DM561 / DM562

Linear Algebra with Applications

Multiple Linear Regression

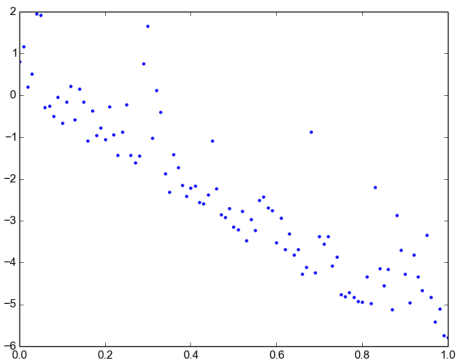
Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Curve fitting - Linear Regression

- In $\mathbb{R}^2$ we are given m points (pairs of numbers) $(x_1, y_1), \ldots, (x_m, y_m)$
  (For example, they can be temperature in the atmosphere taken at different days of the year or the cost of houses given their square meters.)



- We want to determine a function $f(x)$ such that

$$f(x_1) \approx y_1, \cdots, f(x_m) \approx y_m.$$

- The type of function (polynomial, exponential, sine and cosine, ...) may be suggested by the nature of the problem (the underlying physical law)
  In many cases a polynomial of a certain degree will be appropriate.

In Python

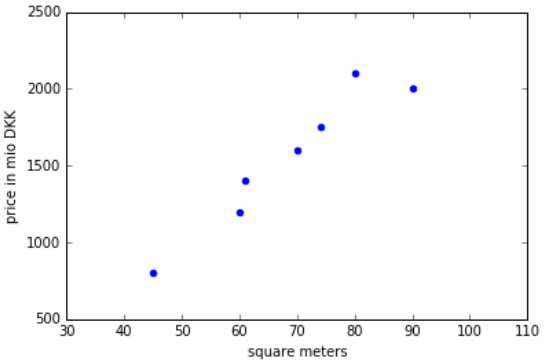

```
#!/usr/bin/python
import numpy as np
import matplotlib.pyplot as plt

np.set_printoptions(precision=3)

m=101
x = np.linspace(0, 1, m)
y = x**3-7*x+np.random.exponential(1,m)  # the unknown target function

plt.plot(x, y, '.')
plt.xlabel('x');  plt.ylabel('y')

plt.show()
```

The script produces two arrays **x** and **y**.
The ith elements of these arrays give us the ith point, (x_i, y_i).

House Price Example

Size in m^2	Price in mio DKK
45	800
60	1200
61	1400
70	1600
74	1750
80	2100
90	2000

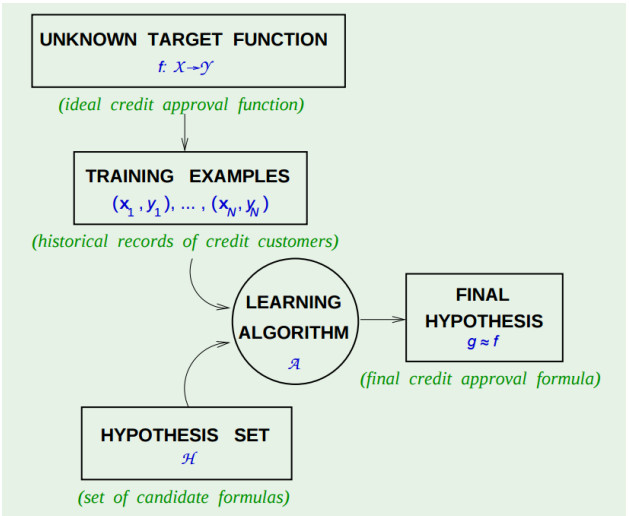
$$\begin{bmatrix} (x_1, y_1) \\ (x_2, y_2) \\ \vdots \\ \vdots \\ (x_m, y_m) \end{bmatrix} \rightsquigarrow \begin{bmatrix} (45, 800) \\ (60, 1200) \\ (61, 1400) \\ (70, 1600) \\ (74, 1750) \\ (80, 2100) \\ (90, 2000) \end{bmatrix}$$



$$f(x) = -489.76 + 29.75x$$

x	$\hat{y}$	y
45	848.83	800
60	1295.03	1200
61	1324.78	1400
70	1592.5	1600

Machine Learning Model



Outline

Linear Regression

The first case we consider is fitting a straight line $y = a + bx$.

- Given the points $(x_1, y_1), \ldots, (x_m, y_m)$ we search for the line through them such that the sum of the squares of the distances of those points from the straight line is minimum, where the distance from (x_i, y_i) is measured in the vertical direction (the y-direction).
- More formally, each point i with abscissa x_i has the ordinate $a + bx_i$ in the fitted line. The distance from the actual data (x_i, y_i) is $|y_i - a - bx_i|$ and the sum of squares of errors is

$$q = \sum_{i=1}^{m} (y_i - a - bx_i)^2$$

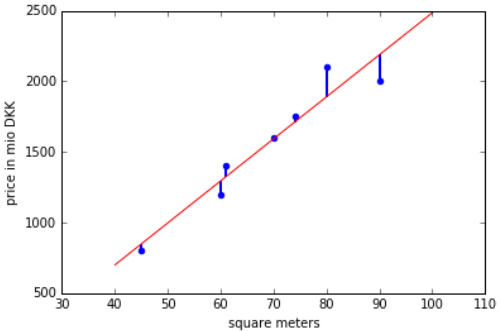
- Hence, q depends on a and b, while the values x_i and y_i are given by the data available. From calculus we know that the minimum of a function occurs where the partial derivatives are zero.

