P.E.R. NEL CALCOLO DEL PAGERANK: UN CONFRONTO SULLA SCELTA DEL PRECONDIZIONATORE

LORENZO BERETTA

1. Introduzione

Il problema del PageRank risulta essere equivalente alla risoluzione del sistema lineare indotto da una M-matrice stocastica.
Nell'articolo [1] gli autori espongono una generalizzazione dei metodi delle potenze e di Jacobi per risolvere tali sistemi che prevede lo splitting di tale matrice indotto dalla proiezione su un'algebra di matrici.
Ne propongono poi una nuova particolarizzazione, il metodo HPER (*Householder preconditioned Euler-Richardson*), ottenuta proiettando rispetto all'algebra sdH delle matrici simultaneamente diagonalizzate da un'opportuna matrice di Householder H.
In quanto segue esporró brevemente il metodo e forniró un'analisi della complessitá ed uno studio sperimentale dell'efficienza.

2. Riduzione ad un sistema lineare

Sia G la matrice di adiacenza del grafo diretto del web, e sia A^T la matrice ottenuta da G sostituendo le righe nulle (in corrispondenza dei *dangling nodes*) con $\mathbb{1}^T$ e rinormalizzando ogni riga in modo che A risulti stocastica.
Allora il problema del pagerank, dati il vettore di personalizzazione v tale che $v^T\mathbb{1} = 1$ ed il paramentro $\tau \in (0, 1)$ diviene

$$(\tau A + (1-\tau)v\mathbb{1}^T)p = p \iff (I - \tau A)p = (1-\tau)v. \tag{2.1}$$

Che equivale ad $Mp = y$ per la M-matrice stocastica $M = I - \tau A$ e per $y = (1-\tau)v$, in quanto segue ci occuperemo della risoluzione di questa classe di sistemi.

3. Il metodo di Euler-Richardson precondizionato

Il metodo PER (*preconditioned Euler-Richardson*) per la risoluzione del sistema $Mx = y$ consiste nella scelta di un precondizionatore non singolare P che attraverso lo splitting $M = P - (P - M)$ ci fornisce il seguente schema iterativo:

$$x_{k+1} = P^{-1}y + (I - P^{-1}M)x_k \tag{3.1}$$

Lo stesso metodo delle potenze puó essere interpretato come un PER con la scelta del precondizionatore $P_{PM} = I - \frac{\tau}{n}\mathbb{1}\mathbb{1}^T$, $P_{PM}^{-1} = I + \frac{\tau}{n(1-\tau)}\mathbb{1}\mathbb{1}^T$.

La scelta naturale per tale precondizionatore é infatti quella che ci garantisce una bassa complessitá per lo operazioni di moltiplicaizone matrice-vettore di $I - P^{-1}M$.
Una classe di trasformazioni di questo tipo é data dalle algebre sdU (*simultaneously diagonalized by U*) dove U é una matrice unitaria.
Data $\mathcal{U} = sdU = \{Udiag(\lambda)U^* | \lambda \in \mathbb{C}^d\}$ la scelta piú naturale per P diviene la

proiezione euclidea $\mathcal{U}_M$ di M su U, infatti questa minimizza $\|M-P\|_2$.
Tale proiezione si puó calcolare come

$$\mathcal{U}_M = Udiag(U^*MU)U^* = I - \tau Udiag(U^*AU)U^* = I - \tau\mathcal{U}_A \tag{3.2}$$

La scelta piú banale é $\mathcal{U} = sdI$, ovvero l'algebra delle matrici diagonali, questa da origine al metodo di Jacobi

