APPENDIX B

LDL^t Factorization and Constrained Linear Systems

Although the standard variational approach to magnetostatics led to a standard linear system of the form $Ax = b$, with A symmetric regular and positive definite, we had to realize that discrete models do not automatically come in this form, but rather constitute what we called "constrained linear systems". So there is a gap, however small, between equations as they emerge from the modelling and numerical methods as proposed by textbooks and software packages. Whether this gap is negligible or not, and how to bridge it in the latter case, are important issues. This appendix is an approach to this problem from the side of *direct* methods, based on Gaussian factorization, such as LDL^t. Some facts about the LDL^t method and its programming are recalled, and we find the adaptation to constrained linear systems feasible, if not totally straightforward.

B.1 NONNEGATIVE DEFINITE MATRICES

A standard result about the factorization of matrices is: A *positive definite* matrix A (see Def. B.1 below), not necessarily symmetric, *has an* LDM^t *decomposition*, i.e., one can express A as the product LDM^t, where D is diagonal, with strictly positive entries, and L and M are unit lower triangular (i.e., with all diagonal entries equal to 1). See for instance [GL], p. 86, for a proof. A corollary is Gauss's LU decomposition: $A = LU$, obtained by setting $U = DM^t$.

There is a need, however, for an analogous result that would hold under the weaker assumption of *semi-positive* or *nonnegative* definiteness (Def. B.2 below): Several times, and notoriously in the case of the rot–rot equation, we found the system's matrix nonnegative definite, but not regular, because of the non-uniqueness of potentials representing the same

field. Our matrices were also symmetric, so we shall make this assumption, although this is not strictly necessary. Then, $M = L$. As we shall see, the LDL^t factorization is an effective tool for the treatment of constrained linear systems of this category. The reason for this lies in a mathematical result, which we shall prove: *Non-negative definite symmetric matrices are the LDL^t-factorizable matrices with* $D \geq 0$ (i.e., all entries of D nonnegative).

As in the main text, we denote by V_n the real vector space of dimension n, but the boldface convention, pointless here, is shunned, and the scalar product is denoted either by (x, y) or by $x^t y$, where the superscript t stands for "transpose". Observe that xy^t is an $n \times \nu$ matrix, called the *dyadic product* of x by y.

For any $n \times n$ matrix A, we set $\ker(A) = \{x \in V_n : Ax = 0\}$ and $\mathrm{cod}(A) = \{Ax : x \in V_n\}$, i.e., the image of V_n under the action of A, also called the range of A. Let us recall that $\ker(A)$ and $\mathrm{cod}(A^t)$ are mutually orthogonal complementary subspaces of V_n, a fact which is expressed as

$$V_n = \ker(A) \oplus \mathrm{cod}(A^t) \equiv \ker(A^t) \oplus \mathrm{cod}(A). \tag{1}$$

Definition B.1. *An* $n \times n$ *matrix* A *(with real entries) is said to be* positive definite *if*

$$x^t A x > 0 \quad \forall x \in V_n,\ x \neq 0. \tag{2}$$

Definition B.2. *Matrix* A *is* nonnegative definite *if*

$$x^t A x \geq 0 \quad \forall x \in V_n. \tag{3}$$

A positive definite matrix must be regular. Beware, a regular non-definite matrix may fail to satisfy $x^t A x \neq 0$: for instance, matrix $I = \{\{0, 1\}, \{-1, 0\}\}$ is regular, but also skew-symmetric, and hence $x^t I x = 0$ for all x. However,

Proposition B.1. *Among* symmetric *matrices, positive definite matrices are the* regular *nonnegative definite matrices.*

Proof. It's the same proof we did in Section 3.1. Assume (3), and suppose $x^t A x = 0$ for some $x \neq 0$. Then A cannot be regular, because for all $y \in V_n$ and all $\lambda \in \mathbb{R}$,

$$0 \leq (x + y)^t A (x + y) = 2\lambda\, y^t Ax + \lambda^2\, y^t A\, y,$$

hence $y^t Ax = 0\ \forall\, y$ (divide by λ, and let it go to 0), and hence $Ax = 0$. Thus, (3) and regularity, taken together, imply (2) in the case of symmetric matrices. ◊

From this point on, let us restrict ourselves to symmetric matrices. Let us recall the following:

Definition B.3. *An* $n \times n$ *matrix* A *is* Cholesky-factorizable *if there exists an upper triangular* $n \times n$ *matrix* S *such that* $A = S^t S$.

