

# Computational Mathematics for Learning and Data Analysis

Federico Matteoni

A.A. 2021/22

# Index

0.1	Introduction . . . . .	2	1.1	Quick recap of linear algebra . . . . .	3
<b>1</b>	<b>Numerical Analysis</b>	<b>3</b>	1.2	SVD Approximation . . . . .	5
			1.3	QR factorization . . . . .	7
			<b>2</b>	<b>Optimization</b>	<b>12</b>
			2.1	Optimization problems . . . . .	13
			2.1.1	Optimization is hard . . . . .	13
			2.1.2	Local Optimization . . . . .	14
			2.1.3	Measuring algorithms speed . . . . .	17
			2.1.4	Global optimization . . . . .	18
			2.2	Unconstrained optimization . . . . .	18
			2.2.1	Optimality conditions . . . . .	20
			2.2.2	Gradient Methods . . . . .	21

## 0.1 Introduction

Exam: project (groups of 2) + oral exam.

This course's goal 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  $\min_{x \in S} f(x)$ , 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  $A \in R^{4 \times 3}, c \in R^3, b \in R^4$

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ \vdots & \vdots & \vdots \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \quad \begin{aligned} b_i &= \sum_{j=1}^3 A_{ij}c_j \\ A_{11}c_1 + A_{12}c_2 + A_{13}c_3 &= b_1 \end{aligned}$$

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  $v_1, v_2, \dots, v_n$  | you can write all vectors  $b$  in a certain space as a linear combination  $v_1\alpha_1 + v_2\alpha_2 + \dots + v_n\alpha_n$  with **unique**  $\alpha_1, \dots, \alpha_n$ . 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  $x$  |  $Ax = 0$  ( $x = 0$  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 |  $A \cdot A^{-1} = A^{-1} \cdot A = I$  identity matrix (1 on the diagonal, 0 otherwise)

Implementation detail: `inv(A) * b` is not the best choice. Better: in Python `scipy.linalg.solve(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  $CA = CB \nRightarrow A = B$  with  $C$  matrix.

If there's  $M$  |  $MC = I$ , 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  $\geq 0$

Also impossible if  $f > -\infty$  but  $\nexists x$ , for example in  $f(x) = e^x$ . However plenty of  $\epsilon$ -approximate solutions ( $\epsilon$ -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

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

Many matrices preserve norm 2

**Orthogonal** A square matrix  $U \in R^{n \times n}$  is orthogonal if  $U^T U = I \vee U U^T = I \vee U^{-1} = U^T$  equivalently

**Theorem:** for an orthogonal matrix  $U \in R^{n \times n}$  and every vector  $x$ ,  $\|Ux\| = \|x\|$

More generally,  $x^T y = (Ux)^T (Uy)$  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  $\lambda$  and **eigenvalue** and  $v$  an **eigenvector** of  $A$ .

For many matrices we can find  $v_1, \dots, v_m$  eigenvectors that are a basis of  $R^m$ , i.e.  $[v_1 | \dots | v_m] = V$  is invertible.

$$A = V \cdot \Lambda \cdot V^{-1} \Leftrightarrow AV = V\Lambda, \text{ with } \Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m \end{bmatrix}$$

Given  $x \in R^m$  I can write  $x = v_1 \alpha_1 + \dots + v_m \alpha_m$  then it's easy to compute the coordinates  $Ax = A(v_1 \alpha_1 + \dots + v_m \alpha_m) = v_1(\lambda_1 \alpha_1) + \dots + v_m(\lambda_m \alpha_m)$

Also  $A^k x = v_1(\lambda_1^k \alpha_1) + \dots + v_m(\lambda_m^k \alpha_m)$

Eigenvalue decompositions do not exist 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 basis.

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

$$\frac{x^T Q x}{x^T x} = \frac{(\alpha z)^T Q (\alpha z)}{\alpha^2}$$

**Generalization for complex matrices**  $\|x\|^2 = |x_1|^2 + \dots + |x_n|^2$ ,  $x^T \rightarrow \overline{x^T} = x^*$ . 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  $\Sigma$  diagonal with  $\sigma_i$  on the diagonal with  $\sigma_1 \geq \dots \geq \sigma_n \geq 0$ .