4. Il metodo HPER

Data una matrice di Householder $H(w) = I - 2ww^*$ consideriamo l'algebra $\mathcal{U} = sdH(w)$ e utilzziamola per la scelta del precondizionatore nel metodo PER.
In particolare desideriamo che $\mathcal{U}$ sia debolmente stocastica, ovvero che valga

$$A\mathbb{1}^T = \mathbb{1} \implies \mathcal{U}_A\mathbb{1}^T = \mathbb{1} \tag{4.1}$$

infatti questo ci garantisce in primo luogo che $\mathcal{U}_M = I - \tau\mathcal{U}_A$ sia invertibile e in secondo luogo, avendo caratterizzato le algebre sdU debolmente stocastiche come tutte e sole quelle contenenti $\mathbb{1}\mathbb{1}^T$, ci assicura che valga $\|M - \mathcal{U}_M\| \leq \|M - P_{PM}\|$ che pur non fornendoci alcuna stima esatta sul rapporto di convergenza suggerisce che tale precondizionatore possa essere una scelta migliore di P_{PM}.

Enunciamo un'altra caratterizzazione delle algebre sdU debolmente stocastiche (facilmente ottenibile dalla precedente): sdU é debolmente stocastica se e solo se U ha una colonna di elementi costanti.
Si ottiene cosí che le uniche algebre $sdH(w)$ debolmente stocastiche sono $sdH(w_k^-)$ e $sdH(w_k^+)$ al variare di k con

$$w_k^- = \beta_n^-(\sqrt{n}e_k + \mathbb{1}), \quad w_k^+ = \beta_n^+(\sqrt{n}e_k - \mathbb{1}), \quad \beta_b^\pm = \frac{1}{\sqrt{2}\sqrt{n}(\sqrt{n} \mp 1)} \tag{4.2}$$

dove n é la dimensione della matrice.

Scegliendo $k = 1$ il nostro precondizionatore risulta dunque

$$\mathcal{U}_M = I - \tau H(w_1^+)diag(H(w_1^+)AH(w_1^+))H(w_1^+) \tag{4.3}$$

5. Complessitá computazionale

Indichiamo con $\chi(B)$ il costo computazionale del prodotto Bv per $v \in \mathbb{R}^n$.

5.1. **Metodo delle Potenze.** Per questo metodo, come visto a lezione, il costo di un'iterazione é $\chi(A)$ e il tasso di convergenza é dato da $\max(Sp(A) \setminus \{1\}) < 1$.

5.2. **HPER.** Il preprocessing consiste nel calcolare $D := diag(H(w)AH(w))$:

$$diag(H(w)AH(w))_{i,i} = (A)_{i,i} - 2w_i((Aw)_i + (w^TA)_i - 2\gamma_n w_i), \tag{5.1}$$

$$\gamma_n = n\beta_n^2((A)_{i,i} - \frac{1}{\sqrt{n}} - \frac{1}{\sqrt{n}}(A\mathbb{1})_i + 1). \tag{5.2}$$

Questo ha costo $\chi(A) + O(n)$.
La matrice di iterazione é

$$I - H(w)(I - \tau D)^{-1}H(w)(I - \tau A). \tag{5.3}$$

Dunque il costo di un'iterazione é $\chi(A)+\chi(H(w))+O(n) = \chi(A)+O(n)$, otteniamo quindi un costo per operazione equivalente a quello del metodo delle potenze.

Purtroppo non abbiamo gli strumenti teorici per controllare gli autovalori della matrice (5.3) quindi per quanto riguarda il tasso di convergenza ci affideremo ai risultati sperimentali.

5.3. **Metodo di Jacobi.** Qui la matrice di iterazione[1] vale

$$\tau(A - diag(A))(I - \tau diag(A))^{-1}. \tag{5.4}$$

quindi il costo per terazione é ancora $\chi(A) + O(n)$.
Il tasso di convergenza é dato dal raggio spettrale di (5.4) che si puó stimare dall'alto grazie al teorema di localizzazione di Gershgorin con

$$\max_j \left(\frac{\tau \sum_{i \neq j} (A)_{i,j}}{1 - \tau (A)_{j,j}} \right) = \max_j \left(\frac{\tau(1 - (A)_{j,j})}{1 - \tau (A)_{j,j}} \right) \tag{5.5}$$

