Calculo numerico de parametros relacionados con el lanzamiento de un misil considerando la resitencia del aire.

September 23, 2018

1 Problema

Un misil es lanzado con una velocidad inicial v_0, sin considerar la resistencia k con el aire (es decir $k = 0$) el problema se reduce a un caso de tiro parabólico, sin embargo al considerarla el tiempo total de vuelo es menor que en el caso del tiro parabólico y consecuentemente tambien el alcance máximo.

Se puede demostrar que el tiempo de vuelo total

El problema consiste en hallar el tiempo de vuelo total T (es cuando $y(t) = 0$ pero $t \neq 0$) mediante un método numérico para distintos valores de velocidad inicial v_0, de coficiente de resistencia con el viento k y ángulo θ de lanzamiento. El método debe hallar los valores de T dónde se cumpla la **ecuación trascendental**:

$$T = \frac{V_0yk + g}{gk}(1 - e^{-kT})$$

1.0.1 Información adicional al problema:

Dirección-x:

Desplazamiento ec.2a

$$x(t) = \frac{V_{0x}}{k}(1 - e^{-kt}) \tag{1}$$

Velocidad ec.2b

$$\dot{x}(t) = V_{0x}e^{-kt} \tag{2}$$

Dirección-y:

Desplazamiento ec.3a

$$y(t) = -\frac{g}{k}t + \frac{V_{0y}k + g}{k^2}(1 - e^{-kt})$$

Velocidad ec.3b

$$\dot{y}(t) = -\frac{g}{k} + \frac{V_{0y}k + g}{k}e^{-kt}$$

2 Solución propuesta

Lo que se hará es hallar el valor de T para el cual los dos miembros de la ecuación trascendental sean iguales, es equivalente al problema de hallar las raices de la siguiente ecuación: ** ec. 1 **

$$f(T) = \frac{V_0 yk + g}{gk}(1 - e^{-kT}) - T$$

Es decir el o los valores de T para los cuales $ f(T) = 0 $ ya que siempre que esto se verifique estaremos hallando los valores de T que satisfacen la ecuación trascendental (la primera ecuación dada).

El método numérico que se empleará para hallar las raices de ec. 1 es el método de Newton-Rhapson.

2.1 a) Calculando T para distintos valores de parámetros

Usando un algoritmo recursivo, crear un código que calcule T para diferentes valores de k, el ángulo y velocidad inicial

2.1.1 Definiciones iniciales

In [1]:
```
import matplotlib.pyplot as plot
import math
import numpy  as np
import pandas as pd

'''
Los valores para los parámetros v_0 y theta:
'''

v_0 = 500 #en m/s

theta = 65   #en grados
theta_rad=(math.pi)/180*theta #conversion a radianes

'''
Velocidades iniciales en la dirección-y y en la dirección-x.
Se asume que sale del orginen por lo que x_0=0 y y_0=0.
'''

v_0y=v_0*math.sin(theta_rad) #La velocidad en y en m/s

v_0x= v_0*math.cos(theta_rad) #La velocidad en x en m/s

'''
La ec.1 y su derivada.
'''

#ec.1
```

```
def f(T,k,v_0=500,theta=65):
    '''
    T: variable independiente
    k: parámetro de fricción contra el viento
    v_0: velocidad inicial
    theta: ángulo dado en grados
    '''
    theta_rad=theta*(math.pi)/180 #conversion a radianes
    v_0y=v_0*math.sin(theta_rad)  #velocidad en y
    g=9.8
    return ((k*v_0y + g)/(g*k))*(1-math.exp(-k*T))-T

#La derivada de la ec.1
def Df(T,k,v_0=500,theta=65):
    theta_rad=theta*(math.pi)/180 #conversion a radianes
    v_0y=v_0*math.sin(theta_rad)  #velocidad en y
    g=9.8
    return (k*v_0y + g)/(g)*math.exp(-k*T) - 1

'''
Funciones de posición en direcciones-x e y
y sus derivadas (velocidades).
'''

#Distancia en direccion x ec.2
def rx(t,k,v_0=500,theta=65):
    theta_rad=theta*(math.pi)/180 #conversion a radianes
    v_0x= v_0*math.cos(theta_rad) #velocidad en x
    return (v_0x/k)*(1.0-math.exp(-k*t))

#Distancia en dirección y ec.3
def ry(t,k,v_0=500,theta=65):
    theta_rad=theta*(math.pi)/180 #conversion a radianes
    v_0y=v_0*math.sin(theta_rad)  #velocidad en y
    g=9.8
    return -(g/k)*t + ((v_0y*k + g)/(k**2.0))*(1.0-math.exp(-k*t))

#Velocidad en direccion x
def vx(t,k,v_0=500,theta=65):
    theta_rad=theta*(math.pi)/180 #conversion a radianes
    v_0x= v_0*math.cos(theta_rad) #velocidad en x
    return (v_0x)*(math.exp(-k*t))

#Velocidad en dirección y
def vy(t,k,v_0=500,theta=65):
    theta_rad=theta*(math.pi)/180 #conversion a radianes
    v_0y=v_0*math.sin(theta_rad)  #velocidad en y
    g=9.8
```

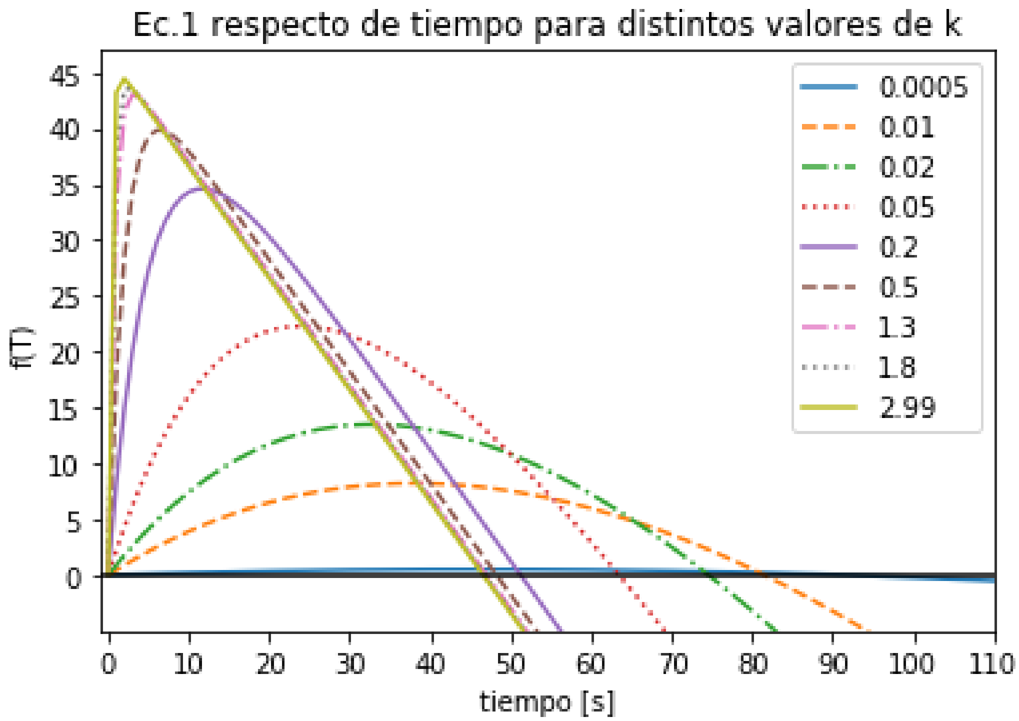
```
    return -(g/k) + ((v_0y*k + g)/(k))*(math.exp(-k*t))
```

Ahora vamos a generar gráficas para la ec. 1 y observar la ubicación aproximada de las raíces. Iremos variando el parámetro k para observar cómo cambia la forma de la curva y así estimar los límites apropiados.

