The Recursive Block Method Applied To The Solution of A Linear System

By

YUE DAI

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirement

for the Degree of

MASTER OF SCIENCE

Department of Applied Mathematics

The University of Manitoba

Winnipeg, Manitoba

ISBN 0-612-13057-6

Name YUE DAI

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

APPLIED MATHEMATICS — SUBJECT TERM

0405 — SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences
Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences
Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

THE RECURSIVE BLOCK METHOD APPLIED TO THE SOLUTION OF A LINEAR SYSTEM

BY

YUE DAI

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

Abstract

A recursive block algorithm for solving a linear system is developed, analyzed and tested in this thesis. The method is basically a recursive block version of Gaussian elimination. The theory of the algorithm is analyzed in detail and the computation cost of the algorithm is estimated. Codes written in both Matlab and FORTRAN were used to test the algorithm and these codes are attached as an appendix. The new method is comparable in execution speed and memory requirements to the standard Gaussian elimination algorithm. The analysis and tests of the algorithm indicate that this algorithm is reliable and effective and can be applied to the solution of a practical problem. It solves a linear system with good precision and high speed as Gaussian elimination. Improvement of the algorithm is recommended in order to apply it to sparse linear systems, and to linear systems with a coefficient matrix of a specialized matrix (e.g. symmetric, banded etc.)

Acknowledgment

I would like to give my heartfelt thanks to my supervisor and friend Dr. Peter Aitchison, whose understanding, support and encouragement in both academic and spiritual aspects are a constant source of inspiration to his students. I am greatly indebted to him for his long time guidance, his knowledge, kindness and patience throughout the past two years. What I have done in this thesis is entirely based on an idea he suggested.

I am also very grateful to Dr. D. Meek who took many hours to read this thesis and made some precious comments. His comments gave me a good opportunity to review my past work on this thesis and to modify it better.

As well, I am very indebted to Dr. Larry Jordan for his support and understanding which let me finish this thesis sooner and better.

I thank from the bottom of my heart the Department of Applied Mathematics, University of Manitoba, for its two-year financial support which enabled me to complete the master program. I am very grateful to the International Center for Students, University of Manitoba, for its loan which helped to tide me over in the difficulty early period of my graduate study in the University of Manitoba. I deeply thank all friends who have helped me to adjust to this new society and to undertake my study in Canada.

Finally, I would like to dedicate this thesis to my best friend Gemai Chen, to whom I am forever grateful for his friendship.

Contents

Chapter I Introduction

§ 1. Purpose of the Thesis

We can roughly classify the methods of solving a linear system

$$A X = B \tag{1.1.1}$$

where $A \in R^{n\times n}$, $X, B \in R^{n\times 1}$

into two general classes. One class deals with the individual numbers of the matrix A and we might as well call these non-block methods. The other class treating the matrix A as being composed of blocks (sub-matrices) are called block methods. The block methods considered are essentially based on the non-block methods. In this chapter we briefly review some block methods which are used to solve a linear system by direct methods (rather than iteration, for example).

In general, the direct methods for block factorization have two common characteristics:

1° The process of solving the system is composed of two sequential steps:

1) Partition matrix A (or factor A into block forms)

2) Solve the linear system by sequentially solving a group of linear sub-systems by the non-block methods such as Gaussian elimination. Normally, there is no more block factorization for the sub matrices of A after the step 1 is done.

2° No any recursive procedure is involved in solving a linear system.

The block methods usually need to do a lot of operations to transform the matrix A to the sub blocks. As we shall review in the next section, the block Gaussian elimination requires that the sub-block A_{11} (see equation (1.2.15)) be non-singular. For this purpose a lot of permutations may be applied to A to get A_{11}. The block triangular method which we shall review in section 2.2 also needs to permute the matrix A to the block triangular form before the practical computation can be done for the solution of the linear system. Sometimes the above matrix permutations take up a lot of time to achieve the required block forms. Considering this fact it reasonable for us to develop another kind of block

methods which do the operations for matrix partition as little as possible and save the costs for doing the "pre-work" for the solution of a linear system.

On the other side the time consumption and memory occupation are very important two factors that usually determinate people to choose the effective algorithms to solve a linear system. Some computer systems have Cache installed, which offers a fast computation to small computational objectives. In general, ordinary methods such as Gassian elimination treat the linear systems in full size. They may lose the speed of computation in solving the large size linear systems because they can not use the Cache. For this reason if we could divide a linear system, the large size system in particular, into smaller sub systems and then solve them successively, we would take advantages from Cache. Therefore it is necessary for us to develop other block methods which solve a linear system by partitioning it into a group of sub systems with size as small as possible so that the Cache can be used to get a speedup in computation. Due to the application of block partition we also expect that the methods to be developed in this thesis can have some potential advantages for memory saving in solving the large size linear systems.

The purpose of the thesis is to establish a new algorithm which applies a recursive block partition method to the solution of linear systems, finds the solution with a precision as good as other ordinary methods can achieve, computes the results in a relatively high speed, and has the potential advantage for memory saving. We hope the method is applicable to practical problem. Towards this end we shall test the algorithm by comparing it with Gaussian elimination in four aspects:

1) Computation cost
2) Stability control
3) Computation precision
4) Memory use

In chapter II we give the detail analysis of the method and discuss some theoretical properties of the method. In chapter III we introduce other two methods which are special cases of the recursive block method and are used as a comparison with the new method we introduced in Chapter II. Test results and conclusion are given in chapter IV. Codes of the algorithms written in both Matlab and FORTRAN are attached in appendix.

§ 2. Background and Review of Literature

In this section we shall briefly review two types of methods for solving a linear system in block form. Before we discuss the details of the methods we review the method of LU-decomposition.

Suppose the matrix A has the factorization

$$A=LU \tag{1.2.1}$$

with $L=(l_{ij})_{n\times n}$, lower triangular with unit diagonal values and $U=(u_{ij})_{n\times n}$, upper triangular. (Methods for achieving these are described in any numerical analysis book such as [1],[2] and [7]). Thus LU-decomposition is unique if it exists. The linear system (1.1.1) is then equivalent to solving

$$LC=B \tag{1.2.2}$$

for the n-by-1 vector C and then solving

$$UX=C \tag{1.2.3}$$

for X, where $C=(c_i)_{n\times 1}$

Because $l_{kk}\neq 0$, then by forward substitution the (1.2.2) is readily found by

$$c_1=b_1/l_{11} \tag{1.2.4}$$

and each other element of C is found by successively applying the formula

$$c_k=(b_k-\sum_{j=1}^{k-1} l_{kj}c_j)/l_{kk} \qquad k=2,3,...,n \tag{1.2.5}$$

If $u_{kk}\neq 0$ for k=1,2,...,n, then back-substitution method gives the final solution by

$$x_n=c_n/u_{nn} \tag{1.2.6}$$

The other elements of X are given by applying the following formula successively for k= n-1, n-2, ..., 2, 1.

$$x_k = (c_k - \sum_{j=k+1}^{n} u_{kj} x_j) \qquad (1.2.7)$$

LU-decomposition methods may also involve row and column interchanges (pivoting) for stability. In this case the decomposition has the form

$$PAQ=LU \qquad (1.2.8)$$

where P and Q are permutation matrices (so that $PP^T=QQ^T=I$).
The solution of (1.1.1) is then equivalent to the solution of the following system since P is non-singular and $QQ^T=I$,

$$(PAQQ^T)X=PB \qquad (1.2.9)$$

And then we have

$$LU(Q^TX)=PB \qquad (1.2.10)$$

Therefore the solution can be achieved by solving the following systems sequentially

1) Compute $T=PB$ (1.2.11)
2) Solve for $S=(L)^{-1}PB$ by forward solution $LS=T$ (1.2.12)
3) Solve for $R=Q^TX$ by backward solution $UR=S$ (1.2.13)
4) Compute $X=QR$ (1.2.14)

§ 2.1 Block Gaussian Elimination

As Duff discussed in the book [1], a matrix A can be partitioned into four blocks in the form

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad (1.2.15)$$

where, A_{11} and A_{22} are square sub-matrices. The LU block factorization of A without pivoting has the form

$$A=\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}=\begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}\begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}=LU \qquad (1.2.16)$$

where L_{11} and L_{22} are lower triangular sub matrices and U_{11} and U_{22} are upper triangular sub matrices. Equating both sides of (1.2.16) gives

$$L_{11}U_{11}=A_{11} \qquad (1.2.17a)$$
$$L_{11}U_{12}=A_{12} \qquad (1.2.17b)$$
$$L_{21}U_{11}=A_{21} \qquad (1.2.17c)$$
$$L_{22}U_{22}=A_{22}-L_{21}U_{12}=\tilde{A}_{22} \qquad (1.2.17d)$$

Therefore the block factorization (1.2.16) can be achieved by the following steps:

1) Decompose A_{11} into LU form in (1.2.17a) to get L_{11} and U_{11}
2) Find the solution U_{12} to the multiple right hand side problem (1.2.17b)
3) Find the solution L_{21} to the multiple right hand side problem (1.2.17c)
4) Decompose $\tilde{A}_{22}$ into its LU forms as in (1.2.17d) to get L_{22} and U_{22}

If we choose L to be unit block triangular, then (1.2.16) becomes

$$A=\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}=\begin{bmatrix} I & \\ \tilde{L}_{21} & I \end{bmatrix}\begin{bmatrix} A_{11} & A_{12} \\ & \tilde{A}_{22} \end{bmatrix} \qquad (1.2.18)$$

Where I is an appropriate size identity matrix, and where $\tilde{L}_{21}A_{11}=A_{21}$, $\tilde{A}_{22}=A_{22}-\tilde{L}_{21}A_{12}$

If we choose U to be unit triangular, (1.2.16) has the form

$$A=\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}=\begin{bmatrix} A_{11} & \\ A_{21} & \tilde{A}_{22} \end{bmatrix}\begin{bmatrix} I & \tilde{U}_{12} \\ & I \end{bmatrix} \qquad (1.2.19)$$

where $A_{11}\tilde{U}_{12}=A_{12}$, $\tilde{A}_{22}=A_{22}-A_{21}\tilde{U}_{12}$

After a block factorization of A has been formed, the solution to the problem (1.1.1) can be obtained by organizing AX=B as L(UX) = B and sequentially solving

$$LY=B$$
$$UX=Y$$

We use the block form of (1.2.16) to demonstrate this process in more detail. For this purpose we divide

$$Y=\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \qquad X=\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

and sequentially solve

$$L_{11}y_1=B_1 \qquad \text{for } y_1$$
$$L_{22}y_2=B_2-L_{21}y_1 \qquad \text{for } y_2$$
$$U_{22}x_2=y_2 \qquad \text{for } x_2$$
$$U_{11}x_1=y_1-U_{12}x_2 \qquad \text{for } x_1$$

Each of the triangular systems can be simply solved by the forward substitution or back-substitution.

Note that in the above discussion we have implicitly assumed that sub-matrix A_{11} is nonsingular. The fact is that, however, A is nonsingular does not guarantee the non-singularity of A_{11}. See [1] for the detailed discussion.

G.H. Golub and C.F. Van Loan also discussed the block Gaussian elimination in their book [2]. J.R. Bunch [6] developed a similar block method for solving sparse linear system based on LU factorization for non symmetric matrix and LDL^T factorization for symmetric matrices. In this thesis we will develop a new method for LU factorization. It is every different from the methods that the above people developed and is not just for four blocks as shown in (1.2.16). The details of this method is discussed in section 8 of Chapter 2.

§ 2.2 Block Triangular Form

In this section we review another general block method using the block triangular form, described in detail in [1].

We consider permutations to the block lower triangular form for

$$PAQ = \begin{bmatrix} A_{11} & & & \\ A_{21} & A_{22} & & \\ \vdots & \vdots & \ddots & \\ A_{m1} & A_{m2} & \cdots & A_{mm} \end{bmatrix} \qquad (1.2.20)$$

where P and Q are permutation matrices.
A block upper triangular version of this can be readily obtained by a minor modification of P and Q as shown in [1]

A special case for the triangular factorization of A with form $A = \begin{bmatrix} A_{n-1} & a_n \\ 0 & a_{nn} \end{bmatrix}$ (where $A_{n-1} \in R^{(n-1)\times(n-1)}, a_n \in R^{(n-1)\times 1}$ and $a_{nn} \in R$) and solution to this linear system is discussed in [7].

We can use the block forms (1.2.20) to solve the linear system (1.1.1), AX=B, by changing it to the equivalent system

$$PAQQ^TX=PB$$

and setting

$$Q^TX=Y$$

so that

$$(PAQ)Y=PB$$

and then using block forward substitution sequentially to solve the following equations (1.2.21) for $y_1, y_2, \ldots y_m$

$$A_{ii}y_i = (PB)_i - \sum_{j=1}^{i-1} A_{ij}y_j \qquad (1.2.21)$$

Here the summation has no value for i=1 and y_i and $(PB)_i$ (i=1, ..., m) refer to the section of Y and PB corresponding in size and position to the block A_{ii} in PAQ

Finally we solve for X by

$$X = Q\,Y \qquad (1.2.22)$$

In the above process we need only to factor the diagonal blocks A_{ii} (i=1, ..., m) for solving (1.2.21). The off-diagonal blocks A_{ij}, i>j are used only in the multiplication $A_{ij}y_j$

There are several methods to permute the matrix A to the form (1.2.20). One of the methods is called two-stage approach [1] and can be described briefly as

1) Use row and column permutations to ensure that the diagonals are all non-zero (we call this process finding a transversal)
2) Use symmetric permutations to find the block form itself.

With the help of the digraphs Sargent and Westerberg [3] successfully established a algorithm based on symmetric permutations and the observation that the graphs of each A_{ii} must be strongly connected. Tarjan [4] [5] followed the work done by Sargent and Westerberg and improved the algorithm. The new algorithm reduce the amount of operations done for the matrix permutation.

The method of the block triangular form is useful for solving the sparse linear system. In that case, the form given by (1.2.20) is more likely to be obtained through the matrix permutation. Then solving a linear system in the form (1.2.20) would save a lot of computation costs than solving it in a full matrix form. We skip the details of discussion.

In this thesis we will develop another kind of block method for solving linear systems. Differing from the above method the new method uses a very little cost for matrix partition and solves a linear system by successively dividing it into 2-by-2 sub systems. It speeds up the computation for solving large size linear systems on the computer system with Cache.

Chapter II Recursive Block Method

Before we start our discussion we first give a brief explanation to the notation we used in this chapter. The superscript usually represents the iterate number. Sometimes it also implies the order of the sub matrices. We will give a special explanation in that case. The subscript represents the index number of the sub matrix or vector.

In this chapter we shall establish a recursive block method for solving a linear system. We start our discussion with an example in section 1. In section 2 we introduce the concept of the general block partition to a matrix. It is the basis of the recursive block methods. In sections 3 to 6 we establish the recursive block methods which we call the Strictly Symmetric Partition (SSP) method and the Left Strictly Symmetric Partition (LSSP) method. The stability of the algorithms is discussed in section 7. In section 8 we analyze the recursive block method in theory and show that it is essentially the same as Gaussian elimination with back-substitution. We also give an estimate of the number of flops involved in the algorithms in section 9. The technique of memory allocation for the algorithms is discussed in section 10. The algorithms are described by the flow charts in section 11.

§ 1. Introduction

Consider a linear system

$$A X = B \qquad (2.1.1)$$

$$(\text{where } A \in R^{n\times n}, X, B \in R^{n\times 1})$$

which we can solve by the usual Gaussian elimination method. We now develop an alternative solution method based on subdividing the problem into smaller sub problems. We will illustrate the formulae by applying them to the following example.

Consider

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 0 & 4 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \quad X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} \qquad (2.1.1a)$$

which has exact solution

$$X=[2,\ 1,\ -1,\ 0]^T$$

First we partition A into four blocks

$$A=\begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ A_{21}^{(1)} & A_{22}^{(1)} \end{bmatrix}$$

where

$$A_{11}^{(1)}=\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix},\ A_{12}^{(1)}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ A_{21}^{(1)}=\begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix},\ A_{22}^{(1)}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Correspondingly, B and X are respectively partitioned into two blocks

$$B=\begin{bmatrix} B_1^{(1)} \\ B_2^{(1)} \end{bmatrix} \quad \text{with } B_1^{(1)}=\begin{bmatrix} 1 \\ 2 \end{bmatrix},\ B_2^{(1)}=\begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

and

$$X=\begin{bmatrix} X_1^{(1)} \\ X_2^{(1)} \end{bmatrix} \quad \text{with } X_1^{(1)}=\begin{bmatrix} X_1 \\ X_2 \end{bmatrix},\ X_2^{(1)}=\begin{bmatrix} X_3 \\ X_4 \end{bmatrix}$$

Then (2.1.1) can be rewritten after carrying out the block multiplication of AX and equating the individual blocks as:

$$A_{11}^{(1)}X_1^{(1)}+A_{12}^{(1)}X_2^{(1)}=B_1^{(1)} \qquad (2.1.2a)$$

$$A_{21}^{(1)}X_1^{(1)}+A_{22}^{(1)}X_2^{(1)}=B_2^{(1)} \qquad (2.1.2b)$$

Assume that $A_{11}^{(1)}$ is non singular, then from (2.1.2a) we have

$$X_1^{(1)}=\left[A_{11}^{(1)}\right]^{-1}\left(B_1^{(1)}-A_{12}^{(1)}X_2^{(1)}\right) \qquad (2.1.2c)$$

Inserting this into (2.1.2b) gives

$$\left(A_{22}^{(1)}-A_{21}^{(1)}\left[A_{11}^{(1)}\right]^{-1}A_{12}^{(1)}\right)X_2^{(1)}=B_2^{(1)}-A_{21}^{(1)}\left[A_{11}^{(1)}\right]^{-1}B_1^{(1)} \qquad (2.1.2d)$$

Set

$$\left[A_{11}^{(1)}\right]^{-1}A_{12}^{(1)}=Z^{(1)} \qquad (2.1.2e)$$

$$\left[A_{11}^{(1)}\right]^{-1}B_1^{(1)}=Y^{(1)} \qquad (2.1.2f)$$

and substitute (2.1.2e) and (2.1.2f) into (2.1.2d) and (2.1.2c) we obtain

$$(A_{22}^{(1)} - A_{21}^{(1)} Z^{(1)}) X_2^{(1)} = B_2^{(1)} - A_{21}^{(1)} Y^{(1)} \qquad (2.1.2g)$$

$$X_1^{(1)} = Y^{(1)} - Z^{(1)} X_2^{(1)} \qquad (2.1.2h)$$

We write (2.1.2e) and (2.1.2f) as

$$A_{11}^{(1)} Z^{(1)} = A_{12}^{(1)} \qquad (2.1.2i)$$

$$A_{11}^{(1)} Y^{(1)} = B_1^{(1)} \qquad (2.1.2j)$$

then solving (2.1.2a-b) for X is equivalent to solving the following systems:

1) Solving (2.1.2i) for $Z^{(1)}$
2) Solving (2.1.2j) for $Y^{(1)}$
3) Solving (2.1.2g) for $X_2^{(1)}$
4) Computing (2.1.2h) for $X_1^{(1)}$

We write these systems in one group

$$A_{11}^{(1)} Z^{(1)} = A_{12}^{(1)} \qquad (2.1.3a)$$

$$A_{11}^{(1)} Y^{(1)} = B_1^{(1)} \qquad (2.1.3b)$$

$$(A_{22}^{(1)} - A_{21}^{(1)} Z^{(1)}) X_2^{(1)} = B_2^{(1)} - A_{21}^{(1)} Y^{(1)} \qquad (2.1.3c)$$

$$X_1^{(1)} = Y^{(1)} - Z^{(1)} X_2^{(1)} \qquad (2.1.3d)$$

$$X^{(1)} = (X_1^{(1)}, X_2^{(1)})^T \qquad (2.1.3e)$$

where $Z^{(1)} \in R^{2\times 2}$, $Y^{(1)} \in R^{2\times 1}$ in the case of example (2.1.1a).

The system (2.1.3a) is a linear system with multiple right hand sides. For conciseness we re-write (2.1.3a) and (2.1.3b) as $A_{11}^{(1)}[Z^{(1)}, Y^{(1)}] = [A_{12}^{(1)}, B_1^{(1)}]$, then (2.1.3) can be expressed as

$$A_{11}^{(1)}[Z^{(1)}, Y^{(1)}] = [A_{12}^{(1)}, B_1^{(1)}] \qquad (2.1.4a)$$

$$(A_{22}^{(1)} - A_{21}^{(1)} Z^{(1)}) X_2^{(1)} = B_2^{(1)} - A_{21}^{(1)} Y^{(1)} \qquad (2.1.4b)$$

$$X_1^{(1)} = Y^{(1)} - Z^{(1)} X_2^{(1)} \qquad (2.1.4c)$$

$$X^{(1)} = (X_1^{(1)}, X_2^{(1)})^T \qquad (2.1.4d)$$

We refer to (2.1.4a-d) as **recursive formula** for systems (2.1.2a-b). Now we can apply the above process again to the equation solutions of (2.1.4a) and (2.1.4b) as follows.

Firstly, in order to solve (2.1.4a) for $Z^{(1)}$ and $Y^{(1)}$ we create the block forms,

$$Z^{(1)} = \begin{bmatrix} Z_{11}^{(1)} & Z_{12}^{(1)} \\ Z_{21}^{(1)} & Z_{22}^{(1)} \end{bmatrix} \quad \text{and} \quad Y^{(1)} = \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix}$$

and for simplicity we set

$$A^{(2)} = A_{11}^{(1)}$$

$$X^{(2)} = \begin{bmatrix} Z^{(1)} & Y^{(1)} \end{bmatrix} = \begin{bmatrix} Z_{11}^{(1)} & Z_{12}^{(1)} & y_1^{(1)} \\ Z_{21}^{(1)} & Z_{22}^{(1)} & y_2^{(1)} \end{bmatrix}$$

$$B^{(2)} = \begin{bmatrix} A_{12}^{(1)}, B_1^{(1)} \end{bmatrix}$$

then (2.1.4a) becomes

$$A^{(2)} X^{(2)} = B^{(2)} \tag{2.1.5}$$

where $A^{(2)} \in R^{2\times 2}$, $X^{(2)}, B^{(2)} \in R^{2\times 3}$ in the case of example (2.1.1a).

If we partition sub matrix $A^{(2)}$ into four blocks and correspondingly $X^{(2)}$ and $B^{(2)}$ into two blocks we obtain in the case of example (2.1.1a)

$$A^{(2)} = \begin{bmatrix} A_{11}^{(2)} & A_{12}^{(2)} \\ A_{21}^{(2)} & A_{22}^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \tag{2.1.6}$$

where $A_{11}^{(2)} = 1$, $A_{12}^{(2)} = 0$, $A_{21}^{(2)} = 0$, $A_{22}^{(2)} = 2$

$$X^{(2)} = \begin{bmatrix} X_1^{(2)} \\ X_2^{(2)} \end{bmatrix} = \begin{bmatrix} Z_{11}^{(1)} & Z_{12}^{(1)} & y_1^{(1)} \\ Z_{21}^{(1)} & Z_{22}^{(1)} & y_2^{(1)} \end{bmatrix} \tag{2.1.7}$$

where $X_1^{(2)} = \begin{bmatrix} Z_{11}^{(1)} & Z_{12}^{(1)} & y_1^{(1)} \end{bmatrix}$, $\quad X_2^{(2)} = \begin{bmatrix} Z_{21}^{(1)}, Z_{22}^{(1)}, y_2^{(1)} \end{bmatrix}$

$$B^{(2)} = \begin{bmatrix} B_1^{(2)} \\ B_2^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} \tag{2.1.8}$$

where $B_1^{(2)} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$, $\quad B_2^{(2)} = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix}$. Then the equivalent equations of (2.1.5) are

$$A_{11}^{(2)} X_1^{(2)} + A_{12}^{(2)} X_2^{(2)} = B_1^{(2)} \tag{2.1.9a}$$

$$A_{21}^{(2)} X_1^{(2)} + A_{22}^{(2)} X_2^{(2)} = B_2^{(2)} \tag{2.1.9b}$$

Similar to (2.1.4), the recursive formula for (2.1.9) has the form

$$A_{11}^{(2)}[Z^{(2)}, Y^{(2)}] = [A_{12}^{(2)}, B_1^{(2)}] \tag{2.1.10a}$$
$$(A_{22}^{(2)} - A_{21}^{(2)} Z^{(2)}) X_2^{(2)} = B_2^{(2)} - A_{21}^{(2)} Y^{(2)} \tag{2.1.10b}$$
$$X_1^{(2)} = Y^{(2)} - Z^{(2)} X_2^{(2)} \tag{2.1.10c}$$
$$X^{(2)} = (X_1^{(2)}, X_2^{(2)})^T \tag{2.1.10d}$$

where $Z^{(2)} \in R$, $Y^{(2)} \in R^{1\times 3}$

Now $A_{ij}^{(2)}$ are numbers, there will be no more partition applicable to them. For $A_{11}^{(2)} \neq 0$, solving (2.1.10a) gives

$$[Z^{(2)}, Y^{(2)}] = [A_{12}^{(2)}, B_1^{(2)}] / A_{11}^{(2)} = [0 \quad 1 \quad 0 \quad 1] \tag{2.1.11a}$$

Inserting (2.1.8) and (2.1.11a) into (2.1.10b) gives

$$2X_2^{(2)} = [0,1,2]$$

or

$$X_2^{(2)} = [0, 1/2, 1] \tag{2.1.11b}$$

Substituting (2.1.11a) and (2.1.11b) into (2.1.10c) gives $X_1^{(2)} = [1, 0, 1]$. Then by (2.1.7) we obtain

$$Z^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1/2 \end{bmatrix}, \quad Y^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.1.12}$$

With reference to the recursive formula (2.1.4), so far we have found $[Z^{(1)}, Y^{(1)}]$ by solving (2.1.4a), our next work is to solve the sub system (2.1.4b), for this purpose we firstly insert $Z^{(1)}$ and $Y^{(1)}$ into (2.1.4b) and then use the simplified notation

$$\tilde{A}^{(2)} = A_{22}^{(1)} - A_{21}^{(1)} Z^{(1)} \tag{2.1.13}$$
$$\tilde{X}^{(2)} = X_2^{(1)} \tag{2.1.14}$$
$$\tilde{B}^{(2)} = B_2^{(1)} - A_{21}^{(1)} Y^{(1)} \tag{2.1.15}$$

Here (2.1.4b) becomes

$$\tilde{A}^{(2)} \tilde{X}^{(2)} = \tilde{B}^{(2)} \tag{2.1.16}$$

where

$$\tilde{A}^{(2)}=\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix},\ \tilde{B}^{(2)}=\begin{bmatrix}1\\ 0\end{bmatrix},\ \tilde{X}^{(2)}=\begin{bmatrix}\tilde{X}_1^{(2)}\\ \tilde{X}_2^{(2)}\end{bmatrix}$$

Again, in order to solve (2.1.16) for $\tilde{X}^{(2)}$ we partition $\tilde{A}^{(2)}$ into four blocks and $\tilde{X}^{(2)}$ and $\tilde{B}^{(2)}$ into two blocks, then we obtain the equivalent equations of (2.1.16) below

$$\tilde{A}_{11}{}^{(2)}\tilde{X}_1^{(2)}+\tilde{A}_{12}{}^{(2)}\tilde{X}_2^{(2)}=\tilde{B}_1^{(2)} \qquad (2.1.17a)$$
$$\tilde{A}_{21}{}^{(2)}\tilde{X}_1^{(2)}+\tilde{A}_{22}{}^{(2)}\tilde{X}_2^{(2)}=\tilde{B}_2^{(2)} \qquad (2.1.17b)$$

where

$$\tilde{A}^{(2)}=\begin{bmatrix}\tilde{A}_{11}{}^{(2)} & \tilde{A}_{12}{}^{(2)}\\ \tilde{A}_{21}{}^{(2)} & \tilde{A}_{22}{}^{(2)}\end{bmatrix}=\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}$$

$$\tilde{B}^{(2)}=\begin{bmatrix}\tilde{B}_1^{(2)}\\ \tilde{B}_2^{(2)}\end{bmatrix}=\begin{bmatrix}1\\ 0\end{bmatrix},\ \text{and}\ \tilde{X}^{(2)}=\begin{bmatrix}\tilde{X}_1^{(2)}\\ \tilde{X}_2^{(2)}\end{bmatrix}$$

The recursive formula for (2.1.17) is

$$\tilde{A}_{11}{}^{(2)}\left[\tilde{Z}^{(2)},\tilde{Y}^{(2)}\right]=\left[\tilde{A}_{12}{}^{(2)},\tilde{B}_1^{(2)}\right] \qquad (2.1.18a)$$
$$(\tilde{A}_{22}{}^{(2)}-\tilde{A}_{21}{}^{(2)}\tilde{Z}^{(2)})\tilde{X}_2^{(2)}=\tilde{B}_2^{(2)}-\tilde{A}_{21}{}^{(2)}\tilde{Y}^{(2)} \qquad (2.1.18b)$$
$$\tilde{X}_1^{(2)}=\tilde{Y}^{(2)}-\tilde{Z}^{(2)}\tilde{X}_2^{(2)} \qquad (2.1.18c)$$
$$\tilde{X}^{(2)}=\left[\tilde{X}_1^{(2)},\tilde{X}_2^{(2)}\right]^T \qquad (2.1.18d)$$

where, $\tilde{Z}^{(2)}=[\tilde{Z}_1^{(2)}]\in R$, $\tilde{Y}^{(2)}=[\tilde{Y}_1^{(2)}]\in R$

For $\tilde{A}_{11}{}^{(2)}\neq 0$, solving (2.1.18a) gives

$$\left[\tilde{Z}^{(2)},\tilde{Y}^{(2)}\right]=\left[\tilde{A}_{12}{}^{(2)},\tilde{B}_1^{(2)}\right]/\tilde{A}_{11}{}^{(2)}=[0,-1] \qquad (2.1.19)$$

Inserting (2.1.19) into (2.1.18b) and solving it we obtain

$$\tilde{X}_2{}^{(2)}=0 \qquad (2.1.20)$$

And then substituting (2.1.19) and (2.1.20) into (2.1.18c) gives $\tilde{X}_1^{(2)}=-1$.
By (2.1.18d) we have $\tilde{X}^{(2)}=[-1,\ 0]^T$ and go back to (2.1.14) we have

$$X_2^{(1)} = \tilde{X}^{(2)} = [-1,0]^T \qquad (2.1.21)$$

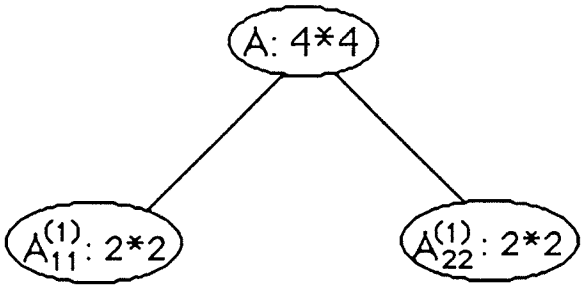
Let's go back to the recursive (2.1.4), we substitute (2.1.21) and (2.1.12) into (2.1.4c)

$$X_1^{(1)} = Y^{(1)} - Z^{(1)} X_2^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

and by (2.1.4d) finally we obtain $X = [X_1^{(1)}, X_2^{(1)}]^T = [2, 1, -1, 0]^T$. This is the same as the exact solution found by Gaussian elimination.

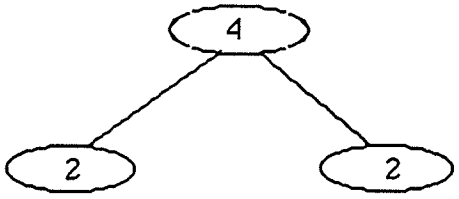
In the above process for solving a linear system we note that the most important step is the use of recursive formula. That is, we recursively apply (2.1.4a-d) to each of linear sub system until the sub matrix on diagonal is divided into 1-by-1 matrix. Then we simply solve the system and substitute its solution to the same level or upper level linear sub system, to which we may apply a deep level partition again if the order of the sub matrix is greater than one. We repeat this process until the all 1-by-1 linear sub systems are solved sequentially.

We refer to the above method for solving a linear system as the **recursive block method**. For clarity we introduce a binary tree to represent the structure of a matrix which is partitioned into four blocks. In the above example the matrix A of order 4 is divided into four blocks $A_{11}^{(1)}, A_{12}^{(1)}, A_{21}^{(1)}$ and $A_{22}^{(1)}$, each of size 2-by-2. We use only $A_{11}^{(1)}$ and $A_{22}^{(1)}$ to represent the partition $A = \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ A_{21}^{(1)} & A_{22}^{(1)} \end{bmatrix}$, that is



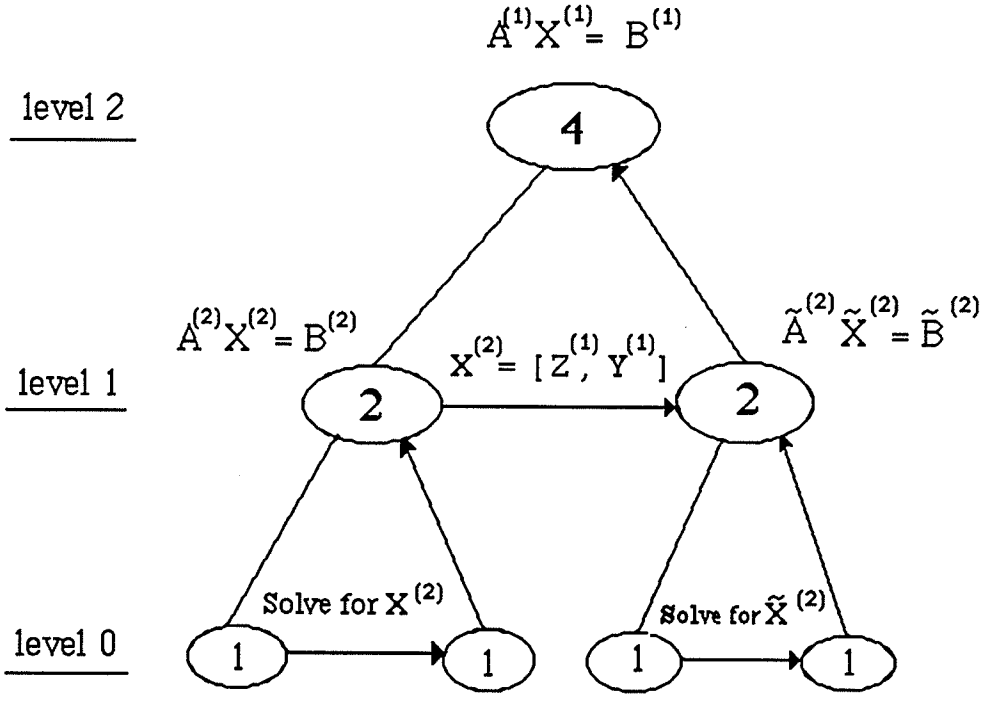
[Fig 2.1.1a]

or more simply as



[Fig 2.1.1b]

We call the branch with $A_{11}^{(1)}$ the left branch and the branch with $A_{22}^{(1)}$ the right branch relative to the upper level where A is on. The number on each node represents the size of each matrix at that node. In above example, $A_{11}^{(1)}$ is partitioned again for solving for $[Z^{(1)}, Y^{(1)}]$ and $A_{22}^{(1)}$ is replaced by $(A_{22}^{(1)} - A_{21}^{(1)}Z^{(1)})$ and then partitioned again for solving $\tilde{X}^{(2)}$. The process of solving (2.1.1) can be expressed by the following binary tree. The arrow indicates the computation sequence or 'direction'



[Fig 2.1.2]

On each step for partitioning matrix or sub matrices into four blocks, the recursive formula (2.1.4 a-d) is generalized to

$$(RF-k)\begin{cases} A_{11}^{(k)}[Z^{(k)},Y^{(k)}]=[A_{12}^{(k)},B_1^{(k)}] & (a-k)\\ (A_{22}^{(k)}-A_{21}^{(k)}Z^{(k)})X_2^{(k)}=B_2^{(k)}-A_{21}^{(k)}Y^{(k)} & (b-k)\\ X_1^{(k)}=Y^{(k)}-Z^{(k)}X_2^{(k)} & (c-k)\\ X^{(k)}=(X_1^{(k)},X_2^{(k)})^T & (d-k)\end{cases} \qquad (2.1.22)$$

where $A^{(k)}=\begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)}\end{bmatrix}$ $B^{(k)}=\begin{bmatrix} B_1^{(k)} \\ B_2^{(k)}\end{bmatrix}$ $X^{(k)}=\begin{bmatrix} X_1^{(k)} \\ X_2^{(k)}\end{bmatrix}$. $Z^{(k)}$ and $Y^{(k)}$ have the same sizes as $A_{12}^{(k)}$ and $B_1^{(k)}$have)

We refer to diagonal sub matrices $A_{11}^{(k)}$ and $A_{22}^{(k)}$ as the matrices at the same level if they are generated by the recursive formula (RF-k)~(2.1.22) with the same index number k. In Figure 2.1.2 there are four 1-by-1 sub matrices at level 0, two 2-by-2 matrices at level 1 and one 4-by-4 matrix on the top, level 3.