House Price Example

$$\begin{bmatrix} (x_1, y_1) \\ (x_2, y_2) \\ \vdots \\ \vdots \\ (x_m, y_m) \end{bmatrix} \rightsquigarrow \begin{bmatrix} (45, 800) \\ (60, 1200) \\ (61, 1400) \\ (70, 1600) \\ (74, 1750) \\ (80, 2100) \\ (90, 2000) \end{bmatrix}$$

$f(x) = -489.76 + 29.75x$

x	$\hat{y}$	y
45	848.83	800
60	1295.03	1200
61	1324.78	1400
70	1592.5	1600
74	1711.48	1750
80	1889.96	2100
90	2187.43	2000



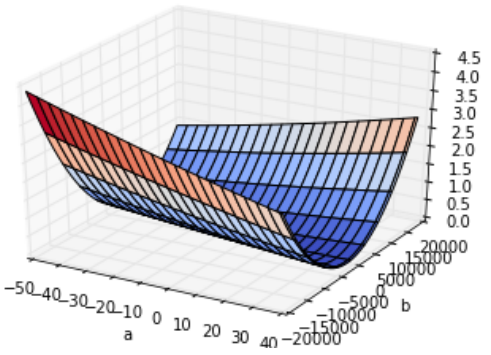
$$\begin{aligned} \hat{q} &= \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \\ &= (800 - 848.83)^2 \\ &\quad + (1200 - 1295.03)^2 \\ &\quad + (1400 - 1324.78)^2 \\ &\quad + (1600 - 1592.5)^2 \\ &\quad + (1750 - 1711.48)^2 \\ &\quad + (2100 - 1889.96)^2 \\ &\quad + (2000 - 2187.43)^2 = 97858.86 \end{aligned}$$

House Price Example

For

$$f(x) = b + ax$$

$$\begin{aligned}
\hat{q}(a,b) &= \sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \\
&= (800 - b - 45 \cdot a)^2 \\
&\quad +(1200 - b - 60 \cdot a)^2 \\
&\quad +(1400 - b - 61 \cdot a)^2 \\
&\quad +(1600 - b - 70 \cdot a)^2 \\
&\quad +(1750 - b - 74 \cdot a)^2 \\
&\quad +(2100 - b - 80 \cdot a)^2 \\
&\quad +(2000 - b - 90 \cdot a)^2
\end{aligned}$$



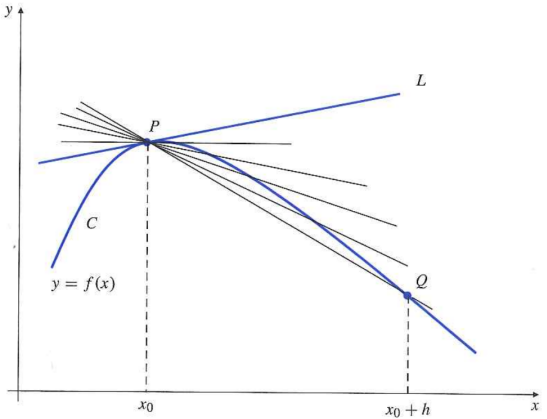
Differentiation

A line L through a point $(x_0, f(x_0))$ of f can be described by:

$$y = m(x - x_0) + f(x_0)$$

The derivative is the slope of the line that is tangent to the curve:

$$y = f'(x_0)(x - x_0) + f(x_0)$$



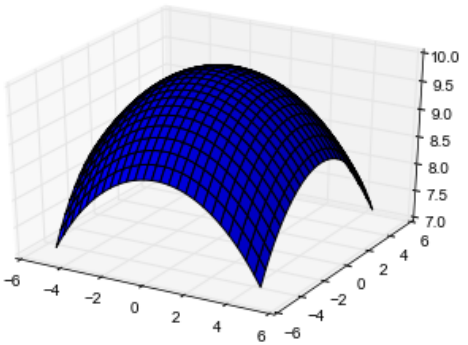
Functions of Several Variables

- A function f of n real variables is a rule that assigns a unique real number $f(x_1, x_2, \ldots, x_n)$ to each point $(x_1, x_2, \ldots, x_n)$

Example in $\mathbb{R}^2$:

$$f(x, y) = \sqrt{10^2 - x^2 - y^2}$$

$$x^2 + y^2 + z^2 = 10$$



Partial Derivatives

- The first partial derivative of the function $f(x, y)$ with respect to the variables x and y are:

$$f_1(x, y) = \lim_{h \to 0} \frac{f(x + h, y) - f(x, y)}{h} = \frac{\partial}{\partial x} f(x, y)$$

$$f_2(x, y) = \lim_{k \to 0} \frac{f(x, y + k) - f(x, y)}{k} = \frac{\partial}{\partial y} f(x, y)$$

