Aplikace matematiky

Evžen Kindler

A heuristical algorithm for simple exponential analysis

Aplikace matematiky, Vol. 18 (1973), No. 6, 391–398

Persistent URL: http://dml.cz/dmlcz/103496

A HEURISTICAL ALGORITHM FOR SIMPLE EXPONENTIAL ANALYSIS

EVŽEN KINDLER

(Received July 3, 1972)

Let us consider a finite sequence of pairs $\{\langle t_i, y_i\rangle\}_{i=0}^{m}$ where m is an integer greater than 0. We can approximate the values y_i by $z_i = \sum_{j=1}^{n} g_j e^{c_j t_i}$. The problem to determine the values g_j and c_j for $j = 1, 2, \ldots, n$, and, if necessary, even n is called the exponential analysis. If neither n nor the statistical properties of the set $\{y_i\}_{i=0}^{m}$ are known the problem of the approximation has no meaning in the classical mathematical sciences. Then we call the corresponding algorithm a *heuristical* one. Usually the values c_j are to be negative, the values t_i are non negative. If it is known a priori that all the values of g_j must be positive then the exponential analysis is called *simple*. One can expect a suitable approximation in the simple exponential analysis only in case that a sufficient number of y_i are positive and that a sufficient number of pairs $\langle y_j, y_k\rangle$ satisfy the condition that $y_j > y_k$ iff $t_j < t_k$.

One can order the values entering into the algorithm so that if $j < k$ then $t_j < t_k$. Moreover, one can norm the values y_i (multiply them e.g. by a factor y_0^{-1}) so that there is a sufficient number of $y_i < 1$. (The obtained results, i.e., all g_j must be then divided by the same factor, e.g. multiplied by y_0.) Both the arrangements must be done if we want to use the present algorithm. Let us note that the words *sufficient number* etc. are rather unclear if we wish to solve a mathematically prepared problem but they are characteristical for the machine heuristics and artificial intelligence: if the uncelar conditions have been satisfied then and only then the results of the algorithm are acceptable; thus the concepts originally unexact turn to be exact parallely with any application of the algorithm, eventually their meaning is transformed to the semantics of other concepts used after the algorithm work (e.g. the concept of a suitable approximation).

The method applied in the algorithm is a game based on a comparison of the coefficients c of the exponential functions ge^{ct} which approximate certain sets of the pairs $\langle t_i, y_i\rangle$ or possibly $\langle t_i, x_i\rangle$ where x_i is a remainder of y_i after subtracting

certain terms. The game is performed by the algorithm but there are facilities that the user might use to enter into the game or to influence it. The complete game has been described in [1]. Certain patterns ge^{ct} are fixed during the game and they enter into the resulting sequence. The approximation by ge^{ct} is computed by the least square method with weight factors.

The algorithm is presented as a procedure written in ALGOL 60. Its formal parameters have the following meaning: c is an identifier of a one-dimensional array where the values of t_i come, and similarly d is a one-dimensional array where the values of y_i come; the identifier n denotes the same value as above; it is supposed that the values of t_i and y_i are assigned to $c[i]$ and $d[i]$ respectively. The final number of the components $g_j e^{c_j t}$ is substituted for the parameter m at the end of the algorithm run. The values of the coefficients g_j and c_j are stored at the end of the algorithm at the parameters $g[i]$ and $k[i]$ $(i = 1, 2, \ldots, n)$. The other parameters of the presented procedure have a different pragmatics than to be input or output values; they form parameters only because of formal reasons (in order not to infringe the rules of ALGOL 60); *line* and *licence* are constants determined more or less by the properties of the used computers: *line* is an integer equal to or less than the number of characters per one line (we have got clear graphical results using *line* = 60), and *licence* is a small positive real number which protects the computation against the overflows during division and drawing the logarithm and during further operations involving the results of the division and logarithms (we have got satisfactory results accepting *licence* = $= 10^{-18}$). The parameters *text*, *newline*, *carriage return*, *print* and *printspace* are the procedures for printing the results and the form of the game; we suppose that a terminal is used which prints characters one by one (e.g. teletype or a typewriter); *carriage return* causes that the terminal carriage returns without the linefeet, *newline* causes the return of the carriage together with the linefeet, *text* (a procedure with one string parameter) causes writing a string of characters, *printspace* (a procedure with one integer parameter i) causes that i spaces are printed and *print* (a procedure with one real parameter) causes the print of the value of its parameter; we do not specify in more detail the form of the printing but we have got good results when printing the integers-indices with one space before and one space after while the reals have been printed in 12 character places: one space, one place for the sign (minus or space), two places for the digits before the decimal point, one place for the decimal point, 6 places for the digits after the decimal point and one space after the printed number.

