SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 2, 21–28 (1972)

MAPLIB–A Data Bank of FORTRAN Functions describing Material Properties

U. SCHUMANN

Institut für Reaktorentwicklung, Gesellschaft für Kernforschung mbH., Karlsruhe, Deutschland

SUMMARY

MAPLIB is a program system which is able to incorporate the values of the properties of many materials in a form suitable for computer-aided design. Thus, MAPLIB is a data bank whose customers are not men but computer programs with their much higher scanning rate. Therefore in MAPLIB the data are not stored in tables as usual but in algorithms implemented as FORTRAN functions.

Such a program system should fulfil a number of conditions related to standardization, flexibility, security, effectivity, transparency and compatibility. In this paper these conditions are discussed in some generality and it is shown how they are implemented in MAPLIB.

KEY WORDS Computer-aided design Documentation Error control Material properties Standardization Storage and retrieval

INTRODUCTION

In order to solve engineering design and design evaluation problems on a computer, the engineer has to provide a program representing the model of the system he wants to simulate. In almost all practical cases, this program needs information about the values of certain properties of certain materials and their dependence upon parameters like temperature, pressure, etc.

Thus, the engineer is often not interested in the data himself but only for his program. Therefore the engineer needs a data bank which furnishes the data in a form suitable for his program.

In this paper it is shown what qualities such a data bank should have and how they are accomplished in MAPLIB (*Ma*terial *P*roperties Program *Lib*rary), a data bank set up in Karlsruhe. Some of the conditions discussed are postulated by Hoare.[1]

GENERAL DEMANDS AND THEIR REALIZATION IN MAPLIB

Standardization

For a correct and unequivocal retrieval of the data, a number of conventions has to be defined with respect to

(1) The form of data storage.
(2) The naming of data.
(3) The documentation of the data.

MAPLIB consists of a multitude of FORTRAN functions. For each property of a material in MAPLIB a function is integrated, which computes the value of the material

Received 23 July 1971
Revised 6 September 1971

property according to the values of its appropriate parameters as defined by the user's program. All data are defined in one consistent system of units—the SI–units (m, kg, s, A,°K, cd).[2]

Properties and materials are identified by unique symbols. Property symbols consists of two characters (e.g. enthalphy: EH), material symbols consists of up to four characters (e.g. liquid sodium: NAL). For the symbols implemented in MAPLIB up to now, refer to Table I and II. The concatenation of a property name and a material name is considered as the particular material property name.

Table I. Symbols of materials implemented up to the present time

AIRV	Air
B4C	Boron-4-carbide
CO2V	Carbon dioxide
H2O	Water, general
H2OL	Water, liquid
H2OV	Steam
HEV	Helium gas
NA	Sodium, general
NAL	Sodium, liquid
NAV	Sodium, vapour
NALS	Saturated liquid sodium
NAVS	Saturated sodium vapour
PUO	Plutonium oxide
UO	Uranium oxide
UPUO	Uranium–plutonium mixed oxide
4961	4961 Grade steel
4981	4981 Grade steel
4988	4988 Grade steel

Table II. Symbols of properties implemented up to the present time

VP	Saturated vapour pressure
VT	Saturated vapour temperature
FT	Melting temperature
FH	Heat of fusion
RH	Heat of recrystallization
VH	Evaporation enthalpy
VS	Evaporation entropy
EH	Enthalpy
ES	Entropy
RO	Density
VO	Specific volume
CP	Specific heat at constant pressure
CV	Specific heat at constant volume
PR	Prandtl number
WL	Thermal conductivity
ZD	Viscosity, dynamic
ZK	Viscosity, kinematic
SB	Rupture strength
SD	Elongation strength
SF	Yield strength
EM	Modulus of elasticity

The function for the property CP and the material NAL (as an example) is named CPNAL.

The documentation is standardized in MAPLIB too. MAPLIB expects that new functions are accompanied by information related to

(1) The author.
(2) The date of the creation or last change.
(3) The reference to the origin of the data.
(4) The accuracy, and so on.

This information is added to the function on comment cards, which contain the literal C in the first column (FORTRAN rule) immediately followed by a character, which identifies the information following on the card. Refer to Table III for these identifications. Figure 1 shows such a function together with the relevant information. This convention makes each function 'self-describing'.

