

Metodo di Jacobi

Definizione (Metodo di Jacobi, 1846)

Sia $A = D - E - F \in \mathbb{R}^{n \times n}$ dove

- D è una matrice diagonale, non singolare
- E è triangolare inferiore con elementi diagonali nulli,
- F è triangolare superiore con elementi diagonali nulli.

Sia fissato $x^{(0)} \in \mathbb{R}^n$. Il **metodo di Jacobi** genera $\{x^{(k)}\}_k$ con

$$x^{(k+1)} = \phi(x^{(k)}) = M^{-1}Nx^{(k)} + M^{-1}b$$

dove

$$M = D, \quad N = E + F.$$

Si vede che la matrice di iterazione $P = M^{-1}N$ è in questo caso

$$P = I - D^{-1}A. \quad (1)$$

Metodo di Jacobi in Matlab

Un codice gratuito del metodo di Jacobi, è [jacobi.m](#):

```
function [x,error,iter,flag]=jacobi(A,x,b,max_it,tol)
% input
% A, REAL matrix
% x, REAL initial guess vector
% b, REAL right hand side vector
% max_it, INTEGER maximum number of iterations
% tol, REAL error tolerance
% output
% x, REAL solution vector
% error, REAL error norm
% iter, INTEGER number of iterations performed
% flag, INTEGER: 0 = solution found to tolerance
%   1 = no convergence given max_it
iter = 0; flag = 0; % initialization
bnrm2 = norm( b );
if ( bnrm2 == 0.0 ), bnrm2 = 1.0; end
r = b - A*x;
error = norm( r ) / bnrm2;
if ( error < tol ) return, end
[m,n]=size(A);
[ M, N ] = split( A , b, 1.0, 1 ); % matrix splitting
for iter = 1:max_it, % begin iteration
    x_1 = x;
    x = M \ ( N*x + b ); % update approximation
    error = norm( x - x_1 ) / norm( x ); % compute error
    if ( error <= tol ), break, end % check convergence
end
flag = 1; % no convergence
```

Definizione (Successive over relaxation, (SOR), 1884-1950)

Sia $A = D - E - F \in \mathbb{R}^{n \times n}$ dove

- D è una matrice diagonale, non singolare
- E è triangolare inferiore con elementi diagonali nulli,
- F è triangolare superiore con elementi diagonali nulli.

Sia fissato $x^{(0)} \in \mathbb{R}^n$ e $\omega \in \mathbb{R} \setminus 0$. Il **metodo SOR** genera $\{x^{(k)}\}_k$ con

$$x^{(k+1)} = \phi(x^{(k)}) = M^{-1}Nx^{(k)} + M^{-1}b$$

dove

$$M = \frac{D}{\omega} - E, \quad N = \left(\frac{1}{\omega} - 1\right)D + F$$

Nota.

Gauss-Seidel è SOR in cui si è scelto $\omega = 1$.

Metodo di SOR in Matlab

Una implementazione in Matlab/Octave è [sor.m](#)

```
function [x, error, iter, flag] = sor(A, x, b, w, max_it, tol)
% sor.m solves the linear system Ax=b using the
% Successive Over-Relaxation Method (Gauss-Seidel method when omega = 1 ).
% input
% A, REAL matrix
% x REAL initial guess vector
% b REAL right hand side vector
% w REAL relaxation scalar
% max_it INTEGER maximum number of iterations
% tol REAL error tolerance
% output
% x REAL solution vector
% error REAL error norm
% iter INTEGER number of iterations performed
% flag INTEGER: 0 = solution found to tolerance, 1 = no convergence given max_it
flag = 0; iter = 0;% initialization
bnrm2 = norm( b );
if ( bnrm2 == 0.0 ), bnrm2 = 1.0; end
r = b - A*x;
error = norm( r ) / bnrm2;
if ( error < tol ) return, end
[ M, N, b ] = split( A, b, w, 2 ); % matrix splitting
for iter = 1:max_it % begin iteration
    x_1 = x; x = M \ ( N*x + b ); % update approximation
    error = norm( x - x_1 ) / norm( x ); % compute error
    if ( error <= tol ), break, end % check convergence
end
b = b / w; % restore rhs
flag = 1; % no convergence
```

