Atividade 1 - Métodos Computacionais para Física

Geraldo da Costa Siqueira Neto

1) Eq. (14): $x_{n+1} = 2x_n - x_{n-1} + a_n \Delta t^2$

Eq. (15): $v_n = \frac{x_{n+1} - x_{n-1}}{2\Delta t}$

Eq. (16): $x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n \Delta t^2$

Eq. (17): $v_{n+1} = v_n + \frac{1}{2}(a_{n+1} + a_n)\Delta t$

i) Mostrando a equivalência de (16):

$$x_{n+1} = 2x_n - x_{n-1} + a_n \Delta t^2$$

Usando (15), substituímos x_{n-1}

$$x_{n-1} = 2v_n \Delta t - x_{n+1}$$

$$x_{n+1} = 2x_n + 2v_n \Delta t - x_{n+1} + a_n \Delta t^2$$

$$2x_{n+1} = 2x_n + 2v_n \Delta t + a_n \Delta t^2 \quad \times 2$$

$$\boxed{x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n \Delta t^2}$$

ii) Mostrando a equivalência de (17):

Partindo de (15)

$$v_n = \frac{x_{n+1} - x_{n-1}}{2\Delta t} \quad ; \text{ substituindo (16) e (14) em (15):}$$

$$v_n = (x_n + v_n \Delta t + \frac{1}{2} a_n \Delta t^2 + x_{n+1} - 2x_n - a_n \Delta t^2)/2\Delta t$$

Usando (15) para x_{n+2}: $x_{n+2} = x_{n+1} + v_{n+1}\Delta t + \frac{1}{2} a_{n+1} \Delta t^2$

$$x_{n+1} = x_{n+2} - v_{n+1}\Delta t - \frac{1}{2} a_{n+1} \Delta t^2$$

$$v_n = (x_n + v_n \Delta t - \frac{1}{2} a_n \Delta t^2 + x_{n+2} - v_{n+1}\Delta t - \frac{1}{2} a_{n+1} \Delta t^2)/2\Delta t$$

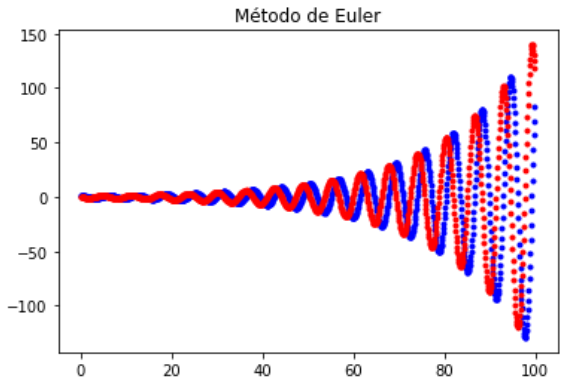
$$v_n = v_{n+1} + \frac{v_n \Delta t - \frac{1}{2}(a_n + a_{n+1})\Delta t^2 - v_{n+1}\Delta t}{2\Delta t}$$

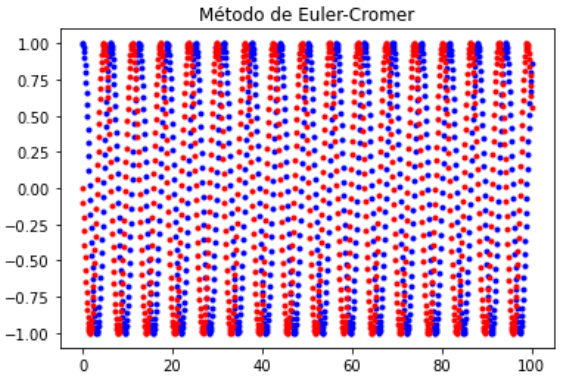
$$v_{n+1} - \frac{v_{n+1}}{2} = v_n - \frac{v_n}{2} + \frac{1}{4}(a_n + a_{n+1})\Delta t$$

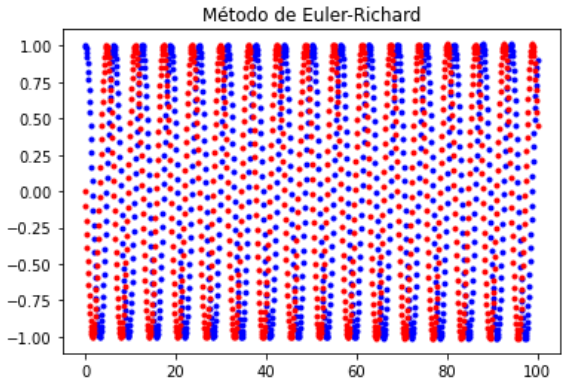
$$\boxed{v_{n+1} = v_n + \frac{1}{2}(a_n + a_{n+1})\Delta t}$$

