UNIVERSITY OF NEWCASTLE-UPON-TYNE
CIVIL ENGINEERING DEPARTMENT

COMPUTER MODELLING OF WATER SUPPLY DISTRIBUTION NETWORKS USING THE GRADIENT METHOD.

by

Rubén O. Salgado-Castro

VOLUME TWO
****

APPENDICES

Thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in Civil Engineering.

November, 1988.

TABLE OF CONTENTS

VOLUME TWO
****

APPENDIX A

EFFICIENT SOLUTION OF LINEAR SYSTEMS OF EQUATIONS IN THE CONTEXT OF THE GRADIENT METHOD FOR THE ANALYSIS OF WATER SUPPLY DISTRIBUTION NETWORKS

APPENDIX B

DERIVATION OF THE KRIGING ESTIMATOR EQUATIONS

APPENDIX C

DERIVATION OF THE BI-CUBIC SPLINES APPROXIMATION EQUATIONS

APPENDIX D

LIST OF FIGURES

LIST OF TABLES

LIST OF MAIN SYMBOLS

APPENDIX A

Symbol	Description
A, B	A capital letter denotes a matrix, particularly:
L	Denotes a lower triangular matrix.
U	Denotes an upper triangular matrix.
D	Denotes a diagonal matrix.
a, $\underline{A}$, $\underline{a}$	A lower case letter, or a capital (or lower case) underlined letter denotes a vector.
α, μ	A greek letter denotes a scalar.
a_i	A lower case subindexed letter denotes a scalar, normally the i-th component of a vector.
a_{ij}	A lower case doubly subindexed letter denotes a scalar, normally the coefficient of a matrix A, located at the intersection of row "i" and column "j".
A_i	A capital subindexed letter denotes either a matrix in a sequence of matrices or a block-partitioned matrix.
$(.)^T$	Denotes transposition, either of a vector or matrix.
A^{-1}	Denotes matrix inversion.
$(.)^{(k)}$	Refers to the k-th value of the variable (scalar, vector or matrix) within an iterative procedure.
$\|\|x\|\|$	Denotes the norm of a variable (vector or matrix).
$\|x\|$	Denotes the absolute value of a scalar.
λ_i	Denotes a particular eigenvalue of a matrix.
$\rho(A)$	Spectral radius of?matrix A.
$\langle x,y \rangle$	Inner (scalar) product of vectors "x" and "y": $\langle x,y \rangle = x^T y = y^T x = \sum_{i=1}^{n} x_i y_i$

APPENDIX B

Symbol	Description
$(\underline{x}, H)$	State variable representing the piezometric head at any point $\underline{x}$ of the domain: $(\underline{x},H) = (x_1, x_2, H)^T$ in a two-dimensional space.
$Z(\underline{x}, H)$	Random function representing the value of the state variable H at a location $\underline{x}$ (in general).
$Z(\underline{x}_o, H)$	Random variable (at the particular location $\underline{x}=\underline{x}_o$. Simplified notation: $Z(\underline{x}_i, H) <=> Z_{\underline{x}_i} <=> Z_i$
$E[Z(\underline{x}_1, H)]$	expected value of the random variable at $\underline{x}=\underline{x}_1$.
$\Gamma(\underline{x}_1, \underline{x}_2)$	Semi-variogram: $\Gamma(\underline{x}_1, \underline{x}_2) \equiv \frac{1}{2} E[\{Z(\underline{x}_1,H)Z(\underline{x}_2,H)\}^2]$
$Cov(Z(\underline{x}_1,H),Z(\underline{x}_2,H))$	Covariance: $Cov(Z(\underline{x}_1,H),Z(\underline{x}_2,H)) = E[Z(\underline{x}_1,H)Z(\underline{x}_2,H)]$
$Var(Z(\underline{x},H))$	Variance of the random variable: $Var(Z(\underline{x},H)) = E[Z(\underline{x}, H)^2] - \{E[Z(\underline{x}, H)]\}^2 = \sigma^2$
λ_o^i	Unknown weight of the Kriging estimator, where"i" refers to the measurement Y_i and "o" to the point being estimated $\underline{x}_o$, i.e.: $Y_o^* = Y_o^*(\underline{x}_o) = \sum_{i=1}^{n} \lambda_o^i Y_i$

APPENDIX C

Symbol	Description
One dimensional splines:	
$s(x)$	General spline polynomial function.
λ_i	Set of knots associated to the spline.
h	Number of knots (λ_i , $i=1,2,..,h$)
$(x_r, f(x_r))$, $r=1,2,..,m$	set of data points.
$M_i(x)$	Cubic Basic (or fundamental) spline (B-spline).
$M_{n,i}(x)$	General B-spline of degree (n-1). For a cubic spline $n=4$, i.e.: $M_{4,i}(x) = M_i(x)$ (simplified notation).
τ_i	Linear (unknown) weights, combining the cubic B-splines into a general cubic spline polynomial: $$s(x) = \sum_{i=1}^{h+4} \tau_i M_i(x)$$
$\underline{\tau}$	(h+4)*1) column vector of linear weights τ_i.
A	Matrix of observation equations (observation matrix), of order m*(h+4) in a cubic spline: $A_{r,i} = M_i(x)$.
$A\,\underline{\tau} = \underline{f}$	Observation equations, overdetermined system.
$\underline{f}$	(m*1) column vector with the value of the spline polynomial at each data point: $f_i = f(x_i) = s(x_i) \quad i=1,2,\ldots,m$
$A^T A\underline{\tau}=A^T\underline{f}$	Normal equations. Linear system of (h+4)*(h+4) equations (determined system).
Bi-cubic splines:	
$s(x,y)$	General spline polynomial function.
λ_i	Set of knots associated to the independent variable x.
μ_j	Set of knots associated to the dependent variable y.

Symbol	Description
h	Number of knots in the x-axis (λ_i , i=1,2,..,h)
k	Number of knots in the y-axis (μ_j , j=1,2,..,k)
R_{ij}	Set of (h+1)*(k+1) panels in which the x-y space is sub-divided.
$(x_r, y_r, f(x_r, y_r))$, r=1,2,..,m	set of data points.
$M_i(x)$	Cubic B-spline related to the independent variable x.
$N_j(y)$	Cubic B-spline related to the dependent variable y.
Γ_{ij}	Linear (unknown) weights, combining the cubic B-splines into a general cubic spline polynomial: $$s(x,y) = \sum_{i=1}^{h+4} \sum_{j=1}^{k+4} \Gamma_{ij} M_i(x) N_j(y)$$
$\underline{\Gamma}$	((h+4)(k+4)*1) column vector of linear weights Γ_{ij}.
A	Matrix of observation equations (observation matrix), of order m*(h+4)(k+4).
$A\,\underline{\Gamma} = \underline{f}$	Observation equations, overdetermined system.
$\underline{f}$	(m*1) column vector with the value of the spline polynomial at each data point: $$f_i = f(x_i) = s(x_i) \quad i=1,2,\ldots,m$$
$A^T A\underline{\Gamma} = A^T \underline{f}$	Normal equations. Linear system of (h+4)(k+4)*(h+4)(k+4) equations (determined system).

APPENDIX A

EFFICIENT SOLUTION OF LINEAR SYSTEMS OF EQUATIONS IN THE CONTEXT OF THE GRADIENT METHOD FOR THE ANALYSIS OF WATER SUPPLY DISTRIBUTION NETWORKS

A.1. Introduction.

A.1.1. Need for efficient linear solvers in water distribution network analysis.

From a mathematical viewpoint, we have already seen in Chapter Two that the water distribution network analysis problem reduces to the solution of a set of non-linear simultaneous equations; different ways of assembling these equations lead to different network analysis algorithms.

Because a direct solution for simultaneous non-linear systems does not exist, the usual approach is to transform the problem into a sequence of linear problems, whose solution approximates step by step the solution of the original non-linear set.

As a result, for solving a non-linear system of equations we have to solve a number of linear systems and consequently , very efficient linear solvers are required in order to keep the computational resources needed under a reasonable limit.

Thus, network analysis algorithms rely heavily on the solution of linear systems of equations and, consequently, efficient linear solvers which can take advantage of the particular

features of the linear systems generated in network analysis are needed. In the context of the gradient network analysis method, we shall be especially interested in linear solvers for symmetric positive definite systems.

This Appendix presents a review of the methods currently available for the solution of linear systems of equations.

Because most of the real water distribution networks generate linear systems in the range of hundreds of unknowns, and since the matrices produced are sparse (i.e. with very few non-zeros per row), special consideration will be paid to those methods allowing the efficient storage and handling of matrices of this kind.

A.1.2. Different classes of linear problems.

In its most general format, the solution of a linear system can be stated as that of finding the "n" unknown values, x_i's, in the following system of "m" linear equations:

$$\begin{aligned} a_{11}\,x_1 + a_{12}\,x_2 + a_{13}\,x_3 + \ldots + a_{1n}\,x_n &= b_1 \\ a_{21}\,x_1 + a_{22}\,x_2 + a_{23}\,x_3 + \ldots + a_{2n}\,x_n &= b_2 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots &\quad \vdots \\ a_{m1}\,x_1 + a_{m2}\,x_2 + a_{m3}\,x_3 + \ldots + a_{mn}\,x_n &= b_m \end{aligned} \qquad (1)$$

where the coefficients a_{ij} and the right hand side terms b_i are all known. These coefficients normally emerge from some physical property of the system under study.

In order to simplify the notation we shall use, whenever possible, capital letters to denote matrices, lower case letters

to represent vectors and greek letters to denote scalar numbers. In some instances, when the use of lower and upper case letters to differentiate vectors from matrices is not possible (or when it is not convenient), for the sake of clarity we shall denote vectors by an underlined letter (e.g.: $\underline{P}$, $\underline{Q}$, $\underline{H}$, $\underline{h}$, $\underline{q}$ are all vectors). Also notice that, in general, sub-indexed lower case letters will represent scalars. Thus, the linear system (1) can be expressed in matrix notation as:

$$A\,x = b \qquad (2)$$

where:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

A : is the <u>matrix of coefficients</u> and it has "m" rows and "n" columns, i.e. it is an (m x n) matrix.

$x = (x_1, x_2, x_3, \ldots, x_n)^T$

x : is the <u>vector of unknowns</u>, a column vector in the space R^n; this is a (n x 1) vector.

and

$b = (b_1, b_2, b_3, \ldots, b_m)^T$

b : is the <u>right hand side vector</u>, a column vector in the space R^m; this is a (m x 1) vector.

Depending on the relative values of "m" and "n", we may face different kinds of problems:

i) <u>Underdetermined systems of equations</u>:

In this particular case $m < n$. Clearly, in this case we have less equations than unknowns and the system has, in general, more

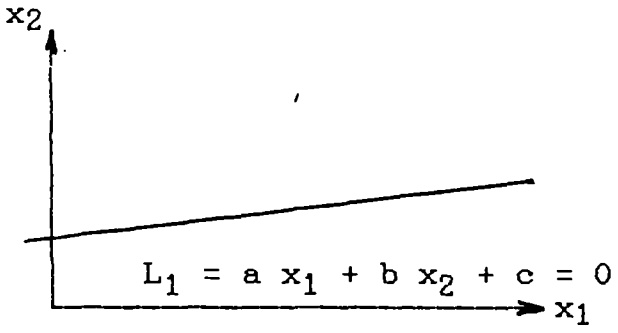
than one "exact" solution, by which we mean that the equality in (1) or (2) holds.

A very simple example of an underdetermined system can be shown in a 2-dimensional plane, as in Fig. A.1.

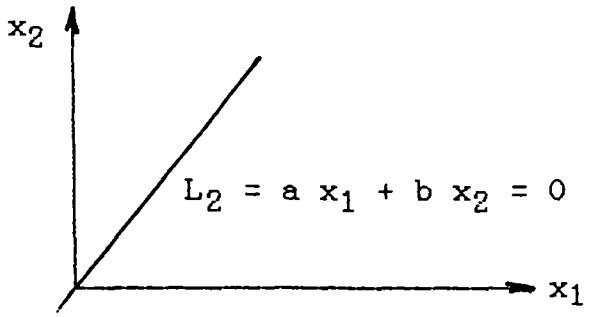
The straight line L_1 in Fig. A.1 a) represents a "system" of just one equation in the unknowns x_1 and x_2. The solution points of this system are simply all the infinite points <u>on</u> L_1.

In underdetermined systems it is possible to find two cases:

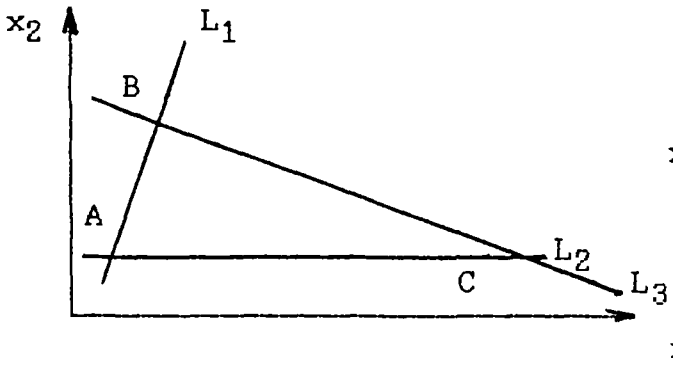
a) Homogeneous case: where the right hand side (RHS) vector b is null (i. e. $b = \underline{0}$, or $b_i = 0$ $i=1,2, \ldots, n$).



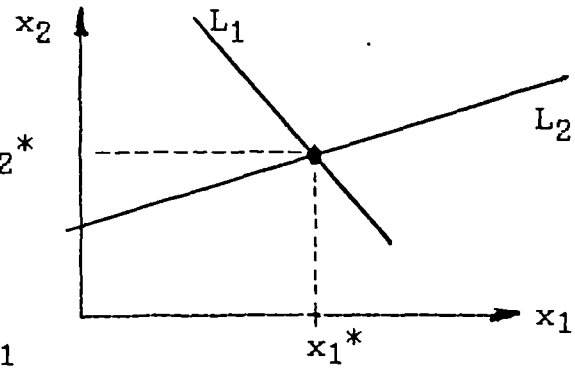
a)Non-homogeneous underdetermined system of equations.



b)Homogeneous underdetermined system of equations.



c) Overdetermined system of linear equations.



d) Determined system of linear equations.

Fig. A.1. Different linear systems of equations.

b) Non-homogeneous case: where some $b_i \neq 0$ (at least one of them must be non-zero).

In the previous example L_1 represents a non-homogeneous problem if and only if $c \neq 0$. If $c = 0$, we are in the homogeneous case and the straight line becomes $L_2 = a\ x_1 + b\ x_2 = 0$, where L_2 passes now through the origin, as shown in Fig. A.1. b)

In practice, it is not often that we need to solve an underdetermined system of equations, but one of these cases arises when, for example, we are trying to find an initial flow distribution in a pipe network, where we need to guarantee the fulfilment of the node balance; in this case, and using the notation defined in Chapter Two (Section 2.3.7), we need to solve the system:

$$A_{21}\ Q = q \qquad (3)$$

where:

A_{21} = A_{12}^T ; with A_{12} the topological (connectivity) matrix. A_{12} is of order (NP x (NN-NS)), where NN is the number of nodes, NS is the number of sources and NP is the number of branches (pipes, valves, etc.). Then, A_{21} is an ((NN-NS) x NP) matrix.

Q : (NP x 1) vector of initial flows (unknown).

and

q : ((NN-NS) x 1) vector of nodal consumptions, which are given as data. This vector can be zero or non-zero, leading either to homogeneous or non-homogeneous problems.

We shall not study in more detail the solution of this kind of problem. O'Neil (1983) and George, Heath and Ng (1984), give a more detailed treatment of this problem.

ii) Overdetermined systems of equations.

In this particular case we have $m > n$, i.e. we have more equations than unknowns. No exact solution is possible and the equality in (1) and (2) does not hold.

In the context of our small previous example, this can be represented in a 2-dimensional space by a set of 3 straight lines L1, L2 and L3, where it is clear that no single point can satisfy simultaneously the 3 equations: point A represents the simultaneous fulfilment of L1 and L2, B that of L1 and L3 and C the corresponding to L2 and L3. See Fig. A.1. c)

The best we can do in an overdetermined problem is to find a solution that minimizes the difference between the RHS vector and the product A x. In other words we can minimize the residual vector:

$$r = A\,x - b \qquad (4)$$

A well-known technique for finding a solution to this problem is the "least square" algorithm, where the unknown "x" is chosen so that the square of the euclidean norm of "r" is minimized:

$$\min \| r \|_2^2 = r^T r = \sum_{i=1}^{m} r_i^2 = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij}\, x_j - b_j \right)^2 \qquad (5)$$

There is a wide variety of numerical methods available to solve this problem, but we shall not include them here; for details see for example Duff and Reid (1976), Golub and van Loan (1983).

In practice, as before, it is not common to have to solve overdetermined systems in civil engineering problems. Nevertheless, one example arises in estimation problems, when using weighted least squares estimators with less parameters to be estimated than the number of measurements available. With this same kind of problem we can get an underdetermined system if the number of measurements is less than the number of parameters to be estimated and also, we can get a determined system when both the number of measurements and parameters to be estimated coincide. In either underdetermined or overdetermined systems, because the matrices are not square matrices, their inverse does not exist and the concept of "pseudo-inverse" or "generalised inverse" is used instead; see Gelb (1974), Rao (1962) and Penrose (1955) for details.

In the context of an explicit calibration algorithm for water distribution networks [see Chapter Six], we have used a bi-cubic splines approach to approximate the piezometric head of unmeasured nodes, based on some measured heads. The estimation of the coefficients of the spline polynomial leads to an overdetermined system of equations, which is solved via the least-square technique (normal equations); see Appendix C for details on the cubic splines fitting.

iii) Determined systems of equations.

In this particular case m = n, then we have as many equations as unknowns. The matrix A is said to be square of order "n" (or "m").

In the context of our small example in a 2-dimensional space, the determined case corresponds to the problem of finding the intersection of two straight lines L1 and L2: i.e. the coordinates of point A: (x_1^*, x_2^*), as shown in Fig. A.1. d)

The condition for the existence of a solution is that A must be invertible, this means that, from the mathematical point of view, one of the following statements must be true:

a) The determinant of A is non-zero (i.e. A is non-singular).
b) The rank of the matrix is equal to the order of the matrix (the rank is defined as the order of the largest submatrix of A with a non-vanishing determinant).

A simple representation of a non-invertible A matrix, arises when the straight lines of the previous figure are parallel. Numerical difficulties are found when both lines are "almost" parallel.

Most of the engineering problems involve the solution of determined systems of linear equations, and different methods of solution have been proposed, depending on the particular properties of the matrix A (symmetry, sparseness, positive or negative definiteness, etc.) and the computational resources available.

Since real engineering problems lead generally to positive definite systems, we shall concentrate our attention on methods for solving such problems, although different methods are available to solve systems of equations with indefinite matrices

[see Duff, (1980)].

The model problem for a determined linear system of equations is:

$$\text{Solve :} \qquad A\,x = b \qquad (6)$$

where:

A : is a known non-singular square matrix of order "n",

b : a known (n x 1) column vector.

We need to find the vector $x = (x_1, x_2, \ldots, x_n)^T$, which satisfies equation (6).

The structure and numerical values of the matrix A and vector b are usually obtained from a mathematical model of the physical problem being studied.

From a strictly algebraic point of view, the solution of the linear system represented by equation (6) can be easily obtained if we know the inverse matrix of A, say A^{-1}, (A^{-1} is such that $A\ A^{-1} = A^{-1}\ A = I$: the identity matrix); thus, premultiplying equation (6) by A^{-1}, we get :

$$A^{-1}\ A\ x = A^{-1}\ b$$

or

$$x = A^{-1}\ b \qquad (7)$$

Although from the algebraic viewpoint the solution looks quite simple, the process of inverting a matrix (especially a large one, as is usual in many practical problems) is a very expensive process, demanding a significant amount of work (n^3 mathematical operations) and storage (n^2 storage positions, since the inverse matrix can be completely filled).

As we shall see, looking at the most computationally efficient way of solving equation (6), the explicit inversion of A is not needed and a variety of methods can be used, according to the specific properties of A. If for some special reason the inverse of the matrix is needed, the efficient way of doing it is through the repeated application of our model problem (6), changing the right hand side vector, but maintaining the matrix A.

A.1.3. Direct methods and iterative methods for solving determined systems of linear equations.

From the numerical standpoint, a first sub-division of methods for solving a linear system of equations like (6) arises:

a) Direct methods: where an exact solution of the linear system is reached in a finite number of steps (or elementary arithmetic operations), provided that no rounding errors exist.

Usually direct methods are associated with algorithms like Gaussian or Gauss-Jordan elimination.

b) Iterative methods: where starting from an approximate initial solution (say $x = 0$, or a better one if available), an appropriate algorithm is applied in order to improve the initial solution and obtain an approximate solution of the linear system.

In this case, the number of steps and elementary arithmetic operations is not finite and it depends on various factors, e.g.: on the algorithm used, on how approximate the final solution required is, or on how close is the initial solution to the final

one.

In general, an approximate solution to the linear system at the k-th step of the iterative process will be denoted by $x^{(k)}$ and is such that:

$$A\, x^{(k)} \approx b \tag{8}$$

In this iterative context, we can define a residual vector "r" such that:

$$r = b - A\, x^{(k)} \tag{9}$$

and the iterative procedure for solving the linear system has the goal of producing a residual vector whose norm (or "length", roughly speaking) is smaller than some pre-specified value (accuracy). In other words, the iterative process will be stopped when the approximate solution $x^{(k)}$ produces a residual vector whose norm is smaller than a pre-specified accuracy (scalar), say ϵ , i.e.

$$b - A\, x^{(k)} = r \qquad \text{such that } ||r|| < \epsilon \tag{10}$$

$x^{(k)}$ will be the solution corresponding to this level of accuracy. However, equation (10) relies on the definition of a norm for the residual vector, which can be any one of a variety of norms available in vector analysis, such as:

i) Euclidean norm : $$||r||_2 = \sqrt{r^T r} \tag{11}$$

ii) Maximum norm : $$||r||_\infty = \max_{1 \le i \le n} |r_i| \tag{12}$$

iii) Weighted p-norm :

$$||r||_{p,w} = \left(\sum_{i=1}^{n} |r_i|^p\, w_i \right)^{(1/p)} \tag{13}$$

Iterative methods for the solution of linear systems are associated with algorithms like Jacobi, Gauss-Seidel, etc. We

shall review most of these methods in some detail in the next sections.

In practice, and since computers do work with a finite word length, direct methods do not produce exact solutions in a finite number of arithmetic operations (effect of roundoff errors). As a result, in the computational sense, all the methods are in practice iterative. Nevertheless, we shall maintain the distinction between both approaches since it is important from the conceptual point of view.

The decision on which approach to be used depends on the characteristics of the matrix A and the computer resources available. Thus, it has been accepted for a long time that, for sparse matrices, iterative methods are the best ones, both from the storage and amount of work point of view; nowadays the availability of efficient direct solutions challenges this assertion. For dense matrices it has been accepted that direct methods are the best choice [Conte & de Boor,(1983)], although either big storage computers or special out-of-core data handling routines should be available for really big problems. A considerable amount of research has been carried out in the last decade in this field, resulting in new efficient methods becoming available, both in the direct and iterative fields.

Since water distribution network analysis using the gradient method produces symmetric linear systems of equations, we shall concentrate our search for efficient linear system solvers on these special systems.

A.2. Review of direct methods for the solution of general dense linear systems of equations.

A.2.1. Gaussian elimination.

i) Introduction.

The basic idea behind all kinds of elimination methods can be expressed in the following terms. Let us suppose that we have a linear system of equations such that:

$$T\,x = b \tag{14}$$

where:

T : is a triangular matrix of coefficients, i.e.: only the upper or lower part has non-zeros. Let us assume that we are dealing with an upper triangular matrix U:

$$T = U \tag{15}$$

then, the problem (14) can be represented as:

$$U\,x = b \tag{16}$$

The system (16) is then of the form:

$$\left.\begin{aligned}
u_{11}x_1 + u_{12}x_2 + u_{13}x_3 + \ldots\ldots\ldots + u_{1n}x_n &= b_1 \\
u_{22}x_2 + u_{23}x_3 + \ldots\ldots\ldots + u_{2n}x_n &= b_2 \\
u_{33}x_3 + \ldots\ldots\ldots + u_{3n}x_n &= b_3 \\
\vdots \qquad\qquad & \vdots \\
u_{n-1n-1}x_{n-1} + u_{n-1n}x_n &= b_{n-1} \\
u_{nn}x_n &= b_n
\end{aligned}\right\} \tag{17}$$

It is easy to see that, from the last equation of the system (17), the solution for the last component of "x" can be obtained in a direct manner:

$$x_n = b_n / u_{nn} \qquad (18)$$

Following the same idea, the $(n-1)^{th}$ component of x (i.e. x_{n-1}) can be computed directly from the $(n-1)^{th}$ equation of (17), provided that we have computed x_n from equation (18) previously:

$$x_{n-1} = (b_{n-1} - u_{n-1n}\, x_n) / u_{n-1n-1} \qquad (19)$$

The key here is that, since we know x_n from equation (18), we can eliminate it from the rest of the system; this is done simply by sending the terms involving x_n to the right hand side vector. The same is valid for x_{n-1}, after computing it via equation (19).

Following the same procedure, we can express any unknown, say x_k, in terms of the previously eliminated unknowns x_{k+1}, x_{k+2}, .. x_n, via the following relationship:

$$x_k = (b_k - \sum_{j=k+1}^{n} u_{kj}\, x_j) / u_{kk} \qquad (20)$$

Equation (20) defines an algorithm to eliminate successively all the variables, producing the values of the "n" unknowns in the linear system represented by equation (16). This process is known as a back-substitution process. Back-substitution arises from the fact that the matrix of coefficients in equation (16) is upper-triangular. In the case of T = L, a lower triangular matrix, equation (14) becomes:

$$L\, x = b \qquad (21)$$

and, instead of back-substitution, forward substitution can be implemented following the same pattern as before, but in the opposite direction, i.e. the order of the elimination starts with x_1, followed by x_2, x_3, ... etc. up to x_n. The equation which

defines the forward substitution algorithm becomes:

$$x_k = (b_k - \sum_{j=1}^{k-1} l_{jk}\, x_k) \;/\; l_{kk} \qquad (22)$$

where:

l_{ij} : is the coefficient corresponding to row i and column j of L.

It can be easily demonstrated that the amount of work demanded by either backward or forward substitution is the same, and is equal to:

$$\left.\begin{array}{lll} \text{Number of subtractions} & = & n(n-1)/2 \\ \text{Number of divisions} & = & n \\ \text{Number of multiplications} & = & n(n-1)/2 \end{array}\right\} \; n(n+1)/2 \qquad (23)$$

Since normally we count the amount of work in terms of the number of multiplications/divisions (dropping the additions and subtractions), equation (23) leads to a simpler result which says that the amount of work for the substitution process is of the order of $n^2/2$.

Also, it is easy to see that the solution of any triangular linear system exists, provided that the diagonal elements (either u_{kk} or l_{kk}) are all non-zero, and that, in this case, the solution is unique.

In summary, we have shown so far that, if we have a linear system of equations where the matrix of coefficients is triangular, then the solution for "x" is easily obtainable via a substitution process.

Hence, to solve a linear system where A is a general matrix, all we need to do is to transform it into a triangular matrix, since we already know that the rest is simply a substitution step.

Gaussian elimination provides an algorithm for transforming a general matrix into a triangular one. This algorithm is based on the following theorem from linear algebra [Conte and de Boor (1983)]:

On solving the linear system Ax = b we are allowed to perform any of the following elementary operations:

- Multiplication of any equation by an arbitrary scalar.
- Add one equation to the multiple of any other.
- Interchange any two equations.

In so doing, we shall obtain a new system, say $A^* x = b^*$, which is equivalent to the original one (Ax = b); i.e. the solution of both systems is exactly the same (x).

Hence, we can use a combination of these elementary operations to transform our original system Ax = b into another one $A^* x = b^*$, adding the additional condition that A^*, the transformed matrix, must be triangular (upper or lower).

From now on, and for simplicity, we shall assume that we want to obtain an upper-triangular transformed matrix. It is straightforward to see that the same reasoning is valid in the case of a lower-triangular matrix.

To produce the upper triangular matrix from the original matrix, all we need to do is to proceed column by column (though a row by row approach is also possible), eliminating (i.e. leaving a zero in) all the positions under the diagonal elements. To do so, we are allowed to perform any combination of the elementary operations mentioned before. This process finishes with an upper triangular matrix.

An example helps to illustrate how Gaussian elimination works. Let us suppose we want to solve the linear system:

$$\begin{aligned} 2\,x_1 + 3\,x_2 - \;\; x_3 &= 5 \\ 4\,x_1 + 8\,x_2 - 3\,x_3 &= 3 \\ -2\,x_1 + 3\,x_2 - \;\; x_3 &= 1 \end{aligned} \qquad (24)$$

The first step is to eliminate all those terms under the diagonal corresponding to the first row (i.e. under 2 in the upper left hand corner of the matrix, which is the so called "pivot element" at this stage); using a combination of elementary operations we get:

$$\begin{aligned} 2\,x_1 + 3\,x_2 - \;\; x_3 &= 5 \\ 2\,x_2 - \;\; x_3 &= -7 \\ 6\,x_2 - 2\,x_3 &= 6 \end{aligned} \qquad (25)$$

The second step is to eliminate the 6 under the diagonal of the second row, which leads to:

$$\begin{aligned} 2\,x_1 + 3\,x_2 - \;\; x_3 &= 5 \\ 2\,x_2 - \;\; x_3 &= -7 \\ x_3 &= 27 \end{aligned} \qquad (26)$$

And we have got an upper triangular linear system, which can be easily solved by back-substitution, leading to the solution:

$$\begin{aligned} x_3 &= 27 \\ x_2 &= 10 \\ x_1 &= 1 \end{aligned}$$

This is the basic algorithm for Gaussian elimination, which can be easily implemented in a computer. The only probable cause of problems can be the eventual presence of zeros in the current pivot element (at any stage of the elimination), in that case the procedure breaks-down. Moreover, even if the pivot is not exactly zero, but very small, the process can produce unacceptable errors leading to a completely wrong solution. The remedy to this stability problem is to determine a different strategy for the selection of the pivot element.

ii) Pivoting strategies.

As far as the stability of Gaussian elimination is concerned, when A is a general matrix, there are two main strategies for the selection of the pivot, in order to avoid a zero (or nearly zero) in the current pivot and to minimize the effect of rounding errors: scaled partial pivoting and total pivoting.

a) Scaled partial pivoting: under this strategy we are allowed to decide which equation (or row) will be picked up as pivotal equation. Let us assume that we are at the k-th step, then we shall choose the j-th equation as the new pivotal equation if and only if the j-th element in column k is the largest one (in absolute value), i.e.:

$$|w_{kj}| \geq |w_{ki}| \quad \forall\ i=k,k+1,\ \ldots,\ n \quad i\neq j$$

Once the new pivotal equation has been found, we perform a permutation between rows "k" and "j" and carry on with the elimination of the elements under this new pivot. Notice that if the original matrix is symmetric, this property is lost on

following this kind of pivoting.

In the previous example, represented by equation (24), partial pivoting, as described before, is performed in the following sequence:

Original system:

$$\begin{aligned} 2\,x_1 + 3\,x_2 - \;\; x_3 &= 5 \\ 4\,x_1 + 8\,x_2 - 3\,x_3 &= 3 \\ -2\,x_1 + 3\,x_2 - \;\; x_3 &= 1 \end{aligned} \qquad (24)$$

The first pivot is found by looking at the first column and finding out that the second row has the biggest (absolute value) element. Then, the first and second rows are permuted, leading to:

$$\begin{aligned} 4\,x_1 + 8\,x_2 - 3\,x_3 &= 3 \\ 2\,x_1 + 3\,x_2 - \;\; x_3 &= 5 \\ -2\,x_1 + 3\,x_2 - \;\; x_3 &= 1 \end{aligned} \qquad (27)$$

The elimination of the terms under the pivot produces:

$$\begin{aligned} 4\,x_1 + 8\,x_2 - 3\,x_3 &= 3 \\ - \;\; x_2 + 1/2x_3 &= 7/2 \\ 7\,x_2 - 5/2x_3 &= 5/2 \end{aligned} \qquad (28)$$

Now, the second pivot is chosen looking at the second column of the system (28); we have to choose between the second and third rows, and we find that the third row is the new pivotal equation. On permuting the second and third rows of (28), we get:

$$\begin{aligned} 4\,x_1 + 8\,x_2 - 3\,x_3 &= 3 \\ 7\,x_2 - 5/2x_3 &= 5/2 \\ - \;\; x_2 + 1/2x_3 &= 7/2 \end{aligned} \qquad (29)$$

Eliminating the term under the pivot (7), we get:

$$\begin{aligned} 4\,x_1 + 8\,x_2 - 3\,x_3 &= 3 \\ 7\,x_2 - 5/2x_3 &= 5/2 \\ 2/14x_3 &= 54/14 \end{aligned} \qquad (30)$$

Hence, the solution obtained via back-substitution becomes:

$$x_3 = 27$$
$$x_2 = 10$$
$$x_1 = 1$$

as before.

Scaled partial pivoting is a powerful complement of the standard Gaussian elimination, which avoids most of the sources of trouble of the original method, at a very low expense.

Although the example does not show the full advantages of scaled partial pivoting in some "pathological" cases, such examples are presented in the literature (e.g. Conte and de Boor, (1983) chapter 4.3.)

b) Total pivoting (complete or full pivoting): in this case we are allowed to decide which equation (row) and which variable (column) will be picked-up as pivotal equation and pivot element. Thus, we have now two degrees of freedom for choosing the pivot, and the search of the pivot will be carried out looking both at the columns and rows to the left and under the last pivot. Obviously we shall choose that corresponding to the element with the maximum absolute value.

In our previous example, represented by equation (24):

$$\begin{aligned} 2\,x_1 + 3\,x_2 - x_3 &= 5 \\ 4\,x_1 + 8\,x_2 - 3\,x_3 &= 3 \\ -2\,x_1 + 3\,x_2 - x_3 &= 1 \end{aligned} \qquad (24)$$

When choosing the first pivot, we find out that the maximum absolute value in the whole matrix is in the second row and second column (8), then, we permute both first and second rows and columns, obtaining:

$$\begin{aligned} 8\,x_2 + 4\,x_1 - 3\,x_3 &= 3 \\ 3\,x_2 + 2\,x_1 - \;\;x_3 &= 5 \\ 3\,x_2 - 2\,x_1 - \;\;x_3 &= 1 \end{aligned} \qquad (31)$$

The elimination of the elements under the pivot leads to:

$$\begin{aligned} 8\,x_2 + 4\,x_1 - 3\,x_3 &= 3 \\ 1/2\,x_1 + 1/8 x_3 &= 31/8 \\ -28\,x_1 + \;\;x_3 &= -1 \end{aligned} \qquad (32)$$

We choose the second pivot looking at all the elements in the second and third rows and we find that the new pivot is -28 (third row, variable x_1), then we have to permute the second with the third rows:

$$\begin{aligned} 8\,x_2 + 4\,x_1 - 3\,x_3 &= 3 \\ -28\,x_1 + \;\;x_3 &= -1 \\ 1/2\,x_1 + 1/8 x_3 &= 31/8 \end{aligned} \qquad (33)$$

Hence, eliminating the element under the second pivot:

$$\begin{aligned} 8\,x_2 + 4\,x_1 - 3\,x_3 &= 3 \\ -28\,x_1 + \;\;x_3 &= -1 \\ 8/3 x_3 &= 216/7 \end{aligned} \qquad (34)$$

which produces by back-substitution:

$$\begin{aligned} x_3 &= 27 \\ x_2 &= 10 \\ x_1 &= 1 \end{aligned}$$

as before.

Although total pivoting is recognised as more powerful than scaled partial pivoting, it is not frequently used, mainly because of the additional amount of work involved, which is relevant in large systems, since this involves a search through the whole matrix .

iii) Amount of work and storage required.

The amount of work involved by Gaussian elimination is:

- Number of additions : $(n^3-n)/3$
- Number of multiplications : $(n^3-n)/3$
- Number of divisions : $n(n-1)/2$

(35)

Hence, we have the following number of multiplications/divisions:

$$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6} \tag{36}$$

and, as far as the amount of work is concerned, the Gaussian elimination process is said to be of the order of $n^3/3$.

Thus, the complete solution of a linear system via Gaussian elimination plus back-substitution, requires the following amount of work:

Stage	Additions	Multiplications/divisions
Factorization	$(n^3-n)/3$	$n^3/3 + n^2/2 - 5/6n$
Substitution	$n^2/2 - n/2$	$n^2/2 + n/2$
Total	$n^3/3 + n^2/2 + 5/6n$	$n^3/3 + n^2 - n/3$

As a result, we note that the solution of linear systems via a sequence of Gaussian elimination and back-substitution requires an amount of work which is, roughly speaking, of the order of $n^3/3$; the factorization process, i.e. the transformation of the original matrix into a triangular one is the most expensive item in the total cost.

As far as the cost of pivoting is concerned, scaled partial pivoting requires of the order of n^2 comparisons, while total pivoting needs of the order of $n^3/3$ comparisons [Golub and Van Loan (1983)].

Since the triangular matrix produced by Gaussian elimination can be completely filled with non-zeros, the amount of storage required is:

$$n(n+1)/2 = n^2/2 + n/2 \qquad (37)$$

that is to say, from the storage point of view, Gaussian elimination, for general dense matrices, requires of the order of $n^2/2$ locations.

Because the explicit inversion of a matrix requires of the order of n^3 multiplications/divisions, it is now clear why Gaussian elimination with substitution is a more efficient method for solving linear systems. Indeed, we can produce the same results with just a third of the work. From the storage viewpoint, the solution of linear systems via Gaussian elimination requires only a half of the storage needed for an explicit inversion of the matrix. Unless it is specially justified, matrix inversion should be avoided.

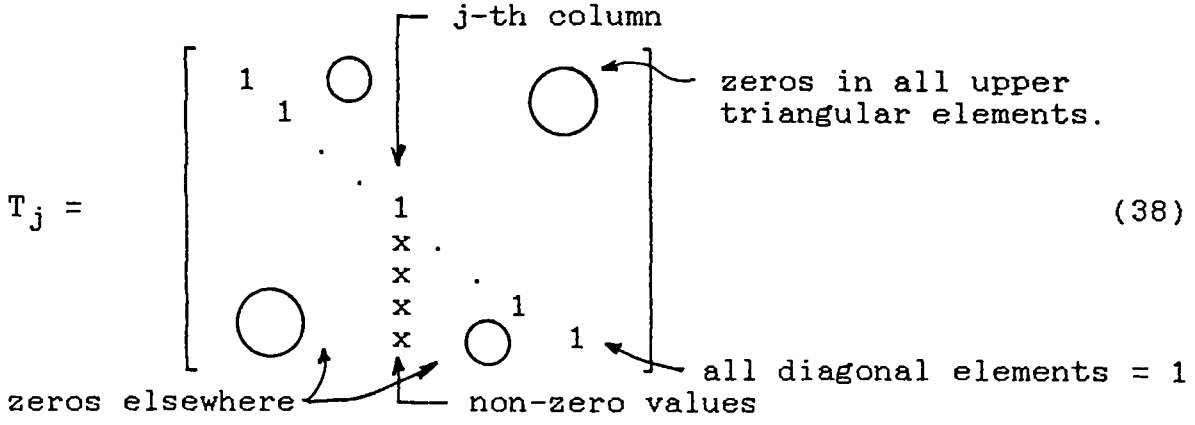
We have to emphasise that, in the evaluation of the work and storage required presented so far, we have not taken into account that, in some cases, the special structure of the matrix (symmetry, sparseness) can be exploited, in order to reduce both the amount of work and the storage needed . Since most of the matrices produced in engineering applications have some special feature, and since in particular, the matrices generated in the context of water distribution analysis via the gradient method are always symmetric and positive definite, then we do have to take these properties into consideration, since we can get important savings in the amount of computation and storage

required.

iv) The product form of Gaussian elimination.

The whole process of Gaussian elimination can be understood as a transformation process, from a general matrix A into a triangular matrix. In fact, each step of Gaussian elimination can be formally expressed as a transformation, which in this case takes the form of the product of a matrix by another; in so doing, the whole elimination process becomes just a sequence of matrix operations. This concept is relevant from the practical point of view, since it can lead to important savings in the amount of work to be done, as will be shown in subsequent sections.

Gaussian elimination is equivalent to premultiplying the original system by a sequence of transformation matrices (T_i). The structure of these matrices is such that, if we are eliminating the terms under the diagonal of the j-th column, the corresponding transformation matrix is of the following form:



where the j-th column has indeed the following form:

$$
\left[\begin{array}{c}
0 \\
\vdots \\
0 \\
1 \\
\dfrac{-a_{j+1,j}}{a_{j,j}} \\
\dfrac{-a_{j+2,j}}{a_{j,j}} \\
\vdots \\
\dfrac{-a_{n,j}}{a_{j,j}}
\end{array}\right]
\begin{array}{l}
\Big\} \text{ j-1 zeros} \\
\leftarrow\!\!- \text{ j-th row}
\end{array}
\qquad (39)
$$

Note that the values of the coefficients a_{ij} are those of the current matrix in the elimination process, rather than those of the original matrix .

Thus, the original system is successively premultiplied by the matrices T_i , i=1,2,...,n, such that:

Original system: $A\,x = b$

1st step : $T_1\,A\,x = T_1\,b$

or $A_1\,x = T_1\,b$

with $A_1 = T_1\,A$

2nd step : $T_2\,A_1\,x = T_2\,T_1\,b$

or $A_2\,x = T_2\,T_1\,b$

with $A_2 = T_2\,A_1 = T_2\,T_1\,A$

n-th step : $T_n\,A_{n-1}\,x = T_n\,T_{n-1}\,....\,T_2\,T_1\,b$ (40)

or $A_n\,x = T_n\,T_{n-1}\,....\,T_2\,T_1\,b$ (41)

with $A_n = T_n\,T_{n-1}\,....\,T_2\,T_1\,A$ (42)

After the n-th step, A_n must be an upper triangular matrix, and the system (41) can be solved by back-substitution.

In the context of our previous small example [equation (24)]:

$$\begin{aligned} 2\,x_1 + 3\,x_2 - \;\; x_3 &= 5 \\ 4\,x_1 + 8\,x_2 - 3\,x_3 &= 3 \\ -2\,x_1 + 3\,x_2 - \;\; x_3 &= 1 \end{aligned} \qquad (24)$$

The sequence of transformation matrices is:

For the first step:

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad \text{-->} \quad A_1 = T_1\,A = \begin{bmatrix} 2 & 3 & -1 \\ 0 & 2 & -1 \\ 0 & 6 & -2 \end{bmatrix}$$

For the second step:

$$T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{bmatrix} \quad \text{-->} \quad A_2 = T_2\,T_1\,A = \begin{bmatrix} 2 & 3 & -1 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that the matrices T_j are highly sparse and that they can be handled implicitly with a vector of length "n".

The practical importance of the product form of Gaussian elimination lies in the fact that, if we need to solve different linear systems with the same matrix of coefficients, for example:

$$A\,x = b$$

$$A\,y = c$$

$$A\,z = d$$

then, in this case, all we need to do is to transform the matrix A into:

$$A_n = T_n\,T_{n-1}\,\ldots\ldots\,T_2\,T_1\,A$$

and apply back-substitution to the transformed systems:

$$\begin{aligned} A_n\,x &= b^* \\ A_n\,y &= c^* \\ A_n\,z &= d^* \end{aligned}$$

with:

$$b^* = T_n\,T_{n-1}\,\ldots\ldots T_2\,T_1\,b$$

$$c^* = T_n\,T_{n-1}\,\ldots\ldots T_2\,T_1\,c$$

and

$$d^* = T_n\,T_{n-1}\,\ldots\ldots T_2\,T_1\,d$$

Hence, in this case, we can compute and store a transformation matrix:

$$T = T_n \; T_{n-1} \; \ldots . T_2 \; T_1 \qquad (43)$$

with T_j as specified in (38) and (39), which can be used as many times as different right hand side vectors are given to us. As can be seen in our small example, the matrix T takes the form:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 7 & -3 & 1 \end{bmatrix} = T_2 \; T_1 \quad \text{(note that } T_3 = I\text{)}$$

which is a lower triangular matrix with 1's in the diagonal.

Another immediate application of the product form of Gaussian elimination is matrix inversion , since this problem can be formulated as the solution of "n" linear systems of the form:

$$A \; x^{(i)} = e^{(i)} \qquad (44)$$

where:

$x^{(i)}$: are the columns vectors of the matrix A^{-1}.

$e^{(i)}$: are vectors of the form $(0,0,\ldots,1,\ldots,0)^T$, i.e. null vectors with a one added in the i-th position.

Hence, we need to solve "n" times the same linear system, with "n" different right hand side vectors; the corresponding "n" solution vectors are assembled by columns to produce the inverse matrix:

$$A^{-1} = [\; x^{(1)} \mid x^{(2)} \mid \ldots \mid x^{(n)}] \qquad (45)$$

v) Gaussian elimination as a LU decomposition.

Gaussian elimination can also be interpreted as an equivalent triangular decomposition of the matrix A, since when multiplying T A we are actually getting an upper triangular matrix, say U :

$$T\ A = U \tag{46}$$

then,

$$A = T^{-1}\ U \tag{47}$$

where T^{-1} is a lower triangular matrix: L , such that:

$$L = T^{-1} \tag{48}$$

and then

$$A = L\ U \tag{49}$$

in terms of our small example:

$$L = T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 3 & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} 2 & 3 & -1 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

Hence, Gaussian elimination is closely related to a family of factorization methods, which shall be reviewed in subsequent sections.

A.2.2. Gauss-Jordan elimination.

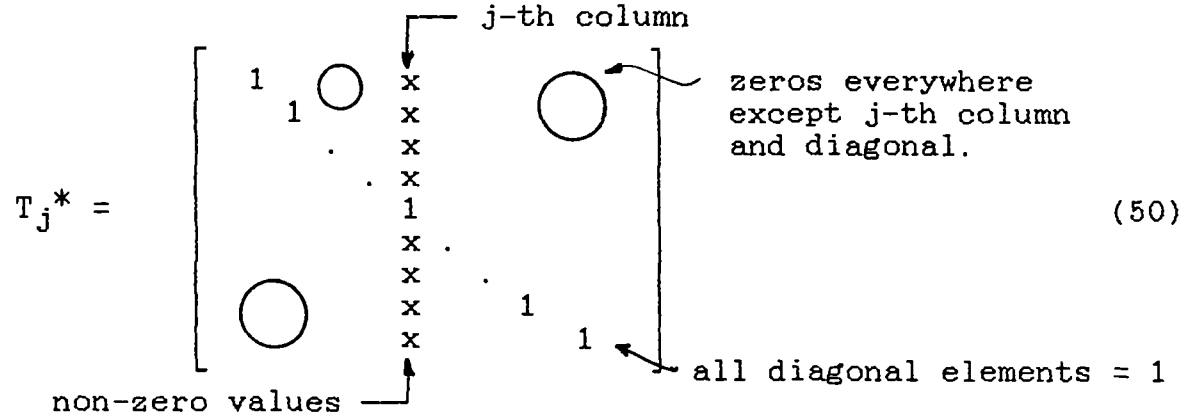
i) Introduction.

Gauss-Jordan elimination is similar to Gaussian elimination, the difference being that instead of producing a triangular matrix, eliminating only the non-zeros below the diagonal, we can carry on the elimination both below and above the diagonal, performing as many elementary operations as needed, until we get a transformed matrix which is an identity matrix .

ii) The product form of the inverse.

In a similar manner to Gaussian elimination, Gauss-Jordan elimination can be explained in terms of the product of a succession of transformation matrices T_j^*.

The structure of these transformation matrices is now the following:



$$T_j^* = \quad (50)$$

the j-th column has the following form:

$$\begin{bmatrix} -\frac{a_{1,j}}{a_{j,j}} \\ -\frac{a_{2,j}}{a_{j,j}} \\ \vdots \\ \frac{1}{a_{j,j}} \\ \vdots \\ -\frac{a_{n-1,j}}{a_{j,j}} \\ -\frac{a_{n,j}}{a_{j,j}} \end{bmatrix} \quad \leftarrow \text{j-th row} \qquad (51)$$

Equations (50) and (51) are for Gauss-Jordan, as equations (38) and (39) are for Gaussian elimination.

Note that the values of the coefficients a_{ij} are those of the current matrix in the elimination process, rather than those of the original matrix.

Thus, the successive premultiplications are carried out in the following sequence:

Original system: $A\,x = b$

1st step : $T_1^*\,A\,x = T_1^*\,b$

or $A_1^*\,x = T_1^*\,b$

with $A_1^* = T_1^*\,A$

2nd step : $T_2^*\,A_1^*\,x = T_2^*\,T_1^*\,b$

or $A_2^*\,x = T_2^*\,T_1^*\,b$

with $A_2^* = T_2^*\,A_1^* = T_2^*\,T_1^*\,A$

n-th step : $T_n^*\,A_{n-1}^*\,x = T_n^*\,T_{n-1}^*\,\ldots\,T_2^*\,T_1^*\,b$ (52)

or $A_n^*\,x = T_n^*\,T_{n-1}^*\,\ldots\,T_2^*\,T_1^*\,b$ (53)

with $A_n^* = T_n^*\,T_{n-1}^*\,\ldots\,T_2^*\,T_1^*\,A$ (54)

after the n-th step, A_n^* must be an identity matrix and the solution is exactly the current right hand side vector, hence no back-substitution is needed, i.e.:

$$A_n^* = T_n^*\,T_{n-1}^*\,\ldots\,T_2^*\,T_1^*\,A = I \qquad (55)$$

then:

$$A^{-1} = T_n^*\,T_{n-1}^*\,\ldots\,T_2^*\,T_1^* \qquad (56)$$

this is the so-called product form of the inverse of the matrix A.

Gauss-Jordan elimination provides us with another way to build-up the inverse of a matrix by simply multiplying "n"

transformation matrices of the form of T_j^*, which are very sparse.

The amount of work needed for Gauss-Jordan elimination is:

- Number of additions : $n^3/2 - n/2$
- Number of multiplications : $n^3/2 + n^2 - n/2$
- Number of divisions : n

Since the inverse of a matrix can be a completely filled matrix, the storage requirements are n^2 positions.

As before, in the case of Gaussian elimination, the usefulness of Gauss-Jordan elimination for the repeated solution of linear systems which differ only in the right hand side vector is quite clear, with no need for further explanation.

A.2.3. Matrix factorization methods.

i) Introduction.

We have already seen that Gaussian elimination is, in essence, equivalent to factorizing the matrix of coefficients into the product of a lower and an upper triangular matrix, although in the standard Gaussian algorithm both matrices are not explicitly determined.

In fact we can follow different strategies for obtaining this factorization, each one leading to a different direct method for the solution of linear systems of equations. We shall review some of these factorization (or decomposition) strategies in the following sections.

The common idea behind any factorization strategy is that instead of having to solve the original system:

$$A\ x = b \tag{57}$$

if we factorize A as:

$$A = L\ U \tag{58}$$

where:

L is a lower triangular matrix.

U is an upper triangular matrix.

then, because the original problem is now :

$$L\ U\ x = b \tag{59}$$

if we define:

$$U\ x = y \tag{60}$$

where y is an auxiliary vector, the solution of the original system for x, can be split up into two substitution stages:

a) Forward substitution:

solve $L\ y = b$ (61)

determining "y", and

b) Backward substitution: with "y" already known

solve equation (60) $U\ x = y$

determining "x", the original unknown.

Of course, this two stage process can be effectively used for solving multiple right hand side problems, where we only need to factorize the original matrix at the beginning, the rest being just a succession of forward and backward substitutions.

ii) LU triangular factorization (Crout method).

