FGV EMAp

João Pedro Jerônimo

Algebra Linear Numérica

Revisão para A1

Rio de Janeiro

2025

Sumário

As aulas abaixo referem-se ao livro de Trefethen sobre álgebra linear numérica

1. Lecture 3 - Normas

Aviso: O capítulo sobre normas tem conceitos bastante abstratos, alguns deles não são muito **intuitivos**, então tente abstrair e aceitar que eles existem por enquanto, mais tarde mostraremos que são muito úteis.

1.1. Normas de vetores

Definição 1.1.1 (Norma): Uma **norma** é uma função $\|\cdot\| : \mathbb{C}^m \to \mathbb{R}$ que satisfaz 3 propriedades:

1. $\|x\| \geq 0$, e $\|x\| = 0 \Leftrightarrow x = 0$
2. $\|x + y\| \leq \|x\| + \|y\|$
3. $\|\alpha x\| = |\alpha|\|x\|$

Normalmente vemos a norma-2, ou a **Norma Euclidiana**, que representa o **tamanho** de um vetor. Com base nisso, podemos definir uma **norma-p**.

Definição 1.1.2 (Norma-p): A **norma-p** de $x \in \mathbb{C}^n$ (ou $\|x\|_p$) é definida como:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$$

Então, podemos ter vários tipos de normas, de 1 até ∞, e também definimos isso!

Definição 1.1.3 (Norma infinita): A **norma infinita** de $x \in \mathbb{C}^n$ (ou $\|x\|_\infty$) é definida como:

$$\|x\|_\infty = \max|x_i|$$

Você provavelmente está se perguntando "por que eu precisaria de algo assim"? Mas acredite, será útil no futuro! Existe um tipo de norma muito útil (de acordo com o livro), chamada **norma ponderada**.

Definição 1.1.4 (Norma ponderada): A **norma ponderada** de $x \in \mathbb{C}^n$ é:

$$\|x\|_W = \|Wx\| = \left(\sum_{i=1}^{n} |w_{ii}x_i|^p\right)^{\frac{1}{p}}$$

Onde W é uma **matriz diagonal** e p é um número arbitrário

1.2. Normas de matrizes

O QUÊ?? MATRIZES TÊM NORMAS???? Sim, meu jovem Padawan! O livro diz que podemos ver uma matriz como um vetor em um espaço $m \times n$, e podemos usar qualquer norma mn para medi-la, mas algumas normas são mais úteis do que as já discutidas.

Definição 1.2.1 (Norma induzida): Dada $A \in \mathbb{C}^{m \times n}$, a norma induzida $\|A\|_{m \to n}$ é o menor inteiro para o qual a desigualdade é válida:

$$\|Ax\|_m \leq C\|x\|_n$$

Em outras palavras:

$$\|A\|_{m \to n} = \sup_{x \neq 0} \frac{\|Ax\|_m}{\|x\|_n}$$

Essa definição pode parecer inútil e estúpida por enquanto, mas será muito útil quando falarmos sobre erros e condicionamento.

Uma norma útil que podemos mencionar é a norma ∞ de uma matriz.

Definição 1.2.2 (Norma infinita de uma matriz): Dada $A \in \mathbb{C}^{m\times n}$, se a_j é a j^{th} linha de A, $\|A\|_\infty$ é definida por:

$$\|A\|_\infty = \max_{1\leq i\leq m}\|a_i\|_1$$

1.3. Desigualdades de Cauchy-Schwarz e Hölder

Quando usamos normas, geralmente é difícil calcular normas-p com valores altos de p, então as gerenciamos usando desigualdades! Uma desigualdade muito útil é a de Hölder:

Definição 1.3.1 (Desigualdade de Hölder): Dados $1 \leq p, q \leq \infty$, e $\frac{1}{p} + \frac{1}{q} = 1$, então, para quaisquer vetores x, y:

$$|x^* y| \leq \|x\|_p \|y\|_q$$

E a desigualdade de Cauchy-Schwarz é um caso especial onde $p = q = 2$.

1.4. Limitando $\|AB\|$

Podemos limitar $\|AB\|$ como fazemos com normas de vetores.

Teorema 1.4.1: Dadas $A \in \mathbb{C}^{l\times m}$,$B \in \mathbb{C}^{m\times n}$ e $x \in \mathbb{C}^n$, então a norma induzida de AB deve satisfazer:

$$\|AB\|_{l\to n} \leq \|A\|_{l\to m}\|B\|_{m\to n}$$

Prova: $\|ABx\|_l \leq \|A\|_{l\to m}\ \|Bx\|_m \leq \|A\|_{l\to m}\ \|B\|_{m\to n}\ \|x\|_n$ □

1.5. Generalização das normas de matrizes

Vimos que uma norma segue 3 propriedades, definimos uma norma geral de matriz da mesma forma!!

Definição 1.5.1: Dadas as matrizes A e B, uma norma $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}^+$ é uma função que segue estas 3 propriedades:

1. $\|A\| \geq 0$, e $\|A\| = 0 \Leftrightarrow A = 0$
2. $\|A + A\| \leq \|A\| + \|B\|$
3. $\|\alpha A\| = |\alpha|\|A\|$

A mais importante é a **Norma de Frobenius**, definida como:

Definição 1.5.2: Dada uma matriz $A \in \mathbb{C}^{m\times n}$, sua **Norma de Frobenius** é definida como:

$$\|A\|_F = \left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2\right)^{\frac{1}{2}} = \sqrt{tr(A^*A)} = \sqrt{tr(AA^*)}$$

Teorema 1.5.1: $\|AB\|_F \leq \|A\|_F\ \|B\|_F$

Prova: $\|AB\|_F = \left(\sum_{i=1}^m \sum_{j=1}^n |c_{ij}|^2\right)^{\frac{1}{2}} \leq \left(\sum_{i=1}^m \sum_{j=1}^n \left(\|a_i\|_2\|b_j\|_2\right)^2\right)^{\frac{1}{2}} = \left(\sum_{i=1}^m \|a_i\|_2^2 \sum_{j=1}^n \|b_j\|_2^2\right)^{\frac{1}{2}} = \|A\|_F\|B\|_F$ □

2. Lectures 4 e 5 - A SVD

Aviso rápido: Quando começarmos a falar sobre a fatoração em si, vamos falar sobre matrizes em $\mathbb{C}^{m\times n}$ com $m \geq n$, porque é o mais comum quando falamos de problemas reais, raramente são situações com mais variáveis do que equações.

Aaaaaah, a SVD, por que ela existe? O que significa? Lembre-se que, em Álgebra Linear, quando temos uma base de um Espaço Vetorial e uma Transformação Linear, sabemos como a Transformação Linear afeta **cada** vetor naquele Espaço Vetorial? Não? Deixe-me refrescar sua memória:

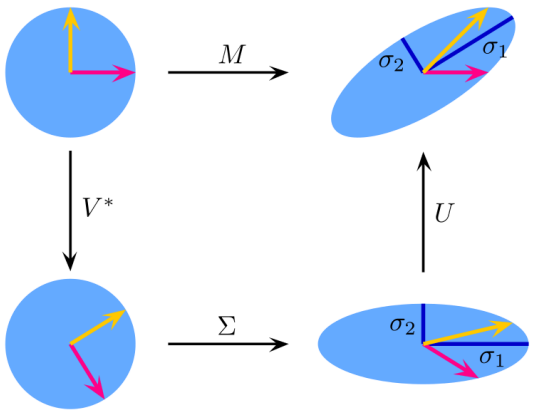
> **Teorema 2.1**: Dada $\{a_j\}$ $(1 \leq j \leq n)$ sendo a base de um Espaço Vetorial e T uma Transformação Linear nesse espaço, se sabemos como T afeta os vetores da base, sabemos como T afeta **cada** vetor nesse espaço.
>
> *Prova*: Sendo v um vetor no Espaço Vetorial descrito, sabemos que v pode ser expresso como
>
> $$v = \alpha_1 a_1 + ... + \alpha_n a_n$$
>
> Aplicando T em v
>
> $$T(v) = T(\alpha_1 a_1 + ... + \alpha_n a_n) \Rightarrow T(v) = \alpha_1 T(a_1) + ... + \alpha_n T(a_n)$$
>
> Isso implica que, se conhecemos uma base do Espaço Vetorial e como T a afeta, sabemos como T afeta cada vetor no espaço. □

CERTO! Memória refrescada, por que eu disse isso? Lembre-se que matrizes são transformações lineares? Então, se temos uma base $\{s_j\}$ de um Espaço Vetorial S, podemos saber o que acontece com cada combinação linear de $\{s_j\}$ se aplicarmos A nela, certo? Certo!

Podemos resumir as operações que fazemos em vetores em duas: **esticar** e **rotacionar**, então, basicamente, quando aplicamos uma transformação linear em um vetor, estamos rotacionando-o e depois esticando-o.

Ok, mas por que estou dizendo isso? Onde diabos está a S.V.D? Bem, eu basicamente já descrevi a S.V.D para você! Quando aplicamos A como uma transformação linear, se fizermos as operações descritas anteriormente, você concorda que podemos decompor A como um produto de matrizes ortogonais e matrizes diagonais? O quê? Por quê? Quando? Espere, jovem Padawan! Lembre-se que eu disse que uma transformação linear pode ser resumida em esticar e rotacionar vetores? Você lembra que tipo de matrizes fazem EXATAMENTE o que eu disse? Sim, matrizes ortogonais fazem rotações e matrizes diagonais fazem alongamento.

Agora podemos introduzir aquela visualização clássica de como a S.V.D funciona, imagine uma base ortonormal em $\mathbb{R}^2$, veja o que acontece se aplicarmos A nela:



$$M = U \cdot \Sigma \cdot V^*$$

(Troque o M na imagem por A)

Com base nisso, podemos definir que, dada $A \in \mathbb{C}^{m\times n}$:

$$Av_j = \sigma_j u_j$$

Onde v_j e u_j são de duas bases ortonormais diferentes e $\sigma_j \in \mathbb{C}$.

2.1. Forma reduzida

Podemos reescrever esta equação como um produto matricial!

$AV = \hat{U}\hat{\Sigma}$

Onde

$$V = \begin{pmatrix} | & & | \\ v_1 & \dots & v_n \\ | & & | \end{pmatrix}, \Sigma = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{pmatrix}, U = \begin{pmatrix} | & & | \\ u_1 & \dots & u_n \\ | & & | \end{pmatrix}$$

Isso é conhecido como a fatoração SVD **reduzida**. Podemos ver que V é uma matriz ortogonal quadrada (para Av_j ser uma multiplicação válida, $v_j \in \mathbb{C}^n$), então podemos reescrever A como:

$$A = \hat{U}\hat{\Sigma}V^*$$

2.2. SVD completa

Ok, se $v_j \in \mathbb{C}^n$ e $Av_j = \sigma_j u_j$, então $u_j \in \mathbb{C}^m$! Isso significa que, além dos vetores u que adicionamos em $\hat{U}$, temos mais $m - n$ vetores ortonormais para as colunas de $\hat{U}$, ao encontrar esses vetores, podemos construir outra matriz U cujas colunas são uma base ortonormal de $\mathbb{C}^m$, o que significa que a nova matriz U é ortogonal!

$$V = \begin{pmatrix} | & & | \\ v_1 & \dots & v_n \\ | & & | \end{pmatrix}, U = \begin{pmatrix} | & & | \\ u_1 & \dots & u_m \\ | & & | \end{pmatrix}$$

Legal! Mas e a matriz $\hat{\Sigma}$? Como ela muda? Bem, queremos manter V e U como queríamos, certo? Bem, o que fizemos foi adicionar colunas a $\hat{U}$, então, na multiplicação, só precisamos que essas colunas desapareçam, como fazemos isso? Multiplicando por 0! Então, antes, $\hat{\Sigma}$ era uma matriz quadrada com os valores singulares na diagonal, especificamente, n valores singulares. Se adicionamos $m - n$ vetores em U, podemos adicionar $m - n$ zeros em $\hat{\Sigma}$, então nossa nova multiplicação matricial é

$$A = U\Sigma V^*$$

$$\begin{pmatrix} | & & | \\ a_1 & \dots & a_n \\ | & & | \end{pmatrix} = \begin{pmatrix} | & & | \\ u_1 & \dots & u_m \\ | & & | \end{pmatrix} \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ - & 0 & - \\ & \vdots & \end{pmatrix} \begin{pmatrix} -v_1- \\ \dots \\ -v_n- \end{pmatrix}$$

2.3. Definição formal

Definição 2.3.1: Dada $A \in \mathbb{C}^{m\times n}$ com $m \geq n$, a Decomposição por Valores Singulares de A é:

$$A = U\Sigma V^*$$

onde $U \in \mathbb{C}^{m\times m}$ é unitária, $V \in \mathbb{C}^{n\times n}$ é unitária e $\Sigma \in \mathbb{C}^{m\times n}$ é diagonal. Para **conveniência**, denotamos:

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \dots \geq \sigma_n$$

Onde σ_j é a j-ésima entrada de Σ

Ok, vimos um método intuitivo para ver que toda matriz tem essa decomposição, mas como provamos isso matematicamente?

Teorema 2.3.1: Toda matriz $A \in \mathbb{C}^{m\times n}$ tem uma decomposição S.V.D

Prova: Seja $\{v_j\}$ uma base ortonormal de $\mathbb{C}^n$, $\{u_j\}$ uma base ortonormal de $\mathbb{C}^m$, $Av_j = \sigma_j u_j$, U_1 e V_1 matrizes unitárias de colunas $\{u_j\}$ e $\{v_j\}$ respectivamente e que, para toda matriz com menos de m linhas e n colunas, a fatoração é válida:

$$A = U_1 S V_1^* \Leftrightarrow U_1^* A V_1 = S$$

Então temos $S = \begin{pmatrix} \sigma_1 & w^* \\ 0 & B \end{pmatrix}$ onde σ_1 é 1×1, w^* é $1 \times (n-1)$ e B é $(m-1) \times (n-1)$. Beleza, mas o que é w? Bem, podemos chegar nesse resultado fazendo algumas manipulações com $\|\begin{pmatrix} \sigma_1 & w^* \\ 0 & B \end{pmatrix}\begin{pmatrix} \sigma_1 \\ w \end{pmatrix}\|_2$:

$$\|\begin{pmatrix} \sigma_1 & w^* \\ 0 & B \end{pmatrix}\begin{pmatrix} \sigma_1 \\ w \end{pmatrix}\|_2^2 \geq \sigma_1^2 + w^* w$$

O quê? Por que isso é válido? Porque:

$$\|Mx\|_2 \leq \|M\|_2 \, \|x\|_2 \Rightarrow \|M\|_2 \geq \frac{\|Mx\|_2}{\|x\|_2}$$

Se definirmos $x = \begin{pmatrix} \sigma_1 \\ w \end{pmatrix}$ e $M = S$, então temos:

$$Mx = \begin{pmatrix} \sigma_1^2 + \|w\|^2 \\ Bw \end{pmatrix} \Rightarrow \|M\|_2 \geq \frac{|\sigma_1^2 + \|w\|^2|^2 + \|Bw\|^2}{\sigma_1^2 + \|w\|^2}$$

Mas observe que o numerador é sempre maior que o denominador, então isso significa

$$\frac{|\sigma_1^2 + \|w\|^2|^2 + \|Bw\|^2}{\sigma_1^2 + \|w\|^2} \geq \sigma_1^2 + \|w\|^2 = (\sigma_1^2 + w^* w)^{\frac{1}{2}} \|\begin{pmatrix} \sigma_1 \\ w \end{pmatrix}\|$$

Agora podemos voltar para ver o que é w! Bem, agora é fácil! Sabemos que $\|S\|_2 = \|U_1^* A V_1\|_2 = \|A\|_2 = \sigma_1$ porque U_1 e V_1 são ortogonais. Isso significa $\|S\|_2 \geq (\sigma_1^2 + \|w\|^2)^{\frac{1}{2}} \Rightarrow \sigma_1 \geq (\sigma_1^2 + \|w\|^2)^{\frac{1}{2}} \Leftrightarrow \sigma_1^2 \geq \sigma_1^2 + \|w\|^2 \Rightarrow w = 0$.

