KYBERNETIKA ČÍSLO 3, ROČNÍK 5/1969

Simple Use of Pattern Recognition in Experiment Analysis

EVŽEN KINDLER

A method is presented for automatic approximation of experimental data by a sum of exponential functions. The description of the corresponding program is given in SIMULA 67. The program has been implemented in the Biophysical Institute for the computer ODRA 1013 (programmed in a symbolic language).

1. INTRODUCTION

In the investigation of the kinetics of substance in living systems one needs often to characterize the structure of ways through which the substance is transported. The abstract systems reflecting the important properties of the transport of the substance are the compartmental systems (see [1], [2]). In the initial steps of the investigation the corresponding compartmental systems are supposed to be constant. The behaviour of such systems can be described by functions f of the form

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

where t is the system time, g_i and k_i are real constants and m is the number of compartments.

The methodology can be arranged so that all g_i are positive, their sum is equal to 1 and all k_i are negative. Thus the mathematical aspect of the whole investigating process can be described by the following phases:

1.1. Experimental data are given in the form of a sequence $A = \{\langle a_j, t_j \rangle\}_{j=1}^{n}$ of pairs, where $0 = t_1 < t_2 < \ldots < t_n$, all a_j are positive and $a_1 \doteq 1$.

1.2. A corresponding sequence $B = \{\langle g_i, k_i \rangle\}_{i=1}^{m}$ of pairs is found so that

$$\sum_{i=1}^{m} g_i e^{k_i t_j} \doteq a_j , \quad j = 1, 2, \ldots, n .$$

Neither the real numbers g_i, k_i nor the integer m are generally known a priori.

1.3. The properties of B are interpreted in the theory of compartmental systems. Namely, the investigated system is approximated by a constant compartmental system C composed of m compartments.

1.4. Special relations between the sequences A and B are investigated and the original system C is modified into a more realistic approximation D. This phase can be done by simulation. It can be completely based on the mental implement of compartmental systems, since there exist suitable simulation programming systems (COSMO – see [3] and MINICOSMO).

The present paper concerns the automatization of the phase 1.2. In order to get a suitable sequence B in case we do not know even the number m we could design various processes joined e.g. with the harmonical analysis or with Monte-Carlo Methods but they are relatively long as they consider a lot of general properties of the exponential functions which need not have any influence relating to all the four phases of the investigating process. The presented method has been programmed for a small automatic computer ODRA 1013 (see [4], [5]): the determination of a sequence B takes about n minutes where n is the length of the source sequence A. Suitable modifications of the computations are possible from outside: according to states of control buttons the computer can display the intermediate results and other information of its work and/or change its steps.

2. APPLIED METHOD

The applied method has a great use of the fact that the sequence B has to be a suitable approximation of the experimentally investigated reality. Thus the following consequences of the known properties of the exponential functions can be applied:

2.1. The sequence B can be ordered so that $k_i \leqq k_{i+1}$.

2.2. If $|k_i - k_{i+1}|$ is rather small the term $g_i e^{k_i t} + g_{i+1} e^{k_{i+1} t}$ can be approximated by $(g_i + g_{i+1}) e^{k_i t}$. Inversely, the last term implies a more simple approximation than the original two ones. Thus we can assume that in the sequence B all k_i are well distinguishable.

2.3. Then there exists a real number $s_m < t_n$ so that $\sum_{i=1}^{m-1} g_i e^{k_i t}$ is relatively small regarding to $g_m e^{k_m t}$ for all $t \geqq s_m$. In other words, for $t > s_m$ the function $\ln \sum_{i=1}^{m} g_i e^{k_i t}$ can be approximated by the linear function $k_m t + \ln (g_m)$.

2.4. The function $\sum_{i=1}^{m-1} g_i e^{k_i t}$ has the analogous properties as the original one. Thus the deduction of 2.3 can be iterated for the original m replaced by $m - 1$, $m - 2, \ldots, 1$.

The presented properties can be used for determining B from the sequence A. We describe the determination in geometrical terms as it has an analogy also in graphical manual methods for the same aim.

2.5. A linear function is found so that it approximates the last points of the set of points $\langle t_j, \ln(a_j)\rangle$. We can assume that there exists an integer s so that for all $j \geqq s$ the points $\langle t_j, \ln(a_j)\rangle$ are not distant from the straight line. The found linear function $f(t) = kt + q$ is transformed into an exponential function $e^f = ge^{kt}$ where $g = e^q$, and the pair $\langle g, k\rangle$ is included into the target sequence B. The function e^f is subtracted from the source function values and the whole process is repeated until the source function values are diminished to very small ones.