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  $[u_1 | u_2 | \dots | u_m] \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_m \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_m^T \end{bmatrix} = u_1 \sigma_1 v_1^T + \dots + u_m \sigma_m v_m^T$  sum of  $m$  rank-1 matrices

Geometric idea: there is an orthogonal basis  $v_1, \dots, v_m$  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  $\sigma_1 \geq \dots \geq \sigma_{\min(m,n)} \geq 0$

So only the first  $n$  vectors of  $U$  matches with non-zero values in  $\Sigma$ , all the last  $n - m$  columns combine with zeroes.  
 $= u_i \sigma_1 v_1^T + \dots + u_n \sigma_n v_n^T$  with  $u_{n+1}, \dots, u_m$  not used. So we can "chop off" the unused parts and get the same result.  
 $= u_i \sigma_1 v_1^T + \dots + u_{\min(m,n)} \sigma_{\min(m,n)} v_{\min(m,n)}^T$

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  $\sigma_i$ .  $\sigma_1 \geq \dots$  so at one point a  $\sigma_r > \sigma_{r+1} = \dots = \sigma_{\min(m,n)} = 0$

Given  $A = U \Sigma V^T$  we can compute  $A^T A = (U \Sigma V^T)^T (U \Sigma V^T) = V \Sigma^T U^T U \Sigma V^T = V \Sigma^T \Sigma V^T$  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  $A^T A$  are the squares of the singular values of  $A$  plus additional zeroes for dimension reasons.

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

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

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

**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  $A = u^T v$ , with  $K = 2$  then  $A = u_1^T v_1 + u_2^T v_2$ . What is "how close":  $\min_{\text{rank}(X) \leq K} \|A - X\|$ . The theorem states that the solution is related to SVD.

The optimal solution of  $\min_{\text{rank}(X) \leq K} \|A - X\|$  is  $X = u_1 \sigma_1 v_1^T + \dots + u_k \sigma_k v_k^T$  where  $A = u_1 \sigma_1 v_1^T + \dots + u_{\min(m,n)} \sigma_{\min(m,n)} v_{\min(m,n)}^T$  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

$X_1 = u_i \sigma_1 v_1^T$  = best approximation score of student  $i \cdot n$ . best approximation difficulty of exercise  $j$

As a statistical estimator: suppose my scores are of the form  $A_{ij} = u_i v_j + \epsilon_{ij}$  with  $\epsilon_{ij}$  being the error in the score for instance gaussian with variance  $\lambda$ .

The rank 1 approx of  $(u_1 \sigma_1)$ ,  $v_1$  given by SVD is the one that minimizes  $\sum (A_{ij} - u_i v_j)^2 = \|A - X_1\|_F^2 = \sum \epsilon_{ij}^2$ . This

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

The best rank 2 approximation is  $X_2 = u_1\sigma_1v_1^T + u_2\sigma_2v_2^T$  which can be viewed as first approximation plus corrections.

$\sigma_1 \gg \sigma_2, \dots$  then  $A$  very close to rank 1.

$\sigma_1, \sigma_2 \gg \sigma_3, \dots$  then  $A$  very close to rank 2, and so on.

**Best approximations**  $X$  = best rank-1 approx of  $I$ ,  $X = u_1\sigma_1v_1^T$ ,  $x_{ij} = (u_1)_i\sigma_1(v_1^T)_j$

Best rank-2  $X_2 = u_1\sigma_1v_1^T + u_2\sigma_2v_2^T$

Best rank-3  $X_3 = u_1\sigma_1v_1^T + u_2\sigma_2v_2^T + u_3\sigma_3v_3^T$  And so on...

The original image was  $256 \times 256 = 2^{16}$  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  $\|A - X_k\|_F = \sqrt{\sum (a_{ij} - x_{ij})^2} = \|U\Sigma V^T - U[\text{main diagonal of } \sigma_i \text{ until } \sigma_k]V^T\| = \|U([\dots = \sqrt{\sum_{i=k+1}^{\min(m,n)} \sigma_i^2}]V^T)\|$

**Linear Least Squares problems** Given vectors  $a_1, \dots, a_n \in R^m$ , so that  $A = [a_1 | \dots | a_n] \in R^{m \times n}$ , and targets  $b \in R^m$ , find  $x_1, \dots, x_n \in R$  |  $a_1x_1 + \dots + a_nx_n = b$

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 if  $A$  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  $\min_{coef} \|Ax - y\|_2^2$

**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  $\text{Ker}(A) = \{0\} \Leftrightarrow$  there is no  $z \in R^n, z \neq 0$  |  $Az = 0 \Leftrightarrow rk(A) = n$  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  $\Leftrightarrow A^T A$  is positive definite.