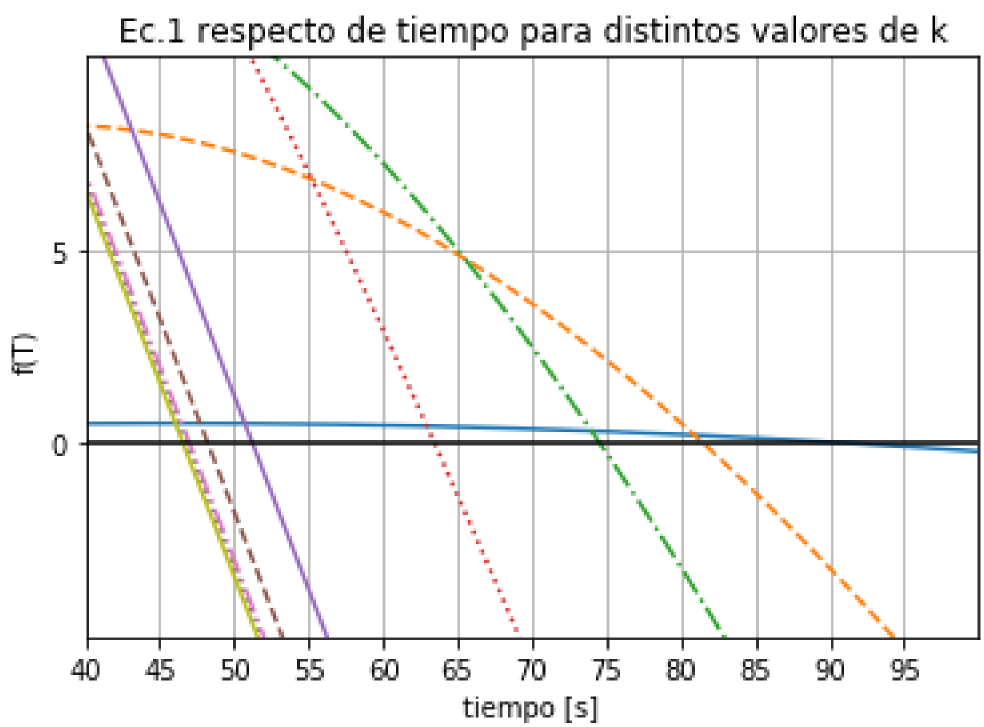
In [2]:
```
'''
Un conjunto de k's generados para cada caso de la ec.1
'''
r=300

k_i=[]
for i in range(r):
    k_i.append(i*0.01) #distintos valores para k
k_i[0]=0.5e-3

#valores para x para evaluar ahí ec.1
x_i=[]
for n in range(r+20):
    x_i.append(n)

f_k=[]
for i in range(r):
    f_kn=[]
    for j in range(r+20):
        f_kn.append(f(x_i[j],k_i[i]))
    f_k.append(f_kn)

'''
Graficando las curvas generadas para cada uno
de los distintos valores de k
'''

print("Ec.1 respecto de tiempo para distintos valores de k con v_0 = 500 m/s y theta = 65 grados")

k_indx=[0,1,2,5, 20,50,130,180,299]

styles = ['-', '--', '-.', ':']
s=0
for i in k_indx:
    plot.plot(x_i,f_k[i], linestyle=styles[s],label = str(k_i[i]))
    if(s<3):
        s=s+1
    else:
        s=0

plot.axis([-1,100,-5,47])
plot.title("Ec.1 respecto de tiempo para distintos valores de k")
```

```
plot.ylabel("f(T)")
plot.xlabel("tiempo [s]")
plot.xticks(np.arange(0, 120, step=10))
plot.yticks(np.arange(0, 50, step=5))
plot.legend()
plot.axhline(y=0, color='k')
plot.show()

print("hacemos un zoom en la región 40<t<100 para estimar los límites")

styles = ['-', '--', '-.', ':']
s=0
for i in k_indx:
    plot.plot(x_i,f_k[i], linestyle=styles[s],label = str(k_i[i]))
    if(s<3):
        s=s+1
    else:
        s=0

plot.axis([40,100,-5,10])
plot.title("Ec.1 respecto de tiempo para distintos valores de k")
plot.ylabel("f(T)")
plot.xlabel("tiempo [s]")
plot.xticks(np.arange(40, 100, step=5))
plot.yticks(np.arange(0, 10, step=5))
#plot.legend()
plot.axhline(y=0, color='k')
plot.grid()
plot.show()
```

Ec.1 respecto de tiempo para distintos valores de k con v_0 = 500 m/s y theta = 65 grados



hacemos un zoom en la región 40<t<100 para estimar los límites



Se requiere acotar el rango dónde se buscarán las raíces por el método de Newton-Rhapson, para ello se solicitan el límite superior y el límite inferior, para estimarlos solo se observan las gráficas variando el parámetro k. Se observó que cuando k tiende a 0 el tiempo tiende aun valor entre 80 y 95 segundos. Si k es muy grande tambien se llega a un límite, v_{0y}/g que para los valores de $v_0 = 500$ esta al rededor de 46.24 seg.

In [3]:
```
'''
limite inferior a, y superior b
'''
a=40.0
b=100.0
```

Usarémos el método de *Newton_Rhapson* con una ligera modificación, combinamos con el método de bisección para asegurar que no diverja.

