Югозападен университет
„Неофит Рилски”

Въведение в MATLAB

Иван Тренчев,

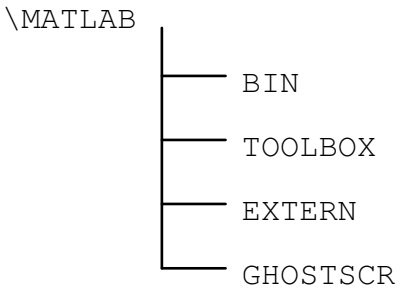
Петър Миланов

ВЪВЕДЕНИЕ В МАТЛАБ

I. Основни сведения за MatLab

MatLab е диалогова програмна система за провеждане на научно-технически пресмятания. Тя интегрира в себе си възможностите за аналитични преобразувания, числени пресмятания и графично представяне на получените резултати. Ориентирана е към работа с масиви от данни – вектори, матрици, многомерни масиви, масиви от клетки и масиви от записи. От тук идва и наименованието MatLab - MATrix LABoratory. Това позволява с един единствен оператор да се извършват едновременно действия над всички елементи на масива, без необходимостта от организиране на цикли. В системата са вградени функции, решаващи основни задачи от линейната алгебра, числения анализ, обработка на експериментални данни, двумерна и тримерна графика, анимация и др.

MatLab се състои от две основни части – ядро и допълнителни компоненти. Ядрото представлява съвкупност от програми, имащи различно предназначение и разположение в различни каталози (директории). Структурата на директориите на MatLab за Windows е:



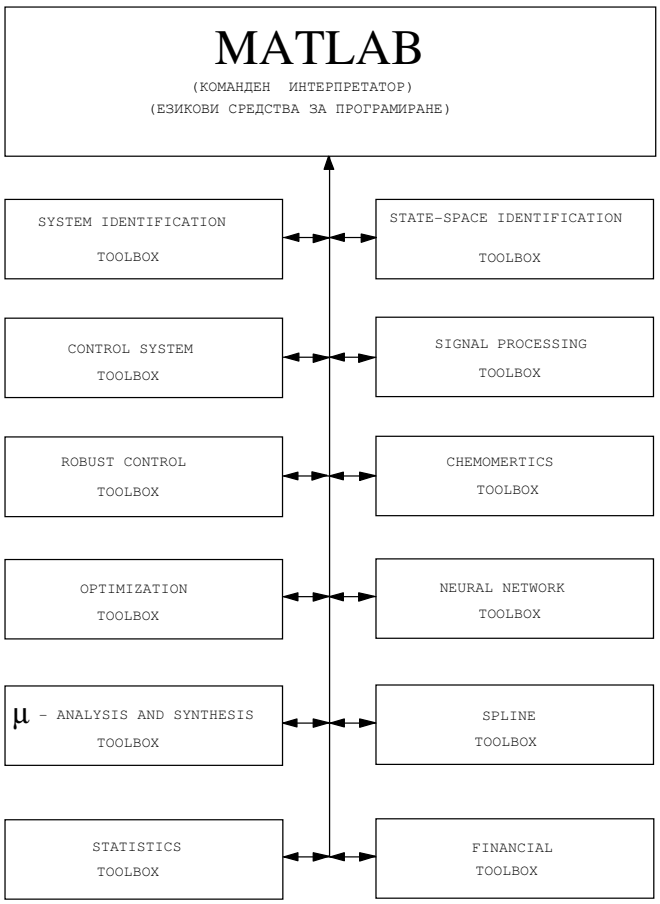
Допълнителните компоненти служат за разширяване на възможностите на MatLab. Всеки компонент представлява съвкупност от функции, написани с езиковите средства на MatLab. Функциите са записани в ASCII код, имат разширение „.m" и затова се наричат накратко „m-файлове".

Кратко описание на видовете компоненти и техните възможности:

- **Компонент за Обработване на сигнали (Signal Processing Toolbox**). Той съдържа функции за обработване на едномерни и двумерни временни редове.
- **Компонент Системи за управление (Control System Toolbox).** Той съдържа функции реализиращи основни процедури от теорията на управлението и системите.
- **Компонент Идентификация на обекти за управление (System Identification Toolbox).** Компонентът включва функции за построяване и анализ на параметрични динамични модели и идентификация на обектите за управление на базата на входно-изходни данни или временни редове.
- **Компонент Оптимизация (Optimization Toolbox).** Той е съвкупност от програми за оптимизация на произволни линейни или нелинейни функции, за линейно и квадратично програмиране и за решаване на линейни уравнения.

- **Компонент Невронни мрежи (Neural Network Toolbox).** Съдържа функции за проектиране и симулиране на невронни мрежи.
- **Компонент Сплайн функци (Slpine Toolbox).** Изграден е от програми за работа със сплайн-функции.
- **Компонент Робастно управление (Robus Control Toolbox**).
- **Компонент μ-Анализ и синтез (μ-Analysis and Synthesis Toolbox).** Състои се от m-файлове за анализ и проектиране на линейни робастни системи за управление с използване на методите за μ-анализ и синтез.
- **Компонент Статистика (Statistical Toolbox).** Съдържа програми за статистически анализ на данни.
- **Компонент Идентификация в пространство на състоянието (State – Space Identification Toolbox).**
- **Компонент Финанси (Financial Toolbox**).
- **Компонент Химически изчисления (Chemometrics Toolbox).** Този компонент съдържа програми за извършване на изчисления в областта на химическата наука и технология.

Връзките между компонентите и ядрото са двупосочни. Директни връзки между компонентите не съществуват тъй като всички операции се извършват под управлението на командния интерпретатор, а компонентите съдържат m-файлове. Обобщена схема на основното ядро и допълнителните компоненти е дадено на фиг. 1



MatLab има вграден програмен език от високо ниво, позволяващ работа на системата не само в режим на калкулатор, но и в програмен режим.

MatLab работи в два режима: команден и програмен. Това деление обаче е условно, защото в много случаи не може да се направи ясно разграничаване на двата режима. Изпълнението на една програма писана с езиковите средства на MatLab се извършва чрез извикване с команда, представляваща нейното име. Командите на MatLab са два вида: вградени и външни. Вградените команди са част от кода на интерпретатора, а външните са програми, записани в ASII код като текстови файлове, които задължително имат разширение „m”.

Две команди имат особено значение за MatLab. Едната е demo. Тя е предназначена за бързо демонстриране на някои възможности на MatLab при решаване на задачи, които се срещат често в научните и инженерните изследвания. Избира се в диалогов режим коя задача да бъде демонстрирана. Втората особена команда е help. Нейната задача е да даде на потребителя помощ за предназначението и начина на използване на всяка от командите и функциите на MatLab, достъпни за интерпретатора, в зависимост от инсталираните компоненти на машината, върху която се работи.

Най-използваните области на приложение на MatLab са:

- Математически и компютърни изчисления;
- Разработка на алгоритми;
- Изчислителни компютърни експерименти;
- Имитационно модулиране;
- Компютърен анализ на данни;
- Изследване и визуализация на данните;
- Научна и инженерна графика;

Изработване на приложения включващи и графичен интерфейс на потребителя.Като средство за компютърно моделиране, което обезпечава изследвания практически във всички известни области на науката и техниката, MatLab дава възможност ефективно да се съчетават двата основни подхода към създаването на модули: аналитичен и имитационен.

Интерфейсът на MATLAB се състои от 5 прозореца (фиг. 2):

Команден прозорец – Command Window

Това е основният прозорец, в който въвеждаме командите след промпта ">>" и получаваме числените резултати, след като натиснем клавиша Enter . Графическите резултати се извеждат в отделен прозорец.

При въвеждане на командите са в сила следните правила:

1. Ако командата завършва със символа ";", резултатите от нейното изпълнение не се извеждат. Това се използва за потискане извеждането на междинните резултати и на много дълги масиви от данни;

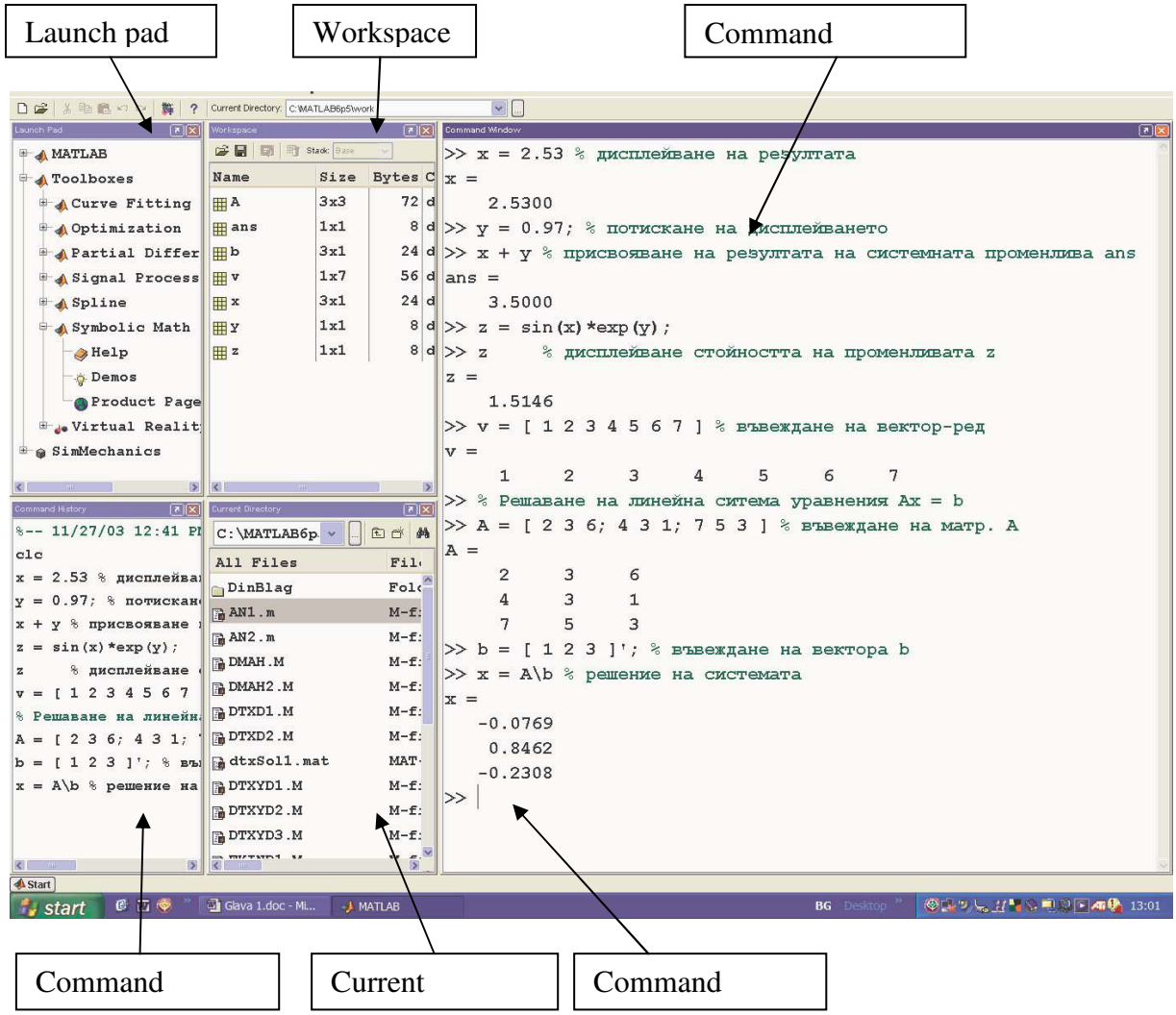
2. Ако командата не завършва със символа ";", резултатите се извеждат;

3. Наличието или отсъствието на символа ";" в края на графични команди не оказва никакво влияние върху резултатите от изпълнението им;

4. Прекалено дългите команди могат да се пренасят на нов ред с помощта на три точки "...", разположени в мястото на пренасяне;

5. Ако се пресмята даден израз, без да присвояваме резултата на

някаква променлива, той автоматично се присвоява на системната променлива "ans" (от answer – отговор);



Фиг. 2

6. Въведените в дадена работна сесия команди се запаметяват от системата. Те могат да бъдат извикани на командния ред (до промпта) с помощта на клавиша ↑ , коригирани и изпълнявани с натискане на Enter . Така се пести време и усилия при последователно въвеждане на сложни изрази, които се различават

незначително един от друг.

7. Няколко команди могат да се въведат на един ред. При това за разделители се използват или запетайки "**,**" (не се потиска извеждането на резултатите) или символа "**;**", потискащ

извеждането на резултата от предходната команда.

8. Символът процент "**%**" се използва за начало на коментар.

Стартов бележник – Launch Pad

Този прозорец ви дава бърз достъп до някои важни инструменти, help-информация и демонстрационни примери за MATLAB и всички инсталирани на компютъра ви Toolbox-ове. Освен това, ако имате Internet връзка, може да щракнете върху Product Page (Web), за да

получите последната информация за съответния продукт от Web сайта на фирмата Math Works, създател на MATLAB.

Работно пространство – Workspace

Всички променливи, които сте дефинирали през текущата сесия, се съхраняват в така нареченото *работно пространство* (Workspace). В прозореца Workspace се извежда пълна информация за тези променливи: име, размерност, големина и клас. Обърнете внимание, че дори и скаларните величини в MATLAB се разглеждат като матрици с размерност 1x1. Цялото работно пространство може да бъде записано във файл с разширение .mat, като изберем от менюто File–>Save Workspace As…Така запазените данни могат да се използват в следваща сесия, като заредим .mat файла с командата File–>Import Data .

Информация за работното пространство, коригиране и запазване на данните във файл, може да става и от командния прозорец.

История на командите – Command History

Системата MATLAB запаметява всички въведени команди не само от текущата, но и от предишни сесии. Тези команди се извеждат в прозореца Command History.

Текуща директория – Current Directory

В този прозорец се извеждат всички файлове от текущата

директория. До тази директория системата има директен достъп, дори и ако не е обявена за достъпна с командата **path**. В нея обикновено се записват и създаваните от вас файлове.

В горния ред на прозореца се намират:

1 – Поле за задаване на текущата директория;

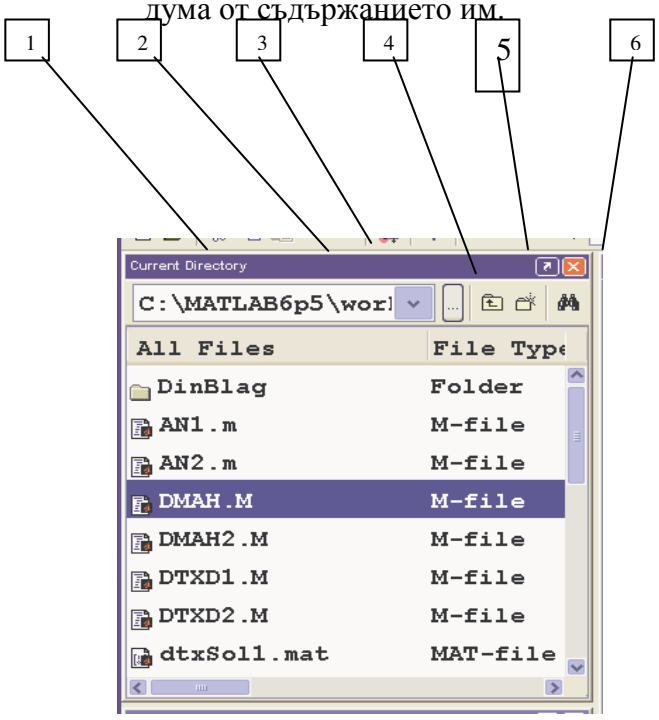
2 – Бутон за извеждане на падащ списък на последните посетени директории;

3 – Бутон за достъп до произволна директория;

4 – Бутон за преминаване към родителската директория (едно ниво нагоре);

5 – Бутон за създаване на нова директория, разположена под текущата;

6 – Бутон за търсене на файлове по ключова дума от съдържанието им.



>> which for

C:\MATLAB7\toolbox\matlab\lang\for.m

```
>> which mean
C:\MATLAB7\toolbox\matlab\datafun\mean.m
>> which mymfile
'mymfile' not found.
```

Ако Вие мислите, че може да създадете дубликат файл, Вие можете да използвате командата, *filename-all* , която да създаде списък на всички файлове с това *filename.*

```
>> which mean -all
C:\MATLAB7\toolbox\matlab\datafun\mean.m
C:\MATLAB7\toolbox\matlab\timeseries\@timeseries\mean.m   % timeseries method
C:\MATLAB7\toolbox\ftseries\ftseries\@fints\mean.m        %fints method
```

1.2.1. ; оператор

Матлаб изисква *(;)* в края на реда за да предотвратява излизането. Точката и запетаята инструктира Матлаб да продължи реда без да връща нищо към командния прозорец. За да получите усет за ефекта *(;)* проследете резултата от тези две команди (забележи резултатите на втората са били съкратени).

```
» x=ones(10,1) ;
» x=ones(10,1)
```

1 1 1

Това е добра идея да потиснеш излизането от твоите команди. В определени случаи, такива като преглед или изучаване на функциите на командата, то може да бъде полезно да ги отстраним докато твоят код функционира правилно.

1.2.2 Коментари

Писането на коментари е основна практика. Коментарите помагат да проследим, какво сте извършили и са полезни, ако някога имате нужда да покажете твоята програма на други. В Матлаб символът за % се използва за обозначаването на коментари. Матлаб ще се спре вършенето на нещо на този ред в дясно от символа % и ще прехвърли на следващия ред. За нещастие Матлаб не подкрепя блокови коментари, така че ти трябва да използваш символът % пред всеки ред.

```
7o This is the start of a
7o comment block.
7o Every line must have a %
°/o symbol before the first text
```

1.2.3 Многоточие

…. е специален израз, където може да бъде използван за продължение на дълго изречение, което не е лесно да се изрази на един ред. **...** инструктира Матлаб да свърже следващия ред със сегашния ред, когато процесира. То помага да подобри разчитането на кода и Вие ще го намерите за полезен.

Тези две изречения са идентични на Матлаб.

```
x = 7;
x = x + x * x - x  + exp(x)  / log(x)  * sqrt(2*pi);

x = 7;
x = x + x * x - x   ...
+ exp(x)  / log(x)  * sqrt(2*pi);
```

1.3 Помощ

Матлаб има разширена система, която е в наличност, както в командния прозорец, така и в отделния лист (*browser*). Помощта базирана в него е изобщо по-пълна, по-завършена и има допълнително предимство дето е индексирано и номерирано.

Два типа помощ ги има на командния ред: инструменти и функция. Инструментите помагат за създаване на списък от налични функции в инструментите. Може да бъде извикано чрез **help** *toolbox,* където *toolbox* е името на инструмента в Матлаб(например **stats, optim, и т.н.**).**. help** , без втория аргумент ще създаде списък на инструментите.

Специфичната функция помощ може да бъде достигната чрез извикване на **help** *function*, help **mean**.

Страницата за помощ може да бъде достигната и чрез натискане на бутона F1, избиращ **Help>Full Product Family Help** на върха на командния прозорец или въвеждане на **help browser** в командния прозорец. Вие можете да стигнете директно в документа на някоя функция в help browser чрез въвеждане на *doc function* в командния прозорец, например *doc mean*.

1.4 Demos

Матлаб съдържа разширена секция от *demos* за да се достигне списъка на наличните *demos,* като просто се въведе **demo** в командния прозорец.

1. Основен (базов) вход

Матлаб не изисква никакво управление на паметта и различните неща могат да бъдат въведени без настройка. Общата форма на изразяване в Матлаб е:

VariableName = Expression.

и изразяванията от дясно различни към тези от ляво. Например:

x = 1;

x= y;

x = some_function(y);

са всичките валидни обозначения на *x*. Първото обозначение 1 за *x*, второто обозначава стойността на другата променлива, *y*, към *x* и третото обозначава изхода на *some_function (y*) към *x*. Обозначаването на променливата към друга, обозначава стойността на тази променлива, а не самата променлива. При това в редовете *y= 1; x= y*, каквито и да са промени на *y* няма да се отразят на стойността на *x*.

```
» y = 1;
» x = y;
» x
x =
     1
» y = 2;
» x
x =
     1


» y
y =
     2
```

2.1. Имена на променливи

Имената на променливата могат да вземат много форми, макар че те могат само да съдържат числа, букви (както по-горни и по-долни) и подчертавания*(_)*. Те трябва да започват с буква и са *CaSe SeNsItIve*. Например:

```
x
X
```

X_1
x_1
dell
dell_returns

са законови и индивидуални имена на променливи, докато останалите на са.

x:
1X
X-1
_x

2.2. Входни вектори

всичките данни в Матлаб са матрици по конструкция, даже ако те са: *1* към *1* (скалар, мащаб), *K* към *1* или *1* към *K* (вектори).Векторите както в редица (*1* към *K*), така и в колона (*K* към *1*) могат да бъдат въведени директно в командния прозорец. Матлабовото обозначение:

x = [1 2 3 4 5]
е въведено в Матлаб по естествен начин.

» x=[1 2 3 4 5] ;
По – горния вход [*and*] са запазени символи матлаб, които са преведени като *begin array* (начално подреждане) и *end array* (крайно подреждане) съответно. Колонен вектор,

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

има малко по-малко интуитивна структура в Матлаб.

» x=[1; 2; 3; 4; 5];
Когато вътре в подреждането, ***;*** означава *нов ред*.

2.3 Входни матрици

Матриците са прост колонен вектор на ред вектори. Например за да въведем

$$x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

в Матлаб въвеждаме матрица на един ред по едно време, отелени редовете с ***;***:

» x=[1 2 3; 4 5 6; 7 8 9];

2.4 Подреждания на по-високо ниво

Матлаб е способен да работи с *N* размерни подреждания, където *N* може да бъде много голямо число (до около 30), зависещо от размера на всеки матричен размер (ниво).Мащабните вектори и матрици, от по-високо ниво на подреждане може да бъде (нивото) конструирано чрез извикване на функции и не могат да бъдат директно локализирани (пример: *zeros* (2,2,2)) . по-високото ниво на подреждане може да бъде полезно за проследяване на матрични стойностни функции през времето (условно).

2.5 Празни матрици

Има една необичайна матрица, която си струва споменаването Празната матрица е просто матрица (вектор), която няма никакви елементи *x = [ ];*.Празните матрици могат да се върнат от функциите в определени случай (ако някои критерий не отговаря) и може да предизвика проблеми.Те служат за полезни цели. Първо те могат да бъдат използвани за построяване на “мързелив ” вектор, използващ повторна верига. Например:

```
» x=[]
x =
     [ ]
» x =[x;1]
x=
     1
» x=[x;2]
x=
     1
     2
» x=[x;3]
x=
     1
     2
     3
```

Второ те са необходими за извикването на някои функции, когато Вие се нуждаете само да увеличите входовете, но не и за да ги посочите всичките. Например *some_function(x,[ ]q y).*

2.6 Свързване

Свързването е процес чрез който един вектор или матрица се добавя към друг. Както хоризонталните така и вертикалните връзки са възможни. Например да предположим

$$x = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \text{ and } y = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

а ти желаеш да изградиш

$$z = \begin{bmatrix} x \\ y \end{bmatrix}.$$

Това може да бъде постигнато чрез третиране на *x* и *y* като елементи от нова матрица.

```
>> x=[ 1 2; 3 4];
>> y=[ 5 6 ; 7 8];
```

z може да бъде дефинирано по естествен начин;

```
>> z=[x; y];
```

Това е пример за вертикално свързване. *x* и *y* могат да бъдат хоризонтално свързани по подобен начин.

```
>> z=[x y];
```

Забележка: Свързването е точно като използването на блок-матрични форми в стандартната матрична алгебра.

2.7 Елементи за достъп в матрицата

Щом като Вие имате данни за вектор или матрица, то е важно да можете да имате достъп до отделните елементи. Матлаб съхранява матрици във форма позната като главна колона (*column major*). Това означава, че елементите са индексирани от или чрез първо отброяване по редици, а после напряко по колоните. Например матрицата:

$$x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

първият елемент на *x* е *1*, вторият е *4*, третият е *7*, четвъртият е *2* и т.н.

Елементите могат да бъдат достъпни чрез номер на елемента използващ интервал (*x (#)*). Дефинираната матрица *x = [1 2 3; 4 5 6; 7 8 9];,* елементите на *x* могат да бъдат достигнати

```
>> x(1)
ans =
    1
>> x(2)
ans =

    4
>> x(3)
ans =
    7
>> x(4)
ans =
    2
>> x(5)
ans =
    5
```

Единичното индексиране работи добре, ако *x* е вектор, в чийто случай индекса съответства директно на реда на елементите на *x*. Обаче в матричния случай единичния индекс е объркващ и обременен. За щастие двойният индекс на матриците е възможен при използването на нотацията (записван) .

```
>> x(1,1)
ans =
    1
>> x(1,2)
ans =
    2
>>x(1,3)
ans =
    3
>> x(2,1)
ans =
    4
>> x(3,3)
ans =
    9
```

Матрици с по-големи размери могат също да бъдат достъпни като използваме интервал *(x(#, #, #)*). Например *x(1, 2, 3)* би върнал елемента в първата редица на втората колона на третия панел от триизмерната матрица *x*.

Колонният оператор *(:)* играе специална роля при достигане на елементите. Когато се използва той, се интерпретира като "всички елементи в това измерение". Например, *x (:, 1)*, се интерпретира като всички елементи от матрицата *x* в колона 1. Подобно *x (2, :)* интерпретира като всички елементи на *x* в редица 2. Удвоеният *(:)* създава всички елементи на оригиналната матрица; естествен, x *(: , :)* връща *x*. Накрая Вие можете да използвате вектори за достъп на елементи на x. Например, *x ([1 2] , [1 2]),* ще върне елементите на *x* в редици *1* и *2*, и колони *1* и *2*, докато *x ([1 2] , :)* ще върне всички колони от редовете *1* и *2* на *x*.

```
>> x(1,:)
ans =
     1     2     3
>> x(2,:)
ans =
    2  5  8
```

```
>> x(:,:)
ans =
     1     2     3
     4     5     6
     7     8     9
>> x
ans =
     1     2     3
     4     5     6
     7     8     9
>> x([1 2],[1 2])
ans =
      1     2
      4     5
>> x([1 3],[2 3])
ans =
      2     3
      8     9
>> x([1 3],:)
ans =
     1     2     3
     7     8     9
```

2.8 Извикване на функции

Извикването на функции има малко по различно съгласуване от другите изрази в Матлаб. Най-голяма разлика можем да видим в повече от един вход и да върне повече от един изход.Структурата на извикване на функция е *[out1, out2, out3,…] = function [in2, in3,…]*. Важните аспекти на тази структура са:

ако е необходим само един изход, скобите *([ ])* са опции, (например, *y=mean(x)*).

ако се изисква множество изходи, изходите трябва да бъдат капсулирани (поставени) в скоби (например*, [y, index] = min(x))*..

броят на изходните променливи определя колко изхода ще бъдат върнати. Питането за повече изходи, отколкото функцията ще резултира е грешка.

както входовете така и изходите трябва да бъдат разделени със запетайки *(,).*

входовете могат да бъдат резултат на други функции, докато функцията върне един изход. Например, *(mean (var (x)*).

входовете могат да съдържат само избрани елементи от матрицата или вектора. Например, *(mean (x ([1 2] , [1 2]))).*

Употребата на извикване на функции ще бъде изяснена, когато те са описани в този документ.

2. Въвеждане и съхраняване на данни

Първото истинско предизвикателство, с което се сблъсквате е получаването на актуални данни във и вън от Матлаб.

3.1 Въвеждане на данни в Матлаб

Въвеждането на данни в Матлаб се простира от трудно до много трудно, в зависимост от данните. Първо няколко общи положения:

файлът, който въвеждате би трябвало да съдържа само цифри. Никакъв текст и никакви формули.

използвайте друга програма, за да манипулирате данните преди да ги въведете . Например *Excel*.

всяка колона трябва да съдържа една променлива.

датите трябва да бъдат въведени като число чрез първо форматиране на колоните, като *Excel* дати, а после да се реформатират като числа (дати с базова година 1900).

3.2 Четене на Excel файлове

Данните в *Excel* листовете могат да бъдат въведени с използването на функцията *xlsread*. Придружавайки този комплект от бележки е *excel* файл, *deciles.xls*, който съдържа връщанията за 10 CRSP (десети) от 1 Януари 2000 до 31 Декември 2004. Първата колона съдържа датите, докато колоните от 2 до 11 съдържат порт фолио (сбора от) връщания десета 1 до десета 10 съответно.За да заредиш данните в Матлаб използувайте командата

```
>> data = xlsread('deciles.xls');
```

Тази команда ще чете данните в лист 1 на файла *deciles.xls* и да го отбележи в данните в Матлаб. *xlsread* може да даде номер или число на други ситуации, включително четене на листа различно от *лист 1* или четене само на определени блокове от клетките. За повече информация вижте *help xlread*. Вие може също да изнесете данни в *Excel* файлове, като възникне нужда от това се използва *xlwrite*. Информация относно *Excel* файловете, като имена на лист може да се получи използвайки командата *xlsinfo.*

Забележка: Матлаб и *Excel* не са съгласувани относно датите. Матлаб разпознава датите като дни след Януари 1, 0000 (включително), докато *Excel* разпознава датите като ден след Януари 1, 1900. За това за да преобразувате внесените дати от *Excel* в дати на Матлаб, Вие трябва да прибавите (добавите) *datenum ('30DEC1899')* до колоната с данни, представляващи дати. Връщайки се към примера по-горе

```
>> [A,finfo]=xlsfinfo('deciles2.xls')
A =
Microsoft Excel Spreadsheet
finfo =
    'Cleaned Data'    'Original Data'
>> data  = xlsread('deciles2.xls','Cleaned Data','A2:K1257');
>> dates = data(:,1);
>> datestr(dates(1))
ans =
03-Jan-0100
>> dates = dates + datenum('30DEC1988');
>> datestr(dates(1))
ans =
03-Jan-2000
```

Този пример използува файла *deciles.xls* , който съдържа два листа, *" оригинални данни и почистени данни "*. Ако Вие отворите файла с *Excel* ще видите, че *"изчистените данни"* съдържат етикети на колоните, също така. За да внесем данни от този файл *xlsread* е нужна да знаете да изтеглите данни от *"изчистени данни"* от клетки *A2:K1275* (по-горе вляво и по-долу в дясно в ъглите на блока). **xlsread ('deciles2.xls , 'Cleaned Data', 'A2:K1257')** вършат точно това. Накрая несъгласието с датите е илюстрирано и корекцията е показана в действие в **раздел 17**.

3.3 CSV данни

CSV данни, или стойности отделяни със запетайка, приличат много на *Excel* данни. Командата за четене на *CSV* данни е всъщност ,

```
>> data = csvread('deciles.csv');
```

когато *xlsread* , Вие можете да използвате други форми за да започнете да четете в специфична клетка

```
>> data = csvread('deciles.csv',0,1);
```

или за да чете определени блокове от клетки

```
>> data = csvread('deciles.csv',1,0,[1 0 1256 10]);
```

Вие може да транспортирате данни във файлове на ***csv***, като използвате командата ***csvwrite***.

3.4 Текст

Четенето в текст, ако е съдържал само числа, е право напред. Стандартната команда е **textread** и е идентична с **xlsread**.

```
>> data = textread('deciles.txt');
```

textread може да Ви предостави разнообразни формати с данни, но препоръчително е Вие да поддържате вкараните данните, толкова просто, колкото е възможно и да използвате таблично неограничени текстови файлове (вижте като пример *deciles.txt*). Вижте **help textread** за повече информация.

3.5 Матлаб данни файлове (. mat)

Основният формат за Матлаб данните са известни като Матлаб данни файлове или *.mat files*. Тези са най-лесните за работа, а данните се зареждат просто чрез въвеждане.

```
>> load deciles.mat
```

Няма никаква нужда за определена входна променлива, понеже . *mat file* съдържа две имена на променливи и данните. Вижте по-долу за съхраняването на данните в . *mat* формат.

3.6 Четене на зле форматиран текст

Матлаб може да бъде убеден да чете почти всеки формат. Матлаб има функциите да чете избирателно текст, който може да бъде преобразуван в числа. Това е напредничава техника и би трябвало да бъдете избягвана, ако може. Обаче има някои ситуации, когато това е необходимо. Например да предположим, че суровите данни са в много голям файл (прекалено голям за Excel) и зле форматиран. В този случаи най-простата процедура е да се напише програма, която да чете файла ред по ред и да обработва (процесира) всеки ред по отделно.

Файла *IBM_TAQ.txt* съдържа прост пример за данни, които са трудни за въвеждане в Матлаб. Този файл беше свален от WRDS и съдържа всички цени за IBM от TAQ база данни в интервал 1 Януари , 2001 до 31 Декември 2001. той има прекалено много редове за да се използва в Excel и има и числа, и дати, и текст на всеки ред. Следният кодов блок показва данните в този файл. Може да им бъде направен разбор използвайки Матлаб:

```
fid=fopen('IBM_TAQ.csv');
%Count number of lines
count=0;
while 1
    line=fgetl(fid);
    if ~ischar(line)
        break
    end
    count=count+1;
end
```

```
%Close the file
fclose(fid);

%Pre-allocate the data
dates = zeros(count-1,1);
time  = zeros(count-1,1);
price = zeros(count-1,1);
%Reopen the file
fid=fopen('IBM.csv');
%Get one line to throw away since it contains the column labels
line=fgetl(fid);
%Use count to index the lines this pass
count=1;
%while 1 and break work well when reading test
while 1
    line=fgetl(fid);
    %If the line is not a character value we've reached the end of the file
    if ~ischar(line)
        break

    end'
    %Find all the commas, they delimit the file
    commas = strfind(line,',');
    %Dates are places between the first and second commas
    dates(count)=datenum(line(commas(1)+1:commas(2)-1),'ddmmmyyyy');
    %Times are between the second and third
    temptime=line(commas(2)+1:commas(3)-1);
    %Times are colon separates, so they need further parsing
    colons=strfind(temptime,':');
    %Convert the text representing the hours, minutes or and seconds to numbers
    hour=str2num(temptime(1:colons(1)-1));
    minute=str2num(temptime(colons(1)+1:colons(2)-1));
    second=str2num(temptime(colons(2)+1:length(temptime)));
    %Convert these values to seconds past midnight
    time(count)=hour*3600+minute*60+second;
    %Read the price from the last comma to the end of the line and convert to number
    price(count)=str2num(line(commas(3)+1:length(line)));
    %Increment the count
    count=count+1;
end
fclose(fid);
```

Този блок от кодове върши няколко неща;

отваря файла директно използвайки *fopen*

чете файла ред по ред използвайки *fgetl*

брои числата в редовете във файла

преразпределя *dates* (датите) , *times* (времената) и *price* (цените) като променливи използва *zeros* (нули).

Препрочита файла правейки разбор на всеки ред чрез местоположението му използвайки *strfind* за да установи (посочи) местоположението на де лимитиращия образ.

използва командата *datenum* да преобразува поредица дати в цифрови дати

използва *str2num* да преобразува поредиците в номера (цифри).

Ако е нужно да четете зле форматирани данни, Вие трябва да видите документа по *fopen*, *fscanf, fread, fgetl, dlmread,* и *textscan* и да се консултирате с раздела за основна манипулация на поредица. (раздел 18).

3.7 Трансфер

Има един финален метод, който си струва споменаването за въвеждането (внос) на данни. *StatTransfer* се намира върху сървъра и е способен за четене и писане на около 20 различни формати, включително Матлаб, GAUSS, Stata, SAS, Excel, CSV и текстови файлове. То позволява на ползващите го да качват данни в един формат и да изваждат част или всичко в друг. Използването на тази програма е извън наблюдението на тези бележки, но Вие трябва да бъдете наясно с нейното съществуване. *StatTransfer* може да направи няколко трудни за управление ситуации (например зле форматирани данни) значително по-лесно. *StatTransfer* има лесен за разбиране help file, който да Ви помогне в използването на програмата, ако е необходимо.

3.8 Изваждане на данни от Матлаб

3.8.1 Съхранение на данни

Щом Вие сте чели данни в Матлаб, Вие ги съхранявате и всякакви промени в основния формат на Матлаб данните. Това е лесно постижимо чрез извикване на:

```
>> save filename
```

Това ще създаде файл *filename.mat* съдържащ всички променливи в паметта. *filename* може да бъде заменен, с който и да е валиден *filename* на твоята операционна система. За да съхраним променливите в паметта извикваме командата

```
>> save filename var1 var2 var3 etc.
```

която създава файл *filename.mat* съдържащ var1, var2q и т.н.

3.8.2 Изнасяне на данни

Ако ти е нужно да използваш данни от Матлаб за употреба в друга програма, най-простия метод е да извикаш ***save*** с аргументи - *double -ascii*. Това ще създаде таблично де лимитиран файл на променливите, които Вие изнасяте. Въобще Вие трябва само да изнесете една програма, използвайки този метод. Изнасянето на повече от една резултира в зле

форматиран файл дето може да бъде трудно да се въведе в друга програма. Например, Вие бихте повикали

```
>> save filename var1 -ascii -double
```

за да запазите *var1* , като таблично не лимитиран (неограничен) текст. Ограничението към единична променлива трябва да бъде видно като ограничение, понеже Вие можете винаги да създадете *var1* от други променливи (например *var1= [var2 var3]).*

Алтернативни методи за износ на данни включват *xlswrite, csvwrite* и *dlmwrire*.

3. Базова (основна) математика

Математиката в Матлаб е интуитивна, ако Вие сте запознати с линейната алгебра(както Вие всеки би трябвало да бъдете). Всичко каквото можете да прави с линейната алгебра, може да се прави в Матлаб; повечето неща, които не са позволени в линейната алгебра, не са и позволени в Матлаб. За пример: да се умножат две матрици заедно те трябва да бъдат подобни по техните вътрешни размери (измерения); опитването да се умножат не съответстващи си матрици предизвиква грешка.

4.1 Оператори

Матлаб има стандартни оператори

Оператор	Значение	Пример	Алгебрично
+	събиране	x + y	$x + y$
-	изваждане	x – y	$x - y$
*	умножение	x * y	xy
/	деление	x / y	$\frac{x}{y}$
\	дясно деление	x \ y	$\frac{y}{x}$
^	на степен	x ^ y	x^y

Когато x и y са скаларни, поведението на тези оператори е очевидно. Когато x и y са матрици нещата са малко по сложни.

4.2 Матрично събиране (+) и изваждане (-)

Събирането и изваждането изискват x и y да имат същите размери или да бъдат скаларни. Ако те и двете са матрици, $z = x + y$ създава матрица с $z(i, j)= x (i, j) + y (i, j)$. Ако x е скаларно, а y е матрица, $z = x+y$, резултира в $z (i, j)= x + y (i, j)$.

Да предположим $z = x + y$:

		y: Scalar	y: Matrix
x	Scalar	Any $z = x + y$	Any $z_{ij} = x + y_{ij}$
	Matrix	Any $z_{ij} = y + x_{ij}$	Both Dimensions Match $z_{ij} = x_{ij} + y_{ij}$

Забележка: Това съответства на стандартните правила на матрично събиране и изваждане.

4.3 Матрично умножение (*)

Умножението изисква вътрешните размери да бъдат еднакви или за един вход да бъдат скаларни. Ако *x* е *N* по *M*, а *y* е *K* по *L* и двете матрици са не скаларни, *x**y* изисква *M=K*. Подобно *y**x* изисква L=N. Ако *x* е скаларно, а *y* е матрица, тогава z=x*y, съдържа *z(i,j)= x*y (i,j).*

Допускаме *z=x*y*

x \ y	Scalar	Matrix
Scalar	Any $z = xy$	Any $z_{ij} = xy_{ij}$
Matrix	Any $z_{ij} = yx_{ij}$	Inside Dimensions Match $z_{ij} = x_i y_j$

Забележка: Тези съответствия потвърждават стандартните правила на матричното умножение. x_j е ред *i* на *x*, а y_j е колоната *j* на *y*.

4.4 Матрично деление (/)

Матричното деление не е изобщо дефинирано в линейната алгебра и неговата употреба е три кована (прилагане на трикове). Интуицията за матрично деление идва от мисленето относно комплект от линеарни уравнения. Да допуснем, че има някакво *z*, *M* при *L* вектор, такъв където

$$yz = x$$

където *x* е *N* по *M*, а *y* е *N* по *L*. Делението открива *z* като решение на линеарното уравнение от най-малките квадрати, така че $z = (y'y)^{-1} (y'x)$.

Допускаме *z=x/y*

x \ y	Scalar	Matrix
Scalar	Any $z = \frac{x}{y}$	Left Dimensions Match $z = (y'y)^{-1}y'x$
Matrix	Any $z_{ij} = \frac{x_{ij}}{y}$	Left Dimensions Match $z = (y'y)^{-1}y'x$

Забележка: Като линейна регресия, матричното деление е само добре дефинирано само ако *y* не е единица.

4.5 Матрично дясно деление (\)

Матричното дясно деление е противоположно на матричното деление.

Да допуснем $z = x\backslash y$

		y: Scalar	y: Matrix
x	Scalar	Any $z = \frac{y}{x}$	Any $z_{ij} = \frac{y_{ij}}{x}$
	Matrix	Right Dimensions Match $z = (x'x)^{-1}x'y$	Right Dimensions Match $z = (x'x)^{-1}x'y$

Забележка: Като линеарна регресия матричното деление е добре дефинирано само ако *x* не е единица.

4.6 Матрично степенуване (^)

Матричното степенуване е дефинирано само ако поне едно *x* или *y* са скаларни.

Да допуснем $z = x \wedge y$

		y: Scalar	y: Matrix
x	Scalar	Any $z = x^y$	y Square Strange, Do Not Use [1]
	Matrix	x Square $z = x^y$	N/A

Забележка: В случая, където *x* е матрица, а *y* е цяло число, и $z = x*x*\ldots*x$ *(y пъти).* Ако *y* не е цяло число, тази функция включва *eigenvalues* (*eigen* стойност) (виж **help mpower**).

4.7 Кръгли скоби ( )

Кръглите скоби могат да бъдат използвани по обичайния начин за да контролират реда, по който математическите изрази са оценявани. Кръглите скоби могат да бъдат поставяни, да създаване на сложни изрази. Вижте оператор *Precedence* (прецедент) за повече информация по това как Матлаб изпълняван математически изрази.

4.8 . оператор

Операторът **.** (чете се *dot* оператор) променя обичайните операции, в операции елемент по елемент. Например, *x* и *y* са *N* по *N* матрица. *z = x . *y* регулира в обичайно матрично, където *z(i, j) = x (i , :)*y (: , j)*, докато *z = x . *y* създава *z*, където *z (i , j) = x (j , j) *y (i , j)*. Умножение *(.*),* деление *(. /),* дясно деление *(. \ )* и степенуване *(.^),* всички имат **.** форми.

```
z=x.*y    z(i,j)=x(i,j)*y(i,j)
z=x./y    z(i,j)=x(i,j)/y(i,j)
z=x.\ y   z(i,j)=x(i,j)\y(i,j)
z=x.^y    z(i,j)=x(i,j)^y(i,j)
```

Забележка: Тези оператори понякога са наричани *Hadamard*, особено .*.

4.9 Transpose (пренасяне в друга част на уравнението с обратен знак)

Матричното транспортиране е намираемо в Матлаб и се изразява в използване на ‘ оператор. Например, ако *x* е *M* по *N* матрица, *x’* е неговата транспортирана, с размери *N* по *M*.

4.10 Оператор Precedence (превъзходство)

Компютърната математика, като стандартната математика има оператор за превъзходство. Това определя как математически израз, като

```
2^3+3^2/7*13
```

са оценени. Редът на оценяване е:

Оператор	Наименование	Ранг
( )	Кръгли скоби	1
‘, ^, .^	Transpose	2
~		3
+, -	Унарен плюс, унарен минус	3
, ., /, ./, \, .\		4
+, -	Бинарно добавяне и изваждане	5
:	Колонен оператор	6
<, <=, >, >=, = =, =	Логични оператори	7
&	Елемент по елемент AND	8
|	Елемент по елемент OR	9
&&	AND	10
||	OR	11

В случай на връзка, операторите се изпълняват от ляво на дясно. Например, *x^y^z* е преведено като *(x^y)^z*.

Забележка: Унарните оператори + или –, се прилагат към единичен елемент. Например, взимаме в предвид израза *(-4),* това е пример за унарност – понеже има само една операция. *(-4)^2* създава *16*. Обаче *-4^2* създава *-16* понеже Матлаб интерпретира това като **-(4^2),** понеже -, не е вече унарен.

4. Базови (основни) функции

Тази секция осигурява (референция) препоръка за комплект от обикновено използвани функции с обсъждане по това, как те се държат (какво е тяхното ползване).

5.1 Дължина (length)

За да намерим размера на максимално измерение на ***x***, използваме **z = length (x).** Отбележете, ако $T > K$, $z = T$. Ако $K > T$, $z = K$. ***length*** е рискова команда, защото стойността, която тя връща може да бъде число от колоната или числото от редовете, зависещо от размера на матрицата.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> length(x)
ans =
     3
>> length(x')
ans =
     3
```

5.2 Размер (size)

За да намерим размера на измерението на матрицата, използваме *z=size (x, DIM),* където *DIM* е измерението. Отбележете, че измерението *1* е число от редовете, докато измерението *2* е число от колоните, така че *z= size (x,1),* връща *T*, докато *z= size (x,2),* връща *K*. Като алтернатива *z= size (x),* връща вектор ***s*** с размера на всяко измерение.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3

     4     5     6
>> size(x,1)
ans =
     2
>> size(x,2)
ans =
     3
>> s=size(x)
s =
     2     3
```

5.3 Сума (sum)

Да изчислим сумарната матрица

$$z = \sum_{t=1}^{T} x_t$$

използваме командата *sum (x). z= sum (x)* е *K* по *1* вектор на сумата на всяка колона, значи z(i) = sum (x(:, i)) = x (1 , i) + x (2, i) + … + x (T, i).

Забележка: Ако *x* е вектор, командата *sum* ще сумира елементите на *x*, докато то е редица или колонен вектор.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> sum(x)
ans =
     5     7     9
>> sum(x')
ans =
     6    15
```

x

5.4 Минимум (min)

Да намерим минималния размер на вектор или редиците на матрица,

$$\min x_{it},\ i = 1, 2, \ldots, K$$

използваме командата *min (x),* ако *x* е вектор, *min (x)* е скаларна величина. Ако *x* е матрица, *min (x)* е *K* по *1* вектор на минималната стойност на всяка колона.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> min(x)
ans =

     1     2     3
>> min(x')
ans =
     1     4
```

5.5 Максимум (max)

Да намерим *max* елемент на вектор или редиците на матрицата,

$$\max x_{it},\ i = 1, 2, \ldots, K$$

използваме командата **max(x).** Ако *x* е вектор, *max(x)* е скаларна величина, Ако *x* е матрица, *max(x)* е *K* при *1* е вектор на максималната стойност на всяка колона.

5.6 Вид (sort)

За да сортираме стойности на вектор или редици на матрица, от най-малко до най-голямо, използваме командата ***sort(x),*** Ако *x* е вектор, *sort(x)* е вектор, където *x (1) = min(x) и x (i) ≤ x(i+1).* Ако *x* е матрица, *sort(x)* е матрица от същия размер, където всяка колона е сортирана от най-малко до най-голямо.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> sort(x)
ans =
     1     2     3
     4     5     6
>> sort(x')
ans =
     1     4
     2     5
     3     6
```

5.7 Показател (exp)

За да вземем ***exp*** на вектор или матрица (елемент по елемент)

$$e^x$$

използваме *exp. z =exp(x)* връща вектор или матрица в същият размер, като *x*, където *z(i,j)=exp(x(i,j)).*

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3


     4     5     6
>> exp(x)
ans =
    2.7183    7.3891   20.0855
   54.5982  148.4132  403.4288
```

5.8 Логаритъм (log)

За да вземе естествения логаритъм на вектор или матрица

$$\log x$$

използваме *log*. *z=log(x)* връща вектор или матрица в същия размер, като *x*, където *z=(i,j)=log(x(j,j))*.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> log(x)
ans =
         0    0.6931    1.0986
    1.3863    1.6094    1.7918
```

5.9 Корен квадратен (sqrt)

Да изчислим елемент по елемент на корен квадратен на вектор или матрица

$$\sqrt{x_{ij}}$$

използваме *sqrt*. *z=sqrt(x)* връща вектор или матрица в същия размер, като *x*, където *z(I,j)=sqrt(x(i.j))*.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> sqrt(x)
ans =
    1.0000    1.4142    1.7321
    2.0000    2.2361    2.4495
```

Забележка: Тази команда създава същия резултат, като *z=x^(1/2)*.

5.10 Среда (mean)

За да изчислим **mean** на вектор или матрица,

$$z = \frac{\sum_{t=1}^{T} x_t}{T}$$

използваме командата *mean(x)*. *z=mean(x)* е *K* при *1* вектор на средните числа на всяка колона, така че *z(i)=sum(x(i, :)/length(x(i, :)*.

Забележка: Ако *x* е вектор, mean ще изчислява средно число на *x*, дали е редица или колонен вектор.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> mean(x)
ans =
    2.5000    3.5000    4.5000
>> mean(x')
ans =
     2     5
```

5.11 Несъответствие (var)

Да изчислим мострата на несъответствие на вектор или матрица,

$$\hat{\sigma}^2 = \frac{\sum_{t=1}^{T}(x_t - \bar{x})^2}{T-1}$$

използваме командата *var(x).* Ако x е вектор, *var(x)* е скаларна. Ако *x* е матрица, *var(x)* е *K* при *1* вектор на мострата променливи от всяка колона.

Забележка: Тази команда винаги използва *T-1* в знаменател.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> var(x)
ans =
    4.5000    4.5000    4.5000
>> var(x')
ans =
     1     1
```

5.12 Ко-несъответствие (cov)

Да изчислим мострата на *ко*-несъответствието на вектор на матрица,

$$\hat{\Sigma} = \frac{\sum_{t=1}^{T}(\mathbf{x}_t - \bar{\mathbf{x}})'(\mathbf{x}_t - \bar{\mathbf{x}})}{T-1}$$

използваме командата *cov(x).* Ако *x* е *вектор cov(x)* е скаларно (и е идентично на *var(x)).* Ако *x* е матрица, *cov(x)* е *K* при *K* матрица с мостра променлива по диагоналите и мостра ко-променливи по крайния диагонал.

Забележка: Тази команда винаги използва *T-1* в знаменателя.

5.13 Стандартно отклонение (std)

Да изчислим ***std*** на вектор или матрица,

$$\hat{\sigma} = \sqrt{\frac{\sum_{t=1}^{T}(x_t - \bar{x})^2}{T-1}}$$

използваме командата ***std(x).*** Ако *x* е вектор, *std(x)* е скаларна величина. Ако **x** е матрица, *std(x)* е *K* при *1* вектор на мострата стандартно отклонение на всяка колона.

Забележка: Тази команда винаги използва *T-1* в знаменателя и е еквивалентна на *sqrt(var(x)).*

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> std(x)
ans =
    2.1213    2.1213    2.1213
>> std(x')
ans =
     1     1
```

5.14 Асимитричност (skewness)

Да изчислим мострата на асиметричния вектор или матрица,

$$skew = \frac{\frac{\sum_{t=1}^{T}(x_t - \bar{x})^3}{T}}{\hat{\sigma}^3}$$

използваме командата ***std(x).*** Ако *x* е вектор,*skewness(x)* е скаларно. Ако *x* е матрица, *skewness(x)* е *K* при *1* вектор на мострата на асимитричност на всяка колона.

Забележка: Тази команда винаги използва *T-1* за изчисляване мострата на стандартното отклонение.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> skewness(x)
ans =
     0     0     0
>> skewness(x')
ans =
     0     0
```

5.15 Стандартно отклонение (kurtosis)

Да изчислим мострата на стандартно отклонение на вектор или матрица,

$$kurtosis = \frac{\frac{\sum_{t=1}^{T}(x_t - \bar{x})^4}{T}}{\hat{\sigma}^4}$$

използваме командата *kurtosis(x).* Ако *x* е вектор, *kurtosis(x)* е скаларно. Ако *x* е матрица, **kurtosis(x)** е *K* при *1* вектор на мострата на стандартното отклонение на всяка колона. **Забележка:** тази команда винаги използва *T-1* за изичсляване на мостра на променлива.

```
>> x=[1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> kurtosis(x)
ans =
     1     1     1
>> kurtosis(x')
ans =
    1.5000    1.5000
```

5.16 : оператор

Операторът *:* има две функции. Едната ще достигне елементи в матрицата на вектора (т.е. *x(1, :)),* които е описан вече. Другата е да създаде ред от вектор с по равно разпределени точки. В този контекст (смисъл), операторът *:* има две форми, *first:last* и *first:increment:last*. Основната форма*, first:last*, показва ред на вектор на формата

$$[first,\ \ first+1, \ldots first+N]$$

където *N* е големи числа, такива че *first* + *N* < = *last*. В обикновена употреба *first* и *last*, ще бъдат числа и *N* = *last – first*. Три примера показват, как работи тази конструкция:

```
>> x=1:5
x =
     1     2     3     4     5
>> x=1:3.5
x =
     1     2     3
>> x=-4:6
x =
    -4    -3    -2    -1     0     1     2     3     4     5     6
```

Вторият пример за оператор *:,* включва *increment*. Като резултат съгласуването ще има формата

$$[first,\ \ first+increment,\ \ first+2(increment), \ldots first+N(increment)]$$

където *N* е най-големя число, такова че *first* + *N(increment)* < = *last*. Вземете под внимание тези два прости примера:

```
>> x=0:.1:.5
x =
         0    0.1000    0.2000    0.3000    0.4000    0.5000
>> x=0:pi:10
x =
         0    3.1416    6.2832    9.4248
```

increment не трябва да бъде положителен. Ако се използва отрицателен *increment*, общата форма е непроменена, но той спира на условието променливата *N* е най-голямо число, такова че *first + N(increment) > = last*. Например:

```
>> x=-1:-1:-5
x =
    -1    -2    -3    -4    -5
>> x=0:-pi:-10
x =
         0   -3.1416   -6.2832   -9.4248
```

Забележа: *first: last*е същото като *first:1: last*, където 1 е *increment* (диференциал).