**Proof**  $Ax \neq 0 \forall z \in R^n, z \neq 0$

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

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

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

$$\Leftrightarrow z^T A^T A z \neq 0 \forall z \in R^n, z \neq 0 \leftarrow \text{definition of } A^T A > 0$$

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

$$\min \|Ax - b\|_2 = \min x^T A^T A x - 2b^T A x + b^T b \Leftrightarrow f(x) = x^T Q x + q^T x + c$$

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
- $\Leftrightarrow 2A^T A = Q \succ 0$  (positive definite)
- $\Leftrightarrow A^T A > 0 \Leftrightarrow A$  has full column rank (for the lemma)

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

### Algorithm

1. Form  $A^T A$ ,  $n \times m \cdot m \times n$  product so it costs  $2mn^2$  floating point operations (flops) plus lower order terms
2. Form  $A^T b$ , costs  $2mn$  flops plus lower order terms
3. Solve  $A^T Ax = A^T b$  (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.  $A^T A$  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  $A^T A$  is positive definite (example: Cholesky factorization, complexity is  $\frac{1}{3}n^3$  flops, half the cost)

**Pseudoinverse**  $x = A^T A^{-1} b$  can be denoted as the product of  $A^+ = A^T A^{-1} A^T$  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  $A^+ A = (A^T A)^{-1} (A^T A) = I \in R^{n \times n}$ , while  $AA^+ = A(A^T A)^{-1} A^T \neq I \in R^{m \times m}$ . 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,  $\|x\| = \left\| \begin{bmatrix} s \\ 0 \\ \vdots \\ 0 \end{bmatrix} \right\| = |s| \Rightarrow s = \pm \|x\|$ . We will find  $H$  in a very specific form

$$\begin{aligned} H &= I - \frac{2}{v^T v} v v^T = \\ &= I - \frac{2}{\|v\|^2} v v^T = \\ &= I - 2uu^T \end{aligned}$$

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  $= I - 4uu^T + 4uu^T = I$

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  $x_1 - s = x_1 + \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$ , 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  $x_1 < 0$ , 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  $| A = QR$

When  $n = 1$  we already solved:  $\forall x \in R^{m \times 1} \exists$  a Householder reflector  $H | Hx = \begin{bmatrix} s \\ 0 \\ \vdots \\ 0 \end{bmatrix} \Leftrightarrow x = H \begin{bmatrix} s \\ 0 \\ \vdots \\ 0 \end{bmatrix}$  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.  $[u_1, s_1] = \text{householder\_vector}(A(:, 1))$   $H_1 = I - 2u_1u_1^T$

$$H_1 A = \begin{bmatrix} s_1 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} = A_1$$

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  $[u_2, s_2] = \text{householder\_vector}(A_1(2:m, 2))$ , which the 2nd to  $m$ th 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  $[u_3, s_3] = \text{householder\_vector}(A_2(3:m, 3))$  we do

$$Q_3 A_2 = \left[ \begin{array}{cc|ccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & & & \\ 0 & 0 & & & \\ 0 & 0 & & & \end{array} \right] A_2 = \left[ \begin{array}{cccc} s_1 & * & * & * \\ 0 & s_2 & * & * \\ 0 & 0 & s_3 & * \\ 0 & 0 & 0 & * \\ 0 & 0 & 0 & * \end{array} \right] = A_3$$

4.  $H_4 = I - 2u_4u_4^T$  from  $[u_4, s_4] = \text{householder\_vector}(A_3(4:m, 4))$

$$Q_4 A_3 = \left[ \begin{array}{c|c} I_3 & 0 \\ \hline 0 & H_4 \end{array} \right] A_3 = \left[ \begin{array}{cccc} s_1 & * & * & * \\ 0 & s_2 & * & * \\ 0 & 0 & s_3 & * \\ 0 & 0 & 0 & s_4 \\ 0 & 0 & 0 & 0 \end{array} \right] = A_4$$

So  $Q_4 Q_3 Q_2 Q_1 A = R$  is an upper triangular matrix.

$$Q_4^T Q_4 Q_3 Q_2 Q_1 A = Q_4^T R$$

$$Q_3^T Q_3 Q_2 Q_1 A = Q_3^T Q_4^T R$$

$$Q_2^T Q_2 Q_1 A = Q_2^T Q_3^T Q_4^T R$$

$$Q_1^T Q_1 A = Q_1^T Q_2^T Q_3^T Q_4^T R$$

So  $A = Q_1^T Q_2^T Q_3^T Q_4^T R$ , 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  $m \geq n$ ,  $b \in R^m$ .

$A = QR = [Q_1 | Q_2] \cdot \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$  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

$$\|Ax - b\| = \|QRx - b\| = \|Q^T(QRx - b)\| = \|Rx - Q^T b\| = \left\| \begin{bmatrix} R_1 \\ 0 \end{bmatrix} x - \begin{bmatrix} Q_1^T b \\ Q_2^T b \end{bmatrix} \right\| = \left\| \begin{bmatrix} R_1 x - Q_1^T b \\ -Q_2^T b \end{bmatrix} \right\|$$

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

1. Compute the QR factorization  $A = QR = Q_1 R_1$  (thin is enough) in  $\frac{4}{3}n^3, 2mn^2$  operations.
2. Compute  $c = Q_1^T b$ , with  $2mn$  operations
3. Solve the linear system  $R_1 x = c$  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  $\Leftrightarrow A^T A$  is positive definite  $\Leftrightarrow A^T A$  invertible  $\Leftrightarrow R_1$  invertible

$$A^T A = (QR)^T QR = R^T Q^T QR = [R_1^T 0] \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = R_1^T R_1$$

