A multilevel AINV preconditioner

Gérard Meurant*

2002

Abstract

In this paper we describe an algebraic multilevel extension of the approximate inverse AINV preconditioner for solving symmetric positive definite linear systems $Ax = b$ with the Preconditioned Conjugate Gradient method. The smoother is the approximate inverse M and the coarse grids and the interpolation operator are constructed by looking at the entries of M. Numerical examples are given for problems arising from discretization of partial differential equations.

1 Introduction

In this paper we describe an algebraic multilevel extension of the AINV preconditioner (c.f. [1]) for solving symmetric positive definite linear systems $Ax = b$ with the Preconditioned Conjugate Gradient (PCG) method.

We would like to explore the idea that important influences between unknowns in a linear system are given by the inverse of the matrix and that these can be more or less captured by an approximate inverse like the one defined by the AINV algorithm. This can be used to define the coarse meshes and the interpolation operators. The method we present here is purely algebraic, the only inputs being the matrix of the linear system and the right hand side.

An important issue for solving very large problems on Tflops scale parallel computers is scalability. One would like to have the computer time constant when the problem size per processor is fixed and the number of processors increases which means that the dimension of the problem is increasing. When using an iterative method like PCG this implies that the number of iterations must be constant when the problem size is increased. But this is not enough since we also need to have a number of operations proportional to the problem size. We will see that the multilevel extension of AINV leads to an algorithm which is scalable for some problems arising from discretization of second order partial differential equations and almost scalable for some other problems. Numerical comparisons with other multilevel algorithms are given in another paper [5].

*CEA/DIF, DCSA/ED, BP 12, 91680 Bruyères le Chatel, France. (gerard.meurant@cea.fr)

The multilevel AINV preconditioner (which we denote by MLAINV) uses the same design principles as the Algebraic Multigrid algorithm (AMG), see [3], [6], [7]. After some smoothing steps, the equation for the error with the residual as the right hand side is solved recursively on a coarser grid. The definition of the coarse grids uses a set (or a matrix) of dependencies. This is constructed by looking at the approximate inverse at a given grid level. An influence set is defined for each unknown as the "transpose" of the set of dependencies. The fine and coarse nodes for each level are found on this basis. Then, knowing the fine and coarse nodes, interpolation weights are computed using the entries of the approximate inverse. The restriction R is the transpose of the interpolation (prolongation) matrix P and the next coarse matrix is defined by a Galerkin formula as $A_c = RAP$. As we said before, the method also uses a smoothing operator which is a Richardson–like iteration defined using the approximate inverse. An iteration (V–cycle) of the MLAINV recursive preconditioner is the following starting from the zero vector:

1. if we are on the coarsest level, solve exactly by Gaussian elimination, otherwise
2. do ν iterations of smoothing
3. restrict the residual r to $r_c = Rr$
4. recursively solve $A_c e_c = r_c$
5. interpolate e_c to $e = Pe_c$
6. add the correction e to the current iterate
7. do ν iterations of smoothing

More generally we can introduce a parameter γ and replace the solve step by doing γ iterations of the same algorithm with one level less. Choosing $\gamma = 1$ is the V–cycle just described and having $\gamma = 2$ is denoted as a W–cycle.

For the MLAINV preconditioner we use the same principles as AMG, see [6], except that:

i) the smoother is a Richardson iteration with an AINV preconditioner,

ii) we compute the influence matrix by using the approximate inverse given by AINV,

iii) the interpolation and restriction operators are defined by entries of the (factors of the) approximate inverse.

In the next sections we describe the algorithm in more details and give some numerical results which have been obtained on several elliptic problems as well as more general linear systems using this method and some minor variations.

2 The AINV preconditioner

The Benzi, Meyer and Tuma [1] approximate inverse M generates a decomposition

$$M = ZD^{-1}Z^T$$

where the matrix Z is upper triangular with $1's$ on the diagonal and D is diagonal. The method is based on computing A–orthogonal vectors using a Gram–Schmidt A–orthogonalization starting from the columns of the identity matrix. To preserve sparsity some elements are thrown away at each step of the algorithm. This is done either by using a given threshold τ or according to the positions of the entries. Another possibility is to retain only the q largest elements of each column of Z satisfying the threshold criterion. If nothing is neglected the algorithm delivers the L^TDL factorization of A^{-1} and consequently the factors are usually dense. In this algorithm, Z is generally computed by columns. The details of the algorithm are the following where z_i denotes the i–th column of Z:

$Z = I$
$d_1 = a_{1,1}$
for $i = 2, \ldots, n$
  drop the entries $z_{k,i}$ such that $|z_{k,i}| \leq \tau \|a_i\|_\infty$
    for $j = i, \ldots, n$
      $d_j = a_i^T z_j$
    end
    for $j = i+1, \ldots, n$
      $z_j = z_j - \frac{d_j}{d_i} z_i$
    end
end

The parameter τ defines which elements are kept. When τ gets smaller we keep more fill–in. Note this is not exactly the threshold criterion used in [1]; hence the values of the threshold could be slightly different from those which are suggested from numerical experiments in [1]. We can use this approximate inverse as a preconditioner in the PCG algorithm. At each iteration we have to solve a linear system with the residual r as a right hand side. This is done as

$$z = Mr,$$

by matrix $\times$ vector multiplies which are a parallel operation. One may think that the AINV is not fully parallel because the construction phase is sequential. However, the construction phase can also be parallelized by using domain decomposition–like orderings. There exists also some other approximate inverses which are computed by minimizing some norms but they are usually not guaranteed to be positive definite or even symmetric, see [4] for a summary.

The AINV preconditioner is feasible for H–matrices (see [4] for a definition) and it can be seen that for problems arising from second order partial differential equations the condition number $\kappa(MA)$ is proportional to h^{-2} where h is the mesh size when the non zero structure of M is comparable to that of A.

Therefore, when the problem gets larger the number of iterations of the PCG algorithm grows. The storage that is used for M also depends on the choice of τ. The smaller is τ, the larger is the storage. There exists a more robust version of this approximate inverse denoted as SAINV which can be used for any matrix, see [2].