In this case the lower-upper triangular factorization takes the following form:

$$A = L\ U \tag{62}$$

or:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & & \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & & \vdots \\ & & & u_{nn} \end{bmatrix} \quad (63)$$

where:

A : is a general unsymmetric matrix of coefficients.

L : is a lower triangular matrix whose diagonal elements are all 1.

U : is an upper triangular matrix.

Under these conditions, L and U are unique factors.

From equation (63) it can be seen that a recurrent scheme can be devised to compute the elements of the factors L and U, from the original elements of A.

On multiplying the first row of L by the columns of U, and identifying it with the first row of A, we find that the first row of U is simply the same first row of A and no computation is needed, i.e.:

$$u_{1i} = a_{1i} \quad \forall\ i=1,2,\ldots,n \quad (64)$$

The following step is to determine the elements of the first column of L; in this case, multiplying the first column of L by the first row of U and identifying the result with the first column of A, we get:

$$l_{i1}\ u_{11} = a_{i1} \quad \forall\ i=2,3,\ldots,n \quad (65)$$

then,

$$l_{i1} = a_{i1}/u_{11} \quad \forall\ i=2,3,\ldots,n \quad (66)$$

At this stage we are able to compute the second row of U, by

multiplying the second row of L by the columns of U and identifying the result with the second row of A, to obtain:

$$l_{21}\, u_{1i} + u_{2i} = a_{2i} \quad \forall\ i=2,3,\ldots,n \tag{67}$$

then,

$$u_{2i} = a_{2i} - l_{21}\, u_{1i} \quad \forall\ i=2,3,\ldots,n \tag{68}$$

Following the same pattern, the whole set of elements of L and U can be computed. Since we need to compute the terms $1/u_{ii}$, the algorithm breaks down when $u_{ii} = 0$. Also, the stability of the method can be affected when a very small u_{ii} is found. As in Gaussian elimination, partial or complete pivoting can help to stabilise the algorithm.

The amount of work demanded by this L U factorization is:

- Number of multiplications : $n^3/3 - n/3$
- Number of divisions : n
- Number of additions : $n^3/3 - n^2/2 + n/6$

Having determined the factors L and U, the solution of the linear system requires a forward and a backward substitution which, as we already know each substitution requires $n(n+1)/2$ multiplications/divisions, giving a total of $n^3/3 + n^2 + 2/3n$ multiplications/divisions for the solution of the linear system (factorization plus substitutions), which is basically the same amount of work needed for the Gaussian elimination process. The storage required is n^2 positions.

iii) L L^T factorization.

In the particular case when A is <u>symmetric and positive definite</u>, it can be seen that Crout's method leads to a symmetric

decomposition:

$$A = L\,U = L\,L^T \tag{69}$$

where $U = L^T$ is the transpose of L.

The amount of work is reduced to almost one half ($n^3/6 + n^2/2$ multiplications/divisions in the factorization stage) and the storage requirements are also halved.

Of course, the solution of a linear system using this symmetric decomposition demands a forward and a backward substitution, as before:

- Solve $L\,y = b$ for y
- Solve $L^T\,x = y$ for x

We shall return to this kind of factorization later on, when dealing with Cholesky factorizations.

iv) L D U factorization.

An extension of the LU factorization consists of decomposing ever further the upper triangular matrix defined in equation (63), into the product of a diagonal matrix (D) and an upper (unit diagonal) triangular matrix:

$$A = L\,D\,U \tag{70}$$

or:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & & \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} d_{11} & & & \\ & d_{22} & & \\ & & \ddots & \\ & & & d_{nn} \end{bmatrix} \cdot \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ & 1 & \cdots & u_{2n} \\ & & & \vdots \\ & & & 1 \end{bmatrix} \tag{71}$$

where:

A : is a general non-symmetric matrix of coefficients.

L : is a lower triangular matrix whose diagonal elements are all 1.

D : diagonal matrix

U : is an upper triangular matrix whose diagonal elements are all 1.

Thus, instead of solving the original system:

$$A\ x = b \tag{72}$$

we are solving:

$$L\ D\ U\ x = b \tag{73}$$

and in this case the solution involves a three stage process:

a) Forward substitution:

solve $L\ z = b$ (74)

for z, an auxiliary vector

b) Diagonal inversion:

solve $D\ y = z$ (75)

for y, by computing directly :

$$y = D^{-1}\ z \tag{76}$$

this stage demands only n divisions, due to the fact that D is diagonal.

c) Back-substitution:

solve $U\ x = y$ (77)

for x, the original unknown vector.

For the factorization, the algorithm requires $n^3/3 + n^2/2 + n/6$ multiplications/divisions. The advantage of this kind of factorization over LU factorization, becomes apparent in the handling of storage and in the implementation of the algorithms.

More details can be found for example in Golub and Van Loan (1983), algorithm 5.1.1.

v) L D L^T factorization.

If the matrix of coefficients of the linear system is a non-singular symmetric matrix, it can be proved [Golub and Van Loan (1983), theorem 5.1.2.] that in the L D U factorization the upper triangular matrix U is equal to the transpose of the lower triangular matrix, i.e.:

$$U = L^T \tag{78}$$

and

$$A = L\ D\ L^T \tag{79}$$

It is easy to see that the amount of work for the decomposition stage now reduces to almost half of that of the L D U decomposition ($n^3/6$ multiplications/divisions, approximately), i.e. it is almost the same amount of work needed for the L L^T factorization.

Of course, the solution of the linear system demands successive forward substitution, diagonal inversion and backward substitution, in a similar way as in the case of the previous L D U decomposition.

From the stability point of view the decomposition is not numerically stable and the condition of non singularity of A does not seem to be sufficient, a more restrictive condition is needed. An example provided by Gill, Murray and Wright (1981) is useful to illustrate this problem:

the matrix $A = \begin{bmatrix} \epsilon & 1 \\ 1 & \epsilon \end{bmatrix}$

has a L D L^T factorization given by:

$$L = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix}, \quad D = \begin{bmatrix} \epsilon & 0 \\ 0 & \epsilon - 1/\epsilon \end{bmatrix} \quad \text{and} \quad L^T = \begin{bmatrix} 1 & 1/\epsilon \\ 0 & 1 \end{bmatrix}$$

hence, for $\epsilon << 1$ the elements of the factors can become very large.

vi) Complete Cholesky factorization.

If we have, as usually occurs in engineering problems, a symmetric positive definite matrix A, the L D L^T factorization previously seen has the property that all diagonal elements of D are strictly positive [see theorem 5.2.1, Golub and Van Loan (1983) for a proof] and then, two alternative decompositions can be developed:

a) L D L^T Cholesky complete decomposition.

The matrix of coefficients of the linear system of equations can be decomposed as:

$$A = L\ D\ L^T \tag{80}$$

that is to say:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} =$$

$$= \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} d_{11} & & & \\ & d_{22} & & \\ & & \ddots & \\ & & & d_{nn} \end{bmatrix} \cdot \begin{bmatrix} l_{11} & l_{12} & \cdots & l_{1n} \\ & l_{22} & \cdots & l_{2n} \\ & & & \vdots \\ & & & l_{nn} \end{bmatrix} \tag{81}$$

where:

$a_{ij} = a_{ji}$, since the matrix A is symmetric, and

$d_{ii} > 0 \ \forall \ i=1,2,\ldots,n$

An algorithm for the computation of the elements of L and D can be developed in a constructive fashion, considering first the product of L D as:

$$L^* = L\ D \tag{82}$$

then,

$$L^* = \begin{bmatrix} l_{11}\,d_{11} & & & \\ l_{21}\,d_{11} & l_{22}\,d_{22} & & \\ l_{31}\,d_{11} & l_{32}\,d_{22} & l_{33}\,d_{33} & \\ \vdots & \vdots & \vdots & \\ l_{n1}\,d_{11} & l_{n2}\,d_{22} & l_{n3}\,d_{33} & \cdots\cdots\ l_{nn}\,d_{nn} \end{bmatrix} \tag{83}$$

Thus, $A = L\ D\ L^T = L^*\ L^T$, and it takes the form:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} =$$

$$= \begin{bmatrix} l_{11}\,d_{11} & & & \\ l_{21}\,d_{11} & l_{22}\,d_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1}\,d_{11} & l_{n2}\,d_{22} & \cdots & l_{nn}\,d_{nn} \end{bmatrix} \cdot \begin{bmatrix} l_{11} & l_{12} & \cdots & l_{1n} \\ & l_{22} & \cdots & l_{2n} \\ & & & \vdots \\ & & & l_{nn} \end{bmatrix} \tag{84}$$

To determine the elements l_{ij} and d_{ij} we have to multiply the elements of L^* by those of L^T and identify the results with the corresponding elements of A. In so doing, we find that we have one degree of freedom at our disposal, for example, we can impose the condition that all the diagonal elements of L^* are 1:

$$l_{ii}\ d_{ii} = 1 \quad \forall \ i=1,2,..,n \tag{85}$$

and then the algorithm becomes:

Step 1 : Compute for j=i

$$l_{ji} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}\ l_{jk}\ d_{kk} \quad j=i,(i+1),\ldots,n \tag{86}$$

(at the beginning j=i=1 and $l_{ji}=a_{ij}$, then $l_{11}=a_{11}$)

Step 2 : Compute

$$d_{ii} = (l_{ii})^{-1} \qquad (87)$$

[this is due to equation (85)]

the algorithm is stable at this stage, since if A is positive definite then $l_{ii} \neq 0 \quad \forall\ i$

Step 3 : Compute

l_{ji} for $j > i$, using equation (86).

The amount of work required for the decomposition stage is of the order of $n^3/6 + n^2$ multiplications/divisions.

A couple of examples will illustrate the main features of the algorithm. First of all, an example with a symmetric matrix which is not positive definite will be considered, in this case we know in advance that the algorithm is not applicable:

Solve $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 0 \\ 3 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \\ 6 \end{bmatrix}$

determinant of A =-18, then the matrix is negative.

the factorization algorithm gives:

$l_{11} = 1$

$d_{11} = 1$

$l_{21} = 2$

$l_{31} = 3$

$l_{22} = 1-l_{11}^2\ d_{11} = 1 - 1 = 0$

$d_{22} = 1/0$ not defined !!, so the factorization is not possible.

An example where the factorization is possible is:

Solve $\begin{bmatrix} 2 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = b$

determinant of A = +1, then the matrix is positive definite

the factorization algorithm gives:

$$\begin{bmatrix} 2 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 2 & 3 & 0 \\ 1 & 1 & 1/6 \end{bmatrix} \cdot \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/3 & 0 \\ 0 & 0 & 6 \end{bmatrix} \cdot \begin{bmatrix} 2 & 2 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & 1/6 \end{bmatrix}$$

$$A = L \cdot D \cdot L^T$$

Note that $d_{ii} > 0 \quad \forall\ i$; also $l_{ij} > 0 \quad \forall\ i,j$.

b) $R\ R^T$ Cholesky complete factorization (square root factorization).

Using the fact that if A is symmetric and positive definite then D is also positive, then the factorization:

$$A = L\ D\ L^T$$

can be expressed as:

$$A = L\ D^{1/2}\ D^{1/2}\ L^T \tag{88}$$

where:

$$D^{1/2} = \begin{bmatrix} d_{11}^{1/2} & & & \\ & d_{22}^{1/2} & & \\ & & \ddots & \\ & & & d_{nn}^{1/2} \end{bmatrix} \tag{89}$$

The matrix $D^{1/2}$ exists since D is positive, the square roots in equation (89) being the reason for the restriction of positiveness in D.

Then, A can be expressed as:

$$A = R\ R^T \tag{90}$$

where:

$$R = L\ D^{1/2}$$

and $$R^T = D^{1/2}\ L^T$$

which is exactly the same L L^T factorization we obtained before.

The algorithm can be derived from the fact that equation (90) is of the form:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} r_{11} & & & \\ r_{21} & r_{22} & & \\ \vdots & \vdots & & \vdots \\ r_{n1} & r_{n2} & \cdots & r_{nn} \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & & \vdots \\ & & & r_{nn} \end{bmatrix} \qquad (91)$$

Then, multiplying R R^T and identifying with A term by term, we get (note that $r_{ij} = r_{ji}$, because A is symmetric):

$a_{11} = r_{11}^2$ ---------> $r_{11} = (a_{11})^{1/2}$

$a_{12} = r_{11}\, r_{12}$ ---------> $r_{12} = a_{12} / r_{11}$

$a_{1n} = r_{11}\, r_{1n}$ ---------> $r_{1n} = a_{1n} / r_{11}$

$a_{22} = r_{12}^2 + r_{22}^2$ -------> $r_{22} = (a_{22} - r_{12}^2)^{1/2}$

etc.

This is a row-oriented version of the Cholesky factorization and, of course, a column-oriented algorithm can be derived following the same pattern; see for example Golub and van Loan (1983), algorithm 5.2.1., which is efficient since the elements a_{ij} are overwritten by r_{ij} ($i \geq j$).

The algorithm requires $n^3/6$ multiplications/divisions and additions, and n square roots. If A is not positive definite, the terms under the square root become negative and the algorithm breaks down. Our previous non-positive small example helps to illustrate this point:

Solve :

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 0 \\ 3 & 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \\ 6 \end{bmatrix}$$

the determinant of A =-18, then the matrix is negative.

the factorization algorithm gives:

$r_{11} = (1)^{1/2}$ ----> $r_{11} = 1$

$r_{11}\, r_{12} = 2$ ----> $r_{12} = 2$

$r_{12}^2 + r_{22}^2 = 1$ ----> $r_{22} = (1 - 2^2)^{1/2}$ not defined !!

The main advantage of this version of Cholesky factorization is the fact that no permutations are needed to ensure stability, like in Gaussian elimination with pivoting. This is due to the step in the algorithm, where the diagonal elements are computed:

$$\sum_{i=1}^{k} r_{ik}^2 = a_{kk} \qquad \forall\ k=1,2,\ldots,n$$

which implies that the terms r_{ik} cannot grow indefinitely (like in the example of the L D L^T factorization) and their maximum limit is indeed $(a_{kk})^{1/2}$. This is always true, even if a very small pivot is found during the factorization. The implications of this will become clearer when dealing with direct sparse methods, because it will allow us to concentrate on reducing the creation of fill-in, without having to worry about stability.

vii) Orthogonalization methods.

A completely different alternative to Gaussian elimination methods is the use of orthogonal factorization. A number of orthogonal methods have been proposed, all of them exploiting the fact that, in an orthogonal matrix, say Q, the product Q^TQ is the

identity matrix; i.e. the inverse of an orthogonal matrix is its transpose.

As a result, if we want to solve our model problem A x = b and if we are able to factorize A such that:

$$A = Q\ U \tag{92}$$

where:

Q is orthogonal

U is an upper triangular matrix

then, the linear system becomes:

$$Q\ U\ x = b \tag{93}$$

and premultiplying by Q^T, we get:

$$U\ x = Q^T\ b \tag{94}$$

where a back substitution determines the unknown "x".

There are thus as many methods as factorizations that can be imagined.

The main advantage of all these methods is that they are especially well suited for ill-conditioned problems, where the matrix A becomes singular or nearly singular.

A method of this kind is the "singular value decomposition", briefly described by Press et al. (1986), where a FORTRAN implementation can be found. A more detailed description is provided by Golub and Van Loan (1983).

Unfortunately, these methods are much more costly than Gaussian elimination [Reid et al. (1986) reported at least twice the arithmetic cost of Gaussian elimination]. On top of that, the methods are not efficient in handling sparse matrices, since the

orthogonal factors get a great deal of fill-in. Because of these reasons, orthogonal methods are not widely used, their use being restricted to solve ill-conditioned problems.

A.2.4. Comparison of the different algorithms for the direct solution of general dense linear systems.

We have briefly reviewed most of methods for solving general dense linear systems of equations, the main algorithm being Gaussian elimination, while all the rest can be understood as variations of that method.

Because the main objective in this section has been to introduce the basic ideas behind the solution of linear systems of equations, for the particular case of general dense matrices, we have avoided a number of details and we shall maintain the same policy in this comparison. This is due to the fact that most of the methods we are interested in are described in the following sections, since sparsity is a feature that must be explicitly considered when solving efficiently linear systems arising from water distribution network analysis.

Because all the methods described so far can be programmed to overwrite the original matrix of coefficients, the storage ceases to be relevant in comparative terms. Hence, the amount of work and stability remain as the only relevant factors of the comparison.

Table A.1 summarises the amount of work and storage requirements of the different algorithms reviewed. From a

Table A.1. Comparison of the algorithms used in the direct solution of dense linear systems of equations.

Algorithm	Matrix	Amount of work required: Multiplications/divisions	Additions/subtractions	Storage required: Positions (*)	Notes
Substitution: back or forward	Triangular	$\frac{n^2}{2} + \frac{n}{2}$	$\frac{n^2}{2} - \frac{n}{2}$		
Gaussian elimination	Unsymmetric	$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5}{6}n$	$\frac{n^3}{3} - \frac{n}{3}$	$\frac{n^2}{2}$	Pivoting and permutations needed for stability.
Gauss elimin. + substitution	Unsymmetric	$\frac{n^3}{3} + n^2 - \frac{1}{3}n$	$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5}{6}n$	$\frac{n^2}{2}$	Idem
Gauss-Jordan elimination	Unsymmetric	$\frac{n^3}{2} + n^2 - \frac{1}{2}n$	$\frac{n^3}{2} - \frac{n}{2}$	n^2	Idem
L U factorization	Unsymmetric	$\frac{n^3}{3} - \frac{1}{3}n$	$\frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$	n^2	
L U factoriz.+ back+forw.subst.	Unsymmetric	$\frac{n^3}{3} + n^2 + \frac{2}{3}n$	$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5}{6}n$	n^2	
$L L^T$ factorization	Symmetric Pos.Defin.	$\frac{n^3}{6} + \frac{n^2}{2}$	$\frac{n^3}{6} + \frac{n^2}{4}$	$\frac{n^2}{2}$	Good from the stability point of view.
LL^T factoriz.+ back+forw.subst.	Symmetric Pos.Defin.	$\frac{n^3}{6} + \frac{3n^2}{2} + n$	$\frac{n^3}{6} + \frac{5n^2}{4} - n$	$\frac{n^2}{2}$	
L D U factorization	Unsymmetric	$\frac{n^3}{3} + \frac{n^2}{2} + \frac{1}{6}n$	$\frac{n^3}{3} - \frac{n^2}{2} + \frac{1}{6}n$	n^2	
LDU factoriz.+ back.+diag.inv+ forw.substitut.	Unsymmetric	$\frac{n^3}{3} + \frac{3n^2}{2} + \frac{13}{6}n$	$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5}{6}n$	n^2	
$L D L^T$ factorization	Symmetric Pos.Defin.	$\frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{2}$	$\frac{n^3}{6} + \frac{3n^2}{4} - \frac{1}{2}n$	$\frac{n^2}{2}$	
LDL^T factoriz.+ back.+diag.inv+ forw.substitut.	Symmetric Pos.Defin.	$\frac{n^3}{6} + 2n^2 + \frac{5}{2}n$	$\frac{n^3}{6} + \frac{7n^2}{4} - \frac{3}{2}n$	$\frac{n^2}{2}$	
Explicit inversion	Unsymmetric	$n^3 - 1$	$n^3 - 2n^2 + n$	n^2	

Notes:
(*) It does not consider the fact that most of the algorithms can overwrite the original matrix.

comparative point of view we can say that:

a) The explicit inversion of the matrix should be avoided when solving a linear system, unless it is explicitly needed for other purposes; in the latter case the product form of the inverse should be used since only requires of the order of $n^3/2$ multiplications and divisions.

b) When solving non-symmetric linear systems, Gaussian elimination either with partial or full pivoting can be the best choice, with Gauss-Jordan elimination being more expensive than Gaussian elimination. L D L^T factorization presents some stability problems and should be used with due care.

c) When solving multiple right hand side linear systems, a factorization of the matrix A is imperative, since this allows the solution for the different right hand sides with additional forward and backward substitutions.

d) For symmetric positive definite systems, the complete Cholesky factorization (square root method) seems to be the best choice, both from the stability and amount of work viewpoints. Symmetry must be exploited, since this reduces both storage and amount of work by 50 % .

e) For ill-conditioned problems try a method based on orthogonalization.

A.2.5. Error analysis in the solution of linear systems.

i) Introduction.

Although we have defined "direct methods" as those where an exact solution is reached in a finite number of elementary mathematical operations, in practice and due to the fact that all computers have a fixed word length, which determines its maximum accuracy, exact solutions are never reached. Then, there are no exact solutions just approximate solutions.

Two main sources of error can be identified when solving a linear system of equations like:

$$A\,x = b \tag{95}$$

a) Both the coefficients of the matrix A and the right had side vector b are computed based on some physical properties of the problem under study. As a result, both A and b are subject to errors in their estimation. This means that in practice we are dealing with a system like:

$$(A + \delta A)\,x = (b + \delta b) \tag{96}$$

where:

δA : error in the estimation of the matrix of coefficients, which is another matrix of the same order of A.

δb : error in the estimation of b, which is another vector of the same order of b.

b) The second source of error appears even if we have exact estimates for A and b, and is due to rounding errors in the process of determining the vector x (Gaussian elimination or whatever method we are using).

Depending on the sensitivity of the solution for the unknowns x, either with respect to small changes in the coefficients of A and b or with respect to the solution process, we may be dealing with a well-conditioned problem (if the sensitivity is small) or with an ill-conditioned problem (if the system is highly sensitive to these changes). In the next sections we shall define more precisely the terms well-conditioned and ill-conditioned. In general, ill-conditioned will refer to systems which are close to being singular.

The main aim of this section is to find ways and means to assess if the computed solutions are accurate and, eventually, how to improve them.

ii) Effect of roundoff errors.

Let us assume that our estimation of A and b is exact. Then, we know that, because of roundoff errors, any computed solution is approximate, irrespective of whether it has been computed via a direct or an iterative method. The question to be resolved is how accurate is that computed solution.

Let $\hat{x}$ be the approximate computed solution of the system (95), then we can define the error of our computed solution as:

$$e = x - \hat{x} \tag{97}$$

Such an error cannot be computed directly from equation (97), since we do not know the value of the exact solution x.

If we premultiply our computed solution by the matrix A, we shall obtain a vector (i.e. A $\hat{x}$), which is not exactly the original right hand side vector b; the difference between both will be referred to as the residual r:

$$r = b - A \hat{x} \tag{98}$$

The residual can be directly calculated from the computed solution $\hat{x}$, and from the right hand side vector b (which is known).

Then, the residual tells us, in an indirect way, how far is the computed solution from the exact solution.

The error and the residual are closely related. In simple terms, it can be seen that if the residual is zero, then it means that our computed solution is exact and the error is zero as well. But the question still remains open in terms of whether a small residual is synonymous with a small error or not.

iii) An algorithm to compute the error: iterative improvement (iterative refinement).

To find a direct relationship between the error and the residual, we have to introduce equation (95) into equation (98):

$$r = b - A \hat{x} = A x - A \hat{x} \tag{99}$$

then, factorizing by A:

$$r = A (x - \hat{x}) \tag{100}$$

which, by equation (97) becomes:

$$r = A e \tag{101}$$

Hence, once we know the residual (r) produced by the computed value ($\hat{x}$), we can compute the error (e) by simply solving the following linear system:

$$A\ e = r \tag{102}$$

which is basically the same original system, but now with a different right hand side vector.

As we saw in previous sections, the solution of the linear system (102) can be obtained very easily if we have stored the original factorization of A from the solution for $\hat{x}$ and, in so doing, implies (n^2+n) multiplications/divisions plus (n^2-n) additions. Having calculated the error e, a new estimate of the vector x can be computed by:

$$x_{new} = \hat{x} + e \tag{103}$$

The process can stop here if $||e||/||x||$ is smaller than a pre-specified accuracy, otherwise a new residual can be computed assigning x_{new} to $\hat{x}$ and repeating the previous sequence from equation (98) up to equation (103).

This is the <u>iterative improvement</u> procedure and it should converge to the exact solution, provided that the matrix A is well-conditioned. A high rate of convergence in the iterative improvement procedure is a good indication of a well-conditioned system.

In the previous computations, since the residual (r) becomes very small, its computation normally involves the use of double-precision arithmetic.

iv) Estimating limits for the error.

Returning to the question of the relationship between residual and error, equation (102) can be written as:

$$e = A^{-1} r \tag{104}$$

This relationship can be understood in the sense that e is proportional to r, with a proportionality constant represented by A^{-1} ; unfortunately, we do not know A^{-1}. Furthermore, we do not have a way for measuring the "size" of a matrix, if this size is going to be used as a proportionality factor. The latter problem is solved in a similar way as in the case of defining the "size" of a vector, i.e. using the concept of "norm", but this time applied to a matrix:

$$|| A || = \max_{x} \frac{|| A x ||}{|| x ||} \tag{105}$$

where the maximum is considered over all non-zero (nx1) vectors and A is a (nxn) matrix.

Again, as in the case of vector norms, we can have different matrix norms: euclidean norm, maximum norm, etc.

On using the following property of norms:

$$|| A B || \leq || A || \cdot || B ||$$

and considering equation (104) we get:

$$|| e || \leq ||A^{-1}|| \; || r || \tag{106}$$

but, on applying the same to equation (101), we get:

$$|| r || \leq || A || \; || e || \tag{107}$$

which can be reordered as:

$$\frac{||\ r\ ||}{||\ A\ ||} \leq ||\ e\ || \qquad (108)$$

Hence, upon combining equations (108) and (106) we get:

$$\frac{||\ r\ ||}{||\ A\ ||} \leq ||\ e\ || \leq ||A^{-1}||\ ||\ r\ || \qquad (109)$$

From equation (109) we can obtain lower and upper bounds for the relative error ($||e||/||x||$) in terms of the relative residual ($||r||/||b||$) :

$$\frac{||\ b\ ||}{||\ A\ ||\ ||\ x\ ||}\ \frac{||\ r\ ||}{||\ b\ ||} \leq \frac{||\ e\ ||}{||\ x\ ||} \leq \frac{||A^{-1}||\ ||\ b\ ||}{||\ x\ ||}\ \frac{||\ r\ ||}{||\ b\ ||} \qquad (110)$$

But, from equation (109), for the particular case when $\underline{x}=\underline{0}$, where e = x and r = Ax = b, we get:

$$\frac{||\ b\ ||}{||\ A\ ||} \leq ||\ x\ || \leq ||A^{-1}||\ ||\ b\ ||. \qquad (111)$$

This means that, from the second part of (111):

$$\frac{||\ x\ ||}{||\ b\ ||} \leq ||A^{-1}|| \qquad (112)$$

and, from the first part of (111):

$$\frac{||\ b\ ||}{||\ x\ ||} \leq ||\ A\ || \qquad (113)$$

Introducing equations (112) and (113) into equation (110) gives:

$$\frac{1}{||\ A\ ||\ ||A^{-1}||}\ \frac{||\ r\ ||}{||\ b\ ||} \leq \frac{||\ e\ ||}{||\ x\ ||} \leq ||A^{-1}||\ ||\ A\ ||\ \frac{||\ r\ ||}{||\ b\ ||} \qquad (114)$$

Equation (114) defines the lower and upper limits for the relative error $||e||/||x||$.

The quantity $||A||\ ||A^{-1}||$ is defined as the condition number of the matrix A:

$$\text{Cond}\ (\ A\ ) \equiv\ ||A||\ ||A^{-1}|| \tag{115}$$

The condition number depends on the norm being used (euclidean, maximum, etc.), but it is always greater than 1.

For the particular case when Cond (A) = 1, equation (114) becomes:

$$\frac{||\ r\ ||}{||\ b\ ||} \leq \frac{||\ e\ ||}{||\ x\ ||} \leq \frac{||\ r\ ||}{||\ b\ ||} \tag{116}$$

which means that, in this case, the relative error is approximately the same as the relative residual. But, the greater the condition number, the less the information brought by the relative residual on the relative error.

In practice, the condition number is difficult to obtain, because the inverse matrix is usually not available.

The simplest estimate of $||e||$ is the scalar $||\hat{e}||$, where $\hat{e}$ is the solution of equation (102): $A\ e = r$.

An ill-conditioned system has a "large" condition number, where the value of "large" must be defined according to the particular computer being used. A large number is normally greater than 10^7 in single precision and greater than 10^{16} in double precision.

v) Effect of the errors in the matrix of coefficients.

Let us assume that the right hand side vector has been estimated accurately, then we are solving:

$$(A + \delta A) x = b \qquad (117)$$

where:

δA : matrix containing the error in the estimation of A.

Let:

$$\hat{A} = A + \delta A \qquad (118)$$

be the estimated A matrix, and $\hat{x}$ be the computed solution of equation (117), then:

$$\hat{A}\,\hat{x} = b \qquad (119)$$

From equation (95):

$$x = A^{-1} b \qquad (120)$$

and introducing equation (119), we get:

$$x = A^{-1} \hat{A}\,\hat{x} \qquad (121)$$

but

$$\hat{A} = A + \hat{A} - A \qquad (122)$$

and equation (121) becomes:

$$x = A^{-1} (A + \hat{A} - A)\,\hat{x} \qquad (123)$$

or

$$x = \hat{x} + A^{-1} (\hat{A} - A)\,\hat{x} \qquad (124)$$

recalling that $\delta A = \hat{A} - A$, from (118), and introducing it into equation (124), we get, after reordering:

$$x - \hat{x} = A^{-1} (\delta A)\,\hat{x} \,. \qquad (125)$$

Then, using the norm property: $||A\,B|| \leq ||A||\;||B||$, we get:

$$|| x - \hat{x} || \leq || A^{-1}||\;||\,\delta A\,||\;||\,\hat{x}|| \qquad (126)$$

but,

$$|| A^{-1}||\;||\,\delta A\,||\;||\,\hat{x}|| = || A^{-1}||\;||A||\;\frac{||\,\delta A\,||}{||\,A\,||}\;||\,\hat{x}||$$

and so, equation (126) becomes:

$$\frac{||\ x - \hat{x}\ ||}{||\hat{x}||} \leq ||\ A^{-1}||\ ||\ A\ || \frac{||\ \delta A\ ||}{||\ A\ ||} \qquad (127)$$

that is to say:

$$\frac{||\ x - \hat{x}\ ||}{||\hat{x}||} \leq \text{Cond}\ (A) \frac{||\ \delta A\ ||}{||\ A\ ||} \qquad (128)$$

Equation (128) establishes that the greater the condition number of A, the greater the effect of the relative errors of the coefficients of A in the relative accuracy of the solution.

In addition, from equation (128), we know that if our estimation of the matrix of coefficients is accurate just up to the s-th decimal, and if Cond (A) $\approx 10^t$ then, the maximum possible relative accuracy for x is 10^{t-s}; there is no point in trying to get a better solution because of the limited quality of the original data.

vi) Concluding remarks.

We have seen that computer solutions of linear systems of equations are *in essence approximate, independent of whether the* solution has been produced via a direct or an iterative algorithm.

The relative error depends on the condition number of the matrix of coefficients but, because condition numbers are difficult to estimate, a good and simpler approximation to the error is the norm $||\hat{e}||$, where $\hat{e}$ is the solution of A e = r.

An approximate solution $\hat{x}$ can be improved in its accuracy via the iterative improvement algorithm, which is not too computationally expensive if the factorization of A has been previously saved. There is a limit in the accuracy of $\hat{x}$ in the iterative improvement stage, since we cannot expect an estimate $\hat{x}$ more accurate than that allowed by the quality of the data.

A.3. Review of sparse direct methods for the solution of linear systems of equations.

A.3.1. Data structures.

As far as many engineering problems are concerned, the corresponding matrices generated in their solution are such that, even though their order may be in the range of hundreds or thousands, the number of non-zero elements is relatively small in comparison with the total number of elements. This is also the case in water distribution network analysis, where typically the matrices generated by the different algorithms have at most 5-6 non-zero elements per row, irrespective of the size of the network. In practice, this means that for a network of, say, 1,000 nodes, i.e. producing a matrix with 1,000,000 elements, only 5,000-6,000 of these elements are actually non-zero. All these kinds of matrices are known as sparse matrices, where the relatively few non-zeros are scattered throughout the matrix.

Obviously, important savings of computer storage and execution time can be obtained by explicitly taking into account the sparsity of the matrix, since it is rather inefficient to store zeros and carry out additions or multiplications by zero. The

larger the size of the network, the bigger the relative savings of computer resources. In addition to that, some sparse matrices have special structures, like symmetric matrices, banded matrices or tridiagonal matrices; by explicitly taking into account the special structure of a matrix we can reduce even further the requirements of storage and computer time.

To take full advantage of the presence of a great deal of zeros in the sparse matrix, we need to store and operate only on the non-zeros and, to do so, we need some efficient way for keeping track of the position of all the non-zeros within the full matrix and, of course, of its numerical value. Roughly speaking, this means that we need to store the "coordinates" corresponding to the position of each non-zero within the matrix, which is normally dealt with by "cheap storage", say Integer*2 in a FORTRAN code, while the numerical value is kept in more lengthy storage, say Real*4 or Real*8 in a FORTRAN program.

There are a number of schemes for accessing the non-zeros in a sparse matrix, with a very different complexity/efficiency ratio. Simple data structures lead to relatively low efficiency schemes, while more complex data structures lead to the more efficient schemes.

Duff et al. (1986), Pissanetzky (1984) and George and Liu (1981), amongst others, give a detailed description of different data structures for sparse matrices; we are going to describe only some of them, especially those used in this Appendix. Since the matrices generated in water distribution network analysis are

symmetric positive definite matrices, we shall look carefully in that direction, although most of the topics covered in the next sections are applicable to more general matrices as well.

a) Coordinate scheme.

In this case we use two integer arrays to store the row and column indices of each non-zero, and a real number to store its non-zero value. The length of all these arrays is the same and equal to the total number of non-zeros within the matrix, say NZ. This data structure allows us to store the non-zeros in any order, and it can be be used straight-away for the case of non-symmetric matrices.

The retrieval on the non-zero elements from the data structure is quite simple; the following example illustrates this point:

Example: Let a lower triangular matrix L of order N = 7 and 16 non-zeros (NZ = 16) be as follows:

$$L = \begin{bmatrix} l_{11} & & & & & & 0 \\ l_{21} & l_{22} & & & & & \\ & & l_{33} & & & & \\ l_{41} & l_{42} & & l_{44} & & & \\ & & l_{53} & l_{54} & l_{55} & & \\ & & l_{63} & & l_{65} & l_{66} & \\ & & & & l_{75} & l_{76} & l_{77} \end{bmatrix}$$

then, the coordinate storage scheme, storing the elements by rows, would be:

Row index: IROW

IROW = (1, 2, 2, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7)

Column index: ICOL

ICOL = (1, 1, 2, 3, 1, 2, 4, 3, 4, 5, 3, 5, 6, 5, 6, 7)

Non-zero values array: VALUE

VALUE= ($l_{11}, l_{21}, l_{22}, l_{33}, l_{41}, l_{42}, l_{44}, l_{53}, l_{54}, l_{55}, l_{63}, l_{65}, l_{66}, l_{75}, l_{76}, l_{77}$)

The length of each array is 16 (= NZ).

b) Implicit storage scheme.

A more elaborate scheme arises from observing that some further storage reduction can be achieved from the previous scheme. This can be done by avoiding the repetition of the row indices (a column oriented implicit scheme is also possible) and storing them in an implicit indexing scheme. The previous row index vector of length NZ is replaced by a new row index vector of length N+1, where N is the order of the matrix; the new row index points to the position of the first non-zero element in each row, which in our previous example means:

New row index: IROWI

IROWI= (1 , 2 , 4 , 5 , 8 , 11 , 14 , 17)

ICOL = (1, 1, 2, 3, 1, 2, 4, 3, 4, 5, 3, 5, 6, 5, 6, 7)

VALUE= ($l_{11}, l_{21}, l_{22}, l_{33}, l_{41}, l_{42}, l_{44}, l_{53}, l_{54}, l_{55}, l_{63}, l_{65}, l_{66}, l_{75}, l_{76}, l_{77}$)

The column and non-zero array are the same as before.

Notice that since the difference between successive elements of the vector IROWI gives the number of non-zeros per row, we then need the element of order N+1 for completeness. Also note that IROWI(N+1)-1 corresponds to the amount of non-zero elements (NZ).

The column-wise implicit scheme is straightforward. A further improvement of this scheme can be obtained by storing the diagonal non-zeros in a separate array, thus reducing the length of the column index array to NZ-N, and keeping the rest of the storage unchanged.

c) Compressed scheme.

One of the most efficient schemes is due to Sherman (1975) which takes advantage of the fact that, in the previous row-wise implicit scheme, the column index can be compressed even further by adding a new integer index of length N (integer array NZSUB). In the case of a column-wise implicit scheme it is the row index that can be compressed further.

In our previous example, presented by George (1981), this scheme is used for storing a lower triangular matrix by columns (the row-wise version is straightforward) :

Diagonal vector: DIAG

DIAG = ($l_{11}, l_{22}, l_{33}, l_{44}, l_{55}, l_{66}, l_{77}$)

Non-diagonal non-zero vector: LNZ

LNZ = ($l_{21}, l_{41}, l_{42}, l_{53}, l_{63}, l_{54}, l_{65}, l_{75}, l_{76}$)

XLNZ = (1, 3, 4, 6, 7, 9, 10)

NZSUB = (2, 4, 5, 6, 5, 6, 7)

XNZSUB = (1, 2, 3, 5, 6, 7, 8)

The compressed scheme splits the non-zero values into two real arrays, one for the diagonal elements DIAG (neither row nor column indices are needed for these elements) and another LNZ for the non-diagonal non-zeros.

The array XLNZ gives the amount of non-zeros per column (excluding the diagonal) and NZSUB gives the corresponding row index of these non-zeros using XNZSUB as a pointer to the first non-zero in NZSUB, as illustrated by the arrows in the previous structure. The extra savings produced by this scheme are due to the fact that in the array NZSUB some elements play a double (or even greater !!) role, like the 4 and 7 in the *second and seventh* position of NZSUB respectively.

The advantages of this more complex scheme become apparent in large systems, of the order of hundreds or thousands of equations.

A.3.2. Fill-in.

On using a direct method for the solution of a general linear system, we already know that the elimination process consists of transforming the original matrix into an upper (or lower) triangular matrix, which is subsequently solved via a backward (or forward) substitution. The fact that the original matrix is sparse does not generally imply that the transformed triangular matrix has the same sparsity pattern; in fact, it may not be sparse at all. Due to the elementary operations performed between the rows (or columns) of the original matrix, new non-zeros can be produced in places where a zero element existed. The newly created non-zeros are referred to as fill-in (or simply fill).

One of the worst cases of fill-in that can be expected is shown in Figures A.2 and A.3. The original system has the arrow shaped matrix shown in Fig. A.2.

If we factorize A into the Cholesky factors $L\,L^T$, then L has the structure shown in Fig. A.3, which corresponds to a lower triangular fully *populated (dense)* matrix.

$$\begin{bmatrix} x & x & x & x & x \\ x & x & & & \\ x & & x & & \\ x & & & x & \\ x & & & & x \end{bmatrix} \cdot \begin{bmatrix} x \\ x \\ x \\ x \\ x \end{bmatrix} = \begin{bmatrix} x \\ x \\ x \\ x \\ x \end{bmatrix}$$

$$A \quad . \; x \; = \; b$$

Fig. A.2. Arrow shaped linear system.

$$L = \begin{bmatrix} x & & & & \\ x & x & & & \\ x & x & x & & \\ x & x & x & x & \\ x & x & x & x & x \end{bmatrix}$$

Fig. A.3. Cholesky factor of arrow shaped matrix (without reordering).

However, if we permute rows and columns 1 and 5 in the original matrix A, producing the reordered matrix A^* shown in Fig. A.4,

$$\begin{bmatrix} x & & & & x \\ & x & & & x \\ & & x & & x \\ & & & x & x \\ x & x & x & x & x \end{bmatrix} \cdot \begin{bmatrix} x \\ x \\ x \\ x \\ x \end{bmatrix} = \begin{bmatrix} x \\ x \\ x \\ x \\ x \end{bmatrix}$$

$$A^* \quad . \; x \; = \; b^*$$

Fig. A.4. Reordered arrow shaped linear system.

The corresponding new Cholesky factor L^* will have the sparsity pattern shown in Figure A.5.

$$L^* = \begin{bmatrix} x & & & & \\ & x & & & \\ & & x & & \\ & & & x & \\ x & x & x & x & x \end{bmatrix}$$

Fig. A.5. Cholesky factor of reordered arrow-shaped linear system.

Comparing A^* with its factor L^*, we can see that no fill-in has been produced.

This extreme example illustrates very clearly the main feature on which most of the sparse techniques are based, namely, the fact that fill occurs only because of the way in which the original matrix has been set up; consequently, fill can be reduced and minimized via a sensible reordering of the original system. Thus, one of the first tasks to be tackled in sparse matrix handling is the search for an efficient ordering scheme that can reduce the amount of fill.

Formally, the reordering process can be understood as applying some permutation matrix P to the original linear system. That is to say, instead of solving:

$$A\,x = b \tag{95}$$

we solve:

$$(\,P\,A\,P^T\,)\,P\,x = P\,b \tag{96}$$

This is because a permutation matrix P is also an orthogonal matrix, i.e.:

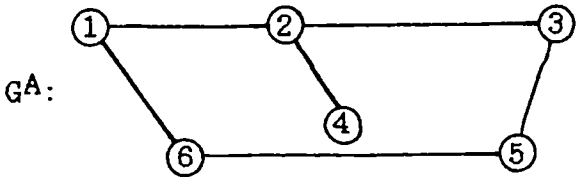
$$P\,P^T = P^T\,P = I \qquad \text{(identity matrix)} \tag{97}$$

In this context, finding an adequate reordering strategy can be equivalent to find the matrix P which minimizes the fill in the matrix $P\ A\ P^T$.

Graph theory provides a good representation of the reordering process; for example, for the symmetric matrix given by equation (98):

$$A = \begin{bmatrix} 1 & 7 & & & & 8 \\ 7 & 2 & 9 & 10 & & \\ & 9 & 3 & & 11 & \\ & 10 & & 4 & & \\ & & 11 & & 5 & 12 \\ 8 & & & & 12 & 6 \end{bmatrix} \qquad (98)$$

the associated graph representation is shown in Fig. A.6. A graph associated with a matrix is the set $G^A = (X,E)$, where X is the set of nodes (one node per row or column) and E is the set of edges (one per each non-diagonal non-zero).



$X = \{1,2,3,4,5,6\}$

$E = \{(6,1),(1,2),(2,4),(2,3),(3,5),(5,6)\}$

Fig. A.6. Undirected graph corresponding to symmetric matrix given by equation (98).

Although graph theory concepts are very useful to understand some aspects of sparse matrices, we shall try to keep its use at a minimum level, to avoid further complexities. Any definition will be given immediately before it is needed; for details see either Wilson (1985), Deo (1974), Harary (1972) or Harris (1970).

Note that for non-symmetric matrices, directed graphs (i.e. graphs whose edges have a direction associated with them) are needed to account for the non-symmetric connectivity.

To permute the 3^{rd} and 4^{th} rows and columns of A is equivalent to transforming A into ($P\ A\ P^T$), with the permutation matrix shown in Fig. A.7. which is an identity matrix with its 3^{rd} and 4^{th} rows interchanged.

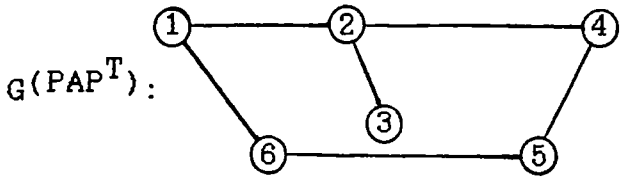
$$P = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 0 & 1 & & \\ & & 1 & 0 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}$$

Fig. A.7. Permutation matrix for interchanging rows 3 and 4.

On performing the pre and post-multiplication of A by P and P^T, we get the matrix and associated graph shown in Fig. A.8.

$$P\ A\ P^T = \begin{bmatrix} 1 & 7 & & & & 8 \\ 7 & 2 & 10 & 9 & & \\ & 10 & 4 & & & \\ & 9 & & 3 & 11 & \\ & & & 11 & 5 & 12 \\ 8 & & & & 12 & 6 \end{bmatrix}$$

a)



b)

Fig. A.8. Reordered matrix [eq.(98)] and its associated graph.

Thus, in graph terms, we can see that reordering does not mean a change in the structure of the matrix graph, but only changes in its labelling ; in the previous example this swap of labels happens between nodes 3 and 4.

From the previous example it is also evident that if A is symmetric, then P A P^T is symmetric as well; this is due to the fact that the premultiplication by P interchanges the rows, while the postmultiplication by P^T swaps the columns.

Another important consideration apparent from the last example is that for finding a "sensible" reordering (or re-labelling in the graph of the original system), we do not actually need its numerical values, but only its structure. This is a very important feature, since it means that we can split the Gaussian elimination problem into a sequence of independent stages. This is true for symmetric positive definite matrices only. Usually the whole process is separated into the following stages:

i) *ANALYSIS stage: this finds the minimum fill-in* ordering and sets up the corresponding data structures.

ii) FACTORIZATION stage: where the numerical part of the factorization of A is carried out.

iii) SOLUTION stage: where the solution of the system takes place, for a given right hand side vector.

This three-stage process allows us to make more efficient the overall solution process, since if, for example, we need to solve a linear system for different right hand side vectors, then we only need to repeat the last SOLUTION stage, provided that we have stored the information from the ANALYSIS/FACTORIZATION stages. At the same time, if only the numerical values of the matrix are changed (but not its structure), then we need to repeat the FACTORIZATION and SOLUTION stages. This maximizes the

performance of the Gaussian elimination process, since it is quite usual that the ANALYSIS needs to be performed only once, while the remaining stages can be repeated as many times as needed, using the same ANALYSIS results.

As pointed out by many researchers in this area: Irons (1970), Reid (1977) and George (1981), the separation of the whole process into 3 independent stages is only suitable for symmetric positive definite matrices, where no risk of instability occurs. That is to say, in symmetric positive definite matrices we do not need to pivot for stability and then, we can concentrate our attention in reducing the fill-in. For more general sparse matrices, permutations are needed for stability reasons and then, the data structures are dependent on the numerical values and not only on the structure (or connectivity) of the matrix. This is clearly a conceptual problem, with very important practical consequences.

As a result, we have seen that the amount of fill produced in the Gaussian elimination process depends on the way we set up and reorder the original matrix. The problem is, as far as reducing the fill is concerned, to find an adequate reordering scheme that minimizes the amount of fill (if such a reordering exists). It must be emphasised that normally minimizing the amount of fill not only leads to a reduction in the storage requirements, but also to a reduction in the amount of computational work to be done.

A.3.3. Sparse methods for banded matrices.

Historically, banded matrices such as those arising from many differential equation problems, have been the starting point of the development of sparse matrix techniques. Even if the original linear system does not emerge in a banded form naturally, the aim has been to reorder it in such a way that the reordered system could have all its non-zeros "clustered" around the diagonal.

The key observation in sparse banded methods is that when factorizing a banded matrix, the eventual fill is concentrated within the band; thus, all the zeros outside the band can be ignored (or "exploited" in the sense of sparse matrix handling), and only the elements within the band are stored, irrespective of the fact that some of them may be zeros.

The storage scheme used for dealing with banded matrices has not been explained before, but simply reshapes the sub-diagonals of the banded matrix into vertical columns as shown in Fig. A.9.

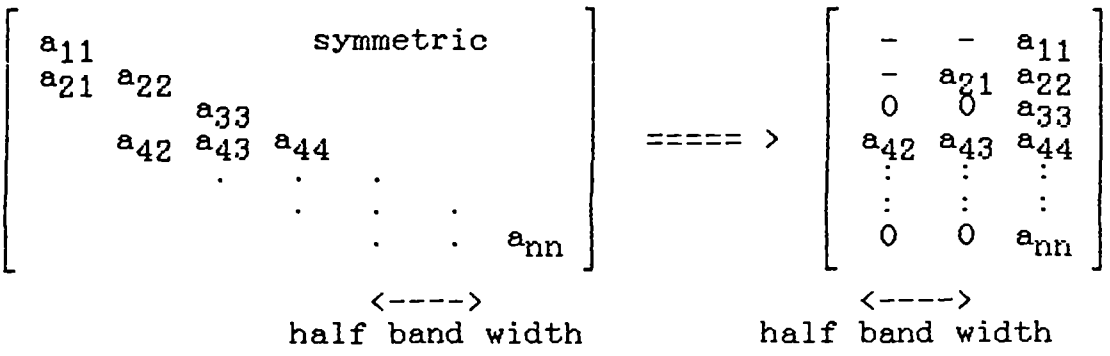


Fig. A.9. Diagonal storage scheme for banded matrices.

The main advantage of this approach is its simplicity, both in terms of the data structures and in terms of programming. Of course, it is evident from Fig. A.9., that it is a sub-optimal

solution, in the sense that some fill-in can still be exploited with very little extra effort. This is done in the next method.

Additionally, the lack of optimality of banded methods is due to other reasons, as can be shown using the same arrow shaped matrix already seen in Fig. A.2, where a band minimization approach produces the matrix ($P\ A\ P^T$) shown in Fig. A.10.

$$P\ A\ P^T = \begin{bmatrix} x & & & x & & \\ & x & x & & & \\ x & x & x & x & x & \\ & & x & x & & \\ & & x & & x & \end{bmatrix}$$

Fig. A.10. Minimum band ordering for arrow shaped linear system.

Clearly, the ordering shown in Fig. A.10 is poorer than that shown in Fig. A.4, where the permutation of rows and columns 1 and 5 produced no fill-in at all in the Cholesky factorization stage.

Cuthill and McKee (1969) developed an algorithm for producing minimum bandwidth orderings. The algorithm needs a starting or "root" node (r), and the original graph is relabelled starting from "r" and renumbering first those nodes with the minimum number of edges connected to them (i.e. minimum degree nodes). The example shown in Fig. A.11, taken from George (1981) illustrates how the algorithm works.

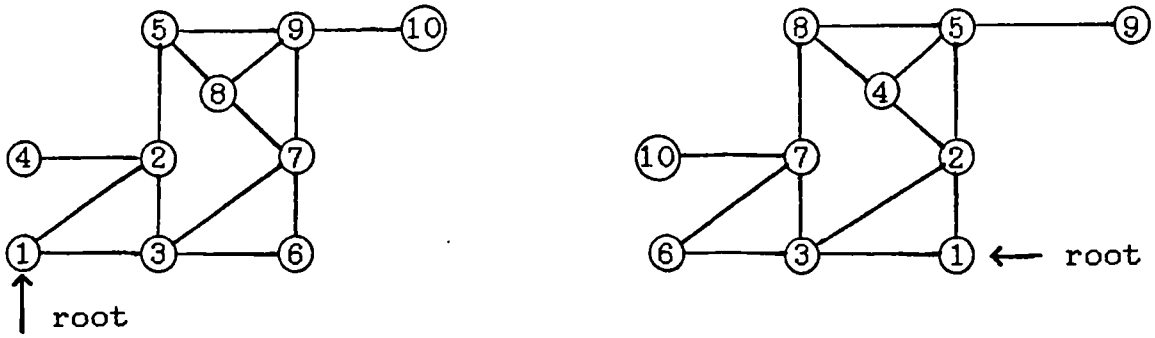


Fig. A.11. Two examples of Cuthill-McKee orderings for reducing bandwidth, from George (1981).

Two important features of the algorithm become apparent from this example:

a) The ordering (or labelling) depends on the selected root node.

b) Even for the same root node, different orderings can be obtained when the neighbouring unnumbered nodes have the same degree (like when numbering nodes 2 and 3 in Fig. A.11).

