14. オブジェクト指向

プログラミング・データサイエンス I

2022/7/21

1　今日の目的

今日の目的

- オブジェクト指向
- フラクタル図形
- アニメーション

今回は、「Python を使うとこんなこともできる」と言う例を見ていきましょう。後で、自分で Python プログラミングをする際の参考にしてください。

プログラミングの考え方は、様々なものがあります。その考え方にうまく対応したプログラミング言語を使うことで、考え方を素直にプログラムとして書くことが出来ます。オブジェクト指向 (object-oriented) もそのような考え方の一つです。現在使われているプログラミング言語の多くに、オブジェクト指向を支援する機能が備わっています。Python も例外ではありません。今日は、Python のオブジェクト指向機能を見ていきます。

オブジェクト指向プログラミングの例としてフラクタル (fractals) を扱うことにします。Google で"fractals"と言うキーワードで画像を検索して見てください。非常に複雑な図形がたくさん出てきます。特定のモノを差していないことがわかります。これらの図形には、何かが共通しているのです。こうした図形を描く、簡単な手法についてみていきましょう。

フラクタル図形として出来上がったものを見るのは楽しいですが、フラクタルになっていく過程を見ることができれば、もっと楽しいでしょう。そこで、最後にアニメーション、つまり図形を次々に示す方法を見ていきましょう。

`https:/github.com/first-programming-saga/fractals`

2　オブジェクト指向プログラミング

オブジェクト指向プログラミング (OOP, Object Oriented Programming) とは、プログラムを書く際に、データの塊を対象 (object) として捉え、その操作・運動としてプログラム全体を書いていく方法、あるいはそのようなプログラミング手法のことです。オブジェクトの類型・型のことをクラス (class) といい、具体的値が入ったオブジェクトをインスタンス (instance) と言います。オブジェクトの操作・運動を記述する関数をメソッド (method) と言います。

Student
name ID creditEarned
registerCredit()

図 1　`Student` クラスイメージ

学生のクラス `Student` を考えましょう (図 1)。このクラスには、学籍番号や氏名、取得した科目等がデータとして含まれているとします。このクラスの操作として、取得した単位を追加するなどがを考えましょう。

個々の学生の情報は、`Student` クラスのインスタンスです。そのため、学生が入学すると `Student` クラスのインスタンスを生成し、名前と学籍番号を登録します。

`Student` クラスを Python で書くと、ソースコード 2.1 のようになります。5 行目の`__init__()` は、コンストラクタ (constructor) と言い、インスタンスを生成する際に使う特別なメソッドです。16 行目のメソッド `registerCredit()` は取得単位を登録するメソッドです。`self` とは自分自身のインスタンスを表しています。変数 `self.__name` は、自インスタンスに属する変数を表しています。変数の前の`__`は、クラスの外部からその変数が見えないようにするための対策です。

`Student` クラスのインスタンスを使う例がソースコード 2.2 です。二つのインスタンス `bob` と `alice` を生成し、それぞれに取得単位を登録しています。

この例からわかることは、インスタンスの後ろにピリオドをつけて、メソッドや属性を使うことです。これまでの講義でも、ピリオドがしばしば登場していました。オブジェクトに付随した、メソッドや属性を表していたのです。

ソースコード 2.1　Student クラス: 一部省略

```python
class Student:
    """
    学生のクラス
    """
    def __init__(self, name:str, id:str):
        """
        parameters
        ---------
        name: 名前
        id: 学籍番号
        """
        self.__name = name
        self.__id = id
        self.__creditEarned = list()

    def registerCredit(self, lecture:str, unit:int):
        """
        取得単位の登録

        parameters
        ---------
        lecture: 科目名
        unit: 単位数
        """
        self.__creditEarned.append((lecture, unit))
```

ソースコード 2.2　Student クラスの利用

```python
bob = Student('bob',1)
alice = Student('alice',2)
bob.registerCredit('English',2)
bob.registerCredit('Math',2)
alice.registerCredit('French',2)
alice.registerCredit('Sci',2)
print(f"{bob.name}'s credit earned")
for c in bob.creditEarned:
    print(c)
print(f"{alice.name}'s credit earned")
for c in alice.creditEarned:
    print(c)
```

