Nodi di Leja per Informatici

INTRODUZIONE: IL PROBLEMA DELL'INTERPOLAZIONE

Cosa stiamo risolvendo?

Immagina di avere una funzione complicata f(x) e di volerla approssimare con un polinomio p(x). L'**interpolazione polinomiale** è una tecnica che, dati n+1 punti $\{(x_0,f(x_0)), ..., (x_n,f(x_n))\}$, trova l'unico polinomio di grado ≤ n che passa esattamente per quei punti.

Il problema centrale: Dove metto i punti x_i?

Il Disastro dei Nodi Equispaziati

Se metti i nodi equispaziati x_0, x_1, ..., x_n su [-1,1], succede una cosa terribile chiamata **fenomeno di Runge**: aumentando il grado n, invece di migliorare l'approssimazione, l'errore **esplode** soprattutto ai bordi dell'intervallo.

Perché? L'interpolazione è intrinsecamente instabile, e la stabilità è misurata dalla **costante di Lebesgue** Λ_n. Per nodi equispaziati, Λ_n cresce come $2^n/n$ - una crescita **esponenziale**!

I NODI DI LEJA: L'IDEA GENIALE

L'Intuizione Matematica

I nodi di Leja sono una sequenza di punti scelti **greedily** per massimizzare la stabilità numerica. L'idea:

1. **Inizio**: Scelgo un punto qualsiasi (diciamo x_0)
2. **Passo ricorsivo**: Dato che ho già $\{x_0, x_1, ..., x_{s-1}\}$, scelgo il prossimo punto x_s che **massimizza** la produttoria:

   ```
   ∏ᵢ₌₀^(s-1) |x - xᵢ|
   ```

Perché questa produttoria? È collegata al determinante della matrice di Vandermonde, che a sua volta controlla la stabilità numerica dell'interpolazione.

La Connessione con Vandermonde

La matrice di Vandermonde ha questa forma:

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix}$$

Il suo determinante ha la proprietà **ricorsiva**:

$$\det(V_{0,\dots,s}) = \det(V_{0,\dots,s-1}) \cdot \prod_{i=0}^{s-1} (x_s - x_i)$$

Quindi **massimizzare la produttoria** ≡ **massimizzare il determinante** ≡ scegliere nodi che rendono il sistema più stabile!

IMPLEMENTAZIONE: DAL CONTINUO AL DISCRETO

Il Problema Computazionale

Trovare il massimo su tutto l'intervallo [-1,1] è computazionalmente costoso. Soluzione pragmatica: **discretizzazione**.

Creo una mesh fitta $X_M = \{x_1, x_2, \dots, x_M\}$ con $M \approx 10^4$ punti e cerco il massimo solo su questi punti candidati.

Due Algoritmi, Due Filosofie

ALGORITMO 1: DLP (Direct Product)

```
function dlp = DLP(x, d)
    dlp = zeros(1, d+1);
    dlp(1) = x(1);  % Primo nodo fisso

    for s = 2:d+1
        % Per ogni punto candidato, calcola la produttoria
        produttoria = prod(abs(x - dlp(1:s-1)), 2);

        % Scegli quello che massimizza
        [~, idx_max] = max(produttoria);
```

```
        dlp(s) = x(idx_max);
    end
end
```

Filosofia: Implementazione diretta dell'idea matematica.
Complessità: $O(M \cdot d^2)$ - per ogni nuovo nodo, valuto M produttorie di lunghezza crescente.

ALGORITMO 2: DLP2 (LU Factorization)

```
function dlp2 = DLP2(x, d)
    % Costruisci Vandermonde con base di Chebyshev
    V = cos(acos(x) * (0:d));

    % Fattorizzazione LU con pivoting
    [~, ~, P] = lu(V, 'vector');

    % I primi d+1 pivot sono i nodi di Leja!
    dlp2 = x(P(1:d+1))';
end
```

Filosofia: Sfrutta la teoria dell'algebra lineare. Il **pivoting** della fattorizzazione LU sceglie automaticamente le righe che massimizzano i sottodeterminanti - esattamente quello che vogliamo!

Complessità: $O(M \cdot d^2)$ per costruire V + $O(\min(M,d)^3)$ per LU.

Perché la Base di Chebyshev?

