KYBERNETIKA — VOLUME 8 (1972), NUMBER 5

Generalization of Pattern Recognition Method in Experiment Analysis

EVŽEN KINDLER

A computer method is described for an automatic approximation of experimental data by a sum of exponential functions one of which is negative and the other positive, under conditions of their initial values (the value and the derivative for the argument equal to zero) and under conditions that the exponential functions composing the sum are in their absolute values decreasing. The description of the corresponding method is given in SIMULA 67. The method has been implemented and tested in the Biophysical Institute of Charles University at the computer ODRA 1013 (programmed in a symbolic language).

1. FORMULATION OF THE PROBLEM

In the paper [1] a method has been presented which automatically approximates a set of experimental data by a sum of positive exponential functions

$$f(t) = \sum_{i=1}^{m} g_i e^{k_i t} \tag{1}$$

where $g_i > 0$ and $k_i < 0$ for all i, so that neither m nor any g_i and k_i are known a priori. At one hand, that problem is a relatively easy one, because one can have use of the fact that for $t \geqq 0$ the function (1) and its second derivative are positive while the first derivative is negative; at the other hand, that problem is a central one in the class of exponential analysis applied in the research of behaviour of linear systems. Namely in biophysics, nuclear medicine and radiology, which is the field which has implied the method presented here both the method presented in [1], the function (1) characterizes the clearing of a substance which has been put into a studied organism during a short moment (see [2], [3]).

Nevertheless the more detailed studying of components which compose the entier system, described ordinarily by the function (1), leads to more complicated problems. One of them is to approximate a set of experimental data by a positive sum of exponential functions so that in the initial time argument the first derivative is equal to

zero. Such a sum cannot be received by an algorithm published in [1], as then it would be

$$f'(0) = \sum_{i=1}^{m} g_i k_i < 0 . \tag{2}$$

Generally, some components would have to be positive while the other (one or more) negative. The non-computer-oriented investigators in the applying branches usually express their wish to determine the sum – if possible – so that only one component might be negative. This wish reflects the tendency of biomedical investigators to describe also the subsystems rather globally and uniformly.

Thus a problem has risen to determine a function

$$f(t) = \sum_{i=1}^{m} g_i e^{k_i t} + g e^{kt} > 0 \tag{3}$$

which approximates a set of experimental data so that $f(0) > 0$ is exactly given, $f'(0) = 0, f'(t) < 0$ for all $t > 0$, $g_i > 0$, $k_i < 0$ for all $i = 1, \ldots, m$, $k < 0$, $g < 0$, if neither g_i and g nor k_i and k are known, but also about m one knows only that it is a positive integer.

2. ANALYSIS OF THE TARGET FUNCTION

The negative component of (3) is determined by the initial conditions:

$$g = f(0) - \sum_{i=1}^{m} g_i , \quad k = -\frac{\sum_{i=1}^{m} g_i k_i}{g} \tag{4}$$

because

$$f(0) = \sum_{i=1}^{m} g_i + g \text{ and } f'(0) = \sum_{i=1}^{m} g_i k_i + gk.$$

Similarly as in [1] we can suppose that $k_1 \ll k_2 \ll \ldots \ll k_m < 0$ because from the commutativity of addition the ordering $k_1 < k_2 < \ldots < k_m < 0$ follows and if some k_i were very near to k_{i+1}, one could put $(g_i + g_{i+1}) . e^{k_i t}$ instead of $g_i e^{k_i t} + g_{i+1} e^{k_{i+1} t}$ and thus join the components not well distinguishable.

From the statements that $f'(0) = 0$ and $f'(0) < 0$ for all $t > 0$ a new statement follows that there is number $d > 0$ so that for all $t \in (0, d)$ the second derivative $f''(t)$ is negative. Moreover: if we subtract one or more positive components from f, the new function will have in the same interval $(0, d)$ its second derivative negative, because the subtracted components have their second derivatives positive (i.e. $g_i k_i^2 e^{k_i t}$).

