Uygulama Notları: 9

FİZ219 - Bilgisayar Programlama I | 23/12/2019

Fonksiyonlar II: Eğik Atış Problemi

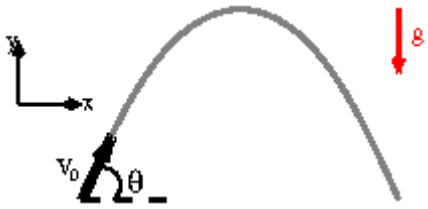
- Problemin Tanımı ve Uçuş Süresinin Hesabı
  - Analitik çözüm
  - Sayısal çözüm: yarılama metodu (bisection)
  - Hazır gelen kök bulma fonksiyonu: roots()
- Konumların hesaplanması
  - Fonksiyonlaştırmak
- Bonus: Güzel grafikler, cici grafikler (Ek bilgi, ilgilenirseniz diye)
- "Meydan okumalar" (*Challenges*)
  - İleri düzey fonksiyonlaştırmak
  - Korsan gemisini vuran top

Emre S. Tasci emre.tasci@hacettepe.edu.tr (mailto:emre.tasci@hacettepe.edu.tr)

Eğik Atış Problemi

Problemin Tanımı ve Uçuş Süresinin Hesabı

v_0 ilk hızı ve yatayla θ açısı yapacak şekilde atılan m kütleli bir cismin üzerine düşey yönde, -y yönünde $F_g = mg$ yerçekimi kuvveti etki ederken, yatay yönde etki eden herhangi bir kuvvet bulunmamaktadır.



Bu durumda, $\vec{r(t)} = \vec{r_0} + \vec{v_0}t + \frac{1}{2}\vec{a}t^2$ genel hareket denklemi, *x_ ve _y* eksenleri yönlerinde bileşenlerine ayrıldığında:

$$v_{0,x} = v_0 \cos\theta$$
$$v_{0,y} = v_0 \sin\theta$$
$$x_0 = y_0 = 0$$
$$a_x = 0$$
$$a_y = -g$$

$$x(t) = x_0 + v_{0,x}t + \frac{1}{2}a_x t^2$$
$$\rightarrow x(t) = v_{0,x}t = v_0 \cos\theta t$$

$$y(t) = y_0 + v_{0,y}t + \frac{1}{2}a_y t^2$$
$$\rightarrow y(t) = v_{0,y}t + \frac{1}{2}(-g)t^2 = v_0 \sin\theta t - \frac{1}{2}gt^2$$

olarak bulunur.

Cismin toplam uçuş zamanı t_d, atıldıktan sonra *y* değerinin 0 olduğu andır. Bu anı bulmak için:

$$y(t_d) = v_0 \sin\theta t_d - \tfrac{1}{2} g t_d^2 = 0$$

denklemini çözeriz.

Problemimizde parametrelerimizi: $v_0 = 30\,\mathrm{m/s}$, $\theta = 60^o$, $g = -9.81\,\mathrm{m/s^2}$ olarak alalım.

Analitik çözüm:

İkinci dereceden $ax^2 + bx + c = 0$ denkleminin çözümü:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

şeklinde bulunur. Katsayıları elimizdeki t_d için uygularsak:

$$a = -\tfrac{1}{2} g,\ b = v_0 \sin\theta,\ c = 0$$

$$t_{d_{1,2}} = \frac{-v_0 \sin\theta \pm \sqrt{(v_0 \sin\theta)^2 - 4(-\frac{1}{2}g)(0)}}{2(-\frac{1}{2}g)} = \frac{-v_0 \sin\theta \pm v_0 \sin\theta}{-g} = 0, \frac{2v_0 \sin\theta}{g}$$

İlk kök ($t_d = 0$) zaten atış anına geldiğinden bariz; bizi ilgilendiren ikinci, $t_d = \frac{2v_0 \sin\theta}{g}$ kökü. (x, y)'nin zamana göre değerlerini hesaplarken aralığımızı sıfırdan bu ana kadar alacağız.

Problemimize özel değerleri yerine koyduğumuzda:

$$t_d = \frac{2v_0 \sin\theta}{g} = \frac{2 \cdot 30 \sin(60)}{9.81}\ \mathrm{s}$$

In [1]:

```
v0 = 30;
theta = 60;
g = 9.81;
t_d = 2*v0*sind(theta)/g
```

```
t_d =  5.2968
```

saniye olarak bulunur. Hatta daha da hassas yazdıralım:

In [2]:

```
printf("Toplam ucus zamani: %9.6f s\n",t_d)
```

```
Toplam ucus zamani:  5.296791 s
```

Sayısal çözüm: yarılama metodu (*bisection*)

Sürekli bir f(x) fonksiyonun verilen belirli bir [a,b] aralığında işaret değiştirdiğini biliyorsak, o halde fonksiyonun o aralıkta en az bir kere yatay ekseni kestiğinin, yani y'nin en az bir kere sıfır değerini aldığının, *yani* en az bir tane kökü olduğunun **garantisi vardır**. O halde yapabileceğimiz bir şey, a ile b'nin tam ortasındaki c değerinde fonksiyonun işaretine bakmak, eğer f(a) ile f(c) aynı işaretse bu sefer kökü aramamıza c ile b arasında devam etmek (bu sefer c'ye a deyip, b'yi aynı tutup, yeni c'yi bu yeni a (*eski c*) ile b'nin orta noktası almak); f(a) ile f(c) zıt işaretliyse, arayışımızı a ile c arasına odaklamak suretiyle gerçek köke istediğimiz hassasiyette yaklaşabiliriz.

