ALGORYTMY

VOL. IV • NO. 7 • 1967

INSTYTUT MASZYN MATEMATYCZNYCH

A L G O R Y T M Y

Vol. IV N°7 1967

P R A C E

Instytutu Maszyn Matematycznych

Adres redakcji: Warszawa, ul. Koszykowa 79, tel.28-37-29

TREŚĆ

CONTENTS

Metody numeryczne równań różniczkowych
Numerical analysis of differential equations

J. Wilczkowski
PROGRAM NUMERYCZNEGO CAŁKOWANIA RÓWNANIA DRGAŃ BELKI 7

K. Moszyński
O PEWNYM WIELOPOZIOMOWYM MODELU KRÓTKOTRWAŁEJ PROGNOZY POGODY 29

Metody statystyczne
Statistical methods

L. Łukaszewska, E. Pleszczyńska
GENEROWANIE REALIZACJI PROCESU POISSONA ZE ZMIENNĄ INTENSYWNOŚCIĄ 49

J. Winkowski
ON CERTAIN BIRTH AND DEATH PROCESSES AND THEIR SIMULATION 61

Teoria programowania
Theory of programming

L. Czaja, P. Szorc
STORAGE ALLOCATION FOR ALGOL 77

L. Czaja, P. Szorc
IMPLEMENTATION OF ALGOL FOR ZAM COMPUTERS 91

K. Moszyński, R. Pogorzelski
THE STRUCTURE OF PROCEDURES ADJOINED TO THE ALGOL SYSTEM FOR ZAM COMPUTERS 113

P. Byrniev i drug.
SISTEMA AVTOMATIČESKOGO PROGRAMMIROVANIJA M I K O D DLA MAŠINY MIŃSK-2 .. 127

P. Byrniev, D. Toszkov
KOMPILIRUJUŠČAJA SISTEMA M I K S DLA ISPOLZOVANIJA BIBLIOTEKI STANDARTNYCH PROGRAM DLA MAŠINY MIŃSK-2 131

NUMERICAL ANALYSIS OF DIFFERENTIAL EQUATIONS

Instytut Maszyn Matematycznych
Algorytmy N° 7

518.12:517.946:624.072.2:534.1

PROGRAM NUMERYCZNEGO CAŁKOWANIA RÓWNANIA DRGAŃ BELKI

Jerzy WILCZKOWSKI

Pracę złożono 19.4.1966 r.

Artykuł poświęcony jest organizacji procedury do rozwiązywania równania drgań belki. Zastosowano metodę prostych sprowadzając problem do zagadnienia Cauchy'ego dla układu równań różniczkowych zwyczajnych. Dowód zbieżności rozwiązania przybliżonego do dokładnego dla równania liniowego hiperbolicznego II-go rzędu podał m.in. Lebiediev [4], Budak [5]. Można go uogólnić dla równania liniowego drgań belki. Analogiczny dowód dla równania rozpatrywanego poniżej nie jest autorowi znany, dlatego też przytoczoną procedurę należy stosować ostrożnie.

1. SFORMUŁOWANIE ZAGADNIENIA

Rozpatrzmy nieliniowe równanie drgań belki o długości l w postaci

$$\frac{\partial^2 u}{\partial t^2} + a(t,x)\,\frac{\partial^4 u}{\partial x^4} = f\left(t,x,u,\frac{\partial u}{\partial t},\frac{\partial u}{\partial x},\frac{\partial^2 u}{\partial t\partial x},\frac{\partial^2 u}{\partial x^2},\frac{\partial^3 u}{\partial x^3}\right) \qquad /1/$$

z warunkami początkowymi

$$\begin{cases} u(0,x) = g_1(x), \\ \\ \dfrac{\partial u}{\partial t}(0,x) = g_2(x), \end{cases} \qquad 0 \leqslant x \leqslant l, \qquad /2/$$

oraz brzegowymi, zapisanymi w postaci

$$\sum_{\sigma=0}^{3} \alpha_{\sigma,\mu} \frac{\partial^{\sigma} u}{\partial x^{\sigma}}(t,0) = \beta_{\mu}(t), \qquad \mu = 1,2 \text{ – dwa warunki na brzegu } x = 0,$$

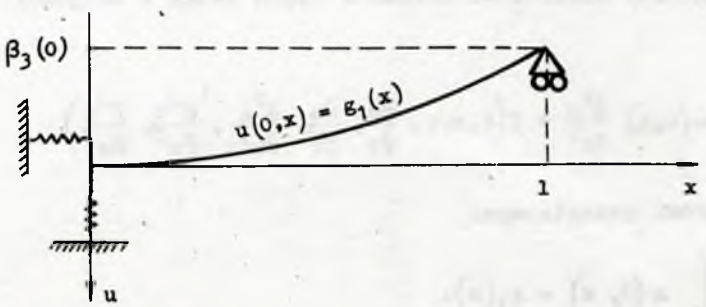
$$\sum_{\sigma=0}^{3} \alpha_{\sigma,\mu} \frac{\partial^{\sigma} u}{\partial x^{\sigma}}(t,1) = \beta_{\mu}(t), \qquad \mu = 3,4 \text{ – dwa warunki na brzegu } x = 1, \qquad /3/$$

przy czym dla $\sigma = 0$ należy położyć $\frac{\partial^{\sigma} u}{\partial x^{\sigma}} = u$.

Warunki /3/ obejmują najczęściej spotykane przypadki. Rozpatrzmy dla przykładu kilka z nich. Przyjmijmy nazywać brzeg $x = 0$ lewym, zaś brzeg $x = 1$ prawym końcem belki.

Przykład 1

Belka o lewym końcu sprężyście zamocowanym i prawym przegubowo podpartym. Niech przy tym podpora wykonuje ruch opisany funkcją $\beta_3(t)$. Musi tu być spełniony warunek $\beta_3(0) = g_1(1)$, oraz $\dot{\beta}_3(0) = g_2(1)$.



Rys. 1.

Warunki brzegowe mają tu postać

$$\begin{cases} b\dfrac{\partial u}{\partial x}(t,0) - \dfrac{\partial^2 u}{\partial x^2}(t,0) = 0 \\ eu(t,0) + \dfrac{\partial^3 u}{\partial x^3}(t,0) = 0 \end{cases} \qquad \begin{cases} u(t,1) = \beta_3(t) \\ \dfrac{\partial^2 u}{\partial x^2}(t,1) = 0 . \end{cases} \qquad /4/$$

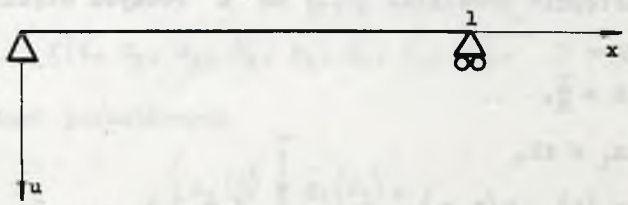
Należy więc tu przyjąć

$$\begin{cases} \alpha_{11} = b, \quad \alpha_{21} = -1, \quad \alpha_{02} = e, \quad \alpha_{32} = 1, \\ \alpha_{03} = 1 = \alpha_{24}, \\ \beta_3 = \beta_3(t). \end{cases} \qquad /5/$$

Pozostałe współczynniki i funkcje β_μ występujące w /3/ są równe zeru.

Przykład 2

Belka na obu końcach przegubowo podparta.



Rys. 2.

Warunki brzegowe:

$$\begin{cases} u(t,0) = 0, \\ \dfrac{\partial^2 u}{\partial x^2}(t,0) = 0, \end{cases} \qquad \begin{cases} u(t,1) = 0, \\ \dfrac{\partial^2 u}{\partial x^2}(t,1) = 0, \end{cases} \tag{6}$$

zatem

$$\begin{cases} \alpha_{01} = 1 = \alpha_{22}, \\ \alpha_{03} = 1 = \alpha_{24}, \end{cases} \qquad \text{pozostałe - zera.} \tag{7}$$

2. METODA NUMERYCZNA

Oznaczmy

$$\frac{\partial u}{\partial x} = p, \quad \frac{\partial^2 u}{\partial t \partial x} = q, \tag{8}$$

$$\frac{\partial^2 u}{\partial x^2} = r, \quad \frac{\partial^3 u}{\partial x^3} = s.$$

Podzielmy następnie przedział $[0,1]$ na n równych części oznaczając:

$$\begin{cases} k = \frac{1}{n}, \\ x_i = ik, \\ u_i(t) = u(t,x_i), \\ a_i(t) = a(t,x_i), \end{cases} \quad i = 0,1, \ldots, n,$$

$$\dot{u}_i(t) = \left.\frac{\partial u(t,x)}{\partial t}\right|_{x=x_i}, \qquad i = 1,2, \ldots, n-1.$$

Zamieńmy w końcu w /8/ pochodne względem x odpowiednimi wyrażeniami różnicowymi i oznaczmy

$$\begin{cases} p_i = \frac{1}{2k}(u_{i+1} - u_{i-1}) \\ q_i = \frac{1}{2k}(\dot{u}_{i+1} - \dot{u}_{i-1}) \\ r_i = \frac{1}{k^2}(u_{i+1} - 2u_i + u_{i-1}) \\ s_i = \frac{1}{2k^3}(u_{i+2} - 2u_{i+1} + 2u_{i-1} - u_{i-2}) \end{cases} \quad /9/$$

Wprowadzając tak określone argumenty funkcji f, oraz przybliżone wyrażenie na czwartą pochodną

$$\frac{\partial^4 u}{\partial x^4} \approx \frac{1}{k^4}(u_{i+2} - 4u_{i+1} + 6u_i - 4u_{i-1} + u_{i-2})$$

do równania /1/ otrzymamy następujący układ (n-3) równań różniczkowych zwyczajnych, określający na prostych $x = x_i$ funkcje $u_i(t)$:

$$\ddot{u}_i = -\frac{a_i(t)}{k^4}(u_{i+2} - 4u_{i+1} + 6u_i - 4u_{i-1} + u_{i-2}) + f(t, x_i, u_i, \dot{u}_i, p_i, q_i, r_i, s_i), \quad i = 2,3, \ldots, n-2, \quad /10/$$

z warunkami początkowymi

$$\begin{cases} u_i(0) = g_1(x_i), \\ \dot{u}_i(0) = g_2(x_i), \end{cases} \quad i = 0,1, \ldots, n. \quad /11/$$

Dla określenia występujących w układzie /10/ funkcji u_0, u_1, u_n, u_{n-1}, $\dot{u}_1$, $\dot{u}_{n-1}$ skorzystamy z warunków brzegowych /3/. Weźmy pod uwagę warunki na lewym końcu. Zastępując pochodne wyrażeniami różnicowymi możemy je napisać następująco /różnice brane są tu "w przód"/:

$$\alpha_0 \mu u_0 + \alpha_1 \mu \frac{u_1 - u_0}{k} + \alpha_2 \mu \frac{u_2 - 2u_1 + u_0}{k^2} +$$

$$+ \alpha_3 \mu \frac{u_3 - 3u_2 + 3u_1 - u_0}{k^3} = \beta\mu, \qquad \mu = 1,2,$$

lub po przekształceniach

$$u_0 A_\mu + u_1 B_\mu = W_\mu, \qquad \mu = 1,2, \qquad /12/$$

gdzie

$$\begin{cases} A_\mu = \alpha_0 \mu k^3 - \alpha_1 \mu k^2 + \alpha_2 \mu k - \alpha_3 \mu \\ B_\mu = \alpha_1 \mu k^2 - 2\alpha_2 \mu k + 3\alpha_3 \mu \\ W_\mu = \beta_\mu k^3 - u_2(\alpha_2 \mu k - 3\alpha_3 \mu) - u_3 \alpha_3 \mu, \qquad \mu = 1,2. \end{cases} \qquad /12'/$$

Związki /12/ stanowią układ równań liniowych, z których możemy wyrazić u_0, u_1 jako liniowe funkcje u_2, u_3. Analogiczny układ otrzymamy dla prawego końca:

$$u_n A_\mu + u_{n-1} B_\mu = W_\mu, \qquad \mu = 3,4, \qquad /13/$$

gdzie A_μ, B_μ określone są identycznie jak w /12/, zaś

$$W_\mu = \beta_\mu k^3 - u_{n-2}(\alpha_2 \mu k - 3\alpha_3 \mu) - u_{n-3} \alpha_3 \mu, \qquad /13'/$$

stąd zaś możemy wyrazić u_n, u_{n-1} poprzez u_{n-2}, u_{n-3}.

Pochodne $\dot{u}_1$, $\dot{u}_{n-1}$ wyznaczymy z równań /12/, /13/ po zastąpieniu występujących tam funkcji u, β ich pochodnymi $\dot{u}, \dot{\beta}$. Na przykład dla warunków brzegowych określonych przez /4/ mamy: na lewym końcu / $\mu = 1,2$/:

$$\alpha_{11} = b, \quad \alpha_{21} = -1, \quad \alpha_{02} = e, \quad \alpha_{32} = 1,$$

wobec tego

$$\begin{cases} A_1 = -bk^2 - k, \\ A_2 = ek^3 - 1, \end{cases} \quad \begin{cases} B_1 = bk^2 + 2k, \\ B_2 = 3, \end{cases} \quad \begin{cases} W_1 = k\,u_2, \\ W_2 = 3\,u_2 - u_3. \end{cases}$$

Funkcje u_0, u_1 należy więc wyznaczyć z układu równań

$$\begin{cases} -(1 + bk)\,u_0 + (bk + 2)\,u_1 = u_2, \\ (ek^3 - 1)\,u_0 + 3\,u_1 = 3\,u_2 - u_3. \end{cases}$$

Na prawym końcu / $\mu = 3,4$/:

$$\alpha_{03} = 1 = \alpha_{24}, \qquad \beta_3 = \beta_3(t),$$

więc

$$\begin{cases} A_3 = k^3, \\ A_4 = k, \end{cases} \quad \begin{cases} B_3 = 0, \\ B_4 = -2k, \end{cases} \quad \begin{cases} W_3 = k^3 \beta_3\, t, \\ W_4 = -u_{n-2}\, k, \end{cases}$$

zatem z /13/:

$$\begin{cases} u_n = \beta_3(t), \\ u_{n-1} = \frac{1}{2}\left[\beta_3(t) + u_{n-2}\right], \quad \dot{u}_{n-1} = \frac{1}{2}\left[\dot{\beta}_3(t) + \dot{u}_{n-2}\right]. \end{cases}$$

Zajmijmy się obecnie układem /10/. Wprowadzając nowe funkcje $u_{n+i} = \dot{u}_i$, możemy go sprowadzić do układu $2(n-3)$ równań pierwszego rzędu:

$$\begin{cases} \dot{u}_i = u_{n+i} \\ \dot{u}_{n+i} = F_i\left(t, x_i, u_0, \ldots, u_n, u_{n+1}, \ldots, u_{2n-1}\right), \quad i = 2,3,\ldots,n-2, \end{cases} \tag{14}$$

gdzie

$$F_i = -\frac{a_i(t)}{k^4}\left(u_{i+2} - 4u_{i+1} + 6u_i - 4u_{i-1} + u_{i-2}\right) + f\left(t, x_i, u_i, u_{n+i}, p_i, q_i, r_i, s_i\right) \tag{15}$$

oraz

$$\begin{cases} p_i = \frac{1}{2k}\left(u_{i+1} - u_{i-1}\right) \\ q_i = \frac{1}{2k}\left(u_{n+i+1} - u_{n+i-1}\right) \\ r_i = \frac{1}{k^2}\left(u_{i+1} - 2u_i + u_{i-1}\right) \\ s_i = \frac{1}{2k^3}\left(u_{i+2} - 2u_{i+1} + 2u_{i-1} - u_{i-2}\right). \end{cases} \tag{16}$$

Warunki początkowe układu /14/, wynikające z /11/ będą:

$$\begin{cases} u_i(0) = g_1(x_i) \\ u_{n+i}(0) = g_2(x_i), \qquad i = 0,1, \ldots, n. \end{cases} \tag{17}$$

Do całkowania układu /14/ można stosować różne ze znanych metod. Zaleca się tu jednak metodę Runge-Kutty-Gilla [1]. Z jednej strony jest ona bardziej stabilna od metod różnicowych tego samego rzę-

du /zwłaszcza dla układu o dużej ilości równań z jakim mamy tu do czynienia/, z drugiej zaś wymaga zarezerwowania mniejszej ilości miejsc roboczych, niż metoda Runge-Kutty [2], czy Mersona [3]. Nie bez znaczenia jest również fakt, że zastosowanie jej prowadzi do najkrótszego programu. Po drobnych modyfikacjach i dostosowaniu jej do układu specjalnego typu jakim jest układ /14/, algorytm tej metody ma postać:

$$\left\{\begin{array}{l} \left.\begin{array}{l} k_{ij} = h\, u_{n+i,j} \\ k_{n+i,j} = hF_i\left(t_j, x_i, u_{oj}, \ldots, u_{nj}, u_{n+1,j}, \ldots, u_{2n-1,j}\right) \end{array}\right\} i=2,3,\ldots,n-2 \\ t_{j+1} = t_j + V_j \\ R_{i,j+1} = T_j\left(k_{ij} - Q_{ij}\right) - S_j\, Q_{ij} \\ u_{i,j+1} = u_{ij} + R_{i,j+1} \qquad i=2,3,\ldots,n-2,n+2,\ldots,2n-2 \\ Q_{i,j+1} = Q_{ij} + 3R_{i,j+1} - U_j\, k_{ij} \\ u_{oj},\ u_{1j},\ u_{n-1,j},\ u_{nj},\ u_{n+1,j},\ u_{2n-1,j} - \text{z /12/, /13/} \end{array}\right. \qquad /18/$$

$$j = 0,1,2,3;$$

przy czym

$$\left\{\begin{array}{l} t_o = t \\ u_{io} = u_i(t) \\ u_{n+i,o} = u_{n+i}(t) = \dot{u}_i(t), \end{array}\right. \qquad \left\{\begin{array}{l} t_4 = t + h \\ u_{i4} = u_i(t+h) \\ u_{n+i,4} = u_{n+i}(t+h) = \dot{u}_i(t+h). \end{array}\right. \qquad /19/$$

Jest to więc metoda jednokrokowa, pozwalająca - poprzez wielokrotne odwoływanie się do niej - obliczać wartości funkcji u_i, $\dot{u}_i$ w

kolejnych punktach t, t+h, t+2h, itd. Przed pierwszym odwołaniem się do niej kładziemy $Q_{io} = 0$, zaś u_{io}, $u_{n+i,o}$ obliczamy z warunków początkowych /17/. W każdym następnym kroku przyjmujemy za Q_{io}, u_{io}, $u_{n+i,o}$ wartości Q_{i4}, u_{i4}, $u_{n+i,4}$ policzone w kroku poprzednim. Stałe występujące we wzorach /18/ podajemy w tabeli 1.

Tabela 1.

j	T_j	S_j	U_j	V_j
0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{h}{2}$
1	$1 - \sqrt{\frac{1}{2}}$	0	$1 - \sqrt{\frac{1}{2}}$	0
2	$1 + \sqrt{\frac{1}{2}}$	0	$1 + \sqrt{\frac{1}{2}}$	$\frac{h}{2}$
3	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{2}$	0

3. PROGRAM OBLICZEŃ NA MASZYNĘ CYFROWĄ

Podamy obecnie organizację programu w postaci zamkniętego bloku realizującego formuły /18/. Może to być zarówno podprogram w języku wewnętrznym, jak i algolowska procedura, czy też blok wywoływany w autokodzie WAT-1*) instrukcją # Wykonaj:. Dla ustalenia uwagi będziemy dalej interesujący nas program nazywali procedurą. Będzie ona korzystała z nazw zewnętrznych, w których będą umieszczone wartości dane /wejściowe/ i wyniki, oraz z nazw własnych, które będą nazwami roboczymi procedury. Przez nazwę /miejsce/ rozumiemy tu adres pamięci wewnętrznej /dla programu napisanego w kodzie wewnętrznym/, ewentualnie zadeklarowaną nazwę zmiennej /dla programu napisanego w autokodzie/.

Wypiszemy nazwy, których będziemy używali w procedurze /w nawiasach podano zakres zmienności indeksów dla zmiennych indeksowanych/.

*)WAT-1 jest autokodem opartym na formalizmie Łukasiewicza, opracowanym w Wojskowej Akademii Technicznej dla maszyny URAŁ-2.