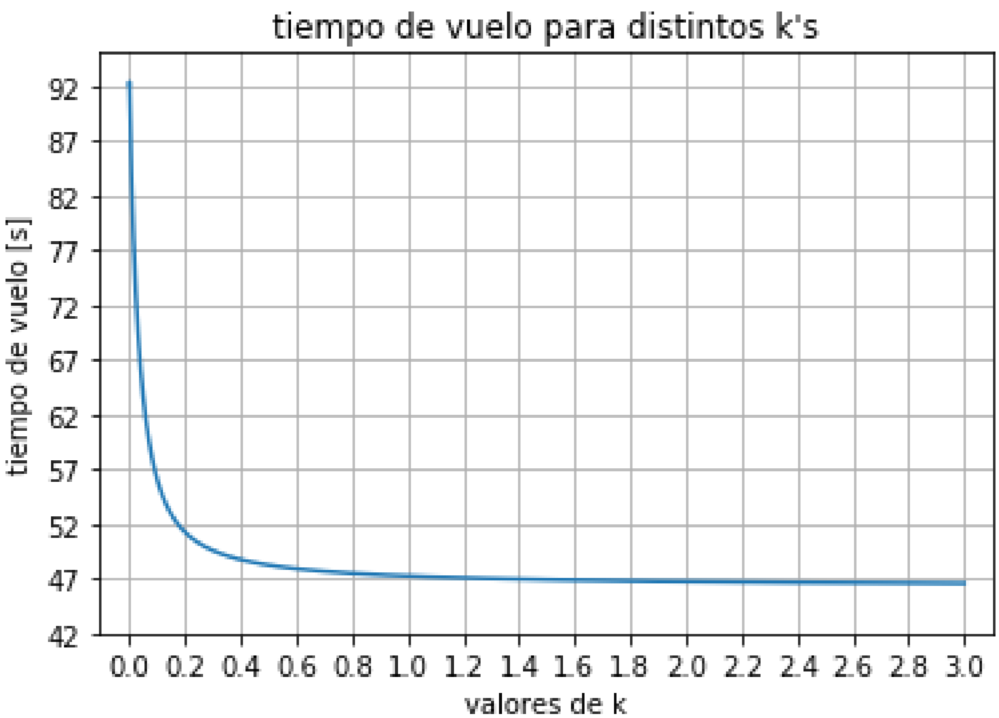
In [41]:
```
def newtonRaphson(k,a=40.0, b=100.0,v_0=500,theta=65):
    '''
    args(k,a=40.0, b=100.0,v_0=500,theta=65)
    k: parámetro de friccion con el aire
    a: limite inferior (tiempo)
    b: limite superior (tiempo)
    v_0: magnitud de la velocidad inicial
    theta: angulo dado en grados
    '''
    #no es necesaria la conversion de theta porque las funciones que usamos la implementan es su definicion
    #tampoco es necesario calcular v_0x, v_oy por la misma razón

    tol=1.0e-9
    i=0

    x = 0.5*(a + b)

    for i in range(301): #mientras que el numero de iteraciones sea menor que 300 haz:

        #criterio de paro por tolerancia
        fx = f(x,k,v_0,theta)
        if abs(fx) < tol:
            return x

        #----------------------------
        '''
        # Ajustar límites (bisección)
        if fa*fx < 0.0:
            b = x
        else:
            a = x
```

```
        '''
        #---------------------------

        # Try a Newton-Raphson step
        dfx = Df(x,k,v_0,theta)

        # If division by zero, push x out of bounds
        try:
            dx = -fx/dfx
        except ZeroDivisionError:
            dx = b-a

        x = x + dx

        #----------------------
        '''
        # If the result is outside the brackets, use bisection
        if (b - x)*(x - a) < 0.0:
            dx = 0.5*(b-a)
        '''
        #----------------------

        #x = a + dx

        # Check for convergence
        if abs(dx) < tol*max(abs(b),1.0):
            return x

    print("Too many iterations in Newton-Raphson")
    return x
```

Ahora haremos variar k y obtendremos los distintos valores de t para cada k.

```
In [5]: r=3e4
        k_i=[]
        for i in range(int(r)):
            k_i.append(i*1e-4) #distintos valores para k
        k_i[0]=1e-9

        t_n=[]
        for k in k_i:
            t_n.append(newtonRaphson(k))

        plot.plot(k_i,t_n)
        yi=42
        ys=95
        xi=-0.11
        xs=3.1
```

```
plot.axis([xi,xs,yi,ys])
plot.xticks(np.arange(0, xs, step=0.2))
plot.yticks(np.arange(yi, ys, step=5))
plot.title("tiempo de vuelo para distintos k's")
plot.xlabel("valores de k")
plot.ylabel("tiempo de vuelo [s]")
plot.axhline(y=0, color='k')
#plot.legend("n")
plot.grid()
plot.show()

print("el tiempo de vuelo tiene un máximo en 0 con un valor de: "+str(max(t_n))+"[s]")
```



el tiempo de vuelo tiene un máximo en 0 con un valor de: 92.33828066625702[s]

Cuando $k \to \infty$ los valores tienden a una asíntota con un valor debajo de 50, lo cual coincide con el límite calculado de 46.

In [6]:
```
'''
Un valor muy grande de k confirma el límite
'''
newtonRaphson(200000)
```

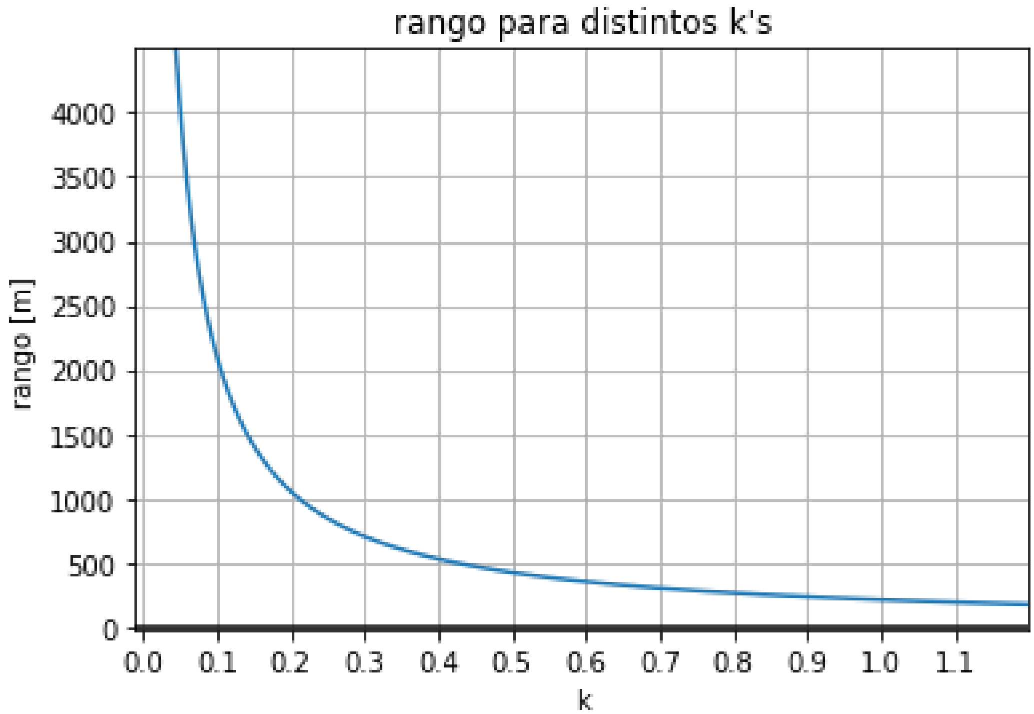
Out[6]: 46.240198216155605

2.2 b) El rango contra K

Con la velocidad inicial de 500 m/s y un ángulo inicial de 65 grados graficar el Rango contra k para k=0,0.05 y otros 3 valores entre 0 y 1. Comparar con la aproximación vista en clase basada en teoría de perturbaciones.

