GLOBAL JOURNAL OF MATHEMATICAL SCIENCES VOL10, NO.1 & 2, 2011: 17-26

 ISSN 1596-6208
www.globaljournalseries.com; Email: info@globaljournalseries.com

A COMPARATIVE APPLICATION OF JACOBI AND GAUSS SEIDEL'S NUMERICAL ALGORITHMS IN PAGE RANK ANALYSIS.

FELIX U. OGBAN

(Received 27, January 2011; Revision Accepted 18, March 2011)

ABSTRACT

In PageRank calculation the Jacobi matrix is given by d T (damping factor times transition matrix), a sparse matrix. The solution of the iteration is x, if the limit exists. The convergence is guaranteed, if the absolute value of the largest eigen value of $(1-M)$ is less than one. In case of PageRank calculation this is fulfilled for 0< d < 1. An improved Gauss-Seidel iteration algorithm, based on the decomposition $M = D + L + U$ where D, L and U are the diagonal, lower triangular and upper triangular parts of M would yield $x_{i+1} = D - 1 * (L * x_{i+1} + U * x_i + b)$. Introducing a relaxation parameter λ ≠ 0 leads to a generalization of the Gauss-Seidel method $x_{i+1} = (1-\lambda)x_i + \lambda D - 1 * (L * x_{i+1} + U * x_i + b)$. This work compares the two methods with a C++ code numerically.

INTRODUCTION

PageRank is Google's system for ranking web pages. A page with a higher PageRank is deemed more important and is more likely to be listed above a page with a lower PageRank.

PageRank relies on the uniquely democratic nature of the web by using its vast link structure as an indicator of an individual page's value. In essence, Google interprets a link from page A to page B as a vote, by page A, for page B. But, Google looks at more than the sheer volume of votes, or links a page receives; it also analyzes the page that casts the vote. Votes cast by pages that are themselves "important" weigh more heavily and help to make other pages "important." (Yibei and Jie, 2004)

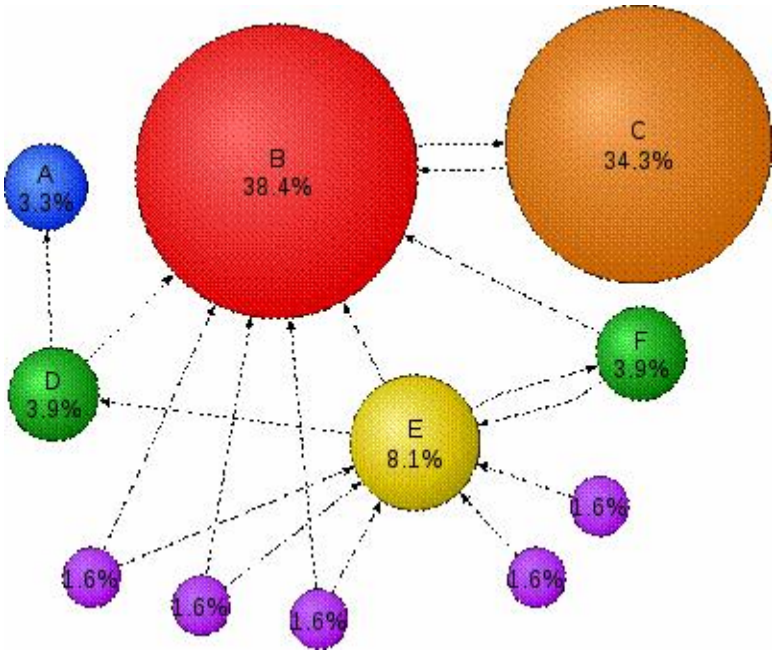
In other words, Google conducts "elections" in which each web page casts votes for web pages with hyperlinks to those pages. But unlike a democracy a page can have more than one vote and links from pages with high PageRank are given more weight (according to their ranking) and thus help to improve the targets' PageRank.

Page Rank is certainly one of the most important algorithms existing today. It is used by Google to assign a value to each page on the Internet, measuring importance or influence of it's content. In order to improve it we need to understand how it works.