2) Seguindo o programa que se encontra mais a frente neste relatório, usamos 3 métodos para calcular as funções: o Método de Euler, o Método de Euler-Cromer e o Método de Euler-Richard. Resolvendo o problema para um oscilador simples, tivemos os seguintes resultados:

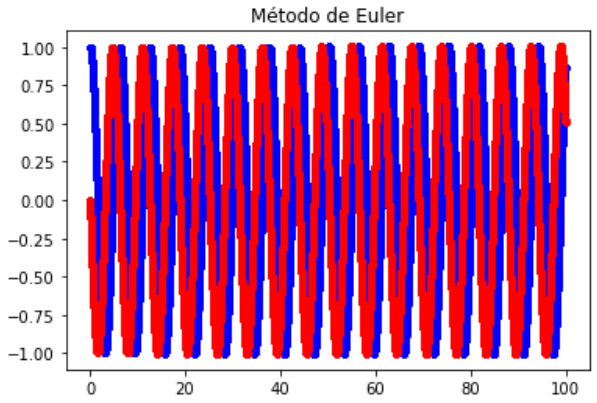
Onde o azul mostra a posição e o vermelho corresponde à velocidade.
Para um dt baixo, por exemplo, 0.1 s:

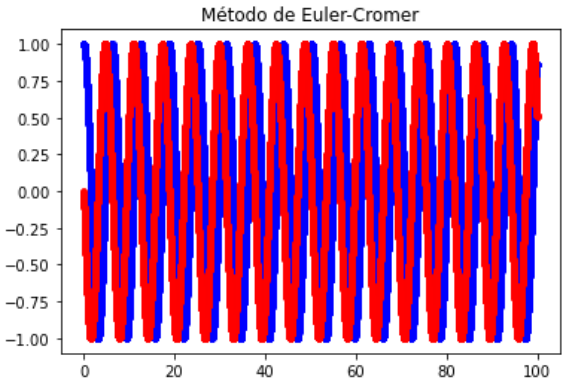


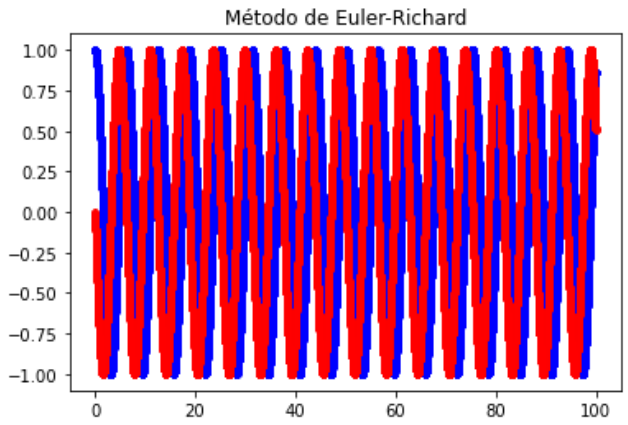




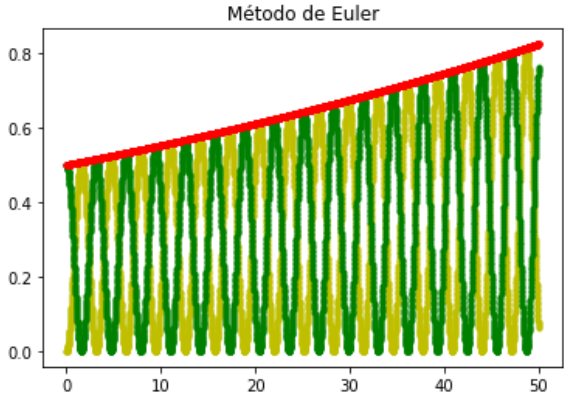
Já, para um dt mais preciso, como 0.001 s, obtemos:

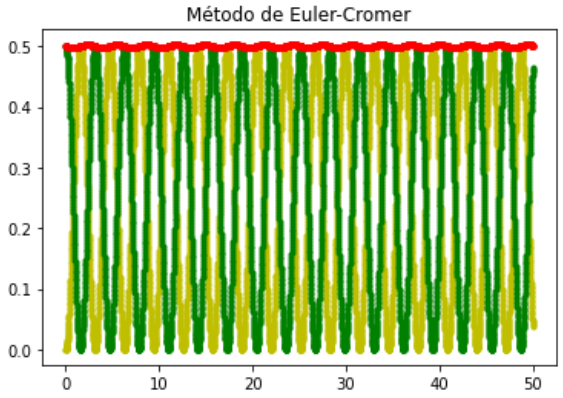


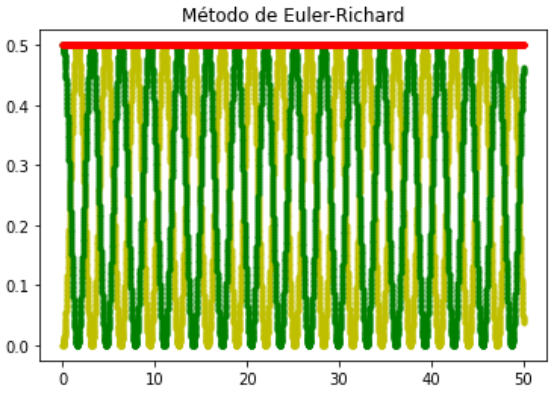




Então, é possível observar que com intervalos de tempo grandes, o método de euler deixa a desejar, pois o erro se acumula de forma que a solução 'explode'. Porém, para um dt pequeno, a partir de 0.001 s, já conseguimos enxergar um comportamento mais desejado. Comparando o resultado dos métodos de Euler-Cromer e Euler-Richard, visualmente os gráficos de 0.1 e 0.001 são similares e satisfatórios à primeira vista. Podemos também observar as energias relacionadas.







Onde: E - vermelho; K - amarelo; U - verde.

Pelo gráfico das energias (dt = 0.01 s), já conseguimos ver que Euler-Richard tem um resultado melhor, por conta da constância de E. Já Euler é completamente inconstante.

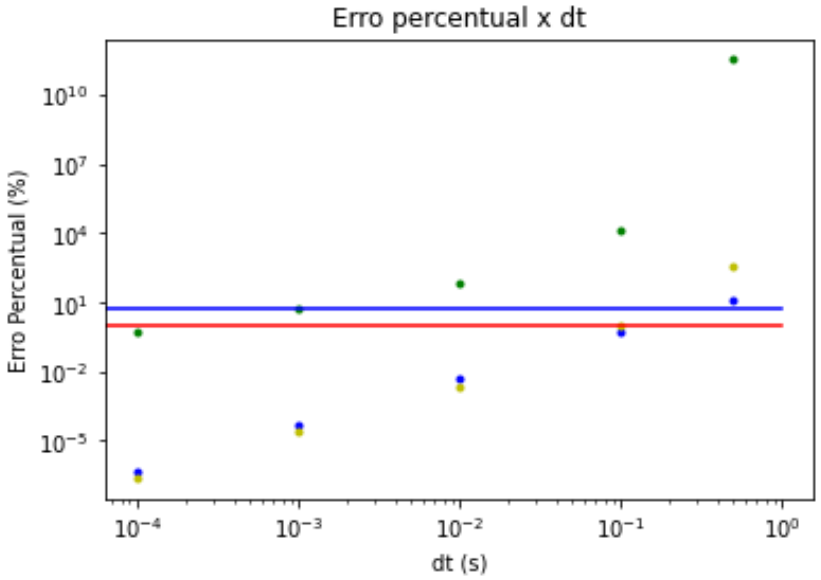
Segue o código utilizado:

```
"""
Created on Fri Jul  1 13:19:16 2022

@author: Geraldo Siqueira
"""

import numpy as np
import matplotlib.pyplot as plt

k = 1
m = 1

def F(x):
    return -k*x

dt = 0.01
t = np.arange(0,50, dt)
Nt = t.size
a = np.zeros(Nt)
x = np.zeros(Nt)
v = np.zeros(Nt)
Amp_e = np.array([])
Amp_ec = np.array([])
Amp_er = np.array([])

x[0] = 1
v[0] = 0

#Méodo de Euler
for n in range(1,Nt):
    a[n-1] = F(x[n-1])/m
    v[n] = v[n-1] + a[n-1]*dt
    x[n] = x[n-1] + v[n-1]*dt
    if v[n]*v[n-1]<0:
        Amp_e = np.append(Amp_e, (x[n]+x[n-1])/2)


plt.title("Método de Euler")
plt.plot(t, x, 'b.')
plt.plot(t, v, 'r.')


erro_euler = np.abs(np.abs(Amp_e) - 1)
plt.plot(erro_euler, 'g.')
print(np.max(erro_euler)*100)

#plt.plot(v, x, 'r.')


#Méodo de Euler-Cromer
```

```
for n in range(1,Nt):
    a[n-1] = F(x[n-1])/m
    v[n] = v[n-1] + a[n-1]*dt
    x[n] = x[n-1] + v[n]*dt
    if v[n]*v[n-1]<0:
        Amp_ec = np.append(Amp_ec, (x[n]+x[n-1])/2)

plt.title("Método de Euler-Cromer")
plt.plot(t, x, 'b.')
plt.plot(t, v, 'r.')


erro_eulerc = np.abs(np.abs(Amp_ec) - 1)
plt.plot(erro_eulerc, 'b.')
print(np.max(erro_eulerc)*100)

#plt.plot(v, x, 'g.')



#Méodo de Euler-Richardon
for n in range(1,Nt):
    a[n-1] = F(x[n-1])/m
    vm = v[n-1] + a[n-1]*(dt/2)
    xm = x[n-1] + v[n-1]*(dt/2)
    am = F(xm)/m
    v[n] = v[n-1] + am*dt
    x[n] = x[n-1] + vm*dt
    if v[n]*v[n-1]<0:
        Amp_er = np.append(Amp_er, (x[n]+x[n-1])/2)


plt.title("Método de Euler-Richard")
erro_eulerr = np.abs(np.abs(Amp_er) - 1)
plt.plot(erro_eulerr, 'r.')
print(np.max(erro_eulerr)*100)

#plt.plot(v, x, 'b.')

K = 0.5*v*v
U = 0.5*  k*x*x
E = K+U

plt.plot(t, K, 'y.')
plt.plot(t, U, 'g.')
plt.plot(t, E, 'r.')
```

