10. Solución de Ecuaciones.

A continuación y con el fin de terminar la parte de la asignatura que tiene que ver con la formulación del MEF, se resume el problema matemático que es necesario resolver para obtener los resultados del análisis, es decir los desplazamientos nodales. Se muestra un diagrama en el que se puede visualizar el proceso general de análisis por EF. Se proporcionan datos que permiten hacerse una idea de los recursos computacionales necesarios para obtener la solución, datos en función del número total de grados de libertad, manejando la matriz de rigidez completa. Se comenta que las propiedades que posee esta matriz hacen que en lugar de manejar todos sus elementos, es posible utilizar únicamente aquellos elementos no nulos, situados a un lado de la diagonal principal, lo que se denomina “matriz en banda”. Y se proporcionan datos de cómo disminuyen los recursos computacionales necesarios en este caso, es decir, cuando sólo de maneja los términos no nulos situados a un lado de la diagonal principal. A continuación se comenta, haciendo referencia a Mathematica, como se lleva a cabo el almacenamiento de esos términos no nulos de la matriz situados a un lado de la diagonal principal. Así como la forma de incluir las condiciones de contorno. Finalizando la lección comentando el proceso de obtención de los desplazamientos aprovechando las propiedades de la matriz de rigidez.

Esta lección figura perfectamente explicado en el Tema 26 del Curso Introductorio al Método de los Elementos Finitos que se cursa en la Universidad de Colorado en Boulder, bajo la dirección del Prof. Carlos A. Felippa, y por ello lo proporcionamos completo en esta sección.

CHAPTER 26. Solución de Ecuaciones.
Carlos A. Felippa.

26

Solving FEM Equations

TABLE OF CONTENTS

§26.1. MOTIVATION FOR SPARSE SOLVERS

In the Direct Stiffness Method (DSM) of finite element analysis, the element stiffness matrices and consistent nodal force vectors are immediately assembled to form the *master stiffness matrix* and *master force vector*, respectively, by the process called *merge*. The basic rules that govern the assembly process are described in Chapter 25. For simplicity the description that follows assumes that no MultiFreedom Constraints (MFCs) are present.

The end result of the assembly process are the master stiffness equations

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{26.1}$$

where $\mathbf{K}$ is the master stiffness matrix, $\mathbf{f}$ the vector of node forces and $\mathbf{u}$ the vector or node displacements. Upon imposing the displacement boundary conditions, the system (26.1) is solved for the unknown node displacements. The solution concludes the main phase of DSM computations.

In practical applications the order of the stiffness system (26.1) can be quite large. Systems of order 1000 to 10000 are routinely solved in commercial software. Larger ones (say up to 100000 equations) are not uncommon and even millions of equations are being solved on suoercomputers. Presently the record is about 50 million.

In *linear* FEM analysis the cost of solving this system of equations rapidly overwhelms other computational phases. Much attention has therefore given to matrix processing techniques that economize storage and solution time by taking advantage of the special structure of the stiffness matrix.

The master force vector is stored as a conventional one-dimensional array of length equal to the number N of degrees of freedom. This storage arrangement presents no particular difficulties even for very large problem sizes. Handling the master stiffness matrix, however, presents computational difficulties.

§26.1.1. The Curse of Fullness

If $\mathbf{K}$ is stored and processed as if it were a *full* matrix, the storage and processing time resources rapidly becomes prohibitive as N increases. This is illustrated in Table 26.1, which summarizes the storage and factor-time requirements for orders $N = 10^4$, 10^5 and 10^6.[1]

As regards memory needs, a full square matrix stored without taking advantage of symmetry, requires storage for N^2 entries. If each entry is an 8-byte, double precision floating-point number, the required storage is $8N^2$ bytes. Thus, a matrix of order $N = 10^4$ would require 8×10^8 bytes or 800 MegaBytes (MB) for storage.

For large N the solution of (26.1) is dominated by the factorization of $\mathbf{K}$, an operation discussed in §26.2. This operation requires approximately $N^3/6$ floating point operation units. [A floating-point operation unit is conventionally defined as a (multiply,add) pair plus associated indexing and data movement operations.] Now a fast workstation can typically do 10^7 of these operations per second, whereas a supercomputer may be able to sustain 10^9 or more. These times assume that the entire matrix is kept in high-speed memory; for otherwise the elapsed time may increase by factors of 10 or

[1] The factor times given in that table reflect 1998 computer technology. To update to 2002, divide times by 5 to 10.

Table 26.1 Storage & Solution Time for a Fully-Stored Stiffness Matrix

Matrix order N	Storage (double prec)	Factor op.units (FLOPS)	Factor time workstation (or fast PC)	Factor time supercomputer
10^4	800 MB	$10^{12}/6$	3 hrs	2 min
10^5	80 GB	$10^{15}/6$	4 mos	30 hrs
10^6	8 TB	$10^{18}/6$	300 yrs	3 yrs

Table 26.2 Storage & Solution Time for a Skyline Stored Stiffness Matrix Assuming $B = \sqrt{N}$

Matrix order N	Storage (double prec)	Factor op.units (FLOPS)	Factor time workstation (or fast PC)	Factor time supercomputer
10^4	8 MB	$10^8/2$	5 sec	0.05 sec
10^5	240 MB	$10^{10}/2$	8 min	5 sec
10^6	8000 MB	$10^{12}/2$	15 hrs	8 min

more due to I/O transfer operations. The elaspsed timed estimated given in Table 26.1 illustrate that for present computer resources, orders above 10^4 would pose significant computational difficulties.

§26.1.2. The Advantages of Sparsity

Fortunately a very high percentage of the entries of the master stiffness matrix **K** are zero. Such matrices are call *sparse*. There are clever programming techniques that take advantage of sparsity that fit certain patterns. Although a comprehensive coverage of such techniques is beyond the scope of this course, we shall concentrate on a particular form of sparse scheme that is widely use in FEM codes: *skyline* storage. This scheme is simple to understand, manage and implement, while cutting storage and processing times by orders of magnitude as the problems get larger.

The skyline storage format is a generalization of its widely used predecessor called the *band storage* scheme. A matrix stored in accordance with the skyline format will be called a *skymatrix* for short. Only symmetric skymatrices will bve considered here, since the stiffness matrices in linear FEM are symmetric.

