Werkbespreking 24 september 1968 RA W19-1
M.H. van Emden
Onderwerp:

ADAPTIVE ROMBERG INTEGRATION

A. Why adaptive Romberg integration?

Romberg's method for numerically integrating a function of one variable has many desirable properties (see: Bauer F.L., H. Rutishauser and E. Stiefel: New aspects in numerical quadrature, Proc. Symp. Appl. Math., vol. 15 (1963)). One of these is that a polynomial of arbitrary high degree may be integrated exactly if enough function values are available. The function values used are spaced at equal distances and this is a drawback when the function shows intense fluctuations in some part of the interval of integration.
A procedure will be called 'adaptive' if there is a provision for adapting the spacing of the function values to the local intensity of fluctuation of the integrand. Adaptive integration procedures have been published and these have the drawback that an integration formula of fixed order is used, although in many cases enough function values are calculated to use a formula of higher order. Because this is not done, unnecessarily many function values are used.
The procedure to be described below combines the full power of Romberg's method with the adaptive feature.

B. The method of building up the Romberg scheme

In executing Romberg's method the scheme is usually (see, for instance, algorithm no.60 by F.L. Bauer, Comm. ACM, 4, p.255) built up as follows. Suppose the Romberg scheme (denoted from now on by 'RS') has been calculated for 2^n+1 function values and suppose that the correction term was not yet small enough. Then between every two existing function values a new function value is inserted, 2^n in all, and their sum is used for extending the existing RS by one row.
It is, however, possible to derive the RS over a whole interval from the RS's belonging to its left and right halves, which in their turn are derived from ... and so on. This process is executed by a recursive ALGOL procedure. This method is somewhat more laborious than the usual one, but it has the advantage that the adaptive feature may be introduced.

Suppose that over a whole interval the correction is too large. If the function happens to be smooth in some subinterval, this will show as a small correction in the appropriate RS and the integration, of that interval at least, may be finished. In this way the calculation of an RS over an interval causes zero or more subintervals to disappear from the rest of the process because a satisfactory approximation for the integral over these has been found. For the remaining subintervals we know that between every two existing function values a new function value has to be inserted ('interstitial function value' in the sequel) before it makes sense to apply the same procedure to them.

C. Storage management

Every interval treated by the procedure 'subinterval' may give rise to several separate (non-adjacent) intervals to be treated likewise by 'subinterval' at a later time. At any one moment only one interval is being handled and several others may be waiting for their turn. Intervals are represented by a set of function values stored contiguously in the array 'fa'. The problem is to manoeuvre the intervals in such a way that all the unused elements of fa are adjacent to each other and to the one interval being treated at the moment.
The solution used in our program may described as follows. The intervals waiting for their turn are stored in two stacks in fa, the 'waning stack' at one end and the 'waxing stack' at the other end. The unused space is in the middle. In general the interval being intergrated sits astride the unused space and is progressively transferred, as the building of the RS proceeds, from the waning stack onto the waxing stack.

This is possible because in principle the building up of the RS requires the function values sequentially. Some past function values are also required, but the number of these is in the order of the logatithm of the total number of function values. They are conveniently stored as a local variable in the various recursive activations of the procedure 'rom'. The 'interstitial values' are placed on top of the waxing stack, alternating with the values coming from the waning stack.
The adaptive feature requires that a part of an interval be deleted if the correction term in its RS is small enough. Whenever an RS is completed, the corresponding interval is on top of the waxing stack. Deletion is effected by a suitable change of the stack pointer transferring the appropriate part of the waxing stack to the unused space and by placing two descriptive elements on top of the waxing stack. In this way the remainder of the interval is made into a separate interval.
When the waning stack has vanished, the direction is reversed and the waxing stack becomes the waning stack.
When both stacks have vanished, the procedure 'integral' is finished.