The classical example of the arrow shaped matrix showed quite clearly that minimizing the bandwidth has nothing to do with minimizing the fill-in (and nothing to do with minimizing the amount of work to be done). As a result, the method is sub-optimal and because the extra effort needed for a further improvement is small, the band methods are not widely used today, except for problems where the banded structure comes up naturally (e.g. tridiagonal matrices and similar).

A.3.4. The envelope ("skyline") method.

The key observation leading to the envelope method is that fill-in actually occurs only in those positions between the first

non-zero in each row and the diagonal of the matrix, that is to say, within the "skyline"-shaped bordering line which surrounds all the non-diagonal non-zeros in the matrix. Figure A.12 shows this feature, by shading the location of possible fill-in.

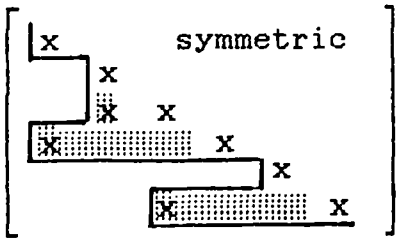


Fig. A.12. Skyline representation.

In this example, fill in may only occur in the fourth and sixth rows, but never outside of the envelope (shaded area).

The number of non-zero elements inside the envelope is usually referred to as the profile of the matrix; thus, in the previous symmetric example [equation (98)], the profile is equal to 7.

In this context, orderings minimizing the profile of a matrix lead to low fill, and algorithms producing minimum profile are efficient from the storage point of view. One of the most widely used profile reduction algorithms was proposed by George (1974), who found that by simply reversing the ordering produced by the Cuthill-McKee algorithm eventually reduces the profile (in fact it does not make it worse), and the algorithm has been called "Reverse Cuthill-McKee" algorithm.

As pointed out in the case of the standard Cuthill-McKee algorithm, the ordering depends on the selection of the root node. Obviously the same happens in the Reverse Cuthill-McKee

(or RCM) algorithm. Hence, we need to find a root node useful for our purposes.

It has been empirically found that good root nodes are those lying on the periphery of the graph (peripheral nodes); these are nodes such that the amount of edges between them and any other node in the graph (which is defined as the "eccentricity" of the node) is maximum.

George and Liu (1981) proposed the following algorithm for finding peripheral nodes; because the algorithm does not guarantee peripheral nodes, the nodes found by it are referred to as "pseudo-peripheral" nodes.

The algorithm relies on the concept of rooted level structure, which is explained immediately. Following the George and Liu (1981) definition, a level structure rooted at node "x" is a partitioning $\mathcal{L}(x)$ of the set of nodes of the matrix graph, such that:

$$\mathcal{L}(x) = \{ L_0(x), L_1(x), \ldots , L_{l(x)}(x) \}$$

where:

x : is the root node of the level structure.

l(x) : is the eccentricity of node "x".

Rooted at "x", the following levels are defined, each one containing the adjacent nodes not previously included in a level:

$L_0(x) = \{ x \}$

$L_1(x) = \{$ Nodes adjacent to $L_0(x) \}$

$\vdots$

$L_i(x) = \{$ Nodes adjacent to $L_{i-1}(x)$, except those in $L_{i-2}(x) \}$
$\forall\ i=2,3, \ldots , l(x)$

Figure A.14 is an example of a rooted level structure, corresponding to the matrix whose graph is shown in Fig. A.13

In the level structure shown in Fig. 14 $\mathcal{L}(x_5)$ the eccentricity of x_5 is $l_{(x5)} = 3$ and

$$\mathcal{L}(x_5) = \{ L_0(x_5),\ L_1(x_5),\ L_2(x_5), L_3(x_5) \}$$

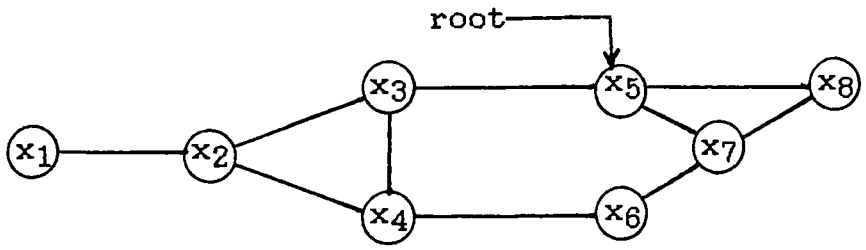
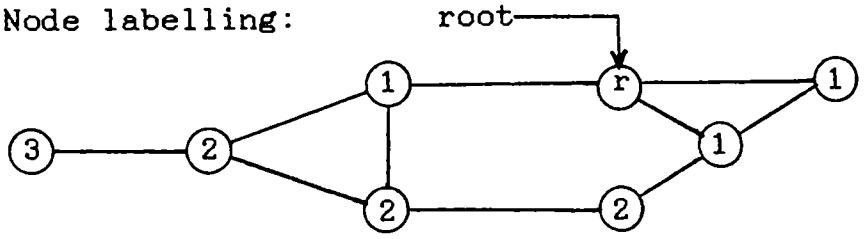
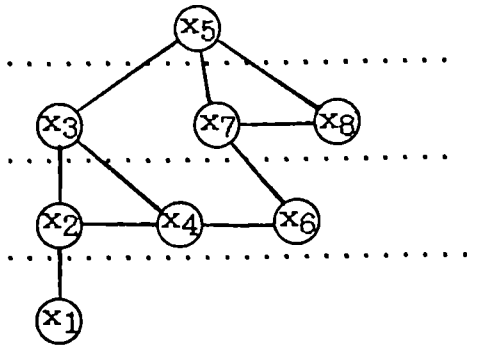


Fig. A.13. Graph for level structure example, from George and Liu (1981).

a) Node labelling:



b) Level structure:



$L_0(x_5) = \{ x_5 \}$

$L_1(x_5) = \{ x_3,\ x_7,\ x_8 \}$

$L_2(x_5) = \{ x_2,\ x_4,\ x_6 \}$

$L_3(x_5) = \{ x_1 \}$

Fig. A.14. Level structure rooted at x_5, for graph shown in Fig. A.13, from George and Liu (1981).

The algorithm proposed by George and Liu (1981) for finding pseudo-peripheral nodes is reported to be based on a previous one, proposed by Gibbs et al. (1976) and its steps are:

a) Pick up an arbitrary root node "r".
b) Generate the level structure rooted at node "r".
c) Choose a node in the last level $L_{l(r)}(r)$ with the minimum possible degree, say node "x".
d) Generate another level structure now rooted at "x".

If the eccentricity of "x" is greater than the eccentricity of "r", then assign r <-- x and repeat the process from step c), otherwise:

e) "x" is a pseudo-peripheral node and stop.

For the previous example, the process of finding a pseudo-peripheral node can be represented as in Fig. A.15.

Notice that in the previous example, node x_1 can be labelled as a pseudo-peripheral node as well.

On using the previous ideas, the following is the sequence for the Reverse Cuthill-McKee algorithm:

Step 1) Determine a pseudo-peripheral node "r" and label it as x_1. This can be done via the algorithm already introduced or another one.

Step 2) Loop : label the unlabelled neighbours of x_i in an increasing order of degree, $\forall$ i=1,2, ... , N. This is the standard Cuthill-McKee algorithm.

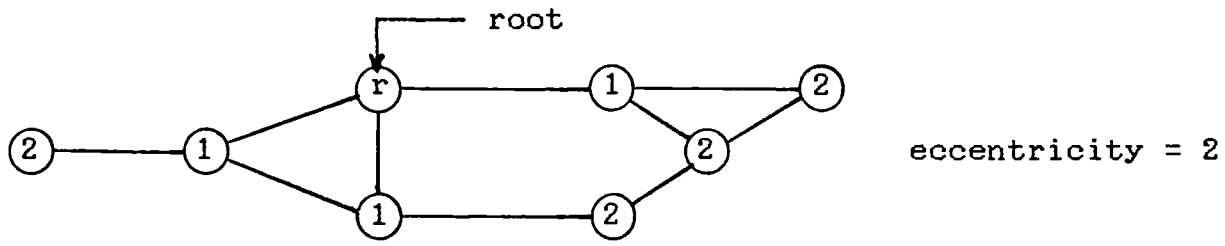
Step 3) Reverse the labelling: producing a new ordering

(labelling), say:

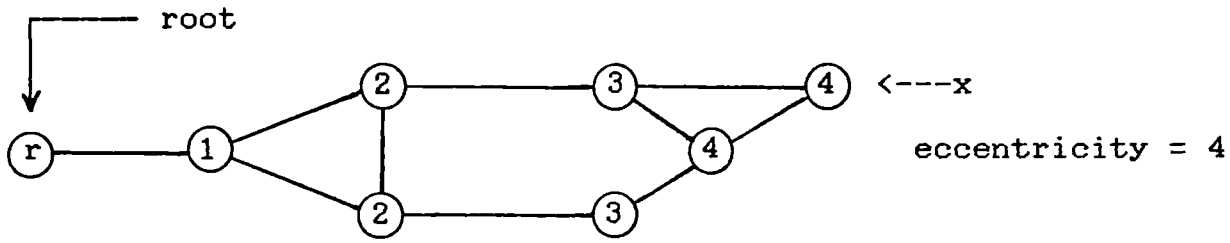
$y_1, y_2, \ldots, y_N$ such that $y_i = x_{(N-i+1)}$ $\forall$ i=1,2,..,N

As we have already pointed out, step 1 can be replaced by any other algorithm or, indeed, the root node can be specified by the user. In the context of water distribution networks which are fed gravitationally, a good candidate can be the node corresponding to the upstream reservoir in the system. In more general networks, with supply coming from gravitational and pumped sources, the previous choice might not be the best one.

First try: steps a) and b), $r = x_3$



Second try: steps c) and d), $x = x_1$



Third try: r <-- x and back to steps c) and d)

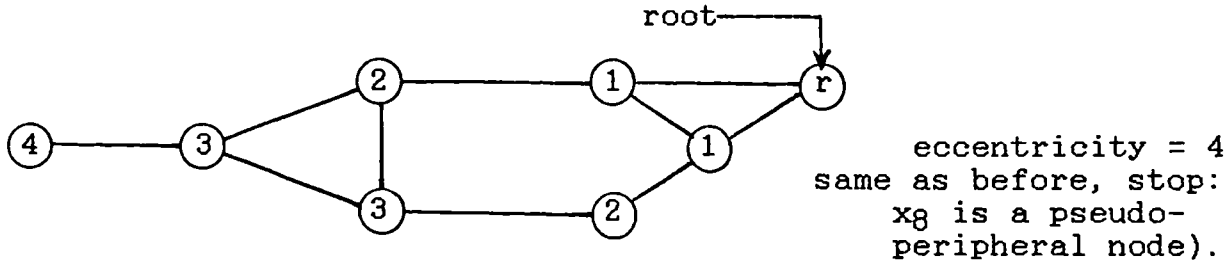


Fig. A.15. Algorithm for finding a pseudo-peripheral node, from George and Liu (1981).

We have implemented the envelope method, with the RCM ordering, using the subroutines published by George and Liu (1981). We used this implementation for solving the linear systems generated in water distribution analysis. We would like to emphasise here that one of the main drawbacks found, deals with the amount of storage needed for the non-zeros out of the diagonal (array ENV). The real storage required is not predictable before running the program and an upper bound has to be defined on a trial-and-error basis using the program with problems of different sizes (although this procedure does not guarantee an upper limit); we found, using synthetic networks of up to 5000 nodes (≈ 5000 equations), that the maximum length of the array ENV tends to $N^{1.5}$. Also, the execution time obtained with this method is of the order of 3 times the corresponding time of the efficient methods. For this reason we do not elaborate more on this algorithm.

A.3.5. Minimum degree algorithms.

One of the main shortcomings of the previous envelope method was its sub-optimality in terms of storage; in fact a great deal of the storage for the non-zeros enveloped within the skyline was indeed occupied by zeros, and the idea is now try to exploit all the zeros. The price to be paid is obviously further complications in data structures, algorithms and programming.

Algorithms based on the minimum degree concept have been used since Markowitz (1957), in linear programming applications, and subsequently by Tinney and Walker (1967), Ogbuobiri, Tinney and

Walker (1970) and Ogbuobiri (1970), in power networks.

Zollenkopf (1970) used a minimum degree approach for pivot selection within a sparse factorization method for symmetric positive definite matrices in power systems. Pivoting is apparently needed in this case because the author considered also the case when the matrix is asymmetrical in the element values (but remaining symmetric in its structure).

Shamir (1973) applied the concepts of minimum degree to water distribution networks, though he used a mixed strategy, pivoting both for size (partial pivoting) and density (to reduce fill-in).

Nowadays it is understood that, for symmetric positive definite linear systems, pivoting is not necessary to ensure stability. As we said earlier, this means that in symmetric positive definite matrices, we can concentrate in the reduction of fill-in exclusively, without risking instabilities. Thus, the ANALYSIS stage does not include the numerical values of the non-zeros, but only their structure. Recall that for general sparse matrices this is not valid, and pivoting for stability is essential.

As we saw earlier, pivoting for stability meant that during the Gaussian elimination process, the next pivot element was chosen from the row which had the biggest diagonal absolute value (partial pivoting). We also mentioned full pivoting as another alternative.

Minimum degree consists basically in choosing the next pivot element, at each stage of the elimination sequence, from the row

with the fewest amount of non-zero elements.

It has to be said that when pursued simultaneously, pivoting for density and pivoting for stability are rather conflicting objectives, but this is not our problem in symmetric positive definite matrices.

An example of the use of the minimum degree algorithm is presented in Figure A.16, which has been taken from George (1981).

In its traditional implementation, which stores the Cholesky factors explicitly, the minimum degree algorithm generates a great deal of fill, requiring important quantities of storage; unfortunately, to make things worst, it happens that the amount of required storage is unpredictable until the factorization is completed. Thus, when using this implementation the user does not know how much storage will be needed, and the only way to tackle this problem is via experimentation.

George and Liu (1981) published a solution to this problem, using an implicit storage scheme, which has a predictable storage at the ANALYSIS stage. Thus, the computer program can determine, before the numerical part of the solution is started, what is the amount of storage required. A so-called fast hybrid implementation, which lies in between the explicit and implicit storage scheme, was compared by George and Liu (1981), chapter 9, and was found to be one of the best available from the point of view of the execution time; that comparison was carried out with problems of up to 2233 equations arising from finite element

problems.

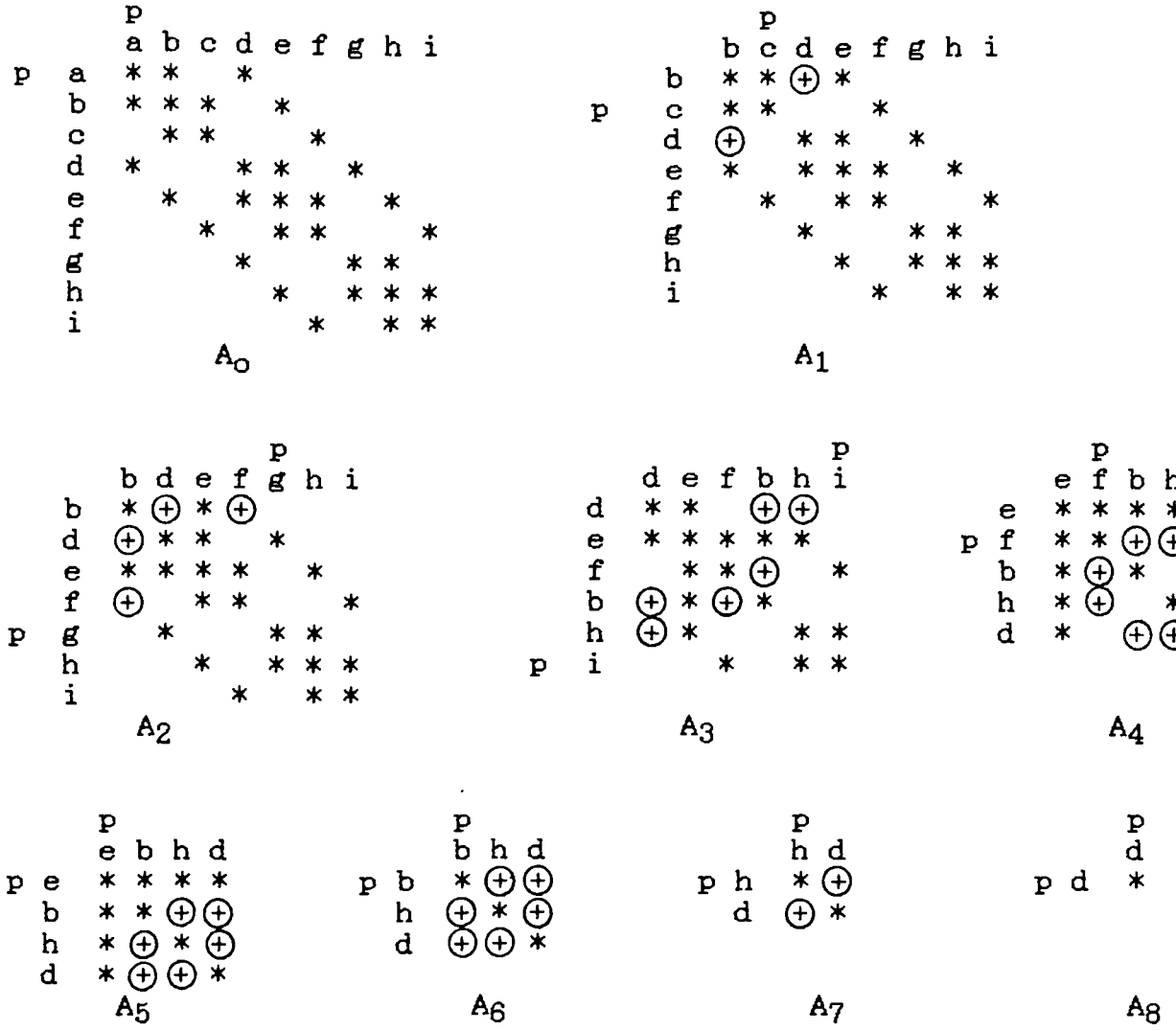
Since we have used this implementation in our comparisons in the context of water distribution networks, we shall spend the rest of this section explaining, in the simplest possible terms, how this scheme works; for a more detailed explanation, see George and Liu (1981, chapter 5). The implementation relies heavily on graph theory concepts, especially in what is called "quotient graph", which is also the base for other forthcoming methods reviewed in this section.

Before explaining what is the fast hybrid implementation of George and Liu, we need to explain what is the implicit scheme. To do so, we shall introduce graphs into this problem, in order to visualise the process of creation of fill-in. The same problem of Gaussian elimination shown in Fig. A.16, can be represented using graphs as in Figure A.17.

Figure A.17 is self-explanatory and shows how the elimination sequence, using the minimum degree algorithm, takes place. The order of elimination is then:

(a , c , g , i , f , e , b , h , d)

In graph terms, fill-in, i.e. the creation of new darker edges in Fig. A.17, can be explained as follows: let us consider for example the elimination of "a" in G^0 (see Fig. A.17), since "d" is "reachable" from "b" through "a". Then, in order to maintain the same "reachability" of the original graph G^0, after the elimination of "a", we need to add a new edge which accomplishes this purpose, otherwise the "reachability" or connectivity



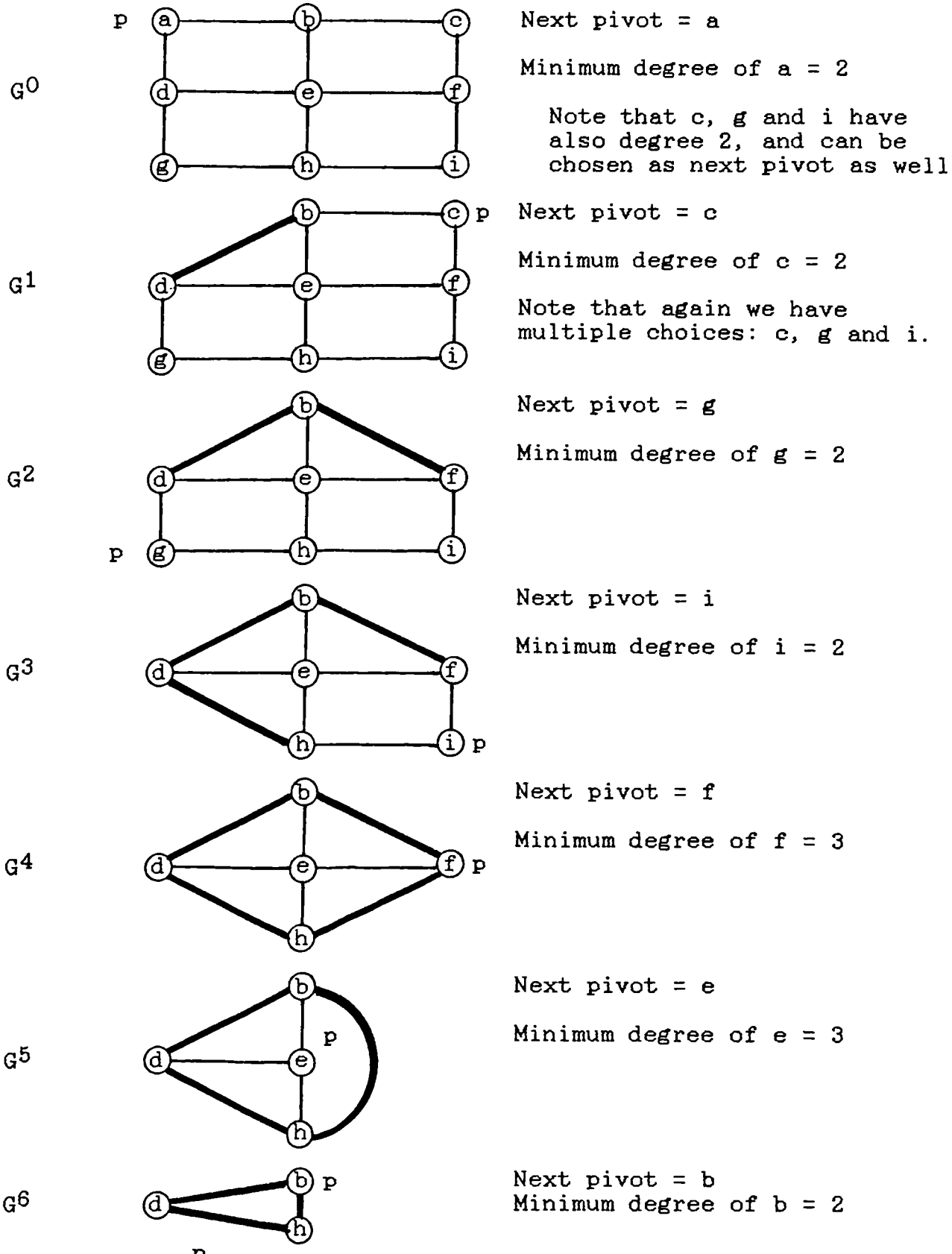
Notes:
*****

(1) "p" marks the new pivot.

(2) ⊕ denotes fill-in.

(3): the pivots are always swapped with the first row (and column), and then eliminated from the following graph.

Fig. A.16. Example of application of minimum degree algorithm, from George (1981).



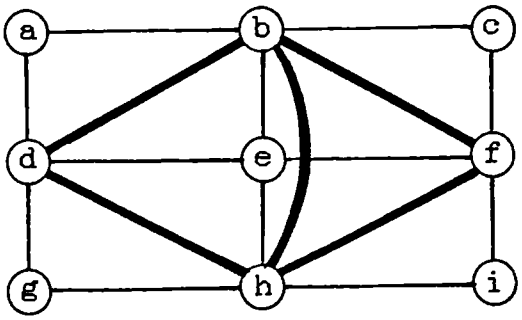
Notes: (1) The darker lines are created by the elimination process and represent the new non-zeros in the factorization.
(2) "p" marks the chosen pivot element.

Fig. A.17. Elimination graphs representing the process of creation of fill-in.

information contained in G^0 will be lost. On using the same idea throughout the whole elimination process, the new graph corresponding to the filled matrix of the original linear system becomes that shown in Fig. A.18.

G^F :

$(F=L+L^T)$



Note: darker lines indicate that fill-in will occur in the original matrix, when the factorization is carried out following a minimum degree algorithm.

Fig. A.18. The filled graph of $F = L + L^T$.

The corresponding filled (reordered) matrix is shown in Fig. A.19, where an "plus" (+) sign denotes fill, which corresponds to the darker lines in Fig. A.18.

	a	c	g	i	f	e	b	h	d
a	*						*		*
c		*			*		*		
g			*					*	*
i				*	*			*	
f		*		*	*	*	⊕	⊕	
e					*	*	*	*	*
b	*	*			⊕	*	*	⊕	⊕
h			*	*	⊕	*	⊕	*	⊕
d	*		*			*	⊕	⊕	*

Fig. A.19. Filled (reordered) matrix.

Notice that G^F (Fig. A.18) has been found using only the information on the matrix structure, the numerical values not having been used at all.

According to Fig. A.17, the whole elimination can be represented as a succession of graphs: G^0, G^1, G^2, ... , G^8. This is an <u>explicit</u> representation.

The <u>implicit</u> scheme arises on formalising the "reachability" concept used to explain the development of Fig. A.17. Following George and Liu (1981), for a graph G = (X, E) and S being a subset of the nodal set X and taking $x \notin S$, then x is said to be reachable from another node "y" if there is a path through S connecting x and y (i.e. a series of nodes in between them). That path is formally written as:

(y, μ_1, μ_2, ..., μ_k, x / from y to x such that $\mu_i \in S$, $1 \leq i \leq k$ with $k \geq 0$)

Note that a neighbouring node of y not in S is also reachable from y (with k=0).

The set of all the reachable nodes from y through S is defined as:

Reach(y,S) = { $x \notin S$ such that x is reachable from y through S }

The practical application of the reach operator, as defined above, becomes clear if we define S as a variable set in the elimination process represented by Fig. A.17, which changes at each stage of the elimination process according to:

$$S_i = \{ x_1, x_2, \ldots , x_i \}$$

where x_i represent the nodes in the order in which they are eliminated.

Hence, for the elimination process shown in Fig. A.17 we have:

$$
\begin{array}{lll}
\text{Reach}(a, S_o) = \{\ b,\ d\ \} & , & S_o = \{\ \emptyset\ \} \\
\text{Reach}(c, S_1) = \{\ b,\ f\ \} & , & S_1 = \{\ a\ \} \\
\text{Reach}(g, S_2) = \{\ h,\ d\ \} & , & S_2 = \{\ a,\ c\ \} \\
\text{Reach}(i, S_3) = \{\ f,\ h\ \} & , & S_3 = \{\ a,\ c,\ g\ \} \\
\text{Reach}(f, S_4) = \{\ e,\ b,\ h\ \} & , & S_4 = \{\ a,\ c,\ g,\ i\ \} \\
\text{Reach}(e, S_5) = \{\ b,\ h, d\ \} & , & S_5 = \{\ a,\ c,\ g,\ i,\ f\ \} \\
\text{Reach}(b, S_6) = \{\ h,\ d\ \} & , & S_6 = \{\ a,\ c,\ g,\ i,\ f,\ e\ \} \\
\text{Reach}(h, S_7) = \{\ d\ \} & , & S_7 = \{\ a,\ c,\ g,\ i,\ f,\ e,\ b\ \} \\
\text{Reach}(d, S_8) = \{\ \emptyset\ \} & , & S_8 = \{\ a,\ c,\ g,\ i,\ f,\ e,\ b,\ h\ \}
\end{array}
\qquad (99)
$$

Then, on comparing the results of equation (99) with Fig. A.19 we can see that the Reach operator applied successively on a, c, g, ... etc., and S_i being the set of previously eliminated nodes, gives the column indices (or row indices because of the symmetry) corresponding to all the non-zeros in the Cholesky factorization in the subset of the uneliminated nodes (i.e. after the diagonal in Fig. A.19).

With this representation and with the Reach operator already defined, we have an implicit scheme; the sequence of graphs is no longer needed and all we have to do is to find a way of handling the Reach operator in the computer.

As we can intuitively assess, the main shortcoming of the implicit scheme is the fact that the amount of work involved in determining the reachable sets can become too large.

The fast hybrid solution proposed by George and Liu (1981) is a compromise between the explicit and implicit schemes. It arises by observing that because the computation of Reach(x_i, S_i) depends on the length of the paths between the node being eliminated and that in its reachable set (through S_i), the longer the paths, the more work needed to find the reachable nodes. The solution is then to reduce the length of these paths. On the

other hand, because we are interested only in finding the reachable nodes from the set of uneliminated nodes, we do not need the already eliminated nodes. In the explicit approach (Fig. A.17) we have actually deleted the eliminated nodes from the sequence of elimination graphs. In the formal implicit model, using the Reach operator, we can reduce the length of the path by "glueing" the neighbouring nodes already eliminated, creating what might be called "supernodes", like those shown in Fig. A.20, which is a graphical representation of the process of finding the reachable sets through the subsets S_i, where each S_i is formed by the already eliminated nodes. Each graph of Fig. A.20 is a quotient graph (or a "supergraph",i.e. a graph consisting of "supernodes").

In so doing, the paths between pivots and reachable nodes through S_i have a length of 2 (when it is through S_i) or 1 (when the reachable node is the neighbour of the pivot and it does not belong to S_i). We have then reduced the path lengths up to a maximum of 2.

The relationship between the fast hybrid approach using quotient graphs (Fig. A.20) and the explicit elimination scheme (Fig. A.17) becomes quite evident. It can also be deduced from Fig. A.20 [proved by a theorem in George and Liu (1981)] that the sequence of quotient graphs can be implemented using the same storage locations used for storing the original matrix. Then, the results of the ANALYSIS stage do not need more storage than that required for storing the original matrix.

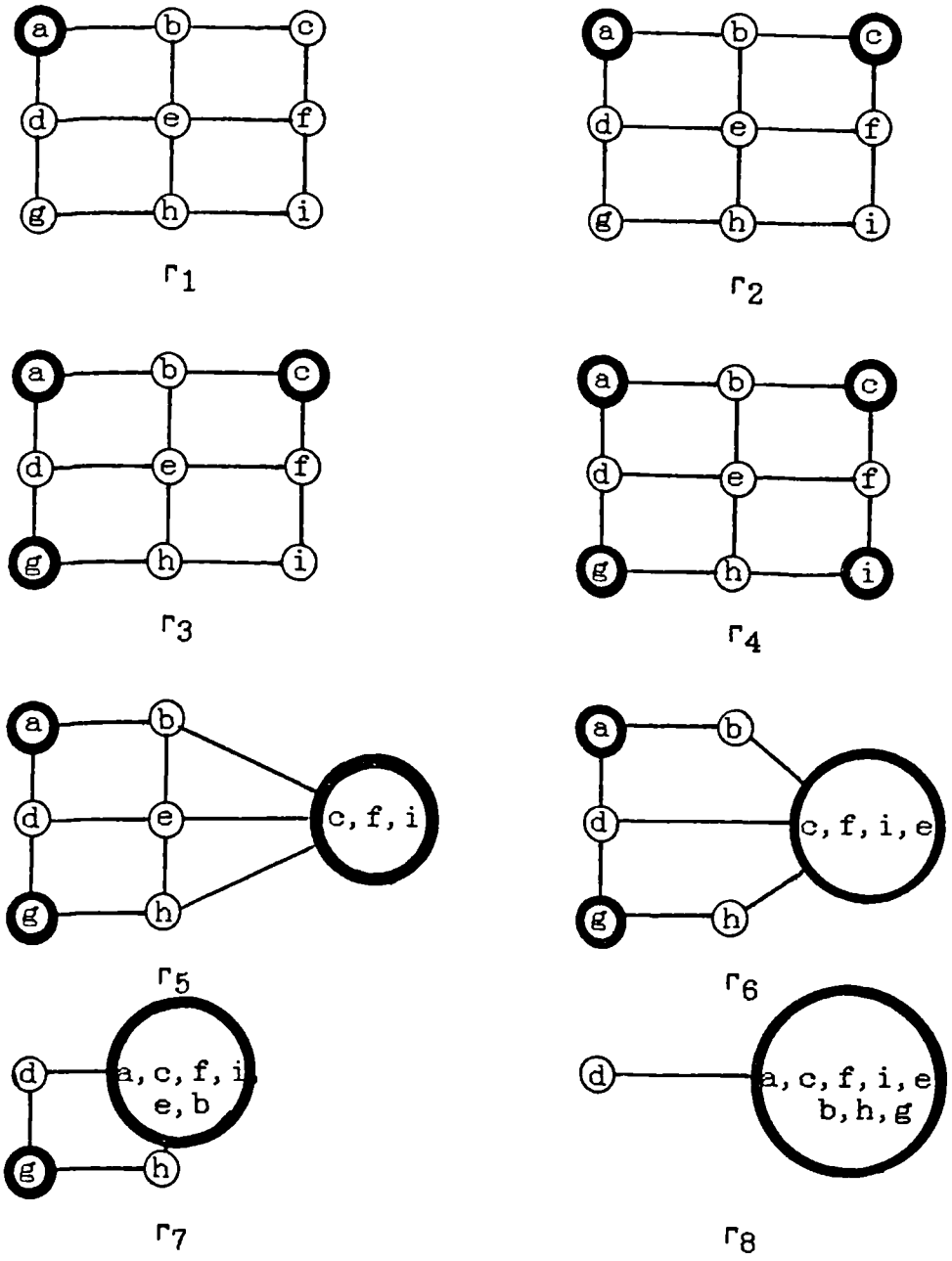


Fig. A.20. Sequence of quotient graphs Γ_i used for finding the reachable set of nodes in the elimination process.

In summary, on applying the fast hybrid scheme of George and Liu (1981), we are able to determine the sequence of reachable sets [equation (99)] for each elimination step, which formally gives the structure of the Cholesky factorization $L\,L^T$ of the original matrix. This means that we get the amount of fill-in and the locations within the factorized matrix where the fill-in

takes place, which allows us to set up the data structures needed for the following stages (FACTORIZATION/SOLUTION). All this can be done, when dealing with a symmetric positive definite matrix, without using the numerical values of the non-zeros . The search of the reachable sets is now meant to be efficient in terms of storage and speed, due to the introduction of "supernodes" and quotient graphs.

The minimum degree concepts give us the order in which the nodes are eliminated (e.g.: a, c, g, ... , h, d in our example) while the hybrid scheme gives the structure of the fill-in which such ordering produces.

An additional improvement has been used in the implementation provided by George and Liu (1981), and it is geared to reducing the time spent on searching for a minimum degree node (see Fig. A.17). The idea is, because we usually find more than one node with minimum degree (as c, g and i in Fig. A.17, during the first and second stages of the elimination), then we can use this information in order to eliminate these nodes as well, without having to carry out another minimum degree search in the next elimination stage. The enhancement used by George and Liu (1981) is based on that idea, and these nodes are said to be indistinguishable with respect to elimination. For more details see George and Liu (1981), chapter 5.

Despite the long reputation of the minimum degree algorithm and the much proclaimed advantages of the George and Liu (1981) implementation, we found in our testing that it is not as

successful either from the storage or from the execution time point of view. Moreover, we could not solve networks larger than 900 nodes within a reasonable execution time. So far, we have not been able to find an explanation for this behaviour; possible causes are the particular computer implementation used and the fact that the connectivity and "shape" of our test networks is too different from that used by George and Liu (1981).

A.3.6. Quotient tree algorithms.

These algorithms take advantage of the fact that an efficient way to deal with the sparseness of a linear system is to partition the matrix in blocks, some of which may eventually be dense matrices, but of a much more reduced order than the original system.

The advantages of this block-partitioning approach can be shown, for example, when instead of solving our model problem:

$$A\,x = b \tag{100}$$

where:

A : is a symmetric positive definite (n x n) matrix.

we solve an equivalent 2x2 block partitioned system:

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \cdot \left[\begin{array}{c} x_1 \\ \hline x_2 \end{array}\right] = \left[\begin{array}{c} b_1 \\ \hline b_2 \end{array}\right] \tag{101}$$

where:

A_{11} : a (p x p) submatrix.

A_{22} : a (q x q) submatrix, such that p + q = n.

A_{21} : a (q x p) submatrix, such that $A_{21} = A_{12}^T$.

x_1, x_2 : partitioned vector of unknowns.

b_1, b_2 : partitioned right hand side vector.

Then, the corresponding Cholesky factorization of $A = L\ L^T$, can be expressed as a block-partitioned matrix as well:

$$L = \left[\begin{array}{c|c} L_B & 0 \\ \hline W^T & L_C \end{array}\right] \tag{102}$$

where:

L_B : is the Cholesky factor of A_{11}, a (p x q) submatrix.

L_C : is the Cholesky factor of A_{22}, a (q x p) submatrix.

W : is a (q x q) submatrix and W^T its transpose.

The Cholesky factors can be computed algebraically by imposing the equivalence $A = L\ L^T$ and using equations (101) and (102):

$$\left[\begin{array}{c|c} L_B & 0 \\ \hline W^T & L_C \end{array}\right] \cdot \left[\begin{array}{c|c} L_B{}^T & W \\ \hline 0 & L_C{}^T \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]. \tag{103}$$

Performing the multiplication on the left hand side of (103) we get:

$$\left[\begin{array}{c|c} L_B\ L_B{}^T & L_B\ W \\ \hline W^T\ L_B{}^T & W^T\ W + L_C\ L_C{}^T \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \tag{104}$$

Then, identifying block by block in (104), we obtain:

a) Upper left hand side block:

$$A_{11} = L_B\ L_B{}^T \tag{105}$$

which simply states that L_B is the Cholesky factor of A_{11}. In other words, we can obtain L_B (or $L_B{}^T$) from A_{11} via Cholesky factorization.

b) Upper right hand side block:

$$L_B\ W = A_{12} \tag{106}$$

which implies that having obtained L_B from (105), we can then obtain W by solving "q" linear systems of the form:

$$L_B\ W(*,i) = A_{12}(*,i) \qquad \forall\ i=1,2,\ ..\ ,\ q \tag{107}$$

where:

$W(*,i)$: denotes the i^{-th} column (vector) of matrix W.

$A_{12}(*,i)$: denotes the i^{-th} column (vector) of matrix A_{12}.

c) Lower right hand side block:

$$W^T\ W + L_C\ L_C{}^T = A_{22} \tag{108}$$

which, on defining the auxiliary matrix C:

$$C = L_C\ L_C{}^T \tag{109}$$

becomes, after reordering :

$$C = A_{22} - W^T\ W \tag{110}$$

Hence, after having obtained W from step b) we can compute the auxiliar matrix C from (110) and, with C, we can get L_C which is the Cholesky factor of C in equation (109).

Thus, in this three stage process we are obtaining the Cholesky factors L and L^T by computing their corresponding submatrices L_B, W and L_C from A_{11}, A_{12} and A_{22}.

With the Cholesky factorization of A, the solution stage of our original problem can be carried out by replacing our original linear system (using equation 103) by:

$$\left[\begin{array}{c|c} L_B & 0 \\ \hline W^T & L_C \end{array}\right] \cdot \left[\begin{array}{c|c} L_B{}^T & W \\ \hline 0 & L_C{}^T \end{array}\right] \cdot \left[\begin{array}{c} x_1 \\ \hline x_2 \end{array}\right] = \left[\begin{array}{c} b_1 \\ \hline b_2 \end{array}\right] \tag{111}$$

and then, because we already know L_B, L_C and W^T, its solution becomes just a series of substitution processes; indeed on defining the auxiliar vector:

$$y = \begin{bmatrix} y_1 \\ \hline y_2 \end{bmatrix} = \left[\begin{array}{c|c} L_B{}^T & W \\ \hline 0 & L_C{}^T \end{array}\right] \cdot \begin{bmatrix} x_1 \\ \hline x_2 \end{bmatrix} \tag{112}$$

equation (111) becomes:

$$\left[\begin{array}{c|c} L_B & 0 \\ \hline W^T & L_C \end{array}\right] \cdot \begin{bmatrix} y_1 \\ \hline y_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ \hline b_2 \end{bmatrix} \tag{113}$$

and the auxiliary vector $y = (y_1 \mid y_2)^T$ can be computed by the following steps:

i) Solving the upper part of (113):

$$L_B\, y_1 = b_1 \tag{114}$$

we obtain y_1, by forward substitution.

ii) Compute, from the lower part of (113):

$$W^T\, y_1 + L_C\, y_2 = b_2 \tag{115}$$

which reordered gives:

$$L_C\, y_2 = b_2 - W^T\, y_1 \tag{116}$$

iii) Having y_1 from step i), y_2 can again be computed by a forward substitution from equation (116), which completes the computation of the auxiliary vector "y".

Now, having computed the vector y, we replace it into equation (112) and solve this last equation for x:

$$\left[\begin{array}{c|c} L_B{}^T & W \\ \hline 0 & L_C{}^T \end{array}\right] \cdot \begin{bmatrix} x_1 \\ \hline x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ \hline y_2 \end{bmatrix} \tag{117}$$

which can be done via the following steps:

iv) Solving the lower part of (117):

$$L_C^T x_2 = y_2 \tag{118}$$

we obtain x_2, by back substitution.

v) Compute the upper part of (117):

$$L_B^T x_1 + W x_2 = y_1 \tag{119}$$

which reordered gives:

$$L_B^T x_1 = y_1 - W x_2 \tag{120}$$

vi) Having x_2 from step v), x_1 can be obtained by back substitution from (120).

What we have done so far is a bit of algebra, but the full advantage of the block partitioning approach has not become clear. The point is that, in the solution stage [steps i) to vi)], we have used the matrices L_B, L_C and W <u>explicitly</u>, and, because W is normally a full (dense) matrix, a great saving of storage can be achieved via a sensible way of handling this matrix. In fact, the ideal approach would be not to store it but to handle it in an implicit way; this can be possible by recalling that W can be written from equation (106) as:

$$W = L_B^{-1} A_{12} \tag{121}$$

i.e. :

$$W^T = A_{12}^T L_B^{-T} \tag{122}$$

This means that whenever we need to multiply W or W^T by a vector, like in equations (116) and (120), we can do it without including W or W^T in an explicit way; in fact, using equation (122), equation (116) becomes:

$$L_C y_2 = b_2 - (A_{12}^T L_B^{-T}) y_1 \tag{123}$$

and, using (121), equation (120) becomes:

$$L_B^T x_1 = y_1 - (L_B^{-1} A_{12}) x_2 \qquad (124)$$

We can also use the implicit form of W and W^T [equations (121) and (122)] to compute the product $W^T W$ in equation (110):

$$W^T W = \{ A_{12}^T [L_B^{-T} (L_B^{-1} A_{12})] \} \qquad (125)$$

where the computations are carried out following the parenthesis sequence in (125) from the inside out, which is referred to as an asymmetric block factorization. The method then relies heavily on back and forward substitution routines, which normally are used in the SOLUTION stage.

With the implicit scheme, the factorization and solution stages require L_B, L_C and A_{12}, instead of L_B, L_C and W. Because A_{12} is much sparser than W, equations (123), (124) and (125) are preferred instead of (116) and (120) and in the evaluation of the product W^T W. Now the full advantage of a block partitioning, with an implicit handling of W becomes clearer: we do not need to store the Cholesky factor L explicitly, we only need its diagonal components L_B and L_C plus the off-diagonal block of the original matrix, i.e. A_{12}.

It can be shown that the combination of block partitioning and asymmetric factorization produces not only storage savings, but also reduces the amount of arithmetic operations.

The process can be extended from a (2x2) block system to a more general one of, say, (rxr) blocks; in that case George and Liu (1981), section 6.2.2. showed that the off-diagonal submatrices of the Cholesky factor L can be computed implicitly as:

$$L_{ij} = A_{ij} L_{jj}^{-T} = (L_{jj}^{-1} A_{ji})^T \qquad i,j=1,2, \ldots , r$$

that is to say, we only need to store the diagonal block components of L (i.e. L_{jj}) and the off-diagonal blocks of the original matrix (i.e. A_{ij} or A_{ji}), which is the same conclusion obtained in the case of a (2x2) block partitioning.

Obviously the greater the number of blocks (r), the greater the efficiency of this scheme.

Having illustrated with a small example the convenience of using the block partitioning approach, the remaining problem is to find some way of producing "good" block partitioning.

First of all, we have to define what we mean by a good partitioning. There is a close link between block partitioning and the concept of quotient graph, which was introduced previously, where the idea of "supernode" and, by extension, that of "supergraph" were introduced. In order to formalise that relationship, we have to return to these graph theory concepts.

Recall that a quotient graph is nothing more than a graph formed by "supernodes", which are simply a subset of nodes "glued" together; formally, if the original linear system has a graph defined by G = (X, E), and if we have a partitioning of X, say P, such that:

$$P = \{ Y_1, Y_2, \ldots , Y_r \} \qquad (126)$$

a quotient graph of G with respect to P, denoted as G/P, is the graph (P, E^*) where:

$$\{ Y_i, Y_j \} \in E^* \text{ if and only if } Adj(Y_i) \cap Y_j \neq \emptyset \qquad (127)$$

Loosely speaking, equation (127) simply says that a link $\{Y_i, Y_j\}$ exists if and only if two supernodes Y_i and Y_j are adjacent.

A particular kind of connected graph in which there are no cycles (loops), is called a tree, say T = (X, E) and, as a result, every pair of nodes is connected by a unique path. Of course, we can also have "supertrees", or quotient graphs without loops. A rooted tree is a tree where every single node emanates from a root node, determining an ancestor-descendent structure. If we label (or number) the nodes of a rooted tree in such a way that every node is numbered before its ancestor, we get a tree which is known as a monotonely ordered tree. Fig. A.21 is an example of a monotonely ordered tree rooted at node 8.

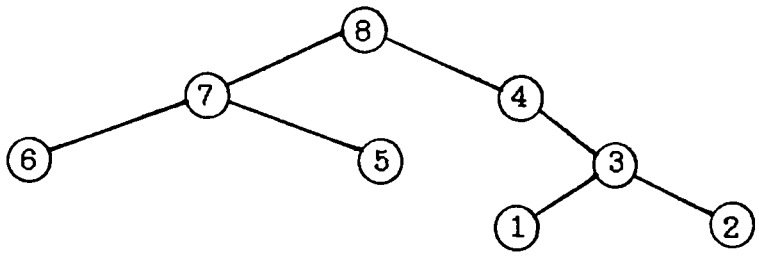


Fig. A.21. An example of a monotonely ordered tree (rooted at node 8).

The matrix corresponding to the monotonely ordered tree shown in Fig. A.21 is as follows:

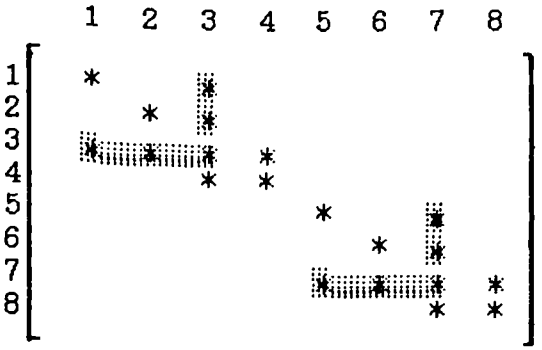


Fig. A.22. Matrix corresponding to the monotonely ordered tree shown in Fig. A.21.

As can be easily seen from Fig. A.22, the matrix of a monotonely ordered tree has a very interesting property indeed: it does not suffer fill-in during the factorization. Because of the labelling system used, where each node is numbered before its ancestor, we are generating a matrix structure which shows a series of arrow shaped arrangements pointing into the diagonal (see shaded "arrows" in Fig. A.22).

In other words, the numbering system adopted allows all the nodes to cluster themselves optimally, i.e. minimizing the profile of the matrix and thus reducing the amount of fill-in to the very minimum (i.e. no fill-in at all).

The results of the previous paragraph look quite interesting, but unfortunately in our water distribution networks we do not get trees (in general); we do get trees in sewage and drainage networks. Because loops are almost always found in water supply networks, we have to adapt this tree-oriented labelling system to a looped network (or graph) and the way of doing it is via the use of supernodes; we shall define the supernodes in such a way that the resulting quotient graph is a monotonely ordered

quotient tree ("supertree"). Thus the origin of the name of these methods has become clear.

To do so, we use again the concept of level structure, more precisely a rooted level structure.

First, note that we are looking for a tree with as many branches as possible; for example, if we use the matrix whose tree is shown in Fig. A.23, its level structure rooted at node "a" is that given in (128).

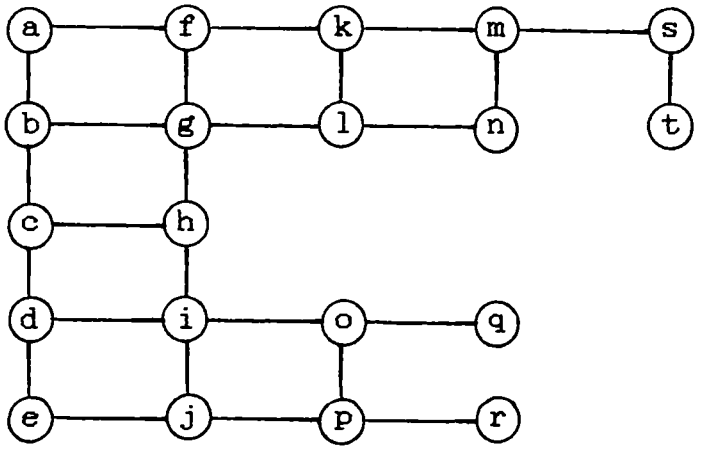


Fig. A.23. Network example for quotient tree methods.

$$
\begin{aligned}
L_0 &= \{ a \} \\
L_1 &= \{ b , f \} \\
L_2 &= \{ c , g , k \} \\
L_3 &= \{ d , h , l , m \} \\
L_4 &= \{ e , i , n , s \} \\
L_5 &= \{ j , o , t \} \\
L_6 &= \{ p , q \} \\
L_7 &= \{ r \}
\end{aligned}
\qquad (128)
$$

in this case, somebody would like to use the level structure itself as a partitioning, e.g.:

$$
\begin{aligned}
Y_1 &= \{\ a\ \} \\
Y_2 &= \{\ b\ ,\ f\ \} \\
Y_3 &= \{\ c\ ,\ g\ ,\ k\ \} \\
Y_4 &= \{\ d\ ,\ h\ ,\ l\ ,\ m\ \} \\
Y_5 &= \{\ e\ ,\ i\ ,\ n\ ,\ s\ \} \\
Y_6 &= \{\ j\ ,\ o\ ,\ t\ \} \\
Y_7 &= \{\ p\ ,\ q\ \} \\
Y_8 &= \{\ r\ \}
\end{aligned}
\qquad (129)
$$

with the corresponding quotient tree shown in Fig. A.24.

This quotient tree (Fig. A.24) generates a block-tridiagonal matrix, since each supernode is connected with only two others in a chained structure. The pattern of the generated block tridiagonal matrix is shown in Fig. A.25.

We can easily see that in this case no fill-in will occur, which is in fact due to the dense characteristics of the non-zero blocks of the matrix. The approach is clearly non-optimal, since if we compare the level structure (for example at level 5) with Fig. A.23, nodes "o" and "t" are quite far from each other and a different partitioning criterion is needed. An alternative is presented in Fig. A.26 where, for the same network of Fig. A.23, the partition is not the level structure itself, but a refinement has been used; in this case, we no longer glue nodes like "o" and "t", because they are in completely different branches. The new partitioning is produced by the following heuristic algorithm:

Let B_j be the graph defined by:

$$B_j = G\left(\bigcup_{i=j}^{l} L_i\right) \qquad (130)$$

which is referred to as the section graph. This graph is simply the union of all the nodes below the j^{-th} level structure, including the nodes in L_j. The j^{-th} level is refined in subsets Y such that:

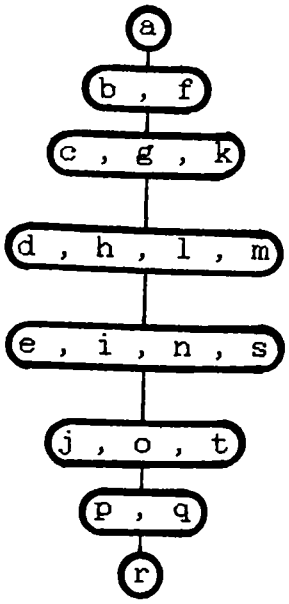


Fig. A.24. Quotient tree corresponding to the tree of Fig. A.23.