Calculamos el tiempo de vuelo total para k=0, k=0.05, k=0.1, k=0.5, k=1, luego evaluamos la ec.2a.

In [7]:
```
rx_max_n=[]
for i in range(int(r)):
    rx_max_n.append(rx(t_n[i],k_i[i]))

plot.plot(k_i,rx_max_n)
plot.axis([-0.01,1.2,-0.2e2,4.5e3])
plot.xticks(np.arange(0, 1.2, step=0.1))
plot.yticks(np.arange(0, 4.5e3, step=500))
plot.title("rango para distintos k's")
plot.ylabel("rango [m]")
plot.xlabel("k")
#plot.legend("n")
plot.axhline(y=0, color='k')
plot.grid()
plot.show()

print("el rango (o alcance máximo) tiene un máximo en k=0 \n con un valor de:\n"+str(max(rx_max_n))
print("Para los valores de k= 0.05, 0.1, 0.5 y 1 los rangos son:\n ")
print("k=0.05\t"+str(rx_max_n[k_i.index(0.05)])+" m\n a los "+str(t_n[k_i.index(0.05)])+" s\n")
print("k=0.1\t"+str(rx_max_n[k_i.index(0.1)])+" m\n a los "+str(t_n[k_i.index(0.1)])+" s\n")
print("k=0.5 \t"+str(rx_max_n[k_i.index(0.5)])+" m\n a los "+str(t_n[k_i.index(0.5)])+" s\n")
print("k=1 \t"+str(rx_max_n[k_i.index(1)])+" m\n a los "+str(t_n[k_i.index(1)])+" s\n")
```



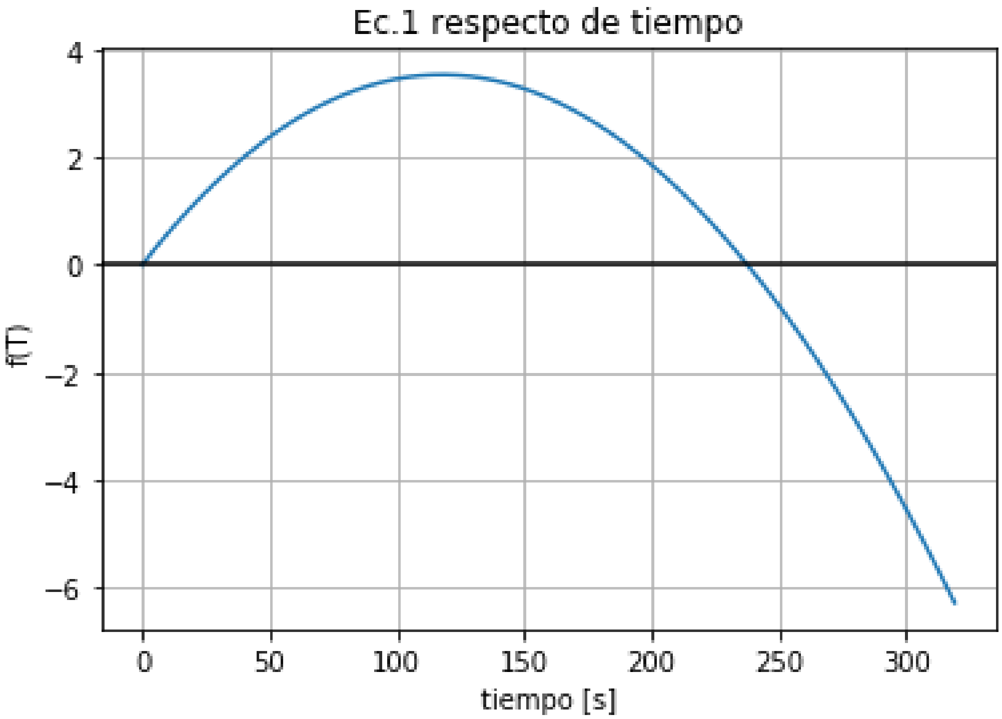

```
el rango (o alcance máximo) tiene un máximo en k=0
 con un valor de:
19469.95680143477 m a los 92.13968947502173 segundos
Para los valores de k= 0.05, 0.1, 0.5 y 1 los rangos son:

k=0.05         4049.2631881056445 m
 a los 63.46719965637157 s

k=0.1         2105.303056743879 m
 a los 56.03290790234983 s

k=0.5          422.6182617265505 m
 a los 48.24019321454057 s

k=1          211.30913087034972 m
 a los 47.24019321615561 s
```

Comparando con lo visto en clase:

In [8]:
```
'''
    Calculando el alcance máximo de Big Bertha. v_0 = 1450 m/s, theta = 55 grados, caso 1: k=0
    '''
```

```python
#primero hacemos una proximacion graficando la ecuacion 1 como anteriormente
r=300

k_i=0.5e-3 #prácticamente 0


f_k=[]
x_i=[]
for i in range(r+20):
    x_i.append(i)
    f_k.append(f(i,k_i,1450,55))

'''
Graficando las curvas generadas para cada uno
de los distintos valores de k
'''

print("Ec.1 respecto de tiempo para k=0, v_0 = 1450 m/s y theta = 55 grados")


plot.plot(x_i,f_k)
#plot.axis([-1,100,-5,47])
plot.title("Ec.1 respecto de tiempo")
plot.ylabel("f(T)")
plot.xlabel("tiempo [s]")
#plot.xticks(np.arange(0, 120, step=10))
#plot.yticks(np.arange(0, 50, step=5))
#plot.legend()
plot.grid()
plot.axhline(y=0, color='k')
plot.show()
print("entonces el limite inferior será a1=200 y el limite superi")
```

Ec.1 respecto de tiempo para k=0, v_0 = 1450 m/s y theta = 55 grados



entonces el limite inferior será a1=200 y el limite superi

```
In [9]: t_caidaEnk0=newtonRaphson(1e-5,200,300,1450,55)
        rx_enK0=rx(t_caidaEnk0,1e-5,1450,55)
        print("tiempo de caida con k=0: "+str(t_caidaEnk0)+" seg.\n rango alcanzado "+str(rx_enK0)+" m")
```

```
tiempo de caida con k=0: 242.30428329058952 seg.
 rango alcanzado 201277.0896356609 m
```

Los resultados obtenidos difieren de los obtenidos en clase por menos del 1%.