3) Para melhor análise, será necessário calcular os erros relativos a cada método. Foi utilizada a amplitude para esse cálculo, pois é um parâmetro fácil de ser analisado e, com as condições iniciais certas, é possível saber o valor esperado exato da amplitude do sistema.

O gráfico a seguir mostra o erro de cada método em relação ao dt utilizado. Foi usada a escala log x log para melhor visualização.



- Euler: verde
- Euler-Cromer: azul
- Euler-Richard: amarelo
- As linhas vermelha e azul representam respectivamente as marcas de 1% de erro e 5% de erro.

Podemos ver claramente que, quanto maior o dt, maior o erro, como é esperado. Também podemos ver a qualidade de cada método em relação a cada dt, e assim, a eficiência do código. Considerando o número de linhas de código necessárias para realizar cada método dentro da interação do '*for*' e o valor do erro, podemos escolher que método utilizar. Por exemplo, para um dt = 0.0001, o método de Euler já tem uma precisão menor que 1%, então, se for essa a precisão desejada, ele seria o método mais eficiente, apesar de não ser o mais preciso. Já para um dt maior, como dt = 0.1, o método de Euler não é útil, pois acumula erros de forma que os valores finais são incoerentes.

Os métodos de Euler-Cromer e Euler-Richard têm precisões muito parecidas, sendo o Euler-Richard um pouco mais preciso para intervalos menores e o Euler-Cromer para intervalos maiores. Logo, além do dt, devemos olhar as linhas de código para determinar a eficiência. Com esse critério o Método de Euler-Cromer se torna o mais eficiente, por ter o laço menor. Então, na hora de escolher um método para ser utilizado, há diversos fatores a se considerar, dependendo da precisão e da velocidade desejada, e da memória da máquina disponível.

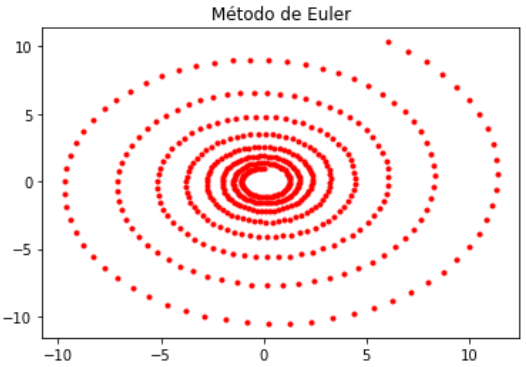
A partir do gráfico é possível ver também que o dt = 0.5 não é satisfatório para nenhum dos métodos, caso seja desejada uma tolerância menor que 5%, que é uma

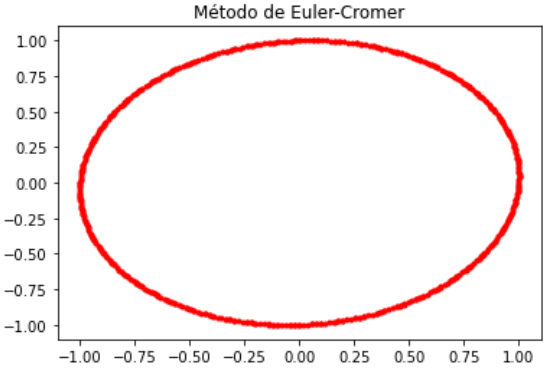
tolerância relativamente alta. Isso se dá porque o intervalo de tempo de 0.5s é muito grande para garantir qualquer tipo de precisão.

