File under: A   Date: 22 Jan 1985   Name: John Strawn

Name: John Strawn

Project: J   Programmer: AWN

File Name: UDP2:SPLMER.SAI[LIB,AWN]

File Last Written: 21:49 21 Jan 1985

Time: 11:20   Date: 22 Jan 1985

Center for Computer Research
in Music and Acoustics
Department of Music
Stanford, California

```
COMMENT ⊗   VALID 00008 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00002 00002    ENTRY eachSplitMerge
C00003 00003    ∂ eachSplitMerge
C00007 00004    ∂             eachSplitMerge:  initialization
C00009 00005    ∂             eachSplitMerge:  SPLIT
C00012 00006    ∂             eachSplitMerge:  MERGE
C00015 00007    ∂             eachSplitMerge:  adjust, finish up
C00017 00008    END   "eachSplitMerge"
C00018 ENDMK
C⊗;
```

```
ENTRY eachSplitMerge;
BEGIN "eachSplitMerge"

DEFINE Dont_Require_approx_Dammit = TRUE;
REQUIRE "DSK:PR.HDR[lib,AWN]" SOURCE_FILE;
REQUIRE "DSK:CPR.hdr[lib,awn]" SOURCE_FILE;
```

```
∂ eachSplitMerge;

INTERNAL PROCEDURE eachSplitMerge(
   REFERENCE RECORD_POINTER (micro) head;
   REAL ARRAY func;
   INTEGER clock,compr,errNorm(integralErrorNorm);
   REAL threshold(.1);
   INTEGER datChan(unopenedChannel);
   BOOLEAN display(false)
);

begin "eachSplitMerge"

∂  called by:
   eachSplitMerge(line,head,errNorm,threshold,datChan,true);

∂  procedure useful in generating line segment approximations for functions.
   This routine accepts a function and attempts to provide initial "rough-
   guess" estimates of where breakpoints should be according to the
   "split-and-merge" algorithm given in
   Pavlidis, Theodosios, and S. L. Horowitz.  "Segmentation of Plane Curves."
   IEEE Transactions on Computers, Vol. C-23, No. 8, pp. 860-870, August,
   1974.
   (referred to as P/H in code below). This version implements only "case 1"
   of P/H.   See also explanation of threshold under inputs, below,
   for more information.
;

∂  coded by JMS April 1980, revised Nov 1980, Dec 1982;

∂ inputs:
   func[funcLower:funcUpper], funcLower and funcUpper determined by this procedure
      contains function to be approximated.  The points in the function
      are assumed to be equally spaced on the X axis.
   head will point to a linked list of records containing the breakpoints
      IF head points to a null record, then this procedure will approximate
         all of FUNC
      IF head points to a record, then the xBeg, xEnd values (see HEIR.HDR) in that
         record will be used as bounds within which to approximate func
   errNorm = maximum, integral, or mean, as per approx.hdr
   threshold (called EMAX in P/H) -- after this procedure is done, the
        error across EACH LINE SEGMENT is ≤ threshold
   display=true →→ show a plot of the function and of its curvature on
        the tty, and wait for crlf from user
   datChan≠false →→ output debugging and other data to some file
        already opened in mode '17;

∂ outputs:
   linked list of line segments, pointed to by head
;
```

```
∂           eachSplitMerge:  initialization;

INTEGER funcUpper, funcLower,
   pointNo, lineNo, funcNumPts, savLine,
   newPt, thisMaxErrPt, secondXMax;
REAL lSlope, thisErr;
STRING footnote;
BOOLEAN endPtChanged,dummy,success;
RECORD_POINTER(micro) rp, nextRp;

IFC includeDatChan THENC
IF datChan NEQ unopenedChannel THEN
