EP4 - Cálculo Numérico

Professor: Arnaldo Gammal
Monitor: Raphael

Aluno: Jhoão Gabriel Martins Campos de Almeida Arneiro
Nº USP: 10819721

Parte I

Questão 1

O nosso problema se consiste em resolver a seguinte equação diferencial ordinária

$$\ddot{y} = 12t^2 + 4t^3 - t^4 + y - \dot{y}$$

Com condições iniciais dadas por $y(0) = 0$ e $\dot{y}(0) = 0$. Podemos calcular sua solução analítica assumindo que ela seja da forma

$$y = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4$$

Assim

$$\dot{y} = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3$$

e

$$\ddot{y} = 2a_2 + 6a_3 t + 12a_4 t^2$$

Manipulando a nossa equação e então inserindo estes valores chegamos a

$$\ddot{y} + \dot{y} - y = 12t^2 + 4t^3 - t^4$$

$$\Rightarrow 2a_2 + 6a_3 t + 12a_4 t^2 + a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 - (a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4) = 12t^2 + 4t^3 - t^4$$

Colecionando os termos proporcionais a cada uma das potência de t

$$\Rightarrow (2a_2 + a_1 - a_0) + (6a_3 + 2a_2 - a_1)t + (12a_4 + 3a_3 - a_2)t^2 + (4a_4 - a_3)t^3 - a_4 t^4 = 12t^2 + 4t^3 - t^4$$

Isto nos dá o seguinte sistema de equações

$$\begin{cases} -a_0 + a_1 + 2a_2 = 0 \\ -a_1 + 2a_2 + 6a_3 = 0 \\ -a_2 + 3a_3 + 12a_4 = 12 \\ -a_3 + 4a_4 = 4 \\ -a_4 = -1 \end{cases}$$

É fácil ver que $a_4 = 1$, de modo que $a_3 = 4 - 4a_4 = 0$, $a_2 = -12 + 3a_3 + 12a_4 = 0$, $a_1 = 2a_2 + 6a_3 = 0$ e $a_0 = a_1 + 2a_2 = 0$, e portanto, nossa solução é simplesmente $y = t^4$ (perceba que esta é uma equação diferencial ordinária não homogênea, e portanto nós deveríamos ter começado nossa solução partindo de um caso onde $\ddot{y} + \dot{y} - y = 0$, contudo a solução homogênea desta equação para as nossas condições iniciais, apenas nos daria que $y_h = 0$, de modo que a solução final é igual à solução particular). Com este resultado é fácil ver que $\dot{y} = 4t^3$, de modo que $y(5) = (5)^4 = 625$ e $\dot{y}(5) = 4(5)^3 = 500$.

Queremos agora calcular os valores de $y(5)$ e $\dot{y}(5)$ numericamente. Para isso, utilizaremos dois métodos diferentes, o método de Euler e o método de Runge-Kutta de 4a ordem.

O método de Euler consiste em transformar a equação diferencial em uma equação de diferenças. Se denotarmos $\dot{y}$ por $f(t, y)$, então

$$\frac{dy}{dt} \simeq \frac{y(t + \Delta t) - y(t)}{\Delta t} = f(y, t)$$

$$\Rightarrow y(t + \Delta t) = y(t) + \Delta t f(y, t)$$

Deste modo, se denotarmos Δt por h, então nós podemos definir um passo do método de Euler como sendo dado por

$$y_{i+1} = y_i + h f(y_i, t)$$

Em nosso caso, nós devemos aplicar um "passo de Euler"tanto para obter o valor futuro de y, quanto outro para obter o valor futuro de $z = \dot{y}$, sendo assim, se nós denotarmos $\dot{z}$ por $\dot{z} = g(t, y, z)$, nós teremos dois passos de Euler, dados por

$$y_{i+1} = y_i + h z_i$$

e

$$z_{i+1} = z_i + h g(t, y_i, z_i)$$

Construímos então o seguinte programa para implementar o método:

```
from numpy import arange

def g(t,y,z):
    ''' A partir dos atuais valores de t, y(t) e y'(t) calcula o valor de
```

```
        y''(t) '''

    g = 12*t**2 + 4*t**3 - t**4 + y - z

    return g

def passo_Euler(t,y,z,h):
    ''' Dados os valores de t, y, z = y' e h, calcula pelo metodo de Euler,
        os valores de y(t+h) e z(t+h) '''

    passo_y, passo_z = y + h*z, z + h*g(t,y,z)

    return passo_y, passo_z

#Condicoes iniciais
y0 = 0
z0 = 0

#Tamanho do passo
h = 0.01

#Iniciamos nossas variaveis y e z pelos seus valores iniciais
y = y0
z = z0

#Implementamos o metodo de Euler indo de t = 0 ate t = 5 a passos de h
for t in arange(0,5,h):

    #Para cada loop, atualizamos os valores de y e z
    y, z = passo_Euler(t,y,z,h)

#Printamos por fim os valores obtidos
print("y(5) = %f, z(5) = %f" %(y,z))

>>>y(5) = 617.669124, z(5) = 495.593364
```

Como podemos ver, pelo método de Euler, a um passo de $h = 0.01$, os resultados que obtemos foram de

$$y(5) = 617.669124 e \dot{y}(5) = 495.593364,$$

nos dando uma diferença de

$$\Delta y = y_{exato} - y_{Euler} = 7.330876 e \Delta \dot{y} = \dot{y}_{exato} - \dot{y}_{Euler} = 4,406636$$

dos valores esperados. Contudo, se realizássemos uma correção no tamanho do nosso passo h, diminuindo ele para $h = 0.001$, já conseguiríamos resultados bem melhores, obtendo

$$y(5) = 624.261285 e \dot{y}(5) = 499.555833$$

Já o método de Runge-Kutta de 4a ordem funciona por uma atualização das nossas variáveis na seguinte forma:

$$y_{i+1} = y_i + \frac{k_{1y} + 2(k_{2y} + k_{3y}) + k_{4y}}{6}$$

$$z_{i+1} = z_i + \frac{k_{1z} + 2(k_{2z} + k_{3z}) + k_{4z}}{6}$$

Onde definimos as variáveis k_{nj}, com $n = 1, 2, 3, 4$ e $j = y, z$, dadas por

$$\begin{aligned}
k_{1y} &= hz_i \\
k_{2y} &= h(z_i + \frac{k_{1z}}{2}) \\
k_{3y} &= h(z_i + \frac{k_{2z}}{2}) \\
k_{4y} &= h(z_i + k_{3z}) \\
k_{1z} &= hg(t_i, y_i, z_i) \\
k_{2z} &= hg(t_i + \frac{h}{2}, y_i + \frac{k_{1y}}{2}, z_i + \frac{k_{1z}}{2}) \\
k_{3z} &= hg(t_i + \frac{h}{2}, y_i + \frac{k_{2y}}{2}, z_i + \frac{k_{2z}}{2}) \\
k_{4z} &= hg(t_i + h, y_i + k_{3y}, z_i + k_{3z})
\end{aligned}$$

Construímos então o programa abaixo para aplicar este método

```
from numpy import arange

def g(t,y,z):
    ''' A partir dos atuais valores de t, y(t) e y'(t) calcula o valor de
        y''(t) '''

    g = 12*t**2 + 4*t**3 - t**4 + y - z

    return g

def passo_RK4(t,y,z,h):
    ''' Dados os valores de t, y, z = y' e h, calcula pelo metodo de Runge-
        Kutta
        de 4a ordem, os valores de y(t+h) e z(t+h) '''

    k1y = h*z
    k1z = h*g(t,y,z)
    k2y = h*(z+0.5*k1z)
    k2z = h*g(t+0.5*h,y+0.5*k1y,z+0.5*k1z)
    k3y = h*(z+0.5*k2z)
    k3z = h*g(t+0.5*h,y+0.5*k2y,z+0.5*k2z)
    k4y = h*(z+k3z)
```

```
    k4z = h*g(t+h,y+k3y,z+k3z)

    passo_y = (k1y+2*(k2y+k3y)+k4y)/6
    passo_z = (k1z+2*(k2z+k3z)+k4z)/6

    return passo_y, passo_z

#Tamanho do passo
h = 0.01

#Condicoes iniciais
y0 = 0
z0 = 0

#Iniciamos nossas variaveis y e z pelos seus valores iniciais
y = y0
z = z0

#Implementamos o metodo de RK4 indo de t = 0 ate t = 5 a passos de h
for t in arange(0,5,h):

    #Para cada loop, calculamos o valor dos passos em y e z a serem tomados
    passo_y, passo_z = passo_RK4(t,y,z,h)

    #Atualizamos entao os valores de y e z a partir do passo calculado acima
    y = y + passo_y
    z = z + passo_z

#Printamos por fim os valores obtidos com 8 casas decimais para observar o
    ponto
#de variacao do resultado
print("y(5) = %.8f, z(5) = %.8f" %(y,z))

>>>y(5) = 624.99999998, z(5) = 499.99999997
```

E como é fácil de perceber, por este método nossos resultados foram de

$$y(5) = 624.99999998 \text{e} \dot{y}(5) = 499.99999997,$$

com uma diferença de apenas

$$\Delta y = y_{exato} - y_{RK4} = 2 \times 10^{-8} \text{e} \Delta \dot{y} = \dot{y}_{exato} - \dot{y}_{RK4} = 3 \times 10^{-8}.$$

Note aqui que o único motivo pelo qual imprimimos tantas casas decimais foi para poder observar este erro, pois se desejássemos observar com 7 ou menos casas decimais o programa já arredondaria o resultado calculado para o valor exato, logo temos uma precisão excelente pelo método de Runge-Kutta de 4a ordem, muito melhor, inclusive, do que a do método de Euler.

É interessante também observar o que ocorre quando alteramos nosso programa para operar em precisão simples. Em uma primeira situação, o programa parece dar resulta-

dos um pouco estranhos (colocados um pouco mais a frente), mas ao analisarmos melhor o que está ocorrendo, vemos que, pela forma como o programa foi escrito, a pura adaptação de passar para precisão simples faz com que o programa passe a calcular, na verdade, $y(5+h) = 630.014771$ e $\dot{y}(5+h) = 503.005890$. Se observamos o valor calculado anterior a este, isto é, os valores de $y(5)$ e $\dot{y}(5)$ propriamente ditos, vemos que eles foram calculados como sendo $y(5+h) = 624.999756$ e $\dot{y}(5+h) = 499.999878$. Sendo assim, ao compararmos com os resultados pela precisão dupla, naturalmente nós observamos uma perda de precisão, diretamente atribuída aos erros por "round-off"realizados pelo programa. Ainda assim, temos uma precisão muito melhor que a do método de Euler e, caso desejássemos uma precisão da ordem de 4 casas decimais, obteríamos os resultados exatos que desejávamos.