Invece dei monomi $\{1, x, x^2, \ldots\}$, uso i **polinomi di Chebyshev** $\{T_0(x), T_1(x), T_2(x), \ldots\}$ dove:

```
Tₖ(x) = cos(k · arccos(x))
```

Vantaggi cruciali:

1. **Stabilità**: $|T_k(x)| \leq 1$ per $x \in [-1,1]$
2. **Ortogonalità**: Miglior condizionamento della matrice
3. **Robustezza**: Niente overflow/underflow

Con i monomi, x^{50} può diventare astronomico ai bordi, rovinando la numerica.

LA COSTANTE DI LEBESGUE: IL TERMOMETRO DELLA STABILITÀ

Definizione Matematica

$$\Lambda_n = \max_{x\in[-1,1]} \sum_{i=0}^{n} |\ell_i(x)|$$

dove $\ell_i(x)$ sono i **polinomi di Lagrange**:

$$\ell_i(x) = \prod_{j\neq i} (x - x_j)/(x_i - x_j)$$

Interpretazione Fisica

La costante di Lebesgue misura quanto l'interpolazione può **amplificare** gli errori:

$$\|f - p_n\|_\infty \leq (1 + \Lambda_n) \cdot E_n(f)$$

dove $E_n(f)$ è il **miglior errore possibile** con un polinomio di grado n.

Metafora: Se $\Lambda_n = 100$, anche un errore minuscolo nei dati può essere amplificato di 100 volte nel risultato finale!

Implementazione Efficiente

```
function L = leb_con(z, x)
    n = length(z);
    lebesgue_vals = zeros(size(x));

    for i = 1:n
        % Calcola ℓᵢ(x) per tutti gli x contemporaneamente
        altri_nodi = [1:i-1, i+1:n];
        lagrange_poly = prod((x - z(altri_nodi)) ./ (z(i) - z(altri_nodi)), 2);

        % Accumula |ℓᵢ(x)|
        lebesgue_vals = lebesgue_vals + abs(lagrange_poly);
    end
```

```
    % Il massimo è la costante di Lebesgue
    L = max(lebesgue_vals);
end
```

Trucco di implementazione: Il `prod(..., 2)` calcola il prodotto **per righe**, permettendo di valutare $\ell_i(x)$ su tutti gli x contemporaneamente. Questo rispetta il vincolo "massimo un ciclo".

ANALISI SPERIMENTALE: LA SCIENZA DIETRO I NUMERI

Setup dell'Esperimento

Parametri scelti scientificamente:

- **N = 10000**: Mesh abbastanza fitta da approssimare bene il continuo
- **d = 1:50**: Range che mostra tutti i fenomeni interessanti
- **f(x) = 1/(x-1.3)**: Funzione con singolarità **strategicamente posizionata**

Perché f(x) = 1/(x-1.3)?

La singolarità a x=1.3 è **perfettamente calibrata**:

- **Abbastanza vicina** a [-1,1] da creare problemi ai nodi equispaziati
- **Abbastanza lontana** da non distruggere completamente l'interpolazione
- **Analitica su [-1,1]** quindi teoricamente interpolabile

È un **stress test intelligente**: distingue metodi robusti da quelli fragili.

Interpretazione dei Grafici

Grafico 1: Confronto Tempi

Cosa vedi:

- DLP (rosso): crescita quadratica pulita
- DLP2 (blu): crescita più controllata, crossover intorno a d≈30-40

Perché:

- DLP ha overhead per calcolare tante produttorie

- DLP2 ha setup iniziale (costruzione V) ma poi la LU scala meglio
- **Lezione**: Per gradi alti, l'algebra lineare avanzata batte la forza bruta

Grafico 2: Costante di Lebesgue

Cosa vedi:

- Crescita moderata, quasi lineare in scala semilog
- Oscillazioni naturali

Perché:

- I nodi di Leja mantengono Λ_n **sotto controllo**
- Le oscillazioni sono normali: aggiungere un nodo può temporaneamente migliorare la distribuzione
- **Confronto**: Per nodi equispaziati, vedremmo crescita esponenziale!

