3. Hafta-2

Kullanıcı Tanımlı Fonksiyonlar

M-dosyaları, script dosyaları, fonksiyon dosyaları
Anonim & inline fonksiyonları
Fonksiyon tanıtıcıları
fzeros, fminbnd fonksiyonları
Çoklu girdi & çıktı
Alt fonksiyonlar, iç içe fonksiyonlar
Ödev şablon fonksiyon
Fonksiyon tipleri
Tekrarlı (recursive) fonksiyonlar, fraktallar

M-dosyaları: script dosyaları ve fonksiyon dosyaları

Script M-dosyaları, komut penceresine (command window) yazılmış gibi çalıştırılacak komutları içerir, yani, birçok komutu tek dosyada toplarlar.

Fonksiyon M-dosyaları, bir **function** tanım satırıyla başlamalı ve giriş değişkenlerini kabul edebilir ve/veya çıkış değişkenlerini geri getirebilir olmalıdır.

Fonksiyon tanım satırı şöyle bir sözdizimi vardır:

```
function [outputs] = func(inputs)
```

Burada, fonksiyon ismi **func** keyfidir ve M-dosyasının ismi ile eşleşmesi gerekir, yani, **func.m**

Bir script M-dosyasında tanımlanan değişkenler, mevcut workspace'in tamamında, yani, script dosyasının içinde ve dışında bilinir.

Script M-dosyaları, fonksiyonlar satıriçi veya anonim tek satırlı fonksiyonlar, örneğin, function-handle @(x) kullanılarak oluşturulanlar, tanımlanmadıkça hiçbir fonksiyon tanımı içermeyebilir.

Bir fonksiyon M-dosyasındaki değişkenler sadece bu fonksiyona göre yereldir ve fonksiyonun dışında tanımlanmazlar (genellikle önerilmeyen global değişkenler olarak tanımlanmadıkça)

Fonksiyon dosyaları, diğer fonksiyonların tanımını alt fonksiyonlar veya iç içe fonksiyonlar olarak içerebilir. Bu, tüm ilgili fonksiyonları tek bir dosyada toplamaya yardımcı olur.

Örnek

```
% rms.m dosyası, x vektörünün karekök ortalama değerini
% ve mutlak ortalama değerini hesaplar.

function [r,m] = rms(x)
  r = sqrt(sum(abs(x).^2)/length(x));
  m = sum(abs(x))/length(x);
```

```
>> x=-4:4;
>> [r,m]=rms(x)

r =

    2.5820


m =

    2.2222

>>
```

Sadece ilk çıktıyı verir.

```
>> r=rms(x)

r =

    2.5820
```

Sadece ikinci çıktıyı verir.

```
>> [~,m]=rms(x)

m =

    2.2222
```

Üç yöntemle fonksiyonlarınızı oluşturabilirsiniz:

1. **function-handle, @(x)**
2. **inline**
3. **M-file**

Örnek 1

$$f(x) = e^{-0.5x}\sin(5x)$$

```
>> f = @(x) exp(-0.5*x).*sin(5.*x)

f =

  function_handle with value:

    @(x)exp(-0.5*x).*sin(5.*x)
```

```
>> g = inline('exp(-0.5*x).*sin(5.*x)')

g =

     Inline function:
     g(x) = exp(-0.5*x).*sin(5.*x)
```

```
% h.m script dosyasını düzenleyip kaydederek
function y=h(x)
y = exp(-0.5*x).*sin(5.*x);
```

.* vektör veya matris girdisine (x) izin verir.

Örnek 2

$$f(x) = e^{ax}\sin(bx)$$

Metod 1: Önce a ve b, sonra da f tanımlanır

```
>> a=0.5; b=5;
>> f = @(x) exp(-a*x).*sin(b*x)
```

Metod 2: a ve b de değişken olarak tanımlanır

```
>> f = @(x,a,b) exp(-a*x).*sin(b*x);
```

```
% Bu, f(x,a,b) fonksiyonunu tanımlar.
% Bundan dolayı, f(x, 0.5, 5)  metod 1'de tanımlanan f ile eşdeğerdir.
```

Örnek 3: Aşağıdaki π serisinin yakınsaklık testi

$$\pi = \lim_{N\to\infty} 2\sqrt{3}\sum_{k=0}^{N}\frac{(-1)^k}{(2k+1)3^k}$$

```
>> g = @(N) 2*sqrt(3)*cumsum((-1).^(0:N)./(2*(0:N)+1)./3.^(0:N));
```

veya

```
% f.m script dosyasını düzenleyip kaydederek
function y = f(N)

k=0:N;

y = 2*sqrt(3)*cumsum((-1).^k./(2*k+1)./3.^k);
```

Yakınsaklık sonuçları

N	f(N) veya g(N)
5	3.141308785462 88
10	3.141593304503 08
15	3.141592651734 00
20	3.141592653595 64
25	3.141592653589 77
30	3.141592653589 79
35	3.141592653589 79
40	3.141592653589 79

Not: f(N) ve g(N) fonksiyonları eşdeğer sonuçlar verir.
g(N) tek satırlı bir tanımdır, fakat okunması daha zordur.
f(N) okunması daha kolaydır, fakat M dosyası gerektirmektedir, f.m

