

**PRECONDITIONERS FOR SOIL-STRUCTURE  
INTERACTION PROBLEMS WITH SIGNIFICANT  
MATERIAL STIFFNESS CONTRAST**

**KRISHNA BAHADUR CHAUDHARY**

**NATIONAL UNIVERSITY OF SINGAPORE  
2010**

**PRECONDITIONERS FOR SOIL-STRUCTURE  
INTERACTION PROBLEMS WITH SIGNIFICANT  
MATERIAL STIFFNESS CONTRAST**

**KRISHNA BAHADUR CHAUDHARY**

*(B.Eng., TU, Nepal)*  
*(M.Eng., AIT, Thailand)*

**A THESIS SUBMITTED  
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY  
DEPARTMENT OF CIVIL ENGINEERING**

**NATIONAL UNIVERSITY OF SINGAPORE  
2010**

# ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to my supervisor, Prof. Phoon Kok Kwang, for his persistent guidance, inspiration, and encouragement to conduct this research. I am also highly indebted to my co-supervisor, Prof. Toh Kim Chuan (Department of Mathematics, NUS), for his critical evaluation and suggestions on mathematics needed for this research work. Without their help and support, the accomplishment of this thesis would be impossible. I am equally grateful to the members of my thesis committee Prof. Tan Thiam Soon and A/Prof Tan Siew Ann for their valuable advice on my research work.

I would also like to thank National University of Singapore (NUS) for providing the 'Research Scholarship'. Without this support, I could not imagine myself to be here.

My sincere thanks go to Dr. Chen Xi and Dr. Cheng Yonggang for their great help, useful materials, and encouragement on various struggling situations during my research work. I would also like to thank Dr. Hong Sze Han (GeoSoft Pte Ltd) for his guidance on the use of GeoFEA software for my thesis. I greatly appreciate our geotechnical lab officers for their readiness to provide any technical support. My vote of thanks also goes to all the members of the geotechnical research group, whose company made my stay at NUS lively and joyful.

Finally, I should not forget to thank my parents, wife, and our entire family in Nepal for their endless love, caring, and understanding, which gave me a peace of mind to fully concentrate on my work.

*(blank)*

# TABLE OF CONTENTS

---

ACKNOWLEDGEMENTS	i
TABLE OF CONTENTS	iii
SUMMARY	vii
LIST OF TABLES	xi
LIST OF FIGURES	xiii
LIST OF SYMBOLS	xix
DEDICATION	xxix
INTRODUCTION	1
1.1. Introduction	1
1.2. Iterative solvers and the role of preconditioning in geotechnical problems	5
1.3. Scope and objective of the study	10
1.4. Computer hardware and software	14
1.5. Thesis outline	16
LITERATURE REVIEW	19
2.1. Iterative solution methods	20
2.2. Preconditioning strategies	22
2.2.1. Diagonal preconditioners	25
2.2.2. SSOR preconditioner	28
2.2.3. Incomplete factorization preconditioners	30
2.2.4. Block preconditioners	32
2.2.5. Others	41
2.3. Sparse storage of the matrix	42
2.4. Convergence criteria	43
2.4.1. Numerical experiment	47
2.5. Conclusions	49
PERFORMANCE OF ILU0 VERSUS MSSOR FOR BIOT'S CONSOLIDATION EQUATIONS	59
3.1. Introduction	59
3.2. Numerical Results	60

3.2.1.	Effect of nodal ordering	62
3.2.2.	Problems with ILU factorization and their stabilization	65
3.2.3.	MSSOR versus ILU0 and GJ preconditioners	68
3.2.4.	Performance of preconditioners on a pile group problem	69
3.3.	Conclusions	70
<b>BLOCK DIAGONAL PRECONDITIONERS FOR DRAINED ANALYSIS</b>		<b>89</b>
4.1.	Introduction	89
4.2.	Soil-structure interaction problem and preconditioning	91
4.2.1.	Block diagonal Preconditioner	94
4.2.2.	Inexact block diagonal preconditioners	97
4.3.	Numerical results	100
4.3.1.	Piled-raft foundation	101
4.3.2.	Tunneling	111
4.4.	Conclusion	113
<b>BLOCK DIAGONAL PRECONDITIONERS FOR BIOT'S CONSOLIDATION EQUATIONS</b>		<b>139</b>
5.1.	Introduction	139
5.2.	Biot's consolidation equations and block diagonal preconditioning	141
5.3.	Numerical experiments	149
5.3.1.	Piled-raft foundation	150
5.3.2.	Tunneling	164
5.4.	Conclusion	167
<b>APPLICATIONS ON CASE HISTORIES</b>		<b>187</b>
6.1.	Introduction	187
6.2.	GeoFEA implementation details	188
6.2.1.	Tutorial manual	191
6.3.	Applications on case histories	192
6.3.1.	Case study 1 – Piled-raft foundation in Germany	193
6.3.2.	Case study 2 – Tunneling in Singapore	200
6.4.	Conclusions	203
<b>CONCLUSIONS AND RECOMMENDATIONS</b>		<b>225</b>
7.1.	Summary and conclusions	225
7.2.	Limitations and Recommendations	228
<b>REFERENCES</b>		<b>231</b>
<b>APPENDIX A. LINEAR ALGEBRA</b>		<b>245</b>

APPENDIX B. BIOT'S CONSOLIDATION EQUATIONS	259
APPENDIX C. ALGORITHMS	267
APPENDIX D. 1D FINITE ELEMENT DISCRETIZATION OF OEDOMETER SETUP	273
APPENDIX E. SOURCE CODES IN FORTRAN 90	275
APPENDIX F. USER DEFINED SOLVER IN GeoFEA	349

*(blank)*

# SUMMARY

---

Three-dimensional finite element analysis of geotechnical problems usually involves a significant large number of variables (or unknowns) and non-uniformity of the materials. Recent advances on solution methods of linear systems show that Krylov subspace iterative methods in conjunction with appropriate preconditioning are potentially more effective than direct solution methods for large-scale systems. A preconditioner is the key for the success of iterative methods. For this reason, a number of publications have recently been devoted to propose effective preconditioners for the solution of large, often ill-conditioned coupled consolidation problems. Some of them may require a number of user-defined parameters, which may limit their practical use. Also, much of the work has been devoted on the ill-conditioning due to small time steps in the consolidation analysis. Little attention has been paid on the ill-conditioning due to significant contrasts in material properties such as stiffness and permeability. This significant difference in material properties may deteriorate the performance (Chen *et al.*, 2007) of so called cheap and effective preconditioners such as generalized Jacobi (GJ) (Phoon *et al.*, 2002) and modified symmetric successive over-relaxation (MSSOR) preconditioner (Chen *et al.*, 2006). Similar degradation in performance was also observed for the standard Jacobi (Lee *et al.*, 2002) and symmetric successive over-

relaxation (SSOR) preconditioners (Mroueh and Shahrour, 1999) for the analysis of drained boundary value problems. On the other hand, pragmatic geotechnical problems often involve materials with highly varied material zones, such as in soil-structure interaction problems. Hence, the prime objective of the thesis was to propose a preconditioner that mitigates these adverse effects and yet remain practical for use.

Firstly, the relative merits and demerits of MSSOR preconditioner was compared with ILU0 (incomplete LU factorization with zero fill-ins) for the Biot's coupled consolidation equations. This is because the ILU-type preconditioners have also frequently been used for Biot's problem (Gambolati *et al.*, 2001, 2002, 2003). The comparison revealed that the ILU0 is occasionally unstable, but may be preferred over MSSOR if its instability problem is resolved and RAM constraint is not an issue. On the other hand, MSSOR and GJ were robust in solving even a severe ill-conditioned system. Secondly, the ill-conditioning due to the presence of different material zones with large relative differences in material stiffnesses was addressed by proposing block diagonal preconditioners. The effect of only stiffness contrasts was considered first (in Chapter 4) and stiffness/permeability contrasts in the consolidation analysis was studied next (in Chapter 5). The inexpensive block diagonal preconditioners for practical use were investigated numerically using preconditioned conjugate gradient (PCG) solver and symmetric quasi-minimal residual (SQMR) solver. Significant benefits in terms of CPU time in comparison to existing preconditioners were demonstrated with the help of a number of soil-structure interaction problems. Finally, the general applicability of the proposed block diagonal

preconditioners for real-world problems was shown using two case history examples in Chapter 6.

**Keywords:** Three-dimensional finite element analysis, preconditioning, iterative solution, stiffness contrast, block diagonal preconditioner, PCG, SQMR

*(blank)*

# LIST OF TABLES

---

Table 3.1. Three-dimensional finite element meshes. ....	84
Table 3.2. Effect of ordering on ILU0 preconditioned SQMR.....	85
Table 3.3. Effect of ordering on MSSOR ( $\omega = 1$ , $\alpha = -4$ ) preconditioned SQMR. ....	86
Table 3.4. Effect of ordering on GJ ( $\alpha = -4$ ) preconditioned SQMR. ....	86
Table 3.5. Statistics that can be used to evaluate an incomplete factorization. ....	87
Table 3.6. ILU statistics and possible reasons of failure for soil 1 (soft clay).....	87
Table 3.7. ILU statistics and possible reasons of failure for soil 2 (sand).....	88
Table 3.8. ILU statistics and possible reasons of failure for soil 3 (layered soil). ....	88
Table 4.1. 25×25×35 mesh: Effect of different approximations of the block $P$ with diagonal approximation of block $G$ in the preconditioner (4.21) for a 9-piled raft problem.....	135
Table 4.2. 25×25×35 mesh: Performance of ILU factorization preconditioners on entire $K$ for a 9-piled raft problem (size of $K = 267,680 \times 267,680$ ).....	136
Table 4.3. Finite element details of piled-raft foundations.....	136
Table 4.4. Material properties of NATM tunnel.....	137
Table 4.5. Comparison of total CPU times for tunnel construction .....	137
Table 5.1. Material properties for piled-raft foundation.....	184
Table 5.2. 25×25×35 mesh: Effect of different approximations of block $P$ with diagonal approximation of blocks $G$ and $\tilde{S}$ in the preconditioner (5.28) for a 9-piled raft problem.....	185
Table 5.3. Problem statistics of the piled-raft foundations.....	186

---

Table 5.4. Comparison of total CPU times for tunnel construction. ....	186
Table 6.1. Properties of Frankfurt clay and piled-raft for FE analysis. ....	223
Table 6.2. Typical G4 soil parameters found in C704. ....	223
Table 6.3. Material properties of liner and grout elements. ....	223

# LIST OF FIGURES

---

Figure 2.1. Guideline for the selection of preconditioned iterative methods.....	51
Figure 2.2. Flow chart of applying sparse preconditioned iterative method in FE analysis (after Chen, 2005). ....	52
Figure 2.3. 8×8×8 mesh: A typical footing problem. ....	53
Figure 2.4. Behavior of various norms using GJ-SQMR for different material properties: (a) Conso1; (b) Conso2; (c) Conso3; (d) Conso4; (e) Conso5; and (f) Conso6. ....	54
Figure 3.1. Twenty-noded displacement finite element coupled with eight-noded fluid elements.....	72
Figure 3.2. 20×20×20 finite element mesh of a symmetric quadrant footing (after Chen, 2005): (a) homogeneous soils 1 and 2, (b) layered soil 3. ....	73
Figure 3.3. Sparsity pattern of A in: (a) Natural ordering, (b) Block ordering, and (d) Reverse Cuthill-McKee technique on natural ordering. ....	74
Figure 3.4. Typical relative residual norm of an unstable ILU0.....	75
Figure 3.5. Interpretation of ILU statistics (after Chow and Saad, 1997). ....	76
Figure 3.6. 20×20×20 mesh: Effect of threshold value on convergence of stabilized ILU0 for different soil profiles.....	77
Figure 3.7. Performance of ILU0 and GJ preconditioners with respect to MSSOR preconditioner for different soil conditions.....	78
Figure 3.8. Eigenvalue distribution of preconditioned matrices with different preconditioners: (a) stab-ILU0; (b) MSSOR; and (c) GJ.....	79
Figure 3.9. RAM usage by preconditioners with SQMR solver.....	80
Figure 3.10. 12×12×12 mesh: 3D FE discretization of a quadrant symmetric 9-pile group foundation in a homogeneous clay (soil 1) with a uniform load (after Chen <i>et al.</i> , 2007). ....	81

Figure 3.11. Performance of preconditioners for 9-pile group problem on homogeneous clay (soil 1). .....	82
Figure 3.12. Ground surface settlement for 9-pile group after loading at 1 <sup>st</sup> time step with pile stiffness of $E'_p = 30000$ MPa and soil stiffness of $E'_s = 1$ MPa. ....	83
Figure 4.1. One-dimensional FE discretization of oedometer test set up to illustrate the effect of different materials in the formulation of FE stiffness matrix. The dots and numbers besides them are finite element nodes and node numbers, $F$ is the applied load, $l$ is the element size, $E'_{ps}$ and $E'_s$ are the effective Young's moduli of porous stone and soil, respectively. ....	116
Figure 4.2. Three-dimensional FE discretization of a typical 9-piled raft foundation (quadrant symmetric): (a) a realistic problem discretized into $25 \times 25 \times 35$ mesh; (b) a model problem discretized into $7 \times 7 \times 7$ mesh to illustrate the spectral properties of the preconditioned system. ....	117
Figure 4.3. $7 \times 7 \times 7$ mesh: Condition number and iteration count of unpreconditioned and theoretical block diagonal preconditioned stiffness matrix $K$ for varying pile-soil stiffness ratios. ....	118
Figure 4.4. $7 \times 7 \times 7$ mesh: Eigenvalue distribution of theoretical exact block diagonal preconditioned system (4.14) at different pile-soil stiffness ratios: (a) $E'_p/E'_s = 1$ (fictitious pile); (b) $E'_p/E'_s = 1000$ ; and (c) $E'_p/E'_s = 41000$ . ....	119
Figure 4.5. $25 \times 25 \times 35$ mesh: Iteration count and total CPU time of inexact block diagonal preconditioner (4.21) for various inexact forms of block $P$ with diagonal approximation of soil block $G$ for a 9-piled raft. ....	120
Figure 4.6. $7 \times 7 \times 7$ mesh: Cumulative distribution of eigenvalues of the preconditioned system with inexact block diagonal preconditioner (4.21) for different inexact forms of block $P$ and diagonal of block $G$ at two different stiffness ratios. ....	121
Figure 4.7. $25 \times 25 \times 35$ mesh: Performance of different inexact forms of block $G$ with Cholesky factorization of block $P$ in the block diagonal preconditioner. ....	122
Figure 4.8. $7 \times 7 \times 7$ mesh: Cumulative distribution of eigenvalues of the preconditioned system for different inexact forms of block	

$G$ with Cholesky factorization of block $P$ in the block diagonal preconditioner. ....	123
Figure 4.9. 25×25×35 mesh: Comparison of SBD preconditioners with other preconditioners for a 9-piled raft in a range of $E'_p/E'_s$ . Preconditioners are: SJ = standard Jacobi; SSOR = symmetric successive over relaxation [Equation (4.27)]; ILU0 = Incomplete LU factorization preconditioner with zero fill-ins; SBD = simplified block diagonal preconditioners [Equations (4.25 and 4.26)]......	124
Figure 4.10. 25×25×35 mesh: Layout of piles in the piled-raft foundation (quadrant symmetric).....	125
Figure 4.11. 25×25×35 mesh: Effect of size of stiff block $P$ (e.g. due to variation in number of piles, raft thickness = 3 m, in the piled-raft problem) in the performance of preconditioners at different stiffness-ratios. ....	126
Figure 4.12. CPU time of SBD preconditioners for a range of stiff DOFs and soil-structure stiffness ratios: (a) SBD <sub>1</sub> versus SJ (b) SBD <sub>2</sub> versus SJ.....	127
Figure 4.13. CPU time of SBD preconditioners for a range of stiff DOFs and soil-structure stiffness ratios: (a) SBD <sub>1</sub> versus SSOR (b) SBD <sub>2</sub> versus SSOR. ....	128
Figure 4.14. CPU time of SBD preconditioners for a range of stiff DOFs and soil-structure stiffness ratios: (a) SBD <sub>1</sub> versus ILU0 (b) SBD <sub>2</sub> versus ILU0. ....	129
Figure 4.15. RAM consumed with different preconditioners for the same size (267,680 × 267,680) of the stiffness matrix $K$ . ....	130
Figure 4.16. Finite element mesh and step-by-step installation of liner in tunneling. ....	131
Figure 4.17. Comparison of iteration count and CPU time of preconditioners for the tunneling example. ....	132
Figure 4.18. 7×7×7 mesh: Sparsity pattern of 2×2 block structured $K$ . (a) Sequential nodal numbering of nodes in x-z plane according to Smith and Griffiths (1997; 2004); and (b) Automatic nodal numbering in GeoFEA. ....	133
Figure 4.19. Surface settlement profile after 40 steps of excavation.....	134
Figure 5.1. 7×7×7 mesh: Effect of varying pile-soil stiffness ratios on spectral condition number and iteration count of	

unpreconditioned and theoretical block diagonal preconditioned matrices. The theoretical preconditioner is as defined by Equation (5.10). .....	169
Figure 5.2. 7×7×7 mesh: Eigenvalue distribution of the preconditioned system with theoretical exact block diagonal preconditioner for different pile-soil stiffness ratios. $r$ is the number of rows with $\ row(\tilde{L}_2)\ _2 \geq 0.3$ [Equation (5.18)]. .....	170
Figure 5.3. 25×25×35 mesh: Iteration count and total CPU time of block diagonal preconditioner (5.28) for different approximations of block $P$ . .....	171
Figure 5.4. 7×7×7 mesh: Cumulative distribution of the eigenvalues (real) of the preconditioned system for different approximations of block $P$ in the preconditioner (5.28). .....	172
Figure 5.5. 25×25×35 mesh: Performance of different approximations of the soil and Schur complement blocks in the block diagonal preconditioner with exact block $P$ . .....	173
Figure 5.6. 7×7×7 mesh: Distribution of the eigenvalues of a preconditioned matrix for different approximations of soil and fluid stiffness blocks in conjunction with an exact block $P$ in the block diagonal preconditioner. $R(\lambda)$ = Real part of the eigenvalue. ....	174
Figure 5.7. 25×25×35 mesh: Effect of contrast in pile-soil permeability on the block diagonal preconditioners $M_1$ and $M_2$ . .....	175
Figure 5.8. 25×25×35 mesh: Comparison of proposed preconditioners $M_1$ and $M_2$ with GJ and MSSOR preconditioners for varying pile-soil stiffness ratios. ....	176
Figure 5.9. 25×25×35 mesh: Effect of size of the pile block $P$ (e.g. due to variation in number of piles in the piled-raft problem) on the performance of preconditioners at different pile-soil stiffness ratios. ....	177
Figure 5.10. CPU time of $M_1$ and $M_2$ preconditioners for a range of stiff DOFs and soil-structure stiffness ratios (a) $M_1$ versus GJ, (b) $M_2$ versus GJ. ....	178
Figure 5.11. CPU time of $M_1$ and $M_2$ preconditioners for a range of stiff DOFs and soil-structure stiffness ratios (a) $M_1$ versus MSSOR, (b) $M_2$ versus MSSOR. ....	179
Figure 5.12. Finite element mesh and step-by-step installation of liner in tunneling. ....	180

Figure 5.13. Comparison of iteration count and CPU time of the preconditioners for tunneling example. ....	181
Figure 5.14. 7×7×7 mesh: Sparsity pattern of 3×3 block structured A. (a) Sequential nodal numbering of nodes in x-z plane according to Smith and Griffiths (1997; 2004); and (b) Automatic nodal numbering in GeoFEA. ....	182
Figure 5.15. Surface settlement profile after 10 steps of excavation. ....	183
Figure 6.1. Westendstrasse 1 building, Frankfurt: (a) Sectional elevation (after Katzenbach <i>et al.</i> , 2000); and (b) Plan with pile layout (after Franke <i>et al.</i> , 2000). ....	205
Figure 6.2. Finite element meshes (a) mesh for entire problem domain, and (b) enlarged mesh for piled-raft. ....	206
Figure 6.3. Frankfurt subsoil stratigraphy and undrained shear strength (after Franke <i>et al.</i> , 2000). ....	207
Figure 6.4. Comparison of performance of SJ (inbuilt) and SJ (user defined) preconditioners with PCG. ....	208
Figure 6.5. Settlement due to different preconditioners with PCG. ....	209
Figure 6.6. Comparison of computed and measured settlements. ....	210
Figure 6.7. Iteration count and CPU time of different preconditioners. ....	211
Figure 6.8. Effect of soil profile on different preconditioners. ....	212
Figure 6.9. Measured time-dependent raft-pile load share for Westendstrasse 1 building, Frankfurt (after Franke <i>et al.</i> , 2000); and (b) Idealized load applied on piled-raft for consolidation analysis. ....	213
Figure 6.10. Comparison of inbuilt and user defined preconditioners with SQMR ( $k_s = 1 \times 10^{-9}$ m/s). ....	214
Figure 6.11. Settlements due to different preconditioners with SQMR ( $k_s = 1 \times 10^{-7}$ m/s). ....	215
Figure 6.12. Comparison of computed and measured settlements. ....	216
Figure 6.13. Iteration count and CPU time of PCG (for drained analysis) and SQMR (for consolidation analysis) solvers. ....	217
Figure 6.14. Iteration count and CPU time of different preconditioners. ....	218

Figure 6.15. Finite element mesh for twin tunnels: (a) isometric view; (b) Front view. ....	219
Figure 6.16. Finite element simulation procedure for Shield tunnel advancement. ....	220
Figure 6.17. Iteration count and CPU time of different preconditioners. ....	221
Figure 6.18. Surface settlement trough due to tunnel advancement. ....	222

# LIST OF SYMBOLS

---

$(\cdot)^{-1}$	inverse of a function
$(\cdot)^T$	transpose of a function
$ \cdot $	absolute value or modulus of a number
$\ \cdot\ $	norm of a function
$\ \cdot\ _F$	Frobenius norm of a function
$\ \cdot\ _p$	p-norm of a function, where $1 \leq p \leq \infty$
1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
$a$	cross-section area
$a_{ij}$	$(i, j)$ entry of matrix $A$
$A$	general matrix; matrix variable
$\tilde{A}$	preconditioned matrix $A$
AINV	approximate inverse
$b$	right hand side vector
$\tilde{b}$	preconditioned right hand side vector
$B$	displacement and pore pressure coupling matrix
$B_1$	coupling matrix between structural displacement and pore pressure DOFs
$B_2$	coupling matrix between soil displacement and pore pressure DOFs
$B_u$	shape function derivative for displacement

BE	boundary element
Bi-CG	biconjugate gradient
Bi-CGSTAB	biconjugate gradient stabilized
BL	base-line (preconditioner)
Blk	block ordering
$c'$	effective cohesion
$c_u$	undrained cohesion
$C$	flow matrix
CG	conjugate gradient
CGS	conjugate gradient squared
CPU	central processing unit
CSC	compressed sparse column
CSR	compressed sparse row
$\det(\cdot)$	determinant of a function
$diag(\cdot)$	diagonal matrix consisting of leading diagonal entries in argument
$D$	effective stress-strain matrix
$\hat{D}$	modified diagonal
$\tilde{D}$	diagonal matrix variable
$\bar{D}$	diagonal matrix with pivots
$D_A$	diagonal matrix whose diagonal entries are identical to those of matrix $A$
$D_{ps}$	effective stress-strain matrix of porous stone
$D_s$	effective stress-strain matrix of soil
D-MCP	diagonal variant of mixed constraint preconditioner
DOFs	degrees of freedom

$e$	index for element number, a vector with ones
$e_k$	error vector at $k$ -th iteration
$E'$	effective Young's modulus
$E'_{clay}$	effective Young's modulus of clay
$E'_p$	effective Young's modulus of pile
$E'_p/E'_s$	soil-structure stiffness ratio
$E'_s$	effective Young's modulus of soil
$E'_{sand}$	effective Young's modulus of clay
EBE	element-by-element
$f$	load vector
$F$	applied load
FE	finite element
FEM	finite element element
$G$	soil effective stiffness matrix
$\hat{G}$	approximate matrix $G$
GJ	generalized Jacobi
GMRES	generalized minimal residual
$H$	matrix variable
$i, j, k$	integer variables
$I$	identity matrix
$I_{(.)}$	identity matrix of the size of the argument
IC	incomplete Cholesky decomposition
IC0	incomplete Cholesky decomposition with no fill-in
ICP	inexact constraint preconditioner
ILLT	symmetric incomplete LU decomposition
ILU	incomplete LU decomposition

ILU0	incomplete LU factorization with no fill-in
ILU0( $\cdot$ )	ILU0 factorization of the matrix in the argument
ILUT	Incomplete LU factorization with dual control parameters for fill-in
$k$	coefficient of permeability
$k_{clay}$	coefficient of permeability of clay
$k_p$	coefficient of permeability of pile
$k_s$	coefficient of permeability of soil
$k_{sand}$	coefficient of permeability of sand
$k_x, k_y$ and $k_z$	coefficient of permeability in x-, y- and z-directions, respectively
$K'$	effective bulk modulus of soil
$K$	effective stress stiffness matrix
$\hat{K}$	symmetric positive definite approximation of $K$
$\tilde{K}$	preconditioned matrix $K$
$K^e$	element effective stress stiffness matrix
$K_w$	bulk modulus of water
$l$	size of 1D element
$L$	soil-structure link matrix
$\tilde{L}$	preconditioned matrix $L$
$\bar{L}$	lower triangular factor
$L_{(\cdot)}$	matrix variable
$L_A$	strictly lower triangular part of $A$
$\bar{m}$	an equivalent of the Kronecker delta; $\bar{m} = [1]$ for 1-D analyses, $[1 \ 1 \ 0]^T$ for 2-D analyses, and $[1 \ 1 \ 1 \ 0 \ 0 \ 0]^T$ for 3-D analyses

$m$	the dimension of block $P$ (e.g. pile DOFs or liner DOFs)
$\max(\cdot)$	maximum value of a function
$\min(\cdot)$	minimum value of the function
$M$	preconditioner or preconditioning matrix
$M_{(\cdot)}$	preconditioner of argument
$M_{BL}$	base-line preconditioner
$M_c$	block constrained preconditioner
$M_d$	block diagonal preconditioner
$M_L$	left preconditioner
$M_R$	right preconditioner
$M_t$	block triangular preconditioner
MCP	mixed constraint preconditioner
MINRES	minimal residual
MJ	modified Jacobi
ModMCP	diagonal variant of mixed constraint preconditioner
MPa	mega Pascal
MSSOR	modified symmetric successive over-relaxation
$n$	soil DOFs (the dimension of block $G$ )
$nd$	total displacement DOFs (the dimension of block $K$ )
$np$	pore pressure DOFs (the dimension of block $C$ )
$N$	the dimension of total linear system, e.g. the dimension of matrix $A$
$N_u$	shape function for displacement
$\bar{N}$	shape function for excess pore water pressure
Nat	natural ordering
NATM	New Austrian Tunneling Method

NDF	total number of degrees of freedom including fixities, if any
$O(\cdot)$	order of a function
OCR	overconsolidation ratio
$p$	excess pore water pressure vector
$P$	effective stiffness matrix of structural elements (or stiff material)
$\bar{P}$	permutation matrix
$\hat{P}$	approximate matrix $P$
PC(s)	personal computer(s)
PCG	preconditioned conjugate gradient
$Q$	a matrix
QMR	quasi-minimal residual
QMR-CGSTAB	quasi-minimal residual variant of Bi-CGSTAB
$r$	integer variable
$r_k$	residual vector at $k$ -th iteration;
$R$	upper triangular factor
$R_{(\cdot)}$	Cholesky factor of the matrix in the argument
$R_E$	relative error norm
$R_i$	relative improvement norm
$R_r$	relative residual norm
$R(\lambda)$	real part of the eigenvalue
RAM	random access memory
RCM	reverse Cuthill McKee (ordering)
R-Nat	natural ordered variables reordered by RCM algorithm
$S$	Schur complement matrix

$\tilde{S}$	Schur complement matrix
$\hat{S}$	symmetric positive definite approximation of Schur complement matrix $\tilde{S}$
$\hat{S}_1$	symmetric positive definite approximation of Schur complement matrix $S$
SBD	simplified block diagonal
SJ	standard Jacobi
SPD	symmetric positive definite
SQMR	symmetric quasi-minimal residual (solver)
SSOR	symmetric successive over-relaxation
SSOR( $\cdot$ )	SSOR factorization of the matrix in argument
stab-ILU0	stabilized incomplete LU factorization with no fill-in
SYMMLQ	symmetric LQ
$t$	time
TFQMR	transpose-free quasi-minimal residual
T-MCP	triangular variant of mixed constraint preconditioner
$x$	local spatial coordinate; vector variable
$x_0$	initial guess for solution
$x_k$	solution vector at $k$ -th iteration
$X$	matrix variable
$u$	unknown function; displacement vector
$U$	matrix variable
$\overline{U}$	upper triangular factor
$U_A$	strictly upper triangular part of $A$
$v$	a vector
$V$	volume domain; matrix variable

$W$	matrix variable
$\widehat{W}$	matrix variable
$y$	vector variable
$z$	an integer variable
$Z$	upper triangular factor
$\alpha$	scalar
$\beta$	scalar
$\Delta p$	pore water pressure increment
$\Delta t$	time integration step
$Y$	a matrix variable
$\Delta u$	displacement increment vector
$\Delta \epsilon$	strain increment vector
$\Delta \sigma$	stress increment vector
$\Delta \sigma_1$	vertical stress increment
$\Delta \sigma_3$	horizontal stress increment
$\epsilon$	perturbation to a value; strain vector
$\epsilon_x, \epsilon_y$ and $\epsilon_z$	normal strain in X-, Y- and Z-directions, respectively
$\phi'$	effective angle of friction
$\gamma$	scalar; bulk unit weight
$\gamma_{\text{bulk}}$	bulk unit weight
$\gamma_w$	unit weight of water
$\eta$	scalar
$\lambda$	eigenvalue
$\lambda_{\text{max}}$	maximum eigenvalue
$\lambda_{\text{min}}$	minimum eigenvalue

$ \lambda $	modulus of eigenvalue
$ \lambda _{\max}$	maximum modulus of eigenvalue
$ \lambda _{\min}$	minimum modulus of eigenvalue
$\Lambda$	diagonal matrix
$\nu$	Poisson's ratio
$\nu'$	effective Poisson's ratio
$\nu'_p$	effective Poisson's ratio of pile
$\nu'_s$	effective Poisson's ratio of soil
$\delta$	scalar
$\rho$	scalar
$\rho_K$	the number of terms stored in each row of the factorization in excess to the non-zeroes of $K$
$\rho_S$	the number of terms stored in each row of $\hat{S}$ in excess to the non-zeroes of $C$
$\rho_{SI}$	the number of terms stored in each row of the factorization in excess to the non-zeroes of $\hat{S}$
$\sigma_x, \sigma_y$ and $\sigma_z$	normal stress in X-, Y- and Z-directions, respectively
$\sigma'_x, \sigma'_y$ and $\sigma'_z$	effective normal stress in X-, Y- and Z-directions, respectively
$\tau_{xy}, \tau_{yz}$ and $\tau_{xz}$	shear stress in XY-, YZ-, and XZ-directions, respectively
$\tau$	scalar
$\tau_Z$	the fraction of the Z diagonal terms below which an extra-diagonal coefficient is dropped in AINV factorization
$\omega$	scalar, a relaxation parameter for SSOR factorization
$\xi$	scalar
$\partial(\cdot)$	partial derivative of a function
$\sum_i(\cdot)$	summation of a function over the range of index $i$

$\sum_{i=1}^n (\cdot)$	summation of a function from index $i = 1$ to $i = n$
$\Re$	set of real numbers
$\Re^N$	vector space of real $N$ -vectors
$\Re^{N \times N}$	vector space of real $N$ -by- $N$ matrices

*To my parents and wife*

*(blank)*

# Chapter 1

---

## INTRODUCTION

### 1.1. Introduction

Traditionally geotechnical design has been carried out using simplified analyses and empirical approaches. Within the past three decades, various numerical techniques have been developed and successfully applied to a wide range of geotechnical problems. Among them, much progress has been made in modeling the behavior of soil and understanding the mechanism of soil-structure interaction using finite element method. The application of finite element method has been proven to be successful in modeling of nonlinear behavior of soils, soil-structure interaction problems, including accounting for the construction sequences (e.g. Balasubramaniam *et al.*, 1992; Potts and Zdravković 1999).

Although the nature of most geotechnical problems is three-dimensional, many simplified analyses have been frequently used in design practice and for the finite element (FE) analyses for last several decades. Most of the FE analyses conducted in geotechnical engineering assume plane strain or axisymmetric conditions. Such an assumption allows a two-dimensional

(2D) treatment of a real three-dimensional (3D) problem for which several computer codes and examples have already been published (e.g. Zienkiewicz *et al.*, 1969; Nayak and Zienkiewicz, 1972; Britto and Gunn, 1987; Smith and Griffiths, 1997) and a number of commercial geotechnical finite element softwares are available (e.g SAGE-CRISP, 2000; GeoFEA, 2006; SIGMA/W, 2007; PLAXIS 2D, 2009). However, real problems, such as those encountered in underground construction works or pile-group foundations, are often intrinsically three-dimensional (3D) in nature and the complete 3D analysis cannot be avoided in many situations (Potts and Zdravković 2001; Brinkgreve and Broere, 2006) mainly because of three reasons: (a) complex interactions between soil and structure; (b) complexity in problem geometry; and (c) spatial variation of soils. Consideration of three-dimensional effects arises particularly due to increased urbanization where various underground structures and high-rise buildings are being erected at a very close proximity to existing structures because of increasing need for office/residence space in a rather small city area. The corresponding geotechnical risks are significantly aggravated by the presence of rather compressible clay layer of significant thickness (over 40 m in some areas, e.g. Singapore, Bangkok, Frankfurt). Hence, a rigorous 3D analysis is necessary to cope with the complexity of these intrinsically 3D problems. At the same time, 3D analysis is generally considered prohibitive to perform because it requires a large amount of computational time and storage (Papadrakakis, 1993b; Smith and Griffiths, 1997; Janna *et al.*, 2009).

For example, in piled-raft foundation, the interaction between pile, raft, and soil is important for supporting the load from upper structure. Such a

situation can only be modeled effectively by means of three dimensional finite element calculations. However, because of the limitation of available computing resources and proper algorithms, a number of simplified calculation methods have been developed over the last three decades in order to minimize the computer memory required to simulate the real 3D behavior for the analyses of load bearing and settlement behavior of piled-raft foundation. Poulos (2001b) categorized these methods into three broad classes: simplified calculation methods (Randolph and Wroth, 1978; Poulos and Davis, 1980); approximate computer-based methods (Clancy and Randolph, 1993; Poulos, 1994); and more rigorous computer-based methods (Butterfield and Banerjee, 1971; Ottaviani, 1975; Kuwabara, 1989; Smith and Wang, 1998). For more details about these methods, the reader is referred to the report by Technical Committee TC18 of the International Society of Soil Mechanics and Geotechnical Engineering (Poulos, 2001b). Comparison of some of these methods shows that the type and quality of results depend on the capabilities of the applied method (e.g. Poulos *et al.*, 1997). Hence, recently, there has been an increasingly use of 3D analysis of the piled-raft problems (e.g. Maleki Javan *et al.*, 2008; Small and Liu, 2008).

Similarly, Finite Element Method (FEM) has been frequently used to model tunneling construction and its effects such as surface settlement, etc. Although tunneling is a three-dimensional process, two-dimensional analyses of tunneling are often used in the practice because of limitations of hardware and software in the past. As a result, a number of two-dimensional FE simplifications have been developed to model the 3D tunnel, for example, axis-symmetric analysis (Rowe and Lee, 1992), plane strain analysis (Pan and

Hudson, 1988; Burd *et al.*, 1994; Addenbrooke *et al.*, 1997). However, many assumptions are required in 2D analysis in order to replicate the real 3D tunnel behavior (Potts and Zdravković 2001). Hence, some researchers have also studied 3D analysis of tunnels (e.g. Katzenbach and Breth, 1981; Lee and Rowe, 1990; Dasari *et al.*, 1996). Recent trend shows that there is a proliferation use of 3D analyses of tunnels (e.g. Galli *et al.*, 2004; Lee *et al.*, 2006; Phoon *et al.*, 2006; Mroueh and Shahrou, 2008; Migliazza *et al.*, 2009).

As mentioned earlier, three-dimensional FE analyses are computationally expensive because a large number of finite elements are required to represent realistically a 3D behavior. This may generate a few tens of thousands to millions degrees of freedom (DOFs), or FE equations. In general, this system of equations is condensed in the following form:

$$Ax = b \quad (1.1)$$

where  $A \in \Re^{N \times N}$  is known as coefficient matrix,  $x \in \Re^N$  is the vector of unknowns, and  $b \in \Re^N$  is the force vector.  $N$  is the total dimension of the linear system. Solution of this system of equations (1.1) is computationally one of the most expensive parts in the finite element analysis. For this reason, efficient and economical solution of the linear system is essential for making 3D finite element analysis to be routinely used in practice. This linear system is solved mainly in two ways: direct method or iterative method. For 1D or 2D FE modeling, the resulting linear system is usually small and the direct solution method [e.g. Gaussian elimination approach or Frontal solver (Irons, 1970)] is always preferred due to its robustness and effectiveness. It has been the basis for many finite element programs. However, for large-scale geotechnical problems, such as those arising from 3D analyses, the size of the

linear system is significantly large. The large memory requirement may limit the application of direct solution method for large-scale (3D) analysis and the out-of-core facility may significantly slow down the computing speed (Lee *et al.*, 2006). For such problems, iterative solvers are helping to meet these demands.

## **1.2. Iterative solvers and the role of preconditioning in geotechnical problems**

In an iterative solution method, a solution guess is provided and is refined iteratively until the solution is sufficiently close to the exact solution. In recent years, there is an increasing interest in the use of iterative rather than direct solvers for 3D geotechnical finite element analyses. Among the most desirable advantages of iterative methods are the low storage (computer memory) requirements and shorter CPU time for the solution of linear FE equations compared with direct methods (e.g. Papadrakakis, 1993b). Whether or not this happens depends on the nature of the coefficient matrix  $A$  (1.1) of the linear system of equations and the preconditioning. In most geotechnical problems, the coefficient matrix can be severely ill-conditioned (see Appendix A for definitions of some algebraic terms), thus calling for the development of robust and efficient preconditioners. A preconditioner is the key to the success of iterative methods. A preconditioner is another matrix which transforms the original linear system to a more favorable linear system and accelerates the convergence of an iterative solution (Axelsson, 1994; Barrett *et al.*, 1994; Kelley, 1995; Saad, 1996; Greenbaum, 1997; Saad and Van Der Vorst, 2000).

Hence, in the preconditioned iterative solution, the transformed system (1.2) is solved instead of the original linear system (1.1):

$$M^{-1}Ax = M^{-1}b \quad (1.2)$$

where  $M$  is the preconditioner.

Probably the first application of a preconditioned iterative method to geotechnical problems may be by Smith *et al.* (1989) and Wong *et al.* (1989). They used preconditioned conjugate gradient (PCG) solver (Hestenes and Stiefel, 1952) in conjunction with different preconditioners for the solution of first order and second order transient problems. The issue of preconditioning in computational geomechanics has been addressed in a number of recent works. For drained boundary value problems, Mroueh and Shahrour (1999) studied the application of standard Jacobi (SJ) and Symmetric Successive Over-Relaxation (SSOR) preconditioners in conjunction with bi-conjugate gradient (Bi-CG) (Lanczos, 1952), bi-conjugate gradient stabilized (Bi-CGSTAB) (van der Vorst, 1992), and quasi-minimal residual variant of Bi-CGSTAB (QMR-CGSTAB) (Chan *et al.*, 1994) iterative solvers for the resolution of 3D soil-structure interaction problems. They concluded that the SSOR preconditioner performs better in comparison to SJ preconditioner for soil-structure interaction problems with highly varied material heterogeneity and plasticity. Payer and Mang (1997) investigated the three preconditioners, namely, diagonal scaling, SSOR, and ILU-type preconditioner with conjugate gradient squared (CGS) method (Sonneveld, 1989), Bi-CGSTAB, and generalized minimum residual (GMRES) method (Saad and Schultz, 1986) for hybrid boundary element-finite element solution of tunneling problem. Based on which, they concluded the hierarchical use of above preconditioners

depending on the problem complexity. Similarly, the SJ preconditioned CGS and GMRES were shown to be more efficient and robust than direct solution method in solving underground construction problems (Kayupov *et al.*, 1998). However, the investigation of SJ preconditioner on three possible geotechnical loading conditions (namely, drained, undrained, and consolidation) by Lee *et al.* (2002) showed that the SJ performs well for the drained problems, but is much less effective for undrained, and counter productive for consolidation problems.

In coupled consolidation analysis, the coefficient matrix  $A$  can be severely ill-conditioned, especially in the early stage of the process where small time steps are required to obtain an accurate transient solution (Chan *et al.*, 2001; Ferronato *et al.*, 2001; Ferronato *et al.*, 2009). Thus, considerable efforts have been made in the development of preconditioning techniques for coupled consolidation problems in recent years. For example, several diagonal preconditioners were proposed in conjunction with symmetric quasi-minimal residual (SQMR) solver (Freund and Nachtigal, 1994b) for the symmetric indefinite linear systems produced by 3D Biot's consolidation equations. The heuristic preconditioner, Modified Jacobi (MJ), by Chan *et al.* (2001) was based on the observation that the standard Jacobi preconditioned SQMR actually performed worse than the unpreconditioned version when the diagonal elements corresponding to flow stiffness matrix is close to zero. In 2002, Phoon *et al.* proposed the generalized Jacobi (GJ) preconditioner, which is an improvement over MJ from both theoretical and numerical perspectives. Effectiveness of GJ to a variety of geotechnical problems has been demonstrated in a number of papers by Phoon and co-workers (Phoon *et al.*,

2003; Phoon, 2004; Lee *et al.*, 2006; Phoon *et al.*, 2006). Observing the breakdown of conventional SSOR preconditioner for consolidation problems, a modified version of the SSOR preconditioner (MSSOR) was proposed by Chen *et al.* (2006) by replacing the original diagonal by GJ in SSOR factorization. Numerical results show that the MSSOR can lead to faster convergence than the GJ preconditioner (Chen *et al.*, 2007) or block constrained preconditioner (Toh *et al.*, 2004, will be discussed later). The most promising advantage of diagonal preconditioners is that they do not incur an additional memory and easy to use in any PC environment. However, the performance of these preconditioners degrades for heterogeneous soil profiles or for soil-structure interaction problems when significant contrasts in stiffness of the materials exist (Chen *et al.*, 2007).

Another type of preconditioners that is commonly encountered for Biot's equations are ILU-type and IC-type incomplete triangular factorization preconditioners (Ferronato *et al.*, 2001; Gambolati *et al.*, 2001, 2002, 2003). Their numerical results suggest that although ILU0 preconditioner accelerates the convergence of Bi-CGSTAB solver, it may breakdown for ill-conditioned systems. The effect of time integration steps on ill-conditioning of the system (Ferronato *et al.*, 2001) and the effect of ordering of the variables on the performance of ILU preconditioners (Gambolati *et al.*, 2001) were taken into account. An optimum ILUT preconditioner (with variable fill-in) was shown to have overcome the problems of ILU0 and accelerates the convergence. However, for the optimal performance of ILUT preconditioners, the user-specified parameters (which control the number of fill-ins) can only be found

empirically via a trial-and-error procedure. Hence, it may be worth comparing the pros and con of ILU0 preconditioner to that of MSSOR preconditioner.

A more recent development for the iterative solution of ill-conditioned coupled consolidation problems is the block preconditioners that exploit the block structure of the coefficient matrix. In fact, the development of GJ or MJ was also based on block structure of  $A$ . Toh *et al.* (2004) systematically investigated three common block preconditioners, namely, block diagonal, block triangular and block constrained preconditioners with different approximation of blocks. A comparison of performance of GJ (Phoon *et al.*, 2002) and block constrained preconditioner (Toh *et al.*, 2004) showed that the latter can be about two times faster than the former, but at the cost of more memory (Phoon *et al.*, 2004). Similar to the work of Toh *et al.* (2004), an inexact constraint preconditioner (ICP) was proposed more recently by Bergamaschi *et al.* (2007) in conjunction with Bi-CGSTAB solver. Their numerical results showed that the ICP preconditioner can be up to more than two times faster than the standard ILU/ILUT preconditioners (Saad, 1994b, 1996) for large-scale 3D problems. Another variant of constrained preconditioner is the so-called mixed constraint preconditioner (MCP) (Bergamaschi *et al.*, 2008) whose basic idea relies on approximating independently the inverse of the structural submatrix and the global Schur complement. However, a major limitation of these preconditioners for practical use is the optimal selection of a number of user-defined parameters (at least 4) (e.g. Ferronato *et al.*, 2010), which is problem dependent. These preconditioners also require relatively more memory in comparison to diagonal preconditioners.

Much of the work described earlier is based on assembled coefficient matrix  $A$  and such preconditioners are commonly known as global preconditioners. A brief description, formulation, and implementation of these global preconditioners are summarized in Chapter 2. An interesting trade-off between the above preconditioning strategies with the aim of limited core memory requirement is the element-by-element (EBE) technique. In EBE implementation, the matrix-vector multiplication is operated at the element level and assembly of the global matrix is not required. Thus, the computer memory usage is significantly reduced. For example, the application of Jacobi preconditioner in EBE strategies has been demonstrated for the iterative solution of a range of geotechnical problems (Smith *et al.*, 1989; Wong *et al.*, 1989; Smith and Griffiths, 1997; Smith and Wang, 1998; Chan, 2002; Lim, 2003; Smith and Griffiths, 2004). Some new EBE preconditioners have been proposed recently (Augarde *et al.*, 2006, 2007) that enhances the convergence better than conventional EBE-Jacobi preconditioning. However, these preconditioners are more suitable for parallel computation using multiple processors. This study focuses on PC based computing platform. Hence, only global preconditioners based on global assembled coefficient matrix are considered in this thesis.

### **1.3. Scope and objective of the study**

As mentioned earlier, direct methods do not appear to be the choice for large-scale 3D problems because of their large memory requirement and preconditioned iterative method is one of the promising approaches for the repeated solution of such linear systems, which are often ill conditioned, in an

efficient way. The overall objective of this study is to allow large-scale 3D finite element modeling of geotechnical engineering problems to be performed economically, both in terms of computational time and resources, so that average practitioners can afford to simulate large and complex problems realistically in a normal PC environment.

The brief review in the preceding Section shows that there has been tremendous development in the use of preconditioned iterative solutions in the last decade, and several preconditioning strategies (e.g. diagonal preconditioning, incomplete factorization preconditioning, block constrained preconditioning, EBE preconditioning) have been proposed with specific application to the geotechnical problems. However, until now, much of the work has been focused on the preconditioners for general consolidation problems and ill-conditioning due to small time-steps and low permeability materials. Only limited attention has been paid on the effect of material properties on the performance of iterative solution (e.g. Augarde *et al.*, 2008). It was observed that the performance (iteration count and runtime) of SJ (Mroueh and Shahrour, 1999; Lee *et al.*, 2002) and SSOR preconditioners (Payer and Mang, 1997) is significantly affected by the material properties, especially due to the contrast in Young's moduli of materials, for drained boundary value problems. Similarly, the performance of GJ and MSSOR is also found to be affected by the heterogeneity of materials in consolidation analysis (e.g. Chen *et al.*, 2007).

Materials with differing stiffness are commonly encountered in pragmatic geotechnical problems because of the variability of natural geomaterials and the involvement of stiff structural elements. Young's

modulus can vary from 1 MPa or less for soft soils to more than 100,000 MPa for rocks. A similar difference in stiffness is common in soil-structure interaction problems where the Young's modulus of a structural material (e.g. reinforced concrete, steel) can be more than four to five orders of magnitude larger than the Young's modulus of the surrounding soil. Such a large stiffness contrast can produce a severely ill-conditioned system (Lee *et al.*, 2002). In consolidation analysis, besides stiffness, there can be a significant contrast in permeabilities of materials. Thus, there is considerable scope for an improvement in the preconditioning technique to minimize the adverse effects due to these differences in material properties to allow fast 3D simulation of large-scale problems involving heterogeneous materials, such as in soil-structure interaction problems.

The specific objectives of this study can be summarized as follows:

1. To compare the MSSOR and ILU0 preconditioning strategies used for Biot's consolidation equations.

Since different preconditioning strategies have been used by different researchers for the effective solution of Biot's consolidation equations, this study sheds light on the advantages and disadvantages of these methods. A thorough investigation of these may help a practicing engineer to select a preconditioner that best suits the problem based on available resources.

2. To develop a preconditioner that mitigates the effect of differences in stiffnesses of materials in drained analysis.

To simplify the problem, a drained boundary value problem was considered first and the source of ill-conditioning due to differences in stiffness of materials was investigated with the help of 1-Dimensional oedometer example. For this, a theoretical block diagonal preconditioner was derived that possesses an attractive eigenvalue clustering property. However, this theoretical form is very expensive. Hence, practical simplified block diagonal (SBD) preconditioners were proposed that are less expensive. The proposed preconditioners effectively mitigated the ill-conditioning due to relative differences in stiffness of materials in the linear system.

3. To develop a preconditioner that mitigates the coupled effect of differences in stiffness and permeability of materials in consolidation analysis.

Given the considerable potential usefulness of the proposed preconditioners from objective (2), the study was extended to cover a more general framework described by Biot's consolidation equations. For consolidation analysis, the permeability can vary (order of magnitude) from 1 m/s for a stone column to  $10^{-12}$  m/s for unsaturated soils. The coupled effects of large relative differences in stiffnesses and permeabilities of materials may produce an even more severely ill-conditioned system. First, a block diagonal preconditioner was derived and shown to have an attractive eigenvalue property theoretically. Finally, some cost-effective approximate block diagonal preconditioners were investigated which effectively mitigate the ill-conditioning due to such variation in material properties.

4. To evaluate the effectiveness of the proposed block diagonal preconditioners in the context of realistic large-scale soil-structure interaction problems.

The effectiveness and general applicability of the proposed preconditioners for a wide range of practical geotechnical problems was demonstrated with the help of two case histories soil-structure interaction problems: a piled-raft foundation problem and a twin tunnel problem.

The mitigation of the effect of material properties on preconditioning by the proposed block diagonal preconditioners has significant impact on saving the computational time over existing SJ, SSOR, and ILU0 preconditioners in drained analysis, and over GJ, MSSOR, and ILU preconditioners in consolidation analysis. Their general applicability to case history examples with varying soil properties suggests that the proposed preconditioners are potentially useful for the problems beyond the examples covered in this study, whenever a difference in Young's moduli of materials exists. However, only a symmetric linear system is considered in this study.

## **1.4. Computer hardware and software**

All the numerical experiments in this thesis are carried out on a DELL Intel Core Duo CPU, 2.4GHz PC with 2GB of RAM running on a Windows XP operating system.

The Fortran source codes used for 3D finite element analysis of Biot's consolidation problems are based on research work by Chen (2005). These

finite element codes are compatible with the codes of Smith and Griffiths (1997). The results of Chapter 3 were obtained by using Compaq Visual Fortran, Professional Edition 6.5.0. Results of all other Chapters were obtained with Intel Visual Fortran Compiler 10.1, Professional Edition. A commercial software GeoFEA (2006) was used for the simulation of complex soil-structure interaction problems, where appropriate preconditioned iterative solvers were implemented as user-defined solvers.

Other publicly available software packages/libraries used for this report are:

1. SPARSKIT: A basic tool-kit for sparse matrix computations. The software package can be obtained from Yousef Saad's homepage:

<http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html>

2. SparseM: The software package is a basic linear algebra package for sparse matrices and can be obtained from:

<http://cran.r-project.org/src/contrib/Descriptions/SparseM.html>

3. Template. It is a package for some popular iterative methods in Fortran, Matlab and C, and can be used to demonstrate the algorithms of the book "Templates for the solution of linear systems: Building blocks for iterative methods."

<http://www.netlib.org/templates/>

4. RCM. This is a Fortran 90 library of routines which computes the reverse Cuthill McKee (RCM) ordering of the nodes of a graph. The package is maintained by Burkardt J.

[http://people.scs.fsu.edu/~burkardt/f\\_src/rcm/rcm.html](http://people.scs.fsu.edu/~burkardt/f_src/rcm/rcm.html)

5. SPARSEPAK. Waterloo sparse matrix package. It is a library of Fortran 90 routines for solving large sparse systems of linear equations.

[http://people.scs.fsu.edu/~burkardt/f\\_src/sparsepak/sparsepak.html](http://people.scs.fsu.edu/~burkardt/f_src/sparsepak/sparsepak.html)

## 1.5. Thesis outline

The material covered in this thesis has been divided up into following logical chapters. Chapter 2 provides a brief overview of iterative methods used in this thesis and review of various preconditioners, convergence criteria, etc. Chapter 3 compares the performance of recently developed MSSOR preconditioner (Chen *et al.*, 2006) with that of standard ILU0 preconditioner and highlights the merits and demerits of each preconditioner for the solution of finite element Biot's consolidation equations. In Chapter 4, the effect of relative difference in material stiffnesses, such as in soil-structure interaction problems, on the iterative solution and its effective mitigation by block diagonal preconditioners are discussed. Only drained boundary value problems are considered to isolate the sole ill-conditioning due to contrasts in stiffnesses only. Owing the effectiveness of block diagonal preconditioning, it is extended to the coupled consolidation analysis in Chapter 5 for the mitigation of coupled effect due to contrasts in both stiffness and permeability of the materials. The application of these preconditioners to two case histories examples is demonstrated in Chapter 6. Finally, Chapter 7 offers some general conclusions with recommendations for the further study.

Appendix A provides the definition of various algebraic terms used in this thesis.

Appendix B explains the finite element discretization of Biot's consolidation equations

Appendix C comprises the algorithms or pseudo codes used in this thesis.

Appendix D details the effect of soil-structure stiffness ratio in 1D example.

Appendix E includes the source codes used for this research.

Appendix D includes the source code for user defined solver in GeoFEA.

*(blank)*

## CHAPTER 2

---

### LITERATURE REVIEW

Finite element analyses of geotechnical problems can broadly be categorized into three types of analyses: drained, undrained, and consolidation depending upon the loading and drainage conditions. For example, sand usually exhibits a drained behaviour because of its high permeability. Drained and undrained are two extreme loading conditions (long term and short term, respectively), while consolidation is the intermediate condition. Note that all the above loading conditions can be represented by Biot's (1941) coupled consolidation equations depending on the time integration step and have wide applications in many engineering problems (e.g. Abbo, 1997; Lewis and Schrefler, 1998). The repeated solution in time of the linear system arising from finite element (FE) discretization of the coupled consolidation equations is the most time consuming computational effort in geotechnical engineering analysis. This could be the reason that the development of most of the preconditioners for geotechnical problems is based on Biot's consolidation equations. See Appendix B for the finite element discretization of the Biot's consolidation

equations. The succeeding Sections provide a review of iterative methods and various preconditioning approaches used in geotechnical engineering analysis.

## 2.1. Iterative solution methods

The linear systems produced by finite element analysis of geotechnical problems can broadly be grouped into definite and indefinite linear systems. The linear system may be symmetric or unsymmetric depending on the constitutive model used for the soil and the formulation of equations. For example, the linear system arising from the FE integration of coupled consolidation equations can also be written in a form of symmetric indefinite (Smith and Griffiths, 1997), unsymmetric indefinite (Gambolati *et al.*, 2001), or unsymmetric positive definite (Ferronato *et al.*, 2009). Although these different formulations are mathematically equivalent, numerically they are not, and different iterative solvers and preconditioners are required. Several iterative methods are available (see, for instance, Barrett *et al.*, 1994) and the choice of an optimal method largely depends on the properties of the coefficient matrix  $A$  [Equation (1.1)].

If an unsymmetric form of Biot's equations were used, Bi-CGSTAB (van der Vorst, 1992) would have been a robust and efficient alternative provided that appropriate preconditioners are used (Gambolati *et al.*, 2001, 2002, 2003). However, based on the study of Ferronato *et al.* (2007), and Toh and Phoon (2008) for symmetric and unsymmetric forms of Biot's consolidation equations, symmetric form is preferable over the unsymmetric form because the former requires less computer memory for storage and a cheaper Krylov subspace method (see Appendix A for the definition) to solve.

Thus, only symmetric linear system of the consolidation equations (Appendix B) is considered in this study, which reads as:

$$\begin{bmatrix} K & B \\ B^T & -C \end{bmatrix} \begin{Bmatrix} \Delta u \\ \Delta p^{ex} \end{Bmatrix} = \begin{Bmatrix} \Delta f \\ Cp^{ex} \end{Bmatrix} \quad (2.1)$$

or, in the compact form:

$$Ax = b. \quad (2.2)$$

Here,  $A \in \Re^{N \times N}$  is a sparse  $2 \times 2$  block symmetric indefinite matrix,  $K \in \Re^{nd \times nd}$  is soil stiffness matrix (symmetric positive definite),  $C = \theta \Delta t H \in \Re^{np \times np}$  is fluid stiffness matrix (symmetric positive semi-definite), and  $B \in \Re^{nd \times np}$  is the displacement-pore pressure coupling matrix. The existing Krylov subspace methods for solving symmetric indefinite linear system include the MINRES (minimum residual) and SYMMLQ (symmetric LQ) methods proposed by Paige and Saunders (1975), as well as a recently developed SQMR (symmetric quasi-minimal residual) method by Freund and Nachtigal (1994b). The convergence of PCG (preconditioned conjugate gradient) (Hestenes and Stiefel, 1952) is not guaranteed for such systems (Paige and Saunders, 1975; Golub and Van Loan, 1989; Barrett *et al.*, 1994). Similarly, the MINRES and SYMMLQ require the use of symmetric positive definite preconditioners, which is rather unnatural restriction when the matrix itself is highly indefinite. Thus, the SQMR is used in this study because it can be combined with indefinite preconditioners, has more stable iterations and usually converges faster than the MINRES and SYMMLQ (Freund and Nachtigal, 1994a, b; Freund, 1997).

Although coupled consolidation equations can be generalized for all kinds of analyses (drained/undrained/consolidation) depending on the loading

and drainage conditions, the drained and undrained (modeled as a nearly incompressible problem) analysis can also be performed using elasticity equations. In this case, the linear system is symmetric positive definite. Solving a definite system is, in general, easier than an indefinite one, unless the system is very ill-conditioned such as that from the undrained analysis with Poisson's ratio,  $\nu \approx 0.5$ . A large amount of powerful techniques are available that can effectively solve the definite linear systems. For example, the Cholesky factorization is about a factor of two faster than other alternative methods (Press *et al.*, 1992) in the direct approach. In iterative approach, PCG (preconditioned conjugate gradient) method proposed by Hestenes and Stiefel (1952) or ChebyShev iteration methods can be used (Barrett *et al.*, 1994). However, PCG is known to be the best for symmetric positive definite linear systems (Papadrakakis, 1993a; Barrett *et al.*, 1994). Its excellent performance is due to its short recurrences in the Krylov subspace and the minimization properties that guarantee a monotonic and regular convergence with economical storage requirement. Thus, the PCG is used for the drained analysis of all the problems in this thesis. See Appendix A for more details about the methods used in this thesis. A guideline for the selection of an appropriate method is provided in Figure 2.1 based on the flowchart by Barrett *et al.* (1994). The implementation of the preconditioned iterative solvers in the finite element modeling is as shown in Figure 2.2.

## 2.2. Preconditioning strategies

If the iterative methods are applied directly to the original linear system (1.1), the rate of convergence may be slow or may even not converge. The rate of

convergence of iterative methods depends on the eigenvalue (spectrum) of the coefficient matrix  $A$ . Many iterative methods have convergence characteristics which vary substantially with the condition number of  $A$ . In general, the larger the condition number, the more likely is the failure to converge.

A preconditioner is a matrix that transforms the original linear system (1.1) to a more favorable linear system to accelerate the convergence of an iterative solution. Hence, it is the key for the success of an iterative method. A preconditioner can be applied in three formats depending on its position with respect to  $A$ . For example, given a preconditioner  $M$ , the preconditioned system can be written as:

$$M^{-1}A x = M^{-1}b \quad (2.3)$$

or,

$$AM^{-1}(Mx) = b \quad (2.4)$$

or,

$$M_1^{-1}AM_2^{-1}(M_2x) = M_1^{-1}b \quad (2.5)$$

where,

$$M = M_1M_2 \in R^{N \times N}. \quad (2.6)$$

The preconditioning approach in (2.3) is known as left preconditioning. Similarly, the approach in (2.4) is right preconditioning and that in (2.5) is left-right preconditioning. Right preconditioning has an advantage in that the right hand side is not required to modify and also the generated residuals are identical to the true residuals. In practice, the choice of preconditioning position often depends on selected iterative method and on properties of the coefficient matrix  $A$ . In general, the preconditioned system is written as:

$$\tilde{A}x = \tilde{b} \quad (2.7)$$

The preconditioner for the iterative solver is such that it best approximates inverse of the coefficient matrix. In general, a preconditioner should have the following three good qualities (e.g. Barrett *et al.*, 1994):

1. The preconditioned system should converge rapidly, i.e., the preconditioned matrix should have good spectral properties.
2. The preconditioner should be cheap to construct and easy to invert within each iteration.
3. The preconditioner should not consume a large amount of memory. It is preferable to avoid massive indirect memory addressing operations to exploit the cache architecture in CPUs.

Generally speaking, these conditions are often conflicting for the selection of a suitable preconditioner. Because the preconditioned procedure requires a matrix-vector product at each iteration, the preconditioner must be such that this product can be performed inexpensively. On the other hand, for a rapid convergence, the preconditioner needs to be a sufficiently good approximation of the inverse of  $A$ . Thus, a well-balanced trade-off between the above requirements is to some extent problem dependent, and is the key factor the success of a preconditioned iterative solver. As we often deal with a “difficult”  $A$  due to material non-uniformity and pore-pressure consideration in geotechnical engineering, understanding well the coefficient matrix and the source of ill-conditioning allow one to control the trade-offs and yet achieve a higher performance. Thus, finding a suitable preconditioner is an active research area and several new preconditioners have recently been proposed for the efficient solution of geomechanical problems. These are categorized into the following broad groups:

### 2.2.1. Diagonal preconditioners

Diagonal preconditioners are among the cheapest and most memory effective ones. It simply scales the coefficient matrix with a diagonal matrix.

#### 2.2.1.1. Standard Jacobi (SJ) preconditioner

The standard Jacobi preconditioner is a diagonal matrix whose diagonal entries are identical to those of the coefficient matrix:

$$M_{SJ} = \text{diag}(A) \quad (2.8)$$

It is simple to construct and the cheapest preconditioner. Payer and Mang (1997) used SJ as well as SSOR (Section 2.2.2) and ILU (Section 2.2.3) preconditioned iterative solvers for the analysis of tunnel drivings. The main purpose was to demonstrate that a considerable acceleration of the solution can be achieved by using iterative solvers in comparison to direct solvers for the systems of equations arising in pure BE (boundary element) simulations and hybrid BE–FE analyses of tunnel drivings. Similarly, SJ preconditioned CGS and GMRES methods are shown to be efficient and robust than the direct Gauss elimination method for underground construction problems in mining and civil engineering using indirect boundary element method (IBEM) (Kayupov *et al.*, 1998). The faster convergence was because of the diagonal dominance of the equations with adaptive integration technique for IBEM. The eigenvalues of strongly diagonally dominant scaled matrices are well clustered around unity (Gershgorin’s theorem), which leads to fast convergent iterative procedures. Using this preconditioner, Smith and Wang (1998) successfully studied the behavior of large piled-raft foundations using 3D finite element analysis. However, Lee *et al.* (2002) observed that the performance of SJ is dependent on the type of analysis (namely, drained,

undrained, and consolidation). It is ineffective for consolidation analysis because the coefficient matrix is indefinite with significant contrast in diagonal entries corresponding to displacement and pore pressure DOFs. Jacobi preconditioning compresses the displacement-dominated eigenvalues, but pore-pressure-dominated eigenvalues are often over-scaled. Similarly, it is also ineffective for undrained problem, modelled as a nearly incompressible problem, because its ill-conditioning arises from the large stiffness ratios of the bulk modulus of water and the bulk modulus of the soil skeleton. However, the acceleration in the convergence is relatively better for the problems with drained boundary conditions. This is because the diagonal scaling transforms the matrix approximately to one of a uniform material and compresses the eigenvalue spread (Lee *et al.*, 2002). Smith and Wang (1998) had also shown that the SJ preconditioner confers good convergence characteristics on well-conditioned problems with Poisson's ratio  $\nu$  less than 0.4; however, its convergence drops down drastically as  $\nu$  approaches close to 0.5, indicating its poor performance for nearly incompressible problems. Hence, the SJ preconditioner is used for the drained problems only in this thesis.

#### **2.2.1.2. Modified Jacobi (MJ) preconditioner**

Chan *et al.* (2001) observed that the SJ scaling is actually counter productive for consolidation equations because the magnitude of the diagonal entries corresponding to flow stiffness matrix [block (2, 2) of the coefficient matrix in Equation (2.1)] are significantly smaller than the off-diagonal ones. To overcome this problem of SJ, they proposed to modify the diagonals of block

(2, 2) heuristically with a scaling factor. The Modified Jacobi preconditioner reads as:

$$M_{MJ} = \begin{bmatrix} \text{diag}(K) & 0 \\ 0 & -Q \text{diag}(C) \end{bmatrix} \quad (2.9)$$

where  $Q$  is a scaling matrix, the entries of which are computed as  $q_{jj} = \max_i |a_{ij}| / |a_{jj}| \geq 1$ ,  $a_{ij}$  is an element of coefficient matrix  $A$ . It has significantly improved the performance compared with SJ preconditioner. Using MJ for consolidation problems involving very low hydraulic permeabilities, the rate of convergence for the SQMR solver (Freund and Nachtigal, 1994b) can be accelerated by roughly one order of magnitude.

### 2.2.1.3. Generalized Jacobi (GJ) preconditioner

The motivation for the construction of GJ preconditioner came from the elegant eigenvalue clustering results given by Murphy *et al.* (2000) for a linear system with similar 2×2 block structure as that of Biot's problems, but with zero (2, 2) block. The GJ preconditioner (Phoon *et al.*, 2002) takes the following form:

$$M_{GJ} = \begin{pmatrix} \text{diag}(K) & 0 \\ 0 & \alpha \text{diag}(\hat{S}) \end{pmatrix} \quad (2.10)$$

where  $\hat{S} = C + B^T \text{diag}(K)^{-1} B$  is an approximate Schur complement matrix;  $\alpha$  is a real scalar and a negative value of  $\alpha$  is recommended for practical use. Specifically,  $\alpha = -4$  has a theoretical significance in the ideal case (Phoon *et al.*, 2002) and has been shown to be effective in the form (2.10) as well (Toh *et al.*, 2004). Thus, only this value of  $\alpha$  is considered throughout this thesis. Numerical results showed that GJ is an efficient choice for solving Biot's linear equation due to its cheap diagonal form and robustness to accelerate the

convergence of SQMR (Phoon *et al.*, 2002; Phoon *et al.*, 2003; Phoon, 2004; Toh, 2004; Toh *et al.*, 2004). The GJ preconditioner can also readily be derived and applied to solve nonsymmetric linear systems (Gambolati *et al.*, 2003; Toh and Phoon, 2008). It is also superior to MJ from both the theoretical and numerical point of view because MJ was essentially constructed from a heuristic basis and its performance outside the scope of study is less assured. The effectiveness of GJ has also been demonstrated in other areas of study such as interior-point methods (Toh, 2004). Hence, the GJ preconditioner is considered as the benchmark for the performance of other sophisticated preconditioners for the consolidation problems in this study.

### 2.2.2. SSOR preconditioner

SSOR (Symmetric Successive Over-Relaxation) iteration belongs to classical iterative methods; however, it is widely used as a preconditioner for Krylov subspace iterative methods. SSOR preconditioning can be regarded as a single iteration of SSOR iterative method with a zero initial vector. In left-right preconditioning approach, it takes the following form:

$$M_1 = \frac{1}{2-\omega} \left( L_A + \frac{D_A}{\omega} \right), M_2 = \left( \frac{D_A}{\omega} \right)^{-1} \left( U_A + \frac{D_A}{\omega} \right) \quad (2.11)$$

where  $L_A$  and  $U_A$  are strictly the lower and upper triangular parts of decomposition of  $A = L_A + D_A + U_A$  and  $\omega$  is a relaxation parameter. The scaling factor  $1/(2-\omega)$  can be neglected when using as a preconditioner for Krylov subspace iterative methods, but it may be important when the iterative method is not scale invariant (e.g. Chow and Heroux, 1998). Similar to diagonal preconditioners, the advantage of SSOR preconditioner is that it can readily be constructed from the coefficient matrix with a minor modification

of the diagonal. For this reason, it is sometimes regarded as a diagonal preconditioner (Ferronato *et al.*, 2009). As this preconditioner involves a triangular solution in each preconditioning step, it is also regarded in the family of incomplete factorization preconditioners (e.g. Saad, 1996). Based on the comparison of SJ and SSOR preconditioners for soil-structure interaction problems, Mroueh and Shahrour (1999) recommended that the left preconditioned SSOR to be used for the problems involving highly varied materials. They observed that the left SSOR with Bi-CGSTAB or QMR-CGSTAB methods can lead to an economy of 70-80% on iterations count and 50-70% on CPU-times in comparison with the results obtained with the SJ preconditioner. This could be because the SSOR factorization involves the off-diagonal terms as well, and hence, a better approximation of the coefficient matrix (Chen *et al.*, 2006). In contrast, Payer and Mang (1997) observed longer CPU times for SSOR than SJ preconditioned systems even though SSOR reduces the iteration counts considerably. It may be because of the ordering of the variables as it involves triangular solutions. Similarly, the preconditioning position (2.3-2.5) can also affect the results (Mroueh and Shahrour, 1999). Chen *et al.* (2006) recommended the left-right preconditioning (2.5) with Eisenstat trick (Eisenstat, 1981) for the efficient implementation of SSOR. It may be because Chen *et al.* studied the SSOR implementation on symmetric linear systems, while the linear system was unsymmetric for the case of Mroueh and Shahrour.

#### **2.2.2.1. MSSOR preconditioner**

Observing the breakdown of conventional SSOR preconditioner for indefinite linear systems arising from consolidation problems, a modified version of the

SSOR preconditioner (MSSOR) was proposed by Chen *et al.* (2006). The preconditioner takes the following form:

$$M_{MSSOR} = \left( L_A + \frac{\hat{D}}{\omega} \right) \left( \frac{\hat{D}}{\omega} \right)^{-1} \left( U_A + \frac{\hat{D}}{\omega} \right) = (L_A + \tilde{D}) (\tilde{D})^{-1} (L_A^T + \tilde{D}) \quad (2.12)$$

where  $\hat{D} = M_{GJ}$ . The MSSOR preconditioner is based on the standard SSOR factorization, but with the diagonal replaced by GJ. The MSSOR preconditioner implemented with Eisenstat trick (Eisenstat, 1981) leads a faster convergence than the GJ preconditioner (2.10) and block constrained preconditioner (Section 2.2.4.1) (Chen *et al.*, 2006). The MSSOR preconditioned global system is also preferable over partitioned iterative methods (Chen *et al.*, 2007).

### 2.2.3. Incomplete factorization preconditioners

We call a factorization incomplete if during the factorization process certain ‘fill’ elements, nonzero elements in the factorization in positions where the original matrix had a zero, have been ignored. Probably the earliest use of ILU (incomplete LU) preconditioning may be originated by Meijerink and van der Vorst (1977) and Kershaw (1978). A broad class of preconditioners is based on incomplete factorizations of the coefficient matrix. A preconditioner is then given in factored form:

$$M = \bar{L}\bar{U} \quad (2.13)$$

with  $\bar{L}$  the lower and  $\bar{U}$  the upper triangular factors of  $A$ . For a symmetric matrix, the corresponding preconditioner can also be formed as  $M = \bar{L}\bar{D}\bar{L}^T$ , where  $\bar{D}$  is the diagonal matrix with pivots in its diagonal and  $\bar{L}$  is the unit lower triangular. When the matrix is symmetric positive definite system, the

corresponding preconditioner is termed as Incomplete Cholesky (IC) preconditioner:

$$M = R^T R \quad (2.14)$$

where,  $R$  is the upper triangular factor.

The efficiency of the preconditioner depends on how well  $M^{-1}$  approximates  $A^{-1}$  depending on allowed fill-in elements. One possibility is to completely discard the fill-in elements in the position other than in original coefficient matrix, so that the preconditioner at worst takes exactly as much space to store as the original matrix. This is commonly known as ILU0 or IC0 factorization (e.g. Kershaw, 1978). This is the most inexpensive factorization. As the factorization is crude, it may result in the Krylov subspace solver requiring more iterations to converge for challenging problems (Saad, 1996). The second possibility is to partially neglect the fill-in elements based on some specified size criterion but, it can be more expensive than ILU0. One such factorization is ILUT( $\rho, \tau$ ), where  $\rho$  and  $\tau$  are user-specified parameters which controls the fill-in process (Saad, 1994b). The parameter  $\rho$  controls the maximum number of fill-in elements to be allowed in  $\bar{L}$  and  $\bar{U}$  factors while the parameter  $\tau$  controls the magnitude of the fill-in element (relative to the corresponding row of  $A$ ) below which the elements are dropped. If  $\rho$  and  $\tau$  are set to  $N$  (the dimension of  $A$ ) and 0, respectively, i.e. ILUT( $N, 0$ ) yields the exact LU decomposition.

For challenging problems, Payer and Mang (1997) recommended to use ILU preconditioners with partial fill-ins over the SJ and SSOR preconditioners. The performance of ILU0 was acceptable only for relatively small differences of the stiffness of the soil and of the shotcrete in tunnel

driving problems. Based on the extensive use of ILU preconditioners on coupled consolidation analysis, Gambolati and co-workers (Ferronato *et al.*, 2001; Gambolati *et al.*, 2001, 2002, 2003) pointed that an optimal ILUT can numerically be obtained which can lead to faster convergence rate and smaller CPU time than ILU0. However, they failed to recommend any particular values for  $\rho$  and  $\tau$  for the optimal performance of ILUT. This is because these user-defined parameters are very much problem dependent and can only be ascertained through numerical experiments. The matrix conditioning may strongly affect the choice of parameters and its contribution to the total computational cost. Hence, getting an optimal performance can be a difficult task. For consolidation problems, Ferronato *et al.* (2001) showed that a critical time step  $\Delta t_{\text{crit}}$  may exist depending on the hydrogeological properties of the subsurface and the mesh discretization, below which the coefficient matrix may suffer from ill-conditioning. In addition, the blind application of ILU preconditioners on indefinite problems may result in failure for many problems (Chow and Saad, 1997). These are further discussed in Chapter 3 for the Biot's problem. Gambolati *et al.* (2003) showed that the diagonal scaling of the coefficient matrix prior to ILU factorization may improve the numerical stability and accelerate the convergence of ILU preconditioned Bi-CGSTAB solver.

#### **2.2.4. Block preconditioners**

Block preconditioners are motivated from the block structure of the coefficient matrix  $A$  (2.1). For geomechanical problems, the numerical performances of three classes of block preconditioners, namely, block constrained, block diagonal and block triangular preconditioners have been studied recently.

#### 2.2.4.1. Block constrained preconditioner

Block constrained preconditioners are the recent development for the iterative solution of large-scale coupled consolidation problems. These preconditioners are originally advanced for the discrete saddle point problems encountered in optimization (Keller *et al.*, 2000). This class of preconditioners is called constrained because they have the same block structure as the native coefficient matrix, but one or more blocks are approximated or ‘constrained’. Specifically, they preserve the off-diagonal blocks and approximate the diagonal ones. The constrained preconditioner proposed by Toh *et al.* (2004) is motivated from the exact inverse of  $A$  and reads as:

$$M_c = \begin{bmatrix} \hat{K} & B \\ B^T & -C \end{bmatrix} \quad (2.15)$$

where  $\hat{K}$  is the symmetric positive definite approximation of  $K$ ,  $\hat{S}$  the symmetric positive definite approximation of Schur complement  $S$  for block  $(2, 2)$ , and is given by:

$$\hat{S} = B^T \hat{K}^{-1} B + C. \quad (2.16)$$

Under the ideal situation where  $\hat{K} = K$  and  $\hat{S} = S$ ,  $M_c^{-1}$  is exactly the inverse of  $A$  and a Krylov subspace method such as GMRES (Saad and Schultz, 1986) would converge in one iteration. Since the preconditioner is symmetric but indefinite for Biot’s problem, SQMR is the most economical solver as mentioned in the Section 2.1. However, the exact  $K$  and  $S$  in the preconditioner are impractical for large-scale problems. Hence, Toh *et al.* studied the performance of the preconditioner (2.15) over a range of approximations of  $K$  and  $S$  for a footing problem. Their numerical results suggest that the best runtimes achieved using memory efficient simple

diagonal approximation of block  $K$  and inexpensive approximations to the Schur complement  $S$  were at most twice the lowest runtime achieved using incomplete Cholesky factorization of  $K$  with about 10% of the number of nonzero elements of  $K$  in its factor. However, the main disadvantage of the latter is that it requires a non-trivial amount of memory in its storage and computation of large-scale problems because the entire  $K$  matrix needs to be stored in the memory for the factorization. The constrained preconditioner (2.15) with simple diagonal approximation of the blocks ( $\hat{K}$  and  $\hat{S}$ ) was shown to be about two times faster than the GJ (2.10), but at the cost of using more memory (Phoon *et al.*, 2004).

#### 2.2.4.2. Inexact constrained preconditioner (ICP)

Bergamaschi *et al.* (2007) argued that the conditioning number of the preconditioned Biot's problem depends on the quality of the approximation of the block corresponding to the structural matrix  $K$ , and if the block  $K$  is not diagonally dominant, the diagonal approximation (preceding Section) may prove to be a poor approximation. According to Toh *et al.* (2004), the block  $K$  in Biot's system is a diagonally significant matrix (a notion weaker than the diagonally dominant, i.e., although diagonal elements are significantly larger than the off-diagonal ones, the diagonal dominance ratio is about 0.11 or larger. The ratio should ideally be 1.0 or above for a diagonally dominant matrix in the traditional sense). Thus, Bergamaschi *et al.* proposed a new constrained preconditioner based on the sparse approximate inverse (AINV) (Benzi *et al.*, 1996; Benzi *et al.*, 2001) preconditioning of block  $K$  and the incomplete Cholesky decomposition of Schur complement  $\hat{S}$ , namely:

$$K^{-1} \approx \hat{K}^{-1} = ZZ^T \quad (2.17)$$

$$S \approx \hat{S} = L_S L_S^T \quad (2.18)$$

where,  $Z$  and  $L_S$  are incomplete upper triangular and lower triangular factors, respectively. Such a preconditioner is termed as inexact constrained preconditioner (ICP). In terms of Equation (2.1), the ICP can be expressed as:

$$M_{ICP}^{-1} = \begin{bmatrix} Z & ZZ^T B L_S^{-T} \\ 0 & L_S^{-T} \end{bmatrix} \begin{bmatrix} Z^T & 0 \\ L_S^{-1} B^T Z Z^T & -L_S^{-1} \end{bmatrix}. \quad (2.19)$$

One major drawback of the ICP preconditioner is its cost of construction and application (in terms of time and memory usage). It requires an explicit formulation the Schur complement (2.16); particularly, the computation of the first term of (2.16) can be very expensive when the linear system is very large. This could be the reason that the performance of ICP (in terms of runtime) was similar to that of ILU-type preconditioners on the studied large-scale Biot's problem (see Bergamaschi *et al.*, 2007). It only showed an advantage (speed-up to 2-times or more) over ILUT preconditioners for relatively smaller problems. However, the objective of our study is the cost-effective large-scale computation of the geotechnical problems. Similarly, another disadvantage of the ICP preconditioner is the requirement of a number of user-specified parameters (dropping tolerances for the AINV factorization and the computation of  $\hat{S}$ , and fill-in parameters for the incomplete decomposition of  $\hat{S}$ ). The optimum values of which are unknown *a priori* and may perform poorer than the traditional ILUT preconditioners if the parameters are not optimally selected (see, for example, Bergamaschi *et al.*, 2008).

#### 2.2.4.3. Mixed constrained preconditioner (MCP)

Motivated from the improvement in performance by ICP, but observing somewhat less satisfactory performance of ICP preconditioners on severely ill-

conditioned consolidation problems (due to small time integration steps), Bergamaschi *et al.* (2008) further proposed another constrained preconditioner. The construction scheme of the new preconditioner is similar to the ICP; however, it uses two different approximations of the structural block  $K$  in the same algorithm, and hence, was termed as mixed constrained preconditioner (MCP). MCP uses the incomplete Cholesky decomposition of  $K$  (2.20) with variable fill-in for preconditioning the structural block  $\hat{K}$ , while its AINV approximation (2.17) is used for the computation of Schur complement  $\hat{S}$  (2.21), before performing its incomplete Cholesky decomposition (2.18) for the preconditioner, namely:

$$K^{-1} \approx \hat{K}^{-1} = (L_K L_K)^{-1} \quad (2.20)$$

$$\hat{S} = B^T \hat{K}^{-1} B + C \approx B^T Z Z^T B + C \quad (2.21)$$

Hence, the MCP preconditioner takes the following form:

$$M_{MCP}^{-1} = \begin{bmatrix} L_K^{-T} & L_K^{-T} L_K^{-1} B L_S^{-T} \\ 0 & L_S^{-T} \end{bmatrix} \begin{bmatrix} L_K^{-1} & 0 \\ L_S^{-1} B^T L_K^{-T} L_K^{-1} & -L_S^{-1} \end{bmatrix} \quad (2.22)$$

MCP preconditioned Bi-CGSTAB was shown to be robust and superior to ICP and ILUT preconditioners (in terms of convergence and runtime) on ill-conditioned problems. The relative gain in the speed-up was up to a factor of 2 for large-scale problems in comparison to ILUT with controlled fill-in (see Bergamaschi *et al.*, 2008). Similar performance gain by MCP was also shown in the modeling of faulted rocks with large penalty terms (Ferronato *et al.*, 2008).

Though MCP is computationally more efficient than ILUT (or ILLT for symmetric positive definite  $A$ ) in terms of runtime, the implementation of MCP is generally not easy for practical application. This is because the MCP

requires a number of user-specified parameters to be set in a more or less optimal way via a trial-and-error procedure, similar to ICP. In particular, the implementation of MCP requires the following parameters to be set (Bergamaschi *et al.*, 2008; Ferronato *et al.*, 2008; Ferronato *et al.*, 2009):

1. The fill-in degree  $\rho_K$ , i.e. the number of terms stored in each row of  $L_K$  in excess to the non-zeroes of  $K$ ;
2. The fill-in degree  $\rho_S$ , i.e. the number of terms stored in each row of  $\hat{S}$  in excess to the non-zeroes of  $C$ ;
3. The AINV tolerance  $\tau_Z$ , i.e. the fraction of the  $Z$  diagonal terms below which an extra-diagonal coefficient is dropped;
4. The fill-in degree  $\rho_{SI}$ , i.e. the number of terms stored in each row of  $L_S$  in excess to the non-zeroes of  $\hat{S}$ ;

Based on the use of MCP for faulted rocks, Ferronato *et al.* (2008) concluded that the right selection of  $\tau_A$  and  $\rho_{SI}$  is crucial for the convergence. The sensitivity analysis of these parameters shows that not all parameters are equally important (Ferronato *et al.*, 2010); however, the selection of optimal combination of parameters is likely to be problem dependent. Thus, with stricter drop tolerances, the preconditioner may become better (in terms of iteration count), but at the same time it also becomes more expensive to be used in practice. In addition, the MCP requires double factorization of the block  $K$  relative to ICP. Note that the size of  $K$  is significantly large [about 90% of the size of  $A$  (Toh *et al.*, 2004)] for practical problems. Hence, the application of MCP is costlier than ILUT and ICP. These drawbacks make the implementation of MCP less attractive to practical problems. Thus, we shall not consider the MCP or ICP in our study.

#### 2.2.4.4. Block triangular preconditioner

Toh *et al.* (2004) studied the block triangular preconditioners as well for solving (2.1). Block triangular preconditioners are formed by considering either only the lower or the upper blocks, as given below:

$$M_t = \begin{bmatrix} \hat{K} & 0 \\ B^T & -\hat{S} \end{bmatrix} \text{ for left preconditioning} \quad (2.23)$$

$$M_t = \begin{bmatrix} \hat{K} & B \\ 0 & -\hat{S} \end{bmatrix} \text{ for right preconditioning} \quad (2.24)$$

In the ideal situation, where  $\hat{K} = K$  and  $\hat{S} = S$ , the preconditioned matrix has 1 as the only eigenvalues and a preconditioned GMRES (Saad and Schultz, 1986) would converge in a small number of iterations (may not converge in one iteration). This is because the triangular preconditioner destroys diagonalizability (Toh *et al.*, 2004). Their study showed that triangular preconditioners hardly offer any advantage over constrained and diagonal preconditioners in terms of runtime and memory usage on the studied Biot's problem. Similarly, Bergamaschi *et al.* (2008) recommended a triangular (T-MCP) and a diagonal (D-MCP, discussed in Section 2.2.4.5) variants of the native MCP for the practical use to lessen the drawbacks of native MCP discussed earlier. The T-MCP (2.25) was found computationally competitive to the native MCP. However, the practical drawbacks of T-MCP are no less than the native MCP, except a minor advantage in terms of computation and memory.

$$M_{T-MCP}^{-1} = \begin{bmatrix} (L_K L_K^T)^{-1} & L_K^{-T} L_K^{-1} B^T L_S^{-T} L_S^{-1} \\ 0 & -(L_S L_S^T)^{-1} \end{bmatrix} \quad (2.25)$$

One notable feature of the triangular preconditioner is that the preconditioned matrix  $M_t^{-1}A$  (or  $M_{T-MCP}^{-1}A$ ) is non-symmetric even for symmetric  $A$  (2.1).

Hence, a different Krylov subspace method such as Bi-CGSTAB (Figure 2.1) is necessary. Note that each iteration of such a solver (e.g. with  $M_t$ ) is about two-times more expensive than each SQMR or MINRES iteration (e.g. with  $M_c$ ) in terms of matrix-vector products and preconditioning steps (Toh *et al.*, 2004; Ferronato *et al.*, 2010). Thus, the gain in convergence (iteration count) is outweighed by a costlier solver application. As we only study symmetric linear systems in this study, we shall not consider the triangular preconditioner for the above reasons.

#### 2.2.4.5. Block diagonal preconditioner

The proposed block diagonal preconditioner for linear system (2.1) takes the following form (Toh *et al.*, 2004):

$$M_d = \begin{bmatrix} \hat{K} & 0 \\ 0 & \alpha \hat{S} \end{bmatrix} \quad (2.26)$$

where  $\alpha$  is a given nonzero real scalar and possibly negative,  $\hat{K}$  and  $\hat{S}$  are as defined in Section 2.2.4.1. For the ideal case, when  $\hat{K} = K$  and  $\hat{S} = S$ , the eigenvalues of the preconditioned matrix are clustered around at most 3 points, namely 1 and  $(1 \pm \sqrt{1+4/\alpha})/2$ . Choosing  $\alpha = -4$  leads to at most 2 clusters of eigenvalues at 1 and 1/2 (Phoon *et al.*, 2002). Particularly, the negative sign of  $\alpha$  is important for the convergence (Toh *et al.*, 2004). The comparison of numerical performances of block constrained, block triangular, and block diagonal preconditioners for a range of block approximations showed that the diagonal preconditioners are reasonably competitive (in terms of runtime) with the more sophisticated ones (Toh *et al.*, 2004). However, according to Bergamaschi *et al.* (2008), the diagonal variant of native MCP [D-MCP (2.27)] is too poor approximation of  $A^{-1}$  and may perform less satisfactorily for very

ill-conditioned system and preferred T-MCP (2.25). On the other hand, the ModMCP [also a diagonal variant of MCP (2.27), but with positive block (2, 2) for the symmetric positive definite linear system, and with its complete Cholesky factorization] was shown to perform comparably with the native MCP for modeling the geological faults (Ferronato *et al.*, 2008).

$$M_{D-MCP}^{-1} = \begin{bmatrix} (L_K L_K^T)^{-1} & 0 \\ 0 & -(L_S L_S^T)^{-1} \end{bmatrix} \quad (2.27)$$

The modified MCP is less expensive relative to native MCP as it only requires one symmetric incomplete factorization ( $L_K$ ) and one symmetric complete factorization ( $C^{-1}$ ). However, its performance is dependent on the user-specified parameter for  $L_K$  and the factorization can be expensive when the block  $K$  (solid stiffness matrix) is very large, which is obvious in large-scale computation.

Block diagonal preconditioners have also been successfully applied in the solution of Navier-Stokes equations (Wathen and Silvester, 1993; Silvester and Wathen, 1994), which have a similar matrix property as from Biot's consolidation equations. Block diagonal preconditioners are less expensive relative to the sophisticated constrained preconditioners as the former neglects extra-diagonal blocks of the preconditioner, and hence, a cheaper preconditioner application. Such a cost effective scheme is even more attractive for the realistic non-linear elasto-plastic problems, where the preconditioner has to be re-computed at every step, unlike elastic problems. Hence, we only focus on block diagonal preconditioners in this study as they provide a cost-effective scheme in overall. The SJ (2.8) or GJ (2.10) can be seen as the limiting form of this preconditioner.

### 2.2.5. Others

Several other preconditioners have been proposed in the literature; for example, polynomial expansion (Johnson *et al.*, 1983), explicit approximate inverse of  $A$  (Benzi *et al.*, 1996; Grote and Huckle, 1997; Huckle, 1999; Benzi *et al.*, 2001), etc. However, such preconditioners are designed for implementation on a parallel computer, and are outside the PC environment we assumed in this thesis. Hence, we shall not consider these preconditioners in this thesis. Similarly, preconditioners based on element-by-element strategy have also been proposed recently (Augarde *et al.*, 2006, 2007). The use of EBE strategy with a diagonal preconditioner to solve first order and second order time dependent partial differential equations has been demonstrated a long time ago (Smith *et al.*, 1989; Wong *et al.*, 1989). Using EBE strategies, a substantial reduction in storage requirement can be achieved as it does not require the assembly of the global coefficient matrix. However, many preconditioners in EBE strategies are based on approximate factorization techniques that perform operations at element level and then globalize the result (Hughes *et al.*, 1983; Nour-Omid and Parlett, 1985; Winget and Hughes, 1985). Thus, the use of such preconditioners invariably results in a significant increase in book-keeping workload and indirect addressing operations. As a result, EBE preconditioners often require a higher total CPU time for the solution, unless some sophisticated techniques such as element amalgamation or regrouping are implemented, to further reduce the iteration counts (cited in Chan *et al.*, 2001; Phoon *et al.*, 2002). However, by doing so, the demand on memory also increases substantially. Thus, these preconditioners could not be competitive with the global preconditioners (in terms of runtime) on scalar

computers. For this reason, this study focuses on global preconditioners only. On the other hand, EBE preconditioners are more attractive to the computations in a parallel environment.

In most of the above studies, ill-conditioning of the Biot's system due to smaller time step and low permeability has been stressed and suitable preconditioning strategies are discussed accordingly (see also Ferronato *et al.*, 2009; Gambolati *et al.*, 2010). Ill-conditioning due to significant contrasts in material properties such as stiffness and/or permeability (e.g. soil-structure interaction problems) and its effective mitigation has rarely been studied, although the occurrence of such ill-conditioning has been pointed out by some researches. Recently, ill-conditioning due to contrasts in large stiffness, but in a different context, with the use of large penalty terms for the modeling of rock faults is discussed by Ferronato *et al.* (2008). The study suggested using either MCP or modMCP to mitigate such ill-conditioning. However, these preconditioners have their own demerits for the practical use as discussed in the preceding sections.

### **2.3. Sparse storage of the matrix**

The finite element discretization of large-scale problems (3D analyses) generally leads to sparse matrices of high order in which more than ninety per cent of the entries are zeros (e.g. Papadrakakis, 1993a). If the full matrix (including those zero entries) is needed to be stored, the demand of storage will prohibitively be large. Hence, sparse storage schemes are used in this thesis to store the matrices. Since, in Fortran, multidimensional arrays are referenced in column-major order (Intel Fortran Guide), the CSC (compressed

sparse column) storage is used wherever possible. CSR (compressed sparse row) storage is also used sparingly (particularly for ILU subroutines). This is because the available codes of ILU subroutines are encoded in CSR format (Saad, 1994a). See Appendix A for more details about the storage schemes.

## 2.4. Convergence criteria

The obvious difference between iterative solvers and direct solvers is that the former provides an approximate solution with an “acceptable” approximate solution of Equation (1.1) while the latter gives an exact solution. That acceptable approximate solution depends on the adopted convergence (stopping) criterion and the prescribed tolerance. A convergence criterion is an essential component of an iterative solver, which determines when to stop the iteration process with a reasonably acceptable approximation. Stopping iterations prematurely may lead to an inaccurate solution while prolonged iterations may increase CPU runtime without a proportionate gain in accuracy. Hence, a suitable convergence criterion and tolerance is of paramount importance for a reliable solution.