Pela hipótese indutiva descrita no início da prova, sabemos que $B = U_2 \Sigma_2 V_2^*$, então podemos facilmente escrever A como

$$A = U_1 \begin{pmatrix} 1 & 0 \\ 0 & U_2 \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & V_2^* \end{pmatrix}^* V_1^*$$

Isso é uma S.V.D de A, usando o caso base de $m = 1$ e $n = 1$, terminamos a prova da existência □

2.4. Mudança de base

Dado $b \in \mathbb{C}^m$, $x \in \mathbb{C}^n$ e $A \in \mathbb{C}^{m \times n}$, $A = U\Sigma V^*$ podemos obter as coordenadas de b na base das colunas de U e x nas colunas de V. Só para lembrar:

Definição 2.4.1: Dado $w \in V$ onde V é um Espaço Vetorial, $\exists! x_1, ..., x_n \in \mathbb{C}$ tal que $w = v_1 x_1 + ... + v_n x_n$ onde $\{v_j\}$ é uma base de V. O vetor $\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$, também denotado como $[w]_v$, é o **vetor de coordenadas** de w na base v

Voltando, podemos expressar $[b]_u = U^* b$ e $[x]_v = V^* x$, mas por quê?

Teorema 2.4.1: Dada uma base ortonormal $\{v_k\}$ de V e $w \in V$, então

$$([w]_v)_j = v_j^* w$$

Prova:

$$w = \alpha_1 v_1 + ... + \alpha_n v_n$$

$$v_j^* w = \alpha_1 v_j^* v_1 + ... + \alpha_n v_j^* v_n$$

Sabendo que $\{v_j\}$ é uma base ortonormal, o produto $\alpha_i v_j^* v_i$ é igual a 0 se $j \neq i$ e igual a α_i se $j = i$, ou seja:

$$v_j^* w = \alpha_j$$

□

Ok, agora que lembramos todas essas propriedades, podemos expressar a relação $b = Ax$ em termos de $[b]_u$ e $[x]_v$, vamos ver:

$$b = Ax \Leftrightarrow U^*b = U^*Ax = U^*U\Sigma V^*x \Leftrightarrow U^*b = \Sigma V^*x$$

$$\Leftrightarrow [b]_u = \Sigma[x]_v$$

Então podemos reduzir A à matriz Σ e b e x às suas coordenadas nas bases u e v

2.5. S.V.D vs Decomposição por Autovalores

Podemos fazer algo semelhante com a decomposição por autovalores. Dada $A \in \mathbb{C}^{m\times m}$ com autovetores linearmente independentes, ou seja, podemos expressar $A = S\Lambda S^{-1}$ com as colunas de S sendo os autovetores de A e Λ sendo uma matriz diagonal com os autovalores de A como entradas.

Definindo $b, x \in \mathbb{C}^m$ satisfazendo $b = Ax$, podemos escrever:

$$[b]_{s^{-1}} = S^{-1}b \text{ e } [x]_{s^{-1}} = S^{-1}x$$

Onde estou denotando s^{-1} como a base expressa pelas colunas de S^{-1}, então a nova expressão expandida é:

$$b = Ax \Leftrightarrow S^{-1}b = S^{-1}Ax = S^{-1}S\Lambda S^{-1}x \Leftrightarrow S^{-1}b = \Lambda S^{-1}x$$

$$[b]_{s^{-1}} = \Lambda[x]_{s^{-1}}$$

2.6. Propriedades de matrizes com SVD

Para as próximas propriedades, seja $A \in \mathbb{C}^{m\times n}$ e $r \leq \min(m, n)$ o número de valores singulares não nulos

Teorema 2.6.1: $\text{rank}(A) = r$

Prova: O posto de uma matriz diagonal é o número de entradas não nulas, bem, se $A = U\Sigma V^*$, sabemos que U e V têm posto completo, então o posto de A deve ser o mesmo que o de Σ, ou seja, r □

Teorema 2.6.2: $C(A) = \text{span}\{u_1, ..., u_r\}, C(A^*) = \text{span}\{v_1, ...v_r\}, N(A) = \text{span}\{v_{r+1}, ..., v_n\}, N(A^*) = \text{span}\{u_{r+1}, ..., u_m\}$

Prova: Vamos lembrar como cada matriz é estruturada:

$$A = \begin{pmatrix} | & & | \\ u_1 & \cdots & u_m \\ | & & | \end{pmatrix} \begin{pmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & 0 & \\ & & & & \ddots \end{pmatrix} \begin{pmatrix} -v_1^*- \\ \vdots \\ -v_n^*- \end{pmatrix}$$

É fácil ver por que $C(A) = \text{span}\{u_1, ..., u_r\}$, porque as entradas de Σ só permitem abranger as primeiras r colunas de U.

Sobre $N(A) = \text{span}\{v_{r+1}, ..., v_n\}$, observe como, se fizermos Av_j $r + 1 \leq j \leq n$, as primeiras r linhas se tornarão 0 (todos os v_k são ortonormais entre si) e, como as entradas diagonais após a r-ésima são 0, então temos U vezes a matriz 0

Para ver as propriedades de A^*, vamos transpor A

$$A^* = \begin{pmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{pmatrix} \begin{pmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & 0 & \\ & & & & \ddots \end{pmatrix} \begin{pmatrix} -u_1^*- \\ \vdots \\ -u_m^*- \end{pmatrix}$$

Então, novamente, é fácil ver que $C(A^*) = \text{span}\{v_1, ...v_r\}$ e, usando o mesmo argumento mostrado antes, $N(A^*) = \text{span}\{u_{r+1}, ..., u_m\}$ □

Teorema 2.6.3: $\|A\|_2 = \sigma_1$ e $\|A\|_F = \sqrt{\sigma_1^2 + ... + \sigma_r^2}$

Prova:
1. $\|A\|_2 = \|U\Sigma V\|_2 = \|\Sigma\|_2$, como denotamos antes, de todas as entradas, σ_1 é a maior, isso significa $\|A\|_2 = \|\Sigma\|_2 = \sigma_1$
2. Sabemos que $\|A\|_F = \sqrt{\text{tr}(A^*A)} = \sqrt{\text{tr}(V\Sigma^*U^*U\Sigma V^*)} = \sqrt{\text{tr}(V\Sigma^*\Sigma V^*)}$. Também sabemos que $\text{tr}(A) = \lambda_1 + ... + \lambda_n$ com λ_j sendo os autovalores de A, e podemos ver claramente que os autovalores de $V\Sigma^*\Sigma V^*$ são σ_j^2, portanto $\|A\|_F = \sqrt{\sigma_1^2 + ... + \sigma_r^2}$

□

Teorema 2.6.4: $\sigma_j = \sqrt{\lambda_j}$ com σ_j sendo os valores singulares de A e λ_j os autovalores de A^*A

Prova: $A^*A = V\Sigma^*U^*U\Sigma V^* = V\Sigma^*\Sigma V^*$ □

Teorema 2.6.5: se $A = A^*$, então os valores singulares de A são os valores absolutos dos autovalores de A

Prova: Pelo Teorema Espectral, sabemos que A tem uma decomposição por autovalores

$$A = Q\Lambda Q^*$$

Podemos reescrevê-la como

$$A = Q|\Lambda|\text{sign}(\Lambda)Q^*$$

Onde as entradas de $|\Lambda|$ são $|\lambda_j|$ e as entradas de $\text{sign}(\Lambda)$ são $\text{sign}(\lambda_j)$. Podemos mostrar que, se Q é unitária, $\text{sign}(\Lambda)Q$ é unitária, o que significa que $Q|\Lambda|\text{sign}(\Lambda)Q^*$ é uma SVD de A □

Teorema 2.6.6: Para $A \in \mathbb{C}^{m\times m}$, $|\det(A)| = \prod_{i=1}^m \sigma_i$

Prova: $|\det(A)| = |\det(U\Sigma V^*)| = |\det(U)\det(\Sigma)\det(V)| = |\det(\Sigma)|$ □

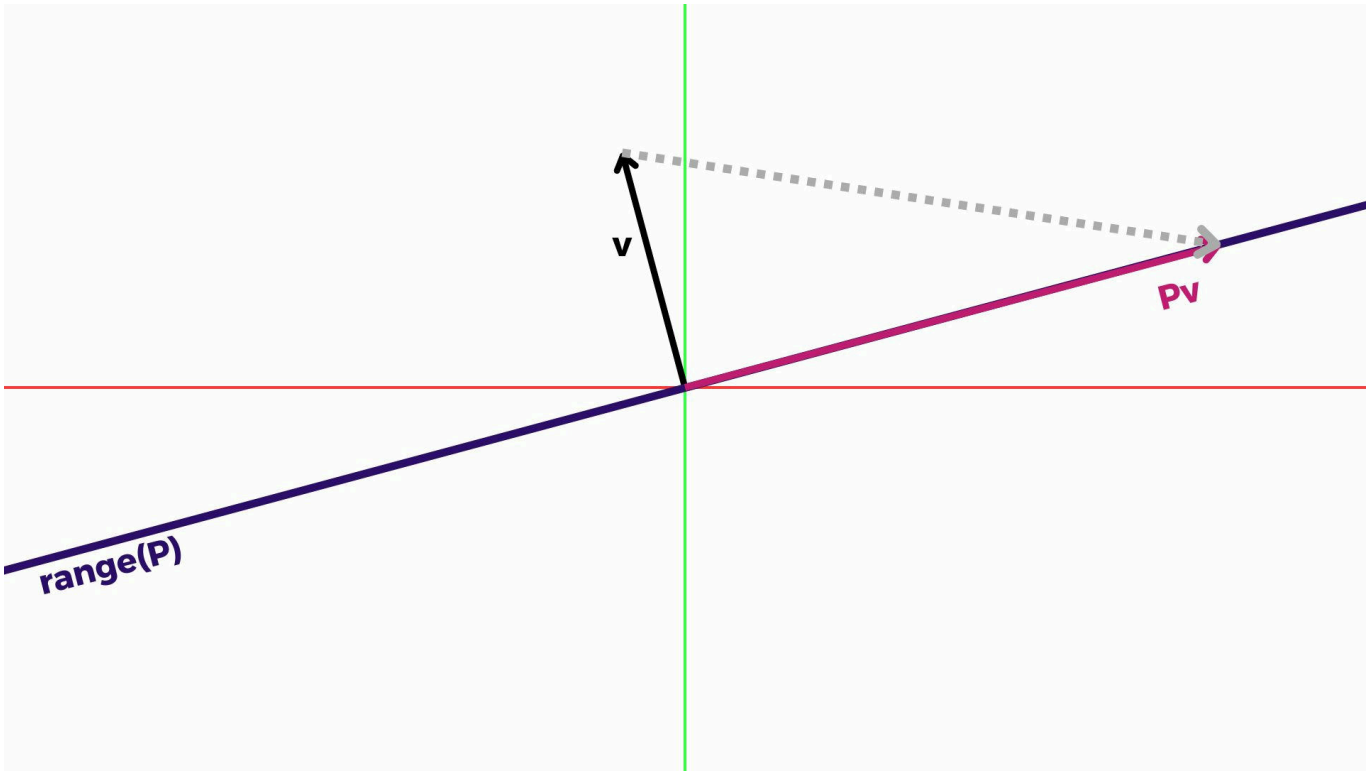
3. Lecture 6 - Projetores

$P \in \mathbb{C}^{m\times n}$ é dito um **Projetor** se

$$P^2 = P$$

também chamado *idempotente*. Você pode se confundir pensando apenas em projeções ortogonais, aquelas em que pegamos o vetor e o projetamos de forma a formar um ângulo de 90 graus no espaço projetado! Mas estamos falando de **todas** as projeções, incluindo as não ortogonais.

Imagine que colocamos uma luz naquele vetor, ele lançará uma sombra em algum lugar, mas você concorda comigo que podemos obter essa sombra de alguma forma, certo? Vamos ver um exemplo em 2D:



Como você pode ver, o vetor tracejado indica a direção em que a luz está projetando a sombra de v sobre P. Podemos expressar essa direção como $Pv - v$. É importante lembrar que, se você está deitado no chão, não terá uma sombra, certo? Ou melhor ainda, sombras não têm sombras! Traduzindo isso para o nosso contexto:

Teorema 3.1: Se $v \in C(P)$, então $Pv = v$

Prova: Todo $v \in C(P)$ pode ser expresso como $v = Px$ para algum x, isso significa $Pv = P^2x = Px = v$ □

Observe que, se aplicarmos a projeção na direção que tínhamos antes

$$P(Pv - v) = P^2v - Pv = Pv - Pv = 0$$

Isso significa que $Pv - v \in \text{null}(P)$. Também observe que podemos reescrever a direção como

$$Pv - v = (P - I)v = -(I - P)v$$

Veja que coisa ainda mais estranha!

$$(I - P)^2 = I - 2P + P^2 = I - P$$

Isso significa que $I - P$ também é um projetor! Um projetor que projeta na direção da projeção de P.

3.1. Projetores complementares

Se P é um projetor, $I - P$ é seu projetor complementar.

Teorema 3.1.1: $I - P$ projeta sobre $\text{null}(P)$ e P projeta sobre $\text{null}(I - P)$

Prova:

1. $C(I - P) \subseteq N(P)$ porque $v - Pv \in N(P)$ e $C(I - P) \supseteq N(P)$ porque, se $Pv = 0$, podemos reescrever como $(I - P)v = v$, isso significa $N(P) = C(I - P)$

2. Se reescrevermos a expressão como $P = I - (I - P)$, então, usando o mesmo argumento de antes, temos $C(P) = N(I - P)$

□

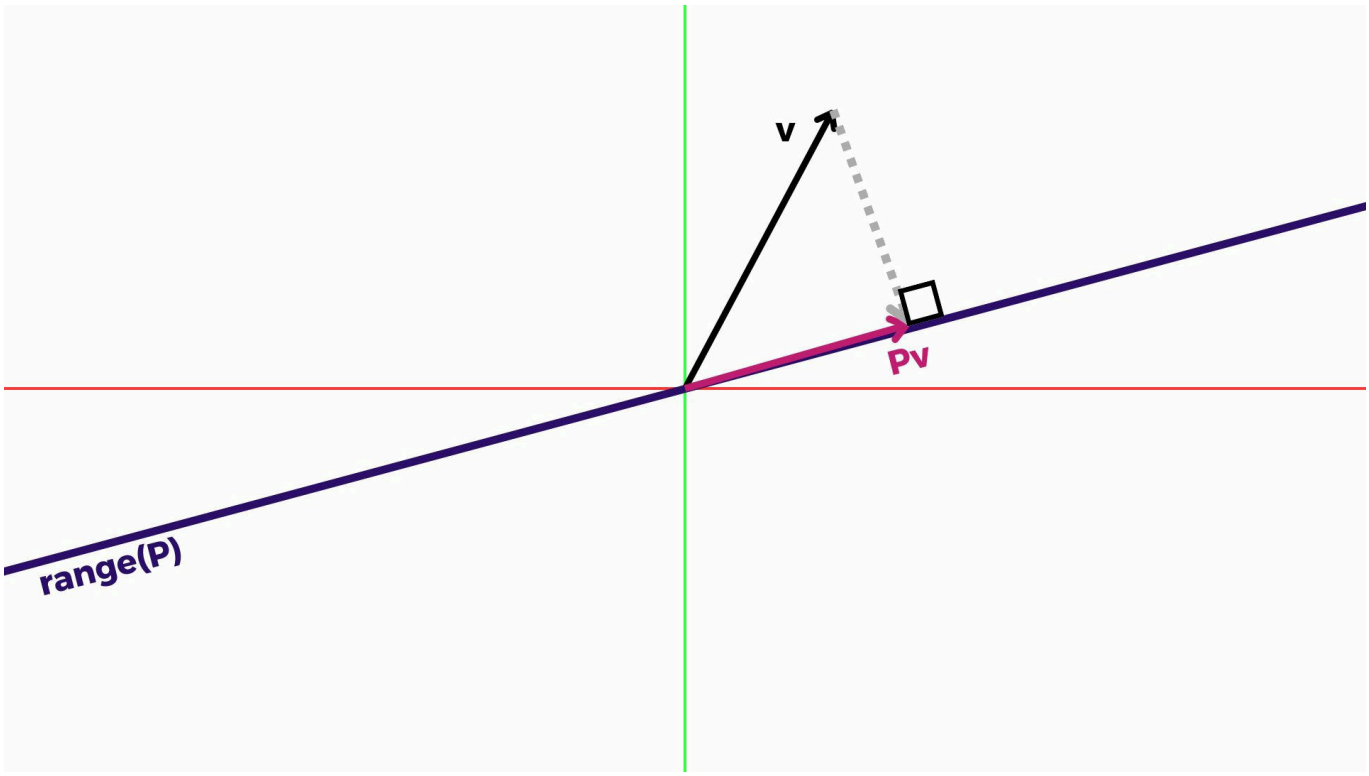
Teorema 3.1.2: $N(I - P) \cap N(P) = \{0\}$

Prova: $N(A) \cap C(A) = \{0\} \Rightarrow N(P) \cap C(P) = \{0\} \Leftrightarrow N(P) \cap N(I - P) = \{0\}$ □

Isso significa que, se temos um projetor P em $\mathbb{C}^{m \times m}$, esse projetor separa $\mathbb{C}^m$ em dois espaços S_1 e S_2, de forma que $S_1 \cap S_2 = \{0\}$ e $S_1 + S_2 = \mathbb{C}^m$.