If a skymatrix of order N can be stored in S memory locations, the ratio $B = S/N$ is called the *mean bandwidth*. If the entries are, as usual, 8-byte double-precision floating-point numbers, the storage requirement is $8NB$ bytes. The factorization of a skymatrix requires approximately $\frac{1}{2}NB^2$ floating-point operation units. In two-dimensional problems B is of the order of $\sqrt{N}$. Under this assumption, storage requirements and estimated factorization times for $N = 10^4$, $N = 10^5$ and $N = 10^6$ are reworked in Table 26.2. It is seen that by going from full to skyline storage significant reductions in computer resources have been achieved. For example, now $N = 10^4$ is easy on a

workstation and trivial on a supercomputer. Even a million equations do not look far-fetched on a supercomputer as long as enough memory is available.

In preparation for assembling **K** as a skymatrix one has to set up several auxiliary arrays related to nodes and elements. These auxiliary arrays are described in the previous Chapter. Knowledge of that material is useful for understanding the following description.

§26.2. SPARSE SOLUTION OF STIFFNESS EQUATIONS

§26.2.1. Skyline Storage Format

The skyline storage arrangement for **K** is best illustrated through a simple example. Consider the 6×6 stiffness matrix

$$\mathbf{K} = \begin{bmatrix} K_{11} & 0 & K_{13} & 0 & 0 & K_{16} \\ & K_{22} & 0 & K_{24} & 0 & 0 \\ & & K_{33} & K_{34} & 0 & 0 \\ & & & K_{44} & 0 & K_{46} \\ & & & & K_{55} & K_{56} \\ symm & & & & & K_{66} \end{bmatrix} \tag{26.2}$$

Since the matrix is symmetric only one half, the upper triangle in the above display, need to be shown.

Next we define the *envelope* of **K** as follows. From each diagonal entry move *up* the corresponding column until the last nonzero entry is found. The envelope separates that entry from the rest of the upper triangle. The remaining zero entries are conventionally removed:

$$\mathbf{K} = \begin{bmatrix} K_{11} & & K_{13} & & & K_{16} \\ & K_{22} & 0 & K_{24} & & 0 \\ & & K_{33} & K_{34} & & 0 \\ & & & K_{44} & & K_{46} \\ & & & & K_{55} & K_{56} \\ symm & & & & & K_{66} \end{bmatrix} \tag{26.3}$$

What is left constitute the *skyline profile* of *skyline template* of the matrix. A sparse matrix that can be profitably stored in this form is called a *skymatrix* for brevity. Notice that the skyline profile may include zero entries. During the factorization step discussed below these zero entries will in general become nonzero, a phenomenon that receives the name *fill-in*.

The key observation is that only *the entries in the skyline template need to be stored*, because *fill-in in the factorization process will not occur outside the envelope*. To store these entries it is convenient to use a one-dimensional *skyline array*:

$$\mathtt{s}: \quad [\, K_{11},\, K_{22},\, K_{13},\, 0,\, K_{33},\, K_{24},\, K_{34},\, K_{44},\, K_{55},\, K_{16},\, 0,\, 0,\, K_{46},\, K_{56},\, K_{66}\,] \tag{26.4}$$

This array is complemented by a $(N+1)$ integer array p that contains addresses of *diagonal locations*. The array has $N+1$ entries. The $(i+1)^{th}$ entry of p has the location of the i^{th} diagonal

entry of **K** in s. For the example matrix:

$$\mathrm{p}: \quad [0,\ 1,\ 2,\ 5,\ 8,\ 9,\ 15] \tag{26.5}$$

In the previous Chapter, this array was called the Global Skyline Diagonal Location Table, or GSDLT.

Equations for which the displacement component is prescribed are identified by a *negative* diagonal location value. For example if u_3 and u_5 are prescribed displacement components in the test example, then

$$\mathrm{p}: \quad [0,\ 1,\ 2,\ -5,\ 8,\ -9,\ 15] \tag{26.6}$$

REMARK 26.1

In Fortran it is convenient to dimension the diagonal location array as `p(0:n)` so that indexing begins at zero. In C this is the standard indexing.

§26.2.2. Factorization

The stiffness equations (26.1) are solved by a direct method that involves two basic phases: *factorization* and *solution*.

In the first stage, the skyline-stored symmetric stiffness matrix is factored as

$$\mathbf{K} = \mathbf{LDU} = \mathbf{LDL}^T = \mathbf{U}^T\mathbf{DU}, \tag{26.7}$$

where **L** is a unit lower triangular matrix, **D** is a nonsingular diagonal matrix, and **U** and **L** are the transpose of each other. The original matrix is overwritten by the entries of $\mathbf{D}^{-1}$ and **U**; details may be followed in the program implementation. No pivoting is used in the factorization process. This factorization is carried out by *Mathematica* module `SymmSkyMatrixFactor`, which is described later in this Chapter.

§26.2.3. Solution

Once **K** has been factored, the solution **u** for a given right hand side **f** is obtained by carrying out three stages:

$$\textit{Forward reduction}: \quad \mathbf{Lz} = \mathbf{f}, \tag{26.8}$$

$$\textit{Diagonal scaling}: \quad \mathbf{Dy} = \mathbf{z}, \tag{26.9}$$

$$\textit{Back substitution}: \quad \mathbf{Uu} = \mathbf{y}, \tag{26.10}$$

where **y** and **z** are intermediate vectors. These stages are carried out by *Mathematica* modules `SymmSkyMatrixVectorSolve`, which is described later.

§26.2.4. Treating MFCs with Lagrange Multipliers

In *Mathematica* implementations of FEM, MultiFreedom Constraints (MFCs) are treated with Lagrange multipliers. There is one multiplier for each constraint. The multipliers are placed at the end of the solution vector.

Specifically, let the `nummul`>0 MFCs be represented in matrix form as $\mathbf{C}\mathbf{u} = \mathbf{g}$, where $\mathbf{C}$ and $\mathbf{g}$ are given, and let the `nummul` multipliers be collected in a vector $\boldsymbol{\lambda}$. The multiplier-augmented master stiffness equations are

$$\begin{bmatrix} \mathbf{K} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix} \tag{26.11}$$

or

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{26.12}$$

where the symmetric matrix $\mathbf{A}$, called a stiffness-bordered matrix, is of order `numdof+nummul`.