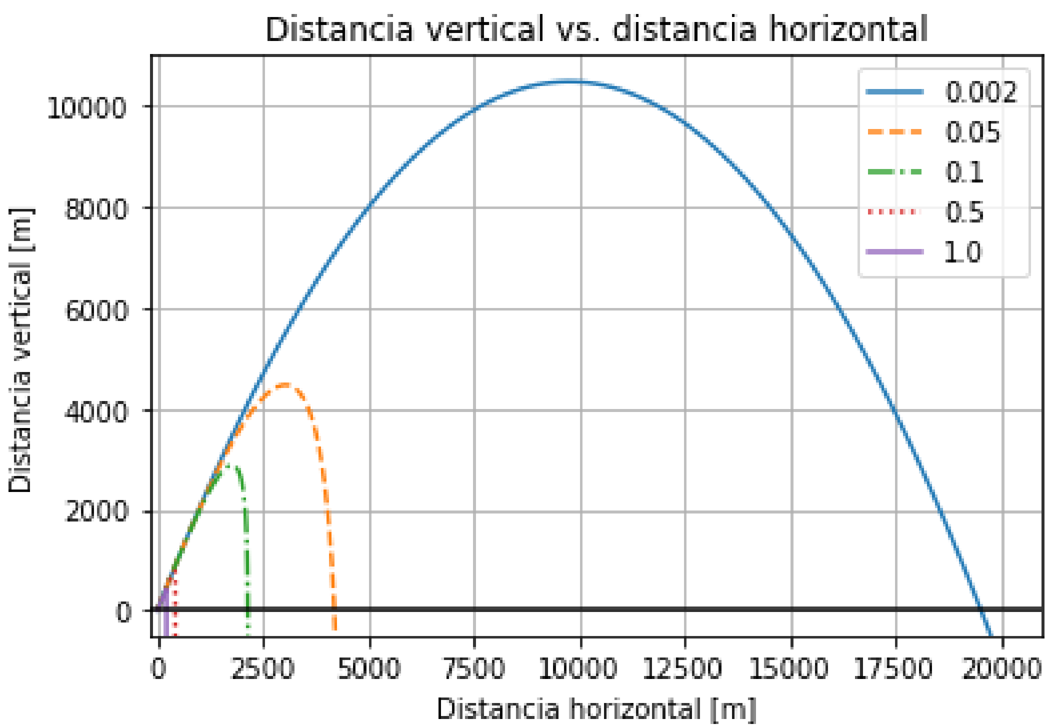
2.3 c) Distancia vertical vs. Distancia horizontal

Usando los mismos datos del incisio anterior graficar distancia horizontal vs. distancia vertical.

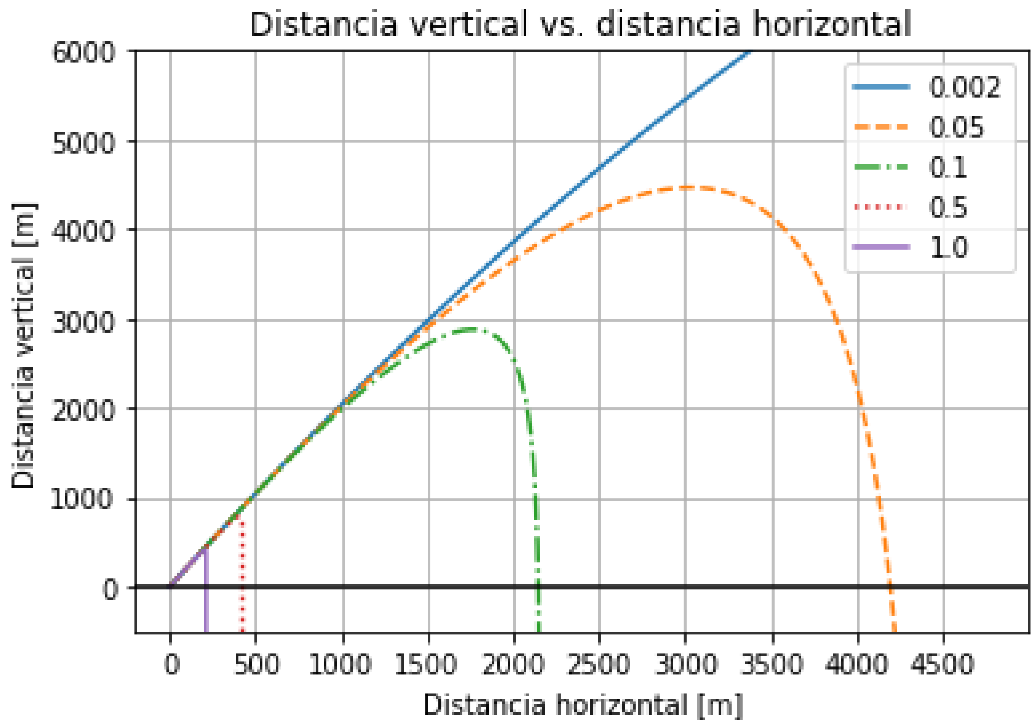
```
In [10]: r=500

         t_i=[]
         for i in range(r):
             t_i.append(i*0.5)

         k_i=[]
```

```python
for i in range(r):
    k_i.append(i*0.002) #distintos valores para k
k_i[0]=1e-6 #para que K no provoque una división por cero

r_x=[]
r_y=[]
k_n=[0,24,49,249,499]
for i in k_n:
    r_xi=[]
    r_yi=[]
    for t in range(r):
        r_xi.append(rx(t_i[t],k_i[i]))
        r_yi.append(ry(t_i[t],k_i[i]))
    r_x.append(r_xi)
    r_y.append(r_yi)



'''
Graficando las curvas generadas para distintos valores de k
'''

#print("Ec.1 respecto de tiempo para distintos valores de k con v_0 = 500 m/s y theta = 65 grados")


styles = ['-', '--', '-.', ':']
s=0
for i in k_n:
    plot.plot(r_x[k_n.index(i)],r_y[k_n.index(i)], linestyle=styles[s],label = str((i+1)*0.002))
    if(s < 3):
        s=s+1
    else:
        s=0

plot.axis([-200,21e3,-500,11e3])
plot.title("Distancia vertical vs. distancia horizontal")
plot.ylabel("Distancia vertical [m]")
plot.xlabel("Distancia horizontal [m]")
#plot.xticks(np.arange(0, 120, step=10))
#plot.yticks(np.arange(0, 50, step=5))
plot.legend()
plot.axhline(y=0, color='k')
plot.grid()
plot.show()

print("hacemos un zoom en la región 0<t<5000")

styles = ['-', '--', '-.', ':']
```

```
s=0
for i in k_n:
    plot.plot(r_x[k_n.index(i)],r_y[k_n.index(i)], linestyle=styles[s],label = str((i+1)*0.002))
    if(s < 3):
        s=s+1
    else:
        s=0

plot.axis([-200,5e3,-500,6e3])
plot.title("Distancia vertical vs. distancia horizontal")
plot.ylabel("Distancia vertical [m]")
plot.xlabel("Distancia horizontal [m]")
plot.xticks(np.arange(0, 5e3, step=500))
#plot.yticks(np.arange(0, 50, step=5))
plot.legend()
plot.axhline(y=0, color='k')
plot.grid()
plot.show()
```