The formal parameter *button* represents an array of buttons which are placed at the terminal or at the control desk of the computer. Formally they can enter into the procedure as parameters (and so the algorithm can be tested in any implementation of ALGOL 60). But the main use of the buttons is that the user can change their position during the computation; this action cannot be written by the standard means of ALGOL 60; if we want to implement it we must transform the if-clauses of the form **if** *button* $[i]$ into special statements of the implementation. The buttons cause the following actions:

button [1]: for every exponential function which enters the game a table or a graph is printed which compares the modelled and the entering results; see *button* [8] and *button* [9].

button [2]: for every exponential function which enters the game its coefficients are printed.

button [3]: for every exponential function which enters the game the minimal and maximal indices are printed; they show the set which is approximated by the corresponding exponential function.

button [4]: the game is influenced so that if it is possible it goes on and no exponential function is fixed to be a component of the results. If the algorithm must nevertheless finish the game the information on it is printed.

button [5]: every component fixed as a new one of the results (at the end of a game) is printed.

button [6]: the game is influenced so that if it is possible it is terminated and the best exponential function of it is fixed as a component of the results. If the algorithm cannot fix any component, the information on it is printed.

button [7]: if it is in the position false the contracted form of the game is performed by the algorithm, otherwise the pure method is performed; both the forms are described in [1] namely in par. 2.6. We can illustrate simply the difference between the both forms so that the pure form tries to use the maximal number of the input data for the games while the contracted one tries to use the minimal number of them. The pure form runs longer and gives a finer approximation but we have used the contracted form more frequently when applying the algorithm.

button [8]: a table is printed; in the first column there are the modelled values (the approximation entering the game or fixed as a component of the results), in the second column there are the original values and in the third column their differences.

button [9]: a graph is printed; by crosses the original values are printed while the modelled values are printed by zeroes. Both the reactions to *button* [8] and *button* [9] are controlled according to the following rules: in case of a new component being fixed the positions of both the buttons are reacted; in case of forming a new exponential function entering the game the prints – though controlled by the buttons in question – are conditioned by the position of *button* [1] (see the description of its meaning).

procedure *KINDLER* 1 (*c*, *d*, *k*, *g*, *m*, *n*, *button*, *line*, *licence*, *newline*, *print*, *printspace*, *carriage return*, *text*); **value** *m*;

real array *c*, *d*, *k*, *g*; **real** *licence*; **integer** *m*, *n*, *line*; **boolean array** *button*;