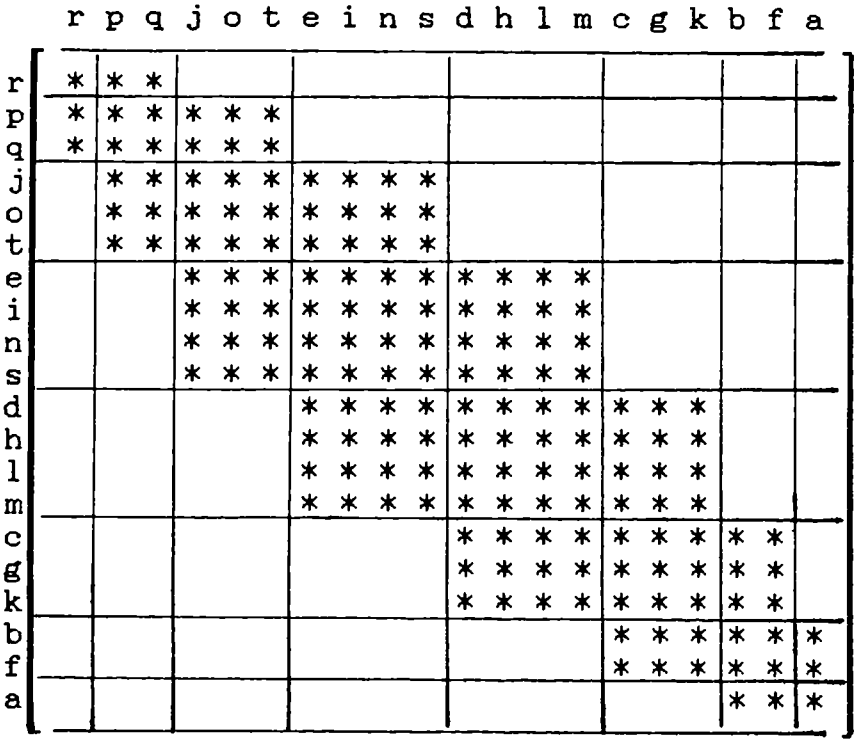
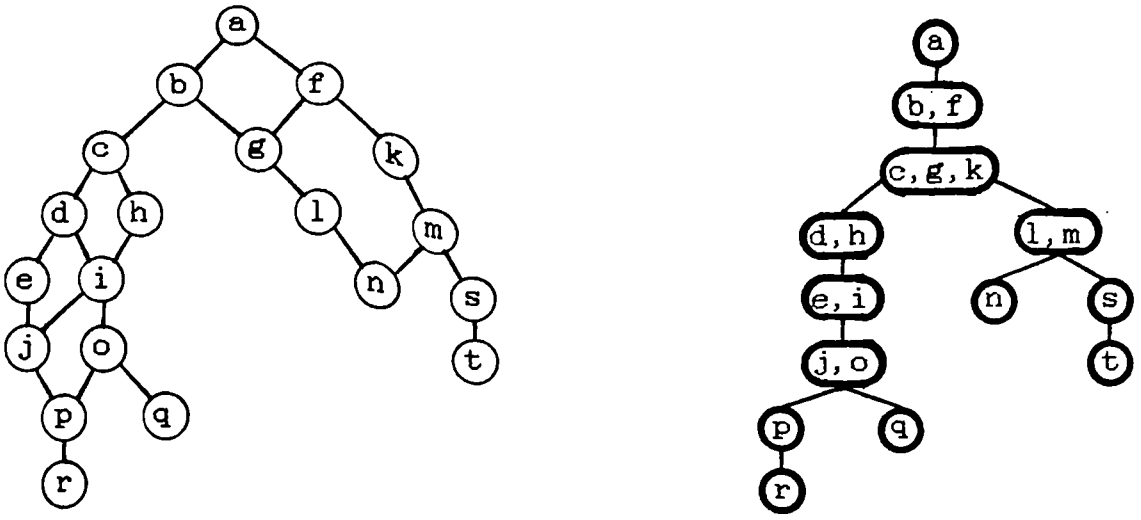


Fig. A.25. Matrix corresponding to the quotient tree of Fig. A.24.



a) Level structure b) Refined quotient tree

Fig. A.26. Level structure rooted at node "a" and its corresponding quotient tree.

{ $Y = L_j \cap C$, $G(C)$ results in a connected component of B_j } (131)

In other words, at a certain level "j", the nodes in that level are kept together (forming a unique supernode), if and only if the corresponding section graph spanned by these nodes is connected; otherwise, the nodes are re-grouped into separate supernodes, like in the case of nodes d, h, l and m at level L_3.

We can see immediately the advantage of this new partitioning approach, in the block tridiagonal matrix (Fig. A.25), we had 174 non-zeros and now in the refined quotient tree rooted at node "a" we have only 112 non-zeros. This means obviously less storage and less arithmetic operations.

As we saw before in the case of the quotient minimum degree algorithm, the level structures are dependent on the root node. In the level structure used in Fig. A.26 we chose "a" as the root

node just by chance, but we have to ask ourselves what is the best choice (if any) for such a root node. Since a partitioning with a maximum number of members will produce a matrix with the minimum number of non-zeros, it has been proposed that the concept of pseudo-peripheral nodes should be used to generate a root [George and Liu (1981), chapter 6]. Following this approach, if we generate the level structure rooted at node "t", and the corresponding refined partitioning of those levels using (131), see Fig. A.28, we get the matrix shown in Fig. A.29, which has only 108 non-zeros.

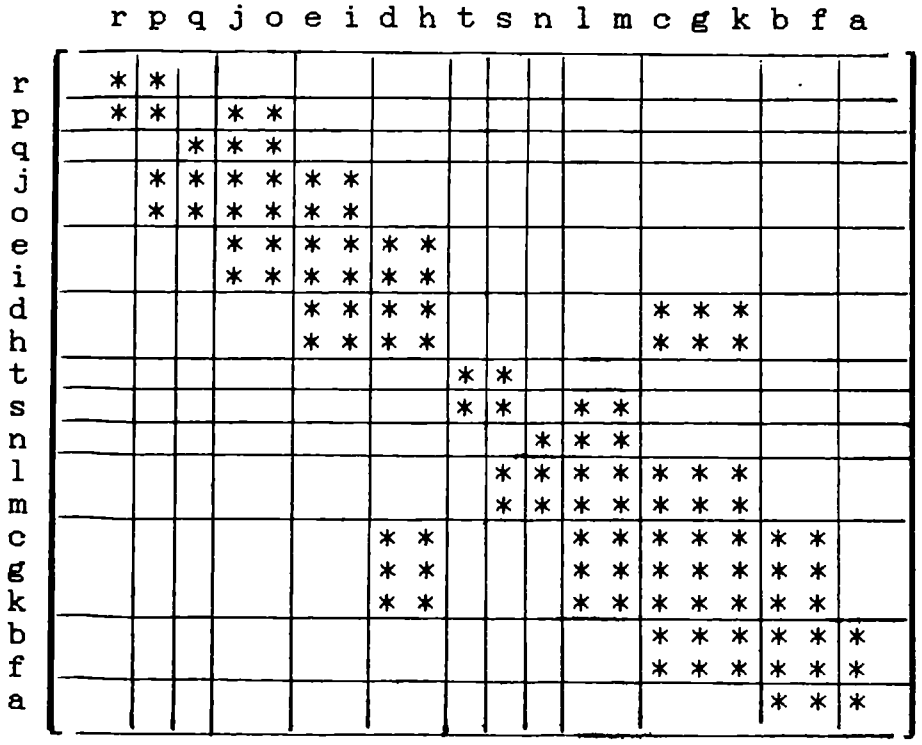
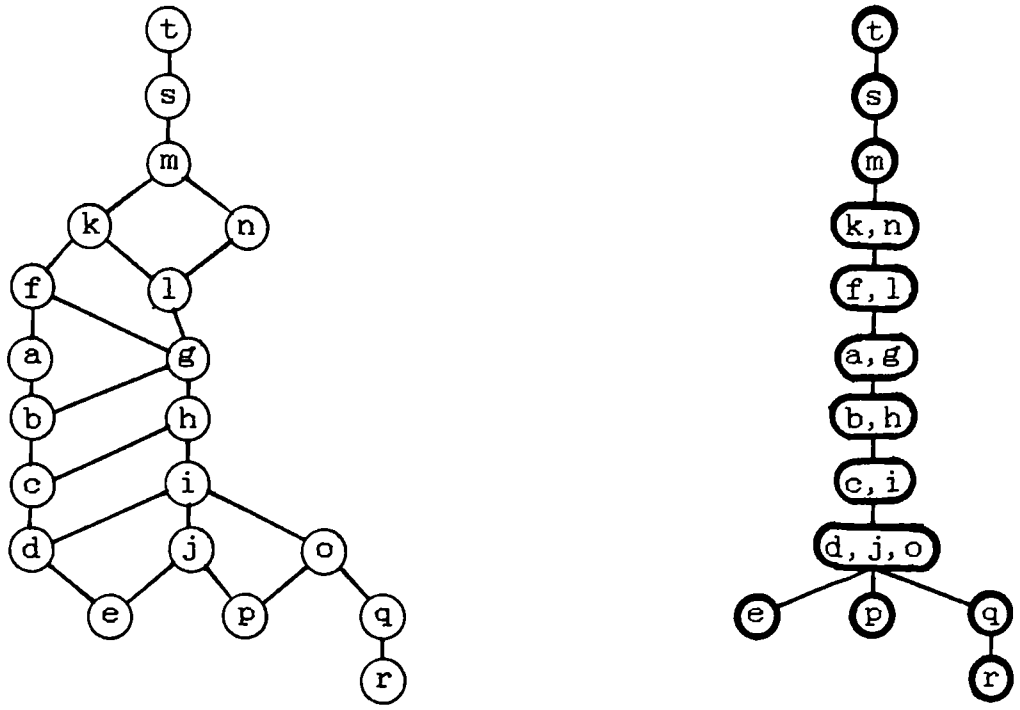


Fig. A.27. Matrix corresponding to refined quotient tree of Fig. A.26. b)



a) Level structure

b) Refined quotient tree

Fig. A.28. Level structure rooted at node "t" and its corresponding quotient tree.

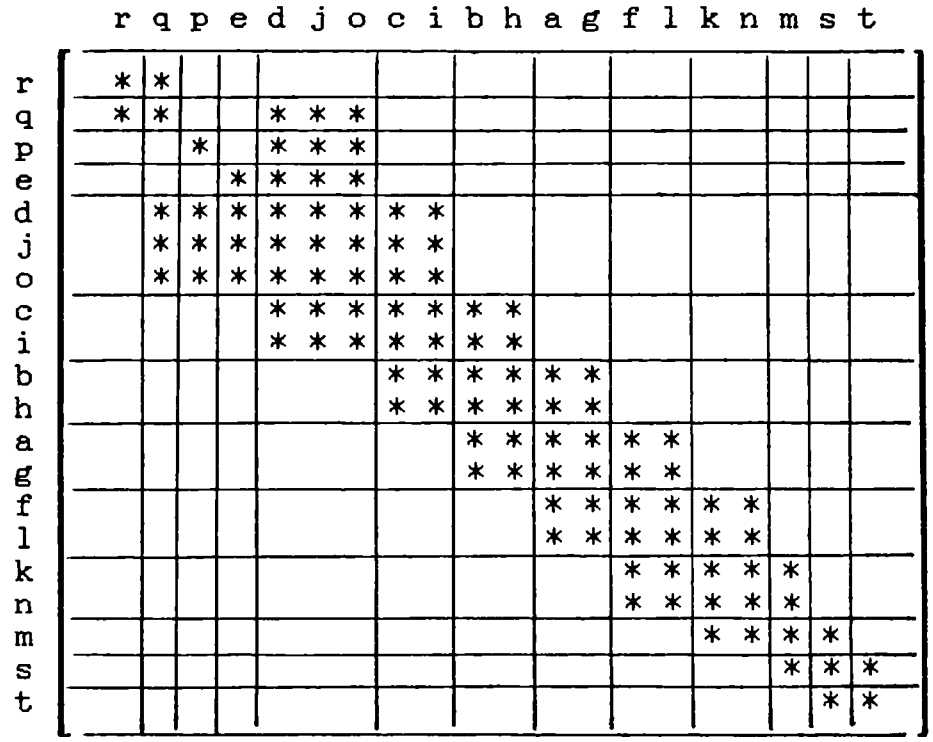


Fig. A.29. Matrix corresponding to refined quotient tree of Fig. A.28. b)

In our water network problems, we believe that this method is dependent on the shape of the networks under study; clearly in regular squared or round-shaped networks we obtain block tridiagonal matrices, whereas in branched networks we tend to obtain the sort of arrow shaped matrices we obtained at the beginning of this section (Fig. A.22).

The usefulness of some of these ideas for sewage and drainage networks looks very attractive and should be explored.

A.3.7. One-way dissection methods.

These methods were originally developed for solving linear systems arising in finite element applications. The background is again the quotient trees, and the one-way dissection algorithms are just another way of producing monotonely ordered supertree partitionings, as an alternative to the partitionings produced by the application of (131).

In essence, all dissection methods seek the division of the graph (and the associated matrix) into different members, in such a manner that the dissected graph produces a matrix with some desirable properties. Of course, these properties are related to low fill-in and the minimum number of arithmetic operations.

The idea of one-way dissection is to split up the network graph, by identifying some subset of nodes of the graph, which will be referred as "dissectors", and numbering each separated member sequentially, following a pre-specified direction. In

Fig. A.30, σ = 2 dissectors were used and 2 σ + 1 members were obtained (each dissector being a member itself).

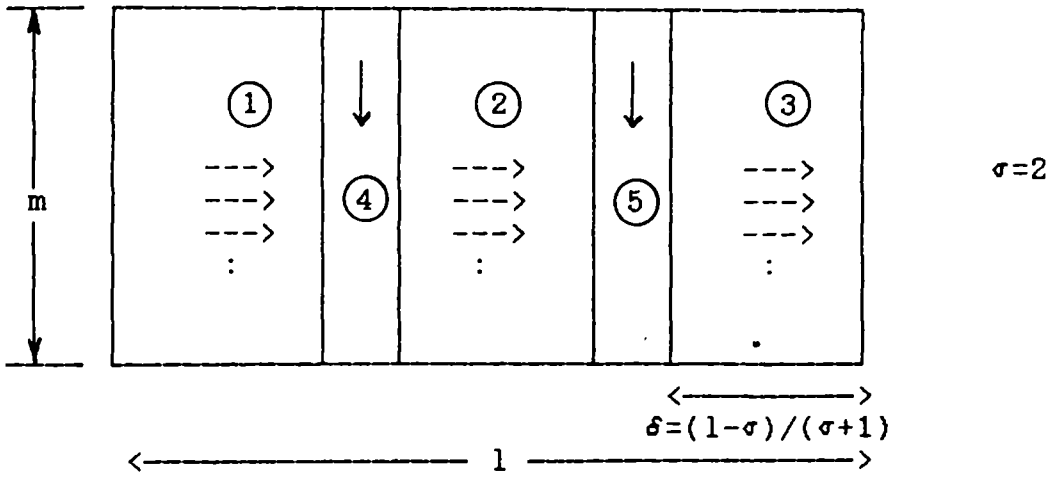


Fig. A.30. Rectangular grid partitioned with 2 dissectors.

Each non-dissector member is numbered following an horizontal left-to right and top-down sequence (see arrows in Fig. A.30), while the dissectors are numbered after all the non-dissectors members have been numbered, following a top-down left-to-right sequence (see Fig. A.30). The same result can be obtained with a right-to-left, down-top strategy.

As a result of this particular numbering system the reordered matrix has a very special form. Fig. A.31 shows an example of a 40 nodes graph, renumbered according to this method, and Fig. A.32 presents the corresponding matrix structure induced by this numbering.

From Fig. A.32 we can see that the labelling system introduced by the one-dissection scheme produces (2 σ + 1) blocks ; (σ+1) blocks of size (mδ x mδ) and σ blocks of size (m x m). All the possible fill in the Cholesky factors is concentrated in the lower part of the matrices (see shaded areas in Fig. A.32).

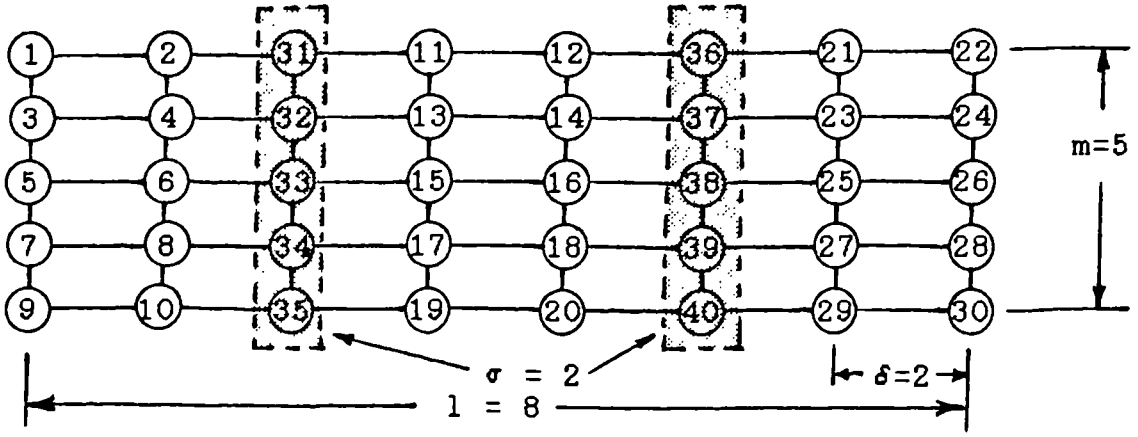


Fig. A.31. Example of a 40 nodes rectangular shaped graph, partitioned using one-way dissection.

The selection of σ presents some practical problems, since it has to be an integer; non-dissectors members of different size can be defined, in order to cope with some cases when l and the chosen σ do not produce a constant $\delta=(l-\sigma)/(\sigma+1)$. A sensible selection of σ allows us to get an optimum matrix structure, which either minimizes fill-in or arithmetic operations.

In fact, because the storage requirements can be expressed as a function of σ (or δ), it is possible to explicitly find the σ^* which minimizes the storage. The same thing can be done to minimize the amount of work in the Cholesky factorization and in its solution stage. Because storage and amount of work are conflicting objectives, we either choose one the other, thus "tuning" the algorithm for one of them, which usually is the storage. According to the testing carried out by George and Liu (1981), their one-way dissection implementation was found to be the least demanding on storage algorithm, when compared with the envelope method, the refined quotient tree method and the nested dissection method.

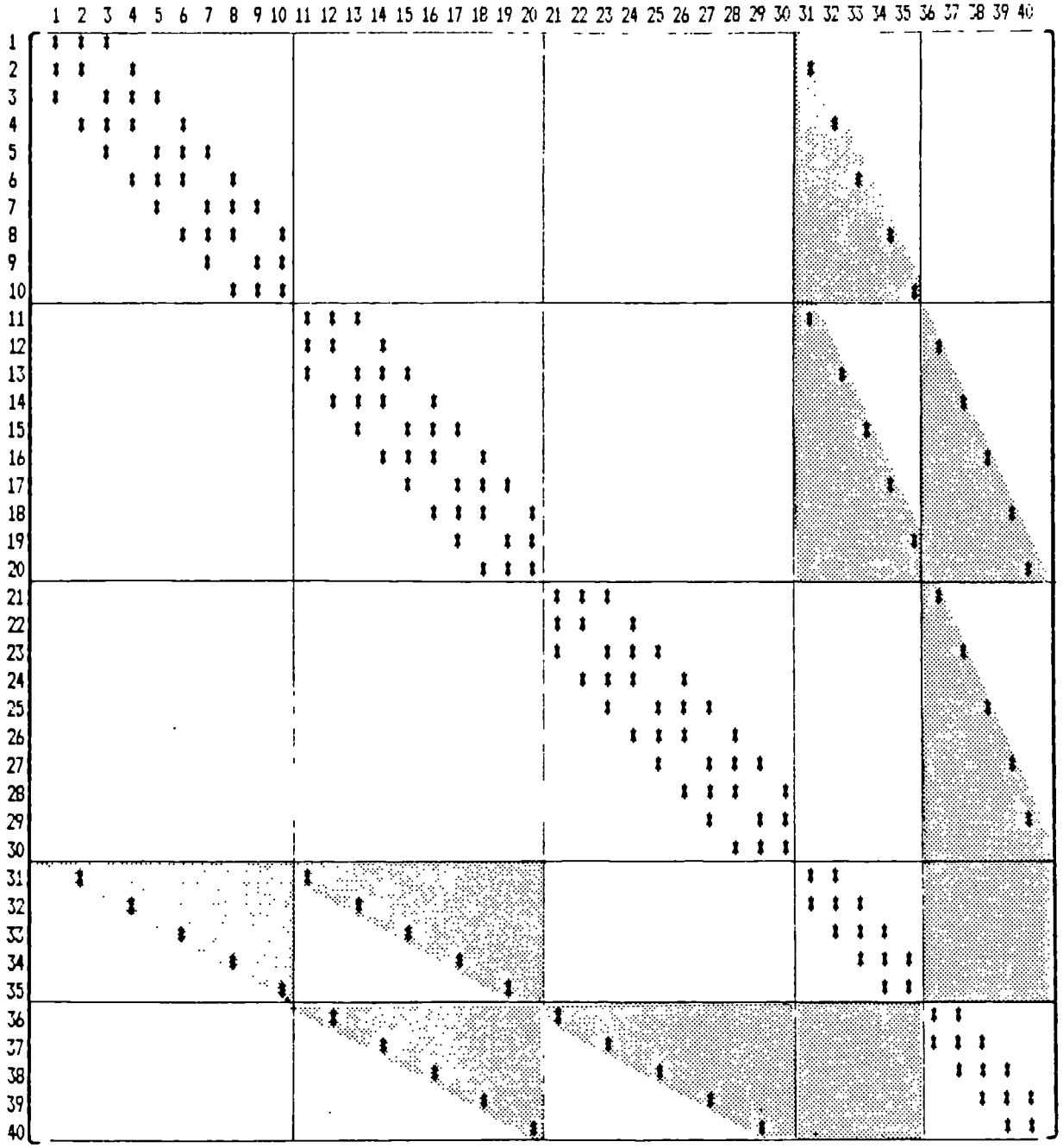


Fig. A.32. Matrix structure corresponding to graph of Fig. A.31.

We implemented the subroutines published by George and Liu (1981), with the gradient method for the solution of water distribution networks, and we can confirm the results of George and Liu. Furthermore, we extended the comparison to include the

MA27 package of the Harwell Subroutine Library and we also found that one-way dissection was still the best of the direct methods, from the storage viewpoint.

A.3.8. Nested dissection methods.

Nested dissection is a further development of one-way dissection and it can be explained as follows.

When applying one-way dissection to the matrix whose graph is Fig. A.31 , we only considered vertical dissectors; actually, there is nothing to prevent us splitting the graph even further by using horizontal dissectors as well, as shown in Fig. A.33.

The result of applying nested dissection to the graph shown in Fig. A.33, as far as the corresponding matrix is concerned, is shown in Fig. A.34. On comparing the structures generated by one-way dissection and nested dissection (see Figs. A.32 and A.34), we can find that one of the main effects of nested dissection is *to reduce the size of the diagonal submatrices, by* increasing its number and its density; this is highly desirable, since we do have to store these diagonal submatrices.

We have used the nested dissection *subroutines published by* George and Liu and obtained the fastest execution time of all the direct methods we have reviewed so far, though in storage the algorithm is not as good as the one-way dissection.

According to George and Liu (1981) and George (1981), their algorithm is not as efficient as other implementations, but it is simpler and suitable for tri-dimensional meshes (in 3-D finite

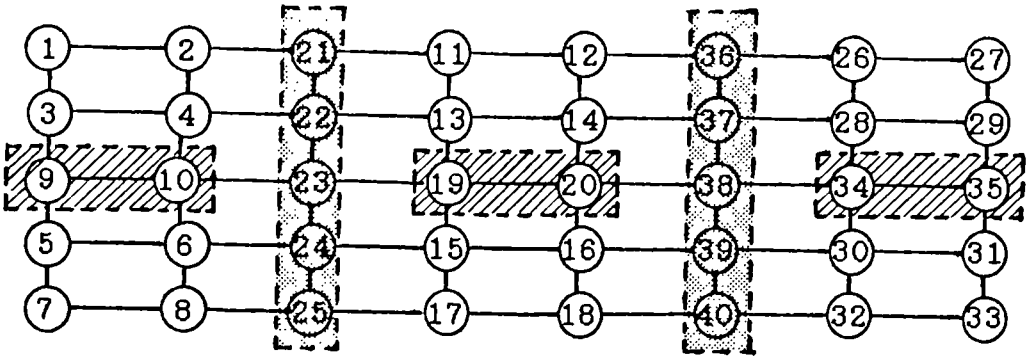


Fig. A.33. Example of a 40 nodes rectangular-shaped graph, partitioned using nested dissection.

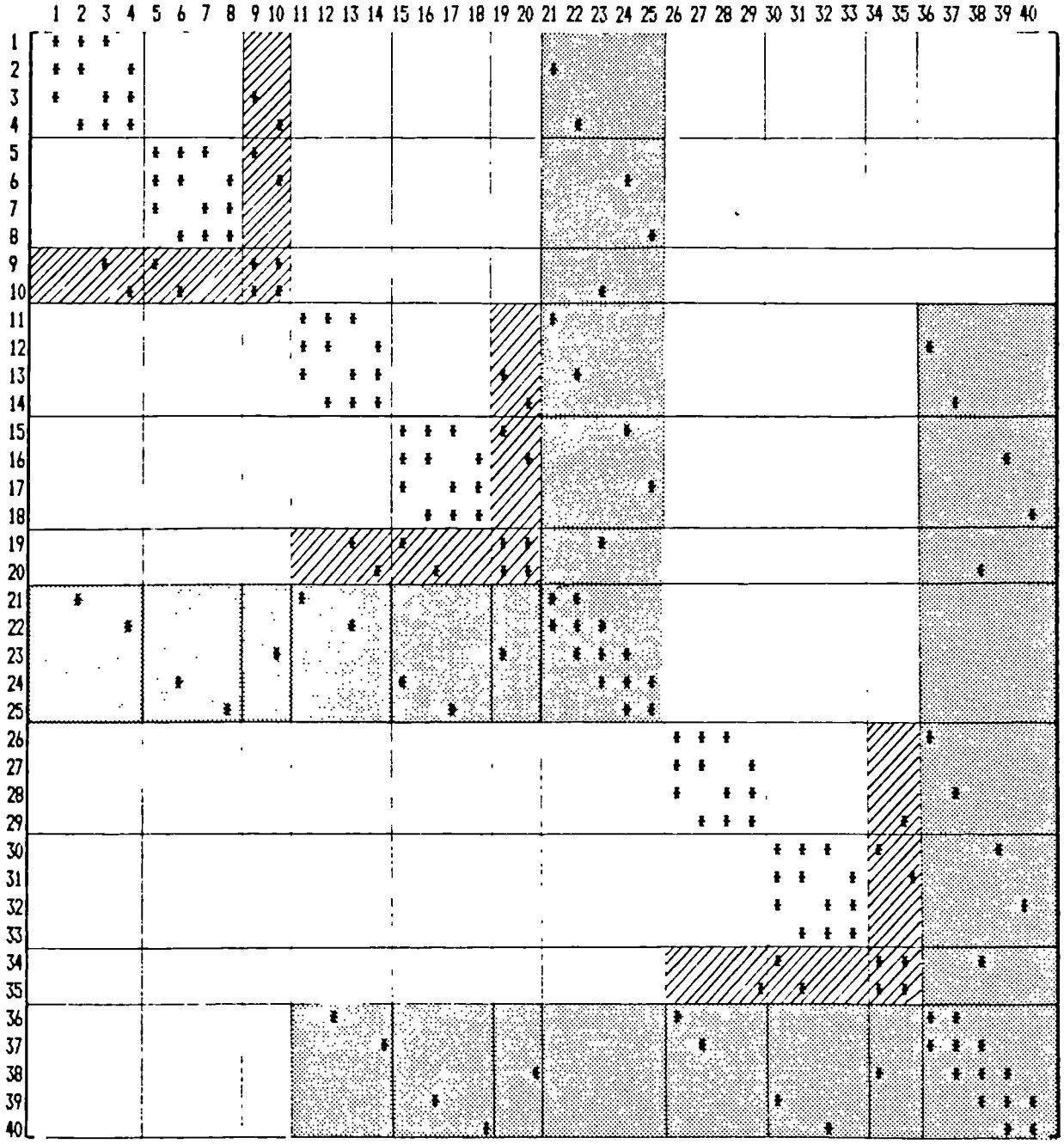


Fig. A.34. Matrix structure corresponding to graph of Fig. A.33.

element problems, for example), whereas some other implementations are restricted to 2-D problems (planar meshes).

Nested dissection seems to be better for problems where the network has a squared or rounded shape, while one-way dissection is more appealing for problems generating a strongly rectangular-shaped graph.

George (1981) warned about the reliability of this routines: "there are no results guaranteeing the quality of its outputs"; in our testing with water distribution networks, we have found full agreement between the results obtained with this method and others.

A.3.9. Frontal and multifrontal methods.

The frontal method derives from the search for efficient methods for solving the huge linear systems generated in the application of the finite element method. Although the ideas behind the method had been used for a long time, the name and its formal presentation are due to Irons (1970).

The frontal method aims primarily at the reduction of storage, since the amount of operations is basically the same as in other Gaussian elimination-based algorithms, though the order in which these operations are carried out is different.

The foundations of the frontal method are strongly tied up with the way in which finite element matrices are produced. In the finite element method (FEM) a problem is discretized over its

continuous domain, aiming at evaluating the unknowns only at certain points (nodes) within the domain, rather than at all its (infinite) points. The discretization usually generates a mesh of nodes, which are joined by edges, forming basic geometric figures (typically triangles and/or rectangles). Each one of the elementary figures constitutes a "finite element" or simply "element".

The global matrix (or "stiffness" matrix, as a reminiscent from the structural analysis problems where FEM was originally developed) is actually assembled from each one of the elementary matrices relating the value of the unknown variable at the finite element nodes with the boundary conditions and with the corresponding physically meaningful parameters. This is usually formalised as:

$$A = \Sigma\ B^{(k)} \qquad (132)$$

and

$$\underline{b} = \Sigma\ \underline{c}^{(k)} \qquad (133)$$

where:

- A : global "stiffness" matrix.
- $B^{(k)}$: is the matrix coming from the k-th finite element, with its order equal to the number of nodes in the k-th element.
- $\underline{c}^{(k)}$: right hand side vector corresponding to the k-th finite element.
- $\underline{b}$: right hand side vector of the global system.

The key observation here is that the finite element definition, i.e. its shape, size and nodes, provides the adequate basis for producing a desirable structure for the Gaussian elimination

process. In all the previous methods (like quotient tree and dissection algorithms), a great deal of effort was spent in producing the adequate orderings. In FEM, this structure comes up "naturally", from the finite element specification. In fact, the elimination order in FEM is determined by the sequence in which the global matrix A is assembled.

Another basic observation leading to the development of the frontal method, deals with the fact that we do not have to wait until the global matrix is fully assembled (and indeed stored) to start the elimination process. In fact all we need for eliminating a row and column, say "1", is to wait until its coefficients in the global matrix have been completely added; at that point, the elimination of the variable "1" can be carried out, and because we no longer need it, the corresponding row can be stored out of core until the substitution stage requires it.

The following example, from Livesley (1983), illustrates the elimination sequence. Fig. A.35 shows the element definition and ordering, while Fig. A.36 shows the corresponding elimination sequence.

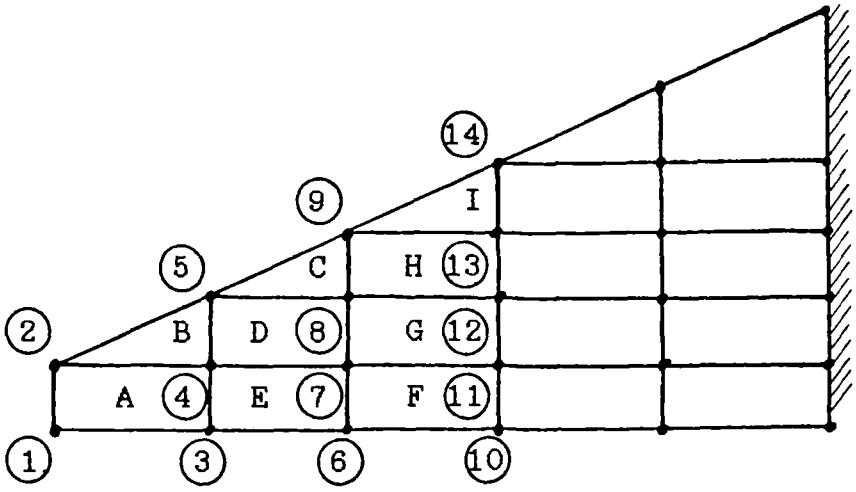


Fig. A.35. Finite element definition and ordering for a frontal solution scheme, from Livesley (1983).

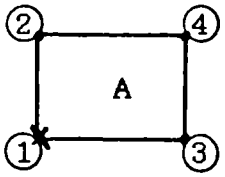
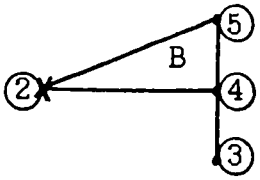
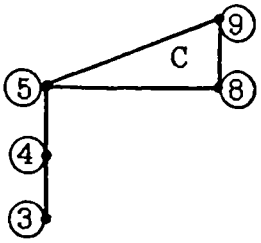
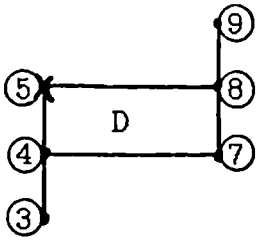
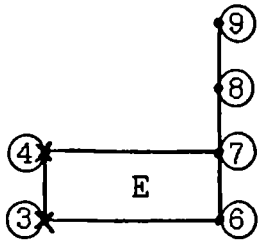
Element introduced	Nodes eliminated	Frontal nodes
	1	2 - 4 - 3
	2	5 - 4 - 3
	none	9 - 8 - 5 - 4 - 3
	5	9 - 8 - 7 - 4 - 3
	3 , 4	9 - 8 - 7 - 6

Fig. A.36. The elimination sequence in a frontal solution scheme from Livesley (1983).

The "front" advances across the mesh following the assembly sequence, which defines the elimination order. The frontal nodes (or variables) lie in the border between the already assembled elements and the un-assembled ones. The rows and columns corresponding to the frontal nodes are usually kept in a full (dense) matrix known as the "frontal matrix".

From Fig. A.36 it is clear that a node cannot be eliminated until the contributions from all its related elements have been assembled, like node 5, which has to wait until element D is assembled.

From the previous example it is also clear that the front-width (i.e. maximum number of frontal nodes) depends on the order in which the finite elements are assembled, which is reflected by its numbering. This leads to the conclusion that an efficient frontal method implementation has to include some way of minimizing the front width. This can be done using some of the ideas described previously for ordering sparse matrices.

In the previous example, the sequence for assembling the global matrix A ,using the notation of equation (132), is defined by:

$$A = [[[[\ [B^{(A)}] + B^{(B)}] + B^{(C)}] + B^{(D)}] + B^{(E)}] + \ldots \qquad (134)$$

where the assembling takes place from the inner brackets outwards. This sequence corresponds with the elimination sequence shown in Fig. A.36.

Equation (134) can indeed be assembled in many different ways, for example:

$$A = \{\ [B^{(A)} + B^{(B)}] + [B^{(C)} + B^{(D)}]\ \} + \{\ [B^{(E)}] + \ldots\} \qquad (135)$$

where, again, the assembling is carried out following the outwards direction of bracketing. This implies that we can actually assemble A in such a way as to produce more than one "front", each one with its corresponding frontal matrix, which can be processed and kept aside temporarily. This is an extension of the frontal method, leading to the "multifrontal" method,

where a sensible selection of the fronts allows extra savings in terms of number of operations.

The complexity of the computer implementation of the frontal methods is considerable. We have used the routine MA27 of the Harwell Library, which consists of 15 subroutines with 2920 Fortran records (cards). As an additional feature, MA27 allows the solution of indefinite sparse symmetric linear systems, because some pivoting is performed within the frontal matrix. The results of the comparison between the multifrontal method implemented in MA27 and other methods, for the solution of water distribution networks, is presented in Chapter Five.

Reid (1981), describes the main features of frontal methods, while Duff and Reid (1982_a, 1982_b) describe the main characteristics of the MA27 routine. Duff, Erisman and Reid (1986, Chapter 10) give an up to date description of frontal and multifrontal methods for finite element problems.

It has to be emphasised that, even though frontal and multifrontal methods have been developed for and from FEM applications, their use is open to non-FEM problems.

A.4. Review of iterative methods for the solution of linear systems of equations.

A.4.1. Introduction.

When solving a linear system of equations there are a number of reasons why, sometimes, an iterative (approximate) approach can be most appropriate:

a) For example, if we are solving a non-linear system of equations via an iterative scheme, which builds up a sequence of linear systems, there is no point in getting an "exact" solution in the first iterations of the non-linear algorithm, since we know beforehand that this is not the true solution of our original non-linear system, and all we want is an approximate one. In those cases, considerable effort can be spared with an approximate/iterative solution of the first successive linear systems, provided that the iterative solution is cheaper than a direct solution.

b) In engineering practice, real problems usually imply linear systems of the order of hundreds or thousands and, in these cases a direct solution can become very expensive from the amount of work and storage point of views. This has been the main argument in favour of iterative methods for quite a long time, but nowadays the development of fast sparse direct algorithms is challenging this argument.

c) The availability of fast and accurate algorithms for the iterative solution of linear systems of equations, on the one hand, and the widespread use of cheaper microcomputers in real-life problems, on the other hand, makes the use of iterative methods for the efficient solution of linear systems very attractive indeed.

Thus, in the past 10-20 years, direct methods have been associated with big computers, with massive storage capacity and iterative methods have been linked to mini and microcomputers.

Though this situation is changing very rapidly, it is still considered like a kind of rule of thumb by a number of people.

In essence, when storage is a critical constraint, an iterative method seems to be the right choice for the solution of linear systems, at least to start with.

We shall review the most widely used iterative methods, in an increasing degree of complexity and sophistication. Starting with some traditional methods (like Jacobi, Gauss-Seidel, Successive over relaxation, etc.), a unified view of all these methods is presented, since all of them can be studied as particular cases of a single general method. Finally, we shall review a conceptually different approach to solving linear systems, based on optimisation methods (conjugate gradients). We shall not include iterative methods like "alternating directions" and "block iterative methods", which are applied especially to solve linear systems arising from differential equations (see Ortega and Poole (1981) and Stoer and Bulirsh (1980) for details on these methods).

Basically, all iterative methods aim at producing a sequence of solution vectors such as:

$$\underline{x}^{(o)}, \underline{x}^{(1)}, \underline{x}^{(2)}, \ldots, \underline{x}^{(k)}$$

which converge towards the desired solution. The different methods have particular ways of producing this sequence.

A.4.2. Jacobi's method (method of simultaneous displacements).

Let us consider our model problem:

Solve $$A\,x = b \qquad (136)$$

that is to say:

$$\left.\begin{array}{l} a_{11}\,x_1 + a_{12}\,x_2 + a_{13}\,x_3 + \ldots. + a_{1n}\,x_n = b_1 \\ a_{21}\,x_1 + a_{22}\,x_2 + a_{23}\,x_3 + \ldots. + a_{2n}\,x_n = b_2 \\ a_{31}\,x_1 + a_{32}\,x_2 + a_{33}\,x_3 + \ldots. + a_{3n}\,x_n = b_3 \\ \quad : \qquad\quad : \qquad\quad : \qquad\qquad : \qquad : \\ a_{n1}\,x_1 + a_{n2}\,x_2 + a_{n3}\,x_3 + \ldots. + a_{nn}\,x_n = b_n \end{array}\right| \qquad (137)$$

If we assume that $a_{ii} \neq 0$ $\forall$ i=1,2,..,n , then we can apply the same idea of the simple iteration method (in one variable) to an n-dimensional space, and calculate $\underline{x}$,in an iterative fashion, by reordering equation (137) as:

$$\begin{array}{l} a_{11}\,x_1 = b_1 - (a_{12}\,x_2 + a_{13}\,x_3 + \ldots + a_{1n}\,x_n) \\ a_{22}\,x_2 = b_2 - (a_{21}\,x_1 + a_{23}\,x_3 + \ldots + a_{2n}\,x_n) \\ a_{33}\,x_3 = b_3 - (a_{31}\,x_1 + a_{32}\,x_2 + \ldots + a_{3n}\,x_n) \qquad (138) \\ \quad : \qquad\quad : \qquad\quad : \qquad\qquad : \\ a_{nn}\,x_n = b_n - (a_{n1}\,x_2 + a_{n2}\,x_2 + \ldots + a_{nn-1}\,x_{n-1}) \end{array}$$

from which the values of the unknowns can be expressed in a compact form as:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij}\, x_j^{(k)} \right) \quad \forall\ i=1,..,n \qquad (139)$$

where (k+1) and (k) refer to the iteration index.

Then, Jacobi's algorithm can be expressed as:

Step 1. Starting with k=0 and an initial solution $\underline{x}^{(o)}$, which can be zero or a better solution, if available.

Step 2. Compute $\underline{x}^{(k+1)}$ using equation (139)

Step 3. Compute $E = || \underline{x}^{(k+1)} - \underline{x}^{(k)} ||$ or

$$E = \frac{|| \underline{x}^{(k+1)} - \underline{x}^{(k)} ||}{||\underline{x}^{(k)}||}$$

or any other index that can be useful as a detention (convergence) criterion.

and, if $E > \epsilon$: a given small termination value, then go to step 4, otherwise stop; $\underline{x}^{(k+1)}$ is the result for that level of accuracy.

Step 4. If $(k+1) > M$: a prescribed maximum number of iterations, then stop, because an accurate solution has not been found in a reasonable number of iterations. Otherwise [$(k+1) \leq M$], increase k to k+1 and go to step 2 again.

The convergence conditions for this algorithm are given in a subsequent section.

As it can be seen from equation (139), when we are computing the value of $x_i^{(k+1)}$, the right hand side uses $x_j^{(k)}$ from the previous iteration. A better approximation can be obtained using the last value (of the current iteration) for those x_j whose index j is smaller than i [i.e. for those values x_j which have been already calculated in the (k+1) iteration]. This idea leads to the next method: Gauss-Seidel method.

A.4.3. Gauss-Seidel method (method of successive displacements).

This is the extension of Jacobi's method using the updated values of x_j in the computation of the following x_i for $i > j$, then equation (139) is replaced by:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j<i} a_{ij}\, x_j^{(k+1)} - \sum_{j>i} a_{ij}\, x_j^{(k)} \right] \quad \forall i=1,..,n \qquad (140)$$

and the algorithm described for Jacobi's method remains the same, replacing equation (139) by (140).

No extra effort is needed for the Gauss-Seidel method, in comparison with Jacobi's method, since all we have to do is to consider the last updated values of the unknowns, rather than those of the previous iteration. As a result, there is no reason for using Jacobi's method instead of Gauss-Seidel.

A.4.4. Successive over (or under) relaxation method.

It has been found that further improvements in the rate of convergence can be achieved by varying the magnitude of the step $[\, x^{(k+1)}-x^{(k)}]$ taken in the Gauss-Seidel method.

In fact, if we define the new value found via equation (140) as an intermediate value, say $\hat{x}_i$, then we can compute the final new value as:

$$x_i^{(k+1)} = x_i^{(k)} + w \{ \hat{x}_i - x_i^{(k)} \} \qquad (141)$$

"w" is called a relaxation parameter, more precisely, we have over-relaxation if w>1 and under-relaxation if w<1. Of course, in the case w = 1, we are simply applying the original Gauss-Seidel method. We shall discuss in the next sections how to choose "w" to get the maximum acceleration in the iterative procedure.

A.4.5. Relationship between the previous methods.

If we express our model problem $A\, x = b$ as:

$$[\, B + (A - B) \,]\, x = b \qquad (142)$$

where B is an arbitrary matrix, then:

$$B\, x + (A - B)\, x = b \qquad (143)$$

Using the same idea behind simple (Picard) iteration, we can solve (143) via the following iterative scheme:

$$B\, x^{(k+1)} + (A - B)\, x^{(k)} = b \qquad (144)$$

where the superscript refers to the iteration counter.

Since we have the freedom to choose B, we can assume that we select a non-singular matrix B; thus, the unknowns in (144) can be computed as:

$$x^{(k+1)} = B^{-1} [\, b - (A - B)\, x^{(k)} \,] \qquad (145)$$

or

$$x^{(k+1)} = B^{-1} b - B^{-1} A\, x^{(k)} + B^{-1} B\, x^{(k)} \qquad (146)$$

then, reordering and considering $B^{-1} B = I$:

$$x^{(k+1)} = x^{(k)} - B^{-1} A\, x^{(k)} + B^{-1} b \qquad (147)$$

or

$$x^{(k+1)} = [\, I - B^{-1} A] \, x^{(k)} + B^{-1} b \qquad (148)$$

Equation (148) represents a common matrix formulation for all the iterative methods already seen.

Although interest in equation (148) is mainly theoretical, since for programming purposes we must consider the subindexed expressions (139), (140) and (141), the fact is that from equation (148) we can generate all the previous methods by choosing an adequate B matrix. The key factor for the successfulness of any method is that B has to be easily invertible, and that B must be as similar to A as possible. Equation (148) can be rearranged in a more general expression:

$$x^{(k+1)} = M\, x^{(k)} + d \qquad (149)$$

where:

$$M = [\, I - B^{-1} A \,]$$

and

$$d = B^{-1} b$$

We shall look at the iterative methods from the perspective of equation (149), especially the conditions for their convergence.

On decomposing matrix A as:

$$A = - E + D - F \tag{150}$$

where:

$$E = - \begin{bmatrix} 0 & & & \\ a_{21} & 0 & & \\ \vdots & \vdots & & \\ a_{n1} & a_{n2} & \cdots & a_{nn-1} \; 0 \end{bmatrix} \tag{151}$$

i.e. a lower triangular matrix with the same corresponding coefficients of A.

$$D = \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & . & \\ & & & a_{nn} \end{bmatrix} \tag{152}$$

i.e. a diagonal matrix with the same diagonal coefficients of A.

$$F = - \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ & & & \vdots \\ & & 0 & a_{n-1n} \\ & & & 0 \end{bmatrix} \tag{153}$$

i.e. an upper triangular matrix with the same corresponding coefficients of A.

Then, assuming $a_{ii} \neq 0 \quad \forall i = 1,..,n$ (i.e. D non-singular):

a) The <u>Jacobi</u> method can be obtained from equation (148) by choosing B such that:

$$B = D \; ; \tag{154}$$

in so doing:

$$M = I - B^{-1} A = I - D^{-1} (- E + D - F)$$

$$= I - D^{-1} (- E - F) - D^{-1} D$$

$$M = D^{-1} (E + F) \tag{155}$$

b) The Gauss-Seidel method can be obtained from equation (148) by choosing B as:

$$B = D - E \; ; \tag{156}$$

in that case:

$$\begin{aligned} M = I - B^{-1} A &= I - (D - E)^{-1}(D - E - F) \\ &= I - [\,(D - E)^{-1}(D - E) - (D - E)^{-1}F] \\ &= I - [\,I - (D - E)^{-1} F] \\ &= (D - E)^{-1} F \end{aligned} \tag{157}$$

c) The iterative improvement method for the correction of a solution of the linear system, which has been previously obtained by a factorization method, can also be expressed as a member of this family of iterative methods.

Suppose we have decomposed the matrix of coefficients A in a LU fashion. If we acknowledge that, because of rounding errors during the factorization stage the equality A = LU does not hold, we have instead:

$$A \approx L\,U \tag{158}$$

Let us take:

$$B = L\,U \tag{159}$$

then

$$B^{-1} = (L\,U)^{-1} = U^{-1}\,L^{-1}$$

and

$$M = I - U^{-1}\,L^{-1}\,A \tag{160}$$

d) The successive over-relaxation method can be obtained from equation (148) by choosing B such that:

$$B = (1/w)\,(\,D - w\,E) \tag{161}$$

where now B and M depend on the relaxation factor (scalar) "w". Then

$$M = I - B^{-1} A = I - w(\,D - w\,E\,)^{-1}(\,-E + D - F\,)$$

$$M = (D - wE)^{-1} [(D - wE) - w(- E + D - F)]$$

$$= (D - wE)^{-1} [(1-w)D + wF] \qquad (162)$$

which, for the particular case w = 1, gives the Gauss-Seidel method (equation 157).

A.4.6. Convergence conditions for the previous methods.

There is a very close relationship between the convergence behaviour of a linear system of equations and the way their eigenvalues are distributed. It is known (Duff, 1985), that slow convergence is associated with evenly distributed eigenvalues, while the presence of clustered eigenvalues implies fast convergence. So, the eigenvalues are the key to the study of the convergence of the iterative methods we are interested in.

Unfortunately, the process for obtaining the eigenvalues of a matrix is quite costly from the computational viewpoint, involving some iterative procedure. Nevertheless, a number of computer codes are available to compute the eigenvalues.

We shall summarise the results of applying some linear algebra concepts to the conditions for convergence of the iterative methods already seen. The details of most of these concepts, fully proven, can be found for example in Stoer and Bulirsh (1980).

We shall not involve ourselves in the mathematics, but we will introduce only the main concepts. A complete review of the computational procedures available to determine eigenvalues and eigenvectors can be found in Ruhe (1977).

The eigenvalue problem is usually defined in terms of finding a set of scalars λ_i for i=1,...,n such that:

$$A\,x = \lambda\,B\,x \qquad \forall x \neq \underline{0} \qquad (163)$$

where B is usually taken as the identity matrix, i.e.:

$$A\,x = \lambda\,x \qquad \forall x \neq \underline{0} \qquad (164)$$

If A is a (nxn) matrix, there are "n" values satisfying this condition, some of them real or complex numbers. Nevertheless, if A is symmetric, its eigenvalues are all real; furthermore, if A is positive definite, its eigenvalues are all positive.

Let λ_1, λ_2, ..., λ_n be the eigenvalues of A, the matrix of coefficients of the linear system (where some of them can be repeated); then the spectral radius of the matrix A is defined as:

$$\rho(A) = \max\{|\lambda_i|,\ i=1,..,n,\ \text{where } \lambda_i \text{ is an eigenvalue of A}\} \qquad (165)$$

Because the spectral radius represents the scatter of the eigenvalues, it can be used to find some information about the convergence rate of an iterative method. In order to avoid the explicit computation of the eigenvalues, it is possible to relate the spectral radius and the norm of the matrix . On applying the norm operator to equation (164), it is easy to prove that:

$$|\lambda| \leq ||A|| \qquad (166)$$

which is valid for <u>any norm</u>. Then, from the definition of spectral radius, we have:

$$\rho(A) \leq ||A|| \qquad (167)$$

This means that any norm of A is an upper limit for its spectral radius. This result is useful, since it avoids the

explicit computation of the spectral radius and eigenvalues, when trying to determine the convergence properties of a method.

