RL-897

Manual of the FEM-ATS code used for computing three-dimensional scattering (preliminary)

A. Chatterjee, J. L. Volakis and M. Nurnberger

August 1993

PRELIMINARY

August 19, 1993

MANUAL OF THE FEM-ATS CODE USED FOR COMPUTING THREE-DIMENSIONAL SCATTERING

Arindam Chatterjee, John L. Volakis
and Mike Nurnberger
Radiation Laboratory
Department of Electrical Engineering
and Computer Science
University of Michigan
Ann Arbor MI 48109-2122

For FEMATS-related questions and bug reports, please call either Arindam Chatterjee at (313) 936-0183 or Mike Nurnberger at (313) 764-0502.

1 Introduction

The FEM-ATS program incorporates first order edge-based finite elements and vector absorbing boundary conditions into the scattered field formulation for computation of the scattering from three-dimensional geometries. The code has been validated extensively for a large class of geometries containing inhomogeneities and satisfying transition conditions (see [1] for formulation). The FEMATS code has been optimized to run on the Cray Y-MP and parallelized to run on the Kendall Square Research (KSR) architecture and the Intel iPSC/860.

2 Installation

FEMATS is designed to run on multiple computing platforms to best utilize various machine capabilities. Because of the large amount of time required to run FEMATS, it has been written to run on a supercomputer, while I-DEAS and most of the preprocessing programs only need to be run on a UNIX workstation. Hence there are two sets of source code included on the tape, along with two installation procedures. Also included is a small sample session, starting with the I-DEAS universal file, and ending with the FEMATS output files.

Note: While FEMATS was designed to run on a supercomputer, it may also run on the same workstation that performs the preprocessing, or any other machine (likewise for the preprocessors). Please see Section 4 for more details.

Installation Instructions

1. Place the distribution tape in the tape drive. If the drive is not the default system drive, you will need to find out what device it is.

2. Retrieve the files from the tape to the appropriate directory. If, for example, you wanted to install FEMATS in your home directory, you would say (assuming the tape drive being used is the system default tape drive):

   ```
   cd
   tar xv * .
   ```

   This will place two files in your home directory:

   ```
   femats.reg.tar.Z
   femats.ksr.tar.Z
   ```

   `femats.reg.tar.Z` is a compressed tar file containing source code and test files for the workstation-based portion of FEMATS.

`femats.ksr.tar.Z` is a compressed tar file containing source code and test files for the KSR-based portion of FEMATS.

3. ftp the supercomputer portion of FEMATS to the supercomputer, putting it in the appropriate directory. (If you are going to run both sections of code on the same machine, don't do this...)

4. If `femats.reg.tar.Z` is not in the directory where you want to install FEMATS, then put it there. **Note**: When the files are 'untar'd, a directory named `femats` will be created, and the appropriate files placed in it.

5. Uncompress `femats.reg.tar.Z`: type

   ```
   uncompress femats.reg.tar.Z
   ```

6. Untar `femats.reg.tar`: type

   ```
   tar xvf femats.reg.tar
   ```

7. Change directories to `femats`, and type `install.reg`. This will compile the preprocessors, and place them in the `femats` directory.

8. Follow the same steps for the supercomputer, starting with step 4, and changing `femats.reg` to `femats.ksr` in all cases.

9. If FEMATS will be run in some other directory than that in which it has been installed, the installation directory must be included in the search path. To do this, insert the following line after the path is set in your .cshrc or .profile startup file (csh or sh):

   ```
   set path=($path femats_dir)
   ```

   where `femats_dir` is the full path of the `femats` directory. For example,

   ```
   set path=($path /l/usr/femats) .
   ```

   Then type `rehash` to ensure the new path takes effect. If any difficulties are encountered, ask your system/site administrator.

10. If any problems occur, don't hesitate to look in the scripts—they are quite simple, and there may be some machine or OS version dependencies that were missed...

3 Data Generation

The computation of scattering from a specific geometry with FEMATS is a multi-stage process, as is shown in Figure 1. Once the geometric parameters of the target are known, a solid model is constructed in the Solid Modeling family of SDRC I-DEAS, a commercial CAD/CAE/CAM software package.

