ВЪВЕДЕНИЕ В MATLAB

Херардо Родригес
Университет на Саламанка

MATLAB (**MAT**rix **LAB**oratory) е интерактивна система за числени пресмятания и графика. Както е заложено в името, системата MATLAB е специално създадена за матрични изчисления: решаване на системи линейни уравнения, разлагане на матрици и т.н. Освен това, тя има и голямо разнообразие от графични възможности и може да се разширява с програми, написани на нейния собствен език.

Системата е създадена на базата на езика MATLAB. Най-простият начин за изпълнение на код на езика е да се запише след командния промпт ,>>, в прозореца за команди, който е елемент на работния екран на MATLAB. Последователност от команди може да се запомни в текстов файл, обикновено с използване на Редактора на MATLAB, като скрипт или вложен във функция, която разширява съществуващите команди.

В следващите секции ще разгледаме някои от най-използваните възможности на MATLAB. Дадени са голям брой примери. Най-добрият начин да се научите да работите с MATLAB е да четете примерите, докато сте стартирали MATLAB, опитвайки примерите и експериментирайки.

Въвеждане на вектори и матрици

Основният тип данни в MATLAB е *n*-мерният масив от числа с двойна точност. Новите типове данни включват сруктури, класове и "клеткови масиви", които са масиви с евентуално различен тип данни.

Следните команди показват как се задават числа, вектори и матрици и как им се присвояват стойности.

```
>> a = 2  //scalar
```

Ако натиснете Enter, ще видите:

```
a =
     2

>> x = [1;2;3] //Vector
```

Натиснете Enter,

```
x =
     1
     2
     3

>> A = [1 2 3;4 5 6;7 8 0] //Matrix
```

Натиснете Enter,

```
A =
     1     2     3
     4     5     6
     7     8     0
```

Забележете, че редовете на матрицата са разделени с точка и запетая, а елементите в реда – с интервал (или запетая).

Полезна команда е “whos”, която показва омената на всички дефинирани променливи и техните типове:

```
>> whos
  Name      Size          Bytes  Class

  A         3x3              72  double array
  a         1x1               8  double array
  x         3x1              24  double array

Grand total is 13 elements using 104 bytes
```

Забележете, че всяка от тези три променливи е масив; втората колона на масива определя точния му тип. Константата ‘a’ е масив с тазмерност 1×1, ‘x’ е 3×1 масив и матрицата ‘A’ е 3×3 масив (сравнете със съответния ред в “size”-размер).

Един начин за въвеждане на n-мерен масив (n>2) е да се слеят два или повече (n-1)-мерни масива с помощта на командата `cat`. Например следната команда слива два 3×2 масива и създава $3\times2\times2$ масив:

```
>> C = cat(3,[1,2;3,4;5,6],[7,8;9,10;11,12])

C(:,:,1) =
     1     2
     3     4
     5     6

C(:,:,2) =
     7     8
     9    10
    11    12

>> whos

  Name      Size          Bytes  Class

  A         3x3              72  double array
  C         3x2x2            96  double array
  a         1x1               8  double array
  x         3x1              24  double array

Grand total is 25 elements using 200 bytes
```

Забележете, че аргументът “3” в командата `cat` посочва, че сливането ставаспоред размерност 3. Ако D и E са $k\times m\times n$ масиви, командата

```
>> cat(4,D,E)
```

е създаде масив $k\times m\times n\times2$ (можете да опитате).

MATLAB позволява и работа с комплексни числа. Комплексната единица $i = \sqrt{-1}$ е предтавена с вградените променливи `i` или `j`:

```
>> sqrt(-1)

ans =
        0 + 1.0000i
```

Този пример показва как се изобразяват комплексните числа в MATLAB; вижда се също, че извличането на корен квадратен е вградената функция sqrt().

