Linear Functions

Outline: Linear Functions

- Linear and affine functions
- Taylor approximation
- Regression model

Outline: Linear Functions

- **Linear and affine functions**
- Taylor approximation
- Regression model

Superposition and linear functions

Notation: The notation $f : \mathbb{R}^n \to \mathbb{R}$ means f is a function mapping n-vectors to numbers.

Definition: We say that f satistifies the superposition property if:

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

holds for all scalars α, β and all n-vectors x, y.

Definition: A function that satisfies superposition is called linear.

Example: the inner product function

Definition: For a an n-vector, the inner product function is defined as

$$f(x) = a^T x = a_0 x_0 + \ldots + a_{n-1} x_{n-1}.$$

We see that $f(x)$ is a weighted sum of the entries of x.

Exercise: Show that the inner production function is linear

Example: the inner product function

In Python:

In [6]:
```
import numpy as np

a = np.array([-2, 0, 1, -3])
f = lambda x: np.inner(a, x); f
```

Out[6]:
```
<function __main__.<lambda>(x)>
```

In [4]:

```
x, y = np.array([2, 2, -1, 1]), np.array([0, 1, -1, 0])
alpha, beta = 1.5, -3.7
```

In [3]:

```
lhs, rhs = f(alpha * x + beta * y), alpha * f(x) + beta * f(y)
lhs, rhs
```

Out[3]:

```
(-8.3, -8.3)
```

All linear functions are inner product functions

Proposition:

- If $f : \mathbb{R}^n \to \mathbb{R}$ is linear,
- Then f can be expressed as $f(x) = a^T x$ for some n-vector a. Specifically, a is given by $a_i = f(e_i)$ for i in $\{0, \ldots, n-1\}$.

Exercise: If $f(x) = a^T x$, show that $a_i = f(e_i)$.

In Python:

In [5]:

```
a = np.array([-2, 0, 1, -3])
f = lambda x: np.inner(a, x)

e0 = np.array([1, 0, 0, 0])
f(e0)
```

Out[5]:

```
-2
```

Affine functions

Definition: A function that is linear plus a constant is called affine. Its general form is:

$$f(x) = a^T x + b \quad \text{with } a \text{ an } n\text{-vector and } b \text{ a scalar}$$

Proposition: A function $f : \mathbb{R}^n \to \mathbb{R}$ is affine if and only if:

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

for all scalars α, β with $\alpha + \beta = 1$ and all n-vectors x, y.

Linear versus affine functions

- f is linear
- g is affine, not linear

Drawing

😱 Sometimes people refer to affine functions as linear.

Outline: Linear Functions

Gradient of a function

Definition: The gradient of f at the n-vector z is defined as the n-vector $\nabla f(z)$ as:

$$\nabla f(z) = \left(\frac{\partial f}{\partial x_1}(z), \ldots, \frac{\partial f}{\partial x_n}(z) \right).$$

Exercise: Consider $f(x) = x_1 + (x_2 - x_1)^3$ defined for any 2-vector x. Compute the gradient of f.

Gradient of a function

Example: Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$.

Drawing

The arrows represent the gradient of f at different points in $\mathbb{R}^2$.

Gradient of a function

In Python, we can use the package `jax` to compute gradients.

In [75]:
```
import jax

f = lambda x: x[0] + (x[1] - x[0]) ** 3
grad_f = jax.grad(f)

grad_f(np.array([1., 2.]))
```

Out[75]:
```
DeviceArray([-2.,  3.], dtype=float32)
```

Drawing

First-order Taylor approximation

Definition: Take $f : \mathbb{R}^n \to \mathbb{R}$. The first-order Taylor approximation of f near the n-vector z is

$$\hat{f}(x) = f(z) + \nabla f(z)^T(x - z) = f(z) + \frac{\partial f}{\partial x_1}(z)(x_1 - z_1) + \ldots + \frac{\partial f}{\partial x_n}(z)(x_n - z_n).$$

- $\hat{f}(z) = f(z)$
- $\hat{f}(x)$ is very close to $f(z)$ when x is very close to z.

First-order Taylor approximation

Drawing

- The first-order Taylor approximation provides an affine approximation *near* z to a function that is not affine.
- 😱 This is a good approximation of f only if x is near z.

