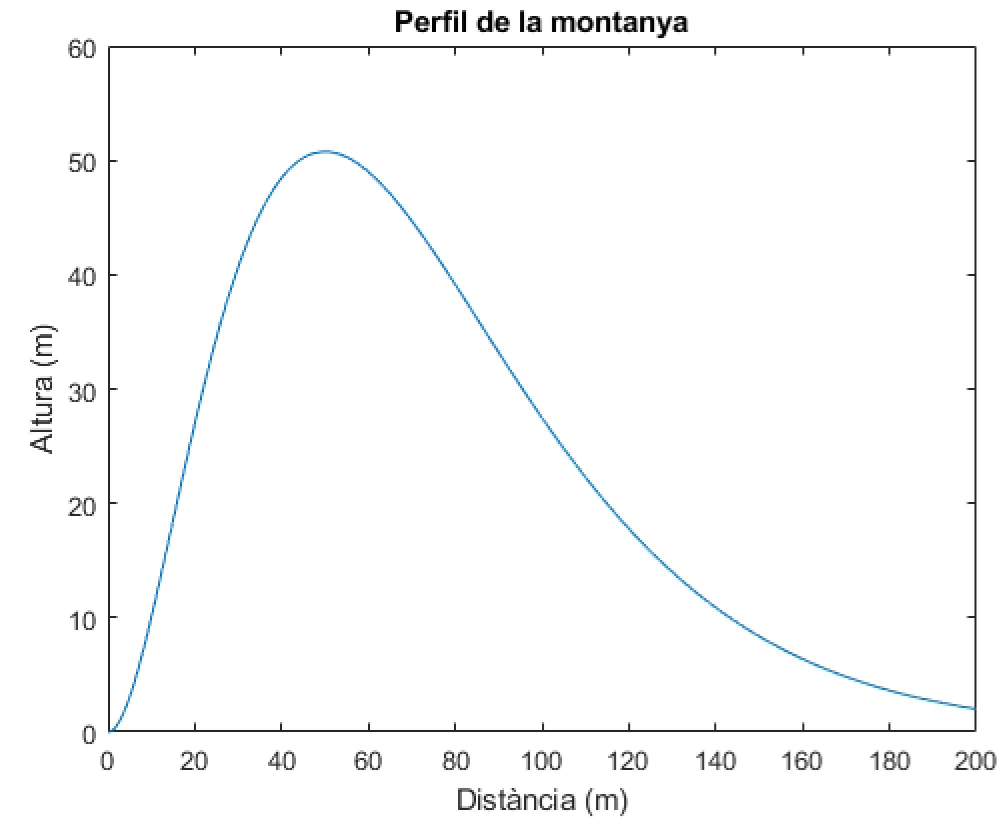
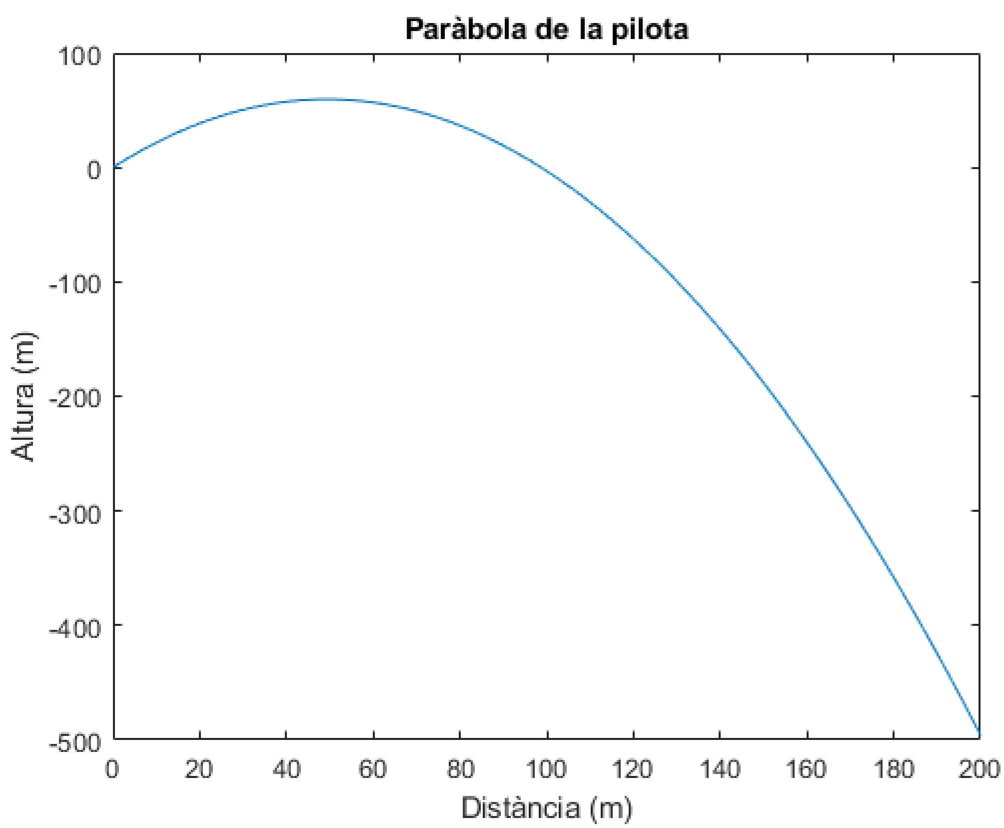
P4: Oleart y Chen

