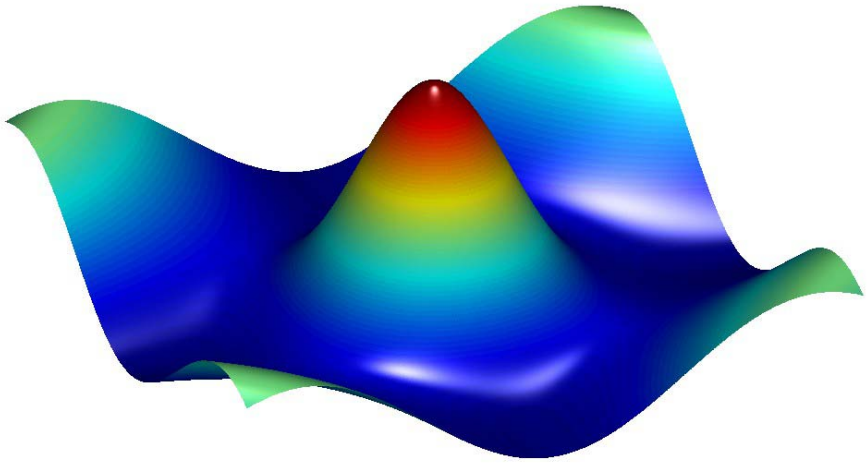
מבוא ל-Matlab



מאת

דורי פלג

גירסה 1.0 – דצמבר 2002

מבוא ומוטיבציה

שפת Matlab היא שפת תכנות מתקדמת לסטודנטים ומהנדסים המאפשרת עיבוד מידע, הדמיות והצגה חזותית מגוונת באמצעות מיעוט בשורות קוד ובסביבה המיועדת לניתוח תוצאות.
ההבדלים המרכזיים בין Matlab ושפות אחרות הם,

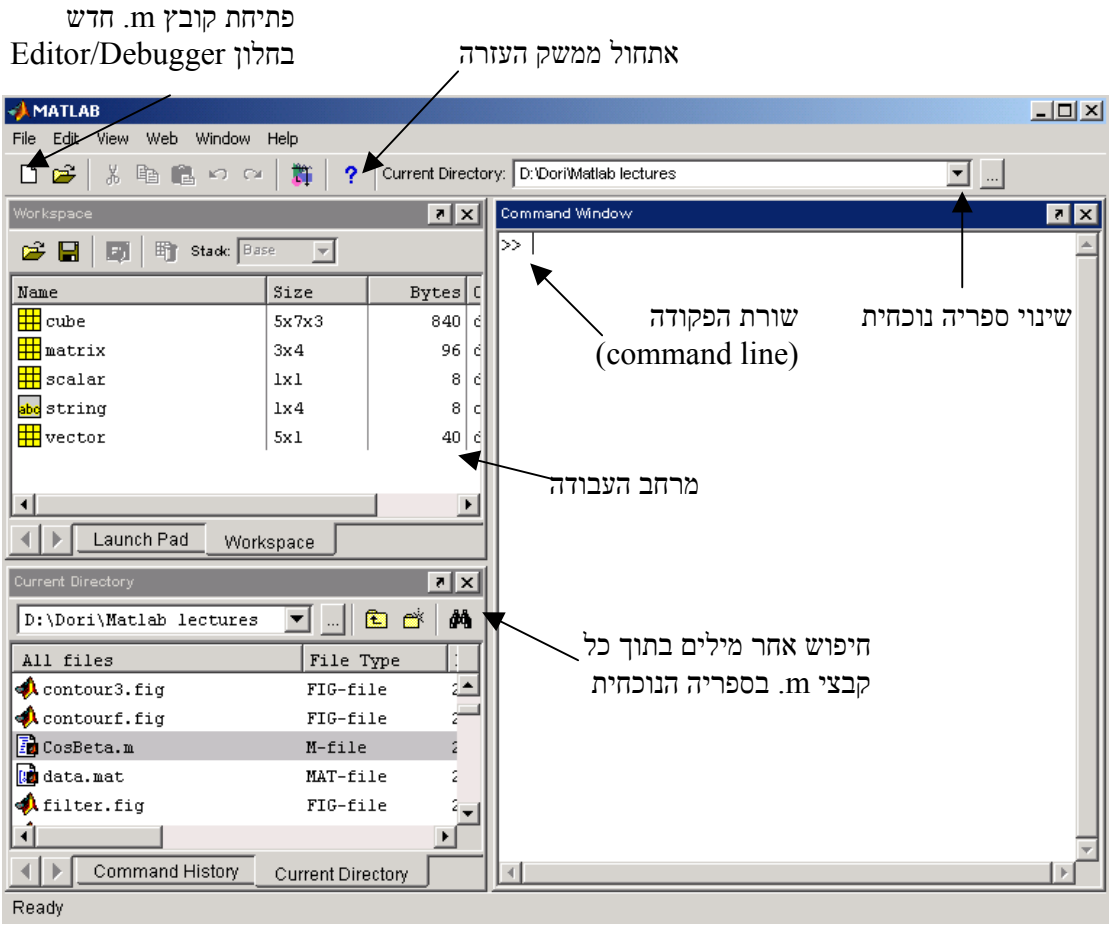
1. ב-Matlab המשתנים הבסיסיים הם וקטורים ומטריצות. למעשה, מקור השם Matlab הוא **Mat**rix **Lab**oratory.
2. מספר רב של פונקציות מובנות (built-in) החוסך תכנות רב מהמשתמש.
3. מאגר נרחב של פונקציות המיועדות לתחומים הנדסיים ספציפיים השמורים בספריות toolbox כמו עיבוד אותות, עיבוד תמונות, בקרה, סטטיסטיקה ואפילו מציאות מדומה.

חוברת זו איננה מתימרת להציג את כל ההיקף הרחב של שפת Matlab. מטרת חוברת זו היא רק להציג את הידע וההבנה הבסיסיים הנדרשים כדי לאפשר את השיטה האמיתית ללמידה ב-Matlab - הלימוד העצמי דרך מדריכי העזרה הנרחבים.

חוברת זו נכתבה עבור Matlab 6.

תוכן עניינים

1. הכרת סביבת Matlab



גרף 1.1: סביבת העבודה

1.1 פרוט החלונות ושימושם

חלון	הסבר
Command Window	הרצת פקודות ע"י הקלדתן ולחיצה על Enter. ניתן לקבל את הפקודה הקודמת ב-command line ע"י לחיצה על כפתור למעלה. מחיקת הרישום ב-Command Window מבוצעת באמצעות פונקצית clc.
Workspace	תצוגת מרחב העבודה המכילה את כל המשתנים שהוגדרו ומידע לגביהם.
Current Directory	פרוט כל הקבצים בספרייה הנוכחית. לחיצה כפולה על כפתור השמאלי של העכבר בחלון הנ"ל על הקבצים הבאים משיגה: קובץ m. - פתיחת הקובץ בחלון Editor/Debugger. קובץ mat. - טעינת המשתנים ל-Workspace. קובץ fig. - פתיחת חלון הגרפיקה.
Command History	תיעוד פקודות שהוקלדו ב-Command Window ואפשרות להעברתן באופן ידני ל-Command Window.
Launch Pad	גישה מהירה למסמכי עזרה, demos, ועזרים נוספים.

טבלה 1.1: פרוט החלונות ושימושם

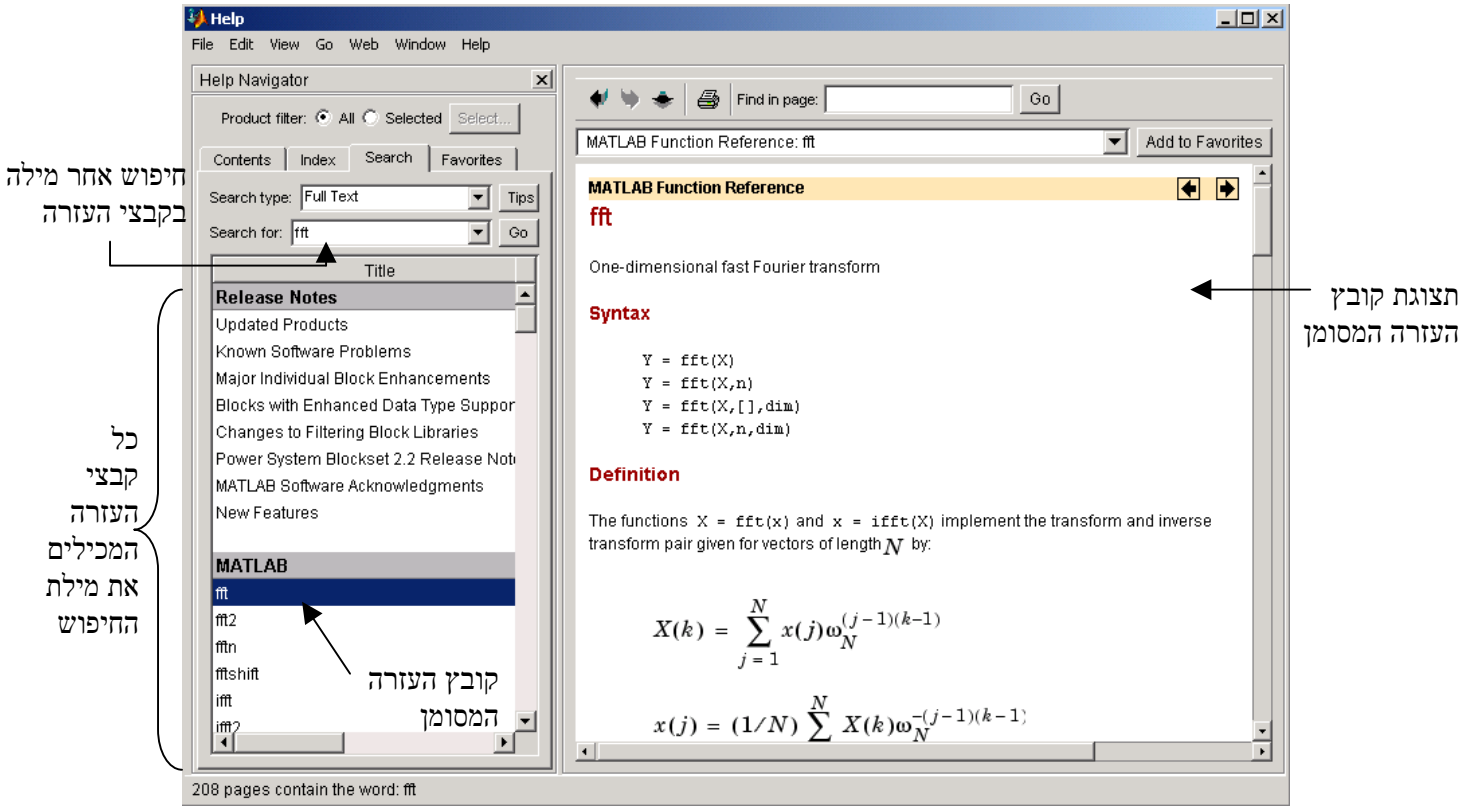
1.2 סוגי קבצי Matlab

ישנם מספר סוגים של קבצים ב-Matlab,

1. קבצים עם סיומת m.
   קבצים אלו מבצעים פקודות Matlab. עריכתם והרצתם מבוצעות דרך חלון Editor/Debugger.
   קיימים שני סוגי קבצי m.,
   א. script: קבצים אלו אינם מקבלים קלט או מחזירים פלט. הם פועלים על מידע הנמצא ב-Workspace. שקול להקלדת הפקודות ב-Command Window. הרצת הקובץ נעשית ע"י לחיצה על F5 (או ע"י לחיצה על כפתור החץ כלפי מטה).
   ב. function: קבצים אלו יכולים לקבל קלט ולהחזיר פלט. כל המשתנים הפנימיים הם לוקליים לפונקציה. כדי להגדיר קבצים כפונקציה משתמשים במילת מפתח function כפי שיוסבר בפרק 8.
2. קבצים עם סיומת mat.
   קבצים אלו מאכסנים משתנים וערכיהם.
3. קבצים עם סיומת fig.
   קבצים אלו מאכסנים גרפים.

1.3 שימוש ב-help של Matlab

ניתן לחפש עזרה לגבי נושאים ופונקציות באמצעות ממשק העזרה,



גרף 1.2: חלון help

קימות מספר דרכים כדי לקבל הסברים לגבי פונקציה,

שם הפונקציה ידוע	שם הפונקציה לא ידוע
כתיבת help ושם הפונקציה מציג ב- Command Window הסבר לגבי פעולת הפונקציה ואופן השימוש בה.	כתיבת lookfor ומילת מפתח מציג ב- Command Window רשימת כל הפונקציות שיש להן קשר למילת מפתח.
כתיבת doc ושם הפונקציה פותח את ממשק העזרה בפונקציה הנ"ל. לעומת אופציית help מכיל הסברים יותר מפורטים ובנוסף גם גרפים ונוסחאות.	חיפוש בממשק העזרה באמצעות אופציית search[1].

קיים מספר גדול של ספרי הדרכה לגבי שימוש ב-Matlab וב-toolbox-ים שלו. ספרים אלו נמצאים בגרסת html דרך אופציית contents בממשק העזרה או בפורמט pdf בחלון Launch Pad.
בנוסף קיים מאגר demos רחב המסבירים שימוש ב-Matlab וביישומים אפשריים של ה-toolbox-ים שלו.

1.4 כללים לקביעת שמות משתנים

ב-Matlab מוקצים עד ל-31 תווים לשמות משתנים, כאשר Matlab מתעלם מכל תו נוסף. כל שם חייב להתחיל באות. שם יכול לכלול אותיות, מספרים ותווי underscore '_'. אסור שיהיו רווחים בשם.
Matlab הוא case sensitive ולכן השם name ו-Name הם שמות שונים.
לעיתים רבות שמות הם שילוב של מספר מילים. קיימות שתי אסכולות פופולריות למתן שמות: underscore ושימוש באותיות גדולות. למשל max force תיכתב max_force או MaxForce בהתאם לטעם אישי.

1.5 תצוגה ב-Command Window

ניתן לשנות את התצוגה באמצעות פונקצית format. קיימות אופציות רבות אך נציג רק שלוש מהן,
1. format compact: ייצוג יותר דחוס הנפטר מרוב הרווחים.
2. format loose: ייצוג בעל מרווחים. למעשה מבטל את format compact.
3. format rat: ייצוג מספרים כמספרים רציונליים.

כדי לחזור לייצוג הסטנדרטי של 5 ספרות הדפיסו format short.
נזכור כי כל השינויים הנ"ל הם רק בתצוגה ואינם משפיעים על אופן שמירת המשתנים.

1.6 מספרים ב-Matlab

בנוסף לרישום העשרוני Matlab מכיר גם ברישום המדעי המשתמש באות e כדי לציין פקטור החזקה העשרונית. כלומר 2e5 הוא $2\cdot10^{5}$ ו- 6e-2 הוא $6\cdot10^{-2}$.
מספרים קומפלקסים מצוינים באמצעות האות i או j. לדוגמא 2+4i, 1-3j הם מספרים קומפלקסים חוקיים ב-Matlab.

[1] אפשרות זו מציגה רק פונקציות המופיעות ב-toolbox-ים, לעומת פונקצית lookfor המציגה גם פונקציות שלא שייכות ל-toolbox-ים.

קיימים מספרים קבועים וחשובים ב-Matlab,

pi	3.14159265...
i	Imaginary unit, $\sqrt{-1}$
j	Same as i
eps	Floating-point relative precision, 2^{-52}
realmin	Smallest floating-point number, 2^{-1022}
realmax	Largest floating-point number, $(2-\epsilon)2^{1023}$
Inf	Infinity
NaN	Not-a-number

הערך Inf מתקבל מחלוקת מספר באפס או בערך הגדול מ- realmax.
הערך NaN מתקבל מהחלוקה $\frac{0}{0}$ או $\frac{Inf}{Inf}$.

1.7 יציאה מ-Matlab

יציאה מסודרת מ-Matlab מבוצעת ע"י כתיבת quit ב-Command Window או ע"י לחיצה על Ctrl+Q.

2. הגדרת מטריצות

2.1 הגדרה ידנית

המשתנה הבסיסי ב-Matlab הוא מטריצה. ניתן לחשוב על וקטורים וסקלרים כמקרה פרטי של מטריצות.
מטריצה היא מערך מלבני של ערכים ומוגדרת באופן כללי,

$$A=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

כלומר A היא מטריצה $[m\times n]$.

הפעולה הבסיסית ביותר ב-Matlab היא יצירת מטריצה והצבת ערכים בתוכה. איברי המטריצה מתוחמים בתוך סוגריים מרובעות, סימן "," מפריד בין איברי השורה של המטריצה וסימן ";" מפריד בין השורות.
למשל נגדיר את המטריצה A באופן הבא,

```
>> A = [2,1,5;6,8,4]
```

בחלון הפעולה יופיע האישור,

```
A =

     2     1     5
     6     8     4
```

נשים לב כי אין צורך להגדיר מראש ב-Matlab את סוג המשתנה.
דרך נוספת להפריד בין איברי השורה של מטריצה היא השמת רווח בין כל איבר בשורה.
כלומר הפקודה,

```
A = [2 1 5;6 8 4]
```

יוצרת את אותה מטריצה בדיוק.

במרחב העבודה (Workspace) מופיעים השמות והגדלים של כל המשתנים הנוכחיים. כעת מופיע משתנה בשם A שהוא מטריצה בגודל $[2\times 3]$.

מה קורה כאשר נגדיר משתנה אך ללא שם משתנה אליו יוצב הערך המחושב?
במקרה כזה Matlab אוטומטית מגדיר משתנה בשם ans שלתוכו מוצב ערך המשתנה. אם יוגדר משתנה נוסף ללא שם, משתנה זה גם יקרא ans והערך הקודם ימחק.
לדוגמא,

```
>> [2 1 5;6 8 4]

ans =

     2     1     5
     6     8     4
```

נוצרה מטריצה $[2\times 3]$ בשם ans הזהה למטריצה A.

במקרים רבים לא נרצה לראות את תוכן המטריצה לאחר פעולת הצבת הערכים ולכן בסוף הפקודה נוסיף ";". כלומר הפקודה,

```
A = [2 1 5;6 8 4];
```

יוצרת את אותה מטריצה A אך איננה מציגה את תוכן המטריצה.

2.1.1 הכנסת ביטויים למשתנים

בנוסף למספרים, איברי מטריצה יכולים להיות גם ביטויים הדורשים חישוב.
לדוגמא,

```
v = [3,49,2^3+1,[7 8 2]];
```

$v \Leftarrow \begin{bmatrix} 3 & 49 & 9 & 7 & 8 & 2 \end{bmatrix}$

חשוב לזכור כי Matlab עוסק במספרים ולא בביטוים סימבוליים. לכן עם רוצים להגדיר וקטור עם ביטויים כללים כמו,

```
v = [x^y + z, w*x]
```

אז Matlab איננו מכיר בביטוי הנ"ל אלא אם כל המשתנים (w, x, y, z) אותחלו בערך מספרי.

2.1.2 מדידת גודל מטריצות ווקטורים

כזכור המטריצה A שהגדרנו היא מטריצה $\begin{bmatrix} 2 \times 3 \end{bmatrix}$. כדי לקבל את ממדי המטריצה (קרי, מספר השורות ומספר העמודות) נשתמש בפונקציה size באופן הבא,

```
[m,n] = size(A);
```

$m \Leftarrow 2$

$n \Leftarrow 3$

לעומת זאת, כדי למדוד אורך וקטור (אין זה משנה אם הוקטור הוא וקטור עמודה או שורה) ניתן להשתמש בפונקציה length באופן הבא,

```
b = [9 7 4];
L = length(b);
```

$L \Leftarrow 3$

גם אם ידוע לנו אורך הווקטור או גודל המטריצה, עדיין רצוי להשתמש בפונקציות length ו- size כי לעיתים רבות פקודות הנכתבות עבור מקרה ספציפי מורחבות למקרה כללי יותר.

2.2 פונקציות להגדרת מטריצות

בנוסף להגדרה הידנית של מטריצות קיימות מספר פונקציות היוצרות מטריצות מיוחדות.

2.2.1 מטריצות אפסים ואחדים

מטריצות אפסים ואחדים הן מטריצות נפוצות לאתחול מטריצה.
יצירת מטריצה $[m\times n]$ שכל איבריה הם אפסים,

```
Z = zeros(m,n)
```

יצירת מטריצה $[m\times n]$ שכל איבריה הם אחדים,

```
Z = ones(m,n)
```

2.2.2 מטריצה אלכסונית

מטריצה אלכסונית היא מטריצה ריבועית $[n\times n]$ המוגדרת כבעלת אפסים בכל האיברים מחוץ לאלכסון הראשי,

$$A=\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix}$$

ניתן ליצר מטריצה אלכסונית באמצעות פונקצית diag ללא כתיבת כל האפסים. פונקציה זו מקבלת כקלט את האלכסון הראשי בתור וקטור ומחזיר מטריצה אלכסונית בהתאם.
לדוגמא הפקודות הנ"ל,

```
b = [7,1,-3,8]';
B = diag(b);
```

מייצרות את המטריצה,

$$B\Leftarrow\begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -3 & 0 \\ 0 & 0 & 0 & 8 \end{bmatrix}$$

ניתן לבצע את הפעולה ההפוכה, כלומר להוציא את האיברים האלכסוניים מהמטריצה הריבועית B באמצעות אותה פונקציה באופן הבא,

```
b2 = diag(B)
```

מקרה פרטי שימושי של מטריצה אלכסונית הוא מטריצת היחידה. כדי לייצר את מטריצת היחידה $I[n\times n]$ משתמשים בפונקציה eye באופן הבא,

```
I = eye(n)
```

2.2.3 מטריצות אקראיות

<u>פונקצית rand</u>
פונקציה זו מייצרת מטריצה בעלת ערכים המוגרלים מהתפלות אחידה $U[0,1]$.
המבנה הכללי שלה הוא,

```
R = rand(m,n);
```

<u>פונקצית randn</u>
פונקציה זו מייצרת מטריצה בעלת ערכים המוגרלים מהתפלות גאוסית (נורמלית) $N(0,1)$. כלומר ממוצע אפס, וריאנס 1.
המבנה הכללי שלה הוא,

```
R = randn(m,n);
```

2.3 יצירת וקטורים בעלי ערכים עוקבים

קיימות שתי דרכים ליצירת וקטורים בעלי ערכים עוקבים.
1. הרישום $s:d:f$.
יצירת וקטור בעל איברים שמתחילים מערך מסוים (s) וגדלים (או קטנים) בפקטור מוגדר (d) עד לערך סיום (f). כלומר כתיבת,

$$x = s:d:f$$

כאשר,
s- ערך התחלתי
d- פקטור גידול או הקטנה
f- ערך סופי

יוצר את הוקטור,

$$x = \begin{bmatrix} s & s+d & s+2d & \cdots & s+(n-1)d \end{bmatrix}$$

נשים לב כי מספר הנקודות (n) לא מוגדר. מעשית, מספר הנקודות יהיה,

$$n = \frac{f-s}{d}+1$$

דוגמא,

```
x = 3:2:9
```

$$x \Leftarrow \begin{bmatrix} 3 & 5 & 7 & 9 \end{bmatrix}$$

ומספר הנקודות הוא $n = \frac{9-3}{2}+1 = 4$.

דוגמא,

```
x = 10:-3:1;
```

$$x \Leftarrow \begin{bmatrix} 10 & 7 & 4 & 1 \end{bmatrix}$$

ומספר הנקודות הוא $n = \frac{1-10}{-3}+1 = 4$.

הערות

- רישום ללא ציון הפקטור d זהה לקביעה d=1.
דוגמא,

```
x = 3:7;
```

$$x \Leftarrow \begin{bmatrix} 3 & 4 & 5 & 6 & 7 \end{bmatrix}$$

- מה קורה כאשר $f \neq s+(n-1)d$?
במקרה כזה מספר הנקודות יעוגל כלפי מטה,

$$\tilde{n} = \left\lfloor \frac{f-s}{d}+1 \right\rfloor$$

ואז הערך הסופי שיתקבל יהיה,

$$\tilde{f} = s+(\tilde{n}-1)d$$

כלומר למעשה ייווצר הוקטור,

$$x = \begin{bmatrix} s & s+d & s+2d & \cdots & s+(\tilde{n}-1)d \end{bmatrix}$$

לדוגמא,

```
x = 1:2:8.999
```

$$\tilde{n} = \left\lfloor \frac{f-s}{d} + 1 \right\rfloor = \left\lfloor \frac{8.999-1}{2} + 1 \right\rfloor = \lfloor 4.9995 \rfloor = 4$$

$$\tilde{f} = s + (\tilde{n}-1)d = 1 + (4-1)\cdot 2 = 7$$

כלומר נוצר הוקטור,

$$x \Leftarrow \begin{bmatrix} 1 & 3 & 5 & 7 \end{bmatrix}$$

השימוש בפונקצית linspace מונע אפשרות של תקלה כזו.

2. פונקצית linspace

יצירת וקטור בעל איברים שמתחילים מערך מסוים וגדלים (או קטנים) בפקטור קבוע עד לערך אחר, אך כעת המשתמש קובע את מספר הערכים הרצויים בין שני ערכי הקצוות והפקטור מחושב באופן אוטומטי. כלומר,

$$x = \text{linspace}(s, f, n)$$

יצור וקטור באורך n בעל פקטור $d = \frac{f-s}{n-1}$. כלומר נוצר הוקטור בעל המרווחים השווים[2] הבא,

$$x = \begin{bmatrix} s & s+d & s+2d & \ldots & s+(n-1)d \end{bmatrix}$$

דוגמא,

```
x = linspace(0,10,100);
```

יוצר וקטור באורך 100 בעל ערכים בין 0 ל-100 במרווח דגימה $\frac{10}{99}$.

מומלץ להשתמש ברישום הראשון כאשר הפקטור d ידוע או חשוב ואילו מספר האיברים n אינו חשוב. לעומת זאת, מומלץ להשתמש ברישום השני כאשר מספר האיברים n ידוע או חשוב, ואילו הפקטור d אינו חשוב.

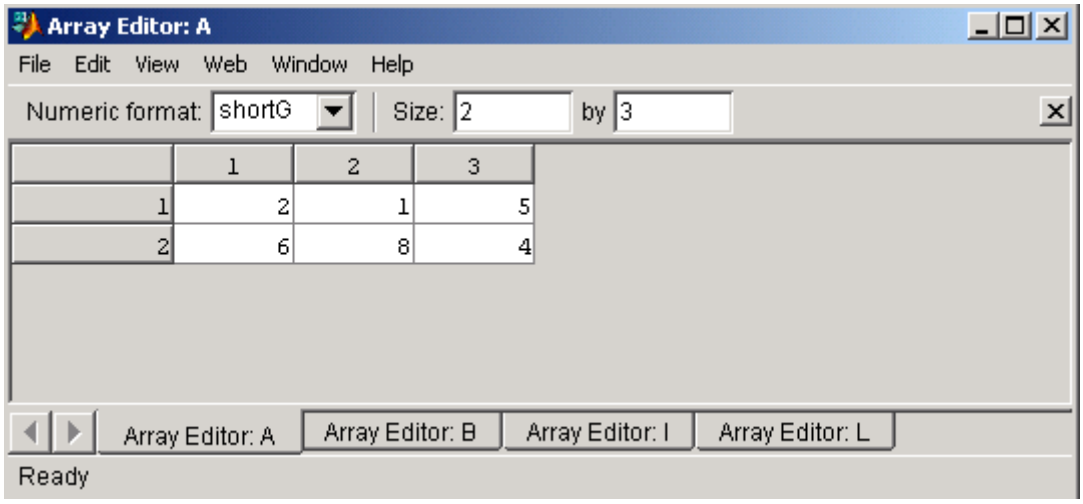
2.4 חלון *Array Editor*

דרך נוחה להתבונן באיברי מטריצות היא ללחוץ פעמיים על הכפתור השמאלי של העכבר על המטריצה הרצויה בחלון ה-Workspace והמטריצה תופיע בחלון Array Editor. לדוגמא לחיצה כפולה על מטריצות A,B,I,L פותח את החלון בגרף2.1.

ניתן לעבור בין משתנה למשתנה ב-Array Editor ע"י לחיצה על הלשונית המתאימה. בנוסף ניתן לשנות את ערכי איברי המטריצה באופן ידני.

דרך נוספת לשנות איברים במטריצה היא באמצעות פעולת הצבה ושימוש באינדקסים.

[2] כדי ליצור וקטור בעל מרווחים שווים בסקאלה לוגריתמית משתמשים בפונקציה logspace באופן הבא, $x = \text{log space}(s, f, n)$ והוקטור שייוצר הוא $x = \begin{bmatrix} 10^s & 10^{s+d} & 10^{s+2d} & \cdots & 10^f \end{bmatrix}$.

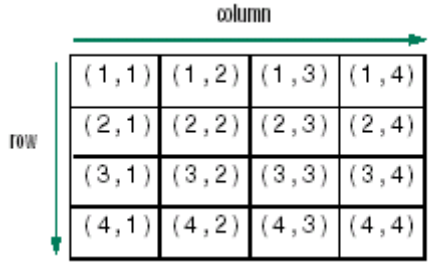


גרף 2.1: חלון Array Editor

3. מניפולציה של מטריצות

3.1 אינדקסים למטריצות

ניתן לגשת לחלק מהאיברים בתוך מטריצה באמצעות סוגריים עגולות ואינדקסים. האיבר הראשון מציין מספר שורה והאיבר השני מציין מספר עמודה[3].
האינדקסים במטריצה הם,

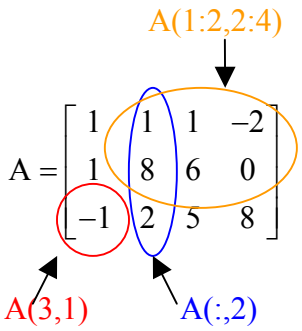


גרף 3.1: אינדקסים במטריצה

לדוגמא,

```
A = [1 1 1 -2;1 8 6 0;-1 2 5 8];
```

$$A \Leftarrow \begin{bmatrix} 1 & 1 & 1 & -2 \\ 1 & 8 & 6 & 0 \\ -1 & 2 & 5 & 8 \end{bmatrix}$$



המשמעות של ":" היא כל האיברים. כלומר A(:,2) היא כל השורות והעמודה השניה. משמעות A(1,:) היא השורה הראשונה וכל העמודות. כלומר הביטוי A(1,:) שקול לביטוי A(1,1:4) או לביטוי A(1,1:end)[4].

ניתן לשנות ערכים בתוך המטריצה באמצעות פעולת הצבה באיברים הרצויים.
דוגמא,
החלפת איבר יחיד במטריצה,

```
A(3,4) = 4;
```

[3] כדי לגשת לאיברי וקטורים אין צורך לציין את גם את השורה וגם את העמודה, מספיק אינדקס יחיד.
[4] כתיבת end בסוף וקטור האינדקס אוטומטית מחזירה את מספר השורה או העמודה המכסימליים (בהתאם למיקום בסוגריים העגולות).

דוגמא,
החלפת תת-מטריצה במטריצה,

```
A(1:2,3:4) = [3 6;8 4];
```

דוגמא,
הצבת ערך זהה בתת מטריצה ע"י פעולת הצבת סקלר,

```
A(2:3,1:2) = 17;
```

ניתן גם להיפטר משורות או עמודות ע"י הצבת סימון מטריצה ריקה - סוגרים מרובעות ריקות [].
לדוגמא,
נניח כי ברצוננו להוריד שורה של מטריצה. לשם המחשה ניצור מטריצה,

```
A = magic(5);
```

$$A \Leftarrow \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

נניח כי השורה השלישית איננה מוצאת חן בעיננו. ניפטר ממנה באופן הבא,

```
A(3,:) = [];
```

$$A \Leftarrow \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

אם ננסה להשתמש בערך הנמצא מחוץ לגבולות המטריצה, תתקבל הודעת שגיאה.
לדוגמא,

```
>> a = A(5,6)
???  Index exceeds matrix dimensions.
```

לעומת זאת, אם ננסה להציב ערך מחוץ לגבולות המטריצה אז היא תורחב כך שבאיברים החדשים שלא הוגדרו יהיו אפסים.
לדוגמא,

```
A(5,6) = 2;
```

$$A \Leftarrow \begin{bmatrix} 17 & 24 & 1 & 8 & 15 & 0 \\ 23 & 5 & 7 & 14 & 16 & 0 \\ 10 & 12 & 19 & 21 & 3 & 0 \\ 11 & 18 & 25 & 2 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

3.1.1 פונקציות find ו-sort

פונקצית find נועדה למציאת ערכים מסוימים בתוך מטריצות או וקטורים[5]. בתור ברירת מחדל, פונקצית find מחזירה את האינדקסים של כל האיברים במטריצה השונים מאפס. עבור מטריצות הרישום הכללי של הפונקציה הוא,

```
[I,J] = find(A)
```

ועבור וקטורים הרישום הוא,

```
I = find(a)
```

דוגמא למטריצות,
נגדיר מטריצה A,

```
A = [0 0 0 3 0;0 0 0 0 0;-2 0 0 0 0];
```

$$A \Leftarrow \begin{bmatrix} 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 \end{bmatrix}$$

```
>> [I,J] = find(A)

I =

     3
     1


J =

     1
     4
```

הוספת ארגומנט שלישי גורמת לפונקצית find להחזיר גם את ערך האיברים הרצויים.
לדוגמא,

```
[I,J,V] = find(A)

I =

     3
     1


J =

     1
     4


V =

    -2
     3
```

דוגמא לוקטורים,

```
a = [zeros(1,4),8,zeros(1,2),-3,0];
index = find(a);
```

[5] אם לא נמצאים הערכים המבוקשים אז הפונקציה מחזירה מטריצה ריקה.

$$a \Leftarrow [0 \quad 0 \quad 0 \quad 0 \quad 8 \quad 0 \quad 0 \quad -3 \quad 0]$$
$$index \Leftarrow [5,8]$$

שימוש אפשרי של אינדקס זה הוא לקבל וקטור המכיל את כל האיברים השונים מאפס בוקטור a,

```
a_no_zero = a(index);
```
$$a\_no\_zero \Leftarrow [8,-3]$$

או ברישום יותר קומפקטי,

```
a_no_zero = a(find(a));
```

פונקצית find איננה מוגבלת רק למציאת איברים השונים מאפס. ניתן לבחור אלמנטים אחרים באמצעות כתיבת תנאי רצוי בסוגריים של פונקצית ה-find,

```
a = [3 2 1 8 9 11 13 22 0 4 5 6 7];
index = find(a>=4 & a<=7);
```
$$index \Leftarrow [10 \quad 11 \quad 12 \quad 13]$$

פונקציה שימושית נוספת לקבלת אינדקסים היא פונקצית sort המסדרת מחדש וקטורים כך שאיבריהם יהיו מהקטן עד לגדול ואת האינדקס המשמש לביצוע הסידור מחדש.
לדוגמא,

```
m = [5 93 -3 -5 1 0 7];
[m_sorted,index] = sort(m);
```
$$m\_sorted \Leftarrow [-5 \quad -3 \quad 0 \quad 1 \quad 5 \quad 7 \quad 93]$$
$$index \Leftarrow [4 \quad 3 \quad 6 \quad 5 \quad 1 \quad 7 \quad 2]$$

3.2 אופרטורים מטריציים

פעולת transpose[6] (חילוף בין שורות ועמודות מטריצה) מבוצעת באמצעות הסימן ',

```
A = [2 1 5;6 8 4];
C = A';
```
$$C \Leftarrow \begin{bmatrix} 2 & 3 \\ 1 & 8 \\ 5 & 4 \end{bmatrix}$$

ניתן לבצע פעולות בסיסיות של חיבור, חיסור וכפל מטריצות באמצעות האופרטורים "+","-","*" בהתאם לחוקי האלגברה הלינארית.
חיבור מטריצות,

```
B = [8 5 1;0 9 2];
C = A + B;
```
$$C \Leftarrow \begin{bmatrix} 10 & 6 & 6 \\ 3 & 17 & 6 \end{bmatrix}$$

כפל מטריצות,

```
C = A'*B;
```

[6] אופרטור הגרש ' מבצע transpose וגם צמוד קומפלקסי. עבור מטריצות ממשיות אין זה משפיע, אך כדי לבצע transpose ללא צמוד קומפלקסי משתמשים באופרטור הגרש עם נקודה לפניו (.').

$$C \Leftarrow \begin{bmatrix} 16 & 37 & 8 \\ 8 & 77 & 17 \\ 40 & 61 & 13 \end{bmatrix}$$

כפל סקלר במטריצה,

```
C = 2*B;
```

$$C \Leftarrow \begin{bmatrix} 16 & 10 & 2 \\ 0 & 18 & 4 \end{bmatrix}$$

סכימה בין שתי מטריצות בעלות ממדים שונים איננה חוקית ב-Matlab מאחר ובאלגברה לינארית פעולה כזו איננה מוגדרת. אבל ב-Matlab קיים מקרה יוצא מהכלל יחיד- סכימה בין מטריצה לסקלר. במקרה כזה הפעולה הנ"ל שקולה לסכימה על כל איברי המטריצה.
לדוגמא,

```
C = B - 1;
```

$$C \Leftarrow \begin{bmatrix} 7 & 4 & 0 \\ -1 & 8 & 1 \end{bmatrix}$$

3.2.1 דטרמיננטה

ניתן לחשב את הדטרמיננטה של מטריצה ריבועית באמצעות פונקצית det.
לדוגמא,

```
A = [1 2;...
     4 3];
det(A);
```

$$ans \Leftarrow -5$$

3.2.2 היפוך מטריצה

ניתן להפוך מטריצה ריבועית[7] $A[n \times n]$ בשתי דרכים שונות,

1. פונקצית inv או אופרטור 1-^.
   לדוגמא,

```
inv(A)
```

או

```
A^-1
```

מפעילים את אותו אלגוריתם הפיכת מטריצה שתוצאתו,

$$A \Leftarrow \begin{bmatrix} -0.6 & 0.4 \\ 0.8 & -0.2 \end{bmatrix}$$

2. אופרטור / ובאמצעות מטריצת יחידה.
   לדוגמא,

```
eye(2)/A
```

[7] בתנאי ש- $\det(A) \neq 0$.

3.3 אופרטור הנקודה

פעולות המבוצעות באמצעות אופרטור הנקודה למעשה משנות את המשמעות של המטריצות למערכים.
כלומר פעולת נקודה בין שתי מטריצות היא פעולה בין שני מערכים- פעולה איבר מול איבר.
למען המחשה נגדיר שתי מטריצות כלליות,

$$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$$

$$B=\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{bmatrix}$$

כפל,

$$C=A.*B=\begin{bmatrix} a_{11}\cdot b_{11} & a_{12}\cdot b_{12} & a_{13}\cdot b_{13} & a_{14}\cdot b_{14} \\ a_{21}\cdot b_{21} & a_{22}\cdot b_{22} & a_{23}\cdot b_{23} & a_{24}\cdot b_{24} \\ a_{31}\cdot b_{31} & a_{32}\cdot b_{32} & a_{33}\cdot b_{33} & a_{34}\cdot b_{34} \end{bmatrix}$$

חילוק,

$$C=A./B=\begin{bmatrix} a_{11}/b_{11} & a_{12}/b_{12} & a_{13}/b_{13} & a_{14}/b_{14} \\ a_{21}/b_{21} & a_{22}/b_{22} & a_{23}/b_{23} & a_{24}/b_{24} \\ a_{31}/b_{31} & a_{32}/b_{32} & a_{33}/b_{33} & a_{34}/b_{34} \end{bmatrix}$$

חזקה,

$$C=A.\hat{}\,B=\begin{bmatrix} a_{11}^{b_{11}} & a_{12}^{b_{12}} & a_{13}^{b_{13}} & a_{14}^{b_{14}} \\ a_{21}^{b_{21}} & a_{22}^{b_{22}} & a_{23}^{b_{23}} & a_{24}^{b_{24}} \\ a_{31}^{b_{31}} & a_{32}^{b_{32}} & a_{33}^{b_{33}} & a_{34}^{b_{34}} \end{bmatrix}$$

הערות

- אופרטור הנקודה ממוקם לפני אופרטור הפעולה החשבונית.
- עבור מטריצות ריבועיות משמעות הכפל והחילוק ללא אופרטור הנקודה היא שונה.

עבור המקרים הבאים אופרטור הנקודה מיותר,
1. חיבור או חיסור.
2. כפל או חילוק כל איברי המטריצה באותו קבוע (ניתן לביצוע באמצעות כפל או חילוק בסקלר).

3.3.1 שימושים לאופרטור הנקודה

כדי לבצע את פעולת החזקה (a^b) עם מספר קבוע (בין אם a הוא הקבוע או b הוא הקבוע) אז אופרטור הנקודה הכרחי.
דוגמא,

```
x = 1:7;
y = 2.^x;
```

$y \Leftarrow [2 \quad 4 \quad 8 \quad 16 \quad 32 \quad 64 \quad 128]$

דוגמא,

```
y = x.^2;
```

$y \Leftarrow [1 \quad 4 \quad 9 \quad 16 \quad 25 \quad 36 \quad 49]$

באמצעות אופרטור הנקודה ניתן לחשב ביטויים מתמטיים מסובכים ללא כל לולאות.

דוגמא,

חישוב הביטוי $s = \sum_{x=1}^{5} x^x$ מבוצע ע"י כתיבת[8],

```
x = 1:5;
s = sum(x.^x);
```

$s \Leftarrow 3413$

דוגמא,
נתונות 3 מטריצות בעלות אותו גודל (לא בהכרח ריבועי) A,B,C ונדרש לחשב את הביטוי

$$f = \left(\frac{\sin(A)}{B \,\hat{}\, C}\right)^3 + 1 .$$

המימוש,

```
f = (sin(A)./(B.^C)).^3 + 1;
```

אחת הבעיות הכי נפוצות ב-Matlab היא בעיית הסוגריים. לעיתים רבות ביטוי זקוק לזוגות רבים ומבלבלים של סוגריים ולכן ניתן בקלות לפספס סוגר ותתקבל הודעת שגיאה. מקרה חמור יותר הוא כאשר מספר הסוגרים תקין אך הצבנו סוגר במיקום לא נכון ואז לא תתקבל הודעת שגיאה אך יחושב ביטוי לא רצוי.

בחלון Editor\Debugger ניתן להתגבר על בעיית הסוגריים ע"י הזזת הסמן מתחת לסוגר. אם אין לסוגר סוגר משלים אז זמנית משורטט קו אופקי דרכו. אם קיים לו סוגר משלים אז מצוייר קו תחתי מתחת לסוגר המסומן והסוגר המשלים שלו.

[8] פונקצית sum מבצעת סכימה על איברי וקטור. לפרטים פנו לפרק 3.4.