We noted that a 2-by-2 matrix is the smallest matrix which can be partitioned into four blocks, in other words a 2-by-2 matrix is the smallest matrix unit to which a recursive formula can be applied. We have the following definitions.

Definition The recursive formula (2.1.22) is called the **unit solution** if it is applied to the solution of a 2-by-2 linear system.

We denote it by the Unit Solution or simply by **US.** Computation process for the US can be described as four steps (note that $A_{ij}^{(k)}$ (i,j=1,2) are numbers for the operation US)

1) If $A_{11}^{(k)}\neq 0$, compute $[Z^{(k)},Y^{(k)}]=[A_{12}^{(k)},B_1^{(k)}]/A_{11}^{(k)}$
2) If $A_{22}^{(k)}-A_{21}^{(k)}Z^{(k)}\neq 0$, compute $X_2^{(k)}=(B_2^{(k)}-A_{21}^{(k)}Y^{(k)})/(A_{22}^{(k)}-A_{21}^{(k)}Z^{(k)})$
3) Compute $X_1^{(k)}=Y^{(k)}-Z^{(k)}X_2^{(k)}$
4) Form $X^{(k)}=(X_1^{(k)},X_2^{(k)})^T$

Definition The following two steps of the process are called **transfer**

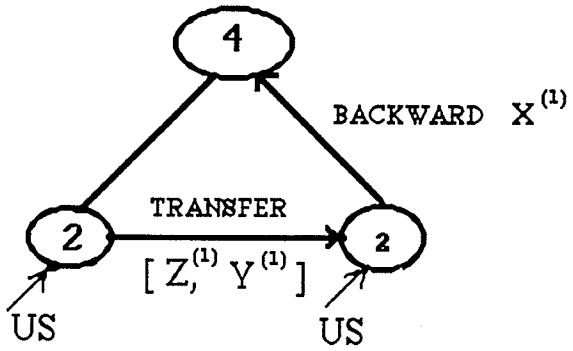
1) $\tilde{A}^{(k)}=A_{22}^{(k)}-A_{21}^{(k)}Z^{(k)}$
2) $\tilde{B}^{(k)}=B_2^{(k)}-A_{21}^{(k)}Y^{(k)}$

where $A_{21}^{(k)}$, $A_{22}^{(k)}$, $Z^{(k)}$and $Y^{(k)}$ are sub matrices

The meaning of transfer is to 'send' (or substitute) the solution $[Z^{(k)}, Y^{(k)}]$ found by solving equation (a-k) in the recursive formula (RF-k)~(2.1.22) to the equation (b-k) for forming a new linear sub system $\tilde{A}^{(k)}\tilde{X}^{(k)}=\tilde{B}^{(k)}$. We denote this in the following by **TRANSFER**.

Definition We refer to the computation $X_1^{(k)}=Y^{(k)}-Z^{(k)}X_2^{(k)}$ with the formation of $X_2^{(k-1)}=X^{(k)}=(X_1^{(k)},X_2^{(k)})^T$ as **back-substitution**, where $Y^{(k)}$, $Z^{(k)}$ and $X_2^{(k)}$ are sub matrices. For clarity we denote the back substitution by **BACKWARD**.

We call the Unit Solution(US), TRANSFER and BACKWARD the **three basic operations**. With usage of these operations, Fig 2.1.2 can be simplified as



[Fig. 2.1.3]

The pattern of the matrix A corresponding to the partition shown in Fig 2.1.2 and Fig 2.1.3 is

$$[A^{(1)} \quad , \quad B^{(1)}]=\begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & B_1^{(1)} \\ A_{21}^{(1)} & A_{22}^{(1)} & B_2^{(1)} \end{bmatrix}=\begin{bmatrix} [A^{(2)}]_{2*2} & [\quad B^{(2)} \quad]_{2*3} \\ [A_{21}^{(1)}]_{2*2} & [\tilde{A}^{(2)}]_{2*2} \quad [\tilde{B}^{(2)}]_{2*1} \end{bmatrix}_{4*5}$$

[Fig. 2.1.4]

Summarizing the above, we conclude that the process of solving a linear system by the recursive block method includes two major steps. The first step is the matrix partition which involves repeated use of the recursive formula (2.1.22). This process will stop as soon as the original matrix is divided into sub matrices of size 2-by-2. Very little computation is involved in this step and very little extra memory is used. The second step is

to solve the all 2-by-2 systems with multiple right hand sides. According to the binary tree, the sub systems are solved following the computation sequence

1) from the bottom level to top level;
2) from the left branch to the right branch.

In the following sections we shall systematically discuss the recursive block method for square matrices of any size.

§ 2. General Block Partition

In this section we consider a general case for a linear system

$$A X = B \tag{2.2.1}$$

where $A = A^{(1)} \in R^{(n\times n)}$, $X = X^{(1)} \in R^{(n\times 1)}$ and $B = B^{(1)} \in R^{(n\times 1)}$

Corresponding to the partition of A, B and X

$$A^{(k)} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix}, X^{(k)} = \begin{bmatrix} X_1^{(k)} \\ X_2^{(k)} \end{bmatrix} \text{ and } B^{(k)} = \begin{bmatrix} B_1^{(k)} \\ B_2^{(k)} \end{bmatrix}, \tag{2.2.2}$$

the recursive formula (RF-k)~(2.1.22) has the form as shown in (2.1.22). Note, however, that the blocks are not necessary the same size.

For example, for k=1, we define an integer S_1 which satisfies $0<S_1<n$, and apply a block partition to $A^{(1)}$ such that

$$A_{11}^{(1)} \in R^{S_1\times S_1}, \quad A_{12}^{(1)} \in R^{S_1\times Q_1}$$

$$A_{21}^{(1)} \in R^{Q_1\times S_1}, \quad A_{22}^{(1)} \in R^{Q_1\times Q_1}$$

and $X_1^{(1)}, B_1^{(1)} \in R^{S_1\times 1}$, $X_2^{(1)}, B_2^{(1)} \in R^{Q_1\times 1}$ where, $Q_1=n-s_1$

In recursive formula (RF-1)~(2.1.22), equation (a-1) defines a Q_1+1 RHS problem with size S_1-by-S_1 while equation (b-1) defines a linear sub system with size $(n-S_1)$-by-$(n-S_1)$. After (a-1) has been solved for $[Z^{(1)},Y^{(1)}]$, substitute those values into (b-1) to obtain a new reduced sub linear system

$$A^{(2)} X^{(2)} = B^{(2)} \tag{2.2.3}$$

where

$$A^{(2)} = A_{22}^{(1)} - A_{21}^{(1)} Z^{(1)} \in R^{Q_1\times Q_1}$$
$$X^{(2)} = X_2^{(1)} \in R^{Q_1\times 1}$$
$$B^{(2)} = B_2^{(1)} - A_{21}^{(1)} Y^{(1)} \in R^{Q_1\times 1}$$

For k=2, we define an integer S_2 which satisfies $0<S_2<(n-S_1)$ and apply a block partition to $A^{(2)}$ such that

$$A_{11}^{(2)} \in R^{S_2 \times S_2} \qquad A_{12}^{(2)} \in R^{S_2 \times Q_2}$$
$$A_{21}^{(2)} \in R^{Q_2 \times S_2} \qquad A_{22}^{(2)} \in R^{Q_2 \times Q_2}$$
$$X_1^{(2)}, B_1^{(2)} \in R^{S_2 \times 1} \qquad X_2^{(2)}, B_2^{(2)} \in R^{Q_2 \times 1} \qquad \text{where } Q_2 = n - s_1 - s_2$$

In recursive formula (RF-2)-(2.1.22), after equation (a-2) has been solved for $[Z^{(2)}, Y^{(2)}]$, substitute these values into (b-2) to get another reduced linear sub system

$$A^{(3)} X^{(3)} = B^{(3)} \tag{2.2.3}$$

where

$$A^{(3)} = A_{22}^{(2)} - A_{21}^{(2)} Z^{(2)} \in R^{(n-S_1-S_2)\times(n-S_1-S_2)}$$
$$X^{(3)} = X_2^{(2)} \in R^{(n-S_1-S_2)\times 1}$$
$$B^{(3)} = B_2^{(2)} - A_{21}^{(2)} Y^{(2)} \in R^{(n-S_1-S_2)\times 1}$$

Repeat the above process until we have obtained a reduced linear sub system

$$A^{(m+1)} X^{(m+1)} = B^{(m+1)} \tag{2.2.4}$$

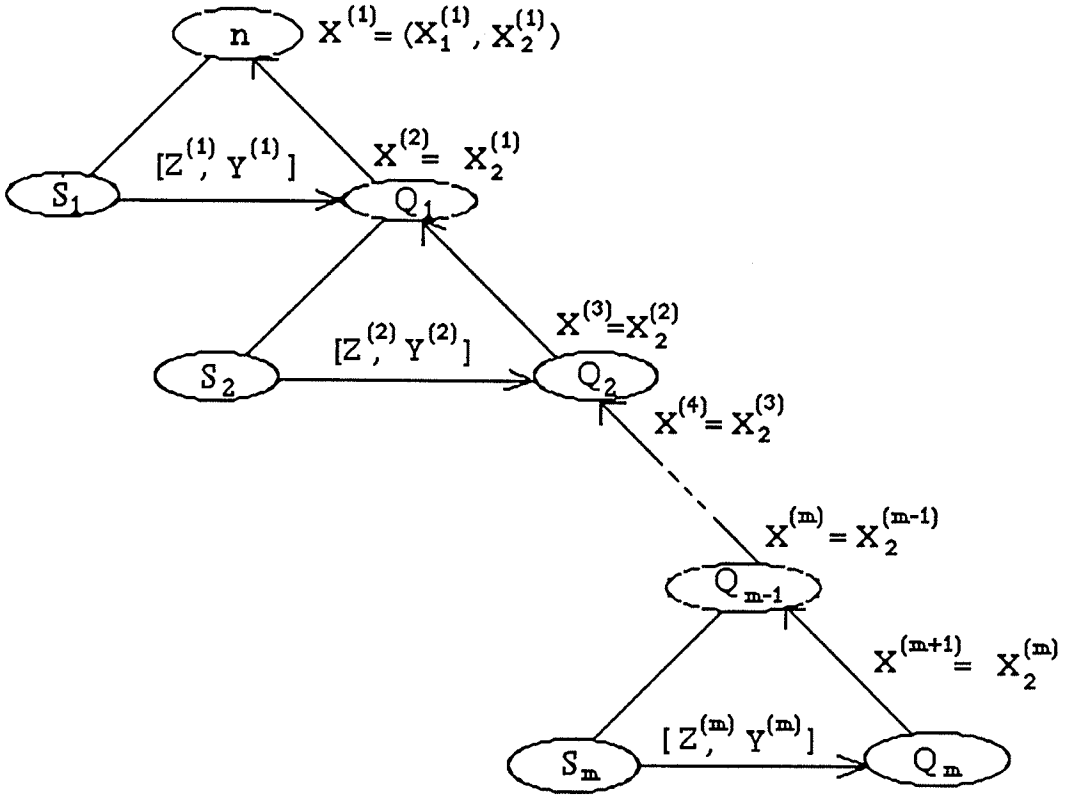
where

$$A^{(m+1)} = A_{22}^{(m)} - A_{21}^{(m)} Z^{(m)} \in R^{Q_m \times Q_m}$$
$$X^{(m+1)} = X_2^{(m)} \in R^{Q_m \times 1}$$
$$B^{(m+1)} = B_2^{(m)} - A_{21}^{(m)} Y^{(m)} \in R^{Q_m \times 1}$$

and
$$Q_m = (n - \sum_{k=1}^{m} S_k) > 0 \tag{2.2.5}$$

Assume equation (2.2.4) is solved for $X^{(m+1)} = X_2^{(m)}$, then from the recursive formula (RF-m)-(2.1.22), $X^{(m)} = (X_1^{(m)}, X_2^{(m)})^T$ can be found by equations (c-m) and (d-m). Again, from the recursive formula (RF-(m-1)), $X^{(m-1)}$ can be found by equations (c-(m-1)) and (d-(m-1)). Repeat this process of back-substitution, finally $X^{(1)} = (X_1^{(1)}, X_2^{(1)})^T$ is found by equations (c-1) and (d-1) in the recursive formula (RF-1). Then problem (2.2.1) is solved.

In the above derivation, S_i (i=1, ..., m) are any arbitrary fixed integers which satisfy (2.2.5), for this reason we refer to the above method of solving a linear system (2.2.1) as the **general block partition (GBP) method**. This process can be illustrated by a binary tree. See Fig. 2.2.1a.



[Fig. 2.2.1a]

Each of the systems S_1, S_2, ..., S_m of Fig 2.2.1a would be solved by the same approach and therefore will have a similar binary tree attached to it, which we have not shown. The details of this process are discussed in the next section.

The matrix pattern corresponding to the partition of Fig. 2.2.1a is shown in Fig.2.2.1b

$$A = \begin{bmatrix} [S_1 * S_1] & & & & \\ & [S_2 * S_2] & & & \\ & & \begin{bmatrix} \ddots & & \\ & [S_m * S_m] & \\ & & [Q_m * Q_m] \end{bmatrix} & & \end{bmatrix}$$

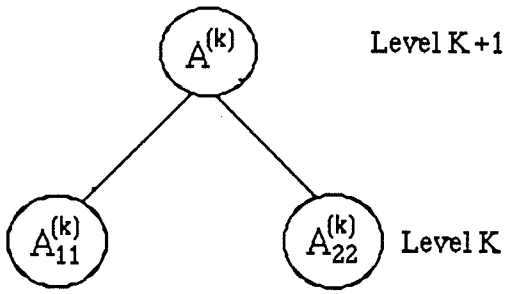
[Fig. 2.2.1b]

§ 3. Strictly Symmetrical Partition (SSP)

In section 1, we regard a 2-by-2 matrix as the smallest matrix to which the recursive formula (RF-k)~(2.1.22) can be applied. We referred to the solution found by applying a recursive formula to a 2-by-2 linear sub system as the Unit Solution (US). In last section, we just assumed that each linear sub system defined by the equation (a-k) in the recursive formula (RF-k)~(2.1.22) was solved for $[Z^{(k)},Y^{(k)}]$, but we didn't describe the details of how to solve these systems. In this section, we try to solve these systems by applying the US to them as we did in the example of section 1. For this purpose we have to partition each sub matrix with size S_i-by-S_i into four blocks and to each of the diagonal blocks we repeat this matrix partition process until all the diagonal sub blocks are partitioned into 2-by-2 sub matrices to which the operation US can be applied. The results of the process depend on the choice of the subdivision in each case (i.e. the choice of $S_1, S_2, ..., S_m$). We introduce a practical method which we call strictly symmetrical partition.

For any partition of matrix $A^{(k)}$:

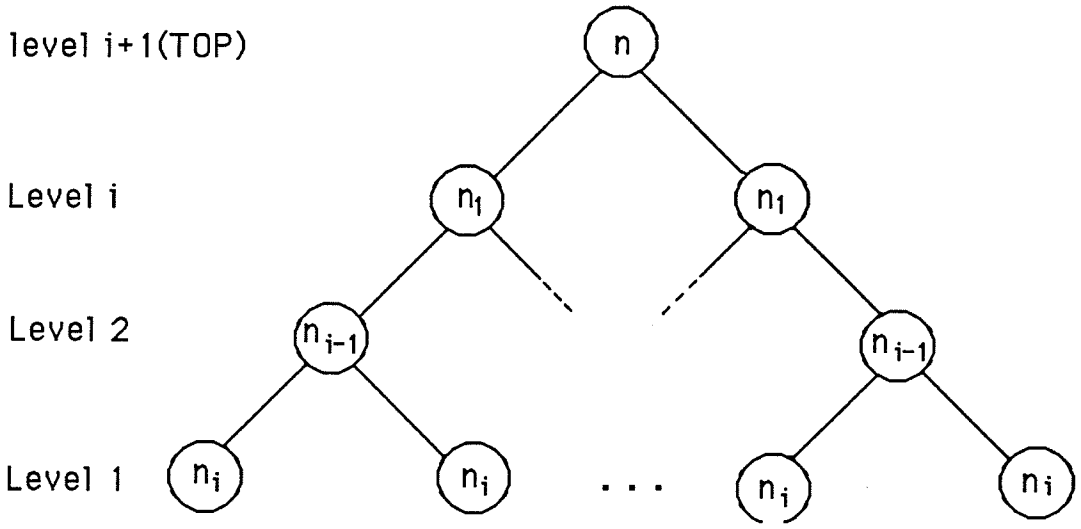
$$A^{(k)} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix}$$



[Fig. 2.3.1]

the partition applied to it is called a **symmetric partition at level k** if the size of $A_{11}^{(k)}$ is equal to the size of $A_{22}^{(k)}$. The symmetric partition at level k to the matrix $A^{(k)}$ implies that the four sub matrices at level k are square matrices of the same size.

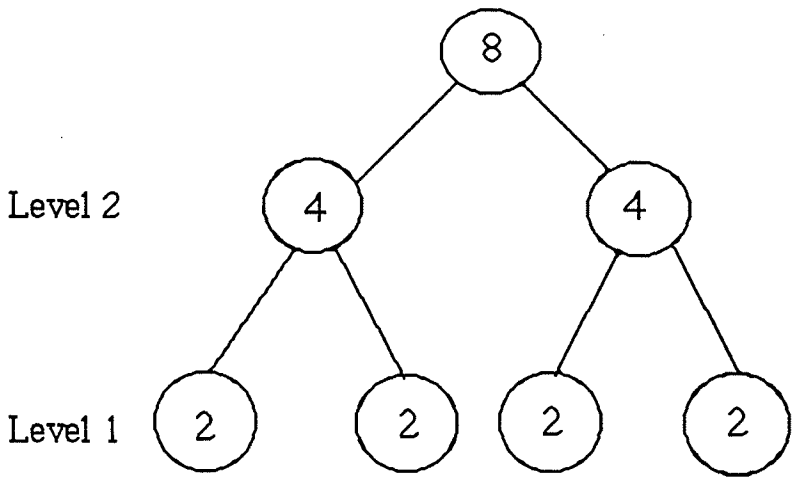
A partition to matrix A is called a **strictly symmetric partition (SSP)** if it is symmetrical at all levels. This implies that all sub matrices at all levels are square.



[Fig. 2.3.2]

A general block partition (GBP) is a symmetric partition if $n=2m_1$ and $S_1=m_1$ where, m_1 is a positive integer. A further symmetric partition can be applied to these two sub blocks at a deeper level if $m_1=2m_2$. Obviously, a matrix can be partitioned into strictly symmetrical blocks right down to 2-by-2's only if its size $n=2^i$.

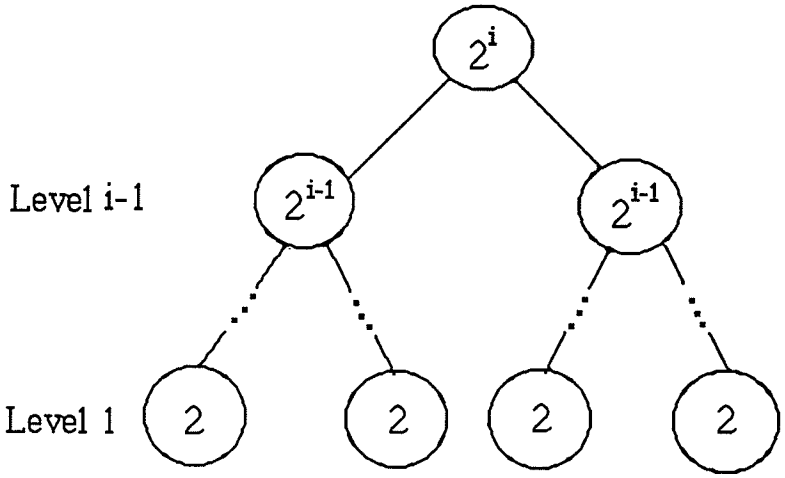
For example, when $n=2^3=8$, the SSP to this matrix has the form below.



[Fig. 2.3.2]

§ 4. The SSP Method for Solving a Linear System

Consider problem (2.2.1) with the assumption that $n= 2^i$ where the integer $i \geq 1$, the SSP applied to matrix A gives forms below.



[Fig. 2.4.1a]

The matrix is partitioned into the form

$$A = \begin{bmatrix} \begin{bmatrix} [2\times2] & & \\ & \ddots & \\ & & [2\times2] \end{bmatrix}_{2^{i-1}\times 2^{i-1}} & \begin{bmatrix} [2\times2] & & \\ & \ddots & \\ & & [2\times2] \end{bmatrix}_{2^{i-1}\times 2^{i-1}} \end{bmatrix}_{n\times n}$$

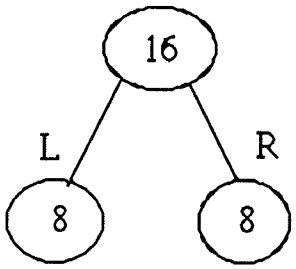
[Fig. 2.4.1b]

As we discussed in section 1 the recursive block partition method is composed of two sequential steps, the first step is matrix partition and the second step is the computation for solving a system. In this section we shall establish an algorithm for the SSP method, and illustrate the method with an example.

Let $A=A^{(1)}$ be a 16-by-16 matrix and $B=B^{(1)}$ is a length 16 vector, that is, $A^{(1)} \in R^{16*16}$ and $B^{(1)} \in R^{16*1}$. To partition A into SSP blocks we divide A into 4 blocks of size 8-by-8. Correspondingly X and B are divided into two blocks, each of length 8. Fig 2.4.2 shows the matrix pattern and binary tree for the first degree partition of A and B.

$$[A^{(1)},B^{(1)}]=\begin{bmatrix} [A_{11}^{(1)}:8*8] & [A_{12}^{(1)}:8*8] & [B_1^{(1)}:8*1] \\ [A_{21}^{(1)}:8*8] & [A_{22}^{(1)}:8*8] & [B_2^{(1)}:8*1] \end{bmatrix}_{16*17}$$

[Fig. 2.4.2a]



[Fig. 2.4.2b]

Corresponding to the recursive formula (RF-k)~(2.1.22) for k=1, partition shown in Fig 2.4.2 defines two multiple RHS linear sub systems, that is

$$A_{11}^{(1)}[Z^{(1)},Y^{(1)}]=[A_{12}^{(1)}\ B_1^{(1)}] \qquad \text{(a-1)}$$

$$(A_{22}^{(1)}-A_{21}^{(1)}Z^{(1)})X_2^{(1)}=B_2^{(1)}-A_{21}^{(1)}Y^{(1)} \qquad \text{(b-1)}$$

The equation (a-1) is a multiple RHS problem which we denote in a simple notation by

$$\text{L:} \quad A^{(2)}\ X^{(2)}=B^{(2)} \qquad (2.4.2a)$$

where $A^{(2)}=A_{11}^{(1)}$, $X^{(2)}=[Z^{(1)},Y^{(1)}]$ and $B^{(2)}=[A_{12}^{(1)},B_1^{(1)}]$. L means the system is on the left branch from the node $A^{(1)}$. We use a similar notation to denote equation (b-1) by

$$\text{R:} \quad A^{(2)}\ X^{(2)}=B^{(2)} \qquad (2.4.2b)$$

where $A^{(2)}=A_{22}^{(1)}-A_{21}^{(1)}Z^{(1)}$, $X^{(2)}=X_2^{(1)}$ and $B^{(2)}=B_2^{(1)}-A_{21}^{(1)}Y^{(1)}$. R means the system is on the right branch from the node $A^{(1)}$.

Equations (2.4.2a) and (2.4.2b) form a new matrix pattern like this:

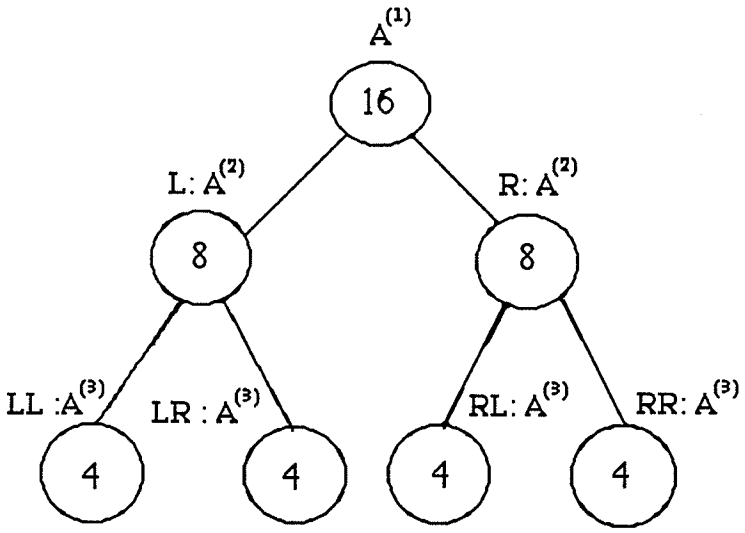
$$[A,B]=\begin{bmatrix}[L:A^{(2)}:8*8] & [L: \quad B^{(2)}: \quad 8*9 \quad] \\ [A_{21}^{(1)}: \quad 8*8] & [R:A^{(2)}:8*8] \quad [R:B^{(2)}:8*1]\end{bmatrix}$$

[Fig. 2.4.2c]

Our purpose is to partition $A^{(1)}$ into sub matrices of sizes 2-by-2, therefore doing a further symmetric partition gives

$$\begin{bmatrix}\begin{bmatrix}[A_{11}^{(2)}:4*4] & [A_{12}^{(2)}:4*4]\\ [A_{21}^{(2)}:4*4] & [A_{22}^{(2)}:4*4]\end{bmatrix} & \begin{bmatrix} B_1^{(2)}: & 4*9\\ B_2^{(2)}: & 4*9\end{bmatrix}\\ A_{21}^{(1)}:8*8 & \begin{bmatrix}\begin{bmatrix}[A_{11}^{(2)}:4*4] & [A_{12}^{(2)}:4*4]\\ [A_{21}^{(2)}:4*4] & [A_{22}^{(2)}:4*4]\end{bmatrix} & \begin{bmatrix}B_1^{(2)}:4*1\\ B_2^{(2)}:4*1\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

[Fig. 2.4.3a]



[Fig 2.4.3b]

Corresponding to the recursive formula (RF-k)~(2.1.22) for k=2, equations (2.4.2a) and (2.4.2b) respectively define two multiple RHS linear sub systems. We denote those systems generated from equation (2.4.2a) by

$$\text{LL:} \qquad A^{(3)} X^{(3)} = B^{(3)} \tag{2.4.3a}$$

where $A^{(3)} = A_{11}^{(2)}$, $X^{(3)} = [Z^{(2)}, Y^{(2)}]$, $B^{(3)} = [A_{12}^{(2)}, B_1^{(2)}]$

and

$$\text{LR:} \qquad A^{(3)} X^{(3)} = B^{(3)} \tag{2.4.3b}$$

where $A^{(3)} = A_{22}^{(2)} - A_{21}^{(2)} Z^{(2)}$, $X^{(3)} = X_2^{(2)}$, $B^{(3)} = B_2^{(2)} - A_{21}^{(2)} Y^{(2)}$

We also denote those systems generated from equation (2.4.2b) by

$$\text{RL:} \qquad A^{(3)} X^{(3)} = B^{(3)} \tag{2.4.3c}$$

where $A^{(3)} = A_{11}^{(2)}$, $X^{(3)} = [Z^{(2)}, Y^{(2)}]$, $B^{(3)} = [A_{12}^{(2)}, B_1^{(2)}]$

and

$$\text{RR:} \qquad A^{(3)} X^{(3)} = B^{(3)} \tag{2.4.3d}$$

where $A^{(3)} = A_{22}^{(2)} - A_{21}^{(2)} Z^{(2)}$, $X^{(3)} = X_2^{(2)}$, $B^{(3)} = B_2^{(2)} - A_{21}^{(2)} Y^{(2)}$

Now a new matrix pattern is formed by equations (2.4.3a) to (2.4.3d). The designations LL, LR, RL, and RR indicate the path from node $A^{(1)}$ to $A^{(3)}$ as shown in Fig 2.4.3b.

$$[A,B] = \begin{bmatrix} [LL{:}A^{(3)}{:}4*4] & [LL: & B^{(3)}{:}4*13 & &] \\ (A_{21}^{(2)}:4*4) & [LR{:}A^{(3)}{:}4*4] & [LR: & B^{(3)}:\ 4*9 &] \\ & & [RL{:}A^{(3)}{:}4*4] & [RL:B^{(3)}:\ 4*5 &] \\ (A_{21}^{(1)}:8*8) & & (A_{21}^{(2)}:4*4) & [RR{:}A^{(3)}{:}4*4] & [B^{(3)}{:}4*1] \end{bmatrix}_{16*17}$$

[Fig 2.4.3c]

A further symmetric partition can be done to sub matrices $A^{(3)}$, this produces eight 2-by-2 sub matrices. With the definitions that

$$A^{(4)} = A_{11}^{(3)},\ X^{(4)} = [Z^{(3)}, Y^{(3)}],\ B^{(4)} = [A_{12}^{(3)}, B_{1}^{(3)}] \qquad (2.4.4a)$$

for the systems on the left branches of the binary tree from nodes $A^{(3)}$

and that

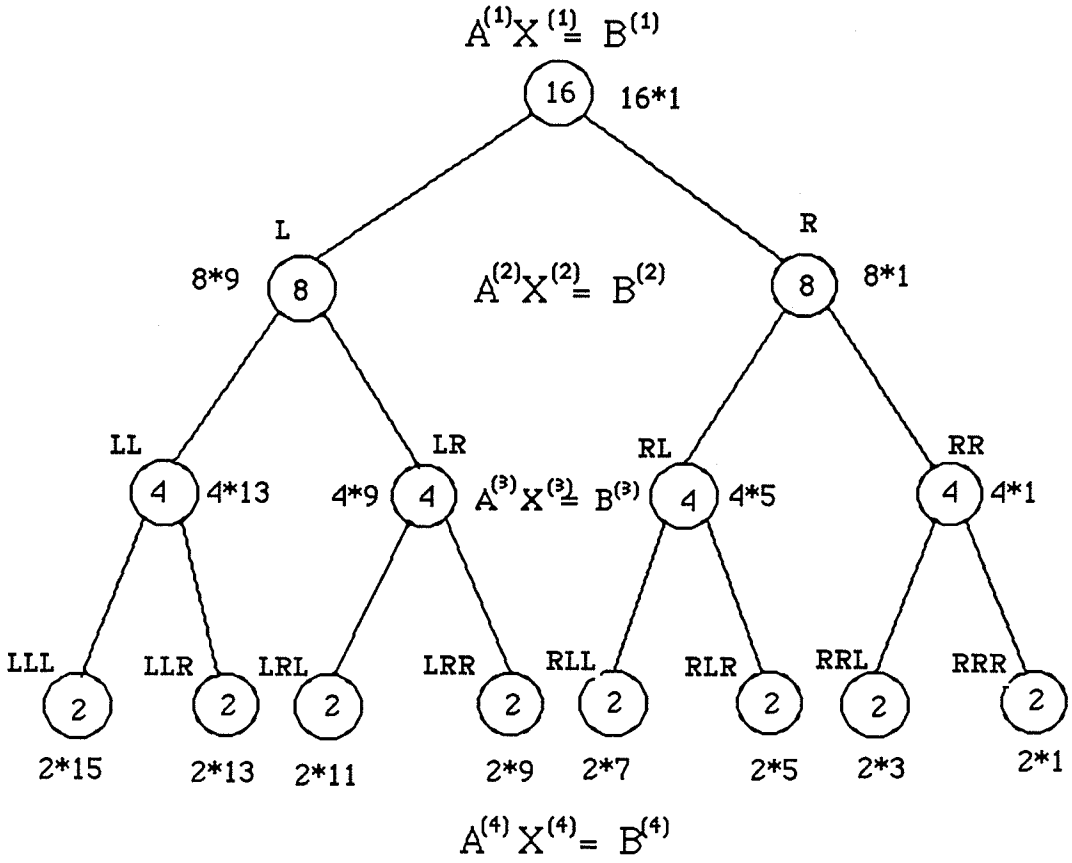
$$A^{(4)} = A_{22}^{(3)} - A_{21}^{(3)} Z^{(3)},\ X^{(4)} = X_{2}^{(3)},\ B^{(4)} = B_{2}^{(3)} - A_{21}^{(3)} Y^{(3)} \qquad (2.4.4b)$$

for the systems on the right branches of the binary tree from nodes $A^{(3)}$, we obtain eight 2-by-2 multiple RHS linear systems with the same form

$$[][][] \qquad A^{(4)}\ X^{(4)} = B^{(4)} \qquad (2.4.5)$$

where [][][] is LLL, LLR, LRL, LRR, RLL, RLR, RRL or RRR and indicates the path as shown in Fig 2.4.4.

The process of forming equation (2.4.4) is based on the recursive formula (RF-k)~(2.1.22) with k=3. Fig 2.4.4 shows a whole picture of the SSP applied to a 16-by-16 matrix.



The Strictly Symmetric Partition to a Matrix with Size 16-by-16
L: Left Branch relatively to the upper level top
R: Right Branch relatively to the upper level top
Numbers beside the nodes represent the sizes of the multiple RHS linear sub systems.

[Fig 2.4.4]

So far we have partitioned the matrix A into eight 2-by-2 sub matrices which produce eight 2-by-2 multiple RHS linear systems. The next step is to solve the sub systems, this will involves three types of computations defined in Section 2, that is, the Unit Solution (US), the TRANSFER and the BACKWARD. Referring to the binary tree the computation goes from the bottom to the top and from the left to the right.

We start computation with the first 2-by-2 sub system, that is

$$\text{LLL:} \qquad A^{(4)} X^{(4)} = B^{(4)} \qquad (2.4.5a)$$

where $A^{(4)}$ is 2-by-2 and $B^{(4)}$ is 2-by-15. They are defined by (2.4.4a).

If $A^{(4)}$ is non singular $X^{(4)}$ can be solved by applying the operation US to equation (2.4.4a), then operation TRANSFER will 'send' it to the right branch to form a new 2-by-2 sub system which we denoted by

$$\text{LLR:} \qquad A^{(4)}\ X^{(4)} = B^{(4)} \qquad (2.4.5b)$$

where $A^{(4)}$ is 2-by-2 and $B^{(4)}$ is 2-by-13, defined by (2.4.4b).