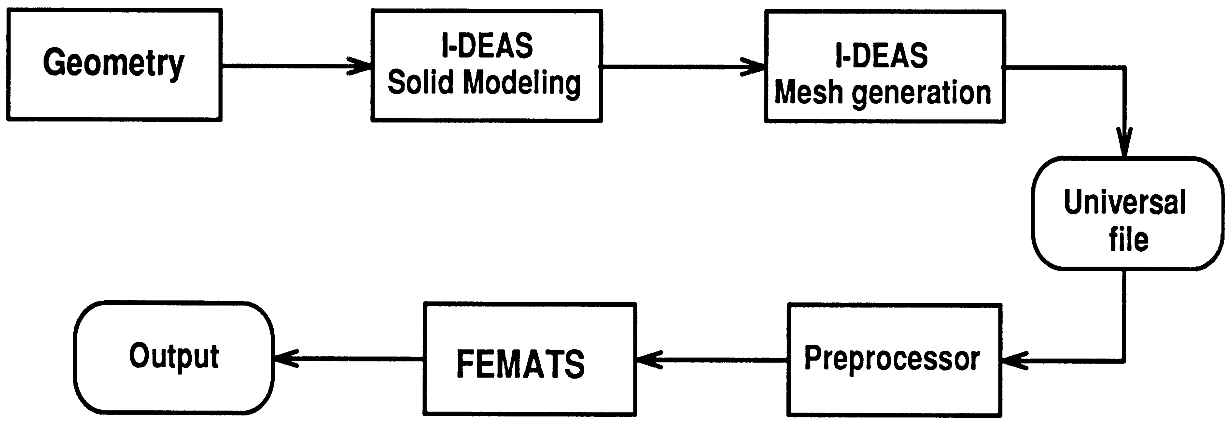

Figure 1: Stages involved in scattering computation from arbitrary 3D geometries

The solid model is then exported to the Finite Element Modeling and Analysis family of I-DEAS, and the nodes and elements necessary for the scattering analysis are generated. This data is written to a output file, called a Universal file, and operated on by several preprocessors, generating the necessary input files for FEMATS.

The process of object modeling and mesh generation is an art, *not* a science. Hence, it cannot be taught, or demonstrated—it must be learned through experience. Hence, this manual is not by any means an I-DEAS FE mesh generation manual. In fact, it assumes (and requires) a working knowledge of, and familiarity with, the I-DEAS Solid Modeling and Finite Element Analysis families of tasks.

3.1 Solid Modeling

Once the geometry of the target is specified, it is constructed using the I-DEAS Solid Modeling Family of tasks. There is a tendency to downplay the importance of the solid model, and treat it only as a stepping stone towards a final product. However, the solid model is the framework for the finite element mesh, and as such, has a direct bearing on the quality of the mesh. Because of this, it is wise to keep the mesh generation problem as simple as possible. This helps to ensure a better mesh, and a more accurate answer.

In general, the object or body being meshed will contain various planes of symmetry. It is nearly always advisable to take advantage of whatever symmetry is available, as doing so will greatly reduce the amount of time necessary to generate the mesh. In fact, the geometry may require such subdivision to make meshing possible. For more details about the creation of the solid model for FEMATS, please see the I-DEAS Solid Modeling User's Guide.

Note: When creating the Solid Model, FEMATS requires the dimensions to be in wavelengths.

3.2 Mesh Generation

I-DEAS generates the finite element mesh by creating mesh areas on surfaces, and then combining these mesh areas into mesh volumes ($2\frac{1}{2}$-D mesh generation). Each mesh volume is then filled with the chosen element type. When the solid model is imported into the Finite Element Modeling and Analysis Family of I-DEAS, these mesh areas and mesh volumes are automatically created. Generally, however, I-DEAS does not choose the correct element order (linear vs. parabolic), and the mesh volumes must be modified to reflect the correct element order. Even if the guidelines mentioned in Section 3.1 are followed, the mesh areas that are auto-created by I-DEAS can become quite complex. It is then prudent to break the mesh volume into smaller, more manageable mesh volumes. For more details on mesh creation, please see the I-DEAS Finite Element Modeling User's Guide.

3.3 Modifying material property labels

After all the mesh volumes have been created , the material property labels of each need to be modified according to the type of material each mesh volume contains. The elements in the volume between the target and the outer boundary usually have a material property label of 1. If the target contains a dielectric-filled volume, the material property labels of the elements in that volume should be 2—actually any integer greater than 1. In this way, the code can accommodate up to 9 different material fillings. If the geometry requires more than 9 different materials, the dimensions of the vectors ϵ and μ should be modified (wherever they appear) to reflect the necessary number of materials.

Please see Section 6.1 for more details.

3.4 Type of meshing

The global element length also needs to be specified (usually .075–.085 units); finer meshing can be done in regions with rapidly changing fields or large curvatures by specifying the local element length or curvature-based size parameters. The geometry is then *free*-meshed using the I-DEAS Mesh Creation Task. It is essential to use *free* meshing and not *mapped* meshing, since the latter maps the mesh volume into a rectangular box and back, thus distorting the elements. No such distortion occurs in space when an electromagnetic wave travels through it; a *mapped* mesh, therefore, alters the physics of the problem and leads to inaccuracies in the final result.