Routine split

La routine `split` chiamata da `jacobi` e `sor`, tratta da Netlib

```
function [ M, N, b ] = split( A, b, w, flag )
% split.m sets up the matrix splitting for the stat.
% iterative methods:jacobi and sor (gauss-seidel, w=1)
% input
% A DOUBLE PRECISION matrix
% b DOUBLE PRECISION right hand side vector (for SOR)
% w DOUBLE PRECISION relaxation scalar
% flag INTEGER flag for method: 1 = jacobi 2 = sor.
% output
% M DOUBLE PRECISION matrix
% N DOUBLE PRECISION matrix such that A = M - N
% b DOUBLE PRECISION rhs vector ( altered for SOR )
[m,n] = size( A );
if ( flag == 1 ), % jacobi splitting
    M = diag(diag(A)); N = diag(diag(A)) - A;
elseif ( flag == 2 ), % sor/gauss-seidel splitting
    b = w * b;
    M = w * tril( A, -1 ) + diag(diag( A ));
    N = -w * triu( A, 1 ) + ( 1.0 - w ) * diag(diag(A));
end;
```

Esercizio in Matlab I

Esercizio

- *Si calcoli la matrice simmetrica e definita positiva minij_{20} di ordine 20 aiutandosi con `help gallery`.*
- *Sia b il vettore composto di componenti uguali a 1, avente lo stesso numero di righe di P_{20} .*
 - *Si risolva col metodo di Jacobi il problema $\text{minij}_{20}x = b$, con tolleranza di $10^{(-6)}$, partendo da $x^0 = [0 \dots 0]$. Converge?*
 - *Si risolva col metodo di SOR con $\omega = 0.01 : 0.01 : 1.99$ il problema $\text{minij}_{20}x = b$, con tolleranza di $10^{(-6)}$, partendo da $x^0 = [0 \dots 0]$.*
 - *Converge?*
 - *Eseguire il plot in scala semilogaritmica, avendo in ascisse ω e in ordinate il numero di iterazioni eseguite. Quale sembra un buon parametro da utilizzare?*
 - *Calcolare il parametro ω che minimizza il numero di iterazioni svolte da SOR.*

Il metodo del gradiente coniugato.

Definizione (Gradiente coniugato, Hestenes-Stiefel, 1952)

Il metodo del **gradiente coniugato** è un metodo di discesa

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$$

con

- $\alpha_k = \frac{(r^{(k)})^T r^{(k)}}{(p^{(k)})^T A p^{(k)}},$
- $p^{(0)} = r^{(0)},$
- $p^{(k)} = r^{(k)} + \beta_k p^{(k-1)}, \quad k = 1, 2, \dots$
- $\beta_k = \frac{(r^{(k)})^T r^{(k)}}{(r^{(k-1)})^T r^{(k-1)}} = \frac{\|r^{(k)}\|_2^2}{\|r^{(k-1)}\|_2^2}.$

Con questa scelta si prova che $p^{(k)}$ e $p^{(k-1)}$ sono *A-coniugati*.

$$(p^{(k)})^T A p^{(k-1)} = 0.$$

Gradiente coniugato in Matlab

Un codice gratuito del Grad. Coniugato, è [cg.m](#) tratto da Netlib:

```
function [x,error,iter,flag]=cg(A,x,b,M,max_it,tol)
flag = 0; iter = 0; bnorm2 = norm( b );
if ( bnorm2 == 0.0 ), bnorm2 = 1.0; end
r = b - A*x; error = norm( r ) / bnorm2;
if ( error < tol ) return, end
for iter = 1:max_it
    z = M \ r; rho = (r'*z);
    if ( iter > 1 )
        beta = rho / rho_1; p = z + beta*p;
    else
        p = z;
    end
    q = A*p; alpha = rho / (p'*q); x = x + alpha * p;
    r = r - alpha*q; error = norm( r ) / bnorm2;
    if ( error <= tol ), break, end
    rho_1 = rho;
end
flag = 1;
```