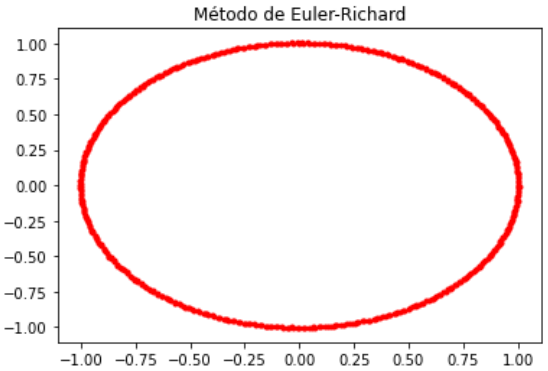
- Tolerância de 5%:
  - Euler: a partir de dt = 0.001
  - Euler-Cromer: a partir de dt = 0.1
  - Euler-Richard: a partir de dt = 0.1
- Tolerância de 1%:
  - Euler: a partir de dt = 0.0001
  - Euler-Cromer: a partir de dt = 0.1
  - Euler-Richard: a partir de dt = 0.1

Analisando agora a estabilidade de cada método a partir dos gráficos no espaço de fase (v × x), temos que, para um sistema estável, esse gráfico deve ter o formato de um círculo, pois a energia se conserva e a velocidade e o espaço oscilam em torno de um ponto central. Temos o espaço oscilando de -A até A nas extremidades, e a velocidade alcançando 0 (zero) nas extremidades e seus valores máximos e mínimos no ponto central do movimento.

Espaço de fase para dt = 0.1s:



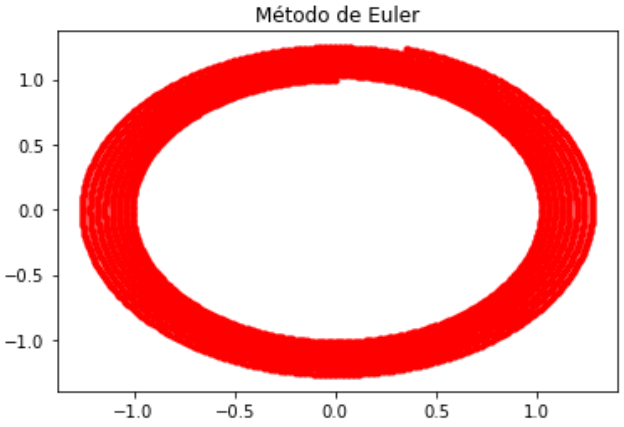




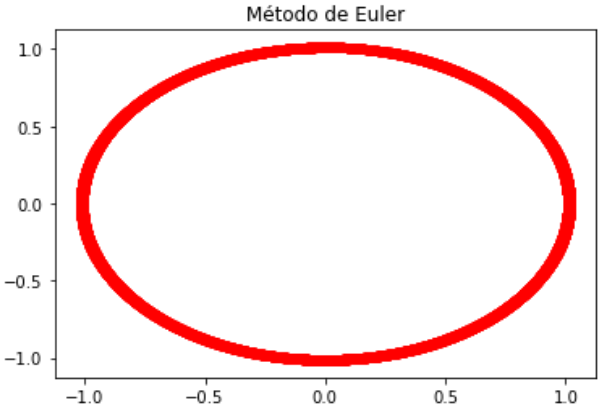
:

Podemos ver que os métodos de Euler-Cromer e Euler-Richard são estáveis mesmo com um dt alto (0.1), já o método de Euler mostra clara instabilidade, espiralando para fora.

Usando agora um dt menor, para procurar um intervalo de tempo em que Euler atinja certa estabilidade e seja viável.

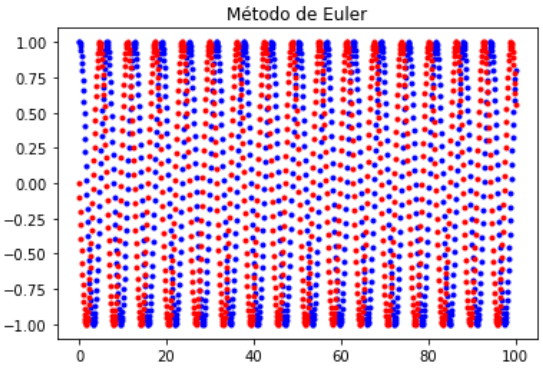


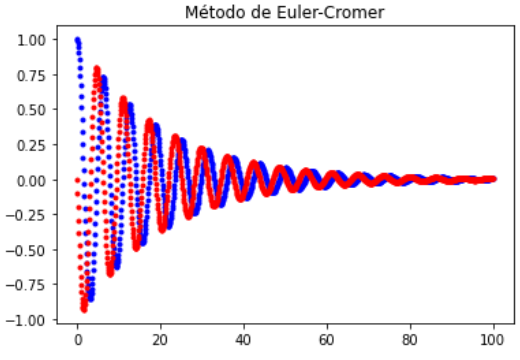
Para um dt = 0.01 s, podemos ver que a forma já se assemelha mais a um círculo, mas ainda sim há um caráter de espiral. Ainda não é um método estável.