3　フラクタル図形

フラクタルのイメージを見ていると感じる「複雑さ」とは何でしょうか。一つの共通的性質は、あるパターンが大きさを変えて、繰り返し現れていることです。自然界にあるフラクタル図形では、正確に同じ形の繰り返しではなく、ほぼ同じ形が繰り返し現れます。

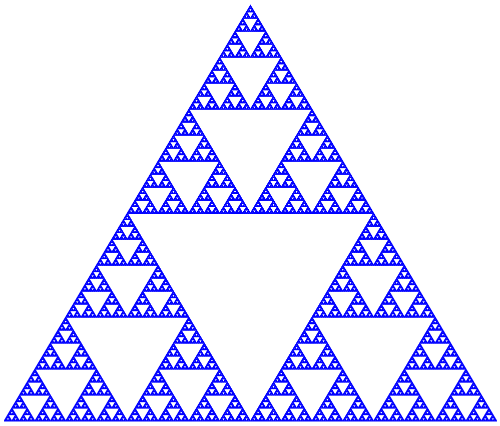

図 2　Sierpinski ガスケット

図 2 は、Sierpinski ガスケットと言うフラクタル図形です。全体と同じ図形、つまり正三角形から中心をクリ抜き、残った正三角形の中心をくり抜くを繰り返した図形が、無数に繰り返されています。このような性質を自己相似 (self-similar) と言います。この図形を生み出す仕組みを見ていきましょう。

図 3　Sierpinski に対応する変換

図 3 を見てください。左の正三角形に対して、中心をくり抜いたものが右になります。

次のステップは、右の各正三角形に同じ操作を行います。この操作を無限回繰り返すとSierpinski ガスケットができます。

私たちが見ている画面は、有限の解像度しかありません。実際には、適当な回数の繰り返しで図 2 のような図形となります。

別の考え方も示しましょう。図 4 をみてください。左の 1×1 の部分を $1/2$ に縮小し、原点をずらして三つ張り合わせています。次のステップは、右の三つの正三角形の全体、つまり各点に対して 1×1 の部分を $1/2$ に縮小し、三つ貼り合わせるのです。これを繰り返します。

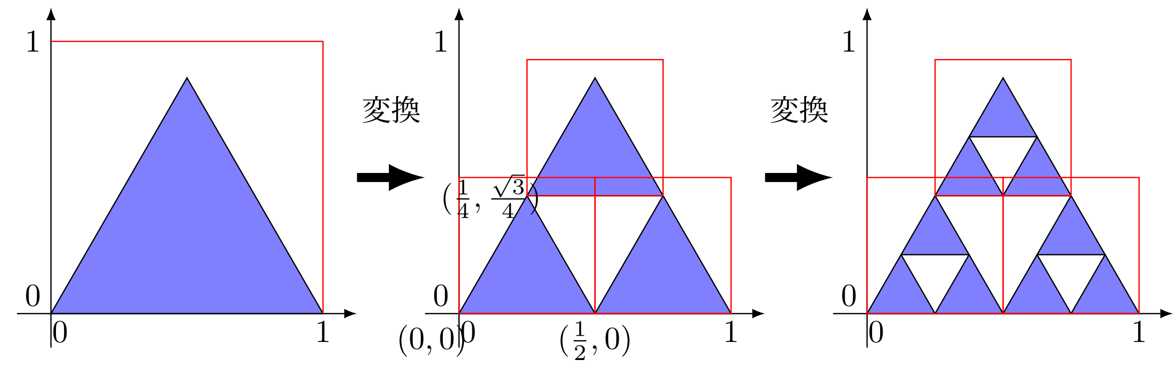

図 4　Sierpinski に対応する変換

後者は以下のような変換に相当します。x 及び y の値が $[0,1)$ の範囲の点 (x,y) を考えます。これに対して以下のような線形変換を行います (図 4)。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{3.1}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1/2 \\ 0 \end{pmatrix} \tag{3.2}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1/4 \\ \sqrt{3}/4 \end{pmatrix} \tag{3.3}$$