hacemos un zoom en la región 0<t<5000



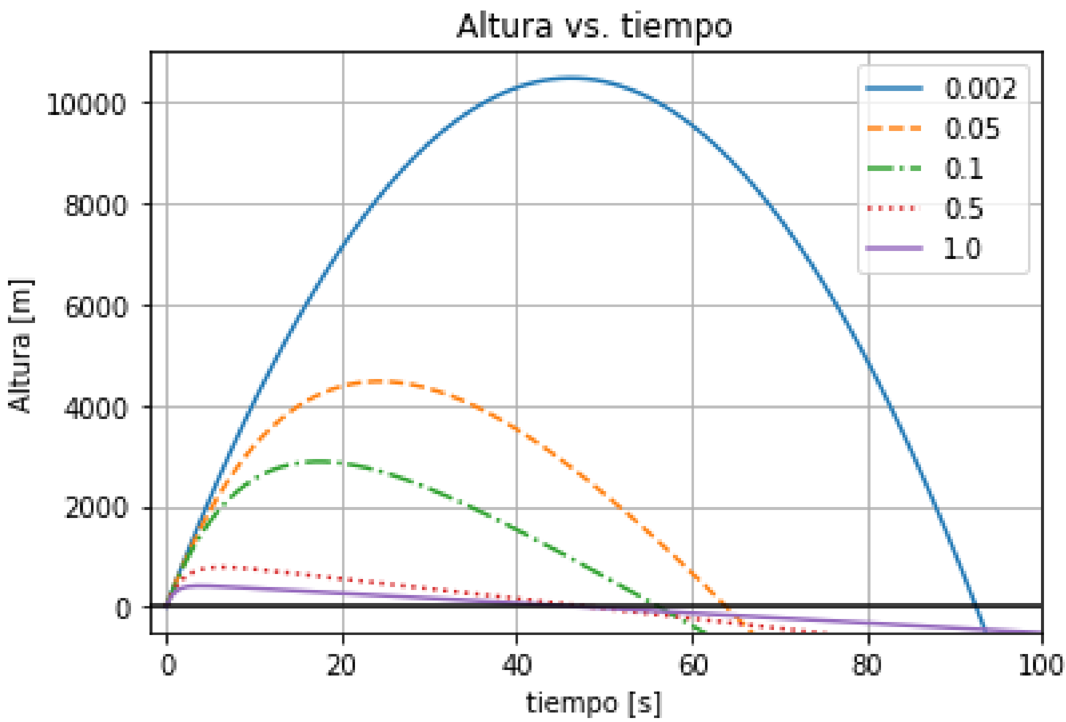
2.4 d)

Usando los mismos datos iniciales que en los puntos anteriores, graficar altura vs. tiempo, velocidad horizontal vs. tiempo y velocidad vertical contra tiempo para los mismos valores de k.

2.4.1 Altura vs. Tiempo

```
In [11]: styles = ['-', '--', '-.', ':']
         s=0
         for i in k_n:
             plot.plot(t_i,r_y[k_n.index(i)], linestyle=styles[s],label = str((i+1)*0.002))
             if(s < 3):
                 s=s+1
             else:
                 s=0

         plot.axis([-2,100,-500,11e3])
         plot.title("Altura vs. tiempo")
         plot.ylabel("Altura [m]")
         plot.xlabel("tiempo [s]")
         #plot.xticks(np.arange(0, 120, step=10))
         #plot.yticks(np.arange(0, 50, step=5))
         plot.legend()
         plot.axhline(y=0, color='k')
```

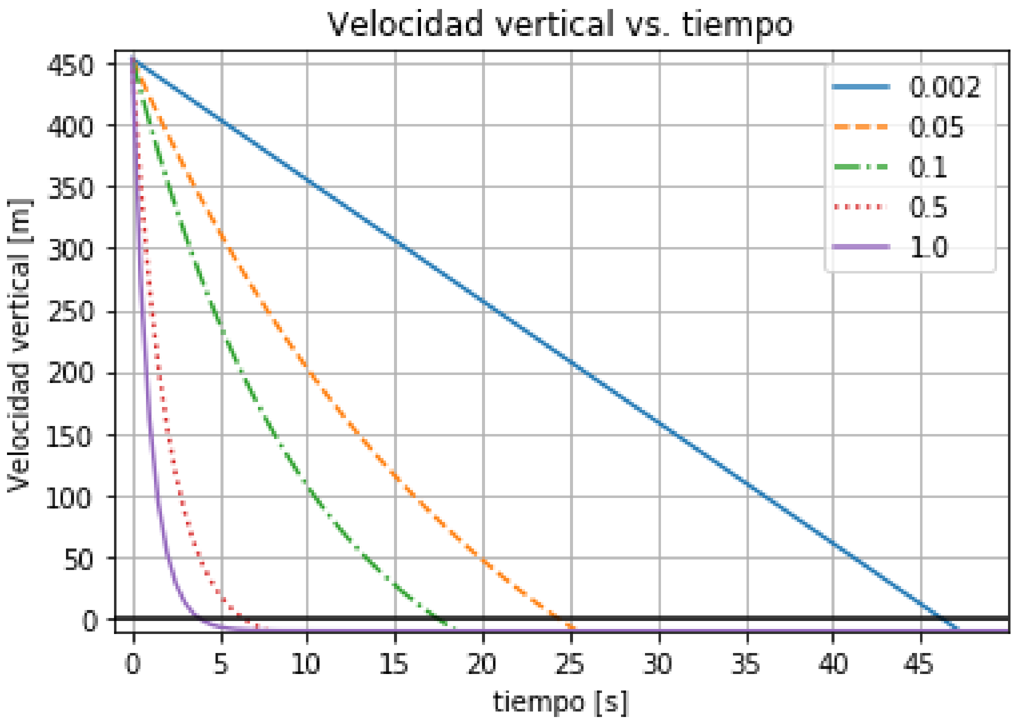
```
plot.grid()
plot.show()
```



2.4.2 Velocidad vertical vs. tiempo

In [12]:
```
r=500

t_i=[]
for i in range(r):
    t_i.append(i*0.5)

k_i=[]
for i in range(r):
    k_i.append(i*0.002) #distintos valores para k
k_i[0]=1e-6 #para que K no provoque una división por cero

v_x=[]
v_y=[]
k_n=[0,24,49,249,499]
for i in k_n:
    v_xi=[]
    v_yi=[]
    for t in range(r):
        v_xi.append(vx(t_i[t],k_i[i]))
```