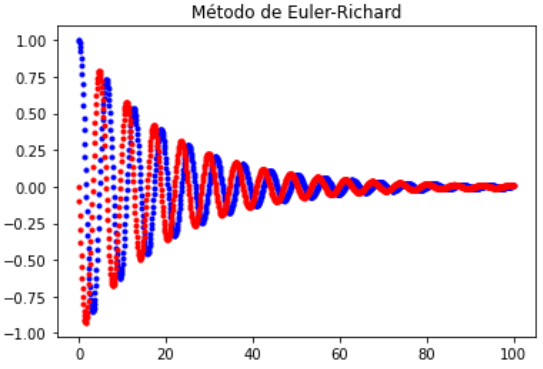


Agora, para um dt = 0.001 s, o método passa a apresentar um comportamento de conservação de energia e estabilidade, diferente do que mostra em intervalos de tempo maiores.

4) Agora, adicionando um termo dissipativo no sistema, obteremos um oscilador harmônico amortecido. Os gráficos a seguir representam o sistema em função do tempo, onde o azul mostra a posição e o vermelho corresponde à velocidade. Para um dt = 0.1s, temos:



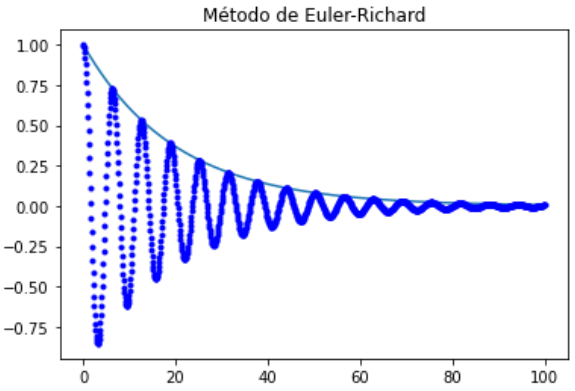




Assim como no oscilador simples, podemos ver que o comportamento de Euler-Cromer e Euler-Richard são coerentes com o sistema analisado e o comportamento de Euler não mostra um resultado satisfatório, apresentando agora um gráfico que se assemelha a um oscilador simples.

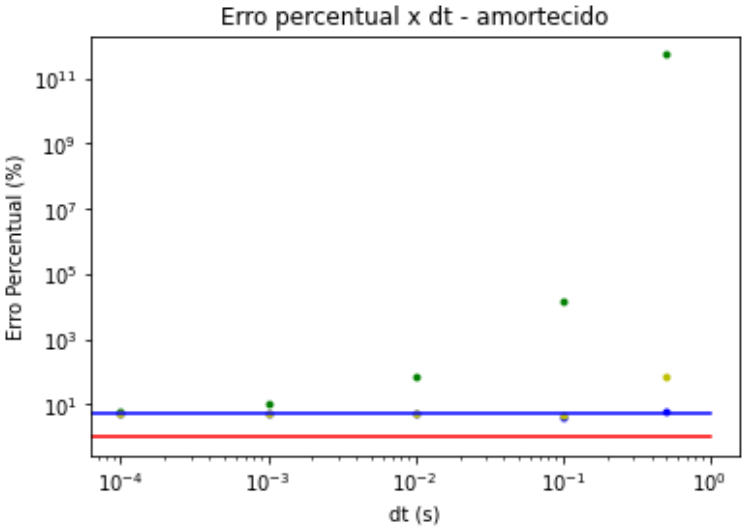
Em relação ao erro, o cálculo relativo ao oscilador amortecido é mais complexo que o oscilador harmônico simples. Uma das grandes dificuldades foi achar uma forma mais eficiente e fácil de calcular esse erro. No caso simples, temos uma amplitude que varia entre -A e A, e o máximo e mínimo não variam com o passar do tempo. Já para o caso com viscosidade, temos um termo dissipativo como parte da solução do sistema, então, essa amplitude varia a cada oscilação. A solução encontrada para calcular o erro, foi usar também a amplitude, mas como ela não é constante, foi preciso achar uma função que mostre o comportamento desses máximos (e mínimos) de amplitude.