A good convergence criterion stops the iteration process when the solution is identified as acceptable enough and controls the maximum iteration time. Ideally we would like to stop when the magnitudes of entries of the error  $e_k = x_k - x$  fall below a user-supplied threshold (tolerance), where  $x_k$  is the approximate solution vector of (1.1) after  $k$ -th iteration and  $x$  is the exact solution vector. The relative error norm criterion is:

$$R_E = \frac{\|e_k\|}{\|e_0\|} = \frac{\|x - x_k\|}{\|x - x_0\|} \leq stop\_tol, \quad k = 1, 2, \dots, max\_it \quad (2.28)$$

where  $x_0$  is the initial guess of the solution and 2-norm is often used, i.e.  $\|v\|_2 = \sqrt{v^T v}$  for any vector  $v$ .  $\text{max\_it}$  and  $\text{stop\_tol}$  are user specified parameters. The integer ‘ $\text{max\_it}$ ’ is the maximum number of iterations the algorithm will be permitted to perform and the real number ‘ $\text{stop\_tol}$ ’ measures how small the user wants the residual  $r_k = Ax_k - b$  (or the error  $e_k$ ) of the ultimate solution  $x_k$  to be (Barrett *et al.*, 1994). More details about  $\text{stop\_tol}$  are discussed shortly. But,  $R_E$  is hard to estimate directly because the exact solution  $x$  is not known *a priori*. Thus, the residual  $r_k$  is commonly used to measure the error, which is more readily computed. However, the above criterion (2.28) based on exact solution determined post-analysis is usually used as a ‘theoretical’ benchmark to gauge the effectiveness of other alternative practical convergence criteria (e.g. Lee *et al.*, 2002; Chen and Phoon, 2009). One commonly used convergence criterion is the relative ‘improvement’ norm (Smith and Griffiths, 1997; Smith and Wang, 1998; Smith and Griffiths, 2004), defined as:

$$R_i = \frac{\|x_k - x_{k-1}\|_\infty}{\|x_k\|_\infty} \leq \text{stop\_tol}, \quad k = 1, 2, \dots, \text{max\_it} \quad (2.29)$$

The obvious advantage of this criterion is that it only depends on the approximate solutions which are outputs of the iterative methods. However,  $R_i$  was found to have dramatic local oscillations for ill-conditioned linear systems (such as those from consolidation analyses) which may lead to premature termination of an iterative solver (Lee *et al.*, 2002). Chen and Phoon (2009) concluded that the reasons for such oscillations of  $R_i$  are the complex eigenvalues of the preconditioned matrix and cautioned the use of  $R_i$ . It is common to have complex eigenvalues of the preconditioned matrices when

the symmetric indefinite linear system (1.1) is preconditioned by GJ or MSSOR (see, for example, Chen *et al.*, 2006). A numerical experiment is presented in the subsequent Section to further validate the above statement. Another convergence criterion routinely used in numerical analyses is the relative ‘residual’ norm (Barrett *et al.*, 1994; Saad, 1996; Mroueh and Shahrour, 1999; Gambolati *et al.*, 2001; Lee *et al.*, 2002; Phoon *et al.*, 2002; van der Vorst, 2003), defined as:

$$R_r = \frac{\|r_k\|_2}{\|r_0\|_2} = \frac{\|b - Ax_k\|_2}{\|b - Ax_0\|_2} \leq stop\_tol, \quad k = 1, 2, \dots, max\_it \quad (2.30)$$

If  $x$  is the displacement vector, then  $R_r$  in  $k$ -th iteration is, in effect, a relative measure of the out-of-balance force remaining after  $k$  iterations (Lee *et al.*, 2002). The numerical studies of different convergence criteria on various geotechnical problems by Lee *et al.* (2002), and Chen and Phoon (2009) suggested that  $R_r$  is more reliable than  $R_i$ . The study by Chen and Phoon suggested that if  $R_i$  were selected as the convergence criterion, the `stop_tol` value of significantly smaller than  $10^{-6}$  should be used in practical finite element computations. However, the `stop_tol` =  $10^{-6}$  appears to be reasonable for  $R_r$ . A numerical experiment for a simple footing problem is presented in the subsequent Section. For more details, see (Lee *et al.*, 2002; Chen and Phoon, 2009). Several other criteria are also being used in other applications. For more details of these, the reader is referred to (Barrett *et al.*, 1994; Arioli, 2004). The study of all these is beyond the scope of this thesis.

The obvious disadvantage of  $R_r$  is that the convergence is strongly dependent on the initial guess  $x_0$ . However, in the absence of no good initial guess, the usual practice is to choose  $x_0 = 0$ . In this study too, zero initial

guess, i.e.  $x_0 = 0$ , is adopted. Since  $r_k = Ae_k$  or  $e_k = A^{-1}r_k$ , the criterion has the following error bound:

$$\|e_k\| \leq \|A^{-1}\| \cdot \|r_k\| \leq stop\_tol \cdot \|A^{-1}\| \cdot \|r_0\| \quad (2.31)$$

Another user specified parameter which controls the error bound is ‘stop\_tol’. The stop\_tol indicates the approximate uncertainty in the entries of  $A$  and  $b$  (1.1) relative to  $\|A\|$  and  $\|b\|$ , respectively. For example, choosing stop\_tol =  $10^{-6}$  means that the user considers the entries of  $A$  and  $b$  to have errors in the range  $\pm 10^{-6} \|A\|$  and  $\pm 10^{-6} \|b\|$ , respectively. The algorithm will compute  $x$  no more accurately than its inherent uncertainty warrants. Barrett *et al.* (1994) recommended that the user should choose stop\_tol to be less than 1 and greater than the machine precision  $\varepsilon$  (on a machine with IEEE Standard Floating Point Arithmetic,  $\varepsilon = 2^{-24} \approx 10^{-7}$  in single precision, and  $\varepsilon = 2^{-53} \approx 10^{-16}$  in double precision). The numerical results on a typical geotechnical problem in the subsequent Section suggest that the stop\_tol =  $10^{-6}$  is quite sufficient to attain the approximate solution to be close to the exact solution. Hence, a global stop\_tol =  $10^{-6}$  is adopted for this study.

A recent study on various convergence criteria (Chen and Phoon, 2009) also suggests that a decoupled or separated residual norm criteria (for systems involving two or more types of variables, e.g. displacement and pore pressure variables in coupled consolidation equations) can be an attractive alternative to the global residual norm criterion because the separated/decoupled residual vectors are shorter than the global vector. However, the evaluation of decoupled criteria will cost an additional computation, and hence, checking these criteria for an interval of  $n$ -th iteration

is recommended to be practical. For this reason, this criterion shall not be considered in this study.

#### 2.4.1. Numerical experiment

Figure 2.3 shows a typical footing problem resting on two different soil profiles. The material is assumed to be linear elastic with an effective Poisson's ratio  $\nu' = 0.3$ . Different material properties (Table 2.1) are considered to see their effect on the convergence behavior with different convergence criteria. The symmetric quadrant of the footing of 10 m cube spatial domain is discretized using 20-noded solid elements coupled with 8-noded fluid elements into  $8 \times 8 \times 8$  finite element mesh. A few points in the mesh (e.g. points:  $a_1$ ,  $a_2$ , and  $a_3$ ) are marked for the settlement comparison with different convergence criteria. The ground water is assumed to be at the ground surface and is in hydrostatic condition at the initial stage. The top surface is free in all directions and free draining with pore pressures assumed to be zero. The base of the mesh is assumed to be fixed in all directions and impermeable. The movement of the side face boundaries is constrained in the perpendicular direction, but is free in in-plane directions. A uniform pressure of 100 kPa is applied instantaneously over the first time step and the time increment is taken as  $\Delta t = 1 \text{ s}$ .

The convergence history with three different criteria for the footing problem with different material properties are shown in Figure 2.4. It can be seen from the Figure that the relative residual norm ( $R_r$ ) closely tracks the relative error norm ( $R_E$ ), while the relative improvement norm ( $R_i$ ) usually lies below the  $R_E$  norm. As shown in Table 2.2, the general accuracy is quite poor with  $\text{stop\_tol} = 1 \times 10^{-4}$  when  $R_i$  is used as the stopping criterion. Even a

tolerance of  $1 \times 10^{-6}$  is less reliable (Table 2.2). The greatest difficulties are met in stiff and low permeable porous media (see also Table 2.4). The only way to gain the accuracy with  $R_i$  is to use a more stricter tolerance, which is consistent to the finding of Chen and Phoon (2009). However, this, in turn, will increase the computational cost. Hence, we discard the use of  $R_i$  in this study.

$R_r$ , on the other hand, usually lies on above the theoretical norm  $R_E$  and tracks it more closely. As shown in Tables 2.3 and 2.4, if  $R_r$  is used as the stopping criterion, the error in the result implied by  $\text{stop\_tol} = 1 \times 10^{-4}$  is very tiny and is quite acceptable; however, their impact on the final result of a longer computation may be amplified if the system is very ill-conditioned and algorithm involves cancellations (small difference in the numbers). Hence, the  $\text{stop\_tol}$  of  $1 \times 10^{-6}$  is selected to use in this study for the following reasons:

- In general, the linear system (1.1) has to be solved repeatedly and with a generally variable time step  $\Delta t$  because the analysis of a geotechnical problem usually involves several steps. The accuracy of solution of one time step can affect the accuracy of the results of the subsequent steps, and ultimately the final solution. In such circumstances, the use of loose tolerance may reflect uncertainties in the final result after a long computation (e.g. 3D simulation).
- The actual convergence behaviour usually depends on the problem, the preconditioner and the solver used (see, for example, Lee *et al.*, 2002; Chen and Phoon, 2009). As this thesis deals with soil-structure interaction problems involving significant contrast in material properties, the resulting coefficient matrix is severely ill-conditioned,

and oscillations in the convergence (even for  $R_r$ ) of the preconditioned solvers are likely to occur. This is particularly true when the preconditioner is not a very close approximation of  $A^{-1}$ . Thus, it is more prudent to use a stricter tolerance (i.e.  $1 \times 10^{-6}$ ) to avoid any potential risk of terminating the iteration prematurely and producing inaccurate results.

- Relatively more accurate solution using a tolerance of  $1 \times 10^{-6}$  than of  $1 \times 10^{-4}$  (Table 2.4) can be achieved with a slight additional iteration counts (Table 2.5). Surprisingly, the additional cost is smaller for more ill-conditioned problems.

A use of similar or even stricter tolerance has been reported in several other literatures as summarized in Table 2.6.

## 2.5. Conclusions

The review of the literatures show that the development of a suitable preconditioner is an active area or several preconditioning strategies have been suggested recently with a particular focus on ill-conditioned Biot's consolidation equations. Some of the key findings are as follows:

- Much of the emphasis has been given to the ill-conditioning of the Biot's system due to small time steps and/or low permeability of the material.
- Preconditioners ranging from a simple diagonal to a complex constrained preconditioner requiring several user-specified parameters have been proposed.

- Although some authors have recognized the worsening condition of the system due to significant contrasts in stiffness of the materials, almost no or less attention has been paid to effectively address such an ill-conditioning with only one paper recently (Ferronato *et al.*, 2008).
- Block diagonal preconditioners have proven to be resilient and competitive (in terms of runtime) to more complex constrained preconditioners, and yet remain memory efficient with easier implementation. Hence, we will emphasis more on block diagonal preconditioners in this study.
- The relative residual norm criterion with a tolerance of  $1 \times 10^{-6}$  as the stopping criterion seems quite reasonable with sufficient accuracy in the solution using iterative methods. Hence, we adopt this as the convergence criterion in this study.

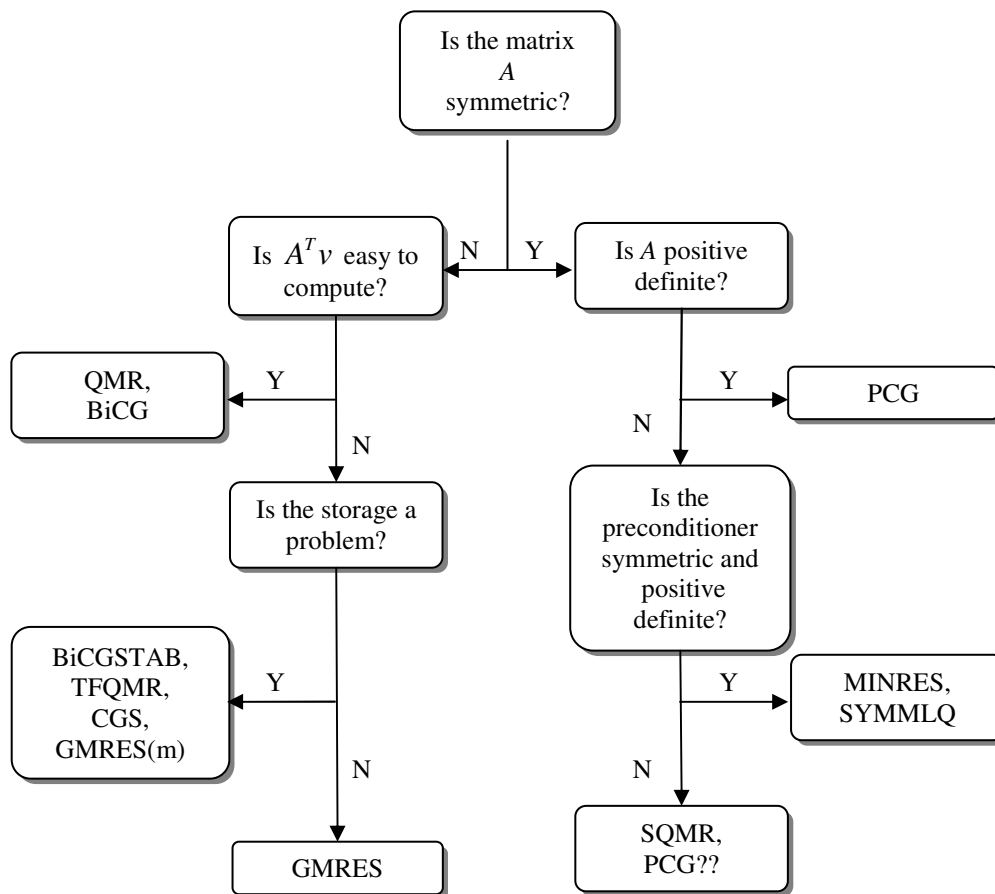


Figure 2.1. Guideline for the selection of preconditioned iterative methods.

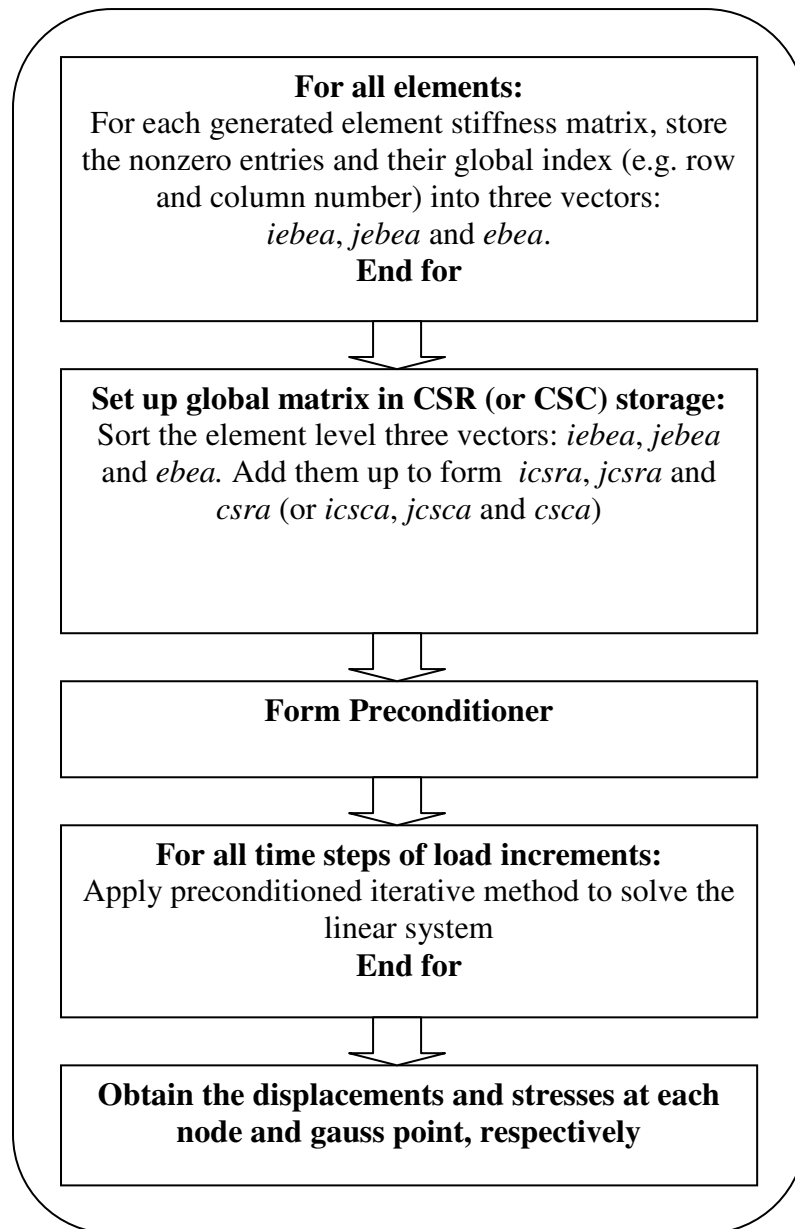


Figure 2.2. Flow chart of applying sparse preconditioned iterative method in FE analysis (after Chen, 2005).

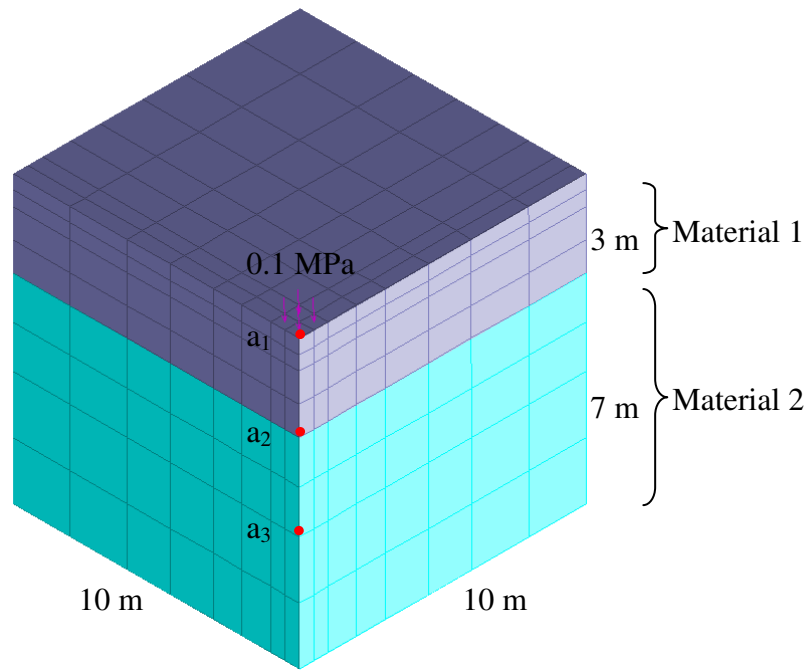


Figure 2.3. 8x8x8 mesh: A typical footing problem.

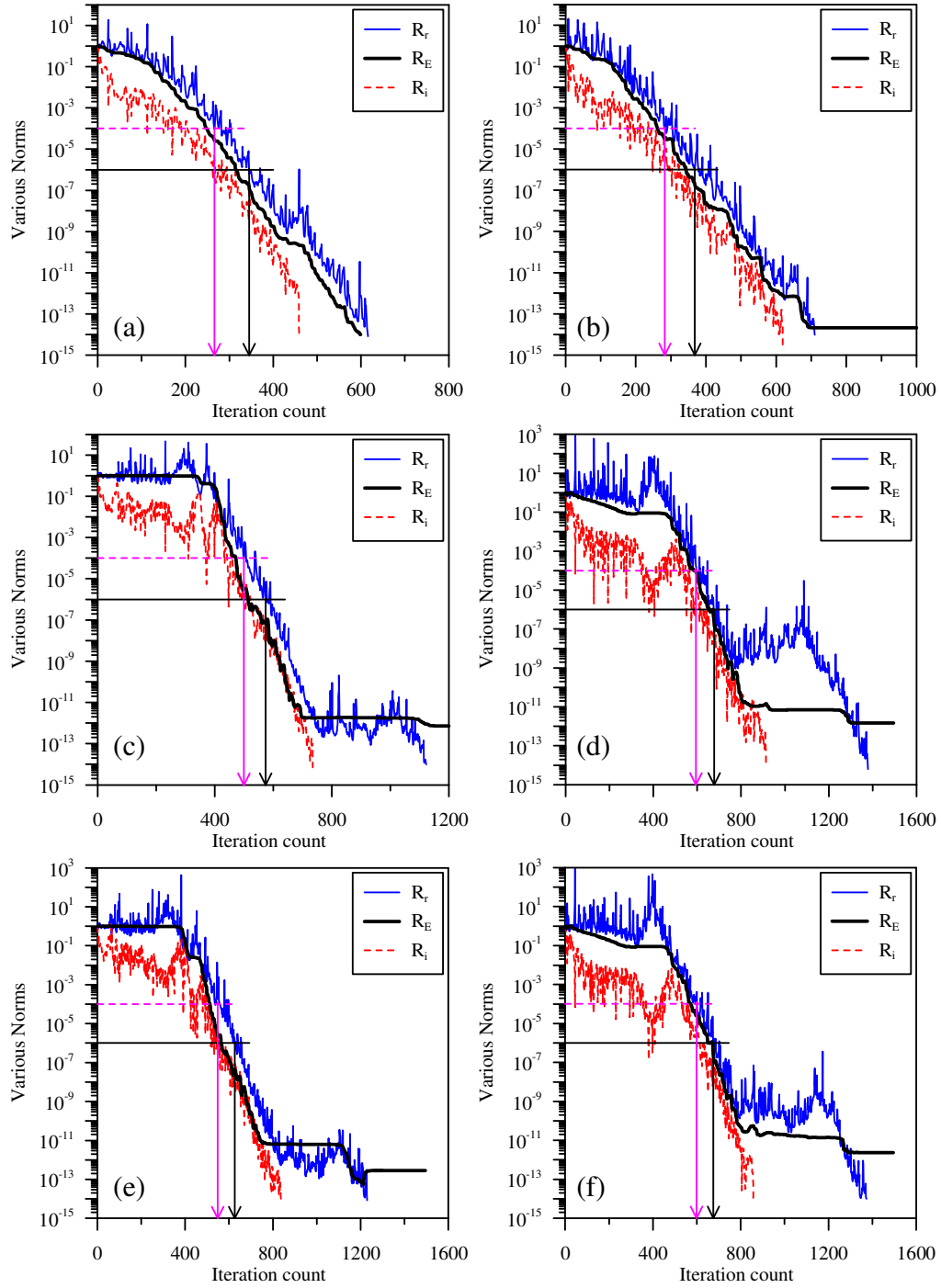


Figure 2.4. Behavior of various norms using GJ-SQMR for different material properties: (a) Conso1; (b) Conso2; (c) Conso3; (d) Conso4; (e) Conso5; and (f) Conso6.

Table 2.1. Material properties for the consolidation analysis of the footing.

Test case	$E'_1$ (MPa)	$E'_2$ (MPa)	$E'_1/E'_2$	$k_1$ (m/s)	$k_2$ (m/s)	$k_1/k_2$	$\Delta t$ (s)
Conso1	1	1	1	1.00E-03	1.00E-03	1	1.0
Conso2	1	1	1	1.00E-09	1.00E-09	1	1.0
Conso3	100000	1	100000	1.00E-03	1.00E-03	1	1.0
Conso4	100000	1	100000	1.00E-09	1.00E-09	1	1.0
Conso5	100000	1	100000	1.00E-03	1.00E-12	1.00E+09	1.0
Conso6	100000	1	100000	1.00E-09	1.00E-12	1.00E+03	1.0

Table 2.2. Computed settlements based on relative improvement norm ( $R_i$ ) criterion.

		stop_tol = $1 \times 10^{-4}$			stop_tol = $1 \times 10^{-6}$	
Test case	Points in Figure 2.3	Actual settlement <sup>#</sup> (m)	Settlement (m)	Error (%)	Settlement (m)	Error (%)
Conso1	a <sub>1</sub>	-1.5238E-01	-1.4485E-01	-4.94	-1.5238E-01	0.00
	a <sub>2</sub>	-3.5547E-02	-2.8091E-02	-20.98	-3.5547E-02	0.00
	a <sub>3</sub>	-9.4622E-03	-5.7640E-03	-39.08	-9.4615E-03	-0.01
Conso2	a <sub>1</sub>	-1.3934E-01	-1.2836E-01	-7.88	-1.3934E-01	0.00
	a <sub>2</sub>	-3.4532E-02	-2.3061E-02	-33.22	-3.4526E-02	-0.02
	a <sub>3</sub>	-9.1328E-03	-3.5013E-03	-61.66	-9.1300E-03	-0.03
Conso3	a <sub>1</sub>	-3.5139E-05	-2.4155E-05	-31.26	-3.5138E-05	0.00
	a <sub>2</sub>	-3.3922E-05	-2.2938E-05	-32.38	-3.3921E-05	0.00
	a <sub>3</sub>	-1.7827E-06	-1.3193E-06	-25.99	-1.7926E-06	0.56
Conso4	a <sub>1</sub>	-3.4871E-06	-1.0482E-06	-69.94	-4.9583E-06	42.19
	a <sub>2</sub>	-2.5529E-06	-2.2377E-07	-91.23	-4.0325E-06	57.96
	a <sub>3</sub>	-9.6137E-07	-2.0919E-08	-97.82	-1.7415E-06	81.15
Conso5	a <sub>1</sub>	-3.4455E-05	-4.5300E-06	-86.85	-3.3882E-05	-1.66
	a <sub>2</sub>	-3.3238E-05	-3.3383E-06	-89.96	-3.2664E-05	-1.73
	a <sub>3</sub>	-3.6133E-06	-4.6812E-07	-87.04	-3.6313E-06	0.50
Conso6	a <sub>1</sub>	-3.4871E-06	-1.0482E-06	-69.94	-4.9659E-06	42.41
	a <sub>2</sub>	-2.5529E-06	-2.2377E-07	-91.23	-4.0403E-06	58.26
	a <sub>3</sub>	-9.6137E-07	-2.0919E-08	-97.82	-1.7424E-06	81.24

<sup>#</sup> Actual settlement is based on the Frontal solver (GeoFEA, 2006).

Table 2.3. Computed settlements based on on relative residual norm ( $R_r$ ) criterion.

Test case	Position in Figure 2.3	Actual settlement <sup>#</sup> (m)	stop_tol = $1 \times 10^{-4}$		stop_tol = $1 \times 10^{-6}$	
			Settlement (m)	Error (%)	Settlement (m)	Error (%)
Conso1	a <sub>1</sub>	-1.5238E-01	-1.5238E-01	0.00	-1.5238E-01	0.00
	a <sub>2</sub>	-3.5547E-02	-3.5547E-02	0.00	-3.5547E-02	0.00
	a <sub>3</sub>	-9.4622E-03	-9.4620E-03	0.00	-9.4622E-03	0.00
Conso2	a <sub>1</sub>	-1.3934E-01	-1.3934E-01	0.00	-1.3934E-01	0.00
	a <sub>2</sub>	-3.4532E-02	-3.4531E-02	0.00	-3.4532E-02	0.00
	a <sub>3</sub>	-9.1328E-03	-9.1328E-03	0.00	-9.1329E-03	0.00
Conso3	a <sub>1</sub>	-3.5139E-05	-3.5139E-05	0.00	-3.5139E-05	0.00
	a <sub>2</sub>	-3.3922E-05	-3.3921E-05	0.00	-3.3922E-05	0.00
	a <sub>3</sub>	-1.7827E-06	-1.7812E-06	-0.08	-1.7828E-06	0.01
Conso4	a <sub>1</sub>	-3.4871E-06	-3.4871E-06	0.00	-3.4871E-06	0.00
	a <sub>2</sub>	-2.5529E-06	-2.5529E-06	0.00	-2.5529E-06	0.00
	a <sub>3</sub>	-9.6137E-07	-9.6129E-07	-0.01	-9.6137E-07	0.00
Conso5	a <sub>1</sub>	-3.4455E-05	-3.4455E-05	0.00	-3.4455E-05	0.00
	a <sub>2</sub>	-3.3238E-05	-3.3238E-05	0.00	-3.3238E-05	0.00
	a <sub>3</sub>	-3.6133E-06	-3.6133E-06	0.00	-3.6133E-06	0.00
Conso6	a <sub>1</sub>	-3.4871E-06	-3.4871E-06	0.00	-3.4871E-06	0.00
	a <sub>2</sub>	-2.5529E-06	-2.5529E-06	0.00	-2.5529E-06	0.00
	a <sub>3</sub>	-9.6137E-07	-9.6107E-07	-0.03	-9.6138E-07	0.00

<sup>#</sup> Actual settlement is based on the Frontal solver (GeoFEA, 2006).

Table 2.4. Error norm of the solution.

Test case	Relative improvement norm ( $R_i$ )		Relative residual norm ( $R_r$ )	
	stop_tol = $1 \times 10^{-4}$	stop_tol = $1 \times 10^{-6}$	stop_tol = $1 \times 10^{-4}$	stop_tol = $1 \times 10^{-6}$
Conso1	1.60E-01	4.20E-05	2.75E-05	1.25E-07
Conso2	2.39E-01	2.83E-04	3.56E-05	2.21E-07
Conso3	8.51E-03	8.49E-07	5.32E-08	1.84E-09
Conso4	1.99E-01	2.44E-02	9.25E-06	7.76E-08
Conso5	2.05E-02	6.06E-04	2.41E-08	7.05E-10
Conso6	1.99E-01	2.43E-02	1.72E-05	8.81E-08

Table 2.5. Iteration counts.

Test case	Relative improvement norm ( $R_i$ )			Relative residual norm ( $R_r$ )		
	stop_tol $= 1 \times 10^{-4}$	stop_tol $= 1 \times 10^{-6}$	Increase in	stop_tol $= 1 \times 10^{-4}$	stop_tol $= 1 \times 10^{-6}$	Increase in
			iteration count (%)			iteration count (%)
Conso1	113	263	132.74	266	345	29.70
Conso2	90	250	177.78	283	368	30.04
Conso3	370	478	29.19	500	574	14.80
Conso4	45	397	782.22	595	678	13.95
Conso5	232	437	88.36	548	626	14.23
Conso6	45	397	782.22	599	676	12.85

Table 2.6. Tolerance values used in the literatures.

S.N.	Author(s)	Convergence Criterion	Tolerance value	Problem description
1	Chan <i>et al.</i> (2001)	$R_r$	$1 \times 10^{-3}$	Linear system arising from Biot's consolidation equations
2	Bergamaschi <i>et al.</i> (2007); Bergamaschi <i>et al.</i> (2008)	$R_E$	$1 \times 10^{-5}$ , $1 \times 10^{-5}$ - $1 \times 10^{-8}$	Linear system arising from Biot's consolidation equations
3	Mroueh and Shahrour (1999)	$R_r$	$1 \times 10^{-6}$	soil-structure interaction problems
4	Kayupov <i>et al.</i> (1998)	$R_r$	$1 \times 10^{-6}$	analysis of mining and underground constructions
5	Lee <i>et al.</i> (2002)	$R_b, R_r, R_E$	$1 \times 10^{-6}$	Linear system from drained, undrained, and consolidation analyses
6	Phoon <i>et al.</i> (2002) Phoon <i>et al.</i> (2004); Toh <i>et al.</i> (2004); Chen <i>et al.</i> (2006); Chen <i>et al.</i> (2007); Toh and Phoon (2008)	$R_r$	$1 \times 10^{-6}$	Linear system arising from Biot's consolidation equations
7	Augarde <i>et al.</i> (2006); Augarde <i>et al.</i> (2007)	$R_r$	$1 \times 10^{-6}$	elastic and elasto-plastic problems
8	Chen and Phoon (2009)	$R_b, R_r, R_E$	$1 \times 10^{-6}$	Linear system from drained, undrained, and consolidation analyses
9	Ferronato <i>et al.</i> (2009)	$R_E$	$1 \times 10^{-8}$	Linear system arising from Biot's consolidation equations
10	Payer and Mang (1997)	$R_r$	$1 \times 10^{-10}$	analysis of tunnel excavation
11	Ferronato <i>et al.</i> (2008)	$R_r$	$1 \times 10^{-10}$	Faulted rocks with penalty approach
12	Gambolati <i>et al.</i> (2002); Gambolati <i>et al.</i> (2010)	$R_r$	$1 \times 10^{-10}$	Linear system arising from Biot's consolidation equations
13	Ferronato <i>et al.</i> (2001)	$R_r$	$1 \times 10^{-12}$	Linear system arising from Biot's consolidation equations
14	Phoon <i>et al.</i> (2003)	$R_r$	$1 \times 10^{-14}$	soil-structure interaction problems
15	Gambolati <i>et al.</i> (2001)	$R_r$	$1 \times 10^{-15}$	Linear system arising from Biot's consolidation equations
16	Gambolati <i>et al.</i> (2003)	$R_r$	Machine precision	Linear system arising from Biot's consolidation equations

## Chapter 3

---

# PERFORMANCE OF ILU0 VERSUS MSSOR FOR BIOT'S CONSOLIDATION EQUATIONS

### 3.1. Introduction

As discussed in Chapter 2, several preconditioners have been proposed for the efficient solution of Biot's consolidation equations. Some have focused on cheaper diagonal preconditioners while others on memory intensive incomplete factorization preconditioners. For example, Phoon *et al.* (2002) derived the generalized Jacobi (GJ) preconditioner (2.10) based on Murphy *et al.*'s (2000) theoretical derivation of block diagonal preconditioner. However, observing GJ's less satisfactory performance on heterogeneous soil profiles, Chen *et al.* (2006) proposed a modified SSOR (MSSOR) preconditioner (2.12). The latter was shown to have a better performance (in terms of iteration count and runtime) than the GJ with no extra memory demand.

Alternatively, Gambolati and co-workers (e.g. Ferronato *et al.*, 2001; Gambolati *et al.*, 2001, 2002, 2003) have focused on incomplete factorization preconditioners with Bi-CGSTAB solver (van der Vorst, 1992). Bi-CGSTAB was selected because of the nonsymmetric  $2 \times 2$  block indefinite matrix derived from finite element discretization of Biot's consolidation equations. Thus, it

would be worth comparing the performance of MSSOR preconditioner to that of incomplete LU factorization with zero fill-ins (ILU0). This is because the MSSOR preconditioner does not include any extra term other than the native coefficient matrix  $A$ . Similarly, ILU0 prescribes the triangular factors  $\bar{L}$  and  $\bar{U}$  that has the same sparsity pattern as the coefficient matrix  $A$ . In addition, both involve backward and forward triangular solutions in their application. Note that the ILU0 preconditioner is symmetric for the symmetric Biot's system (2.1).

As discussed in Section 2.1, symmetric Biot's system is preferred over nonsymmetric ones. Hence, this study compares the performance of MSSOR and ILU0 preconditioners in conjunction with SQMR in relation to a number of factors: (i) problem size ( $N \approx 55,000$ -300,000 degrees of freedom); (ii) nodal ordering; and (iii) soil conditions with the aid of a simple footing problem. The performance of GJ is also included for the completeness. Finally, these preconditioners are also compared for a pile group foundation problem for their effectiveness in soil-structure interaction problems.

## 3.2. Numerical Results

A simple footing problem resting on three different soil profiles is considered:

Soil 1: homogeneous soft clay with  $E'_{clay} = 1$  MPa and coefficient of permeability,  $k_{clay} = 10^{-9}$  m/s,

Soil 2: homogeneous dense sand with  $E'_{sand} = 100$  MPa and coefficient of permeability,  $k_{sand} = 10^{-5}$  m/s,

Soil 3: heterogeneous soil consisting of alternate dense sand and soft clay with

$$E'_{sand} = 100 \text{ MPa}, k_{sand} = 10^{-5} \text{ m/s and } E'_{clay} = 1 \text{ MPa}, k_{clay} = 10^{-9} \text{ m/s.}$$

The material is assumed to be linear elastic with an effective Poisson's ratio,  $\nu' = 0.3$ . The symmetric quadrant of the footing of 10 m cube spatial domain is discretized using 20-noded solid elements coupled with 8-noded fluid elements as shown in Figure 3.1. The studied finite element mesh ranges from  $16 \times 16 \times 16$  producing 55,280 DOFs to  $28 \times 28 \times 28$  producing 291,620 DOFs. Figure 3.2 shows a  $20 \times 20 \times 20$  finite element mesh used for homogeneous and heterogeneous soils. More details of finite element meshes are presented in Table 3.1.

The ground water table is assumed to be at the ground surface and is in hydrostatic condition at the initial stage. The base of the mesh is assumed to be fixed in all directions and impermeable. Side face boundaries are constrained to move only in in-plane directions. The top surface is free in all directions with zero excess pore pressures. A uniform pressure of 100 kPa is applied instantaneously over the first time step and the time increment is taken as  $\Delta t = 1 \text{ s}$ .

Following indicators are used for the comparison of preconditioners:

- 1) Number of iterations required to achieve a residual norm below  $10^{-6}$ .

The maximum iteration is limited to 5,000.

- 2) CPU time required for the solution of the problem using SQMR solver
- 3) RAM usage for the execution of the problem.

The ILU0 and GJ preconditioners are applied in right preconditioning. Right preconditioning is chosen due to the fact that it does not modify the right hand side vector. The MSSOR preconditioner is applied in left-right

preconditioning to exploit the Eisenstat trick (Eisenstat, 1981). For ILU factorization, the storage of full  $A$  in compressed sparse row (CSR) format (e.g. Saad, 1996) is required, while the implementation of GJ and MSSOR preconditioners requires the storage of upper half of symmetric  $A$  only in compressed sparse column (CSC) format (e.g. Chen *et al.*, 2006).

### 3.2.1. Effect of nodal ordering

The convergence of iterative solver preconditioned with incomplete factorization preconditioner is related to the nodal ordering of the unknowns (e.g. Gambolati *et al.*, 2001). This is so because dropping of fill-ins in incomplete factorization is sensitive to the sparsity structure of  $A$  which depends very much on the nodal ordering of unknowns.

In this study, the effect of three different nodal orderings is studied. In nodal ordering 1, all the unknown (displacements and pore pressure) variables at each node are ordered in sequence. We termed this ordering as ‘natural ordering’. The nodal ordering 2 arranges the variables so that displacement unknowns precede pore pressure unknowns. In this case, the coefficient matrix takes the  $2 \times 2$  block form (2.1). Ordering 3 is obtained by applying the reverse Cuthill-McKee or RCM technique (e.g. George and Lui, 1981) on ordering 1. In summary, the studied nodal orderings are:

- 1) Natural ordering: Nat =

$$[x_1, y_1, z_1, p_1, x_2, y_2, z_2, p_2, \dots, x_N, y_N, z_N, p_N]$$

- 2) Block ordering: Blk =

$$[x_1, y_1, z_1, x_2, y_2, z_2, \dots, x_N, y_N, z_N, p_1, p_2, \dots, p_N]$$

- 3) Reverse Cuthill-McKee algorithm applied on natural ordering: R-Nat

where  $x_i$ ,  $y_i$ ,  $z_i$  are the displacement unknowns in x-, y-, z-directions, respectively,  $p_i$  is the excess pore pressure unknown at node  $i$ , and  $N$  is the total dimension of  $A$ . A plot of the sparsity pattern of  $A$  for the above orderings is shown in Figure 3.3. The ordering where all displacements in x-direction (e.g.  $x_1, x_2, \dots$ ) come first followed by displacements in y-direction, z-direction, and then pore pressures is not addressed in the present study since it appeared to be unattractive in the past studies (e.g. Gambolati *et al.*, 2001; Chen *et al.*, 2006). Similarly, no appreciable difference between application of RCM algorithm on nodal ordering 1 or 2 was observed; hence, the latter is not included. The RCM algorithm usually compresses the matrix bandwidth. However, its application cannot make the bandwidth smaller than ordering 1 in this problem (see Figure 3.3) but, it does help in improving the quality of incomplete factorizations (see results latter). Fortran90 subroutines for reverse Cuthill McKee algorithm are obtained from (Burkardt, 2003). The resulting permutation matrix (or vector) is used to permute  $A$  and the right hand side vector via the subroutines from SPARSKIT (Saad, 1994a), which is originally used for ILU0 factorization. The original ILU0 code is modified to produce  $\bar{L}\bar{D}\bar{L}^T$  for symmetric  $A$ , where  $\bar{L}$  is unit lower triangle and  $\bar{D}$  is the diagonal consisting of pivots. The solution process with reordering of variables can generally be divided into three steps (George and Lui, 1981):

- (a) Re-ordering: Permute symmetrically the rows and columns of matrix  $A$  using the permutation matrix generated from RCM algorithm. Suppose the permutation matrix is  $\bar{P}$ , then the re-ordered matrix is  $\bar{P}A\bar{P}^T$ .

(b) Numerical factorization: Perform the incomplete LU factorization

$$\text{so that } \bar{P}\bar{A}\bar{P}^T = \bar{L}\bar{D}\bar{L}^T.$$

(c) Triangular solution: Solve  $\bar{L}\bar{D}\bar{L}^T \bar{P}x = \bar{P}b$  for  $\bar{P}x$  by solving two

triangular linear systems. Then recover  $x$  from  $\bar{P}x$ .

Numerical results are summarized in Tables 3.2-3.4. Note that ILU0 preconditioned SQMR failed to converge in naturally ordered linear system for the soil profiles 1 and 3 (see Table 3.2). Figure 3.4 shows a typical pattern of relative residual norms when ILU0 is unstable. However, the same problem converges when the variables are ordered in the block form. The convergence is further improved (about 2 to 8 times) by reordering the variables with RCM algorithm prior to factorization (if it converges). Surprisingly, RCM ordering can be counter-productive. For example, for the 20×20×20 mesh containing soils 1 and 3, the block ordering system can converge but the RCM system cannot. Physically, the lower permeability in soils 1 and 3 will usually involve smaller entries in the flow equations. The accuracy of floating point arithmetic may be a factor in these problems. This issue is studied in the next Section.

By distinction, the MSSOR and GJ-preconditioned systems (Tables 3.3 and 3.4) are less sensitive to the ordering in comparison to ILU0 preconditioned system. These systems are usually superior when the variables are ordered in the natural ordering, for all soil conditions, consistent to the findings of Chen *et al.* (2006). The 2×2 block and the RCM ordered forms are about 10-20% more expensive than the natural ordering. For ILU0 system to be competitive, this RCM step is necessary. There is an additional cost associated with RCM, which is given in parenthesis in Table 3.2. The practical observation here is that the GJ and MSSOR-preconditioned systems

are robust. These preconditioners do not exhibit convergence problems over the range of problems studied and they are not much sensitive to the ordering scheme. Hence, it can be concluded that although RCM ordering leads ILU0 to converge faster than the GJ and MSSOR preconditioned systems, ILU0 preconditioner may fail to converge in some cases. At present, no clear guidelines are available to advise the user on when such breakdowns will occur.

### 3.2.2. Problems with ILU factorization and their stabilization

Chow and Saad (1997) have successfully applied ILU preconditioners to many indefinite matrices. However, according to their numerical experience, the failure rate of ILU preconditioners is still too high and too unpredictable for them to be useful as black-box library software for general matrices. The common problems which can cause failure of ILU preconditioners are (e.g. Barrett *et al.*, 1994; Saad, 1996; Chow and Saad, 1997):

- 1) Inaccuracy due to very small pivots or zero pivots,
- 2) Unstable triangular solves, and
- 3) Inaccuracy due to dropping of fill-ins.

These problems may occur together or one problem may mask another. Chow and Saad (1997) proposed three statistics to be monitored during factorization or after the factorization has been computed in order to understand what can happen in an incomplete factorization (see Table 3.5). The first statistic ‘condest’ measures the stability of triangular solves. This statistic is also a lower bound for  $\|(\bar{L}\bar{U})^{-1}\|_{\infty}$  and indicates a relation between unstable triangular solves and poorly conditioned  $\bar{L}$  and  $\bar{U}$  factors. We refer to this statistic as the condition estimate of  $(\bar{L}\bar{U})^{-1}$ . The second statistic ‘1/smallest

pivot' is needed to help interpret this condition estimate. The condition estimate will certainly be poor if there are very small pivots. If both, first and second, quantities are about the same size, then we assume that  $\|(\bar{L} \bar{U})^{-1}\|_{\infty}$  is large due to at least one very small pivot. If 'condest' is much larger than  $1/\text{pivot}$  then we assume that recurrences associated with the triangular solves are unstable. The third statistic is the size of the largest element in the  $\bar{L}$  and  $\bar{U}$  factors. A large value of this statistic in relation to the size of the elements in  $A$  indicates an unstable and thus inaccurate factorization.

According to Chow and Saad, these statistics are usually meaningful when their values are very large, e.g. on the order of  $10^{15}$ . Extremely large values, particularly of the condition estimate, can be used to predict when the ILU0 preconditioner will fail. When all three statistics are reasonably small and the ILU preconditioner does not help an iterative method converge, their experience suggests that the cause of failure is inaccuracy due to dropping of fill-ins. Figure 3.5 describes how to interpret the abovementioned statistics. Note that there are no cases of small condest and large  $1/\text{pivot}$ .

The ILU statistics and possible cause of failure of the ILU0 preconditioned system for soil 1 (clay) are presented in Table 3.6. As shown in the Table, a very large 'condest' value ( $>10^{30}$ ) for naturally ordered system indicates that the factorization is useless. The ILU0 preconditioned system can also fail when one of the smallest pivot is in the order of  $10^{-7}$  (e.g. RCM ordered system in  $20 \times 20 \times 20$  mesh). For the latter case, ILU factorization with partial fill-ins may improve the results (e.g. Gambolati *et al.*, 2001). The values of ILU statistics for sand (soil 2) are found to be small (Table 3.7) because no failure was observed for this soil condition. It could be because the

system is well conditioned for materials involving high permeability (Ferronato *et al.*, 2001). However, the similar ILU statistics for heterogeneous soil 3 and for homogeneous soil 1 (compare Tables 3.8 and 3.6) could be because of the presence of same low permeable clay on both cases.

Several ways have been proposed in the literatures to stabilize the ILU factorization, including diagonal perturbation of the matrix (e.g. Kershaw, 1978; Manteuffel, 1980) before or during the factorization when a small or negative pivot is encountered, pivoting, and reordering of the variables. The effect of reordering the variables using RCM algorithm has already been presented in Table 3.2. A preliminary left and right scaling of the matrix can also be implemented in order to improve the stability of factorization (Gambolati *et al.*, 2003). Another way is to simply replace the smaller pivots by larger values (e.g. Chow and Saad, 1997).

This study follows the stabilization technique suggested by Chow and Saad (1997), in which the stabilization is carried out by replacing the pivots whose absolute values are smaller than a parameter ‘threshold’ by itself with the original sign of the pivot during factorization. Parametric studies of the ‘threshold’ values (Figure 3.6) shows a reduction of the condense values (one of the prominent ILU statistics) with appropriate threshold values. However, the number of iteration counts can be significantly large beyond a limited range of threshold values as shown in Figure 3.6. A reasonably good choice of threshold value ranges from 0.008 – 0.07 for both soils 1 and 3. It is possible that the ranges for soils 1 and 3 are similar because the same low permeability clay ( $k_{clay}$ ) appears in both cases. Also, the number of iteration counts significantly differs for the same range of values of ILU statistics; for

example, the iteration counts and ILU statistics for block ordered and RCM ordered systems in Tables 3.6-3.8. Hence, although ILU statistics are useful to diagnose the failure, they may not be sufficient to guide users for optimal performance of ILU0. It is postulated that the determination of proper threshold value (stabilization parameter) is largely problem dependent (mesh and soil type). For this reason, Toh *et al.* (2004) noted that “*this optimal balance can only be identified through numerical experiments – a luxury that practitioners can ill-afford and completely self-defeating if the goal is to solve a problem in the shortest time*”.

### **3.2.3. MSSOR versus ILU0 and GJ preconditioners**

The performance of each preconditioning in its most favorable form is compared. With this, the ILU0 preconditioning includes the stabilized ILU0 with threshold value 0.009, 0.0009, and 0.02 for soils 1, 2, and 3, respectively. RCM ordering prior to factorization is adopted. The GJ and MSSOR preconditioners are applied in naturally ordered form.

Figure 3.7 shows the iteration count and the total CPU time required by each preconditioning. For comparison purpose, the results are scaled with respect to MSSOR. From the practical yardstick of minimizing total runtime, stabilized ILU0 is only slightly effective than MSSOR when the soil is homogeneous (soils 1 and 2). However, it can be up to 2 times faster than MSSOR for heterogeneous soils (e.g. soil 3). The faster convergence of stabilized ILU0 is mainly because the number of iteration counts for ILU0 can be 50-70% smaller than that for MSSOR. This faster rate can be explained by the eigenvalue distribution of preconditioned matrices, as shown in Figure 3.8.

The Figure also shows why GJ performs poorly in comparison to MSSOR or stabilized ILU0.

In terms of cost of construction of preconditioner, ILU0 is obviously more expensive than GJ or MSSOR preconditioner which is simply a diagonal construction. The ILU0 preconditioner demands more memory due to full storage of  $A$ , secondary storage requirement for RCM ordering as well as for the triangular factors. Our numerical experiments show that the RAM required by ILU0 is about 75% more than that of GJ or MSSOR (see Figure 3.9).

#### **3.2.4. Performance of preconditioners on a pile group problem**

This Section compares the performance of above preconditioners for a pile group problem. For this, consolidation analysis of a 9-pile group located in homogeneous soft clay is considered. The same threshold value (0.009) is used for stabilized ILU0 preconditioner because the same soft clay (soil 1; that was considered for the shallow footing problem in the preceding Sections) is used for this example as well. Figure 3.10 shows the  $12 \times 12 \times 12$  finite element mesh discretization used for a symmetric quadrant of the problem. The mesh involves 1,728 elements and 23,604 unknowns (21,576 displacement DOFs, and 2,028 pore pressure DOFs). The pile element is also assumed to be consolidation element with Young's modulus  $E'_p = 3,000$  to  $30,000$  MPa, coefficient of hydraulic permeability  $k_p = 10^{-17}$  m/s (almost impermeable), and Poisson's ratio  $\nu' = 0.15$ . A uniform pressure of 100 kPa is applied to the  $7 \times 7$  m<sup>2</sup> pile-group area. Other boundary conditions are similar to the one explained in Section 3.2.

As shown in Figure 3.11, the stabilized ILU0 converges in much fewer iteration counts than the GJ or MSSOR preconditioners. Because of this, the stabilized ILU0 may save up to 30% CPU time in comparison to MSSOR preconditioner, while GJ can be slower by more than 2 times in comparison to MSSOR. However, the performance of each preconditioner degrades with increasing pile-soil stiffness contrasts. This is similar to the findings of Chen *et al.* (2006). Hence, it can be concluded that the application of ILU0 may be preferred over MSSOR under the following conditions: (a) instability problem of ILU can be resolved effectively, (b) optimum threshold value is known *a priori* from the solution of similar problems, and (c) RAM constraint is not an issue.

The ground surface settlement profile for the above loading is as shown in Figure 3.12. The maximum settlement occurred at the centre of pile group is about 0.1 m after first time step of load application with pile stiffness of  $E'_p = 30,000$  MPa .

### 3.3. Conclusions

The numerical performance of ILU0 and MSSOR was compared for the FE solution of coupled consolidation problems over a range of soil conditions, nodal ordering of variables, and for a soil-structure interaction problem. It is observed that a straightforward application of incomplete factorization preconditioner may lead to convergence failure. In contrast, MSSOR is robust enough to converge over the range of problems studied, albeit, with a higher iteration count than ILU0. Ordering of variables has a significant impact on ILU0 and reordering the variables by Reverse Cuthill-McKee (RCM)

algorithm prior to factorization helps to reduce convergence failures but cannot eliminate such failures completely. Various statistics proposed by Chow and Saad (1997) were found to be helpful in diagnosing the failure of ILU0 factorizations. However, no perfect guidelines exist. It was demonstrated that the factorization could be stabilized by perturbation of the pivots. However, the determination of a proper threshold value (stabilization parameter) is largely problem dependent (mesh and soil type). This optimal balance can only be identified through numerical experiments – a luxury that practitioners can ill-afford and completely self-defeating if the goal is to solve a problem in a shortest time. On the other hand, the stabilized ILU0 preconditioned system (if it converges) is up to two times faster than the MSSOR preconditioned system, particularly in heterogeneous problems. Thus, it can be concluded that the application of ILU0 may be preferred over MSSOR under the following conditions: (a) instability problem of ILU can be resolved effectively, (b) optimum threshold value is known *a priori* from the solution of similar problems, and (c) RAM constraint is not an issue.

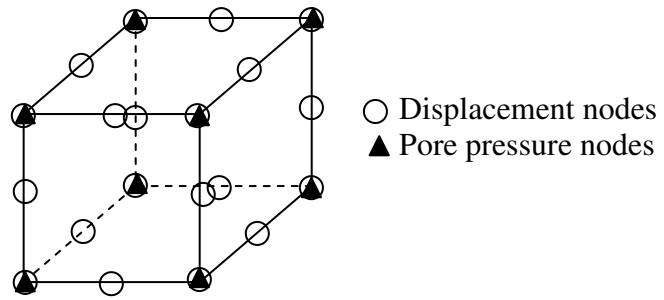


Figure 3.1. Twenty-noded displacement finite element coupled with eight-noded fluid elements.

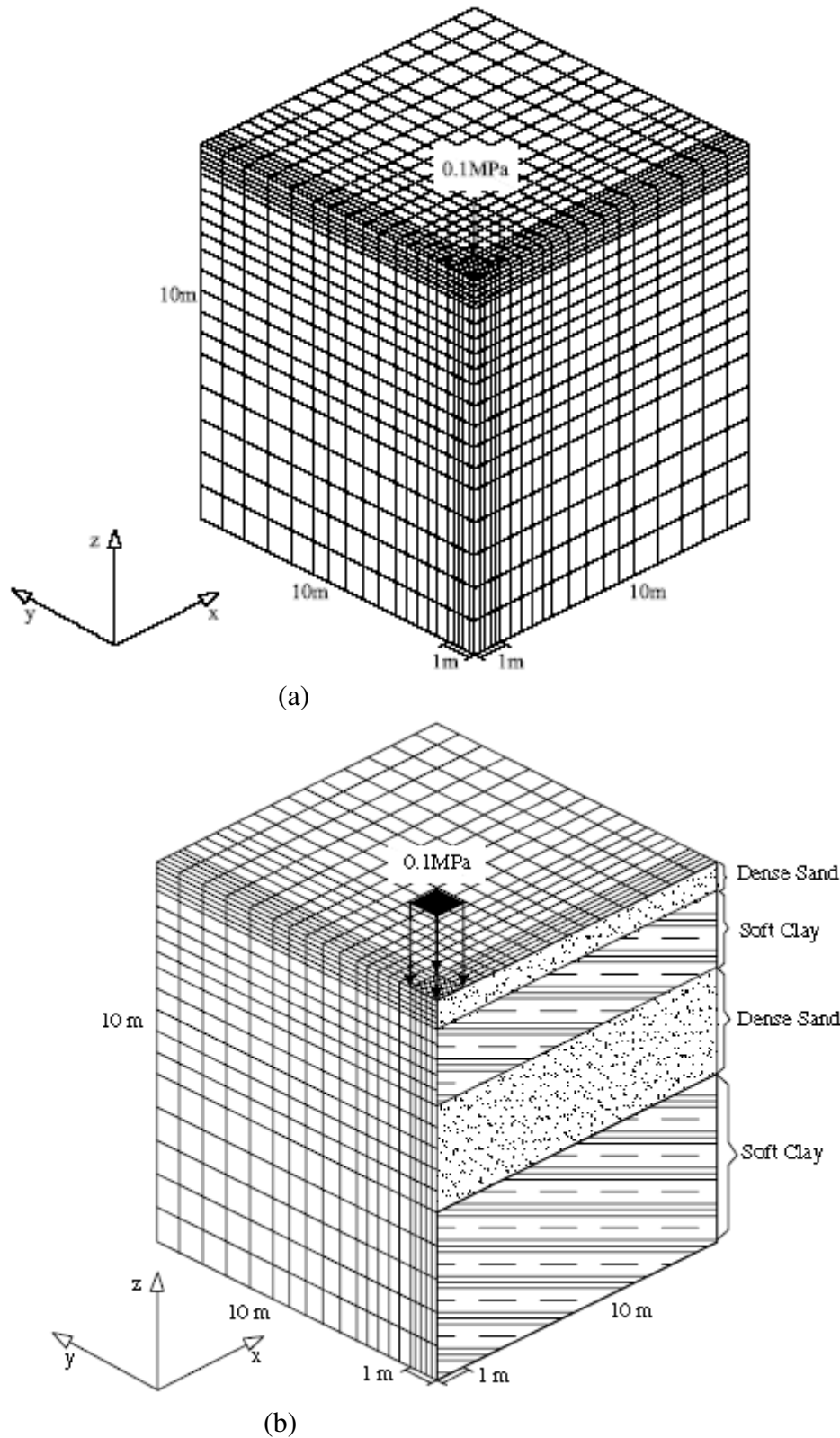
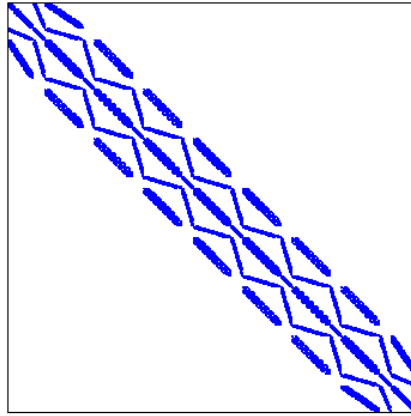
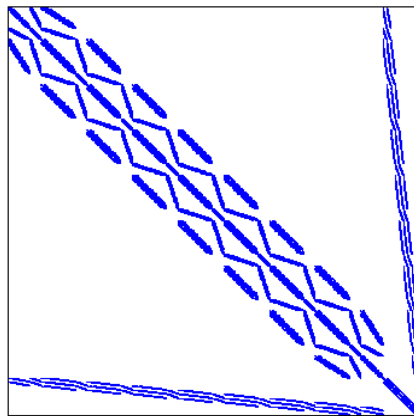


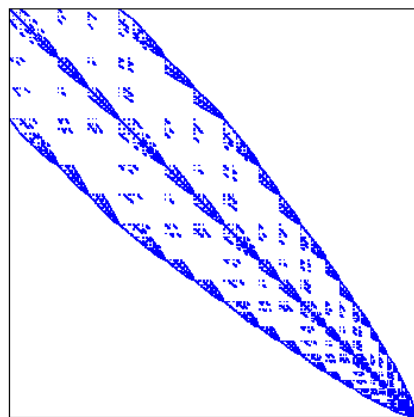
Figure 3.2. 20×20×20 finite element mesh of a symmetric quadrant footing (after Chen, 2005): (a) homogeneous soils 1 and 2, (b) layered soil 3.



(a)



(b)



(c)

Figure 3.3. Sparsity pattern of  $A$  in: (a) Natural ordering, (b) Block ordering, and (d) Reverse Cuthill-McKee technique on natural ordering.

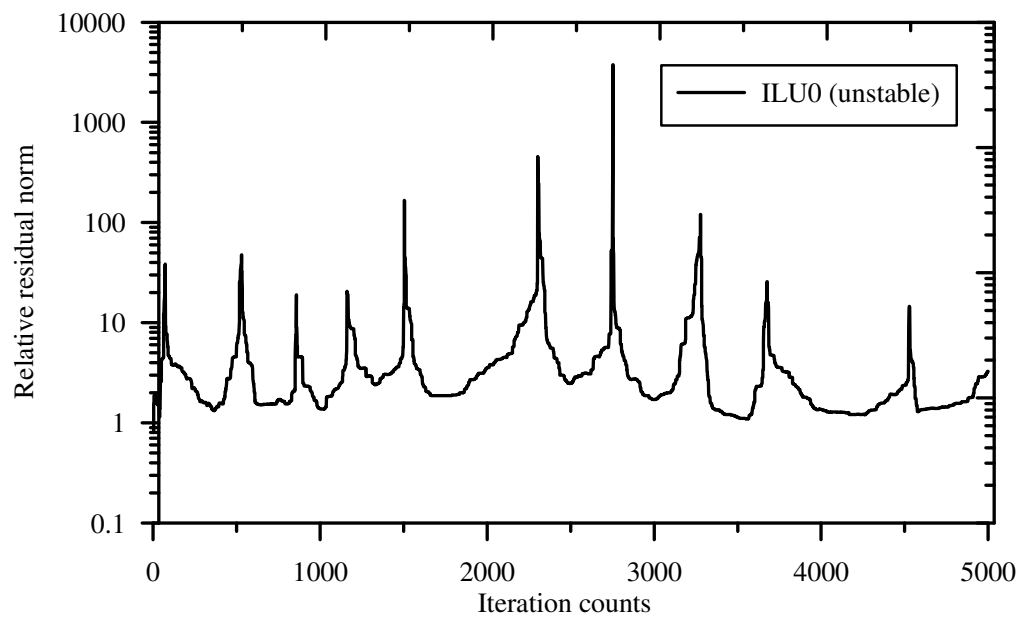


Figure 3.4. Typical relative residual norm of an unstable ILU0.

		<b>Condest</b>	
		Small	Large
<b>1/pivot</b>	Small	Inaccuracy due to dropping	Unstable triangular solve
	Large		Very small pivots

Figure 3.5. Interpretation of ILU statistics (after Chow and Saad, 1997).

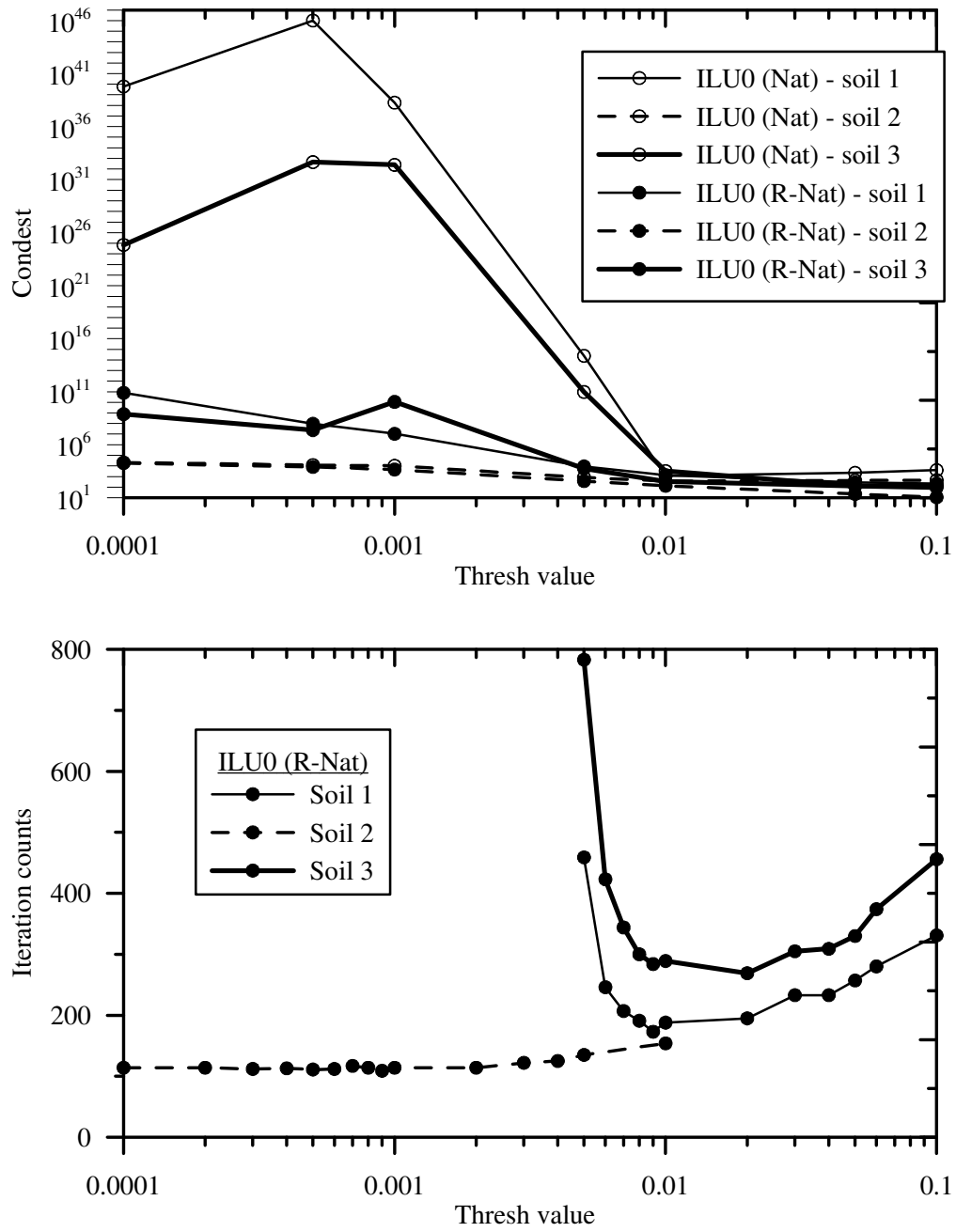


Figure 3.6. 20×20×20 mesh: Effect of threshold value on convergence of stabilized ILU0 for different soil profiles.

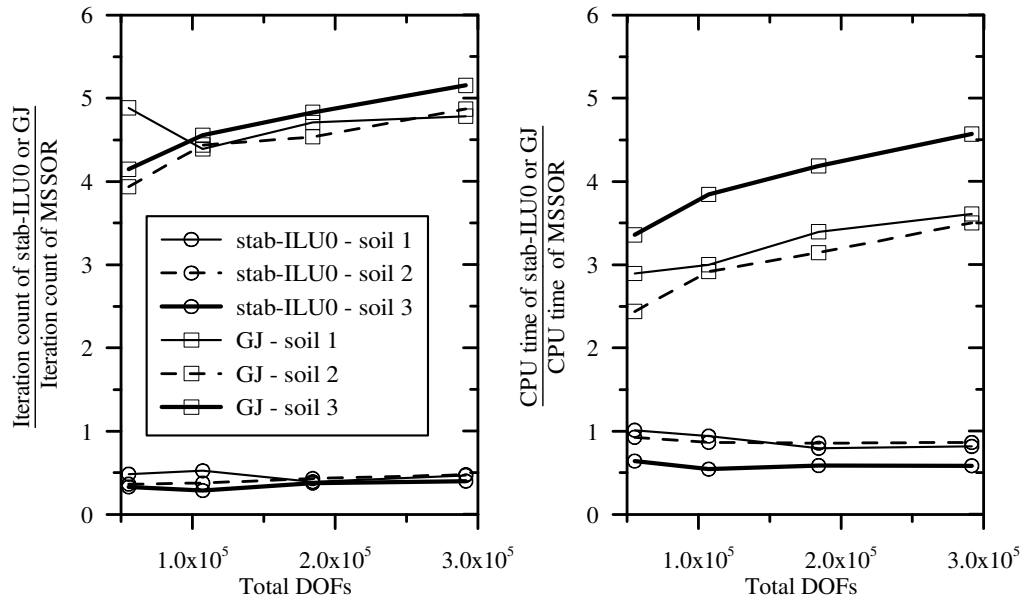


Figure 3.7. Performance of ILU0 and GJ preconditioners with respect to MSSOR preconditioner for different soil conditions.

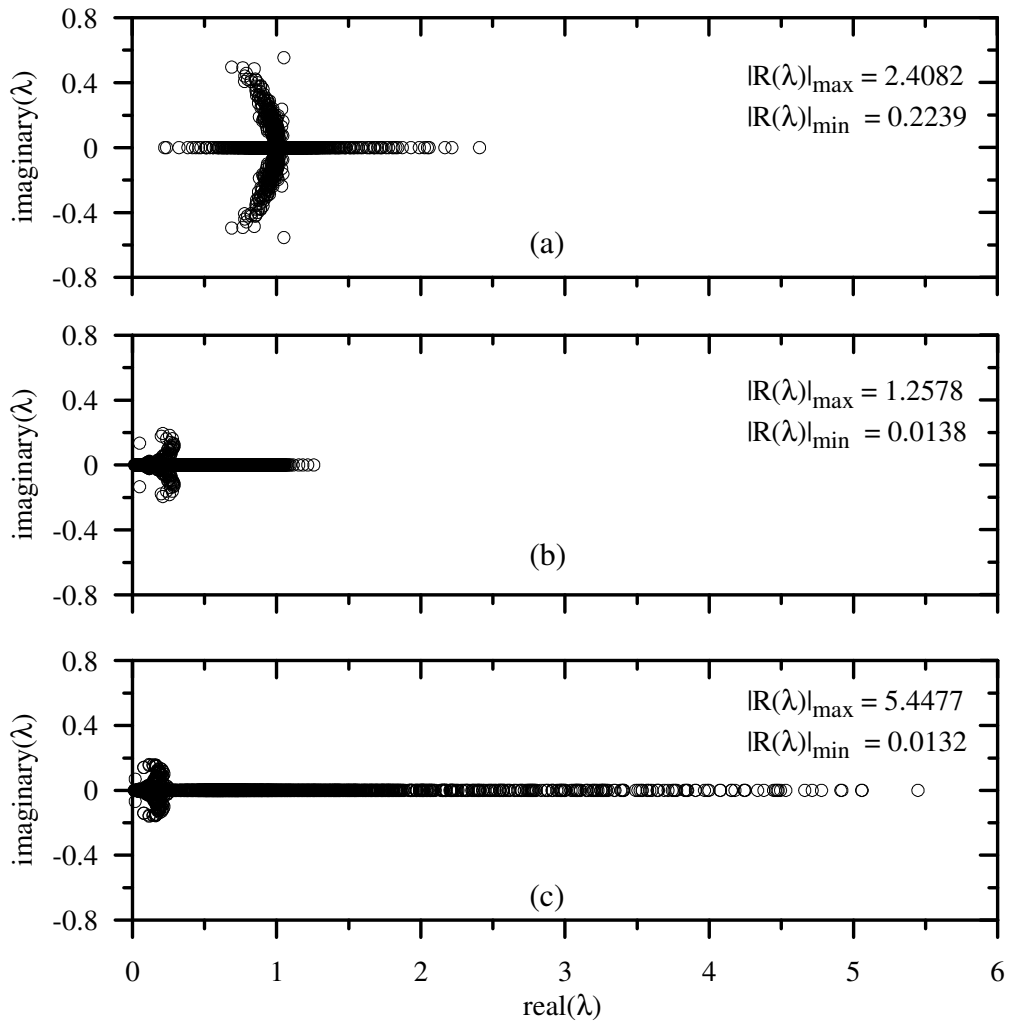


Figure 3.8. Eigenvalue distribution of preconditioned matrices with different preconditioners: (a) stab-ILU0; (b) MSSOR; and (c) GJ.

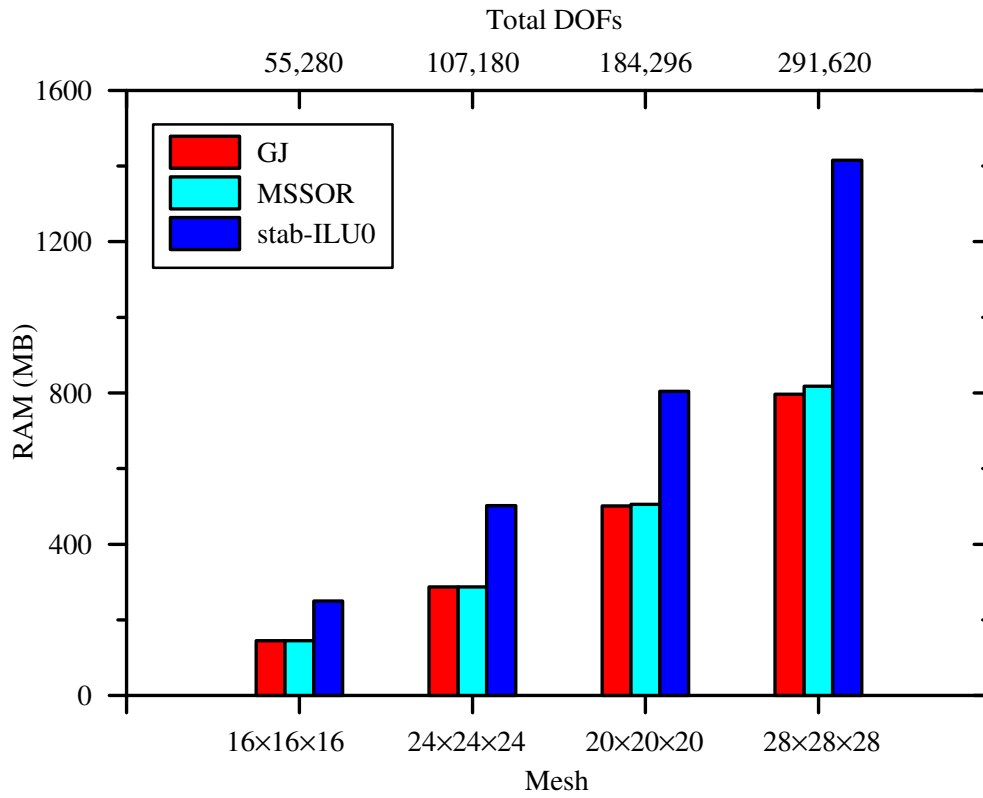


Figure 3.9. RAM usage by preconditioners with SQMR solver.

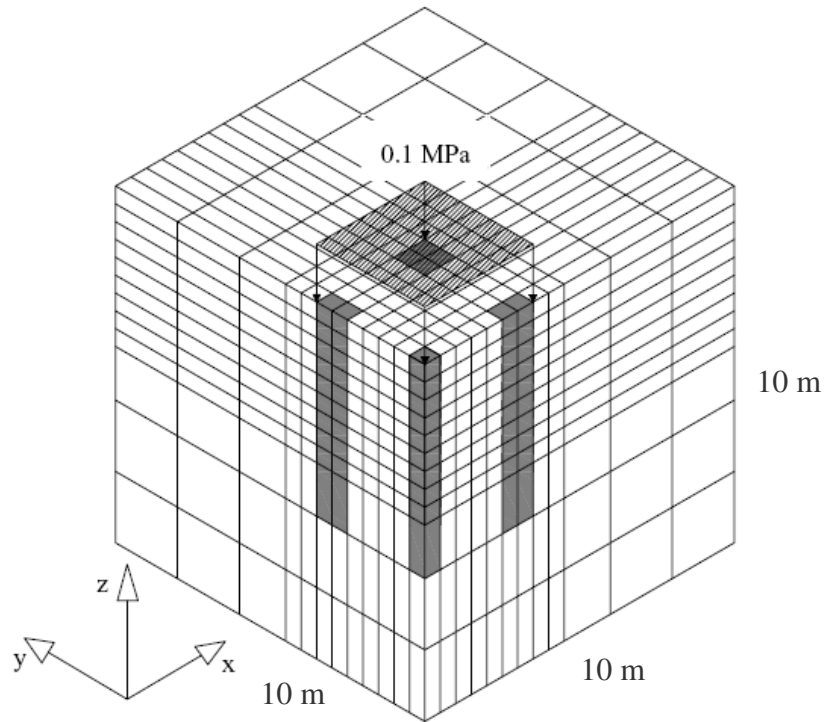


Figure 3.10. 12×12×12 mesh: 3D FE discretization of a quadrant symmetric 9-pile group foundation in a homogeneous clay (soil 1) with a uniform load (after Chen *et al.*, 2007).

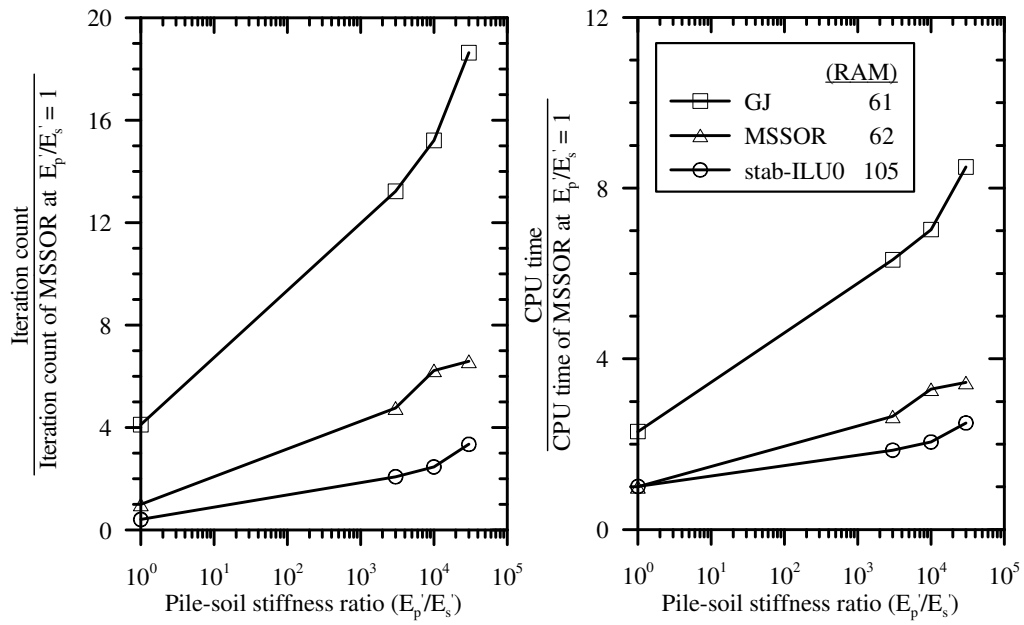


Figure 3.11. Performance of preconditioners for 9-pile group problem on homogeneous clay (soil 1).

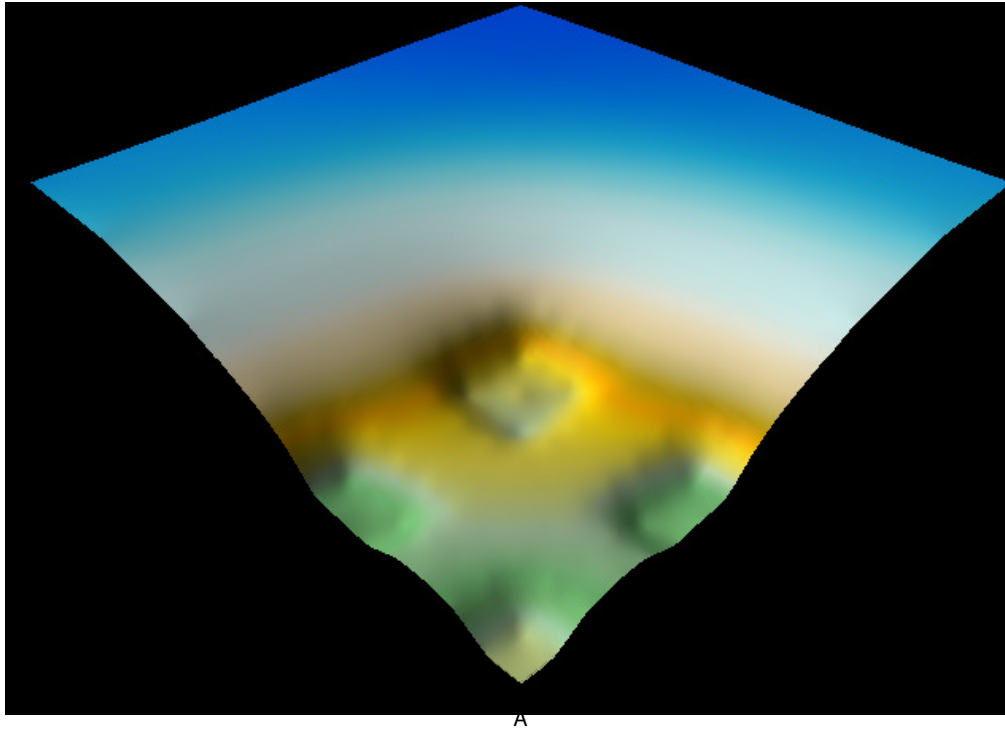


Figure 3.12. Ground surface settlement for 9-pile group after loading at 1<sup>st</sup> time step with pile stiffness of  $E'_p = 30000$  MPa and soil stiffness of  $E'_s = 1$  MPa.

Table 3.1. Three-dimensional finite element meshes.

Mesh size	16×16×16	20×20×20	24×24×24	28×28×28
Number of elements (nels)	4,096	8,000	13,824	21,952
Number of nodes	18,785	35,721	60,625	95,033
nnz ( $K$ )	7,790,768	15,573,896	27,285,204	43,758,934
nnz ( $B$ )	907,652	1,812,577	3,173,905	5,092,775
nnz ( $C$ )	110,446	215,818	373,030	592,450
nnz ( $A$ )	9,716,518	19,414,868	34,006,044	54,536,934
nnz ( $A$ ) / $N^2$ , %	0.32	0.17	0.10	0.06
Displacement DOFs ( $nd$ )	50,656	98,360	169,296	268,072
Pore pressure DOFs ( $np$ )	4,624	8,820	15,000	23,548
Total DOFs ( $N = nd + np$ )	55,280	107,180	184,296	291,620

Table 3.2. Effect of ordering on ILU0 preconditioned SQMR.

Mesh size		Soil 1 (Clay)			Soil 2 (Sand)			Soil 3 (Layered)		
		Nat	Blk	R-Nat	Nat	Blk	R-Nat	Nat	Blk	R-Nat
16×16×16	iter	Fail	166	90	108	142	84	Fail	487	217
	t <sub>t</sub> (s)		141.3	117.0 (3.58)	119.6	133.2	114.8 (3.56)		247.3	159.4 (3.56)
20×20×20	iter	Fail	268	Fail	169	271	114	Fail	928	Fail
	t <sub>t</sub> (s)		347.1		278.2	350.4	248.5 (7.16)		778.4	
24×24×24	iter	Fail	498	134	296	528	150	Fail	1677	320
	t <sub>t</sub> (s)		871.4	459.3 (12.72)	628.3	903.8	474.9 (12.48)		2216.1	670.6 (12.5)
28×28×28	iter	Fail	940	170	639	1277	195	Fail	3872	473
	t <sub>t</sub> (s)		2226.4	803.7 (20.20)	1650.7	2839.6	848.2 (20.16)		7543.2	1357.1 (20.05)

Note: Fail = relative residual shows no sign of decreasing with number of iteration counts (see Figure 3.4), iter = number of iteration counts, t<sub>t</sub> (s) = total runtime of the program including ordering if any, and the term in the parenthesis represents ordering time.

Table 3.3. Effect of ordering on MSSOR ( $\omega = 1$ ,  $\alpha = -4$ ) preconditioned SQMR.

Mesh size		Soil 1 (Clay)			Soil 2 (Sand)			Soil 3 (Layered)		
		Nat	Blk	R-Nat	Nat	Blk	R-Nat	Nat	Blk	R-Nat
16×16×16	iter	215	270	270	220	250	255	700	790	815
	$t_t$ (s)	120.8	139.0	154.8	122.8	131.9	150.8	257.7	284.5	308.4
20×20×20	iter	330	360	375	290	305	315	940	995	1075
	$t_t$ (s)	305.0	326.2	367.2	283.0	295.0	334.7	647.6	683.5	760.5
24×24×24	iter	420	480	475	355	365	395	1240	1455	1465
	$t_t$ (s)	621.0	688.0	742.7	558.1	594.6	661.1	1431.9	1652.0	1717.9
28×28×28	iter	510	560	590	415	470	480	1540	1705	1905
	$t_t$ (s)	1138.4	1231.0	1373.2	989.4	1085.2	1204.9	2766.2	3060.8	3483.3

Table 3.4. Effect of ordering on GJ ( $\alpha = -4$ ) preconditioned SQMR.

Mesh size		Soil 1 (Clay)			Soil 2 (Sand)			Soil 3 (Layered)		
		Nat	Blk	R-Nat	Nat	Blk	R-Nat	Nat	Blk	R-Nat
16×16×16	iter	1050	1057	1040	866	942	839	2903	3222	3046
	$t_t$ (s)	349.5	351.7	364.8	299.3	322.3	309.0	865.7	956.6	922.8
20×20×20	iter	1449	1482	1459	1287	1294	1294	4282	4606	4493
	$t_t$ (s)	915.0	935.1	959.4	826.0	835.5	867.8	2489.8	2663.6	2641.3
24×24×24	iter	1978	2089	2021	1610	1621	1695	5000+	5000+	5000+
	$t_t$ (s)	2109.1	2224.4	2219.2	1755.0	1772.6	1903.2			
28×28×28	iter	2440	2656	2490	2021	1974	1999	5000+	5000+	5000+
	$t_t$ (s)	4109.4	4449.5	4182.5	3465.7	3400.9	3436.0			

Note: 5000+ = maximum iteration count (5000) is reached without satisfying the accuracy criterion for relative residual norm.

Table 3.5. Statistics that can be used to evaluate an incomplete factorization.

Statistic	Meaning
condest	$\ (\bar{L}\bar{U})^{-1} e\ _{\infty}, e = (1, 1, \dots, 1)^T$
1/pivot	Size of reciprocal of the smallest pivot
$\max(\bar{L} + \bar{U})$	Size of the largest element in $\bar{L}$ and $\bar{U}$ factors

Table 3.6. ILU statistics and possible reasons of failure for soil 1 (soft clay).

Ordering	condest	1/Pivot	$\max(\bar{L} + \bar{U})$	iters	Reason for failure
Mesh: 16×16×16					
Nat	<b>8.36E+64</b>	1.75E+04	9.19E+06	Fail	unstable solve
Blk	5.68E+03	1.52E+03	1.09E+01	166	
R-Nat	1.31E+03	4.43E+03	2.22E+01	90	
Mesh: 20×20×20					
Nat	<b>1.56E+37</b>	8.68E+06	8.08E+05	Fail	unstable solve
Blk	1.17E+04	2.99E+03	8.74E+00	268	
R-Nat	<b>2.50E+08</b>	5.45E+07	8.89E+05	5000+	small pivot? inaccuracy due to dropping?
Mesh: 24×24×24					
Nat	<b>3.63E+72</b>	1.45E+05	1.73E+06	Fail	unstable solve
Blk	2.87E+04	5.16E+03	7.29E+00	498	
R-Nat	4.60E+03	1.49E+04	3.91E+01	134	
Mesh: 28×28×28					
Nat	<b>1.17E+51</b>	2.91E+05	1.46E+08	Fail	unstable solve
Blk	5.48E+05	8.18E+03	1.19E+01	940	
R-Nat	7.55E+03	2.34E+04	4.86E+01	170	

Note: Fail = relative residual shows no sign of decreasing with number of iteration counts.

Table 3.7. ILU statistics and possible reasons of failure for soil 2 (sand).

Ordering	condest	1/Pivot	$\max(\bar{L} + \bar{U})$	iters	Reason for failure
Mesh: 16×16×16					
Nat	1.49E+04	5.98E+03	1.12E+03	109	
Blk	1.36E+04	5.76E+03	1.09E+03	143	
R-Nat	1.28E+04	6.07E+03	1.10E+03	82	
Mesh: 20×20×20					
Nat	2.15E+04	7.51E+03	8.89E+02	168	
Blk	1.96E+04	7.33E+03	8.74E+02	258	
R-Nat	1.88E+04	7.64E+03	8.78E+02	114	
Mesh: 24×24×24					
Nat	3.40E+04	9.02E+03	1.40E+03	286	
Blk	2.56E+04	8.88E+03	1.40E+03	508	
R-Nat	2.48E+04	9.26E+03	7.33E+02	151	
Mesh: 28×28×28					
Nat	3.15E+05	1.05E+04	6.89E+02	639	
Blk	5.99E+05	1.99E+04	6.24E+02	1277	
R-Nat	3.12E+04	1.08E+04	6.28E+02	195	

Table 3.8. ILU statistics and possible reasons of failure for soil 3 (layered soil).

Ordering	condest	1/Pivot	$\max$ ( $\overline{L} + \overline{U}$ )	iters	Reason for failure
Mesh: 16×16×16					
Nat	<b>1.16E+35</b>	2.46E+04	8.69E+08	Fail	unstable solve
Blk	1.11E+04	5.76E+03	8.14E+02	504	
R-Nat	9.95E+03	6.07E+03	1.10E+03	221	
Mesh: 20×20×20					
Nat	<b>1.29E+28</b>	3.83E+04	3.20E+06	Fail	unstable solve  small pivot? inaccuracy due to dropping?
Blk	1.70E+04	7.33E+03	6.53E+02	955	
R-Nat	<b>3.56E+08</b>	5.45E+07	8.89E+05	5000+	
Mesh: 24×24×24					
Nat	<b>8.57E+26</b>	1.84E+05	4.50E+06	Fail	unstable solve
Blk	2.37E+04	8.88E+03	5.44E+02	1988	
R-Nat	2.18E+04	1.49E+04	7.33E+02	332	
Mesh: 28×28×28					
Nat	<b>2.10E+32</b>	1.37E+06	3.82E+07	Fail	unstable solve
Blk	2.98E+04	1.04E+04	4.66E+02	3872	
R-Nat	2.84E+04	2.33E+04	6.28E+02	473	

## Chapter 4

---

# BLOCK DIAGONAL PRECONDITIONERS FOR DRAINED ANALYSIS

### 4.1. Introduction

As discussed in Chapter 2, some effective preconditioners have been proposed in the recent years for the iterative solution of geotechnical problems. However, little or no attention has been paid on the ill-conditioning due to relative differences in stiffnesses of materials. Recently, ill-conditioning of the system due to contrasts in large stiffness, but in a different context, with the use of large penalty terms in modeling the rock faults was discussed by Ferronato *et al.* (2008). Similarly, Augarde *et al.* (2008) demonstrated that the material properties such as Poisson's ratio and constitutive matrix are important on the performance of iterative methods.

Of course, the linear system is ill-conditioned (see, for example, Zienkiewicz, 2000) for undrained analysis with  $\nu$  close to 0.5. For such incompressible problems, Phoon *et al.* (2003) demonstrated that cost-effective and exact solution can be achieved without any numerical instability by using two-field mixed formulation. For a typical drained analysis, Poisson's ratio usually varies in the range of  $\nu = 0.2$  to  $0.35$ . This variation was shown to be

less significant for the convergence of SJ preconditioned system (e.g. Smith and Wang, 1998). On the other hand, materials with differing Young's moduli are commonly encountered in geotechnical engineering, mainly because of two reasons: (i) variability in natural geomaterials, and (ii) soil-structure interaction problems. For example, Young's moduli can vary from 1 MPa or less for very soft soils to more than 100,000 MPa for rocks. Similarly, in soil-structure interaction problems, the Young's modulus of a structural material (e.g. reinforced concrete or steel) can be four to five orders of magnitude larger than the Young's modulus of the surrounding soil. The typical Young's moduli of reinforced concrete and steel are 30,000 MPa and 205,000 MPa, respectively. Another common class of problems is ground improvement involving introduction of stiff materials into the ground such as grouting, cement-mixed piles, stone columns, etc. Hence, this is a common practical problem and there is a need to develop effective solution techniques that are capable of exploiting such differences in stiffnesses of materials to accomplish the solutions in acceptable computing time for large-scale problems.

The aim of this study is to investigate the ill-conditioning of the linear system due to large relative differences in material stiffnesses and its mitigation by block diagonal preconditioners. Particular emphasis is given to the numerical performance of the preconditioners based on various inexact forms of the blocks. The results are discussed and evaluated on a wide range of material stiffnesses and structural DOFs with the help of two typical soil-structure interaction problems: piled-raft foundation and tunneling.

## 4.2. Soil-structure interaction problem and preconditioning

Confining our study to drained boundary value problems, finite element (FE) discretization of the continuum leads to a well-known symmetric positive definite linear system (Smith and Griffiths, 1997):

$$Ku = f \quad (4.1)$$

where  $K = K^T \in \Re^{N \times N}$  is the FE stiffness matrix,  $u$  and  $f \in \Re^N$  are the displacement and load vectors respectively,  $N$  is the total number of displacement degrees of freedom (DOFs) excluding any fixities. The stiffness matrix  $K$  is evaluated as follows (e.g. see Smith and Griffiths, 1997):

$$K = \sum_e \left( \int_V B_u^T D B_u dV \right) \quad (4.2)$$

where  $B_u$  is the shape function derivative for displacement, and  $D$  the stress-strain matrix. The integration is performed over the volume domain of each finite element (denoted as  $V$ ) and the global matrix is formed by summing the contribution of each element (symbolically represented by  $\sum_e$ ).

In order to investigate the effect of stiffness of different material zones in the formulation of FE stiffness matrix, the finite element discretization is demonstrated using a simple one-dimensional (1D) constrained compression problem (e.g. Oedometer test set up, see Figure 4.1). For simplicity, Equation (4.2) is evaluated analytically using a simple one-dimensional two-noded element. The continuous displacement variable  $u$  is approximated in terms of discrete nodal values. For a 2-noded element, the field variable is assumed to vary linearly between the nodal values. Hence,

$$u = N_1 u_1 + N_2 u_2 = \begin{bmatrix} N_1 & N_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad (4.3)$$

For a linear element, the shape functions for displacement ( $N_u$ ) in this example is:

$$N_u = \begin{bmatrix} N_1 & N_2 \end{bmatrix} = \begin{bmatrix} 1 - \frac{x}{l} & \frac{x}{l} \end{bmatrix} \quad (4.4)$$

where  $x$  is the local spatial coordinate and  $l$  is the element size. The derivative of  $N_u$  is:

$$B_u = \frac{dN_u}{dx} = \begin{bmatrix} -\frac{1}{l} & \frac{1}{l} \end{bmatrix} = \frac{1}{l} \begin{bmatrix} -1 & 1 \end{bmatrix} \quad (4.5)$$

The element stiffness matrix is now evaluated as:

$$K^e = \frac{D}{l^2} \int_0^l \begin{bmatrix} -1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix} (a \, dx) = \frac{aD}{l} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (4.6)$$

where  $a$  is the cross-sectional area of the element. For constrained 1D compression,

$$D = \frac{(1 - \nu')}{(1 + \nu')(1 - 2\nu')} E' \quad (4.7)$$

is known as constrained modulus;  $E'$  is the effective Young's modulus of a material, and  $\nu'$  is the effective Poisson's ratio. Hence, the element stiffness matrices for the porous stone and the soil element, respectively, are:

$$\begin{aligned} K_{ps}^e &= \frac{aD_{ps}}{l} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \\ K_s^e &= \frac{aD_s}{l} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \end{aligned} \quad (4.8)$$

where the subscripts  $ps$  and  $s$  stand for porous stone and the soil, respectively.

Finite element model of the entire problem is obtained by assembling the

element equations and applying the boundary condition, which yields after rearranging the variables:

$$\frac{a}{l} \begin{bmatrix} D_{ps} & -D_{ps} & 0 & 0 & 0 \\ -D_{ps} & D_{ps} + D_s & 0 & -D_s & 0 \\ 0 & 0 & D_{ps} + D_s & 0 & -D_s \\ 0 & -D_s & 0 & 2D_s & -D_s \\ 0 & 0 & -D_s & -D_s & 2D_s \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_5 \\ u_3 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} F \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \quad (4.9)$$

As can be seen from Equation (4.8), element stiffness matrices are proportional to the Young's moduli of the materials of corresponding elements. In the real field, the porous stone can be a structure (footing, pile, tunnel, retaining wall, etc.) or a stiff geo-material. In an extreme case, the Young's modulus of a structure can be as high as 205,000 MPa, such as for steel structures, and the Young's modulus of soil can be as low as 1 MPa for a very soft clay. The corresponding soil-structure stiffness ratio can be as large as 205,000. The term 'soil-structure stiffness-ratio' is defined here as the ratio of Young's moduli of structural (stiff) material and soil. Geometry and boundary conditions have not been considered in the definition of stiffness-ratio. The FE formulation of such systems will produce entries corresponding to structural (or stiff) elements significantly larger in magnitude than those produced by soil elements in the global  $K$  [see Equation (4.9)]. This is the main source of ill-conditioning when dealing with two very different materials. For example, for a model piled-raft foundation shown in Figure 4.2b, Figure 4.3 shows the proportional increase in the condition number of the unpreconditioned  $K$  with an increase in pile-soil stiffness ratio  $(E'_p/E'_s)$ . More numerical results will be discussed later in Section 4.3, where  $E'_p$  and  $E'_s$  are the Young's moduli of

pile and soil, respectively. For this reason, it is prudent to partition the global stiffness matrix  $K$  into  $2 \times 2$  blocks such that:

$$K = \begin{bmatrix} P & L \\ L^T & G \end{bmatrix} \quad (4.10)$$

where submatrix  $P = P^T \in \Re^{m \times m}$  is structure (or stiff material) stiffness matrix, submatrix  $G = G^T \in \Re^{n \times n}$  is soil stiffness matrix, and submatrix  $L \in \Re^{m \times n}$  is the soil-structure connection matrix.  $m$  and  $n$  are the DOFs corresponding to structure and soil elements, respectively. Partitioning of  $K$  into blocks in this way allows preconditioners for the structural block and the soil block to be chosen independently.