Önce, y yönündeki hareket denklemini fonksiyon olarak tanımlayalım:

In [3]:

```
function f=atis_y(v0,theta,t,g=9.81)
f = v0*sind(theta)*t - 0.5*g*t^2;
endfunction

v0 = 30;
theta = 60;

% atıldıktan az sonraki (t=0.01s) yüksekliğe bakalım:
a = 0.01;
printf("atildiktan %6.3f saniye sonraki yukseklik: %10.5f m.\n",a,atis_y(v0,theta,a

% bir de 10 saniye sonraki yüksekliğe bakalım:
b = 10;
printf("atildiktan %6.3f saniye sonraki yukseklik: %10.5f m.\n",b,atis_y(v0,theta,b
```

```
atildiktan  0.010 saniye sonraki yukseklik:    0.25932 m.
atildiktan 10.000 saniye sonraki yukseklik: -230.69238 m.
```

Görüldüğü üzere, 10 saniye sonra yükseklik -230.7 olarak verildiğinden, 0 ile 10. saniyeler arasında bir yerde cismimiz yatay eksenden geçmiş. Peki ama nerede?

In [4]:

```
% 0.01. ile  10 saniyenin tam ortasındaki anda ne yükseklikte idi?
c = (a + b) / 2;
printf("atildiktan %6.3f saniye sonraki yukseklik: %10.5f m.\n",c,atis_y(v0,theta,c
```

```
atildiktan  5.005 saniye sonraki yukseklik:    7.16334 m.
```

Atıldıktan 5.005 saniye sonra yüksekliği pozitif; demek ki cisim hala yer seviyesinin üzerindeymiş. Bu durumda "yere çarpış", 5.005 ile 10. saniye arasında bir anda yer alıyor. Bu durumda biraz önce a=0.01s, b=10s için yaptığımız hesabı şimdi a=5.005s ile b=10s için yapacağız.. Bunu sistematiğe bağlarsak:

In [5]:

```
clear;

function f=atis_y(v0,theta,t,g=9.81)
f = v0*sind(theta)*t - 0.5*g*t^2;
endfunction

v0 = 30;
theta = 60;

istenilen_hassasiyet = 0.00001;

a = 0.01;
f_a = atis_y(v0,theta,a);

b = 10;
f_b = atis_y(v0,theta,b);

adim = 1;
while(abs(a-b)>istenilen_hassasiyet)
    % arama araliginin boyu |a-b|, istenilen hassasiyetten
    % buyuk oldugu surece, yarilamaya devam ediyoruz
    c = (a+b)/2;
    f_c = atis_y(v0,theta,c);
    printf("%2d: %9.6f  %10.5f | %9.6f  %10.5f | %9.6f  %10.5f\n",adim,a,f_a,c,f_c,
    if(sign(f_c) == sign(f_a))
        % c ile a'nın yükseklikleri aynı işaretli, o zaman
        % arada kök olmasının garantisi yok, arayışa c ile b
        % arasında devam edelim
        a = c;
        f_a = f_c;
    else
        % c ile b'nin yükseklikleri aynı işaretli, o zaman
        % arada kök olmasının garantisi yok, arayışa a ile c
        % arasında devam edelim
        b = c;
        f_b = f_c;
    endif
    adim = adim + 1;
endwhile
```

```
 1:  0.010000     0.25932 |  5.005000     7.16334 | 10.000000  -230.69
238
 2:  5.005000     7.16334 |  7.502500   -81.16955 | 10.000000  -230.69
238
 3:  5.005000     7.16334 |  6.253750   -29.35436 |  7.502500   -81.16
955
 4:  5.005000     7.16334 |  5.629375    -9.18332 |  6.253750   -29.35
436
 5:  5.005000     7.16334 |  5.317187    -0.53195 |  5.629375    -9.18
332
 6:  5.005000     7.16334 |  5.161094     3.43521 |  5.317187    -0.53
195
 7:  5.161094     3.43521 |  5.239141     1.48151 |  5.317187    -0.53
195
 8:  5.239141     1.48151 |  5.278164     0.48225 |  5.317187    -0.53
195
 9:  5.278164     0.48225 |  5.297676    -0.02298 |  5.317187    -0.53
195
10:  5.278164     0.48225 |  5.287920     0.23010 |  5.297676    -0.02
```

```
298
11:  5.287920     0.23010 |  5.292798     0.10368 |  5.297676    -0.02
298
12:  5.292798     0.10368 |  5.295237     0.04038 |  5.297676    -0.02
298
13:  5.295237     0.04038 |  5.296456     0.00871 |  5.297676    -0.02
298
14:  5.296456     0.00871 |  5.297066    -0.00713 |  5.297676    -0.02
298
15:  5.296456     0.00871 |  5.296761     0.00079 |  5.297066    -0.00
713
16:  5.296761     0.00079 |  5.296914    -0.00317 |  5.297066    -0.00
713
17:  5.296761     0.00079 |  5.296837    -0.00119 |  5.296914    -0.00
317
18:  5.296761     0.00079 |  5.296799    -0.00020 |  5.296837    -0.00
119
19:  5.296761     0.00079 |  5.296780     0.00029 |  5.296799    -0.00
020
20:  5.296780     0.00029 |  5.296790     0.00004 |  5.296799    -0.00
020
```

Görüldüğü üzere, bu şekilde, gerçek (analitik) değere verilen hassasiyet mertebesinde yaklaşmış olduk.