3.4 פונקציות סכימה וכפל

קיימות שתי פונקציות סכימה,
1. פונקצית sum
פונקציה זו מקבלת וקטור ומחזירה סקלר המכיל את סכום איברי הוקטור,

$$s = \mathrm{sum}(x) = \sum_{i=1}^{n} x_i$$

לדוגמא,

```
x = 1:4;
s = sum(x);
```

$s \Leftarrow 10$

פונקציה זו פועלת על מטריצות ע"י סכימה לאורך העמודות של המטריצה והחזרת התוצאה לוקטור שאורכו כמספר העמודות.
באופן כללי עבור מטריצה $X\left[m \times n\right]$ פונקצית sum מחזירה את הערכים הבאים,

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \cdots & x_{mn} \end{bmatrix}$$

$$s = \mathrm{sum}(X) = \begin{bmatrix} \sum_{i=1}^{m} x_{i1} & \sum_{i=1}^{m} x_{i2} & \cdots & \sum_{i=1}^{m} x_{in} \end{bmatrix}$$

לדוגמא[9],

```
X = [1 2 3 4;...
     5 6 7 8;...
     9 10 11 12];
s = sum(X);
```

$s \Leftarrow \begin{bmatrix} 15 & 18 & 21 & 24 \end{bmatrix}$

כדי לסכום על פני השורות של המטריצה X נרשום,

```
s = sum(X');
```

מתקבל,

$s \Leftarrow \begin{bmatrix} 10 & 26 & 42 \end{bmatrix}$

2. פונקצית cumsum
פונקציה זו היא פונקצית הסכום המצטבר. פונקציה זו מקבלת וקטור ומחזירה וקטור באותו אורך כאשר בכל איבר מוצב ערך הסכום המצטבר.
באופן כללי עבור וקטור $x\left[n \times 1\right]$ פונקצית cumsum מחזירה את הערכים הבאים,

$$s = \mathrm{cumsum}(x) = \begin{bmatrix} \sum_{i=1}^{1} x_i & \sum_{i=1}^{2} x_i & \cdots & \sum_{i=1}^{n} x_i \end{bmatrix}$$

[9] בדוגמא זו המטריצה מוגדרת בעזרת אופרטור שלוש הנקודות. אופרטור זה נועד לשם נוחות הקלדה ולשם בהירות בקוד.

לדוגמא,

```
x = 1:4;
s = cumsum(x);
```

$$s \Leftarrow \begin{bmatrix} 1 & 3 & 6 & 10 \end{bmatrix}$$

פונקציה זו פועלת על מטריצות ע"י סכימה באופן מצטבר לאורך העמודות של המטריצה והחזרת התוצאה למטריצה באותה גודל כמו מטריצת הקלט.
עבור מטריצה כללית,

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \cdots & x_{mn} \end{bmatrix}$$

פונקצית cumsum מחזירה את מטריצת הסכומים המצטברים,

$$S = \text{cumsum}(X) = \begin{bmatrix} \sum_{i=1}^{1} x_{i1} & \sum_{i=1}^{1} x_{i2} & \cdots & \sum_{i=1}^{1} x_{in} \\ \sum_{i=1}^{2} x_{i1} & \sum_{i=1}^{2} x_{i2} & \cdots & \sum_{i=1}^{2} x_{in} \\ \vdots & \vdots & & \vdots \\ \sum_{i=1}^{m} x_{i1} & \sum_{i=1}^{m} x_{i2} & \cdots & \sum_{i=1}^{m} x_{in} \end{bmatrix}$$

3.4.1 ייצוגים של סכומים באמצעות מכפלת וקטורים ומטריצות

ניתן לייצג סכומים באמצעות מכפלת וקטורים ומטריצות. להלן שתי דוגמאות,

1. מכפלה סקלרית בין שני וקטורים
   נתון,
   a, b – וקטורים $[n \times 1]$

$$\langle a, b \rangle = \sum_{i=1}^{n} a_i \cdot b_i = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \\ b_n \end{bmatrix} = a' * b$$

2. תבנית ריבועית
   נתון,
   x – וקטור $[n \times 1]$
   C – מטריצה $[n \times n]$

$$f = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i c_{ij} x_j = x' * C * x$$

3.4.2 פונקצית כפל prod

פונקציה זו מקבלת וקטור ומחזירה סקלר המכיל את מכפלת איברי הוקטור,

$$p = \text{prod}(x) = \prod_{i=1}^{n} x_i$$

לדוגמא,

```
x = 1:4;
p = prod(x);
```

$p \Leftarrow 24$

אופן פעולת פונקצית prod על מטריצות זהה לאופן פעולת פונקצית sum.

3.5 שכפול מטריצות

קיימות שלוש פונקציות המשכפלות מטריצות, וקטורים או סקלרים,

3.5.1 פונקצית repmat

המבנה הכללי של פונקצית repmat הוא,

$$\text{repmat}(X, r, c)$$

כאשר X היא מטריצה, וקטור או סקלר המיועד לשכפול. r הוא מספר הפעמים שהמטריצה X תשוכפל בכיוון השורות ו-c הוא מספר הפעמים שהמטריצה X תשוכפל בכיוון העמודות.
דוגמא לשכפול סקלר,

```
Z = repmat(2,3,4);
```

$$Z \Leftarrow \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

ניתן לבצע פעולה זאת גם באופן הבא,

```
Z(1:3,1:4) = 2;
```

דוגמא לשכפול ווקטור,

```
v = [5 9 1 7 2];
Z = repmat(v,3,2);
```

$$Z \Leftarrow \begin{bmatrix} 5 & 9 & 1 & 7 & 2 & 5 & 9 & 1 & 7 & 2 \\ 5 & 9 & 1 & 7 & 2 & 5 & 1 & 9 & 7 & 2 \\ 5 & 9 & 1 & 7 & 2 & 5 & 1 & 9 & 7 & 2 \end{bmatrix}$$

דוגמא לשכפול מטריצה,

```
B = 5*eye(2);
Z = repmat(B,2,3);
```

$$Z \Leftarrow \begin{bmatrix} 5 & 0 & 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 & 0 & 5 \\ 5 & 0 & 5 & 0 & 5 & 0 \\ 0 & 5 & 0 & 5 & 0 & 5 \end{bmatrix}$$

3.5.2 פונקצית meshgrid

השימוש העיקרי בפונקציה זו הוא המרת זוג וקטורים המיצגים את הצירים x, y למטריצות X , Y המייצגות את הצירים ומשמשות לפונקציות ליצירת גרפים תלת-ממדיים.
הפקודות,

$$x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix};$$
$$y = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix};$$
$$[X, Y] = \text{meshgrid}(x, y);$$

יוצרת את המטריצות,

$$X \Leftarrow \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 \end{bmatrix}$$
$$Y \Leftarrow \begin{bmatrix} y_1 & y_1 & y_1 & y_1 \\ y_2 & y_2 & y_2 & y_2 \\ y_3 & y_3 & y_3 & y_3 \end{bmatrix}$$

נשים לב כי במטריצה X קיימים שינויים רק בכיוון 'ציר x' ואילו במטריצה Y קיימים שינויים רק בכיוון 'ציר y'.
לדוגמא,

```
x = 1:4;
y = 1:3;
[X,Y] = meshgrid(x,y);
```

$$X \Leftarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$
$$Y \Leftarrow \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \end{bmatrix}$$

אם ברצוננו ליצור ציר x וציר y זהים נרשום רק וקטור אחד בקלט של הפונקציה.
לדוגמא,

```
[X2,Y2] = meshgrid(2:2:8);
```

$$X2 \Leftarrow \begin{bmatrix} 2 & 4 & 6 & 8 \\ 2 & 4 & 6 & 8 \\ 2 & 4 & 6 & 8 \\ 2 & 4 & 6 & 8 \end{bmatrix}$$

$$Y2 \Leftarrow \begin{bmatrix} 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 \\ 6 & 6 & 6 & 6 \\ 8 & 8 & 8 & 8 \end{bmatrix}$$

השימוש בפונקציה זו יוסבר ביתר פירוט בפרק 6.2 הדן בשרטוט משטחים תלת ממדיים.

3.5.2 פונקצית kron

פונקציה זו מבצעת כפל טנזורי בין שתי מטריצות.
משמעות כפל טנזורי בין שתי מטריצות $X_{[m1\times n1]}, Y_{[m2\times n2]}$ היא מכפלת כל איבר במטריצה X בכל המטריצה Y וסידור כל המכפלות האלו במטריצה חדשה בגודל $\left[(m1\cdot m2)\times(n1\cdot n2)\right]$.
לדוגמא,

$$X=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$$

$$Y=\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$$K=\mathrm{kron}(X,Y)=\begin{bmatrix}1\cdot\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} & 2\cdot\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}\\ 3\cdot\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} & 4\cdot\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}\end{bmatrix}=\begin{bmatrix}1 & 1 & 1 & 2 & 2 & 2\\ 1 & 1 & 1 & 2 & 2 & 2\\ 1 & 1 & 1 & 2 & 2 & 2\\ 3 & 3 & 3 & 4 & 4 & 4\\ 3 & 3 & 3 & 4 & 4 & 4\\ 3 & 3 & 3 & 4 & 4 & 4\end{bmatrix}$$

בדוגמא זו $X_{[2\times 2]}, Y_{[3\times 3]}$ ומטריצה K היא בעלת הממדים $\left[(2\cdot 3)\times(2\cdot 3)\right]$.

ניתן ממש לשכפל מטריצות ע"י קביעת X כוקטור או מטריצת אחדים.

3.6 היפוך סדר מטריצה

קיימות שתי פונקציות להיפוך מטריצות,

1. פונקצית fliplr (flip left-right)
   פונקציה זו הופכת את סדר העמודות,

```
X_lr = fliplr(X);
```

$$X\_lr \Leftarrow \begin{bmatrix} 4 & 3 & 2 & 1 \\ 4 & 3 & 2 & 1 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

2. פונקצית flipud (flip up-down)
   פונקציה זו הופכת את סדר השורות,

```
Y_ud = flipud(Y);
```

$$Y\_ud = \begin{bmatrix} 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

3.7 פתירת מערכת משוואות לינארית

נתונה מערכה של m משוואות ו-n נעלמים,

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases}$$

מערכת משוואות זו ניתנת לכתיבה בצורה מטריצית,

$$A \cdot x = b$$

במקרה שבו מספר המשוואות ומספר הנעלמים זהה $(m = n)$ אז המטריצה A היא מטריצה ריבועית ניתן לפתור[10] את מערכת המשוואות הנ"ל באמצעות פונקצית inv באופן הבא,

```
x = inv(A)*b;
```

עבור המקרה הלא ריבועי $(m \neq n)$ יש צורך לחשב את pseudo-inverse,

```
x = inv(A'*A)*A'*b;
```

דרך עדיפה לפתור את מערכת המשוואות הנ"ל היא באמצעות האופרטור \ באופן הבא,

```
x = A\b
```

[10] עבור מטריצה A הפיכה.

3.8 שמירת וטעינת משתנים

שמירת כל המשתנים ב-Workspace נשתמש בפונקציה save באופן הבא,

```
save name
```

פונקציה זו יוצרת קובץ name.mat המכיל את כל המשתנים ב-Workspace.
כדי לשמור רק משתנים ספציפיים נרשום כמקודם ונוסיף את שמות המשתנים לשמירה לאחר שם קובץ ה-mat.
למשל כדי לשמור את המשתנים x yy Z לקובץ mat. בשם name נבצע,

```
save name x yy Z
```

כדי למחוק את כל המשתנים שיצרנו במרחב העבודה נשתמש בפונקציה clear. כדי למחוק משתנים ספציפיים נשתמש בפונקציה clear ונציין את שמות המשתנים למחיקה.
למשל כדי למחוק את המשתנים x Z נבצע,

```
clear x Z
```

אם ברצוננו לטעון משתנים מקובץ mat למרחב העבודה משתמשים בפונקציה load באופן הבא,

```
load name
```

או פשוט לחיצה כפולה על קובץ mat. הרצוי בחלון הספרייה הנוכחית (Current Directory).

3.9 קבלת והצגת מידע ל-Command Window

3.9.1 מחרוזות

כלי מרכזי בהצגת מידע באופן ברור הוא השימוש במחרוזות להצגת טקסט.
מחרוזת (String) ב-Matlab היא למעשה וקטור שכל איבר בו הוא תו יחיד. מחרוזת מוגדרת כטקסט בעל גרשיים משני צדדיו.
לדוגמא,

```
s = 'name';
```

יוצר וקטור באורך 4 מסוג char array (מערך תווים). נשים לב כי,

s(2) ← a

fliplr(s) ← eman

מחרוזת היא מערך בדומה לכל מערך ב-Matlab. לכן ניתן למשל ליצור מטריצה של מחרוזות.
לדוגמא,

```
s2 = [s;fliplr(s)];
```

s2(1,:) ← name

s2(2,:) ← eman

במקרה הנ"ל שתי המחרוזות היו בעלות אותו אורך ולכן לא היתה בעיה ליצור מהן מטריצה. אך אם נרצה להכניס מחרוזות בעלות אורכים שונים נקבל הודעת שגיאה. ניתן להתגבר על בעיה זו עם פונקצית str2mat.

לדוגמא כתיבת,

```
numbers = str2mat('one','three','four')
```

יוצר מטריצה $[3\times5]$ של מערך תווים ומתקבל ב-Command Window,

```
numbers =

one
three
four
```

פונקציה זו מייצרת רווחים בסוף המחרוזות הקצרות יותר.

כדי לשלוף מחרוזת מהמטריצה ללא רווח השתמשו בפונקצית deblank.

3.9.2 הצגת מידע ב-Command Window

ישנן מספר דרכים להצגת ערכים של משתנים ב-Command Window,

1. להשמיט את ";". זוהי הדרך הכי פשוטה, אך הכי פחות יפה ומסודרת.
2. פונקצית disp.
3. פונקצית fprintf.

פונקצית disp מציגה את המחרוזת הנמצאת בסוגריים העגולות שלה.
לדוגמא כתיבת הפקודה,

```
disp('This is the text which will appear in the Command Window')
```

מציגה בחלון Command Window את הטקסט,

```
This is the text which will appear in the Command Window
```

במקרים רבים יש צורך בשילוב של טקסט קבוע והצגת ערכי משתנים מספריים (למשל בהצגת תוצאות של מדידה או בכותרת של גרף). מאחר ופונקציות הצגת טקסט כמו disp יכולות להציג מחרוזות בלבד, משתמשים בפונקצית num2str להמיר את הערכים המספרים למחרוזות.
דוגמא,
התקבל כי אורך אובייקט הוא 9 ס"מ ואנו רוצים להציג זאת ב-Command Window.

```
len_obj = 9;
disp(['The lenght of the object is: ',num2str(len_obj),'cm'])
```

ב-Command Window מוצג,

```
The lenght of the object is: 9cm
```

ניתן גם לשלוט במספר הספרות[11] המומרות למחרוזת ע"י הוספת ארגומנט נוסף.
לדוגמא,

```
len_obj = 29.347347823;
disp(['The lenght of the object is: ',num2str(len_obj,5),'cm'])
```

הוספנו ארגומנט נוסף בעל ערך 5. ארגומנט זה קבע כי מספר הספרות שיומרו למחרוזת הוא 5. לכן הפלט הבא מופיע ב-Command Window,

```
The lenght of the object is: 29.347cm
```

פונקציה אלטרנטיבית ל-disp היא fprintf. פונקציה זו יחסית יותר מורכבת אך מאפשרת שליטה גבוהה יותר בתצוגה. לפרטים חפשו בעזרה של Matlab.

3.9.3 קבלת מידע מה-Command Window

ניתן לקבל ערך מספרי של משתנה מהמשתמש בחלון Command Window באמצעות פונקצית input באופן הבא,

```
x = input('prompt_string');
```

פונקציה זו מציגה ב-Command Window את המחרוזת הנמצאת בסוגריים ומחכה שהמשתמש יקליד ערך וילחץ על Enter. ברגע שזה קורה, מוכנס הערך שהמשתמש בחר למשתנה בשם x.

ניתן להכניס גם מחרוזת לתוך משתנה באופן הבא,

```
name = input('prompt_string','s');
```

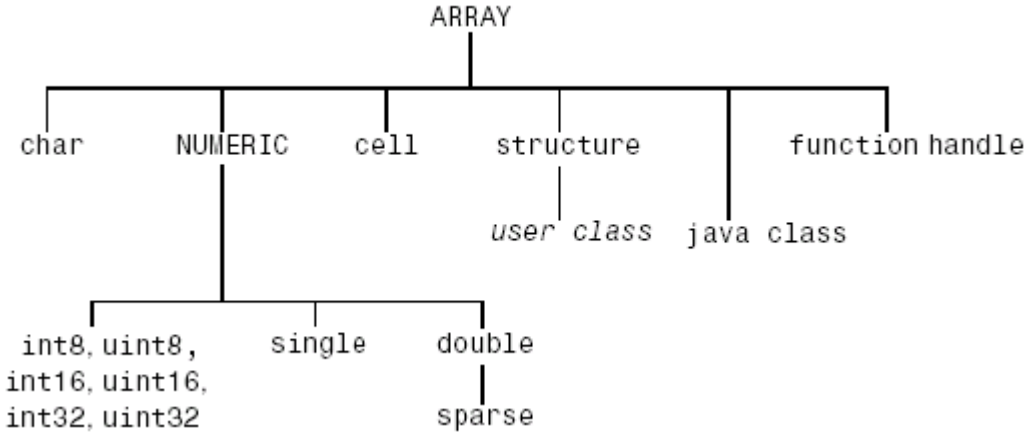
הערות

- ניתן להוסיף \n כדי לרדת שורה בטקסט.
- פונקציה נוספת המקבלת הוראות ישירות מהמשתמש היא פונקצית pause. פונקציה זו משהה את ההרצה עד שהמשתמש ילחץ על כפתור כלשהו.

[11] כולל ספרות לפני הנקודה העשרונית.

4. מבני מידע

עד עתה נחשפנו למספר מבני מידע כמו מטריצות ומחרוזות. ב-Matlab קיימים סוגי מבני המידע הבאים,



גרף 4.1: סוגי מבני מידע של Matlab

כפי שניתן לראות מגרף 4.1 כל מבני המידע ב-Matlab הם מערכים: למשל מערכים נומריים דו-ממדיים נקראים מטריצות ומחרוזות הן מערך של תווים (character array). בפרק זה נלמד להכיר מספר סוגים של מבני מידע נוספים.

4.1 מערכים נומריים רב-ממדיים

לעיתים רבות רצוי ליצור מערכים בעלי יותר משני ממדים. ניתן לחשוב על מערך תלת ממדי כאוסף של מטריצות אחת אחרי השניה כפי שמצוייר בגרף 4.2. לדוגמא, תמונת RGB מורכבת משלוש מטריצות (באותו גודל כמובן) המציינות את עוצמת האדום, ירוק, וכחול בתמונה צבעונית. ניתן לחשוב על מערכים רב-ממדיים גם בתור מידע רב ממדי, לדוגמא מדידות תלת ממדיות של טמפרטורה בחדר מהוות מערך תלת ממדי.
ניתן ליצור מערכים רב ממדיים באמצעות פונקציות יצירת מטריצה בסיסיות כמו zeros, ones, rand, randn ע"י הצבת יותר משני ארגומנטים[12].
לדוגמא הפקודה,

```
A = zeros(3,4,5);
```

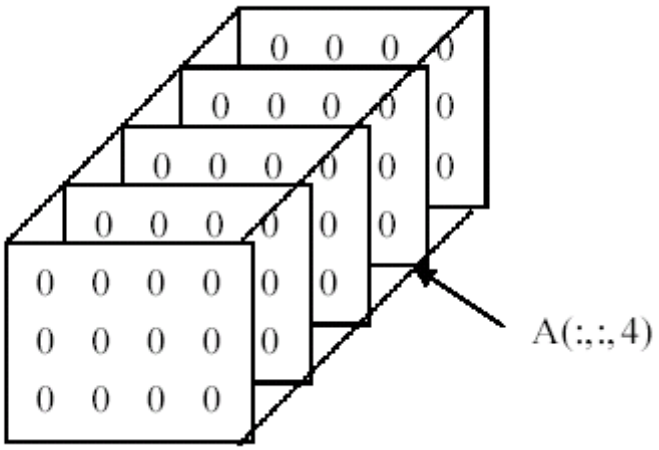
יוצרת מערך 3 על 4 על 5 בעל $3 \cdot 4 \cdot 5 = 60$ איברים שערכם אפס. ניתן לחשוב על האיבר A(i,j,k) כעל איבר (i,j) במטריצה k.

הגודל של A הוא,

```
>> size(A)

ans =

     3     4     5
```

[12] ניתן גם להשתמש בפונקצית repmat ובפונקצית cat.

המחשה גרפית,



גרף 4.2: מערך 3 על 4 על 5 של אפסים

ניתן לבצע פונקציות Matlab רבות על מערכים נומריים רב ממדיים. ישנן פונקציות שבאופן בסיסי פועלות על וקטורים (למשל פונקצית sum) או פועלות על סקלרים, כלומר איבר איבר (למשל פונקצית sin).
פונקציות הפועלות על וקטורים צריכות שיציינו להן באיזה ממד לבצע את פעולתן.
למשל פונקצית sum יכולה לקבל מערך רב-ממדי A וממד סכימה d באופן הבא,

```
sum(A,d)
```

למשל, sum(A,3) יניב מטריצה $[4\times 3]$ שהוא למעשה הסכימה על המטריצות בגרף 4.2.

פונקציות הפועלות איבר איבר פועלות באותו אופן שפעלו על מערכים חד-ממדיים ודו-ממדיים.
למשל,

```
sin(A)
```

מייצרת מערך בגודל זהה למערך A.

4.1.1 שינוי ממדי מערך

ניתן לשנות את סדר הממדים של מערך באמצעות פונקצית permute או לשנות את הממדים עצמם באמתעות פונקצית reshape.

<u>שינוי סדר הממדים</u>

פונקצית permute מאפשרת החלפת סדר הממדים. פונקציה זו היא למעשה הרחבה של פעולת ה-transpose למערכים רב-ממדיים.

למשל נניח כי נתון מערך רב ממדי B בגודל $[7\times 13\times 14\times 8]$. כדי להפוך את סדר הממדים של המערך C ל-$[7\times 8\times 13\times 14]$ נרשום,

```
B = zeros(7,13,14,8);
C = permute(B,[1 4 2 3]);
```

<u>שינוי הממדים</u>

פונקצית reshape משנה את הממדים של מערך. פונקציה זו מקבלת כקלט את המערך ואת ממדיו החדשים. אם מספר האיברים במערך החדש שונה מהמערך בקלט מתקבלת הודעת שגיאה. סדר הדגימה של המערך בקלט וסדר הסידור מחדש של מערך הפלט הוא עמודות, שורות ושאר הממדים לפי הסדר.
לדוגמא,

```
A = [[1:3]',[4:6]',[7:9]',[10:12]'];
A = reshape(A,2,6);
```

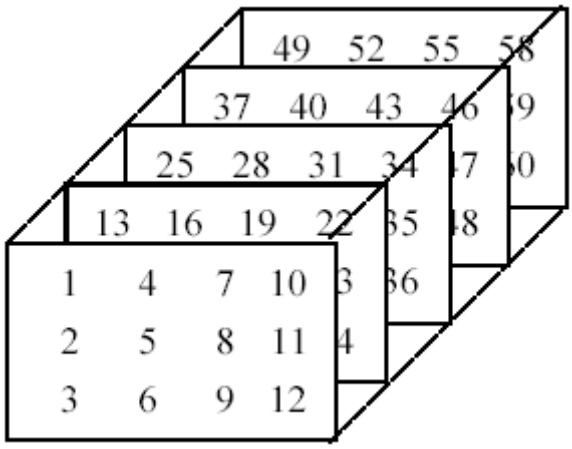
$$A=\begin{bmatrix}1 & 4 & 7 & 10\\ 2 & 5 & 8 & 11\\ 3 & 6 & 9 & 12\end{bmatrix}$$

$$\Downarrow$$

$$A=\begin{bmatrix}1 & 3 & 5 & 7 & 9 & 11\\ 2 & 4 & 6 & 8 & 10 & 12\end{bmatrix}$$

באמצעות פונקציה זו ניתן גם ליצור מערכים רב ממדיים.
לדוגמא,

```
A = 1:60;
A = reshape(A,3,4,5);
```



גרף 4.3: מערך 3 על 4 על 5

<u>טריק</u>

A(:) הופך כל מערך נומרי רב ממדי לוקטור עמודה אחד ארוך באותו סדר דגימה וסידור מחדש של פונקצית reshape. לכן גם ניתן לשחזר מוקטור הנוצר בשיטה זו את המערך הרב ממדי שממנו הוא נוצר באמצעות פונקצית reshape.

לדוגמא,

```
A(:);
A = reshape(A,3,4,5);
```

מניב את אותו מערך רב ממדי A המתואר בגרף 4.3.

4.2 מערכי תאים (cell arrays)

מערך cell מאפשר אכסון משתנים מטיפוסים וגדלים שונים.
מערך cell הוא מערך שכל איבר בו הוא תא המכיל מערך. מערך תאים נוצר ע"י רישום הדומה למטריצה למעט שימוש בסוגריים מסולסלות[13] במקום במרובעות.
לדוגמא,

```
A = [2 5 0 9;...
     1 1 3 7;...
     6 8 9 5];
b = 2*ones(5,1);
c = 'string';
d = 7;

C = {A,b;c,d}
```

יוצר cell array בגודל $[2\times2]$ הבא,

```
C =

    [3x4 double]    [5x1 double]
    'string'        [         7]
```

שתי דרכים נוספות להציג מערך תאים הן,
1. פונקצית celldisp המציגה פירוט כל תא.
לדוגמא,

```
celldisp(C)
```

מייצר ב-Command Window,

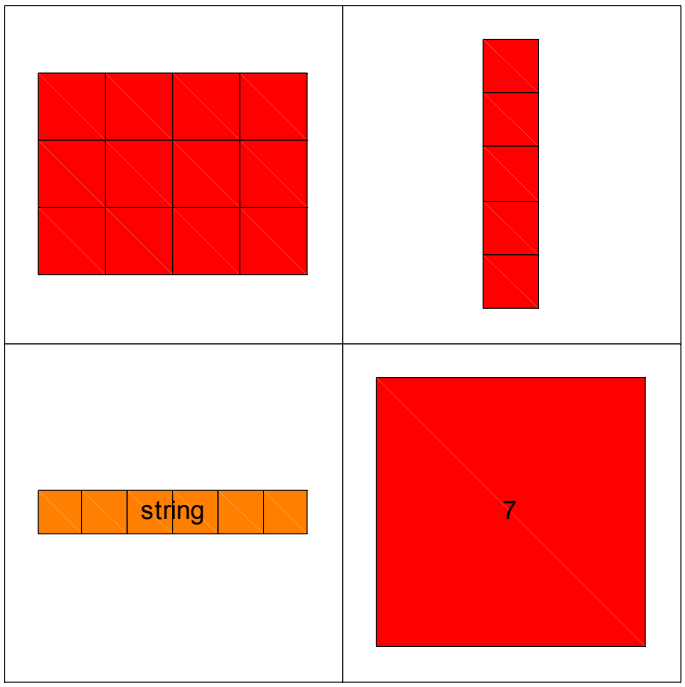
```
C{1,1} =

     2     5     0     9
     1     1     3     7
     6     8     9     5


C{2,1} =

string


C{1,2} =

     2
     2
     2
     2
     2


C{2,2} =

     7
```

[13] מערך תאים ריק מסומן {}.

2. פונקצית cellplot המשרטטת ייצוג גרפי של רכיבי מערך התאים.
לדוגמא,

```
cellplot(C)
```

מייצרת את הגרף,



גרף 4.4: ייצוג של מערך תאים C

חשוב לזכור כי מערך תאים מכיל רק עותקים של מערכים אחרים ולכן שינוי המערכים האחרים לא ישנה את מערך התאים.
למשל בדוגמא הנ"ל שינוי המטריצה A אינו משנה את מערך התאים C.

היתרון המרכזי במערכי תאים היא יכולתם להכיל אובייקטים מסוגים שונים באותו מערך. תכונה זו שימושית, למשל, כאשר יש צורך לשמור מטריצות בגודל משתנה, דבר בלתי אפשרי במערכים נומריים רב ממדיים.

הגישה לכל איבר במערך התאים דומה לגישה לאיברים במטריצה רק שכעת משתמשים בסוגריים מסולסלות.
לדוגמא,

```
>> C{1,1}

ans =

     2     5     0     9
     1     1     3     7
     6     8     9     5

>> C{2,1}

ans =

string
```

כדי לגשת לאיבר (2,3) בתא (1,1) נשרשר את האינדקסים ונרשום,

```
>> C{1,1}(2,3)

ans =

     3
```

מחיקת תאים מבוצעת באופן הבא,

```
C(index) = [];
```

לדוגמא,

```
C(:,2) = [];
```

מוחק את עמודת התאים השניה. כעת המערך C הוא,

```
 C =

    [3x4 double]
    'string'
```

אתחול מערך תאים במטריצות ריקות מבוצע באמצעות פונקצית cell[14].
לדוגמא הפקודה,

```
C = cell(3,4)
```

יוצרת מערך $[3\times4]$ של תאים המכילים מטריצות ריקות,

```
C =

     []     []     []     []
     []     []     []     []
     []     []     []     []
```

באמצעות פונקציה זו ניתן ליצור מערכי תאים רב-ממדיים ע"י הוספת עוד ארגומנטים.
לדוגמא,

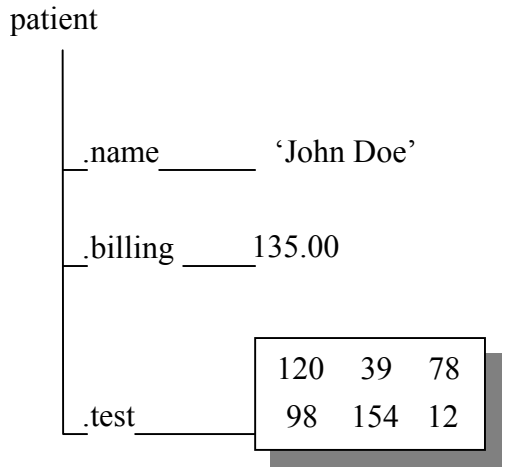
```
C = cell(3,4,5)
```

יוצר מערך תאים $[3\times4\times5]$.

[14] דרך נוספת היא באמצעות פונקצית cat.

4.3 מערכי Structure

מערכי structure דומים למערכי cell בכך שגם הם מאפשרים אכסון של משתנים מטיפוסים וגדלים שונים ומשונים. אך בניגוד למערכי cell, המערכים ב-structure מאוכסנים בתוך שדות של שמות ולא בתוך איברים במערך. כל שדה יכול להכיל סוג שונה של מידע.
לדוגמא,



גרף 4.5: דוגמא למערך structure

בניית מערכי structure נעשית בשני אופנים,
1. באמצעות הצבה. לדוגמא,

```
patient.name = 'John Doe';
patient.billing = 135.00;
patient.test = [120 39 78;...
                98 154 12];
```

מניב את מערך structure בגודל $[1\times1]$ בשם patient הבא,

```
patient =

       name: 'John Doe'
    billing: 135
       test: [2x3 double]
```

כדי ליצור עוד לקוח צריך להרחיב את המערך באופן הבא,

```
patient(2).name = 'Jane Doe';
patient(2).billing = 112.00;
patient(2).test = [163 67 71;...
                   13 18 191];
```

כעת מערך patient הורחב לגודל $[1\times2]$,

```
patient =

1x2 struct array with fields:
    name
    billing
    test
```

2. באמצעות פונקצית struct. הרישום הכללי,

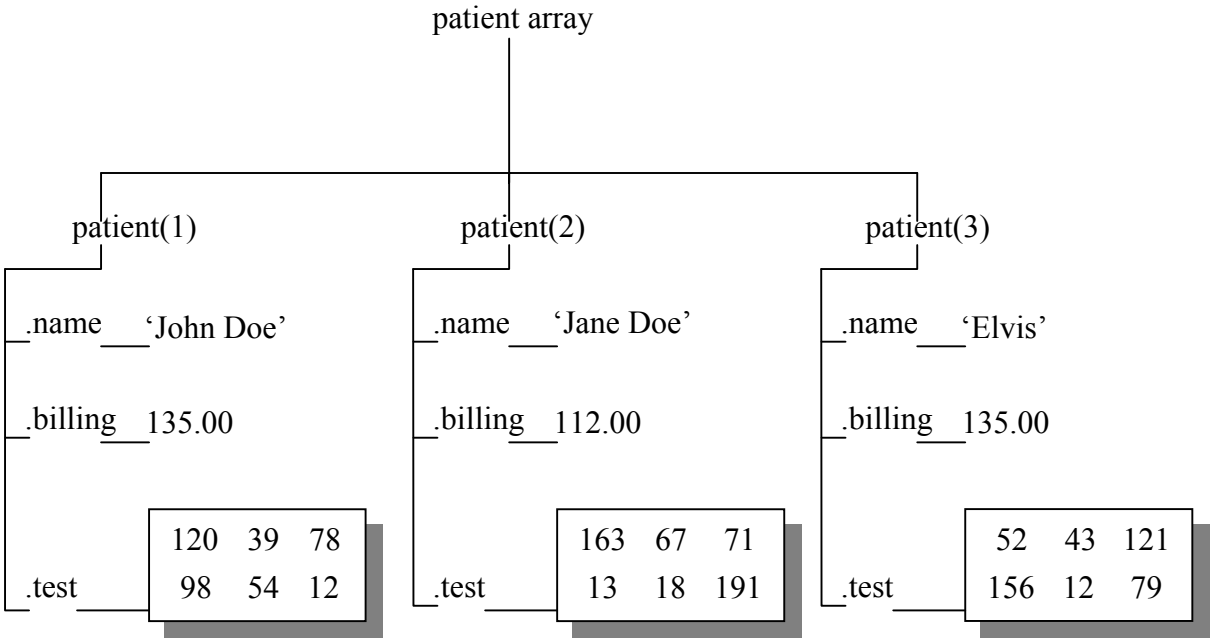
```
struct_array = struct('field1',{values1},'field2',{values2}, ··· )
```

לדוגמא כדי ליצור את אותו מערך patient בגודל $[2\times1]$ נרשום,

```
patient = struct('name',{'John Doe','Jane Doe'},...
                 'billing',{135.00,112.00},...
                 'test',{[120 39 78;98 154 12],...
                         [163 67 71;13 18 191]});
```

4.3.1 גישה למערכי structure

נניח כי נתון מערך ה-structure הבא,



גרף 4.6: דוגמא למערך structure בגודל $[3\times1]$.

ניתן לגשת לכל איבר במערך structure (המכיל ערכים לכל השדות).
לדוגמא,

```
>> patient(1)

ans =

       name: 'John Doe'
    billing: 135
       test: [2x3 double]
```

כדי לגשת לשדה ספציפי הוסיפו נקודה ושם השדה,
לדוגמא,

```
>> patient(1).name

ans =

John Doe
```

כדי לגשת לאיבר בתוך שדה ספציפי הוסיפו אינדקס מתאים.
לדוגמא,

```
>> patient(1).test(1,3)

ans =

    78
```

כדי לקבל שדה שלם נתחום בסוגריים מרובעות את שם המערך, נקודה, ושם השדה הרצוי.
לדוגמא,

```
>> bills = [patient.billing]

bills =

   135   112   157
```

כדי למחוק שדה מ-structure נשתמש בפונקציה rmfield. אופן השימוש הכללי בה הוא,

```
struct_array2 = rmfield(struct_array1,'field');
```

מוגדר מערך structure חדש בשם struct_array2 המכיל את כל השדות שישנם ב-struct_array1 חוץ משדה field.

לדוגמא,
אם ברצוננו להוריד את שדה billing ממערך patient נבצע את הפקודה הבאה,

```
patient = rmfield(patient,'billing');
```

5. גרפיקה דו-ממדית

אחד היתרונות של Matlab היא הצורה הנוחה והמהירה שבה ניתן לייצר גרפים דו-ממדיים ותלת-ממדיים.
פונקציות הגרפיקה מתחלקות לשלושה סוגים,

שליטה בגרפים	יצירת גרפים	סימונים ואפיונים
figure	2-D	legend (2D only)
subplot	plot	text
zoom	fill	title
hold	plotyy	box
		grid
alpha	3-D	axis
shading	plot3	set, get
	surf, surfc	clabel
3-D	mesh, meshz	xlabel
view	contour, contour3, contourf	ylabel
rotate3d	waterfall	
	cylinder	3-D
		zlabel
		colorbar
		colormap

טבלה 5.1: פונקציות גרפיקה

כל פונקציה ליצירת גרפים יוצרת באופן אוטומטי חלון גרפיקה שבו משורטט הגרף. הרצת פונקציה ליצירת גרפים כאשר כבר קיים גרף גורמת למחיקת הגרף הקודם. מסיבה זו ומפני שבמקרים רבים רצוי לייצר מספר גרפים נפרדים משתמשים בפונקציה figure כדי לייצר חלון גרפיקה. הרישום figure(n) מייצר חלון גרפיקה בעל מספר n. הרישום figure ללא מספור יוצר חלון בעל המספר השלם הבא.

הערות

- שמירת גרף מבוצעת ע"י בחירת פונקצית Save או Save as… בתפריט File והגרף ישמר כקובץ fig.
- ניתן להעביר גרפים של Matlab למעבד תמלילים כמו Word באמצעות פונקצית Copy Figure הנמצאת בתפריט Edit. רקע אפור הוא ברירת המחדל לגרפים ב-Matlab. שינוי רקע מבוצע ע"י בחירה בתפריט Edit, Copy Options. למשל כדי להכריח רקע לבן בוחרים באופציה Force white background.

5.1 פונקצית plot

פונקצית plot היא הפונקציה הבסיסית ביותר בשרטוט גרפים דו-ממדיים והיא מיועדת לשרטוט גרפים של נקודות או קווים. הרישום הכללי שלה הוא,

$$plot(x_1, y_1, x_2, y_2, \ldots)$$

כאשר $\{x_i, y_i\}, i = 1, 2, \ldots$ יכולים להיות צמדי סקלרים, צמדי וקטורים בעלי אותו אורך או צמדי מטריצות מאותו גודל[15]. הוספת מחרוזת קצרה לאחר כל צמד מאפשרת שליטה בצבע ובצורת הנקודות או הקווים. באופן כללי הרישום הוא,

$$plot(x_1, y_1, c_1, x_2, y_2, c_2, \ldots)$$

c_i מורכב ממספר תווים (עד לארבעה תווים) המציינים את צבע וסוג הנקודה/קו[16].
למשל המחרוזת 'g' קובעת כי הנקודה/קו יהיו בצבע ירוק (ברירת מחדל היא כחול, ירוק, אדום,...).
למשל המחרוזת 'r*' קובעת כי נקודה בגרף תהיה כוכבית אדומה. למשל המחרוזת 'k--' קובעת כי ישורטט קו מקוטע שחור.
ניתן גם לשלב נקודות עם קווים ע"י ציון שניהם במחרוזת. למשל המחרוזת 'g--s' קובעת כי ישורטט קו ירוק מקוטע עם ריבועים.

טבלה 5.2 מסכמת את כל האפשריות.

צבע		סוג נקודה		סוג קו	
b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star		
y	yellow	s	square		
k	black	d	diamond		
		v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

טבלה 5.2: סיכום אפשריות סימון קווים ונקודות בגרפים

5.1.1 שרטוט גרף של נקודות

קיימות שתי דרכים לשרטט גרפים של נקודות ללא חיבור קווים ביניהם,

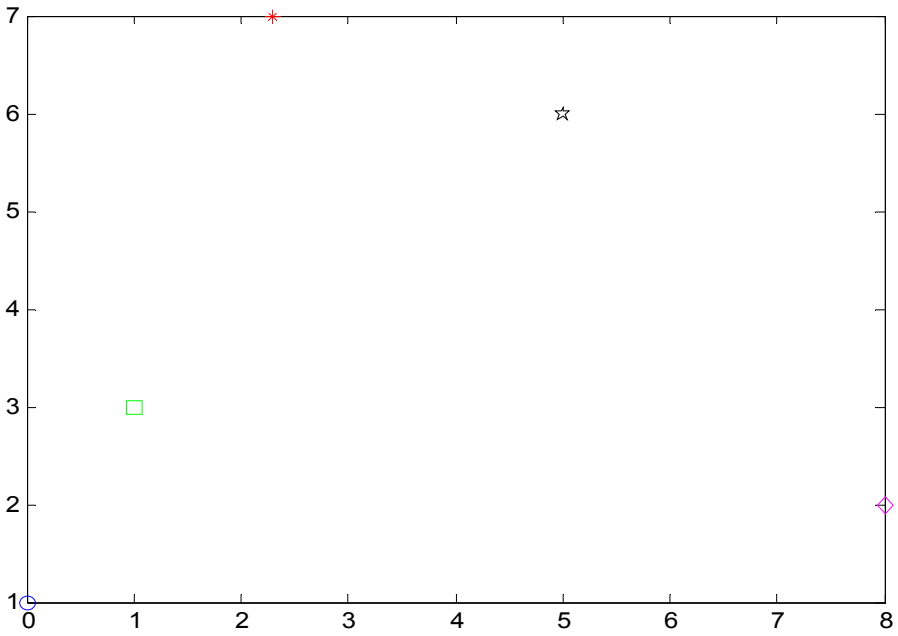
1. הצמדים $\{x_i, y_i\}, i = 1, 2, \ldots$ הם סקלרים

לדוגמא הקוד,

```
plot(0,1,'bo',1,3,'gs',2.3,7,'r*',5,6,'kp',8,2,'md')
```

[15] קיימת גם קומבינציה של וקטור ומטריצה אך זהו מקרה מיוחד שידון בהמשך הפרק.
[16] סדר התווים במחרוזת c איננו משנה.

מניב את גרף 5.1.



גרף 5.1: גרף נקודות

בגרף 5.1 ניתן לראות כי Matlab התמקד על הנתונים כך שחלק מהנקודות הן בדיוק על הגבול ולכן קשה להבחין בהן. ניתן לשלוט ידנית בתצוגת מערכת הצירים ע"י שימוש בפונקצית axis הקובעת את תחום הצגת הגרף.
רישומה הכללי הוא,

$$\text{axis}([x_{min}, x_{max}, y_{min}, y_{max}])$$

כאשר,

x_{min} - ערך מינימלי של ציר x היופיע בגרף

x_{max} - ערך מכסימלי של ציר x היופיע בגרף

y_{min} - ערך מינימלי של ציר y היופיע בגרף

y_{max} - ערך מכסימלי של ציר y היופיע בגרף

כדי ל"רווח" את התצוגה בגרף 5.1 נוסיף את הקוד,

```
axis([-1 9 0 8])
```

וגרף 5.2 מתקבל.