For the Poisson equation in a square with five point finite differences with our chosen criterion, a value of $\tau = 0.06$ gives a matrix Z whose non–zero structure is the same as the upper triangular part of A. This leads to a matrix M with seven non–zero diagonals. Using $\tau = 0.07$ leads to a diagonal matrix. Of course, the smaller is τ, the smaller is the number of iterations since when we keep all the fill, we have the exact factorization and then we only do one iteration. Therefore, there is a balance between the number of iterations and the number of operations per iteration such that there exists a value of τ that gives the smallest number of operations. Numerical experiments with AINV show that it does not pay to retain too much fill since the minimum number of operations is often given when Z has the same structure as the upper triangular part of A. However, being able to change the parameter τ gives an interesting degree of freedom that allows sometimes to solve difficult problems by keeping more fill–in.

3 The multilevel preconditioner MLAINV

We are going to look at the different components of the multilevel algorithm: the smoother, the influence matrix, the coarsening and interpolation algorithms.

3.1 The smoother

On a given level, we will use the matrix $M = ZD^{-1}Z^T$ from AINV as a smoother in a Richardson iteration defined as

$$x^{k+1} = x^k + M(b - Ax^k),$$

when solving $Ax = b$. Let us first consider the multilevel preconditioner in an abstract way by looking at a two–grid algorithm using one step of pre–smoothing and one step of post–smoothing . At each PCG iteration we start from $x^0 = 0$. Applying one step of smoothing amounts to computing

$$\bar{x} = Mb.$$

We compute the residual

$$\bar{r} = b - AMb = (I - AM)b.$$

Restricting the residual $\bar{r}$, solving exactly the coarse problem (defined by the coarse matrix RAP) and interpolating back to the fine level amounts to compute

$$P(RAP)^{-1}R(I - AM)b.$$

This is added to the current iterate $\bar{x}$ giving

$$Mb + P(RAP)^{-1}R(I - AM)b.$$

Finally another step of smoothing is applied leading to

$$x^1 = [M + M(I - AM) + (I - MA)(P(RAP)^{-1}R)(I - AM)]b.$$

Therefore, the two–grid preconditioner $\tilde{M}_0$ is defined as

$$\tilde{M}_0 = M + M(I - AM) + (I - MA)(P(RAP)^{-1}R)(I - AM).$$

More generally, we denote by an index 0 quantities on the finest level and by $1, 2, \ldots$ the coarse levels. Then, it is not difficult to show that we have the following recursive formula for the preconditioner

$$\tilde{M}_l = M_l + M_l(I - A_l M_l) + (I - M_l A_l)(P_l \tilde{M}_{l+1} R_l)(I - A_l M_l),$$

$$A_{l+1} = R_l A_l P_l,$$

and on the coarsest level L, $\tilde{M}_{l+1}$ is replaced by A_L^{-1}.

Matrices M_l from AINV are symmetric positive definite. Obviously $\tilde{M}_l$ is symmetric. The first thing we have to do is to check if $\tilde{M}_l$ is positive definite. We have the following result.

Theorem 1 *Suppose $\lambda_{min}(I - A_l M_l) > -1, \forall l$ then $\tilde{M}_0$ is positive definite as well as all the matrices $\tilde{M}_l, l = 1, \ldots$. Moreover, $\tilde{M}_0 A$ has real eigenvalues located in $]0, 1]$, 1 being a multiple eigenvalue.*

Proof.

The proof goes by induction. Suppose first that we have only two levels. Then,

$$\tilde{M}_0 = M_0 + M_0(I - AM_0) + (I - M_0 A)(P_0(R_0 A P_0)^{-1} R_0)(I - AM_0).$$

Since $M_0 + M_0(I - AM_0) = M_0(2I - AM_0)$, our hypothesis implies that the eigenvalues of $2I - AM_0$ are strictly positive. Therefore the sum of the first two terms in the right hand side is positive definite. It is obvious that the third term is positive semidefinite which proves that $\tilde{M}_0$ is positive definite. The same argument applies for the other levels.

Now let us prove that the eigenvalues of $\tilde{M}_0 A$ belong to $]0, 1]$. Let us look at

$$T = A^{-1} - \tilde{M}_0.$$

Noticing that

$$A^{-1} - M_0 = (I - M_0 A)A^{-1}$$

and

$$M_0(I - AM_0) = M_0 A(I - M_0 A)A^{-1}$$

we have that

$$T = (I - M_0 A)Q_0(I - M_0 A)A^{-1} = U_0 A^{-1},$$

where $Q_0 = I - P_0 \tilde{M}_1 R_0 A$.

Since the matrix T is symmetric we have

$$T = T^T = A^{-1} U_0^T = U_0 A^{-1}.$$

Then,

$$(Tx, x) = (A^{-1}x, U_0^T x) = (A^{-1}x, AU_0 A^{-1}x) = (y, AU_0 y).$$

But

$$AU_0 = A(I - M_0 A)Q_0(I - M_0 A) = (I - AM_0)AQ_0(I - M_0 A).$$

Therefore, all we have to do is to check if AQ_0 is positive semidefinite because then we will have T positive semidefinite and

$$(\tilde{M}_0 x, x) \leq (A^{-1}x, x), \quad x \neq 0$$

and this proves that the eigenvalues of $\tilde{M}_0 A$ are less or equal to 1.

The matrix AQ_0 can be written as

$$AQ_0 = A^{\frac{1}{2}}(I - A^{\frac{1}{2}} P_0 \tilde{M}_1 R_0 A^{\frac{1}{2}})A^{\frac{1}{2}}.$$

Therefore, we have only to consider $I - A^{\frac{1}{2}} P_0 \tilde{M}_1 R_0 A^{\frac{1}{2}}$.

