

## Contents

4	סדר פעולות במאטלאב
4	הכנסת ערכים
4	שמות משתנים
4	אפשרויות הדפסה
4	פונקציות מובנות במאטבלאב
6	שימוש ב- $\log$ ו- $e$ ים
6	פונקציות טריגונומטריות
7	פונקציות שימושיות
7	משתנים מיוחדים (או מספרים מוכתבים)
8	עבודה עם קבצים
8	קבצי $m$
8	פתיחת הקובץ
8	שמירת הקובץ
8	הרצת הקובץ
8	סימנים במאטלאב
8	וקטורים (ומטריצות)
8	הגדרת מטריצה
8	שורה
8	עמודה
8	Transpose
9	פרטים על המטריצה
9	Size & Length
10	פונקציות אינדקסים
10	נקודותיים (יצירת וקטור שורה)
10	linspace
11	שליפת איבר מתוך מטריצה
11	שליפת כל איברי המטריצה לעמודה
12	החלפת ערך במטריצה
13	פונקציות מטריצה
13	יצירת מטריצות "חודיות"
13	Zeros – פונקציה ליצירת מטריצה אפס
13	Ones – פונקציה ליצירת מטריצה של 1'ים
13	Eye – פונקציה ליצירת מטריצה אלכסונית
13	Diag – שליפת אלכסון ממטריצה

13.....	Sort – מיון מטריצות
14.....	Reshape – שינוי מימדי המטריצה
15.....	Rand – יצירת מטריצה מערכים רנדומלים
15.....	SUM – יצירת סכום מאיברי המטריצה
15.....	MEAN – יצירת ממוצע של איברי המטריצה
16.....	MAX – מציאת וקטור מקסימלי במטריצה
16.....	מציאת מטריצה הפוכה
16.....	Fliplr – סיבוב וקטור משמאל לימין
16.....	מספרים מרוכבים
16.....	פעולות בין מטריצות
16.....	מכפלה סקלרית
16.....	DOT
16.....	מכפלה וקטורית
16.....	מכפלה של מטריצות
17.....	פתרון מערכת ליניארית במאטלאב
18.....	פעולת הנקודה
18.....	משתנים שונים (char, string etc.)
18.....	עבודה עם מחרוזות (String/char)
19.....	הפיכת string/integer
20.....	הצגה של ערכים על המסך - disp()
20.....	קליטת ערכים מהמשתמש - input()
20.....	חישוב מספר בעל תנאים מגבילים (מתחלק ב-3, אם נעלה אותו בחזקה יהיה גדול מ.. וכו')
21.....	הצגת ערכים כטבלה - Table
21.....	גרף דו-מימד
22.....	Plot()
23.....	וקטור יחיד ב-plot: plot(y)
24.....	עיצוב הגרף (הפניה להלפ)
24.....	הצגת שתיים או יותר גרפים
25.....	הוספת כתוביות לצירים
25.....	הוספת מקרא
25.....	כותרת לגרף
25.....	יצירת "רשת" - grid
26.....	בחירת תצוגת גרפים בחלונות שונים - figure(n)
26.....	הוספת טקסט בגרף
27.....	קביעת גבולות מערכת צירים

27	הוספת גרף לגרף אחר
28	חלוקה לתתי גרפים בתוך הגרף
29	פונקציות
29	הגדרת פונקציה חדשה
30	שמירת הפונקציה
30	הפעלת(קריאה ל..) הפונקציה
31	פונק' ללא ערכים(או ללא קלט)
32	משתנים לוגיים
32	הגדרת משתנה לוגי חדש
32	בדיקה האם המשתנה הוא לוגי בעזרת – islogical()
32	מעבר בין לוגי לנומרי
33	מטריצות לוגיות
33	המרה של משתנה מנומרי ללוגי
33	יחסים לוגיים
34	דוגמא חשובה
35	פעולות לוגיים(AND/OR/NOT/XOR)
35	פקודות חיפוש
35	Find()- מציאת את המקומות המספרים שהם לא אפס בוקטור
36	משפטי בקרה
36	else/elseif/If
37	לולאת for
37	לולאת while
38	ביצוע פעולות על מספר איברים
38	פולינום
38	פונקציות פולינום
38	Polyval(p1,x)
40	חיבור וחיסור בין פולינומים
40	מכפלה בין פולינומים(conv)
41	חילוק בין פולינומים(deconv)
41	נגזרת של פולינום
41	שורשים של הפולינום(פתרון משוואה אופיינית)
41	יצירת פולינום העובר דרך נקודות מסויימות(polyfit)
42	חישוב ערך הפונקציה בנקודות ביניים(interp1)

## כללי

מאטלאב מתחשב בסדר פעולות, משתנים בודדים (סקלרים) מיוצגים בעזרת מטריצה 1 על 1. כלל המשתנים בתוכנה הם מטריצות, משתנים בודדים (סקלרים) מיוצגים בעזרת מטריצה 1 על 1.

## סדר פעולות במאטלאב

- ( )
- ^
- ~
- /, \*
- -, +
- =, <, >, <=, >=
- AND
- OR

## הכנסת ערכים

במידה ונכניס ערך ללא שיוכו למשתנה, הוא יוכנס למשתנה ברירת מחדל בשם "ans" אשר יידרס כל פעם ע"י ערך חדש שלא ישויך למשתנה.

קביעת ערך למשתנה תעשה ע"י:  $a = 70$ , יצור משתנה בשם a עם הערך 70.

- ניתן לרשום מספר פקודות באותה השורה ע"י הפרדתם עם ";", בצורה הבאה:

$a=1; b=2; c=3;$

בצורה זו לא נראה את ההדפסה בחלון הפקודות, אך הפקודה תתבצע.

ניתן להפריד את הפקודות גם ע"י ",", אשר יגרום להדפסת הפקודות המבוצעות:

$a=1, b=2, c=3$

- שמירת משתנים, תעשה ע"י הפקודה `save <filename>` (כברירת מחדל ישמור לקובץ

`(matlab.mat`

- הטענה, תעשה ע"י הפקודה `load <filename>` (כברירת מחדל יטעין קובץ `matlab.mat`)

## שמות משתנים

חייבים להתחיל באות, יכול להכיל אותיות/מספרים/קו תחתון, והוא Case sensitive.

## אפשרויות הדפסה

Format long – יציג לנו את המספר העשרוני עם 14 ספרות אחרי הנקודה

Format Short – יציג את המספר עם 4 ספרות אחרי נקודה

Format Bank – 2 ספרות אחרי הנקודה

Format Rat – יציג את התשובה כשבר פשוט

Format Compact – בוחרים באחת מהאפשרויות מעלה (בד"כ short), הוא דוחס את התצוגה של

המספרים (מוותר על הרווחים בהדפסה)

Format Loose – מרווח בין השורות (ההפך מ-Compact)

הצגת מספרים גדולים – ניתן להראות מספר גדול כ-  $1e3 = 1 \cdot 10^3$

לדוגמא:  $r1=4e3$  ← יהיה בעצם  $4 \cdot 10^3$

## פונקציות מובנות במאטלאב

פורמט:

Function name(input) = output

אם ה-Output הוא יותר מ 1, יש להכניס אותו בסוגריים מרובעים.

לדוגמא, אם נרצה לחשב ריבוע מסוים:

דרך אחת:

$$R2=(30^2+40^2)^{(0.5)}$$

דרך נוספת, בעזרת פונקציה:

$$R2 = \text{sqrt}(30^2+40^2)$$

או לדוגמא מכנה משותף:

$$R4=\text{gcd}(18,24)$$

אך אם קיימת פונקציה עם 2 משתנים בפלט, נכתוב אותה כך:  
 $\leftarrow [r1,r2] = \text{size } b$  יחזיר לנו את הגודל (שורות, עמודות) של המטריצה b.

- ניתן לראות מידע על הפונקציות בעזרת: `help <function name>`
- ניתן לחפש פונקציות לפי: `lookfor <part of function name>`
- כמו כן קיים פירוט על הפונקציות השונות בהלפ.
- קיימות פונק' שונות, `log` לדוגמא, וכו'.

כמו כן, המספר הכי קטן בו ניתן לחלק את 4 ו-10 לדוגמא (במקרה הזה 20), ימצא כך:  
`Lcm(4,10)`

### שימוש ב-e ו-log

ניתן לייצג את e בחזקת 2 כ- `exp(2)`,  
 לדוגמא:

$$10 = \text{Log}_2(1024)$$

שווה ערך ל:  $\log_2(1024) = 10$

- במאט לאב, הלוג הוא בבסיסו הטבעי e. ולכן לוג רגיל יש צורך לעשות כך: `log10(number)`

### פונקציות טריגונומטריות

- `sin()`, `cos()`, `tan()`, `cotan()`, `asin()`, `acotan()`
- ה-input הינה זווית ב-radian.
- כאשר נרצה שה-input יהיה במעלות, ניתן יהיה לבחור באפשרות כך:  
`Sind()`, `cosd()`, `cotand()`, `asind()`, `acotand()`
- חשוב להבדיל, שבמידה ומופיע ביטוי כזה:

$$\cos^2\left(\frac{5\pi}{6}\right) \sin^2\left(\frac{7\pi}{8}\right)$$

אז יש לחשב כך:

$$\text{Cos}(5*\pi/6)^2 * \sin((7*\pi/8)^2)$$

- במידה ונרצה לדעת באיזה directory אנחנו נמצאים, נסתכל על החלון מעל חלון הפקודות.
- במידה ונרצה לשנות את ה-path, נבחר זאת בעזרת בחירה ב-"..."
- כמו כן, ניתן לראות את ה-path בוא נמצא ע"י הכנסת פקודת pwd.
- Who – רשימת המשתנים במרחב העבודה שלנו
- Who's – רשימת המשתנים עם אינפורמציה עליהם.
- Dir – מציג את רשימת הקבצים בתיקייה הנוכחית.
- Cd <dir name> - כניסה לתיקייה מסוימת(כאשר cd .. מחזיר אותנו לתיקייה אחת מעלה מעל הנוכחית)
- ניתן "למחוק" משתנה בעזרת פקודה clear <name>, clear <name>, כמו כן ניתן להשתמש ב-clearall, בכדי למחוק את כלל המשתנים.
- **אסור** לקרוא למשתנים על שמות הפונק' כי הדבר יגרום לפונקציה לא לעבוד.

**משתנים מיוחדים(או מספרים מרוכבים)**

- Ans – משתנה ברירת מחדל אשר יוצר במידה ולא נכניס את החישוב למשתנה כלשהו.
- Pi – הערך של Pi.
- Nan – Not a number.
- Inf – Infinity
- i/j –  $(-1)^{1/2}$

לייצוג מספרים מרוכבים, לדוגמא:  $a=3i+9$  בעצם ייחבר את המספר המרוכב 3 עם 9 כש-3 הוא החלק המדומה וה-9 הוא החלק הממשי.

כמו כן, ניתן להכניס לתוך i ערך כלשהו, ואז במידה וקיים סימן בין המספרים המאטלאב יתחס ל-i כמו לכל משתנה אחר, לדוגמא:

$a = a*i+9$  כפול המספר שקיים ב-i וחיבור עם 9.

לאומת זאת  $ai+9$  ייצג מספר מרוכב.

- Round() – עיגול המספרים בצורה הרגילה(מעל חצי/פחות חצי)
- Fix() – עיגול המספר לכיוון ה-אפס.
- Ceil() – עיגול לכיוון אינסוף(מעלה)
- Floor() – עיגול לכיוון מינוס אינסוף(מטה)
- Abs() – ערך מוחלט.