2.6. The described method can be *pure* or *contracted*. It is called pure if the length of the source sequence is not diminished after the subtraction. It is called contracted if the source sequence is shortened after each subtraction to its first $(s - 1)$ members: the values the logarithms of which are well approximated by a linear function are removed from the further computation. During the whole computing process, the source values which have been reduced to rather small ones (their logarithm would be either a negative number of a great absolute value or an imaginary one) are omitted in both the methods from the computation. After having done some experience with the described methods, we can state that the pure one is more efficient.

3. DETERMINATION OF THE STRAIGHT SEGMENT

To automatize the determination of the segment where its points would be well approximated by a linear function was the greatest problem. The first method which we tried to implement was that a linear function was determined by the least square method through $\bar{n}$ last points where $\bar{n} = 2$, $\bar{n} + 1$ while the differences did not overpass a certain tolerance (it entered into the computation as a member of input data). Thus the greatest $\bar{n}$ for which the differencies were inside the demanded tolerance was accepted and the corresponding straight line determined the function f of the par. 2.5. This method has failed in almost all the cases because the second or the third approximation has implied an overflow in the computer.

Therefore we found another method: we tried to determine the processes which take place in one's mind if he has to do an equivalent approximation graphically, on the semilogarithmic paper: the substantial aspect of that action can be expressed in the following statement: the sequence of points $\{\langle t_j, \ln(a_j)\rangle\}_{j=1}^{n}$ follows a curve the first derivative of which is negative, for smaller t_j increases and for greater t_j is constant. If we approximate the sequences $\{\langle t_j, \ln(a_j)\rangle\}_{j=K}^{n}$ and $\{\langle t_j, \ln(a_j)\rangle\}_{j=K+1}^{n}$ by two straight lines N and M respectively the derivative of the first one is less than the derivative of the second one for small K. If the derivative of N is greater than the derivative of M (or if they are equal) we can state that the points $\langle t_j, \ln(a_j)\rangle$ for $j = K, K + 1, \ldots, n$ follow a straight line.

The program uses thereby a method which forms the mentioned approximations for $K = 1, K + 1$ while the presented criterion is not satisfied.

Note. For the determination of the first component of the result function, the value of K is assigned as 2 because it is assumed that the first $a_j = a_1$ is equal to 1. Such a value cannot be applied in division by its logarithm, which is used, because we apply the least squares method with the weight $-1/\ln(a_j)$. The resulting formulas for determining the corresponding exponential function $y = be^{zt}$ are the following:

$$z = \frac{TR - VS}{R^2 - UV}, \quad b = e^{(S - zU)/R},$$

where

$$T = n - r + 1, \quad S = \sum_{j=K}^{n} t_j, \quad R = \sum_{j=K}^{n} \frac{t_j}{\ln(a_j)}, \quad V = \sum_{j=K}^{n} \frac{1}{\ln(a_j)}, \quad U = \sum_{j=K}^{n} \frac{t_j^2}{\ln(a_j)}.$$

4. MANUAL CONTROL

As the computing process has a rather clear geometrical sense for the users we have decided to enrich the implementation by possibilities that the computation might be influenced by the human's suggestions. It is possible through the buttons at the control desk and through the input data. Such a manipulation has demanded other facilities of the implementation, namely that certain information can be displayed of the computing run. The display of the information is either automatic (the computer writes of itself what it does) or it can be demanded by other buttons. The exact meaning of all the facilities is described in the next section. Their meaning expressed clearly and legibly for the non-computer-oriented user of the implementation is presented in the following paragraphs.

4.1. Buttons

Button 1 – print all the values of the exponential function which tries just to be a component of the target function. The form of the information is controled by the buttons 9 and 10.

Button 2 – print the coefficients of the exponential function mentioned at the button 1.

Button 3 – print the bounds of the interval inside which the present function (displayed eventually due to the buttons 1 and 2) is the approximation (thereby the value of r and n are printed, the last one need not be equal to the original one in case of the contracted method).

Button 4 – prolongate the process of increasing r regardess of the criterion presented in section 3.

Button 5 – print the coefficients of all the components fixed definitively in B; do so immediately during the computing run after the fixation is clear (note: after printing that information the present component can be immediately removed from the target sequence through the button 4).

Button 6 – understand the last approximation as a definitively fixed component of the target sequence, regardless of the criterion presented in the section 3.

Button 7 – stop before the final prints of the results after the computing process. During this interruption one can arrange the output unit, buttons 9 and 10 etc.