- Nazwy zewnętrzne:
- zmienne rzeczywiste: t, x, p, q, r, s, h, k, l, a, u(0:2n-1), f(2:2n-2), α(0:3)(1:4), β(1:4), $\dot{\beta}$(1:4);
- zmienne całkowite: i, j, n, par;
- Nazwy wewnętrzne:
- zmienne rzeczywiste: R, D_1, D_p, E(1:4), A(1:4), B(1:4), W(1:4), $\dot{W}$(1:4), Q(2:2n-2), T(0:3), S(0:3), U(0:3), V(0:3);
- zmienne całkowite: s(1:4), c(1:4);

Znaczenie poszczególnych nazw będzie objaśnione w schemacie blokowym.

Przed pierwszym odwołaniem się do procedury należy umieścić dane wejściowe w następujących miejscach:

- parametry liczbowe α_{ij} w α_{ij}, i = 0,1,2,3; j = 1,2,3,4.
- długość belki w l
- krok całkowania w kierunku osi t w h
- ilość punktów podziału przedziału [0,l] w n
- liczbę całkowitą 1 w par.

Procedura musi korzystać poza tym z bloków /procedur/ realizujących następujące formuły:

Nazwa bloku	Formuła, którą realizujemy	Miejsce umieszczenia wyniku
[f]	$f(t,x,u_i,u_{n+i},p,q,r,s)$	f_i
[a]	$a(t, x)$	a
[g]	$g_1(x)$, $g_2(x)$	u_i, u_{n+i}
[β]	$\beta_i(t)$, $\dot{\beta}_i(t)$, i=1,2,3,4	β_i, $\dot{\beta}_i$, i=1,2,3,4

Bloki te należy każdorazowo /tzn. dla każdego zadania określonego przez (1), (2), (3)/ zaprogramować w programie głównym.

Po każdym wyjściu z procedury, w miejscach t, u(0:n), u(n+1:2n-1) znajdują się odpowiednio: aktualna wartość zmiennej niezależnej t, wartości funkcji $u_i(t)$, $i = 0,1, \ldots, n$, oraz ich pochodne $\dot{u}_i(t)$, $i = 1,2, \ldots, n-1$.

Schemat blokowy wykorzystuje następujące wzory robocze:

$$\begin{cases} s_1 = 2 = s_2 \\ s_3 = n-2 = s_4, \end{cases} \qquad \begin{cases} o_1 = 3 = o_2 \\ o_3 = n-3 = o_4, \end{cases} \qquad /20/$$

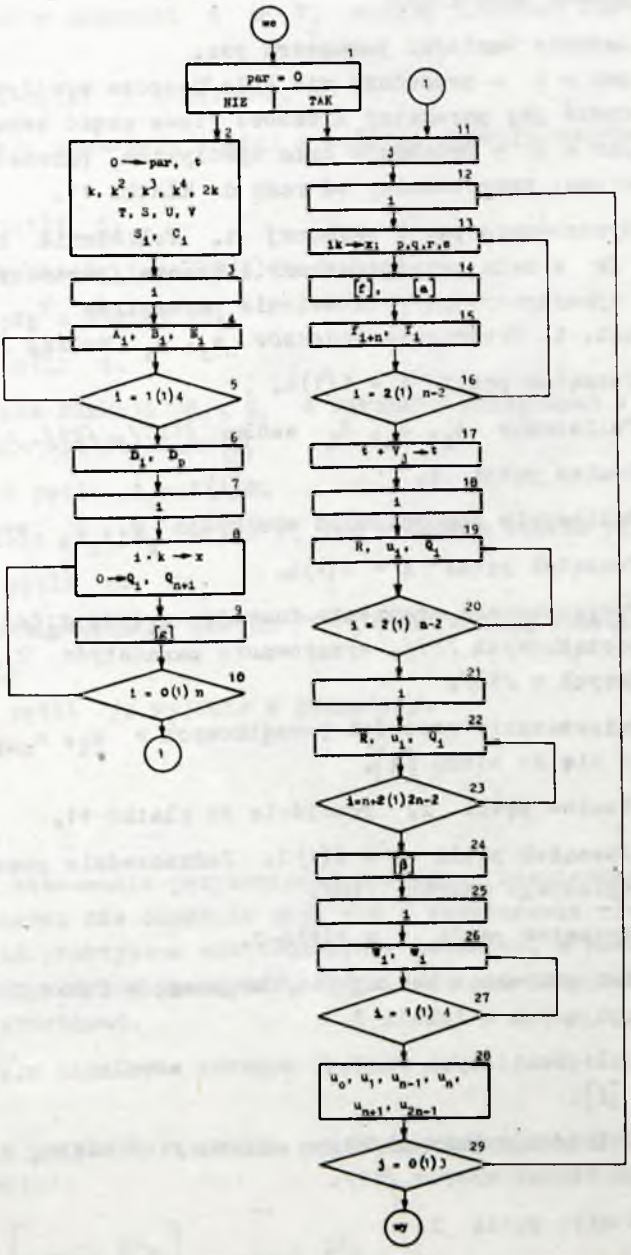
$$E_i = \alpha_{2i}\, k-3\, \alpha_{3i}, \qquad i = 1,2,3,4, \qquad /21/$$

$$\begin{cases} D_1 = A_1 B_2 - B_1 A_2 \\ D_p = A_3 B_4 - B_3 A_4, \end{cases} \qquad /22/$$

$$\begin{cases} W_i = k^3 \beta_i - u_{s_i} E_i - u_{o_i} \alpha_{3i} \\ \dot{W}_i = k^3 \dot{\beta}_i - u_{n+s_i} E_i - u_{n+o_i} \alpha_{3i}, \end{cases} \qquad i = 1,2,3,4, \qquad /23/$$

$$\begin{cases} u_o = \frac{1}{D_1}(W_1 B_2 - B_1 W_2) \\ u_1 = \frac{1}{D_1}(A_1 W_2 - W_1 A_2) \\ u_n = \frac{1}{D_p}(W_3 B_4 - B_3 W_4) \\ u_{n-1} = \frac{1}{D_p}(A_3 W_4 - W_3 A_4) \\ u_{n+1} = \frac{1}{D_1}(A_1 \dot{W}_2 - \dot{W}_1 A_2) \\ u_{2n-1} = \frac{1}{D_p}(A_3 \dot{W}_4 - \dot{W}_3 A_4), \end{cases} \qquad /24/$$

4. Schemat blokowy procedury



Opis schematu blokowego:

Klatka 1: Badanie wartości parametru par.
Jeśli par = 1 - procedura nie była jeszcze wywoływana i należy ustawić jej parametry liczbowe /lewa część schematu/.
Jeśli par = 0 - procedura była wywoływana, parametry zostały ustawione, przechodzimy od razu do klatki 11.

Klatka 2: Wyzerowanie par i zmiennej t. Policzenie $k = \frac{1}{n}$, k^2, k^3, k^4, 2k w celu przyśpieszenia liczenia /parametry wielokrotnie wykorzystywane/. Ustawienie parametrów T_j, S_j, U_j, V_j według tab. 1. Ustawienie indeksów s_i, o_i według /20/.

Klatka 3: Początek pętli i = 1(1)4.

Klatka 4: Policzenie A_i, B_i, E_i według /12'/, /21/.

Klatka 5: Koniec pętli i.

Klatka 6: Policzenie pomocniczych wielkości D_1, D_p według /22/.

Klatka 7: Początek pętli i = 0(1)n.

Klatka 8: Przygotowanie argumentu funkcji $g_1(x)$, $g_2(x)$ z warunków początkowych /17/, wyzerowanie parametrów Q_i, Q_{n+i}, występujących w /18/.

Klatka 9: Umieszczenie wartości początkowych w u_i, u_{n+i}, poprzez odwołanie się do bloku [g].

Klatka 10: Koniec pętli i; przejście do klatki 11.

Klatka 11: Początek pętli j = 0(1)3. Jednocześnie początek algorytmu opisanego wzorami /18/.

Klatka 12: Początek pętli i = 2(1)n-2.

Klatka 13: Przygotowanie według /16/ argumentów funkcji a oraz f, występujących w tabeli 2.

Klatka 14: Policzenie tych funkcji poprzez odwołanie się do bloków [a], [f].

Klatka 15: Policzenie prawych stron układu /14/ według dwóch pierwszych formuł wzorów /18/.

Klatka 16: Koniec pętli i.

Klatka 17: Zwiększenie argument t o V_j według trzeciej formuły wzorów /18/.

Klatka 18: Początek pętli i = 2(1)n-2.

Klatka 19: Realizacja czwartej, piątej i szóstej formuły wzorów /18/.

Klatka 20: Koniec pętli i.

Klatka 21: Początek pętli i = n+2(1)2n-2.

Klatka 22: Identyczna z klatką 19.

Klatka 23: Koniec pętli i.

Klatka 24: Policzenie funkcji β_i, $\dot{\beta}_i$ z warunków brzegowych /3/ poprzez odwołanie się do bloku [β]

Klatka 25: Początek pętli i = 1(1)4.

Klatka 26: Policzenie W_i, $\dot{W}_i$ z /12'/, /13'/ według wzorów /23/.

Klatka 27: Koniec pętli i.

Klatka 28: Rozwiązanie układów równań /12/, /13/ według wzorów /24/.

Klatka 29: Koniec pętli j; wyjście z procedury.

5. PRZYKŁADY

Jako przykłady stosowania przytoczonej procedury rozwiązano dwa równania. Autorowi nie chodziło przy tym o rozwiązanie równań mających jakieś praktyczne zastosowania w technice, a raczej o wskazanie na różnorodność problemów, które dają się przy pomocy tej procedury rozwiązać.

Przykład 1

Drgania belki o zmiennym przekroju wywołane ruchem jej końca. Równanie ma tu postać:

$$\frac{\partial^2 u}{\partial x^2}\left[E\,I(x)\,\frac{\partial^2 u}{\partial x^2}\right] + \rho A(x)\,\frac{\partial^2 u}{\partial t^2} = 0. \qquad /25/$$

Przyjmijmy warunki początkowe /2/ w postaci:

$$\begin{cases} g_1(x) = -\,o\left(\frac{x}{l}\right)^4 \\ g_2(x) = 0, \end{cases} \qquad /26/$$

oraz brzegowe - /4/, tzn.

$$\begin{cases} \alpha_{11} = b, \quad \alpha_{21} = -1, \quad \alpha_{02} = e, \quad \alpha_{32} = 1, \\ \alpha_{03} = 1 = \alpha_{24}, \quad \beta_3(t) = -\frac{o}{2}\left(1 + \cos\omega t\right). \end{cases} \qquad /27/$$

Niech przekrój poprzeczny belki będzie prostokątem o stałej szerokości $\bar{b}$, oraz parabolicznie zmiennej wysokości:

$$h = H\sqrt{1 - \frac{x}{l}},$$

gdzie H jest wysokością belki na lewym końcu. Przekrój i moment bezwładności będą:

$$\begin{cases} A(x) = \bar{b}h = \bar{b}H\sqrt{1 - \frac{x}{l}} \\ I(x) = \frac{\bar{b}h^3}{12} = \frac{H^2}{12}A(x)\left(1 - \frac{x}{l}\right). \end{cases}$$

Po podstawieniu tych funkcji do /25/ i doprowadzeniu równania do postaci /1/ otrzymamy

$$\begin{cases} a = c_1(1 - x) \\ f = c_2\left(4s - \frac{r}{1-x}\right), \end{cases} \qquad /28/$$

gdzie

$$\begin{cases} c_1 = \dfrac{EH^2}{12\,\rho l} \\ c_2 = \dfrac{3}{4}\,c_1 \\ s,\ r - \text{określone przez /8/.} \end{cases}$$

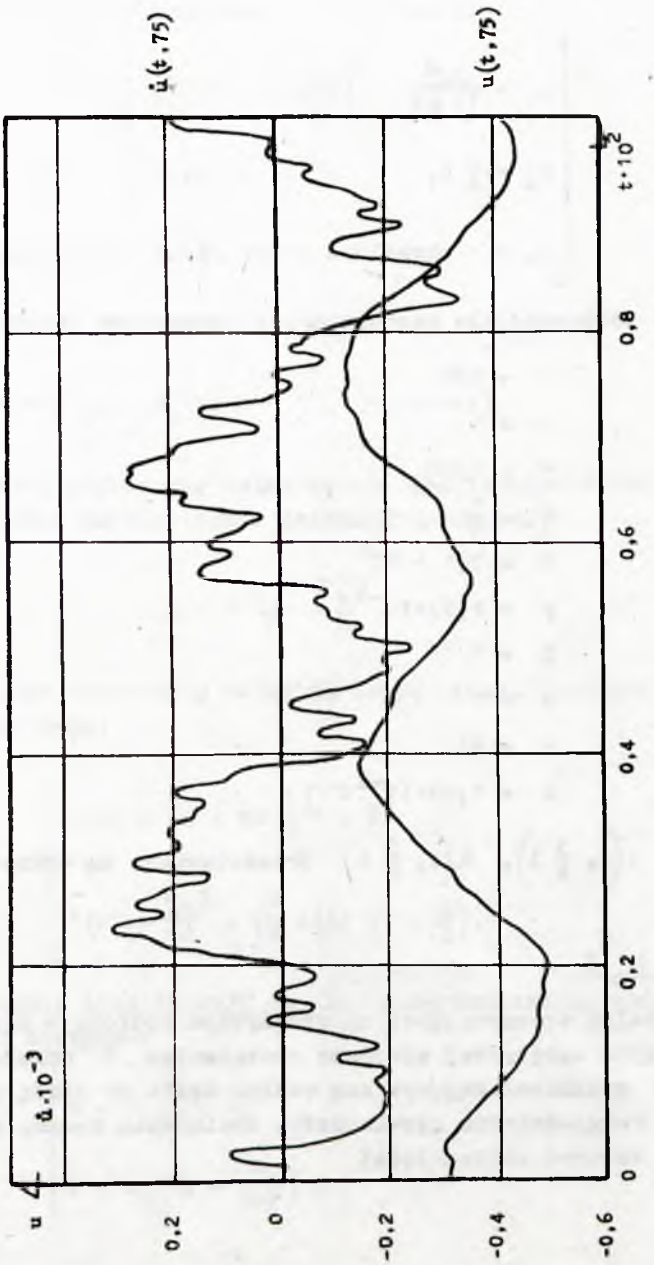
Przykład policzono dla następujących parametrów liczbowych

$$\begin{aligned} l &= 100 \\ o &= 1 \\ \omega &= 1000 \\ \frac{H^2}{l} &= \frac{1}{4} \\ E &= 2{,}1 \cdot 10^{6} \\ \rho &= 1{,}75 \cdot 10^{-5} \\ b &= 1 \\ e &= 1 \\ n &= 24 \\ h &= 1{,}25 \cdot 10^{-5}. \end{aligned}$$

Funkcje $u\left(t, \frac{3}{4}l\right)$, $\dot{u}\left(t, \frac{3}{4}l\right)$ przedstawione są wykresem na rys. 3.

P r z y k ł a d 2

Drgania belki spoczywającej na sprężystym podłożu o nieliniowej charakterystyce sprężystej wywołane oboiążeniem P działającym na odcinku 2λ przemieszczającym się wzdłuż belki ze stałą prędkością v. Po uwzględnieniu bezwładności oboiążenia możemy równanie drgań belki zapisać następująco:



Rys. 3.

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} + a\dfrac{\partial^4 u}{\partial x^4} + c\,u + b\,u^3 = 0 & \text{dla } |x-vt|>\lambda \\ \dfrac{\partial^2 u}{\partial t^2} + a\dfrac{\partial^4 u}{\partial x^4} + cu + bu^3 = \Delta \cdot \left(v\dfrac{\partial}{\partial x} + \dfrac{\partial}{\partial t}\right)^2 u + P & \text{dla } |x-vt|\leq\lambda . \end{cases} \qquad /29/$$

Sprowadzając je do postaci /1/, otrzymamy

$$\frac{\partial^2 u}{\partial t^2} + \bar{a}\frac{\partial^4 u}{\partial x^4} = f, \qquad /30/$$

gdzie

$$f = \begin{cases} -c\,u - b\,u^3 & \text{dla } |x - vt|>\lambda \\ \dfrac{1}{1-\Delta}\left[v\,\Delta\left(vr + 2q\right) + P - cu - bu^3\right] & \text{dla } |x-vt|\leq\lambda \end{cases}$$

$$\bar{a} = \begin{cases} a & \text{dla } |x - vt|>\lambda \\ \dfrac{a}{1-\Delta} & \text{dla } |x - vt|\leq\lambda \end{cases}$$

q, r - określone w /8/.
Przyjmijmy tu warunki początkowe

$$g_1(x) = 0 = g_2(x),$$

oraz brzegowe określone przez /6/, tzn.

$$\alpha_{01} = 1 = \alpha_{22}$$

$$\alpha_{03} = 1 = \alpha_{24}.$$

Do obliczeń przyjęto następujące parametry liczbowe

$$a = 1{,}75 \cdot 10^5$$

$$c = 70$$

$$b = -35$$

$$P = 41{,}65$$

$$\Delta = -4{,}2$$

$$l = 100$$

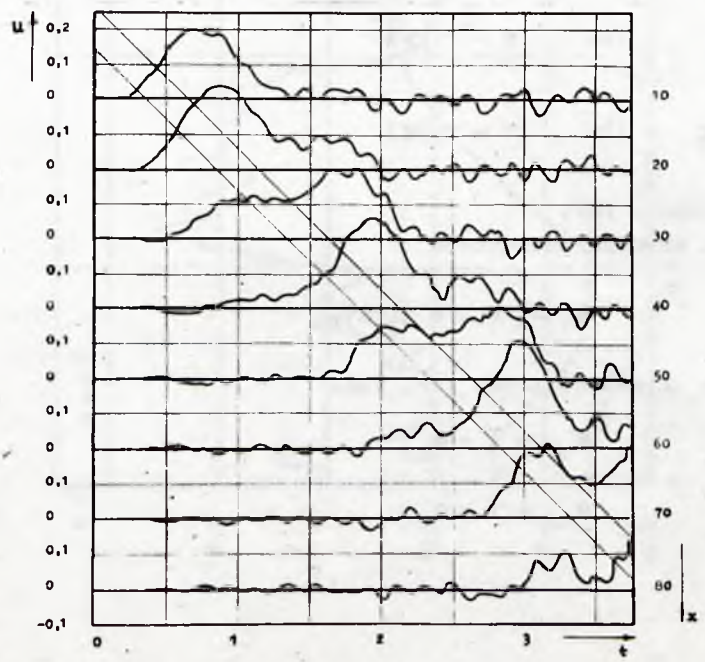
$$v = 20$$

$$\lambda = 3$$

$$n = 30$$

$$h = 0{,}005.$$

Wykresy funkcji $u(t, x_i)$, $i = 3,6,9, \ldots, 24$ podano na rys. 4. Odcinki wycięte na osiach $u = 0$ /które są równocześnie osiami $x = x_i$/ przez ukośny pas ilustrują okresy czasu, w których nad i-tym punktem belki znajduje się obciążenie P.



Rys. 4.

6. UWAGI KOŃCOWE

Opisana procedura wydaje się mieć największe zastosowania do rozwiązywania skomplikowanych równań liniowych, tzn. gdy w równaniu /1/ f jest funkcją liniową swoich argumentów, a wszystkie współczynniki /lub niektóre z nich/ są złożonymi funkcjami argumentów t, x. Rozwiązanie takiego równania metodami przybliżonymi może być, choćby ze względu na trudności rachunkowe, często niemożliwe. Nie wykluoza się oczywiście możliwości stosowania procedury w przypadkach gdy f jest funkcją nieliniową, leoz nie znając skądinąd przybliżonego przebiegu rozwiązania, lub jego pewnych cech /jak np. częstość drgań, czy maksymalna amplituda/ trudno jest zweryfikować otrzymane rozwiązanie.

Odnośnie kroków całkowania $\frac{1}{n}$ oraz h, to w przypadkach, gdy nie można ich ustalić na podstawie przewidywanego rozwiązania, należy je dobrać metodą prób. Brak odpowiednich opracowań teoretycznych uniemożliwia bardziej ekonomiczny tok postępowania.

Literatura

[1] GILL S.: A process for the step - by step integration of diff.equations. Proc. of the Cambridge Phil. Soc., 47, 1951, 96-108.

[2] COLLATZ L.: Metody numeryczne rozwiązywania równań różniczkowych. PWN 1960.

[3] LANCE G.N.: Numerical methods for high speed computers. London 1960.

[4] LIEBIEDIEV W.I.: Urawnienija i schodimost diff.razn. mietoda.

[5] BUDAK B.M.: O mietodie priamych dla niekotorych kraiewych zadać. Wiestnik MGU, Nr 1, 1956, 3-11.

