2. Sequências e séries

2.1 Introdução

Uma sequência é um conjunto ordenado de números reais denotados por uma lista entre parênteses. Por exemplo, $(a_n) = (a_1, a_2, a_3, \ldots)$ denota a sequência cujo n-ésimo termo é o número a_n. Note que o índice n é sempre um número inteiro positivo, isto é, sempre percorre os valores $1, 2, 3, 4, \ldots$.

Veja alguns exemplos de sequência, completando os primeiros termos de cada uma conforme indicado:

- (a_n), onde $a_n = n^2$. Então, $(a_n) = (a_1, a_2, a_3, a_4, \ldots) = (1, 4, 9, 16, \ldots)$.
- (b_n), onde $b_n = \frac{1}{n+4}$ e $n \in N, 1 \leq n \leq 5$. Então, $(b_n) = (\frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}, \frac{1}{9})$.
- (c_n), onde $c_n = \frac{(-1)^n}{n!}$. Então $(c_n) = \left(1, -1, \frac{1}{2}, -\frac{1}{6}, \cdots\right)$
- (d_n), onde $d_n = 5$. Então $(d_n) = (5, 5, 5, 5, \ldots)$

Mais formalmente, uma sequência também pode ser definida como uma função $f : \mathbb{N} \to \mathbb{R}$, onde $\mathbb{N} = \{1, 2, 3, \ldots, n, \ldots\}$ e $\mathbb{R}$ é o conjunto dos números reais. Assim, a sequência é dada pelas imagens $f(n)$, isto é, $(f(1), f(2), f(3), \ldots)$.

Observe que alguns autores pondem usar chaves para denotar sequências, como em Thomas (2012), no entando é mais como que chaves seja, usadas para conjuntos não ordenados e parênteses para sequências.

2.2 Computando os termos de uma sequência

Vejamos como computar os n primeiros termos da sequência $a_n = n^2$.

(Note a diferença: aqui neste notebook, células que contém código são marcadas com "In [x]" à sua esquerda, onde x é um número, ao contrário das células que contém texto).

Agora basta escrever o termo geral da sequência em Python e utilizar um loop para imprimir o número de termos desejado. Podemos acrescentar comentários nos códigos utilizando o símbolo #.

In [1]:

```
ntermos = 10     # número de termos
for n in range(1, ntermos+1):    # for loop, especificando que n varia entre 1 e
    an = n**2
    print("a_%s =" %(n), an)
```

```
a_1 = 1
a_2 = 4
a_3 = 9
a_4 = 16
a_5 = 25
a_6 = 36
a_7 = 49
a_8 = 64
a_9 = 81
a_10 = 100
```

Analogamente, vamos fazer o mesmo para a sequência (b_n):

In [2]:

```
ntermos = 10     # número de termos
for n in range(1, ntermos+1):    # for loop, especificando que n varia entre 1 e
    bn = 1/(n+4)
    print("b_%s =" %(n), bn)
```

```
b_1 = 0.2
b_2 = 0.16666666666666666
b_3 = 0.14285714285714285
b_4 = 0.125
b_5 = 0.1111111111111111
b_6 = 0.1
b_7 = 0.09090909090909091
b_8 = 0.08333333333333333
b_9 = 0.07692307692307693
b_10 = 0.07142857142857142
```

Exercício 1: Modifique o código acima para computar os 50 primeiros termos das sequências dadas pelo termo geral

a) $\frac{cosn}{n}$

b) $\frac{1}{2^n}$

c) $(-1)^n \frac{1}{n}$

d) $\left(\frac{n+1}{n-1}\right)^n$

e) $\frac{ln(n)}{n}$

f) $\left(1+\frac{1}{n}\right)^n$

In [3]:

```
# copie aqui o código anterior para resolver o exercício
```

2.3 Representando sequências graficamente

A seguir vamos ver alguns exemplos de como visualizar o gráfico dos primeiros elementos de uma sequência como pontos de uma função.

Começamos importando a biblioteca numérica `numpy` e a biblioteca gráfica `matplotlib`.

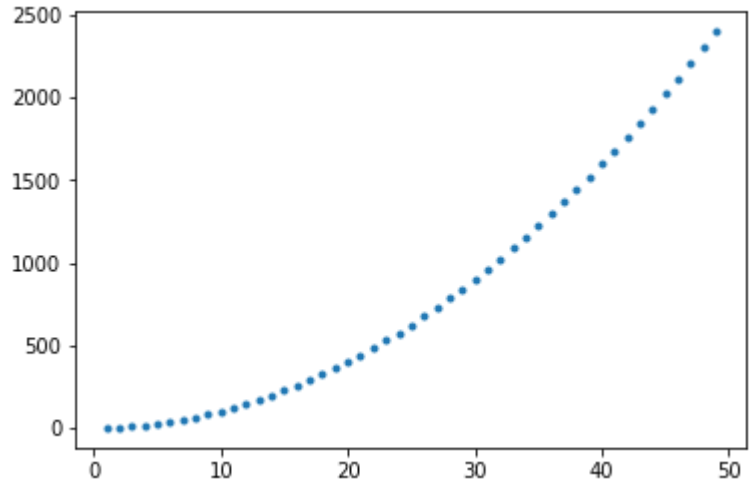
In [4]:

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

Exemplo 1: Considere a sequência dada por $a_n = n^2$ que vimos acima. Vamos plotar um gráfico para visualizar seus primeiros elementos e seu comportamento com o códimo mostrado abaixo.

