实验3: 矩阵

大纲

- 创建矩阵
- 矩阵的索引和切片
- 矩阵的加法和减法
- 矩阵的乘法
- 矩阵的转置
- 单位矩阵与逆矩阵
- 矩阵的维度（dimension）与秩（rank）
- 矩阵的简单应用

创建矩阵

矩阵的定义： 矩阵是一个二维数组，其中的每一个元素被行索引和列索引来确定。例如，下面这个矩阵 M 有三行和三列：

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

In [ ]:
```
import numpy as np

#从二维数组来创建numpy矩阵
M = np.array([  [1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

In [ ]:
```
# 创建一些特殊的numpy矩阵

print(np.ones((3,2)))  # 3x2的全1矩阵

print(np.zeros((3,2))) # 3x2的全0矩阵

print(np.random.random((3,2))) # 3x2的随机矩阵

print(np.diag([1, 2, 3])) # 对角矩阵

print(np.eye(3)) # 单位矩阵
```

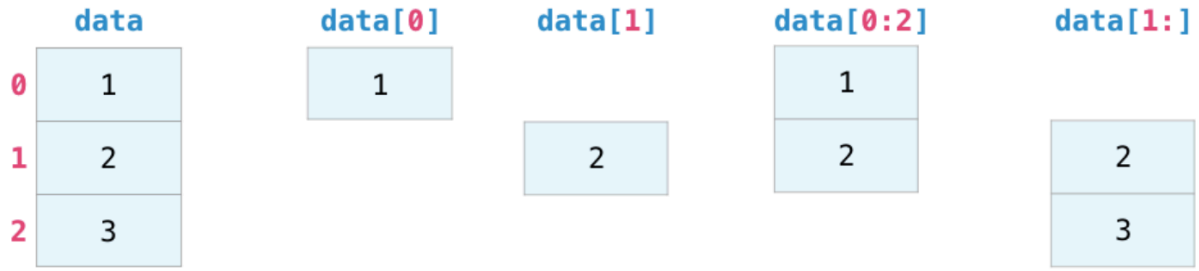
练习3.1

- 创建一个矩阵 A，使得 B 有两行和四列，其中的元素为 $1, 2, 3, 4, 5, 6, 7, 8$。
- 创建任意形状的全零矩阵和全一矩阵。

- 创建对角矩阵，对角线上的元素为 $2, 3, 5$。
- 创建2*2的单位矩阵。

In [ ]:
```
# 在这里编写练习代码


```

向量的索引以及切片



In [13]:
```
import numpy as np

# 创建一个向量
data = np.array([1, 2, 3])

# 通过索引选择元素
print(data[0])   # 选择第一个元素
print(data[-1])  # 选择最后一个元素
print(data[0:2]) # 选择前两个元素
print(data[1:])  # 从索引1开始选择所有元素


```

```
1
3
[1 2]
[2 3]
```

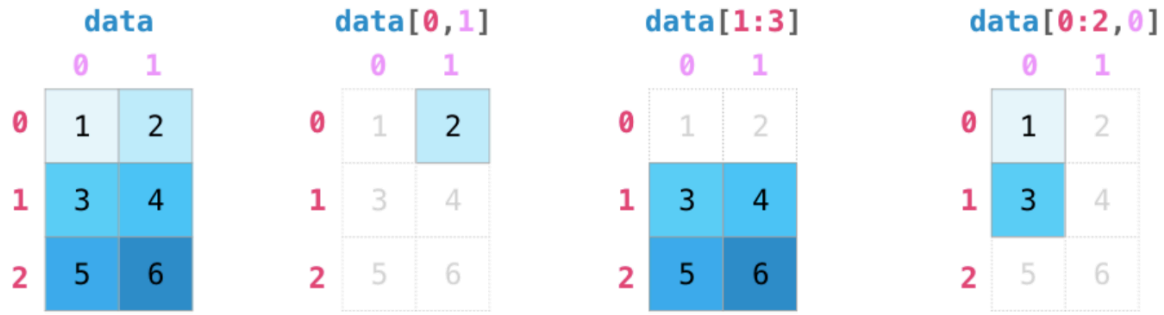
Python语言的切片的语法：

- `start:end:step`，其中 `start` 是起始索引，`end` 是结束索引，`step` 是步长。例如，`1:7:2` 表示从索引1开始，到索引7结束（但是不包括索引7），步长为2。
- start,end,step都可以省略，省略start表示从0开始，省略end表示到最后一个元素结束，省略step表示步长为1。例如，`::2` 表示从0开始，到最后一个元素结束，步长为2。
- start, end, step都可以是负数，负数表示从后往前数。例如，`::-1` 表示从最后一个元素开始，到第一个元素结束，步长为-1，即逆序。

In [14]:
```
import numpy as np