Please see Section 6.1 for more details.

3.5 Grouping nodes

The finite element method essentially solves a boundary value problem; thus, it is crucial to identify surfaces or surface edges on which the boundary conditions are imposed. In the current version of FEMATS, this is carried out by grouping the nodes which lie on the surfaces where the boundary conditions are imposed. If the surface nodes coincide with a perfect electric conductor, the group is labeled C, if the nodes lie on a resistive sheet, the group is labeled R, and so on. Detailed information about node grouping is given in the Appendix.

Care must be taken when grouping surfaces that intersect, since edges connecting two such nodes may not lie on the surface at all. For example, suppose three surfaces intersect at the corner of a cube. If the nodes on each of these surfaces are not grouped separately, the processing program will generate 'surface' edges which actually do not lie on the surface. Another anomaly may arise when the surfaces are separated by a single element. This is due to the fact that the processing program considers an edge to lie on the surface if two nodes in the group connect to form an edge. As a rule of thumb, it is best to group each surface separately. They may be grouped together only when the user is certain that spurious surface edges will not be created by the processing program.

Please see Section 6.1 for more details.

3.6 Universal file

The mesh information obtained from I-DEAS is then written to an ASCII file called the Universal file. The Universal file has a specific format for identifying the nodes, elements and groups which can be obtained from the I-DEAS User's guide.

It should be noted that only the FE entities and Groups need to be included into the Universal file. Also, while this discussion has dealt primarily with I-DEAS, any mesh generator that writes a Universal file will work just fine...

4 Preprocessing

The necessary preprocessing is performed by a number of smaller programs that operate on the Universal file generated by I-DEAS. Because FEMATS is designed to be run on a supercomputer, and I-DEAS and the preprocessors are generally run on a workstation of some sort, some of the preprocessing is performed on the workstation, and some of it on the supercomputer. In both cases, a script runs the necessary preprocessors, and presents the user with the necessary FEMATS input data files. (Note that while FEMATS has been designed to run on a supercomputer, it can peacefully co-exist with the preprocessors, etc. on the workstation. However, if FEMATS is to be run on the workstation, certain variable types must be changed to reflect the decreased precision inherent to most workstations.) To run the script that runs the preprocessors, type

```
    femats.reg file.unv
```

where `file.unv` is the name of the universal file containing the mesh information. The `femats.reg` script extracts the necessary information from each preprocessor, and terminates with instructions informing the user of which files need to be transferred to the supercomputer for further preprocessing.

After the appropriate files have been transferred to the supercomputer, a script is also run there to finish the preprocessing. To run this script, type

```
femats.ksr file
```

where `file` is the name of the original universal file, *without* the ending (.unv). This script will present the user with the necessary input files for FEMATS, along with a list of numbers required for input by FEMATS.

Note: For efficiency, the dimensioning of the arrays in the preprocessors may be changed to reflect the size of the problem. On the UNIX workstation, the relevant dimension statements are contained in the file

```
(path)/femats/src/reg/parmvl
```

For the re-dimensioning to take effect, re-run the `install.reg` script file. Similarly, on the supercomputer, the relevant dimension statements are in the file

```
(path)/femats/src/ksr/fem_data.h
```

For these changes to take effect, the `install.ksr` script must be re-run.

5 Running FEMATS

5.1 KSR-specific runtime information

Before executing FEMATS, the user must inform the operating system of the required number of processors. To do this, type

```
allocate_cells -A ##
```

where `##` is the requested number of cells. This command starts a new shell, and gives that shell control of `##` cells. Because of this, it is important to exit the shell when FEMATS finishes, so that others may use the processor cells.

After the new shell is running, the user must let the operating system know how many total threads it may run on the allocated cells by typing

```
setenv PL_NUM_THREADS ##
```

where `##` is the same as in the previous command. FEMATS may now be run safely.

To execute FEMATS, type

```
femats
```

at the shell prompt. FEMATS will then prompt the user for the necessary input (see following documentation).

FEMATS will also accept commands from standard input. This is necessary for use in batch mode. The commands from above are inserted into a shell script, along with the following statement:

```
femats < input.dat
```

where `input.dat` is a text file containing the input, just as it would be entered at the keyboard when running FEMATS interactively. The example in Section 6.3 shows this format.

