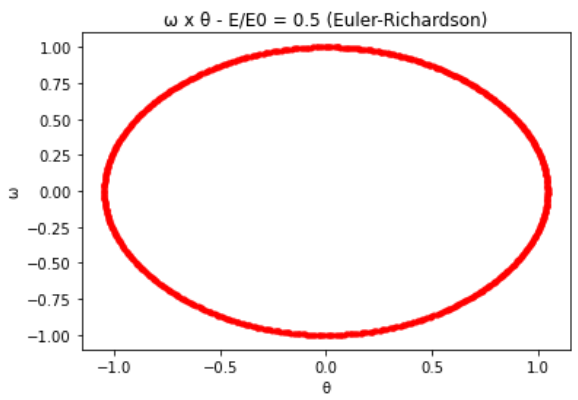
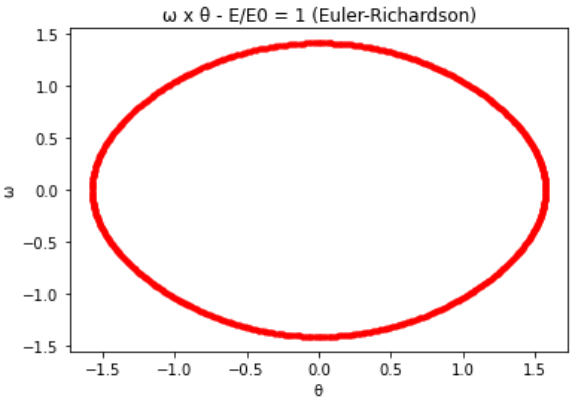
Métodos Computacionais para Física
Geraldo da Costa Siqueira Neto
Atividade 2

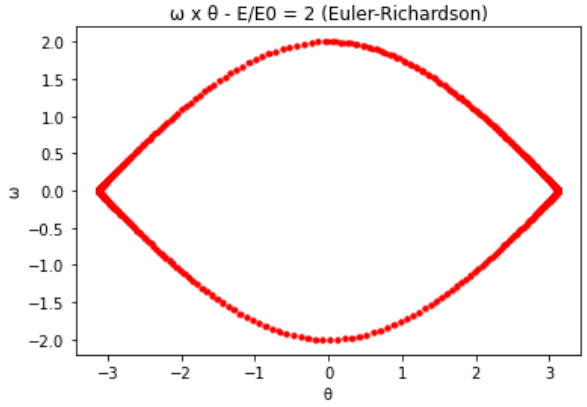
1) Usando a equação do pêndulo simples sem aproximação para ângulos pequenos, vamos analisar os espaços de fase para 4 situações com diferentes valores de energia mecânica: E/E0 = 0.5, 1.0, 2.0 e 3.0 (onde E0 = mgl é a unidade de energia). Esses valores de energia serão obtidos a partir dos parâmetros iniciais, usando a velocidade inicial necessária para obter cada situação. Usaremos dois métodos de integração, o Método de Euler-Richardson e o Método de Verlet, com um dt = 0.05.

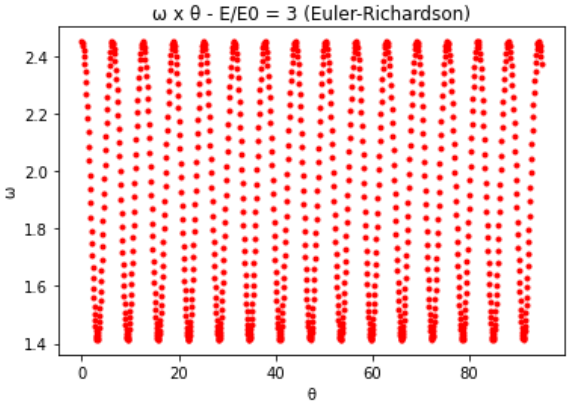
Os gráficos a seguir representam os espaços de fase (ω x θ) para os diferentes métodos. Os códigos utilizados estão no fim do relatório.

