טומוגרפיה גאומטרית

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במתמטיקה שימושית

נדב הראל

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
חשוון תש״ס, חיפה, אוקטובר 1999

תודות

המחקר נעשה בפקולטה למתמטיקה בהנחיית פרופ׳ חבר שי גירון מהחוג למתמטיקה באוניברסיטת חיפה.
אני מודה לטכניון ולקרן וולף על התמיכה הכספית הנדיבה בהשתלמותי.
אני מודה לאבי, ד״ר צבי הראל, על עזרתו ועצותיו המועילות.

חיבור זה מוקדש לזכרו של אחי, גלעד הראל ז״ל (1977–1996).

תוכן עניינים

תקציר **1**

סמלים וקיצורים **3**

1 מבוא **5**

1.1 מהי טומוגרפיה — סקירה היסטורית 5

1.2 סוגי טומוגרפיה 7

1.3 העבודה הנוכחית 10

2 טומוגרפיה גאומטרית **13**

2.1 מבוא 13

2.2 שחזור קבוצות מדידות עם מידה סופית 15

2.2.1 הטלות בשני כיוונים 15

2.2.2 הטלות בשלושה כיוונים או יותר 24

2.3 שחזור קבוצות קמורות 27

2.3.1 אפיון קבוצות כיוונים S שמבטיחות שחזור יחיד של כל קבוצה קמורה 28

2.3.2 שחזור קבוצות קמורות מהטלותיהן בשני כיוונים 30

2.3.3 אימות קבוצות קמורות 31

2.3.4 קביעה בשלבים של קבוצות קמורות 34

2.4 מוגדרות־היטב ויציבות 36

2.4.1 משפט Volčič 36

2.4.2 הערכות Longinetti 37

2.4.3 מגבלות על משפטי יציבות של בעית השחזור 37

2.5 דוגמת הריבוע 38

2.5.1 אינטואיציה לדוגמה הלא־קמורה 40

2.5.2 מציאת המקבילית החסומה 42

2.5.3 הטרנספורמציה למשולשים 44

3 אלגוריתם גרדנר לשחזור מארבעה כיוונים **49**

3.1 מציאת 3 נקודות על שפת K 49

3.2 מציאת נקודות נוספות על שפת K 50

3.3 מימוש האלגוריתם 51

3.4 בעיות באלגוריתם ובמימושו 52
3.5 דוגמאות לשימוש באלגוריתם 53

4 אלגוריתם השחזור "מינברס" 57
4.1 מצולעים כוכביים שכבתיים 58
4.2 האלגוריתם 59
4.2.1 סימולצית הטלה 59
4.2.2 צורת ניחוש, ופונקצית הערכה לניחוש 61
4.2.3 התחלת המינימיזציה 62
4.2.4 התחלת המינימיזציה מחדש 63
4.3 דוגמאות 64
4.3.1 שחזור משני כיוונים 64
4.3.2 שחזור צורות עם חורים 68
4.3.3 שחזור צורות לא קשירות 70

5 סיכום ומסקנות 73

א שיטות מינימיזציה 75
1.א שיטת ה"סימפלקס במורד" הרב־מימדית 75

ב שימוש בתוכניות השחזור 77
1.ב דרישות קדם והתקנה 77
2.ב קבצי מצולעים שכבתיים 77
3.ב שימוש בתוכנית השחזור לפי האלגוריתם של גרדנר 78
4.ב שימוש בתוכנית השחזור מינברס 80

ג התוכנית מינברס 83
minverse.c 83
minverse.h 92
main5.c 92
layered_polygon.h 95
layered_polygon.c 96
xray.h 99
xray.c 99
amoeba.c 102
nrutil.h 104
nrutil.c 104
stats.h 108
stats.c 108

ד תוכנית שחזור לפי האלגוריתם של גרדנר **109**
gardner.c 109
ssp.h 118
ssp.c 119
spline.h 122
spline.c 122
romberg.h 124
romberg.c 124

ביבליוגרפיה **127**

רשימת איורים

1.1 Röntgen ותצלום קרני ה־X הראשון שלו . . . 6
1.2 הספקטרום האלקטרומגנטי . . . 7
1.3 טומוגרפיה קווית: תאור סכימטי של השיטה . . . 8
1.4 טומוגרפיה קווית: תאור סכימטי של השיטה — חתך במישור ניצב . . . 8
1.5 פעולת מכשיר CT . . . 9

2.1 הטלה של קבוצה E . . . 14
2.2 שתי קבוצות (קמורות) עם הטלה זהה בכיוון אחד u . . . 15
2.3 עיגול היחידה המעוות (מקווקו): העיגול (קו מלא) מוגדל ברביעים השני והרביעי, ומוקטן ברביעים הראשון והשלישי. למרות מה שנדמה במבט ראשון, לעיגול המעוות חייבות להיות הטלות שונות בכיווני הצירים מאלו של העיגול המקורי. . . . 18
2.4 דוגמה של רכיב מיתוג . . . 20
2.5 מלבן חסום בעיגול . . . 22
2.6 העיגול הוא איחוד מלבנים החסומים בו: דוגמה של שלושה מהמלבנים . . . 23
2.7 עיגול לא נקבע מהטלותיו בזוג כיוונים לא ניצבים . . . 26
2.8 האיזומטריה ϕ . . . 32
2.9 מצולע קמור שלא ניתן לאימות משני כיוונים . . . 33
2.10 מקרה א' . . . 34
2.11 מקרה ב' . . . 34
2.12 מקרה א' . . . 39
2.13 מקרה ב' . . . 39
2.14 דוגמה לשחזור צורה שונה מהריבוע המסובב . . . 40
2.15 הריבוע וצורה שקולה נוספת . . . 41
2.16 הריבוע וצורה שקולה נוספת, זווית לא ישרה . . . 41
2.17 מציאת המקבילית החסומה בריבוע . . . 43
2.18 שני המשולשים מונחים אחד ליד השני . . . 45
2.19 קווי הגובה ובנית הקדקדים החדשים . . . 45
2.20 שני המשולשים, לפני ואחרי הטרנספורמציה . . . 46

3.1 שחזור אליפסה בשיטת גרדנר. $\epsilon = 5 \cdot 10^{-2}$ . . . 54
3.2 שחזור אותה אליפסה: $\epsilon = 10^{-3}$ . . . 55
3.3 שחזור מעגל מוכלל עם חזקה 1.6: $\epsilon = 5 \cdot 10^{-2}$ . . . 55

4.1 בדיקת חיתוך ישר וקטע 59
4.2 מציאת חיתוך ישר וקטע 60
4.3 מלבן בגוף סימטרי יחסית לציר y 65
4.4 שחזור מעגל מוכלל עם מינברס 66
4.5 שחזור ריבוע עם מינברס 66
4.6 שחזור ה"פטריה" 67
4.7 שחזור עיגול עם חור 68
4.8 שחזור מעושר עם חור מעושר 69
4.9 שחזור ריבוע עם שני חורים ריבועיים 70
4.10 שחזור שלא התכנס של צורה לא קשירה 71
4.11 שחזור מכונס של צורה לא קשירה 72

ב.1 דוגמאות של מצולעים שכבתיים בספריה polygs 79

תקציר

טומוגרפיה עוסקת ב*שחזור* פרוסה (חתך מישורי) בתוך גוף, מתוך מידע שמתקבל מקרני־X (קרני רנטגן). מכיוון שאנו עוסקים בשחזור פרוסות, נדבר מעתה על גופים מישוריים בלבד. קרן־X אחת נותנת לנו מידע על כמות המסה שעברה, ובשם *הטלה* בכיוון מסוים נקרא למידע שנובע מהקרנת גוף מכל הקרניים המקבילות לכיוון הנתון. נאמר שגוף מסוים ניתן *לשחזור יחיד* מהטלותיו, מבין משפחת גופים מסוימת, אם אין אף גוף אחר במשפחת הגופים שנותן את אותן הטלות בדיוק כמו הידועות.

רוב העבודות בנושא טומוגרפיה מניחות שפונקצית הצפיפות (שנותנת את הצפיפות בכל נקודה) היא כללית, והתוצאה היא שנדרשות הטלות ממספר אינסופי של כיוונים כדי להבטיח שחזור יחיד, או מספר סופי ורב של כיוונים (כ־180) כדי לאפשר שחזור בקרוב טוב תחת הנחות מסוימות. גישה נוספת היא להניח שהגוף בעל סימטריה מיוחדת (למשל, סימטריה גלילית), ואז ניתן לשחזר את פונקצית הצפיפות מהטלה אחת בלבד. שיטות טומוגרפיה אלו משמשות בהצלחה רבה באבחון רפואי (מכשיר CT) ובבדיקות־ללא־הרס בתעשיה.

בתחום של *טומוגרפיה גאומטרית* מניחים שהצורה אותה רוצים לשחזר מהטלותיה היא קבוצה *גאומטרית*, כלומר שפונקצית הצפיפות מקבלת את הערכים 0 או 1 בלבד. מכיוון שההנחה שלנו על פונקצית הצפיפות היא חזקה יותר מזו של טומוגרפיה כללית, אנו מצפים לקבל תוצאות חזקות יותר, ולהיות מסוגלים לשחזר קבוצות ממספר קטן של הטלות.

לא ניתן לשחזר קבוצות מהטלה בכיוון יחיד (קל לשנות קבוצה תוך שמירה על הטלה בכיוון אחד) ולכן צפוי היה לשאול מה אפשר לומר מהטלה בשני כיוונים. ללא הגבלת הכלליות נניח ששני הכיוונים ניצבים. התוצאה הראשונה בנושא זה היתה של לורנץ מ־1949, והוא דיבר על שחזור קבוצה מדידה מבין משפחת הקבוצות המדידות. המבנה של הקבוצות הניתנות לשחזור יחיד מבין הקבוצות המדידות ידוע לחלוטין: לורנץ הציג תנאי הכרחי ומספיק עבור זוג פונקציות להיות זוג הטלות מכיוונים ניצבים של איזשהי קבוצה מדידה, ותנאי הכרחי ומספיק להיות קבוצה זו יחידה. מאוחר יותר, נמצאו גם תנאים הכרחיים ומספיקים על קבוצה כדי לאפשר שחזור יחיד שלה מזוג הטלות: הוגדר מבנה הנקרא *רכיב־מיתוג*, שקבוצה ניתנת לשחזור יחיד מזוג הטלות ניצבות אם־ורק־אם היא אינה מכילה רכיב־מיתוג. הוגדרו שתי תכונות נוספות השקולות כל־אחת לאפשרות לשחזור יחיד מזוג הטלות ניצבות: קבוצה *בת־חסימה* וקבוצה *אדיטיבית*. שתי תכונות אלו עוזרות לבדוק ביתר קלות אם קבוצה מסוימת ניתנת לשחזור יחיד מבין הקבוצות המדידות. נמצאה גם נוסחה, שבהנחה שיש לזוג הטלות שחזור יחיד, מוצאת את שחזור זה.

תוצאות חלשות יותר ידועות לגבי שחזור קבוצות מדידות משלושה או יותר כיוונים: ידוע שאין קבוצת כיוונים סופית שמאפשרת שחזור יחיד של *כל* קבוצה מדידה, ולכן מעונינים, כמו קודם, בתנאים על קבוצות שמבטיחים שחזור יחיד בהנתן קבוצת כיוונים. אם מכלילים את הגדרת החסימה, האדיטיביות, והעדרות רכיב־מיתוג לקבוצת כיוונים סופית כלשהי, אפשר לראות שאלו תנאים מספיקים (אך לאו־דווקא הכרחיים) לשחזור יחיד של הקבוצה (מבין כל הקבוצות המדידות) מהטלותיה בכיוונים הללו. שוב, בעזרת משפטים אלו אפשר להראות בדוגמאות מסויימות שיש שחזור יחיד: לדוגמה, אליפסה נקבעת על־ידי

הטלותיה משלושה כיוונים.

אם מגבילים את עצמנו לשחזור קבוצות מתוך משפחה קטנה יותר, מצפים לקבל משפטים שונים, ואפשרות שחזור ממספר קטן של כיוונים במקרים רבים יותר. המר העלה ב־1961 את השאלה מתי ניתן לשחזר צורות מישוריות קמורות מתוך הטלותיהן. מאז פורסמו תוצאות רבות בתחום זה, למרות שנשארו גם בעיות פתוחות רבות. כמה תוצאות ידועות חשובות: ידוע תנאי הכרחי ומספיק לקבוצה סופית של כיוונים שמאפשרת שחזור יחיד של כל גוף קמור, ובפרט שרביעית כיוונים (שמקיימת תנאי מסוים) מאפשרת שחזור יחיד של כל גוף קמור. שלושה כיוונים או פחות לא מספיקים לשחזור <u>כל</u> גוף קמור, אך הוכח שבהנתן זוג כיוונים, *רוב* הקבוצות הקמורות (במובן של קטגוריה) ניתנות לשחזור יחיד. ידועים גם משפטים נוספים מסוגים שונים, לדוגמה: ניתן *לאמת* גוף קמור על־ידי הטלות בשלושה כיוונים (כלומר, בהנתן גוף קמור, אפשר לבחור שלושה כיוונים כך שכל גוף קמור אחר נותן שלישית הטלות שונה מזו של הגוף הנתון). לעומת זאת, קיים מצולע קמור (משושה) שלא ניתן לאימות בעזרת הטלותיו מכל זוג כיוונים.

לצערנו, מרבית שאלות היציבות של בעית השחזור הן פתוחות, ולכן במקרים רבים אין תשובה תאורטית לשאלה האם ניתן לשחזר, באופן מעשי, גופים מסויימים מכיוונים מסוימים. אולם למעשה מימשנו שני אלגוריתמים לשחזור קבוצות מהטלותיהן, שנתנו תוצאות יפות במספר רב של מקרים. לכן אנו משערים שבעתיד יתגלו משפטי יציבות תאורטיים חזקים יותר שיוכיחו את שימושיות האלגוריתמים הנ״ל.

אלגוריתם אחד שמימשנו הוא האלגוריתם של גרדנר לשחזור קבוצה קמורה מארבע הטלות. הוא עובד רק במקרים להם הוא נועד (לדוגמה, אי־אפשר להשתמש בו לשחזור גוף משתי הטלות), אך עובד יפה וביעילות במקרים אלו למרות שהצדקתו התאורטית אינה מלאה. אלגוריתם זה הוצע על־ידי גרדנר (ומומש על־ידי סטודנט שלו) אך לא פורסמו פרטים רבים הנחוצים למימוש. לכן נתקלנו במספר בעיות במהלך המימוש ואותן נאלצנו לפתור.

אלגוריתם שני שמימשנו הוא אלגוריתם חדש שלנו, שקראנו לו ״מינברס״, שפותר את בעית השחזור על־ידי מינימיזציה. אלגוריתם זה מנסה לשחזר צורה כוכבית (למעשה, הכללה שכוללת גם אפשרות לחורים או צורות לא קשירות) בהנתן מספר סופי של תוצאות של קרני־X כאשר הקרניים לאו־דווקא מכוונות בארבע זוויות שונות. הרעיון של אלגוריתם מינברס הוא כזה: לגוף ניחוש מסויים מוצאים את תוצאות ההטלות בקרניים הנתונות, ולוקחים נורמה שמודדת את מרחק וקטור התוצאות מוקטור התוצאות המבוקש. לנורמה זו מנסים לעשות מינימיזציה תוך שינוי הגוף, ומכריזים על הצלחה כשהתקרבנו מספיק לאפס.

ניתן להשתמש באלגוריתם מינברס גם במקרים בהם אין יחידות, ובמקרה זה מינברס מוצא את אחד השחזורים האפשריים. לדוגמה, ידוע שריבוע אינו ניתן לשחזור יחיד משתי הטלות ואכן מינברס מצא צורה נוספת — צורה כוכבית אך לא קמורה שלא הופיעה במאמרים קודמים. במקרים בהם ידועה תוצאה תאורטית שמבטיחה יחידות שחזור, מינברס אכן שחזר את הגוף למרות העדרו של משפט יציבות מתאים. כמו כן, במקרים רבים בהם לא ידוע משפט שמבטיח או פוסל יחידות, מינברס אכן מצא את הגוף המבוקש, דבר שגורם לנו לשער שקיימים משפטי יחידות שעדיין לא התגלו. בין הצורות ששיחזרנו עם מינברס היו צורות קמורות וכוכביות לא קמורות, צורות עם חור ושני חורים, ואפילו צורות לא קשירות.

לסיכום, בעבודה זו פיתחנו כלי שחזור חדש, מינברס, ומימשנו גם אלגוריתם ידוע של גרדנר. כלים אלו מראים שניתן לשחזר באופן מעשי קבוצות כוכביות (או הכללה שלהן) מהטלותיהן ממספר כיוונים קטן. תוצאות מעשיות אלו מצביעות על כך שהסיכוי לשחזור יחיד הוא אופטימי יותר מהנראה מהתוצאות התאורטיות הידועות כיום. אנו משערים שבעתיד יתגלו משפטים תאורטיים חדשים על שחזור יחיד ויציבות בעית השחזור שיסבירו מדוע מינברס פועל כה טוב.

סמלים וקיצורים

$f^{(p)}$	נגזרת p־ית של הפונקציה f
$u^{\perp}$	המרחב הניצב לכיוון u
$A\backslash B$	ההפרש בין הקבוצות A ו־B: $A\backslash B = \{x \in A \mid x \notin B\}$
$A\Delta B$	ההפרש הסימטרי של קבוצות A,B: $A\Delta B = (A\backslash B) \cup (B\backslash A)$
$p^{-1}(y)$	פונקציה הפוכה של $p(x)$
$f(x^{+})$	הגבול מימין $lim_{t\to x, t>x} f(t)$
$f(x^{-})$	הגבול משמאל $lim_{t\to x, t<x} f(t)$
α^{+}	$max(\alpha, 0)$
$g_x(y)$	ההופכי של Kuba & Volčič של הפונקציה $g(x)$
$h_y(x)$	ההופכי של Kuba & Volčič של הפונקציה $h(y)$
$P_{(t,u)}$	העתקה של הקבוצה P בווקטור (t, u)
∂U	שפת קבוצה U
$\bar{A}$	המשלים של קבוצה A
$\langle s_1, s_2, s_3, s_4 \rangle$	יחס־כפול (cross-ratio) של ארבעה שיפועים
$1_E(x)$	הפונקציה הקרקטריסטית של E (מקבלת ערך 1 על E וערך 0 מחוץ ל־E)
Å	Ångström, יחידת אורך ששוה ל־10^{-8} ס״מ
CAT	computerized axial tomography
conv$E$	קמור של הקבוצה E
CT	computed tomography
cl$E$	סגור של הקבוצה E
C^{∞}	משפחת הפונקציות הגזירות ברציפות אינסוף פעמים
D_g	התחום של הפונצה $g(x)$
δ	מטריקת האוסדורף
E	קבוצה מדידה וחסומה במישור
F	משפחת קבוצות או חלק מרכיב־מיתוג
f_u	פונקצית הטלה נתונה לכיוון u
F_{σ}	משפחת הקבוצות שהן איחוד בן־מניה של קבוצות סגורות
G	חלק מרכיב־מיתוג
G_{δ}	משפחת הקבוצות שהן חיתוך בן־מניה של קבוצות פתוחות
I	עוצמת הקרן אחרי שעברה דרך חומר — הקרן המוחלשת
int$E$	פנים E

K	קבוצה קמורה
K_0^2	מרחב הקבוצות הקמורות במישור, עם מטריקת האוסדורף
L_0^2	משפחת הפונקציות f שהן אפס מחוץ לקבוצה חסומה והאינטגרל $\int \lvert f\rvert^2$ הוא סופי
l_u	ישר דרך הראשית המקביל לכיוון u
λ_1	מידת לבג חד־מימדית
λ_2	מידת לבג דו־מימדית
m	שיפוע
n	מספר טבעי: מספר כיוונים
ϕ	טרנספורמצית שיקוף או טרנספורמציה אפינית הפיכה
$P(x)$	פונקציה אינטגרבילית ב־$(-\infty,\infty)$ אי־שלילית כמעט בכל מקום
$p(x)$	סידור לא־עולה של $P(x)$
Q	קבוצת המספרים הרציונליים או מצולע קמור
$Q(y)$	פונקציה אינטגרבילית ב־$(-\infty,\infty)$ אי־שלילית כמעט בכל מקום
$q(y)$	סידור לא־עולה של $Q(y)$
ρ	צפיפות
r	מרחק לאורך קרן או מרחק מציר
R	הישר
R^2	המישור
S	קבוצה סופית של כיוונים במישור
S_uE	סימטרל שטיינר: קבוצה קנונית שנותנת הטלה זהה לזו של E בכיוון u
T	משולש
θ	זווית
u	כיוון (וקטור יחידה במישור)
v	קדקד
V	קבוצה סופית של נקודות
x	(x,y) — זוג כיוונים ניצבים (כיווני הצירים)
X	קרני X (קרני רנטגן)
$X_uE(x)$	ההטלה של E בכיוון u
y	(x,y) — זוג כיוונים ניצבים (כיווני הצירים)

פרק 1

מבוא

1.1 מהי טומוגרפיה — סקירה היסטורית

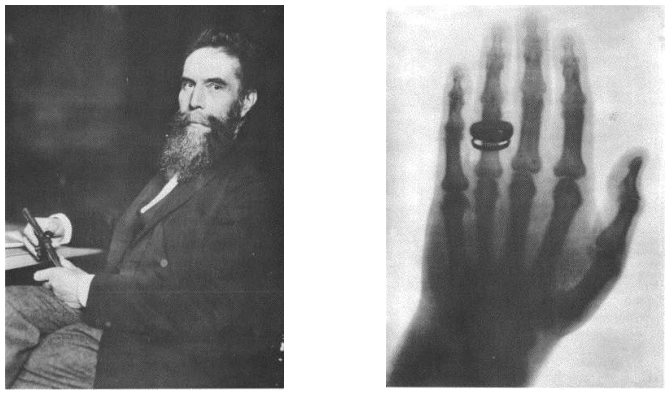
מקור המילה *טומוגרפיה* במילה היוונית $\tau\acute{o}\mu o\varsigma$, שמשמעותה פרוסה. טומוגרפיה היא הטכניקה הרדיולוגית של *שחזור* (מציאת) פרופיל צפיפות של גוף תלת־מימדי, תוך כדי התמקדות במישור מסוים בגוף. שחזור של מישור זה נעשה בעזרת אינפורמציה ממספר רב של קרני־X (חד מימדיות).

הסיבה שטכניקת הטומוגרפיה נחוצה היא שלא ניתן, בעזרת טכנולוגיה עכשיווית, להציץ לתוך גוף תלת־מימדי, ולראות *ישירות* את צפיפותו בנקודות פנימיות מסוימות. מכיוון שאי־אפשר לראות נקודות פנימיות ישירות, מדענים נאלצו למצוא שיטות פחות ישירות להצצה לתוך גופים מוצקים. הצורך בהצצה לתוך מוצקים הוא ברור, וכולל הדמיה רפואית (למציאת גידול והערכת צפיפותו, למשל), ושמושים תעשיתיים (לחיפוש סדקים בחלקים מיוצרים, למשל). ברור שהאלטרנטיבה של חדירה פיזית לגוף אינה מעשית או מאוד יקרה במירב המקרים, כמו חיפוש גידולי מוח, או בדיקת המצאות סדקים במוטות מתכת שיוצאים מפס היצור ללא להרוס אותם על־ידי חיתוך.

הפתרון הראשון לבעיית ההצצה לתוך גופים מוצקים נולד כאשר התגלו קרני ה־X ב־8 בנובמבר, 1895, על־ידי Wilhelm Conrad Röntgen (1845-1923), אז פרופסור לפיזיקה באוניברסיטת Würzburg שבגרמניה. בערבו של יום שישי, בזמן שביצע ניסויים בזרימת זרם חשמלי בשפופרת זכוכית מרוקנת חלקית (שפופרת קרן קתודית), רנטגן שם לב שחתיכת בריום פלטינוציאניד שהיתה מונחת בצד מאירה כאשר השפופרת פעלה. הוא העלה השערה שכאשר הקרניים הקתודיות (אלקטרונים) פגעו בקיר הזכוכית של השפופרת, נוצרה איזושהי קרינה לא ידועה ועברה לאורך החדר, פגעה בכימיקל, וגרמה לפלואורסנציה. בדיקות נוספות גילו שנייר, עץ ואלומיניום, בין השאר, הם שקופים לסוג חדש זה של קרינה. הוא מצא שהיא משפיעה על פלטות צילום, ומכיוון שהיא לא הראתה תכונות של אור, כמו החזרה ושבירה, הוא הסיק בטעות שקרניים אלו לא היו קשורות לאור. בגלל הטבע הלא־ברור שלה, הוא קרא לתופעה קרינת־X, למרות שהיא נהיתה ידועה גם בשם קרינת רנטגן. "חשיפות הרנטגן" הראשונות שהוא עשה היו של חפצי מתכת הנעולים בתוך קופסת עץ, ושל עצמות כף־ידה של אשתו (ראה איור 1.1).

הידיעה על תגליתו של רנטגן היתה מלהיבה, ומיד פורסמה בעיתונים היומיים. ב־13 בינואר 1896 נתבקש רנטגן להדגים את תגליתו בפני הקיסר בארמונו בברלין. ב־23 בינואר הרצה לראשונה רנטגן על תגליתו בפומבי, בפגישה של האגודה הפיזיקלית־רפואית. הקהל הריע ומחא כפיים כאשר רנטגן צילם את ידו של Geheirmat von Kölliker, אנטומאי ידוע.

טבען המדויק של קרניים אלו נותר תעלומה, עד אשר ב־1906 הפיזיקאי הבריטי Charles Glover Barkla הראה שכאשר קרני X מפוזרות בכיוונים מסויימים על ידי גוש פחמן הן נהיות מקוטבות. מניסויים



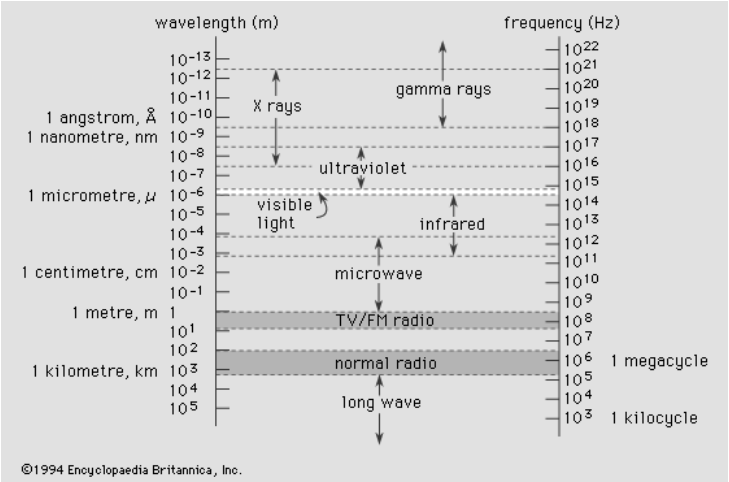
איור 1.1: Röntgen ותצלום קרני ה־X הראשון שלו
Figure 1.1: Röntgen and his first X-Ray exposure

אלו נראה סביר שקרני X הנן גלים אלקטרומגנטיים, בדומה לאור נראה אך באורך גל שונה. ניסויים שנעשו ב־1912 על־ידי הפיזיקאי הגרמני Max Theodor Felix von Laue קבעו שקרני X הן גלים עם אורך גל מסדר גודל של אנגסטרום אחד. עכשיו, אנו קוראים "קרני X" לכל קרינה אלקטרומגנטית עם אורכי גל מ־0.05Å עד בערך 500Å (אנגסטרום, שמסומן ב־Å כקיצור ל־Ångström, היא יחידת אורך ששוה ל־10^{-8} ס"מ). ראה איור 1.2 שמתאר את כל הספקטרום של הגלים האלקטרומגנטיים.

מכיוון שאורך הגל של קרני X קצר מזה של אור נראה, קרני X יכולות לעבור דרך חומרים רכים, כמו רקמה רכה ובגדים, אך הן נבלעות על־ידי חומרים צפופים כגון עצם ומתכת. כדי לבצע צלום קרני X שמים את האוביקט (למשל, אדם) בין מקור פולס של קרני X לבין סרט צילום מיוחד. עוצמת ההשחרה על הסרט נקבעת על־ידי עוצמת הקרן שהצליחה לעבור דרך החומר – הקרן המוחלשת (attenuated), שפרופורציונית ביחס הפוך לאקספוננציאל של כמות המסה שנמצאת בדרכה של הקרן ($I = I_0 e^{-\int \rho dr}$). ובכן תמונות קרני X נותנות *הטלה* של האוביקט, שבה נקודות שנפגעו על־ידי קרן שעברה דרך אזורים צפופים תהיינה לבנות, ואילו נקודות שנפגעו על־ידי קרן שעברה דרך אזורים צפופים פחות תהיינה שחורות (כלומר התמונה באיור 1.1 היא פוזיטיב עם גוונים הפוכים לאלו על סרט הצילום עצמו — הנגטיב). למען האמת, תאור זה של האינטראקציה של קרני X עם חומר היא פשטנית מדי, ומתעלמת מהספקטרום של אלומת הקרניים ומסוגי האטומים שמרכיבים את החומר. תאור מלא יותר ניתן למצוא ב־[17, 154-157].

חלק מאיברי הגוף, כגון הריאות, נראות ברדיוגרמות (כלומר, תמונות קרני X) בזכות הניגוד החד, או ה*קונטרסט*, בין בליעת קרני ה־X של האוויר בתוכם לרקמה הריאתית עצמה. הלב, שמורכב ברובו משרירים ודם, יוצר ניגוד חזק עם הריאות המלאות אוויר שלידו, אך כמעט אין ניגוד עם הכבד שמתחתיו. עצמות נבדלות מהשריר שמסביבם בגלל הסידן הזרחתי שהן מכילות. אך לרוב השימושיות הקלינית של בדיקת קרני ה־X מסתמכת על השימוש באמצעי ניגוד מלאכותיים. החומר האטום הנפוץ ביותר בשימוש הוא בריום גפרתי. כאשר הוא מעורבב במים, ובדרך כלל בחומרי טעם, מלח מתכת לא מסיס זה נבלע על־ידי הנבדק בשביל בדיקת הוושת והקיבה. הוא גם משמש כחוקן בריום בשביל בדיקת פי־הטבעת והמעי הגס. תרכובות אורגניות עם יוד משמשות בבדיקת כיס המרה, דרכי השתן, כלי הדם, הטחול, הכבד, ודרכי המרה.

אך אפילו עם השימוש באמצעי ניגוד, לתמונת קרני X פשוטה עדיין יש חסרונות. מבנים יכולים להיות מוסתרים על־ידי איברים שמעליהם, וקשה להבדיל בין רקמות רכות שלא ידוע אמצעי ניגוד שנכנס רק אל



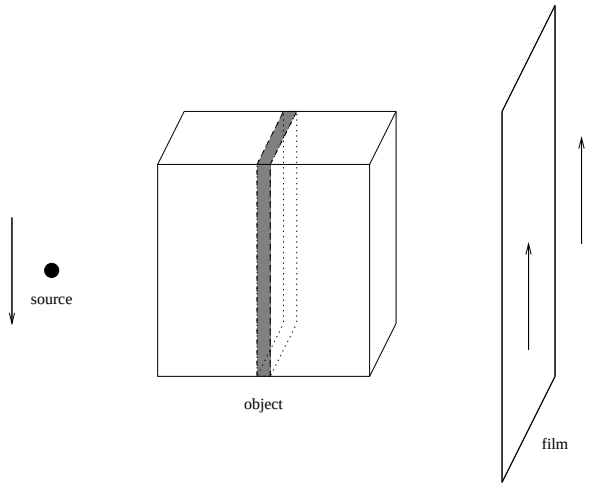
איור 1.2: הספקטרום האלקטרומגנטי
Figure 1.2: The Electromagnetic spectrum

אחד מהם. לכן עלה הצורך בטומוגרפיה. **טומוגרפיה מנסה למצוא את פרופיל הצפיפויות על מישור אחד בתוך הגוף, במקום לסכם את הצפיפויות בכל במישורים כפי שנעשה בתמונת רנטגן רגילה.**

1.2 סוגי טומוגרפיה

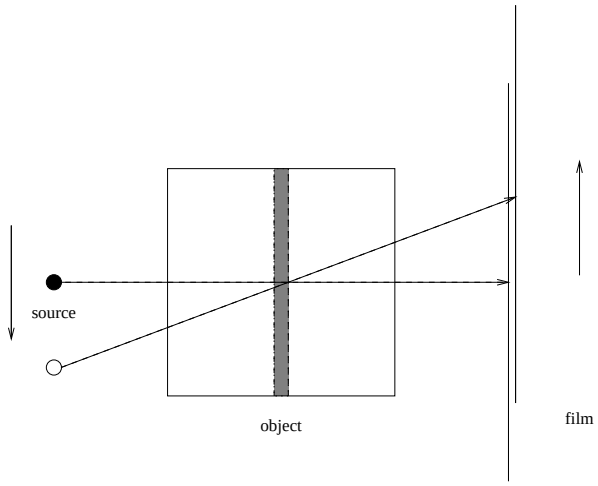
שיטת הטומוגרפיה הפשוטה ביותר היא *טומוגרפיה קווית*, שלפעמים נקראת גם *טומוגרפיה קונבנציונלית*, או *טומוגרפיה גאומטרית* (אותה אין לבלבל עם הנושא של חיבור זה, בו למילה *גאומטרית* משמעות שונה לחלוטין). בשיטה זו, מקור קרני ה־X ולוח הצילום מוזזים בכיוונים הפוכים במישורים מקבילים בצורה מתואמת (ראה איור 1.3). לדוגמה, אם המקור והסרט מוזזים באותה מהירות (בכיוונים הפוכים), אז לחתך מישורי של החולה באמצע הדרך ביניהם תהיה הטלה שנעה עם הסרט. חתך מישורי זה ישאר ממוקד על הטומוגרמה (סרט הצילום), ואילו שאר הרקמות יהיו מטושטשות. באיור 1.4 ניתן למצוא הסבר סכימטי מדוע תמונה של נקודה מסויימת על מישור האמצע (באפור) נשארת על אותה נקודה בסרט, כאשר הסרט והמקור נעים במהירות קבועה והפוכה.

בשיטת המקורית, התנועה המתואמת של המקור והסרט היא לאורך קו ישר. כצפוי, האיזור של הגוף שנשאר ממוקד ויש לו עובי חתך מסויים, קטן ככל שגודל ההעתק הזוויתי (*הזווית הטומוגרפית*) גדל. למרות שטומוגרף כזה מספק בדרך כלל רזולוציה מרחבית טובה, ניגוד התמונה קטן עם הגדלת העובי של החולה או העלאת הזווית הטומוגרפית בגלל טשטוש. הטשטוש יכול לגרום גם להופעת תמונות לא אמיתיות ($phantom\ images$). בעיות אלו עודדו שימוש במסלולי תנועה מישוריים אחרים, כגון מעגל, אליפסה, ספירלה והיפוציקלואידה (שיטות אלו נקראות לפעמים *טומוגרפיה רב־כיוונית*). למרות השיפור, בטומוגרפיה קונבנציונלית הניגוד נשאר קטן, אך שיטת זו נשארה שימושית בבדיקות עם ניגוד גבוה (כגון עצמות, דרכי הנשימה, או חומר ניגוד מלאכותי) בזכות הרזולוציה המרחבית המצוינת שהיא מספקת. שיטות טומוגרפיה אלו שימשו לבדיקת הכליות ומבנים אחרים בבטן שמוקפים ברקמות עם צפיפות כמעט זהה ולכן לא ניתנים להפרדה בבדיקות קרני X רגילות. הם שימשו גם לבדיקת העצמות הקטנות ומבנים אחרים באוזן, שמוקפים בעצם הרקה הצפופה יחסית.



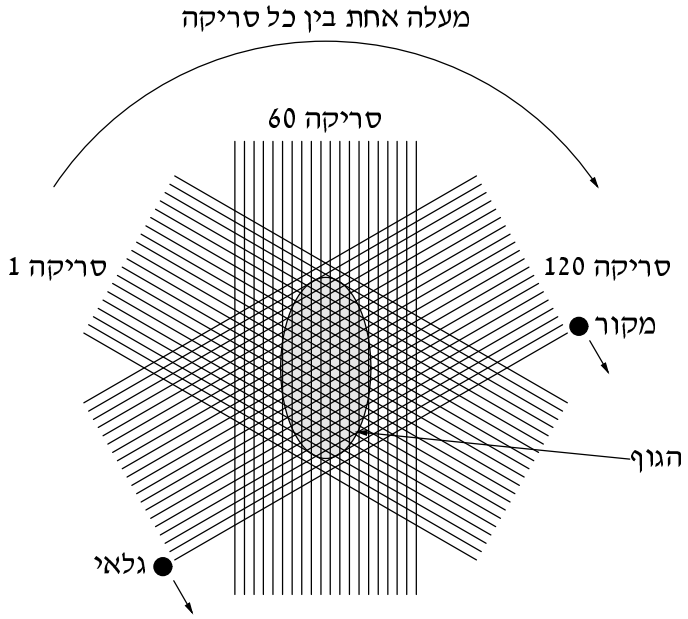
איור 1.3 : טומוגרפיה קווית : תאור סכימטי של השיטה

Figure 1.3: Linear Tomography: schematic depiction of the method



איור 1.4 : טומוגרפיה קווית : תאור סכימטי של השיטה — חתך במישור ניצב

Figure 1.4: Linear Tomography: schematic depiction of the method — perpendicular section



איור 1.5: פעולת מכשיר CT
Figure 1.5: Operation of a CT machine

שיטה מסובכת יותר היא*טומוגרפיה מחושבת*, שנקראת באנגלית בשמות computed tomography (CT) או computerized axial tomography (CAT) (ולעיתים אפילו computerized tomography או computer aided tomography). שיטה זו פותחה בראשית שנות השבעים על־ידי Godfrey New-bold Hounsfield הבריטי ו־Allen Cormack האמריקני, ומאז הפכה לשיטה נפוצה לדיאגנוזה רפואית. טומוגרפיה מחושבת נותנת ניגוד מצוין, אך רזולוציה מרחבית צנועה.

בשיטת הטומוגרפיה המחושבת, מתבצעות *הטלות* מכיוונים רבים. *הטלה* (projection) הוא מושג שנפגוש אותו לכל אורך עבודה זו: כל הטלה היא למעשה סריקה של אלומה צרה של קרני X בכיוון מסויים, הנעה בניצב לכיוונה, ועוברת על פני שטח מהגוף. איור 1.5, מתאר מכשיר מהדור־הראשון של חברת EMI שבה עבד Hounsfield, כפי שמתואר ב־[17]. למעשה, בדור החדש של המכשירים, אין תנועה העתקית, אלא רק סיבוב: קרינת־X ממקור קטן מתפרשת למניפה מישורית (על־ידי קולימציה שמגבילה רק קרינה מחוץ למישור), ומגיעה למספר רב של גלאים. המקור מסתובב, ומעגל הגלאים נשאר במקום. בכל זווית של המקור מקבלים מהגלאים נתונים על מספר רב של כיוונים, ומחשב מסדר את הנתונים בצורה שרוצים, לפי הכיוונים. ביטול התנועה ההעתקית קיצר מאוד את זמן הפעולה של מכשיר ה־CT. חשוב לקצר את זמן הפעולה של המכשיר גם כדי למנוע בעיות שנובעות מתנועת החולה בזמן הבדיקה.

הקרינה שעוברת דרך הגוף נמדדת על־ידי גלאי קרינה ומועברת למחשב. המחשב משתמש בטרנספורמציה מיוחדת (טרנספורם רדון הפוך) לשחזור פרופיל הצפיפות בחתך המישורי מתוך המידע על הטלות אלו ממספר רב של כיוונים. בגלל היותו שימושי כל־כך לדיאגנוזה רפואית, טרנספורם רדון ואפשרות השחזור של פילוג צפיפות כללי מתוך הטלותיו מהרבה כיוונים הוא נושא נחקר ומפותח. סיכום של חלק מהתוצאות הידועות אפשר למצוא ב־[17]. לדוגמה: J. Radon הראה ב־1917 שתחת הנחה מסוימת על פונקצית הצפיפות (שהיא C^∞ ויורדת מהר[1]), f נקבעת על־ידי הטלותיה ברצף הכיוונים $0 \leq \theta \leq 180°$. מאוחר יותר הוכחה טענה חזקה יותר: אם f היא L_0^2, כלומר f היא אפס מחוץ לקבוצה חסומה במישור והאינטגרל

[1] כאן הכוונה שלכל $k > 0$ ולכל p שלם חיובי מתקיים: $lim_{|z|\to\infty} |z|^k f^{(p)}(z) = 0$

$\int_{R^2} |f|^2$ הוא סופי, אז f נקבעת על־ידי הטלותיה מקבוצה בת־מניה של כיוונים. אולם, ב־1973 הוכיח K. T. Smith שקבוצה סופית של כיוונים לא קובעת את f. למרות זאת, מספר רב של משפטים ואלגוריתמים פותחו לאחר מכן כדי לשחזר את פונקצית הצפיפות מההטלות ממספר סופי של כיוונים, תחת הנחות מסוימות על פונקצית הצפיפות ו/או בחירת פונקצית הצפיפות ממרחב פונקציות בעל מימד סופי. לא נרחיב על נושא זה בעבודה זו, מכיוון שאנו מעוניינים לדבר על הנחות שונות לחלוטין על מבנה פונקצית הצפיפות ועל מספר ההטלות, כפי שנראה בפרק הבא.

לשיטת הטומוגרפיה המחושבת, למרות שהיא מספקת חתכי צפיפויות מדויקים, יש את החסרונות שלה. הציוד הנדרש הוא מאוד מסובך ויקר (נדרשים מכונה מסובכת ומחשב חזק מיוחד), והשיטה חושפת את החולה לכמות גדולה של קרינה. סריקת CAT חושפת את החולה ל־50–100 מיליגריי לבדיקה, בהשוואה ל־0.4–10 מיליגריי לצילום קרני X רגיל (מיליגריי, היא יחידה של מנת קרינה נספגת. היא מוגדרת כ $1/1{,}000$ ג'אול של אנרגית קרינה שנספגת בקילוגרם של רקמה).

לבדיקות ללא הרס (non-destructive evaluation) תעשיתיות, ולפעמים גם בהדמיה רפואית לבני אדם, אין צורך בפרופילי צפיפות מדויקים, ומסתפקים בהערכת מיקומו וגודלו של פגם. לכן אנו מתעניינים גם בטרנספורמים אחרים, שבניגוד לטרנספורם רדון ההפוך יצטרכו רק שיקופים ממעט כיוונים לשחזור מספיק טוב של פרופיל הצפיפות בחתך, בהנחת כמה הנחות על הגוף הנבדק. מקרה שימושי במיוחד הוא טרנספורם אבל (Abel). טרנספורם אבל מניח שלגוף סימטריה גלילית, בעל פונקצית צפיפות $f(r)$ שתלויה רק ברדיוס מהציר, r, ומחשב את ההטלה של הגוף על־ידי קרניים מקבילות בכיוון יחיד. אפשר להראות (וזה גם די אינטואיטיבי) שיש טרנספורם אבל הפוך: ניתן לשחזר את הגוף בעל הסימטריה הגלילית מהטלה בכיוון אחד. תאור הבעיה ודוגמה לאלגוריתם לטרנספורם אבל ההפוך ניתן למצוא למשל ב־[14]. רוב העבודות על טרנספורם אבל עוסקות בשיפור מהירות השחזור, בהפיכתו לפחות רגיש לשגיאות מדידה, ובהנחות מיוחדות (למשל, גוף גלילי עם פונקצית צפיפות רדיאלית קבועה למקוטעין). עבודות נוספות, כגון [13], מדברות על שחזור גוף בעלת סימטריה כללית יותר בעזרת הטלה אחת בלבד: לדוגמה, שחזור גוף הבנוי מקליפות אליפטיות שוות־צפיפות.

1.3 העבודה הנוכחית

העבודה הנוכחית עוסקת בתחום נוסף הקשור לטומוגרפיה: *טומוגרפיה גאומטרית*. תחום זה עוסק בשחזור צורות גאומטריות (במקרה שלנו, צורות במישור), מהטלותיהן. נניח שפונקציות הצפיפות על צורות אלו הן קבועות (והקבוע ידוע) ונראה שבמקרה זה אפשר לקבל תוצאות חזקות שבתנאים מסוימים מבטיחות שחזור יחיד של הגוף ממספר קטן של הטלות. יתר על כן, הנחות נוספות על הגוף, כגון קמירות, מאפשרות לקבל תוצאות חזקות עוד־יותר. למרות העבודה הרבה שנעשתה בתאוריה של הטומוגרפיה הגאומטרית, עדיין יש הרבה בעיות פתוחות בנושא ובמקרים רבים לא ידוע האם מובטח שחזור יחיד או האם מובטחת יציבות. בעבודה זו נראה שני אלגוריתמים שמאפשרים שחזור צורות בעלות צפיפות־קבועה מהטלותיהן במספר קטן של כיוונים, ונדגיש שאלגוריתמים אלו פועלים למעשה, גם כשהתאוריה לא יודעת להבטיח יחידות או יציבות. לכן אנו מצפים שבעתיד אפשר יהיה לקבל תוצאות תאורטיות הרבה יותר "אופטימיות" מאלו שידועות היום.

פרק 2 יוקדש לסקר התוצאות הידועות בתחום המעניין אותנו בטומוגרפיה גאומטרית (טומוגרפיה גאומטרית מישורית, של הטלות עם קרניים מקבילות). בנוסף נביא בפרק זה דוגמה מענינת (*דוגמת הריבוע*) של צורה שאיננה ניתנת לשחזור משתי הטלות. דוגמה זו הובאה כבר בעבר במאמרים שונים, אך אנו מצאנו צורה נוספת, לא קמורה, שנותנת את אותן הטלות, ולא הוזכרה בעבר.

פרק 3 יתאר את אלגוריתם גרדנר לשחזור של גוף קמור מהטלותיו מארבעה כיוונים. אלגוריתם זה

תואר לראשונה על־ידי גרדנר אך מימושו אינו פשוט כלל. אנו נציג מימוש של האלגוריתם, את הבעיות בהן נתקלנו במהלך המימוש, ונביא דוגמאות לשימוש באלגוריתם.

פרק 4 יתאר אלגוריתם חדש, "מינברס", שפיתחנו לשחזור של גופים ממספר כלשהו של כיוונים. אנו נדגים את גמישות האלגוריתם, לסוגים שונים של גופים (קמורים, לא קמורים, לא קשירים, גופים עם חור, וכו.) וממספר שונה של הטלות.

פרק 5 נותן סיכום של המחקר ותוצאותיו.

נספח א' מתאר את אלגוריתם המינימיזציה שבו השתמשנו בשני האלגוריתמים מהפרקים הקודמים (גרדנר ומינברס), ונותן מבוא קצר לשיטות מינימיזציה רב־מימדיות בכלל.

נספח ב' מתאר כיצד להשתמש בתוכניות השחזור לפי האלגוריתמים מינברס וגרדנר, ונספחים ג' ו־ד' מכילים את הדפסת תוכניות אלו.

פרק 2

טומוגרפיה גאומטרית

2.1 מבוא

מכשיר CT (והכלי המתמטי המשמש בו: טרנספורם רדון ההפוך) משתמש בנתונים ממספר רב של הקרנות בזוויות שונות לשחזור פרוסות דו־מימדיות של הגוף. אבל, ידוע שלא יתכן שחזור יחיד ממספר סופי של כיווני הקרנה (ראה למשל [17]), ולכן ה־CT משחזר רק קרוב של הגוף.

אם נניח יותר על הגוף — לא גוף בעל מפת צפיפות כללית אלא גוף בעל צפיפות אחידה — נוכל לקוות שהנחה כזו תאפשר שחזור גופים ממספר סופי של הטלות — אולי אפילו מספר קטן של הטלות. ההנחה שלגוף צפיפות קבועה התבררה כהנחה הגיונית במספר רב של שימושים, ברפואה (למשל [3, 4]), ברובוטיקה, במכ״מ, ועוד. יתר על כן, הנחה זו מתאימה גם לשחזור צורת חור (אזור בצפיפות אפס) בגוף בעל צפיפות אחידה מכיוון שהצורה החיצונית של הגוף ניתנת למדידה ישירה, ועל־ידי החסרת ההטלות המדודות מההטלות שהיו אמורות להיות לגוף המלא, מקבלים את ״הטלות״ החור.