Let us define the function $F(t) = \ln (f(t))$. Its two derivatives are

$$F' = \frac{f'}{f} \quad \text{and} \quad F'' = \frac{f'' f - f'^2}{f^2} . \tag{5}$$

The sign of F'' is the same as the sign of $f''f - f'^2$; evidently for $t \in (0, d)$ there is $F''(t) < 0$. One cannot determine generally whether the last relation holds for all positive t, as it will be illustrated in the appendix 1.

Finally, let us suppose that $f(0)$ is not too small. It is a suitable statement as otherwise the recognition of the exact forms in experimental data would not have meaning. For our purpose, we can express it as a hypothesis $f(0) > \sum_{i=1}^{m} g_i \left(1 - \frac{k_i}{k_1}\right) = \sum_{i=1}^{m} g_i c_i$. As $k_1 < k_2 < \ldots < k_m$, a relation holds that $0 = c_1 < c_2 < \ldots < c_m < 1$, so that $f(0)$ must be greater than a sum of certain positive fractions of $g_2, \ldots, g_m$. From this statement it follows:

$$\sum_{i=1}^{m} g_i \left(1 - \frac{k_i}{k_1}\right) - f(0) < 0 , \tag{6}$$

$$\frac{\sum_{i=1}^{m} g_i \left(1 - \frac{k_i}{k_1}\right) - f(0)}{\sum_{i=1}^{m} g_i \frac{k_i}{k_1}} < 0 , \tag{7}$$

$$\frac{\sum_{i=1}^{m} g_i \left(1 - \frac{k_i}{k_1}\right) - f(0) + \sum_{i=1}^{m} g_i \frac{k_i}{k_1}}{\sum_{i=1}^{m} g_i \frac{k_i}{k_1}} < 1 , \tag{8}$$

$$\frac{\sum_{i=1}^{m} g_i - \sum_{i=1}^{m} g_i \frac{k_i}{k_1} - f(0) + \sum_{i=1}^{m} g_i \frac{k_i}{k_1}}{\sum_{i=1}^{m} g_i \frac{k_i}{k_1}} < 1 , \tag{9}$$

$$\frac{\sum_{i=1}^{m} g_i - f(0)}{\sum_{i=1}^{m} g_i \frac{k_i}{k_1}} < 1 , \tag{10}$$

$$\frac{\sum_{i=1}^{m} g_i \cdot \frac{k_i}{k_1}}{\sum_{i=1}^{m} g_i - f(0)} > 1 , \tag{11}$$

$$\frac{-\sum_{i=1}^{m} g_i k_i}{\sum_{i=1}^{m} g_i - f(0)} > -k_1 , \tag{12}$$

$$-k > -k_1 , \tag{13}$$

$$k < k_1 . \tag{14}$$

Notes. The inverse relation (11) can be formed from (10), because the left hand term in (10) is not equal to zero; otherwise the whole target of the presented pattern recognition method would not have meaning, as the only one negative components would degenerate to zero. The transformation of (12) to (13) has been done according to (4).

From the relation (14) a possibility follows that we can join k to the ordered sequence of all k_i: $k < k_1 < k_2 < \ldots < k_m < 0$. This ordering enables to use the "peeling" method, used already in [1]: an exponential function $g_j e^{k_j t}$ with $k_j < 0$, added to similar exponential function $g_i e^{k_i t}$ or to a sum of such exponential functions, so that $k_i < k_j$, has the greatest influence to the sum for greater argument t, while for smaller arguments it can be neglected. For the logarithm of such a sum it holds that the values of it are nearer to a straight line with its first derivative equal k_j for arguments rather greater.

Let us mention that the formulation of the relation (6) is typical for heuristical recognition of forms; that relation cannot be tested before the algorithm because it needs the values determined during the algorithm run. The test whether the algorithm had recognized suitable forms is given only at its end: if $k < k_1$ the relation (6) has been satisfied and the results are suitable; if $k \geqq k_1$ the function (3) determined by the algorithm would not be a suitable approximation of the experimental data; simultaneously, the relation $k \geqq k_1$ is a signalization that the experimental data are not a good material for the recognition of the forms to which the presented algorithm has been designed. It would be the same situation as if an algorithm for recognizing of flying patterns were applied in other fields or if an algorithm for reading manuscripts in Latin letters were applied to read e.g. Arabian letters.