Método de Euler-Richard:

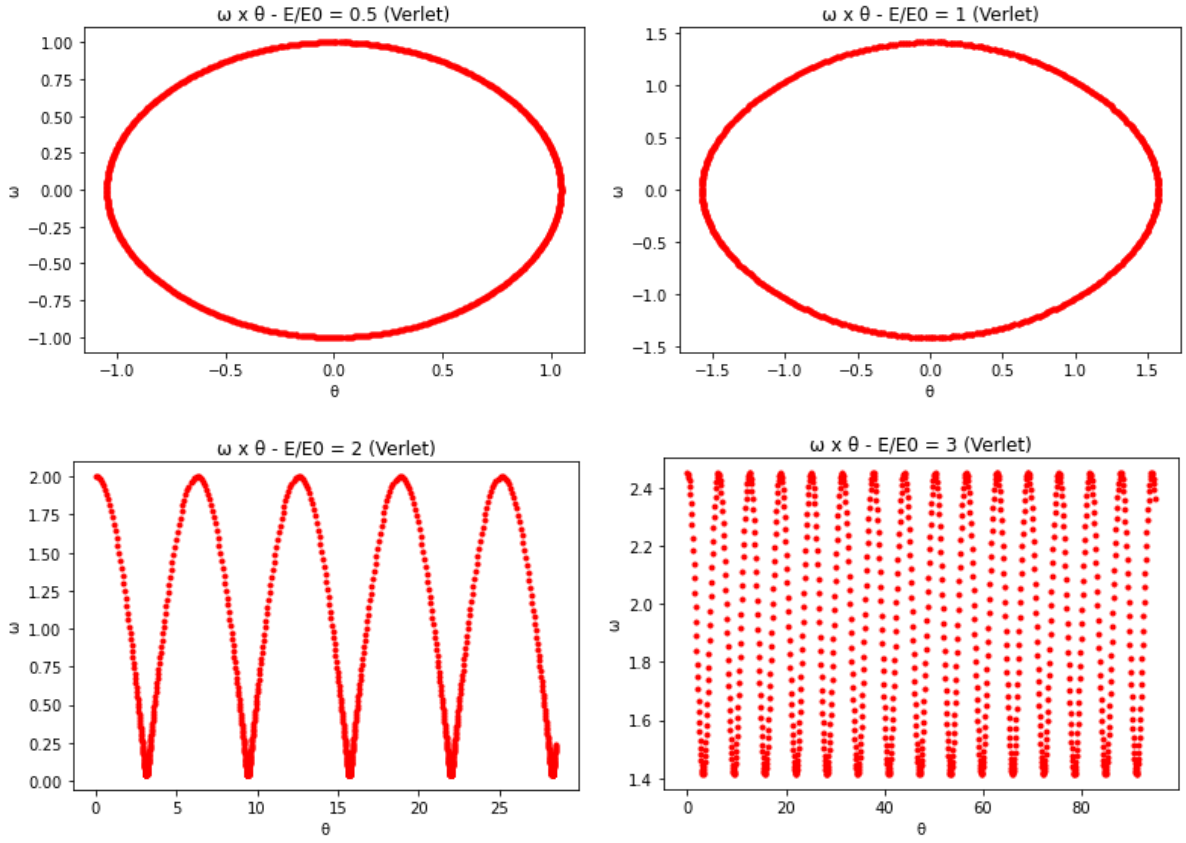








Método de Verlet:



Podemos ver a partir dos gráficos, tanto para Euler-Richardson quanto para Verlet, que abaixo de E/E0 = 2 temos um comportamento estável e cíclico para ω e θ. A partir de E/E0 = 2, mesmo os 2 métodos mostrando gráficos diferentes, é possível ver que esse valor de energia é um valor limítrofe, quando esse valor é atingido, o pêndulo alcança o topo da trajetória, e por isso o formato do seu comportamento no gráfico se altera e é possível ver o fenômeno da libração.

