Computational Mathematics for Learning and Data Analysis

Federico Matteoni

A.A. 2021/22

Index

0.1 Introduction

Exam: project (groups of 2) + oral exam.
This course's goals is to make sense of the huge amounts of data, take something big and unwieldy and produce something small that can be used, a **mathematical model**.
The mathematical model should be accurate, computationally inexpensive and general, but generally is not possible to have all three. General models are convenient (work once, apply many), they are parametric so we need to learn the right values of the parameters. Fitting is finding the model that better represents the phenomenon given a family of possible models (usually, infinitely many). Is an optimization model and usually is the computational bottleneck.
ML is better than fitting because fitting reduces the training error, the empirical risk, but ML reduces the test error, so the generalization error.
Solve general problem $\min x \in Sf(x)$, with Poloni solve $\min x \in R^n ||Ax-b||_2$ which is easier and can be solved exactly.

Capitolo 1

Numerical Analysis

1.1 Quick recap of linear algebra

Matrix - Vector multiplication, with $A \in R^{4\times3}, c \in R^3, b \in R^4$

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ \vdots & \vdots & \vdots \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \qquad \begin{array}{l} b_i = \sum_{j=1}^{4} A_{ij}c_j \\ A_{11}c_1 + A_{12}c_2 + A_{13}c_3 = b_1 \end{array}$$

or linear combination of the columns

$$\begin{bmatrix} A_{11} \\ A_{21} \\ A_{31} \\ A_{41} \end{bmatrix} c_1 + \begin{bmatrix} A_{12} \\ A_{22} \\ A_{32} \\ A_{42} \end{bmatrix} c_2 + \begin{bmatrix} A_{13} \\ A_{23} \\ A_{33} \\ A_{43} \end{bmatrix} c_3 + \begin{bmatrix} A_{14} \\ A_{24} \\ A_{34} \\ A_{44} \end{bmatrix} c_4 = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

with c_1, c_2, c_3 and c_4 called coordinates.

Basis: tuple of vectors $v_1, v_2, \ldots, v_n \mid$ you can write all vectors b in a certain space as a linear combination $v_1\alpha_1 + v_2\alpha_2 + \ldots + v_n\alpha_n$ with **unique** $a_1, \ldots, a_n$. The canonical basis is

$$c_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad c_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad c_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad c_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

and, for example

$$\begin{bmatrix} 3 \\ 5 \\ 7 \\ 9 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \cdot 3 + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot 5 + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \cdot 7 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot 9$$

Image ImA = set of vectors b that we can reach with A

Kernel $KerA$ = set of vectors $x \mid Ax = 0$ ($x = 0$ is certainly one, there may be others)

Invertible A if this problem has exactly one solution.
$\forall\, b \in R^m$, A must be square and the columns of A are a basis of $R^m \Rightarrow x = A^{-1}b$ where A^{-1} is another square matrix $\mid A \cdot A^{-1} = A^{-1} \cdot A = I$ identity matrix (1 on the diagonal, 0 otherwise)
Implementation detail: `inv(A) * b` is not the best choice. Better: in Python `scipy.linalg.solv(A, b)` or, in Matlab, `A  b`.

Cost, with $A \in R^{m\times n}, B \in R^{n\times p}, C \in R^{m\times p}$ (vectors $\Leftrightarrow n \times 1$ matrices), then the cost of multiplication is $mp(2n-1)$ floating point ops (*flops*), or $O(mnp)$.
In particular, A, B squared $\Rightarrow AB$ costs $O(m^3)$. With A, v vector $\Rightarrow Av$ costs $O(m^2)$. Faster alternatives are not worth it usually. And remember that $AB \neq BA$ generally, and also that $CA = CB \not\Rightarrow A = B$ with C matrix.
If there's $M \mid MC = I$, then $A = (MC)A = (MC)B = B$ (multiplying *on the left* by M on both sides)

Why a real valued function? Strong assumption, given x' and x'', I can always tell which one I like best (**total order** of R). Often more than one objective function, with contrasting and/or incomparable units (ex: loss function vs regularity in ML).
But R^k with $k > 1$ has no total order $\Rightarrow$ no *best* solution, only non-dominated ones.
Two practical solutions: maximize return with budget on maximum risk or maximize...
Even with a single objective function optimization is hard, impossible if f has no minimum in X (so, the problem P is unbounded below. Hardly ever happens in ML, because loss and regularization are ≥ 0
Also impossible if $f > -\infty$ but $\nexists x$, for example in $f(x) = e^x$. However plenty of ϵ-approximate solutions (ϵ-optima).
On PC $x \in R$ is in fact $x \in Q$ with up to 16 digits precision, so approximation errors are unavoidable anyway. Exact algebraic computation is possible but usually slow, and ML is going the opposite way (less precision: floats, half, small integer weights...).
Anyway finding the exact x_* is impossible in general.

Norms

$$||x||_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2} = \sqrt{x^T x} = [x_1 \ldots x_n] \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_b \end{bmatrix}$$

Many matrices preserve norm 2

Orthogonal A square matrix $U \in R^{n \times n}$ is orthogonal if $U^T U = I \vee UU^T = I \vee U^{-1} = U^T$ equivalently
Theorem: for an orthogonal matrix $U \in R^{n \times n}$ and every vector x, $||Ux|| = ||x||$
More generally, $x^T y = (Ux)^T (Xy)$ for all vectors $x, y \in R^n$
The columns of a orthogonal matrix are **orthonormal**: $u_i^T u_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$
If $U, V \in R^{n \times n}$ are orthogonal $\Rightarrow UV$ is orthogonal

Eigenvalues and eigenvectors Given a square matrix $A \in R^{m \times m}$, if $Av = \lambda v$ for $v \in R^m, \lambda \in R$ then we call λ and **eigenvalue** and v an **eigenvector** of A.
For many matrices we can find $v_1, \ldots, v_m$ eigenvectors that are a basis of R^m, i.e. $[v_1 | \ldots | v_m] = V$ is invertible.
$A = V \cdot \Lambda \cdot V^{-1} \Leftrightarrow AV = V\Lambda$, with $\Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m \end{bmatrix}$
Given $x \in R^m$ I can write $x = v_1\alpha_1 + \ldots + v_m\alpha_m$ then it's easy to compute the coordinates $Ax = A(v_1\alpha_1 + \ldots + v_m\alpha_m) = v_1(\lambda_1\alpha_1) + \ldots + v_m(\lambda_m\alpha_m)$
Also $A^k x = v_1(\lambda_1^k \alpha_1) + \ldots + v_m(\lambda_m^k \alpha_m)$
Eigenvalue decompositions do not exists unique for all matrices. If v is an eigenvector, then $v \cdot \alpha$ is still one, and if v, w are eigenvectors of the same eigenvalue then $v + w$ is still an eigenvector of the same eigenvalue, and the same for $\alpha v + \beta w$ (linear combination). Some matrices have complex eigenvalues/eigenvectors only, and some do not have enough eigenvectors to make a masis.

Symmetry A is symmetric if $A = A^T$

Spectral Theorem For a symmetric matrix $A \in R^{m \times m}$ we can always write $A = U\Lambda U^{-1}$

Quadratic form Given a symmetric matrix $Q = Q^T \in R^{m \times m}$ we can consider the function $x \in R^m \mapsto f(x) = x^T Q x \in R$
Theorem For all symmetric matrix $Q \in R^{m \times m}$, $\lambda_{min}||x||^2 \leq x^T Q x \leq \lambda_{max}||x||^2$, where $\lambda_{min}, \lambda_{max}$ are the smallest and largest eigenvalue of Q.

Positive Semidefinite $Q = Q^T$ is positive semidefinite if all eigenvalues $\lambda_i \geq 0 \Leftrightarrow \lambda_{min} \geq 0$ hence $x^T Q x \geq 0$ for all $x \in R^m$
Positive definite if all eigenvalues $\lambda_i > 0 \Leftrightarrow \lambda_{min} > 0$ hence $x^T Q x > 0$ for all $x \in R^m, x \neq 0$

Recall theorem $\lambda_{min}(x^Tx) \leq x^TQx \leq \lambda_{max}(x^Tx)$ for all symmetric Q so $\lambda_{min} \leq \frac{x^TQx}{x^Tx} \leq \lambda_{max}$
Slightly different form $\lambda_{min} \leq z^TQz \leq \lambda_{max}$ for all vectors z with $||z|| = 1$. Equivalent to the other form with $x = \alpha z$, for $\alpha = ||x||$ and a vector z with $||z|| = 1$

$$\frac{x^TQx}{x^Tx} = \frac{(\not{\alpha}z)^TQ(\not{\alpha}z)}{\not{\alpha}^{\not{2}}}$$

Generalization for complex matrices $||x||^2 = |x_1|^2 + \ldots + |x_n|^2$, $x^T \longrightarrow \overline{x^T} = x^*$. For orthogonal matrices $U^*U = I \Rightarrow U$ is unitary. For symmetry $Q^* = Q \Rightarrow Q$ is Hermitian.

Singular Value Decomposition Each $A \in R^{n\times n}$ can be decomposed as $A = U\Sigma V^T$ with U, V orthogonal and Σ diagonal with σ_i on the diagonal with $\sigma_1 \geq \ldots \geq \sigma_n \geq 0$.
The first notable difference is it exists for every square matrix. The second difference is V^T which is not the inverse of U.
Another notation is $[u_1|u_2|\ldots|u_m]\begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_m \end{bmatrix}\begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_m^t \end{bmatrix} = u_1\sigma_1v_1^T + \ldots + u_m\sigma_mv_m^T$ sum of m rank-1 matrices
Geometric idea: there is an orthogonal basis $v_1, \ldots, v_m$ so that A maps $\{v_i\}$ into multiples of another orthogonal basis $Av_i = u_i\sigma_i$
$\{\sigma_i\}$: singular values of A, defined uniquely for each A
Rectangular SVD: each $A \in R^{m\times n}$ can be decomposed as $A = U\Sigma V^T$ where $U \in R^{m\times m}$, $V \in R^{n\times n}$ are orthogonal and $\Sigma \in R^{m\times n}$ is diagonal, so $\sigma_{i,j} = 0$ whenever $i \neq j$ again with $\sigma_1 \geq \ldots \geq \sigma_{min(m,n)} \geq 0$
So only the first n vectors of U matches with non-zero values in Σ, all the last $n - m$ columns combine with zeroes.
$= u_i\sigma_1v_1^T + \ldots + u_n\sigma_nv_n^T$ with $u_{n+1}, \ldots, u_m$ not used. So we can "chop off" the unused parts and get the same result.
$= u_i\sigma_1v_1^T + \ldots + u_{min(m,n)}\sigma_{min(m,n)}v_{min(m,n)}^T$
In Matlab, `svd(A, 'econ')` costs $max(m,n)\cdot min(m,n)^2$, still cubic but linear in the largest dimension. the full `[U, S, V] = svd(A)` cannot be linear because one of the outputs will be a huge orthogonal matrix of $max(m,n) \times max(m,n)$, so it will cost more in time and memory.

The rank of a matrix A is equal to the number of non-zero σ_i. $\sigma_1 \geq \ldots$ so at one point a $\sigma_r > \sigma_{r+1} = \ldots = \sigma_{min(m,n)} = 0$
Given $A = U\Sigma V^T$ we can compute $A^TA = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma^TU^TU\Sigma VV^T = V\Sigma^T\Sigma V^T$ with $\Sigma^T\Sigma$ diagonal and $V\Sigma^T\Sigma V^T$ is both an eigenvalue decomposition and an SVD. This proved that the eigenvalues of A^TA Are the squares of the singular values of A plus additional zeroes for dimension reasons.

$$||A||_2 = ||U\Sigma V^T||_2 = ||\Sigma V^T||_2 = ||\Sigma||_2 = \Sigma_1$$

$$||A|| = max_{||z||=1}||\Sigma V^T|| = \sqrt{\sigma_1^2z_1^2 + \ldots + \sigma_n^2z_n^2} \leq \sigma_1\sqrt{z_1^2 + \ldots + z_n^2} = \sigma_1||z|| = \sigma_1$$

$$||A||_F = ||U\Sigma U^T|| = \ldots = \Sigma_1$$

Eckart-Young Theorem Most important property of the SVD decomposition.
We are interested in approximating A with matrices of rank $\leq K$, if $K = 1$ this means find two vectors u, v so that $A = u^Tv$, with $K = 2$ then $A = u_1^Tv_1 + u_2^Tv_2$. What is "how close": $\min rank(X) \leq K||A - X||$. The theorem states that the solution is related to SVD.
The optimal solution of $\min rank(X) \leq X||A - X||$ is $X = u_1\sigma_1v_1^T + \ldots + u_k\sigma_kv_k^T$
where $A = u_1\sigma_1v_1^T + \ldots + u_{min(m,n)}\sigma_{min(m,n)}v_{min(m,n)}^T$ is an SVD, $A = U\Sigma V^T$.

Ranks If A has rank 1, then $A = u \cdot v^T$ and $A_{ij} = u_i \cdot v_j$
If A has rank 2, then $A = u_1 \cdot v_1^T + u_2 \cdot v_2^T$ and $A_{ij} = (u_1)_i \cdot (v_1)_j + (u_2)_i \cdot (v_2)_j$

1.2 SVD Approximation

$X_1 = u_i\sigma_1v_1^T =$ best approximation score of student $i \cdot n\cdot$ best approximation difficulty of exercise j
As a statistical estimator: suppose my scores are of the form $A_{ij} = u_iv_j + \epsilon_{ij}$ with ϵ_{ij} being the error in the score for instance gaussian with variance λ.
The rank 1 approx of $(u_1\sigma_1)$, v_1 given by SVD is the one that minimizes $\sum(A_{ij} - u_iv_j)^2 = ||A - X_1||_F^2 = \sum \epsilon_{ji}^2$. This

is the maximum-likelihood estimation of abilities $(u_1)_i, (v_1)_j$.

The best rank 2 approximation is $X_2 = u_1\sigma_1 v_1^T + u_2\sigma_2 v_2^T$ which can be viewed as first approximation plus corrections.

$\sigma_1 >> \sigma_2, \ldots$ then A very close to rank 1.
$\sigma_1, \sigma_2 >> \sigma_3, \ldots$ then A very close to rank 2, and so on.

Best approximations $X =$ best rank-1 approx of I, $X = u_1\sigma_1 v_1^T$, $x_{ij} = (u_1)_j \sigma_1 (v_1^T)_j$
Best rank-2 $X_2 = u_1\sigma_1 v_1^T + u_2\sigma_2 v_2^T$
Best rank-3 $X_3 = u_1\sigma_1 v_1^T + u_2\sigma_2 v_2^T + u_3\sigma_3 v_3^T$ And so on...
The original image was $256\times256 = 2^{16}$ reals. The compressed version, with $k = 25$ we have $256 \cdot 5 \cdot 2 + 25$ which is about a factor of 5 less.

What is $||A - X_k||_F = \sqrt{\sum (a_{ij} - x_{ij})^2} = ||U\Sigma V^T - U[\text{main diagonal of } \sigma_i \text{ untile } \sigma_k]V^T|| = ||U([\ldots = \sqrt{\sum_{i=k+1}^{min(m,n)} \sigma_i^2}$

Linear Least Squares problems Given vectors $a_1, \ldots, a_n \in R^m$, so that $A = [a_1|\ldots|a_n] \in R^{m\times n}$, and targets $b \in R^m$, find $x_1, \ldots, x_n \in R \mid a_1x_1 + \ldots + a_nx_n = b$
Not always solvable, for example $\begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} x_1 + \begin{bmatrix} 1 \\ 3 \\ 0 \end{bmatrix} x_2 = \begin{bmatrix} 5 \\ 5 \\ 1 \end{bmatrix}$ because the third component is always $0 \neq 1$. As a backup question, how close can I get to b? I can get $\begin{bmatrix} 5 \\ 5 \\ 0 \end{bmatrix}$
In general $\min x \in R^m ||Ax - b||_2 = \min x \in R^m \sqrt{\sum ((Ax)_i - b_i)^2}$
The special case is $m = n$, i.e. # vectors = length, then the problem is solvable $\Leftrightarrow$ the vectors are a basis $\Leftrightarrow$ or A invertible. Typical case il A long thin, we cannot get all vectors b but still $min_{x\in R^m} ||Ax - b||_2$ is a question that makes sense.

