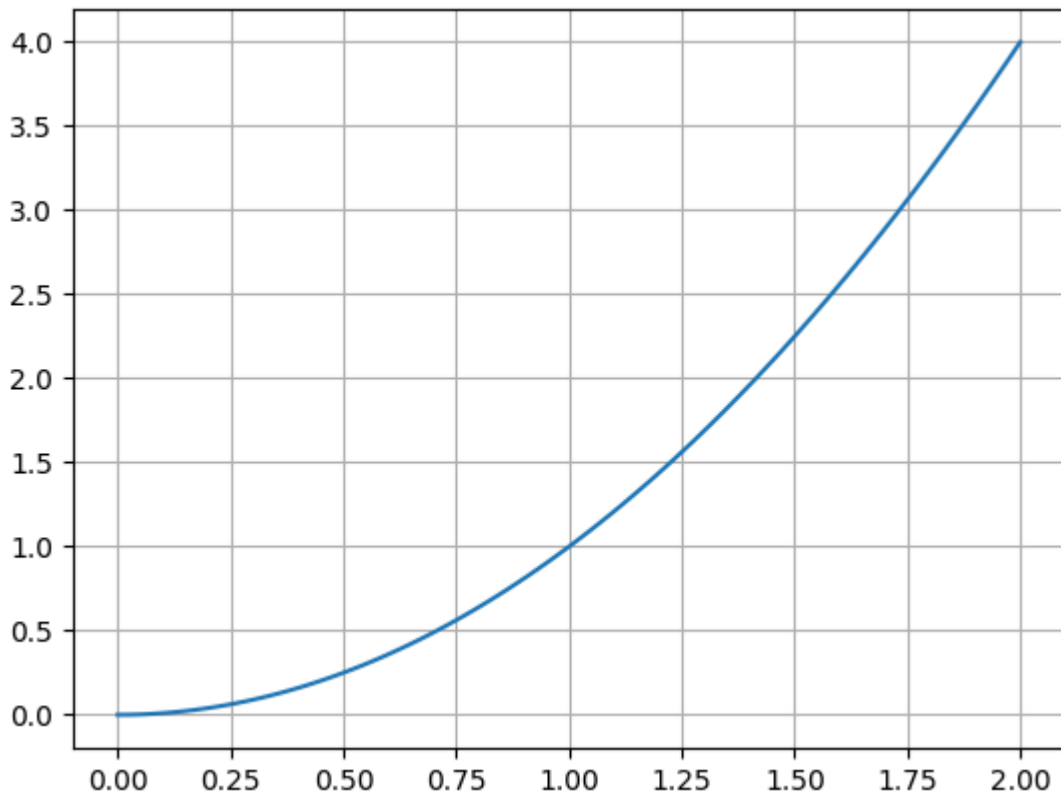


プログラムリスト 5.1 直接関係式を記述してグラフを描く前章までの方法

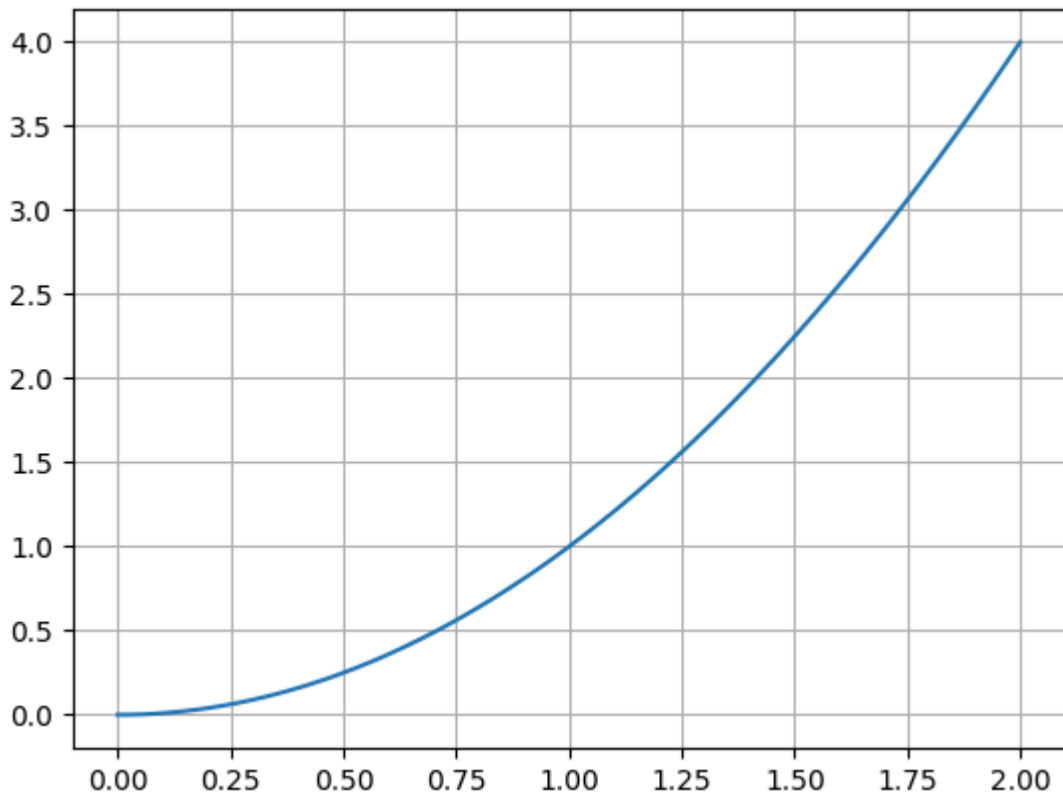
```
In [23]: import numpy as np          # (A1) numpyライブラリをimport
import matplotlib.pyplot as plt    # (A2) matplotlibライブラリをimport
x = np.linspace(0, 2, 100)         # (B1) xの範囲を 0 から +2 に設定
y = x**2                           # (C1) 関数  $y = x^2$  を定義
plt.plot(x, y)                     # (E1)  $y = x^2$  を描画
plt.grid(True)                     # (F1) グリッド線を表示
plt.show()                         # (G) グラフを表示
```



プログラムリスト 5.2 def文で関数を定義してグラフを描く方法

```
In [24]: def f(x):                # (H1) 1つの引数 x を受け取る関数 f(x) を定義
    return x**2                  # (H2) x**2 を計算し、その結果を返す

x = np.linspace(0, 2, 100)       # (B1) xの範囲を 0 から +2 に設定
y = f(x)                         # (H3) f(x) を呼び出して関数値を計算
plt.plot(x, y)                   # (E1) f(x) で定義した関数を描画
plt.grid(True)                   # (F1) グリッド線を表示
plt.show()                       # (G) グラフを表示
```



プログラムリスト 5.3 平均変化率と微分係数

In [25]:

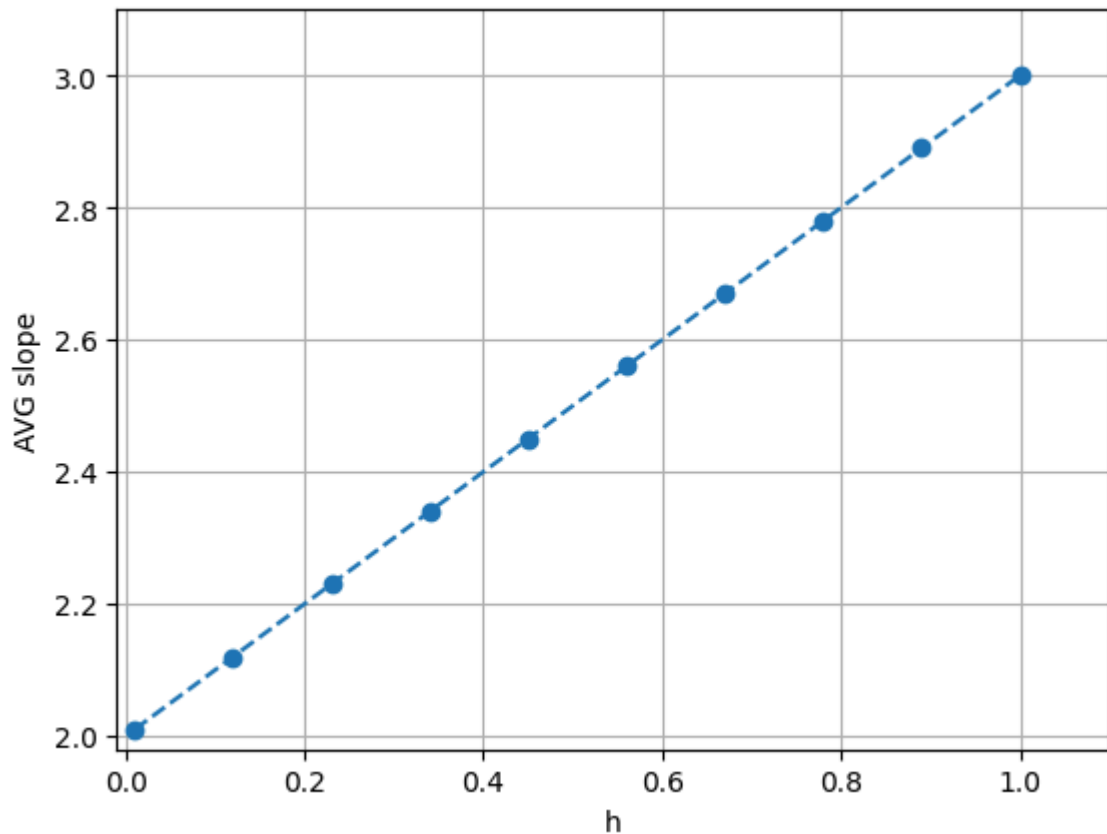
```
import numpy as np          # (A1) numpyライブラリをimport
import matplotlib.pyplot as plt # (A2) matplotlibライブラリをimport

def f(x):                   # (H1) 1つの引数 x を受け取る関数 f(x) を定義
    return x**2             # (H2) x**2 を計算し、その結果を返す
def average_rate(x0, h):    # (H3) 平均変化率を求める関数を定義
    return (f(x0 + h) - f(x0)) / h # (H4) 変化率の式を計算し、結果を返す

h = np.linspace(0.01, 1.0, 10) # (B1) h の範囲として 0.01から 1.0を10分割
x0 = 1                         # (C1) 基準点 x0 を 1 に設定
avg_rates = average_rate(x0, h) # (H5) x0における幅hの平均変化率を計算する関数を呼び出し

plt.plot(h, avg_rates, 'o--') # (E1) 平均変化率を点と破線でプロット

plt.grid(True)               # (F1) グリッド線を表示
plt.xlabel("h")              # (F2) x軸ラベルの設定
plt.ylabel("AVG slope")      # (F3) y軸ラベルの設定
plt.xlim(-0.01, 1.1)         # (F7) x軸の範囲を設定
plt.ylim(1.98, 3.1)          # (F8) y軸の範囲を設定
plt.show()                   # (G) グラフを表示
```



プログラムリスト 5.4 SymPyを用いて導関数を計算する

```
In [26]: import sympy as sp                # (A1) SymPyライブラリをimport
x = sp.symbols('x')                    # (B1) シンボリック変数xを定義

y1 = x**2                             # (B2) 関数 y1 = x^2 を定義
y2 = sp.exp(x)                        # (B3) 関数 y2 = e^x を定義
y3 = sp.sin(x)                        # (B4) 関数 y3 = sin(x) を定義

dy1_dx = sp.diff(y1, x)               # (C1) y1 の導関数を計算
dy2_dx = sp.diff(y2, x)               # (C2) y2 の導関数を計算
dy3_dx = sp.diff(y3, x)               # (C3) y3 の導関数を計算
print(dy1_dx, dy2_dx, dy3_dx)        # (D) 計算結果を表示
```