בעבודה הנוכחית נתרכז בשחזור צורות מישוריות, ונניח שההטלות מתבצעות על־ידי קרניים מקבילות (מבחינה אופטית הנחה זו שקולה להנחה שהקרניים מגיעות ממקור באינסוף). נציין כי בנושא הרחב של טומוגרפיה גאומטרית כלולים גם נושאים נוספים, כמו גופים רב־מימדיים והקרנות ממקור נקודתי. ספר מצוין על הנושא כולו הוא ספרו של Gardner [11].

בהגדרות הבאות E תסמן קבוצה מדידה וחסומה במישור.

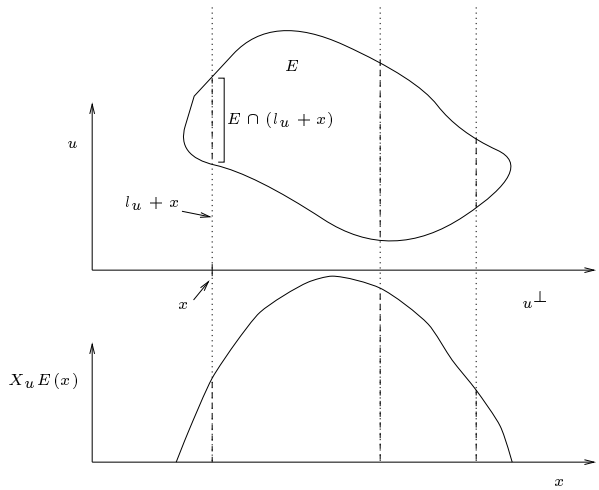
הגדרה 1 *יהיה u כיוון (וקטור יחידה) במישור. ה<u>הטלה</u> (projection) בקרניים מקבילות של E בכיוון u היא פונקציה, שנסמנה ב־$X_uE(x)$, שמוגדרת כמעט־לכל־ (לפי מידת לבג החד־מימדית λ_1) $x \in u^\perp$. $X_uE(x)$ מוגדרת לפי:*

$$X_uE(x) = \lambda_1\left(E \cap (l_u + x)\right)$$

כאשר l_u הוא ישר דרך הראשית המקביל ל־u.

בהנחה ש־E מדידה ובעלת מידה סופית, משפט פוביני מבטיח ש־$X_uE(x)$ אכן מוגדרת כמעט לכל x.

בהמשך עבודה זו נשתמש במילה ״הטלה״ במשמעות זו, ואין לבלבל אותה עם משמעות אפשרית אחרת למילה ״הטלה״ שהיא אורך הצל של הגוף בכיוון מסוים. ראה איור 2.1 להמחשת ההגדרה.



איור 2.1 : הטלה של קבוצה E
Figure 2.1: Projection of a set E

אפשר גם להגדיר את ההטלה בעזרת <u>טרנספורם ההקרנה</u> (X-ray transform) שמוגדר לפונקציה מדידה במישור בצורה הבאה :

$$X_u f(x) = \int_{-\infty}^{\infty} f(x+tu)dt \tag{2.1}$$

בסימון זה, ההטלה $X_u E(x)$ ניתנת לכתיבה כ־

$$X_u E(x) = X_u 1_E(x) \tag{2.2}$$

כאשר $1_E(x)$ היא הפונקציה הקרקטריסטית של E (כלומר, הפונקציה שמקבלת ערך 1 על E וערך 0 מחוץ ל־E).

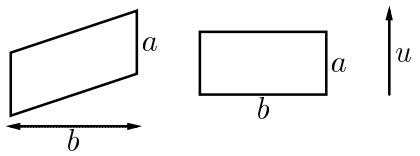
אם מניחים שהקבוצה E היא קמורה, אז חיתוך הקבוצה עם הקרניים יהיה תמיד קטע, ולכן במקרה זה אפשר להסתפק במושגים בסיסיים של אורך ואין צורך בתורת המידה. כפי שנראה בהמשך, הגבלת הדיון לקבוצות קמורות בלבד נותנת משפטים מעניינים.

הגדרה 2 *קבוצה E היא <u>שחזור</u> של פונקציות הטלה נתונות f_u לקבוצת כיוונים S אם לכל $u \in S$ מתקיים $X_u E = f_u$.*

הגדרה 3 *קבוצה E השייכת למשפחת קבוצות F (לדוגמה, משפחת הקבוצות הקמורות) <u>נקבעת</u> (determined) ב־F על־ידי הטלותיה בקבוצת כיוונים S אם לכל $E' \in F$ שמקיים $X_u E' = X_u E$ לכל $u \in S$ נובע ש־$E' = E$. כל שוויוני הפונקציות כאן ובהמשך הם, כמובן, כמעט־בכל־מקום לפי המידה הרלוונטית, ושיוויוני הקבוצות הם עד כדי קבוצה בעלת מידה אפס: $A = B$ אומר $\lambda_2(A \Delta B) = 0$.*

במקרה זה נאמר גם ש־E ניתנת ל<u>שחזור יחיד</u> (unique reconstruction) מבין הקבוצות ב־F בעזרת ההטלות בכיוונים S.

הגדרה 4 *קבוצה E השייכת למשפחת קבוצות F <u>ניתנת לאימות</u> (can be verified) ב־F אם ניתן לבחור קבוצת כיוונים סופית S (התלויה ב־E) כך שאם $E' \in F$ ו־ $X_u E = X_u E'$ לכל $u \in S$ אז $E = E'$.*



איור 2.2: שתי קבוצות (קמורות) עם הטלה זהה בכיוון אחד u

Figure 2.2: Two (convex) sets with an identical projection in the u direction

הגדרה 5 *קבוצה E השייכת למשפחת קבוצות F ניתנת לקביעה בשלבים (can be successively determined) אם ניתן לבחור כיוונים $u_1...u_m$ בשלבים, כשבחירת u_i תלויה רק בהטלות הקודמות $X_{u_1}E, ..., X_{u_{i-1}}E$, כך שאם $E' \in F$ ו־ $X_{u_i}E = X_{u_i}E'$ לכל $i = 1...m$ אז $E = E'$.*

קל להבין שהטלה בכיוון אחד u אינה מספיקה כדי לקבוע אף קבוצה לכל משפחה שימושית של קבוצות (כגון, משפחת הקבוצות המדידות החסומות, משפחת הקבוצות הקמורות, משפחת הקבוצות הכוכביות, וכו.). זאת מכיוון שניתן לקחת את הקבוצה המקורית, "למשוך" קטעים בכיוון u ולקבל קבוצה חדשה, כמו בדוגמה באיור 2.2.

מסיבה זו נוהגים להגדיר לקבוצה E קבוצה קנונית S_uE (הנקראת סימטרל שטיינר) עם הטלה זהה בכיוון u:

הגדרה 6 *תהי E קבוצה כלשהי במישור, ויהי u כיוון. יהיה l_u הישר דרך הראשית המקביל ל־u. לכל $x \in u^\perp$, נגדיר $c(x)$ כך: אם $(l_u + x) \cap E$ ריק או לא λ_1־מדיד, נגדיר $c(x) = \emptyset$. אחרת, נגדיר $c(x)$ כקטע הסגור המקביל ל־u שמרכזו ב־x ואורכו $\lambda_1(E \cap (l_u + x))$. האיחוד של כל הקטעים $c(x)$ נקרא סימטרל שטיינר (Steiner Symmetral) של E בכיוון u, ומסומן ב־S_uE.*

מהטלה בכיוון אחד של קבוצה E ניתן להוציא רק מידע מוגבל — והחשוב ביותר הוא השטח של E, שניתן לחישוב כ־ $\lambda_2(E) = \int_{u^\perp} X_uE(x)dx$.

מכיוון שכיוון אחד לא מעניין, המשפטים המעניינים מתחילים עם שני כיוונים.

2.2 שחזור קבוצות מדידות עם מידה סופית

בסעיף זה נעסוק בשאלה האם קבוצה מדידה עם מידה סופית ניתנת לשחזור יחיד מבין משפחת הקבוצות המדידות בעלות מידה סופית בעזרת קבוצות כיוונים שונות. כפי שנראה, במקרים שונים ישנם משפטים שעונים על שאלה זו. נראה תנאים הכרחיים ומספיקים להיות קבוצה ניתנת לשחזור יחיד. יתר על כן, נראה גם משפטים שבהנתן פונקציות הטלות בודקים האם ישנו שחזור יחיד (ואם כן, מוצאים את השחזור), האם ישנו יותר משחזור אפשרי אחד, או אולי אין כלל קבוצה שנותנת הטלות אלו.

לחלק גדול מהתוצאות שיוזכרו יש גם גרסאות רב מימדיות, שמופיעות במאמרים שיוזכרו או בספרו של גרדנר [11]. אנו נתרכז בשחזור קבוצות מישוריות בלבד.

יש לזכור שבפרק זה, בכל פעם שידובר על *שחזור יחיד*, הכוונה היא לשחזור יחיד מבין כל הקבוצות המדידות.

2.2.1 הטלות בשני כיוונים

כפי שהוסבר, אף קבוצה לא ניתנת לשחזור יחיד בעזרת כיוון אחד בלבד. לכן השאלה הבאה שניתן לשאול היא מה אפשר לומר בעזרת הטלות בשני כיוונים שונים. כדי לפשט את הדיון, נהוג להניח ללא הגבלת

הכלליות ששני הכיוונים הם שני כיווני הצירים. הסיבה לכך היא שאפשר להביא כל זוג כיוונים (ואת הקבוצה הנדונה) למצב הרצוי על־ידי טרנספורמציה לינארית הפיכה מתאימה.

מלבד העניין התאורטי בנושא, לשחזור צורות מהטלותיהן בזוג כיוונים ניצבים נמצאו גם שימושים מעשיים, כגון שחזור חתכים של הלב משתי הטלות בכיוונים ניצבים [3, 4].

הנושא של יחידות שחזור של קבוצה מישורית (מדידה עם מידה סופית) משתי הטלות בכיווני הצירים נחקר מאז מאמרו פורץ־הדרך של לורנץ [21] מ־1949, והמאמרים החשובים האחרים שטיפלו בנושא, [18, 7] הובילו לכך שגרדנר כותב במאמרו [9] ש״כשישנם רק שני כיוונים, המבנה של קבוצות [שניתנות לשחזור יחיד מבין הקבוצות המדידות] ברור לחלוטין״. בסעיף זה נציג את המשפטים שהביאו את גרדנר להכרזה זו.

לורנץ [21] הסתכל על זוג פונקציות אינטגרבילות ב־$(-\infty,\infty)$ אי־שליליות כמעט בכל מקום, $P(x)$ ו־$Q(y)$, ושאל מתי זוג פונקציות זה יכול להיות פונקציות ההטלה בכיווני הצירים של קבוצה מדידה כלשהי (בעלת מידה סופית), מתי קבוצה זו יחידה, ומתי זוג זה אינו יכול כלל להיות זוג הטלות בכיווני הצירים של אף קבוצה.

כדי לנסח את תוצאתו של לורנץ, יש צורך בהגדרה הבאה ובמשפט (שהופיע במאמרו של ריץ [24]):

הגדרה 7 *פונקציות מדידות $P(x)$, $p(x)$ המוגדרות על קבוצות מדידות E, e בהתאמה, נקראות <u>בעלות פילוג זהה</u> (equimeasurable) אם הקבוצות $\{x\in e|\alpha\le p(x)\}$, $\{x\in E|\alpha\le P(x)\}$ הן בעלות מידה שווה לכל α ממשי.*

משפט 1 *תהיה $P(x)$ פונקציה אי־שלילית אינטגרבילית המוגדרת ל־ $-\infty<x<\infty$. קיים <u>סידור לא־עולה</u> של $P(x)$, שהוא פונקציה $p(x)$, $0<x<\infty$, עם פילוג זהה לזה של $P(x)$ ושמקיימת $p(x_1)\ge p(x_2)$ עבור $x_1\le x_2$.*

כמו כן, אנו צריכים את ההגדרה הבאה לפונקציה הפוכה של סידור לא־עולה:

הגדרה 8 *תהיה $p(x)$, $x>0$ פונקציה אי־שלילית לא־עולה בעלת אינטגרל סופי ב־$(0,\infty)$. נגדיר את הפונקציה ההפוכה $p^{-1}(y)$ בצורה הבאה:*

- *אם x_0 נקודת רציפות של $p(x)$, והוא ה־x היחיד בו p מקבלת ערך $y_0=p(x_0)$ זה, נגדיר כצפוי $p^{-1}(y_0)=x_0$.*
- *אם (α,β) קטע בו הפונקציה $p(x)$ בעלת ערך קבוע y_0, נגדיר את $p^{-1}(y_0)$ להיות ערך שרירותי בין α ל־β.*
- *אם x_0 נקודת אי־רציפות של $p(x)$, נגדיר את $p^{-1}(y)$ להיות קבוע x_0 בקטע $p(x_0^-)\le y\le p(x_0^+)$ (כאשר ב־$p(x_0^-)$ ו־$p(x_0^+)$ הכוונה לגבולות משמאל ומימין בהתאמה).*
- *אם y כזה ש־$y>p(x)$ לכל $x>0$ (כלומר, y אינו בטווח של $p(x)$), נגדיר $p^{-1}(y)=0$. הגדרה זו חיונית כדי לקבל פונקציה p^{-1} שמוגדרת לכל $y>0$, אך עדיין לא־עולה ובעלת אינטגרל סופי.*

אגב, בהגדרת p^{-1} ב־[21, 7] לא מופיע מקרה חשוב זה, שבלעדיו ההגדרה לא שלמה.

קל לראות שתי תכונות בסיסיות של ההגדרה הנ״ל:

1. p^{-1} מוגדרת היטב לכל $y>0$ מלבד לכל־היותר מספר בן־מניה של נקודות (ולכן בפרט מוגדרת למעט על קבוצה בעלת מידה אפס). הנקודות y היחידות בהן לא הגדרנו היטב את p^{-1} הן הנקודות

שמתאימות לקטע x שלם בעל ערך y קבוע. אולם, מספר הקטעים הללו הוא לכל היותר בן מניה (כי הרי בכל קטע קיים מספר רציונלי) ולכן גם מספר נקודות ה־y ה"בעייתיות" הוא לכל היותר בן־מניה.

2. $p^{-1}(y)$ היא בעצמה סידור לא־עולה, כלומר היא פונקציה אי־שלילית, לא־עולה, שמוגדרת ל־$y > 0$ ובעלת אינטגרל סופי בתחום $(0,\infty)$. האינטגרל סופי מכיוון שאפשר לראות ש־$\int_0^\infty p^{-1}(y)dy = \int_0^\infty p(x)dx$.

עכשיו, תהינה $p(x)$ ו־$q(y)$ ($x > 0$, $y > 0$) הסידורים הלא־עולים של $P(x)$, $Q(y)$ בהתאמה. נגדיר את התנאים הבאים:

$$\int_0^x p(u)du \leq \int_0^x q^{-1}(u)du \qquad \forall x > 0 \tag{2.3}$$

$$\int_0^y q(u)du \leq \int_0^y p^{-1}(u)du \qquad \forall y > 0 \tag{2.4}$$

ברור שתנאים אלו מתקיימים אם, למשל, $p(x)$ ו־$q(y)$ הן פונקציות הפוכות זו לזו (במקרה זה, אי השוויונים הם שוויונים). אך תנאים אלו יכולים להתקיים גם במקרים אחרים. דוגמה אפשרית אחת היא צמד הפונקציות $p(x) = e^{-x}$ ו־$q(y) = \left(-\frac{1}{2}\,log(\frac{y}{2})\right)^+$ (כאשר ב־α^+ הכוונה ל־$max(\alpha, 0)$) שמקיימות את התנאים, כפי שלא קשה להווכח[1]. אך הן אינן פונקציות הפוכות זו לזו: הפונקציה ההפוכה ל־$p(x) = e^{-x}$ היא $p^{-1}(y) = (-\,log\,y)^+$, שזו כאמור דרך מקוצרת לומר

$$p^{-1}(y) = \begin{cases} -\,log\,y & y \leq 1 \\ 0 & y > 1 \end{cases}$$

לורנץ [21] גילה שקיום תנאים אלו על־ידי הסידורים הלא־עולים של פונקציות הוא בדיוק התנאי לאפשרות שחזור קבוצה שפונקציות אלו הן ההטלות שלה בכיווני הצירים:

משפט 2 *תהינה $P(x)$ ו־$Q(y)$ שתי פונקציות אינטגביליות ואי־שליליות המוגדרות ל־$-\infty < x < \infty$, $-\infty < y < \infty$. תנאי הכרחי ומספיק לכך שקיימת קבוצה מדידה E עם הטלות $P(x)$, $Q(y)$ בכיווני הצירים הוא שהסידורים הלא־עולים $p(x)$, $q(y)$ של הפונקציות הנ"ל מקיימים את התנאים (2.3)-(2.4).*

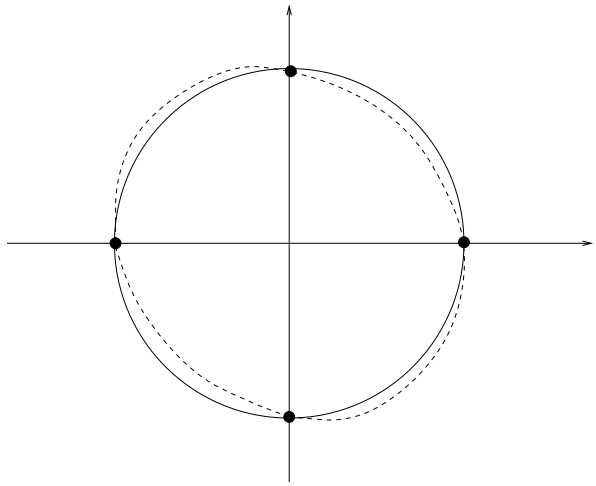
משפט 3 *יתר על כן, הקבוצה E נקבעת ביחידות (עד כדי קבוצות בעלות מידה אפס) על־ידי הטלותיה בכיווני הצירים אם ורק אם הסידורים הלא־עולים $p(x)$ ו־$q(y)$ הם פונקציות הפוכות זו לזו.*

הערה 1 *בצמד התנאים (2.3)-(2.4) ניתן להחליף את אחד התנאים בתנאי*

$$\int_0^\infty p(x)\mathrm{d}x = \int_0^\infty q(y)\mathrm{d}y$$

---

[1] קל לקראות ש־$q^{-1} = 2e^{-2x}$ ($x > 0$), $p^{-1} = (-\,log\,y)^+$ ($y > 0$). אז $\int_0^x p = -e^{-x} + 1$, $\int_0^x q^{-1} = -e^{-2x} + 1$ ומכיוון ש־e^{-x} פונקציה יורדת, מתקיים אי־השיוויון הראשון (2.3). גם אי־השיוויון השני מתקיים, אך הביטויים מסובכים מעט יותר: קל לראות ש־$\int_0^y q = \begin{cases} \frac{y}{2} - \frac{y}{2}\,log\,\frac{y}{2} & y < 2 \\ 1 & y \geq 2 \end{cases}$ וש־$\int_0^y p^{-1} = \begin{cases} y - y\,log\,y & y < 1 \\ 1 & y \geq 1 \end{cases}$ ובאמת מתקיים אי־השיוון השני (2.4).



איור 2.3: עיגול היחידה המעוות (מקווקו): העיגול (קו מלא) מוגדל ברביעים השני והרביעי, ומוקטן ברביעים הראשון והשלישי. למרות מה שנדמה במבט ראשון, לעיגול המעוות חייבות להיות הטלות שונות בכיווני הצירים מאלו של העיגול המקורי.

Figure 2.3: The deformed unit circle (dashed): The circle (solid line) is enlarged in the second and fourth quadrants, and made smaller in the first and third quadrants. Although it seems otherwise at first glance, the deformed circle must have projections in the direction of the axes that are different from those of the original circle

או במילים אחרות בתנאי

$$\int_{-\infty}^{\infty} P(x)\mathrm{d}x = \int_{-\infty}^{\infty} Q(y)\mathrm{d}y \tag{2.5}$$

ולקבל צמד תנאים שקול (ראה הוכחה ב־[21]). במובן מסויים צמד התנאים החדש ברור יותר, מכיוון שהתנאי (2.5) הוא התנאי ההכרחי הצפוי ביותר לכך ש־$P(x)$, $Q(y)$ יהיו זוג הטלות בכיווני הצירים של קבוצה מסויימת, כי במקרה זה שני אגפי (2.5) הם שני ביטויים שונים למידת הקבוצה (לפי משפט פוביני) ולכן כמובן שווים.

דוגמה פשוטה לשימוש במשפט 3 היא דוגמת העיגול:

דוגמה 1 *הורן בספרו [16] על ראית רובוטים שאל האם עיגול היחידה במישור נקבע ביחידות על־ידי הטלותיו על הצירים x, y. התשובה היא חיובית, למרות שבמבט ראשון נראה שאולי אפשר לבנות דוגמה נגדית — משהו כמו איור 2.3.*

אפשר להבין יחידות זו בקלות: לשתי קבוצות עם אותן הטלות בכיווני הצירים יש אותו שטח (לשם כך מספיקה אפילו הטלה אחת זהה — לפי משפט פוביני) ואותו מומנט אינרציה. אך העיגול הוא הקבוצה בעלת מומנט האינרציה המינימלי מכל הקבוצות עם שטח נתון, ולכן השטח ומומנט האינרציה קובעים את העיגול (עד כדי קבוצה עם מידה אפס).

נראה כעת איך משפט היחידות של לורנץ 3 מבטיח את יחידות השחזור של העיגול. ההטלות על הצירים של עיגול היחידה שמרכזו בראשית הם:

$$P(x) = 2\sqrt{(1-x^2)^+}, \qquad Q(y) = 2\sqrt{(1-y^2)^+}$$

כאשר שוב סימנו $\alpha^+ = max(\alpha, 0)$. בגלל המבנה הסימטרי סביב האפס של P (ו־Q) והעובדה ש־P לא־עולה בחצי הציר החיובי, ברור שהסידור הלא־עולה של $P(x)$ הוא פשוט $p(u) = P(u/2)$, $u > 0$. כלומר, הסידורים הלא־עולים של ההטלות הם:

$$p(u) = q(u) = \sqrt{(4 - u^2)^+}, \qquad u > 0$$

ואלו באמת פונקציות הפוכות, כלומר $p^{-1} = q$ (כאשר הפונקציה ההפוכה מוגדרת כמו בהגדרה 8). ובכן, תנאי משפט 3 מתקיימים, ולכן גם מסקנתו: עד כדי קבוצות עם מידה אפס, העיגול הוא הקבוצה היחידה עם הטלות אלו.

הערה: העיגול נקבע על־ידי שתי הטלותיו רק בהנחה ששני הכיוונים ניצבים. אם הכיוונים אינם ניצבים, נראה בהמשך איך העיגול לא נקבע באופן יחיד מהטלותיו — ויתר על כן הוא אפילו לא נקבע באופן יחיד מבין כל הקבוצות הקמורות (יש אליפסה אחרת שנותנת את אותן הטלות).

Kuba & Volčič במאמרם [18] מ־1988 ניסחו את משפטי לורנץ בצורה שקולה, אך במובנים מסויימים יותר אלגנטית (אם כי אולי קשה יותר להבנה אינטואיטיבית). בניסוח שלהם אין צורך להגדיר סידור לא־עולה או פונקציה הפוכה של סידור לא־עולה — אלו מופיעים כמובן בניסוח החדש, אך לא בשמם המפורש. במקום זה, מגדירים לפונקציה האי־שלילית (ובעלת האינטגרל הסופי) $g(x)$ המוגדרת על התחום Dg את הפונקציה $g_x(y)$, $y > 0$ הבאה:

$$g_x(y) = \lambda_1 \left(\{x \in Dg | g(x) \geq y\}\right) \tag{2.6}$$

ובדומה, לפונקציה $h(y)$ מגדירים:

$$h_y(x) = \lambda_1 \left(\{y \in Dh | h(y) \geq x\}\right) \tag{2.7}$$

אם $p(x)$ היא פונקציה המוגדרת ב־$(0, \infty)$ לא־עולה, אי־שלילית ובעלת אינטגרל סופי, אז אפשר לראות שהגדרה זו של $p_x(y)$ בדיוק מתלכדת עם הגדרה 8 של $p^{-1}(y)$! אך היופי בהגדרה של $(..)_x$ הוא שברור מהגדרתה שאם $P(x)$, $p(x)$ הן פונקציות שוות־פילוג (ראה הגדרה 7) אז $p_x = P_x$. לכן אין כלל צורך בהגדרת "סידור לא־עולה": $P_x(y)$, $y > 0$ הוא מיידית הפונקציה ההפוכה של הסידור הלא־עולה: $P_x(y) = p^{-1}(y)$. כדי לקבל את הסידור הלא־עולה עצמו, p, הדרוש גם הוא לניסוח משפט לורנץ, אפשר פשוט להפעיל את הפעולה שוב: $p = (P_x)_y$ (כזכור, מכיוון ש־p^{-1} הוא כבר סידור לא־עולה, הפעולה $(..)_y$ פשוט מוצאת את הפונקציה ההפוכה).

כצפוי, בניסוחם של משפטי לורנץ, בחרו Kuba & Volčič להחליף את התנאי (2.4) בתנאי (2.5) (כפי שהערנו שאפשר לעשות).

בסימונים אלו, שני משפטי לורנץ ניתנים לניסוח בצורת משפט אחד:

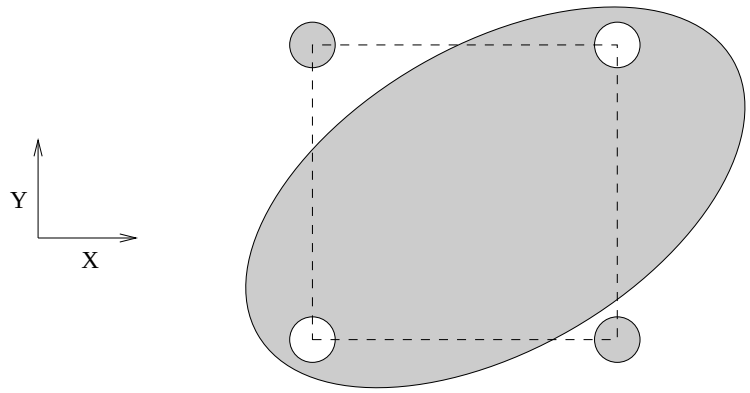
משפט 4 (ניסוח שונה למשפטי לורנץ) *תהינה $P(x)$ ו־$Q(y)$ פונקציות אי־שליליות ואינטגרביליות כך ש־*

$$\int_{-\infty}^{\infty} P(x)\mathrm{d}x = \int_{-\infty}^{\infty} Q(y)\mathrm{d}y$$

אזי:

1. *$P(x)$ ו־$Q(y)$ הן הטלות בכיווני הצירים של קבוצה יחידה אם ורק אם לכל $c > 0$ מתקיים*

$$\int_0^c Q_y(x)\mathrm{d}x = \int_0^c P_{xy}(x)\mathrm{d}x$$



איור 2.4: דוגמה של רכיב מיתוג
Figure 2.4: Example of a switching component

2. *$P(x)$ ו־$Q(y)$ הן הטלות בכיווני הצירים של יותר מקבוצה אחת אם ורק אם*

$$\int_0^c Q_y(x)\mathrm{d}x \geq \int_0^c P_{xy}(x)\mathrm{d}x$$

לכל $c > 0$ וקיים $c > 0$ שעבורו מתקיים אי־שוויון חריף.

3. *$P(x)$ ו־$Q(y)$ אינן הטלות בכיווני הצירים של אף קבוצה אם ורק אם קיים $c > 0$ כך ש־*

$$\int_0^c Q_y(x)\mathrm{d}x < \int_0^c P_{xy}(x)\mathrm{d}x$$

משפטי לורנץ מאפיינים את ההטלות של קבוצות במישור, אך לא מדברים ישירות על הקבוצות עצמן. Kuba & Volčič [18] הציעו דרך לבדוק האם קבוצה מדידה במישור נקבעת (ביחידות) מהטלותיה על הצירים, תוך הסתכלות על מבנה הקבוצה עצמה ולא על הטלותיה.

המבנה הראשון בקבוצה עליו כדאי לדבר בהקשר ליחידות שחזורה הוא רכיב מיתוג (switching component) — מושג ששימש לראשונה בהקשר של אפיון מטריצות בינריות שניתנות לשחזור יחיד [2]. לדוגמה, נבחן את השאלה הבאה: האם אליפסה מסובבת (כמו באיור 2.4) נקבעת על־ידי הטלותיה על הצירים? התשובה לכך שלילית: אפשר להגדיר קבוצה חדשה שמורכבת מאליפסה עם שני חורים עגולים, ובמקומם שני עיגולים חדשים חדשים (ראה קבוצה חדשה באפור באיור 2.4). ברור שההטלות בכיווני הצירים של הקבוצה החדשה זהות לאלו של האליפסה המקורית, ולכן אליפסה זו לא ניתנת לשחזור יחיד. ה"אשמה" בכך שאליפסה זו לא ניתנת לשחזור יחיד היא העובדה שאפשר להניח מלבן שצלעותיו מקבילות לצירים על האליפסה, כאשר שני קדקדים באלכסון בתוך האליפסה (סביבם בנינו את החורים) ושני הקדקדים האחרים מחוץ לאליפסה (סביבם בנינו את העיגולים החדשים). מבנה כזה יקרא רכיב מיתוג. ההגדרה הכללית היא:

הגדרה 9 *תהיה P קבוצה מדידה במישור בעלת מידה סופית. נסמן ב־$P_{(t,u)}$ את העתקת הקבוצה בוקטור (t,u):*

$$P_{(t,u)} = \{(x+t, y+u) | (x,y) \in P\}$$

נאמר שלקבוצה במישור E, מדידה ובעלת מידה סופית, יש <u>רכיב מיתוג</u> אם קיימת קבוצה P בעלת מידה לא־אפס ושני מספרים ממשיים $t \neq 0$, $u \neq 0$, כך ש־$P_{(t,0)} \cup P_{(0,u)} \subset E$, $(P \cup P_{(t,u)}) \cap E = \emptyset$ (כרגיל, עד כדי קבוצה עם מידה אפס).

ברור שאם ל־E יש רכיב מיתוג אז היא לא נקבעת באופן יחיד מהטלותיה בכיווני הצירים, שכן ל־

$$E' = (E \backslash (P_{(t,0)} \cup P_{(0,u)})) \cup P \cup P_{(t,u)}$$

יש אותן הטלות כמו ל־E בכיווני הצירים, ו־E, E' שונים (כלומר $\lambda_2(E \Delta E') \neq 0$). מתברר שקיום רכיב־מיתוג הוא לא רק תנאי מספיק לאי־יחידות השחזור, אלא הוא גם תנאי הכרחי. [18] הוכיחו את המשפט הבא בעזרת משפט לורנץ:

משפט 5 *קבוצה מדידה במישור בעלת מידה סופית ניתנת לשחזור יחיד מהטלותיה בכיווני הצירים אם ורק אם אין לה אף רכיב־מיתוג.*

יתר על כן, מכיוון שמקבוצה שמגדירה רכיב־מיתוג ניתן להוציא רצף של קבוצות שונות מהותית, מקבלים:

משפט 6 *אם הטלות בכיווני הצירים לא מגדירות קבוצה מישורית יחידה (עד כדי קבוצות בעלות מידה אפס) אזי יש עוצמת־רצף של קבוצות שונות מהותית הנותנות הטלות אלו.*

Fishburn, Lagarias, Reeds ו־Shepp הציעו במאמרם [7] מ־1990 הגדרה מפושטת של רכיב מיתוג לקבוצות פתוחות:

משפט 7 *תהיה U קבוצה <u>פתוחה</u> במישור בעלת מידה סופית, ונניח ש־∂U (שפת U) בעלת מידה אפס. אזי שני התנאים הבאים שקולים:*

1. *אין קבוצה פתוחה אחרת השונה מ־U שנותנת אותן הטלות בכיווני הצירים.*
2. *אין "קונפיגורציה 2־רעה", כלומר אין ארבע נקודות המרכיבות מלבן המקביל לכיווני הצירים, $z_1 = (x_1, y_1)$, $z_2 = (x_2, y_2)$, $w_1 = (x_1, y_2)$, $w_2 = (x_2, y_1)$, כאלו ש־ z_1 ו־z_2 בפנים U ואילו w_1 ו־w_2 בפנים $\bar{U}$.*

[7] משערים שהמשפט נכון גם ללא התנאי ש־$\lambda_2(\partial U) = 0$, ונותנים הגדרה לקונפיגורציה k־רעה שלגביה הם יודעים להוכיח את השקילות ללא התנאי על שפת U: קונפיגורציה k־רעה היא שתי קבוצות של k נקודות שונות, $z_1..z_k$, $w_1..w_k$, z_i בפנים U ו־w_i בפנים $\bar{U}$, כך שהטלות שתי הקבוצות על הצירים זהות — כלומר, כל ישר המקביל לציר, $x = c$ (או $y = c$) חותך מספר שווה של z_i־ים ו־w_i־ים.

אם קבוצה נקבעת ביחידות על־ידי הטלותיה, אפשר לתת נוסחה מפורשת לקבוצה, כפי שהציעו [18] (ההוכחה שוב נובעת ממשפט לורנץ):

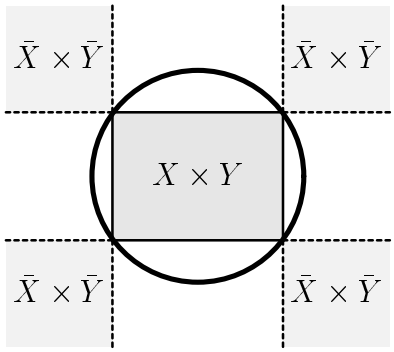
משפט 8 *אם E נקבעת ביחידות על־ידי הטלותיה $P(x)$ ו־$Q(y)$ אז*

$$E = \{(x,y) | P(x) \geq Q_y(Q(y))\} \tag{2.8}$$

עד כדי קבוצה בעלת מידה אפס.

בסימונים של לורנץ,

$$E = \left\{(x,y) | P(x) \geq q^{-1}(Q(y))\right\}$$



איור 2.5 : מלבן חסום בעיגול
Figure 2.5: Rectangle inscribed in a circle

בעזרת משפט 8 ניתן למצוא אפיון גאומטרי נוסף של הקבוצות במישור שנקבעות ביחידות מהטלותיהן על הצירים. אפשר לשכתב את ההצגה של E לפי (2.8) בצורה הבאה :

$$E = \bigcup_{a>0} \{x|P(x) \geq a\} \times \{y|a \geq Q_y(Q(y))\} = \bigcup_{a>0} X_a \times Y_a$$

כאשר

$$X_a = \{x|P(x) \geq a\}\,, \qquad Y_a = \{y|a \geq Q_y(Q(y))\}$$

אפשר להוכיח בקלות גם ש־$\bar{X}_a \times \bar{Y}_a \subset \bar{E}$ (כאשר $\bar{A}$ מסמן את המשלים של הקבוצה A, וההכלה היא כמובן במשמעות של מידה : $A \subset B$ אם $\lambda(A \backslash B) = 0$). בהקשר זה, [18] הגדירו את ההגדרות הבאות :

הגדרה 10 *תהיה E קבוצה מדידה במישור בעלת מידה סופית. תהינה $X, Y \subset R$ קבוצות מדידות בישר. $X \times Y$ תקרא <u>קבוצה מדידה פשוטה</u>. מלבן הוא מקרה פרטי של קבוצה פשוטה, אך ישנן כמובן קבוצות פשוטות שאינן מלבן. נאמר ש־ $X \times Y$ היא <u>חסומה</u> (measurably inscribed) ב־ E אם*

$$X \times Y \subset E \qquad \bar{X} \times \bar{Y} \subset \bar{E}$$

נאמר שהקבוצה E היא <u>בת־חסימה</u> (measurably inscribable) אם היא איחוד של קבוצות פשוטות החסומות בה.

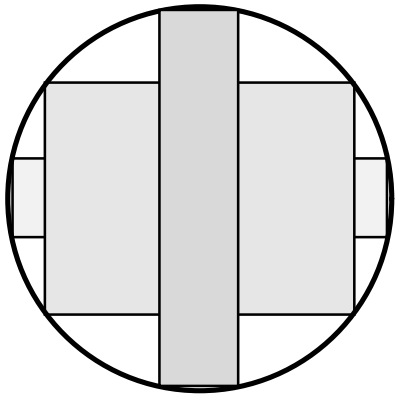
ראה דוגמה למלבן (שהוא מקרה פרטי של קבוצה פשוטה) החסום בעיגול, באיור 2.5.

כפי שראינו, קבוצות במישור הנקבעות ביחידות מהטלותיהן בכיווני הצירים הן גם בנות־חסימה. [18] הוכיחו גם את הכיוון ההפוך, וניסחו משפט :

משפט 9 *קבוצה מדידה במישור בעלת מידה סופית נקבעת ביחידות על־ידי שתי הטלותיה בכיווני הצירים אם ורק אם היא בת־חסימה.*

דוגמה 2 *בעזרת מושג החסימה אפשר לראות הוכחה נוספת לכך שהעיגול נקבע על־ידי הטלותיו בכיווני הצירים מבין כל הקבוצות המדידות. קל לראות שהעיגול ניתן להצגה כאיחוד של מלבנים החסומים בו (ראה איור 2.6). נוסחה מפורשת למלבנים שחסומים בעיגול היחידה C:*

$$C = \bigcup_{x=0}^{1} \left([-x, x] \times [-\sqrt{1-x^2}, \sqrt{1-x^2}]\right)$$



איור 2.6: העיגול הוא איחוד מלבנים החסומים בו: דוגמה של שלושה מהמלבנים
Figure 2.6: The circle is the union of circles inscribed within it: example of three of the rectangles

בדומה, [18] גם מוכיח את המשפט הבא:

משפט 10 *קבוצה מדידה במישור בעלת מידה סופית נקבעת ביחידות על־ידי שתי הטלותיה בכיווני הצירים אם ורק אם*

$$F = \bigcup_{a \in Q} X_a \times Y_a$$

כאשר X_a *ו־* Y_a *הן קבוצות מדידות על הישר כך ש־*

$$\begin{aligned} X_\alpha \subset X_\beta \quad &\Longleftrightarrow \quad \alpha \leq \beta \\ Y_\alpha \supset Y_\beta \quad &\Longleftrightarrow \quad \alpha \leq \beta \end{aligned}$$

כאשר $\alpha, \beta \in Q$ *(*Q *הוא קבוצת המספרים הרציונליים).*

Fishburn, Lagarias, Reeds ו־Shepp הגדירו ב־[7] קריטריון נוסף לבדיקה האם קבוצה ניתנת לשחזור יחיד מהטלותיה בכיווני הצירים:

הגדרה 11 *קבוצה* E *במישור נקראת* <u>*אדיטיבית*</u> *אם קיימות פונקציות מדידות חסומות* $f(x)$ *ו־* $g(y)$ *כך ש־*

$$(x, y) \in E \iff f(x) + g(y) \geq 0$$

לדוגמה, עיגול היחידה הוא אדיטיבי, עם $f(x) = 1/2 - x^2$, $g(y) = 1/2 - y^2$. המשפט הבא נכון במישור (הוא לא נכון, אגב, למימדים גבוהים יותר שם אדיטיביות הוא תנאי מספיק אך לא הכרחי ליחידות, אך אנו נעסוק בעבודה זו רק במישור):

משפט 11 *תהיה* E *קבוצה מדידה במישור בעלת מידה סופית. אזי* E *אדיטיבית אם ורק אם היא נקבעת ביחידות על־ידי שתי הטלותיה בכיווני הצירים.*

2.2.2 הטלות בשלושה כיוונים או יותר

כאשר מדובר בהטלות משלושה כיוונים או יותר, כבר אין קבוצת כיוונים "טבעית", ונסמן ב־

$$S = \{u_1, \ldots, u_n\}$$

את קבוצת הכיוונים במישור בהן נבצע את ההטלות.

ראינו שזוג כיווני הצירים מאפשר שחזור יחיד של רק חלק מהקבוצות המדידות. מתברר שאין אף קבוצת כיוונים סופית S שמאפשרת שחזור יחיד של כל הקבוצות המדידות:

משפט 12 *לכל קבוצת כיוונים סופית S קיימת קבוצה מישורית מדידה (ואפילו קומפקטית) שלא נקבעת ביחידות על־ידי הטלותיה בכיוונים S (מבין כל הקבוצות המדידות).*

ההוכחה (של לורנץ [21]) על־ידי דוגמה: מגדירים קבוצת נקודות סופית שהיא רכיב־מיתוג בצורה הבאה (הדומה להגדרה שהיתה לנו לשני כיוונים): איחוד $F \cup G$ של קבוצות סופיות זרות נקרא S־*רכיב־מיתוג* (*S-switching component*) אם הטלות G,F בכיווני S זהות, כלומר כל ישר באחד מכיווני S חותך מספר שווה של נקודות ב־F וב־G. מראים באינדוקציה על מספר הכיוונים ב־S שלכל קבוצת כיוונים סופית S אפשר לבנות S־רכיב־מיתוג, ואז להקיף כל נקודה בעיגול מספיק קטן לקבלת הדוגמה הדרושה.

יתר על כן, לצערנו מספר סופי של כיוונים לא מספיק כדי להבחין גם מבין גופים ממשפחה קטנה יותר: גופים כוכביים:

משפט 13 *לכל קבוצת כיוונים סופית S קיים מצולע כוכבי בראשית שלא נקבע ביחידות על־ידי הטלותיו בכיוונים S מבין כל המצולעים הכוכביים בראשית.*

שוב ההוכחה על־ידי דוגמה, כלומר בהנתן קבוצת כיוונים S מוצאים שני מצולעים כוכביים־בראשית עם אותן הטלות בכיווני S. שוב בונים את הדוגמה סביב S־רכיב־מיתוג סופי. הבניה המלאה של גרדנר מופיעה ב־[10], וכן ב־[11, עמוד 66].

כפי שנראה בהמשך (משפט 21 בפרק 2.3.1), המצב במשפחת הקבוצות הקמורות הוא מסובך יותר: ישנן קבוצות כיוונים סופיות שכן מאפשרות שחזור יחיד של כל קבוצה קמורה מבין משפחת הקבוצות הקמורות.

נראה עכשיו הכללה של מושגי החסימה, אדיטיביות, וכו. שראינו בפרק הקודם (בהקשר של זוג כיווני הצירים) לקבוצה סופית כלשהי של כיוונים S. בעקבות גרדנר [11], נגדיר את ההגדרות לגוף קמור, דבר שמפשט את ההגדרות. אך יש לזכור שבכל מקום בסעיף זה נתענין בשאלה האם גוף זה ניתן לשחזור יחיד מבין כל הקבוצות המדידות, ולא הקמורות. אפשר להגדיר גם הגדרות כלליות יותר לקבוצה מדידה כלשהי (ראה למשל [7]).

הגדרה 12 *תהיה S קבוצת כיוונים סופית. נאמר שגוף קמור K הוא S־בר־חסימה אם פנים K הוא איחוד של מצולעים קמורים החסומים ב־K (כלומר, שכל קדקדיהם על שפת K), ושכל אחת מצלעותיהם מקבילה לאחד מכיווני S.*

הלמה הבאה שימושית לטיפול בגופים בני־חסימה, אך נראה בה שימושים נוספים בהמשך:

למה 1 *תהיה S קבוצת כיוונים סופית במישור. תהיה K קבוצה קמורה במישור. יהיה Q מצולע קמור החסום ב־K (כך שקדקדי Q על ∂K), שכל אחת מצלעותיו מקבילה לאחד מכיווני S. אזי Q מוכל, עד כדי קבוצה בעלת מידה אפס, בכל קבוצה מדידה E עם אותן הטלות כמו K בכיוונים S.*

הוכחת למה זו פשוטה ויפה: נסמן את צלעות Q ב־e_i, ונסמן ב־E_i את חצי המישור הפתוח המוגדר על־ידי הישר שמכיל את הצלע e_i ושאינו מכיל את Q. לכל קבוצה A, נגדיר $A_i = A \cap E_i$. מכיוון שקדקדי Q על שפת ב־K ושניהם קמורים, מתקיים

$$\lambda_2(K) = \lambda_2(Q) + \sum_i \lambda_2(K_i)$$

עכשיו, תהי E קבוצה מדידה עם אותן הטלות כמו K בכיוונים S. ממשפט פוביני ושוויון ההטלות בכיווני הצלעות e_i, נובע ש־$\lambda_2(K) = \lambda_2(E)$ ו־$\lambda_2(K_i) = \lambda_2(E_i)$, לכל i. עכשיו, $E\setminus(\cup_i E_i) \subset Q$ אבל

$$\begin{aligned}\lambda_2(E\setminus(\cup_i E_i)) &\geq \lambda_2(E) - \sum_i \lambda_2(E_i)\\ &= \lambda_2(K) - \sum_i \lambda_2(K_i)\\ &= \lambda_2(Q)\end{aligned}$$

ולכן $E\setminus(\cup_i E_i) = Q$, שגורר כמובן $Q \subset E$.

■

מלמה זו נובע בפרט המשפט הבא:

משפט 14 *תהיה S קבוצת כיוונים במישור סופית. אז גוף קמור שהוא S־בר־חסימה נקבע מבין כל הקבוצות המדידות על־ידי הטלותיו בכיווני S.*

כדי לפשט את ההגדרה עבור רכיבי־מיתוג, גרדנר בוחר להשתמש בקבוצות סופיות (כפי שראינו לעיל) וקבוצות בעלות פנים לא־ריק. שוב, אפשר היה לכתוב הגדרה כללית יותר בדומה למה שעשינו במקרה של כיווני הצירים בסעיף הקודם.

משפט 15 *תהיה S קבוצת כיוונים במישור סופית. תהיה E קבוצה מדידה כך שיש S־רכיב־מיתוג $F \cup G$ (קבוצות סופיות כפי שהגדרנו קודם) עם $F \subset \mathrm{int}E$ ו־$G \subset \mathrm{int}(R^2\setminus E)$. אז קיימת קבוצה מדידה E' השונה מהותית מ־E (כלומר, לא זהה עד־כדי מידה אפס) עם אותן הטלות כמו E בכיווני S. אם E קומפקטית, אפשר למצוא גם E' קומפקטית.*

לדוגמה, ראינו בסעיף הקודם שעיגול ניתן לשחזור יחיד מהטלותיו בזוג כיוונים ניצבים, מבין כל הקבוצות המדידות. אך אין יחידות כשזוג הכיוונים לא ניצב. באיור 2.7 רואים דוגמה לקבוצה מדידה נוספת (באפור) עם אותן הטלות כמו העיגול בזוג כיוונים לא ניצבים. הדוגמה בנויה כמובן על רכיב־מיתוג. אגב, כשזוג הכיוונים לא ניצב, העיגול לא ניתן לשחזור יחיד אפילו מבין כל הקבוצות <u>הקמורות</u>: טרנספורמצית השיקוף ϕ מהוכחת משפט 28 בהמשך נותנת אליפסה שונה עם אותן הטלות.

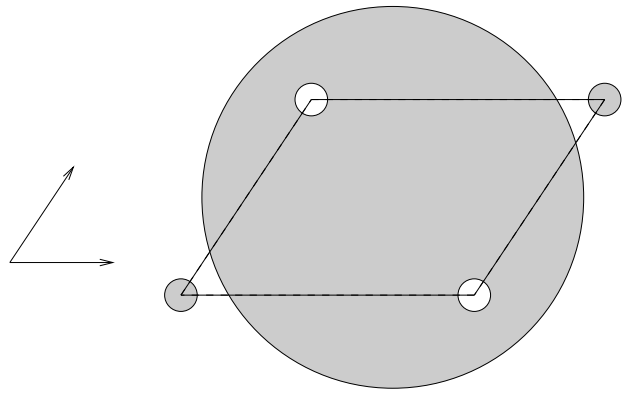
גרדנר [11] מכליל את הגדרת האדיטיביות לקבוצת כיוונים כללית:

הגדרה 13 *פונקציה f נקראת פונקצית רכס (ridge function) בניצב לכיוון u אם הפונקציה קבועה לאורך כל ישר בכיוון u.*

תהיה S קבוצה סופית של כיוונים במישור. קבוצה מדידה E נקראת S־אדיטיבית אם:

$$E = \left\{x \in R^2 : \sum_i f_i(x) > 0\right\}$$

כאשר f_i הן פונקציות רכס מדידות וחסומות על קבוצות קומפקטיות הניצבות לכיוונים u_i ב־S. את ה־$>$ בהגדרה זו אפשר להחליף ב־$\geq$, וגם אז נקרא לקבוצה אדיטיבית.