Hazır gelen kök bulma fonksiyonu: roots()

Bir polinom denkleminin köklerini `roots()` komutu ile hesaplayabiliriz. Fonksiyonun parametreleri en yüksek dereceli terimin katsayısından başlayarak, sabit katsayıya kadar yazılmış katsayılar vektörüdür. Örneğin: $3x^3 - 4x + 7 = 0$ denkleminin kökünü bulmak için:

In [6]:

```
roots([3 0 -4 1])
```

```
ans =

  -1.26376
   1.00000
   0.26376
```

yazarız (burada x^2 olmadığı için, onun katsayısını 0 aldığımıza dikkat edin).

Problemimizdeki ikinci dereceden parabol denklemimiz:

$$y(t_d) = v_0 \sin\theta t_d - \tfrac{1}{2} g t_d^2 = 0$$

idi. Katsayıları roots() fonksiyonunda yerine koyacak olursak:

In [7]:

```
v0 = 30;
theta = 60;
g = 9.81;
roots([-0.5*g v0*sind(theta) 0])
```

```
ans =

   5.29679
   0.00000
```

şeklinde doğrudan kökü bulmuş oluruz. (Bir de, polinom olsun olmasın, en uygun metotla kök bulan `fzero()` fonksiyonu var ama şimdilik onu bir kenarda bırakıyoruz)

Konumların hesaplanması

Artık uçuş süresini bildiğimize göre, bu uçuş süresi boyunca cismimizin koordinatlarını hareket denklemlerinden hesaplayabilir, ardından grafiğini de çizdirebiliriz.

Zaman aralığını $[0, t_d]$ arasında, başlangıç için 5 nokta alıp inceleyelim, bunları yaparken, sıfırdan başlayıp, toparlayalım:

In [8]:

```
clear;

function f=atis_y(v0,theta,t,g=9.81)
f = v0*sind(theta)*t - 0.5*g*t^2;
endfunction

v0 = 30;
theta = 60;
g = 9.81;

t_d = 2*v0*sind(theta)/g

N = 5;

t_ler = linspace(0,t_d,N)

y_konumlari = [];

for t = t_ler
    y_konumlari = [y_konumlari, atis_y(v0,theta,t)];
endfor

y_konumlari

plot(t_ler,y_konumlari,"-*")
```

```
t_d =  5.2968
t_ler =

   0.00000   1.32420   2.64840   3.97259   5.29679

y_konumlari =

    0.00000   25.80275   34.40367   25.80275    0.00000
```

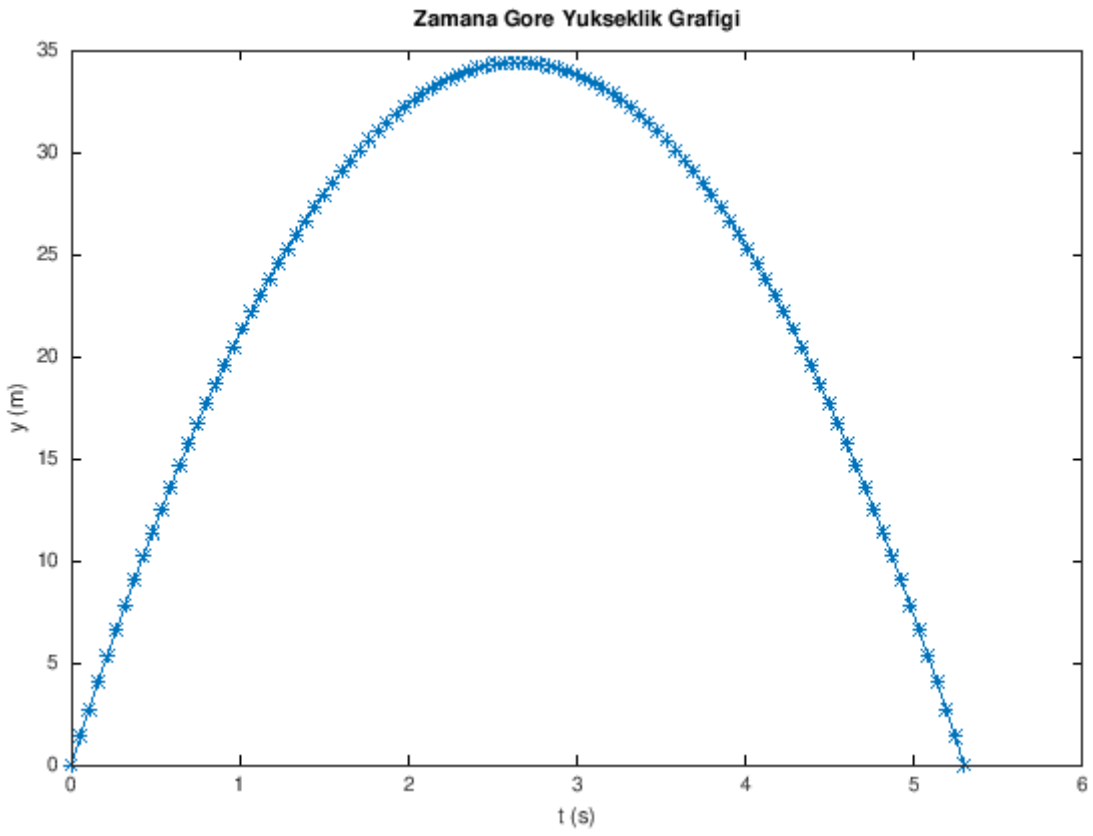
Pek fena görünmüyor, fakat iki şeyi fark ettiyseniz ne mutlu size: programı yazarken bir önceki uygulama notlarında fonksiyonları yazarken yapmamanızı tavsiye ettiğim iki şeyi bizzat ben yapmış durumdayım. Nedir o iki şey?..

