FATTORIZZAZIONE QR
TEORIA ED ESEMPI APPLICATIVI

Marco Cappicciola

Sommario

Nella presente trattazione viene affrontato uno dei problemi tipici relativi al trattamento di dati sperimentali, normalmente affetti da errore. Nel caso in esame è stato implementato un metodo, quello dei minimi quadrati, sviluppato dalle sue basi. L'analisi del metodo parte perciò dalla fattorizzazione QR, algoritmo scelto per la risoluzione dei sistemi sovradimensionati, tipici dell'approssimazione di funzioni mediante polinomi, passando poi alla metodologia per risolvere sistemi di tale tipo, per finire con lo sfruttare tale possibilità. Questo lavoro è corredata di sorgenti tramite i quali sono stati implementati gli algoritmi sviluppati, nonchè da una sezione di test, mediante i quali si è verificata la capacità di compiere le operazioni richieste unitamente all'evidenziazione del carico computazionale necessario.

Indice

1 Introduzione

Nell'analisi e nell'utilizzo di dati sperimentali si ha normalmente a che fare con misure affette da errori, anche se le relazioni che legano le grandezze in gioco sono note con precisione. Ciò significa, tra le altre cose, che un problema considerante le suddette relazioni non è il problema che si vorrebbe in realtà risolvere:

$$A\mathbf{x} = \mathbf{b} \rightarrow A(\mathbf{x} + \delta\mathbf{x}) = (\mathbf{b} + \delta\mathbf{b}) \qquad (1)$$

in cui il vettore $\delta\mathbf{b}$ è proprio l'errore che può essere attribuito, per esempio, alle grandezze misurate, mentre $\delta\mathbf{x}$ è la perturbazione che ne deriva sulla soluzione del problema in oggetto.

In tal modo quindi si può scrivere, utilizzando una norma matriciale consistente:

$$A\delta\mathbf{x} = \delta\mathbf{b}$$

e:

$$\|\delta\mathbf{x}\| \leq \|A^{-1}\| \cdot \|\delta\mathbf{b}\|$$

In questo modo, definendo *numero di condizionamento* di una matrice la quantità:

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

si ricava:

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} = \kappa(A) \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \qquad (2)$$

Si noti come l'errore relativo della soluzione dipenda fortemente dagli errori presenti nei termini noti del problema, secondo un fattore di amplificazione che dipende dalla natura stessa del problema attraverso la matrice A.

E' evidente quindi come diventi utile se non necessario avere a disposizione più re-

lazioni di quante siano effettivamente le incognite del problema, in modo da poter, in qualche modo, attenuare l'effetto che gli errori sperimentali hanno sulla soluzione che è possibile calcolare.

Un problema in cui il numero delle equazioni supera quello delle incognite, cioè un problema detto *sovradeterminato*, non può essere, in generale, risolto in senso classico quando i dati siano affetti da errore. In un caso simile il sistema potrebbe non avere soluzioni, e dunque non è possibile utilizzare, come metodo di risuluzione, quello di Gauss ovvero mediante fattorizzazione LU.

Si parla allora di *problemi ai minimi quadrati*, e si utilizzano metodi in grado di trovare una soluzione per cui **s**ia minimo lo scarto quadratico medio tra il primo e secondo membro del sistema:

$$\min_{x\in\mathbb{R}^n} \|A\mathbf{x}-\mathbf{b}\|_2 \tag{3}$$

Uno di questi metodi prevede la fattorizzazione della matrice (rettangolare) del sistema in esame mediante due matrici, delle quali una è triangolare superiore e l'altra è ortogonale: tale fattorizzazione è detta QR.

La capacità di risolvere un sistema sovradeterminato si traduce quindi nella possibilità di approssimare leggi sperimentali quando i dati noti siano affetti da errori.

2 Fattorizzazione *QR*

Ogni matrice A $m \times n$, sia essa quadrata o rettangolare, può essere fattorizzata nella forma:

$$A = QR$$

in cui la matrice Q, di dimensioni $m \times m$, è ortogonale, mentre R, delle stesse dimensioni di A, è una matrice triangolare superiore.

Realizzando una fattorizzazione di questo tipo, la matrice A può essere sostituita dal prodotto QR:

$$A\mathbf{x} = \mathbf{b} \rightarrow QR\mathbf{x} = \mathbf{b}$$

Poichè $Q^TQ = QQ^T = I$ si può scrivere:

$$Q^TQR\mathbf{x} = Q^T\mathbf{b}$$

e perciò:

$$R\mathbf{x} = Q^T\mathbf{b} = c$$

Il problema di risolvere un sistema lineare si traduce perciò nel trovare la soluzione di un problema del tipo:

$$\begin{cases} Q\mathbf{c} = \mathbf{b} \\ R\mathbf{x} = \mathbf{c} \end{cases} \tag{4}$$

E' possibile dimostrare peraltro che, se $A = QR$, allora:

- $\|A\|_2 = \|R\|_2$
- $\kappa_2(A) = \kappa_2(R)$

Ciò significa che il sistema iniziale è stato trasformato in un sistema equivalente, triangolare superiore, tale per cui il numero di condizionamento sia rimasto immutato.