Parte II

Questão 1

Item a

No intuito de modelar um potencial de poço duplo, nós podemos fazer uso da equação de Duffing, dada por

$$\ddot{x} = \frac{1}{2}x(1-x^2)$$

Nós podemos estudar a dinâmica deste potencial a partir de diagramas de espaços de fases de $\dot{x}(t) \times x(t)$ com diferentes condições iniciais sendo impostas ao nosso sistema. Sendo assim, podemos adaptar o nosso programa da Parte I para resolver a equação de Duffing a partir do método de Runge-Kutta de 4a ordem. Tomaremos como posição inicial $x(0) = 1$ fixa, e iremos analisar o que diferentes velocidades iniciais causam com a dinâmica da partícula. Usaremos aqui os casos em que $\dot{x}(0) = -0.1$, -0.5 e -1. O nosso diagrama foi colocado abaixo, e vale o comentário de que optamos aqui por juntar os três diagramas de espaço de fases em um único diagrama para podermos mais facilmente compará-los.

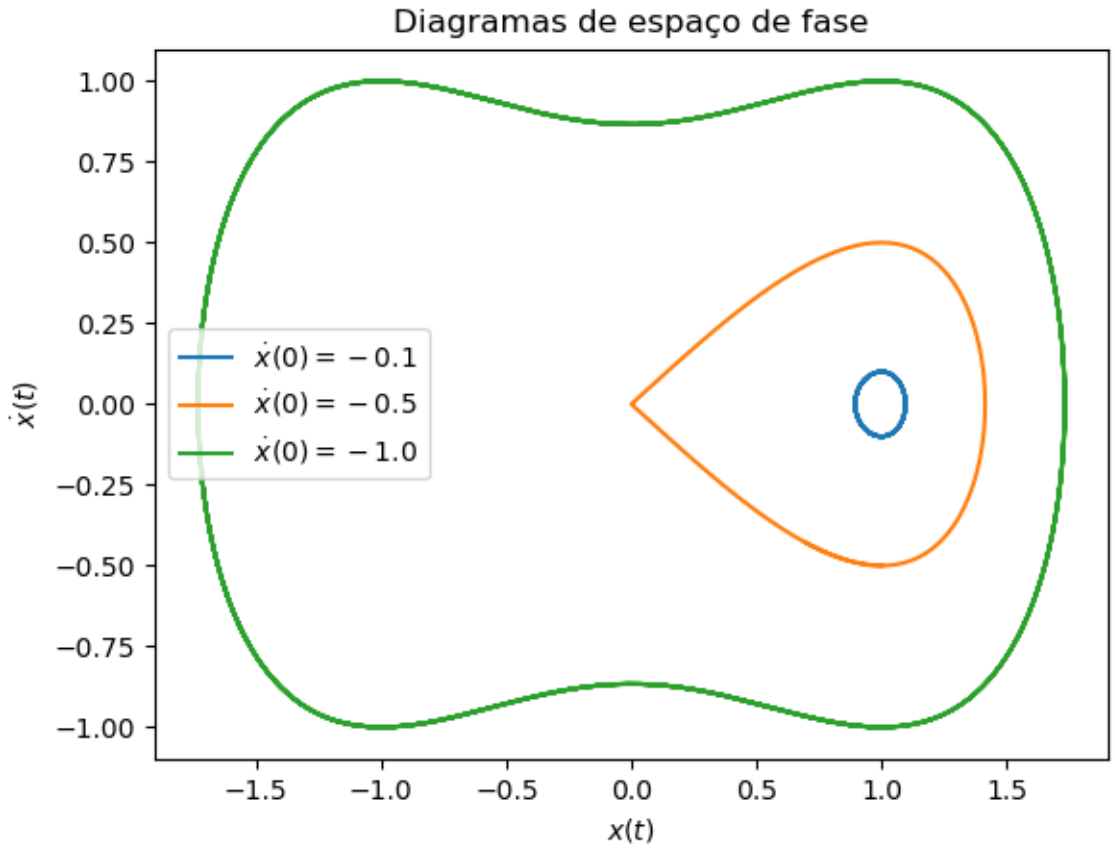

Figura 1: Diagramas de espaço de fases para diferentes condições iniciais de $\dot{x}(0)$

Podemos então analisar que, para uma pequena velocidade inicial (em módulo), de apenas -0.1, a nossa partícula orbita o ponto $(1, 0)$ do espaço de fases continuamente. Por outro lado, ao tomarmos uma velocidade inicial grande (em módulo), no caso, de $\dot{x}(0) = -1$, a nossa partícula passa a orbitar a origem, mas não como uma elipse e sim como uma forma levemente deformada (veremos após nossa discussão sobre o caso em que $\dot{x}(0) = -0.5$ que, na verdade, a partícula orbita ao mesmo tempo os pontos $(1, 0)$ e $(-1, 0)$).

Para analisar o caso em que $\dot{x}(0) = -0.5$ nós devemos ser um pouco mais cuidadosos. Primeiro nós devemos notar que este diagrama foi obtido ao se tomar o passo de Runge-Kutta como sendo de $h = 0.01$, tal qual como na Parte I. Contudo, ao refinarmos nosso passo para $h = 0.001$, nossos cálculos são um pouco mais precisos e isto nos gera um novo diagrama para a condição inicial de $\dot{x}(0) = -0.5$, como podemos observar abaixo

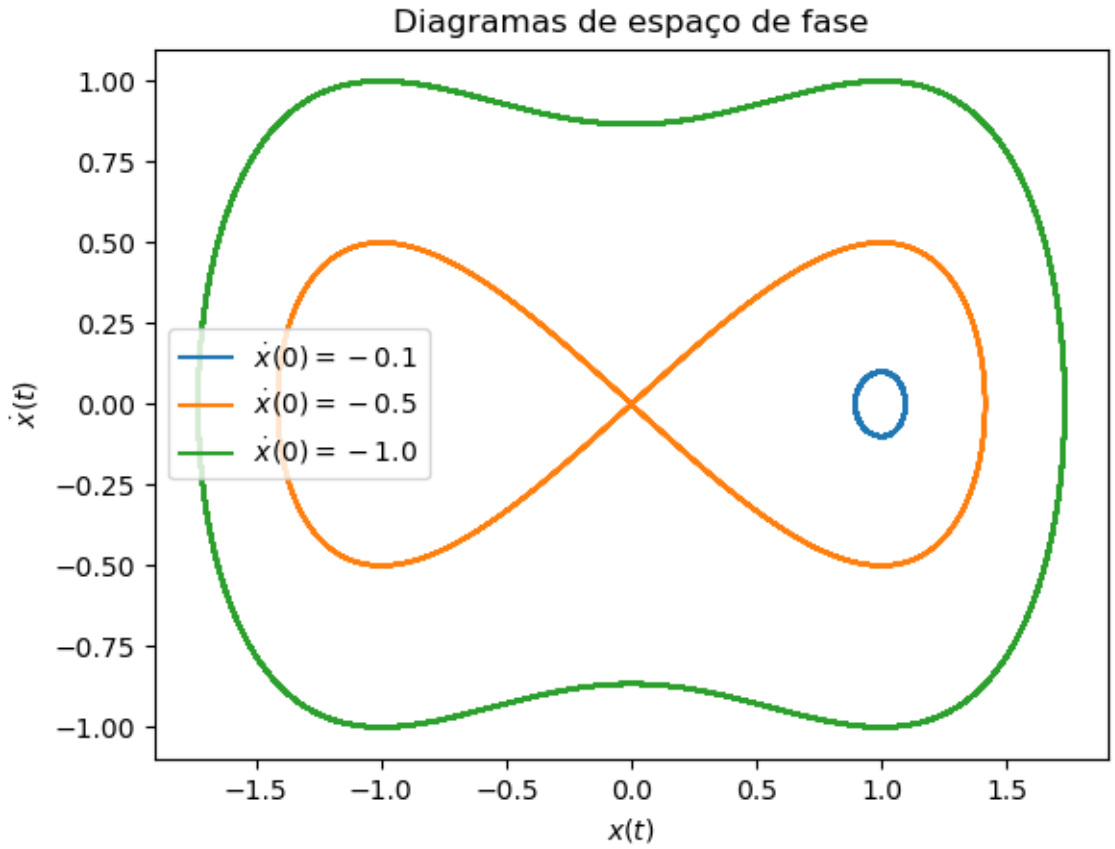

Figura 2: Diagramas de espaço de fases com passo de RK4 refinado para $h = 0.001$

Esta discrepância entre os diagramas pode ser atribuída ao fato de que, na situação em que $\dot{x}(0) = -0.5$ nós estamos em um caso limite, entre os regimes em que $\dot{x}(0) = -1$ e $\dot{x}(0) = -0.1$. Dizemos que este é um caso limite pois, se tomarmos valores um pouco menores (em módulo) do que 0.5 para $\dot{x}(0)$, nós termos uma situação similar a quando $\dot{x}(0) = -0.1$, com a partícula orbitando o ponto $(1, 0)$ do espaço de fases, contudo se $\dot{x}(0)$ for ligeiramente maior (em módulo) do que 0.5, passaremos a ter uma situação similar ao caso em que $\dot{x}(0) = -1$ e a partícula irá orbitar "a origem". Podemos visualizar isto mais claramente se olharmos as situações em que $\dot{x}(0) = -0.45$ e -0.55, como ilustramos abaixo:

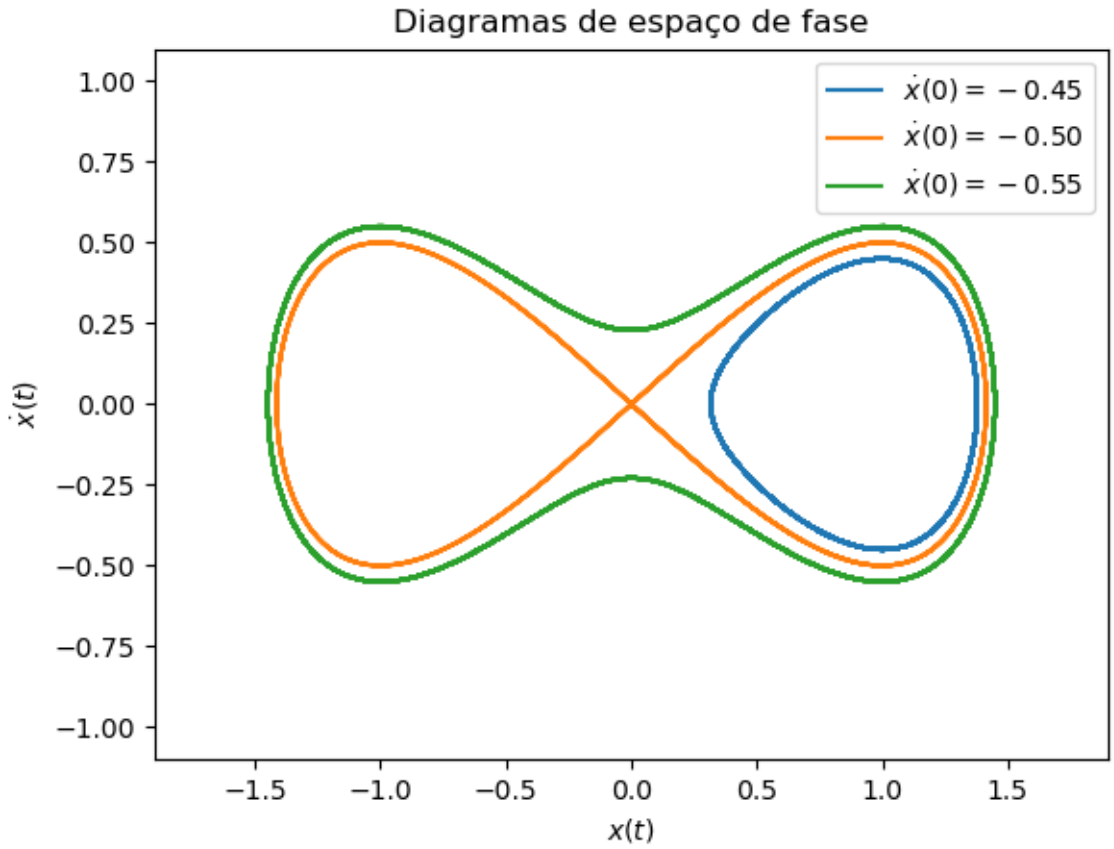

Figura 3: Diagramas de espaço de fases considerando velocidades iniciais próximas a -0.5 (escala mantida igual a das figuras 1 e 2)

Com este último diagrama podemos compreender melhor o que as diferentes condições iniciais para a velocidade causam com a dinâmica da nossa partícula sujeita a um potencial poço duplo. O que ocorre é que, para velocidades menores (em módulo) do que 0.5 (por exemplo, -0.1), a partícula irá orbitar o ponto $(1, 0)$ do espaço de fases, enquanto que para velocidades maiores (em módulo) do que 0.5 (por exemplo, -1), a partícula irá orbitar não a origem, mas ambos os pontos $(-1, 0)$ e $(1, 0)$, mas sem nunca cruzar a origem. Por fim, quando $\dot{x}(0)$ é exatamente igual a 0.5 (ou -0.5), a partícula novamente orbita os pontos $(-1, 0)$ e $(1, 0)$, mas infinitamente cruzando a origem. Por conta disso nós dizemos que $\dot{x}(0) = -0.5$ é um caso limite, pois esta condição divide dois regimes significativamente diferentes e, por se tratar se uma situação tão delicada, nós devemos utilizar passos mais refinados, pois qualquer erro devido ao tamanho do passo podem acarretar na partícula indo para outro regime, que não o caso limite.

Item b

Ao incluirmos amortecimento ao nosso potencial de poço duplo, a equação de Duffing deve ser corrigida para levar este efeito em consideração. Deste modo, escrevemos ela como

$$\ddot{x} = -2\gamma\dot{x} + \frac{1}{2}x(1 - x^2)$$

Com esta correção, podemos estudar a nova dinâmica da partícula analisando o espaço de fases para diferentes valores de γ, e mantendo $x(0)$ e $\dot{x}(0)$ fixos. Optamos aqui por tomar $x(0) = 1$ como no item anterior e fixamos $\dot{x}(0) = -1$. Vamos recordar que neste caso a partícula deveria orbitar os pontos $(1, 0)$ e $(-1, 0)$ do espaço de fases sem nunca cruzar sua orgiem (Figura 2). Para nossa análise, tomaremos dois valores para 2γ (usaremos 2γ no lugar de γ para simplificar o nosso código), sendo estes $2\gamma = 0.25$ e 0.8. Nosso novo espaço de fases pode ser visto abaixo

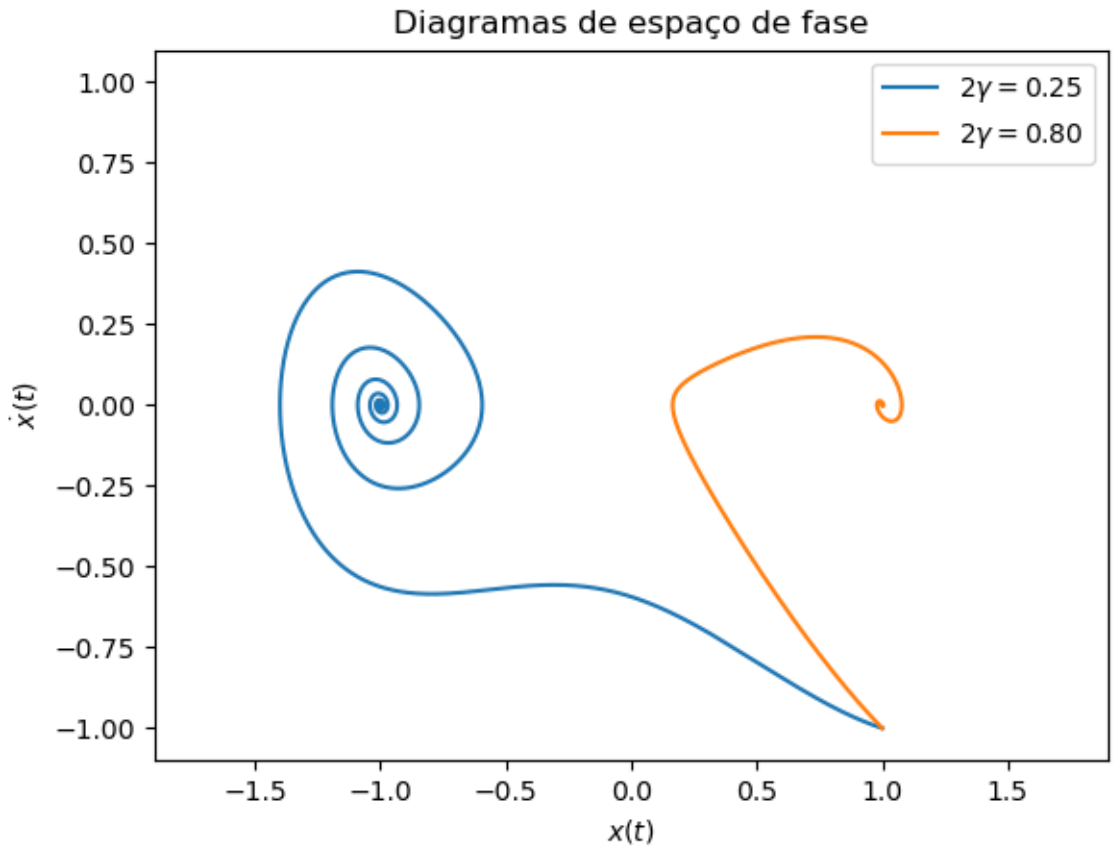

Figura 4: Diagramas de espaço de fases do potencial de poço duplo na presença de amortecimento

Perceba que mantivemos a mesma escala que no item anterior. O que podemos analisar

a partir do nosso espaço de fases é que, na presença do amortecimento, nossa partícula tendo a "cair"nos dois pontos que a princípio ela estaria orbitando. No caso de um amortecimento baixo nós vemos que ela vai parar no ponto mais distante das condições iniciais, enquanto que para um alto amortecimento ela tende ao ponto mais próximo das condições iniciais e perceba ainda que neste caso ela se aproxima significativamente da origem (!!) que é algo que nunca ocorria no caso sem amortecimento quando $|\dot{x}(0)| > 0,5$

Item c

A última correção que faremos ao poço potencial duplo será a de incluir uma força do tipo $F = \cos(\omega t)$ agindo sobre o sistema, de modo que a equação de Duffing será dada por

$$\ddot{x} = -2\gamma\dot{x} + \frac{1}{2}x(1 - x^2) + F\cos(\omega t)$$

Para estudar esta última modificação iremos nos focar apenas no efeito de variar a *intensidade* da força F, mantendo sua frequência fixa como $\omega = 1$. Usaremos as mesmas condições iniciais do item b ($x(0) = 1$, $\dot{x}(0) = -1$) e fixaremos nosso amortecimento como sendo de $2\gamma = 0.25$. Tomaremos como valores de intensidade força $F = 0.22$, 0.23, 0.28, 0.35 e por fim um caso mais exagerado em que $F = 0.6$.

Antes de mostrar nossos resultados, vamos lembrar que, em um sistema de oscilações amortecidas e forçadas nós teremos um regime inicial em que o sistema está mais "maluco"devido a presença dos efeitos da oscilação não forçada somados com os efeitos da força, chamado de transiente. Após a passagem do transiente, passamos a ter o efeito exclusivo da força (e do amortecimento) sobre o nosso sistema. Por conta do comportamento mais caótico do sistema durante o transiente, nós vamos removê-los de nossos diagramas, deixando apenas os resultados após a sua passagem.

Por fim, lembremos que pela Figura 4, na situação em que $F = 0$, devemos ter a partícula orbitando o ponto $(1, 0)$, fatalmente "caindo"neste ponto. Vejamos o que ocorre agora que $F \neq 0$.