Such a matrix is symmetric and, obviously, nonnegative definite. Conversely,

Proposition B.2. *Nonnegative definite, symmetric matrices are Cholesky-factorizable.*

Proof. The proof is by recurrence on the order n. Let us write A, by rows of blocks, as $A = \{\{c, b^t\}, \{b, C\}\}$, where C is of order $n - 1$, and look for its factorization in the form

$$A = \begin{vmatrix} c & b^t \\ b & C \end{vmatrix} = \begin{vmatrix} a & 0 \\ d & T \end{vmatrix} \begin{vmatrix} a & d^t \\ 0 & T^t \end{vmatrix} = SS^t,$$

where T is lower triangular of order $n - 1$. If (3) holds, then, for any $(n - 1)$-vector y and any scalar z,

$$y^t C y + 2 y^t b z + c z^2 \geq 0. \tag{4}$$

If $c = 0$, this shows that $y^t b = 0$ for all $y \in V_{n-1}$, hence $b = 0$. Then $a = 0$ and $d = 0$, and since C is symmetric and nonnegative definite (take $x = \{0, y\}$ in (3)), we do have $C = TT^t$ by the recurrence hypothesis. Therefore, $S = \{\{0, 0\}, \{0, T\}\}$ (by rows of blocks). If $c > 0$, set $a = \sqrt{c}$, which forces $d = b/a$, and requires $C = c^{-1} b b^t + TT^t$, so all we have to do is show that $C - c^{-1} b b^t$ is nonnegative definite. This again results from (4), by setting $z = c^{-1} b^t y$. ◊

Remark B.1. A priori, $\ker(A)$ contains $\ker(S^t)$. But if $A x = 0$, then $S^t x$ belongs to $\ker(S)$, hence $S^t x \perp \mathrm{cod}(S^t)$ after (1), which implies $S^t x \perp S^t x$, and therefore $S^t x = 0$. So $\ker(S^t) = \ker(A)$. ◊

Remark B.2. Diagonal terms of S are all nonnegative, with the foregoing choice of a. Note that whenever one of them vanishes, the whole column below it must vanish, too. ◊

Remark B.3. The proof works just as well if entries are complex (note that A is *not* Hermitian, then), provided $\mathrm{Re}\{x^t A x^*] \geq 0$ for all complex vectors x. *Bisymmetric* matrices (matrices such that both real and imaginary parts are symmetric) are thus LDL^t-factorizable under this hypothesis. This is relevant to the eddy-current problem of Chapter 8. ◊

Now what about the LDL^t factorization? Thanks to Remark B.2, we obtain it by the following sequence of assignments, where s_i, ℓ_i, and 1_i

denote the ith column of S, L, and the unit matrix respectively, p is real, and the ith component of some vector v is v^i:

forall $i \in [1, n]$ **do**
| $p := s_i^i$; $d^i := p * p$;
| $\ell_i := \textbf{if } p > 0 \textbf{ then } s_i / p \textbf{ else } 1_i$

This yields a diagonal matrix D, with nonnegative entries d^i, and a unit lower triangular matrix L. Clearly, $A = LDL^t$.

Remark B.4. A straightforward adaptation of the proof would lead us to the result that *nonnegative definite matrices are the* LDM^t*-factorizable matrices.* ◊

One may then solve $Ax = b$, provided b is in the range of A. First, compute $y = L^{-1}b$, then execute the code

forall $i \in [1, n]$ **do**
| $z^i := \textbf{if } d^i \neq 0 \textbf{ then } y^i / d^i \textbf{ else } 0$,

and finally, solve $L^t x = z$. If b Œ cod(A), one *must* have $y^I = 0$ if $d^i = 0$. The choice $z^i = 0$ in such cases is arbitrary: It selects *one* of the solutions of $Ax = b$, and thus constitutes a gauging procedure. By referring to p. 182, one will see how this applies to the rot–rot equation.