In [5]:

```
n = np.arange(1, 50)
a = n**2
plt.plot(n, a, '.', linewidth=0.5)
plt.show()
```



Para visualizar os termos da série, basta usar o comando print:

In [6]:

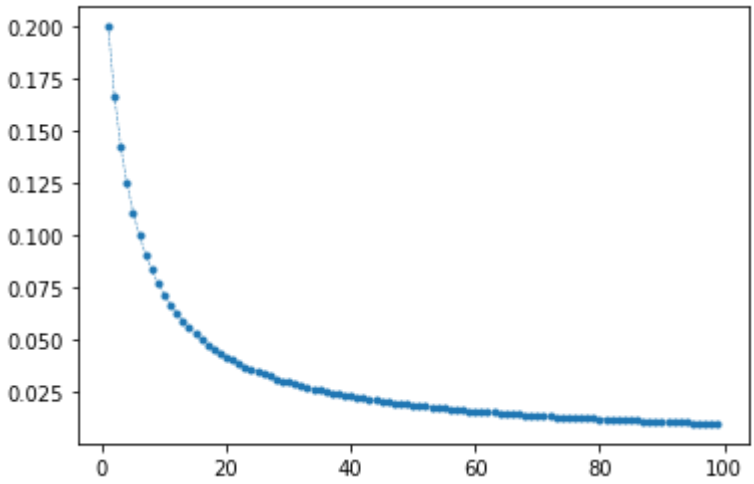
```
print(a)
```

```
[   1    4    9   16   25   36   49   64   81  100  121  144  169
196
  225  256  289  324  361  400  441  484  529  576  625  676  729
784
  841  900  961 1024 1089 1156 1225 1296 1369 1444 1521 1600 1681 1
764
 1849 1936 2025 2116 2209 2304 2401]
```

Exemplo 2: Vamos fazer o mesmo para plotar o gráfico da sequência dada por $b_n = \frac{1}{n+4}$ mas agora incluindo uma linha tracejada entre os pontos e aumentando o número de pontos.

In [7]:

```
n = np.arange(1, 100)
b = 1/(n+4)
plt.plot(n, b, '.--', linewidth=0.5)
plt.show()
```



Observe que os termos da série no Exemplo 1 aumentam cada vez mais eos termos da série do Exemplo 2 parecem se aproximar de 0. Vejamos mais um exemplo.

Importante: As evidências numéricas servem apenas de indicação, elas NÃO permitem concluir definitivamente que uma sequência converge ou diverge. Para isso precisamos argumentar analiticamente. Por exemplo, a sequência $a_n = \frac{2n}{3n-1}$ de fato converge para $\frac{2}{3}$ pois podemos dividir numerador e denominador por n e obter:

$$\lim_{n\to\infty} a_n = \lim_{n\to\infty} \frac{2n}{3n-1} = \lim_{n\to\infty} \frac{\frac{2n}{n}}{\frac{3n-1}{n}} = \lim_{n\to\infty} \frac{2}{3-\frac{1}{n}} = \frac{2}{3}.$$

Exemplo 3: A função $f(n) = \frac{n}{2n+1}$ gera a sequência $\left(\frac{1}{3}, \frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{4}{9}, \ldots, \frac{n}{2n+1}, \ldots\right)$.

É fácil notar que os elementos da sequência se aproximam cada vez mais de 1/2, podemos também concluir que

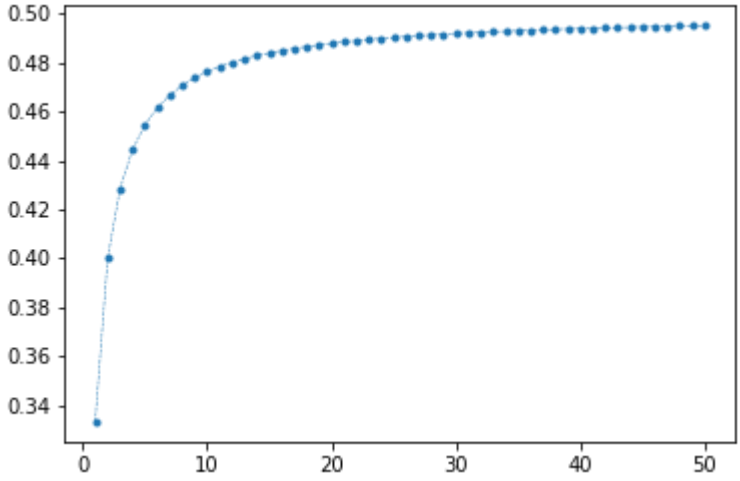
$$\lim_{n\to+\infty} f(n) = 1/2$$

nesse caso, dizemos que a sequência converge para 1/2.

Para completar esse exemplo, vamos usar Python para calcular numericamente os primeiros 50 termos da sequência e plotar o gráfico de f.

In [8]:

```
n = np.linspace(1,50,50)
s = n/(2*n+1)
plt.plot(n,s,'.--', linewidth=0.5)
plt.show()
```



Claramente a função se aproxima cada vez mais de 0.5 a medida que n aumenta. Podemos mostrar analiticamente que isto é verdade calculando o limite do termo geral da série quando $n \to +\infty$.