Every time a page is linked to, it receives a certain amount of Page Rank, depending on how much giving page has accumulated. Also every time a page links to another one, it loses certain amount of page rank, depending on how many outgoing links are there. (Yibei and Jie, 2004)

In other words, page rank flows from page to page, like a juice flowing from glass to glass. At the end, it tends to accumulate on pages which have more incoming links, carrying more Page Rank with them. The process is displayed in the following image, courtesy of (Wikipedia, 2000)

Felix U. Ogban, Department of Mathematics/Statistics and Computer Science, Faculty of Science, University of Calabar, Calabar, Nigeria.



Background

Page Rank is a virtual value meaning nothing until you put it into the context of search engine results.

Higher Page Rank pages will have tendency to rank better in the search engine results provided they are still optimized for the keywords you are searching for. Visitors coming from search engines are most praised kind of visitors because they are generally interested in what you have to say (or sell, if you look at it commercially; they also cost nothing).

How do we improve Page Rank?

The most obvious solution will be to get as many incoming links as you can, while shutting down your site and not linking to anyone else. Wikipedia is one example of such closed system, as every outgoing link on Wikipedia is 'nofollow'. (Yibei and Jie,2004)

However things are not as easy, as Google has tweaked their algorithm over the years and in effort to fight those kind of Page Rank conservation they have probably invented numerous algorithms to detect and even punish such sites.
Other 'white hat' techniques like 'page rank sculpting' allow you to flow the page rank juice to the areas of your site where you want them most. (Zeinalipour-Yazti, and Dikaiakos, 2002)

Page rank juice is hard to control and up to now only the experts could configure the page rank flow on your site.
Thank to a new tool, everyone will be able to get instant Page Rank Juice information for any page. This information will in turn allow you to understand where is your page rank flowing and what you can do to preserve it and channel it to areas you need it most.

Calculating Page Rank

An important point in the PageRank calculation is the matrix inversion. Therefore, we briefly discuss some numerical inversion algorithms, where the equation to solve is:

$$M * x = b \quad \text{.....1.1}$$

Using Jacobi-iteration

A simple method is the Jacobi-iteration named after the German mathematician Carl Gustav Jacob Jacobi. It is defined by:

$$x_0 = b$$
$$x_{i+1} = (1 - M) * x_i + b \quad \text{.....1.2}$$

for i ≥ 0 , where {i} denotes the result after the i[th] iteration. The Matrix $(1-M)$ is called Jacobi matrix. In case of PageRank calculation the Jacobi matrix is given by d T (damping factor times transition matrix), a sparse matrix. The solution of the iteration is x, if the limit exists. The convergence is guaranteed, if the absolute value of the largest eigen value of $(1-M)$ is less than one. In case of PageRank calculation this is fulfilled for 0< d < 1.

There is a generalization of the algorithm. Using the decomposition $(D-M)$, where D is an easy to invert matrix (e.g. the diagonal elements of M), leads to:

$$x_0 = D-1*b$$
$$x_{i+1} = D-1*\left((D-M)*x_i + b\right) \quad \text{.....1.3}$$

The Gauss-Seidel method

An improved algorithm is the Gauss-Seidel iteration. It based on the decomposition (Shkapenyuk, and Suel, 2002)

$$M = D + L + U \quad \text{.....1.4}$$

where D, L and U are the diagonal, lower triangular and upper triangular parts of M.
This yield

$$x_{i+1} = D-1*\left(L*x_{i+1} + U*x_i + b\right) \quad \text{.....1.5}$$

Introducing a relaxation parameter λ ≠ 0 leads to a generalization of the Gauss-Seidel method:

$$x_{i+1} = (1-\lambda)x_i + \lambda D-1*\left(L*x_{i+1} + U*x_i + b\right) \quad \text{.....1.6}$$

The Minimal residue method

Another iteration scheme is the minimal residue iteration. It is given by

$$x_{i+1} = x_i + r_i\left(\frac{r_i\, jMjk\, r_i k}{Mjk\, r_i kMj1\, r_i 1}\right) \quad \text{.....1.7}$$

where the residue is defined by

$$r_i = b - M*x_i \quad \text{.....1.8}$$

The minimal residue iteration is never divergent.

Summarily, calculating the PageRank is nothing else than solving the following linear system of equations

$$M*PR = (1-d) \quad \text{.....1.9.}$$

where 0 < d <1 denotes a damping factor,
PR is an N-dimensional vector und M a N x N-matrix.
N is the number of pages within the system.
The i-th component of the vector PR, i.e. PR_i, is the PageRank of site i.