2*x exp(x) cos(x)

プログラムリスト 5.5 SymPyを用いて多項式・べき乗・積・商の形の関数の導関数を計算する

```
In [27]: import sympy as sp                # (A1) SymPyライブラリをimport
x = sp.symbols('x')                    # (B1) シンボリック変数xを定義

y1 = 3*x**2 - x + 5                   # (B2) 関数 y1 = 3x^2 - x + 5
y2 = 1/x + sp.sqrt(x) + 2/sp.sqrt(x) # (B3) 関数 y2 = 1/x + sqrt(x) + 2/sqrt(x)
y3 = (x**3 - 6*x - 4)*(x**2 + 2)     # (B3) 関数 y3 = (x^3-6x-4)(x^2+2)
y4 = (3*x-7)/(x**2+3*x)              # (B4) 関数 y4 = (3x-7)/(x^2+3x)

dy1_dx = sp.diff(y1, x)               # (C1) y1 の導関数を計算
dy2_dx = sp.diff(y2, x)               # (C2) y2 の導関数を計算
dy3_dx = sp.diff(y3, x)               # (C3) y3 の導関数を計算
dy4_dx = sp.diff(y4, x)               # (C4) y4 の導関数を計算

simplified_dy3 = sp.simplify(dy3_dx)  # (C5) y3 の導関数を整理
simplified_dy4 = sp.simplify(dy4_dx)  # (C6) y4 の導関数を整理
# (D) 計算結果を表示
print(f"[1:] y1 = 3*x**2 - x + 5 の微分:{n {dy1_dx}"])
```

```
print(f"[2:] y2 = 1/x + sqrt(x) +2/sqrt(x) の微分:¥n {dy2_dx}")
print(f"[3:] y3 = (x**3-6x-4)*(x**2+2) の微分:¥n {dy3_dx}")
print(f"> 式の整理後:¥n {simplified_dy3}")
print(f"[4:] y4 = (3*x-7)/(x**2+3*x) の微分:¥n {dy4_dx}")
print(f"> 式の整理後:¥n {simplified_dy4}")
```

```
[1:] y1 = 3*x**2 - x + 5 の微分:
6*x - 1
[2:] y2 = 1/x + sqrt(x) +2/sqrt(x) の微分:
-1/x**2 + 1/(2*sqrt(x)) - 1/x**(3/2)
[3:] y3 = (x**3-6x-4)*(x**2+2) の微分:
2*x*(x**3 - 6*x - 4) + (x**2 + 2)*(3*x**2 - 6)
> 式の整理後:
5*x**4 - 12*x**2 - 8*x - 12
[4:] y4 = (3*x-7)/(x**2+3*x) の微分:
(-2*x - 3)*(3*x - 7)/(x**2 + 3*x)**2 + 3/(x**2 + 3*x)
> 式の整理後:
(-3*x**2 + 14*x + 21)/(x**2*(x**2 + 6*x + 9))
```

プログラムリスト 5.6 SymPyを用いて定数aを含む関数の導関数を計算する

In [28]:

```
import sympy as sp                # (A1) SymPyライブラリをimport
a, x = sp.symbols('a x')         # (B1) シンボリック変数a, xを定義

y1 = a**x                        # (B2) 関数 y1 = a^x
y2 = sp.sin(a*x)                 # (B3) 関数 y2 = sin(ax)

dy1_dx = sp.diff(y1, x)          # (C1) y1 の導関数を計算
dy2_dx = sp.diff(y2, x)          # (C2) y2 の導関数を計算
# (D) 計算結果を表示
print(f"y = a**x の微分: {dy1_dx}")
print(f"y = sin(a*x) の微分: {dy2_dx}")
```

```
y = a**x の微分: a**x*log(a)
y = sin(a*x) の微分: a*cos(a*x)
```