Table III. Information identifications

*CN*AME	Name of routine
*CA*UTHOR	Name(s) of author(s)
*CD*ATE	Date of creation or last change
*CL*ITERATUR	References
*CP*ARAMETER	Explanation of parameters
*C$M*ATERIAL	Explanation of material
*C$P*ROPERTY	Explanation of property
*CB*ESCHREIBUNG	General description
*CI*NPUT	Input description
*CO*UTPUT	Output description
*CF*ILES	Specification of external data sets
*CS*UBROUTINES	List of sub-programs needed
*CE*RROR	Error messages
*CR*EGION	Memory storage requested
C	Additional information

Flexibility

The design program should be flexible with respect to both materials and properties for which it can be used

In MAPLIB the flexibility in material and property is achieved by a special retrieval system. The user's program can obtain access to the functions on three different levels. The program will be most flexible when the highest level is used.

On the lowest level the functions are called directly by their name together with their parameters needed: e.g.

```
CPNAL(T, P)
```

In this way the material and the property to be used are fixed at the time the program is written. Alternatively, at the higher levels the property, the material or both can be specified dynamically at program execution time. This is done by providing functions ('master functions') whose names correspond to the standard pattern, with $ replacing the fields that will be supplied at execution time by the first one or two arguments. Thus the following function calls all have identical effect:

```
CPNAL(T, P)
$$NAL('CP', T, P)
CP$$$$('NAL', T, P)
$$$$$$('CP', 'NAL', T, P)
```

```
MAPLIB-UTILITY        MATERIAL  PROPERTY  FUNCTION

                 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
CARD     1       $          FUNCTION VTCO2V(/PN     /)                                    VTC00001$
CARD     2       $C                                                                       VTC00002$
CARD     3       $CN        ***  M A P L I B  ***    FUNCTION                             VTC00003$
CARD     4       $C                                                                       VTC00004$
CARD     5       $CD        19.10.70                                                      VTC00005$
CARD     6       $CA        H. SCHNAUDER  &  W. ZIMMERER                                  VTC00006$
CARD     7       $C$P       SATTDAMPFTEMPERATUR                              K            VTC00007$
CARD     8       $C$M       CO2-GAS                                                       VTC00008$
CARD     9       $CP        PN      1.E5<=5.28E5<=PN<=7.54E6<=7.8E6                       VTC00009$
CARD    10       $CP        DRUCK                                            N/M2         VTC00010$
CARD    11       $CL        H. SCHNAUDER, IRE-NOTIZ NR.26/68, PROGRAMMBESCHREIBUNG NR.182 VTC00011$
CARD    12       $CS        $NUMBR,NUMBR$,RANGE                                           VTC00012$
CARD    13       $          REAL A(5)/2.98426,-6.22982E-3,1.05784E-4,-9.21483E-7,3.72320E-9/ VTC00013$
CARD    14       $C$F                                                                     VTC00014$
CARD    15       $          CALL RANGE(VTCO2V,'VTCO2V',2,5.28E5,7.54E6,1.E5,7.8E6,PN,&999)   VTC00015$
CARD    16       $          TS= -875.186/(ALOG10(PN*0.986923E-5)-4.73009)-273.16           VTC00016$
CARD    17       $          P=PN*1.019716E-5                                              VTC00017$
CARD    18       $          T= 304.2                                                      VTC00018$
CARD    19       $          PK=75.379                                                     VTC00019$
CARD    20       $        6 TT= TS+273.16                                                 VTC00020$
