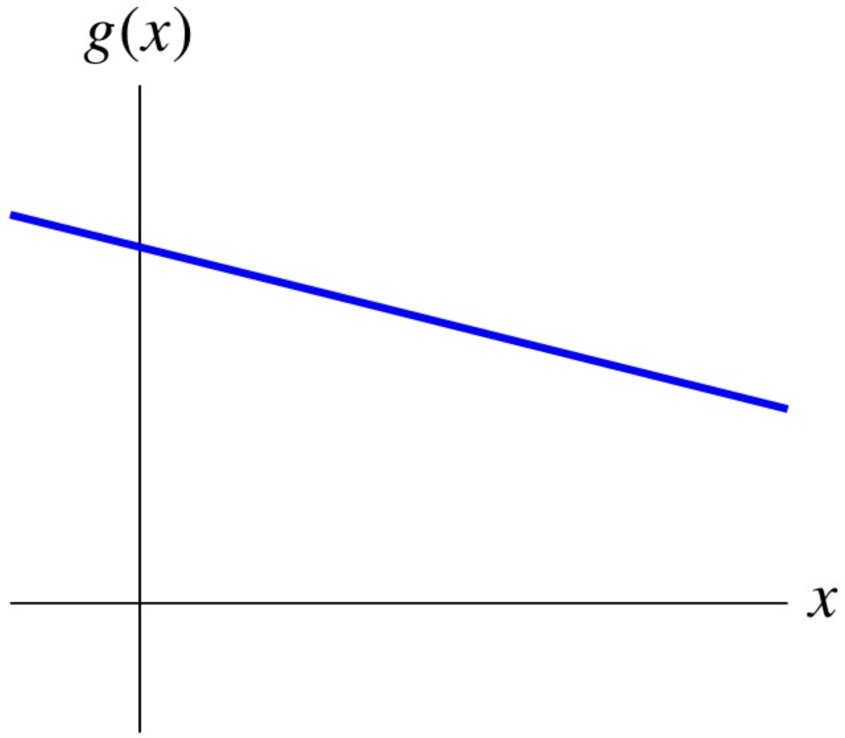
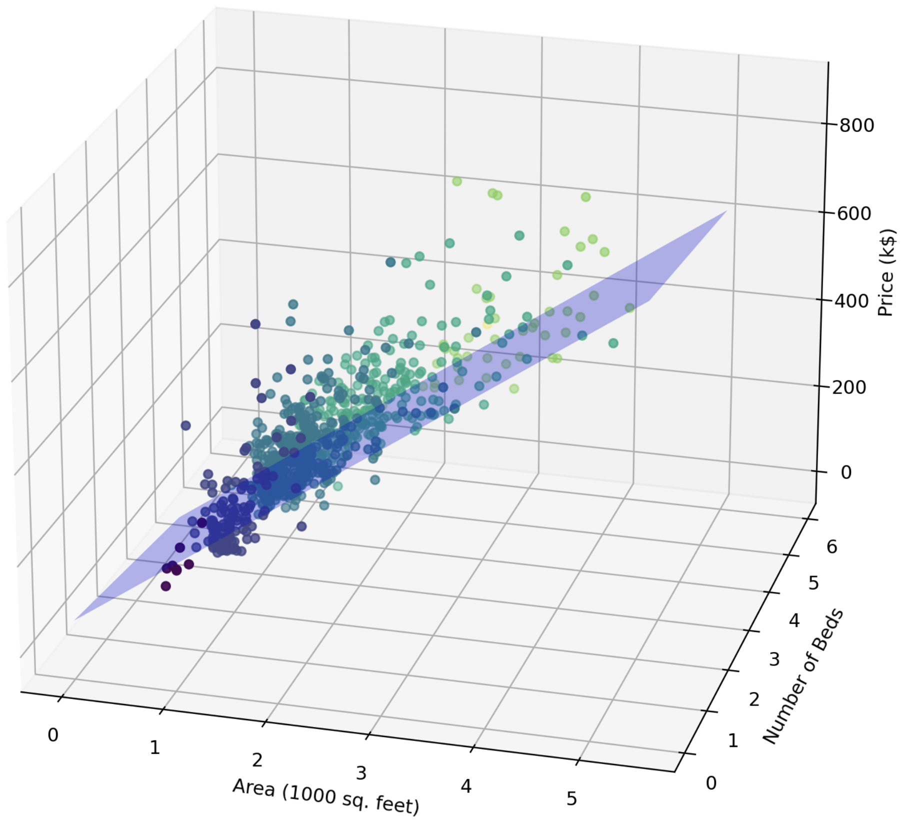
02 Linear Functions