If any problems arise, consult the KSR manuals, and the system administrator. The above steps assume that everything works the way it is supposed to.

5.2 FEMATS input documentation

It is faster to read the input data from a file; however, for the first-time user, interactive input provides more insight (see Section 5.1).

```
Number of edges
```

Input the no. of edges obtained from **proc**.

```
No. of elements with surface edges on 1) pec 2) r-card 3) ibc
4) dielectric 5) outer boundary 6) outer surface of scatterer
```

Enter the no. of elements with surface edges on the various materials as obtained from **proc.f**.

```
If inhomogeneous, enter 0
```

Enter 0 as long as there are two or more material property labels (see Appendix) present in the geometry. This holds for r-cards as well, since the top and bottom elements on a r-card have different material property labels.

```
Number of distinct dielectric materials
```

Enter the no. of distinct material property labels. Note that material property label 1 is free-space by default. If the mesh designates material property label 1 to anything other than free-space, the program won't run.

```
constitutive relative parameters for region ,i
```

Input the epsilon and mu of the dielectric in that order. For a r-card whose top and bottom surface is free-space, enter ϵ_r and μ_r of free space, i.e., unity.

```
If resistive card inside geometry enter 1
```

```
Number of different r-cards
```

Input the no.of r-cards having different resistivity values for the geometry.

```
Input: a) Material property label on top surface of card
       b) Material property label on bottom surface of card
       c) Normalized resistivity
```

The mesh must be constructed in such a way that the material property labels on the top and bottom surfaces of the R-card are different.

```
If impedance sheet inside geometry enter 1

Input: a) Material property label on top surface of impedance sheet
       b) Normalized impedance
```

Most of the data entered until now has been related to the geometry. The data entered from this point will be related to the iteration count, number of look angles, etc.

```
Tolerance, maximum iterations
```

The tolerance of the residual is usually kept between 0.001 and 0.0005. This is 0.1%–0.05% of the solution norm. Max. no. of iterations is determined by trial and error. A typical value for PEC targets is $N/100$ for $N > 25000$ and $N/120$ for $N > 75000$. The largest problem run to date contained 93000 unknowns and converged, on the average, in 800 iterations. The code uses a diagonally preconditioned biconjugate gradient method to solve the system, so the residual error will jump to abnormal values quite frequently.

```
1) Bistatic  2) Backscatter
```

Enter 1 for bistatic pattern, 2 for backscatter

```
All angle values should be integers

Bistatic
--------
Angle of incidence: theta,phi
Fix 1) phi 2) theta to specified angle
Angle of observation: start,end,increment
Polarisation angle: alpha=0(H_z=0); alpha=90(E_z=0)
```

In order to fix ϕ to 90° (say), the input should look like
1 90
To fix θ to 90° (say), the input should be
2 90

```
Backscatter
-----------
Fix 1)phi 2)theta to specified angle
Angle of incidence: start,end,increment
Polarisation angle: alpha=0(H_z=0); alpha=90(E_z=0)

Enter 1 for spherical outer boundary; 2 otherwise
```

The code works for a spherical termination or terminations having flat outer faces. The sphere should be centered at the origin.

6 Appendix

6.1 Stipulations for mesh generation

- the region surrounding the scatterer should have a material property number label of 1, i.e., the least possible value.

- for a surface draped by a resistive card, it is essential to differentiate the top surface from the bottom surface. The only way the program can discern this from the available data is by checking the material property number labels of the elements on the top and bottom surfaces. The material property number label of the top surface must be different from that of the bottom surface.

- when meshing a mesh-volume filled with a dielectric having a certain permeability and permittivity, the material property label number should be different from that of surrounding space.

- when grouping surface nodes, the group labels should start with a

  - **C** if the nodes lie on a perfect electrical conductor
  - **R** if the nodes lie on a resistive card
  - **D** if the nodes lie on a dielectric
  - **A** if the nodes lie in free space (i.e. on the mesh termination boundary)
  - **O** if the nodes lie on the outer surface of the scatterer

  The above order (C, R, D, A, O) *must* be maintained when grouping nodes.

- Nodes on the interfaces of materials having different constitutive parameters must be grouped.

6.2 Code Theory of Operation

6.2.1 proc.f

proc.f converts the mesh information stored in the Universal file into a more usable form for analysis by FEMATS. It first reads in the nodal co-ordinates, nodal connectivity and the grouped nodes from the Universal file. Since FEMATS uses edge-based shape functions, the edges and the nodes connecting them need to be identified. Because each edge is shared by more than one element, care must be taken so that the same edge is not counted more than once. A comparison of the connecting nodes must therefore be made to identify the old edges, and create the new ones. This can be a computationally intensive task if a brute force approach is taken, especially if the problem size is very large. It is necessary to use an algorithm that would scale at most linearly with the number of nodes or edges, i.e. the number of comparisons required for identifying old or new edges should be an $O(N)$ process.