Резултатът от последното изчисление, не е присвоено на променлива, но системата автоматично го присвоява на променливата `ans`, която по-късно може да се използва, както и всяка друга променлива в следващите изчисления. Ето един пример:

```
>> 100^2-4*2*3 //Press enter

ans =
        9976

>> sqrt(ans) //Press enter

ans =
   99.8799

>> (-100+ans)/4 //Press enter

ans =
    -0.0300
```

Аритметичните оператори работят нормално както сме свикнали. Много често използвана константа е π:

```
>> pi  //Press enter

ans =
    3.1416
```

Някои често използвани функции като синус, цосинус, тангенс и логаритъм са предефинирани. Например:

```
>> cos(.5)^2+sin(.5)^2 //Press enter

ans =
     1

>> exp(1)  //Press enter

ans =
    2.7183

>> log(ans)  //Press enter
ans =
     1
```

Ако се съмнявате в синтаксиса на някоя команда или функция, съществува добре разработена он-лайн система за помощ, която можете да ползвате с командата `help <command-name>`. Например:

```
>> help ans

 ANS    The most recent answer.
        ANS is the variable created automatically when expressions
        are not assigned to anything else. ANSwer.

>> help pi

 PI     3.1415926535897....

 PI = 4*atan(1) = imag(log(-1)) = 3.1415926535897....
```

Добро начало е командата `help help`, която обяснява как работи системата за помощ, както и на близки команди. Набирането на `help` от самосебе си показва списък от елементи,за всеки от които съществува помощ; преглеждайки този лист намираме вход към “elfun—елементарни математически функции”. Набираме `help elfun` и получаваме списък с наличните елементарни вградени функции.

Аритметични операции с матрици

MATLAB може да извършва стандартните операции с матрици, вектори и константи: събиране, изваждане и умножение. Освен това, MATLAB въвежда понятието матрично деление, както и “векторизирани” операции. Всички векторизирани операции (вкл. Събиране, изваждане и умножение с константа, както е обяснено по-долу) могат да се прилагат и към *n*-мерни масиви за всяка стойност на *n*, но умножението и делението са ограничени само за матрици и вектори ($n \leq 2$).

Стандартни операции

Ако `A` и `B` са масиви, то MATLAB може да изчисли `A+B` и `A-B` *когато тези операции са определени*. Например да разгледаме следните команди:

```
>> A = [1 2 3;4 5 6;7 8 9];
>> B = [1 1 1;2 2 2;3 3 3];
>> C = [1 2;3 4;5 6];

>> whos

  Name      Size         Bytes  Class

  A         3x3             72  double array
  B         3x3             72  double array
  C         3x2             48  double array

Grand total is 24 elements using 192 bytes

>> A+B
ans =
     2     3     4
     6     7     8
    10    11    12
```

Но ако запишете:

```
>> A+C

??? Error using ==> + because Matrix dimensions must agree.
```

Умножението на матриците е също определено:

```
>> A*C

ans =
    22    28
    49    64
    76   100

>> C*A

??? Error using ==> * because Matrix dimensions must agree.
```

Ако `A` е квадратна матрица и `m` е положително цяло число, тогава `A^m` е произведението на `m` множителя `A`.

Но няма дефиниция за умножение за многомерни масиви с размерност по-голяма от 2:

```
>> C = cat(3,[1 2;3 4],[5 6;7 8])

C(:,:,1) =
     1     2
     3     4

C(:,:,2) =
     5     6
     7     8

>> D = [1;2]

D =
     1
     2

>> whos
  Name      Size         Bytes  Class

  C         2x2x2           64  double array
  D         2x1             16  double array

Grand total is 10 elements using 80 bytes

>> C*D
??? Error using ==> *

No functional support for matrix inputs.
```

Аналогично експоненциалният оператор `^` е дефиниран само за квадратните двумерни масиви (матрици).

