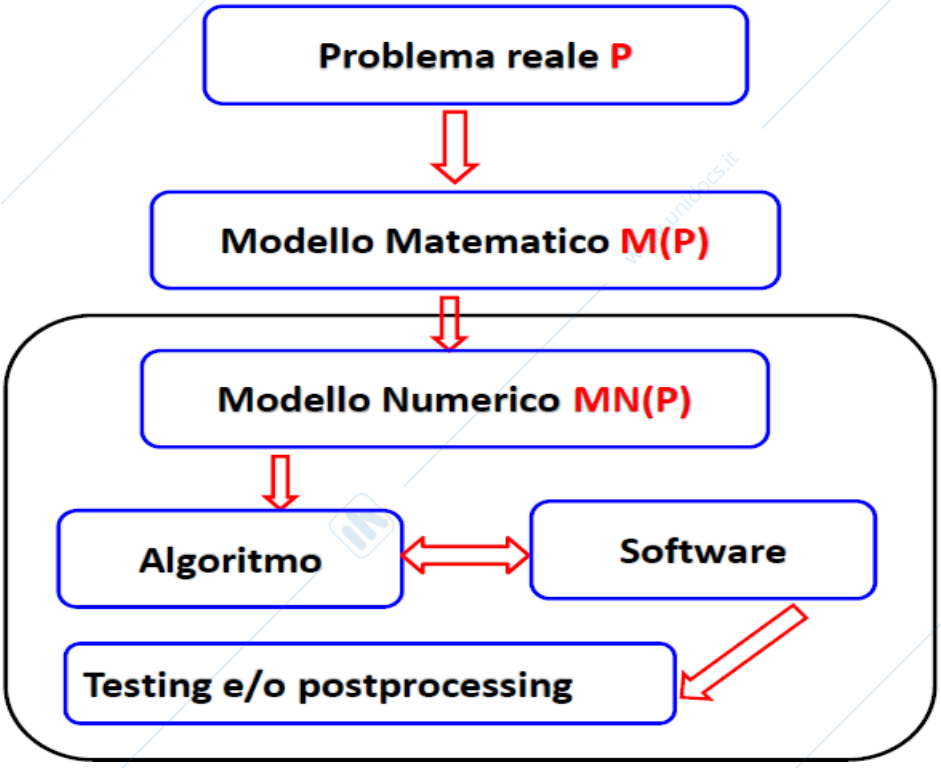
Introduzione al Calcolo Scientifico

Il calcolo scientifico è la disciplina atta a risolvere problemi reali con il calcolatore, quindi fornisce strumenti software per la risoluzione effettiva di problemi della scienza tramite lo sviluppo di modelli algoritmi e simulazioni. Il calcolo numerico invece è lo sviluppo di metodi e algoritmi per la risoluzione dei problemi in vari campi applicativi. La simulazione si utilizza quando un problema è troppo complesso, costoso, pericoloso o con molti dati e quindi se si conosce bene il problema è vantaggiosa in merito al costo e al tempo.

La risoluzione di un problema prevede la progettazione e l'implementazione di un algoritmo e la determinazione di una soluzione approssimata. Infatti non si può mai avere una soluzione esatta con il software, e valutando l'approssimazione, possiamo definire l'accuratezza e



l'efficienza di un algoritmo.

Le fasi evidenziate in figura sono le seguenti:

- Problema Reale
- Modello Matematico: si individuano le entità note, incognite e le loro dipendenze.
- Modello Numerico: è una discretizzazione del modello matematico che opera in R. Più modelli possono definire lo stesso problema.
- Algoritmo: è una sequenza finita di operazioni che a partire dagli input forniscono i dati di output. L'insieme numerico in cui opera è diverso, si parla infatti di Sistema Aritmetico come della rappresentazione dei dati in memoria e degli algoritmi per le operazioni.
- Software: implementazione dell'algoritmo in un linguaggio e in un ambiente di elaborazione che fornisce un risultato del problema in uno specifico ambiente.
- Testing: devo stimare la bontà della soluzione non conoscendola.

Gli errori che si effettuano durante questo processo sono:

- Errore di semplificazione: quando si passa dal problema reale al modello matematico e si trascurano dei parametri che non influenzano di molto.
- Errore di troncamento: dal modello matematico al numerico dovuto alla discretizzazione.
- Errore di round-off: è la differenza tra il risultato prodotto da un algoritmo in R e il risultato prodotto nel sistema a precisione finita dell'elaboratore, dipende quindi dal suo Sistema Aritmetico.

Da ciò si capisce che gli errori non si possono evitare e quindi bisogna trovare una soluzione approssimata valutando l'accuratezza del calcolo. Se y quindi è un'approssimazione del valore vero di x, si parla di:

1. Errore Assoluto: $E_a=|x-y|<10^{-p}$ se hanno p cifre decimali uguali.

2. Errore Relativo: $E_r = |x-y|/|x| < 10^{-p+1}$ se hanno p cifre significative uguali e x è diverso da zero. Questo è più interessante perché tiene conto dell'ordine di grandezza, infatti il valore vero è sempre incognito, quindi l'errore assoluto non esiste, si può solo stimare.

In generale il modello numerico converge a quello matematico e la velocità di convergenza indica l'efficienza. Noi assumiamo come risultato vero il risultato del MATLAB.

L'errore di round off dipende dal Sistema Aritmetico del calcolatore, quindi è importante capire questo come funziona.

Il tipo intero viene rappresentato con un criterio posizionale in base β (β=2=binario) e memorizzato come una stringa di bit in una voce di memoria (32 bit) di cui il primo indica il segno del numero. Esiste quindi un massimo ed un minimo intero rappresentabile e quindi posso avere overflow o underflow.

Il tipo reale è rappresentato con un criterio floating point normalizzato a precisione finita e viene detto l'insieme dei numeri macchina o Sistema Floating Point. Ogni x di R tranne zero può essere rappresentato univocamente in forma floating point normalizzata come: $x = \pm m*\beta^e$ dove m è detta mantissa tale che $\beta^{-1} < |m| < 1$, β è la base, ed e è l'esponente, e la prima cifra di m dopo il punto è diversa da zero. È economico perché si utilizzano solo le cifre significative.

Inoltre si osserva che molti numeri in base due (0.3,0.1) hanno infinite cifre e quindi bisogna memorizzarne un'approssimazione. In particolare in memoria bisogna conservare le cifre dopo il punto della mantissa e l'esponente. Il numero di bit destinati alla mantissa (t) e il numero di bit dell'esponente (n) definiscono univocamente

l'insieme dei numeri macchina. Questo insieme finito è definito come F(β,t,L,U) in cui:

- B= base
- t= precisione, numero di bit per la mantissa, la cui prima cifra è sempre 1.
- [L,U] range per l'esponente.

A questo si aggiunge lo zero che ha due rappresentazioni (±0). Si capisce da ciò che il sistema prevede elementi non equispaziati, ma la distanza è data da β^{-t}, e poiché c'è un minimo e un massimo reale rappresentabile, posso andare in overflow (intervallo infinito) e in underflow (piccolo intervallo).

Quindi, dato un numero reale x si possono avere tre situazioni:

1. Analizzando l'esponente, se sono in underflow o overflow, x non è rappresentabile.
2. Analizzando la mantissa, se ha più di t cifre è rappresentabile con un arrotondamento.
3. Il numero reale è proprio il numero macchina. Quasi impossibile.

Nel secondo caso, che è quello che si verifica più spesso, opero sulla rappresentazione floating point di x che è diversa dal numero reale, infatti si utilizza la tecnica dell'arrotondamento poiché ogni numero macchina rappresenta un intervallo infinito di numeri reali, e quindi la probabilità che un numero reale sia proprio un numero macchina è zero.

Approssimando quindi x con fl(x) commetto un errore di round-off che mi indica che in F posso avere al massimo t cifre significative corrette (singola precisione=6-7, doppia

precisione=14-15). Si definisce quindi questa quantità epsilon macchina che è una costante caratteristica di un sistema Floating Point e rappresenta una misura della massima precisione raggiungibile: corrisponde all'errore relativo di round-off quando rappresento un numero reale in un sistema Floating Point, quindi se un calcolo fornisce come risultato fl(x), esso è accurato a meno di ε|x|.

Sia $F(\beta,t\,L,U)$, $x \in R\ x\#0$
$x=m\times\beta^e, fl(x) = m1 \times\beta^e$
ricordando che $\beta^{-1} \leq |m|$ si ha:

$$\frac{|fl(x)-x|}{|x|} = \frac{|m\cdot}{|r} \quad |fl(x)-x| \leq \varepsilon\,|x| \qquad ^{1-t}$$

Un altro errore di round-off si ha facendo operazioni aritmetiche tra numeri macchina, ossia il risultato di un'operazione tra numeri macchina non è un numero macchina. Le operazioni sono eseguite nei registri dell'ALU i cui registri sono il doppio della mantissa t. Un esempio di ciò si può vedere per la somma, per essere effettuata si devono eseguire i seguenti passi:

$$a(*)b = fl(a*b) = (a*b)(1+\delta) \quad \text{con } |\delta| \leq \frac{1}{2}\beta^{1-t}$$

1. Confronto degli esponenti degli addendi
2. Shift a destra della mantissa con esponente minimo per eguagliare gli esponenti
3. Somma bit a bit a mantisse incolonnate
4. Arrotondamento e normalizzazione del risultato. Questo può sforare perché non è un numero macchina

Per questo non valgono le proprietà associativa e distributiva, ma gli algoritmi devono seguire un certo criterio per sommare, ad esempio si sommano prima tutti i numeri più piccoli che altrimenti dopo un certo tempo non peserebbero più, in modo da ridurre l'errore. Inoltre non c'è l'unicità dell'elemento neutro, infatti ogni numero macchina presenta un insieme di elementi che si comporta

come zero, ossia sommandoli non danno contributo ma riottengo il numero iniziale. Ciò si verifica principalmente quando shifto di più di t bit e il bit t+1 è minore di 5.

In particolare dato il numero 1, il primo numero macchina che sommato ad esso mi da un risultato diverso è l'epsilon macchina. Esso individua la soglia degli zeri del numero 1 e rappresenta la distanza tra 1 e il più vicino numero macchina maggiore di 1. Per ogni numero macchina vale lo stesso e si definisce l'epsilon relativo ad x : ε|x|.

Per ogni $x \in F$ $\exists$ il più piccolo $y \in F$:

$$fl(x+y) > x$$

per $x > y > 0$

$$fl(x+y) = fl(x(1+y/x)) > x \quad \text{se} \quad \frac{y}{x} > \varepsilon$$

$$fl(x+y) = x \quad \text{se} \quad \frac{y}{x} \leq \varepsilon$$

Aritmetica Standard

Presenta le due modalità di singola precisione (32 bit) e doppia precisione (64 bit) in cui il primo bit rappresenta il segno, e il primo bit della mantissa non è memorizzato perché è sempre 1. In singola precisione l'esponente appartiene all'intervallo intero [0,255] (bias), ma il valore in memoria dell'esponente [-127,128]. Ci sono alcune situazioni eccezionali che sono gestite dai valori minimo e massimo dell'esponente:

	β	t	L	U
IEEE SP	2	23	-126	127

Bits	Usage
31	Sign (0 = positive, 1 = negative)
30 to 23	Exponent, biased by 127
22 to 0	Fraction f of the number 1.f

	β	t	L	U
IEEE DP	2	52	-1022	1023

Bits	Usage
63	Sign (0 = positive, 1 = negative)
62 to 52	Exponent, biased by 1023
51 to 0	Fraction f of the number 1.f

- e=255, m≠0: val=NotANumber (operazione non valida)
- e=255, m=0: val =±infinito (divisione per zero/overflow)

- e=0, m≠0: val = ±0
- e=0, m=0: val= subnormal numbers (underflow)

Su questi casi è possibile effettuare delle operazioni, solo se il compilatore le può gestire. Per la gestione dei subnormal numbers, se non posso fare niente li pongo a zero, altrimenti effettuo la denormalizzazione: esprimo il numero con esponente ammissibile avendo però il primo bit della mantissa diverso da 1, pongo l'esponente a zero, a questo punto posso spostare fino al ventitreesimo bit della mantissa. Se non ricado in questi casi lo pongo a zero. Questi non sono numeri macchina.