3.2. Projetores ortogonais

Finalmente! Os projetores que ouvimos falar o tempo todo! Eles projetam um vetor em um espaço fazendo a direção formar um ângulo de 90 graus com a projeção.



Isso significa $(Pv)^*(v - Pv) = 0$

Teorema 3.2.1: P é um projetor ortogonal $\Leftrightarrow P = P^*$

Prova:

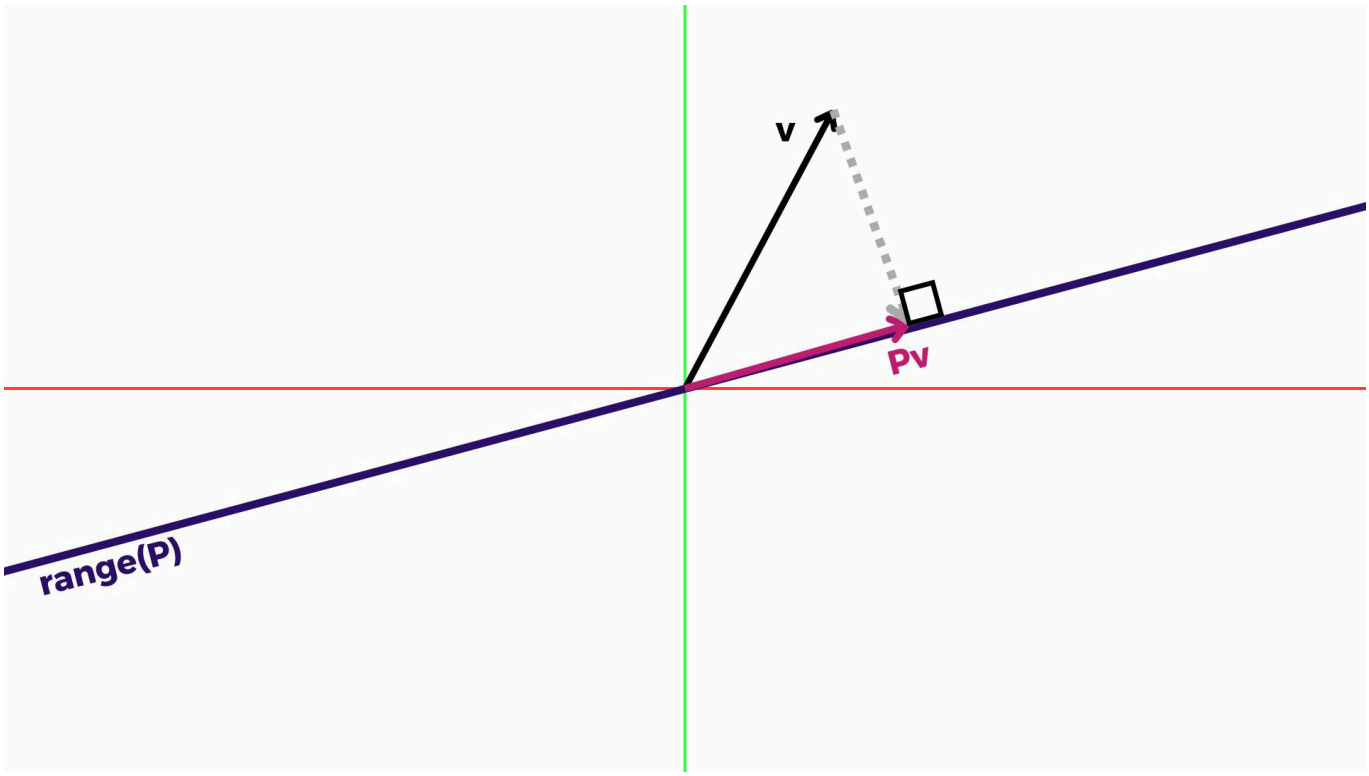
1. $\Leftarrow$) Dado $x, y \in \mathbb{C}^m$, então $x^*P^*(I - P)y = x^*(P^* - P^*P)y = x^*(P - P^2)y = x^*(P - P)y = 0$
2. $\Rightarrow$) Seja $\{q_1, ..., q_m\}$ uma base ortonormal de $\mathbb{C}^m$ onde $\{q_1, ..., q_n\}$ é base de S_1 e $\{q_{n+1}, ..., q_m\}$ é base de S_2. Para $j \leq n$ temos $Pq_j = q_j$ e para $j > n$ temos $Pq_j = 0$, seja Q a matriz com colunas $\{q_1, ..., q_m\}$, temos: $Q = \begin{pmatrix} | & & | \\ q_1 & \cdots & q_m \\ | & & | \end{pmatrix} \Leftrightarrow PQ = \begin{pmatrix} | & & | & | & \\ q_1 & \cdots & q_n & 0 & \cdots \\ | & & | & | & \end{pmatrix} \Leftrightarrow Q^*PQ = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 0 \\ & & & & & \ddots \end{pmatrix}$, o que significa que encontramos uma decomposição SVD para P:

$$P = Q\Sigma Q^* \Leftrightarrow P^* = Q\Sigma^* Q^* = Q\Sigma Q^* = P$$

□

3.3. Projeção ortogonal sobre um vetor

Vamos usar o mesmo exemplo usado anteriormente



Seja q o vetor que gera P, sabemos que $Pv = \alpha q$

$$(v - Pv)^* q = 0 = (v - \alpha q)^* q = 0$$

Agora procuramos o α que torna esta equação válida

$$v^* q - \alpha q^* q = 0 \Leftrightarrow v^* q = \alpha q^* q \Leftrightarrow \alpha = \frac{v^* q}{q^* q}$$

$$Pv = \alpha q \Leftrightarrow Pv = \frac{v^* q}{q^* q} q \Leftrightarrow Pv = \frac{q^* v}{q^* q} q \Leftrightarrow Pv = q\frac{q^* v}{q^* q} \Leftrightarrow Pv = \frac{qq^*}{q^* q} v \Rightarrow P = \frac{qq^*}{q^* q}$$

3.4. Projeção com base ortonormal

Vimos na prova de Teorema 3.2.1 que alguns valores singulares de P são 0, então poderíamos remover essas linhas de Σ e reduzi-lo a I, também removendo as colunas e linhas de Q, obtendo:

$$P = \hat{Q}\hat{Q}^*$$

Seja $\{q_1, ..., q_n\}$ qualquer conjunto de vetores ortonormais em $\mathbb{C}^m$ e sejam eles as colunas de $\hat{Q}$, sabemos que, para qualquer vetor $v \in \mathbb{C}^m$:

$$v = r + \sum_{i=1}^{n} q_i q_i^* v$$

O quê? Quando vimos isso? Calma, deixe-me recapitular para você:

Teorema 3.4.1: Seja $\{q_1, ..., q_n\}$ qualquer conjunto de vetores ortonormais em $\mathbb{C}^m$, então qualquer $v \in \mathbb{C}^m$ pode ser expresso como

$$v = r + \sum_{i=1}^{n} q_i q_i^* v$$

Com r sendo outro vetor em C^m ortogonal a $\{q_1, ..., q_n\}$ e, $\rightarrow n = m \Rightarrow r = 0$ e o conjunto de vetores escolhido é uma base para $\mathbb{C}^m$

Prova: Você sabe que, dada uma base de $\mathbb{C}^m$, qualquer vetor pode ser expresso como uma combinação linear desses vetores. Imagine a base canônica (com algumas rotações, essa lógica pode ser expandida para outras bases ortonormais), você pode imaginar que, se projetar o vetor que você tem sobre qualquer vetor da base canônica, obterá um vetor que, se somar com outro vetor r, obterá seu vetor original novamente! E podemos continuar esse processo até fazermos isso com n vetores da base canônica, obtendo o r original que, se somarmos todas as nossas projeções, obtemos o vetor original novamente, ou seja:

$$v = r + \sum_{i=1}^{n} q_i q_i^* v$$

□

Ok, sabendo que um vetor pode ser expresso assim, podemos ver que a parte da soma é a mesma que fazer:

$$\hat{Q}\hat{Q}^* v$$

Ou seja, $\sum_{i=1}^{n} q_i q_i^* v$ é um projetor sobre $C\left(\hat{Q}\right)$

Teorema 3.4.2: O complemento de um projetor ortogonal também é um projetor ortogonal

Prova:
1. $\left(I - \hat{Q}\hat{Q}^*\right)^2 = I - 2\hat{Q}\hat{Q}^* + \left(\hat{Q}\hat{Q}^*\right)^2 = I - 2\hat{Q}\hat{Q}^* + \hat{Q}\hat{Q}^* = I - \hat{Q}\hat{Q}^*$
2. $\left(I - \hat{Q}\hat{Q}^*\right)^* = I - \left(\hat{Q}\hat{Q}^*\right)^* = I - \hat{Q}\hat{Q}^*$

□

Um caso especial é o projetor ortogonal de posto um, que pega o vetor e obtém o componente em uma única direção q, que pode ser escrito:

$$P_q = qq^*$$

E seu complemento é a matriz de posto $(m - 1)$

$$P_{\perp q} = I - qq^*$$

Esse conceito também é válido para vetores não unitários:

$$P_a = \frac{aa^*}{a^*a}$$

$$P_{\perp a} = I - \frac{aa^*}{a^*a}$$

Só para esclarecer as coisas. Se projetarmos um vetor v sobre um vetor a, estamos restringindo v na direção da projeção, então, se projetarmos no complemento de a, é como se pudéssemos expressar v como uma combinação linear de a e alguns outros vetores, e então remover a parte de a nessa combinação linear, tendo apenas os outros vetores expressando um novo vetor.

3.5. Projeção em base arbitrária

Dada uma base arbitrária $\{a_j\}$, deixamos os vetores dessa base serem as colunas de A. Dado v com $Pv = y \in C(A)$, isso significa $y - v \perp C(A)$, ou seja, $a_j^*(y - v) = 0 \ \forall j$. Sabemos que $y \in C(A)$, então vamos escrevê-lo como $Ax = y$, então podemos reescrever $a_j^*(y - v) = 0 \ \forall j$ como:

$$A^*(Ax - v) = 0 \Leftrightarrow A^*Ax - A^*v = 0 \Leftrightarrow A^*Ax = A^*v \Leftrightarrow x = (A^*A)^{-1}A^*v$$

$$Ax = A(A^*A)^{-1}A^*v \Leftrightarrow y = A(A^*A)^{-1}A^*v$$

$$\Rightarrow P = A(A^*A)^{-1}A^*$$

4. Lecture 7 - Fatoração QR

A assustadora, a parte em que ninguém sabe de nada! Vamos nos acalmar e ver tudo com paciência.

Como funciona essa fatoração? Queremos expressar A como:

$$A = QR$$

Com Q sendo uma matriz ortogonal e R uma matriz triangular superior. Mas por que eu gostaria de fazer tal coisa? O principal motivo é resolver sistemas lineares! Vamos pegar o sistema $b = Ax$, reescrevemo-lo como

$$b = QRx \Leftrightarrow Q^*b = Rx \Leftrightarrow c = Rx$$

Temos um sistema equivalente, e este é um sistema **triangular**, ou seja, um sistema trivial para nós e para um computador resolverem!

4.1. A ideia da fatoração reduzida

Seja $\{a_j\}$ as colunas de $A \in \mathbb{C}^{m \times n}, \ m \geq n$. Em algumas aplicações, estamos interessados nos espaços das colunas de A, ou seja, os espaços sequenciais gerados pelas colunas de A:

$$\text{span}\{a_1\} \subseteq \text{span}\{a_1, a_2\} \subseteq \text{span}\{a_1, a_2, a_3\} \subseteq ...$$

Por enquanto, assumiremos que A tem posto completo n. Primeiro, queremos obter um conjunto de vetores ortonormais com a seguinte propriedade:

$$\text{span}\{a_1, ..., a_j\} = \text{span}\{q_1, ..., q_j\} \text{ com } j = 1, ..., n$$

Bem, acho que uma boa ideia para fazer isso é usar vetores tais que possamos expressar a_j como uma combinação linear de $\{q_1, ..., q_j\}$. Isso significa:

$$a_1 = r_{11}q_1$$

$$a_2 = r_{12}q_1 + r_{22}q_2$$

$$\vdots$$

$$a_n = r_{1n}q_1 + r_{2n}q_2 + ... + r_{nn}q_n$$

Podemos expressar essas equações como um produto matricial!

$$\begin{pmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{pmatrix} = \begin{pmatrix} | & | & & | \\ q_1 & q_2 & \dots & q_n \\ | & | & & | \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r_{22} & & \vdots \\ & & \ddots & \vdots \\ & & & r_{nn} \end{pmatrix}$$

Então temos $A = \hat{Q}\hat{R}$, onde $Q \in \mathbb{C}^{m \times n}$ e $R \in \mathbb{C}^{n \times n}$

4.2. Fatoração QR completa

Vai um pouco além. Sabemos que $\{q_1, ..., q_n\}$ é um conjunto de vetores ortonormais de $\mathbb{C}^m$, isso significa que temos mais $m - n$ vetores ortonormais aos que tínhamos antes, então podemos criar uma base para $\mathbb{C}^m$, adicionando esses vetores como colunas de $\hat{Q}$, temos uma matriz ortogonal Q. Mas o que fazemos para A

permanecer a mesma? Podemos simplesmente adicionar linhas de 0 abaixo de $\hat{R}$, criando $R \in \mathbb{C}^{m\times n}$ $(m \geq n)$, obtendo

$$A = QR$$

4.3. Ortonormalização de Gram-Schmidt

Nossa... a assustadora... Vamos com muita calma. Vimos anteriormente uma maneira de calcular todos os q_j, vamos relembrar:

$$a_1 = r_{11}q_1$$

$$a_2 = r_{12}q_1 + r_{22}q_2$$

$$\vdots$$

$$a_n = r_{1n}q_1 + r_{2n}q_2 + ... + r_{nn}q_n$$

Bem, isso sugere um algoritmo para calcular o próximo q_j, vamos pensar, temos todos os a_j, e cada q_j precisa dos vetores $\{q_1, ..., q_{j-1}\}$. Bem, podemos ter alguma liberdade aqui! Vamos ver o que acontece quando tentamos calcular q_j:

$$a_j = r_{1j}q_1 + ... + r_{jj}q_j$$

Vamos isolar q_j:

$$q_j = \frac{a_j - r_{1j}q_1 - r_{2j}q_2 - ... - r_{2(j-1)}q_{j-1}}{r_{jj}}$$

Bem, isso sugere que r_{jj} é a norma do vetor $a_j - \sum_{k=1}^{j-1} r_{ij}q_i$, mas o que é r_{ij}? Lembra da decomposição em fatores ortogonais? Sim, aquela, $v = r + \sum_{i=1}^{n} q_i q_i^* v$. Se trocarmos v por a_j, temos quase a mesma coisa que definimos anteriormente!

