוירונאוטיקה. במאמר זה מובא פרויקט לבניית טיל אמיתי אשר יכול להגיע לגובה של כחצי קילומטר. לאחר מכן, הטיל נוחת חזרה באמצעות מצנח על מנת שיוכלו להשתמש בו שוב. על מנת לסייע בחישוב המסלול ואופן השיגור האופטימלי, מציעים קמפבל ואוקוצו לפתור את משוואות התנועה באמצעות אלגוריתם רונגה-קוטה. אלגוריתם זה אומנם מוכרב יותר מאלגוריתם אוילר-קרומר בו נשתמש בעבודה זו, אך שניהם דומים בכך שהם אלגוריתמים איטרטיביים.

תופעה מעניינת בנוגע לתנועת רקטות תחת השפעת כוח הכבידה היא תמרון המכונה gravity turn. בתמרון זה אנחנו משתמשים בכוח הכבידה הפועל תמיד לכיוון מרכז כדור הארץ על מנת לגרום לקרטה לשנות את כיוון תנועתה. פאריס ווינדל [7] מתארים את אופן ביצוע התמרון ואת המשוואות הגורמות לפעולתו. תמרון זה נובע מן העובדה שכוח הכבידה פועל לכיוון קבוע אבל הדחף (והמהירות) לא בהכרח מקבילים אליו. במקרה שכזה, נקבל שיש כוח (כבידה) שרכיב שלו פועל בכיוון מאונך לכיוון המהירות. תופעה זו היא מה שגורם לתנועה מעגלית (כוח מאונך למהירות) ולכן נקבל שינוי בזווית. ההבדל המשמעותי שבין gravity turn לבין תנועה מעגלית פשוטה נובע מכך שבתנועת הרקטה גם המהירות תלויה בזמן, בעוד שבתנועה מעגלית פשוטה (למשל כדור הקשור לחוט) אנו מניחים שגודל המהירות לא משתנה ורק הכיוון מושפע מן הכוח הצנטריפטלי. אוניברסיטת סטנפורד פיתחה סימולטור רקטות [8] אשר יודע, מלבד פתרון משוואות התנועה, לקחת בחשבון גם אי ודאויות בפרמטרים השונים. באמצעות סימולטור זה ניתן לחשב מסלולים בצורה סטטיסטית עבור רקטות שאיננו יודעים במדויק את הפרמטרים שלהם. דבר זה חשוב מאוד כי במציאות, יש תמיד סטיות קלות בין התכנון לבין הייצור וחשוב לדעת מה תהיה ההשפעה של הסטיות מן התכנון על הביצועים הסופיים של הרקטה.

בעבודה זו ננתח באופן נומרי באמצעות סימולציית מחשב את תנועתה של רקטה המושפעת גם מחיכוך עם האוויר. הניתוח הנומרי יבוצע אמצעות כתיבת משוואה עבור תאוצת הרקטה תחת השפעת כוח הדחף הנוצר על ידי המנוע, הגרר והעילוי כתוצאה ממהירות הגוף והכבידה הנוצרת על ידי כדור הארץ. את המשוואות נפתור באמצעות האלגוריתם אוילר-קרומר. לאחר מכן נראה כיצד מהירות ההמראה של מטוס מושפעת מפרמטרי העילוי והגרר, וזאת מכיוון שגם מטוסים, כל עוד אין מתייחסים ליכולת הניהוג שלהם, מתנהגים בדיוק כמו רקטות.

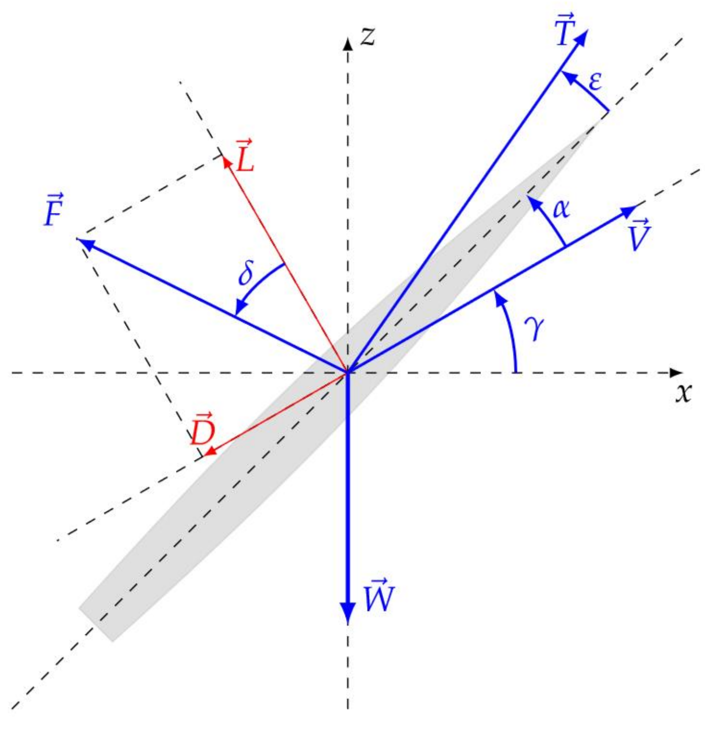
2. מטרת המחקר

בין היתר המטרה הסופית של המחקר היא לבדוק מהו הקשר בין מהירות ההמראה של מטוס ללא יכולת ניהוג, לבין פרמטרים כגון מקדם העילוי, מקדם העילוי (הקובעים את הכוחות האווירודינמיים ומשפיעים רבות על התנועה), המסה של המטוס ולבדוק איך משפיעים הפרמטרים האווירודינמיים על מסלול תנועתו.

3. תיאוריה

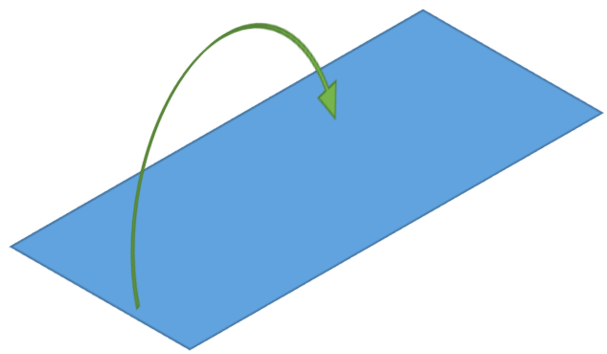
3.1. עקרונות המודל ומגבלותיו

המודל מבוסס על חוקי ניוטון, עקרון סכימת כוחות ומניח כי הרקטה היא גוף נקודתי. כתוצאה מכך שאנו מניחים זאת, זווית הנטייה של הרקטה אינה משתנה. המודל מניח רקטה חד שלבית, אולם ניתן להשתמש במודל זה על מנת לתאר רקטות רב שלביות (ראו נספח א – הנחיות להפעלת התוכנה). הנחה נוספת היא שמנוע הרקטה פולט גזים במהירות קבועה ושקצב שריפת הדלק אחיד (כלומר מסת הרקטה נתונה על ידי $m(t) = M - \alpha t$).



איור 1. תרשים של הכוחות הפועלים על הגוף ביחס למהירות. F הוא הכוח השקול של הגרר והעילוי.

בסימולציה הזווית ϵ בין ציר האורך של הרקטה לדחף מן המנוע קבועה – $\epsilon = 0$.



איור 2. דוגמה לתנועת הגוף מעל המישור

במודל זה אין התייחסות לעקמומיות כדור הארץ ולתזוזתו, כלומר – התנועה מבוצעת מעל מישור (**Error! Reference source not found.**). נקודת השיגור היא קבועה בראשית הצירים אבל גובה השיגור יכול להיות מעל המישור. ניתן להכניס את השפעת סיבוב כדור הארץ בעזרת אתחול המהירות האופקית למהירות שניתנת על ידי סיבוב כדוה״א (ראו נספח א – הנחיות להפעלת התוכנה). המודל מתחשב בשינוי הכבידה וצפיפות האוויר כתלות בגובה מעל פני כדוה״א. בנוסף, המודל אינו מאפשר תנועה של הרקטה מתחת לגובה פני הים.

3.2. ניתוח תיאורטי של הכוחות במערכת

על הרקטה פועלים 4 כוחות: כבידה, דחף, עילוי וגרר. כוח הכבידה שמופעל על ידי כדור הארץ מושך את הרקטה מטה. כוח זה תלוי בגובה הרקטה מעל פני כדור הארץ, במסת כדור הארץ M וברדיוסו R

$$\vec{W} = m(t)\vec{g}(z) = -\frac{GMm(t)}{(z(t)+R)^2}\hat{z}$$

הרקטה מונעת על ידי המנוע שמייצר דחף קדימה. כוח הדחף מושפע מקצב איבוד המסה וממהירות פליטת הגזים

$$\vec{T} = -\frac{dm}{dt}\vec{v}_{gases}$$

על הרקטה פועלים שני כוחות אווירודינמיים. כוח אחד הינו כוח העילוי שפועל אנכית לכיוון התנועה והכוח השני הוא כוח הגרר שפועל בכיוון הפוך למהירות [9]. כוח העילוי $\vec{L}$ מושפע ממהירות הגוף, צפיפות האוויר, שטח החתך וקבוע העילוי התלוי במבנה הגוף. כוח הגרר $\vec{D}$ מתואר באופן דומה עם

שטח חתך שונה ומקדם גרר במקום מקדם עילוי. באיור 1 ניתן לראות את $\vec{F}$ המתאר את השקול של כוחות אווירודינמיים אלו :

$$\vec{L} = \frac{1}{2} C_L S_L \rho(z) V^2(t) \quad \perp \hat{v}$$

$$\vec{D} = -\frac{1}{2} C_D S_D \rho(z) V^2(t)\ \hat{v}$$

על פי סכימת הכוחות נקבל (כאשר הכוחות כוללים את הסימנים עבור הכיוונים) שתי משוואות :

$$F_x = T_x + L_x + D_x$$

$$F_z = T_z + L_z + D_z + W$$

ועל ידי העברת אגפים ומתוך כך שמתקיים F=ma נקבל שתי משוואות דיפרנציאליות :

$$a_x = \frac{dv_x}{dt} = \frac{T_x + L_x + D_x}{m(t)}$$

$$a_z = \frac{dv_z}{dt} = \frac{T_z + L_z + D_z + W}{m(t)}$$

כעת נציב את הביטויים לכוחות ובאמצעות זהויות הטריגונומטריות ניתן לכתוב את המשוואות כך :

$$\frac{dv_x}{dt} = \frac{\frac{dm}{dt} v_{gases} \cos(\theta) - \rho\big(z(t)\big) V^2(t) \frac{C_D S_D \cos\big(\gamma(t)\big) - C_L S_L \sin\big(\gamma(t)\big)}{2}}{m(t)}$$

$$\frac{dv_z}{dt} = \frac{\frac{dm}{dt} v_{gases} \sin(\theta) - \rho\big(z(t)\big) V^2(t) \frac{C_D S_D \sin\big(\gamma(t)\big) + C_L S_L \cos\big(\gamma(t)\big)}{2}}{m(t)} - \frac{GM}{(z(t) + R)^2}$$

$$\gamma(t) = \arctan \frac{v_z(t)}{v_x(t)}$$

כאשר θ היא הזווית (הקבועה) שבין ציר האורך של הרקטה והמישור האופקי.