   BEGIN
   OUT(datChan, ↓&↓&"***** DEBUGGING eachSplitMerge ****"&↓&↓);
   OUT(datChan, "threshold= "&cvf(threshold)&"  ");
   outErrNorm(errNorm,datChan)
   END; ENDC
getFuncBounds
funcNumPts←funcUpper-funcLower+1;

IF head = NULL_RECORD THEN
   BEGIN
   IFC includeDatChan THENC
   IF datChan NEQ unopenedChannel THEN OUT(datChan,"setting up head ..."); ENDC
   head ← NEW_RECORD(micro);
   micro:xBeg[head] ← funcLower;
   micro:xEnd[head] ← funcUpper;
   upDate(head,clock,compr,func);
   newArray(REAL,micro:errors[head],[maximumErrorNorm:MeanErrorNorm]);
   IFC includeDatChan THENC
   IF datChan NEQ unopenedChannel THEN OUT(datChan,"done"&crlf); ENDC
   END;

getMicroError(head,func);
```

```
∂            eachSplitMerge:  SPLIT;

   DO BEGIN "split/merge algorithm"
   endPtChanged←FALSE;
   rp ← head;
   lineNo ← 0;
   footnote←" (split)";

      DO BEGIN "step thru lines"
      IF escISeen THEN RETURN;
      lineNo ← lineNo+1;
      thisErr←micro:errors[rp][errNorm];
      IF thisErr > threshold THEN
         BEGIN "split" ∂ P/H step 1 and Rule A;
         IF micro:Flags[rp] LAND twoMax THEN
            BEGIN
            ∂ maximum error occurs at two different points, find the second;
            lSlope←(func[micro:XEnd[rp]] - func[micro:XBeg[rp]])/
                  (     micro:XEnd[rp]   -      micro:XBeg[rp]);
            secondXMax←micro:maxerrpt[rp];
            do secondXMax←secondXMax+1 until
               secondXMax≥micro:XEnd[rp] v
               (func[secondXMax]-
                  (func[micro:XBeg[rp]]+lSlope*(secondXMax-micro:XBeg[rp])))↑2
                  = micro:errors[rp][maximumErrorNorm];
            newPt ← floor((secondXMax+micro:maxerrpt[rp])/2);
            END
         ELSE newPt←floor((micro:XBeg[rp]+micro:XEnd[rp])/2);
         IFC includeDatChan THENC
         IF datChan NEQ unopenedChannel THEN OUT(datChan,↓&
            "splitting line no. "&cvs(lineNo)&" from xBeg= "&cvs(micro:XBeg[rp])&
               " to micro:XEnd= "&cvs(micro:XEnd[rp])&↓&
            tab&"at "&cvs(newPt)&" due to error of "&cvf(thisErr)&
              ", maximum error occurs at point no "&cvs(micro:maxerrpt[rp])&↓);
            ENDC
         success←micInsert(rp,clock,compr,func,newPt);
         endPtChanged←TRUE;

         IF display THEN listShow(head,func,compr,clock,footnote);
         IF ¬success THEN done "step thru lines";

         END "split"
      ELSE rp←micro:Next[rp];    ∂ P/H Remark after Rule A, p. 862;
      END "step thru lines" until rp = NULL_RECORD;
```

```
∂             eachSplitMerge:  MERGE;


    rp ← head;
    nextRp←micro:next[rp];
    IF nextRp=NULL_RECORD THEN done "split/merge algorithm";
    lineNo ← 0;
    footnote←" (merge)";
    WHILE nextRp≠NULL_RECORD DO
       BEGIN "merge" ∂ P/H step 2;
       IF esclSeen THEN RETURN;
       lineNo←lineNo+1;
       CASE errNorm OF
          BEGIN "find error"
          [maximumErrorNorm]  maxError(micro:xBeg[rp],micro:xEnd[nextRp],func,thisErr,thisMaxErrPt);

          [meanErrorNorm]     MSqError(micro:xBeg[rp], micro:xEnd[nextRp], func, thisErr);

          [integralErrorNorm] IntSqError(micro:xBeg[rp], micro:xEnd[nextRp], func, thisErr)
          END "find error";

       IFC includeDatChan THENC
       IF datChan NEQ unopenedChannel THEN