2.1 Il metodo di Householder

2.1.1 La matrice H

E' possibile costruire una matrice elementare del tipo:

$$H = I - 2\mathbf{w}\mathbf{w}^T \tag{5}$$

in cui $\mathbf{w} \in \mathbb{R}^n$ e $\|\mathbf{w}\| = 1$. Tale matrice è simmetrica e ortogonale. E' possibile determinare il vettore $\mathbf{w}$ tale per cui sia:

$$H\mathbf{x} = k\mathbf{e}_1 \tag{6}$$

con $k \in \mathbb{R}$ e $\mathbf{e}_1$ il primo versore della base canonica di $\mathbb{R}^n$. Poichè H è ortogonale:

$$\|H\mathbf{x}\| = \|\mathbf{x}\| = |k| = \sigma$$

e applicando la definizione di H, e poichè $\|\mathbf{w}\| = 1$, si ottiene:

$$\mathbf{w} = \frac{\mathbf{x}-k\mathbf{e}_1}{2\mathbf{w}^T\mathbf{x}} = \frac{\mathbf{x}-k\mathbf{e}_1}{\|\mathbf{x}-k\mathbf{e}_1\|} \tag{7}$$

in cui:

$$\|\mathbf{x}-k\mathbf{e}_1\| = \sqrt{2\sigma(\sigma+|x_1|)}$$

La conoscenza di questo vettore permette di costruire la fattorizzazione *QR* di una matrice *A*.

2.1.2 Fattorizzazione di Householder

E' possibile scrivere una qualunque matrice *A* come:

$$A = [\mathbf{a}_1\ \mathbf{a}_2\ \dots\ \mathbf{a}_n]$$

Perciò, sfruttando le matrici elementari di Householder precedentemente definite, è possibile calcolare H_1 tale che:

$$H_1\mathbf{a}_1 = k_1\mathbf{e}_1$$

Applicando a sinistra di *A* tale matrice si ottiene una seconda matrice:

$$A^{(2)} = H_1A^{(1)} = \left[\mathbf{a}_1^{(2)}\ \mathbf{a}_2^{(2)}\ \dots\ \mathbf{a}_n^{(2)}\right]$$

in cui ora:

$$\mathbf{a}_1^{(2)} = k_1\mathbf{e}_1$$

Di conseguenza la matrice appena ottenuta ha la struttura:

$$A^{(2)} = \begin{bmatrix} k_1 & \mathbf{v}_1^T \\ \mathbf{0} & \hat{A}^{(2)} \end{bmatrix}$$

Reiterando il procedimento per la sottomatrice $\hat{A}^{(2)}$ si otterrà una matrice elementare $\hat{H}$ di dimensione minore. Tale matrice sarà orlata mediante le righe e le colonne di una matrice identità per ottenerne una del tipo:

$$H^{(2)} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \hat{H}^{(2)} \end{bmatrix}$$

Tale matrice verrà rimoltiplicata per $A^{(2)}$ per ottenere $A^{(3)}$. Perciò, al generico passo *i*, si avrà:

$$A^{(i)} = \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ \mathbf{0} & \hat{A}^{(i)} \end{bmatrix} \tag{8}$$

e quindi, costruita la nuova matrice $\hat{H}_i$, si avrà:

$$\begin{aligned} A^{(i+1)} &= \begin{bmatrix} I_{i-1} & \mathbf{0}^T \\ \mathbf{0} & \hat{H}_i \end{bmatrix} \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ \mathbf{0} & \hat{A}^{(i)} \end{bmatrix} \\ &= \begin{bmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ \mathbf{0} & \hat{H}_i\hat{A}^{(i)} \end{bmatrix} \end{aligned} \tag{9}$$

Al termine dell'algoritmo si ha che l'ultima matrice ottenuta è evidentemente una matrice triangolare superiore. Sicchè:

$$\begin{aligned} R &= A^{(n)} = H_{n-1}A^{(n-1)} = \dots \\ &= H_{n-1}H_{n-2}\cdots H_1A^{(1)} \end{aligned}$$

Poichè $Q = H_1H_2\cdots H_{n-1}$ è ortogonale in quanto prodotto di matrici ortogonali, la relazione appena scritta implica:

$$A = QR$$

Nel caso in cui la matrice A sia quadrata l'algoritmo si arresta dopo $n-1$ passi; se invece la matrice è rettangolare, come nel caso di studio del presente lavoro, l'algoritmo si arresta al passo *n*, poichè tale ulteriore passo è necessario per azzerare gli elementi dell'ultima colonna di *A*.

2.2 Il metodo di Givens

2.2.1 La matrice G

In modo simile al metodo appena descritto, è possibile costruire un altro tipo di matrice elementare, del tipo:

$$G_{ij} = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & \cdots & s & & \\ & & \vdots & & \vdots & & \\ & & -s & \cdots & c & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}$$

in cui $c^2 + s^2 = 1$. Una matrice di questo tipo opera una rotazione piana in $\mathbb{R}^n$. Applicata

ad un vettore $\mathbf{y} = G_{ij}\mathbf{x}$ permette di ottenere:

$$y_k = \begin{cases} cx_i + sx_j & \text{per } k = i \\ -sx_i + cx_j & \text{per } k = j \\ x_k & \text{per } k \neq i, j \end{cases}$$

Da queste relazioni si capisce come si possano determinare i parametri c ed s in modo che la componente j-esima del vettore $\mathbf{x}$ venga annullata, la componente i-esima venga modificata e tutte le altre restino invariate. Si ricava perciò:

$$s = \frac{x_j}{\sqrt{x_i^2 + x_j^2}} \qquad \text{e} \qquad c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}$$

2.2.2 Fattorizzazione di Givens

L'algoritmo di fattorizzazione che si esaminerà consiste nell'applicare ripetutamente matrici elementari di Givens alla matrice iniziale A, aventi i coefficienti c ed s calcolati in modo da azzerare sistematicamente gli elementi del triangolo inferiore di A.

Quel che si ottiene da questa fattorizzazione è:

$$G_{n,m} \cdots G_{13} G_{12} A = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

nel caso in cui la matrice A avesse dimensione $m \times n$. Si può ora porre:

$$Q^T = G_{n,m} \cdots G_{13} G_{12}$$

In cui Q è una matrice ortogonale in quanto prodotto di matrici ortogonali. Ciò implica del resto:

$$A = QR$$

che è proprio il risultato che si voleva ottenere.

