2. Linear functions

2.1. Linear functions

Functions in Python. Python provides several methods for defining functions. One simple way is to use lambda functions. A simple function given by an expression such as $f(x) = x_1 + x_2 - x_4^2$ can be defined in a single line.

In [ ]:
```
f = lambda x: x[0] + x[1] - x[3]**2
f([-1,0,1,2])
```

Out[ ]:
```
-5
```

Since the function definition refers to the first, second and fourth elements of the argument `x`, these have to be well-defined when you call or evaluate `f(x)`; you will get an error if, for example, `x` has three elements or is a scalar.

Superposition. Suppose a is an n-vector. The function $f(x) = a^T x$ is linear, which means that for any n-vectors x and y, and any scalars α and β, the superposition equality

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

holds. Superposition says that evaluating f at a linear combination of two vectors is the same as forming the linear combination of f evaluated at the two vectors.

Let's define the inner product function f for a specific value of a, and then verify superposition in Python for specific values of x, y, α, and β. (This check does not show that the function is linear. It simply checks that superposition holds for these specific values.)

In [ ]:
```
a = np.array([-2,0,1,-3])
x = np.array([2,2,-1,1])
y = np.array([0,1,-1,0])
alpha = 1.5
```

```
beta = -3.7
LHS = np.inner(alpha*x + beta*y, a)
RHS = alpha*np.inner(x,a) + beta*np.inner(y,a)
print('LHS:', LHS)
print('RHS:', RHS)
```

```
LHS: -8.3
RHS: -8.3
```

For the function $f(x) = a^T x$, we have $f(e_3) = a_3$. Let's check that this holds in our example.

In [ ]:
```
a = np.array([-2,0,1,-3])
e3 = np.array([0,0,1,0])
print(e3 @ a)
```

```
1
```

Examples. Let's define the average function in Python and check its value of a specific vector. (Numpy also contains an average function, which can be called with `np.mean`.

In [ ]:
```
avg = lambda x: sum(x)/len(x)
x = [1,-3,2,-1]
avg(x)
```

Out[ ]:
```
-0.25
```

2.2. Taylor approximation

Taylor approximation. The (first-order) Taylor approximation of function $f : \mathbf{R}^n \to \mathbf{R}$, at the point z, is the affine function $\hat{f}(x)$ given by

$$\hat{f}(x) = f(z) + \nabla f(z)^T (x - z)$$

For x near z, $\hat{f}(x)$ is very close to $f(x)$. Let's try a numerical example (see page 36 of textbook) using Python.

In [ ]:
```
f = lambda x: x[0] + np.exp(x[1] - x[0])
grad_f = lambda z: np.array([1 - np.exp(z[1] - z[0]), np.exp(z[1]
↪   - z[0])])
z = np.array([1,2])
```