Let us now face the question of how to turn these results into a practical algorithm. The problem is, of course, imperfect arithmetic: In spite of the theoretical proof that successive "pivots", i.e., the values of p, will all be nonnegative, there is no guarantee that small negative values or (perhaps worse, because the trouble is harder to spot and to cure) very small positive but non-zero values, will not appear.

B.2 A DIGRESSION ABOUT PROGRAMMING

The question belongs to the immense realm of program correctness in presence of floating-point computations [CC]. Some notions on program construction will help in the discussion.

Let's adhere to the discipline of object-oriented programming [Me]. We deal with abstract data types called *INTEGER, REAL, VECTOR, MATRIX*, etc. (a construct such as m : *MATRIX*, for instance, means that m is a program object of type *MATRIX)* and with operations such as

$$order : MATRIX \rightarrow INTEGER,$$
$$column : MATRIX \times INTEGER \rightarrow VECTOR,$$
$$row : MATRIX \times INTEGER \rightarrow VECTOR,$$

$$component : \ VECTOR \times INTEGER \rightarrow REAL,$$
$$length : \ VECTOR \rightarrow INTEGER ,$$

and so forth. The idea is to program in terms of such operations exclusively. (Their practical *implementation*, of course, may require operations of lower level, those that are available in the target programming language.)

The formal definition of the universe of types and operations we need is quite a large job, and I don't attempt it. Let us just agree upon a few notational conventions: *component*(v, i) is abbreviated as v^i, *length*(v) as $|v|$, *column*(a, j) as a_j, *row*(a, i) as a^i, etc. This way, a^i_j refers to the entry on row i and column j of matrix a. We also have the ordinary multiplication $*$, which can be considered as acting either on a *SCALAR* and a *VECTOR*, or on a pair of *VECTORs*, this way:

$$* : \ SCALAR \times VECTOR \rightarrow VECTOR,$$
$$* : \ VECTOR \times VECTOR \rightarrow VECTOR,$$

and is defined as one may guess: $\lambda * v$ is the vector of components $\lambda * v^i$, that is, formally, $(\lambda * v)^i = \lambda \ v^i$, and for two vectors u and v, $(u * v)^i = u^i \ v^i$.

Just for practice (and also in order to introduce without too much fuss some syntactical conventions about programs), let us code a matrix-vector multiplication within this universe of types:

```
program mat–mul–vec (in a: MATRIX, x : VECTOR {order(a)
|       = length(x)}, out y : VECTOR )
|       local n: INTEGER, n := length(x) ;
|       for j := 1 to n do
|       |       y := y + x^j * a_j
```

Note the *assertion* between curly brackets (the program is not supposed to work if this input assertion is not satisfied). Mere *comments* also come within such brackets.

Once *mat–mul–vec* has thus been written, one may denote by $a * x$ the returned vector y. This overloading of $*$ is harmless and can be reiterated after the eventual construction of other similar programs like *mat–mul–mat*, etc. Along with other obvious operations, like *diag* : $VECTOR \rightarrow MATRIX$, *transp* : $MATRIX \rightarrow MATRIX$, etc., all these operations contribute to the step-by-step construction of an *algebra*, i.e., a consistent and organized universe in which to program [Ba].

Actually, the word "algebra" has connotations which suggest a little more. An algebra of types should be "complete", meaning that when the inverse of some operation can be defined, it is included in the algebra. (Stating this formally is difficult, and I shall not attempt it.) For instance, if *mat–mul–vec* is there to allow multiplication of a matrix a by a vector x, yielding vector y, there should also be something to get x from a and y, say,

$$solve(\textbf{in}\ a: MATRIX,\ y: VECTOR, \textbf{out}\ x: VECTOR\ \{y = a * x\}),$$

for which the appropriate syntax[1] might be $x := a \backslash y$. As one knows, this is not a primitive operation, and it requires stepping stones like triangular solvers and (for instance) the LDL^t factorization. Let us therefore return to this.