לדוגמא:

$\text{Round}(5.5)=6$

$\text{Ceil}(5.5)=6$

$\text{Floor}(5.5)=5$

$\text{Fix}(-5.5)=-5$

$\text{Floor}(-5.5)=-6$

$\text{Ceil}(-5.5)=-5$

$\text{Abs}(-5.5)=5.500$

## דוגמא

יש לעגל את הספרות במספר עד לעיגול  $n$  ספרות:

$$\text{round}(c1 \cdot 10^n) / 10^n$$

(כאשר  $n$  הינו משתנה שנקבע כל פעם).

## עבודה עם קבצים

### קבצי m

קובץ טקסט נקי ללא ייצוגים (באסקי), משמש לביצוע רצף פקודות במקום להכניס הכל אחד אחד לתוך התוכנה בחלון הפקודות. נשתמש בקובץ בכדי לרשום רשימה של פקודות, ובעצם הדבר יהיה שקול לכתיבה בחלון הפקודות.

### פתיחת הקובץ

נפתח את הקובץ ע"י File->New->MFile

### שמירת הקובץ

השמירה תעשה ע"י בחירה בחלון (שהסמן יופיע שם), ואז Save as. (כוכבים ליד שם הקובץ יצביע על כך שהתבצע שינוי אך טרם שמרנו את הקובץ).

### הרצת הקובץ

- ייעשה ע"י לחיצה על f5, אשר מבצע שמירה לקובץ ולאחר מכן הרצתו. כמו כן:
- ניתן לכתוב בחלון הפקודות את שם הקובץ, והוא ירוץ גם כן.
- ניתן להריץ את הקובץ ע"י ctrl+enter כאשר אנו בתוך העריכה של הקובץ.
- ניתן לסמן עם העכבר את הפקודות שאנו רוצים שירוצו בתוך הקובץ, וללחוץ על F9.
- 

## סימנים במאטלאב

- % - כל מה שנכתב מימין לאחוז, יהיה בגדר "הערה".
- קווקו אדום/כתוב – יסמן על הערה/בעיה בתוכנה (בדומה לג'אווה).
- % - חלוקה של הקובץ ל-section שונה כל פעם בכדי ליצור סדר מסוים בקובץ, וניתן להריץ תא מסוים באותו הקובץ והדבר מקל על העבודה.
- תא מסויים נריץ ע"י מיקום הסמן בתוך התא שאנו נרצה, לאחר מכן מקש ימני, ובחירה ב-Evaluate Current Cell, או ctrl-enter.
- ניתן ליצור תאים וכמו כן לעבור עליהם במהירות בעזרת המקשי אחוזים שנמצאים מעל חלון העריכה.
- ; - כאשר לא נשים ב-editor את הסימן הזה בסוף השורה, הפלט יודפס למסך.

## וקטורים (ומטריצות)

### הגדרת מטריצה

#### שורה

```
d_row=[32 4 81 exp(2.5) 63 cos(pi/3) 14.12]
```

יצור מטריצה עם הערכים שבפנים, כאשר הם בייצוג שורה (שכן הם מופרדים על ידי רווח או פסיק) ובכך תיווצר מטריצה של 1x7.

#### עמודה

```
d_colm=[32; 4; 81; exp(2.5); 63; cos(pi/3); 14.12;]
```

בדומה לשורה, אבל מופרד ע"י ;

### Transponse

```
d_colm =[32; 4; 81; exp(2.5); 63; cos(pi/3); 14.12;]'
```

(הגרש בסוף הסוגריים)

או

D\_colm'

מעביר שורות לעמודות ולהפך, בדוגמא הנ"ל, יוצר ווקטור שורה במקום וקטור עמודה.



## פרטים על המטריצה

### Size & Length

$[m, n] = \text{size}(A)$

או

$[m \ n] = \text{size}(A)$

יכניס ב-m את מספר השורות, וב-n את מספר העמודות של המטריצה A

$A = \text{length}(\text{vec})$

יכניס ל-a את האורך של הוקטור vec.

דוגמא ליצירת מטריצה של  $2 \times 2$ :

$A = [1, 2 ; 3, 4]$

"יצור בעצם מטריצה:

1 2

3 4

כמו כן, ניתן לבצע עליה פעולות כגון Transpose

$A' =$

3 1

2 4

בנוסף, ניתן לבצע פעולות על מספר וקטורים ומטריצות בצורה הבאה:

$A = [1 \ 2 \ 3 \ 4]$

$B = [5 \ 6 \ 7 \ 8]$

$C = [a ; b]$

## פונקציות אינדקסים

### נקודותיים (יצירת וקטור שורה)

`V1 = a:b`

יוצר וקטור שורה, אשר איברים מתחילים ב-a ומתקדמים בפסיקות של 1 אל b או אל ערך שקרוב אליו מלמטה (b הוא חסם עליון).  
לדוגמא:

`1:3`

`1 2 3`

כמו כן קיימת צורה נוספת:

`V2 = a : n : b`

כאשר ה-n מייצג את מספר ה"צעדים" שנעשה בכל פעם.

דוגמא:

ליצור וקטור עמודה שהולך מהספרה (15) עד לספרה (-25) בקפיצות של -5.

```
colu = 15:-5:-25;  
colu'
```

או בדרך אחרת:

```
colu = [15:-5:-25]'
```

## הערה חשובה:

במידה ונשתמש בפקודה: `0:0.2:2*pi`, בכדי לקבל את הוקטור של מ-0 עד  $2\pi$  בקפיצות של 0.2. הדבר לא מומלץ שכן אנו לא בטוחים ש"נפגע" ב- $2\pi$  בעזרת 0.2 כל פעם, ולכן, עדיף להשתמש ב-`linspace`.

### [linspace](#)

`Linspace(a, b)`

יוצר וקטור שורה בעל 100 איברים, אשר איברים מתחילים ב-a ומסתיימים ב-b, הפקטור בו גדלים האיברים הוא קבוע. (בעצם החישוב נעשה ע"י  $(a-b)/99$ ).

`Linspace(a, b, n)`

בדומה, יוצר וקטור שורה, אבל כאן, ניתן לקבוע את מספר האיברים בוקטור ע"י n, וגודל הפסיעה יהיה קבוע (ויחושב ע"י  $(a-b)/(n-1)$ ).  
לדוגמא:

`D1 = linspace(1, 100)`

ייצור וקטור של 1 עד 100 בקפיצות של 1.

`D2 = linspace(1, 8, 5)`

ייצור וקטור אשר האיברים הראשון בו הוא אחד, והאחרון הוא 8, ויש בו 5 איברים.

### שליפת איבר מתוך מטריצה

נניח כי קיימת מטריצה כזאת:

ונקרא לה **B**

$(1,1) - 1$	$(1,2) - 4$	$(1,3) - 7$
$(2,1) - 2$	$(2,2) - 5$	$(2,3) - 8$
$(3,1) - 3$	$(3,2) - 6$	$(3,3) - 9$

ונרצה לשלוף את האיבר שיושב בתא (2,2) לדוגמא:

$$B_{22} = B(2, 2)$$

נניח ונרצה לשלוף את כל העמודות בשורה מספר 2:

$$B_{r2} = B(2,:)$$

לדוגמא נרצה לשלוף את העמודה השלישית של המטריצה:

$$B_{c3} = B(:,3)$$

## **- נקודותיים במאטלאב מציינים רצף של אינדקסים**

### שליפת כל איברי המטריצה לעמודה

**- END במאטלאב יצביע על האינדקס האחרון.**

תיעשה בצורה הבאה:

$$A = B(:)$$

כמובן שאם נרצה לשלוף את האיברים לשורה, ניתן להפוך ע"י ה-transpose:

$$A = B(:,')$$

למעלה בכחול, מצוין מספר האיבר של המטריצה כפי שמאטלאב רואה אותו, ולכן, במידה ונרצה לשלוף את איבר מספר 5 (שזה בעצם התא 2,2) נבצע זאת כך:

$$B = B(5)$$

במידה ונרצה לשלוף **תת מטריצה**, נבצע זאת כך:

$$Mat = B(1:2, 2:3)$$

ישלוף לנו בעצם את ההצלה של הסוגריים, במקרה שלנו, את התאים של השורות 1,2 בעמודות 2,3.

### דוגמא נוספת

הפקודה:

$A = B([1,2,4], 3)$  - ישלוף את זה לזווקטור עמודה המורכבת מעמודה מס' 3 עם האיברים של השורות שנמצאים באינדקסים 1,2,4

### עוד דוגמא

$$A = B([1\ 3\ 5], [1\ 3])$$

יכניס את האיברים כתת מטריצה של השורות והעמודות המצוינות בסוגריים. בצורה דומה, ניתן לבצע זאת כך:

$$A = B(1:2:end ; 1:2:end)$$

### החלפת ערך במטריצה

תעשה ע"י הפקודה הבאה:

$$B(2,3) = 4$$

יכניס את הערך 4 לתא 2,3 של המטריצה מעלה.

לדוגמא נרצה להכניס תת מטריצה למטריצה קיימת, נבצע זאת כך:

$$B(1:2,1:2) = [3 \ 4 ; 5 \ 6]$$

יכניס לתאים של השורה הראשונה והשנייה ועמודות ראשונה ושנייה את הערכים 3,4,5,6.

(יש לשים לב כי חייבת להיות תת מטריצה כאשר נרצה לשייך אותה למטריצה קיימת!)

לדוגמא ונרצה "לנקות" תא מסוים, נעשה זאת ע"י הכנסת סוגריים ריקים (וקטור ריק) לתא הרצוי.

$$B(3, [1 \ 3]) = 0$$

יאפס את השורה 3 בעמודות 1 ו 3.

### הערה חשובה

כאשר נרצה לשלוף את שורה מספר 1 ו 2, נוכל לבצע זאת ע"י:

$$A([1,2], :)$$

ישלוף בעצם את השורות 1 ו 2.

בצורה דומה:

$$A(:, [1,3])$$

ישלוף את העמודות 1 ו-3.

- במידה ונכניס וקטור לתוך המטריצה, יש צורך להקפיד כי הגודל שלו מתאים למטריצה, לדוגמא אם הדבר הוא עמודות, יש צורך לוודא שהמספר העמודות תואם את המספר במטריצה.
  - במידה ונכניס וקטור גדול יותר מהמטריצה, או קטן יותר מהמטריצה, התוכנה תציב אפסים במקומות בהם לא נמצא תיאום.
- לדוגמא אם נסיף מספר למטריצה בעמודה שלא קיימת, המטריצה תיצור עמודה המורכבת מאפסים והמספר אותו הכנסנו באותה עמודה.

## פונקציות מטריצה

### יצירת מטריצות ייחודיות

#### Zeros – פונקציה ליצירת מטריצה אפס

$\text{Zeros}(m,n)$  - תייצר מטריצה, בגודל  $m \times n$ , שכל האיברים בה הם אפס.  
 $\text{Zeros}(m)$  – תייצר וקטור שורה שהיא אפסים

#### Ones – פונקציה ליצירת מטריצה של 1'ים

$\text{Ones}(m,n)$  – תייצר מטריצה בגודל  $m \times n$ , שכל האיברים בה הם אחדים.  
 $\text{Ones}(m)$  – תייצר וקטור שורה שכל האיברים בו הם 1.  
 $\text{Ones}(3,5) * 4$  – יצור מטריצה 3 על 5 שמלאה בערכים שהם 4.

