Algorithms for multidimensional numerical integration with singularities

J. C. Romão and R. Vilela Mendes

Instituto de Fisica e Matemática, Av. Gama Pinto, 2, Lisboa-4, Portugal

Two algorithms for numerical multidimensional integration are described, which handle efficiently (integrable) singularities lying in arbitrary regions of the integration volume while keeping the memory requirements at the same level as in interval adaptation methods.

(Received September 1977)

The need to integrate numerically functions with (integrable) singularities occurs frequently in computational work. For simple functions and one-dimensional integrals simple standard techniques of truncation of the interval, change of variables, subtraction of the singularity, etc. are applicable (Davis and Rabinowitz, 1967). However for multidimensional integrals and when the functions are so complicated that their behaviour is essentially unknown, it is convenient to use an integration procedure that automatically adapts itself to the rate of variation of the integrand at each point.

An algorithm (RIWIAD) of Monte Carlo integration with automatic interval adaptation for multidimensional integrals has been developed by Sheppey (1964), Dufner (1970) and Lautrup (1971) and successfully used, in particular, for the computation of higher order corrections in quantum electrodynamics (see for example Calmet, 1973).

The RIWIAD algorithm is fairly appropriate to handle integrals with (integrable) singularities lying on the boundary of the integration volume, which is taken to be the unit hypercube. In each iteration the algorithm computes the variance in the slices of the integration volume associated to the subintervals in each integration axis, i.e. for the n'th interval of the i'th axis

$$\sigma_{in}^2 = \sum_{n_1 \ldots n_d} \sigma_{n_1 \ldots n_d}^2 \delta_{n_i, n} \tag{1}$$

In the following iteration the interval sizes are adapted (fixed number of points per interval) to ensure that regions of higher variance will have a higher density of sampling points.

From the very nature of the algorithm it clearly follows that in general it will handle well integrands with singularities or fast rates of variation lying in hyperplanes perpendicular to the integration axis. However if the singularity hyperplanes lie obliquely to the axis (skew singularities) not much improvement should be expected from interval adaptation. In simple cases one might work out a change of variables to put the singularities in the boundary of the integration volume, or at least in hyperplanes perpendicular to the axis. In general, however, this is not possible if many singularity planes exist and in any case it is not practical nor in line with the automatic integration philosophy.

To construct an algorithm to handle skew singularities the obvious brute force method would be to keep in store from iteration to iteration information about the subintegrals and variances $\sigma_{n_1 \ldots n_d}$ in all subvolumes of the integration domain (instead of just in the hyperslices perpendicular to each axis) and adapt the number of points in each subvolume accordingly. Notice however that, if m is the number of intervals in each axis, the number of subvolumes grows as m^d whereas the number of hyperslices grows only as $m \times d$ with the dimensionality d. Thus, for high dimensionality integrals and a fine interval division this method would be too demanding in memory requirements. This is the reason why in RIWIAD the interval adaptation method was preferred.

In the following we will describe the main features of the FORTRAN program TSAIR (Two Step Adaptive Integration Routines) which contains two algorithms that handle satisfactorily skew singularities or fast rates of variation in skew hyperplanes and keep the memory requirements at the same level as RIWIAD. All integrations to be executed by the program should be normalised to the unit hypercube.

Algorithm 1

In this algorithm one aims at computing the integral with minimum variance for a fixed total number L of sampling points. If one allows for variable number of points in the subvolumes, mathematically the problem consists of minimising the function*

$$\sigma^2 = \sum_{n=1}^{N} \frac{V_n^2}{L_n - 1} x_n \tag{2}$$

by variation of L_n with the subsidiary condition

$$\sum_{n=1}^{N} L_n = L \tag{3}$$

L_n is the number of sampling points in the subvolume V_n, N is the total number of subvolumes and

$$x_n = \langle f^2 \rangle_n - \langle f \rangle_n^2 \tag{4}$$

is the second-order moment of the function in the subvolume n. Applying the Lagrange multiplier method the result is as follows

$$L_n = \frac{V_n \sqrt{x_n}}{\sum_{n=1}^{N} V_n \sqrt{x_n}} (L - N) + 1 \tag{5}$$

Practically what the algorithm does is to start with a certain number of iterations where one proceeds to interval adaptation as in RIWIAD. The number of such iterations is never smaller than a number chosen by the user and supplied in a data card to the section CONSTANTS AND COUNTERS (see below). Typically we have been using five such iterations. After these iterations if the variance is still decreasing at a rate faster than a certain amount (10% in our program) the algorithm continues automatically to make interval adaptations until this limit is reached.