The stiffness bordered matrix is also stored in skyline form, and the previous solution procedure applies, as long as the skyline array is properly constructed as described in the previous Chapter.

The main difference with respect to the no-MFC case is that, because of the configuration (26.11), $\mathbf{A}$ can no longer be positive definite. In principle pivoting should be used during the factorization of $\mathbf{A}$ to forestall possible numerical instabilities. Pivoting can have a detrimental effect on solution efficiency because entries can move outside of the skyline template. However, by placing the $\boldsymbol{\lambda}$ at the end such difficulties will not be encountered if $\mathbf{K}$ is positive definite, and the constraints are linearly independent (that is, $\mathbf{C}$ has full rank). Thus pivoting is not necessary.

§26.3. A SKYSOLVER IMPLEMENTATION

The remaining sections of this revised Chapter describe a recent implementation of the skyline solver and related routines in *Mathematica*. This has been based on similar Fortran codes used since 1967.

§26.3.1. Skymatrix Representation

In what follows the computer representation in *Mathematica* of a symmetric skymatrix will be generally denoted by the symbol S. Such a representation consists of a list of two numeric objects:

```
S = { p, s }
```
(26.13)

Here p=GSDLT is the Global Skyline Diagonal Location Table introduced in §11.6, and `s` is the array of skymatrix entries, arranged as described in the previous section. This array usually consists of floating-point numbers, but it may also contain exact integers or factions, or even symbolic entries.

For example, suppose that the numerical entries of the 6 × 6 skymatrix (26.10) are actually

$$\mathbf{K} = \begin{bmatrix} 11 & & 13 & & & 16 \\ & 22 & 0 & 24 & & 0 \\ & & 33 & 34 & & 0 \\ & & & 44 & & 46 \\ & & & & 55 & 56 \\ symm & & & & & 66 \end{bmatrix} \tag{26.14}$$

Cell 26.1 Factorization of a Symmetric SkyMatrix

```
SymmSkyMatrixFactor[S_,tol_]:= Module[
  {p,a,fail,i,j,k,l,m,n,ii,ij,jj,jk,jmj,d,s,row,v},
  row=SymmSkyMatrixRowLengths[S]; s=Max[row];
  {p,a}=S; n=Length[p]-1; v=Table[0,{n}]; fail=0;
  Do [jj=p[[j+1]]; If [jj<0|row[[j]]==0, Continue[]]; d=a[[jj]];
      jmj=Abs[p[[j]]]; jk=jj-jmj;
     Do [i=j-jk+k; v[[k]]=0; ii=p[[i+1]];
        If [ii<0, Continue[]]; m=Min[ii-Abs[p[[i]]],k]-1;
            ij=jmj+k; v[[k]]=a[[ij]];
            v[[k]]-=Take[a,{ii-m,ii-1}].Take[v,{k-m,k-1}];
            a[[ij]]=v[[k]]*a[[ii]],
     {k,1,jk-1}];
      d-=Take[a,{jmj+1,jmj+jk-1}].Take[v,{1,jk-1}];
      If [Abs[d]<tol*row[[j]], fail=j; a[[jj]]=Infinity; Break[] ];
      a[[jj]]=1/d,
  {j,1,n}];
  Return[{{p,a},fail}]
];

SymmSkyMatrixRowLengths[S_]:= Module[
  {p,a,i,j,n,ii,jj,m,d,row},
  {p,a}=S;  n=Length[p]-1; row=Table[0,{n}];
  Do [ii=p[[i+1]]; If [ii<0, Continue[]]; m=ii-i; row[[i]]=a[[ii]]^2;
     Do [If [p[[j+1]]>0, d=a[[m+j]]^2; row[[i]]+=d; row[[j]]+=d],
     {j,Max[1,Abs[p[[i]]]-m+1],Min[n,i]-1}],
  {i,1,n}];  Return[Sqrt[row]];
];
```

Its *Mathematica* representation, using the symbols (26.13) is

```
p= { 0,1,2,5,8,9,15};
s= { 11,22,13,0,33,24,34,44,55,16,0,0,46,56,66};          (26.15)
S= { p, s};
```

or more directly

```
S={ { 0,1,2,5,8,9,15} , {11,22,13,0,33,24,34,44,55,16,0,0,46,56,66} } ;
                                                          (26.16)
```

[The remaining sections on details of skyline processing logic, marked with a *, will not be covered in class. They are intended for a more advanced course.]

Cell 26.2 Factorization Test Input

```
ClearAll[n]; n=5; SeedRandom[314159];
p=Table[0,{n+1}]; Do[p[[i+1]]=p[[i]]+
    Max[1,Min[i,Round[Random[]*i]]],{i,1,n}];
    a=Table[1.,{i,1,p[[n+1]]}];
Print["Mean Band=",N[p[[n+1]]/n]];
S={p,a};
Sr=SymmSkyMatrixLDUReconstruct[S];
Print["Reconstructed SkyMatrix:"]; SymmSkyMatrixLowerTrianglePrint[Sr];
SymmSkyMatrixLowerTriangleMap[Sr];
Print["eigs=",Eigenvalues[SymmSkyMatrixConvertToFull[Sr]]];
x=Table[{N[i],3.,(-1)^i*N[n-i]},{i,1,n}];
Print["Assumed x=",x];
b=SymmSkyMatrixColBlockMultiply[Sr,x];
(*x=Transpose[x]; b=SymmSkyMatrixRowBlockMultiply[Sr,x];*)
Print["b=Ax=",b];
Print[Timing[{F,fail}=SymmSkyMatrixFactor[Sr,10.^(-12)]]];
If [fail!=0, Print["fail=",fail]; Abort[]];
Print["F=",F]; Print["fail=",fail];
Print["Factor:"];
SymmSkyMatrixLowerTrianglePrint[F];
x=SymmSkyMatrixColBlockSolve[F,b];
(*x=SymmSkyMatrixRowBlockSolve[F,b];*)
Print["Computed x=",x//InputForm];
```

§26.3.2. *Skymatrix Factorization

Module SymmSkyMatrixFactor, listed in Cell 26.1, factors a symmetric skymatrix into the product **LDU** where **L** is the transpose of **U**. No pivoting is used. The module is invoked as

```
{ Sf,fail } = SymmSkyMatrixFactor[S,tol]
```

The input arguments are

`S`	The skymatrix to be factored, stored as the two-object list `{p,s}`; see previous subsection.
`tol`	Tolerance for singularity test. The appropriate value of `tol` depends on the kind of skymatrix entries stored in `s`. If the skymatrix entries are floating-point numbers handled by default in double precision arithmetic, `tol` should be set to $8\times$ or $10\times$ the machine precision in that kind of arithmetic. The factorization aborts if, when processing the `j-th` row, $d_j \leq tol * r_j$, where d_j is the computed j^{th} diagonal entry of **D**, and r_j is the Euclidean norm of the j^{th} skymatrix row. If the skymatrix entries are exact (integers, fractions or symbols), `tol` should be set to zero. In this case exact singularity is detectable, and the factorization aborts only on that condition.