#### דוגמא ליישום:

$A = [\text{ones}(2,2), \text{ones}(2,3)*2; \text{ones}(3,1)*3, \text{ones}(3,2)*4, \text{ones}(3,2)*5]$

A =

1	1	2	2	2
1	1	2	2	2
3	4	4	5	5
3	4	4	5	5
3	4	4	5	5

#### Eye – פונקציה ליצירת מטריצה אלכסונית

פונקציה ליצירת מטריצה אלכסונית (בעלת אפסים מחוץ מאלכסון)

$\text{Eye}(m,n)$  – תיצור מטריצה של  $m \times n$  שיהיו לה אחדים אך ורק באלכסון.

#### לדוגמא

$\text{Eye}(6,9)$  - יצור מטריצה של 6 שורות ו 9 עמודות שבאלכסונה יהיה 1.

$\text{Eye}(3) * 3$  – יצור מטריצה 3 על 3, עם הערכים 3 באלכסון.

#### Diag – שליפת אלכסון ממטריצה

$\text{Diag}(b)$  – יצור מטריצה של ערכים של המטריצה b כאלכסון של המטריצה החדשה.

#### Sort – מיון מטריצות

מיון איברים, במאטלאב בצורת ברירת מחדל המיון ייעשה על עמודות.

נניח ונבצע  $\text{sort}(A)$

אם A וקטור – מיון כל איברי A

אם A מטריצה – מיון לפי עמודות

$\text{Sort}(A, \text{dim})$

כאשר ברירת ה-dim, ישפיע על הצורה בה המטריצה תמוין (סדר עולה/יורד, מיון עמודות/שורות, וכו')

## לדוגמא

```
D_row = [4 pi 0 1]
S_d_row=sort(d_row)
Result: s_d_row = [0 1 pi 4]
```

נפעיל מיון על מטריצה:

```
D = [1:4; 0:3; 5:-1:2]
```

sort(D) - יבצע מיון של כל עמודות המטריצה.

sort(D,2) - יבצע מיון לפי שורות.

קיימת דרך נוספת למיון של שורות, בעזרת 2 "מהפכים" שלה בצורה הבאה:

Sort(D') - יבצע מיון של שורות.

Sort(D(:)) - ימיין את כל איברי המטריצה.

Sort(d\_row, 'descend') - ימיין את המטריצה בסדר יורד.

## Reshape – שינוי מימדי המטריצה

Reshape(A, m, n) - משנה את המטריצה כתלות במידות שהוכנסו בפונקציה.

## לדוגמא

נרצה להכניס את הוקטור שורה שרץ מ-1 עד 12, לתוך מטריצה של 2x6:

נניח ו-D, הינה וקטור שורה של מספרים מ-1 עד 12. צורה אחת:

```
D = reshape(A,2,6)
```

ייצור מטריצה D:

1	3	5	7	9	11
2	4	6	8	10	12

לאומת זאת אם נרצה שתיווצר מטריצה עם סדר הכנסה אחר, נעשה זאת כך:

```
D = reshape(A,6,2)
```

ייתן לנו את התוצאה:

1	7
2	8
3	9
4	10
5	11
6	12

כמובן שזה לא מתאים, ולכן נסיף ' שיעזור לנו לשנות את המטריצה לגודל הרצוי:

```
D = reshape(A,6,2)'
```

ייתן לנו:

1	2	3	4	5	6
7	8	9	10	11	12

**חשוב לזכור:** במידה והמטריצה אליה אנחנו מכניסים תהיה לא גדולה מספיק להכיל את הווקטור, אנו נקבל הודעת שגיאה.

### Rand – יצירת מטריצה מערכים רנדומלים.

$R1 = \text{rand}(3, 4)$  – תיצר מטריצה של  $3 \times 4$ , שלכל אחד מהתאים יהיה סיכוי זהה להיות בין 0 ל-1.

לדוגמא ונרצה שהערכים במטריצה יהיו בין  $a$  ל- $b$ , נוכל לבצע זאת כך:  
 $\text{Rand}(m,n) * (b-a) + a$  – ייתן לנו מטריצה של  $m \times n$ , כאשר הערכים בה יהיו מספרים בין  $a$  ל- $b$ .

בכדי ליצור לדוגמא, מספר רנדומלי שמתחלק ב 9, בין 1 ל 10000,

נעשה זאת בצורה הבאה:

$$A = 9, b = 9999$$

$$\text{Round}(\text{Rand} * (9999-9) + 9) * 9$$

### SUM – יצירת סכום מאיברי המטריצה

$$\text{Sum}(A, \text{DIM})$$

כאשר ה-DIM יכול להשתנות כדי להשפיע על הסכום השונה של הפונקציה. לדוגמא:  
 $\text{Sum}(D)$  – ייתן לנו בעצם את סכום האיברים בעמודות הווקטור כווקטור שורה.  
 $\text{Sum}(D, 2)$  – ייתן לנו את סכום השורות של המטריצה כווקטור עמודה.  
 $\text{Sum}(D(:))$  – יהפוך את המטריצה  $D$  לווקטור עמודה, ולאחר מכן יסכום את כל איברי אותו הווקטור, בפועל, נקבל סכום של כל איברי מטריצת  $D$ .  
 $\text{Sum}(\text{Sum}(D))$  – ייתן גם כן את סכום כל איברי המטריצה.

### MEAN – יצירת ממוצע של איברי המטריצה

$$\text{Mean}(D, \text{DIM})$$

כאשר ה-DIM יכול להשתנות כדי להשפיע על הסכום השונה של הפונקציה.

$\text{Mean}(D)$  – ייתן את ממוצע כל האיברים בעמודות המטריצה כווקטור עמודה.  
 $\text{Mean}(D, 2)$  – ממוצע של כל איברי המטריצה לפי שורות כווקטור עמודה.  
 $\text{Mean}(D(:))$  – ממוצע של כל איברי המטריצה  
 $\text{Mean}(\text{Mean}(D))$  – כנ"ל.

בנוסף, ניתן לחשב את הממוצע של איברי המטריצה בצורה הבאה:  
לדוגמא, יהיה מטריצה  $a$  בעלת איברים.  
נוכל לחשב זאת כך:

$$[m \ n] = \text{size}(a);$$

$$\text{meanA} = \text{sum}(a(:))/(m*n);$$

## MAX – מציאת וקטור מקסימלי במטריצה

$$\text{Max}(D, [], \text{DIM})$$

כאשר ה-DIM יכול להשתנות כדי להשפיע על הסכום השונה של הפונקציה.

$\text{Max}(D)$  – ימצא את האיבר המקסימלי בכל עמודה וייצר מהם וקטור שורה.  
 $\text{Max}(D, [], 2)$  – ימצא את האיבר המקסימלי בכל שורה וייצר מהם וקטור עמודה.  
 $\text{Max}(\text{Max}(D))$  – האיבר המקסימלי במטריצה  
 $\text{Max}(D(:))$  – כנ"ל.

## מציאת מטריצה הפוכה

תיעשה ע"י הפקודה:

$$\text{Inv}(A) = A^{-1}$$

## Fliplr – סיבוב וקטור משמאל לימין

$\text{Fliplr}(A)$  – יסובב את הערכים של A משמאל לימין (תהפוך את הוקטור)

## מספרים מרוכבים

$\text{Complex}(a, b)$  – כאשר a – המספר הממשי, ו-b המספר המדומה.

או לחילופין ניתן להגדיר גם:  $a = 3 + 4i$

$\text{Angle}(a)$  – יחזיר את הזווית בין הציר המדומה לציר הממשי ברדיאנים.

## פעולות בין מטריצות

### מכפלה סקלרית

$2 * A$  – יכפול את המטריצה A בסקלר 2.

### DOT

מכפלה של 2 וקטורים:

$\text{Dot}(u, v)$  – יבצע את הפעולת כפל בין וקטור u לוקטור v.

### מכפלה וקטורית

יבצע ע"י ה-cross, בצורה הבאה:

$\text{Cross}(a, b)$  – יבצע מכפלה וקטורית בין 2 הוקטורים.

### מכפלה של מטריצות

$$A * B$$

הדבר כמובן שונה מ-  $B * A$

אפשרית רק כאשר מספר העמודות של המטריצה A שווה למספר השורות של המטריצה B

כלל אצבע:  $[n \times m]$  כאשר ה-m וה-n שווים (לא משנה של מה)



### פתרון מערכת ליניארית במאטלאב

כידוע, מערכת משוואות ליניארית נתונה בצורה הבאה:

$$A * \underline{x} = C$$

כאשר  $A$  – הינה מטריצת המקדמים

$\underline{x}$  – הינה מטריצת הנעלמים

$C$  – הינו וקטור הפתרון

לאחר מספר פעולות פשוטות, נגיע ל:

$$\underline{x} = A^{-1} * C$$

מה שבעצם מהווה את הפתרון של המערכת הזו.

במאטלאב ניתן לבצע זאת כך:

$$A^{-1} * C$$

$$\text{Inv}(A) * C$$

$$A \setminus C$$

לדוגמא:

$$X_1 + 2x_2 + 3x_3 = 366$$

$$4x_1 + 5x_2 + 6x_3 = 804$$

$$7x_1 + 8x_2 = 351$$

טיפטר בצורה הבאה:

$$a = [1 \ 2 \ 3 ; 4 \ 5 \ 6; 7 \ 8 \ 0]$$

$$c = [366 ; 804 ; 351]$$

$$d = (a^{-1}) * c$$

a - מוגדרת ע"י המקדמים של ה-Xים.

c - הינו וקטור הפתרונות

d – מטריצת הנעלמים שאותם נמצא.

### פעולת הנקודה

פעולה זו עוזרת לנו להסתכל על מטריצה כעל מערך שעליו ניתן לבצע פעולות. לדוגמא, פעולה של נקודה בין A ל-B, תבצע את הפעולה המתמטית על כלל האיברים, איבר מול איבר, בצורה הבאה:

$$A.*B$$

יכול את כלל איברי A באיברי B, בצורה הבאה:  $a_{11} * b_{11}, a_{12} * b_{12}, a_{22} * b_{22}, \dots$  כמובן הדבר מחייב ששתי המטריצות יהיו באותו הגודל.

דוגמאות נוספות

$$C2 = A./B$$

תחלק את האיברים של A באיברי B בצורה תואמת.

$$C3 = A.^2$$

יעלה את כל איברי A בחזקת 2

$$C4 = 2.^A$$

יבנה מטריצה בה כל האיברים הם 2 בחזקת האיבר התואם במקום של A.

### משתנים שונים (char, string etc.)

### עבודה עם מחרוזות (String/char)

ניתן לקבוע מחרוזת כמשתנה בצורה הבאה:

$$S='red'$$

ייצור את המחרוזת red בתוך המשתנה S.

ניתן לשלוף אותיות (chars) ממחרוזות שונות, נעשה זאת בצורה הבאה:

$$S1=s(1)$$

יכניס את האות r למשתנה s1, וניתן לבצע זאת עבור כל תו במחרוזת.

כמו כן, ניתן לשלוף אותיות לוקטורים של תווים בצורה הבאה:

X=['ab'; 'cd']

בכדי לצרפם ביחד (את שתי המחרוזות), יש צורך לתחום אותם בסוגריים מרובעים. התוצאה תהיה ווקטור שבשורה הראשונה שלו יהיה את האותיות ab ובשורה השנייה יהיה את האותיות cd.

בצורה דומה, ניתן לבצע זאת כך:

R1=['g', s(2), 't' 'r' s(2) 'a']

וכו'.