Решаване на матрични уравнения с използване на операцията "матрично деление"

Ако A е квадратна неособена матрица, тогава решението на уравнението $Ax = b$ е $x = A^{-1} b$. MATLAB извършва тази операция с използване на оператор с обратнонаклонена черта (\):

```
>> A = rand(3,3)

A =
    0.2190    0.6793    0.5194
    0.0470    0.9347    0.8310
    0.6789    0.3835    0.0346

>> b = rand(3,1)

b =
    0.0535
    0.5297
    0.6711

>> x = A\b

x =
 -159.3380
  314.8625
 -344.5078
```

Забележка: Използването на вградената функция `rand` генерира матрица, чиито елементи са равномерно разпределени случайни числа в интервала (0,1). (Виж `help rand` за повече подробности.)

Това `A\b` е (математически) еквивалентно на умножението на b отляво с A^{-1} (впрочем, MATLAB не изчислява обратната матрица; вместо това той решава линейната система директно). Ако се използва за неквадратни матрици, операторът `\` решава съответната система в смисъла на метода на най-малките квадрати – виж `help slash` за детайли.

Разбира се, както и при другите аритметични операции, матриците трябва да са съвместими по размер. Операторът за делене на матрици не е дефиниран за масиви с размерност n>2.

"Векторизирани" функции и оператори

MATLAB има много команди за създаване на специални матрици; следната команда създава вектор-ред, чиито компоненти са нарастващи естествени числа:

```
>> t = 1:5
t =
     1     2     3     4     5
```

Компонентите могат да се променят и със стъпка, различна от 1:

```
>> x = 0:.1:1
```

```
x =
  Columns 1 through 7
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
  Columns 8 through 11
    0.7000    0.8000    0.9000    1.0000
```

Допуска се и отрицателна стъпка.

Командата `linspace` дава подобни резултати. По-специално, `linspace(a,b,n)` създава вектор с n компонента от вида $a, a+\frac{b-a}{n-1}, a+\frac{2(b-a)}{n-1}, ..., b$:

```
>> linspace(0,1,11)

ans =
  Columns 1 through 7
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
  Columns 8 through 11
    0.7000    0.8000    0.9000    1.0000
```

Има и друга подобна програма `logspace` за създаване на вектори с логаритмично отдалечени компоненти:

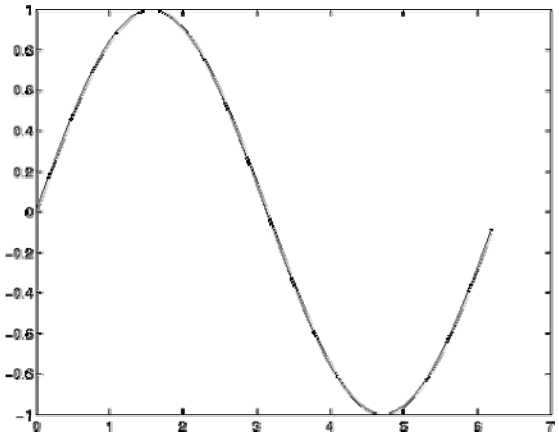
```
>> logspace(0,1,11)

ans =
  Columns 1 through 7
    1.0000    1.2589    1.5849    1.9953    2.5119    3.1623    3.9811
  Columns 8 through 11
    5.0119    6.3096    7.9433   10.0000
```

Виж `help logspace` за детайлите.

Вектор с линейно разположени компоненти е необходим при задаване на едномерна мрежа, която се използва в командите за графика. За начертаване графиката на функцията $y = f(x)$ и съединяване на тичките с отсечки може да се вектор за стойностите на `x` и след това – вектор със съответните стойности на `y`.