B.3 THE LDL^t FACTORIZATION

Let us introduce the *outer product* (or *dyadic product)* of vectors,

$$\times : VECTOR \times VECTOR \to MATRIX,$$

defined by

$$(u \times v)^i_j = u^i v^j.$$

We are looking for a vector d and a lower triangular matrix ℓ such that $\ell^i_i = 1$ and $\ell * diag(d) * transp(\ell) = a$. This specification can be rewritten as

$$\Sigma_{j = 1, \ldots, n} \ell_j \times (d^j * \ell_j) = a,$$

which immediately suggests an algorithm, on the model of the proof of Prop. B.2:

```
program LDLT(in a : MATRIX, out ℓ : MATRIX,
|       d : VECTOR)   {a is nonnegative definite}
|       local c : MATRIX ; c := a ;
|       for j := 1 to order(a) do
|       |       d^j := c^j_j ; ℓ_j := c_j ;
```

[1]This is the syntax of MATLAB [MW]. Obviously, a package such as MATLAB is the implementation of an algebra, in the above sense, and its writing has required at some stage the kind of abstract programming suggested here.

```
|    |    if d^j > 0 then {if d^j = 0 then c_j = 0}
|    |    |    ℓ_j := ℓ_j / d^j ;
|    |    |    c := c − ℓ_j × (d^j ∗ ℓ_j)  {c_j = 0}
|    |    else ℓ^j_j := 1  {ℓ_j = 1_j} ;
|    |    {c_j = 0}
|    |{c = 0 ; a = ℓ ∗ diag(d) ∗ transp(ℓ)}
```

This is the way it works in perfect arithmetic: The crucial assertion **if** $d^j = 0$ **then** $c_j = 0$ is a consequence of the hypothesis on nonnegative definiteness. (Check this point before reading on, if necessary, by rferring to the proof of Prop. B.2.) Note how column j of ℓ happens to be equal to 1_j when the jth pivot is null.

Now, still with perfect arithmetic, we can do this exactly in the same way for *any* entry a, provided the assertions, now not automatically true, are *enforced* when necessary. Hence the following program, which does the same thing as the previous one when a is nonnegative definite, but still does something when this precondition is not satisfied:

```
program LDLT(in a : MATRIX, out ℓ : MATRIX, d : VECTOR)
|    local c : MATRIX ; c := a ;
|    for j := 1 to order(a) do
|    |    d^j := c^j_j ; ℓ_j := c_j ;
|    |    if d^j > 0 then
|    |    |    ℓ_j := ℓ_j / d^j ;
|    |    |    c := c − ℓ_j × (d^j ∗ ℓ_j)
|    |    else ℓ_j := 1_j
```

But the question now is: Could this algorithm fail to be *stable*? Since we may have to divide by arbitrary small pivots d^j, should we fear uncontrolled growth of some terms, and eventual overflow?

The answer seems to be *no*, provided the standard precaution is taken of implementing the test (**if** $d^j > 0$) as **if** $1. + d^j > 1$. This way, the smallest possible pivot will be the machine–epsilon ε, i.e., the number such that $1 + \varepsilon$ is the machine number next to 1 in the (finite) system of numbers the machine offers as an approximation to the ideal *REAL* type. Since the algorithm is a variant of Cholesky, the classical error analysis by Wilkinson should be relevant, and give similar results (cf. [GL], p. 89). Giving a formal proof of this, however, looks like a tough challenge.

In a further attempt to extend the scope of this program, one may replace the clause **if** $d^j > 0$ by **if** $d^j \neq 0$, and **else** $\ell_j := 1$ by a loop exit. What we get then is a program that, *when it doesn't encounter a null pivot*, returns an LDL^t factorization with terms of both signs in D. Then $A = LDL^t$ is regular. Such a program may be a useful tool,[2] but be aware that regularity of A is no guarantee that it will work to the end without falling on a zero pivot: A simple counter-example is given by A = {{0, 1}, {1, 0}} (by rows).