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

$$\|Ax - b\| = \|USV^T x - b\| = \|U^T(USV^T x - b)\| = \|SV^T x - U^T b\| =$$

With  $S = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ & & & 0 \end{bmatrix}$  and  $V^T x = y$  we have

$$= \left\| \begin{bmatrix} \sigma_1 y_1 \\ \vdots \\ \sigma_n y_n \\ 0 \\ \vdots \\ 0 \end{bmatrix} - \begin{bmatrix} u_1^T b \\ \vdots \\ u_m^T b \end{bmatrix} \right\| = \left\| \begin{bmatrix} \sigma_1 y_1 - u_1^T b \\ \vdots \\ \sigma_n y_n - u_n^T b \\ -u_{n+1}^T b \\ \vdots \\ -u_m^T b \end{bmatrix} \right\|$$

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

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

$$x = VS_1^{-1}U_1^T b = A^+ b$$

The solution of  $\min \|Ax - b\|$  is unique  $\Leftrightarrow A^T A$  is invertible  $\Leftrightarrow A^T A = (USV^T)^T USV^T = V S^T U^T U S V^T =$

$$V \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_n^2 \end{bmatrix} V^T \text{ which is the eigenvalue decomposition for } A^T A$$

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

$$\left\| \begin{bmatrix} \sigma_1 y_1 - u_1^T b \\ \vdots \\ \sigma_r y_r - u_r^T b \\ -u_{r+1}^T b \\ \vdots \\ -u_n^T b \\ -u_{n+1}^T b \\ \vdots \\ -u_m^T b \end{bmatrix} \right\|$$

What happens if  $\sigma_1 \geq \sigma_2 \geq \dots \sigma_r > \sigma_{r+1} = \dots \sigma_n = 0$ ? We would have

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  $y_1 = \frac{u_1^T b}{\sigma_1}, \dots, y_r = \frac{u_r^T b}{\sigma_r}, y_{r+1} = 0, \dots, y_n = 0$

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  $\sigma_{r+1}$  gives a huge contribution to the sum. In many case, stopping the sum early is beneficial: **truncated SVD**, with  $i = 1, \dots, k$  and  $k < m$ . The Eckart-Young approximation theorem can help too.

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

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

$$\bar{\sigma}_1, \dots, \bar{\sigma}_n = \text{SVD}(\bar{A})$$

Then  $|\sigma_i - \bar{\sigma}_i| \leq \|E\|_2$

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  $\min_{x \in R^n} \|Ax - b\|^2 + \lambda^2 \|x\|^2 = \min \left\| \begin{bmatrix} A \\ \lambda I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|^2$  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  $\delta$  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  $f : R^m \rightarrow R^n$  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  $K_{abs}(f, x) = \lim_{\delta \rightarrow 0} \sup_{\|d\| \leq \delta} \frac{\|f(x+d) - f(x)\|}{\|d\|} K_{abs}(f, x) = \|\nabla_x f\|$  2-norm of the gradient, if  $f$  differentiable.

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

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

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \delta \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 + \delta \\ x_2 + \delta \end{bmatrix}$$

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

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  $\|\nabla_x f\|$ )

**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  $\bar{b} \in R^n$  and call  $\bar{x}$  the solution of  $A\bar{x} = \bar{b}$ , while  $x$  solution of  $Ax = b$ . The absolute bound is

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

and

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

Putting all together gives

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

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

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

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

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,  $\min_{B \text{ sing}} \|A - B\| = \min_{rk(B) \leq n-1} \|A - B\| = \sigma_n$  because  $B = \sum_{i=1}^{n-1} u_i \sigma_i v_i^T$ . Hence  $\frac{1}{K(A)} = \frac{\sigma_n}{\sigma_1} = \frac{\min_{B \text{ sing}} \|A - B\|}{\|A\|}$

**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  $A \in R^{m \times n}, b \in R^m$

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

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

where

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

## 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  $(x_1, y_1), \dots, (x_m, y_m)$ . 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  $k \ll n, m$ , 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  $y_h = -1$

The **maximum margin separating hyperplane** is the solution of  $\min_w \{ \|w_+\|^2 | y_h(w_+ x_h + w_0) \geq 1 \} \quad h = 1, \dots, m$ , 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  $f : X \rightarrow \mathbb{R}$  any function, we have an optimization problem in the form