Button 8 – if it is in one of its two possible positions the pure method is applied otherwise the contracted one runs (see 2.6).

Button 9 – print the graph of the given function and of the approximation. If a non-definitive one is printed, the approximation is composed of the symbols 0 and the given curve of the symbols +. If a definitively fixed component is printed, the approximation is composed of the symbols □ and the given curve of the symbols *.

Button 10 – print three columns of values: the first column follows the given source values, in the second one there are the approximations and the third one contains the differences between the values occurring in the first column and in the second one. Each line corresponds to one t_j.

Notes. Dependently on the buttons 9 and 10 but regardless of the button 1 the graphs and/or the tables describing the results in the relation to the source information are printed at the end of the computation. They are always preceded by a table of all targets g_i, k_i. The buttons 9 and 10 can be in the active position independently. The buttons 4 and 6 have the opposite functions and thereby they cannot be in the active position simultaneously. During the run time one can however change the activation of them or passivate both of them (the program is made so that a short simultaneous activation of them during manual changing of their activation has no negative influence on the computation).

4.2. Input data

If the time step is constant it is perforated as the first information; it can be preceded by 1 followed by a mark (it is a special character which must not be as a delimiter between two numbers in the input medium). After the step, the values $a_1, a_2, \ldots$ follow; after a_n the integer 3 followed by the mark must take place as a sentinel.

If the time step is not constant the integer 2 followed by the mark initiates the input file. Then the values follow in the order $t_1, a_1, t_2, a_2, \ldots$ After a_n the sentinel 3 followed by the mark must be present.

In both the cases, before the sentinel 3 eventually known components of B can be put into the computation; they are considered as definitively fixed in B. Such

components are preceded by 4 followed by the mark; the components are put into the input medium in the order $g_1, k_1, g_2, k_2, \ldots$ After the last k_i the sentinel 3 mentioned already in the preceding paragraphs must be however present.

After the computation has been finished the program stops; if we let it rerun it reads the data which can modify the results; if we wish to modify the i-th component $\langle g_i, k_i \rangle$ of B we must record a triplet $\langle i, g, k \rangle$ into the input medium (g and k are the new values of g_i, k_i respectively); if we wish to add a new component ge^{kt} we must record an analogous triplet with $i = 0$. After all such triplets a sentinel 5 followed by the mark must be present. Then the program prints out all the components of the new sequence B and depending on the buttons 9 and 10 it can print the graphs or tables regarding to the new values. Then it stops and the modification may be repeated. If the sentinel 6 followed by the mark is present in place of the sentinel 5 the whole process of the analysis is repeated as the present results are the components a priori known.

In the Biophysical Institute the input is switched so that after the initiation (reading of the data until the sentinel 3) the input device is switched from a high speed photoelectric input unit to the teletype through which also the information of the current situation of the computation is printed. There can be applied an equivalent device, e.g. a typewriter.

Another facility exists in the Biophysical Institute implementation, which seems being rather suitable: the computer run (outside the initiation and outside the final printing of the results) can be stopped and by means of a special simple disturbation from the control desk (modification of the control register) it can be arranged so that it performs the present approximation beginning from the K-th point: if the computer is caused to rerun after the disturbation it demands itself the value of K through the teletype. If the button 6 is activated the component received is immediately fixed, otherwise the iteration goes on by its normal way. In the exact description presented in the following section this special action is called *disturbation*.

5. EXACT DESCRIPTION

There is a quasiparallel system (see [6]) composed of the main program, components and the criterion for fixation. The main program generates the criterion which generates new components in suitable moments of the run time. The control uses to be given to the generated components which try to approximate the points following the given values. The operation of the components is interrupted by the operation of the criterion: it is a set, containing generally two components, with a program body, where beside the generating of new components a very simple game is performed the target of which is the decision whether the resumed component is to be fixed.

