Numerical Algorithms. Assignment 3.
Krukovich Palina, a12134146

Introduction

The main goal of this part is to get experience with conjugate gradient method and investigate the impact of different preconditioners on its convergence.

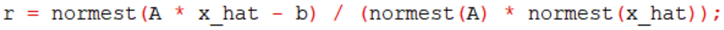
Octave version: 6.3.0.

For testing CG and PCG algorithms I used the limit for residual of $tol = 10^{-7}$, and the maximum number of iterations $maxit = 100000$ to make sure that the convergence can be met.

I used three matrices provided in the assignment: nos5, nos6 and s3rmt3m3. They are sparse, symmetric, and positive-definite which means that they are suitable for the Conjugate Gradients method.

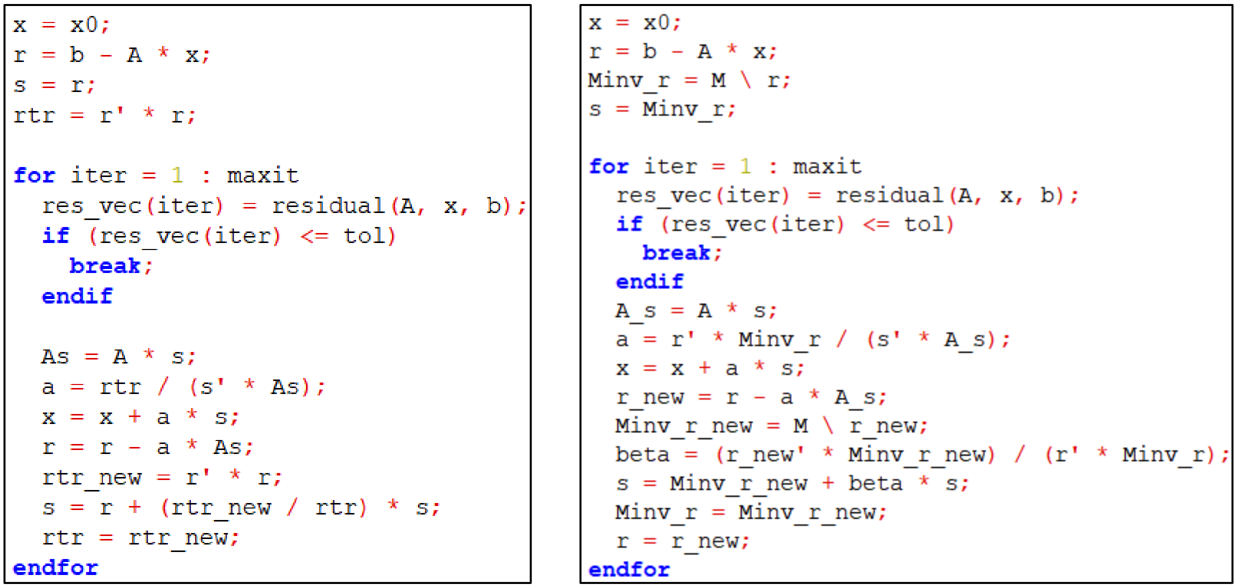
Optimization

Both algorithms work well enough in terms of runtime, but I noticed that the residual method works really slow in its original form and because it repeats in each iteration it slows down the performance of the whole algorithm. The optimization that I used for the residual is the use of norm estimation instead of exact calculation. The function normest computes the estimate of a 2nd norm and is intended to be used with sparse matrices according to the documentation.



```
r = normest(A * x_hat - b) / (normest(A) * normest(x_hat));
```

