# Aufgabe 1.1:

def Augabe1():

ε = 1

while 1 != 1 + ε:

ε = ε / 2

return ε \* 2

Epsilon: 2.220446049250313e-16

Python 3.6.2 auf Intel(R) Core(TM) i5-3210M

CPU @ 2,50GHz, 64-Bit

# Aufgabe 1.2:

import numpy as np

import matplotlib.pyplot as plt

def Zentralerdiff(func, x, h):

return (func( x + h ) - func( x - h)) / (2 \* h)

x = 1

n = np.arange(0, 50)

h = 2.\*\*(-n)

numerische\_Ableitung = [Zentralerdiff(np.cos, x, h\_) for h\_ in h]

plt.plot(h, -np.sin(x) - numerische\_Ableitung , "x")

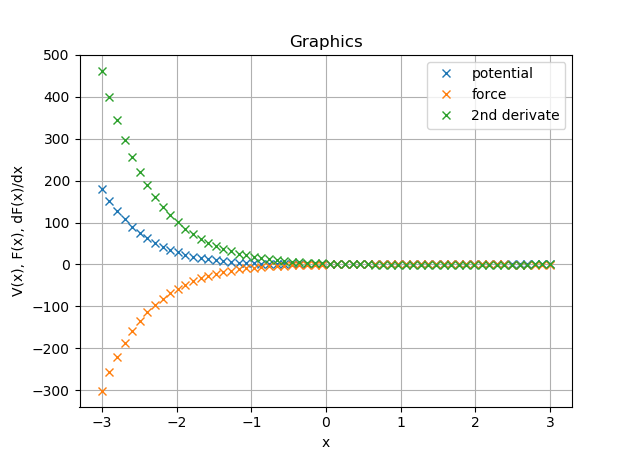
plt.loglog()

plt.show()

Wie bei allen Verfahren sehr gut im Plot zu sehen, erhält man den kleinsten Fehler jeweils nicht beim kleinsten Wert von h im Intervall bis 10^-16. Von 1 bis ca. 10^-5 abfallenden h fällt auch der Fehler der Methode,Zentralerdiffdifferenz, kontinuierlich ab. Doch bei h-Werten kleiner 10^-5, steigt der Fehler in der loglog-Darstellung wieder linear an. Dieser Anstieg kommt hauptsächlich vom numerischen Fehlern der Subtraktion bei x + h. Durch die fehlende Genauigkeit der unterschiedlichen Größenordnungen. Die besten Ergebnisse, den kleinsten Fehler erhällt man bei der Methoden der Zentraldifferenz: h=10^-5 mit einem Fehler=10^-11

# 

# 



# Aufgabe 1.3:

import numpy as np

import math

import matplotlib.pyplot as plt

x = np.linspace(-3,3,60) #array mit datenpunkten

v = x\*x\*math.e\*\*(-x) # Potential, übergabe der datenpunkte lässt v auch zu array werden

vx = (2\*x-x\*\*2)\*math.e\*\*(-x) # erste Ableitung, Kraft

vxx = (x\*\*2-4\*x+2)\*math.e\*\*(-x) # zweite Ableitung

plt.plot(x,v,"x",label='potential')

plt.plot(x,vx,"x",label='force')

plt.plot(x,vxx,"x",label='2nd derivate')

plt.xlabel('x')

plt.ylabel('V(x), F(x), dF(x)/dx')

plt.legend(loc='upper right') # position legende

plt.title('Graphics')

plt.grid(True)

# Gitterlinien

plt.show() # stell alle bisher erwähnten datensätze dar

Nullstellensuche der Kraft ergibt analytisch x=0 und x=2