The matrix M is given by

$$M = 1 - dT \quad \text{.....1.10}$$

where T stands for the transition matrix.
The components of T are given by the number of outgoing links:

$$T_{ij} = 1/C_j \quad \text{(if page j is linking to page i)……1.11}$$
$$T_{ij} = 0 \quad \text{(otherwise) .....1.12}$$

where Cj is the number of links on page j.

The solution of the linear system of equations is

$$PR = M^{-1} * (1 - d) \quad \text{.....1.13}$$

The calculation of the inverse matrix M^{-1} can (in principle) be done analytically. For 0 < d <1 there are no eigen values of M which are zero. Therefore, a unique solution exists. However, for larger N the calculation should be done numerically because it's less time consuming. This is done by some kind of iteration scheme. The simplest method is the Jacobi-iteration:

$$PR_{k+1} = (1 - d) + dT * PR_k \quad \text{.....1.14}$$

PR_k denotes the value of the PageRank vector after the k[th] iteration. $PR_0 = 1$, can be taken as the initial PageRank vector. Of course, the solution is independent from the initial guess. The convergence is guaranteed, since the largest eigen value of dT is smaller than 1. The damping factor specifies the amount of PageRank that is transferred. In case of d = 1 the whole PageRank is passed while for d<1 there is a damping. There are convergence problems if d is close to 1 and the number of iterations for a stable solution increases. In practice about 100 iterations are necessary to get a stable result (Shkapenyuk, and Suel, 2002). Obviously, the number of iterations depends on the linking structure as well as the accuracy. Taking the final result of the last calculation as input for the new iteration reduces the number of iterations (assuming that the linking structure hasn't changed completely).

Rewriting the last equation for the components and omitting the denotation for the iteration yields:

$$PR_i = (1 - d) + d(PRX1 / CX1 + \cdots + PRXn / CXn) \quad \text{.....1.15}$$

Often this equation is referred as PageRank algorithm. However, strictly speaking this is an iteration scheme to solve the matrix inversion for the equation at the top. Anyway the Jacobi-iteration isn't the faster way for computation. In general the following iteration scheme (minimal residue)

$$PR_{k+1} = PR_k + R_k \frac{\sum R_i M_{ij} R_j}{\sum M_{in} R_n M_{ij} R_j} \quad \text{.....1.16}$$

with

$$R_{k+1} = R_k - M * R_k \left(\frac{\sum R_i M_{ij} R_j}{\sum M_{in} R_n M_{ij} R_j} \right) \quad \text{.....1.17}$$

converge faster. There are several other iteration schemes (Householder, 1964) such as Gauss-Seidel, overrelaxation methods, conjugate gradient, preconditioning methods, multigrid and blocking techniques or Chebyshev. However, some iterations schemes are restricted to hermitian matrices.

In some cases the linear system of equations

$$M * PR = \left(\frac{1 - d}{N} \right) \quad \text{.....1.18}$$

is considered instead of the first equation given as 1.9 above. Obviously, the solution vector is the same apart from a normalization factor of $1/N$.

The case d=1 (no damping) has to be treated separately. This corresponds to determine the eigen vector of T with eigen value 1. The problem is degenerate eigen values. They appear for linking structures where not every page can be reached from every other page. This is the case for dead ends (pages without outgoing links) or maple leaves (close structures) (Zeinalipour-Yazti, and Dikaiakos, 2002). In these cases the numerical solution given by the iteration scheme depends on the initial vector. It is a combination of eigen vectors for the eigen value 1. However, for d < 1 dead ends don't cause problem and don't have to be treated differently.

Applications and Discussion

In this section, we develop a computer program that numerically and iteratively solves a given system of equations using the Jacobi and the Gauss siedel's method.