Starting with the last of the fixed RIWIAD iterations the program computes and keeps in the memory the value of $\sum_{n=1}^{N} V_n \sqrt{x_n}$. Once the interval adaptation step is completed the program switches to a second step where the interval structure is kept fixed and the number of points per subvolume L_n is allowed to vary to approach the ideal value of Equation (5). However to apply Equation (5) one needs information on

*For the general mathematical background of Monte Carlo adaptive integration see Lautrup (1971).

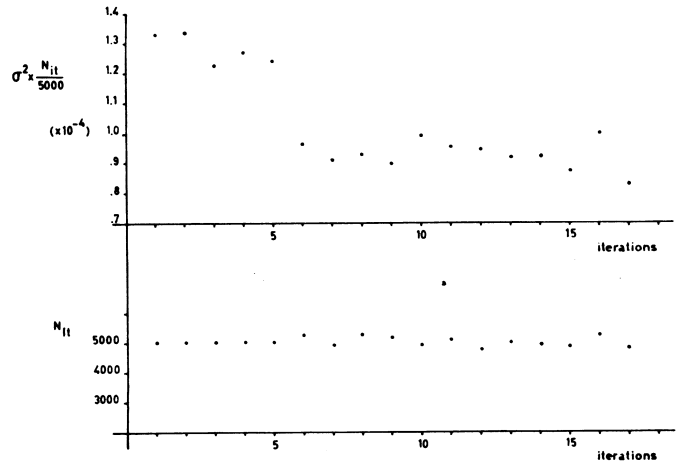


Fig. 1 Variance and number of sampled points per iteration for a typical test of algorithm A

$V_n\sqrt{x_n}$ for each subvolume and, as discussed above, to keep these values from iteration to iteration would overburden the memory requirements.

The procedure we used is the following: each time the program starts sampling inside a subvolume it first computes a small number y of points. Estimates for $V_n\sqrt{x_n}$ and L_n from Equation (5) are then computed from these points. If the estimate for L_n is greater than the number y of points already sampled the computer samples $L_n - y$ more points (up to a maximum Z fixed by the user), otherwise it skips to the next subvolume.

The present algorithm combining interval adaptation and variable number of points per subvolume with essentially the same memory requirements as RIWIAD shows itself as fairly effective in obtaining small variances in the integration of arbitrary functions.

For the operation of Algorithm 1 the user should supply:

Data

NDIM = number of dimensions of the integral

NSUB = total number of subvolumes

IX = initial value for the random numbers subroutine RANDU. The last digit must be odd*

ACC = value of relative error to be obtained. When it is reached the program stops

BL = number of points per subvolume in the interval adaptation iterations

Z = maximum number of points allowed in each subvolume

BLZ = number of points one uses to estimate $V_n\sqrt{x_n}$ and L_n in each subvolume when the total number of points (N_{tot}) is BL_*NSUB. Related to the number y in the text by

$$y = BLZ_*\sqrt{N_{tot}/BL_*NSUB} + \cdot 5$$

Constants and counters

IOP = 1

ITMAX = minimum number of interval adaptation iterations

NMAX If after the interval adaptation step one wants to use all remaining points at one time one uses *NMAX* = maximum number of points. Otherwise one uses *NMAX* = 1 and the computer executes successive iterations until it uses the time allowed in the JOB CARD. The number of points sampled in these iterations is approximately equal to the number of points used in the interval adaptation iterations (i.e. $NSUB_*BL$)

*The random number subroutine RANDU is supposed to work satisfactorily for machines of arithmetic similar to the IBM 360/370 series (UNIVAC 9300/9400, XD5 sigma 5/7/9, etc). For other machines, because of the size of the integer constant involved, it should probably be replaced by some other appropriate subroutine.