Again, if $A^{(4)}$ is non singular, $X^{(4)} = X_2^{(3)}$ can be simply solved by applying the US to (2.4.5b), after that the operation BACKWARD will compute $X_1^{(3)}$ and then $X^{(3)}=(X_1^{(3)}, X_2^{(3)})^T$ is found. According to the definition of $X^{(3)}$ in equation (2.4.3a), $X^{(3)} =[Z^{(2)},\ Y^{(2)}]$ which gives the solution of LL. Then we substitute it into equation (b-2) of the recursive formula (RF-2) and obtain a new 4-by-4 linear sub system which defines the equation (2.4.3b), that is

$$\text{LR:} \qquad A^{(3)}\ X^{(3)} = B^{(3)}$$

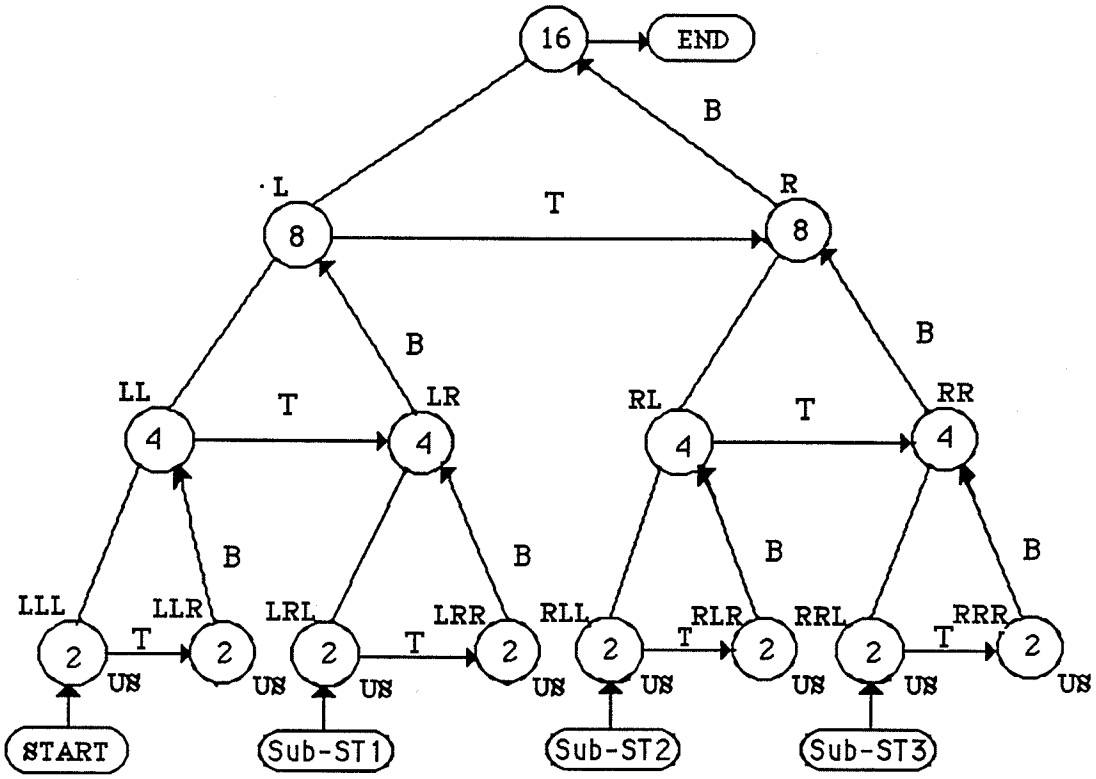
It is partitioned into two 2-by-2 sub system denoted by

$$\text{LRL:} \qquad A^{(4)}\ X^{(4)} = B^{(4)} \qquad (2.4.5c)$$

and

$$\text{LRR:} \qquad A^{(4)}\ X^{(4)} = B^{(4)} \qquad (2.4.5d)$$

Solving the equations (2.4.5c) and (2.4.5d) is the same as solving equations (2.4.5a) and (2.4.5b), After this is done, carrying out two sequential operations BACKWARD will find $X^{(3)}$ for LR (2.4.3b) and $X^{(2)}=[Z^{(1)}, Y^{(1)}]$ for L (2.4.2a). Then we have solved the sub systems on the left branch from the node $A^{(1)}$. We do an operation TRANSFER with respect to the equation (b-1) in the recursive formula (RF-1) to form the 8-by-8 sub system defined by R (2.4.2b). The process of solving equation (2.4.2b) is the exactly same as that of solving (2.4.2a). Repeat what we did in above $X^{(2)}$ will be solved for (2.4.2b) and then we do an operation BACKWARD defined by the (d-1) in (RF-k)~(2.1.22) to compute $X^{(1)}$, the final solution of the original linear system. Fig2.4.5 shows the detail of this process.



US: the Unit Solution. T: Transfer. B: Backward. Sub-ST n: Sub-starting point n

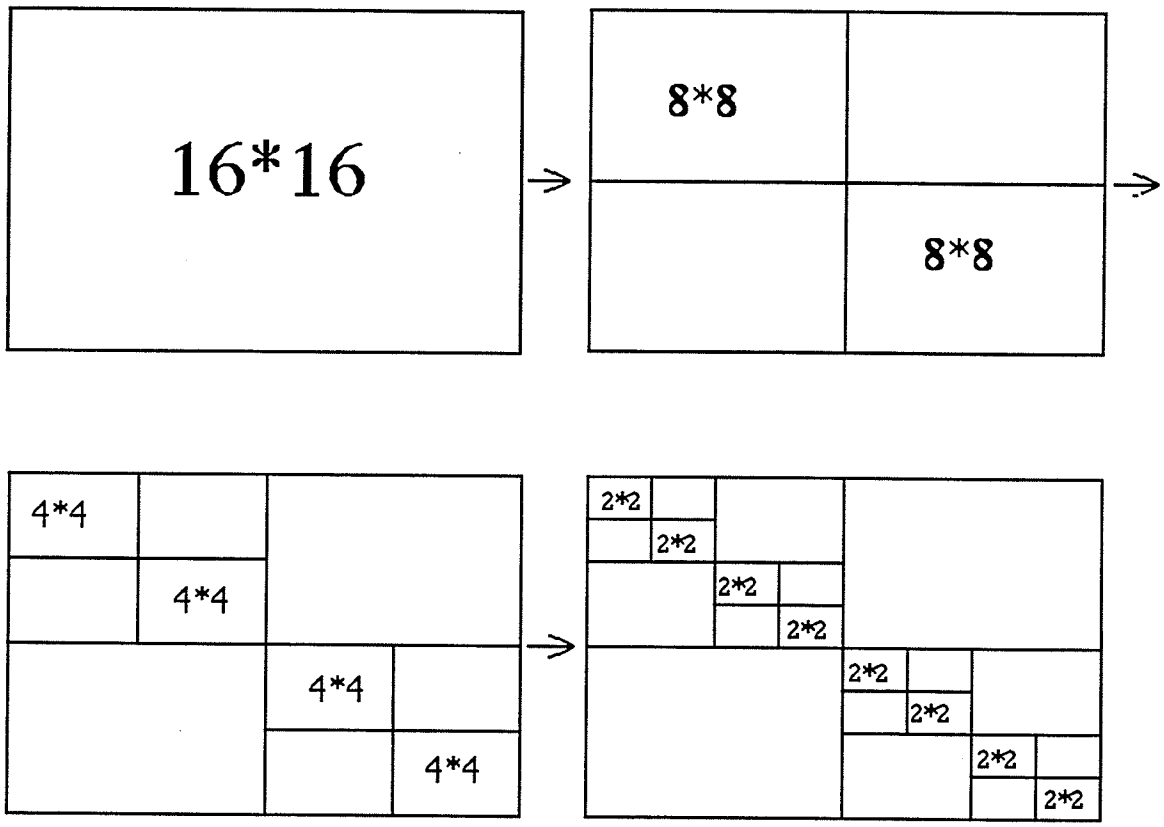
[Fig. 2.4.5]

The following matrix pattern shows the all diagonal 2-by-2 sub matrices involved in L~R coding system

$$A = \begin{bmatrix} [LLL] & & & & & & & \\ & [LLR] & & & & & & \\ & & [LRL] & & & & & \\ & & & [LRR] & & & & \\ & & & & [RLL] & & & \\ & & & & & [RLR] & & \\ & & & & & & [RRL] & \\ & & & & & & & [RRR] \end{bmatrix}_{16\times16}$$

From the above example we have some points to sum up

1) There is no computation involved in partitioning the matrix. The SSP partitions a matrix level by level until all sub matrices are 2-by-2 blocks. This is guaranteed by the assumption that the order of original matrix $n=2^i$. Therefore the SSP applied to a matrix is like a mesh generated level by level until it covers all of the 2-by-2 diagonal sub matrices. We show this process for the above example in Fig. 2.4.6

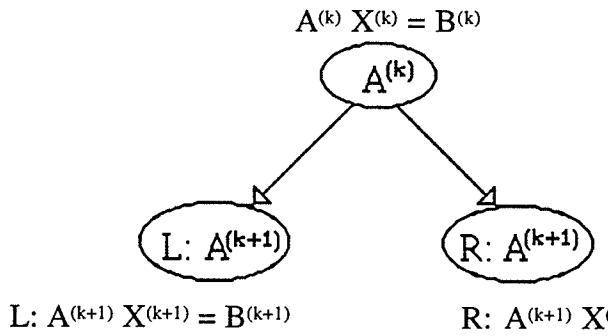


[Fig. 2.4.6]

Assume that at the k^{th} step of the partition we have the following matrices forms :

$$A^{(k)} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} \in R^{(2^k)\times(2^k)}, \quad B^{(k)} = \begin{bmatrix} B_1^{(k)} \\ B_2^{(k)} \end{bmatrix}, \quad X^{(k)} = \begin{bmatrix} X_1^{(k)} \\ X_2^{(k)} \end{bmatrix} \qquad (2.4.6a)$$

the recursive process of partitioning the sub-matrix $A^{(k)}$ into $A^{(k+1)}$ is shown below:



(2.4.6.b)

where, $A^{(1)}=A$ and the arrows represent the direction of matrix partition, and where,

$$L: A^{(k+1)} = A_{11}^{(k)} \qquad R: A^{(k+1)} = A_{22}^{(k)} - A_{21}^{(k)} Z^{(k)}$$

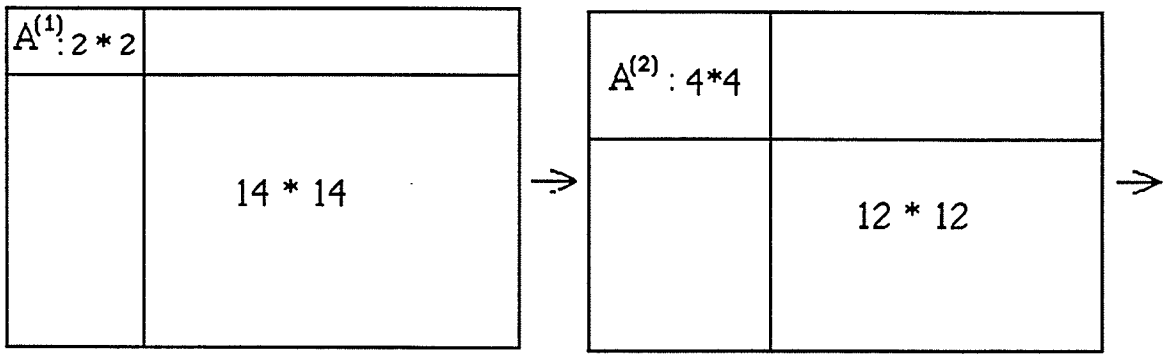
$$X^{(k+1)} = \left[Z^{(k)}, Y^{(k)}\right] \qquad X^{(k+1)} = X_2^{(k)}$$

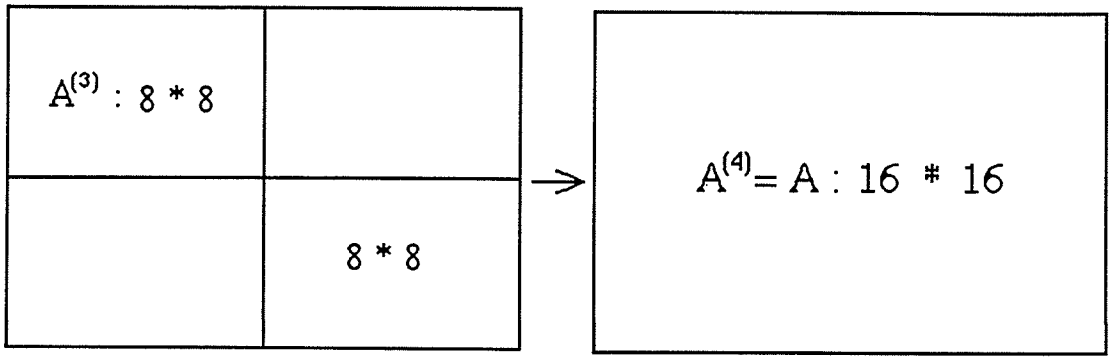
$$B^{(k+1)} = \left[A_{12}^{(k)}, B_1^{(k)}\right] \qquad B^{(k+1)} = B_2^{(k)} - A_{21}^{(k)} Y^{(k)}$$

The matrix patterns have the following forms

$$\left[A^{(k)}, B^{(k)}\right] = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} & B_1^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} & B_2^{(k)} \end{bmatrix} \rightarrow \begin{bmatrix} \left[L: A^{(k+1)}\right] & \left[\quad L: \quad B^{(k+1)}\right] \\ \left[\quad A_{21}^{(k)} \quad\right] & \left[R: A^{(k+1)}\right]\left[R: B^{(k+1)}\right] \end{bmatrix}$$

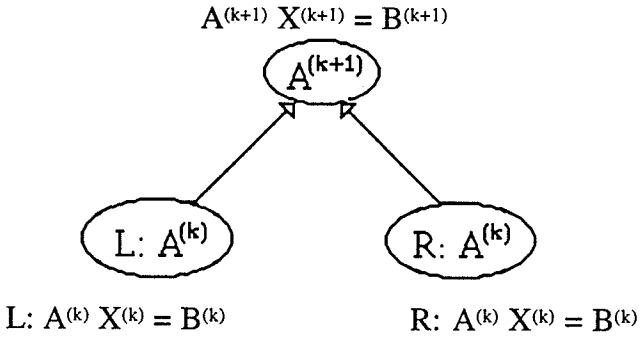
2) For a given matrix A of order $n=2^m$, the number of the 2-by-2 diagonal sub matrices is equal to 2^{m-1}. Therefore the process of partitioning matrix A into strictly symmetric blocks from size 2^m -by -2^m to size 2-by-2 can be also substituted by integrating the diagonal sub matrices from size 2-by-2 to size 2^m -by -2^m. This process goes contrary to the direction of the Fig 2.4.6. For better understanding we shown it in Fig 2.4.7.





[Fig. 2.4.7]

The recursive process of integrating sub-matrices $A^{(k)}$ into $A^{(k+1)}$ is shown below:



(2.4.6c)

where, $A^{(m)}=A$ and the arrows represent the direction of matrix integration, and where,

$$A_{11}^{(k+1)} = L : A^{(k)} \qquad A_{22}^{(k+1)} = R : A^{(k)} = A_{22}^{(k+1)} - A_{21}^{(k+1)} Z^{(k+1)}$$

$$\left[Z^{(k+1)}, Y^{(k+1)}\right] = L : X^{(k)} \qquad X_2^{(k+1)} = R : X^{(k)}$$

$$\left[A_{12}^{(k+1)}, B_1^{(k+1)}\right] = L : B^{(k)} \qquad B_2^{(k+1)} = R : B^{(k)} = B_2^{(k+1)} - A_{21}^{(k+1)} Y^{(k+1)}$$

The matrix patterns have the following forms,

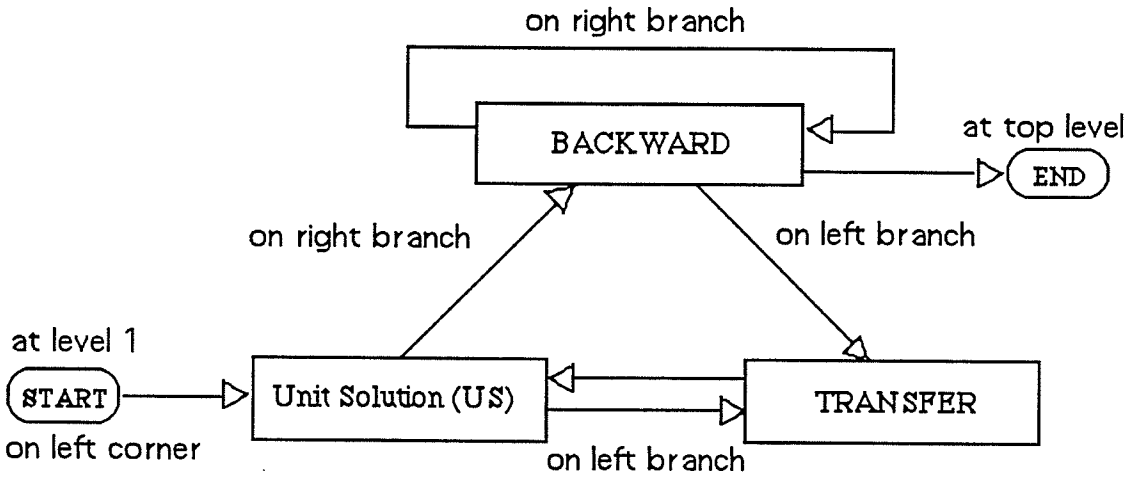
$$[A,B] = \left[\left[A^{(k)}\right]_{(2^k)\times(2^k)} \quad \left[B^{(k)}\right] \atop \ddots\right] \rightarrow$$

$$\left[\left[\begin{bmatrix} A_{11}^{(k+1)} = L : A^{(k)} \\ A_{21}^{(k+1)} \end{bmatrix} \quad \begin{bmatrix} A_{12}^{(k+1)} \\ A_{22}^{(k+1)} = R : A^{(k)} \end{bmatrix}\right]_{(2^{k+1})\times(2^{k+1})} \quad \begin{bmatrix} B_1^{(k+1)} \\ B_2^{(k+1)} = R : B^{(k)} \end{bmatrix} \atop \ddots\right]$$

We use the matrix integration scheme shown by (2.4.6c) in practical computation. There is no essential deference between the matrix partition and integration except to the orders of iterate number of the sub matrices. The matrix partition scheme defines the original matrix $A=A^{(1)}$ in the meantime the matrix integration scheme defines $A^{(1)}$ equal to the first 2-by-2 sub matrix on diagonal of A.

We have demonstrated the process of solving a linear system (2.4.1) by the SSP method in above, before we describe the algorithm for the SSP method we first analyze the relationship between the three basic operations US, TRANSFER and BACKWARD.

A computation process always starts from operation US, after that operations TRANSFER and BACKWARD could follow, and then these three operations are used repeatedly until the process is finally ended at the operation BACKWARD. The following figure shows this relationship.



Left and right branch and level refer to the binary tree

[Fig. 2.4.8]

Now we describe algorithms for the three operations by Matlab language.

1) US

According to the definition the operation US is the recursive formula (RF-k)~(2.1.22) applied to the solution of a 2-by-2 linear system

$$A^{(k)} X^{(k)} = B^{(k)}$$

where $A^{(k)} \in R^{2\times 2}$, $B^{(k)} \in R^{2\times m}$, for some integer m>0

The matrix pattern corresponding to this case has the form of

$$[A^{(k)},\ B^{(k)}] = \begin{bmatrix} \ddots & & \\ & \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix}_{2\times 2} & \begin{bmatrix} B_1^{(k)} \\ B_2^{(k)} \end{bmatrix}_{2\times m} \\ & & \ddots \end{bmatrix}_{n*n}$$

[Fig. 2.4.9]

Algorithm 2.4.1

function [A,B]=US

1° if $A_{11}^{(k)}=0$ then stop the computation, otherwise compute

$Z^{(k)}(1,1) = A_{12}^{(k)} / A_{11}^{(k)}$

$Y^{(k)}(1,1{:}m) = B^{(k)}(1,1:m)/ A_{11}^{(k)}$

2° if $A_{22}^{(k)} - A_{21}^{(k)}Z^{(k)}=0$ then stop the computation, otherwise compute

$X^{(k)}(2,1:m) = (B^{(k)}(2,1:m) - A_{21}^{(k)}Y^{(k)}(1,1:m))/(A_{22}^{(k)} - A_{21}^{(k)}Z^{(k)}(1,1))$

$X^{(k)}(1,1:m) = Y^{(k)}(1,1:m) - Z^{(k)}(1,1) * X^{(k)}(2,1:m)$

2) TRANSFER

In general, the operation TRANSFER deals with the matrix with form shown in Fig 2.4.9, and with $A_{11}^{(k)} \in R^{r*r}$, $A_{12}^{(k)} \in R^{r*s}$, $A_{21}^{(k)} \in R^{s*r}$, $A_{22}^{(k)} \in R^{s*s}$, $B_1^{(k)} \in R^{r*t}$ and $B_2^{(2)} \in R^{s*t}$, integers $r+s+t \le n$. For SSP method, r=s. The recursive formula for TRANSFER is $A^{(k+1)} = A_{22}^{(k)} - A_{21}^{(k)}Z^{(k)}$ and $B^{(k+1)} = B_2^{(k)} - A_{21}^{(k)}Y^{(k)}$
where $Z^{(k)} \in R^{r*s}$ and $Y^{(k)} \in R^{r*t}$.

Algorithm 2.4.2

function [A,B]=TRANSFER

$A^{(k+1)}(1:s,1:s) = A_{22}^{(k)}(1:s,1:s) - A_{21}^{(k)}(1:s,1:r) * Z^{(k)}(1:r,1:s)$

$B^{(k+1)}(1:s,1:t) = B^{(k)}(1:s,1:t) - A_{21}^{(k)}(1:s,1:r) * Y^{(k)}(1:r,1:t)$

We use the symbol **TRANSFER(r→s)** to represent the operation TRANSFER which is applied between matrices $A_{11}^{(k)}$ with order r and $A_{22}^{(k)}$ with order s. For SSP method it is always like TRANSFER (r→r).

3) BACKWARD

The recursive formula for BACKWARD is

$$X_1^{(k)} = Y^{(k)} - Z^{(k)} X_2^{(k)}$$

$$X_2^{(K+1)} = X^{(k)} = (X_1^{(k)}, X_2^{(k)})^T \text{ (for the matrix integration given by (2.4.6c))}$$

where, accord with the pattern of matrix shown in Fig 2.4.9, $X_1^{(k)} \in R^{r*t}$, $X_2^{(k)} \in R^{s*t}$, $Z^{(k)} \in R^{r*s}$, $Y^{(k)} \in R^{r*t}$.

Algorithm 2.4.3

function [A,B]=BACKWARD

$$X^{(k)}(1:r,1:t) = Y^{(k)}(1:r,1:t) - Z^{(k)}(1:r,1:s) * X^{(k)}(r+1:s,1:t)$$

We use the symbol **BACKWARD(k→k+1)** to represent the operation BACKWARD which is done from level k to level k+1 referring to the binary tree.

Based on the three basic operations we describe the SSP algorithm by the flow chart in Fig 2.4.10. The codes written both in Matlab and FORTRAN are attached in appendix.

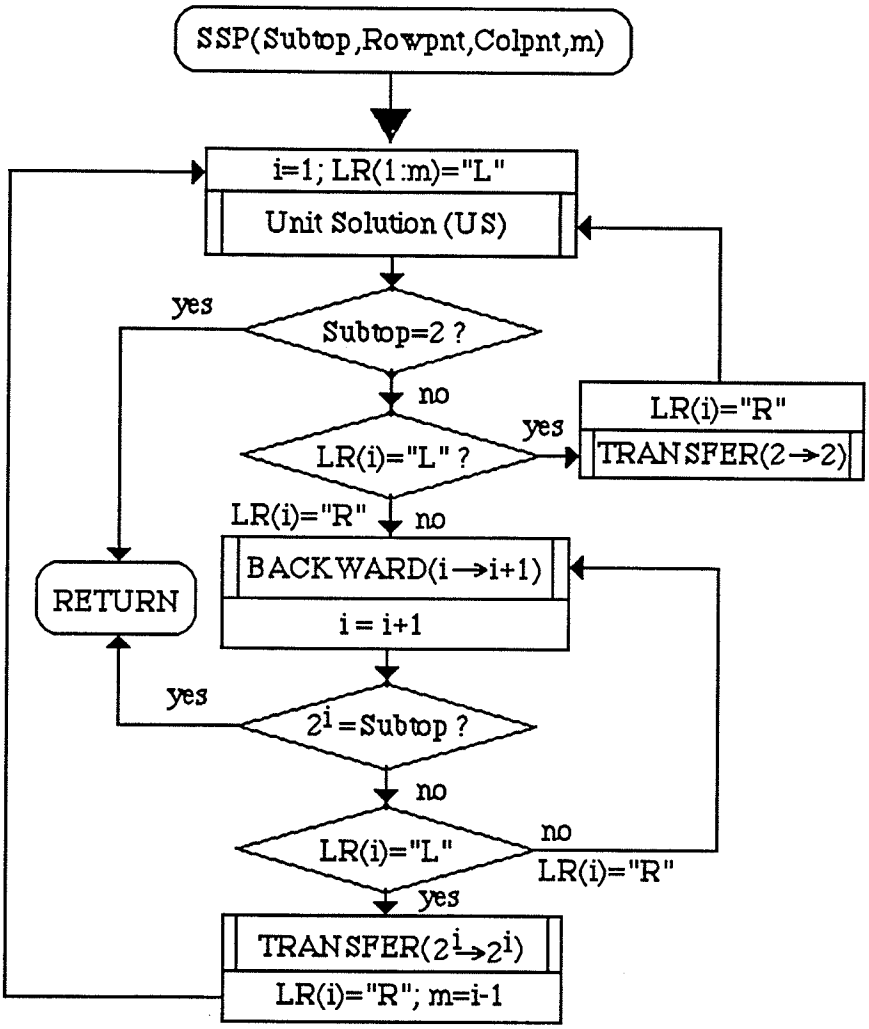
Consider problem

$$A^{(1)} X^{(1)} = B^{(1)}$$

where $A^{(1)} \in R^{n\times n}$, $B^{(1)} \in R^{n\times 1}$ and $n=2^m$

The Strictly Symmetric Partition (SSP) algorithm applied to the above problem is given below.

The Strictly Symmetric Partition (SSP) Algorithm



Subtop: Orders of the sub matrices, m: constant equal to 20,
LR(1:m): indicators of the left and right branches of the binary tree, "L":left, "R":right.
Rowpnt and Colpnt: the points of row and column of the first entry in the matrix $A^{(1)}$

[Fig. 2.4.10]

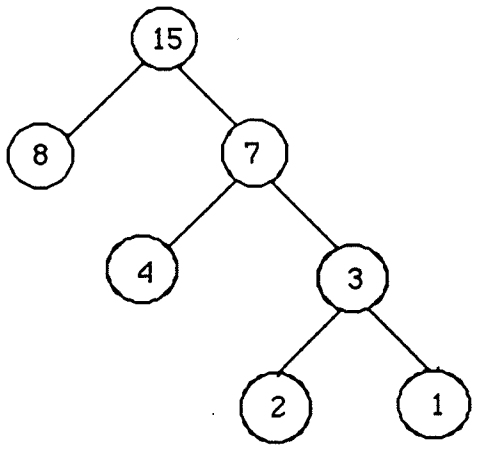
§ 5. The Left Strictly Symmetric Partition

So far we have demonstrated a process of solving a linear system (2.2.1) by applying a SSP method to it. The assumption for this algorithm is that the order of matrix A equals to 2^i, where i is a fixed positive integer and that A is non-singular. Obviously this is a very strict condition. Even in the case of a 3-by-3 linear system the SSP method will break down because a 3-by-3 matrix can not be partitioned into strictly symmetric blocks.

In order to make the SSP method applicable to any size linear system we have to extend the method for a general case. This is the purpose of this section. We firstly introduce a concept called the left strictly symmetric partition which is illustrated by the following example.

For a matrix with size 15-by-15, a general block partition applied to it can give the forms shown in Fig 2.5.2 and Fig 2.5.3. The recursive formula for each level partition is given by

$$A^{(k)} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix}, \quad A^{(k+1)} = \begin{cases} A_{11}^{(k)} & \text{for left branch} \\ A_{22}^{(k)} = A_{22}^{(k)} - A_{21}^{(k)} Z^{(k)} & \text{for right branch} \end{cases} \tag{2.5.1}$$



[Fig. 2.5.2]

$$A^{(1)} = \begin{bmatrix} [A_{11}^{(1)} : 8*8] & \\ A^{(2)} = & \begin{bmatrix} [A_{11}^{(2)} : 4*4] & \\ A^{(3)} = & \begin{bmatrix} [A_{11}^{(3)} : 2*2] & \\ A^{(4)} = & [A_{11}^{(4)} : 1*1] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

[Fig. 2.5.3]

Although the partition is not a strictly symmetric partition, the size of each node on the left branch can be always expressed as 2^i for some i. In other words the SSP is always available to the left branch at each level (not including the top) of the binary tree. Because of this characteristic of the partition we have the following definition.

Definition A general block partition applied to a matrix A is called the **left strictly symmetric partition** if the left branches at each level of the binary tree are partitioned into strictly symmetric blocks. We denote this partition by LSSP.

The LSSP is always applicable to a matrix with arbitrary size n. This is guaranteed by the fact that for any fixed integer $n \geq 2$, there exists decomposition

$$n = 2^{m_1} + 2^{m_2} + \cdots + 2^{m_t} \qquad (2.5.2)$$

where $m_1 > m_2 > \cdots m_{t-1} > m_t \geq 0$ and

$$m_t \begin{cases} =0 & \text{if n is an odd number, i.e. } n=2k+1 \\ \neq 0 & \text{if n is an even number, ie. } n=2k \end{cases} \qquad (2.5.3)$$

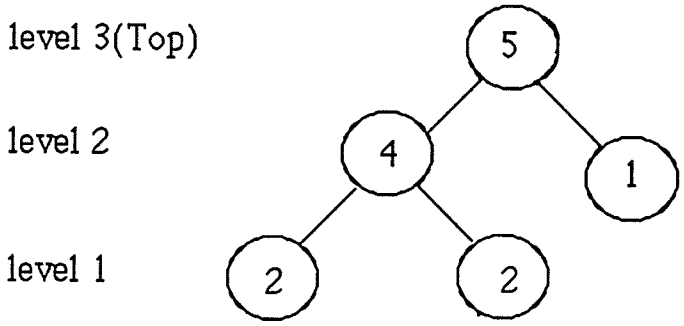
Set

$$S_j = 2^{m_j}, \quad Q_j = \sum_{i=1+j}^{t} 2^{m_i} = \sum_{i=1+j}^{t} S_j \qquad (j=1, \ldots, (t-1)) \qquad (2.5.4)$$

then for $j = 1, \cdots, t-1$ we have

$$Q_j = Q_{j-1} - S_j = n - \sum_{i=1}^{j} S_i$$
$$S_j \geq Q_j \qquad (2.5.5)$$

For j=1, $S_1 \geq Q_1$. This implies that the first block on the left branch is the biggest block in the all blocks. If we refer to the number of levels included in a binary as the height of the tree, then for LSSP the height equals to m_1+1. The following example shows that a 5-by-5 matrix which is partitioned into LSS blocks has the height equal to 3



[Fig. 2.5.4]

§ 6. The LSSP Method for Solving a Linear System

We have established the algorithm for the SSP method applied to the solution of a linear system, in this section we shall establish the algorithm for the LSSP method. The benefit of partitioning a matrix into LSS blocks is that the LSSP algorithm can be readily established by a minor modifications of the SSP algorithm or, in another word, by extending the SSP algorithm to a general case. Consider the problem (2.2.1) to which the LSSP algorithm is applied. Like the SSP algorithm the LSSP algorithm is also composed of two steps. The first step is to partition matrix A into LSS blocks. This actually involves the decomposition of order n in the form given by (2.5.2). It produces the binary tree and matrix pattern shown in Fig 2.5.4. The algorithm for this partition is actually to generate nodes 2^{m_j} (j=1, ... t) with corresponding Q_j defined by (2.5.4) and to store them in two arrays Ltop(1:m) and Rtop(1:m). It is described below.

Algorithm 2.6.1

Start with Top=n, k=1, Ltop(1:m)=Rtop(1:m)=0, c=0

[A,B]=LSSP1(n)

1° If n≤2 then set Ltop(k)=n and return
2° Set Top=int (Top/2), c=c+1
3° if Top>1 then goto 2°
4° 1) Set Ltop(k)=2^c
2) if k=1 then Rtop(k)=n-Ltop(k)
else Rtop(k)=Rtop(k-1)-Ltop(k)
3) Top=Rtop(k), c=0, k=k+1
5° if Top >1 then goto 2°
6° return

where, m is a constant which depends on the height of the binary tree shown in Fig 2.5.4a. In practical computation we set m=20 which corresponds to a matrix with order n=2^{20} (= 1048576). This number is big enough and could be regarded as the upper bound of the set of heights formed by the all heights of binary trees generated by the LSSP method. Ltop(1:m) and Rtop(1:m) are arrays respectively used to store the left block sizes S_j and right block sizes Q_j (j=1, ..., t-1).

The second step for LSSP algorithm is to solve each of the sub systems sequentially by the SSP method. At this stage the matrix pattern in Fig 2.5.4b becomes

$$[A,B]=\begin{bmatrix}[A^{(1)}] & [\ B^{(1)}:\ S_1\times(Q_1+1)\ ] & & \\ & [A^{(2)}] & [\ B^{(2)}:\ S_2\times(Q_2+1)\ ] & \\ & & \ddots & \\ & & [A^{(t-1)}] & [B^{(t-1)}:S_{t-1}\times(Q_{t-1}+1)] \\ & & & \begin{bmatrix} A^{(t)} \end{bmatrix}\ [B^{(t)}:S_t\times 1] \end{bmatrix}_{n\times(n+1)}$$

[Fig. 2.6.1]

Algorithm 2.6.2

function [A,B]=LSSP2([A,B])

1° for k=1, ..., t

a) Solve sub systems $A^{(k)}X^{(k)}=B^{(k)}$ for $X^{(k)}$by the SSP algorithm

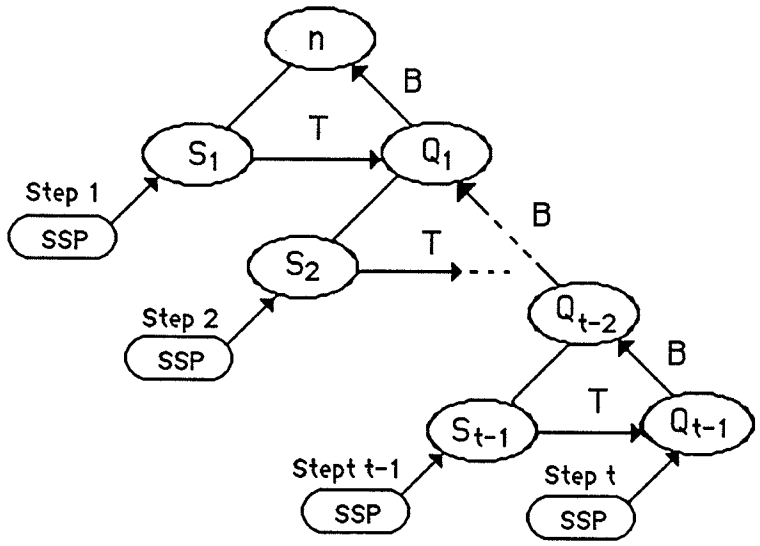
b) Do operation(s) TRANSFER from left branch with order S_k to right branch with order Q_k, that is, TRANSFER($S_k \rightarrow Q_k$)

2° for k= 1, ..., t

Do operation(s) BACKWARD from level K to level K-1, that is,
BACKWARD (K→K-1)
Return

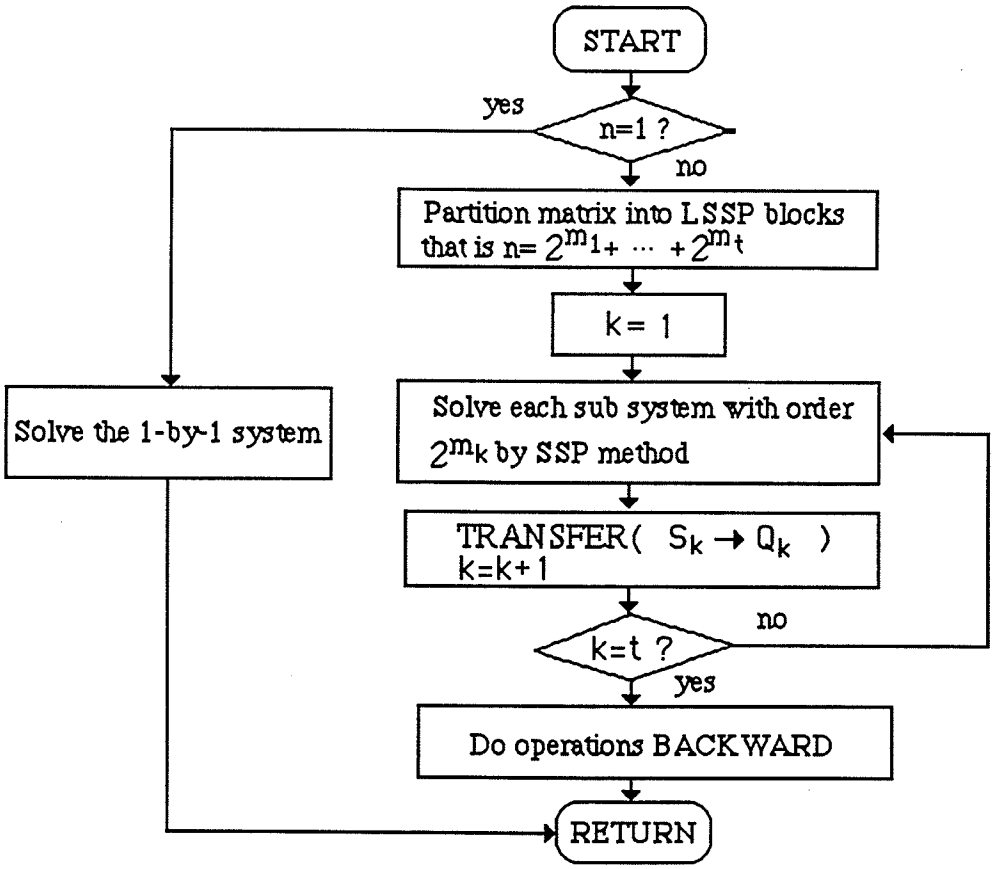
The Fig. 2.6.2 shows the process of solving a system by the above algorithm. It shows that the LSSP method is a special case of the general block partition methods shown in Fig. 2.2.1.