איור 2.7: עיגול לא נקבע מהטלותיו בזוג כיוונים לא ניצבים

Figure 2.7: A circle is not determined by its projections in a pair of non-orthogonal directions

בדיוק כמו במקרה של כיווני הצירים, מראים גם פה:

משפט 16 *קבוצה S־אדיטיבית נקבעת בין הקבוצות המדידות על־ידי הטלותיה בכיווני S.*

גרדנר הראה ב־[9] משפט מעניין שמוכח בעזרת אדיטיביות:

משפט 17 *יהיה E גוף במישור הסימטרי יחסית לציר y (כלומר, $(x,y) \in E \iff (-x,y) \in E$), ונניח שכל חיתוך שלו עם ישר הניצב לציר y נותן קטע (אולי מנוון). אזי E הוא S־אדיטיבי (ולכן גם ניתן לשחזור יחיד מבין הקבוצות המדידות מהטלותיו בכיוון S) כאשר S הוא קבוצה המכילה את כיוון ה־x ושני כיוונים כלשהם נוספים.*

הוכחה: מההנחות שלנו יש פונציה מדידה g כך ש־$E = \{(x,y) \in R^2 : x^2 \le g(y)^2\}$. אגב, זה מיד מראה E אדיטיבית יחסית לזוג כיווני הצירים (ולכן מוכיח את משפט 32 מפרק 4.3.1!).

יהיו $y + a_i x = 0$, $a_i \neq 0$, $i = 1,2$, $a_1 \neq a_2$, שני ישרים שונים כלשהם דרך הראשית (כמובן, הם לא ציר ה־x או ה־y). עכשיו,

$$0 \le g(y)^2 - x^2 \quad = \quad \left(g(y)^2 - \frac{1}{a_1 a_2} y^2\right) + \frac{1}{a_1(a_2 - a_1)}(y + a_1 x)^2 + \frac{1}{a_2(a_1 - a_2)}(y + a_2 x)^2$$

אגף ימין הוא סכום פונקציות רכס בצורה הרצויה, ניצבות לכיוון ציר ה־x, $y + a_1 x = 0$ ו־$y + a_2 x = 0$, כמו שרצינו.

■

כפי שהזכרנו בהוכחה, כאשר לוקחים כשני כיוונים את הכיוונים הניצבים x ו־y, הם מספיקים לשחזור יחיד של גוף כמו זה במשפט ואין צורך בכיוון שלישי.

ההנחה במשפט הקודם, שחיתוך הגוף בניצב לציר y הוא קטע, היא הכרחית. גרדנר [11, עמוד 73] מראה דוגמה של טבעת (עיגול עם חור עגול) שלא נקבעת על־ידי כיוון x ועוד שני כיוונים מסוימים.

בגלל האינווריאנטיות לטרנספורמציה לינארית הפיכה של תכונת האדיטיביות, והפעלת המשפט האחרון על עיגול, מתקבלת המסקנה הבאה:

מסקנה 1 *תהיה S קבוצה של שלושה כיוונים שונים כלשהם במישור. אליפסה היא S־אדיטיבית ולכן נקבעת מבין הקבוצות המדידיות על־ידי הטלותיה בכיוונים של S.*

אגב, יש לשים לב שלקבוצת כיוונים סופית S (בניגוד למקרה של שני כיווני הצירים), תנאי האדיטיביות ותנאי החסימה אינם תנאים הכרחיים לשחזור יחיד. לדוגמה, עיגול ניתן לשחזור מכל שלושה כיוונים (כפי שראינו קודם, מכיוון שהוא אדיטיבי) אך אפשר לראות שאינו S־בר־חסימה עם שלישית הכיוונים בזוויות 0, $\pi/6$, $5\pi/6$ מציר ה־x החיובי. גרדנר [10] מראה שכל גוף S־בר־חסימה הוא גם S־אדיטיבי, אך הכיוון ההפוך אינו נכון.

נעבור כעת לדבר על *אימות* קבוצות מדידות: נתונה קבוצה מדידה E ידועה מראש. נתונה קבוצה מדידה נוספת, E', שרוצים לבדוק בעזרת הטלות האם היא זהה ל־E. במקרה זה מותר לבחור כיוונים התלויים ב־E הידוע, והשאלה אם אפשר לבחור כיוונים כאלו שיבטיחו שההטלות של E ו־E' בכיוונים אלו זהות אם ורק אם E ו־E' זהים (עד כדי קבוצה עם מידה אפס). ראו הגדרה 4 בפרק 2.1.

גרדנר [10] מוכיח את המשפט הבא:

משפט 18 *קיימת קבוצה מדידה שאינה ניתנת לאימות בעזרת אף קבוצת כיוונים סופית.*

הדוגמה שגרדנר בונה כדי להוכיח את המשפט היא קבוצה קומפקטית, שהיא איחוד בן־מניה של מצולעים פשוטים. הרעיון מאחורי ההוכחה הוא שכאשר לוקחים S־רכיב מיתוג, שהוא קבוצה סופית של נקודות, ובונים סביבה עיגולים כדי ליצור קבוצה שלא ניתנת לשחזור יחיד מכיווני S, אז אם ניקח כיוונים *מספיק קרובים* ל־S, הקבוצה שלנו תכיל רכיב־מיתוג גם עבור כיוונים אלו.

גרדנר מוכיח באותו מאמר משפט נוסף:

משפט 19 *כל מצולע קמור ניתן לאימות מבין הקבוצות המדידות על־ידי שלושה כיוונים.*

אגב, בהמשך נראה משפט 27 שאומר שכל גוף קמור ניתן לאימות מבין הקבוצות הקמורות על־ידי שלושה כיוונים. המשפט הנוכחי חזק יותר בכך שמדבר על שחזור מבין כל הקבוצות המדידיות, ומצד שני הוא חלש יותר בכך שהוכחתו עובדת רק עבור מצולעים.

2.3 שחזור קבוצות קמורות

כפי שראינו בפרק הקודם, ידוע הרבה מאוד על האפשרות לשחזר קבוצה מבין כל הקבוצות המדידות, ולהטלות בשני כיוונים ידועים תנאים הכרחיים ומספיקים ליחידות השחזור. אולם, לעיתים קרובות ברור שהצורה שאותה מנסים לשחזר היא קשירה ופשוטת קשר, ומעוניינים לדעת האם קיימת קבוצה יחידה כזו המתאימה להטלות הנתונות, למרות שיתכן שקיימות קבוצות אחרות (אינסוף כאלו, למעשה) שאינן קשירות ומתאימות להטלות הנתונות.

מתברר שקל יותר לענות על שאלות אלו כאשר משפחת הקבוצות עליה מתבוננים היא משפחת הקבוצות הקמורות. היתרון הברור הראשון של קבוצות קמורות הוא שקרן חותכת קבוצה קמורה בקטע, ולא בקבוצה מדידה מסובכת. אולם חלק מהתוצאות על שחזור קבוצות קמורות נובעות בצורה עקיפה יותר מהקמירות.

השאלות מתי ניתן לשחזר או לקבוע צורות מישוריות קמורות מתוך הטלותיהן הועלו לראשונה על־ידי P. C. Hammer [15] ב־1961, בכנס של החברה האמריקנית למתמטיקה (AMS) בנושא קמירות. מאז התבצע מחקר רב בנושא, ופורסמו מאמרים רבים. פרק 1 של [11] מכיל סקירה מקיפה של הידוע בתחום.

מתברר שבשחזור צורות מבין משפחת הצורות הקמורות יש גם צורך שימושי: [3, 4] עוסקים בקרוב חתכים של הלב על־ידי צורות קמורות בעלות סימטריה מרכזית. גם בהקשר של רובוטיקה עלה הצורך בטיפול בהטלותיהם של גופים קמורים (מישוריים ומרחביים).

בפרק זה נביא סקירה של המשפטים הידועים על שחזור קבוצות קמורות. כפי שנראה, ישנם עדיין מספר בעיות שנשארו פתוחות מאז שהמחקר בנושא החל, ולמשל לא ידועים תנאים הכרחיים ומספיקים לשחזור קבוצה קמורה על־ידי הטלות משני כיוונים. הוכחות לרוב המשפטים בסעיף זה מופיעות ב־[11].

2.3.1 אפיון קבוצות כיוונים S שמבטיחות שחזור יחיד של כל קבוצה קמורה

בסעיף זה נעסוק באפיון קבוצות כיוונים S שמאפשרות בנית "מכונת שחזור", כלומר נחפש קבוצות כיוונים S כאלו שמבטיחות שחזור יחיד של <u>כל</u> קבוצה קמורה מהטלותיה בכיווני S.

מיד אפשר לשים לב שאין מספר סופי n כזה שכל n הטלות יבטיחו שחזור יחיד של כל קבוצה קמורה:

משפט 20 *לכל $n \in N$ קיימת קבוצה של n כיוונים שונים כך שקיימים שני מצולעים קמורים שונים עם אותן הטלות בכיוונים אלו.*

ההוכחה היא על־ידי בנית דוגמה פשוטה: נסתכל על מצולע משוכלל בעל n צלעות, שנסמנו ב־Q, ועל סיבובו ב־π/n שנסמנו ב־Q'. הקמור של Q ו־Q' הוא מצולע בעל $2n$ צלעות. לכל אחד מ־n הכיוונים של צלעות אלו, קל לראות שההטלות בכיוון זה של Q ו־Q' הן זהות.

דוגמה זו חשובה יותר מסתם הוכחת המשפט הקודם: מסקנה נוספת ממנה הוא שקבוצה S שמורכבת מתת־קבוצה של כיווני הצלעות של מצולע משוכלל <u>אינה</u> יכולה להבטיח שחזור יחיד של <u>כל</u> קבוצה קמורה.

יתר על כן, בעית השחזור מהטלות מקבילות היא בעיה אפינית בטבעה: אם ϕ היא טרנספורמציה אפינית הפיכה, ונניח ש־K ו־K' הם גופים קמורים עם אותן הטלות בכיוון u, אז מכיוון ש־ϕ מעביר קווים מקבילים לקווים מקבילים, ושומר על יחס בין אורכים של קטעים מקבילים, נובע שהתמונות ϕK ו־$\phi K'$ הן קמורות ובעלות אותן הטלות בכיוון ϕu. מכאן, על־ידי הפעלת טרנספורמציה אפינית על הדוגמה הקודמת, מסיקים את המסקנה הבאה:

מסקנה 2 *אם קבוצת כיוונים S היא תת־קבוצה של קבוצת כיווני הצלעות של מצולע משוכלל־אפינית, אז קיימות שתי קבוצות קמורות שונות שנותנות את אותן הטלות בכל כיווני S (מצולע משוכלל־אפינית הוא תמונה תחת טרנספורמציה אפינית כלשהי של מצולע משוכלל).*

המשפט ההפוך נכון גם הוא, כלומר: התנאי הנ"ל הוא לא רק תנאי מספיק לאי־יחידות שחזור, אלא גם תנאי הכרחי:

משפט 21 *תהיה S קבוצה סופית של כיוונים ש<u>אינה</u> תת־קבוצה של קבוצת הצלעות של אף מצולע משוכלל־אפינית. אזי ניתן לשחזר ביחידות כל קבוצה קמורה מהטלותיה בכיוונים S.*

משפט זה הוכח לראשונה ב־[12], אך הוכחה ברורה יותר ניתן למצוא בספרו המאוחר יותר של גרדנר [11, עמודים 32–37]. רעיון ההוכחה הוא כזה: מגדירים שמצולע קמור לא־מנוון Q הוא <u>S־רגולרי</u> אם יש לו את התכונה הבאה: אם v קדקד של Q, ו־$u \in S$, אז הישר דרך v בכיוון u פוגש קדקד נוסף v' של Q. אז מוכיחים שתי טענות שביחד נותנות את המשפט המבוקש:

1. *אם אין יחידות, כלומר יש שני גופים קמורים שונים K, K' עם אותן הטלות בכיווני S, אז קיים מצולע S־רגולרי.*

הוכחת טענה זו היא קונסטרוקטיבית: לוקחים רכיב קשירות של $K\Delta K'$, מסתכלים על הרכיב "מולו" בגוף השני לפי אחד הכיוונים, וכן הלאה מרכיבים אלו לשאר הכיוונים, עד לקבלת (אפשר להוכיח) מספר סופי של רכיבים כאלו. מראים שמרכזי הכובד של רכיבים אלו יוצרים מצולע S-רגולרי.

2. *אם קיים מצולע S-רגולרי, אז S הוא תת־קבוצה של של כיווני צלעות של מצולע משוכלל־אפינית.*

הוכחת הטענה מתבססת על למה יפה וישנה של דרבו [5], שמתארת בנית סדרה מתכנסת של מצולעים S-רגולרים, שמתחילה במצולע הנתון, ומסיימת במצולע שהוא משוכלל־אפינית, R. אך מצולע גבול זה גם הוא S-רגולרי כמו כל המצולעים בסדרה, וזה אומר שכיווני S הם תת־קבוצה של כיווני הצלעות והאלכסונים של R. אם R מצולע משוכלל בעל n צלעות, נשים לב שידוע שכיווני הצלעות של מצולע משוכלל בעל $2n$ צלעות הם בדיוק כיווני הצלעות והאלכסונים של מצולע משוכלל בעל n צלעות, ולכן נובע ש־S הוא תת־קבוצה של כיווני הצלעות של מצולע משוכלל־אפינית (בעל $2n$ צלעות).

ובכן מצאנו תנאי הכרחי ומספיק על קבוצת הכיוונים S כדי שאפשר יהיה לשחזר ביחידות כל גוף קמור מהטלותיו בכיוונים S. בפרט נובע ש:

משפט 22 *קבוצות מסוימות של ארבעה כיוונים מאפשרות שחזור יחיד של כל גוף קמור, אך אף קבוצה של שלושה כיוונים (או פחות) אינה מאפשרת זאת.*

ברור שקבוצה של שלושה כיוונים לא מספיקה, מכיוון שעל־ידי טרנספורמציה אפינית הפיכה מתאימה ניתן להעביר כל שלישית כיוונים שונים לכיווני הצלעות של משולש שווה צלעות, וכפי שראינו קודם, בכיוונים אלו ישנם שני משולשים שווי צלעות שנותנים את אותן הטלות.

כדי להראות שקבוצות מסויימות של ארבעה כיוונים מספיקות לשחזור יחיד של כל גוף קמור, מראים שיש קבוצות של ארבעה כיוונים ש<u>לא יכולות</u> להיות תמונה אפינית של כיווני צלעות מצולע משוכלל. הרעיון הוא להגדיר יחס־כפול (cross-ratio) של ארבעה שיפועים:

$$\langle s_1, \ldots, s_4 \rangle = \frac{(s_3 - s_1)(s_4 - s_2)}{(s_4 - s_1)(s_3 - s_2)}$$

גרדנר [11] מוכיח שהיחס הכפול של כיווני צלעות מצולע משוכלל הוא מספר אלגברי (כלומר שורש של פולינום עם מקדמים שלמים), וכן שהיחס הכפול נשמר על־ידי טרנספורמציה אפינית. לכן כל קבוצה של ארבעה כיוונים עם יחס־כפול טרנסצנדנטי (לא־אלגברי) אינה יכולה להיות תת־קבוצה של כיווני הצלעות של מצולע משוכלל־אפינית.

הוכחת המשפט הנ"ל אינה קונסטרוקטיבית, במובן שהוכחנו שארבעה כיוונים (עם יחס־כפול טרנסצנדנטי) מספיקים לשחזור כל גוף קמור מהטלותיו, אך לא הראנו כיצד ניתן למצוא את הגוף בהנתן הטלותיו. בפרק 3 נראה אלגוריתם שהציע גרדנר לשחזור מעשי במקרים מסויימים וכשנתונות הטלות בארבעה כיוונים, ואת מימושו. בפרק 4 נראה אלגוריתם שפיתחנו אנו שעובד, בפרט (אך לא רק), כשנתונות הטלות בארבעה כיוונים.

אגב, בקבוצת כיוונים שלא נותנת יחידות, יש הרבה דוגמאות של אי־יחידות השחזור, ולא רק המצולעים החופפים (מסובבים) שראינו קודם. גרדנר [11] מביא את התוצאה הבאה (במקור של Volčič):

משפט 23 *תהיה S תת־קבוצה של כיווני הצלעות של מצולע משוכלל־אפינית. מספר המצולעים הקמורים הלא־חופפים, בעלי n צלעות, שיש להם אותן הטלות בכיוונים S עולה לפחות אקספוננציאלית עם n.*

יש בעיה עקרונית במשפט 22 ודומיו, שגם גרדנר מכיר בה: קבוצת רביעיות הכיוונים ה"גרועים", שלא נותנים יחידות, היא צפופה (כפי שנוכיח במשפט הבא) ולכן באופן מעשי גם אם לוקחים רביעית כיוונים "טובה", פרטורבציה קטנה עלולה להפוך אותה ל"גרועה", ולא תהיה יחידות. עוד על שאלת היציבות של בעית השחזור, בהמשך פרק זה.

כדי להראות את צפיפות הרביעיות שלא נותנות יחידות (וכך גם לכל מספר אחר של כיוונים), אנו מציעים את המשפט הבא:

משפט 24 *נאמר, לצורך משפט זה, שקבוצה של n כיוונים היא "גרועה" אם קיימים שני גופים קמורים שונים עם אותן הטלות בכיוונים הנתונים.*

אזי קבוצת ה־n־יות ה"גרועות" של כיוונים היא צפופה במרחב n־יות הכיוונים. במילים אחרות, אם $(u_i)_{i=1}^n$ קבוצה של n כיוונים, אז יש סדרות u_i^k, $lim_{k\to\infty} u_i^k = u_i$ כך ש־$(u_i^k)_{i=1}^n$ קבוצת כיוונים "גרועה" לכל k.

הוכחה: לכל כיוון נגדיר את זוויתו, שהיא הזווית θ בין ישר בכיוון זה לבין ציר ה־x ($0 \le \theta < \pi$). ברור שהכיוונים בעלי זווית שהיא כפולה רציונלית של π הם צפופים בקבוצת כל הכיוונים במישור. לכן אם נראה שכל n־ית כיוונים עם זוויות שהן כפולות רציונליות של π היא "גרועה", נובעת המסקנה שרצינו להוכיח.

ואכן, אם נתונים n כיוונים עם זוויות θ_i ($i = 1 \dots n$) שהן כפולות רציונליות של π, נמצא מכנה משותף k כך ש: $\theta_i = \frac{m_i}{k}\pi$ (m_i טבעיים). אבל, אם נסתכל על מצולע משוכלל עם $2k$ צלעות שאחת מצלעותיו בכיוון ציר x, אז זוויות צלעותיו הן בדיוק $\frac{m}{k}\pi$ לכל $0 \le m < k$ טבעי (כי הרי זווית חיצונית במצולע עם $2k$ צלעות היא $\frac{2\pi}{2k}$). לכן הכיוונים θ_i הם תת־קבוצה של כיווני הצלעות של המצולע הנ"ל. לכן, ממסקנה 2, קבוצת הכיוונים הנ"ל היא "גרועה".

■

2.3.2 שחזור קבוצות קמורות מהטלותיהן בשני כיוונים

בסעיף הקודם ראינו שאף זוג כיוונים לא מספיק לשחזור יחיד של כל גוף קמור (מבין הגופים הקמורים), ולכל זוג כיוונים יש דוגמאות רבות לגופים קמורים שלא ניתנים שלחזור יחיד. למעשה Volčič [27] מראה שעוצמת קבוצת הדוגמאות הנגדיות (לקבוצת כיוונים נתונה) היא עוצמת הרצף. אולם עדיין מתקבלת ההרגשה שהדוגמאות הנגדיות הן "מעטות", ושאולי בהנתן זוג כיוונים, אפשר עדיין לשחזר ביחידות את "רוב" הגופים בעזרתו. זה נכון, במובן הבא:

הגדרה 14 *קבוצה (במרחב מטרי) נקראת <u>מקטגוריה ראשונה</u> אם היא איחוד בן־מניה של קבוצות שאינן צפופות בשום מקום. קבוצה נקראת <u>רזידואלית</u> אם היא המשלים של קבוצה מקטגוריה ראשונה.*

קבוצה נקראת G_δ אם היא חיתוך בן־מניה של קבוצות פתוחות.

המרחב המטרי השלם עליו אנו מדברים הוא מרחב הקבוצות הקמורות המישוריות, עם מטריקת האוסדורף. נסביר מדוע "קבוצה רזידואלית" הוא המובן המעניין של "רוב הקבוצות": ממבט ראשון אפשר לחשוב שסתם קבוצה צפופה במרחב מטרי מכילה את רוב הנקודות (כזכור, במרחב שלנו כל נקודה מיצגת קבוצה מישורית). אכן, כל נקודה במרחב ניתנת לקרוב על־ידי נקודות בקבוצה, אך עדיין חסר משהו: בישר הממשי למשל, קבוצת הרציונליים היא צפופה, אך קשה להגיד עליה שהיא כוללת "את רוב"

המספרים: למעשה, עוצמתה קטנה מעוצמת כל הישר, והמשלים שלה (המספרים האי־רציונליים) גם הוא צפוף. ההגדרה של קבוצה רזידואלית פותרת בעיה זו: קבוצה מקטגוריה ראשונה היא קבוצה *קטנה*, ולכן קבוצה רזידואלית היא קבוצה שמכילה את *רוב* הנקודות במרחב, פרט לקבוצה *קטנה*.

במרחב מטרי שלם, קבוצה היא רזידואלית אם ורק אם היא מכילה קבוצת G_δ צפופה (ראה הוכחה, למשל, ב־[25, עמודים 158–160].

המשפט הבא, שמראה שאכן "רוב" הקבוצות הקמורות ניתנות לשחזור מזוג כיוונים נתון, הוכח לראשונה ב־[28], אך הוכחה מסודרת נמצאת גם ב־[11].

משפט 25 *בהנתן שני כיוונים שונים, קבוצת כל הגופים הקמורים שנקבעים על־ידי הטלותיהם בכיוונים אלו היא רזידואלית במרחב K_0^2 (מרחב הקבוצות הקמורות במישור, עם מטריקת האוסדורף).*

רעיון ההוכחה הוא כזה: נסמן את קבוצת כל הגופים הקמורים שנקבעים על־ידי הטלותיהם בזוג כיוונים נתון על־ידי D. מכיוון ש־K_0^2 הוא מרחב שלם, כדי להראות ש־D רזידואלית הכרחי ומספיק להראות ש־D היא G_δ צפופה:

1. כדי להראות ש־D היא G_δ מראים שהמשלים הוא F_σ, כלומר איחוד בן־מניה של קבוצות סגורות. המשלים של D היא קבוצת הגופים K כך שקיים K' שונה ולשניהן אותן הטלות בזוג הכיוונים הנתון. לכל n מגדירים A_n להיות קבוצת הגופים הקמורים $K \notin D$ כך שקיים K' (קמור, עם אותן הטלות בשני הכיוונים) שבנוסף מקיים $n \leq \delta(K,K') \leq n+1$ או $1/(n+1) \leq \delta(K,K') \leq 1/n$. ברור ש־$K_0^2 \backslash D = \cup_n A_n$, ומוכיחים שכל A_n הוא סגור.
2. כדי להראות ש־D צפופה, מראים איך אפשר לקרב כל גוף קמור על־ידי גוף שלו בטוח יש שחזור יחיד מהטלותיו בכיוונים הנתונים. בבנית הקרוב בונים גוף שחלקו קרוב חלק וחלקו קרוב מצולע, ומראים שהוא ניתן לשחזור יחיד מהטלותיו.

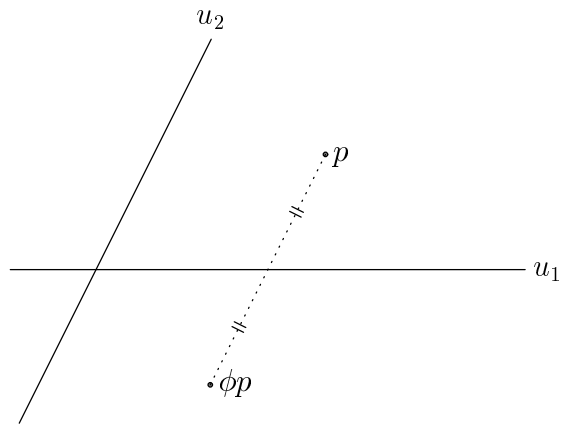
2.3.3 אימות קבוצות קמורות

בסעיפים הקודמים ראינו מה אפשר לומר על כיוונים שמאפשרים שחזור יחיד של כל גוף קמור, וכיוונים שמאפשרים שחזור יחיד של רוב הגופים הקמורים. לעיתים מעונינים בתוצאה קצת שונה: נתון גוף קמור K שצורתו ידועה מראש. נתון גוף קמור נוסף, K', שרוצים לבדוק בעזרת הטלות האם הוא זהה ל־K. במקרה זה מותר לבחור כיוונים התלויים ב־K הידוע, ויש לבחור כיוונים כאלו שיבטיחו שההטלות של K ו־K' בכיוונים אלו זהות אם ורק אם K ו־K' זהים (כלומר, כיוונים שיבטיחו שאף גוף קמור לא נותן אותן הטלות בכיוונים אלו כמו K). ראו הגדרה 4 בפרק 2.1.

בסעיף 2.2.1 ראינו הגדרה של קבוצה בת־חסימה בקבוצת כיוונים מסוימת. גם בהנחות של סעיף זה (שחזור גופים קמורים מבין משפחת הגופים הקמורים) ניתן להגדיר גוף בר־חסימה בקבוצת כיוונים סופית S, ולהוכיח שגוף כזה ניתן לאימות (שחזור יחיד) על־ידי אותה קבוצת כיוונים S:

הגדרה 15 *תהיה S קבוצת כיוונים סופית. נאמר שגוף קמור K הוא בר־חסימה בכיוונים S (S-inscribable) אם פנים K הוא איחוד הפנימים של מצולעים קמורים החסומים ב־K (כלומר, קדקדיהם על שפת K), שכל צלע שלהם מקבילה לאחד הכיוונים ב־S.*

משפט 26 *תהיה S קבוצת כיוונים סופית, ויהיה K גוף קמור בר־חסימה בכיוונים S. אזי K נקבע ביחידות (מבין הקבוצות הקמורות) על־ידי הטלותיו בכיוונים S.*



איור 2.8: האיזומטריה ϕ
Figure 2.8: The isometry ϕ

משפט זה נובע ישירות מלמה 1 בפרק 2.2.2: מלמה זו, כל מצולע שהוזכר בהגדרת "K בר־חסימה" מוכל בכל גוף קמור שמקבל אותן הטלות כמו K בכיוונים S. לכן זה נכון גם לאיחוד המצולעים, אך מההגדרה איחוד המצולעים הוא K, ולכן K מוכל בכל גוף קמור שמקבל אותן הטלות כמו K בכיוונים S. אך מכיוון שמידת גוף שמקבל אותן הטלות כמו K חייבת להיות כמידת K, נובע שגוף כזה לא יכול להכיל ממש את K, וחייב להיות K עצמו.

המשפט הבא מראה שלמרות שהראנו כבר שאין אף שלשת כיוונים שמאפשרת שחזור יחיד של כל גוף קמור, עדיין ניתן לאמת כל גוף קמור על־ידי שלושה כיוונים (התלויים בגוף):

משפט 27 *ניתן לאמת גוף קמור על־ידי הטלות בשלושה כיוונים.*

ההוכחה המקורית היתה של גירינג, אך הוכחה חדשה יותר מופיעה אצל גרדנר [8, 11]. רעיון ההוכחה: שוב נשתמש בלמה 1 מפרק 2.2.2. קבוצת הכיוונים של קטעים ישרים על ∂K היא לכל היותר בת־מניה. נבחר כיוון ראשון u_1 השונה מכל הכיוונים הנ"ל. את הכיוון השני u_2 נבחר להיות מקביל למיתר שמחבר את שני הישרים התומכים ב־K המקבילים ל־u_1. תהיה V הקבוצה הסגורה של הקדקדים של המקביליות החסומות ב־K שצלעותיהן מקבילות ל־u_1 ול־u_2. נסמן ב־E את הקצוות של רכיבי הקשירות של $\partial K \backslash V$. אם $E = \emptyset$ אז מהלמה מספיקים שני הכיוונים u_1,u_2. אם $E \neq \emptyset$, אז E לכל היותר בת מניה, ונבחר את הכיוון השלישי u_3 כך שלא יקביל לאף קו המחבר שתי נקודות של E. אז מראים ששלושת כיוונים אלו מאפשרים לאמת את K.

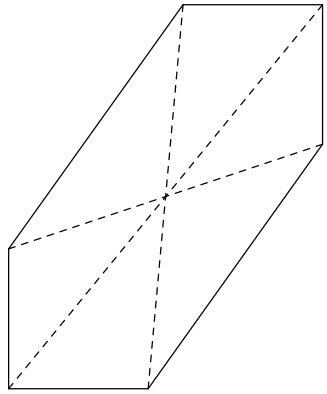
פחות כיוונים לא מספיקים לאמת כל גוף קמור:

משפט 28 *קיים מצולע קמור שלא ניתן לאימות בעזרת הטלותיו מכל זוג כיוונים.*

ההוכחה היא כמובן על־ידי דוגמה, שניתנה לראשונה על־ידי גרדנר [8, 11] (אך מתבססת על טכניקה של גירינג):

יהיה K גוף קמור עם סימטריה מרכזית (סימטרי לגבי הראשית: $x \in K \iff -x \in K$). תהיה ϕ איזומטריה שלוקחת נקודה p לנקודה ϕp על הישר בכיוון u_2 מ־p, כך של־ϕp יש אותו מרחק מרחק כמו p מ־l_{u_1} (הישר מהראשית בכיוון u_1). במילים אחרות, ϕ היא שיקוף דרך l_{u_1} במקביל ל־u_2 (ראה איור 2.8). נסתכל על

$$\phi K = \{\phi p | p \in K\}$$



איור 2.9: מצולע קמור שלא ניתן לאימות משני כיוונים
Figure 2.9: A convex polygon that cannot be verified from two directions

ϕK גם הוא גוף קמור. ברור מהגדרתו של־ϕK אותן הטלות כמו ל־K בכיוון u_2. יתר על כן, קל לראות ש־$S_{u_1}\phi K(x) = S_{u_1}K(-x)$ ומכיוון של־K סימטריה מרכזית (ולכן בפרט $S_{u_1}K(-x) = S_{u_1}K(x)$) נובע של־$K$ ו־ϕK אותן הטלות גם בכיוון u_1.

ריבוע J שמרכזו בראשית כמעט נותן דוגמה לגוף שלא ניתן לאימות משני כיוונים (ראה גם פרק 2.5). לכל זוג כיוונים מלבד כיווני צלעות הריבוע וכיווני אלכסוניו, ϕJ הוא גוף קמור שונה (מקבילית) עם אותן הטלות בזוג הכיוונים. הבעיה היא שלשני זוגות אלו, $\phi J = J$, ולכן נאלץ לקחת דוגמה מסובכת קצת יותר: ניקח משושה עם סימטריה מרכזית ושאלכסוניו לא מקבילים לצלעותיו (ראה איור 2.9). במקרה זה אפשר לבדוק שאכן $\phi K \neq K$ לכל זוג כיוונים (גרדנר לא מביא הוכחה לכך, הוכחה שלנו מופיעה בלמה הבאה).

■

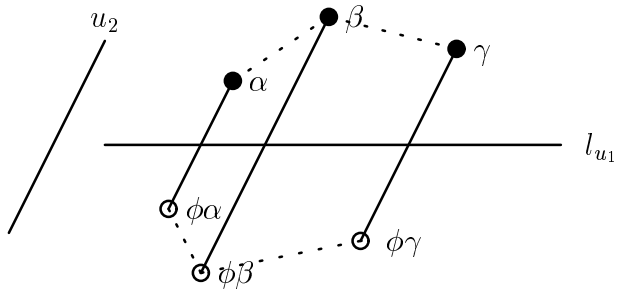
***למה 2** יהיה K משושה בעל סימטריה מרכזית, ונניח שאף־אחד מאלכסוניו לא מקביל לאף־אחת מצלעותיו. אז לכל זוג כיוונים שונים u_1, u_2, מתקיים $\phi K \neq K$ (כאשר ϕ טרנספורמצית השיקוף דרך l_{u_1} במקביל ל־u_2 שתוארה לעיל).*

הוכחה: יהיו u_1, u_2 נתונים, ונניח בשלילה ש־$\phi K = K$. נבדיל בין שני מקרים עיקריים, בהתאם לשאלה האם u_1 הוא אלכסון של המשושה K או לא:

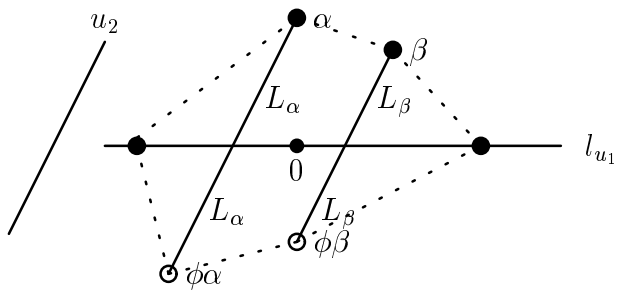
מקרה א': נניח שהישר l_{u_1} (הישר המקביל ל־u_1 העובר דרך הראשית) אינו חותך אף קדקד של המשושה. בגלל הסימטריה המרכזית של המשושה, ברור ששלושה מקדקדיו צריכים להיות מצד אחד של l_{u_1} ושלושה מצידו השני. נסמן שלושה קדקדים מצד אחד של l_{u_1} באותיות α,β,γ (בסדר רכיב u_1 עולה, למשל).

הנחנו ש־$\phi K = K$, דבר שאומר שאם α קדקד של K, אז גם $\phi\alpha$ הוא קדקד של K. איור 2.10 מראה את שלושת הקדקדים ותמונות ϕ שלהם. ברור שאכן תמונות ϕ נותנות נקודות שונות מ־α,β,γ כי הנחנו ששלוש נקודות אלו מצד אחד של קו־השיקוף l_{u_1}. שלוש הנקודות ושלוש התמונות נותנות ביחד ששה קדקדים שונים, ולכן אלו כל קדקדי המשושה. המשושה קמור ולכן ברור שצלעותיו הן המקווקוות באיור. אך אז קיבלנו שיש צלע $\alpha - \phi\alpha$ שמקבילה לאלכסון $\beta - \phi\beta$, בסתירה להנחת המשפט.

מקרה ב': המקרה השני הוא שהישר l_{u_1} עובר דרך קדקד של המשושה. אך ישר זה עובר גם בראשית ולכן מהסימטריה המרכזית הוא חותך שני קדקדים של המשושה. שוב מסימטריה מרכזית, ארבעת



איור 2.10: מקרה א'
Figure 2.10: First case



איור 2.11: מקרה ב'
Figure 2.11: Second case

הקדקדים הנותרים מתחלקים לפי שניים מעל הישר ושניים מתחתיו. נסמן את שני הקדקדים מעל הישר ב־α ו־β (ראה איור 2.11). שוב, מכיוון שהנחנו $\phi K = K$, הנקודות המשוקפות $\phi\alpha$ ו־$\phi\beta$ גם הן קדקדים של K, ולכן מוסיפים אותן בציור.

לא יתכן שהראשית נמצאת מצד אחד של שני הקטעים $\alpha - \phi\alpha$, $\beta - \phi\beta$, כי אז היו מצד אחד של ישר דרך הראשית חמישה קדקדים של K ומצד שני רק אחד, בסתירה להנחה ש־K בעל סימטריה מרכזית. לכן הראשית נמצאת בין שני הקטעים, כפי שרואים באיור 2.11.

נסמן את המרחק בין α לישר l_{u_1} לאורך הקטע בשיפוע u_2 ב־L_α ובדומה ל־β ב־L_β (ראה איור 2.11). בגלל הסימטריה המרכזית, המרחק בין α לראשית שווה למרחק בין $\phi\beta$ לראשית ולכן נובע $L_\alpha = L_\beta$. אך אז הצלע $\alpha - \beta$ מקבילה לישר l_{u_1}, שהוא אלכסון (כי יש עליו את שני הקדקדים כמו בציור), בסתירה להנחה שצלע של K לא יכולה להיות מקבילה לאלכסון של K.

■

אגב, השיקוף ϕ יכול לשמש במקרים רבים נוספים כדי להוכיח שגוף בעל סימטריה מרכזית לא נקבע ביחידות מהטלותיו בזוג כיוונים נתונים — ראה למשל משפט 30 בהמשך (בסעיף 2.5).

2.3.4 קביעה בשלבים של קבוצות קמורות

נניח שצורת גוף קמור אינה ידועה, ורוצים לקבוע אותה בעזרת הטלות. במקום לקבוע את כל כיווני ההטלות מראש, אפשר לקבוע כל כיוון בהסתמך על תוצאות ההטלות בכיוונים הקודמים. אם ניתן בצורה זו להבחין בין הגוף הנתון לכל גוף קמור אחר, נאמר שהגוף ניתן לקביעה בשלבים (ראה הגדרה 5 בפרק 2.1). רעיון זה הופיע לראשונה ב־[6], בו מופיע גם המשפט הבא:

משפט 29 *<u>מצולעים</u> קמורים ניתנים לקביעה בשלבים (מבין כל הגופים הקמורים) על־ידי הטלות משלושה כיוונים.*

ההוכחה קצרה ויפה: קל לראות שגוף קמור הוא מצולע אם ורק אם סימטרל שטיינר שלו הוא מצולע (ראה את הלמה הבאה). לכן אפשר לדעת מההטלות שהגוף הוא מצולע, ולמעשה מספיק להשוות את הגוף למצולעים קמורים (ולא צריך לכל הגופים הקמורים).

יהיה Q מצולע קמור, ו־u_1,u_2 שני כיוונים שונים כלשהם. ישר בכיוון u_i עובר דרך קדקד של Q אם ורק אם הוא פוגש קדקד של סימטרל שטיינר המתאים S_uQ. לכן קדקדי Q חייבים להיות ברשת הסופית F של נקודות החיתוך של שתי משפחות של קווים מקבילים, אלו שמקבילים ל־u_i ועוברים דרך קדקדים של $S_{u_i}Q$ ($i = 1, 2$).

עכשיו בוחרים את הכיוון השלישי u_3 כך שלא יהיה מקביל לאף קו המחבר שתי נקודות מ־F. נקודה היא קדקד של Q אם ורק אם היא נקודה ב־F שישר דרכה בכיוון u_3 עובר דרך קדקד של $S_{u_3}Q$.

■

ממשפט 28, אי אפשר לשפר את הטענה האחרונה לשני כיוונים.

למה 3 *יהיה K גוף קמור. אם סימטרל שטיינר S_uK הוא מצולע, אז גם K מצולע.*

הוכחה: נסמן ב־x קואורדינטה הניצבת לכיוון הנתון u. K קמור, אז אפשר להציג אותו כשתי פונקציות $f_1(x)$ עליונה קעורה ו־$f_2(x)$ תחתונה קמורה (f נותן את הקואורדינטה u של הנקודה). עלינו להראות ש־f_1 ו־f_2 הן פוליגונליות (לינאריות למקוטעין).

נתון ש־S_uK מצולע, ולכן התחום ב־x מחולק למספר סופי של קטעים, בכל אחד מהם ההטלה היא פונקציה לינארית. נראה שבכל תחום כזה גם f_1, f_2 חייבות להיות לינאריות.

יהיה $[x_1, x_2]$ תחום בו הסימטרל לינארי, כלומר

$$f_1(x) - f_2(x) = ax + b \tag{2.9}$$

ל־$x \in [x_1, x_2]$ (a,b קבועים). נוכיח בשלילה ש־f_1 לינארי: כאמור, f_1 הוא קעור. אם היה לא לינארי, מתקיים אי־שוויון חריף:

$$f_1\left(\frac{x_1 + x_2}{2}\right) > \frac{f_1(x_1) + f_1(x_2)}{2}$$

ואז מתקיים מ־(2.9) ומאי השוויון האחרון (שימו לב: a קבוע, לא פונקציה)

$$\begin{aligned} f_2\left(\frac{x_1 + x_2}{2}\right) &= f_1\left(\frac{x_1 + x_2}{2}\right) - a\left(\frac{x_1 + x_2}{2}\right) - b \\ &> \frac{f_1(x_1) + f_1(x_2)}{2} - \frac{ax_1 + b + ax_2 + b}{2} = \frac{f_2(x_1) + f_2(x_2)}{2} \end{aligned}$$

כלומר f_2 פונקציה קעורה ממש — וזה הרי לא יתכן כי הרי כאמור הפונקציה התחתונה f_1 היא קמורה.

ובכן f_1 לינארית בתחום המדובר, ומ־(2.9) גם f_2 לינארית בתחום זה, וזה מה שרצינו להוכיח.

■

2.4 מוגדרות־היטב ויציבות

בהתחלת הפרק ראינו משפטים רבים שמדברים על אפשרות שחזור יחיד של גופים מהטלותיהם. בכל המשפטים הנחנו שידועים לנו במדויק הכיוונים הנבחרים, ההטלות, ותנאים נוספים. השאלה עליה נתעכב בפרק זה היא שאלת היציבות: מה קורה כשהנתונים אינם בעלי דיוק אין־סופי? האם זה הופך חלק מהמשפטים שראינו לחסרי־משמעות במציאות, ולא מוגדרים היטב?

אי־הדיוקים הבאים יכולים להכנס לבעיות שחזור אמיתיות:

1. "מכונת הקרנה" יכולה לדעת את הזוויות בהן הקרינה את הגוף, אך הדיוק סופי ומוגבל. במשפט 22 למשל, קיבלנו אפיון של קבוצות של ארבעה כיוונים "טובים", שמבטיחים שחזור יחיד של כל גוף קמור. אך הקבוצה של רביעיות הכיוונים ה"לא טובים" היא צפופה, ולכן למעשה הכיוונים אותם לקחנו עלולים לא לאפשר שחזור יחיד בגלל טעות אפסילונית.
2. ההטלות נמדדות בדיוק סופי, ועלולות להכיל גם רעשי מדידה. יתר על כן, במציאות ההטלה נדגמת על מספר נקודות (או קרניים) סופי, וקרוב פונקצית־ההטלה הרציפה בעזרת נקודות מדודות אלו גורמים לאי־דיוק נוסף. ראינו משפטים שבהנתן זוג הטלות בכיווני הצירים, קובעים האם הטלות אלו הן של קבוצה אחת, יותר מקבוצה אחת, או של אף קבוצה. טעות אפסילונית בהטלות עלולות להעביר אותן מקטגוריה אחת לשניה. כמו כן, יש לבדוק האם יתכן שלשתי קבוצות בעלות הטלות "קרובות" מתאימים שחזורים "לא קרובים".
3. מספר משפטים הניחו הנחות מסוימות על הקבוצה אותה אנו מטילים: קבוצה קמורה, מצולע, וכו. במציאות, קשה להגדיר תנאים אלו: האם קבוצה "כמעט" קמורה, למשל שנוצר לה חריץ קטן על השפה, מקיימת את המשפטים של קבוצה קמורה? האם יש בכלל קבוצות קמורות שאינן מצולעים, כי הרי כל קבוצה קמורה ניתנת לקרוב טוב כרצונינו על־ידי מצולע?

לצערנו, מרבית שאלות היציבות של בעית השחזור הן פתוחות, ולכן במרבית המקרים אין תשובה תאורטית לשאלה האם ניתן לשחזר, באופן מעשי, גופים מסוימים מכיוונים מסוימים. אולם, בעבודה זו נראה בעזרת שימוש בשני אלגוריתמי שחזור שונים, אחד של גרדנר (פרק 3) ושני שלנו (מינברס, פרק 4) ששחזור אפשרי באופן מעשי, ברוב המקרים אותם ניסינו — גם במקרים בהם אנחנו לא מכירים משפט יחידות שחזור ומשפט יציבות.

ביתר סעיף זה נביא סקירה של משפטי היציבות שבכל־זאת ידועים לבעית השחזור:

2.4.1 משפט Volčič

Volčič [26] הוכיח שבעית השחזור של גוף קמור מארבעה כיוונים שמבטיחים שחזור יחיד היא מוגדרת היטב.

לשם כך הוא נדרש להראות את רציפות ההופכי של הפונקציה שלוקחת גוף (מישורי) קמור לרביעית הגופים הקמורים שהם הסימטרלים של שטינר בארבעת הכיוונים. (אגב, את ההוכחה שסימטרל שטיינר של גוף קמור גם הוא קמור ניתן למצוא למשל ב־[11])

או כדי לנסח יותר במדויק: תהיה S רביעית כיוונים שונים נתונה. יסמנו $S_i(K)$ סימטרלי שטיינר של K ביחס לכיוונים $u_i \in S$ אלו ($i = 1 \ldots 4$). נסמן ב־σ את המיפוי

$$\sigma(K) = (S_1(K), S_2(K), S_3(K), S_4(K))$$

מ־K_0^2 ל־$K_0^2 \times K_0^2 \times K_0^2 \times K_0^2$ (כאשר K_0^2 הוא מרחב הקבוצות הקמורות במישור). כפי שראינו, עבור בחירות מסוימות של רביעיות כיוונים σ היא חד־חד־ערכית, ונניח שאכן רביעית הכיוונים הנתונה

היא כזאת (אגב, הבעיה של אפיון הטווח של σ היא עדיין בעיה פתוחה, ולבעיה של חישוב σ^{-1} ניתנו רק פתרונות ללא הוכחה מלאה, כמו האלגוריתם של גרדנר בפרק 3 ואלגוריתם מינברס שאנו מציעים בעבודה זו בפרק 4). מה ש־Volčič הוכיח הוא שאם לוקחים ב־K_0^2 את הטופולוגיה הנובעת ממטריקת האוסדורף, אז σ זו רציפה, והפונקציה ההפוכה רציפה גם היא. לכן, טוען Volčič, בעית השחזור של Gardner-McMullen (שחזור גוף קמור מארבעה כיוונים) היא מוגדרת היטב.

2.4.2 הערכות Longinetti

Longinetti [20] הוכיח שאם ההטלות של שני גופים קמורים K, K' זהות מ־n כיוונים שונים, אז

$$\lambda_2(K\Delta K') \leq \frac{tan(\pi/n)}{8n}\lambda_1\left(\partial(K\cap K')\right)^2$$

עם שוויון רק במקרה ש־K הוא מצולע משוכלל עם n צלעות ו־K' הוא K מסובב בזווית π/n סביב מרכזו.

אגב, חסם עליון ל־$\lambda_1(\partial(K\cap K'))$ אפשר למצוא מתוך ההטלות. הבעיה בהערכה זו היא שאינה אינווריאנטית תחת טרנספורמציות אפיניות. משפט קודם של Longinetti נתן הערכה אינווריאנטית, שבה $\lambda_1(\partial(K\cap K'))$ מוחלף ב־$\lambda_2(K\cap K')$, אך המקדם שם אינו ידוע ל־$n>6$.

הערכה זו של Longinetti מענינת, אך נראה שלא מלמדת אותנו דבר לגבי יציבות שחזור יחיד: השוויון מתקבל במקרה ה"גרוע" ביותר שכבר הזכרנו, של שני מצולעים משוכללים מסובבים, אך לא לומדים ממנו כלום על מה קורה במקרה "אופיני", לא־כל־שכן במקרה שבו התאוריה מבטיחה יחידות שחזור (ולכן $\lambda_2(K\Delta K')$ אמור להיות אפס).

ב־[19] Longinetti נותן הערכה מענינת נוספת, מסוג שונה: נניח ש־K ו־K' הם גופים קמורים כך שלכל הכיוונים u, $||X_uK - X_uK'||_\infty < \epsilon$. אז $\lambda_2(K\Delta K') \leq C\epsilon^2$ כאשר $C \leq 14.2$ הוא קבוע בלתי תלוי ב־K או ב־K'. כמו כן, אם $K\cap K' \neq \emptyset$ אז $\delta(K,K') < 3\epsilon$ (δ היא מטריקת האוסדורף).

כאמור, התוצאה הנ"ל מדברת על הטלות מכל הכיוונים (ידוע שמהטלות ממספר אין־סופי של כיוונים יש יחידות שחזור). אם מדובר על מספר סופי של כיוונים, תוצאה אפסילונית כזו בלתי אפשרית, כי הרי יש K,K' עם הטלות זהות ב־n כיוונים (למשל, שני המצולעים המשוכללים שהזכרנו), אבל $\lambda_2(K\Delta K')$ אינו אפס. האם אפשר להוכיח תוצאה כזו במקרה של־K ומספר סופי של כיוונים נתונים ידוע משפט יחידות? כרגע לא ידועה תשובה מלאה לשאלה זו, אך אפשר לראות שבכל זאת יש מגבלות ליציבות:

2.4.3 מגבלות על משפטי יציבות של בעית השחזור

הדוגמה הבאה מראה מדוע משפט Volčič בעצם לא פתר את כל בעיות היציבות של בעית השחזור מארבעה כיוונים, ויתר על כן היא מראה לנו איזה סוג של משפטי יציבות *לא* נוכל לצפות לקבל עבור בעית השחזור.

