BACHELOR'S THESIS / ENGINEERING 2025

Desenvolupament d'una API gràfica interactiva per a l'optimització de la probabilitat d'error en comunicacions digitals

Alba Soldevilla González

upf. Universitat Pompeu Fabra Barcelona

Desenvolupament d'una API gràfica interactiva per a l'optimització de la probabilitat d'error en comunicacions digitals

TREBALL DE FI DE GRAU DE
Alba Soldevilla González

Alfonso Martinez & Josep Font-Segura
Grau en Enginyeria de Sistemes Audiovisuals
Curs 2024–2025

Universitat Pompeu Fabra
Barcelona

Escola d'Enginyeria

A quienes me han acompañado en este proceso, en especial a mi familia y a mis profesores, por su apoyo y compromiso.

Abstract

This Bachelor's Thesis is part of a collaborative project that studies the error probability in digital communication systems. In its first phase, the mathematical analysis and optimization of the model were carried out using various computational techniques and methods such as Newton's Method and Hermite interpolation, achieving a notable improvement in computational efficiency.

As a personal contribution, this work presents the development of an API that performs these calculations through a web interface. Implemented with Python and Docker, it provides endpoints for computing and visualizing results. The frontend, built with D3.js, enables the generation of interactive graphs and the application of various functionalities. The result is an accessible tool for the analysis of digital communication systems.

Resum

Aquest Treball de Fi de Grau forma part d'un projecte col·laboratiu que estudia la probabilitat d'error en sistemes de comunicació digital. En una primera fase s'ha treballat l'anàlisi i optimització matemàtica del model mitjançant diferents tècniques de càlcul i mètodes com el Mètode de Newton i la interpolació d'Hermite, obtenint una millora notable en eficiència computacional.

Com a contribució pròpia, es presenta el desenvolupament d'una API que permet executar aquests càlculs mitjançant una interfície web. Implementada amb Python i Docker, ofereix punts d'accés per calcular i visualitzar resultats. El *frontend*, creat amb D3.js, permet generar gràfics interactius i aplicar diverses funcionalitats. El resultat és una eina accessible per a l'anàlisi de sistemes de comunicació digitals.

Prefaci

M'agradaria explicar breument el camí que m'ha portat a estar avui escrivint aquest prefaci. Tot començà dos anys enrere quan el pla d'estudis de la universitat em portà a conèixer una de les assignatures més destacades, i no de manera positiva, de tota la carrera: TIC (Teoria de la Informació i Codificació). Va ser allí que vaig conèixer els qui avui serien els meus tutors, Alfonso i Josep, dos professors ben posats i populars entre els alumnes pels seus famosos seminaris tipus test inacabables i les seves classes indesxifrables. Tanmateix, la dinàmica del curs em va captivar fins al punt de deixar-m'hi la pell per aconseguir l'aprovat, cosa que l'examen final i la recuperació d'aquell mateix any em van impedir. La vaig haver de repetir. Tot i aprovar-la l'any de després, el destí no em volia separar del camí dels meus professors. Me'ls vaig tornar a topar a DDT (Transmissió Digital de Dades), una altra assignatura amb la mateixa fama que TIC. També la vaig acabar repetint. Tot i semblar que mai tenia èxit amb ells, fins al punt que ni els Kahoots de classe se'm feien amens, no vaig dubtar ni un segon a contactar amb ells quan vaig veure els seus noms en la llista de propostes de treballs de fi de grau de la universitat. No sé si era la seva exigència que em cridava l'atenció o les seves peculiars personalitats, però tot alhora em portà un dia al seu despatx per demanar la seva tutoria. He d'admetre que el tema, en general, m'era bastant indiferent; jo només sabia que, amb el seu suport, podia arribar lluny. I així va ser. No sé si jo he arribat a complir les seves expectatives, però ells, sens dubte, han arribat a complir les meves.

Sumari

Introducció

Vivim en una era en què la comunicació digital és omnipresent i essencial en gairebé tots els àmbits de la societat. Cada missatge que enviem, cada dada que compartim i cada interacció que realitzem depenen de sistemes de comunicació dissenyats amb precisió per garantir la integritat, la confidencialitat i la disponibilitat de la informació. Això implica tenir en compte diversos factors, com ara el canal de transmissió, les interferències, la codificació de la informació i les tècniques de detecció i correcció d'errors [2, 3].

Aquest projecte aborda el problema des de dues perspectives complementàries: l'anàlisi teòrica del càlcul de la probabilitat d'error en canals digitals, i el desenvolupament d'una interfície de programació d'aplicacions (API) que permeti accedir de manera senzilla i eficient a aquestes eines de càlcul. Mentre que l'apartat analític ha estat desenvolupat de manera col·laborativa amb els tutors i altres membres del grup de recerca, el disseny i la implementació de la interfície interactiva ha estat una contribució pròpia.

El procés del treball s'ha estructurat en diverses fases. En primer lloc, s'ha dut a terme una anàlisi dels models i mètodes per calcular la funció d'exponent d'error, estudiant-ne l'optimització i les possibles aproximacions. Posteriorment, s'ha dissenyat i implementat una API interactiva en un entorn web utilitzant eines com FastAPI i Docker, que permet invocar aquests càlculs des d'una interfície senzilla.

Un aspecte destacat del projecte ha estat la construcció de la interfície, fent èmfasi en la implementació de gràfiques interactives amb diverses funcionalitats, amb l'objectiu de facilitar la comprensió i l'anàlisi de les solucions obtingudes.

En resum, aquest projecte combina rigor analític i desenvolupament pràctic, posant de manifest la importància de fer accessibles els càlculs tècnics complexos mitjançant solucions tecnològiques modernes. El resultat és una eina que reflecteix tant l'esforç col·lectiu com la iniciativa individual per fer-lo útil i aplicable.

Capítol 1

Anàlisi matemàtica i optimització del model

Aquest capítol presenta l'anàlisi matemàtica del sistema de comunicació considerat, així com les tècniques d'optimització emprades per al càlcul eficient de l'exponent d'error $E(R)$.

Començarem definint el model de transmissió i canal utilitzat, així com les expressions teòriques de la probabilitat d'error i l'exponent associat. A continuació, s'estudia la funció generadora d'exponents $E_0(\rho)$, clau per a l'anàlisi del rendiment del sistema, i es proposen estratègies per calcular-la numèricament de manera eficient.

Finalment, es descriuen diferents mètodes d'optimització, incloent-hi aproximacions derivades, interpolació i mètodes iteratius, que permeten trobar de forma ràpida i precisa el valor òptim de ρ, necessari per calcular la probabilitat d'error en el nostre model.

1.1 Plantejament del problema i model de comunicació

Considerem un sistema on un transmissor genera un senyal d'entrada X, el qual pren valors d'una constel·lació de símbols denotada χ, amb una distribució de probabilitat $Q(x)$. El transmissor envia símbols amb un *rate* R i una longitud de codi n, per tant, el nombre de missatges és $M = 2^{nR}$.

Definim el codi com $\{x^{(1)}, \ldots, x^{(M)}\}$, on $x^{(m)} = (x_1^{(m)}, \ldots, x_n^{(m)})$, i es normalitza de manera que

$$\mathbb{E}\left[\|x\|^2\right] = \sum_{x \in \chi} \|x\|^2 Q(x) = 1. \tag{1.1}$$

El símbol X és transmès a través d'un canal de comunicació que introdueix distorsió en forma de soroll additiu (AWGN) [2, 3]. El soroll es modela com una variable aleatòria Z que segueix una distribució normal complexa amb mitjana nul·la i variància unitària, $\mathcal{CN}(0,1)$. El canal transforma el senyal X i genera una sortida

$$Y = \sqrt{\text{SNR}} \cdot X + Z. \tag{1.2}$$

La probabilitat de transmissió del canal es defineix com:

$$W(y \mid x) = \frac{1}{\pi} e^{-\|y - \sqrt{\text{SNR}} \cdot x\|^2}. \tag{1.3}$$

El receptor rep el vector $(y_1, \ldots, y_n)$ i estima la paraula transmesa com $\hat{m} = g(y_1, \ldots, y_n)$, on g és un descodificador.

A causa de les característiques del canal, la probabilitat d'error pot expressar-se com

$$P_{\text{e}} = 2^{-nE(R)}, \tag{1.4}$$

on $E(R)$ és l'**exponent d'error** del sistema, que determina la taxa de decreixement exponencial de l'error en funció del *rate* de transmissió R. Aquesta funció es defineix com

$$E(R) = \max_{0 \leq \rho \leq 1} \left\{ E_0(\rho) - \rho R \right\}, \tag{1.5}$$

on $E_0(\rho)$ és la **funció generadora d'exponents**, que depèn del paràmetre auxiliar ρ i de les característiques estadístiques del canal [1, 4]. Aquesta es pot escriure com

$$E_0(\rho) = -\log_2 \left[\int_{y \in \mathbb{C}} dy \sum_{x \in \chi} Q(x)\, W(y \mid x) \left(\sum_{\hat{x} \in \chi} Q(\hat{x}) \left(\frac{W(y \mid \hat{x})}{W(y \mid x)} \right)^{\frac{1}{1+\rho}} \right)^{\rho} \right]. \tag{1.6}$$

Per simplificar, definim $G(z) = \frac{1}{\pi} e^{-\|z\|^2}$, de manera que la probabilitat de transmissió queda $W(y \mid x) = G(y - \sqrt{\text{SNR}} \cdot x)$. Amb $z = y - \sqrt{\text{SNR}} \cdot x$ i substituint a $E_0(\rho)$ obtenim

$$E_0(\rho) = -\log_2 \left[\int_{z \in \mathbb{C}} dz \sum_{x \in \chi} Q(x)\, G(z)\, f_\rho(x,z)^\rho \right], \tag{1.7}$$

on

$$f_\rho(x,z) = \sum_{\hat{x} \in \chi} Q(\hat{x}) \cdot \frac{G\left(z + \sqrt{\text{SNR}} \cdot x - \sqrt{\text{SNR}} \cdot \hat{x} \right)^{\frac{1}{1+\rho}}}{G(z)^{\frac{1}{1+\rho}}}, \tag{1.8}$$

$$G(z) = \frac{1}{\pi}\, e^{-|z|^2}. \tag{1.9}$$

Com que la integral és sobre nombres complexos, la dividim en part real i imaginària i apliquem la definició de $G(z)$ en (1.9) a l'equació de $E_0(\rho)$ en (1.7), quedant

$$E_0(\rho) = -\log_2 \left[\frac{1}{\pi} \sum_{x \in \chi} Q(x) \int_{-\infty}^{\infty} dz_R \int_{-\infty}^{\infty} dz_I \, e^{-z_R^2} e^{-z_I^2} f_\rho(x, z_R + j z_I)^\rho \right]. \tag{1.10}$$

Aquesta expressió es pot aproximar numèricament mitjançant la quadratura gaussiana

$$E_0(\rho) \approx -\log_2 \left[\frac{1}{\pi} \sum_{x \in \chi} Q(x) \sum_{i=1}^{N} \sum_{k=1}^{N} w_i w_k f_\rho(x, z_i + j z_k)^\rho \right], \tag{1.11}$$

on $w_1, \ldots, w_N$ són els pesos de la quadratura i $z_1, \ldots, z_N$ són els nodes.

1.2 Optimització del càlcul de $E_0(\rho)$ amb operacions matricials

El càlcul de la funció $E_0(\rho)$ és computacionalment costós a causa de l'estructura de la fórmula i la dependència de múltiples paràmetres dins de bucles niats. Quan s'implementa directament pas a pas, l'enfocament iteratiu fa ús de diversos nivells de bucles `for` per recórrer les quadratures de Gauss-Hermite i avaluar la funció de probabilitat $W(y \mid x)$ per a cada combinació de variables.

L'optimització del càlcul de $E_0(\rho)$ mitjançant operacions matricials és fonamental, ja que *MATLAB* és inherentment més eficient amb aquest tipus d'estructures gràcies al seu disseny intern basat en llibreries optimitzades com BLAS i LAPACK [7].