For CG and PCG optimization I tried to perform time-consuming calculations once per iteration. For CG algorithm the number of multiplications As_k and $r_k^T r_k$ can be reduced. First, we compute $r_0^T r_0$ before the cycle, then use it during the iteration and compute only $r_{k+1}^T r_{k+1}$. The same for PCG, we compute As_k and $M^{-1} r_k$ once. The implementations is shown on the Picture 1.



```
x = x0;
r = b - A * x;
s = r;
rtr = r' * r;

for iter = 1 : maxit
  res_vec(iter) = residual(A, x, b);
  if (res_vec(iter) <= tol)
    break;
  endif

  As = A * s;
  a = rtr / (s' * As);
  x = x + a * s;
  r = r - a * As;
  rtr_new = r' * r;
  s = r + (rtr_new / rtr) * s;
  rtr = rtr_new;
endfor
```

```
x = x0;
r = b - A * x;
Minv_r = M \ r;
s = Minv_r;

for iter = 1 : maxit
  res_vec(iter) = residual(A, x, b);
  if (res_vec(iter) <= tol)
    break;
  endif
  A_s = A * s;
  a = r' * Minv_r / (s' * A_s);
  x = x + a * s;
  r_new = r - a * A_s;
  Minv_r_new = M \ r_new;
  beta = (r_new' * Minv_r_new) / (r' * Minv_r);
  s = Minv_r_new + beta * s;
  Minv_r = Minv_r_new;
  r = r_new;
endfor
```

Picture 1 – CG and PCG algorithms

The final performance of the algorithms is described in table 1.

We can see that algorithms always perform better for nos5 and nos6 matrices than for s3rmt3m3 matrix. It happens because of multiple reasons. First, the third matrix is almost 10 times larger than the others. Also, it has much more non-zero elements, which slows down the multiplications, and it is the most ill-conditioned.

The nos6 matrix showed the best results in terms of runtime. It has only 3255 zero-elements totally. And it is the smallest matrix, however it isn’t the most well-conditioned.

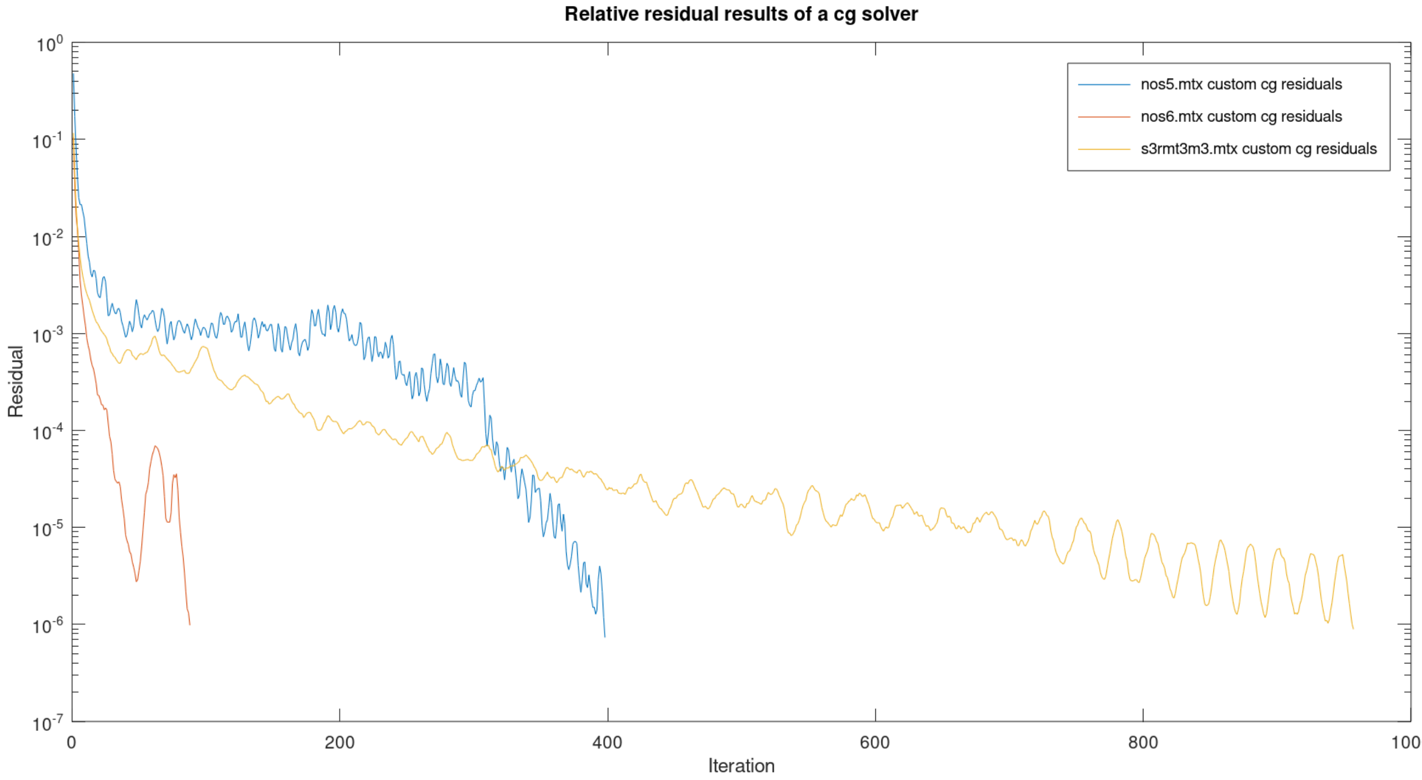
The best performance I got using the PCG algorithm with block-diagonal preconditioner (subblock sizes of 25 and 100) for all three matrices. The worst performance has been met using CG method without preconditioning.