プログラムリスト 5.7 SymPyを用いて関数と接線を描画する

In [29]:

```
import numpy as np                # (A1) numpyライブラリをimport
import matplotlib.pyplot as plt  # (A2) matplotlibライブラリをimport
import sympy as sp               # (A3) SymPyライブラリをimport
x = sp.symbols('x')              # (B1) シンボリック変数xを定義

def f(x):                        # (H1) 1つの引数 x を受け取る関数 f(x) を定義
    return 0.5 * x**2            # (H2) 0.5*x**2 を計算し、その結果を返す

dy_dx = sp.diff(f(x), x)         # (C1) 関数f(x)の導関数を計算
dy_dx_np = sp.lambdify(x, dy_dx, 'numpy') # (C2) numpy形式に変換

x_plot = np.linspace(-1, 2, 200) # (B2) -1から2まで200点を生成
y_plot = f(x_plot)               # (C3) 関数f(x)の値を計算

x0 = 1                           # (C4) 接線をとるx座標
y0 = f(x0)                       # (C5) 接線をとるy座標
trans_y = dy_dx_np(x0)*(x_plot-x0)+y0 # (C6) 接線の式

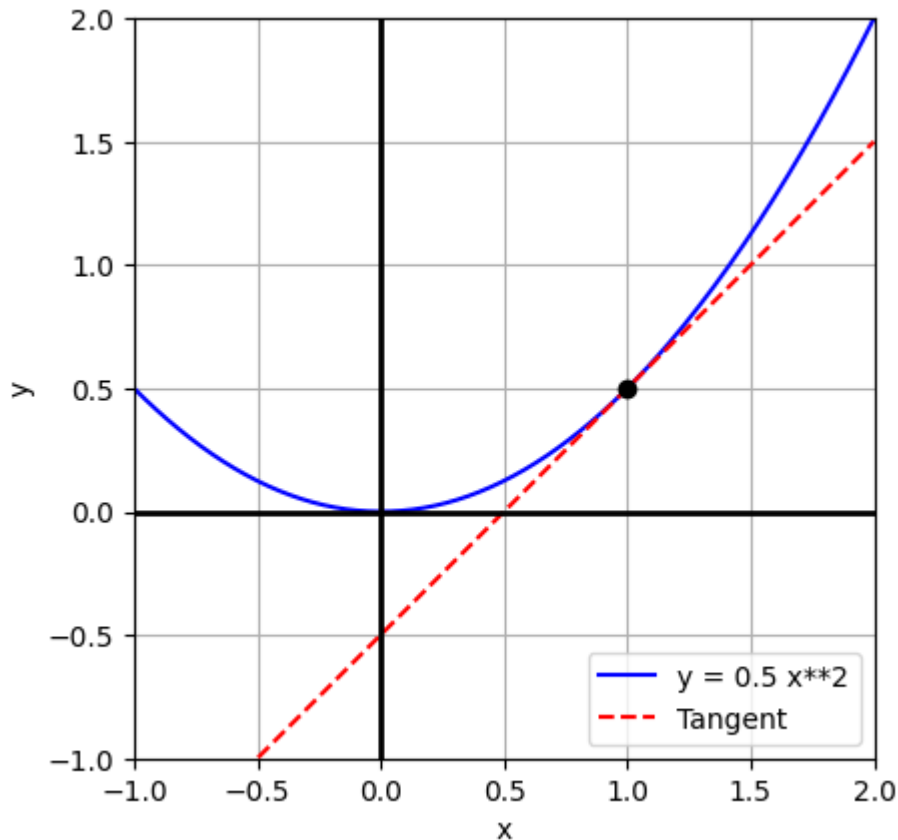
plt.plot(x_plot, y_plot, label="y = 0.5 x**2", color='blue')
# (E1) 関数の描画 (青線)
plt.plot(x_plot, trans_y, label="Tangent", color='red', linestyle='--')
# (E2) 接線の描画 (赤線, 破線)
plt.plot(x0, y0, 'ko')           # (E3) 接点の描画

plt.grid(True)                   # (F1) グリッド線を表示
plt.xlabel("x")                  # (F2) x軸ラベルの設定
```

```

plt.ylabel("y") # (F3) y軸ラベルの設定
plt.axvline(x=0, color='black', linewidth=2) # (F4) y軸の線を太く表示
plt.axhline(y=0, color='black', linewidth=2) # (F5) x軸の線を太く表示
plt.legend() # (F6) 凡例を表示
plt.xlim(-1, 2) # (F7) x軸の範囲を設定
plt.ylim(-1, 2) # (F8) y軸の範囲を設定
plt.gca().set_aspect('equal') # (F12) x軸とy軸のスケールを揃える
plt.show() # (G) グラフを表示