CARD    21       $          X= T-TT                                                       VTC00021$
CARD    22       $          B=0.                                                          VTC00022$
CARD    23       $          XN=1.                                                         VTC00023$
CARD    24       $          DO 300 J=1,5                                                  VTC00024$
CARD    25       $          XN=XN*X                                                       VTC00025$
CARD    26       $          TN=A(J)*XN                                                    VTC00026$
CARD    27       $          B=B+TN                                                        VTC00027$
CARD    28       $      300 CONTINUE                                                      VTC00028$
CARD    29       $          B= -B/TT                                                      VTC00029$
CARD    30       $          DPS=PK*10.**B-P                                               VTC00030$
CARD    31       $          TS=TS-DPS                                                     VTC00031$
CARD    32       $          IF (ABS(DPS) - 1.E-3) 2,2,6                                   VTC00032$
CARD    33       $        2 VTCO2V= TS+273.16                                             VTC00033$
CARD    34       $      999 RETURN                                                        VTC00034$
CARD    35       $          END                                                           VTC00035$
                 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Figure 1. Example of a data function

The data bank must be easily extendable for including new functions

MAPLIB consists of two parts (see Figure 2):

(1) A data set which contains the data functions together with some library control routines.

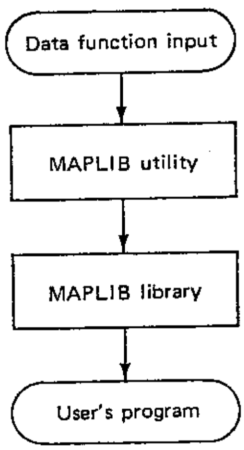

Figure 2. Flow of information in MAPLIB

(2) A utility program which incorporates new functions and control routines in the data set or scratches old ones.

If data for new property–material combinations are to be integrated in MAPLIB, the corresponding functions are coded in FORTRAN. This set of source functions is the input to the utility program. The utility program tests the correctness of the functions in various respects, catalogues the new materials and properties together with the information delivered on the comment cards, completes the functions with the necessary standard control routine calls and then submits the functions to the compiler.

Security

Two aspects of security are considered.

Any incorrect function call must not result in an unpredictable error

In MAPLIB an extensive error control system is built in (see Figure 3). In a normal case each retrieval request is checked with respect to the following possible errors:

(1) The data function requested could be missing.

(2) Too few or too many arguments are passed to the function by the calling program.

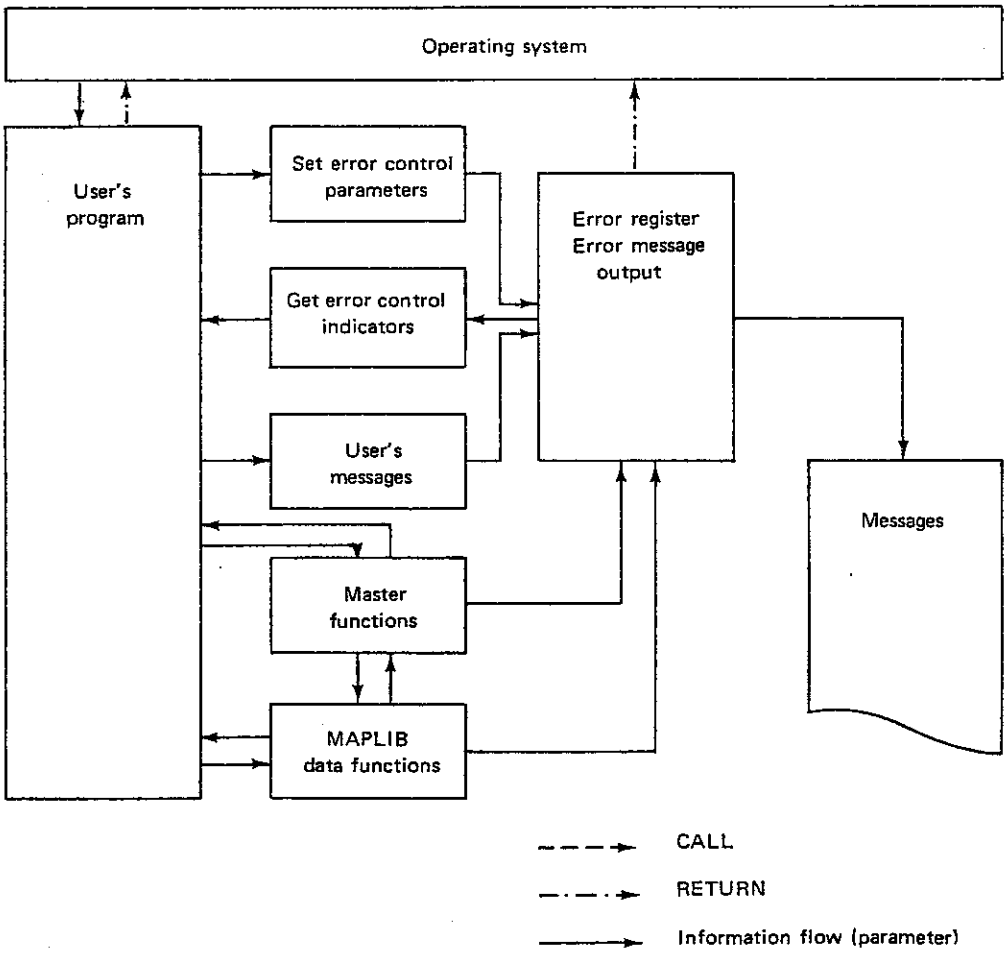

Figure 3. MAPLIB error control system

(3) The values of the parameters could be out of the validity range for which the function is established.

Whenever an error has been detected, MAPLIB normally prints out an error message in English plain text and the first error results in termination of the program execution. But the user has the possibility of varying this standard reaction in several ways. It is especially possible to raise the number of permissible errors (default: 0). Then the system offers special routines which enable the user's program to inquire how many and what kind of error occurred so that the program itself can decide how to behave in a case of error at execution time.

The data function must be protected from unintentional changes