Given that the matrix **A** is:

$$\begin{pmatrix} 10 & 1 & 2 & 1 & 1 \\ 1 & 15 & 1 & 1 & 1 \\ 2 & 2 & 17 & 2 & 1 \\ 3 & 1 & 1 & 24 & 1 \\ 2 & 3 & 2 & 3 & 14 \end{pmatrix}$$

It took the following twenty-two iterations to converge using the Jacobi algorithm as discussed above. The table is as shown below:

Table 3: 1 Program output of convergence after 22 iterations using Jacobi's method

Ite			
0	3	5	4
1	-1.4	-.4	-1.176471
2	.7764846	.6392251	.4820728
3	-8.037908E-02	.1027348	-.2107088
4	.2644467	.3077914	.0731537
5	.1224915	.2227609	-4.138098E-02
6	.1797812	.2571133	5.374017E-03
7	.1564078	.243119	-1.360121E-02
8	.1658969	.2488053	-5.881143E-03
9	.1620369	.2464931	-9.019072E-03
10	.1636059	.2474331	-7.743189E-03
11	.162968	.2470509	-8.261907E-03
12	.1632273	.2472063	-8.051015E-03
13	.1631219	.2471431	-8.136753E-03
14	.1631648	.2471688	-8.101893E-03
15	.1631473	.2471583	-8.116068E-03
16	.1631544	.2471626	-8.110303E-03
17	.1631515	.2471609	-8.112648E-03
18	.1631527	.2471616	-8.111696E-03
19	.1631522	.2471613	-8.112083E-03
20	.1631524	.2471614	-8.11193E-03
21	.1631524	.2471614	-8.111991E-03

The vector B is:

B(1) = 2
B(2) = 4
B(3) = 1
B(4) = 5
B(5) = 1

The solutions for x-values were:

X(1) = .1631524
X(2) = .2471614
X(3) = -8.111967E-03
X(4) = .1797372
X(5) = -4.219831E-02
With an estimated accuracy of **+- 8.009374E-08**

Below is the program listing for Jacobi's method

```
/*
-------------------------------------------------------------------------
 (Jacobi Iteration).
 To solve the linear system  AX = B  by starting with  P_0 = 0
 and generating a sequence  {P_K}  that converges to the
 solution  P  (i.e., AP = B).  A sufficient condition for the
 method to be applicable is that  A  is diagonally dominant.
-------------------------------------------------------------------------
*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
/* ------------------------------------------------------ */
#define Limit  20
void main(void)
{
   double Tol = 10E-6;          /* Tolerance                */
   double Sep  = 1.0;           /* Initialize               */
   int K    = 1;                /* Counter for iterations   */
   int Max  = 99;               /* Maximum number of iterat.*/
   int Cond = 1;                /* Condition of matrix      */
   int R, C, J;                 /* Loop counters            */
   double A[Limit][Limit];      /* A in  AX = B ,  INPUT    */
   double B[Limit];             /* B in  AX = B ,  INPUT    */
   int N;                       /* Dimension of A, INPUT    */
                                /* = Number of equations    */
   double Row;                  /* Variable in dominance check */
   double P[Limit];
   double Pnew[Limit];
   double Sum;

   printf("-----------------------------------------\n");
   do  /* force proper input */
   {
     printf("Please enter number of equations [Not more than %d]\n",Limit);
     scanf("%d", &N);
   } while( N > Limit);

   printf("You say there are %d equations.\n", N);
   printf("-------------------------------------------------------\n");
   printf("From  AX = B enter components of vector B one by one:\n");

   for (R = 1; R <= N; R++)
   {
      printf("Enter %d st/nd/rd component of vector B\n", R);
      scanf("%lf", &B[R-1]);
   }
   printf("-------------------------------------------------------\n");
   printf("From  AX = B enter elements of A row by row:\n");
   printf("-------------------------------------------------------\n");

   for (R = 1; R <= N; R++)
   {
      for (J = 1; J <= N; J++)
      {
          printf(" For row %d enter element %d please :\n", R, J);
          scanf("%lf", &A[R-1][J-1]);
      }
   }

   /* Check for diagonal dominance. */
   for (R = 1; R <= N; R++)
   {
```

```
    Row = 0.0;
    for (C = 1; C <= N; C++) Row += fabs(A[R-1][C-1]);
    if( Row >= 2.0 * fabs(A[R-1][R-1]) ) Cond = 0;
  }

  if( Cond == 0 )
  {
     printf("The matrix is not diagonally dominant.\n");
     printf("Cannot apply this algorithm ---> exit\n");
     exit(1);
  }

  /* Initialize : allow user to do this in order to create a
                sequence of P_k values  */
  for ( J = 0; J < N; J++)
  {
   P[J]    = 0;
   Pnew[J] = 0;
  }

  /* Perform Jacobi iteration */
  while( (K < Max) && (Sep > Tol) )
  {
   for (R = 1; R <= N; R++)
   {
     Sum = B[R-1];
     for (C = 1; C <= N; C++) if(C != R) Sum -= A[R-1][C-1] * P[C-1];
     Pnew[R-1] = Sum / A[R-1][R-1];
   }
   /* Convergence criterion */
   Sep = 0;
   for (J = 1; J <= N; J++) Sep += fabs(Pnew[J-1] - P[J-1]);
   /* Update values and increment the counter */
   for (J = 1; J <= N; J++) P[J-1] = Pnew[J-1];
   K++;
  }

   /* output */
   if(Sep < Tol) printf("The solution to the linear system is :\n");
   else printf("Jacobi iteration did not converge:\n");

   for (J = 1; J <= N; J++) printf("P[%d] = %lf\n", J, P[J-1]);

  }
```