$$a_j - \sum_{k=1}^{j-1} r_{ij}q_i, \;\; v - \sum_{i=1}^{k} q_i q_i^* v$$

E você lembra que r é ortogonal a span$\{q_1, ..., q_k\}$? Isso é exatamente o que q_j é! Tudo isso que acabei de dizer sugere que posso definir r_{ij} como $q_i^* a_j$ $(i \neq j)$. E nosso algoritmo está pronto! Vamos recapitular tudo aqui:

$$q_1 = \frac{a_1}{\|a_1\|_2}$$

$$q_2 = \frac{a_2 - q_1 q_1^* a_2}{\|a_2 - q_1 q_1^* a_2\|_2}$$

$$q_3 = \frac{a_3 - q_1 q_1^* a_3 - q_2 q_2^* a_3}{\|a_3 - q_1 q_1^* a_3 - q_2 q_2^* a_3\|_2}$$

$$\vdots$$

$$q_n = \frac{a_n - \sum_{i=1}^{n-1} q_i q_i^* a_n}{\|a_n - \sum_{i=1}^{n-1} q_i q_i^* a_n\|_2}$$

Escrevendo na forma de um algoritmo:

1 **para** $j = 1$ **até** n
2 $\quad v_j = a_j$
3 $\quad$ **para** $i = 1$ **até** $j - 1$
4 $\quad\quad r_{ij} = q_i^* a_j$
5 $\quad\quad v_j = v_j - r_{ij} q_i$

6 | $r_{jj} = \|v_j\|_2$
7 | $q_j = \frac{v_j}{r_{jj}}$

4.4. Existência e unicidade

Teorema 4.4.1: Toda $A \in \mathbb{C}^{m\times n},\ (m \geq n)$ tem uma fatoração QR completa, portanto também uma fatoração QR reduzida

Prova: Se rank$(A) = n$, podemos construir a fatoração reduzida usando Gram-Schmidt como fizemos antes. O único problema aqui é se, em algum momento, $v_j = a_j - \sum_{k=1}^{j-1} q_k q_k^* a_j = 0$ e, portanto, não pode ser normalizado. Se isso acontecer, significa que A não tem posto completo, o que significa que posso escolher qualquer vetor ortogonal que quiser para continuar o processo. □

Teorema 4.4.2: Cada $A \in \mathbb{C}^{m\times n}\ (m \geq n)$ de posto completo tem uma fatoração QR reduzida única $A = \hat{Q}\hat{R}$ com $r_{jj} > 0$

Prova: Sabemos que, se A é de posto completo $\Rightarrow r_{jj} \neq 0$ e, portanto, em cada passo sucessivo j, as fórmulas mostradas anteriormente determinam r_{ij} e q_j completamente, o único problema é o sinal de r_{jj}, uma vez que dizemos $r_{jj} > 0$, esse problema é resolvido □

5. Lecture 8 - Ortonormalização de Gram-Schmidt

Podemos descrever o algoritmo de Gram-Schmidt usando projetores, mas por que quereríamos isso? Na verdade, isso é uma introdução para outro algoritmo que veremos mais tarde. Quando falamos de algoritmos, queremos que eles sejam estáveis, no sentido de que, se inserirmos uma entrada no computador, ele nos retornará uma resposta próxima da correta (computadores não resolvem problemas contínuos exatamente), e o processo de Gram-Schmidt não é estável (falaremos sobre isso nas próximas aulas).

Lembre-se que eu disse que, se você tem um vetor v e o decompõe como

$$v = r + \sum_{k=1}^{n} q_k q_k^* v$$

A parte $\sum_{k=1}^{n} q_k q_k^*$ é um projetor que projeta na matriz $\hat{Q}\hat{Q}^*$ ($\hat{Q}$ tem colunas $\{q_1, ..., q_n\}$)? Bem, acontece que podemos expressar os passos do algoritmo de Gram-Schmidt da mesma forma! Vamos relembrar. No j-ésimo passo, temos:

$$q_j = \frac{a_j - \sum_{i=1}^{j-1} q_i q_i^* a_j}{\|a_j - \sum_{i=1}^{j-1} q_i q_i^* a_j\|_2}$$

Isso significa que podemos reescrever isso como

$$q_j = \frac{\left(I - \hat{Q}_{j-1}\hat{Q}_{j-1}^*\right)a_j}{\|\left(I - \hat{Q}_{j-1}\hat{Q}_{j-1}^*\right)a_j\|_2}$$

Onde $\hat{Q}_{j-1} = \begin{pmatrix} | & & | \\ q_1 & & q_{j-1} \\ | & & | \end{pmatrix}$. Vamos definir, para simplificação, o projetor P_j como:

$$P_j = I - \hat{Q}_{j-1}\hat{Q}_{j-1}^*$$

5.1. Algoritmo de Gram-Schmidt Modificado

Usando as definições anteriores, vamos reescrever o Algoritmo de Gram-Schmidt.

Para cada valor de j, o algoritmo de Gram-Schmidt original calcula uma única projeção ortogonal de posto $m - (j-1)$. Estou apenas traduzindo para a linguagem usando projetores, ele faz isso:

$$v_j = P_j a_j = \left(I - \hat{Q}_{j-1}\hat{Q}_{j-1}^*\right)a_j$$

Se você voltar ao que eu disse antes, obterá a fórmula original, estou apenas trocando aquele monte de somas e vetores por um produto matricial. O algoritmo original faz esse cálculo usando um único projetor, mas o que veremos faz isso por uma sequência de $j-1$ projetores de posto $m-1$. Pela definição de P_j, podemos afirmar que:

Teorema 5.1.1:

$$P_j = P_{\perp q_{j-1}} ... P_{\perp q_2} P_{\perp q_1}$$

Prova: Lembre-se que $P_{\perp q_k} = I - q_k q_k^*$, e o que isso faz? Ele projeta um vetor v no subespaço ortogonal a $\{q_1, ..., q_{k-1}\}$, ou seja, removendo os componentes $\{q_1, ..., q_{k-1}\}$ de v. O projetor P_j faz exatamente a mesma coisa, certo? Então, você pode pensar que, se eu projeto no complemento de q_1, depois no complemento de q_2 e assim por diante, no j-ésimo passo, terei um vetor que é ortogonal aos anteriores, o que significa que removo todos os componentes anteriores, deixando o vetor projetado como uma combinação linear de $\{q_k, ...\}$ □

Ok! Se definirmos $P_1 = I$, podemos reescrever $v_j = P_j a_j$ como:

$$v_j = P_{\perp q_{j-1}} ... P_{\perp q_2} P_{\perp q_1} a_j$$

O novo algoritmo modificado é baseado nesta nova equação. Podemos obter o mesmo resultado declarado na versão anterior do algoritmo como:

$$v_j^{(1)} = a_j$$

$$v_j^{(2)} = P_{\perp q_1} v_j^{(1)}$$

$$v_j^{(3)} = P_{\perp q_2} v_j^{(2)}$$

$$v_j = v_j^{(j)} = P_{\perp q_{j-1}} v_j^{(j-1)}$$

Podemos reescrevê-lo na forma de pseudocódigo:

para $i = 1$ **até** n
  $v_i = a_i$
para $i = 1$ **até** n
  $r_{ii} = \|v_i\|$
  $q_i = \frac{v_i}{r_{ii}}$
  para $j = i + 1$ **até** n
    $r_{ij} = q_i^* v_j$
    $v_j = v_j - r_{ij} q_i$

5.2. Gram-Schmidt como Ortonormalização Triangular

Podemos interpretar cada passo do algoritmo de Gram-Schmidt como uma multiplicação à direita por uma matriz triangular superior quadrada. Espere, o quê? Por quê? Pegue a matriz R:

$$\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & & \vdots \\ & & \ddots & \vdots \\ & & & r_{nn} \end{pmatrix}$$

Você pode separá-la como:

$$\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & 1 & & \vdots \\ & & \ddots & \vdots \\ & & & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \cdots & 0 \\ & r_{22} & & \vdots \\ & & \ddots & \vdots \\ & & & 1 \end{pmatrix} \cdots$$

Então, podemos ver facilmente que, para a j-ésima matriz, a inversa dela é:

$$\begin{pmatrix} \ddots & & & \\ & 1 & & \\ & & r_{jj} & r_{j(j+1)} & \cdots \\ & & & \ddots \end{pmatrix}^{-1} = \begin{pmatrix} \ddots & & & \\ & 1 & & \\ & & \frac{1}{r_{jj}} & -\frac{r_{j(j+1)}}{r_{jj}} & \cdots \\ & & & \ddots \end{pmatrix}$$

Isso significa que podemos entender o algoritmo de Gram-Schmidt como uma ortonormalização por matrizes triangulares

$$AR_1R_2...R_n = \hat{Q}$$

$$R_1R_2...R_n = R^{-1}$$

6. Lecture 10 - Triangularização de Householder

NÃOOOO, HOUSEHOLDER NÃOOOOO! Espere, espere, espere, vamos entrar nisso passo a passo! Vimos no último capítulo que o algoritmo de Gram-Schmidt pode ser escrito como uma série de multiplicações por matrizes triangulares superiores, certo? Bem, o algoritmo de triangularização de Householder é muito semelhante, mas, como o nome sugere, em vez de obtermos uma matriz ortogonal no final, terminamos com uma matriz triangular superior

$$Q_1Q_2...Q_nA = R$$

É fácil ver que $Q_n^*...Q_2^*Q_1^*$ é uma matriz unitária, o que significa que $A = Q_n^*...Q_2^*Q_1^*R$ é uma fatoração QR completa de A

6.1. Triangularização por Introdução de Zeros

No coração do algoritmo de Householder, temos a ideia de aplicar uma matriz ortogonal que introduz zeros abaixo da diagonal principal! Assim (neste exemplo, x significa uma entrada não nula, $\boldsymbol{x}$ significa uma entrada que mudou desde a última aplicação ortogonal e nada significa 0)

$$\begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}_{A} \rightarrow Q_1A \rightarrow \begin{pmatrix} \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & \boldsymbol{x} & \boldsymbol{x} \\ & \boldsymbol{x} & \boldsymbol{x} \\ & \boldsymbol{x} & \boldsymbol{x} \\ & \boldsymbol{x} & \boldsymbol{x} \end{pmatrix}_{Q_1A} \rightarrow Q_2Q_1A \rightarrow \begin{pmatrix} x & x & x \\ & \boldsymbol{x} & \boldsymbol{x} \\ & & \boldsymbol{x} \\ & & \boldsymbol{x} \\ & & \boldsymbol{x} \end{pmatrix}_{Q_2Q_1A} \rightarrow Q_3Q_2Q_1A \rightarrow \begin{pmatrix} x & x & x \\ & x & x \\ & & \boldsymbol{x} \\ & & \\ & & \end{pmatrix}_{Q_3Q_2Q_1A}$$

6.2. Refletores de Householder

Ok, entendemos como o algoritmo vai funcionar, mas que tipo de matrizes pode fazer tal coisa? É aqui que entram em cena os **refletores de Householder**! Cada Q_k terá esta estrutura:

$$Q_k = \begin{pmatrix} I & 0 \\ 0 & F \end{pmatrix}$$

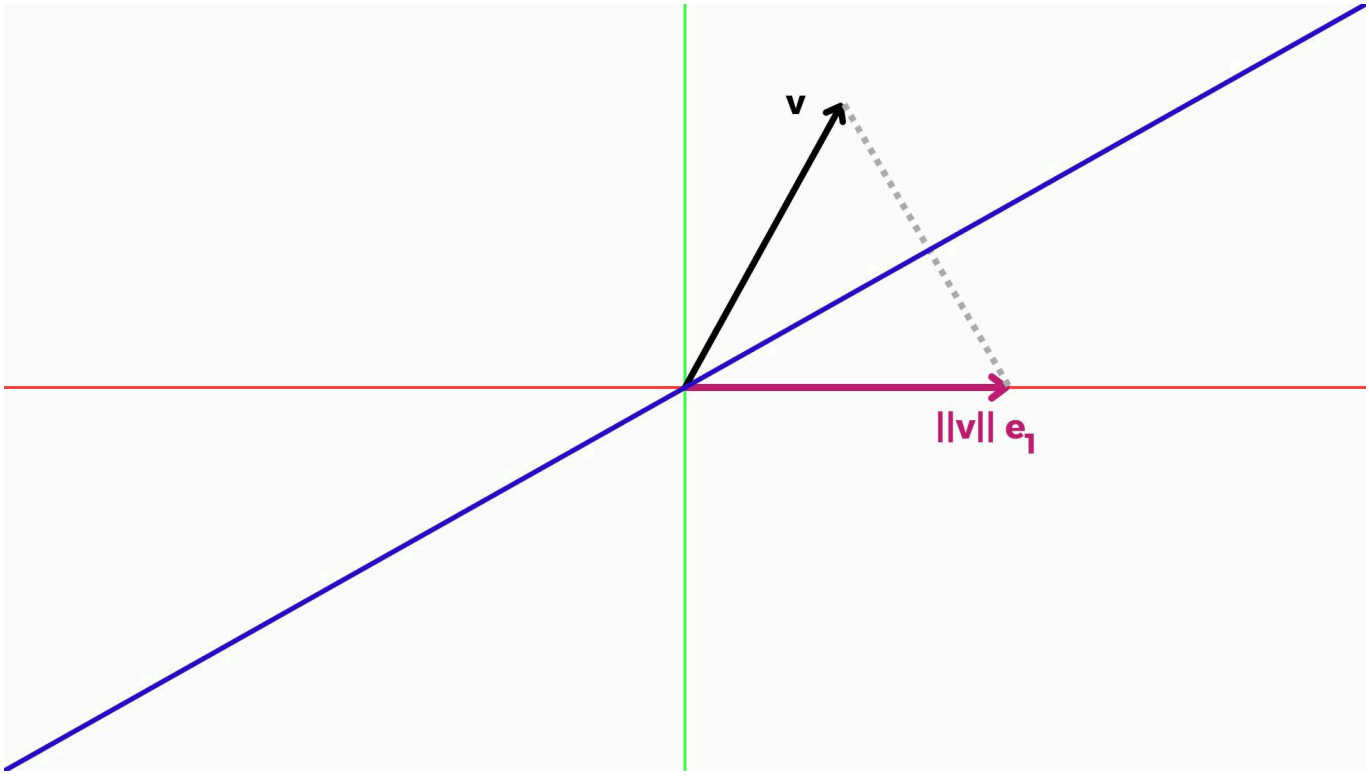
Onde I é a matriz identidade de tamanho $k-1 \times k-1$ e F é uma matriz unitária $m-k+1 \times m-k+1$. Mas por que essa estrutura, onde vimos isso? Bem, lembre-se que, como vimos antes, quando multiplicamos $Q_{k-1}...Q_1A$ por Q_k, queremos manter as linhas 1 a $k-1$ intocadas, então, para fazer isso, criamos uma matriz bloco $k-1 \times k-1$ da identidade para manter essas linhas intocadas.

E por que F é uma matriz unitária? Bem, sabemos que, por causa dos zeros abaixo de I e acima de F, as colunas de F serão ortogonais às colunas de I independentemente do que eu colocar ali, mas queremos que Q_k seja ortogonal, então, se as colunas de I já são ortonormais, só precisamos que as colunas de F também sejam ortonormais, ou seja, F deve ser ortogonal.