First-order Taylor approximation

In Python:

In [37]:
```
f = lambda x: x[0] + (x[1] - x[0]) ** 3
grad_f = jax.grad(f)
```

In [38]:
```
z = np.array([1., 2.])
f_hat = lambda x: f(z) + np.inner(grad_f(z), (x - z))

f(z), f_hat(z)
```

Out[38]:
```
(2.0, 2.0)
```

In [41]:
```
close_to_z = np.array([1.01, 2.02])
f(close_to_z), f_hat(close_to_z)
```

Out[41]:
```
(2.0403010000000004, 2.04)
```

In [42]:
```
far_from_z = np.array([-1., -2.])
f(far_from_z), f_hat(far_from_z)
```

Out[42]:
```
(-2.0, -6.0)
```

Outline: Linear Functions

- Linear and affine functions
- Taylor approximation
- **Regression model**

Regression

Definition: A regression model is the affine function of x defined as

$$\hat{y} = x^T w + b$$

where:

- x is an n-vector, called a feature vector,
- the elements x_is of x are called regressors,
- the n-vector w is called the weight vector,
- the scalar b is called the offset or the intercept, and sometimes the bias,
- the scalar $\hat{y}$ is called the prediction -- of some actual outcome denoted y.

Example: House prices

Example: Model of house prices:

- y is selling price of house in $1000 (in some location, over some period)
- regressor is x = (house area, # bedrooms) (house area in 1000 sq.ft.)
- regression model weight vector and offset are:

$$w = (148.73, -18.85), b = 54.40$$

In Python, we can use this function to return predictions for each house based on beds and area.

In [76]:

```
w, b = np.array([148.73, -18.85]), 54.40
y_hat = lambda x: np.inner(x, w) + b

area, n_beds = 0.846, 1
price = y_hat(np.array([area, n_beds])); price
```

Out[76]:

```
161.37557999999999
```

Example: House prices

In Python, we evaluate how our model compares to real observed data stored in a csv file.

In [78]:

```
import pandas as pd

data = pd.read_csv("data/02_houses.csv")
```

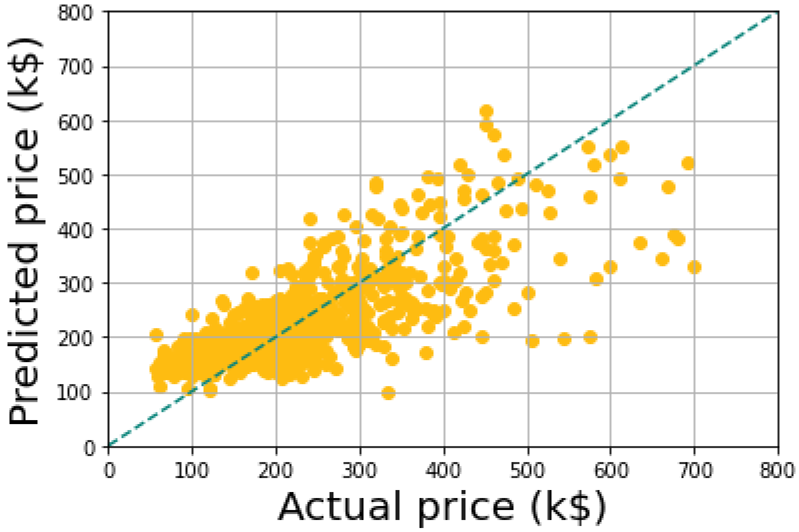
In [79]:

```
price = data["price"]; area = data["area"]; beds = data["beds"]
y_hat_values = np.array([w[0] * area + w[1] * beds]) + b
```

In [80]:

```
import matplotlib.pyplot as plt

plt.scatter(price, y_hat_values, color='#FEBC11')
plt.plot([0, 800],[0, 800], linestyle='dashed', color='#09847A')
plt.xlabel("Actual price (k$)", fontsize=20)
plt.xlim(0, 800)
plt.ylabel("Predicted price (k$)", fontsize=20)
plt.ylim(0, 800)
plt.grid(True)
```



Conclusion: Linear Functions

- Linear and affine functions
- Taylor approximation
- Regression model

Resources

- Ch. 2 of VMSL