式 (3.1)、(3.2)、(3.3) がそれぞれ、左下隅、右下隅、中央上の小さな三角形への変換を表します。この変換では、それぞれの図形を 1/2 に縮小するだけで、回転や捻りを入れていませんから、行列の成分は対角成分は 1/2 だけです。一方、並行移動をしますから、それに対応したベクトルを加えています。

これらの変換は、以下の Affine 変換の特殊な場合です。各パラメタの意味を図 5 に示します。ϕ と ψ は、x 軸及び y 軸からの回転を表しています。$\phi \neq \psi$ の場合には、捻じ

れが生じることを表しています。r と s は、x 軸及び y 軸方向の縮小です。縮小回転の後に、原点を (e,f) へ移動します。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} r\cos\phi & -s\sin\psi \\ r\sin\phi & s\cos\psi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} \tag{3.4}$$

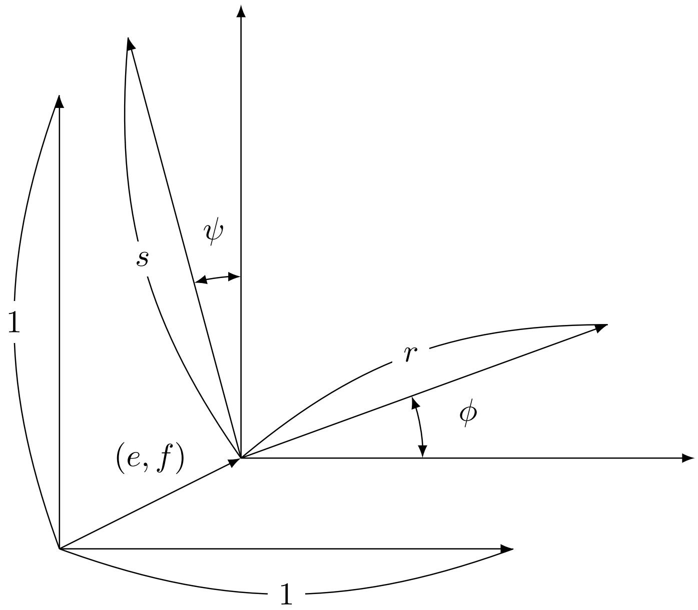

図 5　Affine 変換のパラメタ

角度は 3 時の方向から反時計回りに測ることにします。また、Python の三角関数の引数となる角度は、radian です。$0°$ から $360°$ が、0 から 2π に対応します。つまり、半径 1 の扇型の円周部分の長さで、角度を表すものが radian です。

4　フラクタルクラス

いよいよフラクタルのクラスを定義しましょう。行うべきことは二つです。第一は、Affine 変換のクラスを作っておくことです。Affine 変換を保持し、それに基づいて指定した座標を変換します。第二は、Affine 変換を使って、フラクタル図形を生成するクラスです。

ソースコード 4.1　Affine 変換のクラス: 一部

```
1 class AF:
2     def __init__(self, r:float, s:float, phi:float, psi:float,
3     e:float, f:float):
4         self.__m = np.array([
5                 [r*math.cos(phi),-s*math.sin(psi)],
6                 [r*math.sin(phi),s*math.cos(psi)]])
7         self.__t = np.array([[e], [f]])
8 
9     def trans(self, v:tuple[float, float])->tuple[float,float]:
10         vv = np.array([[v[0]], [v[1]]])
11         r = self.__m @ vv + self.__t
12         return r[0][0], r[1][0]
13 
14     def transList(self,vl:list[tuple[float,float]]):
15         result = list()
16         for v in vl:
17             result.append(self.trans(v))
18         return result
```

4.1　Affine 変換クラス

Affine 変換を行うには、行列とベクトルの演算が必要になります。幸い、Python の `numpy.array` を行列とベクトルのように使うことが出来ます。

Affine 変換のクラスをソースコード 4.1 に示します。コンストラクタでは、Affine 変換のパラメタ (r, s, ϕ, ψ, e, f) を与えます。メソッド `trans()` は、引数で与えた 2 次元面内の座標を変換します。座標は、要素を二つ持つタプルです。11 行目の`@`は行列とベクトルの積を表しています。メソッド `transList()` は、引数で与えた 2 次元面内の座標のリストに対して、それぞれを変換します。