Polynomial Fitting Statistical version: given (x_i, y_i), what is the choice of coefficients that "most likely" generated them? I can get (x_i, y_i) starting from every polynomial, with the right set of random numbers. The **maximum likelihood estimator** on this problem is $min_{coeff} ||Ax - y||_2^2$

Theory of least-squares problems When does $min||Ax - b||_2$ have a unique solution? With $A \in R^{m\times n}$
We know that if $m = n$ then $Ax = b$ has a unique solution $\Leftrightarrow$ A is an invertible matrix. If this happens, then $O = min||Ax - b||$ with unique x.
We say that $A \in R^{m\times n}$ has **full column rank** if $Ker(A) = \{0\} \Leftrightarrow$ there is no $z \in R^n z \neq 0 \mid Az = 0 \Leftrightarrow rk(A) = n$ and this can only happen is $m \geq n$

Theorem The least-squares problem $min||Ax - b||$ has unique solution $x \Leftrightarrow A$ has full column rank.
Lemma: A has full column rank $\Leftrightarrow A^TA$ is positive definite.
Proof $Ax \neq 0 \ \forall\, z \in R^n, z \neq 0$

$\Leftrightarrow\ ||Az||_2 \neq 0 \ \forall\, z \in R^n, z \neq 0$

$\Leftrightarrow\ ||Az||_2^2 \neq 0 \ \forall\, z \in R^n, z \neq 0$

$\Leftrightarrow\ (Az)^T(Az) \neq 0 \ \forall\, z \in R^n, z \neq 0$

$\Leftrightarrow\ z^TA^TAz \neq 0 \ \forall\, z \in R^n, z \neq 0 \longleftarrow$ definition of $A^TA > 0$

By manipulating the original problem $min_{x\in R^n} ||Ax - b||_2$ we obtain

$$min||Ax - b||_2 = minx^TA^TAx - 2b^TAx + b^Tb \Leftrightarrow f(x) = x^TQx + q^Tx + c$$

which is a quadratic problem and find that it has a unique minimum $x \Leftrightarrow$ it is strongly convex $\Leftrightarrow Q \succ 0$ (positive definite)
$f(x)$ convex $\Leftrightarrow Q \geq 0$, strongly/strictly convex $\Leftrightarrow Q \succ 0$ (positive definite)

So the least-squares problem $min_x||Ax - b||$ has unique solution

$\Leftrightarrow$ $f(x)$ has a unique minimum point

$\Leftrightarrow$ $2A^TA = Q \succ 0$ (positive definite)

$\Leftrightarrow$ $A^TA > 0 \Leftrightarrow A$ has full column rank (for the lemma)

The minimum is when $grad\, f(x) = 0 \Leftrightarrow 2Qx + q = 0 \Leftrightarrow 2A^TAx - 2A^Tb = 0$ so when $A^TAx = A^Tb$ square linear system, with A^TA invertible (because positive definite).
x is obtained (intuitively) from multiplying $Ax = b$ on the left with A^T.

Algorithm

1. Form A^TA, $n \times m \cdot m \times n$ product so it costs $2mn^2$ floating point operations (flops) plus lower order terms
2. Form A^Tb, costs $2mn$ flops lower order terms
3. Solve $A^TAx = A^Tb$ (for example with gaussian elimination or LU factorization) costs $\frac{2}{3}n^3$ flops plus lower order terms

If $m \geq n$ then the overall complexity is $O(mn^2)$ same as SVD.
Possible optimizations:

1. A^TA symmetric so can compute only upper triangle then mirror the rest so from $2mn^2$ becomes mn^2 flops
2. Already a cheap step
3. Other algorithms to solve this linear system because the matrix A^TA is positive definite (example: Cholesky factorization, complexity is $\frac{1}{3}n^3$ flops, half the cost)

Pseudoinverse $x = A^TA^{-1}b$ can be denoted as the product of $A^+ = A^TA^{-1}A^T$ and b. A^+ is the pseudoinverse, or **Moore-Penrose pseudoinverse**. The definition is valid only when A has full column rank. If $A \in R^{m\times n}$ then $A^+ \in R^{n\times m}$. Note that $A^+A = (A^TA)^{-1}(A^TA) = I \in R^{n\times n}$, while $AA^+ = A(A^TA)^{-1}A^T \neq I \in R^{m\times m}$. The latter is impossible, because the columns of AA^+ are linear combinations of the columns of A, so AA^+ has rank of at most n. As consequences, if x_1 is solution of $min||Ax - b_1||$ and x_2 is solution of $min||Ax - b_2||$ then $x_1 + x_2$ is solution of $min||Ax - (b_1 + b_2)||$

Sometimes ML problems are formulated "from the left side". With $w \in R^{1\times n}$ row vector of weights, then $X \in R^{n\times m}$ short-fat ($n \leq m$) that has a row for each "feature" in the input pattern.
$y \in R^{1\times m}$ row vector "target"
The problem is $min||wX - y||$, same problem just transposed. Solution $w = yX^+$ with $X^+ = X^T(XX^T)^{-1}$ if X has full row rank.

1.3 QR factorization

Given $x \in R^n$, find an orthogonal matrix H such that Hx is a vector of the form $s \cdot e_1 = \begin{bmatrix} s \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ with $e_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

Since H is orthogonal, $||x|| = ||\begin{bmatrix} s \\ 0 \\ \vdots \\ 0 \end{bmatrix}|| = |s| \Rightarrow s = \pm||x||$. We will find H in a very specific form

$$H = I - \frac{2}{v^Tv}vv^T =$$

$$= I - \frac{2}{||v||^2}vv^T =$$

$$= I - 2uu^T$$

for a certain $v \in R^n$ with $u = \frac{v}{||v||}$, so it has norm = 1 and is parallel to v. This form is called **Householder reflector**.
Lemma: for every $v \in R^n$, H is orthogonal *and* symmetric. Proof:

Symmetric: $H^T = (I - 2uu^T)^T = I^T - (2uu^T)^T = I - 2uu^T = H$

Orthogonal: $H^T H = H^2 = (I - 2uu^T)(I - 2uu^T) = I - 2uu^T - 2uu^T + 4uu^T uu^T$ but $u^T u = ||u||^2 = 1$ so $= I - \not{4}uu^T + \not{4}uu^T = I$

By computing Hx as $x - 2u(u^T x)$, meaning you compute $\alpha = u^T x$ in $O(n)$ and then compute $x - 2u\alpha$ in $O(n)$, you can reduce the complexity of Hx from $O(n^2)$ to $O(n)$. By doing the same on every column you reduce HA for an arbitrary $A \in R^{n \times n}$ from $O(n^3)$ to $O(n^2)$.
Lemma: given any two vectors $x, y \in R^n$ with $||x|| = ||y||$, the Householder reflector built with $v = x - y$ is such that $Hx = y$

Numerical problems If x_1 is close to $s = ||x||$ there are problems, because we do a subtraction between close numbers. Quick fix: switch to $s = -||x||$, then $x_1 - s = x_1 + \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$, an addition between positive numbers whenever $x_1 > 0$. In general, we choose:

if $x_1 \geq 0$, we take $s = -||x||$ and $y = \begin{bmatrix} -||x|| \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

if $x_1 z 0$, we take $s = ||x||$ and $y = \begin{bmatrix} ||x|| \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

This way $x_1 - s$ is never a "true subtraction", because x_1 and s have different signs.

Theorem $\forall\, A \in R^{m \times n}\ \exists\, Q \in R^{m \times m}$ orthogonal, $R \in R^{m \times n}$ upper triangular $\mid A = QR$

When $n = 1$ we already solved: $\forall\, x \in R^{m \times 1}\ \exists$ a Householder reflector $H \mid Hx = \begin{bmatrix} s \\ 0 \\ \vdots \\ 0 \end{bmatrix} \Leftrightarrow x = H \begin{bmatrix} s \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ with x being the $m \times 1$ matrix, H orthogonal and the array is upper triangular.
The general case is $A \in R^{m \times n}$:

1. $[u_1, s_1]$ = `householder_vector(A(:,1))` $H_1 = I - 2u_1 u_1^T$
   $H_1 A = \begin{bmatrix} s_1 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} = A_1$
2. $A_2 = H_2 A_1 = \begin{bmatrix} * & s_2 & * & * \\ * & 0 & * & * \\ * & 0 & * & * \\ * & 0 & * & * \\ * & 0 & * & * \end{bmatrix}$ won't work because it spoils the first column.
   So $[u_2, s_2]$ = `householder_vector(`A_1`(2:`m`,2))`, which the 2nd to mth row of the second column of A_1. Because if we multiply $\left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & H_2 \end{array}\right] \cdot \left[\begin{array}{c} B \\ \hline C \end{array}\right]$ we get $\left[\begin{array}{c} B \\ \hline H_2 \cdot C \end{array}\right]$, so we say that $Q_2 = \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & H_2 \end{array}\right]$
   and $Q_2 A_1 = \begin{bmatrix} s_1 & * & * & * \\ 0 & s_2 & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{bmatrix} = A_2$

3. With H_3 from $[u_3, s_3]$ = `householder_vector(`A_2`(3:`m`,3))` we do

$$Q_3A_2 = \left[\begin{array}{cc|ccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & & & \\ 0 & 0 & & H_3 & \\ 0 & 0 & & & \end{array}\right] A_2 = \left[\begin{array}{cccc} s_1 & * & * & * \\ 0 & s_2 & * & * \\ 0 & 0 & s_3 & * \\ 0 & 0 & 0 & * \\ 0 & 0 & 0 & * \end{array}\right] = A_3$$

4. $H_4 = I - 2u_4u_4^T$ from $[u_4, s_4]$ = `householder_vector(`A_3`(4:`m`,4))`

$$Q_4A_3 = \left[\begin{array}{c|c} I_3 & 0 \\ \hline 0 & H_4 \end{array}\right] A_3 = \left[\begin{array}{cccc} s_1 & * & * & * \\ 0 & s_2 & * & * \\ 0 & 0 & s_3 & * \\ 0 & 0 & 0 & s_4 \\ 0 & 0 & 0 & 0 \end{array}\right] = A_4$$

So $Q_4Q_3Q_2Q_1A = R$ is an upper triangular matrix.

$$\begin{aligned} \not{Q}_4^T \not{Q}_4Q_3Q_2Q_1A &= Q_4^TR \\ \not{Q}_3^T \not{Q}_3Q_2Q_1A &= Q_3^TQ_4^TR \\ \not{Q}_2^T \not{Q}_2Q_1A &= Q_2^TQ_3^TQ_4^TR \\ \not{Q}_1^T \not{Q}_1A &= Q_1^TQ_2^TQ_3^TQ_4^TR \end{aligned}$$

So $A = Q_1^TQ_2^TQ_3^TQ_4^TR$, and $Q_i = Q_i^T$ so we can omit transpose, giving $A = QR$ with Q orthogonal and R triangular.

The QR factorization can be used to solve least squares problems $min||Ax - b||_2$ with $A \in R^{m\times n}$ and $m \geq n$, $b \in R^m$.

$A = QR = [Q_1 \mid Q_2] \cdot \left[\begin{array}{c} R_1 \\ \hline \\ 0 \end{array}\right]$ with $Q_1 \in R^{m\times n}, Q_2 \in R^{m\times m-n}, R_1 \in R^{n\times n}$ and the last part of $n \times m - n$ zeroes. We rework the objective function

$$||Ax - b|| = ||QRx - b|| = ||Q^T(QRx - b)|| = ||Rx - Q^Tb|| = \left|\left|\left[\begin{array}{c} R_1 \\ 0 \end{array}\right] x - \left[\begin{array}{c} Q_1^T \\ Q_2^T \end{array}\right] b\right|\right| = \left|\left|\left[\begin{array}{c} R_1x - Q_1^Tb \\ -Q_2^Tb \end{array}\right]\right|\right|$$

How to make this norm as small as possible? The norm of the second block, $-Q_2^Tb$, doesn't depend on x. The norm of the first block does. Can I obtain $R_1x - Q_1^Tb = 0$? If R_1 is invertible, $R_1x = Q_1^Tb$ is a square linear system of solution $x = R_1^{-1}Q_1^Tb$. Algorithm:

1. Compute the QR factorization $A = QR = Q_1R_1$ (thin is enough) in $\frac{4}{3}n^3, 2mn^2$ operations.
2. Compute $c = Q_1^Tb$, with $2mn$ operations
3. Solve the linear system $R_1x = c$ with back-substitution, n^2 operations

Step 1 is the most expensive, cubic. When is R_1 invertible? Recall that the least-squares problem has unique solution $\Leftrightarrow$ A has full column rank $\Leftrightarrow$ A^TA is positive definite $\Leftrightarrow$ A^TA invertible $\Leftrightarrow$ R_1 invertible

$$A^TA = (QR)^TQR = R^T \not{Q}^T \not{Q}R = [R_1^T 0] \left[\begin{array}{c} R_1 \\ 0 \end{array}\right] = R_1^TR_1$$

The least-squares problem can also be solved with SVD. Given $A = USV^T$ we have

$$||Ax - b|| = ||USV^Tx - b|| = ||U^T(USV^Tx - b)|| = ||SV^Tx - U^Tb|| =$$

With $S = \left[\begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ & 0 & \end{array}\right]$ and $V^Tx = y$ we have

$$= \left|\left|\left[\begin{array}{c} \sigma_1y_1 \\ \vdots \\ \sigma_ny_n \\ 0 \\ \vdots \\ 0 \end{array}\right] - \left[\begin{array}{c} u_1^Tb \\ \vdots \\ u_m^tb \end{array}\right]\right|\right| = \left|\left|\left[\begin{array}{c} \sigma_1y_1 - u_1^Tb \\ \vdots \\ \sigma_ny_n - u_n^Tb \\ -u_{n+1}^Tb \\ \vdots \\ -u_m^Tb \end{array}\right]\right|\right|$$

The first n entries become 0 if $y_i = \frac{u_i^T b}{\sigma_i}$

$$x = Vy = v_1 y_1 + \ldots + v_n y_n = v_1 \frac{u_1^T b}{\sigma_1} + \ldots + v_n \frac{u_n^T b}{\sigma_n}$$

$$x = V S_1^{-1} U_1^T b = A^+ b$$

The solution of $min||Ax - b||$ is unique $\Leftrightarrow$ $A^T A$ is invertible $\Leftrightarrow$ $A^T A = (USV^T)^T USV^T = VS^T \not{U}^T \not{U} SV^T = V \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_n^2 \end{bmatrix} V^T$ which is the eigenvalue decomposition for $A^T A$

So $A^T A$ is invertible $\Leftrightarrow$ non of the σ_i are zeroes $\Leftrightarrow$ S_1 is invertible.

What happens if $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r > \sigma_{r+1} = \ldots \sigma_n = 0$? We would have $\left|\left|\begin{bmatrix} \sigma_1 y_1 - u_1^T b \\ \vdots \\ \sigma_r y_r - u_r^T b \\ -u_{r+1}^T b \\ \vdots \\ -u_n^T b \\ -u_{n+1}^T b \\ \vdots \\ -u_m^T b \end{bmatrix}\right|\right|$

How to choose a special solution here? Let's define another problem: $min||x||$ of all $x \in$ solutions of $min||Ax - b||$. It's minimized when $y_1 = \frac{u_1^T b}{\sigma_1}, \ldots, y_r = \frac{u_r^T b}{\sigma_r}, y_{r+1} = 0, \ldots, y_n = 0$

The problem is that on computers the zeroes are very often not zeroes. Even including a term $v_{r+1} \frac{u_{r+1}^T b}{\sigma_{r+1}}$ with a small σ_{r+1} gives a huge contribution to the sum. In many case, stopping the sum early is beneficial: **truncated SVD**, with $i = 1, \ldots, k$ and $k < m$. The Eckart-Young aproximation theorem can help too.

Effect of noise in data Suppose to know $A + E = \overline{A}$, with A exact data, E error/noise/etc. and $\overline{A}$ the observed data. We have

$\sigma_1, \ldots, \sigma_n = \text{SVD}(A)$

$\overline{\sigma_1}, \ldots, \overline{\sigma_n} = \text{SVD}(\overline{A})$

Then $|\sigma_i - \overline{\sigma_i}| \leq ||E||_2$

The effect of noise is that there are longer relative changes to smaller singular values, another reason why it's beneficial to drop them.

Tikhonov Regularization/Ridge Regression Another solution, alternative to truncated SVD. Change the problem to $min_{x \in R^n} ||Ax - b||^2 + \lambda^2 ||x||^2 = min|| \begin{bmatrix} A \\ \lambda I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} ||^2$ with solution $x = (A^T A + \lambda^2 I)^{-1} A^T b$

Sensitivity or conditioning of a problem A problem maps input to output: $A, b \mapsto X = A^{-1} b$ or $(x_i, y_i) \mapsto$ weights...

If $f(x, y) = x + y$ then $f(x + \delta, y) = x + y + \delta$, the input change is δ and the output change is $|x + y + \delta - (x + y)| = \delta$

$$f(x + \delta) = f(x) + \delta \cdot f'(x) + O(\delta^2)$$

The (absolute) condition number of a function f (of an input x) is the best possible bound K of the form

$$|f(x + \delta) - f(x)| \leq K|\delta| + o(\delta)$$

with K being the **condition number**

The absolute condition number of (differentiable) f in x is $|f'(x)| = K_{abs}(f, x)$ For **multivariate functions** there are possibly different notes of change across different directions. So, the **condition number** of a function $f : R^m \to R^n$ is the best constant K that I can use to bound

$$||f(x + \delta) - f(x)|| \leq K \cdot ||\delta|| + o(||\delta||)$$

The absolute condition number K_{abs} is $K_{abs}(f, x) = \lim_{\delta \to 0} sup_{||d|| \leq \delta} \frac{||f(x+\delta) - f(x)||}{||d||} K_{abs}(f, x) = || \bigtriangledown_x f||$ 2-norm of the gradient, if f differentiable.

The **condition number measures the sensitivity of a problem to changes in input**.

Example $x \in R^2, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, f(x) = x_1 + x_2, d = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \delta \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 + \delta \\ x_2 + \delta \end{bmatrix}$

$$\frac{||f(x+\delta d) - f(x)||}{||\delta d||} = \frac{||x_1 + \delta + x_2 + \delta - x_1 - x_2||}{||\begin{bmatrix} \delta \\ \delta \end{bmatrix}||} = \frac{||2\delta||}{\sqrt{\delta^2 + \delta^2}} = \frac{2\delta}{\sqrt{2}\delta} = \sqrt{2}$$

which is the rate of change of perturbations parallel to d.
With $d = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ we would have $= 0$, and with $d = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ we would have $= 1$
Also checking other directions, it turns out that $\sqrt{2}$ is the largest possible rate of change (no coincidence that it is $|| \bigtriangledown_x f||$)

Linear Systems The condition number of solving square systems of linear equations.
$Ax = b, A \in R^{n \times n}, b \in R^n$, we perturb input b to $\overline{b} \in R^n$ and call $\overline{x}$ the solution of $A\overline{x} = \overline{b}$, while x solution of $Ax = b$.
The absolute bound is

$$||\overline{x} - x|| = ||A^{-1}\overline{b} - A^{-1}b|| = ||A^{-1}(\overline{b} - b)|| \leq ||A^{-1}|| \cdot ||\overline{b} - b||$$

and

$$||b|| = ||Ax|| \leq ||A|| \cdot ||x||$$

Putting all together gives

$$\frac{||\overline{x} - x||}{||x||} \leq ||A^{-1}|| \cdot ||A|| \cdot \frac{||\overline{b} - b||}{||b||}$$

because $K(A) = ||A|| \cdot ||A^{-1}||$ the condition number of A matrix.