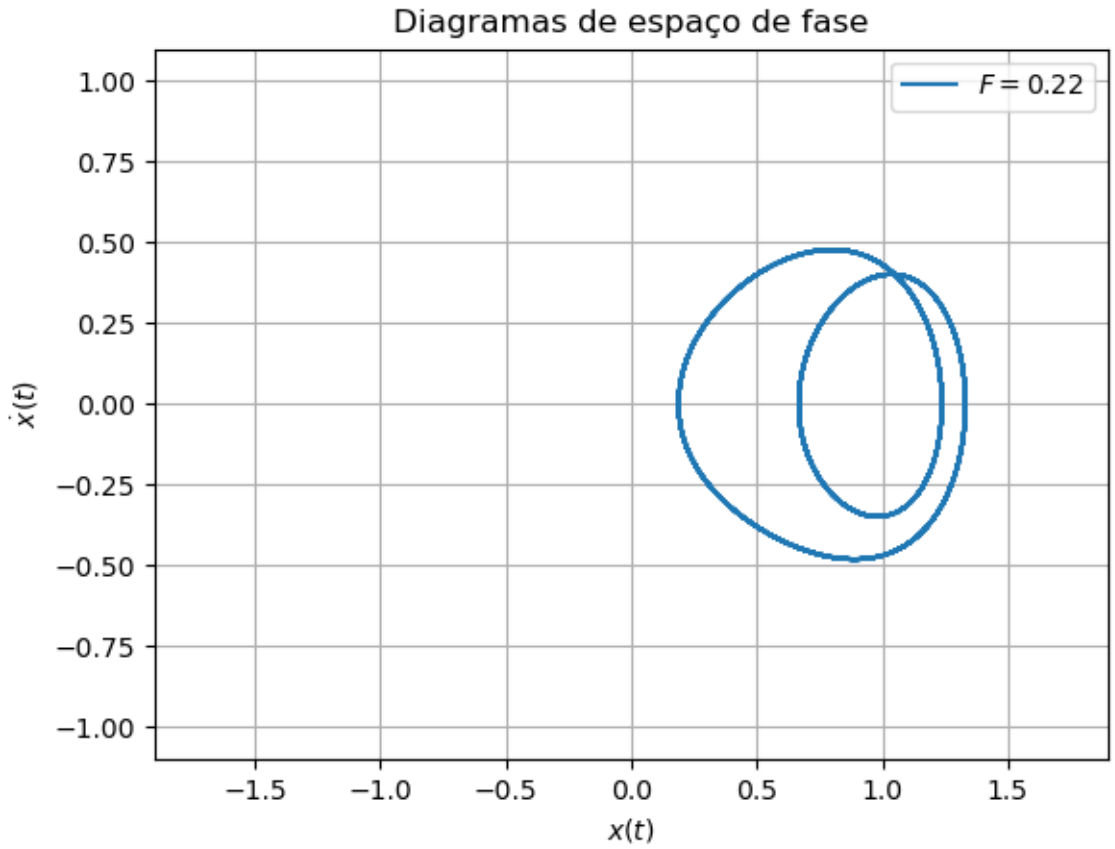

Figura 5: Diagramas de espaço de fases do potencial de poço duplo na presença de uma força de intensidade $F = 0.22$

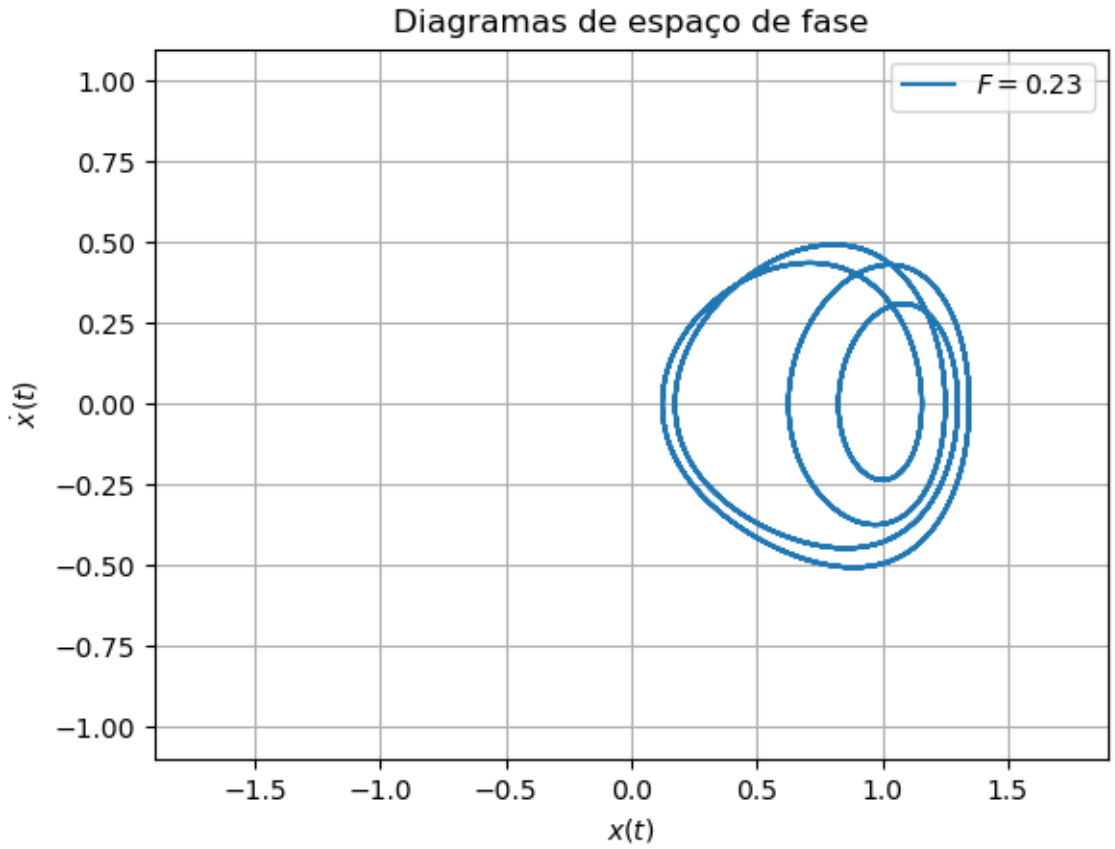

Figura 6: Diagramas de espaço de fases do potencial de poço duplo na presença de uma força de intensidade $F = 0.23$

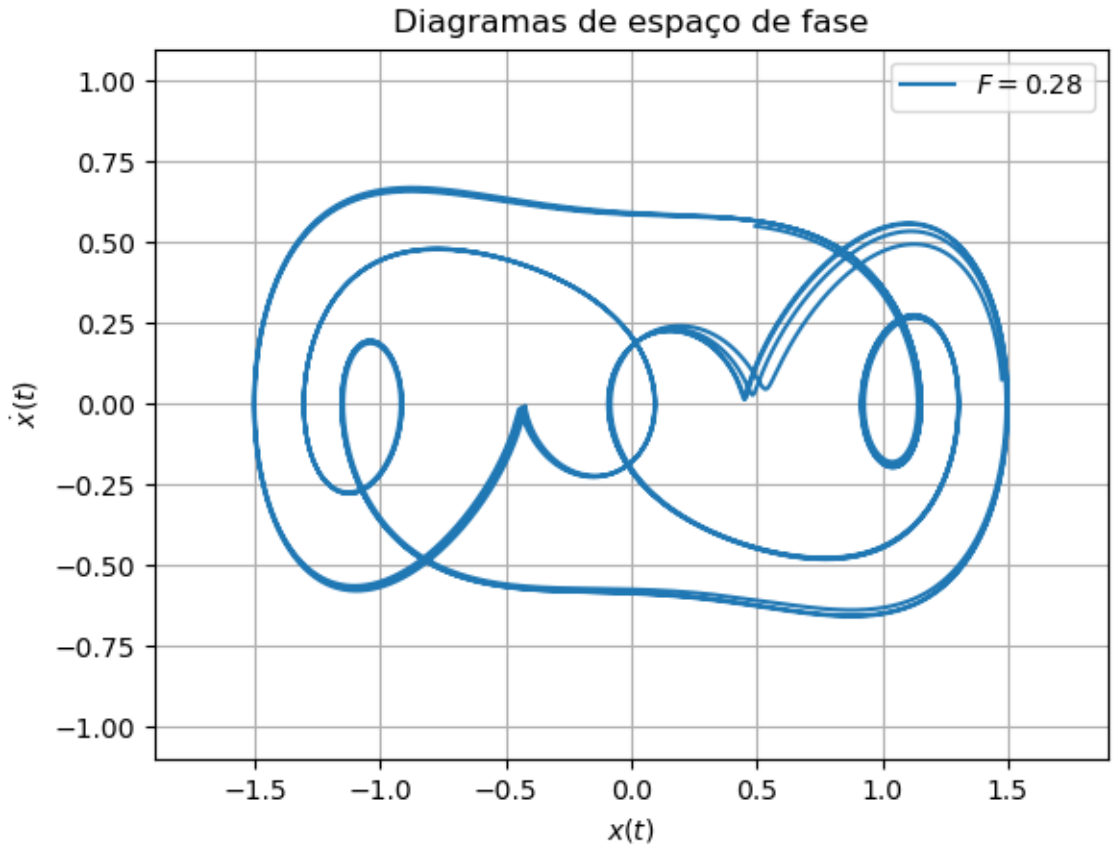

Figura 7: Diagramas de espaço de fases do potencial de poço duplo na presença de uma força de intensidade $F = 0.28$