Un metodo di questo tipo, rispetto alla fattorizzazione di Householder, richiede in generale il doppio delle operazioni in virgola mobile. Del resto però risulta molto più efficiente nel caso in cui si abbia a che fare con matrici sparse, poichè le matrici di Givens tali da annullare gli elementi già nulli non sono necessarie.

3 Problemi ai minimi quadrati

Si è già detto come spesso si renda necessaria la capacità di trovare la soluzione di un problema in generale *mal posto*. L'esempio portato riguarda il caso di un sistema **sovradeterminato**, in cui cioè sono disponibili più equazioni che incognite. In tal caso, dato il problema:

$$A\mathbf{x} = \mathbf{b}$$

si avrà A matrice rettangolare di dimensione $m \times n$, $\mathbf{b} \in \mathbb{R}^m$ mentre $\mathbf{x} \in \mathbb{R}^n$.

Essendo il problema mal posto non si potrà avere una soluzione in senso classico. In particolare, se i dati a disposizione fossero esatti, allora $m - n$ equazioni del sistema sarebbero linearmente dipendenti dalle altre, sebbene non si possa facilmente dire quali. Se viceversa i dati fossero affetti da errore, potrebbe capitare che nessuna equazione possa essere verificata esattamente, rendendo auspicabile l'avere a disposizione un numero sovrabbondante di equazione per ridurre l'influsso degli errori sperimentali.

Portando avanti questo ragionamento, è lecito cercare di riformulare in un problema ben posto quello originale. Si potrebbe ad esempio richedere che lo scarto quadratico medio tra primo e secondo membro sia minimo:

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2$$

Un problema di questo tipo è detto *ai minimi quadrati*, la cui soluzione coincide con:

- il problema originale nel caso in cui questo minimo sia nullo;
- la soluzione nel senso dei minimi quadrati in caso contrario.

3.1 Utilizzo della fattorizzazione QR

Tra gli altri, un metodo efficace per risolvere un problema di questo tipo è quello di sfruttare la fattorizzazione QR. Infatti,

considerando il quadrato del residuo:

$$\begin{aligned} \|A\mathbf{x}-\mathbf{b}\|^2 &= \|QR\mathbf{x}-\mathbf{b}\|^2 \\ &= \|Q\left(R\mathbf{x}-Q^T\mathbf{b}\right)\|^2 \\ &= \|R\mathbf{x}-\mathbf{c}\|^2 \end{aligned} \quad (10)$$

Per ottenere questo risultato è stata sfruttata la proprietà di una matrice ortogonale di non modificare la norma-2 di un vettore. In più, è stato posto $\mathbf{c} = Q^T\mathbf{b}$.

In tal modo, poichè A è rettangolare, anche R è rettangolare; precisamente:

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

Si potrà partizionare in modo coerente anche il vettore $\mathbf{c} \in \mathbb{R}^m$ in:

$$\mathbf{c} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} \quad , \quad \mathbf{c}_1 \in \mathbb{R}^n \quad , \quad \mathbf{c}_2 \in \mathbb{R}^{m-n}$$

A questo punto, tornando alla relazione 10:

$$\begin{aligned} \|R\mathbf{x}-\mathbf{c}\|^2 &= \| \begin{bmatrix} R_1\mathbf{x} \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} \|^2 \\ &= \|R_1\mathbf{x}-\mathbf{c}_1\|^2 + \|\mathbf{c}_2\|^2 \end{aligned}$$

Nel caso in cui sia $\|R_1\| \neq 0$ il sistema:

$$R_1\mathbf{x} = \mathbf{c}_1$$

ammette una e una sola soluzione, ed in corrispondenza di questa soluzione si ha che:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x}-\mathbf{b}\| = \|\mathbf{c}_2\|$$

Nel caso in cui il vettore $\mathbf{c}_2$ sia nullo, allora $\mathbf{x}$ è la soluzione classica del sistema iniziale. Viceversa essa è la soluzione nel senso dei minimi quadrati.

4 Approssimazione

Ci si pone ora l'obiettivo di calcolare il polinomio di **migliore approssimazione**. Ciò consiste nel determinare il polinomio di grado n tale da minimizzare una qualunque norma dell'errore. Ciò significa:

$$\min_{p_n\in\Pi_n} \|p_n - f\| \quad (11)$$

Utilizzando la norma-∞ si ottiene la migliore approssimazione uniforme, mentre utilizzando la norma-2:

$$\|p_n - f\|_2 = \sqrt{\int_a^b [p_n(x) - f(x)]^2 \, dx}$$

si ottiene la migliore approssimazione **nel senso dei minimi quadrati**. E' evidente che per trattare un problema di questo tipo ci si deve ricondurre al caso di sistema sovradeterminato, poichè si è in grado di risolverlo.

Passando al caso discreto infatti, la norma appena enunciata diventa:

$$\|p_n - f\|_2 = \sqrt{\sum_{i=0}^{m} [p_n(x_i) - y_i]^2}$$

in cui $\{x_0, x_1, \ldots, x_m\}$ sono le ascisse degli $m+1$ punti per i quali sono noti i valori $\{y_0, y_1, \ldots, y_m\}$ della funzione $f(x)$. Si noti che tali valori possono o meno essere affetti da errore.

Per risolvere il problema si cercherà di minimizzare il quadrato della norma, poichè i due casi sono equivalenti. Si noti inoltre come, se $m = n$ allora la soluzione coincide con il polinomio interpolante.