	nos5, s		nos6, s		s3rmt3m3, s	
	time, s	iterations	time, s	iterations	time, s	iterations
CG	1.37	398	0.24	88	199.49	957
PCG with diagonal preconditioner	0.77	221	0.15	54	95.17	435
PCG with Block-diagonal (block Jacobi) preconditioner, subblocks of size 5	0.85	225	0.12	43	41.09	198
PCG with Block-diagonal (block Jacobi) preconditioner, subblocks of size 25	0.80	180	0.11	38	24.38	113
PCG with Block-diagonal (block Jacobi) preconditioner, subblocks of size 100	0.60	129	0.08	25	25.64	116
PCG with Incomplete Cholesky factorization	0.78	107	0.09	27	25.60	102
PCG with Incomplete Cholesky factorization, threshold 0.3	0.75	210	0.14	49	61.87	297
PCG with Incomplete Cholesky factorization, threshold 0.5	0.77	220	0.14	54	76.26	374
PCG with Incomplete Cholesky factorization, threshold 0.7	0.74	220	0.15	54	80.71	393

Table 1 – Runtime performance of CG and PCG algorithms

Part 1: CG

The CG algorithm performed the worst in terms of runtime. It also took the largest number of iterations to proceed. The more ill-conditioned matrices are the slower the improvement in path searching is happening (convergence). All the three matrices are ill-conditioned, so we need to use the preconditioning to find the matrix M, which has a smaller condition number than A.



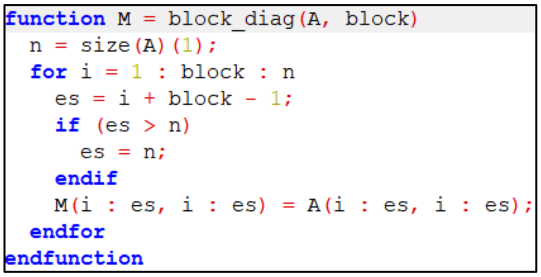
Picture 2 – Relative residual results of a CG algorithm

Part 2: PCG

For the PCG algorithm we use preconditioners. This way we can get a smaller number of iterations and find the solution faster. We choose matrix M so that $M^{-1}A$ has a better condition.

In the PCG algorithm we have a costly inverse operation present. We meet M inverse in the statement $M^{-1}r_k$. To speed up the computations, we don't compute the inverse explicitly, but we solve this system and get $M \backslash r_k$. It is important because the performance of the algorithm depends on how easy we can solve this linear system. The preconditioner helps only if we can solve this system fast because we do it in every iteration. For all the preconditioners used in this homework this system is solved very fast.