Il minimo numero rappresentabile, essendo L=-126, è: $minr=1.1755*10^{-38}$. Il massimo numero rappresentabile, essendo U=127, è: $maxr=3.4028*10^{38}$.

Detto ciò si capisce che quello che, per via degli errori definiti in precedenza, ho in macchina è un problema diverso da quello reale che deve essere il più possibile vicino al vero. Non ha senso quindi testare l'uguaglianza tra due numeri, ma, poiché due numeri sono uguali quando l'errore relativo è circa epsilon, bisogna quindi testare che $|x-y| \sim \varepsilon |x|$, anche perché non posso verificare x-y=0 essendo lo zero non esistente nel calcolatore.

while $|x-1| \geq \varepsilon$ test affidabile

in generale while (x#y) , if(x#y) ...diventa
while($|x-y| \geq \varepsilon |x|$),...
test sulle cifre significative uguali

Criterio Arresto e Metodo Iterativo

Il modello matematico per e^x (serie di Taylor) essendo infinito va approssimato, per cui lo approssimo ad una somma finita fino a N generando un errore di troncamento. Il problema nasce sul come fissare N. Questo deve essere determinato dinamicamente perché ogni N è diverso a seconda della x in ingresso: all'aumentare della x aumenta

anche N per avere il valore ottimale, ma se N è grande effettuo molte operazioni inutili che non aggiungono informazione al risultato. Per questo mi arresto nel while quando l'addendo è minore dell'epsilon relativo a S_{K-1}.

Il metodo iterativo è una formula ricorrente che a partire da un valore iniziale genera una successione x_n che converge alla soluzione x di un problema. Ad ogni passo si genera un errore $e_n=|x- x_n|$ non calcolabile e di cui si può ottenere solo una stima. Questo metodo permette di calcolare la velocità di convergenza, più è grande p più riduco l'errore velocemente ad ogni passo. Mi arresto quando non posso più correggere il valore.

$$|e_{n+1}| \leq c \cdot |e_n|^p$$

Poiché non si sa se l'uscita converge, si inserisce un'uscita di sicurezza n<NMAX, ossia al massimo effettuo NMAX iterazioni. Se il problema non richiede il numero massimo di cifre significative esatte del sistema, ma voglio solo k cifre esatte, si introduce il parametro della tolleranza: $TOL=10^{-K}$ che va a sostituire eps nel criterio d'arresto e può essere definito come parametro d'ingresso dall'utente. Bisogna quindi verificare che TOL sia corretto, ossia deve essere TOL >ε e TOL >=0, altrimenti si utilizza una tolleranza di default avvisando l'utente. Bisogna inoltre, se la successione tende a zero, essendo TOL una quantità piccola, evitare che $TOLX=TOL|x_n|$ vada in underflow, se ci va la pongo uguale a minr.

Condizionamento e Stabilità

Si parla di condizionamento come dell'inevitabile errore di round-off nei dati di input, e di stabilità come l'errore di round-off delle operazioni che possono essere amplificati rispettivamente dal problema e dall'algoritmo. In particolare dato il problema reale con soluzione x e dato il problema nel computer con soluzione y, se x è vicino ad y il problema è bencondizionato (errore soluzione è circa

l'errore dei dati), altrimenti si dice malcondizionato (l'errore della soluzione è molto maggiore dell'errore dei dati) e in tal caso bisogna cambiare modello del problema. Basta una piccola perturbazione per modificare completamente la soluzione del problema (due rette non parallele potrebbero diventarlo).

Si parla di cancellazione catastrofica quando, effettuando la sottrazione in un sistema F con due operandi quasi uguali, l'errore di round-off su una cifra poco significativa si amplifica spostandosi su una cifra più vicina alla testa per effetto della normalizzazione.

Riesco a capire quindi se un sistema è mal condizionato in base ad una stima che fornisce un numero, questo è detto K è indica quante volte l'errore nei dati è maggiore rispetto all'errore della soluzione. Il valore ottimo è K=1, ma in generale $K=10^q$. Se ho quindi t cifre, ne perdo q, ossia l'errore della soluzione è circa 10^{q-t}. Se quindi t è circa q è inutile risolvere il problema poiché è intrattabile.

Algoritmo e Software

Un algoritmo è stabile se l'errore di round-off nelle operazioni dell'algoritmo non si amplifica nel risultato il quale è la soluzione esatta di P', altrimenti si dice instabile. Alcuni algoritmi sono corretti in R, ma non in F quindi posso calcolare il risultato con le formule tradizionali solo se hanno lo stesso segno, ossia non ci sono sottrazioni e quindi cancellazioni catastrofiche (per $x<0$ e^{-x} è instabile). Se un problema è ben condizionato allora esiste un algoritmo stabile che lo risolve, ma può dare errori se risolto con un algoritmo instabile.

Un programma quindi deve basarsi su un buon algoritmo che deve risolvere il problema con efficienza (numero operazioni e variabili utilizzate), ossia in poco tempo e occupando poca memoria, e con stabilità. Il Flops indica il

numero di operazioni di un sistema floating point in un secondo e ne definisce quindi la velocità. Ad esempio l'algoritmo di Gauss è computazionalmente più efficiente del metodo di Cramer, così come per valutare un polinomio, al posto di utilizzare l'algoritmo generale il cui numero di flops è (n^2+n)Moltiplicazioni e nAddizioni, si utilizza l'Algoritmo di Horner il cui numero di flops è nMoltiplicazioni e nAddizioni poiché x moltiplica sempre un polinomio di primo grado.

Un software non è la banale traduzione di un algoritmo. Le sue performance si calcolano in base al tempo di esecuzione, che è il tempo della CPU per eseguire il programma, e l'elapsed time che è il tempo tra il momento in cui il software è mandato in esecuzione ed il suo completamento. Un software deve garantire:

- Affidabilità:
  - Accuratezza: misura dell'effettivo errore di round-off. Dipende se l'algoritmo è stabile o meno.
  - Correttezza codice: congruenza tra le specifiche del programma e i risultati. Può essere analizzata staticamente, dinamicamente e tramite testing strutturale.
- Efficienza: il tempo di elaborazione ed occupazione di memoria. Dipende dal numero di operazioni floating point.
- Robustezza: controllo delle condizioni anomale e loro gestione.

Documentazione Software

È l'interfaccia tra utente e software che contiene le istruzioni di utilizzo, le informazioni sull'organizzazione interna e le esperienze. Può essere interna (commenti esplicativi) o esterna (manuale, help). La documentazione esterna prevede diverse sezioni:

1. Scopo: breve descrizione dei problemi risolubili, dell'algoritmo usato e raccomandazioni sull'uso.
2. Specifiche: intestazione della procedura.
3. Parametri: parametri I/O con tipo e descrizione.
4. Diagnostica: errori previsti.
5. Complessità e accuratezza: informazioni su queste caratteristiche.
6. Esempio d'uso: esempio con input e risultati.

Per effettuare i test di accuratezza e robustezza si crea un live script e lo si salva in pdf.

Zeri di una Funzione – Algoritmo di Bisezione

Una funzione f continua presenta uno zero nell'intervallo [a,b] se il prodotto $f(a)f(b)<0$, ossia deve cambiare di segno nell'intervallo. L'algoritmo di bisezione parte da un intervallo [a,b] che verifica la condizione precedente e ne calcola il valore nel punto medio x_m. Se è uguale a zero si arresta, altrimenti se ha lo stesso segno di f(a) pongo $a=x_m$ e b=b, altrimenti pongo a=a e $b=x_m$. Si ripete ciò finché l'ampiezza dell'intervallo è sufficientemente piccola in modo che ogni punto di esso approssimi lo zero.

È un metodo iterativo in cui l'ampiezza si dimezza ad ogni passo e la successione dei punti medi converge allo zero lentamente. Il criterio di arresto quindi deve verificare che:

1. l'intervallo normalizzato al massimo degli estremi è minore della tolleranza.
2. Il valore della funzione è minore della tolleranza rispetto alla funzione.
3. Il numero di iterazioni non supera il valore di default.

```
if f(a)*f(b)>0  errore  non vi sono zeri
      c =(a+b)/2
      while (criterio di arresto)
            if (f(c )f(a)< 0)  then
               b= c
             else
                a= c
            endif
          c =(a+b)/2
     endwhile
```

Se aumento la TOL sull'intervallo effettuo più passi. Il confronto va effettuato con la fzero del MATLAB ed in particolare nel numero di iterazioni effettuate.

Algebra Lineare Numerica I & II

L'algebra lineare è il settore più importante del calcolo scientifico poiché descrive circa il 70% dei problemi reali. Un sistema lineare è composto da una matrice dei coefficienti A ed un vettore dei termini noti b di cui si vuole calcolare la soluzione x. Bisognerebbe quindi risolvere il sistema $x=A^{-1}b$, ma calcolare l'inversa di una matrice (solo se non è singolare) è computazionalmente impossibile (si dovrebbero risolvere N sistemi lineari); inoltre anche il calcolo del determinante per verificare se la matrice è singolare (detA=0) o meno, ha un tempo di calcolo enorme.

La norma è un numero reale che permette di stimare la grandezza di un vettore e deve soddisfare le seguenti proprietà:

1. ||x||>= 0 e ||x||=0 solo se x=0;
2. ||a x||= |a|*||x||;
3. ||x+y||<=||x||+||y||.

Esistono vari tipi di norme, alcune delle quali elencate in figura che hanno però tutte lo stesso ordine di grandezza. Dalla norma vettoriale si può

$||x||_2 = \sqrt{x_1^2 + .. + x_n^2}$ norma euclidea

$||x||_\infty = \max_i |x_i|$ norma del massimo

$||x||_1 = \sum_i |x_i|$ norma 1

$$\|A\| = M = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$ = norma matriciale

introdurre la norma matriciale, e in particolare, quando una matrice B trasforma il vettore in uno avente norma più grande di un'altra matrice A, si dice che B è più grande di A.

$\|A\|_2 = \sqrt{\rho(A^\top A)}$ **norma euclidea (ρ raggio spettrale(max autovalore))**

$\|A\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$ **norma del massimo**

$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|$ **norma 1**

Condizionamento di un sistema lineare

Poiché i dati di input sono sempre affetti da un errore in realtà non sto risolvendo Ax=b, ma nel sistema floating point ho (A+δA)x'=(b+δb) con le delta che sono gli errori di round-off relativi alla matrice A e al vettore b. Quindi la soluzione del sistema perturbato x'=x+δx è affetta dall'errore δx. Si dimostra quindi che l'errore relativo della soluzione nel

$$\frac{\|\delta x\|}{\|x\|} \le \|A\| \, \|A^{-1}\| \times \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right)$$

$\mu(A) = \|A\| \, \|A^{-1}\|$ **indice di condizionamento**

sistema perturbato è minore del prodotto delle norme della matrice e la sua inversa, per l'errore relativo ai dati. La quantità relativa alle norme è definita indice di condizionamento che non dipende dal termine noto e rappresenta l'amplificazione dell'errore dei dati nel risultato, per cui minore (1) è, meglio condizionato è il nostro sistema.