Summary

This paper is concerned with organisation of the procedure for numerical integrating of partial differential equation:

$$\frac{\partial^2 u}{\partial t^2} + a(t,x)\frac{\partial^4 u}{\partial x^4} = f\left(t,x,u,\frac{\partial u}{\partial x},\frac{\partial u}{\partial t},\frac{\partial^2 u}{\partial x \partial t},\frac{\partial^2 u}{\partial x^2},\frac{\partial^3 u}{\partial x^3}\right),$$

with initial conditions

$$\begin{cases} u(0,x) = g_1(x) \\ \dfrac{\partial u}{\partial t}(0,x) = g_2(x), \end{cases}$$

and boundary values

$$\begin{cases} \displaystyle\sum_{\mu=0}^{3} \alpha_{6\mu} \frac{\partial^6 u}{\partial x^6}(t,0) = \beta_\mu(t), & \mu = 1,2, \\ \displaystyle\sum_{\mu=0}^{3} \alpha_{6\mu} \frac{\partial^6 u}{\partial x^6}(t,1) = \beta_\mu(t), & \mu = 3,4, \end{cases}$$

on the high - speed computers.

This procedure is applicable to the problem of mechanical vibrations of beams. Two examples are given.

Instytut Maszyn Matematycznych
Algorytmy N° 7

517.946:518.61:551.509

O PEWNYM WIELOPOZIOMOWYM MODELU KRÓTKOTERMINOWEJ PROGNOZY POGODY

Krzysztof MOSZYŃSKI

Pracę złożono 1.7.1966 r.

Praca zawiera opis aproksymacji równań pierwotnych prognozy pogody /1/ - /5/ przez inny układ równań różniczkowych /4/.Układ ten nie zawiera już zmiennej niezależnej ζ i wydaje się wygodniejszy dla dalszych badań. Zastosowano tu podobnie jak w [2] metodę Dorodnicyna [3], jednak sposób otrzymania aproksymacji znacznie odbiega od sposobu zastosowanego w [2].

WSTĘP

Procesy zachodzące w atmosferze mogą być opisane przez następujący układ równań różniczkowych:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \omega\frac{\partial u}{\partial \zeta} + \frac{\partial \Phi}{\partial x} - fv = 0 \qquad /1/$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \omega\frac{\partial v}{\partial \zeta} + \frac{\partial \Phi}{\partial y} + fu = 0 \qquad /2/$$

$$\frac{\partial \vartheta}{\partial t} + u\frac{\partial \vartheta}{\partial x} + v\frac{\partial \vartheta}{\partial y} + \omega\frac{\partial \vartheta}{\partial \zeta} = 0 \qquad /3/$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial \omega}{\partial \zeta} = 0 \qquad /4/$$

$$\frac{\partial \Phi}{\partial \zeta} = -R\vartheta\zeta^{\varkappa-1} \qquad /5/$$

Układ /1/ - /5/ stanowi tzw. układ równań pogody w lokalnych współrzędnych ciśnieniowych /por. np. [1]/.

Znaczenie symboli występujących w równaniach jest następujące:

t - czas; $t \geq 0$
x, y - płaskie współrzędne kartezjańskie
ζ - tzw. zredukowane ciśnienie /$0 \leq \zeta \leq 1$/

} Zmienne niezależne

$u(t,x,y,\zeta)$
$v(t,x,y,\zeta)$
$\omega(t,x,y,\zeta)$ - składowe prędkości wiatru

$\vartheta(t,x,y,\zeta)$ - temperatura potencjalna
$\Phi(t,x,y,\zeta)$ - geopotencjał

} Funkcje niewiadome

$f(x,y)$ - współczynnik związany z występowaniem siły Coriolisa
R - stała gazowa
$\varkappa$ - stała; $0 < \varkappa < 1$ $/\varkappa = \frac{R}{C_v + R}$ gdzie C_v - ciepło właściwe powietrza przy stałej objętości/.

Dla pięciu funkcji $u, v, \omega, \vartheta, \Phi$, które należy wyznaczyć z równań /1/ - /5/ stawia się pewne warunki początkowe i brzegowe. Warunki te z reguły można podzielić na trzy grupy:

- warunki brzegowe ze względu na zmienną niezależną ζ ;
- warunki początkowe stawiane dla niektórych funkcji niewiadomych w pewnym obszarze przestrzeni x, y, ζ , dla $t = 0$;
- warunki brzegowe dla tych funkcji, stawiane na brzegu rozważanego obszaru przestrzeni zmiennych x, y, ζ , dla wszystkich wartości t.

Chwilowo sprecyzujemy jedynie warunki brzegowe pierwszej grupy. Dla $\zeta = 0$ będziemy przyjmować

$$\omega = 0 \, , \qquad /6/$$

zaś dla $\zeta = 1$ będziemy żądali spełnienia następującego warunku:

$$\omega = \frac{1}{R\vartheta}\left[\frac{\partial \Phi}{\partial t} + u\frac{\partial \Phi}{\partial x} + v\frac{\partial \Phi}{\partial y} - w_0 g\right], \qquad /7/$$

gdzie

g - przyspieszenie ziemskie

$w_0(t,x,y)$ - znana funkcja. Funkcja ta określa składową prędkości wiatru w kierunku pionowym, na powierzchni izobarycznej w stosunku do której redukuje się ciśnienie /por. [1], [2]/. Przeważnie przyjmuje się $w_0 \equiv 0$.

Celem tej pracy jest wyeliminowanie z rozważań zmiennej niezależnej ζ, występującej w sposób wyraźnie różny, niż pozostałe zmienne t, x, y. Ściślej mówiąc, zmienna ta została w pewien sposób zastąpiona dyskretnym układem punktów /poziomów/ oraz układ równań /1/ - /5/ został sprowadzony w sposób przybliżony do innego układu równań /41/, wprawdzie większego, ale zależnego jedynie od zmiennych t, x, y i posiadającego budowę o wiele bardziej jednolitą, co wydaje się być ważne przy dalszych badaniach otrzymanego układu.

Zastosowano tu, podobnie jak w pracy [2] metodę Dorodnicyna [3], jednak sposób otrzymania aproksymacji układu odbiega dość znacznie od sposobu zastosowanego w [2]. W konsekwencji otrzymano, przy jednakowej ilości poziomów, nieco mniej równań.

Ponadto, w przeciwieństwie do [2], punktem wyjścia do opisanej tu aproksymacji jest układ równań /1/ - /5/, bez żadnych dodatkowych uproszczeń.

ELIMINACJA FUNKCJI ω I Φ

Zauważmy, że korzystając z równania /4/ oraz warunku /6/, można napisać:

$$\omega(t,x,y,\zeta) = -\int_0^{\zeta}\left[\frac{\partial}{\partial x}u(t,x,y,\tau) + \frac{\partial}{\partial y}v(t,x,y,\tau)\right]d\tau . \qquad /8/$$

Podobnie z równania /5/ otrzymujemy:

$$\Phi(t,x,y,\zeta) = \Phi(t,x,y,1) + R\int_{\zeta}^{1} \vartheta(t,x,y,\tau)\tau^{\varkappa-1}\,d\tau \ .$$

Oznaczmy $\Psi(t,x,y) = \Phi(t,x,y,1)$, zatem:

$$\Phi(t,x,y,\zeta) = \Psi(t,x,y) + R\int_{\zeta}^{1} \vartheta(t,x,y,\tau)\tau^{\varkappa-1}\,d\tau \ . \qquad /9/$$

Dla określenia funkcji Ψ można wykorzystać warunek /7/. Oznaczając dla prostoty

$$D(t,x,y,\zeta) = \frac{\partial}{\partial x}u(t,x,y,\zeta) + \frac{\partial}{\partial y}v(t,x,y\zeta)$$

otrzymujemy dodatkowe równanie różniczkowe

$$\frac{\partial\Psi}{\partial t} + u_k\frac{\partial\Psi}{\partial x} + v_k\frac{\partial\Psi}{\partial y} + R\vartheta_k\int_{0}^{1} D\,d\tau - w_o g = 0 \ , \qquad /10/$$

gdzie

$$u_k = u(t,x,y,1); \qquad v_k = v(t,x,y,1); \qquad \vartheta_k = \vartheta(t,x,y,1) \ .^{*)}$$

Ponadto, z zależności /9/ otrzymujemy wyrażenia dla pochodnych $\frac{\partial\Phi}{\partial x}$ i $\frac{\partial\Phi}{\partial y}$ /zakładając dostateczną regularność funkcji ϑ/, oraz po wstawieniu ich do równań /1/, /2/, /3/, /10/ dochodzimy do następującego układu czterech równań różniczkowo-całkowych dla czterech funkcji niewiadomych $u(t,x,y,\zeta)$, $v(t,x,y,\zeta)$, $\vartheta(t,x,y,\zeta)$, $\Psi(t,x,y)$:

---

*) Zauważmy, że funkcja Ψ przedstawia geopotencjał powierzchni izobarycznej $p_o$mb, gdzie $\zeta = p/p_o$; p_o = const. Zakłada się przy tym, że składowa pionowa prędkości wiatru na tej powierzchni w_o jest znana. Wprowadza to w pewnym sensie do rozważań wpływ ukształtowania powierzchni ziemi.

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}-\int_0^{\zeta} D\,d\tau\cdot\frac{\partial u}{\partial \zeta}+R\int_{\zeta}^{1}\frac{\partial \vartheta}{\partial x}\tau^{\varkappa-1}\,d\tau+\frac{\partial \psi}{\partial x}-fv=0 \qquad /11/$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}-\int_0^{\zeta} D\,d\tau\,\frac{\partial v}{\partial \zeta}+R\int_{\zeta}^{1}\frac{\partial \vartheta}{\partial y}\tau^{\varkappa-1}\,d\tau+\frac{\partial \psi}{\partial y}+fu=0 \qquad /12/$$

$$\frac{\partial \vartheta}{\partial t}+u\frac{\partial \vartheta}{\partial x}+v\frac{\partial \vartheta}{\partial y}-\int_0^{\zeta} D\,d\tau\cdot\frac{\partial \vartheta}{\partial \zeta}=0 \qquad /13/$$

$$\frac{\partial \psi}{\partial t}+u_K\frac{\partial \psi}{\partial x}+v_K\frac{\partial \psi}{\partial y}+R\vartheta_K\int_0^{1} D\,d\tau-w_0 g=0\,. \qquad /14/$$

Przekształćmy jeszcze równania /11/ – /14/. Niech $f(z)$ i $g(z)$ będą dowolnymi, dostatecznie regularnymi funkcjami zmiennej rzeczywistej z. Mamy wtedy:

$$\frac{d}{dz}\left[\int_0^{z} f(\tau)\,d\tau\cdot g(z)\right]=f(z)g(z)+\left[\int_0^{z} f(\tau)\,d\tau\right]\frac{dg(z)}{dz}\,. \qquad /15/$$

Zauważmy jeszcze, że jeśli $\varphi(z)$ jest dowolną, dostatecznie regularną funkcją taką, że wyrażenie /16/ ma sens oraz α liczbą rzeczywistą, to funkcja:

$$G(z)=\int_0^{z}\varphi(\tau)\tau^{\alpha+1}\,d\tau+z\int_z^{1}\varphi(\tau)\tau^{\alpha}\,d\tau \qquad /16/$$

spełnia warunki:

$$\frac{dG(z)}{dz}=\int_z^{1}\varphi(\tau)\tau^{\alpha}\,d\tau \qquad /17/$$

$$G(0) = 0$$

Stosując związki /15/, /16/, /17/ do równań /11/, /12/, /13/ otrzymujemy:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + u\cdot D + \frac{\partial \Psi}{\partial x} - fv = \frac{\partial}{\partial \zeta}\left[u\int_0^\zeta D\,d\tau - R\left(\int_0^\zeta \frac{\partial \vartheta}{\partial x}\tau^{\varkappa} d\tau + \zeta \int_\zeta^1 \frac{\partial \vartheta}{\partial x}\tau^{\varkappa-1} d\tau \right)\right] \quad /18/$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + v\cdot D + \frac{\partial \Psi}{\partial y} + fu = \frac{\partial}{\partial \zeta}\left[v\cdot\int_0^\zeta D\,d\tau - R\left(\int_0^\zeta \frac{\partial \vartheta}{\partial y}\tau^{\varkappa} d\tau + \zeta \int_\zeta^1 \frac{\partial \vartheta}{\partial y}\tau^{\varkappa-1} d\tau \right)\right] \quad /19/$$

$$\frac{\partial \vartheta}{\partial t} + u\frac{\partial \vartheta}{\partial x} + v\frac{\partial \vartheta}{\partial y} + \vartheta\cdot D = \frac{\partial}{\partial \zeta}\left[\vartheta\int_0^\zeta D\,d\tau \right] \quad /20/$$

$$\frac{\partial \Psi}{\partial t} + u_K\frac{\partial \Psi}{\partial x} + v_K\frac{\partial \Phi}{\partial y} + R\vartheta_K\int_0^1 D\,d\tau - W_{0g} = 0 \quad /21/$$

Dla $\zeta = 0$ wygodniej będzie dalej korzystać z równań /11/, /12/, /13/. Wprowadzając oznaczenia $u_o = u(t,x,y,0)$; $v_o = v(t,x,y,0)$; $\vartheta_o = \vartheta(t,x,y,0)$, otrzymujemy dodatkowo inne nieco związki dla tego poziomu:

$$\frac{\partial u_o}{\partial t} + u_o\frac{\partial u_o}{\partial x} + v_o\frac{\partial u_o}{\partial y} + \frac{\partial \Psi}{\partial x} - fv_o = -R\int_0^1 \frac{\partial \vartheta}{\partial x}\tau^{\varkappa-1} d\tau \quad /22/$$

$$\frac{\partial v_o}{\partial t} + u_o\frac{\partial v_o}{\partial x} + v_o\frac{\partial v_o}{\partial y} + \frac{\partial \Psi}{\partial y} + fu_o = -R\int_0^1 \frac{\partial \vartheta}{\partial y}\tau^{\varkappa-1} d\tau \quad /23/$$

$$\frac{\partial \vartheta_o}{\partial t} + u_o\frac{\partial \vartheta_o}{\partial x} + v_o\frac{\partial \vartheta_o}{\partial y} = 0 \quad /24/$$

Zauważmy, że równania te mogły być otrzymane jedynie przy założeniu, że:

$$\lim_{\zeta \to +0} \int_0^{\zeta} D d\tau \frac{\partial u}{\partial \zeta} = 0,$$

$$\lim_{\zeta \to +0} \int_0^{\zeta} D d\tau \frac{\partial v}{\partial \zeta} = 0,$$

$$\lim_{\zeta \to +0} \int_0^{\zeta} D d\tau \frac{\partial \vartheta}{\partial \zeta} = 0$$

oraz, że całki występujące po obu stronach równań /22/, /23/ mają sens; $(0 < \varkappa < 1)$.

Scałkujmy teraz stronami równania /18/, /19/, /20/ w przedziale $[0, \zeta]$. Wprowadzając oznaczenia:

$$g_1 = \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} - fv + \frac{\partial \Psi}{\partial x}$$

$$g_2 = \frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} + fu + \frac{\partial \Psi}{\partial y} \qquad /25/$$

$$g_3 = \frac{\partial \vartheta}{\partial t} + u \frac{\partial \vartheta}{\partial x} + v \frac{\partial \vartheta}{\partial y}$$

$$g_4 = \frac{\partial \Psi}{\partial t} + u \frac{\partial \Psi}{\partial x} + v \frac{\partial \Psi}{\partial y} - g w_0$$

doochodzimy ostateczne do równań:

$$\int_0^{\zeta} \left[g_1 + u \cdot D \right] d\tau = u \int_0^{\zeta} D d\tau - R \left[\int_0^{\zeta} \frac{\partial \vartheta}{\partial x} \tau^{\varkappa} d\tau - \zeta \int_0^{\zeta} \frac{\partial \vartheta}{\partial x} \tau^{\varkappa - 1} d\tau + \zeta \int_0^{1} \frac{\partial \vartheta}{\partial x} \tau^{\varkappa - 1} d\tau \right] \qquad /26/$$

$$\int_0^{\xi}\left[q_2+v\cdot D\right]d\tau = v\int_0^{\xi} D\,d\tau - R\left[\int_0^{\xi}\frac{\partial\vartheta}{\partial y}\tau^{\varkappa}d\tau - \xi\int_0^{\xi}\frac{\partial\vartheta}{\partial y}\tau^{\varkappa-1}d\tau + \xi\int_0^{1}\frac{\partial\vartheta}{\partial y}\tau^{\varkappa-1}d\tau\right] \qquad /27/$$

$$\int_0^{\xi}\left[q_3+\vartheta\cdot D\right]_1 d\tau = \vartheta\int_0^{\xi} D\,d\tau \qquad /28/$$

$$q_4 = -R\vartheta_k\int_0 D\,d\tau \quad . \qquad /29/$$

APROKSYMACJA RÓWNAŃ /26/ - /29/ METODĄ DORODNICYNA [3]

Niech $f(z)$ będzie dowolną, dostatecznie regularną funkcją zmiennej rzeczywistej z; $0 \leq z \leq 1$. Wybieramy w przedziale $[0,1]$ układ punktów:

$$0 = \xi_0 < \xi_1 < \xi_2 < \ldots < \xi_k = 1 \quad .$$

Utwórzmy dla funkcji $f(z)$ wielomian interpolacyjny $P(z)$ o węzłach w punktach $\xi_0, \xi_1, \ldots, \xi_k$.
Niech:

$$P(z) = \beta_0 + \beta_1 z + \ldots + \beta_k z^k \quad .$$

Mamy oczywiście dla α rzeczywistego:

$$\int_0^{\zeta_j} z^{\alpha} P(z)dz = \sum_{l=0}^{k} \beta_l \frac{1}{\alpha+l+1}\zeta_j^{\alpha+l+1}, \qquad \text{jeśli} \quad \alpha+l \neq -1 \;^{*)}$$

$$j = 0,1, \ldots, k .$$

*) W rozważanym przypadku jako α będzie występowała jedna z liczb $0, \varkappa-1, \varkappa$, zatem przypadek $\alpha + l = -1$ nie będzie mógł mieć miejsca.

Dla dowolnego $k + 1$ wymiarowego wektora

$$\bar{a} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix}$$

przyjmujemy oznaczenie

$$\bar{\bar{a}} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix}.$$

Jest zatem:

$$\bar{a} = \begin{bmatrix} a_0 \\ \hline \bar{\bar{a}} \end{bmatrix}.$$

Określimy wektor

$$I(\alpha; f) = \begin{bmatrix} 0 \\ \int_0^{\zeta_1} z^{\alpha} P(z)\,dz \\ \vdots \\ \int_0^{\zeta_k} z^{\alpha} P(z)\,dz \end{bmatrix}$$

Składowe tego wektora aproksymują wartości całki

$$\int_0^{\zeta_1} z^{\alpha} f(z)\,dz$$

dla $i = 0,1,\ldots, k$.