Ok, mas agora a parte mais difícil, precisamos que, quando multiplicarmos por F, ela introduza zeros abaixo da k -ésima entrada da diagonal e ainda seja ortogonal. Vamos fazer F estar em $\mathbb{C}^{m-k+1 \times m-k+1}$ e afetar vetores de $\mathbb{C}^{m-k+1}$ (podemos ver as linhas abaixo das k-ésimas entradas de vetores desse espaço), então queremos que a primeira entrada dos vetores seja diferente de 0 e o resto seja 0, sabendo que matrizes ortogonais são rotações em um espaço, podemos fazer F fazer isso:

$$x = \begin{pmatrix} x \\ x \\ x \\ \vdots \\ x \end{pmatrix} \rightarrow Fx = \begin{pmatrix} \|x\| \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \|x\|e_1$$

Como mostrado nesta figura:



Observe que projetar v para obter $\|v\|e_1$ não nos dará uma projeção ortogonal, mas podemos projetá-lo na linha azul, que é a bissetriz do ângulo entre v e $\|v\|e_1$. Essa bissetriz forma um ângulo de 90 graus com $\|v\|e_1 - v$. Este é um plano 2D, então é a bissetriz do ângulo, mas em um espaço de dimensão maior, será um hiperplano ortogonal a $\|v\|e_1 - v$. Vamos definir $w = \|v\|e_1 - v$, então, se H é o hiperplano ortogonal a w, podemos projetar v sobre H fazendo:

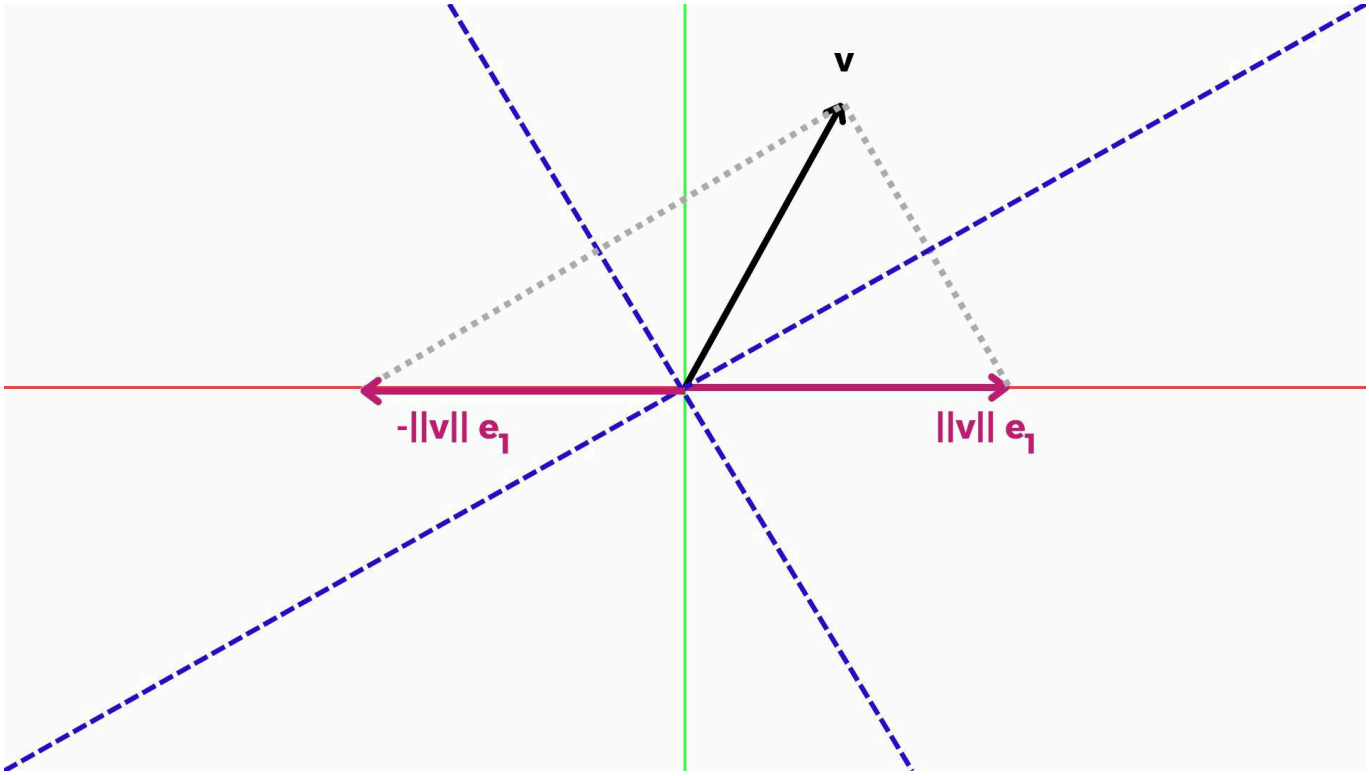
$$Pv = I - \frac{ww^*}{w^*w}v$$

Mas, como podemos ver, se projetarmos v sobre H, para chegar a $\|v\|e_1$, precisamos percorrer duas vezes a distância que acabamos de percorrer, então, a equação final para o refletor de Householder é:

$$F = I - 2\frac{ww^*}{w^*w}$$

6.3. O Melhor de Dois Refletores

Na verdade, podemos ter muitos refletores de Householder, por exemplo, no caso complexo, podemos projetar v em qualquer vetor $z\|v\|e_1$ com $|z| = 1$. No caso real, temos duas alternativas:



Então, o que devo escolher? Qual vetor é melhor para meu algoritmo? Todos serão a mesma coisa? Na verdade, há uma melhor opção que você pode escolher! Matematicamente, todos são a mesma coisa, mas para estabilidade numérica (insensibilidade a erros de arredondamento), escolheremos o $z\|v\|e_1$ que não está muito próximo de v, para alcançar isso, projetaremos em $-\text{sign}(v_1)\|v\|e_1$ onde v_1 é a primeira entrada de v, isso significa:

$$w = -\text{sign}(v_1)\|v\|e_1 - v \vee w = \text{sign}(v_1)\|v\|e_1 + v$$

E podemos definir que:

$$\text{sign}(0) = 1$$

Só para esclarecer por que fizemos essa escolha, imagine que o ângulo entre v e $\|v\|e_1$ é MUITO PEQUENO, isso significa que, quando fazemos $\|v\|e_1 - v$, estamos subtraindo quantidades próximas, dependendo de quais quantidades, isso poderia nos levar a cálculos imprecisos, levando a grandes erros

6.4. O Algoritmo

Agora podemos reescrevê-lo como um algoritmo, mas antes disso:

> **Definição 6.4.1**: Dada a matriz A, $A_{i:i',j:j'}$ é a submatriz $(i' - i + 1) \times (j' - j + 1)$ de A com o elemento do canto superior esquerdo igual a $(A)_{ij}$ e o elemento do canto inferior direito igual a $(A)_{i'j'}$. Se a submatriz for um vetor linha ou coluna, podemos escrevê-lo como $A_{i,j:j'}$ ou $A_{i:i',j}$

Dada essa definição, vamos reescrever o algoritmo

1 **para** $k = 1$ **até** n
2 | $x = A_{k:m,k}$
3 | $v_k = \text{sign}(x_1)\|x\|e_1 + x$
4 | $v_k = \frac{v_k}{\|v_k\|}$
5 | $A_{k:m,k:n} = A_{k:m,k:n} - 2v_k\left(v_k^* A_{k:m,k:n}\right)$

6.5. Aplicando na formação de Q

Observe que não construímos a matriz Q inteira no algoritmo, apenas aplicamos:

$$Q^* = Q_n ... Q_1 \Leftrightarrow Q = Q_1 ... Q_n$$

(Não há asteriscos faltando, porque cada Q_j é hermitiana!)

Fazemos isso porque construir Q requer trabalho extra, então trabalhamos diretamente com Q_j. Por exemplo, lembra que podemos reescrever $b = Ax$ como $Q^*b = Rx$? Bem, podemos fazer isso como no algoritmo anterior:

1 **para** $k = 1$ **até** n
2 | $b_{k:m} = b_{k:m} - 2v_k(v_k^* b_{k:m})$

Observe que fizemos o mesmo processo que fizemos com A, só não explicitei as partes onde defini v_k e o normalizei.

7. Lecture 11 - Problemas de Mínimos Quadrados

Qual é o problema que estamos tentando analisar aqui? Bem, temos um conjunto de m equações com n variáveis, e temos mais equações do que variáveis ($m \geq n$), e queremos encontrar uma solução para esse sistema! Mas você concorda comigo, se fizermos a fatoração QR de A, a maioria dessas equações não terá solução, certo? Porque as entradas abaixo da n-ésima linha de R serão iguais a 0, então, para o vetor Q^*b ter todas as entradas iguais a zero abaixo da n-ésima linha, apenas algumas escolhas específicas de b satisfarão isso!

$$A = QR \Rightarrow Ax = b \Leftrightarrow Rx = Q^*b$$

Então, o que podemos fazer com esse sistema? Ignorá-lo? Bem, de forma alguma! Sabemos que b terá uma solução apenas se estiver em $C(A)$, isso significa que, se $b \notin C(A)$, temos:

$$b - Ax = r \ (r \neq 0)$$

Então, poderíamos encontrar uma maneira de tornar r o menor possível, então nosso novo objetivo é **minimizar** $b - Ax$. Para medir quão pequeno é r, podemos escolher qualquer norma, mas a norma 2 é uma boa escolha e tem algumas propriedades boas para trabalhar.

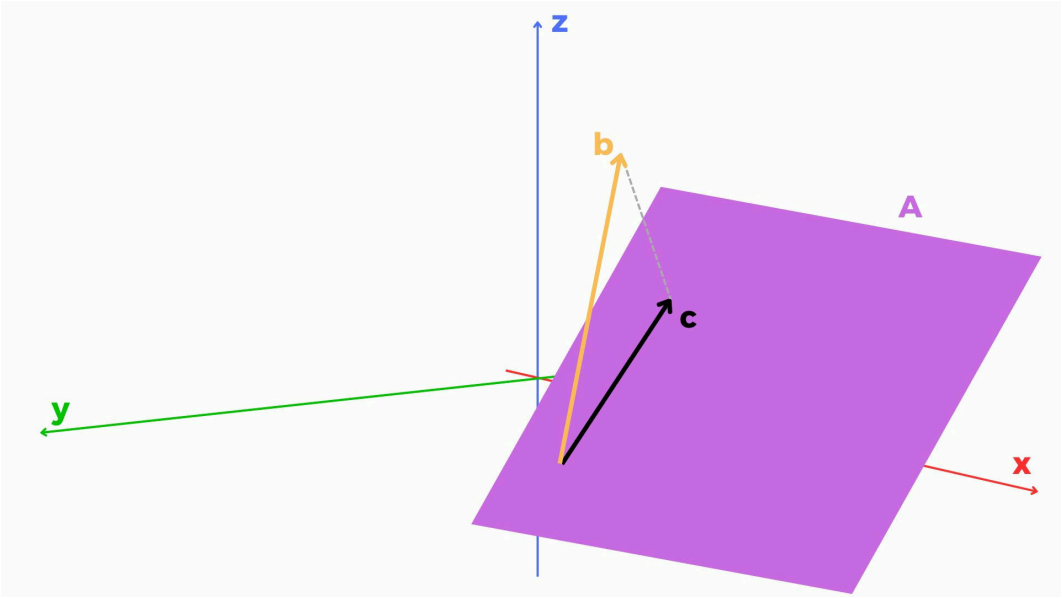
Dado $A \in \mathbb{C}^{m \times n}$, $m \geq n$ e $b \in \mathbb{C}^m$

encontrar $x \in \mathbb{C}^n$ tal que $\|b - Ax\|_2$ seja minimizado

Então, como resolvemos isso? Existe uma maneira fixa para resolver esse problema? O que fazemos? Na verdade, existe uma maneira fixa para resolvê-lo, e ela gira em torno da **projeção ortogonal**

7.1. Projeções Ortogonais e as Equações Normais

Faz sentido que a solução seja obtida por uma projeção de b em $C(A)$, mas que tipo de projeção? Você pode imaginar um plano 3D, se visualizar A como um plano, faz sentido que o Ax que minimiza $\|b - Ax\|_2$, veja a imagem na próxima página:



Ok, parece correto, e podemos pensar nisso intuitivamente, mas está matematicamente correto?

Teorema 7.1.1: Seja $A \in \mathbb{C}^{m \times n}$ $(m \geq n)$, $b \in \mathbb{C}^m$.

$$x \text{ minimiza } \|b - Ax\|_2 \Leftrightarrow b - Ax \perp C(A)$$

Prova: Primeiro, defina como P um projetor ortogonal que projeta sobre $C(A)$ e $c = Pb$

Bem, sabemos que $c \perp b - Ax$ (veja a imagem anterior), então, como c está em $C(A)$, podemos expressar $c = Ay$ para algum $y \in \mathbb{C}^n \neq 0$. Vamos escrever tudo:

$$c^*(b - Ax) = 0 \Leftrightarrow y^*A^*(b - Ax) = 0$$

Sabemos que $y \neq 0$, isso significa $A^*(b - Ax) = 0$

$$A^*(b - Ax) = 0 \Leftrightarrow A^*b - A^*Ax = 0 \Leftrightarrow A^*b = A^*Ax$$

Queremos x que satisfaça esta equação, se A^*A é inversível, então

$$x = (A^*A)^{-1}A^*b$$

E observe que, se aplicarmos A em x, isso me dá a fórmula exata da projeção ortogonal de b sobre A:

$$Ax = A(A^*A)^{-1}A^*b$$

□

A equação $A^*b = A^*Ax$ é conhecida como "equação normal", e usá-la nos permite obter alguns algoritmos diferentes para calcular a solução de mínimos quadrados!

7.1.1. Padrão

Se A tem posto completo, isso significa que A^*A é um sistema de equações quadrado, hermitiano e definido positivo com dimensão n. Então, podemos fazer a Fatoração de Cholesky de A^*A, obtendo R^*R onde R é triangular superior, então podemos fazer a redução:

$$A^*b = A^*Ax \Leftrightarrow A^*b = R^*Rx$$

Então, podemos fazer o algoritmo:

1. Formar a matriz A^*A e A^*b
2. Calcular a Fatoração de Cholesky de A^*A, R^*R
3. Resolver o sistema triangular inferior $R^*w = A^*b$ para w

4. Resolver o sistema triangular superior $Rx = w$ para x

7.1.2. Fatoração QR

Um método "moderno" usa a fatoração QR reduzida. Usando o algoritmo de Householder, calculamos $A = \hat{Q}\hat{R}$ (Lembre-se que $\hat{R}$ é quadrada e $\hat{Q}$ é $m \times n$). Podemos então reescrever o projetor ortogonal $P = (A^*A)^{-1}A$ como $P = \hat{Q}\hat{Q}^*$, porque $C(A) = C(Q)$.

$$\hat{Q}\hat{R}x = \hat{Q}\hat{Q}^*b \Leftrightarrow \hat{R}x = \hat{Q}^*b$$

E se R tem inversa, podemos multiplicá-lo por R^{-1} e ter $A^+ = \hat{R}\hat{Q}^*$

1. Calcular a fatoração QR reduzida $A = \hat{Q}\hat{R}$
2. Calcular o vetor $\hat{Q}^*b$
3. Resolver o sistema triangular superior $\hat{R}x = \hat{Q}^*b$ para x

7.1.3. S.V.D

Se obtivermos $A = \hat{U}\hat{\Sigma}V^*$ (Fatoração S.V.D reduzida), podemos reescrever P como $P = \hat{U}\hat{U}^*$, porque $\hat{U}$ é retangular com colunas ortonormais e $C(A) = C\left(\hat{U}\right)$, então projetar ortogonalmente sobre $C(A)$ é o mesmo que projetar ortogonalmente sobre $C\left(\hat{U}\right)$, análogo ao método QR, temos:

$$\hat{U}\hat{\Sigma}V^*x = \hat{U}\hat{U}^*b \Leftrightarrow \hat{\Sigma}V^*x = \hat{U}^*b$$

Observe que podemos obter uma nova fórmula para A^+, que é $A^+ = V\hat{\Sigma}^{-1}\hat{U}^*$

1. Calcular a S.V.D reduzida $A = \hat{U}\hat{\Sigma}V^*$
2. Calcular o vetor $\hat{U}^*b$
3. Resolver o sistema diagonal $\hat{\Sigma}w = \hat{U}^*b$ para w
4. Definir $x = Vw$

8. Lecture 12 - Condicionamento e Números de Condição

Esta é a parte em que as coisas começam a ficar confusas, então precisaremos passar por isso com muita calma!

8.1. Condicionamento de um Problema

Primeiro, o que é um problema?

> **Definição 8.1.1**: Um *problema* $f : X \to Y$ é uma função de um espaço vetorial normado X de **dados** para um espaço vetorial normado Y de **soluções**

Por que defini um problema assim? Porque, nas próximas aulas, queremos identificar e medir o que acontece quando fornecemos dados ligeiramente diferentes para esse problema. Dado que temos certos dados e os passamos para o problema, e ele nos dá uma solução, a solução difere muito? Apenas um pouco? É a mesma solução?