#### 4.2.1. Block diagonal Preconditioner

For  $2 \times 2$  block structured matrix (4.10), Murphy *et al.* (2000) have shown that preconditioners incorporating an exact Schur complement matrix lead to exactly three distinct eigenvalues of the preconditioned matrices. The preconditioner considered in their paper is:

$$M_{MURPHY} = \begin{bmatrix} P & 0 \\ 0 & S \end{bmatrix} \quad (4.11)$$

where

$$S = G - L^T P^{-1} L \quad (4.12)$$

is the Schur complement matrix. However, when the second diagonal block,  $G$ , is positive definite and most of the eigenvalues of  $L^T P^{-1} L$  are much smaller than the smallest eigenvalue of  $G$  (unlike the saddle-point problems considered in their paper) the subtraction of  $L^T P^{-1} L$  in (4.12) was found to be unfruitful. Hence, the subtraction term is dropped and the block diagonal preconditioner for  $K$  takes the form:

$$M_K = \begin{bmatrix} P & 0 \\ 0 & G \end{bmatrix} \quad (4.13)$$

Let  $R_P$  and  $R_G$  be the Cholesky factors of  $P$  and  $G$ , respectively. Applying left-right preconditioning to (4.10), the preconditioned matrix is given by:

$$\tilde{K} = \begin{bmatrix} I_m & R_P^{-T} L R_G^{-1} \\ R_G^{-T} L^T R_P^{-1} & I_n \end{bmatrix} = \begin{bmatrix} I_m & \tilde{L} \\ \tilde{L}^T & I_n \end{bmatrix} \quad (4.14)$$

with

$$\tilde{L} = R_P^{-T} L R_G^{-1} \quad (4.15)$$

where  $I_m$  and  $I_n$  are identity matrices of appropriate sizes, respectively. Recall that  $m$  and  $n$  are structure (or stiff material) and soil DOFs, respectively. According to the theorem in the Appendix of Phoon *et al.* (2002), if a matrix is of the form:

$$A = \begin{bmatrix} \xi I_m & \beta Y \\ \alpha Y^T & -\eta I_n \end{bmatrix} \quad (4.16)$$

The eigenvalues of  $A$  are given by:

$$\lambda(A) = \begin{cases} \frac{1}{2} \left( \xi - \eta + \sqrt{(\xi + \eta)^2 + 4\alpha\beta[\sigma_j(Y)]^2} \right), & j = 1, 2, \dots, r \\ \frac{1}{2} \left( \xi - \eta - \sqrt{(\xi + \eta)^2 + 4\alpha\beta[\sigma_j(Y)]^2} \right), & j = 1, 2, \dots, r \\ \xi & \text{with multiplicity } m + n - 2r \end{cases} \quad (4.17)$$

where  $\xi$ ,  $\alpha$ ,  $\beta$  and  $\eta$  are arbitrary constants, and  $r$  is the rank of block (1, 2),

which is  $\tilde{L}$  in our case. Let the non-zero singular values of  $\tilde{L}$  be:

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0. \quad (4.18)$$

Comparing Equations (4.14), (4.16) and (4.17), the eigenvalues of the preconditioned matrix  $\tilde{K}$  can be written as:

$$\lambda(\tilde{K}) = \begin{cases} 1 + \sigma_j(\tilde{L}), & j = 1, 2, \dots, r \\ 1 - \sigma_j(\tilde{L}), & j = 1, 2, \dots, r \\ 1 & \text{with multiplicity } m + n - 2r \end{cases} \quad (4.19)$$

*Remark 1*

As can be seen from Equation (4.19), when an exact block diagonal preconditioner is used, the multiplicity of unity eigenvalue of the preconditioned matrix is controlled by the rank  $r$  of  $\tilde{L}$ . Since  $L$  is the link matrix of soil and structure [see (4.9) or (4.10)], the rank  $r$  depends very much on the dimension of block  $P$ , which, in general, is small ( $m \ll n$ ) in the FE analyses of most of the soil-structure interaction problems. In addition, generally,  $r \leq m$  because the entries of  $L$  are contributed from soil-structure interface nodes only and, hence, many rows of  $L$  (e.g. interior structural nodes or boundary nodes that are unconnected to soil) can be filled with zeros; for example, the first row of  $L$  is zero [Equation (D2)] in the Appendix D for 1D FE discretization of oedometer setup.

*Remark 2*

It can be shown, from the first order approximation, that the order of entries of  $\tilde{L}$  (4.15) is inversely proportional to the square root of soil-structure stiffness ratio as below:

$$O(\tilde{L}) = O(R_P^{-T} L R_G^{-1}) \approx O\left(\frac{1}{\sqrt{E'_{ps}}} \times E'_s \times \frac{1}{\sqrt{E'_s}}\right) = O\left(\sqrt{\frac{E'_s}{E'_{ps}}}\right) \quad (4.20)$$

where  $E'_{ps}$  and  $E'_s$  are Young's moduli of porous stone (or structure) and soil (Figure 4.1), respectively. For example, the entries of  $\tilde{L}$  are affected by the inverse of soil-structure stiffness ratio, except the second row, for the

considered 1D problem in Figure 4.1 (see Appendix D for details). In other words, most of the singular values of  $\tilde{L}$  will tend to be smaller as the soil-structure stiffness ratio increases. This, on the other hand, will improve the clustering of the  $2r$  eigenvalues of the preconditioned matrix that are symmetric about unity as indicated by Equation (4.19). Numerical results will be discussed in Section 4.3.1.1.

*Remark 3*

Although the eigenvalue distribution of the preconditioned matrix for the theoretical exact block diagonal preconditioner has an attractive clustering property with increasing soil-structure stiffness ratios, the exact block diagonal preconditioner is impractical when the system is very large. However, the theorem does serve as a useful guide to clarify the convergence behavior that one may expect from a Krylov subspace method when increasingly better approximations of  $P$  and  $G$  are used. The approximations would still produce some eigenvalue clustering, albeit in a more diffused way. The actual convergence behavior associated with more practical preconditioners can only be evaluated by numerical experiments.

#### **4.2.2. Inexact block diagonal preconditioners**

We have observed that the block  $P$  is the main source of ill-conditioning and a better approximation of block  $P$  is needed. For example,  $P$  is close to singular when the soil-structure stiffness ratio is very large [see Equation (D2) in the Appendix D]. Hence, we emphasized the study of the numerical performance of inexact block diagonal preconditioners over a range of approximations for block  $P$ . The approximations considered are from the

crudest level, where  $\hat{P}_1 = \text{diag}(P)$ , to the finest level, where  $\hat{P}_4 = P$ . In the intermediate levels, we use an inexact  $\hat{P}_2$  that is obtained from SSOR (Symmetric Successive Over Relaxation) factorization,  $\hat{P}_2 = (L_p + \tilde{D}_p)(\tilde{D}_p)^{-1}(L_p^T + \tilde{D}_p)$ , where  $L_p$  and  $D_p$  are strictly lower triangular and diagonal part of  $P$ , respectively.  $\tilde{D}_p = D_p/\omega$  and  $\omega$  is the relaxation parameter, which is set to 1 throughout the study. We also use inexact  $\hat{P}_3 = \text{ILU0}(P)$  (incomplete LU factorization with zero fill-ins). For the study of these inexact forms of block  $P$ , we simply use the diagonal approximation of the soil block  $G$ . Incorporating all these, the inexact block diagonal preconditioner for  $K$  takes the following form:

$$\hat{M}_K = \begin{bmatrix} \hat{P} & 0 \\ 0 & \text{diag}(G) \end{bmatrix} \quad (4.21)$$

where the inexact forms used for  $P$  are:

$$\hat{P}_1 = \text{diag}(P), \quad \hat{P}_2 = \text{SSOR}(P), \quad \hat{P}_3 = \text{ILU0}(P), \quad \hat{P}_4 = P \quad (4.22)$$

The approximations  $\hat{P}_1$  and  $\hat{P}_2$  require only the storage of a single  $m \times 1$  vector for the preconditioning purpose and are the cheapest. However, the approximations  $\hat{P}_3$  and  $\hat{P}_4$  require the entire  $P$  matrix to be stored. For the inexact form of  $\hat{P}_3 = \text{ILU0}(P)$ , its computational and storage costs are moderate if  $P$  is sparse. In contrast,  $\hat{P}_4$  may require a large amount of memory to store its Cholesky factor and it can be computationally expensive for a very large soil-structure problems. Nonetheless, it will be significantly less expensive in comparison to exact block diagonal preconditioner (4.13) involving an exact  $G$  block. In practice, the size of the block  $P$  is significantly

less than the size of block  $G$  (i.e.  $m \ll n$ ) because the soil mesh must increase as the size of structure increases, to avoid possible boundary effects. In addition, we also considered the incomplete factorization with partial fill-ins [such as ILUT (Saad, 1994b)] for the approximation of block  $P$  in Section 4.3.1.2.

Once the appropriate inexact form for block  $P$  is identified, we fine-tune the inexact form for block  $G$  by studying the following three possibilities:

$$\hat{G}_1 = \text{diag}(G), \quad \hat{G}_2 = \text{SSOR}(G), \quad \hat{G}_3 = \text{ILU0}(G) \quad (4.23)$$

Note that  $G$  is the soil stiffness matrix and its dimension will be significantly larger than of block  $P$  in all practical geotechnical problems. Hence, any inexact form of  $G$  beyond a simple diagonal or SSOR will become very costly quickly with an increase in problem size and may not be practical. For example, the inexact forms that are based on ILU0 factorization of  $G$  would be extremely expensive to compute (both in terms of storage and time) because the full  $G$  needs to be stored explicitly. Hence,  $\hat{G}_3$  can be considered as the limit of what is practical. On the other hand, the SSOR approximation of blocks has an advantage in that it can exploit the Eisenstat trick (Eisenstat, 1981) for the preconditioned matrix-vector multiplication. It also requires the storage of only the upper triangular part of  $G$ .

For ILU factorization, the matrix is reordered prior to factorization using Reverse Cut-hill McKee (RCM) permutation (George and Lui, 1981) so as to minimize possible incorrectness due to dropping in ILU. The subroutines of ILU factorization are obtained from SPARSKIT (Saad; 1996) and RCM algorithms are from the RCM Package (Burkardt). Note that the factors  $\bar{L}$  and

$\bar{U}$  from ILU subroutines are nonsymmetric for a general matrix. However, the preconditioner needs to be symmetric for a conjugate gradient solver (Barrett *et al.*, 1994). For symmetric positive definite  $K$ , its incomplete Cholesky factorization ( $R^T R$ ) can be obtained by scaling each row of  $\bar{U}$  with the reciprocal of the square root of its pivot, i.e.

$$R_{(i,:)} = \frac{1}{\sqrt{\bar{U}_{(i,i)}}} * \bar{U}_{(i,:)} \quad \text{for } i = 1 \text{ to } N. \quad (4.24)$$

Similarly, prior to Cholesky factorization (Ng and Peyton, 1993), the matrix is reordered using Multiple Minimal Degree (MMD) permutation (George and Lui, 1981) to minimize the possible fill-ins in factorization.

### 4.3. Numerical results

Our purpose now is to evaluate the numerical performance of various inexact block diagonal preconditioners discussed in the preceding Section for some representative soil-structure interaction problems. The evaluation is based on the following:

1. Number of iterations required to converge for a preconditioner
2. Total CPU time taken for the solution
3. Random access memory (RAM) required for the execution of the problem with that preconditioner.

PCG (Hestenes and Stiefel, 1952; Shewchuk, 1994) is taken as an iterative solver (see Section 2.1), which is best for the symmetric positive definite linear systems (Barrett *et al.*, 1994). The two soil-structure interaction problems studied are piled-raft foundation and tunneling. The former is solved using an in-house Fortran code compatible to the code of Smith and Griffiths'

(1997) and latter is solved after implementing the above code into GeoFEA (2006). GeoFEA is a finite element program which can be used for drained, undrained and time dependent analysis of static problems under monotonic loading/unloading conditions (<http://www.geosoft.sg/>). See Chapter 6 for more details about GeoFEA implementation.

#### **4.3.1. Piled-raft foundation**

A piled-raft foundation may lead to significant economical benefits compared to classical piled or raft foundations because the total load coming from the superstructure is partly shared by the raft through contact with soil and the remaining load is shared by piles through skin friction (Katzenbach *et al.*, 2000; Poulos, 2001a; Maharaj, 2004). The use of piled raft foundations has become more popular in recent years primarily because they provide better control of the differential settlement or they reduce the overall settlement (Poulos, 2001a; Reul and Randolph, 2003; Novak *et al.*, 2005; Small and Liu, 2008). It is obvious that the ultimate load capacity of the entire system will be increased as well.

Our purpose now is to study various preconditioning strategies detailed in the Section 4.2 for the mitigation of material ill-conditioning in the iterative solution of a large-scale piled raft foundation in a Desktop PC. The problem of interest involves a typical 9 (3×3) piled-raft problem. The pile cross-section is square with 1 m width. The pile-spacing is equal to 3 m, the raft (pile-cap) thickness is 3 m and is in direct contact with ground and the cap overhang is 0.5 m. The length of the pile, including the raft thickness, is taken as 20 m. A uniform load of 100 kPa is applied to the entire cap area. Symmetry consideration allows only a quadrant of the foundation to be analyzed. The

quadrant is discretized into  $25 \times 25 \times 35$  finite element mesh as shown in Figure 4.2a. The mesh comprises of 21,875 20-noded brick elements, with a total of 12,767 pile displacement degrees of freedom (DOFs) and 254,913 soil displacement DOFs. The pile and raft materials are assumed to be the same. Hence, the number of pile DOFs includes the number of raft DOFs as well. This forms the submatrix  $P$  [see Equation (4.10)] and termed as “pile block” in this example. The dimension of  $P$  is only 4.76% of the total dimension of the stiffness matrix  $K$ . A similar problem involving a  $7 \times 7 \times 7$  coarse mesh (Figure 4.2b) is configured so that it is large enough to study the convergence behavior, and yet small enough for the spectral properties of the preconditioned system to be examined readily. The base of the mesh is assumed to be fixed in all directions. Side face boundaries are constrained in the transverse directions but free in in-plane directions.

All materials (pile, raft, and soil) are assumed as linear elastic with constant Poisson’s ratios of  $\nu'_s = 0.3$  for soil and  $\nu'_p = 0.2$  for pile materials. Young’s modulus of the soil is held constant ( $E'_s = 5$  MPa) but the Young’s modulus of pile is varied ( $E'_p = 100$  to 205,000 MPa) to alter the condition number of the stiffness matrix. The corresponding pile-soil stiffness ratio ( $E'_p/E'_s$ ) ranges from 20 to 41000, which are typical for most soil-structure interaction problems. The lower bound is typical of sand piles used in ground improvement and the upper bound is typical of steel piles. The lower bound also captures the natural variation of soil types. In addition,  $E'_p/E'_s$  equals to 1 is also considered. It is studied for completeness, rather than realism. It refers to the condition where all pile/raft elements are replaced by soil elements, i.e.

a homogeneous problem domain. This configuration is included as a benchmark to gauge the effect of difference in stiffness relative to the homogeneous soil condition on various preconditioning methods. As expected, the condition number of  $K$  increases with increasing pile-soil stiffness ratios, as shown in Figure 4.3.

#### 4.3.1.1. Validation of theory

Figure 4.3 shows that the number of iteration counts of the theoretical exact block diagonal preconditioner (4.13) decreases with the increase in pile-soil stiffness ratios ( $E'_p/E'_s$ ), opposite to the trend of unpreconditioned  $K$ . Hence, the exact block diagonal preconditioner actually exploits the large  $E'_p/E'_s$  to be an advantage. The reduction in iteration count is mainly because of the reduction in condition number of the preconditioned system (Figure 4.3) and the increase in clustering of eigenvalues towards unity with increasing  $E'_p/E'_s$  (Figure 4.4). Similar gain in the convergence behavior was also observed for the problems in 1D and 2D. This is in line with the Remark 2 in Section 4.2.1. However, a stable condition number of  $\tilde{K}$  (Figure 4.3) may be because at least one singular value of  $\tilde{L}$  is unaffected by the stiffness ratios (see the second row of  $\tilde{L}$  for 1D example in the Appendix D), resulting almost constant extreme eigenvalues of  $\tilde{K}$  [Figure 4.4 and Equation (4.19)]. However, the exact block diagonal preconditioner is very expensive for large-scale problems for the reasons mentioned in Remark 3, Section 4.2.1. For example, it cannot be applied to solve the problem in Figure 4.2a (with the size of  $K = 267,680$ ) owing to insufficient memory. Hence, the performance of various inexact forms of blocks is evaluated in the following Sections.

#### 4.3.1.2. Comparison between inexact block $P$

Figure 4.5 compares the numerical performance (iteration count and total runtime) of inexact block diagonal preconditioner (4.21) with various inexact forms of pile block  $P$  (4.22) for the 9-piled raft problem in  $25 \times 25 \times 35$  mesh (Figure 4.2a). In this Section, the soil block  $G$  is a simple diagonal matrix in the preconditioner. As shown in the Figure 4.5, the number of iteration counts of the preconditioner increases with increasing  $E'_p/E'_s$  when the approximations of block  $P$  are not exact. This trend opposes that for the exact block diagonal preconditioner shown in Figure 4.3. The performance improves consistently with more rigorous approximation of  $P$  block; however, the convergence with SSOR( $P$ ) and ILU0( $P$ ) still appear to deteriorate when the relative difference in stiffnesses is large. In contrast, when the block  $P$  is solved directly, such convergence deterioration seems to be suppressed effectively. Figure 4.6 explains the reason for this improvement in the convergence behavior. The magnitude of the smallest eigenvalue of  $\tilde{K}$  decreases by several orders for large  $E'_p/E'_s$ , resulting an ill-conditioned matrix for the crudest approximation of  $P$  [e.g.  $\text{diag}(P)$ ]. This is consistent to the findings of Lee *et al.* (2002), who examined the eigenvalues of the standard Jacobi (SJ) preconditioned system. On the other hand, the eigenvalue profiles are almost identical for different  $E'_p/E'_s$  with Cholesky factorization of  $P$  in the preconditioner. Thus, the material ill-conditioning effect seems to have mitigated. This is different from the theoretical exact block diagonal preconditioner (4.13) where the clustering of eigenvalues increases towards

unity with increasing  $E'_p/E'_s$  (compare Figure 4.4), but is expensive in terms of both time and memory usage.

The additional RAM required by the exact  $P$  block preconditioner over the simplest standard Jacobi [ $diag(P)$ ,  $diag(G)$ ] is minor (only about 0.02% for the studied above problem, Figure 4.15). However, it is obvious that the Cholesky factorization of block  $P$  can become very expensive when the size of block  $P$  is very large. More details on the effect of size of block  $P$  on the preconditioners will be discussed later in Section 4.3.1.5. Note that solving  $P$  (a submatrix of the stiffness matrix  $K$ ) directly does not mean that we have already computed the final answer for the pile DOFs in a single iteration. We will show subsequently that solving  $P$  directly is unproductive for smaller  $E'_p/E'_s$  ratios.

It is possible that a more optimal balance between reduction in iteration count and increase in preconditioning overhead be located by conducting a more refined search between ILU0( $P$ ) and exact  $P$ , such as ILUT (Saad, 1994b) with controlled fill-ins. This Section examines the performance of ILUT( $\rho$ ,  $\tau$ ) approximation of block  $P$ , where  $\rho$  is the number of fill-ins in excess of original number of nonzeros in each row of  $L$  and  $U$ , and  $\tau$  is a dropping parameter below which the fill-ins are discarded. As shown in Table 4.1, the ILUT approximation of block  $P$  does improves the performance in comparison to ILU0 of the same. However, it still shows the effect of material stiffness contrasts on convergence, particularly when  $E'_p/E'_s$  ratio is above 1000. In addition to  $E'_p/E'_s$ , the ill-conditioning of the system is also influenced by other factors such as the size of the stiff block  $P$  as studied

in Section 4.3.1.5. For an effective mitigation of such material heterogeneity and for ILUT to be competitive with  $\hat{P}_4$  in terms of runtime, it requires a large number of fill-ins depending on the problem at hand. For this reason, Toh *et al.* (2004) noted that “*these ‘optimal’ block approximations can only be identified through numerical experiments – a luxury that practitioners can ill-afford and completely self-defeating if the goal is to solve a given problem in the shortest time*”. Moreover, allowing a large number of fill-ins makes ILUT approximation to be almost as expensive as the Cholesky factorization. Thus, the direct factorization of block  $P$  seems to be more appropriate for the preconditioner (4.21) to be effective.

#### 4.3.1.3. Comparison between inexact block $G$

After ascertaining the most appropriate form of block  $P$  from the previous Section, this Section evaluates the effect of inexact forms of block  $G$  in the preconditioner. It is possible that the best approximation for block  $P$  depends on the approximation for block  $G$ .

As shown in Figure 4.7, the number of iteration counts reduces drastically for increasingly better approximation of  $G$ . This can be attributed to the shift of the eigenvalue profiles closer to unity with increasingly better approximation of  $G$  (Figure 4.8). Increasingly better approximation of  $G$  with exact  $P$  in the preconditioner means the preconditioner is closer to exact block diagonal preconditioner (4.13). Thus, the theory of exact block diagonal preconditioner holds true again. However,  $SSOR(G)$  is found to be superior than  $ILU0(G)$  in terms of runtime (see Figure 4.7) despite its slower convergence in terms of iteration counts. This is because the SSOR approximation can exploit the Eisenstat trick (Eisenstat, 1981) for efficient

matrix-vector multiplication. SSOR approximation is also about 50% cheaper than ILU0 approximation, and is equivalent to diagonal approximation, in terms of memory requirement.

Based on above studies, it can be concluded that the preferred preconditioners are:

$$\text{SBD}_1 = \begin{bmatrix} P & 0 \\ 0 & \text{diag}(G) \end{bmatrix} \quad (4.25)$$

$$\text{SBD}_2 = \begin{bmatrix} P & 0 \\ 0 & \text{SSOR}(G) \end{bmatrix} \quad (4.26)$$

from the practical yardstick of minimizing runtime and keeping in mind the memory constraint. In almost all geotechnical problems, the size of block  $G$  is dominant because it represents soil stiffness matrix and the inexact forms  $\text{diag}(G)$  and  $\text{SSOR}(G)$  are the simplest approximations requiring the least storage among all inexact forms. Thus, we denote these inexact block diagonal preconditioners (4.25) and (4.26) as “Simplified Block Diagonal” (SBD) preconditioners.

#### **4.3.1.4. Comparison of SBD, SJ, SSOR, and ILU preconditioners**

Mroueh and Shahrour (1999) noted that the standard SSOR preconditioner is more efficient than the SJ preconditioner for soil-structure interaction problems. The application of SSOR factorization on stiffness matrix  $K$  is analogous to the SSOR factorization of  $P$  or  $G$  used in the preceding Sections. The algorithm of SSOR preconditioned PCG method combined with Eisenstat trick (Eisenstat, 1981) is given elsewhere (e.g. Chen *et al.*, 2007). Algorithm provided in Appendix C specializes the algorithm in (Chen *et al.*, 2007) for the case in which SSOR is applied to block (2,2) only and Appendix

E provides the source code. For solving the linear system (4.1), the SSOR preconditioner can be written as:

$$M_{SSOR} = (L_K + D_K/\omega)(D_K/\omega)^{-1}(L_K^T + D_K/\omega) \quad (4.27)$$

where  $L_K$  and  $D_K$  are strictly lower triangular and diagonal of  $K$  respectively.

Figure 4.9 demonstrates that the performances of SJ, SSOR, and ILU0 preconditioners degrade with increasing pile-soil stiffness ratios ( $E'_p/E'_s$ ) for the same 9-piled raft in  $25 \times 25 \times 35$  mesh. The performance of SJ degrades the most (about 10 times) in terms of runtime, while the runtime of both SSOR and ILU0 degrades by over three times, for the increase in  $E'_p/E'_s$  from 1 to 41000. Similarly, ILUT(50,  $10^{-6}$ ) on entire  $K$  can even be slower than ILU0 despite taking smaller iteration counts than the ILU0 (Table 4.2). The increase in runtime is mainly contributed by the expensive ILUT factorization (85-95% of the total runtime) for a large and ill-conditioned matrix, and to a lesser extent, contributed by the more expensive triangular solves in each preconditioning step. Thus, we will not consider ILUT preconditioners in the further discussion. In contrast, the proposed SBD preconditioners (4.25-4.26) turn out to be superior to these preconditioners as  $E'_p/E'_s$  increases (Figure 4.9) because of their stable convergence. Existing preconditioners (SJ, SSOR or ILU0) are competitive with SBDs for only lower range of stiffness ratios. The crossover points ( $E'_p/E'_s$ ) above which the SBD<sub>1</sub> and SBD<sub>2</sub> are preferable over existing preconditioners are about 400 and 30, respectively. However, these crossover points are likely to be dependent on the problem at hand and the FE mesh. The next Section discusses more on this.

#### 4.3.1.5. Effect of number of piles

As noted in Section 4.3.1.2, the cost of SBD preconditioners is affected by the dimension of block  $P$  because these preconditioners require a Cholesky factorization of block  $P$ . It is also true that the Cholesky factorization of entire stiffness matrix  $K$  is impractical for large-scale problems. Hence, in this Section, the performances of SBD, SJ, SSOR, and ILU0 preconditioners are compared for different dimensions of the block  $P$ . To achieve this, the number of piles is varied from 1 to 49 within a square raft configuration as shown in Figure 4.10. The practical importance of studying the effect of number of piles in the piled-raft foundation is to access the minimum number of piles required to achieve the desired performance of piled-raft. However, from the computational point of view, the number of piles present can influence the computational performance of the iterative solvers. A single pile is representative of a design scenario with limited structural/stiff material (block  $P$ ). The 49-pile group is representative of a design scenario with more extensive structural/stiff material. A raft thickness of 3 m is first considered. Subsequently, it is increased until a maximum  $P$  block is achieved. The maximum  $P$  block is controlled by the available RAM required for Cholesky factorization, which is 2GB in this study. Hence, the dimension of block  $P$  varies from 668 (0.25% of  $N$ ) for a single pile to 97,178 (36.30% of  $N$ ) for 49-piled raft with 5 m raft thickness, where the total dimension of the linear system ( $N$ ) is held constant to 267,680 [Equation (4.10)] for the sake of comparison. The same finite element mesh (Figure 4.2a) is used for all the cases and details of each problem are presented in Table 4.3. Although in

actual practice, the mesh must increase in extent with widening the lateral directions of raft to avoid boundary effects.

Figure 4.11 shows that the number of iteration counts and the CPU times of SJ, SSOR, and ILU0 preconditioners are not only affected by  $E'_p/E'_s$  but also by the number of piles (size of the block  $P$ ) in the group. When the size of block  $P$  is minor (e.g. due to 1-pile), the large  $E'_p/E'_s$  is not a problem for SJ, SSOR, and ILU0; however, as both the size of  $P$  and  $E'_p/E'_s$  increases, the effectiveness of these preconditioners decreases significantly. For such problems, ILU0 performs even worse (or even failed to converge) than that of the simplest standard Jacobi preconditioner. However, the performance of SBD preconditioners remain stable due to the variation in both the size of block  $P$  and  $E'_p/E'_s$ . Thus, SBD preconditioners seem to have effectively mitigated the ill-conditioning effect due to relative differences in material stiffnesses.

Figures 4.12-4.14 show the savings in runtimes by the SBD preconditioners over other preconditioners vary depending on the number of pile (stiff structural) DOFs and soil-structure stiffness ratios. Depending on the problem at hand, SBD preconditioners can even be more than 10 times faster than SJ, SSOR, or ILU0 preconditioners if the  $E'_p/E'_s$  is large. SBD preconditioners seem to be more suitable when the presence of stiff structural DOFs is above 5% in total. However, they may become less effective when the finite element mesh is such that the size of stiff structural DOFs is above one-third of the total. This is because the Cholesky factorization of block  $P$  becomes more expensive to factorize when its size increases, and at the same

time, the triangular solves at each preconditioning step also become more expensive due to a denser Cholesky factor. Since many soil-structure interaction problems involve stiffness ratios larger than 1000 (quite off from the crossover line), the SBD preconditioners are likely to be effective for most practical problems.

As shown in Figure 4.15, the RAM required for SBD preconditioners is merely larger than that for SJ/SSOR preconditioners when the percentage of structural DOFs in the mesh is not very significant and it can grow up to or even larger than that of RAM for ILUT preconditioner when such a percentage is 35 or above.

#### **4.3.2. Tunneling**

The aim of this study is to investigate the effectiveness of the proposed SBD preconditioners for fast 3D analysis of tunnels. In tunneling too, the large relative difference in stiffnesses of liner and soil can deteriorate the solver performance. The problem considered here is the settlement trough analysis of a circular NATM (New Austrian Tunneling Method) tunnel. See Möller (2006) and Vermeer et al. (2001) for more details. A block of  $100 \times 55 \times 28$  was divided into 13074 20-noded brick elements, resulting 151,902 total unknown DOFs as shown in Figure 4.16. The outer boundary conditions are the same as were explained for the piled-raft foundation problem. The excavation and installation of 0.3m thick liner was simulated according to step-by-step procedure as shown in the same Figure. Thus, there is no liner in the first excavation (i.e. no block  $P$ ) and the size of block  $P$  increases in each excavation and reaches up to 13,459 (about 10% of the size of  $K$ ) at the end of 40 excavation steps. Each excavation step consists of 40 increments to account

for nonlinear soil behavior. To facilitate the FE analysis of step-by-step tunneling, a geotechnical software GeoFEA (2006) is used in the present study. The preconditioners (SJ, SSOR,  $SBD_1$  and  $SBD_2$ ) are implemented as user defined solvers. Details of implementation are explained in Chapter 6.

Soil is modeled as Mohr-Coulomb material and the liner as linear elastic. The material parameters considered here are the same as Möller's (2006), except the dilation angle. Here, the dilation angle is the same as the angle of friction to keep the stiffness matrix  $K$  to be symmetric. Table 4.4 summarizes the parameters used in the analysis, which gives the liner-soil stiffness ratio of 476 for the studied problem.

As expected, SJ and SSOR preconditioners require increasingly larger number of iteration counts in each step as shown in Figure 4.17. As explained for piled-raft problem, this can be attributed to the increase in liner (stiff structural) DOFs in each step. SBD preconditioners, on the other hand, consistently converged at small iteration counts at excavation step (see a flattened linear trend of iteration counts as well as CPU times by SBD preconditioners). This again shows the effective mitigation of material ill-conditioning, which ultimately led SBD preconditioners to be about three times faster than SJ and SSOR preconditioners. A comparison of actual total CPU times for the tunnel analysis and that manipulated from Figures 12 and 13 (point T) shows a good agreement between  $SBD_1$ /SJ and  $SBD_2$ /SSOR respectively (Table 4.5), whereas a large variation in other CPU ratios. This variation can be attributed to the unduly slower performance of SSOR than SJ preconditioner, contrary from the piled-raft problem. This has also affected the performance of  $SBD_2$  as it uses SSOR approximation for the soil block [see

Equation (4.26)]. One possible reason for this difference may be due to the differences in sparsity pattern of  $K$  as shown in Figure 4.18. Piled-raft problem was solved using the in-house FORTRAN code in which node numbering is sequential in  $x$ - $z$  plane (e.g. Smith and Griffiths, 1997, 2004), whereas GeoFEA has its own way of nodal numbering. Considering these results, we can say that the Figures 4.12-4.14 can be used for a general guidance for an estimated relative savings in runtimes by SBD preconditioners for modeling soil-structure interaction problems when prevailing conditions are known.

Figure 4.19 shows the computed steady-state surface settlement profile after 40 excavation steps which closely matches with that of Möller (2006). A peculiar large settlement near the left mesh boundary was observed by Möller and was attributed to the lack of immediate support by a tunnel lining for the first excavation. However, the present study using GeoFEA did not observe such behavior.

#### **4.4. Conclusion**

The ill-conditioning of linear systems due to large relative differences in stiffnesses of the materials such as those in soil-structure interaction problems was investigated in this study. For such systems, the performance of SJ, SSOR or ILU preconditioners degrades with increasing relative differences in stiffnesses and the number of elements of stiff materials (e.g. structural elements) in the mesh. A block diagonal preconditioner was shown to exploit such relative differences in stiffnesses to an attractive eigenvalue clustering of the preconditioned system. Various inexact block diagonal preconditioners were systematically studied for the practical usage. Finally, two simplified

block diagonal (SBD) preconditioners were proposed that were shown to be effective in mitigating such material ill-conditionings. Studied examples showed that such mitigation offers a significant improvement in the performance for modeling large-scale soil-structure interaction problems.

The key observations are summarized as follows:

1. A large soil-structure stiffness ratio was shown to be an advantage only for linear systems preconditioned by the theoretical exact block diagonal preconditioner. However, this exact form is not practical in terms of runtime and memory usage.
2. Numerical results showed that inexact block diagonal preconditioners are much cheaper and faster than the theoretical exact block diagonal preconditioner because the large preconditioning costs for the latter will overwhelm the saving accrued from massive reduction in iteration count. Inexact forms are always less memory intensive than the exact form.
3. It was found that the inexact block diagonal preconditioners with Cholesky factorization of block  $P$  effectively mitigated the degradation in performance due to increasing soil-structure stiffness ratios regardless of the approximation of soil block  $G$ .
4. From the runtime as well as memory point of view, the diagonal preconditioners with exact  $P$  combined with diagonal or SSOR approximation of  $G$  were found to be the most practical. These two preconditioners were termed as SBD preconditioners. However,  $\text{SBD}_2$  (4.26) may or may not be faster than  $\text{SBD}_1$  (4.25) depending on the problem at hand and the ordering of variables.

5. SJ, SSOR, or ILU preconditioners are not only affected by large soil-structure stiffness ratios, but also by the size of block  $P$ . In contrast, SBD preconditioners are stable and convergence consistently for a variable size of the block  $P$ . Some charts have been proposed that may be useful for engineers for a general guidance on an estimated saving in runtimes by SBD preconditioners over others.

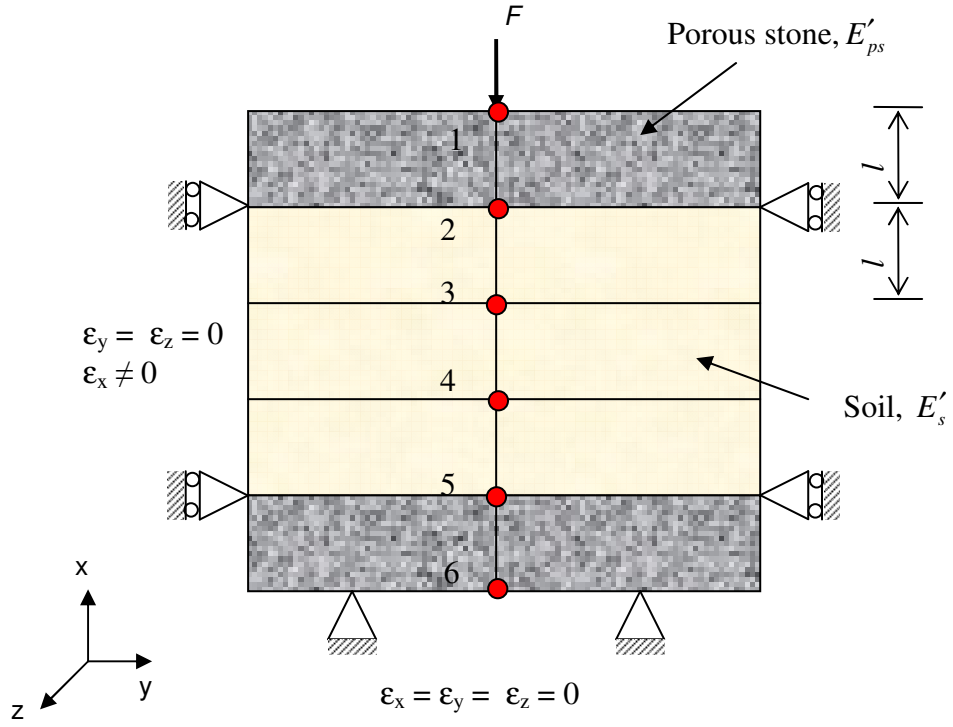
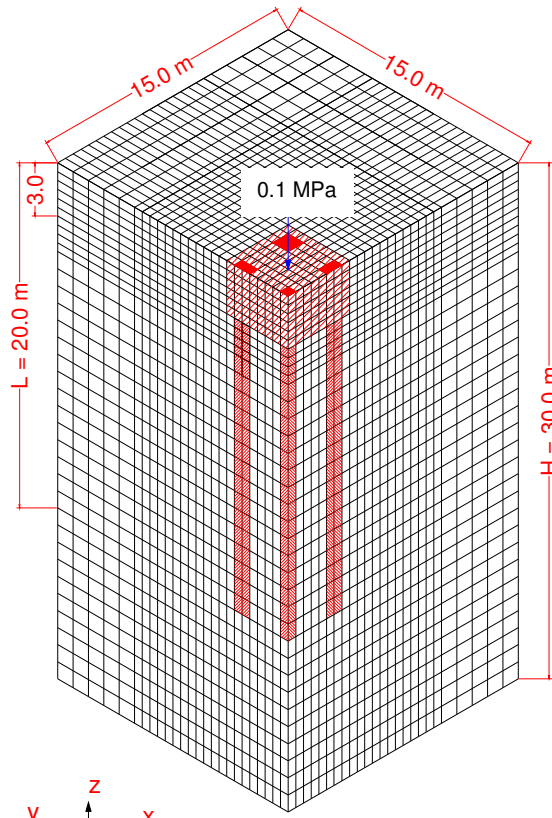


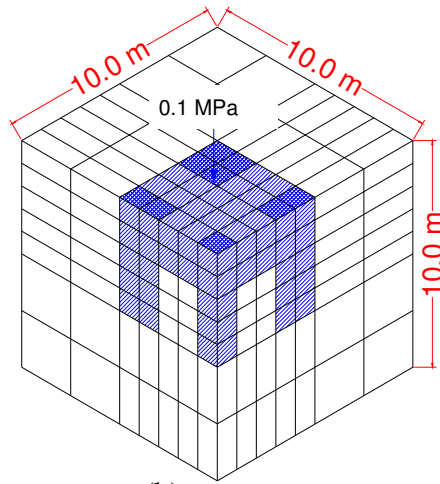
Figure 4.1. One-dimensional FE discretization of oedometer test set up to illustrate the effect of different materials in the formulation of FE stiffness matrix. The dots and numbers besides them are finite element nodes and node numbers,  $F$  is the applied load,  $l$  is the element size,  $E'_{ps}$  and  $E'_s$  are the effective Young's moduli of porous stone and soil, respectively.



(a)

No. of elements	21,875
No. of nodes	94,796
Pile DOFs ( $m$ )	12,767
Soil DOFs ( $n$ )	254,913
Total DOFs ( $N$ )	267,680

Raft	
Thickness	3 m
Overhang	0.5 m



(b)

$m$	1,632
$n$	2,764
$N$	4,396

Figure 4.2. Three-dimensional FE discretization of a typical 9-piled raft foundation (quadrant symmetric): (a) a realistic problem discretized into  $25 \times 25 \times 35$  mesh; (b) a model problem discretized into  $7 \times 7 \times 7$  mesh to illustrate the spectral properties of the preconditioned system.

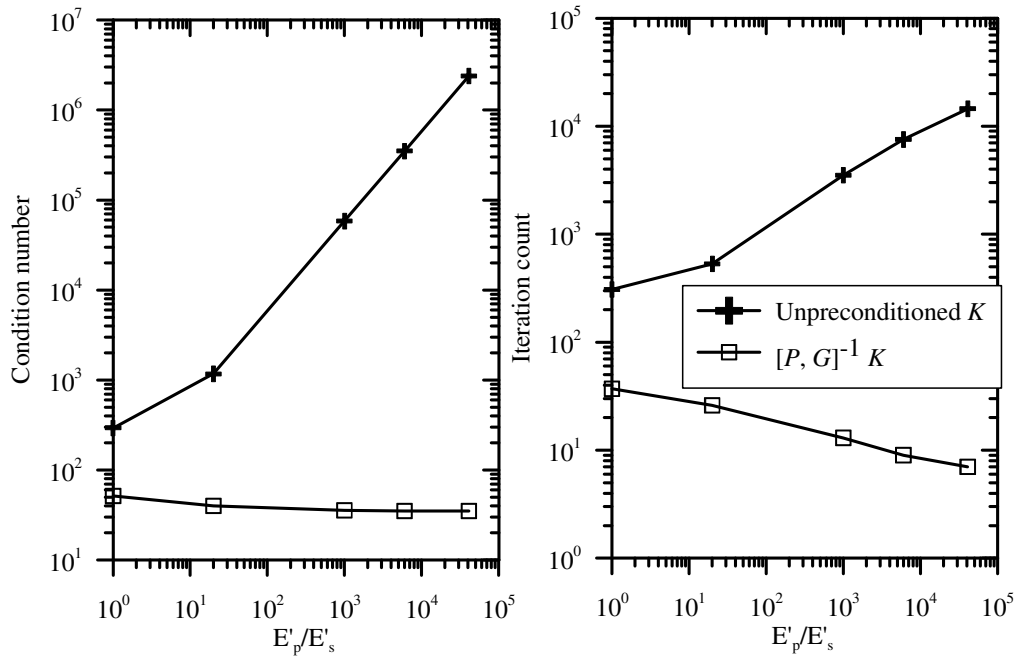


Figure 4.3.  $7 \times 7 \times 7$  mesh: Condition number and iteration count of unpreconditioned and theoretical block diagonal preconditioned stiffness matrix  $K$  for varying pile-soil stiffness ratios.

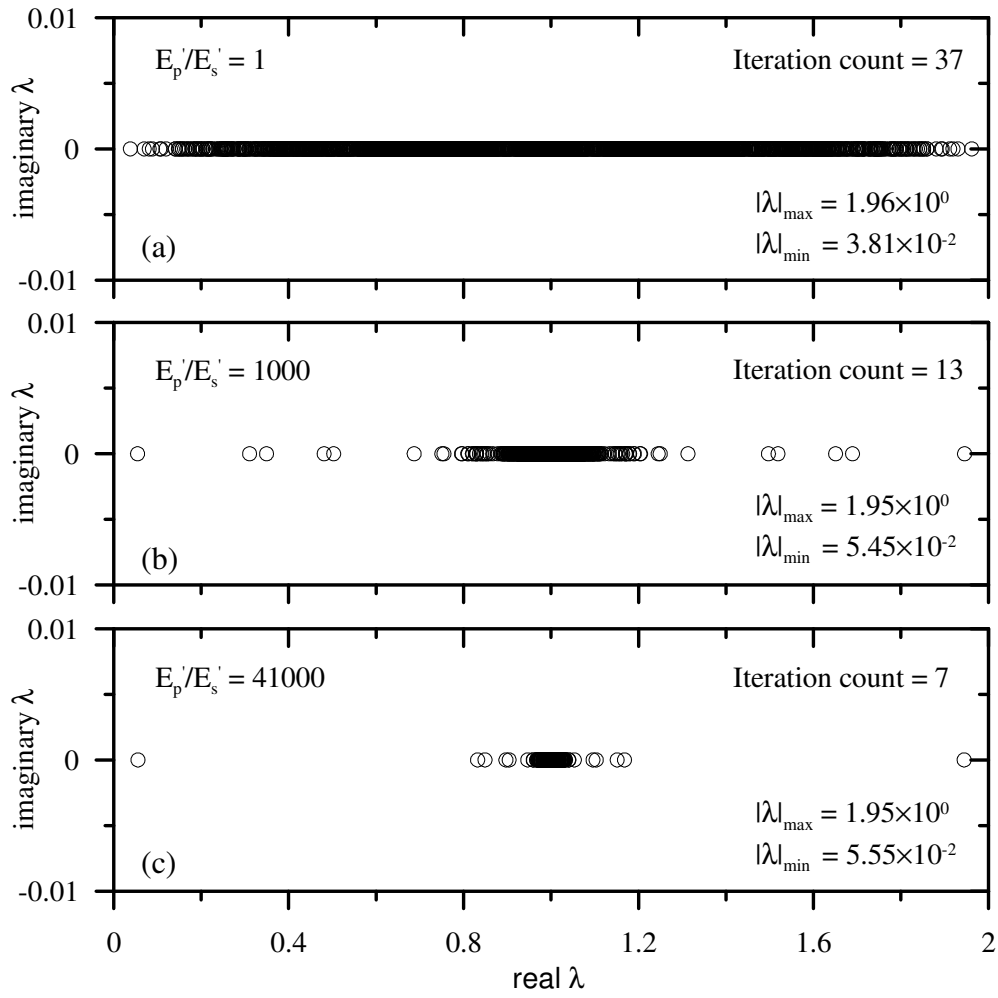


Figure 4.4.  $7 \times 7 \times 7$  mesh: Eigenvalue distribution of theoretical exact block diagonal preconditioned system (4.14) at different pile-soil stiffness ratios: (a)  $E_p'/E_s' = 1$  (fictitious pile); (b)  $E_p'/E_s' = 1000$ ; and (c)  $E_p'/E_s' = 41000$ .

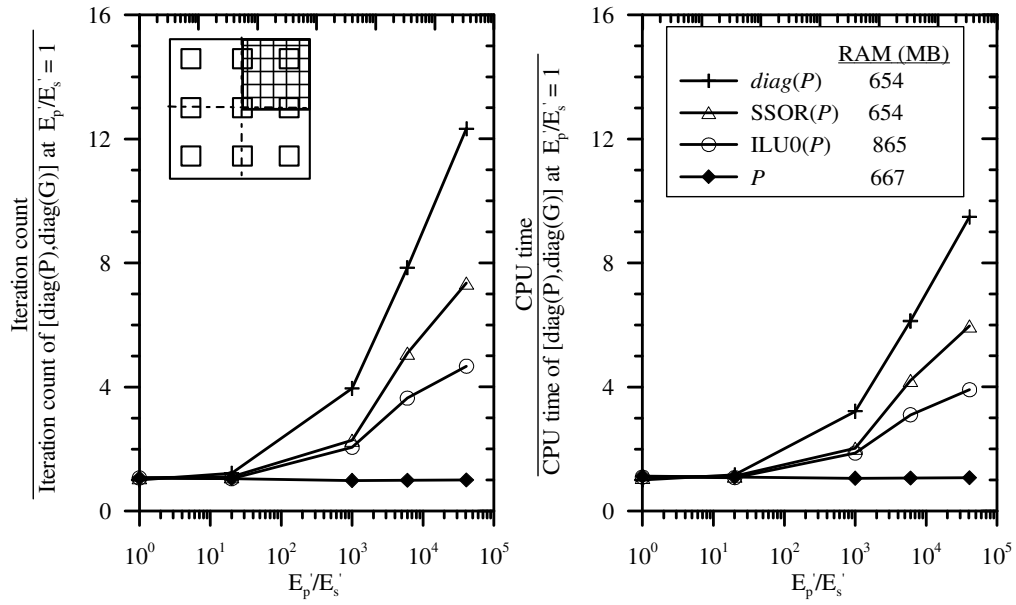


Figure 4.5. 25×25×35 mesh: Iteration count and total CPU time of inexact block diagonal preconditioner (4.21) for various inexact forms of block  $P$  with diagonal approximation of soil block  $G$  for a 9-piled raft.

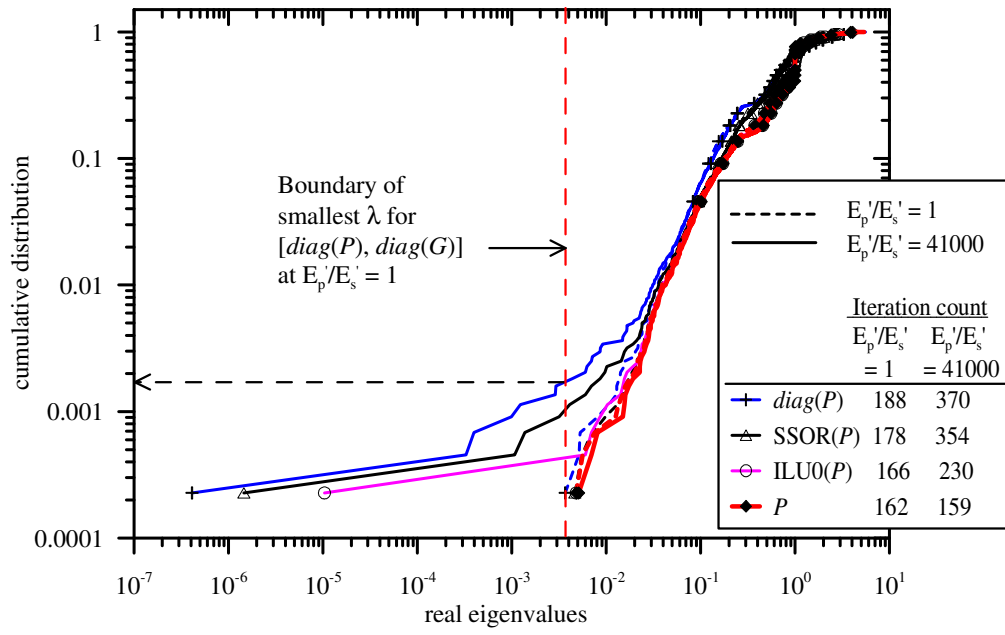


Figure 4.6.  $7 \times 7 \times 7$  mesh: Cumulative distribution of eigenvalues of the preconditioned system with inexact block diagonal preconditioner (4.21) for different inexact forms of block  $P$  and diagonal of block  $G$  at two different stiffness ratios.

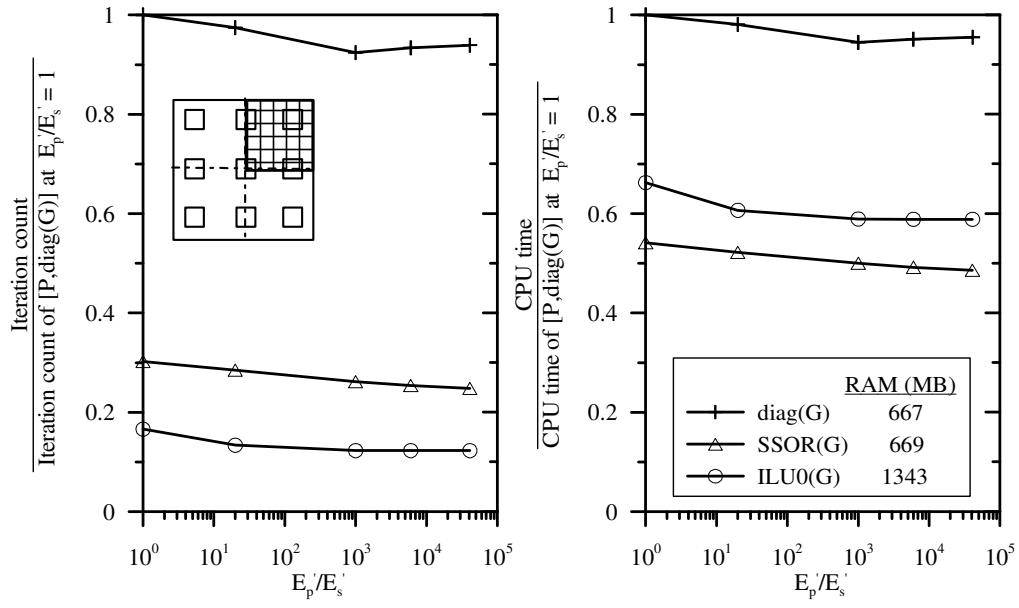


Figure 4.7. 25×25×35 mesh: Performance of different inexact forms of block  $G$  with Cholesky factorization of block  $P$  in the block diagonal preconditioner.

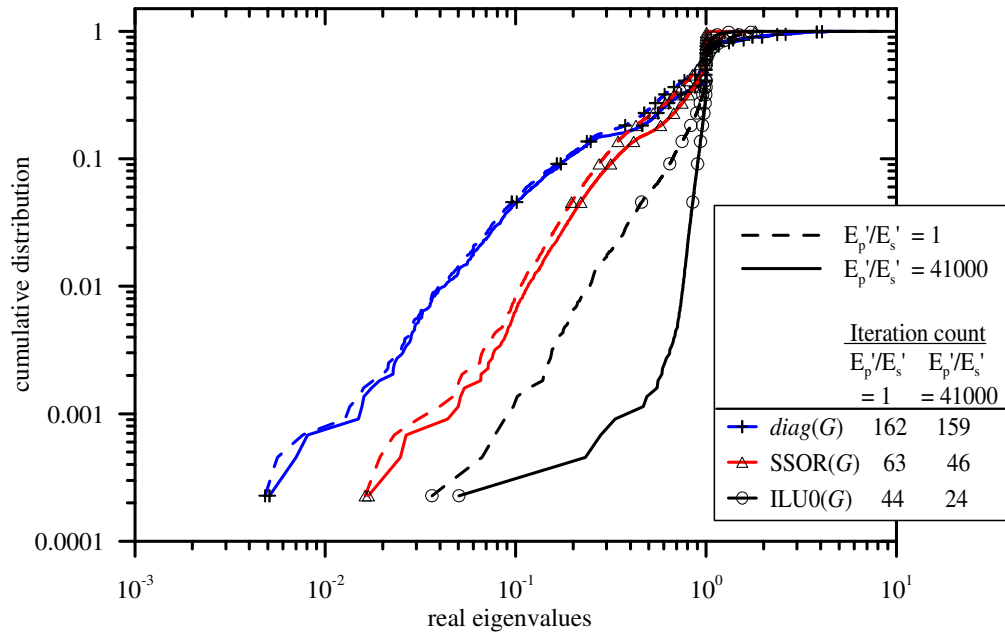


Figure 4.8.  $7 \times 7 \times 7$  mesh: Cumulative distribution of eigenvalues of the preconditioned system for different inexact forms of block  $G$  with Cholesky factorization of block  $P$  in the block diagonal preconditioner.

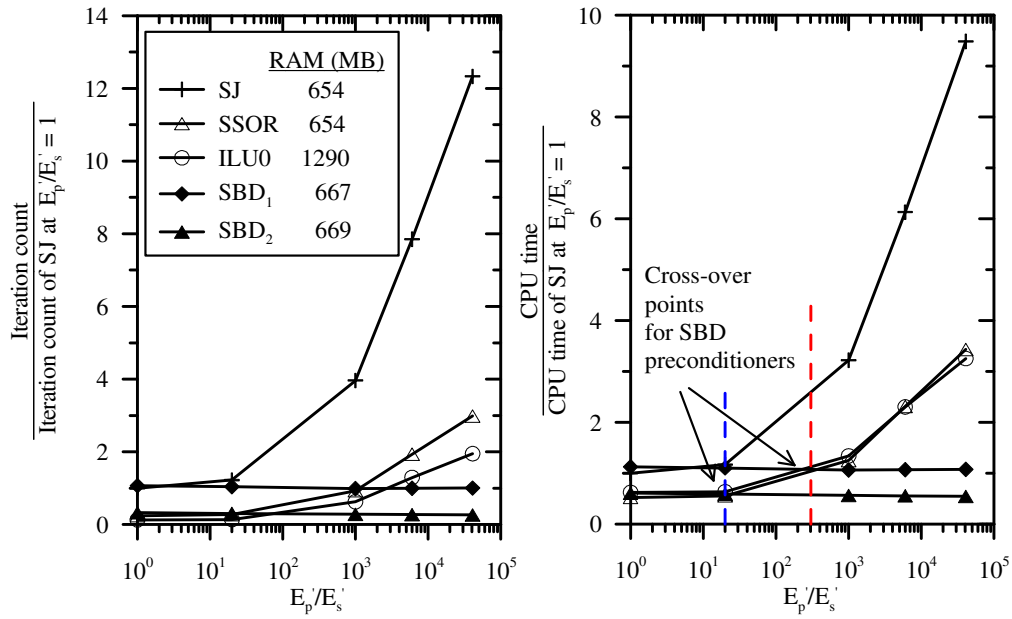


Figure 4.9. 25×25×35 mesh: Comparison of SBD preconditioners with other preconditioners for a 9-piled raft in a range of  $E_p/E_s$ . Preconditioners are: SJ = standard Jacobi; SSOR = symmetric successive over relaxation [Equation (4.27)]; ILU0 = Incomplete LU factorization preconditioner with zero fill-ins; SBD = simplified block diagonal preconditioners [Equations (4.25 and 4.26)].

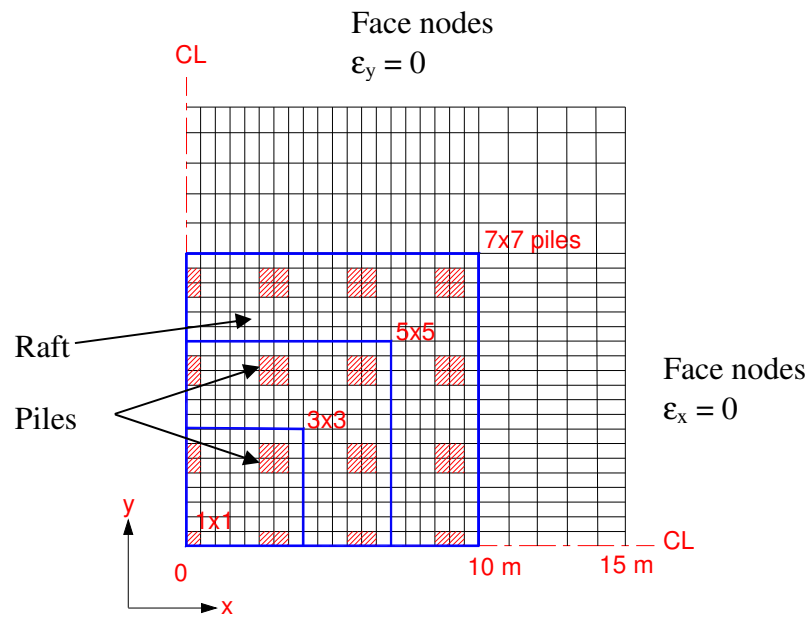


Figure 4.10. 25×25×35 mesh: Layout of piles in the piled-raft foundation (quadrant symmetric).

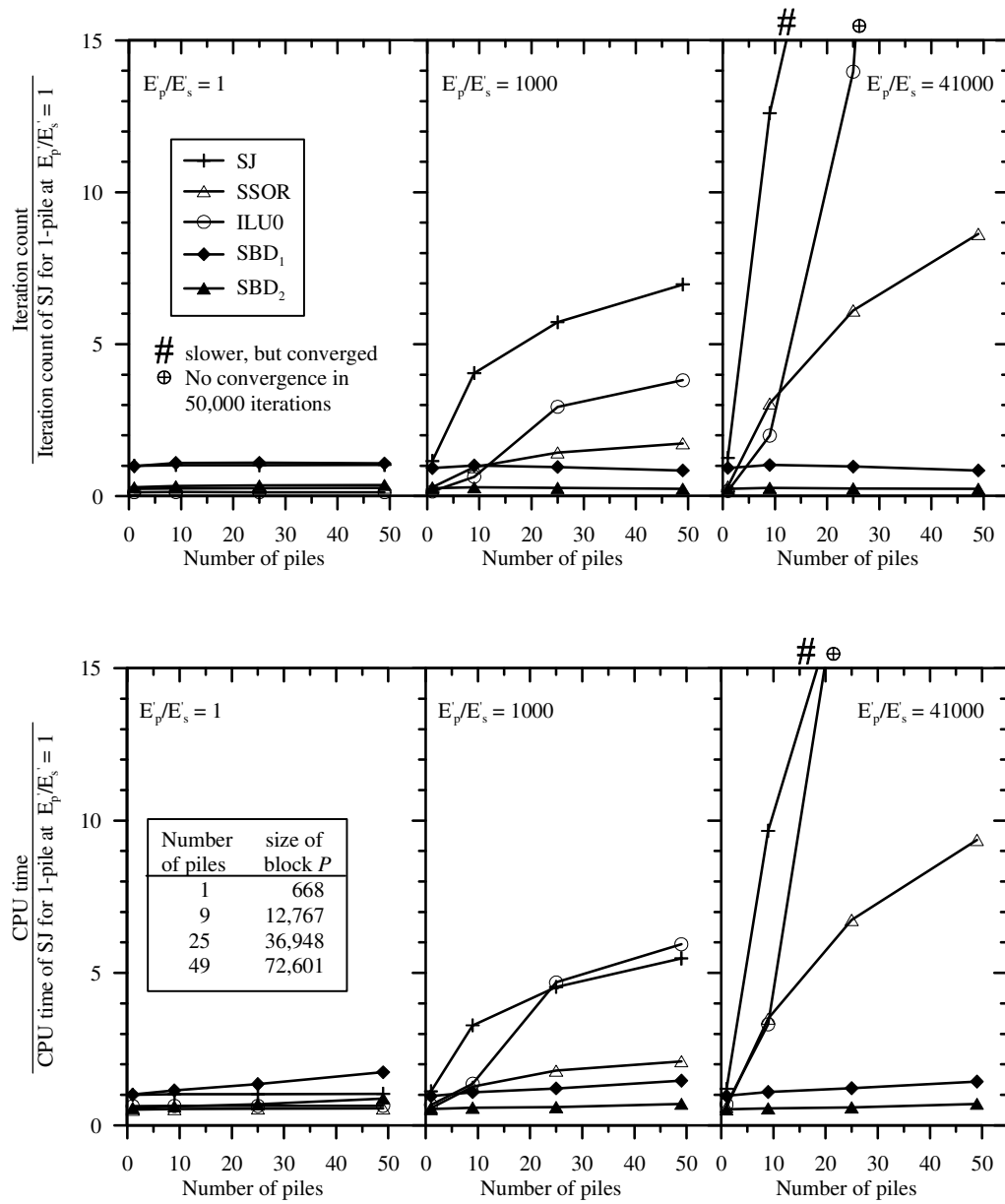


Figure 4.11. 25×25×35 mesh: Effect of size of stiff block  $P$  (e.g. due to variation in number of piles, raft thickness = 3 m, in the piled-raft problem) in the performance of preconditioners at different stiffness-ratios.

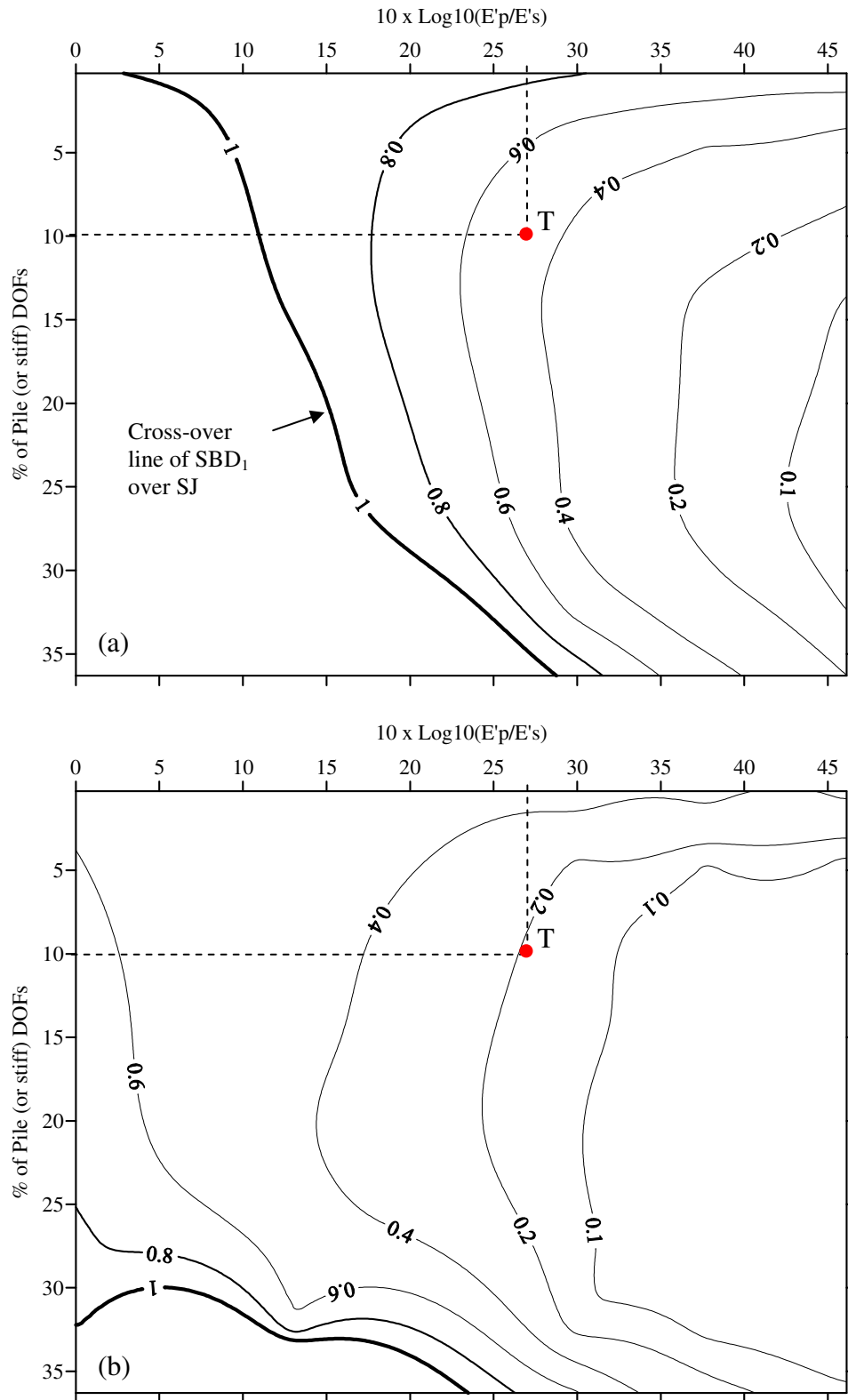


Figure 4.12. CPU time of SBD preconditioners for a range of stiff DOFs and soil-structure stiffness ratios: (a) SBD<sub>1</sub> versus SJ (b) SBD<sub>2</sub> versus SJ.

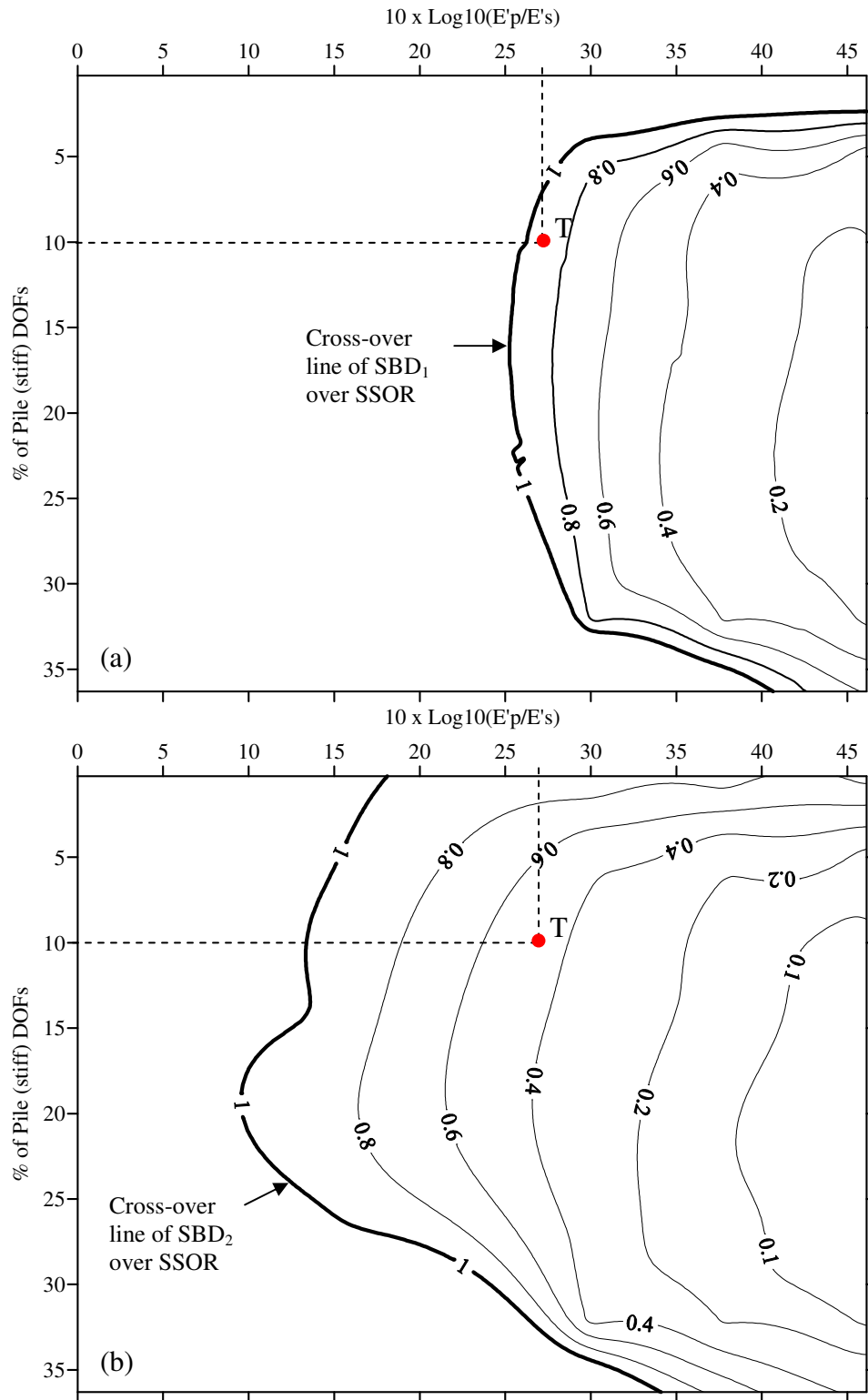


Figure 4.13. CPU time of SBD preconditioners for a range of stiff DOFs and soil-structure stiffness ratios: (a) SBD<sub>1</sub> versus SSOR (b) SBD<sub>2</sub> versus SSOR.

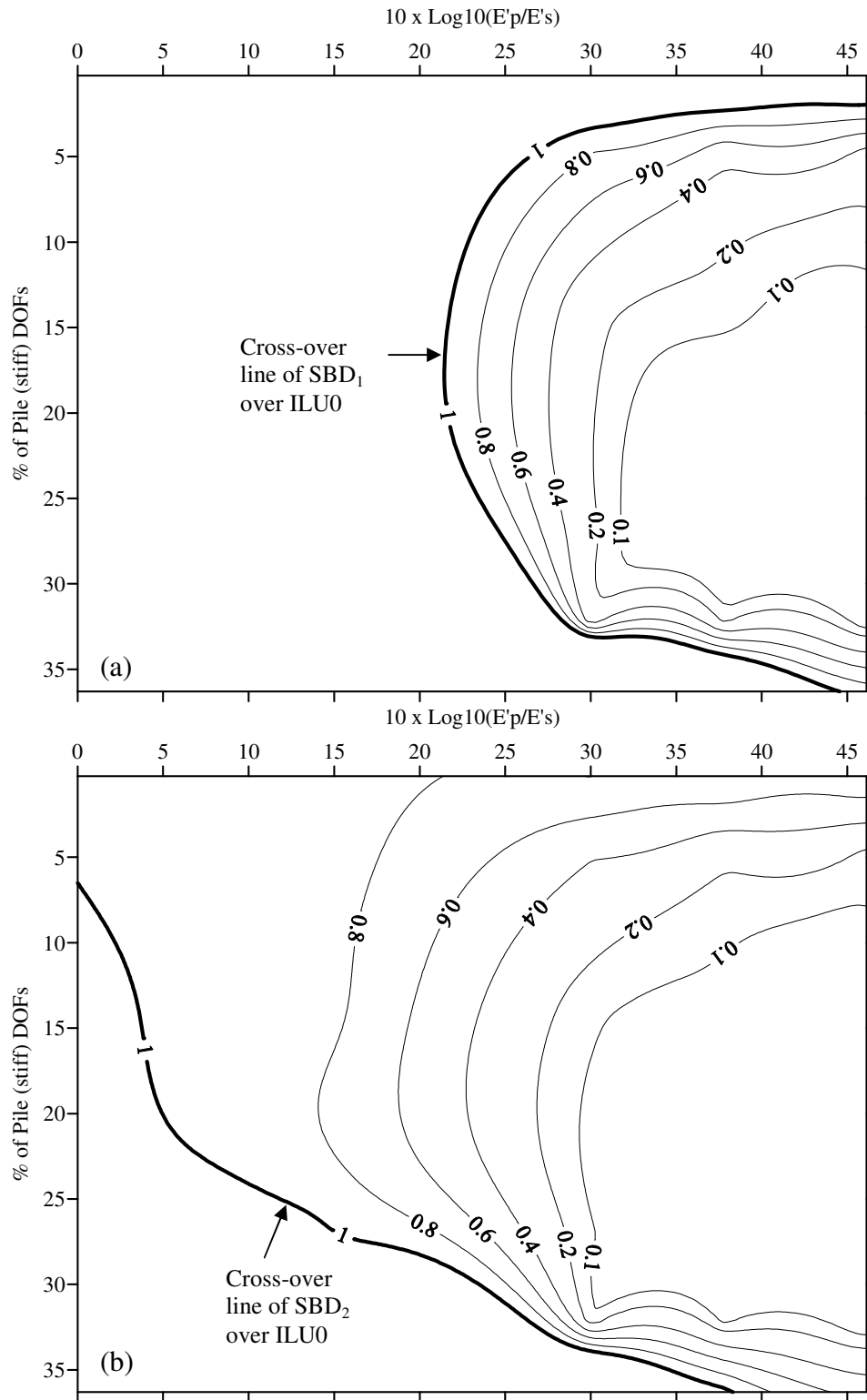


Figure 4.14. CPU time of SBD preconditioners for a range of stiff DOFs and soil-structure stiffness ratios: (a) SBD<sub>1</sub> versus ILU0 (b) SBD<sub>2</sub> versus ILU0.

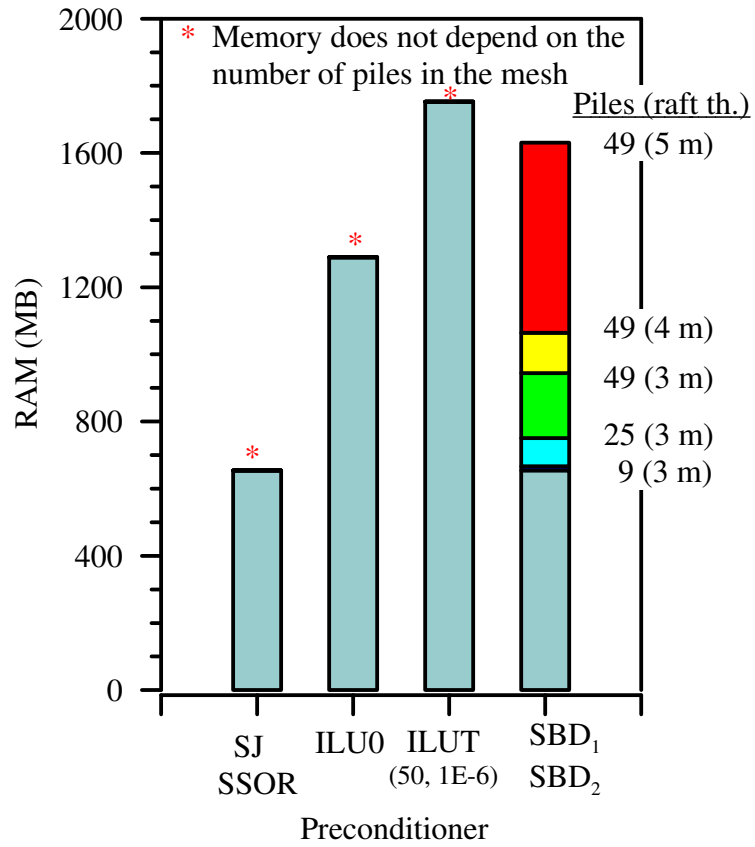


Figure 4.15. RAM consumed with different preconditioners for the same size ( $267,680 \times 267,680$ ) of the stiffness matrix  $K$ .

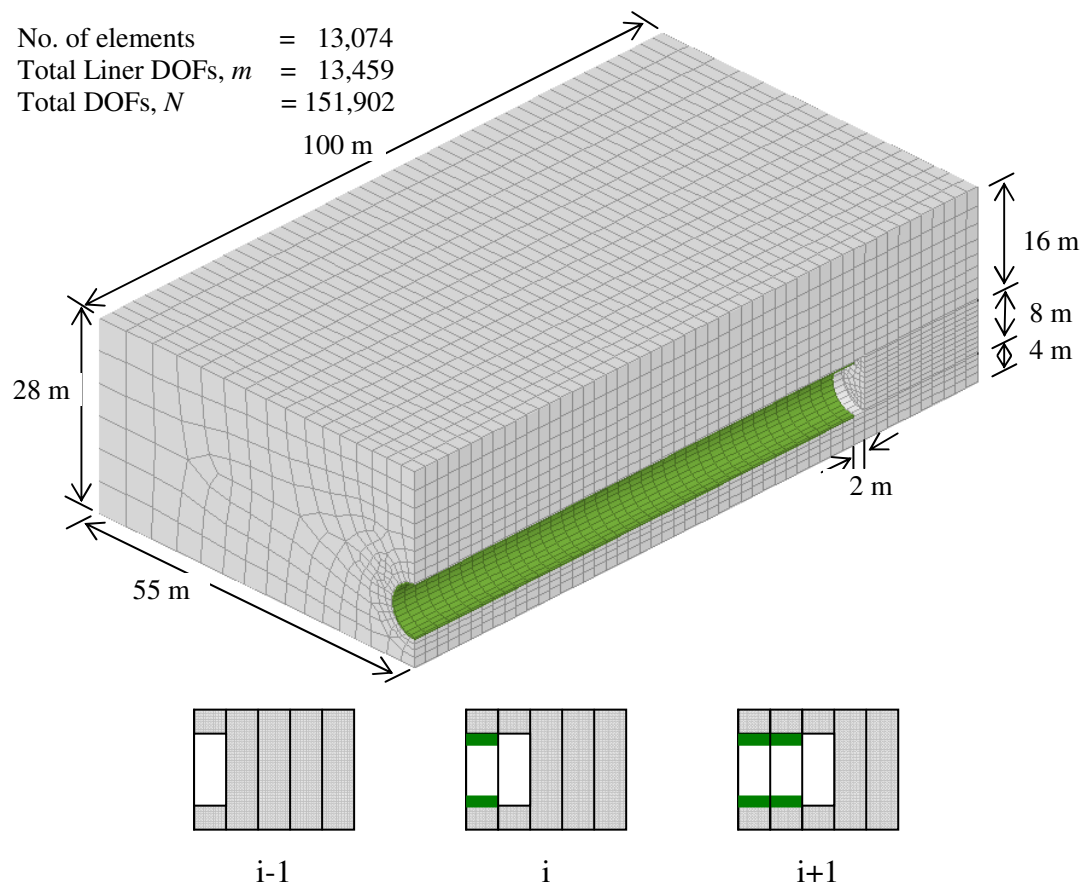


Figure 4.16. Finite element mesh and step-by-step installation of liner in tunneling.

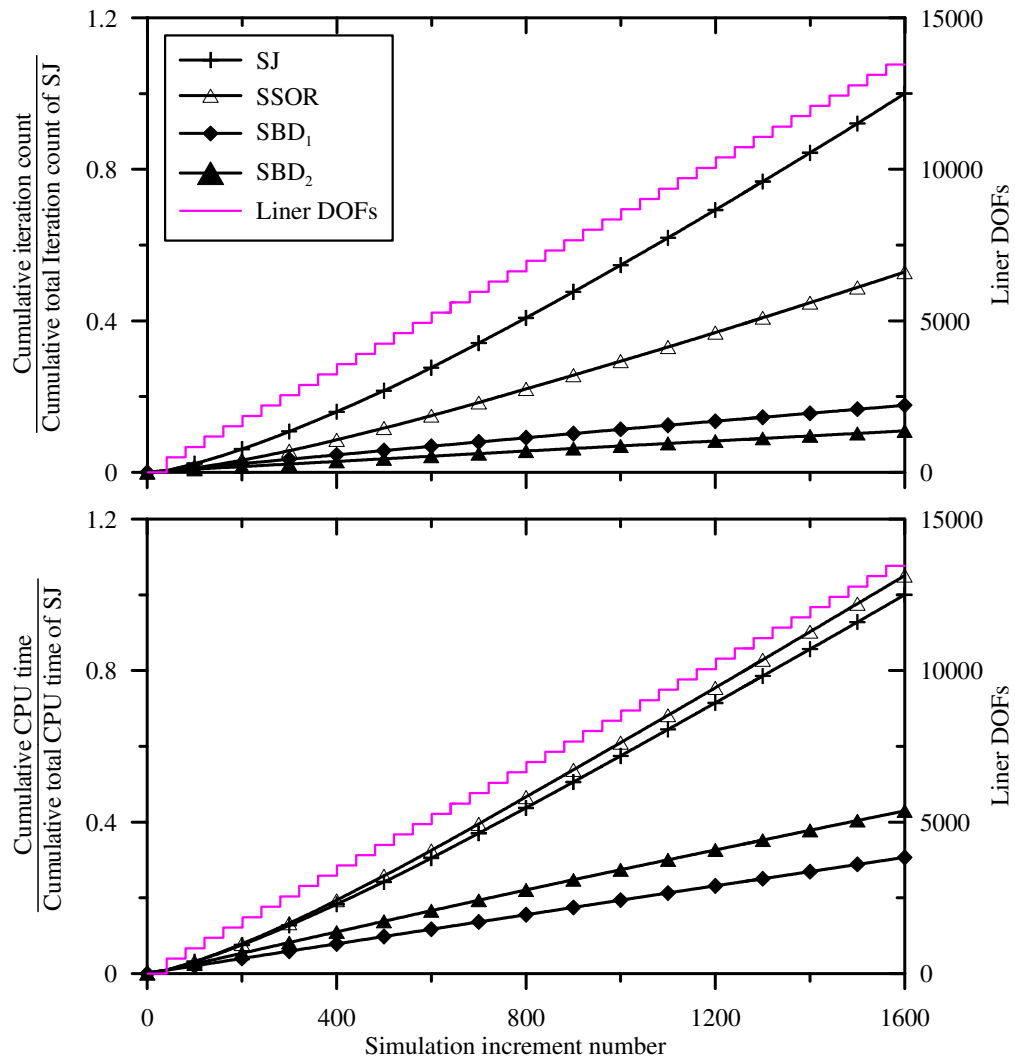


Figure 4.17. Comparison of iteration count and CPU time of preconditioners for the tunneling example.

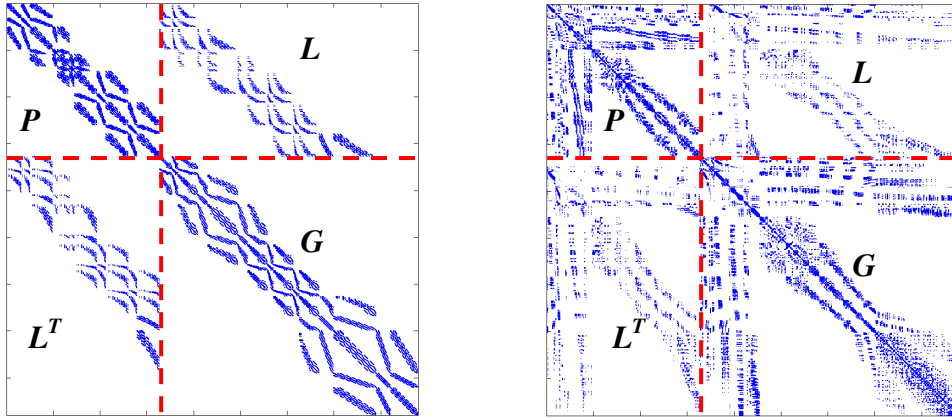


Figure 4.18.  $7 \times 7 \times 7$  mesh: Sparsity pattern of  $2 \times 2$  block structured  $K$ . (a) Sequential nodal numbering of nodes in x-z plane according to Smith and Griffiths (1997; 2004); and (b) Automatic nodal numbering in GeoFEA.

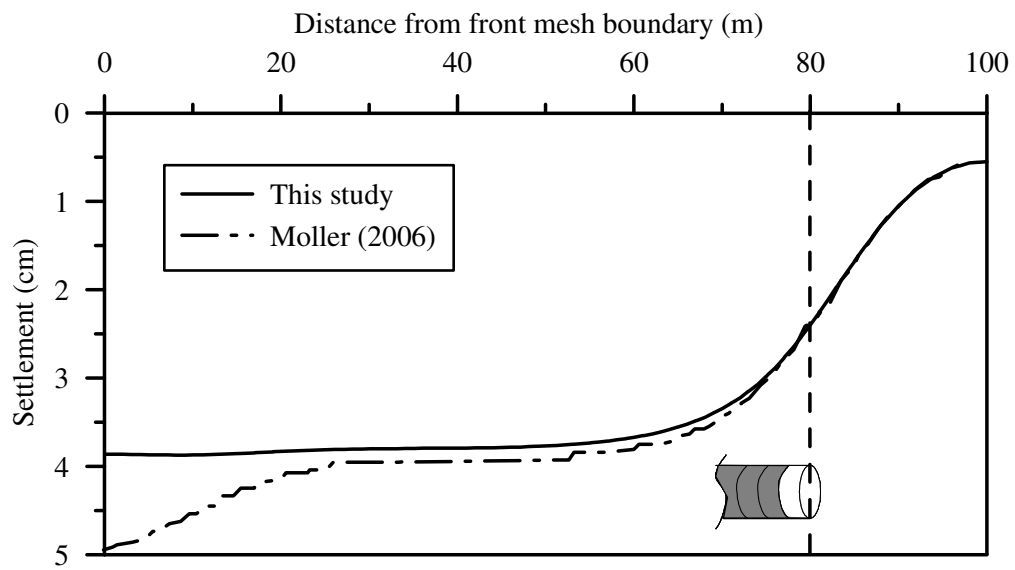


Figure 4.19. Surface settlement profile after 40 steps of excavation.

Table 4.1. 25×25×35 mesh: Effect of different approximations of the block  $P$  with diagonal approximation of block  $G$  in the preconditioner (4.21) for a 9-piled raft problem.

$E'_p/E'_s$	Approximation of block $P$	Iteration count	Total CPU time (s)	Total CPU time ratio (%)	$nnzl(P)$ or $nnzu(P)$
1	$\hat{P}_1 = \text{diag}$	694	174.05	<b>88.91</b>	
	$\hat{P}_2 = \text{SSOR}$	747	190.53	97.33	
	$\hat{P}_3 = \text{ILU0}$	741	190.78	97.46	696,532
	ILUT(50,1E-6)	740	197.59	100.94	1,294,071
	ILUT(100,1E-6)	740	201.81	103.10	1,802,023
	$\hat{P}_4 = \text{Cholesky LL}^T$	741	195.75	100.00	2,366,010
20	$\hat{P}_1 = \text{diag}$	849	203.39	105.92	
	$\hat{P}_2 = \text{SSOR}$	772	195.55	101.83	
	$\hat{P}_3 = \text{ILU0}$	725	187.80	<b>97.80</b>	699,400
	ILUT(50,1E-6)	720	193.94	100.99	1,306,927
	ILUT(100,1E-6)	720	198.05	103.13	1,821,928
	$\hat{P}_4 = \text{Cholesky LL}^T$	722	192.03	100.00	2,382,766
1000	$\hat{P}_1 = \text{diag}$	2749	560.42	303.01	
	$\hat{P}_2 = \text{SSOR}$	1584	353.53	191.15	
	$\hat{P}_3 = \text{ILU0}$	1427	325.31	175.89	699,355
	ILUT(50,1E-6)	713	192.33	103.99	1,308,782
	ILUT(100,1E-6)	684	190.50	103.00	1,825,319
	$\hat{P}_4 = \text{Cholesky LL}^T$	685	184.95	<b>100.00</b>	2,535,800
6000	$\hat{P}_1 = \text{diag}$	5446	1021.59	548.68	
	$\hat{P}_2 = \text{SSOR}$	3532	732.98	393.67	
	$\hat{P}_3 = \text{ILU0}$	2527	540.45	290.27	699,759
	ILUT(50,1E-6)	1691	389.70	209.30	1,309,820
	ILUT(100,1E-6)	972	249.55	134.03	1,810,974
	$\hat{P}_4 = \text{Cholesky LL}^T$	692	186.19	<b>100.00</b>	2,444,136
41000	$\hat{P}_1 = \text{diag}$	8557	1651.08	883.17	
	$\hat{P}_2 = \text{SSOR}$	5103	1039.17	555.85	
	$\hat{P}_3 = \text{ILU0}$	3243	680.86	364.19	699,353
	ILUT(50,1E-6)	2180	488.28	261.18	1,299,575
	ILUT(100,1E-6)	1086	272.61	145.82	1,785,517
	$\hat{P}_4 = \text{Cholesky LL}^T$	696	186.95	<b>100.00</b>	2,427,441

Note: The ILU subroutines store both the upper and lower triangular factors of a matrix. The  $nnzu$  is the number of nonzeros of extracted symmetric upper triangular factor ( $R$ ) for preconditioning. Similarly,  $nnzl$  is the number of nonzeros of symmetric lower triangular factor ( $R^T$ ) from Cholesky subroutines.

Table 4.2. 25×25×35 mesh: Performance of ILU factorization preconditioners on entire  $K$  for a 9-piled raft problem (size of  $K = 267,680 \times 267,680$ ).

$E'_p/E'_s$	Preconditioner	Iteration count	Total CPU time (s)	$nnzu(K)$
1	ILU0	90	108.62	21,901,222
	ILUT(50,1E-6)	38	736.28	35,280,731
20	ILU0	95	110.61	21,904,039
	ILUT(50,1E-6)	37	726.62	35,283,548
1000	ILU0	434	233.06	21,904,032
	ILUT(50,1E-6)	49	717.80	35,283,534
6000	ILU0	897	400.83	21,904,324
	ILUT(50,1E-6)	116	746.48	35,283,818
41000	ILU0	1352	566.16	21,904,016
	ILUT(50,1E-6)	234	799.97	35,283,213

Note: For the given computing configuration (2GB RAM) the memory was insufficient for ILUT(100, 1E-6), where ILUT(50, 1E-6) requires 1753 MB, about 36% more than that for ILU0.

Table 4.3. Finite element details of piled-raft foundations.

Mesh		25×25×35				
No. of elements		21,875				
Number of piles	Raft thk. (metre)	Size( $P$ ) -Pile DOFs ( $m$ )	Size( $G$ ) -Soil DOFs ( $n$ )	Size( $K$ ) -Total DOFs ( $N$ )	Number of nonzeros $nnz(K)$	$m/N$ (%)
1	-	668	267,012	267,680	43,534,492	0.25
9	3	12,767	254,913	267,680	43,534,764	4.77
25	3	36,948	230,732	267,680	43,534,612	13.80
49	3	72,601	195,079	267,680	43,534,640	27.12
49	4	86,145	181,535	267,680	43,562,744	32.18
49	5	97,178	170,502	267,680	43,565,156	36.30

Table 4.4. Material properties of NATM tunnel.

Parameter, symbol, and unit	Soil	Liner
Material model	Mohr-Coulomb	Liner elastic
Effective Young's modulus, $E'$ , MN/m <sup>2</sup>	42	20,000
Effective Poisson's ratio, $\nu'$	0.25	0.15
Effective cohesion, $c'$ , kN/m <sup>2</sup>	20	-
Effective angle of friction, $\phi'$ , degree	20	-
$K_0$	$1 - \sin \phi'$	-
Bulk unit weight, $\gamma_{\text{bulk}}$ , kN/m <sup>3</sup>	20	24
Thickness, m	28	0.30

Table 4.5. Comparison of total CPU times for tunnel construction

Preconditioners	Ratio of CPU times	
	Actual	From Figures 4.12-4.13
SBD <sub>1</sub> vs. SJ	0.31	0.47
SBD <sub>2</sub> vs. SJ	0.43	0.19
SBD <sub>1</sub> vs. SSOR	0.29	0.92
SBD <sub>2</sub> vs. SSOR	0.41	0.47

*(blank)*

## Chapter 5

---

# BLOCK DIAGONAL PRECONDITIONERS FOR BIOT'S CONSOLIDATION EQUATIONS

### 5.1. Introduction

The finite element simulation of Biot's consolidation equations results in a symmetric but indefinite linear system (2.1). Because of low permeability of soils in consolidation analysis, the resulting linear systems are usually ill-conditioned (Chan *et al.*, 2001; Ferronato *et al.*, 2001; Lee *et al.*, 2002), requiring significant computational efforts for the solution of large-scale systems. In the recent years, several preconditioning strategies have been proposed to accelerate the convergence of such systems (see Section 2.2). One main disadvantage of incomplete factorization (Section 2.2.3) and block constrained preconditioners (Sections 2.2.4.1-2.2.4.3) is that their performance (iteration count and CPU time) depends on user-specified *ad hoc* parameters which are problem dependent. For example, the incomplete factorization preconditioners are often rather unstable and may fail to converge for some common problems (e.g. Phoon *et al.*, 2008; Ferronato *et al.*, 2009). Secondly, these preconditioners require a considerably more computer memory than a diagonal preconditioner. In contrast, diagonal preconditioners are simple in

formulation, easy to implement, and require relatively modest computer memory for storage. These preconditioners are more suitable for a desktop PC environment; for example, GJ (Section 2.2.1.3) or MSSOR preconditioner (Section 2.2.2.1). However, the performance of these preconditioners was found to deteriorate when the relative differences in stiffness of materials are large (Chen *et al.*, 2007). Here, the term ‘relative difference in stiffness’ of materials, refers to the ratio of Young’s moduli of involved materials in the consideration. In geotechnical engineering, materials with significant stiffness contrast are commonly encountered for soil-structure interaction problems (pile foundation, tunneling, excavation, etc.) or for problems involving soil and rock materials.