We first consider the two grid case. From [4] page 544, it is known that the matrix AQ_0 has only 0 and 1 as eigenvalues. Hence the same is true for AU_0. Moreover, 1 is an eigenvalue. The matrix M_0 is such that $I - M_0 A$ is non singular. Then, for each $y \in$ Ker Q_0 there exists a z such that $(I - M_0 A)z = y$. We have

$$\tilde{M}_0 A = M_0 A + [P_0 (R_0 A P_0)^{-1} R_0 A + M_0 A Q_0](I - M_0 A).$$

Therefore,

$$\tilde{M}_0 Az = M_0 Az + y = M_0 Az + (I - M_0 A)z = z.$$

The vector z is an eigenvector of $\tilde{M}_0 A$ with eigenvalue 1. The multiplicity of the eigenvalue 1 is the number of independent vectors that span Ker Q_0.

In the multilevel case, we are looking for λ such that

$$(I - A_l^{\frac{1}{2}} P_l \tilde{M}_{l+1} R_l A_l^{\frac{1}{2}})y = \lambda y.$$

We multiply this equation by $R_l A_l^{\frac{1}{2}}$ to get

$$(R_l A_l^{\frac{1}{2}} - A_{l+1} \tilde{M}_{l+1} R_l A_l^{\frac{1}{2}})y = \lambda R_l A_l^{\frac{1}{2}} y.$$

This is because $A_{l+1} = R_l A_l P_l$. Since by the induction hypothesis $I - A_{l+1}\tilde{M}_{l+1}$ is positive definite, this proves that $A_l Q_l$ is positive semidefinite. Moreover,

inductively, we can see that there are non trivial vectors in the null space of Q_l and this shows that 1 is an eigenvalue. □

Note that this result is true whatever the choice of M_l, R_l and P_l. Therefore the condition number of the preconditioned matrix with the multigrid preconditioner depends only on the smallest eigenvalue which of course depends on the way the smoother and the interpolation are chosen. Experimentally we have checked that for the problems we are going to use for the numerical experiments the hypotheses of the theorem are fulfilled.

We also remark that the multilevel preconditioner is a dense matrix which is never explicitly computed nor stored since it is implicitly applied to a vector in the PCG algorithm.

3.2 The influence matrix

An important part of the multilevel algorithm is to decide which unknowns correspond to the fine "nodes" (denoted also as points or unknowns) and which to the coarse nodes. Hence, the set $\mathcal{N} = \{1, \ldots, n\}$ of the unknowns is split into two sets $\mathcal{N} = F \cup C$.

For each unknown i we define the set of dependencies S_i and an influence matrix S whose rows are the vectors S_i's padded with zeros for indices which are not in the set of dependencies. Defining this set can be done in many ways. Rather than using an influence matrix given by the entries of A as in AMG (see [6]), it seems natural to measure the influences between the points by the inverse of A since this precisely describes how the unknowns are linked together in obtaining the solution of the problem. However, since we only have at our disposal the approximate inverse M from AINV, we can define

$$S_i^M = \{j \in \mathcal{N}, j \neq i |\ m_{i,j} \neq 0\}.$$

Note that S_i^M depends on the value of τ that is chosen in AINV. This choice will be denoted as 'm' in the sequel. Generally we do not want to compute M since it is given in factored form and the solve at every iteration can be done with multiplications with Z and Z^T. Thus we will explore the possibility of defining the influence matrix as

$$S_i^Z = \{j \in \mathcal{N}, j \neq i |\ n_{i,j} \neq 0\}.$$

Let Q be a diagonal matrix whose diagonal elements are the square roots of those of D^{-1} and $\tilde{Z} = ZQ$. Then, the matrix N is defined as $N = \tilde{Z} + \tilde{Z}^T - Q$. This choice will be denoted as 'z'.

We will see in the numerical experiments that, depending on the value of τ, this strategy can lead to coarse grids with very few nodes and it will be sometimes difficult to obtain a meaningful interpolation. Therefore, we will also explore a strategy where we use two approximate inverses, one with a threshold τ_1 to compute the coarse grid and another one with a threshold $\tau_2 \leq \tau_1$ as a smoother. The first decomposition can be easily obtained from the second one. This would allow us to be able to smooth more on difficult problems without leading to grids which are too coarse.

3.3 The coarsening algorithm

Once S_i is fixed by any of the previous methods, there are different ways we can follow to decide which are the F and C points. This is also related to the type of interpolation we are going to choose.

What we are going to denote as the "standard" ('st') coarsening algorithm (see [3], [6]) is based on two principles:

1. for each $i \in F$, each node $j \in S_i$ should either be in C or should depend on at least one point in C_i which is the set of coarse points which are going to be used for the interpolation of i.

2. C should be (as most as possible) a maximal subset with the property that no C point depends on another C point.

The first criterion tends to increase the number of C points. The second one is used to limit the number of points in the coarse grid. The standard coarsening algorithm (see [7]) is defined by two passes. However, we will only going to use the first one. The first pass uses weights w_i which are the number of points that depend on i. One step of the algorithm is the following:

1. choose the first point i with maximal weight as a C point,

2. assign the points that i influences as F points,

3. increment by 1 the weights of the points influencing these new F points,

4. decrease by 1 the weights of points that depends on i.

This first pass guarantees that each F point has at least one connection to a C point in the graph of S. It tends sometimes to produce too many F points. A second pass (see [7]) could be added in which some F points are made into C points to enforce the first criterion and to minimize C–C connections.

There are some others ways to generate the F and C points. For instance, we can only flag a percentage of the points that i influences as F points choosing the points with maximal connections (that is large values of $m_{i,j}$ or $\tilde{z}_{i,j}$). Some numerical experiments with other algorithms are reported in [5].

3.4 The interpolation algorithm

The standard multigrid algorithm uses bilinear interpolation, see for instance [4]. However, it is well known that this is not satisfactory for general PDE problems. The AMG algorithm uses instead an interpolation based on the equations of the linear system. It is obtained by writing the equations for $Ae = 0$ and by doing some approximations using averages to define the values of points which are not C points. Note that to use this interpolation we need that each F point must have at least one C point as a neighbor in the graph of A.