Utilizzando la base canonica si ottiene:

$$p_n(x_i) = \sum_{j=0}^{n} a_j x_i^j = (X\mathbf{a})_i$$

in cui $\mathbf{a} \in \mathbb{R}^{n+1}$ è il vettore dei coefficienti del polinomio, mentre X è la matrice di Vandermonde:

$$X^{(m+1)\times(n+1)} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{bmatrix}$$

E' chiaro quindi che il problema sarà:

$$\|p_n - f\|_2^2 = \|X\mathbf{a} - \mathbf{y}\|_2^2$$

che è esattamente il genere di problema già trattato. Utilizzando la fattorizzazione QR si può risalire facilmente al polinomio di

migliore approssimazione.

E' evidente come in questo caso valgano le medesime considerazioni fatte in precedenza in relazione al residuo $\|\mathbf{c}_2\|$. Infatti nel caso in cui tale residuo sia nullo, allora il polinomio di migliore approssimazione *coincide* con il polinomio interpolante.

5 Implementazione

Gli algoritmi necessari per realizzare la fattorizzazione QR, così come l'approssimazione di una funzione, sono stati realizzati mediante il linguaggio di programmazione **c++**.

E' stata compiuta una scelta di questo tipo perchè tale linguaggio garantisce un'ottima elasticità ed espandibilità; il che si traduce, nel caso in esame, nella possibilità di semplificare porzioni di codice attraverso un overloading degli operatori tra matrici, nonchè il conteggio delle operazioni in virgola mobile effettuate.

5.1 Strutture dati

Per ottenere i risultati discussi nel presente lavoro si è reso necessario costruire una struttura dati adatta allo scopo. E' stato quindi creato un oggetto composto da più elementi. Come mostrato in tabella, all'interno dell'oggetto `matrix` sono presenti tre elementi: i primi due, `m` e `n`, sono due valori interi, e sono rispettivamente il numero di righe e di colonne della matrice che l'oggetto rappresenta. L'elemento `array` è invece un puntatore ad un vettore di elementi di tipo `newdoub`, descritto più avanti. Tale vettore, di lunghezza $m * n$, contiene tutti i valori degli elementi della matrice.

matrix	
m	int
n	int
array	*newdoub
i(int, int, newdoub)	
newdoub o(int, int)	

Tabella 1: *Oggetto* `matrix`

Per poter accedere agli elementi della matrice mediante riga e colonna, si è reso necessario realizzare due funzioni di interfaccia:

```
i(int, int, newdoub)
```

che permette l'inserimento dei dati, e

```
newdoub o(int, int)
```

che permette di leggere i valori memorizzati.

5.1.1 Il tipo `newdoub`

Poichè si è rivelato molto complicato compiere un calcolo reale delle operazioni in virgola mobile eseguite dagli algoritmi, si è pensato di riscrivere un nuovo tipo di dato rappresentativo di un numero di tipo `double`, legando però a questo nuovo oggetto una funzione chiamata ogni qualvolta venga utilizzato nel corso degli algoritmi. In tal modo il conto viene fatto in modo del tutto trasparente, ed istante per istante il risultato del conteggio è accessibile mediante un puntatore.

5.2 L'overloading

L'overloading è una tecnica di programmazione tipica di certi linguaggi evoluti. E' stata proprio questa caratteristica a far decidere l'utilizzo del c++ quale strumento per implementare gli algoritmi in esame.

Normalmente gli operatori come addizione, moltiplicazione o assegnazione sono in grado di eseguire operazioni solo su tipi di dato standard, quali sono per esempio gli interi o i numeri in virgola mobile. Per esempio, per eseguire il prodotto di due matrici sarebbe necessario costruire, *ogni volta* che ce ne sia la necessità, due cicli annidati tali da compiere il prodotto righe per colonne. Realizzando però un'overload dell'operatore prodotto, è possibile evitare questo procedimento ogni qual volta si renda necessario una moltiplicazione matriciale, stabilendo una volta per tutte il comportamento nel caso i tipi di dato utilizzati siano matrici.

Si noti che durante l'esecuzione dell'algoritmo il calcolo del prodotto viene effettivamente eseguito ogni volta, sicchè questo

procedimento non rappresenta un vantaggio in termini di numero di operazioni: è però un utile ausilio perchè il programmatore possa fissare l'attenzione in modo più efficace sul funzionamento dell'algoritmo.

5.3 Fattorizzazione di Householder

Per fattorizzare una matrice con il metodo di Householder sono state realizzate due funzioni. La prima si occupa di calcolare la matrice elementare ricevendo in ingresso un vettore della matrice A. La seconda esegue effettivamente la fattorizzazione.

5.3.1 Calcolo di $\hat{H} = \hat{H}(\mathbf{x})$

```
sigma = matNorm2(x);
if (x->o(1,1) > 0)
   k = sigma * (-1);
else
   k = sigma;
c = newdoub(1) / (sigma * (sigma + abs(x->o(1,1))));
x->i(1,1, x->o(1,1)-k);


for (int i=1; i<=x->m; i++) {
  for (int j=1; j<i; j++) {
      H->i(i,j, x->o(i,1) * x->o(j,1));
      H->i(j,i, H->o(i,j));
  }
  H->i(i,i, x->o(i,1)*x->o(i,1));
}
*H = *H * (-c);
for (int i=1; i<=H->n; i++)
   H->i(i,i,H->o(i,i)+1);
```

Si noti come viene sfruttata la proprietà di simmetria della matrice $\mathbf{x}\mathbf{x}^T$.