3.3. מודל לצפיפות אוויר

ישנה חשיבות רבה לצפיפות האוויר מכיוון שהכוחות האווירודינמיים תלויים בה באופן לינארי. לכן, המודל לוקח בחשבון את שינוי צפיפות האוויר כתלות בגובה תחת ההנחות של אטמוספירה סטנדרטית בינלאומית [10].

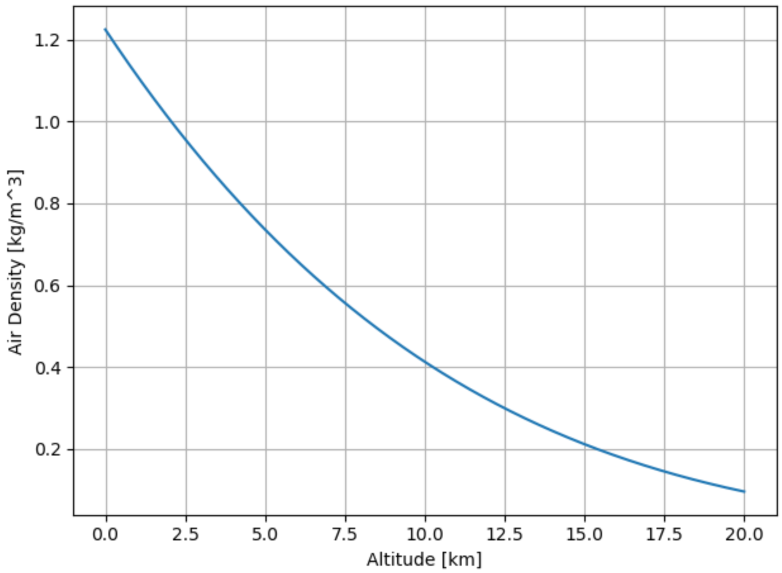
$$\rho(z) = \frac{p_0 M}{R T_0}\left(\frac{T_0 - Lz}{T_0}\right)^{\frac{gM}{RL}-1}$$

כאשר הפרמטרים הם: $p_0 = 101.325\ \mathrm{kPa}$ הוא לחץ האוויר בגובה פני הים, $\mathrm{T_0} = 288.15\ \mathrm{K}$ הטפרטורה בגובה פני הים (במעלות קלווין), $g = 9.80665\ \frac{\mathrm{m}}{\mathrm{s}^2}$ תאוצת הכובד על פני כדור הארץ, $\mathrm{L} = 0.0065\ \frac{\mathrm{K}}{\mathrm{m}}$ קצב שינוי הטמפרטורה ככל שעולים בגובה (במעלות קלווין למטר), קבוע הגז האידיאלי הוא $\mathrm{R} = 8.31447\ \frac{\mathrm{J}}{\mathrm{mol{\cdot}K}}$, והמסה המולרית של אויר היא $\mathrm{M} = 0.0289644\ \frac{\mathrm{kg}}{\mathrm{mol}}$.

לאחר הצבת כל הפרמטרים המספריים נקבל שצפיפות האוויר כתלות בגובה מעל פני כדור הארץ נתונה על ידי המשוואה

$$\rho(z) = 1.225 * (1 - 2.25 * 10^{-5} * z)^{4.255}\left[\frac{kg}{m^3}\right]$$

מחשבון למציאת צפיפות האוויר כתלות בפרמטרים אלו (ונוספים כגון לחות) ניתן למצוא ב- [11]. יש לשים לב כי מודל זה מוגבל לגובה של 18 ק״מ בלבד ואינו תקף מעבר לכך. למרות זאת, על מנת לפשט את המודל הטיסה, נניח שמודל צפיפות האויר תקף גם מעבר לגובה של 18 ק״מ. להנחה זו אין השפעה על התוצאות שיוצגו בהמשך מפני שכולן אינן מגיעות כלל לגבהים אלו. עבור מהירות המילוט ישנה השפעה זניחה על התוצאה מפני שצפיפות האויר יורדת מהר בגבהים גבוהים ולכן השפעתה שם זניחה.



איור 3. צפיפות האויר (ק"ג/מ"ק) כתלות בגובה. מגובה פני הים ועד לגובה 20 ק"מ.

4. אלגוריתם

בפרק 3.2 פיתחנו את המשוואות המתארות את השינוי במהירות הרקטה כתלות בזמן ובפרמטרים השונים. אם נוסיף עוד משוואה עבור שינוי המיקום נקבל שתי משוואות מהצורה הבאה:

$$\frac{d\vec{r}(t)}{dt} = \vec{v}(t)\,, \qquad \frac{d\vec{v}(t)}{dt} = f(\vec{r}, t)$$

כאשר $\vec{v} = \begin{pmatrix} v_x \\ v_z \end{pmatrix}, \vec{r} = \begin{pmatrix} x \\ z \end{pmatrix}$. ניתן לפתור מערכת משוואות מהצורה הזאת באמצעות שיטת אוילר-קרומר [12]. שיטת אוילר-קרומר היא שיטה איטרטיבית לפתרון משוואות דיפרנציאליות. בשיטה זו אנו מתחילים מתנאי ההתחלה של הבעיה $(\vec{r}(t_0),\ \vec{v}(t_0))$ ומהם אנו מתקדמים בצעדים קטנים על מנת לקבל את ערכי המשתנים במהלך הזמן. ההבדל בין שיטת אוילר לשיטת אוילר-קרומר הוא באופן בו אנו מחשבים את ערך המשתנה בצעד הזמן הבא. שיטת אוילר המקורית מתחשבת רק בערכי המשתנה בזמן הנוכחי ובערך הנגזרת בזמן זה. לעומת זאת, בשיטת אוילר-קרומר [12], עבור משתנה אחד אנו משתמשים בשיטת אוילר הרגילה

$$\vec{r}(t_{n+1}) = \vec{r}(t_n) + \vec{v}_n\,\Delta t$$

אולם עבור המשתנה השני אנו משתמשים בערך המשתנה הראשון בזמן n+1 (מסומן באדום מודגש):

$$\vec{v}(t_{n+1}) = \vec{v}(t_n) + f(\boldsymbol{\vec{v}_{n+1}}, \mathrm{t})\,\Delta t$$

כאשר Δt הוא גודל צעד הזמן.

שיטת אוילר-קרומר יציבה יותר מאשר שיטת אוילר הרגילה ובנוסף השגיאה המצטברת קטנה יותר בשיטת אוילר-קרומר מאשר בשיטת אוילר הרגילה עבור אותו גודל של צעד זמן. עם זאת, כמו כל שיטה איטרטיבית גם בשיטה זו השגיאות מצטברות והן תלויות בגודל הצעד. לכן, ישנה עדיפות לצעדי זמן קטנים ככל האפשר אבל דבר זה משפיע על זמן הריצה של הסימולציה ולכן יש למצוא איזון בין זמן הריצה לבין גודל הצעד dt והשגיאה המצטברת.

5. אימות ותיקוף

לשם אימות ותיקוף המודל נבדוק את ההתאמה בין התוצאות הנומריות לבין התוצאות האנליטיות עבור בעיות פשוטות. המקרים אותם נבדוק הם:

1. נפילה חופשית
2. תנועה בליסטית בזויות שונות
3. התאמה לנוסחת ציאולקובסקי
4. מהירות מילוט מכדור הארץ

בכל המקרים אותם נבדוק בחלק אנו מתעלמים מכוחות החיכוך והעילוי (כלומר $C_D = C_L = 0$) ולכן גם אין חשיבות לאטמוספירה. בנוסף, בגבהים נמוכים (מאות מטרים) אין כל חשיבות לשינוי בגרביטציה.

הבדיקה הראשונה היא בדיקה של יציבות ודיוק נומרי. כלומר, אנחנו צריכים לדעת מהו גודל צעד הזמן אשר ייתן לנו תוצאות מדויקות מספיק במהלך החישוב.

מכיוון שהשגיאה בשיטות איטרטיביות מצטברת במהלך הזמן והיא תלויה בגודל צעד הזמן שבוחרים – יש לבדיקה זו חשיבות רבה. בחירת צעד זמן גדול תאפשר לנו לבצע סימולציות על טווחי זמן ארוכים מאוד אבל במחיר של שגיאה מצטברת גדולה. מצד שני – צעד זמן קטן מאוד ייתקן דיוק טוב אבל לא נוכל לבצע ריצות לזמנים ארוכים.

לאחר מכן נבדוק התאמה לתנועה בדו מימד – זריקה בליסטית. כידוע בזריקה בליסטית המרחק המקסימאלי מתקבל בזריקה בזווית של 45 מעלות. בנוסף, המרחק אליו יגיע הגוף בזמן חזרתו לגובה השיגור הוא סימטרי לזווית השיגור סביב 45 מעלות. כלומר שיגור בזווית של 22.5 מעלות יגיע לאותו מרחק אופקי כמו שיגור בזווית של 67.5 מעלות. בבדיקה זו נרצה לראות שמתקיימות שתי התחזיות הללו.

מכיוון ששתי הבדיקות הקודמות עסקו בגופים בעלי מסה קבועה, הבדיקה שהשלישית תהיה עבור גוף עם מסה משתנה. בבדיקה זו נבדוק התאמה בין התוצאות הנומריות לנוסחת ציאולקובסקי כאשר ישנה כבידה וכאשר מתעלמים ממנה.

הבדיקה האחרונה חוזרת שוב לגוף בעל מסה קבועה והמטרתה להראות כי הכבידה אכן משפיעה על גופים המנסים להתרחק מכדור הארץ. בבדיקה זו נשגר שני גופים אחד מעל מהירות המילוט והשני במהירות קטנה יותר.

.5.2 נפילה חופשית

בנפילה חופשית זמן ההגעה של גוף המשוחרר במהירות 0 מגובה h לקרקע נתון על ידי פיתרון המשוואה

$$h = h_0 + v_0 t + \frac{1}{2} g t$$

מכיוון שהגוף מתחיל ממנוחה ($v_0 = 0$) בגובה h מתקבל שזמן הגעתו לקרקע הוא

$$t = \sqrt{\frac{2h}{g}}$$

כלומר, עבור גוף המשוחרר מגובה h=5m נקבל שזמן ההגעה לקרקע הוא בקירוב $t = 1sec$ מפני שתאוצת הכובד היא בערך $g \approx 10\frac{m}{s^2}$. הערכים הנומריים מן הסימולציה והשוואה שלהם לערך התיאורטי מוצגים בטבלה 1. ניתן לראות כי החל מצעד זמן של 10ms מתקבלות תוצאות בעלות שגיאה קטנה מאוד (פחות מ- 0.1%).