- Their value in a point (x_0, y_0) is given by:

$$f_1(x_0, y_0) = \left(\frac{\partial}{\partial x} f(x, y) \right) \Bigg|_{(x_0, y_0)}$$

$$f_2(x_0, y_0) = \left(\frac{\partial}{\partial y} f(x, y) \right) \Bigg|_{(x_0, y_0)}$$

Solution

Remembering that a and b are our variables in the fitting problem, a necessary condition for q to be minimum is

$$\frac{\partial q}{\partial a} = -2\sum_{i=1}^{m}(y_i - a - bx_i) = 0$$
$$\frac{\partial q}{\partial b} = -2\sum_{i=1}^{m}x_i(y_i - a - bx_i) = 0$$

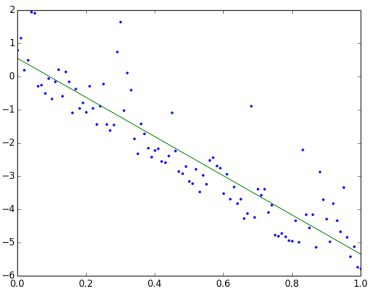
We can rewrite as:

$$\begin{cases} am + b\sum x_i = \sum y_i \\ a\sum x_i + b\sum x_i^2 = \sum x_i y_i \end{cases}$$

How to we solve this?
It is a System of Linear Equations!

In Python

```
a_11 = len(x)
a_12 = sum(x)
a_22 = sum(x**2)
b_1 = sum(y)
b_2 = sum(x*y)

A = np.array([[a_11, a_12],[a_12,a_22]])
b = np.array([b_1, b_2])

coeff = np.linalg.solve(A,b)

plt.plot(x, y, '.')
plt.plot([0,1],[coeff[0], coeff[0] + coeff[1]*1])
plt.show()
```



Fitting a polynomial curve

- We can generalize the polynomial $y = b_0 + b_1 x$ to a polynomial of degree k

  $$p(x) = b_0 + b_1 x + \cdots + b_k x^k$$

  where $k \leq m - 1$. Then q takes the form

  $$q = \sum_{i=1}^{m} (y_i - p(x_i))^2$$

  and depends on $k + 1$ parameters $b_0, \cdots, b_k$.

- As before the setting of the parameters that yields the minimum q can be found via partial derivatives.

Solution

The necessary condition for q to be minimum gives a $k+1$ system of linear equations:

$$\frac{\partial q}{\partial b_0} = 0$$
$$\frac{\partial q}{\partial b_1} = 0$$
$$\vdots$$
$$\frac{\partial q}{\partial b_k} = 0$$

For $k = 3$, the system we obtain is (summations are all from 1 to m):

$$\begin{cases} b_0 m + b_1 \sum x_i + b_2 \sum x_i^2 + b_3 \sum x_i^3 = \sum y_i \\ b_0 \sum x_i + b_1 \sum x_i^2 + b_2 \sum x_i^3 + b_3 \sum x_i^4 = \sum x_i y_i \\ b_0 \sum x_i^2 + b_1 \sum x_i^3 + b_2 \sum x_i^4 + b_3 \sum x_i^5 = \sum x_i^2 y_i \\ b_0 \sum x_i^3 + b_1 \sum x_i^4 + b_2 \sum x_i^5 + b_3 \sum x_i^6 = \sum x_i^3 y_i \end{cases}$$

How do we solve this?

In Python

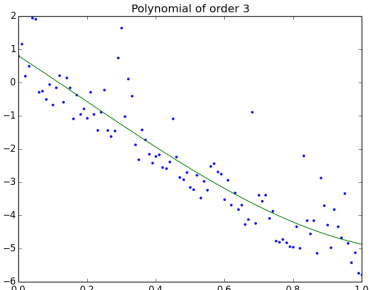
```
a_0 = len(x)
a_1 = sum(x)
a_2 = sum(x**2)
a_3 = sum(x**3)
b_0 = sum(y)
a_4 = sum(x**4)
b_1 = sum(x*y)
a_5 = sum(x**5)
b_2 = sum((x**2)*y)
a_6 = sum(x**6)
b_3 = sum((x**3)*y)
```

```
A = np.array([[a_0, a_1, a_2,a_3],
              [a_1,a_2,a_3,a_4],
              [a_2,a_3,a_4,a_5],
              [a_3,a_4,a_5,a_6]
])
b = np.array([b_0,b_1,b_2,b_3])

coeff = np.linalg.solve(A,b)
```

In Python

```
# Create a callable object for the polynomial f(x) = (x-1)(x-2) = x^2 - 3x + 2.
>>> print(f = np.poly1d([1, -3, 2]))
   2
1 x - 3 x + 2
# Evaluate f(x) for several values of x in a single function call.
>>> f([1, 2, 3, 4])
array([0, 0, 2, 6])
```

```
P=np.poly1d(coeff[::-1])
print(P)
#         3         2
#-1.242 x + 3.445 x - 8.828 x + 1.333

plt.plot(x, y, '.')
xx = np.linspace(0.0, 1.0, 50)
plt.plot(xx, P(xx), '-')
plt.title('Polynomial of order 3');
plt.show()
```



Outline

Using Linear Algebra

- We generalize the method to the space $\mathbb{R}^n$.
- several inputs that influence outputs
  inputs $(x_1, \ldots, x_n)$: independent variables, predictors, features
  outputs y: dependent variables, responses
- Examples:
  The price of a house may depend on the square meters but also on the distance from city center and the year of construction.
  The temperature on the ground may depend, beside on the day of the year, also on the latitude and maybe, in the case of global warming, on the development through years.