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

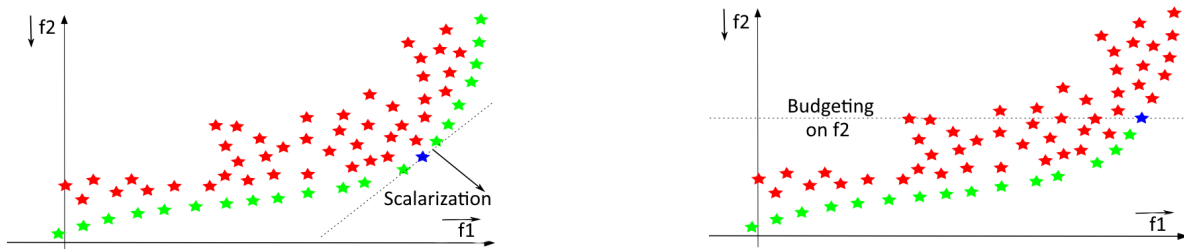
with  $X$  **feasible region** and  $f$  **objective function**.

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

$x \in X$  is the **feasible solution**, often  $X \subset F, x \in F - X$  **unfeasible solution**. We want (any) optimal solution  $x_* \in X \mid \forall x \in X \quad f(x_*) \leq f(x)$  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  $\alpha$ ? **Budgeting**: bound the value of one of the losses to the others:  $\min\{f_1(x) \mid f_2(x) \leq \beta_2, x \in X\}$ , but which  $\beta_2$ ?



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  $\epsilon$ -approximate solutions ( $\epsilon$ -optima)

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

and on computers  $x \in \mathbb{R}$  is actually  $x \in \mathbb{Q}$  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  $X$ :

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

$f$  L-c  $\Rightarrow$  doesn't "jump" and one  $\epsilon$ -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 than  $\Omega(\frac{LD}{\epsilon})$ , 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  $\epsilon$ . Dramatically worse with  $X \subset \mathbb{R}^n$ .

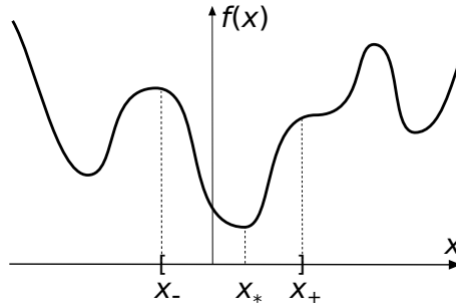
Also generally  $L$  is unknown and not easy to estimate, but algorithms actually require/use it.

## 2.1.2 Local Optimization

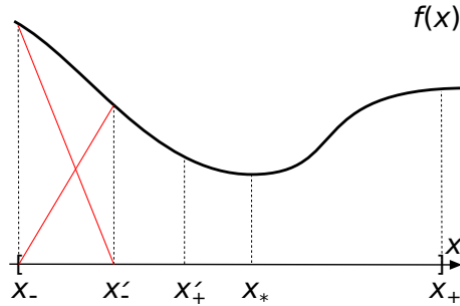
Even if I stumble in  $x_*$  how do I recognize it? This is the difficult thing. It's simpler to start with a weaker condition:  $x_*$  is the local minimum if it solves  $\min\{f(x) \mid f \in X(x_*, \epsilon) = [x_* - \epsilon, x_* + \epsilon]\}$  for some  $\epsilon > 0$ .

Stronger notion: **strict** local minimum if  $f(x_*) < f(y) \quad \forall y \in X(x_*, \epsilon) - \{x_*\}$

$f$  (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  $\exists \epsilon > 0 \mid f$  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**

```

1 procedure x = GRS(f, x1, xr, delta)
2   x12 = x1 + (1-r)(xr-x1)
3   xr2 = x1 + r(xr - x1)
4   compute f(x12), f(xr2)
5   while (xr - x1 > delta):
6     if (f(x12) > f(xr2)):
7       x1 = x12
8       x12 = x
9       x = xr2
10      xr2 = x1 + r(xr - x1)
11      compute f(xr2)
12   else:
13     xr = xr2
14     xr2 = x
15     x = x12
16     x12 = x1 + (1-r)(xr-x1)
17     compute f(x12)

```

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  $\delta$ .

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  $\Rightarrow k \geq \frac{\log \frac{S}{\epsilon}}{\log \frac{1}{r}}$

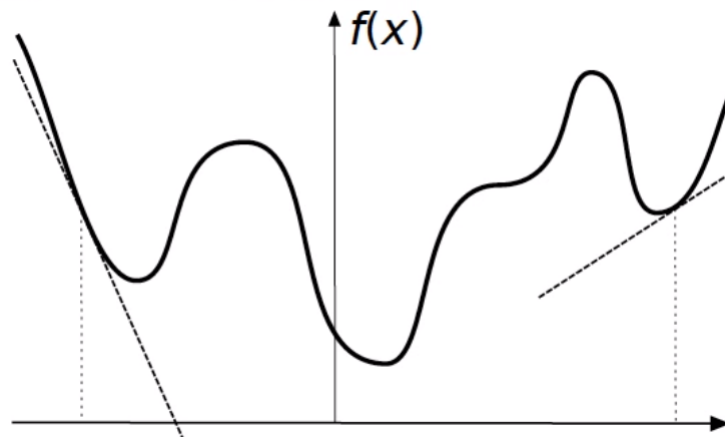
$O(\log(\frac{1}{\epsilon}))$  good, but the constant  $\rightarrow \infty$  as  $r \rightarrow 1$