In order to realize this requirement, the ITPACK scheme [2] is utilized to store the node connectivity information. The ITPACK scheme is attractive because the number of comparisons required while augmenting the connectivity matrix depends only on the locality of the corresponding node and not on the total number of nodes or edges. In the ITPACK storage scheme, the number of rows of the connectivity matrix is equal to the number of nodes and the number of columns equals the maximum number of nodes connected to a particular node. However, this approach wastes space when the number of connecting nodes varies widely, so a modified ITPACK format is used—the number of columns in the connectivity matrix now equals the average number of nodes connected to a particular node, and the number of rows is slightly more than the total number of nodes. The storage requirement for such a matrix is usually $1.1N_n \times 16$ integers, where N_n equals the number of nodes.

After generating the edges, FEMATS uses the same storage scheme for finding the surface edges and elements from the grouped nodes. These surface edges are then sorted in ascending order by element number for the various materials and boundaries on which they lie. All components of the code are extremely fast, with the slowest being the sorting routine.

The output files from **proc.f** are

- **enode**
  contains co-ordinates of all the nodes in the geometry.

- **eglob**
  contains the edges making up each element.

- **edge**
  contains the nodes making up each edge.

- **esurfed**
  contains the element number, node numbers nad corresponding edge numbers of the on-surface edges.

- **otpt**
  contains the number of edges in the geometry and the number of elements with surface edges on the PEC, R-card, dielectric, outer boundary and outer surface of scatterer.

Required storage is about $18N$ real Words, where N is the number of unknowns and is equal to the number of edges making up the mesh.

6.2.2 count.f

count.f asks for the number of edges in the geometry and generates **cntr** as the output. **cntr** contains the number of non-zero entries per row for the finite element system. The number usually varies from 9 to 31 for a typical system. Required storage is about $13N$ real Words, where N again denotes the number of unknowns.

6.2.3 fem.f

This is the main program (FEMATS) which computes backscatter or bistatic patterns after reading in the mesh files created by **proc.f** and **count.f**. Parameters like the number of edges, number of surface elements, type of pattern, etc. can be read in interactively or from a file. The backscatter or bistatic pattern is returned in a separate file. If the code fails to run for some reason, a list of errors is returned in the error file. The flow of control of FEMATS is given in Figure 2. The formulation for the methodology is given in [2].

Input files: The input files containing the mesh information and parameters for running the probelm are read in, usually in binary format. The ASCII format is quite slow for most machines and prohibitively slow on the KSR1. A small program usually converts the mesh files from ASCII to binary.

Processing data: Some preliminary processing is done to find the radius of the outer boundary if a spherical mesh termination scheme is used.

FE matrix generation/assembly: The finite element matrix generation is done on an element-by-element basis. The elemental matrix is first computed, and then assembled into the global sparse matrix. The assembly is simplified since the number of non-zero entries per row of the matrix is known *apriori* and the order of the entries is not important. The non-zero entries of the final sparse matrix are stored in a long complex vector, the corresponding column numbers are stored in an integer vector, and the location of the first non-zero entry for each row is stored in another integer vector. This is the well-known Compressed Sparse Row (CSR) format used in public domain software packages like SLAP and SPARSPAK. The coefficient matrix is not a function of the angle of incidence.

The code also uses a simple diagonal preconditioner for speeding up the iterative process. Other complicated preconditioning strategies are also available. However, except for the block ILU preconditioner, none of them com-

