projeto

March 17, 2017

0.1 Relatório

0.1.1 Daniel Ruhman e Marcelo Terreiro Prado

O projeto a seguir foi realizado para a disciplina de Matemática Multivariada ministrada pelo Professor Fabio Orfali no curso de Engenharia do INSPER. Ele tem como objetivo o desenvolvimento de uma simulação computacional baseada em métodos numéricos (mais especificamente a aproximação da curva por uma linha poligonal composta por n segmentos) para calcular o tempo aproximado que um objeto, sujeito apenas à força da gravidade, leva para percorrer uma trajetória dada por curvas parametrizadas e pontos iniciais e finais ou domínio.

```
In [1]: import matplotlib as mpl
        import numpy as np
        import matplotlib.pyplot as plt
```

0.2 Dedução geométrica da parametrização da cicloide

Considere uma circunferência C de raio r com um ponto P, fixo. Ao rolar a circunferência sobre uma reta (eixo x), o ponto P traça uma curva chamada de ciclóide. Queremos descobrir quanto a curva "caminhou" em cada eixo. A seguir, apresentamos a dedução geométrica para sua parametrização, partindo dos seguintes pressupostos:

- $\theta = 0$ no início, gira θ radianos;
- o ponto P coincide com a origem do sistema de coordenadas no início do movimento;

Podemos então escrever:

Com os deslocamentos de x e y em mãos, precisamos descobrir os segmentos de reta para determinar as parametrizações em relação ao parâmetro θ. Começemos por $|OT|$. Como C está em contato com a reta, deduzimos que:

Podemos deduzir pela figura os outros 3 segmentos restantes:

Agora resta substituir os segmentos encontrados nas equações 1 e 2 e obtemos a parametrização final:

0.3 Modelo Computacional

O modelo desenvolvido pelo grupo encontra-se explicado abaixo. Inicialmente, declaramos as variáveis que irão armazenar as parametrizações, além de definir alguns parâmetros e algumas burocracias de código. Também definimos o dominios das parametrizações. ##### Coloque a sua parametrização e o domínio nas variáveis abaixo

```
In [2]: def retornaParametrizacao (t):
            # Parametrização em X
            xParam = np.cos(t)
            # Parametrização em Y
            yParam = np.sin(t)
            return (xParam, yParam)


        dominioMin = np.pi
        dominioMax = (3*np.pi) / 2

In [3]: v0 = 0
        tTotal = 0
        distTotal = 0
        precisao = 0.01
        delta = np.arange(dominioMin,dominioMax, precisao)
        gravidade = 9.81

        xMax, yMax = retornaParametrizacao(dominioMin)
        listaX = []
        listaY = []
```

Depois, declaramos uma função cujo objetivo é achar os segmentos de reta que usaremos para aproximar a curva. Esses segmentos são, como mostra a imagem, a hipotenusa entre dois pontos da curva. A diferença entre esses pontos é a precisão do cálculo. Quanto menor a diferença entre eles, melhor a aproximação. A função recebe os pontos para os quais calcular o segmento de reta, obtidos através da função acima. Além disso, ela retorna o angulo de inclinação das retas que irá nos ajudar a calcular o tempo.

```
In [4]: def findHipotenusa (x0, x1, y0, y1):
            deltaX = x1 - x0
            deltaY = y1 - y0
            # Em radiano
            angulo = np.arctan(deltaY/deltaX)
            return np.sqrt((deltaX**2) + (deltaY**2)), angulo
```

Então, declaramos uma função que irá nos retornar o tempo de percurso de cada segmento de reta. Ela recebe como parametros o angulo de inclinação, a distancia (hipotenusa) e a velocidade inicial. E ela retorna o tempo de percurso e a velocidade final (que será utilizada como inicial para o próximo segmento na próxima iteração, e assim por diante). Para calcular o tempo, utilizamos a equação: Como já possuimos os parametros ΔS (tamanho do segmento/hipotenusa), $v0$ (inicialmente 0, depois igual a velocidade final no segmento anterior) e a (aceleração da gravidade), só precisamos isolar o t.

```
In [5]: # Formula usada
        # dist = v0*t + (a*(t**2))/2
        def retornaTempo (teta, dist, v0):
            a = gravidade * np.cos(teta)
```

```
        delta = 4*(v0**2) + 8*a*dist
        t1 = (-2*v0 + np.sqrt(delta))/(2*a)
        #t2 = (-2*v0 - np.sqrt(delta))/(2*a)
        v = v0 + a*t1
        return (t1,v)
```