The outputs are:

Cell 26.3 Output from Program of Cells 26.1 and 26.2

```
Mean Band=1.6
Reconstructed SkyMatrix:

            Col   1    Col   2    Col   3    Col   4    Col   5
 Row   1     1.0000
 Row   2                1.0000
 Row   3                1.0000     2.0000
 Row   4                                    1.0000
 Row   5                           1.0000     1.0000     3.0000

    1 2 3 4 5

  1  +
  2    +
  3    + +
  4        +
  5      + + +
eigs={3.9563, 2.20906, 1., 0.661739, 0.172909}
Assumed x={{1., 3., -4.}, {2., 3., 3.}, {3., 3., -2.}, {4., 3., 1.},

   {5., 3., 0}}
b=Ax={{1., 3., -4.}, {5., 6., 1.}, {13., 12., -1.}, {9., 6., 1.},

   {22., 15., -1.}}
{0.0666667 Second, {{{0, 1, 2, 4, 5, 8},

    {1., 1., 1., 1., 1., 1., 1., 1.}}, 0}}
F={{0, 1, 2, 4, 5, 8}, {1., 1., 1., 1., 1., 1., 1., 1.}}
fail=0
Factor:

            Col   1    Col   2    Col   3    Col   4    Col   5
 Row   1     1.0000
 Row   2                1.0000
 Row   3                1.0000     1.0000
 Row   4                                    1.0000
 Row   5                           1.0000     1.0000     1.0000
Computed x={{1., 3., -4.}, {2., 3., 3.}, {3., 3., -2.}, {4., 3., 1.},

   {5., 3., 0.}}
```

`Sf`	If `fail` is zero on exit, `Sf` is the computed factorization of S. It is a two-object list {p, du }, where du stores the entries of $\mathbf{D}^{-1}$ in the diagonal locations, and of **U** in its strict upper triangle.
`fail`	A singularity detection indicator. A zero value indicates that no singularity was detected. If

Cell 26.4 Solving for a Single RHS

```
SymmSkyMatrixVectorSolve[S_,b_]:= Module[
  {p,a,n,i,j,k,m,ii,jj,bi,x},
  {p,a}=S; n=Length[p]-1; x=b;
  If [n!=Length[x], Print["Inconsistent matrix dimensions in",
    " SymmSkyMatrixVectorSolve"]; Return[Null]];
  Do [ii=p[[i+1]];If [ii>=0, Continue[]]; ii=-ii; k=i-ii+Abs[p[[i]]]+1;
      bi=x[[i]]; If [bi==0, Continue[]];
      Do [jj=p[[j+1]], If [jj<0, Continue[]];
         m=j-i; If [m<0, x[[j]]-=a[[ii+m]]*bi; Break[]];
         ij=jj-m; If [ij>Abs[p[[j]]], x[[j]]-=a[[ij]]*bi],
      {j,k,n}],
  {i,1,n}];
  Do [ii=p[[i+1]]; If [ii<0, x[[i]]=0; Continue[]];
     imi=Abs[p[[i]]]; m=ii-imi-1;
     x[[i]]-=Take[a,{imi+1,imi+m}].Take[x,{i-m,i-1}],
  {i,1,n}];
  Do [ii=Abs[p[[i+1]]]; x[[i]]*=a[[ii]], {i,1,n}];
  Do [ii=p[[i+1]]; If [ii<0, x[[i]]=b[[i]]; Continue[]];
     m=ii-Abs[p[[i]]]-1;
     Do [ x[[i-j]]-=a[[ii-j]]*x[[i]], {j,1,m}],
  {i,n,1,-1}];
  Return[x]
];
```

`fail` returns `j>0`, the factorization was aborted at the `j-th` row. In this case `Sf` returns the aborted factorization with ∞ stored in d_j.

A test of `SymmSkyMatrixFactor` on the matrix (26.14) is shown in Cells 26.2 and 26.3. The modules that print and produce skyline maps used in the test program are described later in this Chapter.

§26.3.3. *Solving for One or Multiple RHS

Module `SymmSkyMatrixVectorSolve`, listed in Cell 26.4, solves the linear system $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{x}$, following the factorization of the symmetric skymatrix $\mathbf{A}$ by `SymmSkyMatrixFactor`. The module is invoked as

```
x = SymmSkyMatrixVectorSolve[Sf,b]
```

The input arguments are

`Sf`	The factored matrix returned by `SymmSkyMatrixFactor`.
`b`	The right-hand side vector to be solved for, stored as a single-level (one dimensional) list. If the `i-th` entry of `x` is prescribed, the known value must be supplied in this vector.

The outputs are:

`x`	The computed solution vector, stored as a single-level (one-dimensional) list. Prescribed solution components return the value of the entry in `b`.