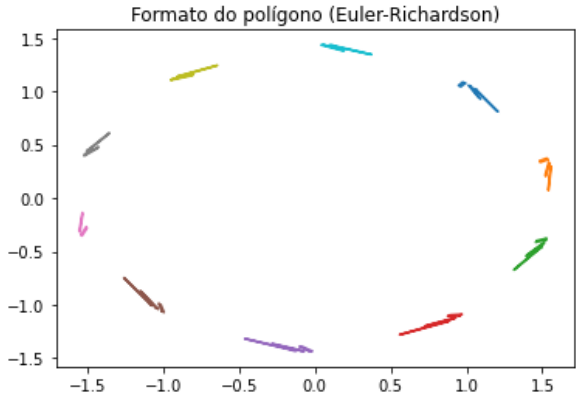
A diferença entre os gráficos de “E/E0 = 2” de Euler-Richardson e de Verlet, apesar das mesmas condições iniciais, se dá por conta da imprecisão numérica. Como os 2 métodos são diferentes, têm imprecisões diferentes, e como estamos tratando de uma situação limítrofe, as imprecisões mostraram comportamentos diferentes. Em um dos gráficos, Euler-Richardson, o pêndulo atinge seu ponto mais alto e logo depois retorna pelo mesmo caminho, dando a volta até atingir o ponto mais alto novamente (libração). Já no gráfico de Verlet, ao atingir o ponto mais alto, o pêndulo continua seguindo na mesma direção, completando assim uma trajetória “circular”.

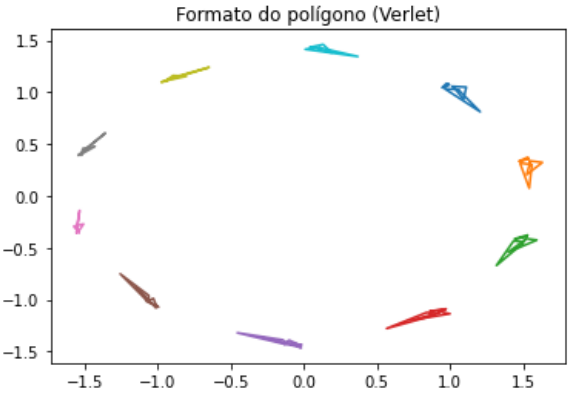
Por fim, os gráficos de E/E0 = 3 mostra o pêndulo com uma solução oscilatória em torno de um ω diferente de zero, então ele passa do ponto máximo da trajetória com uma velocidade maior que zero.

2) Considerando as 4 soluções iniciais em torno de E/E0 = 1 dadas, obtivemos um retângulo inicial. Para analisar a evolução temporal desse retângulo, usaremos 2 ferramentas. Iremos plotar o gráfico desse polígono, para ver se a sua forma se altera com

o passar do tempo e iremos também calcular sua área para cada caso e plotar essa área em função do tempo, para descobrir se esse valor se altera também.

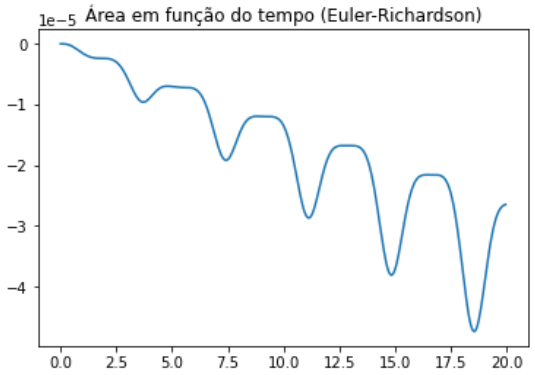
A partir dos métodos de Euler-Richardson e Verlet, chegamos aos seguintes resultados:

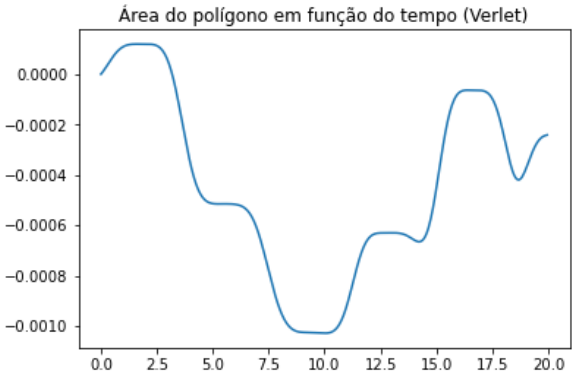




Vemos que são resultados semelhantes para os 2 métodos. Esses gráficos mostram apenas alguns “retratos” de pontos da solução, e nos dois gráficos é possível observar que o formato dos polígonos não se mantém constante, ele se altera com o passar do tempo.