נסתכל על K_1,K_2 שני גופים קמורים עם אותן ההטלות ברביעית כיוונים "גרועה" S (כלומר, רביעית כיוונים שלא מאפשרת שחזור יחיד). המרחק (למשל במטריקת האוסדורף) בין K_1 ל־K_2 הוא חיובי קבוע. נשנה את קבוצת הכיוונים ל־S' קרובה כרצונינו שהיא "טובה" (קבוצת רביעיות הכיוונים שמאפשרות שחזור יחיד של כל גוף קמור היא צפופה בקבוצת כל רביעיות הכיוונים). ההטלה של גוף קמור רציפה בכיוון (קל לראות, לפחות במטריקת L_2 או במטריקה על סימטרלי שטיינר) ולכן הטלות K_1 בכיוונים S' קרובות לאלו ב־S כרצונינו (עלינו לבחור S' לפי ה־ϵ הדרוש). אלו שוות להטלות K_2 ב־S (ההנחה המקורית) ושוב אלו קרובות להטלות K_2 ב־S' כרצונינו.

ובכן, יש $K_1 \neq K_2$ כך שלכל $\epsilon > 0$ יש רביעית כיוונים "טובה" S' כזו ש־K_1 ו־K_2 נותנים אותן הטלות בכיוונים S' עד כדי ϵ (במטריקת L_2).

אפשר בצורה דומה למצוא מגבלה על יציבות השחזור מזוג כיוונים נתונים:

נסתכל על שני גופים קמורים K_1,K_2 עם אותן הטלות בשני כיוונים נתונים (לדוגמה, שני המשושים שראינו במשפט 28). הוכחנו ש"רוב" הצורות הקמורות ניתנות לשחזור יחיד מכיוונים אלו, אז יש K_1' קרוב ל־K_1 כרצונינו ו־K_2' קרוב ל־K_2 כרצונינו כך ש־K_1',K_2' ניתנים לשחזור יחיד מהטלותיהם בזוג הכיוונים הנתון. K_1' ו־K_2' רחוקים (למשל, במטריקת האוסדורף) מכיוון שהם קרובים ל־K_1 ו־K_2 בהתאמה שהיו רחוקים זה מזה. אבל ההטלות של K_1',K_2' קרובות כרצונינו (במטריקת L_2), בגלל רציפות ההטלה לשינוי הגוף.

ובכן, בהנתן זוג כיוונים, יש קבוע $a > 0$ כך שלכל $\epsilon > 0$ יש זוג גופים קמורים K_1', K_2' שהמרחק בינהם לפחות a, שכל אחד מהם ניתן לשחזור יחיד מהטלותיו בכיוונים הנתונים, אך הטלות בכיוונים אלו של שני הגופים קרובים עד כדי ϵ (במטריקת L_2).

2.5 דוגמת הריבוע

השאלה האם ניתן לשחזר באופן יחיד גוף מסוים מהטלותיו בזוג כיוונים מסויים, מבין כל הגופים הקמורים או מבין הגופים הכוכביים או הקשירים, היא שאלה קשה. למרות שראינו במשפט 25 בסעיף 2.3.2 שבהנתן זוג כיוונים, ניתן לשחזר ביחידות את "רוב" הגופים הקמורים (מבין משפחת הקבוצות הקמורות), אין לנו אפיון כללי של גופים קמורים הניתנים לשחזור מהטלות משני כיוונים נתונים — וזאת בניגוד מוחלט למצב של שחזור במשפחת הקבוצות המדידות, שם ידוע משפט לורנץ (משפט 3 בסעיף 2.2.1), וקריטריונים נוספים הכרחיים ומספיקים ליחידות שחזור. גרדנר [11] מציין את בעית אפיון הגופים הקמורים הניתנים לשחזור יחיד (מבין הגופים הקמורים) מהטלות בשני כיוונים נתונים, כאחת הבעיות הפתוחות החשובות של התחום.

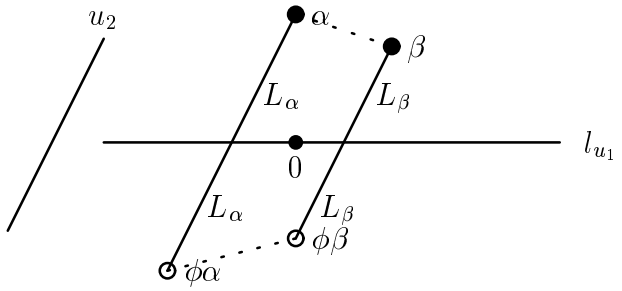
אחד הגופים הפשוטים ביותר שאפשר לחשוב עליהם הוא הריבוע. רוצים היינו לחשוב שאם אפשר לשחזר משני כיוונים את "רוב הגופים", שאפשר לשחזר גם ריבוע. אך לצערנו אי אפשר: לכל זוג כיוונים מלבד זוג כיווני הצלעות של הריבוע או זוג כיווני האלכסונים של הריבוע, הריבוע אינו נקבע ביחידות מבין הקבוצות הקמורות מתוך זוג הטלות אלו (האשם הוא בסימטריה של הריבוע — ראה את המשפט הבא). עבודות קודמות, למשל [11], מזכירות עובדה זו אך ההוכחה שלה ושאר התוצאות בסעיף זה הן מקוריות.

משפט 30 *יהיה J ריבוע. יהיו u_1 ו־u_2 זוג כיוונים שונים. אזי:*

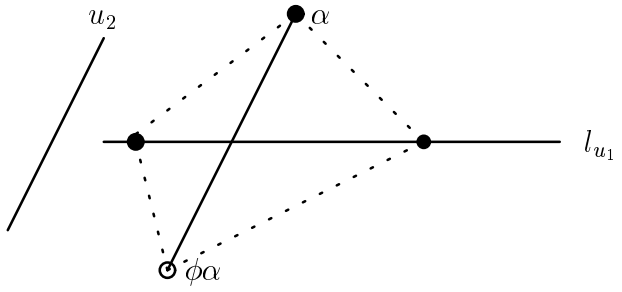
1. *אם u_1, u_2 הם זוג כיווני הצלעות של הריבוע או זוג כיווני האלכסונים אז הריבוע נקבע ביחידות מבין הקבוצות הקמורות, ואפילו מבין הקבוצות המדידות, בעזרת ההטלות מזוג כיוונים אלו.*
2. *לזוגות כיוונים u_1, u_2 אחרים, קיימת צורה קמורה שונה J' (היא מקבילית) בעלת אותן הטלות בכיוונים אלו.*

הוכחה:

1. חלק זה של המשפט נובע ישירות ממשפט 32 (סעיף 4.3.1): כל אחד מזוגות כיוונים אלו הם כיוונים ניצבים, הריבוע סימטרי יחסית אליהם, וחיתוכו עם ישרים בכיוונים אלו הוא תמיד קטע (כמובן, הריבוע אפילו קמור).
2. הוכחת חלק זה דומה להוכחה שראינו ללמה 2 (סעיף 2.3.3): נניח ללא הגבלת הכלליות שמרכז הריבוע בראשית. הריבוע בעל סימטריה מרכזית. נגדיר את ϕ — השיקוף דרך l_{u_1} במקביל ל־u_2 כמו בלמה הנ"ל. כמו בלמה, ל־ϕJ אותן הטלות בכיוונים u_1,u_2 כמו ל־J. ברור שאם J קמור אז



איור 2.12: מקרה א'
Figure 2.12: First case



איור 2.13: מקרה ב'
Figure 2.13: Second case

גם ϕJ, וכמו כן ש־ϕ מעביר ישרים מקבילים לישרים מקבילים (הוא אפילו העתקה לינארית) ולכן ϕJ הוא מקבילית. נותר להראות ש־$\phi J \neq J$:

יש שתי אפשרויות:

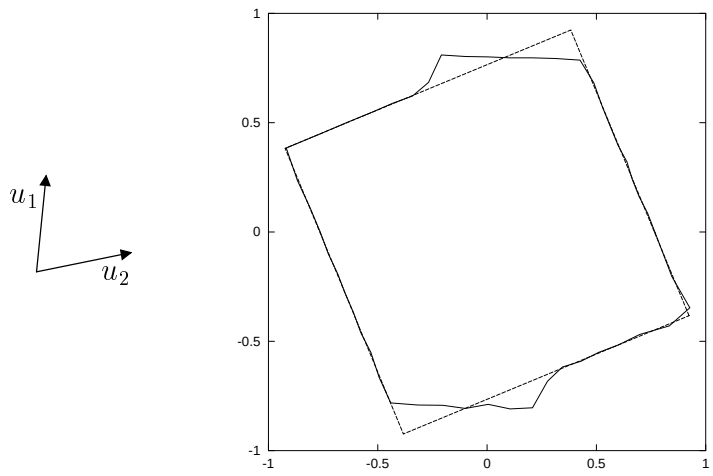
(א) אם אין אף קדקד של J על הישר l_{u_1}: בגלל הסימטריה המרכזית של הריבוע J, שניים מקדקדיו מעל הישר הנ"ל ושנים מתחתיו. נסמן את שני הקדקדים מעל הישר ב־α ו־β. נניח בשלילה ש־$\phi J = J$, כלומר ששיקוף־ϕ של שני קדקדים אלו גם הם קדקדים בריבוע. לכן הריבוע הוא כמו באיור 2.12, וזוג אחד של צלעותיו מקביל ל־u_2. מהסימטריה המרכזית, ברור שהראשית בין צלעות אלו, כמו באיור, ואז מהסימטריה המרכזית נובע גם ש־L_α ו־L_β באיור שווים. אך מזה נובע שצלעות $\alpha - \beta$ ו־$\phi\alpha - \phi\beta$ מקבילות ל־l_{u_1}. לסיכום, זוג הכיוונים שלנו הוא זוג כיווני הצלעות, בסתירה להנחה.

(ב) אם יש קדקד של J על הישר l_{u_1} אז בגלל הסימטריה המרכזית יש שניים על הישר, והנותרים משני צדדיו: קדקד אחד מכל צד שלו. נסמן את הקדקד מעל הישר ב־α. אם נניח בשלילה ש־$\phi J = J$ אז גם $\phi\alpha$ קדקד, ולכן הריבוע נראה כמו באיור 2.13. אך זה אומר ש־u_2 מקביל לאלכסון אחד של הריבוע, $\alpha - \phi\alpha$, ו־u_1 מקביל לאלכסון השני: l_{u_1}. לסיכום, זוג הכיוונים שלנו הוא זוג כיווני האלכסונים, בסתירה להנחה.

ובכן ϕJ הוא ה J' המבוקש.

■

כאשר מדברים על משפחת הקבוצות המדידות, ידוע שכאשר אין יחידות אפשר למצוא מספר אינסופי



איור 2.14: דוגמה לשחזור צורה שונה מהריבוע המסובב
Figure 2.14: Example of reconstruction of a shape different from the rotated square

של קבוצות שנותנות את אותן הטלות (ראה משפט 6 בסעיף 2.2.1). כאשר מדובר על משפחות קטנות יותר, משפט מסגנון זה אינו ידוע. האם המקבילית היא הצורה הקמורה היחידה מלבד הריבוע שנותנת אותן הטלות בזוג הכיוונים הנתון? האם המקבילית היא הצורה הכוכבית היחידה מלבד הריבוע שנותנת אותן הטלות בזוג הכיוונים הנתון?

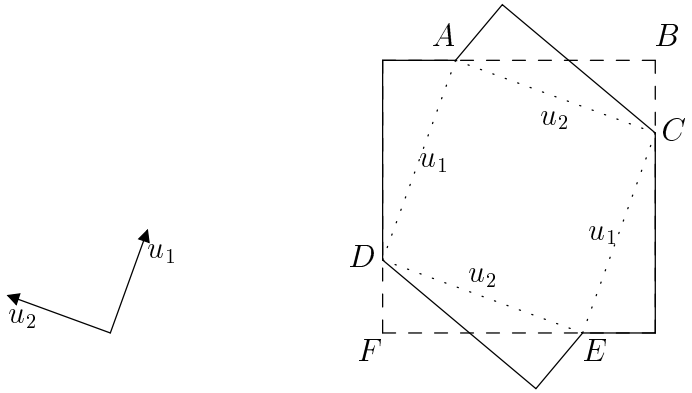
התשובה לשאלה האחרונה היא לא, לפחות לחלק גדול מזוגות הכיוונים (זוגות כיוונים ש"רחוקים מספיק" זה מזה — בפרט זוגות של כיוונים ניצבים). ניסויים בתוכנת השחזור הכללית "מינברס" שכתבנו, ושתתואר בפרק 4, העלו צורה נוספת, לא קמורה, מלבד הריבוע, שנותנת אותן הטלות (ראה איור 2.14). מהסתכלות בצורה שהוציאה התוכנית הצלחנו למצוא את הביטוי האנליטי של הגוף האחר, הכוכבי אך לא קמור, והוכחנו שאכן הוא נותן אותן הטלות.

בהמשך סעיף זה נראה איך בונים את הגוף הלא־קמור הנ"ל שנותן את אותן הטלות כמו הריבוע בזוג הכיוונים הנתונים.

2.5.1 אינטואיציה לדוגמה הלא־קמורה

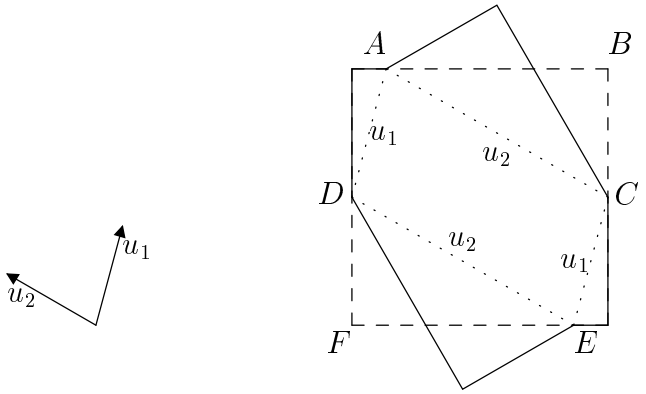
איור 2.15 מראה (בקווים מלאים) דוגמא לצורה נוספת שנותנת אותן הטלות כמו הריבוע (בקווים מקווקוים) בזוג כיוונים ניצבים נתונים, u_1 ו־u_2. זוהי בדיוק הצורה שהתקבלה ב"ניסוי" השחזור של הריבוע בתוכנית המינברס. קל להבין אינטואיטיבית מדוע לצורה החדשה ישנן אותן הטלות בזוג הכיוונים: במקרה זה בצורה החדשה "הפכנו" שני משולשים על בסיסם (ראה איור 2.15). ההטלות בכיוון u_2 לא השתנו מפעולה זו כמובן, ואילו ההטלות בכיוון u_1 לא השתנו מכיוון שבתחום בו הן היו עלולות להשתנות, הסימטריה של הפיכת שני המשולשים נותנת בדיוק את אותן הטלות שהיו בריבוע עם המשולשים המקוריים. מובן שניתן היה להפוך דווקא את צמד המשולשים השני, ובכך לקבל צורה שקולה נוספת. יתר על כן, ניתן לבצע את שני ההיפוכים — ומתברר שמתקבלת מקבילית, אותה מקבילית שראינו כבר קודם.

איור 2.16 מראה דוגמה לצורה שנותנת אותן הטלות כמו הריבוע, בשני כיוונים לא ניצבים u_1, u_2. גם במקרה זה מוצאים מקבילית חסומה בריבוע שצלעותיה מקבילות לכיוונים הנתונים, ומבצעים



איור 2.15: הריבוע וצורה שקולה נוספת

Figure 2.15: The square and another equivalent shape



איור 2.16: הריבוע וצורה שקולה נוספת, זווית לא ישרה

Figure 2.16: The square and another equivalent shape, non-orthogonal directions

טרספורמציה מסוימת על המשולשים (הפעם היא לא היפוך פשוט — נראה מהי בהמשך), ששומרת על ההטלות בשני הכיוונים. כדי שכיוונים u_1,u_2 יאפשרו בנית דוגמה כזו, הם יצטרכו להיות "רחוקים" מספיק (בפרט, ניצבים), כפי שנראה בהמשך, וכמובן בהתאם למשפט הקודם הם לא יכולים להיות כיווני הצלעות או האלכסונים של הריבוע.

ובכן, נדגים את קיומה של צורה שקולה כזו לריבוע, לכל זוג כיוונים "שונים מספיק", בשני שלבים: נראה איך ניתן לחסום בריבוע המקורי מקבילית שצלעותיה מקבילות לכיוונים הנתונים, ואז נראה איך ניתן לשנות את זוג המשולשים תוך שמירה על ההטלות בשני הכיוונים.

חשיבות המקבילית החסומה בריבוע נובעת מהטענה הבאה:

משפט 31 *יהיה K ריבוע במישור. יהיו u_1, u_2 זוג כיוונים כך שקיימת מקבילית Q חסומה ב־K (שקדקדיה על הקף הריבוע), שצלעותיה מקבילות לכיוונים הנתונים. אזי Q מוכל, עד כדי קבוצה בעלת מידה 0, בכל קבוצה מדידה E בעלת אותן הטלות כמו הריבוע K בכיוונים הנתונים.*

כלומר, בבניה שבה שינינו את המשולשים אך שמרנו על המקבילית החסומה ללא שינוי, למעשה נאלצנו לעשות כך.

טענה זו נובעת מלמה כללית יותר: למה 1 בפרק 2.2.2.

2.5.2 מציאת המקבילית החסומה

נתון ריבוע, וזוג כיוונים u_1, u_2. עלינו להוכיח שניתן לחסום בריבוע מקבילית שקדקדיה על צלעות הריבוע, וצלעותיה מקבילות לכיוונים u_1, u_2. לדוגמה, ראינו את המקבילית $ACED$ באיור 2.16.

ראשית נשים לב לא לכל בחירה של זוג כיוונים u_1, u_2 קיימת מקבילית חסומה כמו בציורים, שבה כל קדקד על צלע שונה של הריבוע. ברור שלשם כך חייבים להיות סימני השיפועים הפוכים, וכמו כן אסור שהזווית בין הכיוונים תהיה קטנה מדי (זווית שראשה על צלע אחת של הריבוע, ושוקיה עוברות בשתי הצלעות השכנות לצלע זו, לא יכולה להיות קטנה מ־$45°$).

למעשה, כדי שהבניה שנראה אכן תצליח ותתן צורה לא־קמורה, צריכים להתקיים כל התנאים הבאים:

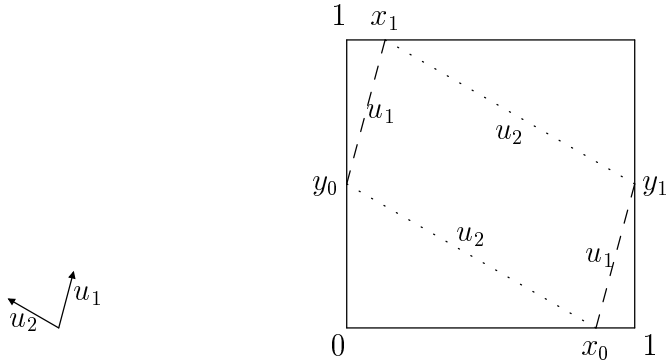
1. אף אחד מהכיוונים אינו כיוון של צלע או אלכסון של הריבוע (אם זוג הכיוונים הוא זוג כיווני הצלעות או זוג כיווני האלכסונים, ראינו כבר שהריבוע נקבע ביחידות. אם אחד הכיוונים הוא כיוון צלע, הצורה הנוספת היא משושה קמור, ואם אחד הכיוונים הוא כיוון אלכסון, אין צורה נוספת לפי בניה זו אבל יש את המקבילית).
2. סימני השיפועים הנתונים הפוכים.
3. אם השיפוע החיובי קטן מ־$45°$, אז השלילי קטן מ־$-45°$. אם השיפוע החיובי גדול מ־$45°$, אז השלילי גדול מ־$-45°$.

הוכחת טענה זו מתבצעת בדומה לבנית המקבילית בהמשך סעיף זה, ונחזור אליה בסוף הסעיף. אגב, בפרט כל זוג כיוונים ניצבים (מלבד כיווני הצלעות וכיווני האלכסונים של הריבוע) מקיימים תנאים אלו.

כעת נמצא נוסחה למקבילית החסומה, כאשר היא קיימת. נסמן ב־m_1 ו־m_2 את שיפועי הכיוונים u_1 ו־u_2 בהתאמה. בהתאם להנחות לעיל, נניח ללא הגבלת הכלליות ש־m_1 הוא חיובי ו־m_2 שלילי.

נעביר שני ישרים מקבילים בכיוון u_1 בשתי נקודות (כרגע נעלמות) $(x_0, 0)$ ו־$(0, y_0)$ כמו באיור 2.17. ישרים אלו חותכים את שתי הצלעות הנוספות של הריבוע בנקודות חדשות $(x_1, 1)$, $(1, y_1)$. מתקיימים הקשרים הבאים:

$$1 - y_0 = m_1 x_1 \tag{2.10}$$



איור 2.17: מציאת המקבילית החסומה בריבוע

Figure 2.17: Finding a parallelogram inscribed in a square

$$y_1 = m_1(1 - x_0) \tag{2.11}$$

עכשיו, כדי שתתקבל מקבילית, היינו רוצים שהכיוון מ־$(x_1, 1)$ ל־$(1, y_1)$, והכיוון מ־$(0, y_0)$ ל־$(x_0, 0)$, יהיו שניהם הכיוון u_2. לשם כך צריכות להתקיים שתי המשואות:

$$y_0 = -m_2 x_0 \tag{2.12}$$

$$1 - y_1 = -m_2(1 - x_1) \tag{2.13}$$

מהצבת משוואות (2.10),(2.11) ב־(2.13) מקבלים:

$$1 - m_1(1 - x_0) = -m_2\left(1 - \frac{1 - y_0}{m_1}\right) \tag{2.14}$$

נציב בזה את (2.12):

$$1 - m_1(1 - x_0) = -m_2\left(1 - \frac{1 + m_2 x_0}{m_1}\right) \tag{2.15}$$

מכאן אפשר לחלץ את x_0:

$$x_0\left(m_1 - \frac{m_2^2}{m_1}\right) = m_1 - 1 - m_2 + \frac{m_2}{m_1}$$

$$x_0\left(m_1^2 - m_2^2\right) = (m_1 - m_2)(m_1 - 1)$$

כלומר

$$x_0 = \frac{m_1 - 1}{m_1 + m_2} \tag{2.16}$$

בעזרת (2.12) נקבל את y_0:

$$y_0 = -m_2 x_0 = \frac{m_2 - m_1 m_2}{m_1 + m_2} \tag{2.17}$$

נשים לב שהתקבל ש־

$$x_1 = \frac{1-y_0}{m_1} = \frac{1+m_2}{m_1+m_2} = 1-x_0$$

ובדומה $y_1 = 1-y_0$, כפי שאכן היה צפוי מטעמי סימטריה (או יותר בדיוק: מחפיפת משולשים עם זוויות שוות משיקולי קווים מקבילים, וצלע אחת שווה).

נחזור כעת בקצרה להוכחת התנאים לקיום המקבילית שראינו קודם: הנוסחאות דומות לאלו שראינו קודם, אבל בשביל לקבל תוצאה סימטרית יותר, נסמן את השיפועים ב־$-u, 1/v$, כאשר $u, v > 0$. אם אכן קיימת מקבילית כמו באיור 2.17, נסמן לשם פשטות $x = x_0$, $y = y_1$ ואז

$$\begin{aligned} v &= \frac{1-x}{y} \\ u &= \frac{1-y}{x} \end{aligned}$$

אם נחלץ את x ו־y נקבל:

$$\begin{aligned} x &= \frac{1-v}{1-uv} \\ y &= \frac{1-u}{1-uv} \end{aligned}$$

אבל, כדי שהמקבילית הנ״ל אכן תהיה קיימת, צריך כמובן ש־$0 < x < 1$ ו־$0 < y < 1$. התנאים ל־u, v שנותנים x, y כאלו הם בדיוק התנאים שאנו מחפשים. קל לראות שהתנאי $0 < x < 1$ נותן:

$$0 < 1-v < 1-uv \quad \text{או} \quad 1-uv < 1-v < 0$$

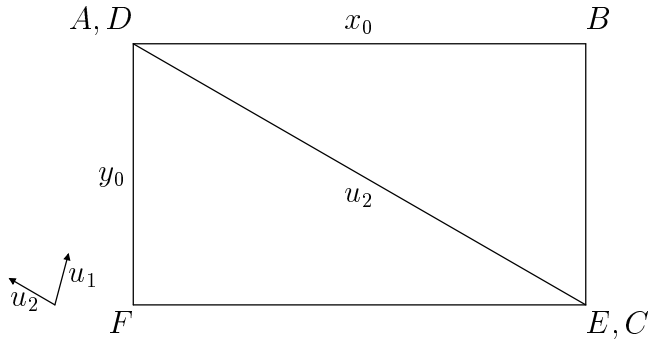
(התנאי $0 < y < 1$ נותן משהו דומה), ואילו זה שקול לכך ש u, v שניהם קטנים מ־1, או שניהם גדולים מ־1. זהו בדיוק התנאי שהזכרנו קודם על השיפועים.

2.5.3 הטרנספורמציה למשולשים

עכשיו נראה את הטרנספורמציה שצריך לעשות למשולשים כדי לשמור על ההטלות. לכיוונים ניצבים, טרנספורמציה זו היא פשוט היפוך המשולשים על בסיסם, כפי שראינו באיור 2.15, אך לכיוונים לא ניצבים היא מסובכת מעט יותר, כפי שנראה עכשיו (ראה איור 2.16). כדי להחליף משולש כזה במשולש אחר עם אותו בסיס ואותן הטלות בכיוון u_2, הכרחי ומספיק שלמשולש החדש יהיה <u>גובה</u> זהה לזה של המקורי, מדמיון משולשים. ובכן הקדקד החדש של המשולש צריך להמצא באיזשהו מקום על הקו בכיוון u_2 שיוצא מהקדקד הישן.

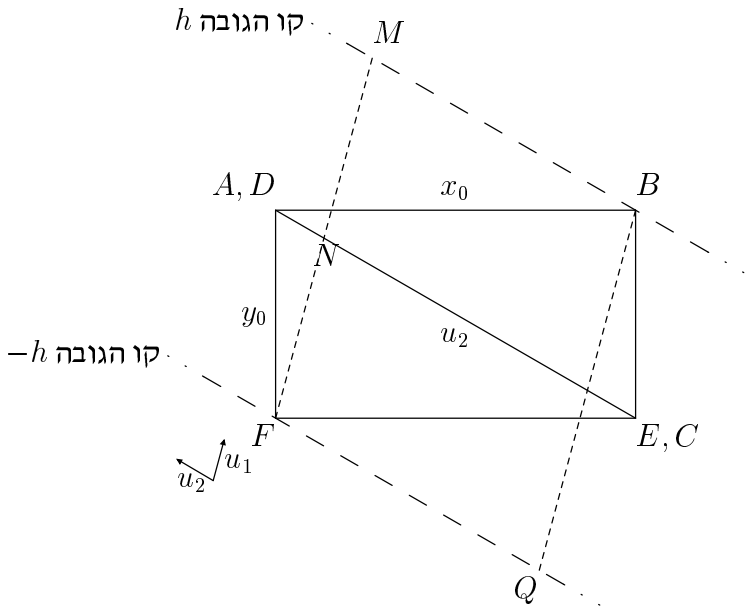
את מיקום קדקד זה על הישר הנ״ל נבחר (ונראה שאפשר באמת לעשות זאת) כדי שההטלות בכיוון u_1 של הצורה החדשה יהיו זהות לאלו של הריבוע המקורי.

ברור שההטלות בכיוון u_1 לא ישתנו מחוץ לתחום שבין הקווים המקבילים ל־u_1 באיור 2.16, וכמו כן מכיוון שאנו לא מתכיוונים לשנות את המקבילית הפנימית, תרומתה להטלות לא משנה וניתן להוציא אותה. במילים אחרות מספיק להסתכל על הציור 2.18: שני המשולשים ABC ו־DEF מחוברים בבסיס משותף ויוצרים מלבן שאחד אלכסוניו הוא הכיוון הנתון u_2. עלינו למצוא צורה נוספת הבנויה משני



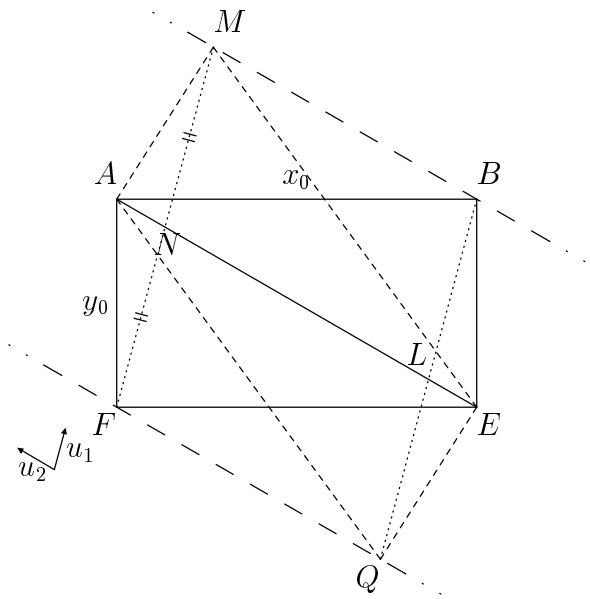
איור 2.18 : שני המשולשים מונחים אחד ליד השני

Figure 2.18: The two triangles laid one beside the other



איור 2.19 : קווי הגובה ובנית הקדקדים החדשים

Figure 2.19: The equidistant lines and the new vertices



איור 2.20: שני המשולשים, לפני ואחרי הטרנספורמציה

Figure 2.20: The two triangles, before and after the transformation

משולשים על אותו בסיס (ולפי ההערה מקודם, עם אותו גובה, כדי לשמור על אותן הטלות בכיוון u_2) שנותנת אותן הטלות בכיוון u_1.

איור 2.19 מראה בקווים מקווקוים את הישרים בכיוון u_2 עליהם עלינו להסיע את קדקדי המשולשים כדי לשמור על גובה שווה (ולכן אותן הטלות בכיוון u_2). כאמור, מטרתנו היא לבנות מעל הבסיס (אלכסון המלבן, AE) זוג משולשים אחר, שקדקדיהם הנוספים על הישרים המקווקוים, כך שישמרו ההטלות גם בכיוון u_1. הרעיון בו נשתמש הוא לדאוג שכל כמות מסה שהקרן עברה במשולש העליון לפני הטרנספורמציה, היא תעבור במשולש התחתון אחרי הטרנספורמציה, ולהפך.

אם נעביר ישר FNM בכיוון u_1 מהקדקד F (ראה איור 2.19) הקטע באורך FN שהיה במשולש התחתון לפני הטרנספורמציה צריך להיות בעליון אחריה, כלומר הקטע NM (ברור $|NM| = |FN|$) צריך להיות במשולש העליון אחרי הטרנספורמציה. כלומר, M הוא קדקד המשולש החדש מעל הבסיס. באותו אופן, Q, שנוצר על הגובה $-h$ מחיתוך של ישר בשיפוע u_1 מ־B, הוא קדקד המשולש החדש מתחת לבסיס.

צריך עדיין להוכיח שנקבל את אותן הטלות בכל ההטלות בכיוון u_1 — ולא רק בהטלות המסוימות שבדקנו ליד הקדקדים. אבל זה ברור מדמיון משולשים: נסתכל על קרן מקבילה ל־MF (כלומר ל־u_1) שחותכת את AE בנקודה $N + \alpha(E - N)$ בקטע NE (בקטע AN הטיפול דומה). לפני הטרספורמציה (קווים מלאים באיור 2.20), קרן זו חותכת במשולש העליון את המסה $|BL| \cdot (|AN| + \alpha)/|AL|$, ובתחתון את המסה $|FN| \cdot \alpha/|NE|$. אחרי הטרנספורמציה, אותו ישר חותך במשולש העליון את המסה $|MN| \cdot \alpha/|NE|$ — שזהה למסה שקודם נחתכה במשולש התחתון (כזכור, $|MN| = |NF|$, כך בדיוק בחרנו את M), ובדומה, הישר חותך במשולש התחתון את המסה $|LQ| \cdot (|AN| + \alpha)/|AL|$ — שזהה למסה שקודם נחתכה במשולש העליון (מכיוון ש־$|BL| = |LQ|$).

כעת, נמצא ביטוי מפורש לנקודה M (ראה איור 2.20): בהנחה ששני הכיוונים אינם הכיוון האנכי, נסמן ב־m_1 ו־m_2 את שיפועי הכיוונים u_1 ו־u_2 בהתאמה. כמו כן נניח שהראשית בנקודה A ($A = (0, 0)$).

אז משוואת הישר BM היא

$$y = m_2 x + y_0 \tag{2.18}$$

כי האלכסון AE בשיפוע m_2 עובר בראשית A, ואילו MB בגובה y_0 מעליו. משוואת הישר FM היא:

$$y = m_1 x - y_0 \tag{2.19}$$

הנקודה M בחיתוך שני הישרים:

$$m_1 x - y_0 = m_2 x + y_0 \tag{2.20}$$

אז

$$x = \frac{-2y_0}{m_2 - m_1} \tag{2.21}$$

$$y = m_1 x - y_0 = -y_0 \frac{m_2 + m_1}{m_2 - m_1} \tag{2.22}$$

כלומר לסיכום

$$M = A - y_0 \left(\frac{2}{m_2 - m_1}, \frac{m_2 + m_1}{m_2 - m_1} \right) \tag{2.23}$$

אגב, כשזוג הכיוונים ניצבים, קיבלנו ש־$|MA| = |BE|$ כלומר המשולש החדש הוא פשוט היפוך הישן על בסיסו. נבדוק זאת:

$$\frac{M - A}{|BE|} = \frac{M - A}{y_0} = -\left(\frac{2}{m_2 - m_1}, \frac{m_2 + m_1}{m_2 - m_1} \right) \tag{2.24}$$

שכאשר הכיוונים ניצבים, כלומר $m_1 = -1/m_2$, הופך ל:

$$\frac{M - A}{|BE|} = -\left(\frac{2m_2}{{m_2}^2 + 1}, \frac{{m_2}^2 - 1}{{m_2}^2 + 1} \right) \tag{2.25}$$

ואז

$$\frac{|MA|}{|BE|} = \frac{(2m_2)^2 + ({m_2}^2 - 1)^2}{({m_2}^2 + 1)^2} = 1 \tag{2.26}$$

פרק 3

אלגוריתם גרדנר לשחזור מארבעה כיוונים

משפט 22 מבטיח את יחידות השחזור של קבוצה קמורה במישור מבין הקבוצות הקמורות, בהנתן הטלות בארבעה כיוונים (תחת מגבלות מסוימות). אולם, הוכחת המשפט איננה קונסטרוקטיבית ואיננה מאפשרת שחזור הגוף, כאשר ידועות הטלותיו.

גרדנר [11, עמוד 47] נותן אלגוריתם לשחזור הגוף, בהנחה ששפת הגוף הקמור היא חלקה וללא קטעים ישרים. אולם ההצדקה התאורטית לאלגוריתם זה לא מלאה, כפי שנראה, גם מבלי לקחת בחשבון את בעית היציבות שהזכרנו בסעיף הקודם. כמו כן, יש לזכור שאלגוריתם זה של גרדר מתאים רק לשחזור מהטלות בארבעה (או יותר) כיוונים, אפילו כשידוע שניתן לשחזר גוף מסוים עם שלוש או אפילו רק שתי הטלות.

רעיון האלגוריתם של גרנר הוא שבהנתן הטלות בארבעה כיוונים, מוצאים נקודות שחייבות להיות על שפת גוף קמור K שמתאים להטלות אלו. נתחיל בלמה שמוצאת 3 נקודות על שפת הגוף, ואז נמשיך, בעזרת ההטלות הנתונות, לקבל עוד ועוד נקודות. כפי שנראה, אפשר להוכיח שנקבל בצורה זו מספר אין־סופי של נקודות על שפת הגוף, אבל השאלה האם הסגור של קבוצת הנקודות שקיבלנו אכן נותן את כל שפת הגוף עודנה פתוחה.

3.1 מציאת 3 נקודות על שפת K

נראה שבהנתן הטלות של K בשלושה כיוונים, נוכל למצוא שלוש נקודות שחייבות להיות על שפת K. למעשה, מכיוון שנתונות לנו הטלות בארבעה כיוונים, אפשר למצוא ארבעה זוגות של משולשים חסומים כאלו — אך לנו מספיק רק משולש חסום אחד.

למה 4 *יהיה K גוף קמור ויהיו u_i, $1 \leq i \leq 3$, שלושה כיוונים שונים. אז קיימים לפחות שני משולשים (אולי מנוונים) החסומים ב־K (כלומר, קדקדיהם על שפת K) שצלעותיהם מקבילות לכיוונים u_i. יתר על כן, אם K חלק, אז משולשים אלו אינם מנוונים.*

למה 5 *יהיה K גוף קמור ויהיו u_i, $1 \leq i \leq 3$, שלושה כיוונים שונים. ניתן למצוא את המשולשים מלמה 4 מהטלות K בשלושת כיוונים אלו.*

הוכחת הלמה האחרונה קונסטרוקטיבית, ומספקת אלגוריתם למציאת משולש כזה בהנתן הטלות משלושה כיוונים:

ל־$1 \le i \le 3$, תהיה x_i נקודה בתחום בו $X_{u_i}K$ לא אפס. נסמן ב־$T = T(x_1, x_2, x_3)$ את המשולש שנוצר על־ידי הישרים $l_{u_i} + x_i$. נסמן ב־E_i את חצי־המישור הסגור שמוגדר על־ידי הישר $l_{u_i} + x_i$ ושאינו מכיל את T. לכל קבוצה A נסמן $A_i = A \cap E_i$. נסתכל על הפונקציה

$$f(x_1, x_2, x_3) = \lambda_2(T) + \sum_{i=1}^{3} \lambda_2\left((S_{u_i}K)_i\right) \tag{3.1}$$

שניתנת לחישוב מההטלות הנתונות בכיוונים u_i, $1 \le i \le 3$. אבל, ממשפט פוביני, $\lambda_2(K_i) = \lambda_2\left((S_{u_i}K)_i\right)$ לכל i. לכן

$$f(x_1, x_2, x_3) = \lambda_2(T) + \sum_{i=1}^{3} \lambda_2(K_i) \tag{3.2}$$

אגף ימין לא קטן מ־$\lambda_2(K)$, מכיוון שהקבוצות T ו־K_i, $1 \le i \le 3$, מכסות את K. יתר על כן, אם קדקדי T לא על שפת K, f מקבלת ערך גדול יותר מ־$\lambda_2(K)$: אם יש ל־T קדקד בתוך K אז הקבוצות K_i נחתכות וסכום השטחים ב־f גדול משטח K, ואילו אם יש ל־T קדקד מחוץ ל־K הרי ש־T מכסה שטח שמחוץ ל־K ולכן איחוד T ו־K_i מכיל שטח שמחוץ ל־K (אך גם את כל K), ובפרט סכום השטחים ב־f גדול משטחו של K.

ולכן הערך המינימלי של f, $\lambda_2(K)$, מתקבל רק במקרה שקדקדי T על שפת K. לכן ערכי x_i שעבורם המשולש T חסום ב־K ניתנים למציאה על־ידי מינימיזציה של הפונקציה הידועה f.

■

3.2 מציאת נקודות נוספות על שפת K

נתחיל עם קבוצת 3 הקדקדים שמצאנו בעזרת הכיוונים u_1, u_2, u_3 על שפת K: $V_1 = \{v_1, v_2, v_3\}$. נסתכל על קבוצת המיתרים ב־K שמקבילים לאחד מארבעת הכיוונים u_i, $1 \le i \le 4$, כשהקצה האחד שלהם נקודה ב־V_1 ושפוגשים את ∂T (יש לשים לב לנקודה זו) בנקודה נוספת. תמיד יהיו 4 מיתרים כאלו: שלוש צלעות T, ומיתר נוסף k שמקביל ל־u_4 ויוצא מאחד הקדקדים של V_1. נסמן ב־V_2 את קבוצת כל הקצוות של מיתרים אלו. אז V_2 מורכב מקדקדי T והקצה השני של k. את הקצה הזה של k אפשר למצוא בעזרת המידע על ההטלות (וההנחה ש־K קמור): ההטלה $X_{u_4}K$ נותנת את אורך k, והעובדה ש־$k \cap \mathrm{int}T \neq \emptyset$ (כי אמרנו ש־k פוגש את ∂K בנקודה נוספת) קובעת את הצד של v שבו k נמצא על הישר $l_{u_4} + v$. ובכן V_2, שכל הקדקדים בו הם קדקדי ∂K, ניתן לחישוב מההטלות הנתונות.

נמשיך באותה צורה באינדוקציה: בשלב ה־n נסתכל על כל המיתרים של K שמקבילים לאחד הכיוונים $u_i, 1 \le i \le 4$, ושלהם קצה אחד ב־V_n ושחותכים את שפת $\mathrm{conv}V_n$ בנקודה נוספת מלבד הקצה הנ״ל. נגדיר את V_{n+1} להיות קבוצת כל הקצוות של מיתרים אלו (כמו קודם, הקצוות החדשים ניתנים לחישוב מההטלות הנתונות).

בצורה זו מוצאים, תוך שימוש במידע על ההטלות הנתונות בלבד, קבוצות סופיות V_n של נקודות על שפת K, כאשר $V_n \subset V_{n+1}$ לכל n. נסמן $V = \cup_n V_n$.

גרדנר ממשיך ומוכיח (במקרה שארבעת הכיוונים הם כאלה שבאמת מבטיחים שחזור יחיד של כל גוף קמור), ש־V היא קבוצה אין־סופית, כלומר שהאלגוריתם הנ״ל לא יעצר בשום שלב. אבל השאלה החשובה, האם $\mathrm{cl}V = \partial K$ עודנה נותרה בעיה פתוחה.

3.3 מימוש האלגוריתם

מימוש האלגוריתם של גרדנר אינו פשוט כפי שנדמה ממבט ראשון. גרדנר עצמו אינו מציע מימוש לאלגוריתם ואינו מתיחס לנקודה זאת כלל, מלבד ההערה שהאלגוריתם מומש על־ידי סטודנט שלו. במימוש האלגוריתם הלכה למעשה נתקלים במספר סיבוכים. בסעיף זה נביא את הרעיונות והאלגוריתמים המרכזיים בהם השתמשנו כדי לממש את האלגוריתם של גרדנר. התוכנית המלאה (בשפת C) מופיעה בנספח ד.

נתחיל במימוש האלגוריתם למציאת שלוש הנקודות על השפה. המינימיזציה מתבצעת על פונקציה של שלושה פרמטרים, והם הזזות הישרים (שכיווניהם נתונים) בכיוון הניצב לכיווני הישרים עצמם. המינימיזציה הרב־מימדית (בעצם, 3 מימדית) נעשית בעזרת האלגוריתם "אמבה" מ־[23], שיתואר בנספח א (הוא ישמש אותנו בהמשך גם לתוכנית "מינברס").

החלק הקשה ביותר בחישוב הפונציה f הוא חישוב השטח $(S_{u_i}K)_i$, שהוא האינטגרל של פונקצית ההטלה בתחום שמעבר לישר i (חצי המרחב שלא כולל את המשולש). מימוש ראשוני שהשתמש בווריאציה על נוסחת הטרפז לאינטגרציה התברר כמאד לא מדויק, ונדרשו כ־2000 קרניים בכל כיוון כדי לקבל בצורה מדויקת מספיק את 3 הנקודות המבוקשות. אך למעשה, פונקצית ההטלה היא פונקציה חלקה (מלבד שתי נקודות בהן הנגזרת לא רציפה) ולכן היינו רוצים לצפות שמספר קרניים קטן יחסית יגדיר במדויק את פונקצית ההטלה, לה נעשה אינטגרציה מדויקת. לכן החלטנו לקרב את פונצית ההטלה על־ידי ספליין קובי, שמקבל בקרניים הנתונות את הערכים הנתונים. מציאת ספליין זה נעשה בעזרת אלגוריתם מ־[23]. האינטגרציה של הספליין יכלה להעשות בצורה אנליטית, אך העדפנו להשתמש באלגוריתם אינטגרציה נומרית כללי: השתמשנו באלגוריתם של רומברג מ־[23] שמתאפיין בהתכנסות מהירה ומדויקת. אחרי שיפור זה בשיטת האינטגרציה הספיקו בבעית דוגמה רק כ־20 קרניים (מאית מהמספר הקודם!) בכל כיוון כדי לקבל את 3 הנקודות בדיוק טוב.

אחרי מציאת שלוש הנקודות הראשונות על שפת הגוף, ממשיך כאמור האלגוריתם במציאת נקודות נוספות. הנקודות נמצאות בסדר לא ידוע מראש, ולכן כדי שנוכל לדעת בכל זמן את מבנה המצולע, בחרנו להשתמש במבנה נתונים יעודי: Sorted Star Polygon (מצולע כוכבי מסודר). למצולע נתון מרכז שאנו בטוחים שנמצא בתוכו (במקרה זה, ממוצע שלוש הנקודות שכבר מצאנו), ויתר הנקודות תהינה מסודרות סביב מרכז זה לפי סדר הזווית. מובן שההנחה שמצולע הוא כוכבי היא סבירה, מכיוון שהנחת המשפט אף חזקה יותר: המצולע שאנו נמצא הוא קמור.

לכל נקודה על המצולע נתון המיקום, הזווית שמשמשת לסידור הנקודות (המחושבת בעזרת הפונציה atan2 והכיוון אל נקודת המרכז הקבועה מראש), מצביעים אל הנקודה הבאה והקודמת במצולע (בסדר הנכון על המצולע!) וכן "דגל", המסמן האם עלינו לטפל בנקודה זו (כלומר, למצוא את הנקודות שממולה) בשלב הבא.

על מבנה נתונים זה מגדירים שתי פונקציות עיקריות: אחת add_ssp_node שמוסיפה נקודה חדשה למצולע, במקומה הראוי בסדר, ומחזירה את המיקום הנ"ל. בגלל שגיאות עיגול וקרוב, אנו עלולים "לגלות" שוב אותה נקודה שכבר נמצאת במצולע, ולכן פונקציה זו מקבלת אפסילון (למשל 10^{-3}) ובמקרה שהנקודה החדשה קרובה בזוויתה (ברדיאנים) עד כדי אפסילון זה לנקודה אחרת במצולע, לא מוספת נקודה חדשה. הפונקציה השניה, ssp_findflagged, מחזירה את רשימת כל הנקודות שהדגל שלהן מורם בזמן זה.

ובכן האלגוריתם למציאת נקודות נוספות על שפת הגוף פועל כך:

- שמים במבנה הנתונים (המצולע הכוכבי המסודר) את שלוש הנקודות הראשונות הידועות, ומרימים את דגליהן.

- עוברים על רשימת הקדקדים עם דגל מורם. לכל קדקד כזה, בודקים האם כל אחד מהכיוונים יוצא ממנו אל תוך הגוף (כלומר, האם אחד הכיוונים נמצא בתוך הזווית של המצולע בקדקד המדובר). אם כיוון מסוים מתאים, מוצאים את הנקודה "מולו" על־ידי אינטרפולציה של פונקצית ההטלה במקום זה, ובחירה נכונה של סימן הכיוון (כך שהקרן מצביעה אל תוך הגוף, לא החוצה ממנו). נקודה חדשה כזו מוספת למצולע (עם דגל מורם), אלא אם (כאמור) היא כבר נמצאת (עד כדי אפסילון) בו.
- חוזרים על השלב האחרון שוב ושוב, עד שאינו מוצא אף נקודה חדשה, או עברנו מספר איטרציות מקסימלי כלשהו שנקבע מראש.

כדאי לשים לב שאין למעשה צורך בקביעת מספר איטרציות מקסימלי, וקביעת האפסילון שתואר לעיל מספיקה למעשה כדי להבטיח את עצירת התוכנית. זאת מכיוון שמספר הנקודות המקסימלי שנוכל לקבל בסוף האלגוריתם, בהנחה שכל קדקד רחוק בזוויתו לפחות ב־ϵ מכל אחד משכניהו, הוא $2\pi/\epsilon$. אבל, בכל שלב נוספת למצולע לפחות נקודה אחרת (אחרת התוכנית היתה עוצרת) ולכן התוכנית תעצור תוך מספר סופי של שלבים. למעשה, בדוגמאות אמיתיות נוספות הרבה יותר מנקודה אחת בכל שלב, ובדוגמאות סבירות האלגוריתם מסיים לאחר כ־10 עד 30 שלבים.