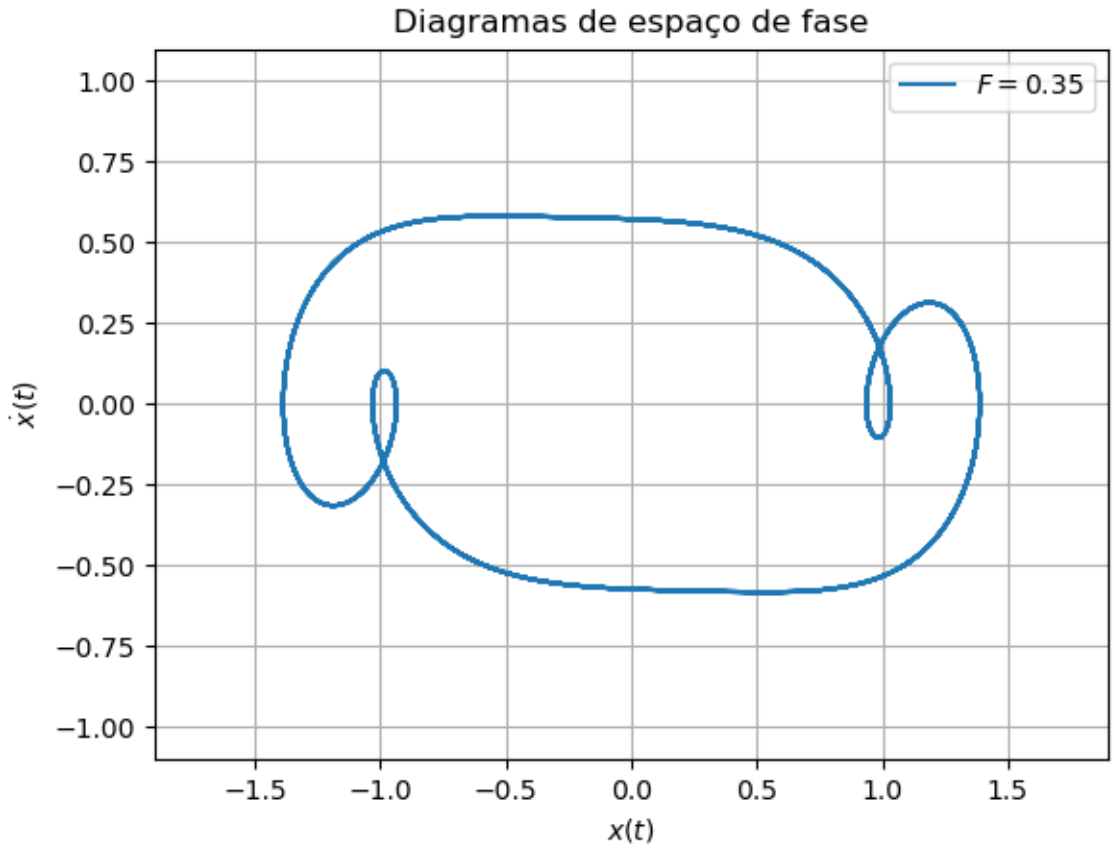

Figura 8: Diagramas de espaço de fases do potencial de poço duplo na presença de uma força de intensidade $F = 0.35$

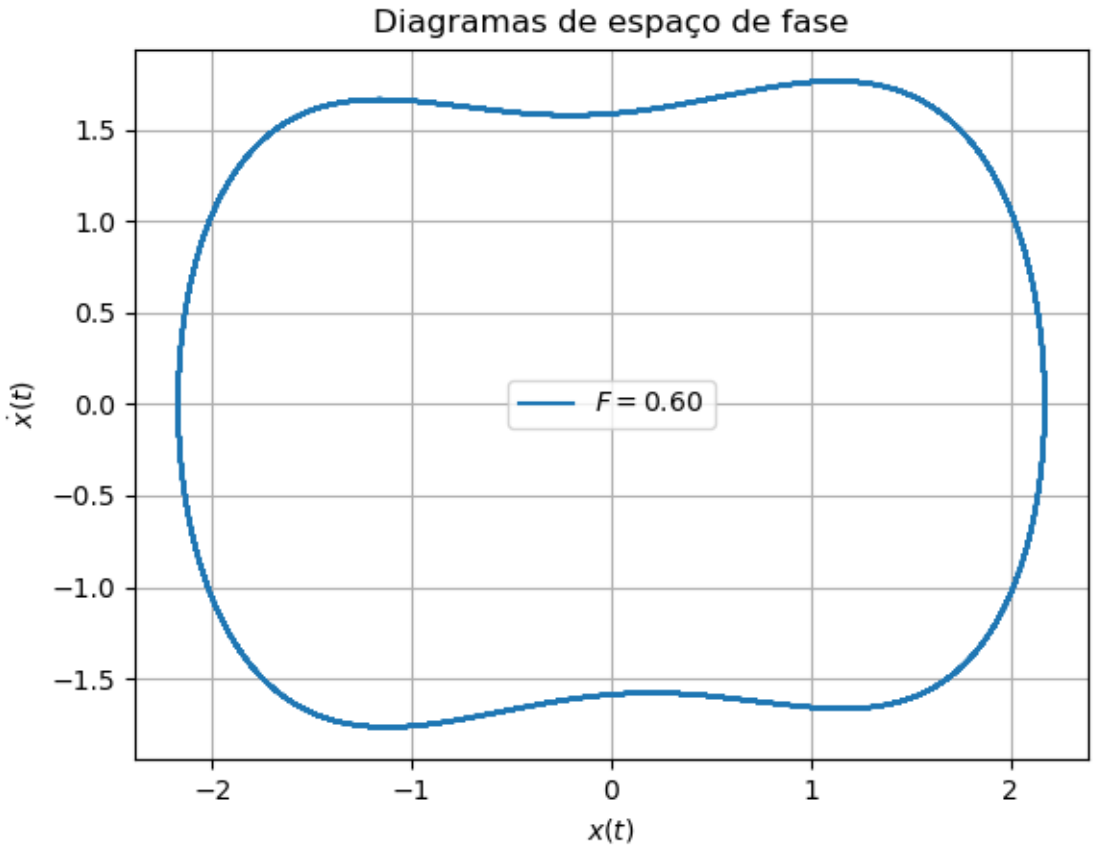

Figura 9: Diagramas de espaço de fases do potencial de poço duplo na presença de uma força de intensidade $F = 0.6$

Novamente nossos diagramas mantiveram a mesma escala dos itens anteriores, com exceção do último caso, onde a dinâmica extrapola os limites do diagrama, e agora separamos os casos em diferentes diagramas para não poluir tanto a visão e ficar mais claro os efeitos de cada caso. Podemos observar que a presença da força devolve a periodicidade do sistema, que havia sido perdida por conta do amortecimento. Vemos também que para alguns valores de F nós temos órbitas fechadas ao redor do ponto $(1, 0)$ ($F = 0.22$ e 0.23) e para valores mais elevados de F temos órbitas fechadas ao redor dos pontos $(1, 0)$ e $(-1, 0)$ simultaneamente ($F = 0.35$ e 0.6). Note que, apesar de serem órbitas fechadas nós temos agora uma espécie de precessão do sistema, em que a partícula completa uma ou mais voltas em torno dos pontos em que ela orbita antes de completar a órbita total, o que era observado nos casos em que $F = 0$.

Uma rápida investigação dos diagramas acima nos permite notar que dois deles se destoam em relação aos outros. O primeiro deles é aquele em que $F = 0.6$, pois aqui podemos ver que a grande intensidade da força simplesmente anula os efeitos do amortecimento do sistema e nós voltamos a observar uma dinâmica muito similar a do caso em que $F = 0$, mas dessa vez

com uma órbita muito maior em relação a anterior, como pode ser observado pela variação da escala do nosso diagrama entre as Figuras 9 e 8, por exemplo (na verdade, pode-se comparar as escalas da Figura 9 com a de qualquer outra figura presente neste EP).

O segundo caso mais destoante, e na verdade, o mais interessante, é aquele em que $F = 0.28$ pois, a princípio, parece que nós não conseguímos tirar todo o transiente da figura e nós deveríamos ter deixado o sistema evoluir por mais tempo antes de começar a colecionar os pontos a serem plotados em nosso diagrama, mas quando nós fazemos isso uma coisa ainda mais curiosa ocorre

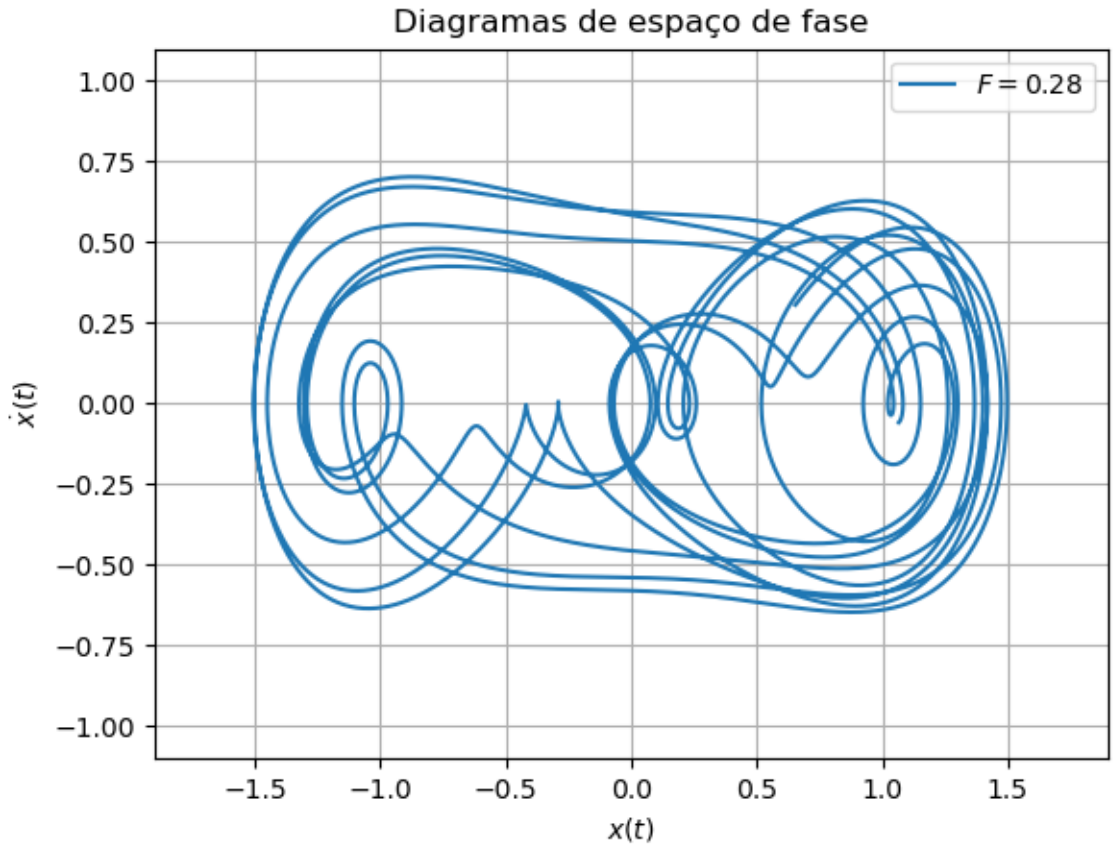

Figura 10: Diagrama de espaço de fases do potencial de poço duplo na presença de uma força de intensidade $F = 0.28$ considerando um maior transiente

O sistema parece ainda mais caótico, como se nós tivéssemos considerando mais pontos antes do fim do transiente sendo que foi exatamente o contrário do que fizemos aqui. O que ocorre aqui, na realidade, é que para este valor de força nossa órbita passa a ser aberta, de modo que ela não retorna mais ao ponto inicial de onde partimos (ela pode chegar extremamente próxima a este ponto em algum momento, mas não retornará a ele). Tendo já caracterizado agora todos os efeitos que os diferentes parâmetros geram em nosso sistema,

podemos seguir nossa discussão adiante.