1. Fonksiyonu ayrı, kendi dosyasında değil, doğrudan programın içinde tanımladım:
   Bunu yapmak biraz mecburiyetten çünkü içeriğini görün istedim.
2. Oradaki *for...* döngüsü:
   Elimde linspace ile çıkardığım t değerlerini içeren diziyi/listeyi/vektörü doğrudan fonksiyona beslemek varken, gayet masraflı bir şekilde for döngüsüyle tek tek fonksiyona besledim -- bu affedilecek bir şey değil. Ama fonksiyona bakacak olursanız bu haliyle çalışamaz (nedenini görebildiniz mi?.. oradaki t^2 işlemi eleman bazlı çalışmalı, bu yüzden t.^2 olarak yazılmalıydı).

Haydi kodu bir daha, bu sefer daha verimli yazalım, hazır yazarken hesaplanan nokta sayısını da 100 yapalım (hatta oldu olacak, x'in konumlarını da hesaplayalım):

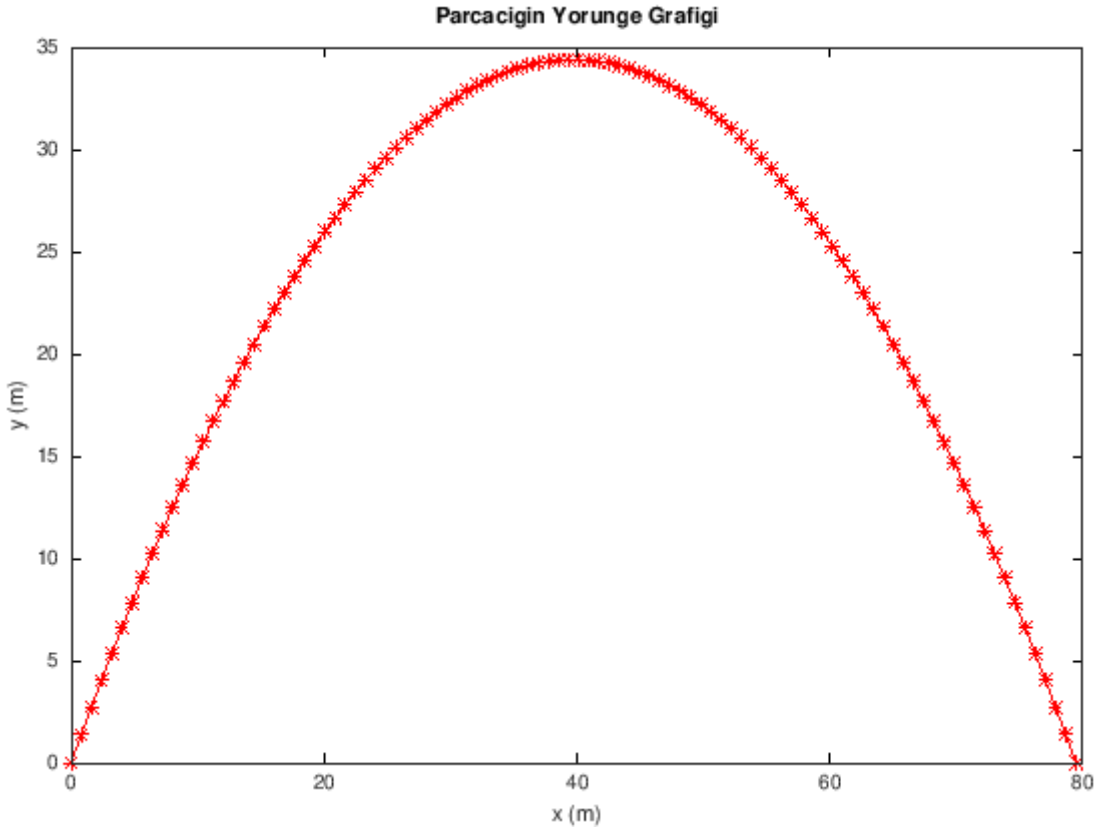
In [9]:

```
clear;

function f=atis_y(v0,theta,t,g=9.81)
f = v0*sind(theta)*t - 0.5*g*t.^2;
endfunction

function f=atis_x(v0,theta,t,g=9.81)
f = v0*cosd(theta)*t;
endfunction

v0 = 30;
theta = 60;
g = 9.81;

t_d = 2*v0*sind(theta)/g;

N = 100;

t_ler = linspace(0,t_d,N);

y_konumlari = atis_y(v0,theta,t_ler);
x_konumlari = atis_x(v0,theta,t_ler);

plot(t_ler,y_konumlari,"-*")
xlabel("t (s)")
ylabel("y (m)")
title("Zamana Gore Yukseklik Grafigi")
```



In [10]:

```
plot(x_konumlari,y_konumlari,"r-*")
xlabel("x (m)")
ylabel("y (m)")
title("Parcacigin Yorunge Grafigi")
```



Fonksiyonlaştırmak

Şimdi öyle bir fonksiyon yazalım ki, ona ilk hızımızı ve atış açımızı (opsiyonel olarak da yerçekimsel ivmeyi) verdiğimizde, bize cismimizin toplamda kaç saniye uçtuğunu, ulaştığı maksimum yüksekliği ve yatay yönde kat ettiği menzili söylesin.