Grafico 3: Errori di Interpolazione

Cosa vedi:

- Leja (blu): decrescita controllata fino a saturazione ~10^{-16}
- Equispaziati (rosso): **esplosione catastrofica** per d > 30

Perché:

- **Fenomeno di Runge**: Gli equispaziati hanno Λ_n che esplode
- **Saturazione Leja**: Raggiungono la precisione macchina IEEE 754
- **Lezione**: Stabilità numerica > accuratezza teorica

DETTAGLI IMPLEMENTATIVI CRUCIALI

Gestione della Numerica

Clipping in DLP2:

```
x = max(-1, min(1, x));  % Previene NaN in acos()
```

Gli errori di rappresentazione floating-point potrebbero dare $|x| > 1$, mandando in crash `acos()`.

Uso di backslash:

```
c = V \ f_z;  % NON c = inv(V) * f_z;
```

Il backslash usa automaticamente la fattorizzazione più stabile disponibile.

Validazione Input

Ogni funzione ha controlli robusti:

```
if ~isvector(x) || ~isscalar(d) || d < 0 || round(d) ~= d || length(x) < d+1
    error('Input non valido...');
end
```

PERCHÉ QUESTO PROGETTO È GENIALE

Aspetti Pedagogici

1. **Collega teoria e pratica**: Dai teoremi di approssimazione al codice MATLAB
2. **Mostra trade-off reali**: Accuratezza vs stabilità vs efficienza
3. **Confronta approcci**: Greedy vs algebra lineare
4. **Quantifica i fenomeni**: Non solo "Runge è male", ma "quanto male"

Rilevanza Informatica

1. **Algoritmi**: Greedy vs ottimizzazione globale
2. **Strutture dati**: Matrici sparse, vettorizzazione
3. **Complessità**: Analisi asintotica pratica
4. **Numerica**: Condizionamento, stabilità, precisione macchina

Applicazioni Reali

Dove usi i nodi di Leja?

- **Computer graphics**: Interpolazione di curve/superfici
- **Simulation**: Riduzione di modelli complessi
- **Machine learning**: Approximation theory per neural networks
- **Financial modeling**: Interpolazione di superfici di volatilità

LEZIONI APPROFONDITE

Lezione 1: La Stabilità È Tutto

Principio: In numerica, un algoritmo stabile e "mediocre" batte sempre uno instabile e "ottimo".

Evidenza: I nodi di Leja non sono teoricamente ottimali (quelli sarebbero i Chebyshev), ma la loro **stabilità pratica** li rende superiori per molte applicazioni.

Lezione 2: L'Algebra Lineare È Potente

Principio: Problemi apparentemente diversi spesso si riducono ad algebra lineare standard.

Evidenza: Il problema geometrico "trova punti ben distribuiti" diventa "fai LU con pivoting su una matrice".

Lezione 3: Le Costanti Contano

Principio: La complessità asintotica non è tutto. Le costanti moltiplicative determinano quale algoritmo usi in pratica.

Evidenza: DLP e DLP2 hanno stessa complessità $O(M \cdot d^2)$, ma DLP2 vince per le costanti migliori.

Lezione 4: Il Testing È Scienza

Principio: Un buon esperimento isola le variabili e misura metriche multiple.

Evidenza: Il main testa tempo, stabilità, accuratezza su un range significativo con funzione calibrata.

DOMANDE PER RIFLETTERE

1. **Cosa succederebbe** se usassi la norma L^2 invece di L^∞ per l'errore?
2. **Come modificheresti** gli algoritmi per intervalli generici [a,b]?
3. **Quale metodo useresti** per interpolazione 2D su un quadrato?
4. **Come implementeresti** l'interpolazione adattiva (aggiunta dinamica di nodi)?
5. **Perché i nodi di Chebyshev** non sono sempre la scelta migliore?

CONCLUSIONE: DAL CODICE ALLA COMPRENSIONE

Questo progetto non è solo "implementa tre funzioni". È un viaggio attraverso:

- **Matematica applicata**: Teoria dell'approssimazione
- **Algoritmi**: Greedy vs globale, complessità pratica
- **Analisi numerica**: Stabilità, condizionamento, floating-point
- **Ingegneria software**: Testing sistematico, validazione
- **Scienza computazionale**: Simulazione, visualizzazione, interpretazione

Il vero apprendimento non è nel codice che funziona, ma nel **capire perché** funziona, **quando** usarlo, e **come** estenderlo.

Messaggio finale: In informatica, i problemi più interessanti stanno all'intersezione tra matematica, algoritmi e applicazioni reali. I nodi di Leja sono un esempio perfetto di come teoria elegante diventi strumento pratico potente.