Örnek 4: Bir fonksiyonun Fourier serisi açılımı

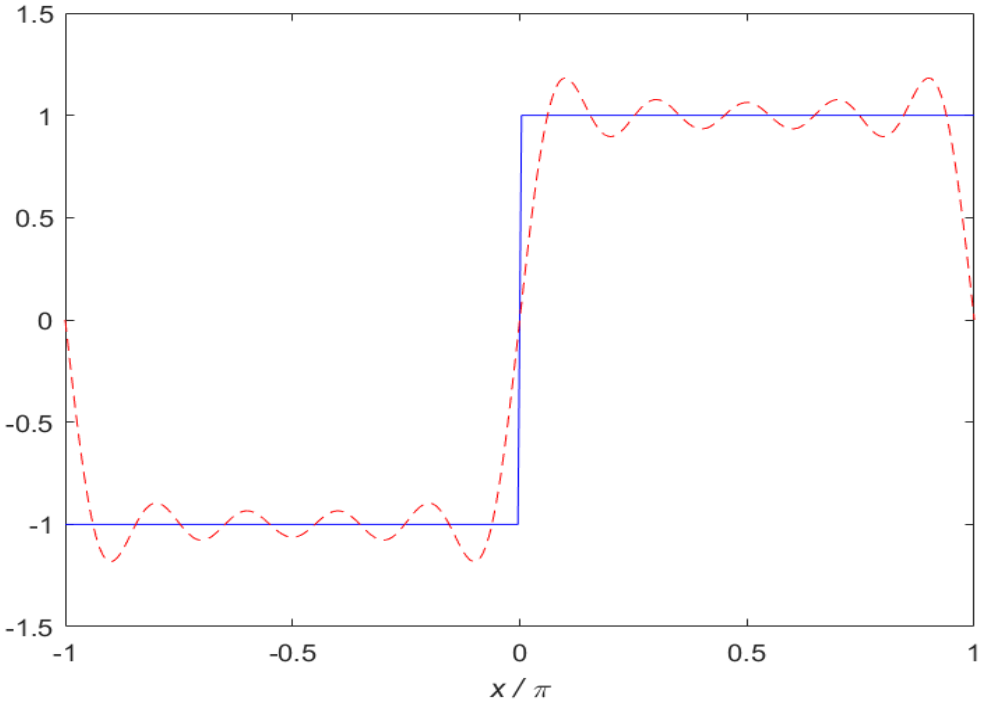
$$f(x)=\begin{cases}+1, & 0<x\le\pi\\ 0, & x=0\\ -1, & -\pi\le x<0\end{cases}$$

$$f(x)=\frac{4}{\pi}\sum_{k=0}^{\infty}\frac{\sin((2k+1)x)}{2k+1}$$

Sadece k=0:4 terimlerini hesaplayalım. F(x) fonksiyonunu tanımlayıp hesaplayalım ve f(x) ve F(x) fonksiyonlarını çizdirelim.

$$F(x)=\frac{4}{\pi}\sum_{k=0}^{4}\frac{\sin((2k+1)x)}{2k+1}$$

```
>> f = @(x) sign(x).*(abs(x)<=pi);
>> F = @(x) 4/pi*(sin(x) + sin(3*x)/3 + sin(5*x)/5 + sin(7*x)/7 + sin(9*x)/9);
>> x=linspace(-pi,pi,501);
>> plot(x/pi,f(x),'b-', x/pi, F(x), 'r--'); xlabel('\it{x} / \pi')
```



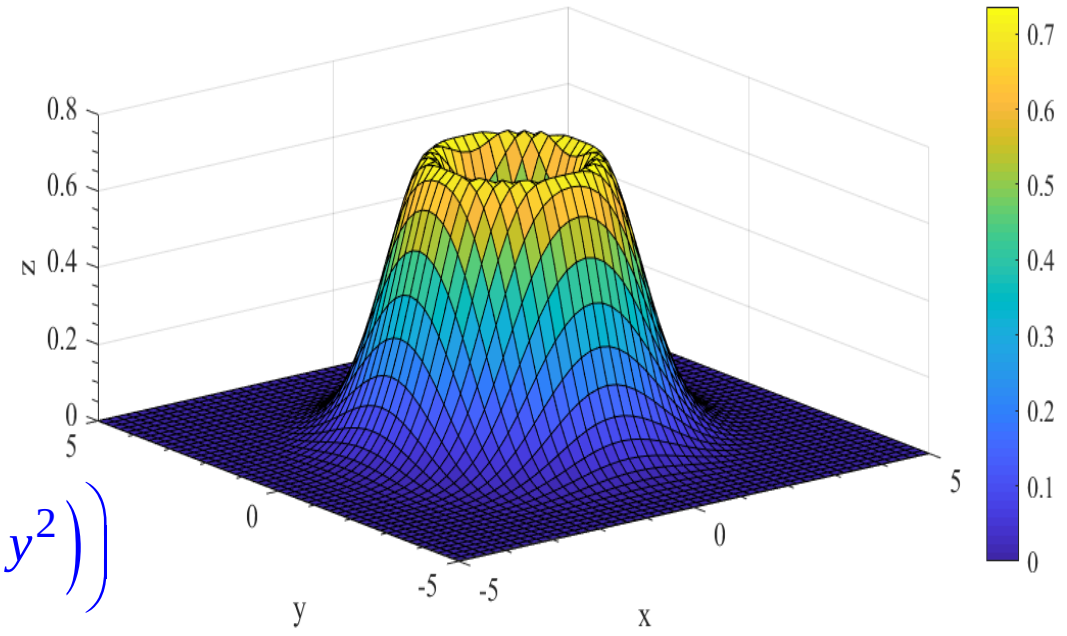
Not: x bir vektör ise

`(abs(x)<=pi)`

mantıksal ifadesi 0 veya 1'den oluşan bir vektör oluşturur

Örnek 5: Çok değişkenli fonksiyonlar

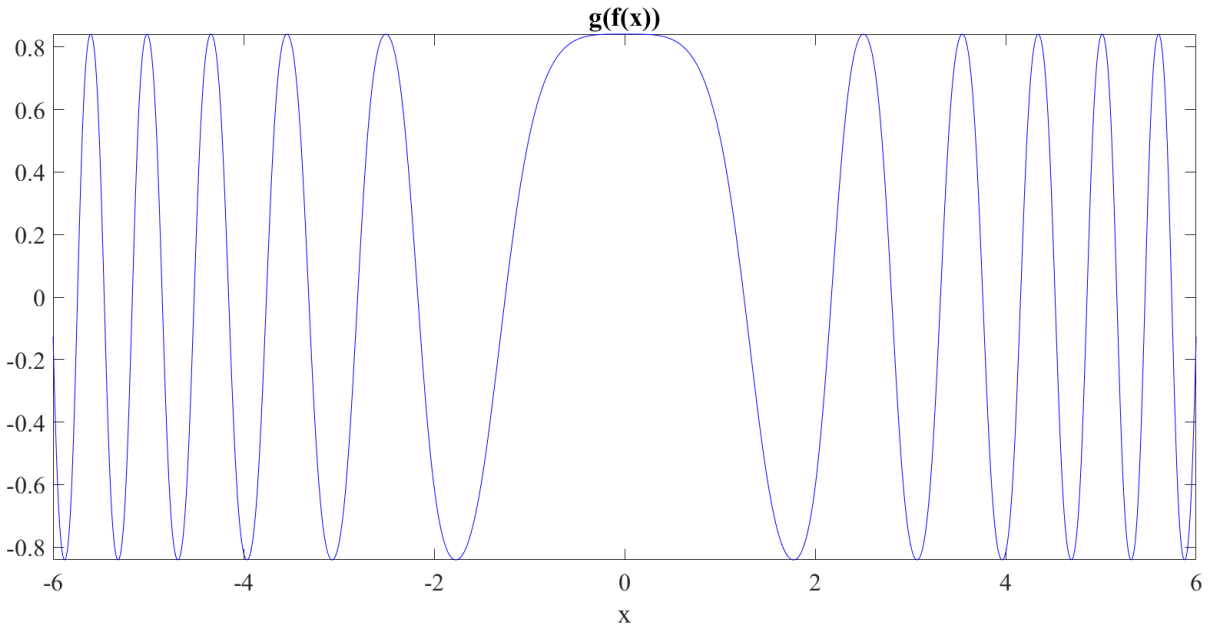
```
>> f = @(x,y) (x.^2 + y.^2).*exp(-(x.^2 + y.^2)/2);
>>
>> x=linspace(-5,5,51);
>> y=linspace(-5,5,51);
>> [X,Y] = meshgrid(x,y);
>>
>> Z = f(X,Y);
>>
>> surf(X,Y,Z); colorbar;
```