Como a solução exata para esse sistema tem um termo dissipativo da forma de exp((-b/2)*t), onde b é o coeficiente de viscosidade, podemos usar essa curva para representar o valor dos pontos de amplitude máxima do sistema, e usaremos esses pontos para comparar com os pontos de amplitudes mostrados por cada método.



A curva representada pelo azul claro mostra o termo dissipativo da solução. E é a partir dela que serão comparados os máximos e mínimos de amplitude. Os valores de máximos e mínimos serão usados em módulo, para que possamos usar apenas uma curva de parâmetro.

O gráfico a seguir mostra o erro de cada método em relação ao dt utilizado. Foi usada a escala log x log para melhor visualização.



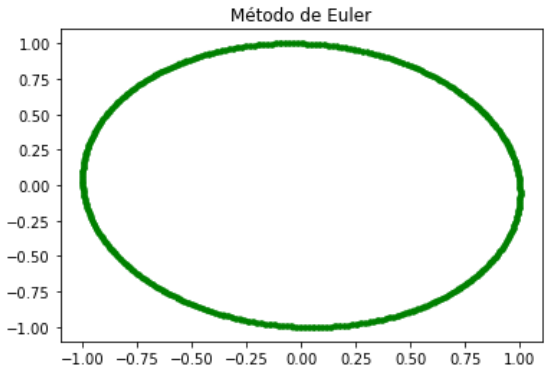
- Euler: verde
- Euler-Cromer: azul
- Euler-Richard: amarelo
- As linhas vermelha e azul representam respectivamente as marcas de 1% de erro e 5% de erro.

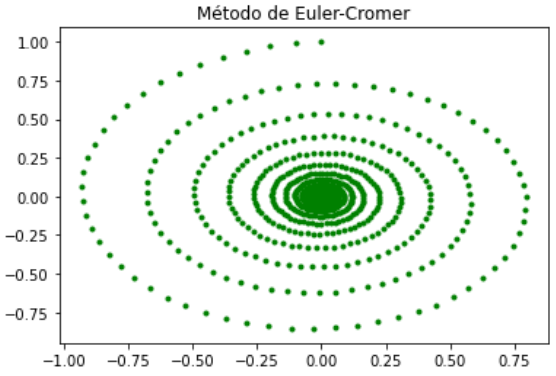
Se comparado com o gráfico do erro relativo do oscilador simples, é possível perceber que com a diminuição do intervalo de tempo, a tolerância não diminui tanto quanto no simples, pois o sistema atual tem um termo dissipativo que o anterior não tinha, adicionando assim mais uma possível fonte de imprecisão no modelo.

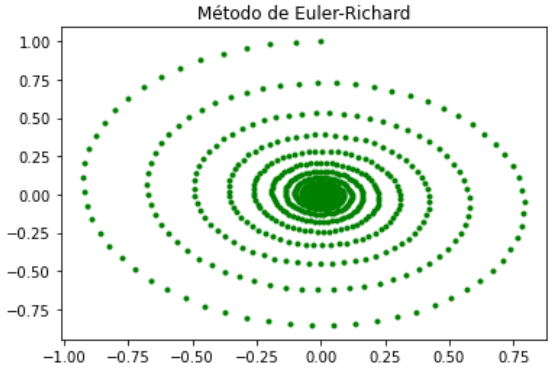
Assim como o oscilador simples, o método de euler só é viável para um dt = 0.0001, e ainda assim, com um erro relativo de aproximadamente 5.63%. Já Euler-Cromer e Euler-Richard são viáveis a partir de dt = 0.1, tendo uma precisão acima de 5% apenas com

dt = 0.5. Porém, ao contrário do sistema anterior, é possível observar que nenhum dos métodos atinge uma tolerância menor que 1% para esse sistema.

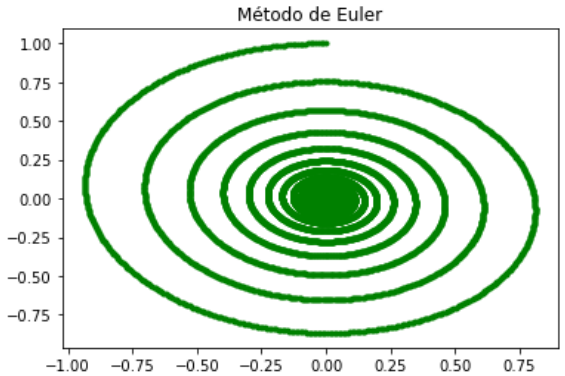
Para a análise da estabilidade a partir do espaço de fase, é esperado que o sistema se comporte como uma espiral para dentro, por conta da dissipação de energia provocada pela viscosidade. Mas é possível ver que o método de Euler não se comporta dessa forma para um dt grande (0.1s). Já Euler-Cromer e Euler-Richard já satisfazem o esperado.