Se A è malcondizionata è vicina ad una matrice singolare e viene detta quasi singolare e il reciproco dell'indice di condizionamento mi rappresenta la distanza tra A e l'insieme delle matrici singolari.

$$\mu(A) \cong 10^{q} \quad , \varepsilon \cong 10^{-t} \text{ (errore di round off)}$$

$$\frac{\|\delta x\|}{\|x\|} \cong 10^{q} \times 10^{-t} = 10^{q-t}$$

$q \geq t$ sistema molto malcondizionato

$q < t$ la soluzione calcolata potrebbe perdere fino a q cifre significative

Ci sono delle matrici che sono di natura malcondizionate e quindi intrattabili; queste sono:

- Matrici di Hilbert: per n che tende ad infinito anche l'indice di condizionamento lo fa.
- Matrici di Vandermonde: l'indice di condizionamento cresce esponenzialmente con n.

Per calcolare il numero di cifre significative corrette si calcola l'errore relativo ossia il rapporto tra la norma della differenza tra la soluzione unitaria e la soluzione calcolata, e la norma della soluzione unitaria.

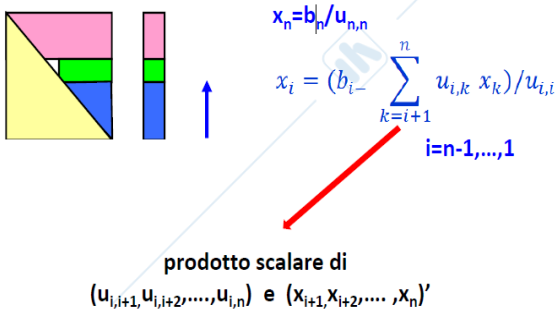
Risoluzione sistemi lineari-Gauss

Risolvere quindi un sistema lineare calcolando l'inversa è inefficiente. Per questo si utilizza l'algoritmo di Gauss che trasforma il sistema in uno di semplice risoluzione, ossia

computazionalmente efficiente, ed in particolare ottengo dei sistemi triangolari da risolvere.

Forward e Back Substitution

Per una matrice triangolare superiore si applica l'algoritmo di back substitution. In pratica poiché nell'ultima equazione ho una sola incognita, la risolvo e poi la sostituisco all'equazione precedente. La complessità di operazioni è:



- Moltiplicazioni: (n(n+1)/2)
- Addizioni: (n(n-1)/2)

Quindi in generale è un $O(n^2/2)$

Per una matrice triangolare inferiore vale esattamente l'opposto, si applica l'algoritmo di forward substitution che ha esattamente la stessa espressione e complessità.

Questi algoritmi si possono applicare soltanto se il determinante della matrice triangolare è diverso da zero, altrimenti sarebbero singolari.

Algoritmo back-substitution

```
for j=1,n
 if u_{j,j}=0 then
  "matrice singolare" ; stop
 endif
endfor
 x_n=b_n/u_{n,n}
for i = n-1,1
    sum=0
   for k=i+1,n
      sum=sum+u_{i,k}x_k
    endfor
    x_i=(b_i-sum)/u_{i,i}
   endfor
```

Algoritmo forward-substitution

```
for j=1,n
  if l_{j,j}=0  then
   "matrice singolare" ;  stop
  endif
endfor
 x_1 =b_1 /l_{1,1}
 for i = 2,n
   sum=0
   for k=1,i-1
       sum=sum+l_{i,k} x_k
    endfor
   x_i =(b_i -sum)/l_{i,i}
 endfor
```

Inoltre per le matrici triangolari, il calcolo del determinante è applicabile con la formula diretta del prodotto degli elementi sulla diagonale. Per testare se l'elemento della diagonale è zero bisogna calcolare lo zero relativo:

'zero'=eps*norm(A). Il secondo ciclo for è vettorializzabile essendo un prodotto scalare.

Algoritmo di Gauss

È un metodo diretto che in un numero finito di passi fornisce un'approssimazione della soluzione senza errore di discretizzazione. L'obiettivo è trasformare il sistema Ax=b nel sistema equivalente Ux=y con U triangolare superiore e poi risolverlo tramite back substitution: ad ogni passo l'algoritmo deve eliminare gli elementi al di sotto del corrispondente elemento della diagonale. Al generico passo k quindi devo:

1. Calcolare i moltiplicatori delle righe k+1..n.
2. Eliminare gli elementi della colonna k.

L'algoritmo quindi dopo n passi modifica la matrice A e il vettore b. La nuova matrice viene memorizzata sempre sulla matrice di partenza e i moltiplicatori vengono memorizzati nelle colonne degli elementi che si azzerano, quindi l'algoritmo è in place, ossia non richiede locazioni di memoria aggiuntive ($S(n)=O(n^2)$) e la complessità temporale è $T=O(n^3/3)$.

$$m_{i,k} = a_{i,k}^{(k-1)} / a_{k,k}^{(k-1)} \rightarrow \text{PIVOT}$$

$$a_{i,j}^{(k)} = a_{i,j}^{(k-1)} - m_{i,k} a_{k,j}^{(k-1)} \qquad i,j=k+1,\ldots,n$$

$$b_i^{(k)} = b_i^{(k-1)} - m_{i,k} b_k^{(k-1)} \qquad 1,\ldots,n$$

Algoritmo di GAUSS in place

```
for k=1,n-1
   for i=k+1,n
      a_{i,k} = a_{i,k} / a_{k,k}   calcolo moltiplicatori
      for j=k+1,n
            modifica elementi matrice attiva
         a_{i,j} = a_{i,j} - a_{i,k} a_{k,j}
      endfor
      modifica vettore termini noti
      b_i = b_i - a_{i,k} b_k
   endfor
endfor
```

Per risolvere un sistema lineare impiego quindi un tempo pari a $T(n)=O(n^3/3)+O(n^2/2)$ che è la somma dei tempi di Gauss e della back substitution.

Se il pivot è zero, l'algoritmo è inapplicabile perché dovrei effettuare una divisione per zero, però una soluzione consiste nello scambiare le righe della matrice per far procedere l'algoritmo. Avrei lo stesso problema anche se il pivot fosse molto piccolo, perché l'algoritmo diventerebbe instabile, infatti, il moltiplicatore diventa sempre più grande, e poiché siamo in un sistema floating point, l'errore di round-off verrebbe amplificato moltissimo perdendo cifre corrette nella soluzione. I due problemi principali sono quindi:

1. Se il pivot è zero l'algoritmo è inapplicabile, quindi ad ogni passo deve essere diverso da zero.
2. Se il pivot è piccolo, l'algoritmo è instabile, quindi deve essere il moltiplicatore in modulo minore o uguale ad 1 ad ogni passo. In questo modo l'errore di round-off si riduce e l'algoritmo diventa stabile.

Per risolvere questi due problemi si utilizza il pivoting parziale: ad ogni passo k scambio le righe in modo che il pivot sia l'elemento massimo in modulo della colonna k. Se l'elemento massimo è zero vuol dire che tutti gli elementi della colonna sono nulli e quindi la matrice A è singolare. Questa tecnica quindi rende stabile Gauss, permette l'esecuzione

Algoritmo di Gauss con pivoting parziale

```
for  k=1,n-1
determinare il più piccolo r:
    |a_{r,k}|=max|a_{i,k}|, i≥k
 if  a_{i,k}#0  then
se r#k scambia righe k ed r, b_r e b_k
      for i=k+1,n
          a_{i,k} =a_{i,k} /a_{k,k}
       for  j=k+1,n
          a_{i,j}=a_{i,j} −a_{i,k} a_{k,j}
       endfor
          b_i =b_i-a_{i,k}b_k
     endfor
   else "A singolare",    (stop)
  endif
endfor
if a_{n,n} = 0 then "sistema singolare"
```

senza divisione per zero e permette di verificare la singolarità di una matrice. Non è necessario effettuare veramente gli scambi, ma basta memorizzarli in un vettore piv(n) che punta alle righe e scambiare gli elementi al suo interno. Quindi al posto dell'elemento $a_{i,j}$ nell'algoritmo devo considerare l'elemento $a_{piv(i),j}$.

Poiché nessun algoritmo può risolvere esattamente un sistema di equazioni lineari, ma solo un problema vicino a quello vero, possiamo definire due misure:

- $e=x-x^*$ errore che misura quanto x^* è vicino a x.
- $r=b-Ax^*=A(x-x^*)=Ae$ residuo che misura quanto x^* soddisfa il sistema perturbato.

Se l'errore è zero lo è anche il residuo, ma un residuo piccolo non implica errore piccolo, infatti se il sistema è malcondizionato la soluzione è sempre lontana da quella vera.

Il teorema di Wilkinson dice che la soluzione x^* calcolata tramite Gauss con pivoting è la soluzione esatta del sistema perturbato $(A+E)x^*=b$. In cui gli errori di round off dell'algoritmo sono assimilabili ad un'unica perturbazione su A, e quindi l'algoritmo è ottimale. Il residuo relativo è sempre piccolo, quindi posso dedurre che x^* risolve "esattamente" il problema perturbato. Una volta calcolato posso quindi definire anche l'errore relativo che è

$$\|E\| \leq C\varepsilon \|A\| \quad C \text{ quasi mai } >>10$$

$$r=b-Ax^*=(A+E)x^*-Ax^*=Ex^*$$

$$\|r\| \leq C\varepsilon \|A\| . \|x^*\|$$

Da cui il residuo relativo è

$$\frac{\|r\|}{\|A\|.\|x^*\|} \leq C\varepsilon$$

$$\frac{\|r\|}{\|A\|.\|x^*\|} \leq C\varepsilon$$

Ma

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

Passando alle norme si ha che:

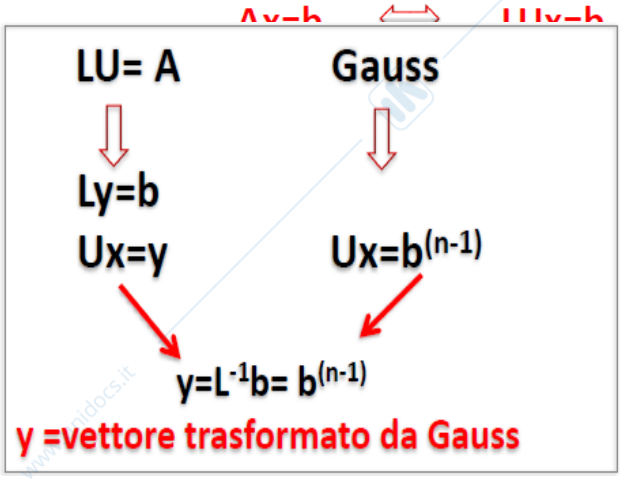
$$\|x-x^*\| \leq C\varepsilon \|A^{-1}\|. \|A\|.\|x^*\|$$

e l'errore relativo è

$$\frac{\|x-x^*\|}{\|x^*\|} \leq \mu(A)\, C\varepsilon$$

grande se l'indice di condizionamento di A è grande, altrimenti è assimilabile al residuo.