Similar adverse effect of material stiffness contrasts on standard Jacobi (SJ) and SSOR preconditioners was also observed for the analysis of drained boundary value problems (Mroueh and Shahrour, 1999; Lee *et al.*, 2002). For such problems, some simplified block diagonal (SBD) preconditioners have been proposed in Chapter 4 to mitigate such adverse effects. However, the large stiffness contrasts coupled with low permeable materials are likely to produce an even more severe ill-conditioned system in the consolidation analysis and demands a thorough study. Fundamentally, the coefficient matrix in the consolidation analysis is different (indefinite) from that of a drained/undrained analysis (positive definite). Thus, both a different iterative solver and a different preconditioning strategy are necessary. The objective of this study is to investigate a block diagonal preconditioner that mitigates such effects and yet remains practical for large-scale problems. The symmetric quasi-minimal residual (SQMR) is selected as an iterative solver (see Section

2.1). Numerical results are evaluated based on three-dimensional Biot's consolidation analyses of a piled-raft foundation and a tunneling problem.

As discussed in Chapter 4, a pile foundation can exhibit an extremely large stiffness contrast between the pile and the surrounding soil (up to an order of  $10^5$  between steel piles and soft clay). A common practical problem is the settlement of soft clay in the vicinity of piles. This settlement of the soft clay introduces negative skin friction (NSF) on the piles (e.g. Fellenius, 1984). The clay surrounding the pile may settle due to reconsolidation after pile driving, ground water lowering, and/or surcharge loading after installation of piles. NSF (or dragload) can severely affect the structural integrity and bearing capacity of the pile foundation, and can cause additional pile head settlement (Fellenius, 2004; Kuhns, 2008; Shen, 2008). Hence, it is important to study the effect of consolidation on a piled-raft.

Similar large stiffness contrasts exist in tunneling in soft soils due to the presence of stiff liner or other structural components [e.g. steel pipe umbrella arch (Yeo *et al.*, 2009)]. In tunneling, deformation is a major design topic, as surrounding structures might be damaged by differential settlements. Hence, numerical analysis such as finite element method is often employed (e.g. Dasari *et al.*, 1996; Möller and Vermeer, 2008; Yeo *et al.*, 2009) owing to the complexity in geometry, soil stratification, and soil behavior that often encountered in the field.

## **5.2. Biot's consolidation equations and block diagonal preconditioning**

The block matrix structure resulting from the finite element discretization of coupled consolidation analysis (see Appendix B) suggests the use of block

preconditioners. Block diagonal preconditioners for the  $2 \times 2$  block indefinite matrix  $A$  (2.1) has been extensively studied in the scientific computing community (Section 2.2.4.5). Phoon *et al.* (2002) generalized the Murphy *et al.*'s (2000) block diagonal preconditioner for an indefinite  $A$  with  $C \neq 0$ , where Murphy *et al.*'s block diagonal preconditioner takes the following form:

$$M_{MURPHY} = \begin{bmatrix} K & 0 \\ 0 & S \end{bmatrix} \quad (5.1)$$

with 
$$S = C + B^T K^{-1} B. \quad (5.2)$$

where,  $S$  is the Schur complement of  $A$  ( $C = 0$  in Murphy's case). The inexpensive approximation of the generalized preconditioner of Phoon *et al.* is given by (2.10). Notice that, unlike the preconditioner (5.1), the block  $K$  is not exact in the preconditioner (2.10). Such an approximation is necessary for practical problems where the size of  $K$  is extremely large. The GJ preconditioner is also the key for the success of the MSSOR (2.12) preconditioner (e.g. see Chen *et al.*, 2006). However, the convergence of these preconditioners deteriorates when they are applied to a problem with significant stiffness contrasts such as a pile-group (Chen *et al.*, 2007). The source of such deterioration in convergence behavior can be investigated by looking closely into the individual blocks of Equation (2.1). Using a 2-noded linear 1D element the individual blocks can be written as (see Phoon *et al.*, 2002):

$$K = \frac{aD}{l} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, B = a \begin{bmatrix} -0.5 & -0.5 \\ 0.5 & 0.5 \end{bmatrix}, \text{ and } C = \frac{ak\Delta t}{\gamma_w l} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (5.3)$$

where,  $D = [E'(1-\nu')]/[(1+\nu')(1-2\nu')]$  is the constrained modulus,  $E'$  the effective stress Young's modulus,  $\nu'$  the effective stress Poisson's ratio,  $a$  the

cross-sectional area of the element,  $l$  the element size,  $k$  the permeability matrix,  $\Delta t$  the time step, and  $\gamma_w$  the unit weight of water.

Notice that the magnitude of the terms of the matrices depends primarily on  $E'$ ,  $k$ ,  $\Delta t$ , and the mesh. The block  $K$  is proportional to  $E'$  of the materials.  $E'$  (orders of magnitude) can vary from 1 MPa for a soft soil to  $10^5$  MPa for a steel. For block  $C$ , the permeability  $k$  (orders of magnitude) can vary from 1 m/s for a stone column to  $10^{-9}$  m/s for a soft clay or even lower for unsaturated soils. This significant differences in the magnitude of blocks  $K$  and  $C$ , particularly at small time step, is the major source of ill-conditioning in the consolidation analysis (Chan *et al.*, 2001; Ferronato *et al.*, 2001). Furthermore, for soil-structure interaction problems (or problems involving very stiff and very soft materials), Chapter 4 suggests that it is expedient to partition the solid stiffness matrix  $K$  into  $2 \times 2$  blocks after reordering the variables corresponding to a stiff material (such as structural components) and a soil. Following the above strategy, the coefficient matrix  $A$  will appear in a  $3 \times 3$  block form for the coupled consolidation analysis:

$$A = \begin{bmatrix} P & L & B_1 \\ L^T & G & B_2 \\ B_1^T & B_2^T & -C \end{bmatrix} \text{ with } K = \begin{bmatrix} P & L \\ L^T & G \end{bmatrix} \quad (5.4)$$

where  $P \in \Re^{m \times m}$  is the structure (or stiff material) stiffness matrix and  $G \in \Re^{n \times n}$  is the soil stiffness matrix. In general,  $\|P\| \gg \|G\| \gg \|C\|$ , where  $\|\cdot\|$  represents the norm of a matrix. The size of the  $P$  is governed by the discretization of stiff/structural elements and DOFs associated with them. In all practical soil-structure interaction problems, the size of the block  $P$  (i.e.  $m$ ) is much smaller than the size of soil block  $G$  (i.e.  $n$ ).  $L^{m \times n}$  is the soil-structure

displacement coupling matrix, the entries of which are contributed from soil-structure interface nodes only. Hence, many rows of  $L$  will be filled with zeros and the rank of  $L \leq m$ . Similarly,  $B_1$  is the coupling matrix between structural displacement and pore pressure DOFs, and  $B_2$  the coupling matrix between soil displacement and pore pressure DOFs. Note that, while the block  $B$  in (2.1) has full column rank, the individual blocks  $B_1$  and  $B_2$  in (5.4) may not have full column rank. This is because the coupling entries from the soil-structure interface nodes will only appear in block  $B_1$ , making corresponding columns in  $B_2$  to be zeros.

For the  $3 \times 3$  coefficient matrix  $A$ , Murphy *et al.*'s preconditioner (5.1) and the GJ preconditioner (2.10) will now take the following respective forms:

$$M_{MURPHY} = \begin{bmatrix} K & 0 \\ 0 & S \end{bmatrix} = \begin{bmatrix} P & L & 0 \\ L^T & G & 0 \\ 0 & 0 & S \end{bmatrix} \quad (5.5)$$

$$M_{GJ} = \begin{bmatrix} \text{diag}(K) & 0 \\ 0 & \alpha \text{diag}(\hat{S}) \end{bmatrix} = \begin{bmatrix} \text{diag}(P) & 0 & 0 \\ 0 & \text{diag}(G) & 0 \\ 0 & 0 & \alpha \text{diag}(\hat{S}) \end{bmatrix} \quad (5.6)$$

where

$$S = C + B^T K^{-1} B = C + \begin{bmatrix} B_1^T & B_2^T \end{bmatrix} \begin{bmatrix} P & L \\ L^T & G \end{bmatrix}^{-1} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \quad (5.7)$$

$$\hat{S} = C + B^T \text{diag}(K)^{-1} B = C + \begin{bmatrix} B_1^T & B_2^T \end{bmatrix} \begin{bmatrix} \text{diag}(P) & 0 \\ 0 & \text{diag}(G) \end{bmatrix}^{-1} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}. \quad (5.8)$$

Let,

$$\tilde{S} = C + B_1^T P^{-1} B_1 + B_2^T G^{-1} B_2 \quad (5.9)$$

Since  $[B_1; B_2] \in \mathbb{R}^{nd \times np}$  has full column rank,  $\tilde{S}$  is nonsingular even when  $C=0$ .

Let the block diagonal preconditioner for  $A$  be:

$$M_A = \begin{bmatrix} P & 0 & 0 \\ 0 & G & 0 \\ 0 & 0 & \tilde{S} \end{bmatrix} \quad (5.10)$$

where  $P \in \Re^{m \times m}$ ,  $G \in \Re^{n \times n}$ , and  $\tilde{S} \in \Re^{np \times np}$ . Recall that  $m$  is the structural displacement DOFs,  $n$  the soil displacement DOFs,  $nd = m+n$  is the total displacement DOFs, and  $np$  the pore pressure DOFs.

Let  $R_P$ ,  $R_G$ , and  $R_S$  be the Cholesky factors of  $P$ ,  $G$ , and  $\tilde{S}$  blocks, respectively. Then the preconditioner (5.10) can be written as:

$$M_{AL} = \begin{bmatrix} R_P^T & 0 & 0 \\ 0 & R_G^T & 0 \\ 0 & 0 & R_S^T \end{bmatrix}, \quad M_{AR} = \begin{bmatrix} R_P & 0 & 0 \\ 0 & R_G & 0 \\ 0 & 0 & R_S \end{bmatrix} \quad (5.11)$$

Let  $W = M_{AL}^{-1} A M_{AR}^{-1}$  be the preconditioned matrix. We have,

$$W = \begin{bmatrix} I_m & R_P^{-T} L R_G^{-1} & R_P^{-T} B_1 R_S^{-1} \\ R_G^{-T} L^T R_P^{-1} & I_n & R_G^{-T} B_2 R_S^{-1} \\ R_S^{-T} B_1^T R_P^{-1} & R_S^{-T} B_2^T R_G^{-1} & -R_S^T C R_S^{-1} \end{bmatrix} = \hat{W} + E \quad (5.12)$$

where

$$\hat{W} = \begin{bmatrix} I_m & 0 & R_P^{-T} B_1 R_S^{-1} \\ 0 & I_n & R_G^{-T} B_2 R_S^{-1} \\ R_S^{-T} B_1^T R_P^{-1} & R_S^{-T} B_2^T R_G^{-1} & 0 \end{bmatrix}, \quad E = \begin{bmatrix} 0 & \tilde{L} & 0 \\ \tilde{L}^T & 0 & 0 \\ 0 & 0 & -R_S^{-T} C R_S^{-1} \end{bmatrix} \quad (5.13)$$

and

$$\tilde{L} = R_P^{-T} L R_G^{-1}. \quad (5.14)$$

It is readily shown that the eigenvalues decomposition of  $\hat{W}$  are given by  $\hat{W} = Q \Lambda Q^T$ , where  $Q$  is orthogonal and  $\Lambda$  is a diagonal matrix whose diagonal elements are the eigenvalues of  $\hat{W}$  given by:

$$\lambda(\widehat{W}) = \begin{cases} \frac{1}{2}(1 + \sqrt{1 + 4\sigma_j^2}) & j = 1, \dots, np \\ \frac{1}{2}(1 - \sqrt{1 + 4\sigma_j^2}) & j = 1, \dots, np \\ 1 & \text{with multiplicity } nd - np \end{cases} \quad (5.15)$$

where, and  $\sigma_j^2, j = 1, \dots, np$  are the eigenvalues of  $X$ ,

$$\begin{aligned} X &= R_s^{-T} (B_1^T R_p^{-1} R_p^{-T} B_1 + B_2^T R_G^{-1} R_G^{-T} B_2) R_s^{-1} \\ &= R_s^{-T} (\tilde{S} - C) R_s^{-1} = I_{np} - R_s^{-T} C R_s^{-1} \end{aligned} \quad (5.16)$$

Assuming that  $\|C\|^2$  is so small that  $\|R_s^{-T} C R_s^{-1}\|_2 \ll 1$ . By the Bauer-Fike Theorem (Golub and Van Loan, 1989, p. 342), the eigenvalues of  $\widehat{W}$  are clustered within 3 discs of radii  $\delta = \|R_s^{-T} C R_s^{-1}\|_2$  and they are centered at  $\gamma_- = (1 - \sqrt{5})/2$  with multiplicity  $np$ , 1 with multiplicity  $nd - np$ , and  $\gamma_+ = (1 + \sqrt{5})/2$  with multiplicity  $np$ .

By the Bauer-Fike Theorem (Golub and Van Loan, 1989, p. 342), the eigenvalues of  $W$  are those of  $\widehat{W}$  up to the perturbation  $\|E\|_2 = \max\{\|\tilde{L}\|_2, \|R_s^{-T} C R_s^{-1}\|_2\} = \|\tilde{L}\|_2$ , assuming that  $\|C\|_2$  is so small that  $\|R_s^{-T} C R_s^{-1}\|_2 \ll \|\tilde{L}\|_2$ . It turns out that there are a few rows of  $\tilde{L}$  which have norms that are much larger than the other rows, and the perturbation  $\|\tilde{L}\|_2$  is too large to give informative bounds on the eigenvalues of  $W$  based on those of  $\widehat{W}$ . By considering the partition  $\tilde{L} = \tilde{L}_1 + \tilde{L}_2$ , where  $\tilde{L}_2$  is formed by extracting those rows of  $\tilde{L}$  with large norms, say  $r$  rows, we have:

$$W = U + V \quad (5.17)$$

where

$$U = \widehat{W} + \begin{bmatrix} 0 & \tilde{L}_1 & 0 \\ \tilde{L}_1^T & 0 & 0 \\ 0 & 0 & -R_s^{-T} C R_s^{-1} \end{bmatrix}, \quad V = \begin{bmatrix} 0 & \tilde{L}_2 & 0 \\ \tilde{L}_2^T & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \quad (5.18)$$

**Theorem 1** Suppose  $U, V \in \mathcal{H}^{N \times N}$  ( $N = m+n+np$ , the size of  $A$ ) are symmetric matrices, and suppose  $V$  has rank at most  $r$ . Assume that the eigenvalues of  $U$  and  $V$  are arranged in ascending order. Then

$$\lambda_k(U+V) \leq \lambda_{k+r}(U) \quad k = 1, \dots, N-2r \quad (5.19)$$

$$\lambda_k(U) \leq \lambda_{k+r}(U+V) \quad k = 1, \dots, N-2r \quad (5.20)$$

**Proof:** See Theorem 4.3.6 in p.184 of Horn and Johnson (1985).

**Theorem 2** For the matrices  $U, V$ , and  $W$  in (5.17), we have

$$\lambda_1(U) \leq \lambda_{1+r}(W) \leq \dots \leq \lambda_{np-r}(W) \leq \lambda_{np}(U) \quad (5.21)$$

$$\lambda_{np+1}(U) \leq \lambda_{np+1+r}(W) \leq \dots \leq \lambda_{N-r}(W) \leq \lambda_N(U) \quad (5.22)$$

**Proof:** We first prove the left and right most inequalities in (5.21). The left most inequality follows from (5.20) with  $k = 1$ , whereas the right most inequality follows from (5.19). Similarly, the left and right most inequalities in (5.22) follows from (5.20) and (5.19) with  $k = np + 1$  and  $k = N - r$ , respectively.

Theorem 2 shows that all the eigenvalues of  $W$ , except  $4r$  of them consisting of  $\lambda_1(W), \dots, \lambda_r(W), \lambda_{np+1-r}(W), \dots, \lambda_{np+r}(W), \lambda_{N+1-r}(W), \dots, \lambda_N(W)$ , are contained in the intervals  $[\lambda_1(U), \lambda_{np}(U)] \cup [\lambda_{np+1}(U), \lambda_N(U)]$ . By the Bauer-Fike Theorem (Golub and Van Loan, 1989, p. 342), the eigenvalues of  $U$  are those of  $\widehat{W}$  up to the perturbation  $\max\{\|\tilde{L}_1\|_2, \|R_s^{-T} C R_s^{-1}\|_2\} =: \varepsilon$ . Since the eigenvalues of  $\widehat{W}$  are clustered at  $\gamma_- = (1 - \sqrt{5})/2$  with multiplicity  $np$ , 1 with multiplicity  $nd - np$ , and  $\gamma_+ = (1 + \sqrt{5})/2$  with multiplicity  $np$ , we know that all the eigenvalues of  $W$ , except  $4r$  of them, are contained in the intervals:

$$\gamma_- - \varepsilon \leq \lambda_1(U) \leq \lambda_{1+r}(W) \leq \dots \leq \lambda_{np-r}(W) \leq \lambda_{np}(U) \leq \gamma_- + \varepsilon \quad (5.23)$$

$$1 - \varepsilon \leq \lambda_{np+1}(U) \leq \lambda_{np+1+r}(W) \leq \dots \leq \lambda_{N-r}(W) \leq \lambda_N(U) \leq \gamma_+ + \varepsilon \quad (5.24)$$

*Remark 1*

Since the order of magnitude of elements of  $\tilde{L}$  is, in general, affected by the inverse of the square root of soil-structure stiffness ratios (see Appendix D), the perturbation  $\|\tilde{L}_1\|_2$  (5.18) will become smaller with the increase of soil-structure stiffness ratios. That is, when the soil-structure stiffness ratio is large, the clustering of eigenvalues of the  $W$  (5.15) increases towards the centre of the discs at 1 and  $(1 \pm \sqrt{5})/2$ . The numerical results in Section 5.3.1.1 partially substantiate this Remark.

*Remark 2*

For a  $3 \times 3$  block matrix  $A$ , a Schur complement  $\tilde{S}$  (5.9) is proposed which ignores the off-diagonal block  $L$ . Since the computation of  $\tilde{S}$  involves only the diagonal blocks of  $K$ , it is simpler in comparison to  $S$  (5.7) given by Murphy *et al.* (2000). The proposed  $\tilde{S}$  has an added advantage in that the blocks  $P$  and  $G$  can be approximated individually (and differently) for large-scale practical problems, whereas an approximation to the entire  $K$  need to be considered for  $S$ . Hence, the preconditioner (5.10) is the special case of preconditioner (5.5), when  $L = 0$ . The numerical results in Section 5.3.1.1 will show that the exclusion of  $L$  in  $\tilde{S}$  has practically no effect on the convergence in terms of iteration count, but reduces the computation cost of Schur complement significantly.

### *Remark 3*

We have assumed that  $\|C\|$  is very small because it is proportional to the permeability of the materials [see Equation (5.3)] and the coupled consolidation analysis usually involves the materials of low permeabilities. However,  $\|C\|$  may not be small if the analysis involves either a large time step or a highly permeable material. When  $\|C\|$  is large, Phoon *et al.* (2002) concluded that the GJ preconditioner is closely related to the standard Jacobi (SJ) preconditioner (2.8). According to several researches (see, for example, Ferronato *et al.*, 2001; Lee *et al.*, 2002; Chen *et al.*, 2006; Ferronato *et al.*, 2009), when  $\|C\|$  is large, the ill-conditioning of the problem reduces and standard Jacobi (SJ) performs well for such problems. On the other hand, there can be a significant contrast in permeabilities of structural elements and soils, the effect of which may not be straightforward. Section 5.3.1.4 will discuss the numerical results on a wide range of contrasts in permeability of materials.

## **5.3. Numerical experiments**

Although the exact block diagonal preconditioner (5.10) has an attractive eigenvalue clustering property, this exact form would be too expensive for practical use because some blocks may not be readily invertible. For large-scale computing, the practical approach is to approximate the blocks so that they are cheap to invert and yet remain fairly effective in clustering the eigenvalues. However, the actual performance of these approximations can only be evaluated numerically. Hence, this Section serves to achieve the following objectives:

- (a) Establish an approximate block diagonal preconditioner as close to a simple diagonal preconditioner as possible. But, it should be effective in mitigating the coupled ill-conditioning contrasts in stiffness and permeability of the materials.
- (b) Fine tune the above preconditioner for higher efficiency, keeping in mind the memory constraint.
- (c) Comparison of the proposed preconditioner with existing GJ, MSSOR, and ILU preconditioners.

Since the coefficient matrix  $A$  is symmetric, only the upper triangular part of  $A$  is stored in compressed sparse column (CSC) format to reduce memory usage (see Section 2.3). The preconditioned system is solved using SQMR method (Freund and Nachtigal, 1995) (see Section 2.1).

The problem of interest involves two representative soil-structure interaction problems: piled-raft foundation and tunneling examples. Similar to Chapter 4, the former problem is solved using in-house Fortran 90 programs compatible with programs of Smith and Griffiths' (1997) and the latter is solved after implementing the above code into GeoFEA (2006).

### **5.3.1. Piled-raft foundation**

The piled-raft foundation problem considered here is the same as that in Chapter 4 for the drained analysis (Section 4.3.1). The FE mesh discretizations are the same as those shown in Figure 4.2 in Chapter 4. For coupled consolidation analysis, the discretization comprises 20-noded hexahedral elements for displacement degrees of freedom (DOFs) coupled with 8-noded hexahedral elements for pore pressure DOFs. Hence, each finite element

(including pile/raft elements) consists of 60 displacement DOFs and 8 pore pressure DOFs.

The ground water table is assumed to be at the ground surface and is in hydrostatic condition at the initial stage. Free draining with zero pore pressures is assumed on the top surface and the base is impermeable. The base of the mesh is assumed to be fixed in all directions; side face boundaries are constrained in the transverse direction but free in in-plane directions. A uniform load of 100 kPa is applied to the entire cap area in the first time step and the time increment is taken as  $\Delta t = 1$ s. Subsequent dissipation of pore pressure and the settlement are studied by using a backward difference technique.

All materials (pile, raft, and soil) are assumed to be linear elastic with a constant effective Poisson's ratios. The effective Young's modulus of pile ( $E'_p$ ) is varied to study potential ill-conditioning due to contrasts in pile-soil stiffnesses. The details of material properties used for the numerical study are shown in Table 5.1. The variation in hydraulic conductivity ( $k$ ) of the materials will be considered later in Section 5.3.1.4. The lower bound of pile-soil stiffness ratio ( $E'_p/E'_s$ ) covers the natural variation of soil types whereas the upper bound covers soil-structure interaction problems, where  $E'_s$  is the Young's modulus of soil. In addition,  $E'_p/E'_s$  equals to 1 is also considered. It refers to the condition where all pile/raft elements are replaced by soil elements, i.e. a fictitious pile. This configuration is included as a benchmark to gauge the effect of differences in stiffness relative to a homogeneous soil condition on various preconditioning methods. This is done for the sake completeness.

### 5.3.1.1. Validation of theory

As expected, the spectral condition number of  $A$  increases by several orders with the increase of pile-soil stiffness ratios ( $E'_p/E'_s$ ), holding other parameters constant, and the unpreconditioned SQMR solver immediately fails to converge as shown in Figure 5.1. Spectral condition number is the ratio of the absolute largest and smallest eigenvalues of the matrix. However, the trend is opposite with the exact block diagonal preconditioner (5.10). Both the spectral condition number and the number of iteration counts decrease with the increase of  $E'_p/E'_s$ . Similar convergence trend was also observed for drained problems with exact block diagonal preconditioner in Chapter 4. The decrease in iteration count can be attributed to the increase in clustering of eigenvalues of the preconditioned matrix towards the three centers of the discs at 1,  $(1+\sqrt{5})/2$ , and  $(1-\sqrt{5})/2$  with the increase of  $E'_p/E'_s$  (Figure 5.2). This substantiates the Remark 1 in Section 5.2. However, some of the eigenvalues (particularly extreme eigenvalues) do not decrease with increasing  $E'_p/E'_s$ , giving nearly the same spectral condition numbers (Figure 5.1). This is because the norms of some rows of  $\tilde{L}$  (i.e.  $\tilde{L}_2$ , see Theorem in Section 5.2) do not decrease with increasing  $E'_p/E'_s$ . For a 1D example, at least one row of  $\tilde{L}$  is independent of soil-structure stiffness ratios (see, for example, Appendix D). The numerical results of the studied 3D problem show that the maximum value of norms of  $\tilde{L}$  lies in between 0.5 and 1.0 depending on the problem. For example, if we assume 0.3 as a perturbation value, the  $r$  number of rows of  $\tilde{L}$  with  $\|\text{row}(\tilde{L})\| \geq 0.3$  is 8 and 1, respectively, for  $E'_p/E'_s = 1000$  and

41000 for the studied problem. The respective bounds of the eigenvalues of the preconditioned matrices are shown in Figure 5.2.

Figure 5.1 also demonstrates that the Schur complement  $\tilde{S}$  or  $S$  for block (3, 3) in  $M_A$  (5.10) makes no difference in the performance. The relative merits of  $\tilde{S}$  compared with  $S$  are discussed in Remark 2 in Section 5.2. However, the exact block diagonal preconditioners are impractical. Even the computation of the simpler  $\tilde{S}$  is very expensive since it requires the Cholesky factorization of the large matrix  $G$  and computation of the matrix  $B_2^T G^{-1} B_2$ . In addition, as the last matrix is typically dense and large, it would also require excessive amount of memory to store it. Hence for practical computation, approximation of the block  $G$  is necessary when selecting a cheaper alternative to  $\tilde{S}$ . We should note that as the elements of  $P$  have much larger magnitudes than those of  $G$ , the contribution of the term  $B_1^T P^{-1} B_1$  to  $\tilde{S}$  is much less significant than the term  $B_2^T G^{-1} B_2$ . Thus it would not make substantial difference to the effectiveness of  $\tilde{S}$  even if we replace the block  $P$  in  $B_1^T P^{-1} B_1$  by a cheaper alternative such as  $\text{diag}(P)$ .

### 5.3.1.2. Effect of approximation of diagonal blocks of the preconditioner

The following approximations for the pile block  $P$  [block (1, 1)] in the block diagonal preconditioner (5.10) are investigated for their effectiveness:

$$\begin{aligned}\hat{P}_1 &= \text{diag}(P) \\ \hat{P}_2 &= \text{SSOR}(P) = (L_p + D_p)(D_p)^{-1}(L_p^T + D_p) \\ \hat{P}_3 &= \text{ILU0}(P) \\ \hat{P}_4 &= P = R_p^T R_p\end{aligned}\tag{5.25}$$

where  $L_p$  and  $D_p$  are strictly lower triangular and diagonal part of  $P$ , respectively, and  $R_p$  is the Cholesky factorization of block  $P$ . Here, the exact block  $P$  (i.e.  $\hat{P}_4$ ) is considered because Chapter 4 concluded that the block  $P$  is the main source of ill-conditioning due to contrast in stiffnesses in soil-structure interaction problems.

The approximations  $\hat{P}_1$  and  $\hat{P}_2$  would require the lowest storage of only a single  $m \times 1$  vector for the preconditioner. For  $\hat{P}_1$  and  $\hat{P}_2$ , the FE simulation can effectively be performed by storing only the upper triangular part of  $P$ . In contrast, the approximations  $\hat{P}_3$  (incomplete LU factorization with zero fill-ins) and  $\hat{P}_4$  require the entire  $P$  matrix to be stored for the preconditioner. For ILU factorization, the matrix is first reordered by Reverse Cut-hill McKee (RCM) permutation (George and Lui, 1981) so as to minimize possible fill-ins. The subroutines for ILU and RCM algorithm are obtained from Saad (1994a) and Burkardt (2003), respectively. For SQMR solver, the symmetric factorizations ( $R^T R$  or  $\overline{L D L^T}$ ) are obtained by manipulating the upper triangular factor from ILU subroutines. Similarly, prior to sparse Cholesky factorization of  $P$ , the matrix is reordered by Multiple Minimal Degree (MMD) permutation (George and Lui, 1981) to minimize the possible fill-ins in the factorization to cut down the cost of factorization and back substitution in each preconditioning step. Subroutines for sparse Cholesky factorization with MMD ordering (Ng and Peyton, 1993) are obtained from SparseM package (Koenker and Ng, 2007). Although the computational and storage costs for  $\hat{P}_3$  are moderate for a sparse  $P$ , the approximation  $\hat{P}_4$  can be computationally expensive, particularly when the size of  $P$  becomes large for

large soil-structure interaction problems. However, it is significantly less expensive in comparison to the exact block diagonal preconditioner (5.10), if the approximations of other diagonal blocks are crude. In addition to this, we also considered the incomplete factorization with partial fill-ins [such as ILUT (Saad, 1994b)] for the approximation of block  $P$ .

As mentioned in Chapter 4, the size of the stiff block  $P$  (pile block in this example) would be much smaller than the size of soil block  $G$  for all soil-structure interaction problems because the soil mesh must increase with the increase of the size of the structural components to avoid boundary effects. Hence, the most practical approximation for the block  $G$  [block (2, 2)] is:

$$\hat{G}_1 = \text{diag}(G) \quad (5.26)$$

The diagonal approximation is easy to invert. It is certainly the cheapest possible approximation as well. Other approximations such as ILU0 or variants of ILU would be significantly more expensive by comparison (both in terms of storage and time) for large-scale computing (see, for example, Chapter 4).

Likewise, the computation of an exact Schur complement  $\tilde{S}$  is also onerous. Therefore, a less expensive approximation of  $\tilde{S}$  [block (3, 3)] is considered:

$$\hat{S}_1 = C + B_1^T \text{diag}(P)^{-1} B_1 + B_2^T \text{diag}(G)^{-1} B_2 \quad (5.27)$$

Note that, although the exact  $S$  and  $\tilde{S}$  [Equations (5.7) and (5.9)] are different, their diagonal approximations [Equations (5.8) and (5.27)] turn out to be identical.

In summary, our proposed practical block diagonal preconditioner with the above approximations would take the following form:

$$\hat{M}_A = \begin{bmatrix} \hat{P} & 0 & 0 \\ 0 & \text{diag}(G) & 0 \\ 0 & 0 & \alpha \text{diag}(\hat{S}_1) \end{bmatrix} \quad (5.28)$$

where  $\hat{P}$  varies from  $\hat{P}_1$  to  $\hat{P}_4$  and the parameter  $\alpha$  is a user supplied real-value. For a cheap approximation of Schur complement such as Equation (5.8), the effectiveness of  $\alpha$  (particularly, the negative sign) has been demonstrated elsewhere (e.g. Phoon *et al.*, 2002; Toh *et al.*, 2004; Chen *et al.*, 2006). In this study,  $\alpha = -4$  is used based on the theoretical results from (Phoon *et al.*, 2002). It should be noted that for the special case of  $\hat{P}_1 = \text{diag}(P)$  and  $\alpha = -4$ , the preconditioner (5.28) is identical to the GJ preconditioner (5.6). In other words, GJ would be the baseline to gauge the proposed preconditioner and any approximation of  $P$  which converges slower than the GJ is deemed inferior.

Figure 5.3 demonstrates that when the block  $P$  is not exact in the proposed preconditioner (5.28), the performance (iteration count and total CPU runtime) of the preconditioners degrades with the increase of pile-soil stiffness ratios ( $E'_p/E'_s$ ). Conversely, the effect of stiffness ratio is stabilized when the (1, 1) block is the exact  $P$  (i.e.  $\hat{P}_4$ ) for the preconditioner (5.28). This is consistent to the findings for drained analysis of the same problem (Chapter 4). Such mitigation of deteriorating behavior due to difference in material properties can be attributed to the almost identical real eigenvalue profiles of the preconditioned system by the preconditioner (5.28) with exact  $P$ , as shown in Figure 5.4. Since SSOR and ILU0 approximations of  $P$  do not

seem to be satisfying the aimed results, they will not be included in the succeeding discussions.

It may be possible that an approximation in between  $\hat{P}_3 = \text{ILU0}(P)$  and  $\hat{P}_4 = P$  exists which can optimize the tradeoff between overhead cost (formation time and computer memory requirement) of preconditioning and overall runtime (particularly for a problem with a very large size of block  $P$ ). For example, Toh *et al.* (2004) studied the ILUT approximations – incomplete Cholesky factorization with partial fill-ins for symmetric matrices (Saad, 1994b) of  $K$  and  $S$  for their block preconditioners. Similarly, Bergamaschi *et al.* (2007, 2008) used ILUT and AINV (Benzi *et al.*, 2001) approximations of block  $K$  for their block constrained preconditioners. See Section 2.2.4 for more details about these block preconditioners. Hence, the performance of  $\text{ILUT}(\rho, \tau)$  approximation of block  $P$  is considered, where  $\rho$  is the number of fill-ins in excess of original number of non-zeroes in each row of lower and upper triangular factors, and  $\tau$  is a dropping parameter below which the fill-ins are discarded. Table 5.2 shows that the ILUT approximation of the block  $P$  does improve the performance in comparison to  $\text{ILU0}(P)$  and is comparable to its Cholesky counterpart when the material stiffness ratio is below 1000. However, the material ill-conditioning is only effectively suppressed when the block  $P$  is solved directly. With ILUT, the effective suppression of such ill-conditioning can only be attained at a cost of allowing more fill-ins, the optimum value of which is unknown *a priori* when the problem changes. As a result, such an approximation is difficult to apply routinely and reliably for the solution of complex soil-structure interaction problems that are of interest. A practicing engineer running an FEM problem is unlikely to know how to

choose an *ad-hoc* parameter without conducting costly parametric studies. Moreover, when more fill-ins are allowed in ILUT, it becomes as expensive as the Cholesky factorization.

Thus, let us denote the preconditioner (5.28) with exact block  $P$  by  $M_1$  so that:

$$M_1 = \begin{bmatrix} P & 0 & 0 \\ 0 & \text{diag}(G) & 0 \\ 0 & 0 & \alpha \text{diag}(\hat{S}_1) \end{bmatrix} \quad (5.29)$$

Observe that  $M_1$  is inferior, in terms of convergence behavior, to the exact form  $M_A$  (5.10), although  $P$  is exact in  $M_1$ . In Figure 5.1, the number of iteration counts of  $M_A$  decreases with the increase of  $E'_p/E'_s$ . In Figure 5.3, the number of iteration counts of  $M_1$  holds steady with the increase of  $E'_p/E'_s$ . It would be unrealistic to expect an approximate form to outperform the exact form in convergence. The approximate form, on the other hand, is much more efficient in terms of runtime. Memory constraints is also far less of a problem. For example, while  $M_1$  took about 620-750 sec and 855 MB RAM for the solution of 9-piled raft in 25×25×35 mesh (Figure 4.2a),  $M_A$  was inapplicable owing to insufficient memory. However, note that, preconditioners based on exact  $P$  alone can be futile without an appropriate Schur complement preconditioning for the flow stiffness block. This is observed when the  $M_1$  preconditioner is compared with the base line (BL) preconditioner, where  $M_{BL}$  (5.30) did not converge in 50,000 iterations for the above problem for even with a homogeneous material ( $E'_p/E'_s=1$ ). The BL preconditioner means preconditioning the displacement DOFs only (see, for example, Phoon *et al.*, 2002):

$$M_{BL} = \begin{bmatrix} P & 0 & 0 \\ 0 & \text{diag}(G) & 0 \\ 0 & 0 & I_{np} \end{bmatrix} \quad (5.30)$$

It indicates the effect of approximation of other blocks (i.e. soil and flow stiffness blocks) in the preconditioner, which is considered in the subsequent Section.

### 5.3.1.3. Effect of approximation of soil and flow stiffness blocks

We only consider approximations of blocks  $G$  (soil stiffness matrix) and  $\tilde{S}$  (Schur complement matrix for flow stiffness block) that require no or small increase in computer memory than of the preconditioner (5.29). As discussed in (Chen *et al.*, 2006), the memory requirement for SSOR (Symmetric Successive Over Relaxation) approximation is as low as a simple diagonal. Hence, it is an obvious choice. Thus, the following different approximations are studied:

$$M_2 = \begin{bmatrix} P & 0 \\ 0 & \text{MSSOR}(H) \end{bmatrix} \text{ with } H = \begin{bmatrix} G & B_2 \\ B_2^T & -C \end{bmatrix} \quad (5.31)$$

$$M_3 = \begin{bmatrix} P & 0 & 0 \\ 0 & \text{SSOR}(G) & 0 \\ 0 & 0 & \alpha \text{diag}(\hat{S}_1) \end{bmatrix} \quad (5.32)$$

where,  $H$  is the lower-right  $2 \times 2$  block of  $A$  (5.4). The diagonal of  $H$  is replaced by the diagonal of  $M_1$  (5.29), similar to the replacement of diagonal of  $A$  by  $GJ$  in the original development of MSSOR preconditioner (Chen *et al.*, 2006). The approximation such as ILU0 of blocks  $G$  or  $H$  is not included because of two reasons (i) it can be slower than SSOR (see, for example, Chapter 4), and (ii) it incurs large memory overhead. The ILU0 is

also unstable for consolidation problems; see detailed comparisons of ILU0 and MSSOR preconditioners in (Phoon *et al.*, 2008).

Figure 5.5 shows that the MSSOR approximation of the block  $H$  (in  $M_2$ ) or SSOR approximation of block  $G$  (in  $M_3$ ) makes the preconditioners to be about 2 times faster in terms of total CPU time while the iteration counts can be reduced by about 3 times in comparison to the diagonal approximation of the same blocks in  $M_1$ . Such an improvement in the performance is because the spectrum of the preconditioned matrices also shrunk by about 3 times with  $M_2$  and  $M_3$  preconditioners than with  $M_1$ , as shown in Figure 5.6. Since there is no appreciable difference in the performance between  $M_2$  and  $M_3$ , a selection of either one is equally preferable. However, one disadvantage of  $M_3$  is that the submatrices in upper symmetric  $A$  (e.g. blocks  $L$ ,  $B_1$ ,  $B_2$ , and  $C$ ) are required to be stored separately for the efficient matrix-vector multiplication (the algorithm is presented in Appendix C), whereas only the separate storage of  $[L \ B_1]$  is required in the case of  $M_2$  preconditioner. From the view point of implementation, the preconditioner  $M_2$  also reduces the coefficient matrix  $A$  to a  $2 \times 2$  form. Thus, we adopt  $M_2$  preconditioner for the rest of the study. Finally it goes without saying that the implementation of  $M_1$  is much simpler than the implementation of  $M_2$  or  $M_3$ .

#### **5.3.1.4. Effect of pile and soil permeabilities**

In the preceding Sections, the permeabilities of pile and soil were held constant. In actual practice, the permeability of soil changes during consolidation process. This, on the other hand, affects the flow stiffness matrix  $C$  [Equation (5.4)]. Also, the permeability of soil varies over several orders of

magnitude ( $k_s = 10^{-3}$  to  $10^{-12}$  m/s) depending on the soil type. Permeability significantly lower than  $10^{-9}$  m/s is common for unsaturated soils, e.g. up to  $10^{-12}$  m/s for Singapore residual soils (Agus *et al.*, 2005). Similarly, the permeability of a pile can be as high as  $k_p = 1$  m/s, e.g. for a stone column (Han and Ye, 2002), to as low as  $k_p = 10^{-17}$  m/s, e.g. for a reinforced concrete pile (Gonilho Pereira *et al.*, 2009). The stone columns are used for ground improvement of a soft soil and the reinforced concrete piles are commonly used for building foundations. This Section demonstrates the effect of this wide range of contrasts in permeability of materials on the above studied new block diagonal preconditioners considering a constant pile-soil stiffness ratio of 30,000. See Table 5.1 for more details of the material properties used for the analysis.

As shown in Figure 5.7, the effect of soil-structure permeability contrasts on  $M_1$  and  $M_2$  is apparent (up to 60% deviation) for the extreme case of ground improvement with highly permeable pile ( $k_p = 1$  m/s) in nearly impermeable soil ( $k_s \leq 10^{-9}$  m/s). However, when the pile (or structure) is less permeable ( $k_p \leq 10^{-9}$  m/s), the effect of relative difference in the soil permeabilities is minor due to  $M_1$ , while such effects are effectively mitigated by  $M_2$ . This indicates that although there is a room for improvement on  $M_1$  and  $M_2$  for a complete mitigation of the effect due to contrasts in permeabilities, it may be achievable only at a cost of more complex preconditioners. On the other hand, in most soil-structure interaction problems, the structural components are less permeable, for which, the simple  $M_1$  and  $M_2$  preconditioners appear to be quite effective.

### 5.3.1.5. Comparison of $M_1$ , $M_2$ , GJ, MSSOR, and ILU preconditioners

Figure 5.8 shows that the mitigation of effect of material heterogeneity (mainly due to stiffness and permeability contrasts) by the block preconditioners ( $M_1$  and  $M_2$ ) enables them to outperform the GJ, MSSOR, and ILU0 preconditioners when  $E'_p/E'_s$  is large. Note that the ILU0 preconditioner considered here is the stabilized ILU0. Stabilization of the factorization is carried out by replacing the pivots dynamically whose absolute values are smaller than a threshold ( $=0.009$  in this study) value (see Chapter 3 for details). As noted in (Phoon *et al.*, 2008), such stabilization is necessary for ILU0 to be successful and to be competitive with MSSOR. However, as shown in the Figure 5.8, ILU0 degrades much more rapidly than MSSOR does for increasing  $E'_p/E'_s$ . On the other hand, ILUT( $10, 10^{-6}$ ) on the entire  $A$  took 4080 s for factorization alone (about six times the total CPU time of  $M_1$ , see Table 5.2) and failed to converge. This indicates that for ILUT to be successful for such ill-conditioned problems, a proper stabilization is mandatory, and underscores the superiority of  $M_1$  and  $M_2$  preconditioners for soil-structure interaction problems. The cross-over stiffness ratios ( $E'_p/E'_s$ ) above which  $M_1$  and  $M_2$  preconditioners are preferable over GJ, MSSOR, and ILU0 preconditioners are about 50 and 1000, respectively. However, these cross-over  $E'_p/E'_s$  points may be problem dependent. The succeeding Section discusses more on this.

### 5.3.1.6. Effect of size of block $P$

One limitation of  $M_1$  or  $M_2$  is the Cholesky factorization of block  $P$ , especially when its size is very large. This Section investigates the effect of the size of block  $P$  in the preconditioners by considering a range of piles from 1 to 49 in a square raft configuration. Firstly, a constant raft thickness of 3m is considered. It is subsequently increased to achieve the maximum size of block  $P$  for which the available RAM (2GB in this study) supports its Cholesky factorization. The layout of piles is the same as that shown in Figure 4.10. The same  $25 \times 25 \times 35$  FE mesh (Table 4.2a) is used for all the problems to keep constant the total number of DOFs for fair comparison, although, in actual practice, the FE mesh domain would also be extended with increasing piled-raft size to avoid boundary effects. Details of problem statistics are presented in Table 5.3.

As shown in Figure 5.9, the performance of GJ, MSSOR, and ILU0 preconditioners are not only affected by the stiffness ratios but also by the number of piles (size of the block  $P$ ) whereas the preconditioners  $M_1$  and  $M_2$  are almost insensitive to both of these factors. When the problem domain is homogeneous (fictitious pile), GJ, MSSOR, and ILU0 are not affected by the size of block  $P$ , because there are actually no piles. However, they become less and less effective as  $E'_p/E'_s$  increases. Interestingly, the proposed block diagonal preconditioners  $M_1$  and  $M_2$  are practically unaffected not only by  $E'_p/E'_s$  but also by the size of block  $P$  indicating the effective mitigation of the ill-conditioning due to material heterogeneity by  $M_1$  and  $M_2$ . Note that the size of block  $P$  is only 668 (less than 0.25% of the size of  $A$ ) for a single pile,

whereas it is 72,601 ( $\approx 25\%$  of the size of  $A$ ) for 49 piles with 3 m raft (see Table 5.3).

Figures 5.10-5.11 show that the saving in runtime by  $M_1$  and  $M_2$  preconditioners over others depends on the problem at hand. If the percentage of stiff DOFs in total is above 5% and the soil-structure stiffness ratio is also large, both  $M_1$  and  $M_2$  can be more than 10 times faster than GJ and MSSOR. However, it is not surprising to know that when the size of  $P$  is larger than 30% of the size of global  $A$ , where the size of global  $A$  is  $291,340 \times 291,340$ , both  $M_1$  and  $M_2$  face the problem of lack of core memory (which is 2GB in the present study). Assuming the size of  $P$  is usually much smaller than the size of  $A$  in most soil-structure interaction problems, this memory requirement for  $M_1$  and  $M_2$  is much less severe than that demanded by an ILUT factorization on the entire  $A$ . Thus, the proposed  $M_1$  and  $M_2$  preconditioners are likely to be useful for the simulation of large-scale ill-conditioned soil-structure interaction problems because of the advantages they offer in mitigating material ill-conditioning.

### 5.3.2. Tunneling

In any tunneling project, long-term settlement will occur with time due to dissipation of excess pore pressure generated during the tunneling process. For large-scale simulation of tunnels, advantages of preconditioned iterative solution methods over direct solution method have been demonstrated by several researchers (e.g. Mroueh and Shahrour, 2003; Lee *et al.*, 2006; Phoon *et al.*, 2006). This study compares the effectiveness of previously discussed preconditioners (GJ, MSSOR,  $M_1$ , and  $M_2$ ) for the 3D FE consolidation analysis of a tunnel. The problem is analyzed using GeoFEA (2006) after

implementing these preconditioners as user defined solvers. Details of GeoFEA implementation are explained in Section 6.2 in Chapter 6. The problem considered here is taken from Möller (2006); see also Vermeer *et al.* (2001). The tunnel with a diameter of 8 m and a cover of 16 m was modeled in a symmetric half with an unsupported excavation of 2m. A block of  $100 \times 55 \times 28$  was divided into 12594 20-noded brick elements, resulting 165,005 unknown DOFs as shown in Figure 5.12. The ground water table is assumed to be at the ground surface and is in hydrostatic condition at the initial stage. More details of the problem are described in Section 4.3.2 in Chapter 4.

Soil is modeled as Mohr-Coulomb material and the liner as linear elastic. The permeabilities of soil and liner are taken as  $10^{-8}$  and  $10^{-12}$  m/s, respectively. Unlike to Möller's parameters, the dilation angle is taken the same as the angle of friction for stiffness matrix  $A$ . The tunnel advancement rate of 4 m/day is assumed. The excavation and installation of 0.3m thick liner was simulated according to step-by-step procedure. Each step simulates an excavated length of 2m by removing the soil elements and the installation of liner in the previously excavated portion. Only 10 steps of excavation are simulated for the demonstration purpose, which results 3229 liner displacement DOFs (about 2% of the size of  $A$ ).

Similar to piled-raft problem, the GJ and MSSOR preconditioners take increasingly larger iteration counts and the CPU times with excavation steps, as shown in Figure 5.13. The inclusion of stiff liner elements may have worsened the ill-conditioning of the system with each excavation step. In contrast, the linear cumulative curve for  $M_1$  and  $M_2$  preconditioners indicates

the mitigation of such ill-conditioning effect. This lets them to outperform the GJ and MSSOR preconditioners by a factor of about 2 (in terms of CPU times) at the end of 10 excavations. Note that, the size of the block  $P$  can be considered to be insignificant (only 2% of the size of  $A$ ) in the studied problem and the liner-soil stiffness ratio is also 476 only (Table 4.4). Table 5.4 shows that the  $M_1$  and  $M_2$  preconditioners have actually performed better than what Figures 5.10-5.11 suggest, except for the case of  $M_2$  versus GJ. This is because of relatively no difference in CPU times of GJ and MSSOR (Figure 5.13), unlike to the piled-raft problem in preceding Sections. This led  $M_2$  to be slower as it uses MSSOR preconditioning for the soil and fluid stiffness blocks [Equation (5.31)]. One possible reason for this discrepancy in CPU times could be due to the differences in sparsity patterns of  $A$  by two different FE algorithms (Figure 5.14). The above findings indicate that the Figures 5.10-5.11 may be taken as a general reference that one can expect from  $M_1$  and  $M_2$  preconditioners for the soil-structure interaction problems, although the actual saving may vary depending on the problem due to some other numerical factors.

Figure 5.15 shows the computed surface settlement profile after 10 excavation steps. The consistent smooth settlement profile indicates the correctness of the simulation. However, the actual settlement depends on many factors such as the rate of excavation, construction sequence, permeability of soils, etc.

## 5.4. Conclusion

The present study proposed some cost effective block diagonal preconditioners that are effective in mitigating the ill-conditioning due to large relative differences in stiffness and permeability of materials for solving large-scale Biot's consolidation equations. These preconditioners were derived from approximations to a theoretical block diagonal preconditioner, which was proven mathematically to possess an attractive eigenvalue clustering property with increasing stiffness contrasts.

Some of the key observations can be summarized as follows:

1. A  $3 \times 3$  block form of the coefficient matrix was proposed for Biot's consolidation analysis of problems involving large relative differences in stiffness of materials. For example, for the soil-structure interaction problems, structural displacement degrees of freedom (DOFs) are separated from soil displacement DOFs and pore pressure DOFs.
2. The  $3 \times 3$  block form of  $A$  offers a greater flexibility in the calculation of Schur complement and its approximate. A simple way of computing Schur complement was proposed that only involves the diagonal blocks of the solid stiffness matrix  $K$ . It simplifies the computation significantly compared with that of Murphy *et al.* (2000). But, no difference in the rate of convergence (in terms of iteration count) was observed for the exact block preconditioners incorporating both Schur complements. The proposed form has an added advantage in that the diagonal blocks of  $K$  can be approximated individually and differently for large-scale practical problems.

3. The approximate block diagonal preconditioners ( $M_1$ ) with exact stiff block  $P$  (stiffness matrix corresponding to stiff materials) with diagonal approximation of soil and Schur complement matrices effectively mitigated the material stiffness contrast effects.
4. The theoretical exact block diagonal preconditioner ( $M_A$ ) shows the number of iteration counts decreases with increasing stiffness contrasts, while proposed approximate forms show almost steady iteration counts with stiffness contrasts. However, latter forms are much cheaper and faster in comparison to  $M_A$ .
5. The MSSOR approximation of soil and flow blocks ( $M_2$ ) or the SSOR approximation of soil block ( $M_3$ ) improves the convergence time by about 55% compared with diagonal approximation of the same blocks in  $M_1$ .
6. The proposed preconditioners demonstrate effective mitigation of ill-conditioning not only due to large stiffness contrasts but also due to large permeability contrasts for most problems.
7. The GJ, MSSOR, and ILU preconditioners were not only affected by the stiffness contrasts but also by the size of the stiff block  $P$ . By contrast,  $M_1$  or  $M_2$  preconditioners are almost insensitive with the increase of both the stiffness contrasts and the size of the stiff block  $P$ . Such mitigation offers significant saving in runtime by latter preconditioners. Some generalized charts have been devised for an estimate of the saving; however, the actual saving may vary depending on the problem at hand.

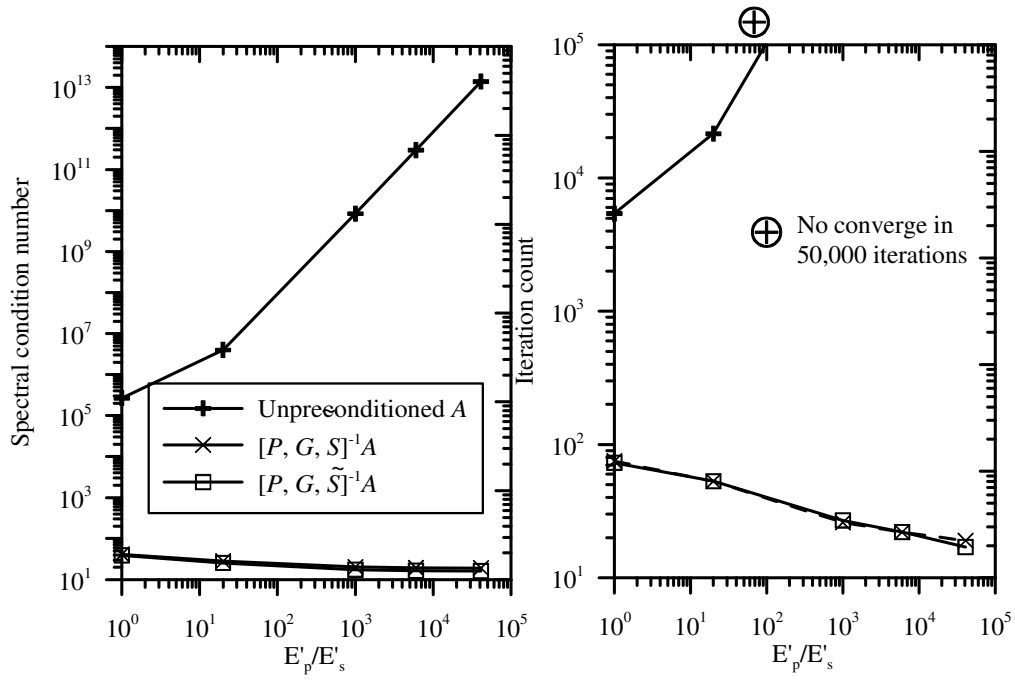


Figure 5.1.  $7 \times 7 \times 7$  mesh: Effect of varying pile-soil stiffness ratios on spectral condition number and iteration count of unpreconditioned and theoretical block diagonal preconditioned matrices. The theoretical preconditioner is as defined by Equation (5.10).

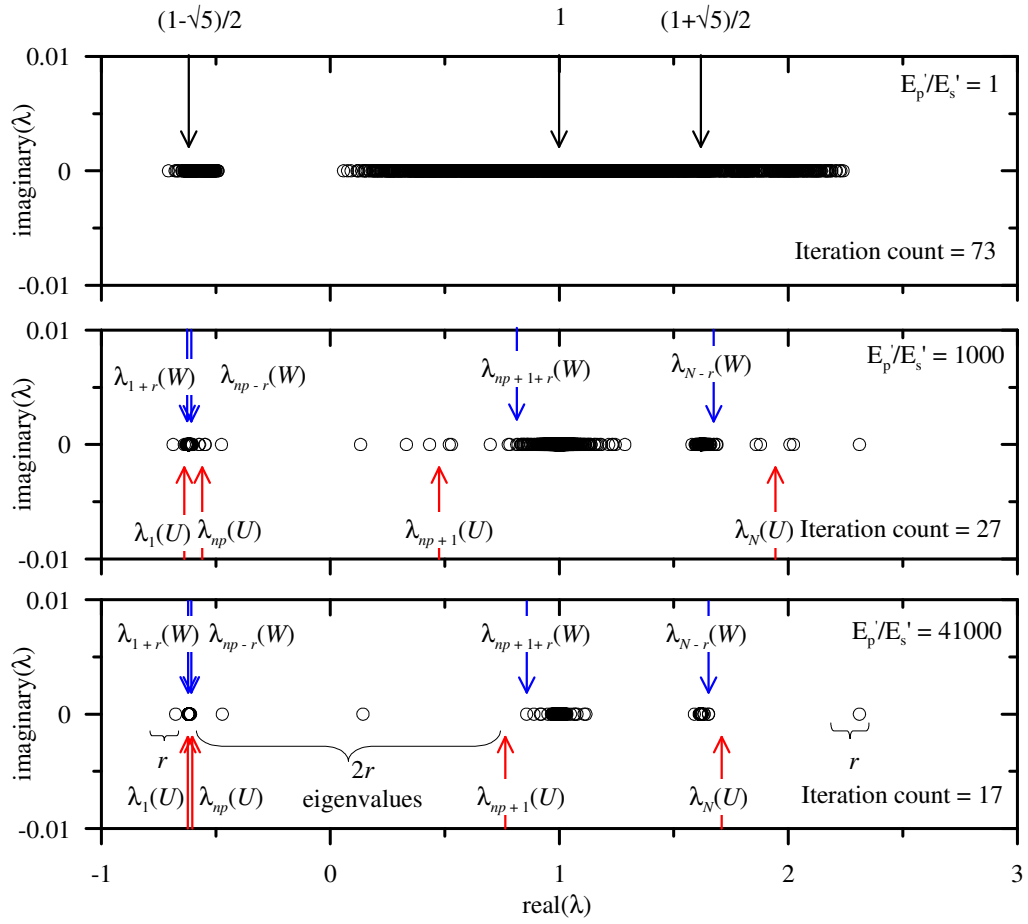


Figure 5.2.  $7 \times 7 \times 7$  mesh: Eigenvalue distribution of the preconditioned system with theoretical exact block diagonal preconditioner for different pile-soil stiffness ratios.  $r$  is the number of rows with  $\|\text{row}(\tilde{L}_2)\|_2 \geq 0.3$  [Equation (5.18)].

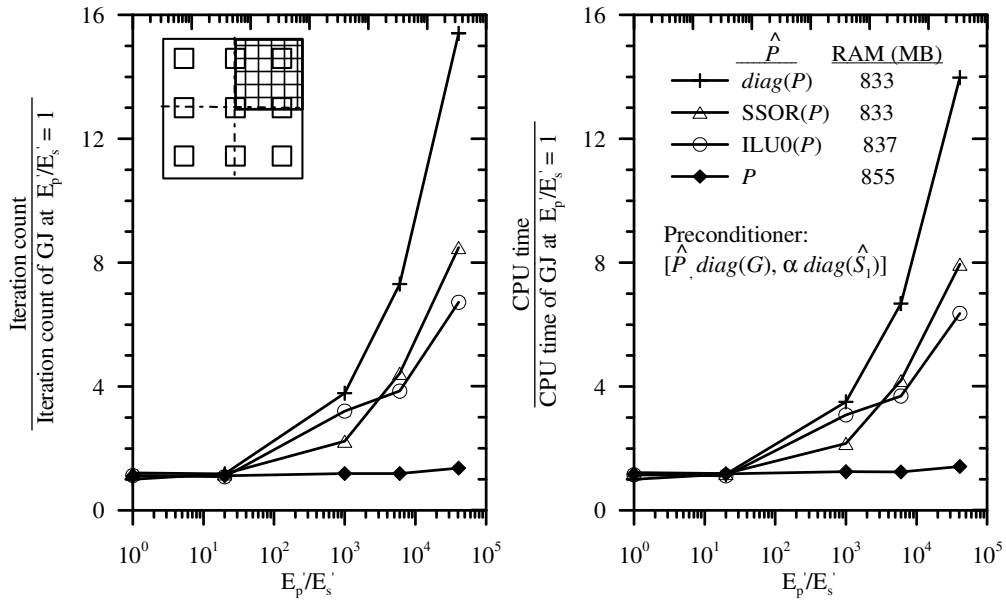


Figure 5.3.  $25 \times 25 \times 35$  mesh: Iteration count and total CPU time of block diagonal preconditioner (5.28) for different approximations of block  $P$ .

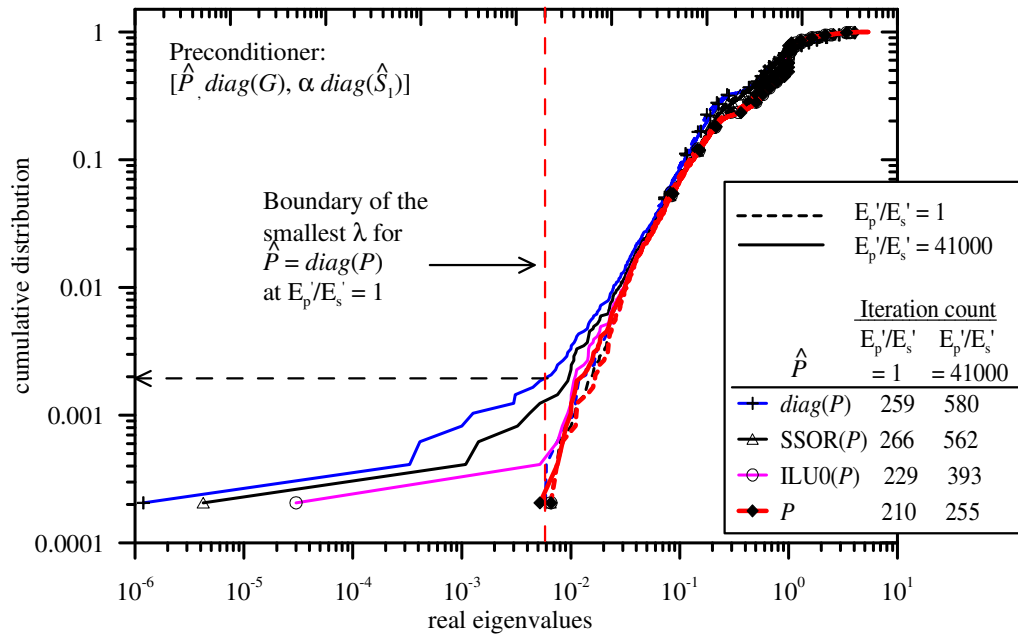


Figure 5.4.  $7 \times 7 \times 7$  mesh: Cumulative distribution of the eigenvalues (real) of the preconditioned system for different approximations of block  $P$  in the preconditioner (5.28).

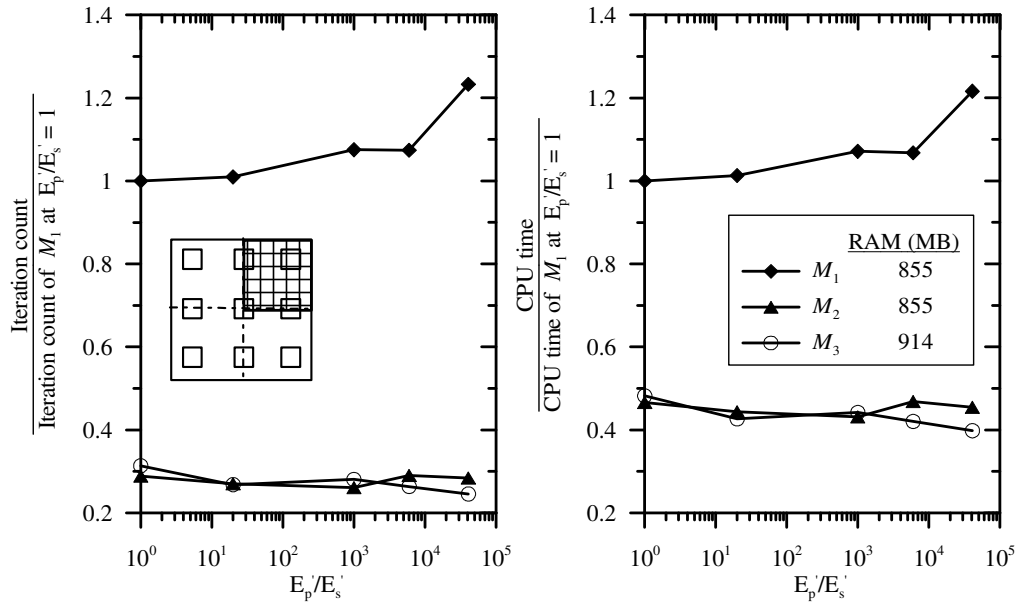


Figure 5.5. 25×25×35 mesh: Performance of different approximations of the soil and Schur complement blocks in the block diagonal preconditioner with exact block  $P$ .

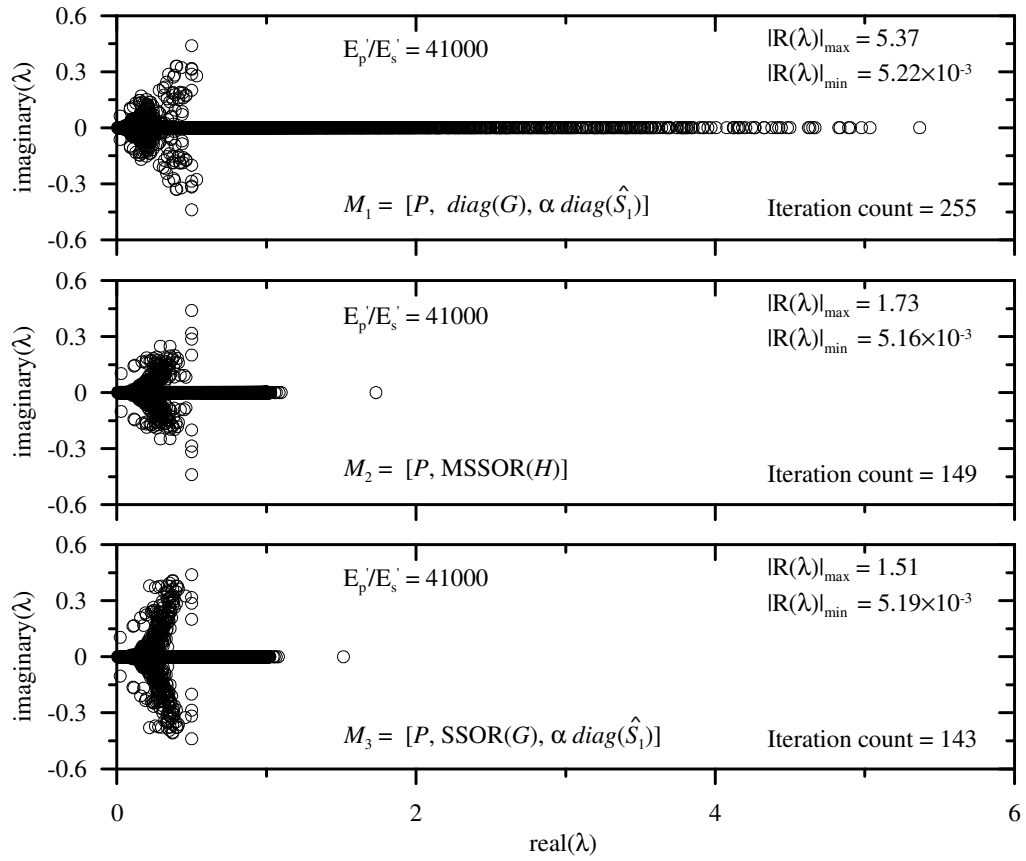


Figure 5.6.  $7 \times 7 \times 7$  mesh: Distribution of the eigenvalues of a preconditioned matrix for different approximations of soil and fluid stiffness blocks in conjunction with an exact block  $P$  in the block diagonal preconditioner.  $R(\lambda) =$  Real part of the eigenvalue.

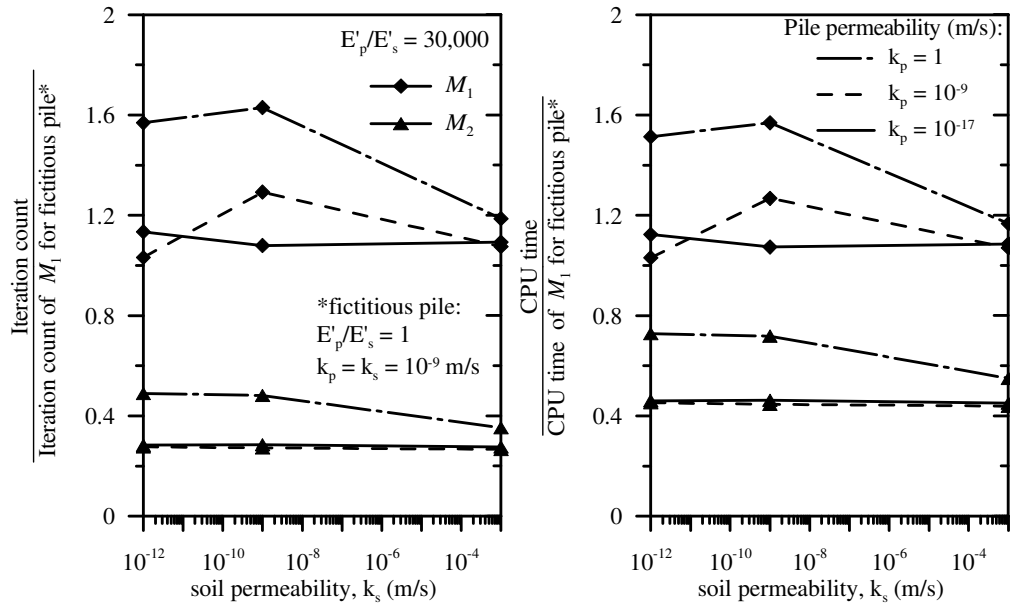


Figure 5.7.  $25 \times 25 \times 35$  mesh: Effect of contrast in pile-soil permeability on the block diagonal preconditioners  $M_1$  and  $M_2$ .

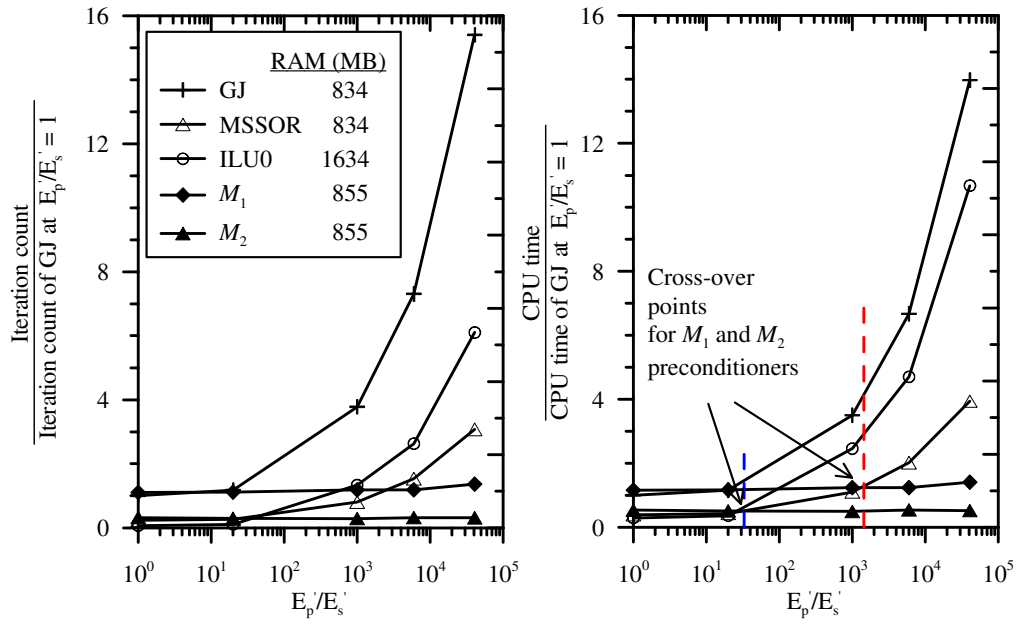


Figure 5.8.  $25 \times 25 \times 35$  mesh: Comparison of proposed preconditioners  $M_1$  and  $M_2$  with GJ and MSSOR preconditioners for varying pile-soil stiffness ratios.



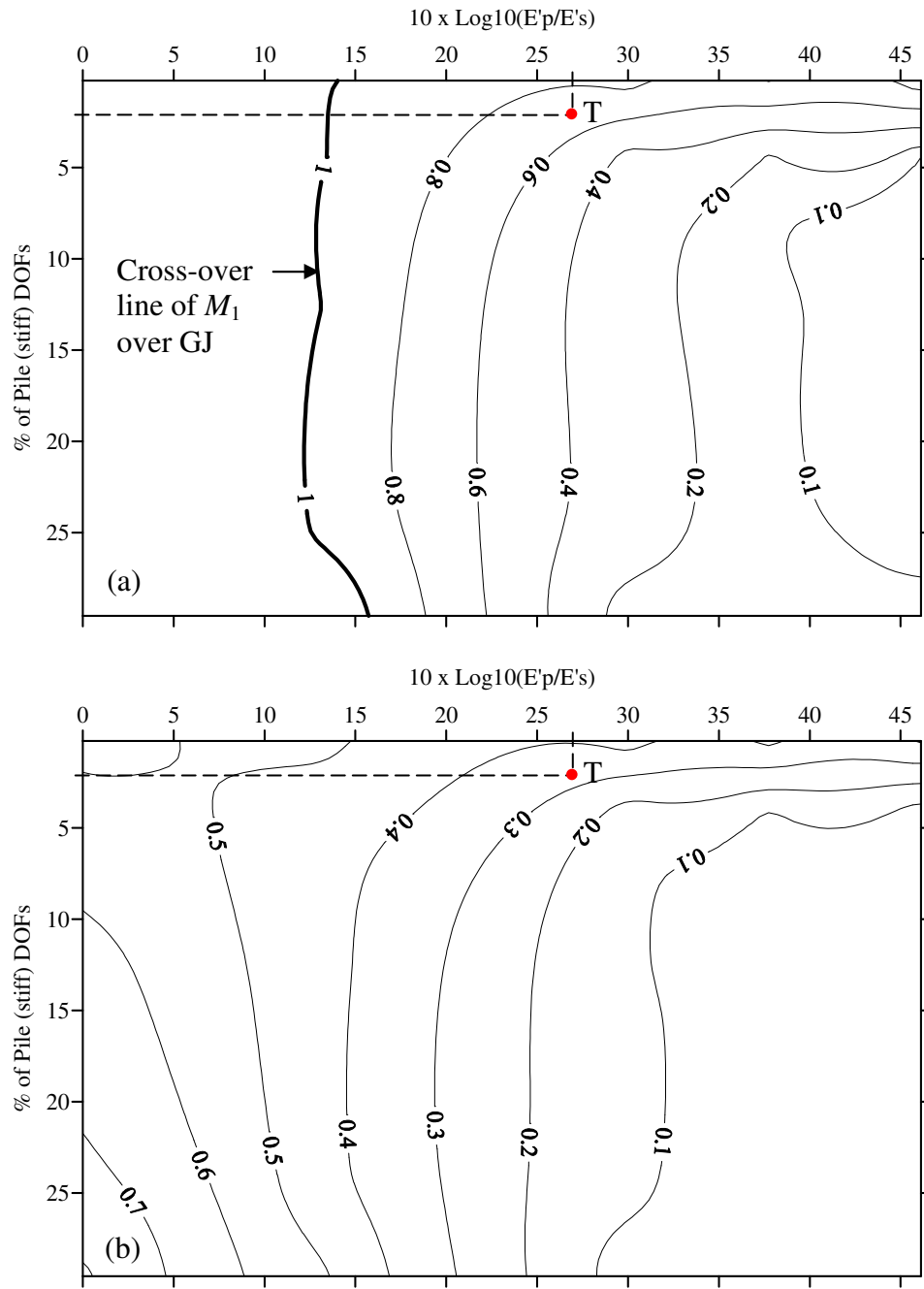


Figure 5.10. CPU time of  $M_1$  and  $M_2$  preconditioners for a range of stiff DOFs and soil-structure stiffness ratios (a)  $M_1$  versus GJ, (b)  $M_2$  versus GJ.

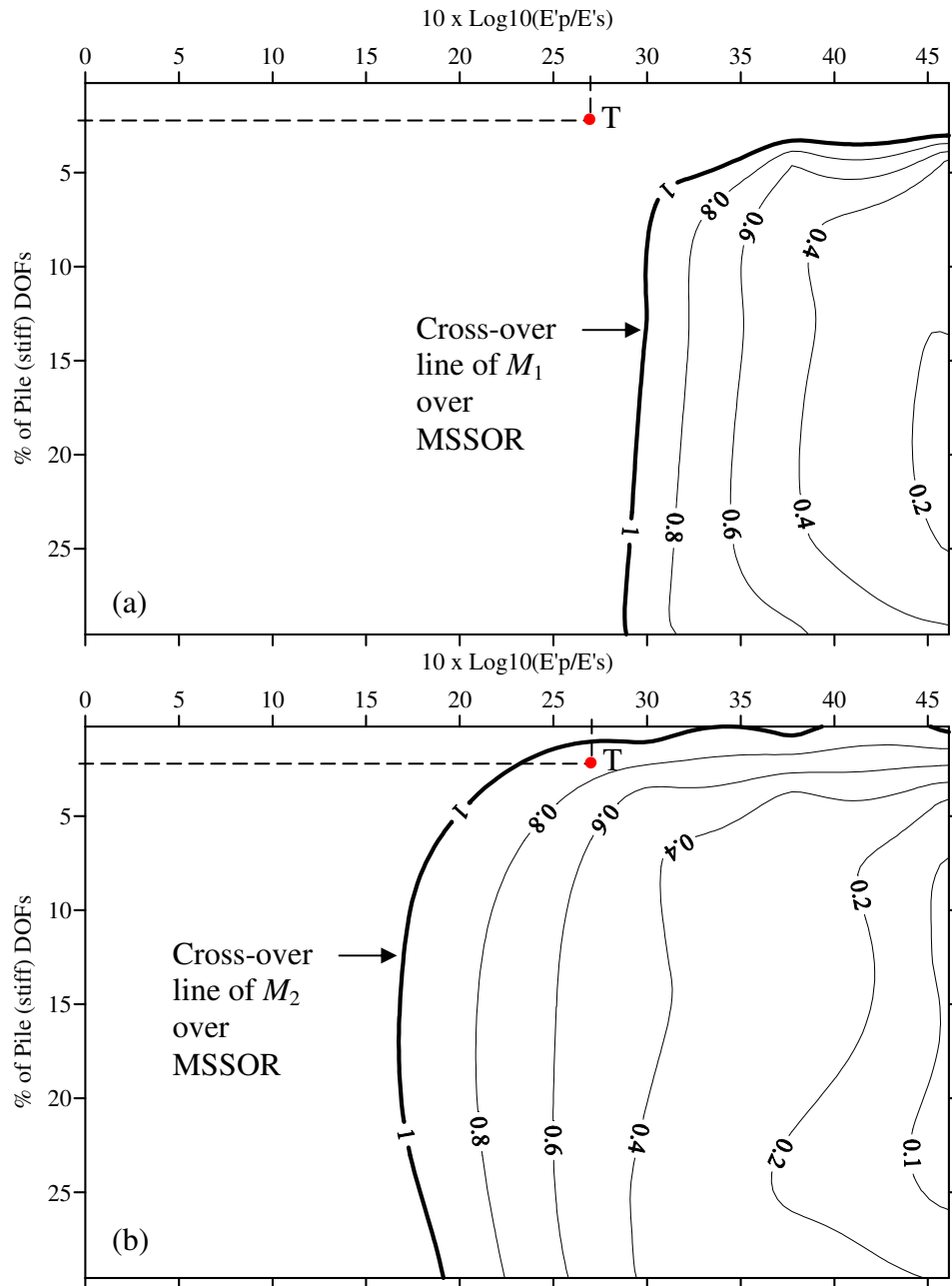


Figure 5.11. CPU time of  $M_1$  and  $M_2$  preconditioners for a range of stiff DOFs and soil-structure stiffness ratios (a)  $M_1$  versus MSSOR, (b)  $M_2$  versus MSSOR.

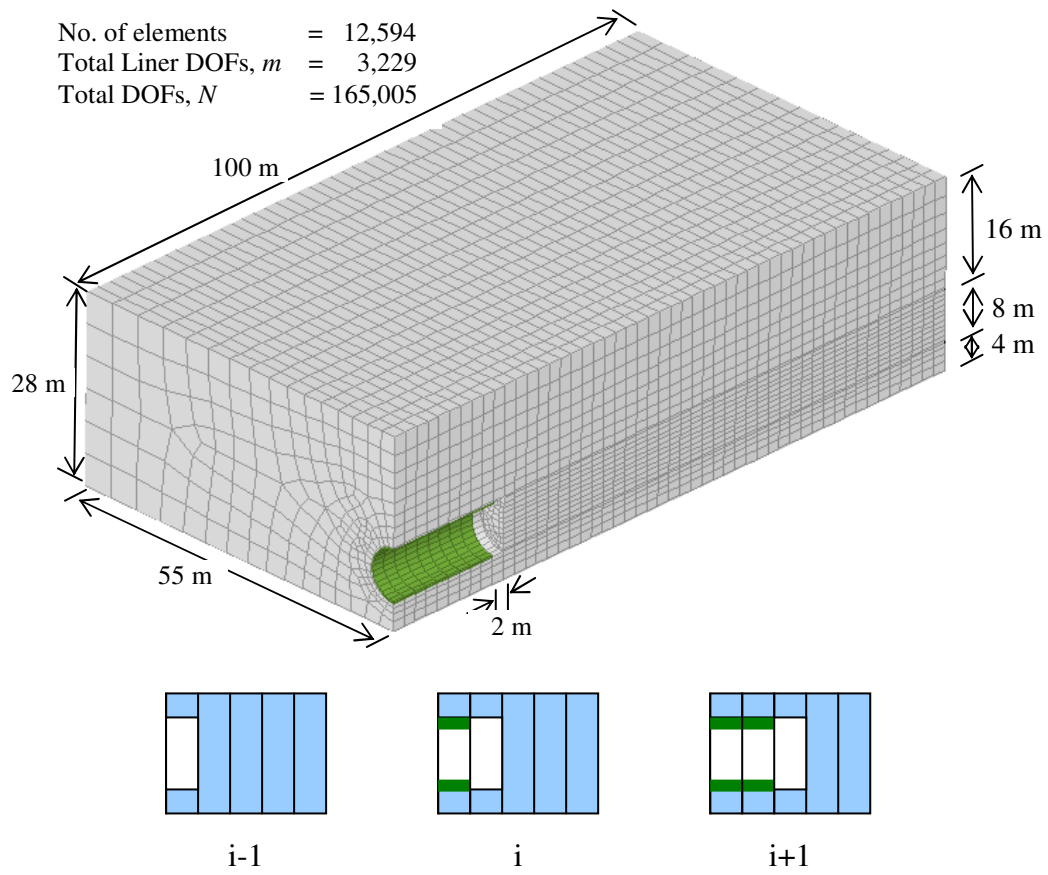


Figure 5.12. Finite element mesh and step-by-step installation of liner in tunneling.

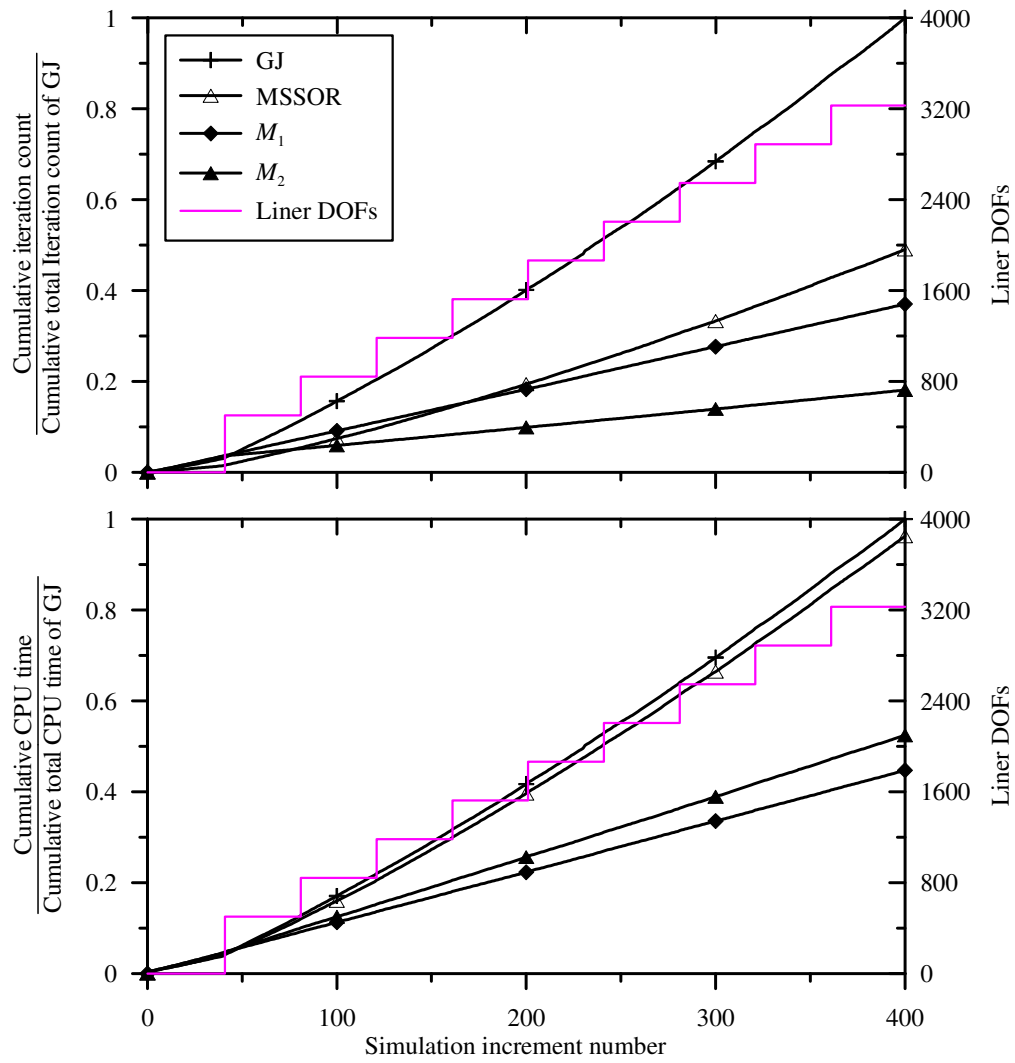


Figure 5.13. Comparison of iteration count and CPU time of the preconditioners for tunneling example.

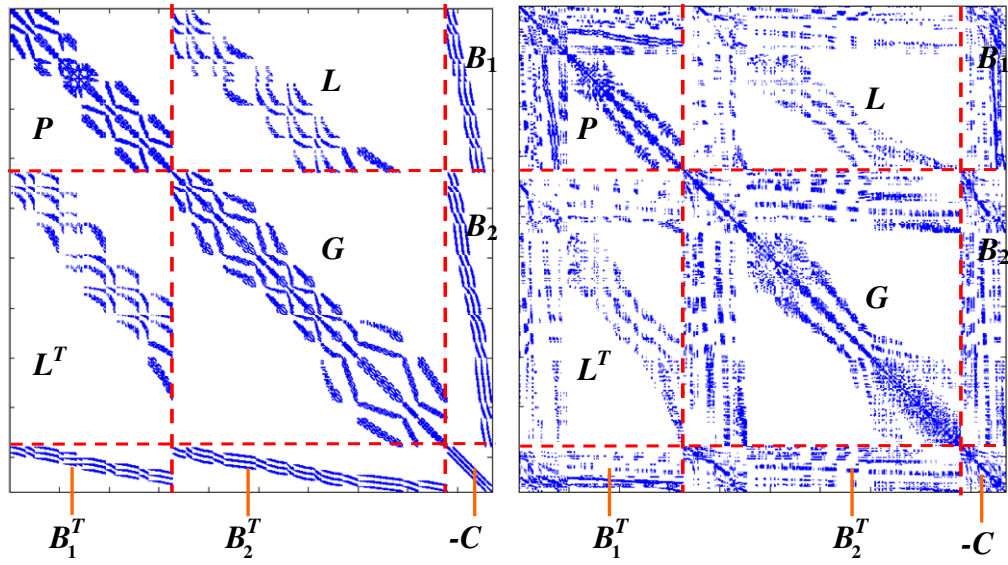


Figure 5.14.  $7 \times 7 \times 7$  mesh: Sparsity pattern of  $3 \times 3$  block structured  $A$ . (a) Sequential nodal numbering of nodes in  $x$ - $z$  plane according to Smith and Griffiths (1997; 2004); and (b) Automatic nodal numbering in GeoFEA.

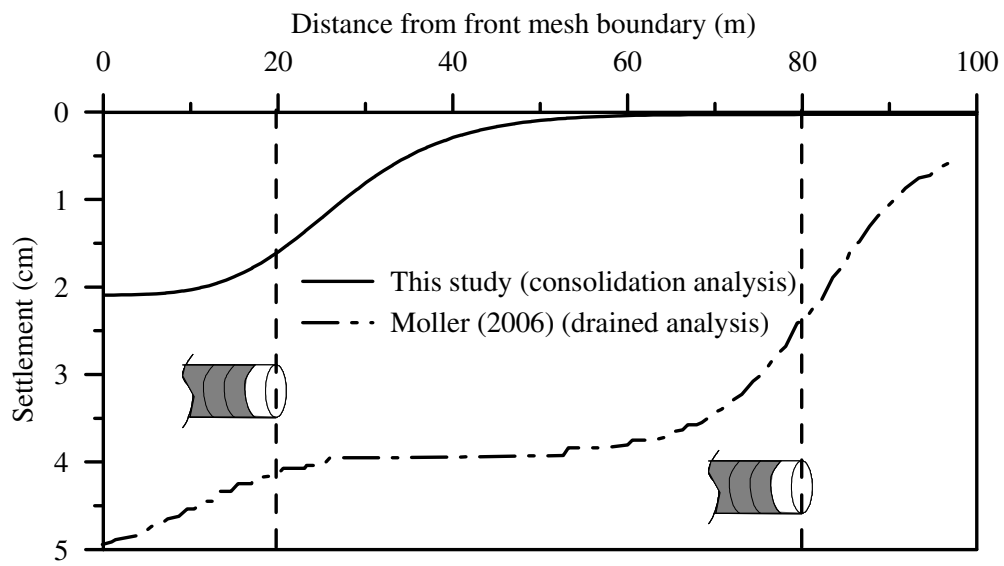


Figure 5.15. Surface settlement profile after 10 steps of excavation.

Table 5.1. Material properties for piled-raft foundation.

(a) To study the effect of pile-soil stiffness contrasts			
Material type	Young's modulus, $E'$ (MPa)	Permeability, $k$ (m/s)	Poisson's ratio, $\nu'$
Pile	100, 5000, 30000, 205000*	$10^{-17}$	0.2
Soil	5	$10^{-9}$	0.3
(b) To study the effect of pile-soil permeability contrasts			
Pile	30000	1, $10^{-9}$ , $10^{-17}$	0.2
Soil	1	$10^{-3}$ , $10^{-9}$ , $10^{-12}$ **	0.3

\* typical of steel piles, \*\* typical of unsaturated soils

Table 5.2. 25×25×35 mesh: Effect of different approximations of block  $P$  with diagonal approximation of blocks  $G$  and  $\tilde{S}$  in the preconditioner (5.28) for a 9-piled raft problem.

$E'_p/E'_s$	Approximation of block $P$	Iteration count	Total CPU time (s)	Total CPU time ratio (%)	$nnzl(P)$ or $nnzu(P)$
1	$\hat{P}_1 = \text{diag}$	2,033	533.30	<b>86.21</b>	
	$\hat{P}_2 = \text{SSOR}$	2,457	650.20	105.10	
	$\hat{P}_3 = \text{ILU0}$	2,283	610.92	98.76	696,409
	ILUT(50,1E-6)	2197	608.64	98.39	1,294,089
	ILUT(100,1E-6)	2225	626.25	101.23	1,802,004
	$\hat{P}_4 = \text{Cholesky LL}^T$	2,247	618.62	100.00	2,383,272
20	$\hat{P}_1 = \text{diag}$	2,398	619.52	98.83	
	$\hat{P}_2 = \text{SSOR}$	2,388	633.09	100.99	
	$\hat{P}_3 = \text{ILU0}$	2,202	592.94	<b>94.59</b>	699,397
	ILUT(50,1E-6)	2272	627.66	100.13	1,307,092
	ILUT(100,1E-6)	2252	634.14	101.16	1,821,992
	$\hat{P}_4 = \text{Cholesky LL}^T$	2,269	626.86	100.00	2,662,570
1000	$\hat{P}_1 = \text{diag}$	7,686	1,868.30	281.83	
	$\hat{P}_2 = \text{SSOR}$	4,535	1,153.53	174.01	
	$\hat{P}_3 = \text{ILU0}$	6,512	1,642.02	247.70	699,327
	ILUT(50,1E-6)	2477	679.20	102.46	1,308,862
	ILUT(100,1E-6)	2219	625.64	<b>94.38</b>	1,825,371
	$\hat{P}_4 = \text{Cholesky LL}^T$	2,416	662.92	100.00	2,516,319
6000	$\hat{P}_1 = \text{diag}$	14,859	3,558.72	538.75	
	$\hat{P}_2 = \text{SSOR}$	8,995	2,234.41	338.27	
	$\hat{P}_3 = \text{ILU0}$	7,822	1,967.69	297.89	699,829
	ILUT(50,1E-6)	6899	1786.19	270.41	1,310,623
	ILUT(100,1E-6)	3372	918.88	139.11	1,812,558
	$\hat{P}_4 = \text{Cholesky LL}^T$	2,413	660.55	<b>100.00</b>	2,399,448
41000	$\hat{P}_1 = \text{diag}$	31,317	7,451.77	990.48	
	$\hat{P}_2 = \text{SSOR}$	17,257	4,237.25	563.21	
	$\hat{P}_3 = \text{ILU0}$	13,654	3,391.73	450.82	699,339
	ILUT(50,1E-6)	9157	2351.27	312.53	1,299,411
	ILUT(100,1E-6)	5227	1390.00	184.76	1,786,296
	$\hat{P}_4 = \text{Cholesky LL}^T$	2,771	752.34	<b>100.00</b>	2,502,579

Note: The ILU subroutines store both the upper and lower triangular factors of a matrix. The  $nnzu$  is the number of nonzeros of the symmetric upper triangular factor for preconditioning. Similarly,  $nnzl$  is the number of nonzeros of symmetric lower triangular factor from Cholesky subroutines.

Table 5.3. Problem statistics of the piled-raft foundations.

Mesh		25×25×35					
No. of elements		21,875					
No. of piles	Raft thick. (metre)	Size of block $P$ -Pile DOFs ( $m$ )	Size of block $G$ -Soil DOFs ( $n$ )	Size of block $C$ -Pore press. DOFs ( $np$ )	Size of $A$ -Total DOFs ( $N$ )	Number of nonzeros $nnz(A)$	$m/N$ (%)
1	-	668	267,012	23,660	291,340	54,299,992	0.23
9	3	12,767	254,913	23,660	291,340	54,299,938	4.38
25	3	36,948	230,732	23,660	291,340	54,300,112	12.68
49	3	72,601	195,079	23,660	291,340	54,300,104	24.92
49	4	86,145	181,535	23,660	291,340	54,327,896	29.57

Table 5.4. Comparison of total CPU times for tunnel construction.