The Left Strictly Symmetric Partition (LSSP) Algorithm



S_i and Q_i are defined by (2.5.4)

[Fig. 2.6.2]



[Fig. 2.6.3]

We show the details of the process of solving a linear system by the LSSP method through the following example.

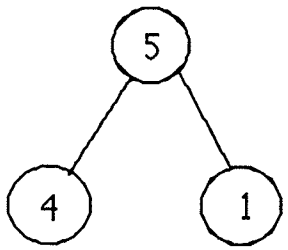
Example 6.1 Consider a 5-by-5 linear system

$$A^{(1)} X^{(1)} = B^{(1)} \qquad (2.6.1)$$

where,

$$A^{(1)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 3 & 5 & 4 \\ 3 & 2 & 1 & 6 & 4 \\ 4 & 3 & 4 & 1 & 5 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix} \quad \text{and} \quad B^{(1)} = \begin{bmatrix} 22 \\ 23 \\ 23 \\ 22 \\ 20 \end{bmatrix}$$

The exact solution to the problem is $X^{(1)}=(1,1,2,2,1)^T$. The LSSP decomposition for order 5 is 5= 4+1, which produces the LSSP binary tree and matrix pattern with form



[Fig. 2.6.4]

and

$$\left[A^{(1)}, \quad B^{(1)}\right] = \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & B_1^{(1)} \\ A_{21}^{(1)} & A_{22}^{(1)} & B_2^{(1)} \end{bmatrix}$$

where, $A_{11}^{(1)} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 5 \\ 3 & 2 & 1 & 6 \\ 4 & 3 & 4 & 1 \end{bmatrix}$, $A_{12}^{(1)} = \begin{bmatrix} 5 \\ 4 \\ 4 \\ 5 \end{bmatrix}$, $A_{21}^{(1)} = [5 \quad 4 \quad 3 \quad 2]$, $A_{22}^{(1)} = 1$,

$$B_1^{(1)} = [22,23,23,22]^T \quad , \quad B_2^{(1)} = 20$$

This partition corresponds to the recursive formula (RF-1) given by (2.1.4). The first step is to solve the multiple RHS linear sub system

$$A^{(2)}X^{(2)} = B^{(2)} \tag{2.6.2}$$

where $A^{(2)} = A_{11}^{(1)} \in R^{4*4}$, $X^{(2)} = [Z^{(1)}, Y^{(1)}] \in R^{4*2}$, $B^{(2)} = [A_{12}^{(1)}, B_1^{(1)}] \in R^{4*2}$ by the SSP method, which gives the solution

$$X^{(2)} = 1/34\begin{bmatrix} -8 & 26 \\ 37 & 71 \\ 20 & 88 \\ 11 & 79 \end{bmatrix}, \text{ i.e. } Z^{(1)} = 1/34\begin{bmatrix} -8 \\ 37 \\ 20 \\ 11 \end{bmatrix}, \quad Y^{(1)} = 1/34\begin{bmatrix} 26 \\ 71 \\ 88 \\ 79 \end{bmatrix}$$

The second step is to do an operation TRANSFER($4 \rightarrow 1$) to form a new sub system which we still denote by

$$A^{(2)}X^{(2)} = B^{(2)} \tag{2.6.3}$$

where $A^{(2)} = A_{22}^{(1)} - A_{21}^{(1)}Z^{(1)}$, $B^{(2)} = B_2^{(1)} - A_{21}^{(1)}Y^{(1)}$, and $X^{(2)} = X_2^{(1)}$

The computation of the TRANSFER($4 \rightarrow 1$) gives $A^{(2)}=B^{(2)}=-156/34$.

$$\because A^{(2)} \neq 0, \text{ then } X^{(2)} = X_2^{(1)} = B^{(2)}/A^{(2)} = 1$$

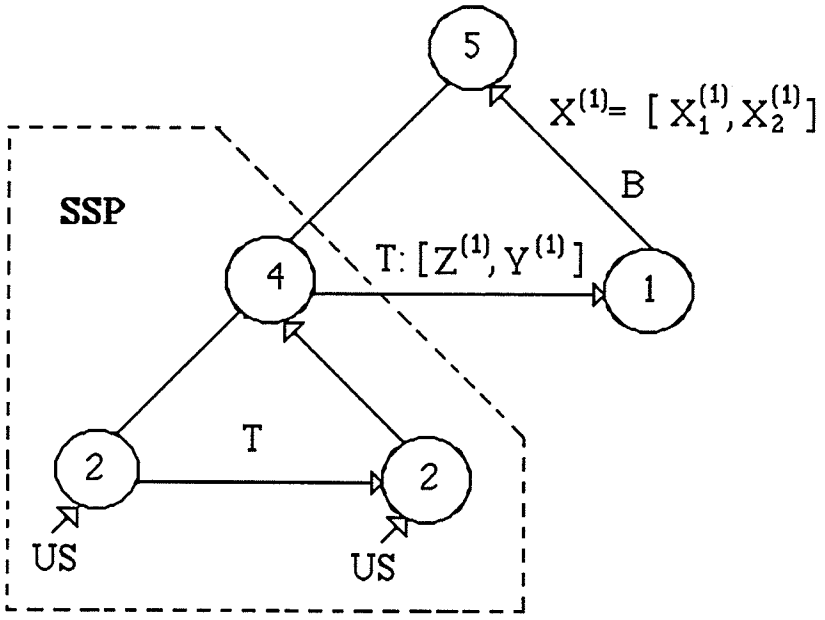
The third step is to do the operation BACKWARD which defined by equation (c-1) in the recursive formula (RF-1) The computation gives

$$X_1^{(1)} = Y^{(1)} - Z^{(1)}X_2^{(1)} = [1 \quad 1 \quad 2 \quad 2]^T$$

The fourth step is to form $X^{(1)}$ by equation (d-1) in (RF-1), that is

$$X^{(1)} = [X_1^{(1)}, \quad X_2^{(1)}]^T = [1 \quad 1 \quad 2 \quad 2 \quad 1]^T$$

It is the correct solution. The computation process can be expressed by the following binary tree.



T: TRANSFER B: BACKWARD

[Fig. 2.6.5]

Finally, we stress the point that the recursive formula (RF-k)~(2.1.22) is the kernel of the recursive block methods applied to the solution of a linear system. Based on it, different methods are developed corresponding to the different partition to matrix A. The SSP and LSSP methods are only two special cases of the general block partition method. The SSP method is recursively established on the operation US . The advantage that the SSP method offers to us is that it is not necessary to introduce an extra work space to store the nodes of the binary tree because of the symmetric structure of the matrix partition (or integration). The LSSP method is based on the SSP method, for this reason it requires relatively small extra memory for the storage of the LSSP nodes. In practical computation we use two arrays of length 20 to store the LSSP nodes.

§ 7. Controlling the Stability of the Algorithm

The operation US is the kernel part of the LSSP algorithm. It is a special case of the recursive formula (RF-k)~(2.1.22) applied to a 2-by-2 linear system with multiple RHS vectors. The formula for the US has the form

$$\begin{aligned} &A_{11}^{(k)}[Z^{(k)}, Y^{(k)}] = [A_{12}^{(k)}, B_1^{(k)}] && (a-k) \\ &(A_{22}^{(k)} - A_{21}^{(k)} Z^{(k)}) X_2^{(k)} = B_2^{(k)} - A_{21}^{(k)} Y^{(k)} && (b-k) \\ &X_1^{(k)} = Y^{(k)} - Z^{(k)} X_2^{(k)} && (c-k) \\ &X^{(k)} = (X_1^{(k)}, X_2^{(k)})^T && (d-k) \end{aligned} \qquad (2.7.1)$$

where $A_{ij}^{(k)}$ (i,j=1,2) are numbers.

We described the algorithm of the US in section 4, where we noted that the SSP algorithm breaks down when $A_{11}^{(k)}=0$ or $(A_{22}^{(k)} - A_{21}^{(k)} Z^{(k)})=0$. In order to avoid this case we have to use a strategy called partial pivoting. In this section we discuss this strategy and give a modified algorithm of the US.

The first step for the LSSP algorithm to solve the linear system (2.2.1) is to solve the first 2-by-2 multiple RHS problem which we denote by

$$A^{(1)} X^{(1)} = B^{(1)} \qquad (2.7.2)$$

by applying the operation US to it.

where $A^{(1)} = \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ A_{21}^{(1)} & A_{22}^{(1)} \end{bmatrix} \in R^{2\times 2}$, $X^{(1)} = \begin{bmatrix} X_1^{(1)} \\ X_2^{(1)} \end{bmatrix}$ and $B^{(1)} = \begin{bmatrix} B_1^{(1)} \\ B_2^{(1)} \end{bmatrix} \in R^{2\times(n-1)}$ and

$$[A,B] = \begin{bmatrix} \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ A_{21}^{(1)} & A_{22}^{(1)} \end{bmatrix} \begin{bmatrix} B_1^{(1)} \\ B_2^{(1)} \end{bmatrix} \\ \ddots \end{bmatrix}_{n\times(n+1)}$$

[Fig 2.7.1]

The formula of the US corresponding to this case has the form (2.7.1) with k=1. For convenience we denote $B_1^{(1)}, B_2^{(1)} \in R^{1\times(n-1)}$ by

$$B_i^{(1)} = \left[A_{i3}^{(1)}, A_{i4}^{(1)}, \cdots, A_{in+1}^{(1)},\right] \quad i=1,2$$

where $A_{ip}^{(1)} (i = 1,2, \quad p = 3,\cdots,n+1)$ are numbers.

The pivoting involved in the operation US for solving (2.7.2) is column pivoting for $A_{11}^{(1)}$ and $A_{22}^{(1)}$. The pivoting for $A_{11}^{(1)}$ is stated as the following three steps

1° Search for $s \in \{1,2,...,n\}$ such that $\left|A_{1s}^{(1)}\right| = \max_{1 \le P \le n} \left(\left|A_{1p}^{(1)}\right|\right)$

Note that $A_{in+1}^{(1)} (i = 1,2)$ are elements of the right hand side vector B, therefore pivoting should exclude them.

2° As soon as $A_{1s}^{(1)}$ is found, interchange the columns s and 1, i.e. $A_{1s}^{(1)} \leftrightarrow A_{11}^{(1)} (i = 1,\cdots,n)$. In this case an extra vector with size n should be introduced to store the changes of column indices in [A,B]. This is because column pivoting changes the order of the unknowns.

3° If $A_{11}^{(1)}$=0, the original matrix A is singular and then stop the computation process. Otherwise $[Z^{(1)}, Y^{(1)}]$ can be readily found through the formula (a-k) in (2.7.1) for k=1, that is, $[Z^{(1)}, Y^{(1)}] = [A_{12}^{(1)}, B_1^{(1)}] / A_{11}^{(1)}$. Then the equation (b-k) in (2.7.1) for k=1 defines a new linear system

$$A_{22}^{(1)} X_2^{(1)} = B_2^{(1)} \qquad (2.7.3)$$

where $A_{22}^{(1)} = A_{22}^{(1)} - A_{21}^{(1)} Z^{(1)}$ and $B_2^{(1)} = B_2^{(1)} - A_{21}^{(1)} Y^{(1)}$.

We will show in section 8 that the above computation (i.e. (a-1) and (b-1)) has the effect of eliminating the entry $A_{21}^{(1)}$. Therefore we can think of Fig 2.7.1 as

$$[A,B] = \begin{bmatrix} \begin{bmatrix} * & * \\ 0 & A_{22}^{(1)} \end{bmatrix} \begin{bmatrix} * \\ B_2^{(1)} \end{bmatrix} \\ (\textit{The same}) \quad \ddots \end{bmatrix}_{n \times (n+1)}$$

[Fig 2.7.2]

The pivoting for element $A_{22}^{(1)}$ is also accomplished by switching columns. Similarly to the above we describe it in three steps.

1° Search for $t \in \{2,...,n\}$ such that $\left|A_{2s}^{(1)}\right| = \max_{2\le P\le n}\left(\left|A_{2p}^{(1)}\right|\right)$

2° Interchange columns t and 2, i.e. $A_{it}^{(1)} \leftrightarrow A_{i2}^{(1)}$ $(i=1,\cdots,n)$ and store the changes of column indices in the ordering vector.

3° If $A_{22}^{(1)}=0$, the original matrix A is singular. Then stop the computation process. Otherwise solve the system (2.7.3) by computing $X_2^{(1)} = B_2^{(1)}/A_{22}^{(1)}$. And $X_1^{(1)}$ can be computed by the formula (C-k) in (2.7.1) for k=1. Then (2.7.2) is solved for $X^{(1)}$.

The second step for the LSSP algorithm to solve the linear system (2.2.1) is to do an operation TRANSFER (2→2) to form a new 2-by-2 linear system which we denote by

$$\tilde{A}^{(1)}\,\tilde{X}^{(1)} = \tilde{B}^{(1)} \qquad (2.7.4)$$

where $\tilde{A}^{(1)} \in R^{2\times 2}$ is the second 2-by-2 diagonal sub matrix in [A,B] and $\tilde{B}^{(1)} \in R^{2\times(n-3)}$.

$$[A,B] = \begin{bmatrix} \left[A^{(1)}\right] & \left[\ B^{(1)}\ \right] \\ \left[A_{21}^{(*)}\right] & \left[\tilde{A}^{(1)}\right]\left[\tilde{B}^{(1)}\right] \\ (\textit{The same}) & \ddots \end{bmatrix}_{n\times(n+1)}$$

[Fig. 2.7.3]

We will show in the section 8 that the above mentioned operation TRANSFER (2→2) has the effect of eliminating the block $A_{21}^{(*)}$, therefore we can think of Fig 2.7.3 as

$$[A,B] = \begin{bmatrix} \begin{bmatrix} * & * \\ 0 & * \end{bmatrix} & \begin{bmatrix} * \\ * \end{bmatrix} \\ \left[\ 0\ \right]_{2\times 2} & \left[\tilde{A}^{(1)}\right]\left[\tilde{B}^{(1)}\right] \\ (\textit{The same}) & \ddots \end{bmatrix}_{n\times(n+1)}$$

[Fig. 2.7.4]

Similarly to system (2.7.3), the sub system (2.7.4) is solved with column pivoting. In general, assume that the computation has been carried out up to k^{th} step and that a sub system

$$A^{(k)} X^{(k)} = B^{(k)} \qquad (2.7.5)$$

is obtained with the matrix pattern (which will be shown in section 8) below

$$[A,B] = \begin{bmatrix} \ddots & & & \\ 0 & * & \cdots & * \\ \vdots & \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} & \begin{bmatrix} B_1^{(k)} \\ B_2^{(k)} \end{bmatrix} & \\ 0 & (\textit{The same}) & & \ddots \end{bmatrix}_{n\times(n+1)}$$

[Fig. 2.7.5]

Where, $A_{ij}^{(k)} \in R$ and $B_i^{(k)} \in R^{1\times m}$ $(i,j=1,2)$ for some integer m>0.

For convenience we denote $B_i^{(k)}$ by

$$B_i^{(k)} = \left[A_{i3}^{(k)}, \cdots, A_{i\,m+2}^{(k)}\right] \qquad i=1,2$$

where $A_{ip}^{(k)}$ $(i=1,2, \quad p=3,\cdots,m+2)$ are numbers.

The column pivoting for $A_{11}^{(k)}$ is stated as the following three steps:

1° Search for $s \in \{1,2,...,m+1\}$ such that $\left|A_{1s}^{(k)}\right| = \max_{1\le p\le m+1}\left(\left|A_{1p}^{(k)}\right|\right)$

2° Interchange the columns of $A_{1s}^{(k)}$ and $A_{11}^{(k)}$, i.e. $A_{is}^{(k)} \leftrightarrow A_{i1}^{(k)}$ $(i=1,...,n)$ and record the changes of column indices.

3° If $A_{11}^{(k)}$=0, then the original matrix A is singular and then stop the computation process. Otherwise find $[Z^{(k)}, Y^{(k)}]$ by (a-k) in (2.7.1), that is, $[Z^{(k)}, Y^{(k)}] = [A_{12}^{(k)}, B_1^{(k)}]/A_{11}^{(k)}$. Then the equation (b-k) in (2.7.1) defines a new linear system

$$A_{22}^{(k)} X_2^{(k)} = B_2^{(k)} \qquad (2.7.6)$$

where $A_{22}^{(k)} = A_{22}^{(k)} - A_{21}^{(k)} Z^{(k)}$ and $B_2^{(k)} = B_2^{(k)} - A_{21}^{(k)} Y^{(k)}$.

We will show in section 8 that the above computation (i.e. (a-k) and (b-k)) has the effect of eliminating the element $A_{21}^{(k)}$. Then Fig 2.7.5 becomes

$$[A,B] = \begin{bmatrix} \ddots & * & \cdots & * \\ 0 & & & \\ \vdots & \begin{bmatrix} * & * \end{bmatrix} & & \begin{bmatrix} * \end{bmatrix} \\ 0 & \begin{bmatrix} 0 & A_{22}^{(k)} \end{bmatrix} & & \begin{bmatrix} B_2^{(k)} \end{bmatrix} \\ & (\textit{The same}) & & \ddots \end{bmatrix}_{n\times(n+1)}$$

[Fig. 2.7.6]

The pivoting for $A_{22}^{(k)}$ can be similarly described as

1° Search for $t \in \{2,...,m+1\}$ such that $\left|A_{2t}^{(k)}\right| = \max_{2\le p\le m+1}\left(\left|A_{2p}^{(k)}\right|\right)$

2° Interchange the columns of $A_{2t}^{(k)}$ and $A_{22}^{(k)}$ i.e. $A_{it}^{(k)} \leftrightarrow A_{i2}^{(k)}$ $(i=1,...,n)$ and store the changes of column indices in the introduced vector with size n.

3° If $A_{22}^{(k)}=0$, the original matrix A is singular. Then stop the computation process. Otherwise solve the system (2.7.6) simply by computing $X_2^{(k)} = B_2^{(k)}/A_{22}^{(k)}$. And $X_1^{(k)}$ can be computed by the formula (C-k) in (2.7.1). Then (2.7.5) is solved for $X^{(k)}$.

The column pivoting applied to the LSSP algorithm is always successful if the original matrix is non-singular. This is because that the operation TRANSFER does not change the non-singular characteristic of the matrix. We will show this claim is true in next section by a theoretical analysis of the recursive block algorithm. A row pivoting is not applicable to the LSSP algorithm because the operation TRANSFER usually affects some parts of the original matrix. These parts do not contain all the rows of the matrix but contain all of the columns required for pivoting. For example, as shown in Fig 2.7.3 and Fig 2.7.4, only two rows of the matrix [A,B], that is the two rows of $\left[\tilde{A}^{(1)}\right]\left[\tilde{B}^{(1)}\right]$, are affected after the

operation TRANSFER(2→2) is done. $A_{21}^{(*)}$ is eliminated by the TRANSFER(2→2). Therefore the columns remained in the rows 3 and 4 of [A,B] contain the all effective columns required for pivoting . Therefore column pivoting is used.

Assume that the computation has been carried out up to the k^{th} step and an array Index(1:n) is defined and initialized (that is Index(i)=i for i=1, ... , n) at the beginning of the computation. Set A(1:n,1:n+1)=[A,B] and assume that the index of row (or column) of $A_{11}^{(k)}$ in [A,B] (Fig 2.7.5) is PT. The modified algorithm of the US with the column pivoting is described below:

Algorithm 2.7.1

function [A]=US(A)

1° Pivoting for $A_{11}^{(k)}$ (i.e. A(PT,PT)):
   a) Search for $s \in \{PT,...,n\}$ such that $|A(PT,PT+s)| = \max_{PT \le k \le n} |A(PT,k)|$
   b) Interchange columns PT and PT+s in both A and Index
   c) If $A_{11}^{(k)}=0$ (i.e. A(PT,PT)=0), then A is singular. Stop.

2° Compute
   $Z^{(k)} = A_{12}^{(k)}/A_{11}^{(k)} \rightarrow A_{12}^{(k)}$ and $Y^{(k)} = B_2^{(k)}/A_{11}^{(k)} \rightarrow B_2^{(k)}$, that is
   A(PT,PT+1:n+1)=A(PT,PT+1:n+1)/A(PT,PT)

3° Compute
   $A_{22}^{(k)} - A_{12}^{(k)}Z^{(k)} \rightarrow A_{22}^{(k)}$ and $B_2^{(k)} - A_{12}^{(k)}Y^{(k)} \rightarrow B_2^{(k)}$, that is,
   A(PT+1,PT+1:n+1)=A(PT+1,PT+1:n+1)-A(PT+1,PT)*A(PT,PT+1:n+1)

4° Pivoting for $A_{22}^{(k)}$ (i.e. for A(PT+1,PT+1)):
   a) Search for $t \in \{PT+1,...,n\}$ such that $|A(PT+1,PT+t)| = \max_{PT+1 \le k \le n} |A(PT,k)|$
   b) Interchange columns PT+1 and PT+t in both A and Index
   c) If $A_{22}^{(k)}=0$ (i.e. A(PT+1,PT+1)=0), then A is singular. Stop.

5° Compute:
   a) $X_2^{(k)} = B_2^{(k)} / A_{22}^{(k)} \rightarrow B_2^{(k)}$, that is
      A(PT+1,PT+2:n+1)=A(PT+1,PT+2:n+1)/A(PT+1,PT+1)
   b) $X_1^{(k)} = Y^{(k)} - Z^{(k)} X_2^{(k)} \rightarrow B_1^{(k)}$, that is
      A(PT,PT+2:n+1)=A(PT,PT+2:n+1)-A(PT,PT+1)*A(PT+1,PT+2:n+1)

Return

§ 8. Theoretical Analysis of the Recursive Block Method

In this section we shall analyze the recursive block method in theory. We try to understand better the process of solving a linear system by the SSP and LSSP methods. Again we start our discussion with the operation US, the basis of the recursive block method.

§ 8.1 The Unit Solution (US)

The problem we consider in this section is still given by (2.2.1). Assume that after k^{th} step computation a matrix pattern is obtained as shown in Fig.2.7.5. The formula of the US is given by equations (2.7.1). Because of the column pivoting we can be sure that if A is non-singular then $A_{11}^{(k)} \neq 0$. $Z^{(k)}$ and $Y^{(k)}$ can be readily found by solving equation (a-k) in (2.7.1), which gives

$$Z^{(k)} = (A_{11}^{(k)})^{-1} A_{12}^{(k)}, \quad Y^{(k)} = (A_{11}^{(k)})^{-1} B_1^{(k)}. \tag{2.8.1}$$

Substituting (2.8.1) into equation (b-k) and (c-k) in (2.7.1) gives

$$(A_{22}^{(k)} - A_{21}^{(k)}(A_{11}^{(k)})^{-1} A_{12}^{(k)}) X_2^{(k)} = B_2^{(k)} - A_{21}^{(k)}(A_{11}^{(k)})^{-1} B_1^{(k)} \tag{2.8.2}$$

and

$$X_1^{(k)} = (A_{11}^{(k)})^{-1} B_1^{(k)} - (A_{11}^{(k)})^{-1} A_{12}^{(k)} X_2^{(k)} \tag{2.8.3}$$

Set $L^{(k)} = -A_{21}^{(k)}(A_{11}^{(k)})^{-1}$ and construct a matrix of the form

$$N^{(k)} = \begin{bmatrix} 1 & \\ L^{(k)} & 1 \end{bmatrix}_{2*2} \tag{2.8.4}$$

then equations (2.8.2) and (2.8.3) can be expressed as

$$\begin{bmatrix} 1 & 0 \\ L^{(k)} & 1 \end{bmatrix} \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} \begin{bmatrix} X_1^{(k)} \\ X_2^{(k)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ L^{(k)} & 1 \end{bmatrix} \begin{bmatrix} B_1^{(k)} \\ B_2^{(k)} \end{bmatrix} \tag{2.8.5}$$

or simply as

$$N^{(k)}[A^{(k)}, \quad B^{(k)}] = \begin{bmatrix} 1 & \\ L^{(k)} & 1 \end{bmatrix} \left[\begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} \begin{bmatrix} B_1^{(k)} \\ B_2^{(k)} \end{bmatrix} \right]$$

$$= \begin{bmatrix} \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ 0 & \tilde{A}_{22}^{(k)} \end{bmatrix} & \begin{bmatrix} B_1^{(k)} \\ \tilde{B}_2^{(k)} \end{bmatrix} \end{bmatrix} \qquad (2.8.6)$$

where $\tilde{A}_{22}^{(k)} = A_{22}^{(k)} + L^{(k)} A_{12}^{(k)}$, $\tilde{B}_2^{(k)} = B_2^{(k)} + L^{(k)} B_1^{(k)}$

$N^{(k)}$ is a lower triangular matrix with $\det(N^{(k)})=1$. Set $\tilde{L}^{(k)} = (N^{(k)})^{-1}$. It is easy to prove that $\tilde{L}^{(k)}$ is also a lower triangular matrix. Then (2.8.6) can be written as

$$\left[A^{(k)}, \ B^{(k)}\right] = \tilde{L}^{(k)} \tilde{R}^{(k)} \qquad (2.8.7)$$

where $\tilde{R}^{(k)} = \begin{bmatrix} \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ 0 & \tilde{A}_{22}^{(k)} \end{bmatrix} & \begin{bmatrix} B_1^{(k)} \\ \tilde{B}_2^{(k)} \end{bmatrix} \end{bmatrix}$ is a generalized upper triangular. Therefore (2.8.7) is the LU factorization of $[A^{(k)},B^{(k)}]$. In other words, the operation US is actually the Gaussian elimination method applied to a 2-by-2 multiple right hand side linear system.

§ 8.2 The SSP Method

The SSP algorithm is recursively established using the operations US, TRANSFER and BACKWARD and based on the assumption that the size of a linear system is 2^m, for some integer $m \geq 1$. The following analysis is based on the computation sequence of the three operations shown in Fig. 2.4.8. As we did before we partition the augmented matrix [A,B] into SSP blocks, that gives

$$[A,B] = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} [2*2] & & \\ & \ddots & \\ & & [2*2] \end{bmatrix}_{2^{m-1}*2^{m-1}} & \begin{bmatrix} [2*2] & & \\ & \ddots & \\ & & [2*2] \end{bmatrix}_{2^{m-1}*2^{m-1}} \end{bmatrix} & \begin{bmatrix} B \end{bmatrix}_{n*1} \end{bmatrix}$$

At each step, matrix partition for any size of blocks is given by form

$$\left[A^{(k)} \quad B^{(k)}\right]=\begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} & B_1^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} & B_1^{(k)} \end{bmatrix}$$

where, $A^{(k)}$ has the size of 2^k-by-2^k and $A_{ij}^{(k)}$ (i,j = 1,2) have sizes $2^{(k-1)}$-by-$2^{(k-1)}$

This form corresponds to the recursive formula (RF-k)~(2.1.22). We denote the first 2-by-2 sub system on the diagonal of the augmented matrix [A,B] by

$$A^{(1)}\, X^{(1)}=B^{(1)} \tag{2.8.8}$$

where $A^{(1)} \in R^{2*2}$, $B^{(1)} \in R^{2*(n-1)}$. The matrix pattern for this case is

$$[A,B]=\begin{bmatrix} \left[\ A^{(1)}\ \right]_{2*2} & \left[\ B^{(1)}\ \right]_{2*(n-1)} \\ \left[\ *\ \right]_{(n-2)*2} & \left[\ *\ \right]_{(n-2)*(n-1)} \end{bmatrix}_{n*(n+1)}$$

$$\left[A^{(1)} \quad B^{(1)}\right]=\begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & B_1^{(1)} \\ A_{21}^{(1)} & A_{22}^{(1)} & B_2^{(1)} \end{bmatrix}$$

As we analyzed in above, solving (2.8.8) by the operation US is the same as Gaussian elimination. Similarly to (2.8.4) we set

$$L^{(1)}=-A_{21}^{(1)}\left(A_{11}^{(1)}\right)^{-1} \tag{2.8.8a}$$

and construct matrix

$$N^{(1)}=\begin{bmatrix} 1 & \\ L^{(1)} & 1 \end{bmatrix}_{2*2} \tag{2.8.8b}$$

then we have

$$N^{(1)}\left[A^{(1)},\ B^{(1)}\right]=\left[\begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ 0 & \tilde{A}_{22}^{(1)} \end{bmatrix} \begin{bmatrix} B_1^{(1)} \\ \tilde{B}_2^{(1)} \end{bmatrix}\right] \tag{2.8.8c}$$

Moreover we construct matrix

$$M_1^{(1)}=\begin{bmatrix} N^{(1)} & \\ 0 & I_{n-2} \end{bmatrix} \tag{2.8.8d}$$

Then (2.8.8c) can be written as

$$M_1^{(1)}[A \quad B] = \begin{bmatrix} \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ 0 & \tilde{A}_{22}^{(1)} \end{bmatrix} \begin{bmatrix} B_1^{(1)} \\ \tilde{B}_2^{(1)} \end{bmatrix} \\ \text{----------} \\ \text{the same} \end{bmatrix} \tag{2.8.8e}$$

If there is column oriented pivoting involved in solving (2.8.8), two permutation matrices $P_1^{(1)}$ and $P_2^{(1)}$ are produced such that

$$M_1^{(1)}[A P_1, \quad B] = \begin{bmatrix} \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ 0 & \tilde{A}_{22}^{(1)} \end{bmatrix} \begin{bmatrix} B_1^{(1)} \\ \tilde{B}_2^{(1)} \end{bmatrix} \\ \text{----------} \\ \text{the same} \end{bmatrix} \tag{2.8.8f}$$

where $P_1 = P_1^{(1)} P_2^{(1)}$

After the equation (2.8.8) is solved, the next step is to solve the second 2-by-2 sub system on the diagonal of the augmented matrix [A,B]. Towards this end we have to go through a procedure called "matrix integration" which is defined by the recursive formula (2.4.6c). That is, we just regard $A^{(1)}$ as the first element $A_{11}^{(2)}$ of a 4-by-4 matrix $A^{(2)}$ which is the first 4-by-4 diagonal block in the augmented matrix [A,B]. Then we obtain a 4-by-4 sub system denote by

$$A^{(2)} X^{(2)} = B^{(2)} \tag{2.8.9}$$

The matrix pattern of this case becomes

$$[A,B] = \begin{bmatrix} \begin{bmatrix} A^{(2)} \end{bmatrix}_{4*4} & \begin{bmatrix} B^{(2)} \end{bmatrix}_{4*(n-3)} \\ \begin{bmatrix} * \end{bmatrix}_{(n-4)*4} & \begin{bmatrix} * \end{bmatrix}_{(n-4)*(n-3)} \end{bmatrix}_{n*(n+1)} \tag{2.8.10}$$

where , $\left[A^{(2)}, B^{(2)}\right] = \begin{bmatrix} A_{11}^{(2)} & A_{12}^{(2)} & B_1^{(2)} \\ A_{21}^{(2)} & A_{22}^{(2)} & B_2^{(2)} \end{bmatrix}$

The recursive formula (RF-2)-(2.1.22) corresponding to this case is

$$A_{11}^{(2)}[Z^{(2)},Y^{(2)}]=[A_{12}^{(2)},B_1^{(2)}] \qquad (a-2)$$
$$(A_{22}^{(2)}-A_{21}^{(2)}Z^{(2)})X_2^{(2)}=B_2^{(2)}-A_{21}^{(2)}Y^{(2)} \qquad (b-2)$$
$$X_1^{(2)}=Y^{(2)}-Z^{(2)}X_2^{(2)} \qquad (c-2)$$
$$X^{(2)}=(X_1^{(2)},X_2^{(2)})^T \qquad (d-2)$$

where, according to the (2.8.8) and the matrix integration method (2.4.6c) we have $A_{11}^{(2)}=A^{(1)}$, and correspondingly $[A_{12}^{(2)},B_1^{(2)}]=B^{(1)}$, $[Z^{(2)},Y^{(2)}]=X^{(1)}$.

The operation TRANSFER$(2\rightarrow 2)$ forms a new equation which we denote by

$$\tilde{A}^{(1)}\tilde{X}^{(1)}=\tilde{B}^{(1)} \qquad (2.8.11)$$

where $\tilde{A}^{(1)}\in R^{2\times 2}$, $\tilde{B}^{(1)}\in R^{2\times(n-3)}$ and

$$\tilde{A}^{(1)}=A_{22}^{(2)}-A_{21}^{(2)}Z^{(2)} \qquad (2.8.12)$$
$$\tilde{B}^{(1)}=B_2^{(2)}-A_{21}^{(2)}Y^{(2)} \qquad (2.8.13)$$

The matrix pattern of this case is

$$[A^{(2)}\quad B^{(2)}]=\begin{bmatrix}A_{11}^{(2)} & A_{12}^{(2)} & B_1^{(2)}\\ A_{21}^{(2)} & A_{22}^{(2)} & B_2^{(2)}\end{bmatrix}=\begin{bmatrix}[A^{(1)}] & \begin{bmatrix} & B^{(1)}\end{bmatrix}\\ [A_{21}^{(2)}] & \begin{bmatrix}\tilde{A}^{(1)} & \tilde{B}^{(1)}\end{bmatrix}\end{bmatrix} \qquad (2.8.14)$$

The operation TRANSFER$(2\rightarrow 2)$ is actually to form the equation (b-2). In the above computation $[Z^{(2)},Y^{(2)}]=X^{(1)}$ is found by solving (2.8.8) by applying the US to it. Here in order to look into the theoretical process of doing the operation TRANSFER$(2\rightarrow 2)$, we might as well assume that $Z^{(2)}=[A_{11}^{(2)}]^{-1}A_{12}^{(2)}$ and $Y^{(2)}=[A_{11}^{(2)}]^{-1}B_1^{(2)}$. Inserting them into (2.8.12) and (2.8.13) gives

$$\tilde{A}^{(1)}=A_{22}^{(2)}-A_{21}^{(2)}[A_{11}^{(2)}]^{-1}A_{12}^{(2)}$$
$$\tilde{B}^{(1)}=B_2^{(2)}-A_{21}^{(2)}[A_{11}^{(2)}]^{-1}B_1^{(2)}$$

Set $L^{(2)}=-A_{21}^{(2)}[A_{11}^{(2)}]^{-1}\in R^{2*2}$ and construct the matrix

$$N^{(2)} = \begin{bmatrix} I_2 & \\ L^{(2)} & I_2 \end{bmatrix}_{4*4} \tag{2.8.15}$$

Then the operation TRANSFER($2 \rightarrow 2$) can be expressed in matrix form

$$N^{(2)}\left[A^{(2)} \quad B^{(2)}\right] = \begin{bmatrix} A_{11}^{(2)} & A_{12}^{(2)} & B_1^{(2)} \\ 0 & \tilde{A}_{22}^{(1)} & \tilde{B}_2^{(1)} \end{bmatrix} \tag{2.8.16}$$

Moreover if we construct

$$M_1^{(2)} = \begin{bmatrix} N^{(2)} & 0 \\ 0 & I_{n-4} \end{bmatrix} \tag{2.8.17}$$

then (2.8.16) can be expressed as

$$M_1^{(2)}[A, \quad B] = \begin{bmatrix} A_{11}^{(2)} & A_{12}^{(2)} & B_1^{(2)} \\ 0 & \tilde{A}^{(1)} & \tilde{B}^{(1)} \\ \text{----------} & & \\ \text{the same} & & \end{bmatrix} \tag{2.8.18}$$

Combining (2.8.8f) with (2.8.18) we obtain

$$M_1^{(2)} M_1^{(1)}[A P_1, B] = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ 0 & \tilde{A}_{22}^{(1)} \end{bmatrix} & \begin{bmatrix} B_1^{(1)} \\ \tilde{B}_2^{(1)} \end{bmatrix} \\ \begin{bmatrix} 0 \end{bmatrix} & \begin{bmatrix} \tilde{A}^{(1)} & \tilde{B}^{(1)} \end{bmatrix} \end{bmatrix} \\ \text{----------} \\ \text{The Same} \end{bmatrix} \tag{2.8.19}$$

Solving (2.8.11) is the exactly same as solving (2.8.8). Similarly we have

$$M_2^{(1)} = \begin{bmatrix} I_2 & & \\ & N^{(1)} & \\ & & I_{n-4} \end{bmatrix}_{n*n} \tag{2.8.20}$$

where $N^{(1)} = \begin{bmatrix} 1 & \\ L^{(1)} & 1 \end{bmatrix}, L^{(1)} = -\tilde{A}_{21}^{(1)}\left[\tilde{A}_{11}^{(1)}\right]^{-1} \in R$

and permutation matrix $P_2 = \tilde{P}_2^{(1)}\tilde{P}_2^{(2)}$, which is involved in searching for pivoting, such that

$$M_2^{(1)}(M_1^{(2)}M_1^{(1)}[AP_1P_2, \quad B]) = \begin{bmatrix} \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ 0 & \tilde{A}_{22}^{(1)} \end{bmatrix} & \begin{bmatrix} B_1^{(1)} \\ \tilde{B}_2^{(1)} \end{bmatrix} \\ 0 & \begin{bmatrix} \tilde{A}_{11}^{(1)} & \tilde{A}_{12}^{(1)} \\ 0 & \tilde{\tilde{A}}_{22}^{(1)} \end{bmatrix} \quad \begin{bmatrix} \tilde{B}_1^{(1)} \\ \tilde{\tilde{B}}_2^{(1)} \end{bmatrix} \\ \hline \text{the same} & \end{bmatrix}$$

At this stage solution $X^{(2)}$ to the equation (2.8.9) can be solved by doing the operation BACKWARD $(2 \longrightarrow 3)$(from level 2 to level 3) defined by (c-2) in recursive formula (RF-2). The next step for solving the third diagonal 2-by-2 sub system in [A,B] will involve in the matrix integration from sizes 4-by-4 to 8-by-8 and then solve each sub system. The details was discussed in section 4.