Para construir as imagens acima (e inclusive as dos itens (a) e (b)) e realizar os cálculos para evoluir o sistema sujeito a equação de Duffing, foi utilizado um programa em Pyhton que resolve a equação de Duffing por meio do método de Runge-Kutta de 4a ordem, já discutido na parte I deste EP. Para construir este programa, nós adaptamos aquele utilizado na parte I e, nesta adaptação, ao invés de calcularmos a evolução do sistema para um certo tempo final t_f dado, nós escolhemos uma certa quantidade de passos n de tamanho h a serem tomados para evoluir o sistema. Sendo assim, se desejamos evoluir o sistema de, por exemplo, $t_i = 0$ até $t_f = 5$ em passos de $h = 0.01$, nós devemos tomar $n = (t_f - t_i)/0.01 = 500$ passos de Runge-Kutta para obter este resultado. Este programa está colocado a seguir

```
from numpy import arange
from math import cos
import matplotlib.pyplot as plt

def f(t,x,v,F):
    ''' A partir dos atuais valores de t, x(t), v(t) e F calcula o valor
        de x''(t) '''

    f = 0.5*x*(1 - x**2) - 0.25*v + F*cos(omega*t)

    return f

def RK4(t,x,v,F,h,n):
    ''' Dado um valor de passos n, calcula a evolucao temporal de um sistema
        com valores iniciais de x, v, t, F dados, em n passos de Runge-Kutta,
        com uma precisao h, tambem dada'''

    #Tomamos um loop de n passos
    for i in range(n):

        #Fazemos entao os calculos referentes a um passo de RK4
        k1x = h*v
        k1v = h*f(t,x,v,F)
        k2x = h*(v+0.5*k1v)
        k2v = h*f(t+0.5*h,x+0.5*k1x,v+0.5*k1v,F)
        k3x = h*(v+0.5*k2v)
        k3v = h*f(t+0.5*h,x+0.5*k2x,v+0.5*k2v,F)
        k4x = h*(v+k3v)
        k4v = h*f(t+h,x+k3x,v+k3v,F)

        passo_x = (k1x+2*(k2x+k3x)+k4x)/6
        passo_v = (k1v+2*(k2v+k3v)+k4v)/6

        #Em seguida, sao atualizados os valores de x, v e t a partir dos
        #calculos acima
        x = x + passo_x
        v = v + passo_v
        t = t + h
```

```
    #Retornamos os valores finais de t, x e v apos os n passos de RK4
    return t, x, v

#Tamanho do passo
h = 0.01

#Condicao inicial de x
x0 = 1
v0 = -1
omega = 1

#Lista com os valores de forca a serem usados
fs = [0.22,0.23,0.28,0.35,0.6]

for F in fs:

    #Listas para guardar os valores de x e v para construir os diagramas de
        fases
    X = []
    V = []

    #Iniciamos nossas variaveis x e v pelos seus valores iniciais e a
        adicionamos
    #as nossas listas
    x = x0
    v = v0
    t = 0

    #Evoluimos nosso sistema em 200000 passos de RK4 para passarmos o
        transiente devido
    #ao amortecimento (equivalente a evoluir o sistema ate um t = 2000)
    t, x, v = RK4(t,x,v,F,h,200000)

    #Tendo passado o transiente, passamos a guardar as atualizacoes de x e v
        para podermos
    #montar nossos espacos de fases. Para isso usamos 15000 passos de RK4 (
        equivalente a
    #evoluir mais t = 150 unidades de tempo
    for i in range(15000):

        t, x, v = RK4(t,x,v,F,h,1)

        X.append(x)
        V.append(v)

    #Ao fim do loop, plotamos o nosso diagrama de fases
    plt.plot(X,V,label="$F = %.2f$" %F)
    plt.title("Diagramas de espaco de fase")
    plt.xlabel("$x(t)$")
    plt.ylabel("$\dot x(t)$")
    if F != 0.6:
```

```
        plt.ylim(-1.1,1.1)
        plt.xlim(-1.9,1.9)
    plt.grid()
    plt.legend()
    plt.show()
```

Item d

Em aula, definimos um atrator como sendo:

"Atrator: Um conjunto de pontos ou subespaço no espaço de fase que permanece quando o transitório (ou transiente) desaparece.

Ex: pontos de equilíbrio ou pontos fixos, ciclos limite, superfície toroidal $\rightarrow$ atrator dinâmicos clássicos"

De outro modo, também foi definido como o conjunto de pontos que se permanecia constante ao longo de diferentes secções de Poincaré.

Analisando então a Figura 2 (e inclusive a Figura 3), que nos mostra o espaço de fases do item (a), nós temos que o sistema é fixo naquelas trajetórias para cada diferente condição inicial, isto é, *não há transientes* para este sistema. Se fôssemos construir secções de Poincaré para aquele sistema nós poderíamos partir de qualquer ponto, que após um certo período de tempo, o sistema iria retornar para aquele ponto inicial, de modo que qualquer ponto do espaço de fases, ou melhor, **todo o espaço** de fases pode ser caracterizado como um atrator deste sistema.

Quando nós adicionamos o amortecimento no item (b), acabamos por obter a figura 4 que já nos mostra uma coisa bem diferente do item (a). Neste caso, se esperarmos tempo suficiente o sistema irá tender a um ponto e neste ponto irá permanecer para sempre, ou seja, após um certo transiente nós sempre iremos encontrar a partícula no ponto $(1, 0)$, para um amortecimento de $2\gamma = 0.80$, ou no ponto $(-1, 0)$ para um amortecimento de $2\gamma = 0.25$. Se fossemos falar em termo de secções de Poincaré, nós poderíamos começar a mapear nossos pontos ao longo do tempo, mas sempre chegaria um momento em que um destes pontos, a depender da intensidade do amortecimento, ficaria fixo. Sendo assim, nesta situação nós temos **um ponto** que é atrator, e que depende da intensidade do amortecimento, sendo ele $(1, 0)$, se $2\gamma = 0.80$ e $(-1, 0)$ se $2\gamma = 0.25$.

Os atratores do item (c) por fim, estão determinados pelas próprias curvas que nós observamos, afinal de contas o que foi feito para gerar as figuras 5, 6, 7, 8 e 9 foi, justamente, esperar um transiente passar para então observar a trajetória da partícula no espaço de fases. O que ocorre é que nós temos todo um **conjunto de pontos** que são o atrator neste item, sendo cada um destes conjuntos que dependem da intensidade da nossa força F. Se mantivermos F fixo em algum daqueles valores, a partícula sempre acabará naquelas trajetórias, e naquelas trajetórias permanecerá, para qualquer período de tempo futuro. Vale notar aqui que para $F = 0.28$ nós ainda temos um conjunto de pontos que é atrator, pois apesar de a trajetória não ficar fixa nesta trajetória, como podemos observar pela Figura 10, nós ainda podemos observar que a partícula está se movendo nas vizinhanças de um conjunto de pontos que são aproximadamente dados pela curva da Figura 7, independente

do tempo, ou melhor, nós sempre poderemos observar este padrão no espaço de fases, "que permanece quando o transiente desaparece", no caso em que $F = 0.28$. Apenas por questão de diferenciar os atratores dos casos em que $F = 0.28$ dos outros, nós dizemos que este tipo de atrator, que é continuamente circundado, mas sem ter a trajetória exatamente igual a ele, de um **atrator estranho**, que é determinado por padrões fractais na secção de Poincaré.

Questão 2

Inspirado na discussão do Item (d) da Questão 1, foi construído um novo programa em cima do anterior para, desta vez, construir as secções de Poincaré. Para isso nós evoluímos o nosso sistema para passar pelo transiente e, após isso, evoluímos ele 100 vezes ao longo de um período $T = \frac{2\pi}{\omega}$. Nós então determinamos o valor de x nas seções, essencialmente analisando o valor de x após cada período. Após termos determinado os valores de x nós plotamos eles em função de F e com isso obtivemos o diagrama de bifurcações do nosso problema, colocado abaixo. O programa desenvolvido para realizar este processo está logo em seguida do diagrama