In [11]:

```
clear;

function [t_d,h_max,R] = ucus_analiz(v0,theta,g=9.81)
v0_y = v0*sind(theta);
t_d = 2*v0_y/g;
t_d_bolu_2 = t_d /2;
h_max = v0_y*t_d_bolu_2 - 0.5*g*t_d_bolu_2^2;
R = v0*cosd(theta) * t_d;
endfunction

ilk_hiz = 30;
aci = 60;
[ucus_zamani,max_yukseklik,menzil] = ucus_analiz(ilk_hiz,aci);
printf("Ucus zamani: %7.4fs | Maksimum Yukseklik: %6.3fm | Menzil: %6.3fm\n",ucus_z
```

```
Ucus zamani:  5.2968s | Maksimum Yukseklik: 34.404m | Menzil: 79.452m
```

ara vermeden, aynı ilk hız için hangi açıda en uzun menzile ulaşacak bulalım:

In [12]:

```
ucus_zamanlari = [];
max_yukseklikler = [];
menziller = [];

acilar = 1:90;
for aci = acilar
    [ucus_zamani,max_yukseklik,menzil] = ucus_analiz(ilk_hiz,aci);
    ucus_zamanlari = [ucus_zamanlari, ucus_zamani];
    max_yukseklikler = [max_yukseklikler, max_yukseklik];
    menziller = [menziller, menzil];
endfor

sonuclar = [acilar' ucus_zamanlari' max_yukseklikler' menziller']
```

```
sonuclar =

    1.00000    0.10674    0.01397    3.20179
    2.00000    0.21345    0.05587    6.39968
    3.00000    0.32010    0.12564    9.58977
    4.00000    0.42665    0.22321   12.76817
    5.00000    0.53306    0.34845   15.93103
    6.00000    0.63932    0.50120   19.07447
    7.00000    0.74538    0.68129   22.19467
    8.00000    0.85121    0.88849   25.28783
    9.00000    0.95679    1.12256   28.35018
   10.00000    1.06207    1.38320   31.37799
   11.00000    1.16703    1.67010   34.36758
   12.00000    1.27163    1.98290   37.31529
   13.00000    1.37585    2.32124   40.21754
   14.00000    1.47964    2.68469   43.07079
   15.00000    1.58299    3.07281   45.87156
   16.00000    1.68586    3.48514   48.61645
   17.00000    1.78821    3.92116   51.30210
   18.00000    1.89001    4.38034   53.92525
   19.00000    1.99124    4.86214   56.48270
   20.00000    2.09187    5.36595   58.97134
   21.00000    2.19185    5.89117   61.38813
   22.00000    2.29117    6.43716   63.73013
   23.00000    2.38979    7.00325   65.99448
   24.00000    2.48769    7.58875   68.17842
   25.00000    2.58482    8.19294   70.27931
   26.00000    2.68117    8.81510   72.29456
   27.00000    2.77670    9.45447   74.22174
   28.00000    2.87139   10.11025   76.05849
   29.00000    2.96520   10.78167   77.80258
   30.00000    3.05810   11.46789   79.45187
   31.00000    3.15008   12.16808   81.00437
   32.00000    3.24110   12.88140   82.45817
   33.00000    3.33113   13.60696   83.81151
   34.00000    3.42014   14.34389   85.06274
   35.00000    3.50811   15.09128   86.21033
   36.00000    3.59502   15.84823   87.25289
   37.00000    3.68083   16.61382   88.18915
   38.00000    3.76551   17.38711   89.01796
   39.00000    3.84905   18.16716   89.73831
   40.00000    3.93142   18.95302   90.34934
   41.00000    4.01259   19.74374   90.85028
   42.00000    4.09254   20.53834   91.24054
   43.00000    4.17124   21.33586   91.51964
```