### הפיכת string/integer

ניתן להפוך מחרוזות למספרים וגם להפוך, נעשה זאת בעזרת הפקודה הבאה:

A = Num2str(2)

ייתן לנו את מספר 2 כ-תו/מחרוזת.

לאומת זאת:

Str2num(A)

ייתן לנו בחזרה את המספר 2 כספרה לכל דבר.

### דוגמא

ניתן "לספור" ולסכום את האיברים השונים בתוך מחרוזת מסוימת או מספר מסוים, וזאת נעשה בצורה כזאת:

חשב סכום ספרות המספר: 90234

X = 90234

X1 = num2str(x)

X2=x1'

יהפוך את הטקסט מוקטור שורה של תווים, לוקטור עמודה של תווים.

כאשר נבצע את הפעולה הבאה:

X3 = str2num(x2)

וכך בעצם נקבל וקטור עמודה של ספרות אותן ניתן לסכום, וזאת נעשה כך:

S = sum(x3)

### הצגה של ערכים על המסך - disp()

ניתן להציג ערכים שונים על המסך בעזרת הפקודה disp.

לדוגמא:

```
A = 5
```

```
Disp(a)
```

יציג את הספרה 5.

```
Disp('hello')
```

כמו כן, ניתן להציג בכמה שורות בצורה הבאה:

```
Disp([blah blah; blihblih])
```

### קליטת ערכים מהמשתמש - input()

ניתן לקלוט מהמשתמש ערכים שונים ע"י הפונקציה, זאת נעשה כך:

```
X = input('time=?', 's')
```

יציג למשתמש את השאלה time=?, ויצפה לקלט ממנו.

במידה ומיריד את ה: 's', אנו נקלוט מספר בלבד, ולא קלט שייקלט כ-string.

### חישוב מספר בעל תנאים מגבילים (מתחלק ב-3, אם נעלה אותו בחזקה יהיה גדול מ.. וכו')

זאת דוגמא בלבד להמחשה כיצד ניתן לחפש מספר בעל תנאים מסויים.

תרגיל: הצג את המספר הקטן ביותר המתחלק ב-3, אי זוגי, ואם נעלה אותו בשלישית הוא יהיה גדול מ-4000.

פתרון:

```
j=0;
i=1;
while j~=1
    i=i+2;
    j=( mod(i,3)==0 & i^3>4000);
end
disp(i)
```

## הצגת ערכים כטבלה – Table

במידה ונרצה להציג את הערכים שלנו בצורת טבלה עם כותרות, נוכל לבצע זאת בצורה הבאה:

```
A=[ (1:10) ' round(exp(1:10) ' ) ] ;  
disp('          x          exp(x) ' )  
disp(A)
```

יציג לנו:

x	exp(x)
1	3
2	7
3	20
4	55
5	148
6	403
7	1097
8	2981
9	8103
10	22026

## גרף דו-מימד

`Plot(x,y,s)`, הינה הפקודה ליצירת התרשים, כאשר  $x, y$  יכולים להיות סקלרים/וקטורים וכו'.

כאשר  $s$ , יסמל אפשרויות נוספות כגון צבע/צורה, או תכונות שונות של הגרף.

הפונק' תצייר את  $x$  כנגד  $y$ .

ניתן לצייר פונק' שונות במאטבלאב ע"י שימוש בוקטור.

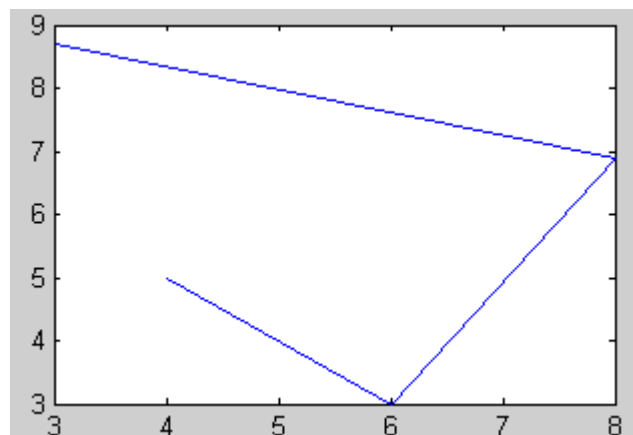
הדבר נעשה ע"י פונק' בשם `plot`.

## Plot()

לדוגמא:

```
Plot([4 6 8 3], [5 3 6.9 8.7])
```

ייצור לנו חיבור של קווים ישרים בוקטורים (לדוגמא: (4,5), (6,3) וכו') והתוצאה תהייה:



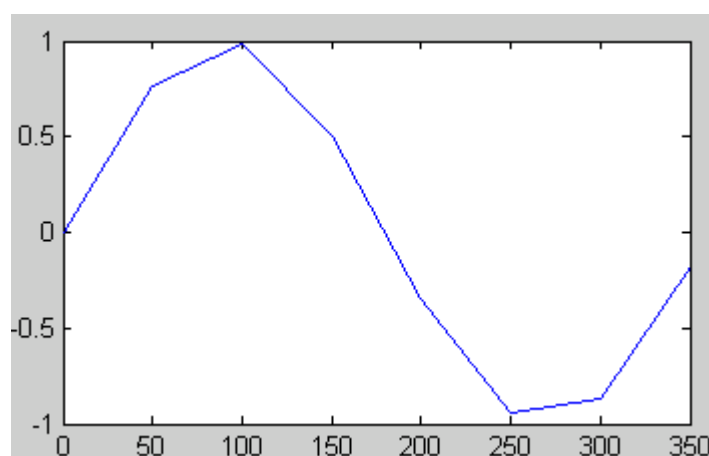
בצורה דומה, ניתן יהיה להציג את הפונק'  $\sin$ :

```
X=0:50:360;
```

```
Y=sind(x)
```

```
Plot(x,y)
```

ניתן לנו את הערכים של הפונק'  $\sin$  בקפיצות של 50 בין 0 ל 360 שתראה כך:

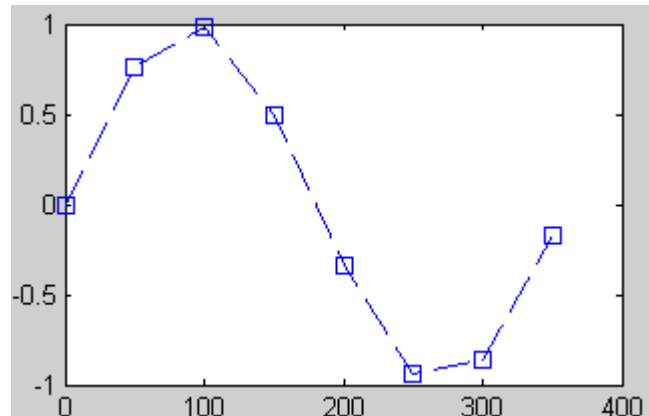


## הגדרות נוספות

ניתן להגדיר הגדרות נוספות על הצורה שבה הנתונים יוצגו, לדוגמא סוג של קו, או כל דבר דומה. פירוט מלא יותר ימצא בהלפ, והעריכה עצמה תתבצע בצורה הבאה:

```
Plot(x,y,'sb--')
```

תשנה את התצורה הקודמת ל:



ניתן למצוא את כלל הפירוט על הטבלאות בהן ניתן להשתמש ב-הלפ של plot.

בנוסף, ניתן גם להשפיע על צורת נקודות/צבע והגדרות נוספות של ה-Plot בעזרת ה-linespec(לחפש בהלפ).

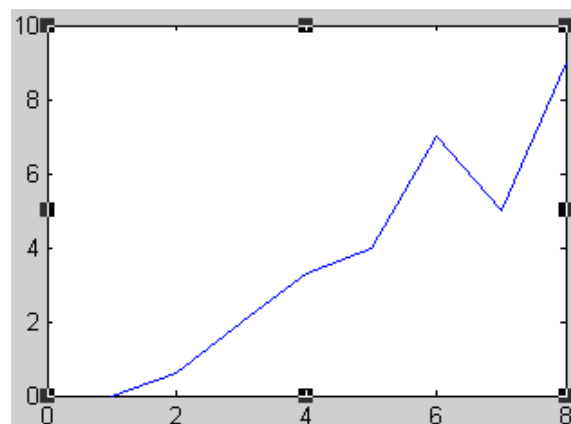
## וקטור יחיד ב-plot(y)

במידה ונרשום plot(y) אנו נצייר את הוקטור y כתרשים של עצמו כנגד האינדקסים שלו.

הדבר יראה כך עבור הפקודה הבאה:

```
y=[0 0.6 2 3.3 4 7 5 9]  
plot(y)
```

התוצאה תהייה:



הדבר דומה לפקודה:

```
plot(1:length(y),y)
```

### הערה חשובה:

במידה ונשתמש בפקודה:  $0:0.2:2\pi$ , בכדי לקבל את הוקטור של מ-0 עד  $2\pi$  בקפיצות של 0.2. הדבר לא מומלץ שכן אנו לא בטוחים ש"נפגע" ב- $2\pi$  בעזרת 0.2 כל פעם, ולכן, עדיף להשתמש ב-linspace.

### עישוב הגרף (הפנייה להלפ)

מקליקים על plot, והולכים ל-help on selection, והדבר יפתח לנו את ההלפ של פלוט (הדבר לוקח הרבה זמן בפעם הראשונה), נלך לKeySpec, וכך נדע את הטבלאות של העישוב.

### הצגת שתיים או יותר גרפים

ניתן להציג במקביל, מספר גרפים בעלי אפשרויות שונות וייחודיות כל אחד להם.

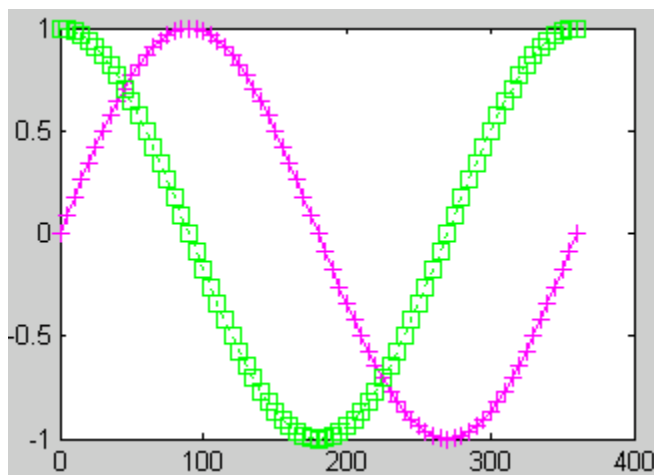
הדבר יעשה פשוט ע"י הוספת הערכים הדרושים ב-plot כמו שהכרנו כבר, הדבר יעשה כך:

```
plot(x1,y1,s1,x2,y2,s2,x3,y3,s3,...)
```

והתוצאה של הפקודה הזאת לדוגמא:

```
plot(x,sind(x),'+-m',x,cosd(x),'g:s')
```

תהיה:





### הוספת כתוביות לצירים

(הכתוביות יכולות להכיל מספרים+אותיות)

ניתן להוסיף שמות לצירים שונים  $(x,y)$  בהתאם לדרישות. נבצע זאת בעזרת הפקודות הבאות:

```
xlabel('X Label text')
```

```
ylabel('X Label text')
```

הדבר יוסיף כתובית לצירים (משמאל לציר  $y$  ומתחת לציר  $x$ ).

### הוספת מקרא

ניתן להוסיף מקרא לגרפים אשר ייתאר לנו כל גרף בנפרד ("ראה כריבוע קטן בצד הימני של הגרף).