The description is done in the language SIMULA 67 (see [7]) which has facilities

for description of quasiparallel systems and which seems to be comprehensible as it is an extension of ALGOL 60.

```
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; ref (criterion) progress;
switch L := constant step, variable step, data inside,
            known components, results, new action;
real x, y; integer K;
pair class component;
begin detach; G : K := K + 1;
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: if K = 1
          then begin text (no more better results); out;
                     resume (this SIMSET) end
          else begin printline
               (I have no exact values for a further approximation);
               lower := prec. lower; upper := prec.upper; go to E end
end end component;
boolean procedure button (n); value n; integer n;
begin ⟨see the note 1 after this SIMULA text⟩ end;
```

```
208   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 (abs(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 (○) 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;
      procedure display(b); value b; boolean b;
          if button (9) then graph (b);
          if button (10) then tabulate (b);
      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); resume (last);
      if C.cardinal ≦ K + 1 then resume (this SIMSET);
      M: new component.into (this set); S: resume (last);
      GAME:
      go to if first.upper < last.upper then black else white;
      white: if button (4) then 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)
          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); clear; 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;
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;
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 button (7) then stop;
results: inspect B when sequence do begin
for P := first, P.suc while P ≠ none do
begin newline; print (P.lower); print (P.upper) end;
display (true) end; stop;
begin comment: the following statements permit eventual
modifications 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 1. The semantics of applied printing and input procedures is related to e.g. a teletype or a typewriter:

text (*x*) — the text *x* in the brackets is printed.

newline — a new line is done in the printing unit, together with a carriage return.

printline **(*x*)** — it is the same as **begin** *newline*; *text* (*x*) **end.**

print (x) – print a space, the value of the actual parameter which must be a number and another space; in this paper the form of the printed real number is not important.

textiter (x, y) – the first actual parameter, which must be a symbol is printed y-times. For a better legibility, the symbol is closed into the string brackets ⊰ and ⊱.

printspace (x) – x spaces in printing, where x has a non-negative integer value.

carriage return – the following print is to begin from the left hand side of the present line.

read (x) – one item is read at the input unit; if it has a usual form (machine-dependent) of a number it is assigned for the actual parameter; if it has a form of an integer n followed by a mark (see 4.2) the jump to $L[n]$ is performed without any assignment.

input (x) – the function is the same as that of *read* but the data enter into the computer through a slow input unit with facilities for manual input and simultaneous printing of the entering information (the same unit as used for output is recommended).

The procedure *button* (x) is a boolean one whose value is **true** iff the x-th button is activated at the control desk. It seems to be suitable for small computers, while for large ones it seems to interpret this procedure as being switched according to the value of the parameter to various blocks in which more complicated criteria can occur. Formally the affair can be established so that instead of the presented *SIMSET* block the class declaration takes place initiated by the head:

SIMSET **class** *analyzer* (*button*);
virtual : **boolean procedure** *button*;

Note 2. The identifier *bound* means a metaconstant which corresponds to the smallest number the natural logarithm of which can be computed by the computer; in the Biophysical Institute the number 10^{-18} has appeared as sufficient.

Note 3. The reaction to the *disturbation* (see 4.2) is that an eventually operating component jumps to its label G and is resummed by *progress*. In the *progress* the jump to the label *disturb* is always performed.

(Received August 14th, 1968.)

REFERENCES

[1] A. Rescigno, G. Segre: La cinetica dei farmaci e dei traccanti radioattivi. Edizioni universitarie, Boringheri, Torino 1961.
[2] C. W. Sheppard: Basic Principles of the Tracer Method. Introduction to Mathematical Tracer Kinetics. J. Wiley & Son, N. York, London 1962.

[3] E. Kindler: COSMO (Compartmental System Modelling), Description of a Programming System. Simulation Programming Languages, Proceedings of the IFIP Working Conference on Simulation Programming Languages (editor J. N. Buxton). North-Holland Publishing Comp., Amsterdam 1968, 402—424.
[4] Automatic Computer ODRA 1013 — General description (in Czech). Kancelářské stroje, n.p., Hradec Králové, 1966.
[5] V. Černý, J. Půr: Programmer's Manual on Automatic Computer ODRA 1013 (in Czech). Kancelářské stroje, n.p., Hradec Králové, 1967.
[6] O.-J. Dahl, K. Nygaard: Class and Subclass Declarations. Simulation Programming Languages, Proceedings of the IFIP Working Conference on Simulation Programming Languages (editor J. N. Buxton). North-Holland Publishing Comp., Amsterdam 1968, 158—171.
[7] O.-J. Dahl, K. Nygaard: SIMULA 67 Common Base Definition. Norvegian Computing Center, Oslo 1967.

VÝTAH

Jednoduchá aplikace rozpoznávání obrazců v analýze pokusů

EVŽEN KINDLER

V práci je uvedena metoda pro strojové proložení součtu exponenciálních funkcí naměřenými hodnotami. Odpovídající program je popsán v jazyku SIMULA 67. Program byl realizován v Biofysikálním ústavu Karlovy university pro počítač ODRA 1013.

RNDr Evžen Kindler, Biofysikální ústav fakulty všeobecného lékařství Karlovy university, Salmovská 3, Praha 2.