4.2　Fractal クラス

Sierpinski ガスケットでは、最初の正三角形が、一辺の長さが 1/2 の 3 つの正三角形に変換されます。次のステップでは、三つの正三角形がそれぞれ三つに分かれ、一辺の長さが 1/4 の 9 個の正三角形になります。一回の処理で、三角形の数が 3 倍になります。実は、最初が正三角形である必要はありません。図 4 で示したように、元の点の集合を 1/2 に縮小し、原点を移動して貼り付けるだけでした。

ソースコード 4.2　`Fractal` クラス

```python
class Fractal:
    """
    Fractal class
    """
    def __init__(self,afList:list[AF], xy = [(0, 0), (0, 1), (1, 1), (1, 0)]):
        self.__afList = afList
        self.__shapes = [xy]

    def iterate(self):
        """
        Iterate one step
        """
        sp = list()
        for xy in self.__shapes:
            for af in self.__afList:
                sp.append(af.transList(xy))
        self.__shapes.clear()
        self.__shapes = list(sp)

    def getShapes(self)->list[pt.Polygon]:
        """
        Returns list of shapes as patch
        """
        sp = list()
        for xy in self.__shapes:
            sp.append(pt.Polygon(xy,fill=True,color='b'))
        return sp
```

それでは、Fractal クラスをみていきましょう (ソースコード 4.2)。Fractal クラスのコンストラクタでは、Affine 変換のリストと、初期の図形を引数とすることにします。しかし、Fractal 図形は、初期の図形には依存しないため、デフォルト値として、1×1 の正方形を入れておきます。

一回の処理では、現在の図形 (実際には、頂点のリスト) に対して、Affine 変換を行い、新たな図形のリストを生成します。

図形を取り出す際には、`matplotlib.patch` として取り出すことにします。

具体的なフラクタル図形を定義するためには、Affine 変換と初期図形を与える必要があります。ソースコード 4.3 は、初期に正三角形を与える、Sierpinski ガスケットの例です。

ソースコード 4.3　Sierpinski ガスケットを定義する

```python
1  def Sierpinski():
2      p = math.pi/3
3      xy = [(0, 0), (1, 0), (1./2, math.sin(p))]
4      r = s = 0.5
5      phi = psi = 0.
6      af = [
7          AF(r, s, phi, psi, 0, 0),
8          AF(r, s, phi, psi, 1./2, 0),
9          AF(r, s, phi, psi, 1./4, math.sin(p)/2)
10         ]
11     return Fractal(af, xy)
```

3 行目では初期の三角形を定義しています。Affine 変換のパラメタは、

$$r = s = \frac{1}{2} \tag{4.1}$$

$$\phi = \psi = 0 \tag{4.2}$$

です。それらを使って、3 個の Affine 変換を以下のように定義しています。

7 行目　1/2 に縮小するだけの変換であり、左下の図形に対応

8 行目　1/2 に縮小し、右に 1/2 移動する変換であり、右下の図形に対応

9 行目　1/2 に縮小し、$(1/3, \sin(\pi/3)/2)$ 移動する変換であり、上中央の図形に対応

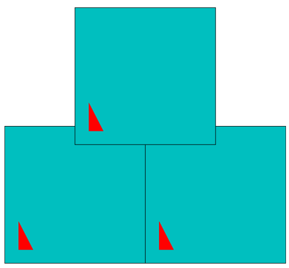

図 6　Sierpinski ガスケットに対応する写像

ソースコード 4.4 は、フラクタル図形を描く部分です。`showMap` を `True` とすると、Affine 変換を表示します。図 6 に、ソースコード 4.3 の場合の写像を示します。図 6 では、回転をしていませんが、赤い三角形の向きで、回転があれば、その角度がわかります。

ソースコード 4.4　フラクタル図形を描くメイン部分

```
tMax = 8
isAnimation = Flase
showMap = True
fractal = Sierpinski()

fig= plt.figure(figsize = (10, 10),facecolor = 'w')
ax = fig.subplots()
if showMap:
    ax = fig.subplots()
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.axis("off")
    for p in fractal.getMap():
        ax.add_patch(p)
    plt.savefig("map.pdf")
    plt.show()
```

showMap を False とすると、繰り返した結果を表示します。例では、8 回繰り返した後の図形を描いています。

初期図形を 1×1 の正方形とする場合をソースコード 4.6 に示します。繰り返しの最初は、初期図形の影響があります。しかし、繰り返し数が増えると、Sierpinski ガスケットになります。