Лесно е да се създадат нужните вектори за построяване на графиката на вградена функция, тъй като функциите на MATLAB са "векторизирани". Това означава, че ако вградената функция, напр. sin() се приложи върху масив, то ефектът е, че веднага се получава нов масив със същата размерност, чийто компоненти са функционалните стойности с аргументи подадения масив. Например (виж Фиг. 1):

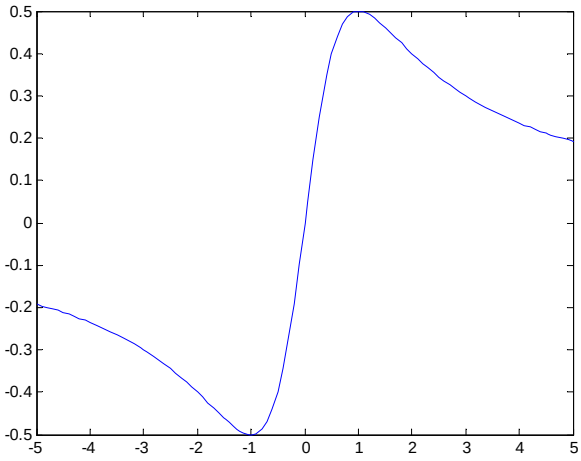
```
>> x = (0:.1:2*pi);
>> y = sin(x);
>> plot(x,y)
```



Фиг. 1: Графика на $y = sin(x)$

MATLAB предлага също и векторизирани аритметични оператори, които са същите като обикновените, но се предхождат от символа "**.**". Например да начертаем $y = \frac{x}{1+x^2}$:

```
>> x = (-5:.1:5);
>> y = x./(1+x.^2);
>> plot(x,y)
```



Фиг. **2:** Графика на $\mathbf{y = x / (1 + x^2)}$

Тук `x.^2` повдига на кавадрат всички `x`, а `x./z` дели всяка компонента на `x` на съответната компонента на `z`. Събирането и изваждането се извършват по координатно и за тях няма операции с предходна точка (.).

Обърнете внимание на разликата между `A^2` и `A.^2`. Първата операция е дефинирана само ако `A` е квадратна матрица, докато втората операция е дефинирана за всеки n-мерен масив `A`.

Някои допълнителни команди

Важен оператор в MATLAB е “ ' ”, който извършва (спрегнато) транспониране:

```
>> A = [1 2;3 4]

A =
     1     2
     3     4

>> A'

ans =
     1     3
     2     4

>> B = A + i*.5*A

B =
   1.0000 + 0.5000i   2.0000 + 1.0000i
   3.0000 + 1.5000i   4.0000 + 2.0000i

>> B'
ans =
   1.0000 - 0.5000i   3.0000 - 1.5000i
   2.0000 - 1.0000i   4.0000 - 2.0000i
```

В редки случаи на транспониране, както и на спрегнато транспониране се налага използване на операцията “**.**'”:

```
>> B.'

ans =
   1.0000 + 0.5000i   3.0000 + 1.5000i
   2.0000 + 1.0000i   4.0000 + 2.0000i
```

(забележете, че `' .'` са еквивалентни за матрици от реални числа).

Често се използват следните команди. Повече подробности можете да намерите в системата за он-лайн помощ.

Създаване на матрици

- `zeros(m,n)` създава $m \times n$ матрица от нули;
- `ones(m,n)` създава $m \times n$ матрица от единици;
- `eye(n)` създава $n \times n$ единична матрица;
- `diag(v)` (`v` е n-вектор) създава $n \times n$ диагонална матрица с `v` по главния диагонал.

Например:

```
>> ones(3,4)

ans =
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

```
>> v=[-1 2 3.5]
v =
   -1.0000    2.0000    3.5000

>> diag (v)
ans =
   -1.0000         0         0
         0    2.0000         0
         0         0    3.5000
```

Командите `zeros` и `ones` могат да имат произволен брой цели аргументи; с k аргумента всяка от тях създава k-мерен масив с указания размер.

Форматиране на дисплея и графиките

Следните команди показват на екрана по различен начин различни резултати.

- `format` : Задава изходния формат