3.4 בעיות באלגוריתם ובמימושו

כאמור, המשפט עליו גרדנר מבסס את אלגוריתם השחזור שלו אינו מבטיח כלל שהנקודות שמקבלים תהיינה צפופות על שפת הגוף אותה אנו מנסים לשחזר. במקרה הגרוע (שגרדנר משער, אך אינו מוכיח, שלא יכול לקרות) אנו עלולים להשאר עם קטע שפה שלם שלא שוחזר ואין עליו אף נקודה. בדוגמאות שניסינו, אגב, לא התגלתה דוגמה נגדית להשערת גרדנר.

יתר על כן, במימוש שלנו (ובכל מימוש מעשי אחר, כנראה) ישנן שתי בעיות נוספות:

- כדי להמנע מ"נקודות מרובות" בגלל אי דיוקים, השתמשנו כאמור באפסילון כדי למנוע נקודות חדשות שקרובות מדי לנקודות קודמות. אולם, עלול להתברר שבגלל שלא הוספנו נקודה חדשה, האלגוריתם נעצר לפני ש"כיסה" קטע שפה מסוים, ואילו כן היינו מוסיפים אותה הוא היה ממשיך (וממשיך ליצר נקודות קרובות זו לזו) ובסוף מכסה גם את קטע השפה ה"חסר".

  אם חושדים בבעיה זו, אפשר לקחת אפסילון קטן (למשל 10^{-3}). הנסיון מראה שאפסילון קטן יותר מזה לא מומלץ (אי־הדיוק של הטלת נקודה לנקודה מולה, לנקודה שלישית, ומשם בחזרה, הוא גדול יותר). אולם גם אפסילון גדול בהרבה (למשל $5 \cdot 10^{-2}$) נתן תוצאות טובות בדוגמאות מעשיות.
- באלגוריתם של גרדנר יש בעיה קשה של אי־דיוק מצטבר שאיננה קיימת באלגוריתמים גלובליים יותר כמו מינברס (שיתואר בפרק הבא). נקודה שנמצאת בשלב ה־20 מחושבת כמובן על־ידי 20 מעברים עוקבים מנקודה לנקודה שמולה, כשבכל מעבר, המרחק בין הנקודות מחושב על־ידי אינטרפולציה פונקצית ההטלה. אם אינטרפולציה זו איננה מדויקת מספיק, הנקודה ה־20 עלולה להיות רחוקה מאוד מהמקום הנכון שלה. האלגוריתם של גרדנר אינו נוטה לתקן טעויות כאלו, וכל טעות כזו מההווה בסיס לטעויות נוספות בעתיד. הסיכוי לטעויות כאלו גדול במיוחד כאשר הנתונים על ההטלות הגיעו ממדידה, ויש בהם אי־דיוק מסוים שאינהרנטי במדידה (במקרה זה יהיה כנראה צורך בהחלקה ותיקון מקדימים של הנתונים).

  יתר על כן, טעויות "מקרוסקופיות" במיקום הנקודות יכול לגרום לפתע למצולע קוצני, לא קמור, ומשמיט את הבסיס להנחות שמשמשות את השלבים הבאים באלגוריתם, ולגרום לבעיות רציניות,

כמו למשל החלטה לא נכונה על סימן הכיוון שצריך לקחת מנקודה ישנה לעבר נקודה חדשה (כאמור, אורך הקטע נתון מפונקצית ההטלה, אך את הכיוון יש לבחור כך שיצביע על תוך הגוף).

בעיות דוגמה הראו את בעיה זו כאשר נלקחו מספר קטן־יחסית של קרניים בכל כיוון, והקרוב של פונקצית ההטלה נעשה על־ידי ספליין, כפי שהוא תואר קודם. הבעיה נובעת מכך שקרוב ספלין ליד נקודות של אי רציפות בנגזרת (יש כאלו בנקודה בין התחום מחוץ לגוף שבו פונקצית ההטלה 0, והתחום בתוכו) הוא מאוד לא־מדיוק. כדי שלא להרגיש בבעיה זו נאלצים לבחור אחת מהאפשרויות הבאות:

1. לקרב את פונקצית ההטלה על־ידי ספליין למקוטעין (או לפחות, ספליין רק במקום בו הפונקציה לא 0, עם נקודה ונגזרת נכונה במקום של אי־הרציפות בנגזרת), במקום ספליין בודד לכל התחום. במקרה שאכן הקרניים הנתונות הן מדויקות (כלומר, זוהי בעיה מלאכותית ולא נובעת ממדידה) קרוב זה יהיה טוב בהרבה וימנע את הבעיות.
2. מראש, למדוד את ההטלה על יותר קרניים. זה פתרון פשוט וקל במיוחד לבעיות דוגמה מלאכותיות, כמובן. העלאת מספר הקרניים מצמצם את תחום אי־הדיוק של הספלין, ומקטין את אי־הדיוק. בדוגמה מסוימת, העלאת מספר הקרניים מ־100 ל־2000 פתר את כל הבעיות.
3. תיקון מספר חלקים באלגוריתם כך שיהיו פחות תלויים באי־דיוקים. ברור שאי הדיוק ישאר, ובמקום מצולע יפה אנו עלולים לקבל ענן נקודות קרובות למצולע (כאשר מחברים את הנקודות לפי הזווית, מקבלים מצולע "קוצני"). אבל, אילו לא היינו מבצעים טעויות "גסות" בקביעת הסימן של הכיוון כפי שתואר לעיל אז הבעיה לא היתה כה נוראה. יתכן שיש צורך לקבוע את הכיוון "אל תוך" הגוף בצורה יותר חכמה, ולא להסתמך על כיוונים מקומיים והנחת הקמירות שכבר לא נכונה כשישנם אי־דיוקים.
4. הגדלת האפסילון שמשמש לקביעה האם נקודה כבר נמצאת במצולע. כשאי־הדיוק רב, "מעגל" הטלות שאמור היה לחזור אל אותה נקודה, חוזר אל נקודה אחרת, לא מדויקת. הגדלת האפסילון יכול למנוע הופעות נקודות לא מדויקות חדשות אלו במצולע.

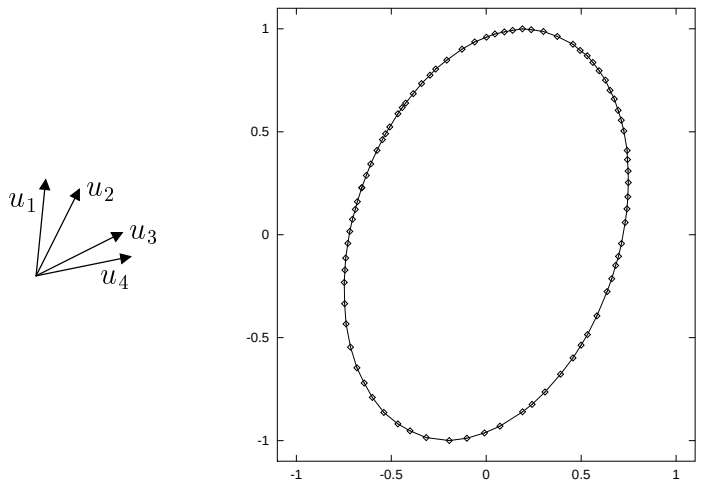
3.5 דוגמאות לשימוש באלגוריתם

כדי לבדוק את האלגוריתם של גרדנר, לקחנו במחשב צורות קמורות חלקות (שמתאימות לתנאי המשפט), ביצענו להן "הטלות" בארבעה כיוונים שבחרנו (בכל כיוון לקחנו מספר רב של קרניים, עד כדי מספר אלפים, בגלל הבעיה המתוארת בסעיף הקודם), ולהטלות שקיבלנו ביצענו שחזור בעזרת האלגוריתם של גרדנר.

ניתן לחזור על התוצאות שנראה בסעיף זה בעזרת התוכנית gardner שהשימוש בה מוסבר בנספח ב. לכל דוגמה נאמר מהו קובץ ההגדרה שיש להשתמש בו כדי לקבל את אותה תוצאה, בתוך סוגריים מרובעים בצורה [problems/...].

הדוגמה הראשונה היתה אליפסה מסובבת [problems/g2]. איור 3.1 מראה את ארבעת הכיוונים שבחרנו, ואת תוצאת השחזור עם אפסילון (הפרדה בין זויות) גדול יחסית. הנקודות, הרחוקות יחסית זו מזו, מחוברות בקווים ישרים.

איור 3.2 מראה את אותה בעיה, עם אפסילון קטן [problems/g4]. הפעם התקבלו 2928 נקודות על שפת הגוף ואנו מראים רק את הנקודות עצמן, ללא קווים בינהם — רואים שאכן הנקודות מכסות את כל שפת הגוף, ללא סימנים ל־"שטחים קרחים". אגב בדוגמה זו רואים שנמצאה נקודה אחת במקום לא נכון.

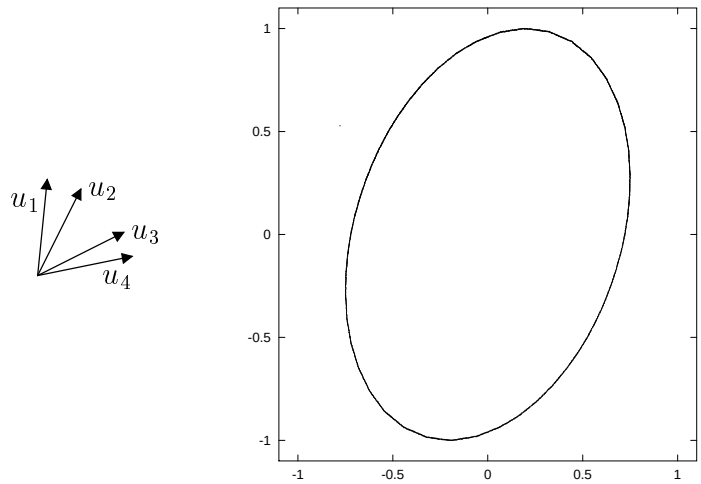


איור 3.1 : שחזור אליפסה בשיטת גרדנר. $\epsilon = 5 \cdot 10^{-2}$

Figure 3.1: Reconstruction of an ellipse using Gardner's algorithm. $\epsilon = 5 \cdot 10^{-2}$

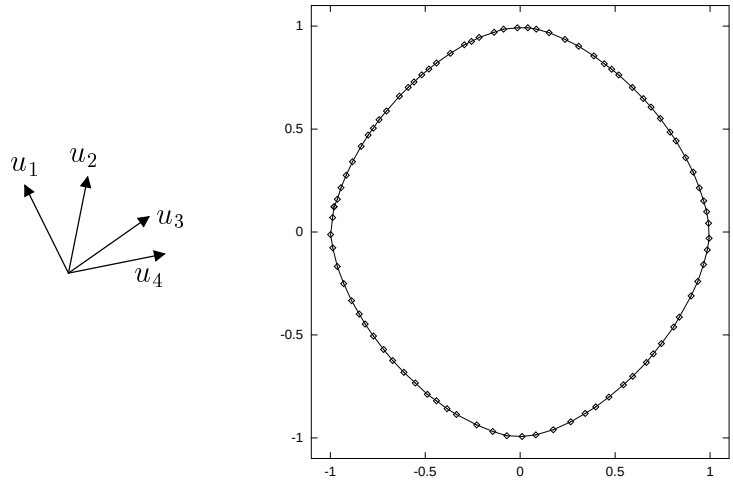
מסיבות שהוסברו קודם, הגדלת מספר הקרניים שלקחנו בכל כיוון מ־2000 ל־3000 [problems/g5] פתרה את בעית הנקודה הלא־נכונה.

הדוגמה הבאה שניסינו היא מעגל מוכלל עם חזקה 1.6 ($x^{1.6} + y^{1.6} = 1$). האלגוריתם שיחזר יפה גם צורה זו, כפי שניתן לראות באיור 3.3. [problems/gball1.6]



איור 3.2 : שחזור אותה אליפסה : $\epsilon = 10^{-3}$

Figure 3.2: Reconstruction of the same ellipse: $\epsilon = 10^{-3}$



איור 3.3 : שחזור מעגל מוכלל עם חזקה 1.6 : $\epsilon = 5 \cdot 10^{-2}$

Figure 3.3: Reconstruction of a generalized circle with power 1.6: $\epsilon = 5 \cdot 10^{-2}$

פרק 4

אלגוריתם השחזור "מינברס"

ראינו בפרק 2 מספר תנאים שונים שמבטיחים שנוכל לשחזר ביחידות גוף מסוים מהטלותיו בקבוצת כיוונים מסוימת. לדוגמה, משפט 22 מבטיח שבעזרת הטלות בארבעה כיוונים (תחת מגבלה מסוימת על בחירתם), ניתן לשחזר ביחידות כל גוף קמור מבין משפחת הגופים הקמורים. הוכחת המשפט הנ"ל איננה קונסטרוקטיבית, אך בפרק הקודם ראינו שלמקרה מיוחד זה של ארבעה כיווני הטלה מצא גרדנר אלגוריתם שחזור, למרות שהוא לא הצליח להוכיח שאלגוריתם זה מצליח תמיד.

[3, 4, 1] מצאו גם הם אלגוריתמים נוספים שמתאימים למקרים מאוד מיוחדים: שני הראשונים מראים אלגוריתם היוריסטי למקרה שהגוף קמור וסימטרי יחסית לשני צירים, והאחרון מראה אלגוריתם למציאת חורים עגולים (בלבד) בגוף קמור.

אך ישנם מיקרים אחרים בהם אנו יודעים בוודאות ששחזור יחיד קיים. לדוגמה, אנו יודעים ממשפט 25 (סעיף 2.3.2) שבהנתן שני כיוונים בלבד, ניתן לשחזר "כמעט כל גוף קמור" בעזרת הטלות בכיוונים אלו באופן יחיד מבין שאר הגופים הקמורים. הבעיה של אפיון הגופים שניתנים לשחזור בעזרת זוג כיוונים נתון עודנה בעיה פתוחה, ולכן כלי שחזור, לו היה לנו כזה, הוא כלי מחקר מעניין בהקשר זה.

כמו כן יתכנו מקרים נוספים בהם קיים משפט יחידות, אך הוא עדיין לא ידוע. לדוגמה, לא ידועים משפטים שאומרים מתי יש יחידות בשחזור קבוצות קשירות, או כוכביות, מבין משפחות הקבוצות הקשירות או הכוכביות בהתאמה. אפשר גם לשאול מתי יש יחידות בשחזור קבוצות עם חורים, קבוצות לא קשירות, וכדומה. לו היה לנו כלי שחזור שעובד גם במקרים אלו, הינו יכולים אולי להעלות השערות לגבי היחידות (או אי־היחידות) במקרים אלו.

לכן פיתחנו אלגוריתם שחזור חדש, "מינברס", עם המטרות הבאות:

- אפשרות לשחזור גופים כלליים יותר מגופים קמורים: הגדרנו הכללה של מצולעים כוכביים, שקראנו לה מצולע כוכבי שכבתי, שמאפשרת להגדיר גם מצולעים עם חורים ומצולעים לא קשירים — ראה סעיף 4.1.
- כלי השחזור יקבל מידע ממספר הטלות כלשהו: נקבל מספר סופי של "תוצאות קרניים", שהן קרניים נתונות ו"התוצאה" של כל קרן (כמות המסה שהקרן עברה דרכה). הקרניים הנ"ל לאו־דווקא מכוונות בארבע זוויות שונות (עקרונית, כל קרן יכולה להיות בזווית שונה, ואפשר להשתמש בכלי זה גם להקרנות מנקודה, אך אנו לא נעשה זאת בעבודה זו).
- הכלי אמור לחפש גוף, שלו היינו מעבירים דרכו את הקרניים הנ"ל, היו מתקבלות אותן תוצאות קרניים. הכלי לא יבטיח שחזור יחיד: תוצאות תאורטיות עלולות להבטיח שבמקרים מסויימים אין שחזור יחיד, או אין שחזור כלל. אך במקרה שישנן אפשרויות שחזור רבות, הוא ינסה למצוא

אחת מהן.

אלגוריתם השחזור שפיתחנו מתבסס על אלגוריתם מינימיזציה. הסבר תמציתי של האלגוריתם הוא כדלקמן: לגוף ניחוש מסויים מוצאים את תוצאות ההטלות, ולוקחים נורמה שמודדת את מרחק וקטור התוצאות מווקטור התוצאות המבוקש. לנורמה זו מנסים לעשות מיניזציה תוך שינוי הגוף, ומכריזים על הצלחה כשהנורמה התקרבה מספיק לאפס.

לאלגוריתם זה, הפותר את הבעיה ההפוכה להטלות (כלומר, בעית השחזור) תוך שימוש במינימיזציה, קראנו מינברס (Minverse), מהמילים האנגליות Minimization ו־Inverse. לסיכום, מינברס היא שיטה כללית לשחזור גופים דו־מימדיים מהטלותיהם.

מובן שכדי שאלגוריתם שחזור כזה (או כל אלגוריתם שחזור אחר) יצליח, לא מספיקה יחידות שחזור, ודרושה גם יציבות של בעית השחזור, בגלל הדיסקרטיזציה הנעשית באלגוריתם (של הגוף, של ההטלות, ושל הכיוונים). לצערנו, כפי שראינו בסעיף 2.4, לרוב המקרים לא ידועים משפטי יציבות מתאימים. למרות זאת, כפי שנראה בהמשך, באופן מעשי קיבלנו ממינברס שחזורים יפים בדוגמאות רבות, ולכן אנו משערים שבעתיד יתגלו משפטי יציבות חדשים שיסבירו את התוצאות הטובות שמינברס נותן.

הדפסה של תוכנית מינברס מופיעה בנספח ג.

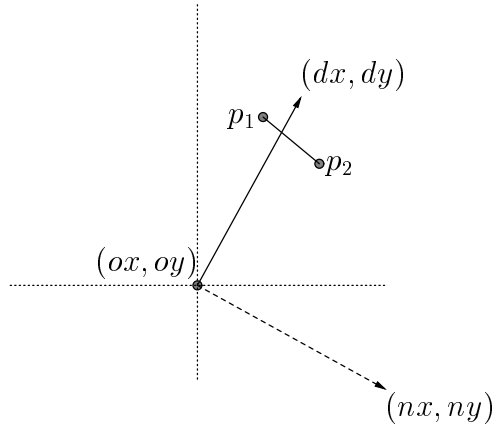
4.1 מצולעים כוכביים שכבתיים

באלגוריתם מינברס, אנו נטפל במצולעים כוכביים עם צפיפות אחידה. צפיפות אחידה זו נתונה מראש, וכך גם מספר הנקודות במצולע (נקבל קרוב טוב יותר לגוף ככל שנגדיל את מספר הנקודות במצולע). כל נקודה תהיה בזווית קבועה מראש ביחס למרכז המצולע (בתוכנית הנוכחית: הנקודות מסודרות במרחק זוויתי שווה זו מזו), ומרחקה מהמרכז הוא המשתנה שהאלגוריתם יצטרך למצוא. מרכז המצולע הכוכבי עצמו גם הוא ישתנה.

בחירת גופים מסוג זה (ולא גופים קמורים, או קשירים כלשהם למשל) מאפשרת הקטנת מספר המשתנים, ומקלה על המינימיזציה, מכיוון שאין צורך להזהר מחיתוך עצמי של קו השפה, או קבלת קו שפה לא חוקי (למשל, לא קמור).

גופים כוכביים הם גם כללים יותר מגופים קמורים. זהו יתרון מסוים של מינברס, אך גם חסרון במקרים מסוימים: אם ברצוננו לקבל שחזור של צורה קמורה כלשהי, ויש לה צורה שקולה שהיא כוכבית, אך אינה קמורה, מינברס עלול לקבלה ולא את הצורה הקמורה הרצויה. דוגמה לכך היא דוגמת הריבוע (ראה פרק 2.5), לו בהנתן זוג כיוונים יש צורה קמורה שקולה נוספת (מקבילית), אך גם צורה לא קמורה (אך כוכבית) שקולה נוספת, שדווקא אותה קיבלנו בריצות דוגמה. אם ברוצים לשחזר רק צורות קמורות, ניתן לתקן את אלגוריתם המינימיזציה ולהכריחו להעדיף צורות קמורות.

למעשה, בתוכנית מינברס לקחנו גופים כללים יותר ממצולעים כוכביים, להם קראנו מצולעים כוכביים שכבתיים: מצולעים שכבתיים בנויים ממספר מצולעים מלאים, לכל אחד מהם צפיפות אחידה קבועה משלו, שמונחים אחד על השני — כאשר בנקודה בה מונחים מספר פנימי מצולעים צפיפות הגוף היא סכום צפיפויות המצולעים המתאימים. לדוגמה, בצורה זו ניתן לתאר ריבוע בצפיפות 1 עם חור משולש קטן, על־ידי כך שלוקחים ריבוע בצפיפות $+1$, ועליו משולש קטן עם צפיפות -1. על־ידי מצולעים שכבתיים ניתן לתאר גם גופים לא קשירים. כאמור, יש צורך לקבוע מראש את מספר המצולעים, ואת צפיפויותיהם, ואז ניתן להתחיל במינימזיצית מינברס.



איור 4.1: בדיקת חיתוך ישר וקטע
Figure 4.1: Testing whether a line and segment intersect

4.2 האלגוריתם

4.2.1 סימולצית הטלה

כדי לבדוק האם גוף ניחוש כלשהו "קרוב" לגוף אותו אנו מחפשים, אנו זקוקים לאלגוריתם "סימולציה" להטלה, כלומר אלגוריתם שנותן, בהנתן גוף ניחוש וקבוצת קרניים את *תוצאת* קרניים אלו — כלומר את אינטגרל הצפיפות של הגוף שעברה כל קרן.

אבן הבנין הבסיסית באלגוריתם היא הרוטינה `xray`, שלוקחת מצולע שכבתי (שהוגדר בסעיף הקודם), ומבצעת חיתוך עם קרן נתונה (שהיא קו ישר), ומחזירה את ה"מסה" שעברה הקרן.

`xray` עובדת בצורה הבאה: ראשית נמצאים כל החיתוכים של הישר עם צלעות המצולעים שמגדירים את המצולע השכבתי. עשינו זאת על־ידי בדיקת חיתוך בין כל צלע לישר. אם מהירות האלגוריתם היא חשובה, ניתן לקחת אלגוריתם יעיל יותר של חיתוך מצולע עם ישר אחד (או ליעלו עוד יותר על־ידי תכנון אלגוריתם לחיתוך מצולע עם קבוצת ישרים באותו כיוון, למשל על־ידי סידור קטעי המצולעים לפי הטליהם על הישר הניצב לקבוצת הישרים).

נניח שהישר נתון על־ידי נקודה (ox, oy) וכיוון (וקטור יחידה) $d = (dx, dy)$ (ראה איור 4.1). נגדיר את קטע המצולע שאת חיתוכו עם הישר אנו מעוניינים לבדוק (ולמצוא) על־ידי שתי נקודות הקצה שלו, p_1 ו־p_2 (בסימון זה נניח שמקואורדינטות נקודות אלו כבר חוסרה הראשית (ox, oy)). נסמן ב־$n = (nx, ny)$ וקטור ניצב לכיוון הישר, למשל $(nx, ny) = (dy, -dx)$. הקטע נחתך עם הישר אם למכפלות הפנימיות (p_1, n), (p_2, n) סימנים הפוכים (או אחת מהן אפס — ואז הישר פוגע בקדקד).

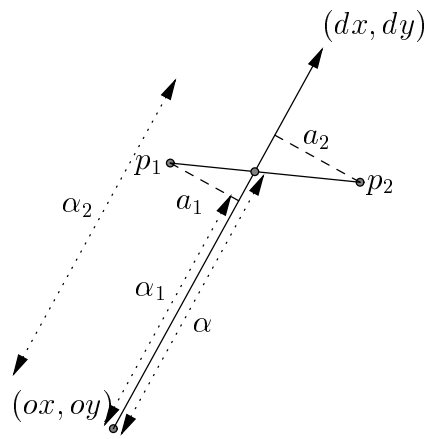
אם הישר חותך את הקטע, אנו מעוניינים לדעת היכן על הישר החיתוך קרה, כי אורך הקטע בין נקודת הכניסה ונקודת היציאה ממצולע קובע את אינטגרל הצפיפות שהקרן עברה. נגדיר פרמטריזציה לנקודת החיתוך של הישר עם הצלע: $(ox, oy) + \alpha(dx, dy)$. נרצה למצוא α זה — ראה איור 4.2.

הטלת הנקודה $o + p_i$ על הישר $o + \alpha d$ נותנת

$$\alpha_i = (p_i, d) \tag{4.1}$$

ה־α הרצוי הוא צרוף לינארי

$$\alpha = \alpha_1 \frac{|a_2|}{|a_1| + |a_2|} + \alpha_2 \left(1 - \frac{|a_2|}{|a_1| + |a_2|}\right) \tag{4.2}$$



איור 4.2: מציאת חיתוך ישר וקטע
Figure 4.2: Finding the intersection of a line and segment

כאשר a_1, a_2 הן ההטלות בציור, כלומר

$$a_i = (p_i, n) \tag{4.3}$$

מכיוון שאם יש חיתוך כבר בדקנו של־a_1, a_2 סימנים הפוכים, הרי שניתן לכתוב את הביטוי גם ללא ערכים מוחלטים:

$$\alpha = \alpha_1 \frac{a_2}{a_2 - a_1} + \alpha_2 \frac{-a_1}{a_2 - a_1} \tag{4.4}$$

בתוכנית מחושב α מתוך (4.4) תוך שימוש בהגדרות מ־(4.1), (4.3).

בצורה זו מוצאים את כל החיתוכים של הישר (הקרן הנתונה) עם המצולעים שמגדירים את המצולע השכבתי הנתון. אז מסדרים את החיתוכים לאורך הישר, בסדר α עולה, ובעזרת סידור זה מחשבים את אינטגרל הצפיפות שהקרן עוברת. הרעיון הוא כזה: כשהקרן נחתכת לראשונה עם מצולע, יודעים שבהמשכה היא במצולע ויש לקחת צפיפות מצולע זה בחשבון בקטע הבא. כשהקרן פוגעת בפעם השניה במצולע זה, יודעים שהקרן יצאה ממנו ויש להפסיק לקחת בחשבון את צפיפות מצולע זה עבור המשך הקרן. מובן שאלגוריתם זה מניח שהמצולע אינו חותך את עצמו, ואכן זה מובטח עבור גופים הכוכביים אותם אנו לוקחים. בקטעים שבהם הקרן בתוך שני מצולעים במצולע השכבתי, יש לקחת בחשבון את סכום הצפיפויות לאורך הקטע.

כדי להבהיר את אלגוריתם חישוב אינטגרל הצפיפות, נביא אותו כעת כפסאודו־קוד (pseudocode). הקוד הבא מניח שנתון מערך []intersections של חיתוכים מסודרים בסדר α עולה, וכאשר לכל חיתוך נתון alpha (α) ו־polygon (איזה מצולע הוא זה שנחתך בחיתוך זה). תוצאת קטע קוד זה הוא המספר mass, שהוא אינטגרל הצפיפות של המצולע השכבתי לאורך הישר הנתון.

```
do i=1..npolygons
    inpoly[i]=0
enddo

density=0
mass=0
```

```
do i=1..nintersections
    /** unless we are at the first point, add the contribution to the **/
    /** mass of the previous section of the line                      **/
    if i != 1 then
        mass = mass + density*(intersections[i].alpha-intersections[i-1].alpha)
    endif
    if not inpoly[intersections[i].polygon] then
        /** entering a polygon **/
        inpoly[intersections[i].polygon = TRUE
        density = density + polygon[intersections[i].polygon].density
    else
        /** exiting a polygon */
        inpoly[intersections[i].polygon = FALSE
        density = density - polygon[intersections[i].polygon].density
    endif
enddo
```

4.2.2 צורת ניחוש, ופונקצית הערכה לניחוש

כאמור, אלגוריתם השחזור מינברס הוא בעיקרו אלגוריתם מינימיזציה. הוא פועל על־ידי איטרציה על סדרת הפעולות: ניחוש צורה, הערכת הריחוק מהצורה המבוקשת (זו תקרא פונקצית הערכה) ושיפור הניחוש. שיטת שיפור הניחוש נוגעת לאלגוריתם המינימיזציה עצמו, ונדון בה בהמשך. בסעיף זה נדון בצורת ההצגה (במחשב) של צורת ניחוש, ובפונקצית ההערכה של הניחוש.

בסעיפים קודמים דובר כבר על כך שצורת הניחוש תהיה מצולע שכבתי, המורכב ממספר ידוע מראש של מצולעים שלהם צפיפות ידועה מראש (מספר המצולעים וצפיפותם לא נמצאים על־ידי האלגוריתם, ועליהם להיות ידועים מראש). דובר גם על כך שבאלגוריתם שמומש בעבודה זו הנחנו שכל מצולע הוא צורה כוכבית ביחס לראשית לא ידועה.

כדי להגדיר את נתוני המצולע השכבתי אותו המשתמש רוצה לשחזר מנתוני ההטלות, עליו להגדיר מראש מצולע שכבתי הנקרא *מצולע הבסיס*. למצולע שכבתי זה מוגדר מספר המצולעים, וצפיפותו של כל אחד מהם. כמו כן, לכל מצולע מוגדר מספר קדקדיו. כדי לקבוע את החלוקה הזוויתית של הקדקדים על הצורה הכוכבית, יש להציב בקדקדי מצולע הבסיס וקטורים, שיוכפלו מאוחר יותר ברדיוסים. רדיוסים אלו, בתוספת מיקום המרכז של כל מצולע, יהוו את הגדרת הניחוש. בדרך כלל נקח את מצולע הבסיס להיות נקודות על מעגל היחידה, בזוויות שוות.

פונקצית ההערכה (הרוטינה check_guess בתוכנית) צריכה כמובן לדעת מהי הבעיה אותה אנו פותרים, או במילים אחרות אלו תוצאות קרניים נתונות שמתוכם צריך להתבצע השחזור. נתונים אלו נמצאים בתוכנית במבנה נתונים הנקרא UserData, והוא מכיל את הנתונים הבאים:

1. מבנה נתונים מסוג RaySet, שמכיל קבוצה סופית של קרניים (כל קרן מוגדרת על־ידי כיוון ונקודה — כמתואר בסעיף הקודם), ולכל קרן *תוצאה* רצויה, כלומר אינטגרל הצפיפות שהצורה הדרושה אמורה לקבל עבור קרן זו.
2. מצולע שכבתי lp שהוא מצולע הבסיס (ראה הסבר לעיל)

3. כתובת התוצאה: מצביע על מצולע שכבתי בו נוכל לשים את התוצאה הסופית.

4. משתנים נוספים המשמשים באופן זמני בזמן החישוב: tryrs, neval, restarts, lastout, mind.

פונקצית הערכה היא בעיקרה סכום הריבועים של ההפרשים בין תוצאת כל קרן במצולע הניחוש והתוצאה הרצויה. אולם לעיתים מוסיפים "קנסות" לפונקצית ההערכה: הרעיון הוא שאלגוריתם המינימיזציה חופשי לטייל על כל מרחב הפרמטרים. אולם ישנם פרמטרים "לא־חוקיים" (לדוגמה, רדיוס שלילי) שבוודאי אינם הפתרון שאנו מחפשים, וישנם פרמטרים לא־רצויים שגם הם אינם קרובים לפתרון הרצוי. כשאנו מזהים ניחוש כזה, אנו מוסיפים לפונקצית ההערכה "קנס" שמגדיל אותה — ולכן אלגוריתם המינימיציה יתרחק מניחושים כאלו. בגרסה הנוכחית של מינברס אנו מוסיפים את הקנסות הבאים:

1. קנס על רדיוס שלילי: אם אחד הרדיוסים בוקטור הניחוש הוא שלילי, מוספים לפונקצית ההערה קנס כדי לגרום לאלגוריתם להתרחק מפתרון זה. כרגע הקנס הוא, באופן די שרירותי $5000|r|$.

2. קנס על אי־מעגליות: כאשר מתחילים את האלגוריתם עם ניחוש התחלתי הרחוק מאוד מהפתרון הנכון, נקודות המצולע צריכות לזוז הרבה עד למקומם הנכון. מכיוון שכל הזזה כזו לכיוון הנכון משפרת את המצב ומקטינה את פונקצית ההערכה, הנקודות נוטות לנוע בצורה לא מתואמת, וליצור צורה "קוצנית" ובעייתית. לכן אנו רוצים, בהתחלת התכנסות האלגוריתם, להוסיף קנס על "קוצניות", בצורת קנס על שונות בין רדיוסים שכנים. מובן שחייבים להפסיק קנס זה בסופו של דבר: הרי הצורה המבוקשת אינה דווקא מעגל.

   כרגע הקנס הוא (בצורה די שרירותית) $50\Sigma_i|r_i - r_{i-1}|$, ומפסיקים את הקנס אחרי חישוב פונקצית ההערכה 5000 פעם, או אחרי שגוף הניחוש כבר קרוב לגוף הרצוי (כרגע, כאשר פונקצית ההערכה ללא הקנס יורדת מתחת ל־20).

ניתן להוסיף קנסות במקרים נוספים: לדוגמה קנס על אי־קמירות כדי להכריח את מינברס להתכנס על הפתרון הקמור (במקרה שקיימים מספר פתרונות כוכביים, אך אחד קמור).

4.2.3 התחלת המינימיזציה

אלגוריתם המינימיזציה בו התמשנו בתוכנית מינברס הוא אלגוריתם ה"סימפלקס במורד" (Downhill Simplex Method) שיתואר בנספח 1.א. כפי שיוסבר שם, האלגוריתם הנ"ל דורש ניחוש התחלתי ו־N "פרטורבציות" עליו, כאשר N הוא מספר המשתנים במינימיזציה (יש לנו משתנה אחד לכל נקודה על המצולע ועוד שני משתנים לכל מרכז של מצולע).

כרגע האפשרויות להגדרת מצולע־שכבתי הניחוש הראשון במינברס הן די מוגבלות. ברירת המחדל היא להתחיל כשכל המצולעים מאותחלים למעגל ברדיוס 1 (אחד על השני), וניתן לשנות רדיוס התחלתי זה על־ידי האופציה `@guesscircle_rad` בקובץ הגדרת הבעיה (כרגע, אותו רדיוס לכל המצולעים במצולע השכבתי ההתחלתי). אפשר לשלוט על מרכזי המצולעים על־ידי האופציות `@guesscircle_cx` ו־`@guesscircle_cy`.

סוג הפרטורבציות המופעלות על הניחוש הראשון לקבלת N הניחושים האחרים נקבע על־ידי האופציה המספרית `@initvar`:

- 0: הפרטורבציה ה־n היא הגדלת המשתנה n בקבוע (כרגע תמיד 0.5). זו האפשרות הממולצת על־ידי [23], אך בריצות מינברס מעשיות היא גורמת בעיות הנובעות מכך שמצולעי הניחוש הם "קוצניים". כדי להקטין בעיות אלו נוסף הקנס על אי־מעגליות (artificial surface tension) שתואר בסעיף הקודם.

- 3: הפרטורבציה ה־n תלויה במהות משתנה n: משתנה שהוא קואורדינטה של מרכז מוגדל בקבוע (כרגע תמיד 0.5), ואילו בשביל פרטורבציה של רדיוס נקודה על המצולע, הרדיוס של הנקודה עצמה מוגדל בקבוע (כרגע תמיד 0.5) ורדיוס הנקודות האחרות מוגדל בגודל היורד לפי המרחק שלה אל הנקודה עליה מדובר.

  מבין האופציות שמנסות למנוע "קוצים" (כי הרי במצולעים המבוקשים אין קוצים), זו האופציה המוצדקת ביותר, ומומלץ להשתמש בה — אך מסיבות של תאימות־לאחור לריצות ישנות, ברירת המחדל היא כרגע 2, ולא 3.

- 1, 2: בפרטורבציה ה־n, המשתנה n מוגדל בקבוע (כרגע תמיד 0.5) והמשתנים הסמוכים מוגדלים בגודל היורד לפי המרחק שלהם אל המשתנה עליו מדובר. באופציה 1 מפסיקים להגדיל כשהמרחק גדול מ־5.

  הבעיה באופציות אלו שנוצר קשר חזק מדי בין הזזת נקודות הנמצאות בווקטור בסמיכות למשתני המרכז, להזזת המרכז עצמו (הרי לסמיכות בוקטור המשתנים אין משמעות אמיתית!), ואילו נעלם הקשר ההגיוני בין התחלת המצולע וסופו. כמו כן נוצר קשר מלאכותי בין נקודות מסוימות במצולע אחד ונקודות אחרות במצולע שני. כל אלו גורמים לעיתים לחוסר־סימטריה בניחושים הבסיסיים שמקשה על התכנסות האלגוריתם לפתרון המדויק. למרות זאת, אופציות אלו הניבו תוצאות טובות במרבית המקרים, ובעבר אופציה 2 היתה ברירת המחדל — ומסיבות של תאימות לאחור היא ברירת המחדל גם כעת.

4.2.4 התחלת המינימיזציה מחדש

מרבית שיטות המינימיזציה ב־n משתנים (ובכללם גם שיטת הסימפלקס במורד ("אמבה") בה השתמשנו במינברס) לא מבטיחות התכנסות למינימום גלובלי (אצלנו מינימום גלובלי יהיה כמובן 0, או קרוב אליו בגלל הדיסקרטיזציה). המינימיזציה עלולה להתקע, ואכן נתקעת בריצות אמיתיות, במינימום מקומי, או בסביבות ניחוש שעדיין אינו קרוב מספיק לפתרון הנכון. מתברר שכדאי מדי פעם לעצור את המינימיזציה, ולהתחילה מחדש כשהניחוש ההתחלתי הוא הניחוש הטוב ביוצר שמצאנו.

המשתמש קובע מתי להתחיל מחדש את המינימיזציה. ברירת המחדל היא להתחיל מחדש אחרי כל 5000 שימושים בפונקצית ההערכה, אך מספר זה ניתן לשינוי על־ידי האופציה `@ngiveup` בקובץ הגדרת הבעיה. אחרי כל עצירה, מספר השימושים בפונקצית ההערכה מוכפל בערך האופציה `@ngiveupmult`, שברירת המחדל שלו הוא 1.0. שימוש בערך גדול מ־1 מאפשר התחלה־מחדש תכופה בראשית הריצה, בה אנו עדיין רחוקים מהפתרון הנכון, אך כשמתקרבים לפתרון הנכון נותנים לאלגוריתם המינימיזציה יותר זמן לפעול בין עצירות.

אופציה נוספת של האלגוריתם הקשורה להתחלה־מחדש של המינימיזציה היא אופצית *המרכוז מחדש*. בעובדה שמרכז כל מצולע הוא משתנה מינימיזציה רגיל, יש בעייתיות מסוימת: כאשר קורה שהמצולע "די־קרוב" למצולע הנכון אך המרכז אותו בחרנו קרוב לשפת המצולע, שום דבר לא ישפר את מיקום המרכז: הזזת המרכז (יחד עם כל המצולע) רק תרע את הניחוש. לכן בריצה בה הניחוש ההתחלתי למצולע רחוק ממצולע הפתרון, והניחוש ההתחלתי למרכז המצולע רחוק ממרכז מצולע הפתרון (יתכן אפילו שניחוש המרכז *מחוץ* למצולע הפתרון), קורה לעיתים קרובות הדבר הבא: מרכז המצולע נע "לאט" מדי, עד שמקבלים מצולע ש"מרכזו" על שפתו, וכמובן חלק מהרדיוסים קרובים לאפס. זהו מצב מאוד בעייתי, מכיוון שרדיוסים שליליים הם אסורים (ראה הערה מקודם).

הפתרון שבחרנו הוא אופצית *מרכוז מחדש*, שמופעלת על־ידי השורה `@auto_recenter` בקובץ הגדרת הבעיה. כשמופעלת אופציה זו, בכל פעם שאלגוריתם המינימיזציה מותחל מחדש משנים את

המרכז להיות ממוצע נקודות המצולע ואת כל הרדיוסים בהתאם.

אגב, יש בעיה נוספת שקשורה לבחירה ההתחלתית של מרכז המצולע: אם למשל מצולע הניחוש ומצולע הפתרון *מופרדים* על־ידי קרן בכל אחד מהכיוונים (לדוגמה, אם הפתרון הוא מעגל יחידה שמרכזו ב־$(10,0)$ אך מצולע ההתחלה הוא מעגל יחידה שמרכזו בראשית) אז האלגוריתם לא יכול כלל להחליט (בעזרת הנורמה שלקחנו) לאיזה כיוון להזיז את מצולע הניחוש! לכן מומלץ לבחור בחוכמה את מרכז ורדיוס המצולעים ההתחלתיים. קל לבחור מרכז ורדיוס הגיוניים, מכיוון שמתוך ההטלות (מלפחות שני כיוונים) אפשר בקלות לדעת מצולע חוסם של הגוף, וכן את מרכז הכובד שלו, ואת שטחו.

הערה נוספת בקשר להתחלה מחדש: כאשר מתחילים מחדש את אלגוריתם המינימיזציה, צריכים להגדיר N פרטורבציות על הניחוש ההתחלתי, כפי שתואר בסעיף הקודם. בתוכנית הנוכחית, גודל הפרטורבציה בזמן התחלה מחדש נלקח גדול כפי שהיה בזמן ההתחלה (0.5). שיטה הגיונית יותר שאפשר יהיה לממש בעתיד היא להקטין את גודל הפרטורבציה בזמן התחלה־מחדש בשלבים מאוחרים יותר של ההתכנסות. הרי צפוי שהשינויים ממצולע הניחוש שאמורים להתבצע בשלבים מאוחרים הם קטנים הרבה יותר מאלו שנדרשו בהתחלה, בה נלקח מצולע ניחוש שהיה די שרירותי ורחוק מאוד מהנכון. מצד שני, כדי להבטיח שנוכל להחלץ ממצב של מינימום־מקומי, יתכן שדווקא השיטה הממומשת כעת הגיונית יותר.

4.3 דוגמאות

בסעיף זה נראה דוגמאות לשימוש בתוכנית מינברס, בהן ניקח גוף נתון (מצולע שכבתי), נמצא את הטלותיו בקבוצת קרניים מסויימת, ואז ננסה לשחזר גוף שמתאים להטלות אלו. נדגים איך מינברס משחזר באופן מעשי גם גופים שלא ידוע עבורם משפט יחידות שחזור, כולל גופים לא קמורים, גופים עם חור, וכו'.

כל הדוגמאות הורצו בעזרת התוכנית `main5` שהשימוש בה מתואר בנספח ב (והדפסתה מופיעה בנספח ג). לכל דוגמה נראה את קובץ הגדרת הבעיה עבור `main5`, ואת שמו בסט הדוגמאות (שאפשר לקבל לפי הוראות בנספח ב) בסוגריים מרובעים, בצורה `[.../problems]`. בצורה זו אפשר לחזור על התוצאות שנראה בסעיף זה.

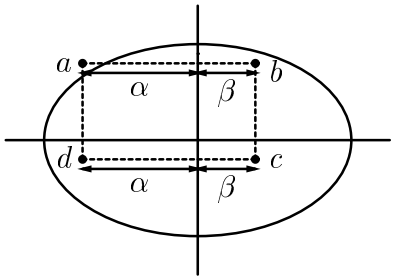
4.3.1 שחזור משני כיוונים

נתחיל בדוגמה `[problems/ball1.6]` בה אנו מצפים לשחזור יחיד: מעגל יחידה מוכלל עם חזקה 1.6 ($x^{1.6}+y^{1.6}=1$), וזוג כיווני הצירים. השחזור היחיד נובע מהמשפט הבא (שהופיע, בצורה קצת פחות כללית, ב־[3]):

משפט 32 *יהיה A גוף סימטרי יחסית לציר y (כלומר, $(x,y)\in A \iff (-x,y)\in A$), ונניח שחיתוכיו בניצב לציר y הם קטעים (זהו תנאי חלש יותר מקמירות). אזי A נקבע ביחידות מבין כל הקבוצות המדידות על־ידי הטלותיו בכיווני הצירים הניצבים x, y.*

הערה 2 *ניתן כמובן להכליל את המשפט למקרה שמרכזו של K מוזז מהראשית, ולמקרה שהגוף וצירי ההטלות מסובבים. אך זה היה מסבך ללא צורך את הסימונים שלנו ולכן הנחנו ללא הגבלת הכלליות שציר הסימטריה הוא ציר y.*

הוכחה: אפשר להוכיח משפט זה בעזרת מושג החסימה (ראה פרק 2.2.1), ובצורה מיידית בעזרת מושג האדיטיביות (ראה הוכחת משפט 17 בפרק 2.2.2 שממנה נובע גם משפט זה), אך ההוכחה של Chang and



איור 4.3: מלבן בגוף סימטרי יחסית לציר y

Figure 4.3: A rectangle inside a body that is symmetric relative to the y axis

Chow משתמשת במושג "רכיב־מיתוג" (ראה פרק 2.2.1): כדי להוכיח ש־A נקבע ביחידות עלינו להראות שלא יכול להיות ב־A רכיב מיתוג.

נניח בשלילה שיש רכיב מיתוג. רכיב מיתוג נוצר על־ידי הזזות של קבוצה שאינה בעלת מידה אפס (זהו חלק הכרחי בהגדרת רכיב מיתוג על קבוצות שהן רק מדידות, ולאו־דווקא קמורות), אך מתוך הקבוצה נוכל לבחור נקודה, ולקבל מלבן d,c,b,a שצלעותיו בכיווני הצירים, וכך שהנקודות a ו־c ב־A ואילו b ו־d מחוץ ל־A. (ראה איור 4.3 לגבי הסימונים).

הנחנו ש־a ב־A ואילו b בחוץ. אילו היו a ו־b מאותו צד של ציר ה־y, אז מובן שהם מצידו <u>ימני</u>, בגלל הסימטריות והעובדה שחיתוך ישר ניצב לציר y עם A הוא קטע. אז גם d ו־c מצדו הימני של ציר ה־y (כי הרי צלעות המלבן מקבילות לצירים), ושוב בכלל הסימטריות והחיתוך עם קטע נובע שחייב להתקיים ש־c ו־d שניהם בתוך A, שניהם בחוץ, או ש־d בפנים ו־c בחוץ, וזה בסתירה להנחתנו ש־c בפנים ו־d בחוץ.

אם דווקא a ו־b משני צידי ציר ה־y, נסמן ב־α את מרחקו של a (ו־d) מציר ה־y, וב־β את מרחקו של b (ו־c) מאותו ציר. הנחנו ש־a בתוך הגוף, ו־b בחוץ. אז מהנחת הסימטריה והחיתוך שהוא קטע, חייב להתקיים ש־$\alpha < \beta$. אך בצורה דומה, מהנתון ש־c בתוך הגוף ו־d בחוץ, נובע ש־$\alpha > \beta$, וזו סתירה.

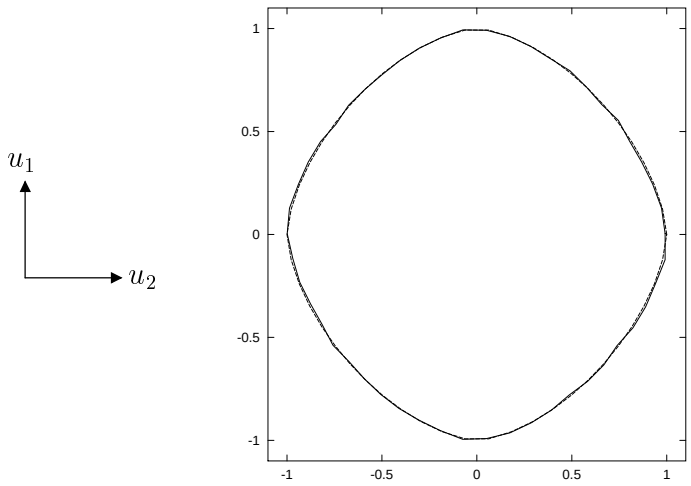
■

באיור 4.4 רואים איך מינברס אכן מצליח לשחזר את המעגל המוכלל מהטלותיו בכיווני הצירים.
כמו בכל הדוגמאות הבאות, נראה את קובץ הקלט `[problems/ball1.6]` של התוכנית main5:

```
@ngiveup 100000        # options start with an "@"
polygs/ball1.6         # the file defining the original layered polygon
1 50 1.0               # guess polygon: one layer, 50 points, density 1.0
300 0.0 1.0 -1.2 1.2   # rays: 300 in direction (0,1), from -1.2 to 1.2,
300 1.0 0.0 -1.2 1.2   #  300 more in direction (1,0) from -1.2 to 1.2.
```

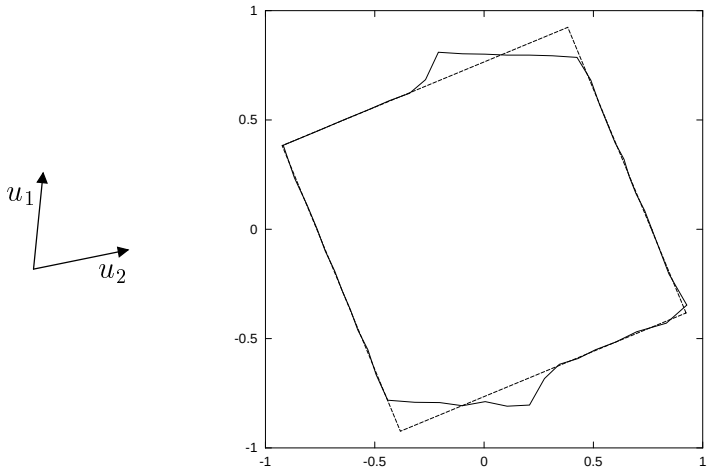
דוגמה של צורה שלא ניתנת לשחזור יחיד מהטלותיה בשני כיוונים היא הריבוע — ראה פרק 2.5. בפרק הנ"ל ראינו שישנן צורות נוספות מלבד הריבוע שנותנות אותן הטלות: מקבלית, וצורה לא־קמורה נוספת. אלגוריתם המינימיזציה עשוי למצוא כל אחד מצורות אלו, והצורה שימצא למעשה תלויה בניחוש ההתחלתי.