Algorithm 2

Together with algorithm 1 an alternative was developed which could be used with a minimal change in the program: the replacement of two cards.

The point of view in this algorithm is the computation of the integral for a specified variance with the minimum possible number of points. The mathematical problem to be solved is then the minimisation of the function

$$L = \sum_{n=1}^{N} L_n$$

with the subsidiary condition

$$\sigma^2 = \sum_{n=1}^{N} \frac{V_n^2}{L_n - 1} x_n$$

By variation of L_n and the Lagrange multiplier method the result is:

$$L_n = \frac{\sum_{i=1}^{N} V_i\sqrt{x_i}}{\sigma^2} V_n\sqrt{x_n} + 1 \qquad (6)$$

In this algorithm a uniform interval division is used. In the first iteration one computes the integral, the variance and also the sum $\sum_{i=1}^{N} V_i\sqrt{x_i}$. On the second and last iteration, by the same procedure as described for Algorithm 1, one computes an estimate for $V_n\sqrt{x_n}$ in each subinterval using the first few sampled points in each subvolume. From this estimate and Equation (6) L_n is then computed and the rest of the necessary points sampled in the subvolume n.

Instructions for the use of Algorithm 2:

Data

Same as in Algorithm 1 with *BL* now being the number of points in the first iteration.

Constants and counters

IOP = 2

ITMAX = 1

The program TSAIR* has been tested and used for the integration of several types of functions. For illustration purposes we display in **Fig. 1** the behaviour from iteration to iteration of Algorithm 1 in the case of a typical 2 dimensional function:

$$f_1(x,y) = -\log\{|x-y| \,.\, |x-1| \,.\, |y-1| \,.\, |2x+y-2| \,.\, |\tfrac{1}{2}x + y - \tfrac{1}{2}|\}$$

which possesses both skew singularities and singularities in lines perpendicular to the axis.

The upper plot shows the variance normalised for the same number of points per iteration. As expected in the first few iterations one obtains some improvement (~6%) with interval adaptation. However after the third iteration no further improvement is observed. After the fifth iteration the interval adaptation ceases, the second part of Algorithm 1 becomes active and a new significant improvement in the variance (~24%) is obtained.

The lower plot displays the number of sampled points per iteration. After the fifth iteration this number is no longer fixed, however it is seen that it deviates very little from the

*IFM program library

Table 1 Integration of $f_1(x,y) \int f_1 = 5.806852\ldots$

Iteration	*Estimated error* ($\times 10^{-2}$)	*Actual error* ($\times 10^{-2}$)
1	3·06	0·018
2	3·02	1·89
3	2·85	− 1·20
4	3·02	1·39
5	2·84	− 2·70
6	2·88	− 1·69
7	2·27	− 2·97
8	2·34	− 3·80
9	2·36	− 1·01
10	2·39	0·6
11	2·39	− 1·19
12	2·59	3·5
13	2·45	− 2·03
14	2·41	− 2·68

$[NSUB = 100;\ BL = 20;\ BLZ = 10]$

Table 2 Integration of $f_2(x,y) \int f_2 = 4.466666\ldots$

Iteration	*Estimated error* ($\times 10^{-2}$)	*Actual error* ($\times 10^{-2}$)
1	7·05	− 9·07
2	10·90	0·52
3	8·18	−10·09
4	7·27	−11·55
5	6·58	− 9·54
6	8·32	1·49
7	3·64	−10·35
8	4·23	− 5·25
9	5·17	− 7·69
10	5·09	− 4·92
11	4·74	−11·17
12	5·77	− 5·82
13	4·99	−10·44
14	5·16	− 8·18

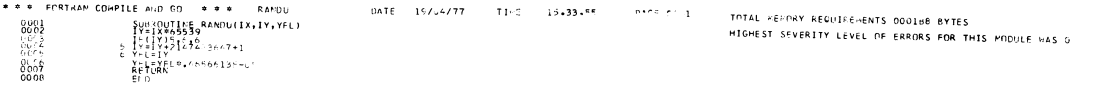
value used in the interval adaptation iterations. This is a good check of the reliability of the method used to estimate the quantities $V_n\sqrt{x_n}$.