Considerant la forma aproximada definida a la secció anterior amb la fórmula (1.11) i desenvolupant a dins $f_\rho(x, z)$, podem escriure:

$$E_0(\rho) \approx -\log_2 \left[\frac{1}{\pi} \sum_{x \in \chi} Q(x) \sum_{i=1}^{N} \sum_{k=1}^{N} w_i w_k \cdot G(z)^{-\frac{\rho}{1+\rho}} \cdot \left(\sum_{\hat{x} \in \chi} Q(\hat{x}) \cdot G\left(z + \sqrt{\text{SNR}} \cdot x - \sqrt{\text{SNR}} \cdot \hat{x} \right)^{\frac{1}{1+\rho}} \right)^\rho \right]. \tag{1.12}$$

En notació matricial compacta

$$E_0(\rho) \approx -\log_2 \left[\frac{1}{\pi} \cdot Q^\top \times \left(\pi \cdot G^{-\frac{\rho}{1+\rho}} \right) \times \left(\left(Q^\top \cdot G^{\frac{1}{1+\rho}} \right)^\rho \right)^\top \right], \tag{1.13}$$

on Q és el vector columna de probabilitats associades a cada símbol de la constel·lació χ, definida com

$$Q = \begin{bmatrix} Q(x_1) \\ \vdots \\ Q(x_M) \end{bmatrix}, \quad \text{where } M = |\chi|.$$

Aquí π és una matriu de pesos de quadratura Gauss-Hermite, repetida per cada símbol

$$\pi = \left[\begin{array}{cccc|cccc|c|ccc} w_1 w_1^{(1,1)} & \cdots & \cdots & w_N w_N^{(1,1)} & \mathbf{0}_{1,1}^{(1,2)} & \cdots & \cdots & \mathbf{0}_{N,N}^{(1,2)} & \cdots & \mathbf{0}_{1,1}^{(1,M)} & \cdots & \mathbf{0}_{N,N}^{(1,M)} \\ \mathbf{0}_{1,1}^{(2,1)} & \cdots & \cdots & \mathbf{0}_{N,N}^{(2,1)} & w_1 w_1^{(2,2)} & \cdots & \cdots & w_N w_N^{(2,2)} & \cdots & \mathbf{0}_{1,1}^{(2,M)} & \cdots & \mathbf{0}_{N,N}^{(2,M)} \\ \vdots & \ddots & & \vdots & \vdots & \ddots & & \vdots & \ddots & \vdots & & \vdots \\ \mathbf{0}_{1,1}^{(M,1)} & \cdots & \cdots & \mathbf{0}_{N,N}^{(M,1)} & \mathbf{0}_{1,1}^{(M,2)} & \cdots & \cdots & \mathbf{0}_{N,N}^{(M,2)} & \cdots & w_1 w_1^{(M,M)} & \cdots & w_N w_N^{(M,M)} \end{array}\right],$$

i G és la matriu que conté l'avaluació de la funció $G(z + \sqrt{\text{SNR}} \cdot x - \sqrt{\text{SNR}} \cdot \hat{x})$ per a cada combinació de símbols i punts de la quadratura:

$$G = \begin{bmatrix} G(z_{11}, x_1, x_1) & \cdots & G(z_{NN}, x_1, x_1) & G(z_{11}, x_1, x_2) & \cdots & G(z_{NN}, x_1, x_2) & \cdots & G(z_{11}, x_1, x_M) & \cdots & G(z_{NN}, x_1, x_M) \\ G(z_{11}, x_2, x_1) & \cdots & G(z_{NN}, x_2, x_1) & G(z_{11}, x_2, x_2) & \cdots & G(z_{NN}, x_2, x_2) & \cdots & G(z_{11}, x_2, x_M) & \cdots & G(z_{NN}, x_2, x_M) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ G(z_{11}, x_M, x_1) & \cdots & G(z_{NN}, x_M, x_1) & G(z_{11}, x_M, x_2) & \cdots & G(z_{NN}, x_M, x_2) & \cdots & G(z_{11}, x_M, x_M) & \cdots & G(z_{NN}, x_M, x_M) \end{bmatrix}.$$

La matriu Q té dimensió $M \times 1$, i tant π com G tenen dimensió $M \times N^2 M$. L'operant $(\cdot)$ representa el producte de Hadamard (element a element), mentre que $(\times)$ representa una multiplicació matricial clàssica.

Ara hem implementat $E_0(\rho)$ en MATLAB per comparar els temps d'execució utilitzant bucles iteratius enfront d'operacions matricials. A la Figura 1.1 es mostren els temps obtinguts en funció de M, que representa el nombre de símbols de la modulació PAM.

Com s'observa clarament, les operacions matricials ofereixen una millora significativa en eficiència computacional a mesura que M augmenta. Mentre que els bucles presenten un creixement gairebé exponencial del temps d'execució, la versió matricial escala molt millor. Això demostra l'avantatge pràctic de vectoritzar càlculs en entorns com MATLAB, especialment en escenaris amb altes dimensions o grans volums de dades.

1.3 Implementació numèrica de la derivació de $E_0(\rho)$

Per tal de calcular numèricament l'exponent $E_0(\rho)$, hem definit una nova funció basada en la seva expressió en termes de $F(\rho)$, donada per

$$E_0(\rho) = -\log_2 \left(F(\rho) \right), \tag{1.14}$$

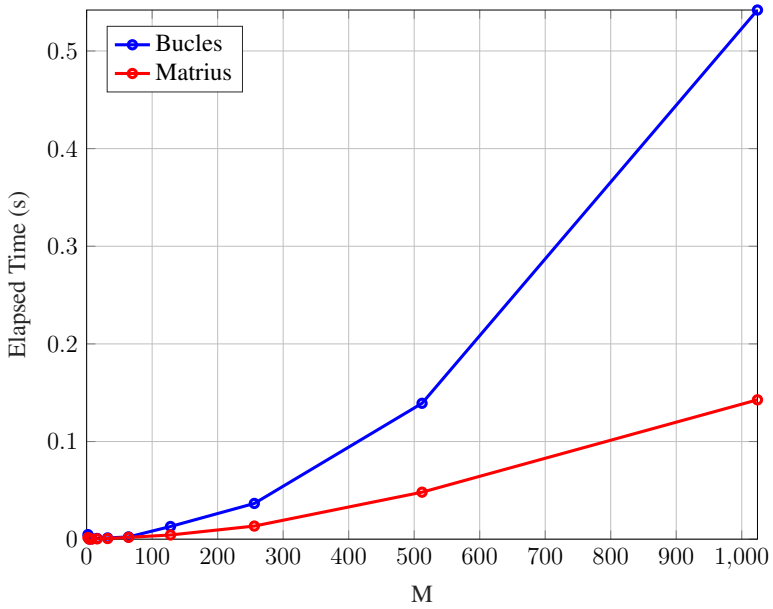


Figura 1.1: Temps d'execució de $E_0(\rho = 1)$ respecte M: bucles vs. matrius

on la funció $F(\rho)$ es defineix com:

$$F(\rho) = \frac{1}{\pi} \sum_{x \in \chi} Q(x) \sum_{i=1}^{N} \sum_{k=1}^{N} w_i \cdot w_k \cdot f_\rho(x, z_i + j z_k)^\rho. \tag{1.15}$$

Amb l'objectiu de poder aplicar, més endavant, mètodes d'optimització per trobar el valor òptim de ρ, ha estat necessari calcular la primera i segona derivada d'$F(\rho)$ per poder definir també les derivades de $E_0(\rho)$ basades en termes d'$F(\rho)$, les quals es defineixen com

$$\frac{\mathrm{d}}{\mathrm{d}\rho} E_0(\rho) = \frac{-\frac{\mathrm{d}}{\mathrm{d}\rho} F(\rho)}{\ln(2) \cdot F(\rho)}, \tag{1.16}$$

$$\frac{\mathrm{d}^2}{\mathrm{d}\rho^2} E_0(\rho) = -\frac{1}{\ln(2)} \left(\frac{\frac{\mathrm{d}^2}{\mathrm{d}\rho^2} F(\rho)}{F(\rho)} - \left(\frac{\frac{\mathrm{d}}{\mathrm{d}\rho} F(\rho)}{F(\rho)} \right)^2 \right), \tag{1.17}$$

on les derivades d'$F(\rho)$ queden definides de la següent manera

$$\frac{\mathrm{d}}{\mathrm{d}\rho} F(\rho) = \frac{1}{\pi} \sum_{x \in \chi} Q(x) \sum_{i=1}^{N} \sum_{k=1}^{N} w_i w_k \cdot f(x, z_i + j z_k, \rho)^\rho \cdot \log f(x, z_i + j z_k, \rho) \tag{1.18}$$

$$\frac{\mathrm{d}^2}{\mathrm{d}\rho^2} F(\rho) = \frac{1}{\pi} \sum_{x \in \chi} Q(x) \sum_{i=1}^{N} \sum_{k=1}^{N} w_i w_k \cdot f(x, z_i + j z_k, \rho)^\rho \cdot [\log f(x, z_i + j z_k, \rho)]^2 \tag{1.19}$$

i les seves formes matricials queden

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\rho}F_0(\rho,Q) = Q^T \times \left(\pi \cdot e^{-s\rho\ln W} \cdot (-s'\rho\ln W - s\ln W)\right) \\ \times \left(e^{\rho\ln(Q^T\cdot e^{s\ln W})}\right)^T + Q^T \times \left(\pi \cdot e^{-s\rho\ln W}\right) \\ \times \left(e^{\rho\ln(Q^T\times e^{s\ln W})} \cdot \left(\ln(Q^T \times e^{s\ln W}) \right.\right. \\ \left.\left. + \rho \cdot \frac{Q^T \times (s'\ln W \cdot e^{s\ln W})}{Q^T \times e^{s\ln W}}\right)\right)^T,\end{aligned} \tag{1.20}$$

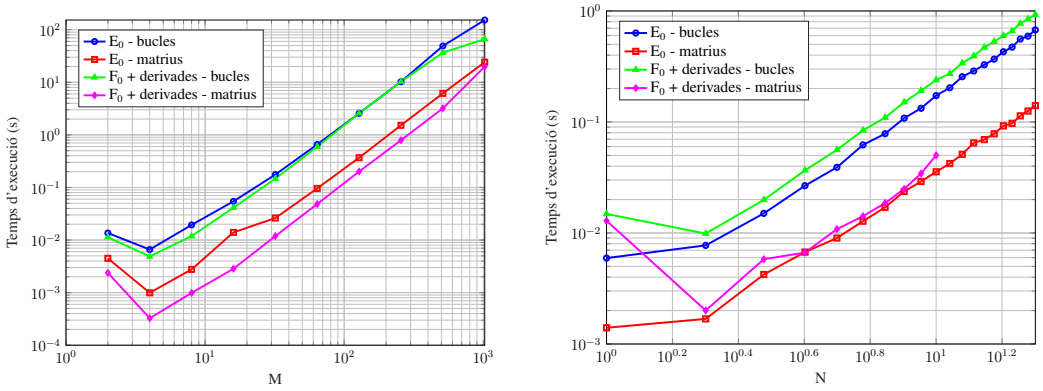
$$\begin{aligned}\frac{\mathrm{d}^2}{\mathrm{d}\rho^2}F_0(\rho,Q) &= Q^T \times \left(\pi \cdot e^{-s\rho\ln W} \cdot (-s'\rho\ln W - s\ln W)^2\right) \times \left(e^{\rho\ln(Q^T\times e^{s\ln W})}\right)^T \\ &+ Q^T \times \left(\pi \cdot e^{-s\rho\ln W} \cdot (-s''\rho\ln W - 2s'\ln W)\right) \times \left(e^{\rho\ln(Q^T\times e^{s\ln W})}\right)^T \\ &+ 2Q^T \times \left(\pi \cdot e^{-s\rho\ln W} \cdot (-s'\rho\ln W - s\ln W)\right) \\ &\quad \times \left(e^{\rho\ln(Q^T\times e^{s\ln W})} \cdot \left(\ln(Q^T \times e^{s\ln W}) \right. + \rho\frac{Q^T \times (s'\ln W \cdot e^{s\ln W})}{Q^T \times e^{s\ln W}}\right)\right)^T \\ &+ Q^T \times \left(\pi \cdot e^{-s\rho\ln W}\right) \times \left(e^{\rho\ln(Q^T\times e^{s\ln W})} \cdot \left(\ln(Q^T \times e^{s\ln W}) \right.\right. \\ &\qquad \left.\left. + \rho\frac{Q^T \times (s'\ln W \cdot e^{s\ln W})}{Q^T \times e^{s\ln W}}\right)^2\right)^T \\ &+ Q^T \times \left(\pi \cdot e^{-s\rho\ln W}\right) \times \left(e^{\rho\ln(Q^T\times e^{s\ln W})} \cdot \left(2\frac{Q^T \times (s'\ln W \cdot e^{s\ln W})}{Q^T \times e^{s\ln W}}\right.\right. \\ &\qquad + \rho \cdot \left(\frac{Q^T \times \left((s''\ln W + (s')^2\ln^2 W)\cdot e^{s\ln W}\right)}{Q^T \times e^{s\ln W}}\right) \\ &\qquad \left.\left. -\rho\left(\frac{Q^T \times (s'\ln W \cdot e^{s\ln W})}{Q^T \times e^{s\ln W}}\right)^2\right)\right)^T\end{aligned} \tag{1.21}$$

on s'ha fet servir que

$$s' = -\frac{1}{(1+\rho)^2}, \qquad s'' = \frac{2}{(1+\rho)^3}. \tag{1.22}$$

Els càlculs d'aquestes derivades s'han implementat tant en la versió escalar com en la matricial, aprofitant l'estructura comuna de les fórmules i reutilitzant gran part del codi de la computació de $E_0(\rho)$. Tot i que l'ús pràctic d'aquestes derivades s'abordarà en seccions posteriors, ja en aquesta fase s'ha volgut analitzar l'impacte computacional del seu càlcul. Per això, s'ha repetit l'experiment de

mesura de temps d'execució amb quatre configuracions: càlcul de $E_0(\rho)$ i càlcul de $F(\rho)$, $\frac{\mathrm{d}}{\mathrm{d}\rho}F(\rho)$, $\frac{\mathrm{d}^2}{\mathrm{d}\rho^2}F(\rho)$, cadascun implementat mitjançant bucles i operacions matricials.