נעשה זאת בעזרת הפקודה:

```
Legend('string1','string2','string3');
```

### כותרת לגרף

ניתן להוסיף כותרת לגרף, הדבר יעשה בעזרת הפקודה:

```
title('string');
```

הכותרת תופיע מעל הגרף.

### יצירת "רשת" – grid

ניתן ליצור רשת הנפרשת בגרף, נעשה זאת בעזרת הפקודה:

```
Grid on
```

כמובן בצורה דומה ניתן "לכבות" את הרשת ע"י `grid off`.

### בחירת תצוגת גרפים בחלונות שונים - figure(n)

ניתן לבחור באיזה חלון הגרפים יציירו, ולאילו מין הגרפים אנו נתייחס בעת הפקודות שלנו.

הדבר ייעשה ע"י הכרזת השם של הגרף לפני רצף הפקודות שיהיו קשורות אליו.

לדוגמא: נרצה ליצור 2 חלונות גרפים שונים, עם סט פקודות שונה לכל אחד כדי שיהיה הבדל ביניהם. נעשה זאת בעזרת הפקודה:

```
figure(1)
x=0:5:360;
plot(x,sind(x),'+-m',x,cosd(x),'g:s')
figure(2)
plot(x,tand(x))
figure(1)
grid on
close(1);
close(2);
close(all);
```

ניתן לראות שסט הפקודות שמתחת ל-figure(1), מתייחס רק לגרף בשם figure(1), קרי, החלון 1. כנ"ל לגבי 2, וכו'.

ניתן לסגור כל חלון בנפרד וכמו כן את כלל החלונות ע"י הפקודה close.

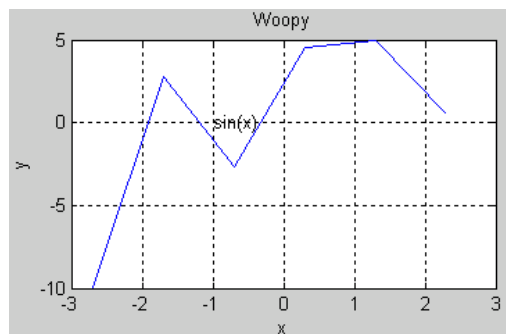
### הוספת טקסט בגרף

ניתן להוסיף טקסט בתוך הגרף ע"י ציון של נק' בהם נכתוב

```
text(x,y,'text')
```

לדוגמא, נרצה להוסיף את הכתובת  $\sin(x)$  ליד הפונק' שלו בגרף. נעשה זאת בעזרת:

```
Text(-1,0,'sin(x)')
```



התוצאה, תראה כך:

### קביעת גבולות מערכת צירים

ניתן לקבוע את הגבולות של מערכת הצירים, בצורה כזאת:

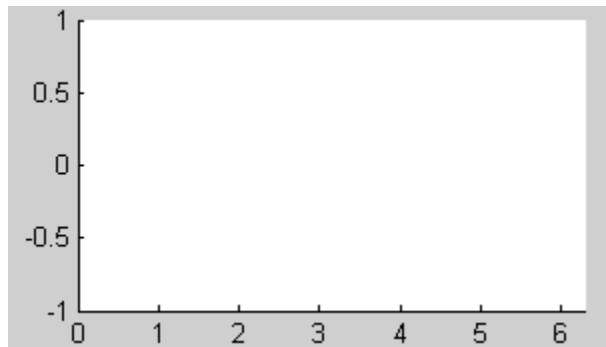
```
Axis([xmin xmax ymin ymax])
```

(ניתן להשתמש ברווח או בפסיק בכדי להפריד בין האיברים).

לדוגמא:

```
Axis([0 2*pi -1 1])
```

התוצאה:



במידה ונרצה לשנות רק את אחד הצירים (דומה ללעשות "זום" על אחד הרכיבים), נוכל לעשות זאת בעזרת הפקודה הבאה:

```
xlim([0 2*pi])
```

כמו כן ניתן גם "למחוק" את הצירים ע"י:

```
Axis off
```

כמו כן ניתן להפוך גרף ל"ריבועי", הדבר הופך אותו לצורת הצגה של ריבוע, אך "מקמט" את הגרף כדי שיתאים לצורה ולכן הוא לא יהיה מדויק (כי לא תהיה פרופורציה בין הצירים). נעשה זאת כך:

```
Axis square
```

### הוספת גרף לגרף אחר

ניתן להמשיך לצייר על אותו הגרף אותו פתחנו בהתחלה, ע"י הפקודה:

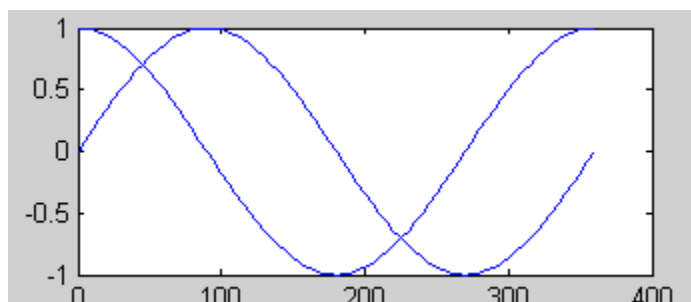
```
Hold on
```

הדבר יגרום לתוכנה להמשיך להתמקד בחלון שעליו אנו כבר עובדים, ובכך "להוסיף" לו עוד גרפים.

לדוגמא:

```
x=0:5:360;  
plot(x,sind(x))  
hold on  
plot(x,cosd(x))
```

ייתן לנו את התוצאה:



### חלוקה לתתי גרפים בתוך הגרף

ניתן לחלק את הגרף (figure) שלנו, לתתי גרפים שיהיו בעצם חלונות בתוכו (subplot).

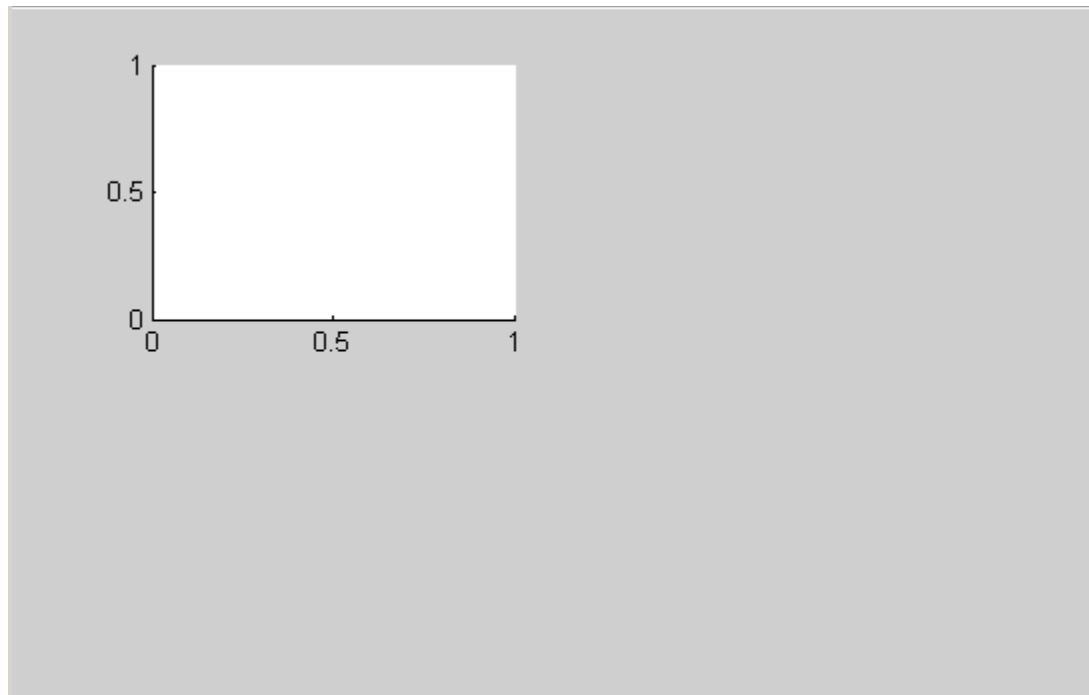
`Subplot(x,y,n)`

כאשר ה-x וה-y יהווה מעין "טבלה" שבה אנחנו מחלקים את הגרפים שלנו (מטריצה) וה-n יצביע על תא מסוים.

נעשה זאת בצורה הבאה:

```
figure(1)
subplot(2,2,1);
```

התוצאה תהיה:

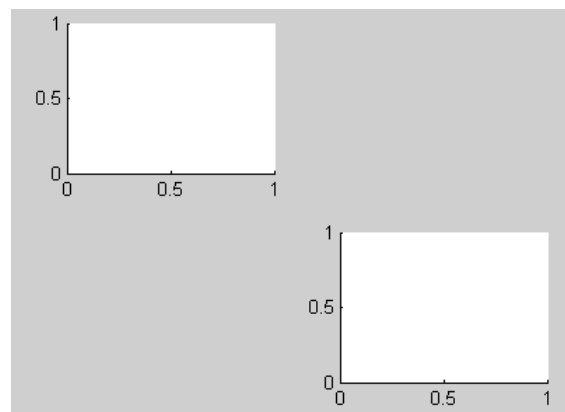


כמובן שניתן להוסיף תתי גרפים נוספים, והדבר יראה כך:

```
subplot(2,2,1);
```

במידה ונרצה ליצור גרף נוסף באותו חלון, אך משמאל למטה. נבצע זאת כך:

```
subplot(2,2,4);
```



במידה ונרצה להציג תצורה כלשהי בתוך הגרף (להתייחס לגרף מסוימת מהאלה שיצרנו) נעשה זאת כך:

```
a1 = 0:15:360;  
figure(1)  
subplot(2,2,1);  
plot(a1,sind(a1),'bo--');  
subplot(2,2,4);  
plot(a1,cosd(a1),'r');
```

ניתן לראות שפקודת הגרף שאנו נצייר בחלון אליו נרצה לכוון, ימצא בידיוק מתחת להגדרת התת גרף (subplot).

## פונקציות

קיימות מספר סוגי פונקציות, בעיקר הן מתחלקות לשתיים, אלו הקיימים במאטלאב (builtin), ואלה המוגדרים ע"י המשתמש (user defined).  
כמו כן קיימים סוגים שונים של פונק' המוגדרות ע"י המשתמש, אך אנו נתמקד בסוג הנקרא M-type Function.

## הגדרת פונקציה חדשה

ייעשה ע"י פתיחת קובץ M-File, אשר יכיל header שיראה כך:

```
Function[a1,a2,a3,a4....]=functionname(I1,I2,I3...)
```

כאשר ה-a מייצג את הפלט של הפונק'  
וה-I מייצג את הקלט שאנו נקבל לפונק'.

- הערה: הפונק' תישמר בקובץ נפרד משלה בלבד (כל פונק' תכתב בקובץ נפרד).
- הקובץ של הפונק' חייב להיות ב-working folder שבו אנו עובדים.

נוכל לעבור ל-working directory בצורה פשוטה יחסית ע"י:

```
Mkdir class_7  
cd class_7
```

## לדוגמא

אם נרצה להגדיר פונק' אשר תעלה בריבוע את ה- $x$  אותו נכניס. חשוב לשים לב, שבמקרה הנ"ל אנו מייעדים את הפונק' גם לווקטורים וגם ל- $x$  לבדו, והדבר יראה כך:

```
function [y] = myfunc1(x)
y = x.^2
```

הנקודה היא בכדי לתת לפונק' אפשרות לטפל גם בוקטורים/מטריצות.