5.3.2 Fattorizzazione

```
  for (int i=1; i<=Q->m; i++)
   for (int j=1; j<=Q->n; j++)
     if (i==j)
       Q->i(i,i,1);
     else
       Q->i(i,j,0);

 max = (A->m == A->n ? A->n-1 : A->n);

 for (int i=1; i<=max; i++) {

   Helem = new matrix<T>(A->m -i+1);
   x = new matrix<T>(A->m -i+1, 1);
   for (int j=i; j<=A->m; j++)
      x->i(j-i+1,1,A->o(j,i));
   householder(x,Helem);

   Mtemp = new matrix<T>(A->m,A->m -i+1);
   for (int j=1; j<=A->m; j++)
     for (int k=i; k<=A->m; k++)
       Mtemp->i(j,k-i+1,Q->o(j,k));
   *Mtemp = *Mtemp * *Helem;
   for (int j=1; j<=A->m; j++)
     for (int k=i; k<=A->m; k++)
       Q->i(j,k,Mtemp->o(j,k-i+1));
   delete Mtemp;

   Mtemp = new matrix<T>(A->m -i+1);
   for (int j=i; j<=A->m; j++)
     for (int k=i; k<=A->n; k++)
       Mtemp->i(j-i+1,k-i+1,A->o(j,k));
   *Mtemp = *Helem * *Mtemp;
   for (int j=i; j<=A->m; j++)
     for (int k=i; k<=A->n; k++)
       A->i(j,k,Mtemp->o(j-i+1,k-i+1));
   delete Mtemp;


   delete Helem;
   delete x;


 }
```

Si noti come, per diminuire il costo computazionale, i prodotti matriciali non vengano realizzati tra le matrici A ed H, bensì tra $\hat{A}$ ed $\hat{H}$ e come poi il risultato venga orlato con gli elementi della matrice calcolata all'iterazione precedente.

Un altro punto degno di nota è il calcolo relativo al termine delle iterazioni. Nel caso in cui la matrice sia quadrata infatti l'algoritmo si arresterà dopo $n-1$ iterazioni. Viceversa sarà necessaria una iterazione aggiuntiva.

5.4 Fattorizzazione di Givens

Anche in questo caso sono state realizzate due funzioni. La prima si occupa di calcolare i parametri della rotazione di Givens, mentre la seconda li sfrutta per azzerare l'elemento considerato nella matrice A.

5.4.1 Calcolo di c ed s

```
if (xj == 0) {
   *c = 1;
   *s = 0;
}
else if (abs(xj) > abs(xi)) {
   t = xi/xj;
   z = sqrt(t*t + 1);
   *s = newdoub(1)/z;
   *c = t* *s;
}
else  {
   t = xj/xi;
   z = sqrt(t*t + 1);
   *c = newdoub(1)/z;
   *s = t* *c;
}
```

Si nota come entra effettivamente in gioco la capacità dell'algoritmo di Givens di diminuire il costo computazionale in presenza di matrici sparse. Infatti nel caso in cui l'elemento da annullare sia già zero non vengono effettuati calcoli.

5.4.2 Fattorizzazione

```
for (int i=1; i<=Q->m; i++)
  for (int j=1; j<=Q->n; j++)
    if (i==j)
      Q->i(i,i,1);
    else
      Q->i(i,j,0);

   for (int k=1; k<=A->n; k++)
     for (int i=k+1; i<=A->m; i++) {

        givens(A->o(k,k), A->o(i,k), &c, &s);
        if (!(abs(A->o(i,k))<eps)) {
          for (int j=k; j<=A->n; j++) {
            t = c*A->o(k,j) + s*A->o(i,j);
            A->i(i,j, c*A->o(i,j) - s*A->o(k,j) );
            A->i(k,j,t);
          }

          for (int j=1; j<=Q->n; j++) {
            t = c*Q->o(k,j) + s*Q->o(i,j);
            Q->i(i,j, c*Q->o(i,j) - s*Q->o(k,j) );
            Q->i(k,j,t);
          }
        }
     }

     matTrasp(Q);
```

Per diminuire il costo computazionale nelle applicazioni della matrice G_{ij} vengono calcolati solo gli elementi che effettivamente vengono modificati.

5.5 Minimi quadrati

Per la risoluzione dei problemi ai minimi quadrati è stata realizzata una funzione che accetta in ingresso una matrice rettangolare $A^{m\times n}$ e un vettore di termini noti $\mathbf{b} \in \mathbb{R}^m$, e restituisce un vettore $\mathbf{x} \in \mathbb{R}^n$ che rappresenta la soluzione nel senso dei minimi quadrati. Il residuo $\|\mathbf{c}_2\|$ è contenuto nella variabile `residuo`.

```
  *r = *a;
  fattQR_G(r,q);
  matTrasp(q);
  *c = *q * *b;

  delete q;

  c1 = new matrix<T>(a->n,1);
  c2 = new matrix<T>(a->m-a->m,1);
  r1 = new matrix<T>(a->n);

  for (int i=1; i<=c1->m; i++)
    c1->i(i,1,c->o(i,1));
  for (int i=1; i<=c2->m; i++)
    c2->i(i,1,c->o(i+c1->m,1));
  for (int i=1; i<=r1->m; i++)
    for (int j=1; j<=r1->n; j++)
      r1->i(i,j,r->o(i,j));

  *residuo = matNorm2(c2);

  delete r;
  delete c;
  delete c2;

  sysTriU(r1,c1,x);

  delete r1;
  delete c1;
```

Come si può notare, la complessità computazionale è relativa essenzialmente alla fattorizzazione QR, poichè è necessario, oltre a ciò, calcolare un prodotto matrice-vettore e calcolare la norma di un vettore. Il resto del codice è funzionale alla partizione della matrice R e del vettore $\mathbf{c}$.

Per finire, l'algoritmo di fattorizzazione scelto è quello di Givens, ma ovviamente non comporterebbe nessuna complicazione aggiuntiva l'utilizzo dell'algoritmo di Householder, dal momento che il sistema è realizzato in modo modulare.