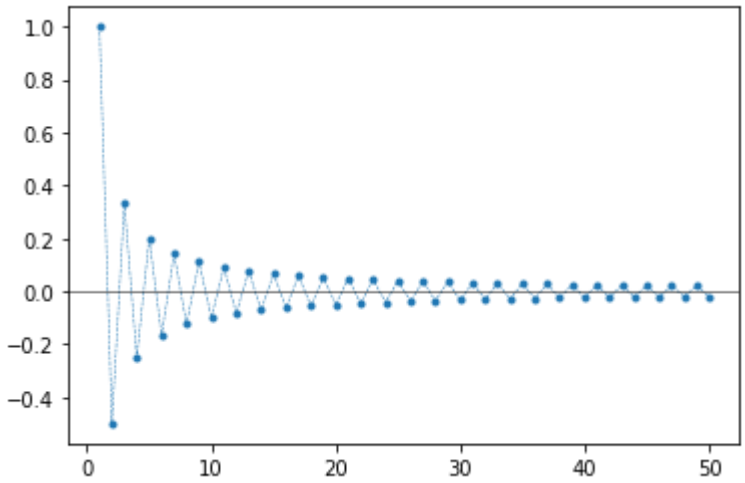
Exemplo 4: Considere a sequência $f(n) = \frac{(-1)^{n+1}}{n}$. Observe que $(-1)^{n+1}$ é igual a -1 se n for par e igual a 1 se n for ímpar. Além disso

$$\lim_{n \to +\infty} f(n) = 0$$

Então os termos dessa sequência vão se aproximando de 0 oscilando entre valores positivos e negativos. Vamos gerar um gráfico para ver:

In [9]:

```
s = (-1)**(n-1)/(n)
plt.plot(n,s,'.--', linewidth=0.5)
plt.axhline(linewidth=0.5, color='black')
plt.show()
```



2.4 Convergência

Dizemos que uma sequência (a_n) converge para a se à medida que n cresce, o termo a_n se torna cada vez mais próximo de a. Nesse caso, escrevemos

$$\lim_{n\to\infty} a_n = a.$$

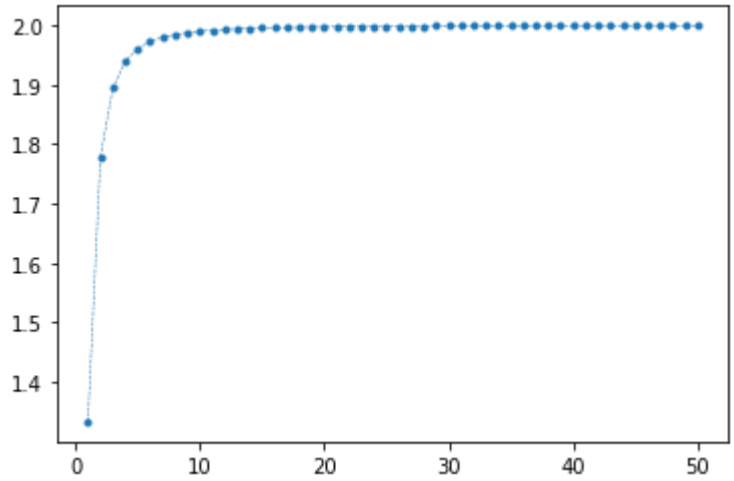
A sequência de termo geral $a_n = n^2$ que estudamos anteriormente, por exemplo, não converge, pois seus termos crescem cada vez mais, não se aproximando de nenhum limite a. Já nos exemplos 2, 3 e 4 as sequências são convergentes.

Exemplo 5: Determine se a sequência $f(n) = \frac{4n^2}{2n^2+1}$ é convergente ou divergente.

Solução: Vamos começar pelo gráfico.

In [10]:

```
s = (4*n**2)/(2*n**2+1)
plt.plot(n,s,'.--', linewidth=0.5)
plt.show()
```



Aparentemente a sequência converge para 2, mas é arriscado afirmar apenas com base em evidências numéricas. Para mostrar que isso é verdade é preciso calcular o limite:

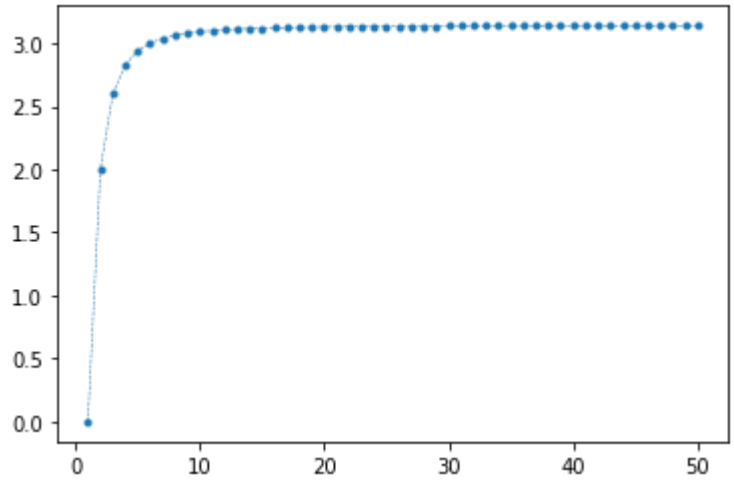
$$\lim_{n\to+\infty} f(n) = \lim_{n\to+\infty} \frac{4n^2}{2n^2+1} = \lim_{n\to+\infty} \frac{4}{1+\frac{1}{n^2}} = 2.$$

Exemplo 6: Determine se a sequência $f(n) = nsen\frac{\pi}{n}$ é convergente ou divergente.

Solução: Vamos começar pelo gráfico.

In [11]:

```
s = n*np.sin(np.pi/n)
plt.plot(n,s,'.--', linewidth=0.5)
plt.show()
```



Aparentemente a sequência converge para algum número próximo de 3. Podemos mostrar a sequência converge para π escrevendo $nsen\frac{\pi}{n}$ como $\frac{sen(\pi/n)}{1/n}$ e calculando o limite usando a regra de L'Hospital, obtemos

$$\lim_{n\to+\infty} f(n) = \lim_{n\to+\infty} \frac{sen(\pi/n)}{1/n} = \lim_{n\to+\infty} \frac{(-\pi/n^2)cos(\pi/n)}{1/n^2} = \lim_{n\to+\infty} \pi cos(\pi/n) = \pi$$

Exercícios:

1. Investigue a convergência das sequências:

(a) $f(n) = \frac{n+1}{2n-1}$

(b) $f(n) = \frac{1}{\sqrt{n^2+1}-n}$

(c) $f(n) = \frac{ln(n)}{n^2}$

(d) $f(n) = \frac{n}{n+1} sen\frac{n\pi}{2}$

2. Utilizando os códigos anteriores, verifique se as seguintes sequências parecem convergir ou não. Em caso positivo, indique o possível limite para a sequência.

