1 15th of November 2018 — F. Poloni

1.1 Stability of algorithms

In this lecture we will try to answer the questio: "Is our algorithm (using floating point) going to compute a good approximation of the answer?"

It is related to sensitivity/conditioning but different. Depends on how we perform the computation.

Do you recall?

Computers work with IEEE arithmetic and the basic idea is the following.
TL;DR: floating point numbers are numbers in base-2 scientific (exponential) notation.
`double` (64-bit numbers):

$$\pm 1.\underbrace{01001011101\ldots101}_{\text{52 binary digits}} \cdot 2^{\pm \underbrace{101\ldots01}_{\text{10 binary digits}}}.$$

We use 1 bit for the sign, 52 bits for the "mantissa" and 11 bits for the exponent and its sign.
Some of these combinations of bits are reserved for special numbers, e.g. `Inf` and `NaN`, `-0`.

This system is subject to approximation errors, exactly like the "usual" decimal arithmetic: for example, if we do $\frac{1}{3} = 0.33333\ldots$ and if we do $\frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 0.99999\ldots \neq 1$.

Whenever we store a number on a computer we need to approximate it, using the representable numbers, as shown in Figure 1.1.

FIGURE 1.1: We have 2^{52} equispaced numbers between $\frac{1}{2}$ and 1 and between 1 and 2, and also 2^{52} between 2 and 4 and so on and so forth, so we have the same number of integers, although the space is enlarging.

Not all numbers are exactly representable, take 0.1 (decimal). It's a periodic number when written in binary, hence we can't represent it exactly as a machine number.

Definition 1.1 (Error bound)**.** *For each $x \in \pm[10^{-308}, 10^{308}]$, there is an exactly representable number $\tilde{x}$ such that $\frac{|\tilde{x}-x|}{|x|} \leq \mathtt{u}$, with $\mathtt{u} = 2^{-52} \approx 2 \cdot 10^{-16}$.*

Let us assume that we have the best possible algorithm that returns $\tilde{y} = f(\tilde{x})$ which is the best representation of $f(\tilde{x})$, then

$$\begin{aligned}\frac{|\tilde{y}-y|}{|y|} &\leq \kappa_{rel}(f,x)\frac{|\tilde{x}-x|}{|x|} + o(\frac{|\tilde{x}-x|}{|x|}) \\ &\leq \kappa_{rel}(f,x)\mathsf{u} + o(\mathsf{u}).\end{aligned}$$

In practice we may ignore $o(u)$, since it's small o of 2^{-16}.

When we are approximating a result, we are approximating it up to a relative error of order of magnitude 10^{-16}.

$$a \oplus b = (a+b)(1+\delta), \quad |\delta| \leq \mathsf{u}$$

$$\Updownarrow$$

$$\frac{|(a \oplus b) - (a+b)|}{|a+b|} = \delta$$

We would like to compute the error on the function $f(a,b) = a^T b$.

$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \; so \; f(a,b) = a_1b_1 + a_2b_2 + a_3b_3$$

Denoting $\odot$ the approximated product and $\oplus$ the approximated sum on a computer, we can write the following:

$$\begin{aligned}\text{computer result} &= a_1 \odot b_1 \oplus a_2 \odot b_2 \oplus a_3 \odot b_3 \\ &= \Big[\Big[a_1b_1(1+\delta) + a_2b_2(1+\delta_2)\Big](1+\delta_3) + a_3b_3(1+\delta_4)\Big](1+\delta_5) \\ &= a_1b_1(1+\delta_1)(1+\delta_3)(1+\delta_5) + a_2b_2(1+\delta_2)(1+\delta_3)(1+\delta_5) \\ &+ a_3b_3(1+\delta_4)(1+\delta_5) \\ &= a_1b_1(1+\delta_1+\delta_3+\delta_5+O(u^2)) \\ &+ a_2b_2(1+\delta_1+\delta_3+\delta_5+O(u^2)) \\ &+ a_3b_3(1+\delta_4+\delta_5+O(u^2)) \\ &\approx a_1b_1(1+\delta_1+\delta_3+\delta_5) + a_2b_2(1+\delta_1+\delta_3+\delta_5) + a_3b_3(1+\delta_4+\delta_5)\end{aligned} \tag{1.1}$$

Where $O(u^2)$ comes from the summation of $\delta_i\delta_j$, for some i, j and allows us to do an approximation up to second order terms of precision.