D. The results of some tests

The following table shows the result of some tests. The performance of ALGOL procedures for numerical integration using adaptive simpson (actually using, due to extrapolation, the 5-point Romberg formula), Romberg and adaptive Romberg techniques may be compared for some integrands. It will be seen that in general less function values are needed at the expense of more processing time per function. When a single function value takes more than a certain minimum amount of time to compute, the adaptive Romberg procedure will be faster.

integral(0, 2, x × cos(3 × x), x):

	abs.tol.= rel.tol.= 10^{-3}			abs.tol.= rel.tol.= 10^{-5}			abs.tol.= rel.tol.= 10^{-7}			abs.tol.= rel.tol.= 10^{-9}		
	result	time	count	result	time	count	result	time	count	result	time	count
adapt. simps.	-.190701485	.02	17	-.190702507	.05	33	-.190702523	.19	121	-.190702523	.62	397
romberg	-.191025415	.02	9	-.190701539	.03	17	-.190702523	.06	33	-.190702523	.05	33
adapt. romberg	-.191025415	.04	9	-.190701539	.08	17	-.190702522	.18	33	-.190702523	.17	33

integral(0, 2, exp(x × x) × sin(exp(x × x)), x):

	abs.tol.= rel.tol.= 10^{-3}			abs.tol.= rel.tol.= 10^{-5}			abs.tol.= rel.tol.= 10^{-7}			abs.tol.= rel.tol.= 10^{-9}		
	result	time	count	result	time	count	result	time	count	result	time	count
adapt. simps.	+.963629789	.41	129	+.963391383	1.21	385	+.963402535	3.93	1245	+.963402539	12.73	4037
romberg	-2.902980336	.20	65	+.964208997	1.49	513	+.963401092	2.94	1025	+.963401092	2.94	1025
adapt. romberg	-2.902634385	.28	41	+.963368480	1.28	181	+.963397889	2.21	305	+.963402538	3.87	529

integral(exp(–n), 1, 1/(n × x), x); n= 5

	abs.tol.= rel.tol.= 10^{-3}			abs.tol.= rel.tol.= 10^{-5}			abs.tol.= rel.tol.= 10^{-7}			abs.tol.= rel.tol.= 10^{-9}		
	result	time	count	result	time	count	result	time	count	result	time	count
adapt. simps.	+1.000248896	.03	29	+1.000000542	.08	69	+1.000000001	.24	213	+1.000000000	.79	685
romberg	+1.074546713	.05	33	+1.002306617	.13	129	+1.000180265	.26	257	+1.000006760	.47	513
adapt. romberg	+1.002397933	.12	25	+1.000007482	.29	57	+1.000000112	.60	113	+1.000000002	.68	129

integral(exp(–n), 1, 1/(n × x), x); n= 10

	abs.tol.= rel.tol.= 10^{-3}			abs.tol.= rel.tol.= 10^{-5}			abs.tol.= rel.tol.= 10^{-7}			abs.tol.= rel.tol.= 10^{-9}		
	result	time	count	result	time	count	result	time	count	result	time	count
adapt. simps.	+.688633649	.06	45	+1.000000466	.17	149	+1.000000001	.53	449	+1.000000000	1.65	1409
romberg	+5.812003280	.14	129	+2.016409633	.47	513	+1.159575641	1.79	2049	none	4.30	5000
adapt. romberg	+.688641583	.22	45	+1.000008292	.59	113	+1.000000186	1.15	217	+1.000000002	1.91	353

integral(exp(−n), 1, 1/(n × x), x); n= 20

	abs.tol.= rel.tol.= 10^{-3}			abs.tol.= rel.tol.= 10^{-5}			abs.tol.= rel.tol.= 10^{-7}			abs.tol.= rel.tol.= 10^{-9}		
	result	time	count	result	time	count	result	time	count	result	time	count
adapt. simps.	+.346587196	.05	45	+.534241574	.20	169	+.727588879	.79	669	+.874535761	3.08	2621
romberg	+.3608$_{10}$+ 4	1.79	2049	none	4.29	5000	none	4.26	5000	none	4.15	5000
adapt. romberg	+.346595133	.21	45	+.589163316	.71	141	+.830077291	2.00	385	+.989774310	4.33	817