In general, we construct matrix $M_k^{(i)}$ in the following three steps

1° $$M_k^{(i)} = \begin{bmatrix} I_{2(k-1)} & & \\ & N^{(i)} & \\ & & I_s \end{bmatrix} \qquad (2.8.21a)$$

2° $$N^{(i)} = \begin{bmatrix} I_{2^{i-1}} & \\ L^{(i)} & I_{2^{i-1}} \end{bmatrix} \qquad (2.8.21b)$$

3° $$L^{(i)} = -A_{21}^{(i)}\left[A_{11}^{(i)}\right]^{-1} \in R^{(2^{i-1})\times(2^{i-1})} \qquad (2.8.21c)$$

where $s=n-(2(k-1)+2^i)$, $k= 1, \ldots, 2^{m-1}$, $i=1,\ldots, m-1$
and where k is the number of the 2-by-2 diagonal sub matrices in [A,B]

According to the computation sequence shown in Fig 2.4.8 we have

$$\prod_{\substack{1\le i\le m-1\\ 1\le k\le 2^m}} M_k^{(i)}\left[A \prod_{1\le k\le 2^m} P_k \quad , \quad B\right]=\begin{bmatrix} * & * & * \\ & * & * \\ 0 & & * \end{bmatrix}= U \tag{2.8.22}$$

Where U is a generalized upper triangular matrix and $\prod_{1\le k\le 2^m} P_k$ are permutation matrices involved in pivoting. For convenience we set

$$M_{ssp}^{(m)} = \prod_{\substack{1\le i\le m-1\\ 1\le k\le 2^m}} M_k^{(i)} \tag{2.8.23}$$

$$P_{ssp}^{(m)} = \prod_{1\le k\le 2^m} P_k \tag{2.8.24}$$

where, 2^m is the size of the original matrix.

Obviously, $M_{ssp}^{(m)}$ is lower triangular and $\det(M_{ssp}^{(m)})=1$. Set $L=\left[M_{ssp}^{(m)}\right]^{-1}$ then from (2.8.22) we have

$$[A P_{ssp}^{(m)}, B]= LU \tag{2.8.25}$$

This implies that the SSP method is essentially same as LU factorization, however it decomposes the matrix A in a recursive block process. We show this in example 2.8.1. Before that we just simply describe the computation sequence for the SSP method below.

For k= 1,..., 2^{m-1} and i=1,...m-1

1) Construct $M_k^{(i)}$ by (2.8.21)

2) Compute $[A,B]=M_k^{(i)}[A,B]$. If there is pivoting involved, assume that the permutation matrix applied is P_k, then Compute $[A,B]=[AP_k,B]$

3) Do operation(s) BACKWARD. If i=m then stop the computation, else goto 1)

Example 2.8.1

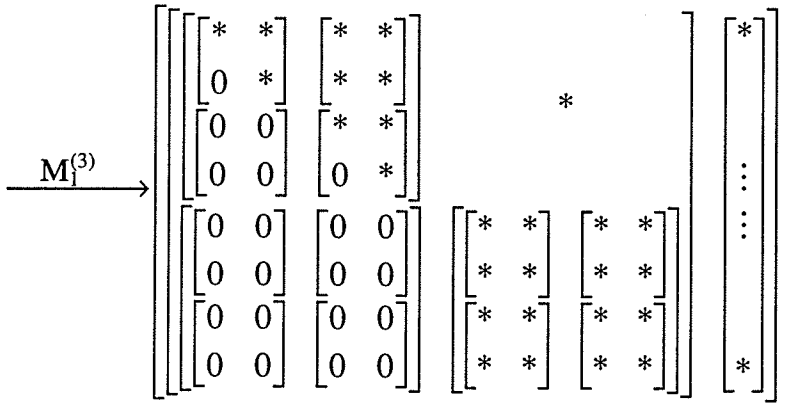
Consider an 8-by-8 linear system AX=B. We firstly partition [A,B] into the strictly symmetric blocks and then solve the system by the SSP method. The following matrix patterns show this process, where BKD represents operation BACKWARD.

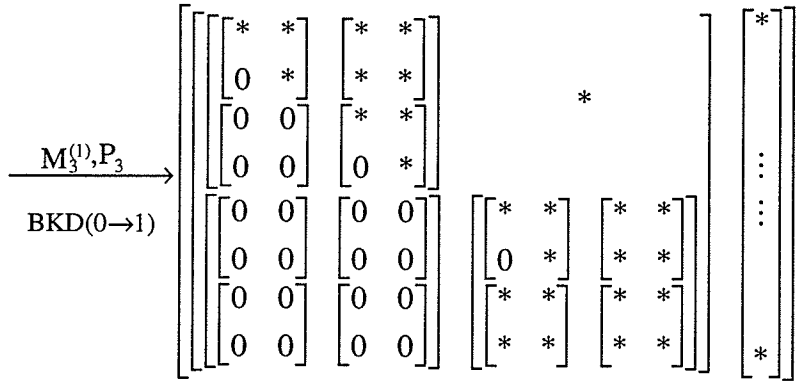
$$[A,B]=\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}[2\times2] & \\ & [2\times2]\end{bmatrix} & \\ & \begin{bmatrix}[2\times2] & \\ & [2\times2]\end{bmatrix}\end{bmatrix} & \begin{bmatrix}B\end{bmatrix}\end{bmatrix}$$

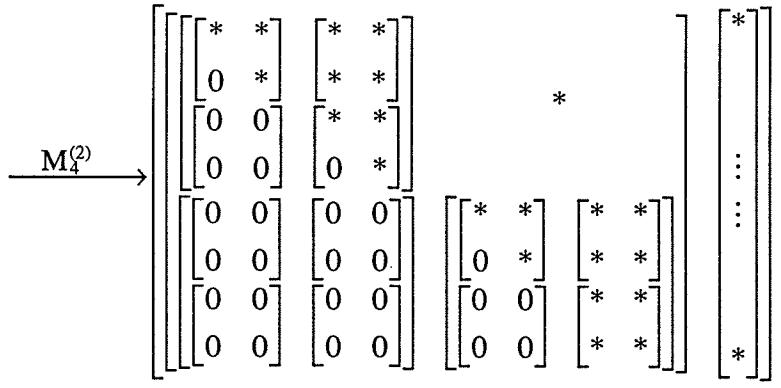
$$\xrightarrow[\text{BKD}(0\to1)]{M_1^{(1)},\ P_1}\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}* & *\\ 0 & *\end{bmatrix} & \begin{bmatrix}* & *\\ * & *\end{bmatrix}\\ * & [2\times2]\end{bmatrix} & \\ & \begin{bmatrix}[2\times2] & \\ & [2\times2]\end{bmatrix}\end{bmatrix} & \begin{bmatrix}*\\ \vdots\\ *\end{bmatrix}\end{bmatrix}$$

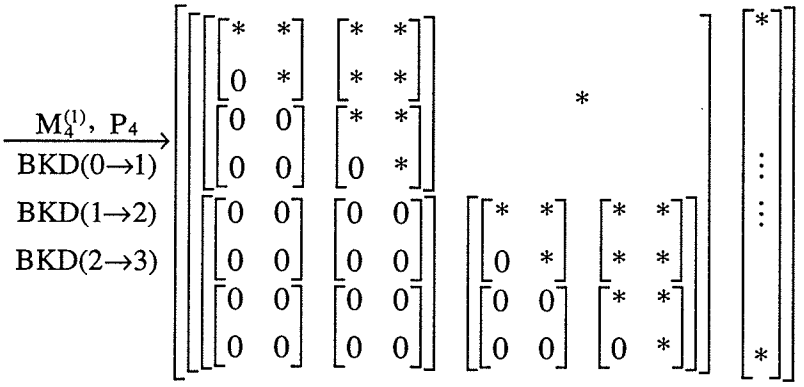
$$\xrightarrow{M_1^{(2)}}\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}* & *\\ 0 & *\end{bmatrix} & \begin{bmatrix}* & *\\ * & *\end{bmatrix}\\ \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} & \begin{bmatrix}* & *\\ * & *\end{bmatrix}\end{bmatrix} & \\ & \begin{bmatrix}[2\times2] & \\ & [2\times2]\end{bmatrix}\end{bmatrix} & \begin{bmatrix}*\\ \vdots\\ *\end{bmatrix}\end{bmatrix}$$

$$\xrightarrow[\text{BKD}(0\to1)\ \text{BKD}(1\to2)]{M_2^{(1)},\ P_2}\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}* & *\\ 0 & *\end{bmatrix} & \begin{bmatrix}* & *\\ * & *\end{bmatrix}\\ \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} & \begin{bmatrix}* & *\\ 0 & *\end{bmatrix}\end{bmatrix} & \\ & \begin{bmatrix}[2\times2] & \\ & [2\times2]\end{bmatrix}\end{bmatrix} & \begin{bmatrix}*\\ \vdots\\ *\end{bmatrix}\end{bmatrix}$$


$M_1^{(3)}$


$M_3^{(1)},P_3$
BKD(0→1)


$M_4^{(2)}$



The above computation sequence can be described as follows

1° $M_1^{(1)}[A P_1 \quad B]$
$BKD(0 \rightarrow 1)$

2° $M_1^{(2)} M_1^{(1)}[A P_1 \quad B]$

3° $M_2^{(1)} M_1^{(2)} M_1^{(1)}[A P_1 P_2 \quad B]$
$BKD(0 \rightarrow 1)$, $BKD(1 \rightarrow 2)$

4° $M_1^{(3)} M_2^{(1)} M_1^{(2)} M_1^{(1)}[A P_1 P_2 \quad B]$

5° $M_3^{(1)} M_1^{(3)} M_2^{(1)} M_1^{(2)} M_1^{(1)}[A P_1 P_2 P_3 \quad B]$
$BKD(0 \rightarrow 1)$

6° $M_4^{(2)} M_3^{(1)} M_1^{(3)} M_2^{(1)} M_1^{(2)} M_1^{(1)}[A P_1 P_2 P_3 \quad B]$

7° $M_4^{(1)} M_4^{(2)} M_3^{(1)} M_1^{(3)} M_2^{(1)} M_1^{(2)} M_1^{(1)}[A P_1 P_2 P_3 P_4 \quad B]$
$BKD(0 \rightarrow 1)$, $BKD(1 \rightarrow 2)$, $BKD(2 \rightarrow 3)$

§ 8.3 The LSSP Method

Theoretical analysis for the LSSP method is similar to that for the SSP method. Consider a matrix partitioned into LSSP blocks with decomposition given by (2.5.2), that is

$$n = 2^{m_1} + 2^{m_2} + \cdots + 2^{m_t}$$

the LSSP method solves each sub system with order 2^{m_j} (j=1, ..., t) by applying the SSP method to it, therefore the method for constructing matrices $M_{ssp}^{(m_j)}$ is the same as we did in last section. The only difference is how to deal with the operation TRANSFER done between two non-symmetric branches of the binary tree, this results in a different form of the $M^{(j)}$, which is defined by the following three steps with the notation of (2.5.4).

1° $$M^{(j)} = \begin{bmatrix} I_{P_j} & \\ & N^{(j)} \end{bmatrix} \qquad (2.8.26a)$$

where, $p_j = \sum_{i=1}^{j-1} 2^{m_i} = \sum_{i=1}^{j-1} S_i$, j=1, 2, ... t. We set p_1=0 and define the unit matrix I_0 as an empty matrix, that is, a matrix with no entry

2° $$N^{(j)} = \begin{bmatrix} I_{S_j} & \\ L^{(j)} & I_{Q_j} \end{bmatrix} \qquad (2.8.26b)$$

3° $$L^{(j)} = -A_{21}^{(j)}\left[A_{11}^{(j)}\right]^{-1} \in R^{Q_j \times S_j} \qquad (2.8.26c)$$

where, $A_{21}^{(j)} \in R^{Q_j \times S_j}$ and $\left[A_{11}^{(j)}\right]^{-1} \in R^{S_j \times S_j}$

For example, when j=1 we have p_1=0, then

$$M^{(1)} = N^{(1)} = \begin{bmatrix} I_{S_1} & \\ L^{(1)} & I_{Q_1} \end{bmatrix}$$

where $L^{(1)} = -A_{21}^{(1)}\left[A_{11}^{(1)}\right]^{-1} \in R^{Q_1 \times S_1}$

When j=t-1, we have $p_{t-1} = \sum_{i=1}^{t-2} S_i$ and then

$$M^{(t-1)} = \begin{bmatrix} I_{p_{t-1}} & \\ & N^{(t-1)} \end{bmatrix}$$

$$N^{(t-1)} = \begin{bmatrix} I_{S_{t-1}} & \\ L^{(t-1)} & I_{Q_{t-1}} \end{bmatrix}$$

$$L^{(t-1)} = -A_{21}^{(t-1)}\left[A_{11}^{(t-1)}\right]^{-1} \in R^{S_{t-1}\times Q_{t-1}}$$

where $A_{21}^{(t-1)} \in R^{S_{t-1}\times Q_{t-1}}$, $\left[A_{11}^{(t-1)}\right]^{-1} \in R^{S_{t-1}\times S_{t-1}}$

Now we have constructed matrix $M^{(j)}$. From the construction process given by (2.8.26) we know that $M^{(j)}$ is lower triangular and $\det(M^{(j)})=1$ for any j=1,..., t-1. Then the process of solving a linear system by the LSSP method can be expressed by

$$M_{ssp}^{(m_t)} M^{(t-1)} M_{ssp}^{(m_{t-1})} \ldots M^{(1)} M_{ssp}^{(m_1)} [AP, B] = U \qquad (2.8.27)$$

where U is upper triangular form and P is multiplication of permutation matrices which are involved in solving each sub system by the SSP method with pivoting.

Set $M_{LSSP} = M_{ssp}^{(m_t)} M^{(t-1)} M_{ssp}^{(m_{t-1})} \ldots M^{(1)} M_{ssp}^{(m_1)}$ and $L = [M_{LSSP}]^{-1}$ then (2.8.27) can be written as [A, B]=LU. This implies that the LSSP method is essentially same as LU factorization, the only difference is that the LSSP method decomposes the matrix by recursive procedure and solve the system in block forms. Similarly we describe the computation process for the LSSP algorithm.

Consider the problem (2.2.1), the computation steps for the LSSP algorithm is

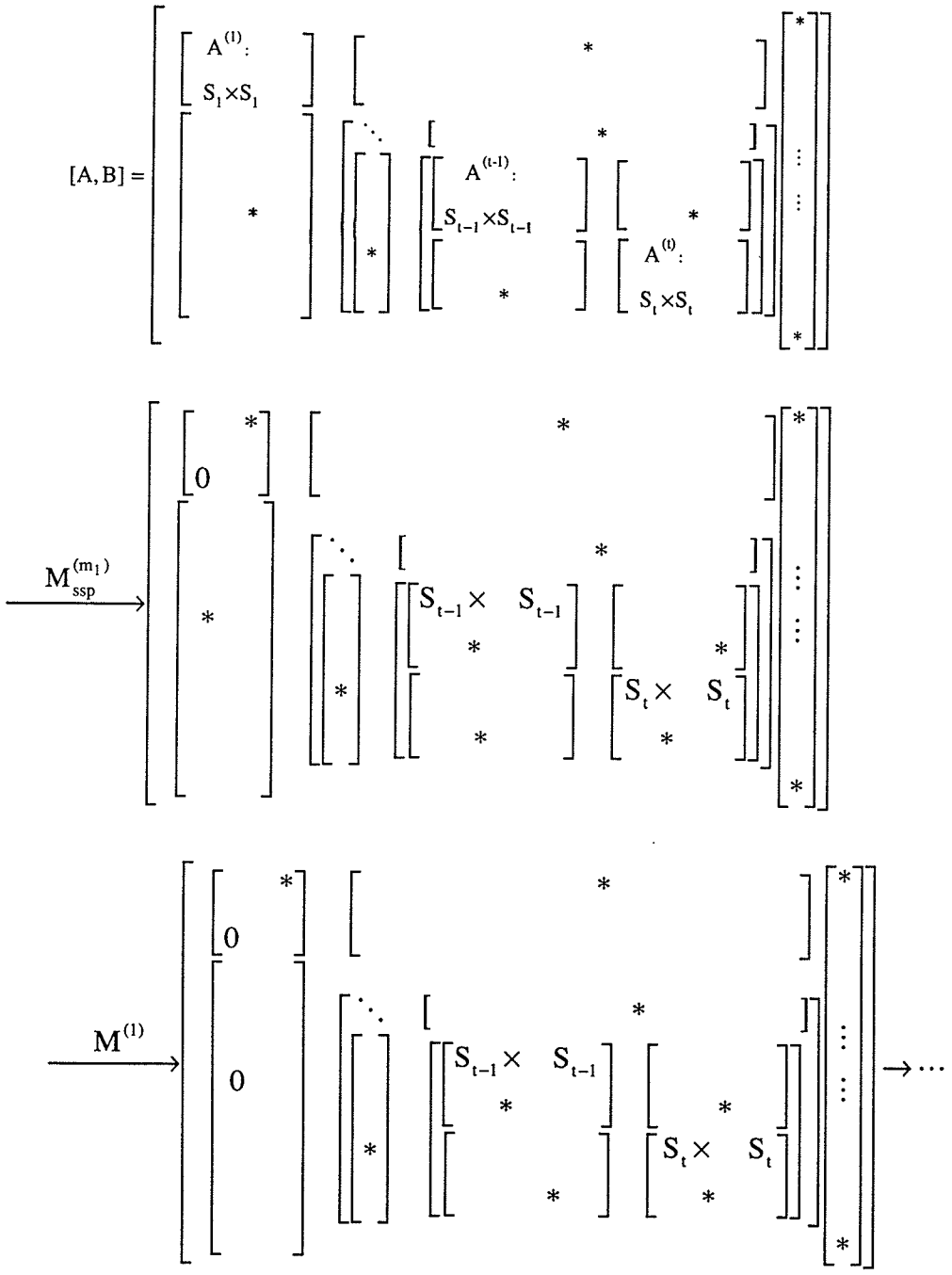
For j= 1, ..., t

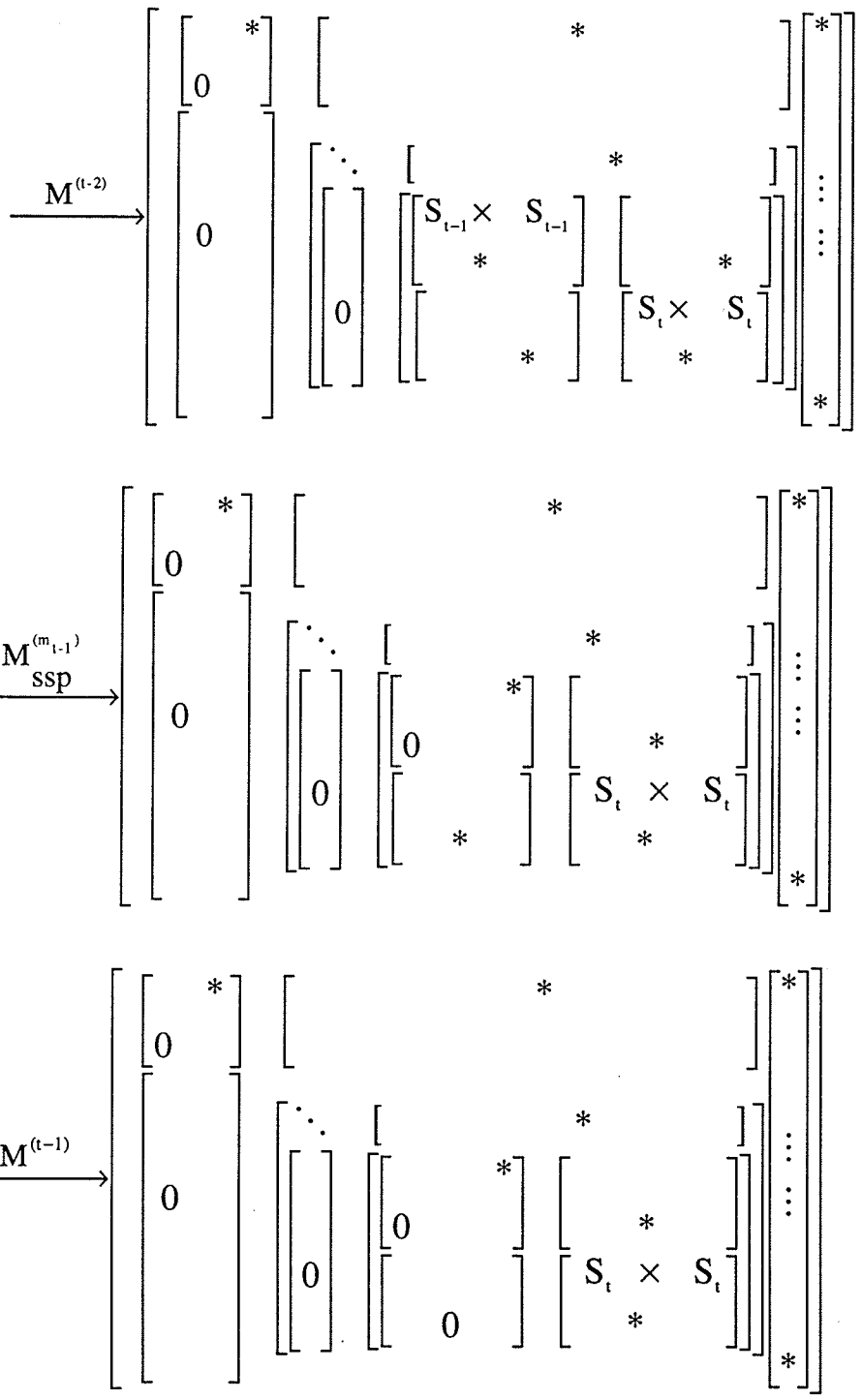
1) Apply the SSP algorithm to the sub system $A^{(j)}X^{(j)} = B^{(j)}$,
where $A^{(j)} \in R^{S_j \times S_j}$, $B^{(j)} \in R^{S_j \times Q_j}$
(This is involved in constructing matrices $M_{ssp}^{(m_j)}$ and computing $[A,B]=M_{ssp}^{(m_j)}[AP_j, B]$)
If j=t, then for i=1, ..., t-1,
do BACKWARD(i $\longrightarrow$ i+1) and stop the computation
otherwise

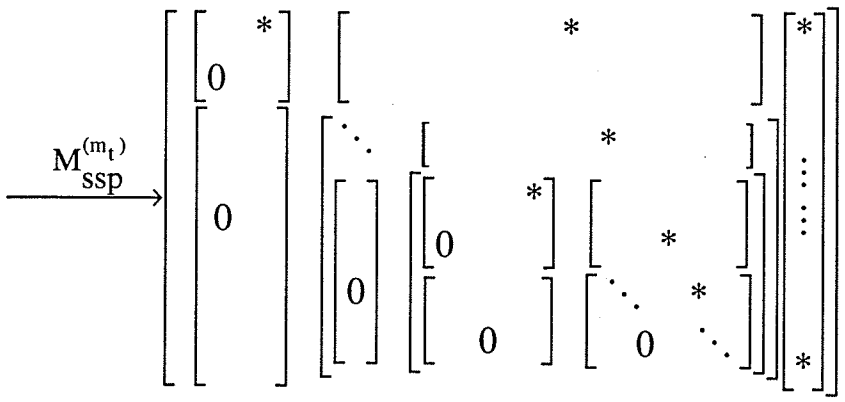
2) Construct matrix $M^{(j)}$

3) Compute $[A,B]=M^{(i)}[A,B]$ and goto 1)

The following matrix patterns show how the LSSP method decompose a matrix A into upper triangular form.




M(t-2)
M(mt-1)ssp
M(t-1)



Summary

In this section we analyzed the recursive block method(i.e. the SSP and LSSP methods) in theory based on the analysis of the basic operation US and TRANSFER. Essentially, the SSP and LSSP methods are LU factorization carried out by the recursive block process which is defined by the recursive formula (RF-k)~(2.1.22). For clarity we rewrote the formula for TRANSFER by substitution of $\left[A_{11}^{(i)}\right]^{-1}A_{12}^{(i)}$ and $\left[A_{11}^{(i)}\right]^{-1}B_1^{(i)}$ for $Z^{(i)}$ and $Y^{(i)}$, and then we introduced matrices $M_k^{(i)}$ (for SSP) and $M^{(j)}$ (for LSSP) to demonstrate the computation process of solving a linear system by the recursive block method. We must stress the point that in practical computation we solve for $Z^{(i)}$ and $Y^{(i)}$ by applying the SSP method to the multiple RHS system $A_{11}^{(i)}\left[Z^{(i)},Y^{(i)}\right]=\left[A_{12}^{(i)},B_1^{(i)}\right]$ instead of computing $\left[A_{11}^{(i)}\right]^{-1}$ directly, and the elimination for blocks $A_{21}^{(i)}$ is never done.

§ 9. Computation Cost

In this section we use the definition of flops as defined by G.H.Golub [2] to describe the arithmetic involved in the LSSP algorithm. For this purpose we first calculate the flops involved in the three basic operations US, TRANSFER and BACKWARD, and then calculate the flops in the SSP algorithm, finally we give an estimation of the flops involved in the LSSP algorithm.

A flop is a floating point operation which is one of the four operations, addition, subtraction, multiplication and division.

Example 2.9.1

A dot product of vectors x and y involves 2n-1 flops because there are n multiplications and n-1 additions, that is, $(x,y) = \sum_{i=1}^{n} x_i y_i$, where $x, y \in R^{n\times 1}$.

We use FP((x,y)) to represent the flops involved in computation (x,y)

i.e. $FP((x,y))=2n-1=2n+O(1)$

Example 2.9.2

The matrix-vector multiplication Z=AX (where $A \in R^{m\times n}$, $X \in R^{n}$, $Z \in R^{m}$) involves 2mn-m flops. In fact, $Z_i = \sum_{k=1}^{n} a_{ik} x_k$, (i=1, ..., m) and $FP(Z_i)=2n-1$, then

$$FP(Z)=m*FP(Z_i)=2mn-m \text{ flops}$$

Now let's consider the SSP algorithm first. We assume that the size of the original matrix A $n=2^m$, where integer m>0.

§ 9.1 Flops in the SSP Algorithm

1) Flops in the US

Assume matrix pattern has the form below after k^{th} step computation.

$$[A,B]=\begin{bmatrix} \ddots & & \\ & \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} & B_1^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} & B_2^{(k)} \end{bmatrix} & \\ & & \ddots \end{bmatrix} \qquad (2.9.0)$$

where $A_{ij}^{(k)} \in R$, $B_i^{(k)} \in R^{1\times(n-2k+1)}$, i,j=1,2

Formula for US can be written as

$$\begin{aligned} V_1 &= [A_{12}^{(k)}, B_1^{(k)}]/A_{11}^{(k)} \\ V_2 &= B_2^{(k)} - A_{21}^{(k)} B_1^{(k)} \\ V_3 &= A_{22}^{(k)} - A_{21}^{(k)} A_{22}^{(k)} \\ V_4 &= V_2/V_3 \\ V_5 &= B_1^{(k)} - A_{12}^{(k)} B_2^{(k)} \end{aligned}$$

Then $FP(V_1)$=n-2k+1, $FP(V_2)$=2(n-2k+1), $FP(V_3)$=2, $FP(V_2/V_3)$=n-2k+1, $FP(V_5)$=2(n-2k+1). The flops in US at k^{th} step is $FP(US^{(k)})$=6(n-2k+1) +2

The total flops in the US involved in the SSP algorithm are

$$FP(US)=\sum_{k=1}^{n/2} FP(US^{(k)})=\sum_{k=1}^{n/2}[6(n-2k+1)+2]=\frac{3}{2}n^2+n \qquad (2.9.1)$$

2) Flops in TRANSFER

Formula for TRANSFER can be simplified as

$$W_i^{(k)} = W_i^{(k)} - A_{21}^{(k)} S_i^{(k)}$$

where $A_{21}^{(k)} \in R^{2^k\times 2^k}$, $W_i^{(k)}, S_i^{(k)} \in R^{2^k\times(n-2^k(2i-1)+1)}$ and k is the number of levels (or the height) of the binary tree, i is the number of the operations TRANSFER involved at level k.

The flops in the operations TRANSFER at k^{th} level are

$$FP(TRANSFER^{(k)})=\sum_{i=1}^{2^{m-(k+1)}}\left[2^k(n-2^k(2i-1)+1)+2*2^k*2^k(n-2^k(2i-1)+1)\right]$$

The total flops in the operations TRANSFER involved in the SSP algorithm are

$$FP(TRANSFER)=\sum_{k=1}^{m-1} FP(TRANSFER^{(k)})$$

$$=\sum_{k=1}^{m-1}\sum_{i=1}^{2^{m-(k+1)}}\left[2^k(n-2^k(2i-1)+1)+2^{2k+1}(n-2^k(2i-1)+1)\right] \qquad (2.9.2)$$

In simplifying (2.9.2) notice that $m = \log_2 n$. We obtain

$$FP(TRANSFER\ )=\sum_{k=1}^{m-1}\left[\frac{n^2+2n}{4}+(n^2+2n)2^{k-1}\right]$$

$$=\frac{1}{2}n^3+\frac{m-1}{4}n^2+\frac{m-5}{2}n$$

$$=\frac{1}{2}n^3+\frac{\log_2 n-1}{4}n^2+\frac{\log_2 n-5}{2}n$$

$$=\frac{1}{2}n^3+O(n^2) \qquad (2.9.3)$$

3) Flops in the BACKWARD

Formula for BACKWARD is $\left(X_1^{(k)}\right)_i = Y_i^{(k)} - Z_i^{(k)}\left(X_2^{(k)}\right)_i$

where $Z_i^{(k)} \in R^{2^k\times 2^k}$, $Y_i^{(k)}, \left(X_1^{(k)}\right)_i, \left(X_2^{(k)}\right)_i \in R^{2^k\times(n-(2^{k+1})*i+1)}$ and

k=1, ... , m-1, i=1, ..., $2^{m-(k+1)}$

k is number of levels of the binary tree,

i is number of operations BACKWARD involved at k^{th} level.

Flops in the operations BACKWARD at k^{th} level are

$$FP(BACKWARD^{(k)})=\sum_{i=1}^{2^{m-(k+1)}}\left[2^k(n-2^{k+1}i+1)+2^{2k+1}(n-2^{k+1}i+1)\right]$$

$$=\frac{n^2+2n}{4}+(n^2+n)2^{k-1}+n2^{2k}$$

Total flops involved the operations BACKWARD in the SSP algorithm are

$$\begin{aligned}
FP(BACKWARD) &= \sum_{k=1}^{m-1} FP(BACKWARD^{(k)}) \\
&= \frac{1}{6}n^3 + \frac{m-3}{4}n^2 + \frac{3m-1}{6}n \\
&= \frac{1}{6}n^3 + \frac{\log_2 n - 3}{4}n^2 + \frac{3\log_2 n - 1}{6}n \\
&= \frac{1}{6}n^3 + O(n^2) \qquad (2.9.4)
\end{aligned}$$

We give the estimation of flops involved in the SSP algorithm by following theorem.

Theorem 9.1.1 The flops involved in the arithmetic computation in the SSP algorithm is $\frac{2}{3}n^3 + O(n^2)$.

Proof. By (2.9.1), (2.9.3) and (2.9.4)

$$\begin{aligned}
FP(SSP) &= FP(US)+FP(TRANSFER)+FP(BACKWARD) \\
&= \frac{2}{3}2^{3m} + (\frac{m}{2}+1)2^{2m} + (m-\frac{8}{3})2^m \\
&= \frac{2}{3}n^3 + (\frac{\log_2 n}{2}+1)n^2 + (\log_2 n - \frac{8}{3})n \\
&= \frac{2}{3}n^3 + O(n^2) \qquad (2.9.5)
\end{aligned}$$

[]

§ 9.2 Flops in the LSSP Algorithm

The decomposition (2.5.2) defines a LSSP blocks for the matrix with size n-by-n. The LSSP algorithm reduces to the SSP algorithm when n is a power of 2 or t=1.

We find it difficult to estimate the exact flops involved in the LSSP algorithm due to the uncertain numbers m_j $(j = 1, \cdots, t)$. Here we just give an upper bound for the flops based on the estimation of the flops involved in the SSP algorithm.

From (2.5.5) we can easily derive that

$$n < 2 \times 2^{m_1} = 2^{m_1+1} \qquad (2.9.6)$$

We set n' be the smallest power of 2 above n, that is, $n' = 2^{m_1+1}$. Then n' is the smallest power of 2 above n. In general, the flops involved in solving a linear system

$$A X = B \qquad (2.9.7)$$

(where $A \in R^{n\times n}$, $X, B \in R^{n\times l}$) by the LSSP algorithm is less than that involved in solving a linear system

$$A'X'=B' \qquad (2.9.8)$$

(where $A' \in R^{n'\times n'}$, $X', B' \in R^{n'\times l}$) by the SSP algorithm. That is,

$$FP(LSSP) < FP(SSP') \qquad (2.9.9)$$

where, FP(SSP') represents the flops in the SSP algorithm for solving system (2.9.8). Then by (2.9.5) the inequality (2.9.9) can be written as

$$FP(LSSP) < \frac{2}{3}(n')^3 + O((n')^2) \qquad (2.9.10)$$

This is the estimation of flops for the LSSP algorithm. For examples, when $n=9=2^3+2^0$, we have $n'=2^4$, then

$$FP(LSSP,\ n=9) < \frac{2}{3}(16)^3 \approx 2730 \ \text{ flops}$$

in the meantime $n=15=2^3+2^2+2^1+2^0$, we have also $n'=2^4$ and

$$FP(LSSP, n=15) < \frac{2}{3}(16)^3 \approx 2730 \ \text{ flops}$$

Both are the same estimations for different size of systems. Therefore the estimation given by the (2.9.10) for the LSSP algorithm is only a rough estimation. The closer n is to n' the more precise the estimation is. Sometime we just use (2.9.5) to calculate the flops for the LSSP algorithm.

§ 10. Memory Allocation Technique

In this section we put forward a scheme of memory allocation for the recursive algorithms established above. The scheme is used in practice.

Assume after k^{th} step computation a matrix pattern is obtained by (2.9.0). The memory allocation scheme is made in the following four steps.