Preconditioners	Ratio of CPU times	
	Actual	From Figures 5.10-5.11 (point T)
$M_1$ vs. GJ	0.45	0.67
$M_2$ vs. GJ	0.46	0.33
$M_1$ vs. MSSOR	0.52	> 1
$M_2$ vs. MSSOR	0.54	0.88

## Chapter 6

---

### APPLICATIONS ON CASE HISTORIES

#### 6.1. Introduction

The study of various preconditioning approaches for drained analysis (Chapter 4) and consolidation analysis (Chapter 5) provides an in-depth understanding of the effect of relative differences in material stiffnesses and permeabilities on the performance of iterative methods. Some effective block diagonal preconditioners have been proposed to mitigate such effects. The cost of such block diagonal preconditioners lies in between simple diagonal preconditioning and incomplete LU factorization preconditioning. However, the application and effectiveness of those preconditioners were illustrated for some idealized homogeneous soil conditions using relatively simple examples. In many practical cases, the soil is non-uniform. One difficulty with non-uniform soil is that the value of soil-structure stiffness ratio is not constant. While there is no reason why those preconditioners cannot be applied to such problems, the effectiveness of the newly proposed preconditioners is still unknown compared with other existing preconditioners for this type of problems. Two case history problems are considered to evaluate the

performance under a more realistic geotechnical engineering context. The first problem is a three-dimensional (3D) finite element analysis of a piled-raft foundation of building Westendstrasse 1, Frankfurt and the second is a twin-tunnel construction in Singapore. These examples are selected to investigate the general applicability of the proposed preconditioners to real-world problems arising in geotechnical engineering.

The geotechnical software package GeoFEA (2006) is used in the present study for the FE simulation of above problems. GeoFEA is a finite element program which can be used for drained, undrained and time dependent analysis of static problems under monotonic loading/unloading conditions (<http://www.geosoft.sg/>). Besides, several inbuilt solvers, an important new feature of GeoFEA is that it allows the users to use their own solvers to solve the linear system of equations. This interface provides maximum flexibility to a user or a researcher in choosing an optimal solution method for the problem at hand. In this study, PCG and SQMR solvers with various preconditioners are implemented as user defined solvers into GeoFEA for drained and consolidation analyses, respectively. Various preconditioners are considered for the comparison of advantages offered by proposed block diagonal preconditioners (Chapter 4 and Chapter 5) over them.

## **6.2. GeoFEA implementation details**

GeoFEA has many inbuilt iterative solvers/preconditioners to simulate the large-scale geotechnical problems. However, for some users, the available solvers may not suit their desired purpose. For this reason, GeoFEA also provides an interface for user defined solvers. The comparisons of

performances of preconditioners in this thesis are based on using this feature of GeoFEA and the coding implementation is given in Appendix F. A tutorial manual to use this feature is detailed in the subsequent Section. In this thesis, the user defined solver option is adopted for the following two reasons:

1. Non-availability of a particular preconditioner and/or solver or the difference in implementation. Hence, it is difficult to make a direct comparison of the performance (CPU time, etc.) with that from the proposed preconditioners. For example, the available inbuilt SJ-PCG [PCG solver in conjunction with Standard Jacobi (2.8) preconditioner] in GeoFEA is implemented to perform the matrix-vector multiplication using element-by-element (EBE) approach (Section 2.2.5), which is different than the approach (global assembly) considered in this research work.
2. Differences in the incorporation of boundary conditions.

As in many finite element packages, GeoFEA also uses penalty method for the prescribed boundary conditions. For more details about the penalty method, the reader is referred to Britto and Gunn (1987). However, by using the penalty method, we are solving some unknowns which we actually know. For example, zero displacement is imposed along the sides and base of the finite element mesh for fixed boundaries. Thus, the size of the linear system ( $N$ ) is same as the total number of degrees of freedom including all fixities (NDF).

An alternate way to incorporate boundary conditions exists, known as the elimination method. In this method, for the prescribed variable values, the corresponding rows and columns in the global stiffness matrix are eliminated by modifying the right hand side vector with the prescribed values multiplied

by column entries of those rows in the stiffness matrix (see, e.g. Chen and Phoon, 2009). However, one difficulty in the above approach is when the prescribed values of variables are non-zeros. A hybrid method is widely used where equation components associated with prescribed zero value of variables (homogeneous boundary conditions) are not assembled into the global stiffness matrix, while a penalty method is used whenever the prescribed values are non-zero. Thus, a smaller size of linear system (smaller than NDF) is solved for the same problem, i.e. only the non-zero nodal values are being solved. An example of implementation of such a hybrid approach is the finite element codes of Smith and Griffiths (1997). The same approach to incorporate the boundary conditions has been implemented in user defined solvers for GeoFEA application. The treatment of boundary conditions and counting of degrees of freedom (DOFs) including previous Chapters follow the same strategy. See Appendix F for the implementation details. The procedure to use user defined solver in GeoFEA is explained in the next Section.

The incorporation of boundary conditions by this hybrid approach may reduce the size of  $A$  (1.1) by about 10%. While for a small linear system (e.g. from 2D analysis), the reward gained by such a reduction in the size of  $A$  may be insignificant, it may contribute a big difference in the iterative solution of large linear systems (e.g. from 3D analysis), such as in the CPU time and memory requirement. More importantly, it may sometimes help to reduce the ill-conditioning of the problem to avoid a possible premature breakdown of the iterative solvers (for example, see Section 6.3.1.3).

### 6.2.1. Tutorial manual

GeoFEA 'PROJECT SETUP' window does not have an option to select 'user defined solver'. The user defined solver interface of GeoFEA can be used in the following way:

1. Users can create a DLL file (USOLV.DLL) for their desired solver (with preconditioner). A sample FORTRAN code (USOLV.F90) for the 'user defined solver' is provided with the software package, which is located at 'C:\Program Files\GeoFEA\Usolv' upon installation. Users are encouraged to modify this code according to their desired solver and preconditioner. To do this, it is assumed that the user has a FORTRAN compiler in his/her PC. This original source code with some modifications to incorporate the boundary conditions, as discussed in the previous section, is provided in Appendix F.
2. Other supplementary FORTRAN files required to compile this DLL file are 'XFLOGM.F90' and 'resource.fd'. These files are needed for some of the functions related to dialogues and convergence history plotting routine in the main subroutine 'UDSOL'.
3. For the stiffness matrix formulation, GeoFEA provides an option to select either element-by-element storage scheme (element stiffness matrices are not required to be assembled) or global assembled sparse storage scheme. Only the latter option is used in the present work.
4. Place the new USOLV.DLL in the directory 'C:\Program Files\GeoFEA'.
5. Create the finite element model with all assignments and boundary conditions as is for other inbuilt solvers.

6. In the 'SOLVE' window, check the box beside 'Generate input files only' and click on 'OK' button. This will generate three input files (geosoil.gad, geosoil.gpd, and geosoil.cnn) at 'C:\Program Files\GeoFEA'.
7. Open the 'geosoil.gad' file using any text editor (such as Notepad/WordPad) and change the very first integer to 99. This is the only change needed by the user to use user interface solver.
8. Go back to the 'SOLVE' window and check the box beside 'Use existing input files (geosoil.gpd, geosoil.gad)'. Click on 'OK' to solve the problem using the user defined solver.

### **6.3. Applications on case histories**

Two representative soil-structure interaction problems are considered where the large relative differences in stiffnesses usually result in ill-conditioned systems (Mroueh and Shahrour, 1999; Lee *et al.*, 2002; Chen *et al.*, 2007). The studied problems include piled-raft foundation (loading type), tunneling (unloading type) with a large number of structural elements and nonhomogeneous soil properties.

Similar to piled-raft construction in urban areas, it is often required to construct new tunnels in close proximity of to existing tunnels. For example, the construction of twin running tunnels in the North East Line Project, Singapore (Lee *et al.*, 2006), twin running tunnels in Jubilee Line Extension Project in London, UK (Harris *et al.*, 1996). The interaction effect due to a new construction of new tunnel in close proximity to existing tunnel is likely to be three-dimensional in nature (Ng *et al.*, 2004). Although plane strain

analysis can be employed to study the interaction effects (e.g. Addenbrooke and Potts, 2001; Hage Chehade and Shahrour, 2008), the results are dependent on the assumed volume loss. The conventional superposition of Greenfield values from individual tunnels may produce erroneous results (Phoon *et al.*, 2006), thus the 3D analysis is necessary. Similar to piled-raft foundation, a tunnel construction also involves significant stiff structural elements such as concrete lining, steel anchors, etc. used for stabilization purposes. However, contrary to the piled-raft analysis, the analysis of tunneling construction involves a repeated cycle of unloading (excavation of soil) and loading (application of grout pressure, installation of liner, etc.) conditions. Also, the number of structural elements (e.g. lining elements) increases incrementally in each step of tunnel advancement.

### **6.3.1. Case study 1 – Piled-raft foundation in Germany**

Many high-rise buildings in Frankfurt, Germany were founded on piled-rafts and extensive observations were made of the behavior of the foundations (Franke *et al.*, 1994; El-Mossallamy and Franke, 1997; Franke *et al.*, 2000; Katzenbach *et al.*, 2000). One of those buildings was Westendstrasse 1. The piled-raft foundation of 208 m tower consists of the raft of an approximately 47 m  $\times$  62 m in area with a thickness 3 m at the edges and 4.65 m in the central part as shown in Figure 6.1. The pile configuration below the main tower is also shown. There are 40 bored piles of length 30.0 m below the raft and of diameter 1.3 m. For the 3D finite element analysis using GeoFEA, the entire problem (soil and foundation) is discretized using 20-noded hexahedron elements (Figure 6.2). This results in 19,854 elements, 86,923 nodes, and 239,040 displacement degrees of freedom (DOFs). In the model, a uniform

raft thickness of 4.5 m is considered and the circular piles are replaced by square piles with the same shaft circumference. This results the pile displacement DOFs of 59,259 ( $\approx 25\%$  of total). Note that the pile DOFs also accounts the raft DOFs because the same material is assumed for both. The piled-raft is assumed to be a reinforced-concrete structure and modeled as linear elastic. The following properties are assigned:  $E'_p = 24,822$  MPa,  $\nu' = 0.2$ ,  $\gamma_{\text{conc}} = 22$  kN/m<sup>3</sup>, which are similar to the parameters adopted in (Novak *et al.*, 2005).

The subsoil condition in Frankfurt am Main, Germany, consists of tertiary soils and rock. The soils at Westendstrasse 1 site consist of 6-8 m thick bed of gravely sand followed by an overconsolidated stiff plastic clay known as Frankfurt clay. The ground water table lies about 5-7 m below the ground surface. The general stratigraphy together with values of undrained shear strength ( $c_u$ ) is shown in Figure 6.3. Only the soil below the foundation level (i.e. Frankfurt clay) is modeled with finite elements. Properties of Frankfurt clay that are used in the modeling are given in Table 6.1, which are based on data presented in (Franke *et al.*, 2000). Mohr-Coulomb elastic- perfectly plastic soil model is used for the analysis. Settlement behavior of Frankfurt clay using other constitutive models have been studied, e.g. Duncan and Chang model (Franke *et al.*, 2000), Cap model (Reul and Randolph, 2003). However, the main focus of this study is not the constitutive models.

The base of the mesh is assumed to be fixed in all directions. Side face boundaries are constrained in the transverse directions but free in in-plane directions. The piled-raft is assumed wished-in-place before the start of simulation. A load of 1,000 MN is applied to the top of the piled raft in 4 steps

and each step is simulated in 3 increments to take into account the nonlinear soil behavior. These loads are converted to an equivalent pressure on the top of the raft using an approximate surface area of 47 m by 62 m (2914 m<sup>2</sup>). The contact between structure and soil is assumed to be perfectly rough. This means no interface elements are considered in between the soil and raft or soil and pile. Settlement analysis and preconditioners' performance comparison in both drained and consolidation conditions are studied.

#### **6.3.1.1. Inbuilt versus user defined solvers for drained analysis**

For drained analysis of a problem, PCG in conjunction with SJ (2.8) is available in GeoFEA. Although SQMR solver available in GeoFEA can also be used for drained problems, only PCG is employed for all the drained problems throughout this work. This is because PCG is known to be the best for symmetric positive definite linear systems (Barrett *et al.*, 1994).

As mentioned in Section 6.2, the inbuilt solver solves the linear system of size  $N = 260,769$  (= NDF), while the user defined solvers only need to solve the linear system of size  $N = 239,040$  (8.33% smaller than NDF) for the given problem because of the difference in incorporation of boundary conditions. This could be a factor for about 45% less CPU time (Figure 6.4) by the user defined solver for the same iteration counts. However, as mentioned earlier, this could also be partly due to different storage schemes for the stiffness matrix. Inbuilt SJ-PCG uses the EBE storage scheme whereas user defined SJ-PCG uses global assembly of the stiffness matrix.

Figure 6.5 shows that the settlements due to all solvers are identical, indicating no loss of accuracy in the results due to smaller linear systems in the user defined solvers. As shown in Figure 6.6, there is a good agreement

between the settlement computed by GeoFEA with preconditioned iterative methods and that from the field measurements. Note that the direct solution of the same problem could not be obtained in the given computing configuration owing to the limited core memory.

### **6.3.1.2. Performance of preconditioners**

This Section compares the performance of preconditioners only in the form of user defined solvers. Figure 6.7 shows the performance of various preconditioners in conjunction with PCG for drained analysis of the problem. The SBD preconditioners are more than 6 times faster in terms of iteration counts and about 1.7 times faster in terms of CPU time than the conventional SJ and SSOR preconditioners. Note that the problem involves a significantly large structural block  $P$  [Equation (4.25)] due to the presence of a thick raft with 40 bored piles. Hence, the Cholesky factorization of block  $P$  is expensive, but the SBD preconditioners are yet superior (in terms of CPU time) than others. Note that the soil-structure stiffness ratio is not unique and lies in range of 160 to 3,546 in the present problem because the stiffness of the soil increases with depth. This indicates the robustness of the SBD preconditioners. However, relatively slower performance of  $SBD_2$  (in terms of CPU time) in comparison to  $SBD_1$  and SSOR in comparison to SJ (Figure 6.7) could be due to a large triangular solution of the blocks. The sparsity pattern of the coefficient matrix may have contributed to this difference, as mentioned in Chapters 4 and 5.

A parametric study of homogenous soil profiles (Figure 6.8) with three different stiffnesses, namely, the minimum, maximum, and average value of Young's modulus of the actual Gibson soil profile suggests that the effective

soil-structure stiffness ratio is somewhat closer to that with an average soil modulus. This finding may help to roughly interpret the soil-structure ratio in the case of nonuniform soils and gives the rough idea on expected saving in CPU time by SBD preconditioners over conventional SJ or SSOR preconditioners. It generalizes that the SBD preconditioners are effective as long as stiffness contrast of the materials exists, the nonhomogeneity of the soil does not affect the characteristic performance of SBD preconditioners, thus the robustness of the preconditioner. As expected, the saving in CPU time by the SBD preconditioner is the highest (about 3.5 times) when the value of soil modulus is the lowest (7 MPa). This is because of the larger soil-structure stiffness ratio. Thus, when the soil is soft, a significant saving in CPU time by SBD preconditioners can be achieved, reinforcing the findings of Chapter 4.

#### **6.3.1.3. Inbuilt versus user defined solvers for consolidation analysis**

For many tall buildings in Germany, almost 70% of the final settlement occurred during the construction time and the settlement had completely stopped after three years of completion of construction of buildings (Breth and Amann, 1975). In view of modes of construction and thick layer of Frankfurt clay, the time dependent settlement of multi-storey buildings becomes particularly important. On the other hand, consolidation analysis requires more computational effort than the drained analysis does. This may be a reason that most of the published literatures have usually considered a single-phase material in their analyses (e.g. Reul and Randolph, 2003; Novak *et al.*, 2005). This Section demonstrates the settlement, and differences in performance of

inbuilt and user defined solvers (SQMR) with various preconditioners for the consolidation analysis of the problem.

The same finite element mesh as shown in Figure 6.2 is used for the analysis. The ground water table is considered to be on the top surface of the mesh since the piled-raft was constructed below the ground water level (Franke *et al.*, 2000). To consider the pore pressure variation, a 20-noded hexahedral element coupled with 8-noded pore pressure nodes is used. This leads the total degrees of freedom (DOFs) to be 259,173 with 20,133 pore pressure DOFs. The number of pile and soil displacement DOFs will remain the same as in drained analysis. The time dependent application of load on the piled-raft is as shown in Figure 6.9. Permeability of the piled-raft is taken as  $1 \times 10^{-17}$  m/s.

As shown in Figure 6.10, the inbuilt GJ-SQMR may sometimes fail to converge for some problems. Because the same finite model is used for both inbuilt and user defined solvers, it is suspected that this could have resulted from the differences in the way boundary conditions are incorporated between inbuilt and user defined solvers, as mentioned in Section 6.2. Similarly, another advantage of using hybrid boundary conditions is the saving in runtime. For example, the user defined MSSOR-SQMR solver is about 20% faster than the inbuilt MSSOR-SQMR (Figure 6.10). This is because a smaller size of the linear system ( $N = 259,173$ , 8.46% smaller than NDF) is solved with the user defined solvers than with the inbuilt solvers ( $N = \text{NDF} = 283,139$ ) for the same problem.

Similar to the drained case, the settlement from all the user defined solvers are identical (Figure 6.11). The computed settlement is compared with

the observed settlement as shown in Figure 6.12. The computed consolidation settlement is sufficiently close to that of the measured ones when the permeability of the soil  $k_s = 1 \times 10^{-7}$  m/s for the loading shown in Figure 6.9. Note that the settlement from consolidation analysis is affected by the two factors: (i) permeability of the materials, and (ii) consolidation time. Drained condition is the end of consolidation stage, which is achieved when either the permeability of the material is very high or the consolidation time is very long. In the studied problem, the computed consolidation settlements with soil permeability  $k_s = 1 \times 10^{-6}$ - $1 \times 10^{-7}$  m/s are identical with that from the drained analysis, except for the first time step. The smaller consolidation settlement in the first time step is because the first load is applied immediately as shown in Figure 6.9. In the numerical simulation, the time step for this step is taken as 1 day. Thus, it may represent the condition close to an undrained state. However, if the soil permeability is taken as  $k_s = 1$  m/s, the consolidation settlement profile is identical to the drained one (Figure 6.12) even though the time step is small (i.e. the same 1 day). Thus, it suggests that one can use Biot's consolidation equations to reproduce the drained or undrained condition depending on the value of time step and permeability of the materials used. But, from the view point of CPU time, there is some penalty using consolidation equations (indefinite linear system) to solve the ideal drained analysis (positive definite linear system, see Chapter 4) as shown in Figure 6.13. The computation of drained analysis (solving the positive definite linear system with PCG solver) may take about 30-45% less time than the consolidation analysis (SQMR solver with similar preconditioners) for the same final result. This is because the PCG solver for drained analysis

converges at much fewer iteration counts (about 50% less in terms of iteration counts) than the SQMR for the corresponding consolidation analysis. However, for undrained analysis with Poisson's ratio close to 0.50 (incompressible problems), consolidation analysis with low permeability and small time step is recommended (see, for example, Phoon *et al.*, 2003).

#### **6.3.1.4. Performance of preconditioners**

The comparison of preconditioners (only user defined ones) in conjunction with SQMR (Figure 6.14) demonstrates that although  $M_2$  offers no significant benefit over  $M_1$ , both preconditioners are about 2 times faster than the GJ and MSSOR counterparts for the given soil condition. This is mainly because of the mitigation of material ill-conditioning by  $M_1$  and  $M_2$  preconditioners (Chapter 5). This can also be interpreted from the iteration count plot in Figure 6.14, where the cumulative plot of iteration counts and CPU times for  $M_1$  and  $M_2$  preconditioners increases linearly. The smaller gradients of these profiles indicate the robustness and applicability of  $M_1$  and  $M_2$  preconditioners for real-world soil-structure interaction problems.

#### **6.3.2. Case study 2 – Tunneling in Singapore**

The tunneling problem studied herein is the North East Line (NEL) tunnel, contract C704, where twin rail tunnels were driven using 9 m long Earth Pressure Balance (EPB) machines. The tunnel extrados diameter was 6 m and the springline of the tunnel varies from a depth of 18 m to 21 m below the ground surface. In the finite element analysis, the springline depth of 21 m is considered. The finite element mesh used for the analysis is as shown in Figure 6.15. The lateral boundary is set at 10 times the tunnel diameter to

avoid boundary effects. A length of 90 m was modeled in the longitudinal direction. Thus the front and back boundaries are located at 7.5 times the diameter of tunnel from the monitored section. The entire problem including the soil and structural parts of tunnel is modeled using 20-noded brick element coupled with 8-noded fluid elements (see Figure 3.1) resulting in a total of 14,640 elements, 57,677 nodes, and 187,880 unknown DOFs.

The vertical side of the mesh is restrained against transverse movement whilst the base is completely fixed. The water table is located at 5 m below the ground surface. The ground condition consists of completely weathered Granite or residual soil with a weathering grade between V and VI (known as G4 type), which behaves more like an over-consolidated soil with over-consolidation ratio (OCR) of about 3. The soil behavior was modeled with associated Mohr-Coulomb model and the parameters adopted are presented in Table 6.2, which are similar to the ones adopted in (Lee *et al.*, 2006). Equivalent soil stiffness were derived based on the Unconsolidated Undrained test results, which gives  $E_u/c_u = 400\sim 480$ . More details of soil properties are explained elsewhere (Lim, 2003).

The concrete tunnel lining is assumed to be impervious. In C704, the overcutting is about 0.5% of the face area (Shirlaw *et al.*, 2001). This is approximately equivalent to an all-round 75-mm gap between the excavated tunnel and the tail skin shield. In the finite element model, compressible “grout” elements of thickness 100 mm are used surrounding the lining elements to fill up the gap between the shield overcut and the tunnel diameter. Both the concrete lining and grout are modeled as elastic materials and the parameters used are summarized in Table 6.3.

Simulation of the tunnel involves a repeated soil excavation and lining installation steps. In the first step, the tunneling process is simulated by removing the soil elements, 6 m in length, followed by application of pressure (due to Shield machine or grout) against exposed surface. A tunnel advance rate of 4 m/day is adopted, this being an average advance rate for tunnels in the studied section. The actual tunnel advance rate varies from 3 m/day to 10.5 m/day (Pang, 2006). In the next step, the concrete lining and grout elements are installed by activating the lining and grout elements. The previously applied pressure is removed simultaneously. This simulation method follows the method adopted by other researchers (Chan, 2002; Möller and Vermeer, 2008) for the tunnel construction. A space of 9 m (size of Shield machine) between the tunnel face and the liner is maintained by the application of 150 kPa pressure against the exposed surface. The finite element steps of tunnel construction are shown in Figure 6.16. To account for soil non-linearity, each excavation step is modeled with 20 increments.

As shown in Figure 6.17 the performance of GJ preconditioner deteriorates rapidly (with total CPU time  $\approx 139$  hours) as the excavation proceeds. However, the linear cumulative curve for  $M_1$  indicates a stable convergence (with total CPU time  $\approx 30$  hours) for each excavation step and about 4 times saving in runtime. Note that the number of stiff structural elements (e.g. lining elements) increases incrementally in each step of tunnel advancement. This is shown in Figure 6.17 as “Liner DOFs”. Thus, the above result suggests that the nonhomogeneity of soil does not hinder the effectiveness of the  $M_1$  preconditioner as long as the stiff block  $P$  exists. Similarly, the effectiveness of the  $M_1$  preconditioner becomes more apparent

as the model accumulates more stiff (liner) elements in the system due to tunnel advancement. For brevity, only the results of GJ and  $M_1$  preconditioners are shown in Figure 6.17 as the pair MSSOR and  $M_2$  follows a similar trend. This suggests the general applicability of  $M_1$  and  $M_2$  preconditioners to realistic subsurface problems.

Figure 6.18 shows the computed settlement trough from the above 3D analyses. The measured values of settlements (e.g. Pang, 2006) are also included for the comparison purpose. As seen in the Figure, the computed settlements closely match the measured ones. It should be noted that the computed settlement is affected by the soil model, shield/grout pressure, construction method adopted, etc. In the field, the settlement is also affected by various other factors such as operational and human factor. For a parametric study on these, the reader is referred to (Lim, 2003; Pang, 2006).

## 6.4. Conclusions

GeoFEA can be used to simulate complex geotechnical problems, while allowing users to incorporate their own specific implementation of iterative solvers. The numerical results suggested that the nonhomogeneous soil, which is present in almost all realistic problems, is not a problem for the convergence of the recently proposed inexact block diagonal preconditioners ( $SBD_1$ ,  $SBD_2$ ,  $M_1$ , and  $M_2$ ). Their convergence behaviors are comparable to those presented for homogeneous problems (Chapters 4 and 5). In other words, the ill-conditioning problem associated with significant contrast in material stiffness has been solved. The effectiveness of the proposed preconditioners increases as the system accumulates more and more stiff elements. For example, the

tunneling example illustrates how stiff elements increase with the installation of liner elements. Thus, the proposed preconditioners have general applicability to realistic problems involving geometrically complex soil-structure interaction problems in complex geological conditions.

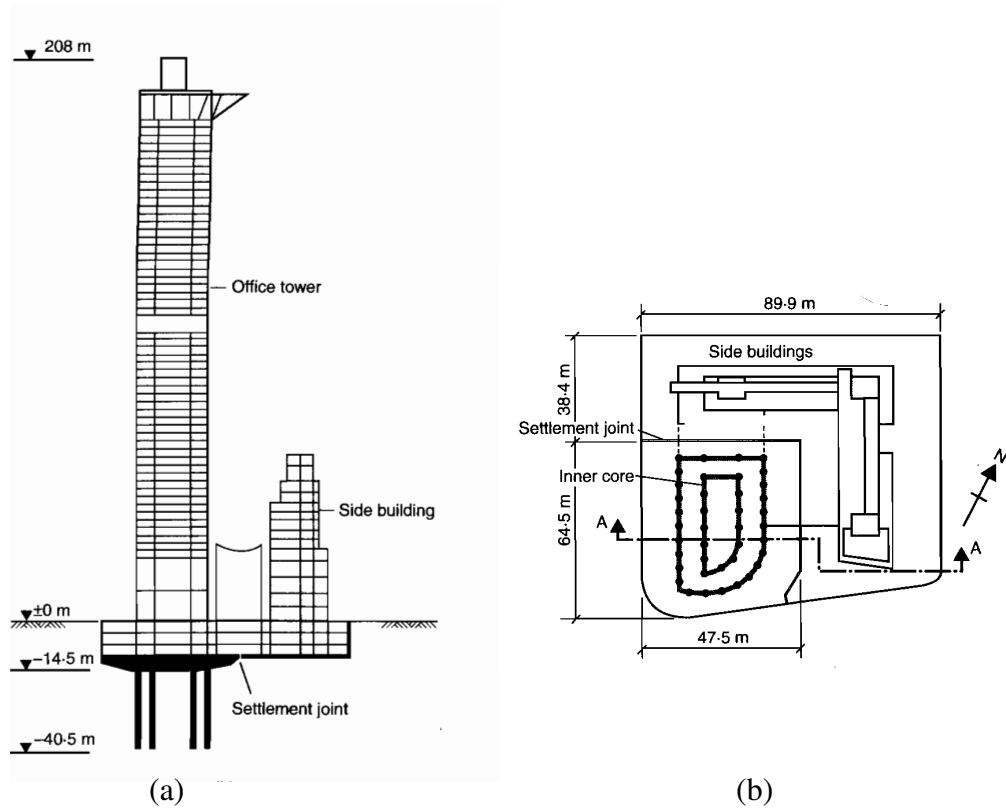
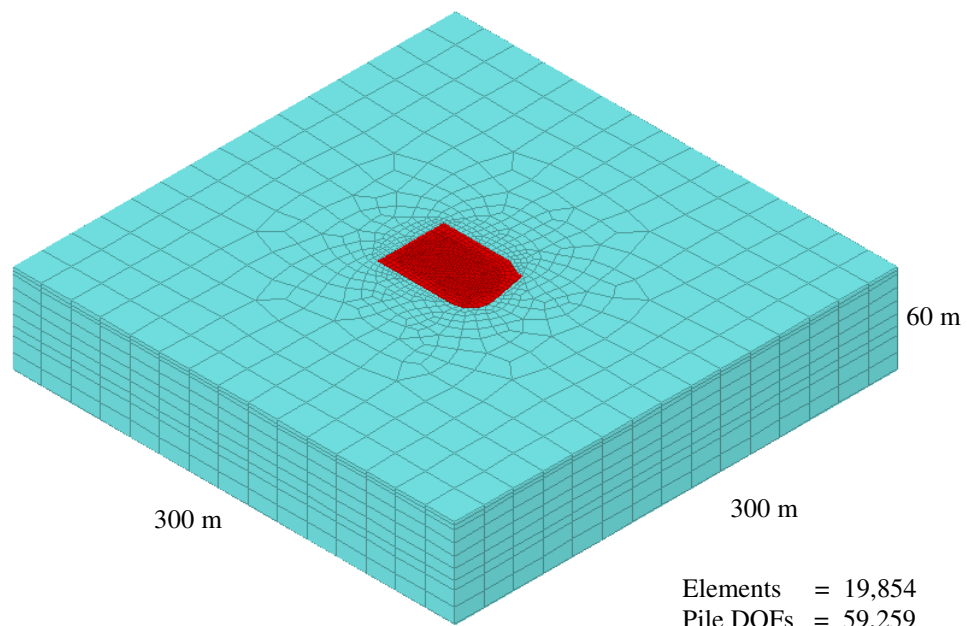
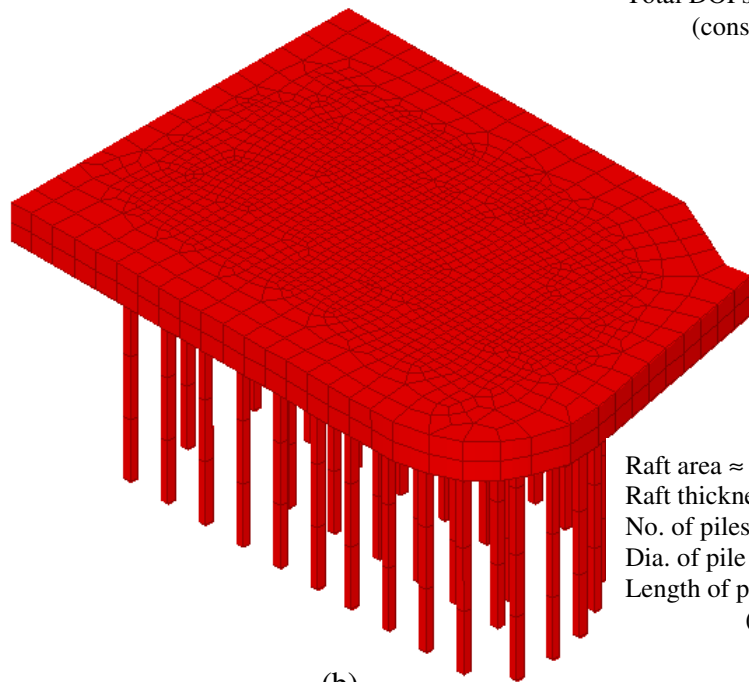


Figure 6.1. Westendstrasse 1 building, Frankfurt: (a) Sectional elevation (after Katzenbach *et al.*, 2000); and (b) Plan with pile layout (after Franke *et al.*, 2000).



(a)

Elements = 19,854  
 Pile DOFs = 59,259  
 Total DOFs = 239,040  
 (drained analysis)  
 Total DOFs = 259,173  
 (conso. analysis)



(b)

Raft area  $\approx 47 \times 62 \text{ m}^2$   
 Raft thickness = 4.5 m  
 No. of piles = 40  
 Dia. of pile = 1.3 m  
 Length of pile = 30 m  
 (below raft)

Figure 6.2. Finite element meshes (a) mesh for entire problem domain, and (b) enlarged mesh for piled-raft.

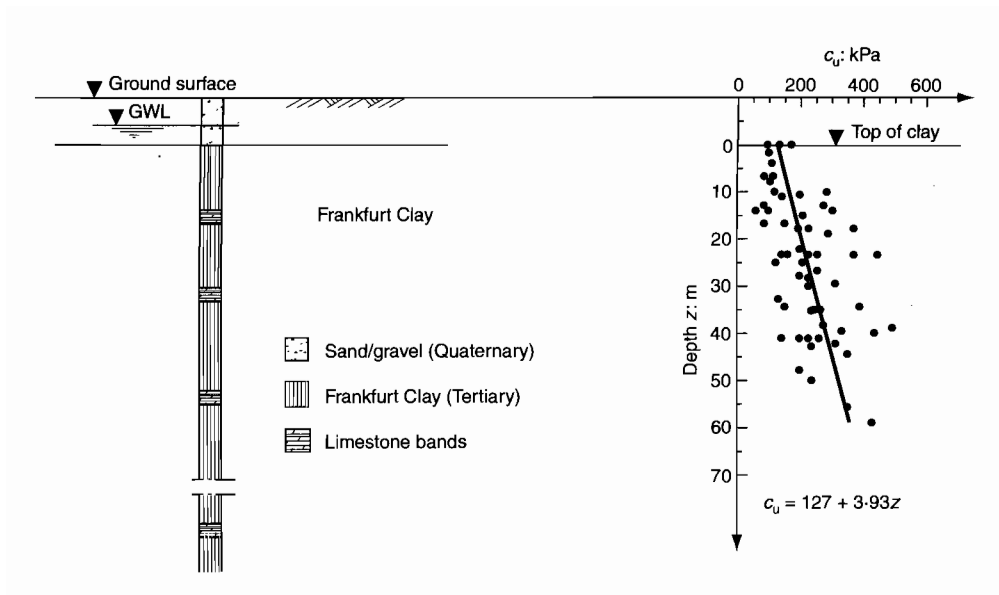


Figure 6.3. Frankfurt subsoil stratigraphy and undrained shear strength (after Franke *et al.*, 2000).

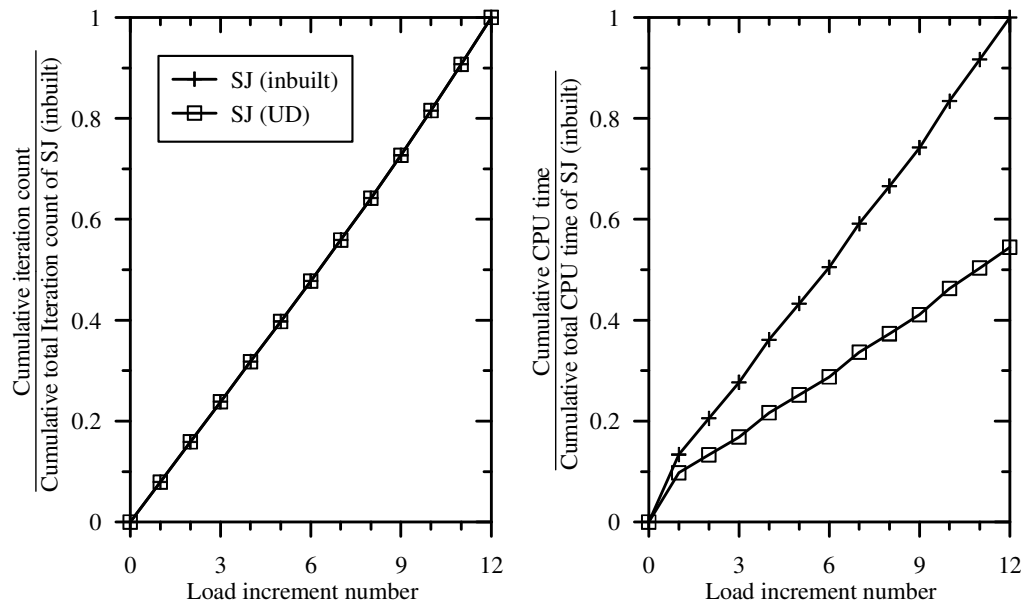


Figure 6.4. Comparison of performance of SJ (inbuilt) and SJ (user defined) preconditioners with PCG.

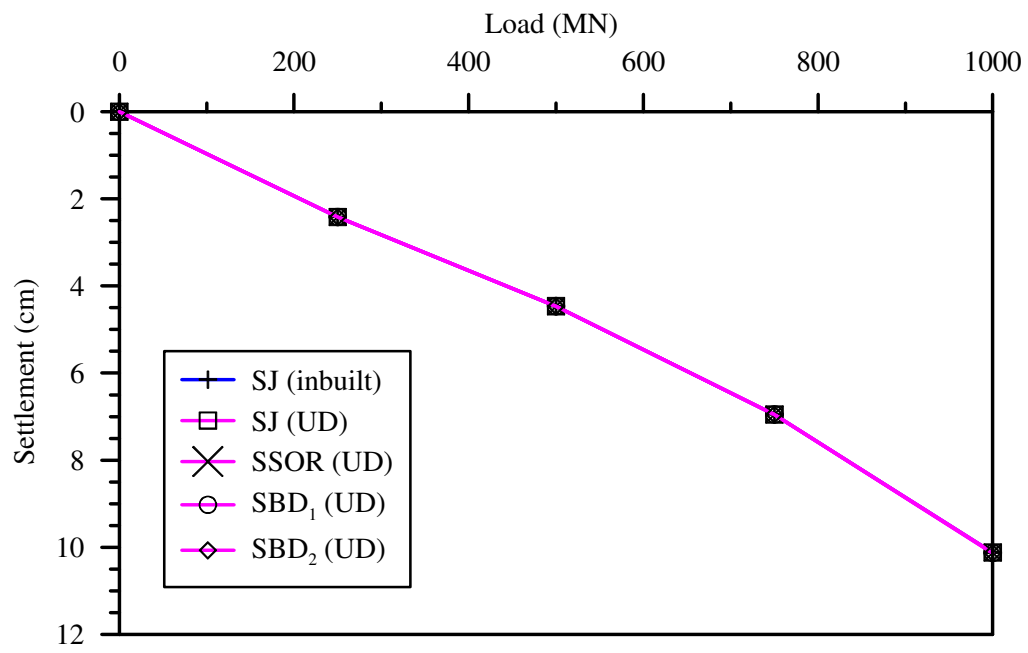


Figure 6.5. Settlement due to different preconditioners with PCG.

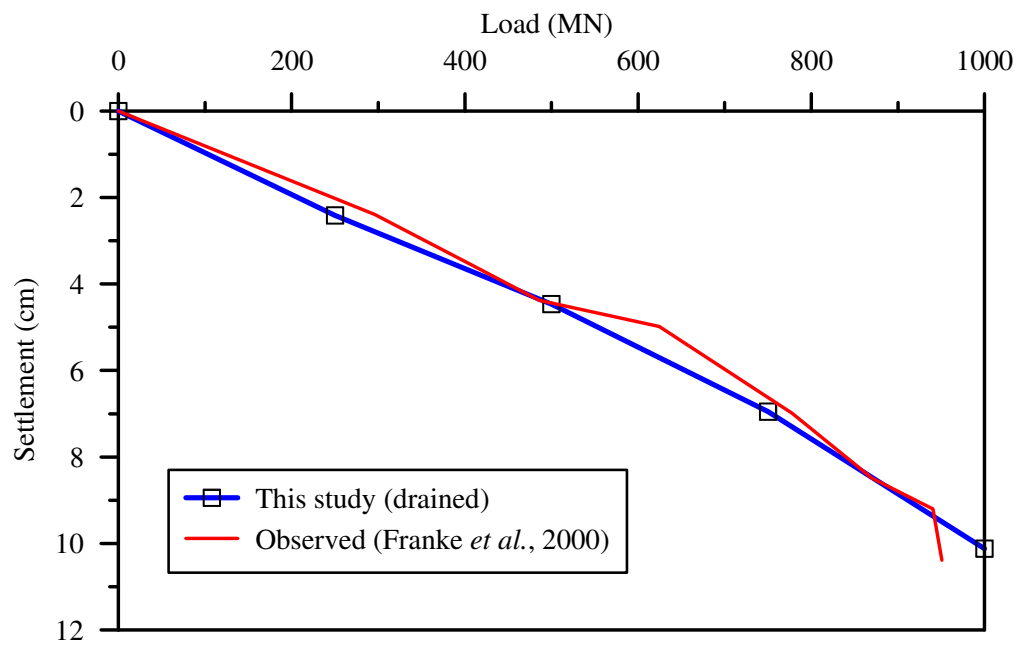


Figure 6.6. Comparison of computed and measured settlements.

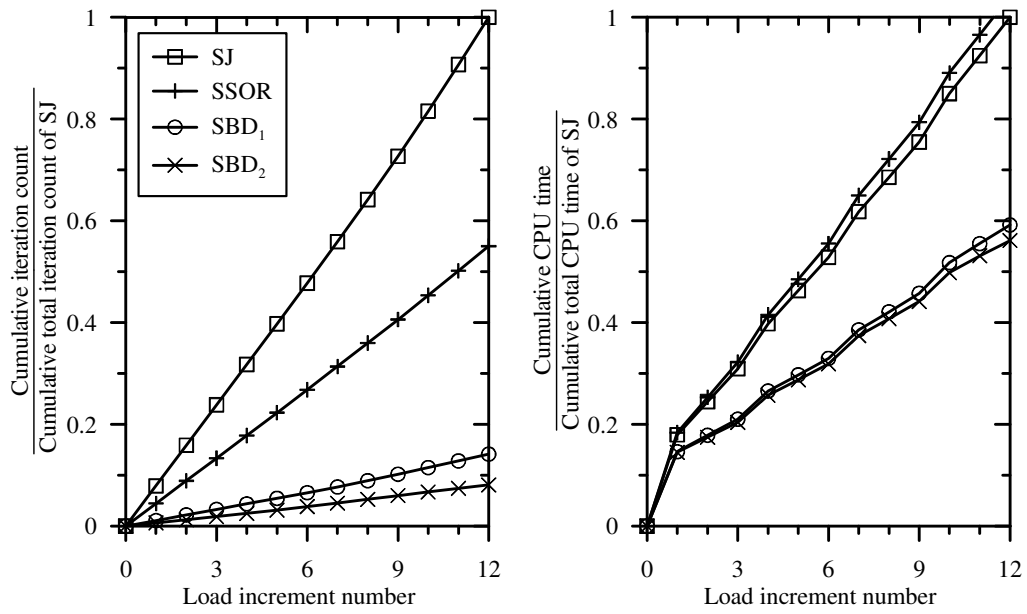


Figure 6.7. Iteration count and CPU time of different preconditioners.

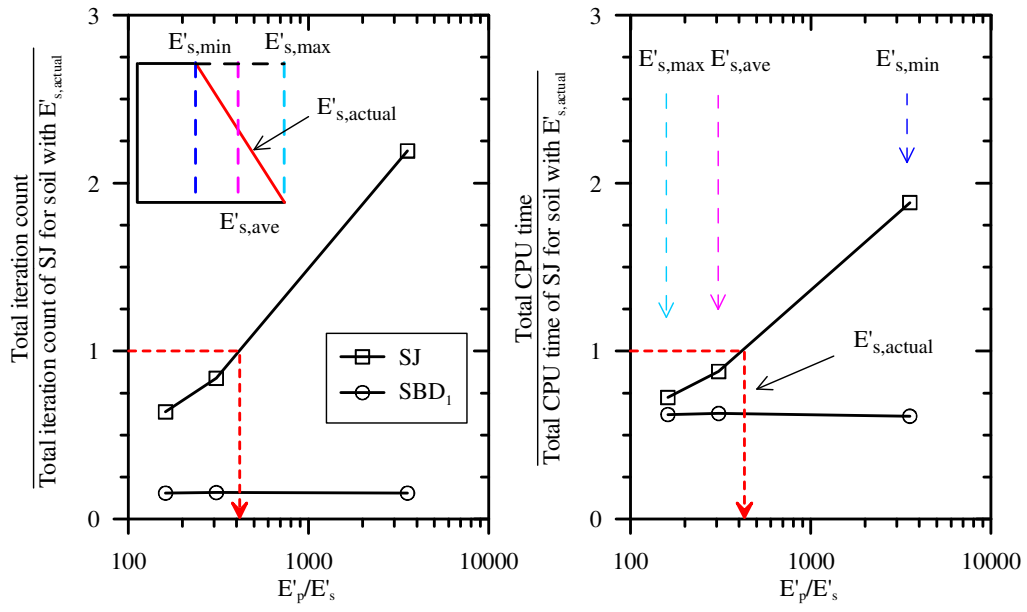


Figure 6.8. Effect of soil profile on different preconditioners.

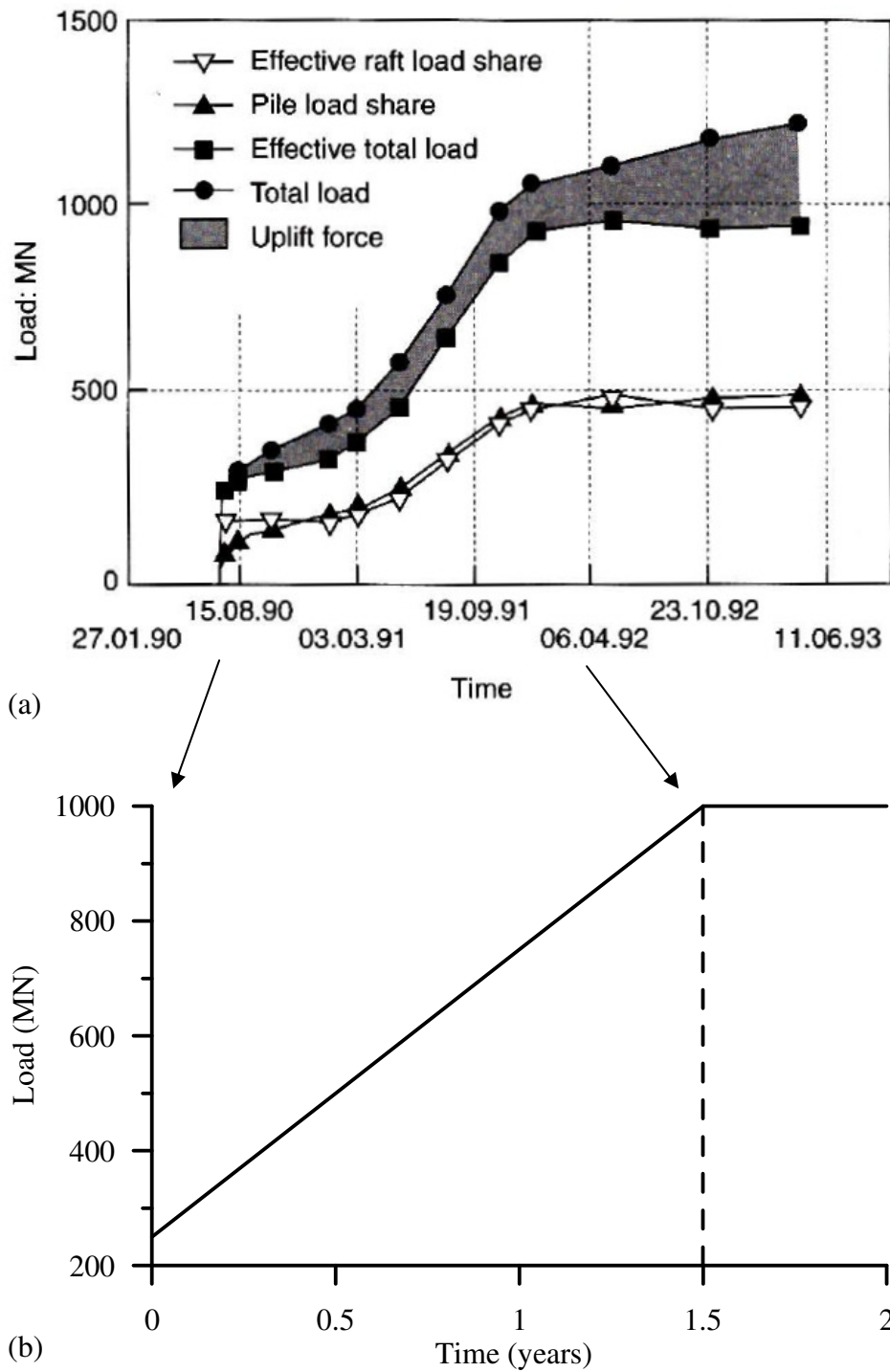


Figure 6.9. Measured time-dependent raft-pile load share for Westendstrasse 1 building, Frankfurt (after Franke *et al.*, 2000); and (b) Idealized load applied on piled-raft for consolidation analysis.

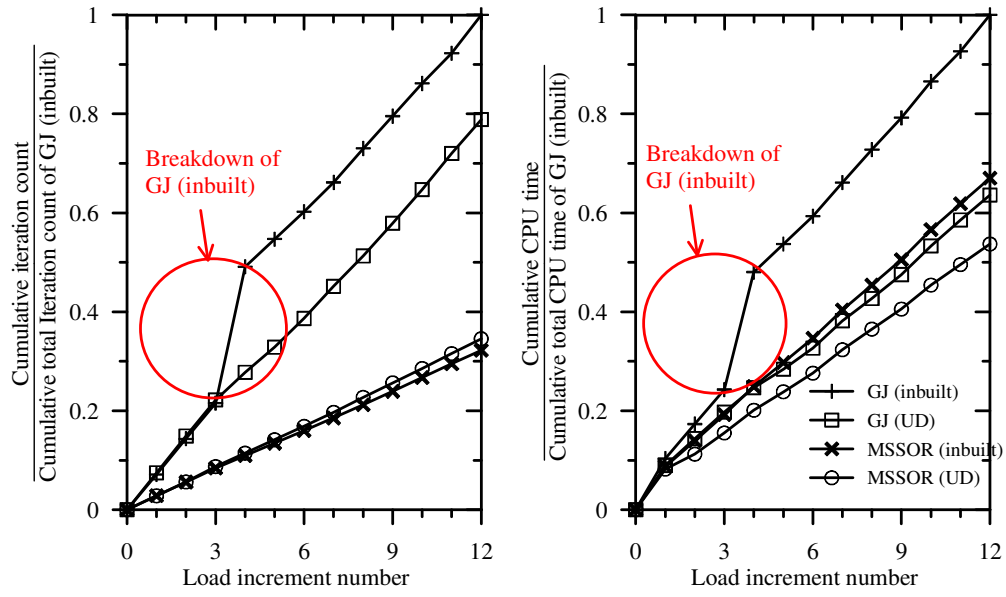


Figure 6.10. Comparison of inbuilt and user defined preconditioners with SQMR ( $k_s = 1 \times 10^{-9}$  m/s).

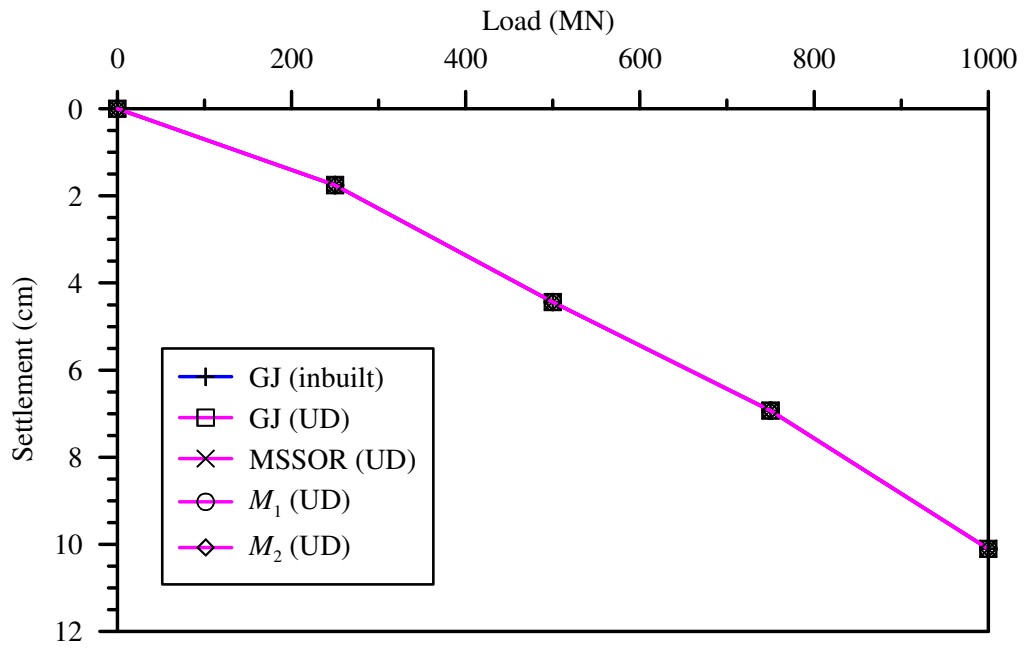


Figure 6.11. Settlements due to different preconditioners with SQMR ( $k_s = 1 \times 10^{-7}$  m/s).

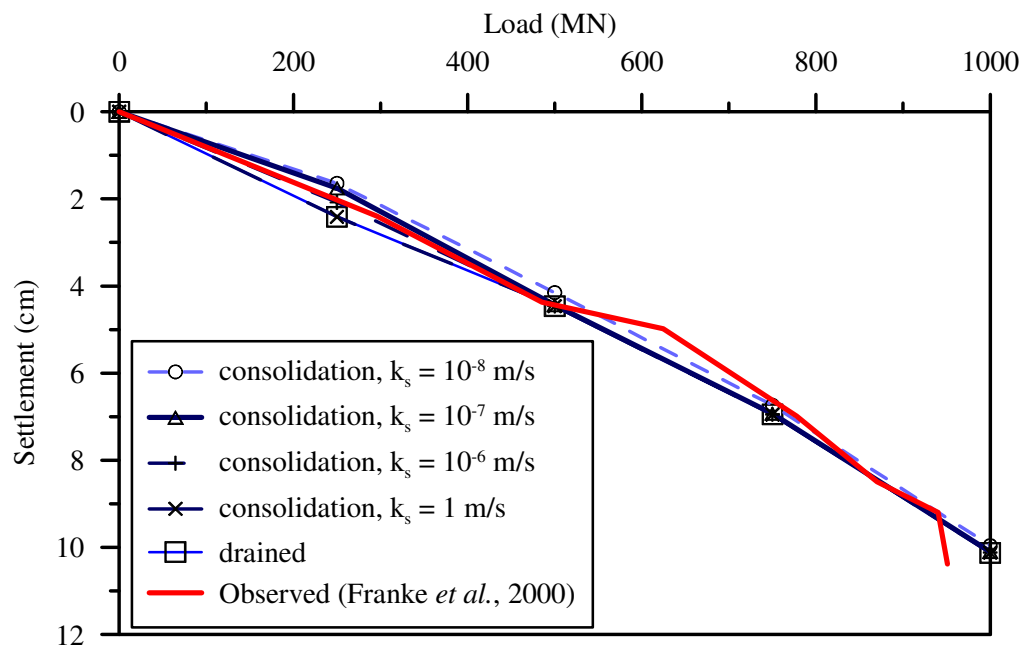


Figure 6.12. Comparison of computed and measured settlements.

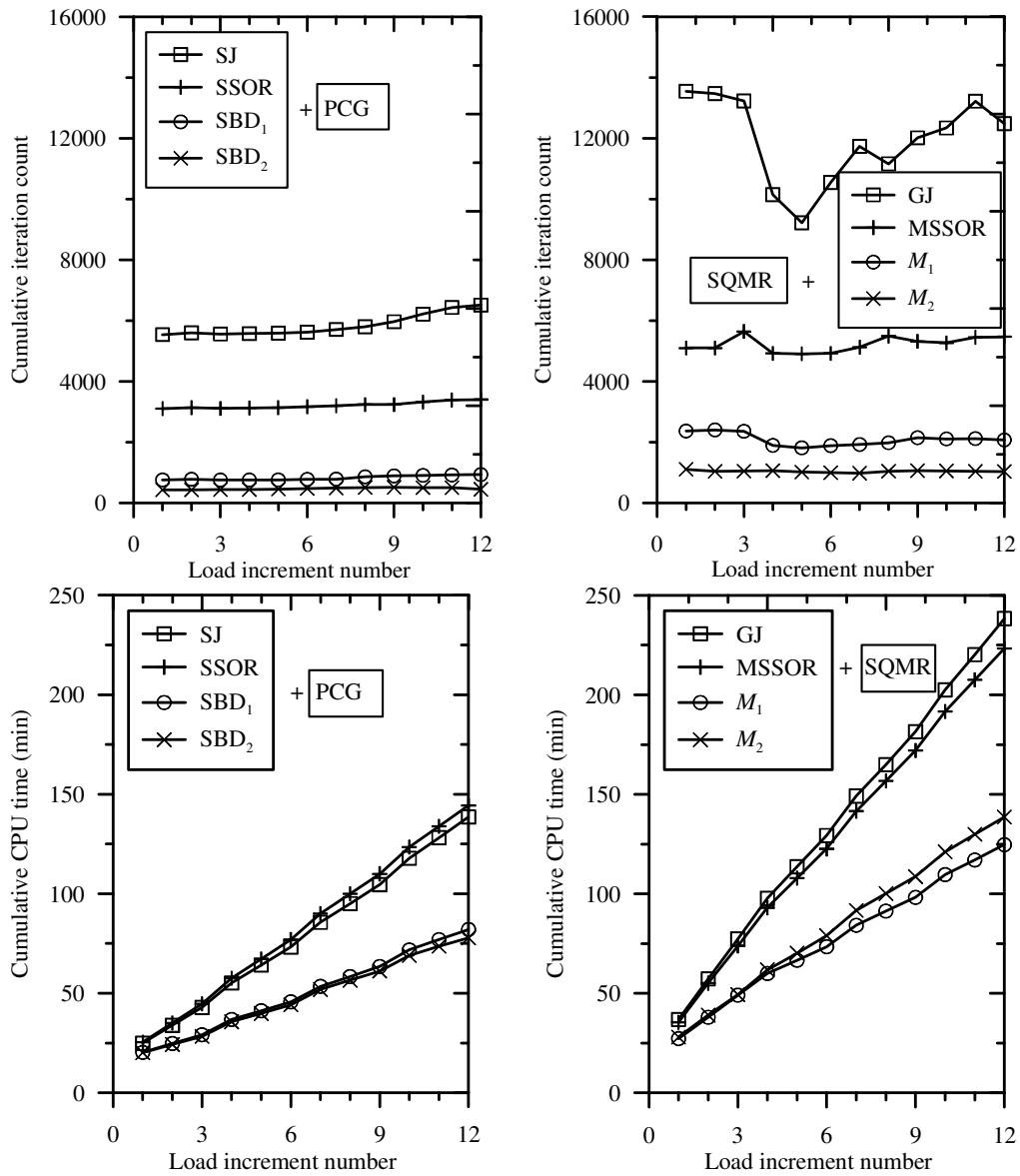


Figure 6.13. Iteration count and CPU time of PCG (for drained analysis) and SQMR (for consolidation analysis) solvers.

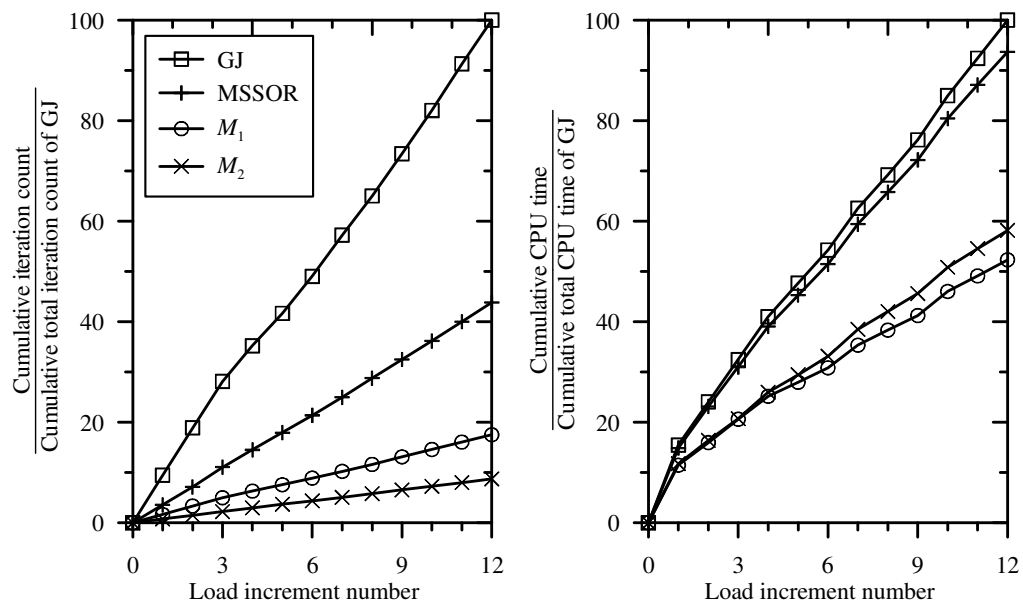
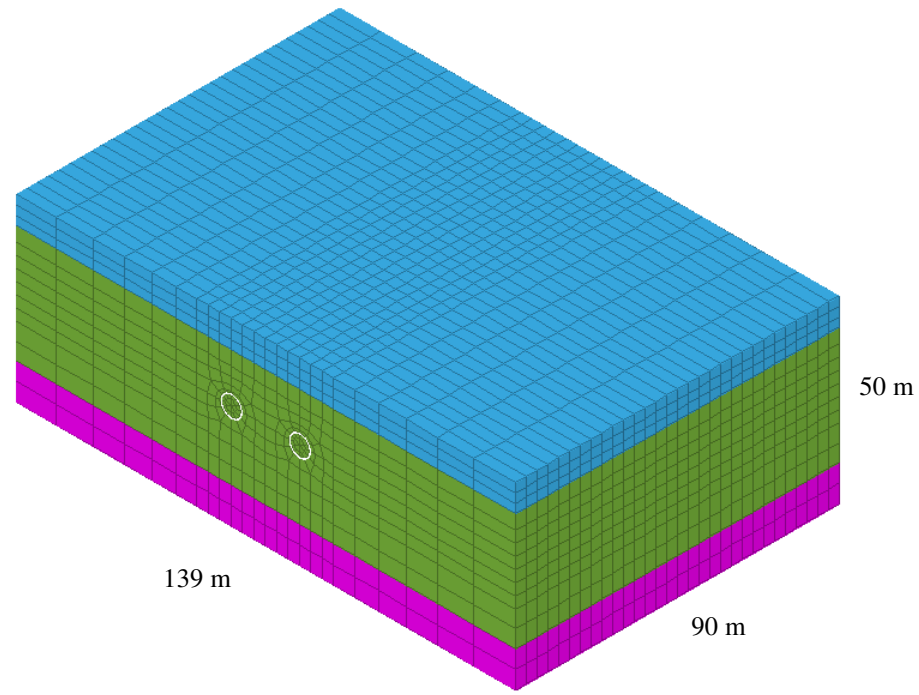
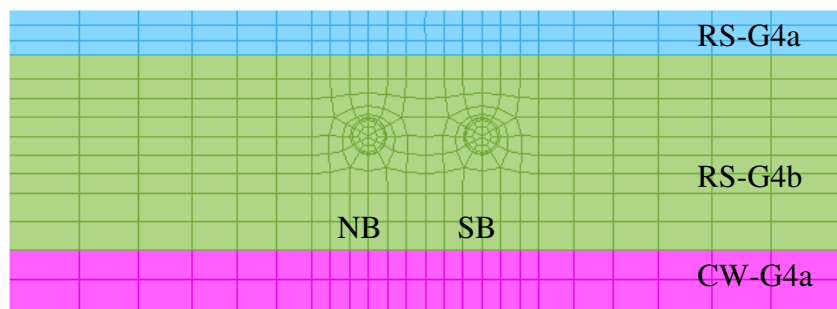


Figure 6.14. Iteration count and CPU time of different preconditioners.



(a)



(b)

Figure 6.15. Finite element mesh for twin tunnels: (a) isometric view; (b) Front view.

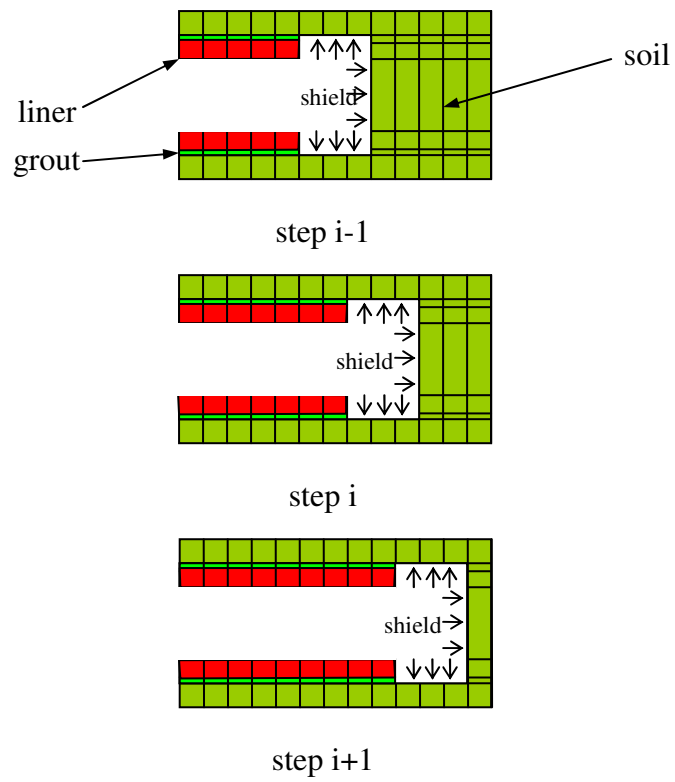


Figure 6.16. Finite element simulation procedure for Shield tunnel advancement.

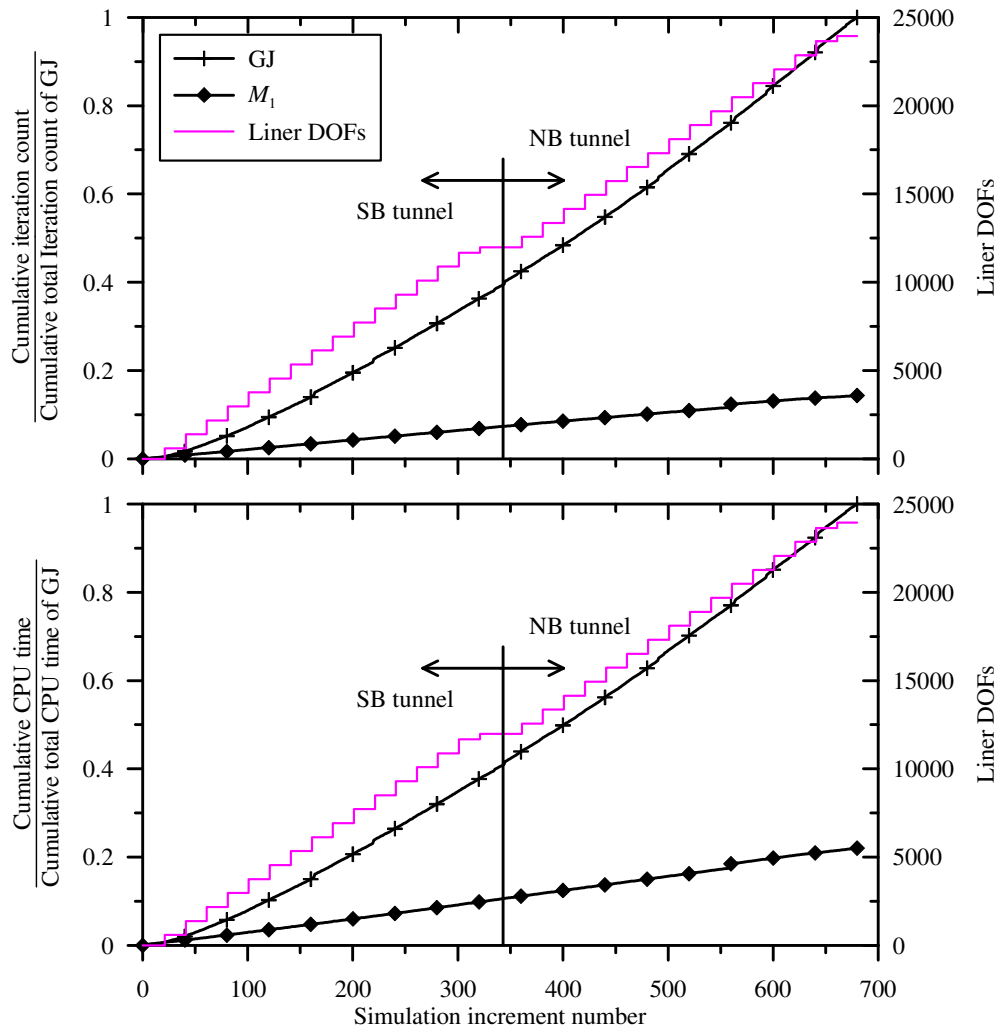


Figure 6.17. Iteration count and CPU time of different preconditioners.

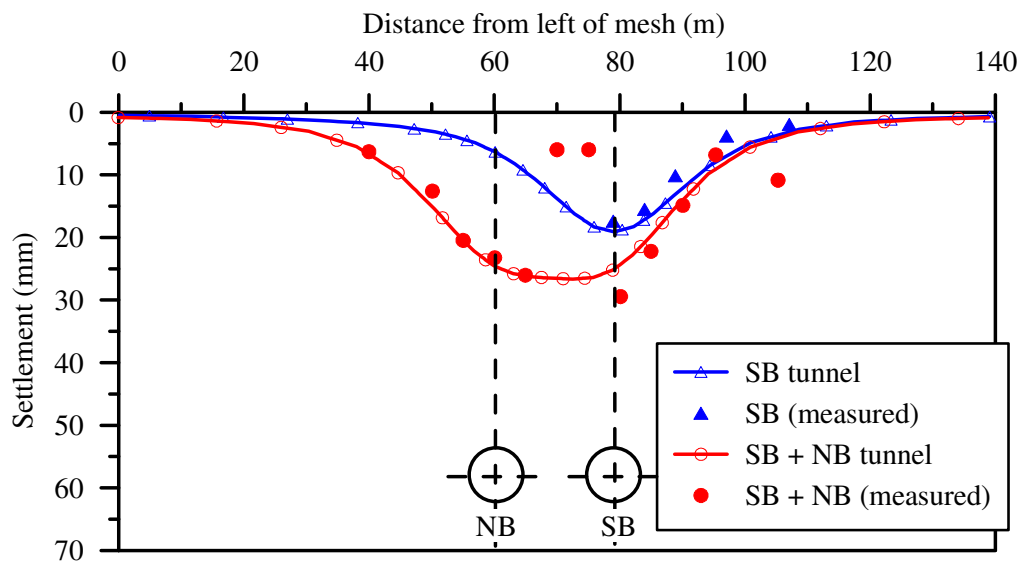


Figure 6.18. Surface settlement trough due to tunnel advancement.

Table 6.1. Properties of Frankfurt clay and piled-raft for FE analysis.

Parameter, symbol, and unit	Soil	Piled-raft
Material model	Mohr-Coulomb	Liner elastic
Effective Young's modulus, $E'$ , MN/m <sup>2</sup>	$7+2.45 z$ ( $z$ is the depth in metres from clay surface)	24,822
Effective Poisson's ratio, $\nu'$	0.3	0.2
Effective cohesion, $c'$ , kN/m <sup>2</sup>	20	-
Effective angle of friction, $\phi'$ , degree	20	-
Coefficient of earth pressure at rest, $K_0$	0.6	-
Bulk unit weight, $\gamma_{\text{bulk}}$ , kN/m <sup>3</sup>	18.5	22

Table 6.2. Typical G4 soil parameters found in C704.

Sub-layer	RS-G4a	RS-G4b	CW-G4a
Depth (m)	0 ~ 7.5	7.5 ~ 40	40 ~ 50
$C_u$ , kN/m <sup>2</sup>	$73 \pm 26$	$80 \pm 29$	$150 \pm 26$
$E'$ , MN/m <sup>2</sup>	$15 + 1.3 z$	$24.75 + 1.2 z$	$63.75 + 1.1 z$
$c'$ , kN/m <sup>2</sup>	19.8	19.6	20
$\phi'$ , degree	19.2	24.2	30.5
$k$ , m/s	$2.16 \times 10^{-7}$	$1.46 \times 10^{-6}$	$1 \times 10^{-8}$
$K_0$	0.86	0.65	0.51

$z$  is the depth in metres measured from top of each soil layer.

Table 6.3. Material properties of liner and grout elements.

Parameter, symbol, and unit	Liner	Grout
Material model	Liner elastic	Liner elastic
Effective Young's modulus, $E'$ , MN/m <sup>2</sup>	28,000	2,800
coefficient of permeability, $k$ , m/s	$1 \times 10^{-12}$	$1 \times 10^{-12}$
Bulk unit weight, $\gamma_{\text{bulk}}$ , kN/m <sup>3</sup>	24	24

*(blank)*

## Chapter 7

---

# CONCLUSIONS AND RECOMMENDATIONS

### 7.1. Summary and conclusions

The linear systems that result from the finite element discretization of problems involving material zones of widely differing stiffness and permeability are usually ill-conditioned. Hence, the efficient solution of large-scale geotechnical problems is a major computational work in the finite element modeling. The existing standard preconditioners [e.g. Standard Jacobi (SJ), symmetric successive over relaxation (SSOR), incomplete LU factorization (ILU) preconditioner] or recently developed so called efficient generalized Jacobi (GJ) and modified SSOR (MSSOR) have been found to be inefficient (require relatively longer runtime) for solving such ill-conditioned problems than for problems involving a homogeneous material. This thesis presented some inexact block diagonal preconditioners to mitigate such ill-conditioning effect (particularly, due to relative differences in the stiffness and permeability of the materials), and hence, resolved a long-standing question. As stated by Barbour and Krahn (2004), “Modeling is more about process than prediction. Modeling of many problems is likely to involve hundreds, if

not thousands, of simulations”, such mitigation by proposed preconditioners can lead to a significant saving in the computational time. For example, dynamic analysis may involve thousands of simulation steps. Hence, the reduction in solution time by the proposed preconditioners for soil-structure interaction problems involving large stiffness ratios will greatly enhance the feasibility of 3D simulation to be used more routinely in actual practice.

The conclusions drawn from the preceding chapters and discussions can be summarized as follows:

1. Different solution approaches have been suggested for the FE solution of Biot’s consolidation analysis of problems. The finite element integration of the Biot’s equations can be written in the form of symmetric indefinite, unsymmetric indefinite and unsymmetric positive definite matrices requiring different iterative solvers. However, symmetric indefinite linear system is preferred over others.
2. Preconditioners ranging from a simple diagonal approximation to more complex block constrained preconditioners have been proposed, with most developments in the last decade.
3. The comparison of MSSOR preconditioner with ILU0 in Chapter 3 revealed that the ILU0 preconditioner should be used with more cautions for Biot’s system. However, MSSOR did not show any breakdown or failure in convergence. On the other hand, the ILU0 may be preferred over MSSOR if (a) instability problem of ILU can be resolved effectively; and (b) RAM constraint is not an issue.

4. In Chapter 4, two simplified block diagonal (SBD) preconditioners were proposed for the practical large-scale soil-structure interaction problems involving significant contrasts in stiffness of the materials. These preconditioners were demonstrated to be insensitive not only to the soil-structure stiffness ratios but also to the number of stiff structural elements present in the system. Hence, SBD preconditioners offered a considerable gain in the solution time relative to the standard SJ, SSOR, and ILU preconditioners.
5. The study in Chapter 4 was extended to Biot's consolidation analysis of the problem in Chapter 5 because these equations can be generalized to produce all geotechnical conditions (drained, undrained, or consolidation condition). Fundamentally, the coefficient matrix in the consolidation analysis is different (indefinite) than a drained analysis (positive definite, Chapter 4), requiring an entirely different iterative solver and preconditioning strategy. Some cost effective block diagonal preconditioners were proposed to effectively mitigate the coupled ill-conditioning of the system due to large relative differences in stiffness and permeability of the materials. The GJ, MSSOR, or ILU preconditioners were found to be affected not only by the stiffness contrasts but also by the number of stiff structural DOFs. The proposed preconditioners were insensitive in both of these respects, which led them to be significantly more efficient (in terms of CPU time) than the other preconditioners when the soil-structure stiffness ratio was large (say 1000 or above).

6. Some charts were presented in Chapters 4 and 5 on when to use the proposed block diagonal preconditioners. These charts may help engineers to select a suitable preconditioner for the problem at hand and an estimated saving in runtime relative to other preconditioners.
7. Finally, the general applicability of the proposed preconditioners was demonstrated by considering two real-world soil-structure interactions problems with nonhomogeneous soil profiles in Chapter 6. It was found that the nonhomogeneous soil profiles are not a problem for the proposed preconditioners. Their convergence behaviors were comparable to those presented for homogeneous problems in preceding Chapters.

In short, the ill-conditioning problem associated with significant contrast in material stiffnesses has been solved. Although the examples demonstrated in this study are limited to piled-raft and tunneling problems, the principles are more generally applicable to all soil-structure interaction problems where significant stiff structural elements are present.

## **7.2. Limitations and Recommendations**

As is said commonly, a PhD thesis is not the end of research, this thesis is no exception. Although this study has provided a breakthrough in the numerical simulation of large-scale problems with significant heterogeneity in material properties, it still has many rooms for further works. In the following, the limitations of this study are listed and areas for further study are subsequently recommended:

1. This study focused particularly on the solution of symmetric large linear systems that arise from drained and time dependent consolidation analysis of static problems under monotonic loading/unloading condition. Dynamic loading condition was not considered. However, there should not be much difficulty in applying the same approach for dynamic problems.
2. The study is limited to symmetric linear system only. Hence, the material behavior that obeys the associated flow rule was taken into consideration. The study can be extended for nonlinear elasto-plastic modeling of soils with non-associated plastic flow rule. For this, however, a nonsymmetric solver needs to be used (Figure 2.1). The central theme of the thesis is that, regardless of chosen soil model, an appropriate preconditioning for the iterative solution of the resulting large system of equations from 3D finite element simulation is at least as important as the characterization of the ground.
3. The results of this thesis are essentially based on numerical experiments in a serial computer. Preconditioners based on element-by-element method and corresponding implementation in parallel computing have not been covered in this study. It would be worth studying the performance of the proposed preconditioning strategies in a parallel computing environment.

*(blank)*

## REFERENCES

---

- Abbo A.J. Finite element algorithms for elastoplasticity and consolidation. PhD Thesis, University of Newcastle, Newcastle. 1997.
- Addenbrooke T.I., and Potts D.M. Twin tunnel interaction: surface and subsurface effects. *International Journal of Geomechanics*, 1(2): pp. 249-271. 2001.
- Addenbrooke T.I., Potts D.M., and Puzrin A.M. The influence of pre-failure soil stiffness on the numerical analysis of tunnel construction. *Geotechnique*, 47(3): pp. 693-712. 1997.
- Agus S.S., Leong E.C., and Rahardjo H. Estimating permeability functions of Singapore residual soils. *Engineering Geology*, 78(1-2): pp. 119-133. 2005.
- Arioli M. A stopping criterion for the conjugate gradient algorithm in a finite element method framework. *Numerische Mathematik*, 97(1): pp. 1-24. 2004.
- Augarde C.E., Ramage A., and Staudacher J. An element-based displacement preconditioner for linear elasticity problems. *Computers and Structures*, 84(31-32): pp. 2306-2315. 2006.
- Augarde C.E., Ramage A., and Staudacher J. Element-based preconditioners for elasto-plastic problems in geotechnical engineering. *International Journal for Numerical Methods in Engineering*, 71(7): pp. 757-779. 2007.
- Augarde C.E., Crouch R.S., Li T., and Ramage A. The effects of geotechnical material properties on the convergence of iterative solvers. In: *Proc. 12th International Conference of International Association for Computer Methods and Advances in Geomechanics (IACMAG)*, Goa, India, 1-6 October. pp. 587-594. 2008.
- Axelsson O. *Iterative solution methods*. Cambridge University Press, Cambridge England ; New York. 1994.
- Balasubramaniam A.S., Loganathan N., Fernando G.S.K., Indraratna B., Phien-wej N., Bergado D.T., and Hanjo Y. Advanced geotechnical analysis : finite element analysis coupled with critical state theories using CRISP computer programme. Asian Institute of Technology, Bangkok. 1992.
- Barbour S.L., and Krahn J. Numerical modelling - Prediction or process? *Geotechnical News*, 22(4): pp. 44-52. 2004.

- Barrett R., Berry M., Chan T.F., Demmel J., Donato J.M., Dongarra J., Eijkhout V., Pozo R., Romine C., and Van der Vorst H. Templates for the solution of linear systems: Building blocks for iterative methods. SIAM press, Philadelphia PA. 1994.
- Benzi M., Meyer C.D., and Tuma M. A sparse approximate inverse preconditioner for the conjugate gradient method. SIAM Journal on Scientific Computing, 17(5): pp. 1135-1149. 1996.
- Benzi M., Cullum J.K., and Tuma M. Robust approximate inverse preconditioning for the conjugate gradient method. SIAM Journal of Scientific Computing, 22(4): pp. 1318-1332. 2001.
- Bergamaschi L., Ferronato M., and Gambolati G. Novel preconditioners for the iterative solution to FE-discretized coupled consolidation equations. Computer Methods in Applied Mechanics and Engineering, 196(25-28): pp. 2647-2656. 2007.
- Bergamaschi L., Ferronato M., and Gambolati G. Mixed constraint preconditioners for the iterative solution of FE coupled consolidation equations. Journal of Computational Physics, 227(23): pp. 9885-9897. 2008.
- Biot M.A. General theory of three-dimensional consolidation. Journal of applied physics, 12(2): pp. 155. 1941.
- Breth H., and Amann P. Time-settlement and settlement distribution with depth in Frankfurt Clay. In: Settlement of Structures, British Geotechnical Society, Pentech Press, London. pp. 141-154. 1975.
- Brinkgreve R.B.J., and Broere W. Plaxis 3D foundation manual, A.A. Balkema, Rotterdam, The Netherlands. 2006.
- Britto A.M., and Gunn M.J. Critical state soil mechanics via finite elements. Ellis Horwood Ltd, Chichester, West Sussex. 1987.
- Burd H.J., Houlsby G.T., Chow L., Augarde C.E., and Liu G. Analysis of settlement damage to masonry structures. In: Proceedings of 3rd European Conference on Numerical Methods in Geotechnical Engineering - ECONMIG 94 Manchester, United kingdom, 7-9 September. pp. 203-208. 1994.
- Burkardt J. Reverse Cuthill McKee Ordering, <[http://people.scs.fsu.edu/~burkardt/f\\_src/rcm/rcm.html](http://people.scs.fsu.edu/~burkardt/f_src/rcm/rcm.html)>. (Access date: August, 2007). 2003.
- Butterfield R., and Banerjee P.K. The elastic analysis of compressible piles and pile groups. Geotechnique, 21(1): pp. 43-60. 1971.
- Chan S.H. Iterative solution for large indefinite linear systems from Biot's finite element formulation. PhD Thesis, National University of Singapore, Singapore. 2002.

- Chan S.H., Phoon K.K., and Lee F.H. A modified Jacobi preconditioner for solving ill-conditioned Biot's consolidation equations using symmetric quasi-minimal residual method. *International Journal for Numerical and Analytical Methods in Geomechanics*, 25(10): pp. 1001-1025. 2001.
- Chan T.F., Gallopoulos E., Simoncini V., Szeto T., and Tong C.H. A quasi-minimal residual variant of the Bi-CGSTAB algorithm for nonsymmetric systems. *SIAM Journal on Scientific Computing*, 15(2): pp. 338-347. 1994.
- Chen X. Preconditioners for iterative solutions of large-scale linear systems arising from Biot's consolidation equations. PhD Thesis, National University of Singapore, Singapore. 2005.
- Chen X., and Phoon K.K. Some numerical experiences on convergence criteria for iterative finite element solvers. *Computers and Geotechnics*, 36(8): pp. 1272-1284. 2009.
- Chen X., Toh K.C., and Phoon K.K. A modified SSOR preconditioner for sparse symmetric indefinite linear systems of equations. *International Journal for Numerical Methods in Engineering*, 65(6): pp. 785-807. 2006.
- Chen X., Phoon K.K., and Toh K.C. Partitioned versus global Krylov subspace iterative methods for FE solution of 3-D Biot's problem. *Computer Methods in Applied Mechanics and Engineering*, 196(25-28): pp. 2737-2750. 2007.
- Chow E., and Saad Y. Experimental study of ILU preconditioners for indefinite matrices. *Journal of Computational and Applied Mathematics*, 86(2): pp. 387-414. 1997.
- Chow E., and Heroux M.A. An object-oriented framework for block preconditioning. *ACM Transactions on Mathematical Software*, 24(2): pp. 159-183. 1998.
- Clancy P., and Randolph M.F. An approximate analysis procedure for piled raft foundations. *International Journal for Numerical & Analytical Methods in Geomechanics*, 17(12): pp. 849-869. 1993.
- Dasari G.R., Rawlings C.G., and Bolton M.D. Numerical modelling of a NATM tunnel construction in London clay. In: the 1st International Symposium on the Geotechnical Aspects of Underground Construction in Soft Ground, A.A. Balkema, Rotterdam, The Netherlands, London. pp. 491-496. 1996.
- Eisenstat S.C. Efficient implementation of A class of preconditioned conjugate gradient methods. *SIAM Journal on Scientific and Statistical Computing*, 2: pp. 1-4. 1981.

- El-Mossallamy Y., and Franke E. Piled rafts: numerical modelling to simulate the behaviour of piled raft foundations. Published by the Authors, Darmstadt, Germany. 1997.
- Fellenius B.H. Negative skin friction and settlement of piles. In: the 2nd International Seminar on Pile Foundations, Singapore, 28-30 November. 1984.
- Fellenius B.H. Unified design of piled foundations with emphasis on settlement analysis. In: Current Practice and Future Trends in Deep Foundations, J. A. DiMaggio and M. H. Hussein (Eds.), ASCE Geotechnical Special Publication 125. pp. 253–275. 2004.
- Ferronato M., Gambolati G., and Teatini P. Ill-conditioning of finite element poroelasticity equations. *International Journal of Solids and Structures*, 38(34-35): pp. 5995-6014. 2001.
- Ferronato M., Janna C., and Gambolati G. Mixed constraint preconditioning in computational contact mechanics. *Computer Methods in Applied Mechanics and Engineering*, 197(45-48): pp. 3922-3931. 2008.
- Ferronato M., Pini G., and Gambolati G. The role of preconditioning in the solution to FE coupled consolidation equations by Krylov subspace methods. *International Journal for Numerical and Analytical Methods in Geomechanics*, 33(3): pp. 405-423. 2009.
- Ferronato M., Bergamaschi L., and Gambolati G. Performance and robustness of block constraint preconditioners in finite element coupled consolidation problems. *International Journal for Numerical Methods in Engineering*, 81(3): pp. 381-402. 2010.
- Ferronato M., Pini G., Gambolati G., and Janna C. Symmetric and unsymmetric block preconditioning for the iterative solution to FE coupled consolidation. In: International Conference on Numerical Analysis and Applied Mathematics, American Institute of Physics, Melville, NY, USA, 16-20 September. pp. 216-219. 2007.
- Franke E., Lutz B., and El-Mossallamy Y. Measurements and numerical modelling of high rise building foundations on Frankfurt Clay. Vertical and Horizontal Deformations of Foundations and Embankments, ASCE Geotechnical Special Publication, 40(2): pp. 1325-1336. 1994.
- Franke E., El-Mossallamy Y., and Wittmann P. Calculation methods for raft foundations in Germany. In: Design applications of raft foundations, J. A. Hemsley (Ed.), Thomas Telford, London. pp. 283-322. 2000.
- Freund R.W. Preconditioning of symmetric, but highly indefinite linear systems. In: Proc. 15th IMACS World Congress on Scientific Computation Modelling and Applied Mathematics, Berlin, Germany, 25-29 August. pp. 551-556. 1997.

- Freund R.W., and Nachtigal N.M. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60(1): pp. 315-339. 1991.
- Freund R.W., and Nachtigal N.M. An implementation of the QMR method based on coupled two-term recurrences. *SIAM Journal on Scientific Computing*, 15(2): pp. 313-337. 1994a.
- Freund R.W., and Nachtigal N.M. A new Krylov-subspace method for symmetric indefinite linear systems. In: *Proc. 14th MIACS World Congress on Computational and Applied Mathematics*, Atlanta, USA. pp. 1253-1256. 1994b.
- Freund R.W., and Nachtigal N.M. Software for simplified Lanczos and QMR algorithms. *Applied Numerical Mathematics*, 19(3): pp. 319-341. 1995.
- Galli G., Grimaldi A., and Leonardi A. Three-dimensional modelling of tunnel excavation and lining. *Computers and Geotechnics*, 31(3): pp. 171-183. 2004.
- Gambolati G., Pini G., and Ferronato M. Numerical performance of projection methods in finite element consolidation models. *International Journal for Numerical and Analytical Methods in Geomechanics*, 25(14): pp. 1429-1447. 2001.
- Gambolati G., Pini G., and Ferronato M. Direct, partitioned and projected solution to finite element consolidation models. *International Journal for Numerical and Analytical Methods in Geomechanics*, 26(14): pp. 1371-1383. 2002.
- Gambolati G., Pini G., and Ferronato M. Scaling improves stability of preconditioned CG-like solvers for FE consolidation equations. *International Journal for Numerical and Analytical Methods in Geomechanics*, 27(12): pp. 1043-1056. 2003.
- Gambolati G., Ferronato M., and Janna C. Preconditioners in computational geomechanics: A survey. *International Journal for Numerical and Analytical Methods in Geomechanics*. DOI: 10.1002/nag.937. 2010.
- GeoFEA. Software, Version 8.0. GeoSoft Pte. Ltd. 2006.
- George A., and Lui J. Computer solution of large sparse positive definite systems. Prentice-Hall, Englewood Cliffs, NJ. 1981.
- Golub G.H., and Van Loan C.F. *Matrix computations* (2nd ed.). Johns Hopkins University Press, Baltimore, USA. 1989.
- Gonilho Pereira C., Castro-Gomes J., and Pereira de Oliveira L. Influence of natural coarse aggregate size, mineralogy and water content on the permeability of structural concrete. *Construction and Building Materials*, 23(2): pp. 602-608. 2009.

- Greenbaum A. Iterative methods for solving linear systems. Society for Industrial and Applied Mathematics, Philadelphia, PA. 1997.
- Grote M.J., and Huckle T. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3): pp. 838-853. 1997.
- Hage Chehade F., and Shahrour I. Numerical analysis of the interaction between twin-tunnels: Influence of the relative position and construction procedure. *Tunnelling and Underground Space Technology*, 23(2): pp. 210-214. 2008.
- Han J., and Ye S.L. A theoretical solution for consolidation rates of stone column-reinforced foundations accounting for smear and well resistance effects. *International Journal of Geomechanics*, 2(2): pp. 135-151. 2002.
- Harris D.I., Menkiti C.O., Pooley A.J., and Stephenson J.A. Construction of low-level tunnels below Waterloo Station with compensation grouting for the Jubilee Line Extension. In: *Aspects of Underground Construction in Soft Ground*, R. J. Mair and R. N. Taylor (Eds.), Balkema, Rotterdam. pp. 361-366. 1996.
- Hestenes M.R., and Stiefel E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6): pp. 409-436. 1952.
- Horn R.A., and Johnson C.R. *Matrix analysis*. Cambridge University Press, Cambridge, UK. 1985.
- Huckle T. Approximate sparsity patterns for the inverse of a matrix and preconditioning. *Applied Numerical Mathematics*, 30(2): pp. 291-304. 1999.
- Hughes T., Levit I., and Winget J. An element-by-element solution algorithm for problems of structural and solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 36(2): pp. 241-254. 1983.
- Intel® Fortran Compiler User and Reference Guides(Document Number: 304970-006US).
- Irons B.M. A frontal solution program for finite element analysis. *International Journal for Numerical Methods in Engineering*, 2(1): pp. 5-32. 1970.
- Janna C., Comerlati A., and Gambolati G. A comparison of projective and direct solvers for finite elements in elastostatics. *Advances in Engineering Software*, 40(8): pp. 675-685. 2009.
- Johnson O.G., Micchelli C.A., and Paul G. Polynomial preconditioners for conjugate gradient calculations. *SIAM Journal on Numerical Analysis*, 20(2): pp. 362-376. 1983.

- Katzenbach R., and Breth H. Nonlinear 3D analysis for NATM in Frankfurt clay. In: Proc. 10th International Conference on Soil Mechanics and Foundation Engineering, Rotterdam, Balkema. pp. 315–318. 1981.
- Katzenbach R., Arslan U., and Moormann C. Piled raft foundation projects in Germany. In: Design Applications of Raft Foundations, J. A. Hemsley (Ed.), Thomas Telford, London. pp. 323-391. 2000.
- Kayupov M.A., Bulgakov V.E., and Kuhn G. Efficient solution of 3-D geomechanical problems by indirect bem using iterative methods. International Journal for Numerical and Analytical Methods in Geomechanics, 22(12): pp. 983-1000. 1998.
- Keller C., Gould N.I.M., and Wathen A.J. Constraint preconditioning for indefinite linear systems. SIAM Journal on Matrix Analysis and Applications, 21(4): pp. 1300-1317. 2000.
- Kelley C.T. Iterative methods for linear and nonlinear equations. Society for Industrial and Applied Mathematics, Philadelphia. 1995.
- Kershaw D.S. The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. Journal of Computational Physics, 26: pp. 43-65. 1978.
- Koenker R., and Ng P. SparseM: A sparse matrix package for R \*, <<http://cran.r-project.org/web/packages/SparseM/index.html>>. (Access date: August, 2007). 2007.
- Kuhns G.L. Downdrag in pile design: The positive aspects of negative skin friction. In: From Research to Practice in Geotechnical Engineering, ASCE, New Orleans, LA, USA, 9-12 March. pp. 489-506. 2008.
- Kuwabara F. An elastic analysis for piled raft foundations in a homogeneous soil. Soils and foundations, 29(1): pp. 82-92. 1989.
- Lanczos C. Solution of systems of linear equations by minimized iterations. Journal of Research of the National Bureau of Standards, 49(1): pp. 33-53. 1952.
- Lee F.H., Phoon K.K., and Lim K.C. Large scale three-dimensional finite element analysis of underground construction. In: Proc. International Conference on Numerical Simulation of Construction Processes in Geotechnical Engineering for Urban Environment, Bochum, Germany, 23-24 March. pp. 141-153. 2006.
- Lee F.H., Phoon K.K., Lim K.C., and Chan S.H. Performance of Jacobi preconditioning in Krylov subspace solution of finite element equations. International Journal for Numerical and Analytical Methods in Geomechanics, 26(4): pp. 341-372. 2002.
- Lee K.M., and Rowe R.K. Finite element modelling of the three-dimensional ground deformations due to tunnelling in soft cohesive soils: Part I -

- Method of analysis. *Computers and Geotechnics*, 10(2): pp. 87-109. 1990.
- Lewis R.W., and Schrefler B.A. The finite element method in the static and dynamic deformation and consolidation of porous media (2nd ed.). John Wiley, New York. 1998.
- Lim K.C. Three-dimensional finite element analysis of earth pressure balance tunnelling. PhD Thesis, National University of Singapore, Singapore. 2003.
- Maharaj D.K. Three dimensional nonlinear finite element analysis to study the effect of raft and pile stiffness on the load-settlement behaviour of piled raft foundations. *Electronic Journal of Geotechnical Engineering*, 9A. 2004.
- Maleki Javan M.R., Noorzad A., and Latifi Namin M. Three dimensional nonlinear finite element analysis of pile groups in saturated porous media using a new transmitting boundary. *International Journal for Numerical and Analytical Methods in Geomechanics*, 32(6): pp. 681-699. 2008.
- Manteuffel T.A. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34(150): pp. 473-497. 1980.
- Marcus M., and Minc H. A survey of matrix theory and matrix inequalities. Dover Publications, New York. 1992.
- Meijerink J.A., and van der Vorst H.A. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 31: pp. 148-162. 1977.
- Meurant G.A. Computer solution of large linear systems. North-Holland: Elsevier, Amsterdam; New York. 1999.
- Migliazza M., Chiorboli M., and Giani G.P. Comparison of analytical method, 3D finite element model with experimental subsidence measurements resulting from the extension of the Milan underground. *Computers and Geotechnics*, 36(1-2): pp. 113-124. 2009.
- Möller S.C. Tunnel induced settlements and structural forces in linings. PhD, Institut für Geotechnik der Universität Stuttgart, Stuttgart. 2006.
- Möller S.C., and Vermeer P.A. On numerical simulation of tunnel installation. *Tunnelling and Underground Space Technology*, 23(4): pp. 461-475. 2008.
- Mroueh H., and Shahrour I. Use of sparse iterative methods for the resolution of three-dimensional soil/structure interaction problems. *International Journal for Numerical and Analytical Methods in Geomechanics*, 23(15): pp. 1961-1975. 1999.