The absolute error then is:

$$\begin{aligned}\mathrm{Err}_a &= |a_1b_1(\cancel{1} + \delta_1 + \delta_3 + \delta_5) + a_2b_2(\cancel{1} + \delta_1 + \delta_3 + \delta_5) + a_3b_3(\cancel{1} + \delta_4 + \delta_5) + \cancel{a_1b_1} + \cancel{a_2b_2} + \cancel{a_3b_3}| \\ &\overset{(1)}{\leq} |a_1b_1|3u + |a_2b_2|3u + |a_3b_3|2u \\ &\leq (|a_1b_1| + |a_2b_2| + |a_3b_3|)3u\end{aligned} \tag{1.2}$$

Where $\overset{(1)}{\leq}$ follows from the observation that $|\delta_i| \leq u$.

The result that we can obtain is weaker than we would have expected: *if* $a_ib_i \geq 0, \forall i = 1, 2, 3$, then

$$\frac{|\text{computer result} - (a_1b_1 + a_2b_2 + a_3b_3)|}{a_1b_1 + a_2b_2 + a_3b_3} \leq 3u$$

which means that the algorithm is stable.

However, if a_1b_1, a_2b_2 and a_3b_3 have different signs, then we can bound the error not with $a_1b_1 + a_2b_2 + a_3b_3$, but only with $|a_1b_1| + |a_2b_2| + |a_3b_3|$. This might be a lot larger than what we want to compute.

Example 1.1. *Take* $\varepsilon = 10^{-16}$. *Compute* $\begin{pmatrix}1 & -1 & 0\end{pmatrix}\begin{pmatrix}1+\varepsilon \\ 1 \\ 1\end{pmatrix} = (1+\varepsilon) - 1 + 0 = \varepsilon$.

In this case we have a subtraction between two very close numbers.

$|$*computer result*$| \leq u(i + |1(1+\varepsilon)| + |-1 \cdot 1| + |0 \cdot 1|) = (2+\varepsilon)u$, *which means that we have* 10 *correct digits.*

The problem is in the fact that we have a very small result ($\varepsilon = O(10^{-6})$) *and a very small error (of the same order of the result), which implies a large relative error.*

An attentive reader might have noticed that it's very long to make this computation. Therefore, since we computed it on easy examples, we have to observe that we can't afford it for SVD or other complicated methods.

Wilkinson trick (from the 60's) comes to help us, to simplify this computation for more complicated problems.

Idea: see the computer result as the exact output of an algorithm run on a slightly perturbed input.

$$\begin{aligned}\tilde{y} &= \dots \\ &= ((a_1b_1(1+\delta_1) + a_2b_2(1+\delta_2))(1+\delta_4) + a_3b_3(1+\delta_3))(1+\delta_5) \\ &= a_1\widetilde{b_1} + a_2\widetilde{b_2} + a_3\widetilde{b_3}\end{aligned} \tag{1.3}$$

where

$$\begin{aligned}\widetilde{b_1} &= b_1(1+\delta_1)(1+\delta_4)(1+\delta_5) = b_1 + 3ub_1 + o(u), \\ \widetilde{b_2} &= b_2(1+\delta_2)(1+\delta_4)(1+\delta_5) = b_2 + 3ub_2 + o(u), \\ \widetilde{b_3} &= b_3(1+\delta_3)(1+\delta_5) = b_3 + 2ub_3 + o(u),\end{aligned} \tag{1.4}$$

$$\widetilde{a_i} = a_i \quad i = 1, 2, 3.$$

And the relative error is $\frac{\|\widetilde{b}-b\|}{\|b\|} \leq 3u + o(u)$.
Hence,

$$\frac{\|\tilde{y}-y\|}{\|y\|} \leq \kappa_{rel,b}\frac{\left\|\widetilde{b}-b\right\|}{\|b\|} \leq k_{rel,b}3u$$

This isn't bad, because it's within a factor 3 of the optimal error for a perfect algorithm. Computing the conditioning is much easier than making all the calculations of δs.