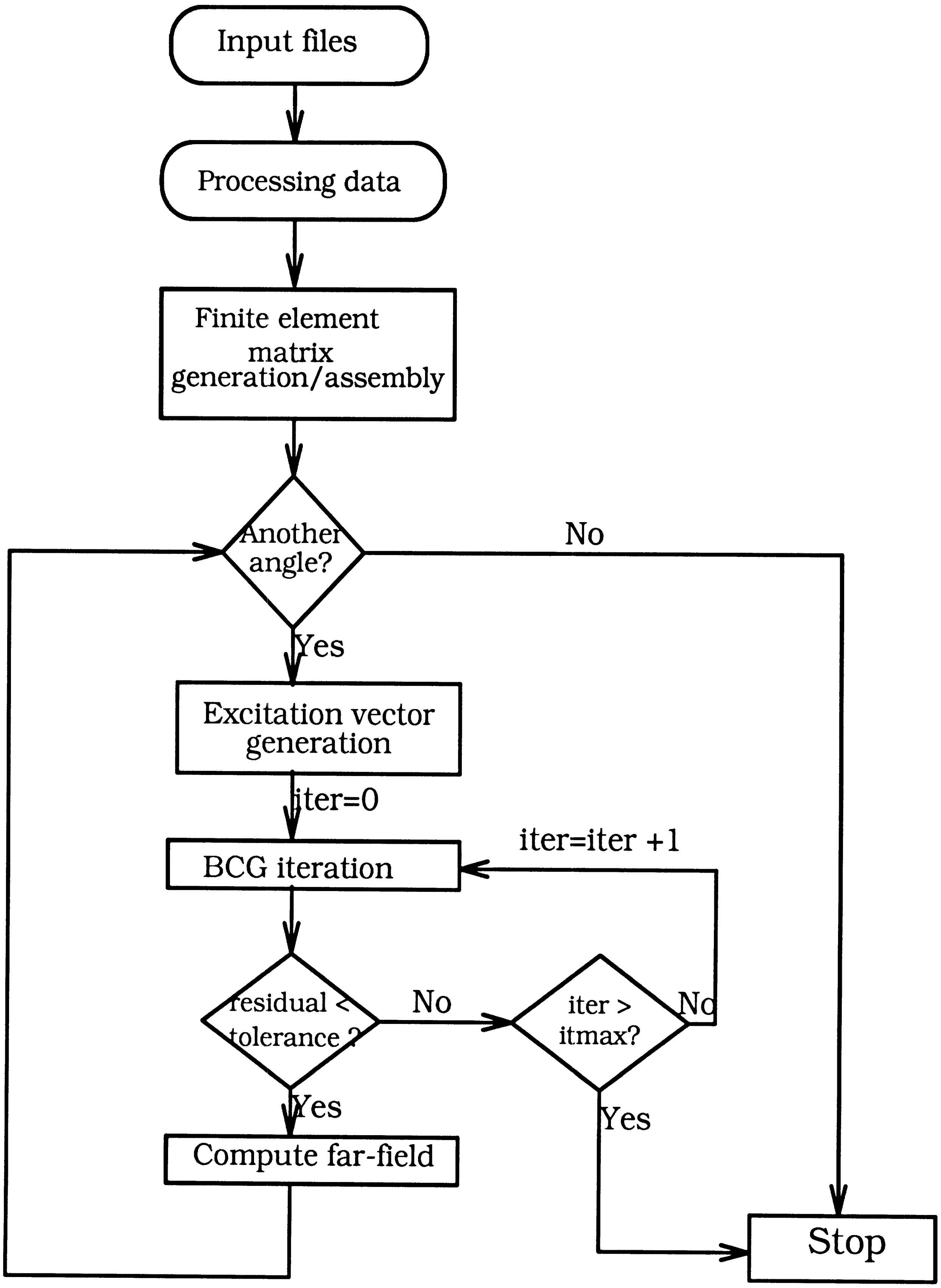

Figure 2: Flowchart for FEMATS

pare favourably with the point diagonal preconditioner in terms of solution time.

Excitation vector generation: The excitation vector generation is not very cpu-intensive, since the vectors are always quite sparse. It is a function of the angle of incidence.

BCG iteration: The biconjugate gradient (BCG) algorithm is used with preconditioning to solve the sparse, symmetric system of linear equations. Each iteration of the algorithm involves 1 sparse-matrix vector product, 3 vector updates and 3 inner products. The norm of the residual vector is computed after every iteration to check for convergence. Reliable results have been obtained by setting the convergence criterion to be

$$||r^k|| < 0.001 * ||b||$$

where r^k is the residual vector after the kth iteration and b is the excitation vector.

Far-field evaluation: The far-field is evaluated by integrating the near-zone fields over a closed surface using the Stratton-Chu integral equation. The surface is usually taken to be very close to or on the body itself to achieve maximum accuracy.

Storage required for the code is at present $36N$ complex Words, where N is the number of unknowns. The storage can be cut by 40% if only the symmetric upper triangular part of the object matrix is stored; the code, however, slows down significantly.