5.6 Approssimazione

Una volta che le funzioni appena descritte siano operative, realizzare l'approssimazione di una funzione nel senso dei minimi quadrati diventa un'operazione molto semplice. Tale funzione accetta in ingresso il vettore dei nodi $\mathbf{x}$, il vettore dei valori $\mathbf{y}$. Restituisce il vettore dei coefficienti $\mathbf{a}$ nonchè il valore del polinomio approssimante nei nodi $\mathbf{z}$. In più, nel caso fosse necessario, restituisce anche una stima del residuo.

```
  for (int i=0; i<=x->m; i++)
    for (int j=0; j<=a->m; j++)
      if (j==0)
        X->i(i+1,j+1,1);
      else
        X->i(i+1,j+1, X->o(i+1,j+1-1) * x->o(i+1,1));

  sysRett(X,y,a,res);
  *z = *X * *a;

  delete X;
```

Come si può vedere, gran parte del codice è necessario solamente per calcolare la matrice di Vandermonde e per calcolare il prodotto $X\mathbf{a}$. Il resto del carico computazionale è imputabile alla risoluzione del sistema rettangolare $X\mathbf{a} = \mathbf{y}$.

6 Test

Per valutare la bontà degli algoritmi sviluppati sono stati effettuati diversi test relativi agli argomenti trattati. Alcuni di essi hanno avuto come scopo il calcolo del carico computazionale nonchè il grado di errore riscontrato. Per altri si è voluto semplicemente evidenziarne il funzionamento.

6.1 Fattorizzazione QR

Per valutare le capacità dei due algoritmi sviluppati, sono stati eseguiti diversi test su matrici di diverso tipo. Entrambi comunque vertevano sul calcolo del numero di operazioni in virgola mobile e sull'errore riportato nella fattorizzazione.

6.1.1 Matrici random quadrate

Il test è stato effettuato su matrici quadrate aventi $2 \leq n \leq 100$. La fattorizzazione è stata effettuata mediante l'algoritmo di Householder e mediante quello di Givens. I risultati sono rappresentati nelle figure 1, 2 e 3, in cui è rappresentato sia l'errore $||A - QR||_2$ che $||Q^T Q - I||_2$.

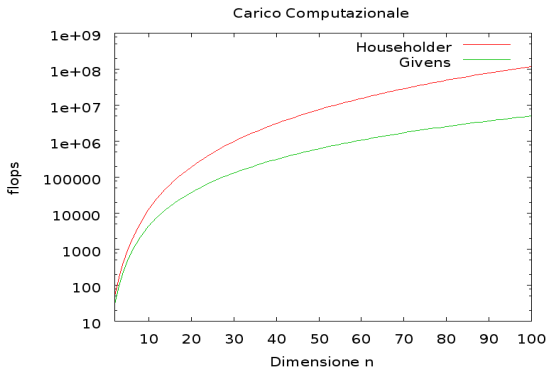

Figura 1: *Householder vs Givens - flops*

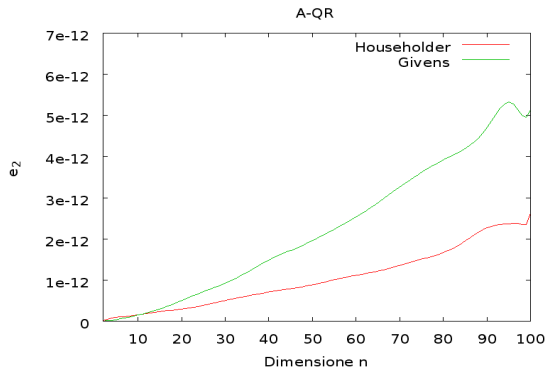

Figura 2: *Householder vs Givens* - $||A - QR||_2$

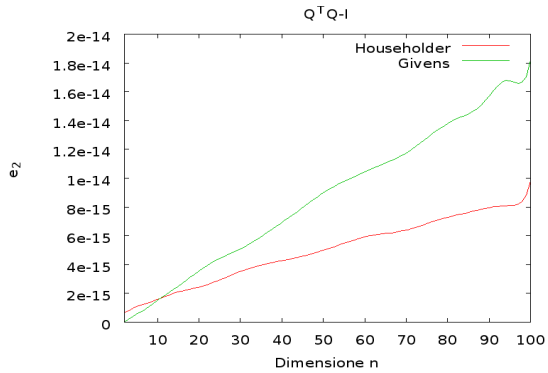

Figura 3: *Householder vs Givens* - $||Q^T Q - I||_2$

Come si può osservare, nelle implementazioni realizzate, l'algoritmo di Givens risulta essere meno oneroso per ciò che riguarda il carico computazionale, benchè utilizzando l'algoritmo di Householder gli errori crescano in modo meno rapido.

6.1.2 Matrici random sparse

Per verificare che effettivamente l'algoritmo di Givens riesca a trarre beneficio dalla presenza di un numero consistente di elementi nulli, sono state effettuate prove su un sistema 20×20 in cui a variare è la percentuale di elementi nulli. I risultati sono rappresentati in figura 4, in cui sono evidenziati i valori ottenuti (punti rossi): si nota come effettivamente al crescere di tale percentuale le operazioni necessarie alla fattorizzazione diminuiscano considerevolmente.

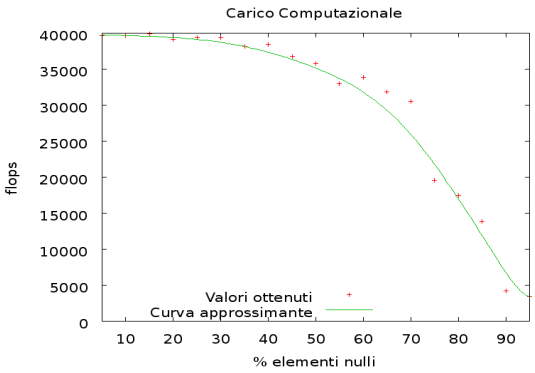

Figura 4: *Sistema Sparse, 20x20*

6.1.3 Matrici di Hilbert

Un'ultima prova per mostrare l'utilizzo degli algoritmi sviluppati consiste nel fattorizzare QR matrici di Hilbert di dimensione crescente. Una matrice di Hilbert è del tipo:

$$B = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{4} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

L'elemento generico è:

$$B_{ij} = \frac{1}{i+j-1}$$

Una matrice di questo tipo è fortemente malcondizionata, il che la rende particolarmente difficile da trattare nei problemi numerici.