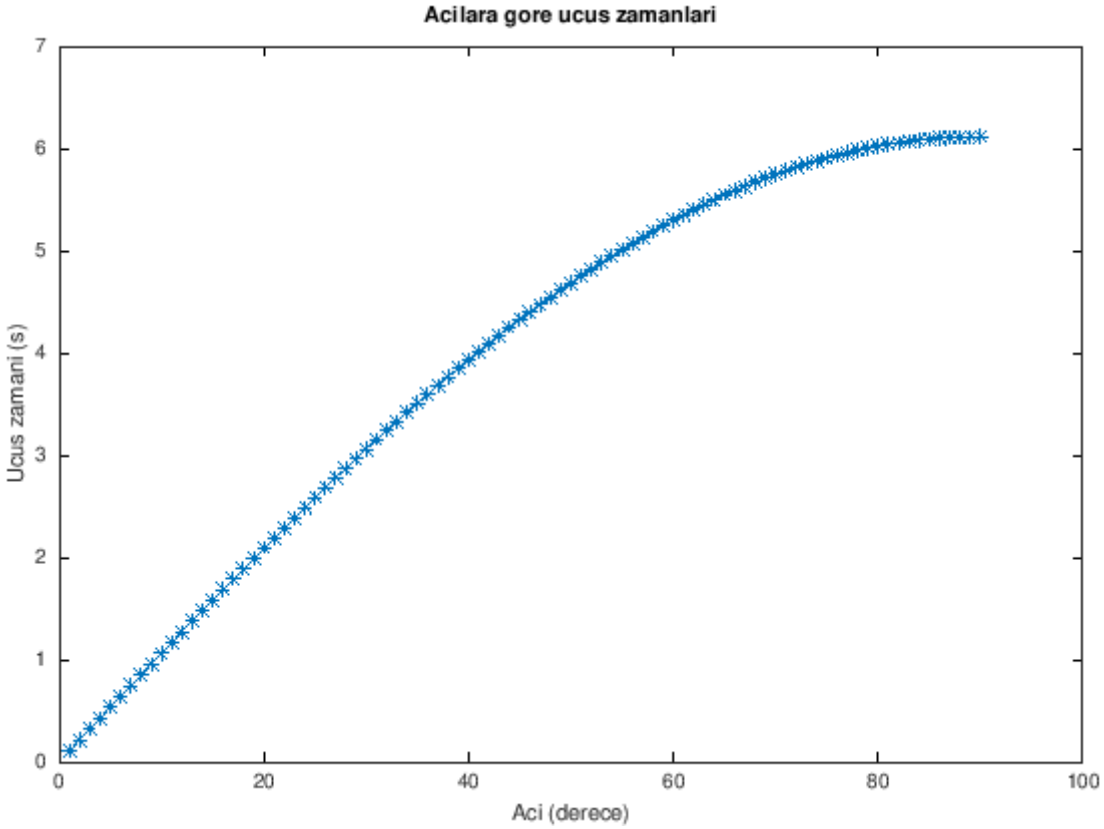
```
44.00000    4.24868   22.13533   91.68723
45.00000    4.32481   22.93578   91.74312
46.00000    4.39963   23.73623   91.68723
47.00000    4.47311   24.53570   91.51964
48.00000    4.54523   25.33322   91.24054
49.00000    4.61596   26.12782   90.85028
50.00000    4.68529   26.91854   90.34934
51.00000    4.75319   27.70440   89.73831
52.00000    4.81964   28.48445   89.01796
53.00000    4.88462   29.25774   88.18915
54.00000    4.94812   30.02333   87.25289
55.00000    5.01010   30.78028   86.21033
56.00000    5.07057   31.52767   85.06274
57.00000    5.12948   32.26460   83.81151
58.00000    5.18684   32.99016   82.45817
59.00000    5.24261   33.70348   81.00437
60.00000    5.29679   34.40367   79.45187
61.00000    5.34936   35.08989   77.80258
62.00000    5.40029   35.76131   76.05849
63.00000    5.44958   36.41709   74.22174
64.00000    5.49721   37.05646   72.29456
65.00000    5.54317   37.67861   70.27931
66.00000    5.58743   38.28281   68.17842
67.00000    5.63000   38.86831   65.99448
68.00000    5.67085   39.43440   63.73013
69.00000    5.70997   39.98039   61.38813
70.00000    5.74736   40.50561   58.97134
71.00000    5.78299   41.00942   56.48270
72.00000    5.81686   41.49122   53.92525
73.00000    5.84896   41.95040   51.30210
74.00000    5.87928   42.38642   48.61645
75.00000    5.90780   42.79875   45.87156
76.00000    5.93453   43.18687   43.07079
77.00000    5.95945   43.55032   40.21754
78.00000    5.98255   43.88866   37.31529
79.00000    6.00384   44.20146   34.36758
80.00000    6.02329   44.48836   31.37799
81.00000    6.04091   44.74900   28.35018
82.00000    6.05669   44.98307   25.28783
83.00000    6.07062   45.19027   22.19467
84.00000    6.08270   45.37036   19.07447
85.00000    6.09293   45.52311   15.93103
86.00000    6.10131   45.64835   12.76817
87.00000    6.10783   45.74592    9.58977
88.00000    6.11248   45.81569    6.39968
89.00000    6.11528   45.85759    3.20179
90.00000    6.11621   45.87156    0.00000
```

Görüldüğü üzere, en uzun uçuş süresine ve en yükseğe tam tepeye atıldığında, $\theta = 90^o$ olduğunda ulaşıyor ($t_d = 6.11621\text{s}$ ve $h_{max} = 45.87156\text{m}$ ile) ama en uzağa 91.74312m ile $\theta = 45^o$ ile atıldığında ulaşıyor ki, Fizik I dersini düşündüğümüzde bu bir sürpriz olmamalı! ;)

Bir de açılara bağlı olarak grafiklerini çizdirip, tamamlayalım:

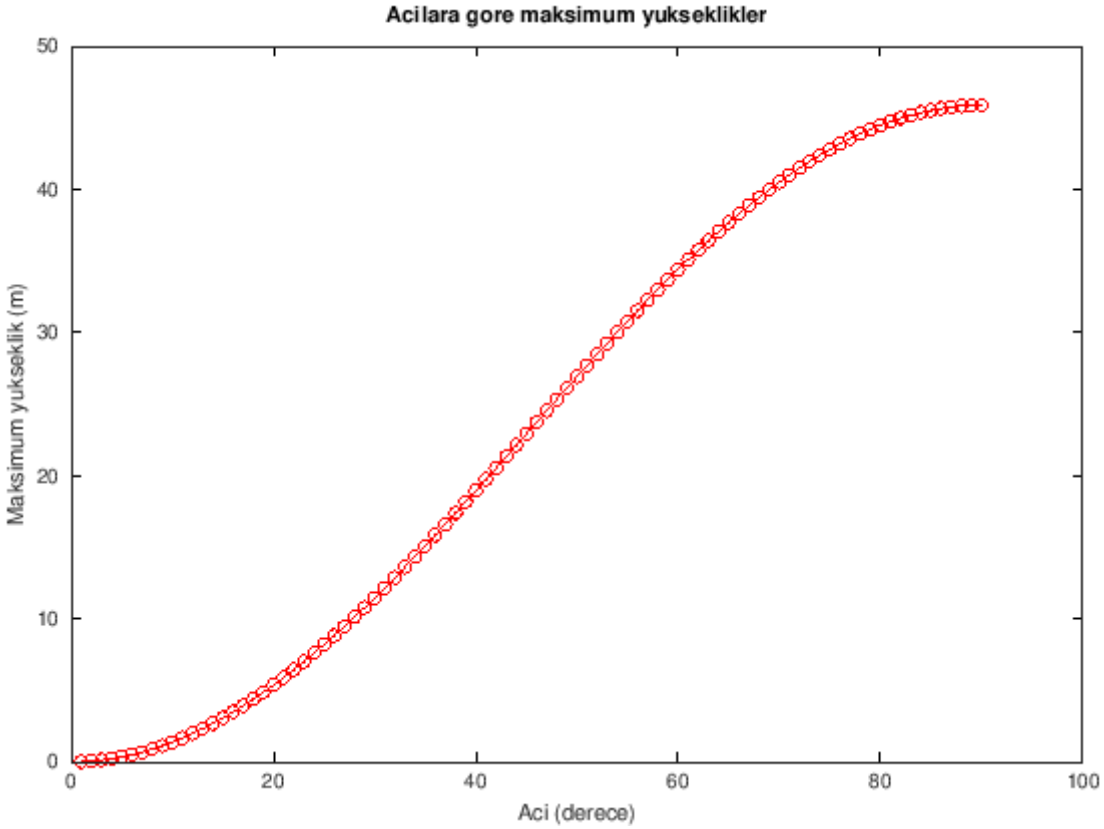
In [13]:

```
plot(acilar,ucus_zamanlari,"-*");
xlabel("Aci (derece)");
ylabel("Ucus zamani (s)");
title("Acilara gore ucus zamanlari");
```