          OUT(datChan,↓&"lineNo= "&cvs(lineNo)&" thisErr= "&cvf(thisErr)); ENDC

       IF thisErr<threshold THEN
          BEGIN "delete breakpoint"
          endPtChanged←true;
          success←delMic(rp,clock,compr,func);
          IF ¬success THEN DONE "merge";
          nextRp←micro:next[rp];
          IFC includeDatChan THENC IF datChan NEQ unopenedChannel THEN OUT(datChan," breakpoint deleted
"&↓); ENDC
          END "delete breakpoint"
       ELSE
          BEGIN
          rp←micro:next[rp];
          IFC includeDatChan THENC
          IF datChan NEQ unopenedChannel THEN OUT(datChan," breakpoint not deleted"&↓); ENDC
          IF rp=NULL_RECORD THEN DONE "merge" ELSE nextRp←micro:next[rp];
          END;
       END "merge";

       IF display THEN listShow(head,func,compr,clock,footnote);
       IFC includeDatChan THENC
       IF datChan NEQ unopenedChannel THEN
          BEGIN
          OUT(datChan,↓&"lines after merge:"&↓);
          dumpList(rp,datChan);
          END; ENDC
```

```
∂            eachSplitMerge:  adjust, finish up;

   ∂ P/H step 3:                 ;
   dummy←adjustBreakpoints(head,func,errNorm,1);
   IF escISeen THEN RETURN;
   endPtChanged ← endPtChanged LOR  dummy;

   footnote←" (after adjustment)";

   IF display THEN listShow(head,func,compr,clock,footnote);
   IFC includeDatChan THENC
   IF datChan NEQ unopenedChannel THEN
      BEGIN
      OUT(datChan,↓&"lines after adjust:"&↓);
      dumpList(rp,datChan);
      END; ENDC
   END "split/merge algorithm" until ¬endPtChanged;  ∂ P/H step 4;


rp ← head; ∂ !! added 12/14/82;
WHILE rp NEQ NULL!RECORD DO
   BEGIN
   upDate(rp,clock,compr,func);
   rp ← micro:next[rp];
   END;

IFC includeDatChan THENC
IF datChan NEQ unopenedChannel THEN OUT(datChan,↓&"***** END debugging eachSplitMerge ****"&↓&↓); ENDC

END "eachSplitMerge";
END   "eachSplitMerge";
```

```
COMMENT ⊗   VALID 00004 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00002 00002    ENTRY adjustBreakpoints
C00003 00003    ∂ low level: adjustBreakpoints
C00017 00004    END "adjust"
C00018 ENDMK
C⊗;
```

```
ENTRY adjustBreakpoints;
BEGIN "adjust"

DEFINE Dont_Require_approx_Dammit = TRUE;
REQUIRE "dsk:pr.hdr[lib,awn]" SOURCE!FILE;
REQUIRE "dsk:cpr.hdr[lib,awn]" SOURCE!FILE;
```

```
∂ low level: adjustBreakpoints;

INTERNAL BOOLEAN PROCEDURE adjustBreakpoints (
RECORD_POINTER (micro) head;
REAL ARRAY func;
INTEGER errNorm(integralErrorNorm), numIter(-1),skip(1), datChan(unopenedChannel), display(false)
);

begin "adjustBreakpoints"

   ∂ Procedure for adjusting breakpoints of line segment approximations to
   a function, in order to minimize some error criterion
   Incorporates an algorithm taken from
   Pavlidis, Theodosios.  "Waveform Segmentation through Functional
      Approximation."  IEEE TRansactions on Computers, Vol. c-22, No. 7,
      July 1973, pp. 689-697.
   ∂ The "+1" in Pavlidis Eq. 5,6 is ignored in this implementation.  It can lead
     to spurious results in certain cases, e.g. where the point P-sub-j+1 in
     Pavlidis Eq. 6 is exactly the point of maximum error, and everything from
     P-sub-j+1 through u-sub-j-super-k is a straight line
   ;

   ∂ inputs:
     func       contains function being approximated
     head       points to linked list of breakpoints of class micro.