The basic (necessary and sufficient) condition for the convergence of any iterative algorithm, represented by its general formulation:

$$x^{(k+1)} = [\ I - B^{-1} A]\ x^{(k)} + B^{-1} b \qquad (148)$$

or

$$x^{(k+1)} = M\ x^{(k)} + d \qquad (149)$$

is :

$$\rho(I-B^{-1}A) < 1 \qquad (150)$$

or simply:

$$\rho(M) < 1 \qquad (151)$$

On applying the result of equation (167), we find that a sufficient condition for convergence of the algorithm represented by (148) is simply:

$$|| \ I - B^{-1} A \ || < 1 \qquad (152)$$

In general, the more distant from 1 the spectral radius or the norm of M is, the faster the convergence of the method. Equation (152) provides a quick means for getting a first check if a method is convergent or not, since this only implies the evaluation of any norm, say for example the maximum norm.

The results of applying these concepts to the iterative methods seen so far, can be summarised as follows:

a) Jacobi's method: in general converges when the matrix A is strictly diagonally dominant ($|a_{ii}| > |a_{ik}|$, $\forall\ k \neq i$ and $\forall\ i$).

b) Gauss-Seidel method: always converges when either

   * A is strictly diagonally dominant, or

* A is a positive definite matrix.

In addition, if $J=D^{-1}E+D^{-1}F$ is non-negative, the Gauss-Seidel method converges faster than Jacobi's method.

c) Successive over-relaxation: this method always converges when A is positive definite or when the relaxation parameter "w" is such that:

$$0 < w < 2$$

In practice, the relaxation parameter must be found through a trial and error procedure, within its range of convergence.

The successive over-relaxation method converges faster than Gauss-Seidel, provided that $D^{-1}E$ and $D^{-1}F$ are non-negative matrices.

In general, although Jacobi and Gauss-Seidel methods may converge, their rate of convergence can be quite slow and, due to rounding errors, the whole process can be spoiled.

A.4.7. Standard conjugate gradient method for the solution of linear systems of equations.(Hestenes and Stiefel, 1952).

When A is a (nxn) symmetric and positive definite matrix, the solution of the model problem Ax = b can be understood as a minimization problem.

The main idea behind this method comes from the fact that the quadratic function:

$$Q = (1/2)\ \underline{z}^T\ A\ \underline{z} - \underline{z}^T\ \underline{b} \qquad (153)$$

has a minimum which is exactly the solution of the model linear problem A z = b. Then, instead of solving the original problem,

we solve:

$$\text{minimize } (1/2)\ \underline{x}^T A\ \underline{x} - \underline{x}^T\ \underline{b} \qquad (154)$$

The method was developed by Hestenes and Stiefel (1952), and is basically an unconstrained minimization algorithm, where the successive direction vectors, are conjugate with respect to the matrix A and parallel to the error vectors (difference between true and computed $\underline{x}$'s).

We shall introduce the algorithm first and review its mathematical background afterwards.

We use $\langle \underline{x}, \underline{y} \rangle$ to denote the inner (scalar) product of two vectors:

$$\langle \underline{x}, \underline{y} \rangle \equiv \underline{x}^T\ \underline{y} = \underline{y}^T\ \underline{x}$$

The standard conjugate gradient method consists in:

1. a) Determine an initial solution vector $\underline{x}^{(o)}$.
   b) Compute the residual *vector:* $\underline{r}^{(o)} = \underline{b} - A\ \underline{x}^{(o)}$
   c) Initialise the counter k=0 and take the direction $\underline{p}^{(o)}=\underline{r}^{(o)}$
2. a) Compute the auxiliary vector $\underline{\mu}^{(k)} = A\ \underline{p}^{(k)}$.
   b) Compute the scalar $\alpha_k = \langle \underline{r}^{(k)}, \underline{p}^{(k)} \rangle / \langle \underline{p}^{(k)}, \underline{\mu}^{(k)} \rangle$
   c) Compute the new vector $\underline{x}^{(k+1)} = \underline{x}^{(k)} + \alpha_k\ \underline{p}^{(k)}$
   d) Compute the new residual $\underline{r}^{(k+1)} = \underline{r}^{(k)} - \alpha_k\ \underline{\mu}^{(k)}$
   e) Check for convergence: is $||\underline{r}^{(k+1)}|| \leq \epsilon$?
      If yes, stop.
      If not, continue
      ϵ is the user specified accuracy.
3. a) Compute the scalar $\beta_k = \langle \underline{r}^{(k+1)}, \underline{\mu}^{(k)} \rangle / \langle \underline{p}^{(k)}, \underline{\mu}^{(k)} \rangle$

b) Compute $\underline{p}^{(k+1)} = \underline{r}^{(k+1)} - \beta_k \underline{p}^{(k)}$

c) Increase k by 1 and go to step 2.a)

Thus, the algorithms builds up a sequence of vectors:

$$\underline{x}^{(o)}, \underline{x}^{(1)}, \underline{x}^{(2)}, \ldots . , \underline{x}^{(n)}$$

which converges to the solution of the linear system $A \underline{x} = \underline{b}$

As we shall see in the next paragraphs, it is implicit in the algorithm that consecutive directions $\underline{p}^{(k)}$ and $\underline{p}^{(k+1)}$ are A-conjugate. This property guarantees that the solution can be reached in at most "n" steps, where "n" is the order of A, provided that the calculations are carried out in exact arithmetic. Then, at least in theory, this method fits into the category of the direct methods but, in practice, since rounding errors are present in all calculations, its behaviour corresponds to an iterative method.

Now we shall review the mathematical concepts behind the algorithm:

In the first step of the algorithm an initial solution vector $\underline{x}^{(o)}$ has to be provided; usually a null vector is the easiest starting point, but if a better approximation is available, the algorithm will find the final solution quicker.

Then, the algorithm moves the solution vector from $\underline{x}^{(k)}$ to $\underline{x}^{(k+1)}$ following the direction $\underline{p}^{(k)}$. The algorithm finds the best step size in the direction $\underline{p}^{(k)}$ via a one-dimensional optimisation of the step length; this is done by minimizing the objective function in that direction:

$$\underset{\alpha}{\text{minimize}} \; (1/2)(\underline{x}^{(k)}+\alpha\underline{p}^{(k)})^T A(\underline{x}^{(k)}+\alpha\underline{p}^{(k)}) - (\underline{x}^{(k)}+\alpha\underline{p}^{(k)})^T\underline{b} \qquad (155)$$

On differentiating equation (155) to determine the optimum value of α, say α_k, we get:

$$A\ (\underline{x}^{(k)} + \alpha_k\ \underline{p}^{(k)}) - \underline{b} = \underline{0}$$

then,

$$A\ \underline{x}^{(k)} + \alpha_k\ A\ \underline{p}^{(k)} = \underline{b}$$

or

$$\alpha_k\ A\ \underline{p}^{(k)} = \underline{b} - A\ \underline{x}^{(k)} = \underline{r}^{(k)} \qquad (156)$$

where $\underline{r}^{(k)}$ is the residual vector at the k-th iteration.

Premultiplying equation (156) by $\underline{p}^{(k)}$ (using the inner product) and recalling that $\langle \underline{x}, \underline{y} \rangle = \langle \underline{y}, \underline{x} \rangle$, we can determine α_k:

$$\alpha_k\ \langle \underline{p}^{(k)},\ A\ \underline{p}^{(k)} \rangle = \langle \underline{r}^{(k)}, \underline{p}^{(k)} \rangle$$

finally,

$$\boxed{\alpha_k = \frac{\langle \underline{r}^{(k)}, \underline{p}^{(k)} \rangle}{\langle \underline{p}^{(k)},\ A\ \underline{p}^{(k)} \rangle}} \qquad (157)$$

which is the scalar we compute in step 2.b) of the algorithm.

In the computer implementation of the algorithm we do not need explicitly the matrix A, only its product by the vector $\underline{p}^{(k)}$, which is kept in an auxiliary vector, say $\underline{\mu}^{(k)}$.

Now we are able to update the vector of unknowns:

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} + \alpha_k\ \underline{p}^{(k)} \qquad (158)$$

and the new residual:

$$\underline{r}^{(k+1)} = \underline{b} - A\ \underline{x}^{(k+1)} \qquad (159)$$

which can be simplified in order to avoid the product $A\ \underline{x}^{(k+1)}$ because, on introducing $\underline{x}^{(k+1)}$ from equation (158) into equation (159), we get:

$$\underline{r}^{(k+1)} = \underline{b} - A\ \{\ \underline{x}^{(k)} + \alpha_k\ \underline{p}^{(k)}\ \}$$

or

$$\underline{r}^{(k+1)} = \underline{b} - A\,\underline{x}^{(k)} - \alpha_k\,A\,\underline{p}^{(k)}$$

then

$$\underline{r}^{(k+1)} = \underline{r}^{(k)} - \alpha_k\,A\,\underline{p}^{(k)} \qquad (160)$$

This means that we can overwrite the residual vector $\underline{r}^{(k)}$ with its new value $\underline{r}^{(k+1)}$ by just subtracting $\alpha_k\,\underline{\mu}^{(k)}$ [since we stored the product $A\,\underline{p}^{(k)}$ in the auxiliary vector $\underline{\mu}^{(k)}$]. In so doing, we have avoided the explicit computation of the product $A\underline{x}^{(k+1)}$ in equation (159). Equation (160) corresponds to step 2.d) of the algorithm.

If convergence is not achieved in step 2.e), we need to generate a new direction for the improvement of the solution; because we are looking for a new direction A-conjugate with the previous one; this means that we need:

$$\langle \underline{p}^{(k+1)},\ A\,\underline{p}^{(k)} \rangle = 0 \qquad (161)$$

We shall find the new direction as a linear combination of the residual and the previous direction, i.e.:

$$\underline{p}^{(k+1)} = \underline{r}^{(k+1)} - \beta_k\,\underline{p}^{(k)}$$

Imposing the A-conjugate condition [equation (161)], we get:

$$\langle \underline{r}^{(k+1)} - \beta_k\,\underline{p}^{(k)},\ A\,\underline{p}^{(k)} \rangle = 0$$

or

$$\langle \underline{r}^{(k+1)},\ A\,\underline{p}^{(k)} \rangle - \beta_k\,\langle \underline{p}^{(k)},\ A\,\underline{p}^{(k)} \rangle = 0$$

which gives us the value of β_k we are looking for:

$$\boxed{\beta_k = \frac{\langle \underline{r}^{(k+1)},\ A\,\underline{p}^{(k)} \rangle}{\langle \underline{p}^{(k)},\ A\,\underline{p}^{(k)} \rangle}} \qquad (162)$$

where again we can use the stored auxiliary vector $\underline{\mu}^{(k)}$ instead of $A\,\underline{p}^{(k)}$.

It is at this stage that rounding errors affect the finite

termination property of the conjugate gradient method, since with exact arithmetic the vectors $\underline{p}^{(k)}$ and $\underline{p}^{(k+1)}$ are exactly A-conjugate (or A-orthogonal), but in the presence of rounding errors orthogonality is lost.

The amount of work needed at each step of the algorithm is mainly due to the product A $\underline{p}^{(k)}$, which makes the method very attractive from the computational point of view; the rest of the computations are mainly scalar operations. Only 4 vectors need to be stored: $\underline{x}^{(k)}$, $\underline{r}^{(k)}$ [which can overwrite the original right hand side vector $\underline{b}$], $\underline{p}^{(k)}$ and $\underline{\mu}^{(k)} = A\ \underline{p}^{(k)}$.

These features have gained the conjugate gradient method a reputation for large sparse linear systems in reduced storage computers. In practice, the method is not recommended for dense (non-sparse) matrices.

The conjugate gradient method was developed in 1952, when it was originally regarded as a direct method, but it was abandoned shortly afterwards, due to convergence problems created by the lack of orthogonality. These problems and a comparison of different versions of the algorithm were analysed by Reid (1971), and interest in the method , now regarded as an iterative algorithm was renewed . Of the several possible versions of the conjugate gradient algorithm we have presented one of the most widely used ones.

Gambolatti and Perdon (1984) compared the conjugate gradient method with other iterative methods and offered a geometrical interpretation of the algorithm, which is summarised in Fig.

A.37. for 2 and 3-dimensional problems.

Parlett (1980) has shown the mathematical relationship between conjugate gradient methods and the Lanczos algorithm for solving eigenvalue problems, the conjugate gradient being a particular case of the Lanczos algorithm. We have used a Lanczos implementation for solving the linear systems in the early stages of the development of the gradient method for pipe distribution network analysis [Lewis (1977)]; as expected, for normally-conditioned problems, the convergence of the preconditioned conjugate gradient was faster than the Lanczos implementation by a factor of about two.

Nevertheless, the Lanczos algorithm is expected to behave better than the conjugate gradient for ill-conditioned problems. For more details on the Lanczos method see, for example, Scott (1981).

A.4.8. Preconditioning.

Any direct or iterative method for the solution of linear systems can be improved in its convergence rate via a technique known as "preconditioning".

In an earlier section (The effect of rounding errors in the solution of linear systems), we defined the condition number of a matrix as the norm product $||A||.||A^{-1}||$, and we associated this number with the stability of the solution of the linear system. Matrices with low condition numbers (about 1) are said to be "well-conditioned" matrices, whereas those with larger condition

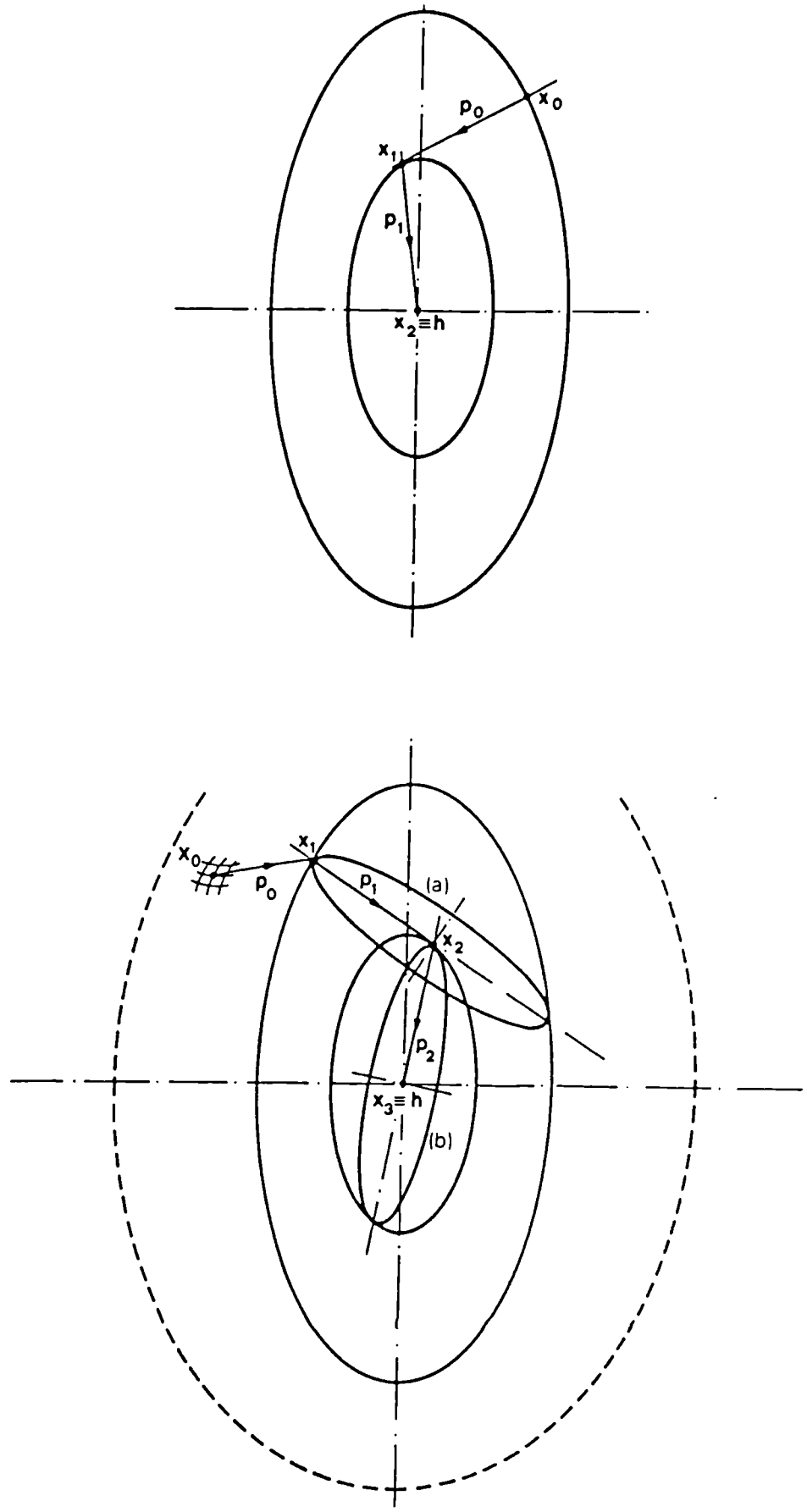


Fig. A.37. Geometric interpretation of the convergence of the conjugate gradient algorithm for n=2 and n=3, from Gambolatti and Perdon (1984), solving A h = b

numbers (say, 10^4 or even greater) are considered as "ill-conditioned". We may get convergence problems with ill-conditioned matrices.

The main idea behind preconditioning is to transform an ill-conditioned problem into a well-conditioned one, via a suitable modification of the original system. Algebraically, this is done by a pre and/or post-multiplication of the original system of equations by some suitable matrices, which are referred to as preconditioning matrices (or simply preconditioners).

Clearly, the aim is to produce a new linear system, with a better behaviour from the rate of convergence viewpoint. A complete review of preconditioning methods, applied both to direct and iterative methods, can be found in Evans (1983).

We shall concentrate on the application of preconditioning to iterative methods.

The concept of preconditioning can be formalised as follows: instead of solving our original model problem $A\,x = b$, solve either

$$(M\,A)\,x = (M\,b) \tag{163}$$

or

$$(N\,A\,N^T)(N^{-T}\,x) = (N\,b) \tag{164}$$

where M and N are square preconditioning matrices.

The form of the preconditioning given by equation (163) seems simpler, but the main advantage of (164) is that, if A is symmetric, the new matrix of coefficients $(N\,A\,N^T)$ remains symmetric, although in that case we need to determine a

transformed unknown variable, say $y = N^{-T} x$. Apart from our main objective, in the sense that the new system has to be better conditioned than the original one, we look for a new system which may be eventually easier to solve as well.

There are as many preconditioning methods as matrices M and N can be imagined. From an ideal point of view, the best choice for M in equation (163) would be $M = A^{-1}$, since then the solution for the vector x could be obtained in one single step. In general, with an adequate selection of M and N, the preconditioned problem will converge faster than the original one but, unfortunately, there is no simple rule to determine a general purpose preconditioning scheme.

Some possible preconditioning matrices are:

a) Diagonal matrices:

For example $M= \text{diag}(m_{11}, m_{22}, \ldots, m_{nn})$ with $m_{ii} = 1/a_{ii}$. This kind of preconditioning matrix is more effective in highly diagonally dominant systems, since in this case (M A) is close to the identity matrix.

b) Complete factorizations of A:

Different kinds of factorizations of A can be used as preconditioners, since in general they are easier to invert than the original matrices. Thus, L U, $L L^T$, L D U, L D L^T, or Cholesky complete factorizations can be used as preconditioners [see section A.2.3.], because all of them produce easily invertible matrix factors.

c) Incomplete factorizations of A:

Since complete factorizations lead to loose the sparseness of the original system (because of fill-in), this kind of factorization does not seem adequate when dealing with large sparse linear systems; instead, an incomplete factorization is usually the correct answer to this problem. In an incomplete factorization we look for factors which have a similar sparseness pattern in relation to the original matrix. That is to say, we are now dealing with factorizations of the form:

$$A \approx L_S \, U_S$$

or

$$A \approx L_S \, D \, U_S$$

where L_S and U_S correspond to sparse matrix factors, whose product approximates the original matrix A. Note that now we are just asking for an approximation to A.

In order to produce an incomplete factorization, there are at least two strategies:

i) Fixed pattern scheme: in this case the original matrix is decomposed in factors that have exactly the same sparsity pattern as A.

Kershaw (1978), used successfully the incomplete Cholesky factorization given by:

$$A = L \, D \, L^T + E \qquad (165)$$

where A = symmetric and positive definite matrix.

L = lower triangular matrix with the same sparsity as A.

D = diagonal matrix

E = error matrix containing all the non-zero elements which lie in the positions where A has zero values.

Then, the following approximation holds:

$$A \approx L\ D\ L^T \tag{166}$$

Of course, in the case of symmetric positive definite systems, the same idea can be applied to the Cholesky "square root" factorization:

$$A = L\ L^T + E \tag{167}$$

with the approximation:

$$A \approx L\ L^T \tag{168}$$

The algorithm to perform this last factorization can be derived easily, from equation (168), expressed term by term:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & \cdots & a_{2n} \\ & & & \vdots \\ \text{symmetric} & & & a_{nn} \end{bmatrix} \approx \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & \cdots & l_{n2} \\ & & & \vdots \\ & & & l_{nn} \end{bmatrix} \tag{169}$$

The first row of L^T is determined multiplying the first row of L by each column of L^T and comparing element by element with the coefficients of A:

$$a_{11} = l_{11}\ l_{11} \quad \text{--->} \quad l_{11} = \sqrt{a_{11}} \tag{170}$$

$$\begin{matrix} a_{12} \\ \vdots \end{matrix} = \begin{matrix} l_{11} \\ \vdots \end{matrix}\ l_{21} \quad \text{--->} \quad \begin{matrix} l_{21} \\ \vdots \end{matrix} = \begin{matrix} a_{12}/l_{11} \\ \vdots \end{matrix}$$

$$a_{1i} = l_{11}\ l_{i1} \quad \text{--->} \quad l_{i1} = a_{ii}/l_{11} \quad \forall\ i=2,3,..,n \tag{171}$$

Equations (170) and (171) give us the first row of the matrix L^T.

The diagonal elements of L^T are determined by multiplying row "i" of L and column "i" of L^T:

$$a_{ii} = l_{i1}\, l_{i1} + l_{i2}\, l_{i2} + \ldots + l_{ii}\, l_{ii}$$

then

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \qquad (172)$$

The i-th row of L^T is determined by multiplying the i-th row of L by each column of L^T:

$$a_{ij} = l_{i1}\, l_{j1} + l_{i2}\, l_{j2} + \ldots + l_{i(i-1)}\, l_{j(i-1)}$$

then

$$l_{ji} = \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik}\, l_{jk}\right) / l_{ii} \qquad \forall j=(i+1),(i+2),..,n \qquad (173)$$

The main advantage of this decomposition lies in the fact that, when a sparse data structure is used to store A , then we only need to store the numerical part of L^T, since the same data structure (row and column indices) describe both A and L^T. Of course, the symmetry of A implies that only half of the matrix is actually stored, i.e. A is also stored like a triangular matrix, and the symmetry is handled implicitly. As a result, great economy in storage is achieved, although an auxiliary (integer) vector is required to scan L^T by columns, when a row-wise data structure is being used.

A couple of examples are shown to illustrate some features of this factorization scheme:

Example a)
****

$$A = \begin{bmatrix} 8.3 & -3.2 & 0.0 & -2.7 \\ -3.2 & 13.7 & -5.6 & 0.0 \\ 0.0 & -5.6 & 10.5 & -2.2 \\ -2.7 & 0.0 & -2.2 & 9.2 \end{bmatrix}$$

$$L^T = \begin{bmatrix} 2.881 & -1.111 & 0.0 & -0.937 \\ & 3.531 & -1.586 & 0.0 \\ & & 2.826 & -0.779 \\ & & & 2.778 \end{bmatrix}$$

$$L\,L^T = \begin{bmatrix} 8.299 & -3.2 & 0.0 & -2.699 \\ -3.2 & 13.7 & -5.6 & 1.041 \\ 0.0 & -5.6 & 10.499 & -2.200 \\ -2.699 & 1.041 & -2.200 & 9.199 \end{bmatrix}$$

We can see that the only apparent discrepancy seems to be the element in the second row and fourth column of $L\ L^T$ which is 1.041 instead of 0.0.

Example b)
****

$$A = \begin{bmatrix} 16.0 & 0.0 & 0.0 & -2.0 & -1.0 \\ 0.0 & 9.0 & -2.0 & -1.0 & 0.0 \\ 0.0 & -2.0 & 10.0 & 1.0 & 0.0 \\ -2.0 & -1.0 & 1.0 & 20.0 & 1.0 \\ -1.0 & 0.0 & 0.0 & 1.0 & 25.0 \end{bmatrix}$$

$$L^T = \begin{bmatrix} 4.0 & 0.0 & 0.0 & -0.5 & -0.25 \\ & 3.0 & -0.67 & -0.33 & 0.0 \\ & & 3.091 & 0.252 & 0.0 \\ & & & 4.424 & 0.198 \\ & & & & 4.990 \end{bmatrix}$$

$$L\,L^T = \begin{bmatrix} 16.0 & 0.0 & 0.0 & -2.0 & -1.0 \\ 0.0 & 9.0 & -1.999 & -0.999 & 0.0 \\ 0.0 & -1.999 & 9.999 & 0.999 & 0.0 \\ -2.0 & -0.999 & 0.999 & 19.999 & 1.0 \\ -1.0 & 0.0 & 0.0 & 1.0 & 25.0 \end{bmatrix}$$

Now the agreement between A and $L\ L^T$ seems to be better than in example a).

These two examples "suggest" some of the most relevant features of this sort of factorization: they behave better in the case of large matrices, and also when a strong diagonally dominant matrix

is factorized.

ii) Relative magnitude scheme: now the criterion for keeping the factors with a similar sparseness changes and, instead of their relative position with respect to the original non-zeros, some terms of the factorization are "rejected" or "thrown away" because their magnitude is less than a pre-established amount.

Meijerink and Van der Vorst (1977) proposed a decomposition along this line, for symmetric M-matrices (i.e. when all the non-diagonal coefficients are less or equal to zero).

Munksgaard (1980) proposed an incomplete L D L^T factorization of this type, for symmetric positive definite matrices (without extra requirements on its structure).

Ajiz and Jennings (1984), following the same idea, presented a L L^T factorization, though some adjustments are recommended to the diagonal values in order to maintain positive definiteness.

The advantage of this proposition is its flexibility, because it allows a wide set of possible incomplete factorizations to be obtained, ranging from a complete Cholesky decomposition up to one where all off-diagonal terms are rejected. The disadvantage is that it is not possible to predict beforehand the fill-in created, and so it is quite possible to run out of storage during the factorization, unless a dense factor L has been dimensioned, in which case it is obvious that a complete factorization should have been used. Less evident as a disadvantage is the fact that this approach needs a more complex data structure, because the original data structure does not represent the incomplete factor.

Mainly because of storage constraints we prefer the incomplete factorization proposed by Kershaw (1978), though in some cases, if storage is less restrictive, the approach of Ajiz and Jennings (1984) can produce faster execution times.

Chen and Tewarson (1986) have shown how incomplete factorization can be applied to solve some banded shifted coefficient matrices.

A.4.9. Preconditioned (modified) conjugate gradient method for the solution of linear systems.

Although there are as many preconditioned conjugate gradient methods as possible preconditioning matrices, we have followed a scheme proposed by Ajiz and Jennings (1984), but with an incomplete Cholesky factorization based on the "square root" decomposition [see equations (170) to (173)], instead of the incomplete factorization proposed by Ajiz and Jennings (based on the magnitude of the non-zero values). This is exclusively in order to keep the storage needs under control.

The subroutine MA31 of the Harwell Library implements a preconditioned conjugate gradient algorithm, based on the incomplete factorization proposed by Munksgaard (1980).

Given the incomplete Cholesky factorization:

$$A \approx L\, L^T \quad , \tag{174}$$

the "standard" conjugate gradient algorithm is applied to the following preconditioned linear system:

$$(L^{-1} A L^{-T}) (L^T \underline{x}) = (L^{-1} \underline{b}) \tag{175}$$

which is equivalent to solving the system:

$$(L^{-1} A L^{-T}) \underline{y} = \underline{b}^* \qquad (176)$$

where:

$(L^{-1} A L^{-T})$ is the new (preconditioned) symmetric matrix

$\underline{y} = L^T \underline{x}$ is the new transformed unknown

$\underline{b}^* = L^{-1} \underline{b}$ is the new transformed right hand side vector.

Thus, the preconditioned conjugate gradient method is, loosely speaking, no more than the standard conjugate gradient applied over a transformed (preconditioned) linear system. When the solution of the preconditioned system has been obtained for the vector $\underline{y}$, a back substitution is needed to compute the original unknown $\underline{x}$.

From the computational point of view, we do not need to store the transformed matrix $(L^{-1} A L^{-T})$ explicitly, because all we need to do with this matrix is to multiply it by the conjugate direction vector $\underline{p}^{(k)}$, to form the auxiliary vector:

$$\underline{\mu}^{(k)} = (L^{-1} A L^{-T}) \underline{p}^{(k)}$$

which is similar to that computed in step 2.a of the standard conjugate gradient method. As a result, we can build up the auxiliary vector $\underline{\mu}$ in a three stage process:

$$\left.\begin{array}{l} \text{a) Back substitution: solve } L^T \underline{v}^{(k)} = \underline{p}^{(k)} \quad \text{for } \underline{v}^{(k)} \\ \text{b) Premultiplication: compute } \underline{w}^{(k)} = A \underline{v}^{(k)} \\ \text{c) Forward substitution: solve } L \underline{\mu}^{(k)} = \underline{w}^{(k)} \quad \text{for } \underline{\mu}^{(k)} \end{array}\right\} \qquad (177)$$

Then, the preconditioned conjugate gradient algorithm can be carried out in the following steps (compare with the standard method in section 6.7):

0. Compute the corrected right hand side vector $\underline{b}^* = L^{-1}\,\underline{b}$ which is a simple premultiplication.

1. a) Determine an initial solution vector $\underline{y}^{(o)}$.
   ($\underline{y}^{(o)}=\underline{0}$ or a better approximation , if available, see Note 1 at the end of the algorithm).
   b) Compute the residual vector: $\underline{r}^{(o)} = \underline{b}^* - (L\text{-}1AL\text{-}T)\,\underline{y}^{(o)}$
   ($\underline{r}^{(o)}=\underline{b}^*$ if we started with $\underline{y}^{(o)}=\underline{0}$, see Note 1)
   c) Initialise the counter k=0 and take the direction $\underline{p}^{(o)}=\underline{r}^{(o)}$

2. a) Compute the auxiliary vector $\underline{\mu}^{(k)}$ with the three stage process given in equation (177)
   b) Compute the scalar $\alpha_k = \langle\underline{r}^{(k)},\underline{r}^{(k)}\rangle / \langle\underline{p}^{(k)},\underline{\mu}^{(k)}\rangle$ (see Note 2).
   c) Compute the new vector $\underline{y}^{(k+1)} = \underline{y}^{(k)} + \alpha_k\,\underline{p}^{(k)}$
   d) Compute the new residual $\underline{r}^{(k+1)} = \underline{r}^{(k)} - \alpha_k\,\underline{\mu}^{(k)}$
   e) Check for convergence: is $||\underline{r}^{(k+1)}|| \leq \epsilon$?
   If yes, go to step 4.
   If not, continue
   ϵ is the user specified accuracy.

3. a) Compute the scalar $\beta_k = \langle\underline{r}^{(k+1)},\underline{r}^{(k+1)}\rangle / \langle\underline{r}^{(k)},\underline{r}^{(k)}\rangle$ (see Note 3).
   b) Compute $\underline{p}^{(k+1)} = \underline{r}^{(k+1)} + \beta_k\,\underline{p}^{(k)}$ (see Note 4).
   c) Increase k by 1 and go to step 2.a)

4. Solve $L^T\,\underline{x} = \underline{y}$ for $\underline{x}$ via a back substitution process and stop, because $\underline{x}$ is the solution of the original system (see Note 5).

Notes:
=====

1. When solving iteratively a system of non-linear equations, via a sequence of linear systems, it appears logical to keep the solution of the k-th linear system (say $\underline{y}^{(k)}$), and use it as an initial solution vector for the (k+1)-th linear system.

We have applied this approach in the context of the gradient method for the water distribution analysis problem but, when solving systems of more than 165 unknowns we reached a break even point, balancing the time saved starting with the vector $\underline{y}$ from the previous (non-linear) iteration with the additional work involved in computing the new matrix (L^{-1} A L^{-T}) via the three stage process (177). In this case, we recommend a start with the initial solution $\underline{y}^{(o)}=\underline{0}$, as in step 1.a), i.e. $\underline{r}^{(o)}=\underline{b}$*

Gambolatti (1980) suggested that further improvement in the convergence rate of the preconditioned conjugate gradient can be obtained via the pre-computation of the initial solution by the Newton iterative algorithm, but we have not experimented with his recommendation.

2. There is an alternative formulation of the algorithm, for step 2.b), which is equivalent to that used in the standard conjugate gradient algorithm:

Compute $\alpha_k = \langle \underline{r}^{(k)}, \underline{p}^{(k)} \rangle / \langle \underline{p}^{(k)}, \underline{\mu}^{(k)} \rangle$

3. Here there is an alternative formulation for step 3.a), equivalent to that used in the standard method:

Compute $\beta_k = \langle \underline{r}^{(k+1)}, \underline{\mu}^{(k)} \rangle / \langle \underline{p}^{(k)}, \underline{\mu}^{(k)} \rangle$

4. If the alternative β_k of Note 3 has been used, the new direction vector $\underline{p}^{(k+1)}$ in step 3.b) must be computed with:

$$\underline{p}^{(k+1)} = \underline{r}^{(k+1)} - \beta_k \underline{p}^{(k)}$$

i.e. only a change of sign.

5. Obviously, because the preconditioned conjugate gradient solved the transformed problem for the auxiliary unknown $\underline{y} = L^T \underline{x}$, we need to find the value of the original unknown $\underline{x}$ via a back substitution process, which is done in step 4.

We shall not repeat here the reasons for the computation of the optimal parameters α^k and β^k, because they remain the same as those explained in the case of the standard conjugate gradient.

Gambolatti and Perdon (1984) interpreted the conjugate gradient method in geometric terms. The standard conjugate gradient involves the minimization of a functional which represents a hyper-ellipsoid in an n-dimensional space; the more "spherical" this hyper-ellipsoid the faster is the convergence of the algorithm. Thus, the preconditioning can be interpreted as an effort to turn the hyper-ellipsoid into a sphere.

The development of conjugate gradient methods for the solution of linear systems of equations is still a very active research field. Sartoretto (1984) compared the preconditioned conjugate gradient with other iterative techniques used in microcomputers, using problems coming from the finite element analysis of structures. Some tests demonstrating the goodness of preconditioning are presented in Jackson and Robinson (1985). Samuelsson, Wiberg and Bernspâng (1986) compared some preconditioned conjugate methods with direct methods for structural mechanics problems.

APPENDIX B

DERIVATION OF THE KRIGING ESTIMATOR EQUATIONS

B.1. Introduction.

The estimation technique known as Kriging owes its name to D. G. Krige, a South African geologist who, in the early fifties, approached the problem of estimating the ore content in a deposit, based on limited sampling.

G. Matheron and co-workers in the School of Mines of Paris at Fontainebleu (France), in the sixties and seventies, extended the ideas of Krige, introducing the terms "geostatistics" and "regionalized variables". Geostatistics refers to the application of statistic concepts to natural phenomena, whereas regionalized variable stands for variables describing phenomena which are spatially and, eventually, temporally distributed, with an underlying structure, the structural characteristics being a consequence of the genesis of the phenomena.

So far, geostatistics has been applied to a vast variety of fields, some of them well apart from the original mining engineering: the petroleum industry, geophysics, geotechnical engineering, forestry, geochemistry, pollution problems, etc. [Huijbregts (1973)]. In water resources engineering, geostatistics has been used to study the spatial variation of rainfall, to interpolate and to determine average values of rainfall depths within a basin [Delhomme, (1978)], to the

identification of parameters in groundwater flow [Hoeksema and Kitanidis (1983), (1984) and (1985)] and other problems .

We shall review the statistical and mathematical background of Kriging, looking at its application to the problem of piezometric head estimation in the context of water supply distribution networks. Most of the concepts presented here can be found, with much more detail, in the publications by de Marsily (1986) and Journel and Huijbregts (1978), but they are reviewed here for completeness.

The geostatistical process is usually carried out in a two-stage procedure:

a) Representation of the model structure: which accounts for the construction of an experimental variogram, based on the field data, and the determination of the best analytical model corresponding to that experimental variogram. This is also known as the "statistical inference" or the "structural analysis" stage.

b) The estimation of the unmeasured states (Kriging): based on the previously determined structure, Kriging provides the best linear unbiased estimates of the unknown states.

Both stages involve completely different activities and require different techniques. In the first stage (statistical inference), we are dealing with an analytical process, trying to specify in statistical terms (through the variogram) the structure of the phenomenon under study; this is clearly an activity which demands

a great deal of judgement, even though the computer can be used in an interactive way to ease the calculations involved. The second stage (Kriging itself) is the numerical part of the process and is usually carried out with the help of the computer, since it basically involves the assembly and solution of linear systems of equations. The success of the overall process relies heavily on the ability of the modeller to incorporate the structure of the phenomenon into the variogram model and, to a lesser extent, into the Kriging itself.

We shall review each stage with some more detail, but first some notation needs to be introduced.

Because we are dealing with a water supply distribution network, which is physically spread under the ground level, the geometrical space over which the problem is defined is a two-dimensional space. It can be argued that the alternative is a one-dimensional space, since we are dealing with pipes, which are basically one-dimensional objects, but we believe the two-dimensional representation is more appropriate because we are (in general) concerned with looped networks, in which case the heads and flows are not dependent on the properties of one isolated pipe, but depend on the interaction of the different elements of the network, which do not necessarily need to be located in a one-dimensional arrangement. In fact, looped water distribution networks are essentially two-dimensional arrangements of pipes and nodes, whereas open (branched) water distribution networks are easily handled in a one-dimensional fashion.

Nevertheless, we restrict ourselves to the study of the piezometric heads over a discrete set of points in the two-dimensional space, these points are the nodes of the water distribution network. Occasionally, we may also consider other points, located in the pipes joining those nodes, but in either case we are dealing with a discrete phenomenon rather than a continuous one. As a result, head measurements are available at some points of the space and head estimates (based on those measurements) are required at unmeasured points.

Let $\underline{x} = (x_1, x_2)^T$ be the two-dimensional vector representing the location, projected into an horizontal plane, of the nodes and any point within the network.

Since we are interested in the description of the piezometric head plane (surface actually) we introduce the state variable H, representing the piezometric head at any point of the network, thus:

$$(\underline{x}, H) = (x_1, x_2, H)^T$$

represents a point in the piezometric plane at coordinates (x_1, x_2) with a piezometric head of value H.

Due to the fact that we are modelling the phenomenon as a stochastic process, instead of a deterministic one, the previous definition is better represented as a <u>random function</u> , say $Z(\underline{x},H)$, whereas $Z(\underline{x}_o,H)$ denotes a <u>random variable</u> at the particular location x_o and $Z(x,H_1)$ denotes a particular <u>realisation</u> of the random function.

In order to simplify the notation of the random variables, sometimes we may drop the state variable (H) or, even simpler, we may use sub-indices to represent the value of a random variable at a particular location; thus, the following expressions are all equivalent:

$$Z(\underline{x}_i,H) \quad <==> \quad Z(\underline{x}_i) \quad <==> \quad Z_i$$

B.2. Statistical inference.

In the geostatistical sense, the fact that $Z(\underline{x}_1,H)$ is a "regionalized variable" (i.e. it has an underlying structure or "spatial distribution"), means that there is a correlation between $Z(\underline{x}_1,H)$ and another variable $Z(\underline{x}_2,H)$, separated by a distance $\underline{d} = \underline{x}_1 - \underline{x}_2$.

The variability of the regionalized variables is characterised by a statistical function referred to as the semi-variogram, which is defined as:

$$\text{Semi-variogram: } \Gamma(\underline{x}_1,\underline{x}_2) \equiv \tfrac{1}{2}\, E\, [\, \{Z(\underline{x}_1,H) - Z(\underline{x}_2,H)\}^2\,] \qquad (1)$$

where E denotes mathematical expectation.

Some authors [de Marsily (1986), for example], refer to the semi-variogram as simply the variogram, the only difference being the scalar factor ½ applied to the expectation in the previous definition. We shall try to keep the present notation (semi-variogram) since it does not lead to any confusion, although we may refer to it occasionally as simply the variogram.

Hence, in theory, the semi-variogram depends on $\underline{x}_1$ and $\underline{x}_2$, and its estimation requires the availability of a great number of realisations: [$Z(\underline{x}_1,H_1)$, $Z(\underline{x}_2,H_1)$] , [$Z(\underline{x}_1,H_2)$, $Z(\underline{x}_2,H_2)$], ... , [$Z(\underline{x}_1,H_r)$, $Z(\underline{x}_2,H_r)$] of the regionalized variable.

In practice, these numerous realisations are not available and we only have <u>one</u> realisation at different points of the space; physically, this corresponds to a set of head measurements at the different nodes in the network.

In order to be able to estimate the semi-variogram from the measured data some simplifying assumptions are made. Firstly, the random process (or random function) is assumed to be ergodic, which allows us to use the information from the unique realisation available to determine the properties of the ensemble of all possible realisations. Secondly, assumptions with different degree of stringency are made with respect to the behaviour of the statistics of different order (mean, variance and higher order moments) associated with the random process. The most stringent assumption used is that of <u>stationarity</u>, which means that all the statistics, including higher order moments, are assumed to be constant. Because for many practical problems this hypothesis is too demanding and far from the real characteristics of the random process under study, a more relaxed assumption known as <u>weak stationarity</u> (or second-order stationarity) is often used, whereby only the mean and variance are assumed to be invariant. Further relaxation in the stationarity of the random process occurs when the weak stationarity is not required for the random process itself, but

for its increments, which is the intrinsic hypothesis. We shall follow the usual procedure of deriving the Kriging equations for different stationarity conditions, starting with the most stringent one (stationarity) and we shall relax it successively to second-order stationarity and second-order stationarity for the increments (intrinsic hypothesis). However, the question concerning which conditions are applicable to a particular problem has to be answered from the data available and from a physical understanding of the process being studied.

On assuming that the intrinsic hypothesis holds (the weakest assumption so far), it is possible to estimate the semi-variogram from the measurement data. Indeed, the intrinsic hypothesis formally establishes that:

$$E[Z(\underline{x}_1,H) - Z(\underline{x}_2,H)] = 0 \qquad (2)$$

$$\text{var}[Z(\underline{x}_1,H) - Z(\underline{x}_2,H)] = 2\,\Gamma(\underline{d}) \qquad (3)$$

where $\underline{d} = \underline{x}_1 - \underline{x}_2$.

With these hypotheses, the semi-variogram $\Gamma(\underline{d})$ can be computed, because from (3):

$$\Gamma(\underline{d}) = \tfrac{1}{2}\,\text{var}[Z(\underline{x}_1,H) - Z(\underline{x}_2,H)] \qquad (4)$$

which can be expanded to:

$$\Gamma(\underline{d}) = \tfrac{1}{2}\,E[\{Z(\underline{x}_1,H) - Z(\underline{x}_2,H)\}^2] - \{E[Z(\underline{x}_1,H)-Z(\underline{x}_2,H)]\}^2 \qquad (5)$$

The last term of (5) vanishes because of the stationarity assumption on the mean of the increments [equation (2)], hence:

$$\Gamma(\underline{d}) = \tfrac{1}{2}\,E[\{Z(\underline{x}_1,H) - Z(\underline{x}_2,H)\}^2] \qquad (6)$$

Equation (6) allows us to estimate the semi-variogram from the data, as the arithmetic mean of the squared differences between two pairs of measurements $[Z(\underline{x}_1,H),Z(\underline{x}_2,H)]$:

$$\gamma^*(\underline{d}) = \frac{1}{2}\left\{ \frac{1}{N(\underline{d})} \sum_{i=1}^{N(\underline{d})} [\, Z(\underline{x}_1,H) - Z(\underline{x}_2,H)\,]^2 \right\} \qquad (7)$$

where $N(\underline{d})$ represents the number of pairs of data at distance "d".

For computational purposes we define classes of distance between pairs of measurements, for example:

Class	Between distances (m)
C_1	(0 , 100]
C_2	(100, 200]
C_3	(200, 300]
C_4	(300, 400]
:	: :

and, for each class, we simply average the squared differences: $\frac{1}{2}\{ [\, Z(\underline{x}_1,H) - Z(\underline{x}_2,H)\,]^2 \}$, for all $\underline{x}_1$ and $\underline{x}_2$ belonging to this particular class. The process can be carried out in a tabular format [de Marsily (1986)], as presented in Table B.1.

Table B.1. Computation of the experimental semi-variogram.

Class	C_1	C_2	C_3	...	C_k	...
Number of pairs: N(d)	N_1	N_2	N_3	...	N_k	...
Average distance: d	d_1	d_2	d_3	...	d_k	...
Average of $\{1/(2N(d))\}[Z_i-Z_j]^2$	a_1	a_2	a_3	...	a_k (*)	...

(*)
$$a_k = \frac{1}{2\,N_k} \sum_{i=1}^{N_k} [Z_i - Z_j]^2$$

The estimated semi-variogram $\gamma^*(\underline{d})$ is the resulting plot with the average distance as abscissa and the corresponding average of the squared differences as ordinate, as shown in Fig. B.1.

The estimated semi-variogram is then a broken line obtained by joining the successive points produced by the previous computation, i.e. we get as many points as distance classes.

Some details on the semi-variogram become relevant:

* Note that for a set of "n" measurements, ,we get n(n-1)/2 pairs; each one has to be classified according to its distance (d) and allocated into the corresponding distance class (C_i). This is without considering the direction, in which case the number of pairs is reduced, since we only have to include the pairs belonging to a certain direction; this implies that a sectorization of the geometrical space is needed for computational purposes.

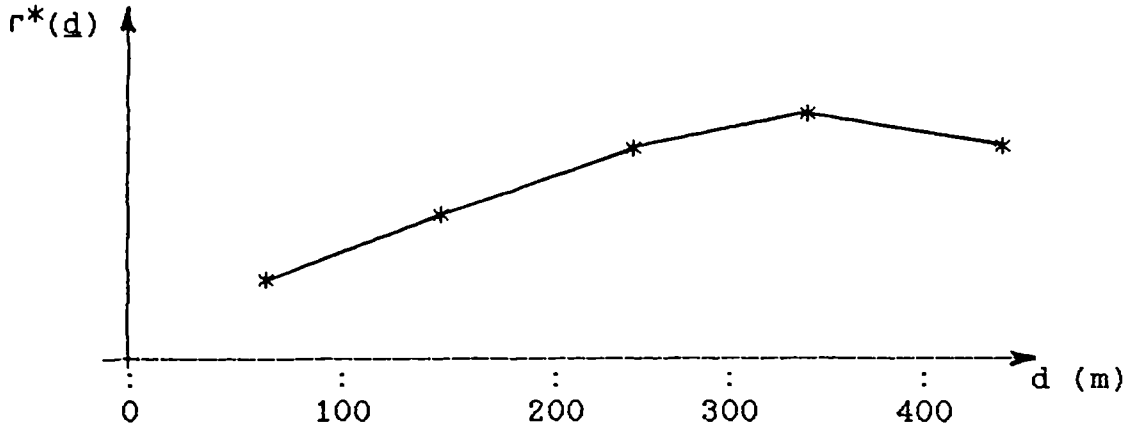


Fig. B.1. Estimated semi-variogram.

* Because the number of pairs decreases with the distance, it means that the uncertainty in the estimated semi-variogram increases with the distance. Because of that, sometimes it can be convenient to discard the pairs too distant from each other; this implies the use of a "moving neighbourhood", made up of all the points surrounding the point to be Kriged, which are contained within a circle of a pre-specified radius, thus reducing the uncertainty in the semi-variogram at longer distances. Also, some improvement in the estimation of the semi-variogram in the shorter distances can be obtained by defining the distance classes with a variable length, taking into consideration the number of pairs included.

* Some data points (outliers) can induce great errors in the estimated semi-variogram, and it might be necessary to eliminate some of them in order to stabilise the numerical computations.

The practicalities in the semi-variogram estimation are key factors in its robustness and representativeness, as a "summary" of the spatial correlation between the regionalized variables; Armstrong (1984) discusses a number of the common causes of problems in estimating the semi-variogram and also proposes the corresponding corrective measures.

Some relevant features of the semi-variogram need to be stressed:

a) Continuity: Due to the spatial continuity of the regionalized variable, the semi-variogram is also continuous. For a particular direction in the distance vector $\underline{d}$, and because of

the increments vanish at a zero distance, the semi-variogram starts, in theory, at the origin [$\Gamma(0)=0$] and it increases with the distance (absolute value of the distance). The way in which the semi-variogram increases with the distance is an indicator of the degree of spatial correlation of the regionalized variable.

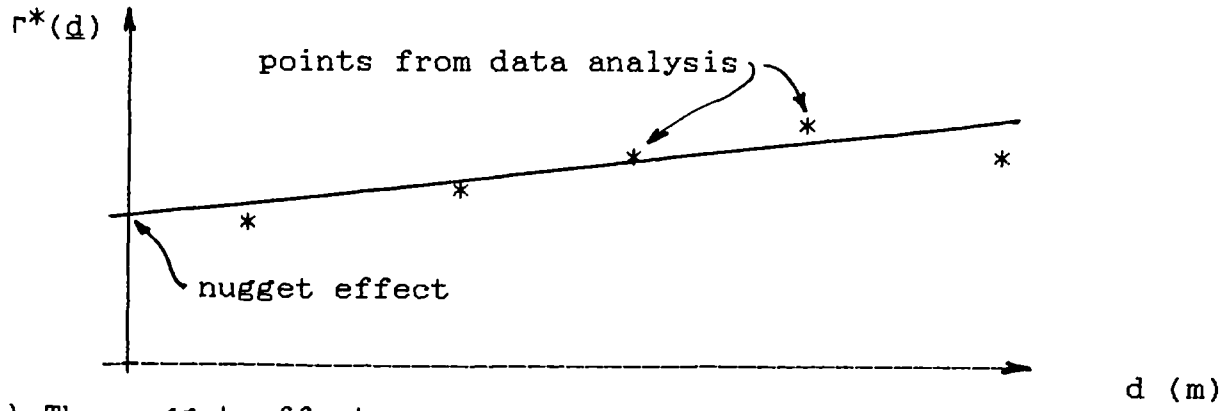
b) Range and sill: we restrict ourselves to a particular direction in the distance vector, and if we carry on increasing the distance (modulus), we may reach a point where the correlation no longer exists; the distance corresponding to this situation, say "r", is referred to as the "range" of the semi-variogram, while its associated value, $\Gamma(r)$ say, is known as the "sill". Changing the direction in the distance vector normally implies that different ranges and sills can be obtained for the same regionalized variable; in the three-dimensional case, in mining or groundwater for example, we may have two completely different values for the range and sill, depending on whether we are moving in an horizontal or a vertical direction. In water supply distribution networks, we may expect different semi-variograms in the case, for example, of long networks where the flow pattern in the longitudinal direction is very different with respect to the flow distribution in the transverse direction. The study of the effect of anisotropies in the regionalized variable on the semi-variogram is important in the statistical inference process, in order to obtain a robust variogram.

c) Existence of the sill: As we implied before, the presence of a range and sill in the semi-variogram may be possible. In fact, there are some cases where the semi-variogram does not reach a

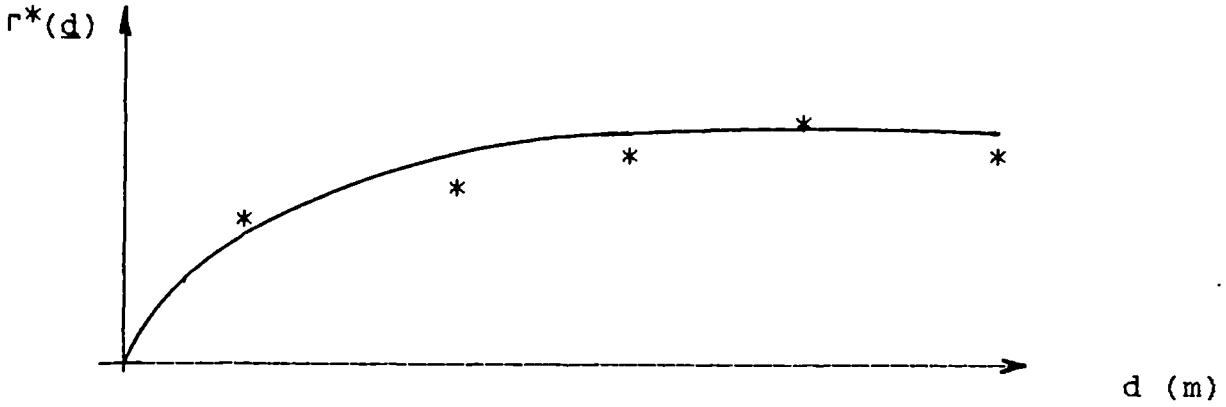
sill for any value of the distance; this implies that the regionalized variable has an infinite variance, and only the introduction of additional assumptions (intrinsic hypothesis) can allow us to study the behaviour of such a phenomenon. We might expect the lack of a sill in the case of water supply distribution networks with a steep piezometric plane.

d) Behaviour near the origin: In theory, the value of the semi-variogram at the origin (distance=0) is null, but this may not become clear from the measurement data. First of all, because we only compute the value of points of the semi-variogram centred at the average distance of each class, we may not get points close enough to the origin (depending on how we have chosen the class length). This may produce an apparent "jump" in the semi-variogram near the origin, as shown in Fig. B.2., this is known as the "nugget effect".