```
>> format short, pi
ans =
    3.1416
>> format short e, pi
ans =
      3.1416e+000
>> format long, pi
ans =
      3.14159265358979
>> format long e, pi
ans =
    3.141592653589793e+000
```

`format compact` премахва излишните редове (всички изходи в тази статия са в компактен формат)

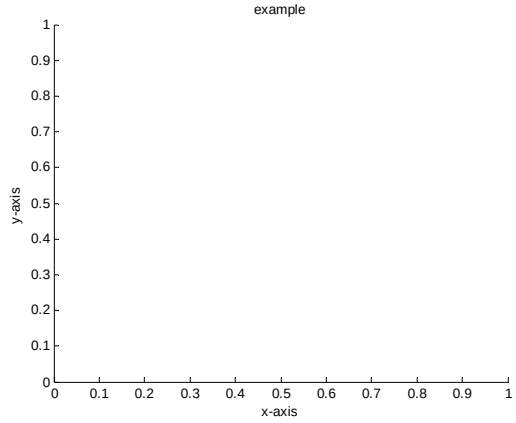
`format loose` връща обратно всички премахнати празни редове

```
>> format loose, pi



ans =
    3.1416
```

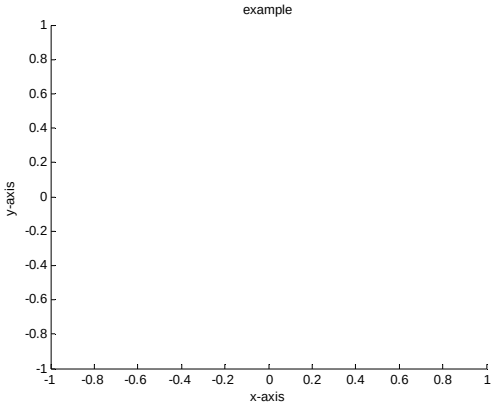
- `xlabel('string'), ylabel('string')` надписва хоризонталната и вертикалната ос в текущата графика;
- `title('string')` добавя заглавие към графиката;
- `axis([a b c d])` сменя прозореша на графиката в областта $a \le x \le b, c \le y \le d$;
- `grid` добавя правоъгълна мрежа към графиката;
- `hold on` съхранява текущата графика така, ч епосле да да я покаже съвместно с друга графика ;
- `hold off` показва текущата графиак; следващата графика ще изтрие тази, преди да се покаже;
- `subplot` слага няколко графики в един екран за графики.

Например:

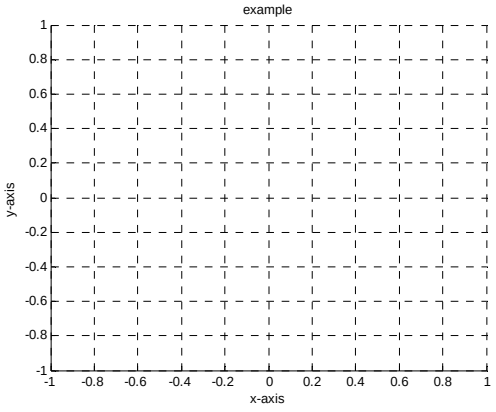
```
>> xlabel('x-axis');
>> ylabel('y-axis');
>> title('example');
```




```
>> axis([-1 1 -1 1]);
```



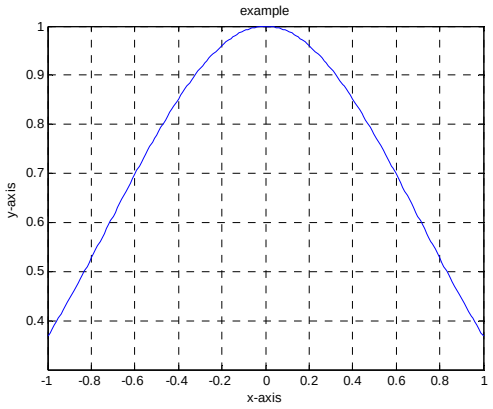

```
>> grid on
```



Ще се начертае графиката на функцията y= exp (-x^2) с означени оси и надпис.

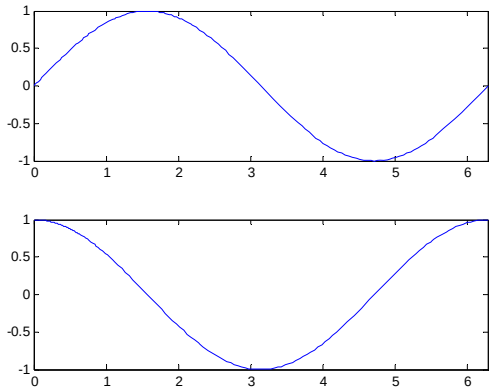
```
>> u=x.^2;
```