What if we perturb $\overline{A} = A + \Delta$? One can prove, if $\overline{x}$ is solution of $\overline{A}\overline{x} = b$, that

$$\frac{||\overline{x} - x||}{||x||} \leq ||A|| \cdot ||A^{-1}|| \cdot \frac{||\overline{A} - A||}{||A||} + o(||\Delta||)$$

In terms of SVD we know that $||A|| = \sigma_1$. For a square invertible matrix $A = USV^T$ we have that $||A^{-1}|| = \frac{1}{\sigma_n}$, so $K(A) = ||A|| \cdot ||A^{-1}|| = \frac{\sigma_1}{\sigma_n}$.
A related quantity is the distance between A and the closest singular matrix. A matrix $B \in R^{n \times n}$ is singular if $rk(B) < n$. By the Eckhart-Young theorem, $min_{B\ sing}||A - B|| = min_{rk(B) \leq n-1}||A - B|| = \sigma_n$ because $B = \sum_{i=1}^{n-1} u_i \sigma_i v_i^T$.
Hence $\frac{1}{K(A)} = \frac{\sigma_n}{\sigma_1} = \frac{min_{B\ sing}||A-B||}{||A||}$

Least Squares Problem **Theorem**: if $A \in R^{m \times n}, m \geq n$ we define $K(A) = \frac{\sigma_1}{\sigma_n}$

Theorem The condition number of the least-squares problem $min||Ax - b||$ for a full column rank matrix $A \in R^{m \times n}, b \in R^m$

$$K_{rel, b \to x} \leq \frac{K(A)}{\cos\theta}$$

$$K_{rel, A \to x} \leq K(A) + K(A)^2 \cdot \tan\theta$$

where

$$\theta = \arccos\frac{||Ax||}{||b||}$$

Condition Number "Local" bound of the form

$$\frac{||\overline{y} - y||}{||y||} \leq k\frac{||\overline{x} - x||}{||x||}$$

for a function $y = f(x)$ and a **small** perturbation $\overline{x}$ of x, $\overline{y} = f(\overline{x})$

1.4 Floating Point Numbers

Quick recap Binary exponential notation.
Theorem $\forall\, x \in [-10^{308}, -10^{-308}] \cup [10^{-308}, 10^{308}]$ there is a double precision floating point $\overline{x}$ such that $\frac{|\overline{x}-x|}{|x|} \leq 2^{-52} \simeq 2.2 \cdot 10^{-16} = u$

Error analysis Given a function $y = f(x)$ and $x \in R$, how accurately can I compute $y = f(x)$ on a computer? In general, I can only ask the computer to compute $f(\overline{x})$ where $\overline{x} = \texttt{floor}(x)$, the closest floating point number to x. How far is $\overline{y} = f(\overline{x})$ from $y = f(x)$?

$$\frac{|\overline{y} - y|}{|y|} \leq K_{rel}(f, x) \cdot \frac{|\overline{x} - x|}{|x|} + O(u^2) \leq K_{rel}(f, x) \cdot u$$

This is called the **intrinsic error**. Whenever one makes an operation, e.g. $a+b$, the computer stores an approximation of the result which we can denote by $(a + b)(1 + \delta)$ with $|\delta| \leq u$, because

$$\frac{\overline{x} - x}{x} = \delta \Leftrightarrow \overline{x} - x = x\delta \Leftrightarrow \overline{x} = x(1 + \delta) \Leftrightarrow |d| \leq u$$

$a \oplus b = (a + b)(1 + \delta)$, the same for $\ominus$, $\otimes$ ecc: **error analysis requires keeping track of all these errors**.

Example Error analysis of the scalar product $a \cdot b$ with $a, b \in R^3$

$$y = a \cdot b = a^T b = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = a_1b_1 + a_2b_2 + a_3b_3$$

We have an algorithm on the computer that performs this, with $|\delta_i| \leq u$

$$\overline{y} = a_1 \otimes b_1 \oplus a_2 \otimes b_2 \oplus a_3 \otimes b_3 = a_1b_1(1 + \delta_1) + a_2b_2(1 + \delta_2) + a_3b_3(1 + \delta_3) =$$

$$= ((a_1b_1(1 + \delta_1) + a_2b_2(1 + \delta_2))(1 + \delta_4) + a_3b_3(1 + \delta_3))\,(1 + \delta_5) =$$

$$= a_1b_1(1 + \delta_1)(1 + \delta_4)(1 + \delta_5) + a_2b_2(1 + \delta_2)(1 + \delta_4)(1 + \delta_5) + a_3b_3(1 + \delta_3)(1 + \delta_5) =$$

$$= a_1b_1 + a_2b_2 + a_3b_3 + a_1b_1(\delta_1 + \delta_4 + \delta_5) + a_2b_2(\delta_2 + \delta_4 + \delta_5) + a_3b_3(\delta_3 + \delta_5) + O(u^2) =$$

Being $a_1b_1 + a_2b_2 + a_3b_3 = y$ we have

$$|\overline{y} - y| = |a_1b_1(\delta_1 + \delta_4 + \delta_5) + a_2b_2(\delta_2 + \delta_4 + \delta_5) + a_3b_3(\delta_3 + \delta_5)| + O(u^2)$$

$$\leq |a_1b_1| \cdot 3u + |a_2b_2| \cdot 3u + |a_3b_3| \cdot 2u + O(u^2)$$

$$\leq (|a_1b_1| + |a_2b_2| + |a_3b_3|) \cdot 3u$$

Noting that $|a_1b_1| + |a_2b_2| + |a_3b_3| \neq y$
A **trick**: $\overline{y}$ is the exact result of a scalar product with a perturbated input. We define

$\hat{b_1} = b_1(1 + \delta_1)(1 + \delta_4)(1 + \delta_5)$

$\hat{b_2} = b_2(1 + \delta_2)(1 + \delta_4)(1 + \delta_5)$

$\hat{b_3} = b_3(1 + \delta_3)(1 + \delta_5)$

This way $\overline{y} = a_1\hat{b_1} + a_2\hat{b_2} + a_3\hat{b_3}$ is the exact result of the operation on perturbated inputs.
We have $\hat{a} = a$ and $\hat{b} \mid |\hat{b_i} - b_i| \leq 3u|b_i| + O(u^2)$, so $||\hat{b} - b|| \leq 3u||b|| + O(u^2)$ which gives that $\frac{||\hat{b}-b||}{||b||} = 3u$
Now we can use the condition number bound to estimate

$$\frac{||\overline{y} - y||}{||y||} \leq K_{rel}(\cdot, b) \cdot \frac{||\hat{b} - b||}{||b||} + O(u^2)$$

Comparing it with the intrinsic error $\frac{|\overline{y}-y|}{|y|} \leq K_{rel}(\cdot, b) \cdot \frac{|\overline{x}-x|}{|x|} + O(u^2) \leq K_{rel}(\cdot, b) \cdot u + O(u^2)$ we get that the algorithm is "as good as possible", apart from the factor 3, given the intrinsic error.

Backward Stability An algorithm to compute $y = f(x)$ is called **backward stable** if it computes $\overline{y}$ such that $\overline{y} = f(\overline{x})$ for an $\overline{x}$ with $\frac{||\overline{x}-x||}{||x||} \leq O(u)$.
Usually $O(u)$ hides polynomial factors in the dimensions, e.g. $3m^2nu = O(u)$.

Theorem Backward stable algorithms are "almost optimal": they compute $\overline{y}$ with an error that is of the same order of magnitude of the (unavoidable) intrinsic error for a generic input $x \in R$.
Proof: $\frac{|\overline{y}-y|}{|y|} \leq K_{rel}(f,x) \cdot \frac{|\overline{x}-x|}{|x|} + O(u^2) \leq K_{rel}(f,x) \cdot O(u) + O(u^2)$ is only a $O(\ )$ factor away from the intrinsic bound $\frac{|\overline{y}-y|}{|y|} \leq K_{rel}(f,x) \cdot \frac{|\overline{x}-x|}{|x|} + O(u^2) \leq K_{rel}(f,x) \cdot u + O(u^2)$

Not all algorithms are backward stable: the outer product ab^T isn't. But multiplying by orthogonal matrices is backward stable. If you want an algorithm that computes $B = QA$ (for instance, the householder product), then typically you can write $\overline{B} = QA + E$ with $||E|| = ||A|| \cdot O(u)$

$$\overline{B} = QA + QQ^TE = Q(A + Q^TE)$$

with $F = Q^TE$ backward error, and

$$||F|| = ||Q^TE|| = ||E|| = ||A|| \cdot O(u)$$

With non-orthogonal Q I'd get an additional factor $K(Q) = ||Q|| \cdot ||Q^{-1}||$

The steps of the QR factorization are backward stable too, also solving least squares via QR is backward stable: it delivers $\overline{x}$ exact solution of a least squares problems with $\overline{A} = A + \Delta A, \overline{b} = b + \Delta b$ with $\frac{||\overline{A}-A||}{||A||} = O(u), \frac{||\overline{b}-b||}{||b||} = O(u)$, so it delivers an error comparable to that caused by the ill-conditioning of the problem (intrinsic error). Similarly, solving least squares problems with SVD is backward stable. Normal equations is not backward stable, of a general linear system with a poisitive definite C

$C = A^TA$

$d = A^Tb$

$x = C \backslash d$

A^T is not orthogonal. Even if the first two steps are perfectly accurate, the third causes error. $C = A^TA \Rightarrow K(C) = \frac{\sigma_1^2}{\sigma_n^2} = K(A)^2) \Rightarrow K_{LS,A \to x} = K(A) + K(A)^2 \cdot \tan\theta$
This may be larger than the condition number of the problem, especially for $\theta \simeq 0$.

Residuals and a-posteriori stability checks Let us start from square linear systems $Ax = b$. Suppose we have a computed $\overline{x}$ that (approximately) solves the problem. How low can $||r|| = ||Ax - b||$ be? Backward stable algorithms usually get residuals of the order of u. If I have a backward stable algorithm to solve $Ax = b$ (square linear system), then $\overline{x}$ is the exact solution of $\overline{A}\overline{x} = \hat{b}$, i.e. $(A + E)\overline{x} = b + f$ with $\frac{||E||}{||A||}, \frac{||f||}{||b||} = O(u)$, so $A\overline{x} + E\overline{x} = b + f$ and $||b|| = ||Ax|| \leq ||A|| \cdot ||x||$

$$||A\overline{x} - b|| = ||f - E\overline{x}|| \leq ||f|| + ||E|| \cdot ||\overline{x}|| = O(u) \cdot [||b|| + ||A|| \cdot ||x||] = O(u) \cdot ||A|| \cdot ||x||$$

$$\frac{||A\overline{x} - b||}{||A|| \cdot ||x||} = O(u)$$

And typically $||A|| \cdot ||x|| \simeq ||b||$
So with backward stable algorithms we have **small** (relative) **residuals** $O(u)$.
Suppose $\overline{x}$ computed somehow has a small $||r|| = ||A\overline{x} - b||$. Is this a quarantee for $\frac{||\overline{x}-x||}{||x||}$ to be small?

Theorem : A square invertible, x exact solution of $Ax = b$ and $\overline{x}$ another vector with $r = A\overline{x} - b$, then

$$\frac{||\overline{x} - x||}{||x||} \leq K(A) \cdot \frac{||r||}{||b||}$$

Proof: $\overline{x}$ is the exact solution of $A\overline{x} = b + r$ and the condition number bound gives $\frac{||\overline{x}-x||}{||x||} \leq K(A)\frac{||(b+r)-b||}{||b||}$
We can ask the same question for LS problems: when is small? Not r in general. The gradient of $f(x) = ||Ax - b||^2$ is though

$$\nabla f(x) = 2A^TAx - 2A^Tb = 2 \cdot (\text{residual of normal equations}) = 2r$$

$\frac{||\overline{x}-x||}{||x||} \leq K(A^TA) = K(A)^2$, and the relative error bound on $A^TAx = A^Tb$ is $\frac{||r||}{||A^Tb||}$, possibly larger than the condition number of the least squares problem.
Recalling that with $A = Q_1R_1$ QR factorization, $||Ax-b|| = ||\begin{bmatrix} Q_1^T(Ax-b) \\ Q_2^T(Ax-b) \end{bmatrix}||$ we have a better error bound

theorem: let $r_1 = Q_1^T(A\overline{x}-b)$ and x the exact solution of $min||Ax-b||$, then $\frac{||\overline{x}-x||}{||x||} \leq R_{rel,b\to x} \cdot \frac{||r_1||}{||b||}$

Proof: replay the solution of the LS problems via QR with $\overline{x}$ instead of x and $\hat{b} = b + Q_1r_1$ in place of b: `todo`

1.5 Algorithms for square linear systems $Ax = b$

1.5.1 Gaussian Elimination

Complexity $\frac{2}{3}n^3$ (half of QR)

Storage problem $n \times n$ matrix, for example with $n = 10^5$ it requires 80GB of RAM. In real life many matrices are sparse, $\simeq$3-10 nonzero elements per row or something like that.
Sparse matrices are stored as lists of nonzeros.

Idea Use factorization of A into simpler pieces: $A = QR$ or $A = USV^T$ for example. These can be used to solve linear systems, and the factorization can be reused if you have many systems with different right-hand sides.

Gaussian elimination $\leftrightarrow$ LU factorization, L lower triangular and U upper triangular matrices. $A = LU$ is $\frac{2}{3}n^3$, but $Ax = b \Leftrightarrow LUx = b \Leftrightarrow \begin{cases} Ly = b \\ Ux = y \end{cases}$ is $2n^2$
Stability can be a problem. Error analysis with $\overline{L}\,\overline{U} = \overline{A}$, we have $||\overline{A} - A|| \leq O(u) \cdot ||L|| \cdot ||U||$, so all fine if $||L||$ and $||U||$ are not too larger than $||A||$. A fix can be using partial pivoting: at step k of LU, swap rows k and $p||b_p| = max|b_i|$ for $i = k, k+1, \ldots, n$
Swapping rows is equivalent to multiplying by P_k, and I with the k and p rows swapped.

$$P_{n-1} \ldots P_1A = L_1^{-1} \ldots L_{n-1}^{-1}U$$

$$PA = LU$$

$$A = P^TLU$$

With an overhead of $O(n^2)$ swaps and comparisons.

Is LU plus pivoting stable? If you ensure at step k that $|b_k| \geq |b_i|$ for all $i > k$, then $|L_{i,k}| = \left|\frac{b_i}{b_k}\right| \leq 1 \Rightarrow ||L||$ stays bounded. However, in the worst case, $\frac{||U||}{||A||} = 2^n$ (exponentially larger). The average case is that $\frac{||U||}{||A||}$ bounded polynomially.

Another problem With sparse matrices, the gaussian elimination often destroys sparsity structure.

1.5.2 Symmetric Gaussian Elimination

Even if A is symmetric, L_2A is not symmetric but $L_1AL_1^T$ is.
After $n-1$ steps we have

$$L_{n-1} \ldots L_1AL_1^T \ldots L_{n-1}^T = D$$

$$A = L_{n-1}^{-1} \ldots L_1^{-1}D(L_{n-1}^T)^{-1} \ldots (L_1^T)^{-1}$$

$$A = LDL^T$$

With L lower triangular with 1 on the diagonal, D diagonal and L^T transpose of L. This is called **LDL factorization**.

Theorem For any $A = A^T$, *if we do not encounter zero pivots* we can find $L, D \mid A = LDL^T$ with L lower triangular with 1 on the diagonal and D diagonal.
Small pivots, even non-zero, brings instability. We will need pivoting (row exchanges). To preserve symmetry, one needs to exchange not only rows but columns too: PAP^T swaps both rows and columns. Unfortunately this is still not enough to ensure large pivots. To avoid this, one needs to allow for 2×2 blocks in D
(L, D) has half the storage costs of (L, U). Also the computational cost is halved: $\frac{1}{3}n^3$ versus $\frac{2}{3}n^3$ for LU, this because we operate on only the lower triangular parts given that $A_{k+1:n,k+1:n} = A_{k+1:n,k+1:n} - L_{k+1:n,k} \cdot A_{k,k+1:n}$ involves only symmetric elements.

Things are better if A is positive definite. Remember that $A = A^T$ and positive definite means $x^T Ax > 0$ for all $x \in R^n, x \neq 0 \Leftrightarrow$ all eigenvalues of A are > 0.

Lemma If $A \in R^{n \times n}$ is positive definite, we have two properties:

$A_{k,k} > 0$ for $k = 1, \ldots, n$

For all invertible $M \in R^{n \times n}$, MAM^T is also positive definite

Let's prove this:

$A_{k,k} = e_k^T A e_k > 0$ with e_k array of zeroes with 1 in k

$x^T MAM^T x = y^T Ay > 0$ for all $x \in R^n, x \neq 0 \Rightarrow y = M^T x \neq 0$

Hence, at each step $L_k \ldots L_1 A L_1^T \ldots L_k^T$ is positive definite and its diagonal entries are $> 0 \Rightarrow$ all pivots are > 0. So for a positive definite A, LDL^T can be performed without breakdown even without row and columns exchanges. In addition, $d_{k,k} > 0$ for every k.