procedure *newline*, *print*, *printspace*, *carriage return*, *text*;

```
begin integer i, j, e, f; real a, b, p, q, r, s, t, u, w, x, y; real array z[0 : m];
procedure h1;
begin comment it generates a new exponential function as an approximation of the
      pairs ⟨c[i], d[i]⟩ or ⟨c[i], z[i]⟩, where z[i] is a remainder of d[i]; the in-
      dices i are not less than f and not greater than e;
      if ¬button [3] then go to 5;
      newline; text('FROM'); print (f); text('TO'); print(e);
   5: w := r := s := t := u := 0;
      for i := f step 1 until e do
      begin y := z[i]; if y ≦ licence then go to 6;
            y := 1/ln(y); t := t + 1; s := s + c[i]; w := w + y;
            x := c[i] × y; r := r + x; u := u + c[i] × x;
   6: end i;
      x := r↑2; if x = 0 then go to 7;
      x := x − u × w; if x = 0 then go to 7;
      p := (t × r − w × s)/x; q := exp((s − u × p)/r);
      if ¬button [2] then go to 12; print(q); print(p);
  12: if ¬button [1] then go to 10; if ¬button [9] then go to 14; newline;
      for i := 1 step 1 until line do text('=');  x := z[0]/line;
      for i := 0 step 1 until e do
      begin newline; text('I'); printspace(abs(q × exp(p × c[i])/x)); text('O');
            carriage return; printspace(abs(z[i]/x)); text('X')
      end i; newline;
  14: if ¬ button [8] then go to 10;
      for i := 0 step 1 until e do
      begin newline; print(q × exp(p × c[i])); y := z[i]; print(y); print(x − y)
      end i; newline; go to 10;
   7: newline; text('IT IS STRANGE, ONLY ZEROES');
      if f = 0 then go to 3; p := a; q := b; f := f − 1;
  10: end h1;
procedure h2;
      begin x := 0; for j := 0 step 1 until n do x := x + g[j] × exp(k[j] × c[i])
      end h2;
procedure h3;
begin newline; for i := 1 step 1 until line do text('−'); s := d[0]/line;
      for i := 0 step 1 until m do
          begin newline; text(':'); h2; printspace(abs(x/s)); text('0'); carriage return;
                printspace(abs(d[i]/s)); text('+')
          end i
end h3;
```

```
procedure h4;
begin if ¬ button [5] then go to 13; newline;
      text('NEW COMPONENT'); print(g[n]); print(k[n]);
  13: if button [9] then h3; if button [8] then h5
end h4;
procedure h5; for i := 0 step 1 until m do
begin h2; newline; print(x); y := d[i]; print(y); print(x - y) end i;
  for i := 0 step 1 until m do z[i] := d[i]; n := 1; e := m; f := 1;
17: h1; if p ≧ 0 then go to 3; if e > 1 then go to 11; k[n] := p; g[n] := q; h4;
18: newline; text('RESULTS');
    for i := 0 step 1 until n do begin newline; print(g[i]), print(k[i]) end;
    if button [8] then h5; if button [9] then h3; go to 9;
 3: newline; text ('I CANNOT DO IT BETTER');
22: n := n - 1; go to 18;
11: f := f + 1;
24: a := p; b := q; h1;
    if p > a then go to 21; if ¬ button [4] then go to 4; f := f + 1;
    newline; text ('FOR THE BUTTON 4: I DO NOT FIX');
    if f < e then go to 24; f := f - 1; newline; text ('TAKE OFF BUTTON 4');
 4: k[n] := a; g[n] := b; h4; n := n + 1;
    if button [7] then f := m; e := f;
    for i := 0 step 1 until f do z[i] := z[i] - b × exp(a × c[i]);
15: if e ≦ 0 then go to 22; if z[e] > licence then go to 16; e := e - 1; go to 15;
16: f := 0; if z[e - 1] > licence then go to 17; e := e - 2; go to 15;
21: if ¬ button [6] then go to 26; if p ≧ 0 then go to 25; a := p; b := q;
    newline; text ('I FIX FOR THE BUTTON 6'); go to 4;
26: f := f + 1; if f < e then go to 24;
    if button [4] then text ('BUTTON 4 OFF'); go to 4;
25: if button [4] then text ('BUTTON 6 CONTRA 4'); go to 4;
 9: end KINDLER 1;
```

The algorithm has been programmed in the language MOST which is a certain dialect of a very limited ALGOL 60 (see [3]). As the presented form has followed the original text as much as possible, we can observe that it uses only minimal facilities of ALGOL 60. Namely: all the identifiers needed in the algorithm are specified and declared in the beginning of the algorithm; the variables are identified by only one letter (except those presenting the metaconstants); labels are integers; there are no sub-blocks of the the main block; the procedures are without parameters and their identifiers have

the united form of the letter *h* followed by an integer; the name of the algorithm and the strings are in capital letters while the variables and the procedures used in the program are in small letters.

The algorithm has been tested at the computer ODRA 1013. It is a small computer with its central memory in a magnetic drum, completed by a small memory (its capactiy being 256 machine words) realized in cores [2]. The run time of the algorithm is substantially influenced by the time for the prints: if we have used the buttons so that there were only prints of the results, the time necessary for the computation and prints would be about one minute for $m = 8$ and about 4 minutes for $m = 25$. Simulating the computer ODRA 1003, which has only the drum memory and performs about 70 operations per second we have obtained a suitable approximation of the run time: the algorithm runs m minutes (where m is the number of input pairs).

Let us make a note on the relation between the presented algorithm and the description of its game in SIMULA 67 in [1]. The original implementation has been in the machine code at the mentioned computer. One could use the wired facilities as indexregisters, maskering, address substitution and their chains; there was no suitable possibility to transform them into any language of the first or the second generation, but the language SIMULA 67 with its facilities of remote identifying, structure definitions and quasiparallel programing was well prepared to express the used hardware facilities (see [4]). Later we have tried to program the algorithm in the algorithmic language MOST. It was necessary to eliminate the chains: in order not to prolongate the run time it was necessary to reorganize the algorithm techniques, to assign the values accessible through long chains to auxiliary identifiers and to minimize the paths of the game. The resulting algorithm presented here needs approximately the same run time, it can be written in ALGOL 60 but it is rather undecipherable: one cannot join comments to it which might characterize the meaning of various statements in simple words.