In the **Tables 1** and **2** we compare estimated and actual errors for integration of $f_1(x,y)$ and

$$f_2(x,y) = |x + y - 1|^{-1/2} + |x - y|^{-1/3}$$

For these simple functions, which we have chosen because their exact integrals are easy to compute analytically for comparison, one sees that the RIWIAD algorithm (first 5 iterations) leads to actual errors not very different from our Algorithm 1; however, the error estimation is on average somewhat more accurate in the latter.

```
// JOB TSA00804 33008040TIME=     00804                         15.32.47
// OPTION LINK
// EXEC FFORTGO
* * *  FORTRAN COMPILE AND GO   * * *    MAINPGM        DATE  19/04/77    TIME   15.32.55      PAGE 0001
C
C         PROGRAM TSAIR
C
C         INITIAL SPECIFICATIONS
C
          DIMENSION A1(250),A2(250),A3(250),A4(250),MA(10),MB(10),MC(10),
         1XA(10),XB(10),XC(10),X(10)
          READ(1,100) NDIM,NSUB,IX
          READ(1,110)ACC,BL,Z,BLZ
C
C         CONSTANTS AND COUNTERS
C
          IOP=1
          NMAX=1
          ITMAX=5
          FACTOR=1.65
          ZERO=1.0E-03
          IND1=0
          IND6=0
          NTOT=0
          SXISN=0.0
          SRV=0.0
          SWI=0.0
          FACT1=1.0
          FACT2=1.0
          FACT3=1.0
          RNDIM=1.0/FLOAT(NDIM)
          ANSUB=NSUB
          PONTOS=BL*ANSUB
          ALFA=0.3-0.2*RNDIM
C
C         COMPUTE INITIAL INTERVALS
C
          K=IFIX(ANSUB**RNDIM+ZERO)
          A=1.0/FLOAT(K)
          L=NDIM*K
          DO 1 I=1,L
        1 A1(I)=A
          DO 2 I=1,NDIM
        2 MA(I)=K
C
C         ENTRY AND REENTRY FOR ITERATIONS
C
        3 IND1=IND1+1
          IND4=0
          NINT=0
          DO 4 I=1,NDIM
          MB(I)=NINT
          K=MA(I)
        4 NINT=NINT+K

* * *  FORTRAN COMPILE AND GO   * * *    MAINPGM        DATE  19/04/77    TIME   15.32.55      PAGE 0002
C
C         INITIALISE INTEGRATION LOOP
C
        5 L=0
          V=1.0
          DO 6 I=1,NDIM
          MC(I)=1
          K=L+1
          L=L+MA(I)
          A=A1(K)
          XA(I)=A
          XB(I)=0.0
          XC(I)=V
          V=V*A
          X1=0.0
          DO 7 J=K,L
          A2(J)=X1
          X1=X1+A1(J)
          A3(J)=0.0
        7 A4(J)=0.0
          A1(L)=A1(L)+1.0-X1
        6 CONTINUE
C
C         INTEGRATION LOOP
C
        8 SF1=0.0
          SF2=0.0
          XSUB=0.0
          IND3=0
          IND5=0