Sierpinski ガスケットの特性を考えると、ソースコード 4.6 のように、3 つの正方形を並べることはあまり意味がないことがわかります。一番大事な点は、1/2 の大きさの図形を 3 個配置するところなのです。そこで、1/2 の大きさの図形を、1×1 の正方形を 4 等分した領域のいずれかに配置することにしましょう。Fractal クラスのコンストラクタで、2 番目の引数を省略すると、初期図形が 1×1 の正方形となります。

課題 1　fractals.ipynb 内の Sierpinski2() で行っていること、特に Affine 変換について理解しなさい。showMap の値を True として写像を確認するとともに、False として、結果を確認しなさい。

一辺の長さを 1/2 にするとともに、回転を加えると、様々な複雑な図形を生成することができます。また、回転を入れると、原点の移動に注意が必要になります。

課題 2　fractals.ipynb 内の Sierpinski3() で行っている変換を図 7 に示します。

ソースコード 4.5　フラクタル図形を描くメイン部分: つづき

```
elif isAnimation:
    imgs = []
    for i in range(tMax):
        ax = fig.subplots()
        ax.clear()
        ax.set_xlim(0, 1)
        ax.set_ylim(0, 1)
        ax.axis("off")
        for p in fractal.getShapes():
            ax.add_patch(p)
        imgs.append(ax.get_children())
        fractal.iterate()
    ani = animation.ArtistAnimation(fig, imgs, interval = 1000)
    display(HTML(ani.to_jshtml()))
else:
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.axis("off")
    for i in range(tMax):
        fractal.iterate()
    for p in fractal.getShapes():
        ax.add_patch(p)
    plt.savefig('fractal.pdf')
    plt.show()
```

ソースコード 4.6　Sierpinski ガスケットを定義する: 初期図形は正方形

```
def Sierpinski1():
    p = math.pi/3
    r = s = 0.5
    phi = psi = 0.
    af = [
        AF(r, s, phi, psi, 0, 0),
        AF(r, s, phi, psi, 1./2, 0),
        AF(r, s, phi, psi, 1./4, math.sin(p)/2)
        ]
    return Fractal(af)
```

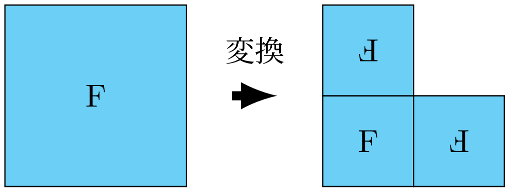

図 7　`Sierpinski3()` に相当する変換

どのような変換かを理解しなさい。

図 8 は、1/2 に縮尺して図形を π 回転させた場合を表している。この状態から、さらに原点を移動し、適切な場所に移動する必要があることに留意しなさい。

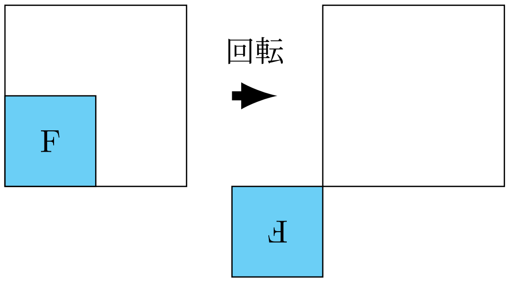

図 8　回転の影響: π 回転した場合

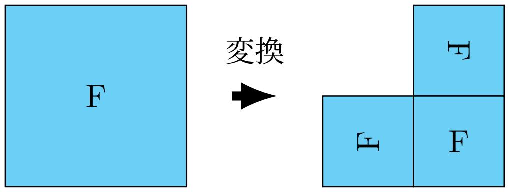

図 9　`Exercise()` に相当する変換

課題 3　確認テスト: 図 9 の回転に対するフラクタル図形を定義し、動作を確認しなさい。

5　アニメーション

Python において、アニメーションとは、紙芝居のようにイメージを次々に示すものを指しています。`isAnimation` を `True` とすることで表示します。

