Contents

* [P4: Cortés y García](#gjdgxs)
* [Problema](#30j0zll)
* [Apartado (a) : Ecuación F](#1fob9te)
* [Apartado (b) : Punto de impacto](#3znysh7)
* [Apartado (c) : Representación gráfica](#2et92p0)
* [Apartado (d) : Orden de convergencia](#tyjcwt)
* [Apartado (e): Ángulo variable](#3dy6vkm)
* [(e) (i) Coordenada de impacto e iteraciones en función del ángulo](#1t3h5sf)
* [(e) (ii)](#4d34og8)
* [(e) (iii)](#2s8eyo1)

## P4: Cortés y García

clear all  
format long

## Problema

g = -9.81; % m/s^2  
  
v0 = 37; % m/s  
alpha0 = 67.5; % º  
  
a = 0.15; %m^-1  
b = 0.04; % m^-1  
x0 = 0; y0 = 0; % m  
  
y\_hill = @(x) a.\*x.^2.\*exp(-b.\*x);

## Apartado (a) : Ecuación F

La pelota describirá una parábola, por lo que su trayectoria será: x = x0 + v0x\*t y = y0 + v0y\*t + 0.5\*g\*t^2

% Si aislamos t en función de x en la primera ecuación y lo substituimos en  
% la segunda, tenemos:  
y\_ball = @(x, v0, alpha0) y0+(x-x0).\*tand(alpha0)+0.5\*g\*((x-x0)/(v0\*cosd(alpha0))).^2;  
  
% Definimos F(x,v0,alpha0). La intersección sucede cuando y = y\_ball. Por  
% lo tanto, definiremos F como la diferencia entre ambas funciones  
% F = y\_ball - y\_hill  
F = @(x, v0, alpha0) y0+(x-x0).\*tand(alpha0)+0.5\*g\*((x-x0)/(v0\*cosd(alpha0))).^2-a.\*x.^2.\*exp(-b.\*x);  
  
x = [0:1:200];  
  
figure(1) % Hacemos los gráficos de interés  
area(x, y\_hill(x), 'LineWidth',2,'FaceColor',[0.50 0.75 0.75])  
hold on  
plot(x, y\_ball(x, v0, alpha0), 'LineWidth', 2)  
plot(x, F(x, v0, alpha0), 'LineWidth', 2, 'LineStyle', '--', 'Color', 'b')  
hold off  
grid on;  
title('Montículo y\_{hill}, parábola y\_{ball} e intersección F(x, v\_0, \alpha\_0)')  
ylim([0, 70]);  
xlabel('Position x (m)', 'fontsize',14)  
ylabel('Position y (m)', 'fontsize',14)  
  
% Tanto la trayectoria de la bola como F, llegan a valores negativos.  
% Puesto que a partir de y = 0 la bola no puede caer más, no se muestran en  
% la gráfica los valores negativos.

## Apartado (b) : Punto de impacto

El punto de impacto es el valor de x para el que F(x, v0, alpha0) = 0

x1 = 75;  
tol = 1e-12;  
itmax = 50;  
param = [v0, alpha0, a, b];  
  
[xkvec, fkvec, it\_last] = newton(x1, tol, itmax, param);  
  
root = xkvec(end) % Coordenada del impacto  
  
% Implementación funciones  
  
% Función F (F.m)  
% Entrada: valor de x, vector con parámetros incluidos en la definición  
% de la función  
% Salida: valor de la función evaluada en x dados los parámetros  
% function [f] = F(x, param)  
% % definimos g como valor auxiliar  
% g = -9.81;  
%  
% % Extraemos los valores de los parámetros  
% v0 = param(1);  
% alpha0 = param(2);  
% a = param(3);  
% b = param(4);  
%  
% % Evaluamos la función deseada  
% f = x.\*tand(alpha0)+0.5\*g\*(x/(v0\*cosd(alpha0))).^2-a.\*x.^2.\*exp(-b.\*x);  
% end  
  
% Función derivada (derivada.m)  
% Entrada: valor de x, vector con parámetros incluidos en la definición  
% de la función  
% Salida: derivada de la función evaluada en x  
% function der = derivada(x0, param);  
% dx = 1e-6;  
% der = (F(x0+dx, param)-F(x0, param))/dx;  
% end  
  