Jak nie trudno sprawdzić

$$I(\alpha; f) = \left[\begin{array}{c|c} 0 & \bar{\bar{0}}^T \\ \hline \bar{\bar{r}}(\alpha) & A(\alpha)\, X^{-1} \end{array}\right] \cdot \left[\begin{array}{c} f_0 \\ \hline \bar{\bar{f}} \end{array}\right]$$

gdzie

$$\bar{\bar{0}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}; \quad \bar{\bar{1}} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}; \quad \bar{\bar{z}}(\alpha) = \begin{bmatrix} \xi_1^{\alpha+1} \\ \xi_2^{\alpha+1} \\ \vdots \\ \xi_k^{\alpha+1} \end{bmatrix}; \quad \bar{\bar{f}} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_k \end{bmatrix}; \quad \bar{f} = \begin{bmatrix} f_0 \\ \hline \bar{\bar{f}} \end{bmatrix}; \quad f_j = f(\xi_j),$$

$$j = 0, 1, \ldots, k,$$

$$\bar{\bar{r}}(\alpha) = \frac{1}{\alpha+1}\bar{\bar{z}}(\alpha) - A(\alpha) \cdot X^{-1}\, \bar{\bar{1}}, \quad *)$$

$$A(\alpha) = \begin{bmatrix} \frac{1}{\alpha+2}\xi_1^{\alpha+2}, & \ldots, & \frac{1}{\alpha+k+1}\xi_1^{\alpha+k+1} \\ \ldots\ldots & \ldots\ldots & \ldots\ldots \\ \frac{1}{\alpha+2}\xi_k^{\alpha+2}, & \ldots, & \frac{1}{\alpha+k+1}\xi_k^{\alpha+k+1} \end{bmatrix}; \quad X = \begin{bmatrix} \xi_1 & \xi_1^2 & \ldots\ldots & \xi_1^k \\ \xi_2 & \xi_2^2 & \ldots\ldots & \xi_2^k \\ \ldots & \ldots & \ldots & \ldots \\ \xi_k & \xi_k^2 & \ldots\ldots & \xi_k^k \end{bmatrix}$$

Oznaczmy dla prostoty:

$$N(\alpha) = A(\alpha) \cdot X^{-1},$$

*) Dalej będziemy oznaczać konsekwentnie jedną kreską nad znakiem funkcji, $k+1$ wymiarowy wektor wartości tej funkcji w punktach $\xi_0, \xi_1, \ldots, \xi_k$, zaś podwójną kreską wektor zbudowany z wartości funkcji w punktach $\xi_1, \xi_2, \ldots, \xi_k$.

wtedy

$$\underset{\sim}{I}(\alpha;\, f) = \left[\begin{array}{c|c} Q & \underline{\underline{0}}^T \\ \hline \underline{\underline{r}}(\alpha) & N(\alpha) \end{array}\right] \cdot \left[\begin{array}{c} f_o \\ \hline \underline{\underline{f}} \end{array}\right] \qquad /30/$$

Oznaczmy jeszcze przez $\underline{\gamma}^T(\alpha)$ ostatni wiersz macierzy

$$M(\alpha) = \left[\begin{array}{c|c} 0 & \underline{\underline{0}}^T \\ \hline \underline{\underline{r}}(\alpha) & N(\alpha) \end{array}\right] .$$

Łatwo sprawdzić, że:

$$\underline{\gamma}^T(\alpha) = \left[\frac{1}{\alpha+1} - \underline{\underline{b}}^T(\alpha) \cdot X^{-1} \cdot \underline{\underline{1}} \;\middle|\; \underline{\underline{b}}^T(\alpha) \cdot X^{-1}\right] , \qquad /31/$$

gdzie

$$\underline{\underline{b}}(\alpha) = \left[\begin{array}{c} \frac{1}{\alpha+2} \\ \vdots \\ \frac{1}{\alpha+k+1} \end{array}\right] .$$

Zauważmy, że wyrażenie

$$\gamma^T(\alpha) \cdot \underline{f} \qquad /32/$$

aproksymuje całkę $\int_0^1 z^\alpha f(z)dz$.

Do równań /26/, /27/, /28/, /29/ podstawiamy kolejno $\zeta = \xi_0$, $\zeta = \xi_1$, ..., $\zeta = \xi_k$. Występujące w otrzymanych równaniach całki zastępujemy przez wyrażenia /30/ i /32/. Wprowadzając dodatkowe oznaczenia

$$U = \begin{bmatrix} u_1 & & 0 \\ & u_2 & \\ & & \ddots \\ 0 & & u_k \end{bmatrix}; \qquad V = \begin{bmatrix} v_1 & & 0 \\ & v_2 & \\ & & \ddots \\ 0 & & v_k \end{bmatrix};$$

$$\Theta = \begin{bmatrix} \vartheta_1 & & 0 \\ & \vartheta_2 & \\ & & \ddots \\ 0 & & \vartheta_k \end{bmatrix}; \qquad Z = \begin{bmatrix} \zeta_1 & & 0 \\ & \zeta_2 & \\ & & \ddots \\ 0 & & \zeta_k \end{bmatrix};$$

otrzymujemy następujący układ równań aproksymujący równania wyjściowe:

$$I(0;\ g_1 + u \cdot D) = \left[\begin{array}{c|c} 0 & \bar{0}^T \\ \hline \bar{0} & U \end{array}\right] \cdot I(0;\ D) -$$

$$- R\left\{ I\left(\varkappa;\ \frac{\partial \vartheta}{\partial x}\right) - \left[\begin{array}{c|c} 0 & \bar{0}^T \\ \hline \bar{0} & Z \end{array}\right] \cdot \left[I\left(\varkappa-1;\ \frac{\partial \vartheta}{\partial x}\right) - \bar{1} \cdot \bar{\gamma}^T(\varkappa-1)\ \frac{\partial \bar{\vartheta}}{\partial x} \right] \right\} \qquad /33/$$

$$I(0;\ g_2 + v \cdot D) = \left[\begin{array}{c|c} 0 & \bar{0}^T \\ \hline \bar{0} & V \end{array}\right] \cdot I(0;\ D) -$$

$$- R\left\{ I\left(\varkappa;\ \frac{\partial \vartheta}{\partial y}\right) - \left[\begin{array}{c|c} 0 & \bar{0}^T \\ \hline \bar{0} & Z \end{array}\right] \cdot I\left[\left(\varkappa-1;\ \frac{\partial \vartheta}{\partial y}\right) - \bar{1} \cdot \bar{\gamma}^T(\varkappa-1)\ \frac{\partial \bar{\vartheta}}{\partial y} \right] \right\} \qquad /34/$$

$$I\left(0;\ g_3 + \vartheta D\right) = \left[\begin{array}{c|c} 0 & \bar{0}^T \\ \hline \bar{0} & \theta \end{array}\right] \cdot I\left(0;\ D\right) \qquad /35/$$

$$g_4 = - R\, \vartheta_k\, \bar{\gamma}^T(0) \cdot \bar{D} \ , \qquad /36/$$

gdzie, zgodnie z przyjętą konwencją $\bar{1}$ oznacza $k + 1$ wymiarowy wektor $\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$.

Powyższy układ zawiera jedynie $3k + 1$ równań wiążących $3k + 4$ funkcje $\bar{u}(t,x,y)$; $\bar{v}(t,x,y)$; $\bar{\vartheta}(t,x,y)$; $\Psi(t,x,y)$. Dodatkowo trzy równania otrzymujemy aproksymując w analogiczny sposób całki w równaniach /22/, /23/, /24/.

Zastępując w równaniach /33/, /34/, /35/ pierwsze, zawsze równe zeru, współrzędne obu stron przez odpowiednie wyrażenia otrzymane z takiej aproksymacji równań /22/, /23/ i /24/, otrzymujemy ostatecznie następujący układ $3k + 4$ równań:

$$A \cdot \left[\bar{g}_1 + U_1 \cdot \bar{D}\right] - U_1 A \bar{D} + R \left\{ M(\varkappa) - S \left[M(\varkappa - 1) - \bar{1} \cdot \bar{\gamma}^T(\varkappa - 1) \right] \right\} \cdot \frac{\partial \bar{\vartheta}}{\partial x} = 0$$

$$A \cdot \left[\bar{g}_2 + V_1 \cdot \bar{D}\right] - V_1 A \bar{D} + R \left\{ M(\varkappa) - S \left[M(\varkappa - 1) - \bar{1} \cdot \bar{\gamma}^T(\varkappa - 1) \right] \right\} \cdot \frac{\partial \bar{\vartheta}}{\partial y} = 0$$

$$A \cdot \left[\bar{g}_3 + \theta_1 \cdot \bar{D}\right] - \theta_1 A \bar{D} = 0$$

$$g_4 + R\, \vartheta_k\, \bar{\gamma}^T(0) \cdot \bar{D} = 0$$

gdzie $A = \left[\begin{array}{c|c} 1 & \bar{\bar{0}}^T \\ \hline \bar{\bar{r}}(o) & N(0) \end{array}\right]$; $U_1 = \left[\begin{array}{c|c} 0 & \bar{\bar{0}}^T \\ \hline \bar{\bar{0}} & U \end{array}\right]$; $V_1 = \left[\begin{array}{c|c} 0 & \bar{\bar{0}}^T \\ \hline \bar{\bar{0}} & V \end{array}\right]$

$$\theta_1 = \left[\begin{array}{c|c} 0 & \bar{\bar{0}}^T \\ \hline \bar{\bar{0}} & \theta \end{array}\right]; \qquad S = \left[\begin{array}{c|c} 1 & \bar{\bar{0}}^T \\ \hline \bar{\bar{0}} & Z \end{array}\right].$$

Łatwo się przekonać, że macierz A jest odwracalna. Stąd:

$$\bar{g}_1 + \left[U_1 - A_1^{-1} U_1 A\right] \cdot \bar{D} + B \cdot \frac{\overline{\partial \vartheta}}{\partial x} = 0 \qquad /37/$$

$$\bar{g}_2 + \left[V_1 - A^{-1} V_1 A\right] \cdot \bar{D} + B \cdot \frac{\overline{\partial \vartheta}}{\partial y} = 0 \qquad /38/$$

$$\bar{g}_3 + \left[\theta_1 - A^{-1} \theta_1 A\right] \cdot \bar{D} = 0 \qquad /39/$$

$$g_4 + R\, \vartheta_k\, \bar{\gamma}^T(0) \cdot \bar{D} = 0\,, \qquad /40/$$

gdzie

$$B = R \cdot A^{-1} \left\{ M(\varkappa) - S \left[M(\varkappa-1) - \bar{1} \cdot \bar{\gamma}^T(\varkappa-1) \right] \right\}.$$

Oznaczmy dodatkowo

$$H(P) = P - A^{-1} PA\,,$$

gdzie P jest jedną z macierzy U_1, V_1, θ_1.

Utwórzmy teraz wektor $3k + 4$ wymiarowy $\bar{\rho}(t,x,y)$

$$\bar{\rho}(t,x,y) = \begin{bmatrix} \bar{u}(t,x,y) \\ \bar{v}(t,x,y) \\ \bar{\vartheta}(t,x,y) \\ \Psi(t,x,y) \end{bmatrix}$$

Po uporządkowaniu, układ /37/ - /40/ przyjmie postać:

$$\frac{\partial \bar{\rho}}{\partial t} + C_1(\rho)\frac{\partial \bar{\rho}}{\partial x} + C_2(\rho)\frac{\partial \bar{\rho}}{\partial y} + C_3(x,y)\bar{\rho} = \bar{W}, \qquad /41/$$

gdzie

$$C_1(\rho) = \left[\begin{array}{c|c|c|c} U_1+H(u) & 0 & B & \bar{1} \\ \hline H(v) & U_1 & 0 & \bar{0} \\ \hline H(\vartheta) & 0 & U_1 & \bar{0} \\ \hline R\vartheta_k\bar{\gamma}^T(0) & \bar{0}^T & \bar{0}^T & u_k \end{array}\right]; \quad C_2(\rho) = \left[\begin{array}{c|c|c|c} V_1 & H(u) & 0 & \bar{0} \\ \hline 0 & V_1+H(v) & B & \bar{1} \\ \hline 0 & H(\vartheta) & V_1 & \bar{0} \\ \hline \bar{0}^T & R\vartheta_k\bar{\gamma}^T(0) & \bar{0}^T & v_k \end{array}\right];$$

$$C_3(x,y) = \left[\begin{array}{c|c|c|c} 0 & -fE & 0 & \bar{0} \\ \hline fE & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \bar{0} \\ \hline \bar{0}^T & \bar{0}^T & \bar{0}^T & 0 \end{array}\right]; \qquad \bar{W} = \left[\begin{array}{c} \bar{0} \\ \hline \bar{0} \\ \hline \bar{0} \\ \hline w_0 g \end{array}\right];$$

E - macierz jednostkowa wymiaru $k + 1$

0 - macierz zerowa wymiaru $k + 1$.

Zauważmy, że macierze $C_1(\rho)$ i $C_2(\rho)$ zależą w sposób liniowy od rozwiązania. Macierz $C_3(x,y)$ jest funkoją x, y.

Rozwiązanie $\bar{\rho}(t,x,y)$ określa funkoje $\bar{u}(t,x,y)$, $\bar{v}(t,x,y)$, $\bar{\vartheta}(t,x,y)$, $\Psi(t,x,y)$ czyli składowe prędkości wiatru i temperaturę potenojalną na poziomach $\zeta_0, \zeta_1, \ldots, \zeta_k$ oraz geopotenojał na poziomie ζ_k.

Korzystając z $\bar{\mathcal{V}}(t,x,y)$ możemy wyznaczyć przy pomocy wzoru /9/ wektor geopotencjału $\tilde{\Phi}(t,x,y)$, określający geopotencjał na poziomach $\zeta_1, \zeta_2, \ldots, \zeta_{k-1}$. W celu przybliżonego wyznaczenia wektora $\tilde{\Phi}(t,x,y)$ można posłużyć się wzorem podobnym do /30/.

Zdefiniujmy w tym celu podobnie jak poprzednio, dla dowolnej funkcji $f(z)$ zmiennej rzeczywistej z następujący wektor $k-1$ wymiarowy:

$$J(\alpha;\ f) = \begin{bmatrix} \int_{\zeta_1}^{1} z^{\alpha} P(z)\,dz \\ \vdots \\ \int_{\zeta_{k-1}}^{1} z^{\alpha} P(z)\,dz \end{bmatrix},$$

gdzie $P(z)$ jest wielomianem interpolacyjnym funkcji $f(z)$, opartym na węzłach $\zeta_0, \zeta_1, \ldots, \zeta_k$.

Składowe wektora $J(\alpha\ ;\ f)$ stanowią aproksymację całki

$$\int_{\zeta_1}^{1} f(z)\, z^{\alpha}\, dz$$

dla $i = 1, \ldots, k-1$. Łatwo sprawdzić, że przy zachowaniu oznaczeń z wzoru /30/ i poprzednich, otrzymujemy:

$$J(\alpha\ ;\ f) = K(\alpha) \begin{bmatrix} f_0 \\ \hline X^{-1}\left(\bar{f} - \bar{1}\, f_0\right) \end{bmatrix}$$

gdzie:

$$K(\alpha) = \begin{bmatrix} \frac{1}{\alpha+1}\left(1-\zeta_1^{\alpha+1}\right), \ldots, \frac{1}{\alpha+k+1}\left(1-\zeta_1^{\alpha+k+1}\right) \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \frac{1}{\alpha+1}\left(1-\zeta_{k-1}^{\alpha+1}\right), \ldots, \frac{1}{\alpha+k+1}\left(1-\zeta_{k-1}^{\alpha+k+1}\right) \end{bmatrix}$$

Ostatecznie dla wyznaczenia wektora geopotencjału na poziomach $\zeta_1, \zeta_2, \ldots, \zeta_{k-1}$, otrzymujemy następujący wzór przybliżony:

$$\tilde{\Phi}(t,x,y) = \Psi(t,x,y)\ \tilde{1} + R \cdot K(\varkappa - 1) \cdot \left[\frac{\vartheta_0(t,x,y)}{\chi^{-1}\left(\bar{\vartheta}(t,x,y)\ \tilde{1}\ \vartheta_0(t,x,y)\right)} \right] \quad /42/^{*)}$$

Zauważmy, że wzór /42/ może być stosowany niezależnie od procesu rozwiązywania równań /41/, dla wybranych wartości zmiennej czasowej t.

Przy wyznaczaniu $\bar{\Phi}$ posługiwać się można oczywiście także i innymi znanymi metodami obliczania przybliżonego całek. Wydaje się jednak celowe zastosowanie jednolitej metody aproksymacji wszystkich funkcji rozważanych. Zauważmy wreszcie, że wzór /42/ uwzględnia przebieg funkcji ϑ na wszystkich poziomach $\zeta_0, \zeta_1, \ldots, \zeta_k$.

Dla równań typu /41/ można próbować stawiać zarówno warunki początkowe, jak też i zagadnienia początkowo-brzegowe typu:

$$\bar{\rho}(0,x,y) = \bar{\rho}_0(x,y) \quad \text{dla} \quad (x,y) \in S$$
$$\bar{\rho}(t,x,y) = \bar{\rho}_b(x,y) \quad \text{dla} \quad (x,y) \in \partial S \ ,$$

*) $\tilde{1}$ oznacza k-1 wymiarowy wektor kolumnowy o wszystkich współrzędnych równych 1.

gdzie ∂S oznacza brzeg obszaru płaskiego S, w którym rozważa się zagadnienie, a

$$\left.\begin{array}{l}\bar{\rho}_0(x,y)\\ \bar{\rho}_b(t,x,y)\end{array}\right\} \text{ funkcje zadane.}$$

Literatura

[1] THOMPSON P.D.: Numerical Weather Analysis and Prediction. The Macmillan C°, New Yrok 1961.

[2] KADYŠNIKOW W.M.: O kratkosrocnom prognozie mieteorologičeskich eliemientow po polnym urawnienijam gidrodinamiki. Trudy wycislitielnogo mieteorologiceskogo cientra, wypusk 1, 1963.

[3] DORODNICYN A.A.: Ob odnom mietodie čisliennogo riešenija niekotorych zadač aerodinamiki. Trudy III wsiesojuznogo matiematiceskogo sjezda, 2. 1956.

ON A CERTAIN MULTILEVEL MODEL OF SHORT-RANGE WEATHER FORECAST

Summary

In this paper, the so called primitive equations of weather forecast /1/ - /5/ have been replaced by a system /41/ of differential equations, still not involving the so called 'pressure variable ξ'

The obtained quasilinear system /41/, being in 'standard form', seems to be more convenient for further investigations.

The method applied here is similar to the one described in [2], but the whole idea of approximation differs in many points.

STATISTICAL METHODS

Instytut Maszyn Matematycznych
Algorytmy No 7

519.214.3

GENEROWANIE REALIZACJI PROCESU POISSONA ZE ZMIENNĄ INTENSYWNOŚCIĄ

Lidia ŁUKASZEWSKA
Elżbieta PLESZCZYŃSKA

Pracę złożono 19.7.1966 r.

Idee generowania realizacji procesu Poissona ze zmienną intensywnością opisano w [2], str. 131-134. Na tej podstawie opracowano przedstawiony w niniejszym artykule ogólną sieć działań odpowiedniego generatora oraz szczegółowe sieci działań w przypadku gdy chwilowa intensywność procesu jest funkcją przedziałami stałą oraz funkcją ciągłą przedziałami liniową.

Proces Poissona ze zmienną intensywnością /patrz [1]/ jest to taki proces stochastyczny $X(t)$, w którym

$$P\left(X(t+\tau) - X(t) = k\right) = e^{-\Lambda(\tau,t)}\frac{[\Lambda(\tau,t)]^k}{k!}, \qquad /1/$$

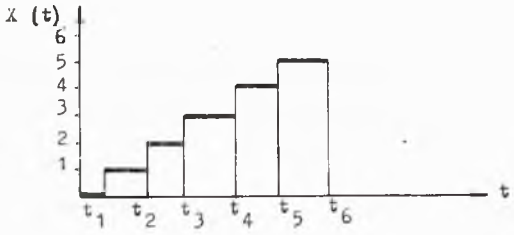
$$(k = 0, 1, 2, \ldots;\ t \geq 0, \tau > 0),$$

gdzie $\Lambda(\tau,t) = \int_t^{t+\tau} \lambda(u)\,du$.

Funkcja $\lambda(t)$ nosi nazwę chwilowej intensywności procesu i jest rzeczywistą nieujemną funkcją parametru t.

W teorii obsługi masowej zakłada się niekiedy, że proces zgłoszeń /proces wejścia/ jest takim właśnie procesem. Okazuje się on często lepszym przybliżeniem rzeczywistości niż proces Poissona ze stałą intensywnością.

Realizacjami procesu $X(t)$ są krzywe schodkowe /rys. 1/



Rys.1. Realizacja procesu $X(t)$.

Jak wiadomo, w procesie Poissona przyrosty tego procesu są równe 1. Tak więc generowanie realizacji procesu $X(t)$ w przedziale $(0,T)$ sprowadza się do generowania ciągu wartości parametru t_1, t_2, ..., $t_n (t_n < T, t_{n+1} > T)$, przy których następują zmiany stanu procesu. Gdy $X(t)$ interpretujemy jako proces zgłoszeń, wówczas wielkości t_i oznaczają chwilę czasu, w których nadchodzą kolejne zgłoszenia. W dalszej części pracy przyjmujemy taką właśnie interpretację. Sposób generowania ciągu $\{t_i\}$ podano w [2].

W niniejszej pracy opisana jest pewna modyfikacja tego sposobu. Wykorzystując odpowiednio generator liczb o rozkładzie wykładniczym zamiast generatora liczb o rozkładzie równomiernym udało się skrócić nieco czas generacji ciągu $\{t_i\}$. Zbyt skomplikowana postać funkcji $\lambda(u)$ może bardzo utrudnić rozwiązanie zadania i w praktyce może okazać się nieopłacalne stosowanie podanego generatora. Dlatego też w części szczegółowej podana jest sieć działań w przypadku, gdy funkcja $\lambda(u)$ jest funkcją przedziałami stałą i ogólniej, gdy jest funkcją przedziałami liniową. W praktyce przypadki te powinny być wystarczające.