Step 1 Solve $A_{11}^{(k)}[Z^{(k)}, Y^{(k)}] = [A_{12}^{(k)}, B_1^{(k)}]$ for $[Z^{(k)}, Y^{(k)}]$ and assign $Z^{(k)}$ to $A_{12}^{(k)}$ and $Y^{(k)}$ to $B_1^{(k)}$, that is the solution replace the right hand side

$$Z^{(k)} \rightarrow A_{12}^{(k)}$$
$$Y^{(k)} \rightarrow B_1^{(k)}$$

Step 2 Compute $A_{22}^{(k)} - A_{21}^{(k)} Z^{(k)}$ and $B_2^{(k)} - A_{21}^{(k)} Y^{(k)}$ and assign them respectively to $A_{22}^{(k)}$ and $B_2^{(k)}$, that is

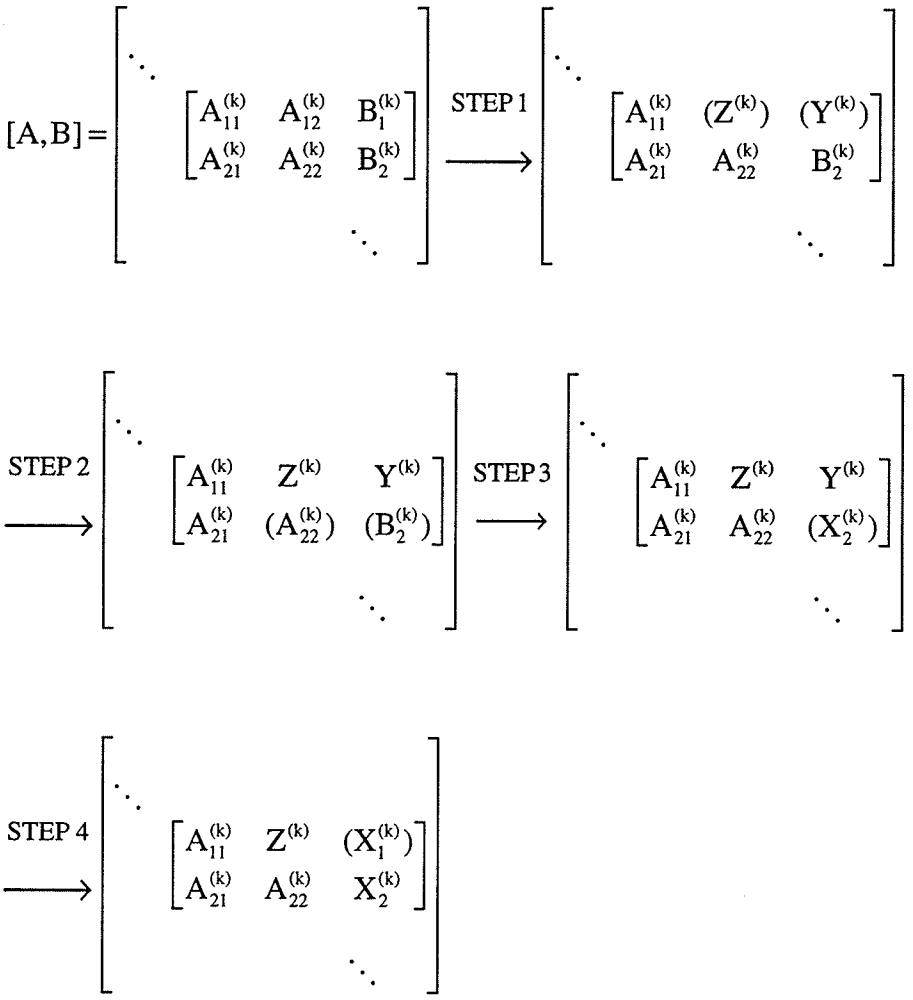
$$A_{22}^{(k)} - A_{21}^{(k)} Z^{(k)} \rightarrow A_{22}^{(k)}$$
$$B_2^{(k)} - A_{21}^{(k)} Y^{(k)} \rightarrow B_2^{(k)}$$

Step 3 Solve $A_{22}^{(k)} X_2^{(k)} = B_2^{(k)}$ for $X_2^{(k)}$ and assign $X_2^{(k)}$ to $B_2^{(k)}$, that is $X_2^{(k)} \rightarrow B_2^{(k)}$

Step 4 Compute $X_1^{(k)} = Y^{(k)} - Z^{(k)} X_2^{(k)}$ and assign $X_1^{(k)}$ to $B_1^{(k)}$, that is $X_1^{(k)} \rightarrow B_1^{(k)}$

The operation US uses all four steps, TRANSFER follows step 2 and BACKWARD follows step 4. Fig. 2.10.1 shows this procedure.

Because the column pivoting is involved in the LSSP algorithm, an extra array of length n is introduced to store the changes of the column indices in matrix A. In addition, two arrays (Ltop(1:m) and Rtop(1:m)) of length m=20 are also introduced to store the sizes of the LSSP blocks at each level of the binary tree. Here m=20 is the maximum height of the binary tree. It corresponds to the matrix of order $n=2^{20}=1048576$.



[Fig. 2.10.1]

§ 11. Flow Chart

In this section we give a flow chart for the recursive block algorithm (LSSP algorithm). The principle of the algorithm is given in the form of flow charts, codes written both in FORTRAN and Matlab are attached to the appendix. The program is reliable and practical and can be applied to real problems. The memory requirement and speed of execution are about the same as the standard Gaussian elimination. But on the computer system with Cache the speed of execution of the LSSP algorithm is much faster than the standard Gaussian elimination with column pivoting. Due to the recursive process is involved the program for LSSP algorithm is much more complicated.

The LSSP algorithm is composed of seven subroutines, they are

1) LSSP: The LSSP main program

2) LRTOP: Matrix partition in LSSP blocks

3) SSP: The SSP algorithm

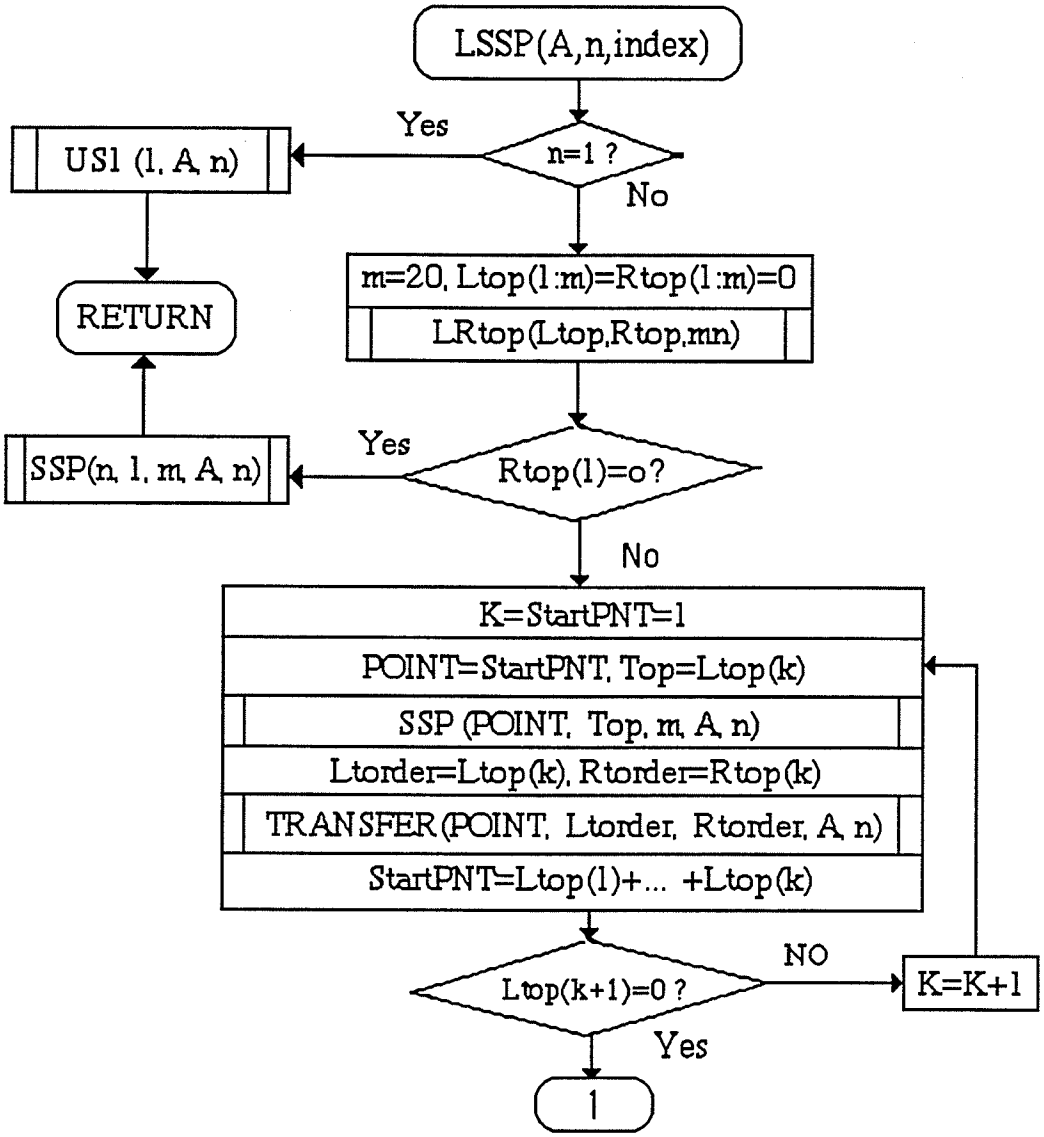
4) US: The operation Unit Solution

5) US1: Special US applied to 1-by-1 multiple RHS linear system

6) TRANSFER: The operation TRANSFER

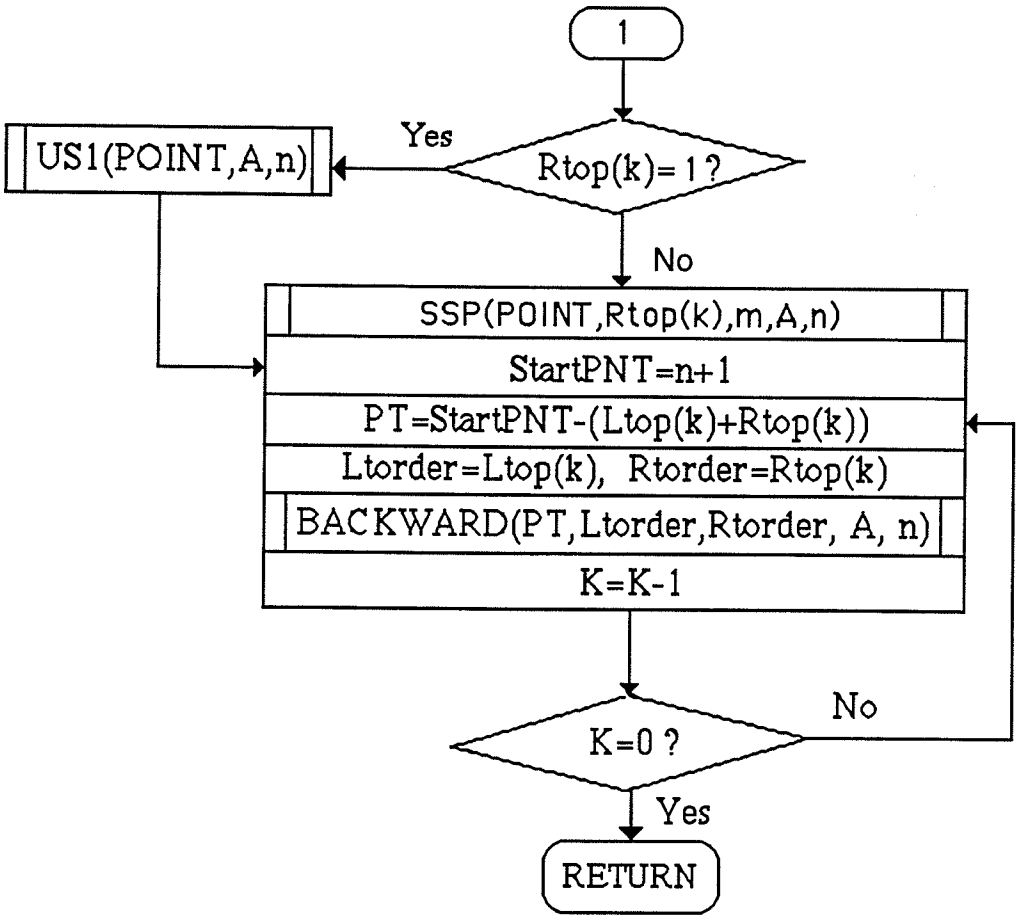
7) BACKWARD: The operation BACKWARD

1.The Left Strictly Symmetric Partition Algorithm



A(1:n,n+1)=[A,B], n: order of A, index(1:n): Column index

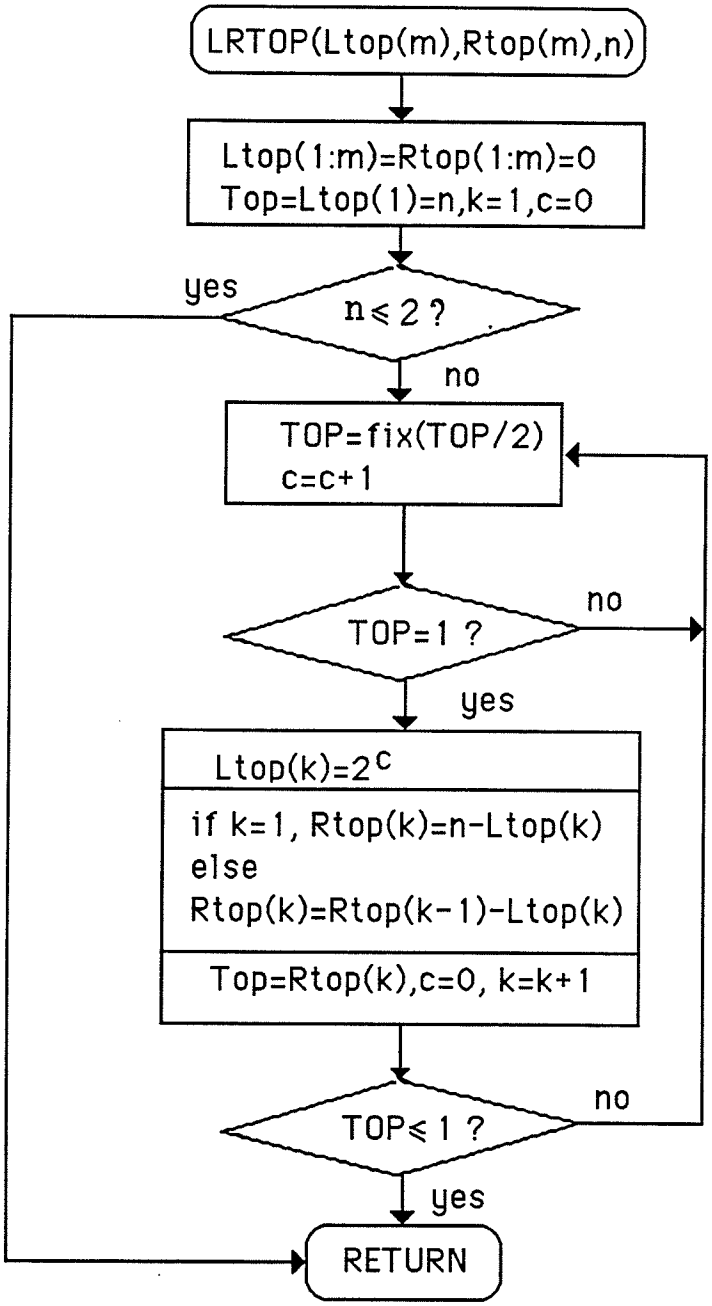
[Fig 2.11.1a]



A(1:n,n+1)=[A,B], n: order of A, index(1:n): Column index

[Fig. 2.11.1b]

2. Partition matrix A into LSS Blocks

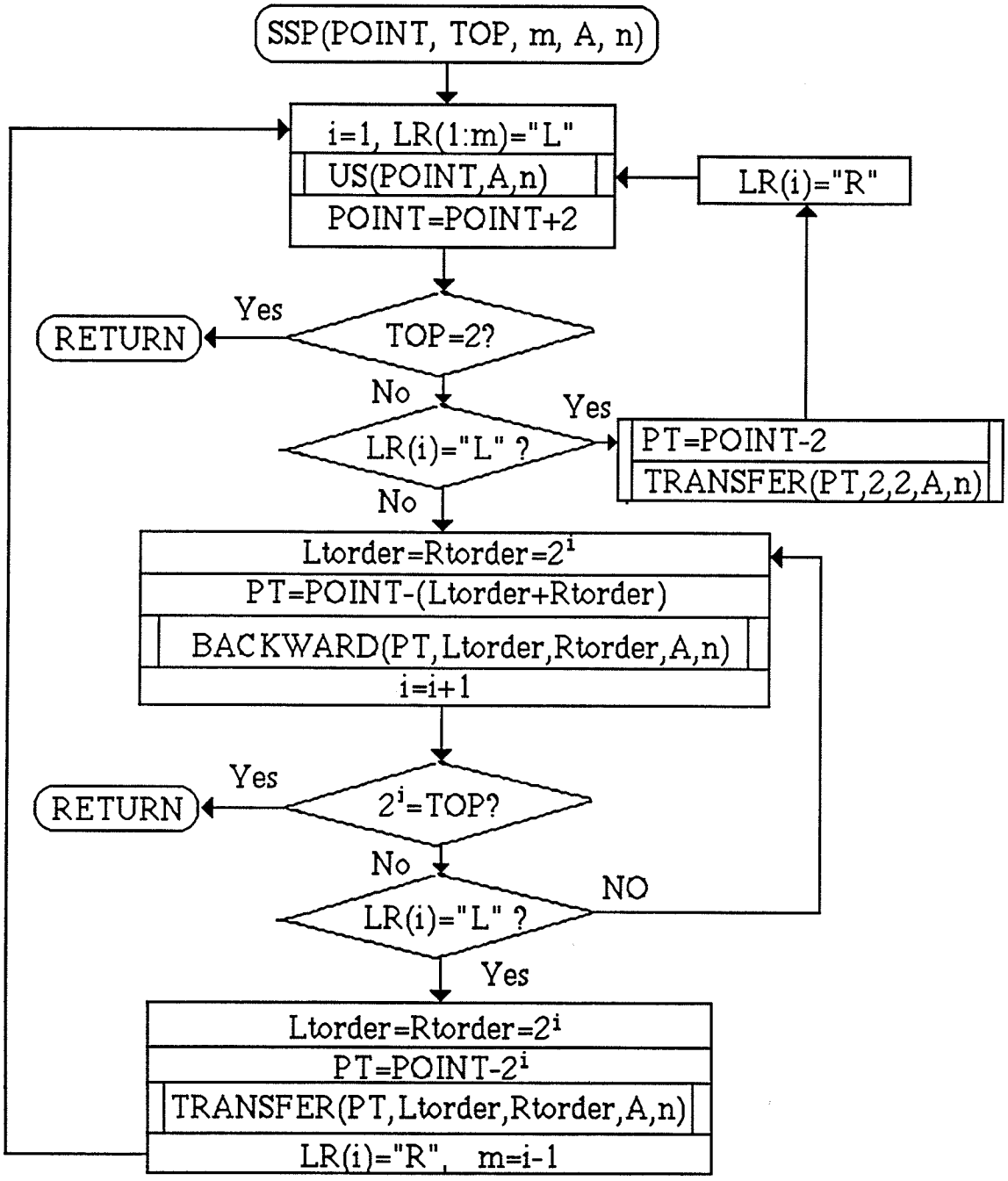


n is decomposed as $n = 2^{m_1} + \cdots + 2^{m_t}$, Set $S_j = 2^{m_j}$. $Q_j = \sum_{i=1+j}^{t} 2^{m_i}$, $j = 0,1,\cdots,t$

Then $Ltop(i) = S_i$, $Rtop(i) = Q_i$. n is order of A, m=20

[Fig. 2.11.2]

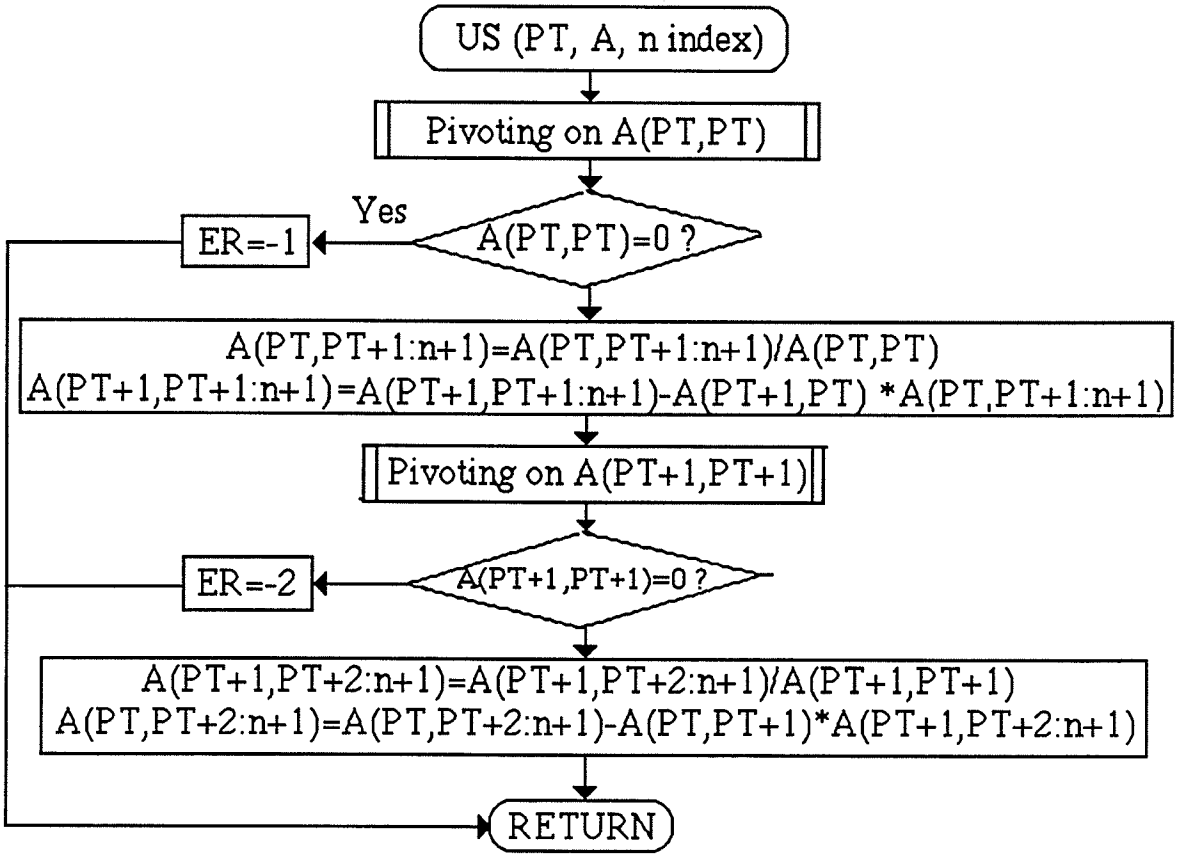
3. The Strictly Symmetric Partition Algorithm



POINT: The position of the first entry in matrix A=[A,B]
TOP: Size of sub blocks in A
n: Size of A, m=20

[Fig. 2.11.3]

4. The Unit Solution (US)

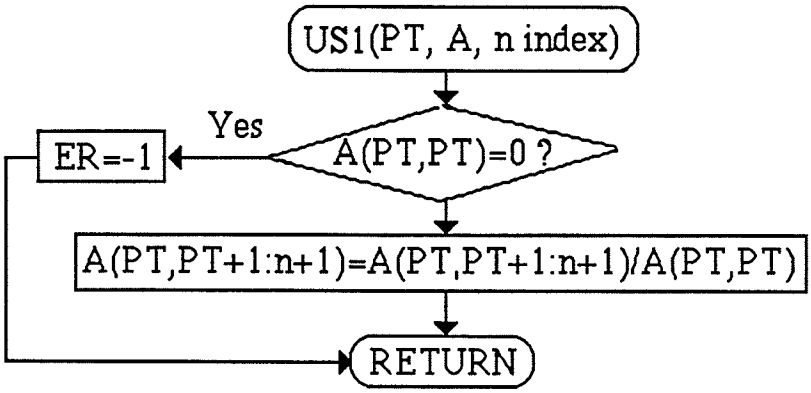


PT:Starting position of $A_{11}^{(k)}$ in A. index:Column index. n: order of A

ER ≥ 0 : A is non singular. ER < 0 : A is singular, ER $= \begin{cases} -1 : A_{11}^{(k)} = 0 \\ -2 : A_{22}^{(k)} = 0 \end{cases}$

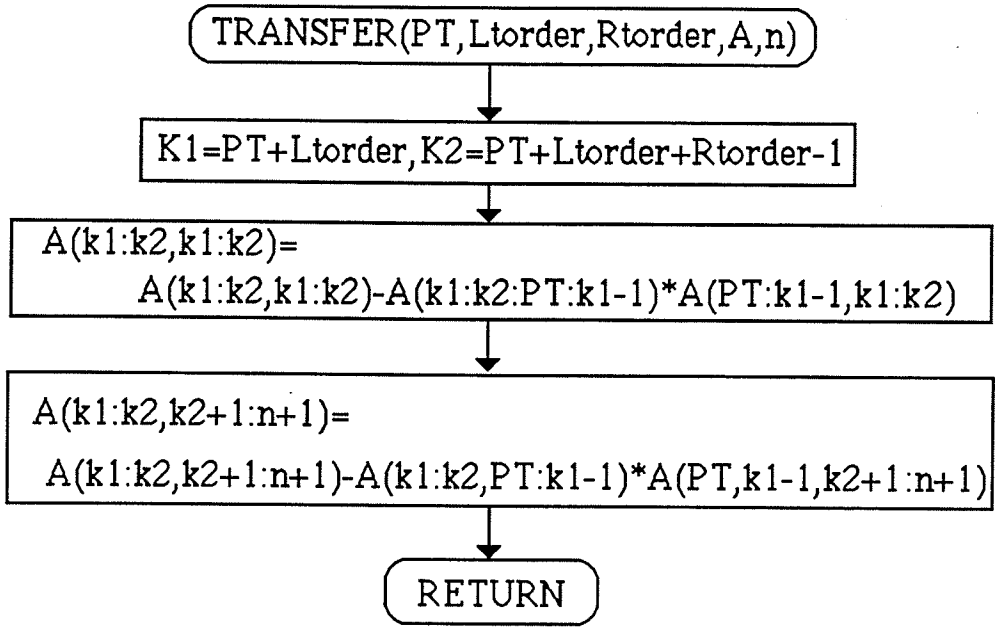
[Fig. 2.11.4]

5. Unit Solution for 1-by-1 Linear System



[Fig. 2.11.5]

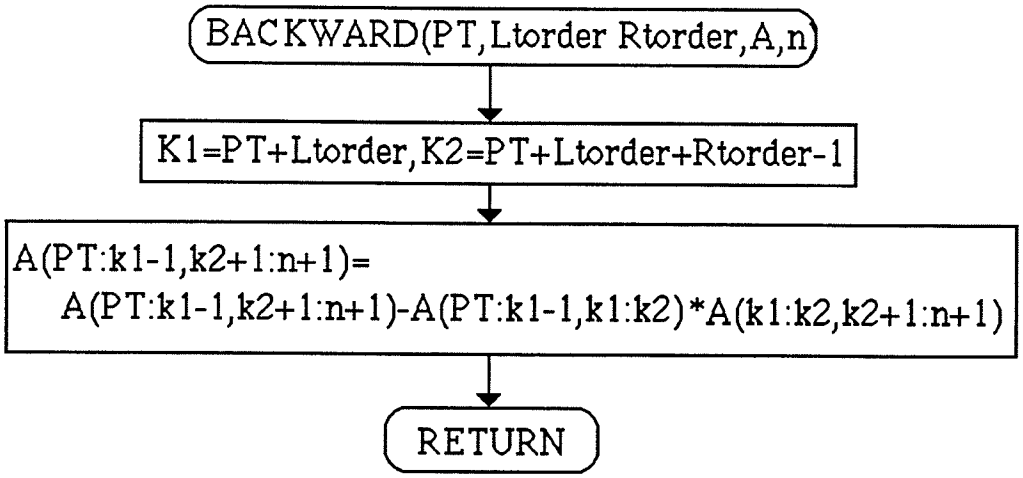
6. TRANSFER



PT: Starting position of $A_{11}^{(k)}$ in A. Ltorder: order of $A_{11}^{(k)}$. Rtorder:order of $A_{22}^{(k)}$

[Fig. 2.11.6]

7. BACKWARD



[Fig.2.11.7]

Chapter III The K-n Partition

In this chapter we discuss a special case for the general block partition discussed in section 2 of Chapter 2. Consider the problem (2.2.1) with general block partition shown in Fig 2.2.1, a special case is when $S_1=S_2=...=S_m=k$, where k is an arbitrary integer which satisfies $mk \le n$.We call this **the K-n partition**. Previously, a 2-by-2 linear system was the smallest system to which the recursive formula (RF-k)~(2.1.22) can be applied. Based on this we successfully established a recursive algorithm called the LSSP algorithm for solving the problem (2.2.1). In this chapter we want to investigate another type of algorithm which does not involve complicated recursive process (that is, the multiple branch recursion) but is generated in the same way as the LSSP algorithm. We consider only the case K=1 in section 1 and K=2 in section 2. Large K would lead to a method similar to the LSSP method because if K>2 a recursive process would be involved in solving this sub system by doing the operation US and then there is no essential difference from the LSSP method.

§ 1. The 1-n Partition

1. Theoretical Derivation

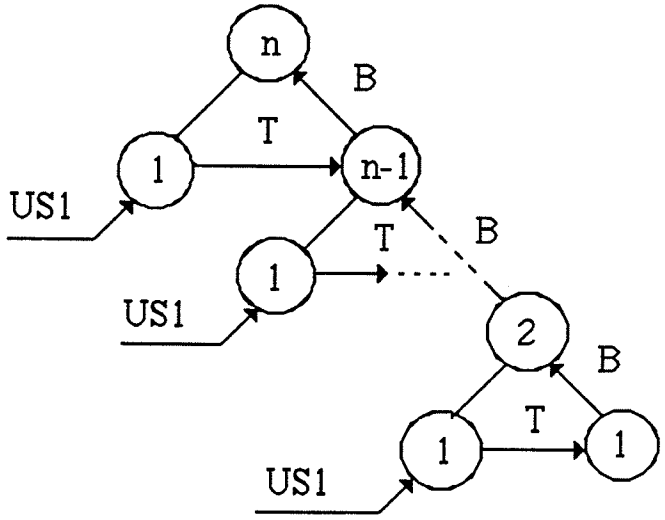
The 1-n partition is the K-n partition for k=1. The Figs 3.1.1 and 3.1.2 show the binary tree and matrix pattern for it. Because $A_{11}^{(k)}$ is always a number for 1-n partition, then for $A_{11}^{(k)} \neq 0$, the recursive formula has the form

$$[Z^{(k)}, Y^{(k)}] = [A_{12}^{(k)}, B_1^{(k)}] / A_{11}^{(k)} \qquad (3.1.1a)$$

$$(A_{22}^{(k)} - A_{21}^{(k)} Z^{(k)}) X_2^{(k)} = B_2^{(k)} - A_{21}^{(k)} Y^{(k)} \qquad (3.1.1b)$$

$$X_1^{(k)} = Y^{(k)} - Z^{(k)} X_2^{(k)} \qquad (3.1.1c)$$

$$X^{(k)} = (X_1^{(k)}, X_2^{(k)})^T \qquad (3.1.1d)$$



1-k partition
US1: Unit Solution applied with a 1-by-1 sub system

[Fig. 3.1.1]

$$A = \begin{bmatrix} \begin{bmatrix} 1\times 1 \end{bmatrix} & \begin{bmatrix} 1\times(n-1) \end{bmatrix} \\ \begin{bmatrix} (n-1)\times 1 \end{bmatrix} & \begin{bmatrix} \begin{bmatrix} 1\times 1 \end{bmatrix} & \begin{bmatrix} 1\times(n-2) \end{bmatrix} \\ \begin{bmatrix} (n-2)\times 1 \end{bmatrix} & \begin{bmatrix} 1\times 1 & 1\times 1 \\ 1\times 1 & 1\times 1 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

[Fig. 3.1.2]

Substitute (3.1.1a) into (3.1.1b) we have

$$A^{(k+1)}X_2^{(k+1)} = B^{(k+1)} \tag{3.1.2a}$$

$$X_1^{(k)} = (B_1^{(k)} - A_{21}^{(k)}X_2^{(k+1)})/A_{11}^{(k)} \tag{3.1.2b}$$

where

$$A^{(k+1)} = A_{22}^{(k)} - A_{21}^{(k)}A_{12}^{(k)}/A_{11}^{(k)} \tag{3.1.2c}$$

$$B^{(k+1)} = B_2^{(k)} - A_{21}^{(k)}B_1^{(k)}/A_{11}^{(k)} \tag{3.1.2d}$$

$$X^{(k+1)} = X_2^{(k)} \tag{3.1.2e}$$

The equations (3.1.2c) and (3.1.2d) can be written in form

$$\left[A^{(k+1)}, B^{(k+1)}\right] = \left[A_{22}^{(k)}, B_2^{(k)}\right] - A_{21}^{(k)}/A_{11}^{(k)}\left[A_{12}^{(k)}, B_1^{(k)}\right] \tag{3.1.3}$$

As before we construct matrix

$$N^{(k)} = \begin{bmatrix} I_1 & \\ L^{(k)} & I_{n-k-1} \end{bmatrix} \tag{3.1.4}$$

where $L^{(k)} = -A_{21}^{(k)}/A_{11}^{(k)}$

The formation of equation (3.1.3) can be then expressed as the second row of the block equation:

$$N^{(k)}\left[A^{(k)}, B^{(k)}\right] = \begin{bmatrix} I_1 & \\ L^{(k)} & I_{n-k-1} \end{bmatrix}\begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} & B_1^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} & B_2^{(k)} \end{bmatrix} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} & B_1^{(k)} \\ 0 & A^{(k+1)} & B^{(k+1)} \end{bmatrix}$$

Furthermore we construct

$$M^{(k)} = \begin{bmatrix} I_k & \\ & N^{(k)} \end{bmatrix} \tag{3.1.5}$$

Such that for k=1, ..., n-1 we have

$$U = M^{(n-1)}\cdots M^{(1)}[A,B] = \begin{bmatrix} A_{11}^{(1)} & A_{21}^{(1)} & B^{(1)} \\ & \begin{bmatrix} A_{11}^{(2)} & A_{21}^{(2)} & B^{(2)} \\ & \begin{bmatrix} \ddots & & \\ & \begin{bmatrix} A_{11}^{(n-1)} & A_{21}^{(n-1)}B^{(n-1)} \\ 0 & A_{11}^{(n)}B^{(n)} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \qquad (3.1.6)$$

The (3.1.6) is upper triangular, then the problem (2.2.1) can be readily solved by back substitution through the following process

For $A_{11}^{(k)} \neq 0$ (k=n,...,1), starting from $X^{(n)} = B^{(n)}/A_{11}^{(n)}$

$$X_2^{(k)} = X^{(k+1)} \qquad (3.1.7a)$$

$$X_1^{(k)} = (B^{(k)} - A_{12}^{(k)}X_2^{(k)})/A_{11}^{(k)} \qquad (3.1.7b)$$

then

$$X^{(1)} = (X_1^{(1)}, X_2^{(1)})^T = (X_1^{(1)}, (X_1^{(2)}, X_2^{(2)})^T)^T$$
$$= \cdots = (X_1^{(1)}, (X_1^{(2)} \cdots (X_1^{(n-1)}, X^{(n)})^T)^T)^T = (X_1^{(1)}, X_2^{(1)}, \cdots, X_n^{(1)})^T$$

In (3.1.6) we set $\tilde{L} = M^{(n-1)}\cdots M^{(1)}$. Since $M^{(i)}$ $(i = 1,\cdots,n-1)$ is lower triangular $\tilde{L}$ is lower triangular, too, and furthermore $\left[\tilde{L}\right]^{-1}$ is lower triangular (see [2] for proof), finally we obtain

$$A=LU \qquad (3.1.8)$$

where $L = \left[\tilde{L}\right]^{-1}$ is lower triangular and U is upper triangular.

This implies that the 1-n partition method is Gaussian elimination. The (3.1.8) is LU factorization of A.

The 1-n partition breaks down when $A_{11}^{(k)} \neq 0$, for this reason pivoting is required at each step of computation. The pivoting is always successful if the original matrix A is non-singular.

2. Computation Cost

The flops involved in the 1-n partition algorithm are $\frac{2}{3}n^3 + O(n^2)$. This is calculated in the following three steps.