% Función newton (newton.m)  
% Entrada: valor inicial de x (x1), tolerancia permitida (tol), máximo de  
% iteraciones deseado (itmax), vector de parámetros auxiliares (param)  
% Salida: vector de xk obtenidas, función evaluada en xk (fk) y valor de la  
% ultima iteración (it)  
% function [xk,fk,it] = newton(x1,tol,itmax, param)  
% it = 0;  
% xk = [x1];  
% fk = [F(x1, param)];  
% ek = 1;  
%  
% while ek > tol && it < itmax  
% x = xk(end) - F(xk(end), param)/derivada(xk(end), param);  
% ek = abs(x - xk(length(xk)));  
% xk = [xk x];  
% fk = [fk F(x, param)];  
% it = it + 1;  
% end  
% end

root =  
  
 76.953686828663578

## Apartado (c) : Representación gráfica

ekvec = abs(xkvec - root); % error  
  
it = [1:1:it\_last+1]; % vector de iteraciones  
  
% Gráfica del error respecto a la iteración  
figure(2)  
plot(it, ekvec, 'LineWidth', 2)  
grid on  
title('Representación del error en cada iteración')  
xlabel('Error \epsilon')  
ylabel('Iteración')  
  
% Vemos que el error tiende a 0 en todo momento y, aunque de manera poco  
% perceptible, se acerca cada vez más.  
  
% Gráfica del valor xk respecto a la iteración  
figure(3)  
plot(it, xkvec, 'LineWidth', 2)  
grid on  
title('Representación del valor de x en cada iteración')  
xlabel('x')  
ylabel('Iteración')  
  
% Vemos que el valor de xk se acerca más al valor de la raíz de la función  
% a cada iteración.

## 

Apartado (d) : Orden de convergencia

Primero calcularemos los vectores Xk e Yk donde: Xk = log(e\_k) Yk = log(e\_{k+1}) El primer y el último elemento de cada vector serán suprimidos porque pueden dar lugar a errores computacionales

Xk = log10(ekvec(2:it\_last-1));  
Yk = log10(ekvec(3:it\_last));  
  
[r,p,d] = reg\_lin(Xk,Yk);  
  
p % valor de p calculado con la regresión lineal  
  
% obtenemos un orden de convergencia de p = 1.2789, por lo que si  
% tuvieramos que dar un valor entero diríamos que p = 2.  
  
% Implementación funciones  
  
% Función reg\_lin (reg\_lin.m)  
% Entrada: vectores x,y  
% Salida: coeficiente de regresión lineal (r), pendiente de la recta (a)  
% y ordenada en el origen (b)  
% function [r,a,b]=reg\_lin(x,y)  
% 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;  
% end

p =  
  
 1.278994600276342

## Apartado (e): Ángulo variable

## (e) (i) Coordenada de impacto e iteraciones en función del ángulo

Declaramos parámetros necesarios

x1 = 1;  
x\_imp =[];  
it\_imp = [];  
  
  
for angle = 5:1:80  
 % Creamos vector de parámetros  
 param = [v0, angle, a, b];  
  
 % Buscamos la raíz para cada ángulo  
 [xkvec, fkvec, it\_last] = newton(x1, tol, itmax, param);  
  
 % Actualizamos los vectores y parámetros necesarios  
 it\_imp = [it\_imp it\_last];  
 x\_imp = [x\_imp xkvec(end)];  
 x1 = abs(x\_imp(end));  
end  
  
alpha = [5:1:80]; % creado para hacer la gráfica  
  
% Hacemos los gráficos necesarios  
figure(4)  
plot(alpha,x\_imp, 'o', 'LineWidth',2)  
hold on  
plot(alpha, it\_imp, 'LineWidth', 2)  
hold off  
axis([5 80 0 100])  
grid on  
title('x\_{imp} y número de iteraciones necesario según \alpha\_0')  
xlabel('\alpha\_0 (º)', 'fontsize',14)  
legend('x\_{imp} (m)','# iteraciones', 'Location', 'northwest')  
  
% Función Newton implementada anteriormente.