integral(0, 1, (n + 1) × x ↑ n, x); n= 5

	abs.tol.= rel.tol.= 10^{-3}			abs.tol.= rel.tol.= 10^{-5}			abs.tol.= rel.tol.= 10^{-7}			abs.tol.= rel.tol.= 10^{-9}		
	result	time	count	result	time	count	result	time	count	result	time	count
adapt. simps.	+1.000000000	.01	9	+1.000000000	.04	29	+1.000000000	.09	69	+1.000000000	.33	249
romberg	+1.000000000	.01	9	+1.000000000	.02	9	+1.000000000	.01	9	+1.000000000	.02	9
adapt. romberg	+1.000000000	.04	9	+1.000000000	.03	9	+1.000000000	.04	9	+1.000000000	.04	9

integral(0, 1, (n + 1) × x ↑ n, x); n= 10

	abs.tol.= rel.tol.= 10^{-3}			abs.tol.= rel.tol.= 10^{-5}			abs.tol.= rel.tol.= 10^{-7}			abs.tol.= rel.tol.= 10^{-9}		
	result	time	count	result	time	count	result	time	count	result	time	count
adapt. simps.	+1.000076868	.02	13	+1.000000088	.06	41	+1.000000019	.18	129	+1.000000000	.47	333
romberg	+1.000000795	.03	17	+1.000000795	.03	17	+1.000000000	.05	33	+1.000000000	.05	33
adapt. romberg	+1.000449551	.04	11	+1.000000237	.13	27	+1.000000019	.18	37	+1.000000000	.29	59

integral(0, 1, (n + 1) × x ↑ n, x); n= 20

	abs.tol.= rel.tol.= 10^{-3}			abs.tol.= rel.tol.= 10^{-5}			abs.tol.= rel.tol.= 10^{-7}			abs.tol.= rel.tol.= 10^{-9}		
	result	time	count	result	time	count	result	time	count	result	time	count
adapt. simps.	+1.000101514	.03	17	+1.000000396	.07	49	+1.000000038	.23	153	+1.000000000	.59	393
romberg	+1.001790156	.03	17	+1.000008630	.05	33	+1.000000007	.09	65	+1.000000000	.18	129
adapt. romberg	+1.002060235	.05	11	+1.000011561	.09	19	+1.000000059	.20	39	+1.000000000	.41	79