Using Linear Algebra

- We assume that the variable y is related to $\mathbf{x} \in \mathbb{R}^n$ linearly, so for some constants b_0 and $\mathbf{b}$:

$$y = b_0 + \mathbf{b}^T \mathbf{x}$$

- Given the set of m points $(y_1, \mathbf{x}_1,), \ldots, (y_m, \mathbf{x}_m)$ in the ideal case, we have that $y_i = b_0 + \mathbf{b}\mathbf{x}_i$, for all $i = 1, \ldots, m$. In matrix form:

$$\begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,k} \\ 1 & x_{2,1} & \cdots & x_{2,k} \\ \vdots & \vdots & & \\ 1 & x_{m,1} & \cdots & x_{m,k} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

This can be written as $A\mathbf{z} = \mathbf{y}$ to emphasize that $\mathbf{z}$ are our unknowns and A and $\mathbf{y}$ are given.

- In general we cannot expect to find a vector $\mathbf{z} \in \mathbb{R}^{n+1}$ for which $A\mathbf{z}$ equals $\mathbf{y}$. Why?
- Instead we can look for a vector $\mathbf{z}$ for which $A\mathbf{z}$ is closest to $\mathbf{y}$.
- For each $\mathbf{z} \in \mathbb{R}^{n+1}$ we can form the residual

$$r(\mathbf{z}) = \mathbf{y} - \mathbf{A}\mathbf{z}$$

Recalling the definition of the norm or length of a vector, the distance between $\mathbf{y}$ and $A\mathbf{z}$ is given by

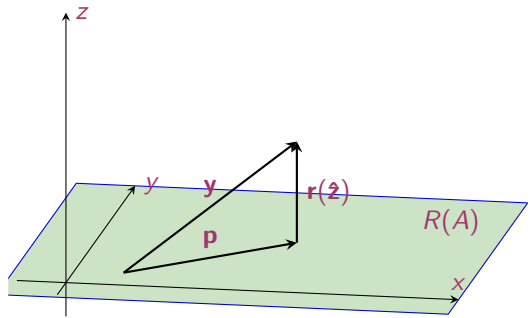
$$\| r(\mathbf{z}) \| = \| \mathbf{y} - A\mathbf{z} \|$$

- Minimizing $\| r(\mathbf{z}) \|$ is equivalent to minimize $\| r(\mathbf{z}) \|^2$. So what we need is for $A\mathbf{z}$ to be the closest point of the form $A\hat{\mathbf{z}}$ to $\mathbf{y}$. That is, $A\mathbf{z}$ has to be the closest point in $R(A)$ to $\mathbf{y}$.
- Hence, a vector $\hat{\mathbf{z}}$ will be the one that minimizes $\| \mathbf{y} - A\mathbf{z} \|^2$ if and only if $\mathbf{p} = A\hat{\mathbf{z}}$ is the vector in $R(A)$ that is closest to $\mathbf{y}$.
- The vector $\mathbf{p}$ is said to be the projection of $\mathbf{y}$ onto $R(A)$.

- It follows that

$$r(\hat{\mathbf{z}}) \in R(A)^{\perp}$$

that is, $r(\hat{\mathbf{z}})$ lays on the subspace of $\mathbb{R}^m$ that is the orthogonal complement of $R(A)$.
(**Def.** If W is a subspace of $\mathbb{R}^n$, then the set of all vectors in $\mathbb{R}^n$ that are orthogonal to every vector in W is called the orthogonal complement of W and is denoted by the symbol $W^{\perp}$.)



$\mathbf{y} \in \mathbb{R}^3$ and A is a 3×2 matrix of rank 2.

- To proceed we need another fact. For an $m \times (n+1)$ matrix A,

$$R(A)^{\perp} = N(A^T),$$

that is, the orthogonal complement of the range of a matrix A is the null space of the transpose of the matrix A.

Proof:

- A vector $\mathbf{w}$ that belongs to $R(A)$ belongs to the linear span of A, ie, it is a linear combination of the columns of the matrix A.
- A vector $\mathbf{v}$ that belongs to $R(A)^{\perp}$, must therefore be orthogonal to each column of the matrix A.
- Consequently $A^T\mathbf{v} = \mathbf{0}$. Thus, $\mathbf{v}$ must be an element of $N(A^T)$ and hence $N(A^T) = R(A)^{\perp}$.

- Thus we have

$$r(\hat{\mathbf{z}}) \in N(A^T)$$

This leads to a $(k+1) \times (k+1)$ system of linear equations in the $\mathbf{z}$ variables. The systems is actually the same as the one derived earlier via calculus.