(a) $a_n = \frac{2n}{3n-1}$

(b) $a_n = \frac{2^n}{n^2}$

(c) $a_n = \frac{2^n}{n!}$

(d) $a_n = \sqrt[n]{99}$

(e) $a_n = n\,\mathrm{sin}(\frac{\pi}{n})$

Desafio: Dentre as séries acima que convergem, você consegue mostrar analiticamente qual o seu limite?

2.5 Séries numéricas

Uma série é a soma dos infinitos termos de uma sequência (a_n), ou seja, $s = \sum_{n=1}^{\infty} a_n$.

As somas parciais de uma série é uma sequência $s = (s_n)$ onde:

$s_1 = a_1$

$s_2 = a_1 + a_2$

$s_3 = a_1 + a_2 + a_3$

$\vdots$

$s_n = a_1 + a_2 + \ldots + a_n$.

Cuidado para não confundir a série $\sum_{n=1}^{\infty} a_n$ com a sequência (a_n).

Exemplo 7: Computando os termos de uma série

Tomemos a série geométrica $\sum_{n=1}^{\infty} \frac{1}{2^n}$.

In [12]:

```
ntermos = 10      # número de termos
sn = 0
for n in range(1, ntermos+1):
    sn += 1/2**n    # a cada iteração, somamos o respectivo termo da série
    print("n=", n, "Termo somado:  a_%s =" %(n), 1/2**n)
    print("     Soma parcial:  s_%s =" %(n), sn)
    print()
```

```
n= 1 Termo somado:  a_1 = 0.5
     Soma parcial:  s_1 = 0.5

n= 2 Termo somado:  a_2 = 0.25
     Soma parcial:  s_2 = 0.75

n= 3 Termo somado:  a_3 = 0.125
     Soma parcial:  s_3 = 0.875

n= 4 Termo somado:  a_4 = 0.0625
     Soma parcial:  s_4 = 0.9375

n= 5 Termo somado:  a_5 = 0.03125
     Soma parcial:  s_5 = 0.96875

n= 6 Termo somado:  a_6 = 0.015625
     Soma parcial:  s_6 = 0.984375

n= 7 Termo somado:  a_7 = 0.0078125
     Soma parcial:  s_7 = 0.9921875

n= 8 Termo somado:  a_8 = 0.00390625
     Soma parcial:  s_8 = 0.99609375

n= 9 Termo somado:  a_9 = 0.001953125
     Soma parcial:  s_9 = 0.998046875

n= 10 Termo somado:  a_10 = 0.0009765625
     Soma parcial:  s_10 = 0.9990234375
```

Assim, vemos que cada vez somamos termos menores (isto é, a_n tende a 0) e as somas parciais se aproximam cada vez mais de 1. Com isso, temos evidências numéricas de que $\sum_{n=1}^{\infty} \frac{1}{2^n} = 1$.

É possível provar analiticamente que essa afirmação é verdadeira.

Exercícios:

3. Estude computacionalmente o comportamento das séries:

(a) $\sum_{n=1}^{\infty} \frac{1}{n}$ (série harmônica)

(b) $\sum_{n=1}^{\infty} \frac{1}{n^2}$ (p-série)

2.6 Série de Taylor

A expansão em série de Taylor de uma função é uma representação por uma série infinita de polinômios em torno de um ponto. Definimos a série de Taylor de uma função $f(x)$ em torno de $x = a$ como:

$$f(x) = \sum_{n=0}^{\infty} f^{(n)}(a)\frac{(x-a)^n}{n!},$$

onde $f^{(n)}$ é a n-ésima derivada de f e $f^{(0)}$ é a própria função f.

Assim, a forma expandida fica:

$$f(x) = f(a) + f^{(1)}(a)(x-a) + f^{(2)}(a)\frac{(x-a)^2}{2!} + f^{(3)}(a)\frac{(x-a)^3}{3!} + \cdots$$

Exemplo 5:

Calcule a expansão da série de Taylor para $f(x) = 3x^2 + 2x + 1$ em torno de $a = 0$ e $a = 1$. Verifique que f e suas expansões são idênticas.

Calculando as derivadas:

$$\begin{aligned} f^{(0)}(x) &= 3x^2 + 2x + 1 \\ f^{(1)}(x) &= 6x + 2 \\ f^{(2)}(x) &= 6 \\ f^{(3)}(x) &= 0 \\ f^{(4)}(x) &= 0 \\ &\vdots \end{aligned}$$

Em torno de $a = 0$:

$$\begin{aligned} f(x) &= f^{(0)}(0)\frac{(x-0)^0}{0!} + f^{(1)}(0)\frac{(x-0)^1}{1!} + f^{(2)}(0)\frac{(x-0)^2}{2!} + \cdots \\ &= \frac{x^0}{0!} + \frac{2x^1}{1!} + \frac{6x^2}{2!} + 0 + 0 + \cdots \\ &= 1 + 2x + 3x^2 \end{aligned}$$

Em torno de $a = 1$:

$$\begin{aligned} f(x) &= f^{(0)}(0)\frac{(x-1)^0}{0!} + f^{(1)}(1)\frac{(x-1)^1}{1!} + f^{(2)}(1)\frac{(x-1)^2}{2!} + \cdots \\ &= 6 + 8\frac{(x-1)^0}{0!} + 6\frac{(x-1)^1}{1!} + 6\frac{(x-1)^2}{2!} + 0 + 0 + \cdots \\ &= 6 + 8(x-1) + 3(x^2 - 2x + 1) \\ &= 3x^2 + 2x + 1 \end{aligned}$$

Note que a expansão em série de Taylor de qualquer polinômio é finita, pois a derivada n-ésima de qualquer

Exemplo 9: Escreva a série de Taylor para $sen(x)$ em torno do ponto $a = 0$.