אם ברצוננו לשנות רק חלק מגבולות הצירים ניתן לבצע זאת באופן הבא[17],
תחילה להציב בוקטור כלשהו את גבולות הצירים,

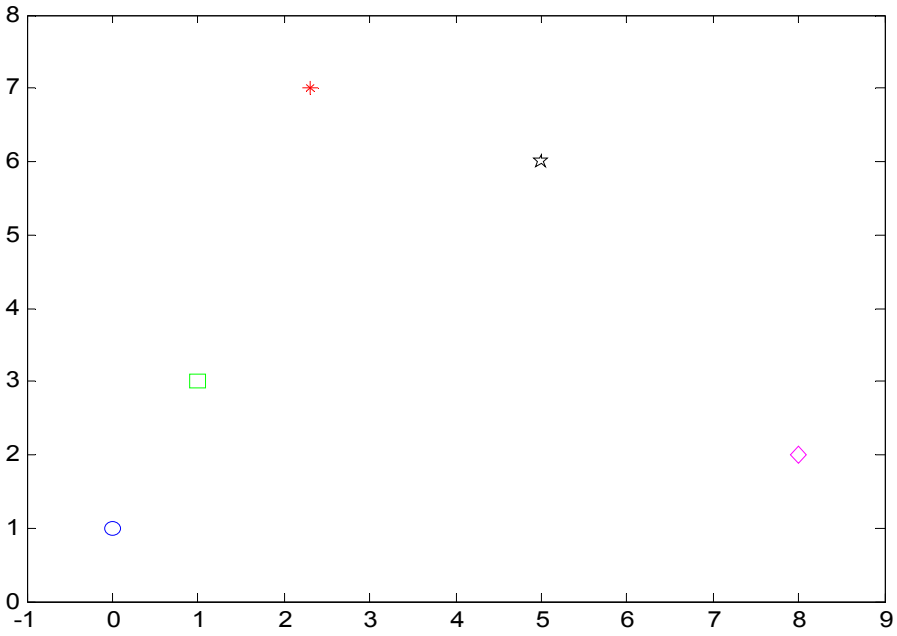
```
tmp = axis;
```

$$tmp \leftarrow [x_{min}, x_{max}, y_{min}, y_{max}]$$

לשנות בוקטור את הקואורדינטות הרצויות ולהחזיר ל-axis את הוקטור המעודכן,

```
axis(tmp)
```

[17] ניתן לבצע את ה"טריק" הנ"ל גם לגרף תלת-ממדי ע"י הוספת עוד שני משתנים z_{min}, z_{max} כדי לתחום את ציר z.

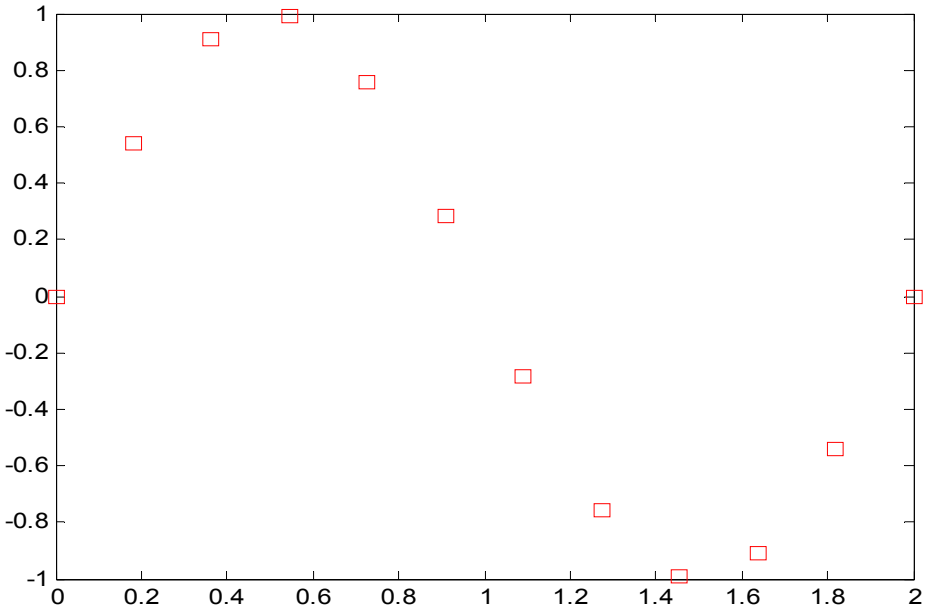


גרף 5.2: גרף נקודות שתצוגת מערכת הצירים שלו שונתה באמצעות פונקצית axis

2. הצמדים $\{x_i, y_i\}, i = 1,2,\ldots$ הם וקטורים + ציון סוג הנקודה אך אי ציון סוג הקו[18]
הוקטור הראשון בצמד מהווה את קואורדינטות ה-x והוקטור השני מהווה את קואורדינטות ה-y.
לדוגמא הקוד,

```
x = linspace(0,2,12);
y = sin(pi*x);
plot(x,y,'rs')
```

מניב את גרף 5.3.



גרף 5.3: שרטוט גרף נקודות ע"י ציון סוג הנקודות

[18] ציון הצבע איננו משפיע על קביעה אם הגרף יהיה גרף נקודות או קווים.

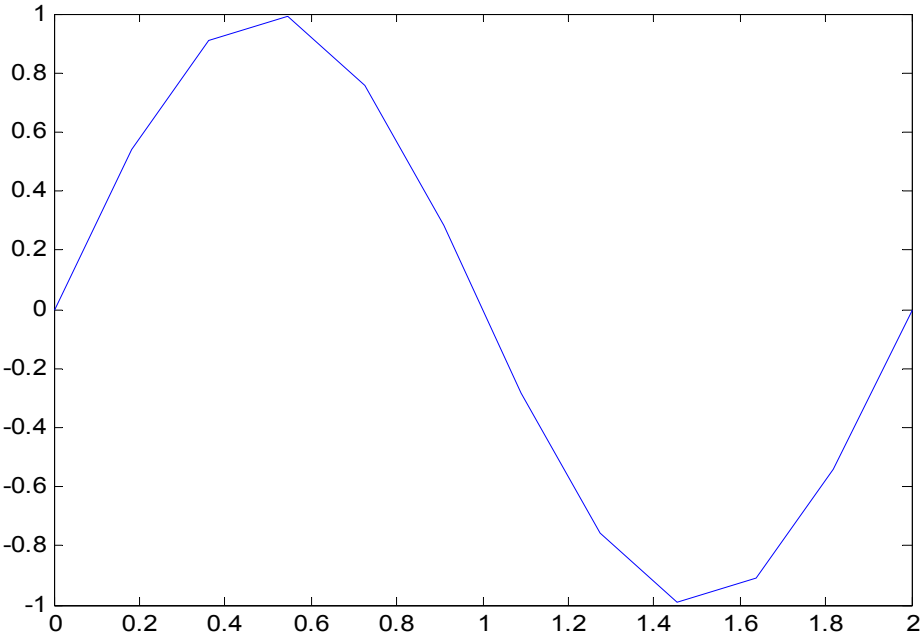
היתרון של הרישום הוקטורי לעומת הרישום הסקלרי הוא שאין צורך לרשום כל איבר בוקטור בנפרד (הופך לבלתי אפשרי עבור וקטורים ארוכים). החסרון של הרישום הוקטורי הוא חוסר שליטה על הצורה והצבע של כל נקודה בנפרד.

5.1.2 יצירת גרף של קווים

ברירת המחדל ליצירת גרפים היא גרפים של קווים. לכן כאשר הצמדים $\{x_i, y_i\}, i = 1,2,\ldots$ הם וקטורים ולא מצוין במפורש סוג הנקודה (האפשרות השניה בפרק 5.1.1) אז Matlab יחבר קווים ישרים בין הנקודות.
לדוגמא הקוד,

```
plot(x,y)
```

מניב את גרף 5.4.



גרף 5.4: גרף של קווים

דוגמא נוספת. הקוד,

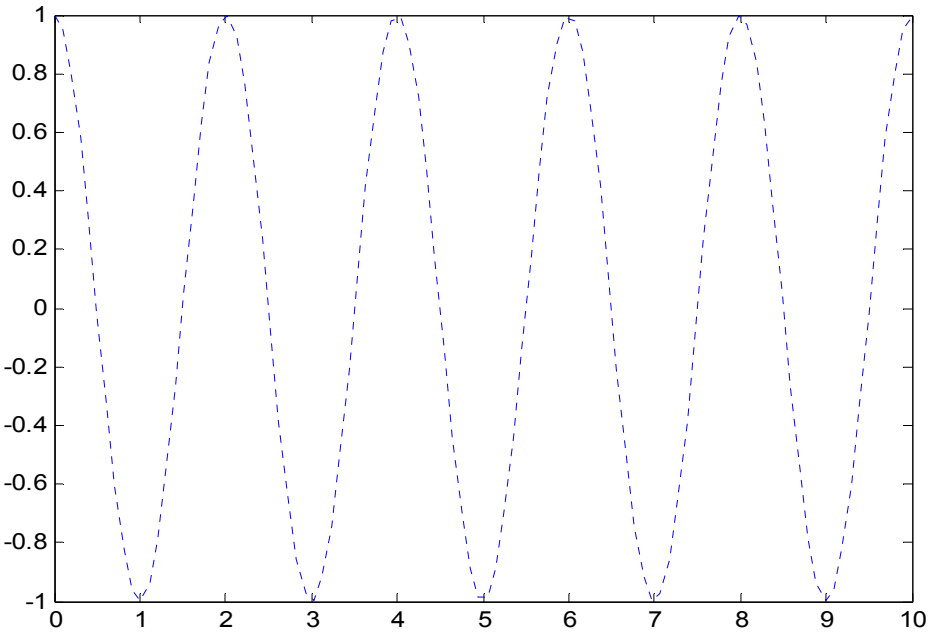
```
x = linspace(0,10,100);
y = cos(pi*x);
plot(x,y,':')
```

מניב את גרף 5.5.

ניתן גם לשרטט גרפים לא רק כנגד ציר-x דגום בצורה לינארית ואחידה. דוגמא אחת היא ציור מעגל. הטרנספורמציה המתמטית לקואורדינטות קרטזיות היא,

$$x = a + r\cos(\theta)$$

$$y = b + r\sin(\theta)$$



גרף 5.5: שרטוט פונקציה בקו מקוקו

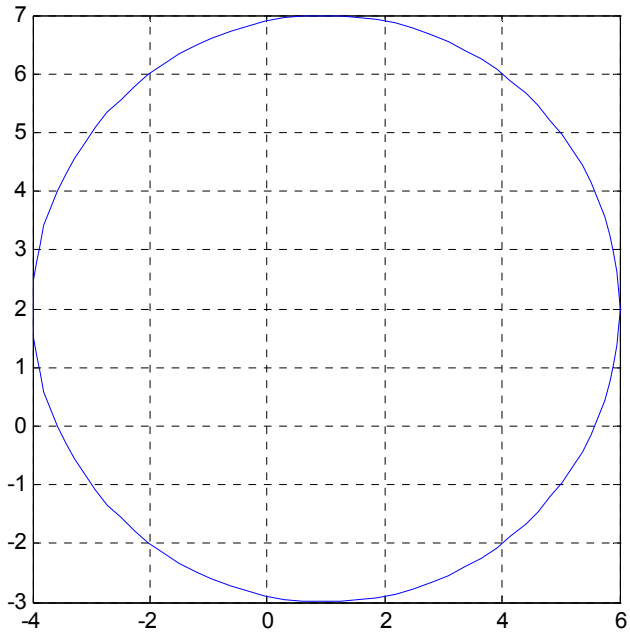
כאשר $0 \le \theta \le 2\pi$. מרכז המעגל הוא בנקודה (a,b) ורדיוסו הוא r.
הקוד,

```
a = 1;
b = 2;
r = 5;

theta = linspace(0,2*pi,100);
plot(a + r*cos(theta),b + r*sin(theta))
grid
axis square
```

מניב את גרף 5.6.

פונקציית grid יוצרת רשת קווים המקבילים לצירים. פונקציית axis square קובעת את הפרופורציות של הגרף כך ששני הצירים יראו באותו גודל. ניתן גם להוריד את תצוגת מערכת הצירים באמצעות פונקציית axis off.[19]

[19] ב-help של פונקציית axis מפורטות עוד מספר אפשריות שימושיות כמו axis tight ו-axis equal.



גרף 5.6: יצירת מעגל

5.1.3 שרטוט מספרים קומפלקסים

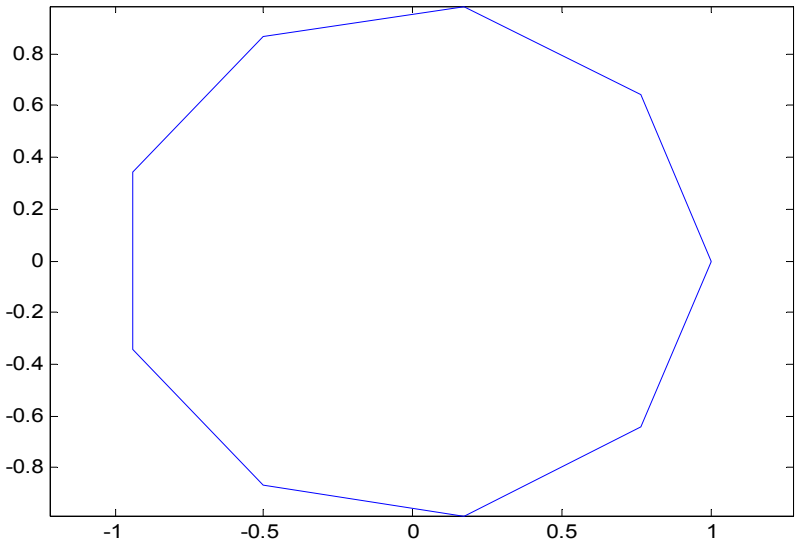
אם הקלט של פונקצית plot הוא מספרים קומפלקסים אז הפונקציה מתעלמת מהחלק המדומה ומשרטטת רק את החלק הממשי, למעט מקרה יוצא דופן יחיד. אם הקלט הוא ארגומנט קומפלקסי יחיד אז פונקצית plot משרטטת את החלק הממשי לעומת החלק המדומה.
כלומר,

```
plot(Z)
 ⇕
plot(real(Z),imag(Z))
```

לדוגמא הקוד,

```
t = linspace(0,2*pi,10);
Z = exp(j*t);
plot(Z)
axis equal
```

מניב את גרף 5.7.



גרף 5.7: שרטוט ערכים קומפלקסים

5.2 שרטוט מספר פונקציות בגרף בודד

קיימות 3 אפשריות לשרטוט מספר פונקציות בגרף בודד.

5.2.1 ציר x וקטור ,מטריצה מכילה את וקטורי הפונקציות

הרישום הכללי הוא,

$$\text{plot}(x,[y_1,y_2,\ldots,y_N])$$

או

$$\text{plot}(x,[y_1;y_2;\ldots;y_N])$$

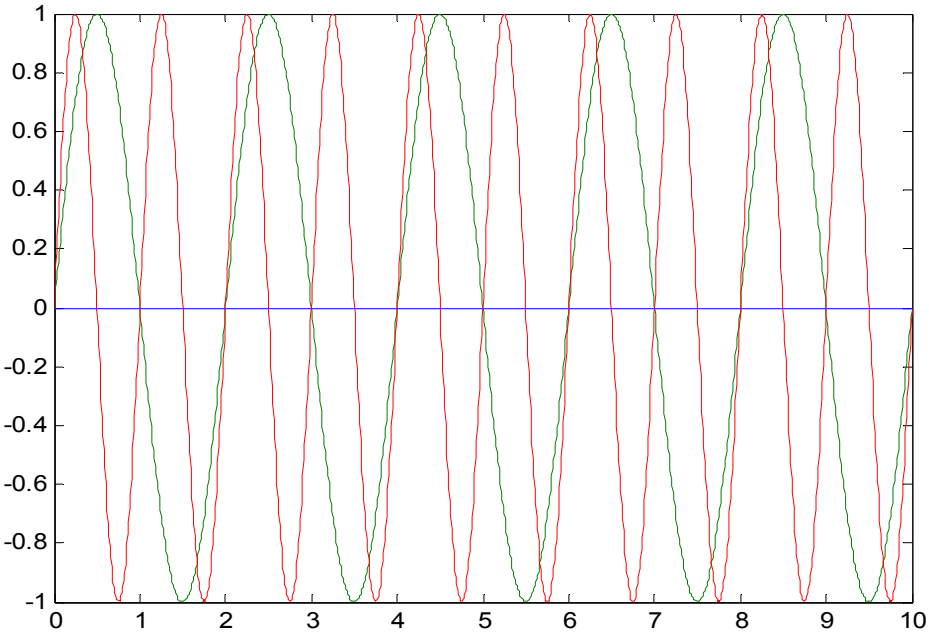
כאשר הצמדים $\{x_i,y_i\},i=1,2,\ldots$ הם וקטורים באותו אורך.

לעיתים רבות רצוי לשרטט מספר פונקציות לעומת אותו ציר x. כדי לבצע זאת ניתן להכניס בארגומנט הראשון את ציר ה-x כוקטור, ואת הפונקציות בתור וקטורי עמודות או שורות במטריצה. פונקצית plot משרטטת את הפונקציות לפי הממד שתואם באורכו את וקטור ציר ה-x.

דוגמא,
שרטוט על גרף יחיד מספר סינוסים בתדרים שונים $(0,\pi,2\pi)$. הקוד,

```
x = linspace(0,10,1000);
w = linspace(0,2*pi,3);
[xx,ww] = meshgrid(x,w);
sin_mat = sin(ww.*xx);
plot(x,sin_mat)
```

מניב את גרף 5.8.



גרף 5.8: שרטוט מספר פונקציות לעומת אותו ציר x (ציר x וקטור ,מטריצה מכילה את וקטורי הפונקציות)

ניתן לקבוע את כל הצבעים לצבע אחיד ע"י כתיבה,

```
plot(x,sin_mat,'b')
```

הערה

- אם פונקצית plot מקבלת בקלט רק מטריצה היא מתייחסת לכל וקטור עמודה בתור גרף ומשרטטת כל וקטור עמודה בצבע שונה על אותו הגרף. ברישום זה אין שליטה על מערכת ציר-x (כאשר ציר x איננו מוגדר אז הוא 1:length(y)).

5.2.2 צמדי וקטורים מיצגים את צירי x ווקטורי הפונקציות

הרישום הכללי הוא,

$$\text{plot}(x_1, y_1, x_2, y_2, \ldots)$$

כאשר הצמדים $\{x_i, y_i\}, i = 1, 2, \ldots$ הם וקטורים באותו אורך.

לדוגמא,
נתונות שלוש פונקציות,

$$\begin{cases} f(x) = 3 + \sin(\pi x) \\ g(x) = \dfrac{x}{2} + 1 \\ h(x) = \left(\ln(x+1)\right)^3 \end{cases}$$

בתחום $0 \le x \le 5$.

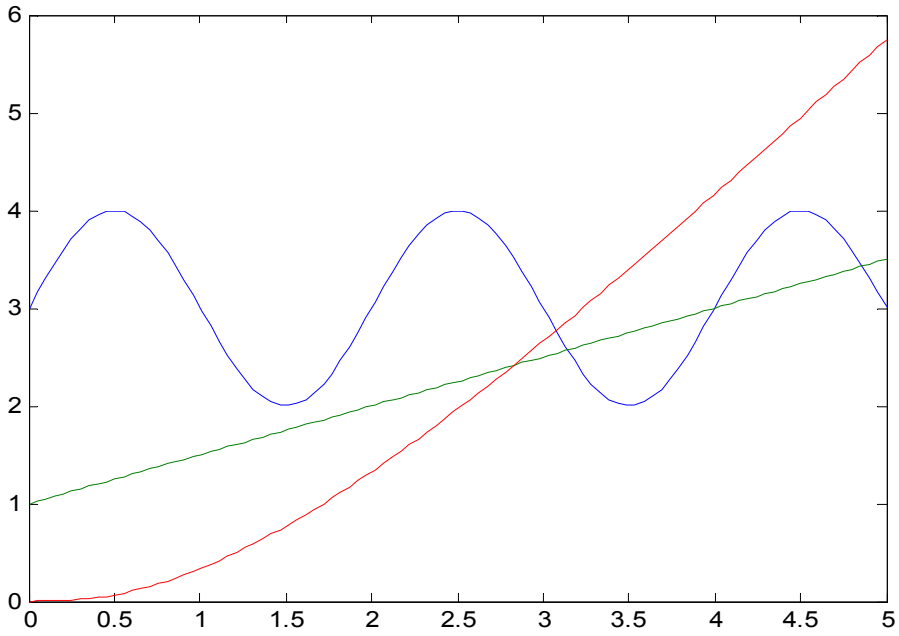
נחשב תחילה,

```
x = linspace(0,5,100);
f = 3 + sin(pi*x);
g = x/2 + 1;
h = (log(x + 1)).^3;
```

נשרטט את שלושת הפונקציות באותו הגרף ע"י רישום כל פונקציה בתור צמד של וקטור x ווקטור ערכי הפונקציה בנקודות אלו. הקוד,

```
plot(x,f,x,g,x,h)
```

מניב את גרף 5.9.



גרף 5.9: שרטוט מספר פונקציות לעומת אותו ציר x (צמדי וקטורים מיצגים את צירי x וקטורי הפונקציות)

במקרה זה כל הפונקציות משורטטות לעומת אותו וקטור x ולכן הרישום,

```
plot(x,[f;g;h])
```

מניב את אותו גרף.

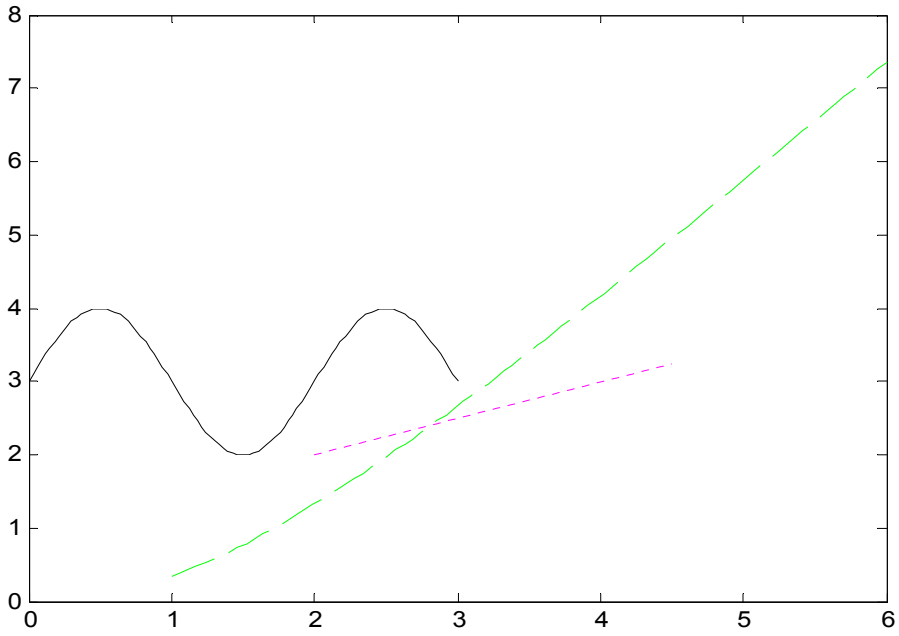
ברישום זה ניתן גם לשרטט מספר פונקציות לעומת משתנים שונים.
בהמשך לדוגמא הנ"ל,

$$\begin{cases} f(x) = 3 + \sin(\pi x), & 0 \le x \le 3 \\ g(y) = \dfrac{y}{2} + 1 & , \ 2 \le y \le 4.5 \\ h(z) = \left(\ln(z+1)\right)^3, & 1 \le y \le 6 \end{cases}$$

הקוד,

```
x = linspace(0,3,100);
y = linspace(2,4.5,54);
z = linspace(1,6,87);

f = 3 + sin(pi*x);
g = y/2 + 1;
h = (log(z + 1)).^3;

plot(x,f,'k',y,g,'m:',z,h,'g--')
```

מניב את גרף 5.10.



גרף 5.10: שרטוט מספר פונקציות לעומת צירי x שונים

נשים לב כי בגרף הנ"ל שרטטנו כל פונקציה במערכת צירים בתחום שונה, מספר נקודות שונה וצבעים וצורות שונים.

5.2.3 פונקצית hold

אופן השימוש בפונקצית hold,

$$\begin{aligned}&\text{plot}(x_1, y_1)\\&\text{hold on}\\&\text{plot}(x_2, y_2)\\&\vdots\\&\text{plot}(x_N, y_N)\\&\text{hold off}\end{aligned}$$

ללא שימוש בפונקצית hold כל פונקצית המייצרת גרף תחליף את הגרף הקודם. הפקודה hold on גורמת לשמירת גרף קודם והשמת כל גרף חדש על אותו גרף ומערכת צירים. כדי להפסיק השמת הגרפים הבאים על אותו גרף ומערכת צירים משתמשים בפקודת hold off. כמו כן, ניתן לעבור ממצב on ל-off ולהפך באמצעות כתיבת hold המחליף את המצב בכל פעם.

לדוגמא הקוד,

```
plot(x,f,'k')
hold on
plot(y,g,'m:')
plot(z,h,'g--')
hold off
```

מניב את גרף 5.10.

5.2.4 סיכום יתרונות וחסרונות של צורות רישום

צורת רישום	משתנה בלתי תלוי זהה לכל הפונקציות	משתנים בלתי תלוים שונים לפונקציות	שילוב של סוגי גרפים שונים
1. וקטור ומטריצה	יתרון: רישום קומפקטי חסרון: אי אפשר לשלוט בסוג ובצבע של כל פונקציה בגרף בנפרד	לא אפשרי ברישום הנ"ל	לא אפשרי ברישום הנ"ל
2. צמדי וקטורים	יתרון: ניתן לשלוט בסוג ובצבע של כל פונקציה בגרף בנפרד חסרון: רישום מסורבל	יתרון: רישום קומפקטי	לא אפשרי ברישום הנ"ל
3. פונקצית hold	יתרון: ניתן לשלוט בסוג ובצבע של כל פונקציה בגרף בנפרד חסרון: רישום מסורבל	חסרון: רישום מסורבל	רק בצורת הרישום הזו אפשרי

טבלה 5.3: סיכום יתרונות וחסרונות של צורות רישום

5.3 הוספת סימונים לגרפים

בעת הצגת תוצאות באמצעות גרפים מקובל גם להוסיף סימונים להבהרת התוצאות. Matlab מאפשר הוספת הסימונים הבאים,

- כותרת לגרף
- כותרות לצירי[20] x ו-y
- תיבת מקרא
- טקסט בכל מקום בגרף
- קווים וחצים בכל מקום בגרף
- סטטיסטיקה בסיסית כמו מקסימום, מינימום וממוצע

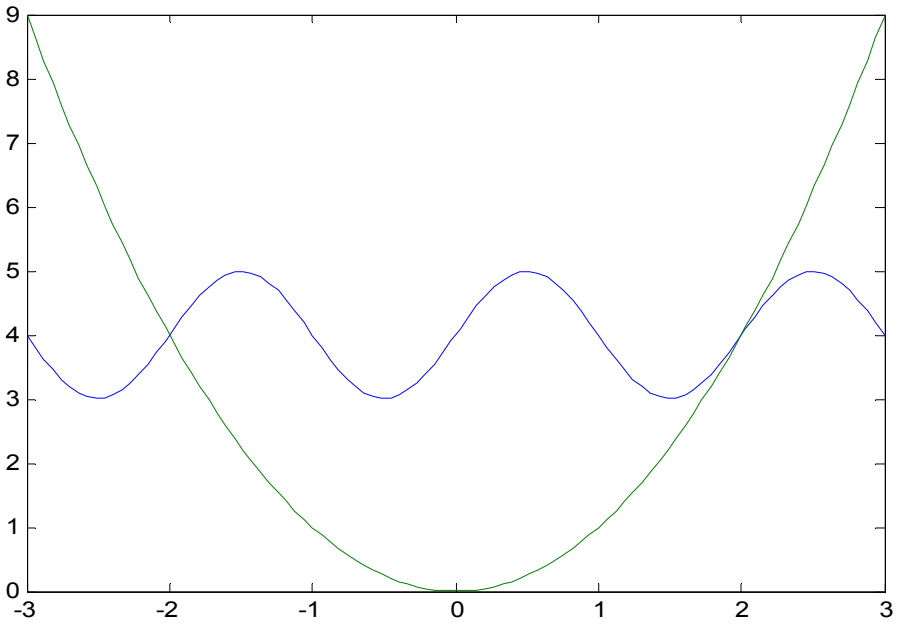
בפרק 5.3 נשתמש בדוגמא הבאה להמחיש הוספת סימונים לגרפים.
נתונות שתי פונקציות,

$$\begin{cases} f(x) = \sin(\pi x) + 4 \\ g(x) = x^2 \end{cases}$$

בתחום $-3 \le x \le 3$.
הקוד,

```
x = linspace(-3,3,100);
f = sin(pi*x)+4;
g = x.^2;

plot(x,[f;g])
```

מניב את גרף 5.11. גרף זה מתאר את שתי הפונקציות הנ"ל ללא תוספת סימונים.



גרף 5.11: גרף ללא סימוני עזר

[20] עבור גרף תלת ממדי גם עבור ציר z.

5.3.1 רישום סימנים מיוחדים

בפקודות להוספת כותרות והשמת טקסט ניתן ליצור אותיות יווניות, כתב עילי, כתב תחתי וסמלים. כתב עילי מיוצר באמצעות ^, כתב תחתי באמצעות _. אותיות יווניות באמצעות \ וכתיבת שם האות היוונית באנגלית. לדוגמא α מיוצרת ע"י כתיבת \alpha. יצירת אותיות יווניות גדולות[21] היא כמו כתיבת אותיות קטנות, רק שהאות הראשונה של שם האות היוונית צריך להיות אות אנגלית גדולה. לדוגמא Γ מיוצרת ע"י כתיבת \Gamma. טבלה 5.4 מסכמת את כל הסמלים האפשריים ליצור באמצעות \.

לעיתים רבות רצוי לאגד מספר סמלים יחד, כמו ביטוי של אקספוננט. ניתן לעשות זאת באמצעות סוגריים מסולסלות {}. לדוגמא הביטוי $e^{\alpha x_1^2}$ נכתוב e^{\alphax_1^2}.

5.3.2 הוספת כותרת

הוספת כותרת מבוצעת ע"י הצבת מחרוזת בקלט של פונקצית title.
לדוגמא,

```
title('An example of graph annotation')
```

5.3.3 הוספת כותרות לציר x ו-y

הוספת כותרת לציר x מבוצעת ע"י הצבת מחרוזת בקלט של פונקצית xlabel. לדוגמא,

```
xlabel('x')
```

הוספת כותרת לציר y מבוצעת ע"י הצבת מחרוזת בקלט של פונקצית ylabel. לדוגמא,

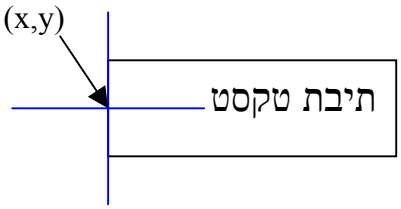
```
ylabel('y')
```

5.3.4 הוספת טקסט

הוספת טקסט בקואורדינטות (x, y) בגרף מבוצעת ע"י הצבת קואורדינטות תיבת הטקסט ומחרוזת הטקסט הרצויה בקלט של פונקצית text.
לדוגמא,

```
text(-2.5,7,'\leftarrowx^2')
text(-0.9,4.5,'sin(\pi\cdotx)+4\rightarrow')
```

בתור ברירת מחדל[22] פונקצית טקסט מציבה את המחרוזת מימין לקואורדינטת x ובמרכז קואורדינטת y.



כדי לדעת איפה בדיוק להציב את הטקסט בצורה זו אין ברירה אלא לבצע מעט ניסוי וטעייה.

[21] בטבלה לא מצוינות האותיות היווניות הגדולות שזהות לאותיות אנגליות גדולות. למשל האות אלפא גדולה היא A.
[22] אפשריות אחרות באמצעות פונקציות HorizontalAlignment ו-VerticalAlignment .

Character Sequence	Symbol	Character Sequence	Symbol	Character Sequence	Symbol
\alpha	α	\upsilon	υ	\sim	~
\beta	β	\phi	ϕ	\leq	≤
\gamma	γ	\chi	χ	\infty	∞
\delta	δ	\psi	ψ	\clubsuit	♣
\epsilon	ε	\omega	ω	\diamondsuit	♦
\zeta	ζ	\Gamma	Γ	\heartsuit	♥
\eta	η	\Delta	Δ	\spadesuit	♠
\theta	θ	\Theta	Θ	\leftrightarrow	↔
\vartheta	ϑ	\Lambda	Λ	\leftarrow	←
\iota	ι	\Xi	Ξ	\uparrow	↑
\kappa	κ	\Pi	Π	\rightarrow	→
\lambda	λ	\Sigma	Σ	\downarrow	↓
\mu	μ	\Upsilon	Υ	\circ	°
\nu	ν	\Phi	Φ	\pm	±
\xi	ξ	\Psi	Ψ	\geq	≥
\pi	π	\Omega	Ω	\propto	∝
\rho	ρ	\forall	∀	\partial	∂
\sigma	σ	\exists	∃	\bullet	•
\varsigma	ς	\ni	∋	\div	÷
\tau	τ	\cong	≅	\neq	≠
\equiv	≡	\approx	≈	\aleph	ℵ
\Im	ℑ	\Re	ℜ	\wp	℘
\otimes	⊗	\oplus	⊕	\oslash	⊘
\cap	∩	\cup	∪	\supseteq	⊇
\supset	⊃	\subseteq	⊆	\subset	⊂
\int	∫	\in	∈	\o	o
\rfloor	⌋	\lceil	⌈	\nabla	∇
\lfloor	⌊	\cdot	·	\ldots	...
\perp	⊥	\neg	¬	\prime	′
\wedge	∧	\times	x	\0	∅
\rceil	⌉	\surd	√	\mid	|
\vee	∨	\varpi	ϖ	\copyright	©
\langle	⟨	\rangle	⟩		

טבלה 5.4: סיכום סמלים הניתנים ליצור באמצעות \

5.3.5 הוספת תיבת מקרא

הוספת תיבת מקרא לגרף מבוצעת ע"י הצבת מחרוזות, המיצגות את שמות הפונקציות בגרף, בקלט של פונקצית legend. בתיבת המקרא יוצג ליד כל שם פונקציה דוגמא של סגנון השרטוט. לדוגמא,

```
legend('sin(\pi\cdotx)+4','x^2')
```

פונקצית legend בתור ברירת מחדל מציבה את תיבת המקרא בפינה הימנית העליונה. לעיתים רצוי להציבה במקומות אחרים בגרף שפחות מסתירים את המידע בגרף. ניתן להזיז את תיבת המקרא באופן ידני (באמצעות Unlock Axes Position) או להוסיף בקלט עוד משתנה המקבל את הערכים הבאים,

ערך	פעולה
1	מיקום תיבת המקרא בפינה הימנית עליונה (ברירת המחדל)
2	מיקום תיבת המקרא בפינה השמאלית עליונה
3	מיקום תיבת המקרא בפינה השמאלית תחתונה
4	מיקום תיבת המקרא בפינה הימנית תחתונה
0	מיקום תיבת המקרא בתוך גבולות הגרף, במיקום שבו התיבה מסתירה כמה שפחות
-1	מיקום תיבת המקרא מחוץ לגבולות הגרף בנפרד, בפינה הימנית עליונה

טבלה 5.5: אפשריות מיקום legend

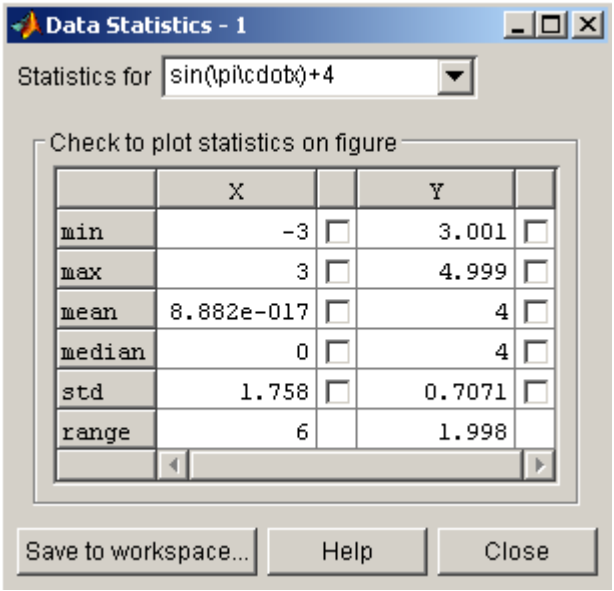
לדוגמא הקוד,

```
legend('sin(\pi\cdotx)+4','x^2',0)
```

מציב את תיבת המקרא בחלק המרכזי עליון כך שתיבת המקרא לא תסתיר דבר.

5.3.6 הוספת סטטיסטיקה בסיסית

חלון Data Statistics מחשב מספר ביטויים סטטסטיים של הפונקציות ובאמצעותו ניתן בקלות להוסיפם לגרף. חלון זה נפתח ע"י לחיצה על Tools ו-Data Statistics כפי שמודגם בגרף 5.12.



גרף 5.12: חלון Data Statistics

חלון זה מציג לכל פונקציה בגרף ערכי מקסימום, מינימום, ממוצע ועוד. כדי לשרטט את המידע הנ"ל על הגרף פשוט סמנו את התיבה המתאימה.
לדוגמא, כדי לסמן את התחום בו משתנה ערך הפונקציה $f(x)=\sin(\pi x)+4$ נסמן ערכי min ו-max עבור Y של פונקציה זו.

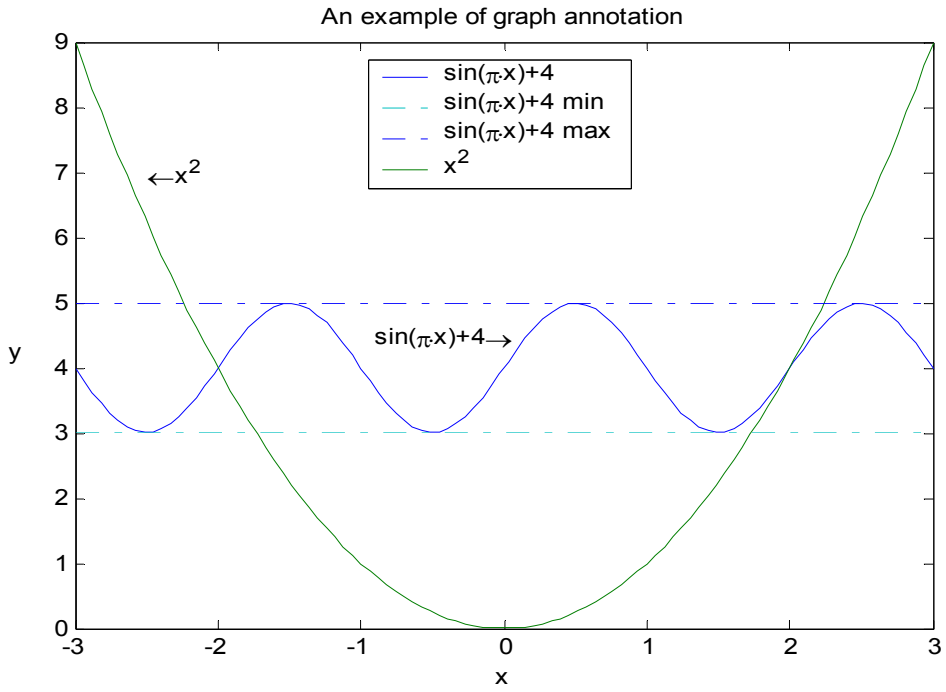
5.3.7 הוספת קווים וחצים

ניתן לייצר חצים בפונקצית text באמצעות האופרטור \. אך חצים אלו יכולים להיות בכיוון ימינה, שמאלה, למעלה ולמטה. שרטוט קווים וחצים כלשהם מבוצע באופן ידני (ראו פרק 7.1).

5.3.8 סיכום הוספת סימונים לגרפים

גרף 5.13 מציג את כל תוספות הסימונים בפרק 5.3[23].

כל פקודות הוספת הסימונים חייבות להיכתב לאחר פונקצית plot וסדר כתיבתן אינו משנה.



גרף 5.13: גרף עם סימוני עזר

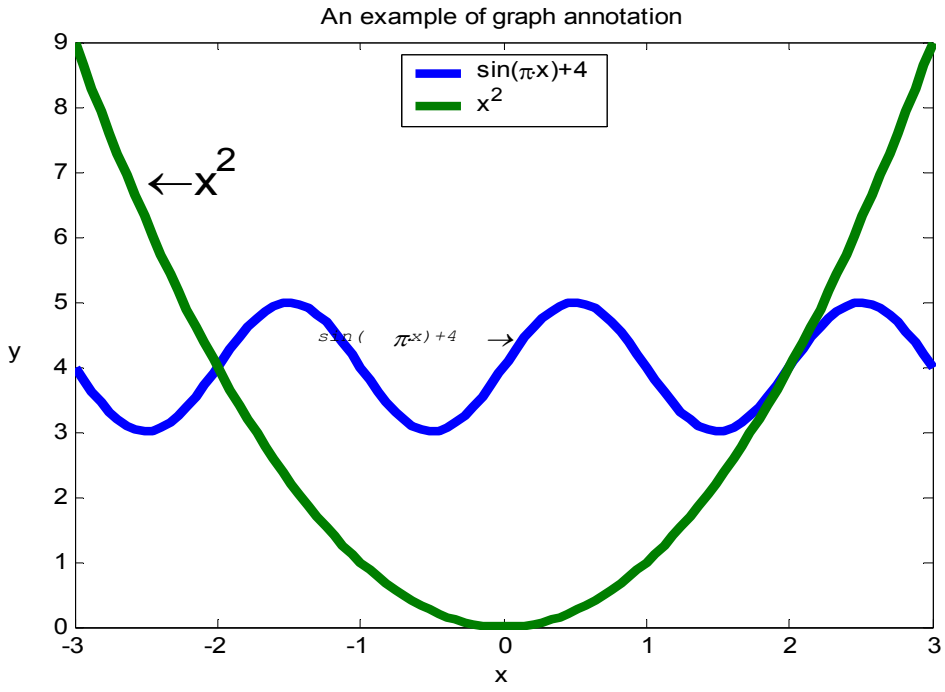
[23] הטקסט לערכי מינימום ומכסימום של הפונקציה $f(x)=\sin(\pi x)+4$ בתיבת המקרא שונו באופן ידני.

5.3.9 קביעת תכונות של אובייקטים גרפיים באמצעות פקודות

כל גרף מורכב ממספר אובייקטים גרפיים, ולכל אובייקט מספר רב של תכונות הניתנות לשינוי. הדרך הפשוטה לשנות תכונות של אובייקטים של גרפיקה באמצעות פקודות היא להוסיף בקלט של פונקציה היוצרת אובייקט גרפי שני משתנים נוספים: מחרוזת של שם התכונה וערך החדש של תכונה זו.
לדוגמא הקוד,

```
plot(x,[f;g],'LineWidth',4)
title('An example of graph annotation')
xlabel('x'),ylabel('y')
text(-2.5,7,'\leftarrowx^2','FontSize',18)
text(-1.3,4.5,'sin(\pi\cdotx)+4\rightarrow','FontName','Courier','FontAngle','Italic')
legend('sin(\pi\cdotx)+4','x^2',0)
```

מניב את גרף 5.14.