Another possibility is to use the approximate inverse again. Let C_i be the set of coarse nodes in S_i. For an F point i, the interpolation weights $w_{i,j}$ are defined as

$$w_{i,j} = \frac{n_{i,j}}{\sum_{l \in C_i} n_{i,l}} \quad j \in C_i,$$

where $N = M$ or $\tilde{Z} + \tilde{Z}^T - Q$. The rationale behind this choice is that the points which are more important for interpolation are the ones with the strongest connections. These choices will be denoted respectively as 'im' and 'iz'.

3.5 Other possibilities

Variations of the previous algorithms include using a parameter τ which varies with the grid level or using only the q largest elements on each column of Z. Another possibility is to use the regular AINV on the finest level and to truncate on the coarsest levels. This can sometimes save some floating point operations and still give the same number of iterations as the full algorithm.

4 Numerical experiments

In this section we first show some numerical results on problems arising from discretization of partial differential equations. We chose second order diffusion equation problems of different types: Poisson equation, an anisotropic problem, a discontinuous problem and a problem with rapidly varying diffusion coefficient.

Then, we will give results for problems arising from the Harwell–Boeing collection.

4.1 The Poisson equation

We solve the Poisson equation with Dirichlet boundary conditions in the unit square using finite differences with m discretization points in each direction (excluding boundaries) with a natural (left to right and bottom to top) ordering. This gives a matrix of order $n = m^2$. The right hand side is a random vector (the same in all experiments of a given dimension) and the starting vector is $x^0 = 0$. The iterations are stopped when the 2–norm of the residual is less than 10^{-10} the 2–norm of the initial residual. We will use a maximum of 7 levels which is enough for the problem sizes we considered. We use the direct solver when the problem size is less than 10. This may not be the optimal choice.

We first consider the two–grid algorithm since, obviously a multilevel method cannot do better. We give the number of iterations and the condition number of $\tilde{M}_0 A$ as given by the eigenvalues of the tridiagonal matrix of the PCG coefficients although this cannot be very reliable when the number of iterations is small. All the numerical results in this paper were obtained using Matlab 5 R11 on a Sony PCG-X9 personal computer with a 500 Mhz Intel Pentium III.

Table 1: PCG for Poisson equation, $\tau = 0.06$, two–grid, `AINV smoothing`, ('m', 'st', 'im', $\gamma = 1$)

	$\nu = 1$	$\nu = 2$	$\nu = 5$
$m = 10$	11 1.61	8 1.19	5 1.05
$m = 20$	13 1.76	9 1.29	7 1.19
$m = 30$	13 1.80	10 1.38	8 1.27
$m = 40$	13 1.78	10 1.46	9 1.34
$m = 50$	13 1.81	11 1.48	10 1.39
$m = 60$	14 1.80	11 1.51	10 1.43

Table 2: PCG for Poisson equation, $\tau = 0.06$, two–grid, `AINV smoothing`, ('m', 'st', 'im', $\gamma = 2$)

	$\nu = 1$	$\nu = 2$	$\nu = 5$
$m = 10$	7 1.17	5 1.03	3 1.002
$m = 20$	9 1.23	6 1.06	5 1.03
$m = 30$	9 1.25	7 1.09	5 1.05
$m = 40$	9 1.24	7 1.11	6 1.07
$m = 50$	9 1.23	8 1.13	7 1.04
$m = 60$	9 1.23	8 1.14	7 1.09

From Table 1, we see that this V–cycle two–grid method is scalable as there is only a very slight increase in the condition number. Moreover the numbers of iterations are much better than when using AINV as a preconditioner.

Table 2 gives the results using a W–cycle. From these results we see that using a W–cycle we have a constant number of iterations since the condition number is (almost) independent of the dimension of the problem. The number of iterations can be improved by using two different thresholds (τ_1, τ_2). This is shown in Tables 3 and 4.

In Tables 5 and 6 we repeat these experiments for ('z', 'st', 'iz'). The results from these tables show that the results for 'z' are as good as when using the approximate inverse in 'm'. Moreover, they are scalable for $\gamma = 1$ (for the number of iterations) and $\gamma = 2$. We note that the grids generated by 'm' and 'z' are not the same. Tables 7 and 8 show the results using two thresholds.

Table 3: PCG for Poisson equation, $\tau = (0.06, 0.02)$, two–grid, `AINV smoothing`, ('m', 'st', 'im', $\gamma = 1$)

	$\nu = 1$	$\nu = 2$	$\nu = 5$
$m = 10$	7 1.18	5 1.16	3 1.07
$m = 20$	9 1.25	7 1.16	5 1.07
$m = 30$	10 1.33	8 1.25	6 1.16
$m = 40$	10 1.39	9 1.31	7 1.22
$m = 50$	11 1.44	9 1.36	8 1.26
$m = 60$	11 1.48	9 1.40	8 1.30

Table 4: PCG for Poisson equation, $\tau = (0.06, 0.02)$, two–grid, `AINV smoothing`, ('m', 'st', 'im', $\gamma = 2$)

	$\nu = 1$	$\nu = 2$	$\nu = 5$
$m = 10$	5 1.02	3 1.001	2 1.
$m = 20$	6 1.04	4 1.02	3 1.004
$m = 30$	6 1.07	5 1.04	4 1.02
$m = 40$	6 1.09	6 1.06	4 1.03
$m = 50$	7 1.10	6 1.07	5 1.05
$m = 60$	7 1.11	6 1.09	6 1.06

Table 5: PCG for Poisson equation, $\tau = 0.06$, two–grid, `AINV smoothing`, ('z', 'st', 'iz', $\gamma = 1$)

	$\nu = 1$	$\nu = 2$	$\nu = 5$
$m = 10$	11 1.72	8 1.22	5 1.02
$m = 20$	14 1.96	9 1.32	6 1.04
$m = 30$	14 2.01	10 1.34	6 1.04
$m = 40$	15 2.04	10 1.35	6 1.04
$m = 50$	15 2.05	10 1.35	6 1.04
$m = 60$	15 2.05	10 1.36	6 1.04

Table 6: PCG for Poisson equation, $\tau = 0.06$, two–grid, `AINV smoothing`, ('z', 'st', 'iz', $\gamma = 2$)