The second practical issue leading to the nugget effect is related to the sampling procedure itself, and it explains the name of the effect; in the mining field, if a sample contains a nugget, i.e. a small piece of mineral in its natural state (gold is the typical case), the ore content will be very high, whereas a very close sample, taken just outside the vein of high mineral content, will show a very low ore content (if any at all). This may explain the erratic behaviour of the semi-variogram at the origin, in some cases. We do not expect this kind of behaviour in the semi-variogram of water supply distribution systems, and we shall assume that a continuous variogram, passing through the origin is the best representation of the phenomenon under study.



(a) The nugget effect



(b) The true semi-variogram

Fig. B.2. Nugget effect and corresponding true semi-variogram

Measurement errors can produce a small nugget effect, and we have to be aware of this in the event that the data show some discontinuity near the origin.

Once the experimental semi-variogram is available from the data, the next step is to fit an analytical expression to it, in order to set up the theoretical basis for the Kriging equations. However, not any analytical function can be used to represent the estimated semi-variogram, due to the main features of the semi-variogram which have been reviewed in the previous paragraphs. In

addition, for reasons which will become apparent later on, only positive definite functions can be accepted as suitable models. In practice, only a few analytical models are commonly used; Table B.2 summarises these common models.

All the previous basic models, except the monomial one, are models with a sill, although only the spherical and cubic models reach the sill in a finite distance, given by the parameter B_o; the exponential and Gaussian models only reach the sill asymptotically. Another difference between all the basic models is concerned with their behaviour near the origin, where different concavities are obtained in different models.

Table B.2. Basic analytical models for the experimental semi-variogram.

Model	Analytical expression for $\Gamma(d)$: $\Gamma(d) = A_o + B_o.[\text{function}(d, C_o)]$	Conditions
Monomial in d^{Co}	$B_o\ d^{Co}$	$C_o<2$ $A_o=0$
Spherical	$B_o[1.5(d/C_o) - 0.5(d/C_o)^3]$	$d<C_o$; $A_o=0$
	B_o	$d>C_o$; $A_o=0$
Exponential	$B_o[\ 1 - e^{(-d/Co)}]$	$A_o=0$
Gaussian	$B_o\{1-e[-(d/Co)^2]\}$	$A_o=0$
Cubic	$B_o[7(d/C_o)-8.75(d/C_o)^3+3.5(d/C_o)^5 - 0.75(d/C_o)^7]$	$d<C_o$ $A_o=0$
	B_o	$d>C_o$; $A_o=0$

Having determined the experimental variogram on the one hand, and having a set of available models on the other hand, the aim is to find out which model (or a combination of them) fits the experimental semi-variogram best. This process can be greatly simplified by a computer program which can be run interactively by the user to produce the best fit model in an interactive way; eventually, if the computer program is designed to compute an index of the goodness of fit (i.e. squared sum of residuals or any other index), the process can be automated to give the best model, leaving to the user the decision to accept it, reject it or modify it. Huijbregts (1975) warns about the automatic fitting of variogram models without performing the physical (structural) interpretation of them, the main shortcoming of this being the loss of insight into the behaviour of the regionalized variable. An important feature of these programs is the ability to display graphically (on the screen or in a printout) both the experimental variogram and data and the fitted analytical model, since such a plot can be very helpful when accepting or rejecting the model.

For a more detailed exposition on statistical inference, in the geostatistical context, see Journel and Huijbregts (1978), Chapter III: Structural Analysis, where some FORTRAN program listings for this purpose are included. Also, the paper by Armstrong (1984) is relevant to this subject.

Perhaps the most important feature in the inference of the semi-variogram, is that everything we have said so far is valid only when the intrinsic hypothesis holds [which is expressed

through equations (2) and (3)]. If a drift or trend is detected (i.e.: $E[Z(\underline{x}+\underline{d})-Z(\underline{x})]=m(\underline{d})\neq 0$), then the semi-variogram cannot be estimated from the data and a more complex method is required, such as that known as "intrinsic random functions" [see Delfiner and Delhomme (1973) and de Marsily (1986)]. We shall review this case later on.

B.3. Estimation via Kriging.

In order to derive the Kriging estimation equations, we shall start from some ideal situation, assuming stationarity in the random function; then we shall see how a non stationary process can be dealt with and also how measurement errors can be handled within Kriging. Finally, the most general version of Kriging, known as "universal Kriging" will be outlined. In this case the mean is not constant and the drift is explicitly handled using the concept of an order-k intrinsic random function.

B.3.1. Kriging under stationary conditions.

A stochastic process is said to be second-order stationary or weakly stationary if its mean is a constant value and its covariance depends only on its lag; in our case, this means that

$$E\ [\ Z(\underline{x},\ H)\ ]\ =\ m \qquad (8)$$

and

$$Cov(\underline{x}_1,\underline{x}_2)\ =\ E\ [\ \{Z(\underline{x}_1,H)-m\}\ \{Z(\underline{x}_2,H)-m\}\ ]\ =\ C(\underline{d}) \qquad (9)$$

where $\underline{d} = \underline{x}_1 - \underline{x}_2$: distance (vector) between points $\underline{x}_1$ and $\underline{x}_2$; we are primarily concerned with the magnitude of this vector rather than its direction.

In these conditions, we can expand the expectation in (9) to give:

$$C(\underline{d}) = E[Z(\underline{x}_1,H)Z(\underline{x}_2,H)] - mE[Z(\underline{x}_1,H) - mE[Z(\underline{x}_2,H)] + m^2$$

then, on introducing the constant mean condition, we obtain:

$$C(\underline{d}) = E[Z(\underline{x}_1,H)Z(\underline{x}_2,H)] - m^2 - m^2 + m^2$$

which reduces to

$$C(\underline{d}) = E[Z(\underline{x}_1,H)Z(\underline{x}_2,H)] - m^2 \ . \qquad (10)$$

The covariance, under stationary conditions, is related to the semi-variogram [see de Marsily, (1986)] through the relationship:

$$\Gamma(\underline{d}) = C(\underline{0}) - C(\underline{d}) \qquad (11)$$

and the relationship can be seen graphically in Fig. B.3, where the semi-variogram ordinates are simply the differences between the horizontal line through C(0) and the covariance curve (shaded area).

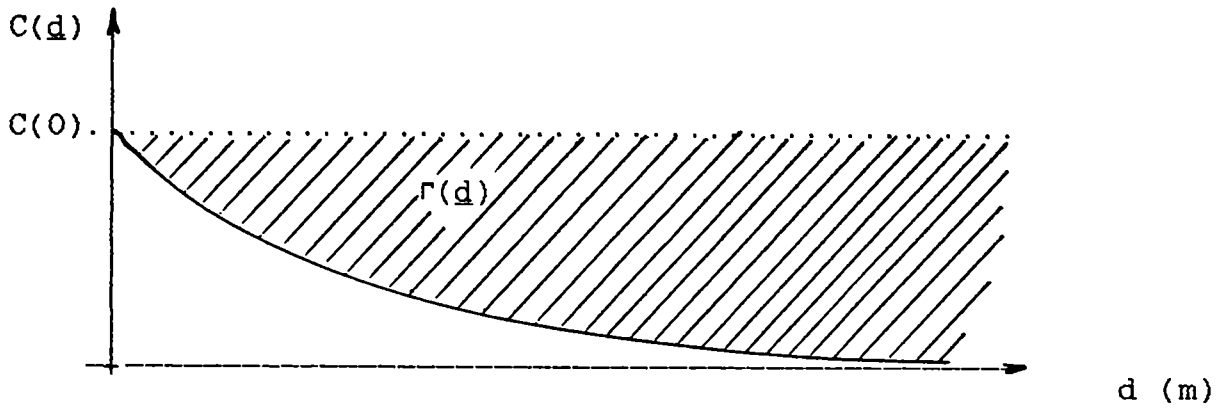


Fig. B.3. Relationship between covariance and semi-variogram.

Thus, because we already know how to estimate the semi-variogram from the measurement data, we can assume that the covariance is actually known. Additionally, we can also assume that the mean "m" is also known and, under these conditions, we can introduce a change of variable, to obtain the following zero mean process (dropping the state H in the notation from now on):

$$Y(\underline{x}) = Z(\underline{x}) - m \qquad (12)$$

i.e.: we are dealing with a zero mean process: $E[Y(\underline{x})]=0$.

Now we can proceed to compute an estimate (Y_o^*) of the unknown variable Y_o at a point $\underline{x}_o$, based on the available measurements at the other points Y_i :

$$Y_i = Y(\underline{x}_i) \quad , \quad i = 1 \, , \ldots , \, n \qquad (13)$$

The Kriging estimate Y_o^* , corresponding to an unmeasured point $\underline{x}_o$, is then computed as a linear weighted combination of the measurements:

$$Y_o^* = Y^*(\underline{x}_o) = \sum_{i=1}^{n} \lambda_o^i \, Y_i \qquad (14)$$

where the λ_o^i are the unknown weights of the Kriging estimator, the superscript "i" refers to the measurement Y_i, whereas the sub-index "o" refers to the point being estimated $\underline{x}_o$. We are dealing basically with a point estimator, where equation (14) is applied one point at a time, using the same set of measurements; we then get a different set of weights for each point to be estimated.

Y_o^* is then an estimate of the true value of the variable Y_o, which is not known. Kriging estimates an optimum Y_o^* in the sense that it minimizes the square of the errors: $Y_o^* - Y_o$; because Y_o is not known, we have to formalise the optimality condition stating that the expected value of the square of the errors is the quantity to be minimized:

$$\min E\left[(Y_o^* - Y_o)^2 \right] \tag{15}$$

In other words, we are minimizing the variance of the errors of the estimation; the expectation is taken at a fixed point, one point x_o at a time, over all the possible realisations of the variable $Y(x_o, H)$.

On developing expression (15) and introducing the Kriging estimator from equation (14), we get:

$$E\left[(Y_o^* - Y_o)^2\right] = E\left[\left(\sum_i \lambda_o^i Y_i - Y_o \right)^2 \right]$$

$$E[(Y_o^* - Y_o)^2] = E[(\sum_i \lambda_o^i Y_i)(\sum_j \lambda_o^j Y_j)]-2 E[(\sum_i \lambda_o^i Y_i Y_o)] +$$

$$+ E[Y_o^2]$$

$$E\left[(Y_o^* - Y_o)^2\right] = \sum_i \sum_j \lambda_o^i \lambda_o^j E[Y_i Y_j] - 2 \sum_i \lambda_o^i E[Y_i Y_o] +$$

$$+ E[Y_o^2] \tag{16}$$

On reordering equation (10), we can compute the expected value of the product $Y_i Y_j$, assuming that the covariance and mean are known:

$$E[Y_i Y_j] = C(x_i - x_j) + m^2 \tag{17}$$

but, because of the variable change (12), Y is a process with zero mean, hence:

$$E[Y_i\ Y_j] = C(\underline{x}_i - \underline{x}_j) \qquad (18)$$

and, in particular, for $\underline{x}_i = \underline{x}_j$:

$$E[Y_o^2] = C(0) \qquad (19)$$

which is the dispersion variance of Y or var(Y).

Introducing the results of equations (18) and (19) into equation (16), we obtain:

$$E\ [(Y_o^* - Y_o)^2] = \sum_i \sum_j \lambda_o^i\ \lambda_o^j\ C[\underline{x}_i - \underline{x}_j] - 2 \sum_i \lambda_o i\ C[\underline{x}_i - \underline{x}_o] + C(0) \qquad (20)$$

which is a quadratic function of the weights λ_o^i. According to our optimality criterion [equation 15], this function ought to be minimized, the variables being the weights λ_o^i. The necessary conditions for a minimum of (20) are given by the following set of equations in the unknowns λ_o^i :

$$\frac{\partial}{\partial\ \lambda_o^i}\ E\ [\ (Y_o^* - Y_o)^2\ ] = 0 \qquad i=1,\ \ldots,\ n \qquad (21)$$

i.e.

$$\left.\begin{aligned} \frac{\partial}{\partial\ \lambda_o^i}\ E[(Y_o^* - Y_o)^2] = 2 \sum_j \lambda_o^j\ C[\underline{x}_i - \underline{x}_j] - 2\ C[\underline{x}_i - \underline{x}_o] = 0 \\ i = 1,\ \ldots,\ n \end{aligned}\right\} \qquad (22)$$

hence:

$$\boxed{\sum_j \lambda_o^j\ C[\underline{x}_i - \underline{x}_j] = C[\underline{x}_i - \underline{x}_o]} \qquad i = 1,\ \ldots,\ n \qquad (23)$$

A sufficient condition for a unique solution of (23) is that the matrix of covariances (C) has to be positive definite, and that is the reason why we wanted positive definite functions for the semi-variogram model.

Some important features of the Kriging estimator become apparent from equation (23), namely:

i) The weights $\lambda_o{}^j$ do not depend explicitly on the value of the measurements Z_i. The weights determined with (23) depend on the structure of the regionalized variable, expressed through the semi-variogram or covariance matrix C. This means that the same set of weights can be used with different sets of measurements for the same location, provided that the information brought up by the new measurements does not alter the structure of the problem (semi-variogram). This rather striking property of Kriging can be used advantageously in real-time estimation, the semi-variogram being updated off-line, if necessary, or a bank of different time-dependent semi-variograms can be made available based on past measurements.

ii) The estimation considers both the distances between the point being estimated and the measurement points $[\underline{x}_i-\underline{x}_o]$ and the distances between the measurement points $[\underline{x}_i-\underline{x}_j]$. The former means that Kriging gives a higher weight to data closer to the point being estimated and the latter means that Kriging discriminates redundant information [see Delfiner and Delhomme (1973; Fig. 1) for an example of an estimator which does not discriminate redundant information].

iii) If we set $\underline{x}_o = \underline{x}_j$, i.e.: $\underline{x}_o$ coincides with one measurement point $\underline{x}_j$, the solution of (23) gives $\lambda_o{}^j=1$ and $\lambda_o{}^k=0 \; \forall \; k\neq j$, which produces $Y_o{}^*=Y(\underline{x}_j)$, i.e.: the estimate coincides with the measurement. This is why Kriging is said to be an <u>exact interpolator</u>, in other words: the estimates "replicate" exactly the measurements.

iv) Armstrong (1984) showed that, even in reduced order cases (4x4 systems), the Kriging linear system can be <u>ill-conditioned</u>. In that case, small changes in the variogram produce such changes in the matrix of coefficients, which may lead to significant changes in the weights ($\lambda_o{}^i$). Armstrong recommended that the computation of the condition number of the Kriging matrix be incorporated as a standard step in the estimation process, so that some warning can be given to the user in the presence of an ill-conditioned system [see Appendix A for details of ill-conditioned linear systems of equations].

v) The right hand side of (23) is the only term dependent on the point to be estimated ($\underline{x}_o$); this means that for estimating $Y_o{}^*$ for different locations, we only need to solve equation (23) for different right hand side vectors. Thus, efficient solutions of the linear system (23) can be obtained via Gaussian elimination with a triangular decomposition, which can be carried out only <u>once</u>, stored and used repeatedly for each different point ($\underline{x}_o$) to be estimated.

Appendix A includes a review of the theory and performance of the most widely used linear solvers; all of them are applicable

to solve the linear systems generated by Kriging, due to the symmetry and positive definiteness of the system.

Having computed the weights $\lambda_o{}^i$, we go back to equation (14) to compute the Kriging estimate $Y_o{}^*$ corresponding to the particular location x_o , i.e.:

$$Y_o{}^* = Y^*(x_o) = \sum_{i=1}^{n} \lambda_o{}^i \, Y_i$$

Now, the last problem to be solved is the determination of the estimation variance. Since we do not know Y_o, the true value of the variable at x_o, we cannot compute the error $Y_o{}^*-Y_o$ directly, only its variance:

$$\mathrm{var}(Y_o{}^*-Y_o) = E[(Y_o{}^*-Y_o)^2] - \{ E[Y_o{}^*-Y_o] \}^2 \qquad (24)$$

Introducing equation (14):

$$E[Y_o{}^* - Y_o] = E[\sum_i \lambda_o{}^i \, Y_i] - E[Y_o]$$

or

$$E[Y_o{}^* - Y_o] = \sum_i \lambda_o{}^i \, E[Y_i] - E[Y_o]$$

but, because of the change of variable (equation 12), in general $E[Y_j]=0 \;\; \forall \; j$, then:

$$E[Y_o{}^* - Y_o] = 0 \qquad (25)$$

As a result, equation (24) simplifies to:

$$\mathrm{var}(Y_o{}^*-Y_o) = E[(Y_o{}^*-Y_o)^2] \qquad (26)$$

Note that the right hand side of (26) is simply the quadratic function we had to minimize [equation (20)]. But we already know

the right hand side in equation (20) after determining the weights λ_o^i by solving the linear system (23); therefore:

$$\mathrm{var}(Y_o^* - Y_o) = \sum_i \sum_j \lambda_o^i \lambda_o^j C(x_i - x_j) - 2 \sum_i \lambda_o^i C(x_i - x_o) + C(0)$$

Introducing (23) into the above equation gives:

$$\mathrm{var}(Y_o^* - Y_o) = \sum_i \lambda_o^i C(x_i - x_o) - 2 \sum_i \lambda_o^i C(x_i - x_o) + C(0)$$

which, on simplifying, yields:

$$\mathrm{var}(Y_o^* - Y_o) = - \sum_i \lambda_o^i C(x_i - x_o) + C(0)$$

and recalling that $C(0) = \mathrm{var}(Y)$:

$$\boxed{\mathrm{var}(Y_o^* - Y_o) = \mathrm{var}(Y) - \sum_i \lambda_o^i C(x_i - x_o)} \qquad (27)$$

In words, (27) means that the estimation variance is less than the dispersion variance of the random function; only when $x_o = x_j$ is the variance of the estimation exactly equal to the measurement variance. The optimum combination of measurements with a given variance produces then, via Kriging, an estimate which, on average, has a minimum mean square error.

Finally, having solved the problem in terms of the *transformed* variable Y, we need to go back to the original variable Z:

$$Z = Y + m$$

Then,

$$\boxed{Z_o^* = Z^*(x_o) = m + \sum_{i=1}^{n} \lambda_o^i (Z_i - m)} \qquad (28)$$

and the Kriging variance (σ_o^2) becomes:

$$\sigma_o^2 = \text{var}(Z_o^* - Z_o) = \text{var}(Z) - \sum_i \lambda_o^i \, C(x_i - x_o) \qquad (29)$$

The second term on the right hand side of equation (29) is known as the variance reduction, i.e.: the variance reduction brought about by the optimal combination of measurements.

It must be stressed that, strictly speaking, the variance given by (27) and (29) is the estimation variance, or the variance of the estimator, rather than the variance of the estimates. In other words, it is a sort of "average" of the variance of the estimates.

B.3.2. Relaxing the second-order stationarity condition: the intrinsic hypothesis.

The second-order stationarity conditions used to derive the previous equations apply to stochastic processes where the mean of the random function is known and where the covariance is finite; then the variogram tends to a sill. The stationarity conditions may not represent the real conditions in some applications where the mean may not be known and the variance may not be finite; to overcome these problems a less stringent set of conditions is introduced, which is referred to as the "intrinsic hypothesis".

In the intrinsic hypothesis the process satisfies the following conditions:

$$E\,[Z(\underline{x}_1,H) - Z(\underline{x}_2,H)] = m$$

where m is an unknown constant.

and

$$\text{var}\,[Z(\underline{x}_1,H) - Z(\underline{x}_2,H)] = 2\,\Gamma(\underline{d})$$

i.e.: the variance is a function of $\underline{d}$, the distance vector given by $\underline{d} = \underline{x}_1 - \underline{x}_2$, and not of $\underline{x}_1$ and $\underline{x}_2$.

Under these conditions we shall define the following Kriging estimator:

$$Z_o^* = Z^*(\underline{x}_o) = \sum_{i=1}^{n} \lambda_o^i \, Z_i \qquad (30)$$

where, as before, λ_o^i is a set of weights which has to be determined.

For the moment we shall assume that the mean is constant, though unknown, i.e.:

$$E[Z_o^*] = E[Z_o] = m \qquad (31)$$

Since we are looking for an unbiased estimator, this means that:

$$E[Z_o^* - Z_o] = 0$$

where:

Z_o is the true value of the variable Z at $\underline{x}_o$, which is unknown.

Introducing the estimator (30) into (31), we get

$$E[\sum_{i=1}^{n} \lambda_o^i \, Z_i \,] = E[Z_o] = m$$

or

$$\sum_{i=1}^{n} \lambda_o^i \, E[\, Z_i \,] - m$$

i.e.:

$$\sum_{i=1}^{n} \lambda_o^i \, m = m$$

which implies that:

$$\sum_{i=1}^{n} \lambda_o^i = 1 \qquad (32)$$

Now, as before in the second-order stationarity case, we impose the condition of minimum variance on the estimator, i.e. we need:

$$\text{minimum } E\,[\,(Z_o^* - Z_o)^2\,]\,. \qquad (33)$$

Introducing the estimator (30) into (33) gives:

$$E\,[(Z_o^* - Z_o)^2] = E\,[\,(\sum_i \lambda_o^i \, Z_i - Z_o)^2] \qquad (34)$$

But now, because of (32), we can have the identity:

$$Z_o = \sum_{i=1}^{n} \lambda_o^i \, Z_o \qquad (35)$$

and introducing (35) into (34):

$$E\,[(Z_o^* - Z_o)^2] = E[(\sum_i \lambda_o^i \, Z_i - \sum_i \lambda_o^i \, Z_o)^2]$$

and factorizing:

$$E\,[(Z_o^* - Z_o)^2] = E[\{\sum_i \lambda_o^i \,(Z_i - Z_o)\}^2]$$

or

$$E\,[(Z_o^* - Z_o)^2] = E[\sum_i \lambda_o^i \,(Z_i - Z_o) \sum_j \lambda_o^j \,(Z_j - Z_o)]$$

which leads to

$$E\,[(Z_o^* - Z_o)^2] = \sum_i \sum_j \lambda_o^i \, \lambda_o^j \, E[(Z_i - Z_o)(Z_j - Z_o)]. \qquad (36)$$

Recalling the definition of the semi-variogram in equation (1):

$$\Gamma(\underline{x}_i,\underline{x}_j) \equiv \tfrac{1}{2}\, E\,[\,\{Z(\underline{x}_i,H) - Z(\underline{x}_j,H)\}^2\,]$$

or simply

$$\Gamma(\underline{x}_i-\underline{x}_j) = \tfrac{1}{2}\, E\,[(Z_i - Z_j)^2] \qquad (37)$$

which can be re-arranged (adding and subtracting Z_o) as

$$\Gamma(\underline{x}_i-\underline{x}_j) = \tfrac{1}{2}\, E\,[\{(Z_i - Z_o) - (Z_j-Z_o)\}^2]$$

On developing the squared binomial:

$$\Gamma(\underline{x}_i-\underline{x}_j) = \tfrac{1}{2}E[(Z_i - Z_o)^2]-E[(Z_i-Z_o)(Z_j-Z_o)]+\tfrac{1}{2}E[(Z_j - Z_o)^2] \qquad (38)$$

and using the semi-variogram definition [equation (37)] again, in the first and third terms on the right side of (38), we get

$$\Gamma(\underline{x}_i-\underline{x}_j) = \Gamma(\underline{x}_i - \underline{x}_o)-E[(Z_i-Z_o)(Z_j-Z_o)]+\Gamma(\underline{x}_j - \underline{x}_o)$$

which allows us to compute the expectation needed in equation (36), i.e.

$$E[(Z_i-Z_o)(Z_j-Z_o)]= - \Gamma(\underline{x}_i-\underline{x}_j) + \Gamma(\underline{x}_i - \underline{x}_o) + \Gamma(\underline{x}_j - \underline{x}_o)$$

which, when introduced into (36) gives:

$$E[(Z_o^*-Z_o)^2] = -\sum_i \sum_j \lambda_o^i\, \lambda_o^j\, \Gamma(\underline{x}_i-\underline{x}_j) + \sum_i \sum_j \lambda_o^i\, \lambda_o^j\, \Gamma(\underline{x}_i-\underline{x}_o)+$$
$$+ \sum_i \sum_j \lambda_o^i\, \lambda_o^j \Gamma(\underline{x}_j - \underline{x}_o) \qquad (39)$$

This can be simplified, because we can factorize by $\Sigma\ \lambda_o^i$ or $\Sigma\lambda_o^j$ in the last two terms of (39), both factors being exactly 1 according to (32); in addition

$$\Sigma\ \lambda_o^i\ \Gamma(\underline{x}_i-\underline{x}_o) = \Sigma\ \lambda_o^j\ \Gamma(\underline{x}_j-\underline{x}_o)$$

whence (39) reduces to

$$E[(Z_o^*-Z_o)^2] = -\sum_i \sum_j \lambda_o^i\, \lambda_o^j\, \Gamma(\underline{x}_i-\underline{x}_j) + 2 \sum_i \lambda_o^i\, \Gamma(\underline{x}_i-\underline{x}_o) \qquad (40)$$

This is the quadratic form corresponding to equation (20) in the second-order stationary case. Equation (40) has to be minimized, subject to the equality constraint represented by equation (32). The minimization can be carried out via the Lagrange multipliers technique, minimizing (without restrictions) the expanded objective function:

$$L(\lambda_o^i, \mu) = \tfrac{1}{2}\, E[(Z_o^* - Z_o)^2] - \mu \left[\sum_i \lambda_o^i - 1\right] \qquad (41)$$

where the factor ½ in the first term and the negative sign in the second have been added for convenience, but do not alter the main objective. The coefficient "μ" is the (unknown) Lagrange multiplier, which has to be determined, together with the set of weights λ_o^i; to do so we have to make the partial derivatives of (41), with respect to μ and λ_o^i ($i=1,\ldots,n$), *equal to zero*, which leads to a set of (n+1) linear equations in (n+1) unknowns:

$$\boxed{\begin{aligned} &\sum_j \lambda_o^j\, \gamma(x_i - x_j) + \mu = \gamma(x_i - x_o) \qquad i=1,\ldots,n \\ &\sum_i \lambda_o^i = 1 \end{aligned}} \qquad (42)$$

On introducing the notation:

$$\gamma_{ij} = \gamma(x_i - x_j)$$

the linear system represented by (42) can be expanded as follows:

$$\begin{bmatrix} 0 & \gamma_{12} & \gamma_{13} & \cdots & \gamma_{1n} & 1 \\ \gamma_{12} & 0 & \gamma_{23} & \cdots & \gamma_{1n} & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \gamma_{n1} & \gamma_{n2} & \gamma_{n3} & \cdots & 0 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 0 \end{bmatrix} * \begin{bmatrix} \lambda_o^1 \\ \lambda_o^2 \\ \vdots \\ \lambda_o^n \\ \mu \end{bmatrix} = \begin{bmatrix} \gamma_{10} \\ \gamma_{20} \\ \vdots \\ \gamma_{n0} \\ 1 \end{bmatrix} \qquad (43)$$

.The solution of (43) exists and is unique provided that the matrix of coefficients is positive definite. The matrix is symmetric and it only needs to be solved once, since it is just the right hand side vector that depends on the point ($\underline{x}_o$) being estimated.

We now proceed to compute the variance of the error of the estimator, which is:

$$var(Z_o^*-Z_o) = E[(Z_o^*-Z_o)^2] - \{ E[Z_o^*-Z_o] \}^2 \qquad (44)$$

which, on introducing the condition of unbiased estimator [equation (29)], reduces to :

$$var(Z_o^*-Z_o) = E[(Z_o^*-Z_o)^2]$$

which has already been computed in (40). In addition, we can use the results of the linear system (42) to get:

$$var(Z_o^*-Z_o) = - \sum_i (\lambda_o^i \, \Gamma(\underline{x}_i-\underline{x}_o)-\mu) + 2\sum_i \lambda_o^i \, \Gamma(\underline{x}_i-\underline{x}_o)$$

or

$$\boxed{\sigma_o^2 = var(Z_o^*-Z_o) = \sum_i \lambda_o^i \, \Gamma(\underline{x}_i-\underline{x}_o) + \mu} \qquad (45)$$

B.3.3. Introducing uncertainty in the measurement data.

In the Kriging estimation procedures already reviewed, we have implicitly assumed that the data were actually error-free. This is not the case in practical applications, where, as in water distribution networks, the data come from measurements, which do have an associated error, say ϵ_i, corresponding to each

measurement Z_i.

To handle the measurement errors within Kriging, the following assumptions are made:

- The errors are random with zero mean: $E[\epsilon_i]=0 \qquad \forall i=1,2,..,n$
- The errors are uncorrelated to each other:

$$Cov(\epsilon_i,\epsilon_j) = 0 \qquad \forall\ i\neq j$$

- The errors are uncorrelated with the measurements:

$$Cov(\epsilon_i,Z(\underline{x})) = 0 \qquad \forall i \qquad \forall \underline{x}$$

- The error variance is known: σ_i^2 and $\sigma_i\neq\sigma_j \qquad \forall i\neq j$

Of course, in the event that some measurements are error-free we only need $\sigma_k=0$, where k is the index of that particular error-free measurement.

Under these assumptions, the impact of measurement errors on the Kriging equations is noticeable only in equation (43), the linear system of equations, where the first "n" zero diagonals are replaced by $-\sigma_i^2$; the restriction for the λ_o^i adding up to 1 and the computation of the Kriging variance remains unchanged. In summary, we get:

To determine the weights λ_o^i and the Lagrange multiplier μ:

$$\begin{aligned} &\sum_j \lambda_o^j\, \Gamma(\underline{x}_i-\underline{x}_j) - \lambda_o^i\, \sigma_i^2 + \mu = \Gamma(\underline{x}_i-\underline{x}_o)_{i=1,\ldots,n} \\ &\sum_i \lambda_o^i = 1 \end{aligned} \qquad (46)$$

or, using the notation

$$\Gamma_{ij} = \Gamma(x_i - x_j)$$

$$\begin{bmatrix} -\sigma_1^2 & \Gamma_{12} & \Gamma_{13} & \cdots & \Gamma_{1n} & 1 \\ \Gamma_{12} & -\sigma_2^2 & \Gamma_{23} & \cdots & \Gamma_{1n} & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \Gamma_{n1} & \Gamma_{n2} & \Gamma_{n3} & \cdots & -\sigma_n^2 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 0 \end{bmatrix} * \begin{bmatrix} \lambda_{o2}^1 \\ \lambda_o^2 \\ \vdots \\ \lambda_o^n \\ \mu \end{bmatrix} = \begin{bmatrix} \Gamma_{10} \\ \Gamma_{20} \\ \vdots \\ \Gamma_{n0} \\ 1 \end{bmatrix} \qquad (47)$$

To determine the Kriging variance:

$$\sigma_o^2 = \mathrm{var}(Z_o^* - Z_o) = \sum_i \lambda_o^i \, \Gamma(x_i - x_o) + \mu \qquad (48)$$

i.e. exactly the same as equation (45).

B.3.4. Universal Kriging and k-th order random functions (k-IRF).

In many practical applications the mean cannot be assumed constant, i.e.: we have $E[Z(\underline{x})] = m(\underline{x})$, a function of $\underline{x}$. Because of this, the real semi-variogram cannot be obtained from the raw measurement data. This is evidenced in:

$$\Gamma_{raw}(x_1 - x_2) = \Gamma_{real}(x_1 - x_2) + \tfrac{1}{2} \{ E[Z(x_1) - Z(x_2)] \}^2 \qquad (49)$$

Thus, the real semi-variogram coincides with that extracted from the raw data only when $E[Z(x_1)] = E[Z(x_2)] = m = \text{constant}$. If a drift exists, i.e.: $m = m(\underline{d})$, then equation (49) says that the semi-variogram estimated from the raw data will be a biased estimate of the real semi-variogram.

Nevertheless, some authors [Delhomme, (1979)] accept the possibility that, when the drift or trend of the mean is "small", the intrinsic hypothesis can still give relatively accurate results for short distances. We shall concentrate on the most general case, where that approximation is not acceptable, i.e. in the case where a strong trend is present; in that case, a generalisation of the intrinsic hypothesis will allow us to tackle the problem within the framework of Kriging.

In fact, in the intrinsic hypothesis, what we did was to assume that if the original process $Z(\underline{x})$ was not stationary, then the differences (or increments) were indeed stationary. The generalisation we are looking at does not stop there; if this difference is still not stationary, then why not take the difference again, and again and again, until a stationary process is found. The successive differences are then removing the non-stationarity. The idea is not new; in fact it is widely used in stochastic processes, particularly in detrending time series data.

Then, from now on, we are going to assume that $Z(\underline{x})$ has been differenced as many times as needed in order to produce a stationary process. Thus, $Z(\underline{x})$ represents the differenced stationary process and not the original one.

We shall follow the same sequence as before, firstly we address the problem of how to infer the semi-variogram from the raw data and, secondly, how to carry out the estimation itself via universal (or "generalised") Kriging.

Following Delhomme (1978) and de Marsily (1986), both based on previous work by Matheron, and for a random function Z, we shall refer to $\sum_{i=0}^{n} \lambda^i Z_i$ as a <u>generalised increment of order k</u> if the weights λ^i satisfy the condition:

$$\sum_{i=0}^{n} \lambda^i f^l(\underline{x}_i) = 0 \qquad (50)$$

for all the monomial functions $f^l(\underline{x}_i)$ of degree less or equal than "k"; e.g. in a two-dimensional geometrical space, where $\underline{x}=(x_1,x_2)^T$, this implies that the maximum degree must be $x_1{}^p * x_2{}^q$ with $0 \leq p+q \leq k$.

In other words, in a first-order generalised increment we are asking for the simultaneous fulfilment of:

$$\left.\begin{array}{l} \sum_{i=0}^{n} \lambda^i = 0 \\ \sum_{i=0}^{n} \lambda^i x_1{}^i = 0 \\ \sum_{i=0}^{n} \lambda^i x_2{}^i = 0 \end{array}\right\} \qquad (51)$$

where the superscript "i" is just an index, not an exponent.

Similarly, in the case of a second-order generalised increment we shall be asking for the extended conditions:

$$\left.\begin{array}{ll} \sum_{i=0}^{n} \lambda^i = 0 & \sum_{i=0}^{n} \lambda^i (x_1^i)^2 = 0 \\ \sum_{i=0}^{n} \lambda^i x_1^i = 0 & \sum_{i=0}^{n} \lambda^i (x_2^i)^2 = 0 \\ \sum_{i=0}^{n} \lambda^i x_2^i = 0 & \sum_{i=0}^{n} \lambda^i x_1^i x_2^i = 0 \end{array}\right\} \qquad (52)$$

Of course, the intrinsic hypothesis already seen in previous sections implies a generalised increment of order zero, i.e. only

$$\sum_{i=0}^{n} \lambda^i = 0$$

is fulfilled.

The concept of the generalised covariance of order k is now introduced simply as the covariance of a generalised increment of order k. The generalised covariance is denoted by $K(\underline{d})$. For k=0, the generalised covariance of order zero is simply the semi-variogram, with a negative sign.

Thus, according to the theory of the intrinsic random functions [Matheron, (1973)], if

$$\sum_{i=0}^{n} \lambda^i Z_i$$

is a generalised increment of order k, its variance is given by:

$$\mathrm{var}\left(\sum_{i=0}^{n} \lambda^i Z_i\right) = \sum_i \sum_j \lambda^i \lambda^j K(x_i - x_j) \qquad (53)$$

As before, in the semi-variogram case, only some functions can be used to represent a generalised covariance; basically, they

must guarantee that the variances of generalised increments must be always positive.

Matheron (1973) showed that appropriate models for the isotropic generalised covariance of order k can be represented by polynomials of order 2k+1; he also demonstrated that the even powers are redundant and, as a result, they do not need to be used in the polynomials.

Delfiner (1976) presented an analytical expression for these polynomials and also formalised the conditions for positive definiteness. Table B.3 presents some of the most widely used models for generalised covariances.

As can be seen from Table B.3, the generalised covariances are linear functions of the parameters A_i, which can then be computed via regression analysis or similar techniques.

Kitanidis (1983) evaluated different techniques for the estimation of the parameters of the generalised covariance models. He stressed the point that the determination of the parameters A_i of the generalised covariance model is a process involving considerable errors; he recommended a maximum likelihood technique and a minimum variance unbiased quadratic technique as adequate methods for fitting the unknown parameters to the generalised covariance.

Table B.3. Possible polynomial models for generalised covariances , from Delhomme (1978).

Drift	Drift order k	Model for the generalised covariance
Constant	0	$K(d) = A_0\delta + A_1 \lvert d\rvert$
Linear	1	$K(d) = A_0\delta + A_1 \lvert d\rvert + A_3 \lvert d\rvert^3$
Quadratic	2	$K(d) = A_0\delta + A_1 \lvert d\rvert + A_3 \lvert d\rvert^3 + A_5 \lvert d\rvert^5$
Constraints:	$A_0 \geq 0$, $A_1 \leq 0$, $A_5 \leq 0$ and	
	$A_3 \geq -(10/3)\sqrt{A_1 A_5}$ (in R^2)	
	$A_3 \geq -\sqrt{10 A_1 A_5}$ (in R^3)	
	$\delta=1$ if $d=0$, $\delta=0$ otherwise	

Having produced and checked a valid model for the generalised covariance, we can proceed to determine the universal Kriging equations, which is done, as usual, imposing the unbiased condition on the estimator Z_o^* and the minimum variance on the residual $Z_o^*-Z_o$. The difference now is in the fact that the mean is no longer constant, but has a trend or drift:

$$E[Z(\underline{x})] = m(\underline{x}) \qquad (54)$$

Although the value of $m(\underline{x})$ is not known, we assume that it can be represented (at least locally) in terms of a linear combination of basis functions (usually polynomials), e.g.: in the two-dimensional case, where $\underline{x} = (x_1, x_2)^T$:

$$m(\underline{x}) = a_1 + a_2x_1 + a_3x_2 + a_4x_1^2 + a_5x_1x_2 + a_6x_2^2 + \ldots \qquad (55)$$

or, in general :

$$m(\underline{x}) = \sum_{k=1}^{w} a_k \, p^k(\underline{x}) \qquad (56)$$

where :

- the coefficients a_i have to be determined by fitting, applying some knowledge of the physical behaviour of the variable Z.

- the polynomials $p^k(\underline{x})$ are usually monomials of first or second degree (linear or quadratic drift), for example in a two-dimensional space:

  * linear drift:

    $$m(\underline{x}) = a_1 + a_2x_1 + a_3x_2$$

  * quadratic drift:

    $$m(\underline{x}) = a_1 + a_2x_1 + a_3x_2 + a_4x_1^2 + a_5x_1x_2 + a_6x_2^2$$

  * constant mean (no drift):

    $$m(\underline{x}) = a_1$$

As before, the Kriging estimator is :

$$Z_o^* = \sum_{i=1}^{n} \lambda_o^i \, Z_i \qquad (57)$$

The unbiased estimator must satisfy:

$$E[Z_o^*] = E[Z_o] \qquad (58)$$

where Z_o is the true (unknown) parameter being estimated.

Introducing (54) and (57) into (58) gives:

$$E[\ \Sigma\ \lambda_o^i \, Z_i\ ] = E[Z_o] = m(\underline{x}_o) \qquad (59)$$

and introducing (56) into (59):

$$\sum_{i=1}^{n} \lambda_o^i \left\{ \sum_{k=1}^{w} a_k p^k(x_i) \right\} = \sum_{k=1}^{w} a_k \, p^k(x_o)$$

or, reordering the first term:

$$\sum_{k=1}^{w} a_k \left\{ \sum_{i=1}^{n} \lambda_o^i \, p^k(x_i) \right\} = \sum_{k=1}^{w} a_k \, p^k(x_o)$$

which is satisfied when:

$$\sum_i \lambda_o^i \, p^k(x_i) = p^k(x_o) \qquad k=1,..,m \tag{60}$$

This is a generalisation of the previous condition $\Sigma \ \lambda_o{}^i=1$ which is obtained from (60) when no drift is assumed, i.e.: $m(x)=a_1$.

The rest of the Kriging equations are obtained from the minimum variance condition. For this purpose we could use the generalised covariance concept, provided that $\Sigma \ \lambda^i Z_i$ is a generalised increment of order k, which is true since:

$$\sum_{i=1}^{n} \lambda_o^i \, Z_i = Z_o$$

can be re-written as:

$$\sum_{i=0}^{n} \lambda_o^i \, Z_i = 0$$

with $\lambda_o{}^o = -1$

and provided that the mean (drift) is expressed as a function of monomials of order "k" or lower.

Then, we can minimize the generalised covariance (equation 53) subject to the condition (60) by making zero all the partial derivatives with respect to $\lambda_o{}^i$ of the Lagrangian function:

$$L(\lambda_o^i,\ \mu_k) = \sum_i \sum_j \lambda_o^i \lambda_o^j K(x_i - x_j) + \mu_k \left[\sum_i \lambda_o^i\ p^k(x_i) - p^k(x_o)\right] \qquad (61)$$

which leads to the following set of linear equations:

and

$$\boxed{\begin{aligned} &\sum_j \lambda_o^j K(x_i - x_j) + \sum_k \mu_k\ p^k(x_i) = K(x_i - x_o)] && i=1,\ldots,n \\ & && j=1,\ldots,n \\ &\sum_i \lambda_o^i\ p^k(x_i) = p^k(x_o)] && k=1,\ldots,m \end{aligned}} \qquad (62)$$

This is a system of (n+m) equations in (n+m) unknowns: λ_o^i i=1,...,n and μ_k k=1,...,m, which can be expanded as:

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} & \cdots & K_{1n} & 1 & p_{12} & \cdots & p_{1k} \\ K_{21} & K_{22} & K_{23} & \cdots & K_{1n} & 1 & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \cdots & \vdots \\ K_{n1} & K_{n2} & K_{n3} & \cdots & K_{nn} & 1 & p_{n2} & \cdots & p_{nk} \\ 1 & 1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2n} & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \cdots & \vdots \\ p_{m1} & p_{m2} & p_{m3} & \cdots & p_{mn} & 0 & 0 & \cdots & 0 \end{bmatrix} * \begin{bmatrix} \lambda_o^1 \\ \lambda_o^2 \\ \vdots \\ \lambda_o^n \\ -\mu_1 \\ -\mu_2 \\ \vdots \\ -\mu_m \end{bmatrix} = \begin{bmatrix} K_{10} \\ K_{20} \\ \vdots \\ K_{n0} \\ 1 \\ p_{20} \\ \vdots \\ p_{m0} \end{bmatrix} \qquad (63)$$

using the notation

$$K_{ij} = K(x_i - x_j)$$

and

$$p_{ij} = p^i(x_j)$$

Note that the system (63) is symmetric. Its size is (n+m)x(n+m) which is not particularly large in water resources applications, since is mainly dominated by "n", the amount of data points (piezometric head measurements in our case). Appendix A includes a review of efficient linear solvers for symmetric positive definite linear systems and, because of that, we do not go into more details here.

In the particular case of the intrinsic random functions of first order, the linear system (62) reduces to:

$$\left.\begin{array}{l} \sum_j \lambda_o^j K(\underline{x}_i-\underline{x}_j)-\mu_1-\mu_2 x_1^i-\mu_3 x_2^i = K(\underline{x}_i-\underline{x}_o)] \quad i=1,\ldots,n \\ \text{and} \quad \underline{x}_i=(x_1^i,x_2^i)^T \\ \sum_{i=1}^n \lambda_o^i = 1 \qquad \sum_{i=1}^n \lambda_o^i x_1^i = x_1^o \qquad \sum_{i=1}^n \lambda_o^i x_2^i = x_2^o \end{array}\right\} \quad (64)$$

and in the case of intrinsic random functions of second order, the linear system becomes:

$$\left.\begin{array}{l} \sum_j \lambda_o^j K(\underline{x}_i-\underline{x}_j) + \sum_{k=1}^{6} \mu_k p^k(\underline{x}_i) = K(\underline{x}_i-\underline{x}_o)] \quad i=1,\ldots,n \\ \text{and} \\ \sum_i \lambda_o^i p^k(\underline{x}_i) = p^k(\underline{x}_o)] \quad k=1,\ldots,m \end{array}\right\} \quad (65)$$

with $p^1(x)$, ..., $p^6(x)$ being 6 polynomials in x_1, x_2 as defined in equation (52).

To compute the corresponding Kriging variance we need to go back to the expression for the generalised covariance (equation 53) and to introduce $\Sigma\lambda^j K(\underline{x}_i-\underline{x}_j)$ from the linear system (62), obtaining:

$$\boxed{\text{var}(Z_o^*-Z_o) = K(0) + \sum_k \mu_k p^k(\underline{x}_o) - \sum_i \lambda_o^i K(\underline{x}_i-\underline{x}_o)} \quad (66)$$

which, for the particular case of an intrinsic random function of first order reduces to:

$$\text{var}(Z_o^*-Z_o) = K(0) + \mu_1 + \mu_2 x_1^o + \mu_3 x_2^o - \sum_i \lambda_o^i K(\underline{x}_i-\underline{x}_o) \quad (67)$$

Equations (62) and (66) are the equations for the universal Kriging and the Kriging variance respectively, in the most general case of intrinsic random function of order k. It is not too difficult to re-derive all the previous equations obtained by assuming stationarity or the intrinsic hypothesis, by simply considering them as particular cases of the most general intrinsic random function of order k.

APPENDIX C

DERIVATION OF THE BI-CUBIC SPLINES APPROXIMATION EQUATIONS

C.1. Introducing cubic splines.

In words, a cubic spline is a set of cubic polynomials joined end to end in a continuous and smooth manner.

Mathematically, and following Hayes and Halliday (1974), a polynomial function s(x) is a cubic spline with ordered interior knots $\lambda_1, \lambda_2, \ldots \lambda_h$ (i.e.: $\lambda_1 < \lambda_2 < \ldots < \lambda_h$) if it satisfies the following conditions:

i) In each interval defined by the knots, the function s(x) is a polynomial of degree less or equal than 3.

ii) The function s(x), its first and second derivatives [s'(x) and s''(x), respectively] are continuous. Higher order derivatives do not need to comply with the continuity condition, in particular, a cubic spline will have third derivatives which are discontinuous at the knots [λ_i , i=1,2, ..., h].

The set of interior knots (another set of knots known as exterior knots will be defined later on) divides the domain over which the independent variable (x) is defined into h+1 intervals. A cubic spline is defined as a piecewise polynomial function (of degree less or equal than 3) on each one of these intervals and, provided that the specified knots are different (i.e. simple knots), the cubic spline will be twice-differentiable over its domain. When two coincident knots are specified, the

differentiability of the spline is reduced in one order, at the particular point where the coincident knots are located. Thus, by an adequate knot selection, we are allowed to specify discontinuities in the derivatives of the splines and, indeed, in the spline polynomial itself.

The one-dimensional splines fitting problem consists of finding the analytic expression for s(x) which best approximates a set of "m" data points $\{ x_r, f(x_r) \}$, r=1,2, ... m , for a given set of knots λ_i , i=1,2, ... h. "Best" here is used in the least-squares sense. Note that we are not asking for the function s(x) to pass through the data points, which is the interpolation problem, but only to approximate them; "fitting" is used as an equivalent term for "approximation".

Cox (1987a) argues that in the case of noisy data, approximation methods should be used instead of interpolation. Besides, reliable computer software for splines interpolation, particularly for two-dimensional splines interpolation with scattered data, is not widely available [Anthony and Cox (1987 b)]. For these reasons, we shall apply a spline approximation approach to the piezometric head estimation problem.

From a mathematical point of view, a spline can be represented in various ways, but the B-splines representation (also known as "basic" or "fundamental" or "normalised" splines) has gained a reputation as being the most adequate and stable representation [see Cox (1972) and Cox (1982b) for a discussion on this subject].

Also, cubic splines, rather than splines of any other order, have became the most widely used splines in engineering applications, mainly due to their continuity properties and to the availability of proven and reliable software. The NAG subroutine library deals with cubic splines, while NPL's Data Approximation Subroutine Library deals with splines of any order.

Cubic B-splines are bell-shaped functions, which are defined to be non-zero only within four adjacent knot intervals (defined by five different knots λ_i), being zero everywhere else. Fig. C.1 shows the shape of a B-spline $M_i(x)$ with knots λ_{i-4}, λ_{i-3}, λ_{i-2}, λ_{i-1} and λ_i. Different basic splines can be defined, leading to different magnitudes for their peak point or area under the curve; this degree of freedom in the definition of a B-spline is used to scale-up the shape of the B-spline.

The following explicit representation of the B-spline, $M_i(x)$, has been shown to have some advantages over earlier divided-difference representations.

$$M_i(x) = \sum_{s=0}^{4} v_{is} \, (x - \lambda_{i-s})_+^3 \qquad (1)$$

with

$$v_{is} = 1 \; / \; \{ \prod_{\substack{r=0 \\ r \neq s}}^{4} (\lambda_{i-s} - \lambda_{i-r}) \} \qquad (2)$$

and where the notation $(x - \lambda_j)_+^3$ stands for:

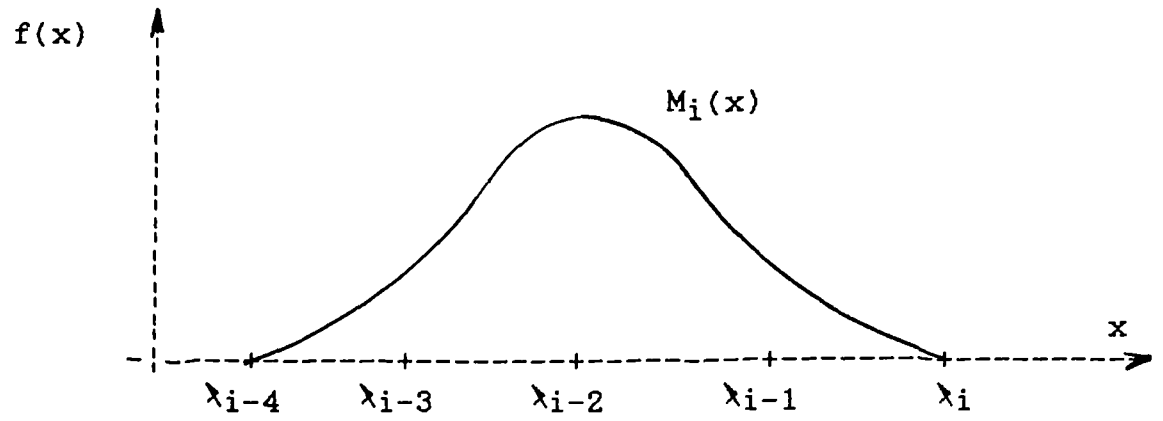


Fig. C.1. Representation of a B-spline.

$$(x - \lambda_j)_+^3 = \begin{cases} (x - \lambda_j)^3 & \text{if } x > \lambda_j \\ 0 & \text{if } x \leq \lambda_j \end{cases} \qquad (3)$$

i.e. the function $(x - \lambda_j)_+^3$ has non-zero non-negative values only from $x = \lambda_j$ onwards.