Fattorizzazione LU

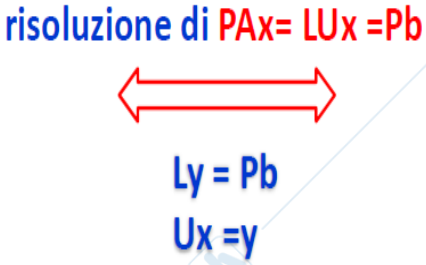


In molte applicazioni bisogna risolvere sistemi con stessa matrice, ma coefficienti diversi quindi conviene separare la trasformazione di A e quella di B. In particolare se A è di dimensione NxN e le sottomatrici principali non sono singolari(più fattorizzazioni), allora esiste una sola fattorizzazione di A: A=L x U dove L è triangolare inferiore e U triangolare superiore. Per risolvere il sistema devo quindi risolvere in ordine due sistemi, il primo applicando la forward substitution e il secondo con la back substitution.

```
for  k=1,n-1
    for i=k+1,n
L       a_{i,k}=a_{i,k} /a_{k,k}
        for  j=k+1,n
U           a_{i,j}=a_{i,j}-a_{i,k}a_{k,j}
        endfor
    endfor
endfor
```

L'algoritmo di Gauss effettua la fattorizzazione LU dove L è la matrice dei moltiplicatori in cui la diagonale è 1, e U è la matrice che si ottiene alla fine del procedimento. In pratica la fattorizzazione consiste nell'algoritmo di Gauss applicato solo ad A. L'algoritmo ad ogni passo calcola una riga di U ed una colonna di L ed effettua praticamente le stesse operazioni dell'algoritmo di Gauss. Anche quest'algoritmo non necessita di spazio aggiuntivo perché memorizza gli elementi di L ed U sempre nella matrice A, e anche la complessità temporale è la stessa di Gauss.

Poiché Gauss effettua il pivoting se effettuo il prodotto LxU, ottengo le righe di A ma non nello stesso ordine, per questo



bisogna introdurre una matrice di permutazione che ricorda gli scambi e in questo modo PxA=LxU, ossia LUx=(PA)x=Pb, devo effettuare quindi gli stessi scambi su b. Per il calcolo del determinante di A mi accorgo che il determinante di L è pari a 1 essendo la diagonale identica, quindi $det(A)=(-1)^s * det(U)$ con s che corrisponde al numero di scambi effettuati, con un tempo $T(n) = O(n^3/3)$ al posto di O(n!).

In realtà non è necessario costruire P, ma utilizzando il pivoting virtuale, memorizza gli scambi con il puntatore

```
for i=1,n
    piv_i=i
endfor
for  k=1,n-1
 determinare il più piccolo r: |a_piv(r),k|=max|a_piv(i),k|, i≥k
     if  a_piv(r),k≠0  then
     se r≠k scambia  piv(r) con piv(k)
        for i=k+1,n
L        a_piv(i),k=a_piv(i),k/a_piv(k),k
          for  j=k+1,n
U          a_piv(i),j=a_piv(i),j-a_piv(i),k a_piv(k),j
          endfor
        endfor
       else “sistema singolare”,  (stop)
      endif
endfor
 if a_piv(n),n =0 then “sistema singolare”
```

alle righe piv.

Algoritmo di F.S. per Ly=Pb

$y_1=b_{piv(1)}$
for i = 2,n
  sum=0
  for k=1,i-1
    sum=sum+$a_{piv(i),k}$ y_k
  endfor
  $y_i=b_{piv(i)}$-sum
endfor

Algoritmo di B.S. per Ux=y

$x_n=y_n/a_{piv(n),n}$
for i = n-1,1
  sum=0
  for k=i+1,n
    sum=sum+$a_{piv(i),k}$ x_k
  endfor
  $x_i=(y_i$-sum$)/a_{piv(i,i)}$
endfor

Per effettuare il calcolo dell'inversa teoricamente dovrei risolvere n sistemi lineari in cui la matrice è sempre la stessa A e il vettore incognito è una colonna dell'inversa differente ad ogni passo. Utilizzando la fattorizzazione LU invece effettuo il pivoting di A una sola volta riducendo la complessità che diventa: $T(n)=O(n^3/3+n*n^2) = O(4n^3/3)$.

for $j = 1 : n$
  pongo $b = e_j$
  risolvo $Ly = Pb$
  risolvo $Ux = y$
  copio x nel vettore colonna j – *simo* della matrice
end

Per valutare il numero di cifre significative corrette dovrei calcolare l'errore relativo err=norm(x-xc)/norm(x) , ma poiché non conosco x, lo posso stimare tramite il residuo relativo (Wilkinson).

res= norm(b-A*xc)/(norm(A)*norm(xc)) ≈
≈ norm(b-A*xc)/norm(b)

e l'errore relativo è

err ≈ cond(A)* res

La fattorizzazione LU con pivoting è la tecnica migliore per risolvere un sistema lineare.

L ed U vengono calcolate anche se la matrice è singolare, è l'algoritmo di Gauss (\) che se ne accorge.

Non è necessario il pivoting se A ha diagonale strettamente dominante e A è simmetrica e positiva definita ($A^T=A, x^TAx>0$ per ogni x diverso da 0), infatti in questo caso il pivot è già l'elemento di massimo modulo.

$$|a_{ii}| > \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}|, i = 1,..n$$

Sistemi con matrice tridiagonale

forward substitution Ly=b

$y_1 = b_1$
for i =2,n
  $y_i=b_i-l_iy_{i-1}$
endfor

T(n)=O(n) S(n)=O(n)

$$A=\begin{pmatrix} d_1 & f_1 & & & \\ a_2 & d_2 & f_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & & f_{n-1} \\ & & & a_n & d_n \end{pmatrix}$$

Senza pivoting si ha che A=LU con L ed U :

$$L=\begin{pmatrix} 1 & & & \\ l_2 & 1 & & \\ & \ddots & \ddots & \\ & & l_n & 1 \end{pmatrix} \qquad U=\begin{pmatrix} u_1 & f_1 & & \\ & u_2 & \ddots & \\ & & \ddots & f_{n-1} \\ & & & u_n \end{pmatrix}$$

back substitution Ux=Y

$x_n = y_n/u_n$
for i=n-1,1
  $x_i=(y_i-f_ix_{i+1})/u_i$
endfor

T(n)=O(n) S(n)=O(n)

Nei problemi reali le matrici spesso sono a banda , diagonali e a diagonale dominante, per cui ci sono tecniche di memorizzazione ed algoritmi appositi per ridurre complessità temporale e spaziale. Infatti l'algoritmo di fattorizzazione LU si specializza se la matrice è tridiagonale, questo perché posso costruire direttamente L ed U evitando calcoli intermedi (se non è necessario il pivoting). Infatti so che L ed U saranno costituite solo da due diagonali, e quindi sono due vettori per cui L x U = A coincide con il prodotto righe per colonne. Quindi l'algoritmo di forward/back substitution consiste di un solo ciclo for.

Se fosse necessario il pivoting la banda delle matrici si allargherebbe e quindi potrebbe essere necessario effettuare degli scambi di righe.

Grafi

Un grafo è una rappresentazione binaria tra oggetti che permette di visualizzare strutture con relazioni complesse. È costituito da un insieme di nodi e archi e può essere orientato (unidirezionale) o meno (bidirezionale). Per tracciare un grafo bisogna tenere conto delle:

- coordinate dei nodi.
- Matrice di adiacenza A composta da 0 e 1 di dimensione pari al numero di nodi i cui elementi sono 1 se i due nodi sono connessi, altrimenti 0. Se il grafo è non orientato la matrice è simmetrica; inoltre la diagonale è nulla (non ci sono autoarchi) e può avere dei pesi relativi al collegamento che vengono memorizzati al posto degli 1.

Per un grafo non orientato si parla di degree di un nodo i come del numero di nodi connessi ad i. Per un grafo orientato si distingue tra outdegree (archi in uscita) e indegree (archi in ingresso). In genere un grafo rappresenta la distribuzione degli elementi diversi da 0 di una matrice sparsa, tramite la matrice di adiacenza, perché solitamente ci sono molti nodi e poche connessioni tra essi.

Catene di Markov

Le catene di Markov sono utilizzate per simulare l'evoluzione di un qualcosa. La dinamica dei cammini è rappresentata tramite la matrice di transizione P in cui gli elementi p_{ij} rappresentano la possibilità di spostarsi da j a i

e quindi possono assumere i valori 1/m se j ed i sono collegate o altrimenti 0.

P è quindi una matrice sparsa di elementi compresi tra 0 e 1 in cui le colonne rappresentano il punto di partenza e le righe il punto di arrivo. La somma degli elementi di una colonna, che rappresenta la probabilità di trovarsi nel luogo i al tempo t, deve essere 1.

$$\sum_{i=0}^{n} x_i = 1$$

Si definisce con P^k la matrice di probabilità di passaggio da un luogo all'altro in k passi e p_{ij}^k la probabilità di trovarsi nel luogo i, partendo da j, dopo k passi.

Una catena di Markov quindi si definisce con questi tre elementi:

- S spazio degli stati
- X(0) distribuzione di probabilità iniziale
- P matrice di transizione

P tale che:

$$p_{ij} \geq 0$$

$$\sum_{i=1}^{n} p_{ij} = 1$$

La situazione all'istante t dipende solo dall'istante t-1, non dal passato.

Una catena si dice irriducibile se

1. Esiste un cammino da i a j per ogni i e j.
2. Se non ci sono stati trappola da cui non posso uscire(p_{ij}=1).
3. Se non c'è un sottoinsieme di stati trappola (loop).

Se la catena è irriducibile, posso rappresentarla come un grafo strettamente connesso in cui i nodi sono gli stati e gli archi li connettono solo se la probabilità è diversa da zero. La matrice di adiacenza è la trasposta di P e ponendo gli elementi non nulli pari ad 1 si ottiene il grafo (i vertici i e j sono invertiti in matlab).

Se è irriducibile si può dimostrare che P ha il più grande autovalore λ pari ad 1, e quindi esiste un unico x, qualunque sia la distribuzione iniziale tale che: x=Px=

$\lim_{k \to \infty} x(k)$, con x distribuzione stazionaria e x_i la frazione di tempo che spendo nello stato i in un lungo periodo.

con $x_i \geq 0$ e $\sum_{i=0}^{n} x_i = 1$

Un grafo è strettamente connesso se non ci sono nodi isolati, e in esso esiste un'unica x distribuzione stazionaria (autovettore di λ). Per trovare la x potrei dire che x=Px=(I-P)x=0, quindi ci sono infinite soluzioni, per cui posso sostituire un'equazione di questo sistema con la sommatoria in figura (normalizzazione). Questo metodo però non è applicabile praticamente, ma è solo teorico, perché non funziona per matrici grandi, quindi la si può calcolare in maniera iterativa approssimando il risultato.

$x \approx P^k x(0)$

Metodi Iterativi per la risoluzione di sistemi sparsi di grandi dimensioni

Nei metodi iterativi si calcola la soluzione del sistema come limite di una successione per i sistemi grandi, questo fa si che oltre all'errore di round-off ci sia un errore dovuto al troncamento. Inoltre se A è malcondizionata c'è ugualmente una perdita di cifre significative. Però per sistemi sparsi di grandi dimensioni, non conviene utilizzare l'algoritmo di Gauss poiché questo sporca la matrice andando a riempire di elementi la matrice sparsa tanto più velocemente tanto più grande è la matrice.