1.5.3 Cholesky factorization

$$D = \begin{bmatrix} d_{1,1} & & 0 \\ & \ddots & \\ 0 & & d_{n,n} \end{bmatrix} = \begin{bmatrix} \sqrt{d_{1,1}} & & 0 \\ & \ddots & \\ 0 & & \sqrt{d_{n,n}} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{d_{1,1}} & & 0 \\ & \ddots & \\ 0 & & \sqrt{d_{n,n}} \end{bmatrix} = D^{\frac{1}{2}} D^{\frac{1}{2}}$$

$A = LDL^T = LD^{\frac{1}{2}} D^{\frac{1}{2}} L^T = R^T R$ with R upper triangular.
Every $A \succ 0$ can be written as $A = R^T R$ where R is upper triangular.
Cholensky and LDL for positive definite matrices are stable even without pivoting $||R|| = ||A||^{\frac{1}{2}}$

1.5.4 Algorithms for solving linear systems

Symmetric Positive Definite $\Rightarrow$ Cholensky, $\frac{1}{3}n^3$

Symmetric $\Rightarrow (P^T)LDL^T(P)$, $\frac{1}{3}n^3$

General $\Rightarrow LU(P)$, $\frac{2}{3}n^3$

All of them come with sparse variants, with "list of non-zeroes"-based storage. **Problem**, fill-in: LU may be much less sparse than A, and at some point time/space becomes scarce.

1.5.5 Iterative Methods

$x^{(1)}, x^{(2)}, x^{(3)}, \ldots$ sequence of vectors that converges to the solution x of $Ax = b$
Black-box methods: they only need a black-box function $\lambda_0 v = Av$ that computer Av given v. Fast whenever computing Av for a given v is fast, $O(nnz)$ for a sparse matrix A with nnz non-zeroes.

Idea If A is positive definite, solving $Ax = b$ is finding $min\, \frac{1}{2}x^T Ax - b^T x$ with $f(x) = \frac{1}{2}x^T Ax - b^T x$ strongly convex.
$\nabla f(x) = Ax - b$
Most gradient-based algorithms produce iterates that belong to the same subspace, $x_0 = 0$

$\nabla f(x_0) = -b$, $x_1 = x_0 + \alpha d =$ multiple of b

$\nabla f(x_1) = Ax_1 - b =$ linear combination of Ab and b, and $x_2 = x_1 + \alpha \nabla f(x_1) =$ linear combination of Ab and b

$\nabla f(x_2) = Ax_2 - b =$ linear combination of A^2b, Ab and b, and $x_3 = x_2 + \alpha\nabla f(x_2) =$ linear combination of A^2b, Ab and b

$\Rightarrow$ all vectors we construct up to stop k belong to $span(b, Ab, \ldots, A^{k-1}b)$ of k different vectors

$$span(b, Ab, \ldots, A^{k-1}b) = \{v = \alpha_0 b + \alpha_1 Ab + \alpha_2 A^2 b + \ldots + \alpha_{k-1}A^{k-1}b \mid \alpha_i \in R\} =$$
$$= \{v = (\alpha_0 I + \alpha_1 A + \alpha_2 A^2 + \ldots + \alpha_{k-1}A^{k-1})b \mid \alpha_i \in R\} =$$
$$= \{v = p(A)b \mid \text{for all polynomials } p(t) = \alpha_0 + \alpha_1 t + \ldots + a_{k-1}t^{k-1} \text{ of degree } < k\}$$

Krylov Subspace This is the Krylov Subspace: $K_k(A, b) \Rightarrow$ **Krylov space methods**
For now we assume that $b, Ab, \ldots, A^{n-1}b$ are linearly independent.
$K_n(A, b)$ is a linear subspace means that $v, w \in K_n(A, b) \Rightarrow \alpha v + \beta w \in K_n(A, b)$
$v \in K_n(A, b) \Rightarrow v = (\alpha_0 I + \alpha_1 A + \ldots + \alpha_{n-1}A^{n-1})b$ with $\alpha_{n-1} \neq 0 \Rightarrow Av = (\alpha_0 A + \alpha_1 A^2 + \ldots + \alpha_n A^n)b \in K_{n+1}(A, b)$ and also $v \in K_n(A, b) - K_{n-1}(A, b) \Rightarrow Av \in K_{n+1}(A, b) - K_n(A, b)$
By composing b, linear combinations of already computed vectors and products of already computed vectors times A, one only gets vectors in $K_n(A, b)$ where $n - 1$ is the number of items you combine (3).
Can we look for the best approximation of the solution of a certain problem, for instance a linear system $Ax = b$, inside $K_n(A, b)$?

$$K_n(A, b) = \{\alpha_0 b + \ldots + \alpha_{n-1}A^{n-1}b \mid \alpha_i \in R\} =$$
$$= \left\{ \left[\ b \mid Ab \mid A^2b \mid \ldots \mid A^{n-1}b\ \right] \cdot \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \end{bmatrix} \mid \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \end{bmatrix} \in R^n \right\} = \{V_n \cdot y \mid y \in R^n\}$$

So each vector in $K_n(A, b)$ is written as $V_n y$ for some $y \in R^n$ and the problem becomes a least squares problem with the matrix $AV_n \in R^{m \times n}$:

$$\min_{x_n \in K_n(A,b)} ||Ax_n - b|| = \min_{y \in R^n} ||(AV_n)y - b||$$

But this problem is very unstable to solve because $V_n = \left[\ b \mid Ab \mid A^2b \mid \ldots \mid A^{n-1}b\ \right]$ has columns that tend to be all multiples of the same vector as n increases.

Arnoldi Algorithm Algorithm to compute an orthonormal basis of the Krylov subspace $K_n(A, b)$
The idea is that given the vectors $q_1, \ldots, q_j$ that are an orthonormal basis of $K_j(A, b)$, we want to compute one more vector q_{j+1} that extends it to an orthonormal basis of $K_{j+1}(A, b)$.
The base step is that an orthonormal basis of $span(b)$ is $q_1 = \frac{1}{||b||} \cdot b$, and the generic step is $j \to j + 1$. The additional invariant is that the last vector q_j satisfies $q_j = p(A)b$ with a p of degree exactly $j - 1$.

1. Generate a vector $\in K_{j+1}$ that wasn't already in K_j

   $$w = Aq_j = q_1\beta_1 + \ldots + q_j\beta_j + q_{j+q}\beta_{j+1}$$

   The β_i are the coordinates of w in the basis $q_1, \ldots, q_{j+1}$

2. Subtract q_1 component, as the q_i are orthonormal

   $$q_1^T w = q_1^T q_1\beta_1 + \ldots + q_1^T q_{j+1}\beta_{j+1} = \beta_1$$

   $$w = w - q_1\beta_1 = q_2\beta_2 + \ldots + q_{j+1}\beta_{j+1}$$

3. Repeat!

   $$q_i^T w = q_i^T q_i\beta_i + q_i^T q_{i+1}\beta_{i+1} + \ldots + q_1^T q_{j+1}\beta_{j+1} = \beta_i$$
   $$w = w - q_i\beta_i = q_{i+1}\beta_{i+1} + \ldots + q_{j+1}\beta_{j+1}$$

4. After j steps, $w = \beta_{j+1}q_{j+1}$, and q_{j+1} must have norm 1 so set $q_{j+1} = \frac{1}{||w||}w$ and $\beta_{j+1} = ||w||$

Factorization For $j = 1, 2, \ldots, n$ we have written $w = Aq_j = q_1\beta_{1,j} + \ldots + q_j\beta_{j,j} + q_{j+q}\beta_{j+1,j} = Q \begin{bmatrix} \beta_{1,j} \\ \vdots \\ \beta_{j+1,j} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

and writing them down one next to the other $A[q_1 \ \ldots \ q_n] = [q_1 \ \ldots \ q_{n+1}]$ so $AQ_n = Q_{n+1}\underline{H}_n$ for some matrix $\underline{H}_n \in R^{(n+1)\times n}$ that contains the coefficients $\beta_{i,j} = (\underline{H}_n)_{i,j}$

GMRES Generalized Minimum Residual: after computing n steps of Arnoldi, $AQ_n = Q_{n+1}\underline{H}_n$, we look for the vector $x_n \in K_n(A, b)$ that minimizes the residual $||Ax_n - b||$ with $x_n = Q_n y$.
I.e. $\min_{x_n \in K_n(A,b)} ||Ax_n - b|| = \min_{y \in R^n} ||AQ_n y - b||$, the least squares problem wit the matrix $AQ_n \in R^{m\times n}$, in $O(mn^2)$
$= ||\underline{H}_n y - e_1||b||\,||$, the least squares problem with matrix $H_n \in R^{(n+1)\times n}$, in $O(n^3)$

Complexity of Arnoldi n multiplications $v \mapsto Av$, worst case $O(m^2) \cdot n$ but could be $O(m) \cdot n$ if sparsity is involved, and $O(mn^2)$ for the orthogonalization inner loop. Total complexity is $O(mn^2 + n \cdot \text{matvec})$ with matvec being the complexity of $v \mapsto Av$.

Capitolo 2

Optimization

Making sense of the huge amounts of data generated and collected means taking something big and unwieldy and producing something small and nimble that can be used: a **mathematical model**. It should be: accurate (describes well the process), computationally inexpensive (fast), general (can be applied to many different processes). Typically impossible to have all three. Developing general models is useful, work once apply many. But the shape of the model controls the computational cost. How to get accuracy for any given application? A **model is parametric** and must **learn the right values of the parameters**. In other words, **fitting**: within the family of (usually) infinitely many models with the given shape, find the one that better represent your phenomenon. This is an optimization problem, and solving fitting is typically the bottleneck.

Fitting means minimizing training error, while machine learning means minimizing the generalization error.

Example: Linear estimation Phenomenon measured by one number y and believed to depend on a vector $x = [x_1, \dots, x_n]$. Available set of observations $(x_1, y_i), \dots, (x_m, y_m)$.
And optimistic assumption: the dependence is linear, hence $y = \sum_{i=1}^{n} w_i x_i + w_0 = w_x + w_0$ for fixed $n+1$ real parameters $w = [w_0, w_+ = [w_1, \dots, w_n]]$ and find the w for which is less untrue $\min_w \ L(w) = ||y - xw||$

Example: Low-rank approximation A large, sparse matrix $M \in R^{n \times m}$ describes a phenomenon depending on pairs. Find a tall and thin $A \in R^{n \times k}$ and a fat and large $B \in R^{k \times m}$, meaning $k << n, m$, such that $A \cdot B = M$ with $\min_{A,B} L(A, B) = ||M - AB||$
A, B can be obtained from eigenvectors of M^T and MM^T, but possibly huge and dense matrix. Efficiently solving this problem requires

Low complexity computation

Avoiding ever explicitly forming $M^T M$ and MM^T (too much memory)

Exploiting the structure of M (sparsity, similar columns...)

Ensuring that the solution is numerically stable.

Example: Support Vector Machines Same setting as the first example, but $y_h \in \{1, -1\}$: want to linearly separate the two sets. Which separating hyperplane to choose? Intuitively, base on the margin: more margin, more robust classification.
The distance of parallel hyperplanes (w_+, w_0) and (w_+, w_0') is $\frac{|w_0 - w_0'|}{||w_+||}$. We can always take the hyperplane in the middle and scale w: $w_+ x_h + w_0 \geq 1$ if $y_h = 1$, $w_+ x_h + w_0 \leq 1$ if $y_h = -$
The **maximum margin separating hyperplane** is the solution of $min_w \{||w_+||^2 \, | \, y_h(w_+ x_h + w_0) \geq 1) \, h = 1, \dots, m\}$, and the margin is $\frac{2}{||w_+||}$ assuming any exists.
If it doesn't exists, soft-margin SVM: $\min_w ||w_+||^2 + C \cdot L(w) = \sum_{h=1}^{m} \ max(1 - y_h(w_+ x_h + w_0), 0)$, C **weight loss**
Non differentiable, reformulation $\min_{w,\xi} ||w_+||^2 + C \sum_{h=1}^{m} \xi_h$

2.1 Optimization problems

Given X *any* set and $f : X \to R$ *any* function, we have an optimization problem in the form

$$(P) \quad f_* = min\{f(x) : x \in X\}$$

with X **feasible region** and f **objective function**.

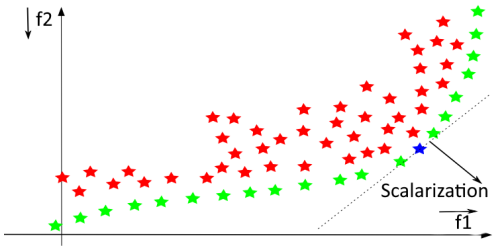
$$min\{f(x) : x \in X\} = -max\{-f(x) : x \in X\}$$

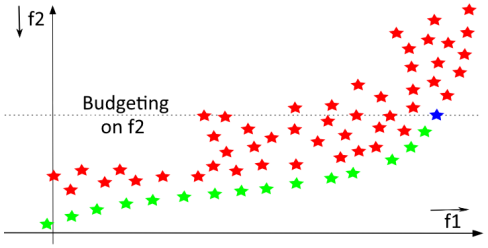
$x \in X$ is the **feasible solution**, often $X \subset F, x \in F - X$ **unfeasible solution**. We want (any) optimal solution $x_* \in X \mid \forall\, x \in X \;\; f(x_*) \leq f(x)$ and we don't care if $\exists\, x' \neq x_* \mid f(x') = f(x_*)$: **all optimal solutions are** $\equiv$. $f_* = f(x_*) = v(P)$ is the **optimal value** (which is **unique**) if x_* exists, which it may not.

Multi-objective optimization Often we have multiple objective function, and often they have incomparable values (for example, loss and regularization in machine learning). There are two practical solutions:

Scalarization, maximize one of the losses compared to the others: $min\{f_1(x) + \alpha f_2(x) \mid x \in X\}$, but which α?

Budgeting: bound the value of one of the losses $min\{f_1(x) \mid f_2(x) \leq \beta_2, x \in X\}$, but which β_2?





We will assume that this is done at modelling stage.

2.1.1 Optimization is hard

Even with single-objective, optimization is hard. It's impossible if f has no minimum in X, for example $f(x) = x$: if (P) is unbounded below, then $v(P) = -\infty$. Solving (P) is actually at least two different things:

finding x_* and **proving** x_* **it's optimal** (how?)

constructively **proving** f **unbounded below** on X (how?)

It's also impossible if $f_* > -\infty$ but $\nexists\, x_*$, for example $f(x) = e^x$ or $f(x) = \begin{cases} 1 & \text{if } x = 0 \\ |x| & \text{if } x \neq 0 \end{cases}$, but there are plenty of ϵ-approximate solutions (ϵ-optima)

$$f(x_\epsilon) \leq f_* + \epsilon \quad \forall\, \epsilon > 0$$

and on computers $x \in R$ is actually $x \in Q$ with up to 16 digits precision, so approximation errors are unavoidable anyway. Exact algebraic computation is possible but too slow, so ML is actually going the opposite way (float, half, small integers...). And anyway, finding the exact x_* is impossible in general.

Optimization need to be approximate

Absolute gap: $\{a_i = A(x_i) = f(x_i) - f_*\}$ so $A(x) = f(x) - f_* \;(\geq 0)$

Relative gap: $\{r_i = R(x_i) = \frac{f(x_i) - f_*}{|f_*|} = \frac{A(x_i)}{|f_*|}\}$ so $R(x) = \frac{f(x) - f_*}{|f_*|} = \frac{A(x)}{|f_*|} \;(\geq 0)$

The relative gap is useful because $\forall\, \alpha > 0$ we have that $(P) \equiv (P_\alpha) = min\{\alpha f(x) \mid x \in X\}$, and for the same x_* we have $v(P_\alpha) = \alpha v(P) \Rightarrow$ same $R(x)$, different $A(x)$
But in general computing the absolute/relative gap is hard because we don't know f_*, which is what we want to estimate. So it's hard to estimate how good a solution is. Could argue that this is the "issue" in optimization: compute an estimate of f_*.

Optimization is really hard Impossible, even, because isolated minima can be anywhere, and restricting to $x \in X = [x_-, x_+]$ with $-\infty < x_- < x_+ < +\infty$ doesn't help: still uncountable many points to try. Also f can have isolated downward spike anywhere. Even on $X = [x_-, x_+]$ the spikes can be arbitrarily narrow.

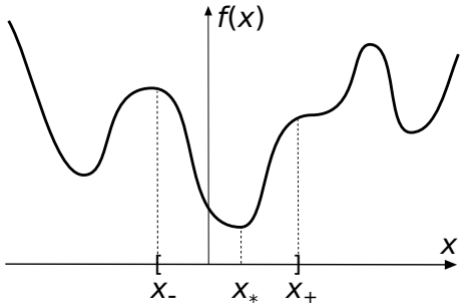
Optimization at least possible We can impose $X = [x_-, x_+]$ with $D = x_+ - x_- < \infty$, meaning with a fixed finite diameter. We can also impose that the f's spikes can't be arbitrarily narrow, so f cannot change too fast $\equiv$ f Lipschitz continuous (L-c) on X:

$$\exists\, L > 0 \mid \ |f(x) - f(y)| \leq L|x - y| \ \ \forall x, y \in X$$

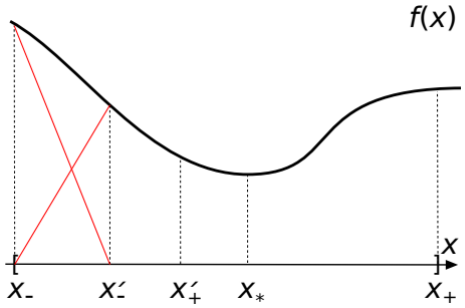
f L-c $\Rightarrow$ doesn't "jump" and one ϵ-optimum can be found with $O(\frac{LD}{\epsilon})$ evaluations by uniformly sampling X with step $\frac{2\epsilon}{L}$. There's a bad news: no algorithm can work in less that $\Omega(\frac{LD}{\epsilon})$, but it's the worst case of f (constant with one spike).
The number of steps is inversely proportional to accuracy: just not doable for small ϵ. Dramatically worse with $X \subset R^n$.
Also generally L is unknown and not easy to estimate, but algorithms actually require/use it.