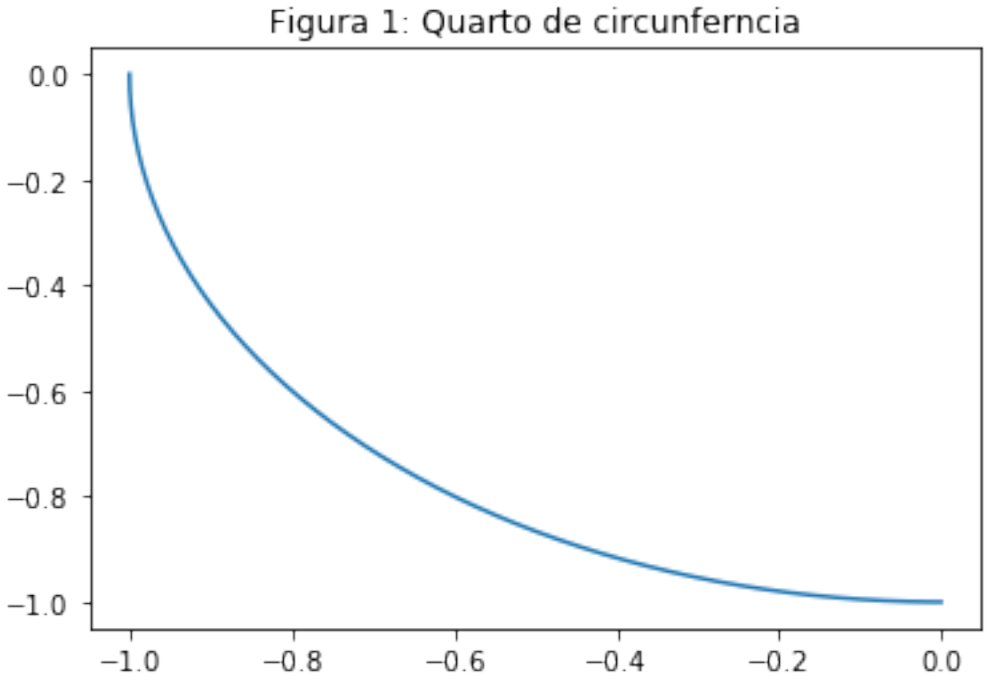
Por fim, contruímos o loop que junta todas essa funções. Ele roda para cada intervalo de precisão definido, e tem o seguinte comportamento: * Primeiro, descobre as coordenadas dos dois pontos do segmento aproximado por reta, com base na precisão pré definida * Depois, descobre o comprimento dessa reta (hipotenusa) e o seu angulo de inclinação, usando seu ponto inicial e final. * Então, calcula o tempo necessário para percorrer tal segmento e o adiciona ao tempo total para percorrer a curva. * Isso é repetido para cada intervalo de precisão definido, até percorrer a curva inteira.

Por fim, imprime os valores (nesse caso, **um quarto de circunferencia**):

```
In [6]: consegueSubir = True

        for t in delta:

            x0, y0 = retornaParametrizacao(t)

            listaX.append(x0)
            listaY.append(y0)
            x1, y1 = retornaParametrizacao(t + precisao)

            if (y1 > yMax):
                consegueSubir = False

            hip, angulo = findHipotenusa(x0,x1,y0,y1)
            distTotal += hip
            t1, v = retornaTempo(angulo, hip, v0)
            v0 = v
            tTotal += t1

        if consegueSubir:
            print("Tempo total (s)")
            print(tTotal)
            print('Distancia total (m)')
            print(distTotal)

        else:
            print('A bolinha não consegue subir essa curva. O Y máximo é {0}'.forma

        plt.plot(listaX,listaY)
        plt.title('Figura 1: Quarto de circunferncia')
        plt.show()
```

```
Tempo total (s)
2.26049822885
```

```
Distancia total (m)
1.57999341667
```



0.4 Validação

Abaixo está a validação da nossa simulação computacional. Nela, utilizamos uma reta parametrizada. Olhando seu domínio, fica claro que a distância percorrida faz sentido. Utilizando pitágoras, pode-se perceber que a distância vale raiz quadradada de 200, o que bate com nosso resultado.