I metodi iterativi invece non alterano la struttura della matrice e non modificano gli elementi nulli, inoltre sono utilizzabili facilmente in maniera parallela. Questi generano una successione di soluzioni approssimate di cui bisogna verificare:

- Convergenza: se il metodo converge.
- Velocità: il numero di iterazioni in cui converge.

Gli algoritmi iterativi vanno applicati a matrici sparse di grandi dimensioni, altrimenti non ne sfrutto i vantaggi, perché ogni operazione effettua il prodotto A*x che è costoso se non sfrutto la sparsità.

Splitting

Un primo esempio di metodo iterativo è lo splitting. Data la matrice A del sistema Ax=b, la divide in due, ossia A=P-N con P non singolare. La convergenza dipende dalla matrice di iterazione $B=P^{-1}N$ e se il limite della successone x^k esiste allora è uguale ad x, ossia il metodo è consistente. ρ(B) è il raggio spettrale e più è piccolo, più rapida è la convergenza. La verifica del raggio spettrale però è molto difficile ed è più costosa da risolvere di un sistema lineare. Per questo poiché vale che ρ(B)<=||B||, allora una condizione sufficiente di convergenza è che ||B||<1. Questo non è un test da inserire nel software, ma permette solo di avere una misura dell'errore. Ovviamente se B

$$Ax=b \longleftrightarrow Px=Nx+b$$

$$x^0 = \text{vettore iniziale}$$

$$Px^{k+1}=Nx^k+b \qquad k\geq 0$$

$$x^{k+1} = P^{-1}Nx^k+P^{-1}b = Bx^k+c \qquad k\geq 0$$

Teorema di convergenza

un metodo iterativo è convergente per ogni x^0

$$\rho(B) = \max_i |\lambda_i| < 1$$

Posto $e^k=x^k-x$: errore al passo k

$$x=\lim_{k\to\infty} x^k \iff \lim_{k\to\infty} e^k=0$$

$$x=Bx+c \text{ e } x^k=Bx^{k-1}+c \longrightarrow x-x^k=B(x-x^{k-1})$$

$$e^k=Be^{k-1}=B(Be^{k-2})=B^2e^{k-2}=\ldots=B^ke^0$$

$$\lim_{k\to\infty} e^k=0 \longrightarrow \lim_{k\to\infty} B^k=0 \longrightarrow \rho(B)<1$$

$$\|e^k\| \leq \|B\|^k \|e^0\|$$

$$||x^{k+1}-x^k|| \leq TOL||x^k||$$

$$k \geq NMAX$$

è molto vicino ad 1, la convergenza sarà molto lenta, quindi si inseriscono i criteri di arresto riguardanti la tolleranza ed il numero massimo di iterazioni.

Metodo di Jacobi

Può essere utilizzato solo se la matrice ha tutti gli elementi della diagonale diversi da zero ed è non singolare. Divide la matrice A in tre matrici della sua stessa dimensione:

$$P = D \quad N = -(L+U) = D - A$$

$$B_j = -D^{-1}(L+U) = D^{-1}(D-A) = I - D^{-1}A$$

$$x^{k+1} = B_j\, x^k + D^{-1}b \qquad k \geq 0$$

- L= matrice degli elementi del triangolo inferiore.
- D= matrice degli elementi sulla diagonale.
- U= matrice degli elementi del triangolo superiore.

B_j è quindi ottenuta andando a dividere ogni riga per l'elemento sulla diagonale, x^{k+1} può anche essere scritta in forma scalare come in figura.

$$x_i^{k+1} = \frac{1}{a_{ii}}\left[b_i - \sum_{\substack{j=1 \\ j \# i}}^{n} a_{ij} x_j^k \right]$$

Inoltre

$$\|B_J\|_\infty = \max_i \sum_{\substack{j=1 \\ i \neq j}}^{n} \frac{|a_{ij}|}{|a_{ii}|}$$

$$\|B_j\|_\infty < 1 \quad \text{se} \quad |a_{ii}| > \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}|, i = 1,..n$$

se la matrice A è a diagonale strettamente dominante, il metodo di Jacobi converge. Ovviamente gli elementi della diagonale non devono essere nulli. L'aggiornamento del vettore può essere fatto simultaneamente, e quindi è più efficiente su architetture vettoriali o parallele.

Metodo Gauss Seidel

Per velocizzare la convergenza, non utilizzo solo la soluzione al passo 0, ma includo nel calcolo di x_i^{k+1}, le sue componenti già calcolate, mentre quelle ancora da

calcolare fanno riferimento sempre alla distribuzione iniziale. Affinchè converga, una condizione sufficiente è che A deve essere a diagonale strettamente dominante, simmetrica e definita positiva. Questo metodo può convergere quando Jacobi non converge ed in generale lo fa più velocemente. Ovviamente gli elementi della diagonale non devono essere nulli. L'aggiornamento del vettore deve essere fatto in maniera sequenziale.

$$x_i^{k+1} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^{k} \right]$$

$P = D+L \quad N = -U \quad B_{GS} = -(D+L)^{-1}U$

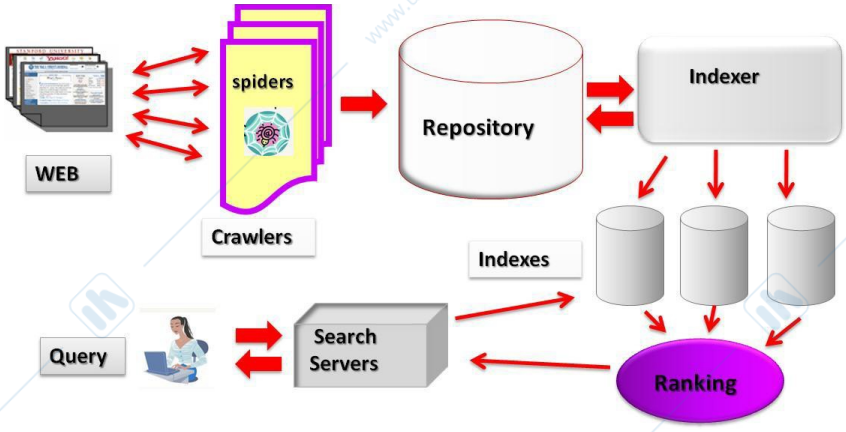
$x^{k+1} = -D^{-1}Lx^{k+1} - D^{-1}Ux^{k} + D^{-1}b \qquad k \geq 0$

La complessità, considerato k il numero di iterazioni, è T=O(n*nz*k).

La scelta quindi del metodo (diretto o iterativo), dipende dalle proprietà della matrice e dalle risorse disponibili. Se la matrice non ha grandi dimensioni conviene utilizzare sempre in matlab lo \.

Algoritmo PageRank di Google

I problemi principali di un web search engine sono i seguenti:



- Memorizzare le informazioni in un archivio.
- Dopo una query fornire i risultati real time.
- Decidere l'ordine in cui mostrare le pagine.

Le fasi di funzionamento quindi sono le seguenti:

- Raccolta informazioni: scansionando testi ed esplorando risorse web.
- Indicizzazione informazioni: catalogando e creando indici per recuperare velocemente le informazioni.
- Ranking risultati: ordinando le pagine pertinenti.

I principali agenti sono:

- Crawler: bot virtuali che prelevano un URL e se già presente la scarta, altrimenti al aggiunge al repository.
- Repository: contiene l'html delle web page.
- Indexer: prende un'informazione dal repository, estrae parole e link, la comprime e genera indici che associano i documenti a dei puntatori e memorizza le informazioni nell'indexes.
- Indexes: conitene il forward index (parole contenute) e l'inverted index (ad ogni parola è associata la lista di pagine che la contengono (hit list)).
- Search server: converte la richiesta in stringa e consente all'inverted index di trovare le pagine rilevanti.
- Ranking: presenta gli indirizzi delle pagine in ordine di rilevanza.

Il ranking è la componente più importante, infatti ci sono moltissime pagine che rispondono ad una richiesta, quindi è fondamentale mostrare le pagine nell'ordine di rilevanza maggiore. Il problema principale è quindi l'affidabilità.

Prima del page rank c'erano due metodi utilizzati:

- Analisi del testo: però era suscettibile allo spamming, ossia bastava inserire nel testo di una pagina molte volte un parola per avere più rilevanza.
- Numeri di link in uscita o in entrata: potevano essere creati fake link per salire di rilevanza.

PageRank

Dopo la indicizzazione e catalogazione delle pagine web, le assegna un voto(rank) che ne definisce l'importanza. Questo viene ricalcolato frequentemente. In pratica una pagina è importante se è puntata da pagine importanti, e distribuisce il suo peso equamente a tutte e pagine cui punta.

$$x_{i=} \sum_{j \in B(i)} \frac{x_j}{N_j}$$ = rank della pagina i

relazione ricorsiva

- B(i) = insieme delle pagine j che puntano ad i
- N_j = numero di link uscenti dalla j-ma pagina

rank = Σ contributi dei rank delle pagine j in B(i)

Il web è gestito come un grafo orientato di dimensioni enormi, in cui le pagine sono i nodi e i link gli archi. Quindi nel page rank un link da A a B è un voto pesato quindi una pagina ha un rank alto se la somma dei rank di link che la puntatno è alta: una pagina con un solo link in ingresso,ma molto importante, avrà un rank più alto di pagine con molti link ma meno importanti.

Viene definita quindi W come l'insieme di n pagine web, e G come la matrice di connettività (matrice di adiacenza enorme e molto sparsa) di W in cui $g_{ij}=1$ se c'è un link dalla pagina j alla pagina i. Nella matrice le colonne rappresentano i link in uscita, e le righe i link in ingresso ad una determinata pagina. Per cui la loro somma costituisce l'outdegree e l'indegree. Vedendo ciò come una catena di Markov, posso individuare la matrice P di

$$p_{ij} = \begin{cases} 1/N_j \ (=g_{ij}/N_j) \text{ se c'è un link da j ad i} \\ 0 \quad \text{altrimenti} \end{cases}$$

$$\sum_{i=1}^{n} p_{i,j} = 1$$

transizione dividendo gli elementi della colonna j di G diversi da zero per il loro numero N_j. Gli elementi di P rappresentano la probabilità di muoversi dalla pagina j alla pagina i seguendo un link. Posso quindi dire che il Rank di una pagina è la distribuzione stazionaria di probabilità (x=Px) che rappresenta il numero di volte che un navigatore, navigando per lungo tempo e passando a caso da una pagina all'altra seguendo i link, visiti una pagina. Ovviamente tutto ciò è possibile solo se la catena è irriducibile.

I problemi da affrontare sono:

- Autoloop: sono i nodi che si autoreferenziano, quindi basta eliminare i link a se stessi ponendo la diagonale di G nulla.
- Dangling node: sono pagine che hanno solo link in ingresso e quindi la loro colonna di G, e quindi di P, è nulla. Ciò indica che P non è una matrice di transizione, quindi collego il nodo a tutti gli altri con rank pari a 1/n, con n pari al numero di pagine.

$$p_{i,j} = \begin{cases} 1/N_j & \text{se c'è un link da j ad i} \\ 1/n & \text{, se } N_j = 0,\ n = \text{numero di pagine} \\ 0 & \text{altrimenti} \end{cases}$$

- Rank sink: è il problema principale in cui c'è un gruppo di pagine che ha un solo link entrante che si referenziano tra di loro creando un loop che va ad aumentare sempre di più il loro rank generando un grafo non fortemente connesso e che quindi non garantisce la convergenza alla probabilità stazionaria. Per evitarlo si utilizza il modello del random surfer, ossia si assegna ad ogni pagina un'uscita di sicurezza a bassa priorità che la collega a tutte le altre, quindi:
  - Si sceglie la pagina con probabilità p normalmente.
  - Si salta ad una pagina a caso con probabilità 1-p.