# 创建一个向量包含从0到9的整数
data = np.arange(10)

print(data[0::2]) # 选择所有偶数元素
print(data[1::2]) # 选择所有奇数元素
print(data[::-1])  # 逆序输出
```

```
[0 2 4 6 8]
[1 3 5 7 9]
[9 8 7 6 5 4 3 2 1 0]
```

矩阵的索引和切片



In [16]:
```
data = np.array([[1, 2], [3, 4], [5, 6]])

# 使用行索引和列索引选择到一个元素
print(data[0, 1])

# 使用行索引的切片
print(data[1:3,:])

# 行索引和列索引同时使用切片
print(data[0:2, ::2])
```

```
2
[[3 4]
 [5 6]]
[[1]
 [3]]
```

练习3.2

- 创建一个矩阵 A，使得 A 有四行和四列，其中的元素为从1到16。
- 可以将矩阵分割为4个2*2的子矩阵 B, C, D, E。
- 使用numpy的切片功能提取这四个子矩阵。

In [ ]:
```
1
```

矩阵的加法和减法

相同形状的矩阵可以直接相加或相减。

In [6]:
```
1 import numpy as np
2
3 A = np.array([[1, 2], [3, 4]])
4 B = np.array([[5, 6], [7, 8]])
5 print(f'A + B = \n {A + B} \n') # 矩阵加法
6 print(f'A - B = \n {A - B}') # 矩阵减法
```

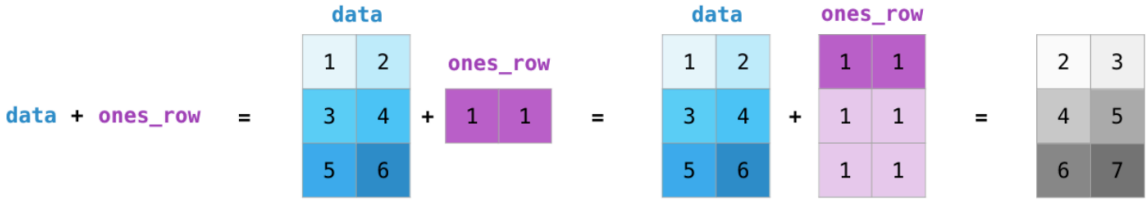
```
A + B =
 [[ 6  8]
 [10 12]]

A - B =
 [[-4 -4]
 [-4 -4]]
```

矩阵和一个标量相加或者相减，会广播到矩阵的每一个元素。

In [7]:
```
1 print(A + 1)
```

```
[[2 3]
 [4 5]]
```

矩阵可以和一个向量相加或者相减，然后向量会被广播到矩阵的每一行。相加时需要矩阵和向量的列数相同。



In [3]:
```
1 import numpy as np
2
3 data = np.array([[1, 2], [3, 4], [5, 6]])
4 print(data + np.array([1, 1]))
```

```
[[2 3]
 [4 5]
 [6 7]]
```

练习3.3

- 创建两个矩阵 A 和 B，使得 A 和 B 的形状都是 2×3，然后计算 $A + B$ 和 $A - B$。
- 计算矩阵 A 和标量 2 的和以及差。
- 计算矩阵 A 和向量 $[1, 2, 3]$ 的和以及差。

In [ ]:
```
# 在这里编写练习代码


```

矩阵的乘法

矩阵乘以一个标量，这是矩阵的数乘。

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$A * 2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * 2 = \begin{bmatrix} 1*2 & 3*2 \\ 3*2 & 4*2 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

In [4]:
```
import numpy as np