5.17 Подреждане на редове (linspace)

linspace е подобен на операторът *:*. Както и да е, по скоро отколкото създаваното ред вектор с определен диференциал, *linspace* създава ред вектор с определен брой възли. Родовата форма е *linspace (lower, upper, N),* където *lower(по-малки)* и *upper (горни)* са двете граници на серията и *N* е броя точки, който ще се получи.

Ако *inc* е дефинирано като *inc= (upper – lower) / (N-1),* получената последователност ще има следната форма:

$$[lower,\ \ lower + inc,\ \ lower + 2(inc), \ldots lower + N(inc)]$$

и командата *linspace(lower,upper,N)* ще създаде същият изход като *lower* :

```
(upper-lower)/(N-1):upper.
```

Забележка: <u>*Помнете:*</u> операторът *:* е с най-нисък приоритет, така че Мие трябва винаги да включвате твърдения в скоби, ако има неща различно на същата линия. При провал да направите това може да резултира в неочаквано поведение. Например:

```
>> N=4;
>> lower=0;
>> upper=1;
>> linspace(lower,upper,N)-(lower:(upper-lower)/(N-1):upper)
ans =
  1.0e-015 *
         0         0   -0.1110         0
>> linspace(lower,upper,N)-lower:(upper-lower)/(N-1):upper
ans =
         0    0.3333    0.6667    1.0000
```

Матлаб (правило, базиран върху неговите правила),тълкува второто като:

```
>> (lower:(upper-lower)/(N-1):upper-lower):(upper-lower)/(N-1):upper
```

5.18 logspace

logspace създава точки еднакво разпределени на *log10* в пространството.*logspace(lower*, *upper, N)* е същото като *10.^linspace(lower, upper, N*).

```
>> logspace(0,1,4)
ans =
    1.0000    2.1544    4.6416   10.0000
```

5. Специални матрици

Матлаб има известен брой специални матрици, с които трябва да бъдете запознати.

6.1 Единична (ones)

ones матрица върши точно каквото трябва: генерира матрици от единици. *ones* е с двоен аргумент, брой на редове и брой на колони

```
oneMatrix = ones(N,M)
```

ще създаде матрица от единици с *N* редове и *M* колони.

Забележка: За да използвате функцията посочена по горе, *N* и *M* трябва да са дефинирани преди това *(m.e. N=10; M=7).*

6.2 eye

eye генерира идентична матрица (матрица с единици по диагонала, нули навсякъде другаде). Идентичната матрица е винаги квадратна, така че тя взима само един аргумент:

```
In = ones(N)
```

6.3 zeros

zeros създава матрица от нули, по същият начин като *ones* създава от единици и е полезна за инициализирането на матрица съдържаща стойности от друга процедура.

```
x = zeros(N,M)
```

6. Функции на матрицата

Матлаб има известен брой функции специално създадени да взимат матрични входове. Някои са математически по природа, например изчисляващи *eign* стойности и *eign* вектори на матрица, докато други са просто функции за манипулиране елементите на матрицата.

7.1 remat

remat заедно с reshape (реформиране) са всред най-полезните не математични функции намиращи се в Матлаб. *remat* репликира матрицата според определен размер вектор. За да разберем как *remat* функционира, представете си формирането на матрица съставена от блокове. Основната форма, от която произлиза *remat* е:

remat(X, M, N)

където *X* е матрицата, която се репликира, *M* е броя на редовете в новата блок матрица, а *N* е броя на колоните в новата блок матрица

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

и ние се нуждаем да формираме блок – матрици

$$Y = \begin{bmatrix} \mathrm{X} & \mathrm{X} & \mathrm{X} \\ \mathrm{X} & \mathrm{X} & \mathrm{X} \end{bmatrix}$$

Това би могло да бъде придружавано от ръчно създаден *y*, като:

```
>> x = [1 2; 3 4];
>> y = [x x x; x x x];
```

Обаче, *y* би могло също да бъде създадено с използването на *remat*.

```
>> x = [1 2; 3 4];
>> y = repmat(x,2,3);
```

remat има две ясни предимства над ръчното локализиране: **(1)** то може да бъде изпълнено на базата на някои параметри, непознати, когато Вие пишете програмата, такива като броя на обясними променливи в един модел и **(2)** той може да бъде използван за арбитриране на измерения. Ръчно създаване на матрица става уморително и със склонност към грешка с поне 5 реда и колони.

7.2 reshape

reshape трансформира матрицата с един комплект измерения в матрица с различен комплект , запазвайки броя на елементите. Да допуснем, че желаете да превърнете *x* в *y*, използвайки *reshape*, където *x* е*M*, при N, а *y* е *K*, при *L*. *reshape* може да трансформира едното в другото, *MN* = *KL* . Най-полезното извикване за реформиране е превключване на матрица във вектор или обратно. Например:

```
>> x = [1 2; 3 4];
>> y = reshape(x,4,1)
y =
     1
     3
     2
     4
>> z = reshape(y,1,4)
z =
     1     3     2     4
>> w = reshape(z,2,2)
w =
     1     2
     3     4
```

Основен детайл, който трябва да помните, когато използвате *reshape*, е че Матлаб винаги ще използва главна колона мутация за определяне формата на новата матрица. Матлаб винаги отброява надолу, а не непряко, така че тя ще обозначи числа към елементи на старата матрица 1,2,… и ще обозначи числа към местоположенията в новата матрица и ще постави стойностите в същите числови (номерирани) местоположения.

7.3 diag

diag може да създаде един от два резултата зависещи от формата на входа. Ако входа е квадратна матрица, той ще върне колонния вектор на елементите по диагонала на матрицата.Ако входа е вектор, той ще върне матрицата с елементите на диагонала по вектора. Вземете под внимание следния пример:

```
>> x = [1 2; 3 4];
>> y = diag(x)

y =
     1
     4
>> z=diag(y)
z =
    1     0
    0     4
```

7.4 det

det изчислява детерминантата на квадратна матрица.

7.5 trace (следа, проследяване, трасе)

trace изчислява трасето, следата на квадратната матрица (сумата от диагоналните елементи) . ***trace(x)*** е еквивалентно на *sum(diag(x)).*

7.6 chol

chol изчислява фактора Чолески на позитивно определена матрица. Фактора Чолески е по-ниска триъгълна матрица и се дефинира C.

$$CC' = \Sigma$$

където Σ е позитив определя матрицата.

7.7 inv

inv изчислява инверсията (размерите) на матрица. *inv(x)* може алтернативно да бъде изчислено използвайки *x^(-1).*

7.8 eig

Командата ***eig*** изчислява числените стойности и числените вектори на квадратна матрица. За да получите и двете стойности и векторите трябва да осигурите два изходни аргумента, *[val, vec]=eig*x).*

7.9 kron

kron изчислява кроникеровия продукт от две матрици,

$$z = x \otimes y$$

и се пише като *z= kron(x,y)* в Матлаб кода.

7. Inf, NaN и числови граници (лимити)

Матлаб има три специални израза, запазени (вие не би трябвало да ги използвате) да посочат определени не цифрови изрази.

Inf е стандарт за безкрайност. Матлаб разбира както *Inf*, така и *–Inf*. *Inf* може да бъде конструиран по много начини. Например *1/0* или *exp(1000).*

NaN се използва за не-число. *NaN* се създават, когато въобще изучавате функция, която не може да бъде ясно дефинирана, като число от безкрайност. Например, *inf/inf* създава *NaN*.

Матлаб има някои числови граници. Най-лесно за разбиране са по-горните и по-долните граници, които са 1.7977e+308 и -1.7977e+-308. Числа по-големи (в абсолютна стойност), отколкото тези са Inf, доколкото Матлаб е свързан. Най-малкото не-нула число, което Матлаб може да разбере е 2.2251e-308. Числата между -2.2251e-308 и 2.2251e-308 са нула, доколкото се отнася до Матлаб.

Обаче, най-твърдата концепция за разбиране на цифри е лимитирана прецизност. Матлаб има прецизност на 2.2204e-016 (командата на Матлаб *eps* дава това число и то зависи от CPU управлението на компютри, което използвате). Това значи, че числата които са отвън от относителния обхват на 2.224e-016 са считани за същите. Например, прегражданe на резултатите от следния вход в Матлаб.

```
>> x=1
 x =
     1
 >> x=x+eps/2
 x =
      1
 >> x-1
 ans =
      0
 >> x=x+2*eps
 x =
     1.0000
 >> x-1
 ans =

4.4409e-016
```

За да разберем какво се има предвид за относителен обхват прегледайте следния изход:

```
>> x=10
x =
    10
>> x+2*eps
ans =
    10
>> x-10
ans =
     0
```

В първия пример, *eps/2* < *eps*, така че то няма никакъв ефект, докато *2*eps* > *eps* има. Обаче втория пример *2*eps/10* < *eps* няма никакъв ефект, когато се добавя. Това е много триков подход за разбиране, но ако Вие правите нещо комплексно с Матлаб, това би трябвало да е нещо, което разбирате.

Раздел 9

Логични оператори и намиране

Логичните оператори са много важни, когато пишем партидни файлове или скриптове (почерк или ръкопис). Логичните оператори, когато са комбинирани с контрол на потока, позволяват комплексни избирания да бъдат компактно изразени.

9.1 >, > =, <, < =, = =, ~ =

Логичните оператори са:

>	по-голям от
> =	по-голям или равен на
<	по-малки от
< =	по-малки или равни на
= =	равни на
~ =	не равни на

Логичните оператори могат да действат върху скалари, вектори или матрици. Всички сравнения се вършат елемент по елемент и се връщат намалени към *1* (логично вярно) или *0* (логично невярно). Например, да предположим, че *x* и *y* са матрици**, *z = (x<y)*,** ще бъде ли матрица от същия размер компонирани или съставена от 0’ или 1’. Алтернативно, ако е едното е скалар, да речем *y*, тогава елементите на *z* са просто ***z(i,j) = (x (i , j)<y)***;.

Логичните оператори могат да се използват за достъп до елементи на вектор на матрица. Например, да допуснем, че $z = x \ L \ y$, където *L* е един от логичните оператори по горе, такъв като <. Следващата таблица изследва поведението, където *x и / или y* са скаларни или матрици.

Да предположим, че *z = x< y*:

x \ y	Scalar	Matrix
Scalar	Any $z = x < y$	Any $z_{ij} = x < y_{ij}$
Matrix	Any $z_{ij} = x_{ij} < y$	Same Dimensions $z_{ij} = x_{ij} < y_{ij}$

Логичните оператори се използват в порции от програми известни като контрол на потока (текущ контрол) (например, *if …else … end* блокове), които ще бъдат обсъждани по-

късно. То е важно да се запомни, че вектор или матрица, логичните операции връщат вектора или матрицата на изхода като блокове на текущ контрол, изискват скаларно логическо изразяване.

9.2 & (AND), | (OR) или ~ (NOT)

Логическите изразявания могат да бъдат комбинирани с три логически оператора,

&	AND
|	OR
~	NOT

Тяхната употреба е право напред, толкова дълго докато помните, че ~ *(NOT)* има по-високо преимущество, отколкото *& (AND)* и | *(OR).* Правилата за използването им са същите, както за използване на оригиналните логически оператори. Ако се използват две матрици, измеренията трябва да бъдат същите. Ако се използват скалар и матрица, ефектът е същият като извикването на логически оператор върху скалар и всеки елемент на матрицата. Да допуснем, че *x* и *y* са логични променливи (единици или нули). Да допуснем, че ***z = x&y,***

		y: Scalar	y: Matrix
x	Scalar	Any $z = x\&y$	Any $z_{ij} = x\&y_{ij}$
	Matrix	Any $z_{ij} = x_{ij}\&y$	Same Dimensions $z_{ij} = x_{ij}\&y_{ij}$

9.3 Логически (logical)

Матлаб командата ***logical*** се използва да промени не логическите елементи в логически. Матлаб третира логически и редовните числа малко по-различно. Логическите елементи покачват само 1 байт от паметта (най-малко единица памет на Матлаб може да се адреси), докато редовните (правилните) числа изискват 2 байта. В определени ситуации се изисква логическа стойност. Един такъв пример е в индексирането на множество.

Както демонстрирах ме преди, елементите на матрицата *x* могат да бъдат достигнати чрез *x* **(#)**, където **#** може да бъде вектор от индекси. Докато елементите на x са индексират 1,2…, опит да възстанови *x (0),* ще бъде грешка. Обаче, ако **#** не е число, но ако е логично, това поведение се променя. Матлаб интерпретира логични индекси като indicator функции. Вземете под внимание поведението от следния код:

```
>> x = [ 1 2 3 4];
>> y = [1 1];
>> x(y)
ans =
   1 1
>> y = logical([1 1]);
>> x(y)
ans =
   1 2
>> y = logical([1 0 1 0]);
>> x(y)
ans =
   1 3
```

ефектът на логичност е ясен: Той принуждава Матлаб да интерпретира индексите, като индикаторни променливи, когато решаваме какво да върнем. За другия пример вземете под внимание следния брои кодове:

```
>> x = [ 1 2 3 4];
>> y = x<=2;
>> x(y)
ans =
   1 2
>> y
ans =
   1 1 0 0
```

Забележка: Логичността превръща всяка не нулева стойност в логически вярна (1), въпреки че Матлаб ще генерира предупреждение.

Например:

```
>> x=[0 1 2 3]
x =
     0     1     2     3
>> logical(x)
Warning: Values other than 0 or 1 converted to logical 1.
ans =
    0     1     1     1
```

9.4 all и any (всичко и което и да е)

Командите на Матлаб **all** и **any** взимат логичността, като вход и само и са само дискрептивни (описателни, нагледни). **all** връща *logical (1)* всички логически елементи във вектор са *1*. Ако **all** извика матрица от логически елементи, тя работи колона по колона, като връща единица, ако всички елементи на колоната са логически верни, в противен случаи връща нула. **any** връща връща **logical (1)**, ако някои елемент на вектора е логически верен. Отново, ако повикаме матрица, **any** работи колона по колона, връщайки логическо вярно, ако някои елемент от тази колона е верен.

```
>> x = [ 1 2 3 4];
>> y = x<=2;
>> all(y)

ans =

0

>> any(y)

ans =

1

>> x = [ 1 2 ; 3 4];
>> y = x<=3;
>> all(y)

ans =

1 0

>> any(y)

ans =

1 1
```

9.5 find (намиране)

find е изключително полезна функция за работа с данни. Тя самата не е логическа, но взема логически входове и връща матрични индекси, където логическото твърдение е вярно. Има два първични начина за извикване на **find**. **indices = find (x < y)** ще върне индекси *(1.2…..numel (x))* докато **[i, j] = find (x < y)** ще върне чифтове от матрични индекси, където съответстват на местата, където *x<y*.

```
>> x = [ 1 2 3 4];
>> y = x<=2
y =
   1 1 0 0
>> find(y)
ans =
   1 2
>> x = [ 1 2 ; 3 4];
>> y = x<=3
ans =
   1 1 1 0
>> find(y)
ans =
   1 2 3
>> [i,j] = find(y)
i =
   1 2 1
i =
```

```
1 1 2
```

9.6 is * (e)

Матлаб осигурява известен брой логически тестове със специална цел, за да определи дали една матрица съдържа елементи от този тип. Някои оперират елемент по елемент и създават матрица от същото измерение, както входната матрица, докато други създават само скаларни. Всички тези функции използват **is**.

`isnan`	1 if `NaN`	element-by-element
`isinf`	1 if `Inf`	element-by-element
`isfinite`	1 if not `Inf`	element-by-element
`isreal`	1 if not complex valued.	scalar
`ischar`	1 if input i s a character array	scalar
`isempty`	1 if empty	scalar
`isequal`	1 if all elements are equal	scalar
`islogical`	1 if input is a logical matrix	scalar
`isscalar`	1 if scalar	scalar
`isvector`	1 if input is a vector $(1 \times K$ of $K \times 1)$.	scalar

Има известен брой други **is*** изрази със специална цел. За повече информация търсете за **is*** във файла за помощ.

```
>> x=[4 pi Inf Inf/Inf]
 x =
     4.0000    3.1416       Inf       NaN
 >> isnan(x)
 ans =
      0     0     0     1
 >> isinf(x)
 ans =
      0     0     1     0
 >> isfinite(x)
 ans =
      1     1     0     0
```

Забележка: isnan(x) + isinf(x) + isfinitex винаги е равно на 1, въвеждайки който и да е елемент се разпада на една от тези категории.

10. Контрол на потока (информация)

В предни раздели използвахме само една употреба на логическите променливи, избирайки елементи от матрица. Логическите променливи имат и втора употреба в програмата контрол на потока. Тази команда позволява различен код да бъде изпълнен в зависимост от това дали конкретни условия са на лице. Две структури за контрола са в наличност.

if …elseif…else u switch…case…otherwise.

10.1 If Elseif Else

if …elseif…else са по обикновените от двата. Тези блокове винаги започват с *if* твърдение, веднага последвано от s**calar** логически израз и трябва да бъдат ограничени с **end**. *elseif* и *else* са опции и могат винаги да бъдат репликирани използвайки вгнездени *if* твърдения и логически комплекси. Основната форма на *if …elseif…else* е

```
if logical_1
        Code to run if logical_1 true
elseif logical_2
        Code to run if logical_2 true and logical_1 false
elseif logical_3
        Code to run if logical_3 true and logical_j false, j < 3
...
...
else
        Code to run all logical's false
end
```

Обаче по простите форми са по често използвани:

```
if logical_1
        Code to run if logical_1 true
end
```

or

```
if logical_1
        Code to run if logical_1 true
else
        Code to run if logical_1 false
end
```

Забележка: Не забравяйте, че всички логични трябва да бъдат скаларни логични стойности.

Няколко прости примера:

```
>> x = 5;
>> if x<5
     x=x+1;
   else
     x=x-1;
   end
>> x
ans =
    4
```

и

```
>> x = 5;
>> if x<5
     x=x+1;
   elseif x>5
     x=x-1;
   else
     x=2*x;
   end
>> x
ans =
    10
```

Тези примери във всичките са използван прости логически изрази. Обаче,някои скаларни логически изрази такива като **(x<0 || x>1) && (y<0 || y>1)** или *isinf(x)* || *isnan(x)*, може би са *if …elseif…else* блокове.

10.2 Switch Case Otherwise

switch…case…otherwise са по напреднал контрол на потока и могат да бъдат изцяло репликирани, използвайки само *if …elseif…else* блокове на контрол на потока. Не се чувствайте задължени да използвате тези, ако не Вие удобно с тях. Основната структура на този блок е да намери някоя променлива, чиято стойност може да бъде използвана, за да избере парче (част) от кода и да го изпълни (променливата ***switch***). В заявка от стойността на тази променлива (нейния ***case***), конкретна част от кода ще бъде изпълнена. Ако никакви случаи не са свързани (**otherwise**) не изпълнен код от кода се изпълнява (**otherwise** може безопасно да бъде пропусната. В този случаи нищо не се върши, ако един от *cases* не са свързани (не съответстват). Обаче, с командата *at most* един блок е свързан. Свързването на ***case*** – командата заставя този кодов блок да изпълни, тогава програмата продължава на следващия ред след *switch…case…otherwise* блок. Основната форма на блока *switch…case…otherwise* е

```
switch variable

case value_1
        Code to run if variable = value_1 true

case value_2
        Code to run if variable = value_2 true

case value_3
        Code to run if variable = value_3 true

...

...

otherwise
        Code to run if variable ≠ value_j ∀j

end
```

Забележка: Има равновесие между *switch…case…otherwise* и *if …elseif…else* блоковете. Обаче, логическите изрази съдържат не равенства, логическите променливи трябва да бъдат създадени приоритетно за използване на *switch…case…otherwise* блока. Също ако имате C- програмираща среда, поведението на *switch…case…otherwise* е различно: единият случаи може да бъде свързан към един блок. След като първият е свързан, блокът е свален и програмата обобщава със следващия ред след блока.

Един прост пример със *switch…case…otherwise:*

```
x=5;
switch x
    case 4
        x=x+1;
    case 5
        x=2*x;
    case 6
        x=x-2;
    otherwise
        x=0;
end
 >> x
 ans =
    10
```

cases може да включва крайни стойности за *switch* променлива, използвайки обозначението **case** {$case_1$, $case_2$,…}. Например:

```
x=5; switch x
    case {4}
        x=x+1;
    case {1,2,5}
        x=2*x;
    otherwise
        x=0;
end
```

```
>> x
ans =
   10
```

11. Loops (Цикли)

Loops са най-полезната програмна структура намираща в Матлаб и може да прави много проблеми, особено когато се комбинира с блоковете на контрол на потока. Матлаб има *loops* от два типа, ***for … end*** и ***while … end***. **for loops** прави примка над предварително определения iterator и докато примките свършат, когато някои логически условия са на лице. Всичко за примките може да бъде изразено, като **while loops**, въпреки че противоположно е **not** истина. Те са почти еквивалент, когато *break* се използва, въпреки че кода е ненужно усложнен в тази ситуация. Две други команди, **return** и **break** са също полезни в кода на *loops*.

11.1 for

for *loops* започват с **for** *iterator = vector* и свършват с **end**.Основна структура на една *for loops* е:

```
for iterator=vector
      Code to run
end
```

iterator е променливо име*, loops* е повтаряща се през. Например, **i** е обикновен *iterator, vector* е вектор от данни. Той може да бъде съществуващ вектор или той може да бъде създаден в движение, използвайки **linspace** или **a:b:c** синтаксис (т.е. 1:10). Един тънък аспект на loops в Матлаб, е че *iterator* (повтарача) може да съдържа, който и да е вектор данни, включително не цяло число and/or отрицателни стойност. Вземете под внимание тези три примера:

```
count=0;
for i=1:100
   count=count+i;
end

count=0;
for i=linspace(0,5,50)
   count=count+i;


 end

 count=0;
 x=linspace(-20,20,500);
 for i=x
    count=count+i;
 end
```

Първата *loops* ще се повтори *през i= 1,2,…,100.* Каква е крайната стойност на броенето? Вторите *loops* през стойностите, създадени от функцията *linspace*, която създава

50 еднакви точки между 0 и 5, включително. Последните *loops* през *x*, един вектор конструиран от повишаван в *linspace*. *Loops* могат също да се повтарят през намаляващи последователности (съответствия):

```
count=0;
x=-1*linspace(0,20,500);
for i=x
    count=count+i;
end
```

or vector with no order:

```
count=0;
x=[1 3 4 -9 -2 7 13 -1 0];
for i=x
    count=count+i;
end
```

Ключът за разбиране поведението на ***for*** *loops* , е че Матлаб винаги се повтаря през елементите на *vector* в реда, в който са представени (например, *vector(1), vector(2),….). Loops* могат също за бъдат вгнездени:

```
count=0; for i=1:10
    for j=1:10
        count=count+j;
    end
end
```

и могат да съдържат променливи на контрол на потока:

```
returns=randn(100,1); count=0; for i=1:length(returns)
    if returns(i)<0
        count=count+1;
    end
end
```

Една особено полезна примкова конструкция е да впримчи през **length** на вектор, като под волево всеки елемент да бъде модифициран по един всеки път.

```
trend=zeros(100,1);
for i=1:length(trend)
    trend(i)=i;
end
```

Накрая тези идеи могат да бъдат комбинирани за да създадат вгнездени разклонения с поточния контрол.

```
matrix=zeros(10,10);
for i=1:size(matrix,1)
    for j=1:size(matrix,2)
        if i<j
            matrix(i,j)=i+j;
        else
            matrix(i,j)=i-j;
        end
    end
end
```

Вие бихте дори могли да конструирате *loops* с вгнездени *loops*, които зависят от някое твърдение на поточния контрол.

```
matrix=zeros(10,10);
for i=1:size(matrix,1)
    if (i/2)==floor(i/2)
        for j=1:size(matrix,2)
            matrix(i,j)=i+j;
        end
    else
        for j=1:size(matrix,2)
            matrix(i,j)=i-j;
        end
    end
end
```

Забележка: Вие ***NOT*** трябва да модифицирате *iterato*r –та променлива вътре в предната *loops* (предходната). Променливата на *iterator* можа да създаде нежелани резултати. Например:

```
for i=1:100;
    i
    i=1;
    i
end
```

създава изход

```
...
...
 i =
    99
i =
     1
i =
   100
i =
     1
```

което може да доведе до непредсказуеми резултати, ако **i** е използван вътре в *loops*.

11.2 while (докато)

while *loops* са полезни, когато броя на необходимите повторения е неизвестен. *while loops* обикновено се използват, когато една *loops* би трябвало сама да спре, ако определено условие е налице, такова като промяната в някои параметър е малка. Основната структура на *while loops* е:

```
while logical
        Code to run
        Update to logical inputs
end
```

Две неща са особено важни, когато използуваме *while loops* :

1. *logical* би трябвало да е вярно, когато loops започва (или разклонението ще бъде игнорирано);
2. входовете към логическата променлива трябва да бъдат оставени временно вътре в *loops*. Ако те не са loops ще продължава завинаги (комбинацията от CTRL+C за да разрушиш грешната *loops*). Най-простите *while loops* се включват със заместванията за *for loops*:

```
count=0;
i=1;
while i<=10
    count=count+i;
    i=i+1;
end
```

което създава същите резултати като:

```
count=0;
for i=1:10
    count=count+i;
end
```

while *loops* би трябвало изобщо да се избягват, когато *for loops* ще се използват, обаче има ситуации, където не съществува еквивалент на for:

```
mu=1;
index=1;
while abs(mu)>.0001
    mu=(mu+randn)/index;
    index=index+1;
end
```

В блока по-горе, броят на изискваните повторения не е известен предварително и понеже **randn** е стандартно, нормално число, то може да отнеме много повторения, докато този критерий е налице. Която и да е ограничена ***for*** *loops* не може да бъде гарантирана за наличието на критерий.

11.3 break

break може да бъде използван за да прекъсне примка и може да прави поведението на ***for*** *loops* почти като на *while loop*:

```
for iterator=vector
      Code to run
      if logical
            break
      end
end
```

Единствената разлика между тази примка и стандартната *while loop* , е че *while loop* би могла потенциално да задейства повече повторения и *iterator*. Командата **break** може също да бъде използвана да прекрати *while loop* преди въвеждането на код вътре в нея. Вземете в предвид тази малко страна примка:

```
while 1
    x = randn;
    if x<0
        break
    end
    y = sqrt(x);
end
```

Използването на *while 1* ще създаде примка, ако е оставено само това ще върви недефинирано. Обаче командата **break** ще спре примката, ако някакво условие е налице. По-важно, е че командата **break** ще предпази кода след нея от това да бъде въведен, което е полезно, ако операциите **break** ще създадат грешки, ако условието не е вярно.

11.4 continue (продължение)

continue**,** когато се използва вътре в примката има ефектът за преместване на примката към следващо повторение и прескачане на някои оставащ код в тялото на примката. Нейното използване може да бъде избегнато, използвайки блокове *if …else*, но то може да направи кода по-подреден. Нейният ефект се вижда най-добре при блок на код:

```
for i=1:10
    if (i/2)==floor(i/2)
        continue
    end
    i
end
```

който създава изход

```
...
...
i =
     7
i =
     9
```

демонстриращ, че командата **continue** принуждава примката към следващо повторение, когато винаги ***i*** е даже *((i /2) = = floor(i / 2)* е логично вярно).

Раздел 12

Plotting Data (графични данни, диаграма)

Матлаб има разтеглени (удължени) приспособления за диаграма и могат вероятно да бъдат убедени (накарани) да създаде някаква графика, където ти желаеш. Обаче, ние ще стигнем до basics.

12.1 Support Functions (поддържащи функции)

Всички графични функции имат комплект от поддържащи функции, които са полезни за поставяне на етикети (надписи) за различни части на графиката или правят надстройки към обхвата. Вие би трябвало винаги да помните изцяло да надписвате вашата графика, така че други (и Вие, когато сте забравили) да можете ясно да кажете, какво е било направено като графика и разделите на нея.

- **legend** (легенда) надписва различните елементи на Вашата графика. Специфичната функция на легендата зависи от типа функция и реда, по който Вие правите графиката на вашите данни. **legend** взима толкова много връзки, колкото вашата графика има уникални елементи. Стандартната употреба на **legend** е **('Series 1', ' Series 2'),** където броят на сериите е цифрова зависимост.
- **title** поставя заглавия отгоре на фигурата. Стандартната употреба е **title ('Figure Title').**
- **xlabel, ylabel** и **zlabel** създават тестови надписи на *x, y* и *z*, (ако 3D) съкращава респективно. Стандартната употреба е **xlabel ('X Data Name').**
- **axis** може да бъде използвана и за двете: да получава axis граници и да постави axis граници. Да върнем текущите axis граници, да въведем **AX = axis () ;**. **AX** ще бъде редица от вектор на формата *[xlow xhigh ylow yhigh (zlow) (zhigh)],* където **zlow** и **zhigh** са включени, ако фигурата е триизмерна (3D). Този axis може да бъде променен чрез извикване на **axis ([xlow xhigh ylow yhigh (zlow) (zhigh)])**, където **z** – променливите са позволени, ако фигурата е 3D. **axis** може да бъде използвана за да затегне axis да включи само мини пространство, необходимо да изрази данните, използвайки командата **axis tight**.

Тези черти са най-важните, но има много допълнителни функции за да се щипната твоите фигури (числа).

12.2 Plot (диаграма)

plot е най-основната команда за диаграма, намираща се в Матлаб. Като повечето команди тя може да бъде използвана по много начини. Обаче най-стандартно използвана е:

`plot(x1,y1,`$format_1$`,x2,y2,`$format_2$`,...)`

където *xi* и *yi* са вектори от същия размер и $format_i$, е форматната линия на формата *“color”(цвят), “shape”(форма), “line spec”*. *“color”* може да бъде, който и да е от следните цветове:

b	синьо
g	зелено
r	червено
c	цианит
m	
y	жълто
k	черно

“shape” може да бъде, който и да е:

o	circle
x	x-mark
+	plus
*	star
s	square
d	diamond
v	triangle (down)
^	triangle (up)
<	triangle (left)
>	triangle (right)
p	pentagram
h	hexagram

и *“Line Spec”* може да бъде, който и да е:

-	непрекъснат
:	точкуван
-.	тире-точка
- -	две тирета
	без линия

Трите аргумента са комбинирани за формата *string* (редица). Например *‘gs-‘* ще създаде зелена непрекъсната линия с квадрати на всяка точка данни, докато *‘r+:’* ще създаде

точкувана червена линия с *+'s* на всяка точка данни. Аргументите, които не са необходимо могат да бъдат изоставени. Например, ако Вие искате зелена линия на точки без никакъв символ, форматната редица би била просто *'g:'*. Ако никаква форматна редица не е осигурена Матлаб ще използва автоматична цветова схема и *plot* (диаграма) непрекъснати линии без никакви маркери. Да предположим, че следните *x* и *y* данни са били създадени в Матлаб.

```
x = linspace(0,1,100);
y1 = 1-2*abs(x-0.5);
y2 = x;
y3 = 1-4*abs(x-0.5).^2;
```

Извиквайки **plot (x,y1,'rs:',x,y2,'bo-.',x,y3,'kp- -')** ще се създаде диаграмата на *фигура 12.1*. вие би трябвало да отбележите, че информацията по цветните линии е загубена, когато отпечатаните документи са в черно и бяло, така че Вие трябва винаги да използвате физичните характеристики за да различите мултипликационни серии (или различни типова линия, или различни маркери, или и двете).

Всички диаграми би трябвало да бъдат ясно надписани и тази диаграма не е изключение.следващия блоков код надписва осите, дава цифрата на заглавие и осигурява легенда. Резултатите от използването на командата **plot** по-горе може да бъде видяно на *фигура 12.2*.

```
xlabel('x');
ylabel('f(x)');
title('Plot of three series');
legend('f(x)=1-|x-0.5|','f(x)=x','f(x)=1-4*abs(x-0.5).^2');
```

Една друга форма на **plot** командата си струва споменаването. Командата **plot (y)**ще постави plot данните във вектор *y* срещу прости серии, които надписват на всяко наблюдение *1,2, …, length (y).* Фактически **plot (y)** е еквивалентно на **plot(1:length(y),y)**, където *y* е вектор. Ако *y* е матрица, командата *plot* ще нарисува всяка колона на *y*, като че ли тя е била отделна серия. Когато *y* е матрица **plot(y)** е еквивалентна на **plot(1:length(y(:,1)), y(:,1), 1:length(y(:, 2)), y(:,2), …)**

12.3 Plot3

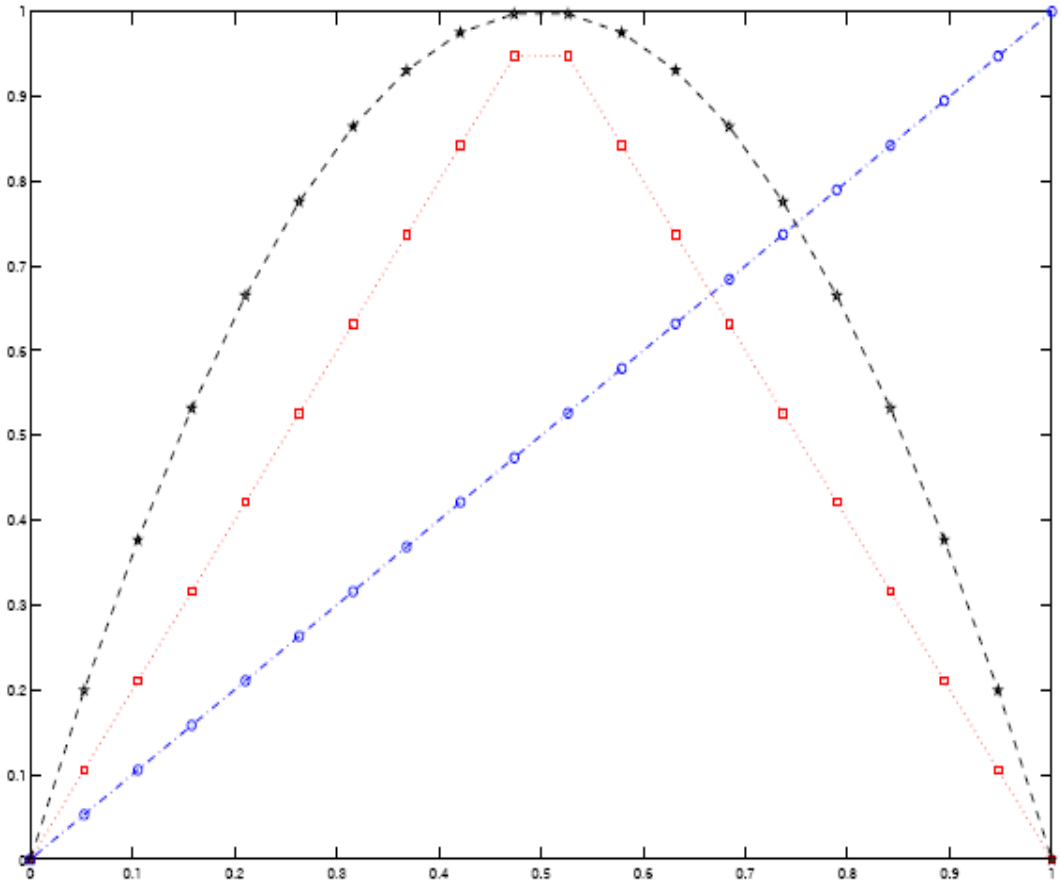
Командата **plot3** по същество е идентична на командата **plot**, с изключение, че тя прави диаграма на серия срещу две други серии в 3D пространството. Всички аргументи са същите и основната форма е:

```
plot3(x1,y1,z1,format1,x2,y2,z2,format2,...)
```

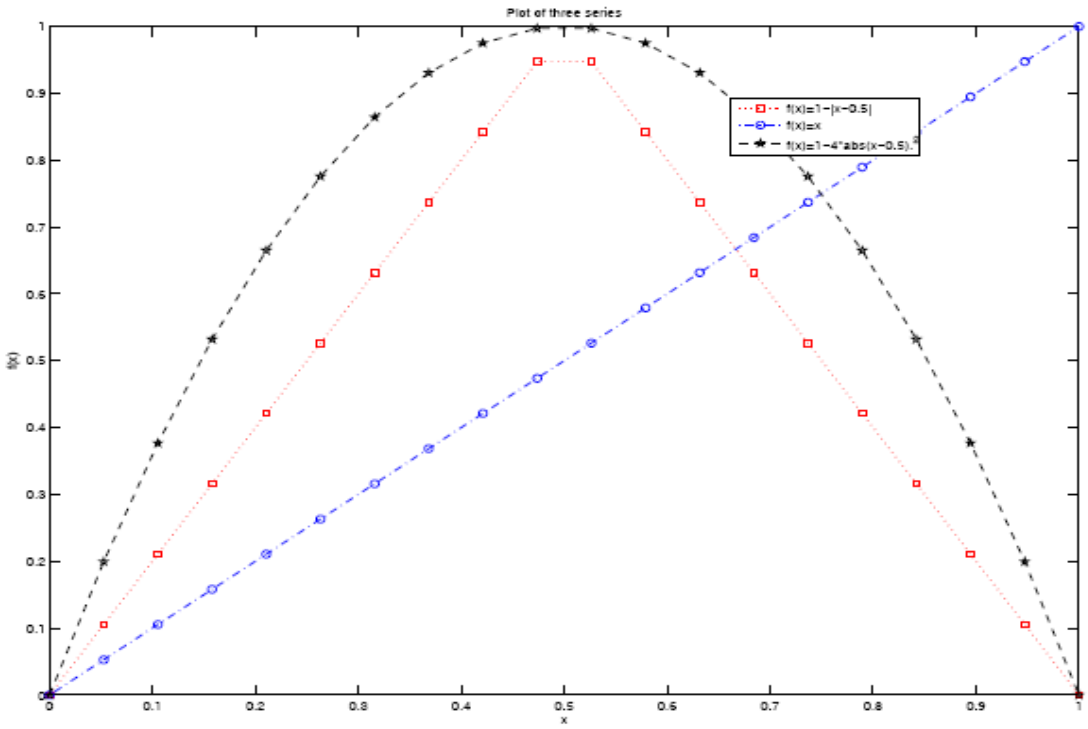
Следващия кодов блок демонстрира употребата на *plot3*:

```
figure(2)
N=200;
x=linspace(0,8*pi,N);
x=sin(x);
y=linspace(0,8*pi,N);
y=cos(y);
z=linspace(0,1,N);
plot3(x,y,z,'rs:');
xlabel('x');
ylabel('y');
zlabel('z');
title('Spiral');
legend('Spiraling Line')
```

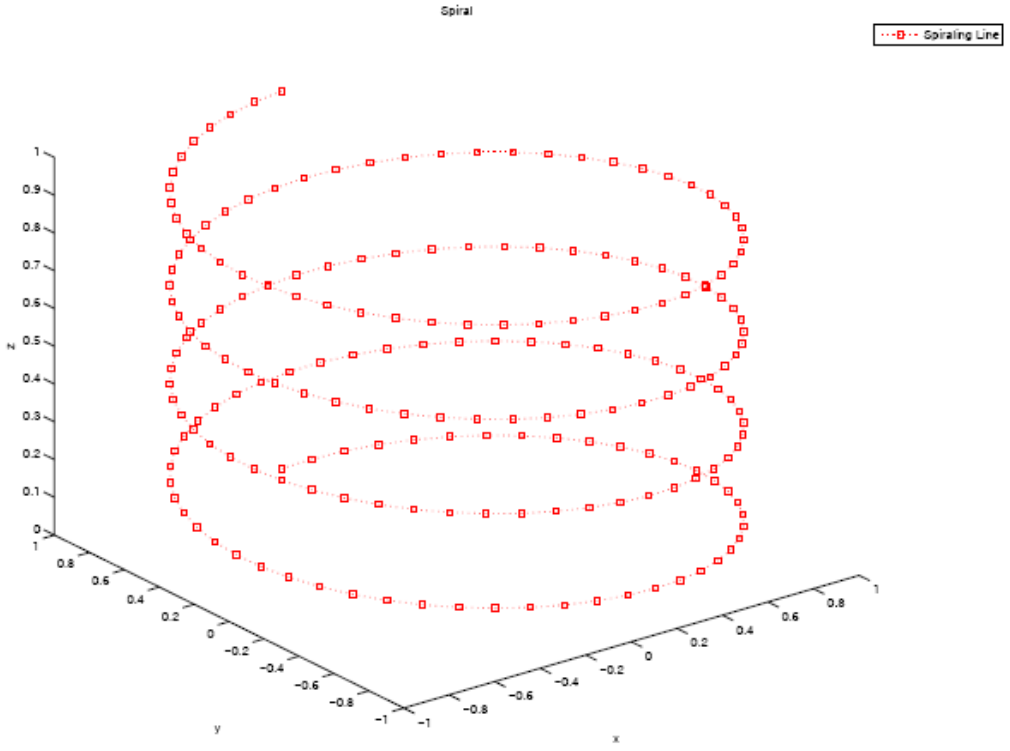
Резултатите от този кодов блок, ще може да видите на *фигура 12.3*.



Фигура 12.1. Проста диаграма от три линии. Линиите са построени в диаграмата с командата **plot (x,y1,'rs:',x,y2,'bo-.',x,y3,'kp- -').**



Фигура 12.2. Надписва диаграма от три линии. Вие би трябвало да бъдете сигурни да надпишете четливо осите и да осигурите заглавие и легенда, така че друг да може да разбере съдържанието на Вашите фигури.



Фигура 12.3. 3D спирална диаграма.Триизмерните линии могат да бъдат направени на диаграма използвайки *plot3*. Тази линия беше направена с командата **plot3(x,y,z,'rs:');.**

12.4 Scatter (разсейвам)

scatter , както повечето графични функции в Матлаб е по същество само описателна. Тя създава разсеяна диаграма от елементи на вектор *x* срещу елементите на вектор *y*. като командата **plot** и командата *scatter* взема допълнително трети аргумент за контрол на форматирането, който е от формата *"color", "shape". "color"* и *"shape"* са същите, както по-горе. Форматирането на цвят или пазарна форма, може само да бъде променена или чрез използването на handle graphics, или ръчно създаване на диаграма. Прост пример за тези са включват в края на този раздел, но Вие можете да се консултирате чрез използването на *help* файла в Матлаб за по подробна информация, за използването на командата **scatter** .

Следващият код създава такава **scatter** диаграма от 1000 псевдо случайни числа от нормално разпределение, всяко с единичен вариант и корелация от 0,5. Резултатите от този код могат да бъдат видени във *фигура 12.4*.

```
figure(4)
x=randn(1000,2);
Sigma=[2 .5;.5 0.5];


x=x*Sigma^(0.5);
scatter(x(:,1),x(:,2),'rs')
xlabel('x')
ylabel('y')
legend('Data point')
title('Scatter plot of correlated normal random variables')
```

12.5 Surf (плъзгам се, сърфирам)

Следващите три графични инструмента, създават диаграма на матрица с *z* данни, срещу вектор на *x* и *y* данни. Ще бъде използвана дву вариантна нормална вероятностна гъстота на функция, за да илюстрираме. *PDF* на би-вариантната нормала е зададен чрез:

$$f_X(x) = -\frac{1}{2\pi|\Sigma|^{-\frac{1}{2}}}\exp(-\frac{1}{2}x'\Sigma^{-1}x)$$

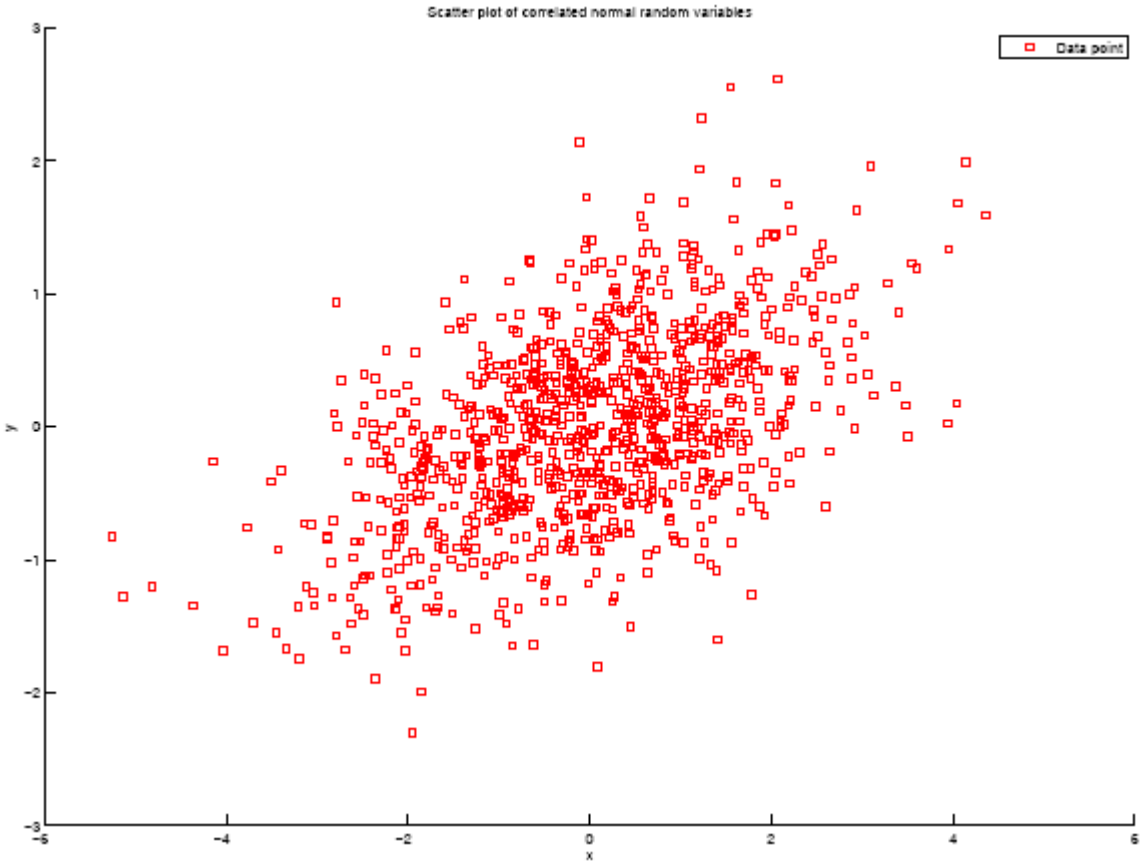
В този пример, *ко*-вариантната матрица *Σ*, беше избрана:

$$\Sigma = \begin{bmatrix} 2 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

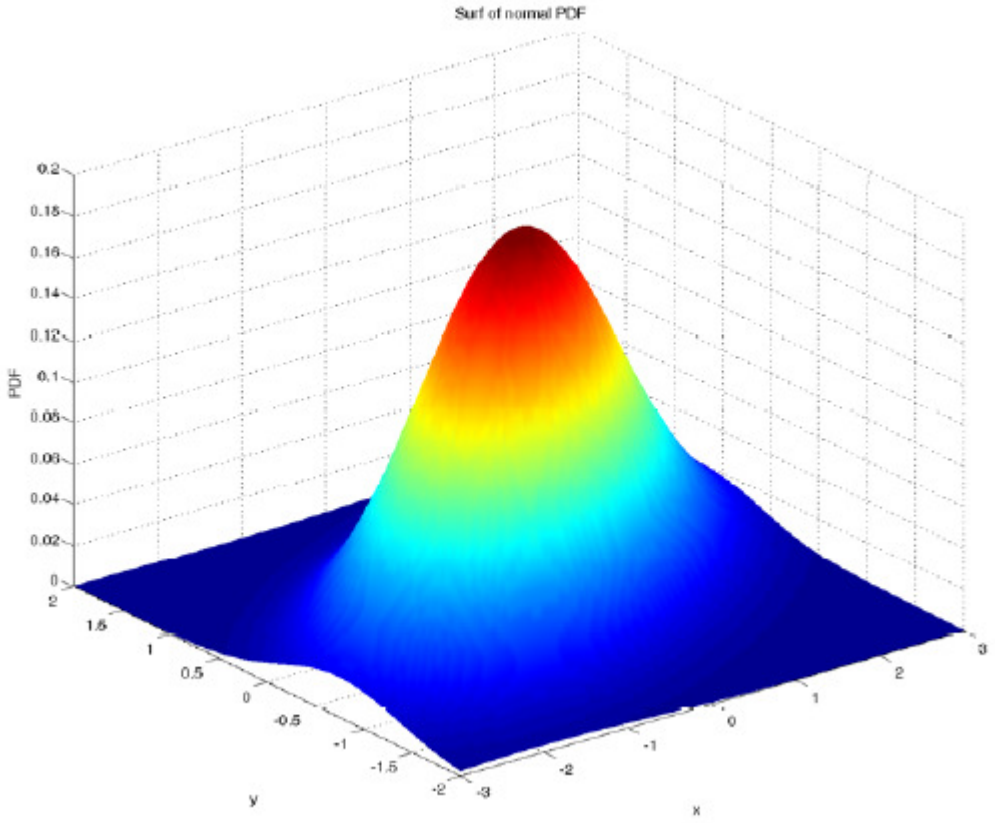
Матрицата от *PDF* стойности, **pdf** беше създадена със следния код:

```
N = 100;
x = linspace(-3,3,N);
y = linspace(-2,2,N);

pdf=zeros(N,N);
for i=1:length(y)
    for j=1:length(x)
        pdf(i,j)=exp(-0.5*[x(j) y(i)]*Sigma^(-1)*[x(j) y(i)]')/sqrt((2*pi)^2*det(Sigma));
    end
end
```



Фигура 12.4. scatter диаграма. Тя съдържа разсеяна диаграма на би-вариантна нормала от случайни отклонения с единица вариант и корелация от 0,5. Тази линия беше извикана чрез командата **scatter(x(: , 1) ,x(: , 2) , 'rs');.**



Фигура 12.5. Surf диаграма. **surf** създава диаграма на 3D повърхност от вектор на *x* и *y* данни и матрица на *z* данни. Това surf съдържа PDF би-вариант на би-вариантна нормала и беше създадена с използването на **surf(x,y,pdf),** където *x, y* и *pdf* са дефинирани в текста.

Първите две линии инициализират стойностите *x* и *y*. Понеже *x* има по-висока варианта, той има по-голям обхват. Командата **surf** (*фигура 12.5.)* е създадена чрез:

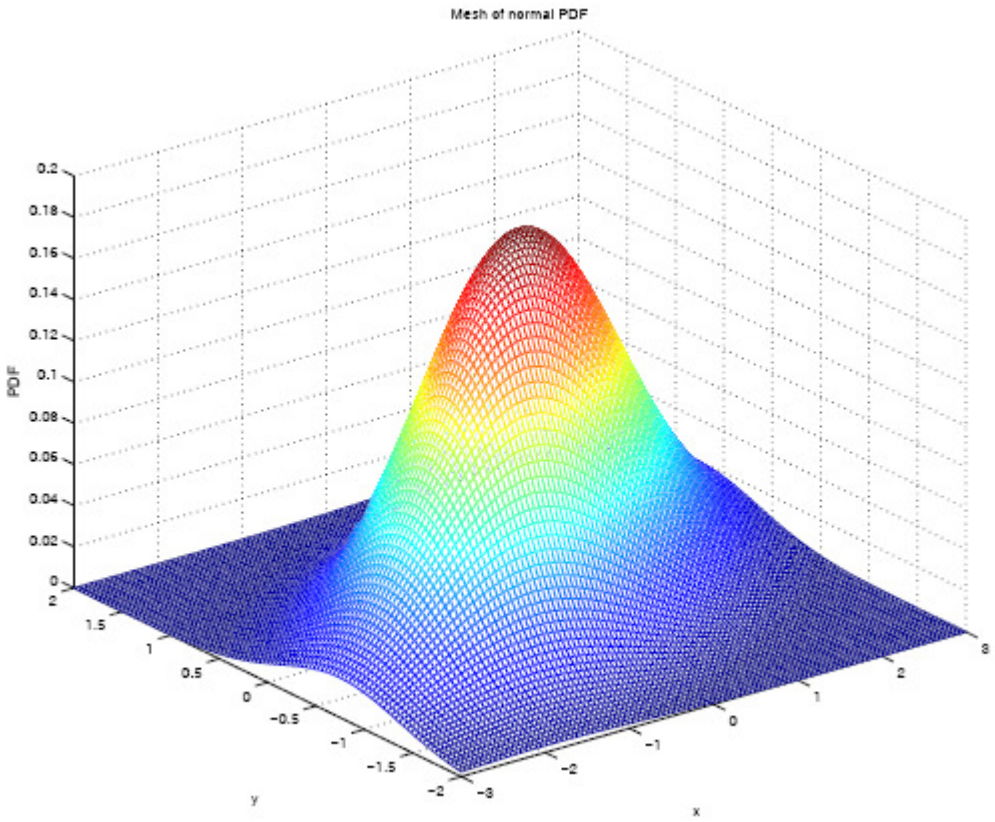
```
surf(x,y,pdf)
xlabel('x')
ylabel('y')
zlabel('PDF')
title('Surf of normal PDF')
shading interp
```

Линията на *shading interp* (сянкова интерпретация) се променя, като цветовете, са приложени като дискретна форма на мрежа на продължителна мрежа. Аз я предпочитам, а Вие може и да не.