Proof: Since $r(\hat{\mathbf{z}}) = \mathbf{y} - A\mathbf{z}$ we have

$$A^T(\mathbf{y} - A\mathbf{z}) = \mathbf{0}$$

Thus to solve the least square system of $A\mathbf{z} = \mathbf{y}$ we have to solve the normal equations:

$$A^T\mathbf{y} = A^TA\mathbf{z}$$

Remembering that we are solving in the $\mathbf{z}$ variables we have a system of size $(k+1) \times (k+1)$ because A is of size $m \times (k+1)$ and therefore A^TA is of size $(k+1) \times (k+1)$.

- If A is an $m \times (k+1)$ matrix of rank $(k+1)$, then the system at the previous point has a unique solution:

$$\hat{\mathbf{z}} = (A^T A)^{-1} A^T \mathbf{y}$$

(Hint: assume that $A^T A\mathbf{u} = \mathbf{0}$ has only the trivial solution $\mathbf{u} = \mathbf{0}$.)

Under the assumption that $A^T A\mathbf{u} = \mathbf{0}$ has only the trivial solution $\mathbf{u} = \mathbf{0}$ then the matrix $A^T A$ is non-singular and hence invertible and the solution unique.

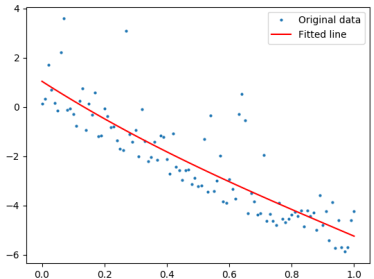
In Python

In polynomial regression, the $m \times (k+1)$ matrix A is called a Vandermonde matrix (a matrix with entries $a_{ij} = x_i^{k-j+1}$, $j = 1..k+1$).
NumPy's `np.vander()` is a convenient tool for quickly constructing a Vandermonde matrix, given the values $x_i, i = 1..m$, and the number of desired columns $(k+1)$.

```
>>> print(np.vander([2, 3, 5], 2))
[[2 1]                              # [[2**1, 2**0]
 [3 1]                              #  [3**1, 3**0]
 [5 1]]                             #  [5**1, 5**0]]

>>> print(np.vander([2, 3, 5, 4], 3))
[[ 4  2  1]                         # [[2**2, 2**1, 2**0]
 [ 9  3  1]                         #  [3**2, 3**1, 3**0]
 [25  5  1]                         #  [5**2, 5**1, 5**0]
 [16  4  1]]                        #  [4**2, 4**1, 4**0]
```

In Python

```
A = np.vander(x,4)

coeff = np.linalg.solve(A,y) ## Error!! Why?

B = A.T @ A
z = np.linalg.inv(B) @ A.T @ y

coeff = np.linalg.lstsq(A, y)[0]
np.allclose(z,coeff)

f=np.poly1d(coeff)
plt.plot(x, y, 'o', label='Original data', ↩
    markersize=2)
plt.plot(x, f(x), 'r', label='Fitted line')
plt.legend()
plt.show()
```



⇝ However, we still need to invert a matrix :(

How to find an ONB?

V inner product space. To find an ONB of V one applies the following **Gram-Schmidt process**.

Pick a basis $x_1, \ldots, x_k$ of the vector space V. Then construct inductively vectors $v_1, \ldots, v_k$, as follows:

$$v_1 := \frac{1}{||x_1||} x_1$$

$$v_2 := \frac{1}{||x_2 - \langle x_2, v_1 \rangle v_1||} (x_2 - \langle x_2, v_1 \rangle v_1)$$

$$\vdots$$

$$v_{j+1} := \frac{1}{||x_{j+1} - \sum_{i=1}^{j} \langle x_{j+1}, v_i \rangle v_i||} \left(x_{j+1} - \sum_{i=1}^{j} \langle x_{j+1}, v_i \rangle v_i \right)$$

$$\vdots$$

QR Decomposition

Theorem Every nonzero finite-dimensional inner product space has an orthonormal basis.

Gram-Schmidt Orthogonalization To convert a basis $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r\}$ into an orthonormal basis $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_r\}$, perform the following computations:

Step 1. $\mathbf{v}_1 = \mathbf{u}_1$

Step 2. $\mathbf{v}_2 = \mathbf{u}_2 - \frac{\langle \mathbf{u}_2, \mathbf{v}_1 \rangle}{\|\mathbf{v}_1\|^2} \mathbf{v}_1$

Step 3. $\mathbf{v}_3 = \mathbf{u}_3 - \frac{\langle \mathbf{u}_3, \mathbf{v}_1 \rangle}{\|\mathbf{v}_1\|^2} \mathbf{v}_1 - \frac{\langle \mathbf{u}_3, \mathbf{v}_2 \rangle}{\|\mathbf{v}_2\|^2} \mathbf{v}_2$

Step 4. $\mathbf{v}_4 = \mathbf{u}_4 - \frac{\langle \mathbf{u}_4, \mathbf{v}_1 \rangle}{\|\mathbf{v}_1\|^2} \mathbf{v}_1 - \frac{\langle \mathbf{u}_4, \mathbf{v}_2 \rangle}{\|\mathbf{v}_2\|^2} \mathbf{v}_2 - \frac{\langle \mathbf{u}_4, \mathbf{v}_3 \rangle}{\|\mathbf{v}_3\|^2} \mathbf{v}_3$

$\vdots$

r steps

Optional Step: To convert the orthogonal basis into an orthonormal basis $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_r\}$, normalize the orthogonal basis vectors.