**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  $\arg\max\{f(x_-), f(x_+)\}$ , 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 transcendental, 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**.

```

1 procedure x = DS(f, xl, xr, eps)
2   while (true) do: # invariant: df(xl) < -eps, df(xr) > eps
3     x = in_middle_of(xl, xr)
4     compute df(x)
5     if (abs(df(x)) <= eps): break
6     if (df(x) < 0):
7       xl = x
8     else:
9       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 ( $\lim_{|x| \rightarrow \infty} f(x) = \infty$ )

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

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

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

**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  $f$ :  $f(x_-), f(x_+), f'(x_-), f'(x_+) \dots$ . 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

$$\{x_i\}$$

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

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

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

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

We have convergence when  $\{a_i\} \rightarrow 0, \{r_i\} \rightarrow 0 \Leftarrow \{d_i\} \rightarrow 0$  (but  $\nRightarrow$ ), but how rapidly? **Rate of convergence**

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

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

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

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

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

$r < 1 \Rightarrow$  linear,  $r_i \rightarrow i \in O(\log(\frac{1}{\epsilon}))$ , 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".

$f \in C^3, f'(x_*) = 0 \wedge f''(x_*) \neq 0 \Rightarrow \exists \delta > 0 \mid x_0 \in [x_* - \delta, x_* + \delta] \Rightarrow \{x_i\} \rightarrow x_*$  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  $L'_i(x) = 0 \simeq f'(x) = 0 \Rightarrow x = x_i - \frac{f'(x_i)}{f''(x_i)}$

```
1 procedure x = NM(f, x, eps)
2   while (abs(df(x)) > eps):
3     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  $x_*$ .

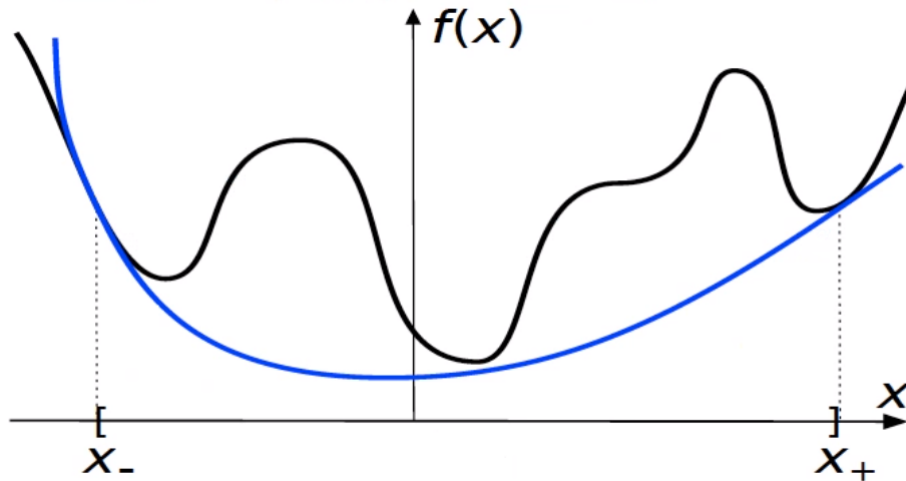
If we get within a short enough distance  $\delta$ , then it will converges extremely fast with  $p = 2$ . Mathematically,  $f \in C^3, f'(x_*) = 0 \wedge f''(x_*) \neq 0 \Rightarrow \exists \delta > 0 \mid x_0 \in [x_* - \delta, x_* + \delta] \Rightarrow \{x_i\} \rightarrow x_*$  with  $p = 2$ .

## 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  $\nRightarrow C^1$ . 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  $\bar{x}$  giving  $\underline{f}(\bar{x}) \leq f_* \leq f(\bar{x})$ . If the gap  $f(\bar{x}) - \underline{f}(\bar{x})$  is too large, we partition  $X$  and iterate. If on some partition  $\underline{f}(\bar{x}) \geq$  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  $f : R^n \rightarrow R$ , i.e.  $f(x_1, x_2, \dots, x_n) = f(x)$  with  $x = [x_i]_{i=1}^n = [x_1, \dots, x_n] \in R^n$ . Note that  $R^n = R \times R \times \dots \times R$ , which is **exponentially larger** than  $R$ .

$I = [x_-, x_+]$ ,  $X = I \times I \times \dots \times I$  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  $O(\cdot)$  constraints. Yet, large scale optimization can be done.