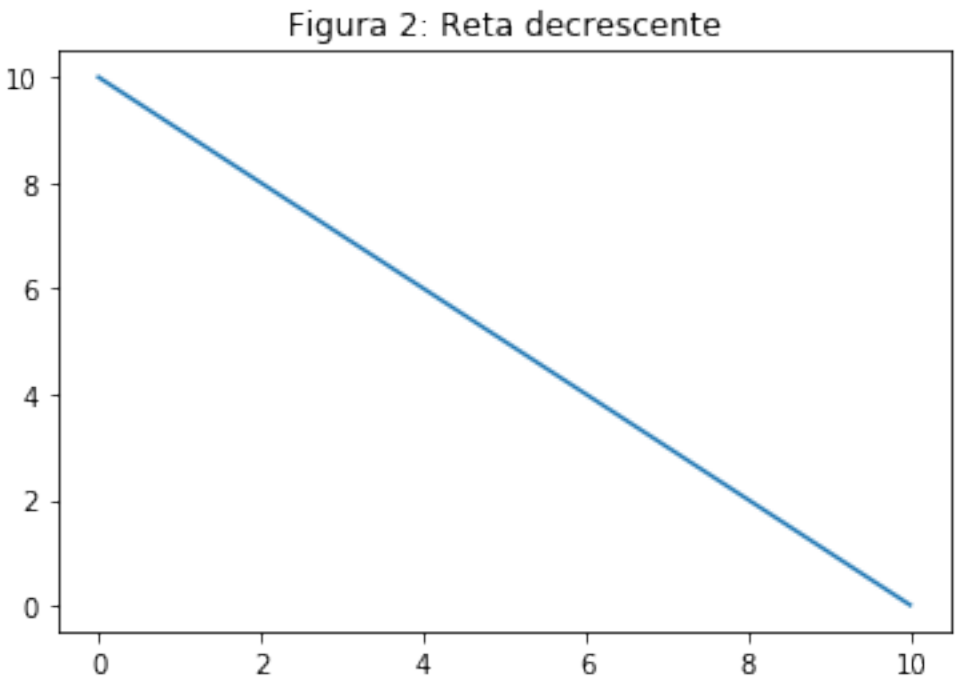
Para o cálculo do tempo, utilizamos uma reta vertical definida com x constante. Em seguida, calculamos o tempo que levaria para a bolinha percorrer o trajeto inteiro e validamos utilizando a física. Esse tempo precisa ser igual ao tempo necessário para ela cair em queda-livre.

0.4.1 Reta para validação da distância

```
In [7]: def retornaParametrizacao (t):
            # Parametrização em X
            xParam = t
            # Parametrização em Y
            yParam = 10 - t
            return (xParam, yParam)


        dominioMin = 0
```

```
        dominioMax = 10
        v0 = 0
        tTotal = 0
        distTotal = 0
        precisao = 0.01
        delta = np.arange(dominioMin,dominioMax, precisao)
        gravidade = 9.81

        xMax, yMax = retornaParametrizacao(dominioMin)
        listaX = []
        listaY = []
        consegueSubir = True

        for t in delta:
            x0, y0 = retornaParametrizacao(t)

            listaX.append(x0)
            listaY.append(y0)
            x1, y1 = retornaParametrizacao(t + precisao)

            if (np.abs(y1) > np.abs(yMax)):
                consegueSubir = False

            hip, angulo = findHipotenusa(x0,x1,y0,y1)
            distTotal += hip
            t1, v = retornaTempo(angulo, hip, v0)
            v0 = v
            tTotal += t1

        if consegueSubir:
            print("Tempo total (s)")
            print(tTotal)
            print('Distancia total (m)')
            print(distTotal)
        else:
            print('A bolinha não consegue subir essa curva. O Y máximo é {0}'.forma

        plt.plot(listaX,listaY)
        plt.title('Figura 2: Reta decrescente')
        plt.show()
```

```
Tempo total (s)
2.01927510938
Distancia total (m)
14.1421356237
```