E. Text of the ALGOL procedure

```
real procedure integral(x,a,b,fx,ae,re,max,full up);
value a,b,ae,re,max; real x,a,b,fx,ae,re; integer max;
label full up;
begin integer r,su,l,i,j,lpo,lpi,rpi,rpo,lpa,s,n;
      real h,hh,hmin,tr,sum,t0,t1,s0,s1,d,c,ba,lva,
      rva,mva;
      boolean cl,left,fr; array fa[0:max],t[0:119];
      procedure rom;
      begin integer llp; real llv,ltr,lt0,lt1; boolean lcl;
            if r=-1 then
            begin lva:= rva; x:= x+h; mva:= fx; rpi:= rpi+s;
                  rva:= fa[rpi]; lpa:= lpi; fa[lpi]:= lva;
                  lpi:=lpi+s; fa[lpi]:= mva; lpi:=lpi+s;
                  n:=n+2;
                  if (lpi-rpi)×s>0 then goto full up;
                  t1:= (lva+rva)×.5; t0:= (t1+mva)×.5;
                  c:= (t0-t1)/3; t0:= t0+c;
                  if abs(c×ba) ≤ abs(t0×re)+ae then
                  begin tr:= abs(h)×t0; cl:= true; n:= n-2;
                        b:= n×hh; if abs(b) > hmin then
                        begin fa[lpi-s]:= a+b; fa[lpi]:= n;
                              lpo:= lpi:= lpi+s
                        end else lpi:= lpo; n:= 0; a:= x+hh
                  end else
                  begin tr:= 0; cl:= false end
            end else
            begin j:= j-r; r:= r-1; l:= l/2; left:= true;
                  rom;
            sr:   llp:= lpa; llv:= lva; ltr:= tr; lcl:= cl;
                  lt1:=t1; lt0:=t0; left:= false;
                  rom;
                  l:= l+1; r:= r+1; j:= j+r; lva:= llv;
                  lpa:= llp; s0:= (lva+rva)×.5;
                  t[j+r]:=
                  if left then (s0×.5) else (t[j+r]+s0×.5);
                  d:= 1; for i:= j+r-1 step -1 until j do
                  begin d:= d×4; s1:= s0; s0:= t[i-r];
                        c:= (s0-s1)/(d-1); s0:= s0+c;
                        t[i]:=
                        if left then (s0×.5) else (t[i]+s0×.5)
                  end;
                  d:= d×4; s1:= s0; s0:= (lt1+t1)×.5;
                  c:= (s0-s1)/(d-1); s0:= s0+c;
                  d:= d×4; t1:= s0; s0:= (lt0+t0)×.5;
                  c:= (s0-t1)/(d-1); t0:= s0+c;
                  if lcl∧cl then
                  begin tr:=tr+ltr; cl:=true end else
                  if abs(c×ba) ≤ abs(t0×re)+ae then
```

```
                begin tr:= abs(h)×t0×l; cl:= true;
                anew: n:= (lpa-lpo)×s; if n ≥ 0 then
                      begin b:= n×hh;
                            if abs(b) > hmin then
                            begin lpi:= lpa+s;
                                  fa[lpi]:= a+b;
                                  lpi:= lpi+s;
                                  fa[lpi]:= n;
                                  lpo:= lpi:= lpi+s
                            end else lpi:= lpo;
                            n:= 0; a:= x+hh
                      end else
                      begin   n:= fa[lpo-s]; b:= fa[lpo-s-s];
                              a:= b-n×hh; lpo:= lpo-(n+3)×s;
                              goto anew
                      end
                end else
                begin tr:=tr+ltr; cl:=false end
          end;
          if r= su then
          begin su:= su+1; if abs(rpo-rpi) ≥ 1 then goto sr
          end
    end;
    procedure suprom;
    if rpi+rpo then
    begin     r:= su:= -1; l:= 1; j:= 0; left:= true; rom;
              sum:= sum+tr; suprom
    end else if n+0 then
    begin     fa[lpi]:= rva; lpi:= lpi+s;
              b:= n×hh; if abs(b)>hmin then
              begin   fa[lpi]:= a+b; lpi:= lpi+s;
                      fa[lpi]:= n; lpo:= lpi:= lpi+s
              end else lpi:= lpo
    end;
    procedure subinterval;
    anew:     if rpo = 0 ∨ rpo= max then
    begin     if ¬ fr then
              begin   fr:= true; h:= -hh; hh:= -.5×hh; s:= -s;
                      lpi:= rpi; rpi:= lpo; lpo:= rpo; rpo:= rpi;
                      goto anew
              end
    end else
    begin     fr:= false;
              rpi:= rpi+s; n:= fa[rpi]; rpi:= rpi+s; a:= fa[rpi];
              rpi:= rpi+s; rva:= fa[rpi]; rpo:= rpi+n×s;
              n:= 0; x:= a-hh; suprom; goto anew
    end;
    ba:= b-a; if b<a then begin h:= b; b:= a; a:= h end;
    h:= abs(ba); hh:= h×.5; hmin:= abs(ba×re);
    x:= b; fa[max]:= fx; x:= a; fa[max-1]:= fx;
    fa[max-2]:= a; fa[max-3]:= 1;
    rpo:= rpi:= max-4; lpo:= lpi:= 0; s:= 1;
    sum:= 0; subinterval; integral:= sign(ba)×sum
end;
```