$$f(x,y)=\left(x^2+y^2\right)\exp\left(-\frac{1}{2}\left(x^2+y^2\right)\right)$$

Fonksiyonlar iç içe yazılabilir

```
>> f = @(x) x.^2;
>> g = @(x) sin(cos(x));
>> h = @(x) g(f(x));
>>
>> fplot(h,[-6,6],'b-'); xlabel('x'); title('g(f(x))');
```



Function Handle

Function handle bir fonksiyonun referanslanmasına ve değerlendirilmesine imkan veren ve aynı zamanda fonksiyonu başka fonksiyonlara, örneğin **fplot, ezplot, fzero, fminbnd, fminsearch**, geçiren bir veri türüdür.

Anonim fonksiyonlarda, örneğin **f=@(x) (ifade)**, tanımlanmış nicelik **f**, bir fonksiyon tanıtıcısıdır (function handle).

M dosyalarındaki hazır veya kullanıcı tanımlı fonksiyonlar için function handle fonksiyon isminin başına @ karakteri eklenerek kullanılır, örneğin

```
f_handle = @sin;
f_handle = @my_function;
```

Fonksiyon fonksiyonları

>> help funfun

Birçok MATLAB fonksiyonu diğer fonksiyonları argümanı olarak kabul eder. Bu tür fonksiyonlar aşağıdaki kategorileri kapsar:

1. **Fonksiyon optimizasyonu (minimizasyon/maksimizasyon), kök bulma, grafik çizme, data uydurma, örneğin fplot, ezplot, fzero, fminbnd, fminsearch, nlinfit, lsqcurvefit**

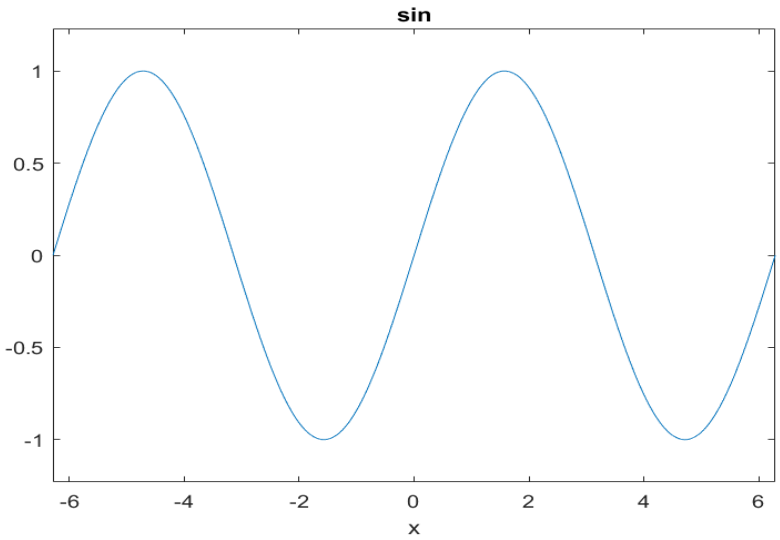
2. **Nümerik integrasyon (kareleştirme), örneğin quad ve onun farklı biçimleri**

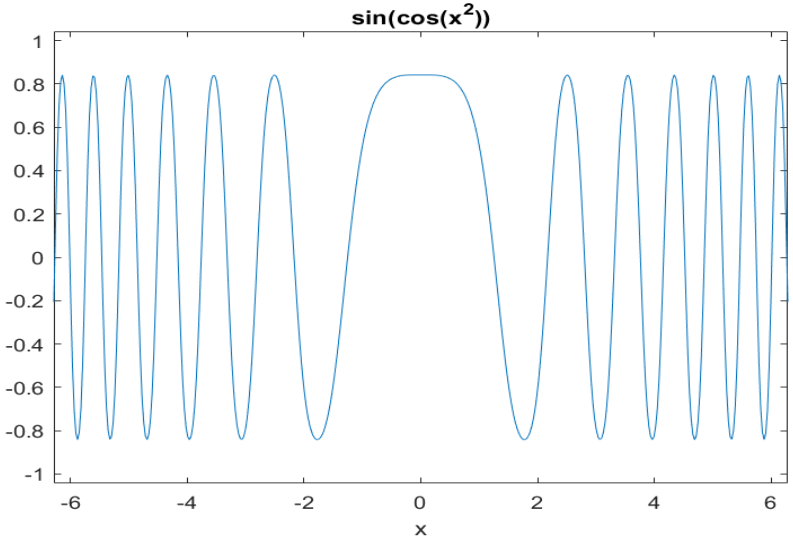
3. **Diferansiyel denklem çözücüleri, örneğin ode45 ve diğerleri**

4. **Başlangıç değer ve sınır değer problemi çözücüleri**

Fonksiyon argümanı bir function handle olarak (yeni metod) ya da fonksiyon ismini belirten bir string olarak (eski metod) iletilir.

```
>> ezplot(@sin);                          % function handle
>> ezplot('sin');                         % eski metod
>> ezplot('sin(x)');
>>
>> f=@(x) sin(cos(x.^2));
>> ezplot(f);                                        % eşdeğer
>> ezplot(@(x) sin(cos(x.^2)));                      % metodlar
>> ezplot(@(x) sin(cos(x^2)));
>> ezplot('sin(cos(x^2))');
```





fzero kullanarak Van der Waals denkleminin çözümü

$$f(V) = \left(P + \frac{n^2 a}{V^2} \right)(V - nb) - nRT = 0$$

```
P = 220; n = 2;
a = 5.536; b = 0.03049;
R = 0.08314472; T = 1000;

V0 = n*R*T/P;              % ideal gaz durumu, V0 = 0.7559

f = @(V) (P+n^2*a./V.^2).*(V-n*b)-n*R*T;

V = fzero(f,V0)

V =
    0.6825
```

V0 civarında **f(V)=0** çözümünü arar.