In the utility program an attempt was made to avoid all possibilities of unintended or erroneous changes as far as possible. Since MAPLIB utility is executed under operating system control, no safeguards can be taken within this program against inadvertent loss or change of information due to errors or breakdown on the operating system level.

Figure 4 shows the data flow of the utility program. The 'SOURCE' data set contains all

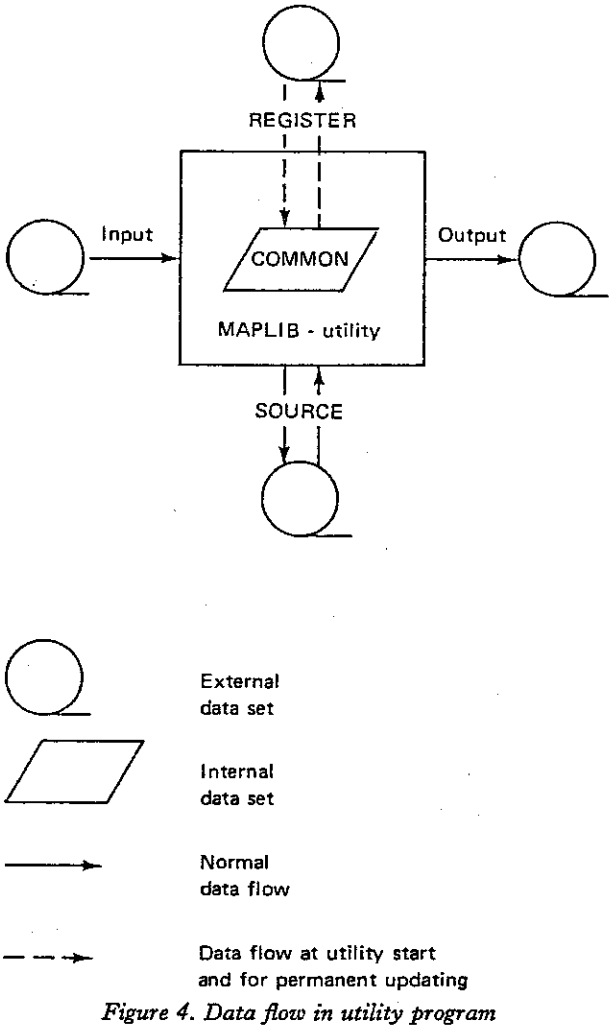

Figure 4. Data flow in utility program

previously integrated data functions in the FORTRAN source version. The 'REGISTER' data set contains the pointers to the functions on the source data set. At the beginning of the utility execution the pointers are read into the COMMON memory. When a new function is found in the input to the program, the completed function is written on the source data set behind the last old record and the pointers in the COMMON are changed. But only when no errors are detected by the utility program and the user has passed the correct password, are the COMMON data rewritten on to the register data set and the source data set is compressed. Thus, in a case of error at the next utility execution all permanent data sets contain the same information as at the beginning of the first faulty execution.

Effectiveness

The data retrieval time must be as low as possible

The demand for effectiveness and the other demands such as flexiblity and security cannot be compatible; the function call takes the least time when the call is done on the lowest level—the most inflexible level. The function furnishes the value quickest when the function call is not checked with respect to the possible errors.

To solve this problem, MAPLIB offers the possibility of being used in an inflexible and dangerous way too; the function can be called on the lowest level with all controls bypassed, if the user's program issues the appropriate calls at execution time. In this case, it is the user's responsibility to safeguard his program against breakdown and incorrect results.

Transparency

The system must be able to deliver to the user all information related to the data which are used by his design program

In MAPLIB the user can ask the utility program for all the information he wants and which MAPLIB knows; this means all information added to the functions on comment cards. Moreover MAPLIB supplies summaries of all in MAPLIB integrated data together with explanations of the symbols defined, the units of the data and the parameters needed by the functions.

Compatibility

The data bank must be applicable to as many users as possible

For this reason, FORTRAN was used as the base language, because FORTRAN is by far the most commonly used language in computer-aided engineering design. However, there exists a great number of 'FORTRANS'. Though it was the intention to make MAPLIB applicable on all computers with a FORTRAN IV compiler, MAPLIB cannot comply with the demand completely. Some restrictions must be accepted:

(1) MAPLIB operates on character strings. Therefore, in the existing version, MAPLIB is only applicable on machines with at least four characters per word.
(2) The utility program needs direct access space.
(3) Without overlay the utility program needs 220K bytes of memory. This large amount of memory is needed because the program helps the user in many ways to implement new functions correctly.
(4) The control of the number of arguments passed by the user's program to the data functions is impossible in pure FORTRAN. For this purpose a small assembly-language function must be used which is described elsewhere.[3]

CONCLUSION