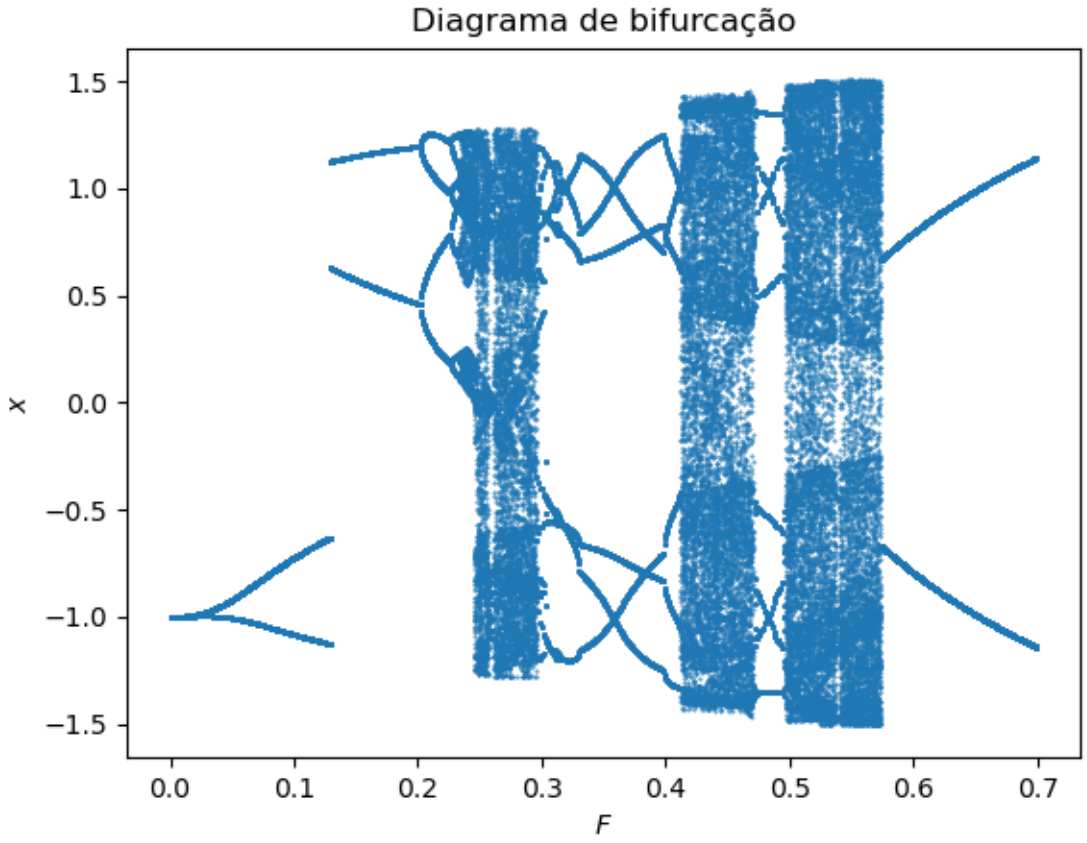

Figura 11: Diagrama de bifurcações de $F \times x$

```
from numpy import arange
from math import cos, pi
import matplotlib.pyplot as plt
from numba import njit

@njit
def f(t,x,v,F):
    ''' A partir dos atuais valores de t, x(t), v(t) e F calcula o valor
        de x''(t) '''

    f = 0.5*x*(1 - x**2) - 0.25*v + F*cos(omega*t)

    return f

@njit
def RK4(t,x,v,F,h,n):
    ''' Dado um valor de passos n, calcula a evolucao temporal de um sistema
        com valores iniciais de x, v, t, F dados, em n passos de Runge-Kutta,
        com uma precisao h, tambem dada'''

    #Tomamos um loop de n passos
    for i in range(n):

        #Fazemos entao os calculos referentes a um passo de RK4
        k1x = h*v
        k1v = h*f(t,x,v,F)
        k2x = h*(v+0.5*k1v)
        k2v = h*f(t+0.5*h,x+0.5*k1x,v+0.5*k1v,F)
        k3x = h*(v+0.5*k2v)
        k3v = h*f(t+0.5*h,x+0.5*k2x,v+0.5*k2v,F)
        k4x = h*(v+k3v)
        k4v = h*f(t+h,x+k3x,v+k3v,F)

        passo_x = (k1x+2*(k2x+k3x)+k4x)/6
        passo_v = (k1v+2*(k2v+k3v)+k4v)/6

        #Em seguida, sao atualizados os valores de x, v e t a partir dos
        #calculos acima
        x = x + passo_x
        v = v + passo_v
        t = t + h

    #Retornamos os valores finais de t, x e v apos os n passos de RK4
    return t, x, v

#Valor da frequencia da forca
omega = 1

#Intervalo de intensidade de forca
DeltaF = 0.0005
```

```
#Condicoes iniciais
x0 = 1
v0 = -1

#Definimos os numeros de passos a serem tomados ate o fim do transiente e
#necessarios para completar um periodo
trans = 200000
periodo = 1000

#Listas para guardar os valores de x e F para construir o diagrama de
    bifurcacoes
X = []
F_list = []

#Realizamos entao um loop para todos os valores desejados para forca em
#intervalos igualmente espacados de DeltaF
for F in arange(0,0.7,DeltaF):

    #Para verificar o andamento do programa usamos um print para verificar
    #em qual valor de F estamos atualmente calculando (opcional)
    #print("F = %f" %F)

    #Iniciamos nossas variaveis x, v e t pelos seus valores iniciais
    t = 0
    x = x0
    v = v0

    #Definimos nossa precisao inicial
    h = 0.01*2*pi/omega

    #Evoluimos o sistema ate o fim do transiente
    t, x, v = RK4(t,x,v,F,h,trans)

    #Passado o transiente, reduzimos o tamanho do nosso passo h
    h = 0.001*pi/omega

    #Fazemos entao um loop para evoluir nosso sistema ao longo de 100 periodos
    for i in range(100):

        #Evoluimos nosso sistema em um periodo, dado por 1000 passos de RK4
        t, x, v = RK4(t,x,v,F,h,periodo)

        #Impressao dos valores de F e x atuais (opcional)
        #print("F = %f, x_i = %f" %(F,x))

        #Com os valores finais apos um periodo, guardamos estes valores em
            listas
        #para construir nosso diagrama
        X.append(x)
        F_list.append(F)
```

```
#Ao fim de todos os caluclos, plotamos o nosso diagrama de bifurcacao
plt.scatter(F_list,X,marker='.',s=.7)
plt.title("Diagrama de bifurcacao")
plt.xlabel("$F$")
plt.ylabel("$x$")
plt.show()
```

Observamos aqui que em nosso programa nós começamos tomando um passo de tamanho $h = 0.01\frac{2\pi}{\omega}$ para evoluir o transiente mais rapidamente, e após isso nós o diminuímos para $h = 0.001\frac{2\pi}{\omega}$ para podermos obter cálculos do diagrama mais precisos. Testes foram realizados mantendo $h = 0.001\frac{2\pi}{\omega}$ ao longo de toda a dinâmica e o gráfico produzido acabou por ser o mesmo (isto é, refinar o transiente não afetou nosso resultado final). Além disso, o programa tem a opção, na linha 94 do código, de imprimir os valores de x e F encontrados para a secção de Poincaré, contudo optamos por manter esta linha opcional, dado que muitas linhas eram impressas, por vezes deixando o código muito mais lento do que o desejado (mas está lá, e podemos observar os pontos x da secção para cada F sem problemas).

A partir do nosso diagrama de bifurcação (Figura 11 nós somos capazes de estimar a constante de Feigenbaum δ. Para fazer isso, nós devemos começar ampliando nosso diagrama para observar as primeiras bifurcações nele, como podemos observar na figura abaixo.

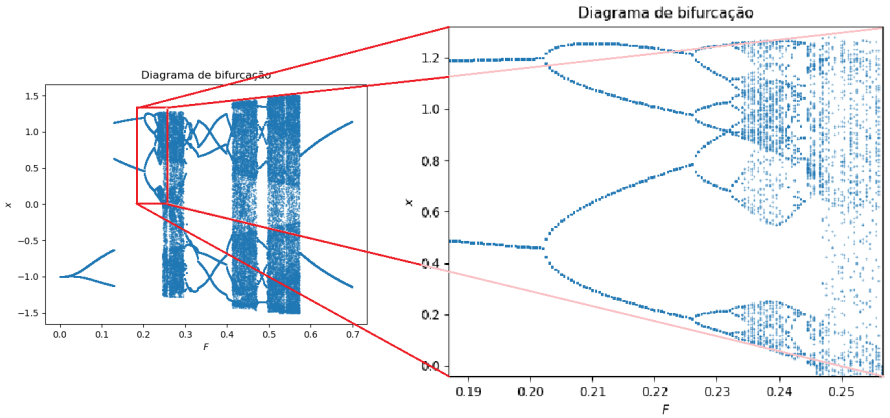

Figura 12: Ampliação do diagrama de bifurcações

Nosso "zoom"das bifurcações é então dado pela figura

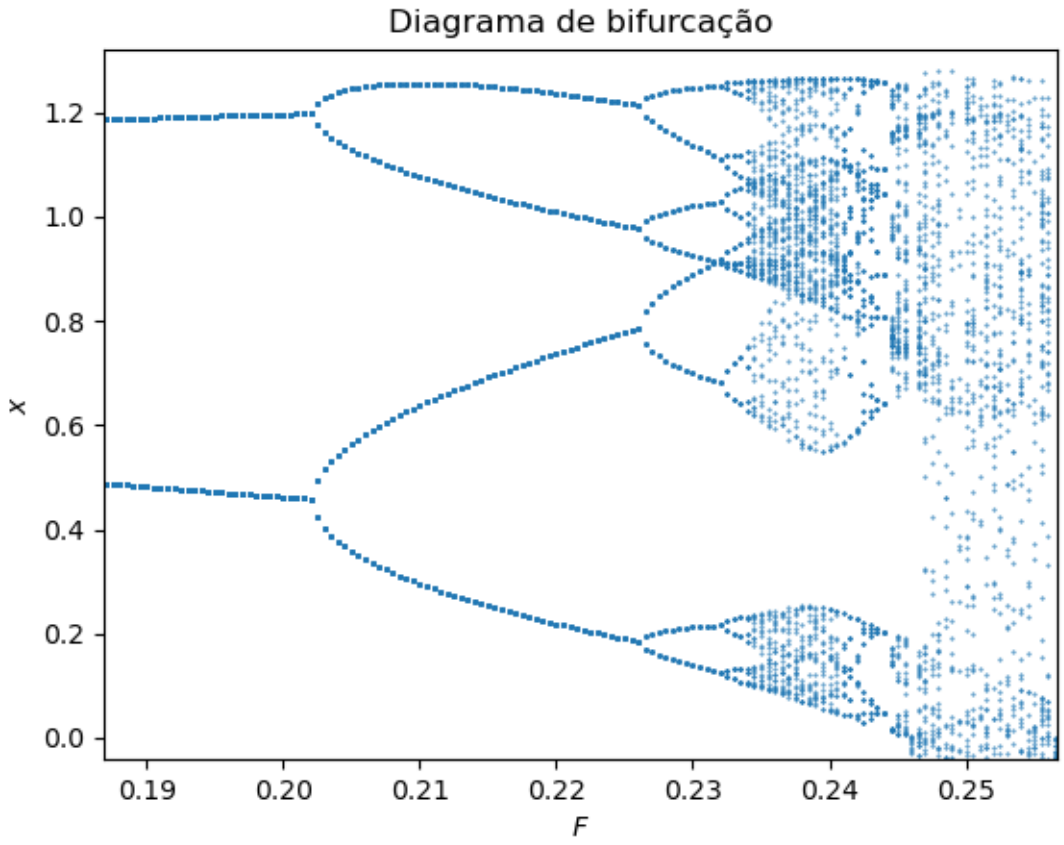

Figura 13: Diagrama de bifurcações ampliado

Se denotarmos a diferença no valor de F entre cada bifurcação por L_i, com $i = 0, 1, 2 \ldots$, então a constante de Feigenbaum pode ser calculada de forma aproximada como sendo