6.2.4 Subroutine functions in fem.f

basis.f

Calculates the two constant vectors of the bases for the finite element discretization as well as the element volume.

calc.f

Computes the volume integral for the finite element discretization analytically.

ccross.f

Takes the cross product of two complex vectors.

cdot.f

Takes the inner product of two vectors.

comput.f

Calculates the basis functions at the mid-point of each edge.

cross.f

Takes the cross product of two real vectors.

crux.f
Computes the element matrix from the volume integral.

cruxd.f
Imposes the boundary condition for dielectric volumes and generates the corresponding excitation vector.

dist.f
Calculates the distance between two points in space.

dot.f
Computes the dot product of two real vectors.

exchg.f
Exchanges one variable with another.

f1.f
Carries out the volume integration of $W_i \cdot W_j$ analytically.

finc.f
Computes the volume integral for a dielectric volume to be used in the excitation vector.

fu.f
Carries out the surface integration for the absorbing boundary condition employed on the mesh termination boundary.

incc.f
Imposes the boundary condition for a perfect electric conductor. If *iter* is 0, the excitation vector is computed, otherwise changes are made to the element matrix.

incd.f
Imposes the boundary condition for a dielectric surface.

incr.f
Imposes the boundary condition for a resistive card.

mult.f
Carries out the sparse matrix-vector multiplication.

norm2d.f
Computes the element normal for a 2D geometry.

norma.f
Computes the element normal for a surface element.

ord.f
Identifies the global nodes and edges in the local context.

sort.f
Sorts the edges in a element according to a specific numbering scheme.

surfint.f
Imposes the absorbing boundary condition on the mesh termination boundary.

value.f
Computes the far-field using the Stratton-Chu integral equation.

volume.f
Calculates the element volume.

6.2.5 References

1. A. Chatterjee, J.M. Jin and J.L. Volakis, "Application of edge-based finite elements and ABCs to 3-D scattering," *IEEE Trans. Antennas Propagat.*, vol. 41, pp. 221–26, February 1993.

2. D.R. Kincaid and T.C. Oppe, "ITPACK on supercomputers," *Numerical Methods, Lecture Notes in Mathematics*, vol. 1005, pp. 151–61, Springer, Berlin, 1982.

6.3 Example FEMATS Run

A perfectly conducting cylindrical inlet was run on the KSR1 machine. The geometry was enclosed by a rectangular outer boundary and the backscatter pattern was sought for $\theta = 0°$–$90°$ aaand $\alpha = 90°$. The problem had 213,832 unknowns and a diagonally preconditioned BCG solver was used.

Input file:

```
213832
13656 0 0 10704 7704
0
2
(1.,0.) (1.,0.)
0
eg
.001 10000
2
1 90
0 90 5
90
2
```