שינוי באחוזים ביחס לזמן התיאורטי $\left(100\frac{\mathrm{t}-\mathrm{t_{theory}}}{\mathrm{t_{theory}}}\right)$	זמן הגעה לקרקע Seconds	
0	1.0098	תיאוריה
שדש-0.9714	1	$dt = 10^{-1}sec$
0.0188	1.01	$dt = 10^{-2}sec$
-0.0802	1.009	$dt = 10^{-3}sec$
-0.0703	1.0091	$dt = 10^{-4}sec$
-0.066	1.00914	$dt = 10^{-5}sec$

טבלה 1. תוצאות נומריות לנפילה חופשית של גוף מגובה 5 מטרים

5.3. תנועה בליסטית בזויות שונות

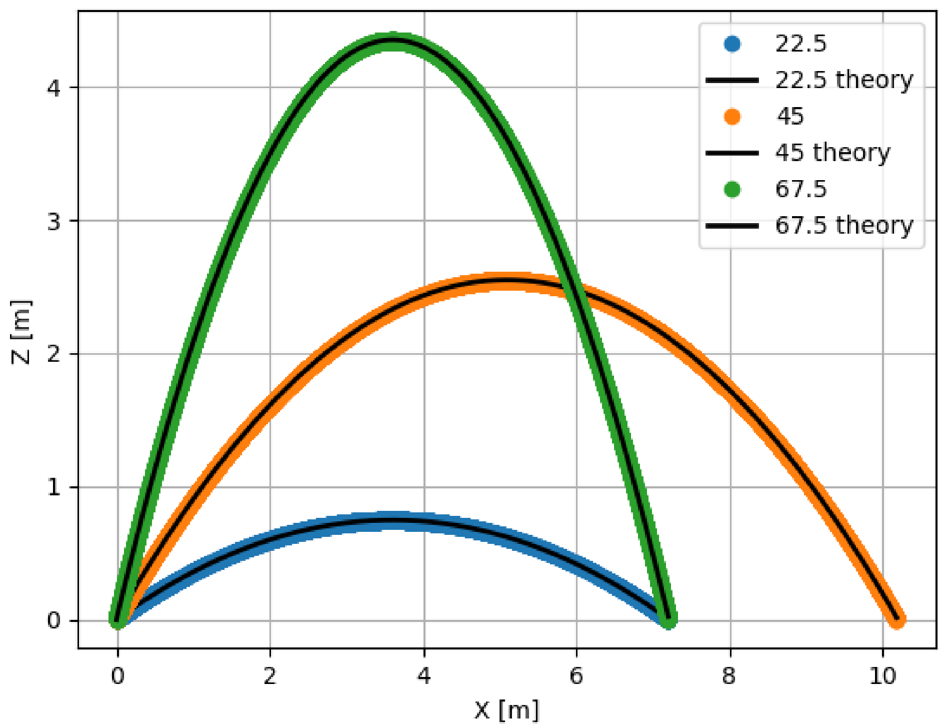
בתנועה בליסטית אנו יורים את הרקטה במהירות התחלתית מסוימת ולאחר מכן ההשפעה היחידה על התנועה נובעת מכוח הכבידה. הזמן עד לנפילת הרקטה בחזרה על הקרקע הוא

$$t = 2\frac{\mathrm{v}_0}{\mathrm{g}}\sin\theta_0$$

והמרחק אותו תעבור הרקטה בזמן זה הוא

$$x = v_0\cos\theta_0\, 2\frac{\mathrm{v}_0}{\mathrm{g}}\sin\theta_0 = \frac{v_0^2}{g}\sin 2\theta_0$$

השוואה של התוצאות הנומריות והתיאוריה מוצגת באיור 4.



איור 4. תנועה בליסטית בזויות שונות. ניתן לראות כי הטווח המקסימלי מתקבל עבור ירי ב-45 מעלות.

הפרמטרים הם: $\mathrm{dt} = 10^{-4} sec$, $\mathrm{v}_0 = 10\frac{\mathrm{m}}{\mathrm{sec}}$sec

5.4. התאמה לנוסחת ציאולקובסקי

שני הסעיפים הקודמים עסקו בירי של קליע בליסטי. נוסחת ציאולקובסקי לוקחת בחשבון גם את השינוי במסה ובמהירות בעקבות פליטת גזים מהרקטה. על פי נוסחת ציאולקובסקי המהירות הסופית של רקטה היא [3]

$$\Delta v = v_{final} - v_{initial} = v_{gases} \ln\left(\frac{m_{initial}}{m_{final}}\right)$$

אבל זה פיתרון עבור מצב שאין גרביטציה. אם ניקח בחשבון את הגרביטציה נקבל משוואת כוחות חדשה (לפי נוסחה 13 ב-[7])

$$m\frac{dv}{dt} = -mg + \alpha\, v_{gases}$$

נניח שקצב איבוד המסה , α, קבוע בזמן ולכן נעביר אגפים ונקבל משוואה דיפרנציאלת עבור המהירות $\frac{dv}{dt} = \frac{1}{m(t)}\alpha v_{gases} - g$ כעת ניתן לבצע אינטגרל

$$\Delta v = \int \left(\frac{1}{m(t)}\, \alpha v_{gases} - g\right) dt$$

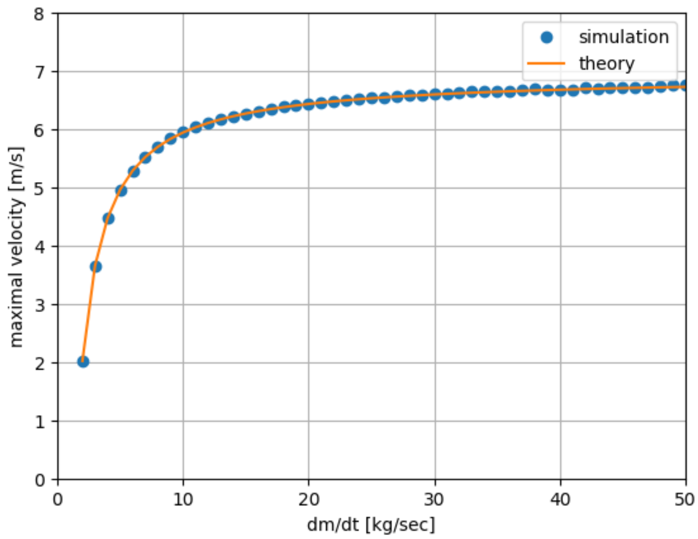
ונקבל את הביטוי למהירות הטיל לאחר זמן t. מכיוון שקצב איבוד המסה קבוע הדלק ייגמר לאחר

$$T = \frac{m_{fuel}}{\alpha}$$

ולאחר הצבת T בפיתרון נקבל

$$\Delta v = \alpha \ln\left(\frac{m_{initial}}{m_{final}}\right) - g\frac{m_{fuel}}{\alpha} = \alpha \ln\left(\frac{m_{final} + m_{fuel}}{m_{final}}\right) - g\frac{m_{fuel}}{\alpha}$$

השוואה בין הערכים התיאורטיים וערכי הסימולציה מוצגים באיור 5.



איור 5. השוואת המהירות הסופית לנוסחאת ציאולקובסקי. הפרמטרים הם:

$$m_{final} = 1kg, \qquad m_{initial} = 11kg, \qquad v_{gases} = 10, \qquad dt = 10^{-4}sec$$

5.5. מהירות מילוט

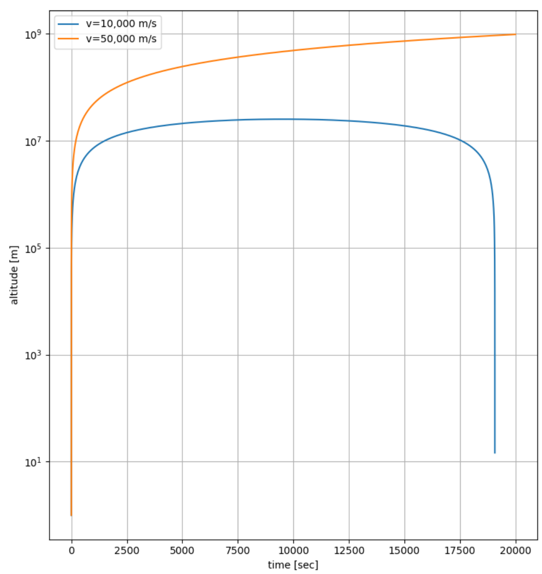
מהירות מילוט היא המהירות בה צריך לשגר קליע כדי שיעזוב את שדה הכבידה של גוף מסוים ויתרחק ממנו לאינסוף. משיקולי אנרגיה מתקיים

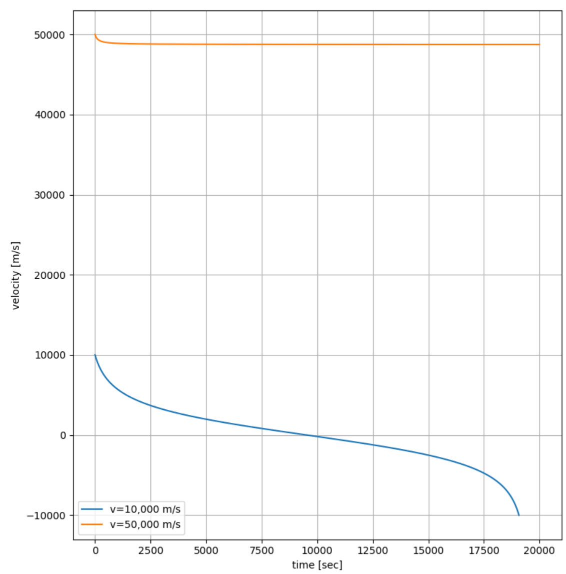
$$\frac{mv_{escape}^2}{2} = mgR_{earth}$$

ומכאן שמהירות המילוט היא

$$v_{escape} = \sqrt{2gR_{earth}} \approx 11200\,\frac{\mathrm{m}}{\mathrm{s}}$$

ומהירות זו אינה תלויה במסת הקליע המשוגר [13]. איור 5 מראה את הגובה של שני גופים המשוגרים אנכית – אחד במהירות הקטנה ממהירות המילוט ואחד מעל מהירות המילוט. הגוף המשוגר במהירות קטנה עולה אומנם לגובה רב אולם בהשפעת הכבידה חוזר בסוף לכיוון כדור הארץ. יש לשים לב כי הגרף הימני (גובה כתלות בזמן) מוצג בסקאלה לוגריתמית.