גרף 5.14: גרף שחלק מתכונות אובייקטים של גרפיקה שלו שונו

הערות

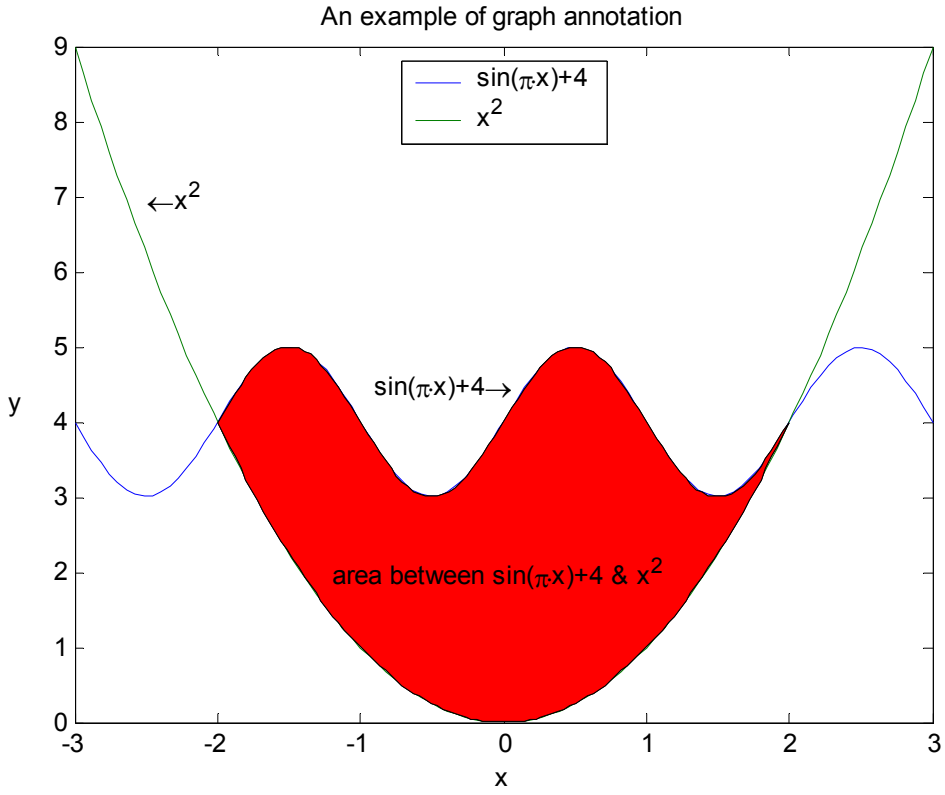
- הדרך המסובכת יותר היא שימוש בפונקציות get ו-set כפי שמפורט בפרק 7.2.
- כדי לראות רשימה של כל אובייקטים של גרפיקה ותכונותיהם התבוננו בקובץ העזרה Handle Graphics Online Documentation (בפורמט html).

5.4 פונקצית fill

ניתן למלא אזור בגרף בין שתי פונקציות באמצעות פונקצית fill. למשל בדוגמא הנ"ל רצוי למלא את האזור בין הסינוס והפרבולה באזור $-2 \le x \le 2$ בצבע אדום.
הקוד,

```
hold
x_fill = linspace(-2,2,100);
f_fill = sin(pi*x_fill)+4;
g_fill = x_fill.^2;
fill([x_fill fliplr(x_fill)],[f_fill fliplr(g_fill)],'r')
text(-1.2,2,'area between sin(\pi\cdotx)+4 & x^2')
hold
```

מניב את גרף 5.15.



גרף 5.15: הדגמת פונקצית fill

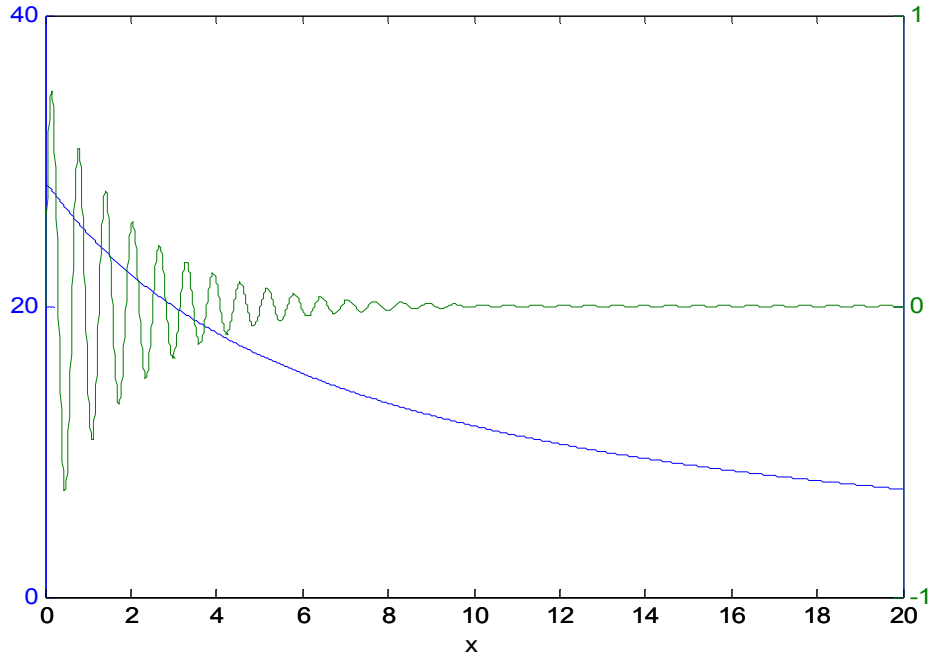
הקלט של פונקצית fill הוא נקודות היוצרות אזור סגור. כלומר נקודות לאורך גבול האזור התחום (לכן נקודת ההתחלה תהיה זהה לנקודת הסיום). בדוגמא זו השרשור בציר y בוצע ע"י הכנסת הפונקציה העליונה תחילה ואח"כ בסדר הפוך את הפונקציה התחתונה. לכן השרשור בציר x בוצע ע"י השמת וקטור x ואחריו את וקטור x בסדר הפוך.

5.5 פונקצית plotyy

פונקצית plotyy מאפשרת שרטוט פונקציות בעלות שני צירי y שונים על אותו גרף. שימוש אפשרי לגרף מסוג זה הוא שרטוט שני גדלים פיסקליים שונים (יחידות שונות) כפונקציה של אותו משתנה בלתי תלוי.
לדוגמא הקוד,

```
x = 0:0.01:20;
y1 = 200./(x+7);
y2 = 0.8*exp(-0.5*x).*sin(10*x);
plotyy(x,y1,x,y2)
xlabel('x')
```

מניב את גרף 5.16.



גרף 5.16: הדגמת פונקצית plotyy

נשים לב כי הסקלה של y_1 ושל y_2 שונות. כדי לציין ylabel שונה לכל אחד, או לשנות את סגנון או צבע הקווים יש להיעזר בפונקצית set וב-help של plotyy.

5.6 קריאת ערכים ישירות מהגרפים

קריאת ערכים ישירות מגרף מבוצעת ע"י כתיבת,

```
[x,y] = ginput
```

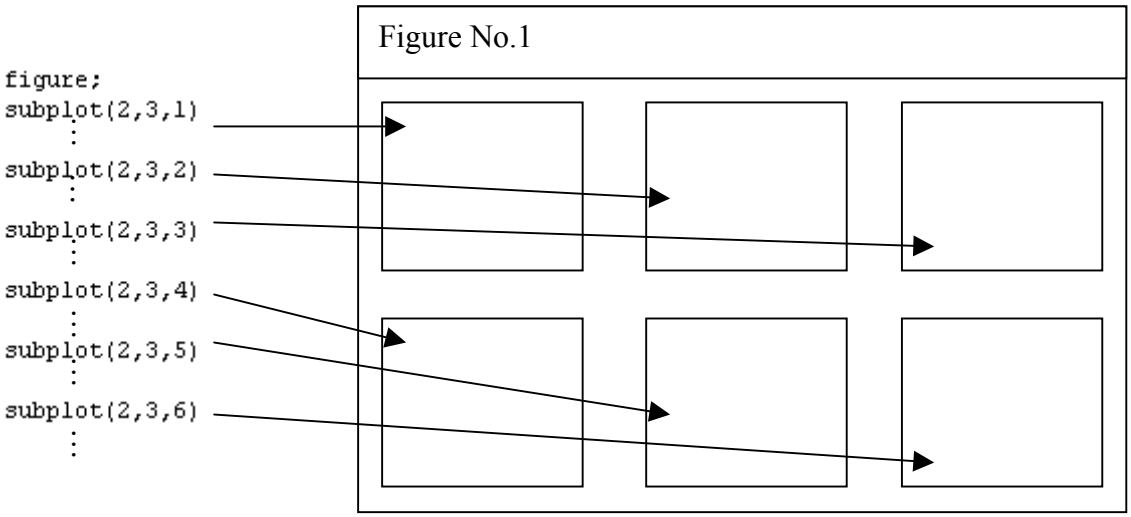
ברגע שלוחצים על Enter ב-Command Window (או ששורה זו מבוצעת בקובץ script) עולה הגרף האחרון אוטומטית ומופיע צלב סביב סמן העכבר. לחיצה על כפתור העכבר הימני או השמאלי בנקודה מסוימת בגרף מציבה את הקואורדינטות של נקודה זו לוקטורים x,y. כל לחיצה נוספת מכניסה עוד צמד קואורדינטות לוקטורים x,y. כדי לסיים לוחצים על Enter.

5.7 השמת מספר גרפים יחד באמצעות פונקצית subplot

ניתן ליצור מספר גרפים באותו חלון גרפיקה באמצעות פונקצית subplot,

```
subplot(rows,cols,indx)
```

שני הארגומנטים הראשונים קובעים את מספר השורות והעמודות של התת-גרפים בתוך חלון הגרפיקה.
הארגומנט השלישי הוא האינדקס של התת-גרפים כאשר הסדר הוא משמאל לימין ואח"כ מלמעלה למטה.
ראו גרף 5.17 להמחשה. נקודה חשובה היא כי כל הפקודות בין subplot אחד ל-subplot הבא אחריו משפיעות רק על התת-הגרף הנידון.



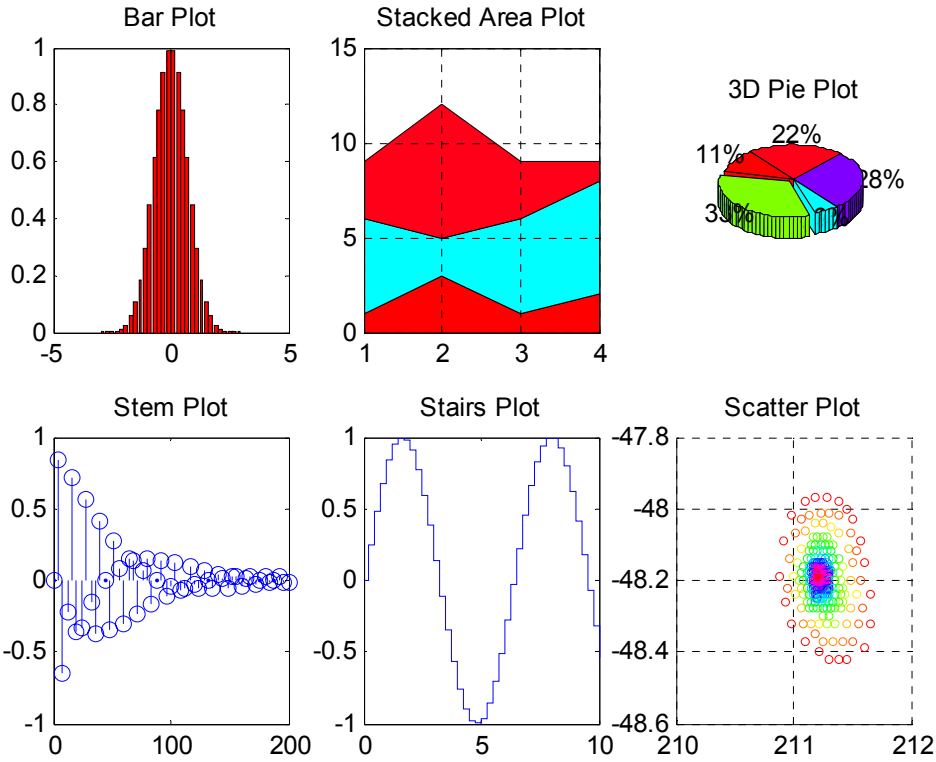
גרף 5.17: פונקצית subplot

לדוגמא הקוד,

```
figure

subplot(2,3,1)
x = -2.9:0.2:2.9;
bar(x,exp(-x.*x))
colormap hsv
title('Bar Plot')

subplot(2,3,2)
Y = [1 5 3;...
     3 2 7;...
     1 5 3;...
     2 6 1];
area(Y)
grid on
colormap summer
set(gca,'Layer','top')
title('Stacked Area Plot')

subplot(2,3,3)
x = [1 3 0.5 2.5 2];
explode = [0 1 0 0 0];
pie3(x,explode)
colormap hsv
title('3D Pie Plot')
```

```
subplot(2,3,4)
alpha = .02; beta = .5; t = 0:4:200;
y = exp(-alpha*t).*sin(beta*t);
stem(t,y)
title('Stem Plot')

subplot(2,3,5)
x = 0:.25:10;
stairs(x,sin(x))
title('Stairs Plot')

subplot(2,3,6)
load seamount
scatter(x,y,5,z)
title('Scatter Plot')
grid
```

מניב את גרף 5.18.



גרף 5.18: דוגמא לשימוש בפונקצית subplot

6. גרפיקה תלת ממדית

פרק זה סוקר יצירת קווים תלת ממדיים ומשטחים תלת ממדיים[24].

6.1 קווים תלת ממדיים

.
הגרסה התלת-ממדית של פונקצית plot היא,

$$plot3(x_1, y_1, z_1, x_2, y_2, z_2, \ldots)$$

כאשר $\{x_i, y_i, z_i\}, i = 1,2,\ldots$ יכולים להיות שלישיות של סקלרים, שלישיות של וקטורים בעלי אותו אורך או שלישיות של מטריצות מאותו גודל. הוספת מחרוזת קצרה לאחר כל שלישיה מאפשרת שליטה בצבע ובצורת הנקודות או הקווים. באופן כללי הרישום הוא,

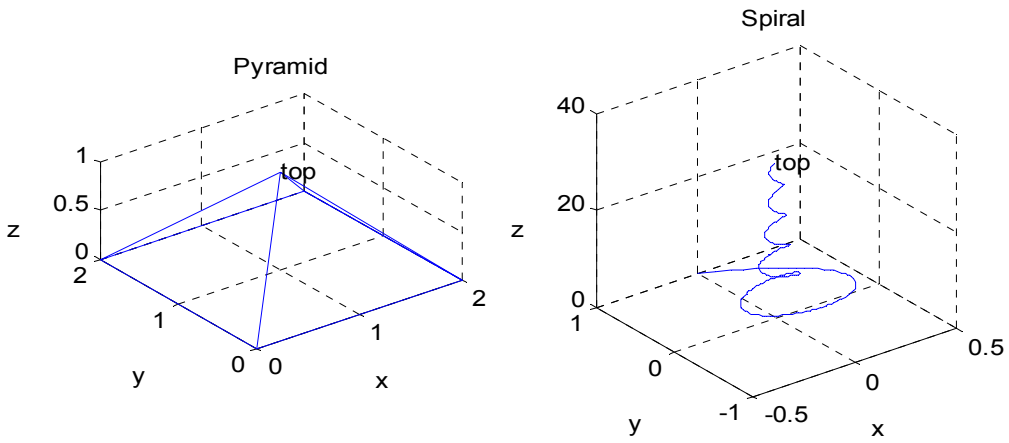
$$plot3(x_1, y_1, z_1, c_1, x_2, y_2, z_2, c_2, \ldots)$$

פונקציה זו מתפקדת בדומה למקבילה הדו ממדית שלה. ההבדלים היחידים הם שבשימוש בפונקצית text צריך להוסיף עוד קואורדינטה בשביל ציר z והתווית של ציר z נקבעת באמצעות פונקצית zlabel.
דוגמא,
נשרטט פירמידה וספירלה אחת ליד השניה. הקוד,

```
figure

subplot(1,2,1)
x = [0 2 2 1 0 0 1 2 2 0];
y = [0 0 2 1 2 0 1 0 2 2];
z = [0 0 0 1 0 0 1 0 0 0];
plot3(x,y,z)
title('Pyramid')
xlabel('x'),ylabel('y'),zlabel('z')
text(1,1,1,'top')
grid
axis equal

subplot(1,2,2)
t = 0:pi/50:10*pi;
x = sin(t)./(t+1);
y = cos(t)./(t+1);
z = t;
plot3(x,y,z)
title('Spiral')
xlabel('x'),ylabel('y'),zlabel('z')
text(x(end),y(end),z(end),'top')
grid on
axis square
```

מניב את גרף 6.1.

[24] ניתן לשרטט גרפים באופן ישיר מחלון Workspace ע"י סימון המטריצה, לחיצה על הכפתור הימני ובחירת סוג הגרף הרצוי. טריק זה חוסך כתיבת מספר שורות קוד אך מטבעו אפשריות התצוגה דרכו מוגבלות.



גרף 6.1: דוגמא לשימוש בפונקצית plot3

6.2 משטחים תלת ממדיים

משטח תלת ממדי מוגדר,

$$z = f(x, y)$$

הפונקציות הבסיסיות ליצירת משטחים תלת ממדיים הן פונקציות mesh ו- surf. רישום כללי של פונקציות אלו,

```
mesh(X,Y,Z)
```

ו-

```
surf(X,Y,Z)
```

כאשר X,Y הן מטריצות הקואורדינטות המופקות מוקטורי קואורדינטות x,y באמצעות פונקצית meshgrid. מטריצה Z היא מטריצה בעלת אותם ממדים כשל מטריצות X,Y.
ההבדל בין שתי הפונקציות הוא שפונקצית mesh יוצרת רשת קווים שצבעם נקבע ע"י מטריצת הגובה Z ולעומת זאת פונקצית surf יוצרת משטחים המורכבים מטלאים שצבעם נקבע ע"י מטריצת הגובה Z[25].
כביכול פונקצית mesh יוצרת טלאים לבנים שצבע מסגרתם נקבע ע"י מטריצת הגובה Z.

דוגמא,
רצוי לשרטט את הפונקציה,

$$f(x, y) = 5\sin\left(\frac{\pi}{15}xy\right)^2 + 10e^{-\left(x^2+y^2\right)} + 1$$

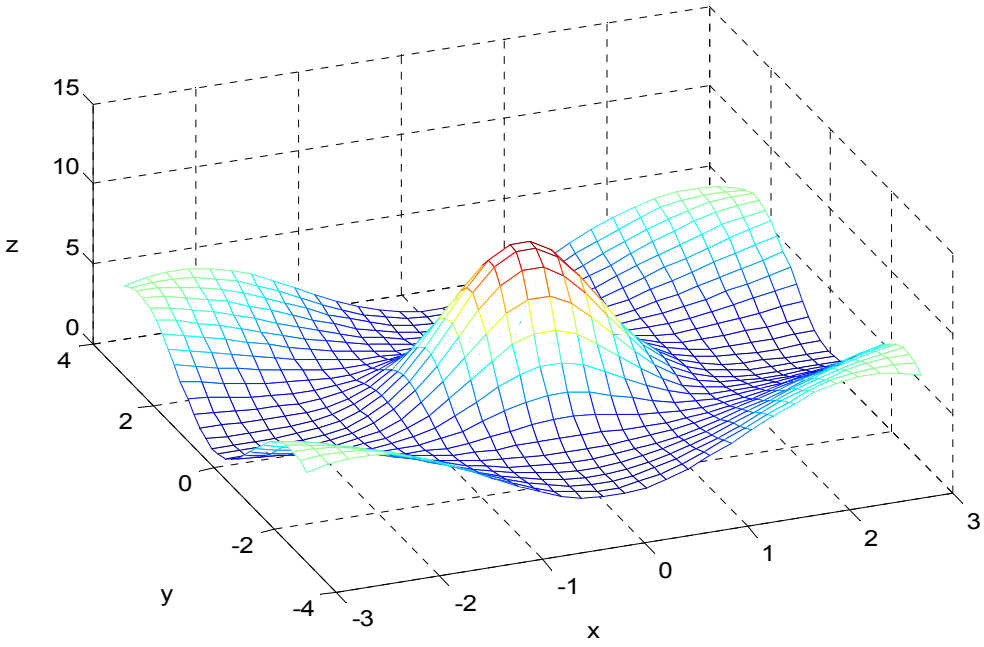
בתחום $\begin{cases} -3 \le x \le 3 \\ -3 \le y \le 3 \end{cases}$.

הקוד,

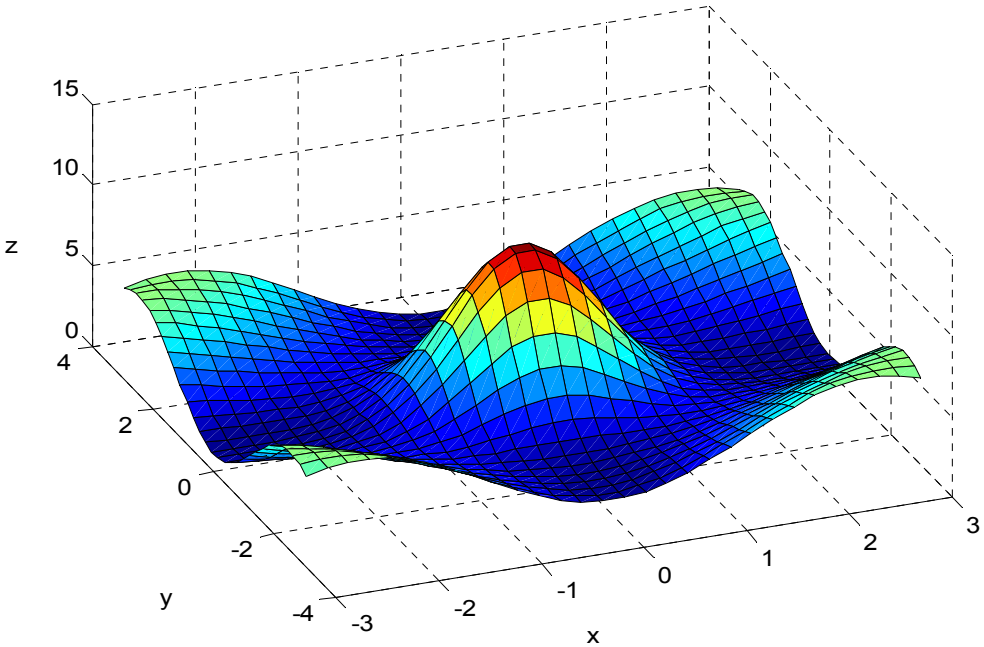
```
close all

xx = linspace(-3,3,30);
yy = linspace(-3,3,30);
[X,Y] = meshgrid(xx,yy);
Z = 5*sin(pi/15*X.*Y).^2 + 10*exp(-(X.^2 + Y.^2))+1;

figure
mesh(X,Y,Z)
xlabel('x'),ylabel('y'),zlabel('z')
view([-21.5 48])

figure
surf(X,Y,Z)
xlabel('x'),ylabel('y'),zlabel('z')
view([-21.5 48])
```

מניב את גרפים 6.2 ו-6.3 בהתאם.

[25] בפונקצית surf רשת הקווים בצבע אחיד (ברירת מחדל שחור).



גרף 6.2: שרטוט באמצעות פונקצית mesh



גרף 6.3: שרטוט באמצעות פונקצית surf

הערה

- אם ברצוננו לשרטט משטח חלק יותר באמצעות פונקציות יצירת גרפים תלת ממדיים אלו, כל שעלינו לעשות הוא להקטין את הרזולוציה באמצעות הוספת יותר נקודות דגימה.

<u>תזכורת לפונקציית meshgrid</u>

פונקציה זו מקבלת שני וקטורים (x,y) כקלט,

כאשר,
x - וקטור $[n\times 1]$
y - וקטור $[m\times 1]$

פונקציה זו משכפלת אותם ויוצרת שתי מטריצות בגודל זהה X,Y בגודל $[m\times n]$ באופן הבא,
הפקודות,

$$x=\begin{bmatrix}x_1 & x_2 & x_3 & x_4\end{bmatrix};$$
$$y=\begin{bmatrix}y_1 & y_2 & y_3\end{bmatrix};$$
$$[X,Y]=\text{meshgrid}(x,y);$$

יוצרות את המטריצות,

$$X\Leftarrow\begin{bmatrix}x_1 & x_2 & x_3 & x_4\\ x_1 & x_2 & x_3 & x_4\\ x_1 & x_2 & x_3 & x_4\end{bmatrix}$$
$$Y\Leftarrow\begin{bmatrix}y_1 & y_1 & y_1 & y_1\\ y_2 & y_2 & y_2 & y_2\\ y_3 & y_3 & y_3 & y_3\end{bmatrix}$$

ומתקיים,

$$\forall j,\ X(i,j)=x(i)$$
$$\forall i,\ Y(i,j)=y(j)$$

לכן שימוש באופרטור הנקודה עם המערכים X ו-Y מניב את אותן תוצאות כמו חישוב מטריצת Z איבר איבר באמצעות שתי לולאות, אבל בזמן קצר בהרבה.
להלן המחשת חישוב מטריצת Z איבר איבר באופן מאד לא יעיל,

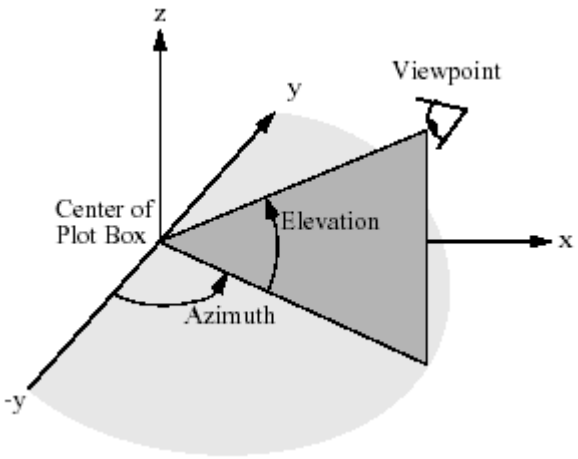
```
x = linspace(-3,3,30); m = length(x);
y = linspace(-3,3,30); n = length(y);

Z = zeros(m,n);
for i=1:m
    for j=1:n
        Z(i,j) = 5*sin(pi/15*x(i)*y(j))^2 + 10*exp(-(x(i)^2 + y(j)^2)) + 1;
    end
end
```

בקוד ליצירת גרף 6.2 ו-6.3 הוצגה בפעם הראשונה פונקצית view. ב-Matlab ניתן לשנות את האורינטציה של הגרף ע"י לחיצה על הכפתור  (ראו גרף 5.15) או באמצעות פונקצית rotate3d.
ברגע שמוצאים זוית צפיה רצויה ורוצים שכל פעם הגרף יוצג בזוית צפיה זו משתמשים בפונקצית view באופן הבא,
מגדירים את האורנטציה באמצעות אזימוט והגבהה. ראה גרף 6.4.
ביצוע הקוד,

```
[az el] = view;
```

מציב ערכים אלו במשתנים הנ"ל. קביעת האורנטציה של הגרף מבוצעת ע"י הצבת ערכים אלו בקלט של פונקצית view[26],

```
view([az el])
```

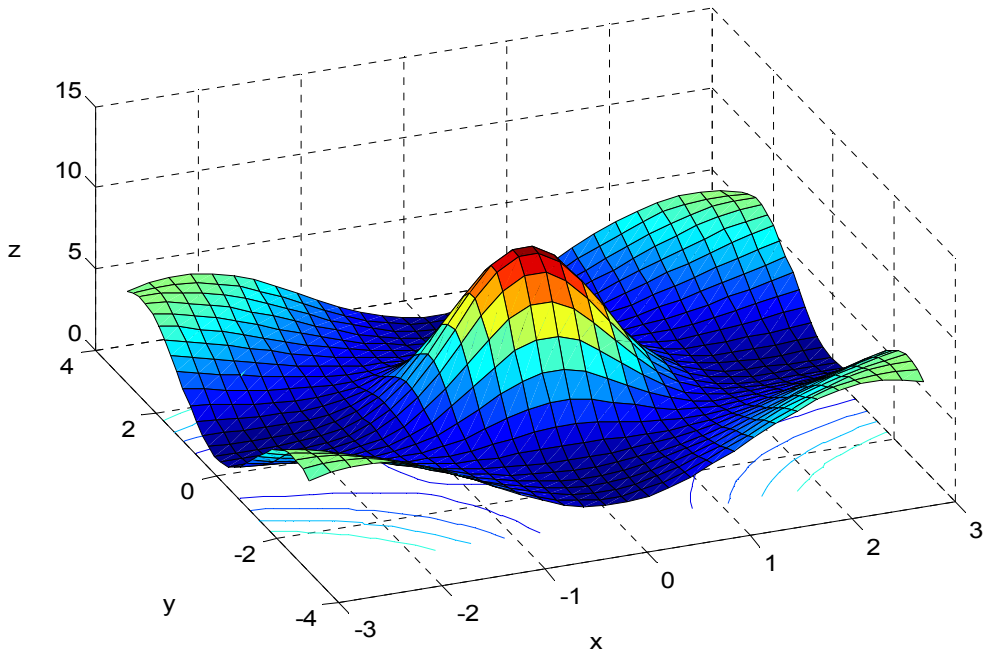


גרף 6.4: משמעות אזימוט והגבהה לפונקצית view

אם ברצוננו לקבל גם קווי מתאר מתחת לגרף עבור פונקציות surf ו-mesh נחליפן בפונקציות surfc ו-meshc בהתאם (האות c מיצגת את המילה contour שמשמעותה קווי מתאר).
לדוגמא הקוד,

```
figure
surfc(X,Y,Z)
xlabel('x'),ylabel('y'),zlabel('z')
view([-21.5 48])
```

מניב את גרף 6.5.



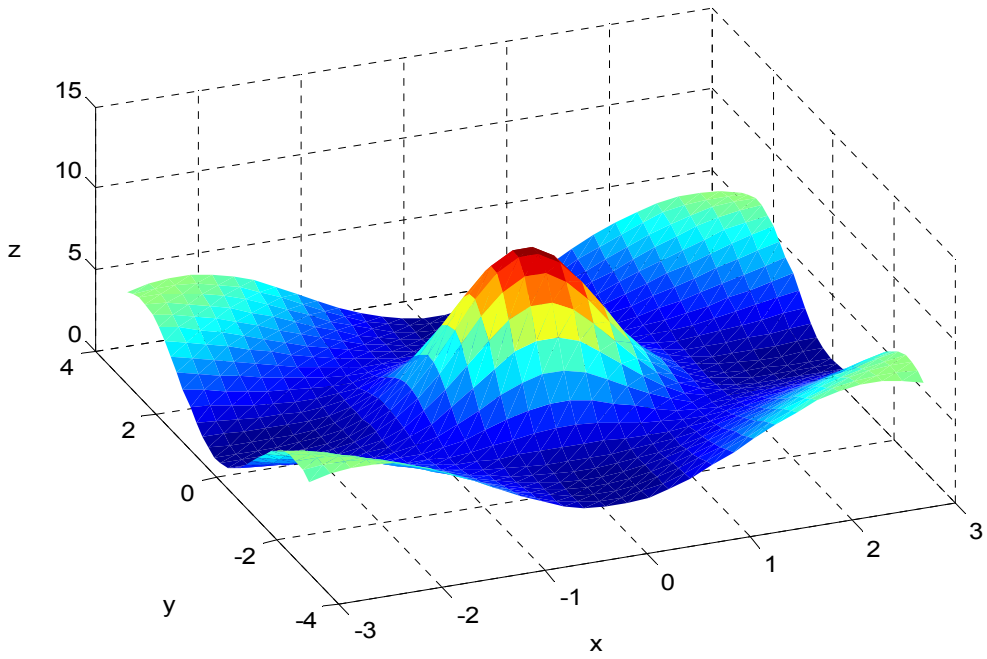
גרף 6.5: שרטוט באמצעות פונקצית surfc

[26] ערכי ברירת המחדל של גרף דו ממדי הם az=0 el=90 וערכי ברירת המחדל של גרף תלת ממדי הם az=-37.5 el=30.

ניתן להוריד את רשת הקווים באופן הבא,
לדוגמא הקוד,

```
figure
surf(X,Y,Z,'EdgeColor','none')
xlabel('x'),ylabel('y'),zlabel('z')
view([-21.5 48])
```

מניב את גרף 6.6.



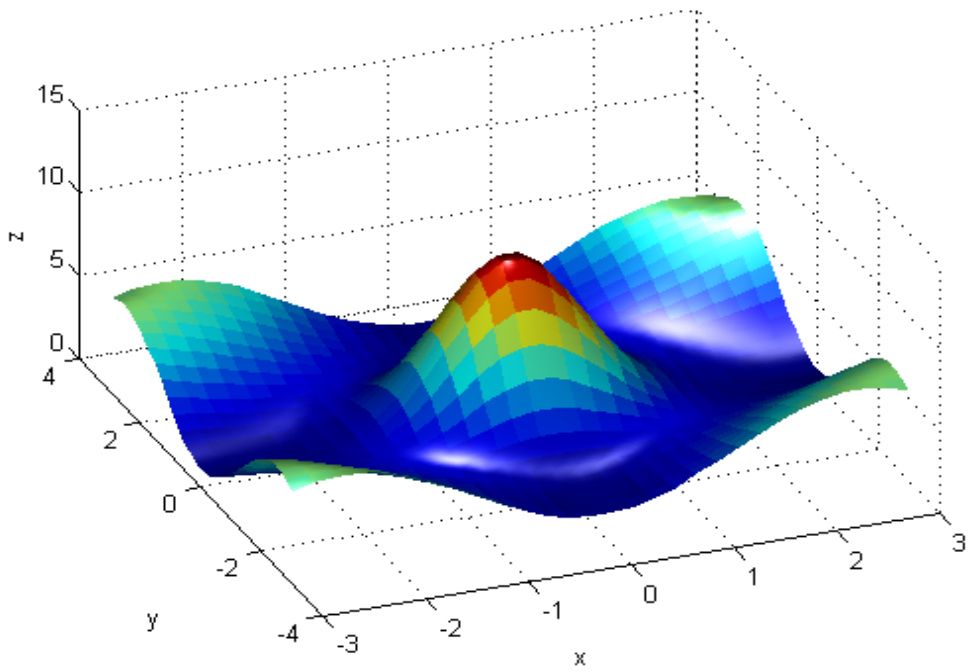
גרף 6.6: שרטוט ללא רשת הקוים

למען יצירת תחושה של אוביקט פיסי ניתן גם להוסיף תאורה[27] בכיוון רצוי באמצעות פונקצית camlight ולבחור את סוג התאורה עם פונקצית lighting. לדוגמא, הוספת השורות הנ"ל,

```
camlight headlight
lighting phong
```

משנות את גרף 6.6 לגרף 6.7.

[27] ניתן ליצור גרפים בעלי תאורה והצללה גם באמצעות פונקצית surfl.



גרף 6.7: הוספת תאורה לגרף

בתור ברירת מחדל הצבעים של הפונקציה משתנים בהתאם לגובה היחסי של הפונקציה. ניתן גם לקבוע כי הצבע יהיה אחיד ללא תלות בגובה באמצעות הפונקציה,

```
colormap(c)
```

כאשר c הוא וקטור בעל 3 איברים בעלי ערכים בין 0 ל-1. איברים אלו קובעים את יחס הצבעים הבסיסיים אדום, ירוק, כחול שמרכיבים את הצבע הרצוי לפי הסדר הנ"ל. להלן טבלה המציגה חלק מהאפשריות,

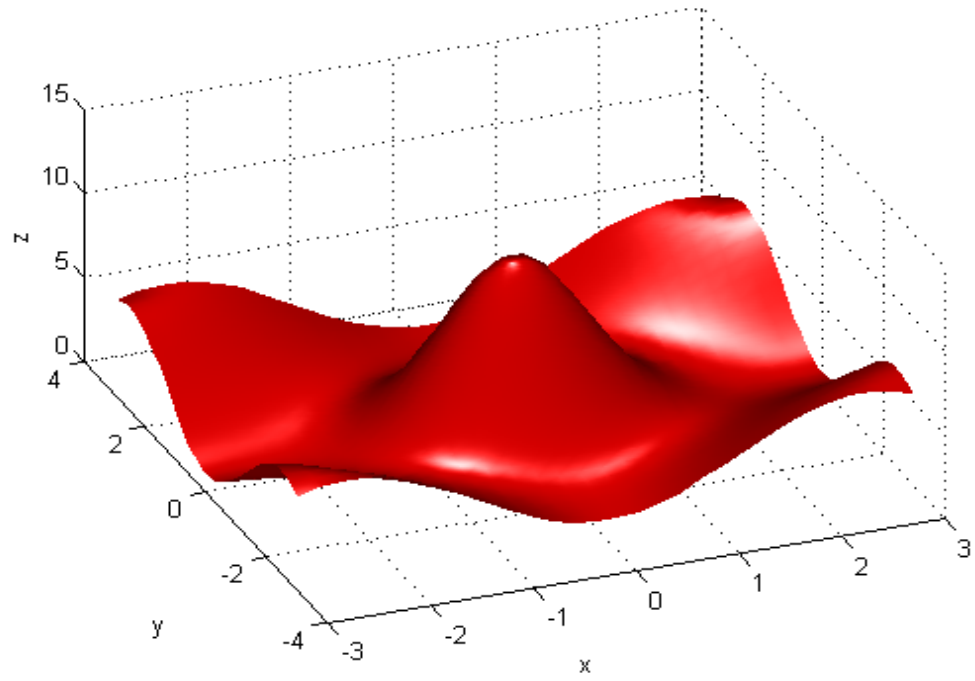
Red	Green	Blue	Color
0	0	0	black
1	1	1	white
1	0	0	red
0	1	0	green
0	0	1	blue
1	1	0	yellow
1	0	1	magenta
0	1	1	cyan
0.5	0.5	0.5	gray
0.5	0	0	dark red
1	0.62	0.40	copper
0.49	1	0.83	aquamarine

טבלה 6.1: מספר צבעים ידועים וערכיהם

לדוגמא הוספת הקוד,

```
colormap([1 0 0])
```

מניבה את גרף 6.8.



גרף 6.8: קביעת צבע אחיד למשטח

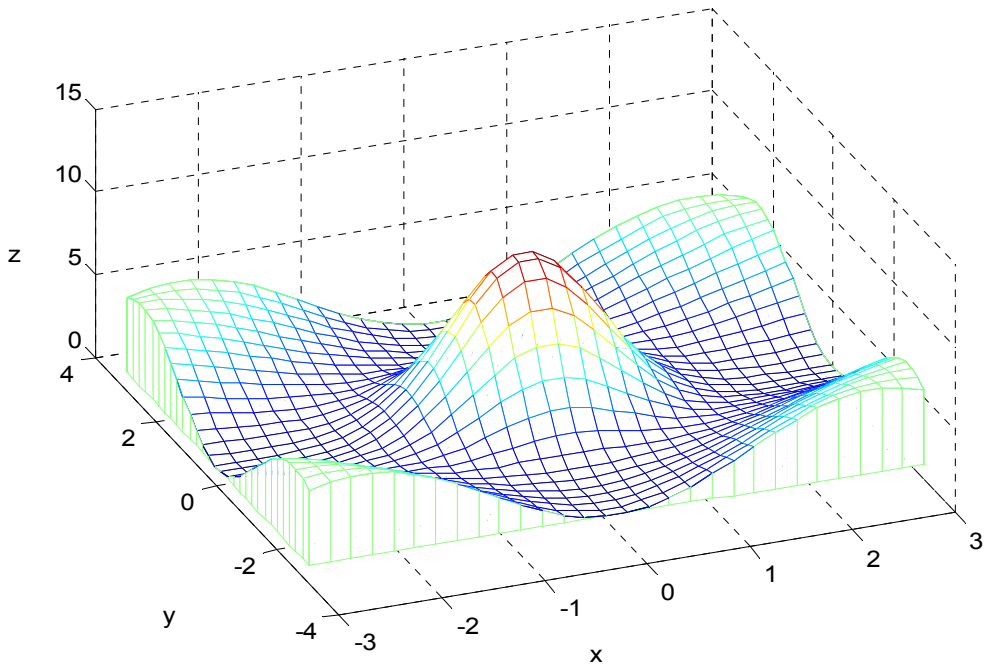
אפשריות נוספות הן,

- כיבוי רשת הצירים ע"י כתיבת grid off והדלקתה חזרה ע"י כתיבת grid on.
- הצבת קופסא סביב הגרף ע"י כתיבת box on והורדתה ע"י כתיבת box off.
- כיבוי מערכת הצירים ע"י כתיבת axis off והדלקתה חזרה ע"י כתיבת axis on.

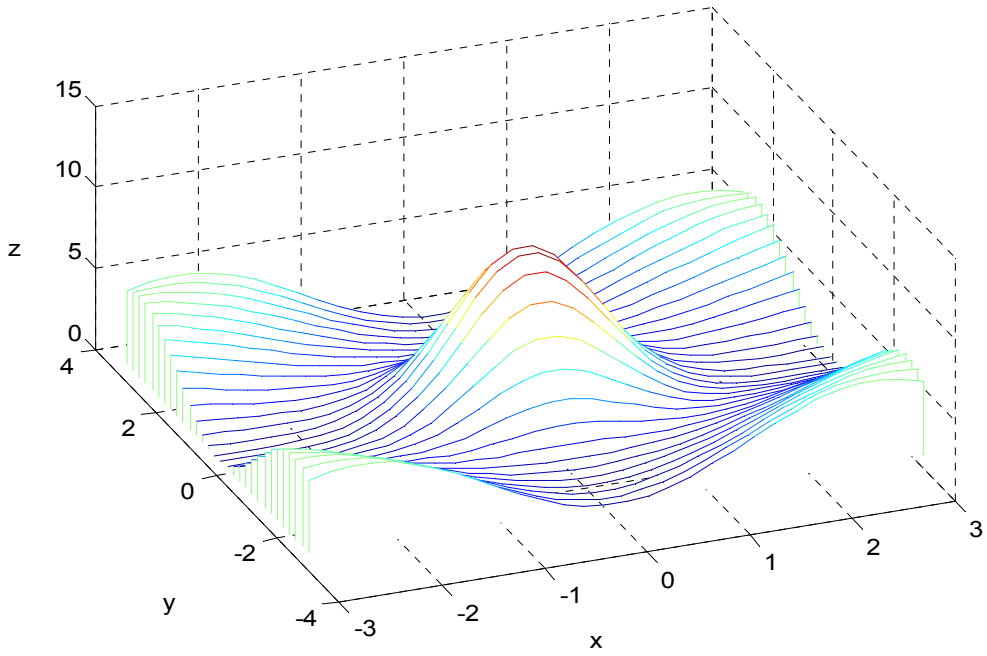
קיימות עוד שתי פונקציות המהוות וריאציות לפונקציות mesh ו-surf, בשם meshz ו-waterfall, המיצרות גרפים מעט שונים.
הקוד,

```
figure
meshz(X,Y,Z)
xlabel('x'),ylabel('y'),zlabel('z')
view([-21.5 48])

figure
waterfall(X,Y,Z)
xlabel('x'),ylabel('y'),zlabel('z')
view([-21.5 48])
```

מניב את גרפים 6.9 ו-6.10 בהתאם.