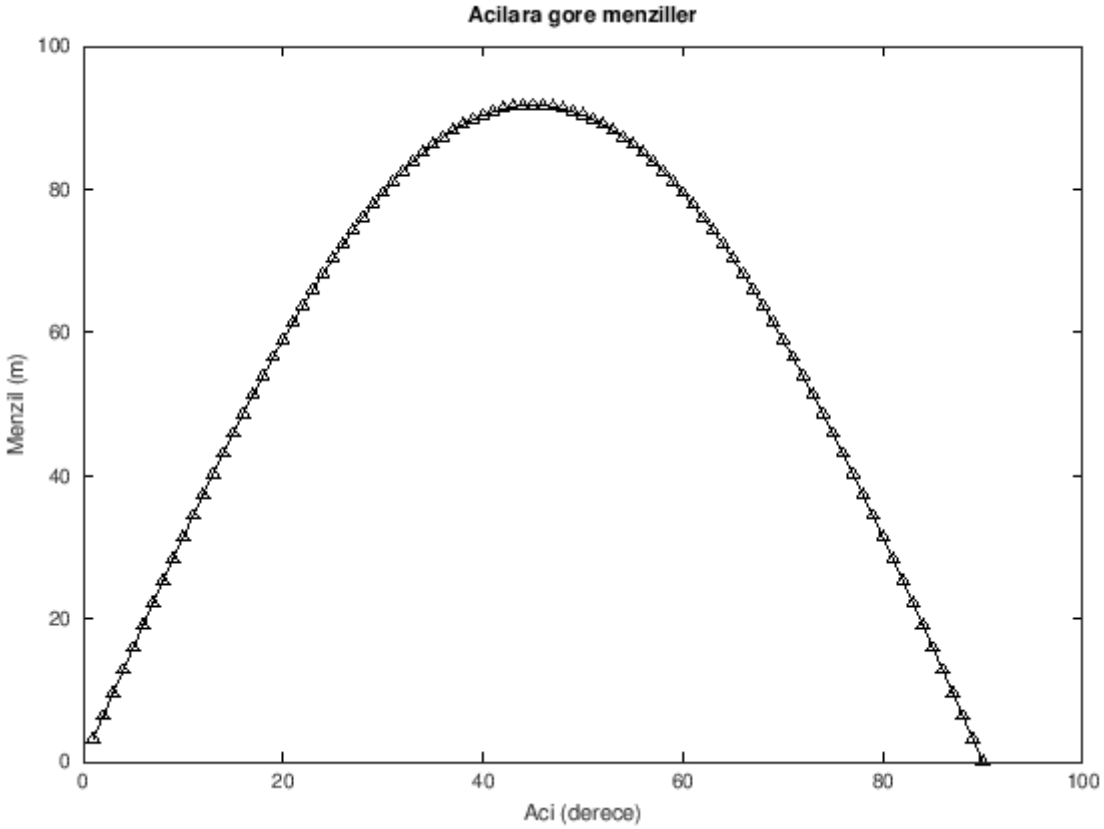
In [14]:

```
plot(acilar,max_yukseklikler,"r-o");
xlabel("Aci (derece)");
ylabel("Maksimum yukseklik (m)");
title("Acilara gore maksimum yukseklikler");
```



In [15]:

```
plot(acilar,menziller,"k-^");
xlabel("Aci (derece)");
ylabel("Menzil (m)");
title("Acilara gore menziller");
```



Bonus: Güzel grafikler, cici grafikler (Ek bilgi, ilgilenirseniz diye)