Seja $f(x) = \sin(x)$ a sua expansão da série de Taylor é:

$$f(x) = \frac{sen(0)}{0!}x^0 + \frac{cos(0)}{1!}x^1 + \frac{-sen(0)}{2!}x^2 + \frac{-cos(0)}{3!}x^3 + \frac{sen(0)}{4!}x^4 + \frac{cos(0)}{5!}x^5 + \cdots$$

$$= \frac{0}{0!}x^0 + \frac{1}{1!}x^1 + \frac{-0}{2!}x^2 + \frac{-1}{3!}x^3 + \frac{0}{4!}x^4 + \frac{1}{5!}x^5 + \cdots$$

$$= \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots$$

ou, de forma compacta,

$$f(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Observe que nessa fórmula, os termos de potências pares na série de Taylor são ignorados, assim, o termo $n = 0$ da fórmula é o termo $n = 1$ na série Taylor, e o termo $n = 1$ da fórmula é o termo $n = 3$ na série Taylor.

Obtendo a série de Taylor com SymPy

Vamos agora usar SymPy para resolver os dois exemplos anteriores. Primeiramente importamos a bibliotece, iniciamos o modo de impressão, e definimos os símbolos. Iste é feito como nas três linhas a seguir.

In [13]:

```
import sympy
sympy.init_printing(use_unicode=True)
x, a = sympy.symbols("x, a", real=True)
```

Definimos a função do **Exemplo 5** e obtemos das derivadas

In [14]:

```
f0 = 3*a**2+2*a+1
f1 = sympy.diff(f0, a)
f2 = sympy.diff(f1, a)
df=(f0,f1,f2)

print("Derivadas: ")
df
```

```
Derivadas:
```

Out[14]:

$$\left(3a^2 + 2a + 1,\ 6a + 2,\ 6\right)$$

Agora vamos desenvolver a série de Taylor:

In [15]:

```
s = 0
for n in range(len(df)):
    s += df[n]*(x-a)**n/sympy.factorial(n)
s
```

Out[15]:

$$3a^2 + 2a + 3(-a + x)^2 + (-a + x)(6a + 2) + 1$$

Calculando para $a = 0$ e simplificando

In [16]:

```
a=0
sympy.simplify(s)
```

Out[16]:

$$3x^2 + 2x + 1$$

Por fim, o mesmo para $a = 1$ e simplificando

In [17]:

```
a=1
sympy.simplify(s)
```

Out[17]:

$$3x^2 + 2x + 1$$

Para finalizar essa seção, vamos usar SymPy para resolver os dois exemplos anteriores. Primeiramente importamos a biblioteca, iniciamos o modo de impressão, e definimos os símbolos. Iste é feito como nas três linhas a seguir

In [18]:

```
x, a = sympy.symbols("x, a", real=True)
f = sympy.sin(a)
s = 0

for n in range(6):
    s += f*(x-a)**n/sympy.factorial(n)
    f = sympy.diff(f,a)
sympy.simplify(s)
```

Out[18]:

$$(-a + x)\cos(a) - \frac{(a - x)^5 \cos(a)}{120} + \frac{(a - x)^4 \sin(a)}{24} + \frac{(a - x)^3 \cos(a)}{6} - \frac{(a - x)^2 \sin(a)}{2} + \sin(a)$$

Para facilitar a nossa vida, `sympy` dispõe de uma função para a serie de Taylor. Então, ao invés de todo

In [19]:

```
f = sympy.sin(x)
sympy.series(f, x, 0, 7)
```

Out[19]:

$$x - \frac{x^3}{6} + \frac{x^5}{120} + O\left(x^7\right)$$

Aproximação de funções por série de Taylor

Muitas vezes é útil representar uma função usando a série de Taylor com um número limitado de termos. Especialmente quando conhecemos um ponto da função e suas derivadas nesse ponto. Dessa forma é possível obter uma aproximação para a função na vizinhanca do ponto em torno do qual a serie de Taylor é desenvolvida.

Por exemplo, podemos facilmente obter a proximações para a função $f(x) = e^x$ em torno de $a = 0$ pois sabemos que as derivadas $f^{(n)}(a) = 1$ para $n = 0, 1, 2, \ldots$. Assim, a série de Taylor para e^x em torno de $a = 0$ tem a forma:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Assim, truncando a série a partir do 3º termo podemos aproximar e^x par algum valor de x próximo do ponto $a = 0$ usando o polinômio obtido da série de Taylor. Por exemplo, uma aproximação para $e^{\frac{1}{2}}$ é:

$$e^x \approx 1 + 1/2 + \frac{(1/2)^2}{2} + \frac{(1/2)^3}{6} = 1.6458333333333333$$

O valor fornecido pela calculadora é $e^x = 1.6487212707001282$, então, o erro naaproximação obtida pela série de Taylor com 4 termos é aproximadamente 0.0029, ou seja, na ordem de 0.17% da solução exata. Podemos verificar facilmente que quanto mais termos são usados na aproximação, menor é o erro. Além disso, para um mesmo número de termos da série, quanto mais próximo x estiver de 0, melhor será a aproximação.

Vamos agora usar Python para realizar alguns experimentos numéricos. Primeiramente vamos criar um pequeno programa com um laço `for` para obter algumas aproximações de $e^{0.5}$ pala série de Taylor com número de termos variando de 1 a 10.