A = np.array([[1, 2], [3, 4]])
print(A * 2) # 矩阵数乘
```

```
[[2 4]
 [6 8]]
```

矩阵乘以向量，矩阵的列数需要和向量的维度相同。

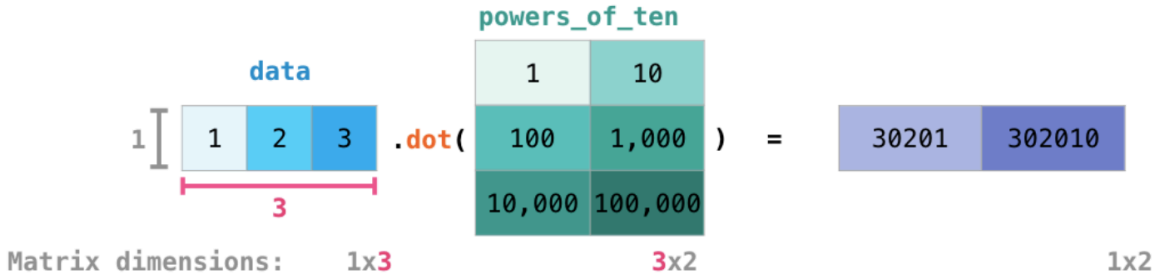
一个更加容易理解的方法是，将矩阵看作是一组列向量，矩阵乘以向量就是将这些列向量线性组合。

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} * \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} * 2 + \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} * 3$$

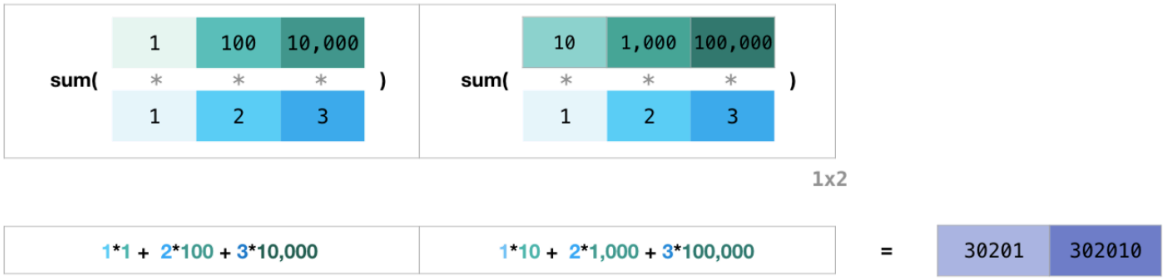
In [19]:
```
import numpy as np

A = np.array([[1, 2], [3, 4], [5, 6]])
b = np.array([2, 3])

# 两种方法来进行矩阵和向量的乘法
print(A.dot(b))
print(A @ b)
```

```
[ 8 18 28]
[ 8 18 28]
```

矩阵与矩阵的乘法本质上与矩阵与向量的乘法相同，向量相当于是一个列数或者行数为1的矩阵。



I've added matrix dimensions at the bottom of this figure to stress that the two matrices have to have the same dimension on the side they face each other with. You can visualize this operation as looking like this:



In [7]:

```
import numpy as np

data = np.array([1, 2, 3])
powers_of_ten = np.array([[1, 10], [100, 1000], [10000, 100000]])

#有两种方法来调用矩阵乘法
print(data.dot(powers_of_ten))
print(data @ powers_of_ten)
```

```
[ 30201 302010]
[ 30201 302010]
```

矩阵的乘法不满足交换律，即 $AB \neq BA$。假设矩阵A的形状是 3 * 2， 矩阵B的形状是 2 * 3，那么 AB 的形状是 3 * 3，而 BA 的形状是 2 * 2。

In [20]:

```
import numpy as np

A = np.array([[1, 2], [3, 4], [5, 6]])
B = np.array([[1, 1, 1], [2, 2, 2]])

print(A.dot(B)) # 矩阵乘法
print(B.dot(A)) # 矩阵乘法
```

```
[[ 5  5  5]
 [11 11 11]
 [17 17 17]]
[[ 9 12]
 [18 24]]
```

练习3.4

- 矩阵A可以和矩阵B什么条件下可以相乘?
- 如果A和B可以相乘，那么AB的形状是什么?
- 如果A和B可以相乘，那么B可以和A相乘吗?
- 使用numpy创建矩阵A和B，验证上面问题的结论。
- 创建一个形状为 2×2 的矩阵 A(包含数值1到4)和一个形状为 2×2 的矩阵 B（包含数值5到8），计算 AB 和 BA， AB 和 BA 是否相等?

In [ ]:
```
# 在这里编写练习代码


```

矩阵的转置与变形

矩阵的转置是将矩阵的行和列互换。例如:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

In [9]:
```
import numpy as np