Definition 1.2 (Backward stability of an algorithm)**.** *An algorithm to compute $\mathbf{y} = f(\mathbf{x})$ is called **backward stable** if the computed output $\tilde{\mathbf{y}}$ can be written as $\tilde{\mathbf{y}} = f(\tilde{\mathbf{x}})$, where $\tilde{\mathbf{x}} = \mathbf{x} + O(\mathsf{u}\,\|\mathbf{x}\|)$ (exact function, perturbed input).*

Observation 1.1. *In real-life usage, this $O()$ notation often hides polynomial factors in the dimension n. Although this may look an illicit simplification, we observe that these factors are much more harmless than the error that we could make otherwise.*

Theorem 1.1. *Backward stable algorithms are as accurate as theoretically possible (given the condition number of a problem), up to some factor that depends only on the dimension (e.g. n, $2n^2 + 18n$, ...).*

Proof.

$$\frac{\|\tilde{\mathbf{y}}-\mathbf{y}\|}{\|\mathbf{y}\|} \leq \kappa_{rel}(f,\mathbf{x})\frac{\|\tilde{\mathbf{x}}-\mathbf{x}\|}{\|\mathbf{x}\|} = \kappa_{rel}(f,\mathbf{x})O(\mathsf{u})$$

while the best attainable accuracy is $\kappa_{rel}(f,\mathbf{x})\mathsf{u}$. □

We may ask ourselves if it's possible to perturb the input in order to get $\widetilde{y}$ for every possible algorithm and the answer is no. Let us see a counterexample, where $f(a,b) = a^T b$ (vector-vector product in the order that produces a matrix).

We observe that in general it's not true that the computed approximation of $f(a,b)$ has rank 1.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} x_1y_1 & x_1y_2 & x_1y_3 \\ x_2y_1 & x_2y_2 & x_2y_3 \\ x_3y_1 & x_3y_2 & x_3y_3 \end{bmatrix} = M$$

While

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \odot \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} x_1 \odot y_1 & x_1 \odot y_2 & x_1 \odot y_3 \\ x_2 \odot y_1 & x_2 \odot y_2 & x_2 \odot y_3 \\ x_3 \odot y_1 & x_3 \odot y_2 & x_3 \odot y_3 \end{bmatrix} = \widetilde{M}$$

And we are looking for $\widetilde{x}$ and $\widetilde{y}$ such that $\widetilde{M} = \widetilde{x}^T\widetilde{y}$

Example 1.2. *Example (with exaggerated errors):*

$$\begin{bmatrix}1\\2\\3\end{bmatrix}\begin{bmatrix}4 & 5 & 6\end{bmatrix} = \begin{bmatrix}4 & 5 & 6\\8 & 10 & 12\\12 & 15 & 18\end{bmatrix}$$

While

$$\begin{bmatrix}1\\2\\3\end{bmatrix} \odot \begin{bmatrix}4 & 5 & 6\end{bmatrix} = \begin{bmatrix}4.01 & 4.99 & 6.01\\7.99 & 10.01 & 12.02\\11.98 & 15.02 & 17.97\end{bmatrix}$$

and this second matrix doesn't have rank 1.

1.2 Backward stability of QR factorization

Do you recall?