The weighting factors v_{is} in equation (2), have been defined such that the area under the cubic B-spline is exactly equal to 1/4. Note that the weights do not depend either on the data points or on the the points to be approximated, but only on the knots; they can therefore be pre-computed and kept in storage, before the actual approximation process is carried out.

The cubic B-splines previously defined are the "building blocks" of splines fitting. The idea is now to express the polynomial function s(x), describing the spline as a linear combination of the B-splines, having one B-spline associated with each knot. As a result, a cubic spline combines four B-splines in an interval $(\lambda_i - \lambda_{i-1})$, i.e.: the B-splines $M_i(x)$, $M_{i+1}(x)$, $M_{i+2}(x)$ and $M_{i+3}(x)$, as shown in Fig. C.2.

Since normally we shall be interested in defining a spline for a limited range of the independent variable x, say $x \in [a,b]$, and due to the fact that for a convenient representation of the spline in any interval we want to combine 4 B-splines (as shown in Fig. C.2), and assuming that:

$$a < \lambda_1 < \lambda_2 < \ldots < \lambda_h < b$$

this implies that, for completeness, we need 8 extra knots, known as exterior knots, to be able to produce 4 B-splines in the first and last intervals: $[a,\lambda_1]$ and $[\lambda_h,b]$, respectively. These exterior knots are normally labelled as:

$$\lambda_{i-3} < \lambda_{i-2} < \lambda_{i-1} < \lambda_o \leq a$$

and

$$b \leq \lambda_{h+1} < \lambda_{h+2} < \lambda_{h+3} < \lambda_{h+4}$$

Thus, the extended set of B-splines looks like that shown in Fig. C.3.

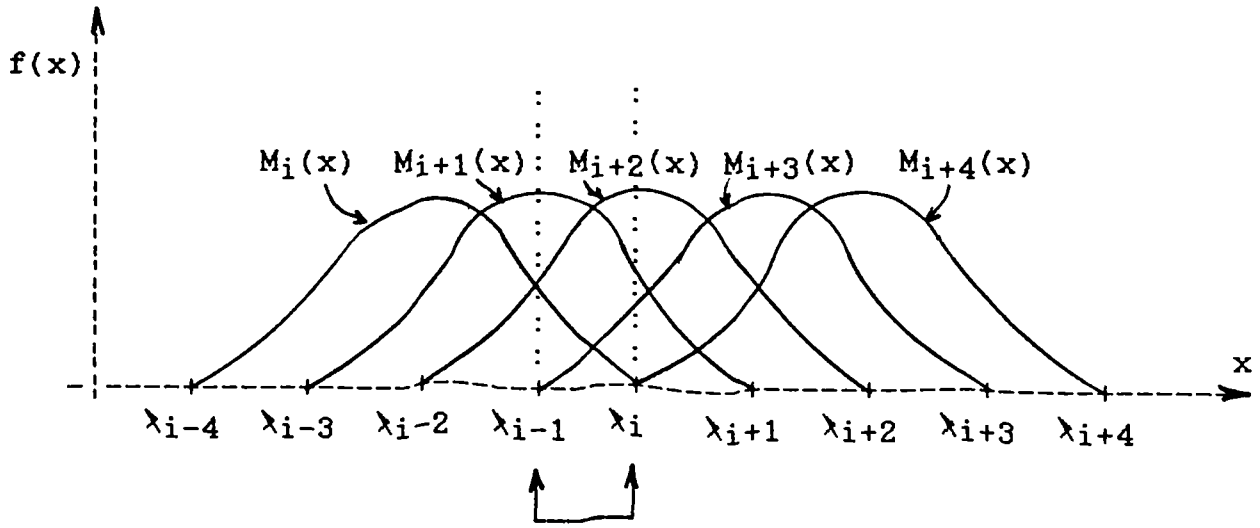


Fig. C.2. B-splines involved in the spline function for the interval $(\lambda_{i-1},\lambda_i)$.

Cox (1972) demonstrated the conveniences of using a recursive relationship for the numerical representation of the B-spline $M_i(x)$, instead of the explicit representation of equation (1). The reasons for such a preference are mainly concerned with stability considerations, particularly when two or more knots are placed very close to of each other, in which case the explicit representation "blows up". This does not change most of what we have previously said, but only changes the way the B-splines are computed.

The recursive expression proposed by Cox (1972), and adopted by Hayes and Halliday (1974) for evaluating B-splines, both in the NPL and NAG libraries is:

$$M_{n,i}(x) = \frac{(x-\lambda_{i-n})\, M_{n-1,i-1}(x) + (\lambda_i - x)\, M_{n-1,i}(x)}{(\lambda_i - \lambda_{i-n})} \qquad (4)$$

where:

$M_{n,i}(x)$: represents a B-spline of degree (n-1), given as a linear combination of B-splines of lower degree. Thus, n=4 for our cubic B-splines and $M_{4,i}(i)$ becomes the equivalent of our previous $M_i(x)$.

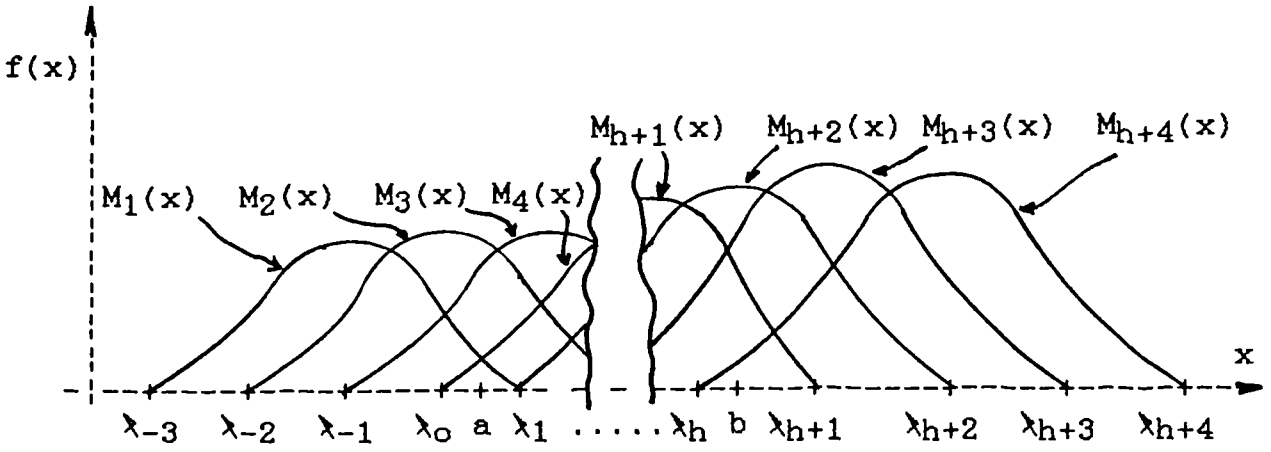


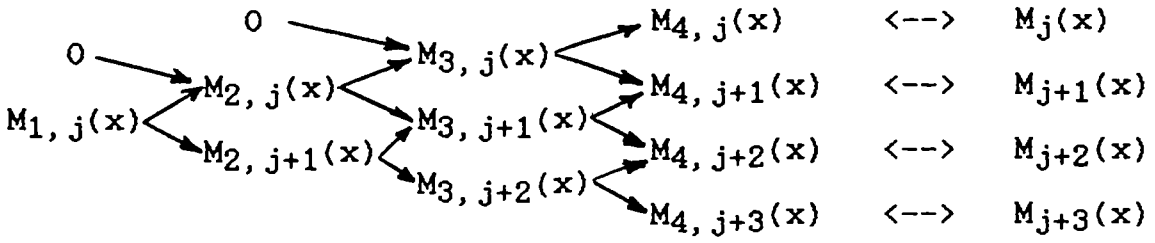
Fig. C.3. Extended set of B-splines: interior and exterior knots.

The evaluation of the B-splines using equation (4) starts with:

$$M_{1,i}(x) = \begin{cases} 1./(\lambda_i - \lambda_{i-1}) & \text{if } x \in [\lambda_{i-1}, \lambda_i) \\ 0.0 & \text{if } x \notin [\lambda_{i-1}, \lambda_i) \end{cases} \qquad (5)$$

The recursive definition for the B-splines given by equations (4) and (5) implies that the area under its curve is exactly 1/n, that is to say, for the cubic B-spline (n=4), the area is exactly the same as before: 1/4.

For a point, either a data point or a point where an approximate value is needed, lying in the interval $(\lambda_{j-1}, \lambda_j)$, the sequence for evaluating a cubic B-spline is given in the following array, where all B-splines not shown are considered as zero, and we are proceeding from left to right:



The recursive scheme for the evaluation of B-splines represented by equation (4) is stable, even in the case of coincident knots ($\lambda_i = \lambda_{i-1}$). Sometimes, it may be desirable to have coincident knots, in order to introduce discontinuities in the second derivatives, first derivatives or in the function itself.

C.3. Data fitting using one-dimensional cubic splines.

It has already been said that the idea is to combine the cubic B-splines linearly, in order to produce a general cubic spline polynomial function s(x), that is to say:

$$s(x) = \sum_{i=1}^{h+4} r_i \, M_i(x) \qquad (6)$$

where:

r_i are the linear weights (to be determined) corresponding to the different B-splines for each interval $(\lambda_{i-1}, \lambda_i)$

The curve fitting problem consists of representing the set of data points (measurements for example): $\{x_r, f(x_r)\}$, $r=1,2,\ldots,m$, in the form of equation (6). This implies that the coefficients r_i have to be determined as the solution of the following linear system of equations:

$$\left.\begin{aligned}
s(x_1) &= \sum_{i=1}^{h+4} r_i \, M_i(x_1) = f(x_1) = f_1 \\
s(x_2) &= \sum_{i=1}^{h+4} r_i \, M_i(x_2) = f(x_2) = f_2 \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad \vdots \\
s(x_m) &= \sum_{i=1}^{h+4} r_i \, M_i(x_m) = f(x_m) = f_m
\end{aligned}\right| \qquad (7)$$

or, in a compact format:

$$\sum_{i=1}^{h+4} r_i \, M_i(x_r) = f(x_r) = f_r \qquad r=1,2,\ldots m \qquad (8)$$

which is clearly a linear system of "m" equations in the h+4 unknowns r_i. Because normally m > h+4, (7) is an overdetermined system of equations and an exact solution does not exist; a solution in the least-squares sense in sought. This system is known as the observation equations.

An even more compact matrix representation of such a system is:

$$A\,\underline{r} = \underline{f} \qquad (9)$$

where:

A is a rectangular mx(h+4) matrix with coefficients:

$$A_{ri} = M_i(x_r).$$

$\underline{r}$ and $\underline{f}$ are column vectors of lengths (h+4)x1 and mx1, respectively.

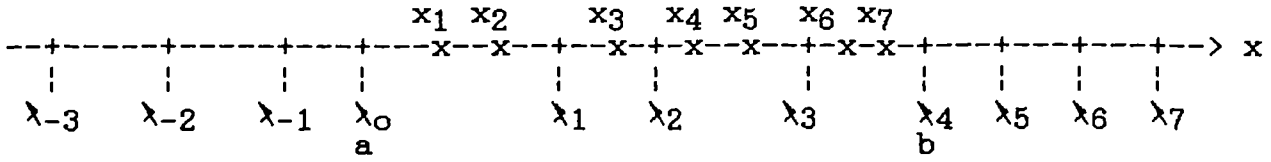
The structure of the matrix A has some important features:

i) Due to the fact that for every data point there are only 4 non-zero B-splines involved, and they are adjacent: $M_{j-3}(x)$, $M_{j-2}(x)$, $M_{j-1}(x)$ and $M_j(x)$ (if the data point lies in the interval $\lambda_{j-1}<x<\lambda_j$), this implies that A has only four non-zero elements per row and they are adjacent. In addition, if the data points are sorted in ascending order, according to their magnitudes, the first of the four non-zeros of each row of A will never lie at the left of the first non-zero of the previous row. Two or more non-zeros will lie in the same column of A, if and only if they lie in the same knot interval, say $(\lambda_{j-1}, \lambda_j)$.

ii) This means that A has a special banded structure and, as a result, this represents some advantages as far as the storage and handling of the system is concerned.

To illustrate this point, some examples of the structure of the observation matrix A are as follows:

Example a:

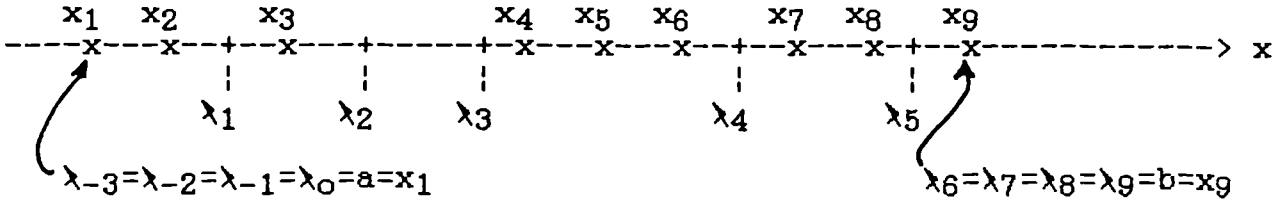


the corresponding structure of the observation matrix is:

$$A = \begin{bmatrix} x & x & x & x & & & \\ x & x & x & x & & & \\ & x & x & x & x & & \\ & & x & x & x & x & \\ & & x & x & x & x & \\ & & & x & x & x & x \\ & & & x & x & x & x \end{bmatrix}$$

where "x" represents a non-zero value.

Example b:



the corresponding structure of the observation matrix is:

$$A = \begin{bmatrix} x & & & & & & & \\ x & x & x & x & & & & \\ & x & x & x & x & & & \\ & & & x & x & x & x & \\ & & & x & x & x & x & \\ & & & x & x & x & x & \\ & & & & x & x & x & x \\ & & & & x & x & x & x \\ & & & & & & & x \end{bmatrix}$$

Since normally m > h+4 , i.e. there are more measurements than unknowns, the linear system (9) is overdetermined. Clearly, a

unique solution is not possible, and the best we can expect is a solution in a statistical sense, such as to minimise some pre-established criterion; this is normally achieved by resolving the overdetermination in the least-squares sense, which leads to the following set of "normal" equations [premultiplying equation (9) by the transpose of the observation matrix A]:

$$A^T A \underline{r} = A^T \underline{f} \qquad (10)$$

where:

$[A^T A]$ is a squared (h+4)x(h+4) banded symmetric matrix, with bandwidth (2n-1).

The system (10) is not a very large linear system; it does not have hundreds or thousands of equations, and its banded structure allows us to use efficient specialised linear solvers.

The numerical solution of (10) can be obtained via a direct method, i.e. a Cholesky factorization [see Appendix A for a review on the solution of linear systems of equations]. However, the system (10) can present instabilities which may require a more stable numerical solution, i.e. orthogonal factorization, Givens rotations or Householder transformations [see Golub and Van Loan (1983) for details on these methods]. The ill-conditioning of (10) becomes manifest particularly when m >>h+4. Iterative solutions (like the conjugate gradient method) of the normal equations have also been reported [Cox (1982a)].

A few words ought to be said with respect to the existence of the solution of the normal equations. The solution indeed exists, if and only if the data fulfil some conditions known as the

Schoenberg and Whitney conditions; for interpolation problems, these conditions establish a relationship between the location of data points and knots, which is normally expressed as:

$$\begin{array}{ccccc} x_1 & < & \lambda_1 & < & x_{1+n} \\ x_2 & < & \lambda_2 & < & x_{2+n} \\ \vdots & & \vdots & & \vdots \\ \vdots & & \vdots & & \vdots \\ x_h & < & \lambda_h & < & x_m \end{array}$$

For one-dimensional approximation problems, the Schoenberg and Whitney conditions are restricted to any subset of the data abscissae; if the conditions hold for any such subset, then the solution of the normal equations exists and is indeed unique.

In essence, the solution of equation (10) leads to the weights $\underline{r}$ corresponding to each inter-knot interval, from λ_1 onwards. The weights $\underline{r}$ represent the way in which the B-splines are linearly combined together to approximate the data set.

Having determined $\underline{r}$ from equation (10), they can be introduced in the expression for the general cubic spline:

$$s(x) = \sum_{i=1}^{h+4} r_i \, M_i(x)$$

to obtain a polynomial function that can be used to approximate any unmeasured point $x \in [a,b]$. For any of these points, say x, we need to evaluate the set of 4 B-splines $M_i(x)$ corresponding to x, using the recursive stable equation (4).

One of the critical aspects of spline fitting is concerned with the placement of the knots λ_i . They not only determine the number of unknowns in the normal equations, but they also

influence the conditioning of the problem and the accuracy of the approximation. In principle, it would seem that adding more knots should lead to an improved approximation, but this is not always true, since too many knots may produce overfitting of the data set, which is manifested through unwanted oscillations of the fitted spline [see Cox, Harris and Jones (1987)]. The automatic placement of knots in splines approximation is a subject of ongoing research.

C.4. Data fitting using bi-cubic B-splines.

So far, the application of cubic splines for approximating data with only one independent variable has been reviewed. In Chapter 6 of this Thesis, the convenience and inconvenience of modelling the piezometric heads of the water distribution system as a continuous surface, "floating" above the network has been discussed; in following this line of thinking, the one-dimensional cubic spline approach is clearly insufficient for describing the piezometric plane, and two independent variables (x,y) are needed.

The one-dimensional cubic B-splines concepts can be extended to the two-dimensional case in a relatively straightforward manner, leading to a bi-cubic spline (or doubly cubic spline). In the two-dimensional case the measurement data and any point in the space, will be represented by the triples: {x, y, f(x,y)}, where (x,y) caters for the horizontal coordinates of each point, and f(x,y) represents the value of the function (piezometric head in

our case) at that particular point (x,y).

As in the one-dimensional case, the basic splines (which have not been defined yet) will be associated with a set of knots. To introduce the knots, let us consider the plane defined by the independent variables (x,y), and let us define a rectangle R on it, such as:

$$R = \{ (x,y) \text{ such that } a \leq x \leq b \text{ and } c \leq y \leq d \}$$

where:

a is the minimum value of the independent variable x.

b is the maximum value of the independent variable x.

c is the minimum value of the independent variable y.

d is the maximum value of the independent variable y.

In this rectangle R, two different set of interior knots λ_i i=1,2, ... h+1 and μ_j j=1,2,... k+1 are introduced, with similar characteristics as in the one-dimensional case although here, for convenience, $b = \lambda_{h+1}$ and $d = \mu_{k+1}$. These two sets of interior knots divide the rectangle R into a set of (h+1)(k+1) panels R_{ij}, as shown in Fig. C.4.

For completeness, similar to the one-dimensional case, let us add exterior knots to the sets λ_i and μ_j such that:

$$\lambda_{-3} < \lambda_{-2} < \lambda_{-1} < \lambda_o = a$$

$$b = \lambda_{h+1} < \lambda_{h+2} < \lambda_{h+3} < \lambda_{h+4}$$

and

$$\mu_{-3} < \mu_{-2} < \mu_{-1} < \mu_o = c$$

$$d = \mu_{k+1} < \mu_{k+2} < \mu_{k+3} < \mu_{k+4}$$

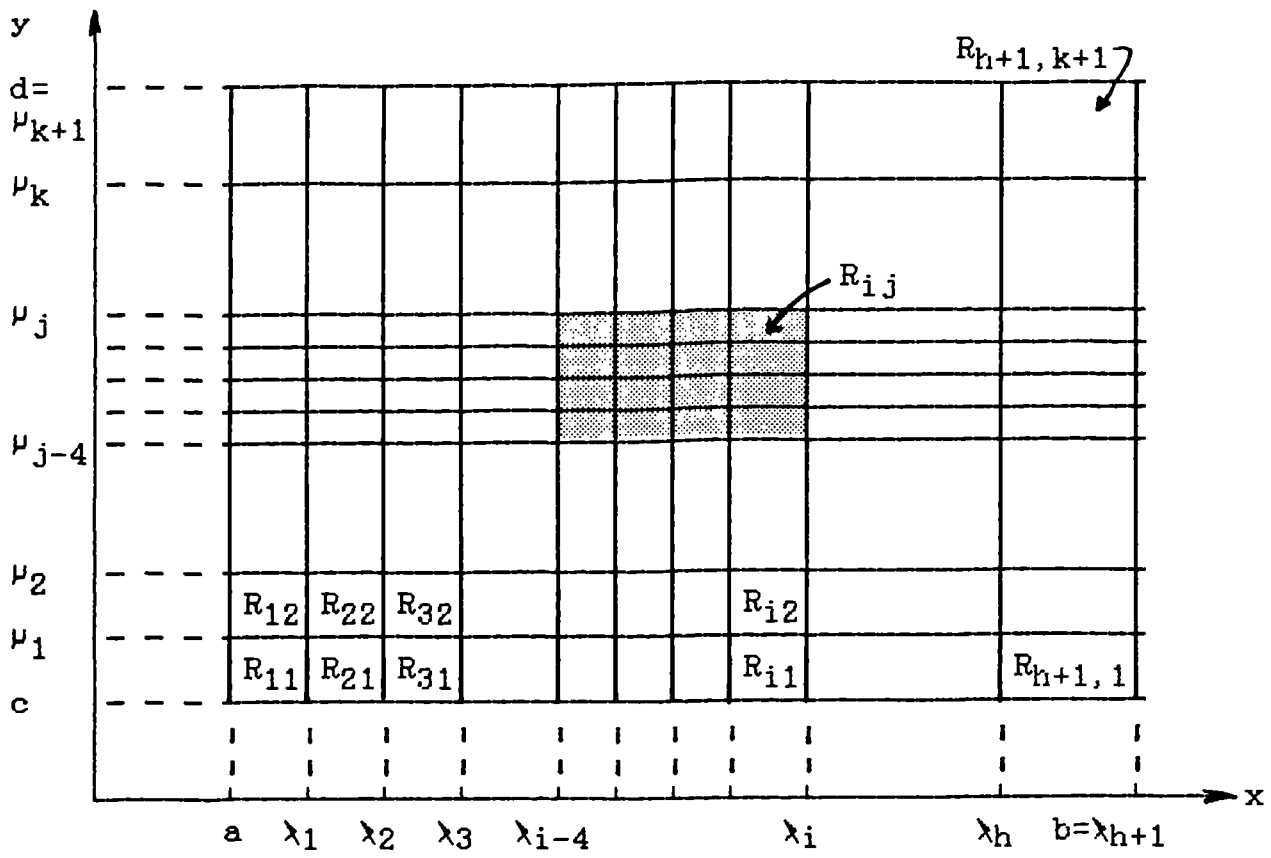


Fig. C.4. Rectangular subspace R for the independent variables x and y, divided into panels R_{ij} by knots λ_i i=1,2, ... h+1 and μ_j j=1,2, ... k+1.

As in the one-dimensional case, where the B-splines were defined as the basis functions (or "building blocks") for representing general cubic splines functions, we need to specify basis functions for the two-dimensional problem. To do so, the concept of B-splines can be extended to the two-dimensional space in the following way.

Let us define two sets of one-dimensional B-splines, one related to the independent variable "x" and to the set of knots λ_i, say $M_i(x)$, and the second related to the independent variable "y" and to the set of knots μ_j, say $N_j(y)$. Both one-dimensional B-splines [i.e. $M_i(x)$ and $N_j(y)$], have similar properties to

those used in the previous section.

Let us think of the sets of B-splines $M_i(x)$ and $N_j(y)$ as two one-dimensional structures; then, we can consider the set of all cross-products (or tensor products) of $M_i(x)$ with $N_j(y)$ as the basis functions for the bi-cubic spline. Although the mathematical concept of a tensor product is rather abstract, in this case it can be thought of simply as a summation of the set of all possible combinations of scalar products of the elements of the B-splines $M_i(x)$ and $N_j(y)$, at each point of the (x,y) space (or at each panel R_{ij} of the subspace R). For more details on tensor products see: de Boor (1978; chapter XVII).

The general bi-cubic spline polynomial function becomes, then, a linear combination of the new basis function $M_i(x)N_j(y)$, i.e.:

$$s(x,y) = \sum_{i=1}^{h+4} \sum_{j=1}^{k+4} r_{ij} M_i(x)N_j(y) \qquad (11)$$

where:

r_{ij} is the new vector of "weights" (to be determined).

Note that, because the original basis functions $M_i(x)$ and $N_j(y)$ were non-zero over four adjacent intervals only (λ_{i-4}, λ_{i-3}, λ_{i-2}, λ_{i-1}, λ_i and μ_{j-4}, μ_{j-3}, μ_{j-2}, μ_{j-1}, μ_j, respectively), and because of the cross-product properties, the new basis function $M_i(x)N_j(y)$ will be non-zero over only 4x4 = 16 adjacent panels [see shaded area in Fig. C.4]. Similarly, as the originals $M_i(x)$ and $N_j(y)$ had only 4 non-zero basis functions at any point "x", the new basis functions $M_i(x)N_j(y)$ will have only 16 non-zero basis functions at any point (x,y). If this (x,y) point lies in

the panel R_{ij}, the non-zero $M_i(x)N_j(y)$ will be those within the shaded area of Fig. C.4.

Thus, with the help of vectorial cross-products (tensor products), the new bi-cubic B-splines have been introduced, based on the one-dimensional B-splines previously defined. The evaluation of each one-dimensional B-spline [$M_i(x)$ or $N_j(y)$] remains exactly as before, i.e. the stable method (recursive algorithm) proposed by Cox (1972) will be used [equations (4) and (5)], rather than the explicit form [equation(1)], based on the same stability reasons mentioned earlier.

Having the basis functions, the problem of approximating the data points $\{x_r, y_r, f(x_r, y_r)\}$, r=1,2,...m has to be tackled. The spline fitting problem now consists of determining the coefficients r_{ij}, as the least-squares solution of the following overdetermined linear observation system:

$$\sum_{i=1}^{h+4} \sum_{j=1}^{k+4} r_{ij}\, M_i(x_r) N_j(y_r) = f(x_r, y_r) = f_r \qquad r=1,2,\ \ldots\ m \tag{12}$$

which, in matrix format, becomes:

$$A\, \underline{r} = \underline{f} \tag{13}$$

where:

A is a (mx(h+4)(k+4)) matrix.

$\underline{r}$ is an (h+4)(k+4)x1 column vector of unknowns.

$\underline{f}$ is a mx1 column vector, with the data values of the function being approximated.

As in the one-dimensional fitting problem, the structure of A is determined by the way the data are re-ordered; thus, if the data points are sorted according to the panel R_{ij} to which they

belong, allowing the index "i" to run faster than "j" (i.e. following the panels in Fig. C.4 from left to right and from downwards upwards), then the observation matrix has the following structure:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & & & & \\ & A_{22} & A_{23} & A_{24} & A_{34} & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & A_{k+1,k+1} & A_{k+1,k+2} & A_{k+1,k+3} & A_{k+1,k+4} \end{bmatrix}$$

where:

A_{ij} is a rectangular (rx(h+4)) sub-matrix.

r is the number of data points within the panels $R_{1,i}$, $R_{2,i}$, ... , $R_{h+1,i}$ (i.e. the number of data points in the i-th row of panels in Fig. C.4).

With this data arrangement, the vector $\underline{r}$ gets its elements ordered by i=1,2, ... h+4 for fixed "j", where j=1,2, ... k+4.

Note that if r_{ij} is associated with the panel R_{ij}, then the arrangement of $\underline{r}$ follows the panels in Fig. C.4, from left to right and from downwards upwards.

The problem now becomes that of solving the overdetermined linear system (13). In principle, we can proceed as in the one-dimensional case, and form the corresponding normal equations by pre-multiplying (13) by A^T:

$$[A^T A]\ \underline{r} = A^T\ \underline{f} \qquad (14)$$

which is now a (h+4)(k+4)x(h+4)(k+4) linear system of equations.

Unfortunately, for bivariate approximation problems with scattered data, there is no equivalent of the Schoenberg and

Whitney conditions, to guarantee the existence and uniqueness of the solution. To make things worse, it is often the case that the observation matrix A is rank-deficient, usually due to the fact that it may happen that some panels R_{ij} remain without data points within them, which implies that the least squares solution of the observation system has infinite solutions. A unique solution is produced via the addition of a new condition, usually in the sense that, of all the infinite possible solutions, a "minimum norm" solution is sought. This forces us to use more sophisticated linear solvers, able to cope with rank deficiency in a stable manner: singular value decomposition and orthogonal transformations are cited as safe methods to solve the problem [Cox (1986), Cox (1987a)]. These methods belong to a very specialised field of numerical linear algebra and, because of that, we do not present more details here.

C.5. The statistical problem in splines fitting.

From a statistical point of view, the values of the "weights", $\underline{r}$, determined through the solution of the normal equations (10) or (14), for the one-dimensional and two-dimensional cases, respectively, can be interpreted in the case of noisy measurements as the expected values of the weights.

Always in the statistical sense of the problem, we may not only be interested in the expected values of the weights, but also in the corresponding dispersion statistics (as the variance and covariance), in order to estimate the error associated with the approximates computed by the cubic-splines at unmeasured points.

This is particularly true when the measurements are subject to errors, where we want to know the impact of those errors on the spline estimates.

The solution to the latter problem can be obtained in a straightforward manner, when solving the least squares estimation problem, as we did through the normal equations. In fact, all the information required is already contained in the inverse of the matrix A^TA [see Walpole and Myers (1985), section 10.4].

Under the assumption that the measurement errors are independent, with zero mean and variance σ^2, the matrix $\sigma^2[A^TA]^{-1}$ represents the variance-covariance matrix, and displays directly the variances of the weights in the main diagonal, and the covariances in the off-diagonal coefficients. Thus, if:

$$C = [A^TA]^{-1} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1r} \\ c_{21} & c_{22} & \cdots & c_{2r} \\ \vdots & \vdots & & \vdots \\ c_{r1} & c_{r2} & \cdots & c_{rr} \end{bmatrix} \qquad (15)$$

then, the variances are given by:

$$\sigma^2_{r_i} = \text{var}(r_i) = \sigma^2 c_{ii} \quad \forall\ i=1,2,..,r \qquad (16)$$

and the covariances by:

$$\sigma_{r_i r_j} = \text{cov}(r_i, r_j) = \sigma^2 c_{ij} \quad \forall\ i \neq j \qquad (17)$$

The problem now is basically a numerical one, namely: how to obtain the coefficients of the matrix $C = [A^TA]^{-1}$ in (15), in an efficient way. In practice, this means <u>without having to compute explicitly the inverse</u> of A^TA.

The solution can be almost direct, and very efficient from the computational point of view, provided that the normal equations have been solved using a direct Gaussian elimination method, with a Cholesky factorization [see Appendix A for details]. This is actually the main argument in favour of direct solutions when solving the normal equations, since if an iterative scheme (conjugate gradient) is used, the statistical information becomes unavailable [see Cox (1982a)].

Let us assume that we do have the Cholesky factorization of the normal system of equations, say:

$$A^TA = R^TR \tag{18}$$

where:

R is an upper triangular matrix and

R^T is a lower triangular matrix.

Hence, following Cox (1986 and 1987a), to compute the covariance cov (r_i, r_j) we need to specify two (rx1) auxiliary column vectors h_1 and h_2, such that:

$$h_1 = (0,0,\ldots,0,1,0,0,\ldots 0)^T \tag{19}$$

and

$$h_2 = (0,0,\ldots,0,0,1,0,\ldots 0)^T \tag{20}$$

where the "1" is in the i-th and j-th position, respectively.

Then, the covariance cov (r_i, r_j) can be computed as:

$$\text{cov}\ (r_i, r_j) = \sigma^2\ h_1{}^T\ [A^TA]^{-1}h_2 \tag{21}$$

on introducing the Cholesky factorization (18):

$$\text{cov}\ (r_i, r_j) = \sigma^2\ h_1{}^T\ [R^TR]^{-1}h_2 \tag{22}$$

which can be re-written as:

$$\text{cov}\ (r_i, r_j) = \sigma^2\ (R^{-T}h_1)^T(R^{-T}h_2) \tag{23}$$

Upon defining the auxiliary vectors u_1 and u_2 such that:

$$R^T u_1 = h_1 \tag{24}$$

and

$$R^T u_2 = h_2 \tag{25}$$

and noting that u_1 and u_2 can be computed via a forward substitution process, since R^T is a lower triangular matrix (the Cholesky factor) and h_1 and h_2 are known vectors, [once r_i and r_j have been chosen, (equations 19 and 20)], then equation (23) can be re-written as:

$$\boxed{\text{cov}\ (r_i, r_j) = \sigma^2\ u_1^T\ u_2} \tag{26}$$

which gives the covariance.

To compute the variance [var(r_i)], the process is similar, though simpler, because now $h_1 = h_2 = h$, such that:

$$h = (0, 0, \ldots, 0, 0, 1, 0, \ldots 0)^T \tag{27}$$

with the "1" in the i-th position. Then, on applying:

$$\text{var}(r_i) = \sigma^2\ h^T\ (A^TA)^{-1}\ h \tag{28}$$

and following the same procedure as before, we need to solve:

$$R^T\ u = h \tag{29}$$

and compute the variance as:

$$\boxed{\text{var}(r_i) = \sigma^2\ u^T\ u} \tag{30}$$

This is the efficient way to compute the variance and covariances, since it only requires two forward substitutions plus one inner product, to compute each covariance, and only one

forward substitution plus an inner product for each variance. The explicit inversion of $[A^TA]$ has been avoided.

The variance can be computed for Γ_i i=1,2,...,r , where "r" is the number of weights, i.e.: r=h+4 and r=(h+4)(k+4), for the one-dimensional and two-dimensional cubic splines, respectively.

Upon having the variance, and because the spline polynomial is just a linear combination of weighted B-splines, we can compute the respective variance of the splines estimates using the definition of the spline, and the following property of the variances of two independent random variables X and Y:

$$\mathrm{var}(aX+bY) = a^2\,\mathrm{var}(X) + b^2\,\mathrm{var}(Y) \qquad (31)$$

Thus, for the one-dimensional spline case:

$$\mathrm{var}(\,s(x)\,) = \mathrm{var}\Big(\sum_{i=1}^{h+4} M_i(x)\,\Gamma_i\Big) = \sum_{i=1}^{h+4} \{\,M_i(x)\,\}^2\,\mathrm{var}(\Gamma_i) \qquad (32)$$

while, for the bi-cubic splines, a similar formula is obtained.

APPENDIX D

NUMERICAL RESULTS OF THE CALIBRATION EXERCISE

Table D.1. Summary of the comparison between true, initial and calibrated piezometric heads. Example A.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	62.136954	1.780894	1.334501				
	INITIAL	62.040502	1.810364	1.345498	0.130198	0.096452	0.001502	0.998448
I	INTERPOL.	62.123656	1.766405	1.329062	0.053762	0.014019	0.000439	0.999786
	KRIGING	62.343461	1.602880	1.266049	0.643541	0.211023	0.088072	1.003323
	SPLINES	36.283353	403.289659	20.082073	51.459823	25.900618	402.249599	0.583926
	TRUE	62.136954	1.780894	1.334501				
	INITIAL	62.040502	1.810364	1.345498	0.130198	0.096452	0.001502	0.998448
II	INTERPOL.	62.122005	1.768055	1.329682	0.055034	0.015274	0.000431	0.999759
	KRIGING	62.356386	1.589679	1.260825	0.657572	0.220374	0.090076	1.003531
	SPLINES	33.503773	617.090146	24.841299	76.383997	28.680198	631.640732	0.539192
	TRUE	62.136954	1.780894	1.334501				
	INITIAL	62.040502	1.810364	1.345498	0.130198	0.096452	0.001502	0.998448
III	INTERPOL.	62.124011	1.765906	1.328874	0.053375	0.013766	0.000383	0.999792
	KRIGING	62.354969	1.591085	1.261382	0.655473	0.219276	0.089847	1.003509
	SPLINES	33.578977	617.007472	24.839635	76.361830	28.604994	631.661541	0.540403
	TRUE	62.136954	1.780894	1.334501				
	INITIAL	62.040502	1.810364	1.345498	0.130198	0.096452	0.001502	0.998448
IV	INTERPOL.	62.109964	1.684000	1.297690	0.195602	0.055025	0.005610	0.999566
	KRIGING	62.355181	1.590768	1.261257	0.654835	0.219255	0.089969	1.003512
	SPLINES	33.560687	624.443795	24.988873	76.545792	28.623284	639.937695	0.540108
	TRUE	62.136954	1.780894	1.334501				
	INITIAL	62.040502	1.810364	1.345498	0.130198	0.096452	0.001502	0.998448
V	INTERPOL.	62.110472	1.798262	1.340993	0.108172	0.026649	0.001488	0.999574
	KRIGING	62.339564	1.606536	1.267492	0.646702	0.209684	0.086764	1.003261
	SPLINES	34.046619	540.108914	23.240243	71.219991	28.137352	546.933422	0.547929

NOTES:
=====
[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))
[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N
[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)
SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS
AVR : AVERAGE ABSOLUTE VALUE RESIDUALS
N : NUMBER OF DATA POINTS
[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.2. Summary of the comparison between true, initial and calibrated piezometric heads. Example B.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	53.068512	35.740184	5.978309				
	INITIAL	52.717053	35.932102	5.994339	0.500262	0.351459	0.023382	0.993377
I	INTERPOL.	53.068335	35.250806	5.937239	0.204632	0.046582	0.004287	0.999997
	KRIGING	53.932906	32.593471	5.709069	2.741375	0.865810	1.712570	1.016288
	SPLINES	34.311754	425.290821	20.622580	49.416708	19.209650	279.293286	0.646556
	TRUE	53.068512	35.740184	5.978309				
	INITIAL	52.717053	35.932102	5.994339	0.500262	0.351459	0.023382	0.993377
II	INTERPOL.	53.065770	35.261951	5.938177	0.204608	0.046917	0.004126	0.999948
	KRIGING	53.938091	32.571324	5.707129	2.747570	0.869579	1.716390	1.016386
	SPLINES	34.253451	425.983291	20.639363	49.413179	19.267953	279.822898	0.645457
	TRUE	53.068512	35.740184	5.978309				
	INITIAL	52.717053	35.932102	5.994339	0.500262	0.351459	0.023382	0.993377
III	INTERPOL.	53.069100	35.247428	5.936954	0.204608	0.046862	0.004300	1.000011
	KRIGING	53.940904	32.559316	5.706077	2.751158	0.872392	1.716841	1.016439
	SPLINES	34.399069	423.970079	20.590534	49.423569	19.122335	278.393582	0.648201
	TRUE	53.068512	35.740184	5.978309				
	INITIAL	52.717053	35.932102	5.994339	0.500262	0.351459	0.023382	0.993377
IV	INTERPOL.	53.102704	33.922002	5.824260	0.749739	0.178470	0.071623	1.000644
	KRIGING	53.934059	32.588918	5.708670	2.742666	0.866494	1.713824	1.016310
	SPLINES	35.306851	372.016585	19.287731	45.259086	18.214553	225.682231	0.665307
	TRUE	53.068512	35.740184	5.978309				
	INITIAL	52.717053	35.932102	5.994339	0.500262	0.351459	0.023382	0.993377
V	INTERPOL.	52.805732	36.798907	6.066210	1.098131	0.266632	0.110035	0.995048
	KRIGING	53.767893	33.305292	5.771074	2.539383	0.887150	1.332804	1.013179
	SPLINES	33.952638	422.206230	20.547658	50.320178	19.568768	274.153415	0.639789

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.3. Summary of the comparison between true, initial and calibrated piezometric heads. Example C.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	62.136956	1.780891	1.334500				
	INITIAL	62.878059	0.938416	0.968719	1.066049	0.741103	0.149333	1.011927
I	INTERPOL.	62.201538	1.641184	1.281087	0.243984	0.082440	0.009005	1.001039
	KRIGING	62.341660	1.604408	1.266652	0.645151	0.210333	0.087045	1.003294
	SPLINES	33.516220	621.797560	24.935869	76.730861	28.667753	636.829002	0.539393
	TRUE	62.136956	1.780891	1.334500				
	INITIAL	62.878059	0.938416	0.968719	1.066049	0.741103	0.149333	1.011927
II	INTERPOL.	62.199914	1.642798	1.281717	0.240220	0.082454	0.008611	1.001013
	KRIGING	62.350918	1.595251	1.263032	0.654034	0.216252	0.089106	1.003443
	SPLINES	33.473344	622.038356	24.940697	76.725757	28.710629	637.057828	0.538703
	TRUE	62.136956	1.780891	1.334500				
	INITIAL	62.878059	0.938416	0.968719	1.066049	0.741103	0.149333	1.011927
III	INTERPOL.	62.202519	1.640165	1.280689	0.245286	0.082802	0.009111	1.001055
	KRIGING	62.355235	1.590922	1.261317	0.658027	0.219529	0.089857	1.003513
	SPLINES	33.602055	621.290876	24.925707	76.734329	28.581918	636.377371	0.540774
	TRUE	62.136956	1.780891	1.334500				
	INITIAL	62.698589	0.981271	0.990591	0.900043	0.561633	0.142984	1.009039
IV	INTERPOL.	62.150291	1.651821	1.285232	0.163188	0.075048	0.006495	1.000215
	KRIGING	62.350108	1.595858	1.263273	0.649373	0.215389	0.089098	1.003430
	SPLINES	33.560637	624.445852	24.988915	76.545904	28.623336	639.939976	0.540108
	TRUE	62.136956	1.780891	1.334500				
	INITIAL	62.741539	1.102719	1.050104	0.847173	0.604583	0.090009	1.009730
V	INTERPOL.	62.186270	1.752223	1.323716	0.208746	0.056611	0.005916	1.000794
	KRIGING	62.314798	1.631318	1.277231	0.619053	0.203224	0.077470	1.002862
	SPLINES	34.226023	523.721954	22.884972	70.013527	27.957950	528.936504	0.550816

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.4. Summary of the comparison between true, initial and calibrated piezometric heads. Example D.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	23.469569	92.318030	9.608227				
	INITIAL	22.684429	93.275043	9.657901	1.141310	0.785140	0.009442	0.966546
I	INTERPOL.	23.391649	90.859003	9.531999	0.868950	0.082335	0.036972	0.996680
	KRIGING	24.309381	154.818543	12.442610	23.097968	1.403829	17.676265	1.035783
	SPLINES	23.644199	118.963877	10.907056	13.247504	0.818050	4.345164	1.007441
	TRUE	23.469569	92.318030	9.608227				
	INITIAL	22.684429	93.275043	9.657901	1.141310	0.785140	0.009442	0.966546
II	INTERPOL.	23.389110	90.875278	9.532853	0.870788	0.083621	0.036733	0.996572
	KRIGING	24.327721	154.661081	12.436281	23.103627	1.385520	17.701572	1.036564
	SPLINES	23.668704	118.754159	10.897438	13.253927	0.804035	4.380930	1.008485
	TRUE	23.469569	92.318030	9.608227				
	INITIAL	22.684429	93.275043	9.657901	1.141310	0.785140	0.009442	0.966546
III	INTERPOL.	23.401347	90.809464	9.529400	0.856382	0.086661	0.035715	0.997093
	KRIGING	24.313210	154.788988	12.441422	23.098135	1.412530	17.727623	1.035946
	SPLINES	23.705721	118.721159	10.895924	13.247997	0.795535	4.367497	1.010062
	TRUE	23.469569	92.318030	9.608227				
	INITIAL	22.684429	93.275043	9.657901	1.141310	0.785140	0.009442	0.966546
IV	INTERPOL.	23.388163	90.417509	9.508812	1.236070	0.123363	0.056530	0.996531
	KRIGING	24.323520	154.703687	12.437994	23.097093	1.412602	17.745359	1.036385
	SPLINES	23.705645	118.719071	10.895828	13.248164	0.795328	4.368942	1.010059
	TRUE	23.469569	92.318030	9.608227				
	INITIAL	22.684429	93.275043	9.657901	1.141310	0.785140	0.009442	0.966546
V	INTERPOL.	23.195262	89.943616	9.483861	1.785770	0.274644	0.184791	0.988312
	KRIGING	23.661852	155.801525	12.482048	22.440190	1.908507	15.371171	1.008193
	SPLINES	22.795223	120.156063	10.961572	12.332561	1.492429	3.159400	0.971267

NOTES:
=====
[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))
[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N
[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)
SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS
AVR : AVERAGE ABSOLUTE VALUE RESIDUALS
N : NUMBER OF DATA POINTS
[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.5. Summary of the comparison between true, initial and calibrated piezometric heads. Example E.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	90.580945	6.435581	2.536845				
	INITIAL	90.565063	6.444084	2.538520	0.098798	0.019310	0.000276	0.999825
I	INTERPOL.	90.576717	6.438456	2.537411	0.091366	0.009282	0.000274	0.999953
	KRIGING	91.090711	10.609632	3.257243	5.828270	0.580414	1.460438	1.005628
	SPLINES	90.641002	8.703889	2.950235	3.736779	0.372636	0.419807	1.000663
	TRUE	90.580945	6.435581	2.536845				
	INITIAL	90.565063	6.444084	2.538520	0.098798	0.019310	0.000276	0.999825
II	INTERPOL.	90.576821	6.438314	2.537383	0.090545	0.009197	0.000275	0.999954
	KRIGING	91.099321	10.613703	3.257868	5.833304	0.580345	1.472233	1.005723
	SPLINES	90.633813	8.681062	2.946364	3.723417	0.369745	0.406742	1.000584
	TRUE	90.580945	6.435581	2.536845				
	INITIAL	90.565063	6.444084	2.538520	0.098798	0.019310	0.000276	0.999825
III	INTERPOL.	90.576989	6.438518	2.537423	0.089223	0.009100	0.000273	0.999956
	KRIGING	91.119078	10.677210	3.267600	5.965369	0.586325	1.521479	1.005941
	SPLINES	90.696569	8.683292	2.946743	3.877208	0.345688	0.448671	1.001276
	TRUE	90.580945	6.435581	2.536845				
	INITIAL	90.565063	6.444084	2.538520	0.098798	0.019310	0.000276	0.999825
IV	INTERPOL.	90.581264	6.431398	2.536020	0.128526	0.013024	0.000556	1.000004
	KRIGING	91.116472	10.675143	3.267284	5.959070	0.585652	1.517881	1.005912
	SPLINES	90.679997	8.646093	2.940424	3.792621	0.342484	0.429829	1.001094
	TRUE	90.580945	6.435581	2.536845				
	INITIAL	90.565063	6.444084	2.538520	0.098798	0.019310	0.000276	0.999825
V	INTERPOL.	90.560459	6.430524	2.535848	0.121307	0.022875	0.000486	0.999774
	KRIGING	91.124170	10.692372	3.269919	5.985603	0.589721	1.531876	1.005997
	SPLINES	90.686189	8.691012	2.948052	3.869142	0.349365	0.444341	1.001162

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.6. Summary of the comparison between true, initial and calibrated piezometric heads. Example F.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	68.544348	376.086468	19.392949				
	INITIAL	68.415341	370.880707	19.258263	0.376937	0.165889	0.014416	0.998118
I	INTERPOL.	68.528709	374.788283	19.359449	0.290282	0.041405	0.003382	0.999772
	KRIGING	69.688396	388.812582	19.718331	8.010388	1.381892	2.680300	1.016691
	SPLINES	55.582129	865.717269	29.423074	36.994771	13.295901	207.565165	0.810893
	TRUE	68.544348	376.086468	19.392949				
	INITIAL	68.415341	370.880707	19.258263	0.376937	0.165889	0.014416	0.998118
II	INTERPOL.	68.518600	374.463647	19.351063	0.306612	0.050655	0.003443	0.999624
	KRIGING	69.696173	389.074867	19.724981	8.031336	1.380769	2.690190	1.016804
	SPLINES	55.349084	894.105688	29.901600	36.911984	13.841272	211.172737	0.807493
	TRUE	68.544348	376.086468	19.392949				
	INITIAL	68.415341	370.880707	19.258263	0.376937	0.165889	0.014416	0.998118
III	INTERPOL.	68.509465	374.194520	19.344108	0.329019	0.058857	0.003935	0.999491
	KRIGING	69.705559	390.055931	19.749834	8.085776	1.395836	2.709800	1.016941
	SPLINES	58.515908	661.971743	25.728812	38.447331	10.777761	139.034831	0.853694
	TRUE	68.544348	376.086468	19.392949				
	INITIAL	68.415341	370.880707	19.258263	0.376937	0.165889	0.014416	0.998118
IV	INTERPOL.	68.497198	373.883877	19.336077	0.460254	0.071618	0.007033	0.999312
	KRIGING	69.700394	389.899959	19.745885	8.076814	1.393295	2.701085	1.016869
	SPLINES	58.511196	661.797609	25.725427	38.451710	10.781083	138.996665	0.853625
	TRUE	68.544348	376.086468	19.392949				
	INITIAL	68.415341	370.880707	19.258263	0.376937	0.165889	0.014416	0.998118
V	INTERPOL.	68.476801	372.963306	19.312258	0.448014	0.091037	0.012138	0.999015
	KRIGING	69.674382	389.029010	19.723818	7.999846	1.381557	2.654147	1.016486
	SPLINES	58.658442	646.886053	25.433955	38.507682	10.611937	134.807513	0.855774

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = { SSAVR - N * AVR**2 } / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.7. Summary of the performance of the calibrated heads. Example A.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	0.981	0.758	0.829	0.915
	KRIGING	-3.584	-35.488	-23.431	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
II	INTERPOL.	0.976	0.810	0.821	0.918
	KRIGING	-4.176	-41.100	-24.508	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
III	INTERPOL.	0.982	0.741	0.832	0.935
	KRIGING	-4.109	-40.483	-24.346	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
IV	INTERPOL.	0.922	-9.810	-1.257	-12.950
	KRIGING	-4.119	-40.622	-24.296	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
V	INTERPOL.	0.925	0.653	0.310	0.019
	KRIGING	-3.413	-34.004	-23.672	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.957	-1.370	0.307	-2.033
KRIGING	-3.880	-38.340	-24.051	-999.999
SPLINES	-999.999	-999.999	-999.999	-999.999

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	100.000	80.000	80.000	80.000
KRIGING	0.000	0.000	0.000	0.000
SPLINES	0.000	0.000	0.000	0.000

Table D.8. Summary of the performance of the calibrated heads. Example B.