A = np.array([[1, 2], [3, 4], [5, 6]])
A.T # 转置矩阵
```

Out[9]:
```
array([[1, 3, 5],
       [2, 4, 6]])
```

使用numpy数列的reshape方法可以改变矩阵的形状。

In [12]:
```
v = np.arange(24)
print(v)

M1 = v.reshape(4, 6)
print(M1)

M2 = v.reshape(8, 3)
print(M2)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]]
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]
 [15 16 17]
 [18 19 20]
 [21 22 23]]
```

练习3.5

- 创建一个形状为 2×3 的矩阵 A，然后计算 A^T。
- 创建一个形状为 3×5 的矩阵 A，包含的数值是从1到15。
- 创建一个形状为 3×5 的矩阵 A，包含的数值30以内的偶数。
- 创建一个对角矩阵B，将其转置后会得到什么？
- 使用 numpy 验证你的结论。

In [ ]:
```
# 在这里编写练习代码


```

单位矩阵与逆矩阵

单位矩阵必须是方阵，单位矩阵的对角线上的元素都是1，其他元素都是0。单位矩阵乘以任何矩阵（包括左乘和右乘）都等于原矩阵。

In [22]:
```
import numpy as np

# 创建一个单位矩阵
I = np.eye(3)
print(I)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

任何向量或者矩阵乘以单位矩阵都等于原向量或者矩阵。

In [24]:
```
print(I @ np.array([1, 2, 3]))
print(np.array([1, 2, 3]) @ I)
```

```
[1. 2. 3.]
[1. 2. 3.]
```

逆矩阵是指一个矩阵的倒数，A的逆矩阵记为A^{-1}

- 不是所有的矩阵都有逆矩阵，只有方阵且行列式不为0的矩阵才有逆矩阵。
- 如果A有逆矩阵，则 $AA^{-1} = A^{-1}A = I$，其中I是单位矩阵。
- 如果A和B（相同大小）都有逆矩阵，则AB的逆矩阵是$B^{-1}A^{-1}$。
- 为什么使用numpy计算$A * A^{-1}$时没有恰好等于单位矩阵？因为计算机的浮点数计算存在误差，有时候我们计算只能得到一个近似值。

In [2]:
```
import numpy as np

A = np.array([[0, 2], [3, 0]])
# 计算矩阵的逆
A_inv = np.linalg.inv(A)
A_inv
```

Out[2]:
```
array([[0.        , 0.33333333],
       [0.5       , 0.        ]])
```

In [26]:
```
print(A @ A_inv)  # 结果是单位矩阵的近似值
```

```
[[1.0000000e+00 0.0000000e+00]
 [8.8817842e-16 1.0000000e+00]]
```

In [27]:
```
print(np.round(A @ A_inv)) # 使用round函数四舍五入得到单位矩阵
```

```
[[1. 0.]
 [0. 1.]]
```

验证AB的逆矩阵是$B^{-1}A^{-1}$。

In [28]:
```
import numpy as np

A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
AB = A @ B

# 计算矩阵的逆
A_inv = np.linalg.inv(A)
B_inv = np.linalg.inv(B)
AB_inv = np.linalg.inv(AB)
print(AB_inv)
```

```
[[ 12.5   -5.5 ]
 [-10.75   4.75]]
```

In [29]:
```
print(B_inv @ A_inv)
```

```
[[ 12.5   -5.5 ]
 [-10.75   4.75]]
```

练习3.6

- 创建一个2*2的矩阵A，计算A的逆矩阵。
- 创建一个对角矩阵，计算对角矩阵的逆矩阵。对角矩阵的逆矩阵有什么规律?
- AB的逆矩阵是$B^{-1}A^{-1}$，那么ABC的逆矩阵（假设这3个矩阵都可逆）应该是什么？使用 numpy 验证你的结论。

In [ ]:
```
# 在这里编写练习代码


```

矩阵的维度（dimension）与秩（rank）

- 使用 numpy 的 shape 方法可以获取矩阵的维度。
- rref（reduced row echelon form）简化行阶梯形矩阵
- 使用 numpy 的 linalg.matrix_rank 方法可以获取矩阵的秩。