Gradiente coniugato in Matlab

Matlab possiede una versione built-in del gradiente coniugato dal nome `pcg`.

```
>> help pcg
PCG    Preconditioned Conjugate Gradients Method.
X = PCG(A,B) attempts to solve the system of linear equations A*X=B for
X. The N-by-N coefficient matrix A must be symmetric and positive
definite and the right hand side column vector B must have length N.
...
X = PCG(A,B,TOL) specifies the tolerance of the method. If TOL is []
then PCG uses the default, 1e-6.
...
[X,FLAG,RELRES,ITER,RESVEC] = PCG(A,B,...) also returns a vector of the
residual norms at each iteration including NORM(B-A*X0).
>> % ESEMPIO
>> A=gallery('minij',5); b=ones(5,1); tol=10^(-6);
>> [X,FLAG,RELRES,ITER,RESVEC] = pcg(A,b,tol);
>> ITER
ITER =
     5
>>
```

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Consideriamo il sistema lineare $Ax = b$ dove A è la matrice tridiagonale a blocchi (di Poisson)

$$A = \begin{pmatrix} B & -I & 0 & \dots & 0 \\ -I & B & -I & \dots & 0 \\ 0 & -I & B & \dots & \dots \\ 0 & \dots & \dots & \dots & -I \\ 0 & 0 & \dots & -I & B \end{pmatrix}$$

con

$$B = \begin{pmatrix} 4 & -1 & 0 & \dots & 0 \\ -1 & 4 & -1 & \dots & 0 \\ 0 & -1 & 4 & \dots & \dots \\ 0 & \dots & \dots & \dots & -1 \\ 0 & 0 & \dots & -1 & 4 \end{pmatrix}$$

Esercizio svolto: $Ax = b$ con A matrice di Poisson

La matrice A è facilmente disponibile, con il comando `gallery` di Matlab. Vediamo un esempio:

```
>> A=gallery('poisson',3); % A sparse.  
>> A=full(A); % A piena.  
>> A  
A =  
     4     -1      0     -1      0      0      0      0      0  
    -1      4     -1      0     -1      0      0      0      0  
     0     -1      4      0      0     -1      0      0      0  
    -1      0      0      4     -1      0     -1      0      0  
     0     -1      0     -1      4     -1      0     -1      0  
     0      0     -1      0     -1      4      0      0     -1  
     0      0      0     -1      0      0      4     -1      0  
     0      0      0      0     -1      0     -1      4     -1  
     0      0      0      0      0     -1      0     -1      4  
>>
```

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Evidentemente A è una matrice di Poisson con B matrice quadrata di ordine 3, dove

$$A = \begin{pmatrix} B & -I & 0 & \dots & 0 \\ -I & B & -I & \dots & 0 \\ 0 & -I & B & \dots & \dots \\ 0 & \dots & \dots & \dots & -I \\ 0 & 0 & \dots & -I & B \end{pmatrix}$$

in cui

$$B = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}$$

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Per ulteriori dettagli sulle origini della matrice di Poisson, si considerino ad esempio [1, p. 557], [2, p. 283], [3, p. 334].

Le matrici di Poisson sono

- simmetriche;
- tridiagonali a blocchi;
- diagonalmente dominanti;
- non singolari (deriva dal primo e dal secondo teorema di Gerschgorin [2, p. 76-80], [3, p. 955]);
- definite positive.

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Per accertarsene, calcoliamo il minimo autovalore della matrice di Poisson con $B \in \mathcal{M}_5$, semplicemente digitando sulla shell di Matlab-Octave

```
>> A=gallery('poisson',5);  
>> m=min(eig(A))  
m =  
    0.5359  
>> % La matrice di Poisson non e' malcondizionata.  
>> condest(A)  
ans =  
    20.7692  
>>
```

