If the ODE is stiff, we might prefer a *Newton–Raphson* method, namely

$$\boldsymbol{y}_{n+s}^{[j+1]} = \boldsymbol{y}_{n+s}^{[j]} - \left[I - \sigma_s h \frac{\partial \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]})}{\partial \boldsymbol{y}} \right]^{-1} [\boldsymbol{y}_{n+s}^{[j]} - \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]}) - \boldsymbol{v}].$$

The justification of the above is as follows: suppose that $y_{n+s}^{[j]}$ is an approximation to the solution. We linearise (4.15) locally about $(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]})$,

$$\boldsymbol{y}_{n+s} - \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}) - \boldsymbol{v} \approx [\boldsymbol{y}_{n+s}^{[j]} - \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]}) - \boldsymbol{v}] + \left[I - \sigma_s h \frac{\partial \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]})}{\partial \boldsymbol{y}} \right] (\boldsymbol{y}_{n+s} - \boldsymbol{y}_{n+s}^{[j]})$$

and choose $\boldsymbol{y}_{n+s}^{[j+1]}$ by equating the right-hand side to $\boldsymbol{0}$.

The snag is that repeatedly evaluating and inverting (i.e. LU-factorizing) the Jacobian matrix in every iteration is *very* expensive. The remedy is to implement the *modified Newton–Raphson method*, namely

$$\boldsymbol{y}_{n+s}^{[j+1]} = \boldsymbol{y}_{n+s}^{[j]} - \left[I - \sigma_s h \frac{\partial \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[0]})}{\partial \boldsymbol{y}} \right]^{-1} [\boldsymbol{y}_{n+s}^{[j]} - \sigma_s h \boldsymbol{f}(t_{n+s}, \boldsymbol{y}_{n+s}^{[j]}) - \boldsymbol{v}]. \tag{4.16}$$

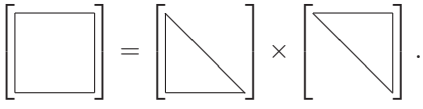
Thus, the Jacobian need be evaluated only *once* a step.

The only role the Jacobian matrix plays in (4.16) is to ensure convergence: its precise value makes no difference to the ultimate value of $\lim_{j\to\infty} \boldsymbol{y}_{n+s}^{[j]}$. Therefore we might replace it with a finite-difference approximation, evaluate it once every several steps etc. *Important observation for future use:* Implementation of (4.16) requires repeated solution of linear algebraic systems *with the same matrix.* We will soon study LU factorization of matrices, and there this remark will be appreciated as important and lead to substantial savings. For stiff equations it is much cheaper to solve nonlinear algebraic equations with (4.16) than using a minute step size with a 'bad' (e.g., explicit multistep or explicit RK) method.

5 Numerical linear algebra

5.1 LU factorization and its generalizations

Let A be a real $n \times n$ matrix. We say that the $n \times n$ matrices L and U are an LU *factorization* of A if ***(1)*** L is lower triangular (i.e., $L_{i,j} = 0$, $i < j$); ***(2)*** U is upper triangular, $U_{i,j} = 0$, $i > j$; and ***(3)*** $A = LU$. Therefore the factorization takes the form



Application 1 Calculation of a determinant: $\det A = (\det L)(\det U) = (\prod_{k=1}^{n} L_{k,k}) \cdot (\prod_{k=1}^{n} U_{k,k})$.

Application 2 Testing for nonsingularity: $A = LU$ is nonsingular iff all the diagonal elements of L and U are nonzero.

Application 3 Solution of linear systems: Let $A = LU$ and suppose we wish to solve $A\boldsymbol{x} = \boldsymbol{b}$. This is the same as $L(U\boldsymbol{x}) = \boldsymbol{b}$, which we decompose into $L\boldsymbol{y} = \boldsymbol{b}$, $U\boldsymbol{x} = \boldsymbol{y}$. Both latter systems are

[1] Corrections and suggestions to these notes should be emailed to `A.Iserles@damtp.cam.ac.uk`. All handouts are available on the WWW at the URL `http://www.damtp.cam.ac.uk/user/na/PartIB/Handouts.html`.

triangular and can be calculated easily. Thus, $L_{1,1}y_1 = b_1$ gives y_1, next $L_{2,1}y_1 + L_{2,2}y_2 = b_2$ yields y_2 etc. Having found $\boldsymbol{y}$, we solve for $\boldsymbol{x}$ in reverse order: $U_{n,n}x_n = y_n$ gives x_n, $U_{n-1,n-1}x_{n-1} + U_{n-1,n}x_n = y_{n-1}$ produces x_{n-1} and so on. This requires $\mathcal{O}(n^2)$ computational operations (usually we only bother to count multiplications/divisions).

Application 4 The inverse of A: It is straightforward to devise a direct way of calculating the inverse of triangular matrices, subsequently forming $A^{-1} = U^{-1}L^{-1}$.