Output file:

```
Number of threads =   56
Backscatter pattern will be computed
Polarisation angle=   90
Incident angle from   0  to   90
in steps of   5
Sweep through theta ; phi=   90
************************************************
               Problem size
Number of nodes =   32453
Number of elements =   176048
Number of edges/unknowns =   213832
************************************************
Finished reading in data
Outer boundary shape is
Rectangular
Time spent for unformatted I/O =     1.0519631999999999  secs
Time spent for I/O =     1.0754451999999999  seconds
 10000
Generating finite element matrix
Generated finite element matrix
No.\ of non-zeros =   3414496
Average no.\ of non-zeros =   15
Total time spent=     25.834841199999996  secs
Time spent in loop=     24.807681599999999  secs
Generated diagonal preconditioner
Time for preconditioner =    1.1077387999999999  secs
************************************************
   90.000000000000000   0.    90.000000000000000
 (   23874.682292021858,  0.)
Time spent in gen. soln. vector =     19.514778000000000  secs
Convergence achieved in   4397  iterations
Time spent in   4397  iterations =     584.26175160000003  secs
Backscatter =     13.132205300654515

   90.000000000000000   5.0000000000000009   90.000000000000000
 (   23874.649727818964,  0.)
Time spent in gen. soln. vector =     20.677881599999978  secs
Convergence achieved in   1878  iterations
Time spent in   1878  iterations =     248.19567480000001  secs
Backscatter =     12.346785646714235

   90.000000000000000   10.000000000000002   90.000000000000000
 (   23874.552995090075,  0.)
Time spent in gen. soln. vector =     20.646470399999998  secs
Convergence achieved in   6561  iterations
Time spent in   6561  iterations =     866.82569879999994  secs
```

```
Backscatter =     10.984172458513957

   90.000000000000000    15.000000000000002    90.000000000000000
 (   23874.394947730074,  0.)
Time spent in gen. soln. vector =     20.678055200000017  secs
Convergence achieved in   6112  iterations
Time spent in   6112  iterations =     807.46988880000004  secs
Backscatter =     8.0404387189358921

   90.000000000000000    20.000000000000004    90.000000000000000
 (   23874.180257205706,  0.)
Time spent in gen. soln. vector =     20.636231199999656  secs
Convergence achieved in   6430  iterations
Time spent in   6430  iterations =     849.45422520000011  secs
Backscatter =     4.5520666697643231

   90.000000000000000    25.000000000000000    90.000000000000000
 (   23873.915286302414,  0.)
Time spent in gen. soln. vector =     20.640396400000100  secs
Convergence achieved in   6303  iterations
Time spent in   6303  iterations =     832.76715800000011  secs
Backscatter =     1.8286943794696267

   90.000000000000000    30.000000000000004    90.000000000000000
 (   23873.607915069253,  0.)
Time spent in gen. soln. vector =     20.624299199999768  secs
Convergence achieved in   4543  iterations
Time spent in   4543  iterations =     600.40641159999996  secs
Backscatter =     2.1870075445461543

   90.000000000000000    35.000000000000007    90.000000000000000
 (   23873.267321948337,  0.)
Time spent in gen. soln. vector =     20.654717999999775  secs
Convergence achieved in   6015  iterations
Time spent in   6015  iterations =     794.75217840000005  secs
Backscatter =     2.9538913638132449

   90.000000000000000    40.000000000000007    90.000000000000000
 (   23872.903724276915,  0.)
Time spent in gen. soln. vector =     20.636641200000668  secs
Convergence achieved in   6215  iterations
Time spent in   6215  iterations =     821.00750359999984  secs
Backscatter =     4.3094572190189613

   90.000000000000000    45.000000000000000    90.000000000000000
 (   23872.528083709803,  0.)
```

```
Time spent in gen. soln. vector =     20.691929999999957  secs
Convergence achieved in   3213  iterations
Time spent in   3213  iterations =     424.64956839999923  secs
Backscatter =     4.0201582083152863

   90.000000000000000    50.000000000000000    90.000000000000000
 (   23872.151783594894,  0.)
Time spent in gen. soln. vector =     20.679000399999495  secs
Convergence achieved in   3196  iterations
Time spent in   3196  iterations =     422.23548159999973  secs
Backscatter =     6.0710219487833603

   90.000000000000000    55.000000000000000    90.000000000000000
 (   23871.786286832561,  0.)
Time spent in gen. soln. vector =     20.649760399999650  secs
Convergence achieved in   5037  iterations
Time spent in   5037  iterations =     665.48518600000079  secs
Backscatter =     6.0386806852857617

   90.000000000000000    60.000000000000007    90.000000000000000
 (   23871.442784137245,  0.)
Time spent in gen. soln. vector =     20.630159599999388  secs
Convergence achieved in   5096  iterations
Time spent in   5096  iterations =     673.35249359999943  secs
Backscatter =     2.9883959797387871

   90.000000000000000    65.000000000000000    90.000000000000000
 (   23871.131843785708,  0.)
Time spent in gen. soln. vector =     20.649902799999836  secs
Convergence achieved in   5096  iterations
Time spent in   5096  iterations =     673.46241600000030  secs
Backscatter =     4.1405354898674034

   90.000000000000000    70.000000000000014    90.000000000000000
 (   23870.863074712492,  0.)
Time spent in gen. soln. vector =     20.627787199999148  secs
Convergence achieved in   4893  iterations
Time spent in   4893  iterations =     646.38884879999932  secs
Backscatter =     3.4527505854226375

   90.000000000000000    75.000000000000014    90.000000000000000
 (   23870.644815085496,  0.)
Time spent in gen. soln. vector =     20.643494799998734  secs
Convergence achieved in   3459  iterations
Time spent in   3459  iterations =     457.10817280000083  secs
Backscatter =   -0.42219432577245863
```

```
   90.000000000000000    80.000000000000014    90.000000000000000
 (   23870.483858181193,  0.)
Time spent in gen. soln. vector =     20.630811200000608  secs
Convergence achieved in   4885  iterations
Time spent in   4885  iterations =     645.37610479999967  secs
Backscatter =     4.9808095185448629

   90.000000000000000    85.000000000000000    90.000000000000000
 (   23870.385226404902,  0.)
Time spent in gen. soln. vector =     20.668981599999825  secs
Convergence achieved in   1924  iterations
Time spent in   1924  iterations =     254.26892960000077  secs
Backscatter =     8.0244739010739181

   90.000000000000000    90.000000000000000    90.000000000000000
 (   23870.352002710646,  0.)
Time spent in gen. soln. vector =     20.646247599999697  secs
Convergence achieved in   1747  iterations
Time spent in   1747  iterations =     231.11860320000051  secs
Backscatter =     8.7459369327904284

Total time =     11978.956639599999  seconds
```