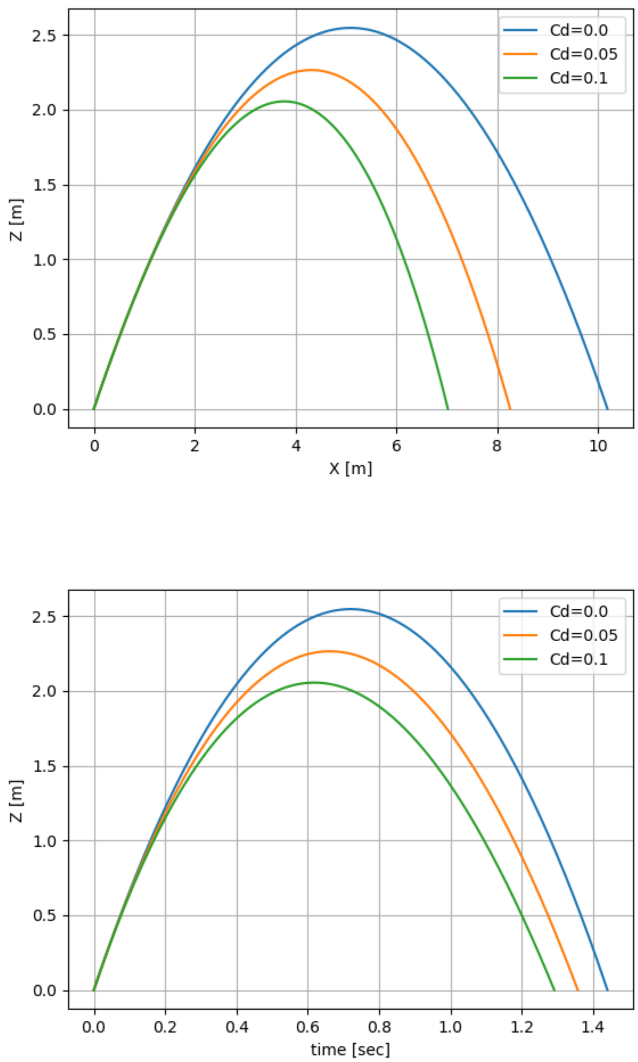


איור 6. תנועה מעל ומתחת למהירות המילוט. הגרף השמאלי מראה את גובה הרקטה כתלות בזמן והגרף הימני את מהירות הרקטה. הפרמטרים הם: (כחול) $v_z = 10{,}000\,\frac{\mathrm{m}}{\mathrm{s}}$, (כתום) $v_z = 50{,}000\,\frac{\mathrm{m}}{\mathrm{s}}$. מסת הגוף המשוגר היא m=1kg.

6. השפעת כוחות החיכוך והעילוי

6.1. זריקה בליסטית עם חיכוך

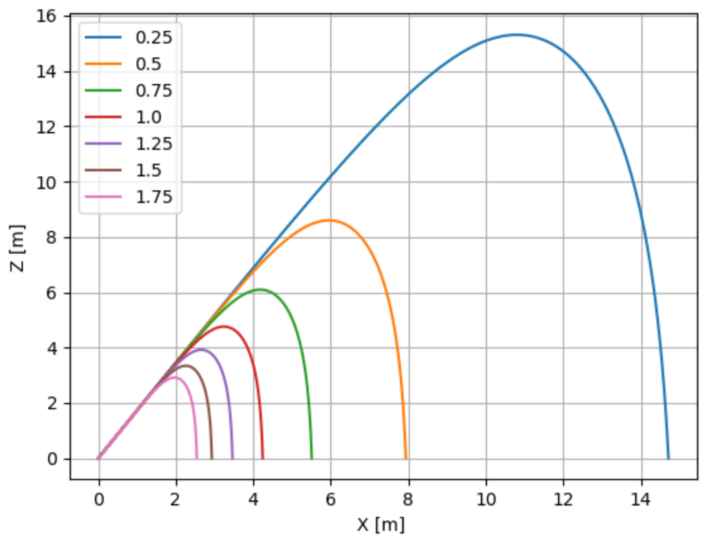
עד עכשיו התעלמנו מקיומו של חיכוך עם האוויר. כאשר לוקחים את החיכוך בחשבון, המהירות קטנה במהלך התנועה. כתוצאה מכך, בזריקה בליסטית הגובה המקסימלי אליו נוכל להגיע וכן המרחק אותו נעבור יקטנו ככל שהחיכוך יגדל. איור 6 מדגים את המסלול (גרף עליון) כן את והגובה כתלות בזמן (גרף תחתון) בזריקה בליסטית כתלות במקדם החיכוך עבור גוף בעל שטח חתך קבוע.



איור 7. זריקה בליסטית עם חיכוך. גרף שמאלי – מסלול דו מימדי (XZ) של הגוף. גרף ימני – גובה הגוף כתלות בזמן. הפרמטרים הם: $m = 1kg, v_0 = 10\frac{m}{s}, \gamma_0 = 45°, Sd = 1m^2$

כדי לבצע השוואה (איכותית בלבד) למאמר, בדקתי את מסלול הרקטה תחת השפעת כוח הגרר כאשר הרקטה משוגרת בזווית של 60 מעלות ובמהירות של 136 מטרים לשנייה. התוצאות מוצגות באיור 8 וניתן לראות שיש התאמה איכותית לגרף במאמר [6]. לא ניתן היה לבצע השוואה כמותית מכיוון שבמאמר לא מפורטים הפרמטרים השונים ובנוסף, התוצאות במאמר [6] אינן עד סוף המסלול אלא רק עד לשיא הגובה.

ניתן להבחין בתופעה נוספת – לאחר זמן מסויים, כוח החיכוך גורם לעצירה כמעט מוחלטת של הרקטה בכיוון X. כתוצאה מכך, סוף המסלול של הרקטה קרוב מאוד למסלול של נפילה חופשית.



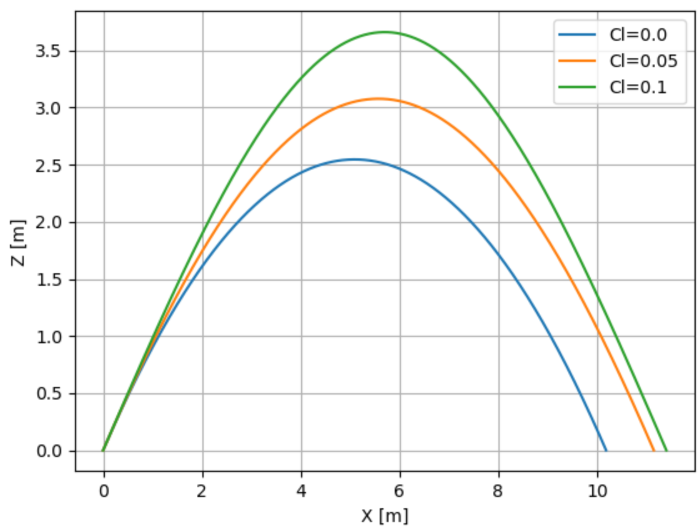
איור 8. תנועת רקטה תחת השפעת כוח גרר. הפרמטרים הם:

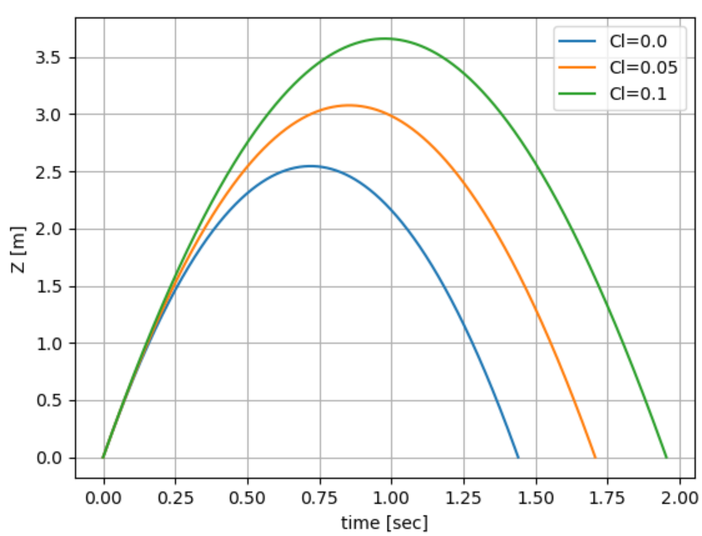
$$m = 1kg, v_0 = 136\frac{m}{s}, \gamma_0 = 60°, Sd = 1m^2$$

.

6.2. זריקה בליסטית עם עילוי

כוח העילוי גורם לכך שהגוף נדחף בניצב לכיוון המהירות כלפי מעלה. כתוצאה מכך, הגובה המרבי אליו יגיע הגוף עולה ככל שמקדם העילוי גדל והמרחק המרבי אליו יגיע הגוף יגדל בהתאם. איור 10 מראה את המסלול (גרף עליון) והגובה כתלות בזמן (גרף תחתון) בזריקה בליסטית כתלות במקדם העילוי עבור גוף בעל שטח חתך קבוע.



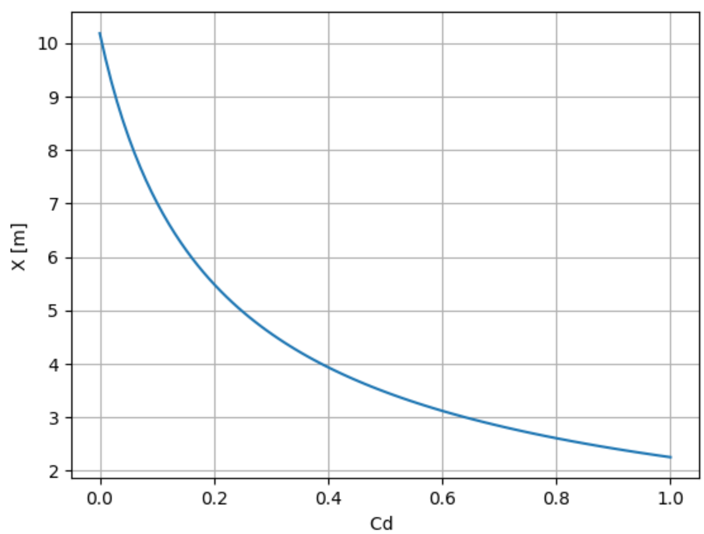


איור 9. זריקה בליסטית עם עילוי. גרף עליון – מסלול דו מימדי (XZ) של הגוף. גרף תחתון – גובה הגוף כתלות בזמן. הפרמטרים הם: $m = 1kg, v_0 = 10\frac{m}{s}, \gamma_0 = 45°, Sl = 1m^2$

7. קשרים בין פרמטרים

7.1. מרחק הגעה בזריקה בליסטית עם חיכוך כתלות במקדם הגרר

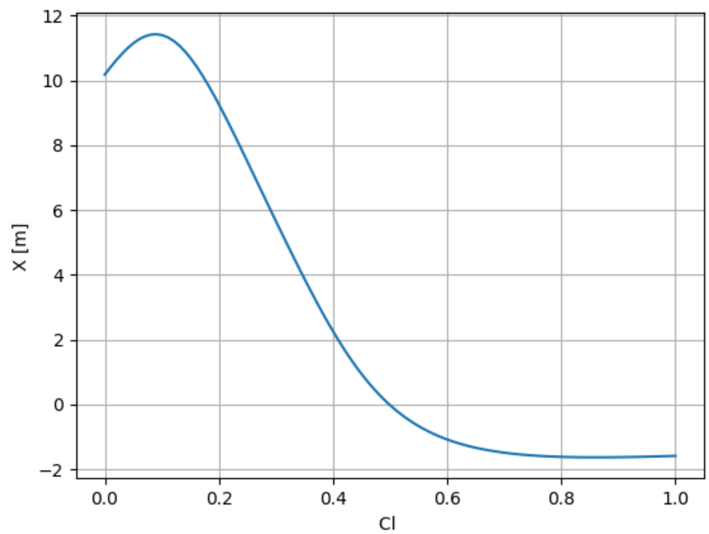
מרחק ההגעה של הרקטה תלוי במקדם הגרר שלה. איור 10 מראה את טווח ההגעה של רקטה המשוגרת בזווית 45 מעלות (זווית אופטימלית) כתלות במקדם הגרר. ניתן לראות כי כמו שהיינו מצפים ככל שהגרר עולה טווח ההגעה יורד.