(a) Respecte la modulació, M

(b) Respecte la quadratura, N

Figura 1.2: Temps d'execució de E_0 i F_0 més les seves derivades amb bucles i matrius

Tal com s'hi observa a la Figura 1.2, el càlcul de les derivades presenta un cost computacional molt similar al càlcul de $E_0(\rho)$, tant en la versió iterativa com en la matricial. Això és degut a l'elevat grau de reutilització d'operacions comunes, especialment en la versió matricial. Aquesta eficiència justifica la incorporació dels càlculs diferencials com a pas previ a l'optimització de ρ, sense que suposin una penalització significativa en el rendiment.

Cal destacar que, per a valors petits tant de M com de N, es produeix una aparent anomalia en els temps mesurats. Aquesta es deu al cost inicial associat a la inicialització de paràmetres i generació de matrius, que provoca una mena d'*overflow* computacional en la fase inicial. Això fa que el temps d'execució no es mantingui constant fins que ambdós valors són prou grans.

1.4 Mètodes d'optimització

En aquesta secció es presenten diversos mètodes d'optimització aplicats al càlcul de l'expressió $E(R) = \max_\rho\{E_0(\rho) - \rho R\}$. L'objectiu és trobar el valor òptim de ρ que maximitza aquesta expressió.

S'exploren diferents estratègies. Es comença amb enfocaments inicials, com la paral·lelització del càlcul, i es continua amb mètodes més sofisticats com el *Gradient Ascent*, el mètode de Newton [6, 5], i la interpolació d'Hermite [6]. També es proposen modificacions pràctiques per reduir el cost computacional.

Cada model es presenta amb la seva justificació, implementació i comparativa de resultats.

1.4.1 Intent inicial: paral·lelització

En un primer intent per optimitzar el càlcul de $E_0(\rho)$, es va explorar la possibilitat de paral·lelitzar els bucles mitjançant la construcció `parfor` de MATLAB. Aquesta eina forma part de la *Parallel Computing Toolbox* i permet executar iteracions d'un bucle en paral·lel, repartint la càrrega de treball entre diversos nuclis del processador o fins i tot entre diferents treballadors (workers) en una xarxa distribuïda.

En concret, es va emprar `parfor` per calcular en paral·lel els sumatoris iteratius que implicaven $E_0(\rho)$. Tot i que teòricament aquesta estratègia havia de reduir significativament el temps de càlcul, en la pràctica no va produir cap millora apreciable en el rendiment respecte a un bucle seqüencial convencional (`for`). Una possible explicació d'aquest comportament és que, per a cada iteració, les funcions implicades inicialitzen estructures de dades complexes que requereixen operacions costoses. Aquest cost d'inicialització és elevat i es replica en cada treballador, fet que pot anul·lar qualsevol benefici de la paral·lelització.

A més, la sobrecàrrega de gestionar els treballadors paral·lels, especialment per a iteracions curtes o lleugerament costoses, sovint fa que l'execució paral·lela sigui menys eficient que la versió seqüencial.

Això va portar a descartar l'ús de `parfor` en favor d'altres enfocaments més estructurats i controlats per a la cerca del valor òptim de ρ, com l'optimització numèrica directa.

1.4.2 Optimització mitjançant *Gradient Ascent* i Newton

A continuació, es va implementar un mètode d'optimització basat en ***Gradient Ascent***, aprofitant les derivades calculades a la secció anterior.

Aquest mètode consisteix a actualitzar iterativament el valor de ρ en la direcció del gradient de la funció a optimitzar $E(R)$ definida a 1.5. En cada iteració, s'aplica la següent actualització:

$$\rho_{n+1} = \rho_n + \alpha \cdot \frac{\mathrm{d}}{\mathrm{d}\rho} E(R)\bigg|_{\rho_n} = \rho_n + \alpha \cdot \frac{\mathrm{d}}{\mathrm{d}\rho} E_0(\rho)\bigg|_{\rho_n} - R. \tag{1.23}$$

L'objectiu és acostar-se al punt on la derivada de $E(R)$ es nul·la, per trobar el punt màxim de la funció, requerit per calcular la probabilitat d'error definida a 1.4. El valor de α controla la magnitud del pas: valors massa grans poden fer que el mètode divergeixi, mentre que valors massa petits poden fer-lo massa lent.

També, per la seva implementació, vam especificar un llindar de convergència (`threshold`), pel cas de voler optar per resultats més o menys precisos. A més, també es va afegir un nombre màxim d'iteracions per evitar bucles infinits.

Per una altra banda, també es va provar d'implementar el **mètode de Newton**. Aquest mètode consisteix a utilitzar tant la primera com la segona derivada de $E(R)$ per determinar el punt òptim amb més precisió i eficiència. La regla d'actualització de es:

$$\rho_{n+1} = \rho_n - \left.\frac{\frac{\mathrm{d}}{\mathrm{d}\rho}E(R)}{\frac{\mathrm{d}^2}{\mathrm{d}\rho^2}E(R)}\right|_{\rho_n} = \rho_n - \left.\frac{\frac{\mathrm{d}}{\mathrm{d}\rho}E_0(\rho) - R}{\frac{\mathrm{d}^2}{\mathrm{d}\rho^2}E_0(\rho)}\right|_{\rho_n}. \tag{1.24}$$

A diferència del mètode de *Gradient Ascent*, que utilitza únicament la primera derivada per actualitzar el valor de ρ en la direcció del pendent, el mètode de Newton també incorpora la segona derivada de la funció a optimitzar, $E(R)$. Aquesta segona derivada proporciona informació sobre la curvatura local, i permet ajustar la mida i direcció del pas d'actualització de forma més precisa. Això fa que el mètode no només avanci cap al màxim, sinó que ho faci de manera adaptativa segons la forma de la funció.

Com a conseqüència, aquest mètode sol convergir en menys iteracions, especialment quan la funció a optimitzar és suau i la derivada segona és definida i no s'anul·la en l'entorn del màxim. Aquesta propietat permet al mètode aprofitar millor la informació local per trobar ràpidament el punt òptim.

També cal mencionar que per a la seva implementació es va especificar un llindar de convergència i un nombre màxim d'iteracions per les mateixes raons que el *Gradient Ascent*.

En la pràctica, es va optar per utilitzar Newton pel fet que va mostrar millors resultats en termes d'eficiència i velocitat de convergència. A la Figura 1.3 es pot observar la diferència de cost computacional respecte al temps i les iteracions fetes entre aquests dos mètodes, usant en ambdós casos el mateix llindar i nombre màxim d'iteracions.

Observem a la Figura 1.3 que els temps d'execució per Newton són menors que en cap altre cas, la qual cosa ens indica que necessita menys iteracions per convergir. D'altra banda, tenim que el *Gradient Ascent*, independentment del pas escollit com a α, no arriba a reduir la velocitat com amb Newton.

1.4.3 Inicialització de ρ mitjançant interpolació d'Hermite

La interpolació d'Hermite és una tècnica que permet construir un polinomi que no només passa per certs punts coneguts, sinó que també respecta les derivades

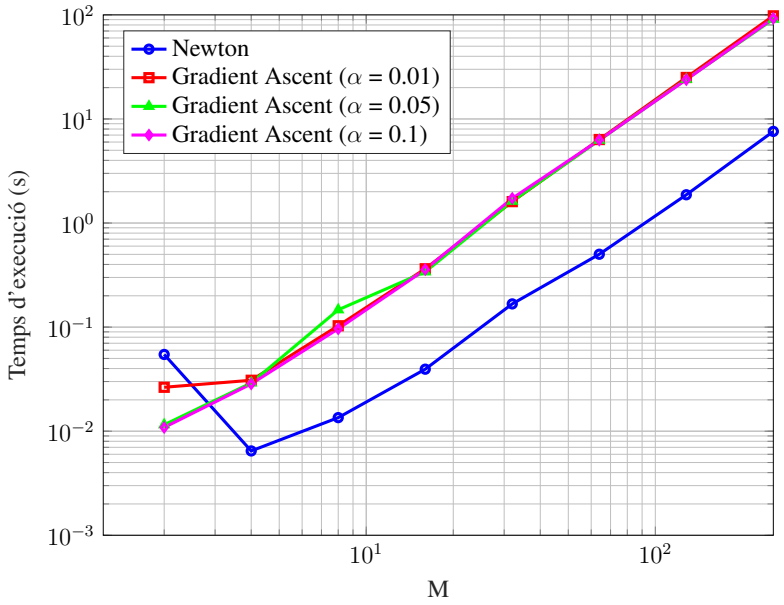


Figura 1.3: Comparació del temps d'execució i nombre entre mètodes d'optimització segons el valor de la modulació. Tots per un màxim de 100 iteracions i un llindar de 10^{-10}

d'aquests punts. En concret, donats dos punts x_0 i x_1, amb els valors de la funció $f(x_0), f(x_1)$ i les seves derivades $f'(x_0), f'(x_1)$, es pot construir un polinomi de grau 3 que satisfà:

$$\begin{aligned} P(x_0) &= f(x_0), \qquad & P'(x_0) &= f'(x_0), \\ P(x_1) &= f(x_1), \qquad & P'(x_1) &= f'(x_1). \end{aligned} \tag{1.25}$$

Això permet obtenir una aproximació suau i coherent amb la forma local de la funció original.

En el nostre cas, es va aplicar aquesta tècnica per aproximar $E(R) = E_0(\rho) - \rho R$, amb el proposit d'obtenir una estimació inicial de ρ propera al màxim d'aquesta funció, la qual després s'optimitza amb el mètode de Newton.

Els punts escollits per a la interpolació van ser

$$f(0) = E_0(0) - 0 \cdot R = 0 \tag{1.26}$$

$$f(1) = E_0(1) - 1 \cdot R \tag{1.27}$$

$$f'(0) = \frac{\mathrm{d}}{\mathrm{d}\rho} E_0(\rho)\bigg|_{\rho=0} - R \tag{1.28}$$

$$f'(1) = \frac{\mathrm{d}}{\mathrm{d}\rho} E_0(\rho)\bigg|_{\rho=1} - R, \tag{1.29}$$

on s'ha fet servir que $E_0(0) = 0$ per definició.

Trobem que $E_0(1)$ es pot calcular fàcilment de manera analítica

$$E_0(1) = -\log_2 \left(\sum_{x,\bar{x}} Q(x)\, Q(\bar{x})\, e^{-\frac{\mathrm{SNR}}{4}|x-\bar{x}|^2} \right). \tag{1.30}$$

Amb aquests quatre valors es construeix un polinomi d'Hermite $P'(\rho)$ de grau 3 que aproxima $E(R)$

$$\begin{aligned} P(\rho) = {} & f(0) + f'(0)\rho + 3(f(1) - f(0))\rho^2 \\ & + [2(f(0) - f(1)) + f'(1) + f'(0)]\, \rho^2(\rho - 1), \end{aligned} \tag{1.31}$$

i la seva derivada corresponent

$$\begin{aligned} P'(\rho) = {} & f'(0) + 6(f(1) - f(0))\rho \\ & + [2(f(0) - f(1)) + f'(1) + f'(0)]\, (3\rho^2 - 2\rho). \end{aligned} \tag{1.32}$$

Un cop obtinguda l'expressió analítica de $P'(\rho)$, es pot identificar com un polinomi de segon grau en la forma general:

$$P'(\rho) = a\rho^2 + b\rho + c, \tag{1.33}$$

on els coeficients a, b i c es calculen a partir dels valors de la funció i les seves derivades:

$$\begin{aligned} a &= 3(f'(1) + f'(0) - 2f(1)), \\ b &= 2(3f(1) - 2f'(0) - f'(1)), \\ c &= f'(0). \end{aligned} \tag{1.34}$$

Resolent l'equació $P'(\rho) = 0$ es poden obtenir dos punts crítics: un màxim i un mínim. En aquest context, s'escull la solució que correspon al màxim local, que és:

$$\rho = \frac{-b - \sqrt{b^2 - 4ac}}{2a}. \tag{1.35}$$

Aquesta estimació serveix com a punt de partida per la inicialització de qualsevol mètode en la maximització de $E(R)$. Utilitzar aquesta aproximació resulta molt eficient, ja que redueix considerablement el nombre d'iteracions necessàries per a la convergència. A la Figura 1.4 es pot observar la diferència entre el polinomi estimat Hermite, i la funció a optimitzar.

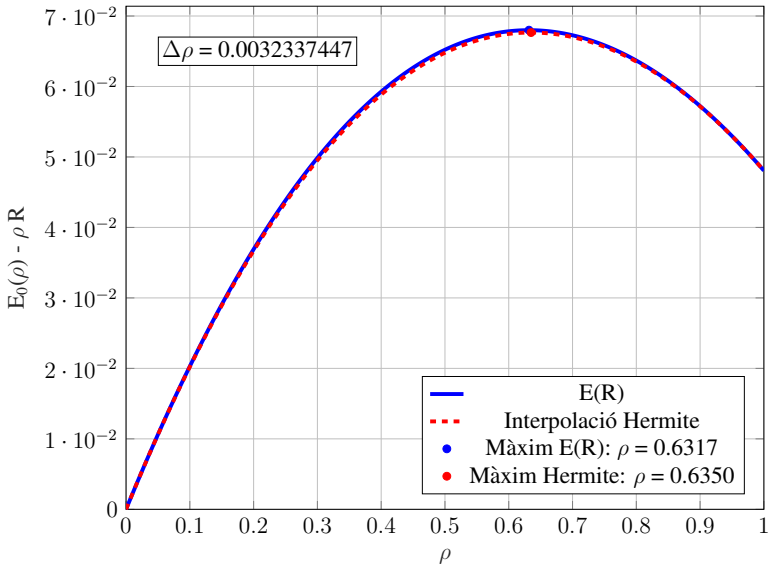


Figura 1.4: Comparació entre el valor inicial estimat per Hermite i el valor òptim trobat amb Newton

1.4.4 Modificació del mètode de Newton

Per reduir el cost computacional de cada iteració, es va considerar una modificació del mètode de Newton consistent a fixar la segona derivada a un valor constant aproximat en lloc de calcular-la dinàmicament a cada pas. Aquesta simplificació redueix significativament el temps d'execució, ja que evita el càlcul de derivades més costoses en temps.