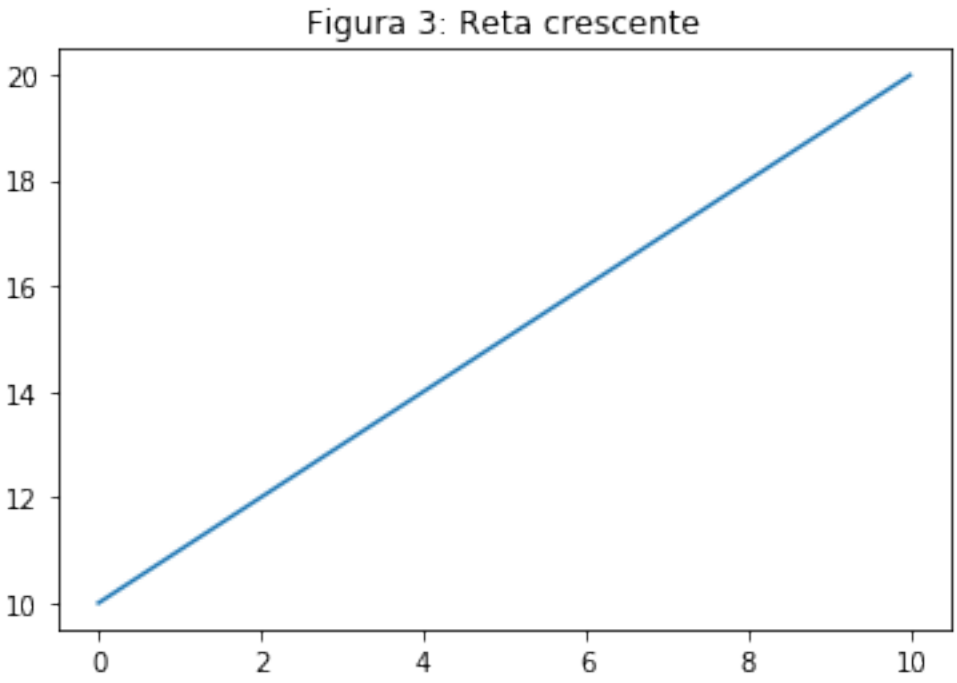


0.4.2 Reta para validação do Y máximo (conservação de energia)

```
In [8]: def retornaParametrizacao (t):
            # Parametrização em X
            xParam = t
            # Parametrização em Y
            yParam = 10 + t
            return (xParam, yParam)


        dominioMin = 0
        dominioMax = 10
        v0 = 0
        tTotal = 0
        distTotal = 0
        precisao = 0.01
        delta = np.arange(dominioMin,dominioMax, precisao)
        gravidade = 9.81

        xMax, yMax = retornaParametrizacao(dominioMin)
        listaX = []
        listaY = []
        consegueSubir = True

        for t in delta:
```

```
            x0, y0 = retornaParametrizacao(t)

            listaX.append(x0)
            listaY.append(y0)
            x1, y1 = retornaParametrizacao(t + precisao)

            if (np.abs(y1) > np.abs(yMax)):
                consegueSubir = False

            hip, angulo = findHipotenusa(x0,x1,y0,y1)
            distTotal += hip
            t1, v = retornaTempo(angulo, hip, v0)
            v0 = v
            tTotal += t1

        if consegueSubir:
            print("Tempo total (s)")
            print(tTotal)
            print('Distancia total (m)')
            print(distTotal)
        else:
            print('A bolinha não consegue subir essa curva. O Y máximo é {0}'.forma

        plt.plot(listaX,listaY)
        plt.title('Figura 3: Reta crescente')
        plt.show()
```

```
A bolinha não consegue subir essa curva. O Y máximo é 10
```



0.4.3 Validação do tempo

Tivemos bastante dificuldade para validar o tempo que a bolinha leva para percorrer determinada curva. Abaixo estão descritas as tentativas realizadas.

1. Calcular o tempo que a bolinha leva para percorrer uma reta. Tivemos problemas para calcular o tempo teórico que levaria (tanto pela literatura matemática quanto pela física), e por isso acabamos trocando de tentativa.