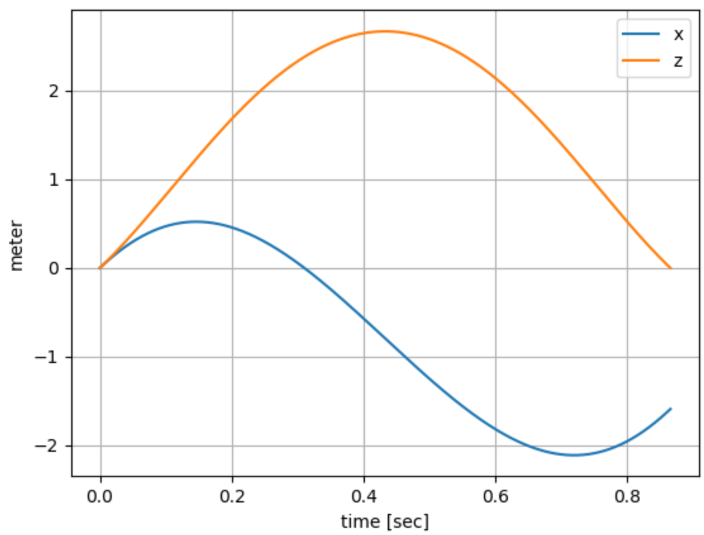
איור 10. מרחק הגעה של רקטה המשוגרת בזווית 45 מעלות כתלות במקדם הגרר. הפרמטרים הם: מסה 1ק"ג, מהירות שיגור 10 מטרים לשניה ושטח החתך של 1 מ"ר.

7.2. מרחק הגעה בזריקה בליסטית עם עילוי כתלות במקדם העילוי

תנועה בהשפעת גרר היא יחסית דבר פשוט להבנה. השפעת כוח העילוי מורכבת יותר בגלל שהוא פועל בניצב למהירות (כלומר שואף לסובב את הגוף). איור 12 מציג את טווח ההגעה של רקטה הנורית בזווית 45 מעלות כתלות במקדם העילוי. באופן לא אינטואיטיבי, הרקטה עשויה להגיע למיקום שלילי בציר X – כלומר לנחות מאחורי המשגר. דבר זה מתרחש בגלל תופעה המכונה "הזדקרות" – הרקטה, בהשפעת כוח העילוי מסתובבת, ומגיעה אל מאחורי המשגר כפי שניתן לראות באיור 13.



איור 11. מרחק הגעה של רקטה כתלות במקדם העילוי. הפרמטרים הם: מסה 1ק"ג, מהירות שיגור 10 מטרים לשניה ושטח חתך של 1 מ"ר.



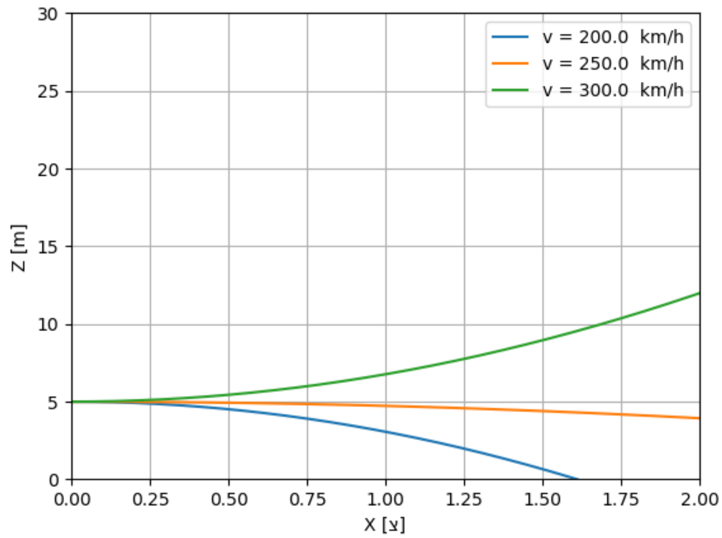
איור 12. מיקום הרקטה בציר X ובציר Z כתלות בזמן עבור מקדם עילוי של 1. הפרמטרים הם: מסה 1ק"ג, מהירות שיגור 10 מטרים לשניה ושטח חתך של 1 מ"ר.

8. מהירות המראה

על מנת לבצע הכללה של התוצאות באמצעות המודל והסימולציה, נחשב את מהירות המראה של מטוס תחת השפעת כוחות העילוי והחיכוך. על מנת שהמטוס ימריא (כאשר זווית המהירות היא 0) צריך שכוח העילוי יהיה גדול מכוח הכובד וכן שהדחף יהיה שווה או גדול מהגרר כלומר:

$$\frac{1}{2}C_L S_L \rho(0) V^2(t) = m(t)g$$

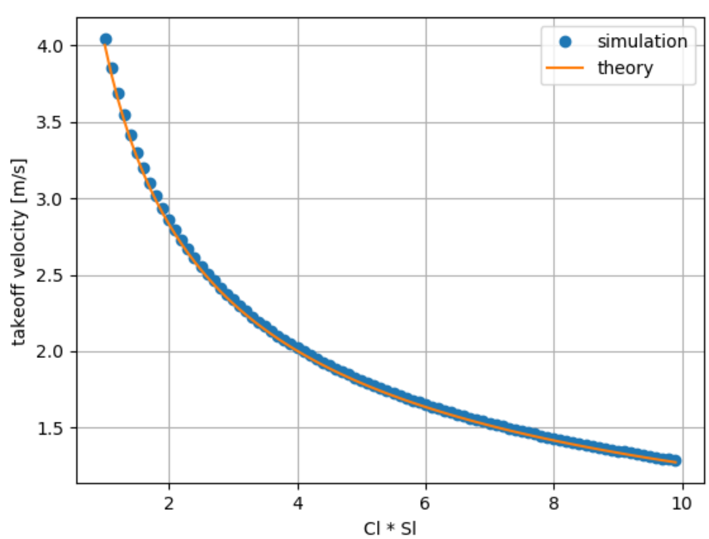
לכל מטוס ישנם פרמטרים שונים של מקדמי גרר ועילוי וכן חתכי פעולה שונים עבורם. למטוס airbus a320 יש שטח כנף של 122.6 מטרים רבועים ומשקלו המקסימלי בהמראה הוא 78000 ק״ג [14]. נניח שמקדם העילוי שלו הוא 2 ושזווית המהירות ההתחלתית היא 0 מעלות (המטוס נע במקביל לקרקע). איור 13 מראה שהמראה תתרחש רק כאשר עוברים מהירות סף מסוימת, שבמטוס זה היא כ-270 קמ״ש.



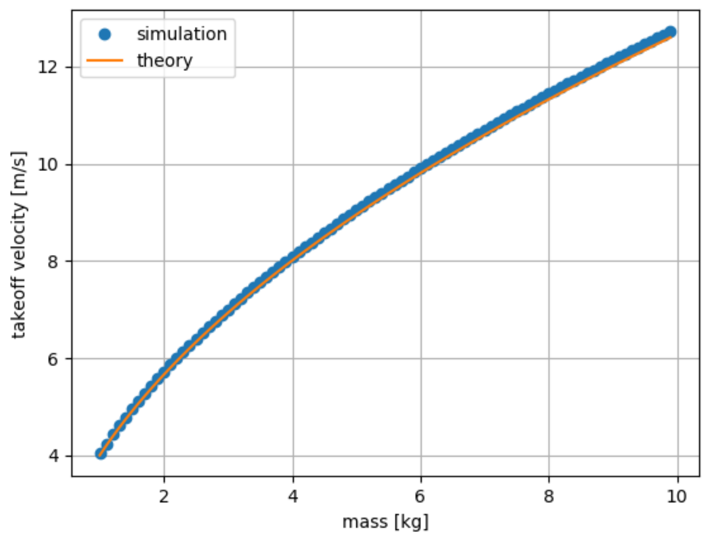
איור 13. הגובה ההתחלתי נקבע ל-5 מטרים כדי להראות שרק במהירות של כבערך 300 קמ"ש המטוס

מצליח להמריא.

מהמשוואה בתחילת העמוד ניתן לראות כי קיים קשר בין הפרמטרי העילוי (המכפלה $C_l S_l$) ובין מהירות ההמראה. בנוסף, קיים קשר בין מסת הגוף ובין מהירות ההמראה. קשרים אלו מוצגים באיורים 14,15.



איור 14. הקשר ההופכי בין מכפלת פרמטרי העילוי $C_l S_l$ ובין מהירות ההמראה. ככל שפרמטרי העילוי גדלים- מהירות ההמראה הולכת ויורדת. הפרמטרים הם: מסה עצמית 1 ק"ג, מהירות קבועה בכיוון אופקי בלבד.



איור 15. הקשר בין פרמטרי העילוי ובין מהירות ההמראה. ככל שפרמטרי העילוי גדלים- מהירות ההמראה הולכת ויורדת. הפרמטרים הם: מכפלת פרמטרי העילוי $C_l S_l = 1$, מהירות קבועה בכיוון אופקי בלבד. ככל שהמסה הולכת וגדלה- מהירות ההמראה תהיה גדולה יותר.

9. סיכום, מסקנות ורפלקציה

רקטה היא גוף הנע בכוח דחף אותו מייצר מנוע השורף דלק. בנוסף לכוח זה פועלים על הרקטה עוד שני כוחות אווירודינמיים: כוח העילוי אשר פועל בניצב למהירות וכוח הגרר הפועל בניגוד למהירות. לכוחות אלו ישנה השפעה רבה על אופן התנהגות הרקטה. בנוסף, לקחתי בחשבון את שינוי צפיפות האוויר ואת הגרביטציה המשתנה כתלות בגובה מעל פני כדור הארץ.

בעבודה זו הראיתי כי ניתן לבצע סימולציה נומרית של אופן תנועת הרקטה באטמוספירה בעלת צפיפות משתנה תוך כדי לקיחת כל הכוחות הללו בחשבון. את משוואות התנועה כתבתי כמערכת משוואות דיפרנציאליות ולאחר מכן המחשב פותר אותן באמצעות שיטת אוילר-קרומר על מנת לקבל את מיקום ומהירות הרקטה בכל רגע נתון.

כדי לאמת ולתקף את תוצאות הסימולציה בחנתי מספר מקרים פשוטים אשר התוצאות התיאורטיות שלהם ניתנות לחישוב. לאחר מכן, בחנתי מקרים בהם ניתן לראות בבירור את השפעת כוחות העילוי והגרר (בנפרד) על תנועת הרקטה ועל המרחק אליו היא תגיע.