	$\nu = 1$	$\nu = 2$	$\nu = 5$
$m = 10$	8 1.21	5 1.03	3 1.0004
$m = 20$	9 1.31	6 1.06	3 1.001
$m = 30$	10 1.34	6 1.07	4 1.001
$m = 40$	10 1.35	7 1.07	4 1.005
$m = 50$	10 1.35	7 1.07	4 1.006
$m = 60$	10 1.36	7 1.07	4 1.007

Table 7: PCG for Poisson equation, $\tau = (0.06, 0.02)$, two–grid, `AINV smoothing`, ('z', 'st', 'iz', $\gamma = 1$)

	$\nu = 1$	$\nu = 2$	$\nu = 5$
$m = 10$	6 1.06	4 1.01	3 1.001
$m = 20$	7 1.12	5 1.03	4 1.01
$m = 30$	8 1.14	6 1.03	4 1.02
$m = 40$	8 1.14	6 1.03	5 1.02
$m = 50$	8 1.14	6 1.03	5 1.02
$m = 60$	8 1.14	6 1.03	5 1.02

Table 8: PCG for Poisson equation, $\tau = (0.06, 0.02)$, two–grid, `AINV smoothing`, ('z', 'st', 'iz', $\gamma = 2$)

	$\nu = 1$	$\nu = 2$	$\nu = 5$
$m = 10$	4 1.003	3 1.0002	2 1
$m = 20$	5 1.01	3 1.001	2 1.0002
$m = 30$	5 1.01	3 1.001	3 1.0003
$m = 40$	5 1.01	3 1.001	3 1.0004
$m = 50$	5 1.02	4 1.001	3 1.0004
$m = 60$	5 1.02	4 1.001	3 1.0005

Table 9: PCG for Poisson equation, $\tau = 0.06$, MLAINV, `AINV smoothing`, ('z', 'st', 'iz', $\gamma = 1$)

m	$\nu = 1$	$\nu = 2$	$\nu = 5$
10	12 op=188340, /n=1883 100-50-19-10 str=2091, /n=20.9 $\kappa = 1.72$	8 op=207426, /n=2074 $\kappa = 1.22$	5 op=292389, /n=2928 $\kappa = 1.02$
20	14 op=987293, /n=2468 400-200-74-36-17 str=9433, /n=23.6 $\kappa = 1.96$	10 op=1161745, /n=2904 $\kappa = 1.35$	6 op=1575981, /n=3940 $\kappa = 1.11$
30	15 op=2448357, /n=2720 900-450-154-69-31-16 str=21701, /n=24.1 $\kappa = 2.02$	10 op=2709534, /n=3011 $\kappa = 1.48$	7 op=4216269, /n=4685 $\kappa = 1.19$
40	15 op=4513421, /n=2821 1600-800-267-130-61-29-13 str=39949, /n=24.9 $\kappa = 2.05$	11 op=5464389, /n=3415 $\kappa = 1.57$	8 op=8785665, /n=5491 $\kappa = 1.25$
50	16 op=7700375, /n=3080 2500-1250-431-208-97-50-21 str=64429, /n=25.7 $\kappa = 2.12$	11 op=8776877, /n=3518 $\kappa = 1.63$	8 op=14118983, /n=5647 $\kappa = 1.29$
60	16 op=11078581, /n=3078 3600-1800-607-283-136-65-27 str=92655, /n=25.7 $\kappa = 2.18$	12 op=13684058, /n=3801 $\kappa = 1.69$	8 op=20310541, /n=5642 $\kappa = 1.32$

Since for this problem the results are as good using 'z' instead of 'm' and moreover, using M is not really practical with AINV, from now on we will only give multilevel results using 'z'.

Let us now consider the multilevel algorithms for ('z', 'st', 'iz'). In addition to the number of iterations, we give the number of nodes on each grid as well as the total storage for the preconditioner under 'str'. The number of operations is also given (under 'op') and the number of operations divided by the problem size. This is used to assess the real scalability of the algorithm.

We remark that there is almost no degradation by using more than two grids. Moreover, it does not pay to use $\nu > 1$. Even though the number of iterations is larger, the V–cycle is much cheaper than the W–cycle at least for these problem dimensions. Since the number of iterations is a little bit less scalable for the V–cycle than for the W–cycle it might be that in the long run, for very large dimensions, it is better to use $\gamma = 2$. Moreover, it can also be interesting to reduce the number of iterations by using $\gamma = 2$ or by using more smoothing

Table 10: PCG for Poisson equation, $\tau = 0.06$, MLAINV, `AINV smoothing`, ('z', 'st', 'iz', $\gamma = 2$)

m	$\nu = 1$	$\nu = 2$	$\nu = 5$
10	8 op=407073, /n=4070 100-50-19-10 str=2091, /n=20.9 $\kappa = 1.21$	5 op=446205, /n=4462 $\kappa = 1.03$	3 op=647685, /n=6477 $\kappa = 1.0004$
20	9 op=2633333, /n=6583 400-200-74-36-17 str=9433, /n=23.6 $\kappa = 1.31$	6 op=3044553, /n=7611 $\kappa = 1.06$	4 op=4751233, /n=11878 $\kappa = 1.002$
30	10 op=7784329, /n=8649 900-450-154-69-31-16 str=21701, /n=24.1 $\kappa = 1.34$	7 op=9400725, /n=10445 $\kappa = 1.07$	4 op=12888993, /n=14321 $\kappa = 1.003$
40	10 op=17718713, /n=11074 1600-800-267-130-61-29-13 str=39949, /n=24.9 $\kappa = 1.35$	7 op=21489005, /n=13431 $\kappa = 1.07$	4 op=29564513, /n=18478 $\kappa = 1.006$
50	10 op=30952313, /n=12381 2500-1250-431-208-97-50-21 str=64429, /n=25.7 $\kappa = 1.35$	7 op=37482165, /n=14993 $\kappa = 1.07$	4 op=51503793, /n=20602 $\kappa = 1.006$
60	10 op=44002793, /n=12223 3600-1800-607-283-136-65-27 str=92655, /n=25.7 $\kappa = 1.36$	7 op=53260013, /n=14794 $\kappa = 1.07$	4 op=73154993, /n=20321 $\kappa = 1.007$

on a parallel computer to avoid some scalar products that are involved for each PCG iteration. Asymptotically both the number of operations and the storage for the preconditioner are scalable.