גרף 6.9: פונקציית meshz



גרף 6.10: פונקציית waterfall

6.3 קווי מתאר

קיימות 3 פונקציות ליצירת גרפים של קווי מתאר,

1. יצירת קווי מתאר דו ממדיים

```
contour(X,Y,Z)
```

לדוגמא הקוד,

```
figure
h = contour(X,Y,Z);
clabel(h)
xlabel('x'),ylabel('y'),zlabel('z')
```

מניב את גרף 6.11.

השימוש בפונקצית clabel באופן הנ"ל מציג את הערכים של קווי המתאר. אם אין צורך בערכים אלו אז משתמשים בפונקציה באופן הרגיל.

2. יצירת קווי מתאר תלת ממדיים

```
contour3(X,Y,Z)
```

לדוגמא הקוד,

```
figure
h = contour3(X,Y,Z);
clabel(h)
xlabel('x'),ylabel('y'),zlabel('z')
view([-19.5 42])
```

מניב את גרף 6.12.

פונקציה זו זהה בתפעולה לפונקציה contour.

3. יצירת קווי מתאר דו ממדיים מלאים

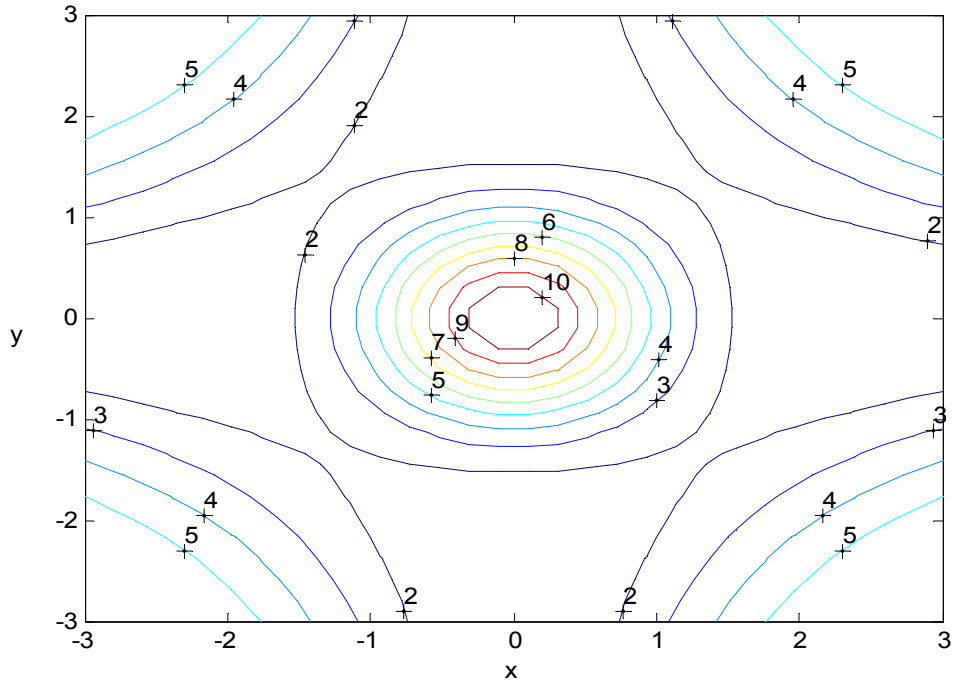
```
contourf(X,Y,Z)
```

פונקציה זו דומה לפונקציה contour רק שכעת המרווחים בין קווי המתאר מולאו בצבע.
בתצוגה כזו שימוש בפונקציה clabel אפשרי, אך לעיתים המספרים אינם ברורים ולכן כחלופה ניתן להשתמש באופצית colorbar. פונקציה זו מציבה עמודת צבע לצד הגרף המעידה על הערך הנומרי של כל צבע. ניתן להשתמש בפונקציה זו בכל גרף תלת ממדי אך היא אפקטיבית בגרפים מלאים (גרפים של surf, surfc, contourf).

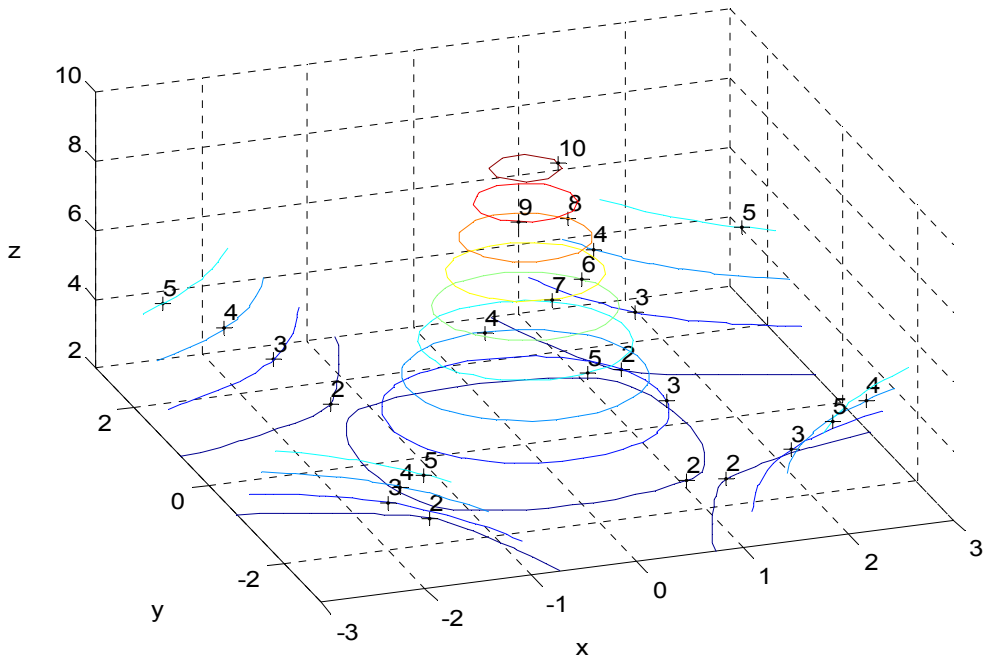
לדוגמא הקוד,

```
figure
contourf(X,Y,Z);
xlabel('x'),ylabel('y'),zlabel('z')
colorbar
```

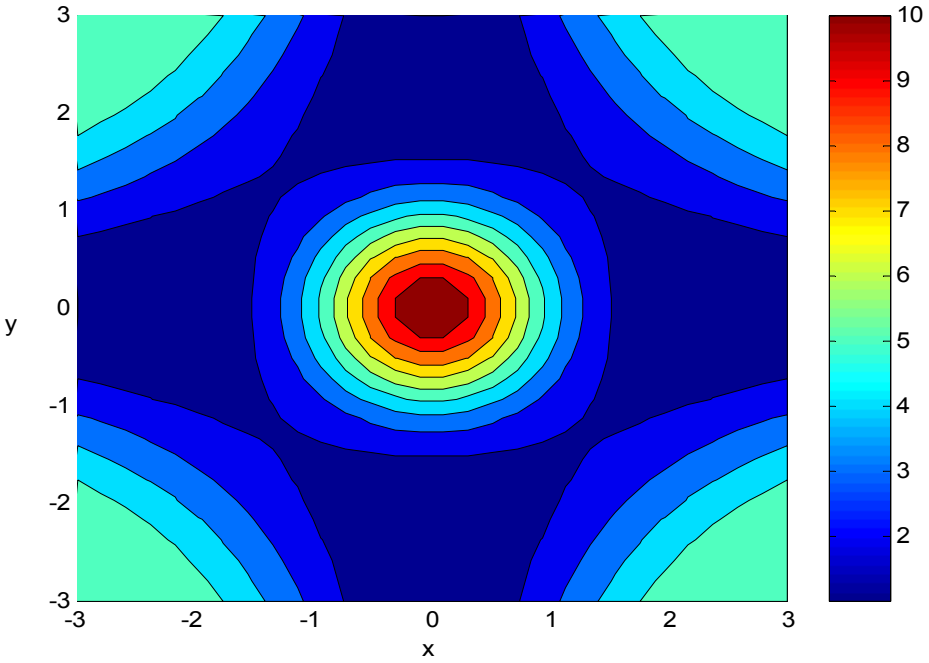
מניב את גרף 6.13.



גרף 6.11: שרטוט קווי מתאר באמצעות פונקציות contour ו-clabel



גרף 6.12: שרטוט קווי מתאר באמצעות פונקציות contour3 ו-clabel



גרף 6.13: שרטוט קווי מתאר באמצעות פונקציות contourf ו-colorbar

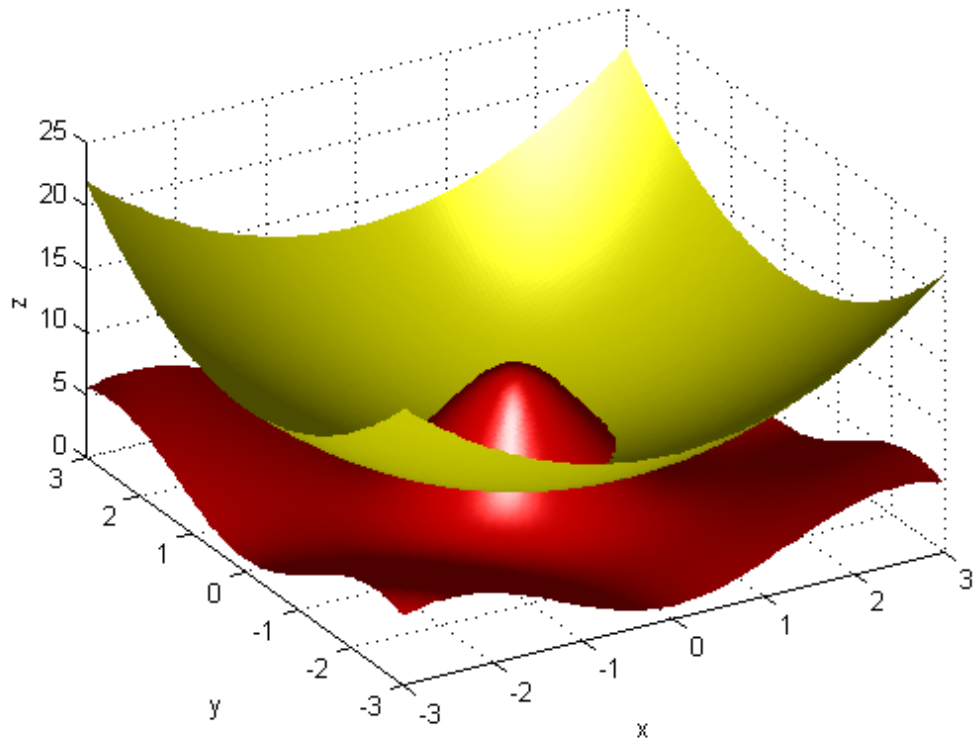
6.4 הצגת מספר משטחים על גרף יחיד

הצגת מספר משטחים על גרף יחיד מבוצעת באמצעות פונקצית hold.

לדוגמא,
נוסיף למשטח הדוגמא מפרק 6.2 ו-6.3 עוד פרבולה $g(x,y) = x^2 + y^2 + 4$.
הקוד,

```
xx = linspace(-3,3,100);
yy = linspace(-3,3,100);
[X,Y] = meshgrid(xx,yy);
Z = 5*sin(pi/15*X.*Y).^2 + 10*exp(-(X.^2 + Y.^2))+1;
figure
surf(X,Y,Z,'EdgeColor','none','FaceColor','red')
xlabel('x'),ylabel('y'),zlabel('z');
camlight headlight
lighting phong

hold on
Z2 = X.^2 + Y.^2 + 4;
surf(X,Y,Z2,'EdgeColor','none','FaceColor','yellow')
view(-30.5,40);
hold off
```

מניב את גרף 6.14.

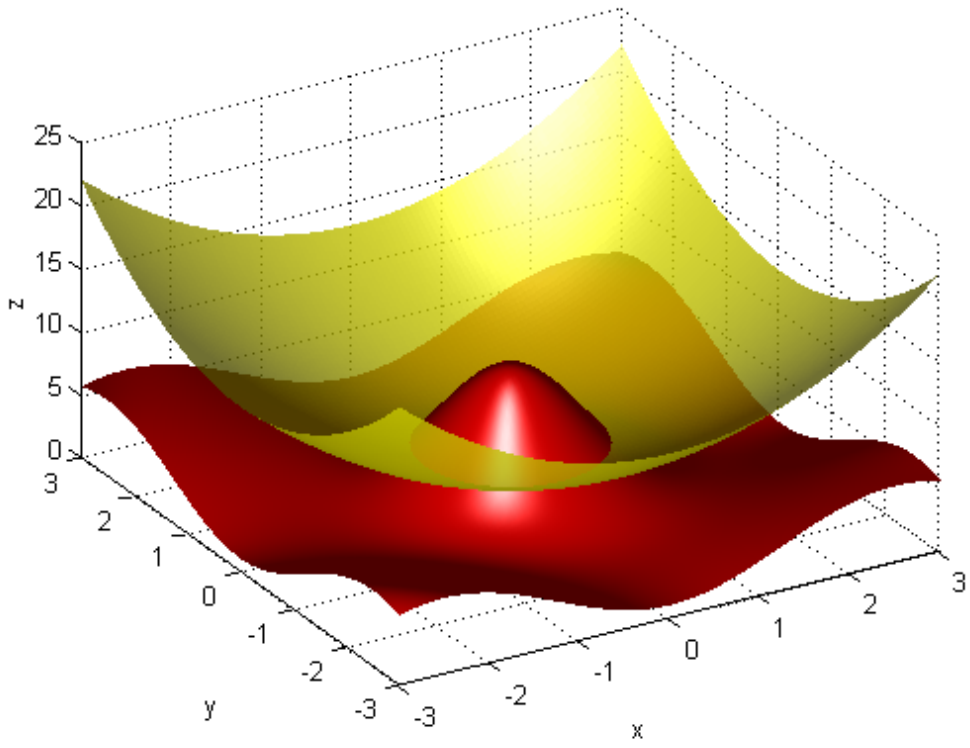


גרף 6.14: שרטוט מספר משטחים על גרף יחיד

בגרף 6.14 שני המשטחים חותכים אחד את השני וכתוצאה מכך הפונקציה התחתונה מוסתרת בחלקה.
לכן אפשרות שימושית היא לגרום למשטחים להיות שקופים במידה מסויימת. ניתן לקבוע את דרגת השקיפות של המשטחים בגרף באמצעות פרמטר alpha. פרמטר זה יכול לקבל סקלר בעל ערכים בין 0 ל-1 כאשר 1 מייצג אטימות מוחלטת, ו-0 שקיפות מוחלטת.
לדוגמא,
נשנה את שקיפות המשטח העליון בלבד.
הקוד,

```
surf(X,Y,Z,'EdgeColor','none','FaceColor','red')
xlabel('x'),ylabel('y'),zlabel('z')
camlight headlight
lighting phong

hold on
Z2 = X.^2 + Y.^2 + 4;
surf(X,Y,Z2,'EdgeColor','none','FaceColor','yellow','FaceAlpha',0.8)
view(-30.5,40)
hold off
```

מניב את גרף 6.15.



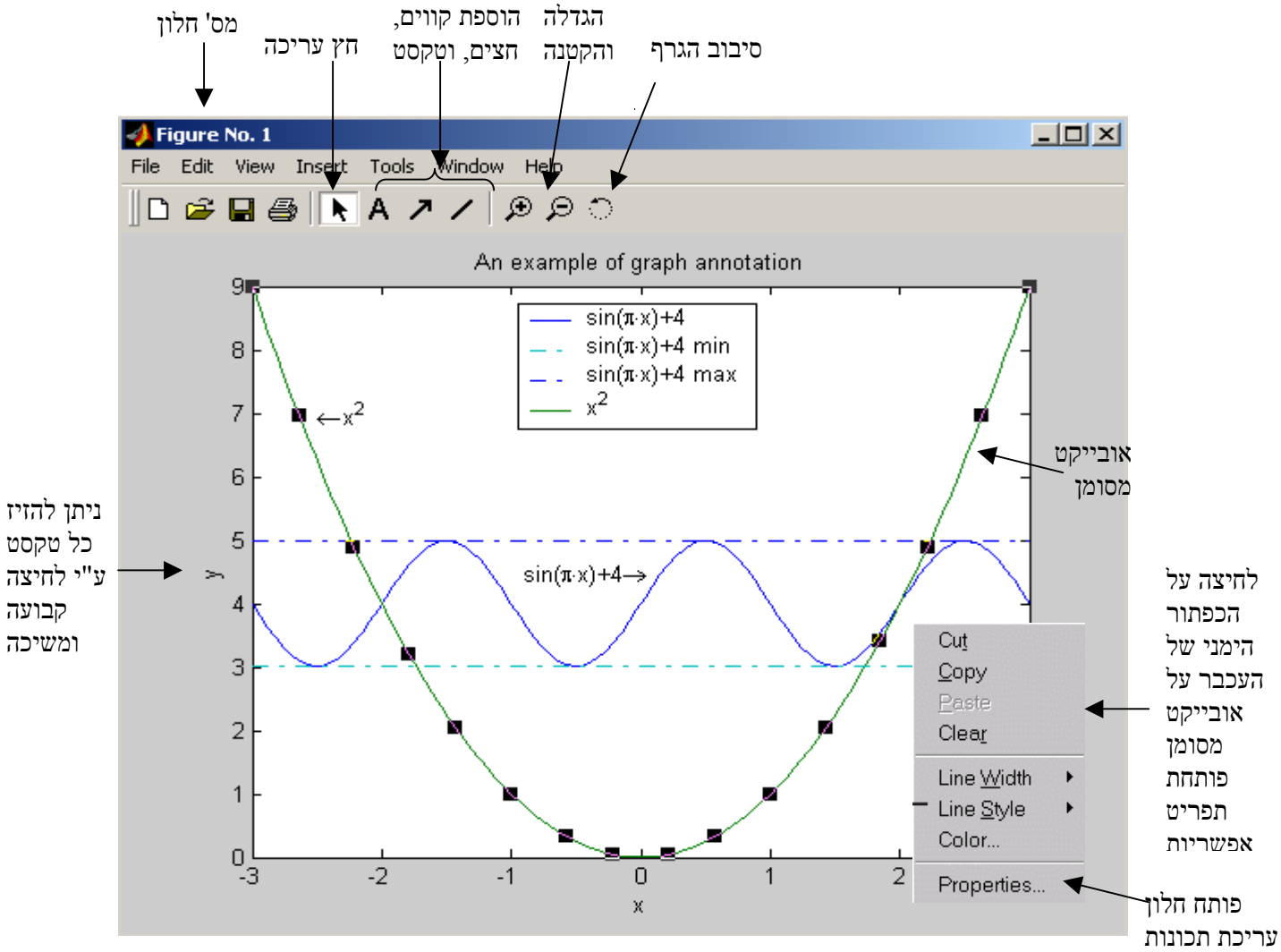
גרף 6.15: שרטוט מספר משטחים על גרף יחיד ויצירת שקיפות עם פרמטר alpha

7. שליטה בגרפים

גרפים ב-Matlab מכילים אובייקטים של צירים, קווים, טקסט ועוד. התכונות של אובייקטים גרפיים ניתנות לשינוי באופן ידני או באמצעות Handle Graphics.

7.1 שליטה ידנית בגרפים

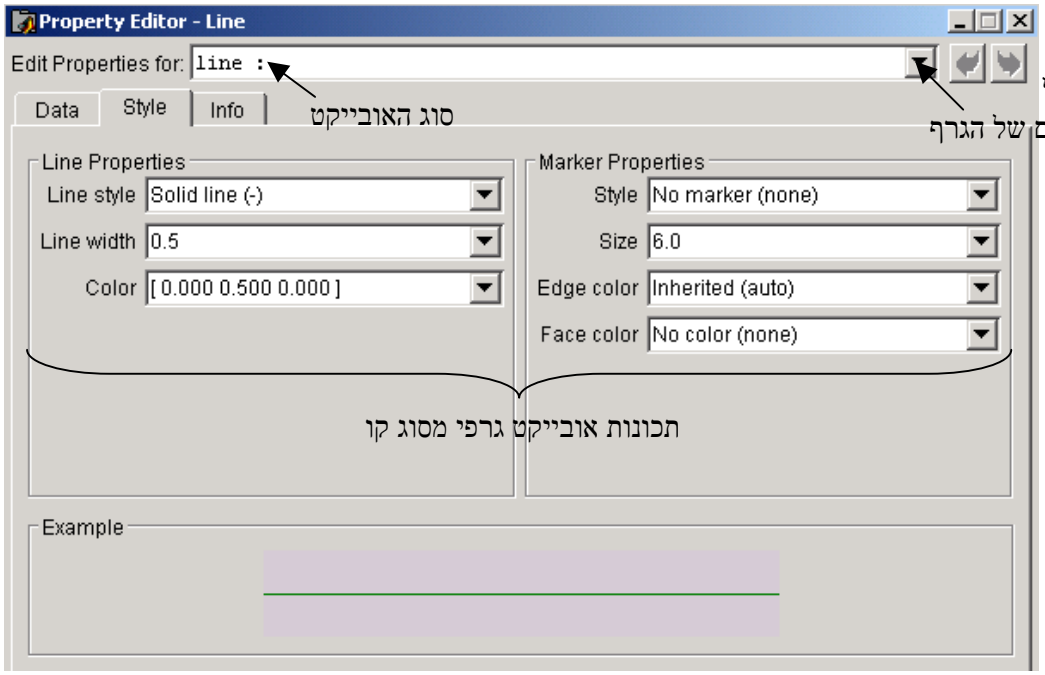
הפקודות בפרק 5.3 ניתנות לקביעה דרך תפריט Insert או דרך סרגל העבודה (toolbar) כפי שמופיע בגרף 7.1.



גרף 7.1: חלון גרפי טיפוסי

התכונות של אובייקט הגרף ואובייקט הצירים ניתנות לשינוי דרך תפריט Edit.
סימון אובייקט לעריכה מבוצע ע"י לחיצה על חץ העריכה ולחיצה על האובייקט הרצוי[28]. לחיצה על הכפתור הימני של העכבר על אובייקט מסומן פותחת אפשריות עריכה כפי שמודגם בגרף 7.1.
כדי לגשת לכל תכונות האובייקט, לחיצה על Properties פותחת את חלון עריכת התכונות. בחלון זה ניתן לשנות פרמטרים רבים כמו צבע, עובי קו, סגנון ועוד. בגרף 7.2 מוצג חלון עריכת תכונות של אובייקט קו.

[28] יוצאים ממצב עריכה ע"י לחיצה על כפתור חץ העריכה.

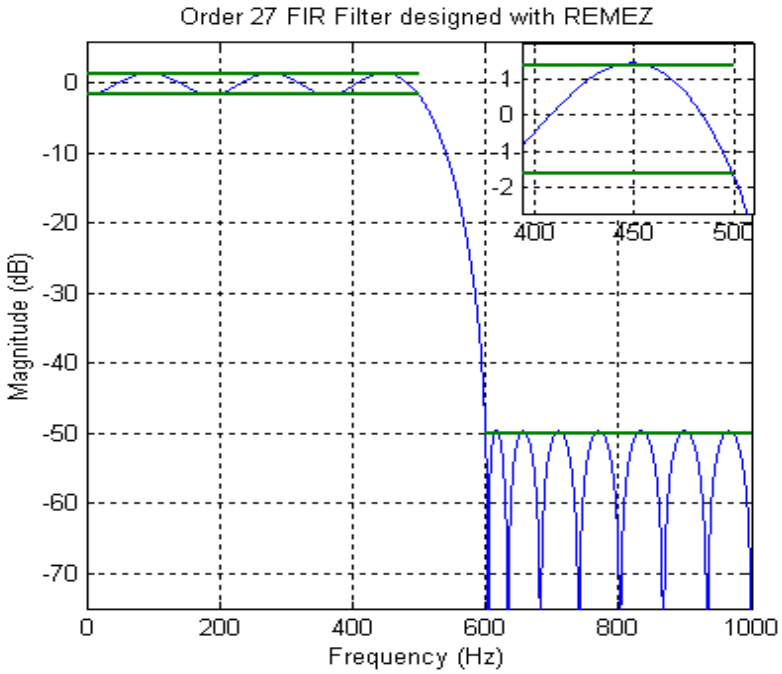


גרף 7.2: חלון עריכת תכונות

ניתן לשנות ידנית את מבנה הגרף ע"י סימונו באמצעות חץ העריכה, לחיצה על הכפתור הימני של העכבר על גבול הגרף ובחירת Unlock Axes Position.כעת ניתן להזיז, למתוח, לכווץ את הגרף. שימוש אפשרי של תכונה זו הוא השמת גרף על גרף.

דוגמא,

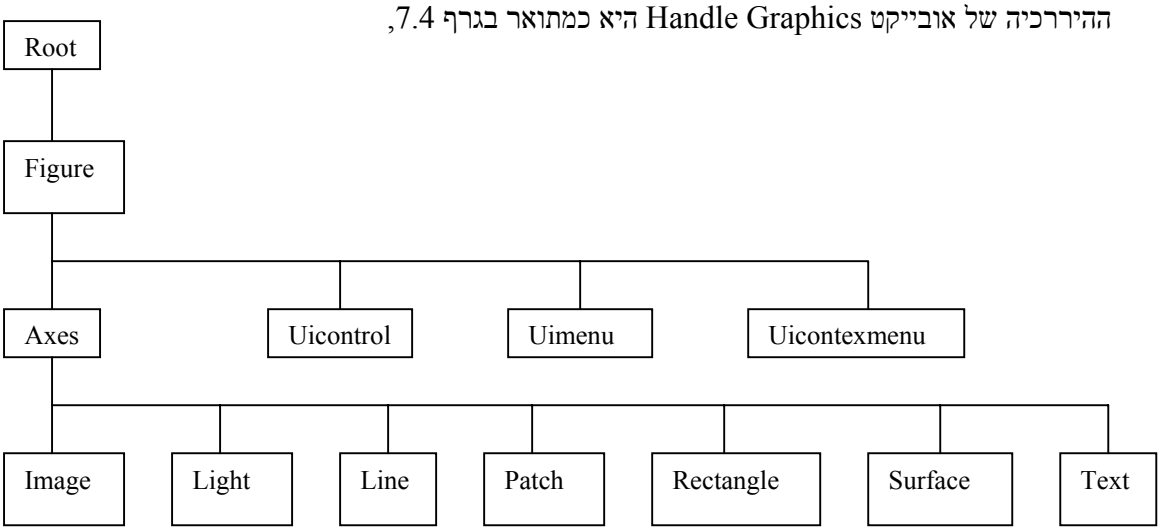
בבניית Low Pass Filter רוצים לבדוק כי הוא עומד בדרישות. כלומר שתגובת התדר שלו נמצאת בתחום המסומן בקווים הירוקים בגרף 7.3. בקצה אזור המעבר הדרישה היא הכי תובענית ולכן רצוי לשים על אותו גרף גם תמונת תקריב של האזור. הדבר נעשה במקרה זה ע"י שחרור הצירים של הגרף כפי שהוסבר, שכפול הגרף באמצעות copy ו-paste, ביצוע zoom in באחד הגרפים והזזתו למיקום הרצוי בגרף השני.



גרף 7.3: הדגמת הדבקת חלון על חלון בצורה ידנית

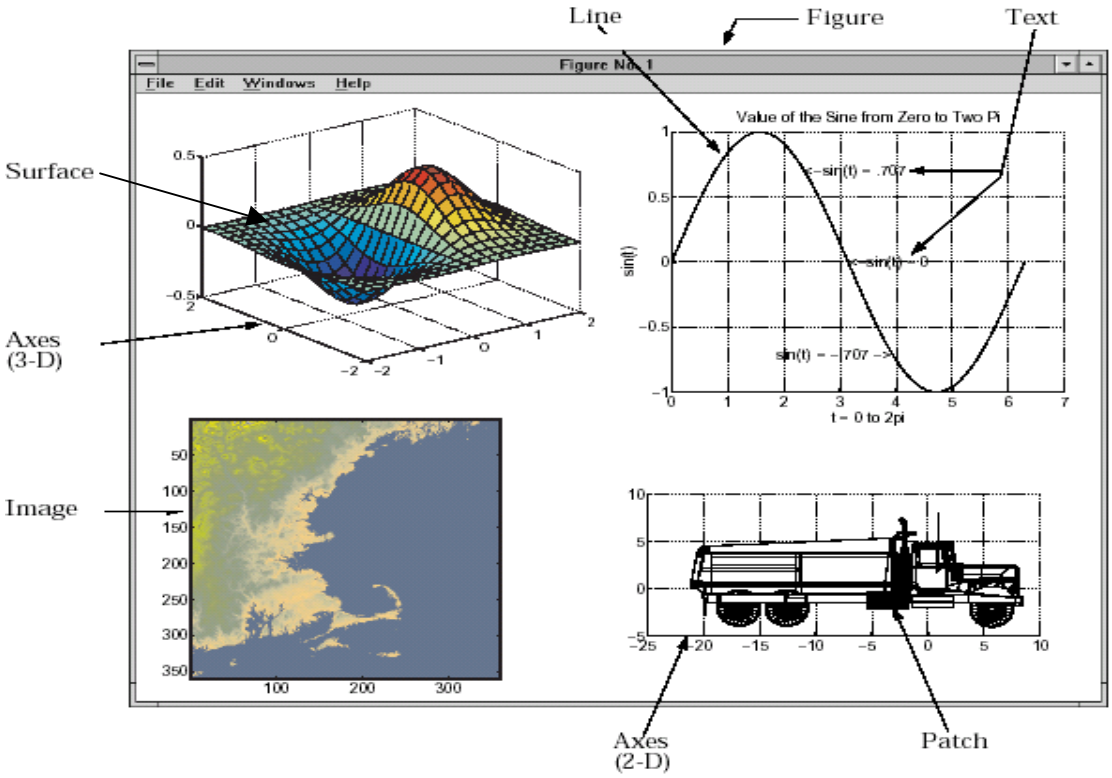
7.2 אובייקטים של Handle Graphics

אובייקטי Handle Graphics הם רכיבי השרטוט הבסיסיים בעזרתם Matlab מציג מידע ויוצר גרפים ו-GUI-ים. לכל אובייקט גרפיקה מקושר handle שדרכו ניתן לשנות את התכונות של האובייקט.
ההיררכיה של אובייקט Handle Graphics היא כמתואר בגרף 7.4,



גרף 7.4: היררכיה של אובייקט Handle Graphics

הטבע ההיררכי של Handle Graphics ב-Matlab נובע מהתלות של אובייקטי הגרף. למשל כדי לשרטט קו (Line) ב-Matlab צריך מערכת צירים (Axes). כדי לצייר צירים חייבים חלון גרפיקה (Figure).
בגרף 7.5 מוצגת דוגמא הממחישה כיצד משולבים מספר אובייקטי גרפיקה מסוגים שונים.

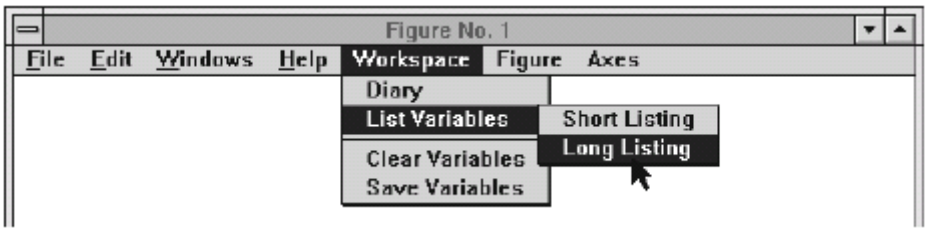


גרף 7.5: שילוב מספר אובייקטי גרפיקה

בטבלה 7.1 מפורטים סוגי אובייקט גרפיקה.

סוג אובייקט גרפיקה	פרוט תפקוד אובייקט גרפיקה
Root	שולט בתצוגת המסך. קיים רק אובייקט אחד כזה וכל שאר האובייקטים הם צאצאים שלו. Matlab יוצר את האובייקט הזה באופן אוטומטי וניתן לשנות את תכונותיו.
Figure	שולטים ביצירת חלונות גרפיקה. Matlab איננו מגביל את מספר חלונות הגרפיקה. כל ה-Figure-ים הם ילדים (children) של ה-root וכל שאר האובייקטים הם הצאצאים של ה-Figure.
Uicontrol	שולטים בבקרים המקבלים קלט בממשק גרפי. למשל תפריטים נפתחים (popup menu). למשל, נשים לב כי לפי ההיררכיה אין תלות בין אובייקט axes לאובייקט Uicontrol.
Uimenu	שולטים בתפריטים הממוקמים בראש חלון גרפיקה כפי שמוצג בגרף 7.6.
Uicontextmenu	שולט בתפריטים הצצים כאשר מקישים על אובייקט גרפיקה באמצעות כפתור העכבר הימני.
Axis	מגדיר תחום בחלון הגרפיקה שבו יוצג הגרף ומסדר את הילדים שלו (image, light, line, patch, rectangle, surface, text) בתוך התחום הנ"ל.
Image	אובייקט תמונה ב-Matlab מורכב ממטריצת תמונה ואולי גם colormap. קיימים שלושה סוגי דרכים בהן מטריצת התמונה מפורשת כצבע הפיקסלים- indexed, intensity, truecolor.
Light	מגדיר מקורות תאורה המשפיעים על אובייקטי patch ו-surface.
Line	האלמנטים הבסיסיים אתם יוצרים קווים לגרפיקה דו-ממדית ותלת-ממדית.
Patch	פוליגונים (רב מצולעים) ממולאים בעלי קצוות. פונקציות כמו fill משתמשות באובייקטים אלו.
Rectangle	משטחים דו-ממדיים בעלי צורות היכולות לנוע בין מלבן לאליפסה.
Surface	ייצוג תלת-ממדי של מטריצה ע"י שרטוט כל אלמנט של מטריצה כגובה מעל מישור x-y.
Text	מחרוזות תווים. פונקציות עליות כמו title, xlabel, ylabel משתמשות באובייקטים אלו.

טבלה 7.1: סוגי אובייקט גרפיקה



גרף 7.6: תפריטים הנשלטים ע"י אובייקט Uimenu

7.2.1 יצירת גרפים באמצעות הפונקציות הבסיסיות ליצירת גרפים

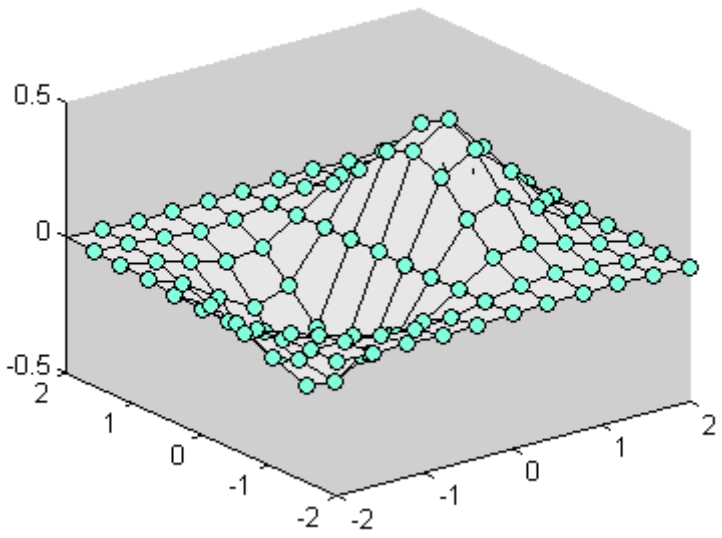
לכל סוג אובייקט גרפיקה קיימת פונקציה ליצירת האובייקט בעלת אותו שם. למשל כדי לייצר אובייקט Surface משתמשים בפונקצית surface. פונקציות "עיליות" יותר כמו contour, surf וכו' משתמשות בפונקציות הבסיסיות האלו כדי לייצר גרפים. לכן ניתן להשתמש בפונקציות הבסיסיות האלו כדי לייצר פונקציות גרפיקה חדשות.
יצירת אובייקט גרף באמצעות הפונקציות הבסיסיות מבוצע באופן כללי,

handle = function('propertyname',propertyvalue,...)

לדוגמא הקוד,

```
[x,y] = meshgrid([-2:.4:2]);
Z = x.*exp(-x.^2-y.^2);
fh = figure('Position',[350 275 400 300],'Color','w');
ah = axes('Color',[.8 .8 .8],'XTick',[-2 -1 0 1 2],...
    'YTick',[-2 -1 0 1 2]);
sh = surface('XData',x,'YData',y,'ZData',Z,...
    'FaceColor',get(ah,'Color')+.1,...
    'EdgeColor','k','Marker','o',...
    'MarkerFaceColor',[.5 1 .85]);
view(3)
```

יוצר את גרף 7.7



גרף 7.7

פרשנות הקוד,
לכל אובייקט handle משלו. תכונת position של אובייקט figure קובעת את מיקום וממדי חלון הגרפיקה ע"ס וקטור בעל 4 איברים,

[left, bottom, width, height]

כאשר left ו-bottom מייצגות את המיקום של הפינה השמאלית התחתונה של חלון הגרפיקה מהפינה השמאלית תחתונה של המסך. width ו-height קובעות את ממדי חלון הגרפיקה. היחידות נקבעות ע"י תכונת Units.
תכונת Color ב-Figure מציינת את צבע הרקע בחלון הגרפיקה. לעומת זאת, תכונת Color ב-Axes מציינת את צבע הרקע של מערכת הצירים.
תכונות Xtick ו-Ytick קובעות את הטיקים במערכת הצירים.
תכונות XData, Ydata, Zdatat קובעות המשטח במישור x-y-z.

7.2.2 קבלת וקביעת תכונות של אובייקט גרפיקה

פונקציות set ו-get קובעות ומחזירות תכונות של אובייקט גרפיקה. בנוסף הן גם מאפשרות פירוט כל הערכים האפשריים של תכונות האובייקט[29].
הרישום הבסיסי לקביעת ערך של תכונת אובייקט גרפיקה,

```
set(object_handle,'PropertyName','NewPropertyValue')
```

הרישום הבסיסי לקבלת ערך של תכונת אובייקט גרפיקה,

```
returned_value = get(object_handle,'PropertyName');
```

נשים לב כי שמות של תכונות PropertyName הם תמיד מחרוזות. לעומת זאת הערך שיוצב בתכונה PropertyName לא חייב להיות בהכרח מחרוזת כפי שמצוין ברישום הבסיסי.

לדוגמא הקוד,

```
cur_marker = get(sh,'Marker');
if strcmp(cur_marker,'o')
    set(sh,'Marker','p')
end
```

משנה את ה-marker של משטח surface לפנטגון אם ה-marker הנוכחי הוא עיגול.

הרישום הבסיסי להצגת רשימת הערכים האפשריים של תכונה מסוימת ללא שינויה הוא,

```
set(obj_handle,'PropertyName')
```

למשל כדי להציג את הערכים האפשריים של תכונת marker נרשום,

```
set(sh,'Marker')
```

ונקבל,

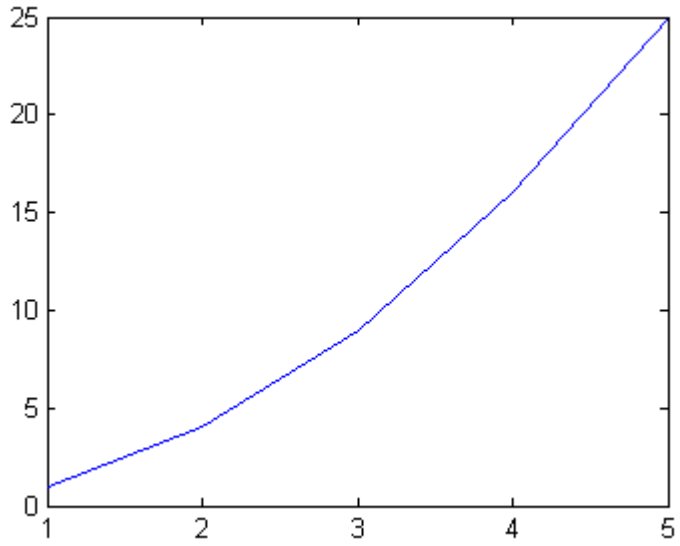
```
[ + | o | * | . | x | square | diamond | v | ^ | > | < | pentagram
| hexagram | {none} ]
```

הסוגריים המסולסלות מציינות את ערך ברירת מחדל.
ניתן לבצע שינויים בגרפים שנוצרו באמצעות פונקציות set, get. למשל הקוד,

[29] רשימה של כל האובייקטים של גרפיקה ותכונותיהם קיימת בקובץ העזרה Handle Graphics Online Documentation (בפורמט html).

```
x = 1:5;
y = x.^2;
h = plot(x,y);
```

מייצר את גרף 7.8.
h הוא handle לאובייקט line שפונקצית plot מייצרת.

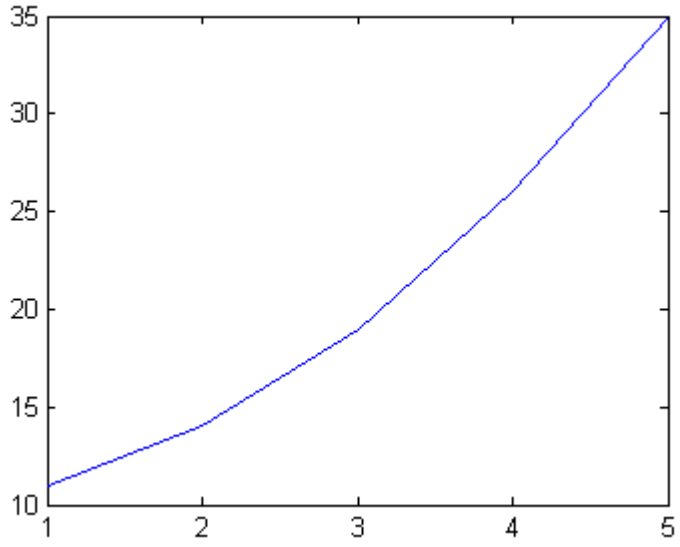


גרף 7.8

הקוד,

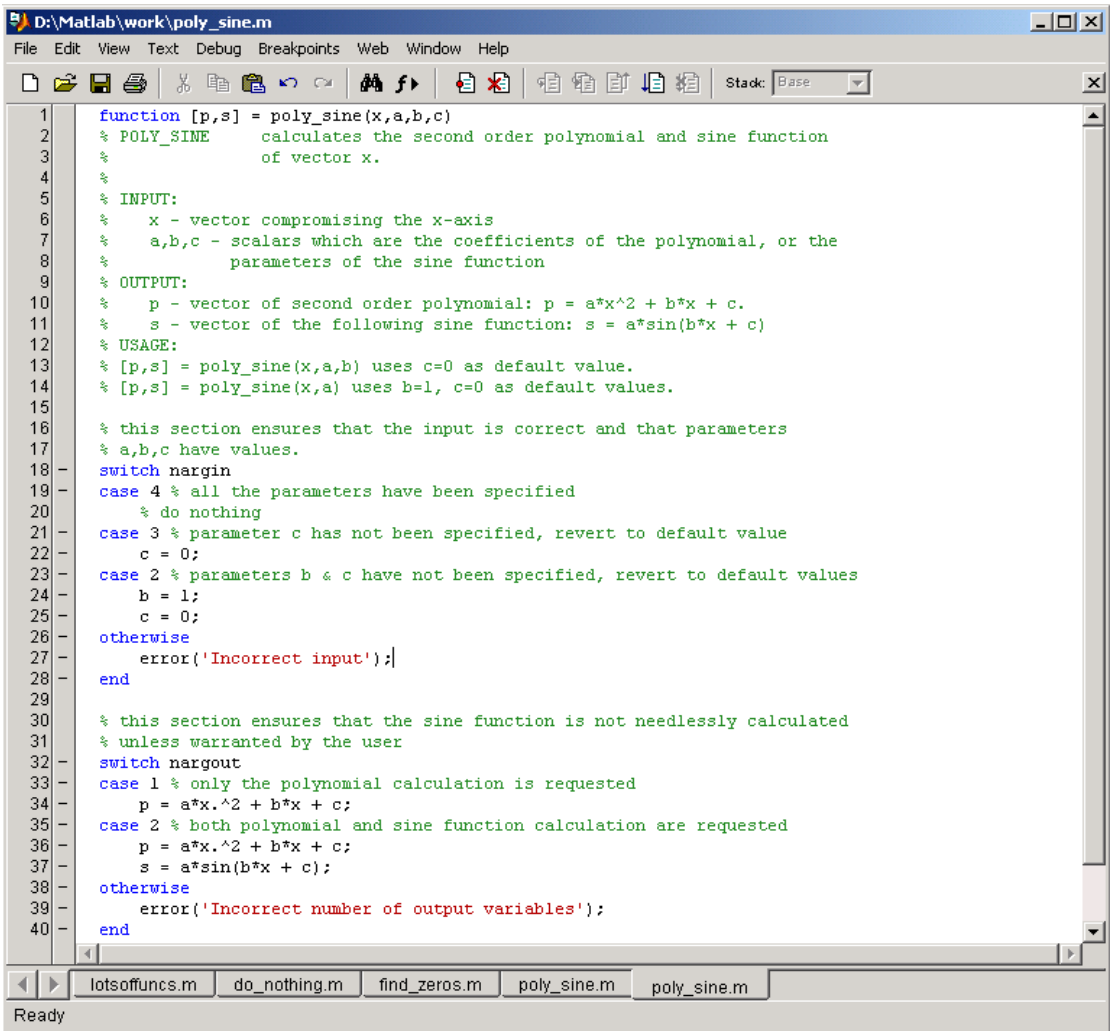
```
a = get(h);
set(h,'YData',a.YData + 10)
```

משנה את גרף 7.8 לגרף 7.9.
a הוא מערך structure המכיל את תכונות אובייקט ה-line.