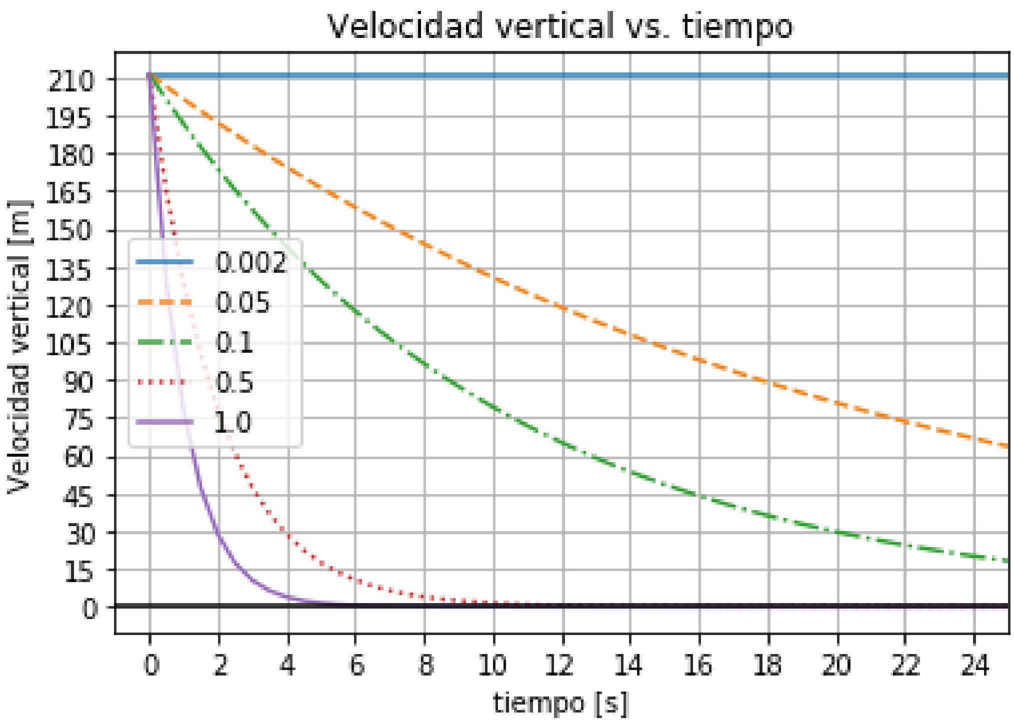
```
        v_yi.append(vy(t_i[t],k_i[i]))
    v_x.append(v_xi)
    v_y.append(v_yi)



'''
Graficando las curvas generadas para distintos valores de k
'''

#print("Ec.1 respecto de tiempo para distintos valores de k con v_0 = 500 m/s y theta = 65 grados")


styles = ['-', '--', '-.', ':']
s=0
for i in k_n:
    plot.plot(t_i,v_y[k_n.index(i)], linestyle=styles[s],label = str((i+1)*0.002))
    if(s < 3):
        s=s+1
    else:
        s=0

plot.axis([-1,50,-10,460])
plot.title("Velocidad vertical vs. tiempo")
plot.ylabel("Velocidad vertical [m]")
plot.xlabel("tiempo [s]")
plot.xticks(np.arange(0, 50, step=5))
plot.yticks(np.arange(0, 500, step=50))
plot.legend()
plot.axhline(y=0, color='k')
plot.grid()
plot.show()
```



2.4.3 Velocidad horizontal vs. tiempo

In [13]:
```
'''
Graficando las curvas generadas para distintos valores de k
'''

#print("Ec.1 respecto de tiempo para distintos valores de k con v_0 = 500 m/s y theta = 65 grados")


styles = ['-', '--', '-.', ':']
s=0
for i in k_n:
    plot.plot(t_i,v_x[k_n.index(i)], linestyle=styles[s],label = str((i+1)*0.002))
    if(s < 3):
        s=s+1
    else:
        s=0

plot.axis([-1,25,-10,220])
plot.title("Velocidad vertical vs. tiempo")
plot.ylabel("Velocidad vertical [m]")
plot.xlabel("tiempo [s]")
plot.xticks(np.arange(0, 25, step=2))
```

```
plot.yticks(np.arange(0, 220, step=15))
plot.legend()
plot.axhline(y=0, color='k')
plot.grid()
plot.show()
```



2.5 e) Ángulo que maximiza el rango

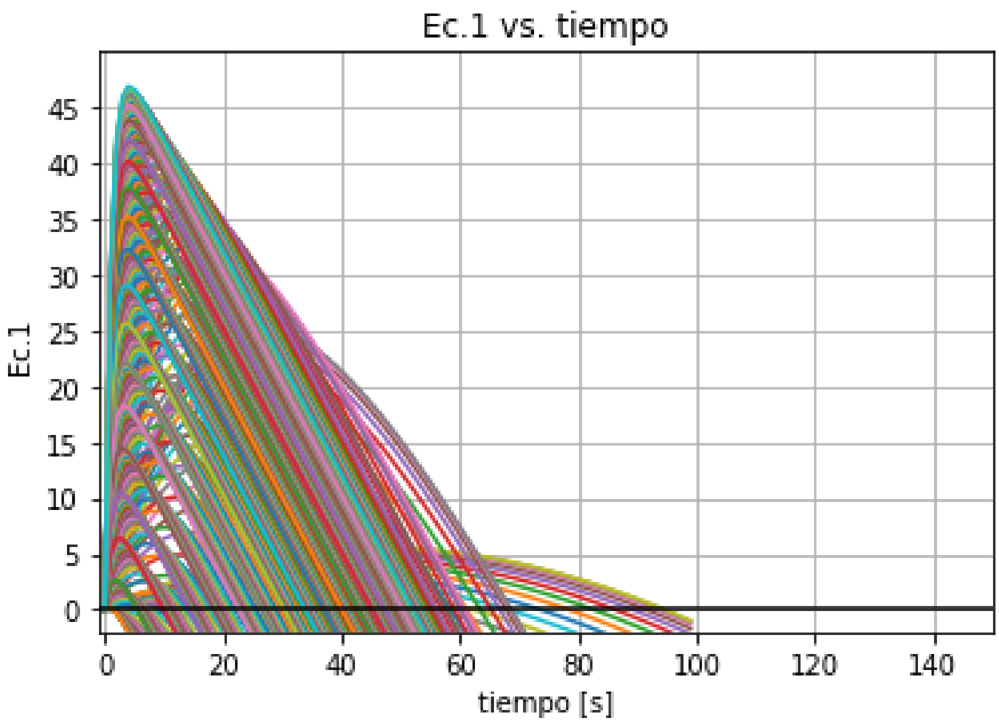
Buscar el ángulo que da la distancia máxima numéricamente para los dato anteriores.