Забележка: *x* и *y* аргументите на **surf**, трябва да свържат измеренията на *z* аргумента. Ако **[M,N]=sizez**, тогава **length(y)** трябва да бъде *M*, а **length(x)** трябва да бъде *N*. Това е вярно за всички 3D диаграмни форми, където се рисуват матрични данни. В кода по-горе **i** е редица *iterator,* която съответства на *y* и *j* е колона *iterator* съответстващ на *x*.

12.6 Mesh (отвор, клетка от мрежа)

Командата **mesh** създава графика подобна на **surf**, но с празно пространство между точките на решетката. **mesh** има предимство, че *hidden* страна може да бъде видяна, потенциално представлява, развива повече от единична графика. То също създава много по-малки файлове, които могат да бъдат важни, когато включвате мултипликационни графики в презентации или доклад. Използвана в същото би-вариантна нормала, следни код създава **mesh** диаграма, посочена на *фигура 12.6*.

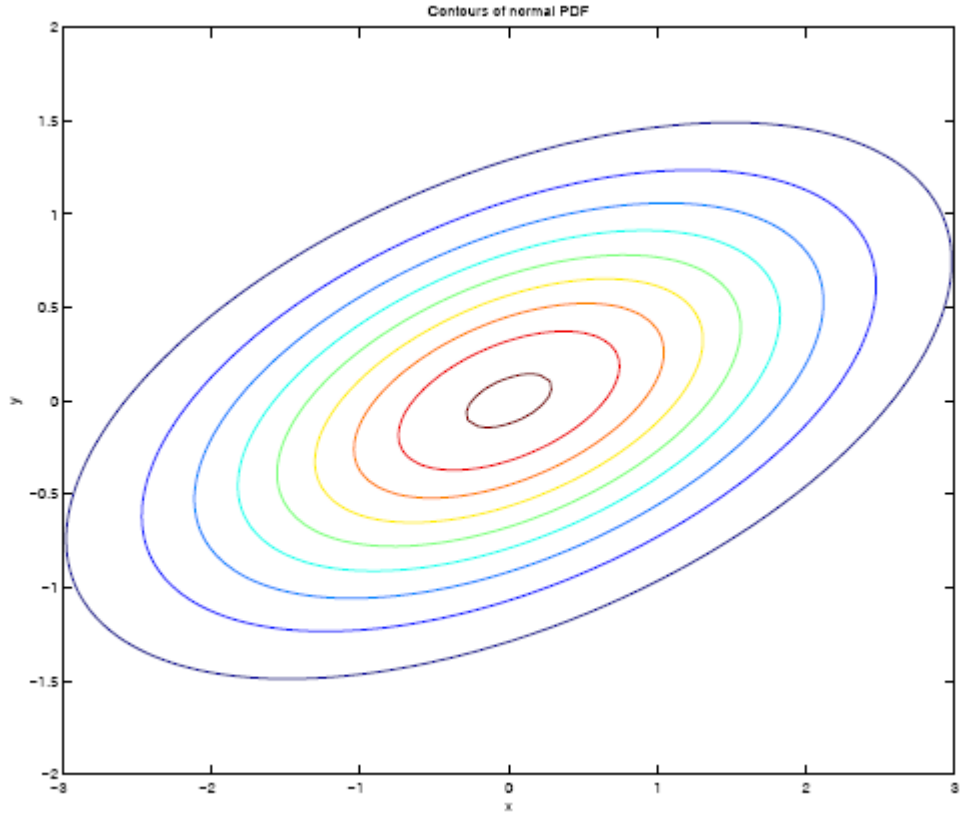


Фигура 12.6. mesh диаграма. *Mesh* създава фигура подобна на **surf**, но с празнини между точките на решетката, което позволява задната страна на фигурата да бъде видяна с един поглед. Тази *mesh* съдържа *PDF* би-варианта на би-вариантна нормала, и беше създадена, използвайки **mesh(x,y,pdf),** където *x* и *y* и *pdf* са дефинирани в текста.

```
mesh(x,y,pdf)
xlabel('x')
ylabel('y')
zlabel('PDF')
title('Mesh of normal PDF')
```

12.7 Contour (контур, очертание)

contour е подобен на **surf** и **mesh** заради трите аргумента *x, y* и *z*. Обаче,той се различава в това, че той създава 2D диаграма. **contour** диаграмите въпреки, че не са толкова хващащи окото, както **mesh** диаграмите, те често са по приемливи и по-добри в предаването на информация. *Contour* диаграмите могат да бъдат наричани или **contour(x,y,z)** или **contour (x,y,z,N),** където N инструктира Матлаб колко контури да създаде. Ако е пропуснато Матлаб ще определи броя на контурите на базата на вариантите на *z* данните. Кодът по-долу и *фигура 12.7* демонстрират ползата от **contour**.



Фигура 12.7. contour диаграма. **contour** диаграмата е мрежа от филийки през **surf** диаграмата. Тази конкретно съдържа изо-възможно линии от разпределението на би-варианта нормала със значение нула, варианти от 2 и 0,5 и корелация от 0,5.

```
contour(x,y,pdf);
xlabel('x')
ylabel('y')
title('Contours of normal PDF')
```

12.8 Subplot (субдиаграма)

Субдиаграмите позволяват мултипликационни графики да бъдат поставени в същата фигура. Всички повиквания към субдиаграмата трябва да съдържат три аргумента: брой на редовете, брой на колоните и в коя клетка да поставим графиката.

Основната форма е:

subplot (M, N, #).

където *M* е броя на редовете, *N* е броя на колоните, а # показва клетката за поставяне на графиката. Клетките в субдиаграмите се броят по-същия начин, както клетките в матрица, първо надолу, а после напряко. Например, при извикване към **subplot (3,2,#),** # би бил

1	4
2	5
3	6

Извикване към **subplot** , би трябвало да бъде незабавно последвано от някакви диаграмни функции, в най-простия случай това, би били извикване към диаграма. Обаче, която и да е графична функция в Матлаб, може да бъде използвана в **subplot.** Кодът по-долу и *фигура 12.8*., показват как различни графични функции могат да бъдат използвани във всяка клетка. Те също показват, някои от многото налични графични функции, не описани в тези бележки.

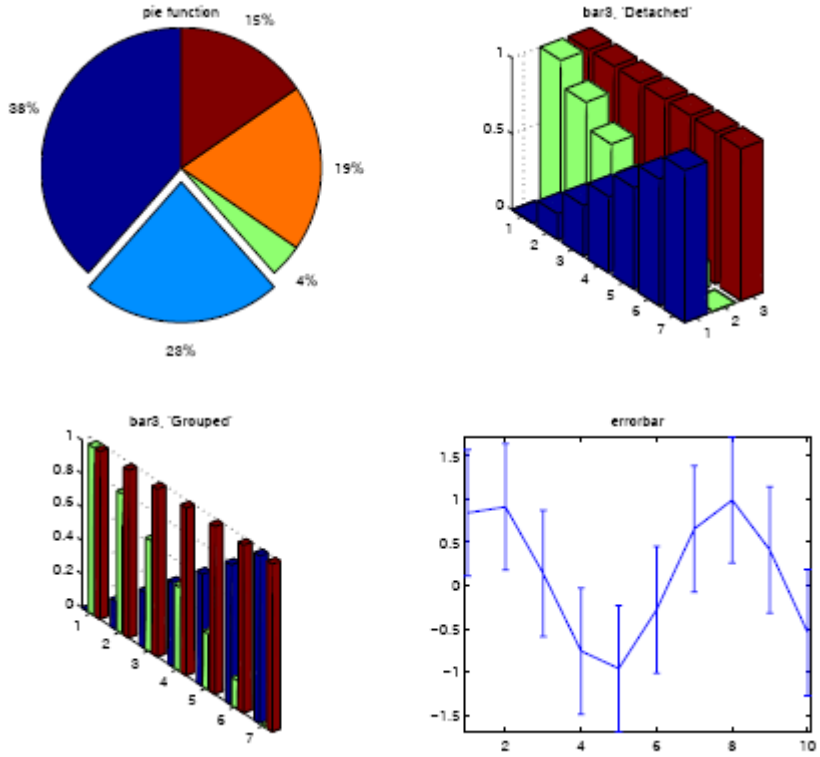
```
subplot(2,2,1);
x = [5 3 0.5 2.5 2];
explode = [0 1 0 0 0];
pie(x,explode)
colormap jet
title('pie function')
axis tight

subplot(2,2,2);
Y = cool(7);
bar3(Y,'detached')
title('Detached')
title('bar3, ''Detached''')
axis tight

subplot(2,2,3)
bar3(Y,'grouped')
title('bar3, ''Grouped''')
axis tight

subplot(2,2,4);
x = 1:10;
y = sin(x);
e = std(y)*ones(size(x));
errorbar(x,y,e)
title('errorbar')
axis tight
```

Забележка: Графичния код във всяка субдиаграма беше взет направа от *help* файловете на Матлаб.



Фигура 12.8. Субдиаграма –пример. Субдиаграмите позволят повече отколкото една графика, да бъде включена във фигура. Тази конкретна субдиаграма съдържа три различни типа графики, с два варианта на триизмерна решетка. В горния ляв ъгъл съдържа извикване на **pie** (питка), в горната дясна част съдържа извикване към **bar3** специфицираща опцията *'grouped'* (групиране), в долната лява част съдържа извикване към **bar3** специфицираща опциите *'detached'*(отделен) и в долната дясна част съдържа резултатите към извикването на грешна решетка *errorbar.*

Help системата е много разбираема и илюстрира повечето функции с примерен код.

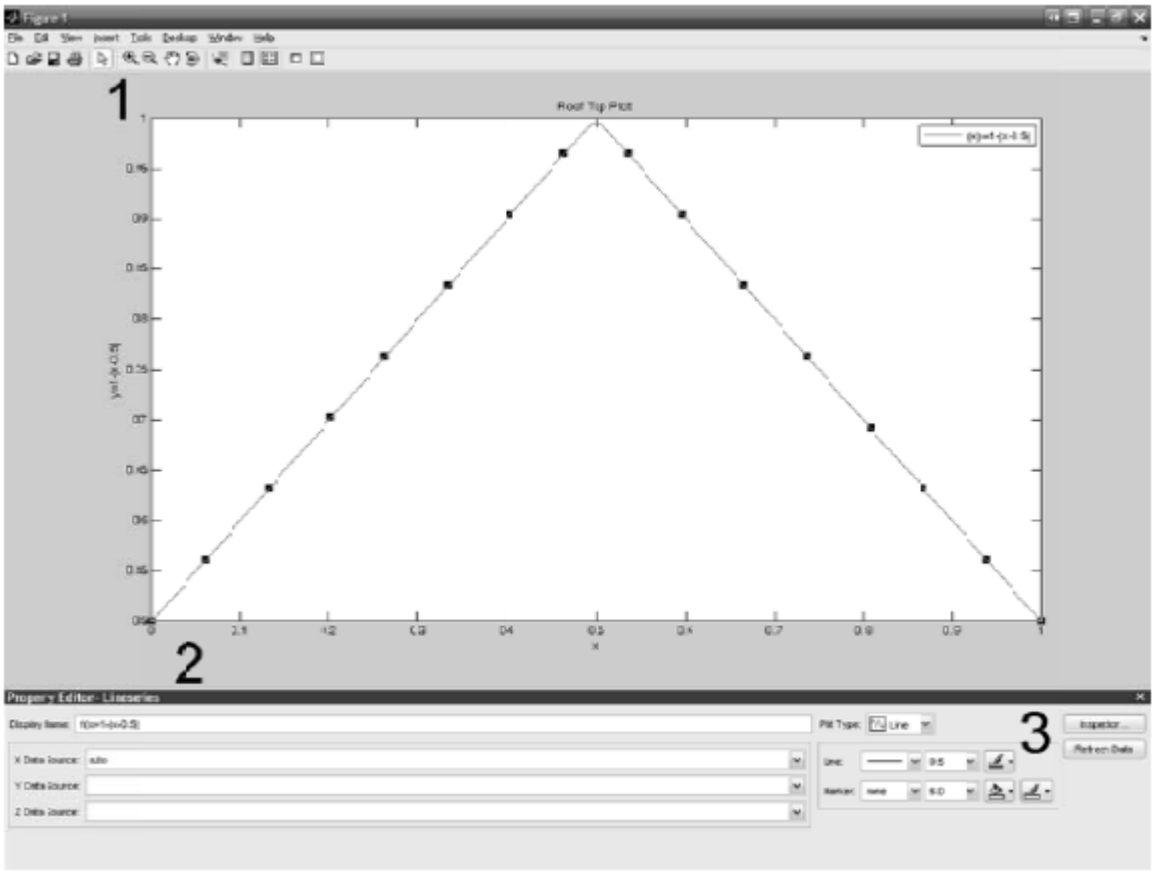
12.9 Advanced (подобрени) графики

Докато стандартните графични функции на Матлаб са много мощни и могат директно да завършат много задачи, понякога те не са достатъчни. Например, ако Вие искате да промените плътността (дебелината) на линията, за да подобрите нейния вид в доклад или добавяте стрелка за да подчертаете определена конкретна особеност от данните. За щастие Матлаб осигурява механизъм за добавяне на почти всичко към една диаграма. Фактически

има две. Първата към, на която ще се позова е *point-and-click* (посочваш и кликваш) е ръчно редактираща диаграмата в прозореца на фигурата. Втората способна на всичко, което Матлаб е способен, е позната като *handle graphics*. Те осигуряват механизъм за програмна промяна на каквото и да е относно някоя графика, дори след като тя е била нарисувана.

12.9.1 Point-and-click

Най-простият метод да подредите вашите графики е да използвате редакторските възможности на фигурните прозорци – директно. Изучавайки редакторният прозорец Вие би трябвало да видите известен брой бутони. Един от тези е стрелка, **1** във *фигура 12.9*. Кликвате върху стрелката и ще Ви позволи да кликнете върху който и да е елемент във Вашата диаграма, такъв като една линия. Двойното кликване върху линията ще доведе редактора на свойствата (**2**), диалог, който съдържа елементи от избрана тема, която Вие можете да промените, като например цвят, ширина на линията или маркер (**3**). За повече информация в редактиране на диаграми потърсете в *help*-а на Матлаб.



Фигура 12.9: point-and-click редактиране. Повечето особености на една диаграма могат да бъдат редактирани чрез използ0ване на интерактивни редактиращи инструменти на фигурния прозорец. Интерактивното редактиране се започва чрез първо избиране на иконата стрелка по горния край на фигура (**1**), след това кликвайки върху елемент, който трябва да

бъде редактиран (примерно линията, осите, тестови надпис). Това ще доведе редактора на свойства (**2**), където тематичните специфични свойства могат да бъдат променени (**3**).

12.9.2 Handle graphics (ръчни графики)

Част от мощта на Матлаб графики, е че те са напълно програмируеми. Какъвто и да е, където Вие може да завършите, използвайки ръчно редактиране на графика, може да бъде използвано чрез използване на *handle graphics*. Когато Матлаб рисува нещо, то е определено ръчно. Ръчното съдържа всичко, дето трябва да се знае относно конкретната графика, като цветове или ширина на линията. Щом се запознаете с *handle graphics* , те могат да бъдат използвани за създаване на ефектни комплекти графики. Обаче, аз ще илюстрирам тяхната употреба чрез прост пример.

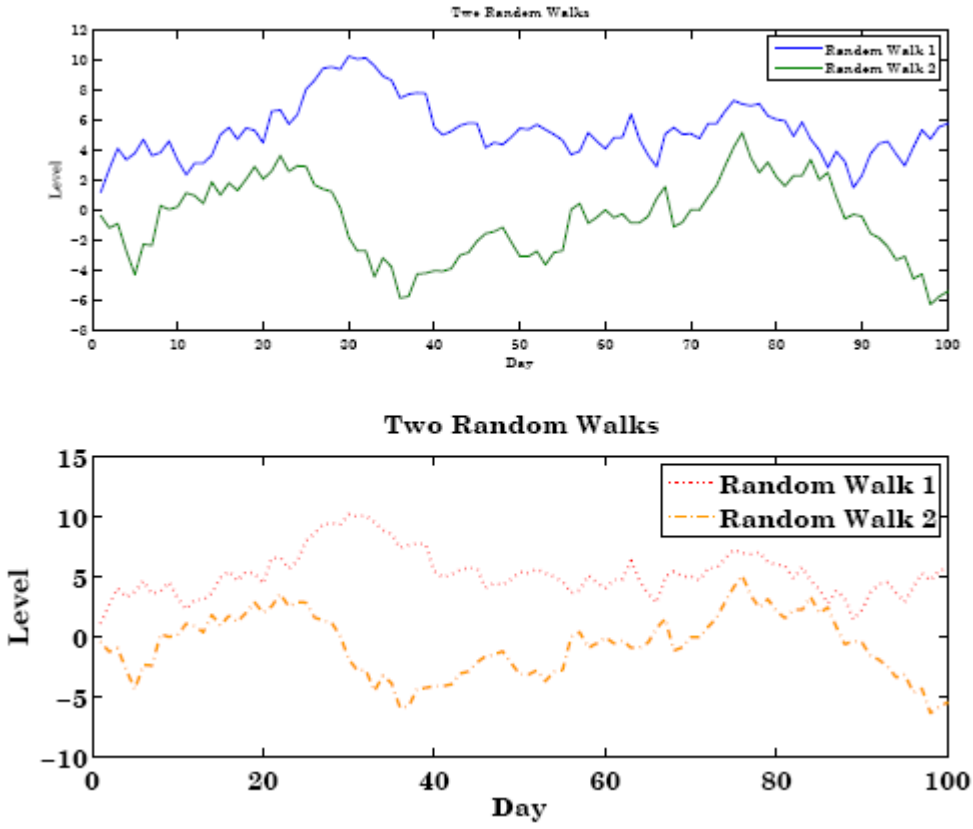
Примерът ще илюстрира използването на *handle graphics* , чрез показване както преди, така и след графики използване на субграфика.

```
e  = randn(100,2);
y  = cumsum(e);
subplot(2,1,1);
plot(y);
legend('Random Walk 1','Random Walk 2')
title('Two Random Walks')
xlabel('Day')
ylabel('Level')

subplot(2,1,2);
h  = plot(y);
l  = legend('Random Walk 1','Random Walk 2')
t  = title('Two Random Walks')
xl = xlabel('Day')
yl = ylabel('Level')
set(h(1),'Color',[1 0 0],'LineWidth',2,'LineStyle',':')
set(h(2),'Color',[1 .6 0],'LineWidth',2,'LineStyle','-.')
set(t,'FontSize',14','FontName','Bookman Old Style','FontWeight','demi')
set(l,'FontSize',14','FontName','Bookman Old Style','FontWeight','demi')
set(xl,'FontSize',14','FontName','Bookman Old Style','FontWeight','demi')
set(yl,'FontSize',14','FontName','Bookman Old Style','FontWeight','demi')
parent = get(h(1),'Parent');
set(parent,'FontSize',14','FontName','Bookman Old Style','FontWeight','demi')
```

В основата всичко дето може да бъде извършено чрез *handle graphics* може да бъде направено използвайки редакторният метод *point-and-click*, подчертан по-горе. Обаче, предимството на *handle graphics* е по очевидно, ако имате нужда да промените една фигура. Да допуснем, че Вие решите да добавите нови серии. Ако използвате *handle graphics* , Вие се нуждаете само да осъвремените кода и да върнете. Ако вие използвате *point-and-click* метода, Вие трябва да приложите отново всяка промяна, която Вие правите към оригиналната диаграма.

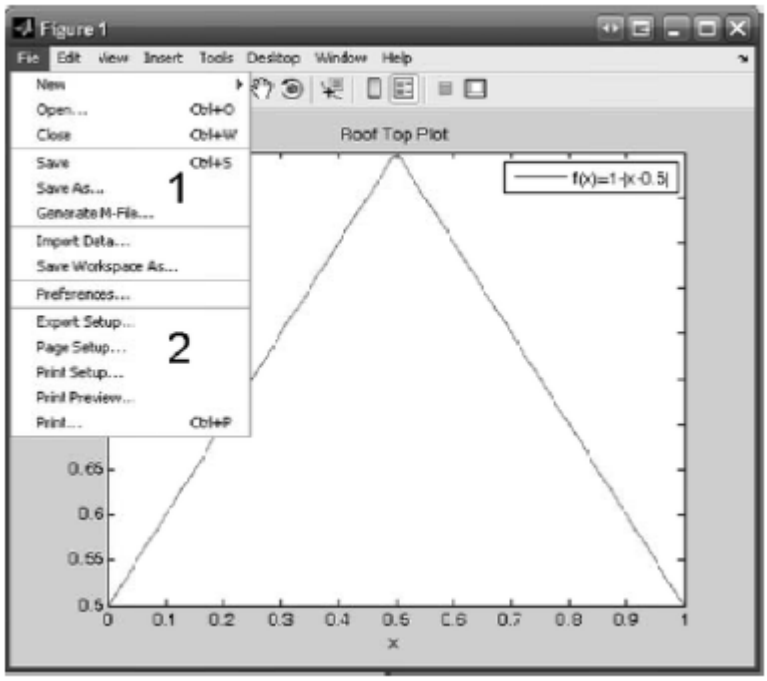
Повече за *handle graphics* моля консултирайте се с *help* файловете на Матлаб.



Фигура 12.10: ***handle graphics***. Субдиаграма от горе е стандартно извикване към **plot**, докато дъното подчертава някои от възможностите, когато са използвани *handle graphics*. Нищо не струва да покажем всички тези промени очевидни в субдиаграмата отдолу, която може да бъде репродуцирана с използване на метода *point-and-click.*

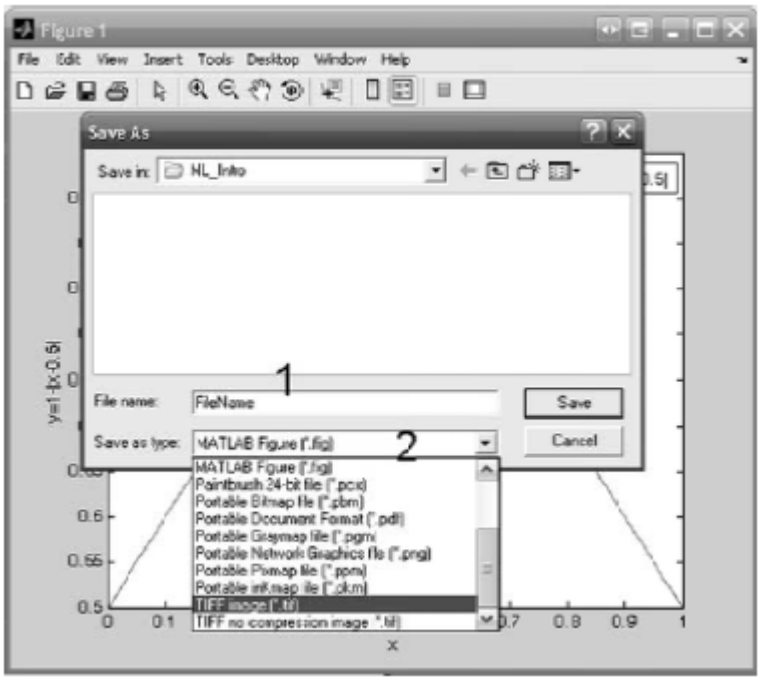
13. Exporting Plots (изнасяне на диаграми)

Щом сте създали диаграмата, която желаете, тя трябва да бъде *exporting* (изнесена), за да бъде включена в задание, доклад или проект. Изнасянето (*exporting)* става право напред. На фигурата кликвате *File*, *Save As* (**1** във *фигура 13.1*). В *Save as type box* избирате желаният формат (Tiff за Microsoft Office употреба, EPS файл за LATEX (**2** *фигура 13.2*)), въвеждате името на файла (**1** във *фигура 13.2*) и запазвате (save). *Фигурите 13.1* и *13.2* съдържат репрезентации на стъпките необходими за *exporting* от фигурната кутийка.



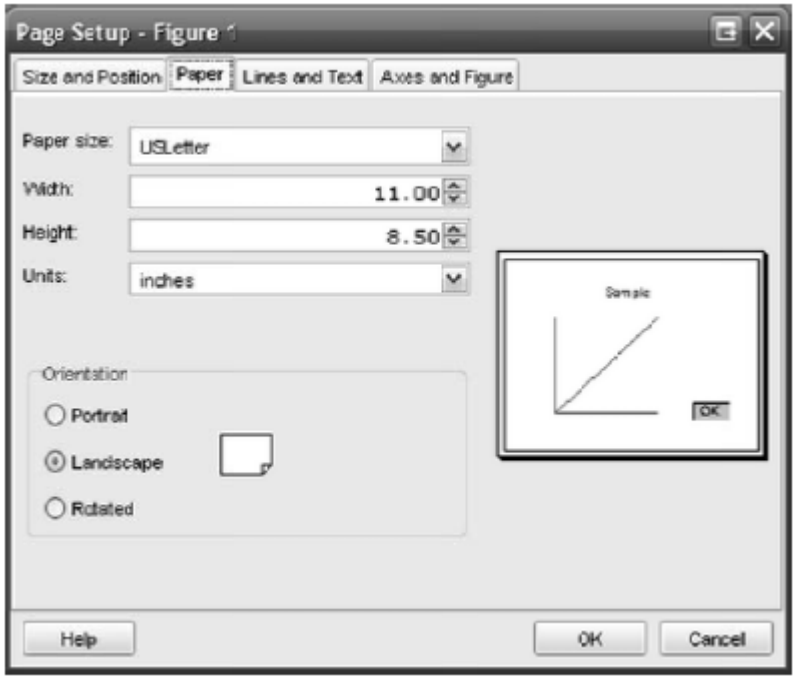
Фигура 13.1: Стъпки за *exporting* на фигурата. За да се *exporting* кликваш на *Save As*, *File* менюто на фигура **(1**). Диалога във *фигура 13.2* ще се появи.

Ако *exporting* фигурата не е точно, каквото Вие искате, Вие може да се наложи да промените някой опции в *Page Setup* (**2** във *фигура 13.1*). като особеност вие може да искате да промените настройката на листа , като след това кликвате *Fill Page*, *Fix* съответно с ratio (ниво) и *Center* (център) (1 ,2 и 3 *фигура 13.4*) върху Size (размер) и Position Tab(позиция). *Фигура 13.3 и 13.4* съдържат представяне на *Page Setup*, нужните екрани за да се промени размера на страницата за *exporting*.

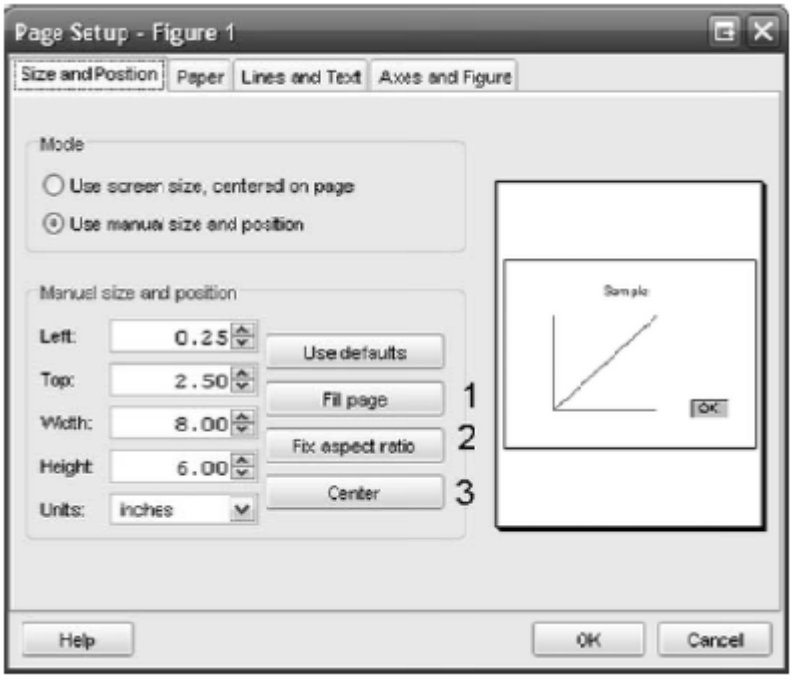


Фигура 13.2: ***Save As*** диалогов прозорец.

Забележка: Фигурите могат да бъдат *exporting* програмируемо, използвайки командата **print**.



Фигура 13.3: Page Setup, Page Tab. Ако Вашите *exporting* са прекалено малки, вие би трябвало да промените настройката на листа до пейзаж и тогава да промените размера *Size* и *Position tab* (*фигура 13.4*).



Фигура 13.4: ***Page Setup, Size*** **и** ***Position tab***. След промяна на ориентацията на листа към пейзаж (*фигура 13.3*), вие би трябвало да се убедите, че страницата е запълнена чрез натискане на (**1**) *Fill Page*, (**2**) Fix по отношение на *ratio* и после (**3**) *Center*. Това ще създаде големи високо качествени из фигури, които могат да бъдат повторно оразмерени в Office и LATEX.

Раздел 14
Custom Functions (обичайни функции)

В допълнение към писмените *batch file* (група файлове) и извикване на предефиниране функция, Матлаб позволява да пишеш твой собствени функции, за да представиш повторни задачи или да използваш като обект на рутинна оптимизация. Всички функции в Матлаб започват с линията на формиране *function [out1, out2, …] = functionname (in1, in2, …),* където *out1, out2,…* са променливи, които функцията връща към командния прозорец, *functioname* е името на функцията, а *in1, in2, …* са входящи променливи. Очевидно функциите могат да вземат множествени входове и ще върне множествени изходи, обаче за да започнем, взимаме в предвид тази пробна функция:

```
function y = func(x)
x = x + 1;
y = x;
```

Тази функция, която не е особено добре написана[1], взима един вход и връща един изход, увеличавайки входната променлива (независимо дали е скаларна, вектор или матрица) с едно.
Функциите имат няколко важни различия от стандартния *m-file* писане.

Функциите оперират върху копие от оригиналните данни. При това Вие можете да ползвате отново същите променливи имена вътре и отвън на функцията рисувайки вашите данни[2].

Както и да са променливите създадени, когато функцията тече, или някакви копия на променливи направени за функцията, са загубени, когато функцията се изпълни.

Във функцията по-горе това означава, че само стойността на *y* е върната, а всяко друго е загубено. Особени промени в *x* не присъстват. Например, до предположим, че Вие въведете следното в Матлаб:

[1] Без коментар, има излишни (ненужни) команди, би трябвало само да има y=x+1;
[2] за някои с познания в програмирането, Матлаб използва модел copy-on-change, където данните само се копират, ако модифицираме. Ако не е, променливите преминават към функции, както се държат, като че ли преминават по компетентност. (чрез препоръка, отношение, пълномощия, еталон, позоваване, препратка).

```
>> x = 1;
>> y = 1;
>> z = func(x);
>> x
ans =
    1
>> y
ans =
    1
>> z
ans = 2
```

При това независимо, че функцията използва имена на променливи *x* и *y*, стойностите на *x* и *y* в работно пространство не се променят, когато функцията е извикана.
Функции с множествени входове и изходи могат също да бъдат изградени. Един пост пример е зададен чрез :

```
function [xpy, xmy] = func(x,y)
xpy = x + y;
xmy = x - y;
```

Тази функция взема два входа и връща два изхода. Важно е да отбележим, че независимо от двата изхода от тази функция, Вие не винаги трябва да я извиквате, очаквайки и двата. Например, взимаме предвид следната употреба на тази функция

```
>> x = 1;
>> y = 1;
>> z1 = func(x, y);
>> z1
ans =
    2
>> [z1, z2] = func(x ,y);
>> z1
ans =
    2
>> z2
ans =
    0
```

Има известен брой предварителни функции, специфични променливи, намиращи се за да прехвърлят параметрите на средата, такива като , колко входящи променливи са били осигурени към функцията (**nargin**), колко изходящи са били потърсени (**nargout**), което позволява променливи числа на входове и изходи (**varargin** и **varargout**, съответно) и позволяват ранно унищожаване на функцията *(return*). Ти би трябвало да можеш да усъвършенстваш, в каквото и да е в този курс, без да използваш някоя от тях, но не са в наличност, ако ти би имал нужда от тях.

14.1 Comments (коментари)

Файловата група от *m-files* са коментирани в обичайните функции, са също направени с използване на символът %. Обаче коментарите имат една допълнителна цел в обичайната функция. Когато и да въведете *help function,* Матлаб ще покаже на екран първия блок от коментари в командния прозорец. Например, функцията **func** е дадена чрез:

```
function y = func(x)
% This function returns
% the value of the input squared.

% The next block of comments will not be returned when
% you enter 'help func'
% This line does the actual work.
y=x.^2;
```

въвеждане на **help func,** би върнало:

```
>> help func

    This function returns
    the value of the input squared.
```

Първоначално коментарите обикновено съдържат възможните комбинации на вход и изход аргументи, а също и описание на функцията. Докато те са стриктно допълнение те би трябвало да бъдат включени, както за да подбутне твоята палитра и да помогне , ако ти разделиш твоята функция.

14.2 Debugging (деблокиране)

Понеже данните модифицирани в твоята функция не са в наличност, когато ти активираш твоята функция, деблокирането може да бъде до някъде трудно. Има три основни стратегии за деблокиране на твоята функция:

- Напишете Вашата функция като ръкопис, а после я превърнете във функция.

- Преустановявате; понеже е необходимо да изпишете стойности на променливите в командния прозорец. Като алтернатива може да използвате *disp.*

- Използвайте **keyboard** и **return** за да прекъснете вашата функция да проверява стойностите.

Първият от тези методи е често най-лесният. Вземете под внимание ръкописната версия на функцията по-горе:

```
x = 1;
y = 2;
%function [xpy, xmy] = func(x,y)
xpy = x + y;
xmy = x - y;
```

Действието с този ръкопис би било еквивалент на извикването на функцията **func(1, 2**). Обаче, когато я извика като ръкопис, ти можеш да изследваш променливите, докато те се променят. Вторият метод може да бъде полезен, но е тромав. Често изходния прозорец става запълнен с числа, така че Вие не ще можете да намерите проблема-точното число. Третият е най-напредничав. Добавяйки **keyboard** към някоя функция, инструктираме Матлаб да прекъсне функцията и да върне контрола към клавиатурата. Когато сте в тази ситуация обичайните >> се променят към *K*>>. Когато Вие сте в настройка клавиатура, Вие можете да взаимодействате с променливите вътре във вашата функция, като че ли те са били ръкописни променливи. Щом свършите с проверката на променливите въведете **return** за да продължите изпълняването на функцията. Един прост пример на командата **keyboard**, може да бъде приложен към функцията по-горе:

```
function [xpy, xmy] = func(x,y)
keyboard
xpy = x + y;
xmy = x - y;
keyboard
```

Извикването на тази функция ще резултира в незабавна клавиатурна сесия (отбележете *K*>>*).* Въвеждането на *whos* ще изброи две променливи, *x* и *y*. Ако Вие въведете *return*, ще видите втора клавиатурна сесия да се отваря. Въвеждането на *whos* сега ще изброи четири променливи, оригиналните две и *xpy* и *xmy*. Когато Вие успешно си деблокирате функцията, като коментарите на командите **keyboard,** така и на премахването им изцяло.

15. Вероятност и статистически функции)

Матлаб чрез статистически инструментариум, съдържа увеличен обхват на статистическа функция. Повечето, са обикновена статистика, а не егонометрична[3]. Обаче, има някакво съображение, което си струва да се вземе предвид.

15.1 quantile (квантил)

Командата **quantile** връща емпиричния квантил на вектор. Обаче неговата функция е проста и може лесно да бъде заменена чрез използването на **sort, length и floor или ceil**.

15.2 prctile (пръктайл)

Командата **prctile** е идентична с **quantile**, с изключение на това, че тя очаква по-скоро аргументи от 0 до 100, отколкото аргументи между 0 и 1.

15.3 regress (връщане назад)

Командата **regress** представлява основно връщане и връща клавиша регресия статистика. Не съм голям фен на Матлаб приложението и препоръчвам да пишете на ваша собствена регресивна функция, като просто учебно упражнение.

15.4 *cdf, *pdf, *rnd, *inv

Най-стойностния код в статистическия инструментариум са CDF, PDF генератори на случаен номер и инвестира (разменя) CDF-та съдържание вътре в него. Повечето разпределения, с които Вие ще се сблъскате в този курс (и където и да е другаде) има пълен комплект от четири осигурени включително:

[3] Надяваме се, че Вие сте узнали разликата между статистика и егонометрика. Ако не, би трябвало да я научите.

- χ^2 (chi2-)
- β (beta-)
- Exponential (exp-)
- Extreme Value (ev-)
- F (f-)
- Γ (gam-)
- Lognormal (logn-)
- Normal (Gaussian) (norm-)
- Poisson (poiss-)
- Student's T (t-)
- Uniform (unif-)

15.5 The JPL Toolbox

JPL инструмент се намира в наличност на http:\\ www.spatial-ecnometics.com , съдържа много егонометрични функции,написани от академици, което е най-доброто – той е безплатен. Той също има много полезни диаграмни функции, такива като **pltdens**, която показва на диаграма (графика) сърцевината (същността) гладкост на емпиричната гъстота (тъпотия, глупост). Аз предлагам да го свалите преди да пишете Ваши собствени функции за да избегнете ненужна намеса.

16 Optimization (оптимизация)

Оптимизацията на инструментите съдържа голям брой числови функции за оптимизиране на обичайните действия. Повечето от тях представляват форма на алгоритъм. Нютон-Рафсън, който използва производни за да открие **минимума** на една функция.

Забележка: Матлаб може само да открие минимумите, така че ако *f* е функция, която желаете да максимализирате, - *f* е функция с минимум на същата точка, както максимума на *f*.

За да използвате Матлаб за оптимизация на функция (например, *log*-подобна или GMM квадратна форма) Вие трябва първо да напишете основна функция, която връща стойността на функцията в мрежа от параметри. Всички оптимизационни цели, трябва да имат параметрите, като на първия аргумент. Например, визамаме предвид намирането на минимума на x^2 . Функцията, която би позволила на Матлаб оптимизатора да работи правилно и би имала формата:

```
function x2 = optim_target(x)

x2=x^2;
```

Когато имате множество параметри (или вектор-параметър), Вие трябва да използвате функцията от формата:

```
function obj = optim_target(params)

x=params(1);
y=params(2);

obj= x^2-3*x+3+y*x-3*y+y^2;
```

Оптимизационните цели могат да имат допълнителни входове, като например:

```
function obj = optim_target(params,hyperparams)

x=params(1);
y=params(2);

c1=hyperparams(1);
c2=hyperparams(2);
c3=hyperparams(3);
obj= x^2+c1*x+c2+y*x+c3*y+y^2;
```

Тази форма е особено полезна в иконометрията, където оптимизационната функция обикновено изисква поне два входа с параметри и данни. Щом сте определили функцията, оптимизационната цел, следващата стъпка е да оставите един от Матлаб оптимизатори да намерят минимума.

16.1 fminunc

fminunc представлява производна основна не принудителна минимизация. Производните могат да бъдат осигурени от ползвател или изчислени числово (чрез Матлаб). Основната форма **fminunc** е: *[p, fval, exitflag]= fminunc ('fun' ,p_0, options, var_1, var_2, …)* ,

където *fun* е оптимизационна цел, p_0 е вектор от начални стойности, *options* е ползвател доставящ допълнителни структури (виж.16.5), а var_1, var_2, … са (опционни) променливи за употреба на данни. Типично три изхода са необходими, параметрите на оптимимума (*fval*) и *flag* да те информират, ако Матлаб вярва, че оптимизацията е била успешна (exitflag). На примера, да предположим:

```
function obj = optim_target(params,hyperparams)

x=params(1);
y=params(2);

c1=hyperparams(1);
c2=hyperparams(2);
c3=hyperparams(3);
obj= x^2+c1*x+c2+y*x+c3*y+y^2;
```

е била нашата обективна функция (и е била спестена като *optim_target.m*). За да минимизирате функцията, Вие бихте извикали:

```
>> options = optimset('fminunc');
>> options = optimset(options,'Display','iter');
>> p0 = [0 0];
>> hyper = [-3 3 -3];
>> [p,fval,exitflag]=fminunc('optim_target',p0,options,hyper)
```

който създава

```
   0
exitflag =
    1
```

fminunc е минимума на тази функция и е върнала оптимизационна стойност на 0 при [11] и *exitflag* на 1 показва, че оптимизацията е била успешна.

16.2 fminsearch

fminsearch също представлява непринудена оптимизация, но използва дериватевен свободен метод (използва симплекс). **fminsearch** използва виртуална амеба да пълзи наоколо в пространството на параметъра, дето винаги ще задвижи към по-ниски стойности на обективната функция. **fminsearch** има същата основна форма като **fminunc**

```
[p,fval,exitflag]=fminsearch('fun',p0,options,var1,var2,...)
```

където *fun* е оптимизационна цел, p_0 е вектор от начални стойности, *options* е ползвател доставящ допълнителни структури (виж.16.5), а var_1, var_2, … са (опционни) променливи за използване на данни. Връщането към предишния пример, но използвайки **fminsearch**

```
>> options = optimset('fminsearch');
>> options = optimset(options,'Display','iter');
>> [x,fval,exitflag]=fminsearch('optim_target',[0 0],options,hyper)
 Iteration   Func-count     min f(x)         Procedure
     0           1               3
     1           3          2.99925         initial simplex
     2           5          2.99775         expand
     3           6          2.99775         reflect
     4           8          2.99475         expand
     5           9          2.99475         reflect
    ...
    ...
    ...
    57          107     8.93657e-009         contract inside
    58          109     3.71526e-009         contract outside
    59          111     1.99798e-009         contract inside
    60          113     5.82712e-010         contract inside
Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-004
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-004
 x =
     1.0000    1.0000
 fval =
   5.8271e-010
 exitflag =
     1
```

fminsearch изисква повече повторения и много повече функционални оценявания и не би трябвало да се използва, ако **fminunc** ще работи. Обаче, при определени проблеми, такива като обективни (реални) функции, които не са способни на диференциране, **fminsearch** е единствената опция.

16.3 fminbnd

fminbnd представлява минимизация на отделни проблеми на параметъра през граничен интервал, използващ златен секционен алгоритъм. Основната форма е:

[p , fval , exitflag] = fminbnd *('fun' ,lb, ub, options, var_1, var_2, …)*

където *fun* е оптимизационна цел, *lb* и *ub* са по-долната и по-горната граница на параметъра, *options* е ползвател представляващ структура на опции (виж 16.5), а var_1, var_2, … са допълнителни променливи за използване на данни.

Понеже **fminbnd** само минимизира инивариат обекти, взимайки предвид минимума към

```
function obj = optim_target2(params,hyperparams)

x=params(1);

c1=hyperparams(1);
c2=hyperparams(2);
c3=hyperparams(3);
obj= c1*x^2+c2*x+c3;
```

and optimizing using `fminbnd`

```
>> options = optimset('fminbnd');
>> options = optimset(options,'Display','iter');
>> hyper=[1 -10 21];
>> [x,fval,exitflag]=fminbnd('optim_target2',-10,10,options,hyper)
Func-count     x          f(x)         Procedure
    1       -2.36068      50.1796       initial
    2        2.36068      2.96601       golden
    3        5.27864     -3.92236       golden
    4              5           -4       parabolic
    5        4.99997           -4       parabolic
    6        5.00003           -4       parabolic
Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-004
x =
     5
fval =
    -4
exitflag =
     1
```

16.4 fmincon

fmincon представлява принудителна оптимизация, използвайки линарна и/или нелинарни ограничения, които могат да бъдат, както равни, така и неравни ограничения. **fmincon** минимизира *f (x)* предмет, до която и да е комбинация от

- $A^{EQ}x = b^{EQ}$
- $Ax \leq b$
- $C^{NEQ}(x) = d^{NEQ}$
- $C(x) \leq d$

където *x* е *K* при *1* параметър вектор, A е *Q x K* матрица, а *b* е *Q x 1* вектор. Във втория комплект от граници, *C(.)* е функция от R^K до R^P , където *P* е броя на нелинарни граници, а *d* е *P x 1* вектор (EQ и NEQ са просто използвани да определят равенството на границите от неравенство на границите).

Забележка: Всички ≥ граници могат да бъдат транферирани в ≤ граници чрез умножение по *-1*.

Основната форма на **fmincon** е:

$$\texttt{[p,fval,exitflag]=fmincon(}'fun',p_0,A,b,A^{EQ},b^{EQ},LB,UB,nlcon,options,var_1,var_2,\ldots\texttt{)}$$

където *fun* е оптимизационна цел, p_0 е вектор то начални стойности, A и A^{EQ} са матрици, както е описано по-горе за неравенство и равенство на граници, съответно b и b^{EQ} са сърфиращи вектори, както е описано по-горе. LB и UB са вектори със същия размер като p_0 дето съдържа по-горна и по-долна граници съответно.

Забележка*: LB и UB* могат винаги да бъдат представени с *A* и *b*. Например, да предположим, че границата беше $-1 \leq p \geq 1$, тогава *A* и *b* биха били:

$$A = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

който Матлаб изрази за $-p \leq 1$ (която е еквивалентна на $p \geq -1$), и $p \leq 1$. *nclon* е нелинеарена граница на функция, която връща стойността на $C(x) - d$ и $C^{NEQ}(x) - d^{NEQ}$ (това е трикова функция. Виж *doc fmincon* за специалисти). *options* е ползвател,представляващ оптимизационни опционни структури, и $var_1, var_2, \ldots$са опционни променливи за употреба на данни.

Да предположим, че Вие искате да оптимизирате CRS Cobb-Douglas утилитарна (полезна, изгодна) функция на формата $U(x_1, x_2) = x_1^{\lambda} x_2^{1-\lambda}$ предмет към бюджетна граница $p_1x_1 + p_2x_2 \leq 1$. Това е нелинеарен функционен предмет t_0 към линеарна граница (отбележете, не ние се нуждаем $x_1 \geq 0$ $x_1 \geq 0$. Първо вие трябва да определите оптимизационната цел.

```
function u = crs_cobb_douglas(x,lambda)

x1=x(1);
x2=x(2);

u=x1^(lambda)*x2^(1-lambda);
u=-u %Must change max problem to min!!!
```

Оптимизационния проблем може да бъде формулиран в Матлаб

```
>> options = optimset('fmincon');
>> options = optimset(options,'Display','iter');
>> prices  = [1 1]; %You can change this set of parameters
```

```
>> lambda  = 1/3;  %You can change this parameter
>> A = [-1 0; 0 -1; prices(1) prices(2)]
A =
   -1     0
    0    -1
    1     1
>> b=[0; 0; 1]
b =
    0
    0
    1
>> p0=[.4; .4]; %Must start form a feasible portion, usually off the constraint
>> [x,fval,exitflag]=fmincon('crs_cobb_douglas',x0,A,b,[],[],[],[],[],options,lambda)
                                max                    Directional   First-order
 Iter F-count        f(x)   constraint    Step-size   derivative   optimality Procedure
    0      3         -0.4         -0.2
    1      6    -0.529134            0            1       -0.106        0.129
    2      9    -0.529134            0            1   -4.14e-025    2.01e-009
Optimization terminated: first-order optimality measure less
 than options.TolFun and maximum constraint violation is less
 than options.TolCon.
Active inequalities (to within options.TolCon = 1e-006):
  lower      upper     ineqlin   ineqnonlin
                         3
x =
    0.3333
    0.6667
fval =
   -0.5291
exitflag =
    1
```

exitflag стойност от 1-ца показва успех. Да предположим, че вие не харесвате полезната максимална формула на този проблем, но предпочитате двойнствена стойностна минимизация. При тази алтернативна формула, оптимизационните проблеми стават:

$$\min_{x_1,x_2} p_1x_1 + p_2x_2 \text{ subject to } U(x_1,x_2) \geq \bar{U}$$

Отново за да разрешим този проблем в Матлаб ние трябва първо да определим обективната функция

```
function cost = budget_line(x,prices,lambda,Ubar)

x1=x(1);
x2=x(2);

p1=prices(1);
p2=prices(2);

cost = p1*x1+p2*x2;
```

и понеже този проблем има нелинеарена единица, ние трябва да определим *nclon* функция,

```
function [C, Ceq] = compensated_utility(x,prices,lambda,Ubar)
```

```
x1=x(1);
x2=x(2);

 u=x1^(lambda)*x2^(1-lambda);

 con=u-uBar; %Note this is a >= constraint
 C=-con; %This turns it into a <= constraint
 Ceq = []; %No equality constraints
```

Забележка: Граничната функция и оптимизацията, *трябва* да вземат същите допълнителни аргументи в същия ред, дори ако те не нуждаят от тях. *target* (цел) може да бъде решен в Матлаб използвайки

```
>> options = optimset('fmincon');
>> options = optimset(options,'Display','iter');
>> prices  = [1 1]; %You can change this set of parameters
>> lambda  = 1/3;  %You can change this parameter
>> A = [-1 0; 0 -1] %Note, we still need x1 and x2>=0
A =
   -1     0
    0    -1
>> b=[0; 0]
b =
    0
    0
>> Ubar = 1;
>> x0 = [1.5;1.5]; %Start with all constraints satisfied, since -1.5+1<0 (-u+ubar).
>> [x,fval,exitflag]=fmincon('budget_line',x0,A,b,[],[],[],[],'compensated_utility',...
                    options,prices,lambda,Ubar)
                                    max                    Directional   First-order
 Iter F-count        f(x)   constraint    Step-size   derivative   optimality Procedure
    0      3           3         -0.5
    1      6         1.9      0.01078            1         -1.1        0.629
    2      9     1.86428      0.01358            1      -0.0357       0.0871
    3     12     1.88984    2.182e-005            1       0.0256      0.00186
    4     15     1.88988    4.278e-007            1    3.95e-005     0.000151  Hessian modified
    5     18     1.88988    3.386e-009            1    7.96e-007    1.06e-007  Hessian modified
Optimization terminated: first-order optimality measure less
 than options.TolFun and maximum constraint violation is less
 than options.TolCon.
Active inequalities (to within options.TolCon = 1e-006):
  lower      upper     ineqlin   ineqnonlin
                                     1

x =
    0.6300
    1.2599
fval =
    1.8899
exitflag =
     1
```

Тези два примера са относително прости проблеми, където ние можем аналитично да проверим отговорите. За нещастие за много проблеми ние не можем да бъдем сигурни, че Матлаб е създала глобален оптимум (например, ако има локален оптимум). В тези случай стандартната практика е да пробваме много различни начални стойности и да използваме най-ниската *fval*. Ако нещата работят добре, много от началните стойности, би трябвало да създадат оценки на параметъра близки до другите с подобни **fvals**.

Забележка: Много аспекти от гранична оптимизация (и оптимизация въобще) са повече черна магия, отколкото наука. По-лошо, те са проблемен клас, специфичен с толкова общи правила, че е трудно да отделим. Единият начин да станем специалисти при минимизацията на функция е просто, като го правим.

16.5 optimset

optimset включва оптимизационни опции и има две отделни форми. Началната всяка **optimset**, би трябвало винаги да бъде по форма **options = optimset (‘fmin*type’*)**, която ще върне грешните опции за този тип. Щом вие имате начална оптимизационни структури, индивидуалните опции могат да бъдат променени чрез извикване на

```
options = optimset(options,'option1',option value1,'option2',option value2,...)
```

Например, да поставим опции за **fmincon**

```
>> options = optimset('fmincon');
>> options = optimset(options,'MaxFunEvals',1000,'MaxIter',1000);
>> options = optimset(options,'TolFun',1e-3);
```

За **help** върху намиране опции или тяхното специфично значение виж ***doc optimset.***

16.6 Други оптимизационни действия

Оптимизационните инструменти на Матлаб, съдържат известен брой от други оптимизационни алгоритми:

fseminf - многоизмерна ограничена минимизация, полуопределени граници;

fgoalattain - многоизмерна оптимизация за целеви знания;

fminimax - многоизмерна минимаксимилизационна оптимизация;

lsqlin - линеарни най-малки квадрати с линеарни граници;

lsqnonneg - линеарни най-малки квадрати с неотрицателни граници;

lsqcurvefit - нелинеарна крива минаваща през най-малките квадрати (с граници);

lsqnonlin - нелинеарна крива минаваща през най-малките квадрати с по-горни и по-долни граници;

bintprog - бинарно линеарно числово програмиране;

linprog - линеарно програмиране;

quadprog - квадратично програмиране.