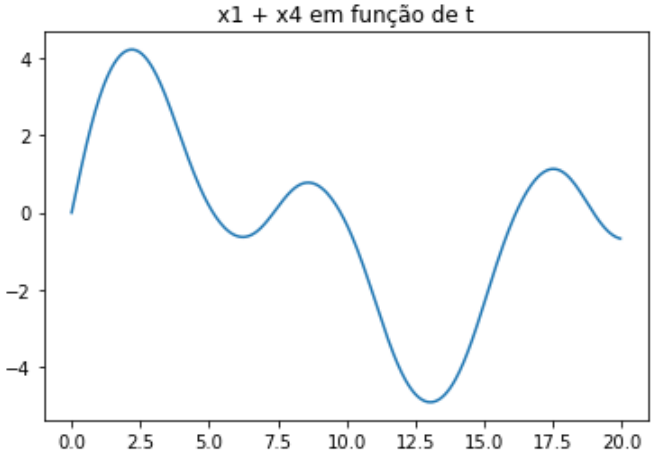
Agora, em questão à área:

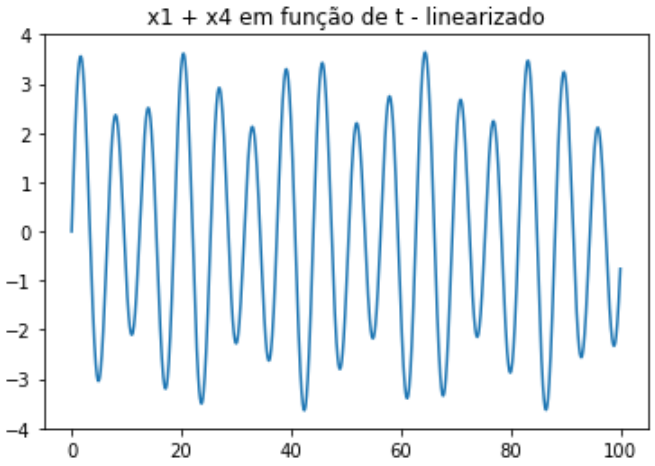




A partir dos gráficos acima, podemos concluir que a área dos polígonos se altera sim, porém de forma muito pequena, cerca de 10^(-5) e 10^(-4). Isso se dá por conta da incerteza numérica proveniente dos métodos de integração computacionais, pois sabemos que essa área deveria se conservar no espaço de fase.

3) Por fim, a partir do Método de Verlet, verificamos se x5 = x1 + x4, é solução da equação. Primeiramente o código foi adaptado para E/E0 = 2 e usamos uma solução para qual E > 2E0 (x1) e outra em que E < 2E0 (x4) e foi plotado o gráfico x5 em função do tempo.





Podemos ver claramente que x5 não é solução do nosso problema pois o gráfico apresentado não se assemelha com nenhuma solução conhecida para o pêndulo simples.

Se fizermos a aproximação linear já podemos ver a formação de um padrão de batimentos a partir da soma das duas soluções.

Logo, podemos concluir que o termo de seno na nossa equação torna o sistema não-linear e por isso não é possível somar duas (ou mais) soluções e formar uma terceira solução a partir disso.

Segue abaixo os 4 códigos utilizados para a resolução da atividade.

- Código de Euler-Richardson utilizado na 1):

```
"""
Created on Fri Jul 15 08:36:56 2022

@author: Geraldo Siqueira
"""

import numpy as np
import matplotlib.pyplot as plt
```

```
g = 1
l = 1

def F(o):
    return -(g/l)*np.sin(o)

dt = 0.05
t = np.arange(0,50, dt)
Nt = t.size
a = np.zeros(Nt)
o = np.zeros(Nt)
w = np.zeros(Nt)

o[0] = 0
w[0] = np.sqrt(6)

#Usando Euler-Richardson
for n in range(1,Nt):
    a[n-1] = F(o[n-1])
    wm = w[n-1] + a[n-1]*(dt/2)
    om = o[n-1] + w[n-1]*(dt/2)
    am = F(om)
    w[n] = w[n-1] + am*dt
    o[n] = o[n-1] + wm*dt

'''
calculando diferentes valores de energia E/E0 = 0.5, 1, 2 e 3
'''

plt.title("ω x θ - E/E0 = 3 (Euler-Richardson)")
plt.ylabel("ω")
plt.xlabel("θ")
plt.plot(o, w, 'r.')
```