In [20]:

```
x = 0.5
soma = 0
fatorial = 1

for n in range(10):
    soma += x**n/fatorial
    fatorial *= n+1

    print('Série com',n+1,'termos:',soma)
```

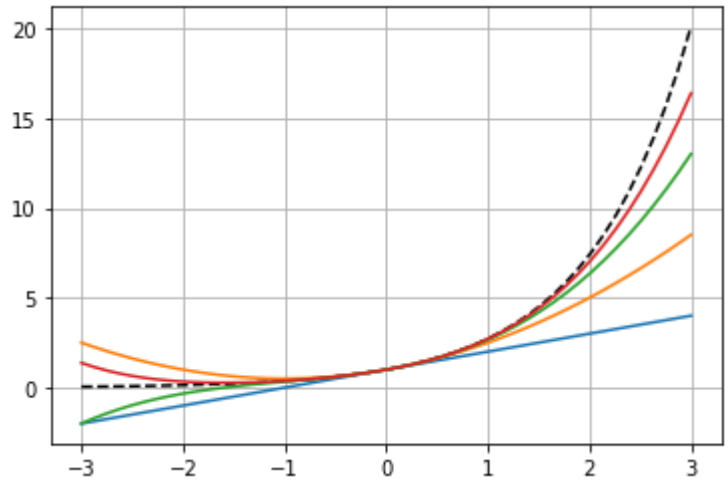
```
Série com 1 termos: 1.0
Série com 2 termos: 1.5
Série com 3 termos: 1.625
Série com 4 termos: 1.6458333333333333
Série com 5 termos: 1.6484375
Série com 6 termos: 1.6486979166666667
Série com 7 termos: 1.6487196180555554
Série com 8 termos: 1.6487211681547618
Série com 9 termos: 1.6487212650359622
Série com 10 termos: 1.648721270418251
```

É possível notar que quanto maior o número de termos, mais o resultado da série de Taylor se aproxima do valor exato de e^x.

Agora vamos escrever explicitamente os polinômios obtidos pela série de Taylor com 2, 3, 4 e 5 termos e fazer o gráfico para comparação.

In [21]:

```
x = np.linspace(-3,3)
s2 = 1 + x
s3 = 1 + x + x**2/2
s4 = 1 + x + x**2/2 + x**3/6
s5 = 1 + x + x**2/2 + x**3/6 + x**4/24
y = np.exp(x)
plt.plot(x,y,'k--',x,s2,x,s3,x,s4,x,s5)
plt.grid()
plt.show()
```



Exemplo 10: Obtenha a série de Taylor para a função $cox(x)$ em torno de $a = 0$ e descubra quantos termos precisam ser retidos para aproximar $cos(0.5)$ com um erro relativo menor que $\epsilon = 0.0001$.

Solução: Desenvolvendo a série de Taylor para a função $cox(x)$ em torno de $a = 0$ chega-se a:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \ldots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

Observe que os termos de potência ímpar são anulados na série de taylor, de modo que para cada valor de n, os termos da série acima correspondem ao termo de número $2n + 1$ na série de Taylor. POr exemplo, para $n = 2$ tem-se o termo $\frac{x^4}{4!}$ que é o 5º termo da série de Taylor.

Vamos agora usar Python para gerar as aproximações, somando um termo por vez e calculando o erro relativo até atingir a precisão desejada. No código abaixo criamos a variável `soma` para armazenar o resultado da série.

In [22]:

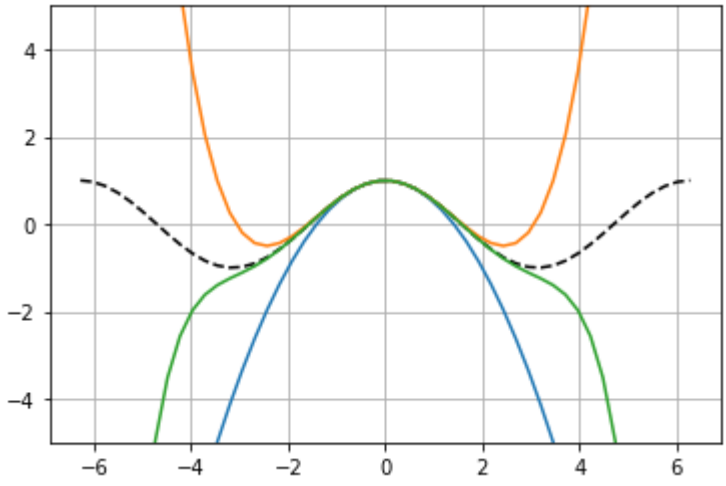
```
x = 0.5
soma = 0
err = 1
n = 0

while err>0.0001:
    soma += (-1)**n * x**(2*n)/np.math.factorial(2*n)
    err = abs(np.cos(x)-soma)/abs(np.cos(x))
    n+=1
    print('Série de Taylor com',2*n+1,'termos:',soma)

print('Valor de np.cos(0.5):',np.cos(0.5))
print('Erro relativo:',err)
```

```
Série de Taylor com 3 termos: 1.0
Série de Taylor com 5 termos: 0.875
Série de Taylor com 7 termos: 0.8776041666666666
Valor de np.cos(0.5): 0.8775825618903728
Erro relativo: 2.461851138807129e-05
```

Para finalizar esse exemplo, vamos fazer o gráfico de dos polinômios obtidos pela série de Taylor com 3, 5 e 7 termos, ou seja, para a somatória mostrada acima, com $n = 1$, $n = 2$ e $n = 3$.