Si è provata la fattorizzazione per matrici di Hilbert di dimensione $2 \leq n \leq 100$, con i risultati visibili in figura 5.

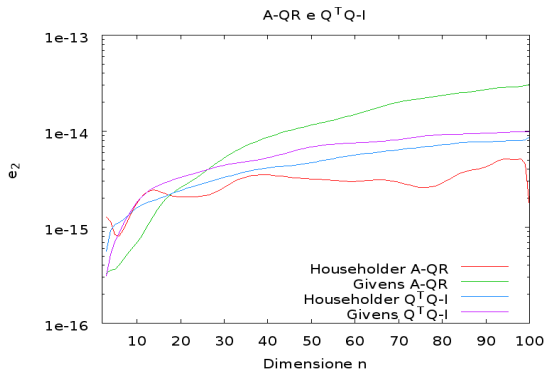

Figura 5: *Errori nella fattorizzazione delle matrici di Hilbert*

6.2 Sistemi rettangolari

Si vuole ora dare qualche esempio di sistema rettangolare, e di come questo tipo di problema venga risolto mediante la fattorizzazione QR. Verrà inoltre affrontato uno studio numerico relativo alla risoluzione di un sistema del tipo $Ax = b$ in cui A è una matrice di dimensione $n \times \frac{n}{2}$ e x vettore unitario.

6.2.1 Sistema con $\|\mathbf{c}_2\| = 0$

Il sistema in esame è costituito da n incognite e da m equazioni di cui $m-n$ linearmente indipendenti. Si vedrà come il metodo calcoli la soluzione esatta e come effettivamente, a meno degli errori di arrotondamento, anche il residuo sia nullo. La matrice A dell'esempio è:

$$A = \begin{bmatrix} -7 & -9 & 7 & 4 \\ 5 & -1 & 6 & -7 \\ -2 & 9 & -6 & 7 \\ 0 & -2 & -7 & -1 \\ -7 & 9 & 4 & 2 \\ 6 & 7 & 5 & 0 \end{bmatrix}$$

Mentre il vettore soluzione e il vettore dei termini noti:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} 12 \\ -7 \\ -30 \\ -21 \\ 32 \\ 35 \end{bmatrix}$$

Dopo la fattorizzazione e i successivi calcoli si ottiene un vettore $\mathbf{c}$ così partizionato:

$$\mathbf{c}_1 = \begin{bmatrix} 3.35 \\ 12.4 \\ -48.7 \\ -35.3 \end{bmatrix} \quad , \quad \mathbf{c}_2 = [1.88e-15]$$

Ed una matrice R_1:

$$R_1 = \begin{bmatrix} -12.2 & -2.29 & 0.0816 & 4.98 \\ 0 & 17.1 & -2.21 & -3.78 \\ 0 & 0 & -14.4 & -1.41 \\ 0 & 0 & 0 & -8.83 \end{bmatrix}$$

Il residuo calcolato è pari a $1.88e-15$.

6.2.2 Sistema con $\|\mathbf{c}_2\| \neq 0$

Il sistema in esame è costituito da n incognite e da m equazioni, tutte linearmente indipendenti. Il metodo calcolerà la soluzione nel senso dei minimi quadrati, riportando effettivamente una stima del residuo non nulla.

$$A = \begin{bmatrix} -6 & 2 & -7 & 3 \\ 6 & -8 & 5 & 7 \\ -4 & -6 & -10 & -9 \\ 9 & -7 & -5 & 8 \\ -6 & -4 & 3 & -2 \\ 8 & 9 & 2 & 2 \end{bmatrix}$$

Il vettore soluzione e quello dei termini noti sono:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} -11 \\ 33 \\ -82 \\ 12 \\ -13 \\ 40 \end{bmatrix}$$

Il vettore rappresentativo dell'errore sperimentale e il termine noto effettivamente utilizzato:

$$\delta\mathbf{b} = \begin{bmatrix} 1.07 \\ 1.07 \\ 0.933 \\ 1.1 \\ 1.03 \\ 1.1 \end{bmatrix} \quad , \quad \hat{\mathbf{b}} = \begin{bmatrix} -9.93 \\ 34.1 \\ -81.1 \\ 13.1 \\ -12 \\ 41.1 \end{bmatrix}$$

Il vettore $\mathbf{c}$ viene calcolato e partizionato come:

$$\mathbf{c}_1 = \begin{bmatrix} -67.5 \\ -33.7 \\ -48.4 \\ 42.4 \end{bmatrix} \quad , \quad \mathbf{c}_2 = \begin{bmatrix} 0.00314 \end{bmatrix}$$

La matrice R_1 risultante è:

$$A = \begin{bmatrix} -16.4 & 0.183 & -3.96 & -9.76 \\ 0 & -15.8 & -3.02 & 1.53 \\ 0 & 0 & -13.7 & -2.04 \\ 0 & 0 & 0 & 10.5 \end{bmatrix}$$

Perciò il vettore soluzione calcolato differisce da quello assegnato:

$$\hat{\mathbf{x}} = \begin{bmatrix} 1.01 \\ 1.96 \\ 2.93 \\ 4.06 \end{bmatrix}$$

Ciò è evidenziato dal residuo calcolato non nullo: $\|\mathbf{c}_2\| = 0.00314$.