2.1.2 Local Optimization

Even if I stumble in x_* how do I recognize it? This is the difficult thing. It's simpler to start with a weaker condition: x_* is the local minimum if it solves $min\{f(x) \mid f \in X(x_*, \epsilon) = [x_* - \epsilon, x_* + \epsilon]\}$ for some $\epsilon > 0$.
Stronger notion: **strict** local minimum if $f(x_*) < f(y) \ \ \forall\, y \in X(x_*, \epsilon) - \{x_*\}$
f (strictly) **unimodal** on X if has minimum $x_* \in X$ and it is (strictly) decreasing on the left $[x_-, x_*]$ and (strictly) increasing on the right $[x_*, x_+]$. If x_*, then typically $\exists\, \epsilon >= \mid f$ is (strictly) unimodal on $X(x_*, \epsilon)$ Most functions are not unimodal, but they are if you focus on the attraction basin of x_* and restrict there. Unfortunately it's true for every local optimum, they all look the same.



Once in the attraction basin, we can restrict it by evaluating f in two points and excluding a part.



How to choose the part so that the algorithm go as fast as possible? Each iteration dumps the left or the right part, don't know which $\Rightarrow$ should be equal $\Rightarrow$ select $r \in (\frac{1}{2}, 1), x'_- = x'_- + (1-r)D, x'_+ = x_- + rD$
Faster if r larger $\Rightarrow r = \frac{D}{2} + \epsilon = x'_\pm = x_- + \frac{D}{2} \pm \epsilon$ but next iteration will have two entirely different x'_-, x'_+ to evaluate f on.

Optimally choosing the iterates A generally powerful concept is to optimize the worst-case behavior $\Rightarrow$ shrink the intervals as quickly as possible.
Each iteration dumps either $[x_-, x'_-]$ or $[x'_+, x_+]$, we don't know which so they should be of equal size $\Rightarrow$ select $r \in (\frac{1}{2}, 1)$ so that $x'_- = x_- + (1-r)D$ and $x'_+ = x_- + rD$
r larger $\Rightarrow$ faster convergence, so $r = \frac{D}{2} + \epsilon \Leftrightarrow x'_\pm = x_- + \frac{D}{2} \pm \epsilon$ but next iteration will have two entirely different x'_-, x'_+ to evaluate f on.
So we actually want to minimize function evaluations by reusing the surviving point.

$$r : 1 = (1-r) : r \Leftrightarrow r \cdot r = 1 - r \Leftrightarrow r = \frac{\sqrt{5}-1}{2} = 0.618 = \frac{1}{g}$$

with g being the golden ratio, $g = \frac{\sqrt{5}+1}{2} = 1.618 \Rightarrow g = 1 + r = 1 + \frac{1}{g}$
Theorems breed algorithms: **golden ratio search**

```
procedure x = GRS(f, xl, xr, delta)
  xl2 = xl + (1-r)(xr-xl)
  xr2 = xl + r(xr - xl)
  compute f(xl2), f(xr2)
  while (xr - xl > delta):
    if(f(xl2) > f (xr2))
      xl = xl2
      xl2 = x
      x = xr2
      xr2 = xl + r(xr - xl)
      compute f(xr2)
    else:
      xr = xr2
      xr2 = x
      x = xl2
      xl2 = xl + (1-r)(xr-xl)
      compute f(xl2)
```

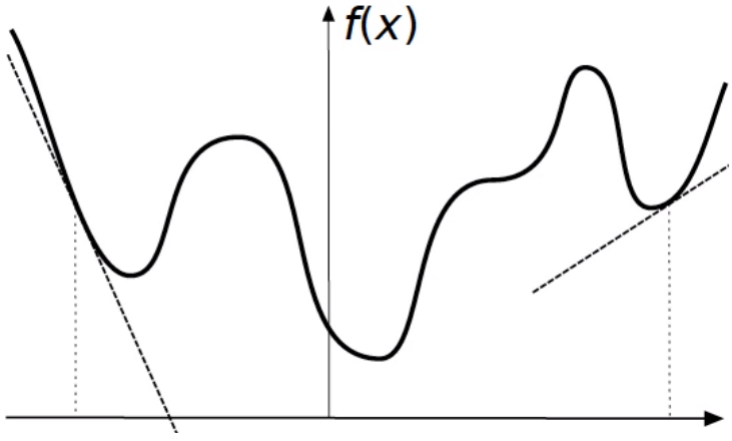
After k iterations, $x_+^k - x_-^k = Dr^k$ stops when $Dr^k \leq \delta$, so when $k = 2\log\frac{D}{\delta}$: exponentially faster, can work with small δ.
Asymptotically optimal if no other information is available. $\delta \neq \epsilon$ but f L-c $\Rightarrow A(x^k) \leq \epsilon$ when $k = 2\log\frac{LD}{\epsilon}$
First example of linear convergence $A(x^k) \leq Sr^k \leq \epsilon$ with $r < 1$, as fast as a negative exponential $\Rightarrow k \geq \frac{\log\frac{S}{\epsilon}}{\log\frac{1}{r}}$

$O(\log(\frac{1}{\epsilon}))$ good, but the constant $\to \infty$ as $r \to 1$

To make it go faster, give it more information Two points are needed to see in which direction f is decreasing. If we could see this directly we could make it with one point, faster. Look at the linear function that best locally approximates f, trusty old first derivative $f'(x)$: slope of the tangent line to the graph of f in x
First order model of f at x:

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

$L_x(y) \simeq f(y) \quad \forall\, y \in [x - \epsilon, x + \epsilon]$ for some small $\epsilon > 0$.
x_* local minimum $\Rightarrow f'(x_*) = 0 \Leftrightarrow$ root of $f' \Leftrightarrow$ stationary point. If $f'(x) < 0$ or $f'(x) > 0$, then x is clearly not a local minimum. Hence, $f'(x) = 0$ for the all local minima (hence in the global minimum as well) but this is true for the local maxima (hence global maximum as well), as well in the plateau and saddle points. To tell them apart, look at the second derivative f''.



In simple cases we get the answer by a closed formula. In $f = bx + c$ linear, if $b > 0$ then the minimum is x_- and maximum is x_+, viceversa if $b < 0$. For $f = ax^2+bx+c$, quadratic, then if $a > 0$ the minimum is $\min\{x_+, max\{x_*, x_-\}\}$ and the maximum is $\text{argmax}\{f(x_-), f(x_+)\}$, and viceversa if $a < 0$.
Only polynomial whose root have a closed formula (degree 3 and some degree 4), with basically no hope for most trascendental, trigonometric and mixed equations. We need an algorithm for solving non-linear equations.

Dichotomic Search f' continuous and the intermediate value theorem gives that

$$f'(x_-) < 0 \wedge f'(x_+) > 0 \Rightarrow \exists\, x \in [x_-, x_+] \mid f'(x) = 0$$

Theorems breed algorithms $\rightarrow$ **Dichotomic Search**.

```
procedure x = DS(f, xl, xr, eps)
  while (true) do:  # invariant: df(xl)<-eps, df(xr)>eps
    x = in_middle_of(xl, xr)
    compute df(x)
    if (abs(df(x)) <= eps): break
    if (df(x) < 0):
      xl = x
    else:
      xr = x
```

With `df` meaning f'
For `in_middle_of(xl, xr)` the obvious choice is `return (xl + xr)/2;`. We have linear convergence with $\gamma = 0.5 < 0.618 \Rightarrow k = 1.45 \log(\frac{LD}{\epsilon}) < 2 \log(\frac{LD}{\epsilon})$
The condition $f'(x_-) < -\epsilon, f'(x_+) > \epsilon$ is important. What if is not satisfied? Obvious solution, moving the interval more and more to the right until the derivative is possible. The same in reverse of x_- with $\Delta x = -1$. This works in practice for all "reasonable" functions. Works if f coercive ($\lim_{|x| \to \infty} f(x) = \infty$)

$f' \in C^0 \Leftrightarrow f \in C^1 \Leftrightarrow$ continuously differentiable $\Rightarrow f \in C^0$

$f'' \in C^0 \Leftrightarrow f \in C^2 \Leftrightarrow f' \in C^1 \Rightarrow f' \in C^0 \Rightarrow f \in C^1 \Rightarrow f \in C^0$

$f \in C^1$ globally L-c on $X \Rightarrow |f'(x)| \leq L \forall\, x \in X$

Extreme value theorem $f \in C^0$ on $X = [x_-, x_+]$ finite $\Rightarrow \max\{f(x) \mid x \in X\} < \infty$, $\min\{f(x) \mid x \in X\} > -\infty$

$f \in C^1$ on X finite $\Rightarrow f$ globally L-c on X
Best possible case is $f \in C^2$ on finite $X \Rightarrow$ both f and f' globally L-c on X

Fastest local optimization Interpolation, for improving the dichotomic search. Choosing x "right in the middle" is the dumbest possible approach, because we know a lot about f: $f(x_-), f(x_+), f'(x_-), f'(x_+) \ldots$. So let's use that, by constructing a model of f based on known information. Much better choosing x close to x_*.
But remember that **the model is an estimate**, so never completely trust the model, but regularize, stabilize... in this case, the minimum guaranteed decrease is with $\sigma < 0.5$, and the worst case is linear convergence with $r = 1 - \sigma$, but hopefully is much faster than that when the model is "right".

2.1.3 Measuring algorithms speed

Given the sequences

$\{x_i\}$

$\{d_i = |x_i - x_*|\}$

$\{f_i = f(x_i)\}$

$\{a_i = A(x_i) = f(x_i) - f_*\}$ absolute gap

$\{r_i = R(x_i) = \frac{f(x_i)-f(x)}{|f_*|})\frac{A(x_i)}{|f_*|}\}$ relative gap

We have convergence when $\{a_i\} \to 0, \{r_i\} \to 0 \Leftarrow \{d_i\} \to 0$ (but $\not\Rightarrow$), but how rapidly? **Rate of convergence**

$$\lim_{i\to\infty}\left(\frac{f_{i+1}-f_*}{f_i-f_*}\right)^p = \lim_{i\to\infty}\left(\frac{a_{i+1}}{a_i}\right)^p = \lim_{i\to\infty}\left(\frac{r_{i+1}}{r_i}\right)^p = r$$

$p = 1$ $r = 1 \Rightarrow$ sublinear

$\frac{1}{i} \Rightarrow k \in O(\frac{1}{\epsilon})$ (bad)

$\frac{1}{i^2} \Rightarrow k \in O(\frac{1}{\sqrt{\epsilon}})$ (a bit better)

$\frac{1}{\sqrt{i}} \Rightarrow k \in O(\frac{1}{\epsilon^2})$ (horrible)

$r < 1 \Rightarrow$ linear, $r_i \to i \in O(\log(\frac{1}{\epsilon})$, good unless $r = 1$

$p \in (1, 2)$ $p = 1, r = 0 \Rightarrow$ superlinear

$p = 2$ $r > 0 \Rightarrow$ quadratic, best we can reasonably hope for
$\frac{1}{2^{2^i}} \Rightarrow i \in O(\log(\log(\frac{1}{\epsilon})))$, which is basically $O(1)$: the number of correct digits double at each iteration

Improving dichotomic search Quadratic interpolation has superlinear convergence if started "close enough".
$f \in C^3, f'(x_*) = 0 \wedge f''(x_*) \neq 0 \Rightarrow \exists\, \delta > 0 \mid x_0 \in [x_* - \delta, x_* + \delta] \Rightarrow \{x_i\} \to x_*$ with $p = \frac{1+\sqrt{5}}{2}$ exponent of superlinear convergence.
x_0 is the starting point of the algorithm.

Four conditions $\Rightarrow$ can fit a cubic polynomial and use its minima. Theoretically pays: quadratic convergence ($p = 2$) and seems to work well in practice.

Newton's method More derivatives, so same information with less points. First order model of f' at x_i

$$L_i'(x) = L_{x_i}'(x) = f'(x_i) + f''(x_i)(x - x_i) \simeq f'(x)$$

and solve $L_i'(x) = 0 \simeq f'(x) = 0 \Rightarrow x = x_i - \frac{f'(x_i)}{f''(x_i)}$

```
procedure x = NM(f, x, eps)
  while (abs(df(x)) > eps):
    x = x - (df(x)/ddf(x))
```

With `df` meaning f' and `ddf` meaning f''.
Alternatively construct a second order model

$$Q_i(x) = Q_{x^i}(x) = f(x^i) + f'(x^i)(x - x^i) + f''(x^i)\frac{(x - x^i)^2}{2}$$

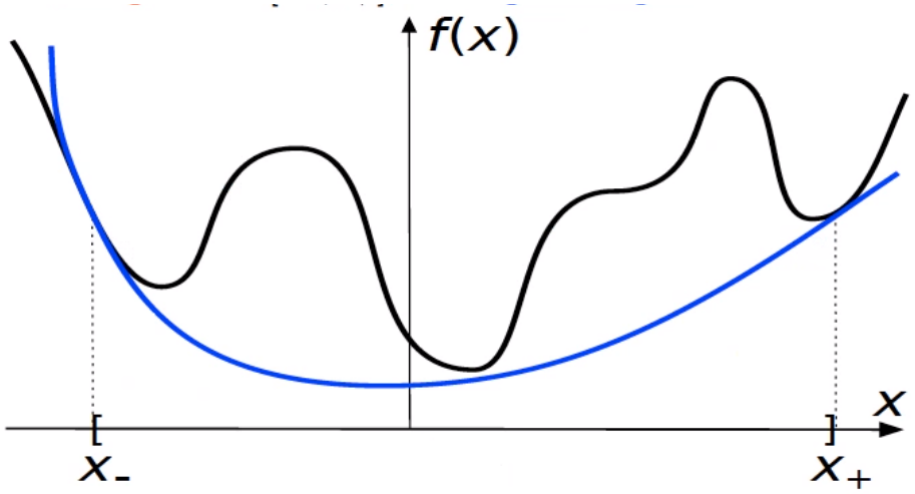
and then minimize it.

Numerically delicate: what if $f''(x) \simeq 0$? Converges (at all) only if started close enough to x_*.
If we get within a short enough distance δ, then it will converges extremely fast with $p = 2$. Mathematically, $f \in C^3, f'(x_*) = 0 \wedge f''(x_*) \neq 0 \Rightarrow \exists\, \delta > 0 \mid x_0 \in [x_* - \delta, x_* + \delta] \Rightarrow \{x_i\} \to x_*$ with $p = 2$.

2.1.4 Global optimization

Unless strong assumptions are made, we can't say much about global optimization.
The obvious one would be unimodal, but not easy to verify/construct. Workable alternative: f convex ($\Rightarrow$ unimodal).

Convexity Convex means that f' monotone non decreasing and $f'' \geq 0$. But convexity $\not\Rightarrow C^1$. Some functions are convex and a few operators preserve convexity. Many models are purposely constructed convex $\Rightarrow$ **Spatial Branch-and-Bound approach**: sift through all $X = [x_-, x_+]$ using clever guide, convex lower approximation $\underline{f}$ of nonconvex f on X.



"Easily" find local $\equiv$ global minimum $\overline{x}$ giving $\underline{f}(\overline{x}) \leq f_* \leq f(\overline{x})$. If the gap $f(\overline{x}) - \underline{f}(\overline{x})$ is too large, we partition X and iterate. If on some partition $\underline{f}(\overline{x}) \geq$ best f-value so far, then that partition is killed.
In the worst case, exponential complexity because you keep dicing and slicing X. But it is exponential in practice too. It depends on how much non-convex f is and how good of a lower approximation $\underline{f}$ is. A cleverer approach is carefully choosing the non-convexities.

2.2 Unconstrained optimization

From now on we'll use $f : R^n \to R$, i.e. $f(x_1, x_2, \ldots, x_n) = f(x)$ with $x = [x_i]_{i=1}^n = [x_1, \ldots, x_n] \in R^n$. Note that $R^n = R \times R \times \ldots \times R$, which is **exponentially larger** than R.
$I = [x_-, x_+], X = I \times I \times \ldots \times I$ hypercube (or hyperrectangle if intervals are disequal).

We need f to be L-c, and sadly no algorithm can work in less than $\Omega((\frac{LD}{\epsilon})^n)$. **Curse of dimensionality**: not really doable unless $n = 3, 5, 10$ tops. Can make to $O((\frac{LD}{\epsilon})^n)$, with multidimensional grid and small enough step (standard approach to hyperparameter optimization). If f analytic, clever B&B can give global optimum. If f black-box (typically, no derivatives), many heuristics can give good solutions, probably not optimal.

Unconstraint global optimization If f is convex, then global $\equiv$ local which is much better: most (but not all) convergence results are dimension independent and if there's dependence it is not exponential. Doesn't mean that all local algorithms are fast: speed may be low (badly linear), cost of f or derivatives computation increases with n dimension (for large n even $O(n^2)$ may be too much) and some dependency on n may be hidden in $O(\,)$ constraints. Yet, large scale optimization can be done.