Lo que debo hacer es fijar el parámetro k, calcular T (tiempo total de caída) para ese k partícular pero variando θ obteniendo T_1, T_2, ..., T_n datos y con esos datos luego calcular el rango pero considerando que el ángulo es el θ_i asociado a ese T_i para ese rango particular.

```
In [45]: '''
         Primero bosquejamos la curva parca k's fijos y theta
         variable para conocer el comportamiento de T con
         distintos ángulos y determinar límites apropiados
         '''

         # 100 valores discretos (N) de tiempo
         t=[]
         for t_i in range(0,100):
             t.append(t_i)
```

```
# tetha va de 0 a 90 de 5 en 5
theta=[]
for theta_i in range (0,91,5):
    theta.append(theta_i)
theta[0]=1.0 #si el primer valor de theta fuese cero eso haría al límite inferior la única raíz

# k va de 5e-3 a 1 en saltos de 0.01
k=[]
for k_i in range(0,100,5):
    k.append(k_i*1e-2)
k[0]=5e-3

#================================
#================================

f_n=[]
for k_i in k: #valores de k de 0 a 1
    f_k_i=[]
    for theta_i in theta: #cálculo para ángulos de 0 a 90 grados en saltos de 15, es decir, 0, 15, 30, 45, etc..
        f_theta_i=[]
        for t_i in t: #100 valores de tiempo
            f_theta_i.append(f(t_i,k_i,500,theta_i))
        f_k_i.append(f_theta_i)
    f_n.append(f_k_i)


#-----------------------------------
styles = ['-', '--', '-.', ':']
s=0
colors=['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink', 'tab:olive', 'tab:c
#-----------------------------------

for k_i in k:
    for theta_i in theta:
        plot.plot(t,f_n[k.index(k_i)][theta.index(theta_i)])

#-----------------------------------
plot.axis([-1,150,-2,50])
plot.title("Ec.1 vs. tiempo")
plot.ylabel("Ec.1")
plot.xlabel("tiempo [s]")
#plot.xticks(np.arange(0, 25, step=2))
plot.yticks(np.arange(0, 50, step=5))
#plot.legend()
plot.axhline(y=0, color='k')
plot.grid()
plot.show()
#-----------------------------------
```

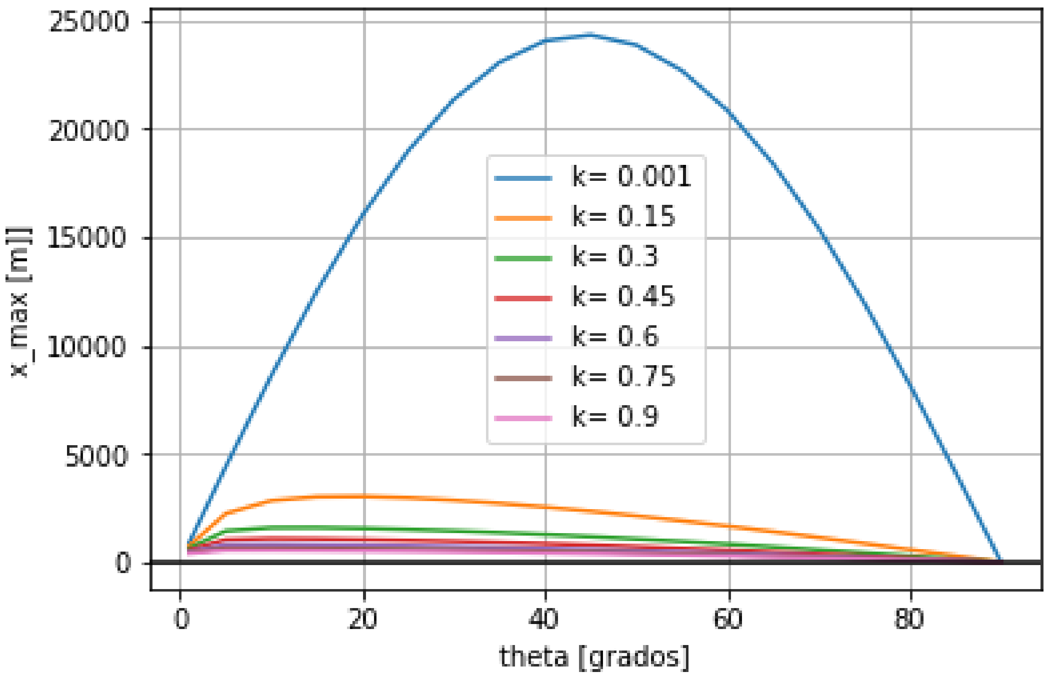
```
print("Con esto hallamos que para valores de k muy pequeños ningún ángulo dado revaza los 100m.")
print("Entonces proponemos como límite 0.1 (ya que 0 es una raíz) y 100")
```



Con esto hallamos que para valores de k muy pequeños ningún ángulo dado revaza los 100m.
Entonces proponemos como límite 0.1 (ya que 0 es una raíz) y 100

In [46]:
```
#Tomaremos menos valores de k para que la visualización de las curvas sea más limpia
k=[]
for k_i in range(0,100,15):
    k.append(k_i*1e-2)
k[0]=1e-3

#Calculando tiempos máximos de caída
T=[]
for k_i in k:
    T_theta_i=[]
    for theta_i in theta:
        T_theta_i.append(newtonRaphson(k_i,0.1,100.0,500.0,theta_i))
    T.append(T_theta_i)


#Evaluando la función de distancia en x en los tiempos de caída máximos
```

```
x_max=[]#2D
for k_i in k:
    x_max_theta_i=[] #1D
    for theta_i in theta:
        T_ij=T[k.index(k_i)][theta.index(theta_i)]
        x_max_theta_i.append(rx(T_ij,k_i,500.00,theta_i))
    x_max.append(x_max_theta_i)


for k_i in k:
        plot.plot(theta, x_max[k.index(k_i)], label="k= "+str(k_i))

#-----------------------------------
#plot.axis([-1,75,-2,50])
#plot.title("Ec.1 vs. tiempo")
plot.xlabel("theta [grados]")
plot.ylabel("x_max [m]]")
#plot.xticks(np.arange(step=10))
#plot.yticks(np.arange(0, 220, step=15))
plot.legend()
plot.axhline(y=0, color='k')
plot.grid()
plot.show()
#-----------------------------------

print("El ángulo que para un k dado maximiza el rango:")
for k_i in k:
        print("para k= "+str(k_i)+"\tes: "+str(theta[x_max[k.index(k_i)].index(max(x_max[k.index(k_i)
```



El ángulo que para un k dado maximiza el rango:
para k= 0.001 es: 45 grados con un rango: 24332.8m,

para k= 0.15 es: 20 grados con un rango: 3038.6m,

para k= 0.3 es: 15 grados con un rango: 1598.2m,

para k= 0.45 es: 10 grados con un rango: 1086.5m,

para k= 0.6 es: 10 grados con un rango: 819.2m,

para k= 0.75 es: 10 grados con un rango: 656.2m,

para k= 0.9 es: 5 grados con un rango: 549.6m,