```



プログラムリスト 5.8 $\sin 2x$ の7次および13次のマクローリン展開を比較する

```

In [30]: import numpy as np # (A1) numpyライブラリをimport
import matplotlib.pyplot as plt # (A2) matplotlibライブラリをimport
import math # (A4) mathライブラリをimport (階乗計算に使用)

def f(x): # (H1) 1つの引数 x を受け取る関数 f(x) を定義
    return np.sin(2*x) # (H2) 元の関数 sin(2x)を計算し、その結果を返す
def maclaurin7(x): # (H3) Maclaurin展開：7次まで
    return ((2*x)
            - (2*x)**3 / math.factorial(3)
            + (2*x)**5 / math.factorial(5)
            - (2*x)**7 / math.factorial(7))
def maclaurin13(x): # (H4) Maclaurin展開：13次まで
    return ((2*x)
            - (2*x)**3 / math.factorial(3)
            + (2*x)**5 / math.factorial(5)
            - (2*x)**7 / math.factorial(7)
            + (2*x)**9 / math.factorial(9)
            - (2*x)**11 / math.factorial(11)
            + (2*x)**13 / math.factorial(13))

x_vals = np.linspace(-2*np.pi, 2*np.pi, 500) # (B1) xの範囲を-2πから2πに設定
y_true = f(x_vals) # (C1) 元の関数 sin(2x)を計算
y_m7 = maclaurin7(x_vals) # (C2) 7次のMaclaurin展開を計算
y_m13 = maclaurin13(x_vals) # (C3) 11次のMaclaurin展開を計算

```

(E) 3種類のグラフを描画

```
plt.plot(x_vals, y_true, label="sin(2x)", color='black')
plt.plot(x_vals, y_m7, '--', label="7th-order Maclaurin", color='red')
plt.plot(x_vals, y_m13, '--', label="13th-order Maclaurin", color='blue')
plt.grid(True) # (F1) グリッド線を表示
plt.xlabel("x") # (F2) x軸ラベルの設定
plt.ylabel("y") # (F3) y軸ラベルの設定
plt.axvline(x=0, color='black', linewidth=2) # (F4) y軸の線を太く表示
plt.axhline(y=0, color='black', linewidth=2) # (F5) x軸の線を太く表示
plt.legend() # (F6) 凡例を表示
plt.ylim(-2, 2) # (F8) y軸の範囲
ticks=np.arange(-2*np.pi, 2.5*np.pi, np.pi) # (F9) x軸目盛の範囲
labels=["-2*pi", "-pi", "0", "pi", "2*pi"] # (F10) x軸目盛にpi(π)表記を使用
plt.xticks(ticks, labels) # (F11) x軸をπで表示
plt.show() # (G) グラフを表示
```