Poniżej podany jest opis, w jaki sposób zbudowano generator.

Niech ξ_t oznacza okres czasu, jaki upływa od chwili t do momentu nadejścia kolejnego zgłoszenia w rozpatrywanym wyżej procesie $X(t)$. Zatem w czasie ξ_t nie ma żadnego zgłoszenia i stąd

$$P(\xi_t < u) = 1 - P(\xi_t \geq u) = 1 - P(X(t+u) - X(t) = 0) . \quad /2/$$

Niech zmienna losowa T_k oznacza moment k-tego zgłoszenia ($k = 1,2, \ldots$). Wobec tego, na podstawie wzoru /1/ i /2/ dystrybuanta zmiennej losowej T_1 ma postać

$$F(y) = 1 - e^{-\Lambda(y,0)} . \qquad /3/$$

Z kolei niech zmienna losowa $Z_k = T_k - T_{k-1}$, $(k = 2, 3, \ldots)$ oznacza czas upływający między $k-1$ -szym a k-tym zgłoszeniem. Zatem wartościami zmiennej losowej Z_k są $z_k = t_k - t_{k-1}$. Ze wzorów /1/ i /2/ dystrybuanty zmiennej losowej Z_k pod warunkiem że zgłoszenie o numerze $k-1$ nastąpiło w chwili t_{k-1} ma postać

$$F_k(y \mid t_{k-1}) = 1 - e^{-\Lambda(y,t_{k-1})} , \qquad (k = 2, 3, \ldots). /4/$$

Na podstawie /3/ i /4/ można kolejno generować t_1, $t_2 = t_1 + z_2$, $t_3 = t_2 + z_3$, ... póki nie przekroczy się granicy przedziału $(0,T)$, co jest równoważne z zakończeniem generacji ciągu $\{t_i\}$.

Z wzorów /3/ i /4/ wynika, że zmienne losowe $\Lambda(T_1,0)$ oraz $\Lambda(Z_k,t_{k-1})$ /dla ustalonego t_{k-1}/ mają rozkłady wykładnicze z parametrem 1. Wobec tego można generować ciąg wartości w_k zmiennej losowej o rozkładzie wykładniczym z parametrem 1 i znajdować t_k ze związków

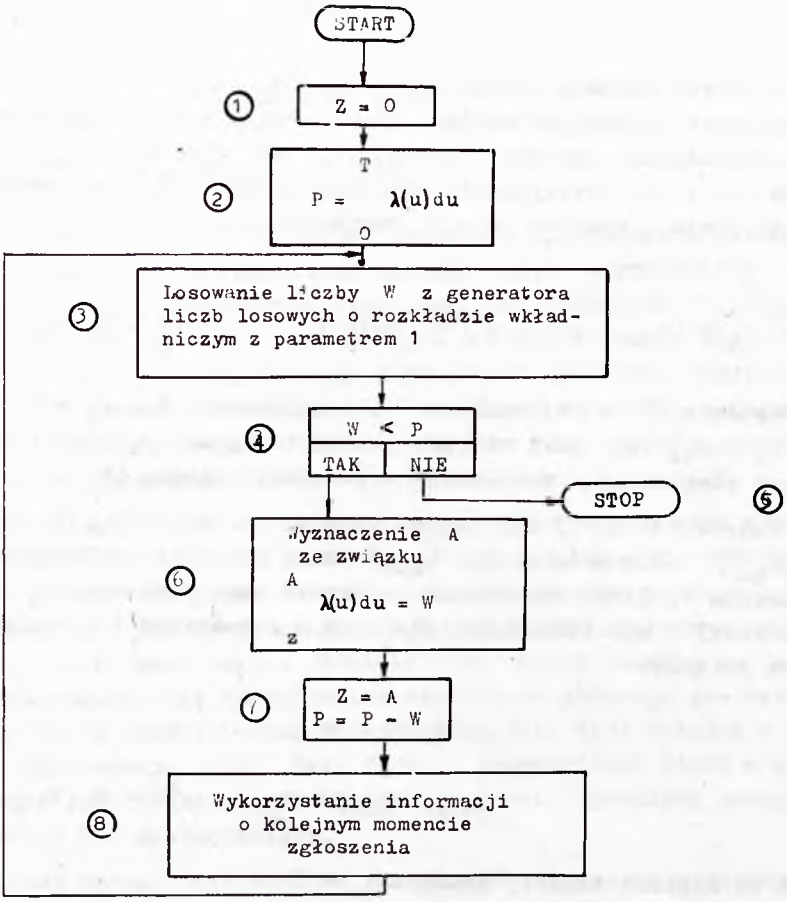
$$\Lambda(t_1, 0) = w_1 ,$$

$$\Lambda(z_k, t_{k-1}) = w_k , \qquad (k = 2, 3, \ldots).$$

Można to zapisać inaczej kładąc $t_0 = 0$

$$\int_{t_{k-1}}^{t_k} \lambda(u)du = w_k , \qquad (k = 1, 2, \ldots). /5/$$

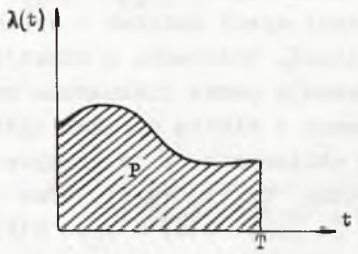
Ogólną sieć działań generatora chwil t_k na odcinku $(0, T)$ przedstawia rysunek 2.



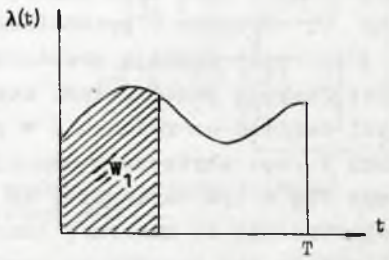
Rys.2. Ogólna sieć działań generatora realizacji procesu Poissona ze zmienną intensywnością.

Jeżeli będzie się wykonywać operacje według tej sieci działań począwszy od skrzynki ①, wówczas przy pierwszym przebiegu na wyjściu ze skrzynki ⑦ otrzyma się jako wartość zmiennej Z moment t_1 nadejścia pierwszego zgłoszenia /jeśli tylko $t_1 < T$/. Informację o chwili t_1 można bądź wykorzystać od razu do innych obliczeń, bądź też zachować w jakimś miejscu pamięci maszyny /zależy to od rozwiązywanego zadania/. Powracając do skrzynki ③ i powtarzając przebieg na wyjściu ze skrzynki ⑦ otrzymuje się wartość Z odpowiadającą momentowi t_2 /jeśli tylko $t_2 < T$/. Powtarzając przebiegi od skrzynki ③ do skrzynki ⑦ otrzymuje się kolejno momenty t_3, t_4, ... Jeśli przy badaniu warunku w skrzynce ④ okaże się, że następny moment zgłoszenia nastąpiłby w chwili $t_{n+1} > T$, wówczas przerywa się proces generacji i przechodzi do skrzynki ⑤ /oznacza to, że cały ciąg $\{t_i\}$ został już wygenerowany/.

W skrzynce ② oblicza się pole P. Na rysunku 3 pole to jest zakreskowane.



Rys. 3.



Rys. 3a.

Przy pierwszym losowaniu /skrzynka ③/ otrzymuje się z generatora liczb losowych liczbę $W = w_1$, która może być większa lub mniejsza od P. Jeżeli $W > P$, oznacza to, że żadne zgłoszenie nie nastąpi w przedziale czasu (0,T) i proces generacji jest zakończony /skrzynka ⑤/.

W przeciwnym przypadku moment zgłoszenia wyznacza się na osi czasu t w taki sposób, aby odpowiednia część powierzchni P z rysunku 3 była równa w_1 /rys. 3a/. Przy ustalaniu następnego momentu zgłoszenia trzeba wygenerować liczbę $W = w_2$ i porównać ją z polem P

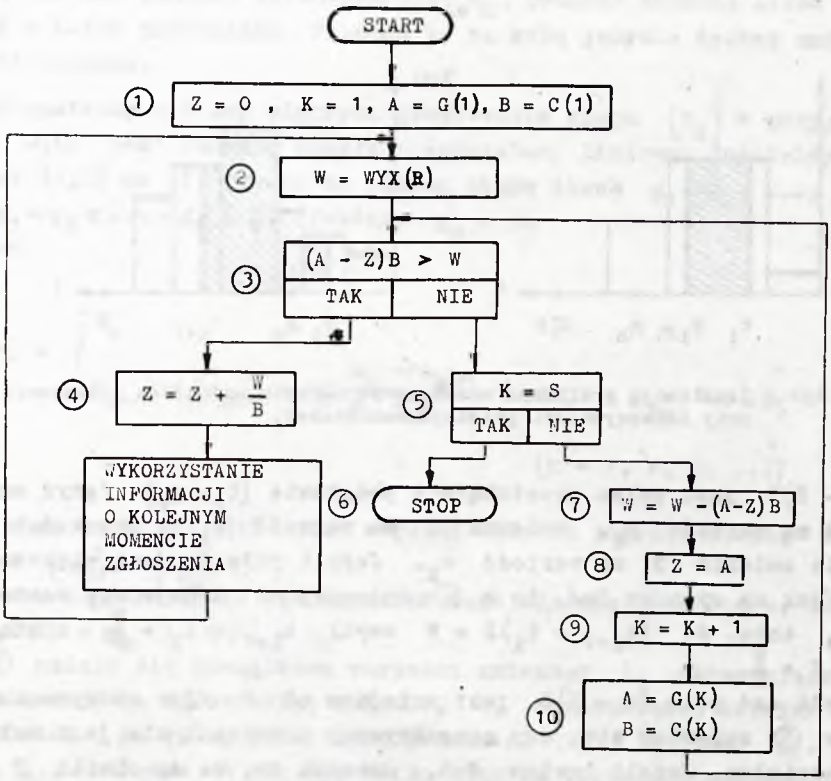
zmniejszonym już o wielkość w_1. W tym celu w skrzynce ⑦ dokonuje się odpowiednie operacje.

Sieć działań z rysunku 2. jest słuszna dla wszelkich postaci funkcji $\lambda(u)$. Nie mniej jednak, jak już zaznaczono powyżej, bardziej złożona postać tej funkcji może być powodem zbytniej powolności działania tego generatora.

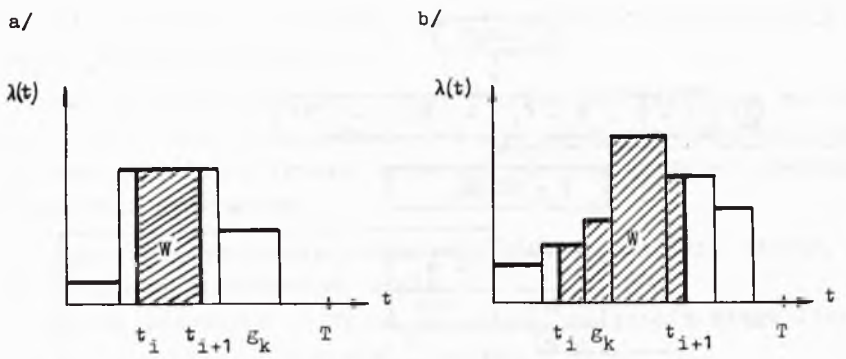
Algorytm generowania ciągu $\{t_i\}$ jest szczególnie prosty, gdy $\lambda(t)$ jest funkcją przedziałami stałą.
Podzielmy przedział $(0,T)$ na S części za pomocą ciągu liczb
$g_1, g_2, \ldots, g_S, \quad 0 < g_1 < g_2 \ldots < g_S = T.$
Niech

$$\lambda(t) = \begin{cases} c_1 & \text{dla} \quad 0 \le t < g_1 , \\ c_k & \text{dla} \quad g_{k-1} \le t < g_k , \quad (k = 2,3, \ldots, S) \end{cases} \quad /6/$$

Funkcja $\lambda(t)$ jest więc zadana za pomocą ciągów $\{c_k\}$, $\{g_k\}$ i liczby S. Rysunek 4. przedstawia wariant sieci działań z rysunku 2, gdy $\lambda(t)$ jest funkcją przedziałami stałą. Założenie o funkcji $\lambda(t)$, że jest funkcją przedziałami stałą powoduje pewne nieznaczne zmiany w sieci działań na rysunku 4 w porównaniu z siecią działań ogólną z rysunku 2./np. zbytecznym okazuje się obliczanie P w skrzynce ②/. Zakłada się w tym wariancie, że parametry $\{c_k\}$, $\{g_k\}$, oraz S są już wprowadzone do maszyny; oznaczono je jako $C(K) = c_k$, $G(K) = g_k$. W ① nadaje się początkowe wartości: zmiennej Z, oznaczającej moment nadejścia zgłoszenia; zmiennej K, oznaczającej kolejny numer przedziału, na które podzielono przedział $(0,T)$ liczbami $g_1, g_2, \ldots, g_S$; zmiennej A, oznaczającej górną granicę k-tego przedziału; zmiennej B, oznaczającej intensywność w k-tym przedziale. W ② generuje się liczbę W, będącą realizacją zmiennej losowej o rozkładzie wykładniczym z parametrem 1. Oznaczenie $W = WYX(R)$ wzięto z pracy [3], gdzie jest podany opis generatora takich liczb. W ③ sprawdza się, czy moment kolejnego zgłoszenia ma nastąpić w rozpatrywanym przedziale czasu /przed chwilą A/, to znaczy, czy W nie przekracza $(A - Z)B$. Ilustrację tego znajdziemy na rysunku 5.



Rys. 4. Sieć działań generatora realizacji procesu Poissona z intensywnością przedziałami stałą.



Rys.5 Ilustracją graficzna wyznaczania momentu nadejścia zgłoszenia przy intensywności przedziałami stałej.

$(A - Z)B$ jest polem prostokąta o podstawie (t_i, g_k) /gdyż zmienna A ma wartość g_k, zmienna Z ma wartość t_i/ i wysokości B, gdzie zmienia B ma wartość c_k. Jeżeli pole to jest większe od W /jak na rysunku 5a/, to w ④ zmiennej Z nadaje się wartość t_{i+1} taką, że $(t_{i+1} - t_i)B = W$ czyli $t_{i+1} = t_i + \frac{W}{B}$, a stąd $Z = Z + \frac{W}{B}$.

Jeżeli zaś pole $(A - Z)B$ jest mniejsze od W /jak na rysunku 5b/, to w ⑤ sprawdza się, czy rozpatrywany przedział nie jest ostatnim przedziałem. Jeżeli bowiem K=S, oznacza to, że do chwili T nie nastąpi nowe zgłoszenie; przechodzi się wtedy do skrzynki ⑥, czyli następuje koniec generacji.

Jeżeli rozpatrywany przedział nie jest ostatnim, należy "przenieść się" do następnego przedziału zmniejszając uprzednio w ⑦ wielkość W o pole $(A - Z)B$ oraz nadając w ⑧ wartość g_k zmiennej Z. Nowy przedział ma numer porządkowy o 1 większy niż poprzedni, zmienia się też intensywność i górna granica przedziału. Trzeba więc wykonać operacje w ⑨ i ⑩ i powrócić do wykonywania operacji w ③ i skrzynkach następnych.

Na wyjściu skrzynki ④ otrzymuje się kolejno wartości zmiennej Z odpowiadające momentom $t_1, t_2, \ldots, t_n (t_n < T)$. Po wykorzystaniu informacji o kolejnym momencie zgłoszenia t_k, chwilę t_{k+1} generuje się wykonując operacje od skrzynki ②. W praktyce wygodnie

jest czasami zamiast intensywności c_k podawać średnią ilość zgłoszeń w k-tym przedziale. Pociąga to za sobą jedynie drobne zmiany sieci działań.

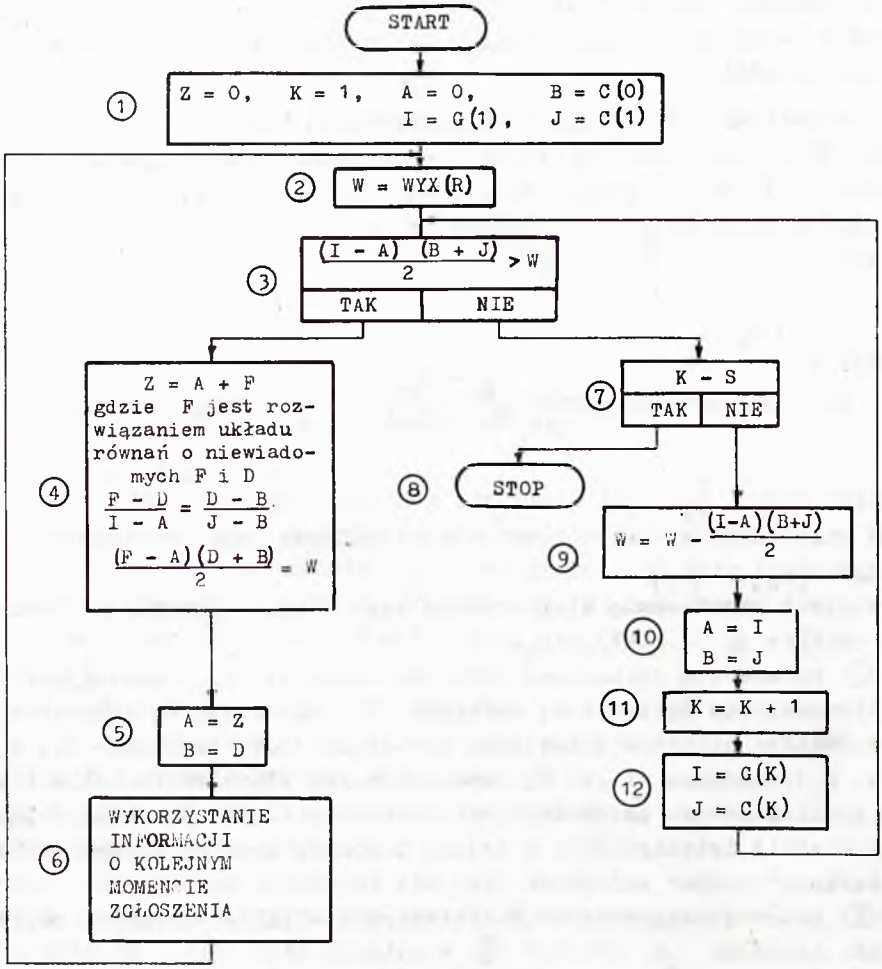
Rozpatrzmy z kolei algorytm generowania ciągu $\{t_i\}$ w przypadku, gdy $\lambda(t)$ jest funkcją ciągłą przedziałami liniową. Podzielmy przedział $(0,T)$ na S części za pomocą ciągu liczb $g_1, g_2, \ldots, g_S$, $0 < g_1 < g_2 < \ldots < g_S = T$. Połóżmy $g_0 = 0$.
Niech

$$\lambda(t) = \begin{cases} c_0 & \text{dla} \quad t = 0 \\ c_{k-1} - (g_{k-1} - t)\dfrac{c_k - c_{k-1}}{g_k - g_{k-1}} & \text{dla} \quad g_{k-1} < t \leq g_k \end{cases}$$

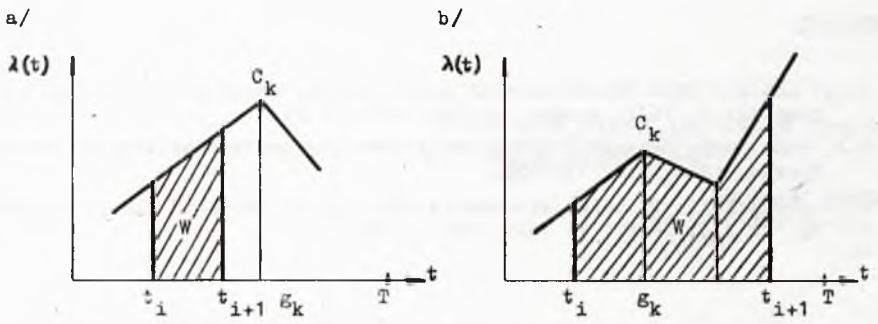
$$(k = 1, 2, 3, \ldots) \qquad /7/$$

Tak więc funkcja $\lambda(t)$ jest w pełni opisana poprzez liczbę S i ciągi $\{g_k\}$, $\{c_k\}$.
Rysunek 6. przedstawia sieć działań tego właśnie generatora. Oznaczono $G(K) = g_k$ i $C(K) = c_k$.
W ① nadaje się początkowe wartości zmiennej Z, oznaczającej moment nadejścia zgłoszenia; zmiennej K, oznaczającej kolejny numer przedziału, na które podzielono przedział $(0,T)$ liczbami $g_1, g_2, \ldots, g_S$; zmiennym A i I, oznaczających odpowiednio dolną i górną granicę k-tego przedziału oraz zmiennym B i J, oznaczających odpowiednie intensywności w dolnej i górnej granicy k-tego przedziału.
W ② generuje się liczbę W /identycznie jak w wariancie poprzednim/.
W ③ sprawdza się, czy moment kolejnego zgłoszenia ma nastąpić w rozpatrywanym przedziale czasu /przed chwilą I/, to znaczy, czy W nie przekracza $\frac{(I-A)(B+J)}{2}$. Ilustrację tego znajdziemy na rysunku 7.