17. Dates (дати)

Запазването на пътека от дати е особено важно, когато работим с данни времеви серии. Матлаб съхранява дати като дни от *1 Януари 0000*, известно като серия дати на Матлаб. Например: *1 Януари 0000* е единица във формата дата в Матлаб. Докато 1 Януари 2000 е *730,486*. Матлаб серии-дати солидират часове, като отделни дни, така че *12:00 1 Януари 2000* е *730,786.5*. Стандартния метод да вкараме дати в Матлаб е да използваме *Excel* за създаване на *Excel* дати и да добавим константа за да картираме *Excel* датите (които са базирани на *1 Януари 1900*) към Матлаб дати. Обаче това не винаги е възможно и Матлаб осигурява известен брой полезни функции за да манипулираме дата – данни.

17.1 datenum

datenum превръща редица дати (*01'JAN200')* или цифри *дати ([2000 01 01])* в Матлабови серия-дати. За да извикаме функцията с редица от дати, използваме **datenum(string_date)** или **datenum(string_date, *format*),** където *format* е съставено от блокове от:

yyyy	четири дигитална година
yy	двойна дигитална година, рисковано понеже може да включи погрешен век
mmmm	пълното име на месеца (например, January)
mmm	първите три букви на месеца (например, JAN)
mm	цифров месец от годината
m	капитализирана първа буква от месеца
dddd	пълното име от седмицата
ddd	първите три букви на деня от седмицата
dd	цифров ден от месеца
d	капитализирана първа буква на деня от седмицата
HH	час, би трябвало да бъде 24-часов формат и се поставя 0 пред цифрата, ако една цифра
MM	минути, трябва да бъде добавяно допълнително 0, ако е една цифра
SS	секунди, трябва да бъде добавяно 0, ако е една цифра

Матлаб автоматично ще разпознае по-важните формати от редица дати. Обаче форматът редици по-горе може да бъде използван при обработка на нестандартни случаи. Те са особено важни, ако аргументите се появяват в странен ред, така всеки като *yyyyddmm* (

например 20000101), или ако датите са неограничени при използване на нестандартни образи, такива като *;* или *,* (например 2000; 01;01).

Няколко примера:

```
>> datenum('01JAN2000')
ans =
      730486
>> datenum('01JAN2000','ddmmmyyyy')
ans =
      730486
>> datenum('01;JAN;2000','dd;mmm;yyyy')
ans =
      730486
>> datenum('01012000','ddmmyyyy')
ans =
      730486
```

datenum също работи върху редици стрелки. Например:

```
>> strdates=strvcat('01JAN2000','02JAN2000','03JAN2000')
strdates =
01JAN2000 02JAN2000 03JAN2000
>> datenum(strdates)
ans =
       730486
       730487
       730488
```

datenum може също да бъде използван за превръщане на цифрови дати, такива като [2000 01 01], към Матлаб формат серия на дата. Например:

```
>> datenum([2000 01 01])
ans =
     730486
>> years=[2000;2000;2000];
>> months=[01;01;01];
>> days=[01;02;03];
>> [years months days]
ans =
        2000           1           1
        2000           1           2
        2000           1           3
>> datenum(years,months,days)
ans =
     730486
     730487
     730488
```

datenum може също да бъде използвана за превод на часове, минути или секунди в части от дни, то би трябвало да е лесно да пишеш твой собствен код за да овладееш това.

17.2 datestr

datestr е инверсия (обратно) на **datenum**. То създава човешка четлива редица от Матлабова серия дата. При грешка, то ще върне редица дати на формата *'dd-mmm-yyyy'*. Обаче, ако се знае номера на стандартните формати, такива като *'mm/dd/yy'* или *'mmm.dd,yyyy'*. За да създаде един от нестандартните дати формати, използва **datestr (serial_date, #),** където # съответства на един от формат редици (виж help **datestr**). **datestr** може също да създаде редици с арбритрирани формати използващи синтаксиса детайлизиран по-горе (например *'dd;mm;yyyy'*, за да създаде редица дати; не ограничители). Няколко примера:

```
>> serial_date=datenum('01JAN2000')
serial_date =
      730486
>> datestr(serial_date)
ans =
01-Jan-2000
>> datestr(serial_date,0)
ans =
01-Jan-2000 00:00:00
>> datestr(serial_date,'mmm;dd;yyyy')
ans =
Jan;01;2000
```

Като **datenum**, **datestr** може да вземе векторен вход и да върне векторен изход.

```
>> serial_date=datenum(strvcat('01JAN2000','02JAN2000','03JAN2000'))
serial_date =
     730486
     730487
     730488

 >> datestr(serial_date)
 ans =
 01-Jan-2000
 02-Jan-2000
 03-Jan-2000
```

17.3 datevec

datevec превръща Матлабови серия дати в човешки раз делими цифрови формати. Особеност, дадена Матлабова серия-дата, **datevec** ще създаде *1* x *6* вектор в формата [година месец ден час минута секунда]. Например:

```
>> serial_date=datenum(strvcat('01JAN2000','02JAN2000','03JAN2000'))
serial_date =
      730486
     730487
     730488
>> datevec(serial_date)
ans =
       2000           1           1           0           0           0
       2000           1           2           0           0           0
       2000           1           3           0           0           0
```

което съответства на *12:00 среднощ Януари 1-3, 2000*.

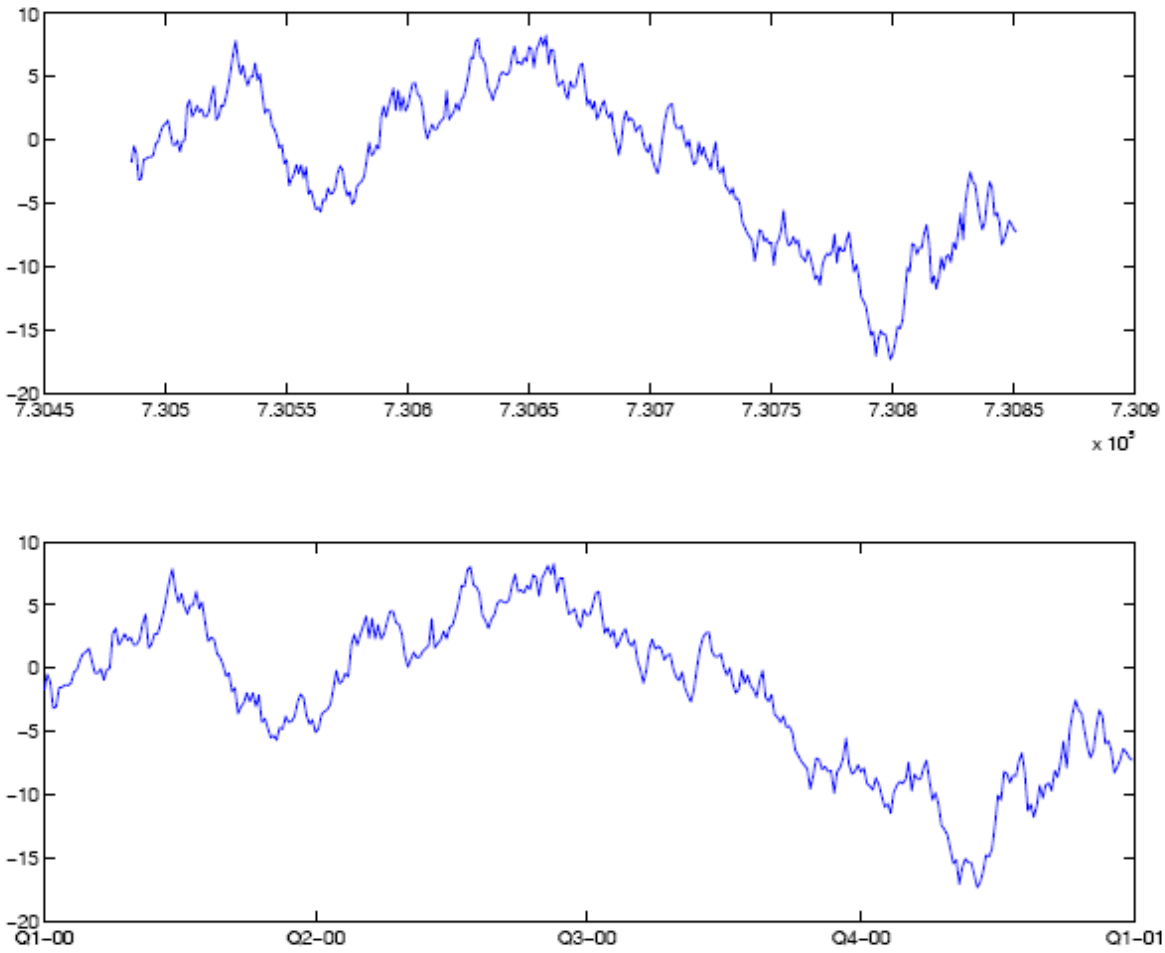
17.4 now and clock

now връща Матлабова серия-дата представена на компютърния часовник. **clock** връща *1* x 6 вектор (същият формата като **datevec**) на компютърния часовник. **datevec (now)** създава същия изход, като **clock**.

17.5 datetick

datetick не е функция за яснота при работа с дати. **datetick** превръща оста на диаграмата изразена в Матлабова серия-дати в текстови дати. Например:

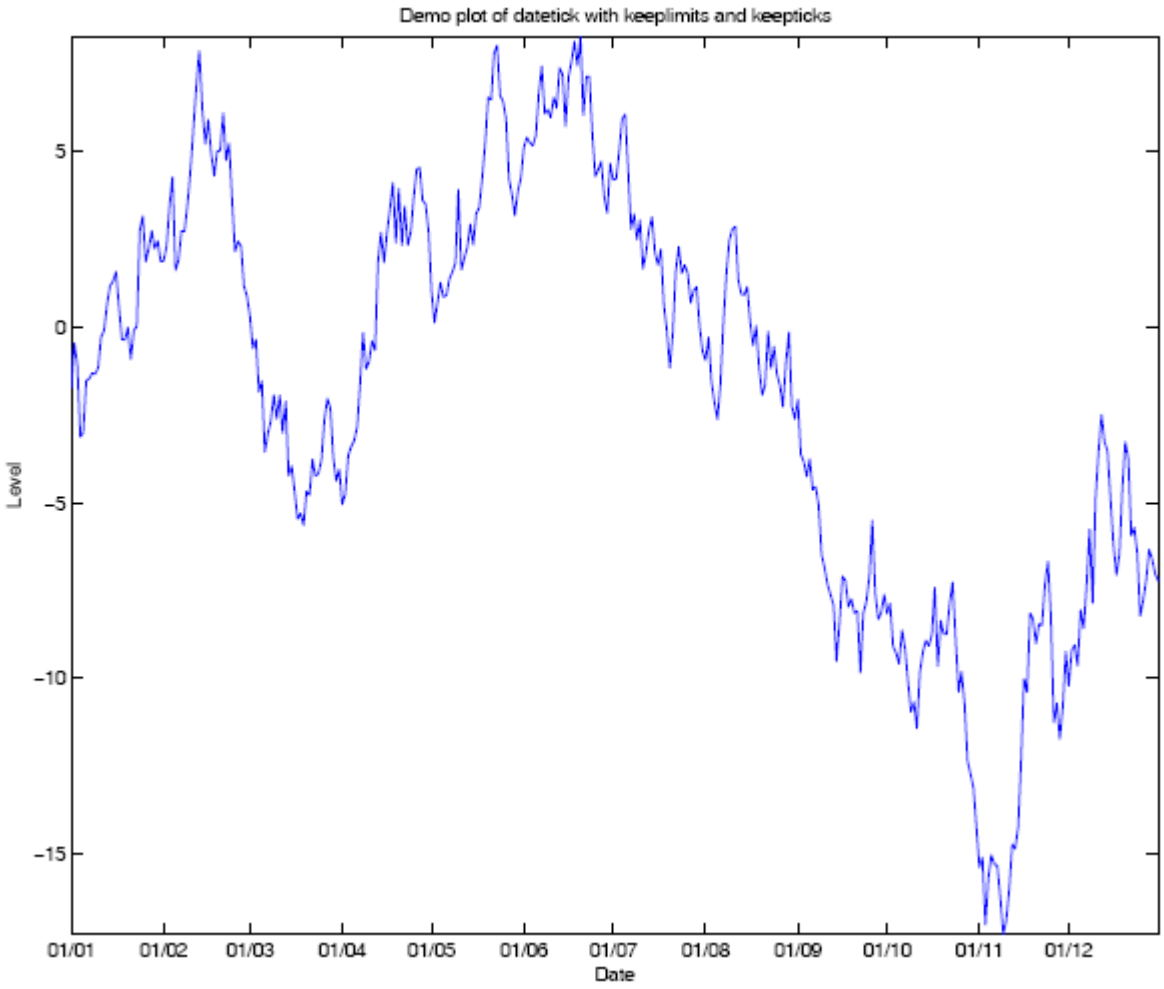
```
>> dates = datenum('01Jan2000'):datenum('31Dec2000');
>> rw    = cumsum(randn(size(dates)));;
>> subplot(2,1,1);
>> plot(dates, rw);
>> subplot(2,2,1);
>> plot(dates, rw);
>> datetick('x')
```



Фигура 17.1: Пример на *datetick*. **datetick** превръща Матлаб серия-дати в текстови редици. За нещастие, то типично променя местоположението на точките и прави, използвайки лоши избори. Решението е да използваме **datetick ('x','keepticks','keeplimits').**

Примера създава две диаграми във *фигура 17.1*. Горната съдържа Матлаб серия-дати по оста *–x*, докато долната съдържа редица дати. **datetick** също разбира две стандартни форматиращи команди (виж **datestr**) и обичайни форматиращи команди (виж **datenum**). Тази функция има нещастната тенденция да създава малко *x*- надписи (етикети). Решението е първо да изберем точките на вашия надпис на оста (в серия дати), и после използваме **datetick ('x','keepticks','keeplimits'),** като илюстрация във *фигура 17.2*.

```
>> h=plot(dates, rw);
>> axis tight
>> serial_dates=datenum(strvcat('01/01/2000','01/02/2000','01/03/2000','01/04/2000',...
                               '01/05/2000','01/06/2000','01/07/2000','01/08/2000',...
                               '01/09/2000','01/10/2000','01/11/2000','01/12/2000'),...
                               'dd/mm/yyyy');
>> parent=get(h,'Parent');
```



Фигура 17.2: datetick с **keepticks** и **keeplimits**. Тези два аргумента осигуряват **datetick** да се държи разумно (нормално). За да ги използваме настройваме Вашата фигура като с вас за да изглежда (но със серия дати) и после извикваме **datetick ('x','keepticks','keeplimits').**

```
>> set(parent,'XTick',serial_dates);
>> datetick('x','dd/mm','keeplimits','keeplimits');
>> xlabel('Date')
>> ylabel('Level')
>> title('Demo plot of datetick with keeplimits and keepticks')
```

18. String Manipulation (редова манипулация)

Докато редовата манипулация не е Матлабов редови подход, той осигурява изключително пълен комплект от инструменти за работа с редици. Редиците в Матлаб са прости матрици с характерни данни. Простите редици могат да бъдат въведени от командната линия

```
str = 'Econometrics is my favorite subject.';
```

Понеже характерните данни са матрици, те се държат по стандартния начин към много команди (str (1:10 например). Обаче използвайки команди създадени за цифрови данни е скучно, а специални целеви редици-функции на Матлаб могат да помагат.

Началната употреба на редови функции в Матлаб е за делени данни. В *раздел 3*, пример за делене на зле форматиран файл беше показан. Той използва известен брой Матлабови функции на редовата манипулация за манипулиране и делене на текст с командата *read-in* от файл.

18.1 char

char променя цифрови стойност от цяло число между 1 и 255 в техния *ASCII* еквивалентни образи. Други стойности създават безсмислени резултати.

```
>> char(65:100)
ans =
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcd
>> char([1.32 pi 256])
ans =
□□□
```

18.2 double

double променя характерните редици в техните цифрови стойности.

```
>> double('Matlab')
ans =
   77    97   116   108    97    98
```

18.3 strvcat

s**trvcat** вертикално свързва две редици. В нормална математическа настройка към матрици *x* и *y* може да бъде свързан вертикално *[x;y]*. Обаче, редиците често имат различна ширина, което прави свързването трудно. **strvcat** го прави лесно.

```
>> strvcat('apple','banana','cherry')
ans =
apple
banana
cherry
>> strvcat('apple','banana','cherry')
ans =
apple banana cherry
>> x=strvcat('alpha','beta');
>> y=strvcat('delta','gamma');
>> strvcat(x,y)
ans =
alpha
beta
delta
gamma
```

18.4 strcat

strcat хоризонтално свързва редиците. **z=strcat (x,y)** е същото като **z=[x y],** когато *x* и *y* имат същото число редици. Ако човек има единична редица **strcat** я свързва към всяка редица от другия вектор.

```
>> strcat(strvcat('a','b'),strvcat('c','d'))
ans =
ac
bd
>> strcat(strvcat('a','b'),'c')
ans =
ac
bc
```

18.5 strfind

strfind връща всички начала, на който и да е блок с текст в друга редица. Той е полезен при намирането на неограничени образи в блок от текст, за да помогне при ръчно деление. Например, а вземем под внимание единична линя от *WRDS TAQ* изход:

```
>> str = 'IBM,02JAN2001,9:30:07,84.5';
>> strfind(str,',')
ans =
     4    14    22
```

strfind връща всички местоположения на *',*'. Ако вие търсите повече от един образ, **strfind** може да създаде частично съвпадащи блокове.

```
>> str = 'abababababa'
str =
abababababa
>> strfind(str,'aba')
ans =
     1     3     5     7
```

18.6 strcmp and strcmpi

strcmp сравнява две редици и връща (логично) една, ако те са еднакви. Тази команда е чувствителна при конкретния случай. **strcmpi** върши същото, но тя не е чувствителна при конкретния случай.

```
>> strcmp('a','a')
ans =
     1
>> strcmp('a','A')
ans =
     0
>> strcmpi('a','A')
ans =
     1
```

18.7 strcmpi and strcnmpi

strcmp сравнява първото *n* образи на две редици и връща (логично) едната, ако те са еднакви. Тя е чувствителна при конкретния случай. **strcmpi** върши същото, но не е чувствителна при конкретния случай.

```
>> strncmp('apple','apple1',5)
ans =
     1
>> strncmp('apple','apple1',6)
ans =
     0
>> strncmp('apple','Apple1',5)
ans =
     0
>> strncmpi('apple','Apple1',5)
ans =
     1
```

18.8 strmatch

strmatch сравнява редици на матричен образ с редица и връща индекс на всички редове, които започват с редицата. За да свържем само определен ред, използваме допълнителна команда **'exact'**.

```
>> str = strvcat('alpha','beta','alphabeta');
>> strmatch('alpha',str)
ans =
     1
     3
>> strmatch('alpha',str,'exact')
ans =
     1
```

18.9 regexp and regexpi

regexp е подобна на **strfind**, но приема стандартен редовен израз на писана команда за да открие сходствата (връзките). Тя е чувствителна към конкретния случай. **regexpi** върши същото, но не е чувствителна към конкретния случай. Например на **regexp**, виж *doc regexp.*

18.10 str2num

str2num превръща стойностите на редицата в цифрови променливи. Вход може да бъде, както векторно, така и матрично стойностен (оценен).

```
>> str2num(strvcat('1','2','3'))
ans =
    1
    2
    3
>> str2num(['1 2 3';'4 5 6'])
ans =
     1     2     3
     4     5     6
```

18.11 num2str

num2str превръща цифровите стойности в редици. Вход може да бъде, както вектор, така и матрично стойностен.

```
>> num2str([1;2;3])
ans =
1 2 3
>> num2str([1 2 3;4 5 6])
ans =
1  2  3
4  5  6
```

19. File System and Navigation (файлова система и навигация)

Матлаб използва стандартната *DOS* файлова система и командите да променя работещите директории. Например, за да променим директорията, пишем

```
cd c:\MyDirectory
```

Други стандартни *DOS* файлови навигационни команди, такава като **dir** и **mkdir** са също в наличност от Матлаб.

Алтернативно текущата директория може да бъде променен чрез кликване на бутона с ….следваща до текуща директория *box*, най-отгоре на командния прозорец (виж *фигура 1.1*).

19.1 The Matlab path (пътеката на Матлаб)

Докато тази секция звучи като будистки ритуал за откъса, пътеката е всъщност много важна променлива в света на Матлаб. Пътеката казва на Матлаб къде да погледне за файлове, които вие възможно искате да извикате. Всичките директории от *toolbox* са автоматично качени върху твоята пътека, но вие можете да се нуждаете да добавите нови директории, ако вие имате свой собствени функции или желаете да използвате обичайния инструментариум.

За да видим текущата пътека въведи командата ***path*** в командния прозорец. Алтернатива има GUI пътека *browser* в наличност под **File>Set Path…** Пътеката е сортирана от най-важната директория до най-важната, с настояща работещата директория (като pwd връща в командния прозорец тихичко) най-отгоре на списъка. Пътеката контролира кои файлове на Матлаб ще се използвате, като въведете функция или името на *batch* файла.

Да предположим, че случайно създадете Ваш собствен файл **mean**. Ако вие въведете **mean** в командния прозорец, Матлаб ще намери всичко свързано с него върху пътеката и ще ги подреди на основния ред на пътеката. После ще изпълнява най-високия от подредените. Естествено вие не би трябвало да използвате същите имена на функция. Обаче грешки наистина се случват и ако вие се разтревожите, че извиквате погрешна функция, коя *function –all* , ще ви покаже всичките файлове с това име (функция*, m*- файлове и *mat* файлове), връщащо ги в реда, по който те се появяват върху пътеката.

Използвайки *File>Set Path…* или *addpath*, нови директори могат да бъдат придадени към пътеката. GUI инструмента осигурява допълнителна функционалност, понеже той може да бъде използван за пренареждане на директориите върху пътеката. За да спестим някои промени, дето Вие сте направили, използвайте командата *savepath* или кликнете върху *Save*

Path *GUI.*

Раздел 20
Quick Function Reference (бързи препоръчителни функции)

Това е кратко обобщение на функция, която вие можете да намерите за полезна. Тя само щрихира повърхността на онова дето Матлаб трябва да предложи. Има около 100 функции изброени тук; Матлаб и статистически инструментариум се обединяват за да създадат поне 1400.

20.1 General Math (обща математика)

20.1.1. abs

Връща абсолютната стойност на елементите на вектор или матрица. Ако се използва върху комплексни данни, връща комплексни модули.

20.1.2. diff

Връща разликата между два съседни елемента на вектор. Ако оригиналния вектор има дължина *T*, върнатия вектор има дължина *T-1*. Ако се използва върху матрица, връща матрица от различия на всяка колона. Върнатата матрица има един ред по-малко, отколкото оригиналната.

20.1.3. exp

Връща показателна функция (e^x) на елементите на вектор или матрица.

20.1.4. log

Връща естествения логаритъм на елементите на вектор или матрица. Връща комплексните стойности за отрицателните елементи на вектор или матрица.

20.1.5. log10

Връща логичната база 10 на елементите от вектор или матрица. Връща комплексните стойности за отрицателните елементи за вектор или матрица.

20.1.6. max

Връща максимума на вектор. Ако се използва върху матрица, връща максимума на всяка колона.

20.1.7. mean

Връща аритметичното значение на вектор. Ако се използва върху матрица, връща аритметичното значение на всяка колона.

20.1.8. min

Връща минимума на вектор. Ако се използва върху матрица, връща минимума на всяка колона.

20.1.9. mod

Връща остатъците от деление на елементите на вектор или матрица.

20.1.10. roots (корени)

Връща корените полиноминал.

20.1.11. sign (знак, подпис)

Връща знака на елементите на вектор или матрица. Знака е определен (дефиниран) на $x / |x|$ и 0 ако $x = 0$.

20.1.12. sum

Връща сумата на елементите на вектор. Ако се използва върху матрица създава сумата на всяка колона.

20.2 Rounding (закръгление)

20.2.1. ceil

Връща следващото по-голямо число. Изходът е същия размер като входа и *ceil* работи елемент по елемент.

20.2.2. floor (под)

Връща следващото по-малко число. Изходът е същия размер, като входа и *floor* оперира елемент по елемент.

20.2.3. round (кръг, объл)

Кръговете към най-близкото число. Изходът е същия размер, като входа и *round* оперира елемент по елемент.

20.3 Statistics (статистика)

20.3.1. corrcoef

Изчислява корелацията (съотношението) на матрицата. Ако *x* е матрица, ако *N* по *M,* връща *M* по *M* корелация, третираща колоните на *X*, като случайни променливи.

20.3.2. cov

Изчислява *ко*-варианта на матрица. Ако матрично *X*, ако *N* по *M*, връща *M* по *M*, *ко*-вариант третиращ колоните на *X* като отделни променливи. Ако се използва върху вектор, създава същия изход като **var**.

20.3.3. kurtosis

Изчислява **kurtosis** на вектор. Ако се използва върху матрица, **kurtosis** на всяка колона се връща.

20.3.4. median

Връща **median** на вектор. Ако се използва върху матрица, връща **median** на всяка колона.

20.3.5. quantile

Изчислява **quantile** на вектор. Ако се използва върху матрица, връща **quantile** на всяка колона на матрицата.

20.3.6. skewness

Изчислява **skewness** на вектора. Ако се използва върху матрица, **skewness** на всяка колона са връща.

20.3.7. std

Изчислява стандартните отклонения на вектор. Ако се използва върху матрица, връща стандартните отклонения на всяка колона.

20.3.8. var

Изчислява варианта на вектор. Ако се използва върху матрица, варианта на всяка колона се връща.

20.3.9. *DIST* pdf

Връща вероятностна гъстота на функционалните стойности за даден *DIST*, където *DIST* взема една от многото форми, такива като **t (tpdf), norm (normpdf)** или **gam (gampdf). T**очните входове варират по разпределение.

20.3.10. *DIST* cdf

Връща съвкупността на разпределение на функционалните стойности на функции за дадено *DIST*, където *DIST* взема едно от многото форми, такива като **t (tcdf), norm (normcdf)** или **gam (gamcdf).** Точните входове варират по разпределение.

20.3.11. *DIST* inv

Връща инвенсирана съвкупност от стойностите на разпределение за дадено *DIST*, където *DIST* взима от многото форми, такива като **t (tinv), norm (norminv)** или **gam (gaminv).** Точните входове варират по разпределение.

20.3.12. *DIST* rnd

Създава псевдо случайни числа за дадено *DIST*, където *DIST* взима едно от многото форми, такива като **t (trnd), norm (normrnd)** или **gam (gamrnd).** Точните входове варират по разпределение.

Забележка: Повечето *DIST* функции са в наличност за следните разпределения: Beta, Binominal, Exponential, Value, F, Gamma, Generalized Extreme Value, Generalized Pareto, Geometric, Hyper geometric, Lognormal, Negative Binominal, Noncentral F, Noncentral t, Noncentral Chi-square, normal, Poisson, Rayleigt, T, Uniform, Discrete, Uniform ,Weibull.

20.4 Random Numbers (случайни числа)

20.4.1. rand

Обобщен генератор за псевдо случайни числа.

20.4.2. randn

Стандартен генератор за нормални псевдо случайни числа.

20.4.3. random

Общ генератор за псевдо случайни числа. Може да генерира случайни числа за следните разпределения: Beta, Binominal, Exponential, Value, F, Gamma, Generalized Extreme Value, Generalized Pareto, Geometric, Hyper geometric, Lognormal, Negative Binominal, Noncentral F, Noncentral t, Noncentral Chi-square, normal, Poisson, Rayleigt, T, Uniform, Discrete, Uniform ,Weibull.

20.5 Logical (логичен)

20.5.1. all

Връща логична истина (1), ако всички елементи на вектора са логично верни. Ако се използва върху матрица връща логично вярно, ако всеки елемент на всяка колона са логично верни.

20.5.2. any

Връща логично вярно (1), ако някои елементи на вектор са логично верни. Ако се използва върху матрица, връща логично вярно, ако някои елементи на всяка колона са логично верни.

20.5.3. find

Връща индекси на елементи на вектор или матрица, които удовлетворяват логичното условие.

20.6 Special Values (специални стойности)

20.6.1. ans

ans е специална променлива, която създава стойността на последната не одобрена операция.

20.6.2. eps

eps е цифрово прецизиране на Матлаб. Числата различни по повече eps са същите в Матлаб.

20.6.3. Inf

Inf представлява безкрайност в Матлаб.

20.6.4. NaN

NaN представлява не число в Матлаб. То се появява като резултат от представяне на операция, която създава неопределен резултат, такъв като *Inf/Inf*.

20.7 Special Matrices (специални матрици)

20.7.1. eye (око)

z=eye (N) връща *N* по *N* идентична матрица.

20.7.2. linspace

z= linspace (L, U, N) връща *1* по *N* вектор от точки еднакво разпределени между *L* и *U* .

20.7.3. logspace (логаритмично пространство)

z= logspace (L, U, N) връща *1* по *N* вектор от точки логаритмично разпределени между 10^L и 10^U .

20.7.4. ones (единици)

z= ones (N, M) връща *N* по *M* матрица от единици.

20.7.5. zeros (нули)

z=zeros (N. M) връща *N* по *M* матрица от нули.

20.7.6. toeplitz

z=toeplitz (x) връща *Toeplitz* матрица конструирана от вектор *x*. Например:

```
>> z = toeplitz([3 2 1]);

>> z

z =

 3 2 1
 2 3 2
 1 2 3
```

20.8 Matrix Functions (матрични функции)

20.8.1. chol

Изчислява Чолески фактор от позитивно определена матрица.

20.8.2. det

Изчислява детерминанта на квадратна матрица.

20.8.3. diag

Връща елементи по диагонала на квадратна матрица. Ако входа към диагонала е вектор, връща матрица с този диагонал.

20.8.4. eig

Връща *eig* –стойности и *eig*-вектори на квадратна матрица.

20.8.5. inv

Връща инверсия на квадратна матрица.

20.8.6. kron

Кроникер продукт на две матрици.

20.8.7. trace

Връща следата на матрицата *(sum (diag(x)))*.

20.8.8. tril

Връща по долната триъгълна версия на входна матрица.

20.8.9. triu

Връща по горната триъгълна версия на входна матрица.

20.9 Matrix Manipulations (матрични)

20.9.1. cat

Свързва две матрици по някакъв размер. Ако *x* и *y* са подобни (съответстващи), *cat(1, x, y)* е същото като *[x; y]* и *cat(2, x, y)* е същото като *[x; y]*.

20.9.2. length (дължина)

Дължината на най-дългия размер на матрица. В 2D случай е еквивалентна на *max(size(x, 1), size(x, 2)).*

20.9.3. numel

Връща броят от елементите в матрицата. Ако матрицата е 2D с размери *N* и *M*, *numel=NM.*

20.9.4. repmat

Репликира на матрица според размерите, които са зададени.

20.9.5. reshape (преоформям)

Преоформя матрица за да има различен размер. Продуктът от размерите трябва да бъде същият преди и след (броят елементи не може да се променя).

20.9.6. size (размер)

Връща размера на матрицата. Размер 1 е броя на редиците, размер 2 е броя на колоните.

20.10 Set Function (функции за настройка)

20.10.1. intersect

Връща междинна секция на два вектора. Може да бъде използвана с допълнителен аргумент ‘rows’ и еднакви по размер матрици за да се създаде междинен сектор от редици на две матрици.

20.10.2. setdiff

Връща разликата между елементите на два вектора. Може да бъде използвана с допълнителен аргумент ‘rows’ и еднакви по размер матрици за да се създаде матрица съдържаща разликата от редиците на двете матрици.

20.10.3. union (съюз, обединение)

Връща обединението на два вектора. Може да бъде използвана с допълнителен аргумент ‘rows’ и еднакви по размер матрици, за да се създаде обединение на редиците на две матрици.

20.10.4. unique (уникален)

Връща уникалността на елементите на вектор. Може да бъде използвана с допълнителен аргумент ‘rows’ и матрица, за да се създаде еднакви редици на матрица.

20.10.5. sort (подреждам)

Създава подреден вектор от най-малкия до най-високия. Ако се използва върху матрица оперира върху всяка от колоните.

20.10.6. sortrows

Сортира редиците на матрица, използва лексико-графично командване, подобно на думи подредени по азбучен ред.

20.11 Flow Control (контрол на потока)

20.11.1. case (случай, дело)

Команда, която може да бъде сравнена към логически вярна или не вярна в, *switch … case … otherwise* блок по контрол на потока.

20.11.2. else (други)

Команда, която е неизпълнението в *if … elseif … else*, блокове на контрол на потока. Ако никое от *if* или *elseif* твърдение не са оценени към логически верни, следва се пътеката *else* .

20.11.3. elseif

Команда, която се използва за да продължим *if … elseif … else* блок на контрол на потока. Би трябвало да бъде незабавно последван от твърдение дето може да бъде оценено като логически вярно или не.

20.11.4. end

Команда показваща края на блок на контрол на потока. Както *if … elseif … else*, така и *switch … case … otherwise* трябва да бъде унищожено с край. Също приключваме и приликите.

20.11.5. if

Команда, която се използва за да започнем *if … elseif … else* блок на контрол на потока. Би трябвало незабавно да бъде последвана от твърдение дето може да бъде оценено като логическо вярно или не.

20.11.6. switch

Команда сигнализираща началото на *switch … case … otherwise* блок на контрол на потока. **Switch** трябва да бъде последвано от променлива съдържаща се от **case**.

20.12 Looping (цикли)

20.12.1. continue (продължавам)

Тази команда извежда текущата примка и продължава програмата по следващата (извън примката) линия.

20.12.2. end (край)

Всички блокове с примки в Матлаб трябва да бъдат унищожени с командата **end**. Също приключва блокове за контрол на потока.

20.12.3. for (за, заради)

Един от двата типа примки в наличност в Матлаб. За примки, примка над предефиниран вектор, докато не приключи с командата **continue.**

20.12.4. while (докато)

Един от двата типа примки в наличност в Матлаб. Докато примките продължават, докато някое логическо условие е оценено като логически невярно (0), докато преждевременно не свърши чрез командата **continue.**

20.13 Optimization (оптимизация)

20.13.1. fmincon

Принудително намаляване на функцията, използвайки търсене базирано на градиент (наклон). При нужда може да бъде линеарно или не-линеарно; равно или неравно.

20.13.2. fminbnd

Минимизиране на функция с ограничения. Намиране min на функция, която съществува между L и U.

20.13.3. fminsearch

Минимизиране на функция използвайки simplex (без дериват (производни)) търсене.

20.13.4. fminunc

Намиране на функция използвайки търсене базирано на градиент.

20.13.5. optimget

Получава структурни опции за оптимизация.

20.13.6. optimset

Подрежда структурните опции за оптимизация.

20.14 Graphics (графики)

20.14.1. axis

Подрежда текущите осеви граници на активна фигура. Може също да бъде използвано за свързване на граници използвайки командата **axis tight.**

20.14.2. bar

Създава **bar** диаграма на вектори или матрица с данни.

20.14.3. bar3

Създава 3D **bar** диаграма на вектор или матрица с данни.

20.14.4. contour

Създава контурна диаграма на нова *z* данни срещу вектори с *x* и *y* данни.

20.14.5. errorbar

Създава диаграма с x данни срещу *y* данни със грешни бари (конфиденциални подреждания) около вяска точка.

20.14.6. figure

Отваря нова фигура прозорец, когато се използва с число. Например **figure1** , отваря прозорец с надпис *figure1*. Ако прозорец с надпис *figure1* е вече отворен, поставя тази фигура като активна фигура.

20.14.7. hist

Създава хистограма от данни. Може също да се използва за изчисляване на bin центрове и височина.

20.14.8. legend (легенда)

Създава легенда от елементи на диаграмата.

20.14.9. mesh

Създава 3D **mesh** диаграма на матрица с *z* данни срещу вектори с *x* и *y* данни.

20.14.10. plot

Диаграми с *x* данни срещу *y* данни.

20.14.11. plot3

Диаграми с *z* данни срещу *x* и *y* данни в 3D подреждане.

20.14.12. scatter

Създава **scatter** диаграма с *x* данни срещу *y* данни.

20.14.13. subplot (под диаграма)

Тази команда позволява за умножаване диаграми да бъдат изписани върху същата фигура. Използвана заедно с други диаграмни команди, такива като **subplot (2.1.1); plot(x.y); subplot (2.1.2); plot(y.x)**.

20.14.14. surf

Създава 3D повърхностни диаграми на матрица с *z* данни срещу вектори с *x* и *y* данни.

20.14.15. title (заглавие)

Създава заглавие на текст от горе на фигурата.

20.14.16. xlabel

Създава надпис на текст по *x*-оста на фигурата.

20.14.17. ylabel

Създава надпис на текст по *y* – оста на фигурата.

20.14.18. zlabel

Създава надпис на текст по *z* – оста на фигурата.

20.15 Date Functions (функции за дата)

20.15.1. date (дата)

Връща редица с текущата дата.

20.15.2. datenum

Променя редицата дати, такава като 1 Януари 1900, в Матлаб серийни дати (числови).

20.15.3. datestr

Променя серийни дати към редица дати.

20.15.4. datetick

Променя етикетите на оста в серийни дати в редица надписи в диаграми.

20.15.5. datevec

Прави разбор на числата на датата и на редицата дати и връща дата вектори по формата **[YEAR MONTH DATE HOUR MIN SEC].**

20.16 File System (файлова система)

20.16.1. cd

Променя директорията. Когато се използва с директория променя работещата директория в онази директория. Когато повикаш **cd …,** променя работната директория в нейния родител. Ако желаната директория има пространство, вие би трябвало да използвате функционалната версия **cd('c:\dir with space\dir2\dir3').**

20.16.2. delete

Изтрива файл от понастоящем работещата директория. **Предупреждение:** Тази команда е опасна; който и да е изтрит файл си отива за винаги (не е в кошчето за боклук).

20.16.3. dir

Връща съдържанието на текущата работна директория.

20.16.4. mkdir

Създава нова дете директория в настоящата работна директория.

20.16.5. pwd

Връща пътеката на настоящата работна директория.

20.16.6. rmdir

Премества поддиректорията в сегашна работеща директория. Поддиректорията трябва да бъде празна.

20.17 Matlab Specific(спецификата на Матлаб)

20.17.1. clc

Изчиства командния прозорец.

20.17.2. clear

Изчиства променливите от паметта. Командите **clear** и **clear all**, премахват всички променливи от паметта, докато **clear var1 var2 …** премахва само онези променливи, които са изброени в списъка.

20.17.3. clf

Изчиства съдържанието на фигурния прозорец.

20.17.4. close

Затваря фигурни прозорци. Може да бъде използвана за затваряне на всички фигурни прозорци, чрез извикване на командата close all.

20.17.5. format

Променя начина, по които номерата са представени в командния прозорец. **format long** показва всички десетични места, докато **format short** показва само до 5 (пет).**format short** е недостатък.

20.17.6. help

Показва на екран в линия help за извикване на функция. Също така може да се използва да направи списък на функциите в *toolbox* (инструмент кутията) или да направи списък на инструмент кутиите **(help).**

20.17.7. helpbrowser

Отваря интегрирана система помощ за Матлаб на последната преглеждаща страница.

20.17.8. helpdesk

Отваря интегрирана система **help** за Матлаб **home** страница.

20.17.9. realmax

Връща най-голямото число, което Матлаб е способно да представи. По-големи числа са **inf.**

20.17.10. realmin

Връща най-малкото положително число, което Матлаб е способен да представи. Числа по близко до нула са нула.

20.17.11. which (кое)

Когато се използва в комбинация с име на функция, връща пълна пътека към функцията. Полезно е ако можете да имате умножени функции със същото име.

20.17.12. whos

Връща списък на всички променливи в паметта, заедно с описание на вида и информация за размера и вида на изискваната памет.

20.18 Input / Output (вход / изход)

20.18.1. csvread

Прочита променливи с *.csv* файлове регистрирани в Матлаб. Очаква всички данни да бъдат числови.

20.18.2. csvwrite

Запазва променливите в Матлаб в *.csv* файл.

20.18.3. load

Генерира (зарежда) до голяма степен съдържанието на файла с данни (*.mat*) в текущата работна среда (workspace). Може също така да се използва и за по-прости текстови документи.

20.18.4. save

Съхранява като цяло променливите с данни на Матлаб във файл (*.mat*). Също така може и да съхрани и определен текстови документ.

20.18.5. xlsfinfo

Връща информация за *.xls* файлове, като търси имената в листата.

20.18.6. xlsread

Прочита променливи в *.xls* файлове в Матлаб. Всички данни би трябвало да бъдат числови, но въпреки това, то съдържа методи, които позволяват и тест да бъде прочетен.

20.18.7. xlswrite

Съхранява променливи от Матлаб в *.xls* файл.

II. Изображения и работа с инструменти в Matlab

1 .Фигури, изображения и графики

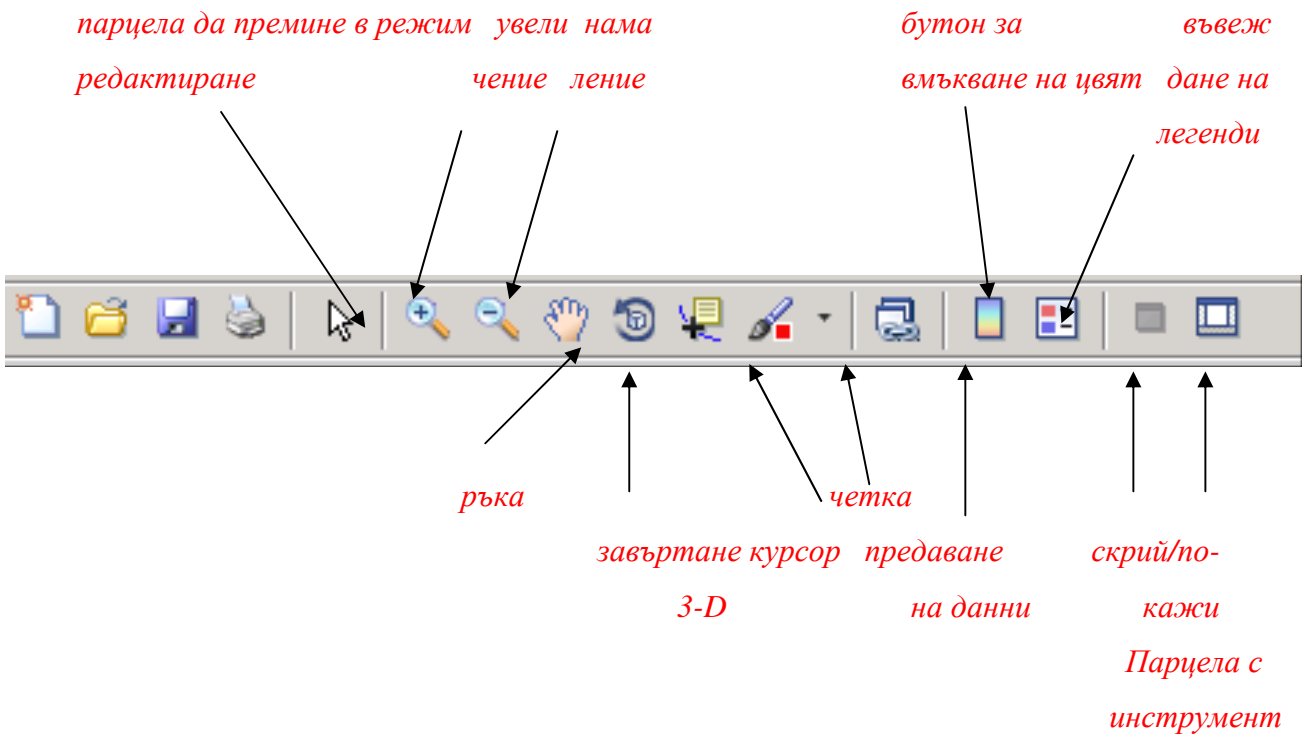
MATLAB предлага разнообразие от данни и функции, които служат за създаване и редактиране на графичен дисплей. Прозорецът MATLAB, съдържа графични дисплеи (обикновено данни с изображения) и UI компоненти. Прозорците са с променливи размери и включват падащи менюта и ленти с инструменти. Изображенията се намират в графичен дисплей, който може да създава рамките на фигури в прозореца. Изображенията могат да се показват в таблични данни, геометрични обекти, както и пояснителни бележки, като например заглавия, легенди, и цветни скали. Фигурите могат да съдържат произволен брой изображения. Всяко изображение може да бъде създадено в рамките на 2-D или 3-D. Те могат да се създадат с осите или с функции.

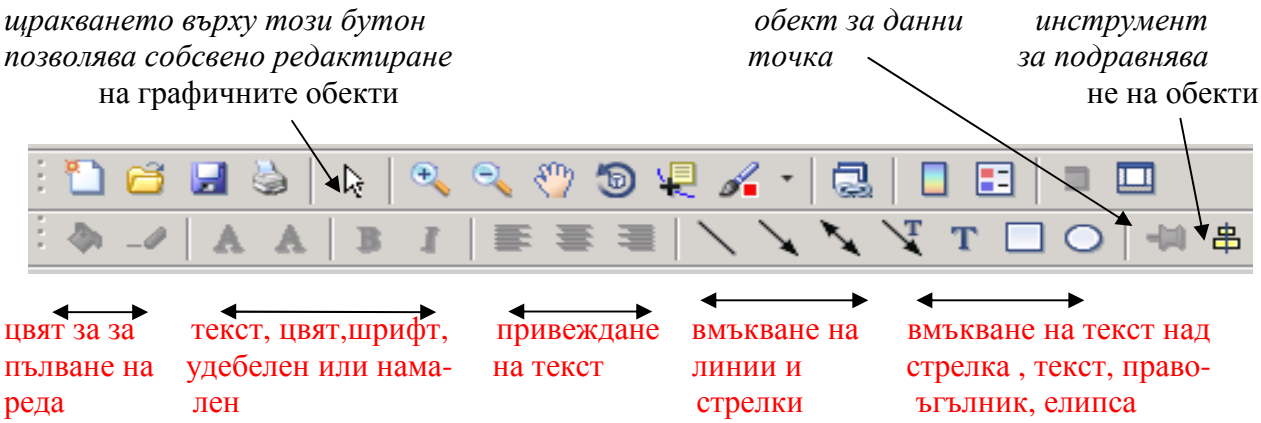
Съдържанието и разновидности на данни, парцели и графики, могат да бъдат обяснени в следващите раздели на MATLAB.

2. Лента с инструменти

Лентата с инструменти дава бърз достъп до често използвани функции. Това включва дейности като: печат, инструменти за интерактивно мащабиране, преместване, завъртане, заявки и редактиране на изображения.

Картинката показва компоненти, предлагани от тази лента с инструменти.





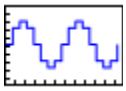
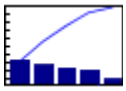
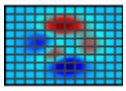
3. Видове MATLAB изображения

MATLAB може да изгражда широка гама от 2-D и 3-D изображения. Следващите две таблици показват вида на изображенията, които могат да се създават. Те включват линия, бар, зона, посока и областен вектор, радиални и графични разсейвания.

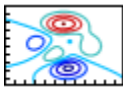
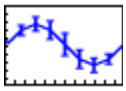
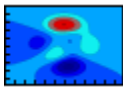
- **Двуизмерно построяване на функции**

Графични линии	Bar графики	Областни графики	Посоки графики	Радиални графики	Точкова графика
plot	Bar	area	feather	polar	scatter
plotyy	Bar(grouped)	pie	guiver	rose	spy
loglog	Bar (stacked)	fill	comet	compass	plotmatrix
Semilogx	barh(подредени)	conturf		ezpolar	
Semilogy	hist	image			

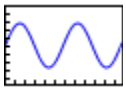
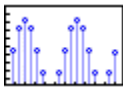
Stairs pareto pcolor

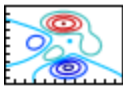
Контури errorbar ezcontourf

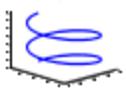
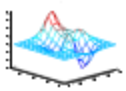
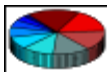
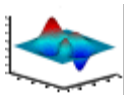
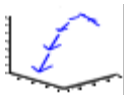
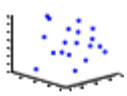
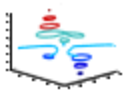
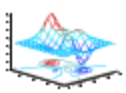
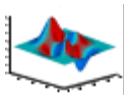
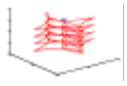
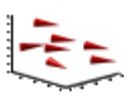
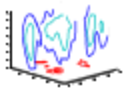
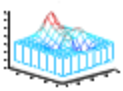
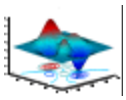
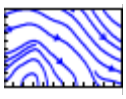
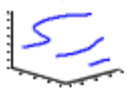
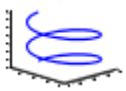
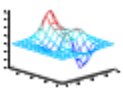
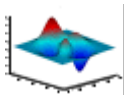
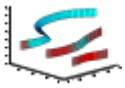
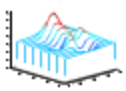
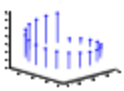
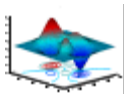
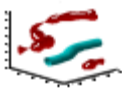
Ezplot stem

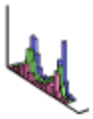
Ezcontour



- **Триизмерно построяване на функции**

Линия Графики	Отворени графики и бар грарфики	Областни графики и конструктивни обекти	Повърхностни графики	Графики за посока	Обемни графики
plot 3	mesh	pie 3	surf	guiver3	scatter3
					
contour 3	meshc	fill 3	surfl	comet 3	coneplot
					
conturs	creeshz	pat	surfc	streamslice	streamline
					
ezplot3	ezmesh	cylinder	ezsurf	streamribbon	
					
waterfall	stem3	ellipsoid	ezsurfc	streamtube	
					

bar3



sphere

Bar3h

Как да изберем изображение в MATLAB ?

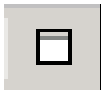
Голяма част от функциите, посочени по – горе са достъпни чрез менюто Figure Palette, това може да се види в падащото меню View. Когато натиснем върху Figure Palette, и то е активно може да се избере едно, две или повече изображения. Избираме това, което ни е необходимо.

4. Работа с инструменти в Matlab

Какво представлява работата с инструменти?

- Създаване на различни видове графики
- Избиране на променливи, които директно от работното пространство на браузъра са изображения
- Лесно създаване и обработка на под изображения
- Добавяне на пояснителни бележки, като стрелки, линии и текст
- Настройване свойствата на графични обекти.

По много начини могат да се отворят лентите с интрументи и по точно работата с инструменти. Най – напред това става от командния блок, а може също дирекно от бутона, с който се запознахме по – горе наречен Show Plot. Негова иконка изглежда по този начин:

Премахването на инструментите става от бутона „премахни” от парцела с инструменти, иконата е следната: 

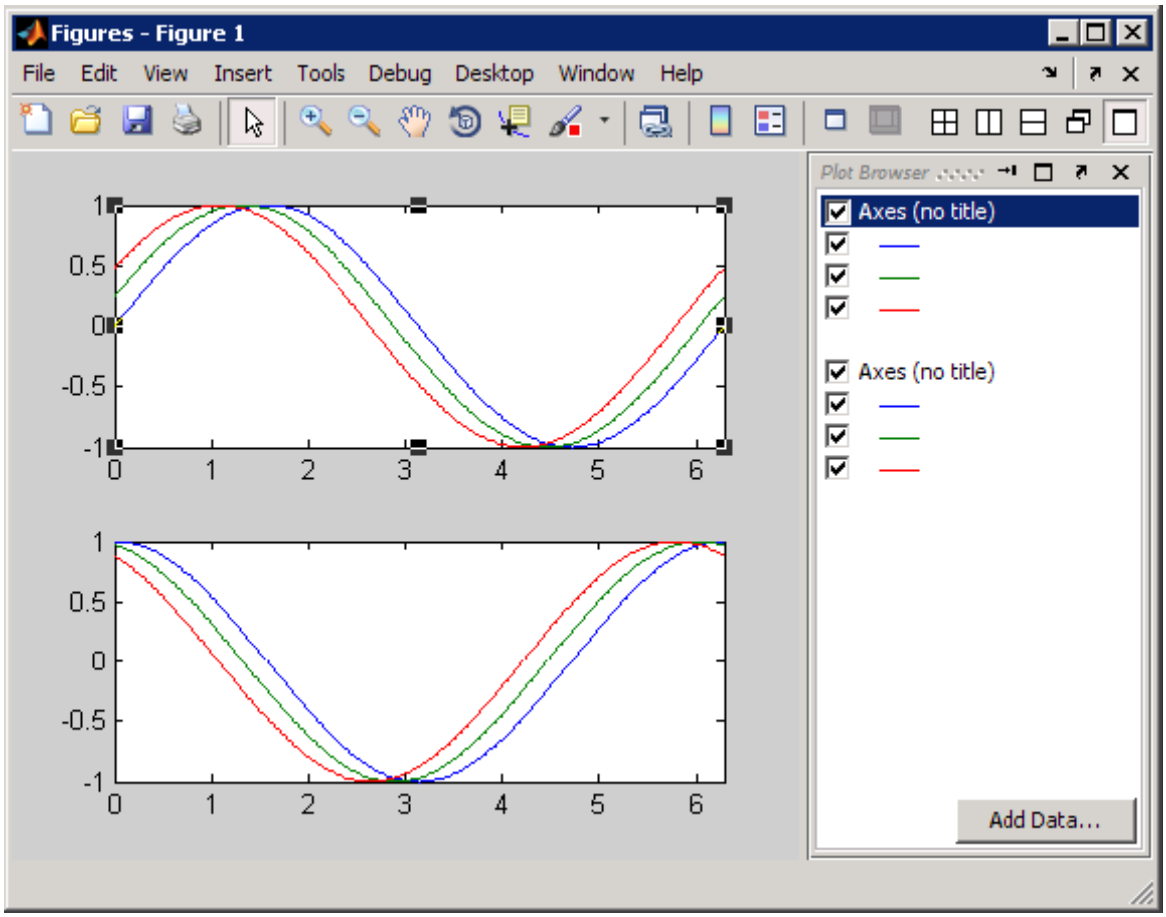
Рабатата с инструментите включва три менюта в Matlab

- Figure Palette - използва се, за да създаде и да осигури подучастък с оси и изглед на изображения.