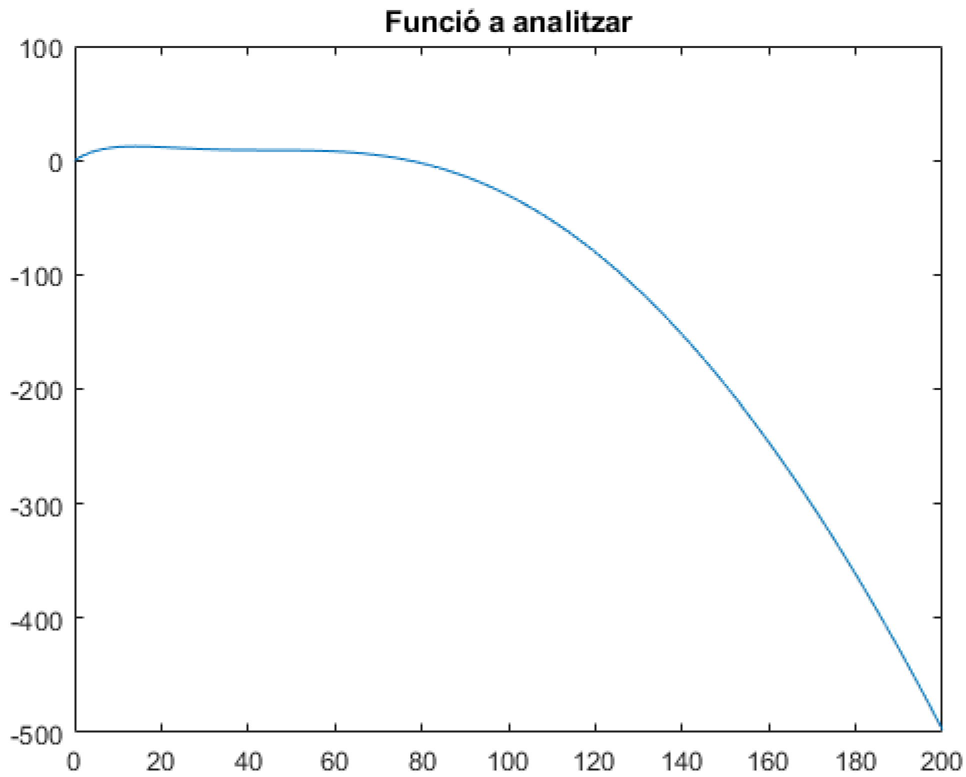
Table of Contents

Apartat a

```
%Paràmetres definits
a = .15;
b = .04;
v = 37;
alpha = 67.5;
% L'angle s'entra en graus, ja s'encarrega la funció de convertir-ho en
% radians

x = (0:1/1000:200);

figure(1)
plot(x, y_mont(x));
title('Perfil de la montanya')
xlabel('Distància (m)')
ylabel('Altura (m)')

figure(2)
plot(x, y_parab(x, alpha));
title('Paràbola de la pilota')
xlabel('Distància (m)')
ylabel('Altura (m)')

figure(3)
plot(x, F(x, alpha));
title('Funció a analitzar')
```