2. Calcular o tempo que a bolinha leva para percorrer uma reta vertical, com x constante. Da mesma forma que na tentativa acima, tivemos problemas. Dessa vez foi com o código. Ele reclamava de nossa equação utilizada na função `retornaTempo`, provavelmente devido a maneira com que dividimos a curva e distribuímos as forças. Sabendo esse tempo, utilizaríamos a fórmula da cinemática de posição em função do tempo para checarmos por valores iguais.

3. Descobrir o tempo real de uma ciclóide e comparar com o tempo calculado pela simulação. Conversamos nosso colega (Eduardo Ferrari) e testamos com a ciclóide construída por ele. Tivemos muita dificuldade para medir o tempo, que da menos de 1 segundo na rampa construída. Além disso, tivemos dificuldade em descobrir os parâmetros exatos da ciclóide construída para equacionar uma semelhante.

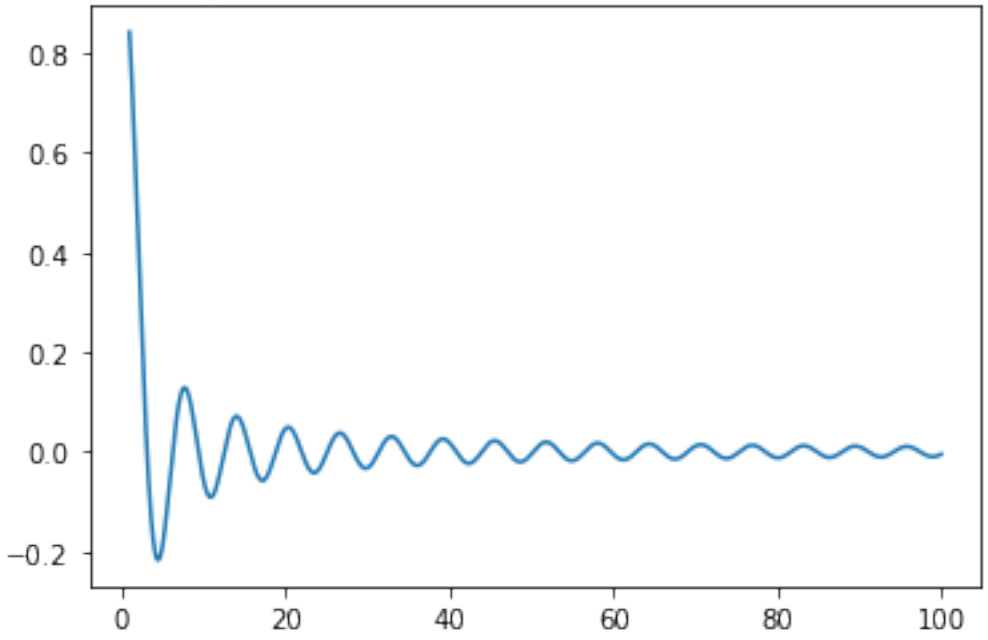
0.5 Plots de Curvas Parametrizadas

0.5.1 1. Curva de um oscilador massa-mola

```
In [9]: def retornaParametrizacao (t):
            # Parametrização em X
            xParam = t
            # Parametrização em Y
            yParam = np.sin(t) / t
            return (xParam, yParam)

        dominioMin = 1
        dominioMax = 100
        v0 = 0
        tTotal = 0
        distTotal = 0
        precisao = 0.01
        delta = np.arange(dominioMin,dominioMax, precisao)
        gravidade = 9.81

        xMax, yMax = retornaParametrizacao(dominioMin)
        listaX = []
        listaY = []
        consegueSubir = True

        for t in delta:
            x0, y0 = retornaParametrizacao(t)

            listaX.append(x0)
            listaY.append(y0)
            x1, y1 = retornaParametrizacao(t + precisao)

            hip, angulo = findHipotenusa(x0,x1,y0,y1)
            distTotal += hip
            t1, v = retornaTempo(angulo, hip, v0)
            v0 = v
            tTotal += t1

        if consegueSubir:
            print("Tempo total (s)")
            print(tTotal)
            print('Distancia total (m)')
            print(distTotal)
        else:
            print('A bolinha não consegue subir essa curva. O Y máximo é {0}'.forma
        plt.plot(listaX,listaY)
        plt.title('Figura 4: Curva de um oscilador massa-mola')
        plt.show()
```

```
Tempo total (s)
4.54522367649
Distancia total (m)
99.2361320764
```

