1.4. Inner product

Inner product. The inner product of n-vector x and y is denoted as $x^T y$. In Python the inner product of `x` and `y` can be found using `np.inner(x,y)`

In [ ]:
```
import numpy as np
x = np.array([-1,2,2])
y = np.array([1,0,-3])
print(np.inner(x,y))
```

```
-7
```

Alternatively, you can use the `@` operator to perform inner product on numpy arrays.

In [ ]:
```
import numpy as np
x = np.array([-1,2,2])
y = np.array([1,0,-3])
x @ y
```

Out[ ]:
```
-7
```

Net present value. As an example, the following code snippet finds the net present value (NPV) of a cash flow vector `c`, with per-period interest rate `r`.

In [ ]:
```
import numpy as np
c = np.array([0.1,0.1,0.1,1.1]) #cash flow vector
n = len(c)
r = 0.05 #5% per-period interest rate
d = np.array([(1+r)**-i for i in range(n)])
NPV = c @ d
print(NPV)
```

```
1.236162401468524
```

In the fifth line, to get the vector `d` we raise the scalar `1+r` element-wise to the powers given in the range `range(n)` which expands to `0, 1,2, ..., n-1`, using list comprehension.

Total school-age population. Suppose that the 100-vector x gives the age distribution of some population, with x_i the number of people of age $i-1$, for $i = 1, \ldots, 100$. The total number of people with age between 5 and 18 (inclusive) is given by

$$x_6 + x_7 + \cdots + x_{18} + x_{19}$$

We can express this as $s^T x$ where s is the vector with entries one for $i = 6, \ldots, 19$ and zero otherwise. In Python, this is expressed as

In [ ]:
```
s = np.concatenate([np.zeros(5), np.ones(14), np.zeros(81)])
school_age_pop = s @ x
```

Several other expressions can be used to evaluate this quantity, for example, the expression `sum(x[5 : 19 ])`, using the Python function `sum`, which gives the sum of entries of vector.

1.5. Complexity of vector computations

Floating point operations. For any two numbers a and b, we have $(a+b)(a-b) = a^2-b^2$. When a computer calculates the left-hand and right-hand side, for specific numbers a and b, they need not be exactly the same, due to very small floating point round-off errors. But they should be very nearly the same. Let's see an example of this.

In [ ]:
```
import numpy as np
a = np.random.random()
b = np.random.random()
lhs = (a+b) * (a-b)
rhs = a**2 - b**2
print(lhs - rhs)
```

```
4.336808689942018e-19
```

Here we see that the left-hand and right-hand sides are not exactly equal, but very very close.

Complexity. You can time a Python command using the `time` package. The timer is not very accurate for very small times, say, measured in microseconds (10^{-6} seconds). You should run the command more than once; it can be a lot faster on the second or subsequent runs.

In [ ]:
```
import numpy as np
import time
a = np.random.random(10**5)
```