Ek parametrelere sahip bir fonksiyon için fzero kullanmak

```
f = @(x,a,b) ... % f(x,a,b), burada tanımlanır
                 % veya ayrı bir M-file'da tanımlanır

% f(x,a,b) = 0 çözümünü tanımlamak için
% burada a,b tanımlanır

x = fzero(@(x) f(x,a,b),x0);
```

Yeni bir anonim fonksiyonu etkili biçimde tanımlar ve function handle biçiminde fzero içinde kullanılır.

Benzer metod fminbnd ve diğer tüm fonksiyon fonksiyonlarında kullanılabilir.

>> doc fzero

Örnek 1

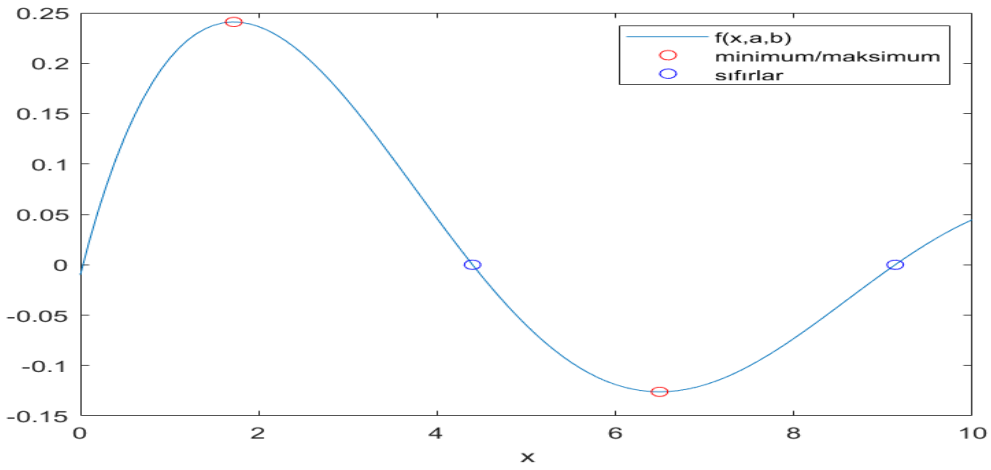
```
P = 220; n = 2;
a = 5.536; b = 0.03049;
R = 0.08314472; T = 1000;

V0 = n*R*T/P;              % ideal gaz durumu, V0 = 0.7559

f = @(V,a,b) (P+n^2*a./V.^2).*(V-n*b)-n*R*T;

V = fzero(@(V) f(V,a,b), V0)

V =
    0.6825
```

Yeni bir anonim fonksiyon tanımlanıp function handle olarak fzero içinde kullanıldı

Örnek 2

```
f = @(x,a,b,c) sin(a*x)./(x + b) + c;
a = 0.7; b = 2; c = -0.01;
[x1,f1] = fminbnd(@(x) -f(x,a,b,c), 0,4); f1 = -f1;
[x2,f2] = fminbnd(@(x) f(x,a,b,c), 6,8);
x3 = fzero(@(x) f(x,a,b,c), 5);
x4 = fzero(@(x) f(x,a,b,c), 9);
x = linspace(0,10,101);
y = f(x,a,b,c);
plot(x,y, [x1,x2], [f1,f2], 'ro', [x3,x4], [0,0], 'bo');
```

$$f(x,a,b,c) = \frac{\sin(ax)}{x+b} + c$$



sinc fonksiyonları çoğu mühendislik uygulamasında kullanılır:

1. Sinyallerin Fourier analizleri
2. Optik sistemler (Mikroskop ve teleskopların çözme gücü)
3. Radar sistemleri
4. DSP uygulamaları ve dijital iletişim
5. Anten dizileri, sonar ve sismik diziler
6. CD ve MP3 çalarların playback sistemleri (sinc interpolasyon filtresi veya örnek dijital filtreler olarak bilinir)
7. ve daha fazlası

$$\mathbf{sinc(x)} = \frac{\sin(x)}{x}, \qquad \mathbf{sinc(x)} = \frac{\sin(\pi x)}{\pi x}$$

matematiksel tanım

MATLAB tanımı

Örnek 3 – 3-dB genişliğinde sinc fonksiyonu

```
fplot(@sinc, [-4,4], 'b-'); hold on;
f = @(x) sinc(x)-1/sqrt(2);
x0 = fzero(f,0.5);                    % x0 = 0.443
plot([-x0,x0],[1,1]/sqrt(2),'r-');
```