Cell 26.5 Solving for a Block of Righ Hand Sides

```
SymmSkyMatrixColBlockSolve[S_,b_]:= Module[
  {p,a,n,nrhs,i,j,k,m,r,ii,jj,bi,x},
  {p,a}=S; n=Length[p]-1; x=b;
  If [n!=Dimensions[x][[1]], Print["Inconsistent matrix dimensions in",
    " SymmSkyMatrixBlockColSolve"]; Return[Null]]; nrhs = Dimensions[x][[2]];
  Do [ii=p[[i+1]];If [ii>=0, Continue[]]; ii=-ii; k=i-ii+Abs[p[[i]]]+1;
     Do [bi=x[[i,r]]; If [bi==0, Continue[]];
        Do [jj=p[[j+1]], If [jj<0, Continue[]];
           m=j-i; If [m<0,x[[j,r]]-=a[[ii+m]]*bi; Break[]];
           ij=jj-m; If [ij>Abs[p[[j]]], x[[j,r]]-=a[[ij]]*bi],
        {j,k,n}],
     {r,1,nrhs}],
  {i,1,n}];
  Do [ii=p[[i+1]]; If [ii<0, Do[x[[i,r]]=0,{r,1,nrhs}];Continue[]];
      imi=Abs[p[[i]]]; m=ii-imi-1;
     Do [ Do [ x[[i,r]]-=a[[imi+j]]*x[[i-m+j-1,r]], {j,1,m}], {r,1,nrhs}],
  {i,1,n}];
  Do [ii=Abs[p[[i+1]]]; Do[x[[i,r]]*=a[[ii]], {r,1,nrhs}], {i,1,n}];
  Do [ii=p[[i+1]]; If [ii<0, Do[x[[i,r]]=b[[i,r]],{r,1,nrhs}];Continue[]];
     m=ii-Abs[p[[i]]]-1;
     Do [ Do [ x[[i-j,r]]-=a[[ii-j]]*x[[i,r]], {j,1,m}], {r,1,nrhs}],
  {i,n,1,-1}];
  Return[x]
];
SymmSkyMatrixRowBlockSolve[S_,b_]:= Module[
  {p,a,n,nrhs,i,j,k,m,r,ii,jj,bi,x},
  {p,a}=S; n=Length[p]-1; x=b;
  If [n!=Dimensions[x][[2]], Print["Inconsistent matrix dimensions in",
    " SymmSkyMatrixBlockRowSolve"]; Return[Null]]; nrhs = Dimensions[x][[1]];
  Do [ii=p[[i+1]];If [ii>=0, Continue[]]; ii=-ii; k=i-ii+Abs[p[[i]]]+1;
     Do [bi=x[[r,i]]; If [bi==0, Continue[]];
        Do [jj=p[[j+1]], If [jj<0, Continue[]];
           m=j-i; If [m<0,x[[j,r]]-=a[[ii+m]]*bi; Break[]];
           ij=jj-m; If [ij>Abs[p[[j]]], x[[r,j]]-=a[[ij]]*bi],
        {j,k,n}],
     {r,1,nrhs}],
  {i,1,n}];
  Do [ii=p[[i+1]]; If [ii<0, Do[x[[r,i]]=0,{r,1,nrhs}];Continue[]];
      imi=Abs[p[[i]]]; m=ii-imi-1;
     Do [ Do [ x[[r,i]]-=a[[imi+j]]*x[[r,i-m+j-1]], {j,1,m}], {r,1,nrhs}],
  {i,1,n}];
  Do [ii=Abs[p[[i+1]]]; Do[x[[r,i]]*=a[[ii]], {r,1,nrhs}], {i,1,n}];
  Do [ii=p[[i+1]]; If [ii<0, Do[x[[r,i]]=b[[r,i]],{r,1,nrhs}];Continue[]];
     m=ii-Abs[p[[i]]]-1;
     Do [ Do [ x[[r,i-j]]-=a[[ii-j]]*x[[r,i]], {j,1,m}], {r,1,nrhs}],
  {i,n,1,-1}];
  Return[x]
];
```

Cell 26.6 Multiplying Skymatrix by Individual Vector

```
SymmSkyMatrixVectorMultiply[S_,x_]:= Module[
  {p,a,n,i,j,k,m,ii,b},
  {p,a}=S; n=Length[p]-1;
  If [n!=Length[x], Print["Inconsistent matrix dimensions in",
    " SymmSkyMatrixVectorMultiply"]; Return[Null]];
  b=Table[a[[ Abs[p[[i+1]]] ]]*x[[i]], {i,1,n}];
  Do [ii=Abs[p[[i+1]]]; m=ii-Abs[p[[i]]]-1; If [m<=0,Continue[]];
     b[[i]]+=Take[a,{ii-m,ii-1}].Take[x,{i-m,i-1}];
     Do [b[[i-k]]+=a[[ii-k]]*x[[i]],{k,1,m}],
  {i,1,n}];
  Return[b]
 ];

(*
ClearAll[n]; n=10; SeedRandom[314159];
p=Table[0,{n+1}]; Do[p[[i+1]]=p[[i]]+
  Max[1,Min[i,Round[Random[]*i]]],{i,1,n}];
a=Table[1.,{i,1,p[[n+1]]}];
Print["Mean Band=",N[p[[n+1]]/n]];
S={p,a};
Sr=SymmSkyMatrixLDUReconstruct[S];
Print["Reconstructed SkyMatrix:"]; SymmSkyMatrixLowerTrianglePrint[Sr];
SymmSkyMatrixLowerTriangleMap[Sr];
x=Table[1.,{i,1,n}];
b=SymmSkyMatrixVectorMultiply[Sr,x];
Print["b=Ax=",b];*)
```

Sometimes it is necessary to solve linear systems for multiple ($m > 1$) right hand sides. One way to do that is to call `SymmSkyMatrixVectorSolve` repeatedly. Alternatively, if m right hand sides are collected as columns of a rectangular matrix **B**, module `SymmSkyMatrixColBlockSolve` may be invoked as

```
X = SymmSkyMatrixVectorSolve[Sf,B]
```

to provide the solution **X** of $\mathbf{SX} = \mathbf{B}$. This module is listed in Cell 26.5. The input arguments and function returns have the same function as those described for `SymmSkyMatrixVectorSolve`. The main difference is that `B` and `X` are matrices (two-dimensional lists) with the righ-hand side and solution vectors as columns. There is a similar module `SymmSkyMatrixRowBlockSolve`, notlisted here, which solves for multiple right hand sides stored as rows of a matrix.

§26.3.4. *Matrix-Vector Multiply

For various applications it is necessary to form the matrix-vector product

$$\mathbf{b} = \mathbf{S}\mathbf{x} \tag{26.17}$$

where S is a symmetric skymatrix and **x** is given.