## (e) (ii)

Encontramos el valor de x que da altura del monticulo 50 (altura maxima)

g\_hill = @(x) a\*x.^2.\*exp(-b.\*x) - 50;  
x1 = 40;  
[xk,fk,it] = minewton(x1,tol,itmax,g\_hill);  
x\_hmax = xk(end);  
  
% Calculamos el ángulo mínimo  
x1 = 1;  
alpha = 5;  
  
% Calculamos los valores de x\_imp y h\_max alcanzada para alpha = 5.  
param = [v0, alpha, a, b];  
[xkvec, fkvec, it\_last] = newton(x1, tol, itmax, param);  
y = y\_ball(xkvec, v0, alpha);  
h = max(y)  
  
% Saldremos del bucle cuando x\_imp sea mayor o igual que la coordenada que  
% da la altura máxima del montículo, y la bola haya superado la altura del  
% mismo  
while xkvec(end) < x\_hmax && h < 50  
 % Recalculamos los parámetros que necesitamos para alphas crecientes  
 alpha = alpha +0.1;  
 param = [v0, alpha, a, b];  
 [xkvec, fkvec, it\_last] = newton(x1, tol, itmax, param);  
  
 % Tomamos los datos  
 x1 = xkvec(end);  
 y = y\_ball(xkvec, v0, alpha);  
 h = max(y);  
end  
  
alpha\_min = alpha

h =  
  
 0.083878331768485  
  
  
alpha\_min =  
  
 61.700000000000607

## (e) (iii)

Vemos que cuando \alpha = \alpha\_{min} la trayectoria de la bola llega a ser tangente al montículo (aunque no llega a haber una intersección). Al aplicar el método de Newton en esta situación, el programa podria llegar a entender (según la tolerancia que se aplique) que hay dos raízes en posiciones muy cercanas. Se trata de un problema mal condicionado, por lo que las soluciones podrían no ser fiables. Si miramos el gráfico obtenido en el apartado (e)(i), vemos que este ángulo está muy cerca de una discontinuidad. También vemos que se da un aumento del número de iteraciones necesario para llegar a la raíz, debido al mal condicionamiento del problema para este ángulo.

x = [0:1:200];  
  
figure(5)  
area(x, y\_hill(x), 'LineWidth',2,'FaceColor',[0.50 0.75 0.75])  
hold on  
plot(x, y\_ball(x, v0, alpha\_min), 'LineWidth', 2)  
plot(x, F(x, v0, alpha\_min), 'LineWidth', 2, 'LineStyle', '--', 'Color', 'b')  
hold off  
grid on;  
title('Montículo y\_{hill}, parábola y\_{ball} e intersección F(x, v\_0, \alpha\_0)')  
ylim([0, 70]);  
xlabel('Position x (m)', 'fontsize',14)  
ylabel('Position y (m)', 'fontsize',14)  
  
% Implementación de funciones  
  
% Función derivative (derivative.m)  
% Entrada: función a evaluar (f), valor de x para evaluar (x0)  
% Salida: Evaluación de f'(x) en x0  
% function der = derivative(f, x0)  
% dx = 1e-6;  
% der = (f(x0+dx)-f(x0))/dx;  
% end  
  
% Función minewton (minewton.m)  
% Entrada: valor inicial de x (x1), tolerancia permitida (tol), máximo de  
% iteraciones deseado (itmax), función de la que se pretende encontrar la  
% raiz (fun)  
% Salida: vector de xk obtenidas, vector de función evaluada en xk (fk)  
% y valor de la ultima iteración (it)  
% function [xk,fk,it] = minewton(x1,tol,itmax,fun)  
% it = 0;  
% xk = [x1];  
% fk = [fun(x1)];  
% ek = 1;  
%  
% while ek > tol && it < itmax  
% x = xk(end) - fun(xk(end))/derivative(fun,xk(end));  
% ek = abs(x - xk(end));  
% xk = [xk x];  
% fk = [fk fun(x)];  
% it = it + 1;  
% end  
% end



[*Published with MATLAB® R2020b*](https://www.mathworks.com/products/matlab/)