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Ora

- poniamo $b = (1, 1, \dots, 1, 1)$ cioè

```
b=ones(size(A,1),1);
```

- risolviamo il sistema $Ax = b$ digitando

```
x_sol=A\b;
```

- nota la soluzione esatta confrontiamo i vari metodi risolvendo il sistema lineare con
 - un numero massimo di iterazioni `maxit` pari a 200,
 - una tolleranza `tol` di 10^{-8}

cioè

```
maxit=200; tol=10^(-8);
```

Esercizio svolto: $Ax = b$ con A matrice di Poisson

A tal proposito consideriamo l'm-file `demo_poisson` contenente il codice

```
maxit=200; tol=10^(-8); siz=5;
A = gallery('poisson',siz);      % MATRICE DI POISSON.
b=ones(size(A,1),1);           % TERMINE NOTO.
x_sol=A\b;                      % SOLUZIONE ESATTA. METODO LU.
x=zeros(size(b));               % VALORE INIZIALE.
% JACOBI.
[x_j,error_j,iter_j,flag_j]=jacobi(A,x,b,maxit,tol);
fprintf('\t \n [JACOBI ][STEP REL., NORMA 2]:%2.2e [REL.ERR.]:%2.2e',error_j,
        norm(x_j-x_sol)/norm_x_sol);
fprintf('\t \n [ITER.]:%3.0f [FLAG]:%1.0f \n',iter_j,flag_j);
% GAUSS-SEIDEL.
w=1;
[x_gs,error_gs,iter_gs,flag_gs]=sor(A,x,b,w,maxit,tol);
fprintf('\t \n [GS ][STEP REL., NORMA 2]:%2.2e [REL.ERR.]:%2.2e',error_gs,norm(
        x_gs-x_sol)/norm_x_sol);
fprintf('\t \n [ITER.]:%3.0f [FLAG]:%1.0f \n',iter_gs,flag_gs);
```

Esercizio svolto: $Ax = b$ con A matrice di Poisson

```
% SOR.
w_vett=0.8:0.025:2;
for index=1:length(w_vett)
    w=w_vett(index);
    [x_sor,error_sor(index),iter_sor(index),flag_sor(index)] = sor(A,x,b,w,maxit
        ,tol);
    relerr(index)=norm(x_sor-x_sol)/norm_x_sol;
end
[min_iter_sor, min_index]=min(iter_sor);
fprintf('\t \n [SOR OTT.] [STEP REL.,NORMA 2]:%2.2e [REL.ERR.]:%2.2e',error_sor(
    min_index),relerr(min_index));
fprintf('\t \n [ITER.]:%3.0f [FLAG]:%1.0f [w]:%2.3f \n',min_iter_sor,flag_sor(
    min_index),w_vett(min_index));
plot(w_vett,iter_sor,'r-');
% GRADIENTE CONIUGATO.
M=eye(size(A));
[x_gc,error_gc,iter_gc,flag_gc]=pcg(A,x,b,M,maxit,tol);
fprintf('\t \n [GC][STEP REL., NORMA 2]:%2.2e [REL.ERR.]:%2.2e',error_gc,norm(
    x_gc-x_sol)/norm_x_sol);
fprintf('\t \n [ITER.]:%3.0f [FLAG]:%1.0f \n',iter_gc,flag_gc);
```

Nota.

Per gli utenti di Octave, si utilizzi [demo_poisson_octave](#) dopo aver scaricato [makefish.m](#)

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Lanciamo la demo nella shell di Matlab e otteniamo

```
>> demo_poisson
[JACOBI ] [STEP REL., NORMA 2]: 8.73e-09 [REL.ERR.]: 5.65e-08
          [ITER.]: 116 [FLAG]: 1

[GAU.SEI.] [STEP REL., NORMA 2]: 9.22e-09 [REL.ERR.]: 2.76e-08
           [ITER.]: 61 [FLAG]: 1

[SOR OTT.] [STEP REL., NORMA 2]: 2.31e-09 [REL.ERR.]: 1.10e-09
           [ITER.]: 21 [FLAG]: 1 [w]: 1.350

[GRA.CON.] [STEP REL., NORMA 2]: 1.04e-17 [REL.ERR.]: 2.95e-16
           [ITER.]: 5 [FLAG]: 1
>>
```

Grafico SOR

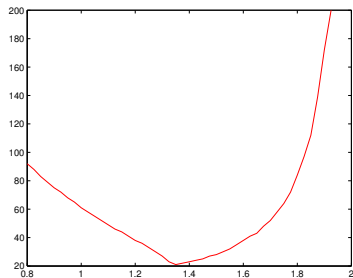


Figura : Matrice di Poisson (ordine 5) e grafico SOR.