$$\delta = \frac{L_i}{L_i + i}$$

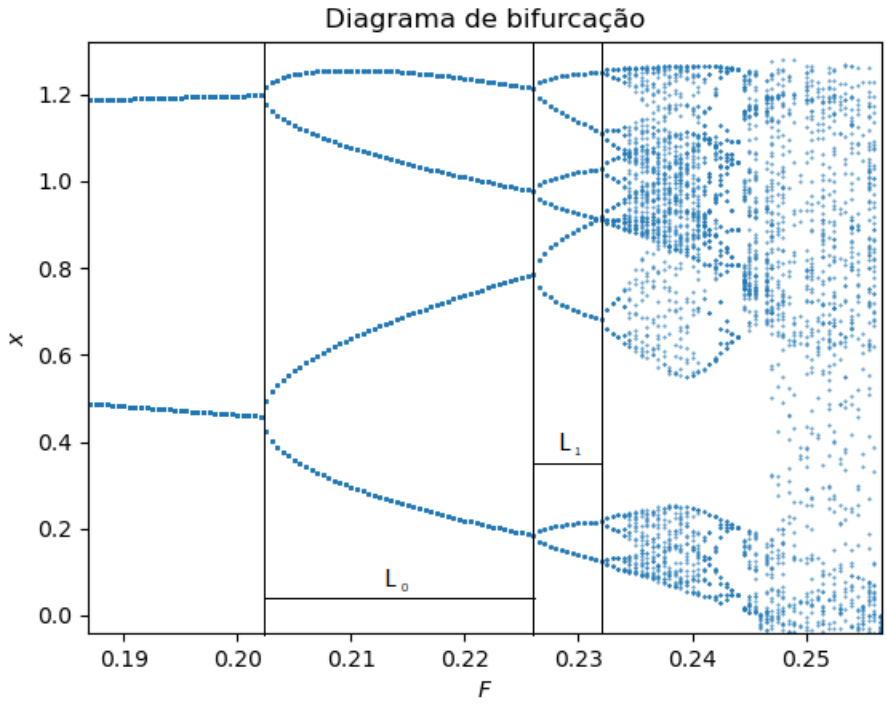

Figura 14: Representação dos L_i para determinar a constante de Feigenbaum

Se então olharmos os valores de L_i para $i = 0, 1$ na Figura 14 iremos obter que

$$L_0 = 0.226 - 0.202 = 0.024$$

$$L_1 = 0.232 - 0.226 = 0.006$$

E assim a constante de Feigenbaum será, aproximadamente

$$\delta = \frac{L_0}{L_1} = 4$$

Que, ao compararmos com a literatura ($\delta = 4.669201\ldots$)[1], vemos ser um resultado grosseiro, embora aproximado da constante de Feigenbaum.

Questão 3

A partir do programa da Questão 2 nós agora fixamos $F = 0.28$ e, ao invés de evoluirmos nosso sistema ao longo de 100 períodos (após o transiente) e então apresentar os valores de F e x calculados para cada período, nós iremos evoluir o sistema ao longo de 20000 períodos (após o transiente), e assim iremos imprimir os valores calculados de x e v após cada período (novamente, tal como na Questão 2, por se tratar de uma quantidade muito

grande de pontos, nós deixamos a impressão destes valores no programa como opcional, bastando retirar o comentário para poder observá-los). Com estes valores de x e v nós podemos plotá-los em um gráfico $\dot{x}(t) \times x(t)$ como pontos (x_i, v_i). Este processo irá nos dar o mapa de Poincaré do nosso sistema ao longo destes 20000 períodos. Note que, se na Questão 2 alterar o tamanho do passo h não alterava as condições do diagrama, para o mapa ele alterava, sendo assim, tomamos $h = 0.001\frac{2\pi}{\omega}$ ao longo de toda a dinâmica. Tanto o Mapa de Poincaré, quanto o código utilizado para gerá-lo foram colocados abaixo

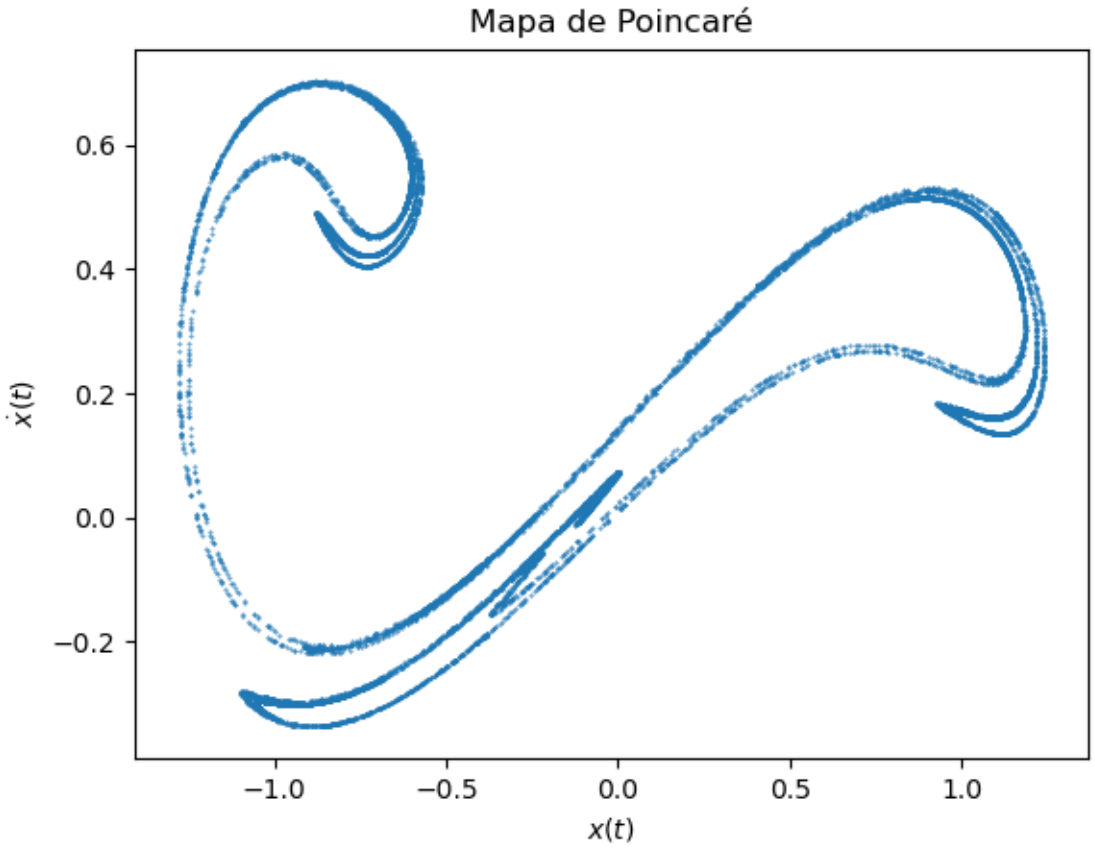

Figura 15: Mapa de Poincaré ao longo de 20000 períodos

```
from math import cos, pi
import matplotlib.pyplot as plt
from numba import njit

@njit
def f(t,x,v,F):
    ''' A partir dos atuais valores de t, x(t), v(t) e F calcula o valor
        de x''(t) '''
```

```
    f = 0.5*x*(1 - x**2) - 0.25*v + F*cos(omega*t)

    return f

@njit
def RK4(t,x,v,F,h,n):
    ''' Dado um valor de passos n, calcula a evolucao temporal de um sistema
        com valores iniciais de x, v, t, F dados, em n passos de Runge-Kutta,
        com uma precisao h, tambem dada'''

    #Tomamos um loop de n passos
    for i in range(n):

        #Fazemos entao os calculos referentes a um passo de RK4
        k1x = h*v
        k1v = h*f(t,x,v,F)
        k2x = h*(v+0.5*k1v)
        k2v = h*f(t+0.5*h,x+0.5*k1x,v+0.5*k1v,F)
        k3x = h*(v+0.5*k2v)
        k3v = h*f(t+0.5*h,x+0.5*k2x,v+0.5*k2v,F)
        k4x = h*(v+k3v)
        k4v = h*f(t+h,x+k3x,v+k3v,F)

        passo_x = (k1x+2*(k2x+k3x)+k4x)/6
        passo_v = (k1v+2*(k2v+k3v)+k4v)/6

        #Em seguida, sao atualizados os valores de x, v e t a partir dos
        #calculos acima
        x = x + passo_x
        v = v + passo_v
        t = t + h

    #Retornamos os valores finais de t, x e v apos os n passos de RK4
    return t, x, v

#Valor da frequencia da forca
omega = 1

#Condicoes iniciais
x0 = 1
v0 = -1
F = 0.28

#Definimos os numeros de passos a serem tomados ate o fim do transiente e
#necessarios para completar um periodo
trans = 200000
periodo = 1000

#Listas para guardar os valores de x e v para construir o mapa de Poincare
X = []
V = []
```

```
#Iniciamos nossas variaveis x, v e t pelos seus valores iniciais
t = 0
x = x0
v = v0

#Definimos nossa precisao
h = 0.001*2*pi/omega

#Evoluimos o sistema ate o fim do transiente
t, x, v = RK4(t,x,v,F,h,trans)

#Fazemos entao um loop para evoluir nosso sistema ao longo de 20000 periodos
for i in range(20000):

    #Para acompanhar o andamento do programa, printamos quantos por cento
    #dos loops ja foram realizados (opcional)
    #if i % 2000 == 0:

    #    prctg = i//200
    #    print(prctg,"%")

    #Evoluimos nosso sistema em um periodo, dado por 1000 passos de RK4
    t, x, v = RK4(t,x,v,F,h,periodo)

    #Impressao dos valores de x e v atuais (opcional)
    #print("x_i = %f, v_i = %f" %(x,v))

    #Com os valores finais apos um periodo, guardamos estes valores em listas
    #para construir nosso grafico
    X.append(x)
    V.append(v)

#Ao fim de todos os caluclos, plotamos o nosso mapa de Poincare
plt.scatter(X,V,marker='.',s=.7)
plt.title("Mapa de Poincare")
plt.xlabel("$x(t)$")
plt.ylabel("$\dot x(t)$")
plt.show()
```

Referências

[1] Bifurcation diagrams and the Feigenbaum constant
`https://sites.google.com/site/logicedges/feigenbaum-diagram`

Acessado em 03/12 às 4:00