inoltre

$$\frac{1 - (A)_{j,j}}{1 - \tau (A)_{j,j}} \leq \tau^{\epsilon} \iff (A)_{j,j} \geq \frac{1 - \tau^{\epsilon}}{1 - \tau^{1+\epsilon}} \tag{5.6}$$

quindi il raggio spettrale della matrice di terazione é limitato da $\tau^{1+\lambda} < 1$ con

$$\lambda = \max \left\{ \epsilon \geq 0 \,\middle|\, \forall i (A)_{i,i} \geq \frac{1 - \tau^{\lambda}}{1 - \tau^{1+\lambda}} \right\} \tag{5.7}$$

6. Risultati sperimentali

Ho implementato in Matlab gli algoritmi per HPER, Jacobi ed il metodo delle potenze sfruttando l'ottimizzazione delle funzioni di libreria nel caso di matrici sparse, i codici utilizzati si possono trovare nell paragrafo 7.

Per valutare le prestazioni degli algoritmi conteggeremo il numero di iterazioni necessarie per raggiungere la condizione di arresto, sceglieremo la stessa condizione trovata in [1], ovvero

$$\|Mx - y\|_2 < \|y\|_2 \, 10^{-13}. \tag{6.1}$$

In [1] gli algoritmi non vegono direttamente eseguiti sulla matrice A definita in (2.1) ma sulla matrice

$$\widetilde{A}(\alpha) = \alpha I + (1 - \alpha)A, \quad \alpha \in (0, 1) \tag{6.2}$$

implementeró quindi varie versioni HPER-α al variare della scelta di $\alpha \in (0, 1)$, a titolo d'esempio ho eseguito gli algoritmi per $\alpha = 0.1, 0.2, 0.3$.
Il motivo che ha spinto gli autori di [1] a questa scelta nelle sperimentazioni é che HPER é sperimentalmente piú efficiente in questo caso, e per $\alpha \to 0$ le soluzioni trovate convergono al vettore di pagerank.

[1] $M = I - \tau A$ risulta strettamente dominante diagonale per colonne, e non per righe come richiederebbe l'usuale condizione sufficiente alla convergenze di Jacobi, per garantire la convergenza l'algoritmo deve quindi essere leggermente modificato affinché risolva $MD^{-1}z = y$ con $D = diag(M)$, $z = Dx$ e la matrice di iterazione risulta quindi $I - MD^{-1}$.

Vedremo poi attraverso dei grafici come gli ordinamenti trovati attraverso questa perturbazione differiscono sempre piú dal vero pagerank al crescere di α.

6.1. **Matrice sintetica.** Useremo una matrice generata con il comando `sprand(n, n,d)` che prende in input il numero n di nodi del grafo e la densitá d di archi.
Per quanto riguarda le dimensioni il massimo numero di entrate non nulle che il mio laptop riusciva contenere in RAM era di circa 10^7, per valori maggiori i dati di Matlab andavano in swap portandolo al crush.
Gli esperimenti sono quindi stati svolti impostando $n = 0.510^7$ e $d = \frac{1}{n}$.

Iterazioni

τ	HPER-0.3	HPER-0.2	HPER-0.1	HPER-0	Jacobi	Power Method
0.75	40	42	45	47	47	47
0.80	45	49	51	53	55	53
0.85	53	54	58	60	73	62
0.90	62	65	67	71	108	71
0.95	75	80	82	82	206	82

Tempi (in secondi)

τ	HPER-0.3	HPER-0.2	HPER-0.1	HPER-0	Jacobi	Power Method
0.75	29.2	29.6	30.6	31.3	15.3	13.1
0.80	30.8	32.3	32.7	33.9	17.4	14.6
0.85	33.7	33.8	35.3	36.0	21.5	17.3
0.90	36.9	37.6	38.3	41.1	30.0	19.9
0.95	41.7	43.3	44.7	44.3	54.1	22.9

Possiamo notare che le varie versioni di HPER-α impiegano meno terazioni rispetto al metodo delle potenze (come riportato in [1]), andando a vedere i tempi peró si percepisce che non si ha un reale vantaggio in efficienza, inoltre vedremo che l'aumentare di α, che ci garantisce un'efficienza maggiore, rende il nostro ordinamento sempre meno aderente al modello.