A generic step of the computation of the QR factorization of a matrix has the following shape:

$$\begin{pmatrix} \mathrm{I} & \\ & \mathrm{H}\ \mathbf{u}_k \end{pmatrix}\begin{pmatrix}*&*&*&*&*\\0&*&*&*&*\\0&0&*&*&*\\0&0&*&*&*\\0&0&*&*&*\end{pmatrix} = \begin{pmatrix}*&*&*&*&*\\0&*&*&*&*\\0&0&*&*&*\\0&0&0&*&*\\0&0&0&*&*\end{pmatrix}$$

Each step of the QR factorization in backward stable.

Let us compute the backward stability of QR factorization:

With some tedious computations like the ones above, one can show that *one* step of the QR factorization is backward stable, i.e., $\widetilde{R}_{k-1} = R_{k-1} + \Delta_{R_{k-1}}$, where $\left\|\Delta_{R_{k-1}}\right\| \leq O(u)\left\|R_{k-1}\right\|$ and this allows us to write the following

$$\frac{\left\|\widetilde{R}_{k-1} - R_{k-1}\right\|}{\|R_{k-1}\|} \leq O(u)$$

At a generic step k we have that:

$$R_k = Q_k R_{k-1}$$

$$\Updownarrow$$

$$\widetilde{R_k} = Q_k(R_{k-1} + \Delta_{R_{k-1}})$$

The idea is doing this procedure one step after the other, starting from step 1, where $R_0 = A$.

$$\widetilde{R_1} = Q_1 \widetilde{R_0} = Q_1(R_0 + \Delta_{R_0}) = Q_1(A + \Delta_{R_0})$$

$$\widetilde{R_2} = Q_2 \widetilde{R_1} = Q_2(R_1 + \Delta_{R_1}) = Q_2 Q_1(A + \Delta_{R_0}) + Q_2 \Delta_{R_1} = Q_2 Q_1(A + \Delta_{R_0} + {Q_1}^T \Delta_{R_1})$$

We go on combining errors like this and we see that all the quantities $(A + \Delta_{R_j})$ are perturbations of the original matrix A.

In the end we may observe that the final computed R_n is the exact result obtained from A plus n perturbations.

Observation 1.2. *We should notice that the norm of each perturbation is small with respect to the norm of A.*

Formally, $\|\Delta_{R_0}\| \leq u \|R_0\| = u \|A\|$ *and* $\left\|{Q_1}^T \Delta R_1\right\| \overset{(1)}{=} \|\Delta R_1\| \leq u \|R_1\| \overset{(2)}{=} u \|A\|$.

Where $\overset{(1)}{=}$ *and* $\overset{(2)}{=}$ *hold because Q_i are orthogonal.*

Mantra

Orthogonal transformations are the key for stability.

1.2.1 Stability of algorithms for least-squares problems

Let us see how various algorithms to solve LS problems (implemented in Matlab) behave in relation to backward stability.

Least squares problem via QR

STEP 1: Computing a thin QR (`qr(A, 0);`) → backward stable;

STEP 2: (`Q1' * b;`)→ backward stable;

STEP 3: (`R1  c;`)→ backward stable;

Scrivere meglio

Least squares problem via SVD

STEP 1: Computing a SVD (`svd(A, 0);`) → backward stable;

STEP 2: (`U' * b;`)→ backward stable;

STEP 3: (`c ./ diag(S);`)→ backward stable;

STEP 4: (`V * d;`)→ backward stable;

Least squares problem via Normal Equations

STEP 1: `C = A' * A;`

STEP 2: `d = A' * b;`

STEP 3: `x = C  d;`

We can also prove that some algorithms aren't backward stable, like normal equations. The issue here is that the same input is needed in more than one operation, so it should satisfy more than one equation and this may lead to a solution set which is empty. Also, in the last equation we solve a linear system with matrix C, and this gives an error of the order of $\kappa(A)^2$, never of $\kappa(A)$: $\kappa(C) = \kappa(A^T A) = \kappa(A)^2$.

Let us take the example of two lectures ago.