Unit 1: Vectors, Book ILA Ch. 1-5

- 01 Vectors
- ***02 Linear Functions***
- 03 Norms and Distances
- 04 Clustering
- 05 Linear Independence

Unit 2: Matrices, Book ILA Ch. 6-11 + Book IMC Ch. 2

Unit 3: Least Squares, Book ILA Ch. 12-14 + Book IMC Ch. 8

Unit 4: Eigen-decomposition, Book IMC Ch. 10, 12, 19

Outline: 02 Linear Functions

- Linear and affine functions
- Taylor approximation
- Regression model

Outline: 02 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. The space $\mathbb{R}^n$ denotes the space of all possible n-vectors, and $\mathbb{R}$ the space of all possible numbers.

Definition: We say that the function $f : \mathbb{R}^n \to \mathbb{R}$ 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 a fixed n-vector, the inner product function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$f(x) = a^T x = a_1 x_1 + \ldots + a_n x_n.$$

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

Exercise: Show that the inner product function is linear.

In Python:

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

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

Out[2]:
```
<function __main__.<lambda>(x)>
```

In [3]:
```
x, y = np.array([2, 2, -1, 1]), np.array([0, 1, -1, 0])
alpha, beta = 1.5, -3.7
```

In [5]:
```
lhs, rhs = f(alpha * x + beta * y), alpha * f(x) + beta * f(y)
lhs, rhs
```

Out[5]:
```
(-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 $\{1, \dots, n\}$ where e_i is the one-hot vector with entry 1 at i.

Exercise: Consider that $f : \mathbb{R}^n \to \mathbb{R}$ is a function such that for all n-vectors x: $f(x) = a^T x$. Show that $a_i = f(e_i)$.

In Python:

In [6]:
```
a = np.array([-2, 0, 1, -3])
f = lambda x: np.inner(a, x)

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

Out[6]:
```
-2
```

Affine functions

Definition: A function $f : \mathbb{R}^n \to \mathbb{R}$ that is linear plus a constant is called affine. Its general form is:

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

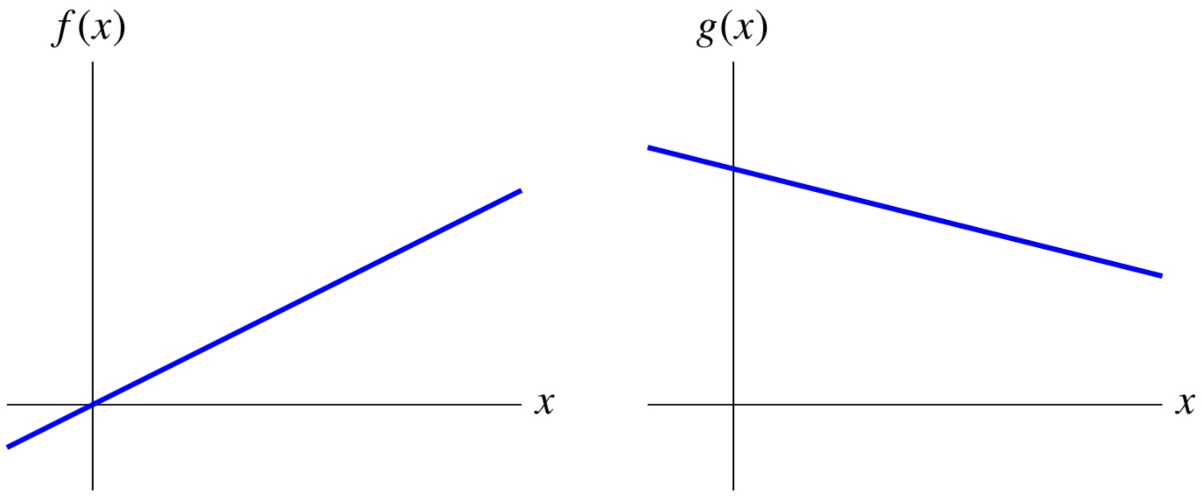
Proposition: A function $f : \mathbb{R}^n \to \mathbb{R}$ is affine if and only if:

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

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

Linear versus affine functions

In this plot, f is linear and g is affine, not linear



😱 Sometimes people refer to affine functions as linear.