Notation

Scalar product $\langle x, y \rangle = x^T y = \sum_{i=1}^n x_i y_i = x_1 y_1 + \ldots + x_n y_n$

Norm $||x|| = \sqrt{x_1^2 + \ldots + x_n^2} = \sqrt{\langle x, x \rangle}$

Distance $d(x, y) = ||x - y|| = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2}$

Ball with center $r \in R^n$ and radius $r > 0$ is $B(x, r) = \{y \in R^n \mid ||y - x|| \leq r\}$

Usually $f : D \to R$ with $D = dom(f)$ domain of f which may not be all R^n, but usually ok to ignore $dom(f)$ and assume $f(x) = \infty$ for $x \notin D$

Graph of f lives in R^{n+1}: $gr(f) = \{(f(x), x) \mid x \in R^n\}$

Epigraph of f lives in R^{n+1}: $epi(f) = \{(v, x) \in R^{n+1} \mid v \geq f(x)\}$

Level set at value v: $L(f, v) = \{x \in R^n \mid f(x) = v\}$

Sublevel set at value v: $S(f, v) = \{x \in R^n \mid f(x) \leq v\}$

We have that $x_* \in S(f, v) \ \ \forall\, v \geq f_*$ and $S(f, v) = \emptyset \ \ \forall\, v < f_*$

Tomography $f : R^n \to R$ with $x \in R^n$ origin and $d \in R^n$ direction. You can define $\phi_{x,d}(\alpha) = f(x + \alpha d) : R \to R$ **tomography** of f from x along d.
$\phi_{x,d}$ can always be pictured, but there are infinitely many of them: which x, d?

$||d||$ only changes the scale: $\phi_{x,\beta d}(\alpha) = \phi_{x,d}(\beta\alpha)$ so often convenient to use normalised direction ($||d|| = 1$)

Simplest case: restriction along i-th coordinate
$f_x^i(\alpha) = f(x_1, \ldots, x_{i-1}, \alpha, x_{i+1}, \ldots, x_n) = \phi_{0,u^i}(\alpha)$ with $||u^i|| = 1$

When x, d clear from context, then $\phi(\alpha)$

Simple Functions

Linear $f(x) = \langle b, x\rangle + c, b \in R^n, c \in R$
Tomography $f(x) = \langle b, x\rangle, x = 0, ||d|| = 1$: $\phi(\alpha) = \alpha\langle b, d\rangle = \alpha||b||cos(\theta)$
Plotting this gives a line, increasing because "b same direction as d", more collinear ⇒ steeper. Collinear means steepest line, less collinear means less steap. 90° angle means a flat line. Decreasing if opposite directions.
min $f(x)$ when $\nexists\, x_*$ if $b \neq 0$ ($\Rightarrow \exists\, d \mid \langle b, d\rangle \neq 0$), $\forall\, x$ if $b = 0$

Quadratic with fixed $Q \in R^{n\times n}, q \in R^n$ we have $f(x) = \frac{1}{2}x^TQx + qx$. If $q = 0$ (no linear term), then **homogeneous quadratic**.
Tomography $\phi(\alpha) = f(\alpha d) = \alpha^2(d^TQd) \Rightarrow$ sign and steepness depend on d^TQd, so we need to know about signs of d^TQd. Steeper when d along one axe, least steep when d along the other axe and intermediate steepness when "in between". Again, steeper along the opposite of one axe and least steep along the opposite of the other axe.
With $q \neq 0$ but Q nonsingular then $\lambda_i \neq 0 \ \ \forall\, i$, then $f(x) = \frac{1}{2}(x - x_*)^TQ(x - x_*)[+c]$ for $x_* = -Q^{-1}q$ and $x_* \neq 0$ center of the level sets (which shapes are determined by the eigenvalues).
$y = x - x_*, f_*(y) = y^TQy[+c]$

Directional/partial derivatives The directional derivative of $f : R^n \to R$ at $x \in R^n$ along the direction $d \in R^n$ is

$$\frac{\partial f}{\partial d}(x) = \lim_{t\to 0} \frac{f(x + td) - f(x)}{t} = \phi'_{x,d}(0)$$

How can $\phi'_{x,d}(0)$, the derivative of the (x, d)-tomography (in 0), be computed? A special case is $\frac{\partial f}{\partial d}(x)$, partial derivative of f with respect to x_i at $x \in R^n$, easy to compute by just treating x_j for $j \neq i$ as constants.
The **gradient** is the column vector of all partial derivatives

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{bmatrix}$$

$$f(x) = \langle b, x\rangle \Rightarrow \nabla f(x) = b$$

$$f(x) = \frac{1}{2}x^TQx + qx \Rightarrow \nabla f(x) = Qx + q$$

f differentiable at x if $\exists$ linear function $\psi(h) = \langle b, h\rangle + f(x)$ such that

$$\lim_{||h||\to 0} \frac{|f(x + h) - \psi(h)}{||h||} = 0$$

$$\Rightarrow \psi(0) = f(x) \Rightarrow c = f(x)$$

ψ is equivalent to the first order model of f at x, the error of this equivalence vanishes faster than linearity. So f differentiable at $x \Rightarrow b = \nabla f(x)$ and

$\Rightarrow \frac{\partial f}{\partial x_i}(x)$ exists for every i (but $\Leftrightarrow$ not true)

$\Rightarrow$ first order model of f at x is $L_x(y) = \nabla f(x)(y - x)$

f differentiable $\Rightarrow$ all relevant objects in R^{n+1} and R^n are smooth. If f is non differentiable $\Rightarrow$ kinks appear and things break,

Jacobian Given a vector-valued function $f : R^n \to R^m$, $f(x) = [f_1(x), f_2(x), \ldots, f_m(x)]$, the partial derivatives are the same with extra index

$$\frac{\partial f_j}{\partial x_i}(x) = \lim_{t \to 0} \frac{f_j(x_1, \ldots, x_{i-1}, x_i + t, x_{i+1}, \ldots, x_n) - f_j(x)}{t}$$

The **Jacobian** is the matrix of all mn partial derivatives

$$Jf(X) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \ldots & \frac{\partial f_1}{\partial x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(x) & \ldots & \frac{\partial f_m}{\partial x_n}(x) \end{bmatrix} = \begin{bmatrix} \nabla f_1(x)^T \\ \vdots \\ \nabla f_m(x)^T \end{bmatrix}$$

A $m \times n$ matrix with gradients as rows.

Hessian The $\frac{\partial f}{\partial x_i} : R^n \to R$ have partial derivatives themselves. **Second order partial derivative**, just do it twice

$$\frac{\partial^2 f}{\partial x_i \partial x_j}$$

$$\frac{\partial^2 f}{\partial x_i \partial x_i} = \frac{\partial^2 f}{\partial x_i^2}$$

So $\nabla f(x) : R^n \to R^n$ have a Jacobian and it's called **Hessian** of f at x

$$\nabla^2 f(x) = J\nabla f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \frac{\partial^2 f}{\partial x_2 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_1}(x) \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) & \frac{\partial^2 f}{\partial x_2 \partial x_n}(x) & \ldots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{bmatrix}$$

Requires $O(n^2)$ to store and at least $O(n^2)$ to compute, unless sparse.
$f(x) = \frac{1}{2}x^T Qx + qx \Rightarrow \nabla^2 f(x) = Q$
Second order model = first order model plus second order term (better)

$$Q_x(y) = L_x(y) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x)$$

Theorem $\exists\, \delta > 0 \mid \forall\, y \in B(x, \delta)$ we have that $\frac{\partial^2 f}{\partial x_j \partial x_i}(y)$ and $\frac{\partial^2 f}{\partial x_i \partial x_j}(y)$ exist and are continuous in x

$\Rightarrow \frac{\partial^2 f}{\partial x_j \partial x_i}(y) = \frac{\partial^2 f}{\partial x_i \partial x_j}(y) \Leftrightarrow \nabla^2 f$ symmetric

$\Rightarrow$ all eigenvalues of $\nabla^2 f(x)$ are real

With $f \in C^2$ we have $\nabla^2 f(x)$ continuous everywhere, so symmetric everywhere. C^2 is the best class for optimization.

2.2.1 Optimality conditions

f differentiable at x and x local minimum $\Rightarrow \nabla f(x) = 0 \equiv$ stationary point ($\not\Leftarrow$): to tell them apart we need to look at the curvature of f. If f quadratic I wold know, looking at the eigenvalues of $Q = \nabla^2 f(x)$, so we could approximate f with a quadratic function: the second order model $Q_x(y) = L_x(y) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x)$
$\nabla Q_x(x) = \nabla L_x(x) = \nabla f(x) \Rightarrow \nabla Q_x(x) = 0$, otherwise not minimum. Meaning that in a local minimum there cannot be directions of negative curvature, $\nabla^2 f(x) \geq 0 \Leftrightarrow x$ (global) minimum of Q_x.
Another condition necessary and almost sufficient: $f \in C^2$.

2.2.2 Gradient Methods

Multivariate optimization algorithms These are iterative procedures: start from an initial guess x_0 and compute some process $x_i \longrightarrow x_{i+1}$ to get a **sequence** $\{x_i\}$ that **should go towards an optimal solution**. $\{x_i\} \to x_*$ is one option, the best one, but not the only possibility.
At least $\{f_i = f(x_i)\} \to f_*$: **minimizing sequence**, and clearly $\{x_i\} \to x_* \Rightarrow \{f_i\}$ minimizing sequence, but $\not\Leftarrow$
Two general forms of the process $x_{i+1} = x_i + \alpha_i d_i$:

> **Line search**: first choose $d_i \in R^n$ (**direction**), then choose $\alpha_i \in R$ (**step size** $\equiv$ learning rate in ML)
>
> **Trust region**: first choose $\alpha_i \in R$ (**trust radius**), then d_i

The crucial concept is that the **model** $f_i \simeq f$ is used to construct x_{i+1} from x_i

First order model $\equiv$ gradient method The first order model is the simplest model

$$L_i(x) = L_{x_i}(x) = f(x_i) + \nabla f(x_i)(x - x_i)$$

Idea: $x_{i+1} \in argmin\{L_i(x) : x \in R^n\} = \emptyset$, L_i unbounded below on R^n
It shouldn't move too far from x_i, L_i is only "good" as $\alpha_i \to 0 \Rightarrow d_i = argmin\{\lim_{t\to 0} \frac{f(x+td)}{t}\} = -\nabla f(x_i) =$ steepest descent direction.
$\frac{\partial f}{\partial d_i}(x_i) < 0$ but $\frac{\partial f}{\partial d_i}(x_i + \alpha d_i)$ likely > 0 when α groes $\Rightarrow f$ grows instead of decreasing $\Rightarrow$ very long steps are bad unless $f_* = -\infty$. Very short steps are bad too: f decreases, but very slowly.

Step selection The issue is to find the Goldilocks Step α_i efficiently (a few function evaluations). Two extreme strategies:

> **Fixed Stepsize** (FS): $\forall\, i \;\; \alpha_i = \overline{\alpha}$ (how is chosen?)
> Most inexpensive.
>
> **(Exact) Line Search** (LS): $\alpha_i \in argmin\{f(x_i + \alpha d_i\} \mid \alpha \geq 0\}$
> Most expensive but may converge faster.

Of course, something in the middle is better. ϕ_i' low-degree polynomial $\Rightarrow f(x_i + \alpha d_i) = \phi_{x_i,d_i}(\alpha) = \phi_i(\alpha)$

Gradient for quadratic functions $f(x) = \frac{1}{2}x^T Q x + qx$ with $Q \geq 0$ otherwise f is unbounded below. x_* solves $Qx = -q$ if it exists, which is linear algebra but the linear system requires at most $O(n^3)$ while computing $d_i = -\nabla f(x_i) = -Qx_i - q$ is $O(n^2)$
Line search is easy, $O(n^2)$ with $\alpha_i = \frac{||d_i||^2}{d_i^T Q d_i}$

```
procedure x = SDQ(Q, q, x, eps):
  while (||nablf(x)||>eps):
    d = -nablf(x)
    alpha = ||d||^2/dT*Q*d
    x = x + alpha*d
```

With `nablf` being ∇f

Analysis Never obvious because we have to use properties of x_* which is unknown, but in this case there's a nifty trick

$$f_*(x) = \frac{1}{2}(x - x_*)^T Q(x - x_*) = f(x) - f_*(x) = A(x)$$

for which, with Q positive definite

$$A(x_{i+1}) = \left(1 - \frac{||d_i||^4}{(d_i^T Q d_i)(d_i^T Q^{-1} d_i)}\right) A(x_i)$$

This becomes linear convergences (λ_1, λ_n being respectively the max and min eigenvalues of Q)

> Simple $A(x_{i+1}) \leq (1 - \frac{\lambda_n}{\lambda_1})A(x_i)$
>
> Elaborated $A(x_{i+1}) \leq \frac{\lambda_1 - \lambda_n}{(\lambda_1 - \lambda_n)^2} A(x_i)$

The good news is that n doesn't appear, it's **dimension-independent** so doable for very large scale machine learning.
The bad news is that $r \to 1$ as conditioning of $Q = \frac{\lambda_1}{\lambda_n} \to \infty$

When linear convergence may not be enough The convergence is fast if $\lambda_1 \simeq \lambda_n$ (one iteration for $||x||^2$) and rather slow if $\lambda_1 >> \lambda_n$. Intuitively, the algorithm zig-zags a lot when level sets are very elongated. Another bad news is that there may be an "hidden dependency": λ_1 and λ_n may depend on n and $\frac{\lambda_1}{\lambda_n}$ may grow as $n \to \infty$.
Let's extend it to every function.

Gradient methods for general functions

Given f a general nonlinear function, the algorithm is almost the same:

```
procedure x = SDQ(f, x, eps):
  while (||nablf(x)||>eps):
    d = -nablf(x)
    alpha = stepsize(f, x, d)
    x = x + alpha*d
```

`stepsize` is the crucial part: FS or (inexact) LS. Need to avoid two opposite problems:

Scylla: α_i not too large to avoid $f(x_{i+1}) > f(x_i)$

Charybdis: α_i not too small to avoid stalling

With $\frac{\partial f}{\partial d_i}(x_i) < 0$ hence $\alpha_i \to 0$ we avoid Scylla but may hit Charybdis.
`stepsize(f, x, d)` $= LS(\phi_{x,d}, [0, \infty], \epsilon')$ is attractive but $\epsilon' = 0$ in general is not possible: how to choose it? Depends on the stopping criterion in $LS()$, let's assume $|\phi_{x,d}(\alpha)| \leq \epsilon'$. A fundamental property is

$$\phi_i'(\alpha) = \frac{\partial f}{\partial d_i}(x_i + \alpha d_i) = \langle \nabla f(x_i + \alpha d_i), d_i \rangle$$

$$\Rightarrow |\phi_i'(\alpha_i)| = |\langle d_i, \nabla f(x_{i+1}) \rangle| = |\langle \nabla f(x_i), \nabla f(x_{i+1}) \rangle|$$

The good news is that only an approximate stationary point of ϕ_i is needed, no global minimum (and not even a local minimum, can be a local maximum or a saddle point) $\Rightarrow$ f convex/unimodal is not needed. Also we can prove that the algorithm works with $\epsilon' = \epsilon ||\nabla f(x_i)||$
A bad news is that the LS should become more accurate as the algorithm proceeds, and the LS can be very approximate ("far from x_*").
Usually works well in practice with arbitrary fixed ϵ'

Notes on the stopping criterion One would want $A(x_i) < \epsilon$ or $R(x_i) < \epsilon$ as stopping criterion, the issue is that f_* is often unknown and cannot be used online. We need a lower bound $\underline{f} \leq f_*$, tight at least towards termination, but in general there are no good $\underline{f}$ available because good estimates of f_* are hard to get.
We can use $||\nabla f(x_i)||$ as proxy of $A(x_i)$ (small $\Rightarrow$ small) but the exact relationship is hard to assess, so choosing ϵ is not obvious.
Sometimes we use a relative stopping condition $||\nabla f(x_i)|| \leq \epsilon ||\nabla f(x_0)||$, and sometimes $||\nabla f||$ has some meaning that can be used. Sometimes, we don't really care is $A(x_i)$ or $R(x_i)$ are small (machine learning).

Efficiency The efficiency is basically the same.
With $f \in C^2, x_*$ local minimum such that $\nabla^2 f(x_*)$ positive definite, exact LS $\{x_i\} \to x_* \Rightarrow \{f_i\}_{i \geq k} \to f_*$ linearly for large enough k, with $r = \frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}^2$ with λ_1, λ_n those of $\nabla^2 f(x_*)$
The result can be extended to inexact LS with $r \simeq 1 - \frac{\lambda_n}{\lambda_1}$ (worse), with "$\simeq$" depending on LS parameters.

Fixed Stepsize

With $\alpha_i = \overline{\alpha}$ for each i it's much simpler but also rigid. Easier to avoid Charybdis: $\sum_{i=1}^{\infty} \alpha_i = \infty$
Yet $d_i = -\nabla f(x_i) \Rightarrow$ one wants $\{||d_i||\} \to 0$, so care is still required. We also have that $\alpha_i \to 0$ surely avoid Scylla, but it's not possible here.
The fundamental trick is $d_i = -\nabla f(x_i) \Rightarrow ||x_{i+1} - x_i||$ automatically changes along iterations even if α_i is fixed.
$d_i = \frac{-\nabla f(x_i)}{||\nabla f(x_i)||} \equiv ||d_i|| = 1$ would necessarily require $\alpha_i \to 0$, yet f varies very rapidly so only very short α_i are possible.
It's crucial to bound how rapidly f changes.