- Plot Browser - използва се, за да се избере и контролира видимостта на оси или графични обекти, изобразени на фигурата. Може да се добавят данните към всички избрани оси, като се щракне върху бутона Add Data.
- Property Editor – използва се, за да се установят общите характеристики на избрания обект. Може също да се отвори Property Editor с помощта на командата propertyeditor. В менюто Property Editor може да се натисне върху бутона More Editor за показване на графичен интерфейс, който показва всички обекти на изображенията и позволява да се променят стойностите на всеки обект.

Всеки инструмент може да бъде закачен или освободен или да се премахне изцяло, като се щракне върху x в десния край на заглавната лента. Ако отхвърлим даден инструмент, но решим че ние необходим отново, можем да го върнем от менюто View или от въвеждането на една от командите. Plottools или Figurepalette.

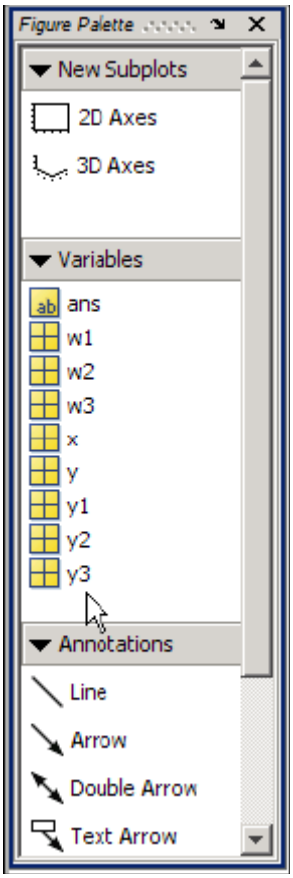
След тези операции на десктопа ще се появи това:

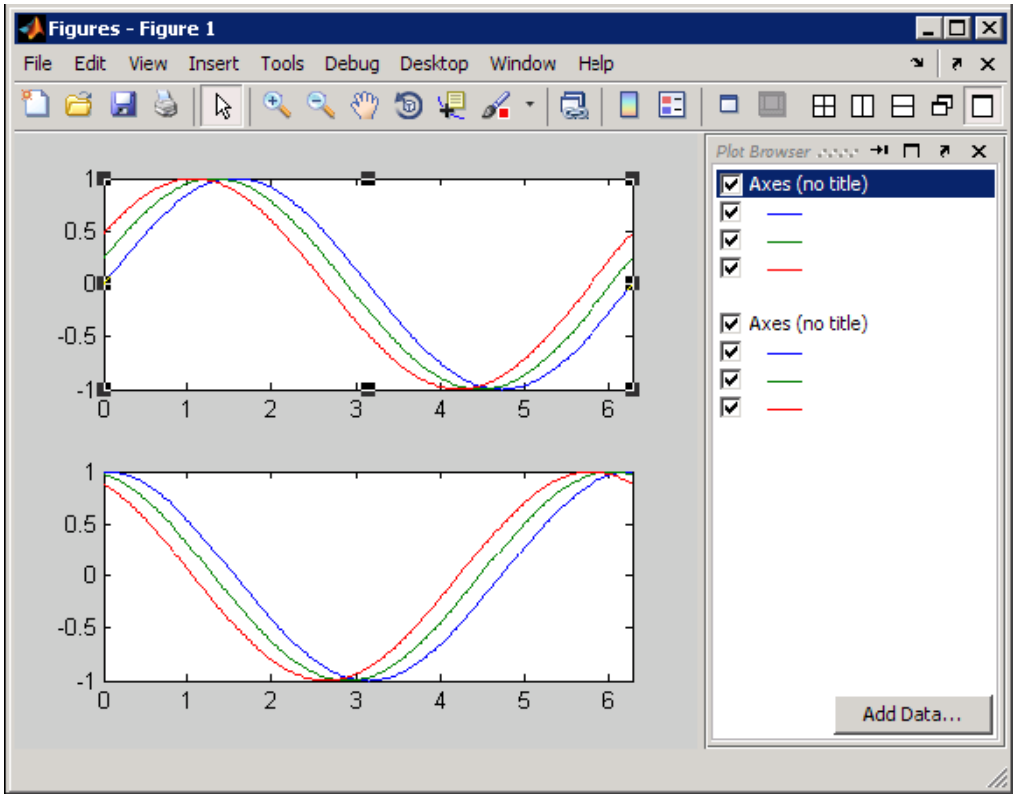


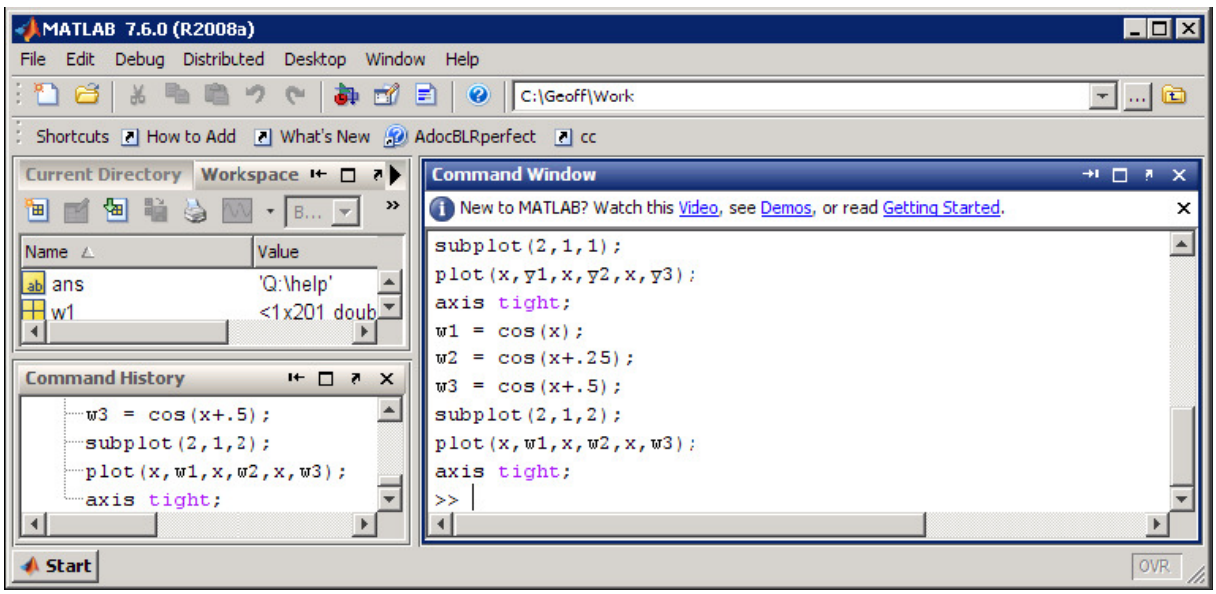
- **Групи от фигури**

Когато се активира всяко едно изображение с даден инструмент (или фигура в работния плот), тя става част от група с фигури. Групите от фигури са на десктопа, те могат да се преместят в работния плот. Отделни данни не могат да се закачват освен в рамките на групата от фигури . При създаването на следващи фигури, те също ще влязат в групата, където те могат да бъдат видяни.

Когато се фиксира изображението в групата от фигури и след това се създаде документ на групата в компютъра, инструментът е включен в тази част на работния плот, както е показано тук:



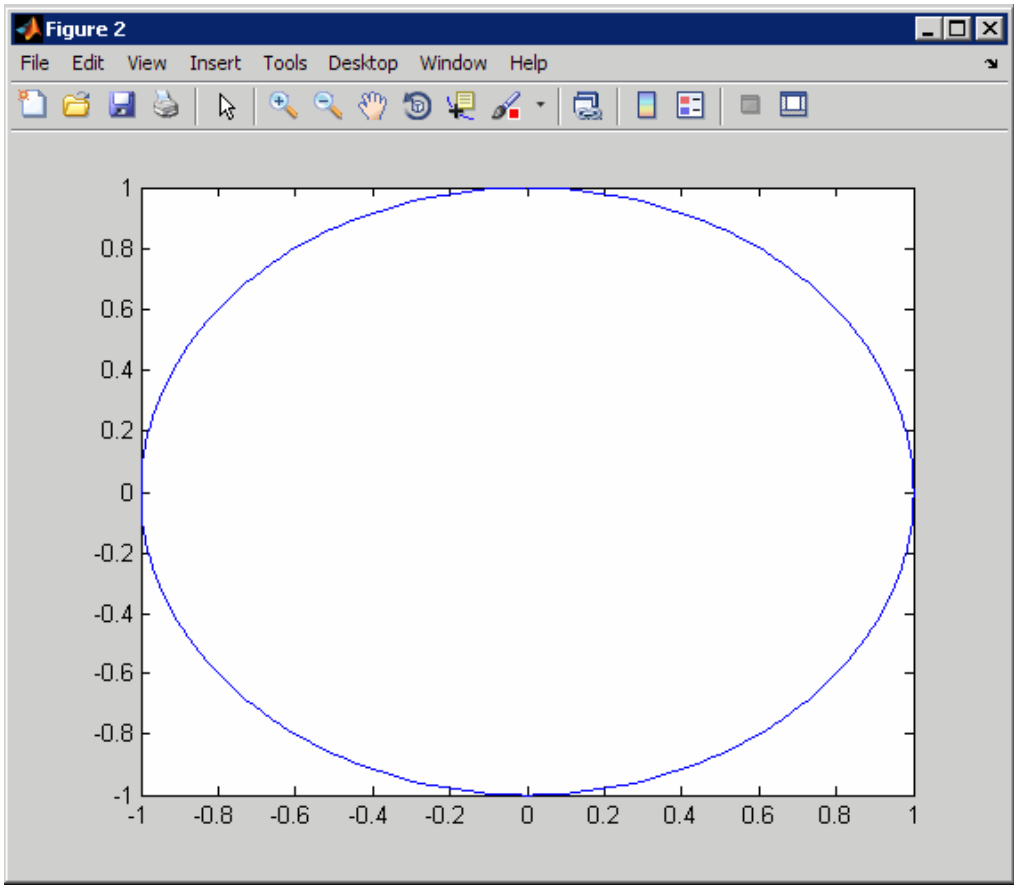




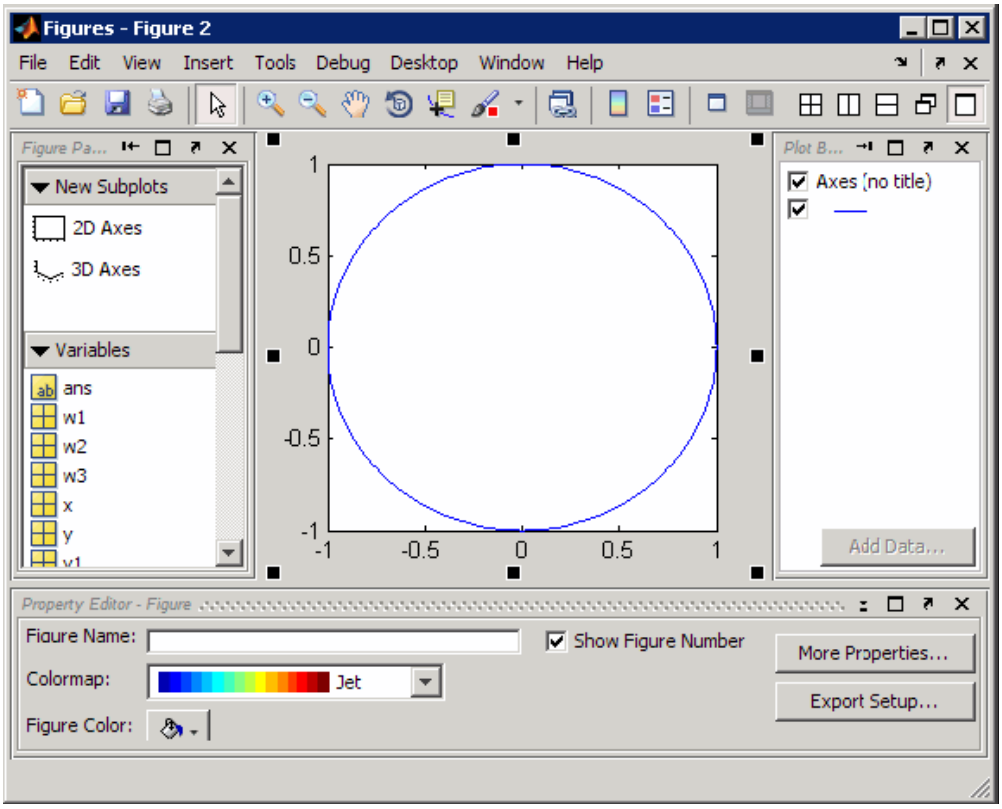
- **Работа с няколко фигури**

Когато се създава нова фигура и изображение в нея, тя е създадена без поддръжка на инструменти, дори ако друга фигура вече е отворена:

Това нещо може да се види тук:



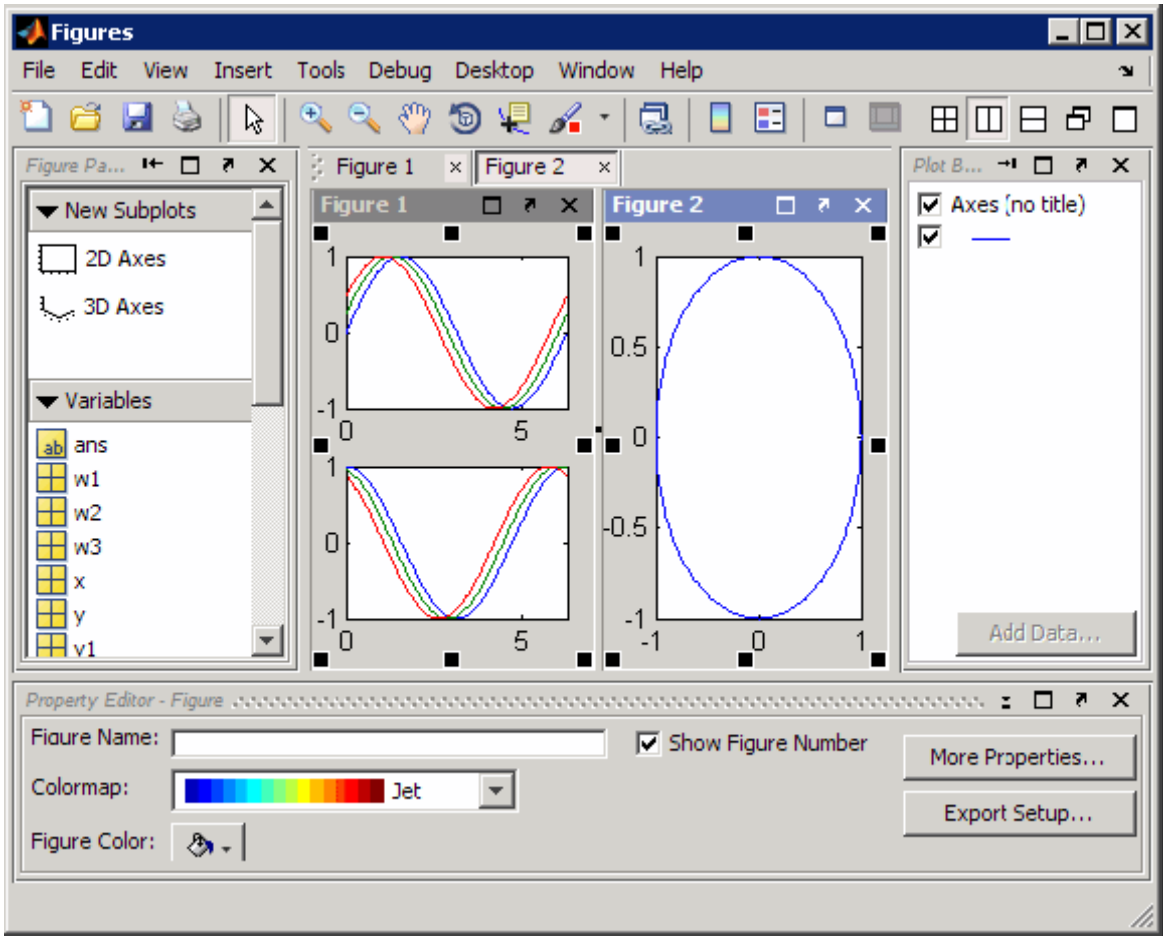
Ако след това се отвори фигурата за инструменти, като щракнете върху Open Plotti или направо се натисне върху иконата се вижда следното фугура на декстопа: 



Новата фигура, която ще се появи на работното поле може да изчезне, ако прозорецът с фигури е скрит, но ще се припокрие с вече съществуващо изображение в рамките на този прозорец. Може да се превключва между двете фигури, като се кликне върху разделите в долната част на фигурата. Трябва да се има впредвид, че като се щракне върху бутона X в дясната част на фигурата раздела изтрива тази фугира изцяло, без да пита за потвърждение.

Ако искате да видите и двете фигури наведнъж, използвайте Tilling Palette

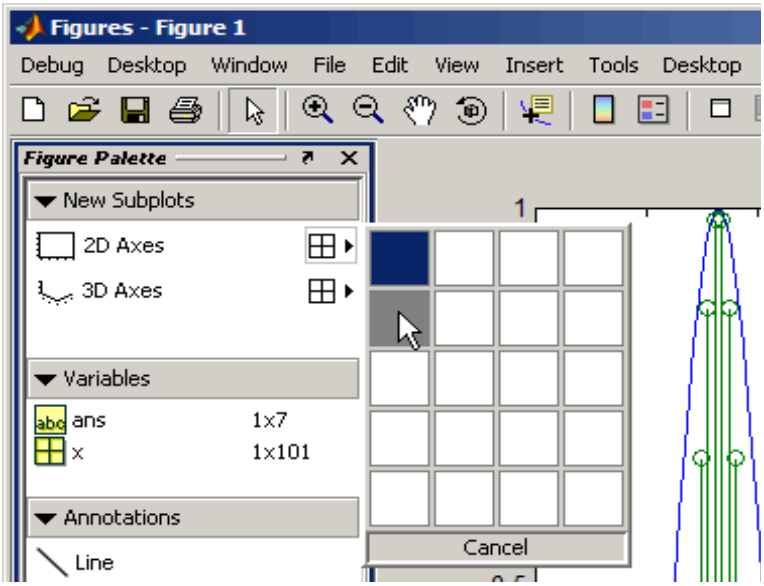
Или тези бутони, които се намират в горния десен ъгъл на прозореца, които служат за подредба на фигурите. Например, ако щракнем върху бутона наляво / надясно можем да видим двете фигури една до друга, ето по този начин:



- **Добавяне на подучастък**

Добавяне на втора ос под текущите оси чрез използване на **New Subplots.** Кликнете в десния край върху линия изправена до 2D оси и движете мишката до потъмняване на две полета едно върху друго.

Това създава подучастъци, оси под съществуващите оси. Съществуващите оси оразмеряват, така че и двете оси се вписват в схемата.



В добавените оси изберете Plot Browser и натиснете Add Data.

При добавяне на данни към диалоговите оси, въведете следните стойности:

- Задайте X Data Source за x.
- Задайте Y Data Source за sin(x).^3.
- Натиснете OK, за да се изобразят графичните данни.

Добавените графики се нанасят върху първата и като се кликне върху Add Data отново може да се определят данните за графиката:

- Задайте X Data Source за x .
- Задайте Y Data Source за sin(x).^9.
- Натиснете OK, за да се изобразят графичните данни.

- **Добавяне на заглавия и етикети (надписи)**

Изберете първата ос в Plot Browser, за да зададете следните свойства в на PropertyEditor:

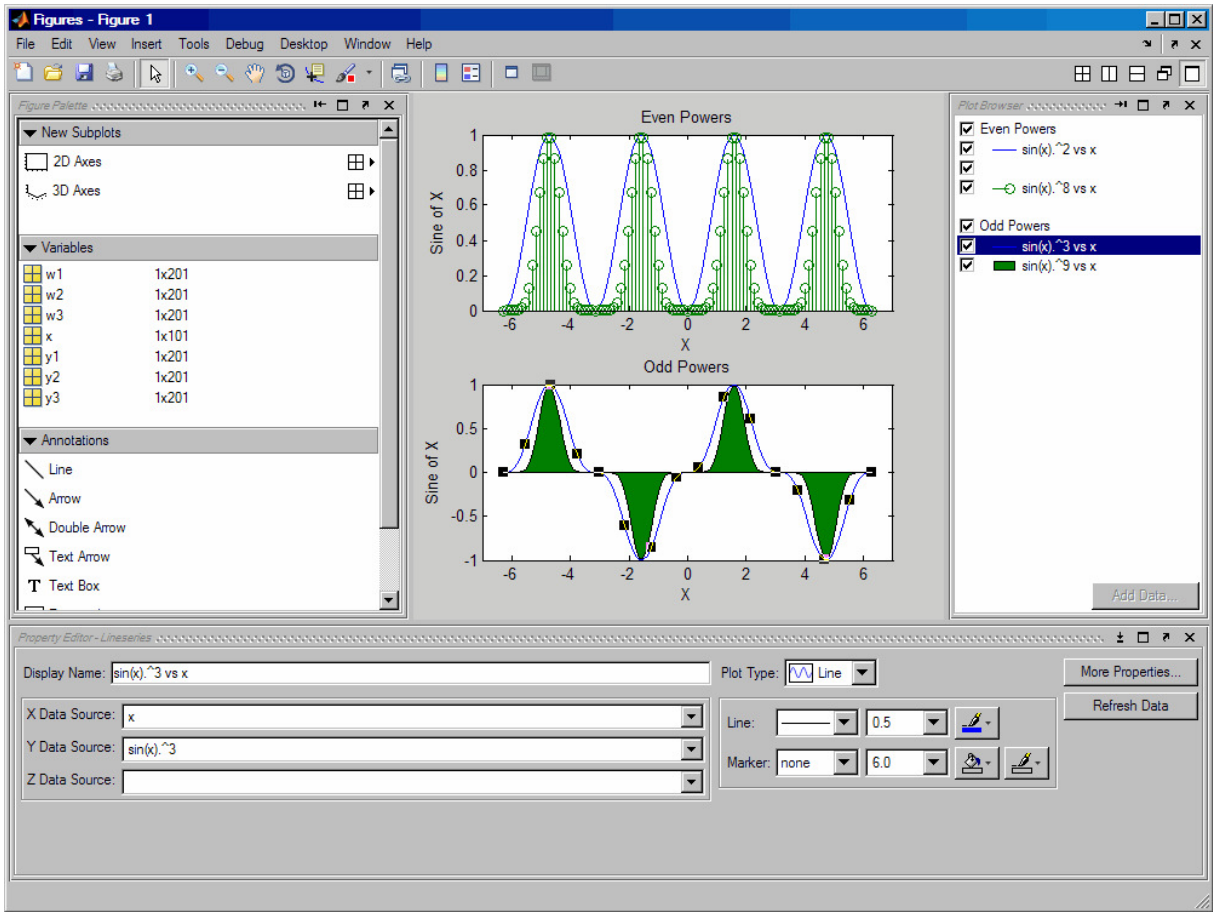
- Задайте име в EvenPowers;
- Задайте X етикет (надпис) за X.
- Кликнете върху оста Y и задайте Y Label за Sine of X

Изберете втора ос в Plot Browser и задайте следните свойства в properties panel:

- Задайте Title to Odd Powers;
- Задайте X Label за X;
- От раздела Y Axis , определяте Y label за Sine X.

В Plot Browser се отразяват наименования на нови оси.

В следващото изображение се показва резултатът от тези стъпки.

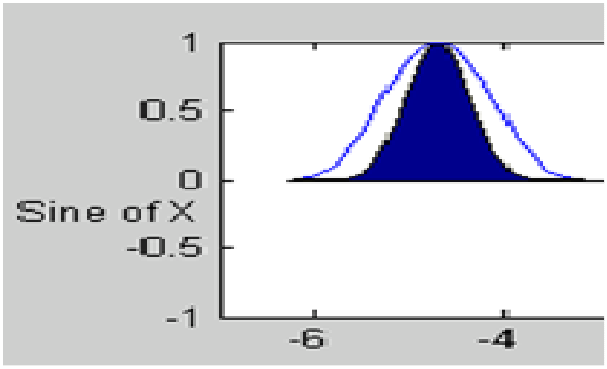


Избира се текста на оста Y на първата ос и се натиска бутона More Properties на PropertyEditor. Задава се Rotation property на 0 и се позиционира текста.

За да се направи повече пространство за оста Y, надписът, който е в хоризонтално положение се премества надясно с мишката.

Тази процедура се повтаря за вторите оси (labeled Odd Powers in the Plot Browser).

Позиционираният текст прилича на следното изображение:



II. Увеличение в 2D и 3D

Увеличение на промените и увеличение на графика може да осъществите без да се променя размера, броя или осите. Увеличението е добре да се види по-подробно в една малка област. Като в обяснението по-долу, мащабирането е различно в зависимост от това дали се прилага по отношение на 2D или 3D изглед.

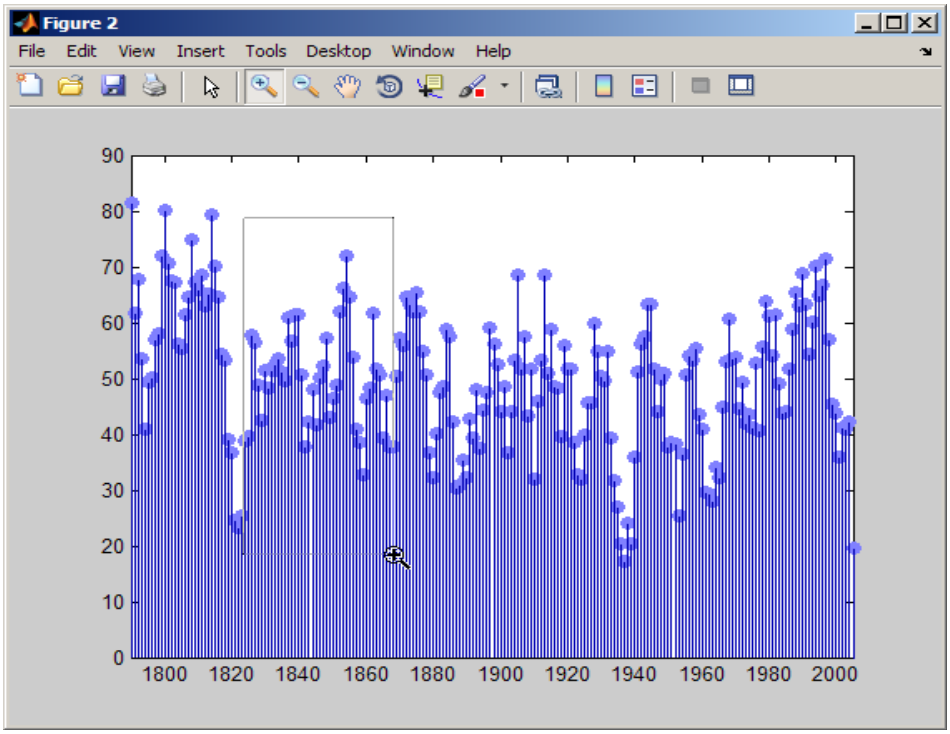
Осъществяването на мащабирането става като се кликне върху една от иконите на увеличение  . Избера се ’’ + ‘’, за увеличение и ‘’- ‘’ за отдалечаване.

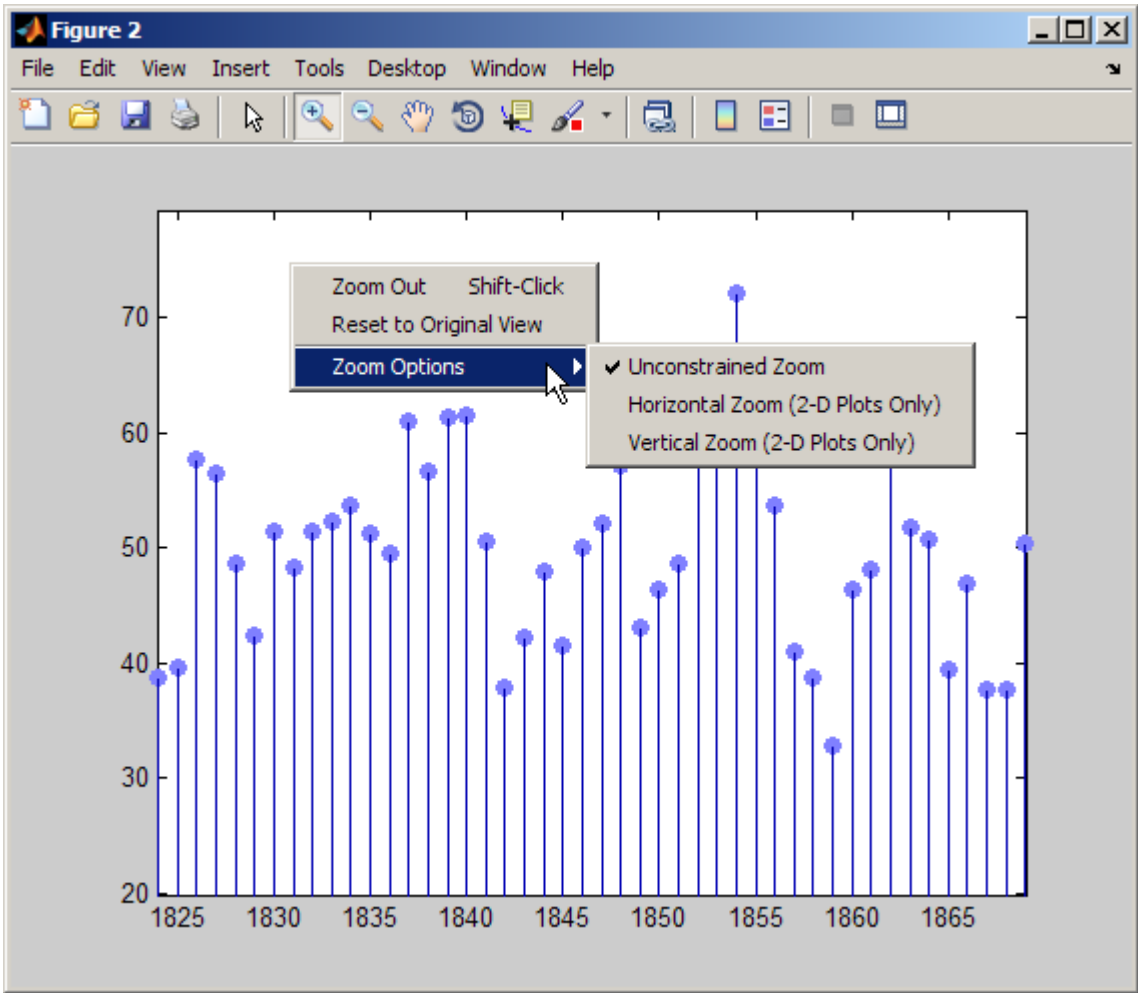
Когато има увеличен режим, можете да се използва Shift + кликване за отдалечаване (т.е. натиска се и се задържа натиснат клавиша Shift, докато щраквате с мишката). Може също да се щракне с десния бутон и да се отдалечи, възстановявате графиката към първоначалната си цел, използвайки контекстното меню.

1. Увеличаване на 2D изображения

В 2D кръгозора се кликва върху зоната на осите, където желаете да увеличите или плъзнете курсора, за да се направи кутия около областта, която желаете да увеличите. Оста е преобразена и промяната на граници за показване на определена територия.

Например избира се областта от следната графика, води до промяната на оси, които да показват само тази област.





Горната фигура показва контекстното меню при щракване с десния бутон в Zoom режим. Тя дава възможност да се:

- Намалява

- Да се проучва графиката, когато е изобразена (връща се една или повече промени на видимост)

- Ограничава се приближаването за разширяване само на оста x (хоризонтално мащабиране)

- Ограничава се приближаването за разширяване само на оста x (вертикално мащабиране).

- **Отмяна на увеличителните действия**

Ако трябва да върнем графиката към първоначалната си цел, се щраква с десния бутон, за да се покаже контекстното меню и се избера Reset to Original View (възстановяване на оригиналната версия). Може също да се използва Undo, точка от менюто Edit, за да се отмени всяка операция, които сте извършили върху графиката.

- **Хоризонтално и вертикално увеличаване**

В 2D изглед може да се ограничи увеличението, за да работи както в хоризонтална, така и във вертикална посока. За да се направи това се щраква с десния бутон, за да покаже контекстно меню, докато е в увеличения режим и се избира желаното ограничение от Zoom Options, както е показано във фигурата. Хоризонталното мащабиране е от полза за проучване на графики, динамични редове, които са плътни интервали от време. Вертикалното мащабиране може да помогне да се видят малки вариации в местата, където Y данни е малък в сравнение с оста Y граници.

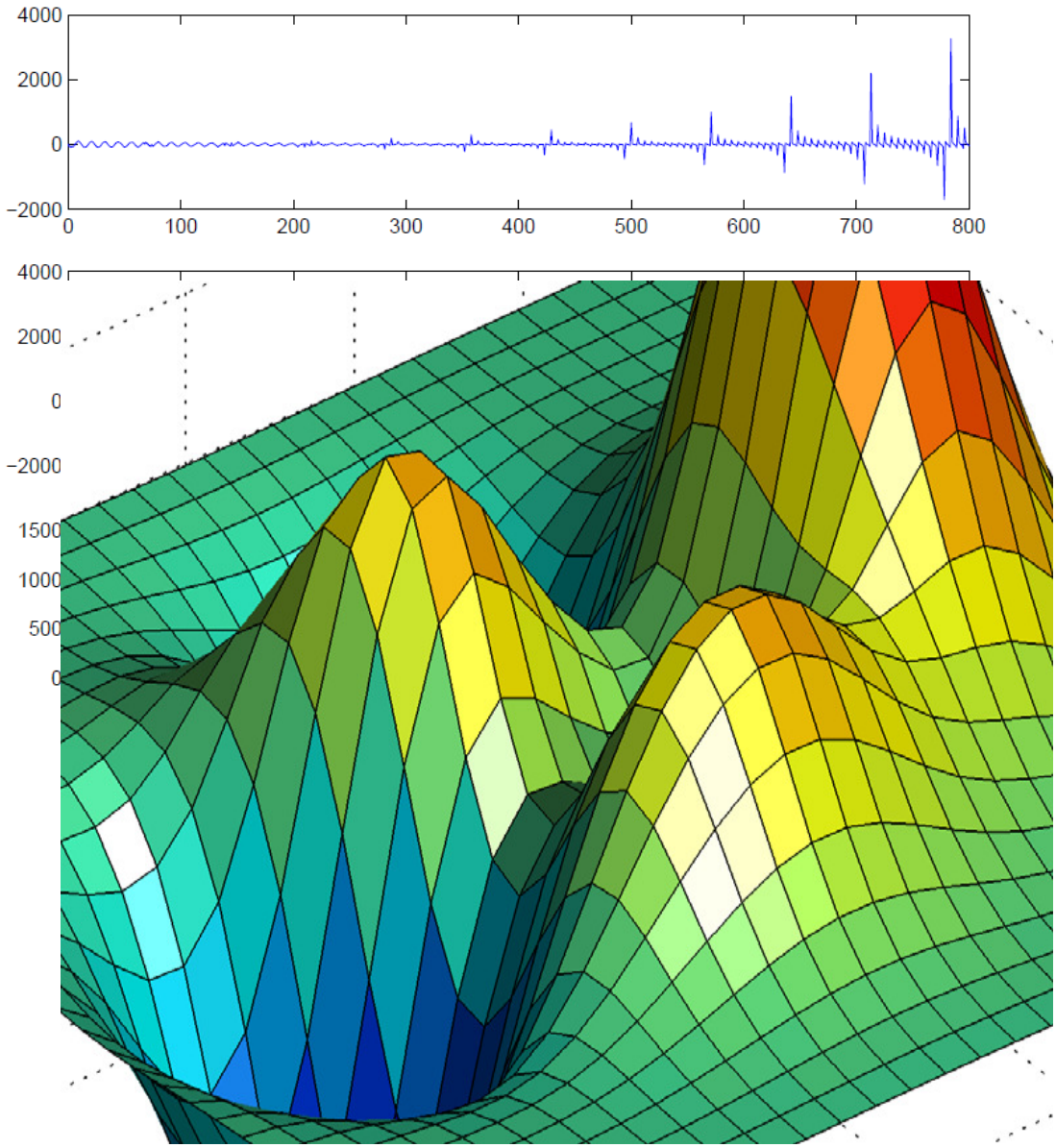
- **Визуално увеличение в 3D**

В 3D изгледи, с преместването на курсора нагоре и надясно, надолу и наляво съответно се увеличава и намалява изображението. И двете лентови икони осигуряват получаването на същия резултат. 3D мащабирането не променя осите, границите, както в 2D мащабиране. Вместо това, той променя становището, като че ли гледа през камера с обектив.

- **Прехвърляне на изгледа върху графика (диаграма)**

Може да се премести изгледа на графиката нагоре и надолу, наляво и надясно с помощта на инструмент. Текстът е от полза, когато е увеличен по графиката и искате да видите различни части от него.

Може да се движите и по двата изгледа 2D и 3D. Тестови изображения за промяна на граничната ос в 2D формат, която разглеждате не променя действителната граница на графиката. Да предположим, че имате времева поредична форма на начупеност, която искате да увеличите, за да видите детайли, но вие искате да е в състояние да сканира цялата графика (диаграма).



В обекта на 3D текстовите ходове на осите изгледа не е приведен в съответствие с X, Y или Z осите. Осите и границите не се променят, както в 2D текстовете.

2. Завъртане на 3D - Интерактивна ротация на 3D изображения

- **Възможност за 3D въртене**

Може лесно да се въртят графиките във всички посоки с мишката. Въртенето предполага преориентиране на осите и всички обекти, които съдържат графики. По тази причина данните за определяне на графични обекти се влияят чрез ротация, вместо ориентацията на X, Y и Z осите и промените по отношение на зрителя.

Има три начина да се даде възможност за режим на 3D завъртане:

- От менюто Tools да се избере 3D Rotate.
- Кликване върху Rotate 3D икона в лентата с инструменти  .
- Изпълняване на rotate3d команда.

Натиска се и се задръжа бутона на мишката, докато се придвижва курсорът и графиката се завърти.

- **Избор за дефиниране на изображения**

При включването на 3D режим, може да се контролират различни варианти на въртене като се кликне с десния бутон в контекстното меню.

Може да се завърти предварително определена точка като се кликне с десния бутон в контекстното меню:

- Възстановяване на оригиналната визия - възстановяване на изгледа по подразбиране - (azimuth37,5 °, elevation30 °).
- Отидете на View - вижте графиката X-Y по Z-оста (azimuth0 °, elevation90 °).
- Отидете на View - вижте графиката X-Z по оста Y (azimuth0 °, elevation 0 °).
- Отидете на View - вижте графиката Y-Z по оста X (azimuth90 °, elevation 0 °).

- **Начин на ротация за комплексни графики**

Може да се избере един от двата стила на въртене като се кликне с десния бутон в контекстното меню:

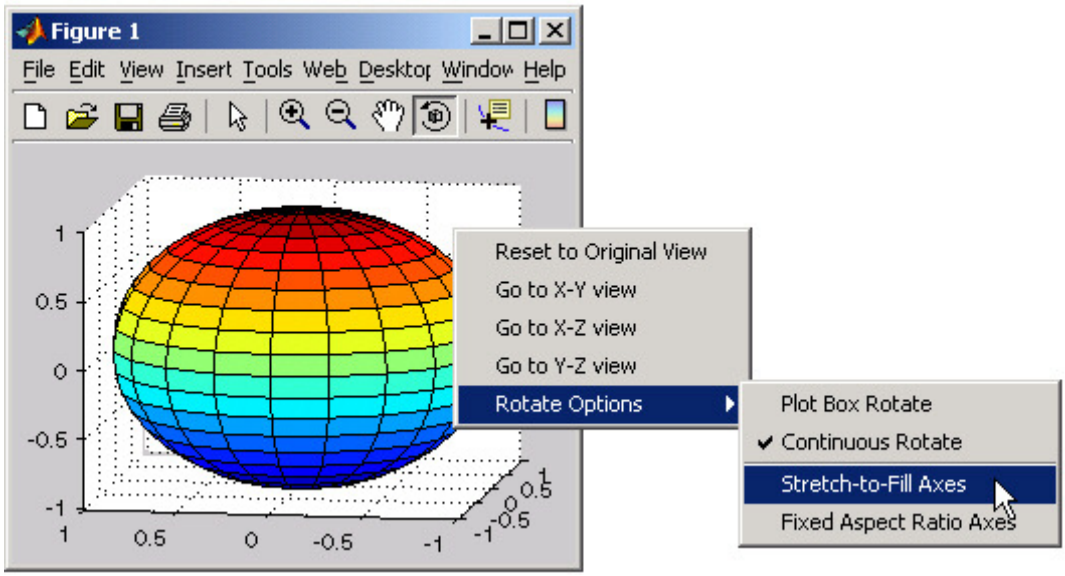
- Завъртане на графика - показване само на осите и границите в кутията за по-бързо въртене на сложни обекти. Използвайте тази опция, ако е по подразбиране непрекъснато завъртане, стилът е неприемливо бавен.
- Непрекъснато завъртане - показване на всички графики по време на въртене.

- **Поведение на оси по време на въртене**

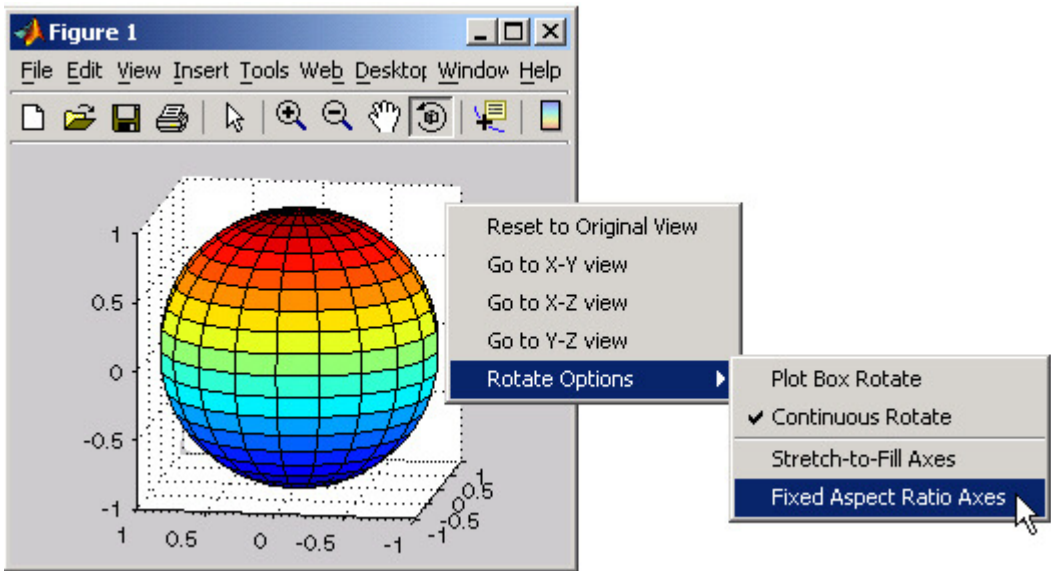
Може да се избере два вида поведение по отношение на пропорциите на осите по време на въртене:

- Просто запълване на оси - по подразбиране поведението на осите е оптимизирането на 2D графики. Графиките отговарят на правоъгълната форма на фигурата.
- Фиксирано съотношение на оси - поддържа фиксирана форма на обекти в осите, тъй като те се редуват. Използвайте тази настройка, когато 3D изображението се върти.

Картинките по-долу илюстрират една сфера, тъй като тя се завърта с разпъване на избрани оси. Забелязва се, че областта не е кръгла, поради избрания аспект.



Следващата снимка показва как съотношението на фиксирани оси и опция води до сфера, която поддържа правилна форма, тъй като тя се завърта.



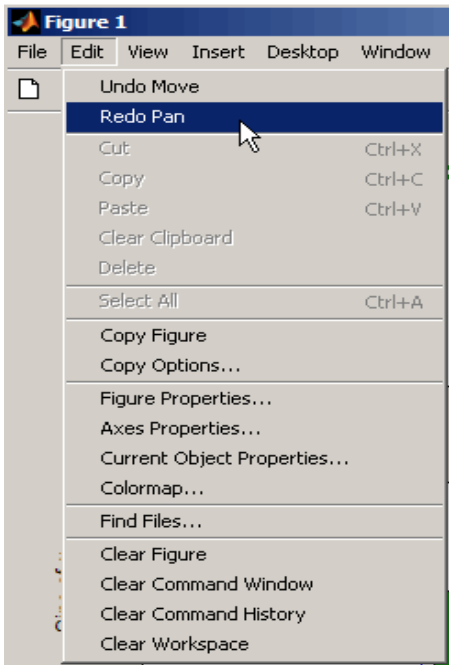
- **Undo / Redo - Премахване на грешки**

Менюто за редактиране съдържа два елемента, които позволяват да се предотвратяват увеличения, вдлъбнатини или завъртане.

- Undo- премахване на ефекта от последната операция.

- Redo (Възстановяване) - извършване отново на последната операция, която отстранява, като изберете Undo.

Например, ако създадете графика, увеличавате вдлъбнатината на прегледа и последваща отмяна на операцията, менюто изглежда по следния начин:



Вие вече можете да отмените предходното мащабиране или да възстановявате функцията вдлъбване.

III. Показване на битово-картографирани изображения

1.Работа с изображения в MATLAB графики

- **Какво е изобразяване на данни?**

Основната MATLAB структура от данни е масивът, подреден набор от реални или сложни елементи. Масивът е естествено подходящ и за представяне на изображения, реално оценени, подредени с набор от цветове или данни за интензивността. (Масивът е подходящ и за комплексно оценени изображения.)

В работното пространство на MATLAB, повечето изображения са представени като двуизмерни масиви (матрици), в които всеки елемент на матрицата съответства на един пиксел на показаното изображение. Например, един образ, съставен от 200 реда и 300 колони от различни цветни точки се съхранява като матрица 200-на-300. Някои изображения, като RGB, изискват триизмерен масив, при което първата равнина, в третото измерение представлява червен интензитет на пиксела, втората равнина представлява зелен интензитет на пиксела, а третата равнина представлява синият интензитет на пиксела.

Тази условност прави работата с изображения с графичен файлов формат подобна на работа с какъвто и да е друг вид матрица от данни. Например, можете да изберете един пиксел от матрицата на изображението използвайки нормална матрица като запишете:

I (2,15)

Тази команда връща стойността на пиксела от ред 2, колона 15 на изображението I.

Следващите раздели описват различните видове данни и изображения, и дават подробности за това, как да се разчитат, записват, и да се работи с тях, и как да се показват графичните изображения; как да се променят параметрите за визуализиране и как да се променя видът на изображението по време на това визуализиране; как да се отпечата изображение; и как да преобразувате типа данни или графичен формат на изображение.

2.Видове данни

MATLAB математика поддържа три различни числови класове за показване на изображенията:

• двойна точност с плаваща точка (double)

• 16-битово положително цяло число (uint16)

• 8-битово положително цяло число (uint8)

Командите за показване на изображенията представят различно стойностите на данните зависейки от числовия клас в който са съхранени.

По подразбиране повечето данни заемат масиви от клас double. Данните в тези масиви се съхраняват като числа с двойна точност с плаваща точка (64-битова). Всички MATLAB функции и възможности работят с тези масиви.

За изображения, съхранени в един от графичните файлови формати поддържани от MATLAB функции, обаче, това представяне на данни не винаги е идеално. Броят пиксели в такова изображение, може да бъде много голям, например, 1000-на-1000. Изображението е с един милион пиксела. Когато на всеки пиксел отговаря най-малко един елемент от масива, изображението изисква около 8 мегабайта памет, ако то се съхранява като клас double.

За да се намалят изискванията към паметта, можете да съхранявате данните на изображението в масиви от клас uint8 и uint16. Данните в тези масиви се съхраняват като 8-битово или 16-битово положително цяло число. Тези масиви изискват една-осма или една-четвърт, от паметта, на данните в double масиви.

3.Функции за разчитане, записване, и показване на изображения

Изображенията в своята същност са двуизмерни матрици, поради тази причина много MATLAB функции могат да работят върху изображения и да ги записват. В следващата таблица са изброени едни от най-полезните. (Следващите раздели описват тези функции по-подробно.)

Функция	**Целта**	**Функционална група**
axis	Осово оразмеряване и появяване на участъка.	Display
image	Визуализация на изображение (създаване на изображение).	Display
imagesc	Оразмеряване и визуализация като изображение.	Display
imread	Разчитане на изображение от графичен файл.	File I/O
imwrite	Записване на изображение в графичен файл.	File I/O

imfinfo	Информация за изображение от графичен файл.	Utility
ind2rgb	Преобразуване на индексирано изображение в RGB изображение.	Utility

4.Видове изображения

- **Индексирани изображения**

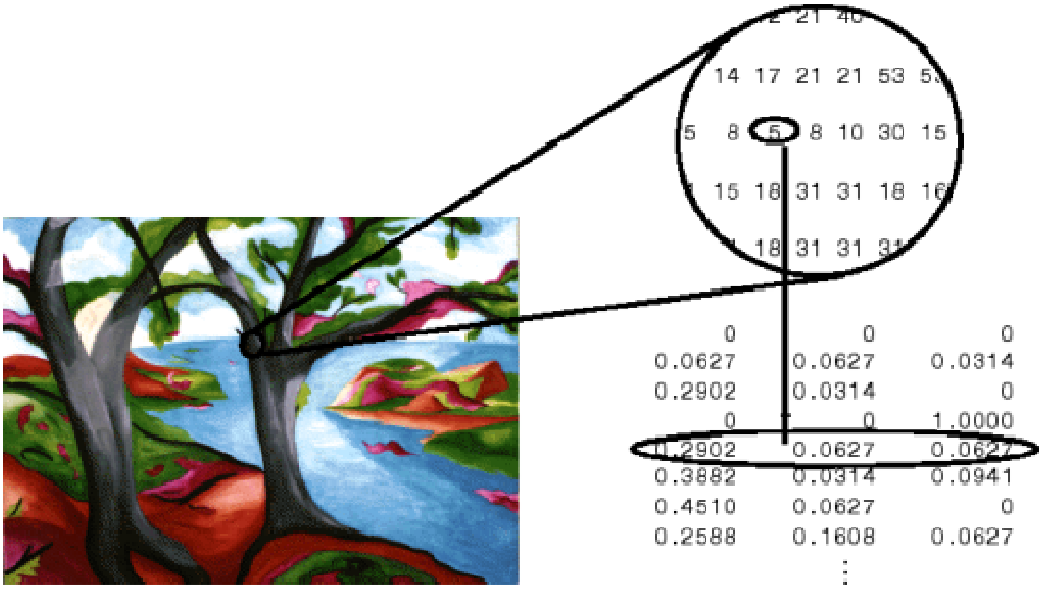
Едно индексирано изображение се състои от матрица от данни, X, и матрица от цветова палитра, map.

Map представлява m-от-3 масив от клас double, съдържащ стойности с плаваща точка в интервала [0, 1]. Всеки ред на map определя червения, зеления и синия компоненти от един цвят. Индексираното изображение използва "direct mapping (пряко картиране)" на стойностите на пиксела за стойности на цветовата палитра. Цветът на всеки пиксел на изображението се определя чрез използването на съответната стойност на X като индекс в map. Следователно стойностите на X трябва да бъдат цели числа. Стойност 1 съответства на първия ред в map,стойност 2 съответства на втория ред, и т.н. Показване на индексирано изображение става със следните твърдения

image (X); colormap (map)

Една цветова палитра най-често се съхранява с индексирано изображение и автоматично се зарежда с изображението, когато използваме функцията imread. Въпреки това не сме ограничени до използването на цветова палитра по подразбиране – можем да използваме каквато и да е цветова палитра по избор. Описанието на характеристиката CDataMapping описва как да променим видът на използваното картиране.

Следващата фигура илюстрира структурата на едно индексирано изображение. Пикселите в изображението са представени от цели числа, които са указатели (индекси) за стойностите на цветовете, съхранявани в цветовата палитра.



Връзката между стойностите в матрицата на изображението и цветовата палитра зависи от класа на матрицата на изображението.

Ако матрицата на изображението е от клас double, стойност 1 отговаря на първия ред от цветовата палитра, стойност 2 съответства на втория ред и т.н.

Ако матрицата на изображението е от клас uint8 или uint16 имаме разминаване (несъответствие) – стойност 0 съответства на първи ред в цветовата палитра, стойност 1 съответства на втори ред и т.н. Несъответствието (офсета) се използва и в графичните файлови формати за максимизиране на броя на цветовете които могат да бъдат поддържани. В предходното изображение матрицата на изображението е от клас double. Поради факта, че нямаме разминаване стойността 5 съответства на пети ред от цветовата палитра.