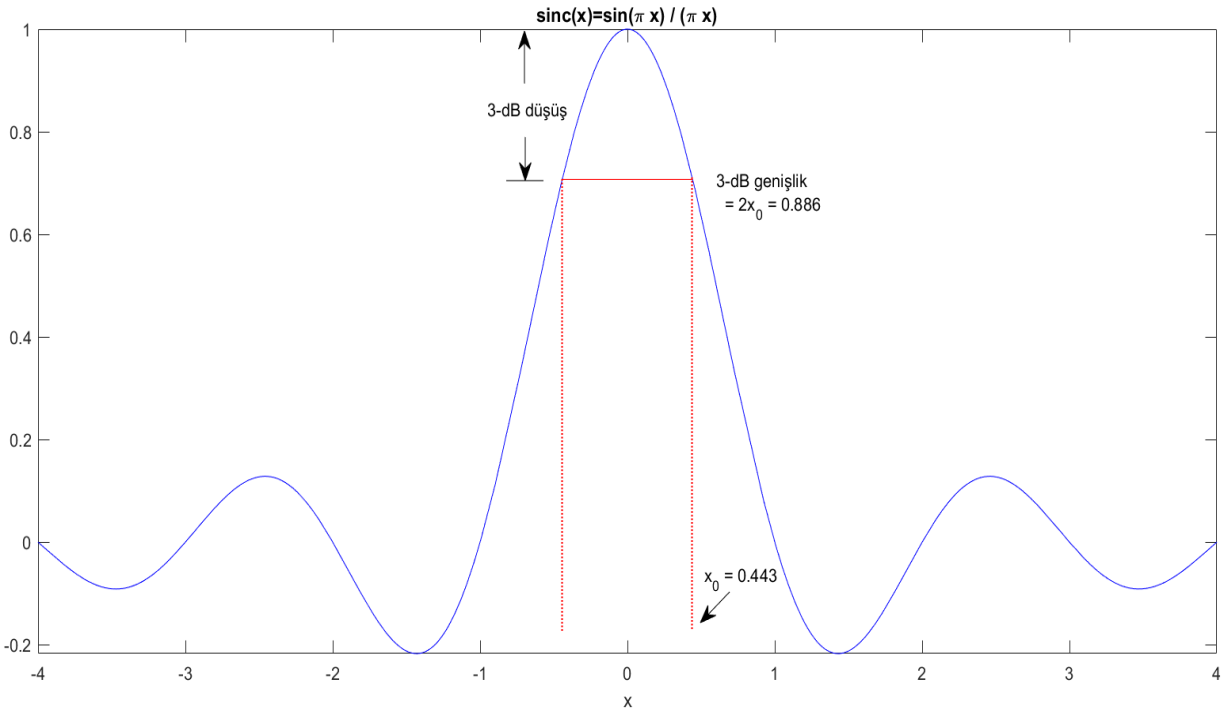
x0 denklemin çözümüdür:

$$\left[\frac{\sin(\pi x)}{\pi x}\right]^2 = \frac{1}{2}$$

ya da,

$$\mathbf{sinc(x)} = \frac{1}{\sqrt{2}}$$

$$10\log_{10}(1/2) = -3\mathbf{dB}$$



Çoklu giriş – çoklu çıkış fonksiyonları

Genel olarak, bir fonksiyon birkaç değişkeni girdi argümanı olarak kabul edebilir ve çıktı olarak birkaç değişken üretebilir.

Girdi argümanları virgüllerle ayrılır ve çıktı değişkenleri parantez içinde listelenir ve farklı boyut ve türlerde olabilir:

```
[out1, out2, ...] = funct(in1, in2, ...)
```

Girdi ve çıktı değişkenlerinin sayısı, ayrılmış değişkenler tarafından sayılır: nargin, nargout

Fonksiyonlar ayrıca varargin, varargout tarafından kontrol edilen girdi ve çıktı değişken sayılarına sahip olabilir.

Örnek: h_0 **yüksekliğinden** v_0 **başlangıç hızıyla yatayda** θ_0 **(derece cinsinden) açı yapacak şekilde düşey yerçekimi ivmesi** g **ile bırakılan ve** t **zamanına bağlı bir vektör doğrultusunda hareket eden kurşunun** x, y **koordinatlarını ve** v_x, v_y **hızlarını hesaplayın.**

```
[x,y,vx,vy] = trajectory(t,v0,th0,h0,g);
```

Hareket denklemleri:

$$x = v_0 \cos\theta_0 t$$

$$y = h_0 + v_0 \sin\theta_0 t - \frac{1}{2} g t^2$$

$$v_x = v_0 \cos\theta_0$$

$$v_y = v_0 \sin\theta_0 - gt$$

trajectory fonksiyonu şu yollarla çağrılabilir:

```
[x,y,vx,vy] = trajectory(t,v0);
[x,y,vx,vy] = trajectory(t,v0,th0);
[x,y,vx,vy] = trajectory(t,v0,th0,h0);
[x,y,vx,vy] = trajectory(t,v0,th0,h0,g);
[~,y,~,vy]  = trajectory(t,v0,th0,h0,g);
```

kullanılmayan çıktı argümanları için

```
      x = trajectory(t,v0,th0,h0);
   [x,y] = trajectory(t,v0,th0,h0);
[x,y,vx] = trajectory(t,v0,th0,h0);
```

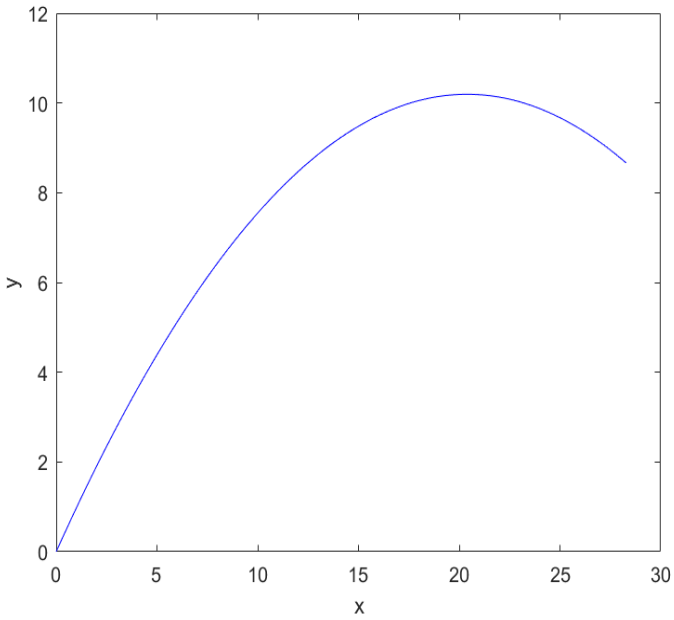
Eğer girilmezse, varsayılan giriş değerleri şöyle olmalıdır:

```
th0 = 90;          % düşey fırlatma, derece
 h0 = 0;           % yer seviyesi
  g = 9.81;        % m/sn^2
```

Sadece listelenen çıktı değişkenleri gösterilir.

```
function [x,y,vx,vy] = trajectory(t,v0,th0,h0,g);

if nargin<=4, g=9.81; end          % varsayılan değerler
if nargin<=3, h0=0; end
if nargin==2, th0=90; end

th0 = th0 * pi/180;        % th0'ı radyana çevirir.

x = v0*cos(th0)*t;
y = h0 + v0*sin(th0)*t - (1/2)*g*t.^2;
vx = v0*cos(th0);
vy = v0*sin(th0) - g*t;
```

```
t = linspace(0,2,201);
v0 = 20; th0=45;
[x,y]=trajectory(t,v0,th0);
plot(x,y,'b');
xlabel('x'); ylabel('y');
```



Alt fonksiyonlar ve içiçe fonksiyonlar

Bir fonksiyon, alt fonksiyonlar olarak adlandırılan diğer fonksiyonların tanımlarını içerebilir.