Example 1.3. *Let $A \in \mathcal{M}(4, 3, \mathbb{R})$ s.t.*

$$A = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 3 & 1 & 4 \\ 1 & 2 & 3 + 10^{-8} \end{pmatrix}$$

We may observe that A is at distance 10^{-8} from a matrix without full column rank, hence $\kappa \approx 10^8$.

What is the condition number of solving this least squares problem? It's about 10^8, since in that problem we generated $b = A \begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$, so b lies in the image of A ($Im(A)$). The geometric idea in Figure 1.2.

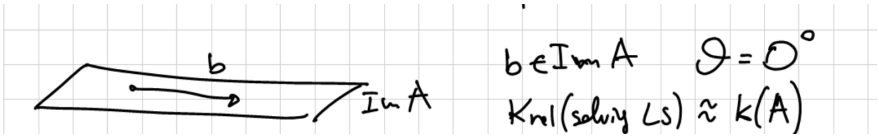

FIGURE 1.2: b lies (at least almost, because of numerical error) on the plane of $Im(A)$. In this case $\kappa_{rel}(\text{solving LS}) \approx \kappa(A)$.

A brief recap may be found in Table 1.

1.2.2 A posteriori checks

Let's assume that Matlab gives us a solution of a problem. We want to be able to check how good this result is.

Example 1.4. *Suppose we have solved a linear system...*

```
» A = randn(4, 4); b = randn(4, 1);
» x = A \ b;
» A * x - b
ans =
```

	Normal eqns	**QR**	**SVD**
$m \approx n$	$\frac{4}{3}n^3$	$\frac{4}{3}n^3$	$\approx 13n^3$
$m \gg n$	mn^2	$2mn^2$	$2mn^2$
	Unstable when *cond* $\approx \kappa(A)$	Backward stable	Backward stable; reveals info on sensitivity, allows regularization

TABLE 1: Brief recap of the complexities of the algorithms we studied for solving the least squares problem. The last row takes into account the stability.

```
           0
-1.3878e-17
           0
 2.2204e-16
```

Definition 1.3 (Residual)**.** *Let $A \in \mathcal{M}(m, \mathbb{R})$ and $\in\mathbb{R}^m$, and x be the solution of $Ax = b$. For a given $\widetilde{x}$ we define **residual** the following $\mathbf{r} = A\widetilde{x} - b$.*

Assume that $\|r\|$ is small; does this mean that $\widetilde{x}$ is close to the exact solution x?

Theorem 1.2. *Let $A \in \mathbb{R}^{m\times m}, \in\mathbb{R}^m$, and x be the solution of $Ax = b$.*

For a given $\widetilde{x}$, the relative error of $\widetilde{x}$ is bounded by the condition of matrix A times the ratio between the norm of the residual and the norm of b.

$$\frac{\|x - \widetilde{x}\|}{\|x\|} \leq \kappa(A)\frac{\|\mathbf{r}\|}{\|b\|}.$$

This theorem tells us that $\widetilde{x}$ is "close to the solution" apart from a factor which is the conditioning of matrix A.

Proof.
Follows from the perturbation results for linear systems. The idea is that $\widetilde{x}$ is the exact solution of the perturbed system:

$$A\widetilde{x} \overset{*}{=} b + \mathbf{r} = \widetilde{b}$$

Where $\overset{*}{=}$ follows from the definition of r and $\frac{\|\widetilde{b}-b\|}{\|b\|} = \frac{\|r\|}{\|b\|}$.
A relative perturbation of size $\frac{\mathbf{r}}{b}$ is amplified by $\kappa(A)$. □

It's important to notice that also computing $A \odot x \ominus b$ is an approximated operation. We choose to simplify things and ignore this error.

1.3 A posteriori check for Least Squares Problems

Can we make an analogous check for the Least Squares Problem?

Take a LSP $\min \|Ax - b\|$, with A tall thin, with full column rank.