> **Definição 8.1.2**: Um problema **bem-condicionado** é aquele em que todas as pequenas perturbações de $x \in X$ geram pequenas diferenças em $f(x) \in Y$, ou seja, se fizermos perturbações em x, as soluções do problema não mudarão muito

> **Definição 8.1.3**: Um problema **mal-condicionado** é aquele em que uma pequena perturbação de $x \in X$ gera uma grande diferença nas soluções $f(x) \in Y$, ou seja, alterar os dados, mesmo que ligeiramente, me dará respostas completamente diferentes para o problema

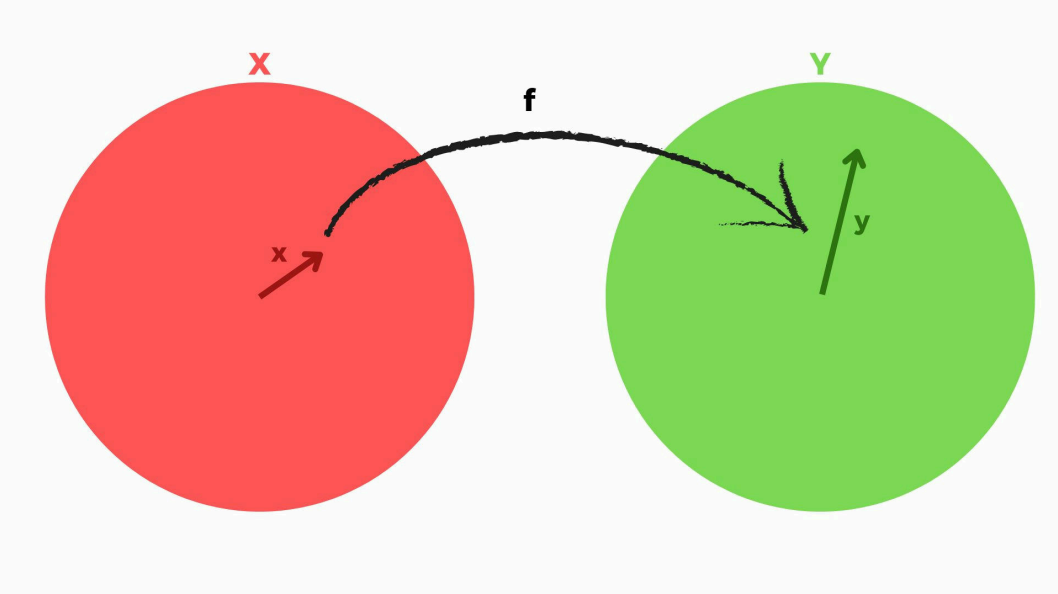
O significado de **pequeno** e **grande** depende do contexto que estou analisando. E como posso medir esse tipo de mudanças? Como quantifico essas mudanças "**pequenas**" e "**grandes**"? Vou defini-lo aqui e mostrar como poderíamos visualizá-lo

Definição 8.1.4 (Número de Condição Absoluto): Seja Δx uma pequena perturbação de x e escreva $\Delta f = f(x + \Delta x) - f(x)$. O **número de condição absoluto** do problema f é definido como

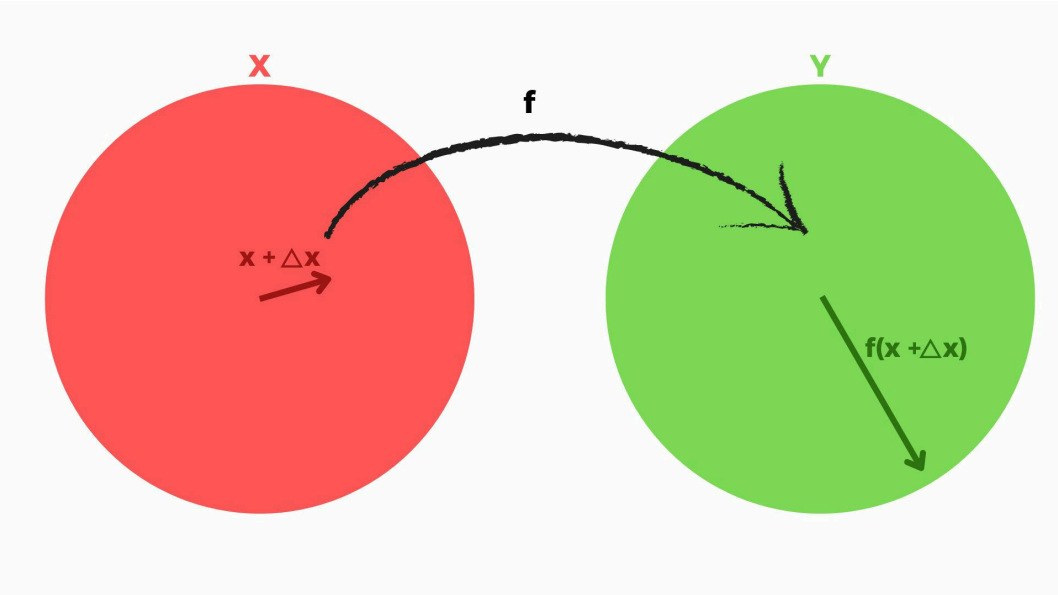
$$\hat{\kappa} = \lim_{\Delta \to 0} \sup_{\|\Delta x\| < \Delta} \left(\frac{\|\Delta f\|}{\|\Delta x\|} \right)$$

Para melhor entendimento, podemos reescrevê-lo como $\hat{\kappa} = \sup_{\Delta x} \frac{\|\Delta f\|}{\|\Delta x\|}$, ou seja, o supremo sobre todas as perturbações infinitesimais em x (entendendo que Δx e Δf são infinitesimais)

Nossa, não entendi NADA! Calma, vamos tentar desenhar aqui:



Primeiro, tenho os espaços normados X e Y e como o problema f aplica uma transformação em x. Agora, vamos fazer um pequeno ajuste em x, tendo Δx, esse novo vetor é uma perturbação infinitesimal em x, quase a mesma coisa, agora vamos dar uma olhada em como f afeta Δx:



NOSSA, observe como uma ligeira perturbação em x mudou a solução MUITO? Isso significa que este é um problema *mal-condicionado*, e o **maior** dessas perturbações é o **número de condição absoluto** de f

Estamos falando de problemas em espaços **normados**, portanto, em sua essência, poderíamos dizer que os problemas são funções de subespaços de $\mathbb{C}^m$ para $\mathbb{C}^n$, o que significa que, se um problema f é *diferenciável* (é uma função, então pode ou não ser diferenciável), ele tem uma *Jacobiana*. Há uma afirmação que diz, *"Se* f :

$X \to Y$ *é diferenciável, então* $f(x+\Delta x) \approx f(x) + J(x)\Delta x$ *quando* $\Delta x \to 0$", bem, estamos trabalhando com $\Delta x \to 0$, então o que acontece se substituirmos $f(x+\Delta x)$ por $f(x) + J(x)\Delta x$:

$$\hat{\kappa} = \lim_{\Delta\to 0} \sup_{\|\Delta x\|<\Delta} \left(\frac{\|\Delta f\|}{\|\Delta x\|} \right) = \lim_{\Delta\to 0} \sup_{\|\Delta x\|<\Delta} \left(\frac{\|f(x+\Delta x) - f(x)\|}{\|\Delta x\|} \right) = \lim_{\Delta\to 0} \sup_{\|\Delta x\|<\Delta} \left(\frac{\|f(x) + J(x)\Delta x - f(x)\|}{\|\Delta x\|} \right)$$

$$\lim_{\Delta\to 0} \sup_{\|\Delta x\|<\Delta} \left(\frac{\|J(x)\Delta x\|}{\|\Delta x\|} \right) \leq \lim_{\Delta\to 0} \sup_{\|\Delta x\|<\Delta} \left(\frac{\|J(x)\|\,\|\Delta x\|}{\|\Delta x\|} \right) = \|J(x)\|$$

Isso significa que o valor supremo sobre todas as variações infinitesimais de x será $\|J(x)\|$, ou seja

$$\hat{\kappa} = \|J(x)\|$$

Definição 8.1.5 (Número de Condição Relativo): Seja Δx uma pequena perturbação de x e escreva $\Delta f = f(x+\Delta x) - f(x)$. O **número de condição relativo** do problema f é definido como

$$\kappa = \lim_{\Delta\to 0} \sup_{\|\Delta x\|\leq\Delta} \frac{\frac{\|\Delta f(x)\|}{\|f(x)\|}}{\frac{\|\Delta x\|}{\|x\|}}$$

Por que definimos um número de condição "relativo"? Porque estou tentando medi-lo relativamente.

Nossa! Você só repetiu a mesma coisa... Calma. Imagine que um engenheiro está construindo um foguete de 1000m, e ele mede um erro de 1m, é um pequeno erro, certo? Mas e se o foguete tiver 2m? É um erro COLOSSAL, certo? Esse é o ponto, estamos tentando medir o erro causado por Δx com base no tamanho de x e sua solução $f(x)$

Bem, lembre-se que podemos representar o número de condição absoluto como

$$\hat{\kappa} = \|J(x)\|$$

Podemos fazer algo semelhante com κ, vamos ver:

$$\kappa = \lim_{\Delta\to 0} \sup_{\|\Delta x\|\leq\Delta} \frac{\frac{\|\Delta f(x)\|}{\|f(x)\|}}{\frac{\|\Delta x\|}{\|x\|}} = \lim_{\Delta\to 0} \sup_{\|\Delta x\|\leq\Delta} \frac{\|\Delta f(x)\|}{\|f(x)\|}\frac{\|x\|}{\|\Delta x\|} = \lim_{\Delta\to 0} \sup_{\|\Delta x\|\leq\Delta} \frac{\|\Delta f(x)\|}{\|\Delta x\|}\frac{\|x\|}{\|f(x)\|} = \|J(x)\|\frac{\|x\|}{\|f(x)\|}$$

8.2. Condicionamento de Matrizes e Vetores

Agora, um conceito importante para estabilidade e condicionamento é como a multiplicação de vetores e matrizes é condicionada, as multiplicações de vetores são *bem-condicionadas*? *mal-condicionadas*? Como as matrizes são condicionadas? Vamos ver

8.2.1. Condição da Multiplicação Matriz-Vetor

Teorema 8.2.1.1: Dado $A \in \mathbb{C}^{m\times n}$ fixo, o problema de calcular Ax com x no espaço normado de dados, o número de condição do problema da matriz é

$$\kappa = \|A\|\,\frac{\|x\|}{\|Ax\|}$$

Se A é quadrada e inversível:

$$\kappa \leq \|A\|\,\|A^{-1}\|$$

Prova: Fixe $A \in \mathbb{C}^{m\times n}$ e o problema de calcular Ax, com x sendo os dados, ou seja, calcularemos a condição desse problema com base em perturbações de x, não de A, A será fixo o tempo todo.

$$\kappa = \sup_{\Delta x}\left(\frac{\|f(x+\Delta x)-f(x)\|}{\|f(x)\|}\frac{\|x\|}{\|\Delta x\|}\right) = \sup_{\Delta x}\left(\frac{\|A(x+\Delta x)-Ax\|}{\|Ax\|}\frac{\|x\|}{\|\Delta x\|}\right) = \sup_{\Delta x}\left(\frac{\|A\Delta x\|}{\|Ax\|}\frac{\|x\|}{\|\Delta x\|}\right)$$

$$\kappa \leq \sup_{\Delta x}\left(\frac{\|A\|\,\|\Delta x\|}{\|Ax\|}\frac{\|x\|}{\|\Delta x\|}\right) = \|A\|\frac{\|x\|}{\|Ax\|}$$

Se A é quadrada e inversível, podemos usar o fato de que $\frac{\|x\|}{\|Ax\|} \leq \|A^{-1}\|$ (prová-lo-emos depois), para expressar κ como:

$$\kappa \leq \|A\|\,\|A^{-1}\| \vee \kappa = \alpha\,\|A\|\,\|A^{-1}\|$$

□

Corolário 8.2.1.1.1: Seja $A \in \mathbb{C}^{m\times n}$ não singular e considere a equação $Ax = b$. O problema de calcular b dado x tem número de condição

$$\kappa = \|A\|\frac{\|x\|}{\|b\|}$$

Teorema 8.2.1.2:

$$\frac{\|x\|}{\|Ax\|} \leq \|A^{-1}\|$$

Prova: Escreva x como $x = A^{-1}(Ax)$, isso significa

$$\|x\| = \|A^{-1}Ax\| \leq \|A^{-1}\|\,\|Ax\| \Leftrightarrow \frac{\|x\|}{\|Ax\|} \leq \|A^{-1}\|$$

□

8.2.2. Número de Condição de uma Matriz

O quê? Por que podemos dar um número de condição a uma matriz? Porque elas são **funções**! Por quê? Lembre-se que podemos representar toda **transformação linear** como uma **multiplicação matricial**? Isso implica que, se uma matriz A está em $\mathbb{C}^{m\times n}$, então ela pode ser representada como $A : \mathbb{C}^n \to \mathbb{C}^m$! Agora que entendemos por que elas podem ter um número de condição, vamos defini-lo

Definição 8.2.2.1: Dado $A \in \mathbb{C}^{m\times m}$ não singular, o número de condição de A, denotado como $\kappa(A)$, é:

$$\kappa(A) = \|A\|\,\|A^{-1}\|$$

Se A é singular

$$\kappa(A) = \infty$$

Se A é retangular

$$\kappa(A) = \|A\|\,\|A^{+}\|,\ \left(A^{+} = (A^{*}A)^{-1}A\right)$$

8.2.3. Condição de Sistemas de Equações

Teorema 8.2.3.1: Dado um sistema $Ax = b$, vamos manter b fixo e considerar o problema $A \mapsto x = A^{-1}b$, o número de condição desse problema é:

$$\kappa(A)$$

Prova: Se perturbarmos A e x, fazemos:

$$(A + \Delta A)(x + \Delta x) = b$$

$$\Leftrightarrow Ax + A(\Delta x) + (\Delta A)x + (\Delta A)(\Delta x) = b$$

$$\Leftrightarrow b + A(\Delta x) + (\Delta A)x + (\Delta A)(\Delta x) = b$$

$$\Leftrightarrow A(\Delta x) + (\Delta A)x + (\Delta A)(\Delta x) = 0$$

Sabemos que $(\Delta A)(\Delta x)$ é duplamente infinitesimal e ambos estão indo para 0, então podemos descartá-lo, e obtemos

$$A(\Delta x) + (\Delta A)x = 0 \Leftrightarrow \Delta x = -A^{-1}(\Delta A)x$$

Esta equação implica que

$$\|\Delta x\| \leq \|A^{-1}\| \, \|\Delta A\| \, \|x\| \Leftrightarrow \|\Delta x\| \, \|A\| \leq \|A^{-1}\| \, \|A\| \, \|\Delta A\| \, \|x\|$$

$$\frac{\|\Delta x\|}{\|x\|} \frac{\|A\|}{\|\Delta A\|} \leq \|A^{-1}\| \, \|A\|$$

Se fizermos $\Delta x \to 0$ e $\Delta A \to 0$, temos

$$\kappa = \|A\| \, \|A^{-1}\| = \kappa(A)$$

□

9. Lecture 13 - Aritmética de Ponto Flutuante

Quando estamos analisando algoritmos e computadores, temos um problema **realmente** grande. Computadores são máquinas discretas, o que significa que, quando falamos de números reais, eles não podem representar **todos** eles, há uma quantidade finita de números que podem representar, dependendo de como esses computadores são construídos. A maioria dos computadores usa um sistema binário para representar números reais, mas eles poderiam usar outros sistemas. Existem dois grandes problemas na representação de números reais:

1. **Underflow & Overflow**: Como eu disse, um computador pode representar um número finito de números reais, isso significa que há um máximo e um mínimo nesse conjunto. Se eu tentar representar um número maior que esse máximo, terei um erro de **overflow**, portanto, tentar representar um número menor, terei um erro de **underflow**. Hoje em dia, isso não é um grande problema, a maioria dos computadores é capaz de armazenar números muito grandes e muito pequenos, suficientes para os problemas com os quais vamos trabalhar
2. **Gap**: Quando tentamos representar números reais, há um problema, porque entre dois números reais, existem infinitos outros números reais, o que nos leva ao problema do **gap**, porque, se o conjunto de números que o computador pode representar é finito, podemos contá-los, e se podemos contá-los, podemos obter uma infinidade de outros números reais entre eles. O problema do **gap** não é realmente um **PROBLEMA**, mas quando estamos criando algoritmos, queremos que eles sejam o mais precisos possível, porque um algoritmo instável pode nos levar a grandes erros de arredondamento

9.1. Conjunto de Ponto Flutuante

O conjunto de números que um computador pode entender e representar é chamado de **Conjunto de Ponto Flutuante**. O livro nos dá uma definição bem complicada, então vou te dar uma mais simples e, depois, entender a definição do livro:

> **Definição 9.1.1** (Definição Simples): Um **Conjunto de Ponto Flutuante** é definido como:
>
> $$F = \{\pm 0, d_1 d_2 ... d_t * \beta^e / e \in \mathbb{Z}\}$$
>
> Onde $0, d_1 ... d_t$ é a **mantissa**, $t \in \mathbb{N}^*$ é a **precisão** da mantissa, $\beta \in \mathbb{N} (\beta \geq 2)$ é a base, $d_j \big(0 \leq d_j < \beta\big)$ são os dígitos da mantissa e $e \in \mathbb{Z}$ é o **expoente**. Em computadores, **nós** definimos um intervalo para t, e e uma base específica β

Vamos analisar tudo. Primeiro, definimos tudo, mas o que e por que defini essas coisas?