                Note that not just any class from hier.hdr can be used,
                because array errors from class micro is needed by this procedure.
     numIter    maximum number of iterations allowed
                if <0, not bounded...
     skip       is called M in Pavlidis' article.  In trying a new breakpoint
                to see if it would result in lower overall error, you
                move an old breakpoint by SKIP points.
     datChan    ≠false →→ output debugging data to some file open in mode '17.
     display    = true show progress of algorithm (for debugging only)
     errnorm    specifies error norm for optimizing line-segment fit to
                func. Used as subscript to array ERRORS in record_class LSEG.
                integralErrorNorm, maximum, and mean square error norms all result in
                exactly the same adjustments to the breakpoints in exactly the
                same number of steps
;

   ∂ outputs
                updated breakpoints, including error for each line segment
     returns TRUE if any breakpoints changed, else FALSE
;


   integer LineNo,iterNo;
   record_pointer (micro) thisLine,nextLine,thisTrial,nextTrial;
   ∂ read "thisTrial" as "current line, modified for trial";
   boolean odd,      ∂ marks evenOdd passes thru breakpoints;
      endPtChanged, ∂ = "flag" in Pavlidis' article;
      anyEndPtChanged; ∂ returned by procedure;

   ∂ the following two macros simply used for output;
   define output1=<begin
      out(datChan,↓&"iteration no. ="&cvs(iterNo)&", examining:"&↓&
         "No"&tab&"xBeg"&tab&"xEnd"&tab&"MaxErrPt"&↓&
         cvs(lineNo)&tab&cvs(micro:XBeg[thisTrial])&tab&cvs(micro:XEnd[thisTrial])&tab&
            cvs(micro:maxerrpt[thisTrial])&↓&
            "errors:"&tab&"max"&tab&tab&"meanErrorNorm"&tab&tab&"int"&↓&tab&
            cvf(micro:errors[thisTrial][maximumErrorNorm])&tab&
            cvf(micro:errors[thisTrial][meanErrorNorm])&tab&
            cvf(micro:errors[thisTrial][integralErrorNorm])&↓&
         tab&tab&cvs(micro:XBeg[nextTrial])&tab&cvs(micro:XEnd[nextTrial])&tab&
            cvs(micro:maxerrpt[nextTrial])&↓&tab&
            cvf(micro:errors[nextTrial][maximumErrorNorm])&tab&
            cvf(micro:errors[nextTrial][meanErrorNorm])&tab&
            cvf(micro:errors[nextTrial][integralErrorNorm])&tab&↓); end;>;
```

```
    define output2=<begin out(datchan,", new errors"&↓&"thisTrial: "&↓&tab&
            cvf(micro:errors[thisTrial][maximumErrorNorm])&tab&
            cvf(micro:errors[thisTrial][meanErrorNorm])&tab&
            cvf(micro:errors[thisTrial][integralErrorNorm])&↓&
         "nextTrial: "&↓&tab&
            cvf(micro:errors[nextTrial][maximumErrorNorm])&tab&
            cvf(micro:errors[nextTrial][meanErrorNorm])&tab&
            cvf(micro:errors[nextTrial][integralErrorNorm])&tab&↓); end;>;

IFC includeDatChan THENC
   IF datChan NEQ unopenedChannel then
      begin
      out(datChan,ffeed&"*****debugging adjustBreakpoints*****"&↓);
      outErrNorm(errNorm,datChan)
      end; ENDC
   anyEndPtChanged←endPtChanged←FALSE;

   if micro:Next[head] = NULL_RECORD then
      begin
        IFC includeDatChan THENC
      IF datChan NEQ unopenedChannel then out(datChan,↓&↓&
      "Only one line found, done debugging adjustBreakpoints"&↓); ENDC
      return(false);
      end;

   ∂ establish error for all breakpoints;
   thisLine←head;
   DO
   BEGIN
      getmicroError(thisLine,func,FALSE  );
      thisLine←micro:next[thisLine];
   END UNTIL thisLine=NULL_RECORD;

   thisTrial ← new_record(micro);
   nextTrial ← new_record(micro);
   newArray(<REAL>,<micro:errors[thisTrial]>,<[maximumErrorNorm:meanErrorNorm]>);