A FORMAL LANGUAGE TECHNIQUE APPLIED TO BUSINESS DATA PROCESSING

A. Introduction

The programming assignment called for an ALGOL-60 program to process a large amount of data (about 6 characters, punched on paper tape) into sales surveys to be output on a line printer. This report will only deal with the part of the program that interprets the input tape. The printed record of the input tape is (because of size and structure) unsuitable for visual check on mistakes.
The tape should produce numbers and these are of different levels: some numbers say something about a whole block of following numbers. These blocks are separated by delimiters, which are hierarchically related because several consecutive blocks may constitute a block of higher level.
At first an ad hoc description was tried for defining the structure of the input tape. After a time it became clear that it was impossible to foresee all permutations of available symbols and to state of each whether it could or could not be allowed. Because of mistakes made in punching the tape, it was to be expected that most of these permutations would occur in the long run.

Essentially, the problem posed by the input tape is that of the correct parsing of its punchings. The use of a context-free grammar for the input tape solved the following problems:

i. providing a concise and unambiguous description of the structure of the tape
ii. providing a framework for a program ('syntax-directed') that parses the tape and processes the information it conveys
iii. outputting a record of the way in which the program interprets the tape

B. Grammar

<tape>::= a|<date><districts>e<tape>
<date>::= d<hexuple>|<date><forgetit><date>
<districts>::= <empty>|<districtheading><visits><districts>
<districtheading>::= s<triple>|<districtheading><forgetit>
<districtheading>
<visits>::= <empty>|<hello><orders><goodbye><visits>
<hello>::= <minus><hexuple>|<hello><forgetit><hello>
<orders>::= <empty>|<positive><items>z<orders>
<items>::= <empty>|<positive><items>
<positive>::= <space><quadruple>|<positive><forgetit><positive>
<goodbye>::= tg|t<forgetit>
<triple>::= <digit><digit><digit>
<quadruple>::= <digit><triple>
<hexuple>::= <digit><digit><quadruple>
<digit>::= 0|1|2|3|4|5|6|7|8|9
<empty>::=
<number>::= <triple>|<quadruple>|<hexuple>
<delimiters>::= d|e|s|<minus>|<space>|<goodbye>

The notation used here is that of the 'Revised Report on the Algorithmic Language ALGOL 60'. The metalinguistic variables for which no production is given are directly represented on the tape.

C. Particulars about the program

The ALGOL-60 program shown below contains only the syntax-directed structure. Statements for processing the semantic content of the information received are summarized by assignments to the integer 'semantic'.
The program is rather loosely related to the grammar, but still closely enough to make it easy to check whether it will perform indeed as the grammar specifies. Departures from the correspondence between grammar and program have been instituted to avoid a proliferation of diminutive procedures. The procedure 'number' processes all productions of <number> and also <forgetit>. The procedure 'find' processes the productions of <delimiters>. 'read' is an integer procedure that delivers the value of the first punching on the tape that represents some metalinguistic variable.
The value of this punching is then stored in x and remains there until either the program achieves such a state that its value can be accepted or until its occurrence is found to be syntactically incoorect.
In the latter case symbols have, in general, to be skipped until the former case is found to be true.
Because the various metalinguistic variables envelop each other, the concomitant delimiters have a different level of priority.
The following are used:

a	6	z	2
d e	5	space	1
s	4	others	0
minus t	3		

A call of 'find' occurs where a certain delimiter is expected or possibly one of a priority not lower. Accordingly, it will have one (whichever situation is, after 0 or more skippings, encountered first) of the three following results:

i. A punching equal to the one sought is found. 'out' is called and execution of the next statement after 'find' follows.
ii. A punching unequal to the sought but of equal priority, is found. Jump to the first label follows.
iii. A punching unequal to the one sought, and of higher priority, is found. Jump to the second label follows.