Cell 26.7 Multiplying Skymatrix by Vector Block

```
SymmSkyMatrixColBlockMultiply[S_,x_]:= Module[
  {p,a,n,nrhs,i,j,k,m,r,ii,aij,b},
  {p,a}=S; n=Length[p]-1;
  If [n!=Dimensions[x][[1]], Print["Inconsistent matrix dimensions in",
    " SymmSkyMatrixColBlockMultiply"]; Return[Null]];
  nrhs = Dimensions[x][[2]]; b=Table[0,{n},{nrhs}];
  Do [ii=Abs[p[[i+1]]]; m=ii-Abs[p[[i]]]-1;
     Do [b[[i,r]]=a[[ii]]*x[[i,r]], {r,1,nrhs}];
     Do [j=i-k; aij=a[[ii-k]]; If [aij==0, Continue[]];
       Do [b[[i,r]]+=aij*x[[j,r]]; b[[j,r]]+=aij*x[[i,r]], {r,1,nrhs}],
     {k,1,m}],
  {i,1,n}];
  Return[b]
 ];

SymmSkyMatrixRowBlockMultiply[S_,x_]:= Module[
  {p,a,n,nrhs,i,j,k,m,r,ii,aij,b},
  {p,a}=S; n=Length[p]-1;
  If [n!=Dimensions[x][[2]], Print["Inconsistent matrix dimensions in",
    " SymmSkyMatrixRowBlockMultiply"]; Return[Null]];
  nrhs = Dimensions[x][[1]]; b=Table[0,{nrhs},{n}];
  Do [ii=Abs[p[[i+1]]]; m=ii-Abs[p[[i]]]-1;
     Do [b[[r,i]]=a[[ii]]*x[[r,i]], {r,1,nrhs}];
     Do [j=i-k; aij=a[[ii-k]]; If [aij==0, Continue[]];
       Do [b[[r,i]]+=aij*x[[r,j]]; b[[r,j]]+=aij*x[[r,i]], {r,1,nrhs}],
     {k,1,m}],
  {i,1,n}];
  Return[b]
 ];
```

This is done by module `SymmSkyMatrixVectorMultiply`, which is listed in Cell 26.6. Its arguments are the skymatrix S and the vector x. The function returns Sx in b.

Module `SymmSkyMatrixColBlockMultiply` implements the multiplication by a block of vectors stored as columns of a rectangular matrix **X**:

$$\mathbf{B} = \mathbf{S}\mathbf{X} \tag{26.18}$$

This module is listed in Cell 26.7. Its arguments are the skymatrix S and the rectangular matrix X. The function returns **SX** in B.

There is a similar module `SymmSkyMatrixRowBlockMultiply`, also listed in Cell 26.7, which postmultiplies a vector block stored as rows.

§26.3.5. *Printing and Mapping

Module `SymmSkyMatrixUpperTrianglePrint`, listed in Cell 26.8, prints a symmetric skymatrix in upper triangle form. Is is invoked as

```
SymmSkyMatrixUpperTrianglePrint[S]
```

where S is the skymatrix to be printed. For an example of use see Cells 262-3.

The print format resembles the configuration depicted in Section 26.1. This kind of print is useful for program development and debugging although of course it should not be attempted with a very large matrix.[2]

There is a similar module called `SymmSkyMatrixLowerTrianglePrint`, which displays the skymatrix entries in lower triangular form. This module is also listed in Cell 26.8.

Sometimes one is not interested in the actual values of the skymatrix entries but only on how the skyline template looks like. Such displays, called *maps*, can be done with just one symbol per entry. Module `SymmSkyMatrixUpperTriangleMap`, listed in Cell 26.9, produces a map of its argument. It is invoked as

```
SymmSkyMatrixUpperTriangleMap[S]
```

The entries within the skyline template are displayed by symbols +, − and 0, depending on whether the value is positive, negative or zero, respectively. Entries outside the skyline template are blank.

As in the case of the print module, there is module `SymmSkyMatrixLowerTriangleMap` which is also listed in Cell 26.9.

§26.3.6. *Reconstruction of SkyMatrix from Factors

In testing factorization and solving modules it is convenient to have modules that perform the "inverse process" of the factorization. More specifically, suppose that $\mathbf{U}$, $\mathbf{D}$ (or $\mathbf{D}^{-1}$) are given, and the problem is to reconstruct the skymatrix that have them as factors:

$$\mathbf{S} = \mathbf{L}\mathbf{D}\mathbf{U}, \qquad \text{or} \quad \mathbf{S} = \mathbf{L}\mathbf{D}^{-1}\mathbf{U} \tag{26.19}$$

in which $\mathbf{L} = \mathbf{U}^T$. Modules `SymmSkyMatrixLDUReconstruction` and `SymmSkyMatrixLDinvUReconstruction` perform those operations. These modules are listed in Cell 26.10. Their argument is a factored form of S: $\mathbf{U}$ and $\mathbf{D}$ in the first case, and $\mathbf{U}$ and $\mathbf{D}^{-1}$ in the second case.

---

[2] Computer oriented readers may notice that the code for the printing routine is substantially more complex than those of the computational modules. This is primarily due to the inadequacies of `Mathematica` in handling tabular format output. The corresponding Fortran or C implementations would be simpler because those languages provide much better control over low-level display.