L-smoothness

$$f \text{ L-smooth } \Rightarrow \phi(\alpha) \leq \phi(0) + ||\nabla f(x)||^2 \left(\frac{L\alpha^2}{2-\alpha}\right)$$

Powerful general idea: find α giving the best worst-case improvement, meaning $v(\alpha) = \frac{L\alpha^2}{2-\alpha}, \alpha_* = \frac{1}{L}$ (constant!), $v(\alpha_*) = -\frac{1}{2L}$ hence

$$f(x_{i+1}) - f(x_i) \leq -\frac{||\nabla f(x_i)||^2}{2L}$$

Can't do better if you trust the quadratic bound (which you should not).
The error decreases sublinearly: a term is subtracted to a_i rather than multiplied $a_{i+1} = f(x_{i+1}) - f_* \leq a_i - \frac{||\nabla f(x_i)||^2}{2L}$
In fact $a_i \leq \frac{2L||x_0 - x_*||^2}{i+3} \Rightarrow i \geq O(\frac{LD^2}{\epsilon})$ (note that the initial point matters)
However we used Q nonsingular $\Leftrightarrow \lambda_n > 0$, which does make a difference.

Stronger forms of convexity f convex means that $\forall\, x, y \in R^n$ we have

$$\alpha f(x) + (1-\alpha) f(y) \geq f(\alpha x + (1-\alpha) y) \quad \forall\, \alpha \in [0,1] \Leftrightarrow f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle$$

Strictly convex means

$$\alpha f(x) + (1-\alpha) f(y) > f(\alpha x + (1-\alpha) y) \quad \forall\, \alpha \in [0,1] \Leftrightarrow f(y) > f(x) + \langle \nabla f(x), y - x \rangle$$

Quadratic with $\lambda_n > 0$ more than that: it grows at least as fast as $\lambda_n ||x||_2^2$, meaning **strongly convex modulus** λ_n $\equiv \lambda_n$-convex.
f **strongly convex modulus** $\tau > 0$ (τ-convex) if $f(x) - \frac{\tau}{2}||x||_2^2$ is convex $\Leftrightarrow$

$$\alpha f(x) + (1-\alpha) f(y) \geq f(\alpha x + (1-\alpha) y) + \frac{\tau}{2}\alpha(1-\alpha)||y-x||^2 \quad \forall\, \alpha \in [0,1] \Leftrightarrow$$

$$\Leftrightarrow f(y) > f(x) + \langle \nabla f(x), y - x \rangle + \frac{\tau}{2}||y-x||^2 \Leftrightarrow \nabla^2 f(x) \succeq \tau I$$

$f \in C^2$, L-smooth and τ-convex $\equiv \tau I \preceq \nabla^2 f \preceq L I \equiv \tau \leq \lambda_n \leq \lambda_1 \leq L$, eigenvalues of $\nabla^2 f$ are bounded both below and above.

Convergence rate with strong convexity Minimize on x both sides independently

$$f(x_*) \geq f(x_i) - \frac{||\nabla f(x_i)||^2}{2\tau} \Rightarrow ||\nabla f(x_i)||^2 \geq 2\tau(f(x_i) - f(x_*))$$

"If $a_i = f(x_i) - f_*$ is large then the gradient must also be large."
L-smooth $\Rightarrow a_{i+1} \leq a_i - \frac{||\nabla f(x_i)||^2}{2L} \Rightarrow a_{i+1} \leq a_i(1 - \frac{\tau}{L})$
A small difference in f makes a big difference in convergence $\Rightarrow$ properties of f more important than the algorithm.

Inexact Line Search

Armijo If FS works, then any rough LS also should work provided that f_i decreases enough.
The **Armijo condition** is $0 < m_1 < 1$

(A) $\phi(\alpha) \leq \phi(0) + m_1 \alpha \phi'(0)$

$\alpha \to 0$ satisfies (A). But if we avoid Charybdis then we "converge".
$\alpha_i \geq \overline{\alpha} > 0$ and (A) holds $\forall\, i \Rightarrow$ either $\{f_i\} \to -\infty$ or $\{||\nabla f(x_i)||\} \to 0$
All accumulation points (if any) of $\{x_i\}$ are stationary. The proof: assume $-\phi_i'(0) = ||\nabla f(x_i)||^2 \geq \epsilon > 0$ and (A) hold $\forall\, i \Rightarrow$

$$f_{i+1} \leq f_i + m_1 \alpha_i \phi_i'(0) \leq f_i - m_1 \overline{\alpha}\epsilon \Rightarrow f_i \leq f_0 - m_1 \overline{\alpha}\epsilon \Rightarrow \{f_i\} \to -\infty$$

Don't even need $\alpha_i \geq \overline{\alpha} > 0$, just $\sum_{i=1}^{\infty} \alpha_i = \infty$ (meaning $\alpha_i \to 0$ "slow enough"), but how do we ensure that α_i does not get too small? We need to add a Charybdis avoiding condition to (A)

Wolfe Goldstein condition $m_1 < m_2 < 1$

(G) $\phi(\alpha) \geq \phi(0) + m_2\alpha\phi'(0)$

Issue: $(A) \cap (G)$ can exclude all local minima **Wolfe condition** $m_1 < m_3 < 1$

(W) $\phi'(\alpha) \geq m_3\phi'(0)$

The derivative has to be a bit closer to 0, but can be $>> 0$: there's a strong Wolfe, too

(W') $|\phi'(\alpha)| \leq m_3|\phi'(0)| = -m_3\phi'(0) [\Rightarrow (W)]$

We have that $(A) \cap (W)$ captures all local minima (and maxima), usually m_1 close to 1, and $(A) \cap (W')$ ensures $\phi'(\alpha) \not> \not< 0$
Such points always exists.

Amijo-Wolfe in practice m_1 small enough so that local minima are not cut: just go for the local minima and stop whenever $(A) \cap (W)$ or (W') holds. Hard to say if m_1 is small enough, usually $m_1 = 0.0001$ is enough. Specialized LS can be constructed for the odd case it's not, with some more logic for the nasty cases.
A simpler version: "backtracking" LS, only check (A)

```
procedure alpha = BLS(phi, alpha, m1, tau)  # tau < 1
  while(phi(alpha) > phi(0) + m1*alpha*dphi(0)):
    alpha = tau*alpha
```

With `dphi(0)` meaning $\phi'(0)$
Recall that $\exists\, \overline{\alpha} > 0 \mid (A)$ is satisfied $\forall\, \alpha \in (0, \overline{\alpha}_i]$. Assuming as input $\alpha = 1$, BLS produces $\alpha \geq \tau^{h_i}$ with $h_i \geq min\{k \mid \tau^k \leq \overline{\alpha}_i\}$

$$\overline{\alpha}_i \geq \overline{\alpha} > 0 \forall\, i \Rightarrow \exists\, h \mid \alpha \geq \tau^h \quad \forall\, i \Rightarrow \textbf{convergence}$$

We need conditions on f to get it:

> f L-smooth $\Rightarrow \phi$ is $[L||d||^2]$-smooth
>
> $\Rightarrow -\phi'(0) = ||d||^2 = ||\nabla f(x)||^2$
>
> ϕ is $[L||d||^2]$-smooth $\Rightarrow \alpha'$ and $\overline{\alpha}$ are "large":
> $L||d||^2(\alpha - 0) \geq \phi'(\alpha') - \phi'(0) > (1 - m_3)(-\phi'(0)) = (1 - m_3)||d||^2$
>
> $\Rightarrow \overline{\alpha} > \alpha' > \frac{1-m_3}{L}$
>
> If f also τ-convex $\Rightarrow$ convergence linear with $r \simeq \frac{1-\tau}{L}$, depending on m_1, m_3

Might be rather slow, need something better.

2.2.3 More-Than-Gradient Methods

General descent methods So far, the crucial assumption was $d_i = -\nabla f(x_i)$
The crucial convergence arguments are:

> $\phi_i'(0) = -||\nabla f(x_i)||^2$, or "far from x_* the derivative is very negative"
>
> "You can get a non-vanishing fraction of the descent promised by $\phi_i'(0)$", the "exact" LS or Armijo or FS + L-smooth $\Rightarrow \alpha_i$ doesn't not go to 0 too fast.

so that there's a significant decrease at each step unless $||\nabla f(x_i)|| \to 0$.
There are **many other directions** that ensure the first argument. The **twisted gradient algorithm**: $d_i = -\nabla f(x_i)$ rotated by 45 degrees $\Leftrightarrow \phi_i'(0) = -||\nabla f(x_i)||^2 \cos(\frac{\pi}{4}) < 0 \Rightarrow$ convergence proofs carry over.
In R^n there are many other such vectors and many other feasible angles: basically, θ not too close to $\frac{\pi}{2}$ so that $\cos(\theta)$ is not too small.

Convergence of general descent methods **Descent direction** is $\frac{\partial f}{\partial d_i}(x_i) < 0 \equiv \langle d_i, \nabla f(x_i) \rangle < 0 \equiv \cos(\theta_i) > 0$ meaning that d_i points roughly in the same direction as $-\nabla f(x_i)$. There's a whole half space of descent directions, a lot of flexibility.
Zoutendijk's Theorem: $f \in C^1 L$-smooth, $f_* > -\infty, (A) \cap (W) \Rightarrow \sum_{i=1}^{\infty} \cos^2(\theta_i)||\nabla f(x_i)||^2 < \infty$
A consequence is that $\sum_{i=1}^{\infty} \cos^2(\theta_i) = \infty \Rightarrow \{||\nabla f(x_i)||\} \to 0 \Leftrightarrow d_i$ doesn't get perpendicular to $\nabla f(x_i)$ "too fast" $\Rightarrow$ convergence.
Very many d_i, but which is better than $-\nabla f$? Need to look farther than the first order model.

Newton's Method For a faster convergence we want a better direction, so a better model. The next better model to the linear (gradient) is the quadratic. $\nabla^2 f(x_i) \succ 0 \Rightarrow \exists$ minimum of second order model $Q_{x_i}(y) \Rightarrow$ Newton's direction $d_i = -[\nabla^2 f(x_i)]^{-1}\nabla f(x_i)$ (just R^n version). No problem with the step, we use $\alpha_i = 1$.
The **Newton's Method** is $x_{i+1} = x_i + d_i$, a step of $\alpha_i = 1$ along d_i.
It's not globally convergent, needs to be globalised. Easy as $\nabla^2 f(x_i) \succ 0 \Rightarrow [\nabla^2 f(x_i)]^{-1} \succ 0 \Rightarrow d_i$ is of descent.

$$\langle \nabla f(x_i), d_i \rangle = -\nabla f(x_i)^T [\nabla^2 f(x_i)]^{-1} \nabla f(x_i) < 0$$

but it's not enough, we need it "negative enough".

Globalised Netwtod Simply add AWLS/BLS with $\alpha_0 = 1$. The convergence requires $f \in C^2, L$-smooth and τ-convex

Theorem 1 $\cos(\theta_i)$ bounded away from $0 \Rightarrow$ global convergence.
Meaning $\cos(\theta_i) \geq \overline{\theta} > 0$

Theorem 2 $f \in C^3, \nabla f(x_*) = 0, \nabla^2 f(x_*) \succ 0 \Rightarrow \exists B(x_*, r) \mid x_0 \in B \Rightarrow$ "pure" Newton sequence with $\alpha_i = 1$ that $\{x_i\} \to x_*$ quadratically.

Theorem 3 If $\{x_i\} \to x_* \exists\, h \mid \alpha_i = 1$ satisfies (A) for all $i \geq h$
Requires $m_1 \leq \frac{1}{2}$, because $m_1 > \frac{1}{2}$ cuts away the minimum when f quadratic.

Global phase (α_i varies) + **pure Newton's phase** (ends in $O(1) \simeq 6$ iterations in practice)
If $\nabla^2 f M$-smooth then global phase also $O(1)$: $O\left(\frac{M^2 L^2 (f(x_0) - f_*)}{\tau^5}\right)$
An interpretation is Newton = Gradient in a twisted space. Q is positive semidefinite ($\succeq 0$), so $Q = RR \Leftrightarrow R = Q^{\frac{1}{2}}$: it exists and it's symmetric $Q = H\Lambda H^T \Rightarrow R = H\sqrt{\Lambda}H^T$
$f(x) = \frac{1}{2}x^T Q x + qx, d = -x - Q^{-1}q \Rightarrow \nabla f(x + d) = 0$ Netwon ends in one iteration. $y = Rx \Leftrightarrow x = R^{-1}y, h(y) = f(R^{-1}y) = \frac{1}{2}y^T I y + qR^{-1}y$ (in y-space, $\nabla^2 f(x_i)$ looks like $I \Rightarrow$ gradient is fast)
$g = -\nabla h(y) = -y - R^{-1}q \Rightarrow \nabla h(y + g) = 0$

Nonconvex case The Newton's method is a space dilation: a linear map making $\nabla^2 f$ "simple", but it's not necessarily $\nabla^2 f(x_i)^{-1}$ especially when $\not\succeq 0$

$$d_i = -H_i \nabla f(x_i), \tau I \preceq H_i \preceq LI, (A) \cap (W) \Rightarrow \text{Global convergence}$$

Any $\epsilon_i > -\lambda_n$ works (but numerical issues). Also algorithmic issues: $\lambda_n(\nabla^2 f(x_i) + \epsilon I)$ is very small, so the axes are very elongated and x_{i+1} far from x_i (not good for a local model)
Simple form $\epsilon = max\{0, \delta - \lambda_n\}$ for appropriately chosen δ. This solves $min\{||H - \nabla^2 f(x_i)||_2 \mid H \succeq \delta I\}$, works for other norms too.
In every case, $\{x_i\} \to x_*$ with $\nabla^2 f(x_*) \succeq \delta I \Rightarrow \epsilon_i = 0 \Leftrightarrow H_i = \nabla^2 f(x_i)$ eventually (quadratic convergence in the tail)

Quasi-Newton The space of H_i that gives fast convergence is big. Superlinear convergence if H_i looks like $\nabla^2 f(x_i)$ along d.
General derivation of Quasi-Newton methods $m_i(x) = \nabla f(x_i)(x - x_i) + \frac{1}{2}(x - x_i)^T H_i (x - x_i), x_{i+1} = x_i + \alpha_i d_i$
Having computed x_{i+1} and $\nabla f(x_{i+1})$, new model

$$m_{i+1}(x) = \nabla f(x_{i+1})(x - x_{i+1}) + \frac{1}{2}(x - x_{i+1})^T H_{i+1}(x - x_{i+1})$$

We would like H_{i+1} to have the following properties:

$H_{i+1} \succ 0$ (new model is strongly convex)

$\nabla m_{i+1}(x_i) = \nabla f(x_i)$ (new model agrees with old information)
Secant equation $H_{i+1}(x_{i+1} - x_i) = \nabla f(x_{i+1}) - \nabla f(x_i)$

$||H_{i+1} - H_i||$ "small" (new model is not too different)

Depending on the choices at iteration i, it may not be possible to achieve both the first and second properties.
Notation $s_i = x_{i+1} - x_i = \alpha_i d_i$, $y_i = \nabla f(x_{i+1}) - \nabla f(x_i)$
Secant equation (S) $H_{i+1}s_i = y_i$
$(S) \Rightarrow s_i y_i = s_i^T H_{i+1} s_i$ and the first and second properties $\Rightarrow s_i y_i > 0$ **curvature condition** (C) (often written as $\rho_i = \frac{1}{y_i s_i} > 0$)
So s_i needs to be properly chosen at iteration i for things to work at $i+1$.
Quasi-Newton: d_i fixed, but s_i also depends on α_i (free). A good news $(W) \Rightarrow (C)$. Assuming an AWLS, (C) can always be satisfied.

DFP With the three properties we have $H_{i+1} = argmin\{||H - H_i|| \mid (S), H \succeq 0\}$
Needs appropriate $||\ ||$: **Davidon-Fletcher-Powell formula**

(DFP) $H_{i+1} = (I - \rho_i y_i s_i^T) H_i (I - \rho_i s_i y_i^T) + \rho_i y_i y_i^T$

So H_{i+1} is a rank-two correction of H_i, $O(n^2)$ to produce H_{i+1} from H_i
Actually need $B_{i+1} = H_{i+1}^{-1}$: **Sherman-Morrison-Woodbury formula**

(SMW) $[A + ab^T]^{-1} = \frac{A^{-1} - A^{-1}ab^T A^{-1}}{1 - b^T A^{-1} a}$

$\Rightarrow (DFP)^{-1}$ $B_{i+1} = \frac{B_i + \rho_i s_i s_i^T - B_i y_i y_i^T B_i}{y_i^T B_i y_i}$

$O(n^2)$ per iteration, just matrix-vector products and no inverses.
This is kind of a learning of $\nabla^2 f$ out of samples of ∇f. Efficient but can do better.

BFGS (S) for B_{i+1} is symmetric, just $B \leftrightarrow H$ and $s \leftrightarrow y$: $s_i = B_{i+1} y_i \Rightarrow B_{i+1} = argmin\{||B - B_i|| \mid (S), B \succeq 0\}$
Broyden-Felcther-Goldfarb-Shanno formulae still $O(n^2)$

$(BFGS)$ $H_{i+1} = \frac{H_i + \rho_i y_i y_i^T - H_i s_i s_i^T H_i}{s_i^T H_i s_i}$

$(BFGS)$ $B_{i+1} = (I - \rho_i s_i y_i^T) B_i (I - \rho_i y_i s_i^T) + \rho_i s_i s_i^T = B_i + \rho_i((1 + \rho_i y_i^T B_i y_i) s_i s_i^T - (B_i y_i s_i^T + s_i y_i^T B_i))$

Conjugate gradient method for quadratic functions Gradient method + exact LS $\Rightarrow \langle \nabla f(x_{i+1}, d_i \rangle = 0 \equiv d_{i+1}$ perpendicular to d_i. Property: x_{i+1} minimum over all the small subspace of d_i. This is lost at $i+2$: zig-zags.
Would be nice if x_{i+1} minimum on the subspace of $\{d_1, \ldots, d_i\}$, getting larger with every iteration.
Possible with quadratic $f \equiv$ linear systems, with two conditions:

all directions are Q-conjugate: $d_i^T Q d_j = 0 \quad \forall\, i, j$

the optimal step is always taken along each d_i

Can't use $d_i = -\nabla f(x_i)$, have to deflect $-\nabla f(x_i)$ using d_{i-1}: $d_0 = 0$ and $d_i = -\nabla f(x_i) + \beta_i d_{i-1}$
The crucial, but only, decision is β_i: **Fletcher-Reeves** (closed) **formula**:

$$\beta_i = \frac{\nabla f(x_i)^T Q d_{i-1}}{d_{i-1}^T Q d_{i-1}} = \frac{||\nabla f(x_i)||^2}{||\nabla f(x_{i-1})||^2}$$

f quadratic + exact LS $\Rightarrow$ quadratic conjugate gradient (CG): $\nabla f(x) = 0 \equiv Qx = -q$ in at most n iterations. If properly preconditioned $<< n$ iterations.
Also many β formulae, all equivalent for quadratic f but not so here.
LS only exact with quadratic f, otherwise AWLS.