Alt fonksiyonlar herhangi bir sırada görünebilir ve her biri birincil fonksiyon içindeki diğerlerinin herhangi biri tarafından çağrılabilir.

Her bir alt fonksiyonun diğer alt fonksiyonlar veya birincil fonksiyon tarafından paylaşılmayan kendi çalışma alanı (workspace) vardır, yani her bir alt fonksiyon sadece çıktı değişkenleri ile iletişim kurar.

İçiçe fonksiyonlar çalışma alanı değişkenlerini birincil fonksiyonunkiler ile paylaşır. Hepsi anahtar kelime end ile bitmelidir, yani birisinde end kullanılıyorsa diğerlerinde de kullanılmalıdır.

Örnek

```
% rms.m 'in alternatif versiyonu

function [r,m] = rms(x)
    r = rmsq(x); % karekök ortalama
    m = mav(x);  % ortalama mutlak değer

function y = rmsq(x)
    y = sqrt(sum(abs(x).^2))/length(x);

function y = mav(x)
    y = sum(abs(x))/length(x);
```

Her bir function anahtar kelimesinin görünümü her bir alt fonksiyonun başlangıcını gösterir.

Örnek

```
% rms.m 'in içiçe versiyonu

function [r,m] = rms(x)
    N = length(x);
    r = rmsq(x);    % karekök ortalama
    m = mav(x);     % ortalama mutlak değer

function y = rmsq(x)
    y = sqrt(sum(abs(x).^2))/N;
end                 % rmsq 'nun sonu

function y = mav(x)
    y = sum(abs(x))/N;
end                 % mav 'ın sonu

end                 % rms 'nin sonu
```

N, içiçe fonksiyonlarda bilinir.

Örnek: Ödev raporları için önerilen yapı

```
function set1
    problem1
    problem2
    problem3
function problem1
    ...
function problem2
    ...
function problem3
    ...
function other1
    ...
function other2
    ...
function other3
    ...
```

Birincil fonksiyon

Ödev sonuçlarını almak için problem alt fonksiyonlarını çalıştırır.

Her problemi uygulayan problem alt fonksiyonlarını tanımlar.

Problem alt fonksiyonları tarafından çağrılabilecek diğer alt fonksiyonları tanımlar.

Fonksiyon tiplerinin özeti

- Birincil fonksiyonlar
- Anonim fonksiyonlar
- Alt fonksiyonlar
- İçiçe fonksiyonlar
- Özel fonksiyonlar
- Aşırı yüklenmiş (overloaded) fonksiyonlar
- Tekrarlı (recursive) fonksiyonlar

Tekrarlı fonksiyonlar

Tekrarlı fonksiyonlar kendisini çağırır, yani kendilerini, kendilerini çağırarak tanımlarlar.

Göründüğü kadar döngüsel değil.
(örneğin, uzun bir kişi uzun olan birisini tanımlar.)

İlginç ve şık bir programlama konseptidir, fakat çok yavaş çalışma eğilimindedir.
(C/C++ ve Java gibi diğer dillerde de mevcuttur.)

Fraktal oluşturmak gibi tekrarlayan işler için çok uygundur.

Örnek 1: Fibonacci sayıları, $f(n) = f(n-1) + f(n-2)$

```
function y = fib(n,c)

if n==1, y = c(1); end
if n==2, y = c(2); end

if n>=3,
   y = fib(n-1,c) + fib(n-2,c);
end
```

```
Başlangıç değerleri:

f(1)=c(1);
f(2)=c(2);

C=[c(1),c(2)];
```

```
y = []; c = [0,1];
for n=1:10,
   y = [y,fib(n,c)];
end
y =
   0   1   1   2   3   5   8   13   21   34
```

Örnek 2: Binom katsayıları, `nchoosek(n,k)`

```
function C = bincoeff(n,k)

if (k==0)|(k==n),    % n>=0, k>=0 varsayılır.
    C = 1;
elseif k>n,
    C = 0;
else
    C = bincoeff(n-1,k) + bincoeff(n-1,k-1);
end
```