- Mroueh H., and Shahrour I. A full 3-D finite element analysis of tunneling-adjacent structures interaction. *Computers and Geotechnics*, 30(3): pp. 245-253. 2003.
- Mroueh H., and Shahrour I. A simplified 3D model for tunnel construction using tunnel boring machines. *Tunnelling and Underground Space Technology*, 23(1): pp. 38-45. 2008.
- Murphy M.F., Golub G.H., and Wathen A.J. A note on preconditioning for indefinite linear systems. *SIAM Journal of Scientific Computing*, 21(6): pp. 1969-1972. 2000.
- Nayak G.C., and Zienkiewicz O.C. Elasto-plastic stress analysis. A generalization for various constitutive relations including strain softening. *International Journal for Numerical Methods in Engineering*, 5(1): pp. 113-135. 1972.
- Ng C.W.W., Lee K.M., and Tang D.K.W. Three-dimensional numerical investigations of new Austrian tunnelling method (NATM) twin tunnel interactions. *Canadian Geotechnical Journal*, 41(3): pp. 523-539. 2004.
- Ng E.G., and Peyton B.W. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM Journal of Scientific Computing*, 14: pp. 1034-1056. 1993.
- Nour-Omid B., and Parlett B.N. Element preconditioning using splitting techniques. *SIAM Journal on Scientific and Statistical Computing*, 6(3): pp. 761-770. 1985.
- Novak L.J., Reese L.C., and Wang S.T. Analysis of pile-raft foundations with 3D finite element method. In: *Proc. 2005 Structures and Congress and 2005 Forensic Engineering Symposium*, ASCE, New York, New York. 2005.
- Nyhoff L.R., and Leestma S.C. *FORTTRAN 90 for engineers and scientists*. Prentice Hall, Upper Saddle River, N.J. 1997.
- Ottaviani M. Three-dimensional finite element analysis of vertically loaded pile groups. *Geotechnique*, 25(2): pp. 159-174. 1975.
- Paige C.C., and Saunders M.A. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4): pp. 617-629. 1975.
- Pan X.D., and Hudson J.A. Plane strain analysis in modelling three-dimensional tunnel excavations. *International Journal of Rock Mechanics and Mining Sciences*, 25(5): pp. 331-337. 1988.
- Pang C.H. The effects of tunnel construction on nearby pile foundation. PhD Thesis, National University of Singapore, Singapore. 2006.

- Papadrakakis M. Solving large-scale linear problems in solid and structural mechanics. In: *Solving Large-scale Problems in Mechanics: The Development and Application of Computational Solution Methods*, M. Papadrakakis (Ed.), Wiley, Chichester. pp. 1–32. 1993a.
- Papadrakakis M. (Ed.). *Solving large-scale problems in mechanics: The development and application of computational solution methods*. Wiley, Chichester. 1993b.
- Payer H.J., and Mang H.A. Iterative strategies for solving systems of linear, algebraic equations arising in 3D BE-FE analyses of tunnel drivings. *Numerical Linear Algebra with Applications*, 4(3): pp. 239-268. 1997.
- Phoon K.K. Iterative solution of large-scale consolidation and constrained finite element equations for 3D problems. In: *International e-Conference on Modern Trends in Foundation Engineering: Geotechnical Challenges and Solutions*, IIT Madras, India, 26-30 January. 2004.
- Phoon K.K., Toh K.C., and Chen X. Block constrained versus generalized Jacobi preconditioners for iterative solution of large-scale Biot's FEM equations. *Computers and Structures*, 82(28): pp. 2401-2411. 2004.
- Phoon K.K., Lee F.H., and Chan S.H. Iterative solution of intersecting tunnels using the generalised Jacobi preconditioner. In: *Proc. International Conference on Numerical Simulation of Construction Processes in Geotechnical Engineering for Urban Environment*, Bochum, Germany, 23-24 March. pp. 155-163. 2006.
- Phoon K.K., Chaudhary K.B., and Toh K.C. Comparison of MSSOR versus ILU(0) preconditioners for Biot's FEM consolidation equations. In: *the 12th IACMAG Conference*, Goa, India, 1-6 October. pp. 185-192. 2008.
- Phoon K.K., Toh K.C., Chan S.H., and Lee F.H. An efficient diagonal preconditioner for finite element solution of Biot's consolidation equations. *International Journal for Numerical Methods in Engineering*, 55(4): pp. 377-400. 2002.
- Phoon K.K., Chan S.H., Toh K.C., and Lee F.H. Fast iterative solution of large undrained soil-structure interaction problems. *International Journal for Numerical and Analytical Methods in Geomechanics*, 27(3): pp. 159-181. 2003.
- PLAXIS 2D. Software, Version 9.02. PLAXIS. 2009.
- Potts D.M., and Zdravković L. *Finite element analysis in geotechnical engineering*. Thomas Telford, London. 1999.
- Potts D.M., and Zdravković L. *Finite element analysis in geotechnical engineering: application*. Thomas Telford, London. 2001.

- Poulos H.G. An approximate numerical analysis of pile-raft interaction. *International Journal for Numerical and Analytical Methods in Geomechanics*, 18(2): pp. 73-92. 1994.
- Poulos H.G. Methods of analysis of piled raft foundations. Report to technical committee TC 18 on piled foundations, International Society on Soil Mechanics and Geotechnical Engineering. 2001a.
- Poulos H.G. Piled raft foundations: design and applications. *Geotechnique*, 51(2): pp. 95-113. 2001b.
- Poulos H.G., and Davis E.H. *Pile foundation analysis and design*. Wiley, New York. 1980.
- Poulos H.G., Small J.C., Ta L.D., Sinha J., and Chen L. Comparison of some methods for analysis of piled rafts. In: *Proc. 14th International Conference on Soil Mechanics and Foundation Engineering*, AA Balkema, Hamburg. pp. 1119-1124. 1997.
- Press W.H., Teukolsky S.A., Vetterling W.T., and Flannery B.P. *Numerical recipes in FORTRAN : The art of scientific computing* (2nd ed.). Cambridge University Press, Cambridge England; New York, NY, USA. 1992.
- Press W.H., Teukolsky S.A., Vetterling W.T., and Flannery B.P. *Numerical recipes in Fortran 90 : The art of parallel scientific computing* (2nd ed.). Cambridge University Press, New York, N.Y. 1996.
- Randolph M.F., and Wroth C.P. Analysis of deformation of vertically loaded piles. *Journal of the Geotechnical Engineering Division*, 104(12): pp. 1465-1488. 1978.
- Randolph M.F., and Wroth C.P. Analysis of the vertical deformation of pile groups. *Geotechnique*, 29(4): pp. 423-439. 1979.
- Reid J.K. On the method of conjugate gradients for the solution of large sparse systems of linear equations. In: *Large Sparse Sets of Linear Equations*, J. K. Reid (Ed.), Academic Press, New York. pp. 231-254. 1971.
- Reul O., and Randolph M.F. Piled rafts in overconsolidated clay: Comparison of in situ measurements and numerical analyses. *Geotechnique*, 53(3): pp. 301-315. 2003.
- Rowe R.K., and Lee K.M. An evaluation of simplified techniques for estimating 3-dimensional undrained ground movements due to tunneling in soft soils. *Canadian Geotechnical Journal*, 29(1): pp. 39-52. 1992.
- Saad Y. SPARSKIT: A basic tool kit for sparse matrix computations, <<http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html>>. (Access date: December, 2006). 1994a.

- Saad Y. ILUT: A dual threshold incomplete LU factorization. *Numerical Linear Algebra with Applications*, 1(4): pp. 387-402. 1994b.
- Saad Y. Iterative methods for sparse linear systems. PWS Publishing Company, Boston. 1996.
- Saad Y., and Schultz M.H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3): pp. 856-869. 1986.
- Saad Y., and Van Der Vorst H. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1-2): pp. 1-33. 2000.
- SAGE-CRISP. Software, Version 4.02b. Sage Consortium. 2000.
- Shen R.F. Negative skin friction on single piles and pile groups. PhD Thesis, National University of Singapore, Singapore. 2008.
- Shewchuk J.R. An introduction to the conjugate gradient method without the agonizing pain, <<http://www-2.cs.cmu.edu/~jrs/jrspapers.html>>. (Access date: December, 2006). 1994.
- Shirlaw J.N., Ong J.C.W., Rosser R.B., Osborne N.H., Tan C.G., and Heslop P.J.E. Immediate settlements due to tunnelling for the North East Line. In: *Proc. Underground Singapore*, Singapore, 29-30 November. pp. 76-90. 2001.
- SIGMA/W. Software, Version 7.15. GEO-SLOPE. 2007.
- Silvester D., and Wathen A.J. Fast iterative solution of stabilised Stokes systems part II: Using general block preconditioners. *SIAM Journal on Numerical Analysis*, 31(5): pp. 1352-1367. 1994.
- Small J.C., and Liu H.L.S. Time-settlement behaviour of piled raft foundations using infinite elements. *Computers and Geotechnics*, 35(2): pp. 187-195. DOI: 10.1016/j.compgeo.2007.04.004. 2008.
- Smith I.M., and Griffiths D.V. *Programming the finite element method* (3rd ed.). Wiley, Chichester, New York. 1997.
- Smith I.M., and Wang A. Analysis of piled rafts. *International Journal for Numerical and Analytical Methods in Geomechanics*, 22(10): pp. 777-790. 1998.
- Smith I.M., and Griffiths D.V. *Programming the finite element method* (4rd ed.). John Wiley & Sons, West Sussex. 2004.
- Smith I.M., Wong S.W., Gladwell I., and Gilvary B. PCG methods in transient FE analysis. Part I. First order problems. *International Journal for Numerical Methods in Engineering*, 28(7): pp. 1557-1566. 1989.

- Sonneveld P. CGS: A fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 10: pp. 36-52. 1989.
- Toh K.C. Solving large scale semidefinite programs via an iterative solver on the augmented systems. *SIAM Journal on Optimization*, 14(3): pp. 670-698. 2004.
- Toh K.C., and Phoon K.K. Comparison between iterative solution of symmetric and non-symmetric forms of Biot's FEM equations using the generalized Jacobi preconditioner. *International Journal for Numerical and Analytical Methods in Geomechanics*, 32(9): pp. 1131-1146. 2008.
- Toh K.C., Phoon K.K., and Chan S.H. Block preconditioners for symmetric indefinite linear systems. *International Journal for Numerical Methods in Engineering*, 60(8): pp. 1361-1381. 2004.
- van der Vorst H.A. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13: pp. 631-644. 1992.
- van der Vorst H.A. *Iterative Krylov methods for large linear systems*. Cambridge University Press. 2003.
- Vermeer P.A., Bonnier P.G., and Möller S.C. On a smart use of 3D-FEM in tunnelling. *Plaxis Bulletin*, 11: pp. 2-7. 2001.
- Wathen A.J., and Silvester D. Fast iterative solution of stabilized stokes systems Part I: Using simple diagonal preconditioners. *SIAM Journal on Numerical Analysis*, 30(3): pp. 630-649. 1993.
- Winget J.M., and Hughes T.J.R. Solution algorithms for nonlinear transient heat-conduction analysis employing element-by-element iterative strategies. *Computer Methods in Applied Mechanics and Engineering*, 52(1-3): pp. 711-815. 1985.
- Wong S.W., Smith I.M., and Gladwell I. PCG methods in transient FE analysis. Part II. Second order problems. *International Journal for Numerical Methods in Engineering*, 28(7): pp. 1567-1576. 1989.
- Yeo C.H., Lee F.H., Tan S.C., Hasegawa O., Suzuki H., and Shinji M. Three dimensional numerical modelling of a NATM tunnel. *International Journal of the JCRM*, 5(1): pp. 33-38. 2009.
- Zienkiewicz O.C. *The finite element method* (5th ed.). Butterworth Heinemann, Oxford. 2000.
- Zienkiewicz O.C., Valliappan S., and King I.P. Elasto-plastic solutions of engineering problems 'initial stress', finite element approach. *International Journal for Numerical Methods in Engineering*, 1: pp. 75-100. 1969.

*(blank)*

# APPENDIX A

---

## LINEAR ALGEBRA

This Section provides the meaning of various mathematical terms used in this thesis. The information provided here is very brief. For more detailed description, the reader is referred to any text book on linear algebra (e.g. Golub and Van Loan, 1989).

### **A.1. Definition of terminologies**

#### **A.1.1. Sparse matrix**

A matrix with a large number of zero entries is called a sparse matrix. A different storage scheme is required to exploit the sparsity in storing the matrix and matrix related operations.

#### **A.1.2. Symmetric positive definite matrix**

A matrix which satisfies  $A = A^T$  is called symmetric and if  $v^T A v > 0$  for a vector  $v$  with  $v \neq 0$  is called symmetric positive definite (SPD) matrix. SPD matrix has a nice property that the Cholesky factorization of  $A$ , in direct method, reduces computations by a factor of two in comparison to alternative methods (Press *et al.*, 1992). In iterative methods, the storage requirement can

be halved as the matrix-vector operation can effectively be performed with a symmetric half matrix only. The eigenvalues of SPD matrix are all positive.

### **A.1.3. Symmetric indefinite matrix**

If the matrix is neither positive definite nor negative definite, it is called symmetric indefinite. Indefinite matrices have both positive and negative eigenvalues.

### **A.1.4. Eigenvalue of a matrix**

Eigenvalues are special values associated with a square matrix, which can be used to analyze the action of pre-multiplying a square matrix to a vector (basically, linear transform). An eigenvalue of a matrix  $A$  is that it is any value  $\lambda$  which is a root of the characteristic equation of  $A$ . The characteristic equation is a polynomial equation of the form:

$$\det(A - \lambda I) = 0 \quad (\text{A1})$$

where  $\det(\cdot)$  denotes the determinant and  $I$  denotes the identity matrix.  $\lambda$  is an eigenvalue of  $A$  if and only if there is a nonzero vector  $v$ , known as an eigenvector, satisfying the following equation:

$$Av = \lambda v \quad (\text{A2})$$

For an  $N \times N$  matrix  $A$ , its characteristic equation (A1), has exactly  $N$  roots, so matrix  $A$  also has  $N$  eigenvalues.

Some important facts about eigenvalues of matrix  $A$  are as follows:

- If  $A$  is singular i.e. the determinant of  $A$  is 0, then 0 is an eigenvalue. In such circumstances,  $A$  cannot be inverted and hence, no unique solution to the linear system.
- If  $A$  is symmetric, all eigenvalues are real (not complex).

- If  $A$  is positive definite, all eigenvalues are positive; if  $A$  is negative definite, all eigenvalues are negative; and if  $A$  is indefinite, eigenvalues consist of both positive and negative values.

Eigenvalues are important in dealing with iterative solution methods because eigenvalues provide some indications on the convergence behavior of an iterative method for a linear system.

### A.1.5. Singular values of a matrix

Eigenvalues apply only to a square matrix; for a rectangular matrix, singular values may be used instead. Singular values and their corresponding singular vectors of an  $m \times n$  rectangular matrix  $A$  are the ones that satisfy the following:

$$A w = \sigma z \quad (\text{A3})$$

$$A^T z = \sigma w \quad (\text{A4})$$

For a symmetric positive definite matrix, the eigenvalues and eigenvectors are identical to its singular values and singular vectors, respectively. But, when  $A$  departs from symmetry or positive definiteness, the difference increases for both sets of parameters. In particular, the singular values of a real matrix are always real, but the eigenvalues of a real and non-symmetric matrix might be complex. Browne's Theorem (Marcus and Minc, 1992; cited in Chan, 2002) correlates eigenvalues and singular values of a real matrix using the following inequalities:

$$\sigma_{\min} \leq |\lambda|_{\min} \leq |\lambda| \leq |\lambda|_{\max} \leq \sigma_{\max} \quad (\text{A5})$$

where  $\sigma_{\min}$  and  $\sigma_{\max}$  denote the minimum and maximum singular values, respectively;  $|\lambda|_{\min}$  and  $|\lambda|_{\max}$  denote the absolute minimum and maximum eigenvalues, respectively; and  $|\lambda|$  denotes the modulus of an eigenvalue.

### **A.1.6. Rank of a matrix**

The rank of a matrix is the common value of its row rank and column rank.

The row (column) rank of a matrix is the dimension of the vector space spanned by its row (column) vectors, or is equal to the maximum number of linearly independent row (column) vectors of the matrix. A matrix is of full rank if its rank is equal to the smallest dimension of the matrix. A matrix is of full row (column) rank if its rank is equal to the dimension of its row (column).

### **A.1.7. Condition number of a matrix**

The stability or sensitivity of a linear system  $Ax = b$  can be determined from the condition number of the coefficient matrix  $A$  and occasionally by its clustering of eigenvalues (e.g. Shewchuk, 1994). The condition number is a positive number used to estimate the significance by which small errors in the right hand side vector  $b$ , or in the matrix  $A$  itself, can affect the solution  $x$ . In other words, the condition number is an approximate index indicating the amplification or diminution of round-off errors. Small values of the condition number indicate that the linear system will not be sensitive to errors, but large values suggest that small data errors or floating point arithmetic errors may incur enormous errors in the solution.

The condition number is usually defined in terms of matrix norms (see the definition next). In general, the condition number of a non-singular matrix  $A$  is defined by:

$$\kappa(A) = \|A\| \|A^{-1}\| \quad (\text{A6})$$

where  $\|\cdot\|$  denotes a matrix norm. For a symmetric positive definite matrix, the condition number can be defined as:

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} \quad (\text{A7})$$

where  $\lambda_{\max}(A)$  and  $\lambda_{\min}(A)$  denote the maximum and minimum eigenvalues of the matrix  $A$ , respectively. Generally, a matrix is said to be well-conditioned if the condition number is close to unity, and ill-conditioned if it is large. If the condition number is infinite, the matrix is singular. Eigenvalues of a well-conditioned matrix also tend to appear in a single tight cluster, while eigenvalues of an ill-conditioned matrix are much more widely spread. The study of condition number is important in predicting the convergence rate in the iterative solution method.

#### A.1.8. Vector and matrix norms

A norm is a way to measure the magnitude of an element (vector or matrix). It is like the absolute value of a number, but in more dimensions and there are more ways to combine the different components. A norm always has real values even for a complex vector space. It is usually denoted by a double bar notation  $\|\cdot\|$ .

The most important class of vector norms in connection with computations is the p-norm defined by:

$$\|x\|_p = \left( |x_1|^p + \dots + |x_n|^p \right)^{1/p} \text{ where } 1 \leq p \leq \infty \quad (\text{A8})$$

In practice, one usually takes  $p = 1, 2$  and  $\infty$ , that is:

$$\text{L1 vector norm: } \|x\|_1 = |x_1| + \dots + |x_n| = \sum_{i=1}^n |x_i| \quad (\text{A9})$$

$$\text{L2 vector norm: } \|x\|_2 = \left( |x_1|^2 + \dots + |x_n|^2 \right)^{1/2} = \left( \sum_{i=1}^n |x_i|^2 \right)^{1/2} = (x^T x)^{1/2} \quad (\text{A10})$$

$$\text{L infinity vector norm: } \|x\|_\infty = \max_{1 \leq i \leq n} |x_i| \quad (\text{A11})$$

L2 vector norm is also known as the *Euclidean vector norm* or *root-mean-square vector norm*.

Matrix norms are frequently used to estimate the effects of solving linear systems, matrix-vector multiplication, or other matrix operations. In particular, they are used in the analyses of error and convergence. The most commonly used matrix norms in numerical linear algebra are the p-norm and the Frobenius norm, as follows:

$$\text{p-norm:} \quad \|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} \text{ where } 1 \leq p \leq \infty \quad (\text{A12})$$

$$\text{Frobenius norm:} \quad \|A\|_F = \left( \sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 \right)^{1/2} \quad (\text{A13})$$

A p-norm is a vector-bound matrix norm, which is a matrix norm that can be derived from vector norms with the supremum (roughly equivalent to "maximum") is taken over all nonzero vector  $x$ , as shown in Equation (A12). Hence,  $\|A\|_p$  can be defined as the p-norm of the largest vector obtained by applying  $A$  to a unit p-norm vector, as follows:

$$\|A\|_p = \sup_{x \neq 0} \left\| A \left( \frac{x}{\|x\|_p} \right) \right\|_p = \max_{\|x\|_p=1} \|Ax\|_p \quad (\text{A14})$$

Similar to vector norms, the matrix norms are usually computed based upon  $p = 1, 2$  and  $\infty$ . In general, the computation of the L2 matrix norm is rather expensive, so it is often simpler to use the more easily computed Frobenius matrix norm (A13) instead. A matrix norm and a vector norm are compatible or consistent if it is true, for all vectors  $x$  and matrices  $A$  that:

$$\|Ax\| \leq \|A\| \|x\| \quad (\text{A15})$$

[Note: most of the above definitions are taken from (Chan, 2002)].

## A.2. Iterative solution methods

The term ‘iterative methods’ refers to a wide range of techniques that use successive approximations to obtain more accurate solutions to a linear system starting from an initial guess. The essential feature of iterative methods is that they require significantly smaller memory and less runtime for large-scale problems. Increasing popularity of iterative methods indicates that they are preferred over direct methods for the solution of linear FE equations arising from large-scale problems.

Iterative methods are broadly categorized into two basic types: stationary or classical iterative methods and nonstationary iterative methods (Barrett *et al.*, 1994). In stationary iterative methods, each iteration follows the same recipe without iteration dependency. The main stationary iterative methods are: Jacobi, Gauss-Seidel, SOR, and SSOR methods. These methods are usually less effective; however, they can be used as preconditioners for nonstationary methods.

Nonstationary methods are relatively recent developments and differ from stationary methods in that the computations involve information that changes at each iteration. Such iterative methods are often referred to as Krylov subspace methods. The attractiveness of these methods for large sparse linear system of equations is that they reference the coefficient matrix only through its multiplication with a vector, or the multiplication of its transpose and a vector. Krylov subspace methods form an orthogonal basis of the sequence of successive matrix powers times the initial residual (the Krylov sequence). That is,

$$K_k(v, A) = \text{span}\{v, Av, \dots, A^{k-1}v\} \quad \text{for } k \geq 0 \quad (\text{A16})$$

where  $K_k(v, A)$  is called  $k$ -th Krylov subspace generated by  $A$  with respect to  $v$ . It is clear that the subspace depends on the initial vector  $v$ . Typically, it is chosen as the initial residual vector of the linear system, i.e.  $v = r_0 = b - Ax_0$ . Then, the iterate  $x_k$  is updated as follows:

$$x_k \in x_0 + K_k(r_0, A), \quad k = 1, 2, \dots \quad (\text{A17})$$

Some of the popular nonstationary iterative methods are:

1. Conjugate Gradient (CG)
2. Minimum Residual (MINRES) and Symmetric LQ (SYMMLQ)
3. Generalized Minimal Residual (GMRES)
4. Biconjugate Gradient (BiCG)
5. Quasi-Minimal Residual (QMR)
6. Transpose-Free Quasi-Minimal Residual (TFQMR)
7. Symmetric Quasi-Minimal Residual (SQMR)
8. Conjugate Gradient Squared (CGS)
9. Biconjugate Gradient Stabilized (Bi-CGSTAB)

These methods have been intensively re-examined over past decades for their merits and demerits (e.g. Axelsson, 1994; Barrett *et al.*, 1994; Saad, 1996). Each of the above mentioned methods has its own specific function and applications. The choice of an optimal method is largely depends on the properties of the coefficient matrix (See Figure 2.1, Chapter 2). The methods that are used in this thesis are:

### **A.2.1. Conjugate Gradient (CG)**

The conjugate gradient (CG) method is originally proposed firstly by Hestenes and Stiefel (1952). CG is presented as an iterative method for large sparse

system of linear equations after Reid (1971). The algorithm begins with an initial guess for  $x$  followed by successive updates based on residuals. The method creates search directions that are orthogonal so that the method must converge in a maximum of  $N$  steps, where  $N$  is the size of the matrix  $A$  (1.1). The convergence rate of CG depends on the condition number of  $A$  (Section 0). However, the conditioner number is only a general indicator; clustered eigenvalues usually results in faster convergence (Shewchuk, 1994). To accelerate the convergence, the method is often used in conjunction with preconditioners. Hence, it is commonly referred as preconditioned conjugate gradient method (PCG). The pseudo code of Preconditioned Conjugate Gradient (PCG) is as given below. The linear system is  $Ax = b$  and  $M$  is the symmetric preconditioner (e.g. Barrett *et al.*, 1994; van der Vorst, 2003).

```

Start: Choose an initial guess  $x_0 \in \mathfrak{R}^N$ 
          Set  $r_0 = b - Ax_0$ 
For  $k = 1$  to  $\text{max\_it}$  Do
    Solve  $Mw_{k-1} = r_{k-1}$ 
     $\rho_{k-1} = r_{k-1}^T w_{k-1}$ 
    if  $k = 1$  then
         $p_k = w_{k-1}$ 
    else
         $\beta_{k-1} = \rho_{k-1} / \rho_{k-2}$ 
         $p_k = w_{k-1} + \beta_{k-1} p_{k-1}$ 
    end if
     $q_k = Ap_k$ 
     $\alpha_k = \rho_{k-1} / p_k^T q_k$ 
     $x_k = x_{k-1} + \alpha_k p_k$ 
     $r_k = r_{k-1} - \alpha_k q_k$ 
    Check convergence. If converged, exit Do loop.
End Do

```

### A.2.2. Symmetric Quasi-Minimal Residual (SQMR)

For symmetric indefinite linear system, SQMR (Freund and Nachtigal, 1994b) is a preferred choice. It can be interpreted as a special case of QMR method (Freund and Nachtigal, 1991, 1994a), which was initially proposed for unsymmetric systems. The pseudo code for SQMR is as given below:

The symmetric linear system is  $Ax = b$  and the symmetric preconditioner is  $M = M_L M_R$  (Freund and Nachtigal, 1994b).

<p><b>Start:</b> Choose an initial guess <math>x_0 \in \Re^N</math></p> <p>Set</p> $r_0 = b - Ax_0, \quad t = M_L^{-1} r_0, \quad q_0 = M_R^{-1} t, \quad \tau_0 = \ t\ _2,$ $\theta_0 = 0, \quad \rho_0 = r_0^T q_0, \quad d_0 = 0$ <p><b>For</b> <math>i = 1</math> <b>to</b> <math>\max\_it</math> <b>Do</b></p> <p>(1) Compute</p> $t = Aq_{i-1}, \quad \sigma_{i-1} = q_{i-1}^T t, \quad \alpha_{i-1} = \frac{\rho_{i-1}}{\sigma_{i-1}}, \quad r_i = r_{i-1} - \alpha_{i-1} t$ <p>(2) Compute</p> $t = M_L^{-1} r_i, \quad \theta_i = \frac{\ t\ _2}{\tau_{i-1}}, \quad c_i = \frac{1}{\sqrt{1 + \theta^2}}, \quad \tau_i = \tau_{i-1} \theta_i c_i$ $d_i = c_i^2 \theta_{i-1}^2 d_{i-1} + c_i^2 \alpha_{i-1} q_{i-1}$ <p>(3) Set</p> $x_i = x_{i-1} + d_i$ <p>Check convergence. If converged, exit Do loop.</p> <p>(4) Compute</p> $s_i = M_R^{-1} t, \quad \rho_i = r_i^T s_i, \quad \beta_i = \frac{\rho_i}{\rho_{i-1}}, \quad q_i = s_i + \beta_i q_{i-1}$ <p><b>End Do</b></p>
---

### A.3. Sparse storage of the matrix

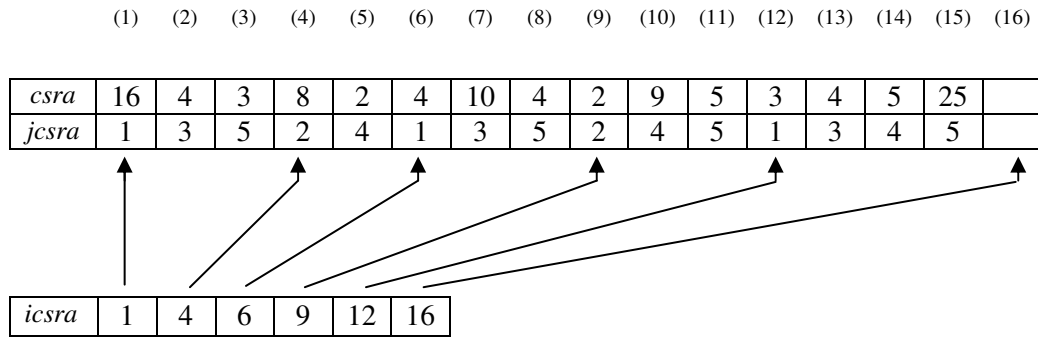
Several sparse storage schemes are available in which only the nonzero elements of the sparse matrix are stored and the matrix operation (multiplication, etc.) can be performed effectively. The details of all these are beyond the scope of this thesis. Readers are referred to the templates by Barrett *et al.* (1994) or the book by Saad (1996) for more details. The storage schemes used in this thesis are described below. These are the standard storage schemes and are widely used in numerical analyses.

#### A.3.1. Compressed sparse row (CSR) storage

In this scheme all the nonzero entries of FE coefficient matrix  $A$  are stored row by row in three one-dimensional arrays as shown below:

Let

$$A = \begin{bmatrix} 16 & 0 & 4 & 0 & 3 \\ 0 & 8 & 0 & 2 & 0 \\ 4 & 0 & 10 & 0 & 4 \\ 0 & 2 & 0 & 9 & 5 \\ 3 & 0 & 4 & 5 & 25 \end{bmatrix}$$

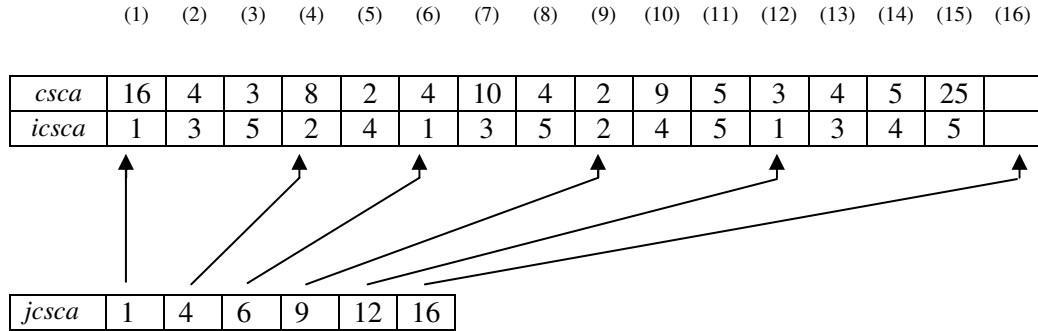


- A real array *csra* contains the real values  $a_{ij}$  stored row by row, from 1 to  $N$ . The length of *csra* is  $nnz$  (number of nonzero entries in  $A$ ).
- An integer array *jcsra* contains the column indices of the elements  $a_{ij}$ . The length of *jcsra* is also  $nnz$ .
- An integer array *icsra* contains the pointers to the beginning of each row in the arrays *csra* and *jcsra*. Thus, the content of *icsra*( $i$ ) is the position in arrays *csra* and *jcsra* where the  $i$ -th row starts. The length of *icsra* is  $N + 1$  with *icsra*( $N + 1$ ) containing the number *icsra*(1) +  $nnz$ , i.e. the address in *csra* and *jcsra* of the beginning of a fictitious row number  $N + 1$ .

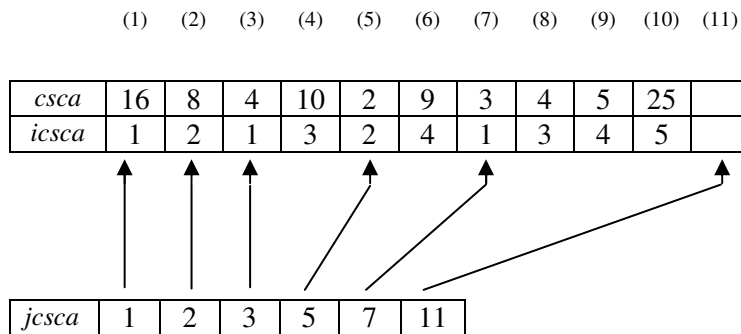
### A.3.2. Compressed sparse column (CSC) storage

The ‘compressed sparse column’ format is identical with the ‘compressed sparse row’ format except that the columns of  $A$  are scanned and stored instead of the rows. In other words, the CSC format is simply the CSR format

for the matrix  $A^T$ . The arrays  $cscA$ ,  $icscA$ , and  $jcsca$  are used for the storage of entries of  $A$ , row indices of the entries, and the pointers to the beginning of each column as shown below:



For symmetric matrices, both CSC and CSR storages are the same. In addition, the matrix operations can be performed by only storing the symmetric upper (or lower) triangular part of  $A$ . For example, the CSC storage of upper  $A$  is shown below:



Clearly, the CSC storage of upper triangular is equivalent to the CSR storage of lower triangular. In this study, the sparse storage of the coefficient matrix  $A$  is done by assembling the global stiffness matrix in element loop (Figure 2.2, Chapter 2). The three vectors ( $iebea$ ,  $jebea$ , and  $ebea$ ) are used to store the global row number, global column number, and corresponding value of  $A$  for the symmetric upper (or lower) triangular part for each element. For CSR

storage, the vector *iebea* (*jebea* for CSC storage) is sorted in ascending order and corresponding reordering on *jebea* and *ebea* is carried out simultaneously. In the next step, *jebea* is sorted with simultaneous reordering of *ebea* for the same *iebea*. Finally, the entries with the same *iebea* and *jebea* number are summed up to get the new three global vectors *icsra*, *jcsra*, and *csra* (or *icsca*, *jcsca*, and *csca* for CSC storage) for global sparse stiffness matrix  $A$ . For sorting the arrays with 16 or less entries, insertion sorting is adopted otherwise quick sorting for efficiency (Press *et al.*, 1996; Nyhoff and Leestma, 1997). The previous three element-level vectors would be deallocated for the memory management.

## APPENDIX B

---

### BIOT'S CONSOLIDATION EQUATIONS

Biot (1941) put forward the three-dimensional soil consolidation theory as a further development of Terzaghi's one-dimensional soil consolidation theory. In terms of Biot's definition, soil consolidation is the process of a gradual ground water flow and solid skeleton deformation of the porous medium. In Biot's formulation the groundwater flow within the soil is fully accounted for based on Darcy's law. This means that if the time increment is very short solution of the coupled consolidation analysis is akin to an undrained analysis and if the time is very long, the analysis will converge to a drained analysis. Another way of understanding this is that the end state of the consolidation process is the drained state.

#### **B.1. Biot's consolidation equations**

In Biot's theory, soil is regarded as a porous skeleton filled with water, and the interaction between soil skeleton and pore water is determined by the principle of effective stress and the continuity relation. When taking an infinitesimal soil element, the equilibrium equations of this element can be expressed as:

$$\begin{aligned}
\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + b_x &= 0 \\
\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + b_y &= 0 \\
\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + b_z &= 0
\end{aligned} \tag{B1}$$

where  $b_x$ ,  $b_y$ , and  $b_z$  are the body forces, per unit volume, in the  $x$ ,  $y$ , and  $z$  directions. With an ordinary gravity field and  $z$  direction vertically downwards,  $b_x$  and  $b_y$  are zero and  $b_z$  is the unit weight,  $\gamma$ , of the material.

In compact form, Equation (B1) can be written as:

$$\tilde{\nabla}^T \sigma + b = 0 \tag{B2}$$

where  $\tilde{\nabla}^T$  is a differential operator given by:

$$\tilde{\nabla}^T = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \tag{B3}$$

According to principle of effective stress, the total stress is equal to the summation of effective stress and pore water pressure, i.e.  $\sigma = \sigma' + \bar{m} p$ .

Thus, Equation (B2) can be written as:

$$\tilde{\nabla}^T (\sigma' + \bar{m} p) + b = 0 \tag{B4}$$

where  $\sigma' = \{\sigma'_x, \sigma'_y, \sigma'_z, \tau_{xy}, \tau_{yz}, \tau_{zx}\}^T$  is the vector of effective stresses,

$\bar{m} = \{1, 1, 1, 0, 0, 0\}^T$  is a second-order Kronecker delta in vectorial form, and

$p = p^{st} + p^{ex}$  is the total pore water pressure decomposed into steady state component,  $p^{st}$ , and excess component,  $p^{ex}$  (pore pressure in excess of that at steady state), respectively.

For a linear elastic solid element, the stress-strain relationship is given as:

$$\sigma' = D^e \varepsilon \quad (\text{B5})$$

where  $\varepsilon = \{\varepsilon_x, \varepsilon_y, \varepsilon_z, \tau_{xy}, \tau_{yz}, \tau_{zx}\}^T$  is the strain vector, and  $D^e$  is the elastic stress-strain matrix given as:

$$D^e = \frac{E'}{(1+\nu')(1-2\nu')} \begin{bmatrix} 1-\nu' & \nu' & \nu' & 0 & 0 & 0 \\ \nu' & 1-\nu' & \nu' & 0 & 0 & 0 \\ \nu' & \nu' & 1-\nu' & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5-\nu' & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5-\nu' & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5-\nu' \end{bmatrix} \quad (\text{B6})$$

where  $E'$  is the effective young's modulus and  $\nu'$  is the effective Poisson's ratio. The strain vector is related to the displacement vector in terms of:

$$\varepsilon = B_u u_e \quad (\text{B7})$$

where  $u_e = \{u_{x1}, u_{y1}, u_{z1}, \dots, u_{xk}, u_{yk}, u_{zk}\}$  is the vector of nodal displacement for a  $k$ -node solid element and  $B_u$  is the strain-displacement matrix, given by  $B_u = \tilde{\nabla} N_u$ . The displacement shape function matrix  $N_u$ , and its derivatives are given by:

$$N_u = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & \cdots & N_k & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & \cdots & 0 & N_k & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & \cdots & 0 & 0 & N_k \end{bmatrix}_{3 \times 3k} \quad (\text{B8})$$

$$B_u = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \dots & \frac{\partial N_k}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & \dots & 0 & \frac{\partial N_k}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & \dots & 0 & 0 & \frac{\partial N_k}{\partial z} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & 0 & \dots & \frac{\partial N_k}{\partial y} & \frac{\partial N_k}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial z} & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial z} & \frac{\partial N_2}{\partial y} & \dots & 0 & \frac{\partial N_k}{\partial z} & \frac{\partial N_k}{\partial y} \\ \frac{\partial N_1}{\partial z} & 0 & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial z} & 0 & \frac{\partial N_2}{\partial x} & \dots & \frac{\partial N_k}{\partial z} & 0 & \frac{\partial N_k}{\partial x} \end{bmatrix}_{6 \times 3k} \quad (B9)$$

Another relation between velocity (or flux) and pore water pressure is given by the continuity equation. Physically, this means that the volume of fluid flowing in or out is equal to the volume of change of the soil mass (if no sources or sinks are considered).

$$\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} + \frac{\partial \varepsilon_v}{\partial t} = 0 \quad (B10)$$

where  $\varepsilon_v = \varepsilon_x + \varepsilon_y + \varepsilon_z = \bar{m}^T \varepsilon$  is the volumetric strain. Equation (B10) can be expressed in compact form as:

$$\text{div } q + \bar{m}^T \dot{\varepsilon} = 0 \quad (B11)$$

Here  $q = \{q_x, q_y, q_z\}$  is the vector of volumetric flow rates per unit area into and out of the element. These components in coordinate directions can be determined by Darcy's law:

$$\begin{Bmatrix} q_x \\ q_y \\ q_z \end{Bmatrix} = \frac{1}{\gamma_w} \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix} \begin{Bmatrix} \frac{\partial p}{\partial x} - b_{wx} \\ \frac{\partial p}{\partial y} - b_{wy} \\ \frac{\partial p}{\partial z} - b_{wz} \end{Bmatrix} \quad (\text{B12})$$

Or, in a compact form:

$$q = \frac{[k]}{\gamma_w} (\nabla p - b_w) \quad (\text{B13})$$

where  $[k]$  is the permeability matrix,  $\gamma_w$  unit weight of pore water pressure, taken as 10 kN/m<sup>3</sup> in this study, and  $b_w = \{b_{wx}, b_{wy}, b_{wz}\}^T$ . Combining Equations (B11) and (B13):

$$\text{div} \left[ \frac{[k]}{\gamma_w} (\nabla p - b_w) \right] + \bar{m}^T \dot{\epsilon} = 0 \quad (\text{B14})$$

Equations (B4) and (B14) constitute the Biot's consolidation equations.

To carry out the finite element analysis of Biot's consolidation problem, it is required to discretize the consolidation equations in space domain and time domain, respectively. Usually weighted residual Galerkin method is adopted for the discretization of space domain. After spatial discretization and applying weighting residual method, we would obtain the global coupled equation as:

$$\begin{aligned} Ku + Bp^{ex} &= f \\ B^T \dot{u} - Hp^{ex} &= 0 \end{aligned} \quad (\text{B15})$$

where,

$$\begin{aligned}
K &= \sum_e \left( \int_V B_u^T D^e B_u dV \right) \\
B &= \sum_e \left( \int_V B_p^T \bar{m} N_p dV \right) \\
H &= \sum_e \left( \int_V B_p^T \frac{[k]}{\gamma_w} N_p dV \right) \\
f &= \sum_e f_e
\end{aligned} \tag{B16}$$

$K$  and  $H$  are solid and fluid stiffness matrices, respectively, and  $B$  is the connection matrix. The vectors  $u$  and  $p^{ex}$  are displacement and excess pore water pressure, respectively, and  $f$  is the current magnitude of load. When nonlinear elasto-plastic soil behavior is considered, the stress-strain relation is determined by the elasto-plastic stress-strain matrix  $D^{ep}$ .

Using finite difference scheme in time domain, we would obtain the incremental formulation of Biot's consolidation equation as:

$$\begin{bmatrix} K & B \\ B^T & -\theta \Delta t H \end{bmatrix} \begin{Bmatrix} \Delta u \\ \Delta p^{ex} \end{Bmatrix} = \begin{Bmatrix} \Delta f \\ \Delta t H p^{ex} \end{Bmatrix} \tag{B17}$$

where  $\theta$  is a time integrating parameter. The choice of  $\theta = 1/2$  leads to the Crank-Nicolson approximation method, however, oscillatory results may be incurred by this approximation, and thus, the fully implicit method by choosing  $\theta = 1$  is often used (e.g. Smith and Griffiths, 1997). For consolidation analysis, Equation (B17) needs to be solved for each time step  $\Delta t$ . This symmetric indefinite linear system (B17) can be written in general form as:

$$\begin{bmatrix} K & B \\ B^T & -C \end{bmatrix} \begin{Bmatrix} \Delta u \\ \Delta p^{ex} \end{Bmatrix} = \begin{Bmatrix} \Delta f \\ C p^{ex} \end{Bmatrix} \tag{B18}$$

In compact form,

$$Ax = b \tag{B19}$$

Here,  $A \in \Re^{N \times N}$  is a sparse 2x2 block symmetric indefinite matrix,  $K \in \Re^{nd \times nd}$  is soil stiffness matrix (symmetric positive definite),  $C = \theta \Delta t H \in \Re^{np \times np}$  is fluid stiffness matrix (symmetric positive semi-definite), and  $B \in \Re^{nd \times np}$  is the displacement-pore pressure coupling matrix. When employing the Sylvesters's inertia theorem, the congruence transform is:

$$A = \begin{bmatrix} K & B \\ B^T & -C \end{bmatrix} = \begin{bmatrix} I & 0 \\ B^T K^{-1} & I \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & -S \end{bmatrix} \begin{bmatrix} I & K^{-1} B \\ 0 & I \end{bmatrix} \quad (B20)$$

indicates the indefiniteness because  $A$  has  $nd$  positive eigenvalues and  $np$  negative eigenvalues (e.g. Wathen and Silvester, 1993).

$$S = C + B^T K^{-1} B \quad (B21)$$

is called Schur complement matrix. For a detailed derivation, the reader is referred to Smith and Griffiths (1997), Lewis and Schrefler (1998), Chen (2005).

*(blank)*

## APPENDIX C

---

### ALGORITHMS

#### C.1. SSOR-PCG

This algorithm describes SSOR ( $\omega=1.0$ ) preconditioned PCG algorithm (Meurant, 1999) for the symmetric positive definite linear system  $Ax = b$  with Eisenstat trick (Eisenstat, 1981).

```
Start: Choose an initial guess  $x_0 \in \Re^n$ , then set  $z_0 = 0$   
Compute  $r_0 = (L_A + D_A)^{-1}b$ ,  $s_0 = D_A r_0$ ,  $p_0 = s_0$   
For  $k = 0$  to  $\text{max\_it}$  Do  
     $t_k = \tilde{A}p_k$  (carried out by Procedure PMatvec)  
     $\alpha_k = (r_k, s_k) / (t_k, p_k)$   
     $z_{k+1} = z_k + \alpha_k p_k$   
     $r_{k+1} = r_k - \alpha_k t_k$   
     $s_{k+1} = D_A r_{k+1}$   
    Check for convergence, if converged,  
        Set  $x_{k+1} = x_0 + (D_A + L_A^T)^{-1} z_{k+1}$   
        Exit Do loop  
    Else  
         $\beta_{k+1} = (r_{k+1}, s_{k+1}) / (r_k, s_k)$   
         $p_{k+1} = s_{k+1} + \beta_{k+1} p_k$   
End Do
```

Procedure PMatvec:

1.  $f = (D_A + L_A^T)^{-1} p_k$
2.  $g = D_A f$
3.  $h = (D_A + L_A)^{-1} (p_k - g)$
4.  $t_k = f + h$

## C.2. SBD<sub>2</sub>-PCG

PCG algorithm (Meurant, 1999) for the symmetric positive definite linear system  $Ku = f$  by  $[P, \text{SSOR}(G)]$  block diagonal preconditioner using Eisenstat trick (Eisenstat, 1981) for block (2,2), see also (Chen *et al.*, 2007).

**Start:** Choose an initial guess  $u_0 \in \Re^N$ , then set  $z_0 = 0$

$$\text{Compute } s_0 = \left\{ \begin{array}{l} R_P^{-T} f_1 \\ \tilde{D}_G^{1/2} (L_G + \tilde{D}_G)^{-1} f_2 \end{array} \right\}, \quad p_0 = s_0$$

**For**  $k = 0$  **to**  $\text{max\_it}$  **Do**

$$t_k = \tilde{K} p_k \quad (\text{carried out by Procedure PMatvec})$$

$$\alpha_k = (s_k, s_k) / (t_k, p_k)$$

$$z_{k+1} = z_k + \alpha_k p_k$$

$$s_{k+1} = s_k - \alpha_k t_k$$

Check for convergence, if converged,

$$\text{Set } u_{k+1} = u_0 + \left\{ \begin{array}{l} R_P^{-1} z_{k+1} \\ (\tilde{D}_G + L_G^T)^{-1} \tilde{D}_G^{1/2} z_{k+1} \end{array} \right\}$$

Exit Do loop

Else

$$\beta_{k+1} = (s_{k+1}, s_{k+1}) / (s_k, s_k)$$

$$p_{k+1} = s_{k+1} + \beta_{k+1} p_k$$

**End Do**

Procedure PMatvec  $[t = \tilde{K}p \rightarrow (t_1; t_2) = \tilde{K}(p_1; p_2)]$ .

1.  $f = (L_G^T + \tilde{D}_G)^{-1} w$  where  $w = \tilde{D}_G^{1/2} p_2$
2.  $q_1 = R_p^{-T} (L^* f)$
3.  $t_1 = p_1 + q_1$
4.  $q_2 = L^T (R_p^{-1} p_2)$
5.  $g = (D_G - 2 * \tilde{D}_G) * f + w$
6.  $y = (L_G + \tilde{D}_G)^{-1} (q_2 + g)$
7.  $t_2 = \tilde{D}_G^{1/2} (y + f)$

### C.3. $M_3$ -SQMR

The coefficient matrix in  $3 \times 3$  block form (5.4) is:

$$A = \begin{bmatrix} P & L & B_1 \\ L^T & G & B_2 \\ B_1^T & B_2^T & -C \end{bmatrix},$$

and the block diagonal preconditioner is:

$$M_3 = \begin{bmatrix} P & 0 & 0 \\ 0 & \text{SSOR}(G) & 0 \\ 0 & 0 & \alpha \text{diag}(\hat{S}_1) \end{bmatrix}$$

Let,

$$M_3 = \begin{bmatrix} R_p^T & 0 & 0 \\ 0 & (L_G + \tilde{D}_G) & 0 \\ 0 & 0 & I_{np} \end{bmatrix} \begin{bmatrix} I_m & 0 & 0 \\ 0 & \tilde{D}_G^{-1} & 0 \\ 0 & 0 & \alpha \text{diag}(\hat{S}_1) \end{bmatrix} \begin{bmatrix} R_p & 0 & 0 \\ 0 & (L_G^T + \tilde{D}_G) & 0 \\ 0 & 0 & I_{np} \end{bmatrix}$$

$$M_3 = \bar{L}_A \bar{D}_A^{-1} \bar{L}_A^T$$

where,

$P = R_p^T R_p$  is Cholesky factorization of block  $P$ ,

$G = (L_G + \tilde{D}_G) \tilde{D}_G^{-1} (L_G^T + \tilde{D}_G)$  is SSOR factorization of block  $G$ , and

$\hat{S}_1 = C + B_1^T \{\text{diag}(P)\}^{-1} B_1 + B_2^T \{\text{diag}(G)\}^{-1} B_2$  is an approximate Schur

complement.

$L_G$  and  $D_G$  are the strictly lower and diagonal parts of  $G$ ,  $\tilde{D}_G = D_G/\omega$ ,  $\omega$  is relaxation parameter (taken as 1 in this study),  $\alpha$  is a real parameter (taken as -4 in this study).

**Start:** Choose an initial guess  $x_0 \in \Re^N$ , then set  $z_0 = 0 \in \Re^N$

Compute  $r_0 = b - Ax_0$

$$s_0 = \bar{L}_A^{-1} r_0 = \begin{Bmatrix} R_P^{-T} r_{01} \\ (L_G + \tilde{D}_G)^{-1} r_{02} \\ r_{03} \end{Bmatrix}, \quad v_0 = s_0, \quad w_0 = \bar{D}_A v_0 = \begin{Bmatrix} v_{01} \\ \tilde{D}_G v_{02} \\ \{\alpha \text{diag}(\hat{S}_1)\}^{-1} v_{03} \end{Bmatrix}$$

Compute  $\tau_0 = s_0^T s_0$ , and  $\rho_0 = s_0^T w_0$

set  $d_0 = 0 \in \Re^N$ , and  $v_0 = 0 \in \Re$

**For**  $k = 1$  **to**  $\max\_it$  **Do**

Compute

$$t_{k-1} = \tilde{A} v_{k-1} \quad (\text{Procedure PMatvec})$$

$$\sigma_{k-1} = w_{k-1}^T t_{k-1}, \quad \text{if } \sigma_{k-1} = 0, \text{ then STOP}$$

$$\alpha_{k-1} = \frac{\rho_{k-1}}{\sigma_{k-1}} \text{ and } s_k = s_{k-1} - \alpha_{k-1} t_{k-1}$$

Compute

$$\vartheta_k = \frac{s_k^T s_j}{\tau_{k-1}}, \quad c_k = \frac{1}{1 + \vartheta_k}, \quad \tau_k = \tau_{k-1} \vartheta_k c_k$$

$$d_k = c_k (\vartheta_{k-1} d_{k-1} + \alpha_{k-1} v_{k-1})$$

$$\text{Set } z_k = z_{k-1} + d_k$$

Check convergence, if converged, then

$$x_k = x_0 + \bar{L}_A^{-T} \bar{D}_A z_k = \begin{Bmatrix} R_P^{-1} z_{1k} \\ (L_G^T + \tilde{D}_G)^{-1} \tilde{D}_G z_{2k} \\ \{\alpha \text{diag}(\hat{S}_1)\}^{-1} z_{3k} \end{Bmatrix}$$

Exit Do loop

Compute

$$q_k = \bar{D}_A s_k = \begin{Bmatrix} s_{1k} \\ \tilde{D}_G s_{2k} \\ \{\alpha \text{diag}(\hat{S}_1)\}^{-1} s_{3k} \end{Bmatrix}, \quad \rho_k = s_k^T q_k, \quad \beta_k = \frac{\rho_k}{\rho_{k-1}}$$

$$v_k = s_k + \beta_k v_{k-1} \text{ and } w_k = \bar{D}_A v_k = \begin{Bmatrix} v_{1k} \\ \tilde{D}_G v_{2k} \\ \{\alpha \text{diag}(\hat{S}_1)\}^{-1} v_{3k} \end{Bmatrix}$$

**End Do**

PMatvec (Preconditioned matrix-vector multiplication  $t = M_3^{-1}Av$  )

$$\text{i.e. } [t_1; t_2; t_3] = \tilde{A}[v_1; v_2; v_3]$$

where,

$$\tilde{A} = \begin{bmatrix} I_m & R_p^{-T} L (L_G^T + \tilde{D}_G)^{-1} \tilde{D}_G & R_p^{-T} B_1 \{\alpha \text{diag}(\hat{S}_1)\}^{-1} \\ (L_G + \tilde{D}_G)^{-1} L^T R_p^{-1} & (L_G + \tilde{D}_G)^{-1} G (L_G^T + \tilde{D}_G)^{-1} \tilde{D}_G & (L_G + \tilde{D}_G)^{-1} B_2 \{\alpha \text{diag}(\hat{S}_1)\}^{-1} \\ B_1^T R_p^{-1} & B_2^T (L_G^T + \tilde{D}_G)^{-1} \tilde{D}_G & -C \{\alpha \text{diag}(\hat{S}_1)\}^{-1} \end{bmatrix}$$

The algorithm is:

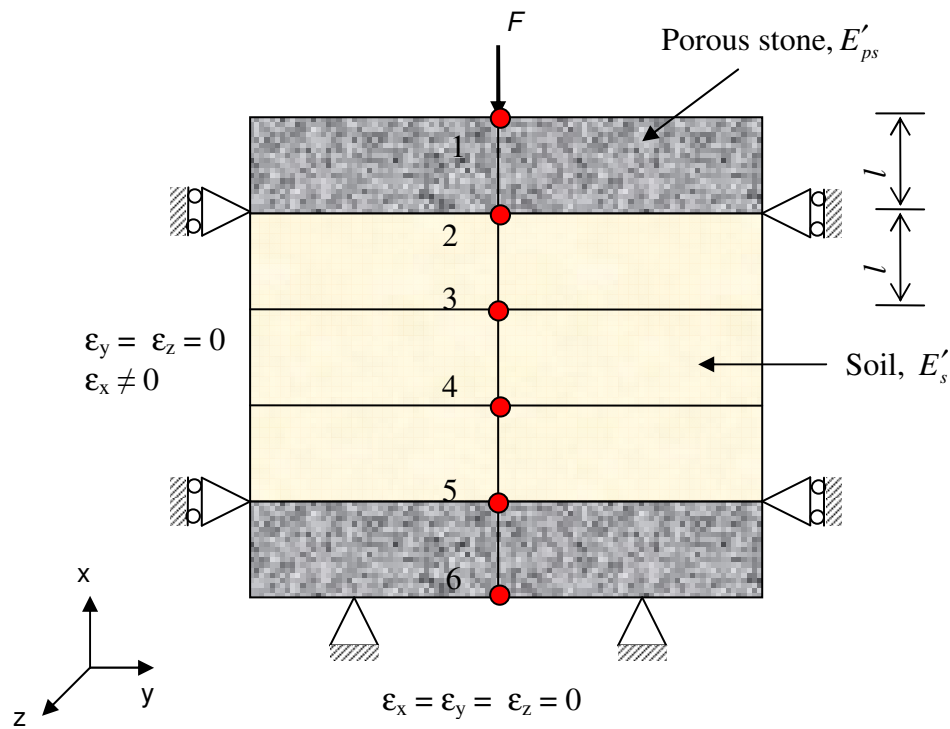
1.  $w_3 = \{\alpha \text{diag}(\hat{S}_1)\}^{-1} v_3$ ;  $f_3 = B_1 w_3$
2.  $w_2 = \tilde{D}_G v_2$ ;  $f_2 = (L_G^T + \tilde{D}_G)^{-1} w_2$
3.  $f_1 = L^* f_2 + f_3$ ;  $q_1 = R_p^{-T} f_1$
4.  $t_1 = v_1 + q_1$
5.  $g_3 = B_2^* w_3$
6.  $g_2 = (D_G - 2^* \tilde{D}_G)^* f_2 + w_2$
7.  $u_1 = R_p^{-1} v_1$ ;  $g_1 = L^T u_1$
8.  $q_2 = g_1 + g_2 + g_3$
9.  $h_2 = (L_G + \tilde{D}_G)^{-1} q_2$
10.  $t_2 = f_2 + h_2$
11.  $y_1 = B_1^T u_1$
12.  $y_2 = B_2^T f_2$
13.  $t_3 = y_1 + y_2 - C w_3$

*(blank)*

## APPENDIX D

---

### 1D FINITE ELEMENT DISCRETIZATION OF OEDOMETER TEST SET UP



The dots and numbers besides them are finite element nodes and node numbers,  $F$  is the applied load,  $l$  is the element size,  $E'_{ps}$  and  $E'_s$  are the effective Young's moduli of porous stone and soil, respectively.

Finite element model of the entire problem is obtained by assembling the element equations and applying the boundary condition, which yields after reordering the variables:

$$\frac{a}{l} \begin{bmatrix} D_{ps} & -D_{ps} & 0 & 0 & 0 \\ -D_{ps} & D_{ps} + D_s & 0 & -D_s & 0 \\ 0 & 0 & D_{ps} + D_s & 0 & -D_s \\ 0 & -D_s & 0 & 2D_s & -D_s \\ 0 & 0 & -D_s & -D_s & 2D_s \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_5 \\ u_3 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} F \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \quad (D1)$$

where  $a$  is the cross-sectional area of the element,  $D_{ps}$  and  $D_s$  are constrained modulus of porous stone and soil elements. Assuming a unit length of element with unit cross-sectional area, the various submatrices can be written as:

$$P = D_{ps} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 + \chi & 0 \\ 0 & 0 & 1 + \chi \end{bmatrix}, \quad G = D_s \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}, \quad \text{and} \quad L = D_s \begin{bmatrix} 0 & 0 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \quad (D2)$$

in which  $\chi = D_s / D_{ps} (\cong E'_s / E'_{ps})$ . Let  $R_P$  and  $R_G$  be Cholesky factors of  $P$  and  $G$ , respectively. Then,  $R_P$  and  $R_G$  are now computed as:

$$R_P = \sqrt{D_{ps}} \begin{bmatrix} 1 & -1 & 0 \\ 0 & \sqrt{\chi} & 0 \\ 0 & 0 & \sqrt{1 + \chi} \end{bmatrix} \quad (D3)$$

$$R_G = \sqrt{D_s} \begin{bmatrix} \sqrt{2} & -\frac{1}{\sqrt{2}} \\ 0 & \sqrt{\frac{3}{2}} \end{bmatrix}. \quad (D4)$$

Now, it can easily be shown that

$$\tilde{L} = R_P^{-T} L R_G^{-1} = \begin{bmatrix} 0 & 0 \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} \\ 0 & -\sqrt{\frac{2}{3}} \sqrt{\frac{\chi}{1 + \chi}} \end{bmatrix} \quad (D5)$$

## APPENDIX E

---

### SOURCE CODES IN FORTRAN 90

These programs are extensions of programs in the book “*Programming the finite element method*” by Smith and Griffiths (1997) for 3D analysis using sparse iterative methods. Hence, the modules ‘new\_library’ and ‘geometry\_lib’ are taken from the above book. Module ‘spchol’ contains the sparse Cholesky factorization routines authored by Ng and Peyton (1993). Module ‘sparselib\_v3’ is an extension of the sparse\_lib package ([http://www.eng.nus.edu.sg/civil/people/cvepkk/sparse\\_lib.html](http://www.eng.nus.edu.sg/civil/people/cvepkk/sparse_lib.html)) [see also Chen (2005)] developed at National University of Singapore. The above mentioned webpage also provides a user manual on how to use this sparse\_lib package with finite element package. Hence, only some new additions to the sparse\_lib package that are more relevant to this thesis are provided in this Section.

#### E.1. Main program for SBD<sub>1</sub>-PCG

```
!-----  
program sbdlpcgmain  
!-----  
! Soil-structure interaction analysis with drained parameters  
!-----  
!   Program for 3-D pile group analysis using 20-node solid  
!   brick elements  
!   PCG solver is used for solving the linear system  $Ax = b$   
!   in each time step  
!    $A = [P \ L ; L' \ G]$ 
```

```

!  ==> P : Pile Block, G = Geo Block, and L = Link matrix
!  Block diagonal preconditioner
!          M(-1) = [ P(-1)          0;
!                   0      (diag(G))-1 ]
!-----
use new_library ; use geometry_lib; use sparselib_v3;
use dfport ; use spchol ;
implicit none
integer:: i,j,k,l,nn,nels,nxe,nye,nze,nip,nodof=3,nod=20, &
nstep=6,ndim=3,ndof,iel,ns,nstep,inc,loaded_nodes, &
neq,nband,nmesh,ebeanz,maxit,itors,isolver,icho, &
ipre,icc,iinc,uanz,np_types,npiles,nels_pile, &
gneq,pneq,ir,k1,k2,pnz,gnz,llnz,anz,nonzp,ierr, &
punz,gunz,iwsiz,nsup,iflag,nnzl,nsuper,nnzlmax, &
tmpsiz,tmpmax,cachsz,level,nxr,nyr,nzr,neltp, &
nelbp,soil_id=1,pile_id=2,raft_id=2,snsoil, &
snpile,snraft,nxlmsh,nylmsh,nr

real(8)::e,v,det,dtim,theta,ttime,loadl,equpval,tol,coef, &
omega,resi,ot1,ot2,it1,it2,tt1,tt2,tim1,tim2,tim3, &
xlen,ylen

logical:: converged
character (len=15):: element = 'hexahedron'
!----- dynamic arrays-----
real(8),allocatable :: prop(:,,:),dee(:,:::), points(:,,:), &
coord(:,,:),jac(:,,:),der(:,,:),deriv(:,,:),weights(:,) &
bee(:,,:),km(:,,:),eld(:),sigma(:), g_coord(:,,:), &
fun(:),widthx(:), widthy(:),depth(:),load_val(:), &
loads(:), ans(:),ebea(:),csrp(:),diagg(:),piv(:), &
tmpvec(:),lnz(:),tmpv(:)

integer,allocatable:: nf(:,,:), g(:), num(:), g_num(:,,:), &
g_g(:,,:),id(:),nodnum(:),iebe(:),jebe(:),etype(:), &
pile_rc(:,,:),iblkord(:),iblkordrev(:),icsrp(:), &
jcsrp(:),adj_row(:),adj(:),perm(:),perm_inv(:), &
xadj2(:), adjncy2(:),colcnt(:),snode(:),xsuper(:), &
iwork(:),xlindx(:), lindx(:),xlnoz(:),split(:)

!----- input and initialization -----
open (10,file='p3dpile.dat',status='old',action='read')
open (11,file='p3dpile.res',status='replace',action='write')
open (12,file='surface.res',status='replace',action='write')
open (13,file='relres.res',status='replace',action='write')

write(*,*)" Iterative Solution Method for pile-group "
write(11,*)" Iterative Solution Method for pile-group "
write(*,*)" The program is running, please wait..... "
call timestamp ( )

read (10,*) nxe,nye,nze,nip,dtim,nstep,theta,maxit,tol

! Input of raft and pile details
read(10,*) nxr,nyr,nzr,npiles,nelbp ; neltp = nzr + 1
! if bottom element of pile < top element
if (nelbp < neltp)then
write(*,*)"There is No pile element"
neltp = 0; nelbp = 0; ! stop
end if
! nxr,nyr,nzr = no. of raft elements
! in x-, y-, and z-directions

```

```

! npiles = the number of piles.
! neltp = no. of top element of pile (in z-direction)
! nelbp = no. of bottom element of pile (in z-direction)
! np_types = the number of material types (zones)
np_types = 3 ! iel = 1-Soil, 2-pile, and 3-raft materials
ndof=nod*nodof; call msh_info(nxe,nye,nze,nels,nn)
write(11,'(a,i16)') ' Number of elements = ', nels
write(11,'(a,i16)') ' Number of nodes    = ', nn
allocate ( prop(2,np_types), etype(nels), &
           dee(nst,nst,np_types), points(nip,ndim), &
           coord(nod,ndim), jac(ndim,ndim), der(ndim,nod), &
           bee(nst,ndof), deriv(ndim,nod), km(ndof,ndof), &
           eld(ndof), sigma(nst), g_g(ndof,nels), fun(nod), &
           nf(ndof,nn), g(ndof), g_coord(ndim,nn), num(nod), &
           weights(nip), g_num(nod,nels), widthx(nxe+1), &
           widthy(nye+1), depth(nze+1), pile_rc(npiles,4) )

read(10,*)(prop(:, i), i=1,np_types)
! prop(1,i) = Effective Young's modulus (E')
! prop(2,i) = Poisson's ratio (v')
read(10,*)(pile_rc(i,:), i = 1, npiles) ;
etype = 1 ! initial set for soil elements
call form_idpile(nxe,nze,npiles,pile_rc,neltp,nelbp,etype, &
                pile_id)
call form_idraft(nxe,nye,nxr,nyr,nzr,etype,raft_id)
read (10,*) widthx, widthy, depth
call nfinfo_drained(nxe,nye,nze,nf,neq)
! do i=1,nn; write(14,*) i, nf(:,i); end do
!-----
do i=1, np_types;
call deemat(dee(:, :, i), prop(1,i), prop(2,i) ) ; end do
call sample(element, points, weights)
allocate( id(1:neq) ) ;
id(:) = 1 ! initial set for soil elements.
! id(:) = 1 --> soil DOFs
! id(:) = 2 --> pile DOFs
! id(:) = 2 --> pile DOFs(pile_id)
! id(:) = 3 --> raft DOFs(raft_id)
! id(:) = 0 --> pore pressure DOFs
!----- loop the elements to set up global arrays-----
!-----
call cpu_time (ttl)
elements_1: do iel = 1, nels
call geometry_20bxz(iel,nxe,nze,widthx,widthy,depth, &
                  coord,num)
inc=0 ;
do i=1,20;
do k=1,3; inc=inc+1; g(inc)=nf(k,num(i)); end do; end do
! -----
do i=1, inc;
if (g(i)/=0) then
if (etype(iel) == pile_id) then ;
id(g(i)) = pile_id ;
elseif (etype(iel) == raft_id) then ;
id(g(i)) = raft_id ;
end if
end if
end do
! -----
g_num(:, iel)=num; g_coord(:, num)=transpose(coord);
g_g(:, iel)= g

```

```

        if(nband<bandwidth(g))nband=bandwidth(g) ;
end do elements_1

write(11,'(a)') "Global coordinates "
do k=1,nn;
    write(11,'(a,i7,a,3e12.4)') "Node",k,"      ",g_coord(:,k);
end do
write(11,'(a)') "Global node numbers "
do k = 1 , nels;
    write(11,'(a,i6,a,20i7)') "Element ",k,"      ",g_num(:,k);
end do
! do k = 1 , nels;
!     write(14,'(a,i6,a,60i7)') "Element ",k,"      ",g_g(:,k);
! end do
write(11,'(2(a,i8))')      &
    "There are ",neq,      &
    " equations and the half-bandwidth is ",nband
!-----
snpile = 0; snraft = 0;
do i = 1,neq;
    if (id(i) == pile_id) then ;
        snpile = snpile + 1;
    else if (id(i) == raft_id) then;
        snraft = snraft + 1;
    end if
end do
snsoil = neq - snpile - snraft
write( *,'(a)') ' '
write( *,'(2(a,i12))') ' Pile DOFs = ',snpile,    &
    ' Raft DOFs = ',snraft
write(11,'(2(a,i12))') ' Pile DOFs = ',snpile,    &
    ' Raft DOFs = ',snraft
write( *,'(2(a,i12))') ' Soil DOFs = ',snsoil,    &
    ' Total DOFs = ',neq ;
write(11,'(2(a,i12))') ' Soil DOFs = ',snsoil,    &
    ' Total DOFs = ',neq ;

allocate( iblkordrev(neq),iblkord(neq) )
call form_3bord(neq,snpile,snraft,pile_id,raft_id,1,id,    &
    iblkordrev,iblkord)
!-----
! xlen,ylen = length and breadh (m) of the loaded area
!(x- and y-direction)
! nxlms,nylms = no. of loaded elements in x & y directions
! equipval = applied load in MPa
read(10,*) xlen,ylen,nxlms,nylms,equipval
! the no. of loaded nodes
loaded_nodes =(nxlms*2+1)*(nylms+1)+(nxlms+1)*nylms
allocate(nodnum(loaded_nodes),load_val(loaded_nodes) )
call load_raft(nxe,nye,nze,nels,nn,xlen,ylen,nxlms,nylms, &
    equipval,loaded_nodes,nodnum,load_val)
!-----
ebeanz = int(ndof*(ndof+1)/2)*nels
write(*,'(a)') ' '
write(*,'(a)') &
    ' Estimated ebeanz for element stiffness integration'
write(*,'(a,i16)') ' and assembly ',ebeanz
write(11,'(a)') ' '
write(11,'(a)') &
    ' Estimated ebeanz for element stiffness integration'
write(11,'(a,i16)') ' and assembly ',ebeanz

```

```

    allocate( iebe(ebeanz), jebe(ebeanz), ebea(ebeanz) )
!----- element stiffness integration and assembly -----
    ebeanz=0 ! used for counting the true number
    call cpu_time (otl) ;
elements_2: do iel = 1 , nels
    num = g_num( : , iel ) ; coord=transpose(g_coord(:,num))
    g = g_g( : , iel ) ;
    km = .0 ;
    gauss_points_1: do i = 1 , nip
        call shape_der(der,points,i); jac = matmul(der,coord)
        det = determinant(jac); call invert(jac);
        deriv = matmul(jac,der); call beemat(bee,deriv);
        km = km + &
            matmul(matmul(transpose(bee),dee(:, :, etype(iel))), bee) &
                *det* weights(i)
    end do gauss_points_1
!---collect nonzero entries from element stiffness matrices---
    ! This fmelspar subroutine is for 2 x 2 block ordering
    call
fmelspar(ndof,g,km,iblkord,iebe,jebe,ebea,ebeanz)
end do elements_2

!-----
write( *, '(a)') ' '
write( *, '(a,i16)') &
    ' Returned true ebeanz after element assembly', ebeanz
write(11, '(a)') ' '
write(11, '(a,i16)') &
    ' Returned true ebeanz after element assembly', ebeanz
call sortadd(ebeanz, jebe(1:ebeanz), iebe(1:ebeanz), &
    ebea(1:ebeanz), neq+1, uanz)
write(11, *) '*****'
write(11, *) '-- The returned true storage for CSC Upper A: --'
write(11, '(a,i9)') ' NNZ of CSC Upper A =', uanz

pneq = snpile + snraft ; gneq = snsoil ;
!-----building the block diagonal preconditioner-----
write( *, '(a)') " Block diagonal preconditioner P3G1"
write(11, '(a)') " Block diagonal preconditioner P3G1"
write( *, '(a)') &
    " MMD-Chol factorization of block(1,1) and Diagonal &
    & of block(2,2)"
write(11, '(a)') &
    " MMD-Chol factorization of block(1,1) and Diagonal &
    & of block(2,2)"
!-----Sparse Cholesky factorization on Blk(1,1)-----
!-----
call cpu_time (tim1)
call countnzblk11 (pneq, neq, jebe(1:neq+1), iebe(1:uanz), &
    ebea(1:uanz), pnz)
allocate (icsrp(pneq+1), jcsrp(pnz), csrp(pnz))

call formblk11 (pneq, neq, jebe(1:neq+1), iebe(1:uanz), &
    ebea(1:uanz), icsrp, jcsrp, csrp, pnz)
write( *, '(a)') ' '
write( *, '(a,i16)') &
    ' Returned ture nnz for block(1,1) i.e. Pile =', pnz
write(11, '(a,i16)') &
    ' Returned ture nnz for block(1,1) i.e. Pile =', pnz
punz = int((pnz+pneq)/2)

```

```

allocate (perm(pneq),perm_inv(pneq),colcnt(pneq),      &
          snode(pneq),xsuper(pneq+1))
write ( *, '(a)' ) ' '
write ( *, '(a)' )      &
'the Multiple Minimal Degree (MMD) algorithm is used for'
write ( *, '(a)' ) 'ordering the coefficient matrixx.'

! No. of adjacency entries excluding diagonal entries
nonzp = pnz-pneq
allocate ( adj_row(pneq+1),adj(nonzp) )
call form_adjacency (pneq,icsrp(1:pneq+1),jcsrp(1:pnz), &
                    adj_row,adj)

iwsiz=7 * pneq + 3
allocate ( xadj2(pneq+1), adjncy2(nonzp),iwork(iwsiz) )

! Save another copy (xadj2,adjncy2) because the
! (xadj,adjncy) structure is destroyed by the minimum
! degree ordering routine ordmmd.
xadj2 = adj_row
adjncy2 = adj
!-----
iwsiz=4 * pneq
call ordmmd (pneq, xadj2, adjncy2, perm_inv, perm, iwsiz, &
            iwork ,nsub,iflag )

if(iflag== -1)then
write( *, '(a)' ) &
'Insufficient Working Storage,IWORK(:), when executing &
& ORDMMD...'
stop
end if
deallocate ( xadj2, adjncy2)
!-----Cholesky factorization

iwsiz=7 * pneq + 3
call sfinit (pneq, nonzp, adj_row, adj, perm, perm_inv, &
            colcnt, nnzl,nsub, nsuper, snode, xsuper, &
            iwsiz ,iwork , iflag )

if(iflag== -1)then
write(*, '(a)' ) 'ERROR: Insufficient Working &
& Storage,IWORK(:),when executing SFINIT '
stop
end if
!-----
allocate( xlindx(nsuper+1),lindx(nsub), xlnz(pneq+1), &
        split(pneq) )
iwsiz = nsuper + 2 * pneq + 1

call symfct (pneq, nonzp, adj_row, adj, perm , perm_inv, &
            colcnt, nsuper,xsuper,snode ,nsub, xlindx, &
            lindx , xlnz , iwsiz ,iwork ,iflag )
if(iflag== -1)then
write(*, '(a)' ) &
'ERROR: Insufficient Working Storage,IWORK(:), &
& when executing SYMFCT... '
stop
elseif(iflag== -2) then
write(*, '(a)' ) &

```

```

'ERROR: Inconsistency in The Input when executing &
& SYMFCT... '
        stop
    end if
    deallocate ( adj_row, adj, colcnt)
!-----
    iwsiz = pneq
!     write(*,'(a)') 'Attention: If program break down &
!                   & here,pls considerto increase-nnzlmax
,

    allocate ( lnz(nnzl) )

write( *,'(a,i9)') 'No. of nonzeros of Cholesky factor &
& of block P, NNZL =',nnzl
write(11,'(a,i9)') 'No. of nonzeros of Cholesky factor &
& of block P, NNZL =',nnzl

call inpnv (pneq, icsrp(1:pneq+1),jcsrp(1:pnz),      &
            csrp(1:pnz), perm,perm_inv, nsuper, xsuper, &
            xlindx, lindx,xlnz, lnz, iwork)

    deallocate ( icsrp,jcsrp, csrp )
!-----
! bfini: Initialization for block factorization
    cachsz = 16
!size of the cache (in kilobytes) on the target
machine
    call bfini(pneq, nsuper, xsuper, snode, xlindx, lindx, &
              cachsz, tmpsiz,split)

! if (tmpsiz > tmpmax) then
!     write(*,'(a)') 'ERROR: tmpsiz > tmpmax when calling &
!                   & symfct; pls increase tmpmax '
!     write(*,'(2(a,i16))') &
!         'tmpsiz=', tmpsiz , '; tmpmax=', tmpmax
!     stop
! endif
!-----
! blkfct: Numerical factorization
    iwsiz = 2 * pneq + 2 * nsuper
    level = 4
! level of loop unrolling while performing numerical
! factorization
    allocate ( tmpvec(tmpsiz) )
    if (level .eq. 1) then
        call blkfct(pneq, nsuper, xsuper, snode, split,      &
                    xlindx, lindx, xlnz,lnz, iwsiz, iwork, &
                    tmpsiz, tmpvec, iflag , mmpy1, smxpy1)
    elseif (level .eq. 2) then
        call blkfct(pneq, nsuper, xsuper, snode, split,      &
                    xlindx, lindx, xlnz, lnz, iwsiz, iwork,&
                    tmpsiz, tmpvec, iflag , mmpy2, smxpy2)
    elseif (level .eq. 4) then
        call blkfct(pneq, nsuper, xsuper, snode, split,      &
                    xlindx, lindx, xlnz,lnz, iwsiz, iwork, &
                    tmpsiz, tmpvec, iflag , mmpy4, smxpy4)
    elseif (level .eq. 8) then
        call blkfct(pneq, nsuper, xsuper, snode, split,      &
                    xlindx, lindx, xlnz, lnz, iwsiz, iwork,&
                    tmpsiz, tmpvec, iflag , mmpy8, smxpy8)

```

```

endif
  if(iflag== -1) then
    write(*, '(a)') 'ERROR: Nonpositive Diagonal &
      & Encountered, when executing BLKFCT... '
    stop
  elseif(iflag== -2) then
    write(*, '(a)') 'Insufficient Working Storage, &
      & TEMP(:), when executing BLKFCT... '
    stop
  elseif(iflag== -3) then
    write(*, '(a)') 'ERROR: Insufficient Working &
      & Storage, IWORK(:), when executing BLKFCT... '
    stop
  end if
  deallocate ( snode, split, iwork, tmpvec )
  !-----
  call cpu_time (tim2)
  !----Diagonal on Blk(2,2)-----

  allocate (diagg(gneq))

  do j = pneq+1, neq
    ir=jebe(j+1)-1 ; diagg(j-pneq) = ebea(ir);
  end do

  diagg(1:gneq) = 1./diagg(1:gneq) ;
  ! inverted diagonals form for preconditioning

  !-----
  call cpu_time (tim3)
  call cpu_time (ot2) ; ot2=ot2-ot1

! ----- enter the time-stepping loop-----
  allocate (loads(0:neq), ans(0:neq) )
  ttime = .0; loads = .0
  time_steps: do ns = 1 , nstep
    write(*, '(a, i5,a)') &
      ' Current Time Step is No.', ns , ' Step. '
    write(11,*) '*****'
    ttime = ttime+dtim;
    write(11, '(a,e12.4)') "The current time is", ttime
    ans=.0;
! ----- Apply Constant Loading -----
    ! the Load is applied at the first step.
    if(ns == 1) then
      do i=1, loaded_nodes
        ans(nf(3,nodnum(i))) = - load_val(i)
      end do
    end if
! Permute the rhs (or load vector) according to block ordering
    call dvperm(neq, ans(1:), iblkord)
!----- Preconditioned Iterative Solver -----
    call cpu_time (it1)

    call sbd1pcg(neq, jebe(1:neq+1), iebe(1:uanz), ebea(1:uanz), &
      diagg, nsuper, xsuper, xlindx, lindx, xlnz, lnz, &
      perm, perm_inv, ans(1:), maxit, tol, iters, resi) ;
    ans(0) = 0.0

    call cpu_time (it2) ; it2=it2-it1

```

```

!   obtain the original(natural) solution from block ordering
call perm_rv ( neq, ans(1:), iblkordrev )
loads(1:neq) = loads(1:neq) + ans(1:neq)

write(11,'(a)') " The nodal displacements are      :"
do k=1,5; write(11,'(i7,a,3e13.5)')      &
    k,"      ",loads(nf(:,k)) ; end do
do k=1,nn; write(25,'(i7,a,3e13.5)')      &
    k,"      ",loads(nf(:,k)) ; end do
! -----
! nodplane = (2*nx+1)*(nz+1)+(nx+1)*nz;
! node number for each x-z plane
! nodbetwplane = (nx+1)*(nz+1) ;
! node number between two x-z planes

! To get the surface settlement of the footing
do i=1, nye+1
    do j = 1, nxe+1
        k = ((2*nxe+1)*(nze+1)+(nx+1)*nze+(nx+1)*(nze+1)) &
            *(i-1)+2*j -1
        write(12,'(2f10.2,e20.8)') widthx(j),widthy(i),      &
            loads(nf(3,k));
    end do
end do

!-----recover stresses at Gauss-points-----
elements_5 : do iel = 1 , nels
    num = g_num(:,iel); coord=transpose(g_coord(:,num))
    g = g_g( : , iel ); eld = loads( g ( 1 : ndof ) )
    ! print*,"The Gauss Point effective stresses for &
    !      & element",iel,"are"
    gauss_points_2: do i = 1,nip
        call shape_der (der,points,i); jac= matmul(der,coord)
        call invert ( jac ); deriv= matmul(jac,der)
        bee= 0.;call beemat(bee,deriv);
        sigma= matmul(dee(:, :, etype(iel)),matmul(bee,eld))
        ! print*,"Point      ",i      ;! print*,sigma
    end do gauss_points_2
end do elements_5
end do time_steps
call cpu_time (tt2) ; tt2=tt2-tt1
write(11,'(a,f10.2)')      &
"      Operation time on blocks(1,1) is: ",tim2-tim1
write(11,'(a,f10.2)')      &
"      Operation time on blocks(2,2) is: ",tim3-tim2
write(11,'(a,f10.2)')      &
"      Overhead time is: ",ot2
write(11,'(a,f10.2)')      &
"      Iterative time (the last time step) is: ",it2
write(11,'(a,f10.2,a)')      &
"      Total runtime for the FEM program is      ",tt2,"seconds."
end program sbd1pcgmain
!-----

```

Input file 'p3dpile.dat' of a small example in 7×7×7 mesh

7 7 7 27 1. 1 1. 10000 1.e-6
5 5 2 4 5

```

5.0 0.3
205000.0 0.2
205000.0 0.2
1 1 1 1
4 5 1 1
1 1 4 5
4 5 4 5
.0 1.0 2.0 3.0 4.0 5.0 7.5 10.0
.0 1.0 2.0 3.0 4.0 5.0 7.5 10.0
.0 -1.0 -2.0 -3.0 -4.0 -5.0 -7.5 -10.0
5.0 5.0 5 5 0.1

```

## E.2. Main program for SBD<sub>2</sub>-PCG

```

!-----
program sbd2pcgmain
!-----
! 3D Soil-structure analysis with drained parameters
!-----
!   Program for 3-D soil-structure analysis using 20-node
!   solid brick elements
!   PCG solver is used for solving the linear system Ax = b
!   in each time step
!   A = [P L ; L' G] ==> P : Pile Block, G = Geo Block,
!   and L = Link matrix
!   Block diagonal preconditioner
!           M(-1) = [ P(-1)      0;
!                   0      ssor(G)-1 ]
!-----

use new_library ; use geometry_lib;  use sparselib_v3;
use dfport ;  use spchol ;

implicit none
integer:: i,j,k,l,nn,nels,nxe,nye,nze,nip,nodof=3,nod=20, &
nst=6,ndim=3,ndof,iel,ns,nstep,inc,loaded_nodes, &
neq,nband,nmesh,ebeanz,maxit,itors,isolver,icho, &
ipre,icc,iinc,uanz,np_types,npiles,nels_pile, &
gneq,pneq,ir,k1,k2,pnz,gnz,llnz,anz,nonzp,ierr, &
punz,gunz,iwsiz,nsup,iflag,nnzl,nsuper,tmpsiz, &
cachsz,level,nxr,nyr,nzr,neltp,nelbp,soil_id=1, &
pile_id=2,raft_id=2,snsoil,snpile,snraft,nxlmsh, &
nylmsh,nr

real(8):: e,v,det,dtim,theta,ttime,loadl,equipval,tol,coef, &
omega,resi,ot1,ot2,it1,it2,tt1,tt2,tim1,tim2,tim3, &
xlen,ylen

logical:: converged
character (len=15):: element = 'hexahedron'
!----- dynamic arrays-----
real(8),allocatable :: prop(:,,:),dee(:,,:), points(:,,:), &
coord(:,,:),jac(:,,:),der(:,,:),deriv(:,,:),weights(:,), &
bee(:,,:),km(:,,:),eld(:,,:),sigma(:,), g_coord(:,,:), &
fun(:,),widthx(:,), widthy(:,),depth(:,),load_val(:,), &
loads(:,), ans(:,),ebea(:,),csrp(:,),csrg(:,),csrug(:,), &
diagg(:,),piv(:,),tmpvec(:,),lnz(:,),tmpv(:,)
! cscs(:,),
integer,allocatable:: nf(:,,:), g(:,), num(:,), g_num(:,,:), &

```

```

        g_g(:, :), id(:, :), nodnum(:, :), iebe(:, :), jebe(:, :), etype(:, :), &
        pile_rc(:, :), iblkord(:, :), iblkordrev(:, :), icsrp(:, :), &
        jcsrp(:, :), icsrg(:, :), jcsrg(:, :), adj_row(:, :), adj(:, :), perm(:, :), &
        perm_inv(:, :), icsru(:, :), jcsru(:, :), xadj2(:, :), adjncy2(:, :), &
        colcnt(:, :), snode(:, :), xsuper(:, :), iwork(:, :), xlindx(:, :), &
        lindx(:, :), xlnz(:, :), split(:)
        !      icsca(:, :), jcsca(:, :),
!----- input and initialization -----
open (10, file='p3dpile.dat', status='old', action='read')
open (11, file='p3dpile.res', status='replace', action='write')
open (12, file='surface.res', status='replace', action='write')
open (13, file='relres.res', status='replace', action='write')

write( *, *) " Iterative Solution Method for pile-group "
write(11, *) " Iterative Solution Method for pile-group "
write( *, *) " The program is running, please wait..... "
call timestamp ( )

read (10, *) nxe, nye, nze, nip, dtim, nstep, theta, maxit, tol
! Input of raft and pile details
read(10, *) nxr, nyr, nzt, npiles, neltp; neltp = nzt + 1
! if bottom element of pile < top element
if (neltp < neltp) then
    write(*, *) 'There is No pile element'
    neltp = 0; neltp = 0; ! stop
end if
! nxr, nyr, nzt = no. of raft elements in x-, y-,
! and z-directions
! npiles = the number of piles.
! neltp = no. of top element of pile (in z-direction)
! neltp = no. of bottom element of pile (in z-direction)

! np_types = the number of material types (zones)
np_types = 3 ! iel = 1-Soil, 2-pile, and 3-raft materials
ndof=nod*nodof; call msh_info(nxe, nye, nze, nels, nn)
write(11, '(a,i16)') ' Number of elements = ', nels
write(11, '(a,i16)') ' Number of nodes = ', nn
allocate ( prop(2, np_types), etype(nels), &
        dee(nst, nst, np_types), points(nip, ndim), &
        coord(nod, ndim), jac(ndim, ndim), der(ndim, nod), &
        bee(nst, ndof), deriv(ndim, nod), km(ndof, ndof), &
        eld(ndof), sigma(nst), g_g(ndof, nels), fun(nod), &
        nf(ndof, nn), g(ndof), g_coord(ndim, nn), num(nod), &
        weights(nip), g_num(nod, nels), widthx(nxe+1), &
        widthy(nye+1), depth(nze+1), pile_rc(npiles, 4) )

read(10, *) (prop(:, i), i=1, np_types)
! prop(1,i) = Effective Young's modulus (E')
! prop(2,i) = Poisson's ratio (v')
read(10, *) (pile_rc(i, :), i = 1, npiles) ;
etype = 1 ! initial set for soil elements
call form_idpile(nxe, nze, npiles, pile_rc, neltp, neltp, etype, &
        pile_id)
call form_idraft(nxe, nye, nxr, nyr, nzt, etype, raft_id)
read (10, *) widthx, widthy, depth
call nfinfo_drained(nxe, nye, nze, nf, neq)
! do i=1, nn; write(14, *) i, nf(:, i); end do
!-----
do i=1, np_types;
    call deemat(dee(:, :, i), prop(1, i), prop(2, i) ) ; end do
call sample(element, points, weights)

```

```

allocate( id(1:neq) ) ;
id(:) = 1      ! initial set for soil elements.
! id(:) = 1 --> soil DOFs
! id(:) = 2 --> pile DOFs
! id(:) = 2      --> pile DOFs(pile_id)
! id(:) = 3      --> raft DOFs(raft_id)
! id(:) = 0      --> pore pressure DOFs
!----- loop the elements to set up global arrays-----
      call cpu_time (ttl)
elements_1: do iel = 1, nels
      call geometry_20bxz(iel,nxe,nze,widthx,widthy,depth,  &
      coord,num)
      inc=0 ;
      do i=1,20;
          do k=1,3; inc=inc+1;g(inc)=nf(k,num(i));
          end do;
      end do
      ! -----
      do i=1, inc;
          if (g(i)/=0) then
              if (etype(iel) == pile_id) then ;
                  id(g(i)) = pile_id ;
              elseif (etype(iel) == raft_id) then ;
                  id(g(i)) = raft_id ;
              end if
          end if
      end do
      ! -----
      g_num(:,iel)=num;g_coord(:,num)=transpose(coord);
      g_g(:,iel)= g
          if(nband<bandwidth(g))nband=bandwidth(g) ;
end do elements_1

write(11,'(a)') "Global coordinates "
do k=1,nn;
    write(11,'(a,i7,a,3e12.4)') "Node",k,"      ",g_coord(:,k);
end do
write(11,'(a)') "Global node numbers "
do k = 1 , nels;
    write(11,'(a,i6,a,20i7)') "Element ",k,"      ",g_num(:,k);
end do
write(11,'(2(a,i8))')          &
    "There are ",neq,          &
    " equations and the half-bandwidth is      ",nband
!-----
snpile = 0; snraft = 0;
do i = 1,neq;
    if (id(i) == pile_id) then ;
        snpile = snpile + 1;
    else if (id(i) == raft_id) then;
        snraft = snraft + 1;
    end if
end do
snsoil = neq - snpile - snraft
write( *,'(a)') ' '
write( *,'(2(a,i12))') ' Pile DOFs = ',snpile,    &
    ' Raft DOFs = ',snraft
write(11,'(2(a,i12))') ' Pile DOFs = ',snpile,    &
    ' Raft DOFs = ',snraft
write( *,'(2(a,i12))') ' Soil DOFs = ',snsoil,    &
    ' Total DOFs = ',neq ;

```

```

write(11,'(2(a,i12))') ' Soil DOFs = ',snsoil,      &
    ' Total DOFs = ',neq ;

allocate( iblkordrev(neq),iblkord(neq) )
call form_3bord(neq,snpile,snraft,pile_id,raft_id,1,id,      &
    iblkordrev,iblkord)
!-----
! xlen,ylen = length and breadth (m) of the loaded area
!           (x- and y-direction)
! nxlms,nylms = no. of loaded elements in
!           x- and y-direction
! equipval = applied load in MPa
read(10,*) xlen,ylen,nxlms,nylms,equipval
! the no. of loaded nodes
loaded_nodes =(nxlms*2+1)*(nylms+1)+(nxlms+1)*nylms
allocate(nodnum(loaded_nodes),load_val(loaded_nodes) )
call load_raft(nxe,nye,nze,nels,nn,xlen,ylen,nxlms,nylms, &
    equipval,loaded_nodes,nodnum,load_val)
!-----
    ebeanz = int(ndof*(ndof+1)/2)*nels
write(*,'(a)') ' '
write(*,'(a)')      &
    ' Estimated ebeanz for element stiffness integration'
write(*,'(a,i16)') ' and assembly ',ebeanz
write(11,'(a)') ' '
write(11,'(a)')      &
    ' Estimated ebeanz for element stiffness integration'
write(11,'(a,i16)') ' and assembly ',ebeanz

allocate( iebe(ebeanz),jebe(ebeanz), ebea(ebeanz) )
!----- element stiffness integration and assembly ----
    ebeanz=0 ! used for counting the true number
    call cpu_time (ot1) ;
elements_2: do iel = 1 , nels
    num = g_num( : , iel ); coord=transpose(g_coord(:,num))
    g = g_g( : , iel ) ;
    km = .0 ;
    gauss_points_1: do i = 1 , nip
        call shape_der(der,points,i); jac = matmul(der,coord)
        det = determinant(jac ); call invert(jac);
        deriv = matmul(jac,der); call beemat(bee,deriv);
        km = km +      &
            matmul(matmul(transpose(bee),dee(:, :, etype(iel))),bee) &
                *det* weights(i)
    end do gauss_points_1
!---collect nonzero entries from element stiffness matrices---
    ! This fmelspar subroutine is for 2 x 2 block ordering
    call
fmelspar(ndof,g,km,iblkord,iebe,jebe,ebea,ebeanz)
end do elements_2

!-----
write( *,'(a)') ' '
write( *,'(a,i16)')      &
    ' Returned true ebeanz after element assembly',ebeanz
write(11,'(a)') ' '
write(11,'(a,i16)')      &
    ' Returned true ebeanz after element assembly',ebeanz

call sortadd(ebeanz,jebe(1:ebeanz),iebe(1:ebeanz),      &
    ebea(1:ebeanz),neq+1,uanz)

```

```

! allocate (jcscs(neq+1),icscs(uanz),cscs(uanz))
! jcscs = jebe(1:neq+1) ; icscs = iebe (1:uanz) ;
! cscs = ebea(1:uanz) ;
! deallocate (jebe,iebe,ebea)
write(11,*) '*****'
write(11,*) '-The returned true storage for CSC Upper A: -- '
write(11,'(a,i9)') ' NNZ of CSC Upper A =', uanz

pneq = snpile + snraft ;   gneq = snsoil ;

!-----building the block diagonal preconditioner-----
write( *,'(a)') " Block diagonal preconditioner P3G1"
write(11,'(a)') " Block diagonal preconditioner P3G1"
write( *,'(a)') " MMD-Chol factorization of block(1,1)    &
                  & and Diagonal of block(2,2)"
write(11,'(a)') " MMD-Chol factorization of block(1,1)    &
                  & and Diagonal of block(2,2)"
!----Sparse Cholesky factorization on Blk(1,1)-----
call cpu_time (tim1)
call countnzblk11 (pneq,neq,jebe(1:neq+1),iebe(1:uanz),      &
                  ebea(1:uanz),pnz)
allocate (icsrp(pneq+1),jcsrp(pnz),csrp(pnz))
call formblk11 (pneq,neq,jebe(1:neq+1),iebe(1:uanz),          &
               ebea(1:uanz),icsrp,jcsrp,csrp,pnz)
write( *,'(a)') ' '
write( *,'(a,i16)') &
' Returned ture nnz for block(1,1) i.e. Pile =', pnz
write(11,'(a,i16)') &
' Returned ture nnz for block(1,1) i.e. Pile =', pnz
punz = int((pnz+pneq)/2)

allocate (perm(pneq),perm_inv(pneq),colcnt(pneq),              &
          snode(pneq),xsuper(pneq+1))
write ( *, '(a)') ' '
write ( *, '(a)') &
'the Multiple Minimal Degree (MMD) algorithm is used for'
write ( *, '(a)') 'ordering the coefficient matrix.'

! No. of adjacency entries excluding diagonal entries
nonzp = pnz-pneq
allocate ( adj_row(pneq+1),adj(nonzp) )
call form_adjacency (pneq,icsrp(1:pneq+1),jcsrp(1:pnz),      &
                    adj_row,adj)

iwsiz=7 * pneq + 3
allocate ( xadj2(pneq+1), adjncy2(nonzp),iwork(iwsiz) )

! Save another copy (xadj2,adjncy2) because the
!(xadj,adjncy) structure is destroyed by the minimum
! degree ordering routine ordmmd.
xadj2 = adj_row
adjncy2 = adj
!-----
iwsiz=4 * pneq
call ordmmd (pneq, xadj2, adjncy2, perm_inv, perm, iwsiz, &
            iwork ,nsub,iflag )

if(iflag== -1)then
write( *,'(a)') 'Insufficient Working Storage, &
& IWORK(:), when executing ORDMMD...'
stop

```

```

    end if
    deallocate ( xadj2, adjncy2)
!-----Cholesky factorization

    iwsiz=7 * pneq + 3
    call sfininit (pneq, nonzp, adj_row, adj, perm, perm_inv, &
                  colcnt, nnzl, nsub, nsuper, snode, xsuper, &
                  iwsiz, iwork, iflag )

    if(iflag==1)then
        write(*,'(a)') 'ERROR: Insufficient Working &
                        & Storage,IWORK(:), when executing SFINIT '
        stop
    end if
!-----
    allocate( xlindx(nsuper+1),lindx(nsub), xlnz(pneq+1), &
             split(pneq))
    iwsiz = nsuper + 2 * pneq + 1

    call symfct (pneq, nonzp, adj_row, adj, perm, perm_inv, &
                colcnt, nsuper,xsuper,snode, nsub,xlindx, &
                lindx, xlnz, iwsiz, iwork, iflag )

    if(iflag==1)then
        write(*,'(a)') 'ERROR: Insufficient Working &
                        & Storage,IWORK(:), when executing SYMFCT... '
        stop
    elseif(iflag==2) then
        write(*,'(a)') 'ERROR: Inconsistency in The Input &
                        & when executing SYMFCT... '
        stop
    end if
    deallocate ( adj_row, adj, colcnt)
!-----
    iwsiz = pneq
    allocate (lnz(nnzl))
    call inpnv (pneq, icsrp(1:pneq+1),jcsrp(1:pnz), &
               csrp(1:pnz), perm,perm_inv, nsuper, xsuper, &
               xlindx, lindx,xlnz, lnz, iwork)

    deallocate ( icsrp,jcsrp, csrp )
!-----
! bfininit: Initialization for block factorization
! size of the cache (in kilobytes) on the target machine
    cachsz = 16
    call bfininit(pneq, nsuper, xsuper, snode, xlindx, lindx, &
                 cachsz, tmpsiz,split)

! blkfct: Numerical factorization
    iwsiz = 2 * pneq + 2 * nsuper
    level = 4
! level of loop unrolling while performing numerical
! factorization
    allocate ( tmpvec(tmpsiz) )
    if (level .eq. 1) then
        call blkfct(pneq, nsuper, xsuper, snode, split, &
                   xlindx, lindx, xlnz,lnz, iwsiz, iwork, &
                   tmpsiz, tmpvec, iflag, mmpyl, smxpyl)
    elseif (level .eq. 2) then
        call blkfct(pneq, nsuper, xsuper, snode, split, &
                   xlindx, lindx, xlnz,lnz, iwsiz, iwork, &

```

```

        tmpsiz, tmpvec, iflag , mmpy2, smxpy2)
elseif (level .eq. 4) then
    call blkfct(pneq, nsuper, xsuper, snode, split,      &
               xlindx, lindx, xlnz,lnz, iwsiz, iwork, &
               tmpsiz, tmpvec, iflag , mmpy4, smxpy4)
elseif (level .eq. 8) then
    call blkfct(pneq, nsuper, xsuper, snode, split,      &
               xlindx, lindx, xlnz,lnz, iwsiz, iwork, &
               tmpsiz, tmpvec, iflag , mmpy8, smxpy8)
endif
if(iflag== -1) then
    write(*,'(a)') 'ERROR: Nonpositive Diagonal      &
    & Encountered, when executing BLKFCT... '
    stop
elseif(iflag== -2) then
    write(*,'(a)') 'Insufficient Working Storage,    &
    & TEMP(:), when executing BLKFCT... '
    stop
elseif(iflag== -3) then
    write(*,'(a)') 'ERROR: Insufficient Working      &
    & Storage,IWORK(:), when executing BLKFCT... '
    stop
end if
    deallocate ( snode, split, iwork, tmpvec )
!-----
    call cpu_time (tim2)
!-----Diagonal on Blk(2,2)-----

allocate (diagg(gneq))

do j = pneq+1,neq
    ir=jebe(j+1)-1 ; diagg(j-pneq) = ebea(ir);
end do

! inverted diagonals form for preconditioning
diagg(1:gneq) = 1./diagg(1:gneq) ;
!-----
call cpu_time (tim3)
call cpu_time (ot2) ; ot2=ot2-ot1

! ----- enter the time-stepping loop-----
allocate(loads(0:neq), ans(0:neq) )
    ttime = .0; loads = .0
time_steps: do ns = 1 , nstep
    write(*, '(a, i5,a)') &
        ' Current Time Step is No.', ns , ' Step. '
    write(11,*) '*****'
    ttime = ttime+dtim;
    write(11,'(a,e12.4)') "The current time is",ttime
    ans=.0;
! ----- Apply Constant Loading -----
        ! the Load is applied at the first step.
        if(ns == 1) then
            do i=1,loaded_nodes
                ans(nf(3,nodnum(i))) = - load_val(i)
            end do
        end if
! Permute the rhs (or load vector) according to block ordering
        call dvperm(neq,ans(1:),iblkord)
!----- Preconditioned Iterative Solver -----
        call cpu_time (it1)

```

```

call sbd2pcg(neq, jebe(1:neq+1), iebe(1:uanz), ebea(1:uanz), &
            diag, nsuper, xsuper, xlindx, lindx, xlnz, lnz, &
            perm, perm_inv, ans(1:), maxit, tol, iters, resi);
ans(0) = 0.0

call cpu_time (it2) ; it2=it2-it1
! obtain the original(natural) solution from block ordering
call perm_rv ( neq, ans(1:), iblkordrev )
loads(1:neq) = loads(1:neq) + ans(1:neq)

write(11, '(a)') " The nodal displacements are      : "
do k=1,5;
write(11, '(i7,a,3e13.5)')k, "      ", loads(nf(:,k)) ; end do
do k=1,nn;
write(25, '(i7,a,3e13.5)')k, "      ", loads(nf(:,k)) ; end do
! -----
! node number for each x-z plane
! nodplane = (2*nx+1)*(nz+1)+(nx+1)*nz;
! node number between two x-z planes
! nodbetwplane = (nx+1)*(nz+1) ;

! To get the surface settlement of the footing
do i=1, nye+1
do j = 1, nxe+1
k = ((2*nxe+1)*(nze+1)+(nxe+1)*nze+(nxe+1)*(nze+1)) &
    *(i-1)+2*j -1
write(12, '(2f10.2,e20.8)') widthx(j), widthy(i), &
    loads(nf(3,k));
end do
end do

!-----recover stresses at Gauss-points-----
elements_5 : do iel = 1 , nels
num = g_num(:,iel); coord=transpose(g_coord(:,num))
g = g_g( : , iel ); eld = loads( g ( 1 : ndof ) )
! print*, "The Gauss Point effective stresses for element", &
! iel, "are"
gauss_points_2: do i = 1, nip
call shape_der (der, points, i); jac= matmul(der, coord)
call invert ( jac ); deriv= matmul(jac, der)
bee= 0.; call beemat(bee, deriv);
sigma= matmul(dee(:, :, etype(iel)), matmul(bee, eld))
! print*, "Point ", i ;! print*, sigma
end do gauss_points_2
end do elements_5
end do time_steps
call cpu_time (tt2) ; tt2=tt2-tt1
write(11, '(a,f10.2)') &
"      Operation time on blocks(1,1) is: ", tim2-tim1
write(11, '(a,f10.2)') &
"      Operation time on blocks(2,2) is: ", tim3-tim2
write(11, '(a,f10.2)') &
"      Overhead time is: ", ot2
write(11, '(a,f10.2)') &
"      Iterative time (the last time step) is: ", it2
write(11, '(a,f10.2,a)') &
"      Total runtime for the FEM program is ", tt2, &
"seconds."

end program sbd2pcgmain
!-----

```

The input file is the same as for the SBD<sub>1</sub>-PCG.