ソースコード 4.5 では、`imgs` という配列に、作図結果を追加し、最後にアニメーションとして再生しています。

付録 A　ベクトルと行列

2 次元の空間を考えます。ベクトル

$$\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix} \tag{付録 A.1}$$

とは、原点から座標 (x, y) への矢印、つまり長さと向きのある量として定義します。

2×2 の行列として

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{付録 A.2}$$

を考えます。ベクトルと行列の積を定義します。

$$A\vec{x} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix} \tag{付録 A.3}$$

積の順序が重要であることを指摘しておきます。

行列 A はベクトルの変換と考えることができます。例えば

$$A = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \tag{付録 A.4}$$

とすると、この行列は x 方向に a 倍、y 方向に b 倍することを表しています。

$$A\vec{x} = \begin{pmatrix} ax \\ by \end{pmatrix} \tag{付録 A.5}$$

また、以下は角度 ϕ の回転を表しています。

$$A = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \tag{付録 A.6}$$

$$\vec{x}' = A\vec{x} = \begin{pmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi + y\cos\phi \end{pmatrix} \tag{付録 A.7}$$

ベクトルと行列は、「線形代数」と言う理工系の学科では初年次必修の科目です。理工学のあらゆる分野で使われます。もちろん、データサイエンスや機械学習を理解する上でも必須の知識・技術です。高校数学では、「数学 C」に行列の初歩が含まれていますから、基礎の部分はそれほど難しくありません。

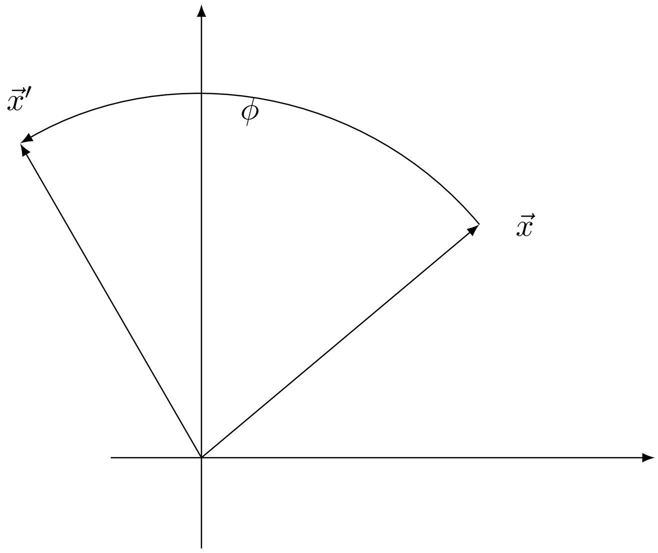

図 10　ベクトルの回転

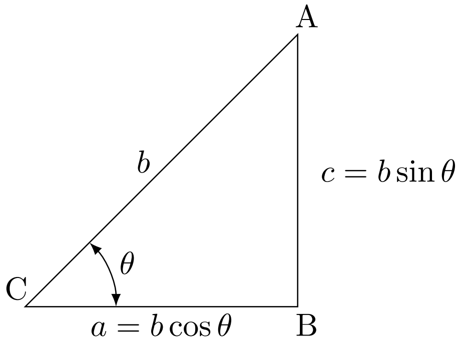

図 11　三角関数

付録 B　三角関数

三角関数は、角度と辺の長さを結びつけます。図 11 を見てください。$\angle\mathrm{ACB} = \theta$ である直角三角形を考えます。辺 AC の長さを b とすると、直角を挟む二辺の長さは、角度 θ の三角関数で表すことができます。

$$a = b\cos\theta \qquad \text{(付録 B.1)}$$

$$c = b\sin\theta \qquad \text{(付録 B.2)}$$

$\sin\theta$ を正弦、$\cos\theta$ を余弦と日本語では言います。直角を挟む二辺の長さの比は

$$\frac{c}{a} = \frac{\sin\theta}{\cos\theta} = \tan\theta \tag{付録 B.3}$$

となり、正接と呼びます。