6.2.3 Errore sulla soluzione di un sistema rettangolare

E' stato realizzato un ulteriore test relativo alla risoluzione di sistemi lineari rettangolari. In particolare i sistemi rettangolari, del tipo

$$Ax = b$$

esaminati hanno dimensione $n \times \frac{n}{2}$, con $2 \leq n \leq 200$. La soluzione è assegnata e vale:

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \end{bmatrix}$$

I sistemi sono stati risolti utilizzando la fattorizzazione di Givens, ed i risultati sono riportati in figura 6.

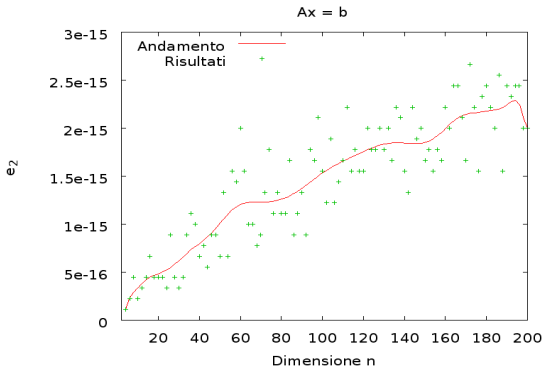

Figura 6: *Sistemi* $n \times \frac{n}{2}$

6.3 Approssimazione

Per verificare che il metodo riuscisse effettivamente a calcolare una approssimazione della funzione in ingresso, lo si è applicato a tre casi distinti. Nel primo caso la funzione approssimata è $f(x) = x^3$. Nel secondo caso alla medesima funzione viene sovrapposto un valore ± 5 atto a simulare l'errore sperimentale.

Infine tramite il terzo caso si vuole mostrare come un polinomio di grado sufficientemente elevato è in grado di offrire una buona approssimazione di una funzione come:

$$f(x) = e^{-x} \cos \pi x$$

6.3.1 $f(x) = x^3$

L'esempio portato mostra come l'algoritmo cerchi di approssimare una funzione, in que-

sto caso una cubica, mediante polinomi con $n = 2,3,4$. Si vede come non riesca a ottenere un risultato soddisfacente con $n = 2$, mentre i polinomi p_3 e p_4 approssimino perfettamente la funzione x^3.

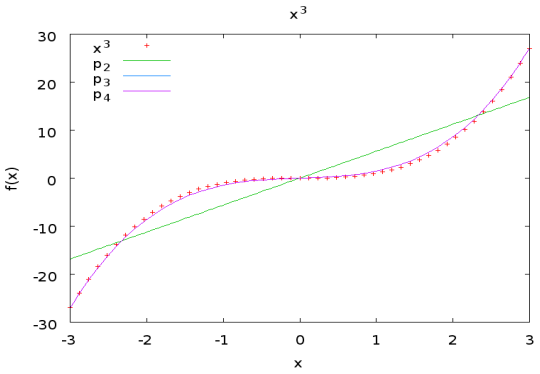

Figura 7: *Approssimazione di x^3 mediante p_2 , p_3, p_4*

6.3.2 $f(x) = x^3 \pm 5$

Una variazione del test precedente prevede la sovrapposizione, rispetto alla funzione x^3 di un valore casuale compreso tra ± 5 in modo da simulare una sorta di errore sperimentale. Come era lecito aspettarsi il polinomio p_2 non è in grado di avvicinarsi alla soluzione, mentre p_3 e p_4 riescono ad approssimare bene la cubica nonostante i valori della funzione non siano quelli esatti.

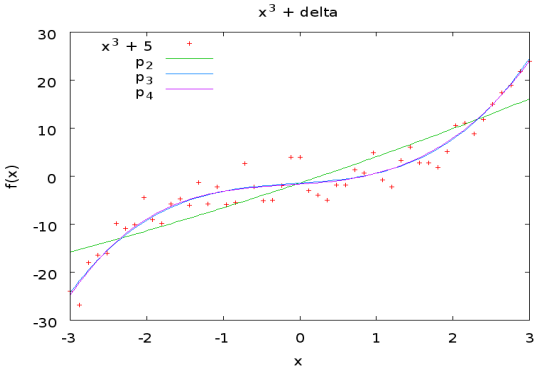

Figura 8: *Approssimazione di $x^3 \pm 5$ mediante p_2 , p_3, p_4*

6.3.3 $f(x) = e^{-x}\cos \pi x$

Come ultimo esempio si è applicato il metodo ad una funzione non polinomiale. Nell'intervallo considerato $[-3;+3]$ il polinomio p_{10} ha dimostrato una buona capacità di approssimazione della funzione in esame. Polinomi di grado inferiore non si sono dimostrati in grado di seguire con la giusta velocità le oscillazioni tipiche della funzione coseno.

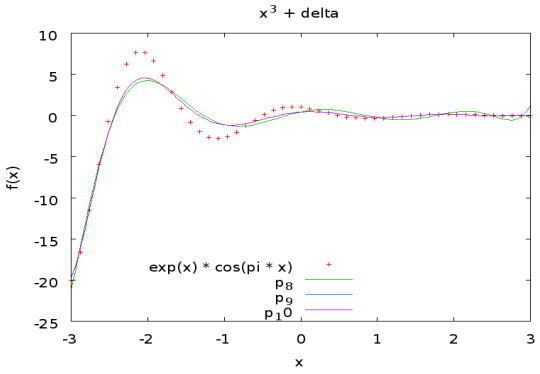

Figura 9: *Approssimazione di $e^x \cos \pi x$ mediante p_8 , p_9, p_{10}*

Riferimenti bibliografici

[1] A.Quarteroni R.Sacco F.Saleri. *Matematica Numerica*. Springer, 2000.

[2] Giuseppe Rodriguez. Dispense di calcolo numerico 2. *Università degli Studi di Cagliari*, 2006.

[3] Deitel & Deitel. *C++ Fondamenti di programmazione*. APOGEO, 2001.