          IF(IND6.EQ.0)GO TO 9
          LB1=BLZ*FACT3+.5
          GO TO 10
        9 LB1=IFIX(BL+ZERO)
       10 DO 11 [illegible]=1,LB1
          IND4=IND4+1
          IND5=IND5+1
          DO 12 I=1,NDIM
          CALL RANDU(IX,IY,YFL)
          IX=IY
       12 X(I)=XB(I)+YFL*XA(I)
          Y=FUN(X,NDIM)
          SF1=SF1+Y
       11 SF2=SF2+Y**2
          AL=IND5
          FMED=SF1/AL
          FQMED=SF2/AL
          XISN=FQMED-FMED**2
          C=V*FMED
          D=V**2*XISN/(AL-1.0)
          [illegible]
          RXISN=[illegible]
          SXISN=SXISN+RXISN
       14 IF(IND6.[illegible])GO TO 13

* * *  FORTRAN COMPILE AND GO   * * *    MAINPGM        DATE  19/04/77    TIME   15.32.55      PAGE 0003
          RXISN=V*SQRT(XISN)
          GO TO(42,43),IOP
       42 PSUB=RXISN/SXISN*(FACT1*FACT2*PONTOS-ANSUB)+1.0
          GO TO 15
       43 PSUB=RXISN*SXISN/VAR+1.0
       15 IF(PSUB.LE.XSUB)GO TO 13
          IF(PSUB.GE.Z)GO TO 16
          LB1=PSUB-AL
          GO TO 17
       16 LB1=Z-AL
       17 IF(IND3.EQ.2)GO TO 13
          IND3=IND3+1
          XSUB=PSUB
          GO TO 10
       13 CONTINUE
          DO 18 I=1,NDIM
          J=MB(I)+MC(I)
          A3(J)=A3(J)+C
       18 A4(J)=A4(J)+D
C
C          LOOP THE  LOOP
C
          I=NDIM
       19 J=MC(I)
          K=MB(I)+1
          IF(J.NE.MA(I))GO TO 20
          MC(I)=1
          XA(I)=A1(K)
          XB(I)=0.0
          I=I-1
          IF(I)21,21,19
       20 MC(I)=J+1
          V=XC(I)
          K=K+J
          A=A1(K)
          XA(I)=A
          XB(I)=A2(K)
          V=V*A
       22 IF(I.EQ.NDIM)GO TO 8
          I=I+1
          XC(I)=V
          V=V*XA(I)
          GO TO 22

          CALCULATE INTEGRAL AND ERROR

       21 J=MA(1)
          SI=0.0
          SV=0.0
          DO 23 I=1,J
          SI=SI+A3(I)
       23 SV=SV+A4(I)
          RV=1.0/SV
          WI=SI*RV

* * *  FORTRAN COMPILE AND GO   * * *    MAINPGM        DATE  19/04/77    TIME   15.32.55      PAGE 0004
          SRV=SRV+RV
          SWI=SWI+WI
          VALUE=SWI/SRV
          FSV=FACTOR*SQRT(SV)
          ERROR=FACTOR/SQRT(SRV)
          NTOT=NTOT+IND4
C
C         PRINT OUT THE RESULTS OF THIS ITERATION
C
          WRITE(3,24)SI,SV,FSV,VALUE,ERROR,IND4,NTOT,(MA(I),I=1,NDIM)
       24 FORMAT(1H0//21H VALUE OF INTEGRAL = ,E15.7/
         121H VARIANCE            = ,E15.7/
         221H ESTIMATED ERROR     = ,E15.7/