Outline: Linear Functions

Gradient of a function

Definition: Consider a function $f : \mathbb{R}^n \to \mathbb{R}$. The gradient of f at the n-vector x is defined as the n-vector $\nabla f(x)$ as:

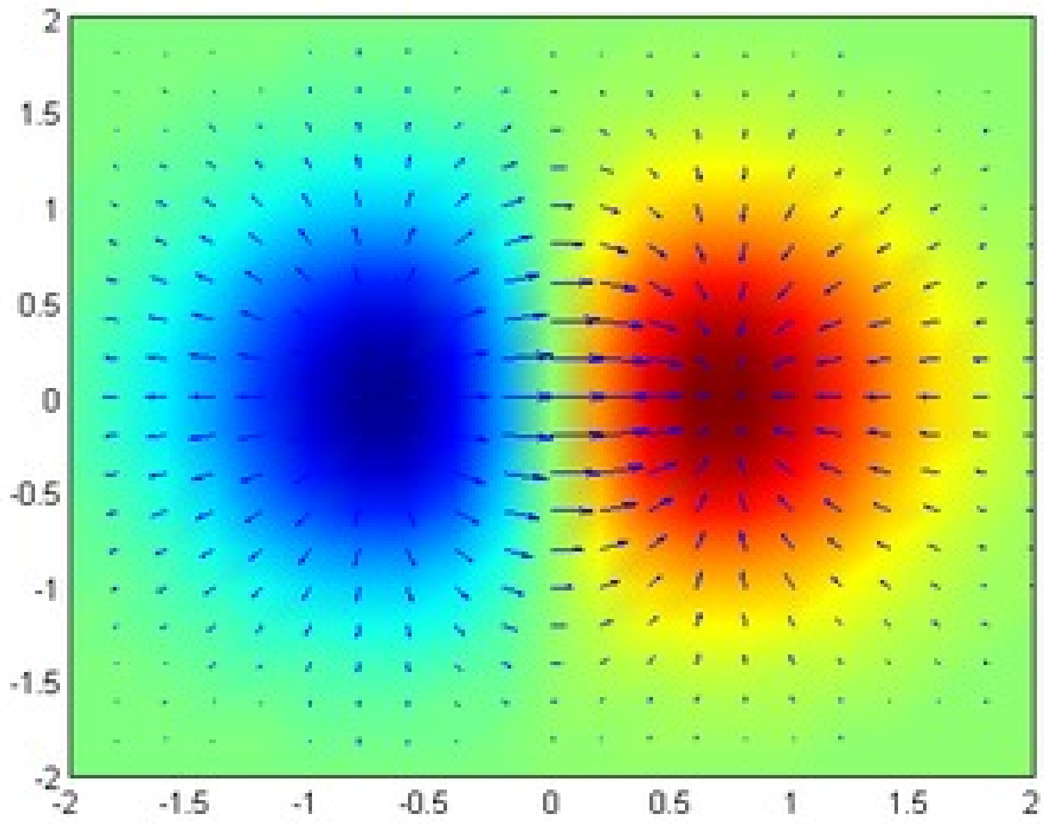
$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_n}(x) \right).$$

where $\frac{\partial}{\partial x_i}$ is the partial derivative of f with respect to the component i of x.

Exercise: Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined as: $f(x) = x_1 + (x_2 - x_1)^3$ for any 2-vector $x = (x_1, x_2)$. Compute the gradient of f.

Gradient of a function

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



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 [7]:
```
import jax

f = lambda x: x[0] + (x[1] - x[0]) ** 3
grad_f = jax.grad(f);
grad_f(np.array([1., 2.]))
```

```
WARNING:absl:No GPU/TPU found, falling back to CPU. (Set TF_CPP_MIN_LOG_LEVEL=0
and rerun for more info.)
```

Out[7]:
```
DeviceArray([-2.,  3.], dtype=float32)
```

Remark: There are other packages to automatically compute gradients:

- `autograd`, `pytorch`, `tensorflow`.

