Iterative methods for solving systems of linear equations with sparse matrices

Consider system of linear equations

\[ Ax = b, \quad (24) \]

where \( N \times N \) matrix \( A \)

- is sparse,

  - number of elements for which \( A_{ij} \neq 0 \) is \( O(N) \).

- Typical example: Poisson equation discretisation on \( n \times n \) mesh, \( (N = n \times n) \)

  - in average 5, nonzeros per \( A \) row
In case of direct methods, like **LU-factorisation**

- memory consumption (together with fill-in): $O(N^2) = O(n^4)$.
- flops: $\frac{2}{3} \cdot N^3 + O(N^2) = O(n^6)$.

**Banded matrix LU-decomposition**

- memory consumption (together with fill-in): $O(N \cdot L) = O(n^3)$, where $L$ is bandwidth
- flops: $\frac{2}{3} \cdot N \cdot L^2 + O(N \cdot L) = O(n^4)$. 
8.2 Jacobi Method

- Iterative method for solving (24)

- With given initial approximation $x^{(0)}$, approximate solution $x^{(k)}$, $k = 1, 2, 3, \ldots$ of (24) real solution $x$ are calculated as follows:

  - $i$-th component of $x^{(k+1)}$, $x_i^{(k+1)}$ is obtained by taking from (24) only the $i$-th row:

    $$A_{i,1}x_1 + \cdots + A_{i,i}x_i + \cdots + A_{i,N}x_N = b_i$$

  - solving this with respect to $x_i$, an iterative scheme is obtained:

    $$x_i^{(k+1)} = \frac{1}{A_{i,i}} \left( b_i - \sum_{j \neq i} A_{i,j}x_j^{(k)} \right) \quad (25)$$
The calculations are in essence parallel with respect to $i$ — no dependence on other components $x_j^{(k+1)}$, $j \neq i$. Iteration stop criteria can be taken, for example:

$$\left\| x^{(k+1)} - x^{(k)} \right\| < \varepsilon \quad \text{or} \quad k + 1 \geq k_{\text{max}},$$

(26)

- $\varepsilon$ — given error tolerance
- $k_{\text{max}}$ — maximal number of iterations

- memory consumption (no fill-in):
  - $N_A \neq 0$ — number of nonzeros of matrix $A$

- Number of iterations to reduce $\left\| x^{(k)} - x \right\|_2 < \varepsilon \left\| x^{(0)} - x \right\|_2$:

$$\# \text{IT} \geq \frac{2 \ln \varepsilon^{-1}}{\pi^2} (n + 1)^2 = O(n^2)$$
• flops/iteration \( \approx 10 \cdot N = O(n^2) \), \( \implies \)

\[
\#IT \cdot \frac{\text{flops}}{\text{iteration}} = Cn^4 + O(n^3) = O(n^4).
\]

Coefficient \( C \) in front of \( n^4 \) is:

\[
C \approx \frac{2 \ln \varepsilon^{-1}}{\pi^2} \cdot 10 \approx 2 \cdot \ln \varepsilon^{-1}
\]

• This is not very good at all... We need some better methods, because

– For LU-decomposition (banded matrices) we had \( C = 2/3 \)
8.3 Conjugate Gradient Method (CG)

Calculate $r^{(0)} = b - Ax^{(0)}$ with given starting vector $x^{(0)}$

for $i = 1, 2, \ldots$

solve $Mz^{(i-1)} = r^{(i-1)}$ # we assume here that $M = I$

$\rho_{i-1} = r^{(i-1)^T}z^{(i-1)}$

if $i == 1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1}p^{(i-1)}$

endif

$q^{(i)} = Ap^{(i)}$; $\alpha_i = \rho_{i-1}/p^{(i)^T}q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_ip^{(i)}$; $r^{(i)} = r^{(i-1)} - \alpha_iq^{(i)}$

check convergence; continue if needed

end
• memory consumption (no fill-in):

\[ N_{A \neq 0} + O(N) = O(n^2), \]

where \( N_{A \neq 0} \) – # nonzeros of \( A \)

• Number of iterations to achieve \( \| x^{(k)} - x \|_2 < \varepsilon \| x^{(0)} - x \|_2 \):

\[ \#IT \approx \frac{\ln \varepsilon^{-1}}{2} \sqrt{\kappa(A)} = O(n) \]

• Flops/iteration \( \approx 24 \cdot N = O(n^2) \), \( \implies \)

\[ \#IT \cdot \frac{\text{flops}}{\text{iteration}} = Cn^3 + O(n^2) = O(n^3), \]
where $C \approx 12 \ln \varepsilon^{-1} \cdot \sqrt{\kappa_2(A)}$.