- **RGB (Truecolor) изображения**

Едно RGB изображение понякога съответстващо на truecolor изображение се съхранява като m-by-n-by-3 масив от данни, който определя компонентите от червен, зелен и син цвят за всеки отделен пиксел. RGB изображенията не използват палитра. Цветът на всеки един пиксел се определя от комбинацията на червения, зеления и синия интензитет съхранен във всяка една цветова равнина, където се намират пикселите. Графичните файлови формати съхраняват RGB изображения като 24-битови изображения, където червените,

зелените и сините компоненти са от по 8-бита всеки. Така се получава потенциал от 16 милиона цвята. Точността с която едно реално изображение може да бъде възпроизведено е довело до израза "truecolor изображение".

Един RGB MATLAB масив може да бъде от клас double, uint8 или uint16. В един RGB масив от клас double всеки един цветен компонент е на стойност между 0 и 1. Един пиксел чиито цветни компоненти са (0,0,0) се изобразява като черен цвят и един пиксел чиито цветни компоненти са (1,1,1) се изобразява като бял. Трите цветни компонента на всеки пиксел се съхраняват заедно с третото измерение на масива от данни.

Например червените, зелените и сините цветни компоненти на пиксела (10,5) са съхранени поотделно в RGB (10,5,1), RGB (10,5,2) и RGB (10,5,3).

5. Работа с 8-битови и 16-битови изображения

- **8-битови и 16-битови индексирани изображения**

Числата с двойна точност с плаваща точка (64-битови) са по подразбиране MATLAB представяне на числови данни. Все пак, за да намалим изискванията към паметта при работа с изображения можем да ги съхраним като 8-битови и 16-битови положителни цели числа, като използваме поотделно числовите класове uint8 или uint16. Изображение чиято матрица от данни има клас uint8 се нарича 8-битово изображение, изображение чиято матрица от данни има клас uint16 се нарича 16-битово изображение.

Функцията image може да показва 8- или 16-битови изображения, директно без да ги преобразува до двойна точност. Командата image интерпретира стойностите на матрицата малко по-различно, когато матрицата на изображението е от клас uint8 или uint16. Конкретната интерпретация зависи от вида изображение.

Ако X е от клас uint8 или uint16, неговите стойности са различни от 1, преди да бъде използван като индекс на цветовата палитра. Стойността 0 отговаря на първия ред от цветовата палитра, стойност 1 отговаря на втория ред и т.н. Командата image автоматично предоставя подходящо изменение, така че методът на изобразяване е еднакъв, когато X е от клас double, uint8 и uint16:

```
image(X); colormap(map);
```

Индексът на изменение на цветовата палитра за данни от uint8 и uint16 е предназначен да поддържа стандартните графични файлови формати, които обикновено съхраняват данни на изображение в индексирана форма с 256-входна цветова палитра. Офсетът ни позволява

да манипулираме и показваме изображения от такава форма като използваме по ефективните за паметта масиви uint8 и uint16.

- **Други 8-битови и 16-битови масиви**

Може да се използва и някои други операции с масиви от uint8 и uint16, в т.ч.:

• Смяна на формата, подреждане и свързване на масиви използвайки функциите reshape, cat, permute.

• Съхраняване и зареждане на масиви от uint8 и uint16 в MAT-файлове използвайки save и load. (Запомнете че ако зареждате или съхранявате едно изображение с графичен файлов формат трябва да използвате командите imread и imwrite.)

• Намерете индексите на нулевите елементи в масивите uint8 и uint16 използвайки командата find. Върнатият масив, обаче, винаги е от клас double.

• Свързващи оператори.

6. Четене, писане и сканиране на графики на графични файлове

- **Работа с графични файлови формати**

В естествената си форма графичните файлови формати на изображението не се съхраняват като MATLAB матрица или непременно като матрица. Повечето графични файлове започват със заглавие, което съдържа конкретни информационни бележки и продължава с растерни данни, които могат да се четат като непрекъснат поток. По тази причина не могат да се използват стандартните MATLAB I/O команди заредени и запаметени за четене и записване на графичните формати на изображението.

Така наречените специални MATLAB функции за четене и записване на данни от графични файлови формати са:

• Да разчита графичните файлови формати на изображението чрез използване на командата imread

• Да напише графичните файлови формати на изображението чрез използване на командата imwrite

• Да получава информация за естеството на графичните файлови формати на изображението чрез командата imfinfo.

Тази таблица дава по-ясна представа за MATLAB командите, които трябва да се използват при типовете изображения.

Способ	Функции за използване
Зареждане или записване на матрица като MAT-файл.	load save
Зареждане или записване графични файлови формати, например BMP, TIFF.	imread imwrite
Показване на всички изображения заредени в MATLAB работното място	image imagesc
Полезност	imfinfo ind2rgb

- **Четене на графични изображения**

Командата imread разчита всякакви поддържани от файловите изображения графики във всяка от възможните битови дълбочини. Повечето от изображенията така прочетени са 8-битови. Когато се четат от паметта, те се съхраняват в клас uint8. Основно изключение от това правило е MATLAB поддръжката на 16-битови данни за PNG и TIFF изображения; когато бъде прочетено 16-битово PNG или TIFF изображение, то се съхранява клас uint16.

ЗАБЕЛЕЖКА: *За индексирани изображения, imread винаги разчита цветовата палитра в масив от клас double, дори когато изображението само по себе си може да бъде от клас uint8 или uint16.*

Следващите команди разчитат изображението ngc6543a.jpg в променливата работна площ RGB и показва изображение използващо командата image.

```
RGB = imread('ngc6543a.jpg');
image(RGB)
```

Можете да запаметите данни на изображение с командата imwrite.

```
load clown % An image that is included with MATLAB
imwrite(X,map,'clown.bmp')
```

Отчетите създават BMP файлове съдържащи клонирано изображение.

- **Писане на графични изображения**

Когато записвате изображение с помощта на imwrite, основният метод е автоматично да се намали битовата дълбочина до uint8. Много от изображенията използвани в MATLAB са 8-битови и повечето графични формати не се нуждаят от двойна точност на данните. Едно изключение от правилото за записване на данни на изображения като uint8 е, че PNG и TIFF изображенията могат да бъдат записани като uint16. Тези два формата поддържащи 16-битови данни могат да заместят стандартното MATLAB поведение като 16uint се посочва като тип данни за imwrite. Следните примери показват написването на 16-битови PNG файлове използвайки imwrite.

```
imwrite(I,'clown.png','BitDepth',16);
```

- **Получаване на информация за графични файлове**

Командата imfinfo ни дава възможност да получим информация за графични файлове в някой от стандартните формати изброени по-рано. Информацията, която получаваме зависи от вида на файла, но винаги включва най-малко следното:

• Име на файла, включително пътя към папката, ако той не е в настоящата папка
• Формат на файла
• Номер на версията на файловия формат
• Датата на промяна на файла
• Размер на файла в байтове
• Широчина на изображението в пиксели
• Височина на изображението в пиксели
• Брой на битовете за пиксел
• Изображения от вида: RGB (truecolor), intensity (grayscale), или indexed

7. Показване на графични изображения

- **Типове изображения и методи за показване**

За показване на графично файлово изображение, използваме image или imagesrc. За пример приемаме RGB изображение,

```
figure('Position',[100 100 size(RGB,2) size(RGB,1)]);
image(RGB); set(gca,'Position',[0 0 1 1])
```

ЗАБЕЛЕЖКА: Това изображение е създадено с космическия телескоп на Научния Институт, управляван от Асоциацията на университетите за научни изследвания в областта на Астрономията. Inc., от договора на NASA NAs5-26555 и е възпроизведен с разрешение от AURA/STScI. Дигиталното предаване на изображения направени от AURA/STScI може да бъде получено безвъзмездно. Кредити: J.P. Harrington и K.J. Orkowski (Университет в Мериленд), NASA)

Тази таблица обобщава методите за показване на трите вида изображения.

Тип изображение	Показващи команди	Използване на цветова палитра
Indexed	image(X); colormap(map)	Да
Intensity	imagesc(I,[0 1]); colormap(gray)	Да
RGB (truecolor)	image(RGB)	Не

8. Контролно съотношение и размер на дисплея

Командата image показва изображението по подразбиране – големина на фигурата и оси. Изображенията се простират или събират, за да отговарят на големината на дисплея.

Понякога искаме дисплея да съвпада с пропорциите на картинната матрица. Най-лесният път до това е чрез използване на командата axis image.

Например тези команди илюстрират earth изображението в demos папка използвайки по подразбиране фигурата и осовите позиции.

```
load earth
image(X); colormap(map)
```

Продълговатата форма на изображението е резултат от разтягане на изображението, за да отговаря на осите. Използваме командата axis image, за да може съотношението на страните да е едно към едно.

```
axis image
```

Командата axis image работи чрез определяне на the DataAspectRatio от осовия обект [111] . Вижте axis и axes за повече информация как да контролирате външния вид на осовите обекти.

Понякога искаме да изобразим картина, така че всеки елемент от матрицата данни да съответства на един пиксел от екрана. За да изобразим картина едно към едно съвпадаща с пикселите от екрана трябва да променим размера на фигурата и осите. Например тези команди показват снимката на земята, така че един елемент от данните съвпада с един пиксел от екрана.

```
[m,n] = size(X);
figure('Units','pixels','Position',[100 100 n m])
image(X); colormap(map)
set(gca,'Position',[0 0 1 1])
```

Позицията на фигурата е с четири векторни елемента, които задават мястото на фигурата на екрана, както и нейния размер. Командата figure позиционира фигурата, така че долният й ляв ъгъл е на позиция (100,100) и височината и ширината на екрана съвпадат с тези на изображението. Настройката на осите в позиция [0 0 1 1] нормализира единиците, които изпълват фигурата. В резултат картината изглежда така:

Графики и 3-D визуализации

Графики

Едно от предимствата на системата MATLAB е изобилието от професионални и удобни за използване средства за визуализация на данни. MATLAB предлага отлична двумерна и тримерна графика, пълен списък на всички графични команди и функции може да се получи с командата **help**:

>> help graph2d – двумерна графика;

>> help graph3d – тримерна графика;

>> help specgraph – специална графика.

Двумерна графика

Обикновени графики

Функциите и командите на тези графики можем да разделим на три групи:

Функции, чертаещи самите графики – **plot**, **loglog**, **semilogx**, **semilogy**, **polar**;

Команди за управление на осите – **axis**, **grid**, **hold**, **subplot**;

Функции за нанасяне на надписи върху графиките – **xlabel**, **ylabel**, **title**, **legend**, **text**, **gtext**, **texlabel**.

Функции за начертаване на графики

Най-използваната функция за начертаване на графики в MATLAB е функцията **plot**. Тя може да се използва с различен брой аргументи:

plot(y) – начертава графиката на функция, чиито стойности са записани във вектора y. Като абсциси се използват индексите на отделните елементи на вектора;

plot(Y) – начертава едновременно графиките на няколко функции, представени чрез отделните колонки на матрицата Y. За абсциси се използват първите индекси на елементите;

plot(x, y) – начертава графиката на функцията y = f(x). Тук x е вектор с абсцисите, а y – вектор с ординатите;

plot(x, Y) – начертава едновременно графиките на няколко функции на един и същ аргумент: y1 = f1(x), y2 = f2(x), …. Стойностите на функциите са записани в отделните колонки на матрицата Y;

plot(x, y1, x, y2, x, y3, …) – аналогично на горния случай, но функциите са представени с двойки вектори: x – на абсцисите, y1, y2, y3,

…- на ординатите;

plot(x, f1(x), x, f2(x), …**)** – аналогично на горната команда, но

стойностите на отделните функции, за всички абсциси от вектора x, се пресмятат директно в командата **plot**;

plot(x, y1, 'str1', x, y2, 'str2', …**)** – тук аргументите са по "тройки".

Третият аргумент от всяка тройка е стринг, състоящ се от един до три *управляващи* символа. С помощта на тези символи можем да

променяме подразбиращите се стойности за *цвят* и *стил на линията* и *вида на маркера*. Всички възможни стойности на управляващите символи може да видите, след като подадете командата **>> help plot**.

Техните стойности са:

Цвят на линия

r – red – червен

g – green – зелен

b – blue – син

y – yellow – жълт

m – magenta – малинов

c – cyan – синьо-зелен

k – black – черен

w – white – бял.

Стил на линия

– непрекъсната линия;

: линията се изобразява с точки;

-. пунктирна линия (тирета с точки);

-- прекъсната линия (само тирета).

Вид на маркера

o – кръгче;

x – кръстче;

+ – символ "+";

*** –** звездичка;

s – квадратче;

d – ромбче;

v – триъгълниче с върха надолу;

^ – триъгълниче с върха нагоре;

> - триъгълниче с върха надясно;

< - триъгълниче с върха наляво.

'**LineWidth**', n – дебелина на линията в pt; n – цяло число;

'**MarkerEdgeColor**', 'symbol' – цвят на маркера (ако е отворен) или цвят на рамката на маркера (ако е затворена фигура). Стрингът 'symbol' е един от горните символи за задаване на цвят;

'**MarkerFaceColor**', 'symbol' – цвят за запълване на маркера (ако е затворена фигура);

'**MarkerSize**', n – размер на маркера в pt; n – цяло число.

Забележки:

- Ако се чертаят едновременно няколко графики, без явно указване цвета на линиите, системата автоматично подбира различен цвят за всяка графика;
- В аргумента 'str' управляващите символи от трите категории могат да се указват в произволен ред;
- Стрингът 'str' може да съдържа 1, 2 или 3 управляващи символа, но не повече от един от дадена категория;
- Командата **plot** може да обработва и масиви от комплексни числа. Ако z = x + y*i, където x и y са вектори от действителни числа, а i е имагинерната единица, то:

plot(z**)** – начертава графиката на функцията y = f(x);

plot(t, z**)** – начертава графиката на функцията x = x(t). При това комплексните части на z се игнорират! Тук t и x са вектори с еднаква дължина.

Останалите три функции **loglog**, **semilogx** и **semilogy** работят аналогично на функцията **plot**. Те могат да имат същите аргументи, като единственото различие се състои в следното:

loglog – използва логаритмичен мащаб по двете оси x и y;

semilogx – използва логаритмичен мащаб по оста x;

semilogy – използва логаритмичен мащаб по оста y.

Функцията **polar**, както показва името ù, начертава графиката на функцията r = r(φ) в полярни координати.

Синтаксис: **polar(**phi , r, 'str'**)** ;

phi – вектор със стойностите на полярния ъгъл в радиани;

r –вектор със стойностите на радиуса;

'str' – стринг, който задава стила на линията.

Команди за управление на осите

axis – управлява мащабирането и изгледа на координатните оси на текущата графика:

axis([xmin xmax ymin ymax]**)** – задава интервалите на изменение на променливите по координатните оси;

axis auto – връщане към стойности, избрани автоматично от системата;

axis equal – установява еднакъв мащаб по двете оси, така че графиката да не е деформирана;

axis square – установява еднакъв диапазон на изменение на променливите по двете оси;

axis normal – премахва всякакви ограничения по мащабирането.

Командата **grid on -** добавя мрежа към текущата графика, а командата **grid off** я премахва.

Командата **hold on** се използва, когато желаем няколко графики, създавани от последователно изпълнени команди за начертаване, да се изобразят в един и същ прозорец, при запазване всички свойства на осите. Излизането от този режим става с командата **hold off**.

Командата **subplot** се използва, когато желаем един прозорец да се раздели на по-малки прозорци, в които да се изобразят различни графики. Самата команда **subplot** не чертае графиките, а подготвя прозорчето, в което ще попадне графиката от непосредствено следващата я функция за начертаване.

Командата **subplot(**m, n, k)разделя основния прозорец на прозорчета,които можем да си представим като матрица с размери m x n.

Третият аргумент k указва номера на прозорчето, в което ще се начертае графиката от следващата я команда **plot** или коя да е друга функция за начертаване. Номерацията на прозорчетата е по редове.

Функции за нанасяне на надписи върху графиките

xlabel('string'**)** – нанасяне надпис по оста x;

ylabel('string') – нанасяне надпис по оста y;

title('string') – нанасяне заглавие в горната част на графиката;

legend('str1','str2', …) – изобразява легенда в горната дясна част

на графичен прозорец. Използва се, когато прозорецът съдържа няколко графики. За всяка графика в легендата е начертана права линия с цвета и стила на линията на графиката, а след нея следва съответния пояснителен текст, коойто е взет от поредния аргумент на функцията **legend**. Последователността на аргументите-стрингове, трябва да отговаря на последователността на построяване на съответните графики. Броят на стринговете трябва да е равен на броя на графиките.

За подробности ползвайте >> **help legend**.

MATLAB 6 предлага функцията **texlabel**, която автоматично конвертира изрази в TeX формат.

С помощта на командите **gtext** и **text** може да се извърши програмно нанасяне на текст на произволно зададено от нас място

в текущия графичен (или текстов) прозорец . Командата **text не е толкова удобна ,но** е по-универсална и е за предпочитане ,когато трябва да въвеждаме по-голям по обем текст в отделен подпрозорец, зададен с командата **subplot**. Общата форма на

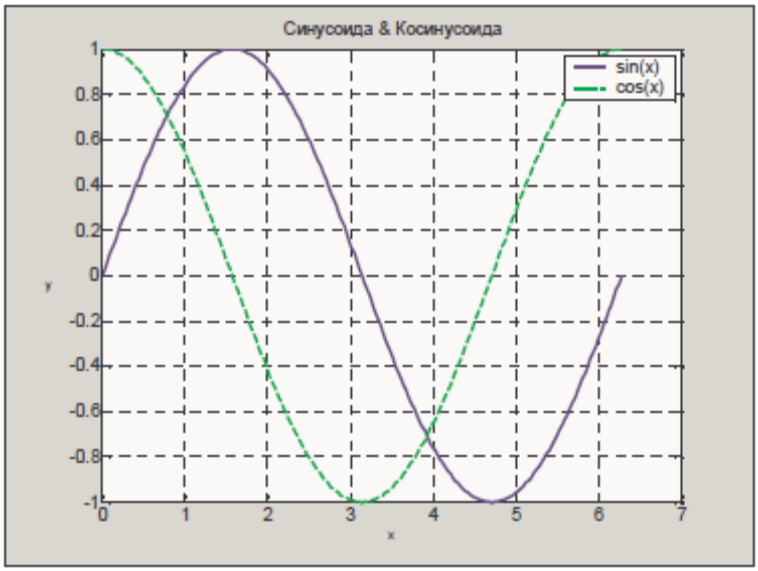
командата има следния вид:

text(x, y, 'string', '**FontName**', '<Име на шрифта>', **...**

'**FontSize**', <Размер на шрифта в pt>)

*Приложение на функцията **text:***

```
x = 0:pi/100:2*pi; % вектор с абсцисите
plot(x,sin(x),'-b',x,cos(x),'--g','LineWidth',3)
grid on, xlabel('x'), ylabel('y') % надписи по осите
set(gca,'FontName','Arial Cyr','FontSize',14) % шрифт
title('Синусоида & Косинусоида') % заглавие
legend('sin(x)','cos(x)')
```



Графика на функциите sin(x) и cos(x)

Специални графики

При обяснение синтаксиса на графичните функции в този раздел

се използват следните обозначения:

x –вектор с абсцисите;

y –вектор със стойностите на функцията;

f(x) – конкретния израз на функцията y = f(x);

f(x,y) – израза на неявно зададена функция f(x,y) = 0;

Там, където не е зададен вектора с абсцисите, функцията по подразбиране използва за абсциси индексите на вектора y.

Към специалните функции за двумерна графика принадлежат:

comet – начертава графиката на функцията y = f(x) с помощта на движеща се опашата комета, създавайки усещането за движение при визуализация на динамични процеси;

Синтаксис: **comet**(y); **comet**(x,y)

Пример:

```
t =0:0.01:50;
x = 4*exp(-0.05*t).*sin(t);
y = 0.2*exp(-0.1*t).*sin(2*t);
comet(x,y);
```

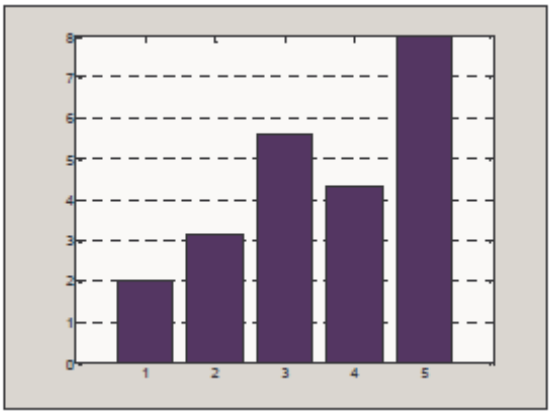
bar – чертае лентова графика;

Синтаксис: **bar**(y); **bar**(x, y)

Пример:

y = [2, pi, 5.6, 4.3, 8];

bar(y)



stem – изобразява графика само с кръгчета, свързани с абсцисната ос. Използва се за визуализация на дискретни функции;

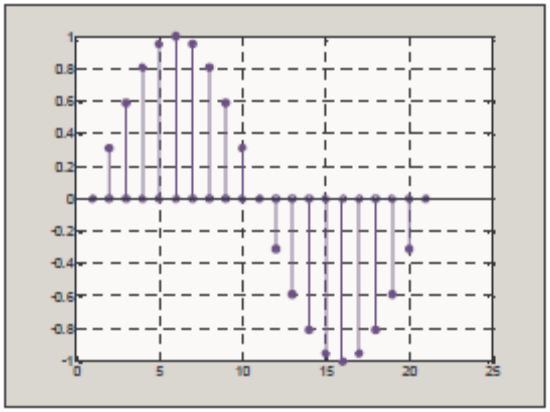
Синтаксис: **stem**(y); **stem**(x, y);

stem(x , y, '**filled**'); Ключовата дума '**filled**' се използва за запълване на кръгчетата.

Пример:

x = 0:pi/10:2*pi;

stem(sin(x),'filled');



pie – служи за построяване на кръгови диаграми;

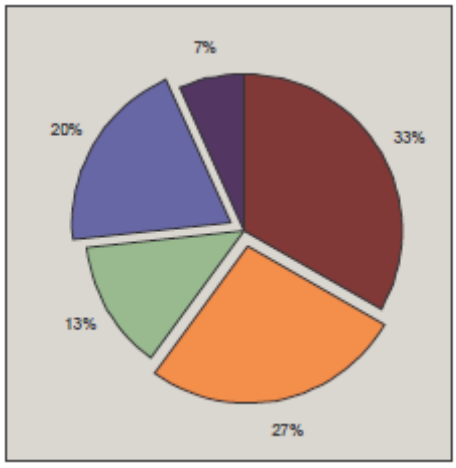
Синтаксис: **pie**(y); **pie**(y, explode);

Вторият аргумент explode е вектор с дължината на y, състоящ се от единици и нули. Секторите, съответстващи на елементите на explode със стойности, равни на 1, се изобразяват изместени от центъра навън. Номерацията на секторите започва от най-горния вляво, в посока обратна на часовата стрелка.

Пример:

y = [1 3 2 4 5];

pie(y,[0 1 0 1 0])



hist – построява хистограми, даващи нагледна представа за разпределението на случайна величина в отделните интервали от стойности.

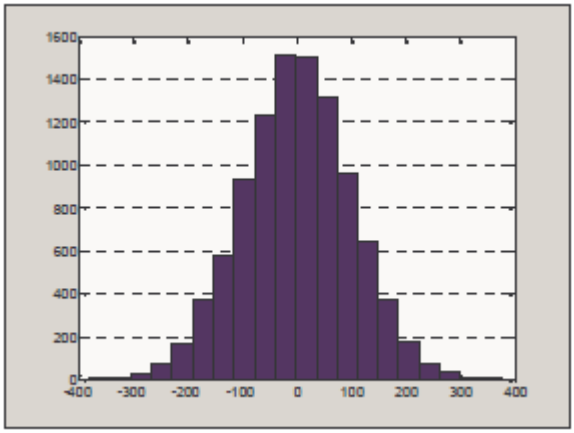
Синтаксис: **hist**(y); **hist**(y, n);

n – брой на интервалите. По подразбиране се приема 10.

Пример:

y = round(100*randn(1,10000));

hist(y,20)



3- D графики

Матлаб разполага с доста средства за визуализиране на триизмерни данни. Предлаганите средства включват:

-вградени функции за чертаене на пространствени криви, повърхнини и засенчени повърхнини

- автоматично създаване на контури

- задаване на светлинни източници

- интерполиране на цветове и сенки

Моделиране на 3-D графики

Следните стъпки показват как се построява обект в 3D:

1.В 3D графиката, един обект се дефинира чрез неговите точки (ръбове) в 3-размерно x-y-z пространство.

2.Когато тези точки се свържат с линии, се получава мрежена рамка (wire frame) представяща обекта

3.След като рамката е образувана, повърхността се прилага към обекта. Повърхността може да има много качества: цвят, текстура, блясък, отразителност и т. н.

4. Най-накрая, всеки обект или излъчва, или поема светлина. Повечето обекти са осветени от светлинен източник и трябва подходящо да се представи сянката им (много трудна компютърна задача).

Всички 3D повърхности могат да се представят като множество от полигони (които са гладки). По-сложни полигони винаги се разделят на триъгълници

3D-визуализирането се характеризира с наличието на редица функции,команди ,класифицирани по различни критерии.

Команди за криви и запълване на площи.

plot3	Чертае криви и точки в тризмерното пространство.
fill3	Чертае запълнен полигон в тризмерното пространство.
comet3	Пространствена траектория на комета.

Изобразяване на повърхнини и мрежови модели.

mesh	3-D мрежова повърхнина
meshc	Комбинация на мрежов модел с контурна графика
surf	Засенчена 3-D повърхнина
surfc	Комбинация на повърхнина с контурна графика
surfl	Засенчена 3-D повърхнина с осветяване

Управление на графичния изглед

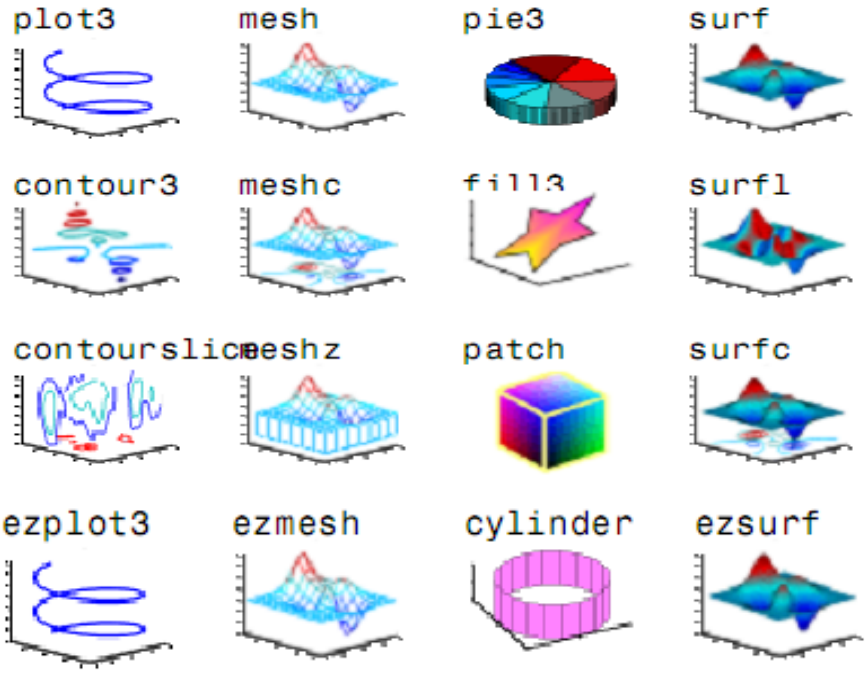
view	Задаване зрителни ъгли на 3-D графиката
hidden	Режим на скриване невидимите линии на мрежа
shading	Режим на цветно засенчване
axis	Управление мащаба и видимостта на координатните оси
caxis	Псевдо цветно мащабиране на осите
colormap	Цветова таблица

Надписване на графиките

title	Наименование на графиката
xlabel	Надпис на ос X
ylabel	Надпис на ос Y
zlabel	Надпис на ос Z (за 3-D графика)
text	Въвеждане на текст
gtext	Въвеждане на текст чрез мишката
grid	Управление на мрежата

3-D тела

cylinder	Създава цилиндър
sphere	Създава сфера



Обикновени 3D графики

Почти всички функции за управление на осите и за нанасяне на надписи, използвани в двумерната графика, се прилагат по същия начин и в тримерната.

Основните функции за изобразяване на повърхнини, описвани от функцията z = F(x,y), зададена в аналитичен или табличен вид са :

mesh – изобразяване с помощта на мрежа;

surf – изобразяване чрез непрекъсната повърхнина;

surfl – използва се допълнително осветяване (**l** от **l**ight - светлина).

Построяването се извършва в следната последователност:

1) Пресмятаме матриците X и Y на координатите x и y на възлите

на мрежата с помощта на функцията **meshgrid** [X,Y] = **meshgrid**(x,y);

x и y са вектори с абсцисите и ординатите;

С командата **meshgrid** може да се създаде мрежа от равномерни точки на данни, на която да се построи графика. Матлаб след това изгражда тази графика чрез свързване на елементите на съседната матрица, като формира мрежа от квадрати.

2) Пресмятаме матрицата Z със стойностите на функцията във

всички възли: Z = F(X,Y);

В дясната част на горното равенство трябва да е въведена конкретната функция с използване на поелементните оператори **.***, **./**, **.^** !

3) Избираме една от трите функции **mesh**, **surf** или **surfl** за

построяване на графиката:

mesh(X,Y,Z); **surf**(X,Y,Z); **surfl**(X,Y,Z)

4) Избираме някои от допълнителните функции:

colorbar – извежда се цветова лента

colormap('colmap') – цветова карта, определяща оцветяването на

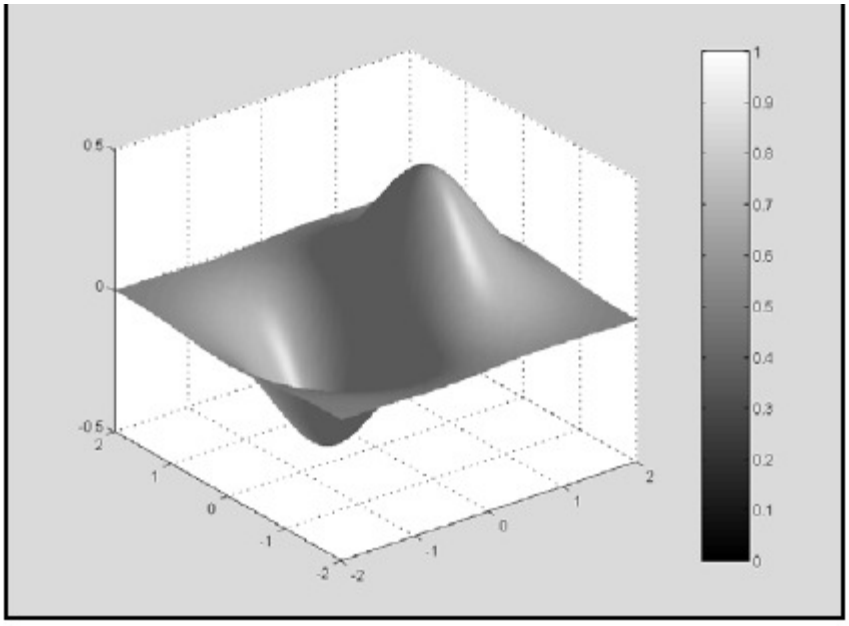
изображението. Опцията 'colmap' може да има стойности '**grey**',

'**copper**', '**pink**', '**spring**', '**summer**', и др. Пълния списък на цветовите карти може да видите с командата >> **help graph3d**.

shading interp – плавен преход на сенките

Пример: Изобразяване на функцията z = x*exp(-x^2 –y^2)

```
[X,Y] = meshgrid(-2:0.05:2, -2:0.05:2);
Z = X.*exp(-X.^2 - Y.^2);
surfl(X,Y,Z), shading interp,
colormap(' spring '), colorbar
```



Графика на функцията z = x*exp(-x^2 –y^2)

Изображения на мрежови модели и повърхнини

Функциите за изобразяване на мрежови модели и повърхнини **mesh** и **surf**, както и техните различни варианти **meshz**, **meshc** и **surfl**, приемат множество възможни аргументи, като най-простите форми са **mesh(Z)** или **surf(Z)**, където **Z** представлява матрица. Обикновено повърхнините се представят чрез стойностите на **z** координатите, изведени срещу мрежа от стойности (**x**, **y**). Следва,че за да се създаде изображение на повърхнина е необходимо най-напред да се генерира мрежа от координати (**x**, **y**) и да се определят височините (координатите **z**) на повърхнината за всяка точка от мрежата.Извършват се действия, необходими за изобразяването на функция на две променливи. За целта в MATLAB се предлага функция **meshgrid** за създаване мрежа от точки в зададен диапазон.

Визуализация на функции на две променливи

1. За да изобразим функции с две променливи трябва да генерираме **x** и **y** матрици, съдържащи повтарящи се редове и колони. Тези матрици се използват за оценка и графиране на функцията.
2. Функцията **meshgrid** трансформира домейн с 2 вектора от малки **x** и **y** матрици с големи **X** и **Y**. Тези матрици се използват ,за да се оцени функцията **Z.f(X;Y).** Редовете от X са копие от векторите на x и колоните У са копие от векторите на y.

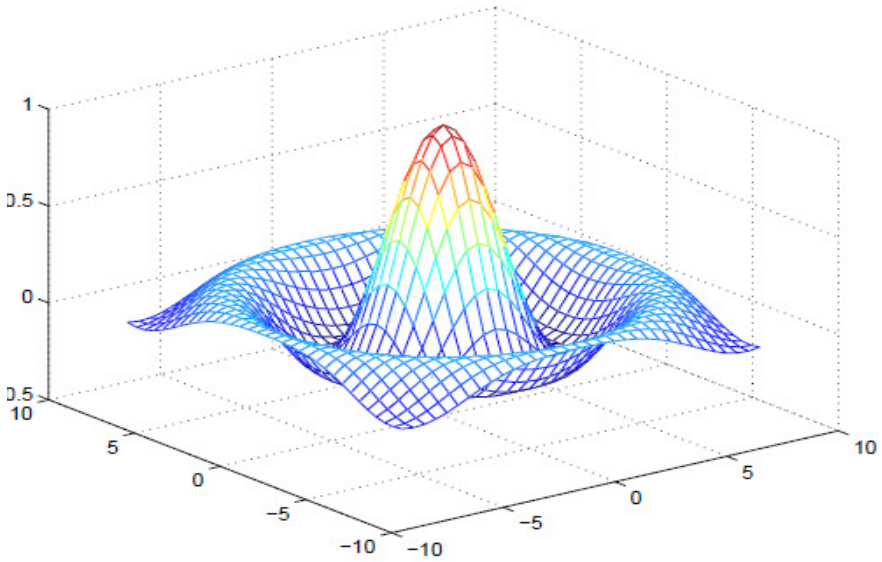
Пример: Илюстрация на използването на **meshgrid**. Стойностите на тази функция са от **-** ∞ до ∞ и двата **x** и **y** от вектора. Преминава се само един векторен аргумент от **meshgrid**, които после се използва в двете посоки.

[X, Y] = meshgrid (-∞: .5:∞);

R = sqrt (X. ^2 + 4. ^ 2) + eps

Матрицата **R** съдържа разтоянието от центъра на матрицата, който е изходната точка. Прибавянето на **eps** представлява деленето на 0 (което се извършва в следващата стъпка) . Форматирането на **sinc** функцията и проектирането на **z** с **mesh** резултатите в 3D повърхността.

```
Z = sin(R)./R;
mesh(X,Y,Z)
```



Изглед - VIEW

Гледната точка на наблюдателя се задава с командата:

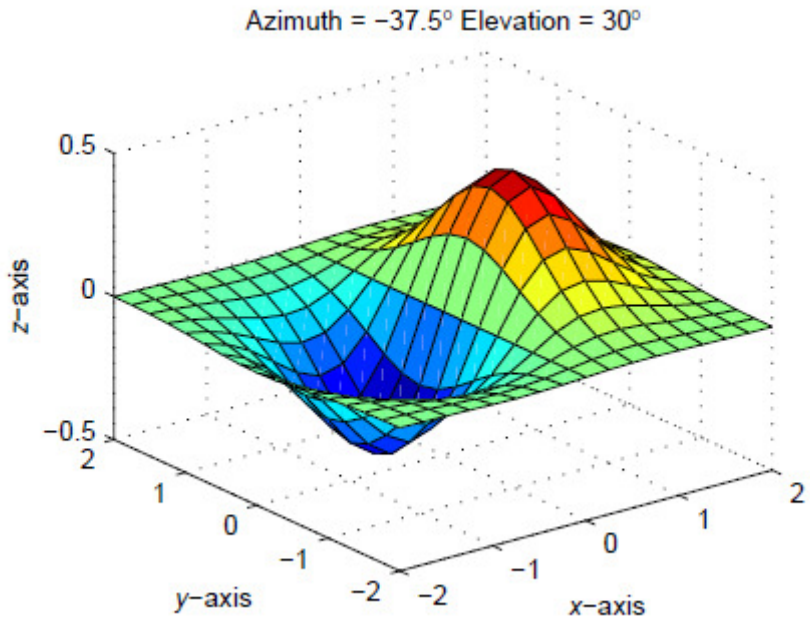
View (azimuth, vertical elevation)

където азимутът в ъглови градуси задава завъртането в хоризонталната равнина **xy**, измерван спрямо ос **z** обратно на часовниковата стрелка, а повишението, в ъглови градуси, задава вертикалния ъгъл, измерван като положителен над равнината **xy**. Заложените по подразбиране стойности на тези ъгли са съответно **-37.5**^{o} и **30**^{o}.

Чрез задаване подходящи стойности на азимута и повишението е възможно проектирането на пространствени обекти в различни проекционни равнини. Например, командата **view(90,0)** поставя наблюдателя срещу отрицателната посока на ос **x**, като се гледа срещу равнината **yz.** Така се създава проекцията на обект в равнината **yz**. Скрипт, използван

за изчисляване на данни, начертаване на криви и получаване различни изгледи с проекции е показан по-долу.

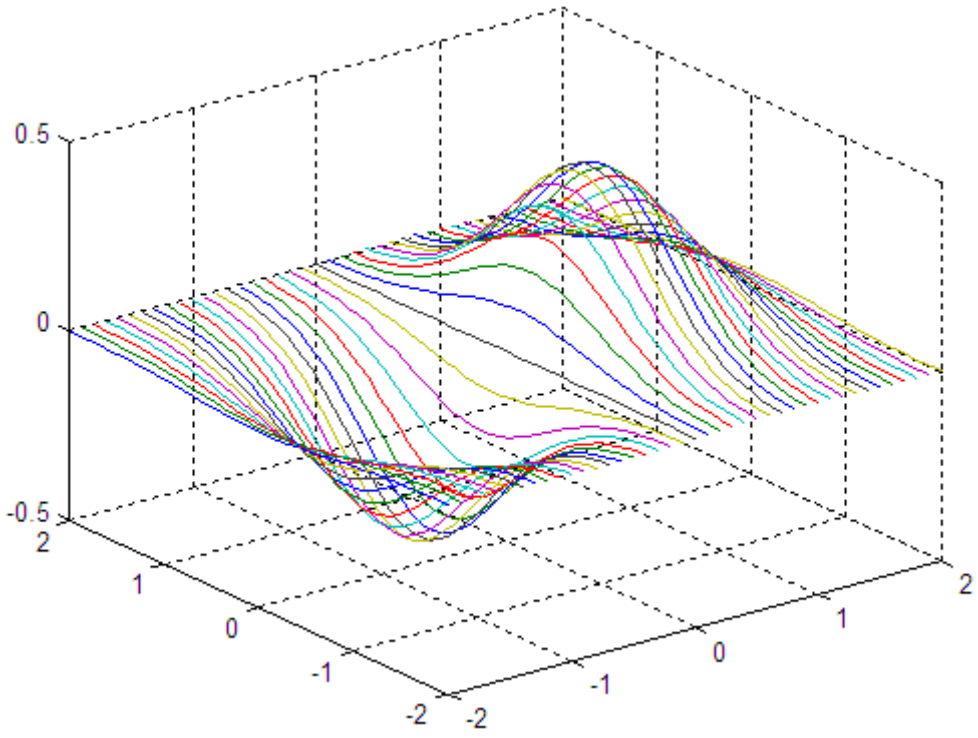
```
[X,Y] = meshgrid([-2:.25:2]);
Z = X.*exp(-X.^2 -Y.^2);
surf(X,Y,Z)
```




```
view([180 0])
```

Изтриване на скрити линии

По подразбиране Матлаб може да премахва линии, които са скрити от цветната проекци. Можем да направим това с командата **hidden off.** На следващата графика е изобразена повърхност със скрити линии:



Оцветяване на мрежи и проекции на повърхността

Матлаб може да изобрази конкретни стойности на данни с цветове, които изрично сме посочили .Също така може да състави карта на целия набор от данни с предварително определен набор от цветове, наречено палитра. Могат да се прилагат различни техники за оцветяване:

1. **Index Color Surface** – повърхността се оцветява като всяка точка от данните представлява индекс от цветната палитра на фигурата. От видът на засенчването, което е използвано, зависи начина, по който Матлаб прилага тези цветове.
2. **Truecolor Surface** - цветовете за оцветяване на повърхностната проекция са изрично посочени от Матлаб. От видът на засенчването, което е използвано, зависи начина, по който Матлаб прилага тези цветове.
3. За да бъдат извършени точно **Truecolor** изисква компютри с 24 битови дисплеи, но Матлаб стимулира **Truecolor** на база индексирани системи.
4. **Texture Mapping** - това показва 2-D изображения върху 3-D повърхност.

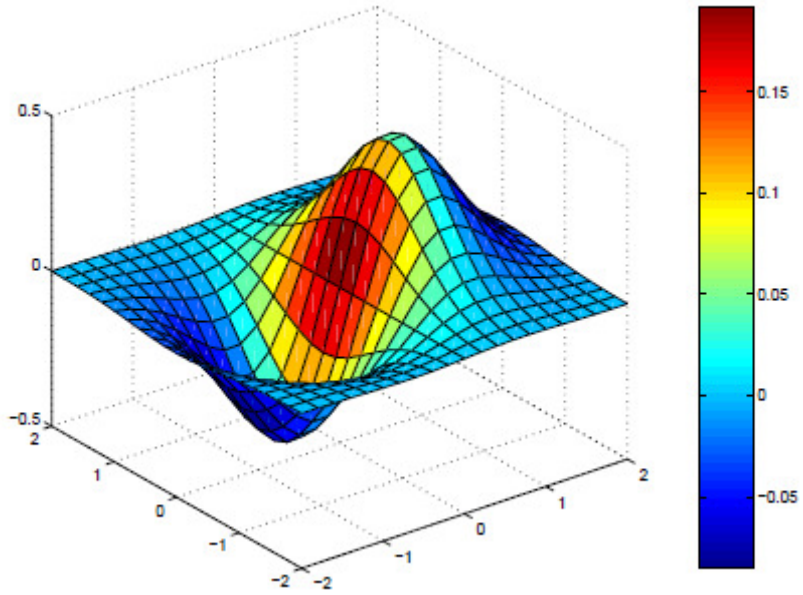
Цветни карти

Всяка фигура в Матлаб има цветна палитра. Цветната карта е проста матрица с 3 колони, чиито дължини са равни на номерата на определените цветове. Всеки ред от матрицата се определя в един цвят от определени три стойности в обхвата от 0 до 1. Тези стойности определят **RGB** компонентите (интензитет на червено, зелено и синьо, видео компонентите).

Изобразяване на цветни палитри

Функцията Colormap изобразява текущата цветна палитра, било то вертикално или хоризонтално, заедно с графиката. Например изразите :

```
[x,y] = meshgrid([-2:.2:2]);
Z = x.*exp(-x.^2-y.^2);
surf(x,y,Z,gradient(Z))
colorbar
```



Създават повърхностна проекция с вертикален стълб на който са изобразени цветовете, съдържащи се в нея (нещо като цветна палитра). На самия цветен стълб са картографирани и стойностите на данните по цветове, по оси с етикети. Матлаб може да използва два метода за индексиране на данни-цветно от цветната палитра: директно и мащабирано чрез скала.

Директно - чрез директно използване на цветните данни, които са индексирани в цветната палитра.

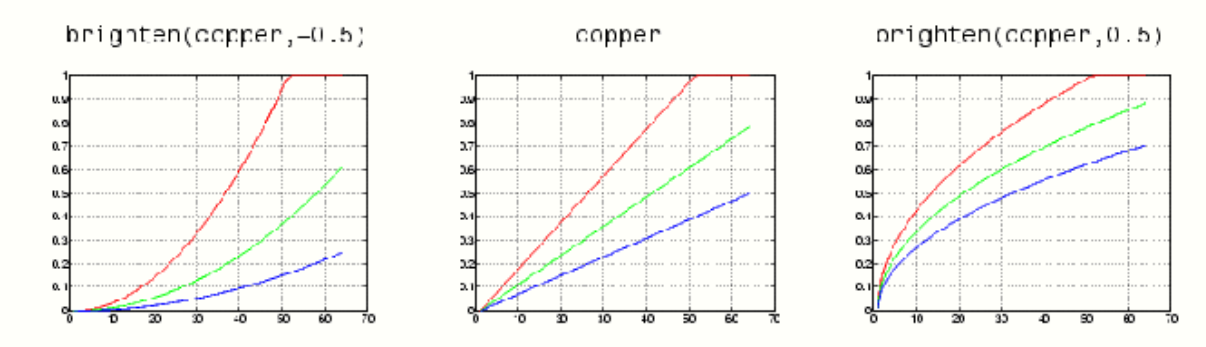
Мащабирано- то използва два елемента- векторите **[cmin, cmax].**

Cmin – определя стойностите на данните от картата на първия цвят от палитрата на цветовете

Cmax - определя стойностите на данните от картата на последния цвят от палитрата на цветовете. Стойностите на данните, които са между тях са линейно трансформирани от втората към следващата и по този начин с всяка следваща до последния цвят.

Изменящи се цветни палитри

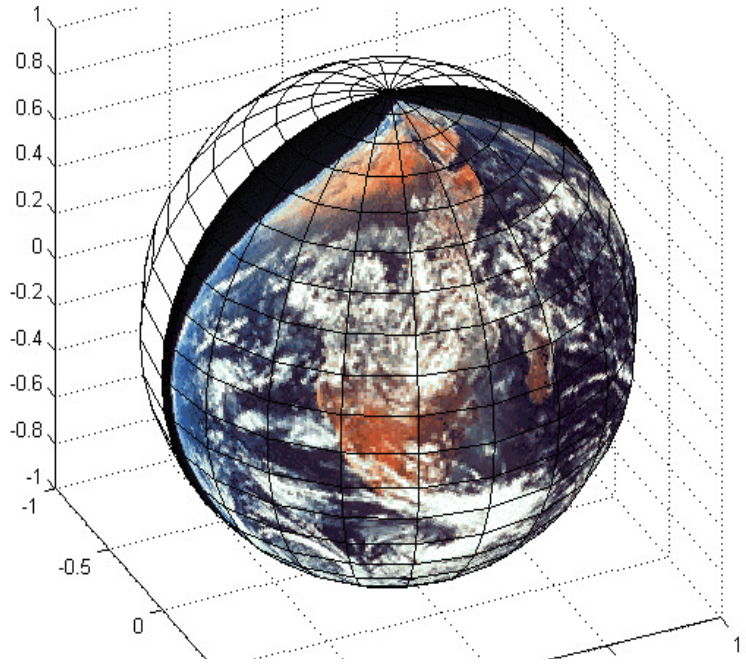
Цветните палитри са матрици и поради това могат да се манипулират като другите набори от данни. Функцията **Brighten** се използва за повишаване или намаляване на наситеността на цветовете. Поставянето на стойностите на **RGB** компоненти от цветната палитра се извършва чрез rgplot ,илюстрирайки ефектите на **Brighten**.



Текстово картографиране

Това е техника за картографиране на 2D изображения на 3D повърхност. Цветните данни се трансформират, така че да се приспособят към проекцията на повърхностт. Използвайки **''texture''** е също като да изобразяваме повърхнини на дървесна шарка без да извършваме геометрично моделиране ,което е необходимо за създаване на повърхност с тази функция. Текстовото картографиране позволява размерите на набора от цветни данни да е различен от данните, определящи повърхностната проекция. Може да се постави изображение или резюме на всяка една повърхност .

Например: Създава се сферична повърхност с изображението на Земята и тъй като това изображение се вижда само от едната страна използваме **Texture Mapping Surface**, за да можем да създадем 3D визуализация на картинката.



Специални графики

Функцията **plot3** е аналог на **plot** и служи за начертаване на пространствена крива. Извикване – **plot3**(x,y,z).

comet3 – анимационно начертаване на пространствена крива;

bar3 – като **bar**, но вместо ленти,както е в двумерните графики, се използват блокчета;

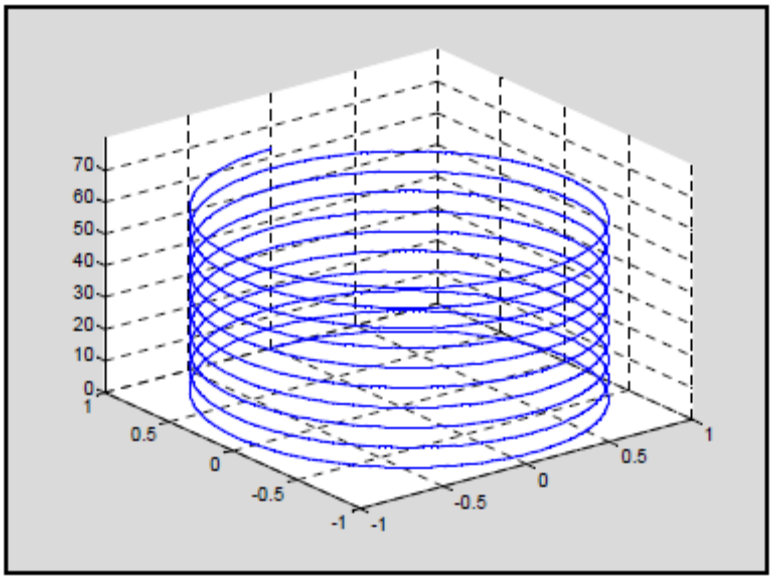
stem3 – изобразява дискретна функция z = f(x,y);

pie3 – като **pie**, но "кръгът" е пространствен.

Пример:

% Пространствена крива

```
t = 0:pi/100:20*pi;
x = sin(t); y = cos(t);
comet3(x,y,t);
plot3(x,y,t);
grid on;
```



Към функциите за облекчено начертаване на повърхнини спадат **ezmesh**, **ezsurf** и **ezgraph3**.

ezgraph3 – облекчено начертаване на повърхнина, зададена с

функцията z = f(x,y). Използва една от трите функции – **mesh**, **surf** или **surfl**, указана от потребителя в първия аргумент 'plotfun'.

Синтаксис:

ezgraph3('plotfun', 'f(x,y)');

ezgraph3('plotfun', 'f(x,y)', [a b]) – в областта a < x < b, a < y < b;

ezgraph3('plotfun', 'f(x,y)', [xmin xmax ymin ymax]);

ezgraph3('plotfun', 'f(x,y)', [a b], 'domstile'

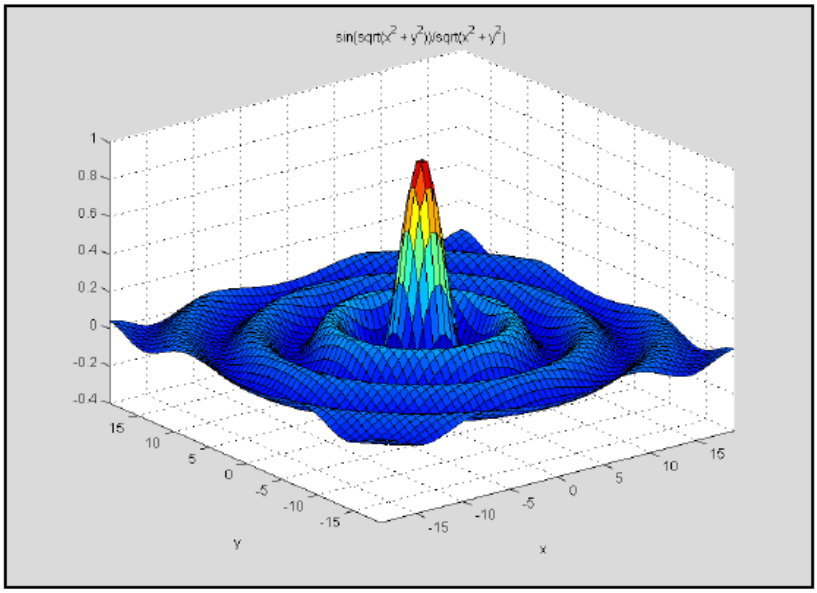
– последният аргумент задава вида на областта (domain):

'**rect**' – правоъгълна; '**circ**'-кръгова.