ניתן לראות שהפונק' מקבלת ערך  $x$ , ומחזירה ערך  $y$ .

## שמירת הפונקציה

הדבר הכי חשוב בעת שמירת הפונק', הוא להקפיד שהשם שבו נשמור את הקובץ של הפונק', יהיה למעשה בשם של הפונק' עצמה!

אם נתייחס לדוגמא של הפונק' מלמעלה, יש צורך לשמור אותה בקובץ בשם: myfunc1.m

כמו כן, יש לוודא שהקובץ שלה אכן נמצא באותה תיקייה כמו של הקובץ אותו אנו מריצים(אותו (Current Directiory).

## הפעלת(קריאה ל..) הפונקציה

תעשה בצורה הבאה:

```
U = -3;
V = myfunc1(U);
```

נתן לנו את התוצאה 3 שתכנס ל-V.

```
r=1:5;
t=myfunc1(r);
```

ייתן את התוצאה של וקטור בריבוע(דומה ללעשות  $r.^2$ ).

**הערה:** כאשר אנו בונים פונק', כדאי לשמור את הפקודות בתוכה עם נקודה פסיק בסוף הפקודה, בכדי שלא נראה הדפסה כפולה בעת ההרצה.

## דוגמא לפונק' עם 2 משתנים:

הפונק':

```
function [s,d] = func2(a,b);  
s = a+b;  
d = a-b;
```

תישמר בקובץ נפרד בשם func2.m, הנמצא באותו working directory כמו ה-script file (הקובץ הראשי של התוכנית).  
נריץ את הפונק' כך:

```
[u,v]=func2(10,3)
```

יכניס לתוך u,v את הערכים של 10+3 ו-10-3 כמו שתיכננו בפונק' עצמה. כמו כן, ניתן לבצע דבר דומה למטריצות:

```
[g b]=func2(ones(3),eye(3));
```

תתן את התוצאה:

```
g =  
    2    1    1  
    1    2    1  
    1    1    2  
b =  
    0    1    1  
    1    0    1  
    1    1    0
```

## פונק' ללא ערכים (או ללא קלט)

ניתן ליצור פונק' void למיניהם במאטבלאב. הדבר ייעשה בצורה הבאה:

```
Function hello()  
disp('hello!');
```

ייצור פונק' שכאשר נקרא לה בצורה הבאה:

```
hello()
```

נקבל הדפס של המסך של hello.

## דוגמא נוספת

נרצה ליצור פונק', שתקבל כל מערך ותדע להחזיר את הגודל שלו (ב 2 משתנים), וכמו כן את ממוצע כל האיברים בו.  
נעשה זאת בצורה הבאה:

```
function [x,y,z] = func3(a);  
[x,y] = size(a);  
z = mean(a(:));
```

ניתן לראות כי הערכים  $x, y$  יתקבלו מהרצת הפקודה size על המטריצה שנכניס לפונק' (a).  
נריץ אותה בקובץ הראשי של התוכנית כך:

```
[m n k] = Func3(a);
```

יחזיר לנו את  $m, n$  כגודל המטריצה, ואת  $k$  כממוצע כלל הערכים במטריצה.


## משתנים לוגיים

עד כה הכרנו 2 סוגי משתנים, משתנה נומרי (double, integer, etc.), ותווים (char).  
במאטלאב ניתן לעשות שימוש במשתנים נוספים שהם נקראים משתנים בוליאנים.  
כמו בשפות תוכנות אחרות, יכול לקבל את הערך true או false.

### הגדרת משתנה לוגי חדש

המשתנה יוגדר בצורה הבאה:

```
c = true
```

הדבר יצביע על כך שהמשתנה  $c$  הוא אכן משתנה בוליאני, בעל ערך true. סימנו בחלון ה-  
workspace הסימן ישתנה ל: 

### בדיקה האם המשתנה הוא לוגי בעזרת islogical()

נוכל להשתמש בפקודה הנ"ל בכדי לבדוק האם המשתנה הוא אכן משתנה לוגי.  
נעשה זאת בצורה הבאה:

```
islogical(c)
```

יחזיר לנו את הערך 1 אם הקלט הוא לוגי, ו-0 אם הוא לא לוגי.

### מעבר בין לוגי לנומרי

במידה ונוסיף למשתנה לוגי ערך כלשהו, אנו נקבל ערך נומרי חדש שיורכב מהפעולה האריטמטית  
הנ"ל.

### לדוגמא

```
x = +c
```

ייתן את התוצאה:

```
X = 1
```

כאשר איקס הוא ערך נומרי, ולא לוגי.

כמו כן, ניתן לבצע פעולות אלו על מטריצות וכדומה.



### מטריצות לוגיות

מטריצה המורכב מערכים שהם לוגים, ניתן ליצור בעזרת הפקודות הבאות:

```
A1 = true(2,3)
A2 = false(4,2)
```

יצור מטריצות 2,3 המלאה בערכי true, ומטריצה 4,2 המלאה בערכי false. כמובן שהערכים בתוך המטריצות יהיו לוגים בלבד.

### המרה של משתנה מנומרי ללוגי

כל ערך ששונה מ-0, יינתן לו ייחוס של 1. לאומת זאת מספרים ששווים ל-0 יקבלו את הערך 0 גם כן. במילים פשוטות, כל מספר יהיה true, ורק 0 יהיה false. נעשה זאת בצורה הבאה:

```
D=[0 1 0 113; 5 0 0 8;0 7 6 12;4 0 7 0]
e = logical(d)
```

ייתן לנו בעצם וקטור שבו כל מקום שיש 0 יתחלף ל-false וכל מקום ששונה מ-0 יהיה true.

### יחסים לוגיים

בדומה לשפות תכנות אחרות, קיימים יחסים בין מספרים אשר ניתן להביט עליהם כעל יחסים לוגיים. בצורה הבאה:

```
A1 = 4;
x = 4 >= A1;
```

ה-x יקבל את הערך 1 (ערך לוגי true). במקרה אחר שהתנאי לא יתקיים, x יקבל את הערך false (0).

### סימנים:

>, < - גדול/קטן.

=, ~ = - שווה/לא שווה.

=>, <= - גדול שווה/קטן שווה.

## לדוגמא

```
a1=4>2
a2=2~=4
a3=4>=4+1
a4=(4>=4)+1
```

ייתן לנו את התוצאות:

```
a1 =
    1
a2 =
    1
a3 =
    0
a4 =
    2
```

## דוגמא נוספת

```
a = [1 2 3];
q1 = a==2;
```

נקבל בעצם את הוקטור הבא, שמציין כי הטענה מתקיימת רק באיבר השני של הוקטור a:

```
q1 = 0 1 0
```

**הערה חשובה:** במידה ונשווה בין וקטורים, הם חייבים להיות בממדים שווים!

## דוגמא חשובה

יהיו לנו 2 וקטורים:

```
v = [4 -2 -1 5 0 1 -3 8 2];
w = [0 2 1 -1 0 -2 4 3 2];
```

נניח שנרצה לעשות וקטור  $y$ , שיהיה מורכב אך ורק מהערכים שמקיימים:  $w > v$  (הערך ב-w גדול מהערך באותו מקום ב-v).

נוכל לעשות זאת ב 2 צורות. האחת, הפשוטה והמוכרת:

```
y = w.*(w>v)
```

שכן  $w > v$  ייתן לנו וקטור של 1 ו-0 ואם נכפול אותו ב-w, זה נקבל את הערכים של w במקומות הרלוונטיים ו-0 במקומות האחרים.

כמו כן ניתן לעשות זאת בצורה הבאה:

```
y = w(w>v)
```

הפקודה הנ"ל תתן לנו את הוקטור w ללא האפסים. כך:

```
y = 2 1 4
```

בנוסף, ניתן יהיה לבנות ווקטור אינדקסים שבנוי מהמקומות בהם מתקיים תנאי מסויים!  
נוכל לעשות זאת כך:

```
x = [4]
y = [4, 2, 3, 4, 5, 6, 7, 8]
a = x==y
```

הדבר יציג את a כוקטור שבמקומות 1 ו-4 שלו יש 1.

### פעולות לוגיים (AND/OR/NOT/XOR)

נוכל להשתמש גם בביטויים של or ו-and וכו' במאטלאב. נוכל לעשות זאת בפשטות כאילו מדובר בפעולת כפל/חילוק:

### לדוגמא

```
A=1:3;
b=3:-1:1
C = b(~=3)&(a>=b)
```

התוצאה שתתקבל הינה וקטור c שנראה כך: [0 1 1]

כאשר אנו נבצע בדיקה של מספר רגיל, במידה והוא שונה מאפס הוא יתקבל כ-אחד.  
במידה והוא 0 הוא יהיה 0.  
ולכן הפקודה:

5&-2

יתן לנו אחד.

### פקודות חיפוש

#### Find() - מציאת את המקומות המספרים שהם לא אפס בוקטור.

נשתמש בפקודה בכדי למצוא את האינדקסים של האלמנטים ששונים מ-0 בוקטור כלשהו.  
נשתמש בה כך:

Find(X1)

יהיה:

X1 = [5 0 6 0 4 0]

הפקודה תחזיר לנו וקטור שהוא [1 3 5].

כעת, ניתן להשתמש בפקודה זו גם כדי לשלוף את הערכים שיושבים במקומות הנ"ל:

Numbers = x1(find(x1))

יתן לנו את הערכים באינדקסים הרלוונטיים(שהם לא אפס).

כמו כן ניתן לשלב אלגברה לוגית בשימוש ב-find:

```
Elemt=x(find(x>3))
```

מה שייצור בתוך ה elemt את כלל האינדקסים בוקטור איקס שיותר גדולים מ-3.

### משפטי בקרה

ניתן לחלק ל-2 סוגים של משפטי בקרה.

האחת היא if/switch, שבעזרתו אנו מבצעים את הפעולות הבודדות לאחר החלטה בודדת של המערכת. לאומת זאת קיים גם את ה-for/while, אשר בעזרתם ניתן לבצע קבוצה של פקודות שעומדות בתנאי.

### else/elseif/If

```
If (<condition>)
```

```
פעולות...
```

```
End
```

לדוגמא:

```
a=6
```

```
if a<8
```

```
disp('small')
```

```
end
```

יציג לנו את המילה כיוון שהתנאי מתקיים.

בנוסף, קיים לנו התנאי else, שיכנס לתוקף בתנאי שהתנאי if לא יתקיים.

לדוגמא:

```
a=9
```

```
if a<8
```

```
disp('small')
```

```
else
```

```
disp('big')
```

```
end
```

במקרה הנ"ל יודפס big.

כמו כן ניתן להשתמש ב:

```
elseif
```

בכדי לדמות מצב של if-else חדש שייבדק.

**טיפ חשוב:** ניתן לסדר את העמודות של האיפים בסימונם, ולחיצה על ctrl+i

### לולאת for

פקודת for הינה פקודה לביצוע לולאות כל עוד מתקיים תנאי מסויים.  
נבצע זאת כך:

```
a=0;
v=[4 8 9 2 6];
for i=v
    if i>7
        a=a+i;
    end
end
```

הלולאה תרוץ 5 פעמים (כמספר העמודות בתוך וקטור v), וכל פעם המשתנה i יקבל את הערך הבא בעמודה. במידה ויהיו לנו 2 שורות לדוגמא (מטריצה), הערך i יקבל אליו את העמודה הבאה.