DeepMind Releases New JAX Libraries for Neural Networks and Reinforcement Learning

Synced Follow
Feb 21, 2020 · 3 min read

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

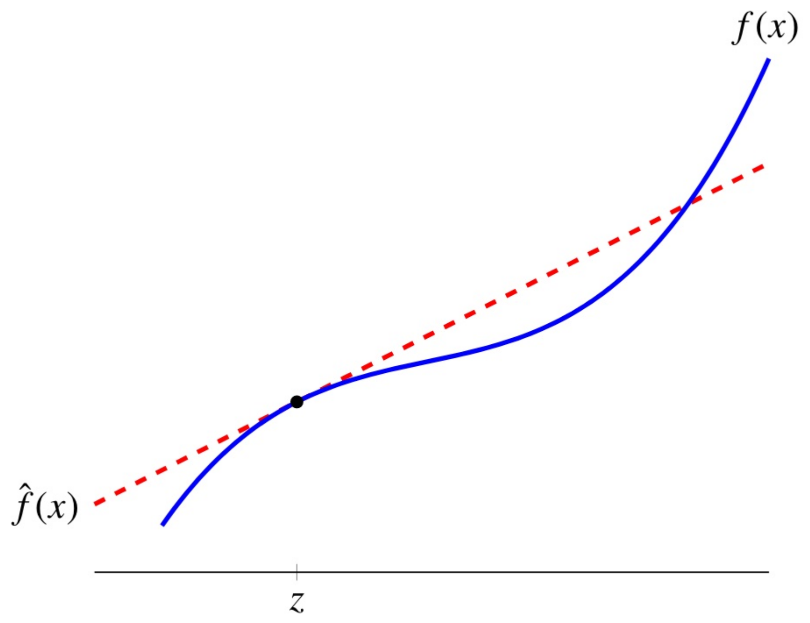
$$\hat{f}(x) = f(z) + \nabla f(z)^T (x - z) = f(z) + \frac{\partial f}{\partial x_1}(z)(x_1 - z_1) + \ldots + \frac{\partial f}{\partial x_n}(z)(x_n - z_n).$$

Properties:

- $\hat{f}(z) = f(z)$, i.e. equality at z.
- $\hat{f}(x)$ is very close to $f(z)$ when x is very close to z.

First-order Taylor approximation

The following graph shows a function $f : \mathbb{R} \to \mathbb{R}$.



- 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.

Outline: Linear Functions

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

Regression

Definition: A linear 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* or *input* vector,
- the elements x_i's of x are called *regressors*,
- the n-vector w is called *weight vector*,
- the scalar b is called *offset* or *intercept* or *bias*,
- the scalar $\hat{y}$ is called *prediction* --- of some actual *outcome* y.

Example: House prices

Example: Model of house prices:

- Outcome of interest is y, selling price of house in \$1000, in some location, over some period of time.
- Available regressors are x = (house area, # bedrooms), where house area is in 1000 sq.ft.
- We are given a regression model with weight vector and offset as:

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

If we are given input regressors for different houses, we can use the regression model to make predictions on their prices.

We use the regression function to return predictions for each house based on beds and area.

In [14]:
```
w, b = np.array([148.73, -18.85]), 54.40
y_hat = lambda x: np.inner(x, w) + b

x_area, x_beds = 0.846, 1
y_hat_price = y_hat(np.array([x_area, x_beds])); y_hat_price
```