It has been shown that MAPLIB accomplishes most, but not all, conditions which should be fulfilled by a good data bank for computer programs. MAPLIB is documented in detail in Reference 4. It is implemented on an IBM 360/65–85 in Karlsruhe and contains today the most important properties and materials requested in the thermohydraulic reactor design. Its performance has been very satisfactory so far.

REFERENCES

1. C. A. R. Hoare, *Simulation Programming Languages*, Proceedings of the IFIP Working Conference on Simulation Programming Languages (Ed. J. N. Buxton), North-Holland Publishing Co., Amsterdam, 1968, pp. 158–174.
2. International Organization for Standardization, *Rules for the Use of Units (SI Units)*, ISO Recommendation R 1000, 1st edn., 1969.
3. A. Pee and U. Schumann, *Vorteile der Dynamisierung von Argumentenlisten in Fortran and Icetran*, Kernforschungszentrum, Karlsruhe, 1971, KFK 1488.
4. U. Schumann, *MAPLIB—Ein Programmsystem zur Bereitstellung von Stoffdaten für Rechenprogramme*, Kernforschungszentrum, Karlsruhe, 1970, KFK 1253.

Comments on "A Fast Computer Method for Matrix Transposing" and Application to the Solution of Poisson's Equation

ULRICH SCHUMANN

In the above correspondence,[1] an algorithm has been described that allows the transposing of a $2^n \times 2^n$ matrix by reading and writing n times at the most via direct access rows of the matrix and by permuting its data elements. The generalization to arbitrary square matrices has not been described, and the generalization to nonsquare matrices has been reported to be impossible.

The algorithm proposed is very similar to one described and proved completely in [1], which needs only two *sequential* data sets; it has been developed to transpose *arbitrary square and nonsquare* matrices. The idea of this algorithm is as follows: The $M \times N$ matrix is assumed to be stored by row on a sequential data set. The number N is the product of its prime factors P_i, $i = 1, 2, \cdots, n$. Then n steps are needed to transpose the matrix. In every step ($i = 1, n$) the data elements are read from the data set in the following order: $1, 1 + P_i, 1 + 2P_i, \cdots, 1 + MN - P_i; 2, 2 + P_i, \cdots, 2 + MN - P_i; \cdots; P_i, P_i + P_i, P_i + 2P_i, \cdots, M \times N$ and written on the second data set sequentially. In doing this, the first data set must be read P_i times, and the second must be written once. Then the data sets are interchanged and the ith step is finished. After all n steps, the last output data set contains the matrix, now stored by column, and for the total process the data have to be read as often as the sum of the prime factors and written n times; so a total of $n + \sum_{i=1}^{n} P_i$ transputs (input + output) are needed. In the case of $M = N = 2^n$ this sum results in $3n$. By combining pairs of 2 to "prime number" 4, the algorithm may be refined to reduce this sum to $2.5n$.

This is much less than the $(N + 1)$ transputs needed when transposing the matrix stored on a sequential data set in a direct way. Because the number of transputs approach this bad number if N is a prime number itself, the matrix should be expanded to a larger one with row length N_{opt} in the first step by adding meaningless data. Using the optimum numbers N_{opt} of Table I, the number of transputs are always less than $3\log_2 N_{opt}$ and equal to $2.5 \log_2 N_{opt}$ in the limit [1].

In Eklundh's algorithm only $2n$ transputs are needed, due to the use of direct access transputs. So his algorithm is superior in this case. But the algorithm described above is more general since it is applicable to nonsquare matrices and arbitrary square matrices. Moreover the permutation described above and presented formally in [1, eq. (1)] may be used to generalize Eklundh's algorithm to arbitrary square matrices.