Figura 4: Curva de um oscilador massa-mola



0.5.2 2. Ciclóide

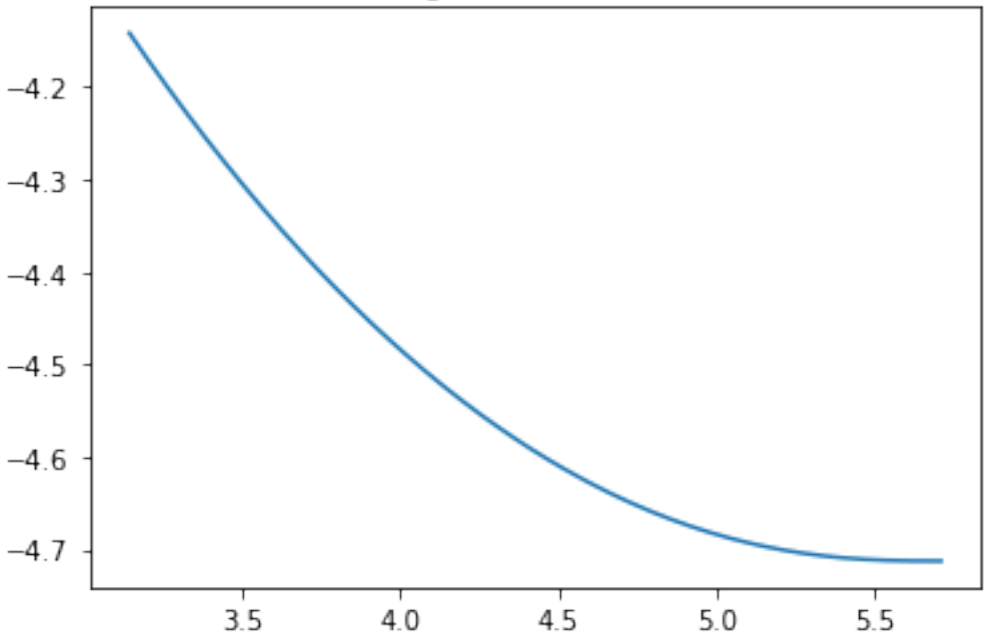
Para r =1:

```
In [10]: def retornaParametrizacao (t):
             # Parametrização em X
             xParam = (t - np.sin(t))
             # Parametrização em Y
             yParam = -(t -np.cos(t))
             return (xParam, yParam)

         dominioMin = np.pi
         dominioMax = 3*np.pi/2
         v0 = 0
         tTotal = 0
         distTotal = 0
         precisao = 0.01
         delta = np.arange(dominioMin,dominioMax, precisao)
         gravidade = 9.81
```

```
        xMax, yMax = retornaParametrizacao(dominioMin)
        listaX = []
        listaY = []
        consegueSubir = True

        for t in delta:
            x0, y0 = retornaParametrizacao(t)

            listaX.append(x0)
            listaY.append(y0)
            x1, y1 = retornaParametrizacao(t + precisao)

            hip, angulo = findHipotenusa(x0,x1,y0,y1)
            distTotal += hip
            t1, v = retornaTempo(angulo, hip, v0)
            v0 = v
            tTotal += t1

        if consegueSubir:
            print("Tempo total (s)")
            print(tTotal)
            print('Distancia total (m)')
            print(distTotal)
        else:
            print('A bolinha não consegue subir essa curva. O Y máximo é {0}'.form
        plt.plot(listaX,listaY)
        plt.title('Figura 5: Cicloide')
        plt.show()
```

```
Tempo total (s)
0.767815164148
Distancia total (m)
2.66911308936
```