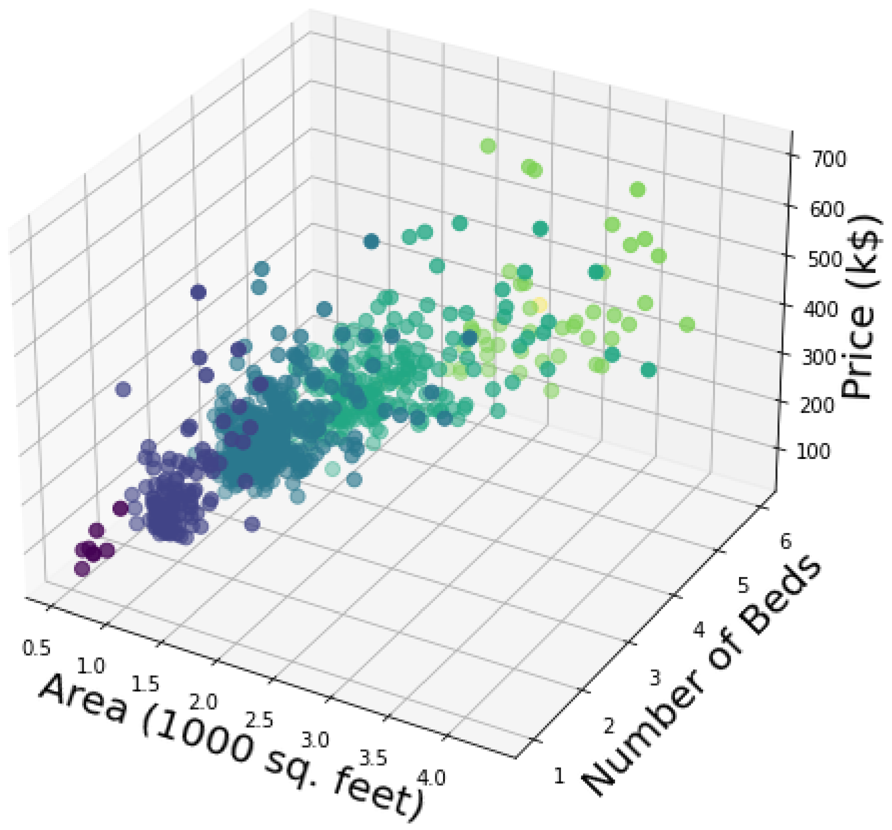
Out[14]:
```
161.37557999999999
```

In Python, we evaluate how our model compares to observed data stored in a csv. We use the package `pandas` to load data from a `.csv`.

In [15]:
```
import pandas as pd
import numpy as np
data = pd.read_csv("data/02_houses.csv")
```

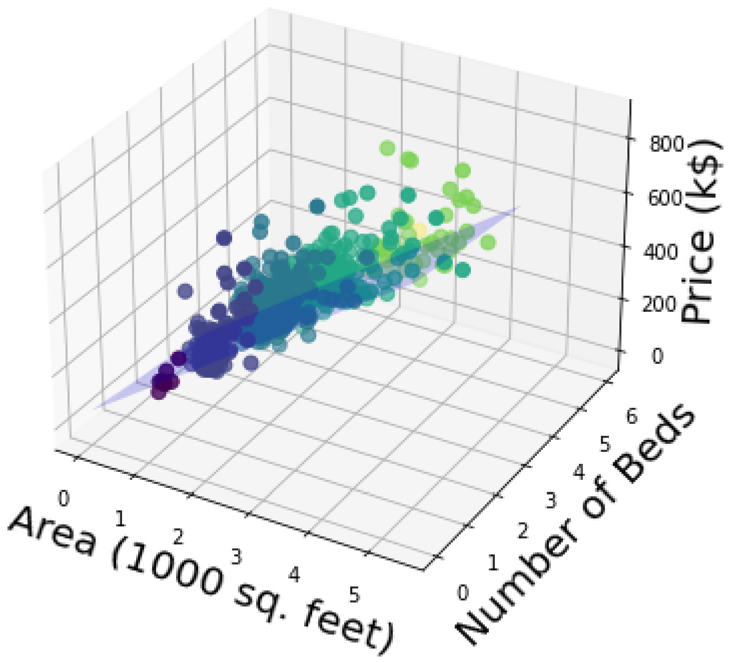
In [16]:
```
y_price = data["price"];
x_area = data["area"]; x_beds = data["beds"]
y_hat_price = np.array([w[0] * x_area + w[1] * x_beds]) + b
```

In [14]:
```
import matplotlib.pyplot as plt
fg = plt.figure(figsize=(8, 8)); ax = fg.add_subplot(projection='3d')
ax.scatter(x_area, x_beds, y_price, c=x_beds, s=50)
ax.set_xlabel("Area (1000 sq. feet)", fontsize=20)
ax.set_ylabel("Number of Beds", fontsize=20)
ax.set_zlabel("Price (k$)", fontsize=20);
```



In [17]:
```
fg = plt.figure(figsize=(6, 6)); ax = fg.add_subplot(projection='3d')
ax.scatter(x_area, x_beds, y_price, c=x_beds, s=50)
ax.set_xlabel("Area (1000 sq. feet)", fontsize=20); ax.set_ylabel("Number of Bed

x_beds_grid = np.arange(0, 6, 0.5); x_area_grid = np.arange(0, 4, 0.5)
X, Y = np.meshgrid(x_beds_grid, x_area_grid)
zs = np.array(y_hat(np.array([np.ravel(X), np.ravel(Y)]).T));
Z = zs.reshape(X.shape)
ax.plot_surface(X, Y, Z, alpha=0.2, color="blue"); plt.show();
```



Conclusion: Linear Functions

- Linear and affine functions
- Taylor approximation
- Regression model

Resources

- Ch. 2 of ILA