Il valore di p è stato posto pari a 0.85 perché è stato testato che la convergenza è la più rapida perché non impiega mai più di cento iterazioni.

Fatte queste considerazioni la matrice P cambia, e viene definita la matrice di Google A su cui si applica il page rank. Gli elementi x_i rappresentano la quantità di tempo che un random surfer spenderà su quella pagina. Per cui A è la matrice di transizione di una catena di Markov irriducibile che ha quinid un'unica distribuzione stazionaria x tale che x=Ax -> (I-A)x=0 dove x è il Rank di Google e ordinandolo si ordinano i siti in base alla loro importanza. Questo non è calcolato all'atto della richiesta. La matrice di Google però, per effetto del rank sink, è una matrice piena, quindi è solo un modello teorico e non si può utilizzare praticamente, ma l'unica matrice che si può gestire è G. Utilizziamo quindi un metodo iterativo che sfrutta la struttura di A e la sparsità di G.

$$a_{ij} = \begin{cases} p(1/N_j) + (1-p)/n & \text{se } N_j \neq 0 \\ 1/n & \text{se } N_j = 0 \text{ (dangling node)} \end{cases}$$

$$\sum_{i=1}^{n} a_{i,j} = p\sum_{i=1}^{n}\frac{1}{N_j} + \sum_{i=1}^{n}\frac{1-p}{n} = 1, \qquad N_j \neq 0$$

$$\sum_{i=1}^{n} a_{i,j} = \sum_{i=1}^{n}\frac{1}{n} = 1, \quad N_j = 0 (\textit{colonna nulla})$$

A = matrice di transizione
della catena di Markov associata a G

$$x_i = \sum_j a_{i,j}x_j$$ = rank della pagina i

Posso esprimere la matrice A come: $A=pGD+ez^T$ dove e è il vettore colonna unitario, D è una matrice diagonale e z è un vettore riga. In pratica il

$$a_{ij} = \begin{cases} p(1/N_j) + (1-p)/n = p(g_{i,j}/N_j) + (1-p)/n & , N_j \neq 0 \\ 1/n & , N_j = 0 \text{ (dangling node)} \end{cases}$$

$$D_{jj} = \begin{cases} 1/N_j & N_j \neq 0 \\ 0 & N_j = 0 \end{cases}$$

$$z_j = \begin{cases} (1-p)/n & N_j \neq 0 \\ 1/n & N_j = 0 \end{cases}$$

primo pezzo rappresenta la matrice P, ed il secondo il termine il contributo dovuto al dangling node. Il primo termine è quindi una matrice sparsa e l'unica matrice che determina è G perché D ed e sono vettori, il secondo termine invece è uno scalare. Il metodo iterativo ha quindi una complessità temporale pari a T(n)=O(nnz*n). Il criterio di arresto utilizza una tolleranza che non deve essere troppo elevata, perché non ho bisogno di alta precisione, e il metodo converge rapidamente perché richiede massimo 100 iterazioni, ma dopo 30 i valori già iniziano a stabilizzarsi.

$$x=(pGC+ez^T)x$$

$$x^{k+1} = pGDx^k + e(z^Tx^k)$$

Questo modello è in continua evoluzione perché cambiano e diventano sempre più efficienti le regole per pesare i link.

Interpolazione I (Fitting Dati)

Spesso si dispone solo di un insieme finito di dati che rappresentano un fenomeno continuo, si vuole quindi un modello che descriva il fenomeno in modo attendibile ossia descrivendo e valutando il fenomeno in punti differenti da quelli dati. Un primo approccio si ha con l'interpolazione.

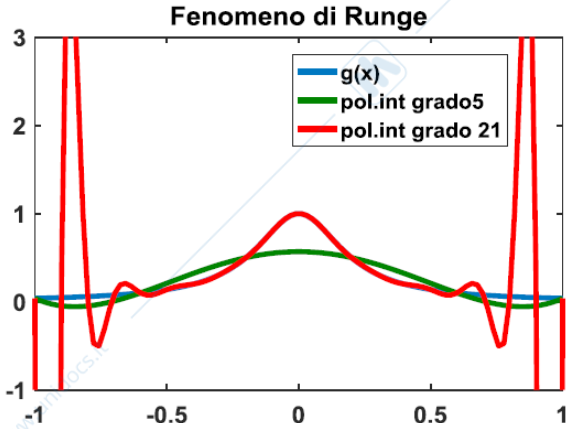
Questa assume che i dati siano esatti e da essi costruisce e valuta una funzione f(x) interpolante che per i punti dati assume valori assegnati. Un problema che si presenta è quello di Lagrange ossia per uno stesso insieme di dati posso avere più funzioni diverse tra cui scegliere, quindi i dati non definiscono una soluzione univoca, ma posso scegliere arbitrariamente la funzione tra:

- Polinomi: la scelta più semplice poichè sono facili da derivare ed integrare, ma presentano problemi di efficienza e accuratezza.
- Polinomi a tratti: accurati ma non regolari
- Spline: la migliore funzione che ha la massima accuratezza ed efficienza.

Polinomi

Dati n punti distinti x_i (ascisse) e n valori y_i devo determinare un polinomio $p_m(x)$ tale che $p_m(x_i)= y_i$. Il grado che deve assumere affinchè sia unico è pari ad n-1, infatti se ho n condizioni allora le incognite sono i coefficienti che sono pari ad n-1. È quindi legato al numero di dati. Un problema è che all'aumentare del grado del polinomio il modello peggiora come descritto dal fenomeno di Runge che dimostra effettivamente ciò. Quindi questo tipo di funzione non è affidabile e conviene utilizzare polinomi di grado basso, ma ciò comporta avere pochi dati e quindi poche informazioni sul fenomeno e quindi non è attendibile. (O(n^2))

fenomeno di Runge
$g(x)=1/(1+25x^2)$ $x\varepsilon$ [-1,1]



Sia $y_i = g(x_i)$, $g \in C^n[x_1,x_n]$

$R_{n-1}=|g(x)-p_{n-1}(x)| = (g^{(n)}(\xi)/n!)(x-x_1)(x-x_2)..(x-x_n)$
$\xi \in (x_1,x_n)$

(non di pratica utilità)

si dimostra che R_{n-1} in molti casi cresce all'aumentare di n

$\lim_{n\to\infty} p_{n-1}(x) \neq g(x)$

Interpolazione polinomiale a tratti

Suddivido l'insieme dei punti [x1,xk] in sottoinsiemi con ascisse consecutive (intervallini lk) e su ogni sottoinsieme costruisco il relativo polinomio interpolante di grado m-1(se di primo grado retta spezzata). Ho quindi polinomi diversi su sottoinsiemi contigui di nodi che hanno un grado basso e quindi il modello è efficiente ed accurato.

Il modello lineare è il più semplice da costruire perché in ogni intervallo la funzione è la retta interpolante i nodi x_i e

x_{i+1}. (O(n) n piccolo). Inoltre all'aumentare di n converge. Il problema principale di quest'interpolazione è che ci possono essere punti di discontinuità dovuti a spigoli.

Sia $y_i = g(x_i)$, $g \in C^2[x_1, x_n]$

$R_n = |g(x) - L_n(x)| \leq (h^2/8)\max(|g''(x)|) = O(h^2)$

$h = \max|x_{i+1} - x_i|$

R_n diminuisce all'aumentare di n

$\lim_{n\to\infty} L_n(x) = g(x)$

convergenza

Interpolazione II (Fitting Dati-Spline)

Le funzioni spline sono polinomi di basso grado e regolari nei nodi nati da problemi reali per minimizzare l'energia potenziale della curva (spline naturale cubica interpolante). In ogni intervallo è un polinomio di grado n e le sue derivate sono continue fino all'ordine m-1, quindi i polinomi in due intervalli diversi, nel nodo in comune sono uguali (no spigoli).

La spline cubica è la più usata nelle applicazioni e in particolare il polinomio è di terzo grado e quindi le derivate fino alla seconda sono continue. Per descriverla ho quindi bisogno di quattro coefficienti in ogni intervallo e quindi di 4(n-1)=4n-4 incognite. Se come condizioni utilizzo solo le n condizioni di interpolazione e le 3(n-2) condizioni di continuità ho solo 4n-6 condizioni e quindi il problema è mal posto. Aggiungo quindi le condizioni aggiuntive al contorno tra cui la più usata è la natural spline. Questa minimizza le oscillazioni tra i nodi e quindi converge più rapidamente (h^4). Questa convergenza si nota anche nelle derivate e al crescere di n il modello

$s''(x_1) = s''(x_n) = 0$
spline cubica naturale (natural spline)

$s'(x_1) = d_1$, $s'(x_n) = d_n$
spline completa (clamped spline)

$s'''(x_2)$, $s'''(x_{n-1})$ continue
spline not-a knot

migliora fino a convergere al fenomeno che ha originato i dati.

Per la costruzione della spline naturale cubica interpolante, non calcolo direttamente i 4(n-1) coefficienti relativi ai singoli intervalli, ma utilizzo le condizioni di interpolazione, di continuità per s' e s'' e al contorno per s''. I coefficienti quindi possono essere espressi in funzione dei valori Mi $=s_i''(x_i)$ che sono solamente n-2:

Sia $y_i = g(x_i)$, $g \in C^4[x_1,x_n]$

$R_n=|g(x)-s(x)| \leq (h^4/384)\max(|g^{IV}(x)|) = O(h^4)$

$h=\max|x_{i+1}-x_i|$

R_n diminuisce all'aumentare di n

$\lim_{n\to\infty} s(x)=g(x)$ convergenza

relazioni analoghe per le derivate fino alla III

- di, bi e ai li ottengo valutando si in xi.
- ci lo ottengo valutando si in xi+1

Per calcolare gli Mi sfrutto la continuità della derivata prima e le condizioni al contorno. (vedere slide dimostrazione)

Fatto ciò quindi ho il risultato in figura in cui il secondo termine è un numero e la matrice non deve essere memorizzata perché le h sono note. (O(n)). La matrice A è tridiagonale simmetrica a diagonale dominante. Si ha quindi che A è ben condizionata a meno che non ci sono nodi vicinissimi o lontanissimi.

$$h_{i-1}M_{i-1}+2(h_{i-1}+h_i)M_i+h_i M_{i+1}=$$
$$=6[(y_{i+1}-y_i)/h_i-(y_i-y_{i-1})/h_{i-1}]$$
$$i=2,\dots,n-1$$

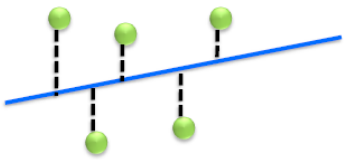
Interpolazione III (Smoothing Dati)

Una curva parametrica rappresenta la traiettoria di un punto P che si muove in un intervallo di tempo, quindi per descrivere il moto di P bisogna definire x(t) e y(t) ognuno dei quali rappresenta un punto P(t). La curva è quindi costituita dai punti P(t). Poiché è impossibile rappresentare queste curve con un'unica interpolante, si utilizza la spline interpolante parametrica. Se bisogna rappresentare curve molto complesse, si memorizzano i punti in ordine in cui si susseguono nella curva e si interpola.