Jeżeli pole trapezu $\frac{(I-A)(B+J)}{2}$ jest większe od liczby W /jak na rysunku 7a/, to w ④ wyznacza się moment zgłoszenia Z w sposób następujący: na osi t znajduje się taki punkt F, iż pole trapezu /punkt F jest jednym z jego wierzchołków/ jest równe W.



Rys. 6. Sieć działań generatora realizacji procesu Poissona z intensywnością przedziałami liniową.



Rys.7. Ilustracja graficzna wyznaczania momentu nadejścią zgłoszenia przy intensywności przedziałami liniowej.

Taki tok postępowania prowadzi do układu 2 równań, w których niewiadomymi są: chwila zgłoszenia i odpowiadająca tej chwili intensywność /oznaczono ją literą D/. Na wyjściu ze skrzynki ④ otrzymuje się więc kolejny moment zgłoszenia i intensywność. Po zarejestrowaniu tych informacji w skrzynce ⑤ i wykorzystaniu ich w skrzynce ⑥ można przejść do generowania następnego momentu zgłoszenia, poczynając od skrzynki ②.

Jeżeli pole trapezu jest mniejsze od W /jak na rysunku 7b/, to w ⑦ sprawdza się, czy rozpatrywany przedział nie jest ostatnim, gdyż wówczas oznaczałoby to, że następne zgłoszenie nastąpiłoby po chwili T, a więc generowanie należy zakończyć w ⑧.

Jeżeli rozpatrywany przedział nie jest ostatnim, wówczas po zmniejszeniu W w ⑨ o pole trapezu obliczonego i "przeniesieniu się" do następnego przedziału w skrzynkach ⑩, ⑪ i ⑫ należy powrócić do skrzynki ③.

Generowanie ciągu $\{t_i\}$ pierwszym algorytmem trwa krócej niż przy stosowaniu drugiego algorytmu. Jednakże w niektórych przypadkach przybliżanie intensywności zgłoszeń funkcją ciągłą przedziałami liniową bardziej zbliża model do rzeczywistości.

Pierwszy algorytm znalazł już zastosowanie w modelowaniu pewnego jednokanałowego procesu obsługi.

Literatura

[1] Józef ŁUKASZEWICZ: Teoria kolejek czyli obsługi masowej. Zastosowania Matematyki, t. VIII, zeszyt 1, 1965, str. 13-26.

[2] N.P. BUSLIENKO: Matematiceskoje modielirovanie proizvodstviennych processov. Moskva, 1964, str. 131-134.

[3] E. PLESZCZYŃSKA: Technika stosowania metod Monte-Carlo na ZAM-2. Algorytmy, t. III, zeszyt 5, 1965, str. 37-64.

GENERATING A POISSON-PROCESS WITH A TIME-DEPENDING INTENSITY

Summary

Formula (1) defines a Poisson-process with a time-depending intensity. Realisation of this process in the interval $(0,T)$ is given by a sequence of moments $\{t_i\}$, in which changes of state occur. The idea of generation of such a sequence is presented in [2], p. 131-134. On this basis the general flow-diagram of the generator has been made /Fig. 2/. Two particular cases are treated in detail: when intensity $\lambda(t)$ is piece-wise constant /flow-diagram on Fig. 4/ and when $\lambda(t)$ is continous and piece-wise linear /flow-diagram on Fig. 6/.

In practice any function $\lambda(t)$ can be approximated by functions of that kind. The described method was applied to generating an arrival process in a queueing model.

Instytut Maszyn Matematycznych
Algorytmy N° 7

519.28:312.1.2

ON CERTAIN BIRTH AND DEATH PROCESSES AND THEIR SIMULATION

by Józef WINKOWSKI
Received May 24-th, 1966

A method of the description of some birth and death processes is given, destined for these processes run simulation on computers. Such birth and death processes are considered the runs of which are determined by lifetimes of certain individuals only. A certain simulation method is described.

The paper aims at:

Presenting a sufficiently general description of the development of population composed of individuals, the lifetimes of which are arbitrary random variables, and the motivation of the above mentioned description,

Presenting the method of simulation of these processes whioh is to be realized on computers.

It is endeavoured to reach such a degree of generality whioh would permit to insert in a proposed soheme processes, the runs of which are determined by the lifetimes of certain individuals. It seems that, besides some biologioal processes, this is the property of many other ones appearing in a mass service or in production /compare examples given in the paper/. This speaks for their uniform treating.

1. THE PROCESS SCHEME

Speaking about the process we shall keep in mind a stochastic process which is a family of random variables on a certain proba-

bility space $(\Omega, \mathcal{F}, P)$. To describe the discussed process we shall consider a system with a finite or countable set S of states. Let Λ be the finite set which is the union of not empty pairwise disjoint sets $\Lambda_1, \ldots, \Lambda_m$.

As considered schemes of birth and death processes we shall understand ordered sets $< S, \Lambda_1, \ldots, \Lambda_m, \pi, \alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m >$ such that:

/I/ to every $s \in S$ corresponds the set $\Gamma(s) \subset S$ /which may be empty/; all states to which the system can pass directly from the state s, belongs to $\Gamma(s)$,

/II/ to every pair of states $s, s_1 \in \Gamma(s)$ corresponds exactly one element $\pi(s, s_1) \in \Lambda$; the set of the value of the function π is identical with Λ,

/III/ for every $i = 1, \ldots, m$ and for the state s there exist at most one state $s_1 \in \Gamma(s)$ such that $\pi(s, s_1) \in \Lambda_i$,

/IV/ for every $i = 1, \ldots, m$ and for $s \in S$ a non negative entire number $\alpha_i(s)$ is determined; $\pi(s, s_1) \in \Lambda_i$ if and only if $\alpha_i(s) > 0$;

/V/ for every $j = 1, \ldots, m$ and $\lambda \in \Lambda$ a non negative entire number $\beta_j(\lambda)$ is determined; if $\lambda = \pi(s, s_1) \in \Lambda_i$ for $s_1 \in \Gamma(s)$ then $\alpha_j(s_1) = \alpha_j(s) + \beta_j(\lambda)$ when $j \neq i$ and $\alpha_i(s_1) = \alpha_i(s) + \beta_i(\lambda) - 1$.

Such a scheme is a mathematical description of the event consisting in the perishing of single individuals and their being replaced by others. $\alpha_i(s)$ represents the number of individuals of the type i in the state s. $\pi(s, s_1)$ characterizes the act of replacing one individual by another while passing from state s to state s_1, indicating, among others, the set Λ_i, i.e. the type of the perishing individual, $\beta_j(\pi(s, s_1))$ constitutes the number of new individuals of the type j, which appear in this act. It only depends on $\pi(s, s_1)$.

Example 1

Let us consider the system composed of three machines and two workers. First, the machines are working alone but they may under-

go to damages. The workers' task is to repair the damages and restore the machines to work. Every damaged machine is being repaired by one worker and while doing this he can't possibly repair another one. Let A_1 denote the working machine, A_2 – the assembly the damaged machine and the repairing worker, A_3 – the damaged but not repaired machine, and A_4 – the idle worker. States that can be written as given bellow belong to the set S.

$$3A_1 + 2A_4$$

$$2A_1 + A_2 + A_4$$

$$A_1 + 2A_2$$

$$2A_2 + A_3 .$$

We determine $\Lambda = \Lambda_1 \cup \Lambda_2$ where $\Lambda_1 = \{\lambda_1, \lambda'_1\}, \Lambda_2 = \{\lambda_2, \lambda'_2\} \cdot \lambda_1$ denotes the damage of the machine and the beginning of its repair, λ'_1 – the damage only, λ_2 – the repair of the machine and the beginning of a new reparation, and λ'_2 – the repairing of the machine only. Functions π, α_i, β_j can be described by means of a graphic representation of the process scheme. For this purpose the pass from one state to another is denoted by a directed line described by a proper value of the function π.

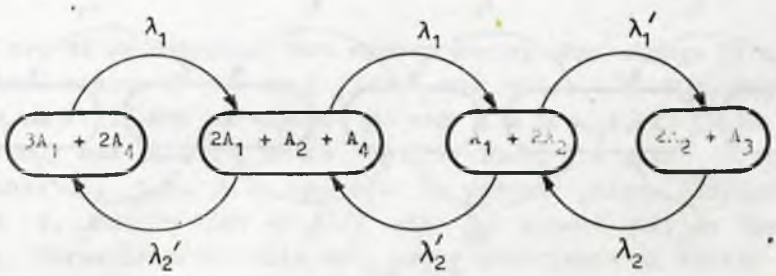


Fig. 1.

Example 2

Let us take into account a system composed of the source of certain units, an unlimited waiting room and two servers that

serve the units. Each unit is served by one server and he can't serve another one at the same time. Let A_1 denote the source of units, A_2 – the set: server – unit which is served, A_3 – the unit waiting to be served, and A_4 – the idle server. States written as

$$A_1 + 2A_4$$

$$A_1 + A_2 + A_4$$

$$A_1 + 2A_2$$

$$A_1 + 2A_2 + A_3$$

$$A_1 + 2A_2 + 2A_3$$

belong to set S.

Let us assume $\Lambda = \Lambda_1 \cup \Lambda_2$, where $\Lambda_1 = \{\lambda_1, \lambda_1'\}$, $\Lambda_2 = \{\lambda_2, \lambda_2'\}$. λ_1 denote the producing of a unit by the source and the starting of its serving λ_1' – the producing only, λ_2 the end of serving and starting a new one, and λ_2' the end of serving only. The scheme of the process of serving may be presented as follows:

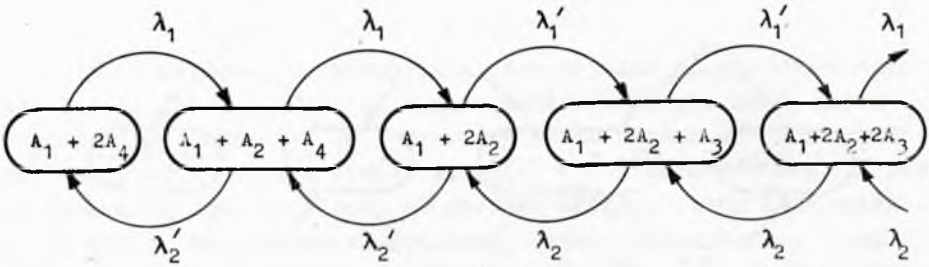


Fig. 2.

2. THE HISTORY OF THE PROCESS

In a general case the future run of the process may depend on the previous one, i.e. on the history of the process. It consists

of the information about all changes concerning the states of the system, those of the population itself, as well as of the moments at which the changes occurred. At this point we shall limit ourselves to the description of the order and the kind of changes only . We shall permanently assume that at the beginning the system is in the state $s_0 \in S$.

Every individual a appearing in the run of the process will have an index - a pair of numbers and will be identified with this assigned index. The first number of the index, further denoted by $i(a)$, is one of the numbers $1, \ldots, m$, and it characterizes the type of the individual. The second number $j(a)$ determines which individual of the given type in the considered run is a. Therefore, if after the individual a appears in the process run the first individual b of the same type as a, then $j(b) = j(a) + 1$. If at the same time several individuals of the same type appears then they are arbitrarily numbered in turn by numbers $j(a) + 1$, $j(a) + 2$, ...

For $i = 1, \ldots, m$ let n_i be the number of this individual of the type i, which appeared last. Let $\bar{n} = (n_1, \ldots, n_m)$. Then

$$H(\bar{n}, \lambda) = \left\{(1, n_1+1), \ldots, (1, n_1+\beta_1(\lambda)), \ldots, (m, n_m+1), \ldots, (m, n_m+\beta_m(\lambda))\right\}$$

is the set of individuals that appear during this change from the actual state s to $s_1 \in \Gamma(s)$, for which $\lambda = \Pi(s, s_1)$. In view of /III/ for a fixed s and $i = 1, \ldots, m$ there exists only one pass for which $\lambda \in \Lambda_1$. Thus, the pair (s, i) determines λ, i.e. $\lambda = \lambda(s, i)$. To various passes correspond various s, but in view of /V/ all β_j depend only on the act λ. Directly after this act, every coordinate of vector $\bar{n}$ increases by an appropriate $\beta_j(\lambda)$. It is not difficult to notice that $\beta_j(\lambda)$ is the number of individuals of the type j in the set $H(\bar{n}, \lambda)$. But the entire number $|H(\bar{n}, \lambda)|$ of elements of this set is

$$\sum_{j=1}^{m} \beta_j(\lambda) .$$

All these numbers do not depend on $\bar{n}$.

Further we shall use the following denotations

$$\bar{\alpha}(s) = (\alpha_1(s), \ldots, \alpha_m(s))$$

$$\bar{\beta}(\lambda) = (\beta_1(\lambda), \ldots, \beta_m(\lambda)) .$$

In the system being initially in the state s_0 - live $\alpha_1(s_0)$ individuals of the type 1, ..., $\alpha_m(s_0)$ individuals of the type m. We shall assume that they are individuals $(1,1)$, ..., $(1, \alpha_1(s_0))$, ..., $(m, 1)$, ..., $(m,\alpha_m(s_0))$. Let $I(o)$ denote set of these individuals, i.e. the initial population, and let $\bar{n}(o) = \bar{\alpha}(s_0)$. The only changes that occur are the result of the individuals' death. Let a_k be the very individual who perishes as the k-th in turn, and let s_k be the state to which then the system passes. If $I(k)$ denotes the population composed of individuals living directly after this and $\bar{n}(k)$ the vector of the greatest numbers of individuals of all types that have lived or are living, then the following recurrence relations take place.

$$s_k \in \Gamma(s_{k-1}) \qquad /1/$$

$$a_k \in I(k-1) \qquad /2/$$

$$\pi(s_{k-1}, s_k) = \lambda(s_{k-1}, i(a_k)) \qquad /3/$$

$$I(k) = (I(k-1) - \{a_k\}) \cup H(\bar{n}(k-1), \lambda(s_{k-1}, i(a_k))) \qquad /4/$$

$$\bar{n}(k) = \bar{n}(k-1) + \bar{\beta}(\lambda(s_{k-1}, i(a_k))) \qquad /5/$$

under the initial conditions

$$I(o) = \{(1,1), \ldots, (1, \alpha_1(s_0)), \ldots, (m,1), \ldots, \qquad /6/$$

$$(m, \alpha_m(s_0))\}$$

$$\bar{n}(o) = \bar{\alpha}(s_0). \qquad /7/$$

These relations result directly from properties /I/ - /V/ of the process. They permit to reproduce its history with the given sequence $a_1, a_2, \ldots, a_k$. Indeed, a_1 determines s_1 in view of the property /III/ and /3/. But the population $I(1)$ is determined according to /3/ and /4/. Generally, if the state s_{k-1}, population $I(k-1)$ and $\bar{n}(k-1)$, are given, then a_k determines s_k, $I(k)$, $\bar{n}(k)$. In virtue of the above one may identify the history of the process with the sequence $q = a, \ldots, a_k$ which satisfies conditions /1/ - /7/. Further on we shall do it. Q will denote the set of all such histories together with the "empty" 0 history, i.e. with the sequence not including any element. A will denote the set of all individuals which might appear in histories belonging to Q.

In the set Q the relation of successiveness can be introduced. We shall say that the history q' follows after q and we shall write $q \rightarrow q'$ if $q' = qa$ for a certain $a \in A$. The relation of successiveness permits to introduce a partial order to Q. We shall say that q precedes q', or is the segment of q, if there exists a sequence of the histories $q = q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q_k = q'$. The record $q < q'$ will signify that q precedes q', and $q \leqslant q'$ that $q < q'$ or $q = q'$. $f(q)$ will denote the last element of q and $g(q)$ will denote the segment of q having, as the last, such an element of q after the death of which appears $f(q)$. Especially $g(q) = 0$ if $f(q)$ appears in the initial population. The segment $g(q)$ will be called the generator of the history q.

Every history $a_1, \ldots, a_k$ is the initial segment of the infinite sequence $a_1, \ldots, a_k\, a_{k+1}, \ldots$ satisfying relations /1/ - /7/. Such sequences will be called the process run.

Example 3

Let us consider the system from Example 1, p. 1, the states of which are:

$$s^{(0)} = 3A_1 + 2A_4$$

$$s^{(1)} = 2A_1 + A_2 + A_4$$

$$s^{(2)} = A_1 + 2A_2$$

$$s^{(3)} = 2A_2 + A_3$$

$s^{(o)}$ being initial. Then

$$I(o) = \{(1,1),\ (1,2),\ (1,3)\}$$

$$\bar{n}(o) = (3,0).$$

Sequence $r = (1,1)\ (2,1)\ (1,4)\ (2,2)\ (1,5)\ \ldots\ (2,k)\ (1,k+3)\ldots$ is the process run, and each of its initial segment is a history. Indeed. Let us consider the sequence of states $s^{(o)}, s^{(1)}, s^{(o)}, s^{(1)}, s^{(o)}, s^{(1)}, \ldots$. Of course, $s_k \in \Gamma(s_{k-1})$. It is easy to see that sets

$$I(1) = \{(1,2),\ (1,3),\ (2,1)\}$$

$$I(2) = \{(1,2),\ (1,3),\ (1,4)\}$$

$$\ldots$$

$$I(2k+1) = \{(1,2),\ (1,3),\ (2,\ k+1)\}$$

$$I(2k+2) = \{(1,2),\ (1,3),\ (1,\ 3+k+1)\}$$

$$\ldots$$

are successive populations of the run. For $k = 0,1, \ldots$

$$\bar{n}(2k+1) = (3+k,\ k+1)$$

$$\bar{n}(2k+2) = (3+k+1,\ k+1).$$

thus

$$I(2k+1) = (I(2k) - \{1,3+k\}) \cup \{(2,\ k+1)\}$$

$$I(2k+2) = (I(2k+1) - \{2,k+1\}) \cup \{(1,3+k+1)\}$$

with

$$\{(2,k+1)\} = H((3+k,k),\ \lambda(s_{2k},\ 1))$$

$$\{(1,3+k+1)\} = H((3+k,\ k+1),\ \lambda(s_{2k+1},\ 2)),$$

and

$$\lambda(s_{2k},\ 1) = \pi(s_{2k},\ s_{2k+1}) = \lambda_1$$

$$\lambda(s_{2k+1},\ 2) = \pi(s_{2k+1},\ s_{2k+2}) = \lambda_2' .$$

The property of the run is that the individuals (1, 2) and (1, 3) belong to all populations I(k), i.e. - they do not perish. It also results from the scheme property that the process has no finite runs, and the set of individuals which may appear /i.e. A/ is identical with the set of all pairs of the form (1,k), (2,k), where k = 1,2, ...

3. PROCESS REALIZATIONS

Let $\{\xi_a\}_{a \in A}$ be the family of the lifetimes of all individuals who may appear in the process run. Assume that these times are arbitrary random variables with not negative real values determined on the probability space $(\Omega, \mathcal{F}, P)$. We accept that the process always starts at the moment $t = 0$.