In [23]:

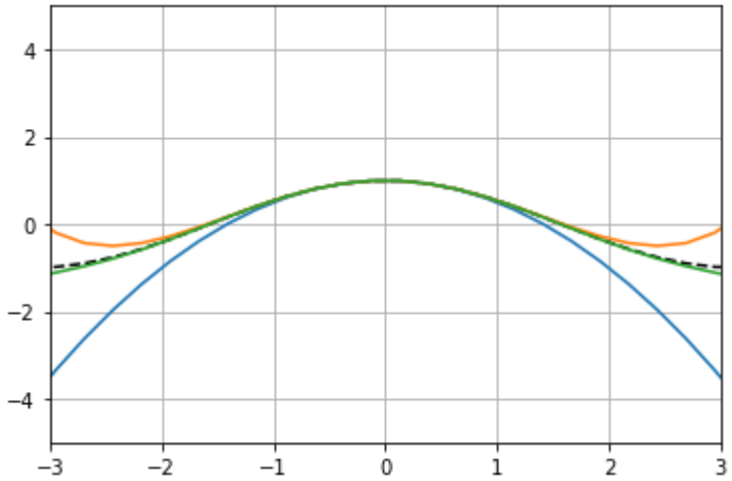
```
x = np.linspace(-2*np.pi,2*np.pi)
s3 = 1 - x**2/2
s5 = 1 - x**2/2 + x**4/24
s7 = 1 - x**2/2 + x**4/24 -x**6/720
y = np.cos(x)
plt.plot(x,y,'k--',x,s3,x,s5,x,s7)
plt.ylim(-5, 5)
plt.grid()
plt.show()
```



Observações: Observe que em torno de $x = 0$ todos os gráficos são próximos do gráfico de $cos(x)$ em preto.O gráfico em azul, um polinômio de Taylor de grau 2, torna-se mais distante de $cos(x)$ do que os outros para valores de x mais distantes de 0.O gráfico em verde, série de Taylor com 7 termos, parece dar uma boa aproximação para $cos(x)$ em um intervalo maior que os outros, entre -3 e 3. Podemos ver melhor se plotarmos novamente, usando apenas valores de x no intervalo $[-3, 3]$.

In [24]:

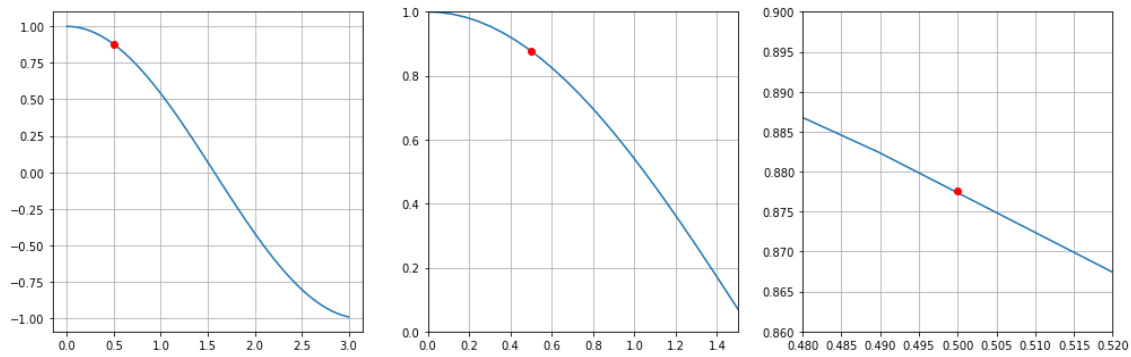
```
plt.plot(x,y,'k--',x,s3,x,s5,x,s7)
plt.ylim(-5, 5)
plt.xlim(-3, 3)
plt.grid()
plt.show()
```



Em muitos casos uma aproximação linear pode ser útil para analisar funções complicadas nas proximidades de um ponto. Assim, é comum usarmos aproximações por série de Taylor de 1ª ordem. De fato, para uma função "suave" se olharmos o gráfico bem de perto a função parece uma reta. Observe, por exemplo, como o gráfico da função $cos(x)$ se parece com uma reta se plotarmos em um intervalo suficientemente pequeno em torno de $x = 0.5$.

In [25]:

```
x = np.linspace(0, 3)
y = np.cos(x)

plt.figure(figsize = (14, 4.5))
plt.subplot(1, 3, 1)
plt.plot(x, y, 0.5, np.cos(0.5),'o r')
plt.grid()

plt.subplot(1, 3, 2)
plt.plot(x, y, 0.5, np.cos(0.5),'o r')
plt.grid()
plt.xlim(0, 1.5)
plt.ylim(0, 1)

plt.subplot(1, 3, 3)
plt.plot(x, y, 0.5, np.cos(0.5),'o r')
plt.grid()
plt.xlim(0.48, 0.52)
plt.ylim(0.86, 0.90)
plt.tight_layout()
plt.show()
```



2.7 Série de Fourier

Funções periódicas, podem ser escritas como uma soma infinita de funções trigonométricas seno e cosseno conhecida como série de Fourier. A representação em série de Fourier de uma função periódica f é dada por:

$$f(x) = \frac{A_0}{2} + \sum_{n=1}^{\infty} A_n\ cos(nx) + B_n\ sen(nx)$$

Mostra-se que os valores de A_n e B_n podem ser calculados pelas seguintes fórmulas:

$$A_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) cos(nx)\ dx$$

$$B_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)sen(nx)\,dx$$

As séries de Fourier são usadas para aproximar algumas funções particularmente desagradáveis, como a função degrau, por exemplo, e formam a base de muitas aplicações de engenharia, como o processamento de sinais. Vamos ver alguns exemplos.

Os coeficiente da série de fourier podem ser obtidos usando integração numérica, por exemplo, por quadratura gaussiana usando a função `quad` disponível em `scipy.integrate` . Vejamos um exemplo a seguir.