Tot i que aquesta aproximació pot provocar un lleuger increment en el nombre total d'iteracions, el seu impacte es veu fortament mitigat pel fet que la inicialització amb la interpolació d'Hermite proporciona una estimació molt propera al valor òptim de ρ. Com que el punt de partida és ja molt proper al màxim, l'ús d'una segona derivada fixa continua donant informació útil sobre la direcció i magnitud del pas, mantenint així una bona eficiència i estabilitat del procés iteratiu.

1.5 Implementació en MATLAB

El desenvolupament computacional del projecte s'ha dut a terme majoritàriament en MATLAB, aprofitant la seva eficiència en càlcul numèric i la seva capacitat per gestionar tant operacions vectoritzades com estructures iteratives. El codi s'ha dissenyat de manera modular, i es divideix en funcions específiques que permeten tant la reutilització com l'adaptació a diferents entorns de càlcul segons els

recursos disponibles.

1.5.1 Estructura general i funció principal

La funció central de la implementació és `exponents.m`, que calcula la funció $E(R) = \max_{\rho \in [0,1]} \{E_0(\rho) - \rho R\}$, on $E_0(\rho)$ és l'exponent d'error i ρ és el paràmetre d'optimització. Aquesta funció rep com a entrada:

- `SNR`: relació senyal-soroll en escala lineal,
- `R`: taxa objectiu (bits/ús de canal),
- `M`: ordre de modulació,
- `threshold`: criteri de convergència pel mètode de Newton.

A partir d'aquests valors, `exponents.m` troba el valor òptim de ρ que maximitza $E(R)$.

Cal comentar que, amb relació a la gestió de memòria i flexibilitat a l'hora de fer certes operacions costoses en espai, la funció mateixa administra de manera automàtica la selecció entre càlcul vectoritzat (amb matrius) i càlcul iteratiu (amb bucles `for`) en funció de la capacitat de memòria del sistema usat, per tal de garantir la màxima compatibilitat. MATLAB tendeix a reservar blocs grans quan es treballa amb operacions matricials, cosa que pot provocar problemes en sistemes amb poca RAM. Per evitar això, el codi comprova la mida dels paràmetres i opta per una implementació iterativa quan es preveu que el càlcul matricial podria excedir la memòria disponible. Aquesta dualitat entre càlcul vectoritzat i iteratiu assegura una major portabilitat i estabilitat del codi.

1.5.2 Funcions auxiliars

El codi inclou diverses funcions auxiliars que es poden classificar segons la seva funcionalitat:

- **Càlcul d'exponents:**
  - `E0_matrix.m`, `F0_matrix_fixed.m`: càlculs vectoritzats utilitzant matrius.
  - `E0_fors_v2.m`, `F0_fors.m`: càlculs iteratius amb bucles `for`.
- **Interpolació i optimització:**
  - `interpolacio.m`: interpolació de la funció $E(R)$ usant Hermite.

  - `newton_method_v2.m`: implementació del mètode de Newton per maximitzar $E(R)$.

- **Generació de dades:**

  - `PAM_generator.m`: generació de modulacions `M-PAM` amb símbols iid.
  - `matrix_generator.m`: generació de les matrius `Q`, `G` i π.
  - `GaussHermite_Locations_Weights.m`: generació dels pesos i nodes de la quadratura.

Per a una visió detallada de totes les funcions i lògica implementada, es pot consultar el codi complet inclòs a l'Apèndix A.

1.5.3 Base de dades i optimització del càlcul

Amb l'objectiu de reduir el temps de càlcul i evitar repetir innecessàriament operacions, s'ha incorporat una consulta a una base de dades en línia des de dins de la funció principal del codi, `exponents.m`. Concretament, s'utilitza el servei DynamoDB d'Amazon Web Services (AWS) per emmagatzemar i recuperar resultats ja calculats prèviament.

La consulta es realitza mitjançant la funció `getAwsData.m`, que, donats uns valors d'entrada M, N, SNR i R, comprova si els resultats corresponents ja existeixen a la base de dades. En cas afirmatiu, es reutilitzen directament; en cas contrari, es realitza el càlcul localment. La funció `getAwsData.m` actua com a pont entre MATLAB i un script Python. Genera i executa una comanda de terminal amb la següent estructura:

```
python get_data.py --table exponents --constellationM
M --nodesN N --SNR SNR --transmissionRate R
```

Aquesta comanda executa l'script `get_data.py`, que utilitza la biblioteca `boto3` per accedir a DynamoDB. L'script aplica filtres sobre els atributs corresponents:

- `constellationM`
- `nodesN`
- `signalNoiseRatio`
- `transmissionRate`

A continuació, escaneja la base de dades i retorna els resultats en format JSON, que són posteriorment analitzats a MATLAB mitjançant `jsondecode`.

```
data = getAwsData(M, N, SNR, R);
disp(data.status);

if data.status == "success"
    disp('Data recovered from AWS.');

    E = data.items.errorExponent;
    rho = data.items.optimalRho;
```

Listing 1.1: Exemple de codi per recuperar dades d'AWS

Aquest enfocament ofereix una manera senzilla però efectiva de reaprofitar resultats anteriors, estalviant temps i recursos quan s'exploren conjunts de paràmetres grans o es repeteixen simulacions.

1.5.4 Estimació del temps d'execució

Amb l'objectiu d'informar l'usuari del temps aproximat necessari per completar el càlcul de $E(R)$, es va implementar una estimació empírica basada en mesures prèvies, recollides en un full de càlcul (vegeu l'Apèndix B). Aquesta estimació és especialment útil en simulacions massives o execucions remotes.

El model parteix de la normalització del temps consumit per cada càlcul en funció del terme $M^2 \cdot N^2$, on M és l'ordre de modulació i N, el nombre de punts en la quadratura de Gauss-Hermite. A partir de diverses mesures, es va obtenir una constant mitjana de proporcionalitat, anomenada *relació de temps normalitzada*, amb valor:

$$\textit{relació de temps normalitzada} = 1{,}81 \cdot 10^{-6} \text{ ms}.$$

Així, el temps estimat es calcula segons:

$$\textit{temps estimat} \approx 1{,}81 \cdot 10^{-6} \cdot M^2 \cdot N^2 \text{ ms} \tag{1.36}$$

Tot i que aquest model proporciona una referència orientativa, la seva precisió és força limitada. L'estimació es basa en una mitjana global molt generalitzada i no contempla amb detall la variabilitat real segons els paràmetres específics. En particular, per a valors d'entrada molt petits o molt grans, la desviació respecte al temps real pot ser considerable. A més, les diferències en el maquinari utilitzat (capacitat de càlcul, arquitectura del processador i quantitat de memòria disponible) poden afectar significativament la durada real del càlcul. Per aquest motiu, l'estimació ha de ser interpretada com una aproximació molt genèrica, útil només com a referència inicial.

Cal destacar que aquesta estimació assumeix l'ús de la versió matricial del càlcul. En sistemes amb restriccions de memòria, on el codi opta per la versió iterativa (`for-loops`), el temps real d'execució pot diferir significativament de l'estimat. En aquests casos, el codi emet un missatge d'avís informant l'usuari que l'execució podria trigar més del previst.

1.5.5 Flux general d'execució

Per tal de sintetitzar el comportament del programa, la Figura 1.5 mostra el flux general d'execució del càlcul.

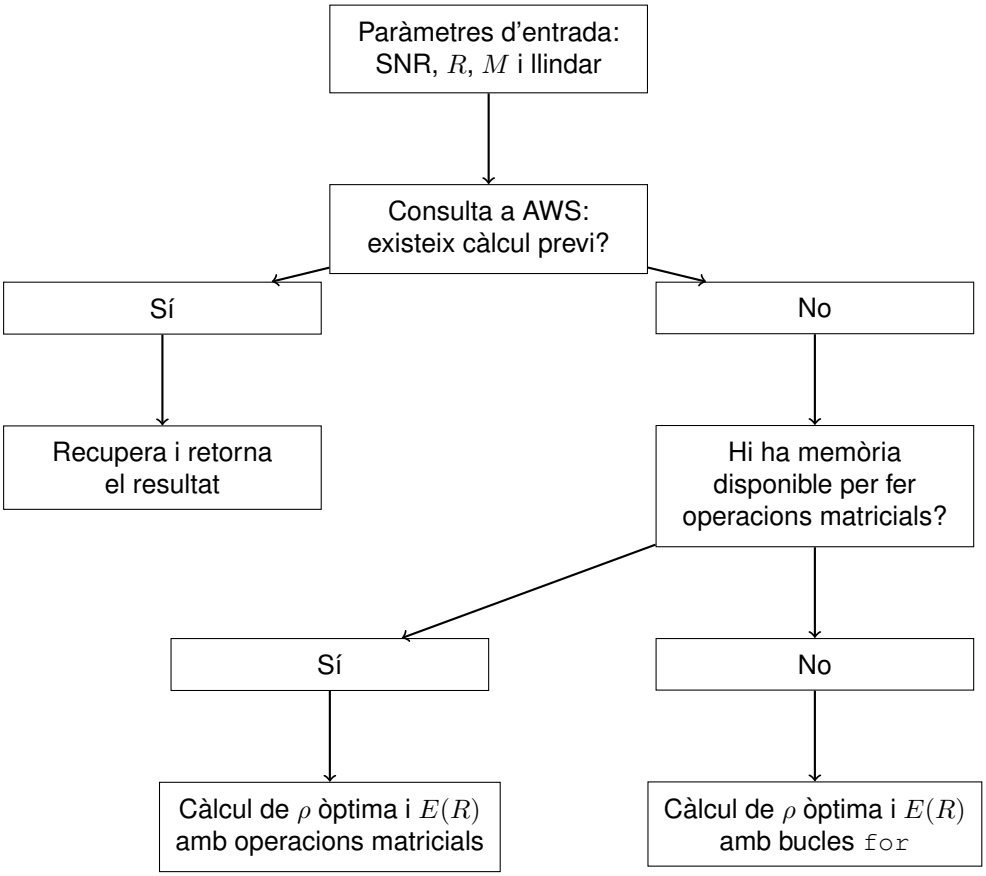


Figura 1.5: Flux simplificat de l'execució del càlcul en MATLAB

1.6 Resultats i comparatives de temps d'execució

Durant el desenvolupament d'aquest projecte es va dur a terme una anàlisi extensiva dels temps d'execució associats a cada mètode de càlcul i optimització.

Per tal de garantir una avaluació objectiva, es van registrar totes les proves en un full de càlcul compartit, anotant les condicions de cada execució: hardware, sistema operatiu, llenguatge de programació, paràmetres de modulació i configuració computacional.

Els resultats es van recollir en dues taules:

- Una primera taula amb temps de càlcul de $E_0(\rho)$ segons el mètode de càlcul (bucles vs operacions matricials), per diferents valors de M i nodes de quadratura N. També especificant l'ús o no de la paral·lelització.
- Una segona taula amb comparatives de mètodes d'optimització, indicant el nombre d'iteracions, tolerància utilitzada i temps total consumit.

M	Llenguatge	Estructura	Paral·lelitzaió	Temps (ms)
128	C++	Bucles	OMP	19
128	C++	Matrius	-	1.337
128	MATLAB	Bucles	Parallel Computing Toolbox	66
128	MATLAB	Matrius	Parallel Computing Toolbox	32
128	MATLAB	Bucles	-	18.0005
128	MATLAB	Matrius	-	2.4231
1024	MATLAB	Bucles	-	1036.77
1024	MATLAB	Matrius	-	436

Taula 1.1: Comparativa de temps de càlcul de $E_0(\rho)$ amb diferents estructures per $N = 2$, SNR $= 1$

Mètode	Tolerància	Iteracions	Temps (ms)	Observacions
Gradient Ascent	10^{-6}	43	39090.4	$\alpha = 0.5$, $\rho_0 = 0.5$
Newton	10^{-6}	10	11604.5	$\rho_0 = 0.5$
Newton	10^{-10}	3	8746.3	Interpolació Hermite

Taula 1.2: Comparativa de mètodes d'optimització per a $M = 256$, $N = 20$, SNR $= 1$, $R = 0.5$

Com es pot observar, l'ús d'operacions matricials redueix dràsticament el temps de càlcul, especialment per valors alts de M. A més, el mètode de Newton amb inicialització per interpolació d'Hermite ofereix una combinació òptima entre rapidesa i precisió, amb un nombre reduït d'iteracions.

Les dades completes dels experiments es poden consultar al full de càlcul inclòs en l'Apèndix B, on es recullen diversos registres amb diferents paràmetres i configuracions de prova.