Step 1. The flops involved in the operation US1

At k^{th} step of computation: $FP(US1^{(k)})=n-k$

The total: $FP(US1)=\sum_{k=1}^{n-1}(n-k)$ (3.1.9)

Step 2. The flops involved in the operation TRANSFER

At k^{th} step of computation: $FP(TRANSFER^{(k)})=2(n-k)(n-k+1)$

The total: $FP(TRANSFER)=\sum_{k=1}^{n-1}2(n-k)(n-k+1)$ (3.1.10)

Step 3. Flops involved in the operation BACKWARD

At k^{th} step of computation: $FP(BACKWARD^{(k)})=2(n-k)$

The total: $FP(BACKWARD)=2\sum_{k=1}^{n-1}(n-k)$

Hence, $FP("1\text{-}nP")=\sum_{k=1}^{n-1}[(n-k)+2(n-k)(n-k+1)+2(n-k)]$

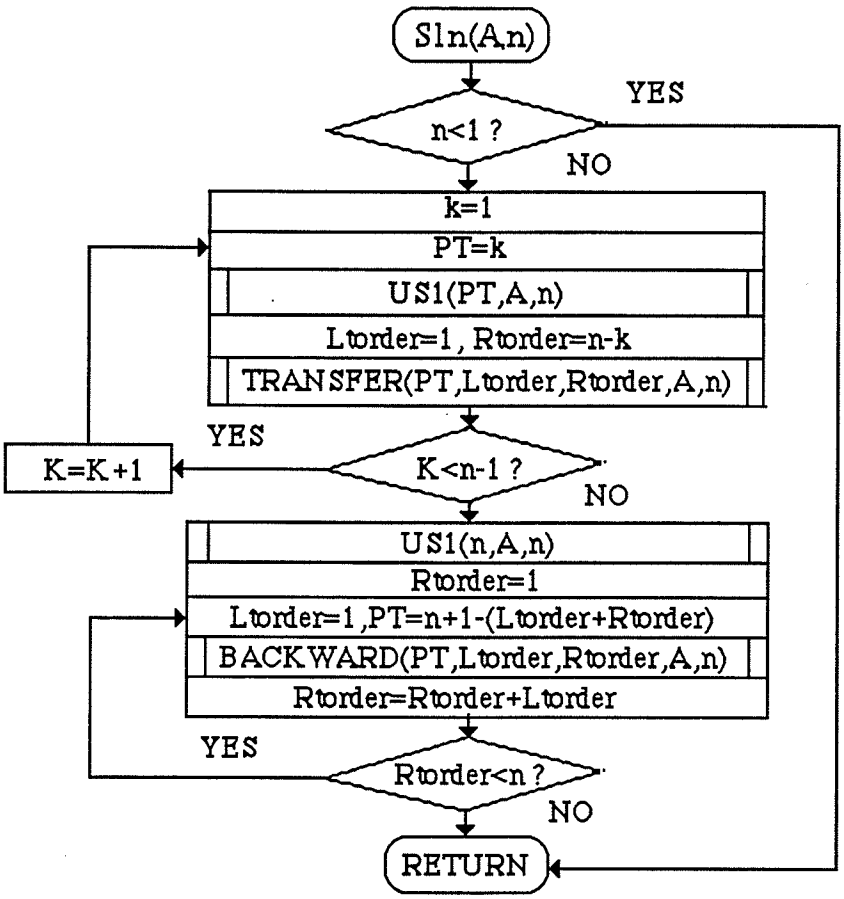
$$=\frac{2}{3}n^3+\frac{3}{2}n^2-\frac{13}{6}n \;=\frac{2}{3}n^3+O(n^2)$$

3. Algorithm

We describe the algorithm of the 1-n partition method by following flow chart, codes written both in Matlab and FORTRAN are attached to appendix.

Memory allocation for the 1-n partition algorithm is easier than that for the LSSP algorithm because the LSSP algorithm needs a few one dimensional arrays that the 1-n partition algorithm doesn't need. In (3.1.3) we need only assign $[A^{(k+1)},B^{(k+1)}]$ to $[A^{(k)},B^{(k)}]$, that is, $[A^{(k)},B^{(k)}]=[A_{22}^{(k)},B_2^{(k)}]-A_{21}^{(k)}/A_{11}^{(k)}[A_{12}^{(k)},B_1^{(k)}]$, then there is no extra memory required. The pivoting we used in the algorithm is row oriented, therefore no extra array is introduced for the storage of column index numbers.

The 1-n Partition Method (Gaussian elimination)

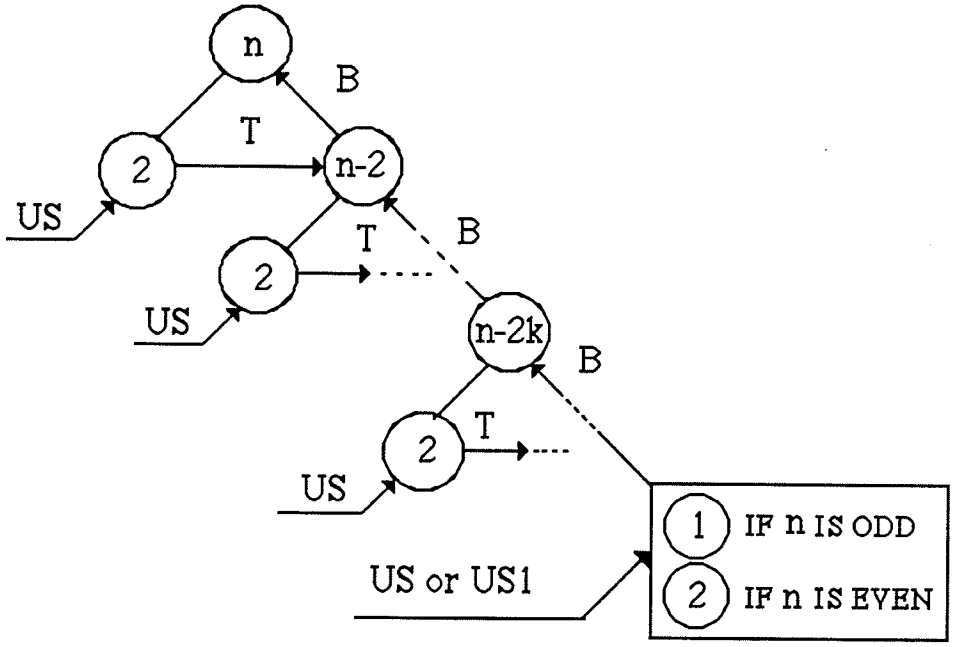


US1: Unit Solution Applied to 1-by-1 sub system

[Fig. 3.1.3]

§ 2. The 2-n Partition

Another special case for the K-n partition method is k=2, we call this case the 2-n partition. In this case $A_{11}^{(k)}$ is always a 2-by-2 matrix to which the Unit Solution(US) can be applied. The matrix pattern and binary tree are shown below.



[Fig. 3.2.1]

$$A = \begin{bmatrix} [2\times 2] & \begin{bmatrix} & & * & & \end{bmatrix} \\ \begin{bmatrix} & \\ * & \\ & \end{bmatrix} & \begin{bmatrix} [2\times 2] & \begin{bmatrix} & & * & & \end{bmatrix} \\ \begin{bmatrix} & \\ * & \\ & \end{bmatrix} & \begin{bmatrix} \ddots & \begin{bmatrix} & & * & & \end{bmatrix} \\ \begin{bmatrix} & \\ * & \\ & \end{bmatrix} & \begin{bmatrix} [2\times 2] & \begin{bmatrix} * \end{bmatrix} \\ \begin{bmatrix} & \\ * & \\ & \end{bmatrix} & \begin{bmatrix} 2\times 2 \\ \text{or} \\ 1\times 1 \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

[Fig. 3.2.2]

Similarly to the derivation for the 1-n partition, we define the matrices

$$N^{(k)} = \begin{bmatrix} I_2 & \\ L^{(k)} & I_{n-(2k+2)} \end{bmatrix} \tag{3.2.1}$$

where $L^{(k)} = -A_{21}^{(k)}[A_{11}^{(k)}]^{-1} \in R^{(n-(2k+2))\times 2}$

and

$$M^{(k)} = \begin{bmatrix} I_{2k} & \\ & N^{(k)} \end{bmatrix} \tag{3.2.2}$$

For k=1, ..., m we have

$$\tilde{U} = M^{(m)}\cdots M^{(1)}[A,B] = \begin{bmatrix} \begin{bmatrix} 2\times 2 \\ 0 \end{bmatrix} & \begin{bmatrix} & & * \\ \begin{bmatrix} 2\times 2 \\ 0 \end{bmatrix} & \begin{bmatrix} \ddots & & * \\ \begin{bmatrix} \\ 0 \end{bmatrix} & \begin{bmatrix} 2\times 2 & \begin{bmatrix} * \\ 2\times 2 \\ \text{or} \\ 1\times 1 \end{bmatrix} \\ 0 & \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \begin{bmatrix} * \\ \vdots \\ * \end{bmatrix} \end{bmatrix} \tag{3.2.3}$$

where m=int[n/2]

$M^{(i)}(i=1,\cdots,m)$ is non singular. Set $\tilde{L} = [M^{(1)}]^{-1}\cdots[M^{(m)}]^{-1}$, then

$$[A,B] = \tilde{L}\tilde{U} \tag{2.3.4}$$

where $\tilde{L}$ is block lower triangular and $\tilde{U}$ is block upper triangular. Then, the 2-n Partition method is a block Gaussian elimination method. The flops involved in the 2-n Partition algorithm are also $\frac{2}{3}n^3 + O(n^2)$. This result is calculated by

$$FP(US) = \sum_{k=1}^{m} 6(n-2k+1) = 6m(n-m) \approx \frac{3}{2}n^2$$

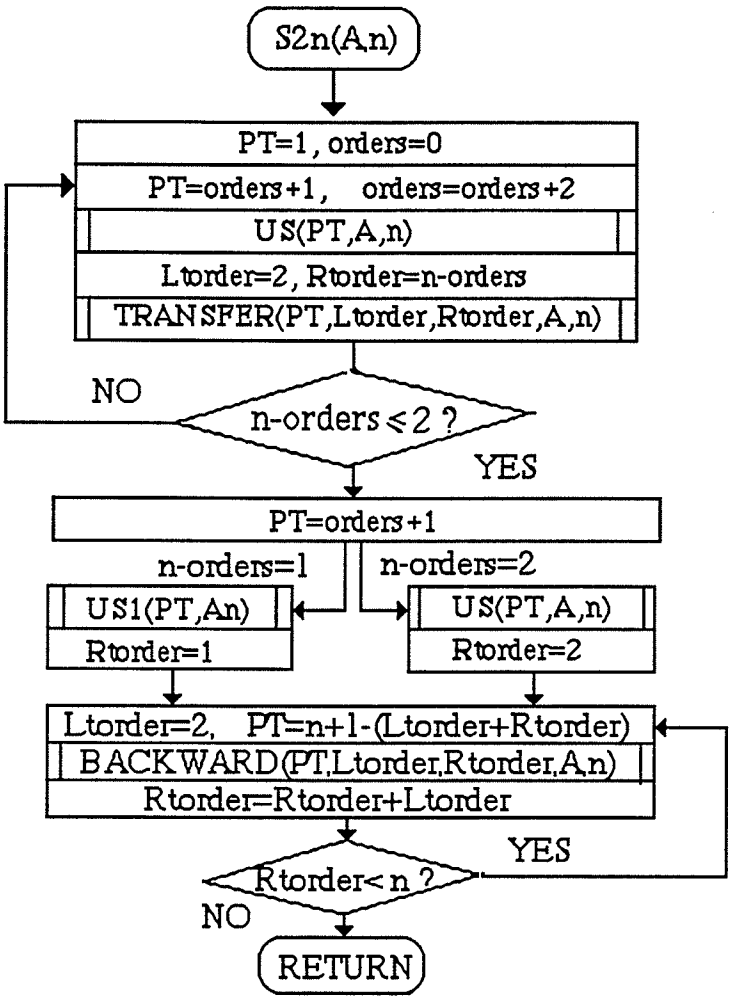
$$FP(TRANSFER) = \sum_{k=1}^{m} 4(n-2k)(n-2k+1) \approx \frac{2}{3}n^3 - n^2 - \frac{2}{3}n$$

$$FP(BACKWARD) = \sum_{k=1}^{m} 4(n-2k) \approx n^2 - 2n$$

Therefore, $FP(\text{"2-nP"}) = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{8}{3}n \approx \frac{2}{3}n^3 + O(n^2)$

The algorithm for the 2-n Partition is given by the following flow chart. Codes for the algorithm written in Matlab and FORTRAN are attached to the appendix.

The 2-n Partition Method (S2n)



[Fig. 3.3]

Chapter IV Results and Conclusion

§ 1. Four Indices for Algorithm Evaluation

We use the following four indices to evaluate the LSSP method applied to the solution of a linear system

$$A X = B \tag{4.1.1}$$

where, $A \in R^{n\times n}$, $X, B \in R^{n\times 1}$

1) CPU Time

This is the time taken by a algorithm running on computer for solving (4.1.1). The less the time consumes, the better the algorithm is for solving a given problem on the same computer.

2) Relative Error : $\|X-\tilde{X}\|_\infty / \|X\|_\infty$

X is an exact solution to (4.1.1) and $\tilde{X}$ is a computed solution which is an approximation to X. We use relative error to evaluate the precision of the computed solution comparing with the exact solution. If the machine base $u=10^{-d}$ and $K_\infty(A) = 10^q$, the computed solution $\tilde{X}$ satisfies

$$\frac{\|X-\tilde{X}\|_\infty}{\|X\|_\infty} \cong 10^{-d+q} \tag{4.1.2}$$

3) Relative Residual: $\|r\|_\infty / (\|A\|_\infty \|\tilde{X}\|_\infty)$, where, r is residual defined by

$$r = B - A\tilde{X} \tag{4.1.3}$$

We have the following formulae involving the residual.

$$\because \quad r = B - A\tilde{X} = AX - A\tilde{X} = A(X-\tilde{X})$$
$$X - \tilde{X} = A^{-1}r$$

$$\therefore \quad \frac{|X-\tilde{X}|}{|X|} \le \frac{\|A^{-1}\|\,|r|}{|X|} \le \frac{\|A^{-1}\|\,|r|}{|B|/|A|} = \frac{|r|}{|B|}K(A) \tag{4.1.4}$$

where $K(A) = \|A^{-1}\| * \|A\|$

If $\tilde{X}$ satisfies the perturbed problem $(A+E)\tilde{X}=B$

then $\|r\| = \|B - A\tilde{X}\| = \|E\tilde{X}\| \le \|E\|\|\tilde{X}\|$

$$\therefore \quad \frac{\|r\|}{\|A\|\|\tilde{X}\|} \le \frac{\|E\|}{\|A\|} \qquad (4.1.5)$$

4) Extra Memory (RAM) Occupation

We use extra memory for one dimensional arrays in the algorithm. In general, the less extra memory an algorithm takes up, the better the algorithm is, assuming there is no loss of computation speed

§ 2. Test Program Design

According to the above four indices used to evaluate an algorithm of solving the problem (4.1.1), we design a test program for this purpose as follows.

1° Generate a n-by-n non-singular matrix A. We shall show the details of how to generate the matrix later.
2° Generate a random n-vector X
3° Calculate B=AX
4° Solve the linear system AX=B by the tested algorithm for the computed solution $\tilde{X}$
5° Record CPU time for solving the problem
6° Compute relative error and relative residual

Repeat this process for different values of n.

1. Generate a random and well conditioned matrix

   1) generate a random n-by-n matrix A_1
   2) Compute its singular value decomposition $A_1=UDV^T$ where U and V are orthogonal.

3) Generate a random n-vector $T=[t_1,\ldots, t_n]$
where $t_i \neq 0$ $(i=1,\cdots,n)$

4) Construct diagonal matrix D_1

$$D_1 = \begin{bmatrix} t_1 & & 0 \\ & \ddots & \\ 0 & & t_n \end{bmatrix}$$

5) Compute $A=UD_1V^T$

2. Banded and Well-conditioned Matrix

$$A = \begin{bmatrix} 2 & 1 & & & \\ 1 & 2 & 1 & & \\ & & \ddots & & \\ & & 1 & 2 & 1 \\ & & & 1 & 2 \end{bmatrix}_{n\times n}$$

3. Hilbert Matrix

$$A(i,j)=1/(i+j-1) \qquad \text{for } i,j=1, \ldots, n$$

4. Vandermonde Matrix

$$A=\begin{bmatrix} 1 & \cdots & 1 \\ x_0 & \cdots & x_{n-1} \\ \vdots & & \vdots \\ x_0^{n-1} & \cdots & x_{n-1}^{n-1} \end{bmatrix}$$

where $(x_0,\cdots,x_{n-1})$ are generated randomly.

§ 3. Discussion and Conclusion

The tables attached to the end of this chapter show the results from running different algorithms on SUN Station-UNIX operating system. Software package used to execute the algorithms was Matlab. Our purpose is to evaluate the LSSP algorithm, and Gaussian elimination (i.e. the 1-n Partition method) and Block Gaussian elimination (i.e. the 2-n

Partition method) on the same problem. To make contrast we also evaluate the standard Gaussian elimination. The following is the conclusion.

1) As derived in Chapter III the flops involved in the algorithms of the LSSP, the 1-nP and the 2-nP for solving problem (4.1.1) are all equal to $\frac{2}{3}n^3 + O(n^2)$. The results in the tables support this conclusion by showing the same CPU time consumption for solving the same problems by the three different algorithms.

2) For well conditioned systems as shown in tables 1, 2, 4 and 6, the solutions found by the LSSP algorithm have the same high precision as that found by Gaussian elimination. The residuals shown in the tables with high precision also indicate that the LSSP algorithm is as stable as Gaussian elimination. The stability of Gaussian elimination algorithm is analyzed by the following discussion (see [2] for details).

Gaussian elimination with partial pivoting produces a permutation matrix P and computed L and U, which we denote by $\tilde{L}$ and $\tilde{U}$, satisfying

$$P\tilde{L}\tilde{U}=A+H \quad \text{with} \quad |H| \le 3(n-1)u(|A|+|\tilde{L}||\tilde{U}|)+O(u^2)$$

where $|H| = \max_{i,j}(|h_{ij}|)$, the largest absolute value entry of H.

If $\tilde{X}$ is computed solution to AX=B, found by using $\tilde{L}$ and $\tilde{U}$, then $\tilde{X}$ satisfies

$$(A+E)\tilde{X}=\tilde{B} \quad \text{with} \quad |E| \le nu(3|A|+5|\tilde{L}||\tilde{U}|)+O(u^2)$$

where u is machine precision. (See [2] for the proof). Therefore

$$\|E\|_\infty \le nu(3\|A\|_\infty + 5n\|\tilde{U}\|_\infty)+O(u^2)$$

where, $\|\tilde{L}\|_\infty \le n$ for Gaussian elimination with pivoting. If the matrix A is well-conditioned, then the computed solution $\tilde{X}$ will have a small relative error (Reference [2] equation (3.5.2)).

In chapter III (section 8) we analyzed the LSSP method in theory and noticed that the LSSP method is essentially the same as LU factorization method. However as we stressed

that the way of factoring A into LU form for the LSSP algorithm is different from that for Gaussian elimination and that the column pivoting in the LSSP algorithm is also different from the row pivoting in the above Gaussian elimination algorithm. Therefore the theoretical analysis on the stability of the LSSP algorithm should be different from the above analysis on that of Gaussian elimination. Thus, further theoretical analysis of roundoff error and stability for the LSSP algorithm is recommended in the future studies.

3) For ill-conditioned matrix as shown in Tables 3 and 4, the relative errors go up while the condition numbers arise. This can be explained by (4.1.2). In our case $u=10^{-17}$, that is, the correct significant digit d=17 and condition number $K(A)=10^q$. The relative errors will be badly bounded when q>17. From the tables 3 and 4 we conclude that both Gaussian elimination and the LSSP algorithms failed to solve the ill-conditioned systems like Vandermonde and Hilbert of high orders.

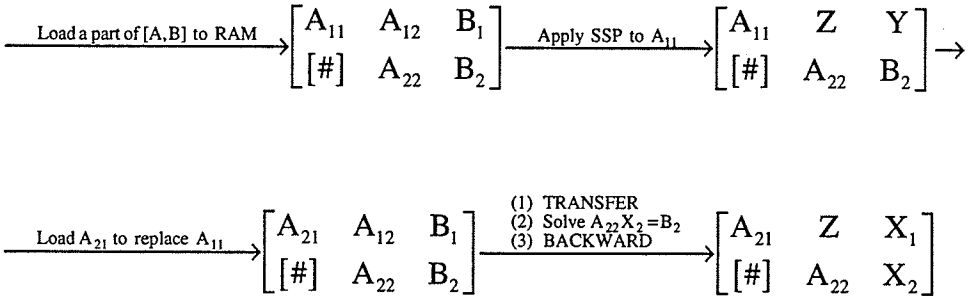
4) Due to the column pivoting involved in the LSSP algorithm, an extra array with length n is introduced to record the change of column indices in A. No such work space is required for Gaussian elimination and block Gaussian elimination. But on the other side if we wisely re-arrange the memory allocation the LSSP algorithm can save almost $\frac{n^2}{4}$ memory units for the matrix of order n. We roughly describe this method below. The LSSP decomposition defined by (2.5.2)~(2.5.4) to the linear system (4.1.1) gives

$$n = S_1 + Q_1 , \text{ where } S_1 = 2^{m_1}, \ Q_1 = \sum_{i=2}^{t} 2^{m_i}$$

and matrix pattern, $$[A,B] = \begin{bmatrix} A_{11} & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \end{bmatrix}$$

where, $A_{11} : S_1 \times S_1$, $A_{12} : S_1 \times Q_1$, $A_{21} : Q_1 \times S_1$, $A_{22} : Q_1 \times Q_1$.

Assume matrix A and vector B are stored in certain media such as disk out side CPU and are loaded into RAM during the computation, the following memory allocation scheme saves the storage of A_{21} which is equal to $\frac{n^2}{4}$ memory units for SSP of the matrix A and $\frac{Q_1 * S_1}{4}$ for LSSP.



where, the symbol [#] represents the RAM unoccupied or saved. This scheme offers benefit to the solution of a large size linear system.

5) We also compared the LSSP algorithm with the standard Gaussian elimination with column pivoting. The results in the Tables 5 and 6 show that the LSSP algorithm is more effective than standard Gaussian elimination in solving linear systems with high orders. This is because the Cache installed in the Sun Station speeds up the computation for solving a small size systems. The LSSP algorithm solves a linear system by dividing it into smaller sub systems. Then it gets a speedup in computation. The standard Gaussian elimination solves a linear system in full size, therefore there is no advantages from using Cache that could be offered to it in solving large size linear systems.

6) From the discussion in chapter 2 and the practical computation we notice that the LSSP algorithm uses a very little costs for the matrix partition, that is, for the binary decomposition for the order of a matrix. Therefore it saves operations in doing "pre-work" for solving linear systems.

Conclusion

The LSSP method is a recursive block method applied to the solution of a linear system. It uses a very little cost for matrix partition. Comparing with Gaussian elimination with partial pivoting the LSSP algorithm with column pivoting is stable. It uses the same computational cost as the Gaussian elimination algorithm in solving the same problem. It computes the solution with as high precision as Gaussian elimination. Because the LSSP method solves a linear system based on a successive process of partitioning the system into 2-by-2 sub systems, the LSSP algorithm works in a higher speed than the standard Gaussian elimination on the computer systems installed with Cache. Due to the recursive process involved the LSSP algorithm is much more difficult to program than Gaussian elimination. And as Gaussian elimination the LSSP method fail to solve the highly ill-conditioned system such as Vandermonde and Hilbert systems with high orders. The improvement of this algorithm is recommended in order to apply it to sparse linear system, and to special linear systems with a coefficient matrix of a specialized nature (e.g. symmetric, banded etc.).

Table 1 (Random Matrix) (Matlab)

Algorithm	Time (Sec)	R-error	R-residual
(Order =10, K(A) = 76.3898, K1=2.8797, K2 =26.6273)			
1-nP	0.0752	8.6491e-16	8.5815e-17
2-nP	0.0746	9.2669e-16	8.5815e-17
LSSP	0.0901	8.6491e-16	8.5815e-17
(Order =30, K(A) = 337.7146 , K1=6.1012, K2 =55.3522)			
1-nP	0.4241	1.5459e-15	9.5014e-17
2-nP	0.8271	1.0435e-15	1.1402e-16
LSSP	1.2934	4.0386e-15	8.5513e-17
(Order =90, K(A) = 1.4328e+03, K1 = 11.7819, K2 = 121.6093)			
1-nP	3.5428	1.1086e-14	1.4288e-16
2-nP	2.7768	8.1295e-15	1.5682e-16
LSSP	1.7981	5.1733e-15	1.5333e-16
(Order 270, K(A) = 3.7244e+03, K1 =15.9884, K2 =232.9456)			
1-nP	37.4081	3.1834e-14	3.5589e-16
2-nP	57.0960	2.7547e-14	5.1459e-16
LSSP	44.4526	1.8531e-14	3.8474e-16
(Order =500, K(A) = 2.0723e+04, K1 =20.9999, K2 =986.8376)			
1-nP	271.7682	5.0936e-14	4.9216e-16
2-nP	248.0661	7.4946e-14	6.4470e-16
LSSP	282.1017	7.7862e-14	3.6552e-16

R-error: Relative Error,
R-residual: Relative Residual,

K1: Infinite Norm of the Tested Matrix,
K2: Infinite Norm of the Inverse Tested Matrix
K(A) : Condition Number = K1 * K2

1-nP: The 1-n Partition Method (Gaussian Elimination)
2-nP: The 2-n Partition Method (Block Gaussian Elimination)
LSSP: The Left Strictly Symmetric Partition Method

Table 2 **(Band Matrix)** **(Matlab)**

Algorithm	Time (Sec)	R-error	R-residual
(Order =10, K(A) =60.0000, K1 =4, K2 =15.0000)			
1-nP	0.0656	5.9332e-16	7.0146e-17
2-nP	0.0570	5.4388e-16	3.2962e-17
LSSP	0.0934	5.7684e-16	6.5925e-17
(Order =30, K(A) = 480.0000, K1 = 4, K2 = 120.0000)			
1-nP	0.2456	1.7635e-15	5.7821e-17
2-nP	0.2675	1.7635e-15	1.1564e-16
LSSP	0.3336	1.8503e-15	1.1564e-16
(Order =90, K(A) = 4.1400e+03, K1 =4, K2 =1.0350e+03)			
1-nP	1.5975	9.2253e-15	1.1115e-16
2-nP	2.3655	9.2253e-15	1.1115e-16
LSSP	2.7917	9.3364e-15	2.2230e-16
(Order = 270, K(A) = 3.6720e+04, K1 = 4, K2 = 9.1800e+03)			
1-nP	42.9319	6.1400e-14	1.1123e-16
2-nP	47.4241	6.0955e-14	1.1123e-16
LSSP	55.3007	6.1066e-14	2.2246e-16
(Order 500, K(A) = 1.2550e+05, K1 = 4, K2 =3.1375e+ 04)			
1-nP	390.6711	3.2926e-13	1.1116e-16
2-nP	380.8113	3.2926e-13	1.1116e-16
LSSP	422.0522	3.2937e-13	2.2232e-16

The above results were obtained by applying the tested algorithms to the linear system AX=B with deferent orders, where vector $X=(X_1, ..., X_n)$ is generated randomly with X_i (i=1,...n) non-zero to form the vector B, and the band matrix has the following form:

$$A = \begin{pmatrix} 2 & 1 & & & \\ 1 & 2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \\ & & & 1 & 2 \end{pmatrix}$$

Table 3 (Hilbert Matrix) (Matlab)

Algorithm	Time (Sec)	R-error	R-residual
(Order =10, K(A) = 3.5354e+13, K1 =2.9290, K2 =1.2070e+13)			
1-nP	0.0817	2.0291e-04	4.2833e-17
2-nP	0.0780	4.8085e-04	8.5666e-17
LSSP	0.0580	1.2520e-04	1.2850e-16
(Order =30, K(A) = 9.2509e+19, K1 =3.9950, K2 =2.3156e+19)			
1-nP	0.6192	5.21513	2.2589e-18
2-nP	2.4383	4.55240	9.1163e-17
LSSP	1.2203	4.74162	1.7381e-17
(Order =90, K(A) =1.1862e+20, K1 =5.0826, K2 =2.3339e+19)			
1-nP	1.9566	327.9953	2.4009e-18
2-nP	2.2475	305.4114	5.5997e-18
LSSP	2.4123	375.9348	1.3548e-17
(Order =270, K(A) =1.1125e+21, K1 =6.1775, K2 = 1.8010e+20)			
1-nP	45.3755	916.6968	2.2073e-18
2-nP	47.0297	1.8414e+03	3.3960e-18
LSSP	35.3328	3.9729e+04	8.6268e-18
(Order =500, K(A) =1.0884e+21, K1 =6.7928, K2 = 1.0884e+21)			
1-nP	357.6326	3.1641e+04	1.1719e-18
2-nP	375.2853	973.8766	9.3870e-18
LSSP	407.7207	2.8924e+04	3.2120e-17

The above results were obtained by applying the tested algorithms to the Hilbert system AX=B with deferent orders, where vector X=(X_1, ..., Xn) is generated randomly with Xi (i=1,...n) non-zero to form the vector B. The Hilbert matrix is defined as A(i,j)=1/(i+j-1), which is a highly ill-conditioned matrix.

Table 4 (Vandermonde Matrix) **(Matlab)**

Algorithm	Time (Sec.)	R-error	R-residual
(Order =10, K(A) =8.2606e+08, K1 =10, K2 = 8.2606e+07)			
1-nP	0.0773	3.3029e-09	4.7874e-17
2-nP	0.0698	7.4637e-10	4.7874e-17
LSSP	0.0845	2.4046e-09	4.7874e-17
(Order =30, K(A) = 7.0770e+21, K1 =30, K2 = 2.3590e+20)			
1-nP	0.2767	7.1373e+03	6.5851e-18
2-nP	0.2987	7.8616e+03	1.2587e-18
LSSP	0.2957	157.4123	1.3792e-17
(Order = 90, K(A) =2.9278e+23, K1 = 90, K2 = 3.2531e+21)			
1-nP	2.5488	6.2297e+04	3.0406e-18
2-nP	3.5049	2.3641e+07	1.3316e-18
LSSP	2.8706	1.7696e+04	4.1811e-18
(Order = 270, K(A) =1.2682e+27, K1 =270, K2 =4.6970e+24)			
1-nP	40.2374	2.8701e+05	4.2562e-19
2-nP	37.2676	4.2516e+06	3.4842e-19
LSSP	39.2772	7.5619e+04	3.6555e-19
(Order =500, K(A) = 1.7632e+23, K1 =500, K2 = 3.5265e+20)			
1-nP	292.5916	4.6198e+05	5.1980e-18
2-nP	329.3078	2.4488e+04	2.2262e-18
LSSP	297.4342	1.4010e+05	1.6361e-18

The above results were obtained by applying the tested algorithms to the Vandermonde system AX=B with deferent orders, where vector X=(X_1, ..., Xn) is generated randomly with Xi (i=1,...n) non-zero to form the vector B.