6.2. **Matrice reale.** Useremo una matrice ricavata da un grafo di adiacenza reale, in particolare la stessa utilizzata nelle sperimentazioni del corso, contenuta nel file `web-BerkStan.txt` scaricato da `https://snap.standoford.edu/data/`.
Essa ha circa 10^6 nodi e 10^7 elementi non nulli, quindi sta dentro i limiti di capacitá del mio calcolatore.

Iterazioni

τ	HPER-0.3	HPER-0.2	HPER-0.1	HPER-0	Jacobi	Power Method
0.75	69	78	86	94	94	94
0.80	88	99	110	121	122	121
0.85	119	135	151	166	167	167
0.9	182	207	232	257	257	257
0.95	369	422	476	529	531	531

Tempi (in secondi)

τ	HPER-0.3	HPER-0.2	HPER-0.1	HPER-0	Jacobi	Power Method
0.75	4.5	4.7	4.9	5.2	3.0	2.7
0.80	5.0	5.4	5.8	6.2	3.7	3.5
0.85	6.1	6.7	7.2	7.7	5.0	4.8
0.90	8.3	9.1	10.2	10.9	7.5	7.3
0.95	14.8	16.7	18.5	20.4	15.1	15.1

Il numero di iterazioni nel caso reale é decisamente a favore di HPER-α. Andiamo quindi a guardare i tempi di esecuzione: si nota che il preprocessing ha un'incidenza importante nel metodo HPER e questo vanifica il vantaggio sul numero di iterazioni. Infatti HPER-0.3 esegue 29.13 iterazionoi al secondo, un contro le 35.24 del power method, ovvero il 17% in meno, mentre il numero di iterazioni totali fatte da HPER-0.3 sono ben il 31% in meno di quelle del power method, i tempi di preprocessing peró sono 2.13 s per HPER contro appena 0.02 s per il metodo delle potenze, confermando il ruolo di questa prima fase nel rendere HPER non cosí vantaggioso.

Qui sotto possiamo osservare 3 grafici che rappresentano la permutazione necessaria per ottenere i primi 100 elementi del vettore di pagerank partendo da quello calcolato da HPER-α per $\alpha = 0.1, 0.2, 0.3$, l'esperimento é stato fatto con i dati della matrice reale di cui sopra e ponendo $\tau = 0.85$, si puó notare come superate le prime 20 pagine l'ordinamento diventi tutt'altro che fedele al modello e come questo peggiori all'aumentare di α (che é proprio ció che ci garantisce le prestazioni migliori!).