### לולאת while

לולאת while הינה לולאה התמשיך להתקיים, כל עוד התנאי בסיס שקבענו לה מתקיים.  
נפעיל אותה בצורה כזאת:

```
While(Condition)
    StatementList
End
```

### לדוגמא

```
X=0;
While(x<10)
    X=x+1
End
```

בדוגמא זו, התוכנה תוסיף כל פעם 1 ל-x עד שהוא יגיע לספרה 9 (קטן מ-10).

### דוגמא נוספת: ביצוע עצרת בעזרת הלולאה:

```
a=1;
flag=1;
while flag<10
    a=a*flag;
    flag=flag+1;
end
```

בדוגמא זו, מחושבת התוצאה של 9!.

### ביצוע פעולות על מספר איברים

במאטלאב קיימות שתי אפשרויות לביצוע פעולות על מספר איברים (לדוגמא זמן משתנה, או דברים בסגנון).

דרך אחת, היא לבצע זאת בעזרת וקטור שנרכיב מהאיברים שאותם נרצה לחשב, נעשה זאת כך:

```
x=1:1000;  
s=sum(x.^2)
```

הפקודה תרכיב את הוקטור s, כסכום המספרים מ-1 עד 1000 כשהם בחזקת שניים. את אותה פעולה, ניתן לבצע כך:

```
for i=x  
    s1=s1+i^2;  
end  
s1
```

הפעם השתמשנו בלולאת for, אשר עושה בדיקת את אותו דבר.

**הערה:** במאטלאב, נהוג להשתמש בשיטת הווקטור מאשר בלולאות לחישובים דמויי אלו.

### פולינום

ניתן לייצג פולינום במטלב.

מגדירים פולינום ע"י וקטור שורה בצורה רגילה כאילו כל מקדם בוקטור מייצג את המקדם של האיבר הרלוונטי בפולינום.

לדוגמא:

```
%% 8x+5  
p1=[8 5]  
%% 6x^2 - 150  
p2=[6 0 -150]
```

אז מה ההבדל? אין הבדל!

ישנן פונקציות שידועות לעבוד עם פולינומים כאלו (וקטורים של 3 משתנים) ועושה חישובים בפולינום.

### פונקציות פולינום

#### Polyval(p1,x)

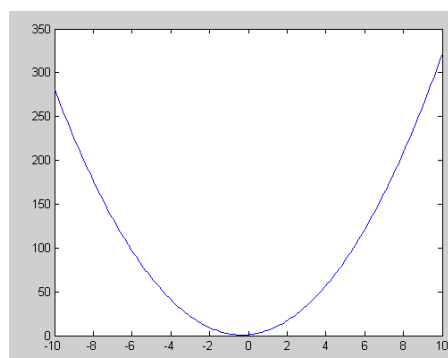
$\text{polyval}(p1, x)$  - פונקציה המחשבת את ערך הפולינום p1 בנקודה x.  
לדוגמא:

```
%% y=3x^2+2x+1  
x=5;  
p1=[3 2 1]  
polyval(p1,x)
```

התוצאה שתוצג הינה: 86 (הפונקציה מציבה את ה-x בפולינום ומחשבת).

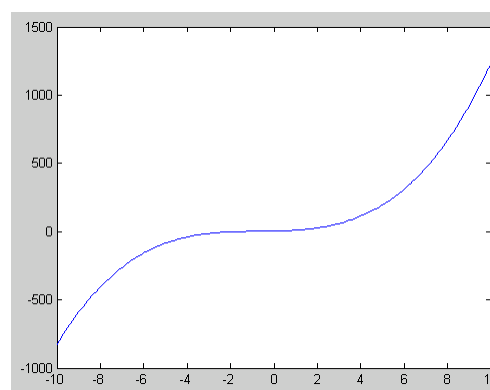
ניתן לעשות את אותו החישוב עבור וקטור  $x$  בעל ערכים מ-10 עד 10 (לטובת הצגת הפונקציה בקטע זה), הפונקציה תחזיר וקטור עם ערך הפולינום בכל ערכי  $x$  בהתאמה.  
לדוגמא:

```
x1=-10:0.1:10;
p1=[3 2 1];
plot(x1,polyval(p1,x1));
```



דוגמא נוספת:

```
p=1:4;
x=-10:0.1:10;
plot(x,polyval(p,x))
```



## חיבור וחיסור בין פולינומים

על מנת לחבר ולחסר בין הפולינומים לא תמיד ניתן להשתמש באופרטור חיבור וחיסור כמו שאנחנו רגילים בין וקטורים. הדבר קורה בגלל שלא תמיד הוקטורים (פולינומים) הם באותו המימד. למשל הפולינום  $x^2 + x + 1 = [1 \ 1 \ 1]$  והפולינום  $x = [1 \ 0]$  אם ננסה לחבר בין 2 הפולינומים בחיבור רגיל ע"י חיבור וקטורים תצא שגיאה כי הוקטורים לא באותו מימד.

תרגיל: כתוב פונקציה שפותרת את הבעיה הנ"ל. כלומר – מבצעת חיבור\חיסור בין 2 פולינומים (גם אם המימד שונה).  
פתרון:

```
function [p]=polyOp (p1,p2,operation)

a=length (p1);
b=length (p2);

if a>b
    p2=[zeros (1,a-b),p2];
elseif b>a
    p1=[zeros (1,b-a),p1];
end

if length (operation)==3
    if (operation=='add')
        p=p1+p2;
    elseif (operation=='sub')
        p=p1-p2;
    else
        p=[]
        disp ('worng string')
    end
else
    p=[]
    disp ('worng string')
end
```

הפתרון בעצם "מרפד" את הוקטור שחסר לו איברים באפסים בכדי להביא אותו למצב של פולינום 3 איברים).

## מכפלה בין פולינומים (conv)

ניתן לכפול בין פולינומים בעזרת הפונקציה `conv(u,v)`. נעשה זאת כאילו והיינו כפלים 2 פולינומים בסוגריים בצורה רגילה. לדוגמא:

```
% u=x^2+2x+3
% v= 4x^2+5x+6
u=[1 2 3];
v=[4 5 6];
a = conv(u,v)
% The Result is: a = [4 13 28 27 18]
% a = 4x^4+13x^3+28x^2+27x+18
```



### חילוק בין פולינומים (deconv)

את חילוק הפולינומים שהיינו עושים בעמודה (חילוק ארוך), המאטלאב עושה בעזרת הפונקציה:

`deconv(u,v)`

הפונקציה תחלק את  $u$  ב- $v$ . ותחזיר 2 ערכים, כאשר הראשון הוא התוצאה (פולינום), והשני הוא השארית.

אם נשתמש בדוגמא הקודמת, ד"י ברור שנקבל:

`[q,r] = deconv(a,u)`

יחזיר לנו את  $v$  (כוקטור) בתוך המשתנה  $q$ , ואילו 0 בתוך השארית  $r$ .

### נגזרת של פולינום (polyder)

ניתן גם לגזור פולינום בצורה הכי פשוטה, המאטלאב עושה זאת בעזרת הפקודה:

`polyder(u);`

במידה ונבצע פקודה זו על הוקטור שכבר יש לנו ( $u = [1 \ 2 \ 3]$ ), זה יהיה דומה ללגזור  $x^2+2x+3$ . והתוצאה תהיה הוקטור  $[2 \ 2]$  שבפועל יציין את הפולינום  $2x+2$ .

### שורשים של הפולינום (פתרון משוואה אופיינית)

ניתן גם לפתור משוואות פולינום בעזרת המאטלאב, נבצע זאת כך:

`roots(u)`

התוצאה שנקבל תהייה:

$-1.0000 + 1.7321i$

$-1.0000 - 1.7321i$

**הערה חשובה:** במידה וניהייה בפורמט short, לא נראה את האיברים המדומים (מספרים מרוכבים).

### יצירת פולינום העובר דרך נקודות מסויימות (polyfit)

ניתן ליצור פולינום העובר דרך נקודות מסויימות, וכמו כן לצייר את הפונקציה עצמה. נעשה זאת בעזרת הפקודה:

`polyfit(x,y,n)`

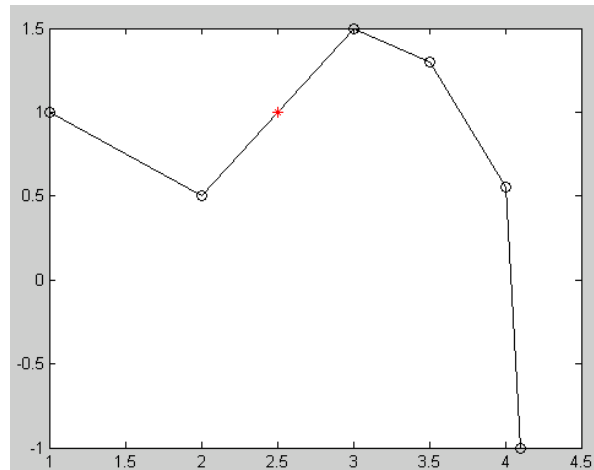
פונקציה שיוצרת פולינום מסדר  $n$  העובר בכל הנקודות שמגדירים בוקטורים  $x$  ו- $y$ . כמובן שעל הוקטורים  $x$  ו- $y$  להיות בעלי אותו מימד.

### חישוב ערך הפונקציה בנקודות ביניים (interp1)

נניח כי אנו יוצרים פולינום ע"י מספר נקודות. כעת ברצוננו לחשב את ערך הפונקציה שיצרנו בנקודות ביניים (לא בנקודות שהגדירה את הפולינום).  
לצורך כך ניתן להשתמש ב `interp1`.

דוגמא:

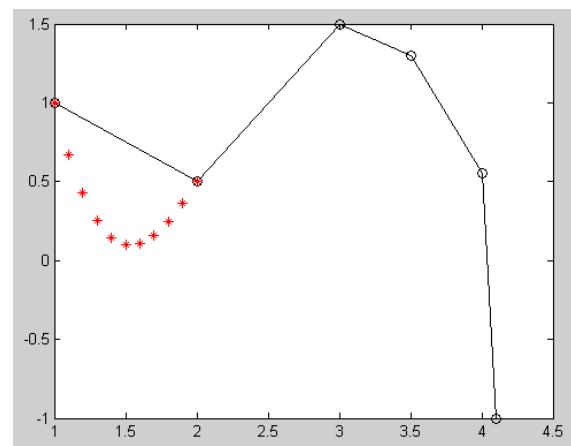
```
x=[1 2 3 3.5 4 4.1]
y=[1 0.5 1.5 1.3 0.55 -1]
xi=2.5
yi=interp1(x,y,xi)
plot(x,y,'ko-')
hold on
plot(xi,yi,'r*')
```



את הנקודה ניתן לחשב בכמה דרכים. ברירת מחדל – לינארי (חיבור קו בין כל 2 נקודות שמגדירות את הפולינום – כמו בדוגמא הנ"ל).

דוגמא מספיק (בדוגמא זו הנקודות מחושבות ע"י פרבולה):

```
%% y1=interp1(x,y,xi,'method')
x=[1 2 3 3.5 4 4.1]
y=[1 0.5 1.5 1.3 0.55 -1]
plot(x,y,'ko-')
hold on
xi1=1:0.1:2
y1=interp1(x,y,xi1,'spline')
plot(xi1,y1,'r*')
```



### ציור פונקציה באמצעות מחרוזת (ציור פונקציות בטווח)

הפונקציה מאפשרת לצייר פונקציה בטווח מסוים כאשר אנו מגדירים את הפונקציה כ-string.