'find' also takes care that skipped symbols are distinguishable from accepted symbols in the output.
In this way, the consequences of an error are restricted to the smallest syntactical entity containing that error. This entity may happen to be the largest. In that case all its constituent entitities would be skipped without being checked syntactically. This possibility is avoided by having 'find' call 'districts' or 'visits' as appropriate.
In this way we achieve that an entity may be checked syntactically even when contained within an incorrect entity.

C. Text of the program

```
begin   comment van Emden, 1609;
        integer i,j,x,a,d,e,forget it,g,s,t,z,space,
        minus,question mark, blank,semantic;
        integer array heptade[0:127]; boolean index;
        integer procedure read;
        begin   integer x,y;
                y:= heptade[REHEP];
                for x:= y while
                x= blank ∨ x= question mark do
                begin   if x= question mark then PRSYM(x);
                        y:= heptade[REHEP]
                end; read:= y
        end;
        procedure out;
        begin   if x = d ∨ x = s then NLCR;
                PRSYM(x); x:= read
        end;
        integer procedure number(n,exit); value n;
        integer n; label exit;
        begin   integer i,sum,tens;
        start:  sum:= 0; tens:= 10 ↑ (n-1);
                for i:= 1 step 1 until n do
                if x > 9 then goto exit else
                begin   sum:= sum + x × tens; tens:= tens/10;
                        out
                end;
                if x = forget it then
                begin   out; goto start end;
                number:= sum
        end;
        procedure find(p,exit0,exit1); value p;
        integer p; label exit0,exit1;
        begin   integer priox,priop;
                integer procedure priority(p); value p;
                integer p; priority:=
                if p = space then 1 else
                if p = z then 2 else
                if p = minus ∨ p = t then 3 else
                if p = s then 4 else
                if p = d ∨ p = e then 5 else
                if p = a then 6 else 0;
                if x ≠ p then
                begin   priox:= priority(x);
                        priop:= priority(p);
                        if priox = priop then goto exit0;
                        if priox > priop then goto exit1;
                        NLCR;
                        PRINTTEXT(‡er wordt gezocht.‡);
                next:   if x = s then districts else
                        if x = minus then visits else out;
```

```
                priox:= priority(x);
                if priox < priop then goto next;
                NLCR;
                PRINTTEXT(⟨einde overgeslagen⟩);
                NLCR;
                if x ≠ p then goto
                if priox = priop then exit0 else exit1
        end; out
end;
procedure items;
begin
start:  find(space,exit,exit);
        semantic:= number(4,start);
        goto start;
exit:
end;
procedure orders;
begin
start:  find(space,exit,exit);
        semantic:= number(4,exit1);
        items;
exit1:  find(z,exit,exit); goto start;
exit:
end;
procedure visits;
begin
start:  find(minus,goon,exit);
        semantic:= number(6,goon);
        orders;
goon:   find(t,start,exit);
        if x = g ∨ x = forget it then out else
        begin   NLCR; PRINTTEXT(⟨g of f ontbreekt⟩) end;
        goto start;
exit:
end;
procedure districts;
begin
start:  find(s,exit,exit);
        semantic:= number(3,start);
        visits; goto start;
exit:
end;
procedure tape;
begin   x:= read;
start:  find(d,goon,exit);
        semantic:= number(6,goon);
        districts;
goon:   find(e,start,exit);
        goto start;
exit:
end;
```

```
        question mark:= 122;
        for i:= 0 step 1 until 127 do
        heptade[i]:= question mark;
        blank:= -1; a:= 10; d:= 13; e:= 14;
        forget it:= 15; g:= 16; s:= 28; t:= 29;
        z:= 35; space:= 93; minus:= 65;
        index:= true;
        for i:=
        0,blank,
        97,a,
        100,d,
        117,e,
        118,forgetit,
        103,g,
        50,s,
        35,t,
        41,z,
        16,space,
        64,minus do
        begin   if index then j:= i else heptade[j]:= i;
                index:= ¬ index
        end;
        j:= 0;
        for i:= 32,1,2,19,4,21,22,7,8,25 do
        begin heptade[i]:= j; j:= j+1 end;
        tape
end
```