Cell 26.8 Skymatrix Printing

```
SymmSkyMatrixLowerTrianglePrint[S_]:= Module[
 {p,a,cycle,i,ii,ij,it,j,jj,j1,j2,jref,jbeg,jend,jt,kcmax,kc,kr,m,n,c,t},
 {p,a}=S; n=Dimensions[p][[1]]-1; kcmax=5; jref=0;
 Label[cycle]; Print[" "];
   jbeg=jref+1; jend=Min[jref+kcmax,n]; kc=jend-jref;
   t=Table[" ",{n-jref+1},{kc+1}];
   Do [If [p[[j+1]]>0,c=" ",c="*"];
      t[[1,j-jref+1]]=StringJoin[c,"Col",ToString[PaddedForm[j,3]]],
     {j,jbeg,jend}];  it=1;
   Do [ii=Abs[p[[i+1]]]; m=ii-Abs[p[[i]]]-1; j1=Max[i-m,jbeg];j2=Min[i,jend];
      kr=j2-j1+1; If [kr<=0, Continue[]]; If [p[[i+1]]>0,c=" ",c="*"];
      it++; t[[it,1]]=StringJoin[c,"Row",ToString[PaddedForm[i,3]]];
      jt=j1-jbeg+2; ij=j1+ii-i;
      Do[t[[it,jt++]]=PaddedForm[a[[ij++]]//FortranForm,{7,4}],{j,1,kr}],
    {i,jbeg,n}];
   Print[TableForm[Take[t,it],TableAlignments->{Right,Right},
         TableDirections->{Column,Row},TableSpacing->{0,2}]];
   jref=jend; If[jref<n,Goto[cycle]];
];
SymmSkyMatrixUpperTrianglePrint[S_]:= Module[
 {p,a,cycle,i,ij,it,j,j1,j2,jref,jbeg,jend,kcmax,k,kc,m,n,c,t},
 {p,a}=S; n=Dimensions[p][[1]]-1; kcmax=5; jref=0;
  Label[cycle]; Print[" "];
   jbeg=jref+1; jend=Min[jref+kcmax,n]; kc=jend-jref;
   t=Table[" ",{jend+1},{kc+1}];
   Do [If [p[[j+1]]>0,c=" ",c="*"];
      t[[1,j-jref+1]]=StringJoin[c,"Col",ToString[PaddedForm[j,3]]],
   {j,jbeg,jend}];  it=1;
   Do [it++;  If [p[[i+1]]>0,c=" ",c="*"];
       t[[it,1]]=StringJoin[c,"Row",ToString[PaddedForm[i,3]]]; j=jref;
       Do [j++; If [j<i, Continue[]]; ij=Abs[p[[j+1]]]+i-j;
          If [ij<=Abs[p[[j]]], Continue[]];
          t[[it,k+1]]=PaddedForm[a[[ij]]//FortranForm,{7,4}],
       {k,1,kc}],
   {i,1,jend}];
   Print[TableForm[Take[t,it],TableAlignments->{Right,Right},
         TableDirections->{Column,Row},TableSpacing->{0,2}]];
   jref=jend; If[jref<n,Goto[cycle]];
];

Sr={{0, 1, 3, 6}, {1., 2., 7., 4., 23., 97.}};
SymmSkyMatrixLowerTrianglePrint[Sr];SymmSkyMatrixUpperTrianglePrint[Sr];
```

Cell 26.9 Skymatrix Mapping

```
SymmSkyMatrixLowerTriangleMap[S_]:=Module[
 {p,a,cycle,i,ii,ij,it,itop,j,jj,j1,j2,jref,jbeg,jend,jt,kcmax,kc,kr,m,n,c,t},
 {p,a}=S; n=Dimensions[p][[1]]-1; kcmax=40; jref=0;
 Label[cycle]; Print[" "];
   jbeg=jref+1; jend=Min[jref+kcmax,n]; kc=jend-jref;
   itop=2; If[jend>9,itop=3]; If[jend>99,itop=4]; If[jend>999,itop=5];
   t=Table[" ",{n-jref+itop},{kc+1}]; it=0;
   If [itop>=5, it++; Do [m=Floor[j/1000];
      If[m>0,t[[it,j-jref+1]]=ToString[Mod[m,10]]], {j,jbeg,jend}]];
   If [itop>=4, it++; Do [m=Floor[j/100];
      If[m>0,t[[it,j-jref+1]]=ToString[Mod[m,10]]], {j,jbeg,jend}]];
   If [itop>=3, it++; Do [m=Floor[j/10];
      If[m>0,t[[it,j-jref+1]]=ToString[Mod[m,10]]], {j,jbeg,jend}]];
   it++; Do[t[[it,j-jref+1]]=ToString[Mod[j,10]],{j,jbeg,jend}];
   it++; Do[If[p[[j+1]]<0,t[[it,j-jref+1]]="*"],{j,jbeg,jend}];
   Do [ii=Abs[p[[i+1]]]; m=ii-Abs[p[[i]]]-1; j1=Max[i-m,jbeg];j2=Min[i,jend];
      kr=j2-j1+1; If [kr<=0, Continue[]]; If [p[[i+1]]>0,c=" ",c="*"];
      it++; t[[it,1]]=StringJoin[ToString[PaddedForm[i,2]],c];
      jt=j1-jbeg+2; ij=j1+ii-i;
      Do [ c=" 0"; If[a[[ij]]>0,c=" +"]; If[a[[ij++]]<0,c=" -"];
          t[[it,jt++]]=c, {j,1,kr}],
      {i,jbeg,n}];
   Print[TableForm[Take[t,it],TableAlignments->{Right,Right},
         TableDirections->{Column,Row},TableSpacing->{0,0}]];
   jref=jend; If[jref<n,Goto[cycle]];
];
SymmSkyMatrixUpperTriangleMap[S_]:=Module[
 {p,a,cycle,i,ij,it,itop,j,j1,j2,jref,jbeg,jend,kcmax,k,kc,m,n,c,t},
 {p,a}=S; n=Dimensions[p][[1]]-1; kcmax=40; jref=0;
 Label[cycle]; Print[" "];
   jbeg=jref+1; jend=Min[jref+kcmax,n]; kc=jend-jref;
   itop=2; If[jend>9,itop=3]; If[jend>99,itop=4]; If[jend>999,itop=5];
   t=Table[" ",{jend+itop},{kc+1}]; it=0;
   If [itop>=5, it++; Do [m=Floor[j/1000];
      If[m>0,t[[it,j-jref+1]]=ToString[Mod[m,10]]], {j,jbeg,jend}]];
   If [itop>=4, it++; Do [m=Floor[j/100];
      If[m>0,t[[it,j-jref+1]]=ToString[Mod[m,10]]], {j,jbeg,jend}]];
   If [itop>=3, it++; Do [m=Floor[j/10];
      If[m>0,t[[it,j-jref+1]]=ToString[Mod[m,10]]], {j,jbeg,jend}]];
   it++; Do[t[[it,j-jref+1]]=ToString[Mod[j,10]],{j,jbeg,jend}];
   it++; Do[If[p[[j+1]]<0,t[[it,j-jref+1]]="*"],{j,jbeg,jend}];
   Do [it++;  If [p[[i+1]]>0,c=" ",c="*"];
       t[[it,1]]=StringJoin[ToString[PaddedForm[i,2]],c]; j=jref;
       Do [j++; If [j<i, Continue[]]; ij=Abs[p[[j+1]]]+i-j;
          If [ij<=Abs[p[[j]]], Continue[]]; c=" 0";
          If[a[[ij]]>0,c=" +"]; If[a[[ij++]]<0,c=" -"]; t[[it,k+1]]=c,
         {k,1,kc}],
   {i,1,jend}];
   Print[TableForm[Take[t,it],TableAlignments->{Right,Right},
         TableDirections->{Column,Row},TableSpacing->{0,0}]];
   jref=jend; If[jref<n,Goto[cycle]];
];
```