9.1.1. Mantissa

É o núcleo de um número, um número nesse conjunto tem apenas uma mantissa, mas uma mantissa pode estar associada a vários números, por exemplo, podemos representar 1 no sistema decimal como $0,1 * 10$ e podemos representar 10 como $0,1 * 10^2$, isso significa que a mesma mantissa pode representar muitos números. É importante dizer que a mantissa é sempre escrita na base β, veremos o que isso significa agora

9.1.2. Base e Precisão

> **Definição 9.1.2.1**: Se temos $x \in \mathbb{R}$, escrever x na base β significa escolher coeficientes $..., \alpha_{-1}, \alpha_0, \alpha_1, ...$ com $0 \leq \alpha_j < \beta$ e α_j são inteiros tais que:
>
> $$x = ... + \alpha_2 \beta^2 + \alpha_1 \beta^1 + \alpha_0 \beta^0 + \alpha_{-1} \beta^{-1} + \alpha_{-2} \beta^{-2} + ...$$
>
> Então escrevemos x na base β como $...\alpha_2 \alpha_1 \alpha_0, \alpha_{-1} \alpha_{-2} ..._\beta$, onde cada α_j é um dígito

Essa é uma maneira de representar cada número real, mas o computador é limitado a uma certa quantidade de α_j, ele não pode armazenar, por exemplo, 1 bilhão de α_j, é aí que entra a precisão, ela diz ao computador quantos dígitos a mantissa pode representar!

Espere... mas vimos que a mantissa só armazena números após a ",", então como o conjunto representa números maiores que 1? É aí que entra o **expoente**

9.1.3. Expoente

O expoente indica a ordem do nosso número, por exemplo, com uma mantissa $0, d_1 d_2 d_3$, podemos representar 4 números diferentes: $0, d_1 d_2 d_3$, $d_1, d_2 d_3$, $d_1 d_2, d_3$, $d_1 d_2 d_3$, eles têm expoentes $0, 1, 2$ e 3, respectivamente. Mas, se eu tenho uma mantissa $0, d_1 ... d_t$, por que multiplicá-la por β^e move a vírgula por e dígitos? (Só para fazer um paralelo, é exatamente assim que nosso sistema decimal funciona, se você tem $0,1$, multiplicá-lo por 10 dá 1)

> **Teorema 9.1.3.1**: Se você tem x escrito na base β, multiplicá-lo por β^e moverá a vírgula, na notação da base β, e dígitos para a direita

Prova:

$$x = ... + \alpha_2 \beta^2 + \alpha_1 \beta^1 + \alpha_0 \beta^0 + \alpha_{-1} \beta^{-1} + \alpha_{-2} \beta^{-2} + ...$$

$$\Leftrightarrow \beta^e x = (... + \alpha_2 \beta^2 + \alpha_1 \beta^1 + \alpha_0 \beta^0 + \alpha_{-1} \beta^{-1} + \alpha_{-2} \beta^{-2} + ...) \beta^e$$

$$\Leftrightarrow \beta^e x = ... + \alpha_2 \beta^2 \beta^e + \alpha_1 \beta^1 \beta^e + \alpha_0 \beta^0 \beta^e + \alpha_{-1} \beta^{-1} \beta^e + \alpha_{-2} \beta^{-2} \beta^e + ...$$

$$\Leftrightarrow \beta^e x = ... + \alpha_2 \beta^{e+2} + \alpha_1 \beta^{e+1} + \alpha_0 \beta^e + \alpha_{-1} \beta^{e-1} + \alpha_{-2} \beta^{e-2} + ...$$

Observe como todos os α_j moveram e dígitos para a esquerda, isso significa que a vírgula move e dígitos para a direita □

Vamos analisar um exemplo para entender melhor:

Exemplo: Criei uma máquina que representa meus números por um conjunto de ponto flutuante F com base decimal ($\beta = 10$), precisão 3 e $e \in [-5, 5]$, responda estas perguntas:

1. Qual é o menor número positivo que F pode representar?

   Bem, se F tem precisão 3, minha mantissa é assim:

   $$0, d_1 d_2 d_{3_\beta}$$

   Então, se queremos o mínimo, precisamos tornar d_j o menor possível, mas ainda diferente de 0, isso significa que fazemos o último dígito (d_3) igual a 1 e o resto igual a 0, isso significa que a menor mantissa que podemos ter é:

   $$0,001$$

   Mas ainda podemos representar um número maior usando o expoente, o menor expoente que temos é -5, isso significa que o menor número positivo que esse conjunto pode representar é

   $$0,001 * 10^{-5} = 0,00000001$$

2. Qual é o maior número positivo que F pode representar?

   Seguindo a mesma lógica, a maior mantissa que podemos ter é:

   $$0,999$$

   Usando o maior expoente possível, o maior número que nosso conjunto pode representar é:

   $$0,999 * 10^5 = 99900$$

3. O número –3921 está no conjunto?

   Vamos testar, se o decompusermos:

   $$-3921 = -0,3921 * 10^4$$

   O expoente está no intervalo $[-5, 5]$, mas a mantissa tem precisão 4, isso significa que -3921 **NÃO ESTÁ NO CONJUNTO F**

4. O número 738000000 está no conjunto?

   Decompondo, temos: $738000000 = 0,738 * 10^9$, como podemos ver, a mantissa tem a precisão desejada, mas o expoente é maior que o intervalo dado, isso significa que 738000000 **NÃO ESTÁ NO CONJUNTO F**

Agora que entendemos essa definição de conjunto de ponto flutuante, vamos ver a definição do livro:

Definição 9.1.3.1 (Definição do Livro): Sendo $F \subset \mathbb{R}$, definimos como:

$$F = \left\{ \pm \left(\frac{m}{\beta^t} \right) \beta^e \right\}$$

Onde t, e e β significam a mesma coisa com as mesmas restrições da definição anterior

Qual é a diferença e por que o livro define assim? Há apenas uma grande diferença aqui: Por que ele está definindo a **mantissa** como $\frac{m}{\beta^t}$? Lembre-se de quando provamos que, se escrevemos x na base β e multiplicamos x por β^e, a vírgula move e dígitos para a direita? O mesmo se aplica se $e < 0$, mas a vírgula vai para a esquerda, e por que estou dizendo isso? Porque, o que o livro não nos diz, é que escrevemos m **NA BASE β**, e essa divisão

por β^t faz com que obtenhamos apenas os primeiros t dígitos do número, significando que obtemos apenas os dígitos que desejamos, SÓ ISSO (Sim, o livro explica isso mal)

9.2. Números não em F

Quando tentamos representar um número que não está em F, o computador pode fazer 2 coisas:

1. **Arredondar**: Obtemos $t+1$ dígitos do número, e verificamos se o $(t+1)$-ésimo dígito é maior ou igual a $\left\lceil \frac{\beta}{2} \right\rceil$, se for, excluímos o $(t+1)$-ésimo dígito e somamos 1 ao t-ésimo dígito. Se o $(t+1)$-ésimo dígito for menor que $\left\lceil \frac{\beta}{2} \right\rceil$, então apenas excluímos o $(t+1)$-ésimo dígito

   Exemplo: Arredonde 10324 sabendo que F tem precisão 4 e $e \in [-\infty, +\infty]$ e $\beta = 10$.

   Convertendo para a notação de mantissa e expoente: $10324 = 0,10324 * 10^5$, temos 5 dígitos, então vamos ver o 5-ésimo. $4 \geq \left\lceil \frac{10}{2} \right\rceil \Leftrightarrow 4 \geq 5$? Não, então o número arredondado será $0,1032 * 10^5$
2. **Truncar**: Se a mantissa do número passar de t dígitos, removemos todos os dígitos após o t-ésimo

Sabendo disso, podemos finalmente entender o que é $\varepsilon_{\text{machine}}$

9.3. Épsilon Máquina

Vamos ver a primeira definição do livro, que é:

$$\varepsilon_{\text{machine}} = \frac{1}{2}\beta^{1-t}$$

Mas o que isso significa? Por que ele definiu assim? Primeiro, $\varepsilon_{\text{machine}}$ é o número que, se fizermos essa operação em F, será válida

$$1 + \varepsilon_{\text{machine}} > 1$$

Isso significa que, se somarmos 1 com um número menor que $\varepsilon_{\text{machine}}$, mesmo que por uma diferença infinitesimal, o número retornado será arredondado ou truncado para 1 em F. O livro diz que essa definição é a distância entre 2 números representáveis em F, mas por que isso?

Teorema 9.3.1: A distância entre 2 números representáveis em F é β^{1-t}

Prova: Se temos x escrito na notação da base β com precisão t, escrevemo-lo como:

$$x = 0, d_1 d_2 ... d_{t_\beta}$$

Se queremos incrementar algo nesse número, mas sem fazer com que ele saia da precisão possível e fazendo o menor incremento possível, podemos adicionar 1 a d_t. Então vamos fazer isso e ver o que acontece, escrevendo x:

$$x = d_1\beta^{-1} + d_2\beta^{-2} + ... + d_t\beta^{1-t}$$

Se somarmos 1 a d_t:

$$\alpha = d_1\beta^{-1} + d_2\beta^{-2} + ... + (d_t + 1)\beta^{1-t}$$

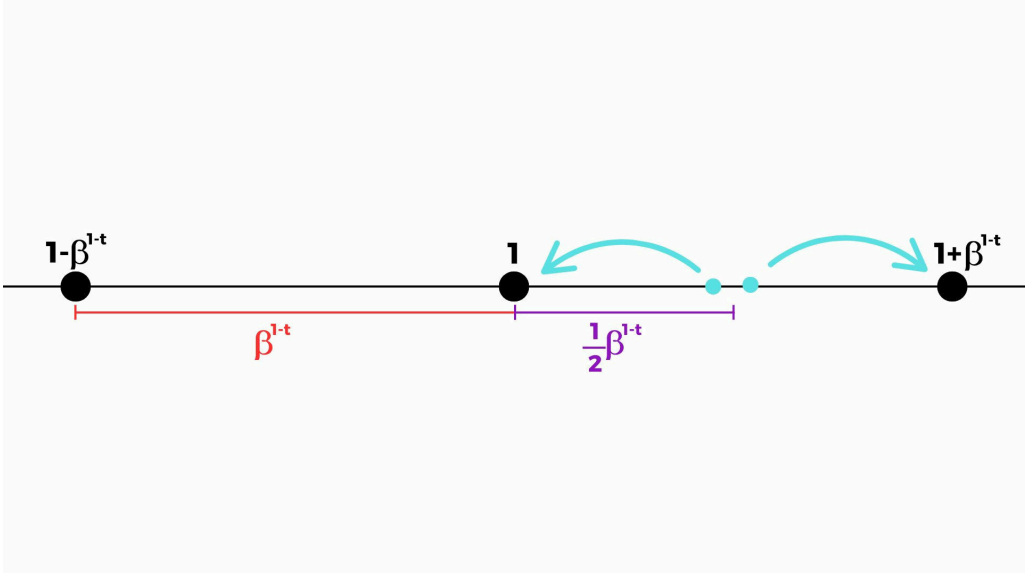
$$\Leftrightarrow \alpha = d_1\beta^{-1} + d_2\beta^{-2} + ... + d_t\beta^{1-t} + \beta^{1-t}$$

Mas observe como podemos reescrever isso como

$$\alpha = x + \beta^{1-t}$$

Isso significa que α (o próximo número representável), é $x + \boldsymbol{\beta^{1-t}}$, ou seja, a distância entre eles □

Agora podemos visualizá-lo como uma linha, onde temos os números representáveis e, se tentarmos representar um número que está no intervalo entre eles, o computador o arredondará com base em $\varepsilon_{\text{machine}}$



Os pontos azul-ciano representam números reais que não podem ser representados inteiramente por F, e as setas mostram para onde o computador os arredonda. Mudaremos essa definição mais tarde, e você entenderá por que depois.

O livro nos mostra uma desigualdade que todo $\varepsilon_{\text{machine}}$ deve satisfazer, mas essa desigualdade pode ser reescrita:

Definição 9.3.1: Seja F um conjunto de ponto flutuante. $\text{fl} : \mathbb{R} \to F$ é uma função que retorna a aproximação arredondada da entrada x no conjunto F

Teorema 9.3.2 (Conversão de Ponto Flutuante): $\forall x \in \mathbb{R}$, existe ε com $|\varepsilon| \leq \varepsilon_{\text{machine}}$ tal que:

$$\text{fl}(x) = x(1 + \varepsilon)$$

O que isso significa? Significa que, sempre que arredondamos um número real para ajustá-lo em F, o número arredondado é equivalente a multiplicar x por $1 +$ um número muito pequeno, você pode visualizá-lo olhando para a representação em linha de F que mostrei antes

9.4. Aritmética de Ponto Flutuante

Precisamos fazer operações com números, certo? Mas temos o mesmo problema, os computadores precisam arredondar porque não conseguem entender todos os números em um intervalo, então como podemos tornar as operações o mais precisas possível? Construímos um computador baseado neste princípio (alguns computadores podem ter mais princípios em seu núcleo, então algumas operações podem ser ainda mais precisas, mas vamos focar apenas neste):

Definição 9.4.1 (Axioma Fundamental da Aritmética de Ponto Flutuante): Dado que $+$, $-$, $\times$ e $\div$ representam operações em $\mathbb{R}$, considere $\oplus$, $\ominus$, $\otimes$ e $\oslash$ sendo operações em F. Seja $\circledast$ definir qualquer uma das operações anteriores em F, então definimos um computador que realiza a operação $x \circledast y$ como

$$x \circledast y = \text{fl}(x * y) = (x * y)$$

Isso significa que construímos um computador tal que $\forall x, y \in F$, existe ε com $|\varepsilon| \leq \varepsilon_{\text{machine}}$ tal que

$$x \circledast y = (x * y)(1 + \varepsilon)$$

Em outras palavras, toda operação em F tem um erro com tamanho **no máximo** $\varepsilon_{\text{machine}}$

9.5. Mais sobre Épsilon Máquina

Agora podemos redefinir $\varepsilon_{\text{machine}}$! Mas por quê? Bem, queremos torná-lo o menor possível, e às vezes aumentar a precisão não é uma opção:

> **Definição 9.5.1**: $\varepsilon_{\text{machine}}$ é o menor valor tal que Teorema 9.3.2 e Definição 9.4.1 são válidos

Isso implica que, para alguns computadores, $\varepsilon_{\text{machine}}$ pode ser ainda menor que $\frac{1}{2}\beta^{1-t}$, o que é uma coisa **muito** boa!