באיור 4.5 רואים תוצאה של שחזור מינברס, בו קיבלנו את הצורה הלא־קמורה המוכרת מפרק 2.5 (קו רציף) ולא את הריבוע (קו מקווקו) שלו אותן הטלות בכיוונים הנתונים u_2,u_1. הקו הרציף אכן קרוב יותר



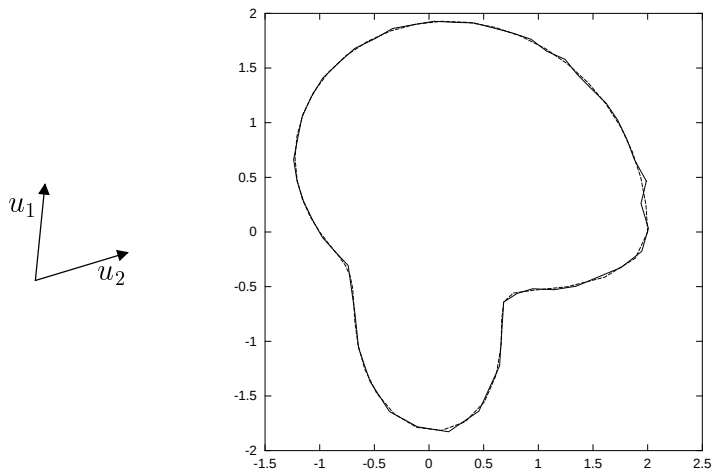
איור 4.4: שחזור מעגל מוכלל עם מינברס

Figure 4.4: Reconstruction of a generalized circle using Minverse



איור 4.5: שחזור ריבוע עם מינברס

Figure 4.5: Reconstruction of a square using Minverse



איור 4.6: שחזור ה"פטריה"
Figure 4.6: Reconstruction of the "mushroom"

לצורת הניחוש ההחלתית ששימשה אותנו במינימיזציה (מעגל ברדיוס 1) ולכן המינימיזציה נוטה להתכנס אליו, ולא אל הריבוע.

למעשה, כפי שהסברנו בפרק 2.5 מצאנו את הצורה מאיור 4.5 בעזרת מינברס לפני שידענו על קיומה באופן תאורטי, ורק אחר־כך ניסינו למצוא פרוש גאומטרי לצורה נוספת זו. זוהי דוגמה לשימושיות מינברס למחקר בנושא טומוגרפיה גאומטרית.

קובץ הקלט [problems/2dir-a] לדוגמת הריבוע:

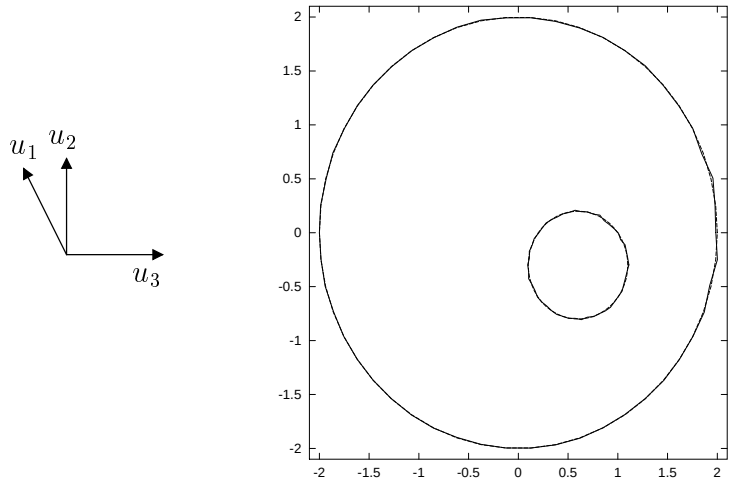
```
polygs/oct-square2
1 50 1.0
100 0.1 1.0 -1.5 1.5
100 1.0 0.2 -1.5 1.5
```

כפי שראינו, מעט ידוע על התאוריה של שחזור יחיד כאשר הצורה אותה אנו מנסים לשחזר היא פשוטת קשר (וכוכבית, כדי שנוכל לשחזרה עם מינברס), אך לא קמורה. מינברס יכול להוות כלי מחקר מעניין בתחום זה (לדוגמה, הצורה הנוספת שראינו שנותנת אותן הטלות כמו הריבוע היא כוכבית ואיננה קמורה), בנוסף להיותו כלי שחזור שימושי במקרים שאכן יש שחזור יחיד.

לדוגמה, איור 4.6 מראה שחזור של גוף לא־קמור, דמוי פטריה, מהטלותיו משני כיוונים (המסומנים באיור). למרות שלא ידוע משפט המבטיח את יחידות השחזור במקרה זה, רואים שאכן שוחזר הגוף הנכון.

[problems/mush1]:

```
polygs/mushroom-1
1 50 1.0
100 0.1 1.0 -2.3 1.5
100 1.0 0.3 -2.0 2.0
```



איור 4.7: שחזור עיגול עם חור
Figure 4.7: Reconstruction of a circle with a hole

4.3.2 שחזור צורות עם חורים

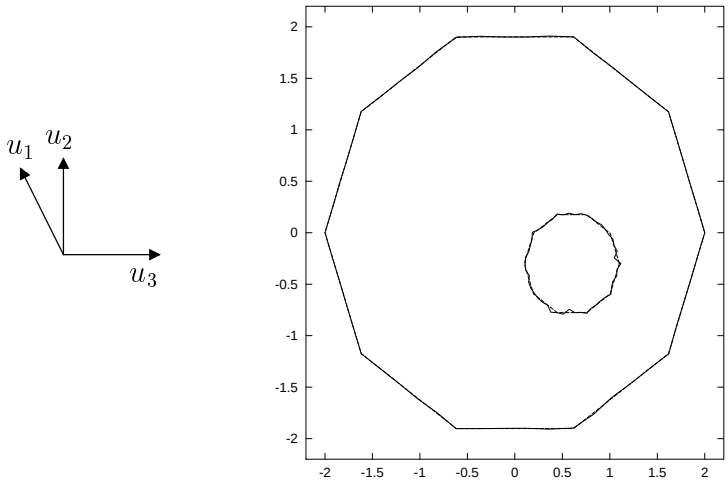
כאמור, הכללנו בתוכנית מינברס את המצולעים להיות *מצולעים שכבתיים*, שמאפשרים לנסות לשחזר גופים המוגדרים על־ידי "ערימת" מספר מצולעים, שמחלקים את המרחב למספר תחומים כשבכל תחום הצפיפות קבועה (וידועה מראש). לדוגמה, אם ברצונינו לשחזר גוף עם צפיפות 1 שבתוכו חור עם צפיפות 0, אז נגדיר את מצולע המטרה (הניחוש) להיות מצולע שכבתי, עם מצולע שצפיפותו 1 ומעליו מצולע שצפיפותו -1.

התאוריה בתחום זה — יחידות השחזור של קבוצות עם חורים (אפילו במקרה הפרטי של convex annuli, שהם גופים קמורים עם חור קמור) — היא ברובה לא־ידועה, ולכן מינברס יכול להוות כלי מחקר מעניין, בנוסף לכלי שימושי לשחזור. [1] עוסק בשחזור גופים קמורים עם חורים מעגליים, ומתוך משפט (שהם מוכיחים) שאומר שניתן לשחזר ביחידות n נקודות בעזרת הטלות מ־n כיוונים הם מראים שנובע שאפשר לשחזר גוף קמור עם n חורים מעגליים בעזרת n הטלות (בהנחה, הדרושה כדי לשחזר את הגוף הקמור עצמו, שהכיוונים אינם תת־קבוצה של כיווני צלעות מצולע משוכלל־אפינית).

דוגמה ראשונה היא שחזור גוף שבנוי מעיגול שמרכזו בראשית ורדיוסו 2, עם חור עגול שמרכזו ב־$(0.6, -0.3)$ ורדיוסו 0.5. איור 4.7 מראה איך מינברס שיחזר את הגוף מהטלותיו בשלושה כיוונים. הניחוש הראשוני היה ששני המצולעים הם מעגלים סביב הראשית ברדיוס 1. אגב, תוצאה דומה קיבלנו גם על־ידי שימוש בשני כיוונים בלבד (שני כיווני הצירים), ושלושת הכיוונים כאן הם רק לשם דוגמה.

[problems/cic3]:

```
@ngiveup 1000
@ngiveupmult 1.5
@guesscircle_rad 1.0
@auto_recenter
polygs/circle-in-circle
2 50 1.0 50 -1.0
```



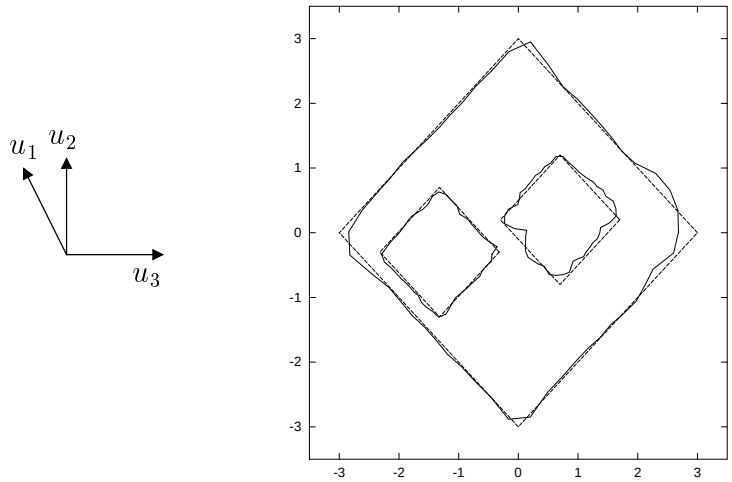
איור 4.8: שחזור מעושר עם חור מעושר
Figure 4.8: Reconstruction of a decagon with a decagonal hole

```
300 1 0 -3.0 3.0
300 0 1 -3.0 3.0
300 -0.5 1 -3.0 3.0
```

דוגמה נוספת היא שחזור מעושר משוכלל שבתוכו חור שגם הוא מעושר משוכלל: ראה איור 4.8. באיור רואים שהמצולע החיצוני שוחזר בצורה כמעט־מושלמת. המצולע הפנימי קרוב מאוד למעושר המדויק, אך בגלל מידותיו הקטנות והשפעתו הקטנה־יחסית על פונקצית ההערכה, הוא שוחזר בדיוק פחות מזה של המצולע החיצוני. [problems/ami]:

```
@ngiveup 1000
@ngiveupmult 1.5
@guesscircle_rad 1.0
@auto_recenter
@initvar 3
polygs/a-more-interesting
2 50 1.0 50 -1.0
300 1 0 -3.0 3.0
300 0 1 -3.0 3.0
300 -0.5 1 -3.0 3.0
```

דוגמה נוספת [problems/2did1] היא שחזור ריבוע מסובב עם שני חורים ריבועיים מסובבים בתוכו: ראה איור 4.9. בדוגמה זו התכנסות האלגוריתם היתה מאוד איטית, והטלות התוצאה באיור עדיין לא זהות מספיק להטלות הרצויות, אך גם כך רואים בברור שאלגוריתם התחיל להתכנס על הפתרון הרצוי.



איור 4.9: שחזור ריבוע עם שני חורים ריבועיים
Figure 4.9: Reconstruction of a square with two square holes

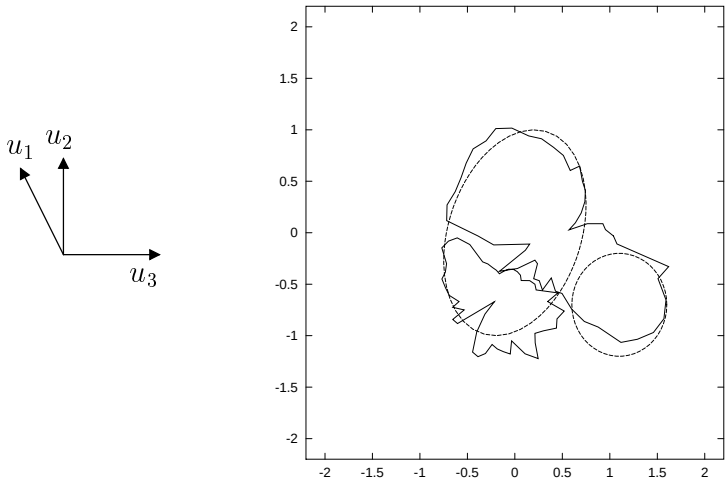
4.3.3 שחזור צורות לא קשירות

בסעיף 2.1 הסברנו מדוע ניתן לחשוב על שיטות הטומוגרפיה הגאומטרית כעל שיטות לשחזור חורים בגופים בעלי צפיפות קבועה: נניח שנתון גוף, תלת מימדי אטום, שידוע שצפיפותו קבועה מלבד חור שיש בו, ומבקשים לשחזר את החור. השיטות שלנו הן מישוריות ולכן נדבר על חתך מישורי אחד מתוך הגוף. את צורתו החיצונית של הגוף ניתן למדוד באמצעים ישירים, וניתן להחסיר את ההטלות שנמדדנו לגוף מההטלות "המצופות" שהיו מתקבלות לו צורתו החיצונית של הגוף היתה מלאה לחלוטין בחומר. תוצאת החיסור היא בדיוק ההטלות שהיו מתקבלות מגוף דמיוני, שהוא בדיוק צורת החור בגוף המקורי מלאה בחומר בצפיפות הגוף המקורי.

בראיה זו, שחזור צורה לא פשוטת־קשר, כפי שעשינו בסעיף הקודם, נראה לא־שימושי מכיוון שמשמעותו שבגוף יש חור ובחור צף גוש חומר — דבר בלתי הגיוני בעליל. שימושי יותר לנסות לשחזר צורה לא קשירה, למשל שני מרכיבי־קשירות שמיצגים שני חורים בגוף. כרגיל, יש לומר למינברס מראש את מספר המצולעים במצולע השכבתי (כל מצולע יהיה רכיב קשירות) ואת צפיפות כל מצולע (אם מדובר בחורים בגוף בעל צפיפות קבועה, יש כאמור לקחת את הצפיפות הנ"ל בתור הצפיפות של כל החורים).

כדוגמה לקחנו צורה לא קשירה שמורכבת מאליפסה ומעיגול (ראה קוים מקווקוים באיור 4.10). נסיון השחזור הראשון שלנו [problems/noncon2], באותו איור, לא הצליח (כלומר פונקצית ההערכה הצליחה לרדת, אך נעצרה ב־0.77) ורואים בו את הסכנה של מציאת מינימום מקומי על־ידי אלגוריתם המינימיזציה. רואים שאמנם מסת הצורות מחולקת בין שני המצולעים, אך זו לא החלוקה "הנכונה". זהו (או משהו קרוב) מינימום מקומי מכיוון שצורות קרובות יתנו הטלות טובות פחות. אך ברור שה"צוואר" של החומר בין האליפסה לעיגול לא יתן לפונקצית ההערכה לרדת עד 0, ונתקענו במינימום מקומי. התחלת המינימיזציה מחדש כפי שתוארה בסעיפים הקודמים לא הצליחה לעבור למינימום הגלובלי, שכנראה רחוק מדי ממינימום מקומי זה וקשה מדי לקפוץ בינהם.

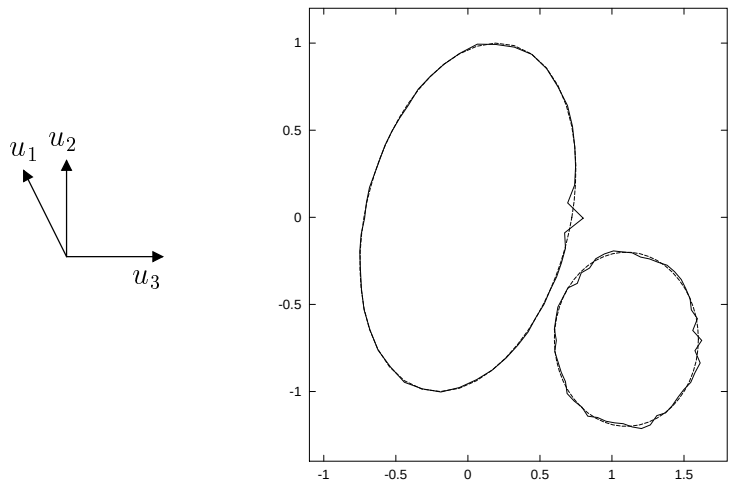
מינימום מקומי זה התקבל כשלקחנו את שני מצולעי הניחוש הראשוניים להיות מעגלים ברדיוס 0.5 בראשית. הבעיה העקרונית בזה היא שבדוגמה זו לשני המצולעים צפיפות זהה, ולכן אלגוריתם



איור 4.10: שחזור שלא התכנס של צורה לא קשירה
Figure 4.10: Unconverged reconstruction of a disconnected shape

המינימיזציה מתקשה "להחליט" על תפקיד שני המצולעים (אחד היה אמור "להתלבש" על העיגול, השני על האליפסה), ואחד המצולעים מנסה למלא את שניהם. לכן פתרנו את הבעיה על־ידי התחלת המינימיזציה עם שני עיגולים במרכז שונה: אחד בראשית ואחד עם מרכז ב־$(0,1)$. קל לקבוע ניחוש כזה (שכלל אינו צריך להיות מדויק) מההטלות: במקרה שלנו שתי הצורות נמצאות בתחום $-1 < x < 1.5$ ולכן בחרנו מרכזים סבירים. עכשיו, האלגוריתם התכנס בדיוק מצויין: ראה איור 4.11. [problems/noncon3]:

```
@ngiveup 1000
@ngiveupmult 1.5
@guesscircle_rad 0.5
@guesscircle_cx  0.0 1.0
@guesscircle_cy  0.0 0.0
@auto_recenter
@initvar 3
polygs/noncon
2 50 1.0 50 1.0
200 1 0 -2.5 2.5
200 0 1 -2.5 2.5
200 -0.5 1 -2.5 2.5
```



איור 4.11: שחזור מכונס של צורה לא קשירה

Figure 4.11: Converged reconstruction of a disconnected shape

פרק 5

סיכום ומסקנות

בעבודה זו חקרנו את התחום של *טומוגרפיה גאומטרית* ואת השאלות הקשורות לאפשרות שחזור חד־ערכי של גוף מהטלותיו במספר סופי של כיוונים. הבאנו סקירה ספרותית נרחבת של הנושא, והראנו את התוצאות הידועות בתחום, כמו למשל התנאים המספיקים והכרחיים של לורנץ לשחזור יחיד של קבוצה מדידה משתי הטלות בכיווני הצירים מבין משפחת הקבוצות המדידות, ומשפט גרדנר שמראה בפרט שבעזרת הטלות מארבעה כיוונים (תחת מגבלה מסויימת על בחירת הכיוונים) ניתן לשחזר ביחידות כל גוף קמור מבין משפחת הקבוצות הקמורות.

ראינו גם שיש בתחום זה מספר רב של בעיות פתוחות, ומקרים בהם לא ידועה תשובה תאורטית לשאלה האם השחזור במקרה מסוים הוא יחיד. לדוגמה, לא ידוע אפיון כל הקבוצות הקמורות שניתנות לשחזור יחיד מהטלותיהן בזוג כיוונים מסוים (למרות שידוע ש"רוב" הקבוצות הקמורות אכן ניתנות לשחזור יחיד כזה). לקבוצות כלליות יותר (כגון, קבוצות כוכביות, קשירות, וכדומה) ידוע מעט מאוד על שאלת היחידות. כמו כן, ראינו שידוע מעט מאוד על שאלת המוגדרות־היטב והיציבות של בעית השחזור, ולכן למרות שבמקרים מסוימים אנו יודעים באופן תאורטי שיש יחידות בשחזור, עדיין נשאלת השאלה האם השחזור יציב וניתן לביצוע באופן מעשי.

לכן מימשנו במחקר זה שני אלגוריתמי שחזור שאפשרו לנו לחקור את נושאי היחידות והיציבות. האלגוריתם הראשון היה האלגוריתם שהציע גרדנר לשחזור גופים קמורים מהטלותיהם בארבעה כיוונים. האלגוריתם השני היה אלגוריתם חדש שפיתחנו, בשם "מינברס". אלגוריתם מינברס נועד לשחזר גופים כלליים, "מצולעים שכבתיים", שכוללים צורות כוכביות, צורות עם חורים וצורות לא קשירות. אלגוריתם מינברס מקבל נתונים כלליים על ההטלות, ואינו מוגבל להטלות מארבעה כיוונים, למשל.

אלגוריתם גרדנר נתן שחזורים יפים (במקרה של הטלות מארבעה כיוונים), וזאת למרות שראינו שההצדקה התאורטית שלו איננה מלאה כרגע, ונשענת על השערה של גרדנר שהנקודות שמצאנו צפופות על שפת הגוף. את פעולת אלגוריתם מינברס הדגמנו במבחר דוגמאות: ראינו מקרים בהם ציפינו לשחזור יחיד ואכן קיבלנו את השחזור לו ציפינו, וראינו מקרים בהם השחזור אינו יחיד ואכן קיבלנו צורה נוספת בשחזור. כמו כן ראינו מקרים, עם גופים קמורים, גופים פשוטי־קשר אך לא קמורים, גופים עם חורים, וגופים לא קשירים, שבהם לא ידועים משפטי יחידות שחזור, אך למרות זאת אלגוריתם מינברס שחזר את הגוף אותו ציפינו לקבל.

לסיכום, בעבודה זו פיתחנו כלי שחזור חדש, מינברס, ומימשנו גם אלגוריתם ידוע של גרדנר. כלים אלו מראים שניתן לשחזר באופן מעשי קבוצות כוכביות (או הכללה שלהן) מהטלותיהן ממספר כיוונים קטן. תוצאות מעשיות אלו מצביעות על כך שהסיכוי לשחזור יחיד הוא אופטימי יותר מהנראה מהתוצאות התאורטיות הידועות כיום. אנו משערים שבעתיד יתגלו משפטים תאורטיים חדשים על שחזור יחיד

ויציבות בעית השחזור שיסבירו מדוע מינברס פועל כה טוב.

נספח א

שיטות מינימיזציה

שני האלגוריתמים לשחזור גופים מהטלותיהם שראינו בעבודה זו, האלגוריתם של גרדנר ואלגוריתם מינברס, משתמשים ב*אלגוריתם מינימיזציה רב־מימדי*: באלגוריתם גרדנר ידועה פונקציה של שלושה משתנים ורוצים למצוא שלשה שנותנת *מינימום גלובלי* של הפונקציה, ובאלגוריתם מינברס יש פונקציה של מספר רב של משתנים (כל נקודת הקף במצולע שמחפשים היא משתנה, ומרכז מצולע הוא שני משתנים נוספים), שגם לה מעונינים למצוא את ה־n־יה שנותנת מינימום גלובלי (מינימום זה צריך להיות 0, או קרוב לכך).

אלגוריתמים שונים למינימיזציה רב־מימדית ניתן למצוא ב־[23, פרק 10]. אחד המאפיינים של אלגוריתם מינימיזציה הוא השאלה האם הוא עושה שימוש בנגזרת (גרדיאנט) הפונקציה שלה מחפשים את המינימום. שימוש בנגזרת יכול להאיץ את התכנסות האלגוריתם, אך מכיוון שהוא דורש גם חישובי נגזרת, לא ברור מראש האם לבעית מינימיזציה מסוימת שימוש בנגזרת מועיל (אגב, מדובר כמובן בחישוב ישיר של הנגזרת, ולא חישוב נומרי כגבול מנות, כי אז היו נדרשות מספר רב של אבלואציות של הפונקציה). בנוסף, שימוש בנגזרת (שניתנת לחישוב בשתי הבעיות בהם אנו דנים — אלגוריתם גרדנר ואלגוריתם מינברס) מסבך לא־מעט את התוכנית. מסיבות אלו החלטנו להשתמש באלגוריתם מינימיזציה שלא דורש נגזרות של הפונקציה הנתונה.

1.א שיטת ה"סימפלקס במורד" הרב־מימדית

השיטה שנתאר בפרק זה (Downhill Simplex Method) תוארה לראשונה על־ידי Nelder ו־Mead [22] ונמצאת גם היא ב־[23].

שיטת הסימפלקס במורד דורשת חישובי הפונקציה בלבד (היא לא משתמשת בנגזרות). היא אינה יעילה במיוחד במספר חישובי־הפונקציה שהיא דורשת, אך היא מצוינת לקבלת תוכנית רובוסטית, ולא מניחה שום הנחות על הפונקציה.

שיטת הסימפלקס במורד קלה להסברה בצורה גאומטרית: סימפלקס הוא פאון N־מימדי עם $N+1$ קדקדים, וכל המקצועות המחברים בינהם. לדוגמה, סימפלקס דו־מימדי הוא משולש (לאו־דווקא שווה־צלעות). אנו מתענינים רק בסימפלקסים לא־מנוונים (עם נפח לא־אפס). אם אחד הקדקדים של סימפלקס לא־מנוון נבחר כראשית, שאר N הנקודות מגדירות וקטורים שפורשים את כל המרחב ה־N־מימדי.

במינימיזציה חד־מימדית אפשר לתחום את המינימום בקטע, ועל־ידי חלוקתו למצוא את המינימום. לצערנו אין שיטה כזו למינימיזציה רב־מימדית. אלגוריתם מינימיזציה רב־מימדי מקבל ניחוש התחלתי

(וקטור N־מימדי), ואמור להמשיך ממנו "במורד" הטופוגרפיה שמוגדרת על המרחב ה־N מימדי, עד שמגיע למינימום (מקומי, לפחות).

את שיטת הסימפלקס במורד מתחילים לא עם נקודה אחת, אלא עם $N+1$ נקודות, המגדירות סימפלקס התחלתי. אם נחשוב על אחת הנקודות (לא חשוב איזו) כעל נקודת ההתחלה P_0, נוכל לקחת את N הנקודות האחרות להיות

$$P_i = P_0 + \lambda e_i \tag{1.א}$$

כאשר e_i הם וקטורי היחידה הפורשים את המרחב ה־N מימדי, ו־λ הוא קבוע כלשהו (למשל, אורך אופיני לבעיה, אך אפשר כמובן לקחת גם קבוע שונה לכל כיוון).

שיטת הסימפלקס במורד מבצעת כעת סדרה של מהלכים. ברוב במהלכים מוצאים את הנקודה בסימפלקס בה ערך הפונקציה הוא הגדול ביותר ומזיזים אותה דרך הפאה מולה לנקודה נמוכה יותר. מהלך כזה נקרא *שיקוף* והוא שומר על נפח הסימפלקס. כשכדאי (נקודת השיקוף טובה יותר מהנקודה הטובה ביותר בסימפלקס), השיטה מגדילה את הסימפלקס באחד הכיוונים על־ידי לקיחת צעד גדול יותר. כשהיא מגיעה ל"רצפת עמק", כלומר נקודת השיקוף גרועה, מכווצים את הסימפלקס בכיוון זה. אם הסימפלקס מנסה להכנס לחור קטן, וגם כיווץ זה לא עזר (הנקודה החדשה גם היא גרועה) מכווצים את הסימפלקס בכל הכיוונים מסביב לכיוון הנקודה הטובה ביותר שלו.

השאלה מתי לעצור את האלגוריתם היא שאלה עדינה. ללא התיחום שיש באלגוריתמים חד־מימדיים, אי־אפשר לדעת מתי אנו קרובים מספיק למינימום. שיטה אחת בה משתמשים היא לעצור את האלגוריתם בצעד שבו הזזנו נקודה למרחק שקטן מאפסילון מסוים (באלגוריתם הנוכחי, זה שקול לעצירה כאשר הסימפלקס מספיק קטן). שיטה שניה היא לעצור כאשר הירידה של ערך הפונקציה בצעד האחרון היתה קטנה מאיזשהו אפסילון. השיטה בה מציעים [23] להשתמש היא לעצור את האלגוריתם כאשר השונות בערכי הפונקציה ב־$N+1$ קדקדי הסימפלקס היא מספיק קטנה.

בכל מקרה, קריטריוני העצירה הנ"ל עלולים להתבלבל בגלל צעד אחד לא טוב, או (במקרה שמחפשים מינימום גלובלי) כניסה לערוץ לא טוב. לכן במקרים רבים זהו רעיון טוב להתחיל מחדש את אלגוריתם המינימיזציה בנקודה בה הוא טוען שמצא מינימום. כדי להתחיל מחדש, יש כמובן להגדיר מחדש את $N+1$ קדקדי הסימפלקס, כאשר שוב נהוג לקחת אחד מהם להיות קדקד ההתחלה, והאחרים מוגדרים לפי (1.א).

מימוש האלגוריתם, כפי שנלקח מ־[23], נמצא ברוטינה `amoeba.c` בנספח ג.

נספח ב

שימוש בתוכניות השחזור

בפרקים הקודמים תארנו שני אלגוריתמים לשחזור צורות מהטלותיהן: האלגוריתם של גרדנר והאלגוריתם החדש שלנו, מינברס. הסברנו שם כיצד פועלים אלגוריתמים אלו, וכיצד מימשנו אותם, ובנספח זה נסביר כיצד להשתמש בתוכניות השחזור שכתבנו כדי לחזור על התוצאות המתוארות בחיבור זה, או כדי לנסות בעיות חדשות. הדפסה של תוכניות השחזור מופיעה בנספחים הבאים, אך בנספח זה נסביר איך לקחת את התוכניות ישירות מהאינטרנט, וכיצד להשתמש בה.

ב.1 דרישות קדם והתקנה

התוכניות שמתוארות בנספח זה נכתבו בשפת ANSI-C, ונבדקו על מספר מערכות Unix שונות (Linux, Solaris, Digital Unix). כדי להציג את התוצאות על המסך (או ליצר קובץ להדפסה), התוכניות משתמשות בתוכנת Gnuplot, תוכנת חינם שמותקנת על רוב מחשבי ה־Unix. הקומפילציה דורשת גם את התוכנית gcc. כדי לא לסבך את הקוראים בפרטים בין מחשבים שונים, ההוראות הבאות נכונות למחשב Leeor של הפקולטה למתמטיקה, אך צריכות לעבוד בצורה דומה גם בכל מחשב עם Linux. הן לא יעבדו על מחשב Techunix, שבו חלק מהתוכנות מותקנות בצורה לא טובה.

כדי להשתמש בתוכניות, ראשית יש להוריד מהאינטרנט את הקובץ

```
http://harel.org.il/nadav/msc/programs.tar.gz
```

ולפתוח את הקובץ בעזרת הפקודה

```
/usr/local/bin/tar zxvf programs.tar.gz
```

אז יש ללכת את הספריה (directory) programs שיצרה הפקודה האחרונה, ולבצע קומפילציה של התוכניות, בעזרת הפקודה

```
make
```

פעולה זו יוצרת בספריה הנוכחית את התוכניות: create-polygs, main-showpolyg, gardner, ו־main5.

ב.2 קבצי מצולעים שכבתיים

תוכניות שחזור בדרך כלל מקבלות נתונים על הטלות, ומנסות לשחזר מתוכן את הגוף. אנו רצינו לבדוק את אלגוריתמי השחזור שלנו, ולשם כך היינו צריכים דוגמאות של הטלות. אין צורך בהקרנת גופים אמיתיים

לשם כך: מספיק לדמות במחשב גופים (במקרה שלנו, מצולעים שכבתיים, כפי שהגדרנו בסעיף 4.1), לבצע להם סימולציה של הטלה בקרניים המבוקשות (כפי שתואר בסעיף 4.2.1), ולהשתמש בתוצאות אלו בתור הנתונים לאלגוריתם השחזור.

ובכן, אחד הנתונים החשובים לתוכניות שנראה בהמשך לבדיקת אלגוריתמי השחזור הוא מצולע שכבתי. מצולע שכבתי הוא קובץ שצורתו כמו בדוגמה הבאה:

`#LPolygon`	שורה שאומרת שזהו קובץ של מצולע שכבתי
`2`	מספר המצולעים (השכבות) במצולע השכבתי
`1`	צפיפות המצולע הראשון
`50`	מספר הנקודות במצולע הראשון
`1.1 2.3`	הנקודה הראשונה במצולע הראשון
`...`	
`1.2 2.4`	הנקודה האחרונה (50) במצולע הראשון
`-1`	צפיפות המצולע השני
`30`	מספר הנקודות במצולע השני
`0.1 0.3`	הנקודה הראשונה במצולע השני
`...`	
`0.2 0.4`	הנקודה האחרונה (30) במצולע השני

בשביל הדוגמאות, כתבנו תוכנית `create-polygs` שיוצרת כמה דוגמאות של מצולעים שכבתיים, ושמה אותם בספריה `polygs`. אם מולאו הוראות ההתקנה לעיל, אז אין צורך להריץ תוכנית זו: הספריה `polygs` מגיעה עם הדוגמאות כבר בתוכה. אולם התוכנית הנ"ל שימושית אם המשתמש רוצה ליצור מצולעים שכבתיים חדשים: ראה את קוד־המקור ב־`src/create-polygs.c`.

ניתן לראות את כל הדוגמאות הנמצאות בספריה `polygs` באיור 1.ב. באיור זה לא רואים את הצפיפות של כל מצולע, אולם בדרך כלל החיצוני הוא עם צפיפות 1, והפנימיים (אם יש) בצפיפות -1.

כדי להסתכל על קובץ מצולע שכבתי, יש להשתמש בתוכנית `showpolyg`. הפקודה

```
showpolyg polygs/ellipse
```

מראה את המצולע השכבתי `polygs/ellipse` על X Window System, ואילו

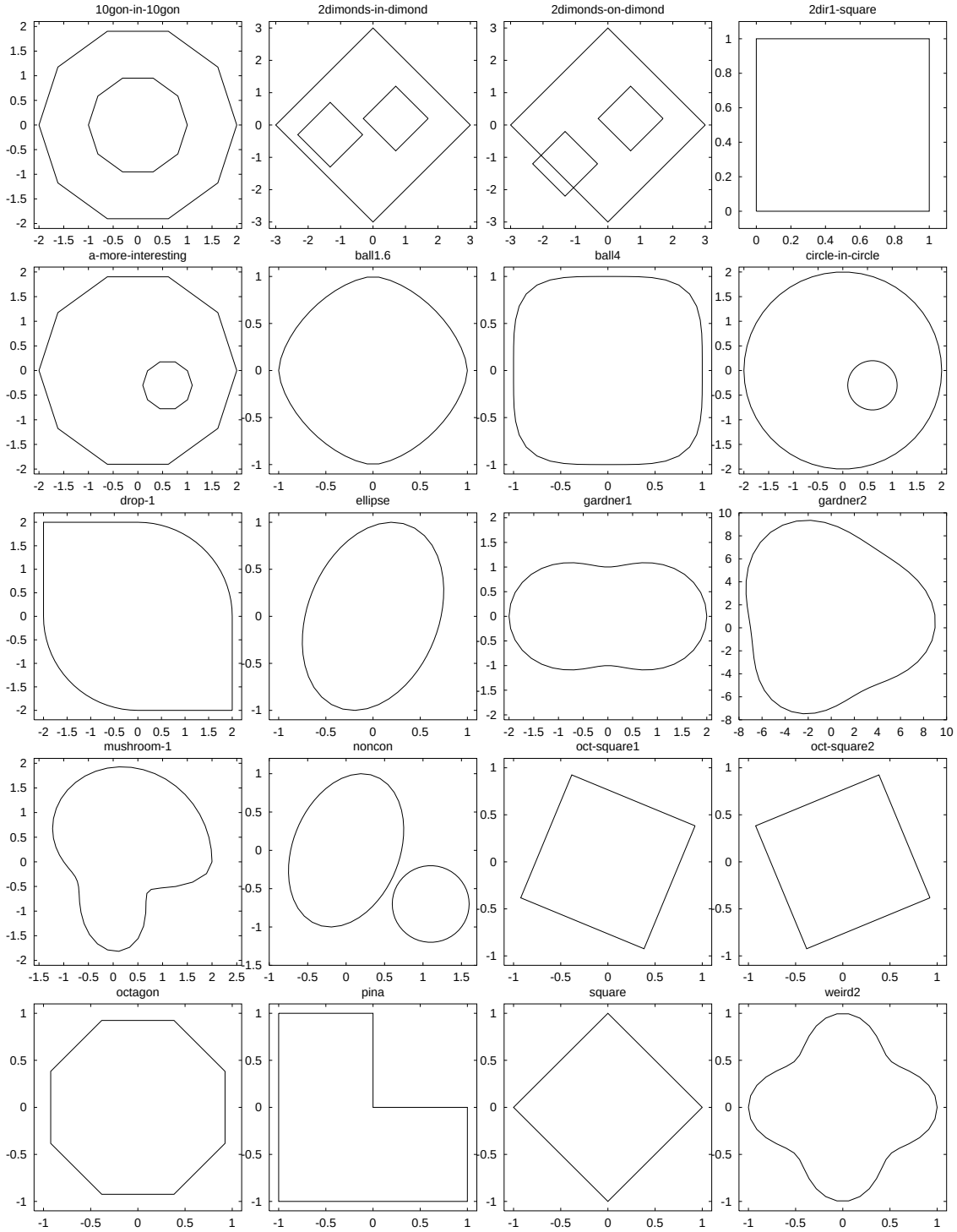
```
showpolyg -ps polygs/ellipse
```

יוצרת ממנו קובץ postscript בשם `ps.ps`.

3.ב שימוש בתוכנית השחזור לפי האלגוריתם של גרדנר

כפי שראינו בפרק 3, האלגוריתם של גרדנר מאפשר שחזור של גוף קמור (חלק וללא קטעים ישרים על שפתו) מהטלותיו בארבעה כיוונים. תוכנית הבדיקה `gardner` שכתבנו מתחילה בגוף דוגמה וארבעה כיוונים, מוצאת את ההטלות בכיוונים אלו, ואז משתמשת באלגוריתם גרדנר לנסות לשחזר את הגוף מהטלותיו.

התוכנית `gardner` מקבלת פרמטר אחד: שם של קובץ הגדרת הבעיה. לדוגמה, קובץ הגדרת הבעיה שנתן את התוצאה המתוארת באיור 3.3 (שאפשר למצוא ב־`problems/gball1.6`) נראה כך:



איור ב.1: דוגמאות של מצולעים שכבתיים בספריה polygs

Figure B.1: Examples of layered polygons in the “polygs” directory

אפסילון (שורה אופציונלית. ברירת המחדל: 5e-2)	`@epsilon 5e-2`
קובץ ממנו לקחת את גוף הדוגמה (מצולע שכבתי)	`polygs/ball1.6`
צורת מצולע ניחוש — משמש במינברס (ראה המשך) אך לא כאן	`0`
כל שורה מציינת את אחד מארבעת הכיוונים	`1000 0.2 1.0 -3.0 3.0`
המספר 1000 מציין 1000 קרניים בכיוון מסוים	`1000 1.0 0.2 -3.0 3.0`
$(-0.5, 1)$ הוא וקטור שמציין כיוון (לאו־דווקא מנורמל)	`1000 -0.5 1 -3.0 3.0`
$(-3.0, 3.0)$ הוא התחום (במרחב הניצב) בו הקרניים במרחקים שווים	`1000 1 0.7 -3.0 3.0`

כמה הערות לגבי קובץ הגדרת הבעיה:

- בגלל סיבות שהוזכרו בסעיף 3.4, כדאי לקחת מספר רב של קרניים בכל כיוון: בדוגמה זו לקחנו 1000, אך לפעמים (כשאפסילון קטן) נאלצנו לקחת עד 3000.
- משמעות האפסילון והצורך בו הוסברו בסעיף 3.3. ברירת המחדל היא 5e-2, ואם רוצים להשתמש בה אז אפשר לוותר על השורה `@epsilon`.
- בדוגמה הנ״ל $(-3.0, 3.0)$ מציין את הפס בו מתבצעות ההטלות בכיוון מסוים: מסתכלים על הפס שמוגדר על־ידי שני ישרים מקבילים לכיוון הנתון, אחד במרחק 3 והשני במרחק -3 מהראשית. בפס זה לוקחים 1000 קרניים (עם מרחק שווה בין הקרניים). חשוב לדאוג שאכן הפס שהוגדר מכסה את הגוף הנתון.

אחרי הרצת התוכנית `gardner`, למשל `gardner problems/gball1.6`, נוצרים קבצי תוצאה בספריה (directory) `out`. התוכנית `gpg` משמשת לצפיה בתוצאות ולהדפסתם. `gpg` ללא פרמטרים מראה את תוצאת ריצת `gardner` האחרונה על X Window System, ועם הפרמטר `-ps` הוא יוצר קובץ postscript בשם `ps.ps`. `gpg` מראה בדרך כלל את הנקודות שהאלגוריתם שחזר, על המצולע המקורי שהתחלנו איתו. אם לא רוצים לראות את המצולע המקורי, יש להשתמש בפרמטר `-noreal`. בספריה `out` נכתב גם המשולש שהאלגוריתם מוצא בתחילתו (קובץ `best-tri`), אך כרגע `gpg` לא מראה אותו.

תוכנית הצגת תוצאות נוספת, `view5` מראה את הגדרת הבעיה: היא מציירת את המצולע הנתון מראש ואת הקרניים שחושבו דרכו. גם לתוכנית זו יש פרמטר `-ps` ליצר קובץ postscript בשם `ps.ps`.

ב.4 שימוש בתוכנית השחזור מינברס

בפרק 4 הצגנו את אלגוריתם מינברס לשחזור כללי: שחזור מצולע שכבתי מהטלות (מספר סופי של תוצאות קרניים) נתונות. תוכנית הבדיקה `main5` שכתבנו מתחילה במצולע שכבתי ידוע ורשימת כיוונים להטלות, מוצאת את ההטלות בכיוונים אלו, ואז משתמשת באלגוריתם מינברס לנסות לשחזר את הגוף מהטלותיו. הגרסה הנוכחית של התוכנית בדרך כלל לא עוצרת, ויש לעצור אותה ידנית (על־ידי interrupt) כשהמשתמש מרוצה מהתכנסותה. אפשר להסתכל מדי פעם על תוצאות הביניים של השחזור גם במהלך הריצה, כפי שנסביר מיד.

התוכנית `main5` מקבלת פרמטר אחד: שם של קובץ הגדרת הבעיה. קובץ הגדרת הבעיה נראה כמו בדוגמה הבאה:

אופציות שנתאר מיד	`@...`
קובץ ממנו לקחת את גוף הדוגמה (מצולע שכבתי)	`polygs/circle-in-circle`
שתי שכבות: מצולע בצפיפות 1.0 עם 50 נקודות ושני בצפיפות -1.0	`2 50 1.0 50 -1.0`
כל שורה מציינת כיוון. 300 מציין 300 קרניים בכיוון מסוים	`300 1 0 -3.0 3.0`
$(0,1)$ הוא וקטור שמציין כיוון (לאו־דווקא מנורמל)	`300 0 1 -3.0 3.0`
$(-3.0,3.0)$: התחום במרחב הניצב בו הקרניים במרחקים שווים	`300 -0.5 1 -3.0 3.0`

כמה הערות לגבי קובץ הגדרת הבעיה:

- בראשית הקובץ יכולות לבוא מספר כלשהו של שורות שמתחילות בתו '@', ושמציינות אופציות. הסבר האופציות השונות מופיע בפרק 4 אך נביא פה שוב רשימה שלהן בקצרה:
  - `@ngiveup <int>`: מציין את מספר אבלואציות של פונקצית ההערכה, שאחריהם מתחילים את אלגוריתם המינימיזציה מחדש (ברירת המחדל: 5000)
  - `@ngiveupmult <double>`: בכל פעם שהמינימיזציה מתחילה מחדש, ערך `@ngiveup` הקודם מוכפל בערך `@ngiveupmult` (ברירת המחדל: 1.0)
  - `@guesscircle_rad <double>`: מצולע הניחוש מתחיל כמעגל ברדיוס זה (ברירת המחדל: 1.0)
  - `@guesscircle_cx <double>...`: רשימת קואורדינטות x של הניחוש ההתחלתי למרכזי מצולעי הניחוש: הראשון מתאים למצולע הראשון במצולע השכבתי אותו אנו מחפשים, וכן הלאה. (ברירת המחדל: כל המרכזים הם בראשית)
  - `@guesscircle_cy <double>...`: כנ"ל, קואורדינטות y
  - `@auto_recenter`: אם מופיעה אופציה זו, מתבצע "מרכוז מחדש" בכל פעם שהמינימיזציה מותחלת מחדש.
  - `@initvar <int>`: בוחר את שיטת הפרטורבציה על הניחוש ההתחלתי. ראה פרוט בפרק 4 (ברירת המחדל: 2)
- בדוגמה הנ"ל $(-3.0,3.0)$ מציין את הפס בו מתבצעות ההטלות בכיוון מסוים: מסתכלים על הפס שמוגדר על־ידי שני ישרים מקבילים לכיוון הנתון, אחד במרחק 3 והשני במרחק -3 מהראשית. בפס זה לוקחים 300 קרניים (עם מרחק שווה בין הקרניים). חשוב לדאוג שאכן הפס שהוגדר מכסה את הגוף הנתון.

בזמן הרצת תוכנית `main5`, כמו למשל `main5 problems/ball1.6` (שנותן את התוצאה באיור 4.4), התוכנית כותבת את פונקצית ההערכה (שאמורה להתכנס לאפס) אחרי כל 100 אבלואציות של הפונקציה, וכל 1500 אבלואציות נוצרים קבצי תוצאות־ביניים בספריה (directory) `out`. תוצאות הביניים מכילות מידע על צורת מצולע הניחוש (הטוב ביותר בסימפלקס) בזמן זה, ועל פונקציות ההטלות של מצולע זה.

התוכנית `gp5` משמשת לצפיה במצולעי הניחוש על X Window System או להדפסתם, וכאמור אפשר להשתמש בה גם בזמן ש־`main5` עדיין רצה. `gp5 -r` מראה את הניחושים בסדר הפוך: ראשון רואים את תוצאת־הביניים האחרונה שנשמרה, ובלחיצת Enter עוברים לתוצאת־הביניים הקודמת. `gp5` ללא `-r` מראה ראשון את תוצאת הביניים הראשונה (שהיא הניחוש ההתחלתי). `gp5 OUT11` מראה את תוצאת ביניים מספר 11 בלבד. הפרמטר `-ps` (למשל, `gp -ps OUT11`) משמש כרגיל לשמירת קובץ postscript בשם `ps.ps`. הפרמטר `-anim` עושה אנימציה (ללא לחכות ל־Enter) והפרמטר `-animf` אנימציה מהירה יותר. הפרמטר `-rays` מצייר גם את הקרניים המבוקשות, והפרמטר `-noreal` מבטלת את הצגת המצולע המקורי בו התחלנו.

כדי להציג את פונקציות ההטלה בתוצאות־הביניים לפונקציות ההטלה המבוקשות, יש להשתמש בתוכנית viewXX, בצורה הבאה: viewXX XX3 כאשר 3 הוא מספר תוצאת הביניים. גם ל־viewXX יש פרמטרים -ps, -anim, ו־-animf (כדי לצייר כמה תוצאות ביניים יש להשתמש בפקודה כמו viewXX out/XX*).

תוכנית הצגת תוצאות נוספת, view5, מראה את הגדרת הבעיה: היא מציירת את המצולע הנתון מראש ואת הקרניים שחושבו דרכו. גם לתוכנית זו יש פרמטר -ps ליצר קובץ postscript בשם ps.ps.