FIGURE 1. HPES-0.1 comparato a PageRank

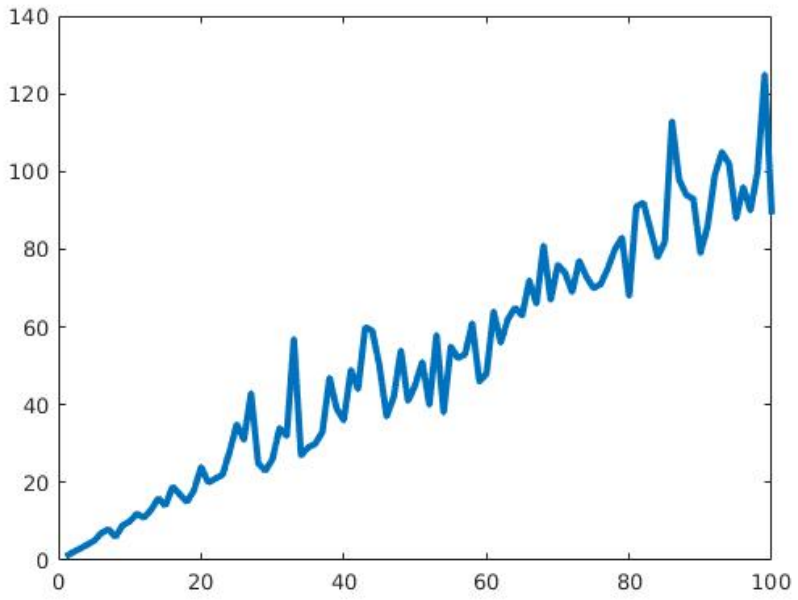


FIGURE 2. HPES-0.2 comparato a PageRank

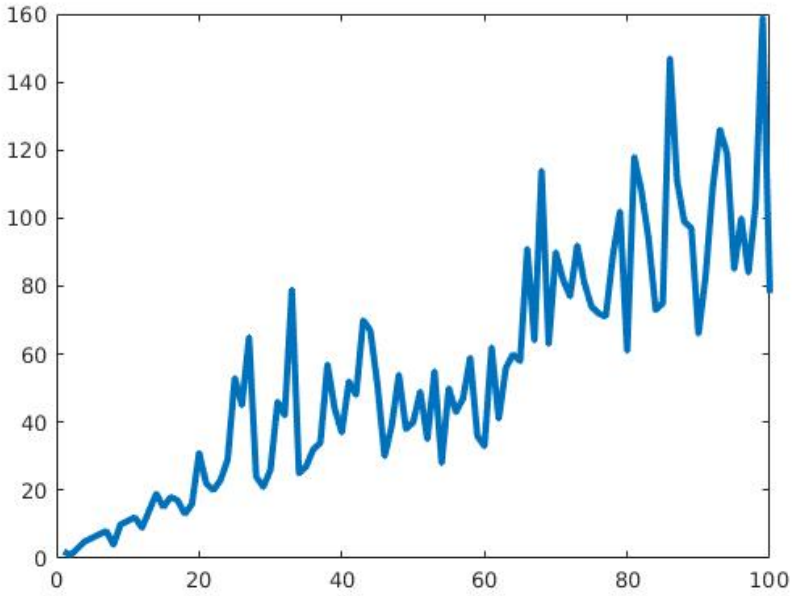
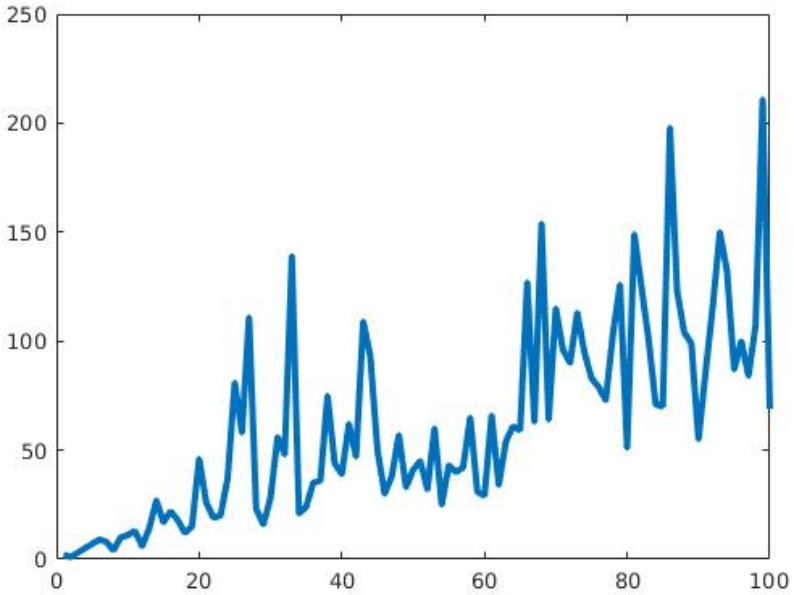