Exemplo 11: Use integração numérica para obter uma aproximação por série de Fourier da função $f(x) = sen(e^x)$ no intervalo $(-3, 3)$. Plote o gráfico da função f e da função aproximadora com 10 termos.

In [26]:

```
from scipy.integrate import quad
```

In [33]:

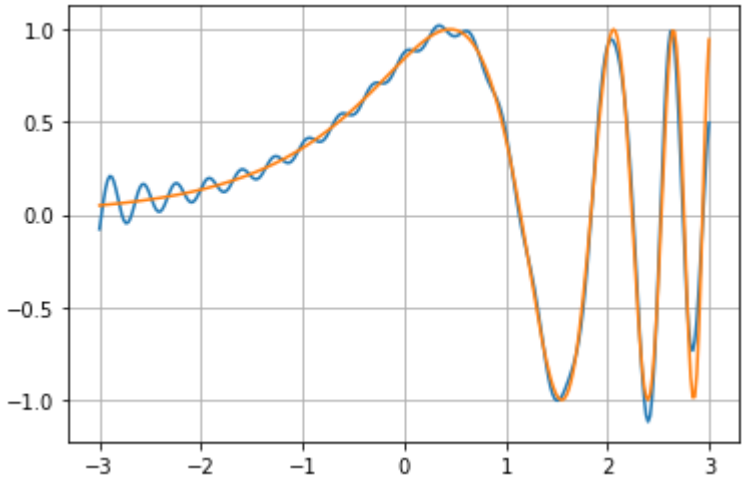
```
f = lambda x: np.sin(np.exp(x))
```

In [34]:

```
def fourier(f,N):
    n=0
    A0 = 1/np.pi * quad(lambda x: f(x)*np.cos(n*x),-np.pi,np.pi)[0]

    xi = np.linspace(-3,3,301)
    yi = []

    soma = A0/2

    for x in xi:
        soma = A0/2
        for n in range(1,N):
            An = 1/np.pi * quad(lambda x: f(x)*np.cos(n*x),-np.pi,np.pi)[0]
            Bn = 1/np.pi * quad(lambda x: f(x)*np.sin(n*x),-np.pi,np.pi)[0]
            soma +=  An*np.cos(n*x)+Bn*np.sin(n*x)
        yi.append(soma)

    plt.plot(xi,yi, xi, f(xi))
    plt.grid()
```

In [35]:

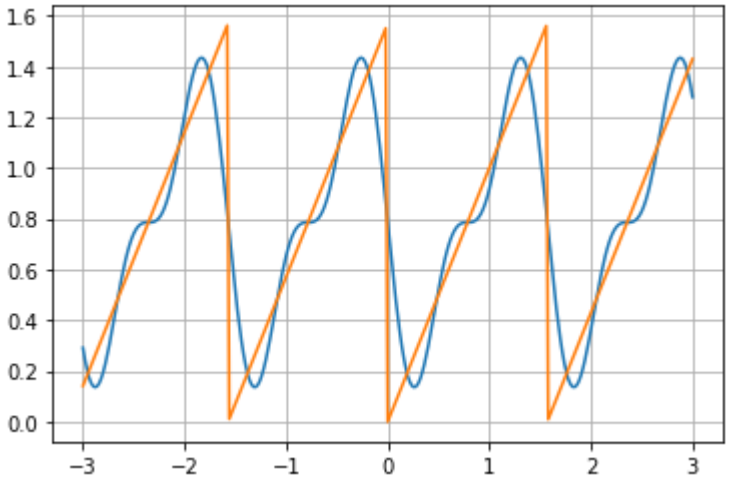
```
fourier(f,20)
```



Exemplo 12:

In [30]:

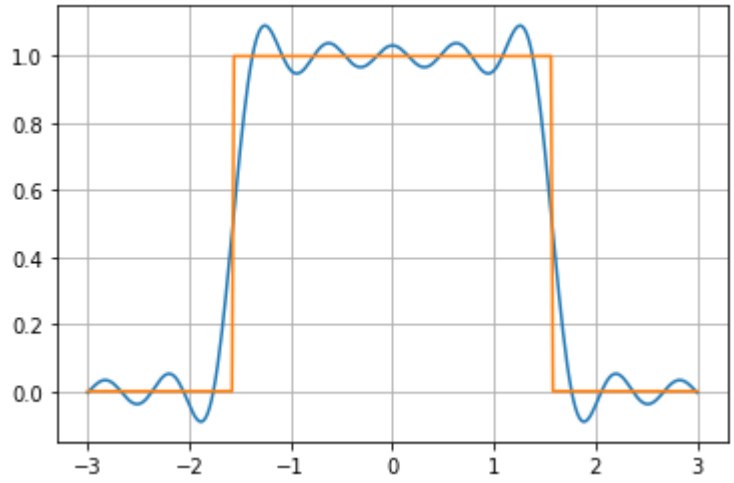
```
f = lambda x: np.mod(x, np.pi/2)
fourier(f,10)
```



Exemplo 13:

In [31]:

```
f = lambda x: (x > -np.pi/2) & (x < np.pi/2)
fourier(f,10)
```



Referências

KONG, Qingkai; SIAUW, Timmy; BAYEN, Alexandre M. Python Programming and Numerical Methods: A Guide For Engineers And Scientists. ed 1, Academic Press, 2021. DOI: https://doi.org/10.1016/C2018-0-04165-1 (https://doi.org/10.1016/C2018-0-04165-1)

LANGTANGEN, Hans Pette. A Primer on Scientific Programming with Python. Texts in Computational Science and Engineering. Springer Berlin, Heidelberg, ed 5, 2016. DOI: https://doi.org/10.1007/978-3-662-49887-3 (https://doi.org/10.1007/978-3-662-49887-3)

In [ ]:

```
```