   newArray(<REAL>,<micro:errors[nextTrial]>,<[maximumErrorNorm:meanErrorNorm]>);

   odd←true;
   iterNo←0;
   do
   begin "evenOdd"
      IF esclSeen THEN RETURN(TRUE);
      if odd then
      begin
         lineNo←-1;
         iterNo←iterNo+1;
         thisLine←head;
         endPtChanged←false;
      end ELSE
      BEGIN
         thisLine←micro:next[head];
         lineNo←0;
      END;

      do
      begin "stepThruLines"
         nextLine←micro:next[thisLine];
         if nextLine=NULL_RECORD then done "stepThruLines";
         lineNo←lineNo+2;
         copymicro(thisTrial,thisLine);
         copymicro(nextTrial,nextLine);
         ∂ improvement in speed given at bottom of l.h. column p. 691 not yet implemented;
         ∂ how about a switch inhibiting moving breakpoint if
           micro:errors[line[lineNo][errNorm]=0?;
         IFC includeDatChan THENC
         IF datChan NEQ unopenedChannel then output1 ENDC
         if micro:errors[thisTrial][errNorm]≠micro:errors[nextTrial][errNorm] then
         begin "move breakPoint"
```

```
            if micro:errors[thisTrial][errNorm] >
               micro:errors[nextTrial][errNorm] then
            begin ∂ Pavlidis' article, eq. 3, p. 691;
               micro:XEnd[thisTrial] ← micro:XEnd[thisTrial] - skip;
               micro:XBeg[nextTrial] ← micro:XBeg[nextTrial] - skip;
            end ELSE
            begin ∂ Pavlidis' article, eq. 4;
               micro:XEnd[thisTrial] ← micro:XEnd[thisTrial] + skip;
               micro:XBeg[nextTrial] ← micro:XBeg[nextTrial] + skip;
            end; ∂ already checked for maxErr[thisTrial] ≠ maxErr[nextTrial] above;
            IFC includeDatChan THENC
            IF datChan NEQ unopenedChannel then out(datChan,"resetting middle breakpoint to "&
               cvs(micro:XEnd[thisTrial]));  ∂ ↓ inside output2; ENDC
            getmicroError(thisTrial,func); ∂ Pavlidis eq. 5;
            getmicroError(nextTrial,func); ∂ Pavlidis eq. 6;
            IFC includeDatChan THENC IF datChan NEQ unopenedChannel then output2 ENDC
            if (micro:errors[thisTrial][errNorm]
                   MAX micro:errors[nextTrial][errNorm])
             < (micro:errors[thisLine][errNorm]
                   MAX micro:errors[nextLine][errNorm])
               then begin "accept changed breakpoint" ∂ Pavlidis eq. 7;
               IFC includeDatChan THENC
               IF datChan NEQ unopenedChannel then out(datChan,"Accepting breakpoint"&↓); ENDC
               copymicro(thisLine,thisTrial);
               copymicro(nextLine,nextTrial);
               endPtChanged←true;
               IF display THEN listShow(head,func,1,1,"adjust, accepted breakpoint",
                  FALSE,"list.plt",datChan);
            end  "accept changed breakpoint";
         end "move breakPoint";
         if (thisLine←micro:next[thisLine]) = NULL_RECORD then
            done "stepThruLines" ELSE
            thisLine←micro:next[thisLine]; ∂ we've now just skipped over one;
      end "stepThruLines" until thisLine=NULL_RECORD;
      odd←¬odd;
      anyEndPtChanged←anyEndPtChanged LOR endptChanged;
   end "evenOdd"
∂  until (iterNo>numIter ∧ odd) ∨  odd ∧ ¬endPtChanged;
   until (odd ∧ iterNo > (IF numIter>0 THEN numIter ELSE iterNo)) ∨
         (odd ∧ ¬endPtChanged);

∂ get rid of testLine record space;
∂ aryel (memory[location (micro:errors[thisTrial])]);
∂ aryel (memory[location (micro:errors[nextTrial])]);
IF display THEN listShow(head,func,1,1,"final adjust",FALSE,"list.plt",datChan);
$recfn(5,thisTrial);
$recfn(5,nextTrial);
IFC includeDatChan THENC
IF datChan NEQ unopenedChannel then out(datChan,↓&"*****done debugging adjustbreakpoints*****"&↓);
ENDC

return(anyEndPtChanged);
end "adjustBreakpoints";
```

```
END "adjust"
```