1.7 Conclusió

Aquest capítol ha permès desenvolupar i validar un conjunt de tècniques eficients per al càlcul i optimització de l'exponent d'error $E(R)$. L'anàlisi numèrica ha demostrat que l'ús d'operacions matricials i mètodes com Newton, combinats amb una bona inicialització, permeten reduir significativament el cost computacional sense comprometre la precisió.

Aquests resultats no només mostren l'efectivitat de les millores proposades, sinó que també ofereixen una base sòlida per a la seva aplicació pràctica en simulacions més complexes o sistemes de comunicació amb altes exigències computacionals.

Capítol 2

Disseny i Implementació de l'API

La creació d'una interfície de programació d'aplicacions (API) pròpia sorgeix de la necessitat d'integrar diverses funcionalitats tècniques dins d'un entorn unificat i accessible. En lloc de dependre d'eines de tercers o biblioteques tancades, es va optar per construir un sistema personalitzat que permetés el control complet sobre el càlcul numèric, la generació de gràfics i la interacció amb l'usuari.

2.1 Limitacions amb MATLAB i migració a C++

Inicialment, es va plantejar implementar els càlculs numèrics utilitzant MATLAB, ja que el prototip original de les funcions d'optimització havia estat desenvolupat en aquest entorn. Durant el procés d'adaptació a contenidors, es va optar per fer servir Docker; no obstant això, integrar MATLAB dins del contenidor va resultar inviable a causa de les seves restriccions de llicència. Tot i intentar obtenir una clau d'accés vàlida a través de l'equip d'informàtica de la universitat, no es va aconseguir cap solució viable.

Davant aquesta limitació, es va decidir migrar el codi a C++, una alternativa més compatible amb Docker i lliure de restriccions. Aquesta tasca de traducció va ser realitzada per un company de l'equip de recerca i va permetre mantenir una gran similitud en els resultats, tal com mostra la Taula 1.1, amb uns temps d'execució similars als obtinguts amb MATLAB. Com a conseqüència, el projecte incorpora un `Makefile`, el directori `build/` i la biblioteca `Eigen`, elements dels quals es detallarà el funcionament en les seccions posteriors.

2.2 Estructura general del projecte

El projecte pràctic s'ha estructurat com una aplicació web basada en una arquitectura modular clara, dissenyada per garantir la separació de responsabilitats, la reu-

tilització del codi i la facilitat de desplegament. S'ha definit un arxiu `main.py`, encarregat d'enllaçar el *frontend* amb els diferents mòduls del *backend*.

Entre aquests mòduls destaca `exponents`, responsable del càlcul de les probabilitats d'error mitjançant els mètodes d'optimització presentats al capítol anterior. També s'inclou el mòdul `chatbot`, desenvolupat de forma aïllada per un altre company de l'equip de recerca, però integrat al sistema a través del *backend*.

A continuació, es descriu l'estructura general del projecte, des dels requisits fins a l'organització modular i les dependències externes.

2.2.1 Requisits funcionals i tècnics

Per assolir els objectius plantejats, es van establir una sèrie de requisits funcionals i tècnics que havien de complir-se durant el desenvolupament:

- Permetre l'accés a funcionalitats de càlcul numèric avançat a través de peticions HTTP.
- Incloure una interfície gràfica basada en web, amb capacitat per generar gràfics interactius a partir dels resultats obtinguts.
- Garantir la modularitat del codi mitjançant una organització clara entre el *backend* (càlcul d'exponents, *chatbot* i *endpoints*) i el *frontend*.
- Assegurar la portabilitat i la facilitat d'execució mitjançant Docker.
- Integrar funcionalitats de càlcul implementades en C++ a través de llibreries compilades.

2.2.2 Estructura de carpetes i arxius

La distribució de fitxers i carpetes del projecte respon a criteris de claredat i escalabilitat. L'estructura principal és la següent:

- `main.py`: nucli de l'API, on es defineixen els punts d'entrada i es configuren els recursos del b*backend*.
- `Dockerfile`: especifica la imatge de Docker necessària per executar l'API en un entorn aïllat.
- `requirements.txt`: llista de dependències de Python (com `FastAPI`, `numpy` o `openai`) que s'instal·len dins del contenidor Docker per al funcionament del *backend*.

- `Makefile`: defineix instruccions per compilar automàticament el codi C++ i generar la llibreria compartida `libfunctions.so`.
- `build/`: emmagatzema els fitxers binaris generats, incloent-hi la llibreria `libfunctions.so` compilada des del codi C++.
- `static/`: directori que conté tot el *frontend* de l'aplicació. Inclou els arxius HTML, CSS, així com els scripts responsables de generar les gràfiques interactives mitjançant la llibreria D3.js [10].
- `exponents/` i `chatbot/`: directoris que contenen els mòduls funcionals del *backend*, cadascun amb el seu propòsit.

Aquesta disposició facilita tant l'execució local com l'adaptació del projecte a contenidors, i permet mantenir una visió clara de la jerarquia funcional.

2.2.3 Esquema del funcionament del sistema

A continuació es presenta un diagrama esquemàtic que mostra la interacció entre els diferents components del sistema: el *frontend*, el *backend* i els mòduls funcionals, així com la seva encapsulació dins del contenidor Docker.

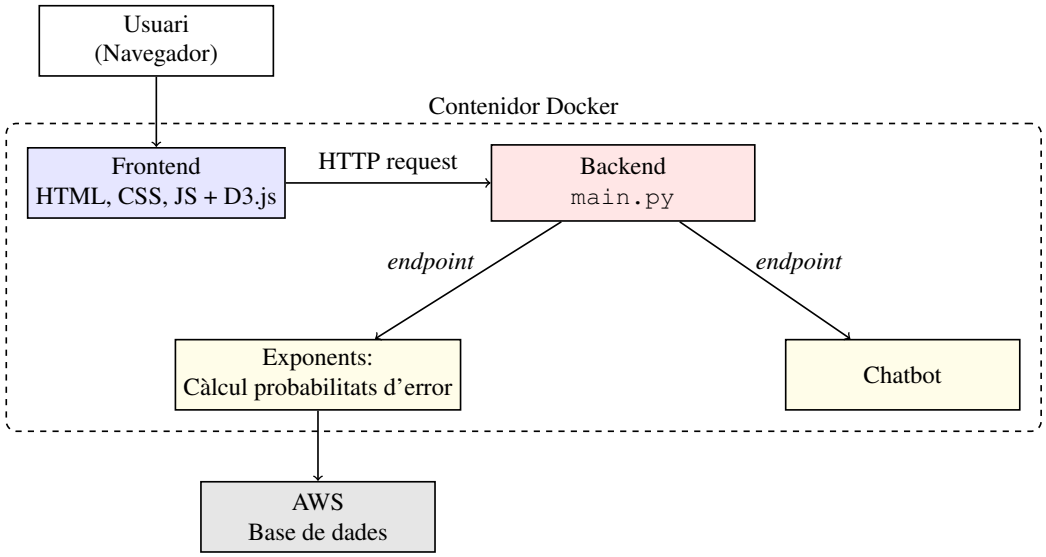


Figura 2.1: Esquema general de l'arquitectura de l'API dins un contenidor Docker.

2.2.4 Recursos externs

Per tal de millorar el rendiment global de l'aplicació i garantir l'eficiència tant en els càlculs matricials i numèrics com en la generació i visualització de gràfiques, el projecte fa ús de diversos recursos externs:

- `Eigen 3.4.0`: biblioteca de càlcul lineal en C++ que proporciona estructures i operadors per a treballar amb matrius i vectors, essencial per implementar eficientment les funcions de `exponents` [7].
- `D3.js`: biblioteca JavaScript utilitzada al *frontend* per construir gràfics SVG interactius. S'ha emprat per mostrar les corbes generades a partir dels resultats del *backend* amb funcionalitats avançades com el zoom, els *tooltips*, la selecció de punts o la superposició de corbes.
- Llibreries de Python:
  - `FastAPI`: *framework* lleuger i altament optimitzat per construir API RESTful. Permet gestionar peticions HTTP, definir rutes i estructurar l'aplicació *backend* [9].
  - `numpy`: emprat per a manipulacions numèriques bàsiques en Python, especialment quan es processen dades abans de retornar-les al *frontend* [8].
  - `ctypes`: llibreria en Python que permet interactuar amb codi C++ i amb llibreries compartides, imprescindible per a la comunicació entre el main.py i la llibreria compilada (`libfunctions.so`). Permet carregar funcions compilades i passar-hi paràmetres de manera segura [9].

L'elecció d'aquestes llibreries respon a criteris d'eficiència, compatibilitat i comunitat de suport. En conjunt, han permès mantenir un equilibri entre el rendiment computacional del *backend* i l'accessibilitat i interactivitat del *frontend*.

2.3 Implementació del `main.py` i els seus enllaços amb els mòduls

Tal com s'ha explicat anteriorment, l'API està estructurada en seccions, de manera que cada component implementa una funcionalitat específica del sistema. El fitxer `main.py` actua com a nucli de coordinació: defineix els *endpoints*, gestiona la comunicació entre el *frontend* i els mòduls del *backend*, i facilita l'accés a les funcionalitats implementades.

El mòdul `exponents` implementa els càlculs relacionats amb la probabilitat d'error i altres mètriques rellevants del sistema. D'altra banda, el mòdul `chatbot` incorpora funcionalitats conversacionals mitjançant models de llenguatge de gran escala (LLMs). Concretament, aquest mòdul rep consultes de l'usuari en llenguatge natural i genera respostes contextualitzades a través d'una interfície amb un model de LLM allotjat localment o accessible via servei extern.

2.3.1 Estructura i descripció dels *endpoints*

- `/exponents`: retorna la probabilitat d'error, l'exponent i el valor òptim de ρ a partir dels paràmetres d'entrada.
- `/plot_function`: genera les dades per construir una gràfica unidimensional variant una sola variable.
- `/plot_contour`: genera dades per construir una gràfica de contorn amb dues variables independents.
- `/chatbot`: rep una entrada textual i retorna una resposta generada pel model de llenguatge integrat.

`/exponents` Aquest *endpoint* rep paràmetres com el tipus de modulació, SNR, taxa de transmissió i criteri de llindar. Internament, aquests valors es passen a la funció `exponents`, la qual escriu els resultats en un *buffer* de sortida.

```
@app.get("/exponents")
async def exponents(M: float, typeM: str, SNR: float, R:
   float, N: float, n: float, th: float):
    result = (ctypes.c_float * 3)()
    lib.exponents(
        ctypes.c_float(M),
        typeM.encode('utf-8'),
        ctypes.c_float(SNR),
        ctypes.c_float(R),
        ctypes.c_float(N),
        ctypes.c_float(n),
        ctypes.c_float(th),
        result
    )
    values = list(result)
    return {
        "Probabilidad de error": values[0],
        "Exponents": values[1],
```

```
        "rho optima": values[2]
    }
}
```

Listing 2.1: Endpoint /exponents

`/plot_function` Aquest *endpoint* permet generar dades per construir corbes on s'estudia la variació d'una magnitud (com la probabilitat d'error, l'exponent o el valor òptim de ρ) respecte a un sol paràmetre (per exemple, R, M, SNR, etc.).

En primer lloc, l'*endpoint* rep una estructura de paràmetres condensada en la classe `FunctionPlotRequest`, definida amb `Pydantic`:

```
class FunctionPlotRequest(BaseModel):
    y: str
    x: str
    rang_x: list[float]
    points: int
    typeModulation: str
    M: float
    SNR: float
    Rate: float
    N: float
    n: float
    th: float
```

Listing 2.2: Classe FunctionPlotRequest

A l'interior de la funció, es genera un conjunt de punts distribuïts uniformement dins el rang especificat per a la variable independent especificada per l'usuari (`x`). Si aquesta variable és de tipus enter (com `M`, `N` o `n`), es fa una comprovació prèvia perquè els punts generats no siguin decimals.

```
@app.post("/plot_function")
async def generate_plot_from_function(plot_data:
   FunctionPlotRequest):
    try:
        if plot_data.x in ["M", "N", "n"]:
            raw = np.linspace(plot_data.rang_x[0],
               plot_data.rang_x[1], plot_data.points)
            x_vals = np.unique(np.round(raw).astype(int))
        else:
            x_vals = np.linspace(plot_data.rang_x[0],
               plot_data.rang_x[1], plot_data.points)
        ...
```

```
}
```

Listing 2.3: Endpoint /plot_function - Generació de punts

A continuació, per a cada punt, es construeix una crida a la funció `exponents`, i s'associa el resultat en funció del valor de `y` demanat. Finalment, es retorna la seqüència de valors `x` i els corresponents `y`.

```
@app.post("/plot_function")
async def generate_plot_from_function(plot_data:
   FunctionPlotRequest):
        ...
        y_vals = []

        for x_point in x_vals:
            args = {
                "M": plot_data.M,
                "SNR": plot_data.SNR,
                ...
            }
            args[plot_data.x] = x_point

            result = (ctypes.c_float * 3)()
            lib.exponents(
                ctypes.c_float(M),
                typeM.encode('utf-8'),
                ...
                result
            )

            y_map = {
                "ErrorProb": result[0],
                "Exponents": result[1],
                "Rho": result[2]
            }
            y_vals.append(y_map[plot_data.y])

        return {
            "x": x_vals.tolist(),
            "y": y_vals
        }
        ...
}
```

Listing 2.4: Endpoint /plot_function - Crida a exponents

Aquest *endpoint* és fonamental per alimentar la funcionalitat del *frontend*, que permet generar gràfiques interactives de manera dinàmica segons les seleccions de l'usuari.