Perfil de la montanya
Altura (m)
Distància (m)
0
10
20
30
40
50
60
0
20
40
60
80
100
120
140
160
180
200


Paràbola de la pilota
Altura (m)
Distància (m)
100
0
-100
-200
-300
-400
-500
0
20
40
60
80
100
120
140
160
180
200



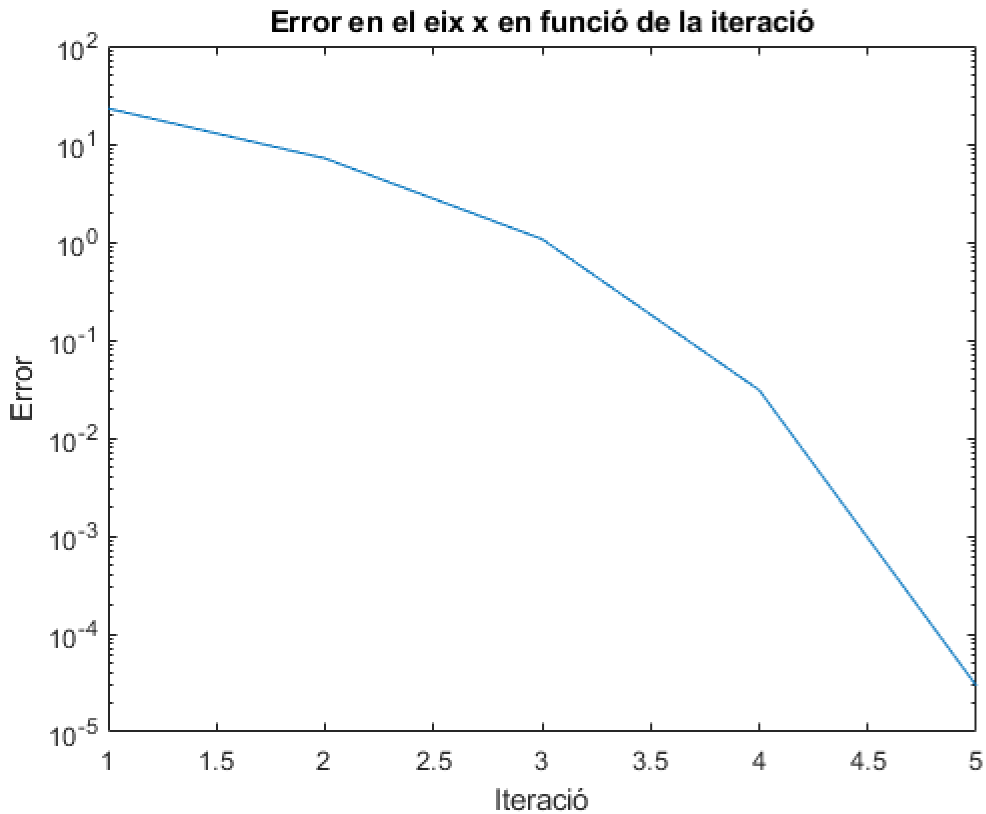
Apartat b

```
[xv, fv, iter] = newton(100, 10^-8, 50, @F, alpha);
disp(['Aproximació a cada iteració: ' num2str(xv)]);
disp(['Error en el eix y a cada iteració: ' num2str(fv)]);
disp(['Nombre de iteracions: ' num2str(iter)])
```