It is interesting that if one uses the classical mathematical methods to solve the same problem on the same computer one needs not only a more detailed mathematical information about it (the statistical properties or the number n of the components) but also more computer time: if we compute by the gradient method or by the least square methods we need about 50 times more computer time. One may hope that this comparison holds also for other computers.

The algorithm has been invented as a part of the effort to equip the computer with such a software which would not demand the user to express his problems mathematically; the users of computers who are not mathematicians often base their computations on contradictory axioms risen by abstraction of various physical properties. The presented algorithm can diminish the number of components according to errors of measurement in case that it has been originally determined great rather by a very fine physical properties which have not been reflected in the measurement (about that methodology of modelling, see [6]). Naturally, one can join any iterative method, which would modify the results of the presented algorithm to be more

exact according to certain criterion. The results of the presented algorithm, used as the first approximation, lead then to absolutely (and not only locally) optimal results and give to the iterative methods necessary input data (e.g. the number of components).

In Table 1 we can see an example. The first column contains the values of t_i while the second one contains the values of y_i which enter the algorithm. The results are

$$z(t) = 0{\cdot}29291e^{-0.40921t} + 0{\cdot}448e^{-3.34576t} + 0{\cdot}25908e^{-14.98461t}.$$

Tab. 1.

t_i	y_i	z_i	$z_i - y_i$	i
0·000	1·000	1·00000000	0·00000000	0
0·125	0·613	0·61300000	0·00000000	1
0·250	0·402	0·46464430	0·06264430	2
0·500	0·335	0·32295292	−0·01204708	3
1·000	0·182	0·21033030	0·02833030	4
2·000	0·129	0·12976795	0·00076795	5
4·000	0·086	0·05699932	−0·02900068	6
5·000	0·023	0·03785696	0·01485696	7
7·000	0·017	0·01669966	−0·00030034	8

The third column of Table 1 presents the computed approximations of y_i, the fourth column of the same table contains the differences between the original values and the computed ones. The reader can find more information in the article [5], which – in spite of its radiobiological specialisation – offers a series of 16 problems solved by the algorithm completed by reproductions of computer prints corresponding to various buttons.

References

[1] *E. Kindler:* Simple use of pattern recognition in experiment analysis. Kybernetika 5, No. 3, pp. 201–211, ACADEMIA, Prague 1969.

[2] Automatic computer ODRA 1013 – General description (in Czech), Kancelářské stroje, Hradec Králové, 1966.

[3] *J. Szczepkowicz:* Programming in the autocode MOST 1 (in Polish), Wroczlaw, Elwro, Publication 03-VI-1.

[4] *O. J. Dahl K. Nygaard:* SIMULA 67 common base definition. Norvegian computing center, Oslo 1967.

[5] *E. Kindler F. Vítek:* Automatic preparation of computer models. Acta Universitatis Carolinae medica 16, No 3/4, pp. 261–280, 1970.

[6] *E. Kindler:* L'intelligence artificielle et détermination de modèle (in French). Materials of the International congress on natural and artificial intelligences (Nice, 1971). Published in REMESTA 1972/4, pp. 16–27, Prague 1972.

Souhrn

HEURISTICKÝ ALGORITMUS PRO JEDNODUCHOU ANALÝZU EXPONENCIÁLNÍCH FUNKCÍ

EVŽEN KINDLER

V článku je popsán v jazyce ALGOL 60 algoritmus, který zpracuje množinu bodů tak, že ji aproximuje funkcí $\sum_{j=1}^{n} g_j e^{k_j t}$, kde t je nezávisle proměnná. Algoritmus určí nejen hodnoty všech g_j a k_j, ale i číslo n. Výsledky jsou následujících vlastností: všechna g_j jsou kladná a všechna k_j jsou záporná. Vzhledem k tomu, že algoritmus nepotřebuje ani počet členů n ani statistické vlastnosti vstupujících hodnot, nejde o algoritmus statistiky, nýbrž oboru rozpoznávání forem; algoritmus napodobuje heuristickou práci výzkumníka, který analýzuje naměřené hodnoty graficky, aby získal vhodný matematický model (včetně jeho struktury) zkoumaného systému. Modelovaná inteligence algoritmu počítá s možností synthésy s podněty uživatele, který pomocí vnějších zásahů (realizovaných tlačítky na terminálu počítače) může ovlivnit hru, která je v algoritmu simulována.

Author's address: PhDr. RNDr. *Evžen Kindler* CSc., Biofysikální ústav Karlovy university, Salmovská 3, 120 00 Praha 2.