26–17

Cell 26.10 Skymatrix Reconstruction from Factors

```
SymmSkyMatrixLDUReconstruct[S_]:= Module[
  {p,ldu,a,v,n,i,ii,ij,j,jj,jk,jmj,k,m},
  {p,ldu}=S; a=ldu; n=Length[p]-1; v=Table[0,{n}];
  Do [jmj=Abs[p[[j]]]; jj=p[[j+1]]; If [jj<0, Continue[]];
      jk=jj-jmj; v[[jk]]=ldu[[jj]];
     Do [ij=jmj+k; i=j+ij-jj; ii=p[[i+1]]; If [ii<0, v[[k]]=0; Continue[]];
         If [i!=j, v[[k]]=ldu[[ij]]*ldu[[ii]]];
         m=Min[ii-Abs[p[[i]]],k]; a[[ij]]= v[[k]];
         a[[ij]]+=Take[ldu,{ii-m+1,ii-1}].Take[v,{k-m+1,k-1}],
        {k,1,jk}],
     {j,1,n}];  Return[{p,a}];
  ];

SymmSkyMatrixLDinvUReconstruct[S_]:= Module[
  {p,ldu,a,v,n,i,ii,ij,j,jj,jk,jmj,k,m},
  {p,ldu}=S; a=ldu; n=Length[p]-1; v=Table[0,{n}];
  Do [jmj=Abs[p[[j]]]; jj=p[[j+1]]; If [jj<0, Continue[]];
      jk=jj-jmj; v[[jk]]=1/ldu[[jj]];
     Do [ij=jmj+k; i=j+ij-jj; ii=p[[i+1]]; If [ii<0, v[[k]]=0; Continue[]];
         If [i!=j, v[[k]]=ldu[[ij]]/ldu[[ii]]];
         m=Min[ii-Abs[p[[i]]],k]; a[[ij]]= v[[k]];
         a[[ij]]+=Take[ldu,{ii-m+1,ii-1}].Take[v,{k-m+1,k-1}],
        {k,1,jk}],
     {j,1,n}];  Return[{p,a}];
  ];

 p={0,1,2,5,8,9,15}; s={11,22,13,0,33,24,34,44,55,16,0,0,46,56,66};
 S={p,s};
Sr=SymmSkyMatrixLDinvUReconstruct[S]; Print[Sr//InputForm];
Print[SymmSkyMatrixFactor[Sr,0]];
```

§26.3.7. *Miscellaneous Utilities

Finally, Cell 26.11 lists three miscellaneous modules. The most useful one is probably `SymmSkyMatrixConvertToFull`, which converts its skymatrix argument to a fully stored symmetric matrix. This is useful for things like a quick and dirty computation of eigenvalues:

```
Print[Eigenvalues[SymmSkyMatrixConvertToFull[S]]];
```

because *Mathematica* built-in eigensolvers require that the matrix be supplied in full storage form.

Cell 26.11 Miscellaneous Skymatrix Utilities

```
SymmSkyMatrixConvertToFull[S_]:= Module[
  {p,a,aa,n,j,jj,jmj,k},
  {p,a}=S; n=Length[p]-1; aa=Table[0,{n},{n}];
  Do [jmj=Abs[p[[j]]]; jj=Abs[p[[j+1]]]; aa[[j,j]]=a[[jj]];
     Do [aa[[j,j-k]]=aa[[j-k,j]]=a[[jj-k]],{k,1,jj-jmj-1}],
  {j,1,n}];  Return[aa];
  ];

SymmSkyMatrixConvertUnitUpperTriangleToFull[S_]:= Module[
  {p,ldu,aa,n,j,jj,jmj,k},
  {p,ldu}=S; n=Length[p]-1; aa=Table[0,{n},{n}];
  Do [jmj=Abs[p[[j]]]; jj=Abs[p[[j+1]]]; aa[[j,j]]=1;
     Do [aa[[j-k,j]]=ldu[[jj-k]],{k,1,jj-jmj-1}],
  {j,1,n}];  Return[aa];
  ];

SymmSkyMatrixConvertDiagonalToFull[S_]:= Module[
  {p,ldu,aa,n,i,j,jj,jmj,k},
  {p,ldu}=S; n=Length[p]-1; aa=Table[0,{n},{n}];
  Do [jj=Abs[p[[j+1]]]; aa[[j,j]]=ldu[[jj]],
  {j,1,n}];  Return[aa];
  ];
```

Homework Exercises for Chapter 26

Solving FEM Equations

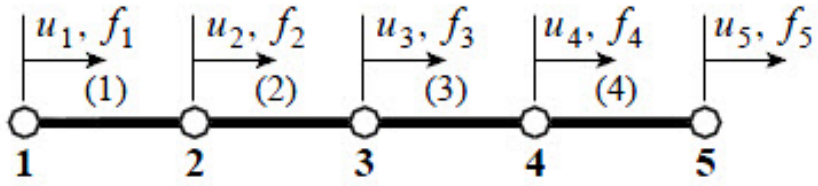

Figure 26.1. Structure for Exercise 26.1

EXERCISE 26.1

[A/C:10+10+15] Consider the 4-element assembly of bar elements shown in Figure 26.1. The only degree of freedom at each node is a translation along x. The element stiffness matrix of each element is

$$\mathbf{K}^{(e)} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{E26.1}$$

(a) Assemble the 5×5 master stiffness matrix $\mathbf{K}$ showing it as a full symmetric matrix. Hint: the diagonal entries are `1, 2, 2, 2, 1`.

(b) Show $\mathbf{K}$ stored as a skyline matrix using a representation like illustrated in (26.15). Hint `p = { 0, 1, 3, 5, 7, 9}`.

(c) Perform the symmetric factorization $\mathbf{K} = \mathbf{L}\mathbf{D}\mathbf{L}^T$ of (26.7), where $\mathbf{K}$ is stored as a full matrix to simplify hand work.[3] Show that the entries of $\mathbf{D}$ are 1, 1, 1, 1 and 0, which mean that $\mathbf{K}$ is singular. Why?

[3] You can do this with *Mathematica* using the function `LUDecomposition`, but hand work is as quick.