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Una breve analisi ci dice che

- Come previsto dalla teoria, il metodo di Gauss-Seidel converge in approssimativamente metà iterazioni di Jacobi;
- Il metodo SOR ha quale costante quasi ottimale $w = 1.350$;
- Il metodo del gradiente coniugato converge in meno iterazioni rispetto agli altri metodi (solo 5 iterazioni, ma si osservi il test d'arresto differente). Essendo la matrice di Poisson di ordine 25, in effetti ciò accade in meno di 25 iterazioni come previsto. Vediamo cosa succede dopo 25 iterazioni. Il residuo relativo, seppur non nullo risulterà molto piccolo.

Esercizio svolto: $Ax = b$ con A matrice di Poisson

```
% OCTAVE
% >> maxit=25;tol=0;
% >> A=makefish(5); b=ones(size(A,1),1);
% >> x=zeros(size(b)); M=eye(size(A));
% >> [x_gc,error_gc,iter_gc,flag_gc]=cg(A,x,b,M,maxit,tol);
% >> error_gc =
%      8.3759e-39
% MATLAB
>> maxit=25;tol=0;
>> A=gallery('poisson',5);
>> A=full(A); b=ones(size(A,1),1);
>> x=zeros(size(b)); M=eye(size(A));
>> [x,flag,relres,iter]=pcg(A,b,tol,maxit,[],[]);
>> relres
relres =
      8.6910e-16
>>
```

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Un punto delicato riguarda la **scelta del parametro ω ottimale** ω^* (cioè minimizzante il raggio spettrale di SOR).

Nel nostro codice abbiamo calcolato per forza bruta ω^+ , tra i numeri reali $\omega^+ \leq 2$ del tipo $w_j = 0.8 + j \cdot 0.025$ quello per cui venivano compiute meno iterazioni.

E' possibile **determinare ω^* esplicitamente** nel caso della matrice di Poisson la risposta è affermativa. Da [3, Teor.5.10, p.333]

$$\omega^* = \frac{2}{1 + \sqrt{1 - \rho^2(B_J)}}$$

dove $\rho(B_J)$ è il massimo degli autovalori in modulo della matrice B_J la matrice di iterazione del metodo di Jacobi.

Il **raggio spettrale** della matrice di iterazione di SOR ott. vale $\omega^* - 1$.

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Vediamo di calcolare questo valore nel caso della sopracitata matrice di Poisson. Dalla teoria, con ovvie notazioni,

$$B_J = I - D^{-1}A$$

e quindi

```
>> format long;
>> D=diag(diag(A));
>> BJ=eye(size(A))-inv(D)*A;
>> s=eigs(BJ); % SOLO I 6 AUTVL. PIU' GRANDI IN MODULO!
>> s_abs=abs(s);
>> rho=max(s_abs);
>> w=2/(1+sqrt(1-rho^2))
w =
    1.333333333333333
>> maxit=50; tol=10^(-8);
>> b=ones(size(A,1),1);
>> [x_sor, error_sor, iter_sor, flag_sor] = sor(A, x, b, w, maxit, tol);
>> iter_sor
iter_sor =
    22
>>
```

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Nota.

Si rimane un po' sorpresi dal fatto che per $w = 1.350$ il numero di iterazioni fosse inferiore di quello fornito dal valore ottimale teorico $w^ = 1.333 \dots$*

Il fatto è che questo è ottenuto cercando di massimizzare la velocità asintotica di convergenza. Purtroppo questo minimizza una stima del numero di iterazioni k minime da compiere e non quello effettivo.

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Abbiamo detto che un punto chiave è la grandezza del **raggio spettrale** delle matrici di iterazione e che è desiderabile che questo numero oltre ad essere strettamente minore di uno sia il **più piccolo possibile**. Vediamo i raggi spettrali dei metodi esposti.