Lo smoothing di dati si utilizza quando i dati sono affetti da errore, in numero elevato e si vuole l'andamento del fenomeno, si vuole estrapolare informazioni al di fuori dell'intervallo e si vuole pesare l'errore. Si vuole quindi ottenere una curva liscia che si scosti poco dai dati ossia che conserva le informazioni esatte dei dati riducendo l'errore e ricostruendo il fenomeno senza errore. Per fare ciò ho bisogno quindi di tutte le informazioni, non solo quelle dell'ultimo intervallo.

Minimi Quadrati Polinomiali

$r_i = y_i - p(x_i)$ **residuo**



La funzione di smoothing è un polinomio di primo grado p(x)=a+bx e lo scostamento è la distanza verticale tra il punto (xi,yi) e la retta p(x). Quindi lo scostamento globale è la somma dei quadrati di questa differenza, per questo p(x) è detta retta di minimi quadrati che minimizza l'errore medio e si avvicina ai dati scegliendo opportunamente i coefficienti a e b. Faccio ciò ponendo le derivate rispetto ad a e b pari a zero ottenendo un sistema lineare.

Se le ascisse sono tutte distinte tra loro, il sistema ha una sola soluzione ed è definita quindi la retta dei minimi quadrati. Questa però non è sempre la soluzione migliore,

ma posso considerare un polinomio di secondo grado $p(x)=a+bx+cx^2$.

Se invece i dati non hanno tutti lo stesso peso, si usa il modello dei minimi quadrati pesati.

$$\sum_{i=1}^{m} w_i(y_i - f(x_i))^2 = minimo$$

Ovviamente si possono costruire polinomi di grado maggiore, ma non è detto che funzionino meglio.

Estrapolazione e previsione

Con le spline di smoothing posso fare previsioni di un fenomeno al di fuori dell'intervallo de dati con una buona precisione.

Interpolazione bidimensionale

Ho un polinomio di primo grado in due variabili $f(x_i,y_i)=z_{ij}$. Dati questi punti in ogni sottoregione rettanfgolare si determina una funzione interpolante del tipo: $p(x,y)=a+bx+cy+dxy$.

Equazioni Differenziali Ordinarie (ODE) I

I metodi analitici per la risoluzione delle equazioni differenziali non sono utilizzabili praticamente. Le equazioni differenziali sono equazioni la cui incognita è una funzione: $y'=f(t,y(t))$. Ammette più soluzioni che sono dipendenti dalla costante arbitraria c. Questo valore corrisponde quindi al valore della soluzione all'istante iniziale. Il problema descritto in questo modo prende il nome di problema differenziale di Cauchy del primo ordine a valori iniziali e descrive fenomeni di

$$y(t) - y(t_0) = \int_{t_0}^{t} y'(t)dt = \int_{t_0}^{t} f\big(t, y(t)\big)dt$$

$$y(t) = \int_{t_0}^{t} f\big(t, y(t)\big)dt + c$$

evoluzione nel tempo. Se ho un'equazione di ordine n ho bisogno di n-1 valori iniziali ottenendo in questo modo un sistema di n equazioni nel primo ordine. Solitamente però la y(x) non è calcolabile e quindi c'è bisogno di modelli numerici.

$$\begin{cases} y_1'(t) = y_2(t) \\ y_2'(t) = y_3(t) \\ \dots \\ y_n'(t) = f(t, y_1(t), y_2(t), \dots, y_n(t)) \\ \\ y_1(t_0) = \alpha_1 \\ y_2(t_0) = \alpha_2 \\ \dots \\ y_n(t_0) = \alpha_n \end{cases}$$

Condizionamento

Il condizionamento analizza come l'errore nei dati(condizione iniziale) si riflette nella soluzione, infatti, commettendo l'errore sul valore iniziale, se lo cambio , cambio la curva soluzione. Se il problema è bencondizionato la distanza tra soluzioni ottenute da condizioni iniziali vicine diminuisce, se il problema è malcondizionato aumenta al crescere di t. In gererale il condizionamento di un problema dipende da un segno.

Ci possono essere casi inoltre in cui il problema è bencondizionato o malcondizionato a seconda dell'intervallo in cui mi trovo, questo perché la derivata della funzione rispetto alla y dipende da t.

Discretizzazione

Consiste nel suddividere l'intervallo continuo in un vettore di punti distanti di un certo passo h=(t-t0)/n. Quindi la nuova soluzione è un vettore di valori yi. Il valore y0 è noto e a partire da esso costruisco un vettore di punti cercando di seguire la curva della soluzione fedelmente. In realtà però i punti successivi non si troveranno su di essa ma su altre curve della famiglia con diversa condizione iniziale. Ad ogni passo quindi si genera una perturbazione che mi fa passare da una curva all'altra la quale si amplifica o riduce a sceonda se il problema è malcondizionato o meno.

Metodo Eulero esplicito

È il metodo di discretizzazione più semplice ma è poco efficiente. Si basa sulla formula di Taylor, infatti considero y(t+h) sviluppandola con Taylor arrestandomi al secondo ordine e trascuro il resto poiché z è ignoto. In pratica sto approssimando la derivata con il rapporto incrementale. Viene detto esplicito perché il valore ad un istante dipende solo dall'istante precedente. Dal punto di vista geometrico traccio la tangente alla curva in ti e prendo come valore della soluzione il punto corrispondente a ti+1 sulla tangente appena disegnata. Questo processo non può essere vettorizzato perché devo completare l'elaborazione precedente per calcolare la successiva. Se devo risolvere invece un sistema di equazioni posso vettorizzare calcolando la soluzione per tutte le equazioni contemporaneamente.

$y(t+h) = y(t)+h\,y'(t) +y''(z)h^2/$

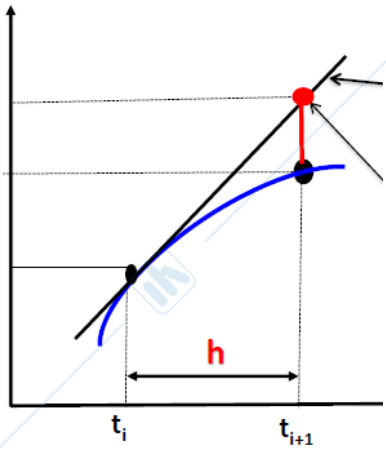
Si trascura

y_{i+1}

$y(t_{i+1})$

$y_{i+1}= y_i+h\,f(t$

$y(t_i)$



I problemi principali da affrontare sono la stabilità (propagazione errori), la convergenza (soluzione approssimata tende alla vera per h che va a zero e con quale velocità) e se si può ottenere la soluzione con un'accuratezza voluta. Per quest'ultimo problema, se si suppone il metodo convergente e stabile, allora è verificata l'ipotesi scegliendo h opportunamente, ossia fissato TOL esiste un h piccolo tale che l'errore è minore di TOL.

$|\,y(t_k)-y_k\,| < TOL$

Per quanto riguarda convergenza e stabilità bisogna valutare l'errore:

- errore locale: è l'errore che si commette al singolo passo e corrisponde alla distanza tra il punto sulla

curva e il corrispondente punto sulla tangente. $e_{k+1}=y(t_{k+1})-y_{k+1}$.

$$e^T_{k+1} = y''(z)h^2/2 = O(h^2)$$

- errore globale: corrisponde all'errore locale di tutti i passi più la loro propagazione. Può quindi essere maggiore o minore dell'errore locale a seconda se il metodo sia instabile o stabile. Questo corrisponde all'errore globale del passo precedente più l'errore locale del passo corrente più un termine dipendente da h. Il fattore |1+hJ| è il fattore di amplificazione e quindi se minore di 1 l'errore diminuisce (stabile), altrimenti il metodo è instabile. Per cui se J è positivo il metodo è sempre instabile. Se J è negativo invece il problema è stabile solo se $h<2/|J|$, quindi più grande è |J| più piccolo deve essere il passo h per garantire la stabilità. $E_{k+1}=y(t_{k+1})-y_{k+1}$.

$$E_{k+1} = y(t_{k+1})-y_{k+1} = E_k + h\,JE_k + e^T_{k+1}$$

$$E_{k+1} = (1+hJ)E_k + e^T_{k+1}$$

- errore round off: dipende dal sistema floating point.

L'ordine di convergenza di questo metodo è p=1, ossia il più basso possibile. Indica quindi che se dimezzo h, l'errore diminuisce di un fattore ½, per cui settando opportunamente h posso avere la convergenza voluta. Per ciò che riguarda l'errore di round off, se questo ad ogni passo è pari ad ε, l'errore totale sarà $O(h+ \varepsilon/h)$, ossia se h è molto piccolo si accumulano gli errori di round off e la soluzione potrebbe divergere di molto. Esiste un h ottimale che è difficile da valutare che corrisponde all'intersezione tra l'errore di troncamento e di round-off.

Il problema principale quindi delle ODE è la determinazione ottimale di h.

Equazioni Differenziali Ordinarie (ODE) II

La complessità computazionale di un metodo si misura in base al numero di valutazioni di una funzione, ma il numero di passi dipende dall'ordine del metodo e dal passo h. Per cui h deve essere grande in modo da ridurre l'errore di round-off, ma deve contemporaneamente soddisfare l'accuratezza richiesta e garantire stabilità. Molto spesso quindi h risulta impraticabile. Per questo si utilizzano metodi con stabilità più ampia o di ordine maggiore.

Eulero Implicito

Si approssima la derivata con il rapporto incrementale all'indietro, $y_{i+1} = y_i + hf(t_{i+1}, y_{i+1})$ ossia il valore nel punto i+1 dipende da se stesso. È molto difficile quindi da calcolare perché ad ogni passo devo risolvere un'equazione non lineare, ma sono sempre stabili (se il problema è bencondizionato). Infatti il fattore di amplificazione è 1/(1-hJ) che è sempre minore di 1 se J è negativo, quindi è stabile per ogni valore di h.

Runge Kutta

Sono metodi di ordine maggiore di 1 in cui l'errore è del tipo $O(h^p)$, ossia $y_{k+1} = y_k + h\,F(t_k, y_k, h)$ dimezzando h l'errore diminuisce di un fattore 2^p. Essi valutano una combinazione lineare dei valori di F in più punti dell'intervallo, sono semplici da implementare. Tra i più utilizzati ci sono il metodo di ordine 2 e il metodo di ordine 4 in cui le condizioni di stabilità sono rispettivamente $h|\lambda|<2$ e $h|\lambda|<2.7853$. Dopo n passi rispetto ad Eulero guadagno più cifre significative.

Possiamo quindi dividere i metodi in:

- espliciti: sono condizionatamente stabili ma la condizione di stabilità può imporre un h troppo piccolo.
- impliciti: sono incondizionatamente stabili, quindi non ho limitazioni su h. Sono più costosi, ma si utilizzano per problemi stiff.