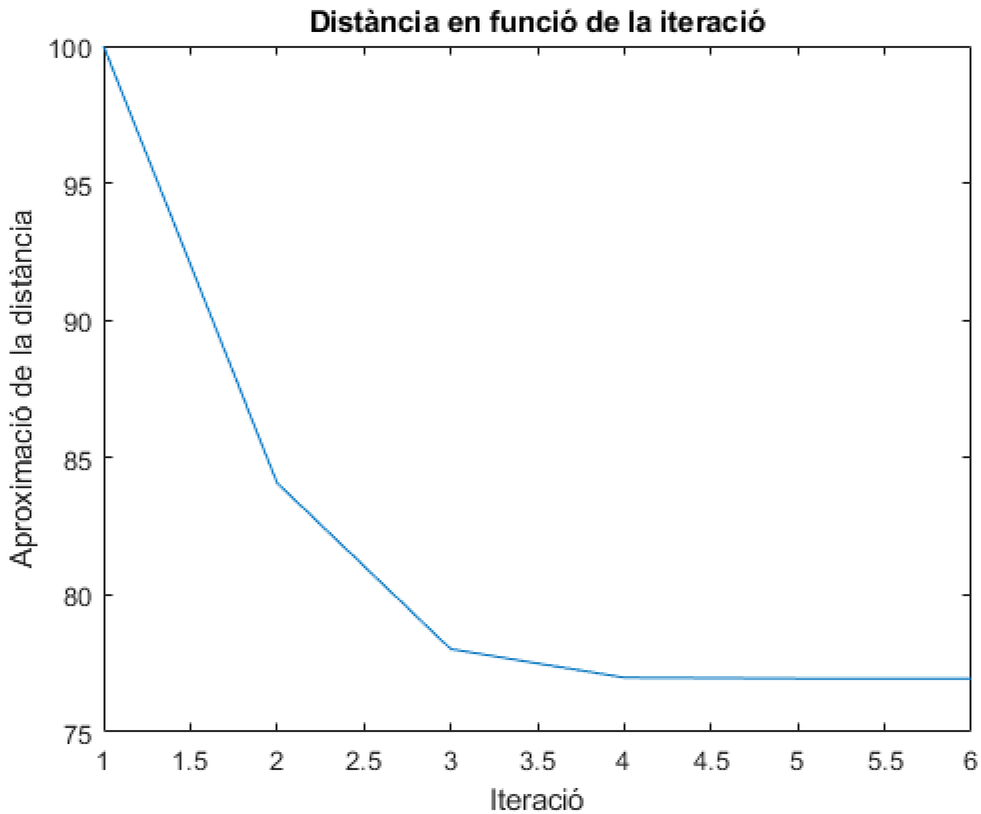
```
Aproximació a cada iteració: 100       84.08407       78.02427
76.98444       76.95372       76.95369
Error en el eix y a cada iteració: -30.7083       -6.692     -0.858017
-0.0239535 -2.29235e-05 -8.03365e-09
Nombre de iteracions: 5
```

Apartat c

```
exact = xv(end); % Considerem l'últim element del vector com al valor real
its = (1:iter+1);
figure(1)
semilogy(its, abs(xv-exact));
title('Error en el eix x en funció de la iteració')
xlabel('Iteració')
ylabel('Error')
```




```
figure(1)
plot(its, xv);
title('Distància en funció de la iteració')
xlabel('Iteració')
ylabel('Aproximació de la distància')
```