         321H ACCUMULATED VALUE   = ,E15.7/
         421H ACCUMULATED ERROR   = ,E15.7/
         521H NUMBER OF POINTS    = ,I9/
         621H TOTAL NO OF POINTS= ,I9/
         723H INTERVAL STRUCTURE... ,(10I4)/)
          IF(NTOT.GE.30000)GO TO 30
C
C         CHECK IF ACCURACY IS REACHED
C
       25 ACC1=ERROR/ABS(VALUE)
          IF(ACC1-ACC)26,26,38
       26 WRITE(3,28)IND1,VALUE,ERROR,NTOT
       28 FORMAT(1H0////31H DESIRED ACCURACY REACHED AFTER,I5,
         111H ITERATIONS//
         214H FINAL VALUE =,E15.7//
         314H FINAL ERROR =,E15.7//
         423H TOTAL NO OF POINTS = ,I10//)
          GO TO 30
C
C         COMPUTE NEW CONSTANTS FOR NEXT ITERATION
C
       38 CONTINUE
          IF(IOP.EQ.1)GO TO 41
          IND6=IND6+1
          VAR=SV/2.0
          XIND4=IND4
          FACT3=(XIND4/PONTOS)**0.5
          GO TO 3
       41 CONTINUE
          IF(IND1.GE.ITMAX)GO TO 31
          GO TO 27
       31 CONTINUE
          IF(IND1.GE.ITMAX)GO TO 40
          L=0
          DO 39 I=1,NDIM
          K=L+1
          M=MA(I)
          L=L+M
       39 WRITE(3,120)(A1(J),J=K,L)
      120 FORMAT(1H0,12F10.7/)
       40 CONTINUE

* * *  FORTRAN COMPILE AND GO   * * *    MAINPGM        DATE  19/04/77    TIME   15.32.55      PAGE 0005
          XIND4=IND4
          FACT1=FACT1*(PONTOS/XIND4)**0.5
          IF(VAR/SV-1.1)34,37,37
       34 IND6=IND6+1
          IF(IND6.GT.1)GO TO 35
          GO TO 33
       35 IF(NMAX.EQ.1)GO TO 36
          IF(IND6.EQ.3)GO TO 30
          FACT2=NMAX/NTOT
       36 GO TO 33
       37 IF(IND6.NE.0)GO TO 33
C
C         COMPUTE NEW INTERVALS
C
       27 L=0
          DO 32 I=1,NDIM
          K=L+1
          M=MA(I)
          L=L+M
          SOMA=0.0
          DO 29 J=K,L
          A=A1(J)
          D=A4(J)
          P=(ALOG(SV/D)*(1.0-A))**ALFA
          Q=((1.0-D/SV)*ALOG(1.0/A))**(-ALFA)
          A=A*P*Q
          A1(J)=A
       29 SOMA=SOMA+A
          CONST=1.0/SOMA
          DO 32 J=K,L
       32 A1(J)=CONST*A1(J)
       33 VAR=SV
          GO TO 3
C
C         END OF PROGRAM
C
       30 CONTINUE
      100 FORMAT(I2,2I10)
      110 FORMAT(4E13.7)
          STOP
          END

* * *  FORTRAN COMPILE AND GO   * * *    MAINPGM        DATE  19/04/77    TIME   15.32.55      PAGE 0006
TOTAL MEMORY REQUIREMENTS 00240C BYTES
HIGHEST SEVERITY LEVEL OF ERRORS FOR THIS MODULE IS 0
```