נספח ג

התוכנית מינברס

תוכניות אלו, שתוארו בפרק 4, נכתבו בשפת $ANSI - C$ והורצו על מערכת Unix (Linux על מחשב PC). התוכנית הראשית (המבצעת בעיות דוגמה) היא main5.c.

minverse.c

```
#include <stdio.h>
#include <math.h>
#include "layered_polygon.h"
#include "xray.h"
#include "nrutil.h"
#include "stats.h"
#ifdef CONVEXPENALTY
#include "convex.h"
#endif

#include "minverse.h"


#define TENSION 50.0 /* AST (artificial surface tension) 50.0 */
#if 0
#define QUITENEAR 10.0
#endif

/* the check_guess routine accepts two parameters: the guess (a vector of
   doubles) and the UserData - a structure containing the information of
   HOW to check the guess (i.e. what should the rays be) and some more
   fields for statistics
*/
typedef struct {
    RaySet *rs; /* pointer to an *evaluated* rayset (with evaluated results)
                */
    LPolygon lp; /* a base polygon, i.e. a polygon with the same structure
                    as the required polygon solution (number of polygons,
                    points in each one) with all points being the vectors
                    the coeeficients in the minimized vector multiply.
                    Currently this polygon is set to a equal angle rendition
                    of the unit circle
                  */
    RaySet tryrs; /* a rayset with same rays as rs, but the check_guess may
                    evaluate into it */
    LPolygon *resultlp; /* where check_guess may put it's guess polygon. it's
                           usually the lp given to minverse. NOTE IT IS A
                           POINTER!!! */

    /* the following variables are used for statistics and outputs */
    int neval;
```

```
    int restarts;
    double lastout; /* value of neval when last output a picture */
    double mind; /* minimum delta (value of function) encountered */
} UserData;


static void guess_recenter(double *d, LPolygon lp);
static void output_minverse_result(double *guess, UserData *ud, double d);

static double
check_guess(double *guess, UserData *ud)
{
    int k,i,j;
    double d;
    double penalty=0;
    /* realpenalty variable - for theoretically plausable penalties (i.e.,
       ones for which the real solution will have penalty 0 - like the
       unconvexity penalty, and unlike the AST penalty.
       Typically, such penalties will be small, since they should not
       effect the beginning of the convergence to the solution, but at the
       end they should move us to the correct one.
       */
    double realpenalty=0;
    /* unpack guess into result polygon */
    k=1;
    for(i=0;i<ud->resultlp->npolygons;i++){
        double centerx,centery,r;
        double oldr,r0; /* for artificial surface tension */
        centerx=guess[k++]; /* center of radial polygon */
        centery=guess[k++];
        /* AST (artificial surface tension) is still experimental. It also
           may not be good, since it tries to make circles out of shapes which
           are NOT. The TENSION should be chosen somehow to prevent that.
           perhaps a second derivate surface tension (SDATS) will be better -
           we wants the sum of numerical second deritives ("sudden direction
           changes") to be minimal. */
        r0=oldr=guess[k]; /* for artificial surface tension */
        for(j=0;j<ud->resultlp->polygons[i].npoints;j++){
            r=guess[k++];
#define ABS(x) ((x)<0 ? -(x) : (x))
            penalty+=ABS(r-oldr)*TENSION; /* for artificial surface tension */

            if(r<1e-6){
                /* We can't allow a negative radius, so we give a penalty for
                   one. The penalty is currenty arbitrary but should be changed
                   to something continuous. If continuous, how do we choose
                   the constant? It should depend on typical r scale. Perhaps
                   the non-continuous is better, but then we should use a
                   multiplying penalty, not an adding penalty. then we don't
                   need to stop the penalty - when the real function value is
                   0, multiplying it by a penalty doesn't hurt!
                */
                penalty+=5000*ABS(r);
                fprintf(stderr,"penalty for negative radius:  %g\n",5000*ABS(r));
                statPenaltyNeg++;
                r=1e-5;
            }
            /* ud.lp contains radius vectors (set in minverse(), to
               equal angles
             */
            ud->resultlp->polygons[i].points[j].x=
                centerx+r * ud->lp.polygons[i].points[j].x;
            ud->resultlp->polygons[i].points[j].y=
                centery+r * ud->lp.polygons[i].points[j].y;
            oldr=r; /* for artificial surface tension */
        }
        penalty+=ABS(r-r0)*TENSION; /* artificial surface tension */
```

```
#undef CONVEXPENALTY
#ifdef CONVEXPENALTY
        realpenalty+=0*(ud→neval/10000.0)*convexPenalty(ud→resultlp→polygons[i], centerx, centery);
#endif
    }

    /* now that we have the guess polygon in *(ud.resultlp), we evaluate the
       rayset
     */
    xrayset(*(ud→resultlp),ud→tryrs);
    /* now calculate the distance between this rayset results and the given
       ones in rs
    */
    d=0;
    for(i=0;i<ud→rs→nrays;i++){
        double e=ud→rs→rays[i].result-ud→tryrs.rays[i].result;
#undef INFNORM
#ifdef INFNORM
#define QUITENEAR 0.1
        if(ABS(e)>d) d=ABS(e);
#else
        d+=e*e;
#endif
    }

    /* We ouput a picture if two conditions are true: the current d is
       lower then the minimum d got before, and at least some number (1500)
       of evaluations have been done since the previous output.
       This ensures two things:
       1. outputs are not too close to one another
       2. each output is not random, but shows a result which is guaranteed
          to be the best result got until now.
    */
    if(ud→neval > ud→lastout+1500 && d < ud→mind){
        output_minverse_result(guess,ud,d);
        ud→lastout = ud→neval;
    }
    if(d<ud→mind) ud→mind=d;
    /* add the penalty, unless quite near or the penalty is big */
#ifndef QUITENEAR
#define QUITENEAR 20.0
#endif
#define BIGPENALTY 500.0
#define ENDOFPENALTY 5e3 /* stop the penalty after some time. */
#if 1
    if((d≥QUITENEAR && (ud→neval < ENDOFPENALTY)) || penalty > BIGPENALTY)
#else
    if(!ud→restarts || penalty > BIGPENALTY) /*** TRYING - penalties until restarts **/
#endif
        d+=penalty;
    else {
        penalty=0.0;
        d+=penalty;
    }
    d+=realpenalty;
    if((ud→neval%100)==0) fprintf(stderr,"%d:  %g (of which penalty:  %g,
realpenalty=%g)\n",ud→neval,d,penalty,realpenalty);

    ud→neval++;
    statEval++;
    return d;
}

/* miverse takes a complete rayset (i.e. the result field should be set for
   each ray), and tries to find a layered radial polygon which best fits
   this data. This polygon is put in the given lp variable (which should be
   preallocated with the number of polygons and points you want), and
   currently the density (In a future version, we can try adding the densities
```

```
    to the unknown vector too, and see what that gives us).
*/

#define CONST_PI 3.14159265358979323846
void
minverse(RaySet dataRays, LPolygon lp,
          const struct minverse_options *opts)
{
    double **p,*y,a;
    int ndim,i,j,k,m,count,ngiveup;
    UserData ud;

    ud.resultlp=&lp;
    ud.rs=&dataRays;
    ud.neval=0;
    ud.restarts=0;
    ud.lastout=-1e6;
    ud.mind=1e77;
    /* copy the rayset to a place where check_guess may evaluate */
    NewRaySet(ud.tryrs,dataRays.nrays);
    for(i=0;i<dataRays.nrays;i++){
        ud.tryrs.rays[i].ox= dataRays.rays[i].ox;
        ud.tryrs.rays[i].oy= dataRays.rays[i].oy;
        ud.tryrs.rays[i].dx= dataRays.rays[i].dx;
        ud.tryrs.rays[i].dy= dataRays.rays[i].dy;
    }

    ngiveup=opts->ngiveup;

    /* The minimization problem has the following parameters: they are the
       centers of polygons in lp (2 variables for each polygon), and one
       radius for each point in lp. we must count them (and put the number
       in ndim), then allocate and set the matrices needed by ameoba by
       some initial guess for a polygon.
       We also copy the size of lp to the userdata.
    */
    NewLPolygon(ud.lp, lp.npolygons);
    ndim=0;
    for(i=0;i<lp.npolygons;i++){
        NewPolygon(ud.lp.polygons[i],lp.polygons[i].npoints,lp.polygons[i].density);
        ndim+= lp.polygons[i].npoints + 2;
        for(j=0;j<lp.polygons[i].npoints;j++){
            double tet=2*CONST_PI/(lp.polygons[i].npoints)*j;
            ud.lp.polygons[i].points[j].x=cos(tet);
            ud.lp.polygons[i].points[j].y=sin(tet);
        }
    }
    p=dmatrix(1,ndim+1,1,ndim);
    y=dvector(1,ndim+1);

    /* pack initial data into vector p[1]. Later this initial data might be
       given to minverse (this stupid inital data make all polygons the same
       circle - although the radius and centers may be given as parameters.
    */
    k=1;
    for(i=0;i<lp.npolygons;i++){
        /* initial center */
        p[1][k++]=opts->guesscircle_cx[i];
        p[1][k++]=opts->guesscircle_cy[i];
        for(j=0;j<lp.polygons[i].npoints;j++){
            /* initial radii */
            p[1][k++]=opts->guesscircle_rad;
        }
    }
    /* use 'goto RESTART' if you want to restart using a different initial
       polygon. Before doing the goto, however, remember to set p[1] (as
       above) to the polygon you want (or leave it as is to restart the
       minimization from the current guess
```

```
    */
  RESTART:

    /* optional: find a new center when restarting minimization */
    if(opts→auto_recenter){
        /* output the new polygon before recentering (useful for debugging)
         */
        output_minverse_result(p[1],&ud,check_guess(p[1],&ud));

        guess_recenter(p[1], ud.lp);
        /* output the new polygon after recentering */
        output_minverse_result(p[1],&ud,check_guess(p[1],&ud));
    }

    /* We have an initial guess, and we need a whole "simplex" of guesses,
       so we make perturbations on the initial guess, in one of several
       optional ways:
    */
    switch(opts→initvar){
    case 0:
        /* make the rest of the initial matrix, as suggested in page 306 of
           NRC lambda is taken as 0.5 (this should be an option or something!)
           This tends to create large "spikes" in the guess polygons, and
           measures such as the AST (artificial surface tension) should be
           taken agains this problem.
        */
        for(i=2;i≤ndim+1;i++){
            for(j=1;j≤ndim;j++){
                if(j==(i-1)) p[i][j]=p[1][j]+0.5;
                else p[i][j]=p[1][j];
            }
        }
        break;
    case 1:
        /* an alternative to the previous kotiz */
        for(i=2;i≤ndim+1;i++){
            for(j=1;j≤ndim;j++){
                int dist=ABS(j-(i-1));
                if(dist≤5) p[i][j]=p[1][j]+0.5/(1+dist);
                else p[i][j]=p[1][j];
            }
        }
        break;
    case 2:
        /* This was the only option activated in older versions, so it
           remains the default for backward-comatibility reasons. While
           the justification for such a method of perturbation is dubious
           at best, it actually produced very good results in many tests
           (while appearantly causing nonoptimal results in others).
        */
        for(i=2;i≤ndim+1;i++){
            for(j=1;j≤ndim;j++){
                int dist=ABS(j-(i-1));
#define FA1 0.5
                p[i][j]=p[1][j]+FA1/(1+dist);
            }
        }
    case 3:
        /* This option also tries to circumvent the "spike" problem present
           in initvar=0, but unlike initvar=1 or 2 it gives a much better
           treatment to the "center" variables which are treated independently,
           and to multiple polygons (also treated independently).
           This is the recommended option (it is NOT the default, for
           backward-compatibility reasons).
        */
#define CENTER_P 0.5 /* size of perturbations. maybe these should be */
#define RADIUS_P 0.5 /* options. */
#define RADIUS_S 1.0 /* the bigger this is, the sharper the spikes will be */
```

```
        for(m=1;m≤ndim;m++){
            k=1;
            for(i=0;i<lp.npolygons;i++){
                int minpolyg,dist;
                /* center x coordinate */
                p[m+1][k]=p[1][k] + ((k==m) ? CENTER_P : 0);
                k++;
                /* center y coordinate */
                p[m+1][k]=p[1][k] + ((k==m) ? CENTER_P : 0);
                k++;

                /* is m inside the next polygon? */
                if(m≥k && m< k+lp.polygons[i].npoints)
                    minpolyg=m-k;
                else
                    minpolyg=(-1);
                for(j=0;j<lp.polygons[i].npoints;j++){
                    /* if m is not inside this polygon, leave this
                       polygon as is, otherwise add to it a smoothed-out
                       spike.
                    */
                    if(minpolyg≥0){
                        dist=ABS(j-minpolyg);
                        /* make dist cyclic */
                        if(lp.polygons[i].npoints-dist < dist)
                            dist=lp.polygons[i].npoints-dist;
                        p[m+1][k]=p[1][k]+RADIUS_P/(1+dist*RADIUS_S);
                    } else {
                        p[m+1][k]=p[1][k];
                    }
                    k++;
                }
            }
        }
        break;
    default:
        fprintf(stderr,
                "minverse():  got invalid value for initvar option (%d)\n",
                opts→initvar);
        exit(1);
        break;
    }

    /* evaluate the check_guess at the n+1 initial vectors */
    fprintf(stderr,"checking initial %d guesses:  ",ndim+1);
    for(i=1;i≤ndim+1;i++){
        y[i]=check_guess(p[i],&ud);
        fprintf(stderr,".");
    }
    fprintf(stderr,"done.\n");

    count=ngiveup;
    if(amoeba(p,y,ndim,1e-5,check_guess,&i,&ud,&count)){
        fprintf(stderr,"**** amoeba converged\n");
    } else {
        fprintf(stderr,"**** amoeba did not converge\n");
    }

    /* choose the best p[i], move it to p[1] where 'goto RESTART'
       expect the initial condition to be.
    */
    k=1;
    a=y[1];
    for(i=2;i≤ndim+1;i++){
        if(y[i]<a){
            k=i;
            a=y[i];
        }
```

```
    }
    fprintf(stderr,"Best of simplex guesses:  node %d:  %g\n",k,a);
    for(i=1;i≤ndim;i++){
        p[1][i]=p[k][i];
    }

#if 0
    /* smoothing */
    k=1;
    for(i=0;i<lp.npolygons;i++){
        k++; /* leave center alone */
        k++;
        for(j=0;j<lp.polygons[i].npoints;j++){
            /* initial radii */
            k++;
            if(j≠0 && j≠(lp.polygons[i].npoints-1))
            p[1][k]=(p[1][k-1]+2*p[1][k]+p[1][k+1])/4.0;
        }
    }
#endif
    ud.restarts++;
    /* multiply ngiveup by opt->ngiveupmult so thar next time we'll let
       the minimization algorithm do more iterations before restarting
       it.
    */
    ngiveup = (double)ngiveup * opts→ngiveupmult;
    goto RESTART;

    /* clean up */
    free_dmatrix(p,1,ndim+1,1,ndim);
    free_dvector(y,1,ndim+1);
    /* TODO: FREE LPOLYGON UD.LP as well! */
    FreeRaySet(ud.tryrs);
}

/* In the minverse algorithm, the parameters that the minimization algorithm
   tries to find are the center of the polygon, and the radiuses at fixed
   angles.
   There's a problem, however, that it's hard for the minimization algorithm
   to correct a "bad" choice for a center, because moving the center moves
   the entire polygon. Sometimes, when the initial guess polygon is very
   far (and off-center) from the correct polygon, the center doesn't move
   fast enough, and may even be very close the the boundary of the polygon -
   in which case we start getting "negative radius" penalties to try to
   fix that problem.
   The following routine can be use whenever the minimization algorithm is
   stopped and restarted - to choose a new center for the polygon (currently,
   simple average of the polygon's points), and resample the polygon at the
   given fixed angles from the new center.
   Note: the d vector must start at 1, as used in the minimization
   algorithm above (see check_guess) and is a packing of the centers and
   radiuses. d is the output too.
*/
#include "spline.h"

static void
guess_recenter(double *d, LPolygon lp)
{
    double *p;
    double ncx,ncy;
    int np,i,j,k;
    double *r, *t;
    struct splinedata *sd;

    p=d+1; /* d[0] is undefined */

    for(i=0;i<lp.npolygons;i++){
#define CENTERX p[0]
```

```
#define CENTERY p[1]
#define RADIUS(n) p[2+(n)]
        np=lp.polygons[i].npoints;

        fprintf(stderr,"recentering polygon %d.  Old center:  %g %g\n",
                i,CENTERX,CENTERY);
        /* calculate new center */
        ncx=ncy=0;
        for(j=0;j<np;j++){
            ncx += CENTERX + RADIUS(j)*lp.polygons[i].points[j].x;
            ncy += CENTERY + RADIUS(j)*lp.polygons[i].points[j].y;
        }
        ncx/=np;
        ncy/=np;
        /* prepare arrays r[j], t[j], where t[j] is the angle of the
           j'th point relative to the new center, and r[j] is the radius
           relative to the new center. Note that it is the real radius:
           it should be divided by the norm of lp.polygons[i].points[j]
           before being kept in the RADIUS array.
        */
        r=(double *)malloc(sizeof(double)*(np+1));
        t=(double *)malloc(sizeof(double)*(np+1));
        for(j=0;j<np;j++){
            double x, y;
            x = CENTERX + RADIUS(j)*lp.polygons[i].points[j].x - ncx;
            y = CENTERY + RADIUS(j)*lp.polygons[i].points[j].y - ncy;
            /* find angle in [0,2PI] */
            t[j]=atan2(y,x); /* atan2 finds in [-PI,PI] */
            if(t[j]<0) t[j]+=CONST_PI*2;
            r[j]=sqrt(x*x+y*y);
            /*fprintf(stderr,"t,r[%d]=%g,%g\n",j,t[j],r[j]);*/
        }
        /* reorder t,r: make t increasing. Note that if polygon is around
           the center just once, there is only one point of non-increasing
           and we need to fix that - but in fact the polygon might not
           be star-shaped from the new center... We ignore this problem
           and force it becoming star-shaped (albiet, maybe a little
           different!)
           We use a stupid simple sorting because this operation is done
           only a few times during the algorithm.
        */
        k=1;
        while(k){
            double tmp;
            k=0;
            for(j=1;j<np;j++){
                if(t[j]<t[j-1]){
                    tmp=t[j];
                    t[j]=t[j-1];
                    t[j-1]=tmp;
                    tmp=r[j];
                    r[j]=r[j-1];
                    r[j-1]=tmp;
                    k++;
                }
            }
        }

        for(j=0;j<np;j++){
            fprintf(stderr,"t,r[%d]=%g,%g\n",j,t[j],r[j]);
        }

        /* again the first point */
        t[np]=t[0]+2*CONST_PI; /* probably 2*PI */
        r[np]=r[0];


        /* check that t is increasing (this can't be false after our
```

```
            sorting above!)
        */
        for(j=1;j<np;j++)
            if(t[j]≤t[j-1]){
                fprintf(stderr,"WARNING: GUESS_RECENTER FAILED - T ORDER\n");
                free(r);
                free(t);
                return;
            }
        /* note: 0 is given to prepare spline as adhoc. the real condition
            would be to make the spline circular, with the two ends actually
            the same point */
        sd=prepare_spline(t,r,np+1,0.0,0.0);
        /* resample the spline, calculating new radiuses and putting them
            in r.
        */
        for(j=0;j<np;j++){
            double tt;
            /* at what angle to sample? */
            tt=atan2(lp.polygons[i].points[j].y,
                     lp.polygons[i].points[j].x);
            if(tt<0) tt+=CONST_PI*2;
            r[j]=interpolate_spline(sd,tt);
            if(r[j]<0){
                fprintf(stderr,"WARNING: GAUSS_RECENTER FAILED - NEW R<0\n");
                free(r);
                free(t);
                free_spline(sd);
                return;
            }
        }

        /* finally copy the results back into the vector */
        CENTERX=ncx;
        CENTERY=ncy;
        fprintf(stderr," New center:  %g %g\n", CENTERX,CENTERY);

        for(j=0;j<np;j++){
            RADIUS(j)=r[j];
        }

        free(r);
        free(t);
        free_spline(sd);

        p+=np+2; /* skip to next polygon */
    }
}

/* output_miverse_result outputs the latest minverse layered polygon in
   a gnuplot format suitable for the 'gp5' viewing script.
*/
static void
output_minverse_result(double *guess, UserData *ud, double d)
{
    static int noutputs=0;
    FILE *fp;
    char fn[512];
    int ii,kk,i;
    double centerx,centery;
    extern char problem_name[];

    fprintf(stderr,"outputting polygon...  %d\n",ud→resultlp→npolygons);
    sprintf(fn,"out/OUT%d",++noutputs);
    if((fp=fopen(fn,"w"))==NULL){
        perror(fn);
    } else {
        fprintf(fp,"# %d %d %g A%s ",noutputs,ud→neval,d,problem_name);
```

```
        /* output centers (similar to unpack code in check_guess) */
        kk=1;
        for(ii=0;ii<ud→resultlp→npolygons;ii++){
            centerx=guess[kk++]; centery=guess[kk++];
            kk+=ud→resultlp→polygons[ii].npoints;
            fprintf(fp,"%g,%g ",centerx,centery);
        }
        fprintf(fp,"\n");
        _outputLPolygon(*(ud→resultlp),fp);
        fclose(fp);
    }
    sprintf(fn,"out/XX%d",noutputs);
    if((fp=fopen(fn,"w"))==NULL){
        perror(fn);
    } else {
        for(i=0;i<ud→rs→nrays;i++){
            fprintf(fp,"%d %g %g\n",i,ud→rs→rays[i].result,ud→tryrs.rays[i].result);
        }
        fclose(fp);
    }
    printStats(stdout);
}
```

minverse.h

```
#ifndef INCLUDED_MINVERSE_H
#define INCLUDED_MINVERSE_H

struct minverse_options {
    int ngiveup; /* after how many iterations give up the minimization,
                    and restart it with new perturbations. */
    double ngiveupmult; /* every time we give up, we multiply ngiveup
                           again my ngiveupmult (so that next time we'll
                           let it run for more iterations) */
    double guesscircle_rad; /* radius of initial guess circle */
#define GUESSCIRCLE_MAXN 10
    double guesscircle_cx[GUESSCIRCLE_MAXN]; /* centers of upto 10 polygons */
    double guesscircle_cy[GUESSCIRCLE_MAXN];
    int auto_recenter; /* recenter when restarting minimization */
    int initvar; /* type of perturbations to make on initial guess to
                    create the guess simplex. */
};

void minverse(RaySet dataRays, LPolygon lp,
              const struct minverse_options *opts);
#endif
```

main5.c

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "layered_polygon.h"
#include "xray.h"
#include "minverse.h"


double RadPolyg_circle(tet,id)
    double tet,id;
{
    return id;
}

/* file format:
        # some comments
        # and more comments
```

```
        @options
        polygs/polygontocheck
        3 50 -1 50 -1 50 1 <- definition of guess polygon
        50 0.1 1.0 -3 3 <- rayset
        50 1.0 0.2 -3 3 <- another rayset (more until EOF)
 */

int loadProblem(prs,plp,pguesslp,popts,fn)
RaySet *prs;
LPolygon *plp;
LPolygon *pguesslp;
struct minverse_options *popts;
char *fn;
{
    FILE *fp;
    int c,i,nrs;
    double g;
    char s[80];
    struct {
        int nrays;
        double dx,dy, c1,c2;
    } def[10];

    if((fp=fopen(fn,"r"))==NULL){
        perror(fn);
        return 0;
    }
    /* swallow comments and '@' options in beginning */
    fprintf(stderr,"---------------------------------------- reading %s...\n",fn);
    while((c=getc(fp))=='#'||c=='@'){
        if(c=='#'){
            while((c=getc(fp))≠'\n' && c≠EOF) putc(c,stderr);
            putc('\n',stderr);
        } else { /* '@' options */
            if(!fgets(s,sizeof(s),fp)) goto FAIL;
            if(!strncmp(s,"ngiveup ",8)) {
                if(sscanf(s,"ngiveup %d",&popts→ngiveup)≠1) goto FAIL;
            } else if(!strncmp(s,"ngiveupmult ",12)) {
                if(sscanf(s,"ngiveupmult %lg",&popts→ngiveupmult)≠1) goto FAIL;
            } else if(!strncmp(s,"guesscircle_rad ",16)) {
                if(sscanf(s,"guesscircle_rad %lg",&popts→guesscircle_rad)≠1) goto FAIL;
            } else if(!strncmp(s,"guesscircle_cx ",15)) {
                if(sscanf(s,"guesscircle_cx %lg %lg %lg %lg %lg %lg %lg %lg %lg %lg",
                            &popts→guesscircle_cx[0],&popts→guesscircle_cx[1],
                            &popts→guesscircle_cx[2],&popts→guesscircle_cx[3],
                            &popts→guesscircle_cx[4],&popts→guesscircle_cx[5],
                            &popts→guesscircle_cx[6],&popts→guesscircle_cx[7],
                            &popts→guesscircle_cx[8],&popts→guesscircle_cx[9]
                        )<1) goto FAIL;
            } else if(!strncmp(s,"guesscircle_cy ",15)) {
                if(sscanf(s,"guesscircle_cy %lg %lg %lg %lg %lg %lg %lg %lg %lg %lg",
                            &popts→guesscircle_cy[0],&popts→guesscircle_cy[1],
                            &popts→guesscircle_cy[2],&popts→guesscircle_cy[3],
                            &popts→guesscircle_cy[4],&popts→guesscircle_cy[5],
                            &popts→guesscircle_cy[6],&popts→guesscircle_cy[7],
                            &popts→guesscircle_cy[8],&popts→guesscircle_cy[9]
                        )<1) goto FAIL;
            } else if(!strncmp(s,"auto_recenter",13)){
                popts→auto_recenter=1;
            } else if(!strncmp(s,"initvar ",8)) {
                if(sscanf(s,"initvar %d",&popts→initvar)≠1) goto FAIL;
            } else {
                fprintf(stderr,"unknown option line @%s\n",s);
                goto FAIL;
            }
        }
    }
  if(c==EOF) goto FAIL; else ungetc(c,fp);
```

```
  fprintf(stderr,"---------------------------------------- reading done.\n",fn);

  /* read polygon file name */
  fscanf(fp,"%s\n",s);
  if(!loadLPolygon(plp,s)) goto FAIL;
  fprintf(stderr, "Polygon loaded:  %s.\n",s);

  /* read guess polygon size (in the future we might make this a file name) */
  if(fscanf(fp,"%d",&nrs)==1){
    NewLPolygon(*pguesslp,nrs);
    for(i=0;i<nrs;i++){
      if(fscanf(fp,"%d %lg",&c,&g)≠2) goto FAIL;
      NewPolygon(pguesslp→polygons[i],c,g);
    }
  } else goto FAIL;

  /* read rayset definitions */
  nrs=0;
  while(fscanf(fp,"%d %lg %lg %lg %lg\n", &(def[nrs].nrays),
                &(def[nrs].dx), &(def[nrs].dy), &(def[nrs].c1), &(def[nrs].c2))
        == 5)
    nrs++;
  fprintf(stderr, "%d raysets found.\n",nrs);
  if(!nrs) goto FAIL;

  for(c=0,i=0;i<nrs;i++)
    c+=def[i].nrays;
  NewRaySet(*prs, c);
  for(c=0,i=0;i<nrs;i++){
    eqdRays(prs→rays+c, def[i].nrays, def[i].dx, def[i].dy,
            def[i].c1+1e-10, def[i].c2+1e-10);
    c+=def[i].nrays;
  }

  fclose(fp);
  return 1;

FAIL:
  fclose(fp);
  return 0;
}

char problem_name[100];

main(argc, argv)
int argc;
char *argv[];
{
    LPolygon lp,guesslp;
    RaySet rs;
    struct minverse_options opts;

    double dx,dy,ox,oy;
    double d,weight;
    int i;
    FILE *fp;
    long seed,time();

    if(argc≠2){
      fprintf(stderr, "Usage:  %s problem-file\n", argv[0]);
      exit(1);
    }

    /* Default options. The choice of defaults is mainly for backwar
       d-compatibility with old problem setups, is are NOT necessarily the
       best.
    */
    opts.ngiveup=5000;
```

```
    opts.ngiveupmult=1.0;
    opts.guesscircle_rad=1.0;
    opts.auto_recenter=0;
    opts.initvar=2;
    for(i=0;i<GUESSCIRCLE_MAXN;i++)
        opts.guesscircle_cx[i]=opts.guesscircle_cy[i]=0.0;

    if(!loadProblem(&rs, &lp, &guesslp, &opts, argv[1])){
      fprintf(stderr, "Failed loading problem file %s\n",argv[1]);
      exit(2);
    }
    strncpy(problem_name,argv[1],sizeof(problem_name)-1);

    outputLPolygon(lp,"out/gp%d.dat",1);
    xrayset(lp,rs);


    /* show the configuration of the problem, to be viewed by view5 */
    outputLPolygon(lp,"out/main5-polyg",1);
    debug_xray=fopen("out/main5-rays","w");
    xrayset(lp,rs);
#undef NOISE
#ifdef NOISE
#define E 0.05/ /* maximal error 5% */
    seed=time((long*)0);
    srand48(seed);
#define RANDOM ((drand48()-0.5)*2*E) /* random number in [-E,E] */
    for(i=0;i<rs.nrays;i++){
      rs.rays[i].result *= 1+RANDOM;
    }
#endif
    fclose(debug_xray);
    debug_xray=(FILE *)0;


    fprintf(stderr,"minverse options:  ngiveup=%d, guesscircle_rad=%g, "
            "auto_recenter=%d, ngiveupmult=%g, initvar=%d\n",
            opts.ngiveup,opts.guesscircle_rad,opts.auto_recenter,
            opts.ngiveupmult,opts.initvar);

    minverse(rs,guesslp,&opts);

    exit(0);
}
```

layered_polygon.h

```
/* A layered polygon is 2D shape composed of multiple overlayed polygons
   of various densities. A density of point is the sum of densities on it.
   For example, a hole within a unit density polygon will be in itself a
   polygon with density -1.
   This formulation will allow, for example, a hole to "move outside" the
   enclosing polygon during minimization iterations, without causing fatal
   errors only because such polygons cannot be represented. Of course a
   real solution will finally have the hole *inside* the other polygon.
 */
typedef struct {
        double x;
        double y;
} Point;

typedef struct {
    int npoints; /* number of points in polygon */
    Point *points; /* polygon's points. the last point is connected to
                                  the first in the polygon */
    double density;
```

```
} Polygon;

typedef struct {
        int npolygons;
        Polygon *polygons;
} LPolygon;

/* use NewLPolygon to allocate a n-polygon Layered-polygon in the variable lp.
   use NewPolygon to allocate a n-point polygon with density d FOR EACH of the
   polygons in the LPolygon (lp.polygons[i]).
*/
#define AllocateLPolygon(lp) ((lp).polygons=(Polygon*)malloc((lp).npolygons*sizeof(Polygon)))
#define NewLPolygon(lp,n) ((lp).npolygons=(n),AllocateLPolygon(lp))
#define AllocatePolygon(p) ((p).points=(Point*)malloc((p).npoints*sizeof(Point)))
#define NewPolygon(p,n,d) ((p).density=(d),(p).npoints=(n),AllocatePolygon(p))
```

layered_polygon.c

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "layered_polygon.h"

/*****************************************************************************/
/* puts a radial polygon in Polygon p, based on the function f (with second
   parameter id so a similar f can serve a few polygons) sampled at equidistant
   angles (as many as there are points allocated in p), and moved to center
   (centerx,centery).
*/

#define CONST_PI 3.14159265358979323846
void SetRadialPolygon(p,centerx,centery, f,id)
    Polygon p;
    double (*f)();
    double id;
    double centerx,centery;
{
    int i;
    double r,tet;
    for(i=0;i<p.npoints;i++){
        tet=2*CONST_PI/(p.npoints)*i;
        r=f(tet,id);
        p.points[i].x=centerx+r*cos(tet);
        p.points[i].y=centery+r*sin(tet);
    }
}
/* a more advanced version */
void SetARadialPolygon(p,centerx,centery,rot,f,id)
    Polygon p;
    double (*f)();
    double id;
    double centerx,centery;
    double rot; /* rotation (1/2 means 180degrees, or 2PI*1/2 radians)
                   rot must be between -1 to 1 */
{
    int i;
    double r,tet;
    for(i=0;i<p.npoints;i++){
        tet=(1.0/(p.npoints)*i + rot);
        if(tet>1) tet-=1;
        if(tet<0) tet+=1;
        tet=2*CONST_PI*tet;
        r=f(tet,id);
        p.points[i].x=centerx+r*cos(tet);
        p.points[i].y=centery+r*sin(tet);
    }
}
```

```
/* for example, with the following function,
        double RadPolyg_circle(tet,id)
            double tet,id;
        {
            return id;
        }
    one can do:
     NewLPolygon(lp,2);
     NewPolygon(lp.polygons[0],10,1.0);
     SetRadialPolygon(lp.polygons[0], 0.0,0.0, RadPolyg_circle, 2.0);
     NewPolygon(lp.polygons[1],10,-1.0);
     SetRadialPolygon(lp.polygons[1], 0.0,0.0, RadPolyg_circle, 1.0);
*/

/*****************************************************************************/
/* outputLPolygon outputs an Lpolygon in gnuplot format.
    example usage: outputLPolygon(lp,"output%d.dat",7)
    the resulting file num must be less then 512 chars long.
 */
void outputLPolygon(lp,filetemplate,n)
     LPolygon lp;
     char *filetemplate;
     int n;
{
     int i,k;
     char fn[512];
     FILE *fp;

     sprintf(fn,filetemplate,n);
     if((fp=fopen(fn,"w"))==NULL){
          perror(fn);
          return;
     }

     for(i=0;i<lp.npolygons;i++){
          for(k=0;k<lp.polygons[i].npoints;k++){
               fprintf(fp,"%g %g\n",lp.polygons[i].points[k].x,
                                   lp.polygons[i].points[k].y);
          }
          /* and return to the first point, and leave an empty line for the
              end of polygon
              */
          fprintf(fp,"%g %g\n\n",lp.polygons[i].points[0].x,
                                 lp.polygons[i].points[0].y);
     }
     fclose(fp);
}

/* similar, but doesn't open the file. should be called in outputLPolygon
    instead of replicating the code */
void _outputLPolygon(lp,fp)
     LPolygon lp;
     FILE *fp;
{
     int i,k;
     for(i=0;i<lp.npolygons;i++){
          for(k=0;k<lp.polygons[i].npoints;k++){
               fprintf(fp,"%g %g\n",lp.polygons[i].points[k].x,
                                   lp.polygons[i].points[k].y);
          }
          /* and return to the first point, and leave an empty line for the
              end of polygon
              */
          fprintf(fp,"%g %g\n\n",lp.polygons[i].points[0].x,
                                 lp.polygons[i].points[0].y);
     }
}
/* compact format for saving and later inputting LPolygons. note that
```

```
    the load allocates the new polygon */
int saveLPolygon(lp,fn)
     LPolygon lp;
     char *fn;
{
     int i,k;
     FILE *fp;

     if((fp=fopen(fn,"w"))==NULL){
          perror(fn);
          return;
     }

     fprintf(fp,"#LPolygon\n%d\n",lp.npolygons);
     for(i=0;i<lp.npolygons;i++){
          fprintf(fp,"%g\n",lp.polygons[i].density);
          fprintf(fp,"%d\n",lp.polygons[i].npoints);
          for(k=0;k<lp.polygons[i].npoints;k++){
               fprintf(fp,"%g %g\n",lp.polygons[i].points[k].x,
                                   lp.polygons[i].points[k].y);
          }
     }
     fclose(fp);
}

int loadLPolygon(plp,fn)
     LPolygon *plp;
     char *fn;
{
     int i,k;
     FILE *fp;
     char s[80];

     if((fp=fopen(fn,"r"))==NULL){
          perror(fn);
          return 0;
     }

     fgets(s,80,fp);
     if(strcmp(s,"#LPolygon\n")){
          fprintf(stderr,"not a polygon %s:  %s",fn,s);
          return 0;
     }

     fscanf(fp,"%d\n",&plp→npolygons);
     AllocateLPolygon(*plp);

     for(i=0;i<plp→npolygons;i++){
          fscanf(fp,"%lg\n",&plp→polygons[i].density);
          fscanf(fp,"%d\n",&plp→polygons[i].npoints);
          AllocatePolygon(plp→polygons[i]);
          for(k=0;k<plp→polygons[i].npoints;k++){
               fscanf(fp,"%lg %lg\n",&plp→polygons[i].points[k].x,
                                   &plp→polygons[i].points[k].y);
          }
     }
     fclose(fp);
     return 1;
}

void freeLPolygon(lp)
     LPolygon lp;
{
     int i,k;
     for(i=0;i<lp.npolygons;i++){
          free(lp.polygons[i].points);
     }
     free(lp.polygons);
```

```
}
```

xray.h

```
#include <stdio.h>

/* a single xray: has a direction (dx,dy) and origin (ox,oy). Note that the
   origin is only used for the affine translation - the specific origin point
   is not important
*/
typedef struct {
    double ox,oy,dx,dy;
    double result;
} Ray;

/* a set of X-rays */
typedef struct {
    Ray *rays;
    int nrays;
} RaySet;

/* use NewRaySet to allocate a n-ray rayset in the variable rs. */
#define AllocateRaySet(rs) ((rs).rays=(Ray*)malloc((rs).nrays*sizeof(Ray)))
#define NewRaySet(rs,n) ((rs).nrays=(n),AllocateRaySet(rs))
#define FreeRaySet(rs) (free((rs).rays))

double xray();
void xrayset();
void eqdRays();

extern FILE *debug_xray;
```

xray.c

```
#include "layered_polygon.h"
#include "xray.h"
#include <math.h>
#include <stdio.h>

#define EPSILON 1e-14

FILE *debug_xray=(FILE *)0; /* set to a file pointer to see the lines xray
                                    does */

/* xray returns the integral of density of polygons a line.
   the line is a vector (dx,dy) starting at (ox,oy)
   MAKE SURE THAT (dx,dy) is a normalized vector. Xray currently doesn't
   do it.
*/

typedef struct { /* structure specifying an intersection with a polygon */
    int polygon; /* polygon number intersected */
    double alpha; /* where on the line it cuts this polygon edge. The
                        intersection is (ox,oy)+alpha*(dx,dy) */
} Intersect;

int comparealpha(i1,i2)
    Intersect *i1,*i2;
{
    if(i1->alpha < i2->alpha) return -1;
    if(i1->alpha > i2->alpha) return 1;
    return 0;
}

double xray(lp, ox,oy,dx,dy)
```

```
    LPolygon lp;
    double ox,oy,dx,dy;
{
    int i,k;
#define MAXINTER 100
#define MAXPOLY 100
    static Intersect intersections[MAXINTER]; /* TODO: make this malloced on
                                                 demand. this limit is the limit on the
                                                 number of polygon edges which can be
                                                 intersected with the line */
    int nintersections=0;
    static int inpoly[100]; /* TODO: make this malloced on demand - according
                                 to lp.npolygons. I can even add a 'flag' field
                                 in the Polygon structure, but this is ugly */


    double nx, ny; /* normal vector to (dx,dy). */
    double x1,y1, x2,y2; /* two points of edge we are working on, after
                              the origin (ox,oy) is subtracted
                            */
    double a1,a2; /* will be defined as the shortest (i.e. perpendicular)
                       (signed) distance between x1,x2 and the line.
                    */
    double density, mass; /* this is the value we are trying to calculate
                                 density is used temorarily */

    nx=dy; ny=-dx; /* orthogonal to (dx, dy) but not normalized currently */


    for(i=0;i<lp.npolygons;i++){
        for(k=0;k<lp.polygons[i].npoints;k++){
            int km; /* (k-1) mod lp.polygons[i].npoints */
            if(k>0){
                km=k-1;
            } else {
                km=lp.polygons[i].npoints - 1;
            }


            x1=lp.polygons[i].points[km].x - ox;
            y1=lp.polygons[i].points[km].y - oy;

            x2=lp.polygons[i].points[k].x - ox;
            y2=lp.polygons[i].points[k].y - oy;

            /* if they the projections have opposite signs (see picture "xray"
               on page (1) of my notes) the edge cuts the line */
            a1=x1*nx+y1*ny;
            a2=x2*nx+y2*ny;
            /* ONLY UNTIL I FIX THE PROBLEM: */
#define ABS(x) ((x)<0 ? -(x) : (x))
                if(ABS(a1)<1e-15 || ABS(a2)<1e-15){
                    fprintf(stderr,"NOT YET DONE: CASE NOT HANDELED IN XRAY.C\n");
                    abort();
                }

            if( a1 * a2 ≤ 0 && a2≠0){
                /* The reason for the "a2!=0" check: if a line intersects in a
                   vertex exactly, we will get two cuts: one of one edge (with
                   a1=0) and one of a second edge (with a2=0). So we don't
                   count the intersection with a2=0 - only the other
                */
                intersections[nintersections].alpha=
                    (x1*dx+y1*dy)*(a2/(a2-a1)) + (x2*dx+y2*dy)*(a1/(a1-a2));
                intersections[nintersections].polygon=i;

                nintersections++;
            }
```

```
        }
    }

    /* all intersections of the line with the polygons have been found. we
       will now sort them in order of increasing alphas, i.e. sort the
       intersection points along the given line (which has a direction: that
       of (dx,dy). */
    /* in the future I might convert this to a simple sort, which will
       probably even be faster for sorting ~4 numbers (which will require
       at most 16 compares, but not calling large functions and compare
       functions.
       */
    qsort(intersections, nintersections, sizeof(Intersect), comparealpha);

    if(debug_xray){
        for(i=0;i<nintersections;i++){
            fprintf(debug_xray,"%g %g\n",ox+dx*intersections[i].alpha,
                    oy+dy*intersections[i].alpha);
            if(i==nintersections-1)fprintf(debug_xray,"\n");
        }
    }

    /* now that we have a sorted the intersections along the line, we will
       go along this sorted list and find the total mass under the line */

    for(i=0;i<lp.npolygons;i++) inpoly[i]=0;
    density=0;
    mass=0;
    for(i=0;i<nintersections;i++){
        /* unless we are at the first point, add the contribution to the
           mass of the previous section of the line */
        if(i){
            mass += density*(intersections[i].alpha-intersections[i-1].alpha);
        }
        if(inpoly[intersections[i].polygon]==0){
            /* entering a polygon: */
            inpoly[intersections[i].polygon] = 1;
            density += lp.polygons[ intersections[i].polygon ].density;
        } else {
            /* exiting a polygon: */
            inpoly[intersections[i].polygon] = 0;
            density -= lp.polygons[ intersections[i].polygon ].density;
        }
    }

    if(density≠0){
        fprintf(stderr,"INTERNAL INCONSISTENCY: density!=0 after xray finishes\n");
        fprintf(stderr," given:  (ox,oy)=(%g,%g), (dx,dy)=(%g,%g)",ox,oy,dx,dy);
        fprintf(stderr," calculated:  mass=%g, density=%g\n",mass,density);
        abort(); /* this is a bug if density!=0 this is cause by the
                    not good a2!=0 fix */
    }
    return mass;
}

/* ray sets */
void xrayset(lp,rs)
    LPolygon lp;
    RaySet rs;
{
    int i;
    for(i=0; i<rs.nrays; i++){
        rs.rays[i].result = xray(lp, rs.rays[i].ox, rs.rays[i].oy,
                                 rs.rays[i].dx, rs.rays[i].dy);
    }
}

/* eqdRay builds an equidistant ray set in the Ray vector rays (nrays
```

```
    rays). It can also be used to build a *part* of a bigger rayset (called
    multiple times with several ofsets into the rays array.
    the rays have a direction dx,dy, and they are equally distributed between
    two parallel lines in that direction - whose coordinates orthogonal to this
    direction (starting at 0) are c1..c2.

    The ray directions c1,c2 don't have to be normalized - eqdRayset will do it
    for you.
*/
void eqdRays(rays,nrays, dx,dy, c1,c2)
     Ray *rays;
     int nrays;
     double dx,dy, c1,c2;
{
     double nx,ny,d;
     int i;

     /* normalize dx,dy */
     d=sqrt(dx*dx + dy*dy);
     dx=dx/d; dy=dy/d;

     /* orthogonal vector to dx,dy */
     /* the sign is chosen here so if a x-axis vector is given, the y-axis
        vector (to the right) is the orthogonal. I might reverse the sign
        if I want.
        */
     nx=-dy;
     ny=dx;

     /* make the equidistant rays */
     for(i=0; i<nrays; i++){
          rays[i].dx=dx; /* note that (dx,dy) is already normalized */
          rays[i].dy=dy;
          rays[i].ox=nx*(c1+(c2-c1)*(((double)i)/(nrays-1)));
          rays[i].oy=ny*(c1+(c2-c1)*(((double)i)/(nrays-1)));
     }
}
```

amoeba.c

```
/* Multidimensional minimization. Taken from "Numerical Recipes in C - The Art
    of Scientific Computing", 1988, page 305
*/
#include <math.h>

#define NMAX 1e8 /* The maximum allowed number of function evaluations, */
#define ALPHA 1.0 /* and three parameters defining expansions and */
#define BETA 0.5 /* contractions. */
#define GAMMA 2.0

#define GET_PSUM for (j=1;j≤ndim;j++) { for (i=1,sum=0.0;i≤mpts;i++)\
                                        sum += p[i][j]; psum[j]=sum;}

/* I have added a 'count' variable, which is decremented on every step,
    and amoeba stops (returning 0) when gets to 0 */
int amoeba(p,y,ndim,ftol,funk,nfunk,userData,count)
double **p,y[],ftol,(*funk)();
int ndim,*nfunk,*count;
char *userData; /* a general pointer, to be given to the evaluated function
                   so that it will know its constant parameters */

/* multidimensional minimization of the function func(x) where x[1..ndim] is
    a vector in nim dimensions, by the downhill simplex method of Nelder and
    Mead. The matrix p[1..ndim+1][1..ndim] is input. Its ndim+1 rows are
    ndim-dimensional vectors which are the vertices of the starting simplex.
    Also input is the vector y[1..ndim+1], whose components must be
    pre-initialized to the values of funk evaluated at the ndim+1 vertices
```

```
    (rows) of p; and ftol the fractional convergence tolerence to be achived
    in the function value (n.b.!). On output, p and y will have been reset to
    ndim+1 new points all within ftol of a minimum function value, and nfunk
    gives the number of function evaluations taken.
*/
{
        int i,j,ilo,ihi,inhi,mpts=ndim+1;
        double ytry,ysave,sum,rtol,amotry(),*psum,*dvector();
        void nrerror(),free_dvector();

        psum=dvector(1,ndim);
        *nfunk=0;
        GET_PSUM
        for (;;) {
            /* first we must determine which point is the highest (worst),
               next-highest, and lowest (best), by looping over the points
               in the simplex.
            */
                ilo=1;
                ihi = y[1]>y[2] ? (inhi=2,1) : (inhi=1,2);
                for (i=1;i≤mpts;i++) {
                        if (y[i] < y[ilo]) ilo=i;
                        if (y[i] > y[ihi]) {
                                inhi=ihi;
                                ihi=i;
                        } else if (y[i] > y[inhi])
                                if (i ≠ ihi) inhi=i;
                }
                rtol=2.0*fabs(y[ihi]-y[ilo])/(fabs(y[ihi])+fabs(y[ilo]));
                /* compute the fractional range from the highest to lowest and
                   return if satisfactory */
                if (rtol < ftol) break;
                if (((*count)--)==0){ /*** NYH ***/
                     free_dvector(psum,1,ndim);
                     return 0;
                }
                if (*nfunk ≥ NMAX) nrerror("Too many iterations in AMOEBA");
                /* begin a new interation. First extrapolate by a factor ALPHA
                   through the face of the simplex across from the high point,
                   i.e. reflect the simplex from the high point. */
                ytry=amotry(p,y,psum,ndim,funk,ihi,nfunk,-ALPHA,userData);
                if (ytry ≤ y[ilo])
                     /* gives a result better than the best point, so try an
                        additional extrapolation by a factor GAMMA */
                      ytry=amotry(p,y,psum,ndim,funk,ihi,nfunk,GAMMA,userData);
                else if (ytry ≥ y[inhi]) {
                     /* the reflected point is worse than the second-highest, so
                        look for an intermediate lower point,i.e. do a
                        one-dimensional contraction. */
                      ysave=y[ihi];
                      ytry=amotry(p,y,psum,ndim,funk,ihi,nfunk,BETA,userData);
                      if (ytry ≥ ysave) {
                           /* can't seem to get rid of that high point. better
                              contract around the lowest (best) point. */
                            for (i=1;i≤mpts;i++) {
                                    if (i ≠ ilo) {
                                            for (j=1;j≤ndim;j++) {
                                                    psum[j]=0.5*(p[i][j]+p[ilo][j]);
                                                    p[i][j]=psum[j];
                                            }
                                            y[i]=(*funk)(psum,userData);
                                    }
                            }
                            /* keep track of function evaluations */
                            *nfunk += ndim;
                            GET_PSUM
                      }
                }
```

```
        } /* go back for the test of doneness and the next iteration */
        free_dvector(psum,1,ndim);
        return(1);
}

double amotry(p,y,psum,ndim,funk,ihi,nfunk,fac,userData)
double **p,*y,*psum,(*funk)(),fac;
int ndim,ihi,*nfunk;
char *userData;
/* exterpolates by a factor fac through the face of the simplex across from
   the high point, tries it, and replaces the high point if the new point is
   better */
{
        int j;
        double fac1,fac2,ytry,*ptry,*dvector();
        void nrerror(),free_dvector();

        ptry=dvector(1,ndim);
        fac1=(1.0-fac)/ndim;
        fac2=fac1-fac;
        for (j=1;j≤ndim;j++) ptry[j]=psum[j]*fac1-p[ihi][j]*fac2;
        ytry=(*funk)(ptry,userData); /* evaluate the function at the trial point */
        ++(*nfunk);
        if (ytry < y[ihi]) {
            /* if it's better than the highest, then replace the highest */
                y[ihi]=ytry;
                for (j=1;j≤ndim;j++) {
                        psum[j] += ptry[j]-p[ihi][j];
                        p[ihi][j]=ptry[j];
                }
        }
        free_dvector(ptry,1,ndim);
        return ytry;
}

#undef ALPHA
#undef BETA
#undef GAMMA
#undef NMAX
```

nrutil.h

```
float *vector();
float **matrix();
float **convert_matrix();
double *dvector();
double **dmatrix();
int *ivector();
int **imatrix();
float **submatrix();
void free_vector();
void free_dvector();
void free_ivector();
void free_matrix();
void free_dmatrix();
void free_imatrix();
void free_submatrix();
void free_convert_matrix();
void nrerror();
```

nrutil.c

```
#include <stdlib.h>
#include <stdio.h>

void nrerror(error_text)
```

```
char error_text[];
{
        void exit();

        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}



float *vector(nl,nh)
int nl,nh;
{
        float *v;

        v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

int *ivector(nl,nh)
int nl,nh;
{
        int *v;

        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");
        return v-nl;
}

double *dvector(nl,nh)
int nl,nh;
{
        double *v;

        v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl;
}



float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        float **m;

        m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
                if (!m[i]) nrerror("allocation failure 2 in matrix()");
                m[i] -= ncl;
        }
        return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        double **m;
```

```
        m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
        if (!m) nrerror("allocation failure 1 in dmatrix()");
        m -= nrl;

        for(i=nrl;i≤nrh;i++) {
                m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
                if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
                m[i] -= ncl;
        }
        return m;
}

int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i,**m;

        m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
        if (!m) nrerror("allocation failure 1 in imatrix()");
        m -= nrl;

        for(i=nrl;i≤nrh;i++) {
                m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
                if (!m[i]) nrerror("allocation failure 2 in imatrix()");
                m[i] -= ncl;
        }
        return m;
}



float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
        int i,j;
        float **m;

        m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure in submatrix()");
        m -= newrl;

        for(i=oldrl,j=newrl;i≤oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

        return m;
}



void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
        free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
        free((char*) (v+nl));
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
        free((char*) (v+nl));
}
```

```
void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i≥nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i≥nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i≥nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}



void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}



float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int nrl,nrh,ncl,nch;
{
        int i,j,nrow,ncol;
        float **m;

        nrow=nrh-nrl+1;
        ncol=nch-ncl+1;
        m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
        if (!m) nrerror("allocation failure in convert_matrix()");
        m -= nrl;
        for(i=0,j=nrl;i≤nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
        return m;
}



void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}
```

stats.h

```
extern int statPenaltyNeg; /* number of negative radius penalties */
extern int statEval; /* number of check_guess calls */

extern void printStats();
extern void initStats();
```

stats.c

```
#include "stats.h"
#include <stdio.h>

int statPenaltyNeg;
int statEval;

void
initStats(){
    statPenaltyNeg=0;
    statEval=0;
}

void
printStats(f)
    FILE *f;
{
    fprintf(f,"\nStatistics:\n-----------\n");
    fprintf(f,"PenaltyNeg:  %d\n",statPenaltyNeg);
    fprintf(f,"Eval:  %d\n", statEval);
    fprintf(f,"-----------\n\n");
}
```

נספח ד

תוכנית שחזור לפי האלגוריתם של גרדנר

תוכניות אלו, שתוארו בסעיף 3.3, נכתבו בשפת $ANSI - C$ והורצו על מערכת Unix (Linux על מחשב PC).