`/plot_contour` Aquest *endpoint* és una extensió de `/plot_function`, amb la diferència que permet variar dues variables independents alhora. A partir d'aquestes dues dimensions, es construeix una matriu de valors de sortida per representar gràfiques de contorn o mapes de calor en dues dimensions.

Primer es rep la classe de paràmetres `ContourPlotRequest`, que defineix les dues variables independents, el rang i nombre de punts per a cadascuna, així com els valors fixos per la resta de paràmetres:

```
class ContourPlotRequest(BaseModel):
    y: str
    x1: str
    x2: str
    rang_x1: list[float]
    rang_x2: list[float]
    points1: int
    points2: int
    typeModulation: str
    M: float
    SNR: float
    Rate: float
    N: float
    n: float
    th: float
```

Listing 2.5: Classe ContourPlotRequest

En aquest *endpoint* es genera el conjunt de punts distribuïts uniformement de la mateixa manera que `/plot_function`, però ara per les dues variables independents. Seguidament es recorren totes les combinacions possibles de valors (`x1`, `x2`) i es calcula, mitjançant la funció `exponents`, el valor de `y` demanat. El resultat s'emmagatzema com a matriu (llista de llistes) per ser interpretat com una superfície:

```
async def generate_contour_plot(plot_data:
   ContourPlotRequest):
    ...
    z_matrix = []

    for val1 in x1_vals:
        row = []
```

```
        for val2 in x2_vals:
            args = {
                "M": plot_data.M,
                "SNR": plot_data.SNR,
                ...
            }
            args[plot_data.x1] = val1
            args[plot_data.x2] = val2

            result = (ctypes.c_float * 3)()
            lib.exponents(
                ctypes.c_float(M),
                typeM.encode('utf-8'),
                ...
                result
            )

            y_map = {
                "ErrorProb": result[0],
                "Exponents": result[1],
                "Rho": result[2]
            }
            row.append(y_map[plot_data.y])
        z_matrix.append(row)

    return {
        "x1": x1_vals.tolist(),
        "x2": x2_vals.tolist(),
        "z": z_matrix
    }
    ...
}
```

Listing 2.6: Endpoint /plot_contour - Crida a exponents i retorn de dades

Aquest *endpoint* és essencial per a visualitzacions en dues dimensions, com mapes de calor o corbes de nivell, i permet explorar com dues variables afecten simultàniament el comportament de les mètriques estudiades.

`/chatbot` Aquest *endpoint* permet la interacció amb un model conversacional. L'usuari envia una cadena de text (`message`) i l'API retorna una resposta generada pel mòdul `chatbot`.

Els paràmetres d'entrada estan definits en la classe `ChatbotRequest`:

```python
class ChatbotRequest(BaseModel):
    message: str
```

Listing 2.7: Classe ChatbotRequest

La implementació consisteix a cridar la funció `respond`, definida dins del mòdul `chatbot`. Si no hi ha cap error, es retorna la resposta en format JSON:

```python
@app.post("/chatbot")
async def chatbot_with_bot(request: ChatbotRequest):
    try:
        response = respond(request.message)
        return {"response": response}
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error
            in chatbot: {str(e)}")
```

Listing 2.8: Endpoint /chatbot

Vegeu Apèndix C per al codi complet.

2.4 Empaquetament del projecte amb Docker

Per garantir una execució homogènia en qualsevol entorn, es va optar per utilitzar Docker, una eina que encapsula l'aplicació i totes les seves dependències en un contenidor lleuger executat sobre el nucli de Linux.

2.4.1 Estructura i configuració del `Dockerfile`

El fitxer `Dockerfile` defineix pas a pas el procés de construcció del contenidor. L'objectiu és encapsular tant el *backend* (en Python i C++) com el *frontend* (en HTML i JavaScript) dins d'un entorn autosuficient i portàtil.

La imatge es basa en `python:3.11`, sobre la qual s'estableix un directori de treball (`/app`) i s'hi copien tots els fitxers del projecte. Tot seguit, es fa la instal·lació de les dependències de Python a partir de `requirements.txt`, així com les eines de compilació necessàries per construir la llibreria C++. Amb la crida a `make`, es genera la biblioteca compartida `libfunctions.so`, que després serà enllaçada al *backend* mitjançant `ctypes`. Finalment, l'última instrucció del `Dockerfile` indica com s'ha d'executar l'API un cop el contenidor estigui en marxa: es llança el servidor amb Uvicorn escoltant a la porta 8000.

```dockerfile
FROM python:3.11

```

```
WORKDIR /app

COPY . .

RUN pip install --no-cache-dir -r requirements.txt

RUN apt-get update && apt-get install -y g++ make

RUN make

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port",
    "8000"]
```

Listing 2.9: Dockerfile simplificat

Amb aquesta configuració, s'aconsegueix que l'aplicació quedi preparada per executar-se automàticament dins del contenidor, garantint una posada en marxa ràpida i sense errors de dependències.

Cal destacar que el `Dockerfile` presentat correspon a una versió del projecte en què el xatbot encara no ha estat integrat. La seva implementació, desenvolupada de manera paral·lela, requerirà previsiblement l'afegiment de noves dependències i instruccions al contenidor. Per tant, és probable que aquest fitxer evolucioni en futures iteracions del projecte per adaptar-se a aquesta funcionalitat i mantenir la coherència del sistema empaquetat.

2.4.2 Portabilitat i desplegament

Gràcies a Docker, l'API es pot desplegar fàcilment en qualsevol sistema que disposi del motor Docker, independentment del sistema operatiu o de la configuració prèvia de la màquina. El contenidor resultant encapsula tota l'aplicació, cosa que permet una execució immediata mitjançant les ordres següents:

```
docker build -t api-project .
docker run -p 8000:8000 api-project
```

Listing 2.10: Ordres per construir i executar el contenidor

Un cop executat, el servei queda disponible a través del navegador accedint a `http://localhost:8000`, o bé a través de l'adreça IP pública si es desplega en un servidor remot.

Aquest enfocament garanteix una elevada portabilitat, facilita el desplegament en entorns de producció o de recerca, i assegura que l'aplicació es comporti de manera consistent independentment de l'entorn d'execució.

A més, una forma senzilla i eficient de verificar el funcionament correcte de l'API és mitjançant la documentació automàtica que ofereix FastAPI, accessible a `http://localhost:8000/docs`. Aquesta interfície permet enviar peticions als diferents *endpoints* i visualitzar-ne les respostes de forma immediata, sense necessitat de desenvolupar primer el *frontend*. De fet, aquest mètode es va utilitzar durant les primeres etapes del desenvolupament per testejar i depurar el *backend* abans de procedir amb la seva integració completa amb la interfície web.

En conjunt, l'ús de Docker ha permès encapsular totes les dependències del projecte en un entorn controlat, millorant així la seva robustesa, sostenibilitat i escalabilitat.

Vegeu Apèndix C per al codi complet.

2.5 Desenvolupament del *frontend*

El *frontend* del projecte ha estat desenvolupat íntegrament amb HTML, CSS i JavaScript, i permet la interacció directa de l'usuari amb les funcionalitats de l'API. Aquesta capa és responsable de la recollida de paràmetres d'entrada, la visualització dels resultats i la generació de gràfics interactius mitjançant la llibreria D3.js.

2.5.1 Disseny de la interfície - HTML i CSS

La interfície web del projecte està definida a l'arxiu `index.html`, situat dins el directori `static/`, el qual és servit automàticament per `FastAPI` com a ruta principal. Aquest arxiu conté l'estructura bàsica de la pàgina i proporciona accés a totes les funcionalitats mitjançant formularis i seccions diferenciades.

El document es divideix principalment en tres blocs:

- **Càlcul d'exponents**: Permet introduir valors com la modulació (`M`), tipus de modulació, relació senyal-soroll (`SNR`), taxa de codi (`R`), entre altres. En prémer el botó `Compute`, es crida a `calculateExponents()`, una funció definida a `script.js`, que envia les dades al *backend* i mostra el resultat.

- **Generació de gràfiques**: Aquesta secció permet visualitzar resultats a cop de la generació automàtica de gràfiques a partir de la funció `exponents` del *backend*, amb la crida a la funció `plotFromFunction`, definida a `script.js`.

  L'usuari pot configurar diversos paràmetres de la visualització: el tipus de gràfica (lineal, log-log, etc.), l'estil i color de la línia, el nombre de punts i el rang de les variables. En el cas de gràfiques de tipus *contour*, es poden

definir dues variables independents per generar una representació bidimensional.

- **Chatbot**: Permet introduir consultes en llenguatge natural, que es processen amb la funció `sendMessage()` de `script.js`, la qual envia el text al *backend* i mostra les respostes en una caixa de missatges.

L'estructura de la interfície es fonamenta en una distribució mitjançant contenidors com `<section>` i `<aside>`, que organitzen la pàgina en tres blocs principals: configuració de paràmetres, àrea de gràfiques i assistent. Aquesta divisió facilita la navegació i la mantenibilitat del codi. Els formularis d'entrada es construeixen a través d'una jerarquia estructurada basada en les classes `input-group` i `input-group-block`, que agrupen i alineen els camps de manera coherent. Finalment, els estils visuals estan definits a `styles.css`, amb una estètica minimalista en tons neutres, que prioritza la llegibilitat, l'equilibri visual i l'adaptabilitat responsive.

La integració entre HTML, CSS i JavaScript permet una experiència d'usuari dinàmica: quan s'envia una petició, aquesta es processa mitjançant `script.js`, que actualitza els resultats i gràfics sense necessitat de recarregar la pàgina.

2.5.2 Implementació de `script.js` i comunicació amb els *endpoints*

L'arxiu `script.js` constitueix la lògica principal del *frontend* i controla la interacció entre l'usuari i el *backend*. Aquest fitxer s'encarrega de capturar els valors introduïts en els formularis HTML, validar-los, construir les peticions a l'API mitjançant `fetch()`, i finalment actualitzar la interfície amb els resultats.

```
function calculateExponents(event) {
    event.preventDefault();
    const M = document.getElementById('M').value;
    const typeM = document.getElementById('TypeModulation')
        .value;
    const SNR = document.getElementById('SNR').value;
    const R = document.getElementById('R').value;
    const N = document.getElementById('N').value || 20;
    const n = document.getElementById('n').value;
    const th = document.getElementById('th').value || 1e-6;
    const resultDiv = document.getElementById('result');
    ...
}
```

Listing 2.11: Exemple de recollida de paràmetres a calculateExponents()

A continuació s'expliquen les funcions principals que connecten la lògica del *frontend* amb els *endpoints* definits al *backend* (`main.py`). Aquestes funcions tenen un doble objectiu: d'una banda, gestionar les peticions a l'API (tant de tipus `GET` com `POST`) amb els paràmetres adequats; i de l'altra, interpretar les respostes i transmetre els resultats al sistema de visualització de gràfiques implementat en el mateix `script.js`.

1. Càlcul d'exponents: Quan l'usuari introdueix els valors dels paràmetres i prem el botó `Compute error`, s'executa la funció `calculateExponents()`. Aquesta recull els valors dels camps del formulari, construeix l'URL amb els paràmetres codificats, i envia una petició `GET` a l'*endpoint* `/exponents`. Si la resposta és correcta, es mostren els resultats a la secció corresponent; en cas d'error, es mostra un missatge d'advertència.

```
fetch(`/exponents?M=${M}&typeM=${typeM}&SNR=${SNR}&R=${R}&N
   =${N}&n=${n}&th=${th}`)
    .then(response => {
        if (!response.ok) {
            throw new Error("Server response not OK");
        }
        return response.json();
    })
    .then(data => {
        resultDiv.innerHTML = `
            <p><strong>Probability error:</strong> ${data["
               Probabilidad de error"].toFixed(4)}</p>
            <p><strong>Error exponent:</strong> ${data["
               Exponents"].toFixed(4)}</p>
            <p><strong>Optimal rho:</strong> ${data["rho
               optima"].toFixed(4)}</p>
        `;
        resultDiv.classList.add('show');
    })
```

Listing 2.12: Exemple de crida a /exponents

2. Gràfiques automàtiques: Pel cas de la graficació se segueix una estructura similar al punt anterior. La funció `plotFromFunction()` recull els paràmetres d'entrada i envia una petició `POST` a l'*endpoint* `/plot_function` o `/plot_contour`, segons el tipus de gràfica seleccionat.

En el cas de `/plot_function`, un cop es rep una resposta vàlida del `POST`, s'obtenen els punts computats del gràfic, que s'envien juntament amb el context dels paràmetres a la funció `drawInteractivePlot()`. Aquesta s'encarrega de representar la gràfica a la interfície i d'oferir informació contextual mitjançant eines interactives.

```
fetch('/plot_function', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(payload)
})
.then(response => {
    if (!response.ok) throw new Error("Error en /
       plot_function");
    return response.json();
})
.then(data => {
    const metadata = {N, th};
    drawInteractivePlot(data.x, data.y, {
        color: color,
        lineType: lineType,
        plotType: plotType,
        metadata
    });
});
```

Listing 2.13: Exemple de crida a /plot_function

Pel que fa a `/plot_contour`, el funcionament és anàleg: un cop rebudes les dades computades, es crida la funció `drawContourPlot()` amb els valors de `x1` i `x2`, que corresponen a les variables independents, i la matriu `zMatrix` resultant, utilitzada per generar la malla del gràfic.