3. BASIC PRINCIPLES OF THE ALGORITHM

The mathematical properties mentioned in the preceding paragraph have given the following principle of the algorithm:

3.1. A pair of straight lines is generated so that one of them approximates the set of points obtained from the experimental data $\langle t_1, y_1 \rangle, \ldots, \langle t_n, y_n \rangle$ by transformation $\ln y_j \to y_j$, the other one approximates the same set of points with the exception of the first one.

3.2. A certain game is performed between the straight lines (see below). If the game is successful for the first line, that line is transformed to an exponential function and fixed as a component of the target sum and its values are subtracted from the experimental results.

3.3. If at least two results after the subtraction are positive the array of further points for the algorithm run is dilated to the only pairs with positive y_j and the algo-

rithm is iterated from the point 3.1. If one or no results are positive the negative component is formed according to the formulas (4) and the run of the algorithm is finished.

3.4. If in the game mentioned in point 3.2 the first straight line does not win the point with minimal t-coordinate is released from the consideration and the game is repeated.

The game is a net of much aspects (see its exact description in the following paragraph) but its main principle is that the first straight line must have at least once its first derivative less than that of the second straight line and at least one must then hold an inverse relation between both the derivatives.

The determination of the straight line is performed according to the same formulas as in the method [1] (see there the paragraph 3, page 204).

The operator can influence manually the game from the terminal and he can let print the information about the algorithm run. He does it by the same buttons as in [1] (paragraph 4.1, page 204 and 205). The function of the buttons is the same as there mentioned. Also the form of input data is the same as in [1] (paragraph 4.2, page 205 and 206), including an eventual input of g_i, k_i which are known before the algorithm run. After the printing of results one can manually modify the results by the same way as in [1], paragraph 4.2, page 206.