CASE	PROCEDURE	PERFORMANCE INDEX AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	1.000	-5.502	0.833	0.966
	KRIGING	-5.049	-267.833	-29.029	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
II	INTERPOL.	1.000	-5.209	0.833	0.969
	KRIGING	-5.122	-271.631	-29.165	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
III	INTERPOL.	1.000	-5.592	0.833	0.966
	KRIGING	-5.161	-273.701	-29.244	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
IV	INTERPOL.	0.991	-88.752	-1.246	-8.383
	KRIGING	-5.065	-268.612	-29.057	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
V	INTERPOL.	0.441	-29.432	-3.819	-21.146
	KRIGING	-2.960	-159.964	-24.767	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.886	-26.898	-0.513	-5.326
KRIGING	-4.671	-248.348	-28.252	-999.999
SPLINES	-999.999	-999.999	-999.999	-999.999

PROCEDURE	FREQUENCY OF IMPROVEMENT (%) AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	100.000	0.000	60.000	60.000
KRIGING	0.000	0.000	0.000	0.000
SPLINES	0.000	0.000	0.000	0.000

Table D.9. Summary of the performance of the calibrated heads. Example C.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	0.992	0.973	0.948	0.996
	KRIGING	0.924	0.956	0.634	0.660
	SPLINES	-999.999	-999.999	-999.999	-999.999
II	INTERPOL.	0.993	0.973	0.949	0.997
	KRIGING	0.917	0.951	0.624	0.644
	SPLINES	-999.999	-999.999	-999.999	-999.999
III	INTERPOL.	0.992	0.972	0.947	0.996
	KRIGING	0.913	0.949	0.619	0.638
	SPLINES	-999.999	-999.999	-999.999	-999.999
IV	INTERPOL.	0.999	0.974	0.967	0.998
	KRIGING	0.856	0.946	0.479	0.612
	SPLINES	-999.999	-999.999	-999.999	-999.999
V	INTERPOL.	0.993	0.998	0.939	0.996
	KRIGING	0.913	0.951	0.466	0.259
	SPLINES	-999.999	-999.999	-999.999	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.994	0.978	0.950	0.997
KRIGING	0.905	0.951	0.564	0.563
SPLINES	-999.999	-999.999	-999.999	-999.999

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	100.000	100.000	100.000	100.000
KRIGING	100.000	100.000	100.000	100.000
SPLINES	0.000	0.000	0.000	0.000

Table D.10. Summary of the performance of the calibrated heads. Example D.

CASE	PROCEDURE	PERFORMANCE INDEX AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	0.990	-1.324	0.420	-14.333
	KRIGING	-0.144	-999.999	-408.581	-999.999
	SPLINES	0.951	-774.217	-133.729	-999.999
II	INTERPOL.	0.989	-1.273	0.418	-14.135
	KRIGING	-0.195	-999.999	-408.782	-999.999
	SPLINES	0.936	-762.062	-133.860	-999.999
III	INTERPOL.	0.992	-1.485	0.437	-13.308
	KRIGING	-0.155	-999.999	-408.587	-999.999
	SPLINES	0.910	-760.159	-133.739	-999.999
IV	INTERPOL.	0.989	-2.944	-0.173	-34.845
	KRIGING	-0.183	-999.999	-408.550	-999.999
	SPLINES	0.910	-760.038	-133.742	-999.999
V	INTERPOL.	0.878	-5.156	-1.448	-382.031
	KRIGING	0.940	-999.999	-385.586	-999.999
	SPLINES	0.262	-845.138	-115.761	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.968	-2.436	-0.069	-91.730
KRIGING	0.053	-999.999	-404.017	-999.999
SPLINES	0.794	-780.323	-130.166	-999.999

PROCEDURE	FREQUENCY OF IMPROVEMENT (%) AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	100.000	0.000	60.000	0.000
KRIGING	20.000	0.000	0.000	0.000
SPLINES	100.000	0.000	0.000	0.000

Table D.11. Summary of the performance of the calibrated heads. Example E.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	0.929	0.886	0.145	0.014
	KRIGING	-999.999	-999.999	-999.999	-999.999
	SPLINES	-13.299	-999.999	-999.999	-999.999
II	INTERPOL.	0.933	0.897	0.160	0.007
	KRIGING	-999.999	-999.999	-999.999	-999.999
	SPLINES	-10.081	-999.999	-999.999	-999.999
III	INTERPOL.	0.938	0.881	0.184	0.022
	KRIGING	-999.999	-999.999	-999.999	-999.999
	SPLINES	-52.001	-999.999	-999.999	-999.999
IV	INTERPOL.	1.000	0.758	-0.692	-3.058
	KRIGING	-999.999	-999.999	-999.999	-999.999
	SPLINES	-37.897	-999.999	-999.999	-999.999
V	INTERPOL.	-0.664	0.646	-0.508	-2.101
	KRIGING	-999.999	-999.999	-999.999	-999.999
	SPLINES	-42.912	-999.999	-999.999	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.627	0.813	-0.142	-1.023
KRIGING	-999.999	-999.999	-999.999	-999.999
SPLINES	-31.238	-999.999	-999.999	-999.999

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	80.000	100.000	60.000	60.000
KRIGING	0.000	0.000	0.000	0.000
SPLINES	0.000	0.000	0.000	0.000

Table D.12. Summary of the performance of the calibrated heads. Example F.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	0.985	0.938	0.407	0.945
	KRIGING	-77.643	-4.976	-450.616	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
II	INTERPOL.	0.960	0.903	0.338	0.943
	KRIGING	-78.716	-5.225	-452.981	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
III	INTERPOL.	0.927	0.868	0.238	0.925
	KRIGING	-80.021	-6.201	-459.157	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
IV	INTERPOL.	0.866	0.821	-0.491	0.762
	KRIGING	-79.329	-6.041	-458.137	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
V	INTERPOL.	0.726	0.640	-0.413	0.291
	KRIGING	-75.728	-5.181	-449.428	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.893	0.834	0.016	0.773
KRIGING	-78.288	-5.525	-454.064	-999.999
SPLINES	-999.999	-999.999	-999.999	-999.999

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	100.000	100.000	60.000	100.000
KRIGING	0.000	0.000	0.000	0.000
SPLINES	0.000	0.000	0.000	0.000

Table D.13. Summary of the comparison between true, initial and calibrated flows. Example A.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	2.856538	14.284065	3.779427				
	INITIAL	2.859369	14.272675	3.777919	0.047100	0.019569	0.000320	1.000991
I	INTERPOL.	2.859375	14.271362	3.777746	0.050200	0.018475	0.000272	1.000993
	KRIGING	2.817956	14.436289	3.799512	0.617300	0.169644	0.041886	0.986494
	SPLINES	2.880875	14.210631	3.769699	0.389500	0.137150	0.021196	1.008520
	TRUE	2.856538	14.284065	3.779427				
	INITIAL	2.859369	14.272675	3.777919	0.047100	0.019569	0.000320	1.000991
II	INTERPOL.	2.855188	14.280479	3.778952	0.039200	0.012700	0.000137	0.999527
	KRIGING	2.851169	14.420433	3.797424	0.196100	0.057406	0.004129	0.998121
	SPLINES	2.840825	14.138351	3.760100	0.392700	0.101812	0.019210	0.994499
	TRUE	2.856538	14.284065	3.779427				
	INITIAL	2.859369	14.272675	3.777919	0.047100	0.019569	0.000320	1.000991
III	INTERPOL.	2.858187	14.280074	3.778898	0.029200	0.010775	0.000092	1.000578
	KRIGING	2.856588	14.384611	3.792705	0.191700	0.063800	0.003917	1.000018
	SPLINES	2.877188	14.141629	3.760536	0.369200	0.090925	0.013911	1.007229
	TRUE	2.856538	14.284065	3.779427				
	INITIAL	2.859369	14.272675	3.777919	0.047100	0.019569	0.000320	1.000991
IV	INTERPOL.	2.859187	14.282264	3.779188	0.160800	0.049100	0.002967	1.000928
	KRIGING	2.856431	14.387656	3.793106	0.190100	0.078156	0.004605	0.999963
	SPLINES	2.879231	14.147690	3.761342	0.280900	0.102869	0.009580	1.007944
	TRUE	2.856538	14.284065	3.779427				
	INITIAL	2.859369	14.272675	3.777919	0.047100	0.019569	0.000320	1.000991
V	INTERPOL.	2.857250	14.282359	3.779201	0.037200	0.011712	0.000186	1.000249
	KRIGING	2.953675	14.927282	3.863584	0.493500	0.175388	0.023376	1.034005
	SPLINES	2.967669	14.557097	3.815376	0.750500	0.175556	0.049815	1.038904

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.{ABS(TRUE-ESTIMATED)}

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = { SSAVR - N * AVR**2 } / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.14. Summary of the comparison between true, initial and calibrated flows. Example B.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	6.364231	59.767118	7.730920				
	INITIAL	6.370150	59.722451	7.728030	0.120100	0.052619	0.002130	1.000930
I	INTERPOL.	6.370125	59.721357	7.727959	0.119800	0.052544	0.002133	1.000926
	KRIGING	6.364119	59.754468	7.730101	0.129000	0.054500	0.001539	0.999982
	SPLINES	6.403319	59.590010	7.719457	0.625300	0.228337	0.050065	1.006142
	TRUE	6.364231	59.767118	7.730920				
	INITIAL	6.370150	59.722451	7.728030	0.120100	0.052619	0.002130	1.000930
II	INTERPOL.	6.362062	59.781664	7.731860	0.118800	0.039594	0.001774	0.999659
	KRIGING	6.362281	59.814570	7.733988	0.142300	0.051950	0.001880	0.999694
	SPLINES	6.355156	59.281006	7.699416	0.545200	0.167837	0.041532	0.998574
	TRUE	6.364231	59.767118	7.730920				
	INITIAL	6.370150	59.722451	7.728030	0.120100	0.052619	0.002130	1.000930
III	INTERPOL.	6.364687	59.778066	7.731628	0.118800	0.034981	0.002042	1.000072
	KRIGING	6.364988	59.814052	7.733955	0.094900	0.040094	0.001251	1.000119
	SPLINES	6.372306	59.203720	7.694395	0.580600	0.177437	0.042499	1.001269
	TRUE	6.364231	59.767118	7.730920				
	INITIAL	6.370150	59.722451	7.728030	0.120100	0.052619	0.002130	1.000930
IV	INTERPOL.	6.365875	59.765616	7.730822	0.119800	0.036169	0.001956	1.000258
	KRIGING	6.364444	59.832539	7.735150	0.340500	0.103550	0.017282	1.000033
	SPLINES	6.381637	58.795146	7.667799	1.008700	0.327106	0.129761	1.002735
	TRUE	6.364231	59.767118	7.730920				
	INITIAL	6.370150	59.722451	7.728030	0.120100	0.052619	0.002130	1.000930
V	INTERPOL.	6.360250	59.776458	7.731524	0.495800	0.185394	0.024362	0.999374
	KRIGING	6.636481	63.835945	7.989740	1.157000	0.382650	0.096651	1.042778
	SPLINES	6.681875	62.332049	7.895065	3.847700	0.910531	1.259681	1.049911

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.15. Summary of the comparison between true, initial and calibrated flows. Example C.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	2.856538	14.284065	3.779427				
	INITIAL	2.858787	14.277518	3.778560	0.073300	0.031275	0.000477	1.000788
I	INTERPOL.	2.858813	14.280935	3.779013	0.139800	0.058012	0.002114	1.000796
	KRIGING	2.843750	14.320438	3.784235	0.204500	0.061863	0.004110	0.995523
	SPLINES	2.883269	14.204004	3.768820	0.427900	0.146269	0.025934	1.009358
	TRUE	2.856538	14.284065	3.779427				
	INITIAL	2.858787	14.277518	3.778560	0.073300	0.031275	0.000477	1.000788
II	INTERPOL.	2.864813	14.226220	3.771766	0.138800	0.046013	0.001928	1.002897
	KRIGING	2.853413	14.393494	3.793876	0.161600	0.041737	0.002391	0.998906
	SPLINES	2.841006	14.132745	3.759354	0.378800	0.099606	0.018068	0.994563
	TRUE	2.856538	14.284065	3.779427				
	INITIAL	2.858787	14.277518	3.778560	0.073300	0.031275	0.000477	1.000788
III	INTERPOL.	2.857250	14.293493	3.780674	0.144200	0.051950	0.002589	1.000249
	KRIGING	2.856688	14.386401	3.792941	0.195100	0.057787	0.003410	1.000053
	SPLINES	2.880925	14.132468	3.759317	0.360100	0.093737	0.015501	1.008537
	TRUE	2.856538	14.284065	3.779427				
	INITIAL	2.865569	14.252602	3.775262	0.150500	0.062294	0.003243	1.003162
IV	INTERPOL.	2.858187	14.284406	3.779472	0.067600	0.026787	0.000526	1.000578
	KRIGING	2.855669	14.370472	3.790841	0.151000	0.067819	0.003061	0.999696
	SPLINES	2.879238	14.147652	3.761337	0.280900	0.102875	0.009580	1.007947
	TRUE	2.856538	14.284065	3.779427				
	INITIAL	2.971906	14.869644	3.856118	0.383700	0.161781	0.013103	1.040388
V	INTERPOL.	2.843750	14.330026	3.785502	0.326600	0.093087	0.008230	0.995523
	KRIGING	2.950088	14.859979	3.854864	0.342900	0.155000	0.013841	1.032749
	SPLINES	2.966525	14.538424	3.812929	0.733500	0.178225	0.045836	1.038504

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.16. Summary of the comparison between true, initial and calibrated flows. Example D.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	5.000003	46.933771	6.850823				
	INITIAL	4.999995	46.941706	6.851402	0.523500	0.064099	0.008231	0.999998
I	INTERPOL.	5.000039	47.312672	6.878421	3.840300	0.294291	0.307374	1.000007
	KRIGING	4.999993	62.404206	7.899633	16.024600	2.165528	10.264412	0.999998
	SPLINES	4.999997	57.302259	7.569826	10.014300	2.081771	5.146496	0.999999
	TRUE	5.000003	46.933771	6.850823				
	INITIAL	4.999995	46.941706	6.851402	0.523500	0.064099	0.008231	0.999998
II	INTERPOL.	4.997800	47.360765	6.881916	4.315300	0.308802	0.355305	0.999559
	KRIGING	5.061939	69.411966	8.331384	21.050000	2.454491	15.383614	1.012387
	SPLINES	5.116812	57.417879	7.577459	9.752800	2.017170	5.133868	1.023362
	TRUE	5.000003	46.933771	6.850823				
	INITIAL	4.999995	46.941706	6.851402	0.523500	0.064099	0.008231	0.999998
III	INTERPOL.	5.005050	47.203915	6.870511	3.332300	0.220570	0.224825	1.001009
	KRIGING	4.993333	72.943998	8.540726	33.209000	2.555598	18.477858	0.998666
	SPLINES	5.074139	57.927176	7.610991	12.566800	2.059450	5.654480	1.014827
	TRUE	5.000003	46.933771	6.850823				
	INITIAL	4.999995	46.941706	6.851402	0.523500	0.064099	0.008231	0.999998
IV	INTERPOL.	5.002828	47.563499	6.896630	3.905700	0.437233	0.312139	1.000565
	KRIGING	4.994833	74.154957	8.611327	33.654300	2.615462	19.334366	0.998966
	SPLINES	5.073988	57.994937	7.615441	12.644100	2.067881	5.680241	1.014797
	TRUE	5.000003	46.933771	6.850823				
	INITIAL	4.999995	46.941706	6.851402	0.523500	0.064099	0.008231	0.999998
V	INTERPOL.	4.999900	51.326560	7.164256	9.799800	1.020781	2.927971	0.999979
	KRIGING	5.072791	75.137247	8.668174	34.481700	2.632107	19.451386	1.014557
	SPLINES	5.152630	59.058413	7.684947	12.571500	2.103636	5.817014	1.030525

NOTES:
=====
[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))
[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N
[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)
SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS
AVR : AVERAGE ABSOLUTE VALUE RESIDUALS
N : NUMBER OF DATA POINTS
[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.17. Summary of the comparison between true, initial and calibrated flows. Example E.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	-0.714126	21.495997	4.636378				
	INITIAL	-0.717209	21.352192	4.620843	0.269600	0.044977	0.002412	1.004318
I	INTERPOL.	-0.716406	21.372964	4.623090	0.443100	0.071807	0.004806	1.003193
	KRIGING	-0.336327	18.456918	4.296152	4.446700	1.235058	1.213173	0.470963
	SPLINES	-0.647950	23.764316	4.874866	5.367700	1.073869	1.134717	0.907333
	TRUE	-0.714126	21.495997	4.636378				
	INITIAL	-0.717209	21.352192	4.620843	0.269600	0.044977	0.002412	1.004318
II	INTERPOL.	-0.706772	21.416409	4.627787	0.365100	0.058980	0.004430	0.989703
	KRIGING	-0.318991	18.719249	4.326575	4.053400	1.094189	1.018508	0.446687
	SPLINES	-0.657779	24.470366	4.946753	9.343000	1.092908	1.740104	0.921097
	TRUE	-0.714126	21.495997	4.636378				
	INITIAL	-0.717209	21.352192	4.620843	0.269600	0.044977	0.002412	1.004318
III	INTERPOL.	-0.714139	21.494502	4.636216	0.383100	0.053401	0.003729	1.000019
	KRIGING	-0.695466	24.594553	4.959290	9.004900	1.113822	2.452523	0.973870
	SPLINES	-0.694970	24.164441	4.915734	8.409400	1.006849	1.309989	0.973176
	TRUE	-0.714126	21.495997	4.636378				
	INITIAL	-0.717209	21.352192	4.620843	0.269600	0.044977	0.002412	1.004318
IV	INTERPOL.	-0.713978	21.502128	4.637039	0.491700	0.074714	0.008010	0.999793
	KRIGING	-0.695793	24.834116	4.983384	9.338200	1.148873	2.625475	0.974329
	SPLINES	-0.699783	26.184839	5.117112	13.391900	1.251661	2.556300	0.979916
	TRUE	-0.714126	21.495997	4.636378				
	INITIAL	-0.717209	21.352192	4.620843	0.269600	0.044977	0.002412	1.004318
V	INTERPOL.	-0.715056	21.522782	4.639265	0.835100	0.127837	0.018792	1.001302
	KRIGING	-0.685712	23.741603	4.872536	7.792400	0.989923	1.780893	0.960212
	SPLINES	-0.685941	24.260616	4.925507	8.177600	1.040444	1.295714	0.960533

NOTES:
=====
[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.{ABS(TRUE-ESTIMATED)}
[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N
[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)
SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS
AVR : AVERAGE ABSOLUTE VALUE RESIDUALS
N : NUMBER OF DATA POINTS
[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.18. Summary of the comparison between true, initial and calibrated flows. Example F.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	4.255524	31.234204	5.588757				
	INITIAL	4.274763	31.246031	5.589815	0.838200	0.084158	0.009166	1.004521
I	INTERPOL.	4.276528	31.246759	5.589880	0.875700	0.102532	0.016960	1.004936
	KRIGING	4.269430	35.985537	5.998795	9.800500	1.573010	2.849809	1.003268
	SPLINES	2.848872	43.911869	6.626603	28.928800	4.924292	18.056179	0.669453
	TRUE	4.255524	31.234204	5.588757				
	INITIAL	4.274763	31.246031	5.589815	0.838200	0.084158	0.009166	1.004521
II	INTERPOL.	4.251584	31.305232	5.595108	0.822700	0.082948	0.012840	0.999074
	KRIGING	4.256825	36.907174	6.075128	11.222500	1.557878	3.487768	1.000306
	SPLINES	2.807373	32.011668	5.657885	14.799500	4.107767	10.423112	0.659701
	TRUE	4.255524	31.234204	5.588757				
	INITIAL	4.274763	31.246031	5.589815	0.838200	0.084158	0.009166	1.004521
III	INTERPOL.	4.252107	31.164734	5.582538	0.860700	0.075950	0.011884	0.999197
	KRIGING	4.281368	35.606763	5.967140	7.750000	1.369353	2.605558	1.006073
	SPLINES	2.961984	37.289550	6.106517	18.144300	3.621890	10.546634	0.696033
	TRUE	4.255524	31.234204	5.588757				
	INITIAL	4.274763	31.246031	5.589815	0.838200	0.084158	0.009166	1.004521
IV	INTERPOL.	4.252837	31.158797	5.582007	0.865700	0.069032	0.009102	0.999369
	KRIGING	4.281750	35.596979	5.966320	7.757700	1.364954	2.608005	1.006163
	SPLINES	2.960927	37.272008	6.105081	18.150000	3.628024	10.512316	0.695784
	TRUE	4.255524	31.234204	5.588757				
	INITIAL	4.274763	31.246031	5.589815	0.838200	0.084158	0.009166	1.004521
V	INTERPOL.	4.248747	31.307175	5.595282	1.871000	0.195638	0.109416	0.998407
	KRIGING	4.275504	36.298322	6.024809	7.906800	1.393187	2.585588	1.004695
	SPLINES	2.961483	37.480801	6.122157	18.416000	3.614879	10.534920	0.695915

NOTES:
=====
[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))
[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N
[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)
SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS
AVR : AVERAGE ABSOLUTE VALUE RESIDUALS
N : NUMBER OF DATA POINTS
[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.19. Summary of the performance of the calibrated flows. Example A.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	-0.004	-0.244	-0.136	0.278
	KRIGING	-184.733	-177.615	-170.771	-999.999
	SPLINES	-72.902	-40.567	-67.387	-999.999
II	INTERPOL.	0.773	0.901	0.307	0.817
	KRIGING	-2.597	-142.343	-16.335	-165.491
	SPLINES	-29.806	-162.665	-68.515	-999.999
III	INTERPOL.	0.661	0.877	0.616	0.917
	KRIGING	1.000	-76.926	-15.565	-148.833
	SPLINES	-52.206	-155.384	-60.444	-999.999
IV	INTERPOL.	0.124	0.975	-10.655	-84.968
	KRIGING	0.999	-81.717	-15.290	-206.090
	SPLINES	-63.255	-142.358	-34.568	-895.254
V	INTERPOL.	0.937	0.978	0.376	0.662
	KRIGING	-999.999	-999.999	-108.782	-999.999
	SPLINES	-999.999	-573.619	-252.898	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.498	0.697	-1.898	-16.459
KRIGING	-237.066	-295.720	-65.349	-504.082
SPLINES	-243.633	-214.918	-96.763	-979.050

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	80.000	80.000	60.000	80.000
KRIGING	40.000	0.000	0.000	0.000
SPLINES	0.000	0.000	0.000	0.000

Table D.20. Summary of the performance of the calibrated flows. Example B.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	0.008	-0.050	0.005	-0.003
	KRIGING	1.000	0.920	-0.154	0.478
	SPLINES	-42.610	-14.722	-26.108	-551.471
II	INTERPOL.	0.866	0.894	0.022	0.306
	KRIGING	0.891	-0.129	-0.404	0.221
	SPLINES	-1.351	-117.440	-19.608	-379.195
III	INTERPOL.	0.994	0.940	0.022	0.081
	KRIGING	0.984	-0.104	0.376	0.655
	SPLINES	-0.861	-158.095	-22.371	-397.106
IV	INTERPOL.	0.923	0.999	0.005	0.157
	KRIGING	0.999	-1.145	-7.038	-64.831
	SPLINES	-7.648	-472.515	-69.540	-999.999
V	INTERPOL.	0.548	0.956	-16.042	-129.818
	KRIGING	-999.999	-999.999	-91.807	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.668	0.748	-3.198	-25.855
KRIGING	-199.225	-200.091	-19.805	-212.695
SPLINES	-210.494	-352.554	-227.525	-665.554

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	100.000	80.000	80.000	60.000
KRIGING	80.000	20.000	20.000	60.000
SPLINES	0.000	0.000	0.000	0.000

Table D.21. Summary of the performance of the calibrated flows. Example C.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	-0.023	0.771	-2.638	-18.641
	KRIGING	-31.332	-29.866	-6.784	-73.242
	SPLINES	-140.271	-148.540	-33.078	-999.999
II	INTERPOL.	-12.538	-77.063	-2.586	-15.337
	KRIGING	-0.931	-278.370	-3.860	-24.126
	SPLINES	-46.695	-533.205	-25.706	-999.999
III	INTERPOL.	0.900	-1.074	-2.870	-28.460
	KRIGING	0.996	-243.327	-6.084	-50.106
	SPLINES	-116.581	-535.163	-23.135	-999.999
IV	INTERPOL.	0.967	1.000	0.798	0.974
	KRIGING	0.991	-6.542	-0.007	0.109
	SPLINES	-5.318	-17.798	-2.484	-7.726
V	INTERPOL.	0.988	0.994	0.275	0.605
	KRIGING	0.342	0.033	0.201	-0.116
	SPLINES	0.091	0.811	-2.654	-11.237

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	-1.941	-15.074	-1.404	-12.172
KRIGING	-5.987	-111.615	-3.307	-29.496
SPLINES	-61.755	-246.779	-17.411	-603.792

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	60.000	60.000	40.000	40.000
KRIGING	60.000	20.000	20.000	20.000
SPLINES	20.000	20.000	0.000	0.000

Table D.22. Summary of the performance of the calibrated flows. Example D.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	-19.250	-999.999	-52.814	-999.999
	KRIGING	-0.563	-999.999	-936.003	-999.999
	SPLINES	0.437	-999.999	-364.938	-999.999
II	INTERPOL.	-999.999	-999.999	-66.950	-999.999
	KRIGING	-999.999	-999.999	-999.999	-999.999
	SPLINES	-999.999	-999.999	-346.077	-999.999
III	INTERPOL.	-999.999	-999.999	-39.519	-745.078
	KRIGING	-999.999	-999.999	-999.999	-999.999
	SPLINES	-999.999	-999.999	-575.257	-999.999
IV	INTERPOL.	-999.999	-999.999	-54.663	-999.999
	KRIGING	-999.999	-999.999	-999.999	-999.999
	SPLINES	-999.999	-999.999	-582.368	-999.999
V	INTERPOL.	-164.766	-999.999	-349.430	-999.999
	KRIGING	-999.999	-999.999	-999.999	-999.999
	SPLINES	-999.999	-999.999	-575.688	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	-636.803	-999.999	-112.675	-949.015
KRIGING	-800.112	-999.999	-987.200	-999.999
SPLINES	-799.912	-999.999	-488.865	-999.999

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.000	0.000	0.000	0.000
KRIGING	0.000	0.000	0.000	0.000
SPLINES	20.000	0.000	0.000	0.000

Table D.23. Summary of the performance of the calibrated flows. Example E.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	0.453	0.268	-1.701	-2.970
	KRIGING	-999.999	-445.618	-271.042	-999.999
	SPLINES	-459.738	-247.806	-395.403	-999.999
II	INTERPOL.	-4.690	0.694	-0.834	-2.373
	KRIGING	-999.999	-371.842	-225.047	-999.999
	SPLINES	-333.037	-426.801	-999.999	-999.999
III	INTERPOL.	1.000	1.000	-1.019	-1.390
	KRIGING	-35.633	-463.270	-999.999	-999.999
	SPLINES	-37.607	-343.325	-971.949	-999.999
IV	INTERPOL.	0.998	0.998	-2.326	-10.028
	KRIGING	-34.361	-537.835	-999.999	-999.999
	SPLINES	-20.644	-999.999	-999.999	-999.999
V	INTERPOL.	0.909	0.965	-8.595	-59.700
	KRIGING	-83.941	-242.848	-834.416	-999.999
	SPLINES	-82.577	-368.592	-919.051	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	-0.266	0.785	-2.895	-15.292
KRIGING	-430.787	-412.283	-666.101	-999.999
SPLINES	-186.721	-477.304	-857.280	-999.999

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	80.000	100.000	0.000	0.000
KRIGING	0.000	0.000	0.000	0.000
SPLINES	0.000	0.000	0.000	0.000

Table D.24. Summary of the performance of the calibrated flows. Example F.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	-0.192	-0.127	-0.091	-2.424
	KRIGING	0.478	-999.999	-135.710	-999.999
	SPLINES	-999.999	-999.999	-999.999	-999.999
II	INTERPOL.	0.958	-35.067	0.037	-0.962
	KRIGING	0.995	-999.999	-178.260	-999.999
	SPLINES	-999.999	-999.999	-310.745	-999.999
III	INTERPOL.	0.968	-33.502	-0.054	-0.681
	KRIGING	-0.804	-999.999	-84.489	-999.999
	SPLINES	-999.999	-999.999	-467.582	-999.999
IV	INTERPOL.	0.980	-39.651	-0.067	0.014
	KRIGING	-0.858	-999.999	-84.659	-999.999
	SPLINES	-999.999	-999.999	-467.876	-999.999
V	INTERPOL.	0.876	-37.067	-3.983	-141.496
	KRIGING	-0.079	-999.999	-87.983	-999.999
	SPLINES	-999.999	-999.999	-481.720	-999.999

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.718	-29.083	-0.832	-29.110
KRIGING	-0.054	-999.999	-114.220	-999.999
SPLINES	-999.999	-999.999	-545.584	-999.999

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	80.000	0.000	20.000	20.000
KRIGING	40.000	0.000	0.000	0.000
SPLINES	0.000	0.000	0.000	0.000

Table D.25. Summary of the comparison between true, initial and calibrated C's. Example A.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	122.890938	596.145498	24.416091	6.221000	2.022688	2.243039	1.003191
I	INTERPOL.	119.379625	453.853417	21.303836	17.267000	5.442625	37.148887	0.974528
	KRIGING	116.812375	452.467591	21.271286	17.267000	11.506250	21.444892	0.953570
	SPLINES	111.619937	418.452325	20.456107	17.267000	11.454188	12.881961	0.911183
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	122.890938	596.145498	24.416091	6.221000	2.022688	2.243039	1.003191
II	INTERPOL.	119.185125	459.350265	21.432458	17.267000	5.537625	36.315976	0.972940
	KRIGING	116.401375	477.925369	21.861504	17.267000	11.809750	15.638503	0.950215
	SPLINES	111.619937	418.452325	20.456107	17.267000	11.454188	12.881961	0.911183
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	122.890938	596.145498	24.416091	6.221000	2.022688	2.243039	1.003191
III	INTERPOL.	119.301563	456.534883	21.366677	17.267000	5.457187	36.825283	0.973890
	KRIGING	116.396250	478.276939	21.869544	17.267000	11.814875	15.567570	0.950173
	SPLINES	111.619937	418.452325	20.456107	17.267000	11.454188	12.881961	0.911183
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	123.736213	702.925795	26.512748	19.671900	6.396437	22.428561	1.010092
IV	INTERPOL.	122.858750	788.387341	28.078236	18.461000	10.110000	29.841839	1.002929
	KRIGING	117.225062	578.875420	24.059830	22.372000	12.627313	36.986530	0.956939
	SPLINES	112.284687	494.513023	22.237649	22.372000	11.053438	32.087600	0.916610
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	122.890938	596.145498	24.416091	6.221000	2.022688	2.243039	1.003191
V	INTERPOL.	124.703875	923.377391	30.387125	18.461000	8.890500	37.744147	1.017991
	KRIGING	116.555125	467.769095	21.627970	17.267000	11.656000	18.185262	0.951470
	SPLINES	111.619937	418.452325	20.456107	17.267000	11.454188	12.881961	0.911183

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.26. Summary of the comparison between true, initial and calibrated C's. Example B.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	122.890938	596.145498	24.416091	6.221000	2.022688	2.243039	1.003191
I	INTERPOL.	119.657187	424.691198	20.608037	17.267000	5.532688	31.803443	0.976793
	KRIGING	122.622563	675.892952	25.997941	18.461000	12.730688	14.750940	1.001001
	SPLINES	113.200125	465.662323	21.579210	17.267000	11.631000	13.326076	0.924083
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	122.890938	596.145498	24.416091	6.221000	2.022688	2.243039	1.003191
II	INTERPOL.	118.330875	418.961833	20.468557	17.267000	6.111125	36.077826	0.965966
	KRIGING	122.622563	675.892952	25.997941	18.461000	12.730688	14.750940	1.001001
	SPLINES	113.200125	465.662323	21.579210	17.267000	11.631000	13.326076	0.924083
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	122.890938	596.145498	24.416091	6.221000	2.022688	2.243039	1.003191
III	INTERPOL.	118.872063	436.324698	20.888387	17.267000	5.567937	31.063965	0.970384
	KRIGING	122.622563	675.892952	25.997941	18.461000	12.730688	14.750940	1.001001
	SPLINES	113.200125	465.662323	21.579210	17.267000	11.631000	13.326076	0.924083
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	123.736213	702.925795	26.512748	19.671900	6.396437	22.428561	1.010092
IV	INTERPOL.	118.616813	360.368703	18.983380	17.267000	8.423312	30.098341	0.968301
	KRIGING	127.067375	935.437515	30.584923	32.639000	13.667000	65.855510	1.037285
	SPLINES	113.903062	551.062498	23.474720	22.372000	11.612562	34.150292	0.929821
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	122.890938	596.145498	24.416091	6.221000	2.022688	2.243039	1.003191
V	INTERPOL.	120.268812	574.629776	23.971437	17.843000	10.084187	26.602814	0.981786
	KRIGING	124.253625	700.411627	26.465291	18.461000	12.791250	14.790742	1.014315
	SPLINES	114.718687	490.040616	22.136861	17.267000	11.816563	13.506217	0.936479

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.{ABS(TRUE-ESTIMATED)}

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.27. Summary of the comparison between true, initial and calibrated C's. Example C.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	147.219062	860.341005	29.331570	34.684000	24.719062	40.564291	1.201788
I	INTERPOL.	136.497437	980.301513	31.309767	46.620000	13.997438	158.885550	1.114265
	KRIGING	137.472625	734.690323	27.105172	35.814000	14.972625	112.078918	1.122226
	SPLINES	132.877187	663.085449	25.750446	15.518000	10.377187	10.626910	1.084712
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	147.219062	860.341005	29.331570	34.684000	24.719062	40.564291	1.201788
II	INTERPOL.	134.497375	874.595216	29.573556	46.506000	11.997375	103.276303	1.097938
	KRIGING	137.472688	734.686817	27.105107	35.930000	14.972688	112.107661	1.122226
	SPLINES	132.877187	663.085449	25.750446	15.518000	10.377187	10.626910	1.084712
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	147.219062	860.341005	29.331570	34.684000	24.719062	40.564291	1.201788
III	INTERPOL.	136.498625	980.540824	31.313588	46.634000	13.998625	158.953740	1.114274
	KRIGING	137.380813	730.457963	27.026986	35.959000	14.880813	108.113889	1.121476
	SPLINES	132.877187	663.085449	25.750446	15.518000	10.377187	10.626910	1.084712
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	148.4835001012.213738		31.815307	45.606000	25.983500	140.251862	1.212110
IV	INTERPOL.	137.3849381041.147562		32.266818	47.007000	14.884938	158.527745	1.121510
	KRIGING	138.228250	799.584177	28.276920	38.470000	15.728250	163.225904	1.128394
	SPLINES	134.533500	729.374341	27.006931	30.045000	12.033500	69.358936	1.098233
	TRUE	122.500000	566.666667	23.804761				
	INITIAL	147.219062	860.341005	29.331570	34.684000	24.719062	40.564291	1.201788
V	INTERPOL.	138.0623121208.461234		34.762929	54.153000	15.562312	217.351680	1.127039
	KRIGING	138.420125	757.052799	27.514592	41.867000	15.920125	151.195141	1.129960
	SPLINES	133.010000	652.365222	25.541441	15.518000	10.510000	10.524881	1.085796

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.28. Summary of the comparison between true, initial and calibrated C's. Example D.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.958833	7.499320	2.738489	8.522000	2.237533	2.480356	0.999706
I	INTERPOL.	139.192178	86.478615	9.299388	18.792000	7.526600	30.338138	0.994230
	KRIGING	136.527167	181.567801	13.474710	20.412000	13.389444	13.492355	0.975194
	SPLINES	137.622156	190.145060	13.789310	20.412000	13.640400	8.779885	0.983015
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.958833	7.499320	2.738489	8.522000	2.237533	2.480356	0.999706
II	INTERPOL.	138.995589	95.519798	9.773423	20.379000	7.970922	32.827118	0.992826
	KRIGING	136.333611	184.074020	13.567388	20.412000	13.601756	11.615097	0.973812
	SPLINES	136.728044	180.292340	13.427298	20.412000	13.450600	9.179766	0.976629
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.958833	7.499320	2.738489	8.522000	2.237533	2.480356	0.999706
III	INTERPOL.	139.758500	75.524815	8.690501	18.792000	6.708589	30.497248	0.998275
	KRIGING	136.327722	184.567057	13.585546	20.412000	13.657544	10.616752	0.973769
	SPLINES	135.940489	182.494425	13.509050	21.670000	13.761400	8.680518	0.971003
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.941799	14.998452	3.872783	12.052300	3.164327	4.960670	0.999584
IV	INTERPOL.	141.093283	139.496700	11.810872	20.018000	10.585194	28.183798	1.007809
	KRIGING	136.465333	193.623291	13.914859	23.067000	13.705056	17.406331	0.974752
	SPLINES	135.895606	189.202131	13.755077	24.847000	13.765272	15.688661	0.970683
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.958833	7.499320	2.738489	8.522000	2.237533	2.480356	0.999706
V	INTERPOL.	140.077217	158.886754	12.605029	21.670000	11.411817	28.092593	1.000552
	KRIGING	136.113194	184.420192	13.580140	20.412000	13.702972	10.852023	0.972237
	SPLINES	136.254539	183.084631	13.530877	20.412000	13.649317	9.902152	0.973247

NOTES:

=====

[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.(ABS(TRUE-ESTIMATED))

[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N

[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = { SSAVR - N * AVR**2 } / (N-2)

SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS

AVR : AVERAGE ABSOLUTE VALUE RESIDUALS

N : NUMBER OF DATA POINTS

[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.29. Summary of the comparison between true, initial and calibrated C's. Example E.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.958833	7.499320	2.738489	8.522000	2.237533	2.480356	0.999706
I	INTERPOL.	139.959117	85.458171	9.244359	19.986000	7.114017	34.762087	0.999708
	KRIGING	133.260300	167.756177	12.952072	21.670000	14.315844	7.386423	0.951859
	SPLINES	134.701350	166.904531	12.919154	20.412000	13.577439	9.815186	0.962153
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.958833	7.499320	2.738489	8.522000	2.237533	2.480356	0.999706
II	INTERPOL.	139.964678	83.243194	9.123771	19.986000	6.903667	35.515988	0.999748
	KRIGING	133.278372	161.905021	12.724190	21.670000	14.057872	8.658260	0.951988
	SPLINES	134.189333	163.253766	12.777080	20.412000	13.731356	7.645472	0.958495
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.958833	7.499320	2.738489	8.522000	2.237533	2.480356	0.999706
III	INTERPOL.	140.051139	84.346600	9.184040	19.986000	7.002450	35.237848	1.000365
	KRIGING	132.354694	147.510161	12.145376	21.670000	14.047183	7.905840	0.945391
	SPLINES	134.730900	167.586697	12.945528	21.670000	13.562789	10.587471	0.962364
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.941799	14.998452	3.872783	12.052300	3.164327	4.960670	0.999584
IV	INTERPOL.	139.628606	104.934688	10.243763	19.986000	8.321639	35.635932	0.997347
	KRIGING	132.401378	156.283385	12.501335	24.847000	14.074833	15.222409	0.945724
	SPLINES	134.281450	171.450543	13.093912	24.847000	13.686539	16.056918	0.959153
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.958833	7.499320	2.738489	8.522000	2.237533	2.480356	0.999706
V	INTERPOL.	137.235400	118.755303	10.897491	20.298000	9.453600	36.776641	0.980253
	KRIGING	132.666867	152.716219	12.357840	21.670000	14.022656	9.108982	0.947620
	SPLINES	134.993917	171.974287	13.113897	21.670000	13.682317	8.973663	0.964242

NOTES:
=====
[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.{ABS(TRUE-ESTIMATED)}
[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N
[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)
SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS
AVR : AVERAGE ABSOLUTE VALUE RESIDUALS
N : NUMBER OF DATA POINTS
[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.30. Summary of the comparison between true, initial and calibrated C's. Example F.

CASE	PROCEDURE	AVERAGE	VARIANCE	STAN.DEV.	MAXIMUM RES.[1]	AVERAGE RES.[2]	VARIANCE RES. [3]	RATIO AVERAGES EST/TRUE [4]
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.969736	7.491670	2.737092	8.522000	2.222908	2.537376	0.999784
I	INTERPOL.	139.866701	52.512202	7.246530	20.412000	5.242322	25.033985	0.999048
	KRIGING	132.113661	118.103721	10.867554	21.670000	12.927937	12.632965	0.943669
	SPLINES	133.594437	165.135903	12.850522	21.670000	14.063609	7.519413	0.954246
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.969736	7.491670	2.737092	8.522000	2.222908	2.537376	0.999784
II	INTERPOL.	139.530891	54.774347	7.400969	20.412000	5.397684	25.841654	0.996849
	KRIGING	132.308672	119.922215	10.950900	21.670000	12.835730	13.792095	0.945062
	SPLINES	133.784931	162.353426	12.741798	21.670000	13.870092	7.757154	0.955607
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.969736	7.491670	2.737092	8.522000	2.222908	2.537376	0.999784
III	INTERPOL.	139.469500	50.787031	7.126502	20.412000	5.150810	24.527662	0.996211
	KRIGING	133.351609	141.238452	11.884379	21.670000	13.129943	12.374684	0.952511
	SPLINES	134.002477	167.204936	12.930775	21.670000	13.913879	8.718445	0.957161
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.957221	14.983226	3.870817	12.052300	3.143651	5.074737	0.999694
IV	INTERPOL.	139.767529	59.377350	7.705670	19.255000	5.887805	24.707900	0.998339
	KRIGING	133.245954	144.269356	12.011218	24.847000	13.060161	18.704550	0.951757
	SPLINES	134.011236	176.107339	13.270544	24.847000	14.000236	15.127812	0.957223
	TRUE	140.000000	0.000000	0.000000				
	INITIAL	139.969736	7.491670	2.737092	8.522000	2.222908	2.537376	0.999784
V	INTERPOL.	140.410747	94.910647	9.742210	20.412000	7.831782	33.583106	1.002934
	KRIGING	133.654310	140.366317	11.847629	21.670000	12.962885	11.928107	0.954674
	SPLINES	134.123264	171.372956	13.090949	21.670000	14.042563	7.820383	0.958023

NOTES:
=====
[1] : MAXIMUM ABSOLUTE VALUE RESIDUAL = MAX.{ABS(TRUE-ESTIMATED)}
[2] : AVERAGE OF THE RESIDUALS=(SUMMATION OF ABSOLUTE VALUES OF THE RESIDUALS)/N
[3] : VARIANCE OF THE ABSOLUTE VALUE OF THE RESIDUALS = (SSAVR - N * AVR**2) / (N-2)
SSAVR : SUMMATION SQUARES ABSOLUTE VALUES RESIDUALS
AVR : AVERAGE ABSOLUTE VALUE RESIDUALS
N : NUMBER OF DATA POINTS
[4] : RATIO AVERAGE ESTIMATED VALUES/AVERAGE TRUE VALUES

Table D.31. Summary of the performance of the calibrated C's. Example A.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	-62.709	-13.645	-6.704	-273.295
	KRIGING	-210.664	-14.007	-6.704	-90.406
	SPLINES	-773.546	-24.279	-6.704	-31.983
II	INTERPOL.	-70.898	-12.253	-6.704	-261.133
	KRIGING	-242.359	-8.062	-6.704	-47.609
	SPLINES	-773.546	-24.279	-6.704	-31.983
III	INTERPOL.	-65.936	-12.957	-6.704	-268.537
	KRIGING	-242.769	-7.990	-6.704	-47.169
	SPLINES	-773.546	-24.279	-6.704	-31.983
IV	INTERPOL.	0.916	-1.648	0.119	-0.770
	KRIGING	-17.207	0.992	-0.293	-1.719
	SPLINES	-67.284	0.720	-0.293	-1.047
V	INTERPOL.	-30.780	-145.424	-7.806	-282.156
	KRIGING	-230.244	-10.255	-6.704	-64.730
	SPLINES	-773.546	-24.279	-6.704	-31.983

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	-45.881	-37.185	-5.560	-217.178
KRIGING	-188.649	-7.865	-5.422	-50.327
SPLINES	-632.293	-19.279	-5.422	-25.796

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	20.000	0.000	20.000	0.000
KRIGING	0.000	20.000	0.000	0.000
SPLINES	0.000	20.000	0.000	0.000

Table D.32. Summary of the performance of the calibrated C's. Example B.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	-51.879	-22.196	-6.704	-200.036
	KRIGING	0.902	-12.729	-7.806	-42.248
	SPLINES	-564.898	-10.740	-6.704	-34.296
II	INTERPOL.	-112.730	-24.106	-6.704	-257.706
	KRIGING	0.902	-12.729	-7.806	-42.248
	SPLINES	-564.898	-10.740	-6.704	-34.296
III	INTERPOL.	-85.120	-18.550	-6.704	-190.796
	KRIGING	0.902	-12.729	-7.806	-42.248
	SPLINES	-564.898	-10.740	-6.704	-34.296
IV	INTERPOL.	-8.867	-1.292	0.230	-0.801
	KRIGING	-12.650	-6.325	-1.753	-7.621
	SPLINES	-47.362	0.987	-0.293	-1.318
V	INTERPOL.	-31.573	0.927	-7.227	-139.664
	KRIGING	-19.121	-19.584	-7.806	-42.482
	SPLINES	-395.178	-5.757	-6.704	-35.257

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	-58.034	-13.043	-5.422	-157.801
KRIGING	-5.813	-12.819	-6.596	-35.369
SPLINES	-427.447	-7.398	-5.422	-27.893

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.000	20.000	20.000	0.000
KRIGING	60.000	0.000	0.000	0.000
SPLINES	0.000	20.000	0.000	0.000

Table D.33. Summary of the performance of the calibrated C's. Example C.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	0.679	-0.984	-0.807	-14.342
	KRIGING	0.633	0.673	-0.066	-6.634
	SPLINES	0.824	0.892	0.800	0.931
II	INTERPOL.	0.764	-0.099	-0.798	-5.482
	KRIGING	0.633	0.673	-0.073	-6.638
	SPLINES	0.824	0.892	0.800	0.931
III	INTERPOL.	0.679	-0.986	-0.808	-14.355
	KRIGING	0.638	0.689	-0.075	-6.104
	SPLINES	0.824	0.892	0.800	0.931
IV	INTERPOL.	0.672	-0.134	-0.062	-0.278
	KRIGING	0.634	0.727	0.288	-0.354
	SPLINES	0.786	0.867	0.566	0.755
V	INTERPOL.	0.604	-3.776	-1.438	-27.710
	KRIGING	0.585	0.580	-0.457	-12.893
	SPLINES	0.819	0.915	0.800	0.933

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.680	-1.196	-0.782	-12.433
KRIGING	0.625	0.668	-0.077	-6.525
SPLINES	0.815	0.892	0.753	0.896

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	100.000	0.000	0.000	0.000
KRIGING	100.000	100.000	20.000	0.000
SPLINES	100.000	100.000	100.000	100.000

Table D.34. Summary of the performance of the calibrated C's. Example D.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	-384.064	-131.976	-3.863	-148.606
	KRIGING	-999.999	-585.184	-4.737	-28.590
	SPLINES	-999.999	-641.875	-4.737	-11.530
II	INTERPOL.	-594.284	-161.234	-4.719	-174.161
	KRIGING	-999.999	-601.478	-4.737	-20.929
	SPLINES	-999.999	-576.977	-4.737	-12.697
III	INTERPOL.	-33.414	-100.423	-3.863	-150.180
	KRIGING	-999.999	-604.710	-4.737	-17.321
	SPLINES	-999.999	-591.182	-5.466	-11.248
IV	INTERPOL.	-351.862	-85.504	-1.759	-31.279
	KRIGING	-999.999	-165.657	-2.663	-11.312
	SPLINES	-999.999	-158.133	-3.250	-9.002
V	INTERPOL.	-2.518	-447.881	-5.466	-127.279
	KRIGING	-999.999	-603.746	-4.737	-18.142
	SPLINES	-999.999	-595.019	-4.737	-14.938

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	-273.228	-185.404	-3.934	-126.301
KRIGING	-999.999	-512.155	-4.322	-19.259
SPLINES	-999.999	-512.637	-4.585	-11.883

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.000	0.000	0.000	0.000
KRIGING	0.000	0.000	0.000	0.000
SPLINES	0.000	0.000	0.000	0.000

Table D.35. Summary of the performance of the calibrated C's. Example E.

CASE	PROCEDURE	PERFORMANCE INDEX: AVERAGE RESID.	PERFORMANCE INDEX: VARIANCE RESID.	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	0.014	-128.856	-4.500	-195.419
	KRIGING	-999.999	-499.395	-5.466	-7.868
	SPLINES	-999.999	-494.328	-4.737	-14.659
II	INTERPOL.	0.264	-122.212	-4.500	-204.031
	KRIGING	-999.999	-465.098	-5.466	-11.185
	SPLINES	-999.999	-472.896	-4.737	-8.501
III	INTERPOL.	-0.543	-125.500	-4.500	-200.832
	KRIGING	-999.999	-385.901	-5.466	-9.159
	SPLINES	-999.999	-498.385	-5.466	-17.220
IV	INTERPOL.	-39.720	-47.949	-1.750	-50.605
	KRIGING	-999.999	-107.576	-3.250	-8.416
	SPLINES	-999.999	-129.673	-3.250	-9.477
V	INTERPOL.	-999.999	-249.762	-4.673	-218.845
	KRIGING	-999.999	-413.693	-5.466	-12.487
	SPLINES	-999.999	-524.876	-5.466	-12.089

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX: AVERAGE	AVERAGE INDEX: VARIANCE	MAX.RES	VAR.RES
INTERPOL.	-207.997	-134.856	-3.985	-173.947
KRIGING	-999.999	-374.333	-5.023	-9.823
SPLINES	-999.999	-424.031	-4.731	-12.389

PROCEDURE	FREQUENCY OF IMPROVEMENT (%): AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	40.000	0.000	0.000	0.000
KRIGING	0.000	0.000	0.000	0.000
SPLINES	0.000	0.000	0.000	0.000

Table D.36. Summary of the performance of the calibrated C's. Example F.

CASE	PROCEDURE	PERFORMANCE INDEX			
		AVERAGE	VARIANCE	MAXIMUM RESID.	VARIANCE RESID.
I	INTERPOL.	-18.400	-48.132	-4.737	-96.340
	KRIGING	-999.999	-247.525	-5.466	-23.788
	SPLINES	-999.999	-484.876	-5.466	-7.782
II	INTERPOL.	-239.267	-52.456	-4.737	-102.722
	KRIGING	-999.999	-255.237	-5.466	-28.545
	SPLINES	-999.999	-468.641	-5.466	-8.346
III	INTERPOL.	-306.269	-44.857	-4.737	-92.442
	KRIGING	-999.999	-354.426	-5.466	-22.785
	SPLINES	-999.999	-497.128	-5.466	-10.806
IV	INTERPOL.	-28.531	-14.705	-1.552	-22.705
	KRIGING	-999.999	-91.712	-3.250	-12.585
	SPLINES	-999.999	-137.148	-3.250	-7.886
V	INTERPOL.	-183.203	-159.499	-4.737	-174.175
	KRIGING	-999.999	-350.050	-5.466	-21.099
	SPLINES	-999.999	-522.272	-5.466	-8.499

NOTE: A VALUE OF -999.999 INDICATES THAT THE INDEX IS ACTUALLY LESS THAN OR EQUAL TO -999.999

PROCEDURE	AVERAGE INDEX			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	-155.134	-63.950	-4.100	-97.677
KRIGING	-999.999	-259.790	-5.023	-21.760
SPLINES	-999.999	-422.013	-5.023	-8.664

PROCEDURE	FREQUENCY OF IMPROVEMENT (%)			
	AVERAGE	VARIANCE	MAX.RES	VAR.RES
INTERPOL.	0.000	0.000	0.000	0.000
KRIGING	0.000	0.000	0.000	0.000
SPLINES	0.000	0.000	0.000	0.000