3. Xatbot: La funció `sendMessage()` rep una cadena de text i envia una petició `POST` a l'*endpoint* `/chatbot`. La resposta s'afegeix com un nou missatge dins l'historial de conversa, gestionat per la interfície de xat.

Totes les funcions que contacten amb el *backend* implementen validacions locals abans d'enviar la petició. Si s'introdueix un rang invàlid (p. ex. `max < min`) o un camp queda en blanc, es mostra un missatge destacat en vermell i no s'envia cap petició.

```
for (const [key, value] of Object.entries(inputs)) {
  if (key !== x && (value === '' || isNaN(parseFloat(value)
     ))) {
      resultDiv.innerHTML = `<p style="color: red; font-
         weight: bold;">Please enter a valid value for ${
         key}.</p>`;
      resultDiv.classList.add('show');
      return;
  }
```

```
}
```

Listing 2.14: Exemple de gestió d'errors a /plot_function

També s'inclou la funció auxiliar `clearChat()`, que es limita a esborrar tot el contingut del xat. Aquesta acció es duu a terme assignant una cadena buida a l'element HTML amb identificador `chat-messages`:

```
function clearChat() {
    const chatBox = document.getElementById('chat-messages');
    chatBox.innerHTML = '';
}
```

Listing 2.15: Esborrat del xat

La funció es crida automàticament quan l'usuari fa clic al botó `Clear Chat`, situat a la part inferior de la secció de xat, dins de la barra lateral dreta de la interfície.

Finalment, el fitxer `script.js` inclou diverses funcions auxiliars per adaptar dinàmicament la visualització del formulari i millorar la interacció amb l'usuari. Per exemple, la funció `onLineTypeChange()` s'encarrega d'activar o desactivar certs camps del formulari en funció del tipus de gràfica seleccionat, com ara mostrar o amagar els camps corresponents a una segona variable independent en el cas de les gràfiques de tipus `contour`. També s'inclouen funcionalitats com `toggleLeftSidebar()` i `toggleRightSidebar()`, que permeten expandir o contraure les barres laterals de configuració i assistent, respectivament, millorant l'accessibilitat en pantalles petites. Aquest conjunt de funcions contribueix a una experiència d'usuari més fluida, clara i personalitzable.

2.5.3 Generació de gràfics interactius mitjançant la integració de D3.js

Per tal de generar i gestionar gràfics interactius a la interfície web, s'ha desenvolupat un script basat en la llibreria `D3.js`. Aquesta subsecció descriu detalladament totes les funcionalitats interactives del fitxer `script.js`.

Abans de descriure cada funcionalitat, cal entendre el procés d'inicialització del sistema quan es carrega la pàgina. El punt d'entrada principal és un gestor d'esdeveniments que escolta el moment en què el DOM (*Document Object Model*) està completament carregat. En aquest punt, s'executen una sèrie de funcions clau entre les quals trobem `initializeChart()`, on es construeix tota l'estructura del gràfic, `drawDefaultGrid()` que dibuixa una graella bàsica i `plotInitialGraph()`, la qual genera automàticament un gràfic inicial amb

valors predefinits per tal de mostrar una visualització significativa immediatament després de la càrrega de la pàgina. També s'invoca per defecte la funció `onLineTypeChange()`, per gestionar la visibilitat dels camps associats als modes `contour`, i la funció `plotInitialGraph()`,

```
window.addEventListener('DOMContentLoaded', () => {
    initializeChart();
    drawDefaultGrid();
    plotInitialGraph();
    onLineTypeChange();
});
```

Listing 2.16: Inicialització automàtica quan es carrega el DOM

Quan la pàgina carrega, s'executa la funció `initializeChart()`, que com s'ha mencionat, crea l'estructura bàsica del gràfic. Aquesta estructura inclou un contenidor `flex` amb dues columnes: una per al gràfic SVG, i una altra per a la llegenda. SVG és un format vectorial escalable que permet una renderització precisa dels elements gràfics a qualsevol mida. Aquesta funció també inicialitza variables globals, que es fan servir des de la resta del codi per mantenir referències persistents a escales, eixos i capes SVG.

```
const svg = container.append('svg')
  .attr('viewBox', `0 0 ${width} ${height}`);

window.__svg = svg;
window.__xScale = d3.scaleLinear().range([0, window.
    __innerWidth]);
window.__yScale = d3.scaleLinear().range([window.
    __innerHeight, 0]);
window.__gX = window.__g.append('g').attr('class','axis x-
    axis');
window.__gY = window.__g.append('g').attr('class','axis y-
    axis');
```

Listing 2.17: Inicialització del contenidor gràfic SVG i escales de les dades dins initializeChart()

En el nucli del sistema trobem la funció `renderAll()`, que s'encarrega de construir, actualitzar i redibuixar totes les gràfiques visibles segons l'estat actual d'`activePlots`, una llista que conté totes les representacions actives al gràfic. Aquesta funció s'executa automàticament cada vegada que s'afegeix, elimina o modifica una corba, garantint que la representació visual sigui coherent amb les dades. La funció efectua diverses operacions clau de manera seqüencial:

1. Si no hi ha cap gràfica activa, es netegen totes les capes (línies, punts, contours) i es dibuixa la graella per defecte amb `drawDefaultGrid()`.

2. Es detecta si algun dels eixos ha de ser logarítmic segons el tipus de gràfiques actives, i es redefineixen les escales `xScale` i `yScale` en conseqüència.

3. Es reinicialitza el zoom i s'actualitzen eixos i graella amb `zoomed()`.

4. Es dibuixen les gràfiques de línia. Per cada corba s'afegeix:

   - Un `path` SVG amb l'estil indicat (`solid`, `dashed`, etc.).
   - Un grup de punts `circle` amb interacció `mouseover`.

```
const lineGen = d3.line()
  .curve(d3.curveLinear)
  .x((_, i) => window.__xScale(d.x[i]))
  .y((_, i) => window.__yScale(d.y[i]));

g.select('path.line')
  .datum(d.y)
  .attr('fill', 'none')
  .attr('stroke', d.color)
  .attr('stroke-width', 2)
  .attr('stroke-dasharray', dashMap[d.dashStyle] || '')
  .attr('d', lineGen);
```

Listing 2.18: Generació línia amb SVG

5. En cas que el tipus de gràfica sigui *contour*, aquest es dibuixa i es gestiona la barra de colors:

```
for (let i = 0; i < p.x1.length; i++) {
  for (let j = 0; j < p.x2.length; j++) {
    g.append('rect')
      .attr('x', xs(p.x1[i]))
      .attr('y', ys(p.x2[j]))
      .attr('width', dx)
      .attr('height', dy)
      .attr('fill', colorScale(p.z[i][j]))
      .attr('opacity', 0.7);
  }
}
```

Listing 2.19: Construcció de la malla de contours

6. Si està activada l'opció `Show Points`, es mostren els valors numèrics de cada punt mitjançant petits textos al costat dels cercles. Aquesta funcionalitat s'activa automàticament dins `renderAll()` mitjançant la crida a `drawDotsIfEnabled()`, i facilita la lectura precisa de les dades mostrades.

7. Finalment, s'actualitzen les etiquetes dels eixos segons la darrera gràfica afegida, i es crida a `updatePlotListUI()` per mantenir la llegenda sincronitzada amb el conjunt de gràfiques actives.

Aquest procés garanteix que l'usuari pugui superposar múltiples corbes, afegir *contours*, fer zoom i rebre retroalimentació visual immediata en cada acció. `renderAll()` és el centre funcional de tot el sistema de visualització.

A més de la gestió i actualització dels gràfics ja existents, el sistema també incorpora mecanismes per controlar la generació de noves gràfiques de manera segura i coherent. Abans de generar-ne una de nova, es comprova si ja hi ha representacions actives. En cas afirmatiu, i si es detecta un canvi en variables clau com `Y` o `X1`, es mostra automàticament un avís a través d'una finestra modal. Aquest comportament, gestionat per la funció `showWarningIfOverwritingPlot()`, permet alertar l'usuari que les gràfiques existents se sobreescriuran, evitant així la pèrdua involuntària d'informació visual.

Per una altra banda, ens trobem amb la funció `drawInteractivePlot(x, y, opts)`. Aquesta funció s'utilitza per representar gràfiques de línia a partir de dades numèriques `X1` i `Y`. Accepta també un objecte `opts` que conté opcionalment el color, l'estil de línia (`solid`, `dashed`, etc.), el tipus d'escala (`linear`, `logX`, `logY`, `logLog`) i una etiqueta amb les metadades corresponents. Primer es genera un identificador únic per a la nova gràfica (`plotId`) i es construeix una etiqueta per defecte amb les dades de la configuració. Després es filtren punts no vàlids (com valors negatius o zero per escales logarítmiques). Finalment, les dades es guarden a `activePlots` amb els paràmetres associats i es crida a `renderAll()` i `updatePlotListUI()` per actualitzar la visualització i la llegenda.

```
activePlots.push({
  plotId,
  x: xCopy,
  y: yCopy,
  color,
  dashStyle,
  label,
  plotType,
  metadata: opts.metadata || {}
});
```

```
renderAll();
```

Listing 2.20: Inicialització d'una corba a drawInteractivePlot()

La funció `drawContourPlot(x1, x2, zMatrix)` s'utilitza per representar gràfiques de tipus *contour*, a partir de dues variables independents i una matriu de valors associats. Es genera automàticament una etiqueta per defecte i s'aplica un gradient de colors interpolat. Les dades es guarden dins la variable global `activePlots` com a objecte de tipus `contour`, i es crida a `renderAll()`.

```
activePlots.push({
  plotId,
  type: 'contour',
  x1, x2, z: zMatrix,
  label,
  color: gradient
});
renderAll();
```

Listing 2.21: Inicialització d'un contour a drawContourPlot()

Respecte al sistema de zoom, aquest es basa en `d3.zoom()` amb escales entre 1 i 10. Quan l'usuari fa *zoom* o desplaça el gràfic amb el ratolí, es dispara automàticament la funció `zoomed(event)`. El paràmetre `event` conté una transformació que indica quan s'ha fet zoom i quant s'ha mogut el gràfic en aquesta interacció, a través de `event.transform`. En total, la funció `zoomed()` actualitza les escales redimensionades, redibuixa els eixos amb format decimal o logarítmic, adapta la graella, i transforma visualment les capes de dades. El gruix de les línies i la mida dels punts també s'ajusten dinàmicament.

```
const t = event.transform;
const newX = t.rescaleX(window.__xScale);
const newY = t.rescaleY(window.__yScale);

window.__gX.call(d3.axisBottom(newX).ticks(6));
window.__gY.call(d3.axisLeft(newY).ticks(6));
```

Listing 2.22: Actualització dels eixos dins de la funció zoomed()

Per una altra banda, també s'ha definit la funció `resetZoom()`, la qual permet restaurar les escales originals segons les dades visibles. Reuneix totes les coordenades dels gràfics actius i reestableix el zoom amb transicions suaus. Aquesta funció es crida mitjançant el botó `Reset Zoom`, que es defineix a la funció `initializeChart()` i està situat sota el gràfic, dins la secció de controls visuals.

De manera complementària, la funció `changePlotScale(button)` permet modificar dinàmicament l'escala dels eixos del gràfic. Aquesta funcionalitat s'associa a un conjunt de botons visuals etiquetats com a `Linear`, `LogX`, `LogY` i `LogLog`, els quals alteren respectivament el comportament dels eixos horitzontal i vertical. Quan l'usuari fa clic en un d'aquests botons, la funció actualitza les escales globals `xScale` i `yScale`, redefineix els valors mínims i màxims segons les dades visibles, i invoca `renderAll()` per actualitzar immediatament la representació.

Tot seguit, també ens trobem amb la funció `updatePlotListUI()`. És l'encarregada de construir i actualitzar la llegenda que mostra totes les gràfiques actives al gràfic. Per a cada objecte de la llista global `activePlots`, aquesta funció genera una fila amb:

- Un petit quadrat de color representant la corba.
- Una etiqueta amb la descripció de la corba.
- Un botó `X` per eliminar la gràfica.