לאחר שהראיתי שהסימולציה נותנת את התוצאות הנכונות וכוחות העילוי והגרר משפיעים על הרקטה, בחנתי מקרה אמתי של המראת מטוס נוסעים.

כשבדקתי את השפעת כוח החיכוך ומה הקשר בין מרחק ההגעה בזריקה בליסטית עם חיכוך כתלות במקדם הגרר, הראיתי שככל שהגרר עולה טווח ההגעה יורד.

כשבדקתי את השפעת כוח העילוי ומה הקשר בין מרחק הגעה בזריקה בליסטית עם עילוי כתלות במקדם העילוי באופן לא אינטואיטיבי, היה ניתן לראות שהרקטה עשויה להגיע למיקום שלילי בציר X – כלומר לנחות מאחורי המשגר. בהשפעת כוח העילוי הרקטה יכולה להסתובב, ולהגיע אל מאחורי המשגר בגלל תופעת ההזדקרות. ראינו ככל שפרמטרי העילוי גדלים, מהירות ההמראה הולכת ויורדת. ניתן היה לראות אף שככל שהמסה גדלה, מהירות ההמראה הייתה צריכה להיות גדולה יותר.

כמו כן, בסוף, ראינו שמטוס ללא יכולת ניהוג יכול להמריא בהשפעת הכוחות האווירודינמיים רק אם יצליח לנוע במהירות הגדולה ממהירות סף מסוימת בתחילת תנועתו.

לסיכום, הפרויקט היה מאד מאתגר. מאד נהניתי לכתוב את העבודה הזו ובמיוחד נהניתי לכתוב את הקוד. במהלך שעות השקעתי רבות בחיפוש יסודי את האינפורמציה אותה הנדרשת לעבודה וניסיתי לעשות אותה בצורה הטובה ביותר. למדתי ונחשפתי להרבה נושאים חדשים שלא ידעתי, למשל על תנועת גופים באוויר ועל רקטות, מטוסים ללא היגוי בפרט, המושפעים מהכוחות האווירודינמיים. למדתי רבות על האטמוספירה וגורמי ההשפעה של כדור הארץ על תנועתו של הגוף. למדתי על מנגנוני ההנעה של רקטות ושל מטוסים בעת קריאה ועל השפעת כוח הדחף של רקטה/מטוס בעת שיגור. בנוסף, התנסיתי ולמדתי איך להביא לכדי יישום מודל מדעי נומרי המשתמש במשוואות דיפרנציאליות וכיצד לבצע מחקר בנושא. העבודה הייתה מספקת מאד ואף רציתי להמשיך לחקור מתוך סקרנות... באמצעות הסימולציה יהיה ניתן להמשיך ולבחון מקרים מעניינים אחרים. אפשר, למשל, לבדוק כיצד יתנהג טיל רב שלבי באמצעות ביצוע סימולציה עבור השלב הראשון וקביעת תנאי ההתחלה לשלב השני על פי התוצאות שיתקבלו בסוף השלב הראשון.

10. ביבליוגרפיה

[1] י. גרוס, "רקטות – מסלולים ויירוטים," 14 יולי 2014. [מקוון]. Available: https://davidson.weizmann.ac.il/stub-44.

[2] Wikipedia contributors, "Rocket," 2018 11 27. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Rocket&oldid=870813835. [Accessed 2018 11 28].

[3] תורמי ויקיפדיה, "נוסחת ציאולקובסקי," 2018 מאי 31. [מקוון]. Available: https://he.wikipedia.org/w/index.php?title=%D7%A0%D7%95%D7%A1%D7%97%D7%AA_%D7%A6%D7%99%D7%90%D7%95%D7%9C%D7%A7%D7%95%D7%91%D7%A1%D7%A7%D7%99&oldid=20831200. [התבצעה גישה ב- 2018 נובמבר 28].

[4] תורמי ויקיפדיה, "גרר (כוח)," 2018 יולי 22. [מקוון]. Available: https://he.wikipedia.org/w/index.php?title=%D7%92%D7%A8%D7%A8_(%D7%9B%D7%95%D7%97)&oldid=23459825. [התבצעה גישה ב- 2018 נובמבר 28].

[5] תורמי ויקיפדיה, "כוח עילוי," 2018 ספטמבר 16. [מקוון]. Available: https://he.wikipedia.org/w/index.php?title=%D7%9B%D7%95%D7%97_%D7%A2%D7%99%D7%9C%D7%95%D7%99&oldid=23832294. [התבצעה גישה ב- 2018 נובמבר 28].

[6] T. A. J. Campbell and M. Okutsu, "Model Rocket Project for Aerospace Engineering Course: Trajectory Simulation and Propellant Analysis," *arXiv,* 2017.

[7] J. Peraire ו S. Widnal, "Variable Mass Systems: The Rocket Equation - MIT OpenCourseWare," 2008. [מקוון]. Available: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-07-

.dynamics-fall-2009/lecture-notes/MIT16_07F09_Lec14.pdf [התבצעה גישה ב- 09 December 2018].

[8] W. J. Eerland, S. Box and A. Sóbester, "Cambridge Rocketry Simulator – A Stochastic Six-Degrees-of-Freedom Rocket Flight Simulator," *Journal of Open Research Softwar,* vol. 5, no. 5, 2017.

[9] NASA, "Aerodynamics Forces," 05 May 2015. [Online]. Available: https://www.grc.nasa.gov/www/k-12/airplane/presar.html. [Accessed 07 December 2018].

[10] Wikipedia contributors, "Density of air," 13 September 2018. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Density_of_air&oldid=859335905.

[11] Engineers Edge, "Air Density and Specific Weight Equations and Calculator," [Online]. Available: https://www.engineersedge.com/calculators/air-density.htm. [Accessed 2018 December 2].

[12] Wikipedia contributors, "Semi-implicit Euler method," 2018 July 30. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Semi-implicit_Euler_method&oldid=852709261. [Accessed 2018 December 1].

[13] תורמי ויקיפדיה, "מהירות מילוט," 28 אוגוסט 2018. [מקוון]. Available: https://he.wikipedia.org/w/index.php?title=%D7%9E%D7%94%D7%99%D7%A8%D7%95%D7%AA_%D7%9E%D7%99%D7%9C%D7%95%D7%98&oldid=23688541. [התבצעה גישה ב- 04 דדצמבר 2018].

[14] "Airbus A320 Short to Medium-Range Jetliner," [Online]. Available: http://www.aerospaceweb.org/aircraft/jetliner/a320/. [Accessed 04 December 2018].

11. נספחים

11.1. נספח א – הנחיות להפעלת התוכנה

יש לייבא את כל הפונקציות מתוך rocket_project

לאחר מכן יש להשתמש בפונקציה set_parameters על מנת לקבל מילון של הפרמטרים של הרקטה אותו ניתן להעביר לאחר מכן לפונקציה set_initial_values. פונקציה זו משמשת לקביעת נתונים שאינם משתנים במהלך הסימולציה. הפרמטרים אותם יש לקבוע הם:

- theta_rocket_degree – הזוית במעלות שבין המהירות לציר האורך של הרקטה
- gas_velocity – מהירות פליטת הגזים במטרים לשניה
- m0 – המסה העצמית של הרקטה ללא הדלק בק"ג
- Cd – מקדם הגרר
- Cl – מקדם העילוי
- Sd – שטח חתך לכוח הגרר במטרים רבועים
- Sl – שטח חתך לכוח העילוי במטרים רבועים
- dmdt – קצב השינוי במסה (=קצב איבוד הדלק) ביחידות של ק"ג לשנייה

כעת יש לקבוע את תנאי ההתחלה של הסימולציה באמצעות הפונקציה set_initial_values. הפונקציה מקבלת את הפרמטרים הבאים ומחזירה מילון של ערכי ההתחלה:

- x – מיקום התחלתי בציר x
- z – מיקום התחלתי בציר z (גובה)
- velocity – המהירות ההתחלתית (גודל בלבד) במטרים לשנייה

- theta_velocity_degree – כיוון המהירות ההתחלתית במעלות. 0 מעלות משמעותו בכיוון החיובי של ציר X ו- 90 מעלות בכיוון החיובי של ציר Z.
- fuel_mass – מסת הדלק ההתחלתית בק״ג

עכשיו ניתן להריץ את הסימולציה באמצעות קריאה לפונקציה Euler_Cromer. הפרמטרים אותם מקבלת הפונקציה הם

- values – מילון ערכי ההתחלה שהתקבל מהקריאה לפונקציה set_initial_values
- parameters – מילון הפרמטרים שהתקבל מהקריאה לפונקציה set_parameters
- t_init – זמן ההתחלה של הסימולציה בשניות.
- t_final – הזמן בשניות אליו שואפת הסימולציה להגיע אם הרקטה לא ירדה לפני כן מתחת לגובה פני הים.
- dt – צעד הזמן לסימולציה בשניות. ברירת המחדל היא 1 מילישנייה
- rhs_dict – מילון המכיל את המשוואות אותם יש לפתור. משתנה זה מקבל כברירת מחדל את אוסף המשוואות של הרקטה על פי הגדרה במודול rocket_project.

הפונקציה מחזירה אובייקט מסוג pandas.DataFrame שהאינדקס שלו הוא ציר הזמן של הסימולציה והערכים מתאימים לערכי הסימולציה (איור 9). דוגמאות להרצה ניתן לראות בנספח ג.

	mass	vx	vz	x	z
time					
0.0000	1.0	3.826834	9.238795	0.000000	0.000000
0.0001	1.0	3.826834	9.237813	0.000383	0.000924
0.0002	1.0	3.826834	9.236831	0.000765	0.001848
0.0003	1.0	3.826834	9.235849	0.001148	0.002771
0.0004	1.0	3.826834	9.234867	0.001531	0.003695
0.0005	1.0	3.826834	9.233886	0.001913	0.004618
0.0006	1.0	3.826834	9.232904	0.002296	0.005542
0.0007	1.0	3.826834	9.231922	0.002679	0.006465
0.0008	1.0	3.826834	9.230940	0.003061	0.007388
0.0009	1.0	3.826834	9.229958	0.003444	0.008311

איור 9. דוגמא לתוצאות המתקבלות מהפונקציה Euler_Cromer.