QR Decomposition

- If A is an $m \times n$ matrix with linearly independent column vectors, and if Q is the matrix that results by applying the Gram-Schmidt process to the column vectors of A, what relationship, if any, exists between A and Q?

- A and Q can be written in partitioned forms as:

$$A = \begin{bmatrix} \mathbf{u}_1 & | & \mathbf{u}_2 & | & \dots & | & \mathbf{u}_n \end{bmatrix} \qquad \text{and} \qquad Q = \begin{bmatrix} \mathbf{q}_1 & | & \mathbf{q}_2 & | & \dots & | & \mathbf{q}_n \end{bmatrix}$$

- $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_r$ are expressible in terms of the vectors $\{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n\}$ as:

$$\begin{aligned} \mathbf{u}_1 &= \langle \mathbf{u}_1, \mathbf{q}_1 \rangle \mathbf{q}_1 + \langle \mathbf{u}_1, \mathbf{q}_2 \rangle \mathbf{q}_2 + \dots + \langle \mathbf{u}_1, \mathbf{q}_n \rangle \mathbf{q}_n \\ \mathbf{u}_2 &= \langle \mathbf{u}_2, \mathbf{q}_1 \rangle \mathbf{q}_1 + \langle \mathbf{u}_2, \mathbf{q}_2 \rangle \mathbf{q}_2 + \dots + \langle \mathbf{u}_2, \mathbf{q}_n \rangle \mathbf{q}_n \\ &\vdots \\ \mathbf{u}_n &= \langle \mathbf{u}_n, \mathbf{q}_1 \rangle \mathbf{q}_1 + \langle \mathbf{u}_n, \mathbf{q}_2 \rangle \mathbf{q}_2 + \dots + \langle \mathbf{u}_n, \mathbf{q}_n \rangle \mathbf{q}_n \end{aligned}$$

QR Decomposition

$$
\begin{aligned}
\mathbf{u}_1 &= \langle \mathbf{u}_1, \mathbf{q}_1 \rangle \mathbf{q}_1 + \langle \mathbf{u}_1, \mathbf{q}_2 \rangle \mathbf{q}_2 + \ldots + \langle \mathbf{u}_1, \mathbf{q}_n \rangle \mathbf{q}_n \\
\mathbf{u}_2 &= \langle \mathbf{u}_2, \mathbf{q}_1 \rangle \mathbf{q}_1 + \langle \mathbf{u}_2, \mathbf{q}_2 \rangle \mathbf{q}_2 + \ldots + \langle \mathbf{u}_2, \mathbf{q}_n \rangle \mathbf{q}_n \\
&\vdots \\
\mathbf{u}_n &= \langle \mathbf{u}_n, \mathbf{q}_1 \rangle \mathbf{q}_1 + \langle \mathbf{u}_n, \mathbf{q}_2 \rangle \mathbf{q}_2 + \ldots + \langle \mathbf{u}_n, \mathbf{q}_n \rangle \mathbf{q}_n
\end{aligned}
$$

In matrix form:

$$
\begin{bmatrix} \mathbf{u}_1 & | & \mathbf{u}_2 & | & \ldots & | & \mathbf{u}_n \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1 & | & \mathbf{q}_2 & | & \ldots & | & \mathbf{q}_n \end{bmatrix} \begin{bmatrix} \langle \mathbf{u}_1, \mathbf{q}_1 \rangle & \langle \mathbf{u}_2, \mathbf{q}_1 \rangle & \ldots & \langle \mathbf{u}_n, \mathbf{q}_1 \rangle \\ \langle \mathbf{u}_1, \mathbf{q}_2 \rangle & \langle \mathbf{u}_2, \mathbf{q}_2 \rangle & \ldots & \langle \mathbf{u}_n, \mathbf{q}_2 \rangle \\ & \vdots & & \\ \langle \mathbf{u}_1, \mathbf{q}_n \rangle & \langle \mathbf{u}_2, \mathbf{q}_n \rangle & \ldots & \langle \mathbf{u}_n, \mathbf{q}_n \rangle \end{bmatrix}
$$

$$
A = QR
$$

QR Decomposition

$$\begin{bmatrix} \langle \mathbf{u_1}, \mathbf{q_1} \rangle & \langle \mathbf{u_2}, \mathbf{q_1} \rangle & \dots & \langle \mathbf{u_n}, \mathbf{q_1} \rangle \\ \langle \mathbf{u_1}, \mathbf{q_2} \rangle & \langle \mathbf{u_2}, \mathbf{q_2} \rangle & \dots & \langle \mathbf{u_n}, \mathbf{q_2} \rangle \\ \vdots & & & \\ \langle \mathbf{u_1}, \mathbf{q_n} \rangle & \langle \mathbf{u_2}, \mathbf{q_n} \rangle & \dots & \langle \mathbf{u_n}, \mathbf{q_n} \rangle \end{bmatrix}$$

From Gram-Schmidt, for $j \geq 2$ the vector $\mathbf{q}_j$ is orthogonal to $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{j-1}$.