Apartat d

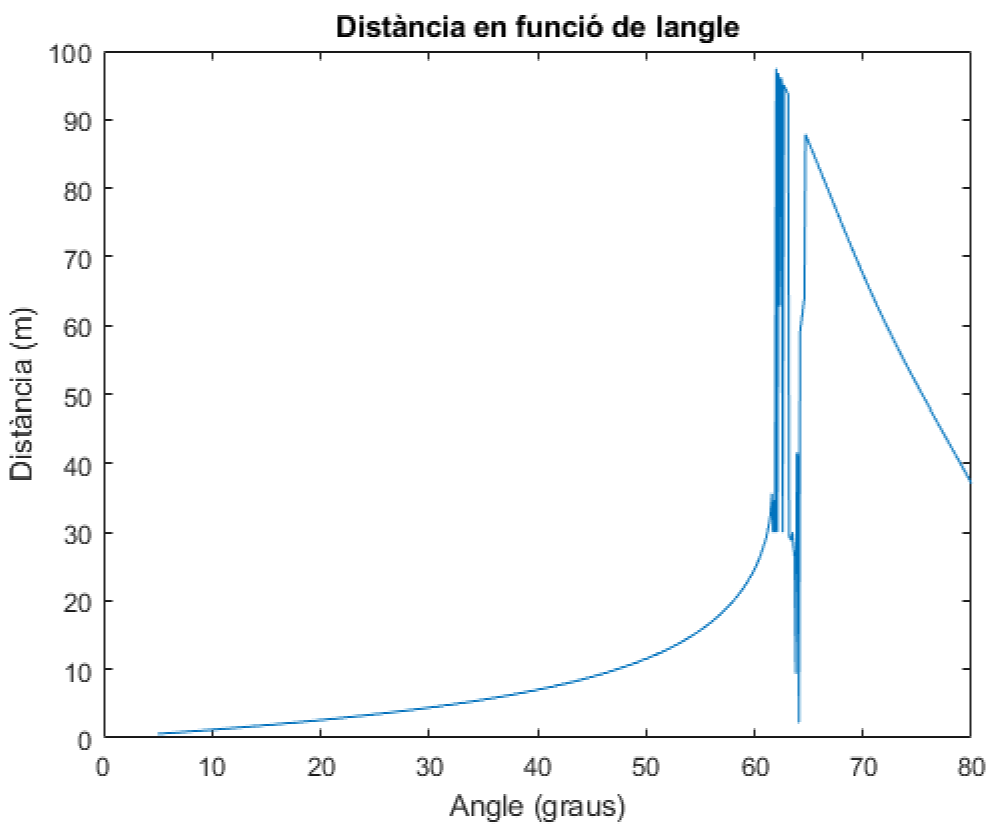
```
exact = xv(end); % Es pren l'última iteració de newton com el valor exacte

% De la regresió es treuen els punts dels extrems per tenir una millor
% regresió
E_n1 = xv(3:end-1)-exact;
E_n = xv(2: end-2)-exact;
[r,a,b] = reg_lin(log(E_n), log(E_n1));
disp(['p=' num2str(a)])
% Ha quedat una mica desviat del valor esperat (p=2), probablement degut a
% la poca quantitat de punts que hi ha
```

p=1.932

Apartat e

```
%i)
angles = (5:1/10:80);
x_imp = [];
iters = [];

amin = 61.6;
for alpha = angles
    [xv, fv, iter] = newton(20, 10^-10, 100, @F, alpha);
```

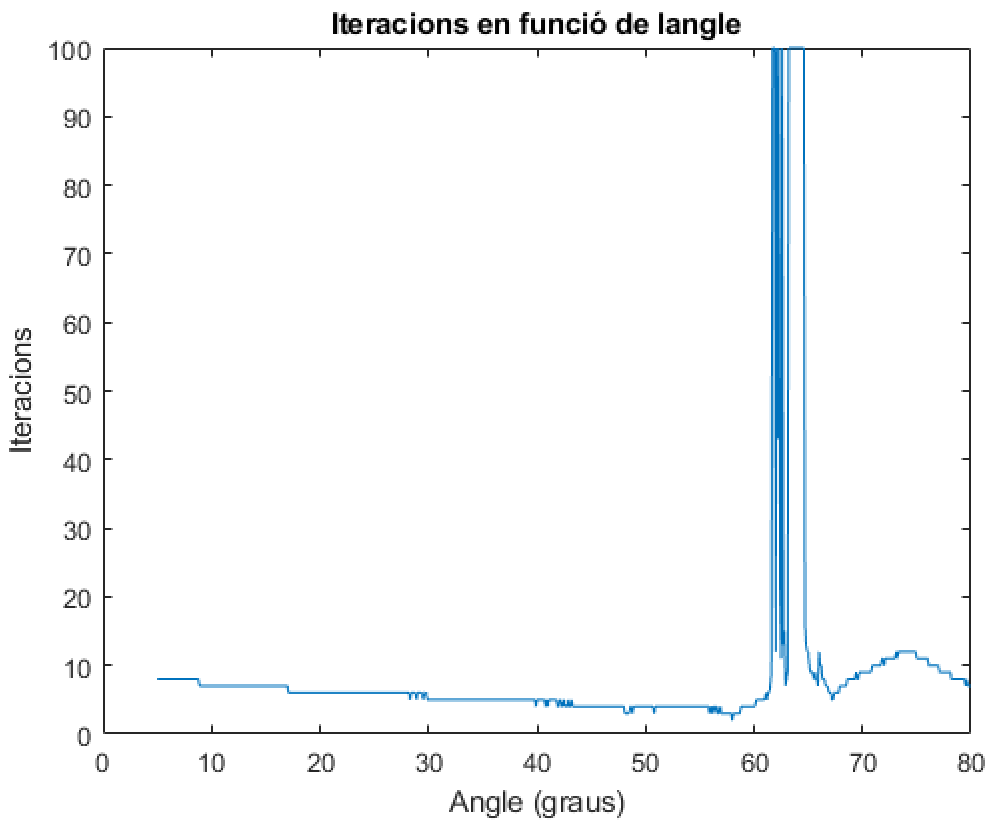
```
    x_imp = [x_imp xv(end)];
    iters = [iters, iter];
end
figure(1)
plot(angles, x_imp);
title('Distància en funció de langle')
xlabel('Angle (graus)')
ylabel('Distància (m)')
```




```
figure(2)
plot(angles, iters);
title('Iteracions en funció de langle')
xlabel('Angle (graus)')
ylabel('Iteracions')

%ii)
%l'angle mínim tal que passi el monticle correspon al punt just abans del
% màxim del vector x_imp (sense comptar discontinuitats)
[m,i] = max(x_imp);
disp(['Angle mínim: ' num2str(angles(i))])

%iii)
% A prop de l'angle mínim el mètode de Newton falla perquè la derivada
% s'aproxima a 0 i assigna un nou valor de x molt lluny de l'arrel, fent
% que costi molt convergir en la solució.

% Com es pot veure a la gràfica de les iteracions, a prop del mínim les
```

```
% iteracions son discontinues i moltes arriben a les 50 iteracions (el
% màxim imposat sobre per la funció). Per tant, es pot veure que a prop del
mínim
% la funció convergeix molt més lent.
```