11.2. נספח ב – תדפיס של מודול rocket_project.py

```
# coding: utf-8

import numpy as np

#

def set_initial_values(x, z, velocity, theta_velocity_degree, fuel_mass,

                       parameters):

    """

    function for initalizing all the variables of the simulations

    x, z: the initial position of the rocket in meters

    velocity: the magnitude of the rocket velocity at t=0 in m/s

    theta_velocity_degree: the degree between the rocket velocity

        and the x axis

    fuel_mass: fuel mass at t=0 in kg

    parameters: dict returned from set_parameters function

    returns: a dictionary of the rocket values at t=0

    """

    values = {

        "x": x,

        "z": z,

        "vx": velocity*np.cos(np.radians(theta_velocity_degree)),

        "vz": velocity*np.sin(np.radians(theta_velocity_degree)),

        "mass": fuel_mass+parameters["m0"]}

    print(values)

    return values
```

```
def set_parameters(theta_rocket_degree, gas_velocity, m0, Cd, Cl, Sd, Sl,
                   dmdt):
    """
    function for initalizing all the constant parameters of the simulations
    theta_rocket_degree: the degree between the rocket and the x axis.
        This value impact the direction of thrust
    gas_velocity: the magnitude of the rocket gas velocity in m/s
    m0: self mass of the rocket in kg. this is the mass after all fuel is gone
    Cd, Cl: drag and lift coefficients, respectivly
    Sd, Sl: drag and lift surface size in m^2
    dmdt : mass loss coefficient (referd to as alpha in the work)

    returns: a dictionary of the rocket constant parameters
    """
    parameters = {
        "theta_rocket": np.radians(theta_rocket_degree),
        "gas_velocity": gas_velocity,
        "m0": m0,
        "Cd": Cd,
        "Cl": Cl,
        "Sd": Sd,
        "Sl": Sl,
        "dmdt": dmdt}
    print(parameters)
    return parameters
```

```
def Rho(z):
    """
    air density at altitude of z meters based on
        https://en.wikipedia.org/wiki/Density_of_air#Altitude
    z: altitude in meters

    returns: air density in kg/m^3
    """
    p0 = 101.325e3  # sea level standard atmospheric pressure, Pa
    T0 = 288.15     # sea level standard temperature, K
    g = 9.80665     # earth-surface gravitational acceleration, m/s²
    L = 0.0065      # temperature lapse rate, K/m
    R = 8.31447     # ideal (universal) gas constant, J/(mol·K)
    M = 0.0289644   # molar mass of dry air, kg/mol
    T = T0-L*z
    if T < 0:
        return(0)
    p = p0 * (T/T0)**(g*M/(R*L))
    rho = p*M/(R*T)
    return(rho)


def Gravity(z, me_kg=5.972e24, re_m=6371e3):
    """
    Gravity at altitude z meters above sea level
```

```
    z: altitude in meters

    returns: gravity aceleration in m/sec^2

    """

    from scipy.constants import G

    return(G*me_kg/(re_m+z)**2)



def Drag(values, parameters):

    """

    calculate the drag force = 0.5*Cd*Sd*(v^2)

    values and parameters - the dictionaries of the current rocket state

    returns drag force in N

    """

    return(0.5 * parameters["Cd"] * Rho(values["z"]) * parameters["Sd"] *

           (values["vx"]**2 + values["vz"]**2))



def Lift(values, parameters):

    """

    calculate the lift force = 0.5*Cl*Sl*rho*(v^2)

    values and parameters - the dictionaries of the current rocket state

    returns lift force in N

    """

    return(0.5 * parameters["Cl"] * Rho(values["z"]) * parameters["Sl"] *

           (values["vx"]**2 + values["vz"]**2))
```

```
def Thrust(t, values, parameters):
    """
    calculate the thrus force = alpha * (v_gas)
    values and parameters - the dictionaries of the current rocket state
    returns thrust force in N
    """
    return dm_dt(t, values) * parameters["gas_velocity"]


def theta_velocity(vx, vz):
    # returns the velocity degree using its x and z elements
    return(np.arctan2(vz, vx))


# % EQUATIONS
def dm_dt(t, values, parameters):
    # equation for the mass change
    # dm/dt = alpha (if there is fuel to burn)
    # dm/dt = 0 (if there is no more fuel to burn)
    m = values["mass"]
    if m > parameters["m0"]:  # rocket mass contains fuel
        return(-parameters["dmdt"])
    else:  # no change in mass if we are at the rocket self mass
        return(0)
```

```
def dvz_dt(t, values, parameters):

    # acellaration in z direction - see the related work for more details

    # returns the rhs of the equation dvz/dt = az

    z = values["z"]

    vx = values["vx"]

    vz = values["vz"]

    m = values["mass"]

    theta_rocket = parameters["theta_rocket"]

    gas_velocity = parameters["gas_velocity"]

    # the froces in Z direction

    F_thrust_z = \

        -dm_dt(t, values, parameters) * gas_velocity * np.sin(theta_rocket)

    F_drag_z = Drag(values, parameters)*np.sin(theta_velocity(-vx, -vz))

    # F_lift is perpendicular to v

    F_lift_z = -Lift(values, parameters)*np.cos(theta_velocity(-vx, -vz))

    return ((F_thrust_z + F_drag_z + F_lift_z)/m-Gravity(z))



def dvx_dt(t, values, parameters):

    # acellaration in x direction - see the related work for more details

    # returns the rhs of the equation dvx/dt = ax

    vx = values["vx"]

    vz = values["vz"]

    m = values["mass"]

    theta_rocket = parameters["theta_rocket"]
```

```
    gas_velocity = parameters["gas_velocity"]

    # the froces in X direction

    F_thrust_x = \

        -dm_dt(t, values, parameters) * gas_velocity * np.cos(theta_rocket)

    # drag is opposite to velocity

    F_drag_x = Drag(values, parameters) * np.cos(theta_velocity(-vx, -vz))

    # F lift is perpendicular to v

    F_lift_x = Lift(values, parameters) * np.sin(theta_velocity(-vx, -vz))

    return ((F_thrust_x + F_drag_x + F_lift_x)/m)



def dz_dt(t, values, parameters):

    # an euqation for the position using the relation dz/dt = vz

    # returns the rhs of the equation dz/dt = vz

    vz = values["vz"]

    return(vz)



def dx_dt(t, values, parameters):

    # an euqation for the position using the relation dx/dt = vx

    # returns the rhs of the equation dx/dt = vx

    vx = values["vx"]

    return(vx)
```

```
# % Euler Cromer Solver
# equation to solve. see Euler_Cromer_Step function for more details
rhs_dict = {"mass": dm_dt, "x": dx_dt, "z": dz_dt, "vx": dvx_dt, "vz": dvz_dt}


def Euler_Cromer_Step(rhs_dict, values, parameters, t0, dt=1e-3):
    new_values = values.copy()
    """
     rhs_dict holds the equations to solve in the form {variable: eqaution}
     the update step according to euler is
     new_value = old_value + dt * (equation evaluated at this time)
     since we update each field in a seperate step (sequential update)
     we get the euler-cromer method. for example:
     key1 is update
     key2 is updated using the values of the updated key1
     which i exactly euler-cromer method.
     (in regular euler key2 uses old key1)
    """
    for key in rhs_dict.keys():
        new_values[key] += rhs_dict[key](t0, values, parameters) * dt
    return new_values


def Euler_Cromer(values, parameters, t_init, t_final, dt=1e-3,
                 rhs_dict=rhs_dict):
    import pandas as pd     # for returning a dataframe object
```

```
from tqdm import tqdm   # for displaying a progress bar during simulation

t_values = np.arange(t_init, t_final, dt)  # time vector to simulate

# we collect the results at each time step into the list results

results = []

results.append(values.copy())

for t in tqdm(t_values[1:]):

    # run the euler-cromer step onec to go from t to t+dt

    results.append(Euler_Cromer_Step(rhs_dict, values=results[-1],

                                     parameters=parameters, t0=t, dt=dt))

    if results[-1]["z"] < 0:  # exit condition if rocket under sea level

        results.pop()

        break

# return a DataFrame object for better reading of the results

df = pd.DataFrame(results, index=t_values[:len(results)])

df.index.name = "time"

return df
```