```
>> y= exp(-u);
>> plot (x,y);
```



Сега ще покажем много графики едновременно с функцията `subplot`.

```
>> subplot(2,1,1);
>> fplot('sin(x)',[0 2*pi])
>> subplot(2,1,2);
>> fplot('cos(x)',[0 2*pi])
```



Други

- `max(x)` дава най-големият елемент на x, ако x е вектор; виж `help max` за случая, когато `x` е k-мерен масив;
- `min(x)` аналогично на `max`;
- `abs(x)` получава масив със същия размер като `x`, чийто елементи са големините `на x`;
- `size(A)` дава $1 \times k$ вектор с брой редове, колони и т.н. на k-мерен масив `A`;
- `length(x)` дава дължината на масива, т.е. `max(size(A))`.
- `save fname` запомня текущите променливи във файл с име fname.mat;
- `load fname` зарежда променливите от файла fname.mat;
- `quit` затваря MATLAB.

Например:

```
>> x=[ 3 -4 60 -71 -13 12];
>> max(x)
ans =
    60
>> min(x)
ans =
   -71
>> abs(x)
ans =
     3     4    60    71    13    12
```

Пример: Числено интегриране

Численото интегриране е приближено изчисление на интеграл с използване на изчислителни техники. Съществуват голям брой методи за числено интегриране. В този пример ще използваме три от тях: правило на правоъгълниците, на трапеците и на Симпсън за изчисляване стойността на интеграла

$$\int_0^1 (1-x)^{1/2}\,dx$$

1. Правило на правоъгълниците

Първата много проста схема е да приближим функцията $f(x)$ във всеки подинтервал с константа и да разгледаме площта на правоъгълника с ширина $h = x_{i+1} - x_i$ и височина $f(x_i)$ за i-тия интервал. Тогава е просто нещо да съберем лицата на всички правоъгълници в интервала.
Интервалът ще се разделя на 2, 4, 8, 16, 32, 64 и 128 подинтервала.
Най0напред пример с 2 подинтервала.

Необходими са три точки в (0,1).

```
>> X=linspace(0,1,3)

X =
         0    0.5000    1.0000
```

Стойностите на функцията $\sqrt{1-x}$ са запомнени в променливата Y.

```
>> Y=sqrt(1-X)

Y =
    1.0000    0.7071         0
```

Първият правоъгълник има височина $f(0)$, така че е изчислен първият компонент на вектора Y .

```
>> q=Y(1)

q =
     1
```

Вторият правоъгълник има височина *f(1),* т.е. намерихме и втория компонент на вектора Y .

```
>> q=q+Y(2)

q =
    1.7071
```

Ширината на правоъгълниците е 0.5, следователно сумата от височините се умножава по тази стойност и се намира площта.

```
>> q=q*1/2

q =
    0.8536
```

С цел да разширим броя подинтервали ще направим цикъл за работа с масивите. 129 точки са необходими за разделяне на интервала на 128 подинтервала.

```
>> X=linspace(0,1,129);
```

Стойностите на функцията $\sqrt{1-x}$ се запомнят пак във вектор Y .