The reader is invited to complete the coding of *solve* (p. 324) by writing out the triangular solvers, and the division by D. Note how this object-oriented style automatically provides "vectorized" programs [Bo].

B.4 APPLICATION TO CONSTRAINED LINEAR SYSTEMS

If working with potentials leads to nonnegative definite system matrices, working with fields directly generates constrained linear systems, as we have seen. Actually, such systems are rather the rule than the exception in numerical modelling. Let us recall the paradigm: Given a symmetric, nonnegative definite matrix A of order N, an N-vector b, a rectangular matrix B, and a vector c of same height, find x such that $(Ax, x) - 2(b, x)$ be minimized over the affine subspace $\{x : Bx = c\}$. By introducing a Lagrange multiplier λ, this problem is transformed into a linear system of the form

$$\begin{vmatrix} A & B^t \\ B & 0 \end{vmatrix} \begin{vmatrix} x \\ \lambda \end{vmatrix} = \begin{vmatrix} b \\ c \end{vmatrix}. \tag{5}$$

If $\ker(A) \cap \ker(B) = \{0\}$, which we assume, x is unique. There is no loss of generality if we also assume that B is surjective (i.e., $\ker(B^t) = 0$), in which case λ is unique too. The large block-matrix at the left-hand side of (5), M say, is thus regular,even when A is not.

However, standard off-the-shelf packages will not, in general, be able to factor M, in order to solve (5). Though regular, M is certainly not positive definite or even semi-positive definite, so Cholesky is out. The existence of an LDL^t-factorization, on the other hand, is not ruled out a priori, provided both signs are allowed for the entries of D. If A is regular, the modified version of *LDLT*, which accpets negative pivots,

[2]In particular, according to Sylvester's "law of inertia" [Kn], the number of positive entries of D is the number of positive eigenvalues of A. Running the program on $A - \sigma 1$ thus allows one to count the eigenvalues larger than σ . . . when the algorithm succeeds.

will work. But otherwise it can fail, as the counter-example $A = \{\{1, 0\}, \{0, 0\}\}$ and $B = \{0, 1\}$ will show: Though regular, the matrix $\{\{1, 0, 0\}, \{0, 0, 1\}, \{0, 1, 0\}\}$ has no LDL^t factorization.

So what is to be done? Remark that $A + B^tB$ is (strictly) positive definite, for $((A + B^tB)x, x) = 0$ implies $(Ax, x) = 0$ and $Bx = 0$, therefore $x \in \ker(A) \cap \ker(B)$. And since $Bx = c$ if x is solution, (5) and the following system are equivalent:

$$(6) \qquad \begin{vmatrix} A + B^tB & B^t \\ B & 0 \end{vmatrix} \begin{vmatrix} x \\ \lambda \end{vmatrix} = \begin{vmatrix} b + B^tc \\ c \end{vmatrix}.$$

But now the new (augmented) matrix M is LDL^t-factorizable. Working by blocks to begin with, we get

$$(7) \qquad \begin{vmatrix} A + B^tB & B^t \\ B & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0 \\ \beta & 1 \end{vmatrix} \begin{vmatrix} A + B^tB & 0 \\ 0 & -\gamma \end{vmatrix} \begin{vmatrix} 1 & \beta^t \\ 0 & 1 \end{vmatrix},$$

with $\beta = B(A + B^tB)^{-1}$ and $\gamma = B(A + B^tB)^{-1}B^t$. Assuming the programming environment is an LDL^t package (even the standard one, that assumes $D > 0$, will do), complete with its factorizer and downward and upward triangular solvers, the essential task consists in factorizing $A + B^tB$ and $\gamma = B(A + B^tB)^{-1}B^t$, which are both positive definite, since we have assumed $\ker(B^t) = 0$. The factorization of γ is necessary in order to get λ, by solving

$$B(A + B^tB)^{-1}B^t\lambda = B(A + B^tB)^{-1}(b + B^tc) - c,$$

and that of $A + B^tB$ to obtain x, by solving

$$(A + B^tB)x = b + B^t(c - \lambda),$$

hence a solution in two steps. Since these factorizations allow passing from the block form (7) to the full-fledged LDL^t factorization of the augmented M, we may do it all in one stroke by applying the LDL^t package to system (5), provided the program lets negative pivots pass.