11.3. נספח ג – תדפיס של הסקריפט rocket.py

```
# coding: utf-8

"""

Graphs for the work

"""

from rocket_project import *  # import everything from rocket_project.py

from matplotlib import pyplot as plt

import numpy as np

# %% Rho(z) - fig 3

plt.figure()

z = np.arange(0, 20e3)

plt.plot(z/1e3, list(map(Rho, z)))

plt.grid(True)

plt.xlabel("Altitude [km]")

plt.ylabel("Air Density [kg/m^3]")

# %% G(z) - not in the work

plt.figure()

z = np.arange(0, 1e7, 1e4)

plt.plot(z/1e3, list(map(Gravity, z)))

plt.grid(True)

plt.xlabel("Altitude [km]")

plt.ylabel("Gravity [N]")

# %% Comparing Angles - Figure 4

plt.figure()

theta0_range = [22.5, 45, 45+22.5]
```

```
for theta0 in theta0_range:
    parameters = set_parameters(theta_rocket_degree=0, gas_velocity=0, m0=1,
                                Cd=0, Cl=0, Sd=0, Sl=0, dmdt=0)
    values = set_initial_values(x=0, z=0, velocity=10,
                                theta_velocity_degree=theta0,
                                fuel_mass=0, parameters=parameters)
    results = Euler_Cromer(values, parameters, t_init=0, t_final=3,
                           dt=1e-4)
    plt.plot(results.x, results.z, 'o', label=str(theta0))
    t = results.index.values
    plt.plot(values['vx'] * t, values['vz'] * t - 0.5 * 9.80665 * (t**2),'k-',
             linewidth=2,
             label=str(theta0) + " theory")
plt.legend()
plt.grid(True)
plt.xlabel("X [m]")
plt.ylabel("Z [m]")
# %% Comparing dt - not in the work
plt.figure()
dt_range = np.logspace(-1, -3, 3)
for dt in dt_range:
    parameters = set_parameters(theta_rocket_degree=0, gas_velocity=10, m0=1,
                                Cd=0, Cl=0, Sd=0, Sl=0, dmdt=0)
    values = set_initial_values(x=0, z=0, velocity=10,
theta_velocity_degree=45,
                                fuel_mass=0, parameters=parameters)
```

```
    results = Euler_Cromer(values, parameters, t_init=0, t_final=3000,
                           dt=dt)
    plt.figure(num='rocket simulation')
    plt.subplot(2, 2, 1)
    results.x.plot()
    plt.subplot(2, 2, 2)
    results.z.plot()
    plt.subplot(2, 1, 2)
    plt.plot(results.x, results.z)
plt.legend(dt_range)
# %% Tsiolkovsky rocket equation - Figure 5
plt.figure()
res = []
gas_velocity = 10
fuel_mass = 1
body_mass = 1
dmdt_range = np.arange(2, 100)
for dmdt in dmdt_range:
    parameters = set_parameters(theta_rocket_degree=90,
                                gas_velocity=gas_velocity, m0=1,
                                Cd=0, Cl=0, Sd=0, Sl=0, dmdt=dmdt)
    values = set_initial_values(x=0, z=10, velocity=0, theta_velocity_degree=0,
                                fuel_mass=1, parameters=parameters)
    results = Euler_Cromer(values, parameters, t_init=0,
                           t_final=2, dt=1e-4)
```

```
    res.append(results.vz.max())

time_till_end_of_fuel = fuel_mass/dmdt_range

maximal_velocity = gas_velocity * np.log(2/1) - 9.8 * time_till_end_of_fuel

plt.plot(dmdt_range, res, 'o', label="simulation")

plt.plot(dmdt_range, maximal_velocity, label="theory")

plt.xlabel("dm/dt [kg/sec]")

plt.ylabel("maximal velocity [m/s]")

plt.grid()

plt.legend()

# %% escape velocity - Figure 6

plt.figure()

res = []

for v in [10e3, 50e3]:

    parameters = set_parameters(theta_rocket_degree=0, gas_velocity=0, m0=1,

                                Cd=0, Cl=0, Sd=0, Sl=0, dmdt=0)

    values = set_initial_values(x=0, z=1, velocity=v, theta_velocity_degree=90,

                                fuel_mass=0, parameters=parameters)

    results = Euler_Cromer(values, parameters, t_init=0,

                           t_final=20000, dt=1e-2)

    res.append(results)

plt.subplot(1, 2, 1)

plt.semilogy(res[0].z)

plt.semilogy(res[1].z)

plt.grid(True)

plt.legend(["v=10,000 m/s", "v=50,000 m/s"])
```

```
plt.subplot(1, 2, 2)
plt.plot(res[0].vz)
plt.plot(res[1].vz)
plt.grid(True)
plt.legend(["v=10,000 m/s", "v=50,000 m/s"])
# %% ballistic throw with drag - Figure 7
plt.figure()
res = []
cd_range = np.linspace(0, 1e-1, 3)
for cd in cd_range:
    parameters = set_parameters(theta_rocket_degree=0, gas_velocity=0, m0=1,
                                Cd=cd, Cl=0, Sd=1, Sl=0, dmdt=0)
    values = set_initial_values(x=0, z=0, velocity=10,
                                theta_velocity_degree=45,
                                fuel_mass=0, parameters=parameters)
    results = Euler_Cromer(values, parameters, t_init=0,
                           t_final=5, dt=1e-4)
    res.append(results)
for cd, r in zip(cd_range, res):
    plt.plot(r.x, r.z, label=f"Cd={cd}")
plt.grid(True)
plt.legend()
plt.xlabel("X [m]")
plt.ylabel("Z [m]")
plt.figure()
```

```
for cd, r in zip(cd_range, res):
    plt.plot(r.index, r.z, label=f"Cd={cd}")
plt.grid(True)
plt.legend()
plt.xlabel("time [sec]")
plt.ylabel("Z [m]")
# %% ballistic throw with lift range vs Cd - Figure 8
plt.figure()
res = []
cd_range = np.arange(0.25, 2, 0.25)
for cd in cd_range:
    parameters = set_parameters(theta_rocket_degree=0, gas_velocity=0, m0=1,
                                Cd=cd, Cl=0, Sd=1, Sl=0, dmdt=0)
    values = set_initial_values(x=0, z=0, velocity=136,
                                theta_velocity_degree=60,
                                fuel_mass=0, parameters=parameters)
    results = Euler_Cromer(values, parameters, t_init=0,
                           t_final=5, dt=1e-3)
    res.append(results)
for cd, r in zip(cd_range, res):
    plt.plot(r.x, r.z, label=f"Cd={cd}")
plt.grid(True)
plt.legend()
plt.xlabel("X [m]")
plt.ylabel("Z [m]")
```

```
# %% ballistic throw with lift - Figure 9

plt.figure()

res = []

cl_range = [0, 0.05, 0.1]

for cl in cl_range:

    parameters = set_parameters(theta_rocket_degree=0, gas_velocity=0, m0=1,

                                Cd=0, Cl=cl, Sd=0, Sl=1, dmdt=0)

    values = set_initial_values(x=0, z=0, velocity=10,

                                theta_velocity_degree=45,

                                fuel_mass=0, parameters=parameters)

    results = Euler_Cromer(values, parameters, t_init=0,

                           t_final=5, dt=1e-3)

    res.append(results)

for cl, r in zip(cl_range, res):

    plt.plot(r.x, r.z, label=f"Cl={cl}")

plt.grid(True)

plt.legend()

plt.xlabel("X [m]")

plt.ylabel("Z [m]")

plt.figure()

for cl, r in zip(cd_range, res):

    plt.plot(r.index, r.z, label=f"Cl={cl}")

plt.grid(True)

plt.legend()

plt.xlabel("time [sec]")
```

```
plt.ylabel("Z [m]")
# %% ballistic throw with drag range vs Cd - Figure 10
plt.figure()
res = []
cd_range = np.arange(0, 1, 0.01)
plt.close('all')
for cd in cd_range:
    parameters = set_parameters(theta_rocket_degree=0, gas_velocity=0, m0=1,
                                Cd=cd, Cl=0, Sd=1, Sl=0, dmdt=0)
    values = set_initial_values(x=0, z=0, velocity=10,
                                theta_velocity_degree=45,
                                fuel_mass=0, parameters=parameters)
    results = Euler_Cromer(values, parameters, t_init=0,
                           t_final=5, dt=1e-3)
    res.append(results.x.tolist()[-1])
plt.plot(cd_range, res)
plt.xlabel("Cd ")
plt.ylabel("X [m]")
plt.grid(True)
# %% ballistic throw with lift range vs Cl - Figure 11
plt.figure()
res = []
cl_range = np.linspace(0, 1, 100)
for cl in cl_range:
    parameters = set_parameters(theta_rocket_degree=0, gas_velocity=0, m0=1,
```

```
                                Cd=0, Cl=cl, Sd=0, Sl=1, dmdt=0)
    values = set_initial_values(x=0, z=0, velocity=10,
                                theta_velocity_degree=45,
                                fuel_mass=0, parameters=parameters)
    results = Euler_Cromer(values, parameters, t_init=0,
                           t_final=5, dt=1e-4)
    res.append(results.tail(1))
plt.plot(cl_range, [r.x.tolist()[0] for r in res])
plt.xlabel("Cl ")
plt.ylabel("X [m]")
plt.grid(True)
plt.figure()
plt.plot(results.index, results.x, label="X")
plt.plot(results.index, results.z, label="X")
plt.xlabel("Cl ")
plt.ylabel("meter")
plt.grid(True)
# %% aircraft - Figure 12
plt.figure()
res = []
v_range = np.linspace(200, 300, 3) * 1000 / 3600
for v in v_range:
    parameters = set_parameters(theta_rocket_degree=0, gas_velocity=0, m0=78e3,
                                Cd=0, Cl=2, Sd=0, Sl=122.6, dmdt=0)
    values = set_initial_values(x=0, z=5, velocity=v,
```

```
                                    theta_velocity_degree=0,
                                    fuel_mass=0, parameters=parameters)
    results = Euler_Cromer(values, parameters, t_init=0, t_final=2, dt=1e-3)
    res.append(results)
for v, r in zip(v_range, res):
    plt.plot(r.index, r.z, label=f"v = {v*3600/1000}  km/h")
plt.xlim([0, 2])
plt.ylim([0, 30])
plt.grid(True)
plt.legend()
plt.xlabel("X [m]")
plt.ylabel("Z [m]")
# %% aircraft - Figure 14 - takeoff velocity vs. lift coefficient
"""
0.5 Cl Sl rho(0) v^2 = m g  ==> v = sqrt( 2g/rho(0) ) * sqrt( m / (Cl Sl) )
"""
plt.clf()
res = []
m0 = 1
sl = 1
z0 = 5
cl_range = np.arange(1, 10, .1)
for cl in cl_range:
    v_theory = np.sqrt( 2*Gravity(0)/Rho(0) ) * np.sqrt( m0 / (cl*sl) )
    v_range = np.arange(v_theory*0.98, 1.02*v_theory, v_theory/100)
```

```
    for v in v_range:
        parameters = set_parameters(theta_rocket_degree=0, gas_velocity=0, m0=m0,
                                    Cd=0, Cl=cl, Sd=0, Sl=sl, dmdt=0)
        values = set_initial_values(x=0, z=z0, velocity=v,
                                    theta_velocity_degree=0,
                                    fuel_mass=0, parameters=parameters)
        results = Euler_Cromer(values, parameters, t_init=0, t_final=1e-3, dt=1e-4)
        if results.z.tolist()[-1] > z0:  # takeoff
            res.append(v)
            break # no need to check larger v values
plt.plot(cl_range, res, 'o', label="simulation")
plt.plot(cl_range, np.sqrt( 2*Gravity(0)/Rho(0) ) * np.sqrt( m0/(cl_range*sl)),
         label="theory")
plt.grid(True)
plt.xlabel("Cl * Sl")
plt.ylabel("takeoff velocity [m/s]")
plt.legend()
# %% aircraft - Figure 15 - takeoff velocity vs. lift mass
"""
0.5 Cl Sl rho(0) v^2 = m g  ==> v = sqrt( 2g/rho(0) ) * sqrt( m / (Cl Sl) )
"""
plt.clf()
res = []
m0 = 1
```

```
cl = 1

sl = 1

z0 = 5

m_range = np.arange(1, 10, .1)

for m0 in m_range:

    v_theory = np.sqrt( 2*Gravity(0)/Rho(0) ) * np.sqrt( m0 / (cl*sl) )

    v_range = np.arange(v_theory*0.98, 1.1*v_theory, v_theory/100)

    for v in v_range:

        parameters   =   set_parameters(theta_rocket_degree=0,   gas_velocity=0,
m0=m0,

                                    Cd=0, Cl=cl, Sd=0, Sl=sl, dmdt=0)

        values = set_initial_values(x=0, z=z0, velocity=v,

                                    theta_velocity_degree=0,

                                    fuel_mass=0, parameters=parameters)

        results  =  Euler_Cromer(values,  parameters,  t_init=0,  t_final=1e-3,
dt=1e-4)

        if results.z.tolist()[-1] > z0:  # takeoff

            res.append(v)

            break # no need to check larger v values

plt.plot(m_range, res, 'o', label="simulation")

plt.plot(m_range, np.sqrt( 2*Gravity(0)/Rho(0) ) * np.sqrt( m_range/(cl*sl)),

         label="theory")

plt.grid(True)

plt.xlabel("mass [kg]")

plt.ylabel("takeoff velocity [m/s]")

plt.legend()
```

```
# %% Comparing dmdt
plt.figure()
dmdt_range = np.logspace(0, 1, 2)
for dmdt in dmdt_range:
    parameters = set_parameters(theta_rocket_degree=90, gas_velocity=10, m0=1,
                                Cd=0, Cl=0, Sd=0, Sl=0, dmdt=dmdt)
    values = set_initial_values(x=0, z=10, velocity=0, theta_velocity_degree=0,
                                fuel_mass=1, parameters=parameters)
    results = Euler_Cromer(values, parameters, t_init=0, t_final=50,
                           dt=1e-4)
    print(results.tail(1))
    plt.figure(num='rocket simulation')
    plt.subplot(2, 2, 1)
    results.x.plot()
    plt.subplot(2, 2, 2)
    results.z.plot()
    plt.subplot(2, 1, 2)
    plt.plot(results.x, results.z)
plt.legend(dmdt_range)
plt.figure()
v = np.sqrt(results.vx**2 + results.vz**2)
v.plot()
results.vx.plot()
results.vz.plot()
```