Why not Cramer's rule? For the uninitiated, a recursive definition of a determinant may seem to be a good method for its calculation (and perhaps even for the solution of linear systems with Cramer's rule). Unfortunately, the number of operations increases like $n!$. Thus, on a 10^9 flop/sec. computer

$$n = 10 \quad \Rightarrow \quad 10^{-4} \text{ sec.}, \qquad n = 20 \quad \Rightarrow \quad 17 \text{ min}, \qquad n = 30 \quad \Rightarrow \quad 4 \times 10^5 \text{ years}.$$

The calculation of LU factorization We denote the *columns* of L by $\boldsymbol{l}_1, \boldsymbol{l}_2, \ldots, \boldsymbol{l}_n$ and the *rows* of U by $\boldsymbol{u}_1^\top, \boldsymbol{u}_2^\top, \ldots, \boldsymbol{u}_n^\top$. Hence

$$A = LU = \begin{bmatrix} \boldsymbol{l}_1 & \boldsymbol{l}_2 & \cdots & \boldsymbol{l}_n \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1^\top \\ \boldsymbol{u}_2^\top \\ \vdots \\ \boldsymbol{u}_n^\top \end{bmatrix} = \sum_{k=1}^{n} \boldsymbol{l}_k \boldsymbol{u}_k^\top. \tag{5.1}$$

Since the first $k-1$ components of $\boldsymbol{l}_k$ and $\boldsymbol{u}_k$ are all zero, each rank-one matrix $\boldsymbol{l}_k\boldsymbol{u}_k^\top$ has zeros in its first $k-1$ rows and columns.

Assume that the factorization exists (hence the diagonal elements of L are nonzero) and that A is nonsingular. Since $\boldsymbol{l}_k\boldsymbol{u}_k^\top$ stays the same if we replace $\boldsymbol{l}_k \to \alpha\boldsymbol{l}_k$, $\boldsymbol{u}_k \to \alpha^{-1}\boldsymbol{u}_k$, where $\alpha \neq 0$, we may assume w.l.o.g. that all diagonal elements of L equal one. In other words, the kth row of $\boldsymbol{l}_k\boldsymbol{u}_k^\top$ is $\boldsymbol{u}_k^\top$ and its kth column is $U_{k,k}$ times $\boldsymbol{l}_k$.

We begin our calculation by extracting $\boldsymbol{l}_1$ and $\boldsymbol{u}_1^\top$ from A, and then proceed similarly to extract $\boldsymbol{l}_2$ and $\boldsymbol{u}_2^\top$, etc.

First we note that since the leading $k-1$ elements of $\boldsymbol{l}_k$ and $\boldsymbol{u}_k$ are zero for $k \geq 2$, it follows from (5.1) that $\boldsymbol{u}_1^\top$ is the first row of A and $\boldsymbol{l}_1$ is the first column of A, divided by $A_{1,1}$ (so that $L_{1,1} = 1$).

Next, having found $\boldsymbol{l}_1$ and $\boldsymbol{u}_1$, we form the matrix $A_1 = A - \boldsymbol{l}_1\boldsymbol{u}_1^\top = \sum_{k=2}^{n} \boldsymbol{l}_k\boldsymbol{u}_k^\top$. The first row & column of A_1 are zero and it follows that $\boldsymbol{u}_2^\top$ is the second row of A_1, while $\boldsymbol{l}_2$ is its second column, scaled so that $L_{2,2} = 1$.

The LU algorithm: Set $A_0 := A$. For all $k = 1, 2, \ldots, n$ set $\boldsymbol{u}_k^\top$ to the kth row of A_{k-1} and $\boldsymbol{l}_k$ to the kth column of A_{k-1}, scaled so that $L_{k,k} = 1$. Further, calculate $A_k := A_{k-1} - \boldsymbol{l}_k\boldsymbol{u}_k^\top$ before incrementing k.

Note that all elements in the first k rows & columns of A_k are zero. Hence, we can use the storage of the original A to accumulate L and U. The full LU factorization requires $\mathcal{O}(n^3)$ computational operations.

Relation to Gaussian elimination The equation $A_k = A_{k-1} - \boldsymbol{l}_k\boldsymbol{u}_k^\top$ has the property that the jth row of A_k is the jth row of A_{k-1} minus $L_{j,k}$ times $\boldsymbol{u}_k^\top$ (the kth row of A_{k-1}). Moreover, the multipliers $L_{k,k}, L_{k+1,k}, \ldots, L_{n,k}$ are chosen so that the outcome of this *elementary row operation* is that the kth column of A_k is zero. This construction is analogous to Gaussian elimination for solving $A\boldsymbol{x} = \boldsymbol{b}$. An important difference is that in LU we do not consider the right hand side $\boldsymbol{b}$ until the factorization is complete. This is useful e.g. when there are many right hand sides, in particular if not all the $\boldsymbol{b}$'s are known at the outset: in Gaussian elimination the solution for each new $\boldsymbol{b}$ would require $\mathcal{O}(n^3)$ computational operations, whereas with LU factorization $\mathcal{O}(n^3)$ operations are required for the initial factorization, but then the solution for each new $\boldsymbol{b}$ only requires just $\mathcal{O}(n^2)$ operations.