$$\begin{bmatrix} \langle \mathbf{u_1}, \mathbf{q_1} \rangle & \langle \mathbf{u_2}, \mathbf{q_1} \rangle & \dots & \langle \mathbf{u_n}, \mathbf{q_1} \rangle \\ 0 & \langle \mathbf{u_2}, \mathbf{q_2} \rangle & \dots & \langle \mathbf{u_n}, \mathbf{q_2} \rangle \\ \vdots & & & \\ 0 & 0 & \dots & \langle \mathbf{u_n}, \mathbf{q_n} \rangle \end{bmatrix}$$

QR Decomposition

Theorem If A is an $m \times n$ matrix with linearly independent column vectors, then A can be factored as

$$A = QR$$

where Q is an $m \times n$ matrix with orthonormal column vectors, and R is an $n \times n$ invertible upper triangular matrix.

Fast Computation of the Normal Equations

- If A is full rank (which it usually is in applications) its QR decomposition provides an efficient way to solve the normal equations.
- Let $A = QR$ be the reduced QR decomposition of A, so Q is $m \times n$ with orthonormal columns and R is $n \times n$, invertible, and upper triangular.
- Since $Q^{\mathsf{T}}Q = I$, and since R^{T} is invertible, the normal equations can be reduced as follows:

$$\begin{aligned} A^{\mathsf{T}}A\hat{\mathbf{z}} &= A^{\mathsf{T}}\mathbf{y} \\ (QR)^{\mathsf{T}}QR\hat{\mathbf{z}} &= (QR)^{\mathsf{T}}\mathbf{y} \\ R^{\mathsf{T}}Q^{\mathsf{T}}QR\hat{\mathbf{z}} &= R^{\mathsf{T}}Q^{\mathsf{T}}\mathbf{y} \\ R^{\mathsf{T}}R\hat{\mathbf{z}} &= R^{\mathsf{T}}Q^{\mathsf{T}}\mathbf{y} \\ R\hat{\mathbf{z}} &= Q^{\mathsf{T}}\mathbf{y} \end{aligned}$$

- Thus $\hat{\mathbf{z}}$ is the least squares solution to $A\mathbf{z} = \mathbf{y}$ if and only if $R\hat{\mathbf{z}} = Q^{\mathsf{T}}\mathbf{y}$.
- Since R is upper triangular, this equation can be solved quickly with back substitution.

Best Approximation

The problem of finding the vector on a space that "best approximates" a given vector **y**, where by "best approximation", we just mean closest, comes up again and again:

- in least-squares,
- image compression,
- in principal component analysis, another data analysis technique, and
- in latent semantic analysis, an information retrieval technique.

Fitting a Circle

Suppose the set of m points $\{(x_i, y_i)\}_{i=1}^m$ are arranged in a nearly circular pattern. The general equation of a circle with radius r and center (c_1, c_2) is

$$(x - c_1)^2 + (y - c_2)^2 = r^2. \tag{1}$$

The circle is uniquely determined by r, c_1, and c_2, so these are the parameters that should be solved for in a least squares formulation of the problem. However, (1) is not linear in any of these variables.

Fitting a Circle

$$
\begin{aligned}
(x - c_1)^2 + (y - c_2)^2 &= r^2 \\
x^2 - 2c_1x + c_1^2 + y^2 - 2c_2y + c_2^2 &= r^2 \\
x^2 + y^2 &= 2c_1x + 2c_2y + r^2 - c_1^2 - c_2^2 \qquad (2)
\end{aligned}
$$

The quadratic terms x^2 and y^2 are acceptable because the points $\{(x_i, y_i)\}_{i=1}^m$ are given. To eliminate the nonlinear terms in the unknown parameters r, c_1, and c_2, define a new variable $c_3 = r^2 - c_1^2 - c_2^2$. Then for each point (x_i, y_i), (2) becomes

$$2c_1x_i + 2c_2y_i + c_3 = x_i^2 + y_i^2.$$

These m equations are linear in c_1, c_2, and c_3, and can be written as the linear system

$$
\begin{bmatrix} 2x_1 & 2y_1 & 1 \\ 2x_2 & 2y_2 & 1 \\ \vdots & \vdots & \vdots \\ 2x_m & 2y_m & 1 \end{bmatrix}
\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} =
\begin{bmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \\ \vdots \\ x_m^2 + y_m^2 \end{bmatrix}. \qquad (3)
$$

Fitting a Circle

After solving for the least squares solution, r can be recovered with the relation $r = \sqrt{c_1^2 + c_2^2 + c_3}$.