Nel software quindi l'utente fornisce solo ciò che conosce:l'accuratezza desiderata, l'intervallo, la funzione e la tolleranza, ma mai il passo h che è la cosa più difficile da valutare. Il software si arresta quindi quando |Ek|<TOL. Per fare ciò, visto che non conosco la soluzione vera, devo calcolare una stima della soluzione. Tipicamente se io sto risolvendo ad esempio con Runge Kutta, assumo la soluzione vera quella risolta con RungeKutta di ordine 5 e stimo la soluzione per quella di ordine 4 in modo che l'errore sia la loro differenza. Per quanto riguarda la scelta di h ovviamente non posso ottenerla tramite il calcolo dello Jacobiano (oneroso), inoltre può essere che in parti diverse dell'intervallo necessiti di diverse h, per cui lo si sceglie adattivamente e ad ogni passo calcolo l'approssimazione successiva, se verifica il test vado avanti, altrimenti riduco h e calcolo un nuovo errore.

$$|e^T_k| \leq TOL\ h_k$$

Problemi Stiff

Un'equazione differenziale è stiff se rappresenta un processo fisico con scale temporali molto diverse tra loro o tali che la scala temporale è piccola se paragonata all'intervallo di integrazione. Per cui potrei avere in un intervallo un h molto piccolo che genera quindi un errore enorme e tempi di calcolo eccessivi. In generale la soluzione è composta da un termine stazionario che non varia molto con t e un transiente che in un piccolo intervallo ha una grande variazione e poi decade rapidamente a zero. Il transiente influenza la stabilità e lo

stazionario l’accuratezza, quindi posso trovarmi un h molto più piccolo di quello necessario per l’accuratezza. Per questi problemi si utilizzano i metodi impliciti. Non c’è una definizione precisa di ODE stiff, ma c’è ne solo una numerica perché il problema nacque quando si voleva risolvere un’equazione via software, quindi si dice che se un problema è difficile per ode45 e facile per ode15s, il problema è stiff.

Per verificare se TOL è giusta la diminuisco mano mano, se le curve al passo corrente e precedente si sovrappongono, allora la tolleranza è giusta, altrimenti devo ridurre ancora.

Trasformata discreta di Fourier

Consiste nello scomporre un fenomeno composto da più onde per vederne quelle elementari ed analizzarle. Queste strutture sono

$f = (f_0, f_1, \ldots, f_{N-1})$ reale o complesso

DFT(f)

$$F_k = \sum_{i=0}^{N-1} f_j w_N^{-jk}$$

K=0,...N-1

Vettore complesso

$k \sin(2\pi\nu t), k \cos(2\pi\nu t)$ armoniche elementari

ν= frequenza

T=1/ν =periodo

k= ampiezza

$2\pi\nu$= frequenza radiale

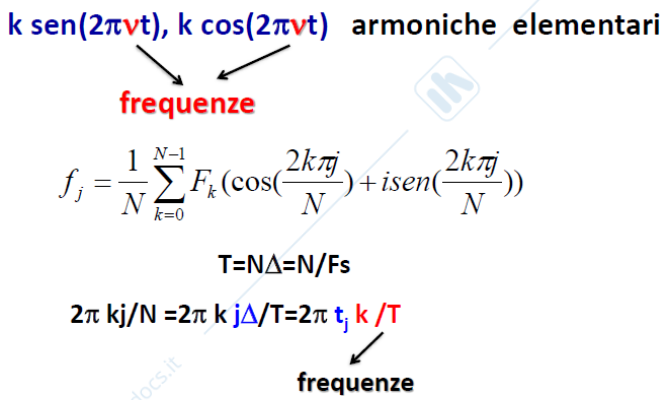
combinazioni di seno e coseno e molte volte hanno un carattere ciclico. L'analisi di Fourier costituisce un ponte tra il dominio del tempo e delle frequenze. Poiché nella maggior parte dei casi i dati sono discreti si utilizza la trasformata discreta di Fourier (DFT); essa trasforma un vettore in un altro dove le componenti sono combinazioni lineari di seni e coseni. Ne esiste la inversa IDFT la quale cambia solo per il fatto che il segno dell'esponente j è positivo. Si intuisce quindi che è ciclica. Le uniche due componenti reali sono la componente F_0 detta componente continua che tipicamente si trascura perché non contiene informazioni, e la componente $F_{N/2}$.

Considerando quindi una funzione periodica di periodo T, ossia g(x+T)=g(x), posso dire che h(x)=g(vx) è periodica di periodo T/v. Posso quindi definire le armoniche elementari come in figura e di esse sono interessato alla frequenza e all'ampiezza.(Analisi spettrale). Se t è in secondi, v è in cicli per secondo, se t è in metri, v è la lunghezza d'onda e 1/v è detto numero d'onda.

Per fare ciò ho bisogno di discretizzare l'intervallo [0 T], quindi scelgo un passo di campionamento Δ ,con il quale campiono N valori, che mi permette quindi di campionare agli istanti $t_j = j\,\Delta$ in modo che $f_j = f(t_j)$. Definisco $T = N\,\Delta$ il

periodo fondamentale e $F_S=N/T=1/\Delta$ la frequenza di campionamento. Se f è periodica lo è anche la sua discretizzazione.

Nel discreto le frequenza che corrispondono alle v nel domino del tempo sono del tipo k/T. Queste sono tutte (tranne la F_0) multiple della frequenza fondamentale $\Delta f=1/T=1/N\Delta=Fs/N$ e



corrisponde alla minima distanza alla quale si devono trovare due frequenze per distinguerle. La frequenza più alta invece è calcolabile con il Teorema di Shannon e viene definita come FN=Fs/2. Se quindi una qualsiasi frequenza in modulo è minore di questo valore, allora essa è completamente determinata dal campionamento, altrimenti bisogna diminuire il passo. Ciò può causare il fenomeno dell'aliasing ossia le frequenza più alte disturbano quelle più basse che vengono approssimate male.

Nel continuo k va da 0 a N-1, mentre nel discreto va da -N/2 a N/2, in realtà però c'è la periodicità delle frequenze dove $F_{N+m}=F_m=F_{m-N}$, quindi le frequenze assumono la suddivisione in figura.

freq > 0	**$1 \leq k \leq N/2-1$**
freq < 0	**$N/2+1 \leq k \leq N-1$**
freq = 0	**k=0**
Nyquist	**k=N/2**

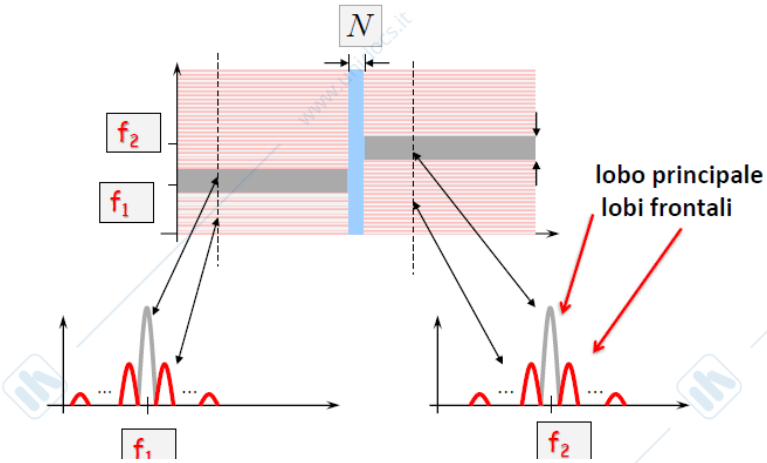
Se il vettore è pari allora esiste la frequenza di Nyquist e si trova al campione N/2+1; altrimenti il campione non esiste. Per il campione F_0 non c'è simmetria.

Tutto ciò viene visualizzato su un periodogramma nel quale si visualizzano

- ampiezza: modulo delle frequenze Fk o potenza Pk=|Fk|^2 la cui somma da l'energia del fenomeno.
- Frequenze: valori k/t dove se f è reale considero solo k da 0 a N/2. L'onda della frequenza più bassa è detta prima armonica e le successive sono numerate di conseguenza.

L'implementazione in Matlab abbassa la complessità da $O(n^2)$ a $O(n\log_2 n)$ con l'idea base di non memorizzare gli indici i,j,k in base 10 , ma in base 2. L'unica differenza è che per ottenere la vera ampiezza bisogna moltiplicare l'uscita per 2/N.

Il filtraggio consiste nella riduzione del rumore cui è affetto un segnale e ricostruire il segnale originale, poiché queste componenti sono invisibili nel dominio del tempo bisogna ricorrere alla trasformata di Fourier, individuare le componenti anomale operando solo da 0 a N/2 per effetto della simmmetria, e azzerarle insieme alle loro simmetriche, posso quindi ricostruire il segnale con la IDFT.



I segnali non stazionari hanno frequenza diverse in tempi diversi, e non avrei strumenti per avere informazioni sull'evoluzione spettrale del segnale nel tempo. Quindi si suppone che il segnale abbia caratteristiche stazionarie in piccoli intervalli e si calcola localmente lo spettro. Per fare ciò devo moltiplicare il segnale in ogni istante per una finestra in modo da isolare le componenti che voglio considerare e ne calcolo la DFT (Short Time Fourier Transform). Posso quindi rappresentare tutto in uno

spettrogramma dove gli assi rappresentano frequenza e tempo. Nella pratica vorrei una perfetta risoluzione nel tempo(N/Fs) e in frequenza (Fs/N), ma la lunghezza della finestra N, influenza entrambe in modo opposto, quindi se N è piccolo, ho migliori intervalli ma non distinguo bene le frequenze e viceversa (principio di incertezza). Le finestre più usate sono la rettangolare, la Hamming e la Blakmann; queste ultime due effettuano l'overlap, ossia i segmenti sono leggermente accavallati di una durata pari alla metà dell'intervallo per ottenere una descrizione graduale e meglio interpolata dello spettro.

Analisi di segnali sonori

Dual Tone Multi-Frequency System(DTMF) è il sistema di codifica di codici numerici in segnali sonori che utilizza la DFT. Ogni tono è la somma di due sinusodi una ad alta e una a bassa frequenza in modo che non siano confondibili. La frequenza di campionamento Fs=8000Hz. La cifra viene riconosciuta qundi individuando le frequenza corrispondenti ai picchi di massima potenza e tramite DFT ottengo la cifra corrispondente.

Un suono sono delle vibrazioni che si propagano in un mezzo percepibili dall'orecchio umano. Sono caratterizzate da:

- Altezza: suoni acuti(alte frequenze) e bassi
- Intensità: ampiezza delle frequenze misurabili in decibel
- Timbro: dipende dalle armoniche presenti

La qualità del suono è determinata dalla frequenza di registrazione. In Matlab sono memorizzati come vettori di numeri.

Parallel MATLAB

Matlab permette di eseguire parallelamente più task in due diverse modalità:

- Sfruttando la CPU andando con il comando parpool a creare un numero di workers al massimo pari al numero di core del calcolatore. Si possono quindi eseguire varie istruzioni ottimizzate per il calcolo parallelo (parfor). La cosa importante è che le istruzioni devono essere indipendenti; inoltre è stesso il MATLAB a far comunicare i vari worker in modo da fornire un output coerente rendendo questa pratica trasparente al programmatore. Ovviamente si ha un guadagno rispetto al caso seriale solo se le dimensioni dei dati sono notevoli.
- GPU: sfrutta i core presenti sulla GPU che è nata per eseguire numerose operazioni matematiche contemporaneamente. Inoltre permette anche di eseuire in parallelo senza dover imparare il linguaggio di programmazione CUDA. Per fare ciò dopo aver elaborato i dati nella GPU, bisogna esportarli nuovamente nella memoria della CPU.