```
>> Y=sqrt(1-X);
>> q=Y(1);
```

Този път сумата е от *f(0)* до *f(127)*, т.е. от първия компонент *Y(1)* до последния *Y(128).*

```
>> for j=1:127
q=q+Y(j+1);
end
>> q=q*1/128;
```

Накрая ще използваме следната програма за изчисляване на интеграла с използване на 2, 4, 8, 16, 32, 64 и 128 подинтервала:

```
function rectangleRule=rectangleRule(Y)
q=zeros(7,1);
N=2; %Number of subintervals
for i=1:7
    h=1/N;
    q(i)=Y(1);
    for j=1:N-1
        q(i)=q(i)+Y(j*(256/N)+1);
    end
    q(i)=q(i)*h;
    N=N*2;
end
disp('Approximations with the rectangle rule:')
      q
```

Входът на тази функция е вектор с 256 компонента и съответните стойности на подинтегралната функция в интервала [0,1].

```
>> X=linspace(0,1,257);
>> Y=sqrt(1-X);
>> rectangleRule(Y)
```

```
Approximations with the rectangle rule:

q =
   0.85355339059327
   0.76828304624275
   0.72063022162445
   0.69483119687723
   0.68118393627894
   0.67408331137851
   0.67043190729683
```

2. Правило на трапеците

В този пример предишната програма се модифицира за приближаване на интеграла по метода на трапеците. Ще използваме следната програма:

```
function trapezoidalRule=trapezoidalRule(Y)
q=zeros(7,1);
N=2; %Number of subintervals
for i=1:7
   h=1/N;
   q(i)=0.5*Y(1);
   q(i)=q(i)+0.5*Y(257);
   for j=1:N-1
      q(i)=q(i)+Y(j*(256/N)+1);
   end
   q(i)=q(i)*h;
   N=N*2;
end
disp('Approximations with the trapezoidal rule:')
q
```

Входът на тази функция е вектор с 256 компонента и съответните стойности на подинтегралната функция в интервала [0,1].

```
>> X=linspace(0,1,257);
>> Y=sqrt(1-X);
>> trapezoidalRule(Y)
Approximations with the rectangle rule:

q =

   0.60355339059327
   0.64328304624275
   0.65813022162445
   0.66358119687723
   0.66555893627894
   0.66627081137851
   0.66652565729683
```

3. Правило на Симпсън

Накрая прилагаме правилото на Симпсън. Сега програмата се променя така:

```
function simpsonRule=simpsonRule(Y)
q=zeros(7,1);
N=2; %Number of subintervals
for i=1:7
   h=1/N;
```

```
    q(i)=Y(1);
    q(i)=q(i)+Y(257);
    k=0;
       for j=1:N-1
          k=k+1;
          q(i)=q(i)+4*Y(k/2*(256/N)+1);
          k=k+1;
          q(i)=q(i)+2*Y(k/2*(256/N)+1);
       end
       k=k+1;
       q(i)=q(i)+4*Y(k/2*(256/N)+1); %Last interval middle point.
       q(i)=q(i)*h/6;
       N=N*2;
end
disp('Approximations with Simpson rule:')
q
```

Входът на тази функция е вектор с 256 компонента и съответните стойности на подинтегралната функция в интервала [0,1].

```
>> X=linspace(0,1,257);
>> Y=sqrt(1-X);
>> simpsonRule(Y)
Approximations with Simpson rule:

q =

   0.65652626479257
   0.66307928008502
   0.66539818862815
   0.66621818274618
   0.66650810307836
   0.66661060593627
   0.66664684620310
```

Резултатите от трите приближения са представени в следната таблица:

Брой подинтервали	Правило на правоъгълниците	Правило на трапеците	Правило на Симпсън
2	0.85355339059327	0.60355339059327	0.65652626479257
4	0.76828304624275	0.64328304624275	0.66307928008502
8	0.72063022162445	0.65813022162445	0.66539818862815
16	0.69483119687723	0.66358119687723	0.66621818274618
32	0.68118393627894	0.66555893627894	0.66650810307836
64	0.67408331137851	0.66627081137851	0.66661060593627
128	0.67043190729683	0.66652565729683	0.66664684620310