In the same context, matrix **A** above is inputed for the Guass-siedel method as discussed earlier with the same initial values. However, the method converged after only nine (9) iterations.

Ite			
0	3	5	4
1	-1.4	-.1066667	-5.803922E-02
2	.1646323	.2211324	-4.096943E-02
3	.1720267	.2485602	-8.968539E-03
4	.1634565	.24738	-7.917108E-03
5	.1631041	.24716	-8.098174E-03
6	.1631484	.2471598	-8.11272E-03
7	.1631525	.2471613	-8.112095E-03
8	.1631524	.2471614	-8.111968E-03

The vector B is:

B(1) = 2
B(2) = 4
B(3) = 1
B(4) = 5
B(5) = 1

The solutions are:

X(1) = .1631524
X(2) = .2471614
X(3) = -8.11197E-03
X(4) = .1797372
X(5) = -4.219831E-02

With an estimated accuracy of **+- 4.004687E-08**

Below is the program listing for Guass-siedel's method.

```
/*
-------------------------------------------------------------------------
(Gauss-Seidel-Iteration).
 To solve the linear system  AX = B  by starting with  P_0 = 0
 and generating a sequence  { P_K}  that converges to the
 solution  P  (i.e., AP = B).  A sufficient condition for the
 method to be applicable is that  A  is diagonally dominant.
-------------------------------------------------------------------------
*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
/* ------------------------------------------------------- */
#define Limit  20

void main(void)
{
   double Tol = 10E-6;             /* Tolerance                */
   double Sep  = 1.0;              /* Initialize               */
   int K    = 1;                 /* Counter for iterations   */
   int Max  = 99;                /* Maximum number of iterat.  */
   int Cond = 1;                 /* Condition of matrix      */
   int R, C, J;                  /* Loop counters            */
   double A[Limit][Limit];       /* A in  AX = B ,  INPUT    */
   double B[Limit];              /* B in  AX = B ,  INPUT    */
   int N;                        /* Dimension of A, INPUT    */
                                 /* = Number of equations    */
   double Row;                   /* Variable in dominance check */
   double P[Limit];
   double Pold[Limit];
   double Sum;

   printf("-----------------------------------------\n");
   do  /* force proper input */
   {
     printf("Please enter number of equations [Not more than %d]\n",Limit);
     scanf("%d", &N);
   } while( N > Limit);

   printf("You say there are %d equations.\n", N);
   printf("-----------------------------------------------------\n");
   printf("From  AX = B enter components of vector B one by one:\n");

   for (R = 1; R <= N; R++)
   {
      printf("Enter %d st/nd/rd component of vector B\n", R);
      scanf("%lf", &B[R-1]);
   }
   printf("-----------------------------------------------------\n");
   printf("From  AX = B enter elements of A row by row:\n");
   printf("-----------------------------------------------------\n");
```

```
for (R = 1; R <= N; R++)
   {
      for (J = 1; J <= N; J++)
      {
          printf(" For row %d enter element %d please :\n", R, J);
          scanf("%lf", &A[R-1][J-1]);
      }
   }

   /* Check for diagonal dominance. */
   for (R = 1; R <= N; R++)
   {
     Row = 0.0;
     for (C = 1; C <= N; C++) Row += fabs(A[R-1][C-1]);
     if( Row >= 2.0 * fabs(A[R-1][R-1]) ) Cond = 0;
   }

   if( Cond == 0 )
   {
      printf("The matrix is not diagonally dominant.\n");
      printf("Cannot apply this algorithm ---> exit\n");
      exit(1);
   }

   /* Initialize : allow user to do this in order to create a
                   sequence of P_k values  */
   for ( J = 0; J < N; J++)
   {
    P[J]    = 0;
    Pold[J] = 0;
   }

   /* Perform Gauss-Seidel iteration */
   while( (K < Max) && (Sep > Tol) )
   {
    for (R = 1; R <= N; R++)
    {
      Sum = B[R-1];
      for (C = 1; C <= N; C++) if(C != R) Sum -= A[R-1][C-1] * P[C-1];
      P[R-1] = Sum / A[R-1][R-1];
    }
    /* Convergence criterion */
    Sep = 0;
    for (J = 1; J <= N; J++) Sep += fabs(P[J-1] - Pold[J-1]);
    /* Update values and increment the counter */
    for (J = 1; J <= N; J++) Pold[J-1] = P[J-1];
    K++;
   }

    /* output */
    if(Sep < Tol) printf("The solution to the linear system is :\n");
    else          printf("Gauss-Seidel iteration did not converge:\n");

    for (J = 1; J <= N; J++) printf("P[%d] = %lf\n", J, P[J-1]);

}
```