In [9]:
```
import numpy as np

A = np.array([
    [1, 2, 1, 3],
    [2, 4, 0, 4],
    [3, 6, 3, 9]
])

# 打印矩阵的形状
print(A.shape)
```

```
(3, 4)
```

在线性代数中，在线性代数中，RREF（Reduced Row Echelon Form，简化行阶梯形矩阵）是矩阵的一种标准形式。RREF是通过一系列行变换将矩阵化简得到的，它具有以下特性：

1. 每个非零行的第一个元素是1（主元）。
2. 每个非零行的第一个元素所在的列，其他元素都是0。
3. 任何一行的1（主元）都在前一行的1（主元）的右边。
4. 每个零行都在矩阵的底部。

一个简单的RREF如下所示（其中a, b, c可以是任意数字）：

$$\begin{bmatrix} 1 & a & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

我们可以使用 sympy 库中的 rref 函数来获得一个矩阵的简化行阶梯形矩阵。

In [ ]:
```
# 安装sympy
%pip install sympy
```

In [11]:
```python
import sympy as sp

# 定义矩阵
matrix = sp.Matrix([
    [1, 2, 1, 3],
    [2, 4, 0, 4],
    [3, 6, 3, 9]
])

# 计算RREF
rref_matrix, pivot_columns = matrix.rref()

# 输出RREF
print("RREF of the matrix:")
rref_matrix
```

```
RREF of the matrix:
```

Out[11]:
$$\begin{bmatrix} 1 & 2 & 0 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

In [12]:
```python
# 主元列的索引，(0, 2)表示第1列和第3列是主元列
print("Pivot columns:", pivot_columns)
```

```
Pivot columns: (0, 2)
```

将矩阵转换为简化行阶梯形矩阵可以很容易的得到矩阵的秩（rank）。矩阵的秩是矩阵非常重要的性质，它可以告诉我们矩阵的包含的所有的向量的维度。我们也可以直接使用 numpy 的 linalg.matrix_rank 方法来获取矩阵的秩。

In [10]:
```python
import numpy as np

A = np.array([
    [1, 2, 1, 3],
    [2, 4, 0, 4],
    [3, 6, 3, 9]
])

print(np.linalg.matrix_rank(A))
```

```
2
```

练习3.7

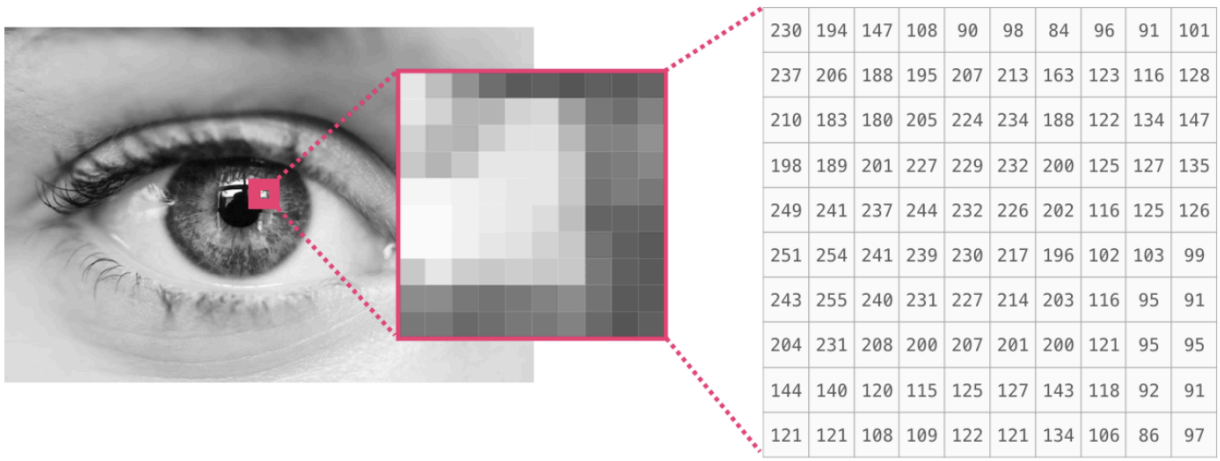
- 将一组（大于3个）在一条直线上的向量转换为矩阵，然后计算矩阵的秩。思考一下为什么会得到这个结果。
- 将一组（大于3个）在z=0平面上的3维向量转换为矩阵，然后计算矩阵的秩。思考一下为什么会得到这个结果。
- 如何构建一个4*3的矩阵，使得矩阵的秩为3。这个形状的矩阵的秩有可能等于4吗？为什么？
- 使用 numpy 编程验证上面这些问题的结果。