### Notation

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

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

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

**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  $f : D \rightarrow R$  with  $D = \text{dom}(f)$  domain of  $f$  which may not be all  $R^n$ , but usually ok to ignore  $\text{dom}(f)$  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  $x_* \in S(f, v) \quad \forall v \geq f_*$  and  $S(f, v) = \emptyset \quad \forall v < f_*$

**Tomography**  $f : R^n \rightarrow R$  with  $x \in R^n$  origin and  $d \in R^n$  direction. You can define  $\phi_{x,d}(\alpha) = f(x + \alpha d) : R \rightarrow R$   
**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

$f_x^i(\alpha) = f(x_1, \dots, x_{i-1}, \alpha, x_{i+1}, \dots, x_n) = \phi_{0,u^i}(\alpha)$  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  $\Rightarrow$  steeper. Collinear means steepest line, less collinear means less steep.  $90^\circ$  angle means a flat line. Decreasing if opposite directions.

min  $f(x)$  when  $\beta x_*$  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  $f(x) = \frac{1}{2}x^T Q x + qx$ . If  $q = 0$  (no linear term), then **homogeneous quadratic**.

Tomography  $\phi(\alpha) = f(\alpha d) = \alpha^2(d^T Q d) \Rightarrow$  sign and steepness depend on  $d^T Q d$ , so we need to know about signs of  $d^T Q d$ . 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  $\lambda_i \neq 0 \quad \forall i$ , then  $f(x) = \frac{1}{2}(x - x_*)^T Q (x - x_*) + c$  for  $x_* = -Q^{-1}q$  and  $x_* \neq 0$  center of the level sets (which shapes are determined by the eigenvalues).

$y = x - x_*, f_*(y) = y^T Q y + c$

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

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

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

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

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

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

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

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

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

$\psi$  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  $f : R^n \rightarrow R^m$ ,  $f(x) = [f_1(x), f_2(x), \dots, f_m(x)]$ , the partial derivatives are the same with extra index

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

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

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

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

**Hessian** The  $\frac{\partial f}{\partial x_i} : R^n \rightarrow R$  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  $\nabla f(x) : R^n \rightarrow R^n$  have a Jacobian and it's called **Hessian** of  $f$  at  $x$

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

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  $x$

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

$\Rightarrow$  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 ( $\neq$ ): to tell them apart we need to look at the curvature of  $f$ . If  $f$  quadratic I would 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  $x_i \rightarrow x_{i+1}$  to get a **sequence**  $\{x_i\}$  that **should go towards an optimal solution**.  $\{x_i\} \rightarrow x_*$  is one option, the best one, but not the only possibility.

At least  $\{f_i = f(x_i)\} \rightarrow f_*$ : **minimizing sequence**, and clearly  $\{x_i\} \rightarrow x_* \Rightarrow \{f_i\}$  minimizing sequence, but  $\nRightarrow$   
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:  $x_{i+1} \in \operatorname{argmin}\{L_i(x) : x \in R^n\} = \emptyset$ ,  $L_i$  unbounded below on  $R^n$

It shouldn't move too far from  $x_i$ ,  $L_i$  is only "good" as  $\alpha_i \rightarrow 0 \Rightarrow d_i = \operatorname{argmin}\{\lim_{t \rightarrow 0} \frac{f(x_i + td)}{t}\} = -\nabla f(x_i)$  = 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  $\alpha$  grows  $\Rightarrow f$  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  $\alpha_i$  efficiently (a few function evaluations). Two extreme strategies:

**Fixed Stepsize** (FS):  $\forall i \quad \alpha_i = \bar{\alpha}$  (how is chosen?)

Most inexpensive.

**(Exact) Line Search** (LS):  $\alpha_i \in \operatorname{argmin}\{f(x_i + \alpha d_i) \mid \alpha \geq 0\}$

Most expensive but may converge faster.

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

**Gradient for quadratic functions**  $f(x) = \frac{1}{2}x^T Q x + q x$  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  $\alpha_i = \frac{\|d_i\|^2}{d_i^T Q d_i}$

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

With `nablf` being  $\nabla f$

**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 ( $\lambda_1, \lambda_n$  being respectively the max and min eigenvalues of  $Q$ )

$$\text{Simple } A(x_{i+1}) \leq (1 - \frac{\lambda_n}{\lambda_1}) A(x_i)$$

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

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  $r \rightarrow 1$  as conditioning of  $Q = \frac{\lambda_1}{\lambda_n} \rightarrow \infty$

**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  $\lambda_1 \gg \lambda_n$ . Intuitively, the algorithm zig-zags a lot when level sets are very elongated. Another bad news is that there may be an "hidden dependency":  $\lambda_1$  and  $\lambda_n$  may depend on  $n$  and  $\frac{\lambda_1}{\lambda_n}$  may grow as  $n \rightarrow \infty$ . Let's extend it to every function.