גרף 7.9

8. חלון Editor/Debugger



גרף 8.1: חלון Editor/Debugger

חלון Editor/Debugger הוא החלון דרכו מבוצע הרוב המכריע של התכנות ב-Matlab. בחלון זה כותבים קבצי m. שהם קבצי הקוד ב-Matlab.
קיימים שני פורמטים לכתיבת קבצי m. האחד הוא קבצי script, כלומר הפקודות מבוצעות אחת אחרי השניה כאילו שהן נכתבו ב-Command Window. השימוש המרכזי בפורמט script הוא בחסיכת חזרה על רצף פקודות ב-Command Window. הפורמט השני הוא פונקציות המקבלות קלט ומחזירות פלט שמשתניה הם פנימיים ולא גלובליים. חוץ ממספר פונקציות בסיסיות המקומפלות ב-C, כל הפונקציות של Matlab בנויות כפונקציה בקובץ m. וניתן לראות פונקציות אלו באמצעות פונקצית type[30].
הרצת קבצי m. מבוצעת ע"י לחיצה על הסימן  או על כפתור F5 בחלון Editor/Debugger או דרך ה-Command Window.

הערות

- ניתן לשלב את חלון Editor/Debugger יחד עם שאר החלונות ע"י בחירת אופציית Dock בתפריט View. לחיצה על הסימן  בפינה הימנית עליונה בחלון כלשהו פותחת אותו בנפרד.
- סימן * בכותרת של קובץ m. מעיד על כך שהגרסה הנוכחית של הקובץ איננה שמורה.

[30] פונקצית type מציגה את תוכן פונקציות סטנדרטיות של Matlab (חוץ ממספר פונקציות המקומפלות ב-C).

8.1 יצירת הערות

יצירת הערות מבוצעת באמצעות סימן האחוזים % וכתיבת הטקסט הרצוי לאחריו. הערות מסומנות בצבע ירוק ו- Matlab מתעלם מהן לחלוטין.
ניתן להפוך מקטע שלם להערה ע"י הקלדת Ctrl+R או בחירת אופציית Comment בתפריט Text. ניתן להסיר את האחוזים ע"י סימון המקטע הרצוי והקלדת Ctrl+T או בחירת אופציית Uncomment בתפריט Text.

8.2 מלות מפתח

קיימות מספר מלות מפתח שאסור בכל מקרה להשתמש בהן אלא לפי ההגדרות של Matlab. מלות מפתח מסומנות בצבע כחול. להלן מלות המפתח,

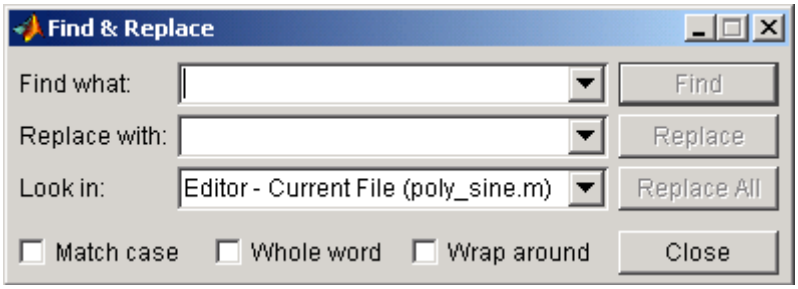
```
break,case,catch,continue,else,elseif,end,for,function,
global,if,otherwise,persistent,return,switch,try,while
```

דוגמא לשימוש פסול במלות המפתח הוא,

```
>> if = 3;
??? if = 3;
       |
Error: Expected a variable, function, or constant, found "=".
```

8.3 מציאת והחלפת תווים

לחיצה על הסימן  או לחיצה על Ctrl+F פותחת את חלון Find & Replace שבאמצעותו ניתן למצוא ולהחליף תווים,



גרף 8.2: חלון חיפוש והחלפה תווים

8.4 סימנייה (Bookmark)

הוספת והורדת סימנייה מבוצע ע"י השמת הסמן בשורה הרצויה ולחיצה על Ctrl+F2 או בחירת אופציית Set/Clear Bookmark בתפריט Edit.
לחיצה על F2 או בחירת אופציית Next Bookmark בתפריט Edit מעבירה את הסמן לסימניה הבאה.
לחיצה על Shift+F2 או בחירת אופציית Prev Bookmark בתפריט Edit מעבירה את הסמן לסימניה הקודמת.

אופציה זו שימושית בעיקר עבור קבצי m. ארוכים בהם ברצוננו לעבור בין שורות ספציפיות הרחוקות אחת מהשניה.

הערה

- סימניות אינן נשמרות בעת סגירת קבצים.

8.5 קיצורים שימושיים

להלן סיכום של מספר קיצורים שימושיים,

קיצור	פעולה
Ctrl+C	copy
Ctrl+V	paste
Ctrl+X	cut
Ctrl+Z	undo
Ctrl+Y	redo
Ctrl+A	select all
Ctrl+B	Balance Delimiters[31]
Ctrl+W	close file
Ctrl+N	new file
Ctrl+G	go to line
Ctrl+F	find & replace
Ctrl+R	add remark
Ctrl+T	remove remark
Ctrl+F2	Set/clear Bookmark
F2	Next Bookmark
Shift+F2	Prev Bookmark
Ctrl+S	Save
Ctrl+C	exit execution
Ctrl+Break	exit execution
Ctrl+Q	exit Matlab

טבלה 8.1: מספר קיצורים שימושיים

[31] משמעות אופציה זו היא סימון כל הקוד הנמצא בין הסוגריים שבהן נמצא הסמן.

9. בקרת זרימה (Flow Control)

להלן פקודות בקרת הזרימה ב-Matlab,

	פקודה	שימוש
פקודות התניה		יחד עם פקודות elseif ו-else מבצעת מקבץ פקודות על סמך תנאים לוגיים
	switch	יחד עם פקודות case ו-otherwise מבצעת מקבץ פקודות שונות ע"ס ערך של משתנה
פקודות לולאה	try…catch	משנה את ניתוב הפקודות אם מתגלה טעות בזמן הביצוע
		מבצעת מקבץ פקודות מספר בלתי מוגבל של פעמים, ע"ס תנאי לוגי
	for	מבצעת מקבץ פקודות מספר פעמים קבוע
פקודות יציאה מוקדמת	continue	מדלג לאיטרציה הבאה של לולאת for או while ללא ביצוע הפקודות הנותרות בלולאה
	break	מפסיק את ביצוע לולאת for או while
	return	גורם להרצה לחזור לפונקציה שקראה לה

טבלה 9.1

כל הפקודות הנ"ל שולטות בסדר ביצוע הקוד. לפקודות הלולאה וההתניה מצורפת פקודת end הקובעת את סיום הפקודה. בפקודות if ו-while תנאי ביצוע הקוד הוא תנאי לוגי. תנאי לוגי יכול להיות אמת[32] (ערך נומרי 1) או שקר (ערך נומרי 0). כלומר שתי פקודות אלו מבצעות קוד רק אם התנאי הלוגי הוא אמת.
ניתן לבנות תנאי לוגי מאופרטורים לוגיים או יחסיים המפורטים בטבלה 9.2.

אופרטורים יחסיים		אופרטורים לוגיים	
אופרטור	משמעות	אופרטור	משמעות
<	קטן	&	AND לוגי
<=	קטן שווה	|	OR לוגי
>	גדול	~	NOT לוגי
>=	גדול שווה	xor	XOR לוגי
==	שווה[33]	all	נכון אם כל האיברים שונים מאפס
~=	אינו שווה (שונה)	any	נכון אם קיים לפחות איבר אחד שונה מאפס

טבלה 9.2

ניתן גם לבנות תנאי לוגי באמצעות פונקציות הבודקות מצבים מהסגנון *is. למשל הפונקציה isempty בודקת אם מטריצה היא מטריצה ריקה.

דוגמאות,
התנאי,

```
1/3 > 1/2
```

מניב ערך 0.

התנאי,

```
(2>=1) & (4~=3)
```

מניב ערך 1.

[32] ב-Matlab כל ערך נומרי השונה מאפס נחשב לאמת לוגי.
[33] טעות נפוצה מאד היא כתיבת סימן = במקום ==. חשוב לזכור כי משמעות סימן = היא הצבה ומשמעות == היא השוואה.

מה קורה כאשר משתמשים במטריצות במקום סקלרים בתנאי לוגי?
התנאי הלוגי הוא אמת רק אם כל האיברים מקיימים אותו. למשל, נניח כי A, B מטריצות מאותו גודל[34].
משמעות התנאי A>B הוא שכל איברי A גדולים מכל איברי B התואמים.

[34] אם המטריצות אינן באותו גודל אז מתקבלת הודעת שגיאה.

9.1 פקודת if

הצורה הבסיסית ביותר של פקודת if היא,

```
if logic_condition
    statements
end
```

אם התנאי הלוגי הוא true אז כל הפקודות עד ה-end יבוצעו ו-Matlab ימשיך בביצוע הפקודות לאחר ה-end. אם התנאי הלוגי הוא false אז Matlab מדלג על כל הפקודות עד ה-end וממשיך בביצוע הפקודות לאחר ה-end.
לדוגמא,

```
if a<0
    disp('a is negative')
end
```

ב-Matlab ערך true מוגדר ככל ערך שונה מאפס. שני הרישומים הבאים שקולים,

```
if a~=0
    disp('a is not zero')
end
```

```
if a
    disp('a is not zero')
end
```

התנאי הלוגי של פקודת if יכול להיות גם מטריצה. במקרה זה התנאי הלוגי הוא אמת כאשר כל איברי המטריצה שונים מאפס.
למשל,

```
if A
    disp('all the values of matrix A are not zero')
end
```

השוואת בין מטריצה לסקלר,

```
if A==7
    disp('all the values of matrix A are equal to seven')
end
```

9.1.1 פקודות else ו-else if

פקודות else ו-elseif מוסיפות התניה נוספת.

<u>פקודת else</u>

אין לה תנאי לוגי. הפקודות הקשורות לה מבוצעות אם התנאי הלוגי של פקודת ה-if (או פקודת else if) היה false.
דוגמא,

```
if a<0
    disp('a is negative')
else
    disp('a is non negative')
end
```

פקודת else if

אם התנאי הלוגי של פקודת ה-if (או פקודת else if) הקודמת היה false אז התנאי הלוגי של פקודת ה-else if מחושב ואם הוא true אז הפקודות הקשורות לה מבוצעות.
לדוגמא,

```
if a<0
    disp('a is negative')
else if a>0
        disp('a is positive')
    else
        disp('a is 0')
    end
end
```

ביחד עם פקודת if ניתן להשתמש ב-else יחיד, ומספר בלתי מוגבל של פקודות else if.
דוגמא מורכבת,

```
if a<0
    disp('a is negative')
    if num == 2
        disp('a is negative and num equals 2')
    else
        disp('a is negative and num does not equal 2')
    end
else if a > 0
        disp('a is positive')
    else
        disp('a is equal to zero')
    end
end
```

מומלץ מאד להשתמש בפיסוק המבוצע באופן אוטומטי ע"י לחיצה על Enter בסוף כל שורה כדי להקל על הבנת הנכתב. ללא השימוש ב-indenting של Matlab מהר מאד הקוד הופך לבלתי מובן לחלוטין.
למשל אותו קוד ללא indenting,

```
if a<0
disp('a is negative')
if num == 2
disp('a is negative and num equals 2')
else
disp('a is negative and num does not equal 2')
end
else if a > 0
disp('a is positive')
else
disp('a is equal to zero')
end
end
```

הערה

ישנן מספר אפשריות לריווח (Indenting) של שורות קוד מסומנות,

1. הזזת הקוד המסומן ב-tab אחד שמאלה- Ctrl+[.
2. הזזת הקוד המסומן ב-tab אחד ימינה- Ctrl+].
3. ריווח לפי מלות המפתח- Ctrl+I.

9.2 פונקצית switch

פונקצית switch מבצעת פקודות ע"ס ערך של משתנה (סקלר או מחרוזת).
המבנה הכללי של פונקצית switch הוא,

```
switch expression
case value_1
    statements_1 % executes if expression is equal to value_1
case value_2
    statements_2 % executes if expression is equal to value_2
 ⋮

otherwise
    statements_other % executes if expression does not match any case
end
```

בפונקצית switch רק case אחד מבוצע בהתאם לערך של expression. אם ערך expression הוא לא אחד מהערכים המצוינים ב-case-ים אז מבוצעות הפקודות ב-otherwise. לעתים רבות במקרה otherwise מפסיקים את ההרצה[35] בגלל שהתקבל ערך לא רצוי או צפוי של expression.
דוגמא,
a הוא סקלר,

```
switch a
case 1
    disp('a=1')
case 2
    disp('a=2')
case 3
    disp('a=3')
otherwise
    disp('otherwise')
end
```

דוגמא,
name הוא מחרוזת,

```
switch name
case 'Shmulik'
    disp('name is Shmulik')
case 'Orit'
    disp('name is Orit')
otherwise
    disp('Unknown name')
end
```

פונקצית switch יכולה לבדוק מספר תנאים בו זמנית כאשר ביטוי התנאי הוא מערך תאים.
לדוגמא,

```
switch a
case 1
    disp('a=1')
case {2,3,4,5,6,7,8,9,10}
    disp('a is one of 2,3,4,...,10')
otherwise
    disp('otherwise')
end
```

[35] מציבים פונקצית error המפסיקה את ביצוע ההרצה.

9.3 פונקציות try ו-catch

המבנה הכללי של פונקציות try ו-catch הוא,

```
try
    statements_try
catch
    statements_catch
end
```

במבנה זה הפקודות statements_try מבוצעות עד אשר נוצרת שגיאה. אם לא נוצרה שגיאה בפקודות אלו, ההרצה נפסקת ללא ביצוע פקודות statements_catch. אם נוצרה שגיאה בפקודות statements_try אז Matlab עובר לביצוע הפקודות statements_catch. אם גם ב-statements_catch נוצרה שגיאה אז Matlab מפסיק את ההרצה.

דוגמא,

```
try
    x(0) = 7;
    disp('No errors have occured in the try statements')
catch
    disp('An error has occured in the try statements')
    x(-4) = 8;
    disp('No errors have occured in the catch statements')
end
```

פקודה ב-try מנסה להציב לווקטור באינדקס 0 וזוהי שגיאה. לכן מבוצעות פקודות catch עד לשגיאה הראשונה ב-statements_catch.
השימוש העיקרי בפונקציות try ו-catch הוא כאשר ידוע לנו שיש סיכוי טוב שבעת ביצוע פקודות statements_try תקרה תקלה שתפסיק את ביצוע הקוד. במקרה כזה, במקום שההרצה תיפסק, יבוצעו הפקודות האלטרנטיביות statements_catch.

9.4 לולאת while

לולאת while חוזרת על הפקודות הנמצאות בין פקודת while עד ל-end מספר בלתי מוגבל של פעמים עד שהתנאי הלוגי הופך ל-true. הרישום הכללי של לולאת while,

```
while logic_condition
    statements
end
```

כאשר אי-העמידה בתנאי הלוגי מחושב כל פעם בסוף ביצוע הפקודות statements.
נשים לב כי בלולאת while התנאי מחושב פעם אחת לפני כניסה לתוך הלולאה ולכן רצוי להציב ערך התחלתי כך שבפעם הראשונה התנאי הלוגי לא יתקיים כדי להיכנס לתוך הלולאה.

דוגמא,
הקוד הבא מסכם את איברי הוקטור עד לאיבר הראשון השלילי (לא כולל).

```
sum_x = 0;
ii = 1;
while x(ii)>=0 & ii <= length(x)
    sum_x = sum_x + x(ii);
    ii = ii + 1;
end
```

חשוב לשים לב כי תנאי העצירה יתקיים מתישהו, אחרת יתכן מצב שבו הלולאה לעולם לא תיפסק בעצמה. עצירת הרצה לפני סיומה מבוצעת ע"י לחיצה על Ctrl+C או Ctrl+Break.

9.5 לולאות for

לולאות for מבצעות מקבץ פקודות מספר קבוע מראש פעמים. צורתה הכללית,

```
for i = s:d:f
    statements
end
```

למרות שהמשתנה i לא חייב להיות אינדקס או אפילו מספר שלם, לעתים קרובות חלק מהפקודות משתמשות במשתנה i בתור אינדקס לוקטורים או מטריצות.
דוגמא,
הקוד הבא מחפש כמה פעמים מופיע המספר num בוקטור x.

```
N = length(x);
num = 7;
sum_num = 0;
for i=1:N
    if x(i)==num
        sum_num = sum_num + 1;
    end
end
```

מומלץ **מאד** להימנע מלולאות for ככל האפשר. Matlab היא שפה הפועלת באופן מהיר ויעיל על מטריצות. שימוש בפונקציות סטנדרטיות הפועלות על מטריצות ישפר את מהירות הקוד בסדרי גודל.
לדוגמא, ניתן ליישם את כל הקוד הנ"ל באופן הבא,

```
num = 7;
sum_num = length(find(x==num));
```

9.5.1 וקטור אינדקס

כדי לקבל וקטור אינדקס בכל איטרציה נשתמש במטריצה בהגדרת פקודת ה-for. משמעות הרישום הכללי,

```
for index = A
    statements
end
```

היא שבכל איטרציה k וקטור האינדקס הוא העמודה ה-k-ית של מטריצה A, כלומר A(:,k).
לדוגמא,
נתונה המטריצה,

$$A = \begin{bmatrix} 1 & 5 & 9 & 13 & 17 \\ 2 & 6 & 10 & 14 & 18 \\ 3 & 7 & 11 & 15 & 19 \\ 4 & 8 & 12 & 16 & 20 \end{bmatrix}$$

הקוד,

```
iter = 1;
for index = A
    disp(['The index in iteration No.',num2str(iter),' is:'])
    disp(index)
    iter = iter + 1;
end
```

מייצר ב-Command Window,

```
The index in iteration No.1 is:
     1
     2
     3
     4

The index in iteration No.2 is:
     5
     6
     7
     8

The index in iteration No.3 is:
     9
    10
    11
    12

The index in iteration No.4 is:
    13
    14
    15
    16

The index in iteration No.5 is:
    17
    18
    19
    20
```

נשים לב כי מספר האיטרציות הוא כמספר העמודות במטריצה.

9.5.2 הצבת איברים במטריצה באמצעות לולאות

בלולאות while ו-for לעתים רבות מציבים ערכים בתוך וקטורים ומטריצות כאשר i מסמן אינדקס.
קיימות שלוש דרכים עיקריות להציב איברים במטריצה באמצעות לולאות,

1. איתחול בסוגריים ריקות + הרחבה
משמעות אתחול משתנה בסוגריים ריקות הוא שגודל המשתנה לא ידוע, ויתעדכן בהתאם לפקודות הבאות.
דוגמא,
יישום פונקצית abs באמצעות לולאת for.

```
x_abs = [];
for i=1:length(x)
    if x(i)>=0
        x_abs = [x_abs x(i)];
    else
        x_abs = [x_abs -x(i)];
    end
end
```

בכל איטרציה x_abs גדל באיבר אחד.

2. הצבה ישירה ללא אתחול כלשהו

```
for i=1:length(x)
    if x(i)>=0
        x_abs(i) = x(i);
    else
        x_abs(i) = -x(i);
    end
end
```

שתי הדרכים הנ"ל שקולות ומניחות שלא ידוע מראש מה יהיה גודל המטריצה אליה מוצבים הערכים.

3. איתחול המטריצה לגודל ידוע מראש

בדוגמא זו ברור כי אורך הוקטור x_abs צריך להיות זהה לאורך הוקטור המקורי x. לכן ניתן לאתחל את x_abs כוקטור אפסים ולעדכנו באופן הבא,

```
x_abs = zeros(size(x));
for i=1:length(x)
    if x(i)>=0
        x_abs(i) = x(i);
    else
        x_abs(i) = -x(i);
    end
end
```

באיזו שיטה להשתמש?

אם ידוע מראש גודל המשתנה אליו רוצים להכניס ערכים אז עדיף לאתחל אותו במטריצת אפסים. הסיבה לכך היא שכאשר Matlab יודע מראש את גודל המשתנה הוא יודע כמה זיכרון הוא צריך להקצות לו ואינו צריך לעדכן את גודל המשתנה או את כמות הזיכרון שהוא מקצה לו.

Continue 9.6

פקודת continue בתוך לולאת for או while מדלגת על שאר הפקודות הנותרות באיטרציה הנוכחית וההרצה ממשיכה מהאיטרציה הבאה. אם פקודת continue מבוצעת מתוך לולאה הנמצאת בתוך לולאה חיצונית אז פקודת continue מדלגת על הפקודות הנותרות רק בתוך הלולאה הפנימית בלבד.

דוגמא,
נתונה מטריצה X [5×7] הבאה,

$$X=\begin{bmatrix}1&4&2&3&5&3&2\\1&2&3&4&5&3&1\\1&3&4&3&3&3&5\\4&3&2&1&2&5&4\\3&2&2&5&5&3&3\end{bmatrix}$$

ואנו רוצים לחשב מטריצה חדשה X_new אשר איבריה הם ממוצע האיבר התואם ב-X עם שני שכניו מאותה שורה. הנוסחא לחישוב איברים אלה היא,

$$X\_new(i,j)=\frac{X(i,j-1)+X(i,j)+X(i,j+1)}{3}$$

למשל האיבר ה-(3,3) במטריצה X_new יהיה,

$$X\_new(i,j)=\frac{X(i,j-1)+X(i,j)+X(i,j+1)}{3}=\frac{3+4+3}{3}=3.3333$$

הבעיה היחידה היא באזור הקצוות. למשל כדי לחשב את $X\_new(4,1)$ המשוואה דורשת את האיבר $X(4,0)$ שכמובן לא קיים. כדי לפתור בעיה זו נגדיר כי העמודה השמאלית ביותר והימנית ביותר של מטריצה X_new יורכבו ישירות מעמודות X. עבור עמודות אלו אין צורך לבצע את החישוב הנ"ל ולכן נשתמש בפקודת continue כדי לדלג על ביצוע החישובים,

```
for i = 1:5
    for j =1:7
        if j==1 | j==7
            X_new(i,j) = X(i,j);
            continue
        end
        X_new(i,j) = (X(i,j-1) + X(i,j) + X(i,j+1))/3;
    end
end
```

Break 9.7

פקודת break בתוך לולאת for או while מפסיקה את הלולאה ויוצאת ממנה. אם פקודת break מבוצעת מתוך לולאה הנמצאת בתוך לולאה חיצונית אז פקודת ה-break יוצאת מחוץ ללולאה הפנימית בלבד. אם פקודת break מבוצעת בתוך קובץ m. אך לא בתוך לולאה אז ביצוע הקובץ נעצר.

דוגמא,
נניח כי כעת רצוי לבצע את המיצוע לפי שכנים קרובים מאותה עמודה.

$$X=\begin{bmatrix}1&4&2&3&5&3&2\\1&2&3&4&5&3&1\\1&3&4&3&3&3&5\\4&3&2&1&2&5&4\\3&2&2&5&5&3&3\end{bmatrix}$$

כלומר המשוואה היא,

$$X\_new(i,j)=\frac{X(i-1,j)+X(i,j)+X(i+1,j)}{3}$$

כעת משוואת המיצוע ממוקמת בתוך לולאת שרצה על העמודות (j) בעוד שאנו רוצים לדלג על השורות הקיצוניות. לשם כך נשתמש בפקודת break כדי לצאת מלולאת העמודות,

```
for i = 1:5
    for j =1:7
        if i==1 | i==5
            X_new(i,:) = X(i,:);
            break
        end
        X_new(i,j) = (X(i-1,j) + X(i,j) + X(i+1,j))/3;
    end
end
```

9.8 יציאה מוקדמת מהרצת קבצי m.

פקודת return מחזירה את ההרצה לקובץ m. שקרא לקובץ m. הנוכחי או ל-Command Window אם קובץ m. הנוכחי הורץ דרכו.

דרך שימושית לצאת מהרצת קובץ m. כליל (כלומר לא רק יציאה לקובץ שקרא לקובץ הנוכחי) היא באמצעות פונקצית error. פונקציה זו מקבלת כקלט מחרוזת שאותה היא מציגה ב- Command Window לאחר שהפסיקה את ההרצה.
דוגמא,
נדרש להציב את הערך a במשתנה b אבל הערך a חייב להיות בתחום $0 < a < 10$. אחרת, חייבים להפסיק את המשך ביצוע ההרצה.
הקוד,

```
if a<=0
    error('a must be positive')
else if a>=10
        error('a must be smaller than 10')
    else
        b = a;
    end
end
```

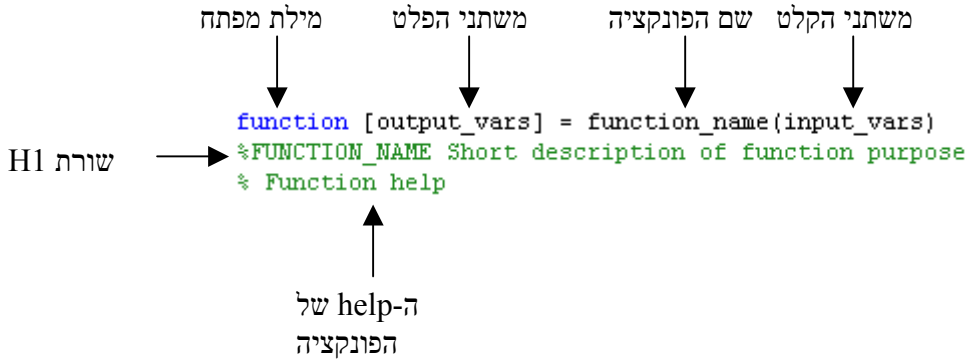
הערות

- שימוש נפוץ של פונקציה זו היא לבדיקת קלט שמקבלת פונקציה.
- כמו כן ניתן להשתמש בפונקצית warning כדי לשלוח הודעות אזהרה ל- Command Window אך לא להפסיק את ביצוע הקוד.

10. כתיבת פונקציות

להלן פירוט אפשרי של מבנה של פונקציה כללית,
1. שורת הגדרה – שורה זו מגדירה את שם הפונקציה ואת משתני הקלט והפלט שלה.
2. הסברים - מטרת הפונקציה, תיאור הקלט והפלט (סוג משתנה, גודל, משמעות).
3. בדיקת קלט - חלק זה מוודא כי הקלט הוא חוקי ותקין כדי שהפונקציה תתפקד כרצוי.
4. אתחול - קביעת ערכים התחלתים למשתנים.
5. חישוב - החלק המרכזי בו מבוצעים כל החישובים.

השורות הראשונות של פונקצית Matlab הן מהצורה הבאה,



הערות מיד לאחר שורת ההגדרה מרכיבות את ה-help של הפונקציה. כלומר הקלדת help ושם הפונקציה ב-Command Window מציגות את ההערות משורת H1 עד לשורה הראשונה שאיננה הערה[36]. שורת H1 מהווה הסבר קצר לפונקציה. הסבר זה מופיע בחלון Current Directory בתור Description ובנוסף פונקצית lookfor מחפשת אחר מילת מפתח בהסבר של כל פונקציה. כלומר פונקצית lookfor מחפשת מילה המופיעה בשורה H1 של הפונקציה.
דוגמא,

```
function [p,s] = poly_sine(x,a,b,c)
%POLY_SINE Calculates the second order polynomial & sine function
%
% INPUT:
%    x - vector comprimising the x-axis
%    a,b,c - scalars which are the coefficients of the polynomial, or the
%            parameters of the sine function
% OUTPUT:
%    p - vector of second order polynomial: p = a*x^2 + b*x + c.
%    s - vector of the following sine function: s = a*sin(b*x + c)
% USAGE:
%    [p,s] = poly_sine(x,a,b) uses c=0 as default value.
%    [p,s] = poly_sine(x,a) uses b=1, c=0 as default values.

p = a*x.^2 + b*x + c;
s = a*sin(b*x + c);
```

[36] כולל שורה ריקה.

כתיבת lookfor Calculates ב-Command Window מניבה,

```
>> lookfor Calculates
SIN_FUNC Calculates the sine function
POLY_SINE Calculates the second order polynomial & sine function
sin_func_varargin.m: %SIN_FUNC Calculates the sine function
BCKPRPNN Calculates backpropagation network output for NNPLS
⋮
```

כתיבת help poly_sine ב-Command Window מניבה,

```
>> help poly_sine

 POLY_SINE Calculates the second order polynomial & sine function

  INPUT:
     x - vector compromising the x-axis
     a,b,c - scalars which are the coefficients of the polynomial, or the
             parameters of the sine function
  OUTPUT:
     p - vector of second order polynomial: p = a*x^2 + b*x + c.
     s - vector of the following sine function: s = a*sin(b*x + c)
  USAGE:
  [p,s] = poly_sine(x,a,b) uses c=0 as default value.
  [p,s] = poly_sine(x,a) uses b=1, c=0 as default values.
```

שמות המשתנים בפונקציה אינם חייבים להיות זהים לשמות המשתנים שבאמצעותם קוראים לפונקציה.
למשל כתיבת,

```
x = 1:0.01:10;
a = 1;
b = 4;
c = -10;
[p,s] = poly_sine(x,a,b,c);
```

מניבה את אותן התוצאות כמו,

```
xx = 1:0.01:10;
d = 1;
e = 4;
f = -10;
[p,s] = poly_sine(xx,d,e,f);
```

למעשה הרישום,

```
[p,s] = poly_sine(xx,d,e,f);
```

מבצע הצבת המשתנים בסוגריים למשתנים הלוקליים[37] של הפונקציה,

```
x = xx;
a = d;
b = e;
c = f;
```

לכן ניתן גם להכניס ישירות את הערכים בתוך הסוגריים.
לדוגמא,

```
[p,s] = poly_sine(1:0.01:10,1,4,-10);
```

מניב את אותן התוצאות.

[37] נזכור כי ממשתנים לוקליים לא מוגדרים מחוץ לפונקציה.

10.1 פונקציות *nargin, nargout, nargchk*

10.1.1 פונקצית nargin (number of input arguments)

פונקצית nargin מחזירה את מספר המשתנים בקלט של פונקציה.
לדוגמא, קריאה לפונקציה poly_sine באופן הבא,

```
[y,z,w] = poly_sine(x,a,b,c);
```

nargin ← 4

פונקציה זו חשובה בעיקר בשני מקרים,
1. חלק חשוב של כל פונקציה הוא בדיקת תקינות הקלט. הצבת מספר משתנים גדול מדי גורר שגיאה והסבר.
לדוגמא הצבת 5 משתנים בקלט,

```
>> [p,s] = poly_sine(1:0.01:10,1,4,-10,5)
??? Error using ==> poly_sine
Too many input arguments.
```

הצבת מספר משתנים קטן מדי יגרור הודעות שגיאה על משתנים לא מוגדרים. אם ברצוננו לשלוט על הודעות השגיאה ולהגדיר הודעות שגיאה יותר ברורות לטעמנו אנו זקוקים לפונקציה המחזירה את מספר הארגומנטים בקלט של פונקציה.

2. חלק ניכר של הפונקציות הסטנדרטיות של Matlab ניתנות להרצה עם מספר שונה של משתנים.
לדוגמא מה-help של פונקצית diff אנו למדים כי ניתן להריצה עם 1-3 ארגומנטים בקלט,

```
DIFF Difference and approximate derivative.
   DIFF(X), for a vector X, is [X(2)-X(1)  X(3)-X(2) ... X(n)-X(n-1)].
   DIFF(X), for a matrix X, is the matrix of row differences,
      [X(2:n,:) - X(1:n-1,:)].
   DIFF(X), for an N-D array X, is the difference along the first
      non-singleton dimension of X.
   DIFF(X,N) is the N-th order difference along the first non-singleton
      dimension (denote it by DIM). If N >= size(X,DIM), DIFF takes
      successive differences along the next non-singleton dimension.
   DIFF(X,N,DIM) is the Nth difference function along dimension DIM.
      If N >= size(X,DIM), DIFF returns an empty array.
```

כאשר מריצים פונקציה עם פחות משתנים מהמקסימום אז הערכים שלא הוגדרו מקבלים ערכי ברירת מחדל (default). כך משתמשים אינם צריכים לפרט את כל האפשריות אם הם משתמשים באפשריות ברירת המחדל. לכן כדי לאפשר לפונקציות לטפל במספר שונה של משתני קלט אנו זקוקים לפונקציה המחזירה את מספר הארגומנטים בקלט של פונקציה.

10.1.2 פונקצית nargout (number of output arguments)

פונקצית nargout מחזירה את מספר המשתנים בפלט של הפונקציה.
לדוגמא, קריאה לפונקציה poly_sine באופן הבא,

```
[y,z,w] = poly_sine(x,a,b,c);
```

nargout ← 3

פונקציה זו חשובה בעיקר כאשר רוצים לחסוך בחישובים מיותרים.
מה קורה כאשר פונקציה מוגדרת כך שהיא מחזירה מספר משתנים אך המשתמש מבקש פחות משתנים?

התשובה היא שכל החישובים בפונקציה יבוצעו, אך רק המשתנים הראשונים יוחזרו.
לדוגמא, אם ברצוננו לקבל רק את הפולינום נרשום,

```
w = poly_sine(1:0.01:10,1,4,-10);
```

ובמשתנה w יוצב ערך הפולינום (התקבל מקודם ב-p).
אך גישה זו היא בזבזנית מאחר ומושקעים משאבים לחישוב ערכי משתני פלט שהמשתמש כלל לא ביקש.
בדוגמא זו שימוש ב-nargout יחסוך רק שורת קוד אחת אך באופן כללי, קיימים שני מקרים בהם חשוב להשתמש ב-nargout,

1. חישוב חלק ממשתני הפלט תובעני מבחינת עומס חישוב.
2. החישוב לא תובעני, אך מספר הקריאות לפונקציה גדול מאד.

לסיכום,
בד"כ משתמשים בפונקצית nargin כדי לוודא שמספר המשתנים בקלט תקין ולבחור בין אפשריות ערכי ברירות מחדל. בד"כ משתמשים בפונקצית nargout כדי לחסוך בחישוב משתנים שהמשתמש לא ביקש אותם.
נשנה את פונקצית poly_sine בהתאם,

```
function [p,s] = poly_sine(x,a,b,c)
%POLY_SINE Calculates the second order polynomial & sine function
%
% INPUT:
%    x - vector compromising the x-axis
%    a,b,c - scalars which are the coefficients of the polynomial, or the
%            parameters of the sine function
% OUTPUT:
%    p - vector of second order polynomial: p = a*x^2 + b*x + c.
%    s - vector of the following sine function: s = a*sin(b*x + c)
% USAGE:
% [p,s] = poly_sine(x,a,b) uses c=0 as default value.
% [p,s] = poly_sine(x,a) uses b=1, c=0 as default values.

% this section ensures that the input is correct and that parameters
% a,b,c have values.
switch nargin
case 4 % all the parameters have been specified
    % do nothing
case 3 % parameter c has not been specified, revert to default value
    c = 0;
case 2 % parameters b & c have not been specified, revert to default values
    b = 1;
    c = 0;
otherwise
    error('Incorrect input');
end

% this section ensures that the sine function is not needlessly calculated
% unless warranted by the user
switch nargout
case 1 % only the polynomial calculation is requested
    p = a*x.^2 + b*x + c;
case 2 % both polynomial and sine function calculation are requested
    p = a*x.^2 + b*x + c;
    s = a*sin(b*x + c);
otherwise
    error('Incorrect number of output variables');
end
```

הערה

דרך אלטרנטיבית לשימוש בפונקצית switch לביצוע חישובים בצורה סלקטיבית היא שימוש ב-if, else יחד עם return ליציאה מוקדמת מהפונקציה.
לדוגמא,

```
% this section ensures that the sine function is not needlessly calculated
% unless warranted by the user
if nargout == 1 | nargout == 2
    % the number of output variables is legal
    p = a*x.^2 + b*x + c;
    if nargout == 1
        return
    end
    s = a*sin(b*x + c);
else
    error('Incorrect number of output variables');
end
```

10.1.3 פונקצית nargchk

וריאציה חביבה על nargin היא פונקצית nargchk. פונקציה זו בודקת אם מספר הארגומנטים בקלט נע בין ערך low לערך high. רישומה הכללי,

```
msg = nargchk(low,high,number)
```

את המספר number מקבלים מפונקצית ה-nargin ומחזירים את ההודעה לפונקצית error או warning.

לדוגמא נתונה הפונקציה,

```
function [p,s] = poly_sine(x,a,b,c)
```

נקבע כי מספר המשתנים התקין בקלט שלה הוא בין 2 ל-4. נרשום,

```
error(nargchk(2,4,nargin))
```

אם, למשל, נכניס 5 משתנים בקלט נקבל את הודעת השגיאה,

```
??? Error using ==> poly_sine
Too many input arguments.
```

אם, למשל, נכניס רק משתנה יחיד בקלט נקבל את הודעת השגיאה,

```
??? Error using ==> poly_sine
Not enough input arguments.
```

ניתן גם להשתמש בפונקציה זו גם לבדיקת מספר המשתנים בפלט ע"י הצבת nargout במקום nargin.
לדוגמא הוספת,

```
error(nargchk(1,2,nargout));
```

בפונקצית poly_sine וקריאה לפונקציה עם יותר מדי ארגומנטים גוררת הודעת שגיאה,

```
??? Error using ==> poly_sine
Too many output arguments.
```

10.2 פונקציות של פונקציות

במקרים רבים רצוי לכתוב פונקציה המבצעת חישובים על כל פונקציה שהיא מקבלת במקום לכתוב פונקציה לכל מקרה. דוגמא פשוטה היא פונקציה המחשבת את האפסים של פונקציות מתמטיות כלשהן (סינוסים, פולינומים וכו').
פונקציה של פונקציות צריכה להיכתב כך שבאמצעות קוד כללי,

1. מבוצעות פונקציות אחרות ע"ס שם הפונקציה.
2. מבוצעות פונקציות אחרות בעלות מספר בלתי ידוע מראש של ארגומנטים בקלט מסוגים בלתי ידועים מראש.

10.2.1 ביצוע פונקציות ע"ס שמן באמצעות פונקצית feval

פונקצית feval משתמשת במושג function handle[38] שמוגדר ב-Matlab באמצעות סימן @.
function handle הוא מבנה מידע המכיל מידע לגבי הפונקציה. פונקצית feval משתמשת ב- function handle (ארגומנט ראשון) כדי לקרוא לפונקציה הרצויה ומציבה בה את שאר הארגומנטים לפי הסדר.
לדוגמא, שני הרישומים הבאים שקולים,

```
func = @eig;
[V,D] = feval(func,A)
```

⇕

```
[V,D] = eig(A)
```

קיימות מספר פונקציות סטנדרטיות המשתמשות בפונקצית feval. להלן מבחר קטן מתוכן,
fzero- מוצאת פתרונות למשוואה $f(x)=0$.
fminbnd- מוצאת נקודת מינימום לוקלית של $f(x)$ באינטרוול מוגדר.
fsolve- פותרת מערכת משוואות לא לינאריות באופן נומרי.
ode45- פותרת מערכת משוואות דיפרנציאליות רגילות באופן נומרי.
quadl- מבצעת אינטגרציה על $f(x)$ באופן נומרי.

<u>אובייקט inline</u>

צורה מקוצרת ליצירת פונקציה היא באמצעות אובייקט inline.
פונקצית inline מקבלת מחרוזות ומייצרת פונקצית Matlab המחשבת שורת קוד יחידה ומחזירה ארגומנט פלט יחיד. הארגומנט הראשון מייצג את שורת הקוד שתבוצע. שאר הארגומנטים מייצגים את המשתנים והם מתפקדים כמו משתני קלט ברישום של פונקציה סטנדרטית. פונקצית inline סורקת את המחרוזת בארגומנט הקלט הראשון שלה ומחפשת אחר אופרטורים ופונקציות מוכרות (כמו sin,+,*,...) ואחר המשתנים שהוגדרו.

לדוגמא שני הרישומים הבאים יוצרים פונקציות המתפקדות באופן זהה,
1. רישום סטנדרטי של פונקציה

```
function  f = sin_func(x,A,w,phi)
f = A*sin(w*x + phi);
```

2. אובייקט inline

[38] ניתן גם לכתוב את שם הפונקציה בתור מחרוזת במקום להשתמש ב-function handle אך מסיבות המפורטות ב-help עדיף להשתמש ב-function handle.

```
f = inline('A*sin(w*x + phi)','x','A','w','phi');
```

כדי לחשב את פונקצית הסינוס בשתי הגישות הנ"ל נרשום כרגיל,

```
f(x,A,w,phi)
```

היתרון באובייקט inline הוא בכך שרישום זה מהיר יותר עבור כתיבת פונקציות בעלות חישוב בודד כמו נוסחה ואין צורך להגדיר פונקציה בקובץ m. נפרד.
החסרון ברישום זה הוא שאי אפשר לכתוב פונקציה בעלת יותר מפקודה אחת, או להשתמש באובייקט inline אחר, או להחזיר בפלט יותר ממשתנה אחד.