For the set $B \subset A$ and for numbers $l_a \geq 0 (a \in B)$, we determine $\mathrm{ind}(B, l_a)$ as the element from B, for which l_a is the smallest. If there are several such elements then $\mathrm{ind}(B, l_a)$ should have the smallest i(a), and in the area of these, the smallest j(a). From the definition we have for all $a \in B$

$$l_{\mathrm{ind}(B,\ l_a)} \leq l_a .$$

To every $\omega \in \Omega$ the run $r(\omega)$ can be uniquely ascribed. For this purpose we determine its succeeding elements $a_1(\omega), a_2(\omega), \ldots$ and numbers $\tau_a(\omega)$ for individuals of the population of the run $r(\omega)$ as follows:

if $a \in I(o)$ then $\tau_a(\omega) = \xi_a(\omega)$,

$a_1(\omega) = \mathrm{ind}\left(I(o), \tau_a(\omega)\right)$,

if $a_{k-1}(\omega)$, s_{k-1}, $I(k-1)$, $\bar{n}(k-1)$ and $\tau_a(\omega)$

for $a \in I(k-1)$ are determined, then ,

$$a_k(\omega) = \mathrm{ind}\left(I(k-1), \tau_a(\omega)\right),$$

$$s_k \in \Gamma(s_{k-1}),$$

$$\pi(s_{k-1}, s_k) = \lambda(s_{k-1}, i(a_k(\omega))),$$

$$I(k) = \left(I(k-1) - \left\{a_k(\omega)\right\}\right) \cup H\left(\bar{n}(k-1), \lambda\left(s_{k-1}, i(a_k(\omega))\right)\right),$$

$$\bar{n}(k) = \bar{n}(k-1) + \bar{\beta}\left(\lambda(s_{k-1}, i(a_k(\omega)))\right),$$

$$\tau_a(\omega) = \tau_{a_k(\omega)}(\omega) + \xi_a(\omega) \quad \text{for} \quad a \in H(\bar{n}(k-1), \lambda(s_{k-1}, i(a_k(\omega)))) .$$

It should be noticed that the states, population and $\bar{n}$ also depend on ω. It is easy to see that the number $\tau_a(\omega)$ is the moment at which the death of the individual a occurs, and $\tau_{a_k(\omega)}(\omega)$ is the moment of its appearance.

Let R^+ be the set of all not negative real numbers. We extend the history set by adding the element χ, called improper history. Q will further denote the extended set. Physical sense of the notion of improper history will be cleared up below. Now, we accept that every history is the segment of χ.

To every pair $(\omega, t) \in \Omega \times R^+$ one can unequally ascribe the history $q(\omega, t) \in Q$. Namely, we put:

$$q(\omega, t) = \begin{cases} 0 & \text{if } t < \tau_{a_1(\omega)}(\omega), \\ a_1(\omega) \dots a_k(\omega), & \text{if } \tau_{a_k(\omega)}(\omega) \leq t < \tau_{a_{k+1}(\omega)}(\omega) \\ \chi & \text{if } \tau_{a_k(\omega)}(\omega) \leq t \text{ for } k = 1,2, \dots \end{cases} \quad /8/$$

For a fixed ω we thus get the funotion $q(\omega,\cdot)$ of the variable t. We shall oall it the realization of the process. In turn, with a fixed t, $q(\cdot,t)$ is the funotion of ω. In view of the oountability of the set Q all histories oan be numbered and treated as natural numbers. Then, $q(\cdot,t)$ oan be treated as a real funotion. If it is $\mathcal{F}$ - measurable, it is a random variable. Then the family $\{q(\cdot,t)\}_{t \in R^+}$ is a stoohastio prooess whioh is going to be proved. For this purpose it is suffioient to prove the following.

Theorem

For every $t \in R^+$ and $q \in Q$ the set $\{\omega \in \Omega : q(\omega, t) = q\}$ is $\mathcal{F}$ - measurable.

Proof

First assume that $0 \neq q \neq \chi$. Then $q = a_1, \dots, a_k$. For every history $q \in Q$, $0 \neq q \neq \chi$ we determine the set $U(q)$ as follows:

a/ if $g(q) = 0$, then $U(q) = \{f(q)\}$,

b/ if $g(q) \neq 0$, then $U(q) = \{a_{i_1}, \dots, a_{i_1}\}$, where

$$a_{i_1} = f(q_2), \dots, a_{i_j} = f(q_{j+1}), \dots, a_{i_1} = f(q), \text{ and}$$

$$q_1 = g(q), \dots, q_i = g(q_{i+1}), \dots, 0 = g(q_1).$$

Its elements are individuals $a_{i_1}, \ldots, a_{i_l}$, $\left(a_{i_1} = f(q)\right)$ such that at the moment of death of the current one, starts the life of the next one. Therefore, if we determine $\tau_{(q)} = \tau_{f(q)}$, then

$$\tau(q) = \sum_{a \in U(q)} \xi_a \,. \qquad /9/$$

Let $[q]$ be the set of $\omega \in \Omega$ to which corresponds the history $q(\omega, t) = q$ for a certain $t \in R^+$. It is $\mathcal{F}$ - measurable. Indeed

$$[a_1] = \Big\{\omega \in \Omega : \xi_a(\omega) < \xi_{a_1}(\omega) \quad \text{or} \quad \xi_a(\omega) = \xi_{a_1}(\omega), \text{and} \qquad /10/$$

$$i(a) > i(a_1) \quad \text{or} \quad i(a) = i(a_1), \quad \text{and} \quad j(a) > j(a_1)$$

$$\text{for} \quad a \in I(o)\Big\}$$

is an $\mathcal{F}$ - measurable set. But if $[a_1, \ldots, a_{k-1}]$ is $\mathcal{F}$ - measurable, then

$$[a_1 \ldots a_k] = [a_1 \ldots a_{k-1}] \cap \Big\{\omega \in \Omega : \tau(a_1 \ldots a_{k-1} a_k) < \tau(a_1 \ldots a_{k-1} a),$$

$$\text{or} \quad \tau(a_1 \ldots a_{k-1} a_k) = \tau(a_1 \ldots a_{k-1} a) \quad \text{and} \quad i(a) > i(a_k) \qquad /11/$$

$$\text{or} \quad i(a) = i(a_k) \quad \text{and} \quad j(a) > j(a_k) \quad \text{for} \quad a \in I(k-1) - \{a_k\}\Big\}$$

is also $\mathcal{F}$ - measurable, as $I(k-1) - \{a_k\}$ is finite, and functions $\tau(a_1 \ldots a_{k-1} a_k)$, $\tau(a_1 \ldots a_{k-1} a)$ are $\mathcal{F}$ - measurable in view of /9/.

$\mathcal{F}$ - measurability of the set $\{\omega \in \Omega : q(\omega, t) = a_1 \ldots a_k\}$ results from the following equality

$$\{\omega \in \Omega : q(\omega, t) = a_1 \dots a_k\}$$

$$= [a_1 \dots a_k] \cap \Big(\bigcap_{a \in I(k)} \{\omega \in \Omega : \tau(a_1 \dots a_k) \leq t < \tau(a_1 \dots a_k a)\} \Big). \qquad /12/$$

The truth of the theorem for $q = 0$ results from the fact that the set

$$\{\omega \in \Omega : q(\omega, t) = 0\} = \{\omega \in \Omega : \xi_a(\omega) > t \quad \text{for} \quad a \in I(0)\} \qquad /13/$$

is $\mathfrak{F}$ - measurable.

If $q = \chi$, then

$$\{\omega \in \Omega : q(\omega, t) = \chi\} = \bigcup_{\substack{q \in Q \\ 0 \neq q \neq \chi}} [q] \cap \{\omega \in \Omega : \tau(q) \leq t\}, \qquad /14/$$

therefore the theorem holds too QED.

Usually the stochastic birth and death process is defined in a different way than we did here. Namely, let $|B|_i$ denote the number of individuals of the type i in the set $B \subset A$. Then, the vector $(|B|_1, \dots, |B|_m)$ represents the amount of individuals of all types in the set B. As a birth and death process one understands usually the stochastic process $\{\bar{N}(\cdot, t)\}_{t \in R^+}$ where

$$\bar{N}(\omega, t) = \big(|I(k)|_1, \dots, |I(k)|_m\big),$$

and $I(k)$ is the population corresponding to the history $q(\omega, t) = a_1 \dots a_k$. For $q(\omega, t) = \chi$ one may put e.g. $\bar{N}(\omega, t) = (\infty, \dots, \infty)$

Having the process $\{q(\cdot, t)\}_{t \in R^+}$ one can determine $\{\bar{N}(\cdot, t)\}_{t \in R^+}$. However, the consideration of the first one is advantageous because the first contains the description of the fates of separate individuals.

4. ALGORITHM OF SIMULATION

Assume the possibility of chosing at random the elements of the set Ω, according to the probability measure P, determined on σ-field $\mathcal{F}$ of subsets of Ω. Then, the process $\{q(\cdot,t)\}_{t\in R^+}$ can be simulated as follows.

We chose at random $\omega \in \Omega$. Then we determine the lifetime of all individuals of the initial population $I(o)$. These are the values of random variables ξ_a $a \in I(o)$ corresponding to the chosen $\omega \in \Omega$, i.e. the numbers $\xi_a(\omega)$. We determine the moments of death $\tau_a(\omega) = \xi_a(\omega)$ $(a \in I(o))$. We choose the individual $a_1 = \mathrm{ind}(I(0), \tau_a(\omega))$, we determine its type $i(a_1)$ and put $q(\omega,t) = 0$ for $t < \tau_{a_1}(\omega)$. Having s_0 and $i(a_1)$ we determine $\lambda(s_0, i(a_1))$ and $s_1 \in \Gamma(s_0)$ such that $\pi(s_0, s_1) = \lambda(s_0, i(a_1))$. If it appears that such s_1 does not exist, we put $q(\omega, t) = a_1$ for $t \geq \tau_{a_1}(\omega)$ and we finish the simulation. In an opposite case s_1 can be determined uniquely. This results from the process property. We determine $\bar{n}(1)$ and the set $H(\bar{n}(o), \lambda(s_0, i(a_1)))$, determine the life times $\xi_a(\omega)$ for its elements and next the moments of death $\tau_a(\omega) = \tau_{a_1}(\omega) + \xi_a(\omega)$. We eliminate the individual a_1 from the population $I(o)$, and add to it all individuals from the set $H(\bar{n}(o), \lambda(s_0, i(a_1)))$. In such a way we obtain the population $I(1)$. We chose from among it the individual $a_2 = \mathrm{ind}(I(1), \tau_a(\omega))$, we put $q(\omega, t) = a_1\, a_2$ for $\tau_{a_1}(\omega) \leq t < \tau_{a_2}(\omega)$ and we proceed like in the case of the initial population $I(o)$. Continuing this procedure, we determine the realization $q(\omega,\cdot)$ of the process on a still bigger segment of time.

The described proceeding is just the simulation of the birth and death process. It is not difficult to imagine the ways of its realization on computers.

References

[1] M. LOÈVE: Probability Theory. Princeton 1960.
[2] T.E. HARRIS: The Theory of Branching Processes. Berlin 1963.

THEORY OF PROGRAMMING

Instytut Maszyn Matematycznych
Algorytmy N° 7

681.3.06:691.327.2

STORAGE ALLOCATION FOR ALGOL

by Ludwik CZAJA
Piotr SZORC

Received September 30th, 1966

The paper deals with a method of storage allocation for ALGOL implemented on the ZAM computer. Storage allocation is carried out automatically by the ALGOL compiler. It is a so-called ''static - dynamic'' allocation since it is carried out partly during the translation and partly during the run time.

1. INTRODUCTION

Some basic storage allocation problems arise because of the existence of several kinds of storages in the computer, the capacity and access time of which are different. Generally speaking such an allocation in storages of a program and its variables which ensures the shortest run time should be introduced. This can be done by organizing a moderate number of storage transfers and also keeping the most frequently used information in high-speed store. The problem formed in the above way can be solved only for every computer separately, a more general solution being unknown because of its strong dependence on computer characteristics.

It appears that the method of compiling object programs depends to a great extent on storage allocation. The system of executing programs translated from ALGOL /as described in [6] /which enriches the "pure" ZAM computer and provides facilities to construct such programs, is the consequence of the described method

of storage allocation. In particular this method requires a separate facility of entry and exit from blocks and procedures.

To avoid complicating the description, the existence of so-called external array will not be considered, i.e. we assume all the variables to be allocated in the high-speed store. We also assume that there are only two stores: high-speed store and the external one. The method of allocating information in the computer storage will be called the storage allocation.

2. GRAPHIC REPRESENTATION OF THE ALLOCATION FOR TWO STORAGES

It is convenient to present the storage allocation in a rectangular set of coordinates. The x-axis stands for the external store and the y-axis stands for the high-speed store. The point on the axis indicates the address. The fact that the storage allocation is discrete is of no importance for the allocation problems. Transfers from the external store to the high-speed store can be treated as projections of a fragment of the external store on to the fragment of the high-speed store. It is, therefore, a line on the plane x, y. We are naturally interested only in segments forming a 45° angle with the axis. It is particularly easy to draw a border line between the high-speed store area reserved for variables and the area reserved for the program; the border line is usually varying in time. For instance, for the program having the block structure ((( )( ))(( ))( )) and stored in the external store the border line can be presented as follows:

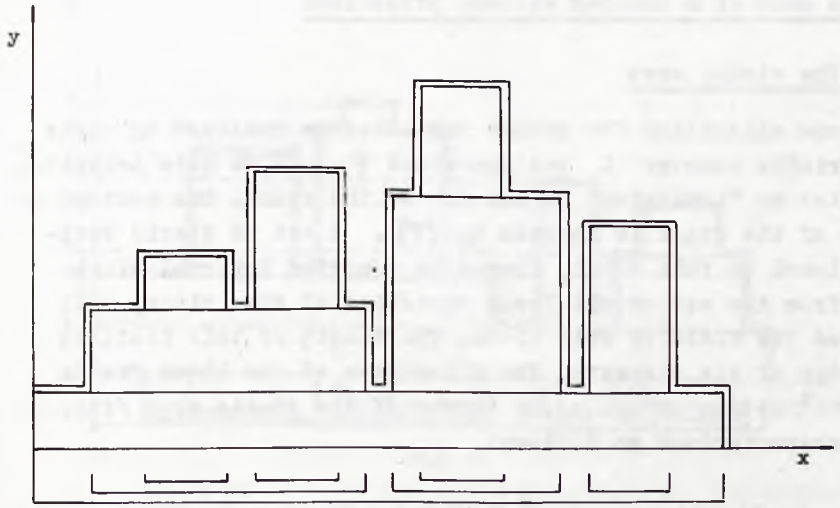


Fig. 1.

The area under the double line is for variables /which are addressed acoording to the mutual localization of the blocks/, while the area above the double line is for the program. The border line treated as a funotion of x /more precisely - its horizontal segments only/ represents the number of variables available from x, and is called the level of variables in x.

3. ALLOCATION OF VARIABLES

Among several known allocations mentioned in [7] the one called "statio-dynamic" variable allocation is chosen. It consists in place allocation in high-speed store of simple variables during the compiling time, while to arrays and recursive procedures fields the place is allocated during the program execution. To all variables real addresses are assigned in the high-speed store, the mutual localization of the blocks being considered. All the above being done in the ALGOL-optimum way.

3.1. The case of a program without procedures

3.1.1. The static area

Storage allocation for simple variables is realized by means of a variable counter L and the stack φ. To be more accurate, φ denotes an "indicator" or the top of the stack. The content of the top of the stack is denoted by (φ). A set of static variables* local to this block, formed by removing internal block fields from the set of all local variables of this block, will be called the field of this block. The length of this field is the number of its elements. The allocation of the block fields i.e. construction of the upper border of the static area /fig. 2/ can be characterized as follows:

entry to the block: $\varphi := \varphi + 1$,
$L := L +$ length of the block field,
$(\varphi) := L$;

exit from the block $\varphi := \varphi - 1$,
$L := (\varphi)$.

It is obvious that this algorithm realizes the ALGOL locality rule in the optimum way i.e. for each x the level of static variables in x, constructed by this algorithm, is the smallest of all the acceptable ones /i.e. those ensuring the correct program execution/.

It is also obvious that the level of static variables is the same in each place of the given simple block** called the block level.

E x a m p l e

The program having the following block structure $(((\,)((\,))(\,)((\,)))$ lies on the drum and has the following levels of the static variables of the successive blocks: S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8. The allocation of its static variables is presented in the graphic form:

* The static variable is a simple variable or an auxiliary variable introduced by the compiler /cf [6]/.

** The simple block is built up by removing all the internal blocks.

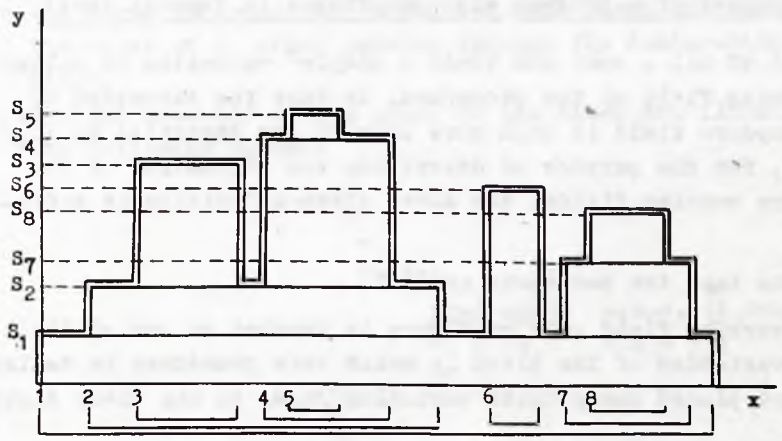


Fig. 2.

3.1.2. Dynamical area /for arrays/

The dynamical area is limited by two bounds and its structure is the structure of these bounds.

The lower bound is defined during the translation time and is constant during the run time; it is the highest level of static variables.

The upper bound is defined during the run time in the way similar to the one described in section 3.1.1., except that the actual values of L /which is now called the L register/ i.e. the actual level of dynamic variables are taken down for each block not in the stack φ but in a special cell /the auxiliary variable/ lying within the block. The detailed description can be found in [6]. Definition of the upper bound during the translation process is not possible, because of the existence of dynamic arrays.

Above the upper bound of the dynamic area there is the area designed for the program.

3.2. The case of a program with procedures /a general case/

A set of all normal and local - simple* variables is called the working field of the procedure. In fact the structure of the procedure field is much more complex /as described in [6]/, however, for the purpose of describing the allocation of the procedure working fields, the above given definition is sufficient.

Let us take the two basic rules:

a. the working field of a procedure is treated as one of the local variables of the block in which this procedure is declared and placed among these variables /i.e. in the block field/,

b. if it is possible /on account of the locality rule/ to call for procedure G from procedure F and if the procedures are not declared one within the other, then, their fields are disjointed.

While the rule b is connected with the correct program execution, the realization of the rule a is convenient as it gives a clear picture of the static area structure and the optimum use of the storage, however, it is not the only one acceptable. When treating the entire procedure field as one variable, our considerations become much simpler.

The allocation of simple variables in a general case is realized by means of two parameters L and M, the stack φ and the two translator states'being in the procedure' and 'not being in the procedure'. The parameter M denotes the maximum level of static variables in the procedure. L and φ have the previously given meaning.

The upper bound of the static area in a general case will be formed in the following way:

* As distinguished from the block field, the working field of a procedure consists of all the block fields which it contains as well as of all the working fields of the internal procedures.

B – the local variable introduced for each block and containing the value of L after passing through the declaration of this block.
Actions executed at the entry to the block are illustrated by the following example

```
                -
                -
                -
          begin
                real array  a,b,c, [1 :n, 0:m ];
                M, N [k + 2 : n - 1];
                -
                -
                -
```

1. Evaluating the dope vector for the first array of the segment; it should be noticed that the number of dope vectors is equal to the number of array segments but not to that of the arrays.

```
DV1 [1 ] := 2
DV1 [2 ] := n-1+1
DV1 [3 ] := m-0+1
DV1 [4 ] := 1xDV1 [3] +0 .
```

2. Reservation of location determination of the array addresses and storing the pairs /the dope vector address, the array address/

```
a := (< DV > ,  L )
L := L+DV1 [2] xDV1 [3]
b := (< DV1 > ,  L )
L := L+DV1 [2] xDV1 [3]
c := (< DV1 > ,  L )
L := L+DV1 [2] xDV1 [3] ,
```

and then in an analogous way:

```
DV2 [1]  := 1
DV2 [2]  := n-1- ( k+2 ) +1
DV2 [3]  := k+2
```

M := (< DV2 >, L)
L := L+DV2 [2]
N := (< DV2 >, L)
L := L+DV2 [2] .

3. After performing all these activities for all array declarations of a given block, we store the level of the dynamic storage inside this block, i.e. the assignment B := L is executed.

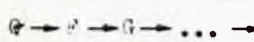
Exit from the block through end or through go to a non-local label must regenerate the register L to the state in which it was at the moment of passing through the block declaration. To do that it is necessary only to perform the formula L := B, where B is always local in the block which was entered. Thus, we place such a formula after end of each block and after each label.

Due to the above-described mechanism of block exit and entry, the level of dynamic store can be regenerated while returning to the given level of the program block structure. This dynamically realizes the system similarily to the described for the static storage [4].

