8. Linear equations

8.1. Linear and affine functions

Matrix-vector product function. Let's define an instance of the matrix-vector product function, and then numerically check that superposition holds.

In [ ]:
```
A = np.array([[-0.1,2.8,-1.6],[2.3,-0.6,-3.6]]) #2 by 3 matrix A
f = lambda x: A @ x
#Let's check superposition
x = np.array([1,2,3])
y = np.array([-3,-1,2])
alpha = 0.5
beta = -1.6
LHS = f(alpha*x + beta*y)
print('LHS:', LHS)
RHS = alpha*f(x) + beta*f(y)
print('RHS:', RHS)
print(np.linalg.norm(LHS - RHS))
```

```
LHS: [ 9.47 16.75]
RHS: [ 9.47 16.75]
1.7763568394002505e-15
```

In [ ]:
```
f(np.array([0,1,0])) #Should be second column of A
```

Out[ ]:
```
array([ 2.8, -0.6])
```

De-meaning matrix. Let's create a de-meaning matrix, and check that it works on a vector.

In [ ]:
```
de_mean = lambda n: np.identity(n) - (1/n)
x = np.array([0.2,2.3,1.0])
de_mean(len(x)) @ x #De-mean using matrix multiplication
```

Out[ ]:
```
array([-0.96666667,  1.13333333, -0.16666667])
```

```
In [ ]: x - sum(x)/len(x)
```

```
Out[ ]: array([-0.96666667,  1.13333333, -0.16666667])
```

Examples of functions that are not linear. The componentwise absolute value and the sort function are examples of nonlinear functions. These functions are easily computed by `abs` and `sorted`. By default, the `sorted` function sorts in increasing order, but this can be changed by adding an optional keyword argument.

```
In [ ]: f = lambda x: abs(x) #componentwise absolute value
        x = np.array([1,0])
        y = np.array([0,1])
        alpha = -1
        beta = 2
        f(alpha*x + beta*y)
```

```
Out[ ]: array([1, 2])
```

```
In [ ]: alpha*f(x) + beta*f(y)
```

```
Out[ ]: array([-1,  2])
```

```
In [ ]: f = lambda x: np.array(sorted(x, reverse = True))
        f(alpha*x + beta*y)
```

```
Out[ ]: array([ 2, -1])
```

```
In [ ]: alpha*f(x) + beta*f(y)
```

```
Out[ ]: array([1, 0])
```

8.2. Linear function models

Price elasticity of demand. Let's use a price elasticity of demand matrix to predict the demand for three products when the prices are changed a bit. Using this we can predict the change in total profit, given the manufacturing costs.

```
In [ ]: p = np.array([10, 20, 15]) #Current prices
        d = np.array([5.6, 1.5, 8.6]) #Current demand (say in thousands)
        c = np.array([6.5, 11.2, 9.8]) #Cost to manufacture
        profit = (p - c) @ d #Current total profit
```