10. Lecture 14 e 15 - Estabilidade

Quando falamos de **estabilidade**, estamos tentando verificar se um algoritmo tem fidelidade no computador! Isso significa, se os arredondamentos que o computador faz nas entradas e saídas não mudarão o resultado para algo muito diferente do original. Mas primeiro, precisamos definir matematicamente o que é um algoritmo!

> **Definição 10.1**: Seja um problema $f : X \to Y$ e um computador com sistema de ponto flutuante que satisfaz Definição 9.4.1 fixado. O algoritmo de f, $\tilde{f} : X \to F^n \subset Y$, é uma função que representa uma série de passos e suas implementações no computador dado com o objetivo de resolver o problema f
>
> **Nota:** $F^n \subset Y$ apenas representa que tenho um vetor de números que podem ser representados por F e esse vetor está em Y

Bem, sabemos que, se passarmos $x \in X$ para esse algoritmo, o resultado $\tilde{f}(x)$ pode ser afetado por erros de arredondamento! Na maioria dos casos, $\tilde{f}$ não é uma função contínua, mas o algoritmo precisa aproximar $f(x)$ da melhor forma possível.

Vamos fazer uma definição para um algoritmo **estável** também! Primeiro, vamos definir a **precisão** de um algoritmo

> **Definição 10.2** (Precisão do Algoritmo): Um algoritmo $\tilde{f}$ é preciso se:
>
> $$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O(\varepsilon_{\text{machine}})$$

O QUÊ? O QUE DIABOS $O(\varepsilon_{\text{machine}})$ SIGNIFICA??? Calma, calma, farei uma definição formal mais tarde, por agora, você pode entender que um algoritmo é preciso se o erro entre a saída do algoritmo e a saída original não ultrapassa $\varepsilon_{\text{machine}}$

Em algoritmos mal-condicionados, a igualdade mostrada na definição é muito ambiciosa, porque erros de arredondamento são inevitáveis, nesse tipo de algoritmos, esse arredondamento pode causar grandes erros, ultrapassando os erros desejados que queríamos

Agora podemos definir um algoritmo **estável**

> **Definição 10.3** (Algoritmo Estável): Um algoritmo $\tilde{f}$ para um problema f é estável se
>
> $$\forall x \text{ é válido que } \frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = O(\varepsilon_{\text{machine}})$$
>
> $$\text{para um } \tilde{x} \text{ com } \frac{\|\tilde{x} - x\|}{\|x\|} = O(\varepsilon_{\text{machine}})$$

Espere, o quê? O que essa definição significa?

Significa que, todos os dados semelhantes aos meus dados originais passados para o meu algoritmo retornarão saídas muito semelhantes às soluções corretas (isso é mostrado com $\varepsilon_{\text{machine}}$, ou seja, a diferença entre as soluções dadas pelo algoritmo e as soluções reais não ultrapassará $\varepsilon_{\text{machine}}$)

Existe outro tipo de **estabilidade**, muito poderoso:

Definição 10.4 (Estabilidade Retroativa): Um algoritmo $\tilde{f}$ para f é **estável retroativamente** se:

$$\forall x \in X \text{ é válido que } \exists \tilde{x} \text{ com } \frac{\|\tilde{x} - x\|}{\|x\|} = O(\varepsilon_{\text{machine}}) \text{ tal que } \tilde{f}(x) = f(\tilde{x})$$

Isso significa que, se eu passar os dados para o algoritmo, posso encontrar uma perturbação muito pequena $\tilde{x}$ tal que a solução do problema se eu passar essa perturbação é a **mesma** que se eu passar os dados originais para o algoritmo!

Exemplo: Dado o dado $x \in \mathbb{C}$, verifique se o algoritmo $x \oplus x$ para calcular o problema de somar dois números iguais (solução é $2x$) é estável retroativamente:

Temos que

$$f(x) = x + xe\tilde{f}(x) = x \oplus x$$

Isso significa

$$\tilde{f}(x) = (2x)(1 + \varepsilon)$$

Vamos verificar se esse algoritmo é **estável**. Primeiro, defina $\tilde{x} = x(1 + \varepsilon)$, sabemos que $\varepsilon = O(\varepsilon_{\text{machine}})$, vamos verificar se o erro relativo entre x e $\tilde{x}$ é $O(\varepsilon_{\text{machine}})$:

$$\frac{\|\tilde{x} - x\|}{\|x\|} = \frac{\|x(1 + \varepsilon) - x\|}{\|x\|} = \frac{\|x(1 + \varepsilon - 1)\|}{\|x\|} = |\varepsilon| = O(\varepsilon_{\text{machine}})$$

Então $x(1 + \varepsilon)$ é uma definição **válida** para $\tilde{x}$, deixando isso claro, vamos verificar se $\tilde{f}$ é estável

$$\frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = \frac{\|2x(1 + \varepsilon) - 2x(1 + \varepsilon)\|}{\|f(\tilde{x})\|} = 0 = O(\varepsilon_{\text{machine}})$$

Isso significa que $\tilde{f}$ é estável, mas é **estável retroativamente**? Precisamos de uma definição de $\tilde{x}$ que ainda satisfaça a condição de x definida em Definição 10.3. Vamos verificar se nossa definição satisfaz, já verificamos que a condição é válida, mas ela satisfaz $f(\tilde{x}) = \tilde{f}(x)$?

$$\tilde{f}(x) = 2x(1 + \varepsilon)$$

$$f(\tilde{x}) = 2x(1 + \varepsilon)$$

Isso significa que esse algoritmo **É** de fato **estável retroativamente**

10.1. Definição Formal de $O(\varepsilon_{\text{machine}})$

Vou escrever a definição aqui e explicar o que significa logo depois

Definição 10.1.1: Dadas as funções $\varphi(t)$ e $\psi(t)$, a sentença

$$\varphi(t) = O(\psi(t))$$

significa que $\exists C > 0$ tal que $\forall t$ suficientemente próximo de um limite conhecido (por exemplo, $t \to 0$, $t \to \infty$), é válido que:

$$\varphi(t) \leq C\psi(t)$$

O que isso significa? Significa que, se escrevemos $\varphi(t) = O(\psi(t))$, e sabemos para onde t está indo, existe $C > 0$ tal que os valores de $\varphi(t)$ nunca serão maiores que $C\psi(t)$. Na maioria das vezes, eu nem me importo com o que é C, só me importo com sua existência!

Falando sobre como estamos tratando $\varepsilon_{\text{machine}}$, o limite implícito aqui é $\varepsilon_{\text{machine}} \to 0$, e escrever que $\varphi(t) = O(\varepsilon_{\text{machine}})$ significa que temos uma constante que limita o erro a uma quantidade de $\varepsilon_{\text{machine}}$, ou seja, o erro nunca será maior que, por exemplo, 3 vezes $\varepsilon_{\text{machine}}$, 2 e meio vezes $\varepsilon_{\text{machine}}$.

Podemos fazer uma definição formal mais forte (mas mais confusa) para a notação O

Definição 10.1.2: Dado $\varphi(s, t)$, temos que:

$$\varphi(s, t) = O(\psi(t)) \text{ uniformemente em } s$$

garante que $\exists! C > 0$ tal que:

$$\varphi(s, t) \leq C\psi(t)$$

e isso é válido para qualquer s que eu escolher

É uma definição semelhante, estou apenas adicionando uma variável que posso escolher e que não mudará nada.

Em computadores reais, $\varepsilon_{\text{machine}}$ é um número fixo, então quando estamos trabalhando com o limite implícito $\varepsilon_{\text{machine}} \to 0$, estamos selecionando uma família **ideal** de computadores!

10.2. Dependência de m e n

Na prática, quando falamos de erros de arredondamento, a estabilidade de algoritmos envolvendo uma matriz A não depende da própria A, mas de m e n (suas dimensões). Podemos ver isso analisando o seguinte problema:

Suponha que eu tenha um algoritmo para resolver um sistema não singular $m \times m$ $Ax = b$ para x e garantimos que a solução $\tilde{x}$ dada pelo algoritmo satisfaz

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\kappa(A)\varepsilon_{\text{machine}})$$

Isso significa que existe uma constante C que satisfaz

$$\|\tilde{x} - x\| \leq C\kappa(A)\varepsilon_{\text{machine}} \ \|x\|$$

Isso mostra que, mesmo C não dependendo nem de A nem de b, acaba dependendo das dimensões de A porque, se mudarmos m ou n, os dados passados para o problema mudam, o que significa que teremos um **novo** problema porque estamos mudando seu domínio e $\kappa(A)$ também mudará se alterarmos suas dimensões!

10.3. Independência da Norma

Você pode ter notado que, quando estamos definindo coisas em estabilidade com normas, denotamos $\|\cdot\|$ como **qualquer** tipo de norma, mas, se escolhermos uma certa norma, a definição pode não ser válida, certo? Como, pode ser válida para $\|\cdot\|_2$, mas não para $\|\cdot\|_3$, certo? Na verdade, errado! Podemos mostrar que **precisão**, **estabilidade** e **estabilidade retroativa** mantêm suas propriedades para **qualquer** tipo de norma! Isso significa que, quando escolhemos uma norma, podemos escolher uma que facilite os cálculos!

Teorema 10.3.1: Para problemas f e seus algoritmos $\tilde{f}$ em espaços normados de dimensão finita X e Y, as propriedades de **precisão**, **estabilidade** e **estabilidade retroativa** são válidas ou não independentemente de qual norma eu escolher para fazer a análise

Prova: Se provarmos que, se $\|\cdot\|$ e $\|\cdot\|'$ são duas normas em X e Y, então $\exists c_1, c_2$ tais que:

$$c_1\|x\| \leq \|x\|' \leq c_2\|x\|$$

então o teorema mostrado antes é válido, porque isso mostra:
1. Se uma sequência converge ou é muito pequena em uma norma, ela será em todas as outras normas também
2. Pequenos erros em uma norma serão pequenos em todas as outras normas também

Mas precisamos provar a afirmação anterior, certo? Vamos fazer isso! (O livro apenas diz que é fácil, lol)

Primeiro, vamos reduzir o problema a uma esfera unitária das normas, vamos definir:

$$S = \{x \in \mathbb{C}^n / \ \|x\| = 1\}$$

esse conjunto é **fechado** porque a norma é **contínua** e é **limitado** (uma esfera, lol). Agora vamos definir a função $f(x) = \|x\|'$, porque $f(x)$ é contínua em $\mathbb{C}^n$ e S é fechado e limitado, podemos encontrar o **máximo** e o **mínimo** valor de f em S, vamos definir:
1. $m = \min_{x \in S} \|x\|'$
2. $M = \max_{x \in S} \|x\|'$

$m > 0$ porque $0 \notin S$. Agora, vamos tentar generalizar em $\mathbb{C}^n$. Se queremos generalizar para todo $x \neq 0 \in \mathbb{C}^n$, vamos escrever:

$$x = \|x\| \left(\frac{x}{\|x\|} \right)$$

Você pode ver claramente que $\frac{x}{\|x\|} \in S$ porque $\|\frac{x}{\|x\|}\| = 1$. Então, vamos ver o que acontece se tomarmos $\|x\|'$:

$$\|x\|' = \| \ \|x\| \left(\frac{x}{\|x\|} \right) \ \|' = \|x\| \ \| \frac{x}{\|x\|} \|'$$

Se você olhar de perto, $\frac{x}{\|x\|}$ é um vetor em S, isso significa que $\|\frac{x}{\|x\|}\|' \in [m, M]$ e, por causa de $\|x\|$ (número escalar positivo), podemos ver que

$$\|x\| \ \| \frac{x}{\|x\|} \|' \in [\|x\|m, \|x\|M]$$

Podemos reescrever isso como

$$m\|x\| \leq \|x\|' \leq M\|x\|$$

Isso significa que essas duas constantes existem e provam o teorema estabelecido antes □

10.4. Estabilidade da Aritmética de Ponto Flutuante

Teorema 10.4.1: As operações $\oplus$, $\ominus$, $\otimes$ e $\oslash$ são **estáveis retroativamente**

Prova: Defina $\circledast$ como qualquer uma das 4 operações mostradas antes. Dado um problema $f : X \to Y$ que está calculando $x_1 * x_2$, o algoritmo $\tilde{f}$ para resolver esse problema é $\tilde{f}(x) = \text{fl}(x_1) \circledast \text{fl}(x_2)$ onde $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$.

Temos que:

$$\tilde{f}(x) = \text{fl}(x_1) \circledast \text{fl}(x_2)$$

$$= (\text{fl}(x_1) * \text{fl}(x_2))(1 + \varepsilon_3)$$

$$= (x_1(1 + \varepsilon_1) * x_2(1 + \varepsilon_2))(1 + \varepsilon_3)$$

$$= x_1(1 + \varepsilon_1)(1 + \varepsilon_3) * x_2(1 + \varepsilon_2)(1 + \varepsilon_3)$$

$$= x_1(1 + \varepsilon_4) * x_2(1 + \varepsilon_5)$$

Onde $\varepsilon_4 = O(\varepsilon_{\text{machine}})$ e $\varepsilon_5 = O(\varepsilon_{\text{machine}})$. Calculamos $\tilde{f}(x)$, agora vamos ver $f(\tilde{x})$. Primeiro, vamos definir:

$$\tilde{x} = \begin{pmatrix} x_1(1+\varepsilon_4) \\ x_2(1+\varepsilon_5) \end{pmatrix}$$

Se definirmos $\tilde{x}$ assim, podemos ver claramente que

$$f(\tilde{x}) = x_1(1+\varepsilon_4) + x_2(1+\varepsilon_5) = \tilde{f}(x)$$

Mas a condição $\frac{\|\tilde{x}-x\|}{\|x\|} = O(\varepsilon_{\text{machine}})$ é satisfeita?

$$\frac{\|\tilde{x}-x\|}{\|x\|} = \frac{\left\|\begin{pmatrix} x_1(1+\varepsilon_4) \\ x_2(1+\varepsilon_5) \end{pmatrix} - \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right\|}{\left\|\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right\|} = \frac{\left\|\begin{pmatrix} x_1\varepsilon_4 \\ x_2\varepsilon_5 \end{pmatrix}\right\|}{\left\|\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right\|}$$

Usando a norma 1

$$\frac{x_1\varepsilon_4 + x_2\varepsilon_5}{x_1+x_2} = \frac{x_1 O(\varepsilon_{\text{machine}}) + x_2 O(\varepsilon_{\text{machine}})}{x_1+x_2} = \frac{(x_1+x_2)O(\varepsilon_{\text{machine}})}{x_1+x_2} = O(\varepsilon_{\text{machine}})$$

Isso mostra que $\oplus$, $\ominus$, $\otimes$ e $\oslash$ são **estáveis retroativamente** □

10.5. Precisão de um Algoritmo Estável Retroativamente

Falamos de números de condição antes da estabilidade, vamos tentar associar ambos!

Teorema 10.5.1: Suponha que um algoritmo estável retroativamente $\tilde{f}$ é aplicado para um problema $f : X \to Y$ com número de condição κ em um computador que satisfaz Teorema 9.3.2 e Definição 9.4.1, então, o erro relativo satisfaz:

$$\frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = O(\kappa(x)\varepsilon_{\text{machine}})$$

Prova: Por definição, temos $\tilde{f}(x) = f(x+\delta x)$ com $\frac{\|\delta x\|}{\|x\|} = O(\varepsilon_{\text{machine}})$. Usando Definição 8.1.5 (Número de Condição Relativo), temos que:

$$\kappa(x) = \lim_{\delta x \to 0} \left(\frac{\|f(x+\delta x) - f(x)\|}{\|f(x)\|} \right) \left(\frac{\|x\|}{\|\delta x\|} \right)$$

$$\kappa(x) = \lim_{\delta x \to 0} \left(\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} \frac{\|x\|}{\|\delta x\|} \right)$$

Usando algumas definições formais (nem eu entendo, então se tentar explicar aqui, só perderei tempo, lol), podemos reescrever isso como:

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} \leq (\kappa(x) + o(1)) \frac{\|\delta x\|}{\|x\|}$$

Onde $o(1) \to 0$ quando $\varepsilon_{\text{machine}} \to 0$ □