דוגמא לשימוש באובייקט inline עבור פונקצית Matlab סטנדרטית,
נתונה הפונקציה המתמטית,

$$f(x)=\frac{e^{\cos(\pi x)}}{x+1},\quad x\in[0,5]$$

ומטרתינו היא למצוא נקודת מינימום של הפונקציה הזו בתחום הנ"ל. כדי למצוא את נקודת המינימום באופן אנליטי צריך להשוות את הנגזרת לאפס ולפתור את המשוואה הבאה,

$$f'(x)=\frac{-e^{\cos(\pi x)}\left[\pi(x+1)\sin(\pi x)+1\right]}{(x+1)^2}=0$$

פתירת משוואה זו קשה מאד ולכן נחפש אחר נקודת המינימום באופן נומרי באמצעות פונקצית fminbnd הקוד,

```
f = inline('exp(cos(pi*x)./(x+1))');
x_min = fminbnd(f,0,5);

y_min = f(x_min);
x = linspace(0,5,1000);
y = f(x);
plot(x,y,x_min,y_min,'r*')
title('{f(x)=e^{cos(\pix)}}/{(x+1)}')
xlabel('x'),ylabel('y')
legend('e^{cos(\pix)}/(x+1)','x_{min}')
```

מניב את גרף 10.1.

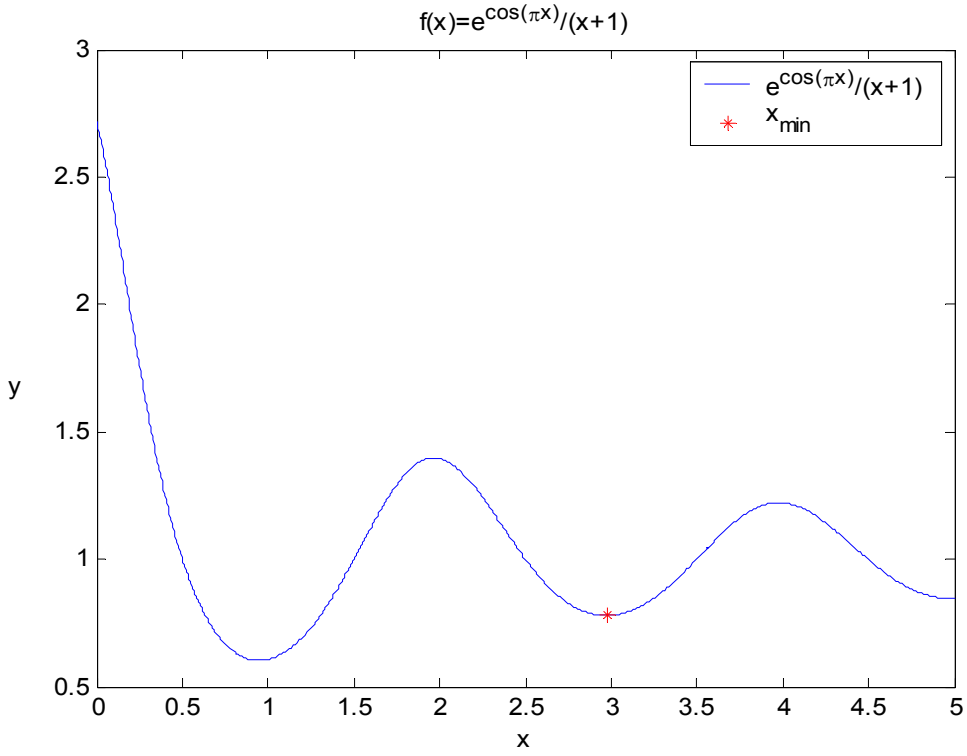
לסיכום, הארגומנט הראשון של פונקצית feval יכול להיות,
1. function handle (הוגדרה פונקציה בקובץ m. נפרד).
2. מחרוזת (הוגדרה פונקציה בקובץ m. נפרד).
3. אובייקט inline וללא גרשיים.

הערה

- קיימת פונקציה בשם eval המבצעת כל מחרוזת הנמצאת בסוגריים שלה. למשל,

```
eval('clear')
```

מבצעת את הפקודה clear.



גרף 10.1: פונקצית fminbnd מצאה נקודת מינימום לוקלית

10.2.2 הגדרת פונקציות המקבלות מספר בלתי מוגדר מראש של משתנים

ישנן פונקציות החייבות להיות מסוגלות לקלוט מספר בלתי מוגדר מראש של משתנים. הדוגמא הכי פשוטה של פונקציה כזו היא פונקצית plot היכולה לקבל מספר כלשהו של גרפים לשרטוט[39]. אפשרות זו חיונית בהגדרה של פונקציה כללית הפועלת על כל פונקציה באמצעות פונקצית feval.
המחשה,
אנו רוצים למצוא את האפסים של שתי הפונקציות (המתמטיות) הבאות,

$$\begin{cases} f(x) = A\sin(\omega x + \phi) \\ f(x) = ax^4 + bx^3 + cx^2 + dx + e \end{cases}$$

לפונקצית הסינוס ישנם 4 פרמטרים של קלט: x, A, ω, ϕ
לפונקצית הפולינום הנ"ל ישנם 6 פרמטרים של קלט: x, a, b, c, d, e
פונקציות אלו מקודדות באופן הבא,

```
function  f = sin_func(x,A,w,phi)
f = A*sin(w*x + phi);

function  f = poly_func(x,a,b,c,d,e)
f = a*x.^4 + b*x.^3 + c*x.^2 + d*x + e;
```

כדי להשתמש ב-feval עבור פונקצית הסינוס נרשום,

```
func = @sin_func;
f = feval(func,arg1,arg2,arg3,arg4);
```

ומתבצעת ההצבה הבאה,

$arg\,1 \leftarrow x$ $arg\,3 \leftarrow w$
$arg\,2 \leftarrow A$ $arg\,4 \leftarrow phi$

אך כדי לחשב את פונקצית הפולינום צריך עוד שני ארגומנטים.
כלומר כדי להשתמש ב-feval עבור פונקצית הפולינום נרשום,

```
func = @poly_func;
f = feval(func,arg1,arg2,arg3,arg4,arg5,arg6);
```

ומתבצעת ההצבה הבאה,

$arg\,1 \leftarrow x$ $arg\,3 \leftarrow b$ $arg\,5 \leftarrow d$
$arg\,2 \leftarrow a$ $arg\,4 \leftarrow c$ $arg\,6 \leftarrow e$

כלומר כדי להשתמש ב-feval עם פונקצית הסינוס חייבים 4 ארגומנטים בדיוק[40]. כדי להשתמש ב-feval עם פונקצית הפולינום חייבים 6 ארגומנטים בדיוק. לכן בגישה זו אין כל ברירה אלא לכתוב פונקצית feval **שונה** לכל מקרה. אך נזכור כי מטרת השימוש בפונקצית feval היא פונקציה כללית.

קיימות שתי גישות לפתרון בעיה זו,
1. פונקצית varargin
2. שימוש במערך structure

[39] למעשה פונקציה זו כתובה ב-C ולכן איננה משתמשת ב-varagin.
[40] לא כולל Function Handle.

פונקציית varargin

אופן השימוש בפונקצית varargin הוא,

```
function [output_vars] = function_name(input_vars,varargin)
```

הקלט מורכב ממספר ארגומנטים והארגומנט האחרון הוא varargin. כל הארגומנטים של הקלט עד לארגומנט במיקום של varargin הם ארגומנטי קלט רגילים. כל ארגומנט קלט מהמיקום של varargin והלאה מוצב בתוך מערך תאים בשם varargin.
דוגמא,
נתונה פונקציה המוגדרת באופן הבא,

```
function how_many_inputs(x,varargin)
```

קריאה לפונקציה הנ"ל באופן הבא,

```
how_many_inputs(2,'one',2,3*eye(2))
```

מציבה במשתנה x ערך 2 ומייצרת מערך תאים בשם varargin בעל הערכים,

```
varargin{1} =

one


varargin{2} =

     2



varargin{3} =

     3     0
     0     3
```

כלומר כל הארגומנטים החל מהארגומנט השני הוצבו במערך התאים בשם varargin.
באופן הזה ניתן להגדיר פונקציה המקבלת מספר כלשהו של ארגומנטי קלט. אך כיצד משתמשים במערך התאים varargin בתוך הפונקציה?
אם ידועים סוגי מבני המידע שיהיו במערך התאים varargin אז ניתן לחלצם מ-varargin ולהציבם לתוך משתנים מתאימים.

לדוגמא מוגדרת הפונקציה,

```
function x = scalars2vector(varargin)
%SCALARS2VECTOR Converts scalars into a row vector
%
% INPUT:
%    varargin - a cell array containing the scalar
% OUTPUT:
%    x - a vector containg the scalars of varargin

n = length(varargin);
x = zeros(1,n);

for i=1:n
    x(i) = varargin{i};
end
```

הממירה סקלרים לתוך וקטור שורה. כלומר,

```
x = scalars2vector(1,2,3,4,5);
```

$x \Leftarrow [1 \quad 2 \quad 3 \quad 4 \quad 5]$

במקרה הנ"ל היה ידוע כי ב-varargin יוצבו סקלרים ולכן ניתן היה להגדיר כי x הוא וקטור שיקלוט את הסקלרים מתאי varargin. אך מה קורה כאשר סוגי מבני המידע שיוצבו ב-varargin אינם ידועים מראש?
המקרה הכי בולט הוא פונקצית feval הצריכה לחלץ את המשתנים מסוגים שונים מתוך מערך התאים, ולתוך הסוגריים שלה. הפתרון לבעיה זו הוא האופרטור היחודי {:} כפי שיודגם בדוגמת הסיכום.

כדוגמת סיכום של פונקציות feval, varargin, function handles נבחן את הפונקציה הבאה המחשבת את האפסים של פונקציה כללית ונדגים עבור פונקצית סינוס. לשם נכתבו שלושה קבצי m.,

1. קובץ script.
2. פונקצית הסינוס.
3. פונקצית find_zeros_varargin.

.1

```
clear
clc
close all

x_range = [0 20];
func = @sin_func_varargin;

zero_locations = find_zeros_varargin(func,x_range,0.1,1e-4,1,pi/4,pi/2);

% graph section
x_axis = linspace(min(x_range),max(x_range),1000);
figure
plot(x_axis,feval(func,x_axis,1,pi/4,pi/2))
xlabel('x'),ylabel('y')
grid

disp(['The zeros of the function in the range [',num2str(x_range),'] are:'])
disp(zero_locations)
```

הארגומנטים בקלט של הפונקציה שיוצבו במערך התאים varargin

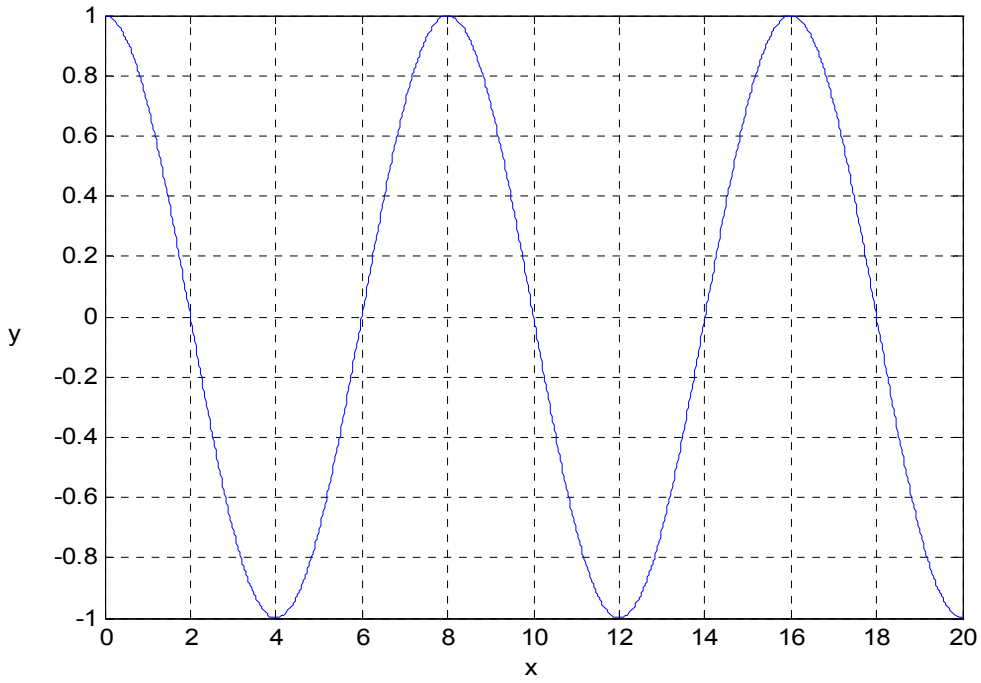
.2

```
function f = sin_func_varargin(t,A,w,phi)
%SIN_FUNC Calculates the sine function
%               f(t) = A*sin(w*t + phi)
%
% INPUT:
%    t - a vector specifying the x-axis.
%    A - a scalar specifying the amplitude
%    w - a scalar specifying the radial frequency
%    phi - a scalar specifying the phase
% OUTPUT:
%    f - a vector of the sine function

f = A*sin(w*t + phi);
```

.3

```
function zero_locations = find_zeros_varargin(func,x_range,dx,tol,varargin)
%FIND_ZEROS Finds all the zeros of function 'func' within specified range
%
% INPUT:
%    func - a string of the name of the mathematical function
%           whose zeros the user wants to locate.
%    x_range - a [2 x 1] vector containing the range in which the
%              algorithm will search for zeros.
%    dx - a scalar specifying the initial increment factor.
%    tol - a scalar specifying the degree of tolerance to which the
%          algorithm will search.
%    varagin - a cell array containing the parameters of the function func.
% OUTPUT:
%    zero_locations - a vector containing all the zero locations found in
%                     the specified range.
% Alternative USAGE:
%    find_zeros_varargin(func,x_range,dx,parameters) - tol is set to default 1e-6
%    find_zeros_varargin(func,x_range,parameters) - tol is set to default 1e-6 and
%                                                   dx is set to default 1e-2


% this section ensures that the input is correct and that parameters
% dx,tol have values.
switch nargin
case 4 % tol parameter has not been specified, revert to default value
    tol = 1e-6;
case 3 % dx, tol parameters have not been specified, revert to default values
    dx = 1e-2;
    tol = 1e-6;
case {1,2}
    error('Not enough input arguments');
end

% initial values
dx_original = dx;
x = min(x_range); % begin the search from left to right
x_end = max(x_range);
zero_locations  = [];
sign_x = sign(feval(func,x,varargin{:}));

% main loop
while x<=x_end
    sign_x_plus_dx = sign(feval(func,x + dx,varargin{:}));

    if sign_x ~= sign_x_plus_dx
        dx = dx/2;
    else
        x = x + dx;
        % the sign is of the function at x + dx is the same as at location x
    end

    if dx <= tol
        % a zero has been found because the interval [x,x+dx] in which the zero
        % must be located is smaller than tol.
        zero_locations = [zero_locations x + dx/2]; %the zero is in the middle of the interval
        sign_x = sign_x_plus_dx; % the sign has switched
        x = x + dx; % update x
        dx = dx_original;
    end
end
```

גרף 10.2 מתקבל.



גרף 10.2: פונקצית הסינוס $f(x)=\sin\left(\frac{\pi}{4}x+\frac{\pi}{3}\right)$ בתחום $0\le x\le 20$

ב-Command Window מוצגות התוצאות הבאות,

```
>> The zeros of the function in the range [0  20] are:
    1.9999    6.0000   10.0000   14.0000   18.0000
```

הטריק {:}varargin שקול להצבת מערך התאים איבר איבר בתוך הסוגריים של פונקציה. כלומר,

```
function_name(varargin{:})
      ⇕
function_name(varargin{1},varargin{2}, ... )
```

הערות

- קיימת פונקציה משלימה עבור משתני הפלט בשם varargout.
- פונקציות Matlab סטנדרטיות משתמשות בפונקציות varargin, ו- varargout.

שימוש במערך structure

משמעות השימוש בפונקצית varargin היא הצבת משתנים מסוגים שונים בתוך cell array בקלט וחילוצם מתוך מערך התאים לשימוש בתוך הפונקציה. גישה אלטרנטיבית היא הצבה במערך structure.

בחלק זה נשכתב את שלושת קבצי m. שהשתמשו בפונקצית varargin כך שהאלגוריתם יבוצע באמצעות שימוש במערך structure. להלן פירוט הקוד המשוכתב,
שלושה קבצי m. המשוכתבים הם,

1. קובץ script.
2. פונקצית הסינוס.
3. פונקצית find_zeros.

.1

```
clear
clc
close all

x_range = [0 20];

params = struct('A',1,'w',pi/4,'phi',pi/2);
func = @sin_func;

zero_locations = find_zeros(func,x_range,params,0.1,1e-4);

% graph section
x_axis = linspace(min(x_range),max(x_range),1000);
figure
plot(x_axis,feval(func,x_axis,params))
xlabel('x'),ylabel('y')
grid

disp(['The zeros of the function in the range [',num2str(x_range),'] are:'])
disp(zero_locations)
```

.2

```
function f = sin_func(t,params)
%SIN_FUNC Calculates the sine function
%               f(t) = A*sin(w*t + phi)
%
% INPUT:
%    t - a vector specifying the x-axis.
%    params - a strcture containing all the neccesary parameters
%       A - a scalar specifying the amplitude
%       w - a scalar specifying the radial frequency
%       phi - a scalar specifying the phase
% OUTPUT:
%    f - a vector of the sine function

w = params.w;
phi = params.phi;
A = params.A;

f = A*sin(w*t + phi);
```

.3

```
function zero_locations = find_zeros(func,x_range,params,dx,tol)
%FIND_ZEROS Finds all the zeros of function 'func' within specified range
%
% INPUT:
%    func - a string of the name of the mathematical function
%           whose zeros the user wants to locate.
%    x_range - a [2 x 1] vector containing the range in which the
%              algorithm will search for zeros.
%    dx - a scalar specifying the initial increment factor.
%    tol - a scalar specifying the degree of tolerance to which the
%          algorithm will search.
%    params - a structure containing the parameters of the function func.
% OUTPUT:
%    zero_locations - a vector containing all the zero locations found in
%                     the specified range.
% Alternative USAGE:
%    find_zeros_varargin(func,x_range,dx,parameters) - tol is set to default 1e-6
%    find_zeros_varargin(func,x_range,parameters) - tol is set to default 1e-6 and
%                                                   dx is set to default 1e-2

% this section ensures that the input is correct and that parameters
% dx,tol have values.
switch nargin
case 5
    % do nothing
case 4 % tol parameter has not been specified, revert to default value
    tol = 1e-6;
case 3 % dx, tol parameters have not been specified, revert to default values
    dx = 1e-2;
    tol = 1e-6;
otherwise
    error('Incorrect input');
end

% initial values
dx_original = dx;
x = min(x_range); % begin the search from left to right
x_end = max(x_range);
zero_locations  = [];
sign_x = sign(feval(func,x,params));

% main loop
while x<=x_end
    sign_x_plus_dx = sign(feval(func,x + dx,params));

    if sign_x ~= sign_x_plus_dx
        dx = dx/2;
    else
        x = x + dx;
        % the sign is of the function at x + dx is the same as at location x
    end

    if dx <= tol
        % a zero has been found because the interval [x,x+dx] in which the zero
        % must be located is smaller than tol.
        zero_locations = [zero_locations x + dx/2]; %the zero is in the middle of the interval
        sign_x = sign_x_plus_dx; % the sign has switched
        x = x + dx; % update x
        dx = dx_original;
    end
end
```

השינויים בקבצים הם,
1. קוד ה-script
לפני קריאה לפונקציה find_zeros איתחול מערך ה-structure בערכים מתאימים. הוגדר מערך בשם params המכיל את הפרמטרים של כל פונקציה,

```
params = struct('A',1,'w',pi/4,'phi',pi/2)
params =

      A: 1
      w: 0.7854
    phi: 1.0472
```

2. פונקצית הסינוס
חילוץ המשתנים[41] מתוך מערך ה-structure הנ"ל,

```
w = params.w;
phi = params.phi;
A = params.A;
```

3. פונקצית find_zeros
שימוש רגיל ב-feval מאחר ומספר הארגומנטים של הקלט ידוע מראש.

מתקבל אותו גרף ותוצאות כמו באמצעות פונקצית varargin.

סיכום פרק פונקציות של פונקציות

פונקציה של פונקציות צריכה להיכתב כך שבאמצעות קוד כללי,
1. מבוצעות פונקציות אחרות ע"ס שם הפונקציה באמצעות feval + (function handle או מחרוזת או אובייקט inline).
2. מבוצעות פונקציות אחרות בעלות מספר בלתי ידוע מראש של ארגומנטים בקלט מסוגים בלתי ידועים מראש באמצעות פונקצית varargin או מערך structure.

[41] שלב חילוץ המשתנים מתוך מערך ה-structure איננו הכרחי וניתן להציב את ערכי המשתנים מה-structure באופן ישיר עם ציון נקודה ושם השדה.

10.3 משתנים גלובליים

משתנים בתוך פונקציה הם לוקליים, כלומר מוגדרים רק בתוך אותה פונקציה ולא בפונקציות שקראו לה או ב-Command Window. לעומת זאת משתנה גלובלי מוגדר בכל פונקציה וניתן לשנות את ערכו בכל מקום (כלומר שינוי ערך של משתנה גלובלי אינו משנה רק עותק של המשתנה אלא את המשתנה עצמו). הגדרת משתנים גלובליים מבוצעת באמצעות פונקצית global.

לפני שימוש במשתנה גלובלי צריך להגדירו, בד"כ בקובץ script או ב-Command Window, כמשתנה גלובלי ולהציב בו ערך. למרות שאין זה הכרחי, מומלץ לציין משתנה גלובלי באותיות גדולות כדי לזכור כי זהו אכן משתנה גלובלי.

ישנם מספר שימושים למשתנים גלובליים. למשל, קביעת גדלים קבועים שאין ברצוננו לשנות ואין ברצוננו להגדירם לכל פונקציה (כמו קבוע פיסיקלי כמהירות האור). ניתן גם להשתמש במשתנים גלובליים בתור מונים הסופרים, למשל, את מספר הפעמים שקוראים לפונקציה. לדוגמא, בקובץ script נרשום,

```
global NUM_CALL
NUM_CALL = 0;
```

ונגדיר פונקציה,

```
function b = do_nothing(a)
global NUM_CALL
NUM_CALL = NUM_CALL + 1;
b = a;
```

בכל פעם שנקרא לפונקציה הנ"ל המונה NUM_CALL גדל ב-1. נשים לב כי בפונקציה צריך לקרוא למשתנה הגלובלי לפני שמשתמשים בו.
אם לא היינו משתמשים במשתנה גלובלי במקרה זה היינו צריכים להכניס בקלט פונקצית do_nothing את המונה ולהוציאו בפלט. כלומר,

```
function [b,num_call] = do_nothing(a,num_call)
num_call = num_call + 1;
b = a;
```

דרך זו פחות טובה משימוש במשתנים גלובליים משתי סיבות,

1. כדי ליישמה צריך לשנות את אופן הקריאה לפונקציה.
2. אם הפונקציה שלה אנו רוצים לספור את מספר הקריאות היא פונקציה שפונקציות אחרות קוראות לה, אז צריך לשנות גם את הפלט והקלט של כל הפונקציות הקוראות לה.

הערה

- שימוש פשוט בפונקצית clear עם משתנה גלובלי מוחק אותו רק מה-Workspace אך לא מהזיכרון. כדי למחוק משתנה גלובלי כמוNUM_CALL נרשום,

```
clear global NUM_CALL
```

10.4 תת פונקציות

קובץ m. מסוג פונקציה יכול להכיל יותר מפונקציה אחת. הפונקציה הראשונה היא הפונקציה המרכזית והיא הפונקציה שמבוצעת כאשר קוראים בשם הפונקציה. כל פונקציה המוגדרת אחרי הפונקציה המרכזית היא תת פונקציה הנגישה רק לפונקציה המרכזית ולתת פונקציות אחרות של אותה פונקציה מרכזית. לכל תת פונקציה שורת הגדרה משלה. סדר התת פונקציות אינו משנה כל עוד הן מוגדרות לאחר הפונקציה המרכזית.
דוגמא,

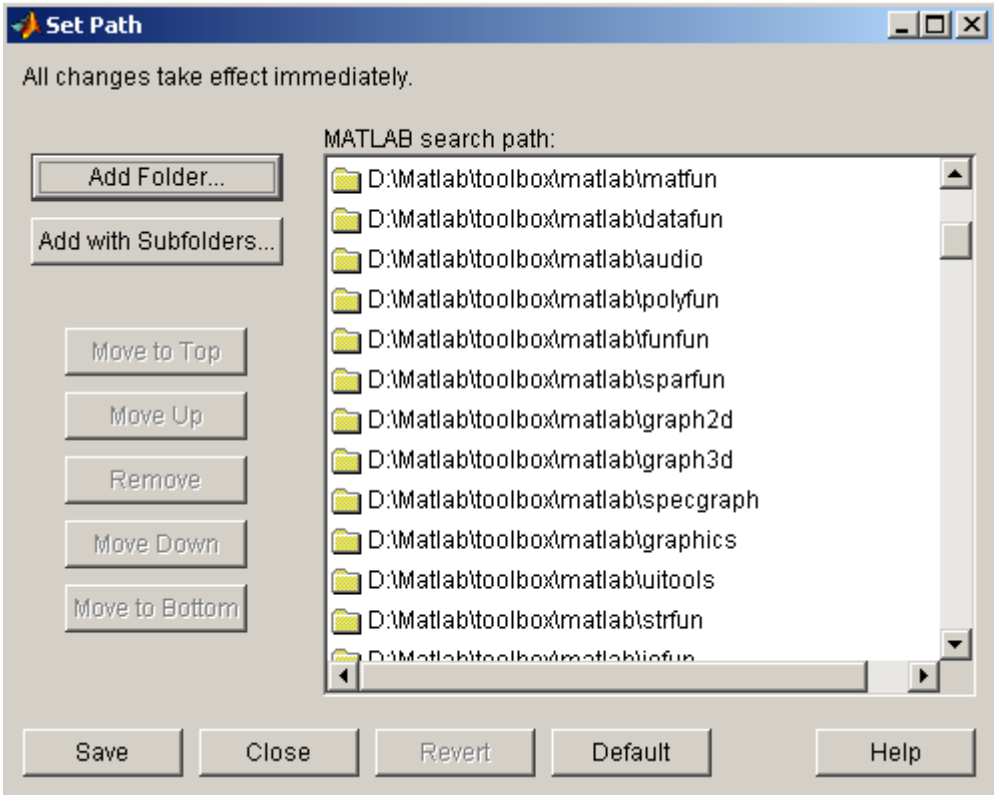
```
function [avg,med] = newstats(u) % Primary function
% NEWSTATS Find mean and median with internal functions.
n = length(u);
avg = mean(u,n);
med = median(u,n);


function a = mean(v,n)              % Subfunction
% Calculate average.
a = sum(v)/n;

function m = median(v,n)            % Subfunction
% Calculate median.
w = sort(v);
if rem(n,2) == 1
    m = w((n+1)/2);
else
    m = (w(n/2)+w(n/2+1))/2;
end
```

הפונקציה newstats מחשבת את הממוצע והמדיאן של וקטור. הפונקציה המרכזית היא newstats והוגדרו שתי תת פונקציות שלה בשם mean, median.
בעת קריאה לפונקציה, Matlab מחפש תחילה אחר תת פונקציות בעלות אותו שם, אחר כך פונקציות פרטיות[42] ולבסוף פונקציות סטנדרטיות ב-search path. לכן בדוגמא הנ"ל כאשר הגדרנו תת פונקציה mean הפונקציה המרכזית newstats תשתמש בתת הפונקציה במקום בפונקציה הסטנדרטית בעלת אותו שם.

[42] להסבר לגבי פונקציות פרטיות חפשו ב-help אחר הנושא private functions.

10.5 יצירת toolbox

כדי ליצור toolbox צריך תחילה ליצור תיקיה המכילה את כל הפונקציות הרצויות. בחלון הראשי בתפריט File לחצו על Set Path... וחלון Set Path יופיע בדומה לגרף 10.3. בחלון זה מוצגים כל התיקיות שבהן Matlab מחפש אחר פונקציות. כדי להוסיף את תיקיית ה-toolbox החדשה שלכם בחרו באופציית Add Folder… ובחרו את התיקייה הרצויה.



גרף 10.3: חלון הגדרת ה-toolbox-ים

יצירת toolbox מאפשרת גישה לפונקציות ב-toolbox מכל נתיב. למשל כדי להשתמש בפונקצית mean הסטנדרטית אין צורך לגשת לתיקיה בה מוגדרת פונקצית mean. בד"כ שמים ב-toolbox פונקציות השימושיות למספר פרויקטים הנמצאים בתיקיות שונות כדי שלא להעתיק אותן לכל תיקיה מחדש.

11. מציאת ותיקון שגיאות (Debugging)

קיימים שני סוגים של שגיאות,

1. שגיאות syntax - למשל שכיחת סוגר, כתיבת שם עם אות גדולה במקום אות קטנה וכו'.
   לעיתים רבות רצוי לבדוק שגיאות כתיב ללא הרצת כל התוכנה כדי לחסוך את זמן ההרצה. ניתן לעשות זאת באמצעות כתיבת pcode ושם קובץ ה-m.
2. שגיאות הרצה (runtime). למשל כתוצאה מחישובים מתקבלים ערכים לא חוקיים.

שגיאות syntax הן שגיאות שקל (יחסית) למצוא ולתקן. לעומת זאת שגיאות runtime הן שגיאות חמקמקות יותר. בנוסף, כאשר מתקבלות שגיאות runtime ההרצה מופסקת וכל המשתנים הלוקליים ב-Workspace נמחקים וכך קשה מאד לנתח את סיבת הבעיה. לכן ניתן להשתמש באופציית ה-Debugging כדי לבצע את הרצת הקוד שורה שורה תוך כדי נגישות למשתנים הלוקליים.

כדי להמחיש את תהליך ה-Debugging נציג גרסה שגויה של פונקצית מציאת האפסים שהוצגה בפרק 10. הרצת קבצים מבוצעת ע"י הקלדת שם הקובץ ב-Command Window או כאשר בחלון Editor/Debugger הפונקציה הרצויה מסומנת לוחצים על F5 או על כפתור Run ().

קובץ ה-script,

```
1 -  clear
2 -  clc
3 -  close all
4
5 -  x_range = [0 20];
6
7 -  params = struct('A',1,'w',pi/4,'phi',pi/2);
8 -  func = @sin_func;
9
10 -  zero_locations = find_zero_wrong(func,x_range,params);
11
12    % graph section
13 -  x_axis = linspace(min(x_range),max(x_range),1000);
14 -  figure
15 -  plot(x_axis,feval(func,x_axis,params))
16 -  xlabel('x'),ylabel('y')
17 -  grid
18
19 -  disp(['The zeros of the function in the range [',num2str(x_range),'] are:'])
20 -  disp(zero_locations)
```

הפונקציה,

```
function zero_locations = find_zeros_wrong(func,x_range,params,dx,tol)
%FIND_ZEROS Finds all the zeros of function 'func' within specified range
%
% INPUT:
%    func - a string of the name of the mathematical function
%           whose zeros the user wants to locate.
%    x_range - a [2 x 1] vector containing the range in which the
%              algorithm will search for zeros.
%    dx - a scalar specifying the initial increment factor.
%    tol - a scalar specifying the degree of tolerance to which the
%          algorithm will search.
%    params - a structure containing the parameters of the function func.
% OUTPUT:
%    zero_locations - a vector containing all the zero locations found in
%                     the specified range.
% Alternative USAGE:
%    find_zeros_varargin(func,x_range,dx,parameters) - tol is set to default le-6
%    find_zeros_varargin(func,x_range,parameters) - tol is set to default le-6 and
%                                                   dx is set to default le-2

% this section ensures that the input is correct and that parameters
% dx,tol have values.
switch nargin
case 4 % tol parameter has not been specified, revert to default value
    tol = le-6;
case 3 % dx, tol parameters have not been specified, revert to default values
    dx = le-2;
    tol = le-6;
otherwise
    error('Incorrect input');
end

% initial values
x = min(x_range); % begin the search from left to right
x_end = max(x_range);
zero_locations  = [];
sign_x = sign(feval(func,x,params));

% main loop
while x<=x_end
    sign_x_plus_dx = sign(feval(func,x + dx,params));

    if sign_x ~= sign_x_plus_dx
        dx = dx/2;
    else
        x = x + dx;
        % the sign is of the function at x + dx is the same as at location x
    end

    if dx <= tol
        % a zero has been found because the interval [x,x+dx] in which the zero
        % must be located is smaller than tol.
        zero_locations = [zero_locations x + dx/2]; %the zero is in the middle of the
        sign_x = sign_x_plus_dx; % the sign has switched
        x = x + dx; % update x
    end
end
```

למראית עין הקוד נראה תקין אך הרצת קובץ ה-script מניבה את התוצאה הבאה,

```
??? Undefined function or variable 'find_zero_wrong'.

Error in ==> D:\Dori\Technion\Matlab Lectures\find_zeros_script_wrong.m
On line 10  ==> zero_locations = find_zero_wrong(func,x_range,params);
```

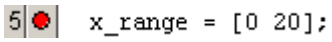
התקבלה הודעת שגיאה.

טריק

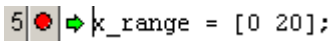
אם נוצרת שגיאה, מופיעה הודעת שגיאה בצבע אדום ב-Command Window. ניתן להגיע ישירות לשורת השגיאה בקובץ הנכון ע"י השמת סמן העכבר על מיקום השגיאה המסומן עם קו תחתון[43] בהודעת השגיאה ולחיצה על הכפתור השמאלי של העכבר.

נשים את הסמן על מיקום ההודעה ונלחץ על הכפתור השמאלי של העכבר ונגיע לשורה שבה התחוללה השגיאה. הטעות במקרה זה הייתה ששם הפונקציה היא find_zeros_wrong ולא find_zero_wrong.
לאחר תיקון הטעות נריץ את קובץ ה-script והתוכנית לא תסיים בזמן סביר. כדי לעצור את הרצה נלחץ על Ctrl+C או Ctrl+Break בחלון Command Window.
לעומת באג ה-syntax הקודם, קשה מאד מהסתכלות בקוד בלבד לראות מהי הבעיה, או אפילו באיזה קובץ הבעיה.
השלב הראשון בדיבוג הוא קביעת נקודות עצירה (breakpoints) שבהן הרצת הפונקציה תיפסק. נשים לב כי משמאל לקוד קיימות שתי עמודות. השמאלית ביותר מציגה את מספר השורה, ומימינה קיימת עמודה שבחלק משורותיה מופיעים קווים -. לחיצה על הכפתור השמאלי של העכבר על הקווים הנ"ל גורם להופעת עיגול אדום המציין נקודת עצירה[44]. השורות בהן אי אפשר לקבוע נקודת עצירה הן שורות ריקות או שורות המכילות הערות.

נציב נקודת עצירה בשורה 5,



נריץ שוב את קובץ ה-script וכעת ההרצה תיעצר בשורה 5,



בחלון ה-Workspace כעת מופיע הסימן K>> כדי לציין כי Matlab הוא במצב של Debug.

החץ הירוק המלא מציין כי כעת ההרצה נמצאת בשורה הנוכחית אך לא ביצעה אותה עדיין. כדי לבצע את השורה המסומנת ולעבור לשורה הבאה נלחץ על כפתור Step  .
לחיצה על Step מבצעת את שורה 5 ומקדמת את החץ הירוק המלא לשורה 6. ב-Workspace כעת מופיע המשתנה x_range.
עד לשורה 10 רק מבוצעת הגדרת משתנים ולכן מיותר לעבור על כל השורות הנ"ל שורה שורה. נציב נקודת עצירה בשורה 10 ונלחץ על כפתור Continue[45]  . כפתור זה מבצע את כל השורות עד לנקודת העצירה הבאה או עד סיום כל ההרצה במקרה ואין עוד נקודת עצירה.
נשים לב כי כעת ניתן לבחון ואף לשנות את ערכם של כל המשתנים שהוגדרו ולוודא שהם תקינים.
ניתן לבחון את המשתנים ב-Command Window, באמצעות חלון Array Editor או ע"י הזזת העכבר על המשתנים בחלון Editor/Debugger וכך יצוצו חלונות הנקראים data tips המציינים מידע לגבי המשתנה.

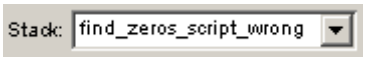
---

[43] צורת הרישום הזו דומה ל-hyperlink באינטרנט.
[44] לחיצה על העיגול האדום מורידה את נקודת העצירה.
[45] נשים לב כי במצב Debug כפתור Run הופך לכפתור Continue.

לדוגמא,

```
 7 -   params = struct('A',1,'w',pi/4,'phi',pi/2);
 8 -   func =
 9
10 ●  zero_l      w: 0.7854 zeros_wrong(func,x_range,params);
11          phi: 1.5708
12    % graph section
```

עד עתה לא נצפו דברים בלתי צפויים ולכן רוב הסיכויים שהבעיה נמצאת בתוך פונקצית find_zeros_wrong. לחיצה על כפתור Step לא תאפשר לנו לבחון את תוכן הפונקציה הנ"ל ולכן נשתמש בכפתור Step in  . כפתור זה נכנס לתוך פונקציה פנימית בשורה של החץ הירוק המלא.
נשים לב כי חלון Stack מציין בתוך איזה פונקציה ההרצה נמצאת כרגע. לפני לחיצה על כפתור Step in חלון Stack הציג,



לאחר לחיצה על כפתור Step in נעבור לשורה הראשונה[46] של פונקצית find_zeros_wrong וחלון Stack מציין זאת. נשים לב כי בחלון של קובץ find_zeros_script_wrong החץ הפך לריק,

```
10 ●  zero_locations = find_zeros_wrong(func,x_range,params);
```

המשמעות של סימון זה הוא שכעת ההרצה נמצאת בתוך הפונקציה הפנימית שקראו לה בשורה המסומנת.

נבצע מספר שורות עד לשורה 40. נשים לב כי נוספו מספר משתנים. משתנים אלו הם משתנים לוקליים של פונקצית find_zeros ואינם מוגדרים בקובץ ה-script שקרא לפונקציה.
כדי לבחון את הערכים המוגדרים בקובץ ה-script ניתן להשתמש בחלון stack ואכן שני המשתנים החדשים dx, tol לא מוגדרים בקובץ ה-script. נשים לב כי שלושת המשתנים func, params, x_range מוגדרים בשני הקבצים אך בפונקציה find_zeros_wrong המשתנים הנ"ל הם משתנים לוקליים. כלומר הם רק העתקים של המשתנים בקובץ ה-script ולכן שינויים בתוך הפונקציה לא ישנו את ערכם בקובץ ה-script.
מעבר על הפקודות עד למציאת האפס הראשון אינן מעלות בעיות הנראות לעין. כדי לחסוך בזמן נציב נקודת עצירה בשורה 53 (נוריד את נקודת העצירה בשורה 40). נשים לב כי כעת ערך x נשאר על ערך 2 ולכן אין התקדמות. למעשה, קיימת התקדמות אך היא קטנה מאד. ניתן להבחין בה ע"י פתיחת המשתנה x בחלון Array Editor ובחירת יצוג longG.
הסיבה לבאג היא שלאחר שצומצם ערך הקידום dx צריך להחזירו לערכו המקורי 0.01 כי כעת הוא קטן מ-10^{-6} ולכן ערך x כמעט ואינו מתקדם בכל איטרציה. דרך פשוטה לפתור את הבעיה היא להגדיר את הערך המקורי של dx כ-dx_original ולהציבו ב-dx כאשר נמצא אפס.
כדי לבצע את השינויים צריך לצאת מ-Debug Mode ע"י לחיצה על כפתור Exit Debug Mode  .
כדי להוריד את כל נקודות העצירה נלחץ על כפתור Clear all breakpoints  .
שני השינויים הם,

```
33    % initial values
34 -  dx_original = dx;
35 -  x = min(x_range); % begin the search from left to right
56 -          x = x + dx; % update x
57 -          dx = dx_original;
58 -      end
```

[46] השורה הראשונה לא כולל שורות ריקות או הערות.

נשים לב כי עלינו לשמור את השינויים שביצענו בפונקציה לפני שנריץ את קובץ ה-script מחדש.
נוסיף נקודת עצירה בשורה 54 ולאחר הרצה נמצא שהאפס הראשון הוא 2 וכעת האלגוריתם מקדם את x באופן הרצוי. כדי לסיים את הרצת הפונקציה נוריד את נקודת העצירה ונלחץ על כפתור Step out שתפקידו הוא לסיים הרצת הפונקציה הפנימית שנכנסנו אליה דרך כפתור Step in. כדי לסיים את הרצת הפונקציה כולה נלחץ על כפתור Continue ואז יתקבל גרף 10.2 והתוצאות המתאימות.

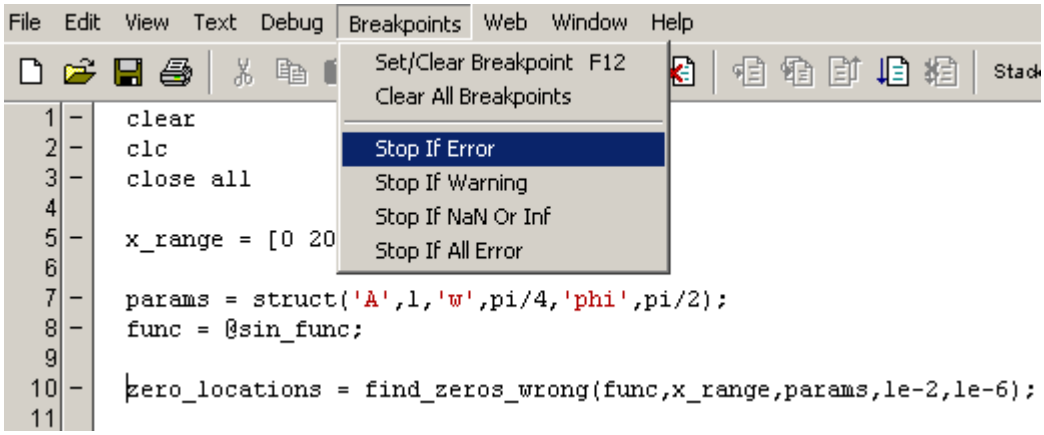
תמיד רצוי לבדוק מספר ערכים שונים של משתני קלט כדי לוודא כי האלגוריתם פועל כראוי.
הכל נראה תקין אך עדין נותר עוד באג אחד בתוכנה. מה יקרה אם בעת הקריאה לפונקציה find_zeros_wrong בקובץ ה-script נכניס גם את ערכי ברירת המחדל של dx ו-tol ,

```
zero_locations = find_zeros_wrong(func,x_range,params,1e-2,1e-6);
```

מתקבלת השגיאה,

```
Incorrect input

Error in ==> D:\Dori\Technion\Matlab_Lectures\find_zeros_script_wrong.m
On line 10  ==> zero_locations = find_zeros_wrong(func,x_range,params,1e-2,1e-6);
```

לחיצה על מיקום השגיאה מביא את הסמן לשורה 10 של קובץ ה-script אך לא ברור מדוע נוצרה השגיאה. לכן אופציה מאד שימושית היא להריץ את התוכנה עד לשורה שבה מתבצעת השגיאה ללא ביצועה. כך נוכל לראות בדיוק איפה התבצעה השגיאה ואת כל ערכי המשתנים רגע לפני שמתחוללת השגיאה.
בתפריט Breakpoints נלחץ על אפשרות Stop if Error באופן הבא,



הרצת התוכנה כעת מובילה אותנו לשורה,

```
30 -      error('Incorrect input');
```

כלומר פונקצית switch הביאה אותנו להודעת שגיאת קלט למרות שתכננו שקלט של 5 משתנים יהיה קלט תקין. פתרון הבעיה הוא ע"י הוספת אופציה נוספת לפונקצית switch,

```
switch nargin
case 5
    % do nothing
case 4 % tol parameter has not been specified, revert to default value
    tol = 1e-6;
case 3 % dx, tol parameters have not been specified, revert to default values
    dx = 1e-2;
    tol = 1e-6;
otherwise
    error('Incorrect input');
end
```

כעת הקוד מתפקד באופן תקין.