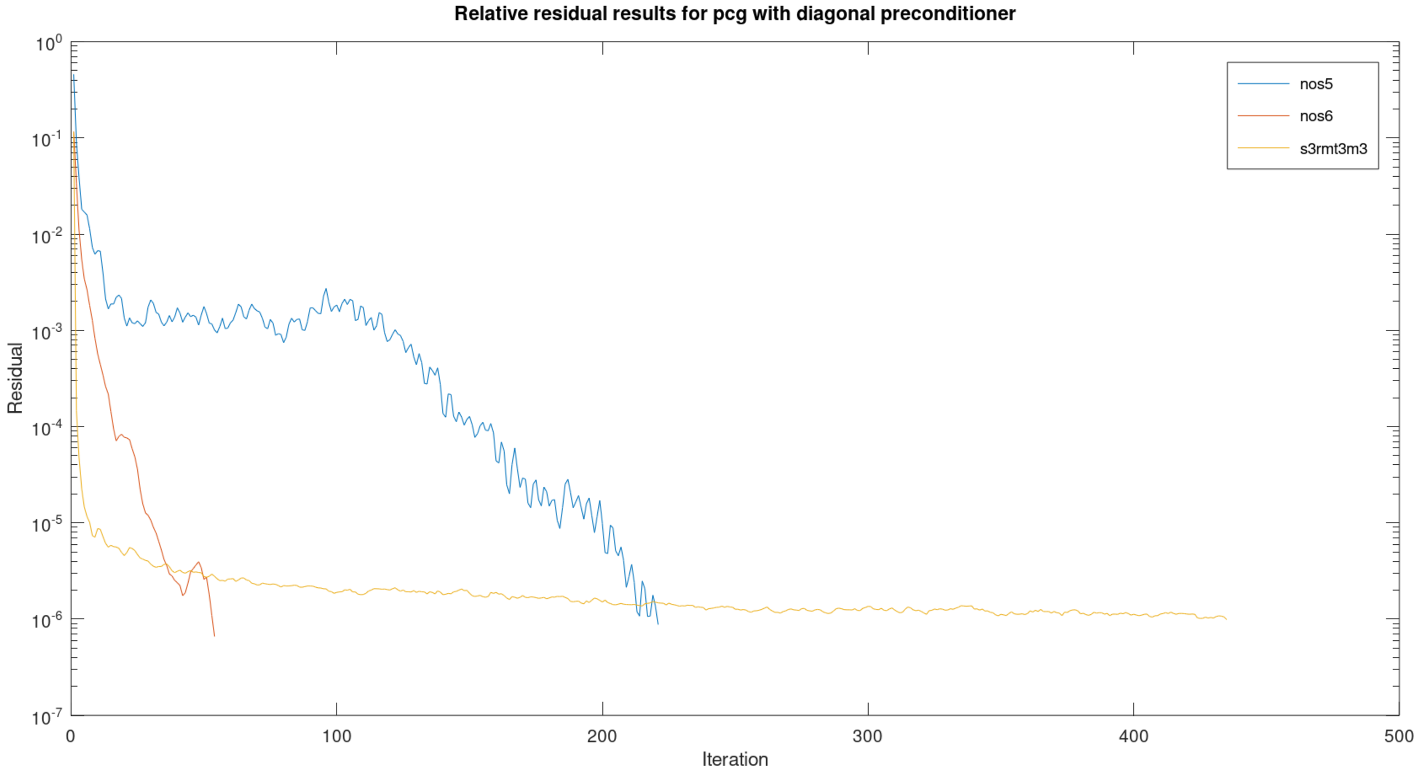
For diagonal preconditioner we apply the diag Octave function. To compute the preconditioner for block-diagonal preconditioner I used the method shown on Picture 3, which transfers diagonal blocks of configurable size containing elements from matrix A to matrix M. And for Cholesky factorization I used the Octave function ichol.