- Código de Verlet utilizado para a 1):

```
"""
Created on Fri Jul 15 09:58:12 2022

@author: Geraldo Siqueira
"""
import numpy as np
import matplotlib.pyplot as plt

g = 1
l = 1
```

```
def F(o):
    return -(g/l)*np.sin(o)

dt = 0.05
t = np.arange(0,50, dt)
Nt = t.size
a = np.zeros(Nt)
o = np.zeros(Nt)
w = np.zeros(Nt)

o[0] = 0
w[0] = np.sqrt(6)

#Método de Verlet
for n in range(1,Nt):
    a[n-1] = F(o[n-1])
    o[n] = o[n-1] + w[n-1]*dt + 0.5*a[n-1]*dt**2
    a[n] = F(o[n])
    w[n] = w[n-1] + 0.5*(a[n] + a[n-1])*dt


plt.title("ω x θ - E/E0 = 3 (Verlet)")
plt.ylabel("ω")
plt.xlabel("θ")
plt.plot(o, w, 'r.')
```

- Código de Euler-Richardson utilizado na 2):

```
"""
Created on Fri Jul 15 10:46:51 2022

@author: Geraldo Siqueira
"""

import numpy as np
import matplotlib.pyplot as plt
```

```
g = 1
l = 1

def F(x):
    return -(g/l)*np.sin(x)

dt = 0.05
dx = 0.01
dw = 0.02
t = np.arange(0,20, dt)
Nt = t.size

#x1
a1 = np.zeros(Nt)
x1 = np.zeros(Nt)
w1 = np.zeros(Nt)

x1[0] = 0 + dx
w1[0] = np.sqrt(2) + dw

#Usando Euler-Richardson para x1
for n in range(1,Nt):
    a1[n-1] = F(x1[n-1])
    wm1 = w1[n-1] + a1[n-1]*(dt/2)
    xm1 = x1[n-1] + w1[n-1]*(dt/2)
    am1 = F(xm1)
    w1[n] = w1[n-1] + am1*dt
    x1[n] = x1[n-1] + wm1*dt


'''
'''
#x2
a2 = np.zeros(Nt)
x2 = np.zeros(Nt)
w2 = np.zeros(Nt)

x2[0] = 0 + dx
w2[0] = np.sqrt(2) - dw

#Usando Euler-Richardson para x2
for n in range(1,Nt):
    a2[n-1] = F(x2[n-1])
    wm2 = w2[n-1] + a2[n-1]*(dt/2)
    xm2 = x2[n-1] + w2[n-1]*(dt/2)
    am2 = F(xm2)
    w2[n] = w2[n-1] + am2*dt
    x2[n] = x2[n-1] + wm2*dt
```

```
'''
'''
#x3
a3 = np.zeros(Nt)
x3 = np.zeros(Nt)
w3 = np.zeros(Nt)

x3[0] = 0 - dx
w3[0] = np.sqrt(2) + dw

#Usando Euler-Richardson para x3
for n in range(1,Nt):
    a3[n-1] = F(x3[n-1])
    wm3 = w3[n-1] + a3[n-1]*(dt/2)
    xm3 = x3[n-1] + w3[n-1]*(dt/2)
    am3 = F(xm3)
    w3[n] = w3[n-1] + am3*dt
    x3[n] = x3[n-1] + wm3*dt

'''
'''
#x4
a4 = np.zeros(Nt)
x4 = np.zeros(Nt)
w4 = np.zeros(Nt)

x4[0] = 0 - dx
w4[0] = np.sqrt(2) - dw

#Usando Euler-Richardson para x4
for n in range(1,Nt):
    a4[n-1] = F(x4[n-1])
    wm4 = w4[n-1] + a4[n-1]*(dt/2)
    xm4 = x4[n-1] + w4[n-1]*(dt/2)
    am4 = F(xm4)
    w4[n] = w4[n-1] + am4*dt
    x4[n] = x4[n-1] + wm4*dt



for n in range(1,Nt):
    if n % 15 == 0:
        X=[x1[n],x2[n],x4[n], x3[n], x1[n]]
        W=[w1[n],w2[n],w4[n], w3[n], w1[n]]
        #plt.plot(X,W,'-')

#X=[x1[0],x2[0],x4[0], x3[0], x1[0]]
```

```
#W=[w1[0],w2[0],w4[0], w3[0], w1[0]]


area = 0.5*(x1*w2 + x2*w3 + x3*w4 + x4*w1) - 0.5*(x2*w1 + x3*w2 + x4*w3 + x1*w4)
#plt.plot(t, area)
```