Пример:

ezsurf('sin(sqrt(x^2 + y^2))/sqrt(x^2 + y^2)',[-6*pi 6*pi])

title(texlabel('sin(sqrt(x^2 + y^2))/sqrt(x^2 + y^2)'))



Графика на функцията sin(sqrt(x^2 + y^2))/sqrt(x^2 + y^2)

Редактиране, печат, съхраняване и вмъкване на графика

в документ

Всяка една от създадените графики може да бъде редактирана, отпечатвана, съхранявана във файл с едно или друго разширение и вмъквана в някакъв документ.

Редактиране

В горната част на всеки графичен прозорец е разположено основно меню, а под него - бутони с инструменти. Ако желаете да нанесете допълнително пояснителни надписи в областта на графиката трябва най- напред да преминете в режим на редактиране, като натиснете бутон № 1.

След това щракате върху бутон № 2, позиционирате курсора на желаното място и натискате левия бутон. Въвеждате съответния текст в появилото се поле. След това може да свържете текста с определено място от графиката с помощта на стрелка или права линия . За целта щракате върху бутон № 3 или 4, позиционирате курсора (който променя формата си на кръстче) в началната точка, натискате левия бутон на мишката, придвижвате курсора до крайната точка и отпущате бутона. Стрелката или линията е готова.

Отпечатване

Отпечатването на графика се извършва по следния начин: От основното меню изберем : File–>Print… или натиснем съответния бутон.

Вмъкване в Word документ

С помощта на командата Edit–>Copy Figure прехвърляме графиката

в Clipboard-а. Отваряме с Word съответния документ, позиционираме курсора на желаното място и избираме Edit–>Paste.

Запис в MATLAB файл .fig

Избираме Save As … , задаваме име на файла fname.fig и натискаме бутона **save** .

Отваряне на графичен файл

Избираме File–>Open…, намираме файла, маркираме го и натискаме бутона **open** .

Запис на графика в друг формат

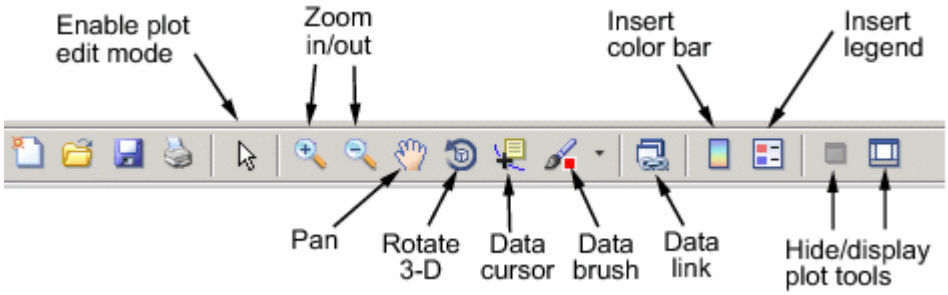
Избираме File–>Export …, задаваме име на файла със желаното Разширение и натискаме бутона **save** .

Вмъкване на графика от файл в Word документ

Избираме Insert–>Picture->From File…, намираме файла и щракаме върху него.

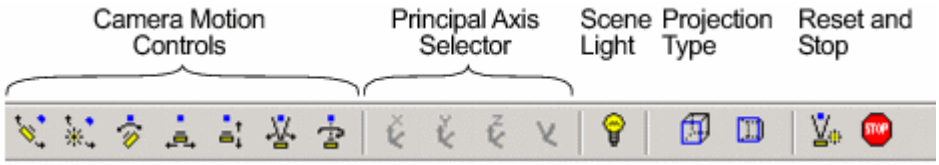
ленти с инструменти

Лентите с инструменти предоставят бърз достъп до често използвани функции. Това включва дейности като: спестяване и печат, както и инструменти за интерактивно мащабиране, преместване, завъртане, заявки и редактиране. В следната картина са показани компоненти от тази лента с инструменти.

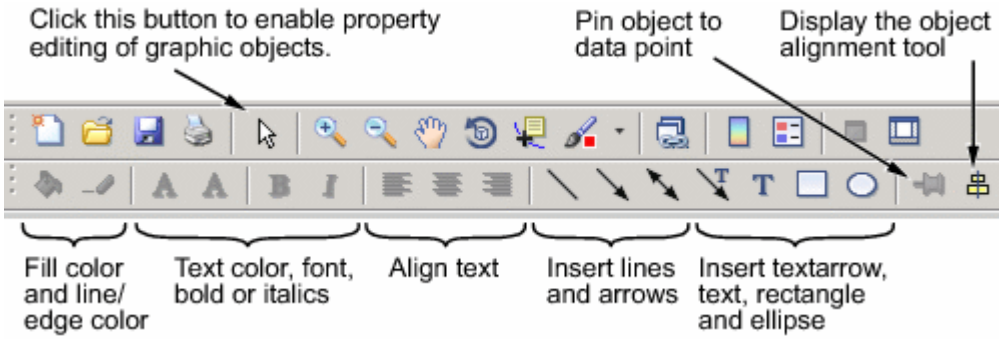


Може да се включат две други ленти с инструменти от меню View:

- Kamera Toolbar - използва се за манипулиране на 3-D изглед.



- Edit Toolbar - използва се за отбелязване и определяне свойствата на обекта.



Осветлението като средство за визуализация

1. Светлинни обекти

Светлинен обект се създава, като се използва светлинна функция.

Трите важни свойства на светлинния обект са:

• Цвят – цвят на светлината, хвърлена върху светлинния обект

• Вид – или безкрайно отдалечен (фабричен) или местен

• Позиция – Посока (за безкрайни светлинни източници) или местоположение (за местни светлинни източници)

Свойството на цвета определя цвета на ръководната светлина от светлинния източник. Цветът на обекта в една сцена се определя от цвета на обекта и светлинния източник. Свойството на вида определя дали светлинният източник е точков източник (Вид, настроен за място), който се разпространява от определена позиция към всички посоки или светлинен източник, поставен на безкрайност (Вид, настроен за безкрайност), който се появява от посоката на специфичната позиция с паралелни лъчи. Свойството на Позицията определя мястото на светлинния източник в осови единици с данни. В случай на светлинен източник на безкрайност, Позицията определя посоката към светлинния източник. Светлините влияят на повърхността и пач обектите, които са в същите оси като светлината. Тези обекти имат

редица свойства, които променят начина, по който изглеждат, когато са осветени от светлините.

Примери за контрол на осветлението

Осветлението е техника за добавяне на реализъм към графичната сцена. Това се извършва чрез стимулиране на силно осветени точки или тъмни места, които се появяват върху обекти под естествено осветление (напр., ръководната светлина, която идва от слънцето). За да се създадат осветителни ефекти, МАТЛАБ определя графичен обект, наречен светлина.

МАТЛАБ прилага осветлението към повърхността и пач обектите. Тези примери илюстрират употребата на осветление в контекста на визуализацията.

• Проследяване на линията на въздушен поток през масата – поставят се свойства на повърхността, пачове и светлини

• Използване на парчета повърхности и конусни участъци – поставят се светлинни характеристики на обектите в сцена независимо, за да се постигне желан резултат

• Осветяване на многобройни парчета повърхности независимо, за да визуализират потока на течността .

• Комбиниране на повърхности, осветени с един цвят с интерполирано оцветяване.

Осветяването като средство за визуализация

• Употребяване на осветление за разкриване на формата на повърхността. Примерът с изоповърхността на потока на течността и примерът за участъка на повърхността на sinc-функцията илюстрират тази техника.

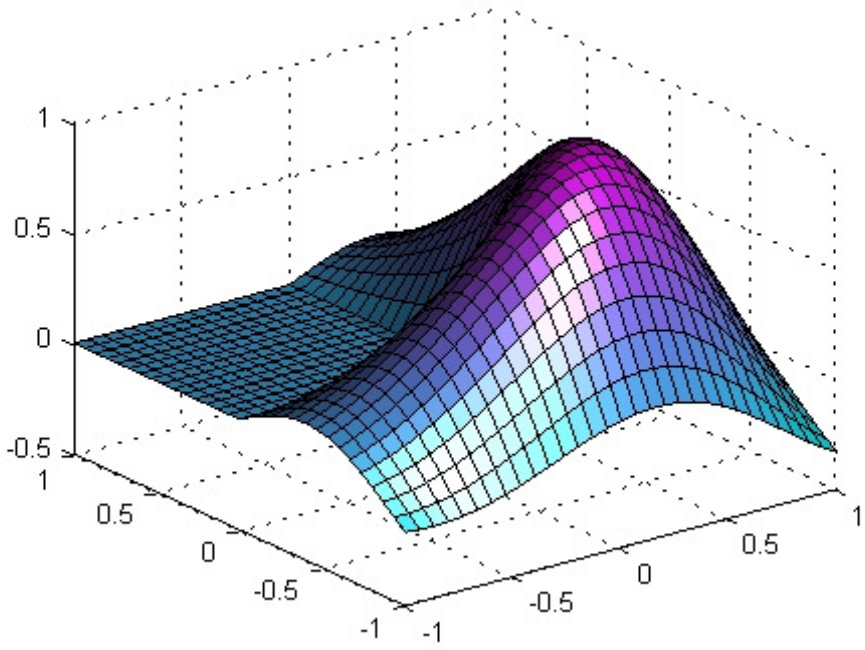
Пример: добавяне на светлини към сцена

Този пример показва мембранната повърхност и я осветлява със светлинен източник, произтичащ от посоката, определена от позиционния вектор [0 -2 1]. Този вектор определя посоката от началото на осите, преминаващи през точката с координатите 0, -2, 1. Светлината тръгва от тази посока и отива към началото на осите.

Мембрана

Светлина(‘Позиция’, [0 -2 1])

Създаването на светлина задейства редица свойства, свързани със светлината, контролирайки характеристики като заобикаляща светлина и отражателни свойства на обектите. Също така съществува превключване към изпълнение на Z-буфер, ако вече не е в този режим.



---

2. Избиране на светлинен метод

Светлинни методи на лице и ръб

Когато добавите светлини към осите, МАТЛАБ изпълнителният софтуер определя ефектите, които тези светлини имат върху пача и повърхностните обекти, които са показани в тези оси. Има различни методи, които се използват за изчисление на оцветяването на осветените обекти по лицето и ръба и този, който изберете, зависи от резултатите, които искате да постигнете.

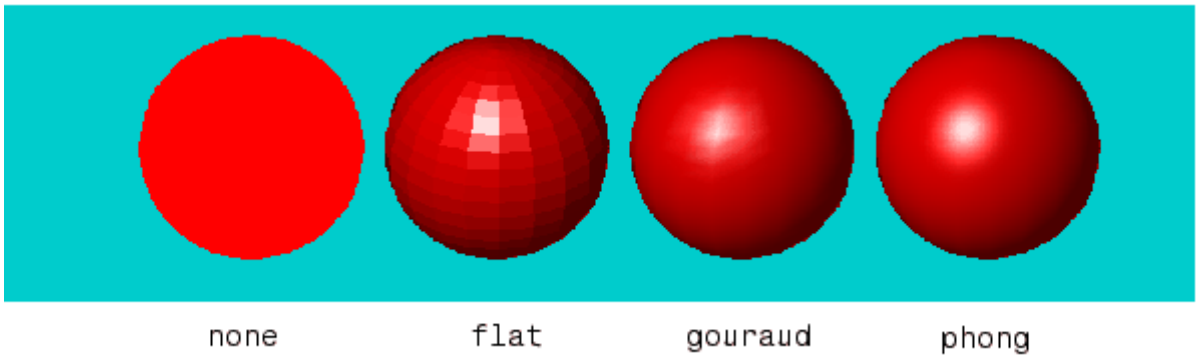
МАТЛАБ подържа три различни алгоритъма за светлинно изчисляване посредством избиране на FaceLighting и EdgeLighting свойствата на всяко едно от местата и повърхностите на даден обект в самата сцена. Всеки алгоритъм произвежда до известна степен различни резултати:

Flat lighting – произвежда еднакви цветове по всяко едно от лицата на обекта. Този метод се избира за наблюдаване на шлифовани обекти.

Gouraud lighting – преизчислява цветовете на вертикалите и след това вмъква цветовете по лицата. Този метод се избира, за да се наблюдават извити повърхности.

Phong lighting – вмъква връхните норми напряко по всяко едно лице и преизчислява рефлектирането на всеки един пиксел. Тази опция се избира за извити повърхности. Phong

lighting произвежда генерално по-добри резултати от Gouraud Lighting, но отнема повече време за изпълнение. Тази илюстрация показва как една червена сфера изглежда, използвайки всеки един от методите за осветление с един бял светлинен източник.



---

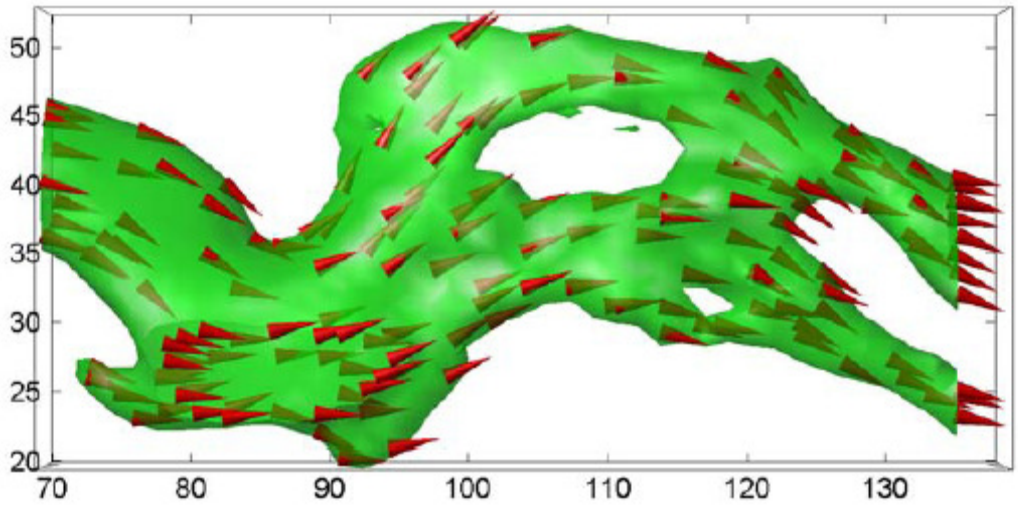
3. Прозрачност

Правенето на графични обекти полупрозрачни е полезна техника в 3-D визуализацията, за да се направи възможно виждането на обекта, докато в същото време се вижда каква информация обектът би скрил, ако е напълно непрозрачен. Също така прозрачността може да се използва като друго измерение за показ на информация; по същия начин цветът се използва в графиките на МАТЛАБ. Прозрачността на графичния обект определя степента на виждане през обекта. Може да се уточни продължителен обсег на прозрачност от напълно прозрачно (невидимо) до напълно невидимо (непрозрачно).

Обекти, които подържат прозрачност са:

- •Изображение
- •Място
- •Повърхност

Следващото изображение показва ефекта от прозрачността. Зелената изоповърхност показва конусния участък, който лежи в интериора.



4.Уточняване на прозрачността.

Стойностите на прозрачността, която варира от [0 1] се наричат стойности *алфа*. Алфа стойност от 0 означава напълно прозрачно (невидимо), алфа стойност от 1 означава напълно непроницаемо (никаква прозрачност).

МАТЛАБ третира прозрачността сходно, както третира цвета към съответния обект.

•Пачове и повърхности могат да определят единично лице и ръб алфа стойност или да използват хоризонтална или вмъкната прозрачност, базирана на стойностите на алфа картата на дадената фигура.

•Изображения, пачове и повърхности могат да определят алфа информацията, която е използвана като показатели върху алфа картата или директно като алфа стойности.

•Осите определят алфа лимита, който контролира скицирането на информацията на даден обект към алфа стойности.

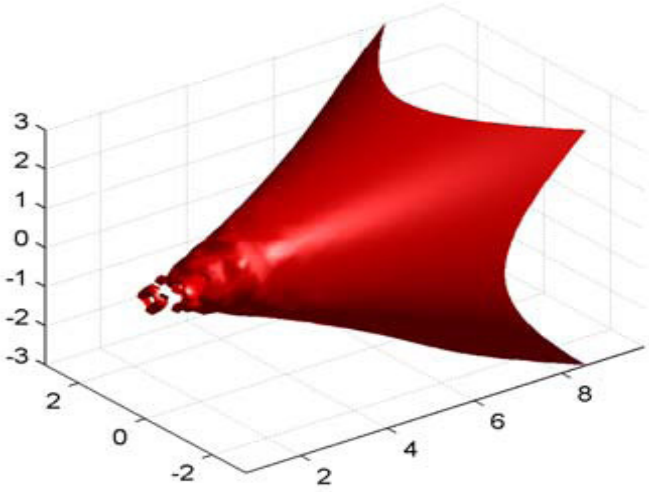
•Фигурите съдържат алфа карти които са m-by-1 подредба от алфа стойности.

Пример – прозрачна изоповърхност

Определянето на единична прозрачна стойност за графични обекти е полезно, когато искаме да покажем структура, която е затъмнена с непрозрачни обекти. За пачове и повърхности използваме свойствата на FaceAlpha и EdgeAlpha, за да уточним прозрачността на лицата и ръбовете. Следващият пример показва това. Този пример използва функцията на поток, за да създаде информация за скоростния профил на потънал джет в безкраен резервоар. Един от начините да се визуализира тази информация е чрез създаването на

изоповърхност, илюстрираща къде степента на потока е равна на специфична стойност.

```
[x y z v] = flow;
p = patch(isosurface(x,y,z,v,-3));
isonormals(x,y,z,v,p);
set(p,'facecolor','red','edgecolor','none');
daspect([1 1 1]);
view(3); axis tight; grid on;
camlight; lighting gouraud;
```



Какво е Алфа информация?

Алфа информацията е подобна на цветната информация (напр., CData свойство на повърхностите).

Когато създаватме повърхност, МАТЛАБ изпълнителният софтуер навигира всеки един елемент в цветната информационна подредба към цвят от цветната карта. По подобен начин всеки елемент от алфа информационната карта определя стойност на прозрачност в алфа картата. Трява да се уточнете повърхностна и изобразителна алфа информация със алфа информационни свойства. За пач обекти използваме FaceVertexAlphaData свойството. Можем да контролираме начина, по който МАТЛАБ интерпретира алфа информацията със следните свойства:

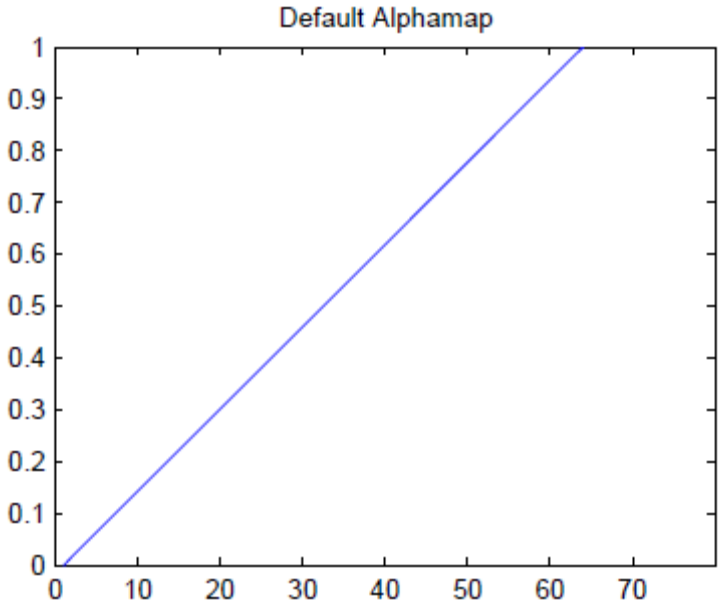
•FaceAlpha and EdgeAlpha (лице алфа и ръб алфа)

•AlphaDataMapping and Alim (определяне на алфа информация и алим)

•Alphamap (алфа карта)

Какво е алфа карта?

Накратко алфа карта е редът от стойности, които варират от 0 до 1. Размерът на реда може да бъде или m-by-1 или 1-by-m. Алфа картата с предварителни настройки съдържа 64 стойности, вариращи линейно от 0 до 1, както може да видите на следната графика.



Пример – Изменение на алфа картата

Този пример използва парчета повърхности, за да прегледа данните за масата. Парчетата повърхности използват данните за цвета за алфа информация и също така използват рампа алфа карта (стойностите варират от 1 до 0):

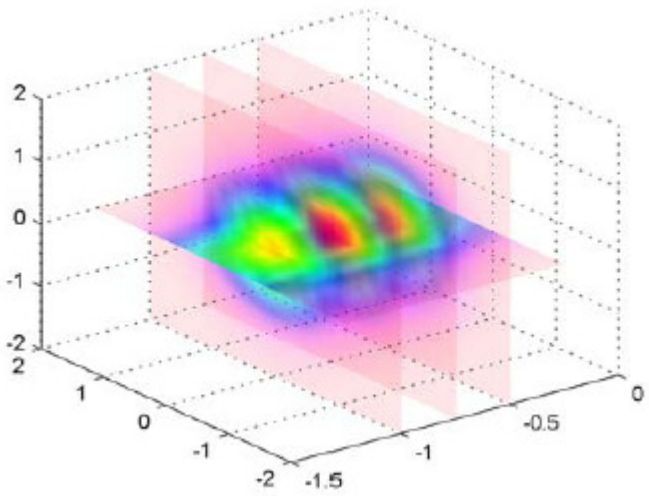
Създаване на данни за маса чрез изчисление на функция от три променливи.

Създаване на парчета повърхности, настройване на алфа информацията да е равна на данните за цвета и определяне на интерполирана FaceAlpha.

Инсталиране на рампа алфа карта и увеличаване на всяка стойност в алфа картата с. 1, за да се постигне желаната стойност на прозрачност. Определете нюанс, наситеност и стойност (hsv) на цветната карта.

Тази алфа карта поражда най-малките стойности от функцията (около нула), за да бъде показана с най-малката прозрачност и най-големите стойности, които да се опишат с

най-голямата прозрачност. Това позволява видимост през парчетата повърхности, докато в същото време се запазва информацията около нула.



Матрици и масиви

Раздел 1 : Създаване и свързване на матриците

1.Общ преглед на матриците:

Най-основната MATLAB структурата на данните е матрицата, която е двуизмерна, правоъгълно оформена структура на данните с възможност за съхраняване на множество елементи от данни в лесно достъпна форма. Тези информационни елементи могат да бъдат числа, символи, логически състояния на истина или лъжа, или дори на други MATLAB структурни видове.

MATLAB използва тези двуизмерни матрици за съхранение на един брой,а също и за линейна поредици от числа.

В тези случаи когато размерите са записани „1-по-1”и „1-по-N” съответно, където N е дължината на цифровата серия. MATLAB поддържа структури от данни, които имат повече от две измерения. Тези структури от данни се наричат масиви в документацията MATLAB.

MATLABе една матрица, основаваща изчислителни среди. Всички данни, които се въвеждат в MATLAB се съхраняват под формата на матрица, или многомерен масив. Дори една числова стойност, като 100 се съхранява като матрица (в този случай, матрица с размери 1-от-1):

A = 100;

Име	Размер	Байт(Byte)	Клас
A	1x1	8	двоен масив.

Независимо от класа им,дали те са:числови,символни или логически,верни или неверни данни MATLAB ги съхранява под формата на матрица или масив.

Например в низа "Hello World" е една до 11 матрици от индивидуален характер елементи в MATLAB. Може,също така да се изградят матрици, съставени от по-сложни класове, като структурите на MATLAB и клетъчните масиви.

2.Създаване на проста матрица

Най-простият начин за създаване на матрици с MATLAB е да се използва матричен конструкторен оператор [ ].

Създаване на ред в матрицата се осъществява чрез въвеждане на елементите (Е) в скоби.

Отделете всеки елемент със запетая или пространство:

Ред = [E1, E2, ..., Em] ред = [E1 E2 ... Em]

Например, за да се създаде един матричен ред от петте елемента,тип:

A = [12 62 93 -8 22];

За да започнете нов ред,прекратявате текущия ред с точка и запетая:

A = [row1; row2; ...; row n]

Този пример изгражда на 3 ред, 5 колона (или 3-по-5) матрица от числа.

A = [12 62 93 -8 22; 16 2 87 43 91; -4 17 -72 95 6]

A = 12 62 93 -8 22

16 2 87 43 91

-4 17 -72 95 6

3.Специализирани функции на матрицата

MATLAB има редица функции, които създават различни видове матрици. Някои създават специализирани матрици като Hankel или Vandermonde матрица. Функциите, посочени в таблицата по-долу създават матрици за по-широка употреба.

Функция	Описание
ones	Създаване на матрица, или масив от всички единици.
zeros	Създаване на матрица, или масив с всички нули.
eye	Създаване на матрица с единици на диагонал и нули на друго място.

accumarray	Разпределянето на елементи на входа на матрица с определени места в изходната матрица, също позволява натрупване
diag	Създаване на диагонална матрица от вектор
magic	Създаване на квадратна матрица с редове, колони и диагонали , които допринасят за едни и същи числа.
rand	Създаване на матрица, или набор от равномерно разпределени случайни числа.
randn	Създаване на матрица, или набор от нормално разпределени случайни числа и масиви
randperm	Създаване на вектор (1-по-N матрица), съдържащ произволна промяна на определените цели числа.

Повечето от тези функции връщат матрици от тип Double (двойна точност,плаваща запетая). Въпреки това, лесно може да се изградят основни масиви на всеки цифров тип, като се използват единици, нули, и разгледаните функции.

За целта се определя клас MATLAB име като последен аргумент:

```
A = нули (4, 6)
A =     0     0     0     0     0     0
0       0     0     0     0     0
        0     0     0     0     0     0
        0     0     0     0     0     0
```

Примери:

1)Създаване на матричен магически квадрат. Един магически квадрат е матрица, в която сумата от елементите във всяка колона или ред, или на всеки главен диагонал е един и същ. За да се създаде 5-по-5 матричен магически квадрат , се използва функцията „magic” по следнич начин:

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

Да се има в предвид, че елементите на всеки ред, всяка колона и всеки главен диагонал допринасят за една и съща стойност: 65.

2) Създаване на диагонална матрица. Използва се „DIAG" за създаването на диагонална матрица от вектор. Може да се постави вектора на главния диагонал на матрицата, или на диагонал който е над или под главния:

„B -1" се въвежда местоположението на вектора един ред под главния диагонал:

```
A =   [12  62  93  -8  22];
B =   diag (A, -1)
B =   0     0     0     0     0     0
12    0     0     0     0     0
0     62    0     0     0     0
0     0     93    0     0     0
0     0     0     -8    0     0
0     0     0     0     22    0
```

4. Съединяване на матрицата

Свързването на матрици е процесът на присъединяване на една или повече матрици за да се направи нова матрица. Операторът [ ] в този раздел служи не само като конструктор, но и като свързващ MATLAB оператора. Изразът C = [A B] хоризонтално свързва матрици A и B. Изразът C = [A; B] ги свързва вертикално .

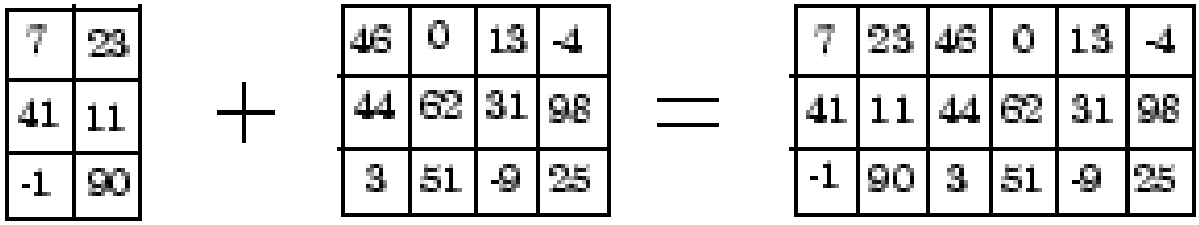
Този пример изгражда нова матрица C, чрез свързването на матрици A и B във вертикална посока:

```
A = първите (2, 5) * 6;          2%-от-5 матрицата  6
B = границата (3, 5);            3%-от-5 матрицата от случайни числа
C = [A; B]                       % Вертикално свързване на A и B
C =   6.0000   6.0000   6.0000   6.0000   6.0000
6.0000   6.0000   6.0000   6.0000   6.0000
0.9501   0.4860   0.4565   0.4447   0.9218
0.2311   0.8913   0.0185   0.6154   0.7382
0.6068   0.7621   0.8214   0.7919   0.1763
```

Поддържане на правоъгълна матрица.

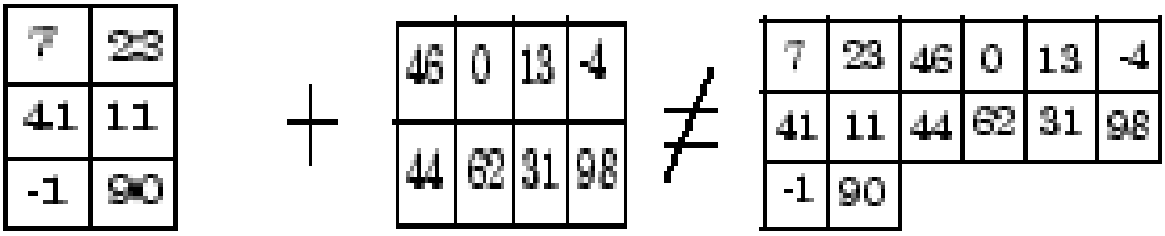
Може да се изградят матрици, или дори многомерни масиви, като се използва свързването, докато получената матрица няма неправилна форма (както е във втория пример показани по-долу). Ако матрицата е изграждана хоризонтално, тогава всеки от компонентите на матрицата трябва да има същия брой редове. При вертикалното изграждане , всеки компонент трябва да има еднакъв брой колони.

Тази диаграма показва две матрици на една и съща височина (т.е. един и същ брой редове), които са комбинирани хоризонтално за да сформират нова матрица.



3 - от - 2 3 - от - 4 3 - от - 6

Следващата диаграма илюстрира опит за хоризонтално комбиниране на две матрици с различна височина. MATLAB не позволява това.



5.Свързващи функции на матрицата.

Следните функции съчетават съществуващите матрици за сформирането на нова матрица.

Функция	Описание
cat	Свързване на матрици по посоченият размер
horzcat	Хоризонтално свързване на матрици
vertcat	Вертикално свързване на матрици
repmat	Създаване на нова матрица, чрез копиране на съществуващи и обработени матрици
blkdiag	Създаване на блок диагонална матрица от съществуващите матрици

Примери: За това как може да се използват тези функции:

1)Свързване на матрици и масиви. Една алтернатива на използването на []оператор за свързването са трите функции Cat, horzcat и vertcat. С тези функции може да се изградят матрици (или многомерни масиви по определен размер. Всяка една от следните команди изпълнява същата задача като командата:

```
C = [A; B]
C = cat(1, A, B);          %Свързване през първото измерение
C = vertcat(A, B);         %Вертикално свързване
```

2) Пресъздаване на матрица. Използвайки функцията repmat се създава съставна матрица,копие на съществуваща матрица. Когато се въведат repmat(M, v, h):

MATLAB възпроизвежда въведената матрица M „v" пъти вертикално и „h" пъти хоризонтално: Например, за да се възпроизведе съществуващата матрица „A" в нова матрица „B", се използва:

```
A = [8  1  6; 3  5  7; 4  9  2]
A =     8     1     6
        3     5     7
        4     9     2
B = repmat(A, 2, 4)
B =     8     1     6     8     1     6     8     1     6     8     1     6
        3     5     7     3     5     7     3     5     7     3     5     7
        4     9     2     4     9     2     4     9     2     4     9     2
        8     1     6     8     1     6     8     1     6     8     1     6
        3     5     7     3     5     7     3     5     7     3     5     7
        4     9     2     4     9     2     4     9     2     4     9     2
```

3)Създаване на Блок диагонална матрица: „blkdiag" функцията съчетава матрици в диагонална посока, създавайки това, което се нарича блок диагонал матрица M. Всички други елементи на новосъздадената матрица M са нулеви:

```
A = magic(3);
B = [-5 -6 -9; -4 -4 -2];
C = eye(2)*8;
D = blkdiag(A, B, C)
```

D = 8 1 6 0 0 0 0 0

3 5 7 0 0 0 0 0

4 9 2 0 0 0 0 0

0 0 0 -5 -6 -9 0 0

0 0 0 -4 -4 -2 0 0

0 0 0 0 0 0 8 0

0 0 0 0 0 0 0 8

6. Изготвяне на цифрова последователност.

Тъй като цифровите последователности често могат да бъдат полезни в изграждането и индексирането на матрици и масиви, MATLAB предвижда специален оператор за подпомагане на създаването им:

Colon Оператор.

Colon операторът (първо: последно) генерира 1-по-N матрицата (или вектора) на последователни числа от първата стойност до последната. По подразбиране е последователността съставена от единични стойности, всяка една по-голяма от предишната:

```
A = 10:15
A =    10    11    12    13    14    15
```

Цифровата последователност не трябва да се състои от цели положителни числа. Тя може да включват отрицателните числа и дробни числа, както и:

```
A = -2.5:2.5
A =   -2.5000   -1.5000   -0.5000    0.5000    1.5000    2.5000
```

По подразбиране, MATLAB винаги нараства с точно 1 при създаването на последователност, дори ако крайната стойност е с неинтегрирана дистанция от самото начало:

```
A = 1:6:3
A =    1    2    3    4    5    6
```

Също така, по подразбиране сериите, генерирани от colon оператора винаги се увеличават вместо да намаляват. В операцията показана в този пример се прави опит за нарастване от 9 към 1 и по този начин MATLAB връща празна матрица:

```
A = 9:1
A =
Празна матрица: 1-с-0
```

Следващият раздел обяснява как да се генерират числови редове, които не са основни (не са по подразбиране).

Използване на Colon оператор със синхронизирана стойност:

За да се генерира поредица, която не използва по подразбиране увеличаването с 1, се посочва допълнителна стойност с „*colon"* оператора (първо:стъпка:последно) . Между началната и последната стойност е стъпка стойност, която казва на MATLAB колко да добави (или да намали , ако стъпката е отрицателна) между всеки номер който генерира.

За да се генерира поредица от номера 10-50, увеличавайки се с 5, се използва

```
A = 10:5:50
A =     10     15     20     25     30     35     40     45     50
```

Нарастването може да се увеличава със стойности, които не са цели числа. В този пример нарастването е с 0.2:

```
A = 3:0.2:3.8
A =3.0000    3.2000    3.4000    3.6000    3.8000
```

За да се създаде последователност със намаляващ интервал се посочва отрицателна синхронизирана стойност:

```
A = 9:-1:1
A =     9     8     7     6     5     4     3     2     1
```

Раздел 2: Индексиране на матрицата.

1. Достъп до отделните елементи.

За връзка с конкретен елемент в една матрица, се определят неговите номера редове и колони, като се използва следния синтаксис, където A е матрица променлива. Винаги се определя първи ред и втора колона:

```
A ( ред, колона )
```

Например, за 4 по 4 магически квадрат A,

```
A = magic(4)
A =     16     2     3     13
```

5 11 10 8

9 7 6 12

4 14 15 1

За осъществяване на достъп до елемент от ред 4, колона 2: A(4, 2) ; ans =14

За масиви с повече от две измерения, се определят допълнителни индекси последвани от ред и колона индекси.

2. Линейно индексиране.

Елементите на MATLAB матрица с един индекс, A (к) се отнасят до линеиното индексиране. MATLAB съхранява матрици и масиви не под формата под която фигурират когато се показват в MATLAB командния прозорец, а като една колона от елементи. Тази една колона се състои от всички колони от матрицата, всяка, приложена към последната:

И така, матрицата A:

A = [2 6 9 ; 4 2 8 ; 3 5 1]

A = 2 6 9

4 2 8

3 5 1

Мтрицата „A" се съхранява в паметта на последователността (поредицата; цикъла):

2, 4, 3, 6, 2, 5, 9, 8, 1

Елементите на ред 3, колона 2 на матрицата A (стойност = 5) също могат да бъдат идентифицирани като елемент 6 в действителната последователност на съхранение. За достъп до този елемент, е нужна възможност за избор за използване на образец A (3,2) синтаксис, или да се използва A (6), което е по линейно индексиране.

Ако трябва да се представи с указателни знаци, MATLAB изчислява индекс в склад колона на базата на размерите, който възлага на масив. Например,поеме двумерен масив като A е размер [d1 d2], където d1 е броя на редовете в масива, а d2 е броя на колоните. Ако доставените два указателни знака (I, J), представляващи ред-колона индекси, компенсирането е: (j-1) * d1 + i

Като се има предвид израз A (3,2), MATLAB изчислява компенсирането на склад A колона (2-1) * 3 + 3, или 6. Броят се шест елемента в колоната с достъп до стойност 5.

4.Функции , които контролират Индексирането

Ако има ред-колона указателни знаци, но трябва да се използва линейно индексиране вместо това , може да се конвертира към него, използвайки функцията *„sub2ind"* функция.

В 3-по-3 матрицата A, използвана в предишния раздел, „*sub2ind”* променя стандартния индекс ред-колона на (3,2), към линеен индекс на 6:

A = [2 6 9; 4 2 8; 3 5 1];

linearindex = sub2ind(size(A), 3, 2) ..... , следователно : linearindex =6

За да се получи ред-колона еквивалентен на линеен индекс, се използва функцията „*ind2sub*” :

[row col] = ind2sub(size(A), 6) ,

където : row =3 ; col =2 .

4. Достъп до множество елементи

За 4-по-4 матрицата A, представена по-долу, е възможно да се изчисли сумата на елементи в четвъртата колона на A, като се въвеждат:

A = magic(4);

A(1,4) + A(2,4) + A(3,4) + A(4,4)

Може да се намали размера на този израз използвайки „*colon*” оператора. Долните индексирани изрази, включващи колони се отнасят до части от една матрица. Като такива се считат:

A(1:m, n)

Това се отнася до елементи в редове от 1 до *m* от колона *n* на матрицата A. Използвайки тази бройна система, може да се изчисли сумата на четвъртата колона от A по - кратко:

sum(A(1:4, 4))

<u>Непоследователни елементи :</u> За да се отнасят до непоследователните елементи в една матрица, се използва оператор *colon* със синхронизирана стойност *m:3:n* - в този израз означава да се направи заданието към всеки трети елемент в матрицата. Трябва да се има в предвид, че този пример използва линейно индексиране:

B = A;

B(1:3:16) = -10

B = -10 2 3 -10

5 11 -10 8

9 -10 6 12

-10 14 15 -10

MATLAB поддържа тип масивно индексиране, който използва един масив като индекс в друг масив. Може да се базира този тип индексиране от двете стойности или позициите на елементите в индексирания масив .

Ето пример на базирана индексирана стойност , където масив B индексира в елементите 1, 3, 6, 7, и 10 от масив A. В този случай числовите стойности на масив B определят предназначените елементи на A:

```
A = 5:5:50
A =     5     10     15     20     25     30     35     40     45     50
B = [1 3 6 7 10];
A(B)
ans =   5     15     30     35     50
```

Ако имате индекс на вектор с друг вектор, ориентацията на индексирания вектор е удостоен за изход: A(B')

```
ans =   5     15     30     35     50
A1 = A';      A1(B)
ans =   5
15
30
35
```

Ако има индекс на вектор с не вектор, форматът на индексите е удостоен.:

```
C = [1 3 6; 7 9 10];
```

A(C) , следователно ; ans = 5 15 30

35 45 50

Ключовата дума *end*: MATLAB предвижда ключова дума за обозначаване на последния елемент в специално измерение на масив. Тази ключова дума може да бъде полезна в случаите, където програма не знае колко редове или колони са в матрицата. Може да се замени изразът в предишния пример със:

```
B(1:3:end) = -10
```

Забележка:Ключовата дума *end* има няколко значения в MATLAB. Може да се използва като обясненото по-горе, или да прекрати условен блок от код , като например, *if* и *for* блокове, или да прекрати една вложена функция.

Определяне на всички елементи от ред или колона: Colon от само себе си се отнася до всички елементи в един ред или колона на матрица. С помощта на следния синтаксис, може да се изчисли сумата на всички елементи на втората колона от 4 по 4

магически квадрат A:sum(A(:, 2)) , следователно : ans =34.Използвайки colon със линейно индексиране, можете да отнася до всички елементи на цялата матрицата. Този пример показва всички елементи на матрицата A, връщайки ги в една последователна колона:

A(:) ... , следователно : ans = 16

5

9

4

.

.

12

1

5.Използване на логически (logical) индексиран масив.

Логическият индексиран масив определя елементите на масива на база на техните позиции в индексирания масив, B, а не тяхната стойност. В този тип маскиране на операция, всеки истински елемент в индексирания масив се разглежда като позиционен индекс в използвания(достъпния) масив.

В следващия пример, B е матрица на логически единици и нули. Позицията на тези елементи в B определя кои елементи на A се определят от изразът A (B):

A = [1 2 3; 4 5 6; 7 8 9]

A = 1 2 3

4 5 6

7 8 9

B = logical([0 1 0; 1 0 1; 0 0 1]);

B = 0 1 0

1 0 1

0 0 1

A(B)

ans = 4

2

6

9

Find функцията може да бъде полезна при логически масиви, тъй като връща линейните индекси от ненулеви елементи B, и по този начин помага да се тълкува A (B):

find(B)

```
ans =   2
4
8
9
```

Този пример създава (logical)логически масив B ,отговаря на изискванията A > 0.5, и използва позициите на тези в B към индекса на A:

```
A = rand(5);
B = A > 0.5;
A(B) = 0
A =         0    0.0975    0.1576    0.1419         0
         0    0.2785         0    0.4218    0.0357
    0.1270         0         0         0         0
         0         0    0.4854         0         0
         0         0         0         0         0
```

По-лесен начин да се изрази това е:

```
A(A > 0.5) = 0
```

Логическо индексиране - Пример 2

Следващият пример подчертава мястото на простите числа в магическия квадрат използвайки логическо индексиране за да се зададат(настроят) числата които не са прости за 0:

```
A = magic(4)
A =    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
B = isprime(A)
B =     0     1     1     1
     1     1     0     0
     0     1     0     0
     0     0     0     0
A(~B) = 0;                          %Логическо индексиран
A =     0     2     3    13
     5    11     0     0
     0     7     0     0
```

```
0     0     0     0
find(B)
ans =   2
5
6
7
9
13
```

Логическо индексиране с по-малък индексиран масив

В повечето случаи логически индексирания масив трябва да има същия брой елементи като масива,в който е индексиран, но това не е изискване. Индексирания масив може да има по-малки размери:

```
A = [1 2 3;4 5 6;7 8 9]
A =     1     2     3
4     5     6
7     8     9
B = logical([0 1 0; 1 0 1])
B =     0     1     0
1     0     1
isequal(numel(A), numel(B))
ans =0
A(B)

ans =   4
7
8
```

MATLAB третира липсващите елементи на индексирания масив както ако те са били настоящи и свеждани до нула, така и в масива B по-долу:

```
%Добавени нули към индексирания масив C  за да му се даде същия брой от
%елементи като A.
C = logical([B(:);0;0;0]);

isequal(numel(A), numel(C))
ans =1
A(C)
```

```
ans =   4
7
8
```

6. Single Colon индексиране с различни видове индексиран масив:

Когато се индексира в стандартен MATLAB масив който използва Single Colon, MATLAB връща колона вектор. Когато се индексира в структура или cell масив посредством Single Colon colon ,се получава списък разделен със запетая;

Създаване на три вида масиви:

```
n = [1 2 3; 4 5 6];
c = {1 2; 3 4};
s = cell2struct(c,{'a', 'b'}, 1); s(:,2)=s(:,1);
```

Използвайки Single Colon за всяка:

```
n(:)            c{:}            s(:).a
ans =           ans =           ans =
1                       1               1
4               ans=            ans =
2                       3               2
5               ans=            ans =
3                       2               1
6               ans=            ans =
4               2
```

7. Индексиране за прехвърляне:

При прехвърляне на стойности от една матрица в друга матрица, може да се използват някои от типовете на индексирания масив, обхванати в този раздел.Матричното прехвърляне отчита и следните изисквания.

В прехвърлянето A (J, K,...) = B (M, N,...), указателни знаци J, K, M, N и т.н. могат да са скаларни, векторни, или масиви, при условие че всички изброени са верни:

Броят указателни знаци определени за B, с изключение на остатъчните указателни знаци равни на 1, не надвишава ndims (B).

Броят на не скаларни указателни знаци определени за A е равен на броя на не скаларни указателни знаци определени за B. Например, A (5, 1:4, 1, 2) = B, (5:8) е валидно, защото и двете страни на уравнението използват един не скаларен указателен знак.

Редът и дължина на всички които не са скаларни указателни знаци определени за A съответства на реда и дължината на които не са скаларен указателни знаци определени за B. Например, A (1:4, 3, 3:9) = B (05:08, 01:07), е валидна, защото и двете страни в уравнение (без този, скаларен указателен знак 3) да се използва 4-указателен знак , последван от 7-указателен знак.

<u>Раздел 3:</u> Информация за Матрицата

1. Размери на Матрицата

Тези функции връщат информация за формата и размера на матрицата.

Функция	Описание
Length	**Връща дължината на най-дългата страна. (Дължината на една матрица,или масив с нулев размер е нула .)**
Ndims	**Връща броя на размерите.**
Numel	**Връша броя на елементите**

Size	**Връща дължината на всеки размер**

Следните примери показват няколко лесни начина да се използват тези функции. Двете използват 3-по-5 матрица A показана тук:

```
A = 10*gallery('uniformdata',[5],0);
A(4:5, :) = []
A =     9.5013    7.6210    6.1543    4.0571    0.5789
2.3114    4.5647    7.9194    9.3547    3.5287
6.0684    0.1850    9.2181    9.1690    8.1317
```

Пример , използващ numel

С помощта на функцията **numel** се намира средната стойност на всички стойности в матрца A.

```
sum(A(:))/numel(A)
ans =  5.8909
```

Пример , използващ ndims, numel, и size

Използването на **ndims, numel, и size** се проверява матрицата и се намират онези стойности които са между 5 и 7, включително:

```
if ndims(A) ~= 2
  return
end
[rows cols] = size(A);
За  m = 1:rows
    for n = 1:cols
        x = A(m, n);
     if x >= 5 && x <= 7
        disp(sprintf('A(%d, %d) = %5.2f', m, n, A(m,n)))
      end
   end
end
```

Кодът връща следното:

```
A(1, 3) = 6.15
A(3, 1) = 6.07
```

2.Класове, използвани в Матрицата

Тези функции тестват елементите на матрицата за определен тип данни.

Функция	**Описание**
Islogical	Определя дали входа е логически масив.
Isnumeric	Определя дали входа е числов масив.
Isreal	Определя дали входа е масив от реални числа
Isstruct	Определя дали входа е MATLAB структурен масив

3.Структури от данни, използвани в Матрицата

Тези функции тестват елементи на матрица за конкретна структура от данни.

Функция	Описание
Isempty	Определяне на това дали въвеждането има всички измерение с размер нула
Isscalar	Определяне на това дали въвеждането е 1-по-1 матрица.
Issparse	Определяне на това дали въвеждането е разредени матрица.
Isvector	Определяне на това дали въвеждането е 1-по-n или n-по-1 матрица.

Разадел 4: *Преоразмеряване и преустройството на матрици*

1. Увеличаване на размера на матрицата

Можете да разширите размера на всички съществуващи матрици, докато това не даде резултат на матрица в неправилна форма. Например, може да се комбинира вертикално

4-по-3 матрица и 7-по-3 матрица, тъй като всички редове от тази матрица имат същия брой колони (3).

Двата начина за разширяване на размера на съществуваща матрица са:

• Сбор от нови елементи на матрицата

• Съхраняване на място, и извън пределите на матрицата

Забележка: Ако трябва да се разшири размера на една матрица многократно с течение на времето, тъй като изисква повече място (обикновено се прави в програмен цикъл), е препоръчително предварително да се разпредели пространство за матрицата, когато първоначално се създадава.

2. Намаляване на размера на матрицата

Матриците дават възможност да се изтриват редове и колони от тях, чрез възлагане на празен масив [] към тези редове или колони. Започва се с :

```
A = magic(4)
A =     16      2      3     13
          5     11     10      8
          9      7      6     12
          4     14     15      1
```

След това се изтрива втората колона от A използвайки :

A(:, 2) = [] .Това променя матрица A в:

```
A =     16     3     13
5     10      8
9      6     12
4     15      1
```

Ако изтриете един елемент от една матрица, резултатът вече не е матрица.Така че изрази като: A(1,2) = [] , водят до грешка. Въпреки това, може да се използва линейно индексиране, за да се изтрие един елемент, или поредица от елементи. Това оформя останалите елементи

във вектор-ред:

A(2:2:10) = []

Резултата:

A = 16 9 3 6 13 12 1

3.Промяна на матрица

Следните функции променят формата на матрица.

Функция	Описание
Reshape	Промяна на формата на матрица.
rot90	Завъртане на матрицата на 90 градуса.
Fliplr	Обръщане на матрицата по отношение на вертикалната ос.
Flipud	Обръщане на матрица спрямо хоризонтална ос.
Flipdim	Обръщане на матрица по определената посока.
Transpose	Обръщане на матрица около основния си диагонал, превръщайки ред вектори във колона вектори и обратно.
Ctranspose	Транспониране на матрица и заменяне на всеки елемент с комплекс съединения.

Примери

Ето някои примери, които да илюстрират някои от начините, с които можете да променяте матрици.

Промяна на матрица.

Променяне на 3-на-4 матрица с размери A към 2-на-6:

A = [1 4 7 10 ; 2 5 8 11 ; 3 6 9 12]

A = 1 4 7 10

2 5 8 11

3 6 9 12

B = променяте (A, 2, 6)

B = 1 3 5 7 9 11

2 4 6 8 10 12

Транспониране на матрица.

Транспортираме A за да станат редът елементи колони. Можете да използвате функцията transpose или transpose оператора (.') to do this:

```
B = A.'
B =                                                    1     2     3
4     5     6
7     8     9
10    11    12
```

Има отделна функция, наречена *ctranspose* която изпълнява комплекс спрегнато транспониране на матрица. Равностойността на оператор за *ctranpose* на матрица A е A ":

```
A = [1+9i   2-8i   3+7i;    4-6i   5+5i   6-4i]
A =                                                    1.0000 + 9.0000i       2.0000 -8.0000i
3.0000 + 7.0000i
4.0000 -6.0000i       5.0000 + 5.0000i      6.0000 -4.0000i
B = A'
B =1.0000    -9.0000i       4.0000  +  6.0000i
2.0000   +  8.0000i      5.0000   -5.0000i
3.0000    -7.0000i       6.0000  +  4.0000i
```

Завъртане на Матрицата. Завъртане на матрицата на 90 градуса:

```
B = rot90(A)
B =     10        11        12
 7          8         9
 4          5         6
 1          2         3
```

Обръщане на матрица. Обръщане на A в ляво към дясна посока:

```
B = fliplr(A)
B =     10    7     4     1
11    8     5     2
12    9     6     3
```

Раздел 5: *Прехвърляне и сортиране на матрици*

1. Прехвърляне и функции за сортиране

Използвайте тези функции за прехвърляне или сортиране на елементите от матрицата.

Функция	**Описание**
circshift	Кръгова смяна на съдържанието на матрицата
sort	Сортиране на масив елементи във възходящ или низходящ ред.
sortrows	Сортиране на редовете във възходящ ред.
issorted	Определяне на това дали елементите на матрицата са сортирани по ред.

Можете да сортирате матрици, многомерни масиви и клетка масиви от поредица , от което и да е измерение и във възходящ или низходящ ред на елементите. Функцията sort връща допълнителен набор от индекси, показващи реда в който елементите са пренаредени по време на операцията сортиране.

2. Прехвърляне на мястото на елементите на матрицата.

Circshift функцията сменя елементите от матрица по кръгов начин през хода на едно или повече измерения. Редове или колони които са прехвърлени от матрица се движат обратно към противоположния край. Например, промяната на 4-по-7 матрица с едно място в ляво движи елементите в колони от 2 до 7 към колони от 1 до 6, и движи колона 1 към колона 7.