Besides the applications mentioned in [1] and those for fast Fourier transform (FFT), the algorithm has been used for the direct solution of the Poisson equation $\nabla^2 p = d$ in two and three dimensions over a Cartesian and cylindrical grid with about 10^6 nodes. This method will be described shortly for the two-dimensional case of the discrete Poisson equation.

$$(p_{i-1,j} - 2p_{i,j} + p_{i+1,j})/\Delta x^2 + (p_{i,j-1} - 2p_{i,j} + p_{i,j+1})/\Delta y^2 = d_{i,j}$$

over the Cartesian grid

$$x_{i,j} = (i-1)\,\Delta x \quad y_{i,j} = (j-1)\,\Delta y, \qquad i = 1, 2, \cdots, N;\ j = 1, 2, \cdots, N$$

with Dirichlet, Neumann, or periodical boundary conditions. The solution is found in the following way (mathematically according to Hockney [2]).

Step 1: Expand $d_{i,j}$ for one coordinate (e.g., x) into "eigenfunktionen" $d_{i,j} = \sum_{l=1}^{N} A_{l,j} \exp(\sqrt{-1}\, 2\pi (i-1)(j-1)/N)$, and evaluate $A_{l,j}$ by FFT (N times for j = constant).

Step 2: Expand $p_{i,j}$ accordingly: $p_{i,j} = \sum_{l=1}^{N} B_{l,j} \exp(\sqrt{-1}\, 2\pi (j-1)(i-1)/N)$; substitution into the above Poisson equation results in N tridiagonal linear equation systems for N unknowns $B_{l,j}$ (i = constant), which may be solved in a direct way very easily.

Step 3: Compute $p_{i,j}$ out of $B_{i,j}$ by N FFT's, each for j = constant. The arrays $d_{i,j}$; $A_{i,j}$; $B_{i,j}$; $p_{i,j}$ ($i = 1, N$; $J = 1, N$) may overstore one array $S_{i,j}$ successively. For $N \geq 1000$ the N^2 elements of S cannot be stored in the main storage of today's computers. S must be cut into pages which are swapped by hardware or software between a background storage device and the main storage.

Initially it is suitable to design the pages so that just one column (j = constant) can be stored in one page. Steps 1 and 3 may be executed without problems—for each of the N FFT's one page is needed, and the pages are to be read and written $4N$ times.

But in executing Step 2 for each of the N linear equation solutions, a row (i = constant) of S is needed. This means just one data element out of each page. Therefore, this step requires $2N^2$ transputs of pages and, thus, $4N + 2N^2$ are needed for the complete solution.

This unacceptable high number of page swaps can be decreased dramatically by transposing the matrix S according to [1] between Steps 1, 2 and 2, 3. Then only approximately (exact for $N = 2^n$, n even) $8N + 5N \log_2 N$ transputs are required to get the complete solution. Thus, in the case of $N = 1024$, by using the matrix transposing algorithms, the number of page swaps are reduced by a factor of about 35.

TABLE I
OPTIMUM VALUES N_{OPT} OF THE ROW LENGTH

1	2	3	4	5
6	7	8	9	10
12	15	16	18	20
21	24	25	27	28
30	32	36	40	45
48	50	54	56	60
64	72	75	80	81
84	90	96	100	108
112	120	125	128	135
144	150	160	162	168
180	192	200	216	225
240	243	256	270	288
300	320	324	336	360
375	384	400	405	432
448	450	480	486	500
512	540	576	600	625
640	648	675	720	729
768	800	810	864	900
960	972	1024	1080	1125
1152	1200	1215	1280	1296
1344	1350	1440	1458	1500
1536	1600	1620	1728	1792
1800	1875	1920	1944	2000
2025	2048	2160	2187	2304
2400	2430	2500	2560	2592
2700	2880	2916	3072	3125
3200	3240	3375	3456	3600
3645	3840	3888	4096	

REFERENCES

[1] U. Schumann, "Ein Verfahren zum Transponieren grosser, sequentiell gespeicherter Matrizen," *Angew. Inform.*, pp. 213–216, May 1972.

[2] R. W. Hockney, "A fast direct solution of Poisson's equation using Fourier analysis," *J. Ass. Comput. Mach.*, vol. 12, pp. 95–113, 1965.

Manuscript received September 1, 1972; revised October 31, 1972.
The author is with the Institut für Reaktorschungszentrum, Karlsruhe, West Germany.