Numerical Algorithms for Visual Computing 2008/09

Example Solutions for Assignment 9

Problem 1 (Getting symmetric)

1. As a reminder, the original matrices for the standard Gauss-Seidel method are given by

$$\begin{aligned} M_{\text{GS}} &= -(D+L)^{-1}U && (1)\\ N_{\text{GS}} &= (D+L)^{-1}, && (2) \end{aligned}$$

so by plugging this into our equations (3) and (4), we will get

$$\begin{aligned} x_{m+1} &= -(D+U)^{-1}L(-(D+L)^{-1}Ux_m+(D+L)^{-1}b)+(D+U)^{-1}b\\ &= \underbrace{(D+U)^{-1}L(D+L)^{-1}U}_{=M_{\text{SGS}}}x_m-(D+U)^{-1}L(D+L)^{-1}b+(D+U)^{-1}b\\ &= M_{\text{SGS}}x_m+(D+U)^{-1}(-L(D+L)^{-1}+I)b\\ &= M_{\text{SGS}}x_m+(D+U)^{-1}(-L(D+L)^{-1}+(D+L)(D+L)^{-1})b\\ &= M_{\text{SGS}}x_m+(D+U)^{-1}(-L+D+L)(D+L)^{-1}b\\ &= M_{\text{SGS}}x_m+\underbrace{(D+U)^{-1}\overset{-1}{D}(D+L)^{-1}}_{N_{\text{SGS}}}b \end{aligned}$$

So we have

$$\begin{aligned} M_{\text{SGS}} &= (D+U)^{-1}L(D+L)^{-1}U && (3)\\ N_{\text{SGS}} &= (D+U)^{-1}D(D+L)^{-1}. && (4) \end{aligned}$$

2. Similarly to the case of the SGS-case, we take the original matrices for the SOR-method, i.e.

$$\begin{aligned} M_{\text{SOR}} &= (D+\omega L)^{-1}[(1-\omega)D-\omega U] && (5)\\ N_{\text{SOR}} &= \omega(D+\omega L)^{-1}. && (6) \end{aligned}$$

Again, analogously, we derive a reverse method of SOR, i.e.

$$\begin{aligned} M_{\text{RSOR}} &= (D+\omega U)^{-1}[(1-\omega)D-\omega L] && (7)\\ N_{\text{RSOR}} &= \omega(D+\omega U)^{-1}. && (8) \end{aligned}$$

As in the first exercise, we plug both together, giving us

$$x_{m+1} = M_{\text{RSOR}}M_{\text{SOR}}x_m+(M_{\text{RSOR}}N_{\text{SOR}}+N_{\text{RSOR}})b \qquad (9)$$

We compute now the matrices directly.

$$\begin{aligned}
M_{SSOR} &= M_{\mathrm{RSOR}}M_{\mathrm{SOR}} = (D+\omega U)^{-1}[(1-\omega)D-\omega L](D+\omega L)^{-1}[(1-\omega)D-\omega U] \\
N_{SSOR} &= M_{\mathrm{RSOR}}N_{\mathrm{SOR}} + N_{\mathrm{RSOR}} \\
&= (D+\omega U)^{-1}[(1-\omega)D-\omega L]\omega(D+\omega L)^{-1} + \omega(D+\omega U)^{-1} \\
&= \omega(D+\omega U)^{-1}[(1-\omega)D-\omega L](D+\omega L)^{-1} + I] \\
&= \omega(D+\omega U)^{-1}[(1-\omega)D-\omega L](D+\omega L)^{-1} + (D+\omega L)(D+\omega L)^{-1}] \\
&= \omega(D+\omega U)^{-1}[(1-\omega)D-\omega L + (D+\omega L)](D+\omega L)^{-1} \\
&= \omega(D+\omega U)^{-1}[D-\omega D+D](D+\omega L)^{-1} \\
&= \omega(2-\omega)(D+\omega U)^{-1}D(D+\omega L)^{-1}
\end{aligned}$$

Problem 2 (Preposterous in Hilbert's space)

1. The solution is trivially a vector $(1,\dots,1)^\top$.

2. For this, we compute directly the condition number with the given functions of scilab. With the function `spec`, one can get a vector containing all eigenvalues of a given matrix. With the `min` and `max`-functions respectively we can compute the minimal and maximal eigenvalues. Alternatively, one can use the function `cond` which directly computes $\|A\|\cdot\|A^{-1}\|$. As we are dealing with a numerically difficult matrix, this matrix may lead to highly unstable results in the computations. The reason is the fact that the matrix is very close to being singular. This leads to numerical problems.

Size	`cond`	λ_1/λ_n
2	19.281470	19.281470
3	524.056778	524.056778
5	476607.250242	476607.250241
10	16025285352250.179688	16026572276505.589844
15	1105429325591602048	-32987464356204376
25	7055851363138485248	-51047659168545928

3. We can use the properties of the CG-method, i.e. we only iterate at most n times for a $n\times n$-Matrix. Furthermore, we have again the problem of the bad-posedness of the matrix, so we may have problems computing $x - A^{-1}b$. For this, we can use the $(1,\dots,1)^\top$ directly.

Size	$\lVert x - A^{-1}b \rVert$	$\lVert x - (1, \dots, 1)^\top \rVert$
2	0	0
3	0	0
5	0.003565	0.3565
10	0.000824	0.000167
15	12.000044	0.000072
25	159.999986	0.000021

4. We use in this part of the exercise the preconditioner that we computed in exercise part 1, i.e. $P = N_{\text{SGS}}$.

Size	$\lVert x - A^{-1}b \rVert$	$\lVert x - (1, \dots, 1)^\top \rVert$
2	0	0
3	0	0
5	0	0
10	0.053069	0.053069
15	11.981658	0.047450
25	160.005963	0.042730

Here, we see that the error is even worse than with the standard CG-method. However, if we use another preconditioner, for example the Jacobi preconditioner, then we end up with similar results than with the standard CG method. This is due to the size of the condition number. As we know, as a rule of thumb, one should use for a preconditioner a matrix with a relative small condition number. It turns out that the condition number from the Jacobi method is much better than the Symmetric Gauss-Seidel method.