Plotting a circle is best done with polar coordinates. Using the same variables as before, the circle can be represented in polar coordinates by setting

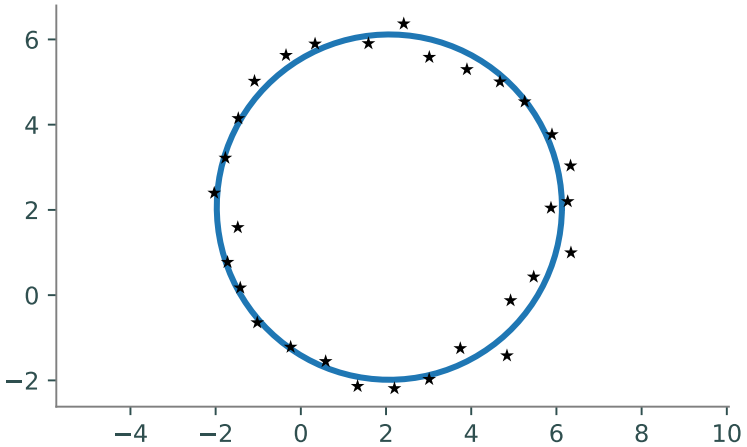
$$x = r\cos(\theta) + c_1, \qquad y = r\sin(\theta) + c_2, \qquad \theta \in [0, 2\pi]. \tag{4}$$

To plot the circle, solve the least squares system for c_1, c_2, and r, define an array for θ, then use (4) to calculate the coordinates of the points the circle.

In Python

```
# Load some data and construct the matrix A and the vector b.
>>> xk, yk = np.load("circle.npy").T
>>> A = np.column_stack((2*xk, 2*yk, np.ones_like(xk)))
>>> b = xk**2 + yk**2

# Calculate the least squares solution and solve for the radius.
>>> c1, c2, c3 = la.lstsq(A, b)[0]
>>> r = np.sqrt(c1**2 + c2**2 + c3)

# Plot the circle using polar coordinates.
>>> theta = np.linspace(0, 2*np.pi, 200)
>>> x = r*np.cos(theta) + c1
>>> y = r*np.sin(theta) + c2
>>> plt.plot(x, y)                  # Plot the circle.
>>> plt.plot(xk, yk, 'k*')          # Plot the data points.
>>> plt.axis("equal")
```

In Python



Outline

Solution of Systems of Linear Equations

Methods for solving systems of linear equations:

- Gaussian-Jordan elimination
- QR decomposition
- Numerical methods (aka iterative methods)
- LU decomposition (only for square matrices)

Iterative Methods

- A matrix A is said to be ill conditioned if relatively small changes in the entries of A can cause relatively large changes in the solutions of $A\mathbf{x} = \mathbf{b}$.
- A is said to be well conditioned if relatively small changes in the entries of A result in relatively small changes in the solutions of $A\mathbf{x} = \mathbf{b}$.
- Reaching RREF as in Gauss-Jordan requires more computation and more numerical instability hence disadvantageous.
- Gauss elimination is a direct method: the amount of operations can be specified in advance. Indirect or Iterative methods work by iteratively improving approximate solutions until a desired accuracy is reached. Amount of operations depend on the accuracy required. (way to go if the matrix is sparse)

Gauss-Seidel Iterative Method

Example

$$
\begin{array}{rcrcrcrcl}
x_1 & - & 0.25x_2 & - & 0.25x_3 & & & = & 50 \\
-0.25x_1 & + & x_2 & & & - & 0.25x_4 & = & 50 \\
-0.25x_1 & & & + & x_3 & - & 0.25x_4 & = & 25 \\
& - & 0.25x_2 & - & 0.25x_3 & + & x_4 & = & 25
\end{array}
$$

$$
\begin{array}{rcrcrcrcl}
x_1 & = & & & 0.25x_2 & + & 0.25x_3 & + & 50 \\
x_2 & = & 0.25x_1 & & & & 0.25x_4 & + & 50 \\
x_3 & = & 0.25x_1 & & & & 0.25x_4 & + & 25 \\
x_4 & = & & & 0.25x_2 & + & 0.25x_3 & + & 25
\end{array}
$$

We start from an approximation, eg, $x_1^{(0)} = 100, x_2^{(0)} = 100, x_3^{(0)} = 100, x_4^{(0)} = 100$, and use the equatiuons above to find a perhaps better approximation:

$$
\begin{array}{rcrcrcrcrcl}
x_1^{(1)} & = & & & 0.25x_2^{(0)} & + & 0.25x_3^{(0)} & + & 50.00 & = & 100.00 \\
x_2^{(1)} & = & 0.25x_1^{(1)} & & & & 0.25x_4^{(0)} & + & 50.00 & = & 100.00 \\
x_3^{(1)} & = & 0.25x_1^{(1)} & & & & 0.25x_4^{(0)} & + & 25.00 & = & 75.00 \\
x_4^{(1)} & = & & & 0.25x_2^{(1)} & + & 0.25x_3^{(1)} & + & 25.00 & = & 68.75
\end{array}
$$

$$
\begin{array}{rcrcrcrcl}
x_1^{(2)} & = & & 0.25x_2^{(1)} & + & 0.25x_3^{(1)} & & + & 50.00 = 93.750 \\
x_2^{(2)} & = & 0.25x_1^{(2)} & & & & + & 0.25x_4^{(1)} + & 50.00 = 90.625 \\
x_3^{(2)} & = & 0.25x_1^{(2)} & & & & + & 0.25x_4^{(1)} + & 25.00 = 65.625 \\
x_4^{(2)} & = & & 0.25x_2^{(2)} & + & 0.25x_3^{(2)} & & + & 25.00 = 64.062
\end{array}
$$

Summary

1. Calculus Approach

2. Linear Algebra Approach

3. Other Methods