$$C(n,k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

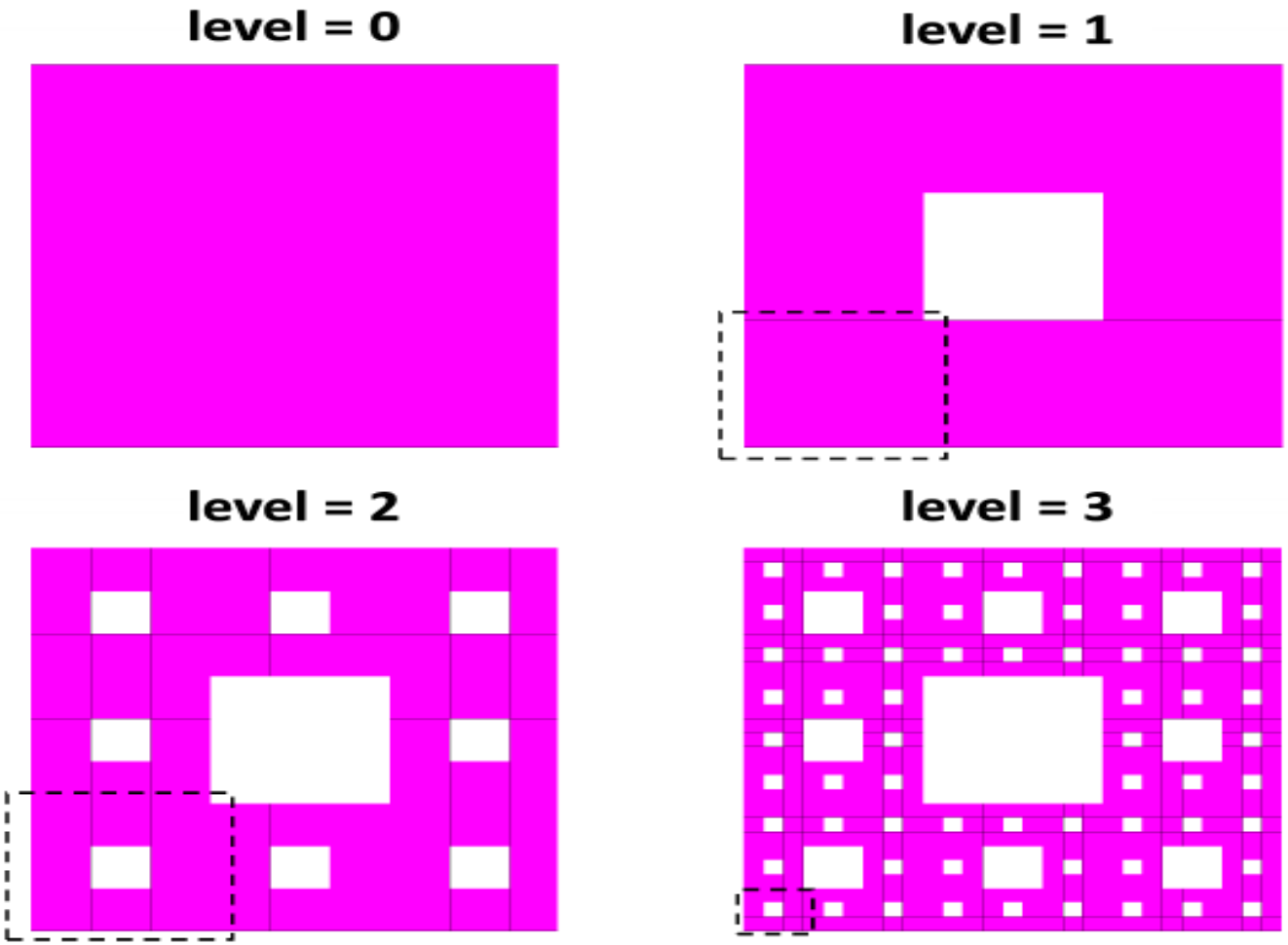
$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

```
for n=0:6,
  C=[];
  for k=0:n,
     C = [C,bincoeff(n,k)];
  end
  disp(C);
end
```

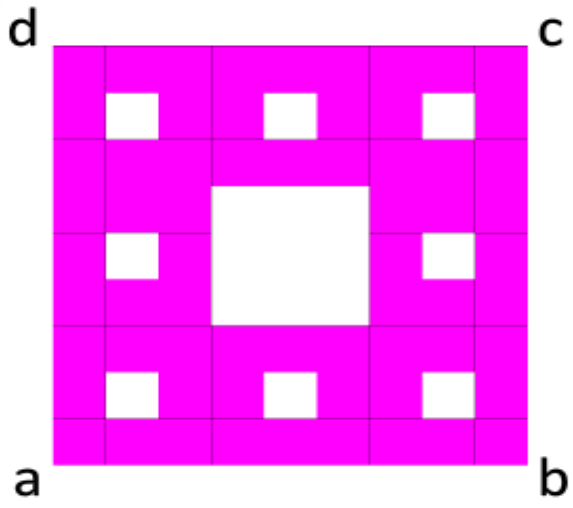
```
 1
 1     1
 1     2     1
 1     3     3     1
 1     4     6     4     1
 1     5    10    10     5     1
 1     6    15    15    15     6     1
```

Pascal üçgeni

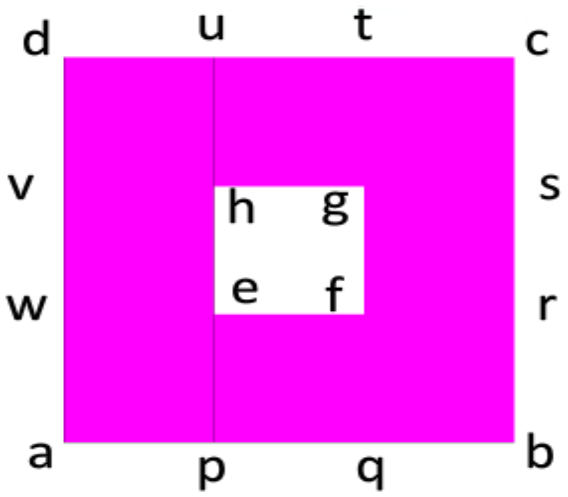
Örnek 3: Sierpinsky halısı



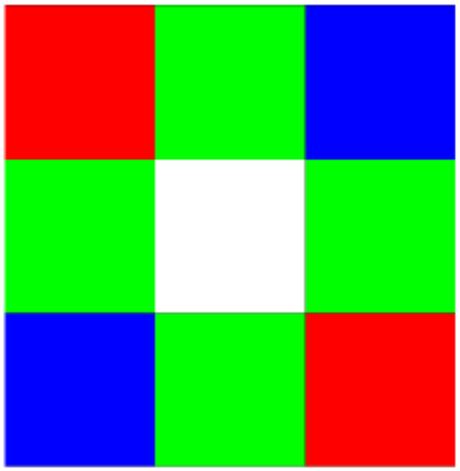
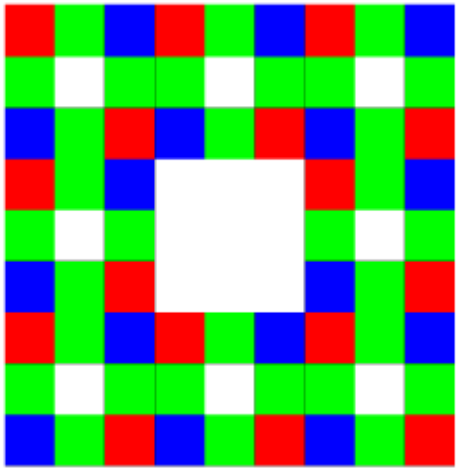
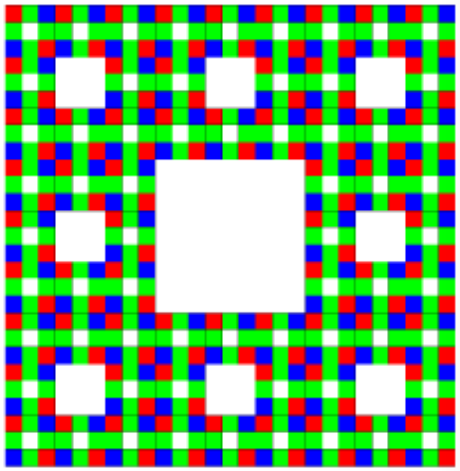
```
level = 2;

a = [0,0]; b = [1,0]; c = [1,1]; d = [0,1];

carpet(a,b,c,d,level);

axis equal; axis off;
```