The problem is that $\|Ax - b\|$ isn't small at all, indeed it could be as large as b, as you can see from Figure 1.3.

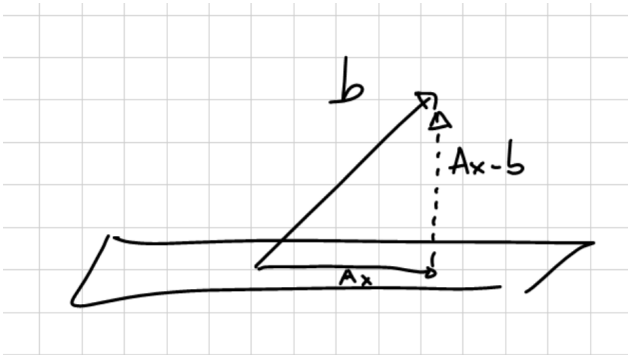

FIGURE 1.3: A can be as large as b if b is perfectly orthogonal.

Observation 1.3. *If you solve LSP via QR,* $\|Ax - b\| = \left\| \begin{pmatrix} R_1 x - {Q_1}^T b \\ -{Q_2}^T b \end{pmatrix} \right\|$. *We said that the entries in the second block are fixed irrespective of* x*, but we could make the entries in the first block zero, by choosing* $x = R_1^{-1} Q^T b$*. This information let us infer something about the values of the vectors in Figure 1.3, in particular the minimum of the value that we can get is* $\left\| {Q_2}^T b \right\| = \|Ax - b\|$.

With some algebra we may also check that $\left\| {Q_1}^T b \right\| = \|Ax\|$.

Since this is a minimum problem, we know that the gradient of the function is small near the optimum value: $\min \|Ax - b\|^2 = \min x^T A^T A x - 2b^T A x + b^T b$.

$\nabla_{\widetilde{x}} f = 2(A^T A x - A^T b) \to 0$

Theorem 1.3. $\frac{\|\widetilde{x} - x\|}{x} \leq \kappa(A)^2 \frac{\|A^T A x - A^T b\|}{\|A^T b\|}$. *Although we might have wanted to have the condition number of the problem, instead of the condition number of* A *and this could lead to underestimating the error.*

Another idea could be using as error the first entry of the vector obtained via QR (namely ${R_1}^T x - {Q_1}^T b$), by imposing $R_1 x = {Q_1}^T b$

We may observe that this is a truly backward stable measure:

given $r = \left\| {R_1}^T \widetilde{x} - {Q_1}^T b \right\|$, there exists $\widetilde{b}$ with $\left\| \widetilde{b} - b \right\| = \|r\|$ such that $\widetilde{x}$ is the exact solution of $\min \left\| Ax - \widetilde{b} \right\|$.

We have proved the following

Fact 1.4. $\frac{\|\widetilde{x} - x\|}{\|x\|} \leq \kappa_{rel,LS} \frac{\|\widetilde{b} - b\|}{\|b\|} = \kappa_{rel,LS} \frac{\|r\|}{\|b\|}$.

Theorem 1.5. *Let $A = Q_1 R_1$ be a thin QR factorization. Let $\mathbf{r}_1 = Q_1^T(A\widetilde{\mathbf{x}} - \mathbf{b})$. Then, $\widetilde{\mathbf{x}}$ is the exact solution of the LS problem:*

$$\min \|A\mathbf{x} - (\mathbf{b} + Q_1\mathbf{r}_1)\|$$

so the backward error of $\widetilde{\mathbf{x}}$ is $\|Q_1\mathbf{r}_1\| = \|\mathbf{r}_1\|$.

Proof.
Idea: replay the solution of a LS problem with QR factorization, and use ${Q_1}^T T Q_1 = I$. You will get in the first block $R_1 x = {Q_1}^T b + \mathbf{r}_1$, i.e., ${Q_1}^T(Ax - b) = \mathbf{r}_1$, which is verified by $\widetilde{x}$. □