In [ ]:
```
# 在这里编写练习代码


```

矩阵的简单应用

矩阵的应用在图形学、机器学习、物理学、工程学可以说是无处不在。在这里介绍一些简单的例子：如何使用矩阵表示和处理位图。

下图显示了如何使用一个二维矩阵来表示一个简单的灰色位图。每一个位图都有一个分辨率，例如1024*768，表示这个位图有1024列和768行。矩阵中的每个元素存储的是对应位置的像素的灰度值。灰度值通常是一个0到255之间的整数，0表示黑色，255表示白色。



230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97

In [ ]:
```
# 安装用来处理图片的Python库pillow
%pip install pillow scipy
```

In [22]:
```
# 使用numpy处理灰色位图的代码

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# 根据文件路径打开文件
gray_image = Image.open('./img/gray-image.jpg').convert('L')

# 将图像转换为 NumPy 数组
image_array = np.array(gray_image)

# 打印图像数组的形状
print(image_array.shape)
```

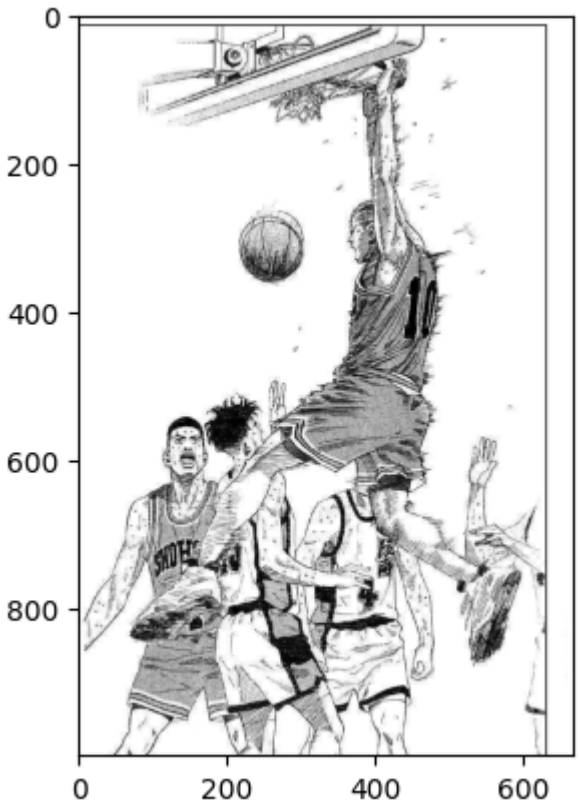
```
(999, 667)
```

In [23]:
```
# 打印矩阵包含的数值
print(image_array)
```

```
[[255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 ...
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]]
```

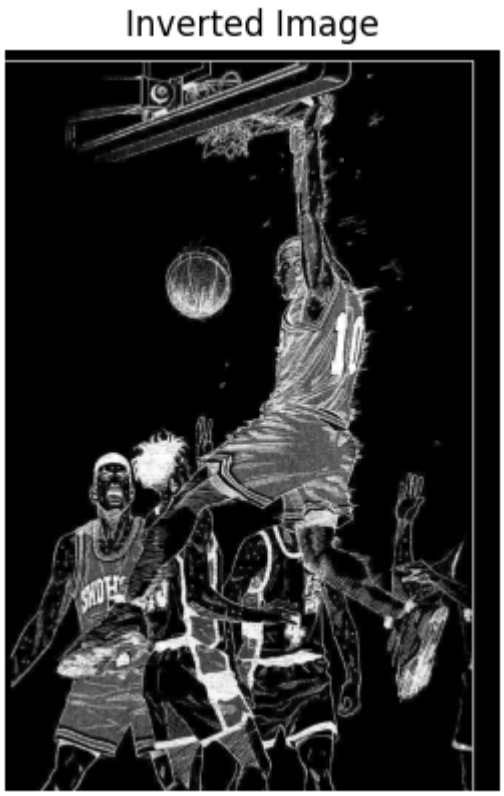
In [24]:
```
# 显示图像
plt.imshow(image_array, cmap='gray')
```