References

CALMET, J. (1973). Proc. 3rd Colloquium on Advanced Computing Methods in Theoretical Physics, Marseilles.

CZYZ, W., SHEPPEY, G. C. and WALECKA, J. D. (1964). *Nuovo Cimento*, Vol. 34, p. 420.

DAVIS, P. J. and RABINOWITZ, P. (1967). *Numerical integration*, Blaisdell Publishing Company.

DUFNER, A. J. (1970). Proc. 1st Coll. on Advanced Computing Methods in Theoretical Physics, Marseilles.

LAUTRUP, B. E. (1971). Proc. 2nd Coll. on Advanced Computing Methods in Theoretical Physics, Marseilles.

Book reviews

Linguistic Structures Processing, Fundamental Studies in Computer Science, Volume 5, edited by A. Zampolli, 1977; 585 pages. (*North-Holland*, Dfl. 110·00)

There is little doubt that those who carry out research in artificial intelligence and even those who restrict themselves to 'ordinary' computational linguistics face problems of an extraordinary complexity. As far as computers are concerned it is well known that provided valid requirements can be expressed unambiguously to the computer then correct results may be expected but, of course, the major and as yet unaccomplished task of linguistics is to define all the rules of language and speech behaviour concisely and unambiguously. The proverbial man in the street feels that speech is formed almost instinctively and he is totally unaware of any hidden 'rules' which might govern his utterances. Add to this the fact that those who have tried to formulate rules to be used as a sort of algorithmic basis for computation involving language data find that their rules are broken constantly and in many different ways; they find, in fact, that they are unable, by and large, to penetrate through to what might be called the algebra of language. To ask a specific question: what goes on in the brain as an utterance is formed? This question begs, of course, two further ones: firstly, what are the parameters of the channel of communication between two humans in contact with each other; and secondly, how does a person on the receiving end of a message perceive it and interpret it? Only when we know more of the highly structured thought processes and of the equally complex linguistic data structures which model them shall we be able to make advances in the field of human understanding in general and in the field of artificial intelligence and computational linguistics in particular.

Research in this field is being carried out in several countries, notably in the United States, and there are obvious advantages in having specialists get together and if, at the same time, they can pass on their knowledge to students so much the better. Professor Zampolli provides such a setting by organising a summer school at Pisa every two years which is now justifiably regarded as the best of its kind. Those lecturers who attended the school in 1973 have contributed to this volume. Some wrote papers, whilst others submitted the text of lectures. They are not all 'articles' as stated in the preface and it is suspected that not all 'reflect the research activity of the author in the period following the school'.

About one matter, however, there can be no doubt: the 'nominal roll' of those contributing to this volume is an impressive list of names. It includes those who are recognised to be at the forefront of research in their particular sector, scholars such as Fillmore, Hays, Kay, Wilks, Winograd, and Woods. Kay's exposition of the state of the art as far as morphological and syntactic analysis is concerned occupies something like one fifth of the book and is a valuable contribution. Winograd's contribution on artificial intelligence is of similar length and can also be highly recommended. Wilks performs a useful service by contextualising natural language understanding systems within the AI paradigm. Woods gives the reader a valuable account of the performance and potential of natural language question answering systems. The remaining eight authors contribute the last 40% of the book, so to speak, and from amongst them one must single out Allen for his description of how to synthesise speech from unrestricted text.

The editorial standards of this book are high but are not impeccable. There are quite a number of instances of odd English and spelling errors. Errors of this sort and the presentation of text that is not right-justified should not occur in a volume that costs so much. These minor blemishes notwithstanding, this text can be recommended to students of artificial intelligence and computational linguistics, especially to those students who are searching for a general orientation or state of the art report. What a pity it is, however, that it took four years to publish this book. A 1977 publication that represents a state of thinking in 1973 is an unfortunate occurrence which correlates negatively with this rapidly developing field.

JOAN SMITH (Manchester) and F. E. KNOWLES (Birmingham)

Microprogramming Primer, by H. Katzan Jr., 1977; 254 pages. (*McGraw-Hill*, £13·45)

This is a practically oriented treatment of microprogramming and machine emulation. Based on the reasonable assumption that the reader is familiar with the basic concepts of computers, programming, operating systems and language translators, the book first presents the concepts of microprogramming by relating them to the organisation of a modern computer. The concepts of emulation are introduced, including a description of a suitable model computer and its microcode. Throughout the book, examples are given, and exercises set, based on a simple machine, called the D-machine. By the end of the book this machine is being used to emulate more complex machines, with stacks and multiple registers.

Because of its style and content, this book is most suitable for readers at a final-year degree level. Indeed, it would make an excellent course text for a study of microprogramming and emulation. The subject matter is well ordered and the text well supported by many examples. Each chapter ends with a list of questions and exercises, together with an up-to-date vocabulary list. I would strongly recommend that readers tackle the problems before proceeding to the next chapter.

It is, of course, possible to find things wrong with the book. The use of a particular model computer will not please everyone, but critics should note that the author has made available an instructors' guide and an emulator/translator package written in FORTRAN (remember FORTRAN?) which should be easily transportable to any *real* computer. My own major criticism is the price. I know that £13-odd is not a lot by some standards, but why cannot publishers produce cheaper (paper) editions that students will buy?

To sum up—an excellent introduction to the field of microprogramming, which makes a worthwhile addition to any computer science library.

ALAN E. CHANTLER (Yelvertoft)