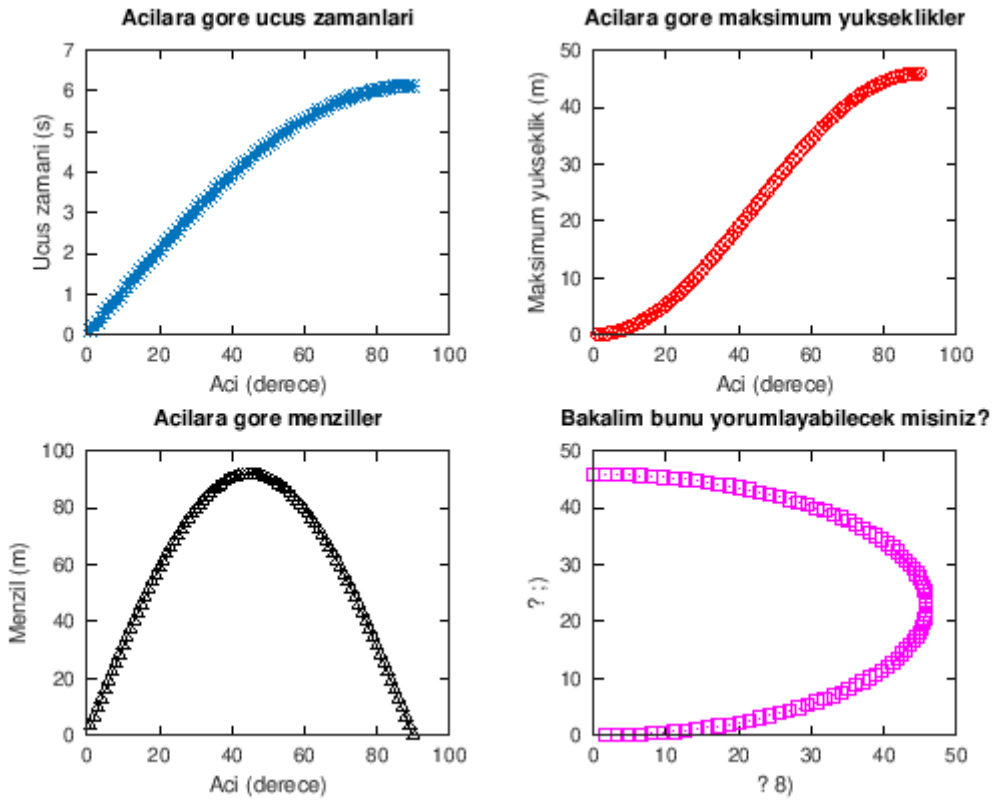
Şimdiye kadar grafiklerimizi hep bir grafik bir panelin içinde ya da iki veya daha fazla grafiği üst üste bindirip, yine tek bir panel içinde olacak şekilde çizegeldik.

Fakat, elimizde birden fazla grafik olduğunda bunları derli toplu yerleştirmek için `subplot()` komutunu kullanıyoruz. Bu komut, panelimizi belirttiğimiz satır ve sütuna bölüyor, sonra soldan sağa, yukarıdan aşağıya giderek bunları numaralandırıyor. Örneğin, subplot(2,3,5) ile kast ettiğimiz panelimizin 2 satır, 3 sütuna bölünüp, alttaki satırın orta sütununu aktif hale getirip, bir sonraki plot() komutumuzun sonucunun oraya çizdirilmesi. 5.'nin orası olduğunu nasıl hesapladım? En üst, en soldakine 1 deyip, soldan sağa, yukarıdan aşağı ilerleyip numaralandırarak:

```
+-----+-----+-----+
|  1  |  2  |  3  |
+-----+-----+-----+
|  4  |  5  |  6  |
+-----+-----+-----+
```

In [16]:

```
subplot(2,2,1);
plot(acilar,ucus_zamanlari,"-*");
xlabel("Aci (derece)");
ylabel("Ucus zamani (s)");
title("Acilara gore ucus zamanlari");

subplot(2,2,2);
plot(acilar,max_yukseklikler,"r-o");
xlabel("Aci (derece)");
ylabel("Maksimum yukseklik (m)");
title("Acilara gore maksimum yukseklikler");

subplot(2,2,3);
plot(acilar,menziller,"k-^");
xlabel("Aci (derece)");
ylabel("Menzil (m)");
title("Acilara gore menziller");

subplot(2,2,4);
plot(menziller/2,max_yukseklikler,"ms");
xlabel("? 8)");
ylabel("? ;)");
title("Bakalim bunu yorumlayabilecek misiniz?");
```



Meydan okumalar (*Challenges*)

Kendini zorlamak isteyenlere iki tane meydan okuma. Ödev değildir, kesinlikle sınavınızdan önce ilgilenmenizi tavsiye etmiyorum, örneğin finallerden sonraki ara tatilde canınız sıkılırsa aklınızda olsun. Çözebilirseniz bana e-posta ile gönderebilirsiniz ama ancak bu dönemin notlarını verdikten sonra (ve o zamandan sonra da hatırlatırsanız!) incelerim. (Özetle demek istediğim, ilginizi çekerse yapmaya çalışın fakat yapın ya da

yapmayın, ders notunuza doğrudan bir katkısı olmayacak, o nedenle eğer ilk iki denemenizde olmuyorsa, finallerin sonrasına bırakmanızı tavsiye ederim -- her hâlükârda notlar teslim edilene kadar kontrol etmeyeceğim 8)

İleri düzey fonksiyonlaştırmak

Yukarıda, "Fonksiyonlaştırmak" başlıklı kısımda, cismi en uzağa taşıyacak açıyı bulmaya çalışırken, açılar üzerinden *for...* döngüsü ile ilerledik. Halbuki bize yakışan:

```
[ucus_zamanlari,max_yukseklikler,menziller] = ucus_analiz(ilk_hiz,acilar);
```

şeklinde, çat! diye, tek bir çağırışta, hiçbir *for...* döngüsü kullanmadan sonuca ulaşabilmek olmalıydı. Fonksiyonu bu şekilde (ve tabii ki fonksiyonun içinde de *for...* kullanmadan yazabilir misiniz?

Korsan gemisini vuran top

Halliday & Resnick'in "Fiziğin Temelleri" kitabında değişik bir örnek vardır: Sahildeki bir top $v_0 = 82\mathrm{m/s}$ ilk hızla gülle atabiliyor. $R = 560\mathrm{m}$ ötedeki bir korsan gemisini vurması için gerekli açıyı sorar. Soru çok ilginç değil ama çözümü sonucunda bir değil, iki doğru açı çıkar.

Bu açıları, analitik yoldan değil de, nümerik yoldan tespit edip, iki atışın yörüngesini (yani x-y grafiklerini) aynı grafik üzerinde çizdirebilir misiniz?