A més, afegeix comportament interactiu en passar el cursor per sobre de cada element, cridant a les funcions `highlightPlot()` i mostrant metadades contextuals en una caixa flotant.

```
const btn = document.createElement('button');
btn.textContent = 'X';
btn.onclick = () => removePlot(p.plotId);

item.addEventListener('mouseover', () => highlightPlot(p.
    plotId, true));
item.addEventListener('mouseout', () => highlightPlot(p.
    plotId, false));
```

Listing 2.23: Interaccions definides a updatePlotListUI()

La funció `removePlot(plotId)` s'encarrega d'eliminar una gràfica específica de la llista `activePlots`, actualitzar la llegenda i tornar a renderitzar tot el contingut gràfic. En cas que no resti cap gràfica activa després de l'eliminació, es crida automàticament a `drawDefaultGrid()` per restaurar una graella per defecte.

```
activePlots = activePlots.filter(p => p.plotId !== plotId);
updatePlotListUI();
renderAll();
if (activePlots.length === 0) drawDefaultGrid();
```

Listing 2.24: Eliminació de gràfiques a removePlot()

També, relacionat amb el contenidor de llegendes `activePlots`, trobem a funció `makePlotListDraggable` que permet que el panell amb la llista de gràfiques actives (identificat amb l'ID `plot-list`) es pugui arrossegar lliurement per la zona de la gràfica. El funcionament es basa a escoltar l'esdeveniment `pointerdown` sobre l'element. Quan es detecta un clic (sempre que no sigui sobre un `<input>`, `<button>` o `<textarea>`), es captura la posició relativa entre el cursor i la cantonada superior esquerra del panell, la qual es fa servir per moure'l adequadament quan el cursor es desplaça.

```
function makePlotListDraggable() {
  const el = document.getElementById('plot-list');
  if (!el) return;

  let isDragging = false;
  let offsetX, offsetY;

  el.addEventListener('pointerdown', (e) => {
    if (
      e.target.closest('input') ||
      e.target.closest('button') ||
      e.target.closest('textarea')
    ) return;

    isDragging = true;
    const containerRect = document.getElementById('plot-
       output').getBoundingClientRect();
    const elRect = el.getBoundingClientRect();
    offsetX = e.clientX - elRect.left;
    offsetY = e.clientY - elRect.top;

    el.setPointerCapture(e.pointerId);
  });
}
```

Listing 2.25: Inicialització de l'arrossegament del panell de gràfiques actives

Convé remarcar que totes les funcionalitats interactives descrites han estat possibles gràcies a l'ús intensiu de la llibreria `D3.js`, la qual ha facilitat la construcció i manipulació dinàmica de l'estructura SVG, la generació d'escales visuals (tant lineals com logarítmiques), l'actualització contínua dels eixos i la implementació del sistema de zoom i desplaçament mitjançant `d3.zoom()`. També s'ha fet ús de `d3.line()` per generar les línies interpolades, de cercles SVG per mostrar punts amb `tooltip`, i de `d3.interpolateTurbo` per a la coloració dels *contour plots*, entre altres funcionalitats. En conjunt, la llibreria ha estat fonamen-

tal per aconseguir una experiència visual dinàmica, rica i plenament integrada amb les dades generades.

Aquest conjunt funcional constitueix la base del sistema de visualització implementat. Vegeu Apèndix C per al codi complet.

Cal afegir, a més, que el desenvolupament d'aquesta part del projecte ha comptat amb l'ajuda constant de les eines ChatGPT i DeepSeek, les quals han estat essencials no només per resoldre errors puntuals o refinar el codi, sinó també per guiar-me en l'aprenentatge d'aquest nou entorn, especialment en l'ús de JavaScript i la llibreria `D3.js`, amb la qual no tenia cap experiència prèvia. Aquesta assistència m'ha permès incorporar funcionalitats interactives que, inicialment, m'haurien resultat molt difícils de desenvolupar de manera autònoma.

2.5.4 Resultat final de la interfície gràfica

La Figura 2.2 mostra el resultat final del desenvolupament de l'API. L'usuari pot calcular les mètriques d'error introduint els paràmetres rellevants a la secció *Configure Simulation* o mitjançant l'ús del xatbot. La generació de gràfics també es pot gestionar a través del xatbot o per via de la secció específica *Configure Plot*. La gràfica resultant es mostra dinàmicament a la secció central de la interfície junt amb la seva llegenda.

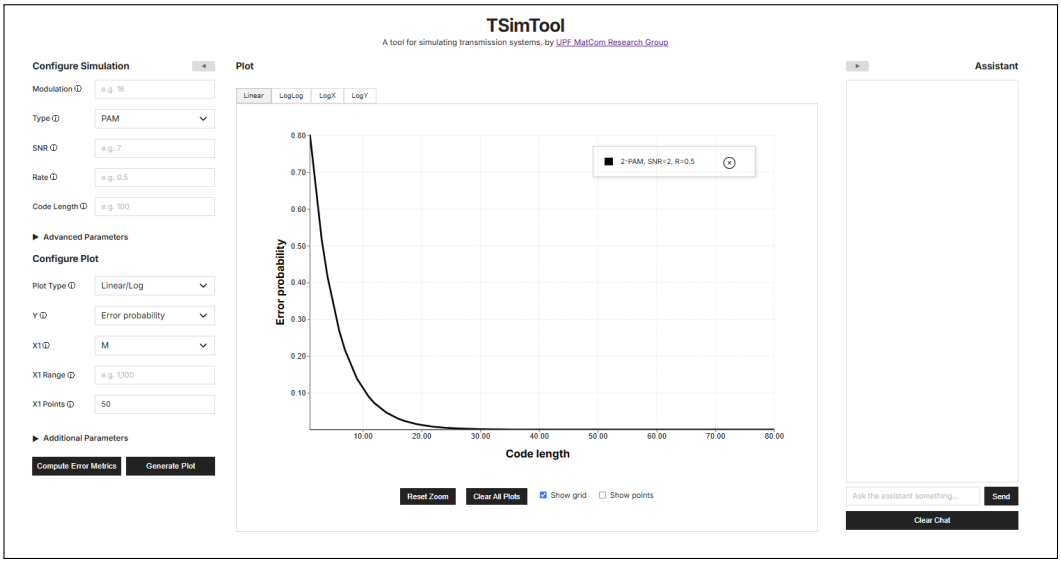


Figura 2.2: Captura de pantalla del *frontend* final de l'API gràfica interactiva.

2.6 Dificultats i decisions durant el desenvolupament

Durant el desenvolupament de la interfície web i la seva integració, es van presentar diversos reptes tècnics i de disseny que van requerir solucions creatives, adaptació constant i presa de decisions fonamentades sota limitacions reals.

Pel que fa a la construcció de la interfície, aquesta es va anar definint de manera incremental, afegint funcionalitats a mesura que es validaven resultats i es comprovava la viabilitat dels components. En tractar-se d'un entorn nou per a l'autora, van sorgir dificultats tant en l'estructuració del codi com en l'ús de biblioteques com `D3.js`. Per exemple, es van presentar problemes amb la gestió de gràfiques actives, la implementació de la interacció o coses tan simples com el redimensionat dels eixos per a representar dades en escala logarítmica. Aquestes qüestions es van anar resolent progressivament mitjançant proves, depuració i reorganització del codi.

Un cas particularment rellevant va ser la proposta inicial d'implementar un sistema de pestanyes per a visualitzar gràfiques de diferents tipus (per exemple, lineals i *contours*) en finestres separades. Tot i que es van estudiar diverses opcions per encapsular diferents visualitzacions en contenidors independents, aquesta funcionalitat no es va arribar a implementar, principalment per la seva complexitat estructural. Requeria reescriure part significativa de la lògica de renderització i comportava una reorganització profunda de l'estat global de la pàgina. La funció central `renderAll()`, responsable de dibuixar totes les gràfiques, ha estat dissenyada amb una visió unificada de la superfície de renderització. Permetre múltiples contenidors interactius en paral·lel, amb escalat independent, llegenda associada i zoom sincronitzat, hauria duplicat el volum de lògica necessària i afectat negativament la sostenibilitat del sistema.

D'altra banda, en fases més avançades del projecte es va identificar una limitació important: la mida final de la imatge Docker generada. Un cop construïda amb `make` i empaquetada amb totes les dependències (Python, biblioteques, binaris compilats i fitxers estàtics), però encara sense incloure el mòdul del xatbot, la imatge resultant ocupa aproximadament **1.69 GB**, una quantitat considerablement elevada per a una API d'aquestes característiques.

A l'anàlisi de la història de la imatge mitjançant `docker image history`, es pot veure que el bloc que contribueix més significativament a aquest pes és la instrucció següent:

```
1 RUN /bin/sh -c apt-get update && apt-get install -y build-
    essential python3-dev ...
```

Listing 2.26: Instal·lació de paquets base amb APT dins el Dockerfile

Aquest pas introdueix aproximadament 915 MB de dades, majoritàriament degudes a paquets de compilació i dependències associades com `g++`, `make`,

`cmake`, i capçaleres del sistema. Aquestes eines són necessàries en temps de construcció per compilar la llibreria `libfunctions.so`, però no són imprescindibles en temps d'execució.

Una possible millora en aquest aspecte seria la implementació d'un Docker multistage build, una tècnica que permet separar el procés de compilació del de distribució. Així, es pot utilitzar una imatge amb totes les eines per compilar el codi, però copiar només els binaris finals i els scripts necessaris a una segona imatge més lleugera, que serviria com a imatge final d'execució. Aquesta pràctica pot reduir dràsticament la mida total del contenidor, fent-lo més eficient i portable.

En resum, les dificultats experimentades durant la implementació han estat part essencial del procés d'aprenentatge i disseny. Algunes funcionalitats han estat descartades per motius justificats de complexitat, mentre que d'altres com l'optimització del pes del contenidor s'identifiquen com a millores pendents que podrien abordar-se en versions futures del projecte.

Conclusions i valoració personal

El desenvolupament d'aquest projecte ha suposat un repte significatiu en l'àmbit tècnic i personal. L'objectiu inicial d'integrar una llibreria C++ dins d'un entorn web interactiu no només ha requerit coneixements en programació, sinó també una comprensió profunda de sistemes com FastAPI, Docker, D3.js i la comunicació entre llenguatges.

Al llarg del procés, he viscut una gran varietat de moments: des de fases de fluïdesa i entusiasme, fins a etapes d'incertesa i frustració davant problemes tècnics inesperats. Aspectes com l'eliminació correcta de gràfiques o el maneig d'escales logarítmiques en el `frontend` m'han obligat a revisar, experimentar i reajustar el codi múltiples vegades. Tot i això, aquestes dificultats han estat oportunitats d'aprenentatge que m'han aportat solidesa i confiança com a desenvolupadora.

També he après la importància de la disciplina i la constància. Les reunions regulars amb els meus tutors m'han empès a mantenir un progrés sostingut i a treballar amb més rigor. Tot i que en alguns moments m'han semblat especialment exigents pel que fa a l'aparença i la coherència visual de la interfície, aquest grau de detall, que de vegades ha estat esgotador, ha contribuït de manera valuosa a millorar la qualitat del treball.

Pel que fa al resultat final, considero que el projecte compleix els objectius proposats i constitueix una base sòlida per a futures ampliacions. Ara bé, també hi ha espai de millora, especialment pel que fa a la interfície web. Un disseny més polit en l'àmbit visual i la implementació de pestanyes per separar gràfiques de diferents tipus són millores que podrien afegir valor tant a escala funcional com d'usabilitat. Una altra millora rellevant seria adaptar completament la interfície a qualsevol sistema operatiu o dispositiu, optimitzant l'experiència d'usuari en contextos diversos.

Pel que fa al codi, especialment a la gestió dels *endpoints*, també hi ha marge per a optimitzacions. Actualment, en alguns casos es realitzen múltiples crides redundants als mòduls C++ durant la generació de gràfiques, cosa que es podria millorar adaptant millor la lògica per a treballar amb estructures vectoritzades. Així mateix, un altre aspecte a reforçar seria l'estudi sistemàtic dels temps d'exe-

cució: calcular de manera més precisa la duració de cada càlcul i mostrar aquesta informació per pantalla. En aquest projecte, l'anàlisi de rendiment s'ha fet de manera empírica i puntual, però una avaluació quantitativa i automatitzada aportaria una visió més rigorosa del comportament del sistema.

Aquest projecte no només m'ha permès posar en pràctica diversos coneixements tècnics, sinó que ha representat una experiència d'aprenentatge completa. A banda de les competències adquirides en programació i desplegament d'aplicacions, també he desenvolupat habilitats d'organització, planificació i resolució de problemes en contextos reals. Deixo també una base de codi funcional i documentada que, esperem, pugui ser útil com a punt de partida per a futurs estudiants que vulguin continuar o expandir aquesta línia de treball.

Bibliografia

[1] R. G. Gallager, *Information Theory and Reliable Communication*. New York, NY, USA: John Wiley & Sons, 1968.

[2] J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. New York, NY, USA: McGraw-Hill, 2008.

[3] S. Haykin, *Communication Systems*, 4th ed. New York, NY, USA: John Wiley & Sons, 2001.

[4] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Hoboken, NJ, USA: Wiley-Interscience, 2006.

[5] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[6] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[7] G. Strang, *Computational Science and Engineering*. Wellesley, MA, USA: Wellesley-Cambridge Press, 2007.

[8] C. R. Harris *et al.*, "Array programming with NumPy,"*Nature*, vol. 585, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.

[9] G. van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA, USA: CreateSpace, 2009.

[10] M. Bostock, V. Ogievetsky, and J. Heer, "D3: Data-Driven Documents,"*IEEE Trans. Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, 2011, doi: 10.1109/TVCG.2011.185.

Apèndix A

Codi `MATLAB`

Aquest apèndix inclou un enllaç al codi font original utilitzat per implementar els càlculs per a l'optimització de $E(R)$, abans de la migració a `C++`.

Enllaç: Repositori `MATLAB` a GitHub

Apèndix B

Taules dels temps d'execució

Aquest apèndix inclou un enllaç al full de càlcul compartit on s'han recopilat els resultats dels experiments de temps d'execució, comparant les implementacions en `MATLAB` i `C++` per diversos paràmetres.

Enllaç: Full de càlcul compartit

Apèndix C

Codi API

Aquest apèndix inclou un enllaç al repositori de la `API`, amb tots els fitxers necessaris per al seu funcionament complet.

Enllaç: Repositori de l'API a GitHub