$\implies C$ depends on condition number of $A$! This paves the way for preconditioning technique.
8.4 Preconditioning

Idea:
Replace \( Ax = b \) with system \( M^{-1}Ax = M^{-1}b \).
Apply CG to
\[
Bx = c, \tag{27}
\]
where \( B = M^{-1}A \) and \( c = M^{-1}b \).

But how to choose \( M \)?

Preconditioner \( M = M^T \) to be chosen such that

(i) Problem \( Mz = r \) being easy to solve

(ii) Matrix \( B \) being better conditioned than \( A \), meaning that \( \kappa_2(B) < \kappa_2(A) \)
Then
\[ \#IT(27) = O(\sqrt{\kappa_2(B)}) < O(\sqrt{\kappa_2(A)}) = \#IT(24) \]

but
\[ \frac{\text{flops}}{\text{iteration}}(27) = \frac{\text{flops}}{\text{iteration}}(24) + (i) > \frac{\text{flops}}{\text{iteration}}(24) \]

- We need to make a compromise!
- (In extreme cases \( M = I \) or \( M = A \))
- **Preconditioned Conjugate Gradients (PCG) Method**
  - obtained if to take in previous algorithm \( M \neq I \)
8.5 Preconditioner examples

Diagonal Scaling (or Jacobi method)

\[ M = \text{diag}(A) \]

(i) \[ \frac{\text{flops}}{\text{Iteration}} = N \]

(ii) \[ \kappa_2(B) = \kappa_2(A) \]

⇒ no large improvement to be expected
Incomplete LU-factorisation

\[ M = \tilde{L}\tilde{U}, \]

- \( \tilde{L} \) and \( \tilde{U} \) – approximations to actual factors \( L \) and \( U \) in \( LU \)-decomposition

  - nonzeros in \( L_{ij} \) and \( U_{ij} \) only where \( A_{ij} \neq 0 \) (i.e. fill-in is ignored in LU-factorisation algorithm)

(i) \[ \frac{\text{flops}}{\text{Iteration}} = O(N) \]

(ii) \( \kappa_2(B) < \kappa_2(A) \)

Some improvement at least expected!

\[ \kappa_2(B) = O(n^2) \]
Gauss-Seidel method

\[
\text{do } k=1,2,\ldots \\
\quad \text{do } i=1,\ldots,n \\
\quad \quad x_i^{(k+1)} = \frac{1}{A_{i,i}} \left( b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} A_{i,j} x_j^{(k)} \right) \\
\text{endo}
\]

endo

Note that in real implementation, the method is done like:

\[
\text{do } k=1,2,\ldots \\
\quad \text{do } i=1,\ldots,n \\
\quad \quad x_i = \frac{1}{A_{i,i}} \left( b_i - \sum_{j\neq i} A_{ij} x_j \right) \\
\text{endo}
\]

endo

But the preconditioner is not symmetric, which makes CG not to converge!
Symmetric Gauss-Seidel method

To get the symmetric preconditioner, another step is added:

\[
do \text{k}=1,2,\ldots \\
\quad \text{do } i=1,\ldots,n \\
\quad \quad x_i = \frac{1}{A_{i,i}} \left( b_i - \sum_{j\neq i} A_{ij} x_j \right) \\
\quad \text{enddo} \\
\text{enddo} \\
\text{enddo} \\
\text{enddo}
\]
9 Some examples

9.1 Google PageRank problem

9.1.1 Overview

WWW is a huge collection of data distributed around the globe, in constant change and growth.

<table>
<thead>
<tr>
<th># pages indexed by Google (estimate)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>May-June 2000</td>
<td>1 billion</td>
</tr>
<tr>
<td>November-December 2000</td>
<td>1.3 billion</td>
</tr>
<tr>
<td>July - August 2002</td>
<td>2.5 billion</td>
</tr>
<tr>
<td>November - December 2002</td>
<td>4 billion</td>
</tr>
<tr>
<td>January - February 2004</td>
<td>4.28 billion</td>
</tr>
<tr>
<td>November - December 2004</td>
<td>8 billion</td>
</tr>
<tr>
<td>August 2005</td>
<td>8.2 billion</td>
</tr>
<tr>
<td>January 2007</td>
<td>14 billion</td>
</tr>
</tbody>
</table>

Was roughly, doubling every 16 months.
• According to http://www.livescience.com/54094-how-big-is-the-internet.html
  
  – 4.66 billion Web pages online as of mid-March 2016
  – but this is only searchable web!

• Need really good tools for navigating, searching, indexing the information

(According to some estimates, number of pages indexed by search engines has not grown very much last years)
How does Internet look like?