### E.3. Main program for $M_1$ -SQMR

```
!-----
program p3dbiot
!-----
! 3D Biot Consolidation Analysis of Soil-structure
! interaction problem
!-----
!   Program for 3-D consolidation analysis using 20-node solid
!   brick elements coupled to 8-node fluid elements,
!   incremental formulation. SQMR solver is used for solving
!   the linear system in each time step, M1 preconditioner
!   is available.
!
!   A = [Pile block           = [ P       L       B' ;
!         Soil Block          =  L^T      G       B  ;
!         Fluid block]        =  B'^T    B^T    -C  ]
!-----

use new_library ; use geometry_lib; use sparselib_v3;
use dfport ; use spchol ;

implicit none
integer:: i,j,k,l,nn,nels,nxe,nye,nze,nip,nodof=4,nod=20, &
nodf=8,nst=6,ndim=3,nodofs=3,ntot,ndof,iel,ns, &
nstep,inc,loaded_nodes,neq,nband,sneq,fneq,nmesh, &
ebeanz, maxit, iters, isolver, icho, ipre, icc, iinc, &
uanz, np_types, npiles, nels_pile, gneq, pneq, ir, kl, &
k2, pnz, gnz, llnz, anz, nonzp, ierr, punz, gunz, iwsiz, &
nsub, iflag, nnzl, nsuper, nnzlmax, tmpsiz, tmpmax, &
cachsz, level, nxr, nyr, nzt, neltp, nelbp, soil_id=1, &
pile_id=2, raft_id=2, snsoil, snpile, snraft, nr, &
nxlmsh, nylmsh

real(8):: permx,permy,permz,e,v,det,dtim,theta,ttime,loadl, &
equipval,tol,coef,omega,resi,ot1,ot2,it1,it2,ttl, &
tt2,tim1,tim2,tim3,xlen,ylen

logical:: converged
character (len=15):: element = 'hexahedron'
!----- dynamic arrays-----
real(8), allocatable :: prop(:,,:), dee(:,,:), points(:,,:), &
coord(:,,:), derivf(:,,:), jac(:,,:), kay(:,,:), der(:,,:), &
deriv(:,,:), weights(:,,:), derf(:,,:), funf(:,,:), coordf(:,,:), &
bee(:,,:), km(:,,:), eld(:,,:), sigma(:,,:), kp(:,,:), ke(:,,:), &
g_coord(:,,:), kd(:,,:), fun(:,,:), c(:,,:), bk(:,,:), vol(:,,:), &
volf(:,,:), widthx(:,,:), widthy(:,,:), depth(:,,:), load_val(:,,:), &
loads(:,,:), ans(:,,:), ebea(:,,:), tmpv(:,,:), diag(:,,:), &
diagal(:,,:), csrp(:,,:), tmpvec(:,,:), lnz(:,)

integer, allocatable:: nf(:,,:), g(:,,:), num(:,,:), g_num(:,,:), &
g_g(:,,:), nodnum(:,,:), iebe(:,,:), jebe(:,,:), id(:,,:), etype(:,,:), &
pile_rc(:,,:), iblkordrev(:,,:), iblkord(:,,:), adj_row(:,,:), &
adj(:,,:), perm(:,,:), perm_inv(:,,:), icsrp(:,,:), jcsrp(:,,:), &
xadj2(:,,:), adjncy2(:,,:), colcnt(:,,:), snode(:,,:), xsuper(:,,:), &
iwork(:,,:), xlindx(:,,:), lindx(:,,:), xlnz(:,,:), split(:,)

!----- input and initialization -----
open (10, file='p3dbiot.dat', status='old', action='read')
```

```

open (11,file='p3dbiot.res',status='replace',action='write')
open (12,file='surface.res',status='replace',action='write')
open (13,file='relres.res',status='replace',action='write')
  call timestamp ( )
  print *,      " Iterative Solution Method for pile-group "
  write(11,*)" Iterative Solution Method for pile-group "
  print *,      " The program is running, please wait..... "

read (10,*) nxe,nye,nze,nip,dtim,nstep,theta,maxit,tol,      &
          coef,omega,isolver,icho,ipre,icc,iinc

! Input of raft and pile details
! nxr,nyr,nzr = no. of raft elements in x-, y-, and
! z-directions
! npiles = the number of piles.
! neltp = no. of top element of pile (in z-direction)
! nelbp = no. of bottom element of pile (in z-direction)
read(10,*) nxr,nyr,nzr,npiles,nelbp ; neltp = nzr + 1
! if bottom element of pile < top element
if (nelbp < neltp)then
  write(*,*)'There is No pile element'
  neltp = 0; nelbp = 0; ! stop
end if

! np_types = the number of material types (zones)
np_types = 3 ! iel = 1-Soil, 2-pile, and 3-raft materials
ndof=nod*3; ntot=ndof+nodf;
call msh_info(nxe,nye,nze,nels,nn) !,nr)
write(11,'(a,i16)')' Number of elements = ',nels
write(11,'(a,i16)')' Number of nodes   = ',nn
allocate(prop(5,np_types),etype(nels),      &
          dee(nst,nst,np_types),points(nip,ndim),      &
          coord(nod,ndim),jac(ndim,ndim),derivf(ndim,nodf),      &
          kay(ndim,ndim),der(ndim,nod),deriv(ndim,nod),      &
          derf(ndim,nodf),funf(nodf),coordf(nodf,ndim),      &
          bee(nst,ndof),km(ndof,ndof),eld(ndof),sigma(nst),      &
          kp(nodf,nodf),ke(ntot,ntot),g_g(ntot,nels),fun(nod),      &
          c(ndof,nodf),vol(ndof),nf(nodof,nn),g(ntot),      &
          volf(ndof,nodf),g_coord(ndim,nn),num(nod),weights(nip),      &
          g_num(nod,nels),widthx(nxe+1),widthy(nye+1),      &
          depth(nze+1),pile_rc(npiles,4) )
!
! kay=0.0; kay(1,1)=permx; kay(2,2)=permy; kay(3,3)=permz
read(10,*)(prop(:, i), i=1,np_types)
! np_types material properties
! prop(4,i) for Effective Young's modulus E' ;
! prop(5,i) for Poisson's ratio ;

read(10,*)(pile_rc(i,:), i = 1, npiles) ;
etype = 1 ! initial set for soil elements
call form_idpile(nxe,nze,npiles,pile_rc,neltp,nelbp,etype, &
  pile_id)
call form_idraft(nxe,nye,nxr,nyr,nzr,etype,raft_id)
read (10,*) widthx, widthy, depth
call nfinfo(nxe,nye,nze,nf,neq,sneq,fneq)
!-----
do i=1, np_types;
  call deemat(dee(:, :, i),prop(4,i),prop(5,i) ) ; end do
call sample(element,points,weights)
allocate( id(1:neq) ) ;
id(:) = 1 ! initial set for soil elements.
! id(:) = 1 --> soil DOFs

```

```

! id(:) = 2      --> pile DOFs(pile_id)
! id(:) = 3      --> raft DOFs(raft_id)
! id(:) = 0      --> pore pressure DOFs
!----- loop the elements to set up global arrays-----
    call cpu_time (ttl)
elements_1: do iel = 1, nels
    call geometry_20bxz(iel,nxe,nze,widthx,widthy,depth, &
        coord,num)
    inc=0 ;
    do i=1,20;
    do k=1,3; inc=inc+1;g(inc)=nf(k,num(i));end do;end do
    ! -----
        do i=1, inc;
            if (g(i)/=0) then
                if (etype(iel) == pile_id) then ;
                    id(g(i)) = pile_id ;
                elseif (etype(iel) == raft_id) then ;
                    id(g(i)) = raft_id ;
                end if
            end if
        end do

    ! -----
    do i=1,7,2; inc=inc+1;g(inc)=nf(4,num(i)); end do
    do i=13,19,2; inc=inc+1;g(inc)=nf(4,num(i)); end do
    ! -----
    do i=61, inc; if(g(i)/=0) id(g(i)) = 0 ; end do
    ! -----
    g_num(:,iel)=num;g_coord(:,num)=transpose(coord);
    g_g(:,iel)= g
        if(nband<bandwidth(g))nband=bandwidth(g) ;
end do elements_1
write(11,'(a)') "Global coordinates "
do k=1,nn;
    write(11,'(a,i7,a,3e12.4)') "Node",k,"      ",g_coord(:,k);
end do
write(11,'(a)') "Global node numbers "
do k = 1 , nels;
    write(11,'(a,i6,a,20i7)') "Element ",k,"      ",g_num(:,k);
end do
write(11,'(2(a,i8))')      &
    "There are ",neq,      &
    " equations and the half-bandwidth is      ",nband
snpile = 0; snraft = 0;
do i = 1,neq;
    if (id(i) == pile_id) then ;
        snpile = snpile + 1;
    else if (id(i) == raft_id) then;
        snraft = snraft + 1;
    end if
end do
snsoil = sneq - snpile - snraft
write( *,'(a)') ' '
write( *,'(2(a,i12))') ' Pile DOFs = ',snpile,      &
    ' Raft DOFs      = ',snraft
write(11,'(2(a,i12))') ' Pile DOFs = ',snpile,      &
    ' Raft DOFs      = ',snraft
write( *,'(2(a,i12))') ' Soil DOFs = ',snsoil,      &
    ' Pore Press. DOFs = ',fneq
write(11,'(2(a,i12))') ' Soil DOFs = ',snsoil,      &
    ' Pore Press. DOFs = ',fneq
write( *,'(a,i12)')      ' Total DOFs = ',neq ;

```

```

write(11,'(a,i12)')      ' Total DOFs = ',neq

write ( *,'(a)') ' '
write ( *,'(a)') ' Block ordering of the coefficient matrix '
write (11,'(a)') ' Block ordering of the coefficient matrix '
allocate( iblkordrev(neq),iblkord(neq) )
call form_4bord(neq,snpile,snraft,snsoil,pile_id,raft_id,    &
               1,0,id,iblkordrev,iblkord)
!-----
! xlen,ylen = length and breadth (m) of the loaded area
! (x- and y-direction)
! nxlms,nylms = no. of loaded elements in x- & y- direction
! equipval = applied load in MPa

read(10,*) xlen,ylen,nylms,nylms,equipval
! the no. of loaded nodes
loaded_nodes =(nylms*2+1)*(nylms+1)+(nylms+1)*nylms
allocate(nodnum(loaded_nodes),load_val(loaded_nodes) )
call load_raft(nxe,nye,nze,nels,nn,xlen,ylen,nylms,nylms, &
               equipval,loaded_nodes,nodnum,load_val)
!-----
ebeanz = int(ntot*(ntot+1)/2)*nels
write(*,'(a)') ' '
write(*,'(a)')      &
' Estimated ebeanz for element stiffness integration'
write(*,'(a,i16)') ' and assembly ',ebeanz
write(11,'(a)') ' '
write(11,'(a)')      &
' Estimated ebeanz for element stiffness integration'
write(11,'(a,i16)') ' and assembly ',ebeanz

allocate( iebe(ebeanz),jebe(ebeanz), ebea(ebeanz) )
!----- element stiffness integration and assembly ----
ebeanz=0 ! used for counting the true number
call cpu_time (ot1) ; kay =.0 ;
elements_2: do iel = 1 , nels
  num = g_num( : , iel ); coord=transpose(g_coord(:,num))
  g = g_g( : , iel ) ;
  coordf(1 : 4 , : ) = coord(1 : 7 : 2, : )
  coordf(5 : 8 , : ) = coord(13 : 19 : 2, : )
  km = .0 ; c = .0 ; kp = .0
  !-----forming Kay for each element -----
  ---
  kay(1,1) = prop(1,etype(iel));
  kay(2,2) = prop(2,etype(iel));
  kay(3,3) = prop(3,etype(iel));
  gauss_points_1: do i = 1 , nip
    call shape_der(der,points,i); jac = matmul(der,coord)
    det = determinant(jac ); call invert(jac);
    deriv = matmul(jac,der); call beemat(bee,deriv);
    vol(:)=bee(1,:)+bee(2,:)+bee(3,:);
    km=km+matmul(matmul(transpose(bee),dee(:, :, etype(iel))),bee) &
      *det* weights(i);
  !-----now the fluid contribution-----
  call shape_fun(funf,points,i);
  call shape_der(derf,points,i) ;
  derivf=matmul(jac,derf)
  kp=kp + matmul(matmul(transpose(derivf),kay),derivf)*det &
    *weights(i)*dtim ;
  do l=1,nodf; volf(:,l)=vol(:)*funf(l); end do
  c= c+volf*det*weights(i)

```

```

        end do gauss_points_1
        ! for incremental formula
        call formke(km,kp,c,ke,theta)
!---collect nonzero entries from element stiffness matrices---
        call fmelspar(ntot,g,ke,iblkord,iebe,jebe,ebea,ebeanz)
    end do elements_2
!-----
write( *, '(a)') ' '
write( *, '(a,i16)')    &
    ' Returned true ebeanz after element assembly',ebeanz
write(11, '(a)') ' '
write(11, '(a,i16)')    &
    ' Returned true ebeanz after element assembly',ebeanz

call sortadd(ebeanz, jebe(1:ebeanz), iebe(1:ebeanz),      &
    ebea(1:ebeanz), neq+1, uanz)
write( *, '(a)') ' '
write( *, '(a)') ' The returned true storage for CSC Upper A: '
write( *, '(a,i9)') ' NNZ of CSC Upper A =', uanz
write(11, '(a)') '*****'
write(11, '(a)') ' The returned true storage for CSC Upper A: '
write(11, '(a,i9)') ' NNZ of CSC Upper A =', uanz

pneq = snpile + snraft ;    gneq = snsoil ;

!-----building the block diagonal preconditioner-----

write( *, '(a)') " Block diagonal preconditioner M1"
write(11, '(a)') " Block diagonal preconditioner M1"
!-----Sparse Cholesky factorization on Blk(1,1)-----
call cpu_time (tim1)
call countnzblk11 (pneq, neq, jebe(1:neq+1), iebe(1:uanz),      &
    ebea(1:uanz), pnz)
allocate (icsrp(pneq+1), jcsrp(pnz), csrp(pnz))

call formblk11 (pneq, neq, jebe, iebe, ebea, icsrp, jcsrp, csrp, pnz)
write( *, '(a)') ' '
write( *, '(a,i16)')    &
    ' Returned ture nnz for block(1,1) i.e. Pile =', pnz
write(11, '(a,i16)')    &
    ' Returned ture nnz for block(1,1) i.e. Pile =', pnz
punz = int((pnz+pneq)/2)

allocate (perm(pneq), perm_inv(pneq), colcnt(pneq),      &
    snode(pneq), xsuper(pneq+1))
write ( *, '(a)') ' '
write ( *, '(a)') 'the Multiple Minimal Degree (MMD) &
    & algorithm is used for'
write ( *, '(a)') 'ordering the coefficient matrix.'

! No. of adjacency entries excluding diagonal entries
nonzp = pnz-pneq
allocate ( adj_row(pneq+1), adj(nonzp) )
call form_adjacency (pneq, icsrp(1:pneq+1), jcsrp(1:pnz),      &
    adj_row, adj)

iwsiz=7 * pneq + 3
allocate ( xadj2(pneq+1), adjncy2(nonzp), iwork(iwsiz) )

! Save another copy (xadj2,adjncy2) because the
! (xadj,adjncy) structure is destroyed by the minimum

```

```

! degree ordering routine ordmmd.
      xadj2 = adj_row
      adjncy2 = adj
!-----
      iwsiz=4 * pneq
call ordmmd (pneq, xadj2, adjncy2, perm_inv, perm, iwsiz, &
              iwork, nsub, iflag )

      if (iflag== -1) then
          write ( *, '(a)' ) 'Insufficient Working Storage,   &
                          &   IWORK(:), when executing ORDMMD...'
          stop
      end if
      deallocate ( xadj2, adjncy2)
!-----Cholesky factorization

      iwsiz=7 * pneq + 3
call sfinit (pneq, nonzp, adj_row, adj, perm, perm_inv, &
              colcnt, nnzl, nsub, nsuper, snode, xsuper, &
              iwsiz, iwork, iflag )

      if (iflag== -1) then
          write ( *, '(a)' ) 'ERROR: Insufficient Working   &
                          &   Storage, IWORK(:), when executing SFINIT '
          stop
      end if
!-----
      allocate ( xlindx(nsuper+1), lindx(nsub), xlnz(pneq+1), &
              split(pneq) )
      iwsiz = nsuper + 2 * pneq + 1

call symfct (pneq, nonzp, adj_row, adj, perm, perm_inv, &
              colcnt, nsuper, xsuper, snode, nsub, xlindx, &
              lindx, xlnz, iwsiz, iwork, iflag )
      if (iflag== -1) then
          write ( *, '(a)' ) 'ERROR: Insufficient Working   &
                          &   Storage, IWORK(:), when executing SYMFCT... '
          stop
      elseif (iflag== -2) then
          write ( *, '(a)' ) 'ERROR: Inconsistency in The Input &
                          &   when executing SYMFCT... '
          stop
      end if
      deallocate ( adj_row, adj, colcnt)
!-----
      iwsiz = pneq
      allocate ( lnz(nnzl) )

write ( *, '(a,i9)' ) &
'No. of nonzeros of Cholesky factor of block P, NNZL =', nnzl
write (11, '(a,i9)' ) &
'No. of nonzeros of Cholesky factor of block P, NNZL =', nnzl

call inpnv (pneq, icsrp(1:pneq+1), jcsrp(1:pnz), &
              csrp(1:pnz), perm, perm_inv, nsuper, xsuper, &
              xlindx, lindx, xlnz, lnz, iwork)

      deallocate ( icsrp, jcsrp, csrp )
!-----
! bfnit: Initialization for block factorization
! size of the cache (in kilobytes) on the target machine

```

```

        cachsz = 16
    call bfini(pneq, nsuper, xsuper, snode, xlindx, lindx, &
        cachsz, tmpsiz, split)
    !-----
    !   blkfct: Numerical factorization
    iwsiz = 2 * pneq + 2 * nsuper
    level = 4
    !   level of loop unrolling while performing numerical
    !   factorization
    allocate ( tmpvec(tmpsiz) )
    if (level .eq. 1) then
        call blkfct(pneq, nsuper, xsuper, snode, split,      &
            xlindx, lindx, xlnz,lnz, iwsiz, iwork, &
            tmpsiz, tmpvec, iflag , mmpyl, smxpy1)
    elseif (level .eq. 2) then
        call blkfct(pneq, nsuper, xsuper, snode, split,      &
            xlindx, lindx, xlnz, lnz, iwsiz,iwork, &
            tmpsiz, tmpvec, iflag , mmpy2, smxpy2)
    elseif (level .eq. 4) then
        call blkfct(pneq, nsuper, xsuper, snode, split,      &
            xlindx, lindx, xlnz,lnz, iwsiz, iwork, &
            tmpsiz, tmpvec, iflag , mmpy4, smxpy4)
    elseif (level .eq. 8) then
        call blkfct(pneq, nsuper, xsuper, snode, split,      &
            xlindx, lindx, xlnz,lnz, iwsiz, iwork, &
            tmpsiz, tmpvec, iflag , mmpy8, smxpy8)
    endif
    if(iflag== -1) then
        write(*,'(a)') 'ERROR: Nonpositive Diagonal      &
            & Encountered, when executing BLKFCT... '
        stop
    elseif(iflag== -2) then
        write(*,'(a)') 'Insufficient Working Storage,    &
            & TEMP(:), when executing BLKFCT... '
        stop
    elseif(iflag== -3) then
        write(*,'(a)') 'ERROR: Insufficient Working      &
            & Storage,IWORK(:), when executing BLKFCT... '
        stop
    end if
    deallocate ( snode, split, iwork, tmpvec )
    !-----
    call cpu_time (tim2) ;

    !-form GJ diagonal
    allocate( diag(neq), diaga(neq),diagal(neq) )
    call formda_pg(neq,iebe(1:uanz),jebe(1:neq+1),ebea(1:uanz), &
        icho,ipre,sneq,coef,omega,diag,diaga,diagal)
    !-----
    call cpu_time (tim3);
    call cpu_time (ot2) ; ot2=ot2-ot1

    ! ----- enter the time-stepping loop-----
    allocate(loads(0:neq), ans(0:neq), tmpv(neq) )
    ttime = .0; loads = .0
    time_steps: do ns = 1 , nstep
        write(*, '(a, i5,a)') &
            ' Current Time Step is No.', ns , ' Step. '
        write(11,*) &
            '*****'
        ttime = ttime+dtim;

```

```

        write(11,'(a,e12.4)') "The current time is",ttime
        ans=.0;
        call kpu(iebe(1:uanz),jebe(1:neq+1),ebee(1:uanz),theta, &
            id,loads(1:),ans(1:));
        ans(0)=.0
! ----- Apply Constant Loading -----
        ! the Load is applied at the first step.
        if(ns == 1) then
            do i=1,loaded_nodes
                ans(nf(3, nodnum(i)))=-load_val(i)
            end do
        end if

! Permute the rhs (or load vector) according to block ordering
        call dvperm(neq,ans(1:),iblkord)

!----- Preconditioned Iterative Solver -----
        call cpu_time (it1)

        call mlsqmr(neq,iebe(1:uanz),jebe(1:neq+1),ebee(1:uanz), &
            nsuper,xsuper,xlindx,lindx,xlnz,lnz,perm, &
            perm_inv,diagal,ans(1:),maxit,tol,icc,itors, &
            resi);
        ans(0)=.0 ;

        deallocate (perm,perm_inv)
        call cpu_time (it2) ; it2=it2-it1

!      obtain the original(natural) solution from block
ordering
        do i = 1, neq; tmpv(iblkordrev(i)) = ans(i) ; end do
        loads(1:neq)=loads(1:neq) + tmpv

        write(11,'(a,i5,a,e12.4)') " Psolver took", iters, &
            " iterations to converge to ",resi

        write(11,'(a)') &
            " The nodal displacements and porepressures are : "
        do k=1,5; write(11,'(i7,a,4e13.5)')k," ",loads(nf(:,k)) ;
        end do
! -----
! node number for each x-z plane
! nodplane = (2*nx+1)*(nz+1)+(nx+1)*nz;
! node number between two x-z planes
! nodbetwplane = (nx+1)*(nz+1) ;
        do i=1, nye+1
            do j = 1, nxe+1
                k = ((2*nxe+1)*(nze+1)+(nx+1)*nze+(nx+1)*(nze+1)) &
                    *(i-1)+2*j -1
                write(12,'(2f10.2,e16.5)') widthx(j),widthy(i), &
                    loads(nf(3,k));
            end do
        end do

!-----recover stresses at Gauss-points-----
        elements_5 : do iel = 1 , nels
            num = g_num(:,iel); coord=transpose(g_coord(:,num))
            g = g_g( : , iel ); eld = loads( g ( 1 : ndof ) )
            ! print*, &
            ! "The Gauss Point effective stresses for element",iel,"are"
            gauss_points_2: do i = 1,nip
                call shape_der (der,points,i); jac= matmul(der,coord)

```

```

        call invert ( jac );      deriv= matmul(jac,der)
        bee= 0.;call beemat(bee,deriv);
        sigma= matmul(dee(:, :, etype(iel)),matmul(bee,eld))
        ! print*, "Point      ",i          ;! print*,sigma
    end do gauss_points_2
end do elements_5
end do time_steps
call cpu_time (tt2) ; tt2=tt2-tt1
write(11, '(a,f10.2)')      &
"      Operation time on blocks(1,1) is: ",tim2-tim1
write(11, '(a,f10.2)')      &
"      Operation time on blocks(2,2) is: ",tim3-tim2
write(11, '(a,f10.2)')      &
"      Overhead time is: ",ot2
write(11, '(a,f10.2)')      &
"      Iterative time (the last time step) is: ",it2
write(11, '(a,f10.2,a)')    &
"      Total runtime for the FEM program is: ",tt2,      &
"      seconds."
end program p3dbiot
!-----

```

Input file 'p3dbiot.dat' of a small example in 7×7×7 mesh

```

7 7 7 27 1. 1 1. 10000 1.e-6 -4.0 1.0 1 2 2 2 5
5 5 2 4 5
1.e-7 1.e-7 1.e-7 5.0 0.3
1.e-15 1.e-15 1.e-15 30000.0 0.2
1.e-15 1.e-15 1.e-15 30000.0 0.2
1 1 1 1
4 5 1 1
1 1 4 5
4 5 4 5
.0 1.0 2.0 3.0 4.0 5.0 7.5 10.0
.0 1.0 2.0 3.0 4.0 5.0 7.5 10.0
.0 -1.0 -2.0 -3.0 -4.0 -5.0 -7.5 -10.0
5.0 5.0 5 5 0.1

```

## E.4. Main program for $M_2$ -SQMR

```

!-----
program p3dbiot
!-----
! 3D Biot Consolidation Analysis of Soil-structure
! interaction problem
!-----
!   Program for 3-D consolidation analysis using 20-node
!   solid brick elements coupled to 8-node fluid elements,
!   incremental formulation. SQMR solver is used for
!   solving the linear system in each time step.
!
!   A = [Pile block                      = [ P      L      B1 ;
!       Soil Block                      L'     G      B2 ;

```

```

!               Fluid block]               B1'   B2'   -C ]
!
!   A block diagonal preconditioner is used.
!   Preconditioner M = [P               0;
!                       0   MSSOR(H)]
!
!       where P = pile block
!       MSSOR(H) = Modified SSOR for soil-fluid block H
!       H = [G   B2;
!            B2' -C]
!           diag(C) is replaced by alpha*diag(S)
!   S = C + B1' * inv(diag(P)) * B1 + B2' * inv(diag(G)) * B2
!       = approximate Schur complement
!-----
use new_library ; use geometry_lib;   use sparselib_v3;
use dfport ; use spchol ;

implicit none
integer:: i,j,k,l,nn,nels,nxe,nye,nze,nip,nodof=4,nod=20, &
nodf=8,nst=6,ndim=3,nodofs=3,ntot,ndof,iel,ns, &
nstep,inc,loaded_nodes,neq, nband,sneq,fneq, &
nmesh,ebeanz, maxit,iters,isolver,icho,ipre,icc, &
iinc,uanz,np_types,npiles,nels_pile,gneq,pneq,ir, &
k1,k2,pnz,gnz,llnz,anz,nonzp,ierr,punz,gunz, &
iwsiz,nsup,iflag,nnzl,nsuper,nnzlmax,tmpsiz, &
tmpmax,cachsz,level,nxr,nyr,nzr,neltp,nelbp, &
soil_id=1,pile_id=2,raft_id=2,snsoil,snpile, &
sraft,nr,nxlmsh,nylmsh,hneq,enz

real(8):: permx,permy,permz,e,v,det,dtim,theta,ttime,loadl, &
equipval,tol,coef,omega,resi,ot1,ot2,it1,it2,ttl, &
tt2,tim1,tim2,tim3,xlen,ylen

logical:: converged
character (len=15):: element = 'hexahedron'
!----- dynamic arrays-----
real(8),allocatable :: prop(:,,:),dee(:,,:), points(:,,:), &
coord(:,,:),derivf(:,,:),jac(:,,:),kay(:,,:),der(:,,:), &
deriv(:,,:),weights(:,,:),derf(:,,:),funf(:,,:),coordf(:,,:), &
bee(:,,:),km(:,,:),eld(:,,:),sigma(:,,:),kp(:,,:), ke(:,,:), &
g_coord(:,,:), kd(:,,:), fun(:,,:), c(:,,:), bk(:,,:),vol(:,,:), &
volf(:,,:), widthx(:,,:), widthy(:,,:),depth(:,,:), &
load_val(:,,:),loads(:,,:), ans(:,,:), ebea(:,,:),tmpv(:,,:), &
diag(:,,:),diaga(:,,:),diagal(:,,:), csrpf(:,,:),tmpvec(:,,:), &
lnz(:,,:),csce(:,,:)

integer,allocatable:: nf(:,,:), g(:,,:), num(:,,:), g_num(:,,:), &
g_g(:,,:),nodnum(:,,:),iebe(:,,:),jebe(:,,:),id(:,,:),etype(:,,:), &
pile_rc(:,,:),iblkordrev(:,,:), iblkord(:,,:),adj_row(:,,:), &
adj(:,,:),perm(:,,:),perm_inv(:,,:),icsrp(:,,:),jcsrp(:,,:), &
xadj2(:,,:), adjncy2(:,,:),colcnt(:,,:),snode(:,,:),xsuper(:,,:), &
iwork(:,,:),xlindx(:,,:), lindx(:,,:),xlnz(:,,:),split(:,,:), &
icsce(:,,:),jcsce(:,,:)

!----- input and initialization -----
open (10,file='p3dbiot.dat',status='old',action='read')
open (11,file='p3dbiot.res',status='replace',action='write')
open (12,file='surface.res',status='replace',action='write')
open (13,file='relres.res',status='replace',action='write')
call timestamp ( )
print *, " Iterative Solution Method for pile-group "
write(11,*)" Iterative Solution Method for pile-group "
print *, " The program is running, please wait..... "

```

```

read (10,*) nxe,nye,nze,nip,dtim,nstep,theta,maxit,tol,      &
      coef,omega,isolver,icho,ipre,icc,iinc

! Input of raft and pile details
read(10,*) nxr,nyr,nzr,npiles,nelbp ; neltp = nzr + 1
! nxr,nyr,nzr = no. of raft elements in x-, y-,
! and z-directions
! npiles = the number of piles.
! neltp = top element of pile (the no. in z-direction)
! nelbp = bottom element of pile (the no. in z-direction)
! if bottom element of pile < top element
if (nelbp < neltp)then
  write(*,*) 'There is No pile element'
  neltp = 0; nelbp = 0; ! stop
end if
! np_types = the number of material types (zones)
np_types = 3 ! iel = 1-Soil, 2-pile, and 3-raft materials
ndof=nod*3; ntot=ndof+nodf;
call msh_info(nxe,nye,nze,nels,nn)
write(11,'(a,i16)') ' Number of elements = ',nels
write(11,'(a,i16)') ' Number of nodes = ',nn
allocate(prop(5,np_types),etype(nels),dee(nst,nst,np_types), &
  points(nip,ndim),coord(nod,ndim),jac(ndim,ndim), &
  derivf(ndim,nodf),kay(ndim,ndim),der(ndim,nod), &
  deriv(ndim,nod),derf(ndim,nodf),funf(nodf), &
  coordf(nodf,ndim),bee(nst,ndof),km(ndof,ndof), &
  eld(ndof),sigma(nst),kp(nodf,nodf),ke(ntot,ntot), &
  g_g(ntot,nels),fun(nod),c(ndof,nodf),vol(ndof), &
  nf(ndof,nn),g(ntot),volf(ndof,nodf), &
  g_coord(ndim,nn),num(nod),weights(nip), &
  g_num(nod,nels),widthx(nxe+1),widthy(nye+1), &
  depth(nze+1),pile_rc(npiles,4) )

! kay=0.0; kay(1,1)=permx; kay(2,2)=permy; kay(3,3)=permz
! np_types material properties
read(10,*)(prop(:, i), i=1,np_types)
! prop(1:3,i) = permx,permy, and permz, respectively
! prop(4,i) = Effective Young's modulus E' ;
! prop(5,i) = Poisson's ratio ;

read(10,*)(pile_rc(i,:), i = 1, npiles) ;
etype = 1 ! initial set for soil elements
call form_idpile(nxe,nze,npiles,pile_rc,neltp,nelbp,etype, &
  pile_id)
call form_idraft(nxe,nye,nxr,nyr,nzr,etype,raft_id)
read (10,*) widthx, widthy, depth
call nfinfo(nxe,nye,nze,nf,neq,sneq,fneq)
!-----
do i=1, np_types;
  call deemat (dee(:, :, i), prop(4,i), prop(5,i) ) ; end do
call sample(element,points,weights)
allocate( id(1:neq) ) ;
id(:) = 1 ! initial set for soil elements.
! id(:) = 1 --> soil DOFs
! id(:) = 2 --> pile DOFs(pile_id)
! id(:) = 2 --> raft DOFs(raft_id)
! id(:) = 0 --> pore pressure DOFs
!----- loop the elements to set up global arrays-----
call cpu_time (ttl)
elements_1: do iel = 1, nels

```

```

call geometry_20bxz(iel,nxe,nze,widthx,widthy,depth, &
    coord,num)
inc=0 ;
do i=1,20;
do k=1,3; inc=inc+1;g(inc)=nf(k,num(i));end do;end do
! -----
do i=1, inc;
if (g(i)/=0) then
if (etype(iel) == pile_id) then ;
id(g(i)) = pile_id ;
elseif (etype(iel) == raft_id) then ;
id(g(i)) = raft_id ;
end if
end if
end do
! -----
do i=1,7,2; inc=inc+1;g(inc)=nf(4,num(i)); end do
do i=13,19,2; inc=inc+1;g(inc)=nf(4,num(i)); end do
! -----
do i=61, inc; if(g(i)/=0) id(g(i)) = 0 ; end do
! -----
g_num(:,iel)=num;g_coord(:,num)=transpose(coord);
g_g(:,iel)= g
if(nband<bandwidth(g))nband=bandwidth(g) ;
end do elements_1
write(11,'(a)') "Global coordinates "
do k=1,nn;
write(11,'(a,i7,a,3e12.4)') "Node",k," ",g_coord(:,k);
end do
write(11,'(a)') "Global node numbers "
do k = 1 , nels;
write(11,'(a,i6,a,20i7)') "Element ",k," ",g_num(:,k);
end do
write(11,'(2(a,i8))') &
"There are ",neq,&
" equations and the half-bandwidth is ",nband
snpile = 0; snraft = 0;
do i = 1,neq;
if (id(i) == pile_id) then ;
snpile = snpile + 1;
else if (id(i) == raft_id) then;
snraft = snraft + 1;
end if
end do
snsoil = sneq - snpile - snraft
write( *,'(a)') ' '
write( *,'(2(a,i12))') ' Pile DOFs = ',snpile, &
' Raft DOFs = ',snraft
write(11,'(2(a,i12))') ' Pile DOFs = ',snpile, &
' Raft DOFs = ',snraft
write( *,'(2(a,i12))') ' Soil DOFs = ',snsoil, &
' Pore Press. DOFs = ',fneq
write(11,'(2(a,i12))') ' Soil DOFs = ',snsoil, &
' Pore Press. DOFs = ',fneq
write( *,'(a,i12)') ' Total DOFs = ',neq ;
write(11,'(a,i12)') ' Total DOFs = ',neq

write ( *,'(a)') ' '
write ( *,'(a)') &
' Block ordering of the coefficient matrix '
write (11,'(a)') &

```

```

' Block ordering of the coefficient matrix '
allocate( iblkordrev(neq),iblkord(neq) )
call form_4bord(neq,snpile,snraft,snsoil,pile_id,raft_id, &
1,0,id,iblkordrev,iblkord)
!----- loading -----
read(10,*) xlen,ylen,nxlmsh,nylmsh,equpval
! xlen,ylen = length and breadth (m) of the loaded area
! (x- and y-direction)
! nxlmsh,nylmsh = no. of loaded elements in x- & y-direction
! equpval = applied load in MPa
loaded_nodes =(nxlmsh*2+1)*(nylmsh+1)+(nxlmsh+1)*nylmsh
! loaded_nodes = total no. of loaded nodes
allocate(nodnum(loaded_nodes),load_val(loaded_nodes) )
call load_raft(nxe,nye,nze,nels,nn,xlen,ylen,nxlmsh,nylmsh, &
equpval,loaded_nodes,nodnum,load_val)
!-----
ebeanz = int(ntot*(ntot+1)/2)*nels
write(*,'(a)') ' '
write(*,'(a)') &
' Estimated ebeanz for element stiffness integration'
write(*,'(a,i16)') ' and assembly ',ebeanz
write(11,'(a)') ' '
write(11,'(a)') &
' Estimated ebeanz for element stiffness integration'
write(11,'(a,i16)') ' and assembly ',ebeanz
allocate( iebe(ebeanz),jebe(ebeanz), ebea(ebeanz) )
!----- element stiffness integration and assembly ----
ebeanz=0 ! used for counting the true number
call cpu_time(ot1) ; kay =.0 ;
elements_2: do iel = 1 , nels
num = g_num( : , iel ); coord=transpose(g_coord(:,num))
g = g_g( : , iel ) ;
coordf(1 : 4 , : ) = coord(1 : 7 : 2, : )
coordf(5 : 8 , : ) = coord(13 : 19 : 2, : )
km = .0 ; c = .0 ; kp = .0
!-----forming Kay for each element -----
---
kay(1,1) = prop(1,etype(iel));
kay(2,2) = prop(2,etype(iel));
kay(3,3) = prop(3,etype(iel));
gauss_points_1: do i = 1 , nip
call shape_der(der,points,i); jac = matmul(der,coord)
det = determinant(jac ); call invert(jac);
deriv = matmul(jac,der); call beemat(bee,deriv);
vol(:)=bee(1,:)+bee(2,:)+bee(3,:);
km=km+matmul(matmul(transpose(bee),dee(:, :, etype(iel))),bee) &
*det* weights(i);
!-----now the fluid contribution-----
call shape_fun(funf,points,i);
call shape_der(derf,points,i) ;
derivf=matmul(jac,derf)
kp=kp+matmul(matmul(transpose(derivf),kay),derivf) &
*det*weights(i)*dtim ;
do l=1,nodf; volf(:,l)=vol(:)*funf(l); end do
c= c+volf*det*weights(i)
end do gauss_points_1
! for incremental formula
call formke(km,kp,c,ke,theta)
!---collect nonzero entries from element stiffness matrices---
call fmelspar(ntot,g,ke,iblkord,iebe,jebe,ebea,ebeanz)
end do elements_2

```

```

!-----
write( *, '(a)' ) ' '
write( *, '(a,i16)' )    &
    ' Returned true ebeanz after element assembly', ebeanz
write(11, '(a)' ) ' '
write(11, '(a,i16)' )    &
    ' Returned true ebeanz after element assembly', ebeanz
call sortadd(ebeanz, jebe(1:ebeanz), iebe(1:ebeanz),      &
    ebea(1:ebeanz), neq+1, uanz)
write( *, '(a)' ) ' '
write( *, '(a)' ) ' The returned true storage for CSC Upper A: '
write( *, '(a,i9)' ) ' NNZ of CSC Upper A =', uanz
write(11, '(a)' ) '*****'
write(11, '(a)' ) ' The returned true storage for CSC Upper A: '
write(11, '(a,i9)' ) ' NNZ of CSC Upper A =', uanz

pneq = snpile + snraft ;   gneq = snsoil ;
!-----building the block diagonal preconditioner-----

write( *, '(a)' ) " Block diagonal preconditioner M2"
write(11, '(a)' ) " Block diagonal preconditioner M2"
!-----Sparse Cholesky factorization on Blk(1,1)-----
call cpu_time (tim1)
call countnzblk11 (pneq, neq, jebe(1:neq+1), iebe(1:uanz),      &
    ebea(1:uanz), pnz)
allocate (icsrp(pneq+1), jcsrp(pnz), csrp(pnz))

call formblk11 (pneq, neq, jebe, iebe, ebea, icsrp, jcsrp, csrp, pnz)
write( *, '(a)' ) ' '
write( *, '(a,i16)' )    &
    ' Returned ture nnz for block(1,1) i.e. Pile =', pnz
write(11, '(a,i16)' )    &
    ' Returned ture nnz for block(1,1) i.e. Pile =', pnz
punz = int((pnz+pneq)/2)

allocate (perm(pneq), perm_inv(pneq), colcnt(pneq), &
    snode(pneq), xsuper(pneq+1))
write ( *, '(a)' ) ' '
write ( *, '(a)' ) 'the Multiple Minimal Degree (MMD) &
    & algorithm is used for'
write ( *, '(a)' ) 'ordering the coefficient matrix.'

! No. of adjacency entries excluding diagonal entries
nonzp = pnz-pneq
allocate ( adj_row(pneq+1), adj(nonzp) )
call form_adjacency (pneq, icsrp(1:pneq+1), jcsrp(1:pnz), &
    adj_row, adj)

iwsiz=7 * pneq + 3
allocate ( xadj2(pneq+1), adjncy2(nonzp), iwork(iwsiz) )

! Save another copy (xadj2,adjncy2) because the
! (xadj,adjncy) structure is destroyed by the minimum
! degree ordering routine ordmmd.
xadj2 = adj_row
adjncy2 = adj
!-----
iwsiz=4 * pneq
call ordmmd (pneq, xadj2, adjncy2, perm_inv, perm, iwsiz, &
    iwork , nsub, iflag )

```

```

        if(iflag== -1) then
            write( *, '(a)' ) 'Insufficient Working Storage,    &
                                &      IWORK(:), when executing ORDMMD...'
            stop
        end if
        deallocate ( xadj2, adjncy2)
!-----Cholesky factorization

        iwsiz=7 * pneq + 3
call sfinit (pneq, nonzp, adj_row, adj, perm, perm_inv, &
            colcnt, nnzl, nsub, nsuper, snode, xsuper,    &
            iwsiz, iwork , iflag )

        if(iflag== -1) then
            write( *, '(a)' ) 'ERROR: Insufficient Working    &
                                &      Storage, IWORK(:), when executing SFINIT '
            stop
        end if
!-----
        allocate( xlindx(nsuper+1), lindx(nsub), xlnz(pneq+1), &
            split(pneq))
        iwsiz = nsuper + 2 * pneq + 1

call symfct (pneq, nonzp, adj_row, adj, perm , perm_inv, &
            colcnt, nsuper, xsuper, snode , nsub, xlindx,    &
            lindx , xlnz , iwsiz , iwork , iflag )
        if(iflag== -1) then
            write( *, '(a)' ) 'ERROR: Insufficient Working    &
                                &      Storage, IWORK(:), when executing SYMFCT... '
            stop
        elseif(iflag== -2) then
            write( *, '(a)' ) 'ERROR: Inconsistency in The Input    &
                                &      & when executing SYMFCT... '
            stop
        end if
        deallocate ( adj_row, adj, colcnt)
!-----
        iwsiz = pneq
allocate (lnz(nnzl))
call inpnv (pneq, icsrp(1:pneq+1), jcsrp(1:pnz),    &
            csrp(1:pnz), perm, perm_inv, nsuper, xsuper,    &
            xlindx, lindx, xlnz, lnz, iwork)

        deallocate ( icsrp, jcsrp, csrp )
!-----
! bfninit: Initialization for block factorization
! size of the cache (in kilobytes) on the target machine
        cachsz = 16
call bfninit(pneq, nsuper, xsuper, snode, xlindx, lindx, &
            cachsz, tmpsiz, split)
!-----
! blkfct: Numerical factorization
        iwsiz = 2 * pneq + 2 * nsuper
        level = 4
        ! level of loop unrolling while performing
        ! numerical factorization
        allocate (tmpvec(tmpsiz))
        if (level .eq. 1) then
            call blkfct(pneq, nsuper, xsuper, snode, split,    &
                xlindx, lindx, xlnz, lnz, iwsiz, iwork, tmpsiz, &
                tmpvec, iflag , mmpyl, smxpyl)

```

```

elseif (level .eq. 2) then
    call blkfct(pneq, nsuper, xsuper, snode, split,      &
               xlindx, lindx, xlnz,lnz, iwsiz, iwork, tmpsiz, &
               tmpvec, iflag , mmpy2, smxpy2)
elseif (level .eq. 4) then
    call blkfct(pneq, nsuper, xsuper, snode, split,      &
               xlindx, lindx, xlnz,lnz, iwsiz, iwork, tmpsiz, &
               tmpvec, iflag , mmpy4, smxpy4)
elseif (level .eq. 8) then
    call blkfct(pneq, nsuper, xsuper, snode, split,      &
               xlindx, lindx, xlnz,lnz, iwsiz, iwork, tmpsiz, &
               tmpvec, iflag , mmpy8, smxpy8)
endif
if(iflag==1)then
    write(*,'(a)') 'ERROR: Nonpositive Diagonal  &
    & Encountered, when executing BLKFCT... ' ;
stop
elseif(iflag==2) then
    write(*,'(a)') 'Insufficient Working Storage, &
    & TEMP(:), when executing BLKFCT... ' ;
stop
elseif(iflag==3) then
    write(*,'(a)') 'ERROR: Insufficient Working    &
    & Storage,IWORK(:), when executing BLKFCT... ' ; stop
end if
deallocate ( snode, split, iwork, tmpvec )
!-----
call cpu_time (tim2) ;

!----- formation fo GJ diagonal for block H -----
allocate( diag(neq), diaga(neq),diagal(neq) )
hneq = gneq + fneq ;
call formdh(neq,iebe(1:uanz),jebe(1:neq+1),ebee(1:uanz),      &
            pneq,sneq,coef,omega,diag,diaga,diagal)
! diag = original diag of H, diag(1:pneq) = 1.0
! diaga = modified diag of H for MSSOR
! diagal = inverse of diaga
!----- extraction of block E -----
call cscnnz(neq,1,pneq,pneq+1,neq,jebe(1:neq+1),    &
            iebe(1:uanz),enz)
allocate ( icsce(enz),jcsce(hneq+1),csce(enz) )
call cscsubmat(neq,1,1,pneq,pneq+1,neq,ebee(1:uanz),      &
               jebe(1:neq+1),iebe(1:uanz),pneq,hneq,csce,  &
               jcsce,icsce)
write( *,'(a,i16)') ' Returned ture nnz of block E =', enz
!-----
call cpu_time (tim3);
call cpu_time (ot2) ; ot2=ot2-ot1
! ----- enter the time-stepping loop-----
allocate(loads(0:neq), ans(0:neq), tmpv(neq) )
ttime = .0; loads = .0
time_steps: do ns = 1 , nstep
    write(*, '(a, i5,a)') &
    ' Current Time Step is No.', ns , ' Step. '
    write(11,*) &
    '***** '
    ttime = ttime+dtim;
    write(11,'(a,e12.4)') "The current time is",ttime
    ans=.0;

    call kpu(iebe(1:uanz),jebe(1:neq+1),ebee(1:uanz),theta, &

```

```

        id,loads(1:),ans(1:));          ans(0)=.0
! ----- Apply Constant Loading -----
! the Load is applied at the first step.
    if (ns == 1) then
        do i=1,loaded_nodes
            ans(nf(3, nodnum(i)))=-load_val(i)
        end do
    end if

! Permute the rhs (or load vector) according to block ordering
    call dvperm(neq,ans(1:),iblkord)

!----- Preconditioned Iterative Solver -----
    call cpu_time (it1)

    call m2sqmr (neq,iebe(1:uanz),jebe(1:neq+1),ebee(1:uanz),      &
        nsuper,xsuper,xlindx,lindx,xlnz,lnz,perm,                &
        perm_inv,pneq,sneq,hneq,icsce,jcsce,csce,diag,          &
        diaga,diagal,ans(1:),maxit,tol,iinc,itors,resi) ;
        ans(0)=.0 ;

    deallocate (perm,perm_inv,icsce,jcsce,csce)

    call cpu_time (it2) ; it2=it2-it1

! obtain the original(natural) solution from block ordering
    do i = 1, neq; tmpv(iblkordrev(i)) = ans(i) ; end do
        loads(1:neq)=loads(1:neq) + tmpv

    write(11,'(a,i5,a,e12.4)') " Psolver took", iters, &
        " iterations to converge to ",resi

    write(11,'(a)')      &
        " The nodal displacements and porepressures are      :"
    do k=1,5;
        write(11,'(i7,a,4e13.5)')k,"      ",loads(nf(:,k)) ; end do
        ! -----
        ! node number for each x-z plane
        ! nodplane = (2*nx+1)*(nz+1)+(nx+1)*nz;
        ! node number between two x-z planes
        ! nodbetwplane = (nx+1)*(nz+1) ;
        do i=1, nye+1
            do j = 1, nxe+1
                k = ((2*nxe+1)*(nze+1)+(nxe+1)*nze+(nxe+1)*(nze+1)) &
                    *(i-1)+2*j -1
                write(12,'(2f10.2,e16.5)') widthx(j),widthy(i),      &
                    loads(nf(3,k));
            end do
        end do

!-----recover stresses at Gauss-points-----
    elements_5 : do iel = 1 , nels
        num = g_num(:,iel); coord=transpose(g_coord(:,num))
        g = g_g( : , iel ); eld = loads( g ( 1 : ndof ) )
! print*, &
! "The Gauss Point effective stresses for element",iel,"are"
        gauss_points_2: do i = 1,nip
            call shape_der (der,points,i); jac= matmul(der,coord)
            call invert ( jac ); deriv= matmul(jac,der)
            bee= 0.;call beemat(bee,deriv);
            sigma= matmul(dee(:, :, etype(iel)),matmul(bee,eld))

```

```

        ! print*, "Point      ", i          ;! print*, sigma
    end do gauss_points_2
end do elements_5
end do time_steps
call cpu_time (tt2) ; tt2=tt2-tt1
write(11, '(a,f10.2)')      &
"      Operation time on blocks(1,1) is: ", tim2-tim1
write(11, '(a,f10.2)')      &
"      Operation time on blocks(2,2) is: ", tim3-tim2
write(11, '(a,f10.2)')      &
"      Overhead time is: ", ot2
write(11, '(a,f10.2)')      &
"      Iterative time (the last time step) is: ", it2
write(11, '(a,f10.2,a)')    &
"      Total runtime for the FEM program is: ", tt2,      &
"      seconds."

end program p3dbiot
!-----

```

The input file is the same as for the  $M_1$ -SQMR.

## E.5. New routines for module sparselib\_v3

```

module sparselib_v3

  use dfport ; use spchol
  contains

  !-----
  subroutine form_idpile(nxe,nze,npiles,pile_rc,neltp,nelbp,      &
    etype,pile_id)
    ! This subroutine forms the element id of pile
    !      etype(i) = 1          - soil element
    !      etype(i) = pile_id    - pile element
    ! pile_rc = pile location in terms of finite elements in
    !      row-column form
    ! npiles = no. of piles
    ! neltp = top element of pile (in z-direction)
    ! nelbp = bottom element of pile (in z-direction)
    ! etype = array of element identification
    ! pile_id = a given id for pile
    implicit none
    integer, intent(in) :: nxe,nze,npiles,pile_rc(:,,:),neltp,      &
      nelbp,pile_id
    integer, intent(inout) :: etype(:)
    integer :: i,j,k,l,t,kx1,kx2,ky1,ky2,nels_pile

    nels_pile = nelbp - neltp + 1
    ! no. of elements in length of pile
    do i = 1, npiles
      kx1 = pile_rc(i,1); kx2 = pile_rc(i,2);
      ky1 = pile_rc(i,3); ky2 = pile_rc(i,4);
      do j=kx1, kx2
        do k = ky1, ky2
          t = nxe*nze*(k-1) + (neltp-1)*nxe + j ;
          etype(t) = pile_id;
          do l = 1, nels_pile-1;
            etype(t + l*nxe) = pile_id ;

```

```

        end do
    end do
end do
!
return
end subroutine form_idpile
!-----
subroutine form_idraft(nxe,nze,nxr,nyr,nzr,etype,raft_id)
! This subroutine forms the element id for raft elements
!     etype(i) = raft_id - raft element
!     etype = array of element identification
!     raft_id = a given id for raft elements
!-----
implicit none
integer,intent(in):: nxe,nze,nxr,nyr,nzr,raft_id
integer,intent(inout):: etype(:)
!local variables
integer:: i,j,k,nelxz,m
m = 0
do j = 1,nyr
    nelxz = (j-1)*(nxe*nze)
    do k = 1,nzr
        m = (k-1)*nxe + nelxz
        do i = 1,nxr
            m = m+1; etype(m) = raft_id ! raft elements
        end do
    end do
end do
!
return
end subroutine form_idraft
!-----
subroutine form_4bord(neq,snb11,snb22,snb33,idb11,idb22,      &
                    idb33,idb44,id,iordrev,iord)
! This subroutine generates "iord" array to form 4 x 4 block
! structured matrix
!     neq = total number of DOFs
!     snb11,snb22,snb33
!         = no. of DOFs for block 11, 22, and 33 respectively
!         e.g., no. of DOFs for pile, raft, and soil
!         block 44 is for pore pressure DOFs (in this case)
!     idb11,idb22,idb33
!         = material id of block 11, 22, and 33 respectively
!         e.g., idb11 = id_pile, idb22 = id_raft, idb33 = id_soil
!     idb44 = In consolidation analysis, idb44 = 0,
!             pore pressure DOFs
!             = In drained analysis, idb44 = id of 4th material,
!             if any
!     block matrix= [Pile
!                   Raft
!                   Soil
!                   Pore Pressure] 4 x 4 matrix
!     id(:, :) = array of id for DOFs corresponding to
!                 material and pore pressure
!     iord,iordrev = array for re-ordering the matrix and inverse
!                   re-ordering
!-----
implicit none
integer,intent(in):: neq,snb11,snb22,snb33,idb11,idb22,      &
                    idb33,idb44,id(:)

```

```

integer,intent(out)::iordrev(:),iord(:)
integer:: i,j,ic0,ic1,ic2,ic3,sn          ! local variables
!
sn = snb11 + snb22 + snb33
ic0 = 0; ic1 = snb11 ; ic2 = snb11 + snb22; ic3 = sn
do i = 1,neq
  if (id(i) == idb11) then                ! block 11 (e.g. pile)
    ic0 = ic0 + 1; iordrev(ic0) = i; iord(i) = ic0
  else if (id(i) == idb22) then           ! block 22 (e.g. raft)
    ic1 = ic1 + 1; iordrev(ic1) = i; iord(i) = ic1
  else if (id(i) == idb33) then           ! block 33 (e.g. soil)
    ic2 = ic2 + 1; iordrev(ic2) = i; iord(i) = ic2
  else if (id(i) == idb44) then
    ! pore pressure block or 4th material
    ic3 = ic3 + 1; iordrev(ic3) = i; iord(i) = ic3
  end if
end do
!
return
end subroutine form_4bord
!-----
subroutine form_3bord(neq,snb11,snb22,idb11,idb22,idb33,id,    &
  iordrev,iord)

! This subroutine generates "iord" array to form 3 x 3 block
! structured matrix
!       neq = total number of DOFs
! snb11,snb22 = no. of DOFs for block 11 and 22 respectively
!       e.g., no. of DOFs for pile, raft, and soil
!       block 33 can be pore pressure DOFs
! idb11,idb22,idb33
!       = material id of block 11, 22, and 33 respectively
! e.g., idb11 = id_pile, idb22 = id_raft, idb33 = id_soil
!       idb33 = In consolidation analysis, idb33 = 0,
!       pore pressure DOFs
!       = In drained analysis,
!       idb33 = id of 3rd material, if any
! A = [Pile           [Pile
!       Raft           OR      Soil
!       Soil]         Pore Pressure] 3 x 3
! id(:, :) = array of id for DOFs corresponding to material
! and pore pressure
! iord,iordrev = array for re-ordering the matrix and inverse
! re-ordering
!-----
implicit none
integer,intent(in):: neq,snb11,snb22,idb11,idb22,idb33,id(:)
integer,intent(out)::iordrev(:),iord(:)
integer:: i,j,ic0,ic1,ic2,sn          ! local variables
!
sn = snb11 + snb22
ic0 = 0; ic1 = snb11 ; ic2 = sn
do i = 1,neq
  if (id(i) == idb11) then                ! block 11 (e.g. pile)
    ic0 = ic0 + 1; iordrev(ic0) = i; iord(i) = ic0
  else if (id(i) == idb22) then           ! block 22 (e.g. raft)
    ic1 = ic1 + 1; iordrev(ic1) = i; iord(i) = ic1
  else if (id(i) == idb33) then
    ! block 33 (e.g. soil or pore pressure)
    ic2 = ic2 + 1; iordrev(ic2) = i; iord(i) = ic2
  end if
end do

```

```

    end do
    !
    return
end subroutine form_3bord
!-----
subroutine formda_pg(n,icsc,jcsc,csca,icho,ipre,sn,coef,      &
    omega,d,da,dal)
! This subroutine forms GJ diagonal vector - da (d and dal);
! In this routine:
!       n: dimension of coefficient matrix A;
! icsc,jcsc,csca:
!       CSC storage of upper triangular part of matrix A;
!       icho: choose standard or modified preconditioner.
!             =1: standard preconditioner.
!             =2: generalized or modified preconditioner.
!       ipre: choose preconditioner,
!             =1: Jacobi preconditioner.
!             =2: SSOR preconditioner.
!       coef: the scaling factor for GJ diagonal vector.
!       omega: relaxation parameter, which is applied to MSSOR.
!       id: a vector to indicate the type of current DOF,
!            id(j)= 0 for pore water pressure DOF;
!            id(j)= 1 for displacement DOF.
!       d: diagonal of A;
!       da: modified diagonal for MSSOR preconditioner;
!       dal: inverse of da;
implicit none
real(8):: coef,omega,d(:),da(:),dal(:),csca(:),absv,      &
    maxabs,minabs
integer:: n,sn,s1,j,r,k,k1,k2,icho,ipre,icsc(:), jcsc(:)
!
    s1 = sn + 1
do j=1, n;
    r=jcsc(j+1)-1 ; da(j) = csca(r);
end do
!
! Transfer diagonal of A from da to d;
if(ipre==2) d = da ;
! For generalized or modified preconditioner
if(icho == 2)then
!
    do j=s1, n
        k1=jcsc(j) ; k2=jcsc(j+1)-2
        do k=k1 , k2
            r = icsc(k)
            if( r > sn)exit
            da(j)=da(j)-csca(k)**2/da(r) ;
        end do
    end do

!
    coef = coef/omega ;
! coef-scaling factor (negative is preferred)
    do j=s1, n
        da(j)=coef*abs(da(j))
    end do
    do j = 1, sn
        da(j)=da(j)/omega
    end do
end if
    dal = 1./da ;

```

```

!
  return
end subroutine formda_pg
!-----
subroutine formdh(n,icsc,jcsc,csca,np,sn,coef,omega,dh,mdh, &
mdh1)
! This subroutine forms GJ diagonal vector for block H where
!   A = [Pile block           = [ P       L       B1 ;
!       Soil Block           L'      G       B2 ;
!       Fluid block]         B1'     B2'     -C ]
!
! and   H = [G       B2;
!            B2'    -C]
!
!   A block diagonal preconditioner is used.
!   Preconditioner M = [P           0;
!                      0   MSSOR(H)]
!
!       where P = pile block
!       MSSOR(H) = Modified SSOR for soil-fluid block H
!                 diag(C) is replaced by alpha*diag(S)
!   S = C + B1' * inv(diag(P)) * B1 + B2' * inv(diag(G)) * B2
!   = approximate Schur complement
! This subroutine forms:
!   mdh = [ I           0           0;
!           0   diag(G)/omega   0;
!           0           0   (alpha*diag(S))/omega]
!
! In this routine:
!       n: dimension of coefficient matrix A;
!   icsc,jcsc,csca: CSC storage of upper triangular part
!                   of matrix A;
!       np: pile displacement DOFs
!       sn: total (pile + soil) displacement DOFs
!       nh: dimension of block H
!            (soil + pore pressure DOFs)
!       coef: the scaling factor for GJ diagonal vector for H
!       omega: relaxation parameter, which is applied to MSSOR.
!       dh: original diagonal of H;
!       mdh: modified diagonal for MSSOR preconditioner;
!       mdh1: inverse of mdh;
!-----
implicit none
real(8),intent(in):: csca(:),omega ;
real(8),intent(inout):: coef
real(8),intent(out):: dh(:),mdh(:),mdh1(:)
integer,intent(in):: n,np,sn,icsc(:),jcsc(:)
integer:: s1,j,ir,k,k1,k2          ! local variables
!
  s1 = sn + 1
  do j=1, n;
    ir=jcsc(j+1)-1 ; dh(j) = csca(ir);
  end do
!
  mdh = dh ; ! Transfer diagonal of A from dh to mdh;
!
!For generalized or modified diagonal for MSSOR preconditioner
  do j=s1, n
    k1=jcsc(j) ; k2=jcsc(j+1)-2
    do k=k1 , k2
      ir = icsc(k)

```

```

        if( ir > sn) exit
        mdh(j)=mdh(j)-csca(k)**2/dh(ir) ;
    end do
end do
!
mdh(1:np) = 1.0 ;
dh(1:np) = 1.0 ; ! Identity for pile block
coef = coef/omega ;
! coef-scaling factor (negative is preferred)
do j=s1, n
    mdh(j)=coef*abs(mdh(j))
end do
    do j = np+1, sn
        mdh(j)=mdh(j)/omega
    end do

mdh1(np+1:n) = 1./mdh(np+1:n) ;
!
return
end subroutine formdh
!-----
subroutine cscnnz(n,i1,i2,j1,j2,ja,ia,nnz)
! This subroutine counts the number of nonzeros in the
! submatrix A(i1:i2,j1:j2)
! In this subroutine,
!     n      = column dimension of the matrix A
!     i1,i2 = two integers with i2 .ge. i1 indicating the range
!              of rows to be extracted.
!     j1,j2 = two integers with j2 .ge. j1 indicating the range
!              of columns to be extracted.
! * There is no checking whether the input values for
!     i1, i2, j1, j2 are between 1 and n.
! a,
! ja,
! ia  = matrix in compressed sparse column format
!     ia = array containing the row indices and
!     ja = pointer to the beginning of the each
!         column in array a
!-----
implicit none
integer,intent(in):: n,i1,i2,j1,j2,ia(:),ja(:)
integer,intent(out):: nnz
integer:: i,j,k,k1,k2,jj,nr,nc

    nr = i2-i1+1      ! number of rows of submatrix
    nc = j2-j1+1      ! number of columns of submatrix

if ( nr <= 0 .or. nc <= 0) return

if (nr == n .and. nc == n) then
    nnz = ja(n+1)-1
    return
end if

nnz = 0;

do j = 1,nc
    jj = j1+j-1
    k1 = ja(jj)
    k2 = ja(jj+1)-1
    do k = k1,k2

```

```

        i = ia(k)
        if (i >= i1 .and. i <= i2) then
            nnz = nnz+1
        end if
    end do
end do

return
end subroutine cscnnz
!-----
subroutine cscsubmat (n,job,i1,i2,j1,j2,a,ja,ia,nr,nc,ao,jao, &
    iao)
! This subroutine extracts the submatrix A(i1:i2,j1:j2) and
! puts the result in matrix ao,iao,jao
!---- In place: ao,jao,iao may be the same as a,ja,ia.
!-----
! on input
!-----
! n    = column dimension of the matrix
! i1,i2 = two integers with i2 .ge. i1 indicating the range
!         of rows to be extracted.
! j1,j2 = two integers with j2 .ge. j1 indicating the range
!         of columns to be extracted.
!       * There is no checking whether the input values
!         for i1, i2, j1,j2 are between 1 and n.
! a,
! ja,
! ia    = matrix in compressed sparse column format.
!
! job = job indicator: if job .ne. 1 then the real values
!       in a are NOT extracted, only the column indices
!       (i.e. data structure) are. Else, values as well
!       as column indices are extracted...
!
! on output
!-----
! nr = number of rows of submatrix
! nc = number of columns of submatrix
! * if either of nr or nc is nonpositive the code will quit.
!
! ao,
! jao,iao = extracted matrix in general sparse column
! format with iao containing the row indices, and jao being the
! pointer to the beginning of the each column, in arrays a,ia.
! Reference: submat in SPARSKIT2, Y. Saad, Sep. 21 1989
!
!-----
implicit none
integer,intent(in):: n,job,i1,i2,j1,j2,ia(:),ja(:)
integer,intent(out):: nr,nc,jao(:),iao(:)
real(8),intent(in):: a(:) ; real(8),intent(out):: ao(:)
integer:: i,j,k,jj,k1,k2,klen      ! local variables
    nr = i2-i1+1
    nc = j2-j1+1
!
!   if ( nr <= 0 .or. nc <= 0) return
!
!   klen = 0
!
!   simple procedure. proceeds column-wise...
!

```

```

do j = 1,nc
  jj = j1+j-1
  k1 = ja(jj)
  k2 = ja(jj+1)-1
  jao(j) = klen+1
!-----
do k = k1,k2
  i = ia(k)
  if (i >= i1 .and. i <= i2) then
    klen = klen+1
    if (job .eq. 1) ao(klen) = a(k)
    iao(klen) = i - i1+1
  endif
end do
end do
jao(nc+1) = klen+1
return
!-----end-of submat-----
!-----
end subroutine cscsubmat
!-----
subroutine msh_info(nxe,nye,nze,nels,nn) !,nr
! This subroutine calculates the no. of elements,nodes,
! restrained nodes
!-----
implicit none
integer,intent(in):: nxe,nye,nze
integer,intent(out):: nels,nn !,nr
nels = nxe*nye*nze
nn = ((2*nxe+1)*(nze+1) + (nxe+1)*nze)*(nye+1) + &
      (nxe+1)*(nze+1)*nye
! nr = 3*nxe*nye*nze + 4*(nxe*nye+nye*nze+nze*nxe) &
!       + nxe+nye+nze + 2
return
end subroutine msh_info
!-----
subroutine nfinfo(nxe,nye,nze,nf,neq,sneq,fneq)
! This subroutine generates nf array with only 0 and 1
! integer value for 3D Biot's consolidation problems, and
! this subroutine is restricted to the geometric model
! discussed in this thesis.
!       nn: total number of nodes;
!       nodof: number of freedoms per node;
!       nxe, nye, nze: number of elements in each direction
!       nf: generated nodal freedom array,
!           nf(:, :) = 1, free DOF;
!           nf(:, :) = 0, restricted DOF.
!       neq: number of DOFs in the mesh ;
!       sneq: number of DOFs corresponding to
!             displacement;
!       fneq: number of DOFs corresponding to pore
!             pressure;
!             and there exists "neq = sneq + fneq".
!-----

implicit none
integer,intent(in):: nxe,nye,nze ;
integer,intent(out):: nf(:, :), neq, sneq, fneq;
integer:: i, j, k, nn, nodof, nodplane, nodbetwplane
!-----
!nn = ubound(nf,2); nodof = ubound(nf,1);

```

```

! node number for each x-z plane
nodplane=(2*nxe+1)*(nze+1)+(nxe+1)*nze ;
! node number between two x-z planes
nodbetwplane=(nxe+1)*(nze+1)
!
nf = 1 ;      ! initialize all nodes unrestricted
xdirection1: do i=0, nye
  xloop1: do j=0, nze
    ! right element corner nodes in
nodplane
nf(1,i*(nodplane+nodbetwplane)+2*nxe+1+j    &
      *(3*nxe+2))=0 ;
    ! left element corner node in nodplane
nf(1,i*(nodplane+nodbetwplane)+1+j          &
      *(3*nxe+2))=0 ;
  end do xloop1
!
  xloop2: do j=1, nze
    ! right element midside node in nodplane
nf(1,i*(nodplane+nodbetwplane)+j*(3*nxe+2)) &
    =0 ;
    ! left element midside node in nodplane
nf(1,i*(nodplane+nodbetwplane)+j          &
      *(3*nxe+2)-nxe)=0 ;
  end do xloop2
end do xdirection1
!
xdirection2: do i=1, nye
  xloop3: do j=1, nze+1
    ! Right element mid node in nodbetwplane
nf(1,i*nodplane+(i-1)*nodbetwplane+j      &
      *(nxe+1))=0 ;
    ! Left element mid node in nodbetwplane
nf(1,i*nodplane+(i-1)*nodbetwplane+(j-1)  &
      *(nxe+1)+1)=0 ;
  end do xloop3
end do xdirection2
!-----
ydirection1: do i=1, nodplane
  ! front face nodplane
nf(2,i)=0
  ! back face nodplane
nf(2,nye*(nodplane+nodbetwplane)+i)=0
end do ydirection1
!-----
xyzdirection1: do i=0, nye
  xyzloop1: do j=1, 2*nxe+1
    ! bottom nodes in nodplanes
nf(1:3,i*(nodplane+nodbetwplane)+nodplane &
      -(2*nxe+1)+j)=0 ;
  end do xyzloop1
end do xyzdirection1
!
xyzdirection2: do i=1, nye
  xyzloop2: do j=1, nxe+1
    ! bottom nodes in nodbetwplanes
nf(1:3,i*(nodplane+nodbetwplane)-(nxe+1)+j)=0 ;
  end do xyzloop2
end do xyzdirection2
!-----

```

```

!Pore pressure boundary condition
nf(4,:) = 0 !Initialization including top B.C.
ppressure: do i = 0,nye
  ploop1: do j = 1,nze
    ploop2: do k = 1,nxe+1
      ! remaining corner nodes of elements
      nf(4,i*(nodplane+nodbetwplane)+j*(3*nxe+2) &
        +(2*k-1))=1 ;
    end do ploop2
  end do ploop1
end do ppressure
!
fneq = sum(nf(4,:));
call formnf(nf); neq=maxval(nf); sneq = neq - fneq ;
!
return
end subroutine nfinfo
!-----
subroutine nfinfo_drained(nxe,nye,nze,nf,neq)
! This subroutine generates nf array with only 0 and 1
! integer value for 3D drained problems
! with Dirichlet Boundary condition
! nn: total number of nodes;
! nodof: number of freedoms per node;
! nxe, nye, nze: number of elements in each direction
! nf: generated nodal freedom array,
! nf(:, :) = 1, free DOF;
! nf(:, :) = 0, restricted DOF.
! neq: number of DOFs in the mesh
!-----

implicit none
integer,intent(in)::nxe,nye,nze ;
integer,intent(out)::nf(:, :),neq;
integer::i,j,k,nn,nodof,nodplane,nodbetwplane
!-----
!nn = ubound(nf,2);nodof = ubound(nf,1);
! node number for each x-z plane
nodplane=(2*nxe+1)*(nze+1)+(nxe+1)*nze ;
! node number between two x-z planes
nodbetwplane=(nxe+1)*(nze+1)
!
nf = 1 ; ! initialize all nodes unrestricted
xdirection1: do i=0, nye
  xloop1: do j=0, nze
    ! right element corner nodes in nodplane
    nf(1,i*(nodplane+nodbetwplane)+2*nxe+1+j &
      *(3*nxe+2))=0 ;
    ! left element corner node in nodplane
    nf(1,i*(nodplane+nodbetwplane)+1+j &
      *(3*nxe+2))=0 ;
  end do xloop1
  !
  xloop2: do j=1, nze
    ! right element midside node in nodplane
    nf(1,i*(nodplane+nodbetwplane)+j*(3*nxe+2))=0;
    ! left element midside node in nodplane
    nf(1,i*(nodplane+nodbetwplane)+j &
      *(3*nxe+2)-nxe)=0 ;
  end do xloop2

```

```

end do xdirection1
!
xdirection2: do i=1, nye
  xloop3: do j=1, nze+1
    ! Right element mid node in nodbetwplane
    nf(1,i*nodplane+(i-1)*nodbetwplane+j      &
      *(nxe+1))=0 ;
    ! Left element mid node in nodbetwplane
    nf(1,i*nodplane+(i-1)*nodbetwplane+(j-1)  &
      *(nxe+1)+1)=0 ;
  end do xloop3
end do xdirection2
!-----
ydirection1: do i=1, nodplane
  ! front face nodplane
  nf(2,i)=0
  ! back face nodplane
  nf(2,nye*(nodplane+nodbetwplane)+i)=0
end do ydirection1
!-----
xyzdirection1: do i=0, nye
  xyzloop1: do j=1,2*nxe+1
    ! bottom nodes in nodplanes
    nf(1:3,i*(nodplane+nodbetwplane)+nodplane  &
      -(2*nxe+1)+j)=0 ;
  end do xyzloop1
end do xyzdirection1
!
xyzdirection2: do i=1, nye
  xyzloop2: do j=1,nxe+1
    ! bottom nodes in nodbetwplanes
    nf(1:3,i*(nodplane+nodbetwplane)-(nxe+1)+j)=0 ;
  end do xyzloop2
end do xyzdirection2
!
call formnf(nf); neq=maxval(nf);
!
return
end subroutine nfinfo_drained
!-----
subroutine load_raft(nxe,nye,nze,nels,nn,xlen,ylen,nxlmsh,      &
  nylmsh,lval,nlnod,lnn,lnv)
! This subroutine computes nodal loads from uniform pressure.
! nxe,nye,nze = no. of elements in x,y, and z directions
!   nodplane = x-z plane containing corner nodes of
!             20 noded finite element
!   nodbetwplane = x-z plane containing the mid-nodes of
!             20 noded finite element
!   xlen = length of raft in x-direction (m)
!   ylen = breath of raft in y-direction (m)
!   nxlmsh = no. of loaded elements in x-direction
!   nylmsh = no. of loaded elements in y-direction
!   lval = applied uniform pressure (MPa)
!   nlnod = no. of loaded nodes
!   nn = total no. of nodes
!   a = length of each loaded element
!   b = breadth of each loaded element
!   lnn = array of loaded nodes
!   lnv = array of loaded node values
!   sval = the special value (1/12 of element load)
! Modified by: Krishna Bahadur Chaudhary

```

```

!               on: March 31, 2008
implicit none
integer,intent(in):: nxe,nye,nze,nels,nn,nxlmsh,nylmsh, &
    nlnod
real(8),intent(in):: xlen,ylen,lval
integer,intent(out):: lnn(:) ;
real(8),intent(out):: lnv(:)
! local variables
integer:: i,j,k,m,l,nodplane,nodbetwplane
real(8):: sval,a,b
!
nodplane = (2*nxe+1)*(nze+1)+(nxe+1)*nze
nodbetwplane = (nxe+1)*(nze+1)
a = xlen/nxlmsh ; b = ylen/nylmsh ;

! the special value (1/12 of element load)
sval=lval* a*b/12.
!
m = 0;
loop1: do i = 1,nylmsh
    k = (i-1)*(nodplane+nodbetwplane)+1
    ! write(11,'(a,i2,a,i7)')'1st node of No.',2*i-1, &
    ! ' plane :', k
    if(i==1)then !!!
        do j=0,2*nxlmsh
            if(mod(j,2)==0)then
                m=m+1; lnn(m)=k+j ; lnv(m)=-2*sval
            else
                m=m+1; lnn(m)=k+j ; lnv(m)=4*sval
            end if
        end do
    else !!!
        do j=0,2*nxlmsh
            if(mod(j,2)==0)then
                m=m+1; lnn(m)=k+j ; lnv(m)=-4*sval
            else
                m=m+1; lnn(m)=k+j ; lnv(m)=8*sval
            end if
        end do
    end if !!!
    !
    k = i*nodplane+(i-1)*nodbetwplane+1
    !write(11,'(a,i2,a,i7)')'1st node of No.',2*i,' plane :', k
    do j=0,nxlmsh
        if(j==0.or.j==nxlmsh)then
            m=m+1; lnn(m)=k+j; lnv(m)=4*sval
        else
            m=m+1; lnn(m)=k+j; lnv(m)=8*sval
        end if
    end do
end do loop1
!
k = nylmsh*(nodplane+nodbetwplane)+1
! write(11,'(a,i2,a,i7)')'1st node of No.',2*nylmsh+1, &
! ' plane :', k
do j=0,2*nxlmsh
    if(mod(j,2)==0)then
        m=m+1; lnn(m)=k+j ; lnv(m)=-2*sval
    else
        m=m+1; lnn(m)=k+j ; lnv(m)=4*sval
    end if
end do

```

```

        end do
!-----Modify the two sides-----
    do i = 1,nylmsh+1
        l=(i-1)*(3*nxlmsh+2)+1; lnv(1)=-2*sval
        l=(i-1)*(3*nxlmsh+2)+2*nxlmsh+1; lnv(1)=-2*sval
    end do
        lnv(1)=-sval; lnv(2*nxlmsh+1)=-sval
        lnv(nlnod)=-sval; lnv(nlnod-2*nxlmsh)=-sval
!-----
    !
    return
end subroutine load_raft
!-----
subroutine fmelspar(ntot,g,ke,icount,iebea,jebea,ebea,ebeanz)
    ! forming the element level 3 vectors storing the nonzero
    ! entries of upper A matrix
    implicit none
    real(8):: ke(:,,:),ebea(:)
    integer:: i,j,s,t,ntot,ebeanz,g(:),icount(:),iebea(:),jebea(:)
    !-----
        do j=1, ntot
            do i=1, j
                ! forming A (upper triangle column by column)
                if(g(i)/=0.and.g(j)/=0) then
                    if(ke(i,j)/=.0)then
                        s = icount(g(i)) ; t = icount(g(j))
                        if( s <= t )then
                            ebeanz=ebeanz+1 ; iebea(ebeanz) = s
                            jebea(ebeanz)= t ; ebea(ebeanz)=ke(i,j)
                        else
                            ebeanz=ebeanz+1 ; iebea(ebeanz) = t
                            jebea(ebeanz) = s ; ebea(ebeanz)=ke(i,j)
                        end if
                    end if
                end if
            end do
        end do
end subroutine fmelspar
!-----
subroutine countnzblk11 (pneq,neq,jcsca,icsca,csca,pnz)
    ! This subroutine counts the no. of nonzeros in block(1,1),
    ! i.e. P from upper triangular storage of matrix A
    ! A = [P L ; L^t G] ==> count nnz of P
    ! parameters:
    !   On input:
    !       pneq = Pile DOFs
    !       neq = order (size) of the matrix A
    !       jcsca,icsca,csca = CSC storage of the upper part of the
    !           symmetric matrix A dimension:
    !           jcsca(neq+1), icsca(uanz), csca(uanz)
    !           uanz is the no. of nonzeros in upper A
    !   On output:
    !       pnz = returned true no. of nonzeros in P
    !-----
    implicit none
    integer,intent(in):: pneq,neq,jcsca(:),icsca(:)
    real(8),intent(in):: csca(:)
    integer,intent(out):: pnz
    ! local variables
    integer:: i,j,k,l,k1,k2,m

```

```

!-----Extracting Pile Block i.e. block(1,1)-----
m = 0
do j = 1,pneq
  k1 = jcscsca(j) ; k2 = jcscsca(j+1)-1
  ! upper triangular nonzero terms
  do k = k1, k2 ; m = m+1 ; end do
end do
pnz = 2*m - pneq ;
!
return
end subroutine countnzblk11
!-----
subroutine formblk11 (pneq,neq,jcscsca,icsca,csca,icsrp,jcsrp, &
  csrp,pnz)
! This subroutine extracts block(1,1) i.e. P from upper
! triangular storage of matrix A
! A = [P L ; L^t G] ==> extract P
! parameters:
! On input:
!       pneq = Pile DOFs
!       neq = order (size) of the matrix A
!       jcscsca,icsca,csca = CSC storage of the upper part of the
!       symmetric matrix A dimension:
!       jcscsca(neq+1), icsca(uanz), csca(uanz)
!       uanz is the no. of nonzeros in upper A
!       pnz = On input, estimated no. of nonzeros in P
!       On return, true no. of nonzeros in P
! On output:
!       icsrp,jcsrp,csrp = CSR storage of full matrix P (Pile block)
!       For symmetric case CSR and CSC storage
!       are equivalent dimension:
!       icsrp(pneq+1), jcsrp(pnz), csrp(pnz)
!       pnz = returned true no. of nonzeros in P
!-----
implicit none
integer,intent(in):: pneq,neq,jcscsca(:),icsca(:)
real(8),intent(in):: csca(:)
integer,intent(out):: icsrp(:),jcsrp(:)
integer,intent(out):: pnz
real(8),intent(out):: csrp(:)
! local variables
integer:: i,j,k,l,k1,k2,m,ir,ic,epnz
integer,allocatable:: rowp(:),rowg(:)

!-----Extracting Pile Block i.e. block(1,1)-----
epnz = size(jcsrp)
allocate (rowp(epnz))
m = 0
do j = 1,pneq
  k1 = jcscsca(j) ; k2 = jcscsca(j+1)-1
  do k = k1, k2-1 ! off-diagonal terms
    m = m+1 ; rowp(m) = icsca(k); jcsrp(m) = j;
    csrp(m) = csca(k)
    m = m+1 ; rowp(m) = j; jcsrp(m) = icsca(k);
    csrp(m) = csca(k)
  end do
  ! diagonal terms
  m = m+1 ; rowp(m) = icsca(k2); jcsrp(m) = j;
  csrp(m) = csca(k2)
end do

```

```

pnz = m ;
call sort_3(rowp(1:m),jcsrp(1:m),csrp(1:m),m)

icsrp(1) = 1 ; l = 1
do j = 2, m
    if(rowp(j) /= rowp(j-1)) then
        l = l+1; icsrp(l) = j
    end if
end do
icsrp(l+1) = pnz+1

do i = 1, pneq
    k1 = icsrp(i) ; k2 = icsrp(i+1)-1
    j = k2-k1+1
    call sort_2(jcsrp(k1:k2),csrp(k1:k2),j)
end do
deallocate (rowp)
!
return
end subroutine formblk11
!-----
subroutine form_adjacency (neq,icsr,jcsr,adj_row,adj)
! form adjacency structure of the matrix
! Parameters:
! Input:
!     neq   = the number of equations
!     icsr  = row pointer for CSR storage
!     jcsr  = column index for CSR storage
!     adj_row(neq+1) = node pointers for adj
!     adj(nonz)      = adjacency information
!
implicit none
integer,intent(in)::neq,icsr(:),jcsr(:)
integer,intent(out)::adj_row(:),adj(:)
! local variables
integer:: nonz,totnonz,i,j,k,kstrt,kend

nonz = 0
totnonz = 0
do i = 1,neq
    kstrt = icsr(i)
    kend  = icsr(i+1)-1
    do j = kstrt,kend
!         if (jcsr(j) == i) then
!             adj_row(i) = adj_row(i-1)
!         end if

        if (jcsr(j) /= i) then
            adj_row(i) = totnonz + 1
            nonz = nonz + 1
            adj(nonz) = jcsr(j)
        end if
    end do
    totnonz = nonz
end do
adj_row(neq+1) = totnonz + 1
!
return
end subroutine form_adjacency
!-----

```

```

subroutine lsolve_2(n,n1,dal,icsc,jcsc,csca,b, x)
! This subroutine performs forward solve of MSSOR(G),
! that is, (LG+DG)xG = bG
! LG and DG are strict lower and diagonal of G
! A = [P, L; LT, G] --but ONLY SSOR of G (block(2,2))
!           n: dimension of coefficient matrix A;
!           n1: dimension of block P
! icsc,jcsc,csca: CSC storage of upper triangular part
!                 of matrix A;
!           dal: inverse of da(G) (da(G): modified diagonal
!                 for MSSOR);
!           b: it is right hand vector b (size: dim(G,1))
!           x: solution vector (size: dim(G,1));
!-----

implicit none
real(8):: dal(:), csca(:), b(:), x(:), tmp
integer:: i, j, k1, k2, n, n1, icsc(:), jcsc(:)
! ----- forward substitution -----
x(1)=b(1)*dal(1);
do j = n1+2, n
  k1=jcsc(j); k2=jcsc(j+1)-1 ; tmp=.0
  do i=k1, k2-1
    if (icsc(i) > n1) then
      tmp = tmp + csca(i) * x(icsc(i)-n1) ;
    end if
  end do
  tmp = b(j-n1) - tmp
  x(j-n1) = tmp*dal(j-n1) ;
end do
!
return
end subroutine lsolve_2
!-----

subroutine usolve_2(n,n1, dal,icsc,jcsc,csca,b, x)
! This subroutine performs backward solve of MSSOR(G),
! that is, (DG+UG)xG = bG
! DG and UG are diagonal and strict lower of G
! A = [P, L; LT, G] --but ONLY SSOR of G (block(2,2))
!           n: dimension of coefficient matrix A;
!           n1: dimension of block P
! icsc,jcsc,csca: CSC storage of upper triangular part of
!                 matrix A;
!           dal: inverse of da(G) (da(G): modified
!                 diagonal for MSSOR);
!           b: it is right hand vector b (size: dim(G,1))
!           x: solution vector (size: dim(G,1));
!-----

implicit none
real(8):: dal(:), csca(:), b(:), x(:)
real(8), allocatable:: tmp(:)
integer:: j, ic, ir, k, k1, k2, n, n1, icsc(:), jcsc(:)
allocate(tmp(n-n1) )
! ----- backward substitution -----
tmp = b ;
do k = n, n1+2, -1
  ic = k-n1;
  x(ic) = tmp(ic)*dal(ic)
  do j = jcsc(k), jcsc(k+1)-2
    ir = icsc(j);