4. EXACT DESCRIPTION OF THE ALGORITHM

The algorithm is described in the same language SIMULA 67 [4] which has been used for the description of its predecessor in [1]. The reasons are the same, i.e. that language has suitable facilities for describing quasiparallel systems which are really formed during the algorithm run. Moreover, the reader can compare both the algorithms because in the present paper only the necessary modification of the algorithm presented in [1] have been done. All which is common for both the algorithms is written by equivalent SIMULA texts in both the papers. Nevertheless there is a lot of differences between both the algorithms so that it would not be readable to mention here only the modifications of the SIMULA text regarding to that or [1]. Special procedures for inputs, prints and buttons which have not been built into [4] have the same semantics as described in [1], paragraph 5, pages 209 and 210.

```
SIMSET begin
link class pair; begin real lower, upper; end;
set class sequence;
begin
ref (link) procedure order (n); value n; integer n;
begin integer i; ref (link) X; X :- first;
      for i := 2 step 1 until n do X := X. suc;
      order := X
end order;
```

```
real procedure func (z, b); value z, b; real z; boolean b;
    if empty then func := 0 else
    begin ref (link) X; real Y; Y := 0;
        for X := if b then first else progress. last,
        X. suc while X ≠ none do
        Y := Y + X . lower × exp (X . upper × z);
        func := Y;
    end;
end sequence;
ref (sequence) A, B, C, R; ref (pair) P, Q, NEG; ref (criterion) progress;
switch L: = constant step, variable step, data inside, known components, results, new action,
        derivative;
real x, y, der; integer K;
pair class component;
begin detach; G : K := K + 1;
if abs (1 -- C . order (K)) . upper < bound go to G;
begin real R, S, T, U, V; R := S := T := U := V := 0;
      for Q := C . order (K), Q . suc while Q ≠ none do
      if Q . upper > bound then begin
        y := 1/ln (Q . upper); T := T + 1; S := S + Q . lower; V := V + y;
        R := R + Q . lower × y; U := U + Q . lower ↑ 2 × y end;
      y := R ↑ 2 — U × V; if R = 0 ∨ y = 0 then go to D;
      upper := (T × R -- V × S)/y; lower := exp ((S — U × upper)/R);
      if button (3) then begin
        printline (from); print (K); text (to); print (C . cardinal);
        if button (2) then begin print (lower); print (upper) end
      end;
      if button (1) then display (false);
      E : resume (progress); go to G;
      D : printline (I have no exact values for the further approximation);
      lower := prec . lower; upper := prec . upper; go to E
end
end component;
boolean procedure button (n); value n; integer n;
begin ⟨see the explications in the text of this paper⟩ end;
procedure graph (b); boolean b;
begin newline; textiter (if b then ≮ = ≯ else ≮ — ≯, 65);
      R := if b then A else C; y := 60/R . first . upper;
      for P := R . first, P . suc while P ≠ none do cycle:
      begin newline; printspace (abc (entier (P . upper × y)));
      if b then text (*) else text (+); carriage return;
      printspace (abs (entier (B . func (P . lower, b) × y)));
      if b then text (□) else text (0) end cycle;
end graph;
procedure tabulate (b); value b; boolean b;
begin newline; R := if b then A else C;
      for P := R . first, P . suc while P ≠ none do cycle:
      begin real z; newline; print (P . upper);
      z := B . func (P . lower, b); print (z); print (P . upper — z); end cycle
end tabulate;
```

```
390   procedure display (b); value b; boolean b;
      begin if button (9) then graph (b);
            if button (10) then tabulate (b);
      end display;
      set class criterion;
      begin procedure inform;
            begin if button (5) then begin printline
                  (new fixed component:);
                  print (first . lower); print (first . upper) end;
                  display (true)
            end inform;
      detach;
      new component . into (this set);
      E: if C . cardinal ≦ K + 1 then resume (this SIMSET);
      M: new component . into (this set); S: resume (last);
      PREGAME: go to if C . cardinal ≦ K + 1 then E else S;
      if last . upper < first . upper then
        if button (6) then go to black else begin resume (last); go to PREGAME end;

      GAME:
      go to if first . upper < last . upper ∧ last . upper < 0 then black else white;
      white: if button (4) then
               if first . upper = last . upper then printline
                     (put out button 4)
               else begin printline
                     (I want to prolongate due to button 4);
               go to black end;
            inform; if button (4) then go to black;
      fix: for P := C . first, P . suc while P ≠ none do
           P . upper := P . upper − first . lower × exp (first . upper × P . lower);
           first . into (B); if button (8) then
           begin E: inspect C . order (K) when link do
           begin out; go to E end;
           F: inspect C . last when link do if upper < bound
             then begin out; go to F end
             otherwise resume (this SIMSET);
             inspect C . last . pred when link do if upper < bound then
             begin out; C. last. out; go to F end
             otherwise resume (this SIMSET)
           end button 8;
           K := 0; go to M;
      black: if button (6) then begin printline
             (I want to fix due to button 6); go to fix end;
             if C . cardinal ≦ K + 1 then
             begin if button (4) then printline
                   (I cannot reflect the button 4); printline
                   (the input curve is too short); go to fix
             end;
      first . lower := last . lower; first . upper := last . upper;
      resume (last); go to GAME;
```

```
disturb: printline
        (I try to fix the approximation from);
        input (K); K := K − 1;
new component . into (this set); new component . into (this set);
resume (first);
last . upper := first . upper; last . lower := first . lower;
go to if button (6) then black else S
end criterion;
A := new sequence; B := new sequence; C := new sequence; der := 0;
constant step : read (y);
for x := 0, x + y while A is sequence do
begin P := new pair; P . lower := x; read (P . upper); Q := new pair; Q . lower := P . lower;
      Q . upper := P . upper; P . into (A); Q . into (C)
end;
derivative: read (der); go to constant step;
variable step: P := new pair; read (P . lower); read (P . upper);
Q := new pair; Q . lower := P . lower; Q . upper := P . upper;
P . into (A); Q . into (C); go to variable step;
known components: P := new pair;
read (P . lower); read (P . upper); P . into (B);
for Q := C . first, Q . suc while Q ≠ none do
    Q . upper := Q . upper − P . lower × exp (P . upper × Q . lower);
go to known components;
data inside: K := 1;
analyzer: progress := new criterion; resume (progress);
if last . upper > 0 then printline
    (no more better results);
if button (7) then stop;
results: x := A . first . lower; y := der; inspect B when sequence do
begin for P := first, P . suc while P ≠ none do
        begin newline; print (P . lower); print (P . upper);
              x := x − P . lower; y := y − P . lower × P . upper
        end;
NEG . lower := x; NEG . upper := y/x; NEG . into (B); display (true) end;
NEG . out; stop;
begin comment: the following statements permit eventual modification of the results;
M: input (K); inspect B when sequence do
  begin if K = 0 then
    begin new pair . into (B); K := cardinal end;
    input (order (K) . lower); input (order (K) . upper); go to M end
end of modifications;
new action: P := C . first;
for Q := A . first, Q . suc while Q ≠ none do
    begin P . upper := Q . upper − func (Q . lower, true); P := P . suc end;
K := 0; go to analyzer
end program;
```