### ציור הפונקציה בתחום מסוים (fplot).

```
fplot(function string, [range], <type of line>)
```

לדוגמא

```
F1 = 'x^3-exp(x)'  
fplot(f1, [-2 2], 'r^--')  
grid on
```

כאשר ה-grid נותן לנו את המרשתת מאחור בגרף (צירים וכו').

### ציור הפונקציה בטווח ברירת מחדל (ezplot).

```
ezplot(f1)
```

הפונקציה יכולה לצייר לנו את הפונקציה ללא קביעת טווח מסוים. ובתצורה רגילה היא תצייר פונקציה בין  $-2\pi$  לבין  $2\pi$ . כמובן שניתן להכניס לה טווח בכל מקרה אבל הדבר לא יעיל. כמו כן, בפונקציה הנ"ל לא ניתן לערוך את התצורה בה הפונקציה תצויר. בנוסף, נוסף label למעלה מעל הגרף ע"י הפונקציה.

### מציאת נקודות חיתוך של פונקציה (fzero)

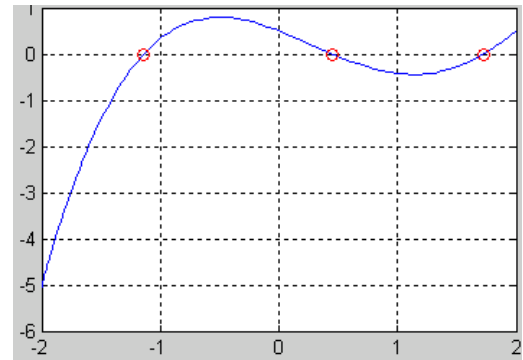
בעזרת פונקציה זו נוכל לדעת את נקודת החיתוך של הפונקציות, למעשה הדבר מבטא  $y=0$  ומחשב את ה-x (החיתוך של הפונקציה עם ציר ה-x).

```
fzero(<function string>, <point near the "x axis">)
```

כאשר יש צורך להכניס את הסביבה בה נרצה למצוא את נקודת החיתוך.  
לדוגמא:

```
f1='0.8*x^3-exp(x)+1.5'  
r1=fzero(f1,-1)  
r2=fzero(f1,0.5)  
r3=fzero(f1,1.5)  
hold on  
plot([r1 r2 r3], [0 0 0], 'ro')
```

הפונקציה בעצם "תסמן" לנו את נקודות החיתוך בסביבות שצינו ב-fzero. התוצאה תהייה:



### מציאת ערכי מקסימום/מינימום של הפונקציה

ניתן למצוא נקודות מינימום ומקסימום בפונקציה באזור סביבה מסוימת, ע"י שימוש במאטלאב, בצורה הבאה:

```
[Xmin Ymin] = fminsearch(f1,x)
```

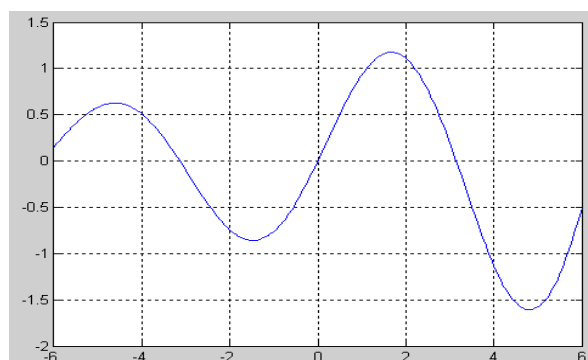
כמו כן, ניתן למצוא את המינימום הראשון שניתן לאתר בטווח מסוים.

```
[Xmin Ymin] = fminbnd(f1,a,b)
```

### לדוגמא

```
h1='sin(x)*exp(0.1*x)'  
fplot(h1, [-6 6])  
grid on
```

לאחר שנפעיל את הפונקציה, נראה כי בציור קיימות 2 נקודות מינימום:

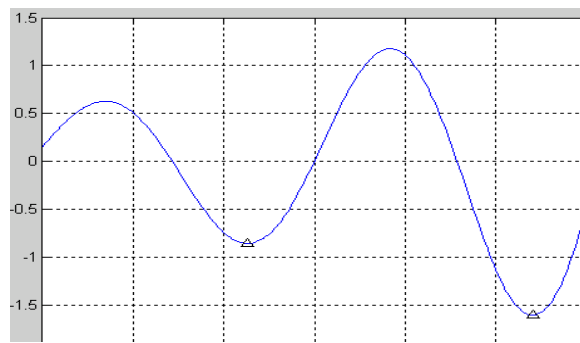


אחת בסביבת 4 והשנייה בסביבת -2.

ובעזרת הפונקציות שלמדנו ניתן למצוא נקודות מינימום אלו בדיוק:

```
[xmin1 ymin1] = fminsearch(h1,-2)  
[xmin2 ymin2] = fminsearch(h1,4)  
hold on;  
plot([xmin1 xmin2], [ymin1 ymin2], '^k')
```

התוצאה תהייה:



### מציאת ערך מקסימום של פונקציה

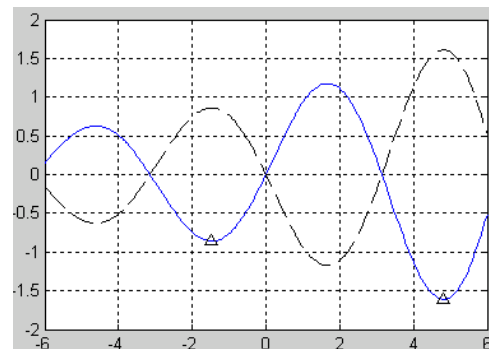
במאטלאב לא קיימת אפשרות לחשב ערך מקסימום של פונקציה, ולכן אנו נהפוך את הגרף (ניצור תמונת מראה שלו) בכדי למצוא את ערכי המקסימום ע"י שימוש בפונקציית המינימום. נעשה זאת בצורה הבאה:

$h1m = [-(' h1 ')]$

בפועל, אנחנו מבטאים מחרוזת חדשה, שבה אנחנו לוקחים את הפונקציה הקיימת שלנו, ומכניסים אותה לתוך סוגריים עם הסימן מינוס לפני. התוצאה תהייה:

$-(\sin(x)*\exp(0.1*x))$

בציור הבא ניתן לראות את 2 הפונקציות, המקורית (צבע כחול) וההפוכה שלה (בצבע ירוק).



ובכך, נוכל לקבל את אותה הפונקציה, אבל הפוך. ואז כשנחפש את ערכי המינימום שלה, אנו בעצם נמצא את ערכי המקסימום של הפונקציה המקורית. נעשה זאת כך:

$[xmax1 \ ymax1] = fminsearch(h1m, -6)$

$ymax1 = -ymax1$

בפועל, ה-x נשאר אותו איקס, אבל ה-ymax, יהיה שווה ל(-ymax), בכדי לפצות על המינוס של הפונקציה ההופכית.

### קירוב לנגזרת של פונקציות

כל נגזרת של פונקציה ניתן לתאר גם כ:

$$\lim_{n \rightarrow 0} \frac{f(x+n) - f(x)}{(x+n) - x}$$

למעשה זאת הנסחה של חישוב נגזרת לפי הגדרה.

במאטלאב, ניתן לעשות זאת בעזרת הפונקציה:

`diff(<function>,<how many times>)`

פונקציה זו, תחשב רק את החלק העליון של המשוואה (למעשה, מדובר בהפרשים בין אלמנטים סמוכים).

כאשר נוכל להגדיר כמה פעמים נבצע את החיסור הנ"ל.  
לדוגמא

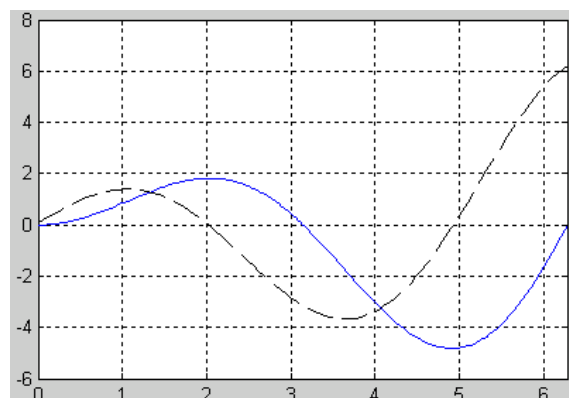
```
x=(1:5).^2
dx=diff(x)
dxx=diff(dx)
dxx=diff(x,2)
fplot('x*sin(x)', [0,2*pi])
grid on
x=linspace(0,2*pi);
dy_dx=diff(x.*sin(x))./diff(x)
```

חשוב לזכור, שבכל `diff`, אנו בעצם "מורידים" את מימד הווקטור עליו אנו עובדים (מקצרים אותו באיבר).

ולכן, נצטרך "לתקן" את הקיצור הנ"ל של הווקטור, כדי שבגרף יהיה לנו כנגד מה לצייר את הפונקציה, נעשה זאת ע"י הפעולה הבאה:

```
xnew=linspace(0,2*pi,99);
hold on
plot(xnew,dy_dx,'k--')
```

התוצאה, תהייה בעצם הנגזרת של הפונקציה (מקווקו), ובנוסף הציור של הפונקציה עצמה (שעשינו בחלק הראשון):



## קירוב לאינטגרל של פונקציות

### חישוב אינטגרל (trapz)

ניתן לחשב את השטח שמתחת לגרף באמצעות טרפזים, הפונקציה תחבר ליניארית בין נקודות על הפונקציה, ותיצור טרפז מול ציר ה-X. את הטרפזים האלו הפונקציה תסכום בכדי לחשוב שטח. ככל שיותר נקודות על הפונקציה יהיו, האינטגרל יהיה מדויק יותר.

```
trapz(x,y)
```

### לדוגמא

```
x=linspace(0, pi, 20);  
y=sin(x);  
z=trapz(x,y)
```

כאשר השטח שמתחת לגרף יינתן בתוך משתנה z.

### חישוב אינטגרל לפונקציית מחרוזת

ניתן לחשב גם פונקציות שניתנות כמחרוזות, נעשה זאת בעזרת הפקודה:

```
quad('sin(x)', 0, pi)  
quadl('sin(x)', 0, pi)
```

קיים הבדל ביניהם אך הוא לא קריטי.

למרות שהפונקציה הוא מחרוזת וזה לא נראה כמו חישוב וקטור, ההגדרה היא כן של חישובים וקטורים, ולכן נצטרך להוסיף את אופרטור הנקודה **בתוך המחרוזת** בכדי לבצע את החישוב כמו שצריך.

### דוגמא

```
%trapz use  
x = [0:5];  
z1 = trapz(1./(0.8*x.^2+0.5*x+2), x);  
  
%quad use  
f = '1./(0.8*x.^2+0.5*x+2)'  
z = quad(f, 0, 5)
```

### תרגיל לדוגמא - (מציאת נקודות חיתוך, וחישוב שטח ביניהם).

חישוב שטח של פונקציה (מציאת נקודות חיתוך, וחישוב שטח ביניהם).

```
sumi = 0;
x = [4 6 10 12];
ezplot('sqrt(x)+5*sin(x)', [1:14])
f = 'sqrt(x) + 5*sin(x)';
j=1;

for i=x
    r(j) = fzero(f,i);
    j = j+1;
end

sumi = sumi + quad(f,1,r(1)) + quad(f,r(4),14);

for k=[1:3]
    sumi = sumi + quad(f,r(k),r(k+1));
end
grid on;
sumi
```