```

```

        if (ir > n1) then
            tmp(ir-n1) = tmp(ir-n1) - x(ic)*csca(j)
        end if
    end do
end do
x(1) = tmp(1)*da1(1)
!
return
end subroutine usolve_2
!-----
subroutine pallssora22x(n,icsc,jcsc,csca,nsuper,xsuper,      &
    xlindx,lindx,xlnz,lnz,perm1,perm_inv1,n1,n2,icsce, &
    jcsce,csce,md2,md21,dsub,x,y )
! This subroutine performs the preconditioned matrix vector
! multiplication
!      A = [A11   A12;           and  M = [A11           0;
!      A12'  A22]                0   MSSOR(G)]
!
!      y = (inv(M)*A) * x
! Left-right preconditioning: M = L*inv(D)*U
!      (similar to SSOR factorization)
! Eisenstat trick is applied for block (2,2)
!
! In this routine:
!      n: dimension of coefficient matrix A;
!      icsc,jcsc,csca: CSC storage of coefficient matrix A;
!      icsce,jcsce,csce: CSC storage of block E = A(1,2)
!      nsuper,xsuper,xlindx,lindx,xlnz,lnz: Sparse Cholesky
!      factorization of A11
!      perm1,perm_inv1: MMD Permutation and Inverse permutation
!      vectors for A11
!      n1: size of block (1,1)
!      n2: size of block (2,2)
!      md2: modified diagonal for MSSOR of block(2,2)
!      md11: inverse of md2;
!      dsub: d2-2*md2
!      x: input vector
!      y: output vector
!
! Modified by: Krishna B. Chaudhary, NUS
! on: January 3, 2009
!-----
implicit none
integer,intent(in):: n,icsc(:),jcsc(:),nsuper,xsuper(:),      &
    xlindx(:),lindx(:),xlnz(:),perm1(:),      &
    perm_inv1(:),n1,n2,icsce(:),jcsce(:)
real(8),intent(in):: csca(:),lnz(:),csce(:),md2(:),md21(:), &
    dsub(:),x(:)
real(8),intent(out):: y(:)
! local variables
real(8),allocatable:: f(:),tvec1(:),tvec2(:),g(:),q2(:)
allocate( f(n2),tvec1(n1),tvec2(n2),g(n2),q2(n2) )

call usolve_2(n,n1+1,n,md21(n1+1:n),icsc,jcsc,csca,      &
    x(n1+1:n),f)
call cscbx(n2,icsce,jcsce,csce,f,tvec1)
call dvperm(n1,tvec1,perm_inv1)      ! permutation of vector
call fwblkslv(nsuper,xsuper,xlindx,lindx,xlnz,lnz,tvec1)
y(1:n1) = x(1:n1) + tvec1 ;
!-----
tvec1 = x(1:n1)

```

```

call bwblkslv(nsuper,xsuper,xlindx,lindx,xlnz,lnz,tvec1)
! obtain the original solution from permuted solution
call perm_rv( n1, tvec1, perm1 )
call cscbtx(n2,icsce,jcsce,csce,tvec1,tvec2)
g = dsub*f + x(n1+1:n) ;
q2 = g + tvec2 ;
call lsolve_2(n,n1+1,n,md2l(n1+1:n),icsc,jcsc,csca,q2,      &
              y(n1+1:n))
y(n1+1:n) = y(n1+1:n) + f ;
!
return
end subroutine pallssora22x
!-----
subroutine ccrb_2(n,i,icsc,jcsc,csca,nsuper,xsuper,xlindx,      &
                  lindx,xlnz,lnz,permp,perm_invp,np,nh,da,      &
                  dal,s,z,iinc,tol,nrmb,rhs,ic,itors,relres)
! nrmr,
! This subroutine performs convergence check in terms of
! relative residual with initial guess x0 =.0 is chosen,
! Preconditioner M = [Rp'  0;    inv[I  0;    [Rp  0;
!                    0  Lh]    [  Dh]    [ 0  Uh]
! Cholesky factorization of block(1,1) and
! SSOR factorization of block(2,2)
!
! In this subroutine,
! n: i.e. neq - number of total DOFs or equations;
! i: current iteration # ;
! icsc,jcsc,csca:
! CSC storage of upper triangular part of matrix A;
! nsuper,xsuper,xlindx,lindx,xlnz,lnz: Sparse Cholesky
! factorization of P
! permp,perm_invp: MMD Permutation and Inverse permutation
! vectors for P
! np: pile displacement DOFs
! nh: dimension of block H
! (soil + pore pressure DOFs)
! da: modified diagonal for MSSOR preconditioner;
! dal: inverse of da;
! s: preconditioned residual;
! z: "preconditioned" solution;
! iinc: Check convergence every 'iinc' iteration.
! tol: it is the user-defined stopping tolerance;
! nrmb: computed initial residual (b) norm
! multiplied by tol;
! rhs: at input, it is right hand vector b;
! at convergence,it is returned approximate
! solution x;
! ic: = 1 converged (ic is a identifier);
! = 0 doesn't satisfy the convergence criterion;
! iters: returned iteration count when converged;
! nrmb: norm of true residual.
! relres: returned relative residual when converged.
!-----
implicit none
integer,intent (in):: n,i,icsc(:),jcsc(:),nsuper,      &
                      xsuper(:),xlindx(:),lindx(:),      &
                      xlnz(:),permp(:),perm_invp(:),      &
                      np,nh,iinc
integer,intent (inout):: ic ;
integer,intent (out):: iters
real(8),intent (in):: csca(:),lnz(:),da(:),dal(:),s(:),      &

```



```

!*****
!*****
!*****      FWBLKSLV ... BLOCK FOWARD TRIANGULAR SOLUTIONS
!*****
!*****
!
!  PURPOSE:
!    GIVEN THE CHOLESKY FACTORIZATION OF A SPARSE SYMMETRIC
!    POSITIVE DEFINITE MATRIX, THIS SUBROUTINE PERFORMS THE
!    LOWER TRIANGULAR SOLUTION.  IT USES OUTPUT FROM BLKFCT.
!
!  INPUT PARAMETERS:
!    NSUPER          -   NUMBER OF SUPERNODES.
!    XSUPER          -   SUPERNODE PARTITION.
!    (XLINDX,LINDX)  -   ROW INDICES FOR EACH SUPERNODE.
!    (XLNZ,LNZ)     -   CHOLESKY FACTOR.
!
!  UPDATED PARAMETERS:
!    RHS             -   ON INPUT, CONTAINS THE RIGHT HAND
!                      SIDE.  ON OUTPUT, CONTAINS THE
!                      SOLUTION.
!*****
!
!  SUBROUTINE FWBLKSLV (  NSUPER, XSUPER, XLINDX, LINDX , &
!                      XLNZ ,LNZ , RHS )
!*****
!
!  INTEGER          NSUPER
!  INTEGER          LINDX(*)      , XSUPER(*)
!  INTEGER          XLINDX(*)     , XLNZ(*)
!  DOUBLE PRECISION LNZ(*)       , RHS(*)
!
!*****
!
!  INTEGER          FJCOL , I      , IPNT , IX      , &
!                  IXSTOP, IXSTRT, JCOL , JPNT , &
!                  JSUP  , LJCOL
!  DOUBLE PRECISION T
!
!*****
!
!  IF ( NSUPER .LE. 0 ) RETURN
!
!  -----
!  FORWARD SUBSTITUTION ...
!  -----
!
FJCOL = XSUPER(1)
DO 300 JSUP = 1, NSUPER
  LJCOL = XSUPER(JSUP+1) - 1
  IXSTRT = XLNZ(FJCOL)
  JPNT = XLINDX(JSUP)
  DO 200 JCOL = FJCOL, LJCOL
    IXSTOP = XLNZ(JCOL+1) - 1
    T = RHS(JCOL)/LNZ(IXSTRT)
    RHS(JCOL) = T
    IPNT = JPNT + 1
! DIR$ IVDEP
    DO 100 IX = IXSTRT+1, IXSTOP
      I = LINDX(IPNT)
      RHS(I) = RHS(I) - T*LNZ(IX)
      IPNT = IPNT + 1

```

```

100          CONTINUE
              IXSTRT = IXSTOP + 1
              JPNT    = JPNT + 1
200          CONTINUE
              FJCOL = LJCOL + 1
300          CONTINUE
!
          RETURN
      END SUBROUTINE FWBLKSLV
!-----
!  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
!  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
!  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
!
!  Version:          0.3
!  Last modified:    Krishna Bahadur Chaudhary, NUS
!  Authors:         Esmond G. Ng and Barry W. Peyton
!                  (December 27, 1994)
!
!  Mathematical Sciences Section, Oak Ridge National
!  Laboratory
!
!  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
!  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
!  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
!
!  *****          BWBLKSLV ... BLOCK BACKWARD TRIANGULAR SOLUTIONS
!  *****
!
!  PURPOSE:
!  GIVEN THE CHOLESKY FACTORIZATION OF A SPARSE SYMMETRIC
!  POSITIVE DEFINITE MATRIX, THIS SUBROUTINE PERFORMS THE
!  UPPER TRIANGULAR SOLUTION.  IT USES OUTPUT FROM BLKFCT.
!
!  INPUT PARAMETERS:
!      NSUPER          -   NUMBER OF SUPERNODES.
!      XSUPER          -   SUPERNODE PARTITION.
!      (XLINDX,LINDX)  -   ROW INDICES FOR EACH SUPERNODE.
!      (XLNZ,LNZ)     -   CHOLESKY FACTOR.
!
!  UPDATED PARAMETERS:
!      RHS             -   ON INPUT, CONTAINS THE RIGHT HAND
!                          SIDE.  ON OUTPUT, CONTAINS THE
!                          SOLUTION.
!  *****
!
!  SUBROUTINE BWBLKSLV (  NSUPER, XSUPER, XLINDX, LINDX , &
!                        XLNZ   , LNZ   , RHS )
!
!  *****
!
!      INTEGER          NSUPER
!      INTEGER          LINDX(*)          , XSUPER(*)
!      INTEGER          XLINDX(*)         , XLNZ(*)
!      DOUBLE PRECISION LNZ(*)           , RHS(*)
!
!  *****
!
!      INTEGER          FJCOL , I        , IPNT , IX      , &
!                      IXSTOP, IXSTRT, JCOL , JPNT , &
!      DOUBLE PRECISION JSUP , LJCOL
!      T

```

```

!
! *****
!
      IF ( NSUPER .LE. 0 ) RETURN
!
! -----
! BACKWARD SUBSTITUTION ...
! -----
      LJCOL = XSUPER(NSUPER+1) - 1
      DO 600 JSUP = NSUPER, 1, -1
          FJCOL = XSUPER(JSUP)
          IXSTOP = XLNZ(LJCOL+1) - 1
          JPNT = XLINDX(JSUP) + (LJCOL - FJCOL)
          DO 500 JCOL = LJCOL, FJCOL, -1
              IXSTRT = XLNZ(JCOL)
              IPNT = JPNT + 1
              T = RHS(JCOL)
! DIR$          IVDEP
              DO 400 IX = IXSTRT+1, IXSTOP
                  I = LINDX(IPNT)
                  T = T - LNZ(IX)*RHS(I)
                  IPNT = IPNT + 1
400          CONTINUE
              RHS(JCOL) = T/LNZ(IXSTRT)
              IXSTOP = IXSTRT - 1
              JPNT = JPNT - 1
500          CONTINUE
          LJCOL = FJCOL - 1
600      CONTINUE
!
      RETURN
      END SUBROUTINE BWBLKSLV
! -----
! *****
! ----- Sorting Routines -----
! *****
! -----
subroutine sortadd(uanz,arr,brr,crr,ni,nnz)
! For the same arr index, subsort brr, and at the same time,
! crrchanges correspondingly with brr. After this work, adding
! up all crrcomponents with the same (arr, brr) or (brr, arr)
! index, and thezero-value crr entry will be removed. Finally
! forming the Compressed Sparse Row (CSR) format or Compressed
! Sparse Column (CSC) format to overwrite arr,brr,crr.
!      uanz: the nonzero number of arr (or brr, crr).
!      arr,brr,crr: three vectors required to be sorted.
!      ni: = n + 1 (n is dimension of A)
!      nnz: the nonzero number of crr.
! -----
      integer, intent(inout):: arr(:),brr(:)
      integer, intent(in):: uanz,ni
      integer, intent(out):: nnz
      real(8), intent(inout):: crr(:)
      integer:: i,j,k,k1,k2,m
      integer, allocatable:: itep(:)
      real(8):: aa

      allocate (itep(ni))
      call sort_3(arr,brr,crr,uanz) ; ! sorting three vectors
      k=1; itep(1)=1
      do i=2, uanz

```

```

        if(arr(i)/=arr(i-1)) then
            k=k+1 ; itep(k)=i
        end if
    end do
    itep(k+1)=uanz+1
!-----
do i=1, k
    k1=itep(i); k2=itep(i+1)-1
    j=k2-k1+1
        ! call sort_2(brr(k1:k2),crr(k1:k2),j)

        ! sub-brr sorting by Insertion sort if j <= 16.
        if(j<=16) then
            call inssort_2(brr(k1:k2),crr(k1:k2),j)
        else ! quick sorting when j is larger (>16).
            call sort_2(brr(k1:k2),crr(k1:k2),j)
        end if
    end do
!-----
    m = 0 ;
do i=1, k
    k1=itep(i); k2=itep(i+1)-1 ; m=m+1;
    arr(i) = m ; brr(m) = brr(k1) ; aa = .0
do j=k1, k2-1
    aa = aa + crr(j) ;
    if(brr(j+1)/=brr(j) ) then
        if(aa /=.0) then
            crr(m) = aa
            m=m+1 ;
            brr(m)= brr(j+1)
            aa = .0
        else ! aa is removed when it is zero.
            brr(m)= brr(j+1)
        end if
    end if
end do
    crr(m) = aa + crr(k2)
    if(crr(m)==.0) m=m-1
end do
arr(k+1)=m+1; nnz=m
!
return
end subroutine sortadd
!-----
subroutine sort_3(arr,brr,crr,uanz)
! This subroutine - sorts arr into ascending order, brr and
! crr change correspondingly by using Quicksort method.
! quicksort chooses a "pivot" in the set, and explores the
! array from both ends, looking for a value > pivot with the
! increasing index (left to right), for a value <= pivot with
! the decreasing index (right to left), and swapping them when
! it has found one of each. ! The array is then subdivided in
! 2 ([3]) subsets: { values <= pivot} {pivot} {values > pivot}.
! One then call recursively the program to sort each subset.
! When the size of the subarray is small enough, one uses a
! selection sort that is faster for very small sets. Sorting
! an array arr(1:n) into ascending order with quicksort, while
! making the corresponding rearrangements of arrays brr(1:n)
! and crr(1:n).(Revised from ORDERPACK codes)
! uanz: the nonzero number of arr (or brr, crr).
! reference: Chen (2005) and Num. Recipes in Fortran (2002)

```

```

!-----
implicit none
real(8), intent(inout):: crr(:)
integer, intent(inout):: arr(:),brr(:)
integer:: uanz
!
! uanz = size(arr) ;
call quicksort_3(arr,brr,crr,1, uanz) ;
call inssort_3(arr,brr,crr,uanz) ;
!
return
end subroutine sort_3
!-----
Recursive subroutine quicksort_3(arr,brr,crr,first,last)
! This subroutine sorts arr from first to last in ascending
! order
implicit none
integer, intent(inout):: arr(:),brr(:)
real(8), intent(inout):: crr(:)
integer, intent(in) :: first, last
integer :: left,right,low,high,mid,xpivot,nins = 16
! Max data for insertion sort
! for < 16 data, insertion sort will sort
them
low = first
high = last

if ((high - low) > nins) Then
mid = (low + high) / 2
! One chooses a pivot, median of 1st, last, and
middle
! values
if (arr(mid) < arr(low)) Then
call swap_i(arr(mid),arr(low))
call swap_i(brr(mid),brr(low))
call swap_r(crr(mid),crr(low))
end if
!
if (arr(mid) > arr(high)) Then
call swap_i(arr(mid),arr(high))
call swap_i(brr(mid),brr(high))
call swap_r(crr(mid),crr(high))
!
if (arr(mid) < arr(low)) Then
call swap_i(arr(mid),arr(low))
call swap_i(brr(mid),brr(low))
call swap_r(crr(mid),crr(low))
end if
end if
!
xpivot = arr(mid)
!
! One exchanges values to put those > pivot in the
! end and those <= pivot at the beginning
!
left = low
right = high
! Repeat the follwoing while left and right haven't
! met
outer: do !-----
if (left >= right) exit

```

```

        ! scan right to left to find element < pivot
        do
            if (arr(right) <= xpivot) exit
            right = right -1
        end do

        ! scan left to right to find element > pivot
        do
            if(arr(left) > xpivot) exit
            left = left + 1
            if (left >= right) exit
        ! the last value < pivot is always less than
        ! left-1
        end do

        ! if left and right haven't met, exchange the
        ! items
        if (left < right) then
            call swap_i(arr(left),arr(right))
            call swap_i(brr(left),brr(right))
            call swap_r(crr(left),crr(right))
        end if
    end do outer      !-----
    !
    ! One now sorts each of the two sub-intervals
    !
    call quicksort_3(arr,brr,crr,first,left-1)
    call quicksort_3(arr,brr,crr,right,last)
end if
!
return
end subroutine quicksort_3
!-----
subroutine inssort_3(arr,brr,crr,uanz)
! This subroutine sorts arr into increasing order
! (Insertion sort)
! reference: Numerical Reciepts '90 book & Chen (2005)
!-----
integer,intent(inout) :: arr(:),brr(:)
integer, intent(in):: uanz
integer :: left,right,xswap,yswap
real(8),intent(inout) :: crr(:)
real(8) :: zswap
!
do right = 2, uanz      ! Pick out each element in turn
    xswap = arr(right);    yswap = brr(right);
    zswap = crr(right);
    ! Look for the place to insert it
    do left = right-1, 1,-1
        if (arr(left) <= xswap) exit
        arr (left+1) = arr (left)
        brr (left+1) = brr (left)
        crr (left+1) = crr (left)
    end do
    ! insert it
    arr(left+1) = xswap;  brr(left+1) = yswap;
    crr(left+1) = zswap
end do
!
return
end subroutine inssort_3

```

```

!-----
!-----
subroutine sort_2(brr,crr,n)
! This subroutine - sorts brr into ascending order and crr
! changes correspondingly by using Quicksort method.
! quicksort chooses a "pivot" in the set, and explores the
! array from both ends, looking for a value > pivot with the
! increasing index (left to right), for a value <= pivot with
! the decreasing index (right to left), and swapping them when
! it has found one of each. ! The array is then subdivided in
! 2 ([3]) subsets: { values <= pivot} {pivot} {values > pivot}.
! One then call recursively the program to sort each subset.
! When the size of the subarray is small enough (say < 16,
! in this case), one uses a selection sort that is faster for
! very small sets.(Revised from ORDERPACK codes)
! n : size of brr (or crr) to be sorted
!-----

    implicit none
    real(8), intent(inout):: crr(:)
    integer, intent(inout):: brr(:)
    integer, intent(in):: n
    !
    call quicksort_2(brr,crr,1,n) ;
    call insort_2(brr,crr,n) ;
    !
    return
end subroutine sort_2
!-----

Recursive subroutine quicksort_2(brr,crr,first,last)
    implicit none
    integer, intent(inout):: brr(:)
    real(8), intent(inout):: crr(:)
    integer, intent (in) :: first, last
    integer :: left,right,low,high,mid,xpivot,nins = 16
        ! Max data for insertion sort
        ! for < 16 data, insertion sort will sort them
    low = first
    high = last

    if ((high - low) > nins) Then
        mid = (low + high) / 2
        ! One chooses a pivot, median of 1st, last,
        ! and middle values
        if (brr(mid) < brr(low)) Then
            call swap_i(brr(mid),brr(low))
            call swap_r(crr(mid),crr(low))
        end if
        !
        if (brr(mid) > brr(high)) Then
            call swap_i(brr(mid),brr(high))
            call swap_r(crr(mid),crr(high))
            !
            if (brr(mid) < brr(low)) Then
                call swap_i(brr(mid),brr(low))
                call swap_r(crr(mid),crr(low))
            end if
        end if
        !
        xpivot = brr(mid)
        !
        ! One exchanges values to put those > pivot in the

```

```

! end and those <= pivot at the beginning
!
left = low
right = high
! Repeat the follwoing while left and right haven't
! met
outer: do !-----
    if (left >= right) exit
    ! scan right to left to find element < pivot
    do
        if (brr(right) <= xpivot) exit
        right = right -1
    end do

    ! scan left to right to find element > pivot
    do
        if(brr(left) > xpivot) exit
        left = left + 1
        if (left >= right) exit
        ! the last value < pivot is always less than
        ! left-1
    end do

    ! if left and right haven't met, exchange the
    ! items
    if (left < right) then
        call swap_i(brr(left),brr(right))
        call swap_r(crr(left),crr(right))
    end if
end do outer !-----
!
! One now sorts each of the two sub-intervals
!
call quicksort_2(brr,crr,first,left-1)
call quicksort_2(brr,crr,right,last)
end if
!
return
end subroutine quicksort_2
!-----
subroutine inssort_2(brr,crr,n)
! This subroutine sorts brr into increasing order
!(Insertion sort)
! and at the same time, crr changes correnspondingly with brr
! Insertion sort is considered faster for small size of arrays
!-----
integer, intent(inout) :: brr(:)
integer, intent(in):: n
integer :: left,right,yswap
real(8), intent(inout) :: crr(:)
real(8) :: zswap
!
do right = 2, n ! Pick out each element in turn
    yswap = brr(right); zswap = crr(right);
    ! Look for the place to insert it
    do left = right-1, 1,-1
        if (brr(left) <= yswap) exit
        brr (left+1) = brr (left)
        crr (left+1) = crr (left)
    end do
    ! insert it

```

```

        brr(left+1) = yswap;  crr(left+1) = zswap
    end do
    !
    return
end subroutine inssort_2
!-----
subroutine swap_i(a,b)
    ! swap the contents of a and b
    ! reference: Numerical Recipes in Fortran 90 (2002),p.1366
    integer, intent(inout):: a,b
    integer:: temp
    temp = a; a = b ; b = temp
    !
    return
end subroutine swap_i
!-----
subroutine swap_r(a,b)
    ! swap the contents of a and b
    ! reference: Numerical Recipes in Fortran 90 (2002),p.1367
    real(8), intent(inout):: a,b
    real(8):: temp
    temp = a; a = b ; b = temp
    !
    return
end subroutine swap_r
!-----
    end subroutine dperm
    subroutine dvperm (n, x, perm)
    integer:: n, perm(:)
    real(8):: x(:)
!-----
! this subroutine performs an in-place permutation of a real
! vector x according to the permutation array perm(*),
! i.e., on return, the vector x satisfies,
!
!     x(perm(j)) := x(j), j=1,2,..., n
!
!-----
! on entry:
!-----
! n    = length of vector x.
! perm    = integer array of length n containing the
permutation
!         array.
! x    = input vector
!
! on return:
!-----
! x    = vector x permuted according to x(perm(*)) := x(*)
!
!-----
!
        Y. Saad, Sep. 21 1989
!-----
! local variables
    real*8 tmp, tmp1
!
    init      = 1
    tmp      = x(init)
    ii       = perm(init)
    perm(init) = -perm(init)
    k        = 0

```

```

!
! loop
!
6      k = k+1
!
! save the chased element --
!
      tmp1      = x(ii)
      x(ii)     = tmp
      next      = perm(ii)
      if (next .lt. 0 ) goto 65
!
! test for end
!
      if (k .gt. n) goto 101
      tmp       = tmp1
      perm(ii)  = - perm(ii)
      ii        = next
!
! end loop
!
      goto 6
!
! reinitilaize cycle --
!
65     init      = init+1
      if (init .gt. n) goto 101
      if (perm(init) .lt. 0) goto 65
      tmp       = x(init)
      ii        = perm(init)
      perm(init)=-perm(init)
      goto 6
!
101    continue
      do 200 j=1, n
          perm(j) = -perm(j)
200    continue
!
      return
!-----end-of-dvperm-----
!-----
subroutine perm_rv ( n, rhs, perm )
! This subroutine is from sparspak.f90
!
!! PERM_RV undoes the permutation of the right hand side.
!
! Discussion:
!
!     This routine should be called once the linear system has
!     been solved and the solution returned in RHS. The
!     routine then undoes the permutation of RHS, restoring the
!     original ordering. To do this, it needs the PERM vector
!     which defined the reordering used by the solver.
!
! Modified:
!
!     24 February 2007
!
! Author:
!
!     Alan George, Joseph Liu

```

```

!
! Reference:
!
! Alan George, Joseph Liu,
! Computer Solution of Large Sparse Positive Definite Systems,
! Prentice Hall, 1981,
! ISBN: 0131652745,
! LC: QA188.G46.
!
! Parameters:
!
!     Input, integer N, the number of equations.
!
!     Input/output, real ( kind = 8 ) RHS(N).
!     On input, the solution of the permuted linear system.
!     On output, the solution of the original linear system.
!
!     Input, integer PERM(N), the permutation information.
!     PERM(I) = K means that the K-th equation and variable
!     in the original ordering became the I-th equation and
!     variable in the reordering.
!
implicit none

integer::n,iput,istart,perm(:)
real(8):: pull,put,rhs(:)
! integer n

! integer iput
! integer istart
! integer perm(n)
! real ( kind = 8 ) pull
! real ( kind = 8 ) put
! real ( kind = 8 ) rhs(n)
!
! Mark PERM with negative signs which will be removed
! as each permuted element is restored to its rightful place
!
perm(1:n) = -perm(1:n)
!
! Search for the next element of PERM which is the first
! element of a permutation cycle.
!
istart = 0

20 continue

do

    istart = istart + 1

    if ( n < istart ) then
        return
    end if

    if ( 0 < perm(istart) ) then
        cycle
    end if

    if ( abs ( perm(istart) ) /= istart ) then
        exit
    end if
end do

```

```

        end if

        perm(istart) = abs ( perm(istart) )

    end do

!
!   Begin a cycle.
!
    perm(istart) = abs ( perm(istart) )
    iput = istart
    pull = rhs(iput)

    do

        iput = abs ( perm(iput) )
        put = rhs(iput)
        rhs(iput) = pull
        pull = put

        if ( 0 < perm(iput) ) then
            go to 20
        end if

        perm(iput) = abs ( perm(iput) )

    end do

end subroutine perm_rv
!*****
!----- Preconditioned Iterative Solvers -----
!*****
!-----
subroutine sbd1pcg(n,jcsca,icsca,csca,diagg,nsuper,xsuper,      &
                  xlindx,lindx,xlnz,lnz,permp,perm_invp,rhs, &
                  maxit,tol,itors,relres)

! This subroutine uses diagonal block PCG to solve Ax=b
! linear system with a right diagonal preconditioner.
!           A = [P L ; L' G]
!           M^(-1) = [ P^-1      0;
!                     0      (diag(G))^-1 ]
!
! Parameters:
!   On input:
!       n: dimension of coefficient matrix A;
!       jcsca,icsca,
!           csca: CSC storage of coefficient matrix A;
!           diagp: diagonal of block(1,1) in inverted form
!       nsuper,xsuper,
!       xlindx,lindx,
!           xlnz,lnz: Sparse Cholesky factorization of
block(2,2),
!       permg: MMD permutation vector
!       perm_invp: Inverse of MMD permutation vector
!       rhs: at input, it is right hand vector b;
!            at output,it is returned approximate
!            solution x;
!       maxit: user-defined maximum iteration count;
!       tol: it is the user-defined stopping tolerance;
!            relative residual norm criterion (x0=.0)
!       for convergence

```

```

!           ic: identifier of convergence;
!           = 1, solver converged;
!           = 0, not converge.
!   On output:
!           rhs: approximate solution x
!           iters: the iterative count when PCG converges;
!           relres: the relative residual when PCG converges.
!   Reference: Book: Van der Vorst (2003) or Template (1994)
!-----
implicit none
integer,intent(in):: n,jcsca(:),icsca(:),nsuper,xsuper(:), &
                    xlindx(:),lindx(:),xlnz(:),permp(:), &
                    perm_invp(:),maxit

real(8),intent(in):: csca(:),diagg(:),lnz(:),tol
integer,intent(out):: iters
real(8),intent(out):: relres
real(8),intent(inout):: rhs(:)
! local variables
integer:: i,ic,np,ng
real(8):: nrmb,nrmr,rho,rho0,alpha,beta
real(8),allocatable:: r(:),z(:),p(:),q(:),x(:),xold(:), &
                    tvec(:)
allocate (r(n),z(n),p(n),q(n),x(n),xold(n) )

ng = size(diagg) ;    ! size of block(2,2)
np = n-ng           ! size of block(1,1)
allocate (tvec(np))

!   b -- is RHS vector when inputting, while it is the
!   Solution Vector when returning.
x = .0 ; ! x0=0 is the initial solution guess
r = rhs ;
nrmb=sqrt(dot_product(rhs, rhs))*tol;
ic=0 ; xold = x ;

pcg_iter: do i=1, maxit
!-----preconditioning step----- z = pr*r -----
tvec = r(1:np)
! permute the rhs according to MMD permutation
call dvperm(np,tvec,perm_invp)
call blkslv(nsuper,xsuper,xlindx,lindx,xlnz,lnz,tvec)
! obtain the original solution from permuted solution
call perm_rv ( np, tvec, permp )
z(1:np) = tvec
!-----
z(np+1:) = diagg*r(np+1:) ;
!-----
rho = dot_product(r, z) ;
if ( i > 1 )then                ! direction vector
    beta = rho/rho0;
    p = z + beta*p;
else
    p = z;
end if

!-----q=Ap, Matrix-vector product-----
call cscax(icsca,jcsca,csca,p,q)
!-----
alpha = rho/dot_product(p, q) ;
x = x + alpha * p ;

```

```

r = r - alpha * q ;
nrmr=sqrt(dot_product(r, r)) ;

call pccrb(n,i,r,x,rhs,tol,nrmb,ic,itors, nrmr,relres)

if(ic==1)return ;          ! PCG converged
rho0 = rho
end do pcg_iter
  write(11,'(a)') ' '
  write(11,'(a)')      &
    ' PCG does not converge to user-defined tolerance. '
  write(11,'(a,i7)') ' at iteration =',maxit
  write(11,'(a)')      &
    '***** '
  write(11,'(a)')      &
    ' PCG does not converge to user-defined tolerance. '
  relres=nrmr*tol/nrmb ;  iters = maxit ;  rhs=x
return
end subroutine sbd1pcg
!-----
subroutine sbd2pcg(n,jcsca,icsca,csca,diagg,nsuper,xsuper,      &
                  xlindx,lindx,xlnz,lnz,permp,perm_invp,rhs, &
                  maxit,tol,itors,relres)

! This subroutine uses diagonal block PCG to solve Ax=b
! linear system with a right diagonal preconditioner.
!           A = [P L ; L' G]
!           M^(-1) = [ P^-1          0;
!                     0      SSOR(G)^-1 ]
!
! Parameters:
!   On input:
!     n: dimension of coefficient matrix A;
!     jcsca,icsca,
!       csca: CSC storage of coefficient matrix A;
!       diagg: diagonal of block(2,2) in inverted form
!     nsuper,xsuper,
!     xlindx,lindx,
!       xlnz,lnz: Sparse Cholesky factorization of block(1,1)
!       permp: MMD permutation vector
!     perm_invp: Inverse of MMD permutation vector
!     rhs: at input, it is right hand vector b;
!          at output,it is returned approximate
!          solution x;
!     maxit: user-defined maximum iteration count;
!     tol: it is the user-defined stopping tolerance;
!          relative residual norm criterion (x0=.0)
!          for convergence
!     ic: indentifier of convergence;
!         = 1, solver converged;
!         = 0, not converge.
!   On output:
!     rhs: approximate solution x
!     iters: the iterative count when PCG converges;
!     relres: the relative residual when PCG converges.
! Reference: Book: Van der Vorst (2003) or Template (1994)
!-----
implicit none
integer,intent(in):: n,jcsca(:),icsca(:),nsuper,xsuper(:), &
                    xlindx(:),lindx(:),xlnz(:),permp(:), &

```

```

                                perm_invp(:),maxit

real(8),intent(in):: csca(:),diagg(:),lnz(:),tol
integer,intent(out):: iters
real(8),intent(out):: relres
real(8),intent(inout):: rhs(:)
! local variables
integer:: i,j,ic,np,ng
real(8):: nrmb,nrmr,rho,rho0,alpha,beta
real(8),allocatable:: r(:),z(:),p(:),q(:),x(:),xold(:),      &
                                tvec1(:),tvec2(:),tvec3(:)
allocate (r(n),z(n),p(n),q(n),x(n),xold(n) )

ng = size(diagg) ;      ! size of block(2,2)
np = n-ng              ! size of block(1,1)
allocate (tvec1(np),tvec2(ng),tvec3(ng))

! b -- is RHS vector when inputting, while it is the
! Solution Vector when returning.
x = .0 ; r = rhs ; ! x0=0 is the initial solution guess
nrmb=sqrt(dot_product(rhs, rhs))*tol;
ic=0 ; xold = x ;

pcg_iter: do i=1, maxit
!-----preconditioning step----- z = pr*r -----
tvec1 = r(1:np)
! permute the rhs according to MMD permutation
call dvperm(np,tvec1,perm_invp)
call blkslv(nsuper,xsuper,xlindx,lindx,xlnz,lnz,tvec1)
! obtain the original solution from permuted solution
call perm_rv ( np, tvec1, permp )
z(1:np) = tvec1
!-----
tvec2 = r(np+1:n)
call lsolve_2(n,np, diagg,icsca,jcsca,csca,tvec2,tvec3);
do j = 1,ng ; tvec3(j) = tvec3(j)/diagg(j) ; end do
call usolve_2(n,np,diagg,icsca,jcsca,csca,tvec3,z(np+1:n));

! z(np+1:) = diagg*r(np+1:) ;
!-----
rho = dot_product(r, z) ;
if ( i > 1 )then                ! direction vector
    beta = rho/rho0;
    p = z + beta*p;
else
    p = z;
end if

!-----q=Ap, Matrix-vector product-----
call cscax(icsca,jcsca,csca,p,q)
!-----
alpha = rho/dot_product(p, q) ;
x = x + alpha * p ;
r = r - alpha * q ;
nrmr=sqrt(dot_product(r, r)) ;

call pccrb(n,i,r,x,rhs,tol,nrmb,ic,iters, nrmr,relres)

if(ic==1)return ;              ! PCG converged
rho0 = rho
end do pcg_iter

```

```

        write(11,'(a)') ' '
        write(11,'(a)')      &
        ' PCG does not converge to user-defined tolerance. '
        write(11,'(a,i7)') ' at iteration =',maxit
        write(11,'(a)')      &
        '*****'
        write(11,'(a)')      &
        ' PCG does not converge to user-defined tolerance. '
        relres=nrmr*tol/nrmb ;  iters = maxit ;  rhs=x
    return
end subroutine sbd2pcg
!-----
subroutine mlsqmr(n,icsc,jcsc,csca,nsuper,xsuper,xlindx,      &
                 lindx,xlnz,lnz,permp,perm_invp,invgj,rhs,    &
                 maxit,tol,icc,qmriters,relres)
! This subroutine uses SQMR to solve Ax=b linear system
! with a right diagonal preconditioner.
! In this routine:
!       n: dimension of coefficient matrix A;
! icsc,jcsc,csca: CSC storage of coefficient matrix A;
!       pr: right preconditioner
!             (which is inverted at input);
!       rhs: at input, it is right hand vector b;
!             at output,
!             it is returned approximate solution x;
!       maxit: user-defined maximum iteration count;
!       tol: it is the user-defined stopping tolerance;
!       icc: choice for convergence criterion;
!             = 1, relative improvement norm criterion
!             = 2, relative residual norm criterion
!       ic: identifier of convergence;
!             = 1, solver converged;
!             = 0, not converge.
!       qmriters: the iterative count when SQMR converges;
!       relres: the relative residual when SQMR converges.
implicit none
integer,intent(in)::n,maxit,icc,icsc(:),jcsc(:),nsuper,      &
                    xsuper(:),xlindx(:),lindx(:),xlnz(:),    &
                    permp(:),perm_invp(:)
real(8),intent(in)::tol,csca(:),lnz(:),invgj(:)

        ! pr(:)
integer,intent(out):: qmriters
real(8),intent(out):: relres ;
real(8),intent(inout):: rhs(:)
integer:: i,ic,np
real(8):: tao,theta0,rho0,rho,nrmb,nrmr,sigma,alpha,beta,    &
        theta,cj
real(8),allocatable::x(:),xold(:),r(:),t(:),q(:),d(:),u(:), &
        z(:)
allocate(x(n),xold(n),r(n),t(n),q(n),d(n),u(n) )

np = size(permp) ;
allocate ( z(np) )
!----- Initial vectors of SQMR iterations -----
x=.0                                ! assumed initial guess
r=rhs;
t=r;                                ! left preconditioning
!----right preconditioning---- q = M2^-1 * t -----
z = t(1:np) ; ! z is a temporary vector
        ! permute the rhs according to MMD permutation

```

```

call dvperm(np,z,perm_invp)
call blkslv(nsuper,xsuper,xlindx,lindx,xlnz,lnz,z)
! obtain the original solution from permuted solution
call perm_rv ( np, z, permp )
q(1:np) = z
!-----
q(np+1:) = invgj(np+1:) * t(np+1:)
!-----

tao=sqrt(dot_product(t, t ));
theta0=.0;
rho0=dot_product(r,q);
nrmb=sqrt(dot_product(r, r))*tol;
d=.0; ic=0 ; xold = x ;

!----- Sart SQMR iterations -----
iteration: do i=1, maxit
! t=A*q, Matrix-vector product.
call cscax(icsc,jcsc,csc,q,t)
sigma=dot_product(q,t);
alpha=rho0/sigma ;
r = r-alpha*t ;
t= r; ! left preconditioning
theta=sqrt(dot_product(t, t ))/tao ;
cj=1./sqrt(1+theta*theta);
tao=tao*theta*cj;
d=(cj*theta0)**2*d+(cj*cj)*alpha*q ;
x = x + d;
nrmr=sqrt(dot_product(r, r)) ;
select case (icc)
case (1)
call pccri(n,i,xold,x,rhs,tol,ic,qmriters,relres)
xold = x ;
case (2)
call pccrb(n,i,r,x,rhs,tol,nrmb,ic,qmriters,nrmr, &
relres)
end select
if(ic==1) return ; ! SQMR converged
!u=pr*t ; ! right preconditioning
!----- right preconditioning u = M2^-1 * t -----
z = t(1:np) ; ! z is a temporary vector
call dvperm(np,z,perm_invp)
call blkslv(nsuper,xsuper,xlindx,lindx,xlnz,lnz,z)
! obtain the original solution from permuted solution
call perm_rv ( np, z, permp )
u(1:np) = z
!-----
u(np+1:) = invgj(np+1:) * t(np+1:)
!-----

rho=dot_product(r,u);
beta=rho/rho0; q = u + beta*q ;
!
rho0=rho; theta0=theta;
end do iteration
write( *, '(a)' ) ' '
write( *, '(a)' ) &
'SQMR does not converge to user-defined tolerance. '
write( *, '(a,i7)' ) ' at iteration =' ,maxit
write(11, '(a)' ) &
'***** '
write(11, '(a)' ) &
'SQMR does not converge to user-defined tolerance. '
relres=nrmr*tol/nrmb ; qmriters = maxit ; rhs=x

```

```

!
! return
end subroutine m1sqmr
!-----
subroutine m2sqmr(n, icsc, jcsc, csca, nsuper, xsuper, xlindx,      &
                 lindx, xlnz, lnz, permp, perm_invp, np, sn, nh,    &
                 icsce, jcsce, csce, d, da, dal, rhs, maxit, tol,   &
                 iinc, qmriters, relres)
! This subroutine uses SQMR to solve Ax=b linear system with
! a left-right blockdiagonal preconditioner.
!
!   A = [Pile block           = [ P       L       B1 ;
!       Soil Block           L'       G       B2 ;
!       Fluid block]         B1'      B2'      -C ]
!
!   Preconditioner M = [P           0;
!                     0   MSSOR(H)]
!
!       where P = pile block
!       MSSOR(H) = Modified SSOR for soil-fluid block H
!       H = [G       B2;
!           B2'      -C]
!           diag(C) is replaced by alpha*diag(S)
!   S = C + B1' * inv(diag(P)) * B1 + B2' * inv(diag(G)) * B2
!       = approximate Schur complement
!
! In this routine:
!       n: dimension of coefficient matrix A;
!       icsc, jcsc, csca: CSC storage of coefficient matrix A;
!       icsce, jcsce, csce: CSC storage of block E, E = [L B1]
!       nsuper, xsuper, xlindx, lindx, xlnz, lnz: Sparse Cholesky
!       factorization of P
!       permp, perm_invp: MMD Permutation and Inverse
!                       permutation vectors for P
!       np: pile displacement DOFs
!       sn: total (pile + soil) displacement DOFs
!       nh: dimension of block H
!           (soil + pore pressure DOFs)
!       d: original diagonal of H (d(1:np) = 1.0);
!       da: modified diagonal for MSSOR preconditioner;
!       dal: inverse of da;
!       rhs: at input, it is right hand vector b;
!           at output,
!           it is returned approximate solution x;
!       maxit: user-defined maximum iteration count;
!       tol: it is the user-defined stopping tolerance;
!       ic: identifier of convergence;
!           = 1, solver converged;
!           = 0, not converge.
!       qmriters: the iterative count when SQMR converges;
!       relres: the relative residual when SQMR converges.
!-----
implicit none
integer, intent(in) :: n, maxit, icsc(:), jcsc(:), nsuper,      &
                      xsuper(:), xlindx(:), lindx(:), xlnz(:),  &
                      permp(:), perm_invp(:), np, sn, nh, icsce(:), &
                      jcsce(:), iinc
integer, intent(out) :: qmriters
real(8), intent(in) :: tol, csca(:), lnz(:), csce(:), d(:), da(:), &
                      dal(:)
real(8), intent(out) :: relres ;

```

```

real(8),intent(inout):: rhs(:)
! local variables
integer:: i,ic
real(8):: tao,theta0,rho0,rho,nrmb,nrmr,sigma,alpha,beta, &
    theta,cj
real(8),allocatable::x(:),xold(:),r(:),s(:),v(:),w(:), &
    t(:),q(:),c(:),z(:),tvec1(:),tvec2(:),tvec3(:), &
    f(:),invdiagh(:),d2(:)
allocate( x(n),xold(n),r(n),s(n),v(n),w(n),t(n),q(n),c(n), &
    z(n),tvec1(np),tvec2(nh),tvec3(nh),f(nh), &
    invdiagh(nh),d2(nh) )

!----- Initial vectors of SQMR iterations -----
x = .0 ; r = rhs; z = .0 ! assumed initial guess
tvec1 = r(1:np);
! permute the rhs(1:np) according to MMD permutation
call dvperm(np,tvec1,perm_invp)
call
fwblkslv(nsuper,xsuper,xlindx,lindx,xlnz,lnz,tvec1)
s(1:np) = tvec1 ; ! s is left preconditioned residual
!-----
call lsolve_2(n,np+1,n,dal(np+1:n),icsc,jcsc,csca, &
    r(np+1:n),s(np+1:n))
v = s ; w(1:np) = v(1:np) ;
w(np+1:n) = da(np+1:n) * v(np+1:n) ;
!-----
tao = dot_product(s, s) ; theta0 = .0;
rho0 = dot_product(s,w);
! preconditioned residual
nrmb = dsqrt(dot_product(s, s))*tol;
! nrmb = dsqrt(dot_product(r, r))*tol; ! true residual
c = .0; ic = 0 ; xold = x ;
d2 = d(np+1:n) - 2.0*da(np+1:n)
!----- Sart SQMR iterations -----
iteration: do i = 1, maxit
! t=A~*v, (w = da*v) Preconditioned matrix-vector
! multiplication
call pallssora22x(n,icsc,jcsc,csca,nsuper,xsuper,xlindx, &
    lindx,xlnz,lnz,permp,perm_invp,np,nh, &
    icsce,jcsce,csce,da,dal,d2,w,t )

sigma=dot_product(w,t);
if(sigma==.0) then
write(11,*) 'SQMR stops due to Sigma=0 '; stop
end if
alpha = rho0/sigma ; s = s-alpha*t ;
theta = dot_product(s, s)/tao ; cj = 1./(1.+theta);
tao = tao*theta*cj;
c = cj*(theta0*c + alpha*v) ;
z = z + c;
! Check for convergence
call ccrb_2(n,i,icsc,jcsc,csca,nsuper,xsuper,xlindx, &
    lindx,xlnz,lnz,permp,perm_invp,np,nh,da,dal, &
    s,z,iinc,tol,nrmb,rhs,ic,qmriters,relres)
if(ic==1) return ; ! SQMR converged

q(1:np) = s(1:np) ; q(np+1:n) = da(np+1:n) * s(np+1:n) ;
rho = dot_product(s,q); beta = rho/rho0;
v = s + beta*v ;
w(1:np) = v(1:np) ; w(np+1:n) = da(np+1:n) * v(np+1:n) ;
!

```

```

rho0 = rho; theta0 = theta;
end do iteration
  write( *, '(a)') ' '
  write( *, '(a)') &
    'SQMR does not converge to user-defined tolerance. '
  write( *, '(a,i7)') ' at iteration =' ,maxit
  write(11, '(a)') &
    '***** '
  write(11, '(a)') &
    'SQMR does not converge to user-defined tolerance. '
  qmriters = maxit ;
  tvec1 = z(1:np);
  call bwblkslv(nsuper,xsuper,xlindx,lindx,xlnz,lnz,tvec1)
  ! obtain the original solution from permuted solution
  call perm_rv ( np, tvec1, permp )
  rhs(1:np) = tvec1
  !-----
  z(np+1:n) = da(np+1:n) * z(np+1:n) ;
  call usolve_2(n,np+1,n,dal(np+1:n),icsc,jcsc,cscs, &
    z(np+1:n),rhs(np+1:n))
  !
  return
end subroutine m2sqmr
!-----
end module sparselib_v3

```

*(blank)*

## APPENDIX F

### USER DEFINED SOLVER IN GeoFEA

#### F.1. Source code for user defined solver

This is the source code of user defined solver (USOLV.F90). Some modifications are made to properly run the original sample USOLV.F90. These are as follows (Sequence numbers follows the box numbers indicated in the source code):

1. Changes in INTENT (IN and INOUT) declaration of the variables in the original subroutine UDSOL. The variables that are modified within the subroutine are declared as INTENT (INOUT).

```
Integer(4), Intent (INOUT) :: IHND_PIC, NCORR (NTPE, NEL),      &  
    TF (7, 50000), IPOS (NDF), IRPOS (NDF),                    &  
    IYIELD_CODE (NIP, NEL), NPLax
```

```
Real(8), Intent (INOUT) :: DTIMEI, FRACLD, TOLD, TOLT,          &  
    XYZ (NDIM, NN), DA (NDF), P (NDF), PCOR (NDF),            &  
    REAC (NDF), STR (NVRN, NIP, NEL),                          &  
    VARINT (NVRN, NIP, NEL), W (120), PORINS (NN),             &  
    PR (NPR, NMT), L (4, 120), DXYT (7, 50000)
```

2. Modification in the local variables as per necessary.
3. A new subroutine SPSNEQ is embedded. This subroutine counts the free degrees of freedom (DOFs) excluding fixities with zero prescribed values. In order not to disturb the global arrays KGVN,

IPOS, and IRPOS from GeoFEA, corresponding arrays KGVN2, IPOS1, and IRPOS1 were introduced to take into account the changes according to new number of DOFs. The meaning of variables is explained in the next Section.

```
CALL SPSNEQ ( IWO,NDIM,NDFR,NDF,MXDF,NN,KGVN,NTPE,NEL,      &
               NCORR,LINFO,NF,MF,TF,DXYT, LTYP,IPOS,IRPOS, &
               NEQ,KGVN2,IPOS1,IRPOS1,IDOF,IDFX,RGF, REAC, &
               ITERP,FACLD,DA,PORINS,ValSML,NUMFX,NFXDF)
```

Subsequent changes in the assembly procedures in the subroutines SPSMAT, FMSPS2, formation of GJ preconditioner in FMGJDIAG, and in SQMR solver have been made.

4. There is a major change in subroutine SPSMAT. It is almost rewritten differently to accommodate the changes in item 3. There are several changes in this subroutine. Particularly, the reaction fixing and assigning fixities Section were removed as these are taken care by subroutine SPSNEQ.
5. Subsequent changes in the subroutine FMSPS2 are indicated in box number 5.
6. Changes in the size of allocatable arrays according to NEQ (number of equations excluding equations for zero prescribed fixities) and formation of GJ preconditioner in the subroutine FMGJDIAG2.
7. Minor changes in the SQMR solver are indicated by box number 7.

For demonstration purpose, only the code for generalized Jacobi preconditioned SQMR (GJ-SQMR) solver is presented here.

```

Subroutine UDSOL (DTIMEI, NN, MXDF, NEL, NDF, NTPE, NIP, NPR, NMT,      &
                 KES, NS, NB, NPLax, NDIM, NDMX, NVRS, NPMX, INXL, MDFE,  &
                 KSS, XYZ, DI, DA, NVRN, STR, P, PCOR, REAC, VARINT, NCORR, &
                 KGVN, NMOD, NL, W, PORINS, LTYP, MRELVV, MAT, L, PR, NTY, &
                 FRACLD, IDNA, NEL_NA, MFZ, IPOS, IRPOS, NDFR, DXYT,      &
                 MINFO, LINFO, TF, MF, NF, TOLD, TOLT, MAXIT, StartDlg,   &
                 Dlg_iter, IHND_PIC, IUPD, ITERP, IYIELD_CODE)
  !DEC$ ATTRIBUTES DLLEXPORT :: UDSOL
  Use Xflogm ;   Use Ifwin ;
  Implicit None

  !
  ! -----
  !  GLOBAL VARIABLES (CHANGE NOT ALLOWED REGION)
  ! -----

  Type (DIALOG), Intent (INOUT) :: Dlg_iter
  Logical, Intent (INOUT) :: StartDlg
  Integer (4), Intent (IN) :: NN, MXDF, NEL, NDF, NTPE, NIP, NPR, NMT, &
    KES, NS, NB, NDIM, NDMX, NVRS, NPMX, INXL, MDFE, KSS, &
    NVRN, NL, MAXIT, ITERP, IUPD, KGVN (MXDF, NN), &
    NMOD (NIP, NEL), LTYP (NEL), MRELVV (NEL), MAT (NEL), &
    NTY (NMT), IDNA (NEL), NEL_NA, MINFO (6, 30, 20), &
    LINFO (50, 20), MF (50000), NF

  (1) Integer (4), Intent (INOUT) :: IHND_PIC, NCORR (NTPE, NEL), &
    TF (7, 50000), IPOS (NDF), IRPOS (NDF), &
    IYIELD_CODE (NIP, NEL), NPLax, NDFR

  Integer (8), Intent (IN) :: MFZ

  (1) Real (8), Intent (INOUT) :: DTIMEI, FRACLD, TOLD, TOLT, &
    XYZ (NDIM, NN), DA (NDF), P (NDF), PCOR (NDF), &
    REAC (NDF), STR (NVRN, NIP, NEL), &
    VARINT (NVRS, NIP, NEL), W (120), PORINS (NN), &
    PR (NPR, NMT), L (4, 120), DXYT (7, 50000)

  Real (8), Intent (Out) :: DI (NDF)

  !=====START
  !  CONVERGENCE HISTORY PLOTTING INTERFACE    (IF USER PREFER)
  ! -----

  Pointer (IDPLOT, CHPLOT)
  Interface
    Subroutine CHPLOT (XDlg, StartDlg, IHND_PIC, ISTAGE, IUP,      &
                      ITERP, ISque, ISubInc, iSolve, IDcore, ITERS, &
                      RelRes, TruRes, iPoints)
    !DEC$ ATTRIBUTES DLLIMPORT :: CHPLOT
    Use XFLOGM
    Implicit None

  ! -----
  !  GLOBAL VARIABLES
  ! -----

  Include 'resource.fd'
  Type (DIALOG), Intent (INOUT) :: XDlg
  Logical, Intent (INOUT) :: StartDlg
  Integer, Optional, Intent (IN) :: IUP, ITERP, ISque,      &
    ISubInc, iSolve, IDcore, iPoints, ITERS (:)
  Real (4), Optional, Intent (IN) :: RelRes (:), TruRes (:)
  Integer, Intent (INOUT) :: IHND_PIC, ISTAGE

```

```

        End Subroutine CHPLOT
    End Interface

!=====END
!-----
!   LOCAL VARIABLES   (CHANGE ALLOWED REGION)
!-----

    Integer(4) :: IWO, IStage, IUP, IDcore, nPoints, iPoints, &
        IPLOTDLL, IDPLOT, ISque, ISubInc, iSolve, NDF1, &
        I, IRR, IJK, ITER, NEBE, NNZ, NEQ, J, K, K1, K2, NUM, &
        NUMFX, NFXDF(30000)
    Integer(4), Allocatable :: IDFX(:), Iters(:), IDOF(:), &
        IEBE(:), JEBE(:), KGVN2(:, :), IPOS1(:), IRPOS1(:)
    Real(8) :: ValSML, RhsNrm, ResNrm, RelNrm, TAO, TAO1, TAO2, &
        TETA, TETA1, RHO, RHO1, SIGMA, ALPHA, CN, CNN1, &
        CNN2, BETA, DISP1, DISP2, DISP
    Real(4), Allocatable :: RelRes(:), TruRes(:)
    Real(8), Allocatable :: EBEA(:), DINEW(:), PG(:), AG(:), &
    (2) RG(:), BG(:), BG0(:), RGF(:), BGF(:), DG(:), AD(:), &
        RES(:)

!-----
!   INFORMATION OUTPUT   (IF USER PREFER)
!-----

        IWO = 166
    OPEN(IWO, FILE="USOLV.OUT", FORM='FORMATTED', STATUS='REPLACE')
!-----
!   CONVERGENCE HISTORY PLOTTING   (IF USER PREFER, BUT INTERFACE
!   MUST BE PROVIDED)
!-----

    IF(.not. STARTDLG) THEN
        !       WRITE (IW6,*) ' ***** Error: Iterative      &
        !                               & Solver dialog not found ! '
    ELSE
        IPLOTDLL = LOADLIBRARY("./iterplot.dll"C)
        IF(IPLOTDLL == 0) THEN
            !       WRITE(IW6,*) " *** ERROR - IterPlot.DLL      &
            !                               & CANNOT BE FOUND, PROGRAM      &
            !                               & RUNS WITHOUT CONVERGENCE      &
            !                               & HISTORY PLOTTING ! "
            STARTDLG = .FALSE.
        ELSE
            WRITE(IWO,*) " ---IterPlot.dll has been FOUND! "
            IDPLOT = GETPROCADDRESS(IPLOTDLL, "CHPLOT"C)
            IF(IDPLOT == 0) THEN
                !       WRITE(IW6,*) " **** ERROR - PROCEDURE      &
                !                               & for CHPLOT cannot BE LOCATED &
                !                               & IN THE DYNAMIC LIBRARY      &
                !                               & (IterPlot.dll)! "
                STARTDLG = .FALSE.
            ELSE
                WRITE(IWO,*) "          ---  CHPLOT has been      &
                & LOCATED IN THE DYNAMIC      &
                & LIBRARY (IterPlot.dll)! "
            END IF
        END IF
    END IF
    END IF

!-----
!   GLOBAL MATRIX ASSEMBLY   (CHANGE ALLOWED REGION)
!-----

    IF (NDIM == 3) NPLax = 0
    ALLOCATE ( IDOF(NDF), IDFX(NDF), KGVN2(MXDF, NN), &
        IPOS1(NDF), IRPOS1(NDF), BG0(NDF), RGF(NDF), &

```

```

          BGF(NDF) )
      RGF = 0.D0 ;      BGF = 0.D0 ;      BG0 = 0.D0;
      DI = 0.D0 ;      ValSML= 1.D-40

```

(3)	<b>CALL</b> SPSNEQ(IWO,NDIM,NDFR,NDF,MXDF,NN,KGVN,NTPE,NEL, & NCORR,LINFO,NF,MF,TF,DXYT, LTYP,IPOS,IRPOS,& NEQ,KGVN2,IPOS1,IRPOS1,IDOF,IDFX,RGF, REAC, & ITERP,FACLD,DA,PORINS,ValSML,NUMFX,NFXDF)
-----	---

(3)	NEBE = 69*34*NEL ! ESTIMATED EBE STORAGE NDF1 = NEQ + 1 <b>ALLOCATE</b> (IEBE(NEBE), JEBE(NEBE), EBEA(NEBE) )
-----	---

(4)	<b>CALL</b> SPSMAT(IWO,DTIMEI,NN,MXDF,NEL,NDF,NDF1,NTPE, NIP, & NPR,NMT,KES,NS,NB,NPLax, NDIM,NDMX,NVRS, & NPMX,INXL,MDFE,KSS,XYZ,DA,NVRN,STR,P,PCOR, & REAC,VARINT,NCORR,KGVN,NMOD,NL,W,PORINS, & LTYP,MRELVV,MAT,MINFO,LINFO,TF,MF,NF,DXYT, & IDFX,L,PR,NTY,FACLD,IUPD,ITERP,IDOF, & NEBE,IEBE,JEBE,EBEA,NNZ,IPOS1,IRPOS1,NEQ, & RGF,BGF,BG0,KGVN2,NUMFX,NFXDF,IYIELD_CODE)
-----	--

```

!-----
! FORMING PRECONDITIONER (CHANGE ALLOWED REGION)
!-----

```

(6)	<b>ALLOCATE</b> ( RG(NEQ),PG(NEQ),AG(NEQ),DINEW(NEQ),DG(NEQ), & BG(NEQ),AD(NEQ),RES(NEQ) )  BG(1:NEQ) = BGF(1:NEQ) ! Transfer of diagonals RG(1:NEQ) = RGF(1:NEQ) ! Transfer of RHS <b>DEALLOCATE</b> (RGF, BGF) <b>CALL</b> FMGJDIAG2(NNZ,IEBE,JEBE,EBEA,IPOS1,IRPOS1,NEQ, & NDF,IDOF,BG)
-----	---

```

!-----
! SQMR ITERATIVE SOLVER (CHANGE ALLOWED REGION)
!-----

```

```

      nPoints = 5000 ;      iPoints = 0
ALLOCATE( ITERS(nPoints),RelRes(nPoints),TruRes(nPoints))

```

```

      PG = BG*RG
      TAO = DSQRT(DOT_PRODUCT(RG,RG))
      IF(DABS(TAO)<ValSML) THEN
        Stop
      End if

```

```

!
      RhsNrm = 0.D0

```

(7)	<b>DO</b> I = 1, NEQ RhsNrm = RhsNrm + RG(I)*RG(I) <b>END DO</b>
-----	--

```

      RhsNrm = DSQRT(RhsNrm)

```

```

!
      TETA = 0.D0
      RHO = DOT_PRODUCT(RG,PG)

```

```

!
      DG = 0.D0
      DINEW = 0.D0

```

```

!-----
! START SQMR ITERATIONS
!-----

```

```

      DO ITER = 1, MAXIT
        IF(ITER.NE.1) THEN

```

```

        TAO = TAO1
        TETA = TETA1
        RHO = RHO1
    END IF
! ===== PERFORM --- AG = A*PG =====START
    AG=0.D0
    CALL UDMATVEC3 (NEQ,NDF1,IEBE(1:NNZ),JEBE(1:NDF1),      &
        EBEA(1:NNZ),NNZ,PG,AG)
! ===== PERFORM --- AG = A*PG =====END
    SIGMA = DOT_PRODUCT(PG,AG)
    IF (DABS(SIGMA)<ValSML) THEN
        Write(IWO,*) " SIGMA=0, ALPHA CANNOT BE COMPUTED IN ", &
            ITER,"-TH ITERATION! "

        STOP
    END IF
    ALPHA=RHO/SIGMA
    RG=RG-ALPHA*AG
    TAO2=DSQRT(DOT_PRODUCT(RG, RG))
    TETA1=TAO2/TAO
    CN=1.D0/DSQRT(1.D0+TETA1*TETA1)
    TAO1=TAO*TETA1*CN
    CNN1=(CN*TETA)*(CN*TETA)
    CNN2=CN*CN*ALPHA
    DG=CNN1*DG+CNN2*PG
    DINEW=DINEW+DG

!

    RES = RG
    ResNrm = 0.D0
(7) DO I = 1, NEQ
        ResNrm = ResNrm + RES(I)*RES(I)
    END DO
    ResNrm = DSQRT(ResNrm)

!

    IF (RhsNrm>ValSML) RelNrm=ResNrm/RhsNrm
    IF (MOD(ITER,100).EQ.0) THEN
        WRITE(IWO,90) ITER,RelNrm,ResNrm
    END IF

! CONVERGENCE PLOTTING
    IF (MOD(ITER,10)==0) THEN
        IF (STARTDLG) THEN
            iPoints = iPoints + 1
            IF (iPoints <= nPoints) THEN
                ITERS(iPoints) = ITER
                RelRes(iPoints) = SNGL(RelNrm)
                TruRes(iPoints) = SNGL(ResNrm)
                ISTAGE = 1 ; IUP = 2
                CALL CHPLOT(Dlg_ITER,StartDlg,IHND_PIC,ISTAGE, &
                    IUP,ITERP,ISque,ISubInc,iSolve,IDcore, &
                    ITERS,RelRes,TruRes,iPoints)

                END IF
            END IF
        END IF

!

        IF (RelNrm<TOLD .AND. ResNrm<TOLT) THEN
(7) DO I = 1, NEQ
            IRR = IRPOS1(I)
            DI (IRR)=DINEW(I)
        END DO
        GOTO 100
    END IF

```

```

      AG = BG*RG
      RHO1 = DOT_PRODUCT(RG, AG)
      IF (DABS(RHO)<ValSML) THEN
        WRITE(IWO,*) " RHO=0, BETA CANNOT BE COMPUTED"
        STOP
      END IF
      BETA=RHO1/RHO
      PG=AG+BETA*PG
    END DO
  !
100 WRITE(IWO,80) NDF,ITER,RelNrm,ResNrm !TOLD,
70  FORMAT(1X,'***** ITERATION RESULTS : SPARSE           &
      & GENERALISED JACOBIAN SQMR'/)
80  FORMAT(5X,'NDF =',I7,4X,'ITER=',I7,4X,'RELRES=',      &
      E12.3,4X,'TRURES=',E12.3/)
90  FORMAT(5X,'ITER=',I7,4X,'REL. DISP=',E16.7,4X,      &
      'TRUE DISP=',E16.7/)

      WRITE(IWO,'(A,I7)') 'NUMBER OF EQUATIONS = ',NEQ

      Return
End Subroutine UDSOL
!
!-----END OF MAIN USER DEFINED SUBROUTINE-----
!=====

Subroutine SPSMAT(IWO,DTIMEI,NN,MXDF,NEL,NDF,NDF1,NTPE,      &
      NIP,NPR,NMT,KES,NS,NB,NPLax,NDIM,NDMX,NVRS,NPMX,      &
      INXL,MDFE,KSS,XYZ,DA,NVRN,STR,P,PCOR,REAC,VARINT,      &
      NCORR,KGVN,NMOD,NL,W,PORINS,LTP,MRELVV,MAT,          &
      MINFO,LINFO,TF,MF,NF,DXYT,IDFX,L,PR,NTY,FRACLD,      &
      IUPD,ITERP,IDOF,NEBE,IEBE,JEBE,EBEA,NNZ,IPOS,        &
      IRPOS,NDFR,RG,BG,BG0,KGVN2,NUMFX,NFXDF,IYIELD_CODE)

! *****
!   THIS SUBROUTINE IS TO FORM GOBAL COMPRESSED MATRIX FROM
!   ELEMENT STIFFNESS MATRICES AND CONSTRUCT GJ
!   PRECONDITIONER FOR SYMMETRIC ITERATIVE SOLVER.
! *****
      Use Ifwin
      Implicit None

! -----
!   Global Variables
! -----

      Integer(4), Intent(In):: IWO,NN,MXDF,NEL,NDF,NTPE,NIP,      &
      NPR,NMT,KES,NS,NB,NPLax,NDIM,NDMX,NVRS,NPMX,INXL,      &
      MDFE,NL,IUPD,KSS,NVRN,NF,ITERP,NEBE,NDFR,            &
      KGVN(MXDF,NN),NMOD(NIP,NEL),LTP(NEL),MAT(NEL),        &
      NTY(NMT),MRELVV(NEL),MINFO(6,30,20),LINFO(50,20),      &
      MF(50000),KGVN2(MXDF,NN),IDOF(NDF),IDFX(NDF),          &
      NUMFX,NFXDF(200)

      Integer(4), Intent(InOut):: NNZ,NDF1,NCORR(NTPE,NEL),      &
      TF(7,50000),IEBE(NEBE),JEBE(NEBE),IPOS(NDF),          &
      IRPOS(NDF),IYIELD_CODE(NIP,NEL)

      Real(8), Intent(In):: DTIMEI,FRACLD,L(4,120),            &
      XYZ(NDIM,NN),DA(NDF),REAC(NDF),W(120),PORINS(NN),      &
      STR(NVRN,NIP,NEL)

      Real(8), Intent(InOut):: RG(NDF),BG(NDF),BG0(NDF),      &
      P(NDF),PCOR(NDF),DXYT(7,50000),EBEA(NEBE),            &
      PR(NPR,NMT),VARINT(NVRS,NIP,NEL)

! -----

```

```

!      Local Variables
! -----
      Integer(4) :: I, J, K, K1, K2, KL, NE, LT, ICONSO, MUS, KC, NDOF,      &
                  NNE, IL, IL1, IG, NA, IJ2, IJ3, IJ4, IJ5, JN0, MN1, NUM, MN,      &
                  NZEBE, IRR, IELSTDLL, IDASSDLL, IDELST, IDASSEMB
      Real(8) :: ALAR
      Integer(4), ALLOCATABLE :: IEE(:)
      Real(8), Allocatable :: ES(:), AG(:), RG1(:)
! =====START
      POINTER (IDELST, ELESTF)
      INTERFACE
      Subroutine ELESTF(IWO, Lt, Ne, Mus, Inxl, Sg, Ksg, Dtimei,      &
                      Nn, Mxdf, Nel, Ndf, Ntpe, Nip, Npr, Nmt, Ns, Nb, NL, NPLax,      &
                      Ndim, Ndmx, Nvrs, Npmx, Kss, Xyz, Da, Nvrn, Str, P, Varint,      &
                      Ncorr, Kgvn, Nmod, Mat, W, L, Pr, Nty, LInfo, Iupd,      &
                      Iyield_code)
      !DEC$ ATTRIBUTES DLLIMPORT::ELESTF
      Implicit None
! -----
!      Global Variables
! -----
      Integer(4), Intent(In) :: IWO, Lt, Ne, Mus, Inxl, Ksg,      &
                      Nn, Mxdf, Nel, Ndf, Ntpe, Nip, Npr, Nmt, Ns, Nb, NL, NPLax,      &
                      Ndim, Ndmx, Nvrs, Npmx, Kss, Nvrn, Iupd, Ncorr (NTPE, NEL), &
                      KGVN (MXDF, NN), NMOD (NIP, NEL), MAT (NEL), NTY (NMT),      &
                      LINFO (50, 20)
      Integer(4), Intent(InOut) :: IYIELD_CODE (NIP, NEL)
      Real(8), Intent(In) :: Dtimei, Xyz (NDIM, NN), DA (NDF),      &
                      STR (NVRN, NIP, NEL), W (120), L (4, 120)
      Real(8), Intent(InOut) :: PR (NPR, NMT),      &
                      VARINT (NVRN, NIP, NEL)
      Real(8), Intent(Out) :: P (NDF), Sg (Ksg)
      End Subroutine ELESTF
      END INTERFACE
! =====END
! =====START
      POINTER (IDASSEMB, ETOS)
      INTERFACE
      Subroutine ETOS (UANZ, ARR, BRR, CRR, NI, NNZ)
      !DEC$ ATTRIBUTES DLLIMPORT::ETOS
      Implicit None
! -----
!      Global Variables
! -----
      Integer(4), Intent(InOut) :: UANZ, NI, NNZ, ARR (UANZ),      &
                      BRR (UANZ)
      Real(8), Intent(InOut) :: CRR (UANZ)
      End Subroutine ETOS
      END INTERFACE
! =====END
      Allocate ( IEE (MDFE), ES (KES), AG (NDF) )
!
      IELSTDLL = LOADLIBRARY("./ELESTF.dll"C)
      IF (IELSTDLL == 0) THEN
      WRITE(IWO,*) "      --- ELESTF.dll CAN NOT be FOUND ! "
      ELSE
      WRITE(IWO,*) "      --- ELESTF.dll loaded ! "
      IDELST = GETPROCADDRESS (IELSTDLL, "ELESTF"C)
      IF (IDELST == 0) THEN
      WRITE(IWO,*) "      Warning ! *** Subroutine ELESTF      &
                      & Can Not be FOUND in ELESTF.dll ! "

```

```

ELSE
  WRITE(IWO,*) " ELESTF SUBROUTINE IS LOCATED! "
END IF
END IF

!
IDASSDLL = LOADLIBRARY("./Assemb.dll"C)
IF(IDASSDLL == 0) THEN
  WRITE(IWO,*) " --- Assemb.dll CAN NOT be FOUND ! "
ELSE
  WRITE(IWO,*) " --- Assemb.dll loaded ! "
  IDASSEMB = GETPROCADDRESS(IDASSDLL, "ETOS"C)
  IF(IDASSEMB == 0) THEN
    WRITE(IWO,*) " Warning ! *** Subroutine ETOS Can &
      & Not be FOUND in Assemb.dll ! "
  ELSE
    WRITE(IWO,*) " ETOS SUBROUTINE IS LOCATED! "
  END IF
END IF

ALAR=1.D+45
IEBE = 0 ; JEBE = 0 ; EBEA = 0.D0
AG = 0.D0
KL = 0 ;
NZEBE = 0 ;

DO NE = 1, NEL
  LT = LTYP(NE)
  IF( LT < 0) CYCLE

!
  ICONSO = 1
  IF( (LT-2)*(LT-4)*(LT-6)*(LT-8)*(LT-10)*(LT-14)*
    (LT-16)==0 ) ICONSO = 0

!
  MUS = MRELVV(NE)
  CALL ELESTF(IWO,LT,NE,MUS,INXL,ES,KES,DTIMEI,NN, &
    MXDF,NEL,NDF,NTPE,NIP,NPR,NMT,NS,NB,NL, &
    NPLax,NDIM,NDMX,NVRS,NPMX,KSS,XYZ,DA,NVRN, &
    STR,P,VARINT,NCORR,KGVN,NMOD,MAT,W,L,PR,NTY, &
    LINFO,IUPD, Iyield_Code)

  NDOF = LINFO(16,LT)
  NNE = LINFO(1,LT)

!
  CALL FMSPS2(IWO,NN,NE,LT,NDOF,NNE,MXDF,NTPE,NEL, &
    KGVN,KGVN2,NCORR,ES,KES,MDFE,IEBE,JEBE,EBEA, &
    NEBE,NZEBE,IPOS,NDF,NDFR,ICONSO)

END DO

!
WRITE(IWO, ' (1X,A14,1X,F6.2,A1) ' ) 'SPARSE RATIO =', &
  DFLOTJ(NZEBE)*2.D0/(DFLOTJ(NDFR) &
  *(DFLOTJ(NDFR)+1.D0))*1.D2 , '%'

!
CALL ETOS(NZEBE,JEBE(1:NZEBE),IEBE(1:NZEBE), &
  EBEA(1:NZEBE),NDF1,NNZ)

!
!-----
! FIX REACTION BUG AND ASSIGN FIXITIES WITH THEIR PRESCRIBED
! VALUES
!-----
! ADDING PENTALY TERM TO THE DIAGOAL OF A

```

```

        IF (NUMFX > 0) THEN
            DO J = 1, NUMFX
                NUM = NFXDF(J)
                I = IPOS(NUM)
                K = JEBE(I+1)-1
                EBEA(K) = EBEA(K) + ALAR
            END DO
        END IF

! EXTRACTING THE DIAGONAL OF A
    BG = 0.D0
    DO I = 1, NDFR;
        K = JEBE(I+1)-1 ;
        BG(I) = EBEA(K) ;
    END DO

! -----
! GET INITIAL VALUES OF DINOW, RG, BG, DG AND PG
! -----

    ALLOCATE ( RG1(NDF) ) ! RG1 is a temp. vector to store RG
    RG1 = RG + P + PCOR ; ! Right hand side vector
    NCORR = IABS(NCORR) ;

!
    DO I = 1, NDFR;
        RG(I) = RG1(IRPOS(I))
    END DO

    BG0 = 1.D0

!
    DO I=1,NDFR
        IRR = IRPOS(I)
        IF (BG(I)>1.D20) THEN
            IDFX(IRR) = 2
!             AG(I)=2.D0 !! CX
            BG0(IRR)=1.D0/DSQRT(BG(I))
            BG(I)=1.D0
            RG(I)=RG(I)*BG0(IRR)
        ELSEIF (BG(I)<-1.D10) THEN
            IDFX(IRR) = 3
!             AG(I)=2.D0
            BG0(IRR)=1.D0/DSQRT(DABS(BG(I)))
            BG(I)=-1.D0
            RG(I)=RG(I)*BG0(IRR)
        END IF
!
    END DO

!
!     HERE, BG0 IS THE SCALING VECTOR.
!
!     PERFORM SYMMETRIC DIGONAL SCALING WITH BG0
    DO I = 1, NDFR
!         IF (NDFR/=NDF) THEN
!             IRR = IRPOS(I)
!         ELSE
!             IRR = I
!         END IF
!
!         IF (AG(IRR)>1.9D0) THEN
!             IF (IDFX(IRR) /= 0) THEN
!                 K1 = JEBE(I); K2 = JEBE(I+1)-1
!                 DO K = K1, K2
!                     EBEA(K) = EBEA(K)*BG0(IRR)

```

```

        END DO
        EBEA(K2) = EBEA(K2)*BG0(IRR)
        DO J = I+1, NDFR
            K1 = JEBE(J); K2 = JEBE(J+1)-1
            DO K = K1, K2
                IF(IEBE(K).GT.I) EXIT
                IF(IEBE(K).EQ.I) EBEA(K) = EBEA(K)*BG0(IRR)
            END DO
        END DO
    END IF
END DO

Return
End Subroutine SPSMAT
!=====END SUBROUTINE SPMAT =====
Subroutine FMSPS2(IWO,NN,NE,LT,NDOF,NNE,MXDF,NTPE,NEL,KGVN,    &
    KGVN2,NCORR,SG,KSG,MDFE,IEBE,JEBE,EBEA,NEBE,NZEBE,    &
    IPOSONDF,NDFR,ICONSO)
!*****
! THIS SUBROUTINE COLLECT NON-ZERO ENTRIES FOR EACH NEW
! GENERATED ELEMENT STIFFNESS MATRIX, FORMING THE ELEMENT-LEVEL
! THREE VECTORS WHICH STORE NONZERO ENTRIES OF UPPER TRIANGULAR
! PART OF A.
! THIS SUBROUTINE IS LOCATED IN ELEMENT LOOP TO COLLECT ELEMENT
! STIFFNESS MATRIX ENTRIES.
!     NE      : CURRENT ELEMENT NUMBER
!     NTOT    : TOTAL FREEDOMS OF CURRENT ELEMENT (i.e. NDOF);
!     KGVN    : ELEMENT STEERING VECTOR
!     KGVN2   : ELEMENT STEERING VECTOR(includes only free DOFs);
!     NCORR   : ELEMENT NODE & ELEMENT CORRELATION MATRICES;
!     SG      : COLUMN-WISE UPPER TRIANGULAR ELEMENT "STIFFNESS"
!               MATRIX STORED IN ONE VECTOR.
!     IEBA    : GLOBAL ROW INDEX;
!     JEBA    : GLOBAL COLUMN INDEX;
!     EBEA    : CORRESPONDENT VALUE OF THE NONZERO ELEMENT
!               STIFFNESS ENTRY;
!     NZEBE   : ACCUMULATED TOTAL NUMBER OF NONZERO ELEMENT-
!               LEVEL ENTRIES (NOT ESTIMATED NUMBER (NEBE) ANY
!               MORE WHEN RETURNED).
!*****
Implicit None
Integer(4):: IL,I,J,K,IR,NA,ICOUNT,KSG,NN,NE,LT,NDOF,    &
    NNE,MXDF,NTPE,NEL,MDFE,NEBE,NZEBE,KGVN(MXDF,NN),    &
    NCORR(NTPE,NEL),IEBE(NEBE),JEBE(NEBE),NDF,    &
    IPOSONDF,NDFR,ICONSO,KGVN2(MXDF,NN),IWO
Real(8):: SG(KSG),EBEA(NEBE)
Integer(4), Allocatable :: IEE(:)
!--- STORING UPPER TRIANGLE OF ELEMENT STIFFNESS COLUMN BY
! COLUMN ---
!--- ASSUMING TO SOLVE SYMMETRIC PROBLEMS -----
Allocate(IEE(MDFE))
!
    IL = 0 ; IEE = 0 ;
    DO J=1,NNE
        NA = IABS(NCORR(J,NE))
        L3: DO K=1,MXDF
            IR = KGVN(K,NA)
            IF( IR > 0 ) THEN
                IF(K.GT.4.AND.LT.NE.12) CYCLE L3
                IF(ICONSO == 0.AND.K > 3) CYCLE L3
                IF(K.EQ.4.AND.LT.EQ.12) CYCLE L3

```

```

(5)      IF (LT == 1.AND.K > 3) CYCLE L3
          IL = IL + 1
          IEE(IL) = KGVN2(K,NA)
        END IF
      END DO L3
    END DO

    !
    ICOUNT = 0
    DO J = 1, NDOF
      DO I = 1, J
        ICOUNT = ICOUNT + 1
        IF (DABS(SG(ICOUNT)) > 1.D-40) THEN
          IF (IEE(I)>0 .AND. IEE(J)>0) THEN
            IF (IEE(I).LE.IEE(J)) THEN
              NZEBE = NZEBE + 1 ; IEBE(NZEBE) = IEE(I)
              JEBE(NZEBE) = IEE(J) ;
              EBEA(NZEBE)=SG(ICOUNT)
            ELSE
              NZEBE = NZEBE + 1 ; IEBE(NZEBE)=IEE(J)
              JEBE(NZEBE) = IEE(I) ;
              EBEA(NZEBE)=SG(ICOUNT)
            END IF
          END IF
        END IF
      END IF
    END DO
  END DO

  !
  Return
End Subroutine FMSPS2
!=====END SUBROUTINE FMSPS2=====
Subroutine FMGJDIAG2 (NNZ,IEBE,JEBE,EBEA,IPOS,IRPOS,NDFR,      &
  NDF,IDOF,BG)
!*****
! THIS SUBROUTINE IS TO CONSTRUCT GJ PRECONDITIONER FOR
!   SYMMETRIC
!   BG = IN INPUT, DIAGONAL OF A
!   = IN OUTPUT, INVERTED DIAGONAL (GJ FORM)
!*****
  Implicit None
  Integer(4) :: J,K,K1,K2,IPOS(NDF),IRPOS(NDF),NDFR,NDF,      &
    IRR,IRRJ,NNZ,IEBE(1:NNZ),JEBE(1:NDFR+1),IDOF(NDF)
  Real(8) :: BG(NDF),COEF,EBEA(1:NNZ)

  !
  DO J = 2, NDFR
    K1 = JEBE(J) ; K2 = JEBE(J+1)-2
    IRRJ = IRPOS(J)
    !
    IF (IDOF (IRRJ).EQ.1) THEN
      DO K = K1 , K2
        IRR = IRPOS (IEBE(K))
        !
        IF (IDOF (IRR).EQ.0 ) THEN
          BG (IEBE(K)) =      &
            BG (IEBE(K)) -EBEA(K)*EBEA(K)/BG(J) ;
        END IF
      END DO
    ELSE
      ! IDOF (IRRJ)==0
      DO K = K1 , K2
        IRR = IRPOS (IEBE(K))
        !

```

```

(6)      IF (IDOF (IRR) .EQ. 1 ) THEN
          BG (J) = BG (J) - EBEA (K) * EBEA (K) / BG (IEBE (K)) ;
          END IF
        END DO
      END IF
    END DO
    !
    COEF = -4.D0
    ! COEF-SCALING FACTOR (NEGATIVE IS PREFERRED)
    DO J=1, NDFR
      IRRJ = IRPOS (J)
      IF (IDOF (IRRJ) .EQ. 0) THEN
        ! MODIFIED DIAGONAL WITH RELAXATION PARAMETER.
        BG (J) = COEF * DABS (BG (J))
      END IF
      BG (J) = 1.D0 / BG (J) ;
    END DO
    !
    RETURN
  End Subroutine FMGJDIAG2
!=====END SUBROUTINE FMGJDIAG2 =====
SUBROUTINE UDMATVEC3 (NDF, NDF1, ICSC, JCSC, CSCA, NNZ, VIN, VOUT)
! ~~~~~ THIS ROUTINE IS USED IN SYMMETRIC ITERATIVE SOLVER~~~~~
! MATVEC3 - performs MATRIX-VECTOR PRODUCTS,
!   'VIN = A*VOUT', in ITERATIVE SOLVER,
!   SPARSE IMPLEMENTATION FOR SYMMETRIC MATRIX
!*****
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
INTEGER :: ICSC (NNZ), JCSC (NDF1)
REAL (8) :: CSCA (NNZ), VIN (NDF), VOUT (NDF)
!
  VOUT = 0.D0
  DO J=1, NDF
    IF (VIN (J) .NE. 0.D0) THEN
      K1 = JCSC (J); K2 = JCSC (J+1) - 1 ;
      DO K = K1, K2
        IR = ICSC (K) ;
        VOUT (IR) = VOUT (IR) + CSCA (K) * VIN (J) ;
      END DO
    END IF
  !
    TMP = 0.D0 ; K1 = JCSC (J); K2 = JCSC (J+1) - 2 ;
    DO K = K1, K2
      IR = ICSC (K) ;
      TMP = TMP + VIN (IR) * CSCA (K) ;
    END DO
    VOUT (J) = VOUT (J) + TMP ;
  END DO
!
  Return
END SUBROUTINE UDMATVEC3
!=====END SUBROUTINE UDMATVEC3 =====
Subroutine SPSNEQ (IWO, NDIM, NDFR, NDF, MXDF, NN, KGVN, NTPE,      &
  NEL, NCORR, LINFO, NF, MF, TF, DXYT, LTYP, IPOS, IRPOS, NEQ,    &
  KGVN2, IPOS1, IRPOS1, IDOF, IDFX, RG, REAC, ITERP, FRACLD,      &
  DA, PORINS, ValSML, NUMFX, NFXDF)
! This subroutine is to find NEQ by avoiding the fixities with
! prescribed zero displacement or excess pore pressure
! COUNT FREE DOFS AND ASSIGN IDENTITY FOR FREE AND FIXED DOFS
! IDOF = 1 FOR DISPLACEMENT DOFS (INCLUDES NON-ZERO
!   PRESCRIBED VALUES)

```

```

!      = 0  FOR PORE PRESSURE DOFS (INCLUDES NON-ZERO
!      PRESCRIBED VALUES)
!      = 99 FOR FIXED DOFS
! IDFX = 1  FOR FIXED DOFS
!      = 0  FOR FREE DOFS
! NEQ  =    number of equations for unknown variables only
!      (free DOFs)
! KGVN2(MXDF,NN)
!      =    Modified steering vector accroding to NEQ DOFs
! IPOS1(NDF)
!      =    Position in recalculated DOFs (= NEQ)
! IRPOS1(NDF)
!      =    Position in Global total DOFs(including fixities)
! NUMFX=    No. of DOFs with non-zero prescribed values
! NFXDF(NUMFX)
!      =    Arrays containing NUMFX DOFs
!-----
implicit none

integer(4),intent(in):: IWO,NDIM,NDFR,NDF,MXDF,NN,NTPE,NEL, &
    KGVN(MXDF,NN),LINFO(50,20),NF,MF(50000), &
    LTYP(NEL),ITERP,NCORR(NTPE,NEL),IPOS(NDF),IRPOS(NDF)
integer(4),intent(inout):: TF(7,50000),IPOS1(NDF),IRPOS1(NDF)
real(8),intent(in):: FRACLD,DA(NDF),REAC(NDF),PORINS(NN), &
    ValSML
real(8),intent(inout):: DXYT(7,50000),RG(NDF)
integer(4),intent(out):: NEQ,KGVN2(MXDF,NN),IDOF(NDF), &
    IDFX(NDF),NUMFX,NFXDF(30000)
!-----
!    Local Variables
!-----
integer(4):: J,K,NA,NDOF,LT,NNE,IJ3,IJ4,NE,MN1,NUM,ICONSO, &
    MN,ICOUNT,IC
real(8):: ALAR

NEQ = NDFR
KGVN2 = KGVN
ALAR=1.D+45
IDOF = 1    ! Initialize for all FREE DOFs
IDFX = 0    ! Initialize for all FIXED DOFs
IC    = 0    ! Initialize counting prescribed displacement or
!            ! excess pore pressure
NFXDF = 0

DO NE = 1, NEL
    LT = LTYP(NE)

    IF( LT < 0) CYCLE

!
    ICONSO = 1
    IF( (LT-2)*(LT-4)*(LT-6)*(LT-8)*(LT-10)*(LT-14)* &
        (LT-16)==0 ) ICONSO = 0

    NDOF = LINFO(16,LT)
    NNE = LINFO(1,LT)
    JNO = LINFO(6,LT) ! TOTAL PORE PRESSURE DOFS

L1: DO J=1,JNO
    NA = IABS(NCORR(J,NE))
    K = KGVN(NDIM+1,NA)
    IDOF(K) = 0    ! IDOX = 0 FOR PORE PRESSURE DOFS

```

```

END DO L1

      IJ3 = 0
L2: DO J=1,NNE
      NA = IABS(NCORR(J,NE)) ; MN1 = 1 ;
      L3: DO K=1,MXDF
      NUM = KGVN(K,NA) ;
      IF(NUM.EQ.0) CYCLE L3
      IF(NUM>NDF) CYCLE L3
      IF(K.GT.4.AND.LT.NE.12) CYCLE L3
      IF(ICONSO == 0.AND.K > 3) CYCLE L3
      IF(K.EQ.4.AND.LT.EQ.12) CYCLE L3
      IF(LT == 1.AND.K > 3) CYCLE L3
      !
      ! IF(LT==12.AND.K > 4) IDOF(NUM)=2
      !
      ! IJ3 = IJ3 + 1
      IF(NCORR(J,NE)>0) CYCLE L3
      ! AG(NUM) = 1.D0
      IF(K.NE.1) MN1=MN
L4: DO MN = MN1, NF
      IF(MF(MN).NE.NA) CYCLE L4
      IF(TF(K,MN).EQ.0) THEN
        RG(NUM) = REAC(NUM)
        CYCLE L3
      END IF
      ! IJ4 = IJ3*(IJ3+1)/2
      ! ES(IJ4) = ES(IJ4) + ALAR
      ! IDFX(NUM) = 1
      IDOF(NUM) = 99
      !
      IF(TF(K,MN).EQ.1) THEN
!-----
! ZERO DISPLACEMENT FOR NEWTON-RAPHSON'S ITERATION WHEN ITERP>0
!-----
        IF (DABS(DXYT(K,MN)) > ValSML) THEN
          IF(ITERP==0) THEN
            ! IJ4 = IJ3*(IJ3+1)/2
            ! ES(IJ4) = ES(IJ4) + ALAR

            IC = IC+1
            NFXDF(IC) = NUM;
            IDOF(NUM) = 1 ; IDFX(NUM) = 2
            RG(NUM) =
            RG(NUM)+ALAR*DXYT(K,MN)*FRACLD &
            IF ( K == NDIM+1 ) THEN
              ! PORE PRESSURE DOF
              IDOF(NUM) = 0; IDFX(NUM) = 3
            END IF
          END IF
        ELSE
          NEQ = NEQ-1
          KGVN2(K,NA) = 0
        END IF
      END IF
!=====( NEWTON-RAPHSON METHOD )=====END
! RG(NUM) = RG(NUM)+ALAR*DXYT(K,MN)*FRACLD ! - 'RG'
      CYCLE L3
    ELSE
      IF( TF(K,MN).EQ.2) THEN
        DXYT(K,MN) = DXYT(K,MN) - DA(NUM)
        NEQ = NEQ-1

```

```

                                KGVN2(K,NA) = 0
ELSEIF( TF(K,MN) .EQ. 3) THEN
    IC = IC+1
    NFXDF(IC) = NUM ;
    IDOF(NUM) = 0 ; IDFX(NUM) = 3
    DXYT(K,MN) =
    DXYT(K,MN)-DA(NUM)-PORINS(NA)      &
END IF
IF (DABS(DXYT(K,MN)) > ValSML)
    RG(NUM)=RG(NUM)+ALAR*DXYT(K,MN)
    DXYT(K,MN)=0.D0
    TF(K,MN)=1
EXIT L3
END IF
END DO L4
END DO L3
END DO L2
END DO
!-----
NUMFX = IC
!
! WRITE(IWO,'(A,I9)') 'Number of unknown equations, NEQ =',NEQ
ICOUNT = 0 ; IPOS1 = 0; IRPOS1 = 0

DO J = 1,NN
    L5: DO K = 1,MXDF
        NUM = KGVN2(K,J) ;
        IF(NUM.EQ.0) CYCLE L5
        IF(NUM>NDF) CYCLE L5
        IF(K.GT.4.AND.LT.NE.12) CYCLE L5
        IF(ICONSO == 0.AND.K > 3) CYCLE L5
        IF(K.EQ.4.AND.LT.EQ.12) CYCLE L5
        IF(LT == 1.AND.K > 3) CYCLE L5
        IF(IPOS(NUM) == 0) THEN
            KGVN2(K,J) = 0 ; IDOF(NUM) = 99
            CYCLE L5
        END IF
        ICOUNT = ICOUNT+1
        KGVN2(K,J) = ICOUNT
        IPOS1(KGVN(K,J)) = ICOUNT ; IRPOS1(ICOUNT) = KGVN(K,J);
    END DO L5
END DO
Return
!
end subroutine SPSNEQ
!=====END SUBROUTINE SPSNEQ =====

```

## F.2. List of variables used in subroutine UDSOL

<b>Type</b> (DIALOG), <b>Intent</b> (INOUT)::	
Dlg_iter	handle for convergence history plotting dialog
<b>Integer</b> (4), <b>Intent</b> (IN)::	
NN	total number of nodes
MXDF	maximum possible number of variables at any node
NEL	total number of elements
NDF	Total number of d.o.f.
NTPE	maximum number of nodes in any element in the mesh
NIP	total number of integration point in the element
NPR	number of properties per material (16)
NMT	maximum allowable number of different material zones (25)
KES	size of element stiffness matrix ES
NS	number of stress/strain components
NB	number of columns in B matrix (=NDIM $\times$ NDMX)
NDIM	number of dimensions to problem
NDMX	maximum number of displacement nodes in any element
NVRS	number of stress components and parameters
NPMX	maximum number excess pore pressure nodes in any element in current analysis
INXL	index to array LINFO
MDFE	maximum number of d.o.f. in any element

KSS	size of solid element stiffness matrix SS (upper triangular)
NVRN	number of stress-strain components
NL	number of area coordinates
MAXIT	program given maximum number of iteration for user iterative solver, MAXIT can be adjusted by user.
ITERP	Newton-Raphson nonlinear iteration number
IUPD	switch for updating geometry  0 – coordinates are not updated  1 – coordinates are updated
KGVN(MXDF,NN)	list of indexes of first d.o.f. associated with each node to global arrays P, DI and DA.
NMOD(NIP,NEL)	list of switches to indicate state of stress of integration points for MPT(Material Property Type) 5 (0 – elastic, 1 – first yield, 2 – continuous yield)
LTyp(NEL)	list of element type numbers
MRELVV(NEL) ,	User element numbers for program element numbers
MAT(NEL)	list of material zone numbers of elements
NTY(NMT)	material type numbers of different material zones
IDNA(NEL)	Currently, identifier array for elements with non-associated plastic flow material properties and yielding integration points.  0 – other elements including the elements with non-associated plastic flow material, but without yielding integration points.

	1 – elements with non-associated plastic flow material properties.
NEL_NA	Currently, the total number of elements with non-associated plastic flow material properties and yielding integration points (for non-symmetric solver)
MINFO(6,30,20)	To supplement the array LINFO, it gives the unique number for each of the variables of a node.
LINFO(50,20)	details (i.e. number of vertex nodes, midside nodes and d.o.f. of each node) of different element types
MF(50000)	list of nodes with fixities
NF	counter of nodes with fixities (i.e., nodes with one or more d.o.f. which have prescribed values)
<b>Integer(4),Intent(INOUT)::</b>	
IHND_PIC	handle for convergence plotting area
NCORR(NTPE,NEL)	list of element nodal links (i.e. list of nodes associated with each element)
TF(7,50000)	list of fixity codes
IPOS(NDF)	The recalculated number d.o.f. correspondent to global d.o.f. (for sparse solver)
IRPOS(NDF)	IRPOS(1:NDFR) stores correspondent global number of d.o.f. (for sparse solver)
IYIELD_CODE(NIP, NEL)	List of elements with their yielding information at integration points
NPLax	plane strain / axisymmetric / 3-D analysis option  1        axisymmetric

	0 otherwise (plane strain / 3-D analysis)
NDFR	Recalculated total number of d.o.f. due to removed elements, if there is no removed elements, NDFR = NDF (for sparse solver)
<b>Integer(8),Intent(IN)::</b>	
MFZ	allocated array size for array ELPA
<b>Real(8),Intent(IN)::</b>	
DTIMEI	time increment
FRACLD	load ratio for current increment
TOLD	tolerance for relative residual norm convergence  criterion defined by program, TOLD can be adjusted by user.
TOLT	tolerance for true residual norm convergence criterion  defined by program, TOLT can be adjusted by user.
XYZ(NDIM,NN)	nodal coordinates
DA(NDF)	global vector of cumulative displacements
P(NDF)	global incremental load vector
PCOR(NDF)	correcting load vector
REAC(NDF)	global reactions vector (at nodes with described displacements)
STR(NVRN,NIP,NE L)	cumulative strains at integration point
VARINT(NVRS,NIP	current values of variables Sx, Sy, Sz, Txy (Tyz, Tzx),

,NEL)	U, E and Pc for all integration points
W(120)	Weighting factors for integration points
PORINS(NN)	insitu stage hydrostatic pore water pressure
PR(NPR, NMT)	table of material properties
L(4,120)	list of area coordinates of integration points for different element types
DXYT(7,50000)	list of prescribed displacements and excess pore pressures at nodes
<b>Real(8), Intent(Out) ::</b>	
DI(NDF)	Returned global vector of incremental displacements

*(blank)*