Note. The presented algorithm has been programmed in a machine dependent symbolic language for the same small drum-memory computer ODRA 1013, as the algorithm presented in [1] (see [5], [6]). The duration of all the programmed algo-

rithm runs have been very approximately to n minutes if n input data have been processed. If we have use of the computer small core memory (256 words) the runs have given the results always in one minute (prints excluded, as they can prolongate the computation dependently on the button-formulated demands of the user). Similarly as in [1], the size of the quasi-constant *bound*, empirically determined as 10^{-18}, has given suitable results.

APPENDIX 1

In the paragraph 2 the course of the function F has been partially studied (see the equations (5)). It has been proved the e that there is a positive d so that for all $t \in (0, d)$ the second derivative of F is negative.

(i) The statement that there exists a function F so that its second derivative is negative for all positive t is a consequence of the following theorem:

If $m = 1$ then the second derivative of F is negative always if $k \neq k_1$ and naturally if $g < 0, g_1 > 0$ and the function f has the course defined at the end of paragraph 1.

Proof. Let us write without indices $F(t) = \ln(g\mathrm{e}^{kt} + h\mathrm{e}^{ct})$ where $k, c, h < 0$ and $g > 0$. Naturally $g\mathrm{e}^{kt} + h\mathrm{e}^{ct}$ must be positive for all $t > 0$; this is satisfied if $k > c$ and $g > -h$. Then it holds

$$\operatorname{sgn}(F'') = \operatorname{sgn}(gk^2\mathrm{e}^{kt} + hc^2\mathrm{e}^{ct})(g\mathrm{e}^{kt} + h\mathrm{e}^{ct}) - (gk\mathrm{e}^{kt} - hc\mathrm{e}^{ct})^2$$

the left hand side can be algebraically modified by the following way:

$$g^2k^2\mathrm{e}^{2kt} + ghk^2\mathrm{e}^{(k+c)t} + ghc^2\mathrm{e}^{(k+c)t} + h^2c^2\mathrm{e}^{2ct} - g^2k^2\mathrm{e}^{2kt} - 2ghck\mathrm{e}^{(k+c)t} - \\ - c^2h^2\mathrm{e}^{2ct}$$

which is equal to

$$\mathrm{e}^{(k+c)t} \cdot gh(k - c)^2 .$$

The last expression is negative while the factor h is negative and all the other factors are positive.

(ii) The statement that there is a pattern of the function F so that its second derivative is positive for a certain value $t_0 > 0$ can be proved: we present an example of a function F so that F'' is positive for all t greater than certain $\bar{t}$; the example of such a function F is

$$F(t) = \ln f(t) \quad \text{where} \quad f(t) = \mathrm{e}^{-t} + \mathrm{e}^{-2t} - \mathrm{e}^{-3t} .$$

The function satisfies all the necessary properties formulated in the paragraph 1. Namely:

$$F(0) = 1 , \quad f'(0) = 0 \quad \text{and} \quad f(t) > 0 \quad \text{for all} \quad t \geqq 0 .$$

Moreover, $f'(t) < 0$ for all $t > 0$ because $f'(t) = -e^{-t} - 2e^{-2t} + 3e^{-3t} =$ $= -e^{-t} + e^{-3t} - 2e^{-2t} + 2e^{-3t} = -e(1 - e^{-2t}) - 2e(1 - e^{-t})$ and for $t > 0$ both $1 - e^{-2t}$ and $1 - e^{-t}$ are positive.