Remark B.5. The Lagrangian of (5) was $\mathcal{L}(x, \lambda) = (Ax, x) + 2(\lambda, Bx) - 2(b, x)$, and although strictly convex when x is restricted to $\ker(B)$, it was not strictly convex in x. The "augmented" Lagrangian of (6), $\mathcal{L}(x, \lambda) = (Ax, x) + |Bx|^2 + 2(\lambda, Bx) - 2(b, x)$, is. (Note that one may search for its saddle point $\{x, \lambda\}$ by some iterative method, such as the Arrow–Hurwicz–Uzawa algorithm [AH]. We don't discuss this alternative here, having direct methods in view.) One may think of a

more general, possibly better form for it: $\mathcal{L}(x, \lambda) = (Ax, x) + \rho\,|Bx|^2 + 2(\lambda, Bx) - 2(b, x)$, where ρ is a positive constant, the optimal value of which depends of course on how B has been built. Note that $\rho = 0$ is allowed if A is regular, the easy case. $\Diamond$

Remark B.6. It has been proposed [Ve] that (1) be replaced by the obviously equivalent system

$$(8) \qquad \begin{vmatrix} -1 & B & 1 \\ B^t & A & B^t \\ 1 & B & -1 \end{vmatrix} \begin{vmatrix} \mu \\ x \\ \lambda \end{vmatrix} = \begin{vmatrix} c \\ b \\ c \end{vmatrix},$$

that is, *two* Lagrange multipliers instead of one. The matrix of (6) is indeed LDL^t-factorizable, under the above hypotheses:

$$\begin{vmatrix} -1 & B & 1 \\ B^t & A & B^t \\ 1 & B & -1 \end{vmatrix} = \begin{vmatrix} 1 & & \\ -B^t & 1 & \\ -1 & 2\beta & 1 \end{vmatrix} \begin{vmatrix} -1 & & \\ & A + B^t B & \\ & & -4\gamma \end{vmatrix} \begin{vmatrix} 1 & -B & -1 \\ & 1 & 2\beta^t \\ & & 1 \end{vmatrix},$$

with the same β and γ as above. So one can solve (1) this way, by running an LDL^t package on (8). But the numerical effort involved is no less than was required by the above method (a bit *more*, actually, since some arithmetic is wasted on numerically retrieving the first column block of L, that is $\{1, -B^t, -1\}$, which is already known). As the heuristic leading from (5) to (8) is quite obscure, in comparison with the easily motivated passage from (5) to (6), this "double-multiplier" approach is more of a curiosity than a real alternative. $\Diamond$

REFERENCES

[AH] K. Arrow, L. Hurwicz, H. Uzawa: **Studies in Nonlinear Programming**, Stanford U.P. (Stanford), 1958.

[BA] J. Backus: "Can Programming be Liberated from the Von Neumann Style? A Functional Style and its Algebra of Programs", **Comm. ACM, 21,** 8 (1978), pp. 613–641.

[CC] F. Chaitin-Chatelin, V. Fraysse: **Lectures on Finite Precision Computation,** SIAM (Philadelphia), 1996.

[GL] G.H. Golub, C.F. Van Loan: **Matrix Computations**, North Oxford Academic (Oxford) & Johns Hopkins U.P. (Baltimore), 1983.

[Kn] D.E. Knuth: "A permanent inequality", **Amer. Math. Monthly, 88** (1981), pp. 731–740.

[MW] **MATLAB™ for Macintosh Computers, User's Guide**, The MathWorks, Inc. (Natick, Ma, USA), 1991.

[Me] B. Meyer: **Object-oriented Software Construction**, Prentice Hall (New York), 1988.

[Ve] Int. report by M. Verpeaux, CEA-DEMT, Saclay. Cf. J. Pellet: **Dualisation des conditions aux limites**, Document ASTER R3.03.01 (EdF, Clamart), 27 11 91.