We also note that we could obtain better results if the matrix A is symmetrically scaled before using MLAINV. Finally, $\tau = 0.06$ is not the optimal threshold parameter regarding the number of operations. Slightly better results are obtained using $\tau = 0.04$ although the storage is larger.

4.2 An anisotropic problem

We would like to solve an anisotropic diffusion problem with constant coefficients. The coefficient is 1 in the x–direction and 100 in the y–direction. This is a though problem for approximate inverses since the decrease in the elements of the inverse is very slow in one direction. In fact, the fill–in for the approximate factors is very sensitive to the threshold parameter.

We symmetrically scale the matrix A to have a unit diagonal (obtaining A_d). However, for this problem, the condition number of the scaled matrix is the same as for the original one. The condition number of A_d for $m = 50$ is almost 1050. The scaling is necessary to have a threshold parameter which is not varying too much with the problem. We use the same right hand side, starting vector and stopping criteria as for the Poisson equation. Table 11 gives the results for the MLAINV preconditioner. We remark that for this problem we have to keep a lot of fill. For the same value of τ the fill is much larger than for the Poisson problem. This is because the elements of the inverse decrease very slowly in one direction. The unknowns are very strongly coupled in that direction so we have almost a one dimensional problem. This gives coarse grids with very few nodes since there are many nodes which are strongly coupled together.

For the anisotropic problem the results in Table 11 are not fully scalable, even with the W–cycle although the increase in the number of iterations is quite small. We note that if we use $\nu = 2$ with the W–cycle the results are almost scalable even if it is more costly for small problems. These results can be improved by using a couple of threshold parameters. Table 12 gives the results with $\tau = (1, 0.01)$. This leads to coarse grids with more nodes and approximate inverses with more fill–in. We see from the results that the number of iterations is almost constant. However, the storage and the number of operations grow with the problem dimension. Moreover, the number of operations is much larger than when using $\tau = 0.1$.

4.3 A discontinuous problem

Here we are concerned with an isotropic diffusion problem with constant but discontinuous coefficients. The diffusion coefficient is 1 except in the strip $[0, 1] \times [1/4, 3/4]$ where its value is 100. In the discretization we were not really cautious about the average of the coefficients; we just took their pointwise values. Therefore, we will have sometimes to use problem dimensions different from

Table 11: PCG for the anisotropic problem, $\tau = 0.1$, MLAINV, `AINV smoothing`, ('z', 'st', 'iz')

m	$\gamma = 1$	$\gamma = 2$
10	6 op=75272, /n=752 100-15 str=1400, /n=14 $\kappa = 1.05$	4 op=97633, /n=976 $\kappa = 1.002$
20	8 op=481448, /n=1204 400-20 str=6543, /n=16.4 $\kappa = 1.19$	5 op=592397, /n=1481 $\kappa = 1.03$
30	10 op=1582331, /n=1758 900-60-11 str=17618, /n=19.5 $\kappa = 1.38$	6 op=1949849, /n=2166 $\kappa = 1.08$
40	11 op=3365229, /n=2103 1600-120-14 str=34749, /n=21.8 $\kappa = 1.50$	7 op=4415917, /n=2759 $\kappa = 1.12$
50	13 op=6393915, /n=2558 2500-175-26-13 str=56497, /n=22.6 $\kappa = 1.81$	8 op=8188289, /n=3275 $\kappa = 1.25$
60	14 op=10046068, /n=2791 3600-210-32-16 str=82475, /n=22.9 $\kappa = 2.06$	9 op=13272893, /n=3687 $\kappa = 1.35$

Table 12: PCG for the anisotropic problem, $\tau = (1, 0.01)$, MLAINV, `AINV smoothing`, ('z', 'st', 'iz')

m	$\gamma = 1$	$\gamma = 2$
10	4 op=90038, /n=900 100-50-24-12 str=2497, /n=24.9 $\kappa = 1.008$	3 op=235557, /n=2356 $\kappa = 1.0001$
20	5 op=630197, /n=1575 400-200-66-33-14 str=13850, /n=34.6 $\kappa = 1.03$	3 op=1452157, /n=3630 $\kappa = 1.0008$
30	6 op=2305820, /n=2562 900-450-159-79-38-17 str=42850, /n=47.6 $\kappa = 1.05$	4 op=6502553, /n=7225 $\kappa = 1.002$
40	6 op=5028431, /n=3143 1600-800-251-126-58-29-15 str=92607, /n=57.9 $\kappa = 1.07$	4 op=14665673, /n=9166 $\kappa = 1.002$
50	7 op=10738437, /n=4295 2500-1250-414-207-81-34-15 str=172056, /n=68.8 $\kappa = 1.09$	4 op=27527546, /n=11011 $\kappa = 1.003$
60	7 op=18011516, /n=5003 3600-1800-616-308-126-63-31 str=288139, /n=80 $\kappa = 1.11$	4 op=49557238, /n=13766 $\kappa = 1.003$

Table 13: PCG for the discontinuous problem, $\tau = 0.06$, MLAINV, `AINV smoothing`, ('z', 'st', 'iz')

m	$\gamma = 1$	$\gamma = 2$
10	7 op=111261, /n=1126 100-15 str=1882, /n=18.8 $\kappa = 1.18$	5 op=154989, /n=1550 $\kappa = 1.03$
20	11 op=824933, /n=2062 400-44-13 str=8782, /n=21.9 $\kappa = 3.52$	8 op=1244657, /n=3111 $\kappa = 2.04$
30	12 op=2137984, /n=2375 900-100-28-13 str=21099, /n=23.4 $\kappa = 3.9$	9 op=3505293, /n=3895 $\kappa = 2.19$
39	14 op=4368457, /n=2872 1521-166-47-21-10 str=37356, /n=24.5 $\kappa = 4.99$	10 op=7164021, /n=4710 $\kappa = 2.68$
50	16 op=8321810, /n=3329 2500-271-51-22 str=62850, /n=25.1 $\kappa = 6.5$	11 op=12523013, /n=5009 $\kappa = 3.32$
59	17 op=12617568, /n=3625 3481-387-69-21-16 str=89907, /n=25.8 $\kappa = 7.9$	12 op=20087094, /n=5770 $\kappa = 3.8$

those of the previous problems in order not to have points where the coefficients are evaluated on the discontinuities.