Maps of the Internet (http://www.opte.org/maps/)

OK, these are just servers. Imagine, how would the WWW look like?
9.1.2 PageRank Problem Description


- one of the reasons why Google is so effective
- a method for computing the relative rank of web pages
- based on web link structure
- has become a natural part of modern search engines
- Also, a useful tool applied in many other search technologies, for example
  - Web spam detection [Z. Gyöngyi et al 2004]
  - crawler configuration
  - P2P trust networks [S. D. Kamvar et al 2003]
Surfing the web, going from page to page by randomly choosing an outgoing link

- can lead to dead ends (*dangling nodes*)
- cycles

Sometimes choosing simply a random page from the Web.

*Markov chain* or *Markov process*

The limiting probability that an infinitely dedicated random surfer visits any particular page is its *PageRank*.
9.1.4 Mathematical formulation of PageRank problem

Problem setup

\( W \) - set of web pages reachable in a chain following hyperlinks from a root page

\( G \) - corresponding \( n \times n \) connectivity matrix:

\[
g_{ij} = \begin{cases} 
1 & \text{if } \exists \text{ hyperlink } i \leftarrow j \\
0 & \text{otherwise.}
\end{cases}
\]

- \( G \) can be huge, is sparse, column \( j \) shows the links on \( j \)th page

- \# nonzeros in \( G \) - the total number of hyperlinks in \( W \)
Let $r_i$ and $c_j$ be the row and column sums of $G$:

\[ r_i = \sum_j g_{ij}, \quad c_j = \sum_i g_{ij}. \]

- $r_i$ - in-degree of the $i$th page
- $c_j$ - out-degree of the $j$th page.

Let $p$ - the probability that the random walk follows a link.

- A typical value is $p = 0.85$
- $1 - p$ is the probability that some arbitrary page is chosen
- $\delta = (1 - p)/n$ - probability that a particular random page is chosen.
Let $B$ be the $n \times n$ matrix with elements $b_{ij}$:

$$b_{ij} = \begin{cases} 
  \frac{p_{ij}}{c_j} + \delta & : c_j \neq 0 \\
  1/n & : c_j = 0 
\end{cases}$$

Notice that:

- $B$ is not sparse
- most of the values $= \delta$ (the probability of jumping from one page to another without following link)
- If $n = 4 \cdot 10^9$ and $p = 0.85$, then $\delta = 3.75 \cdot 10^{-11}$
- $B$ - the transition probability matrix of the Markov chain
- $0 < b_{ij} < 1$
- $\sum_{i=1}^{n} b_{ij} = 1, \forall j$
Matrix theory: *Perron-Frobenius theorem* applies:

∃! (within a scaling factor) solution \( x \neq 0 \) of the equation

\[
x = Bx.
\]

If the scaling factor is chosen such that \( \sum_i x_i = 1 \) then \( x \) is the state vector of the Markov chain and is Google’s PageRank; \( 0 < x_i < 1 \).
9.1.1 Power method

Algorithm: Power method

Input: Matrix $B$, initial vector $x$, threshold $\varepsilon$

Output: PageRank vector $y$

repeat
  $x \leftarrow Bx$
until $\|x - Bx\| < \varepsilon$

$y \leftarrow x / \|x\|$

In practice, matrix $B$ (or $G$) is never formed.
9.1.2 Transfer to a linear system solution

the first idea: the solution of the problem

\[ x = Bx \]

being equivalent to

\[ (I - B)x = 0 \]

But, the non-sparsity of \( I - B \)!

Is there a better way?
Yes: Note that

\[ B = pGD + ez^T, \tag{30} \]

where \( D \) - diagonal matrix

\[
d_{jj} = \begin{cases} 
1/c_j & : c_j \neq 0 \\
0 & : c_j = 0
\end{cases}, \quad e = \begin{bmatrix} 1 \\
1 \\
\vdots \\
1 \end{bmatrix}, \quad z_j = \begin{cases} 
\delta & : c_j \neq 0 \\
1/n & : c_j = 0
\end{cases}
\]

- \( ez^T \) - rank-one matrix - the random choices of Web pages that do not follow links.
The equation

\[ x = Bx \]

is becoming thus due to (30):

\[ x = (pGD + ez^T)x \]

\[ x - pGDx = e_{z^T}x \]

\[ (I - pGD)x = \gamma e \]
we get the system of linear equations to solve:

\[ Ax = e \]  

(We temporarily take \( \gamma = 1 \).) After solution of (31), the resulting \( x \) can be scaled so that \( \sum_i x_i = 1 \) to obtain PageRank.

Note that the matrix \( A = I - pGD \) is

- sparse
- nonsingular, if \( p < 1 \)
- nonsymmetric
- huge in size