Figura 5: Cicloide



0.5.3 3. Elipse

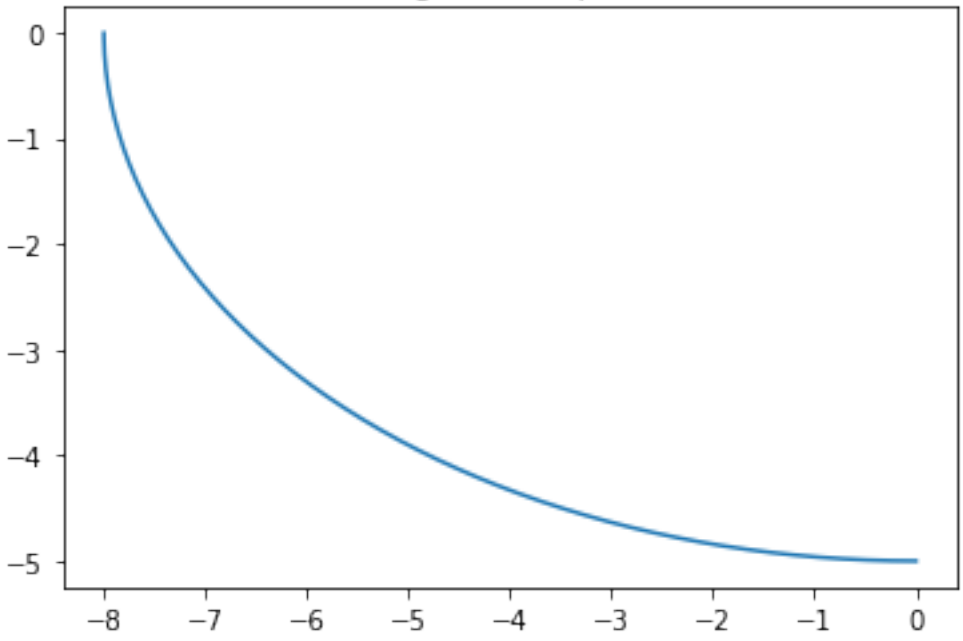
Com a = 8 e b = 5

```
In [11]: def retornaParametrizacao (t):
             # Parametrização em X
             xParam = 8 * np.sin(t)
             # Parametrização em Y
             yParam = 5 * np.cos(t)
             return (xParam, yParam)


         dominioMin = np.pi
         dominioMax = 3*np.pi/2
         v0 = 0
         tTotal = 0
         distTotal = 0
         precisao = 0.01
         delta = np.arange(dominioMin,dominioMax, precisao)
         gravidade = 9.81

         xMax, yMax = retornaParametrizacao(dominioMin)
         listaX = []
         listaY = []
         consegueSubir = True
```

```
        for t in delta:
            x0, y0 = retornaParametrizacao(t)

            listaX.append(x0)
            listaY.append(y0)
            x1, y1 = retornaParametrizacao(t + precisao)

            if (np.abs(y1) > np.abs(yMax)):
                consegueSubir = False

            hip, angulo = findHipotenusa(x0,x1,y0,y1)
            distTotal += hip
            t1, v = retornaTempo(angulo, hip, v0)
            v0 = v
            tTotal += t1

        if consegueSubir:
            print("Tempo total (s)")
            print(tTotal)
            print('Distancia total (m)')
            print(distTotal)
        else:
            print('A bolinha não consegue subir essa curva. O Y máximo é {0}'.form
        plt.plot(listaX,listaY)
        plt.title('Figura 6: Elipse')
        plt.show()
```

```
Tempo total (s)
1.48227279029
Distancia total (m)
10.3925447961
```

Figura 6: Elipse



1 Conclusões

Um dos objetivos do projeto era identificar se a bolinha caía mais rápido em uma ciclóide do que em outra curva qualquer. Comparando as figuras 1 e 5, percebemos que de fato a ciclóide apresenta um menor tempo de queda, com 0.767 segundos em comparação aos 2.260 segundos da circunferência. Entretanto, precisamos analisar os eixos. Tivemos problemas na hora de plotá-las com ponto final e inicial definidos, já que nosso código foi estruturado com base em domínios. Dessa forma, não podemos comparar seus tempos com exatidão.

O modelo computacional criado é válido. Entretanto, tivemos algumas dificuldades em validá-lo usando a física e a matemática. Por conta disso não conseguimos corrigir alguns problemas, principalmente os relacionados com a física do projeto. Acreditamos ter criado um método correto, porém não podemos ter certeza pela falta de uma validação coerente. Em uma segunda iteração desse projeto, iremos verificar a equação utilizada na função `retornaTempo` e nos aprofundar em alguma validação.