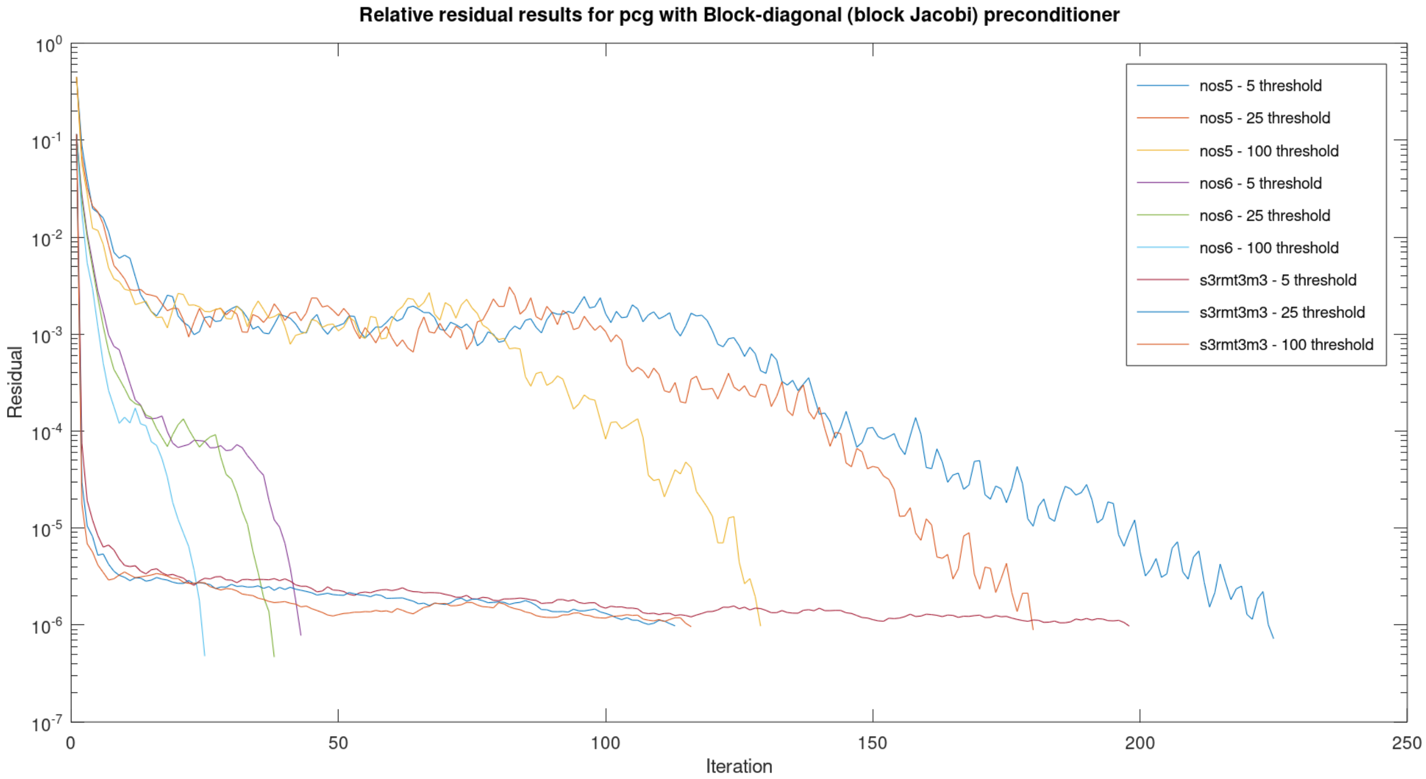
```
function M = block_diag(A, block)
  n = size(A)(1);
  for i = 1 : block : n
    es = i + block - 1;
    if (es > n)
      es = n;
    endif
    M(i : es, i : es) = A(i : es, i : es);
  endfor
endfunction
```