FIGURE 3. HPES-0.3 comparato a PageRank



7. CODICI

LISTING 1. HPER-α

```
function y = HPER_alpha(A, v, tau, itmax, alpha)
unalpha=1-alpha;
n = size(A,1);
v = (1-tau)/sum(v)*v;
```

```
e = ones(n,1);
d = A*e;
dang = d==0;
d = d+n*dang;
d = 1./d;
A = A';
beta = 1/sqrt(2*sqrt(n)*(sqrt(n)-1));
w = -ones(n,1)*beta;
w(1) = w(1)+beta*sqrt(n);
Aw = A*(w.*d)+(dang'*(w.*d))*e;
Aw = alpha*w+unalpha*Aw;
wtA = d'.*(w'*A)+sum(w)*(dang'.*d');
wtA = alpha*w'+unalpha*wtA;
A11=alpha+unalpha*(A(1,1)+dang(1))*d(1);
sumA1=alpha+unalpha*sum((A(1,:)+dang').*d');
gamma = A11-(sumA1+1)/sqrt(n)+1;
gamma = n*beta^2*gamma;
s = zeros(n,1);
for i = 1 : n
    s(i) = Aw(i)+wtA(i)-2*gamma*w(i);
    Aii=alpha+unalpha*d(i)*(A(i,i)+dang(i));
    s(i) = Aii-2*w(i)*s(i);
end
s = e-tau*s;
s = 1./s;
y0 = v-2*(w'*v)*w;
y0 = y0.*s;
y0 = y0-2*(w'*y0)*w;
x = rand(n,1);
x = x/sum(x);
dang_red = dang.*d;
for it = 1 : itmax
    Ax = A*(x.*d)+(dang_red'*x)*e;
    Ax = alpha*x+unalpha*Ax;
    y = x-tau*Ax;
    err = norm(v-y);
    y = y-2*(w'*y)*w;
    y = y.*s;
    y = y-2*(w'*y)*w;
    y = y0+x-y;
    if err<1.e-13*norm(y)
        break
    end
    x=y;
end
end
```

LISTING 2. Metodo di Jacobi

```
function y = Jacobi(A,v,tau,itmax, mod)
n = size(A,1);
e = ones(n,1);
v = (1-tau)/sum(v)*v;
d = A*e;
dang = d==0;
d = d+n*dang;
d = 1./d;
A = spdiags(d,[0],n,n)*A;
A = A';
dang = dang/n;
s = e-tau*(diag(A)+dang);
s = 1./s;
x = rand(n,1);
x = x/sum(x);
for it=1 : itmax
    y = s.*x;
    y = y-tau*(A*y+(dang'*y)*e);
    err = norm(y-v);
    y = v+x-y;
    x=y;
    if err<1.e-13*norm(y)
        break;
    end
    y=s.*y;
end
end
```

LISTING 3. Metodo delle Potenze

```
function y = Power_method(A, v, tau, itmax, mod)
n = size(A,1);
e = ones(n,1);
d = A*e;
d = d';
dang = d==0;
d = d + dang*n;
dang = dang'/n;
d = 1./d;
x = rand(1,n);
x = x/sum(x);
v = (1-tau)/sum(v)*v;
v=v';
for it=1:itmax
    y = x.*d;
    y = y*A+(x*dang)*e';
    y = tau*y+v;
    err = norm(x-y);
    x = y;
```

```
        if  err<1.e-13*norm(y)
            break
        end
end
end
```

References

[1] Cipolla Stefano, Di Fiore Carmine, Tudisco Francesco. *Euler-Richardson method preconditioned by weakly stochastic matrix algebras: a potential contribution to Pagerank computation.* Electronic Journal of Linear Algebra, Volume 32, pp. 254-272, 2017.