Salviamo in [raggispettrali.m](#) il seguente programma principale

```
maxit=50; tol=0;

siz=5;
A = gallery('poisson',siz); % MATRICE DI POISSON.
b=ones(size(A,1),1); % TERMINE NOTO.

[ M, N ] = split( A , b, 1.0, 1 ); % JACOBI.
P=inv(M)*N;
rho_J=max(abs(eigs(P)));
fprintf('\n \t [RAGGIO SPETTRALE][JACOBI]: %2.15f',rho_J);

[ M, N, b ] = split( A , b, 1, 2 ); % GS.
P=inv(M)*N;
rho_gs=max(abs(eigs(P)));
fprintf('\n \t [RAGGIO SPETTRALE][GAUSS-SEIDEL]: %2.15f',rho_gs);
```

Esercizio svolto: $Ax = b$ con A matrice di Poisson

```
D=diag(diag(A)); E=-(tril(A)-D); F=-(triu(A)-D);  
w=1.350;  
M=D/w-E; N=(1/w-1)*D+F; P=inv(M)*N;  
rho_sor=max(abs(eigs(P)));  
fprintf('\n \t [RAGGIO SPETTRALE][SOR BEST]:%2.15f',rho_sor);  
w=1.3333333333333333;  
[ M, N, b ] = split( A, b, w, 2 ); % SOR OPT.  
M=D/w-E; N=(1/w-1)*D+F; P=inv(M)*N;  
rho_sor_opt=max(abs(eigs(P)));  
fprintf('\n \t [RAGGIO SPETTRALE][SOR OPT]: %2.15f',rho_sor_opt);
```

Di seguito:

```
>> raggispettrali  
[RAGGIO SPETTRALE][JACOBI]: 0.866025403784438  
[RAGGIO SPETTRALE][GAUSS-SEIDEL]: 0.750000000000000  
[RAGGIO SPETTRALE][SOR BEST]: 0.350000000000001  
[RAGGIO SPETTRALE][SOR OPT]: 0.333333380707781  
>>
```

Esercizio svolto: $Ax = b$ con A matrice di Poisson

- Il valore del raggio spettrale della matrice di iterazione del metodo SOR per parametro ottimale, per quanto visto anticipatamente vale $\omega^* - 1$, e l'esperimento numerico lo conferma.
- Abbiamo osservato che in questo caso la velocità di convergenza del metodo di Gauss-Seidel è il doppio di quella di Jacobi. Poste B_{GS} , B_J le rispettive matrici di iterazione, e detta R la velocità di convergenza, osserviamo che da

$$R(B_J) := -\ln(\rho(B_J)) \quad (2)$$

$$R(B_{GS}) := -\ln(\rho(B_{GS})) \quad (3)$$

$$R(B_{GS}) := 2R(B_J) \quad (4)$$

si ha

$$-\ln(\rho(B_{GS})) = R(B_{GS}) = 2R(B_J) = -2\ln(\rho(B_J)) = -\ln(\rho(B_J))^2$$

da cui essendo il logaritmo una funzione invertibile

$$\rho(B_{GS}) = (\rho(B_J))^2.$$

Esercizio svolto: $Ax = b$ con A matrice di Poisson

Il raggio spettrale della matrice di iterazione di Gauss-Seidel coincide quindi col quadrato di quella di Jacobi ed infatti come è facile verificare

```
>> 0.866025403784438^2  
ans =  
0.750000000000000  
>>
```

Esercizio in Matlab II

- Si calcoli la matrice di Poisson P_{20} di ordine 20 aiutandosi con

```
>> help gallery
```

- Sia b il vettore composto di componenti uguali a 1, avente lo stesso numero di righe di P_{20} .
 - Si risolva col gradiente coniugato il problema $P_{20}x = b$, con tolleranza di $10^{(-12)}$, partendo da $x^0 = [0 \dots 0]$.
 - Quante iterazioni servono?
 - E migliore questo risultato di quello ottenuto con Jacobi e Gauss-Seidel?

Bibliografia



K. Atkinson, *Introduction to Numerical Analysis*, Wiley, 1989.



D. Bini, M. Capovani e O. Menchi, *Metodi numerici per l'algebra lineare*, Zanichelli, 1988.



V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, Mc Graw-Hill, 1990.