- Código de Verlet utilizado para a 2) e 3):

```
"""
Created on Fri Jul 15 13:42:10 2022

@author: Geraldo Siqueira
"""

import numpy as np
import matplotlib.pyplot as plt

g = 1
l = 1

def F(x):
    return -(g/l)*np.sin(x)

dt = 0.05
dx = 0.01
dw = 0.02
t = np.arange(0,100, dt)
Nt = t.size

#x1
a1 = np.zeros(Nt)
x1 = np.zeros(Nt)
w1 = np.zeros(Nt)

x1[0] = 0 + dx
```

```
w1[0] = np.sqrt(2) + dw

#Usando o Método de Verlet para x1
for n in range(1,Nt):
    a1[n-1] = F(x1[n-1])
    x1[n] = x1[n-1] + w1[n-1]*dt + 0.5*a1[n-1]*dt**2
    a1[n] = F(x1[n])
    w1[n] = w1[n-1] + 0.5*(a1[n] + a1[n-1])*dt



'''
'''
#x2
a2 = np.zeros(Nt)
x2 = np.zeros(Nt)
w2 = np.zeros(Nt)

#x2[0] = 0 + dx
#w2[0] = np.sqrt(2) - dw

#Usando o Método de Verlet para x2
for n in range(1,Nt):
    a2[n-1] = F(x2[n-1])
    x2[n] = x2[n-1] + w2[n-1]*dt + 0.5*a2[n-1]*dt**2
    a2[n] = F(x2[n])
    w2[n] = w2[n-1] + 0.5*(a2[n] + a2[n-1])*dt

'''
'''
#x3
a3 = np.zeros(Nt)
x3 = np.zeros(Nt)
w3 = np.zeros(Nt)

x3[0] = 0 - dx
w3[0] = np.sqrt(2) + dw

#Usando o Método de Verlet para x3
for n in range(1,Nt):
    a3[n-1] = F(x3[n-1])
    x3[n] = x3[n-1] + w3[n-1]*dt + 0.5*a3[n-1]*dt**2
    a3[n] = F(x3[n])
    w3[n] = w3[n-1] + 0.5*(a3[n] + a3[n-1])*dt

'''
'''
#x4
```

```
a4 = np.zeros(Nt)
x4 = np.zeros(Nt)
w4 = np.zeros(Nt)

x4[0] = 0 - dx
w4[0] = np.sqrt(2) - dw

#Usando o Método de Verlet para x1
for n in range(1,Nt):
    a4[n-1] = F(x4[n-1])
    x4[n] = x4[n-1] + w4[n-1]*dt + 0.5*a4[n-1]*dt**2
    a4[n] = F(x1[n])
    w4[n] = w4[n-1] + 0.5*(a4[n] + a4[n-1])*dt



for n in range(1,Nt):
    if n % 15 == 0:
        X=[x1[n],x2[n],x4[n], x3[n], x1[n]]
        W=[w1[n],w2[n],w4[n], w3[n], w1[n]]
        #plt.plot(X,W,'-')

#X=[x1[0],x2[0],x4[0], x3[0], x1[0]]
#W=[w1[0],w2[0],w4[0], w3[0], w1[0]]


area = 0.5*(x1*w2 + x2*w3 + x3*w4 + x4*w1) - 0.5*(x2*w1 + x3*w2 + x4*w3 + x1*w4)
#plt.plot(t, area)

x5 = x1 + x4
plt.plot(t, x5)
```