We symmetrically scale the matrix A to have a unit diagonal and we use the same right hand side, starting vector and stopping criteria as for the Poisson equation. The scaled matrix is an M–matrix but it is not diagonally dominant.

For the discontinuous problem, the results in Table 13 are not scalable for the V–cycle. The increase for the number of iterations is really small for the W–cycle. However, the number of iterations can be reduced if we use coarse grids with more points with $\tau = (0.2, 0.06)$ as in Table 14. It can be further reduce if we smooth a little more using $\tau = (0.2, 0.01)$ at the expense of a larger storage.

Table 14: PCG for the discontinuous problem, $\tau = (0.2, 0.06)$, MLAINV, `AINV smoothing`, ('z', 'st', 'iz')

m	$\gamma = 1$	$\gamma = 2$
10	6 op=127051, /n=1270 100-40-15 str=2552, /n=25.5 $\kappa = 1.15$	4 op=218273, /n=2183 $\kappa = 1.01$
20	10 op=988121, /n=2470 400-147-53-23-10 str=11898, /n=29.7 $\kappa = 2.72$	6 op=1889889, /n=4725 $\kappa = 1.59$
30	10 op=2410906, /n=2679 900-304-117-53-24-12 str=28896, /n=32.2 $\kappa = 2.45$	6 op=6298817, /n=5887 $\kappa = 1.22$
39	11 op=4742431, /n=3118 1521-511-214-102-43-17 str=52274, /n=34.4 $\kappa = 2.9$	6 op=10171765, /n=6687 $\kappa = 1.31$
50	13 op=9484653, /n=3794 2500-837-356-174-67-31-17 str=89472, /n=35.8 $\kappa = 4.01$	7 op=21897205, /n=8759 $\kappa = 1.57$
59	13 op=13619971, /n=3913 3481-1163-510-248-88-40-17 str=128477, /n=36.9 $\kappa = 4.31$	8 op=36180434, /n=10390 $\kappa = 1.63$

Table 15: PCG for the rapidly varying coefficient problem, $\tau = 0.06$, MLAINV, `AINV smoothing`, ('z', 'st', 'iz')

m	$\gamma = 1$	$\gamma = 2$
10	7 op=104509, /n=1045 100-13 str=1737, /n=17.4 $\kappa = 1.21$	4 op=120433, /n=1204 $\kappa = 1.03$
20	9 op=655513, /n=1649 400-46-15 str=8464, /n=21.2 $\kappa = 1.6$	6 op=929041, /n=2322 $\kappa = 1.16$
30	11 op=1928857, /n=2143 900-103-29-13 str=20670, /n=22.9 $\kappa = 1.8$	7 op=2764629, /n=3072 $\kappa = 1.23$
40	12 op=3849758, /n=2406 1600-180-37-15 str=38032, /n=23.8 $\kappa = 2.00$	8 op=5709825, /n=3568 $\kappa = 1.29$
50	13 op=6677681, /n=2671 2500-279-52-23-10 str=61230, /n=24.5 $\kappa = 2.15$	8 op=9535625, /n=3813 $\kappa = 1.34$
60	14 op=10475023, /n=2910 3600-402-66-28-12 str=89631, /n=24.9 $\kappa = 2.31$	9 op=15479973, /n=4300 $\kappa = 1.39$

4.4 A problem with rapidly varying coefficients

This problem is isotropic. The diffusion coefficient is $1 + 1000|x - y|$. We symmetrically scale the matrix A to have a unit diagonal and we use the same right hand side, starting vector and stopping criteria as for the Poisson equation. The condition number of A_d for $m = 50$ is almost 614.

For this problem, as we can see in Table 15, the number of iterations is almost constant with $\gamma = 2$, the number of operations as well as the condition number being slightly increasing. Table 16 gives the results with $\tau = (1, 0.01)$. Then, the number of iterations is constant but the number of operations is increasing more than linearly. The same is true for the storage.

Table 16: PCG for the rapidly varying coefficient problem, $\tau = (1, 0.01)$, MLAINV, `AINV smoothing`, ('z', 'st', 'iz')

m	$\gamma = 1$	$\gamma = 2$
10	4 op=175818, /n=1758 100-50-24-14 str=4726, /n=47.2 $\kappa = 1.008$	3 op=403493, /n=4035 $\kappa = 1.0001$
20	5 op=1631519, /n=4079 400-200-99-53-26-12 str=35029, /n=87 $\kappa = 1.05$	3 op=3848317, /n=9621 $\kappa = 1.002$
30	6 op=5849563, /n=6499 900-450-215-111-54-23-10 str=106604, /n=118 $\kappa = 1.11$	4 op=14671353, /n=16302 $\kappa = 1.009$
40	7 op=14237629, /n=8898 1600-800-385-199-104-45-22 str=226485, /n=141 $\kappa = 1.18$	4 op=31440793, /n=19650 $\kappa = 1.02$
50	8 op=27883151, /n=11153 2500-1250-605-314-157-72-37 str=393652, /n=157 $\kappa = 1.30$	5 op=65753549, /n=26301 $\kappa = 1.03$
60	8 op=43150516, /n=11986 3600-1800-875-454-233-108-53 str=608523, /n=169 $\kappa = 1.47$	5 op=102536912, /n=28482 $\kappa = 1.05$