## Gradient methods for general functions

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

```

1 procedure x = SDQ(f, x, eps):
2   while (||nablf(x)|| > eps):
3     d = -nablf(x)
4     alpha = stepsize(f, x, d)
5     x = x + alpha*d

```

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

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

Charybdis:  $\alpha_i$  not too small to avoid stalling

With  $\frac{\partial f}{\partial d_i}(x_i) < 0$  hence  $\alpha_i \rightarrow 0$  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

$$\begin{aligned} \phi'_i(\alpha) &= \frac{\partial f}{\partial d_i}(x_i + \alpha d_i) = \langle \nabla f(x_i + \alpha d_i), d_i \rangle \\ \Rightarrow |\phi'_i(\alpha_i)| &= |\langle d_i, \nabla f(x_{i+1}) \rangle| = |\langle \nabla f(x_i), \nabla f(x_{i+1}) \rangle| \end{aligned}$$

The good news is that only an approximate stationary point of  $\phi_i$  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  $\epsilon'$

**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  $\epsilon$  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 if  $A(x_i)$  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  $\{x_i\} \rightarrow x_* \Rightarrow \{f_i\}_{i \geq k} \rightarrow f_*$  linearly for large enough  $k$ , with  $r = \frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}^2$  with  $\lambda_1, \lambda_n$  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  $\alpha_i = \bar{\alpha}$  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  $\{\|d_i\|\} \rightarrow 0$ , so care is still required. We also have that  $\alpha_i \rightarrow 0$  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  $\alpha_i$  is fixed.

$d_i = \frac{-\nabla f(x_i)}{\|\nabla f(x_i)\|} \equiv \|d_i\| = 1$  would necessarily require  $\alpha_i \rightarrow 0$ , yet  $f$  varies very rapidly so only very short  $\alpha_i$  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  $\alpha$  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**  $\lambda_n \equiv \lambda_n$ -convex.

$f$  **strongly convex modulus**  $\tau > 0$  ( $\tau$ -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  $\tau$ -convex  $\equiv \tau I \preceq \nabla^2 f \preceq LI \equiv \tau \leq \lambda_n \leq \lambda_1 \leq L$ , 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."

$$L\text{-smooth} \Rightarrow a_{i+1} \leq a_i - \frac{\|\nabla f(x_i)\|^2}{2L} \Rightarrow a_{i+1} \leq a_i(1 - \frac{\tau}{L})$$

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  $0 < m_1 < 1$

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

$\alpha \rightarrow 0$  satisfies (A). But if we avoid Charybdis then we "converge".

$\alpha_i \geq \bar{\alpha} > 0$  and (A) holds  $\forall i \Rightarrow$  either  $\{f_i\} \rightarrow -\infty$  or  $\{\|\nabla f(x_i)\|\} \rightarrow 0$

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  $\forall i \Rightarrow$

$$f_{i+1} \leq f_i + m_1 \alpha_i \phi'_i(0) \leq f_i - m_1 \bar{\alpha} \epsilon \Rightarrow f_i \leq f_0 - m_1 \bar{\alpha} \epsilon \Rightarrow \{f_i\} \rightarrow -\infty$$

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



**Wolfe Goldstein condition**  $m_1 < m_2 < 1$

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

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

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

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

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

We have that  $(A) \cap (W)$  captures all local minima (and maxima), usually  $m_1$  close to 1, and  $(A) \cap (W')$  ensures  $\phi'(\alpha) \not\approx 0$

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  $(A)$

```

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

```

With  $dphi(0)$  meaning  $\phi'(0)$

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

$$\bar{\alpha}_i \geq \bar{\alpha} > 0 \forall i \Rightarrow \exists h \mid \alpha \geq \tau^h \quad \forall i \Rightarrow \text{convergence}$$

We need conditions on  $f$  to get it:

$f$   $L$ -smooth  $\Rightarrow \phi$  is  $[L\|d\|^2]$ -smooth

$$\Rightarrow -\phi'(0) = \|d\|^2 = \|\nabla f(x)\|^2$$

$\phi$  is  $[L\|d\|^2]$ -smooth  $\Rightarrow \alpha'$  and  $\bar{\alpha}$  are "large":

$$L\|d\|^2(\alpha - 0) \geq \phi'(\alpha') - \phi'(0) > (1 - m_3)(-\phi'(0)) = (1 - m_3)\|d\|^2$$

$$\Rightarrow \bar{\alpha} > \alpha' > \frac{1-m_3}{L}$$

If  $f$  also  $\tau$ -convex  $\Rightarrow$  convergence linear with  $r \simeq \frac{1-\tau}{L}$ , depending on  $m_1, m_3$

Might be rather slow, need something better.