12 שיפור ביצועים ב-Matlab

קיימות מספר דרכים לשפר את הביצועים של קוד Matlab,

1. שימוש בפונקציות מטריציות במקום לולאות for ו-while ככל הניתן. למעשה קשה להחליף לולאות while בפעולות מטריציות ולעומת זאת במקרים רבים ניתן להחליף לולאות for בפעולות מטריציות.
2. עדיף לאתחל משתנים שגודלם ידוע מראש. הסיבה לכך היא שכאשר משתנה איננו מאותחל אז Matlab צריך לבזבז משאבים בשינוי גודלו ושינוי הקצאת הזיכרון בשבילו. עבור מערך נומרי נשתמש בפונקצית zeros, עבור מערך תאים בפונקצית cell ועבור מערך structure בפונקציה struct.
3. הימנעות מיצירת משתנים זמניים גדולים ומחיקת משתנים לא נחוצים באמצעות פונקצית clear.

הערה,
כמות הזיכרון והמשאבים הדרושים לחישוב שני המקרים הבאים הוא זהה,

1.
```
y = function2(function1(x));
```

2.
```
y = function1(x);
y = function2(y);
```

Profiler 12.1

שיטת ה-Profiling היא שיטה למדידת היכן התוכנה מבלה את זמנה. מטרת ה-Profiler היא מציאת צווארי הבקבוק של תוכנות כדי לאפשר ביצועים מהירים יותר. ה-Profiler מראה איזה פונקציות גוזלות את זמן ההרצה והמשתמש יכול לנסות למצוא דרכים לעקוף פונקציות אלו.
לקוד בד"כ יש מספר שכבות ולכן תוכנות שהן על פניהן כתובות בצורה יעילה יתכן והן קוראות לפונקציות בשכבה יותר נמוכה שמבזבזות משאבים.
כדי ללמוד יותר על הכלי השימושי הזה כתבו doc profile. למען המחשה, להלן גרף התוצאות של הרצת התוכנה למציאת אפסים.
הקוד,

```
profile on -detail builtin
find_zeros_script
profile report name
```

מניב את החלון בגרף 12.1.

הערה

ניתן גם למדוד זמן הרצת קוד שהוא לא בהכרח פונקציה באמצעות הפונקציה cputime, וצמד הפונקציות tic ו-toc.

Summary | Function Details

MATLAB Profile Report: Summary

Report generated 09-Nov-2002 09:12:37

Total recorded time:	1.60 s
Number of Builtin-functions:	51
Number of M-functions:	21
Number of M-scripts:	1
Number of M-subfunctions:	6
Clock precision:	0.010 s

Function List

Name	Time		Calls	Time/call	Self time		Location
find_zeros_script	1.602	100.0%	1	1.602000	0.060	3.7%	D:\Dori\Tec
figure	0.821	51.2%	1	0.821000	0.821	51.2%	*Builtin-function*
plot	0.561	35.0%	1	0.561000	0.050	3.1%	*Builtin-function*
newplot	0.511	31.9%	1	0.511000	0.000	0.0%	D:\Matlab\t
newplot/ObserveAxesNextPlot	0.311	19.4%	1	0.311000	0.020	1.2%	D:\Matlab\t
clo	0.291	18.2%	1	0.291000	0.000	0.0%	D:\Matlab\t
allchild	0.261	16.3%	1	0.261000	0.000	0.0%	D:\Matlab\t
get	0.261	16.3%	31	0.008419	0.261	16.3%	*Builtin-function*
gca	0.200	12.5%	4	0.050000	0.150	9.4%	D:\Matlab\t

גרף 12.1: חלון profiler

13. פעולות גרפיות מתקדמות

13.1 יצירת סרטים

במקרים רבים גרפים דו-ממדיים ותלת-ממדיים אינם מצליחים להמחיש דינמיקה של תהליכים. לשם כך ניתן לייצר סרטים מגרפים ע"י הרצת הגרפים אחד אחר השני בצורה חלקה.
קיימות שתי דרכים ליצירת סרטים ב-Matlab,

1. אופציית Erase Mode: אובייקטים בגרף משורטטים ונמחקים on-line. שיטה זו יעילה כאשר השינויים בין הפריימים הוא קטן.
2. פונקצית movie: יצירת מספר גרפים ושמירתם יחד כסרט off-line. שיטה זו יוצרת סרטים יותר חלקים ולכן נתמקד בה.

הרעיון של פונקצית movie פשוט ביותר: יצרו גרף, שימרו אותו באמצעות פונקצית getframe לתוך משתנה סרט מסוג structure, והציגו את הסרט באמצעות פונקצית movie.
לדוגמא נבצע מספר תוספות לאלגוריתם מציאת אפסים (התוספות מסומנות בצבע כתום) כדי לבנות סרט הממחיש את התקדמות האלגוריתם,
קובץ ה-script,

```
clear all
clc
close all

x_range = [0 20];

params = struct('A',1,'w',pi/4,'phi',pi/2);
func = @sin_func;

[zero_locations,M] = find_zeros_movie(func,x_range,params,0.1,0.01);

% graph section
movie2avi(M,'f_zeros_movie','compression','none') % convert Matlab movie into avi movie
movie(M,1) % run movie one time
```

הפונקציה,

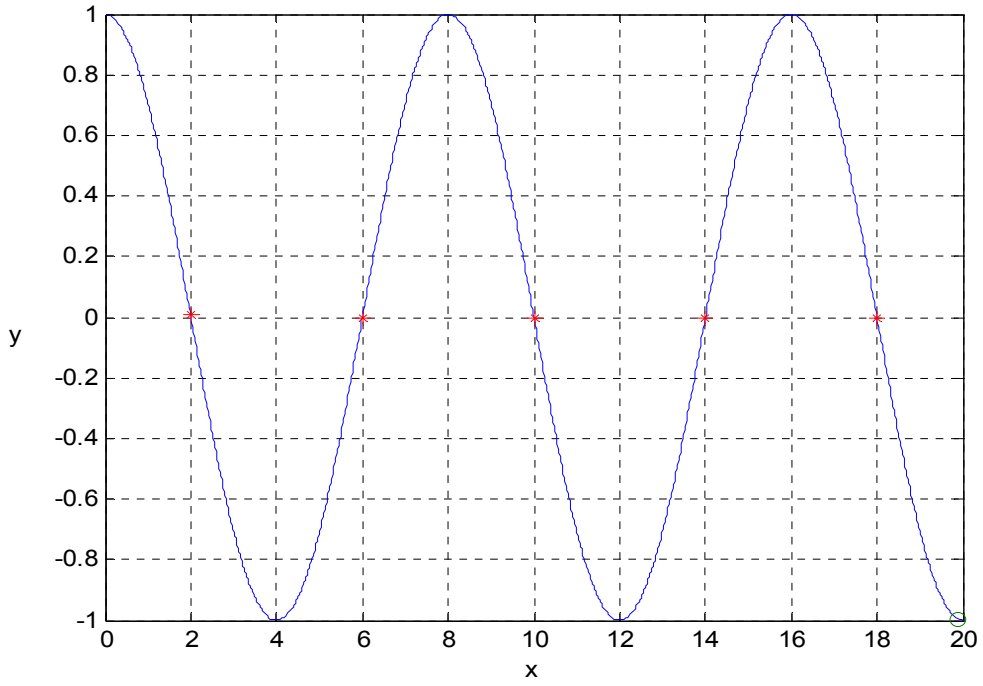
```
function [zero_locations,M] = find_zeros_movie(func,x_range,params,dx,tol)
%FIND_ZEROS Finds all the zeros of function 'func' within specified range
%
% INPUT:
%    func - a string of the name of the mathematical function
%           whose zeros the user wants to locate.
%    x_range - a [2 x 1] vector containing the range in which the
%              algorithm will search for zeros.
%    dx - a scalar specifying the initial increment factor.
%    tol - a scalar specifying the degree of tolerance to which the
%          algorithm will search.
%    params - a structure containing the parameters of the function func.
% OUTPUT:
%    zero_locations - a vector containing all the zero locations found in
%                     the specified range.
%    M - movie detailing the progress of the algorithm
% Alternative USAGE:
%    find_zeros_varargin(func,x_range,dx,parameters) - tol is set to default 1e-6
%    find_zeros_varargin(func,x_range,parameters) - tol is set to default 1e-6 and
%                                                   dx is set to default 1e-2
```

```
% this section ensures that the input is correct and that parameters
% dx,tol have values.
switch nargin
case 5
    % do nothing
case 4 % tol parameter has not been specified, revert to default value
    tol = 1e-6;
case 3 % dx, tol parameters have not been specified, revert to default values
    dx = 1e-2;
    tol = 1e-6;
otherwise
    error('Incorrect input');
end

% initial values
dx_original = dx;
x = min(x_range); % begin the search from left to right
x_end = max(x_range);
zero_locations  = [];
sign_x = sign(feval(func,x,params));

% movie varaibles
i = 1;
x_axis = linspace(min(x_range),max(x_range),1000);

% main loop
while x<=x_end

    % movie update
    plot(x_axis,feval(func,x_axis,params),x,feval(func,x,params),'o');
    xlabel('x'),ylabel('y')
    grid

    sign_x_plus_dx = sign(feval(func,x + dx,params));

    if sign_x ~= sign_x_plus_dx
        dx = dx/2;
    else
        x = x + dx;
        % the sign is of the function at x + dx is the same as at location x
    end

    if dx <= tol
        % a zero has been found because the interval [x,x+dx] in which the zero
        % must be located is smaller than tol.
        zero_locations = [zero_locations x + dx/2]; %the zero is in the middle of the
        sign_x = sign_x_plus_dx; % the sign has switched
        x = x + dx; % update x
        dx = dx_original;
    end

    % movie update
    if isempty(zero_locations)~=1
        hold on
        plot(zero_locations,feval(func,zero_locations,params),'r*');
        hold off
    end
    M(i) = getframe;
    i = i + 1;

end
```

M הוא מערך structure המכיל את כל הפריימים של הסרט. במהלך כל איטרציה של לולאת while משורטט גרף. בסיום כל איטרציה של הלולאה, הגרף מוצב ב-M באמצעות פונקצית getframe. לאחר סיום הרצת הפונקציה, היא מחזירה את M לקובץ ה-script. קובץ ה-script ממיר את סרט ה-Matlab M לפורמט avi הניתן לצפייה מחוץ ל-Matlab. בנוסף הקוד,

```
movie(M,1) % run movie one time
```

מריץ את הסרט M פעם אחת[47].



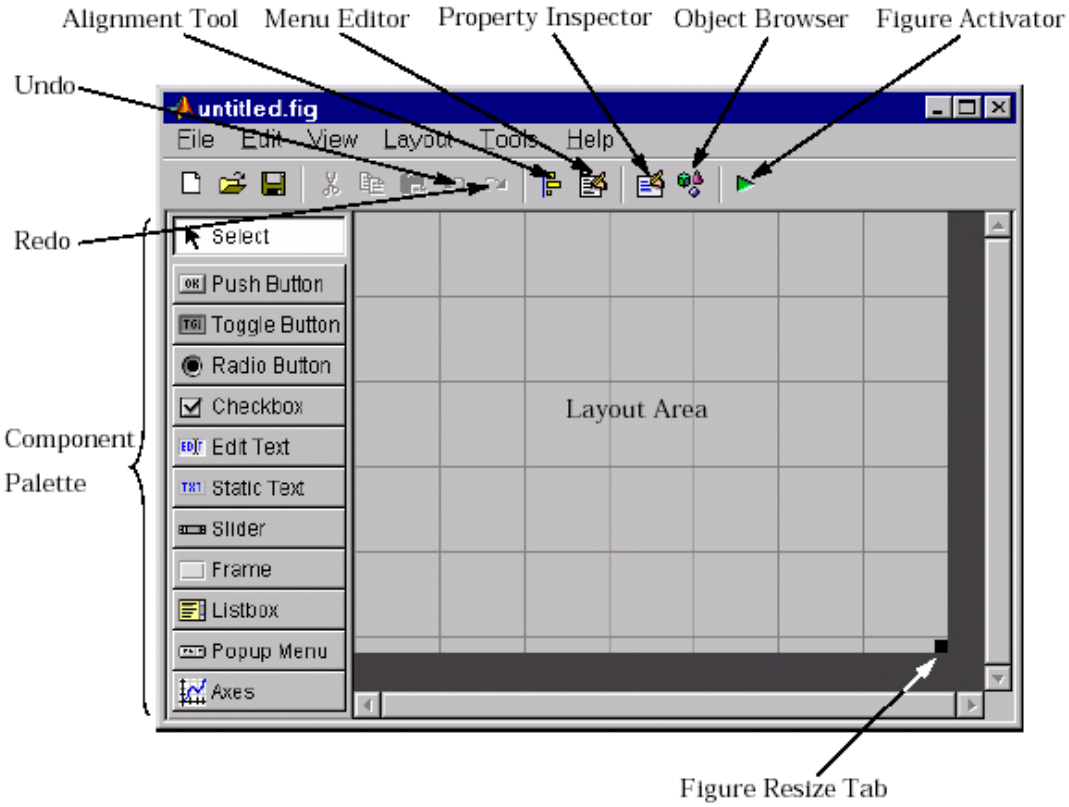
גרף 13.1: הפריים האחרון של הסרט

[47] הארגומנט השני קובע את מספר הפעמים שהסרט יוצג ברצף.

GUI (Graphical User Interface) 13.2

פרק זה סוקר רק בקצרה את אפשריות יצירת ממשק גרפי. להיכרות יותר מעמיקה קראו את buildgui.pdf שעל פיו נכתב פרק זה.

קיימים ב-Matlab ממשקים גרפיים במרבית ה-toolbox-ים. תכנון ממשק גרפי מבוצע באמצעות פונקציית guide הפותחת את חלון תכנון ממשק הגרפי[48] המתואר בגרף 13.2.



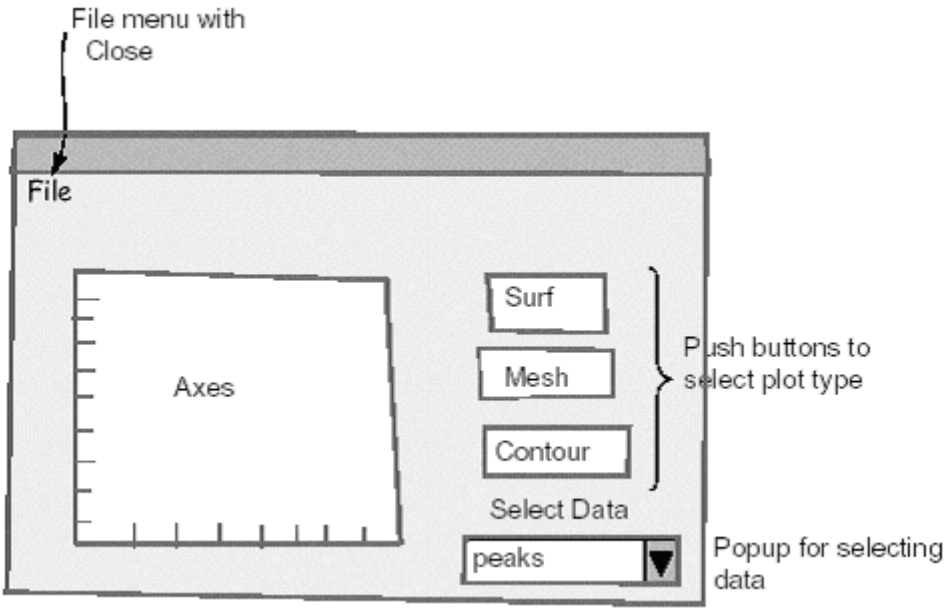
גרף 13.2: חלון תכנון ממשק גרפי (Layout Editor)

תהליך יצירת GUI מתחלק לשלושה שלבים,

1. תכנון ה-GUI- מומלץ לתכנן מראש כיצד הממשק הגרפי יראה.
2. יצירת גרף GUI- יצירת המעטה החיצוני של הממשק הגרפי באמצעות ה-Layout Editor.
3. תכנון ה-GUI- בעת שמירת גרף ה-GUI מיוצר קובץ m. שדרכו ניתן לתכנן את התפקוד של כל אובייקט (כפתורים, תפריטים, גרפים).

בפרק זה נמחיש את העקרונות הבסיסיים של יצירת GUI באמצעות דוגמא,
שרטוט הפונקציות התלת-ממדיות peaks, membrane, sinc באמצעות הפונקציות ליצירת גרפים surf, mesh, contour. גרף 13.3 הוא תרשים סכמתי של ממשק הגרפיקה הרצוי.

[48] כדי לקבל חלון זה צריך בתפריט File לבחור ב-Preferences באפשרות Show names in component palette.

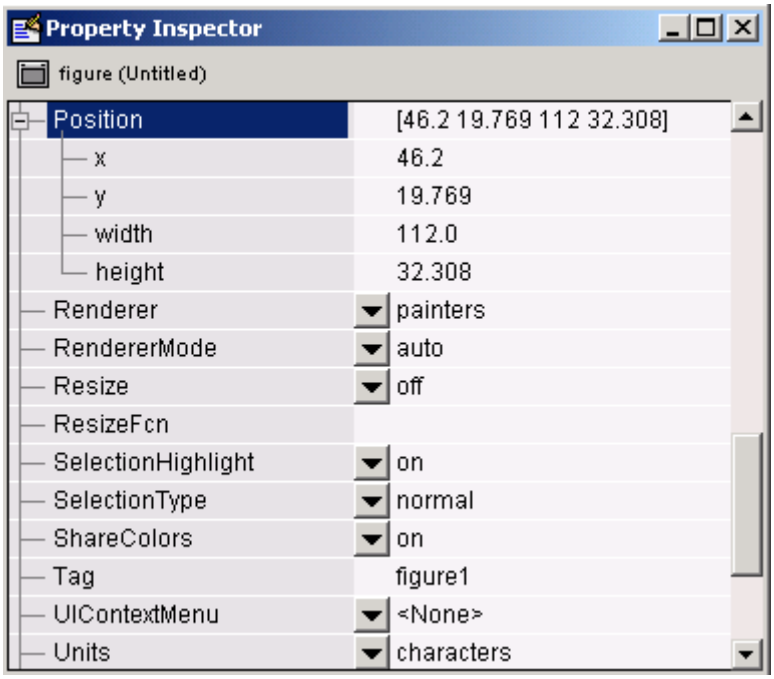


גרף 13.3: שרטוט סכמתי של הממשק הרצוי

הרצת guide ב-Command Window פותחת את החלון בגרף 13.2. מה-Component Palette ניתן להציב אובייקטים בשטח המסומן בגרף 13.2 כ-"Layout Area" שהוא יהווה את חלון הממשק הגרפי.

13.2.1 קביעת גודל החלון

קיימות שתי אפשריות לקביעת גודל החלון,
1. ידנית- לחיצה וגרירה על הריבוע השחור בפינה הימנית תחתונה של ה-Layout Area.
2. Property Inspector- לחיצה על כפתור Property Inspector פותחת את החלון בגרף 13.4.



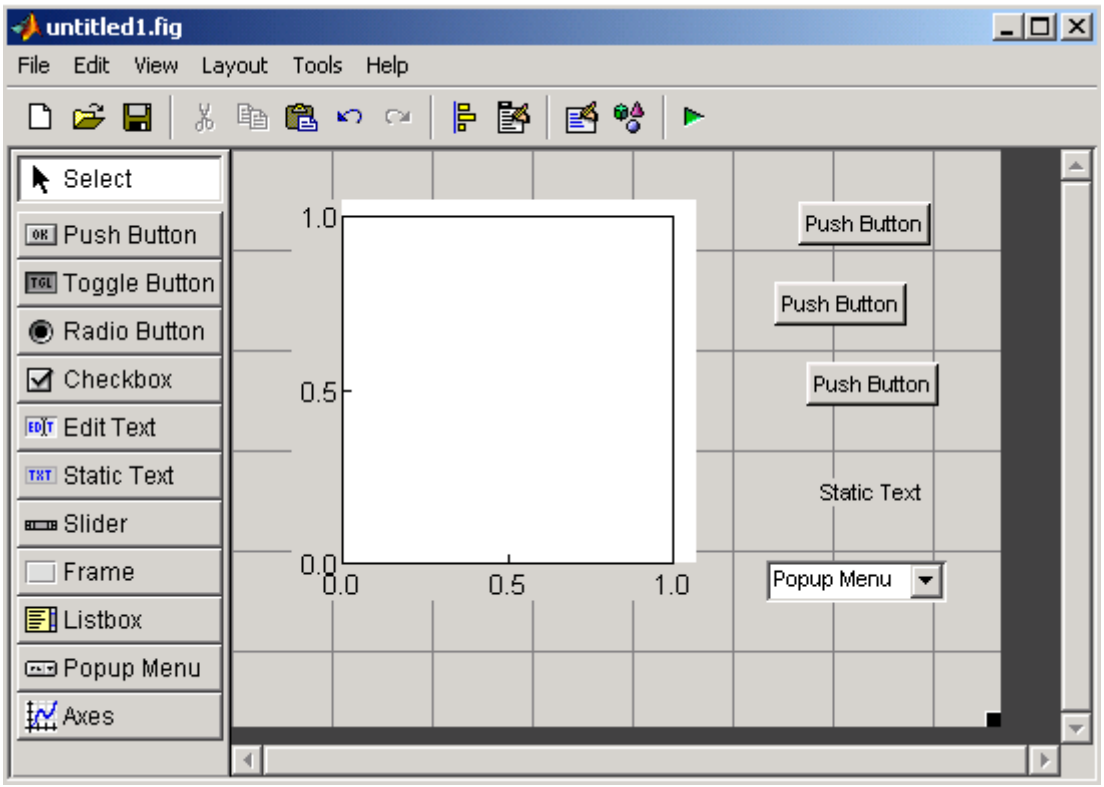
גרף 13.4: חלון Property Inspector

בחלון Property Inspector ניתן לשנות את התכונות של כל אובייקט. ניתן לשנות את רוחב החלון עצמו ע"י שינוי תת תכונות width ו-height בתכונת position. כדי לקבוע את היחידות הרצויות בוחרים בתכונת Units.

נשים לב כי (כפי שהוזכר בפרק 7.2 אובייקטים של Handle Graphics) ניתן לקבוע תכונות של אובייקט גרפיקה באמצעות פונקציות set ו-get. למעשה, חלון Property Inspector הוא ממשק גרפי המציג את כל התכונות של אובייקט גרפיקה באמצעות פונקצית get ומשנה תכונות אלו באמצעות פונקצית set.

13.2.2 הוספת רכיבים

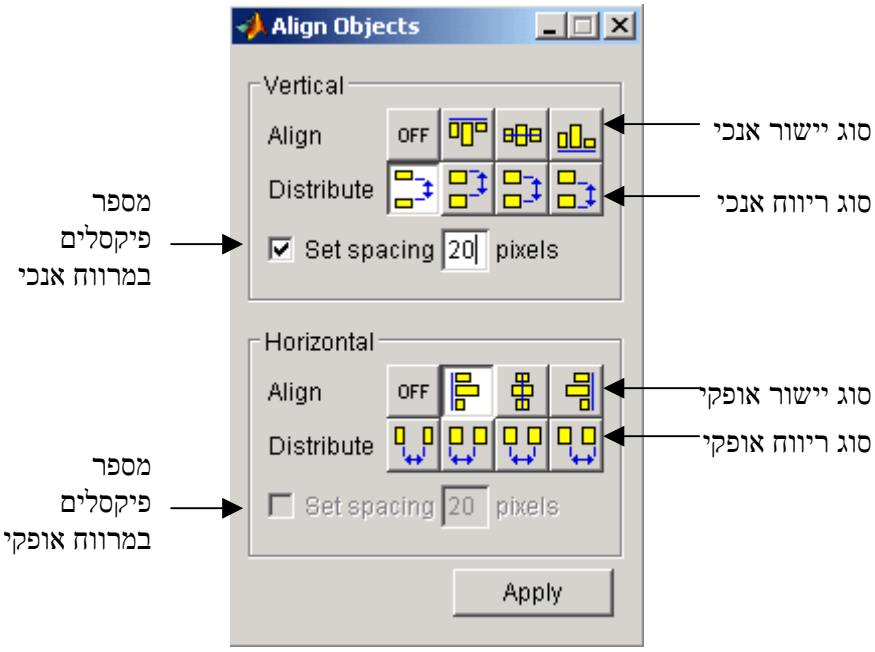
הוספת רכיבים מבוצעת ע"י בחירת רכיב מה- Component Palette וגרירתו למיקום הרצוי ב- Layout Area. בדוגמא שלנו נציב ב- Layout Area שלושה כפתורי לחיצה (push buttons), טקסט סטטי, תפריט נפתח (popup menu), ומערכת צירים אחת.
התוצאה מוצגת בגרף 13.5.



גרף 13.5: GUI נוכחי

13.2.3 יישור אובייקטים

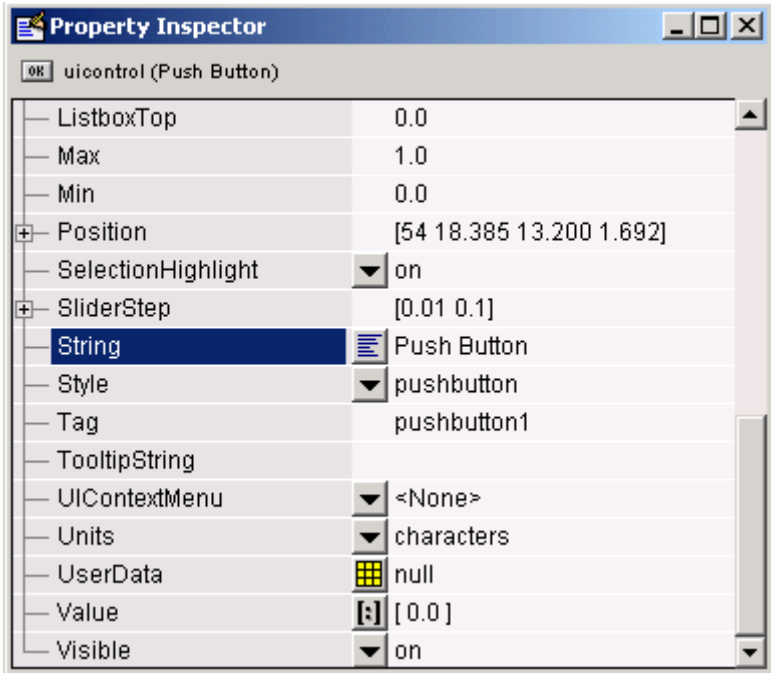
יישור אובייקטים מבוצע ע"י בחירת קבוצת האובייקטים ולחיצה על כפתור Align Objects הפותח חלון Align Objects כמו בגרף 13.6. בחלון זה ניתן לשלוט ביישור אנכי ואופקי. ניתן לקבוע את המרווח האופקי והאנכי בין כל אובייקט וגם את סוג היישור (לימין, שמאל, מרכז או ללא יישור).
לדוגמא, בגרף 13.6 קבענו כי המרווח האנכי מורכב מ-20 פיקסלים והיישור הרוחבי יהיה לשמאל.



גרף 13.6: חלון Align Objects

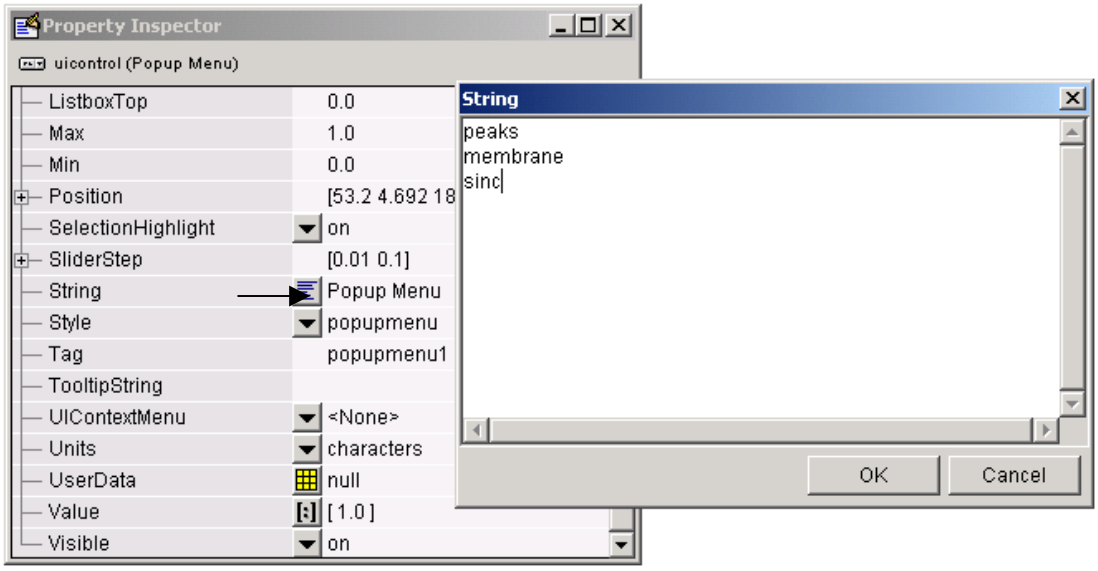
13.2.4 קביעת תכונות של אובייקטים

בוחרים באובייקט, לוחצים על הכפתור הימני של העכבר ובוחרים ב-Property Inspector. למשל, כדי לשנות את התכונות של ה-Push Button העליון ביותר, נבצע את הפעולות הנ"ל ויפתח חלון Property Inspector כפי שמוצג בגרף 13.7.



גרף 13.7: חלון Property Inspector של ה-Push Button העליון

לפי הסכמה בגרף 13.3 אנו רוצים לשנות את שם הכפתור ל-Surf. בתכונה String נשנה את השם ל-Surf במקום Push Button. באותו אופן[49] נשנה את שני כפתורי Push Button הנותרים ואת הטקסט הסטטי בהתאם לסכמה בגרף 13.3.
כדי לקבוע את התפריט נפתח (Popup Menu) נלחץ על הלחצן ליד תכונת String (מסומן בחץ בגרף 13.8) וחלון String יופיע. כל אלמנט בתפריט הנפתח יצוין בשורה נפרדת.



גרף 13.8: חלון Property Inspector של ה-Push Button העליון

לכל אובייקט ב-GUI קיימת תווית (Tag) המציינת אותו. לקראת השלב הבא, שבו נתכנן את התפקוד של כל אובייקט, רצוי לקבוע תווית קלות לזיהוי.
למשל, התווית של אובייקט כפתור הלחיצה Surf נקבעת באופן אוטומטי כ-pushbutton1 וה-Callback הוא <automatic>. לאחר שמירת הגרף, מיוצר קובץ m. שקורא לאובייקטים בחלון GUI באמצעות תת פונקציה Callback ששמה נקבע באמצעות המחרוזת בתכונת Tag.
בדוגמא נשנה את התוויות לפי טבלה 13.1.

אובייקט	תווית ברירת מחדל	תווית חדשה
Surf (Push Button)	pushbutton1	surf_ pushbutton
Mesh (Push Button)	pushbutton2	mesh_ pushbutton
Contour (Push Button)	pushbutton3	contour_ pushbutton
Popup menu	popupmenu1	data_popup

טבלה 13.1

תכונה נוספת שרצוי לשנות היא השם של ה-GUI. נסמן את כל הגרף ונשנה את תכונת name ל-simple_gui.

כדי לראות את ה-GUI בפעולה נלחץ על כפתור Activate Figure. בשמירה הראשונה נוצר גם קובץ m. הקובע את התפקוד של כל אובייקט.

[49] נשים לב כי כדי שהשינוי ישמר יש להוריד את הסימון מהתכונה האחרונה ששונתה.

```
function varargout = simple_gui(varargin)
% SIMPLE_GUI Application M-file for simple_gui.fig
%    FIG = SIMPLE_GUI launch simple_gui GUI.
%    SIMPLE_GUI('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 26-Nov-2002 17:26:16

if nargin == 0  % LAUNCH GUI

    fig = openfig(mfilename,'reuse');

    % Use system color scheme for figure:
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

    % Generate a structure of handles to pass to callbacks, and store it.
    handles = guihandles(fig);
    guidata(fig, handles);
```

אתחול

```
    if nargout > 0
        varargout{1} = fig;
    end
```

אתחול ופתיחת ה-GUI

```
elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try
        if (nargout)
            [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
        else
            feval(varargin{:}); % FEVAL switchyard
        end
    catch
        disp(lasterr);
    end

end


%| ABOUT CALLBACKS:
%| GUIDE generates an empty callback function when you
%| closing parenthesis.

% --------------------------------------------------------------------
function varargout = surf_pushbutton_Callback(h, eventdata, handles, varargin)



% --------------------------------------------------------------------
function varargout = mesh_pushbutton_Callback(h, eventdata, handles, varargin)



% --------------------------------------------------------------------
function varargout = contour_pushbutton_Callback(h, eventdata, handles, vararg



% --------------------------------------------------------------------
function varargout = data_popup_Callback(h, eventdata, handles, varargin)
```

שמות תת-פונקציות ה-Callback נקבעו על סמך התוויות

תת פונקציות הקובעות את תפקוד האובייקטים של ה-GUI

13.2.5 תכנות GUI

תת פונקציות Callback מבוצעות כאשר האובייקט המתאים מופעל ע"י המשתמש. לאחר שמירת הגרף תת פונקציות אלו ריקות ויש לתכנת אותן.
אלמנט חשוב ב-GUI הוא מערך structure בשם handles. למערך זה שתי מטרות,

1. שמירת כל ה-handle-ים של כל האובייקטים ב-GUI. למשל ה-handle של אובייקט data_popup הוא handles.data_popup.
2. שמירת מידע גלובלי. ניתן לשמור מידע בתת פונקציה כך שמידע זה יהיה זמין בעת ביצוע תת פונקציות אחרות. למשל הקוד,

```
handles.x_data = X;
guidata(h,handles)
```

יוצר שדה חדש במערך handles בעל ערך X ושורת הקוד השניה למעשה שומרת את הגרסה החדשה של handles. כדי לגשת ל-X ב-Callback אחר נרשום פשוט,

```
X = handles.x_data;
```

נתכנן כעת את חלון ה-popup,

```
% --------------------------------------------------------------------
function varargout = data_popup_Callback(h, eventdata, handles, varargin)

val = get(h,'Value');
switch val
case 1 % User selected peaks
    handles.data = peaks(35);
case 2 % User selected membrane
    handles.data = membrane;
case 3 % User selected sinc
    [x,y] = meshgrid(-8:.5:8);
    r = sqrt(x.^2+y.^2) + eps;
    z = sin(r)./r;
    handles.data = z;
end
guidata(h,handles) % Save the handles structure after adding data
```

הערך 'Value' מכיל את האפשרות שהמשתמש בחר בחלון ה-popup. לכן, בהתאם לערך זה נייצר data מתאים, נציב ונשמור ב-handles.

נתכנן כעת את לחצני הכפתורים,
כל Push button אמור לייצר גרף מסוג שונה מהמידע הנקבע ע"י חלון popup. לכן ה-Callback-ים שלהם לוקחים את המידע מ-handles ומשרטטים את הגרף המתאים.

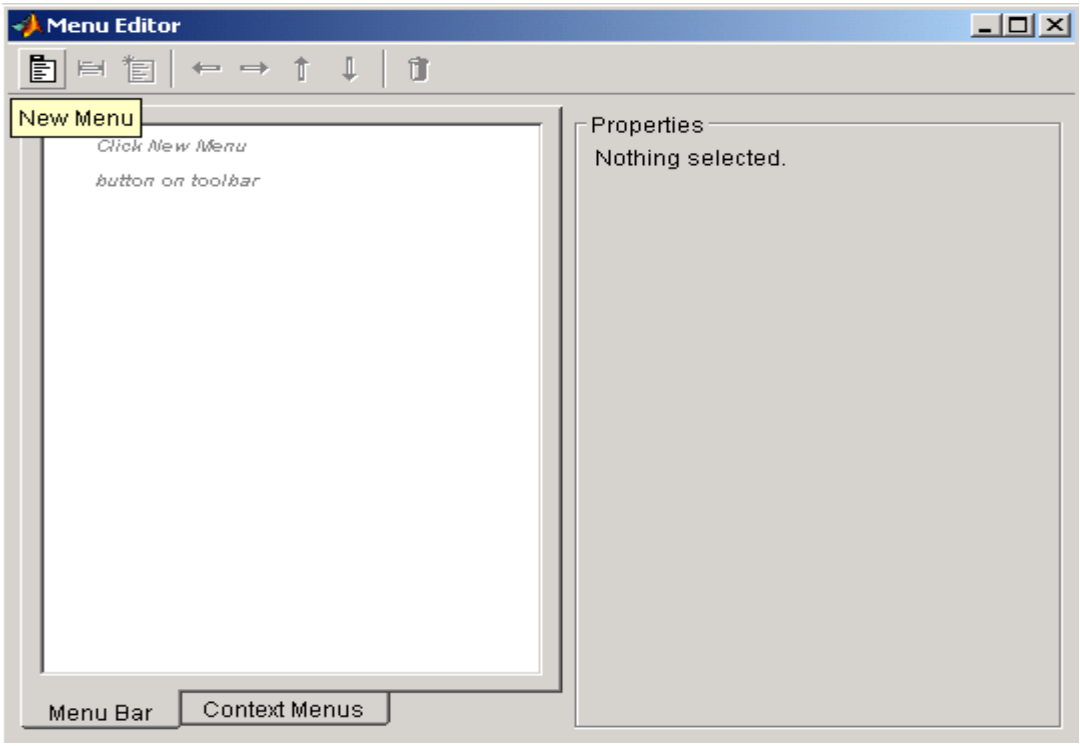
הקוד,

```
function varargout = surf_pushbutton_Callback(h,eventdata,handles,varargin)
z = handles.data; % Load data from handles structure
surf(z);

function varargout = mesh_pushbutton_Callback(h,eventdata,handles,varargin)
z = handles.data; % Load data from handles structure
mesh(z)
```

```
function varargout = contour_pushbutton_Callback(h,eventdata,handles,varargin)
z = handles.data; % Load data from handles structure
contour(z)
```

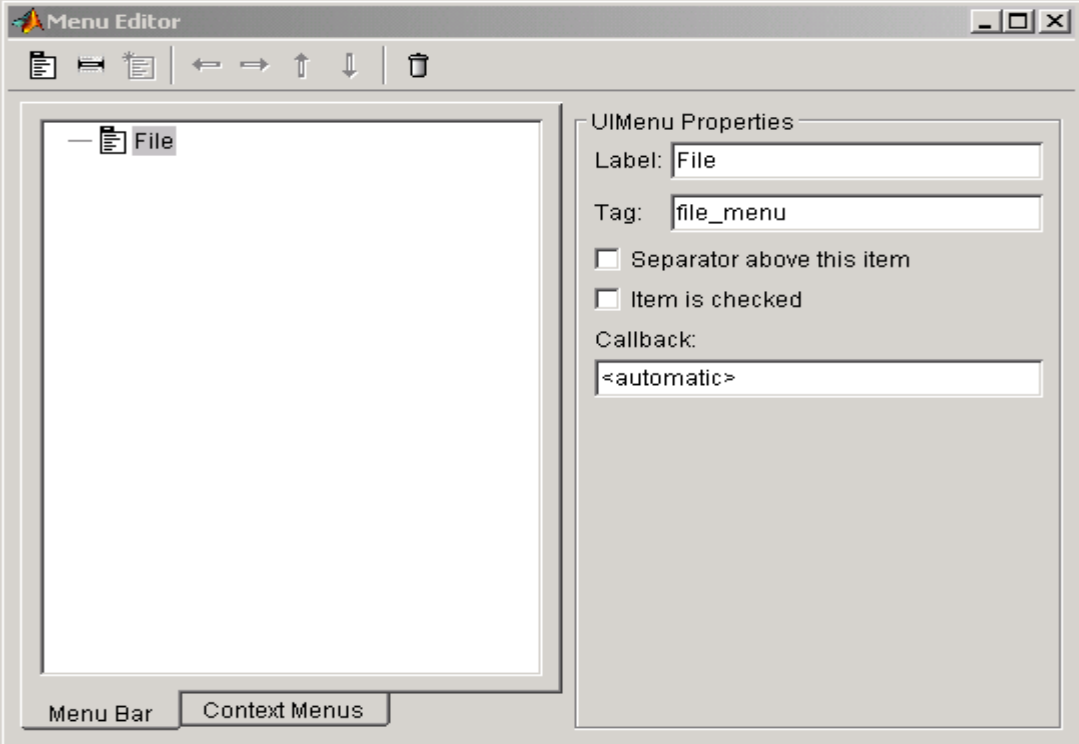
כעת הממשק הגרפי מוכן לפעולה. הריצו אותו מה-Command Line.

ניתן גם לשלוט בתפריטים בראש הגרף.
לחצו על Menu Editor בתפריט Tools ובחרו ב-New Menu כפי שמופיע בגרף 13.9.



גרף 13.9: חלון Menu Editor

כעת בחרו את שם התפריט ע"י ציון Label כ-File. תנו שם לפונקצית ה-Callback שתיווצר ע"י ציון ה-Tag.



גרף 13.10: חלון Menu Editor- יצירת תפריט File

כדי ליצור תת תפריט של תפריט File בחרו ב-File ולחצו על New Menu Item. באותו אופן קיבעו את ה-Label וה-Tag של תת תפריט close.

לאחר שמירה נותר רק לתכנן את ה-Callback-ים של שני התפריטים File ו-Close. למעשה תפריט File יפתח אוטומטית בעת לחיצה ואין ביישום זה צורך לתכנן אותו. ה-Callback של תת תפריט close משתמש בפונקצית delete כדי לסגור את חלון ה-GUI. מאחר וה-Tag של חלון הגרפיקה הוא figure1 אז הקוד הוא,

```
function varargout = close_menu_Callback(h, eventdata, handles, varargin)
delete(handles.figure1)
```

רשימת מונחים

S

T

U

V

K

L

M

N

O

P

Q

R

W

X

Y

Z

מקורות

- Matlab help and PDF user manuals
- Magrab, Azarm, Balachandran, Duncan, Herold, Walsh, An Engineer's Guide to MATLAB