Table 17: PCG for the random Laplacian problem, $\tau = 0.06$, MLAINV, `AINV smoothing`, ('z', 'st', 'iz')

m	$\gamma = 1$	$\gamma = 2$
10	9 op=146023, /n=1460 100-50-24-12 str=2134, /n=21.3 $\kappa = 1.40$	6 op=335049, /n=3350 $\kappa = 1.09$
20	10 op=731590, /n=1829 400-200-91-46-23-12 str=9489, /n=23.7 $\kappa = 1.43$	7 op=2729133, /n=6822 $\kappa = 1.10$
30	11 op=1850653, /n=2056 900-450-204-96-49-25-13 str=21933, /n=24.4 $\kappa = 1.57$	7 op=7510421, /n=8345 $\kappa = 1.15$
40	12 op=3624403, /n=2265 1600-800-381-176-90-40-20 str=39655, /n=24.8 $\kappa = 1.87$	8 op=15291297, /n=9557 $\kappa = 1.27$
50	12 op=5724579, /n=2289 2500-1250-595-288-129-65-36 str=62822, /n=25.1 $\kappa = 1.78$	8 op=24772013, /n=9909 $\kappa = 1.24$

4.5 A random Laplacian

To show that MLAINV is not only working for M–matrices we consider matrices arising from the Poisson equation as in the first example but with the signs of the non zero non diagonal coefficients chosen at random. This implies that the matrix is an H–matrix. The results in Table 17 show that the method is working nicely even with non M–matrices.

4.6 Other problems

All the preceding examples but the last one arise from two dimensional diffusion equations discretized with a finite differences five point scheme. We would like to see how these methods behave on examples coming from other areas of scientific computing. We chose some symmetric matrices from the Harwell–Boeing collection or from the Boeing collection stored in the Tim Davis' collection (`http://www.cise.ufl.edu`). We note that there are not many symmetric matrices in these collections and their orders are small. We had to normalize these matrices to be able to use the same values of τ as before. Of course, since the orders of the matrices are given we cannot check if there is a dependence of

the number of iterations on the size of the problem. Moreover, some of the problems are quite small and it can be that AINV is faster than its multilevel counterpart. A solution with a direct method is also much faster. We use the following examples:

1. *1138-bus.* An admittance matrix of order 1138 with 4054 non–zeros. It is normalized. This matrix is almost singular. To obtain a meaningful problem we add 0.01 to the diagonal elements of the normalized matrix.

2. *bcsstk01.* A stiffness matrix of order 48 with 400 non-zeros. It is normalized. The condition number of the normalized matrix is 1361. This matrix is not diagonally dominant, nor an M–matrix, but nevertheless positive definite.

3. *gr3030.* A matrix arising from a nine point approximation to the Laplacian on the unit square with a 30×30 mesh. It has order 900 and 7744 non–zeros. The condition number of A_d is 194.

4. *bcsstk34.* A stiffness matrix of order 588 with 21418 non-zeros. This matrix was normalized.

4.6.1 Results on 1138-bus modified

We use $\tau = 0.06$. The number of nodes on the grids are $1138-221-88-39-15$ with a storage of $19597, /n = 17.2$. The number of iterations with $\gamma = 1$ is 15 and the number of operations is $2402769, /n = 2111$. The condition number is 3.10. With $\gamma = 2$ we obtain 10 iterations and the number of operations is $4873665, /n = 4283$, the condition number being 1.85.

4.6.2 Results on bcsstk01

We use $\tau = 0.2$. The number of nodes on the grids are $48 - 10$ with a storage of $867, /n = 18.1$. The number of iterations with $\gamma = 1$ is 14 and the number of operations is $92294, /n = 1922$. With $\gamma = 2$ we obtain 10 iterations and the number of operations is $120709, /n = 2514$.

4.6.3 Results on gr3030

We use $\tau = 0.06$. The number of nodes on the grids are $900 - 117 - 33 - 14$ with a storage of $24975, /n = 27.7$. The number of iterations with $\gamma = 1$ is 9 and the number of operations is $1916455, /n = 2129$. The condition number is 1.39. With $\gamma = 2$ we obtain 6 iterations and the number of operations is $2917241, /n = 3241$, the condition number being 1.07.

4.6.4 Results on bcsstk34

We use $\tau = 0.2$. The number of nodes on the grids are $588-295-150-74-32-14$ with a storage of $55814, /n = 94.9$. The number of iterations with $\gamma = 1$ is 6

and the number of operations is $2662494, /n = 4528$. The condition number is 1.06. With $\gamma = 2$ we obtain 4 iterations and the number of operations is $8696762, /n = 14790$, the condition number being 1.003.

5 Conclusion

We have seen that the approximate inverse AINV can be extended to a multilevel preconditioner that gives good results when used with PCG for solving sparse symmetric positive definite linear systems. For some problems, the number of iterations is constant, for some more difficult ones it is only very slightly increasing.

We have explored the possibility to define the influences between nodes by using the approximate inverse. It tends to give coarse grids with a smaller number of nodes than other approaches, see [5]. This leads to cheaper solutions when it works. However, by using a couple of threshold parameters we were always able to obtain a good smoothing and coarse grids which are not too coarse.

It remains to be seen what is the performance of the method on parallel computers for very large problem dimensions. Parallelism can be introduced by using a preordering of the unknowns with domain decomposition–like principles. This will be presented in a forthcoming paper. Numerical comparisons with other multilevel methods are given in [5].

Acknowledgments

The author thanks Michele Benzi for interesting conversations about using AINV and the referees for helpful comments.

References

[1] M. Benzi, C.D. Meyer and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., vol 17, (1996), pp 1135-1149.

[2] M. Benzi, J. Cullum and M. Tuma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM J. Sci. Comput., vol 22, (2000), pp 1318-1332.

[3] V.E. Henson, *An algebraic multigrid tutorial*, MGNET, http://www.mgnet.org, (1999).

[4] G. Meurant, *Computer solution of large linear systems*, North–Holland, (1999).

[5] G. Meurant, *Numerical experiments with algebraic multilevel preconditioners*, accepted in ETNA.

[6] J.W. Ruge and K. Stuben, *Algebraic multigrid*, in Multigrid methods, S.F. Mc Cormick ed., SIAM, (1987), pp 73-130.

[7] C. Wagner, *Introduction to algebraic multigrid*, Course notes version 1.1, University of Heidelberg, (1998).