Angle mínim: 62



Altres funcions

```
%{
function y = parabola_y(x, al)
    g = 9.81;
    vo = 37;
    al = al *2*pi/360;
    v_ox = vo*cos(al);
    v_oy = vo*sin(al);
    y=(v_oy/v_ox).*x-(1/2)*g*(x/v_ox).^2;
end
%}

%{
function y = y(x)
    a = .15;
    b = .04;
    y = a*x.^2.*exp(-b*x);
end
```

```
%}

%{
function f = F(x, al)
    f = y_parab(x, al)-y_mont(x);
end
%}

%{
function deriv = deriv_param(f, x0, param)
    Ax = 10 ^ -10; % arbitrary small number, to represent infinitessimal
increase
    deriv = (f(x0 + Ax, param) - f(x0, param)) / Ax;
end
%}

%{
function [xk, fk, it] = newton(x1, tol, itmax, fun, param)
    %x1: punt inicial       tol: interval tolerancia
    it = 0;
    xk = x1;
    fk = [fun(x1, param)];
    while (abs(fun(xk(end), param)) > tol) && (it < itmax)
        new_x = xk(end) - (fun(xk(end), param) / deriv_param(fun, xk(end),
param));
        if new_x < 0  % En aquest exercici no es volen les arrels negatives
            new_x = x1+10;
        end
        xk = [xk, new_x];
        fk = [fk, fun(xk(end), param)];
        it = it + 1;
    end
end
%}

%{
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Ajusta una recta y = a x + b als vectors x, y.
%
%  Torna el coeficient de regressió r, el pendent a i
%  l''ordenada a l''origen b.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [r,a,b]=reg_lin(x,y)

if iscolumn(x)
 x=x';
end

if iscolumn(y)
 y=y';
end
```

```
if size(x,2) ~= size(y,2)
 fprintf('La mida dels dos vectors no es la mateixa al cridar a reg_lin. \n')
 return
end

n=size(x,2);

ax=sum(x)/n; ay=sum(y)/n;

ax2=sum(x.^2)/n; ay2=sum(y.^2)/n; axy=sum(x.*y)/n;

a=(axy-ax*ay)/(ax2-ax^2);
c=(axy-ax*ay)/(ay2-ay^2);
r=sqrt(a*c);
b=ay-a*ax;
d=ax-c*ay;

%n,r,a,b,ax,ay,ax2,ay2,axy;

end
%}
```

Published with MATLAB® R2024a