Picture 3 – method for computing the block-diagonal preconditioner

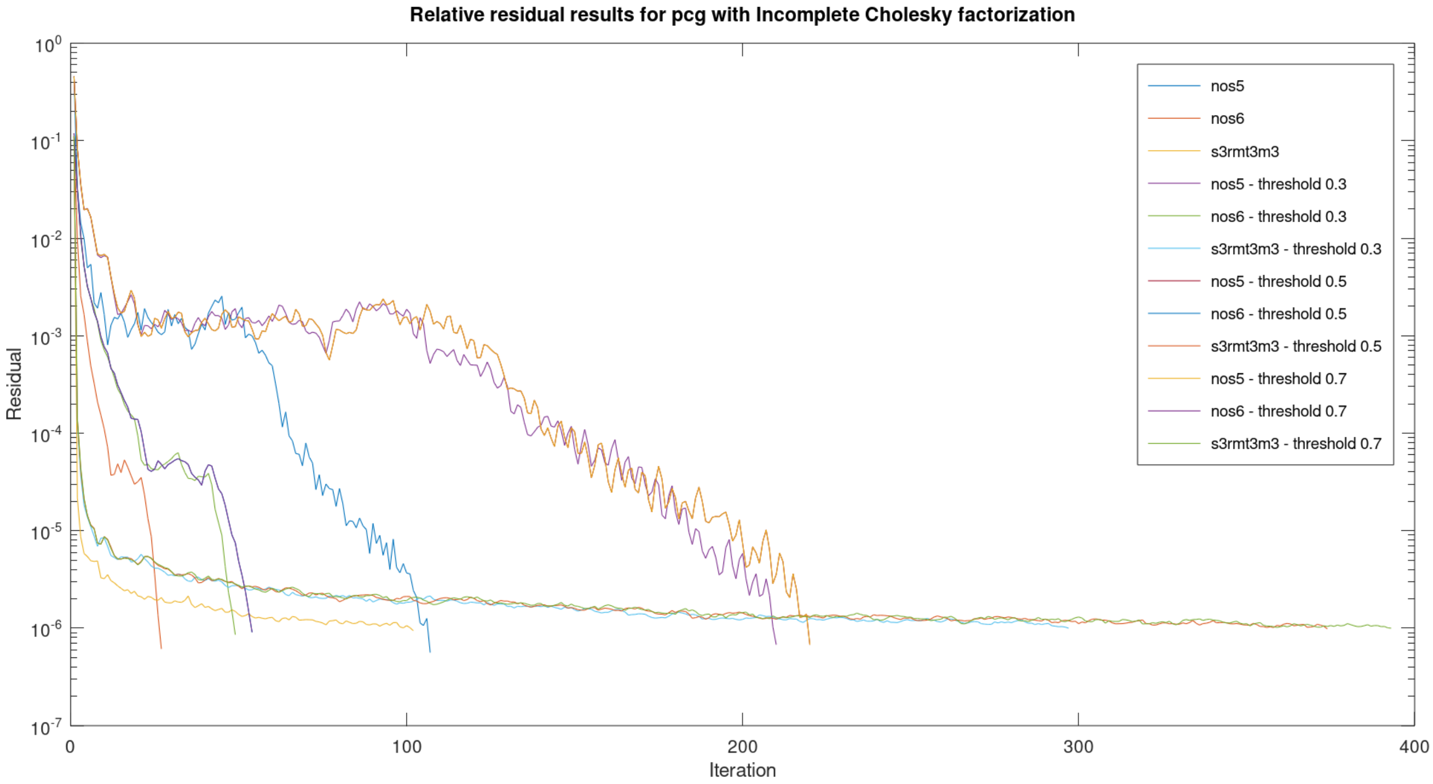
The PCG method determines the type of preconditioner in the very beginning. If only matrix M1 is presents than it applies the block-diagonal preconditioner. If only matrix M2 is present it applies the diagonal preconditioner. And if both matrices M1 and M2 are present it applies the Cholesky factorization by assuming that M1 and M2 are L and L^T, so it just multiplies them. The resulted relative residuals are presented on the pictures below.



Picture 4 - Relative residual results of a PCG algorithm with diagonal preconditioner



Picture 5 – Relative residual results of a PCG algorithm with block-diagonal preconditioner



Picture 6 – Relative residual results of a PCG algorithm with incomplete Cholesky factorization

We can see that s3rmt3m3 matrix is very fast to converge up to a certain residual level and then it slows down, and it takes a lot of iterations to finally converge. On the other hand, other two matrices perform better in these terms, and in most of the cases converge in a few iterations.

The diagonal preconditioner reduced the number of iterations for each matrix almost in two times, as well as the time spent on computations. Such a result is reached because matrices have most of their non-zero elements outside the main diagonal and this preconditioner really changes them and makes better conditioned.

PCG with block-diagonal preconditioner (subblocks of sizes 5 and 25) worked almost the same as diagonal one, because makes almost the same changes to the matrix by leaving small blocks of elements on the main diagonal. Block-diagonal preconditioner with a bigger block size (100) performed the best for all matrices in terms of both runtime and number of iterations. As we can see on the Picture 5 the convergence was fast. All those three matrixes have their elements around the main diagonal and this preconditioner filtered them well.

PCG with Cholesky factorization without threshold (0) worked better on these matrices than with the threshold (0.3, 0.5, 0.7) and its performance is very close to the block-diagonal preconditioner with size of 100 for blocks.

We may conclude that Conjugate Gradients method needs to be used with preconditioning. Otherwise, the convergence is very slow. We can determine the best preconditioner depending on the properties of matrix A:

- matrix size
- number of zero and non-zero elements
- matrix well- or ill-conditioned
- the distribution of elements inside the matrix
- what are the values of matrix elements

Some preconditioners may be suitable for a lot of cases, some could be specific. So, the choice of preconditioner depends on the data and area of use.