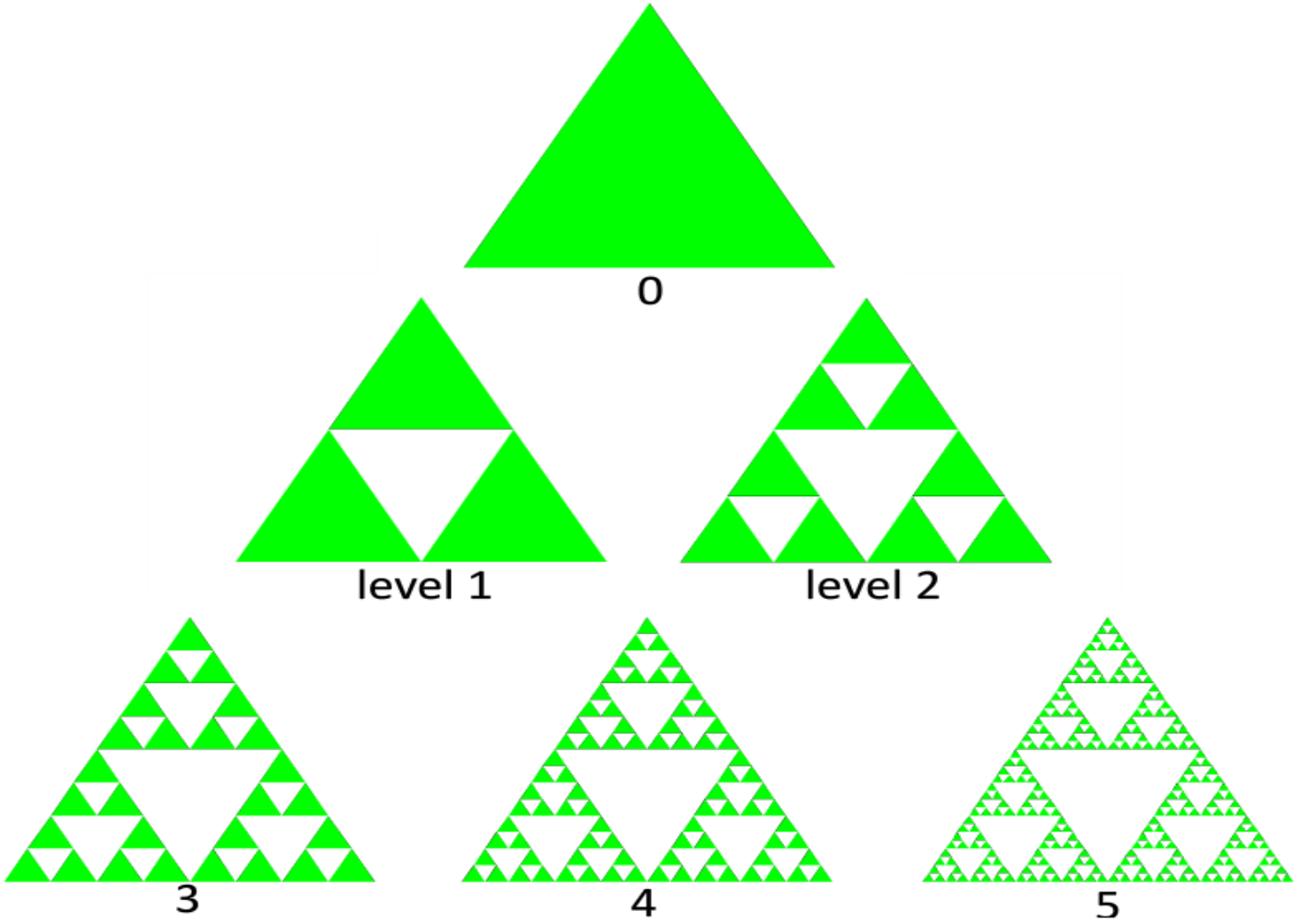
```
function carpet(a,b,c,d,level)

p = (2*a+b)/3; q = (a+2*b)/3;
r = (2*b+c)/3; s = (b+2*c)/3;
t = (2*c+d)/3; u = (c+2*d)/3;
v = (2*d+a)/3; w = (d+2*a)/3;

e = (2*w+r)/3; f = (w+2*r)/3;
g = (2*s+v)/3; h = (s+2*v)/3;
```



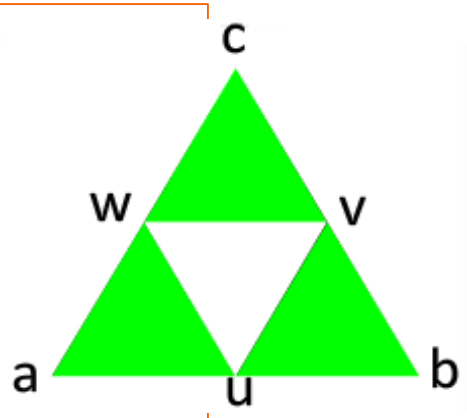

```
if level==0,
   fill([a(1),b(1),c(1),d(1)],[a(2),b(2),c(2),d(2)],'m');
   hold on;
else
   carpet(a,p,e,w, level-1); carpet(p,q,f,e, level-1); %tekrarlı
   carpet(q,b,r,f, level-1); carpet(f,r,s,g, level-1); %çağrılar
   carpet(g,s,c,t, level-1); carpet(h,g,t,u, level-1);
   carpet(v,h,u,d, level-1); carpet(w,e,h,v, level-1);
end
```

Örnek 4: Sierpinsky contası



```
function gasket(a,b,c,level)

u = (a+b)/2;
v = (b+c)/2;
w = (c+a)/2;

if level==0,
   fill([a(1),b(1),c(1)],[a(2),b(2),c(2)],'g');
   hold on;
else
   gasket(a,u,w, level-1); carpet(p,q,f,e, level-1);
   gasket(u,b,v, level-1); carpet(f,r,s,g, level-1);
   gasket(w,v,c, level-1); carpet(h,g,t,u, level-1);
end
```




```
level = 2;
a = [0,0]; b = [1,0]; c = [1,sqrt(3)]/2;
gasket(a,b,c,level);
axis equal; axis off;
```