$$\operatorname{sgn}(F''(t)) = \operatorname{sgn}(f''(t) \,.\, f(t) - f'^2(t)) =$$

$$= \operatorname{sgn}\left[(e^{-t} + 4e^{-2t} - 9e^{-3t})(e^{-t} + e^{-2t} - e^{-3t}) - (e^{-t} + 2e^{-2t} - 3e^{-3t})^2\right] =$$

$$= \operatorname{sgn}(e^{-2t} + e^{-3t} - e^{-4t} + 4e^{-3t} + 4e^{-4t} - 4e^{-5t} - 9e^{-4t} - 9e^{-5t} + 9e^{-6t} -$$

$$- e^{-2t} - 4e^{-4t} - 9e^{-6t} - 4e^{-3t} + 6e^{-4t} + 12e^{-5t}) = \operatorname{sgn}(e^{-3t} - 4e^{-4t} - e^{-5t}) .$$

The argument of the *sgn* can be written as a product of a positive factor e^{-3t} with a factor $1 - 4e^{-t} - e^{-2t}$. The last factor is positive for $t \geqq 3$ because then

$$1 - 4e^{-t} - 2e^{-2t} \geqq 1 - \frac{4}{e^3} - \frac{1}{e^6} > 1 - \frac{4}{8} - \frac{1}{64} = \frac{1}{2} - \frac{1}{64} > 0 .$$

APPENDIX 2

We have got the statement that there is a positive d so that for all $t \in (0, d)$ the second derivative of F is negative, from the other properties of the function f, namely from the statement that $f'(0) = 0$. We can generalize the described algorithm so that it is valid for all the functions f of given form but independently on their value $f'(0)$ so that the natural logarithm of f must have its second derivative positive for t from a certain non-empty interval $(0, d)$. This property need not be immediately tested at the beginning of the algorithm run but it is tested at the end by the quality of the approximation. For this reason the algorithm has been generalized so that before or after all the data the value of the derivation of the target function for $t = 0$ can be read, preceded by the signal *7* followed by the mark (compare with the paragraph 4.2 in [1]). If the value is not read it is automatically assigned as zero. This facility is completely reflected in the SIMULA text in the paragraph 4: the switch by the sentinel *7* is reflected in the switch declaration of the identifier *L* by the identifier *derivative*. This label performs that the value prepared in the input unit is assigned for the identifier *der* which has been before that action assigned by zero.

(Received August 10, 1971.)

REFERENCES

[1] E. Kindler: Simple use of pattern recognition in experiment analysis. Kybernetika *5*, (1969) 3, 201—211.

[2] A. Rescigno, G. Segre: La cinetica dei farmaci e dei traccianti radioattivi. Edizioni universitarie, Boringheri, Torino 1961.

[3] C. W. Sheppard: Basic principles of the tracer method. Introduction to mathematical tracer kinetics. J. Wiley & Son, N. York—London 1962.
[4] O.-J. Dahl, K. Nygaard: SIMULA 67 common base definition. Norwegian computing centre. Oslo 1967.
[5] Automatic computer ODRA 1013 — General description (in Czech). Kancelářské stroje, n. p., Hradec Králové 1966.
[6] V. Černý, J. Půr: Programmers manual on automatic computer ODRA 1013 (in Czech). Kancelářské stroje, n. p., Hradec Králové 1967.

VÝTAH

Zobecnění metody rozpoznávání forem v analýze pokusů

EVŽEN KINDLER

V práci je popsána metoda pro strojové proložení součtu exponenciálních funkcí naměřenými hodnotami, jsou-li požadovány předem tyto vlastnosti: jeden člen součtu exponenciel musí být záporný, ostatní kladné, všechny členy musí být v absolutní hodnotě klesající funkce, proložený součet musí probíhat naměřenou hodnotou pro argument rovný nule a pro tentýž argument musí mít proložený součet danou hodnotu derivace. Metoda je jistým zobecněním metody použité při řešení jednoduššího problému v [1].

PhDr. RNDr. Evžen Kindler, CSc., Biofysikální ústav fakulty všeobecného lékařství Karlovy university (Biophysical Institute, Charles University), Salmovská 3, Praha 2.