Създаите 5-по-8 матрица A и я прехвърлете от дясно през второто(хоризонтално)измерение с три,места.(Можете да използвате

[0, -3] за да прехвърлите в ляво с три места).

```
A = [1:8;   11:18;    21:28;    31:38;    41:48]
A =     1     2     3     4     5     6     7     8
11     12    13    14    15    16    17    18
21     22    23    24    25    26    27    28
31     32    33    34    35    36    37    38
41     42    43    44    45    46    47    48
B = circshift(A, [0, 3])
B =     6     7     8     1     2     3     4     5
16     17    18    11    12    13    14    15
26     27    28    21    22    23    24    25
36     37    38    31    32    33    34    35
```

46 47 48 41 42 43 44 45

Вземаме A и я прехвърляме през две величини: три колони вдясно и два реда по – нагоре:

```
A = [1:8; 11:18; 21:28; 31:38; 41:48];
B = circshift(A, [-2, 3])
B =  26  27  28  21  22  23  24  25
     36  37  38  31  32  33  34  35
     46  47  48  41  42  43  44  45
      6   7   8   1   2   3   4   5
     16  17  18  11  12  13  14  15
```

Тъй като circshift разпространява изместените редове и колони навсякъде до другия край на матрицата, прехвърлянето на точния размер от A връща всички редове и колони до първоначалните им местоположение:

```
B = circshift(A, size(A));
all(B(:) == A(:))        %Всички елементи от B съответстват ли на A?
ans =
1                        % Да
```

3.Сортиране на данните във всеки ред и колона

<u>Сортиране на данните във всяка колона.</u>Функцията *sort* сортира елементите на матрицата през определена(установена)величина.Синтаксисът за тази функция е:

sort(matrix, dimension)

За да сортирате колоните на матрицата задайте 1 като величина аргумент. За да сортирате по реда, определете като величина, 2. Този пример прави 6-по-7 произволен матрица тест:

```
A=floor(gallery('uniformdata',[6 7],0)*100)
A =  95  45  92  41  13   1  84
     23   1  73  89  20  74  52
     60  82  17   5  19  44  20
     48  44  40  35  60  93  67
     89  61  93  81  27  46  83
     76  79  91   0  19  41   1
```

Сортиране всяка колона на A във възходящ ред:

```
c = sort(A,  1)
c =  23   1  17   0  13   1   1
```

48 44 40 5 19 41 20

60 45 73 35 19 44 52

76 61 91 41 20 46 67

89 79 92 81 27 74 83

95 82 93 89 60 93 84

issorted(c(:, 1))

ans =1

Сортиране на данните във всеки ред.Използвайте issorted да сортирате данните във всеки ред. Използвайки горния пример, ако сортирате всеки ред на A в низходящ ред, issorted проверява за възходящ ред. Можете да обърнете вектора за проверка за сортиран низходящ ред:

A=floor(gallery('uniformdata',[6 7],0)*100);

r = sort(A, 2, 'descend')

r = 95 92 84 45 41 13 1

89 74 73 52 23 20 1

82 60 44 20 19 17 5

93 67 60 48 44 40 35

93 89 83 81 61 46 27

91 79 76 41 19 1 0

issorted(fliplr(r(1, :)))

ans =1

Когато зададете втори изход, *sort* връща индексите на оригиналната матрица A разположени в реда, те се появяват в изходната матрица. В следващия пример, вторият ред на индекса съдържа поредица 4 3 2 5 1, която означава, че сортираните елементите в изходната матрица R са взети от A (2,4), A (2,3), A (2,2), A (2,5) и A (2,1):

[r index] = sort(A, 2, 'descend');

index

index =1 3 7 2 4 5 6

4 6 3 7 1 5 2

2 1 6 7 5 3 4

6 7 5 1 2 3 4

3 1 7 4 2 6 5

3 2 1 6 5 7 4

4.Сортиране на Ред Вектори

Sortrows функцията запазва елементите на всеки ред в първоначалната им последователност, но сортира целия ред от вектори съгласно реда на елементите в определения колона.

Следващият пример създава произволна матрица A:

```
A=floor(gallery('uniformdata',[6 7],0)*100);
```

```
A =    95    45    92    41    13     1    84
       23     1    73    89    20    74    52
       60    82    17     5    19    44    20
       48    44    40    35    60    93    67
       89    61    93    81    27    46    83
       76    79    91     0    19    41     1
```

За да сортирате във възходящ ред, основан на стойностите в колона 1, можете да извикате функцията sortrows само с въвеждане на един аргумент:

```
sortrows(A)
```

```
r =    23     1    73    89    20    74    52
       48    44    40    35    60    93    67
       60    82    17     5    19    44    20
       76    79    91     0    19    41     1
       89    61    93    81    27    46    83
       95    45    92    41    13     1    84
```

Да се основе сортирането на колона, различна от първата, повикайте sortrows с втори вход аргумент, който показва номера на колоната, колона 4 в този случай:

```
r = sortrows(A, 4)
```

```
r =    76    79    91     0    19    41     1
       60    82    17     5    19    44    20
       48    44    40    35    60    93    67
       95    45    92    41    13     1    84
       89    61    93    81    27    46    83
       23     1    73    89    20    74    52
```

Раздел 6: *Управляване на диагонални матрици*

1. Функции на диагоналната матрица

Има няколко MATLAB функции които работят специално за диагоналните матрици.

Функция	Описание

Blkdiag	Изграждане на блок диагонална матрица от въведени аргументи.
Diag	Връщане диагонал матрица, или с диагонал на матрицата.
Trace	Изчислява се сумата от елементите по главния диагонал.
Tril	Връщане долната триъгълна част от матрицата.
Triu	Връщане горната триъгълна част от матрицата.

2. Изграждане на матрица от диагонален вектор

„Diag" функцията има две операции, с които може дса изпълнява.Може да се използва за генериране на диагонална матрица:

```
A = diag([12:4:32])
A =    12     0     0     0     0     0
0     16     0     0     0     0
0     0     20     0     0     0
0     0     0     24     0     0
0     0     0     0     28     0
0     0     0     0     0     32
```

Можете да използвате и diag функцията за сканиране на съществуваща матрица и връщане изчислените стойности на един от диагоналите:

```
A = magic(5)
A =    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
diag(A,    2)          % Връщане съдържанието на втория диагонал от A
ans =   1
14
22
```

3. Връщане на триъгълна част от матрица

tril и triu функциите връщат триъгълна част от една матрица, предишното връща част от долния ляв ъгъл и последната от горния десен. По подразбиране, главния диагонал на матрицата разделя тези два сегмента. Можете да използвате алтернативен диагонал чрез определяне на един разстояние от основният диагонал като втори въведен аргумент :

```
A = magic(6);

B = tril(A,      -1)

B =     0      0      0      0      0      0
        3      0      0      0      0      0
       31      9      0      0      0      0
        8     28     33      0      0      0
       30      5     34     12      0      0
        4     36     29     13     18      0
```

4. Свързване на матрици по диагонал

Можете диагонално да съединявате матрици за да образуват една матрица използвайки blkdiag функцията. Вижте "Създаване на Блок диагонална матрица"на страница 1-9 за повече информация как работи.

Раздел 7: *Празни матрици,скаларни величини и вектори.*

1. Общ преглед

Въпреки че матриците са равнинни, те изглежда не винаги имат правоъгълна форма. 1-по-8 матрица, например, има две измерения и все още е линейна. Тези матрици са описани в следните раздели:

„Празни матрици”

Празната матрица има повече от една величина която е равна на 0.Двумерна матрица A на която двете величини са равни на 0 се появява в системата на MATLAB като [].Изразът A = [] възлага на 0-по-0 празна матрица, за A.

„Скаларни величини”

A скаларна е 1-от-1 и се появява в MATLAB като един реален или сложен номер (например, 7, 583.62, -3.51, 5.46097e-14, 83 и 4).

„Вектори”

Векторът е 1-по-N или N-по-1, и се появява в MATLAB като ред или колона от реални или комплексни числа:

Колона Вектори Ред Вектори

53.2 53.2 87.39 4-12i 43.9

87.39

4-12i

43.9

2. Празна матрица

Матрица която има повече от една величина равна на 0 се нарича празна матрица.Най – простата празна матрица е 0 -по -0 ,по отношение на размера.Примери за по – сложни матрици са тези с величини 0-по-5 или 10-по-0.

За да създадете по 0-0-матрица, използвайте квадратен оператор скоба, без посочена стойност:

A = [ ];

Име	Размер	Байтове	Клас
A	0x0	0	двоен масив

Можете да създадете празни матрици (и масиви) от други размери използвайки функциите zeros,ones, rand, или eye .За да създадете 0-по-5 матрица,за пример използвайте

A = zeros(0,5)

<u>Операции с празни матрици</u>

Основният модел за празни матрици е, че всяка операция, която е определена за m-по-n матрици, и доведе до резултат, чийто величина е някаква функция от m и n, все пак трябва да се разреши, когато m или n е равна на нула. Размерът на резултата от тази операция е в съответствие с размера на резултатите, генерирани при работа с не празни стойности, а вместо това се оценява на нула.

Например хоризонтално свързване

C = [A B]

изисква A и B да имат същия брой редове. Така че, ако A е m-по-n и B е m-по-p, след което C е m-по-(n + P). Това все още е вярно, ако m или n и p е равна на нула.

Както при всички матрици в MATLAB, трябва да следвате правилата по отношение на съвместими размери. В следващия пример, се прави опит за добавяне на 1-по-3 матрица към 0-по-3 празна матрица довежда до грешка:

[1 2 3] + ones(0,3)

??? Error using ==> +

Matrix dimensions must agree.(Матричната величина трябва да се съгласи)

<u>Общи операции</u>.Следните операции връщат нула от празен масив

A = [ ];

size(A), length(A), numel(A), any(A), sum(A)

Тези операции връщат стойност,която не е нулева от празен масив:

A = [ ];

ndims(A), isnumeric(A), isreal(A), isfloat(A), isempty(A), ...

all(A), prod(A)

Използването на Празни матрици в релационни операции

Можете да използвате празни матрици в релационни операции като "равен" (==) или "Повече от" (>), докато и двата операнда имат еднакви размери, или непразната операнда е скаларна.Резултатът от която и да е релационна операция означава,че празната матрица е празна матрица. Дори сравняването на празна матрица с равенство към себе си не връща истина, а вместо това предава празна матрица:

```
x = ones(0,3);
y = x;
y == x
ans =
Empty matrix: 0-by-3
```

Използване на празни матрици в логически операции

MATLAB има два различни вида логически оператори:

Short-circuit (& &, | |) - Използва се при тестване на множество логически състояния (например, x> = 50 & & x <100), където всяко условие се оценява до скаларна величина вярна или невярна.

- Element-wise (&, |) - Извършва логическо AND(и), OR(или),или NOT(не) за всеки елемент на една матрица, или масив.

Short-circuit операции. Правилото за операнди, използвани в Short-circuit операции е, че всяка операнда трябва да бъде преработен в логическа скаларна величина стойност. Заради това правило, празни матрици не могат да бъдат използвани в логически операции Short-circuit. Тези операции връщат грешка.

Единственото изключение е в случаите, когато MATLAB може да определи, резултатът от логично твърдение, без да направи оценка на целия израз. Това е вярна за следващите две твърдения, тъй като резултатът от цялите твърдения, са известни като се има предвид само първия член:

```
true || []
ans =1
false && []
```

ans =0

Element-wise операции. За разлика от операторите short-circuit, всички Element-wise операции на празни матрици се считат за валидни, докато размерите на операндите се съгласят, или непразната операнда е скаларна величина.

Element-wise операциите на празни матрици винаги връщат празна матрица:

true | [ ]

ans =[ ]

Забележка: Това поведение е в съответствие с начина, по който MATLAB прави скаларна величина разширена с двоични оператори, в която нескаларната величина операнда определя размера на резултата.

3. Скаларни величини

Всяко индивидуално реално или комплексно число е представено в MATLAB като 1-по-1 матрица наречена скаларна величина стойност:

A = 5;

ndims(A) % Провери броя на размерите в A

ans =2

size(A)%Проверете стойността на размерите на редовете и колоните

ans = 1 1

Използвайте isscalar функцията, за да разберете дали дадена променлива има скаларна величина стойност:

isscalar(A)

ans =1

4. Вектори.

Матрици с величина,равна на едно и друга равна на повече от едно са наричани вектори.Ето пример за цифров вектор:

A = [5.73 2-4i 9/7 25e3 .046 sqrt(32) 8j];

size(A) % Проверете стойността на размерите на редовете и колоните.

ans = 7

Можете да изградите вектор от други носители, при условие че критичните размери са съгласни. Всички компоненти от вектор-ред, трябва да бъдат скаларни или друг ред вектори. По същия начин, всички компоненти от една колона вектор трябва да са скалари или друга колона вектори.

A = [29 43 77 9 21];

B = [0 46 11];

C = [A 5 ones(1,3) B]

C = 29 43 77 9 21 5 1 1 1 0 46 11

Събиране на празна матрица с вектор има не оказва влияние върху векторния резултат. Празната матрица се игнорира в този случай:

A = [5.36; 7.01; []; 9.44]

A = 5.3600

7.0100

9.4400

Използвайте функцията isvector да разберете дали дадена променлива притежава вектор:

isvector(A)

ans = 1

Раздел 8: Пълни и разредени матрици

1. Общ преглед

Не е необичайно да има матрици с голям брой елементи с нулева стойност и тъй като софтуера MATLAB съхранява нули по същия начин, по който съхранява всяка друга числова стойност, тези елементи могат да използват памет ненужно и понякога може да изисква допълнително време за изчисления.

Разредените матрици са начин за по - ефективно съхраняване на данни които имат голям процент от нулеви елементи. Докато пълните матрици вътрешно съхраняват всеки елемент в памет независимо от стойността, разредените матрици съхраняват само не нулеви елементи и техните ред показатели. Използването на разредените матрици може значително да намали размера на паметта, необходима за съхранение на данни.

Всички аритметични, логически, и индексиращи операции, вградени в MATLAB могат да се прилагат за разредени или смеси от разредени и пълни матрици.Операции за разредени матрици връщане на разредените матрици и операции за пълни матрици връщането на пълните матрици.

2. Функции на разредените матрици

Тази таблица показва някои от функциите, най-често използвани при работа с разредените матрици.

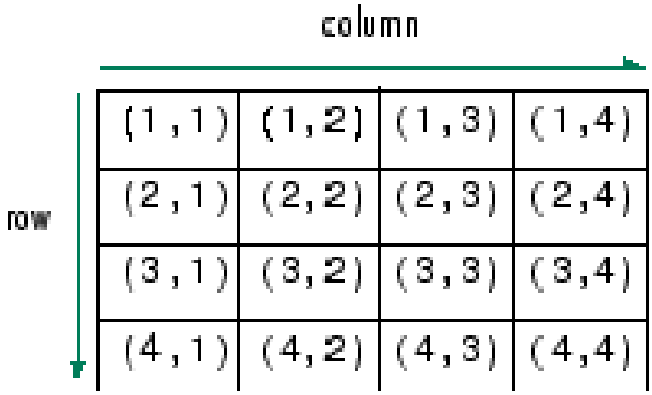
Функция	**Описание**

Full	Конвертиране на разредените матрица към пълни матрица.
Issparse	Определяне дали една матрица е разредена.
Nnz Nonzero Nzmax Spalloc	Връщане броя на нулевите елементи от матрица. Връщане на нулевите елементи от матрицата. Връщане размера на отпуснатите за съхранение не нулеви елементи. Разпределяне на място заразредените матрици.
Sparse Speye Sprand	Създаване на разредени матрици или конвертирането им от пълни в разредени. Създаване на разредени идентични матрици. Създаване на разредени равномерно разпределени случайни матрици.

<u>Раздел 9:</u> Многомерни масиви

1. Общ преглед

Масив с повече от две измерения се нарича многомерен масив в прилагането в MATLAB. Многомерни масиви в MATLAB са продължения на нормалната двуизмерна матрици. Матриците трябва да имат две величини: реда измерение и колоната величина.

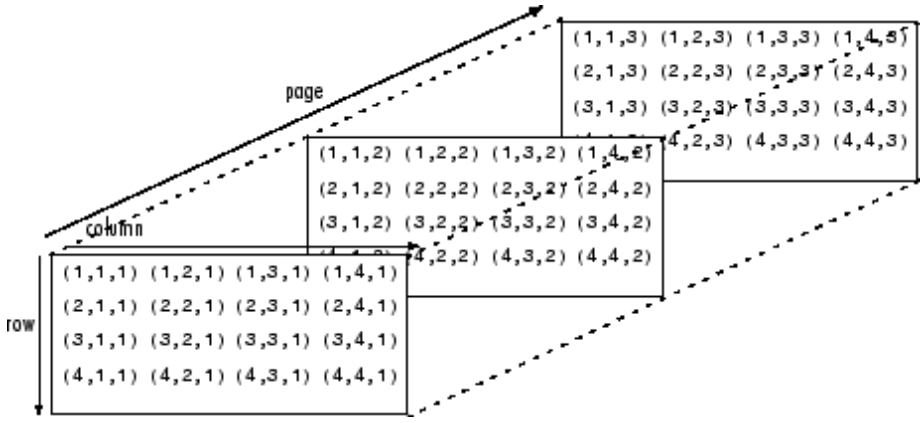


Можете да получите достъп до двумерен матрица елемент с два Указателни знака: първо представляващия ред индекс, и вторият представляващ колоната индекс.

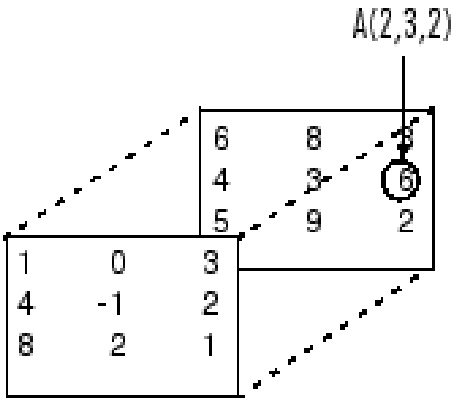
Многомерните масиви използват допълнителни указателни знаци за индексиране. А триизмерна матрица, например, използва три указателни знака:

- Първият масив приписва(обяснява) величина 1, реда.
- Вторият приписва величина 2, колоната.

- Третият приписва величина 3. Този пример използва концепцията за страница, за да представлява величини 3 и по-високи.



За достъп до елемент на втория ред, трета колоната от страница 2, например, използвате указателни знака (2,3,2).




```
A(:,:,1) =

     1     0     3
     4    -1     2
     8     2     1

A(:,:,2) =

     6     8     3
     4     3     6
     5     9     2
```

Когато добавяте към масив, величини, можете да добавите указателни знаци. А четири-мерен масив, например, има четири указателни знака. Първите два обясняват ред-колона двойка, а вторите две достъпа на третите и четвъртите величини на данните.

Повечето от операциите, които можете да извършвате с матрици (т.е. двумерен масиви)могат да се направят също и с многомерни масиви.

Забележка:Функциите на общите многомерни масиви са присъщи на типовете данни директория.

2. Създаване на многомерни масиви

Можете да използвате същите техники за създаване на многомерни масиви,като тези за двумерна матрица. В допълнение, MATLAB предвижда специална съединяваща функция, която е полезна за изграждане на многомерни масиви.

Генериране на масиви използвайки индексиране

Един от начините за създаване на многомерен масив е да се създаде двумерен масив и да сеудължи. Например, започнете с прост двумерен

масива A.

```
A = [5  7 8; 0 1 9; 4 3 6];
```

A е 3-по-3 масив, чийто реда величина е 3 и колона величина е 3. За да добавите трето измерение на A,

```
A(:,:,2) = [1 0 4; 3 5 6; 9 8 7]
```

MATLAB отговаря с

```
A(:,:,1) =      5      7      8
                0      1      9
                4      3      6
A(:,:,2) =      1      0      4
                3      5      6
                9      8      7
```

Можете да продължите да добавяте редове, колони или страници към масива, като използвате подобни задавания на изложение.

Разширяване на Многомерните масиви

За да удължите A във всяко измерение:

• увеличете или добавете подходящите указателни знаци и задайте желаните стойности.

• задайте същия номер от елементи към съответните масивни величини. За числови масиви, всички редове трябва да имат еднакъв брой елементи, всички страници трябва да има еднакъв брой редове и колони, и така нататък.

Можете да се възползвате от възможностите на MATLAB за разширена скаларна величина заедно с оператора colon за да запълни цяла величина с една стойност:

```
A(:,:,3) = 5;
A(:,:,3)
ans =   5      5      5
        5      5      5
        5      5      5
```

За да се превърне в-3 по-3-по-3-по-2, четири-мерен масив, въведете

```
A(:,:,1,2) = [1 2 3; 4 5 6; 7 8 9];
A(:,:,2,2) = [9 8 7; 6 5 4; 3 2 1];
A(:,:,3,2) = [1 0 1; 1 1 0; 0 1 1];
```

Имайте предвид, че след първите две задачи MATLAB наслагва A с нули, като необходимост за поддържане на съответните размери от величини.

Генериране на масиви,използвайки функциите на MATLAB

Можете да използвате MATLAB функции като randn, ones, и zeros за да генерирате многомерни масиви по същия начин, по който ги използвате за двуизмерни масиви. Всеки аргумент който предоставяте показва размера на съответните величини в резултата масив. Например, за да създаде 4-по-3-по-2 масив от нормално разпределени случайни числа:

```
B = randn(4,3,2)
```

За да се генерира масив пълен с една постоянна величина, използвайте repmat функция. repmat възпроизвежда масив (в този случай 1-по-1 масив) чрез вектор от масив измерения.

```
B = repmat(5, [3 4 2])
B(:,:,1) =     5     5     5     5
               5     5     5     5
               5     5     5     5

B(:,:,2) =     5     5     5     5
               5     5     5     5
               5     5     5     5
```

Забележка: Всяко величина на масив може да има размер нула, което го прави вид празен масив. Например, 10-по-по-0-20 е валиден размер за многомерен масив.

Изграждане на многомерни масиви с *cat* функция

Cat функцията е прост начин за изграждане на многомерни масиви, тя свързва списък от масиви по определена величина:

```
B = cat(dim, A1, A2...)
```

където A1, A2 и т.н. са масивите за свързване, и dim е величина, която свързва масивите.

Например да създадем нов масив с функцията *cat:*

```
B = cat(3, [2 8; 0 5], [1 3; 7 9])
B(:,:,1) =     2     8
               0     5
```

B(:,:,2) = 1 3

7 9

Cat функцията приема всяка комбинация от съществуващи и нови данни.В допълнение на това можете да влагате повиквания към *cat* функцията.Например редовете долу създават четири-мерен масив.

A = cat(3, [9 2; 6 5], [7 1; 8 4])

B = cat(3, [3 5; 0 1], [5 6; 2 1])

D = cat(4, A, B, cat(3, [1 2; 3 4], [4 3;2 1]))

cat автоматично добавя указателни знаци на 1 между величини, ако е необходимо. Например, за да създадете 2-по-2-по-1-по-2 масив, въведете

C = cat(4, [1 2; 4 5], [7 8; 3 2])

В предишния случай, *cat* вмъква толкова лъжливи величини, колкото е необходимо за създаване на четири-мерен масив, чиито последна величина не е лъжлива величина. Ако dim аргумента е 5, предходното заявление би довело до 2-по-2-по-1-по-1-по-2 масива. Това добавя допълнително 1s за индексирано изразяване за масива. За достъп до стойността 8 в четириизмерен случай използвайте

C (1 , 2 , 1 , 2)

Лъжлива индексирана величина

Достъп до свойствата на многомерни масиви

Можете да използвате следните функции на MATLAB за да получите информация за многомерните масивите, които сте създали.

Достъп до свойствата на многомерни масиви

Можете да използвате следните функции на MATLAB за да получите информация за многомерните масивите, които сте създали.

- size – Връща размера на всяка величина на мсива.

size(C)

ans = 2 2 1 2

rows columns dim3 dim4

- ndims – Връща броя на величините в масива.

ndims(C)

ans 4

- whos - Предоставя информация за формата и съхранението на масива.

whos

Name	Size	Bytes	Class
A	2x2x2	64	double array
B	2x2x2	64	double array
C	4-D	64	double array
D	4-D	192	double array

Всичко общо е 48 елемента,използващи 384 байта.

3. Индексиране на многомерните масиви.

Много от понятията, които се прилагат за двуизмерна матрица обхващат многомерните масивите достатъчно добре.

За достъп до един елемент от многомерен масив, се използват целочислени указателни знаци.Всички указателни знаци индексират величина - първите индексират реда величина,вторите- колоната величина, третите индексират първата страница величина, и така нататък.

Разгледайте 10-по-5-по-3 масив nddata от случайни числа:

```
nddata = fix(8 * randn(10,5,3));
```

За достъп до елемент (3,2) на страница 2 от nddata, например, използвайте nddata (3,2,2).

Можете да използвате вектори като масив указателни знаци. В този случай всеки елемент вектор трябва да бъде валиден указателен знак, които е в границите, определени от величините на масива. За достъп до елементи (2,1), (2,3) и (2,4) на стр. 3 от nddata, използвайте:

```
nddata(2,[1 3 4],3);
```

Colon оператор и индексиране на многомерен масив.

MATLAB colon индексира разширяването на многомерни масиви.Например, за да получите достъп до цялата третата колона на страница 2 от nddata, използвайте nddata (:, 3,2).

Colon оператора е полезен и за достъп до друга подмножества от данни. Например, nddata (2:3,2:3,1) води до 2-по-2 масив, част от данните, на стр. 1 от nddata. Тази матрица се състои от данни в редове 2 и 3, колони 2 и 3, на първата страница на масива.

C olon оператора може да се появи като масив указателен знак от двете страни на едно възложено изявление. Например, за да се създаде 4-по 4 масив от нули:

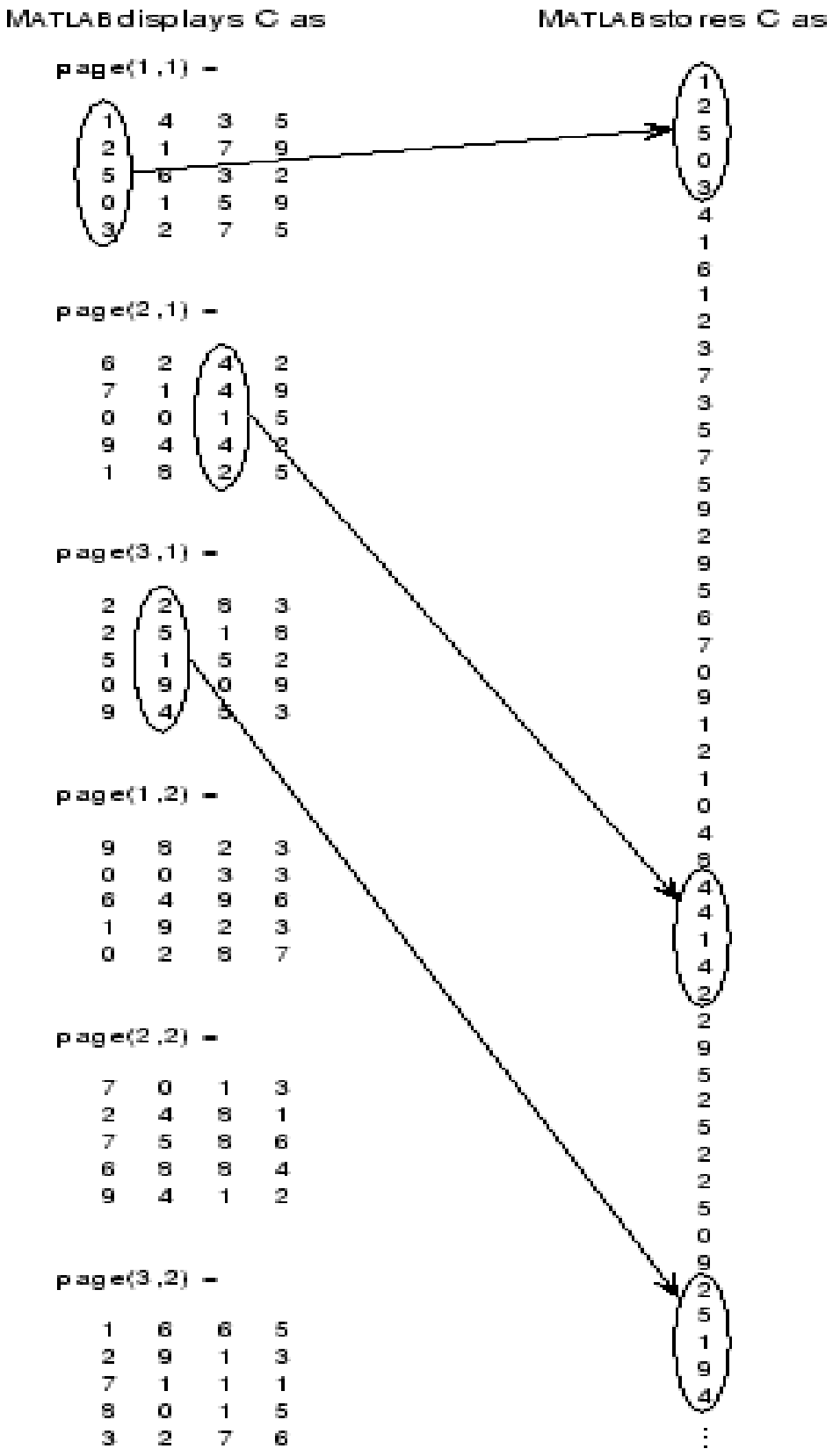
C = zeros(4, 4)

Сега се възлага на 2-по-2 подмножество от масив nddata на четирите елемента в центъра от C.

C(2:3,2:3) = nddata(2:3,1:2,2)

Линейно индексиране с многомерни масиви.

MATLAB линейно индексиране се отнася и за многомерните масивите . В този случай, MATLAB работи страница по страница на база да създаде съхранена колона, отново добавя елементи columnwise. Да разгледаме например 5-по-4-по-3-по-2 масив C.



Отново един указателен знак(долен индекс) индексират директно в тази графа. Например,

C (4) произвежда резултата

ans = 0

Ако зададете два указателни знака (I, J), което показва реда колони, индекси,MATLAB изчислява изместването, описано по-горе. Два указателни знака достигат

до първата страница на многомерения масив, при условие че те са в границите на оригиналните размери на масива.

Ако повече от един указателен знак е налице, всички указателни знаци трябва да съответстват на оригиналните размери масив. Например, C (6,2) е невалиден, тъй като всички страници от C имат само пет реда.

Ако посочите повече от два указателни знака MATLAB съответноразширява индексираната схема. Например за четири указателни знаци (I, J, K, Л) в четири-мерен масив с размер [d1 d2 d3 d4]. MATLAB изчислява разширението в съхранената колона:

(l-1)(d3)(d2)(d1)+(k-1)(d2)(d1)+(j-1)(d1)+i

Например ако индексирате масива C, използвайки указателни знаци (3, 4, 2, 1), MATLAB връща стойност 5 (индекс 38 в съхранената колона).

Като цяло разклонената формула за масив с величини [d_1 d_2 d_3 ... d_n],използвайки указателни знаци (s_1 s_2 s_3 ... s_n) е

$(s_n-1)(_{dn-1})(_{dn-2})...(d_1)+(s_{n-1}-1)(d_{n-2})...(d_1)+...+(s_2-1)(d_1)+s_1$

Благодарение на тази схема, можете да индексирате масив с произволен брой указателни знаци. Можете да добавите произволен брой 1s към списъка указателни знаци(долни индекси), тъй като тези изрази са равни на нула. Например:

C(3,2,1,1,1,1,1,1) е еквивалент на C(3,2)

Избягване на неяснота при Многомерното индексиране

За да разрешите в неопределеността, бъдете сигурни ,че ви е осигурена достатъчно информация относно дестинацията на възложените данни, и че данните и предназначението им е със същата форма. Например:

A(1,:,2) = 1:10;

4. Промяна на многомерни масиви

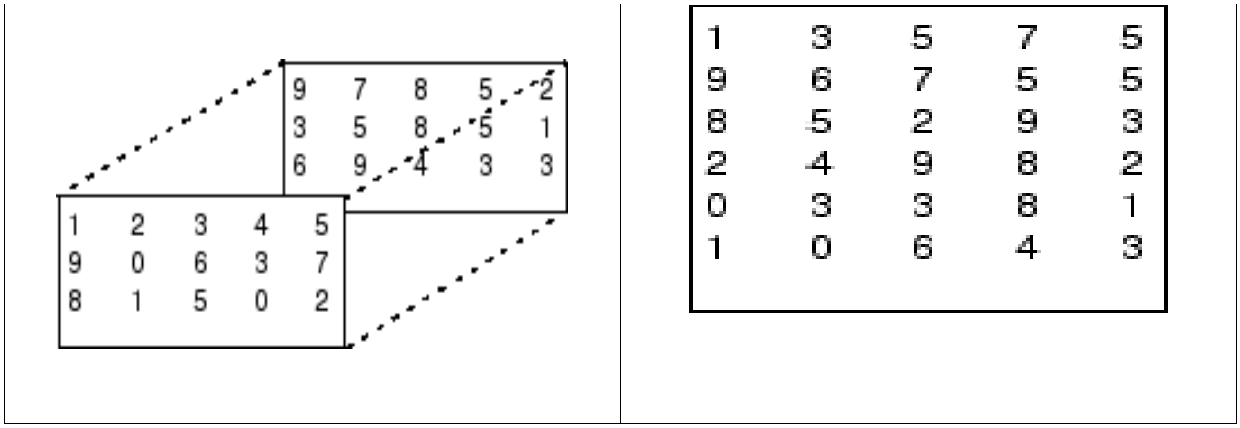
Освен ако не промени своята форма или размер, MATLAB масива запазва величината посочена в създаването му. Вие променяте размера на масива чрез добавяне или изтриване на елементи.

Вие променяте формата на масива като неуточнявате реда на масива е, колоната или страницата на величината като се запазват същите елементи. *reshape* функцията изпълнява последната операция. За многомерните масивите формата е:

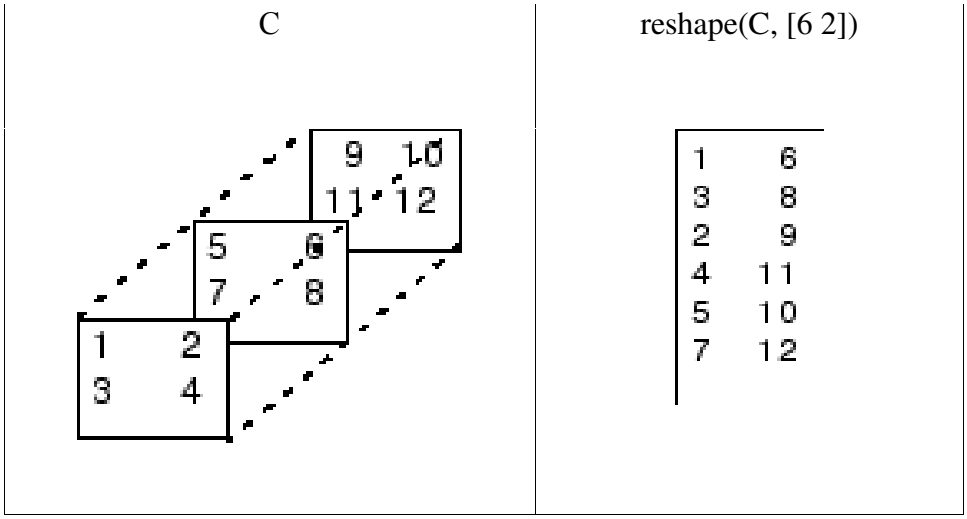
B = reshape(A,[s1 s2 s3 ...])

s_1, s_2 и т.н. представлява желания размер за всяко измерение от променената матрица.

Имайте предвид, че промененият масив трябва да има същия брой елементи като оригиналния масива.



reshape функцията работи по последователен начин.Тя създава променена матрица като взима последователни елементи от всяка една колона от първоначалните построени данни.



Ето и няколко нови масиви от преобразуването nddata:

B = reshape(nddata, [6 25])

C = reshape(nddata, [5 3 10])

D = reshape(nddata, [5 3 2 5])

Премахване на лъжливи величини

MATLAB създава лъжливи величини,ако изрично не се конкретизират когато се създава или преструктурира масив,или ако извършвате изчисление,което води до величина на масив едно:

B = repmat(5, [2 3 1 4]);

size(B)

ans = 2 3 1 4

Squeeze функцията отстранява лъжливите величина на масива:

C = squeeze(B);

size(C)

ans = 2 3 4

Squeeze функцията не засяга двумерните масиви и реда вектори.

5. Разместване величините на масивите

Permute функцията пренарежда размерите на масива.

B = permute(A, dims);

dims е вектор посочващ новия ред за размерите на A, където
1 отговаря на първата величина (редове), 2 отговаря на вторите величини (колони), 3 отговаря за страниците, и така нататък.

За по-подробен поглед върху функцията разместваме, разгледайте масива A,който е с четири измерения от размери 5-по-4-по-3-по-2.Пренареждане на величините,поставяне на колоната величина на първо място,последвана от втората страница величина,първата страница величина ,а след това на реда величина. Резултатът е масив 4-по-2-по-3-по-5.

Преместете величина 2 от A в първата позиция долен индекс от B, величина 4 във втората позиция долен индекс и т.н.

B = permute (A , [2 4 3 1])

Реда на величините в листа с аргументи на permute определя размера и формата на изходящия масив.Например втората величина се премества на първа позиция.Защото втората величина на оригиналния масив е с размер 4,първата величина на изходящия масив също е с размер 4.

Величина	1	2	3	4
Размер	5	4	3	2

Входящ масив A

Величина	1	2	3	4
Размер	4	2	3	5

Изходящ масив B

Можете да мислите за permute операцията като продължение на функцията transpose,която изключва реда и колоната величини от матрицата.За permute редът от входящи списъци величини определя пренареждането на указателни знаци(долни индекси). В горния пример, елемент (4,2,1,2) A става елемент (2,2,1,4) от B, елемент (5,4,3,2) от A става елемент (4,2,3,5) от B и така нататък.

Обратно изменение

Функцията ipermute е обратната на permute. Като се има предвид входящия масив A и вектор с величини v ipermute произвежда масив B така че permute(B,v) връща A. Например тези твърдения създават масив E който е еквивалент на входящия масив C.

```
D = ipermute(C, [1 4 2 3]);
E = permute(D, [1 4 2 3])
```

Можете да получите оригиналния масив,след като го разместите(промените реда му)викайки ipermute със същия вектор от величини.

6. Изчисления с многомерни масиви

Много от изчислителните и математическите функции на MATLAB приемат многомерните масиви като аргументи.Тези величини работят по конкретни величини на многомерни масиви,тоест те работят върху отделни елементи ,вектори или матрици.

Работа с вектори

Функциите, които работят за векторите,като like sum, mean,по подразбиране обикновено работят върху първата нелъжлива величина на многомерния масив. Повечето от тези функции по избор ви позволяват да зададете конкретна величина, върху която да

работят. Има изключения, обаче. Например,cross функцията, която намира пресечният резултат от два вектора, работи по първата нелъжлива величина със степен 3.

Забележка: В много случаи тези функции имат други ограничения за въвеждане на аргументи - например, някои функции, които приемат много масиви изискват масивът да е със същия размер.

Работа елемент по елемент

MATLAB функции, които работят елемент-по-елемент на двумерени масиви, както и тригонометрични и експоненциални функции в elfun директории, работят по абсолютно същия начин, относно многомерните случаи.

Например sin функцията връща масив със същия размер като функцията за въвеждане на аргумент. Всеки елемент на изходния масив е задължително да съответветства(кореспондира) с елемент от входния масив.

По подобен начин, аритметични, логически, и релационни оператори, всички работят със съответните елементи на многомерни масиви, които са със същия размер във всички измерения. Ако един операнд е скаларна величина и един масив, операторът прилага скаларната величина на всеки елемент на масива.

Работа с матрици и равнини

Функции, които работят с равнини или матрици, като линейната алгебра и функциите на матрица в matfun директория, не приемат многомерни масиви като аргументи. Това означава, че не можете да използвате функциите в matfun директория, или масив оператори *, ^, \ или /, със много измерения аргументи. Снабдяване на многомерни аргументи или операнди във тези случаи води до грешка.

Можете да използвате индексиране за приганането на матрична функция или матричен оператор в многомерен масив.Например създайте триизмеренмасив A:

```
A = cat(3, [1 2 3; 9 8 7; 4 6 5], [0 3 2; 8 8 4; 5 3 5], ... [6 4 7; 6 8 5; 5 4 3]);
```

Прилагане на eig функцията за целия многомерен масив води до грешка:

```
eig(A)
??? Error using ==> eig
Input arguments must be 2-D.
```

Въпреки това можете да прилагате eig за равнини в рамките на масива.Например,използвайте colon обозначение за да индексирате само една страница (в този случай) от масива:

```
eig(A(:,:,2))
ans =   12.9129
        -2.6260
```

2.7131

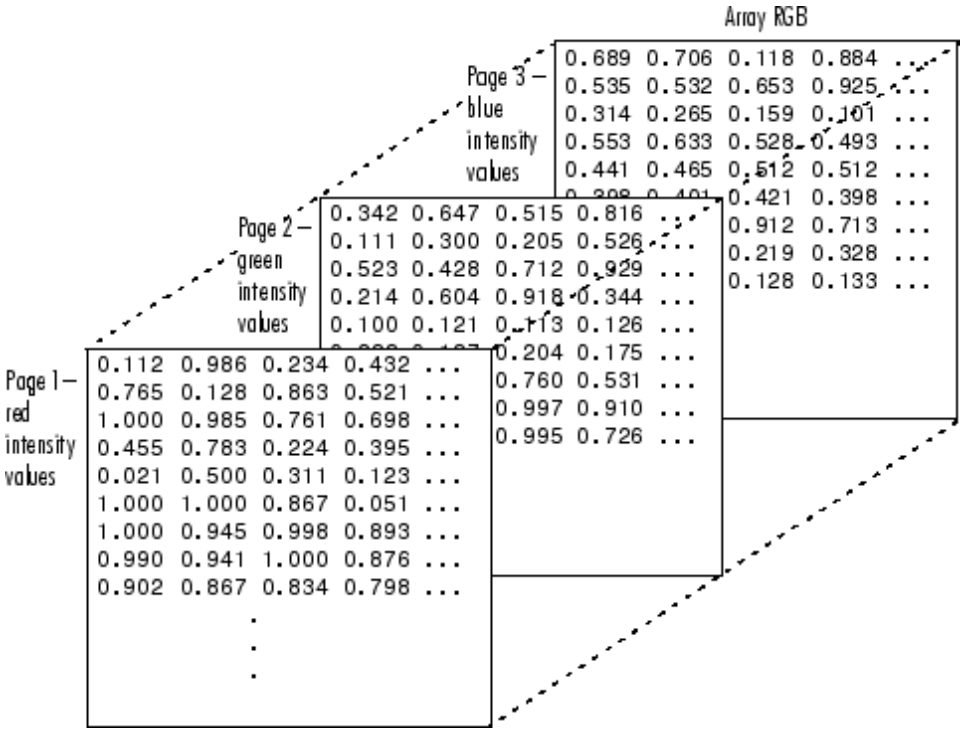
Забележка: В първия случай указателните знаци не са запетаи,трябжа да използвате squeeze за да избегнете грешки. Например, EIG (A (2,:,:)) води до грешка, защото размера на въвеждане е [1 3 3]. Изразът eig(squeeze (A (2,:,:))), обаче одобрява валидна двумерена матрица за eig.

7. Организиране на данни във многомерни масиви

Можете да използвате многомерни масиви за представяне на данните по два начина:

- Като равнини или страници на двумерни данни. След това можете да третира тези страници като матрици.
- Като многовариантни или многомерни данни. Например, може да се имате четиримерен масив, където всеки елемент съответства на температура или за измерване налягането на въздуха, взети от еднакво разпределени точки в една стая.

За пример разгледайте изображение RGB.За снимка с многомерни масиви , може би е най – лесния начин за съхранение и достъп до данни.



За достъп до цялата равнина на изображението, използвайте:

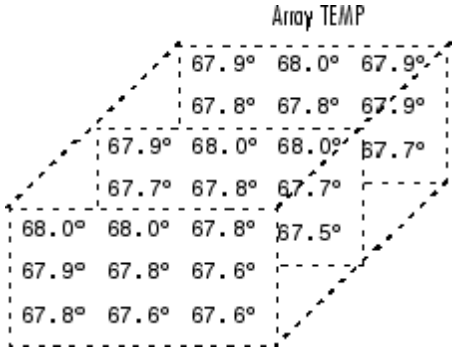
redPlane = RGB (:,:, 1);

За достъп под изображението, използвайте:

subimage = RGB (20:40,50:85,:);

Изображението RGB е добър пример за данните, които трябва дабъдат достъпни в равнинатаза операции, като дисплея или филтриране. В други случаи, обаче,самите данни може да са с много измерения(многомерни). Помислете например за набор на температурата, замерванията, осъществени в точки, равномерно разположени в една стая.

Тук мястото на всяка една стойност е неразделна част от набор от данни,физическото разположение на три-пространството на всеки елемент е един от аспектите на информацията.Такива данни също придават,себе си като изображение на многомерен масив.



Сега, за да намерите средната стойност на всички измервания, използвайте

mean(mean(mean(TEMP)));

За да получите вектора на "средните" стойности (елемент (2,2)) във стаята за всяка страница използвайте:

B = TEMP(2,2,:);

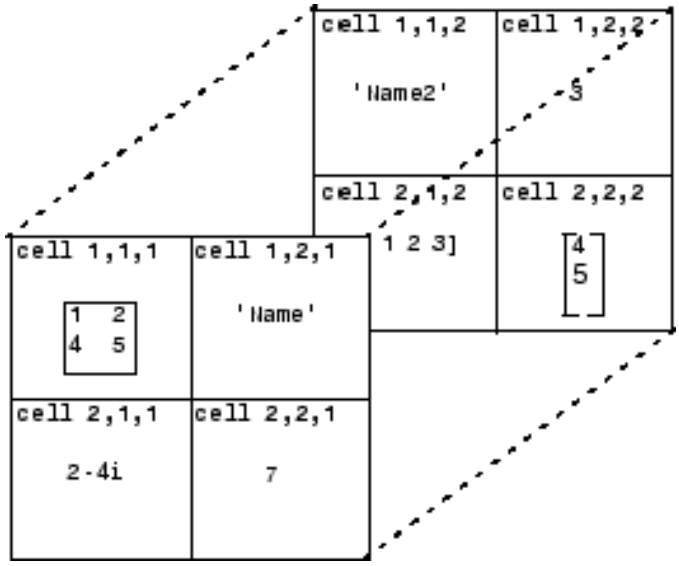
8. Многомерни масиви

Подобно на цифровите масиви, структурата относно клетките многомерни масиви във MATLAB е разщирение на двумерен модел на клетка масив. Можете да използвате функцията *cat* за изграждане на многомерни клетки масиви, също както я използвате за числови масиви.

Например, създаването на прости триизмерни масиви C:

```
A{1,1} = [1 2;4 5];
A{1,2} = 'Name';
A{2,1} = 2-4i;
A{2,2} = 7;
B{1,1} = 'Name2';
B{1,2} = 3;
B{2,1} = 0:1:3;
B{2,2} = [4 5]';
C = cat(3, A, B);
```

Указателните знаци за клетките от C изглеждат така:



9. Структура на многомерните масиви

Структурата на многомерните масиви е продължение на правоъгълната структура на масивите. Както другите видове на многомерни масиви и тези може да изградите чрез пряко прехвърляне или с функцията *cat*:

patient(1,1,1).name = 'John Doe';

patient(1,1,1).billing = 127.00;

patient(1,1,1).test = [79 75 73; 180 178 177.5; 220 210 205];

patient(1,2,1).name = 'Ann Lane';

patient(1,2,1).billing = 28.50;

patient(1,2,1).test = [68 70 68; 118 118 119; 172 170 169];

patient(1,1,2).name = 'Al Smith';

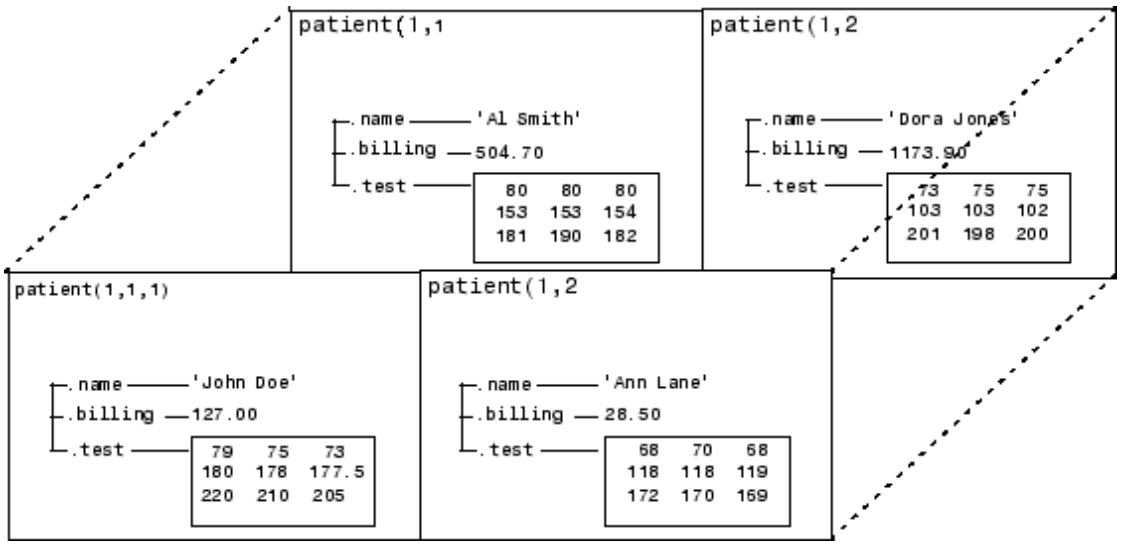
patient(1,1,2).billing = 504.70;

patient(1,1,2).test = [80 80 80; 153 153 154; 181 190 182];

patient(1,2,2).name = 'Dora Jones';

patient(1,2,2).billing = 1173.90;

patient(1,2,2).test = [73 73 75; 103 103 102; 201 198 200];



Прилагане на функции в структурите на многомерните масиви.

За да приложите функции в структурата на многомерните масиви управлявайте полетата и полетата с елементи ,използвайки индексиране.Например намерете сумата от колоните от *test* (тествания) масив patient(пациент) на (1,1,2):

sum((patient(1,1,2).test));

По същия начин ,добавете всички *billing* (фактурирани)полета в масива на *patient* (пациента):

total = sum([patient.billing]);

Раздел 10: *Обобщение на матрицата и Функциите на масивите*

Този раздел обобщава основните функции, използвани за създаване и обработка на матрици. Повечето от тези функции работят така добре и с многомерни масиви.

Функции за създаване на матрица

Функции	Описание
[a,b] or [a;b]	Създаване на матрица от определени елементи, или свързване на матрици заедно.
accumarray	Построяване на матрица чрез натрупване.
blkdiag	Изграждане на блок диагонална матрица.
cat	Свързване на матрици по определена величина.

diag	Създаване на диагонална матрица от вектор.
horzcat	Хоризонтално свързване на матрици.
magic	Създаване на квадратна матрица с редове, колони и диагонали, които допринасят за един и същ номер.
ones	Създаване на матрица от всички такива.
rand	Създаване на матрица от равномерно разпределени случайни числа.
repmat	Създаване на нова матрица, чрез копиране или обработване на друга такава.
vertcat	Свързване на две или повече матрици вертикално.
zeros	Създаване на матрица от всички нули.

Функции за промяна формата на Матрицата

Функции	Описание
ctranspose	Обръщане на матрица от главния диагонал и заменяне на всеки елемент с комплекс съединение.
flipdim	Обръщане на матрица по определена величина.
fliplr	Обръщане на матрица по отношение на вертикалната ос.

Функции за промяна формата на матрицата(продължение)

Функция	Описание
flipud	Обръщане на матрица спрямо хоризонталнаос.
reshape	Промяна на размерите на матрица.
rot90	Завъртане на матрицата с 90 градуса.
transpose	Обръщане на матрицата около главния диагонал.

Функции за намиране формата или структурата на матрица

Функция	Описание
isempty	Връща true за 0-по-0 или 0-по-n матрици
isscalar	Връща true за 1-по-1 матрици
issparse	Връща true за разредени матрици.
isvector	Връща true за 1-по-n матрици.

length	Връща дължината на вектора.
ndims	Връща броя на величините в матрица
numel	Връща броя на елементите в матрица
size	Връща размера на всяка величина.

Функции за определяне на клас

Описание	Функции
iscell	Връща true (истина) ako матицата е клетка масив.
ischar	Връща true ако елементите на матрицата са символи или поредици.
isfloat	Определя дали входа е масив с плаваща запетая.
isinteger	Определя дали входа е интегриран масив.
islogical	Връща true ако елементите на матрицата са логически.
isnumeric	Връща true ако елементите на матрицата са числови.

Функции за определяне на клас (продължение)

Функция	Описание
isreal	Връща true ако елементите на матрицата са реални числа.
isstruct	Връща true ако елементите на матрицата са MATLAB структури.

Функции за сортиране и промяна на елементите в матрицата

Функция	Описание
circshift	Кръгова смяна съдържанието на матрицата
issorted	Връща true ако елементите на матрицата са сортирани
sort	Сортира елементите в низходящ или възходящ ред
sortrows	Сортира елементите във възходящ ред

Функции които работят с диагоналите от матрицата

Функция	Описание
blkdiag	Изграждане на блок диагонална матрица.
diag	Връща диагоналите на матрицата
trace	Изчислява сумата от елементите по главния диагонал
tril	Връща долната триъгълна част от матрицата.

triu	Връща горната триъгълна част от матрицата

Функции за промяна на типа индексиране

Функция	Описание
ind2sub	Преобразува линеен индекс в ред-колона индекс
sub2ind	Преобразува ред-колона индекс в линеен индекс

Функции за работа с многомерни масиви

Функция	Описание
cat	Свързва матрици
circshift	Променя кръгово масив
ipermute	Размества обратно величината на масива
ndgrid	Създава масиви за n-измерими функции и интерполация(вмъкване,прибавяне)
ndims	Връща номера на величината на масива
permute	Размества (променя реда на) величината на масива
shiftdim	Променя величината на масива
squeeze	Премахва лъжливи измерения