Com um dt = 0.01, já é possível ver Euler mostrando um comportamento mais condizente com o sistema representado.



5) Como visto anteriormente, os diferentes métodos têm diferentes vantagens e desvantagens, o método de Euler é inconstante e têm muito erro em grande parte das situações, mas até com o dt correto é possível de ser utilizado de forma eficiente. Já a escolha de utilização de Euler-Cromer e Euler-Richard são mais voltadas à sua optimização e velocidade que à seu erro, pois são muito precisos quando comparados com Euler, mas usam mais linhas de código e, consequentemente, mais memória da máquina.

Segue abaixo o código utilizado para o cálculo do oscilador amortecido:

```
"""
Created on Fri Jul  1 13:19:16 2022

@author: Geraldo Siqueira
"""

import numpy as np
import matplotlib.pyplot as plt

k = 1
m = 1
ni = 0.1

def F(x, v):
    return -k*x - ni*v

dt = 0.1
t = np.arange(0,100, dt)
Nt = t.size
a = np.zeros(Nt)
x = np.zeros(Nt)
v = np.zeros(Nt)
Amp_e = np.array([])
Amp_ec = np.array([])
Amp_er = np.array([])
tempo_er = np.array([])
tempo_ec = np.array([])
tempo_e = np.array([])

x[0] = 1
v[0] = 0


#Méodo de Euler
for n in range(1,Nt):
    a[n-1] = F(x[n-1], v[n-1])/m
    v[n] = v[n-1] + a[n-1]*dt
    x[n] = x[n-1] + v[n-1]*dt
    if v[n]*v[n-1]<0:
        Amp_e = np.append(Amp_e, (x[n]+x[n-1])/2)
        tempo_e = np.append(tempo_e, 1+(n*dt))
```

```
#plt.title("Método de Euler")

plt.plot(t, x, 'b.')
plt.plot(t, v, 'r.')


tempo2_e = np.exp(-(0.1/2)*tempo_e)
erro_euler = np.abs(np.abs(Amp_e) - tempo2_e)/tempo2_e
plt.plot(erro_euler, 'g.')
print(np.max(erro_euler)*100)


#plt.plot(v, x, 'g.')


#Méodo de Euler-Cromer
for n in range(1,Nt):
    a[n-1] = F(x[n-1], v[n-1])/m
    v[n] = v[n-1] + a[n-1]*dt
    x[n] = x[n-1] + v[n]*dt
    if v[n]*v[n-1]<0:
        Amp_ec = np.append(Amp_ec, (x[n]+x[n-1])/2)
        tempo_ec = np.append(tempo_ec, 1+(n*dt))


#plt.title("Método de Euler-Cromer")

plt.plot(t, x, 'b.')
plt.plot(t, v, 'r.')


tempo2_ec = np.exp(-(0.1/2)*tempo_ec)
erro_eulerc = np.abs(np.abs(Amp_ec) - tempo2_ec)/tempo2_ec
plt.plot(erro_eulerc, 'b.')
print(np.mean(erro_eulerc)*100)

#plt.plot(v, x, 'g.')


#Méodo de Euler-Richardon
for n in range(1,Nt):
    a[n-1] = F(x[n-1], v[n-1])/m
    vm = v[n-1] + a[n-1]*(dt/2)
    xm = x[n-1] + v[n-1]*(dt/2)
    am = F(xm, vm)/m
    v[n] = v[n-1] + am*dt
    x[n] = x[n-1] + vm*dt
    if v[n]*v[n-1]<0:
```

```
        Amp_er = np.append(Amp_er, (x[n]+x[n-1])/2)
        tempo_er = np.append(tempo_er, 1+(n*dt))


#z = -np.exp(-(0.1/2)*t)
#plt.plot(t,z)
ze = np.exp(-(0.1/2)*t)
plt.plot(t,ze)



plt.title("Método de Euler-Richard")

plt.plot(t, x, 'b.')
#plt.plot(t, v, 'r.')


tempo2_er = np.exp(-(0.1/2)*tempo_er)
erro_eulerr = np.abs(np.abs(Amp_er) - tempo2_er)/tempo2_er
#plt.plot(erro_eulerr, 'r.')
print(np.mean(erro_eulerr)*100)

plt.plot(v, x, 'g.')


K = 0.5*v*v
U = 0.5*k*x*x
E = K+U


'''
plt.plot(t, K, 'y.')
plt.plot(t, U, 'g.')
plt.plot(t, E, 'r.')
'''
```