CONCLUSION

Some people will have the idea that generating millions of pages is a good way to produce PageRank and improve ranking of their website after studying the PageRank algorithm. Theoretically this should work assuming that an appropriate linking structure is chosen. However, practically it doesn't work! Google changed their algorithm years ago. One of the changes prevents the generation of PageRank. Moreover in practice you even need PageRank to get larger websites completely crawled. Google also made some minor changes of the algorithm, e.g. they changed the

value of the damping factor which originally was d=0.85. what Google had not done is consider other methods that can numerically estimate the page rank as they are linked.

The table below would help us conclude that the Gauss-Siedel's method which converges faster and with a very high accuracy is best recommended for use. One would have thought that its earlier convergence may have an effect on the x-values produced.

	Accuracy	No of Iterations	X1	X2	X3	X4	X5
Jacobi Method	**+- 8.009374E-08**	22	0.16	0.25	-0.008	0.18	-0.04
Gauss-Siedel	**+- 4.004687E-08**	9	0.16	0.25	-0.008	0.18	-0.04

Since the accumulated PageRank of all pages of the web equals the total number of web pages, it follows directly that an additional web page increases the added up PageRank for all pages of the web by one. But far more interesting than the effect on the added up PageRank of the web is the impact of additional pages on the PageRank of actual websites and its effect if not numerically recalculated.

Whereas several web page ranking algorithms avoid numerical analysis of their updates and results thus leaving the user with no good choice than to take what ever is given; relevant or not.

REFERENCES

Housholder A. S., 1964. "The Theory of Matrices in Numerical Analysis" Prentice Hall, New York, International Editions.

John H. Mathews., 1995. NUMERICAL METHODS for Mathematics, Science and Engineering, 2nd Ed, 1992, Prentice Hall, Englewood Cliffs, New Jersey, 07632, U.S.A.

Risvik, K. M. and Michelsen, R., 2002. Search Engines and Web Dynamics. Computer Networks, (39): 289–302, June 2002.

Shkapenyuk, V. and Suel, T., 2002. Design and implementation of a high performance distributed web crawler. In Proceedings of the 18th International Conference on Data Engineering (ICDE), 357-368, San Jose, California. IEEE CS Press.

Yibei Ling and Jie Mi., 2004. An optimal trade-off between content freshness and refresh cost, Journal of applied probability, 2004, 41, (3): 721-734.

Zeinalipour-Yazti, D. and Dikaiakos, M. D., 2002. Design and implementation of a distributed crawler and filtering processor. In Proceedings of the Fifth Next Generation Information Technologies and Systems (NGITS), volume 2382 of Lecture Notes in Computer Science, 58–74, Caesarea, Israel. Springer.