ראה גם את amoeba.c (אלגוריתם המינימיזציה) מהנספח הקודם, המשמש גם הוא בתוכנית זו (אך אין צורך לחזור על הדפסתו). כמו כן, xray.[ch], layered_polygon.[ch], ו־xray.[ch] מאותו נספח משמשים בהגדרת ההטלות בבעית הדוגמה.

gardner.c

```
/* Gardner's reconstruction algorithm for a convex set, given 4 raysets
   (X rays in 4 mutually nonparallel directions).
   The algorithm is described in
        Richard J. Gardner, "Geometric Tomography", Encyclopedia of
        Mathematics and its Applications vol. 58, Cambridge University Press,
        1995
   in pages 48-49.
   The implementation is described in my thesis.
*/

/* TODO: in the present version, the interpolation of the function is done
   by a single spline. This is excelent for finding the first 3 points, but
   for determining subsequent points the relatively-large inaccuracy of the
   spline approximation near a point of derivative discontinuity (i.e., the
   point where the xrays start covering the body) causes substantial errors
   in some ray lengths when the number of rays is not very big. These errors
   then cause WRONG decisions in the direction of res (see below) which causes
   completely wrong points to be generated... Maybe that last problem can
   be solved.
   But the most thorough solution is to use a piecewise-spline approximation,
   or more easily: figure out (by exrapolation) the exact points where the
   function stops being 0 at both ends, and in between do a spline
   approximation with a given (extrapolated) derivative at both ends.
   This problem can also be circumvented by increasing ALPHA_EPS.
   See discussion in my thesis.
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "layered_polygon.h"
#include "xray.h"
#include "nrutil.h"
#include "spline.h"
#include "romberg.h"
#include "ssp.h"
```

```
/* ALPHA_EPS: angle epsilon - two points with an angle of less then ALPHA_EPS
   between them are considered "too close", and the new point is not added
   to the polygon (see discussion in my thesis). The default value is 5e-2,
   and it can be changed with a @epsilon option in the problem file.

   note that my experience shows that the accuracy of the 3 points we find
   in the first step of the algorithm is above alpha_eps=3e-4, so 1e-3 is a
   reasonable ALPHA_EPS. ALPHA_EPS of 5e-2 generates much better results
   when considering the bad interpolation near a derivative discontinuity of
   small number of rays, and the erronous decision of res sign.
*/

double ALPHA_EPS=5e-2;

/* problem file format:
          # some comments
          # and more comments
          @epsilon 2e-5 <- optional line
          polygs/polygontocheck
          0 <- definition of guess polygon
          50 0.1 1.0 -3 3 <- rayset
          50 1.0 0.2 -3 3 <- another rayset (more until EOF)

  for gardner.c, the guess polygon (used by minverse) is useless, hence the
  lone 0 on th line, and 4 raysets must be supplied.
*/


int loadProblem(prs,plp,fn)
RaySet prs[4];
LPolygon *plp;
char *fn;

{
    FILE *fp;
    int c,i,nrs;
    double g;
    char s[80];
    struct {
        int nrays;
        double dx,dy, c1,c2;
    } def[10];

    if((fp=fopen(fn,"r"))==NULL){
        perror(fn);
        return 0;
    }
    /* swallow comments in beginning */
    fprintf(stderr,"---------------------------------------- reading %s...\n",fn);
    while((c=getc(fp))=='#'||c=='@'){
        if(c=='#'){
            while((c=getc(fp))≠'\n' && c≠EOF) putc(c,stderr);
            putc('\n',stderr);
        } else { /* '@' options */
            if(!fgets(s,sizeof(s),fp)) goto FAIL;
            if(!strncmp(s,"epsilon ",8)) {
                if(sscanf(s,"epsilon %lg",&ALPHA_EPS)≠1) goto FAIL;
                fprintf(stderr,"using epsilon=%g\n",ALPHA_EPS);
            } else {
                fprintf(stderr,"unknown option line @%s\n",s);
                goto FAIL;
            }
        }
    }
    if(c==EOF) goto FAIL; else ungetc(c,fp);
    fprintf(stderr,"---------------------------------------- reading done.\n",fn);
```

```
    /* read polygon file name */
    fscanf(fp,"%s\n",s);
    if(!loadLPolygon(plp,s)) goto FAIL;
    fprintf(stderr, "Polygon loaded:  %s.\n",s);

    /* read and ignore guess polygon size (which is irrelevant in gardner,
       and is a relic from the minverse program)
    */
    if(fscanf(fp,"%d",&nrs)==1){
        if(nrs>0)
            for(i=0;i<nrs;i++)
                if(fscanf(fp,"%d %lg",&c,&g)≠2) goto FAIL;
    } else goto FAIL;

    /* read rayset definitions */
    nrs=0;
    while(fscanf(fp,"%d %lg %lg %lg %lg\n", &(def[nrs].nrays),
          &(def[nrs].dx), &(def[nrs].dy), &(def[nrs].c1), &(def[nrs].c2))
          == 5)
        nrs++;
    fprintf(stderr, "%d raysets found.\n",nrs);

    if(plp→npolygons≠1){
        fprintf(stderr,
            "Exactly one polygon is required by gardner's algorithm\n");
        goto FAIL;
    }
    if(nrs≠4){
        fprintf(stderr,
             "Exactly 4 raysets are required by gardner's algorithm\n");
        goto FAIL;
    }

    for(i=0;i<nrs;i++){
        NewRaySet(prs[i], def[i].nrays);
        eqdRays(prs[i].rays, def[i].nrays, def[i].dx, def[i].dy,
                def[i].c1+1e-10, def[i].c2+1e-10);
    }

    fclose(fp);
    return 1;

 FAIL:
    fclose(fp);
    return 0;
}

static double rs_interpolate_1(double), rs_interpolate_2(double),
    rs_interpolate_3(double), rs_interpolate_4(double);

/* global variables (I could have used the userdata method as in minverse
   instead).
*/
static RaySet rs[4];
static struct splinedata *rs_sd[4]; /* spline data built on rayset */
static double (*rs_interpolate[4])(double) =
{ /* interpolating functions */
    rs_interpolate_1, rs_interpolate_2, rs_interpolate_3, rs_interpolate_4
};
#define RS_SD_XMIN(m) (rs_sd[(m)]→x[0])
#define RS_SD_XMAX(m) (rs_sd[(m)]→x[rs_sd[(m)]→n-1])
static int ncount=0;
static int dodraw=0;
static double bestresult=1e10;
static Point besttri[3];

static double rs_interpolate_1(double x){
    return interpolate_spline(rs_sd[0],x);
```

```
}
static double rs_interpolate_2(double x){
    return interpolate_spline(rs_sd[1],x);
}
static double rs_interpolate_3(double x){
    return interpolate_spline(rs_sd[2],x);
}
static double rs_interpolate_4(double x){
    return interpolate_spline(rs_sd[3],x);
}


main(argc, argv)
    int argc;
    char *argv[];
{
    LPolygon lp;

    double dx,dy,ox,oy;
    double d,weight,thef();
    int i,m;
    FILE *fp;
    long seed,time();

    if(argc≠2){
      fprintf(stderr, "Usage:  %s problem-file\n", argv[0]);
      exit(1);
    }
    if(!loadProblem(rs, &lp, argv[1])){
      fprintf(stderr, "Failed loading problem file %s\n",argv[1]);
      exit(2);
    }

    outputLPolygon(lp,"out/gp%d.dat",1);
    xrayset(lp,rs[0]);
    xrayset(lp,rs[1]);
    xrayset(lp,rs[2]);
    xrayset(lp,rs[3]);

    /* show the configuration of the problem, to be viewed by view5 */
    outputLPolygon(lp,"out/main5-polyg",1);
    debug_xray=fopen("out/main5-rays","w");
    xrayset(lp,rs[0]);
    xrayset(lp,rs[1]);
    xrayset(lp,rs[2]);
    xrayset(lp,rs[3]);
    fclose(debug_xray);
    debug_xray=(FILE *)0;

#if 1
    /* debugging: show mass - stupid integration */
    {
        int i,j; double mass,nn,np,nx,ny;
        for(i=0;i<4;i++){
            nx= -rs[i].rays[0].dy; ny=rs[i].rays[0].dx;
            mass=0;
            for(j=0;j<rs[i].nrays-1;j++){
                nn=rs[i].rays[j].ox*nx+rs[i].rays[j].oy*ny;
                np=rs[i].rays[j+1].ox*nx+rs[i].rays[j+1].oy*ny;
                mass+=rs[i].rays[j].result*(np-nn);
            }
            printf("MASS from direction %d:  %g\n",i,mass);
        }
    }
#endif
    gardner(lp,rs);
```

```
    exit(0);
}


/* linereptotri takes a "lines representation" - three lines defined by
   their slope (dx,dy), and their distance r from the line of that slope
   at origin (0,0).
   The intersections of the three lines defines a triangle (3 points (x,y))
   which we'll return.

   linereprtorei guarantees that the returned point i is opposite the
   line i.
*/
void linereptotri(const double dx[3], const double dy[3],
                  const double r[3],
                  Point p[3])
{
    double nx[3], ny[3];
    int i;

    /* these vectors are normalized, so it is very easy to find the
       orthogonal */
    for(i=0;i<3;i++){
        nx[i]= -dy[i]; ny[i]= dx[i];
    }
    /* now, we'll be working in 3 coordinate systems: for each set we have
       it's direction line (dx,dy), and the orthogonal coordinate in the
       directiond of (nx,ny).
    */
    /* nn(i,j) is inner product of n(i) with n(j). These are actually
       independent on x1,x2,x3 so they can be in theory calculated only once.
       alpha(i,j) is defined so that the intersection of the lines i,j
       is o+r[i]n[i]+alpha(i,j)d[i]
    */
#define nn(i,j) (nx[i]*nx[j] + ny[i]*ny[j])
#define dn(i,j) (dx[i]*nx[j] + dy[i]*ny[j])
#define alpha(i,j) ((r[j]-r[i]*nn(i,j))/dn(i,j))

    /* first find the actual (x,y) coordinates of the three points 0 1 and 2 */
    /* note: the points are numbered so that the triangle edge of direction i
       is opposite to the vertex p[i] */
#define interx(i,j) (r[i]*nx[i] + alpha(i,j)*dx[i])
#define intery(i,j) (r[i]*ny[i] + alpha(i,j)*dy[i])
    p[2].x= interx(0,1);
    p[2].y= intery(0,1);

    p[1].x= interx(0,2);
    p[1].y= intery(0,2);

    p[0].x= interx(1,2);
    p[0].y= intery(1,2);
}


/* thef - as defined in Gardner, page 48 */
/* we assume each rayset has a constant direction (dx,dy), but we do
   not assume that rays are equidistant.
*/

double thef(thex)
        double thex[4];
{
    /* we need this because thef expects an array to start in 0, but
       the meaninful data is in positions 1..3.
    */
    double *x=thex+1;
    double dx[3], dy[3], nx[3], ny[3];
    Point p[3];
```

```
    double result;
    int i;

    /* We'll be working in 3 coordinate systems: for each set we have
       its direction line (dx,dy), and the orthogonal coordinate in the
       directiond of (nx,ny).
       First, find the directions of the 3 raysets. We assume, of course,
       that each rayset is in only one direction.
    */
    for(i=0;i<3;i++){
        dx[i]=rs[i].rays[0].dx; dy[i]=rs[i].rays[0].dy;
    }

    /* these vectors are normalized, so it is very easy to find the
       orthogonal */
    for(i=0;i<3;i++){
        nx[i]= -dy[i]; ny[i]= dx[i];
    }

    /* find the three points of intersection of the given lines, which
       defines a triangle - and then calculate the area of that triangle
       (this is the first part of thef).
    */
    linereptotri(dx,dy,x, p);

#define ABS(x) ((x)<0 ? -(x) : (x))
    result=0.5*ABS( (p[1].x*p[2].y-p[1].y*p[2].x)
                    -(p[0].x*p[2].y-p[0].y*p[2].x)
                    +(p[0].x*p[1].y-p[0].y*p[1].x) );

    /* now add the second (sum part) of thef */

    for(i=0;i<3;i++){
        /* find area of (SuiK)i (see my thesis). We have integrate the xrays
           from x[i] to infinity or -infinity, depending on where the triangle
           is (we integrate the half line which does not contain the triangle).

           Note that the accuracy of the integration is extremly important,
           as simple integration (such as the trapezoid rule) requires a
           huge number of xrays to give accurate results. Our solution is
           to make a cubic-spline approximation function of the known xrays
           and to integrate this function using a very accurate romberg
           integration algorithm (note that we could have even integrated
           this function analytically by using the spline representation,
           but using the general-purpose romberg integration is easier).
        */
        /* point p[i] is the triangle point which is opposite to the triangle
           edge i. That edge has the n[i] coordinate x[i], and p[i] has
           the n[i] coordinate npi which we will now find: */
        double npi=p[i].x*nx[i] + p[i].y*ny[i];

        if(x[i]>npi){
            /* order is RS_SD_XMIN(i) < npi < x[i] < RS_SD_XMAX(i)
               we should integrate from x[i] to RS_SD_XMAX(i)
            */
            result += qromb(rs_interpolate[i],x[i],RS_SD_XMAX(i));
        } else {
            /* order is RS_SD_XMIN(i) < x[i] < npi < RS_SD_XMAX(i)
               we should integrate from RS_SD_XMIN(i) to x[i].
            */
            result += qromb(rs_interpolate[i],RS_SD_XMIN(i),x[i]);
        }
    }
    printf(">> [%g %g %g]:  %g\n",x[0],x[1],x[2],result);

    if(result<bestresult){
        besttri[0]=p[0];
        besttri[1]=p[1];
```

```
        besttri[2]=p[2];
        bestresult=result;
    }
    return result;
}

gardner(LPolygon lp, RaySet rs[4])
{
    int i,j,m;
    int nfunk,count;
    double **p, *y;
    double cx, cy;
    struct ssp *V;
    struct ssp_node **flagged;
    int nflagged;
    double dx[4],dy[4],nx[4],ny[4];

    /*** STAGE 1: preperations ***/
    /* prepare a spline approximation of the rays in each direction
       note that we ASSUME that each rayset is of a constant direction,
       but we do not necessarily assume that the rays are equidistant.
    */
    for(m=0;m<4;m++){
        double *x,*y; /* note: y hides y from the previous block */
        int n;
        n=rs[m].nrays; /* number of rays in this direction */
        x=(double *)malloc(sizeof(double)*n);
        y=(double *)malloc(sizeof(double)*n);
        dx[m]=rs[m].rays[0].dx;
        dy[m]=rs[m].rays[0].dy;
        /* these vectors are normalized, so it is very easy to find the
           orthogonal */
        nx[m]= -dy[m]; ny[m]= dx[m];
        for(i=0;i<n;i++){
            x[i]=rs[m].rays[i].ox*nx[m] + rs[m].rays[i].oy*ny[m];
            y[i]=rs[m].rays[i].result;
        }
        /* make sure that the first and last y's are zero (otherwise, our
           x rays failed cover the entire body!) and that x's are in
           increasing order (if not, we'll have to switch their orders -
           let's hope they are
        */
        if(fabs(y[0])>1e-6 || fabs(y[n-1])>1e-6){
            fprintf(stderr,"gardner():  illegal y's\n");
            exit(1);
        }
        for(i=1;i<n;i++){
            if(x[i]≤x[i-1]){
                fprintf(stderr,"gardner():  non-increasing x's\n");
                exit(1);
            }
        }
        /* prepare spline, with 0 first derivatives at both endpoints (since
           it is expected for the projections to be a constant (zero) there).
        */
        rs_sd[m]=prepare_spline(x,y,n,0.0,0.0);
        free(y);
        free(x);
    }
    /* debugging: print the mass using the 4 directions */
    for(m=0;m<4;m++){
        printf("mass using spline integration on direction %d:  %g\n",m,
                qromb(rs_interpolate[m],RS_SD_XMIN(m),RS_SD_XMAX(m))
                );

    }
#if 0
    /* debugging: print spline interpolations to file */
```

```
    for(m=0;m<4;m++){
        char s[100];
        double d;
        FILE *fp;
        sprintf(s,"zz%d.data",m);
        fp=fopen(s,"w");
        for(i=0;i<rs_sd[m]→n;i++){
            fprintf(fp,"%g %g\n",rs_sd[m]→x[i],rs_sd[m]→y[i]);
        }
        fclose(fp);
        sprintf(s,"zz%d.interp",m);
        fp=fopen(s,"w");
        for(d=RS_SD_XMIN(m);d<RS_SD_XMAX(m);d+=0.01){
            fprintf(fp,"%g %g\n",d,(rs_interpolate[m])(d));
        }
        fclose(fp);
    }
    exit(0);
#endif

#define ndim 3
    p=dmatrix(1,ndim+1,1,ndim);
    y=dvector(1,ndim+1);
    p[1][1]=0; p[1][2]=0; p[1][3]=0;

    /*** STAGE 2: find the inscribed triangle ***/
    for(i=2;i≤ndim+1;i++)
        for(j=1;j≤ndim;j++)
            p[i][j]=p[1][j]+(j==(i-1))*1.0;


    fprintf(stderr,"checking initial %d guesses:\n",ndim+1);
    for(i=1;i≤ndim+1;i++)
        y[i]=thef(p[i]);
    fprintf(stderr,"done.\n");

    count=1000;
    amoeba(p,y,ndim,1e-6,thef,&nfunk,(void *)0,&count);
    if(count)
        printf("Amoeba converged after %d iterations\n",nfunk);
    else
        printf("Amoeba did not converge...\n");

    /* clean up after minimization algorithm */
    free_dmatrix(p,1,ndim+1,1,ndim);
    free_dvector(y,1,ndim+1);

    /* output the best triangle to a file */
    {
        FILE *bestf;
        printf("Outputting best triangle...\n");
        bestf=fopen("out/best-tri","w");
        fprintf(bestf,"#%d:  %g\n",ncount,bestresult);
        fprintf(bestf,"%g %g\n%g %g\n%g %g\n%g %g\n\n",
                besttri[0].x,besttri[0].y,besttri[1].x,besttri[1].y,
                besttri[2].x,besttri[2].y,besttri[0].x,besttri[0].y);
        fclose(bestf);
    }

    /*** STAGE 3: find more points ***/

    /* start by putting the 3 points we found so far in a "sorted star
       polygon" data structure, whose center is the center of the found
       triangle (that center is necessarily inside the polygon which
       we assume is convex).
    */
    cx=cy=0;
    for(i=0;i<3;i++){
```

```
        cx+=besttri[i].x;
        cy+=besttri[i].y;
    }
    cx/=3; cy/=3;

    V=new_ssp(cx,cy);
    for(i=0;i<3;i++){
        add_ssp_node(V,besttri[i].x,besttri[i].y,0.0);
    }

    debug_show_ssp(V);

    count=0;
    while(1){
        double c,res,newx,newy;
        struct ssp_node *node;
        int nadded;

        count++;
        nadded=0;
        fprintf(stderr,"iteration %d:\n",count);
        flagged=ssp_findflagged(V,&nflagged);
        for(i=0;i<nflagged;i++){
            fprintf(stderr,"processing flagged point %g %g\n",flagged[i]→x,flagged[i]→y);
            for(m=0;m<4;m++){
                flagged[i]→flag=0;
                /* if the direction m from our point i does not go into the
                   convex polygon V, we skip it. We check if the direction
                   is outside the angle made by the point and its neighbors.
                   Note that we check >= eps instead of >= 0 to illiminate
                   appearance of double points just because of truncation
                   errors. TODO: calibrate eps and add units to it.
                */
                if(((flagged[i]→next→x-flagged[i]→x)*nx[m]+
                     (flagged[i]→next→y-flagged[i]→y)*ny[m]) *
                   ((flagged[i]→prev→x-flagged[i]→x)*nx[m]+
                     (flagged[i]→prev→y-flagged[i]→y)*ny[m]) ≥ -1e-6)
                     continue;
                /* Find the coordinate of point i normal to direction m, and
                   the interpolated xray value there. Note that since we have
                   4 fixed directions, we could have saved their 4 coordinates
                   and interpolated values in the node, but the speedup will
                   be small (if any) because our flagging system usually
                   prevents us from having to recalculate a node again many
                   times.
                */
                c=flagged[i]→x*nx[m] + flagged[i]→y*ny[m];
                res=(rs_interpolate[m])(c);
                if(res<0){
                    /* I think there's a problem with the spline approximating
                       the ray function badly around the derivative
                       discontinuity (i.e., at the end of the body). res<0
                       is only part of the problem... this fix doesn't seem
                       to improve anything... */
                    fprintf(stderr,"WARNING: res=%g\n",res);
                    res=0;
                }
                /* adding plus or minus res (times d[m]) to the point will
                   give us the new point. Choosing plus or minus depends
                   whether plus or minus d[m] is inside the angle (we already
                   know one of them is, because of our check above).
                   Because this is a convex body, we know that the part of
                   the angle that is <180 degrees is the part which is
                   inside the body.
                */
                /* we know (see ssp.h) that if the orientation field of an
                   SSP is +1 when the inside of the polygon is to the
                   right of the (node TO node->next) edge. (or to the left
```

```
                    for orientation -1).
                    Note that instead of taking a normal to the edge, it's
                    easier to use the precalculated normal of the m direction.
                */
                if( (flagged[i]→next→x - flagged[i]→x)*nx[m] +
                     (flagged[i]→next→y - flagged[i]→y)*ny[m] > 0)
                     res= V→orientation * res;
                else
                     res= -V→orientation * res;

                newx=flagged[i]→x + res*dx[m];
                newy=flagged[i]→y + res*dy[m];

                printf("new point:  %g %g\n", newx,newy);

                /* add the new point, and flag the points neighboring to
                    this new points (because their angle changes and so
                    suddenly one of the directions can succeed of being
                    in that angle
                */
                node=add_ssp_node(V,newx,newy,ALPHA_EPS);
                if(node){
                     /* note that node may be 0 if there's no need to add
                         a new node! */
                     node→next→flag=1;
                     node→prev→flag=1;
                     nadded++;
                } else {
                     fprintf(stderr, "ignored same new point\n");
                }
            }
        }
        free(flagged);
        if(nadded){
            fprintf(stderr,"added %d new points in this iteration\n",nadded);
        } else {
            FILE *fp;
            fp=fopen("out/gardner-polyg","w");
            fprintf(stderr,"no new points to be added (up to precision epsilon=%g)\nTotal:  %d points in
%d iterations.\n",ALPHA_EPS,V→n,count);
            output_ssp(V,fp);
            fclose(fp);
            return;
        }
        if(count==30){ /* just for debugging */
            FILE *fp;
            fp=fopen("out/gardner-polyg","w");
            output_ssp(V,fp);
            fclose(fp);
            fprintf(stderr,"total found before debugging stop:  %d points\n",
                     V→n);
            return;
        }
    }

    /* final clean up */
    /* TODO: free V with free_ssp */
    for(m=0;m<4;m++)
        free_spline(rs_sd[m]);
}
```

ssp.h

```
#ifndef INCLUDED_SSP_H
#define INCLUDED_SSP_H
/* Sorted Star Polygon datastructure - used for Gardner's algorithm */
```

```
/* currently the SSP datastructure is kept as a doubly linked list.
   In the future some more efficient method (e.g., binary tree) for keeping a
   sorted list might be adopted. We need an efficient "add element and keep
   list sorted" operation and an efficient "find all flaged element" operation
   - currently this is not done very efficiently. We also need efficient
   "next node" and "previous node" (in the circular sorted list) operation.
*/
struct ssp_node {
    double x,y;
    double alpha;
    struct ssp_node *next, *prev;
    int flag; /* 1 if this node should be revisited in the next step */
};
struct ssp {
    int n; /* not really important - can be found by traversing the list */
    struct ssp_node *head;
    double cx,cy; /* center for star polygon */
    /* orientation is 1 if following the polygon in order (from node to
       node->next), the inside of the polygon is on the right.
       orientation is -1 if the inside is on the left.
       In other words, orientation +1 means a *clockwise* polygon around the
       inside.
       The orientation is automatically determined once the polygon reaches
       3 points (it is assumed that once added, points are not changed).
       For polygons with less than 3 points, the orientation is meaningless,
       and is set to 0.
    */
    int orientation;
};

struct ssp *new_ssp(double cx, double cy);
struct ssp_node *add_ssp_node(struct ssp *p, double x, double y,
                              double alpha_eps);
void debug_show_ssp(struct ssp *p);
struct ssp_node **ssp_findflagged(const struct ssp *p, int *num);
void output_ssp(struct ssp *p, FILE *fp);

#endif
```

ssp.c

```
/* Sorted Star Polygon datastructure - used for Gardner's algorithm */

/* currently the SSP datastructure is kept as a linked list. In the future
   some more efficient method (e.g., binary tree) for keeping a sorted list
   might be adopted. We need an efficient "add element and keep list sorted"
   operation and an efficient "find all flaged element" operation - currently
   this is not done very efficiently.
   NOTE: this is a circular list connectivity (the prev and next pointers),
   but the sorting of alpha is not circular, i.e., the node pointed by head
   has the lowest alpha.
*/

#include <math.h>
#include <stdlib.h>
#include <stdio.h>

#include "ssp.h"

struct ssp *
new_ssp(double cx, double cy)
{
    struct ssp *p;
    p=(struct ssp *)malloc(sizeof(struct ssp));
    p->n=0;
    p->head=0;
    p->cx=cx;
```

```
    p→cy=cy;
    p→orientation=0;
    return p;
}

struct ssp_node *
add_ssp_node(struct ssp *p, double x, double y, double alpha_eps)
{
    double alpha;
    struct ssp_node *newnode;
    struct ssp_node **nodep;

    alpha=atan2(y-p→cy, x-p→cx);
    fprintf(stderr,"adding %g %g (%g)\n",x,y,alpha);
    /* prepare a new node */
    newnode=(struct ssp_node *) malloc(sizeof(struct ssp_node));
    newnode→x=x;
    newnode→y=y;
    newnode→alpha=alpha;
    newnode→flag=1;
    /* find where to put this node */
    if(!p→head){
        /* the first and only node */
        newnode→next=newnode→prev=newnode;
        p→head=newnode;
        goto NODEADDED;
    } else {
        nodep=&(p→head);
        do {
            if(alpha<(*nodep)→alpha){
                if(fabs(alpha-(*nodep)→alpha)<alpha_eps ||
                    fabs(alpha-(*nodep)→prev→alpha)<alpha_eps){
                    /* we already have a node with this alpha, so we don't
                       need to add it.
                       TODO: make sure that the x,y of this node is also close
                       to the new ones!!
                    */
                    free(newnode);
                    return 0;
                }
                /* insert newnode just before this node */
                newnode→prev=(*nodep)→prev;
                newnode→next=(*nodep);
                (*nodep)→prev=newnode;
                if( (*nodep)→next==(*nodep) )/*(*nodep)->next usually */
                    (*nodep)→next=newnode; /*unchanged */
                newnode→prev→next=newnode;
                (*nodep)=newnode; /* same as previous line, except for head */
                goto NODEADDED;
            }
            nodep=&((*nodep)→next);
        } while(*nodep≠p→head);
        /* we're still here (did not return, so we need to put this node at
           the end. */
        if(fabs(alpha-p→head→alpha)<alpha_eps ||
            fabs(alpha-p→head→prev→alpha)<alpha_eps){
            /* we already have a node with this alpha, so we don't
               need to add it.
               TODO: make sure that the x,y of this node is also close
               to the new ones!!
            */
            free(newnode);
            return 0;
        }
        newnode→prev=p→head→prev;
        p→head→prev→next=newnode;
        newnode→next=p→head;
        p→head→prev=newnode;
```

```
        goto NODEADDED;
    }
    /* there's no way we can get to this line - all case have "return"
       or goto NODEADDED */
 NODEADDED:
    p→n++;
    /* if we reached 3 points, it's time to define the orientation.
       See comment in ssp.h for the definition of the orientation we're
       about to calculate here.
    */
#define P0 (*(p→head))
#define P1 (*(p→head→next))
#define P2 (*(p→head→next→next))

    if(p→n==3){
        /* there's only need to define the orientation once, when we reach
           3 points. Afterwards, we assume that the 3 points are not changed,
           and that other points are added in correct, sorted, order.
        */
        if((P1.x*P2.y-P1.y*P2.x) - (P0.x*P2.y-P0.y*P2.x) +
           (P0.x*P1.y-P0.y*P1.x) < 0 )
            p→orientation= 1;
        else
            p→orientation= -1;
        fprintf(stderr,"ORIENTATION DEBUG:\n%g %g\n%g %g\n%g %g\norientation
%d\n\n",P0.x,P0.y,P1.x,P1.y,P2.x,P2.y,p→orientation);
    }
    return newnode;
}

void
debug_show_ssp(struct ssp *p)
{
    struct ssp_node *node;
    node=p→head;
    if(!node){
        fprintf(stderr,"No points in SSP.\n");
        return;
    }
    do {
        fprintf(stderr,"x=%g y=%g alpha=%g\n",node→x,node→y,node→alpha);
        node=node→next;
    } while (node≠p→head);
}

/* outputs an ssp for gnuplot (first point is output also at the end to
   close the loop
*/
void
output_ssp(struct ssp *p, FILE *fp)
{
    struct ssp_node *node;
    node=p→head;
    if(!node){
        fprintf(stderr,"No points in SSP.\n");
        return;
    }
    do {
        fprintf(fp,"%g %g\n",node→x,node→y);
        node=node→next;
    } while (node≠p→head);
    fprintf(fp,"%g %g\n",p→head→x,p→head→y);
}

/* ssp_findflagged makes a list of nodes that are currently flagged.
   The Gardner algorithm then processes each of these nodes, to create
   new nodes.
   Remember to free the returned vector!
```

```
*/

struct ssp_node
**ssp_findflagged(const struct ssp *p, int *num)
{
    int n;
    struct ssp_node *node;
    struct ssp_node **flaggednodes;
    /* count flagged nodes first. this is probably not the most efficient
       solution.
    */
    node=p→head;
    n=0;
    do {
        if(node→flag)
            n++;
        node=node→next;
    } while (node≠p→head);

    flaggednodes=(struct ssp_node **)malloc(n*sizeof(struct ssp_node *));

    node=p→head;
    n=0;
    do {
        if(node→flag){
            flaggednodes[n]=node;
            n++;
        }
        node=node→next;
    } while (node≠p→head);

    *num=n;
    return flaggednodes;
}
```

spline.h

```
#ifndef INCLUDED_SPLINE_H
#define INCLUDED_SPLINE_H

struct splinedata {
    int n;
    double *x;
    double *y;
    double *y2;
};

struct splinedata *prepare_spline(const double *x, const double *y, int n,
                                  double yp0, double ypnm);
double interpolate_spline(const struct splinedata *sd, double x);
void free_spline(struct splinedata *sd);

#endif /* INCLUDED_SPLINE_H */
```

spline.c

```
/* The following routines have been adapted from:
   "Numerical Recipes in C, The Art of Scientific Computing" second edition,
   by William H. Press, Saul A. Teukolsky, William T. Vettering, and Brian
   P. Flannery. Cambridge University Press, 1994.

   Note that changes have been made from the original source. In particular,
   arrays here start in 0, as expected in C programs.
*/
#include <stdlib.h>
```

```c
#include "spline.h"

/* prepare_spline(): given arrays x[0..n-1] and y[0..n-1] containing tabulated
   functions, i.e., y[i]=f(x[i]), with x[0]<x[1]<...<x[n-1], and given values
   yp0 and ypnm for the first derivative of the interpolating functions at
   points 0 and n-1, respectively, this routine returns an array y2[0..n-1]
   that contains the second derivatives of the interpolating functions at the
   the tabulated points x[i] - this information is used by the interpolation
   function interpolate_spline() (see below).
   If tp0 and/or ypnm are 1e30 or larger, the routine is signaled to set the
   corresponding boundary condition for a natural spline, with zero second
   derivative on that boundary.

   the returned splinedata structure is later used to interpolate the spline,
   and contains all the necessary information: x and y may be freed after
   the call to prepare_spline. splinedata must be eventually freed by the
   user, by calling free_spline().
*/

struct splinedata *prepare_spline(const double *x, const double *y, int n,
                                  double yp0, double ypnm)
{
    int i,k;
    double p, qnm, sig, unm, *u, *y2;
    struct splinedata *sd;

    u=(double *) malloc(sizeof(double)*(n-1));
    y2=(double *) malloc(sizeof(double)*n);
    if(yp0 > 0.99e30)
        y2[0]=u[0]=0.0;
    else {
        y2[0]=-0.5;
        u[0]=(3.0/(x[1]-x[0]))*((y[1]-y[0])/(x[1]-x[0])-yp0);
    }
    /* this is the decomposition loop of the tridiagonal algorithm. y2
       and u are used for temporary storage of the decomposed factors */
    for(i=1;i<n-1;i++){
        sig=(x[i]-x[i-1])/(x[i+1]-x[i-1]);
        p=sig*y2[i-1]+2.0;
        y2[i]=(sig-1.0)/p;
        u[i]=(y[i+1]-y[i])/(x[i+1]-x[i]) - (y[i]-y[i-1])/(x[i]-x[i-1]);
        u[i]=(6.0*u[i]/(x[i+1]-x[i-1])-sig*u[i-1])/p;
    }
    if(ypnm > 0.99e30)
        qnm=unm=0.0;
    else {
        qnm=0.5;
        unm=(3.0/(x[n-1]-x[n-2]))*(ypnm-(y[n-1]-y[n-2])/(x[n-1]-x[n-2]));
    }
    y2[n-1]=(unm-qnm*u[n-2])/(qnm*y2[n-2]+1.0);
    /* this is the backsubstitution loop of the tridiagonal algorithm */
    for(k=n-2;k>=0;k--)
        y2[k]=y2[k]*y2[k+1]+u[k];
    free(u);

    sd=(struct splinedata *)malloc(sizeof(struct splinedata));
    sd->x=(double *) malloc(sizeof(double)*n);
    sd->y=(double *) malloc(sizeof(double)*n);
    for(i=0;i<n;i++){
        sd->x[i]=x[i];
        sd->y[i]=y[i];
    }
    sd->y2=y2;
    sd->n=n;

    return sd;
}
```

```
void free_spline(struct splinedata *sd)
{
    free(sd→x);
    free(sd→y);
    free(sd→y2);
    free(sd);
}

double interpolate_spline(const struct splinedata *sd, double x)
{
    int klo,khi,k;
    double h,b,a;
    /* we will find the right place in the table by means of bisection.
       This is optimal if sequential calls to this routine are at random
       values of x. if sequential calls are in order, and closely spaced,
       one would do better to store previous values of klo and khi and
       test if they remain appropriate on the next call.
    */
    klo=0;
    khi=sd→n-1;
    while(khi-klo>1){
        k=(khi+klo)≫1;
        if(sd→x[k]>x) khi=k;
        else klo=k;
    }
    h=sd→x[khi]-sd→x[klo];
    if(h==0.0){
        printf("bad x input to interpolate_spline\n");
        abort();
    }
    a=(sd→x[khi]-x)/h;
    b=(x-sd→x[klo])/h;
    return a*sd→y[klo]+b*sd→y[khi]+((a*a*a-a)*sd→y2[klo]+(b*b*b-b)*sd→y2[khi])*(h*h)/6.0;
}
```

romberg.h

```
#ifndef INCLUDED_ROMBERG_H
#define INCLUDED_ROMBERG_H

double qromb(double (*func)(double), double a, double b);

#endif
```

romberg.c

```
/* The following routines have been adapted from:
   "Numerical Recipes in C, The Art of Scientific Computing" second edition,
   by William H. Press, Saul A. Teukolsky, William T. Vettering, and Brian
   P. Flannery. Cambridge University Press, 1994.
*/
/* NOTE: I left the 1-oriented arrays here because their use is only
   internal to these routines. */

/* Romberg Integration qromb():
   qromb() returns the integral of the function func from a to b.
   Integration is performed by Romberg's method of order 2K, where, e.g.,
   K=2 is Simpson's rule.
*/


#include <math.h>
#include <stdio.h>
#include <stdlib.h>
```

```c
#include "nrutil.h"

#include "romberg.h"

/* This routine computes the nth stage of refinement of an extended
   trapezoidal rule. func is input as a pointer to the function to be
   integrated between limits a and b, also input. When called with n=1,
   the routine returns the crudest estimate of \int_a^b f(x)dx.
   Subsequent calls with n=2,3,.. (in that sequential order) will improve
   the accuracy by adding 2^(n-2) additional interior points.
*/
static double
trapzd(double (*func)(double), double a, double b, int n)
{
    double x,tnm,sum,del;
    static double s;
    int it,j;

#define FUNC(x) ((*func)(x))
    if(n==1){
        return (s=0.5*(b-a)*(FUNC(a)+FUNC(b)));
    } else {
        for(it=1,j=1;j<n-1;j++) it <<= 1;
        tnm=it;
        del=(b-a)/tnm; /* this is the spacing of the points to be added */
        x=a+0.5*del;
        for(sum=0.0,j=1;j<=it;j++,x+=del) sum+=FUNC(x);
        s=0.5*(s+(b-a)*sum/tnm); /* this replaces s by its refined value. */
        return s;
    }
}

/* polint() is given arrays xa[1..n] and ya[1..n] and a value x. It returns
   a value y, and an error estimate dy. if P(x) is the polynomial of degree
   N-1 such that P(xa_i)=ya_i, i=1..n, then the returned value y=P(x).
*/
static void
polint(double xa[], double ya[], int n, double x, double *y, double *dy)
{
    int i,m,ns=1;
    double den,dif,dift,ho,hp,w;
    double *c, *d;

    dif=fabs(x-xa[1]);
    c=dvector(1,n);
    d=dvector(1,n);
    for(i=1;i<=n;i++){
        /* here we find the index ns of the closest table entry */
        if((dift=fabs(x-xa[i]))<dif){
            ns=i;
            dif=dift;
        }
        c[i]=ya[i]; /* and initialize the tableau of c's and d's. */
        d[i]=ya[i];
    }
    *y=ya[ns--]; /* this is the initial approximation to y. */
    for(m=1;m<n;m++){ /* for each column of the tableau */
        for(i=1;i<=n-m;i++){ /* we loop over the current c's and d's and update
                                them. */
            ho=xa[i]-x;
            hp=xa[i+m]-x;
            w=c[i+1]-d[i];
            if((den=ho-hp)==0.0){
                /* this error can occur only if two input xa's are (to
                   within roundoff) identical. */
                fprintf(stderr, "Error in routine polint.\n");
                exit(1);
            }
```

```
            den=w/den;
            d[i]=hp*den; /* here the c's and d's are updated */
            c[i]=ho*den;
        }
        /* after each column in the tableau is completed, we decide which
           correction, c or d, we want to add to our accumulating value of
           y, i.e., which path to take through the tableau - forking up or
           down. We do this in such a way as to take the most "straight line"
           route through the tableau to its apex, updating ns accordingly
           to keep track of where we are. This route keeps the partial
           approximations centered (insofar as possible) on the target x.
           The last dy is added is thus the error indication.
           */
        *y+=(*dy=(2*ns<(n-m)?  c[ns+1] : d[ns--]));
    }
    free_dvector(d,1,n);
    free_dvector(c,1,n);
}



/* EPS is the fractional accuracry desired, as determined by the extrapolation
   error estimate. JMAX limits the total number of steps. K is the number of
   points used in the extrapolation.
*/

#define EPS /*1e-6*/ 1e-9
#define JMAX 20
#define JMAXP (JMAX+1)
#define K 5

double qromb(double (*func)(double), double a, double b)
{
    double ss,dss;
    /* these store the successive trapezoidal approximations and their
       relative stepsizes */
    double s[JMAXP+1],h[JMAXP+1];
    int j;

    h[1]=1.0;
    for(j=1;j≤JMAX;j++){
        s[j]=trapzd(func,a,b,j);
        if(j≥K){
            polint(&h[j-K],&s[j-K],K,0.0,&ss,&dss);
            if(fabs(dss)≤EPS*fabs(ss)) return ss;
        }
        /* this is a key step: the factor is 0.25 even though the stepsize
           is decreased by only 0.5. This makes the extrapolation a
           polynomial in h^2 as allowed by equation (4.2.1), not just
           ap polynomial in h.
        */
        s[j+1]=s[j];
        h[j+1]=0.25*h[j];
    }
    fprintf(stderr,"too many step in routine qromb.\n");
    exit(1);
}
```

ביבליוגרפיה

[1] G. Bianchi and M. Longinetti. Reconstructing plane sets from projections. *Descrete Comput. Geom.*, 5:223–242, 1990.

[2] S. K. Chang. The reconstruction of binary matrices from their projections. *Coommun. ACM*, 14:21–24, 1971.

[3] Shi Kuo Chang and C. K. Chow. Automatic reconstruction of the heard chanmber from biplane cineangiogram. Technical Report RC 3681, IBM Research, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, November 4 1971.

[4] Shi-Kuo Chang and C. K. Chow. The reconstruction of three-dimensional objects from two orthogonal projections and its application to cardiac cineangiography. *IEEE Trans. Comput.*, 22:18–28, 1973.

[5] M. G. Darboux. Sur un problème de géométrie élémentaire. *Bull. Sci. Math*, 2:298–304, 1878.

[6] H. Edelsbrunner and S. S. Skiena. Probing convex polygons with x-rays. *SIAM J. Comp.*, 17:870–882, 1988.

[7] P. C. Fishburn, J. C. Lagarias, J. A. Reeds, and L. A. Shepp. Sets uniquely determined by projections on axes i. continuous case. *SIAM J. Appl. Math.*, 50(1):288–306, February 1990.

[8] R. J. Gardner. Symmetrals and x-rays of planar convex bodies. *Arch. Math.*, 41:183–189, 1983.

[9] R. J. Gardner. Sets determined by finitely many x-rays. *Geometriae Dedicata*, 43:1–16, 1992.

[10] R. J. Gardner. X-rays of polygons. *Discrete Comput. Geom.*, 7:281–293, 1992.

[11] R. J. Gardner. *Geometric Tomography*, volume 58 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1995.

[12] R. J. Gardner and P. McMullen. On hammer's x-ray problem. *J. London Math. Soc. (2)*, 21:171–175, 1980.

[13] Shay Gueron and Moshe Deutsch. Extensions of the abel transform to non-circular symmetries.

[14] Shay Gueron and Moshe Deutsch. A fast abel inversion algorithm. Technical Report CTC93TR135, Cornell Theory Center, Center for Applied Mathematics, May 1993.

[15] P.C. Hammer. Problem 2. In *Proceedings of Symposia in Pure Mathematics*, volume VII: Convexity, pages 498–499, Providence, RI, 1963. American Mathematical Society.

[16] B. K. P. Horn. *Robot Vision*. MIT Press/McGraw-Hill, New York, 1985.

[17] Myron Bernard Katz. *Questions of Uniqueness and Resolution in Reconstruction from Projections*, volume 26 of *Lecture Notes in Biomathematics*. Springer, 1978.

[18] A. Kuba and A. Volčič. Characterisation of measurable plane sets which are reconstructable from their two projections. *Inverse Problems*, 4:513–5–27, 1988.

[19] M. Longinetti. Some questions of stability in the reconstruction of plane convex bodies from projections. *Inverse Problems*, 1:87–97, 1985.

[20] M. Longinetti. An isoperimetric inequality for convex polygons and convex sets with the same symmetrals. *Geom. Dedicata*, 20:27–41, 1986.

[21] G. G. Lorentz. A problem of plane measure. *Amer. J. Math*, 71:417–426, 1949.

[22] J.A. Nelder and R. Mead. Computer journal. 7:308–313, 1965.

[23] William H. Press, William T. Vetterling, Saul A. Teukolsky, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, second edition edition, 1994.

[24] F. Riesz. Sur un inégalité intégrale. *Journal of the London Mathematical Society*, 5:162–168, 1930.

[25] H. L. Royden. *Real Analysis*. Macmillan, New York, 3rd edition, 1988.

[26] A. Volčič. Well-posedness of the Gardner-McMullen reconstruction problem. In *Proc. Conf. Measure Theory, Oberwolfach, 1983*, Lecture Notes in Mathematics 1089, pages 199–210, Berlin, 1984. Springer.

[27] A. Volčič. Ghost convex bodies. *Boll. Unione Mat. Italiana (6)*, 4-A:287–292, 1985.

[28] A. Volčič and T. Zamfirescu. Ghosts are scarce. *J. London Math. Soc. (2)*, 40:171–178, 1989.