Out[24]: <matplotlib.image.AxesImage at 0x190b5405c10>

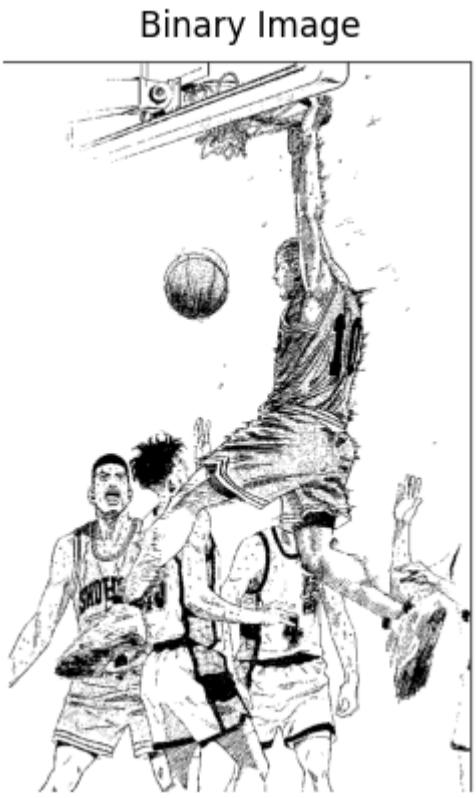


In [25]:

```
# 将每个像素值从255减去，得到反转图像
inverted_image = 255 - image_array

# 显示反转后的图像
plt.imshow(inverted_image, cmap='gray')
plt.title('Inverted Image')
plt.axis('off')
plt.show()
```



In [26]:
```
# 设置阈值
threshold = 128

# 将图像二值化，意味着将所有大于阈值的像素值设为255（白色），所有小于阈值的像素值设为0
binary_image = (image_array > threshold) * 255

# 显示二值化后的图像
plt.imshow(binary_image, cmap='gray')
plt.title('Binary Image')
plt.axis('off')
plt.show()
```



如何使用矩阵来表示彩色图片？彩色图片通常使用RGB（红绿蓝）三个通道来表示，每个通道都是用一个矩阵来表示，矩阵中的元素存储的分别是红绿蓝三种颜色的亮度值（从0到255）。因此，彩色图片可以看作是一个三维矩阵。

In [36]:

```python
# 使用numpy处理彩色位图的代码

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# 打开图像并将其转换为RGB模式
colored_image = Image.open('./img/colorful-image.jpg').convert('RGB')

# 将图像转换为 NumPy 数组
image_array2 = np.array(colored_image)

# 打印图像的分辨率
print(image_array2.shape)

# 显示原始图像
plt.imshow(image_array2)
plt.title('Original Image')
plt.axis('off')
plt.show()
```

```
(1040, 736, 3)
```

In [37]:

```python
# 提取单个颜色通道
red_channel = image_array2[:, :, 0]
green_channel = image_array2[:, :, 1]
blue_channel = image_array2[:, :, 2]

# 打印图像的分辨率
print(red_channel.shape)
print(green_channel.shape)
print(blue_channel.shape)

# 显示红色通道
plt.imshow(red_channel, cmap='Reds')
plt.title('Red Channel')
plt.axis('off')
plt.show()
```

```
(1040, 736)
(1040, 736)
(1040, 736)
```

In [30]:
```
# 显示绿色通道
plt.imshow(green_channel, cmap='Greens')
plt.title('Green Channel')
plt.axis('off')
plt.show()
```

In [31]:

```
# 显示蓝色通道
plt.imshow(blue_channel, cmap='Blues')
plt.title('Blue Channel')
plt.axis('off')
plt.show()
```

In [34]:

```
# 将三个颜色通道的矩阵合并还原成彩色图像
original_image = np.stack((red_channel, green_channel, blue_channel), axis=-1)
plt.imshow(original_image)
plt.title('Original Image')
plt.axis('off')
plt.show()
```

In [35]:

```
# 彩色图像反转
inverted_image = 255 - image_array2
plt.imshow(inverted_image)
plt.title('Inverted Image')
plt.axis('off')
plt.show()
```