Table 5 **(Random Matrix)** **(Matlab)**

Algorithm	Time (Sec)	R-error	R-residual
Order = 5 **K(A) = 3.4670**			
SGE	0.0225	1.2813e-16	5.0139e-17
LSSP	0.0774	1.4643e-16	7.2110e-17
Order =10 **K(A)= 11.8374**			
SGE	0.0772	1.1338e-15	1.7484e-16
LSSP	0.1030	5.6690e-16	7.4931e-17
Order =15 **K(A) =19.5213**			
SGE	0.1844	3.9593e-16	1.0446e-16
LSSP	0.1300	6.2218e-16	7.8342e-17
Order =30 **K(A) = 185.7966**			
SGE	2.1208	2.8259e-15	1.2542e-16
LSSP	0.6605	5.2953e-15	6.2710e-17
Order = 90 **K(A) = 98.3045**			
SGE	12.4159	3.8446e-15	1.5444e-16
LSSP	1.83280	5.0485e-15	1.2355e-16
Order = 270 **K(A) = 475.0792**			
SGE	231.3939	3.0909e-14	5.0587e-16
LSSP	60.3790	4.4568e-14	3.2694e-16
Order = 500 **K(A) =1.6307e+03**			
SGE	832.9084	7.3133e-14	4.7107e-16
LSSP	297.7588	1.0903e-13	3.5681e-16

R-error: Relative Error, R-residual: Relative Residual, K(A) : Condition Number
SGE: Standard Gaussian Elimination with Partial Pivoting LSSP: The Left Strictly Symmetric Partition Method

The above results were obtained by applying the algorithms to the linear system AX=B with deferent orders, where vector $X=(X_1, ..., X_n)$ is generated randomly with X_i (i=1,...n) non-zero to form the vector B. The matrix A is generated by the method 1 as illustrated in Discussion

Table 6 **(Band Matrix)** **(Matlab)**

Algorithm	Time (Sec)	R-residual
Order = 5 **K(A) = 13.9282**		
SGE	0.0228	0
LSSP	0.0688	0
Order=10 **K(A) =48.3742**		
SGE	0.1066	0
LSSP	0.1326	0
Order =30 **K(A) =388.8121**		
SGE	3.7782	5.9212e-17
LSSP	1.2316	5.9212e-17
Order =90 **K(A) =3.3555e+03**		
SGE	81.5772	3.9475e-17
LSSP	24.1947	7.8949e-17
Order =270 **K(A) =2.9764e+04**		
SGE	975.3678	1.0527e-16
LSSP	162.9532	1.0527e-16
Order=520 **K(A)= 1.1001e+5**		
SGE	3.1701e+3	5.4657e-17
LSSP	746.9573	1.0931e-17

R-residual: Relative Residual, K(A): Condition number of matrix A
SGE: Standard Gaussian Elimination with Column Pivoting LSSP: The Left Strictly Symmetric Partition Method
The above results were obtained by applying the algorithms to the linear system AX=B with deferent orders, where vector $B=(B_1,...,B_n)$ was generated by setting $B_i=i$ for (i=1, ... n). The band matrix has the following form:

$$A = \begin{pmatrix} 2 & 1 & & & \\ 1 & 2 & 1 & & \\ & \ldots & & & \\ & & 1 & 2 & 1 \\ & & & 1 & 2 \end{pmatrix}$$

Reference

[1] I.S. Duff, A M Erisman, J K Reid
Direct Methods for Sparse Matrices
Oxford Science Publications (1986)

[2] Gene H. Golub, Charles F. Van Loan
Matrix Computation (Second Edition)
The Johns Hopkins University Press (1989)

[3] Sargent, RWH and Westerberg, AW
Speed-up in Chemical Engineering Design
Trans, Inst. Chem. Engrg. 42 (1964), 190-197

[4] Tarjan, RE
Depth-first Search and Linear Graph Algorithm
SIAM, Computing 1 (1972), 146-160

[5] Tarjan, RE
Efficiency of a Good But Not Linear Set Union Algorithm
J. ACM22 (1975), 215-225

[6] James R Bunch, Donald J. Rose
Sparse Matrix Computation
Academic Press Inc (1976)

[7] G.W. Stewart
Introduction to Matrix Computations
Academic Press INC (1973)

[8] T. J. Godin, Roger Haydock
The Block Recursion Library: Accurate Calculation of Resolvent Submatrices Using the Block Recursion Method
Computer Physics Communications 64 (1991), 123-130

[9] X. Yu, N.K. Loh, W.C. Miller
New Recursive Algorithm for Solving Linear Algebraic Equations
Electronics Letters (2nd Oct, 1992), Vol 28, No.22

[10] Wilkinson J.H.
Error Analysis of Direct Methods of Matrix Inversion
J. ACM 8 (1961), 281-330

[11] Reid J. K.
A Note on the Stability of Gaussian Elimination
J. Inst. Maths. Applcs 8 (1971), 374-365

[12] John R Rice
Matrix Computations and Mathematical Software
McGraw-Hill Book Company (1981)

[13] Jones - JCP, Billings-SA
Recursive Algorithm for Computing the Frequency-Response of a Class of Non-Linear Difference Equation Methods
International Journal of Control (1989), Vol 50. ISS 5, pp 1925-1940

[14] Mohideen-S, Cherkassky-V
On Recursive Calculation of the Generalized Inverse of A Matrix
ACM Transaction on Mathematical Software (1991), Vol 17, ISS 1, pp 130-147

Appendix

```
%
%       The Left Strictly Symmetric Partition (LSSP) Algorithm
%
%       A :     an n*(n+1) matrix, the last column is the RHS vector of the linear system
%       n :     the order of the linear equation system
%
%       Solution  to the system is stored in the last column of A
%
        function [A]=LSSP(A,n)
        m=20;
        if n==1, [A]=US1(1,A,1);return,end
        for ii=1:m,
                Ltop(ii)=0;
                Rtop(ii)=0;
        end
        for ii=1:n
                index(ii)=ii;
        end
%
%       Applying the Left Strictly Symmetrical Partition to matrix A
%
        [Ltop,Rtop]=LRtop(Ltop,Rtop,m,n);

        if Rtop(1)==0
                [A,ER,index]=SSP(1,n,m,A,n,index);
%
%       Re-arrange the order of solutions X=(x1,...,xn)',ie.
%       re-order the last column of matrix A,ie. A(:,n+1) according  to the order of index(i)
%
                for ii=1:n
                        TMP(index(ii))=A(ii,n+1);
                end
                A(1:n,n+1)=TMP(1:n)';
                return
        end
        kk=1; StartPNT=1;
        while Ltop(kk)~=0
                POINT=StartPNT;
                [A,ER,index]=SSP(POINT,Ltop(kk),m,A,n,index);
                if ER<0, return,end
                POINT=StartPNT;
                [A]=TRANSFER(POINT,Ltop(kk),Rtop(kk),A,n);
                StartPNT=0;
                for ii=1:kk
                        StartPNT=StartPNT+Ltop(ii);
                end
                StartPNT=StartPNT+1;
                POINT=StartPNT;
                kk=kk+1;
        end
        kk=kk-1;
        if Rtop(kk)==1
                [A]=US1(POINT,A,n);
```

```
        else
                [A,ER,index]=SSP(POINT,Rtop(kk),m,A,n,index);
        end
%
%       BACKWARD
%
        StartPNT=n+1;
        while kk~=0
                PT=StartPNT-(Ltop(kk)+Rtop(kk));
                [A]=BACKWARD(PT,Ltop(kk),Rtop(kk),A,n);
                kk=kk-1;
        end
%
%       Re-arrange the order of solutions X=(x1,...,xn)',ie.
%       re-order the last column of matrix A,ie. A(:,n+1) according  to the order of index(i)
%
        for ii=1:n
                TMP(index(ii))=A(ii,n+1);
        end
        A(1:n,n+1)=TMP(1:n)';
        return

%       ------------------------------------------------------------------------------------------
%
%       The  Strictly Symmetric Partition  (SSP)  Algorithm
%
%       POINT: the starting position in matrix A
%       Top:    the sub-order of the submatrix within matrix A
%       m:      equals to 20
%       A:      n*(n+1) matrix, its' last column is the RHS vector of  he inear system
%       n:      the order of  the linear system
%       ER:     error message:
%                       ER=-1, A(1,1)=0
%                       ER=-2, A(2,2)=0
%                       ER=0, no error occurs during computation.
%       index:  Work space for recording the column indices of  the solution
%
        function [A,ER,index]=SSP(POINT,TOP,m,A,n,index);
        Ctrl1=0;M=m;
        while Ctrl1==0;
                i=1;
                for j=1:M;
                        LR(j)=0;
                end
                Ctrl2=0;
                while Ctrl2==0;
                        [A,ER,index]=US(POINT,A,n,index);
                        if ER<0, return, end
                        POINT=POINT+2;
                        if TOP==2, return,end
                        if LR(i)==0;
                                PT=POINT-2;
                                [A]=TRANSFER(PT,2,2,A,n);
```

```
                    LR(i)=1;
                else
                    Ctrl2=1;
                end
            end
            Ctrl3=0;
            while Ctrl3==0;
                Ltorder=2^i; Rtorder=2^i;
                PT1=POINT-(Ltorder+Rtorder);
                [A]=BACKWARD(PT1,Ltorder,Rtorder,A,n);
                i=i+1;
                if TOP==2^i, return, end
                if LR(i)==0, Ctrl3=1;end
            end
            Ltorder=2^i; Rtorder=2^i;PT=POINT-2^i;
            [A]=TRANSFER(PT,Ltorder,Rtorder,A,n);
            LR(i)=1;
            M=i-1;
        end
        return
%       ------------------------------------------------------------------------------------------
%
%       LSSP decomposition to a  matrix  with order  n
%
        function [Ltop,Rtop]=LRtop(Ltop,Rtop,m,n)

        for i=1:m, Ltop(i)=0; Rtop(i)=0; end
        TOP=n; k=1;c=0;
        if n <= 2; Ltop(1)=n; return; end;
        while TOP>2
            while TOP ~=1; TOP=fix(TOP/2); c=c+1; end
            Ltop(k)=2^c;
            if k==1;
                Rtop(k)=n-Ltop(k);
            else
                Rtop(k)=Rtop(k-1)-Ltop(k);
                if Rtop(k)==0, Ltop(k)=0; end
            end
            TOP=Rtop(k);k=k+1;c=0;
        end
        return

%       ------------------------------------------------------------------------------------------
%       The Unit Solution (US)  (for solving 2*2  system with multiple RHS vectors)

        function [A,ER,index]=US(PT,A,n,index);
%
%       Column Pivoting for A(1,1), that is,  searching for k in {PT,...,n} such that
%       abs(A(PT,K))=max(abs(A(PT,i)) for i=PT, ... n,
%       and then  interchange columns Pt  and k in arrays A and index
%
```

```
ER=0;
TMP=A(PT,PT:n);
[Y,I]=max(abs(TMP));
I=I+PT-1;
TMP=A(1:n,I);
A(1:n,I)=A(1:n,PT);
A(1:n,PT)=TMP;
ind=index(PT)
index(PT)=index(I);
index(I)=ind;
if A(PT,PT)==0;
        ER=-1;
        error('A(1,1)=0, Matrix is Singular, Computation Stopped !');
        return;
end
A(PT,PT+1:n+1)=A(PT,PT+1:n+1)/A(PT,PT);
A(PT+1,PT+1:n+1)=A(PT+1,PT+1:n+1)-A(PT+1,PT)*A(PT,PT+1:n+1);
%
%       Column Pivoting for A(2,2), ithat is, searching for k in {PT+1,...,n} such that
%        abs(A(PT+1,K))=max(abs(A(PT+1,i)) for i=PT+1, ... n,
%       and then interchange colunm Pt+1 with column k in  arrays A  and index
%
TMP=A(PT+1,PT+1:n);
[Y,I]=max(abs(TMP));
I=I+PT;
TMP=A(1:n,I);
A(1:n,I)=A(1:n,PT+1);
A(1:n,PT+1)=TMP;
ind=index(PT+1);
index(PT+1)=index(I);
index(I)=ind;
if A(PT+1,PT+1)==0;
        ER=-2;
        error('A(2,2)=0, Matrix is Singular, Computation Stopped !');
        return;
end
A(PT+1,PT+2:n+1)=A(PT+1,PT+2:n+1)/A(PT+1,PT+1);
A(PT,PT+2:n+1)=A(PT,PT+2:n+1)-A(PT,PT+1)*A(PT+1,PT+2:n+1);
return
%       ---------------------------------------------------------------------------
%       The Special Unit Solution (US1)  for solving  a 1-by-1 system

function [A,ER]=US1(POINT,A,n)
PT=POINT;
ER=0;
 if A(PT,PT)==0;
ER=-1;
error('A(1,1)=0, ie. Matrix is Singular, Computation Stopped ! ');
return
end
A(PT,PT+1)=A(PT,PT+1)/A(PT,PT);
return
```

```
%     ------------------------------------------------------------
%     Algorithm of the TRANSFER

      function [A,Ltorder,Rtorder]=TRANSFER(PT,Ltorder,Rtorder,A,n)

      k1=PT+Ltorder;
      k2=PT+Ltorder+Rtorder-1;
      A(k1:k2,k1:k2)=A(k1:k2,k1:k2)-A(k1:k2,PT:(k1-1))*A(PT:(k1-1),k1:k2);
      A(k1:k2,k2+1:n+1)=A(k1:k2,k2+1:n+1)-A(k1:k2,PT:k1-1)*A(PT:k1-1,k2+1:n+1);
      return

%     ------------------------------------------------------------
%     Algorithm of the BACKWARD

      function [A]=BACKWARD(PT,Ltorder,Rtorder,A,n)

      k1=PT+Ltorder;
      k2=PT+Ltorder+Rtorder-1;
      A(PT:k1-1,k2+1:n+1)=A(PT:k1-1,k2+1:n+1)-A(PT:k1-1,k1:k2)*A(k1:k2,k2+1:n+1);
      return
```

```
%
%       The 1-n Partition Algorithm (1-nP)
%
%       A : an n*(n+1) matrix, the last column is the RHS vector of the system
%       n : the order of the linear system
%
%       Solution to the system is stored in the last column of A
%
        function [A]=S1n(A,n);

        if n<1, return, end
        k=1;
        while k<n

                PT=k;
                [A,ER]=US1n(PT,A,n);
                if ER<0, return,end

                Ltorder=1;Rtorder=n-k;
                [A]=TRANSFER(PT,Ltorder,Rtorder,A,n);
                k=k+1;
        end
        PT=n;
        [A,ER]=US1n(PT,A,n);
        if ER<0, return,end
        Rtorder=1;
        while Rtorder<n
                Ltorder=1;
                PT=n+1-(Ltorder+Rtorder);
                [A]=BACKWARD(PT,Ltorder,Rtorder,A,n);
                Rtorder=Rtorder+Ltorder;
        end
        return

%       ------------------------------------------------------------------

%       Block for solving a 1-by-1 system with row pivoting (US1)

        function [A,ER]=US1n(PT,A,n);
%
%       Row Pivoting for A(PT,PT) from row PT to n,
%       that is, searching for A(k,Pt) such that
%       asb(A(k,Pt))=max(abs(A(Pt,Pt)) for k=Pt, ... n, and then interchange row Pt with row k

        TMP=A(PT:n,PT);
        [Y,I]=max(abs(TMP));
        I=PT+I-1;
        TMP=A(I,1:n+1);
        A(I,1:n+1)=A(PT,1:n+1);
        A(PT,1:n+1)=TMP;
        if A(PT,PT)==0;
                ER=-1;
                error('A(1,1)=0, Matrix is Singular, Computation Stopped !');
```

```
        return;
    end
    A(PT,PT+1:n+1)=A(PT,PT+1:n+1)/A(PT,PT);
    return

%   ---------- TRANSFER ----------

    function [A,Ltorder,Rtorder]=TRANSFER(PT,Ltorder,Rtorder,A,n)

    k1=PT+Ltorder;
    k2=PT+Ltorder+Rtorder-1;
    A(k1:k2,k1:k2)=A(k1:k2,k1:k2)-A(k1:k2,PT:(k1-1))*A(PT:(k1-1),k1:k2);
    A(k1:k2,k2+1:n+1)=A(k1:k2,k2+1:n+1)-A(k1:k2,PT:k1-1)*A(PT:k1-1,k2+1:n+1);
    return

%   ---------- BACKWARD ----------

    function [A]=BACKWARD(PT,Ltorder,Rtorder,A,n)

    k1=PT+Ltorder;
    k2=PT+Ltorder+Rtorder-1;
    A(PT:k1-1,k2+1:n+1)=A(PT:k1-1,k2+1:n+1)-A(PT:k1-1,k1:k2)*A(k1:k2,k2+1:n+1);
    return
```

```
%
%       The 2-n Partition Algorithm (2-nP)
%
%       A : an n*(n+1) matrix, the last column is the RHS vector of the linear  system
%       n : the order of the linear system
%
%       Solution  to the system is stored in the last column of A
%
        function [A]=S2n(A,n);
        if n==1
                [A,ER]=US1(1,A,n);
                return;
        end
        if n==2
                [A,ER]=US2(1,A,n);
        end
        PT=1;orders=0;
        while (n-orders)>2

                PT=orders+1;orders=orders+2;
                [A,ER]=US2(PT,A,n);
                if ER<0, return,end
                Ltorder=2;Rtorder=n-orders;
                [A]=TRANSFER(PT,Ltorder,Rtorder,A,n);
        end

        PT=orders+1;
        if(n-orders)==1
                [A,ER]=US1(PT,A,n);
                Rtorder=1;
        else
                [A,ER]=US2(PT,A,n);
                Rtorder=2;
        end
        if ER<0, return,end
        while Rtorder<n
                Ltorder=2;
                PT=n+1-(Ltorder+Rtorder);
                [A]=BACKWARD(PT,Ltorder,Rtorder,A,n);
                Rtorder=Rtorder+Ltorder;
        end
        return

%       ---------- UNIT SOLUTION (US) ----------

%       Block for solving 2*2 quation systems

        function [A,ER]=US2(PT,A,n);
%
%       Row oriented Pivoting on A(1,1),i.e. searching for k in {PT,...,n}
%       such that abs(A(K,PT))=max(abs(A(i,PT)) for i=PT, ... n, and then
%       interchange colunm PT and column k  in arrays A and index
        ER=0;
```

```
TMP=A(PT:n,PT);
[Y,I]=max(abs(TMP));
I=PT+I-1;
TMP=A(I,1:n+1);
A(I,1:n+1)=A(PT,1:n+1);
A(PT,1:n+1)=TMP;

if A(PT,PT)==0;
        ER=-1;
        error('A(1,1)=0, Matrix is Singular, Computation Stopped !');
        return;
end

A(PT,PT+1:n+1)=A(PT,PT+1:n+1)/A(PT,PT);

TMP=0;
for i=1:n-PT
        TMP(i)=A(PT+i,PT+1)-A(PT+i,PT)*A(PT,PT+1);
end ;
%
%       Row oriented Pivoting on A(2,2),i.e. searching for k in {PT+1,...,n}
%       such that abs(A(K,PT+1))=max(abs(A(i,PT+1)) for i=PT+1, ... n, and
%       then interchange colunm PT+1 and  column k in arrays A and index
%
[Y,I]=max(abs(TMP));
I=PT+I;
TMP=A(I,1:n+1);
A(I,1:n+1)=A(PT+1,1:n+1);
A(PT+1,1:n+1)=TMP;
A(PT+1,PT+1:n+1)=A(PT+1,PT+1:n+1)-A(PT+1,PT)*A(PT,PT+1:n+1);
if A(PT+1,PT+1)==0;
        ER=-2;
        error('A(2,2)=0, Matrix is Singular, Computation Stopped !');
        return;
end
A(PT+1,PT+2:n+1)=A(PT+1,PT+2:n+1)/A(PT+1,PT+1);
A(PT,PT+2:n+1)=A(PT,PT+2:n+1)-A(PT,PT+1)*A(PT+1,PT+2:n+1);
return

%       ---------- US1 ----------

function [A,ER]=US1(POINT,A,n)

PT=POINT;
ER=0;
if A(PT,PT)==0;
ER=-1;
error('A(1,1)=0, ie. Matrix is Singular, Computation Stopped ! ');
return
end
A(PT,PT+1)=A(PT,PT+1)/A(PT,PT);
return
```

```
%          ---------- TRANSFER ----------

           function [A,Ltorder,Rtorder]=TRANSFER(PT,Ltorder,Rtorder,A,n)

           k1=PT+Ltorder;
           k2=PT+Ltorder+Rtorder-1;
           A(k1:k2,k1:k2)=A(k1:k2,k1:k2)-A(k1:k2,PT:(k1-1))*A(PT:(k1-1),k1:k2);
           A(k1:k2,k2+1:n+1)=A(k1:k2,k2+1:n+1)-A(k1:k2,PT:k1-1)*A(PT:k1-1,k2+1:n+1);
           return

%          ---------- BACKWARD ----------

           function [A]=BACKWARD(PT,Ltorder,Rtorder,A,n)

           k1=PT+Ltorder;
           k2=PT+Ltorder+Rtorder-1;
           A(PT:k1-1,k2+1:n+1)=A(PT:k1-1,k2+1:n+1)-A(PT:k1-1,k1:k2)*A(k1:k2,k2+1:n+1);
           return
```

```
C
C       The Left Strictly  Symmetic Partition (LSSP) Algorithm
C
C       A : an n*(n+1) matrix, the last column is the RHS vector of the linear  system
C       n : the order of the linear system
C
C       index: work space for reording the index  number of  the solution
C
C       Solution  to the system is stored in the last column of A
C
        subroutine LSSP(A,n,index)

        parameter(m=20)
        integer n,m,Top, StartPNT, POINT,Pt,Ltorder, Rtorder,ER
        integer Ltop(m), Rtop(m),index(n)
        Character*1 LR(m)
        Real A(n,n+1),TMP

        if (n.EQ.1) then
                call US1(1,A,n,ER)
                goto 500
        endif
        Do 100 i=1,m
                Ltop(i)=0
                Rtop(i)=0
100     continue
        Do 110 i=1,n
                index(i)=i
110     continue
C
C       Apply the Left Strictly Symmetrical Partition to matrix A
C
        call LRtop (Ltop,Rtop,m,n)

        IF (Rtop(1) .EQ. 0) THEN

C               Matrix A is a strictly symmetric linear system.
C               Then call the subroutine SSP algorithm for solving this system
C
                POINT=1
                Top=n
                call ssp (POINT,Top,m,A,n,ER,LR,index)
                goto 500
        END IF
        kk=1
        StartPNT=1
200     POINT=StartPNT
        Top=Ltop(kk)
C
C       Solution to the linear system with strictly symmetic partition.
C
        call ssp (POINT,Top,m,A,n,ER,LR,index)
        if (ER .LT. 0) goto 700
```

```
      POINT=StartPNT
      Ltorder=Ltop(kk)
      Rtorder=Rtop(kk)
      call TRANSFER(POINT,Ltorder,Rtorder,A,n)
      StartPNT=0
      Do 300 i=1,kk
            StartPNT=StartPNT+Ltop(i)
300   continue
      StartPNT=StartPNT+1
      POINT=StartPNT
      if (Ltop(kk+1) .NE. 0) then
            kk=kk+1
            goto 200
      endif

      if (Rtop(kk) .EQ.1 ) then
            call US1(POINT,A,n,ER)
      else
            Top=Rtop(kk)
            call ssp(POINT,Top,m,A,n,ER,LR,index)
      endif
      if (ER .LT.0 ) goto 700
C
C     BACKWARD
C
      StartPNT=n+1
400   Pt=StartPNT-(Ltop(kk)+Rtop(kk))
      Ltorder=Ltop(kk)
      Rtorder=Rtop(kk)
      call BACKWARD(Pt,Ltorder,Rtorder,A,n)
      kk=kk-1
      if (kk .NE. 0) goto 400
C
C     re-order the sequences of solutions according to the order of  index(i)
C
500   DO 600 i=1,n
            DO 610 k=1,n
                  if(index(k) .EQ. i) then
                        TMP=A(i,n+1)
                        A(i,n+1)=A(k,n+1)
                        A(k,n+1)=TMP
                        index(k)=index(i)
                        index(i)=i
                        goto 600
                  endif
610         continue
600   continue
700   END
```

```
C     ------------------------------------------------------------------------
C     The Strictly Symmetric Partition Algorithm (SSP)
C
C     POINT: the starting position in matrix A
C     Top:   the sub-order of the submatrix within matrix A
C     m:     equals to 20
C     A:     n*(n+1) matrix, its' last column is the RHS vector of the linear system
C     n:     the order of  the linear system
C     ER:    error message:
C                          ER=-1, A(1,1)=0
C                          ER=-2, A(2,2)=0
C                          ER=0, no error occurs during computation.
C     LR:    indictor of left and right submatrix in the binary tree
C     index: index of column of matrix A
C
C     Solution is stored in the last columns of matrix A
      subroutine ssp(POINT,Top,m,A,n,ER,LR,index)
      integer Pt,POINT,Top,ER,Ltorder,Rtorder,m,n
      character*1 LR(m)
      integer index(n)
      Real A(n,n+1)
      ER=0
      MM=m
100   i=1
      Do 200 j=1,MM
            LR(j)='L'
200   continue
300   call US(POINT,A,n,ER,index)
      IF  ((ER .LT. 0) .or. (Top .EQ. 2 )) goto 500
      POINT=POINT+2
      if (LR(i) .EQ. 'L') then
            Pt=POINT-2
            Ltorder=2
            Rtorder=2
            call TRANSFER(Pt,Ltorder,Rtorder,A,n)
            LR(i)='R'
            goto 300
      endif
400   Ltorder=2**i
      Rtorder=2**i
      Pt=POINT-(Ltorder+Rtorder)
      call BACKWARD (Pt,Ltorder,Rtorder,A,n)
      i=i+1
      if ((2**i) .EQ. Top)  goto 500
      if (LR(i) .NE. 'L') goto 400
            Ltorder=2**i
            Rtorder=2**i
            Pt=POINT-2**i
            Call TRANSFER (Pt, Ltorder, Rtorder,A,n)
            LR(i)='R'
            MM=i-1
            goto 100
500   END
```

```
C     ------------------------------------------------------------------------------
C     Left Strictly Symmetrical Partition ( Binary decompositions of n )
C
      subroutine LRtop (Ltop,Rtop,m,n)
      integer m,n,c,Top,Ltop(m),Rtop(m)
      Do 100 i=1,m
            Ltop(i)=0
            Rtop(i)=0
100   continue
      Top=n
      k=1
      c=0
      if (n.LE. 2) then
            Ltop(1)=n
            goto 400
      endif

      Do 300 while (Top .GT. 2)
            Do 200 while (Top. NE. 1)
                  Top=int(Top/2)
                  c=c+1
200         continue
            Ltop(k)=2**c
            if (k.EQ.1) then
                  Rtop(k)=n-Ltop(k)
            else
                  Rtop(k)=Rtop(k-1)-Ltop(k)
                  if (Rtop(k) .EQ. 0) then
                  Ltop(k)=0
                  endif
            endif
            Top=Rtop(k)
            k=k+1
            c=0
300   continue

400   END
C     ------------------------------------------------
C     The Unit Solution (US)
C
      subroutine  US(Pt,A,n,ER,index)

      integer Pt,n, ER,ind,TMP
      integer index(n)
      Real A(n,n+1),max
      ER=0
C
C     Column Pivoting for A(1,1), i.e. searching for A(PT,S) such that
C     abs(A(PT,S))=max(abs(A(Pt,Pt)) for k=Pt, ... n, and then interchange
C     colunm Pt and  column S in A and index
C
      max=A(Pt,Pt)
      ind=Pt
```

```
      DO 100 j=Pt,n
            if(abs(max) .LT. abs(A(Pt,j))) then
                  max=A(Pt,j)
                  ind=j
            endif
100   continue
      TMP=index(ind)
      index(ind)=Pt
      index(Pt)=TMP
      Call Colchange (A,Pt,ind,n)
      if (A(Pt,Pt) .EQ. 0 ) then
            ER=-1
            write(*,*)'A(1,1)=0, Matrix is Singular, Computation Stopped !'
            goto 500
      endif
      DO 200 k=Pt+1,n+1
            A(Pt,k)=A(Pt,k)/A(Pt,Pt)
            A(Pt+1,k)=A(Pt+1,k)-A(Pt+1,Pt)*A(Pt,k)
200   continue
C
C     Column pivoting for A(2,2), that is, searching for k in {Pt+1,...n}
C     such that abs(A(Pt+1,k)=max(abs(A(Pt+1,i)) for i=Pt+1,...,n, and
C     then interchange columns Pt+1 and k
C
      max=A(Pt+1,Pt+1)
      ind=Pt+1
      DO 300 j=Pt+1,n
            if(abs(max) .LT. abs(A(Pt+1,j))) then
                  max=A(Pt+1,j)
                  ind=j
            endif
300   continue
      TMP=index(ind)
      index(ind)=Pt+1
      index(Pt+1)=TMP
      Call Colchange (A,Pt+1,ind,n)
      if (A(Pt+1,Pt+1) .EQ. 0 ) then
            ER=-2
            write(*,*)'A(2,2)=0, Matrix is Singular, Computation Stopped !'
            goto 500
      endif

      Do 400 k=Pt+2, n+1
            A(Pt+1,k)=A(Pt+1,k)/A(Pt+1,Pt+1)
            A(Pt,k)=A(Pt,k)-A(Pt,Pt+1)*A(Pt+1,k)
400   continue
500   END
```

```
C       ------------------------------------------------------------------------
C       Column Interchange  (Colchange)
C
C       interchange column i  column j in matrix A of order n
C
        Subroutine Colchange(A,i,j,n)

        real A(n,n+1), exc
        DO 100 k=1,n
                exc=A(k,i)
                A(k,i)=A(k,j)
                A(k,j)=exc
100     continue
        END
C
C       ------------------------------------------------------------------------
C       Special Unit Solution to  a 1-by-1 Linear  System (US1)
C
        subroutine  US1(Pt,A,n,ER)
        integer Pt,n,ER
        real A(n,n+1)
        ER=0
        if (A(Pt,Pt) .EQ. 0) then
                ER=-1
                write(*,*)'A(1,1)=0,  Computation Stopped !!'
                goto 100
        endif
        A(Pt,Pt+1)=A(Pt,Pt+1)/A(Pt,Pt)
100     END
C
C       ------------------------------------------------------------------------
C       BACKWARD
C
        subroutine BACKWARD(Pt, Ltorder, Rtorder, A, n)

        integer Pt, Ltorder,Rtorder,n
        real A(n,n+1)
        real TMP
        k1=Pt+Ltorder
        k2=Pt+Ltorder+Rtorder
        Do 300 i=Pt, k1-1
                Do 200 j=k2,n+1
                        TMP=0
                        Do 100 k=k1,k2-1
                                TMP=TMP+A(i,k)*A(k,j)
100                     continue
                        A(i,j)=A(i,j)-TMP
200             continue
300     continue
        END
```

```
C
C     ------------------------------------------------------------------------
C     TRANSFER
C
      subroutine  TRANSFER(Pt,Ltorder,Rtorder,A,n)

      integer Pt,Ltorder,Rtorder,n
      Real A(n,n+1)
      real TMP
      k1=Pt+Ltorder
      k2=Pt+Ltorder+Rtorder-1
      DO 300 i=k1,k2
            Do 200 j=k1,k2
                  TMP=0
                  Do 100 k=Pt,k1-1
                        TMP=TMP+A(i,k)*A(k,j)
100               continue
                  A(i,j)=A(i,j)-TMP
200         continue
300   continue
      DO 600 i=k1,k2
            Do 500 j=k2+1,n+1
                  TMP=0
                  Do 400 k=Pt,k1-1
                        TMP=TMP+A(i,k)*A(k,j)
400               continue
                  A(i,j)=A(i,j)-TMP
500         continue
600   continue
      END
```

```
C
C       The 1-n Partition  (1-nP) Algorithm
C
C       A : an n*(n+1) matrix, the last column is the RHS vector of the system
C       n : the order of the linear system
C
C       Solution  to the system is stored in the last column of A
C
        Subroutine S1n(A,n)

        integer n,PT,Ltorder,Rtorder,ER
        real A(n,n+1)

        if (n.Lt.1) goto 300

        k=1
        DO 100 while (k .LT. n)
C
C       1-n Unit Solution with row pivoting.
C
                PT=k
                call US1n(PT,A,n,ER)
                if (ER .LT. 0) goto 300
C
C       Transfer from 1-order to (n-k) order
C
                Ltorder=1
                Rtorder=n-k
                call TRANSFER (PT,Ltorder,Rtorder,A,n)
                k=k+1
100     continue
C
C       Solve the last 1*1 matrix
C
        PT=n
        call US1n(PT,A,n,ER)
        if (ER .LT. 0) goto 300
C
C       Backward
C
        DO 200 while (Rtorder .LT. n)

                Ltorder=1
                PT=n+1-(Ltorder+Rtorder)
                call BACKWARD(PT,Ltorder,Rtorder,A,n)
                Rtorder=Rtorder+Ltorder

200     continue
300     END
```

```
C     ------------------------------------------------------------------------------
C     Subroutine for solving a 1-by-1 system with row pivoting
C
      subroutine  US1n(Pt,A,n,ER)
      integer Pt,n, ER,ind
      Real A(n,n+1),max
      ER=0
C
C     Row Pivoting for A(1,1), i.e. searching for A(k,Pt) such that
C     asb(A(k,Pt))>abs(A(Pt,Pt) for k=Pt, ... n, and then interchange
C     row Pt with row k
C
      max=A(Pt,Pt)
      ind=Pt
      DO 100 i=Pt,n
            if(abs(max) .LT. abs(A(i,Pt))) then
                  max=A(i,Pt)
                  ind=i
            endif
100   continue
C
C     interchange row Pt with row ind
C
      Call Rowchange (A,Pt,ind,n)

      if (A(Pt,Pt) .EQ. 0 ) then
            ER=-1
            write(*,*)'A(1,1)=0, Matrix is Singular, Computation Stopped !'
            goto 300
      endif
      DO 200 k=Pt+1,n+1
            A(Pt,k)=A(Pt,k)/A(Pt,Pt)
200   continue
300   END
C
C     ------------------------------------------------------------------------------
C     Subroutine for interchanging row i with row j
C
      Subroutine  Rowchange (A,i,j,n)

      real TMP
      real A(n,n+1)
      DO 100 K=1,n+1

            TMP=A(i,k)
            A(i,k)=A(j,k)
            A(j,k)=TMP
100   Continue
      END
```

```
C     ------------------------------------------------------------

      subroutine  TRANSFER(Pt,Ltorder,Rtorder,A,n)
      integer Pt,Ltorder,Rtorder,n

      Real A(n,n+1)
      real TMP
      k1=Pt+Ltorder
      k2=Pt+Ltorder+Rtorder-1

      DO 300 i=k1,k2
            Do 200 j=k1,k2
                  TMP=0
                  Do 100 k=Pt,k1-1
                        TMP=TMP+A(i,k)*A(k,j)
100               continue
                  A(i,j)=A(i,j)-TMP
200         continue
300   continue
      DO 600 i=k1,k2
            Do 500 j=k2+1,n+1
                  TMP=0
                  Do 400 k=Pt,k1-1
                        TMP=TMP+A(i,k)*A(k,j)
400               continue
                  A(i,j)=A(i,j)-TMP
500         continue
600   continue
      END
C     ------------------------------------------------------------------------------

      subroutine BACKWARD(Pt, Ltorder, Rtorder, A, n)

      integer Pt, Ltorder,Rtorder,n
      real A(n,n+1)
      real TMP
      k1=Pt+Ltorder
      k2=Pt+Ltorder+Rtorder
      Do 300 i=Pt, k1-1
            Do 200 j=k2,n+1
                  TMP=0
                  Do 100 k=k1,k2-1
                        TMP=TMP+A(i,k)*A(k,j)
100               continue
                  A(i,j)=A(i,j)-TMP
200         continue
300   continue
      END
```

```
C
C       The 2-n Partition (2-nP) Algorithm
C
C       A : an n*(n+1) matrix, the last column is the RHS vector of the linear system
C       n : the order of the linear system
C
C       Solution  to the system is stored in the last column of A

        Subroutine S2n(A,n)

        integer n,PT,orders,Ltorder,Rtorder,ER
        real A(n,n+1)
        if (n.LE.2) then
                if (n.EQ.2) then
                        PT=1
                        call US1(PT,A,n,ER)
                else
                        PT=2
                        call US2(PT,A,n,ER)
                endif
                goto 400
        endif
        PT=1
        orders=0
        DO 100 while (n-orders .GT. 2)
                PT=orders+1
                orders=orders+2
                call US2(PT,A,n,ER)
                if (ER .LT. 0) goto 400
                Ltorder=2
                Rtorder=n-orders
                call TRANSFER (PT,Ltorder,Rtorder,A,n)
100     continue
        PT=orders+1
        if (n-orders .EQ. 1) then
                call US1 (PT,A,n,ER)
                Rtorder=1
                goto 200
        else
                call US2(PT,A,n,ER)
                Rtorder=2
        endif
200     if (ER .LT. 0) goto 400
        DO 300 while (Rtorder .LT. n)
                Ltorder=2
                PT=n+1-(Ltorder+Rtorder)
                call BACKWARD(PT,Ltorder,Rtorder,A,n)
                Rtorder=Rtorder+Ltorder
300     continue
400     END
```

```
C     ------------------------------------------------------------------

      subroutine  US2(Pt,A,n,ER)
      integer Pt,n, ER,ind,Row
      Real A(n,n+1),EX,max
      ER=0
C
C     Row Pivoting for A(1,1), i.e. searching for A(k,Pt) such that
C     asb(A(k,Pt))>abs(A(Pt,Pt) for k=Pt, ... n, and then interchange
C     row Pt with row k
C
      max=A(Pt,Pt)
      ind=Pt
      DO 100 i=Pt+1,n
            if(abs(max) .LT. abs(A(i,Pt))) then
                  max=A(i,Pt)
                  ind=i
            endif
100   continue

      Call Rowchange (A,Pt,ind,n)
      if (A(Pt,Pt) .EQ. 0 ) then
            ER=-1
            write(*,*)'A(1,1)=0, Matrix is Singular, Computation Stopped !'
            goto 500
      endif
      DO 200 k=Pt+1,n+1
            A(Pt,k)=A(Pt,k)/A(Pt,Pt)
200   continue
C
C     Row Pivoting for A(2,2), i.e. searching for A(k,Pt+1) such that
C     A(Pt+1,Pt+1)-A(Pt+1,Pt)*A(Pt,Pt+1) is not equal to 0, and then
C     interchange the row k with row Pt+1
C
C     where k=Pt+1, ..., n
C
      EX=A(Pt+1,Pt+1)-A(Pt+1,Pt)*A(Pt,Pt+1)
      Row=Pt+1
      DO 300 while (EX.EQ. 0)
            if (Row .GT. n) then
            ER=-2
            Write(*,*)'A(2,2)=0, Matrix is Singular,Computation Stopped !'
            goto 500
            endif
            Row=Row+1
            EX=A(Row,Pt+1)-A(Row,Pt)*A(Pt,Pt+1)
300   continue
      Call Rowchange(A,Pt+1,Row,n)
      A(Pt+1,Pt+1)=EX
      Do 400 k=Pt+2, n+1
            A(Pt+1,k)=(A(Pt+1,k)-A(Pt+1,Pt)*A(Pt,k))/A(Pt+1,Pt+1)
            A(Pt,k)=A(Pt,k)-A(Pt,Pt+1)*A(Pt+1,k)
400   continue
500   END
```

```
C
C     ------------------------------------------------------------------------------
C
      Subroutine  Rowchange (A,i,j,n)
C
C     interchange row i with row j
C
      real TMP
      real A(n,n+1)
      DO 100 K=1,n+1
            TMP=A(i,k)
            A(i,k)=A(j,k)
            A(j,k)=TMP
100   Continue
      END
C     ------------------------------------------------------------------------------
C     solving for a 1-by-1 system

      subroutine  US1(Pt,A,n,ER)

      integer Pt,n,ER
      real A(n,n+1)
      ER=0
      if (A(Pt,Pt) .EQ. 0) then
            ER=-1
            write(*,*)'A(1,1)=0,  Computation Stopped !!'
            goto 100
      endif

      A(Pt,Pt+1)=A(Pt,Pt+1)/A(Pt,Pt)

100   END
c     ------------------------------------------------------------------------------

      subroutine  TRANSFER(Pt,Ltorder,Rtorder,A,n)

      integer Pt,Ltorder,Rtorder,n
      Real A(n,n+1)
      real TMP
      k1=Pt+Ltorder
      k2=Pt+Ltorder+Rtorder-1
      DO 300 i=k1,k2
            Do 200 j=k1,k2
                  TMP=0
                  Do 100 k=Pt,k1-1
                        TMP=TMP+A(i,k)*A(k,j)
100                 continue
                  A(i,j)=A(i,j)-TMP
200         continue
300   continue
      DO 600 i=k1,k2
            Do 500 j=k2+1,n+1
                  TMP=0
                  Do 400 k=Pt,k1-1
```

```
                        TMP=TMP+A(i,k)*A(k,j)
400                 continue
                    A(i,j)=A(i,j)-TMP
500             continue
600     continue
        END
c       ------------------------------------------------------------------------------

        subroutine BACKWARD(Pt, Ltorder, Rtorder, A, n)

        integer Pt, Ltorder,Rtorder,n
        real A(n,n+1)
        real TMP
        k1=Pt+Ltorder
        k2=Pt+Ltorder+Rtorder
        Do 300 i=Pt, k1-1
                Do 200 j=k2,n+1
                        TMP=0
                        Do 100 k=k1,k2-1
                                TMP=TMP+A(i,k)*A(k,j)
100                     continue
                        A(i,j)=A(i,j)-TMP
200             continue
300     continue
        END
```