Convergence and efficiency Depends on β-formula. F-R requires $m_1 < m_2 < \frac{1}{2}$ for $(A) \cap (W')$ to work.
$(A) \cap (W') \not\Rightarrow d_i$ of P-R is of descent, unless $\beta_{PR,i} = max\{\beta_i, 0\}$
Restart: from time to time take plain $-\nabla f$. It's a good idea especially for F-R: one bad step leads to many bad steps, restarting cures this. Typically restart after n steps.
n CG steps $\simeq$ 1 Newton steps, in n steps CG exactly solves a quadratic function. Powerful approach, not easy to manage.

Deflected Gradients method CG's idea: use previous direction while computing the current one. Simple form: $x_{i+1} = x_i - \alpha_i \nabla f(x_i) + \beta_i(x_i - x_{i-1})$ with β_i called **momentum**, x_i **heavy** and $\nabla f(x_i)$ the **force** steering the trajectory.
Not a descent algorithm, may zig-zags: specific analysis. For L-smooth and τ-convex, better linear convergence

$$||x_{i+1} - x_*|| \leq \left[\frac{\sqrt{L} - \sqrt{\tau}}{\sqrt{L} + \sqrt{t}}\right] ||x_i - x_*|| \quad (\sqrt{L} << L)$$

Gridsearch required to find α_i and β_i in practice. For non-convex f, converges if $\beta \in [0, 1), \alpha \in (0, 2\frac{1-\beta}{L})$
∇f computed after momentum but before descent: not at all a gradient-like method, almost entirely different.
It's **optimal** $O(\frac{LD^2}{\sqrt{\epsilon}}$ for L-smooth not τ-convex.

2.2.4 Less-Than-Gradient Methods

Stochastic Gradient The motivation comes from the incremental a.k.a. stochastic gradient in ML.
We have $I = \{1, \ldots, m\}$ indexes, $X = \{X_i \in R^h\}_{i \in I}$ inputs and $y = \{y_i \in R^k\}_{i \in I}$ outputs and a arbitrarily complex predictor $\pi(x; w) : R^h \to R^k$ parametric on $w \in R^n$ with $l : R^k \times R^k \to R$ **loss function** (could be l_i), fitting

$$min\{f(w) = \sum_{i \in I} (f_i(w) = l(y_i, \pi(X_i; w))) \mid w \in R^n\}$$

$\nabla f(w) = \sum_{i \in I} \nabla f_i(w)$ with $m >> n$ we have a sum of a very large number of terms.
A trivial case: $\pi(x; w) = \langle x, w \rangle$, $l = \frac{d^2}{2} \equiv$ linear least squares.

$$f_i(w) = \frac{(y_i - \langle X_i, w \rangle)^2}{2} \qquad \nabla f_i(w) = -X_i(y_i - \langle X_i, w \rangle)$$

Each ∇f_i is cheap, but computing the full ∇f is already costly.
Intuition: X_i are independent and identically distributed $\Rightarrow \nabla f_i$ are i.i.d. too $\Rightarrow$ "many of them will cancel out" so a **small sample** is enough to compute a close $\simeq$ to the "true" ∇f: $K \subset I$ "small", so $\nabla f_K(w) = \sum_{i \in K} \nabla f_i(w) =$ incremental gradient. Cheaper but $-\nabla f_K$ is not a descent direction, a different analysis is needed.

How to choose K? What is the better $|K|$? Apparently, no better way than **random**: **stochastic gradient**. With $K = I$ "batch" and $|K| < m$ mini-batch. An extreme version, on-line: observations used one-by-one and discarded (no memory).

Nondifferentiable functions We would add a regularizer $\Omega(w)$ so that it becomes

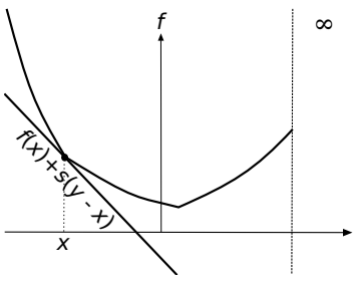
$$min\{\sum_{i \in I} l(y_i, \pi(X_i; w) + \mu\Omega(w) \mid w \in R^n\}$$

with μ hyperparameter. For example, a simple ridge regularization $\Omega(w) = \frac{||w||_2^2}{2} \in C^1$ with $\nabla\Omega(w) = w$.
The regularization simplifies the model and reduces the number of parameters (feature selection).

Smooth methods fail on nonsmooth functions With a nonsmooth function, many directions point outside, not giving a descent direction. But if f is convex, it can be exploited.

Convex Nondifferentiable Functions

Subgradients and subdifferentials s **subgradient** of f at x: $f(y) \geq f(x) + s(y - x)$, $\forall\, y \in R^n$
By changing s we have **too much information** of the first-order. so s it's a **direction**.



For x on the border of $\text{dom}(f)$, $||s|| \to \infty$, while for $s = 0 \Rightarrow x$ local/global minimum (but there can be many $s \neq 0$ at local minimum).

Subdifferential is a set $\partial f(x) = \{s \in R^n \mid s \text{ is a subgradient at } x\}$

$\partial f(x) = \{\nabla f(x)\} \Leftrightarrow f$ differentiable at x

$\frac{\partial f}{\partial d}(x) \leq \langle s, d\rangle \ \forall s \in \partial f(x) \Rightarrow d$ is a descent direction $\Leftrightarrow \langle s, d\rangle < 0 \ \forall s \in \partial f(x)$

$s_* = -\text{argmin}\{||s|| \mid s \in \partial f(x)\}$ **steepest descent direction**

x **global minimum** $\Leftrightarrow 0 \in \partial f(x)$

Subgradients in R^n If $\partial f(X) = \{g = \nabla f(x)\}$, $g \perp (f, f(x))$ i.e. $-g$ points towards x_*, then we can take $d | \langle g, d\rangle < 0$ (descent direction). But if f is nondifferentiable in x, there are many different g and all of them are $\perp S(f, f(x))$ (x = kink point), but **not all of them are descent directions**.
However, any (-)subgradient points towards x_*:

$$f(x_*) \geq f(x) + \langle g, x_* - x\rangle \Rightarrow \langle g, x_* - x\rangle \leq f(x_*) - f(x) \leq 0$$

Enough for gradient-type approaches.

Convex nondifferentiable optimization is hard Nondifferentiable optimization is **orders of bagnitude slower**

$f \in C^1$	τ-convex	L-smooth	$O\left(\log\left(\frac{1}{\epsilon}\right)\right)$
$f \notin C^1$	τ-convex	L-Lipschitz	$\Omega\left(\frac{L^2}{\epsilon}\right)$
$f \in C^1$	convex	L-smooth	$O\left(\log\left(\frac{1}{\sqrt{\epsilon}}\right)\right)$
$f \notin C^1$	convex	L-Lipschitz	$\Omega\left(\frac{L}{\epsilon^2}\right)$

Furthemore, "fixed-step" cannot work for $f \notin C^1$

$f(x) = L|x|$, $x_0 = -\frac{\alpha L}{2}$

$g_1 = -L$, $x_1 = x_0 - \alpha g_1 = \frac{\alpha L}{2}$

$g_2 = L$, $x_2 = x_1 - \alpha g_2 = -\frac{\alpha L}{2} = x_0$

$g_3 = -L \ldots$

For $f \in C^1$ **the gradient is unique**, $d = -\nabla f(x)$:

$f(x + \alpha d) < f(x) \ \forall \alpha > 0$ small enough

$||d||$ is a two-sided proxy of $A(x)$: $||d||$ "small" $\Leftrightarrow f(x)$ "close" to $f_* \equiv ||d|| \leq \epsilon$ is an effective stopping criterion

Can use fixed step since $||x_{i+1} - x_i|| \to 0$ automatically: $||d_i|| \to 0$ even if $\alpha_i \geq \overline{\alpha} > 0$

For $f \notin C^1$, there can be **many different subgradients** $d = -[g \in \partial f(x)]$, and for any one of them (can't choose):

$f(x + \alpha d)$ may be $\geq f(x) \ \forall \alpha$

$||d||$ is a two-sided proxy of $A(x)$: $||d||$ "small" $\Rightarrow f(x)$ "close" to f_*, but $f(x)$ "close" $\not\Rightarrow ||d||$ "small"! So $||d|| \leq \epsilon$ is an ineffective stopping criterion (almost never happens)

Can't use fixed step since $||d||$ can be big even if $x = x_*$: to ensure $||x_{i+1} - x_i|| \to 0$ one has to force $\alpha_i \to 0$ (but not too fast).

Subgradient methods

Fundamental relationship Any (-)subgradient "points towards x_*", so an **appropriate step** along $-g$ brings closer to x_*, meaning that $x_{i+1} = x_i - \alpha_i g_i$ **makes sense with the right** α_i.
A fundamental relationship:

$$||x_{i+1} - x_*||^2 = ||x_i - \alpha_i g_i - x_*||^2 =$$

$$= ||x_i - x_*||^2 + 2\alpha_i \langle g_i, x_* - x_i \rangle + \alpha_i^2 ||g_i||^2 \leq ||x_i - x_*||^2 \quad + \quad \underbrace{2\alpha_i (f_* - f(x_i))}_{<0} \quad + \quad \underbrace{\alpha_i^2 ||g_i||^2}_{>0}$$

As $\alpha \to 0$ (short step), the $2\alpha_i(f_* - f(x_i))$ term dominates $\Rightarrow x_{i+1}$ closer to x_* than x_i.
Short but not too short: **Diminishing-Square Summable** stepsize (**DSS**)

$$\sum_{i=1}^{\infty} \alpha_i = \infty \wedge \sum_{i=1}^{\infty} \alpha_i^2 < \infty$$

So $\alpha_i \to 0$ but not fast enough that the series converges ($\alpha_i = \frac{1}{i}$).
DSS just works: $\lim_{i \to \infty} inf\{f_h \mid h \geq i\} = f_*$
Incredibly robust result: α_i chosen a priori, $f(x_i)$ not even used (only g_i), but **convergence is very slow**.
If we knew f_*, we could estimate α_i, let's assume we do: $min\{a\alpha_i^2 + 2b\alpha_i\} \equiv \alpha_i = -\frac{b}{a} = \frac{f(x_i) - f_*}{||g_i||^2} \geq 0$.
Polyak stepsize (**PSS**):

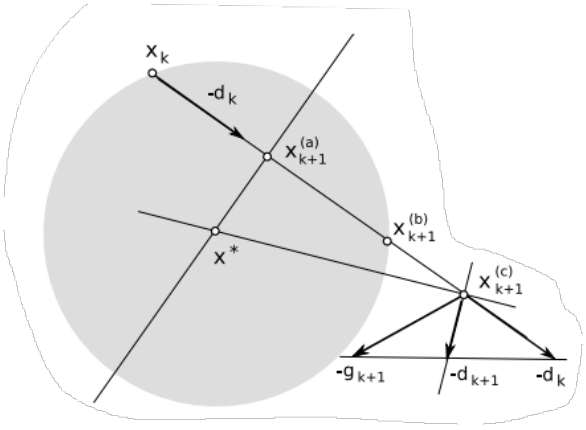
$$\alpha_i = \beta_i \frac{f(x_i) - f_*}{||g_i||^2}$$

With $\beta_i = 1$ we have a "optimal step", but works $\forall\, \beta_i \in (0, 2)$ since this $\Rightarrow ||x_{i+1} - x_*||^2 < ||x_i - x_*||^2 \Rightarrow \{x_i\} \to x_*$
In pratice is vastly better, but as far as it can go (not much): $min\{f(x_h) \mid g \leq i\} - f_* \leq L\frac{||x_1 - x_*||}{\sqrt{i}} \Rightarrow O(\frac{1}{\epsilon^2}$
From $\epsilon = 1e{-}3$ to $\epsilon = 1e{-}4$ we get 100x the iterations. The good news is that PSS would be optimal if we knew f_*, which we don't $\to$ **target level stepsize** (vanishing), "if you don't know it estimate it, but be ready to revise your estimate".
Reference value $f_{ref}-$ threshold δ = target level $\simeq f_*$. With a "good improvement", f_{ref} gets smaller and also does the target level. If there are too many steps without improvement, δ gets smaller so the target level increases.
Too many parameters $\rho \in (0, 1), \beta \in (0, 2), \delta_0 > 0, R > 0$. $\{f_{rec,i}\} \to f_*$ but there's no reasonable stopping criterion, just "stop after a while". So slow convergence.

Deflected Subgradient Better model, better direction. There's no second-order information but deflection is possible

$$d_i = \gamma_i g_i + (1 - \gamma_i) d_{i-1},\ x_{i+1} = x_i - \alpha_i d_i$$

"Conjugate subgradient". To have theoretical convergence, there are some rules:



Stepsize-restricted, deflection-first

$$\alpha_i = \beta_i \frac{f(x_i) - f_*}{||d_i||^2} \text{ and } \beta_i \leq \gamma_i$$

"As deflection increases, stepsize has to decrease".

Deflection-restricted, stepsize-first

$$\frac{\alpha_{i-1}||d_{i-1}||^2}{(f(x_i) - f_*) + \alpha_{i-1}||d_{i-1}||^2} \leq \gamma_i$$

"As $f(x_i)$ approaches f_*, deflection has to decrease"

In both cases, target level to replace f_*.
A closed formula, $\gamma_i \in \text{argmin}\{||\gamma g_i + (1 - \gamma) d_{i-1}||^2 \mid \gamma \in [0, 1]\}$

Smoothed Gradient Methods

The speed is a property of the space, so let's change the space. Requires $f(x) = max\{x^T Az \mid z \in Z\}$ convex and assumed to be easy to compute. $\overline{z}$ optimal for $x \Rightarrow A\overline{z} \in \partial f(x) \Rightarrow f \notin C^1$ (there can be many different $\overline{z}$).
Smoothed $f_\mu(x) = max\{x^T Az - \mu\frac{||z||^2}{2} \mid z \in Z\} \in C^1$ hopefully easy.
In practice is slowish, superlinear in a doubly-logarithmic chart after a long flat leg. Subgradients are faster but flatline at $\epsilon \simeq$ 1e−4, smoothed does at $\epsilon \simeq$ 1e−6.

Bundle Methods

Basic Idea Cutting-plane model. Being f convex, the first-order information that we have is globally valid: I can collect it along the way and use it all.
$\{x_i\} \rightarrow$ bundle $B = \{(x_i, f_i = f(x_i), g_i \in \partial f(x_i))\}$ The **cutting-plane model** f_B if f is

$$f_B(x) = max\{f_i + \langle g_i, x - x_i\rangle \mid (x_i, f_i, g_i) \in B\} \leq f(x) \ \forall x$$

a $(1+\epsilon)$-order model.
$x_{B,*} \in$ argmin$\{f_B(x)\}$, $f_B(x_{B,*}) \leq f_*$: use $x_{B,*}$ as the next iterate a-la Newton.
$f_B \notin C^1$ but computing $x_{B,*}$ is a linear program $\Rightarrow$ easy if $\#B$ "small": $min\{f_B(x)\} = min\{v \mid v \geq f_i + \langle g_i, x - x_i\rangle \ (x_i, f_i, g_i) \in B\}$

Cutting Plane Algorithm $min\{f_B(x)\}$ is the master problem, and (x_*, v_*) the optimal solutions $\rightarrow$ new $(x_*, f(x_*), g_* \in \partial f(x_*))$
$f(x_*) \leq v_* \Rightarrow x_*$ optimal, otherwise $B \leftarrow B \cup (x_*, f(x_*), g_*) \Rightarrow f_B$ becomes a better model.
The min in the master problem focuses $\{x_*\}$ in the right place, with a practical stopping criterion (unlike any subgradient-style algorithm). But $\#B \rightarrow \infty \Rightarrow$ the cost per iteration of the master problem $\rightarrow \infty$.
So the practical convergence is often horrible, can be $O\left(\left(\frac{1}{\epsilon}\right)^{\frac{n}{2}}\right)$

Why $x_{B,*}$ may be infinitely far from x_*, the iterates have no local property ($||x_{i+1} - x_i||$ can be very large and doesn't go smoothly to 0, no fast convergence in the tail.
It's unavoidable: linear functions have no curvature, and we need many of them to make a quadratic one. Pruning B is possible but not easy, and no a priori bound on $\#B$.

Stabilizing $\overline{x}$ as stability center (the best x_i so far) with μ stability parameter (how far from $\overline{x} f_B$ is a good model of f).
Stabilized master problem

$$min\{f_B(x) + \mu\frac{||x - \overline{x}||^2}{2}\}$$

Keeps $\{x_*\}$ "close" to $\overline{x}$ (perhaps too close if μ too large). When μ too small, unstabilized cutting plane.