II. IMPLEMENTATION OF THE PROGRAM WITH RECURSIVE PROCEDURES AND SWITCHES

The system of allocation as described in paper [4] permits us to realize programs having no recursive procedures and switches, without any additional mechanisms. It results from the fact that fields of each of the two procedures or switches which can work simultaneously, are disjointed. To realize full ALGOL /with recursive procedures and switches/ a system of programs, so-called "running system" was written. We usually refer to it by extra-codes when using procedures. The system works in the following way.

Definitions

Let us consider the dynamic sequences of steps

which begin with the main program and end

with the actually executed procedure. These sequences are formed according to the following rules:

a. each entry to the procedure causes adding its name to the dynamic sequence of steps,

b. each end of the procedure execution through end or go to leading outside the procedure, causes eliminating the end of the sequence up to the name of the procedure to which we pass, if passing through end only the last element is eliminated; if through go to several elements of the sequence may be eliminated.

The procedure will be called:

1. activated - if its name occurs in DSP /dynamic sequence of steps/,
2. recursive - if earlier elements of the same name occur in DSP,
3. active - if no later elements of the same name occur in DSP.

We will also use the notions non-activated, non-recursive, non-active procedures, which are negations of the original concepts.

The element directly preceding the procedure in DSP is called dynamic predecessor of the procedure.

These definitions make it obvious if the same procedure name occurs many times in DSP, it is treated as several various procedures.

The structure of the procedure vector and of the procedure field

According to the first part of this paper, a vector of procedure and of switches /P.S.V./ is being formed during the translation. This vector contains the following information for each declared procedure and switch:

1. the address of the field,
2. the address of the program /procedure body/,
3. the size of the field,

4. the number of formal parameters of the procedure or the number of elements of the switch list.
5. Activation /this is a dynamic two state information, the initial state for each procedure is non-activated/.

One group of this information takes two words in the ZAM-21 computer.

The field of the procedure consists of:

1. Cells of procedure and running system cooperation:

   E - the address of the field of the recursive predecessor,
   D - the dynamic shift of the field,
   N - the name of the dynamic predecessor,
   R - the address of the field of actual parameters,
   T - the drum address of return.

2. The fields of formal parameters of the procedure.
3. The fields of local quantities of the procedure body.

The cells of the procedure and running system cooperation, as well as the field of formal parameters of the procedure, are loaded at the entry to the procedure.

Entry and exit from the procedures

There are two registers which are of importance during the run of the object program, namely:

1. Register L - the actual level of the dynamic storage.
2. Register P - the name of the procedure being executed, and represented by VP /n/, where VP /n/ denotes the address of this procedure in the procedure vector. The register P has the initial value 0 /the name of the main program/.

Before entering the procedure it must be made sure whether it is a procedure of the language /e.g. INPUT or SINUS/, or a procedure declared in the program. In the case of a standard procedure, we communicate with it by means of the program evaluating its actual parameters and storing the results in a special buffer. The

program then goes to execute the proper language procedure without checking recursiveness, since language procedures are not recursive.

Entering the ALGOL procedure, we check whether it is activited according to the information contained in the procedure vector. If it is not activated, we move its fields to the free place in the dynamic storage according to L, thus releasing the field for the procedure which is to be executed.

At the same time, in the N-th cell of the procedure we store the name of the procedure which we enter to. Owing to that, it is possible, on exit from the procedure, the name of which is in the register P /i.e. the procedure being executed/, to reproduce the sequence reverse to DSP, leading from the content of P to 0 through successively indicated dynamic predecessors.

The second chain formed for the exit from the procedure is the chain of recursive predecessors stored in the cells E of particular procedures. Entering a non-activated procedure, we write into this procedure E the own procedure address, while entering an activated one, we write the actual value of L, i.e. the address of the location of the recursive predecessor. Let us notice that:

1. The content of the cells E and N for non-activated procedures /i.e. not belonging to DSP/ is inessential.
2. Since at no time of the program run there can be two identical values of the cells E, the content of the cell E of the procedure can be considered as the dynamical name of this procedure, distinguishing it from all other elements of the sequence BSP. The exit from the procedure through <u>end</u> reproduces the state from before activating the procedure, therefore the name of the dynamic predecessor is written into P. And depending on whether we go out from the recursive procedure or not, we remove the field of its recursive predecessor or make it non-activated. After completing all these activities, we execute the return jump instruction. The return address which is sent to the procedure while it is being called for, is stored in its field in the cell T.

Exit from the procedure through go to is slightly more complicated as go to can pass through several elements of DSP /in case of go to to the formal label/ the DSP elements can be the same.

Therefore, the labels are represented by the address and by the dynamic name of the procedure to which this label belongs. With this kind of procedure representation it is enough to act as in the case of end /obviously except the return jump/ until the required procedure appears, and then to jump to the indicated address. The lowering of the level of the dynamic storage, which is generally necessary after the exit from the procedure, is realized by the mechanism of assignments L := B, as described in the first part of this paper. The assignments are made next to the labels.

Evaluation and substitution of actual arguments in the procedure

Because of the substituting of expressions by name and because of side-effects, it is necessary in ALGOL to enclose actual parameters of procedures in subroutines and send the subroutine addresses to the procedure; the moment of their execution is determined by the procedure itself. As Input-Output procedures have no-specified formal parameters, it was necessary to transmit information on the kind of the substituted parameter.

The following were then distinguished as being of various kinds: a simple variable, an indexed variable, expressions /arithmetic or logic/, an array, a designational expression and a string. Distinguishing between arithmetic expressions and variables is made for time - optimalization of the object program, as in the case of calling by name, it is enough to call once only for the subroutine which substitutes the simple variable, while in the case of the subroutine substituting expression we call for the subroutine on each passing through the corresponding formal variable.

At the exit from the procedure, besides the above described cells E and N, the procedure cells D and R are used to evaluate the actual argument subroutine.

D - /dynamic shift of field/ contains zero if the field of this procedure is in the static storage as a result of the previously described operations. In the opposite case D contains its shift /the difference between the dybamic address and the statio one of the field of this procedure/.

If x is a local variable of the procedure F /active or non-active/, then by adding the content of cell D of procedure F to the statio address x, one obtains its dynamio address, whioh is or is not equal to the statio one. This is essential when substitutiong suoh variables where just the address /especially the dynamio one/ and not the value should be substituted, since the formal parameter oan be the left side of the assignment statement.

The actual parameter programs should be treated as subroutines: one has to get baok from them. It is therefore neoessary to store the return labels /i.e. the drum address and the prooedure name/ in their working field. Since it is done by the programmed instruotion calling for the argument, the address of this working field has to be sent to the called prooedure and stored in a fixed looation in its field and this looation is denoted by R. Thus, R oontains the address of the working field of the aotual parameter programs substituted in this prooedure. This is the address from the statio field of the dynamio prodeoessor of this prooedure. The oan be proved that during the exeoution of the subroutine of the aotual argument of the prooedure DSP should be the same as before the entry to. Obviously, after oalling for the argument it must be possible to reproduoe the situation whioh distinguishes suoh a transfer from an ordinary ending of the prooedure exeoution and from the eliminating of the element DSP.

Let F→G be the last element of DSP, /usually it is not important whether F is G and whether G is F/. In the oase of oalling for the argument from G to F only two possibilities oan be ooneidered:

1. G non-reoursive:

a. let us write the return label into G in the field indicated in R,

b. execute all exit activities of G i.e. activating it and placing F in register P.

2. G - recursive:

a. let us store the values R and N of field G,
b. exchange field G with its recursive predecessor,
c. write the return label into G in the field indicated in the stored value R,
d. place the stored value N /it being F/ in the register P.

The return from the actual argument subroutine reproduces the state from before calling for the argument, changing again the register P and exchanging the fields in the case of return to the recursive procedure. The content of cell D determines wheter we go back to the recursive procedure; if it differs from 0 it means the return to the recursive procedure. Only the exchange of fields when calling for the argument could cause the existence of the active procedure with D different from 0.

Final notes

The above-described system realizing procedures in ALGOL has some drawbacks, of which the main is its logical complexity particularly in evaluating the actual arguments of recursive procedures.

A fixed reservation of place for working fields of procedures is also an important drawback of the system. The advantage of the system is a greater speed of the execution of object programs having no recursive procedures.

Another advantage of the accepted system is also an easy transfer to simplifield and considerably faster system which realizes ALGOL without recursive procedures, even by means of optimising comments informing about the non-recursiveness of the program.

The final estimation of the described system might be made after its longer exploitation and after the comparison of times of program execution in this and other systems of implementation of ALGOL for the same computer.

III. FLOWCHARTS OF THE RUNNING SYSTEM PROGRAM

The RUNNING SYSTEM program consists of several separate subroutines which enrich the hardware of the ZAM computer. Thus a new machine is obtained, the instruction list od which contains several instructions of the ZAM hardware as well as several instructions of the extra-code.

The flowcharts given in this section, define the extra-code instructions. The ALGOL compiler for ZAM produces the object program written in the instructions of the enriched machine.

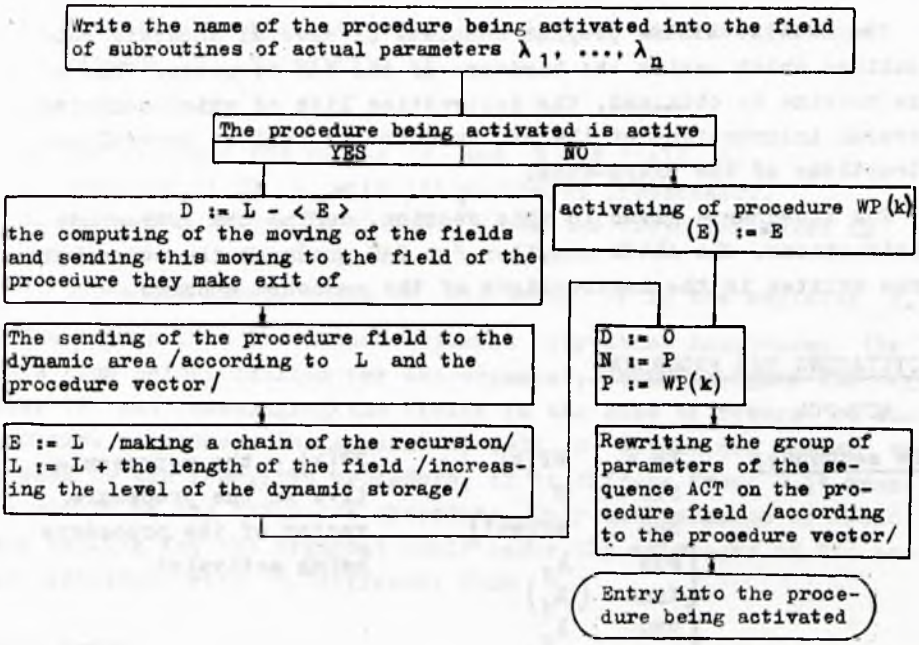
ACTIVATING THE PROCEDURE

ACT-P04

The sequence:

Po 4	WP(k)	WP(k) - the representative in the procedure vector of the procedure being activated
STS	R	
SEG	+2(n+1)	
STS	λ_1	
Kind	(λ_1)	
STS	λ_2	
Kind	(λ_2)	
STS	λ_n	
Kind	(λ_n)	

The subroutines:

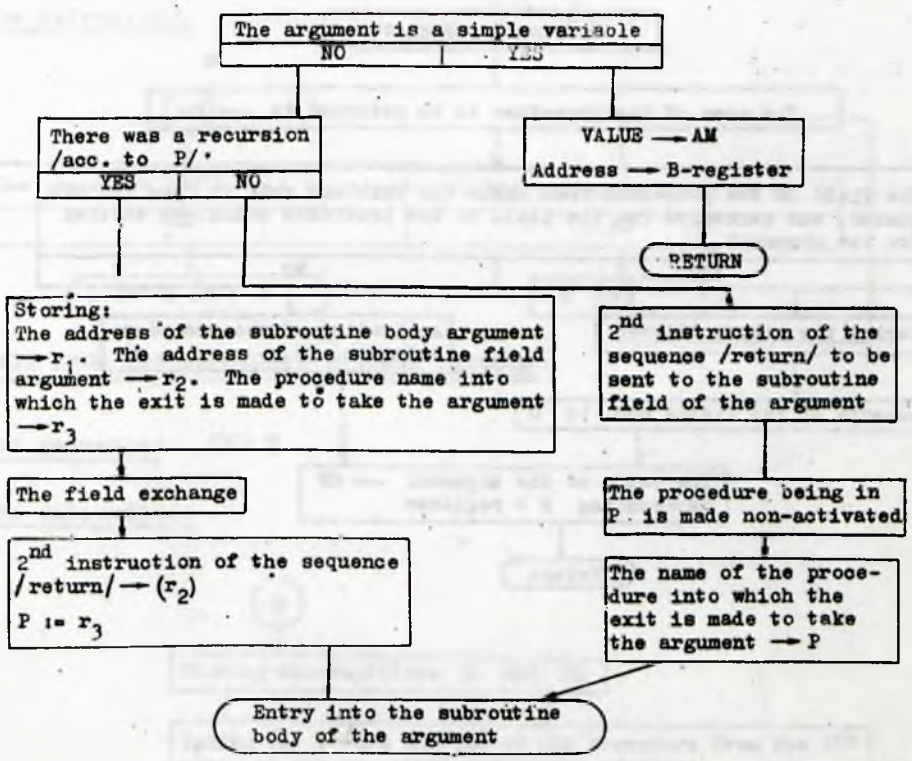


GOING FOR THE ARGUMENT

ARG – P06

The sequence: P06 α
SEG + 1

The subroutine:

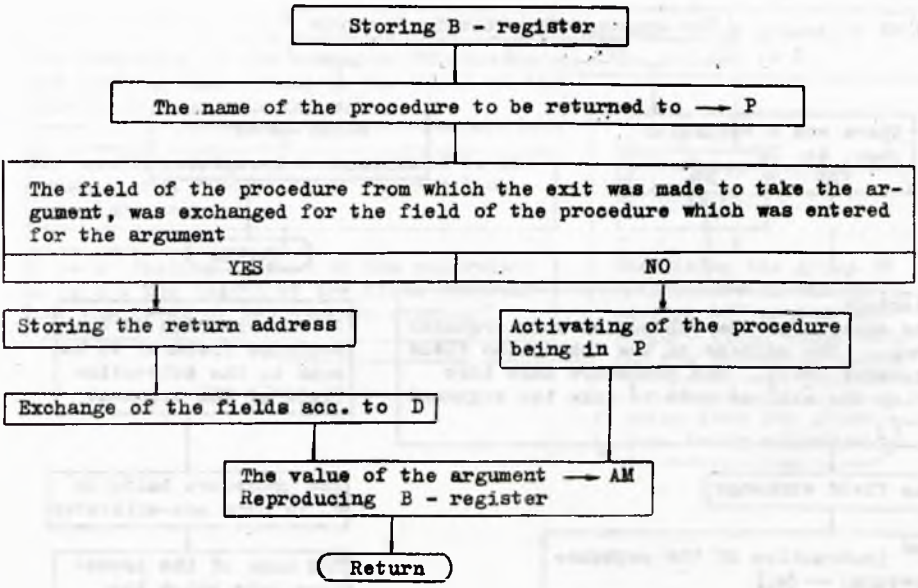


RETURN FROM THE ARGUMENT

WAR - P07

The sequence: P07 R_{λ}

The subroutine:

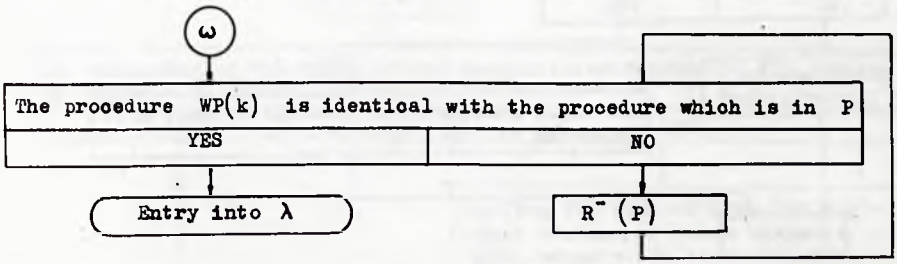


EXIT FROM THE PROCEDURE THROUGH GO TO

The sequenoe: UBA λ
UAK WP(k)
SKO ω

λ - the label
WP(k) - the representative in the procedure vector of the procedure to which λ belongs

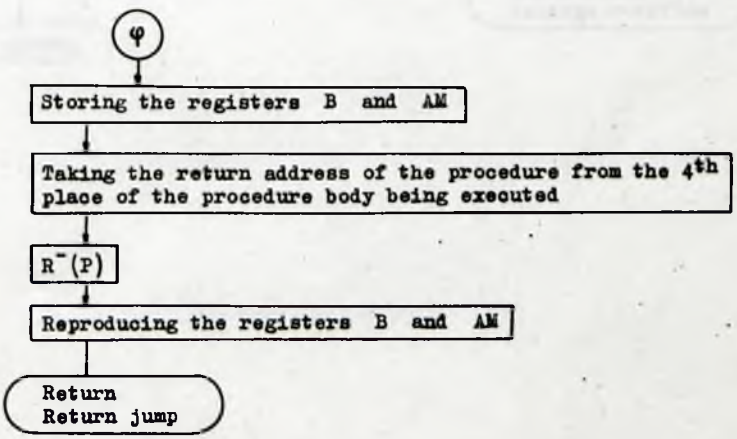
The subroutine:



EXIT FROM THE PROCEDURE THROUGH RETURN

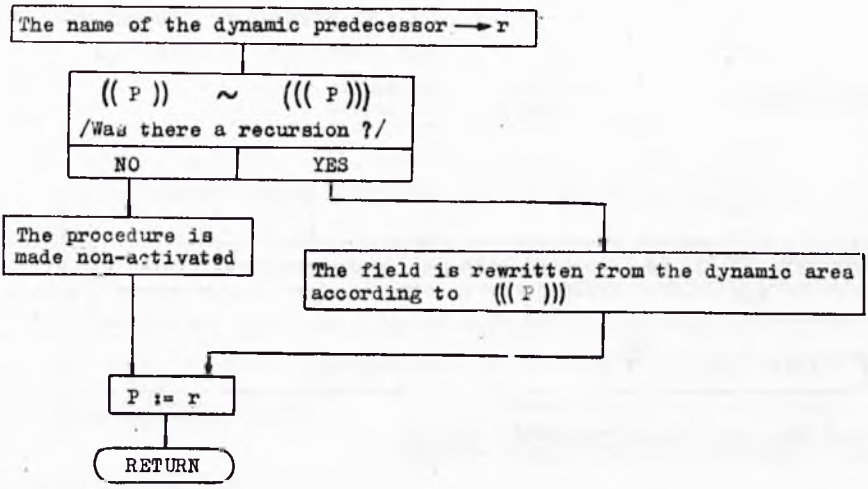
The sequenoe: SKO φ

The subroutine:



SUBROUTINE $R^{\sim}(P)$

The subroutine is carried out using the register P.

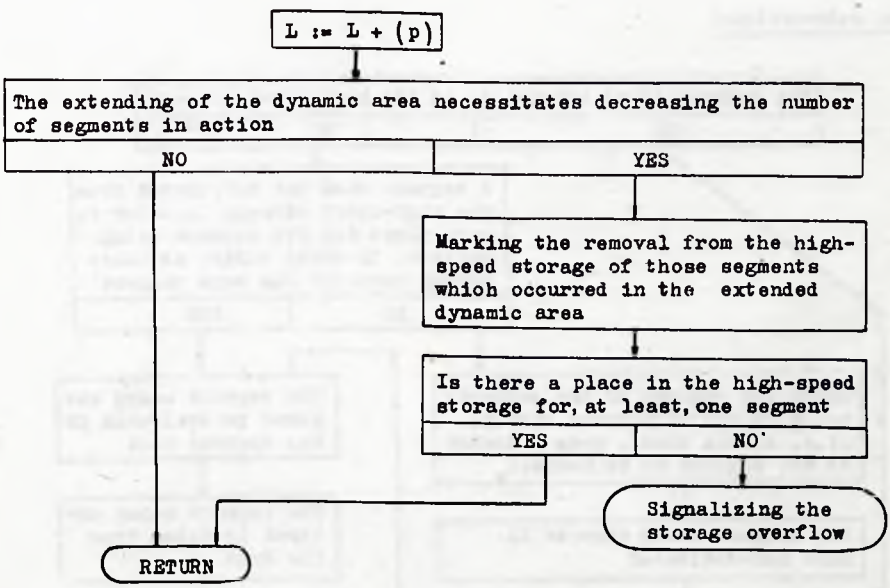


INCREASING REGISTER L

ZWL - P22

The sequence: P22 p p - the address containing the number of the increase of L

The subroutine:



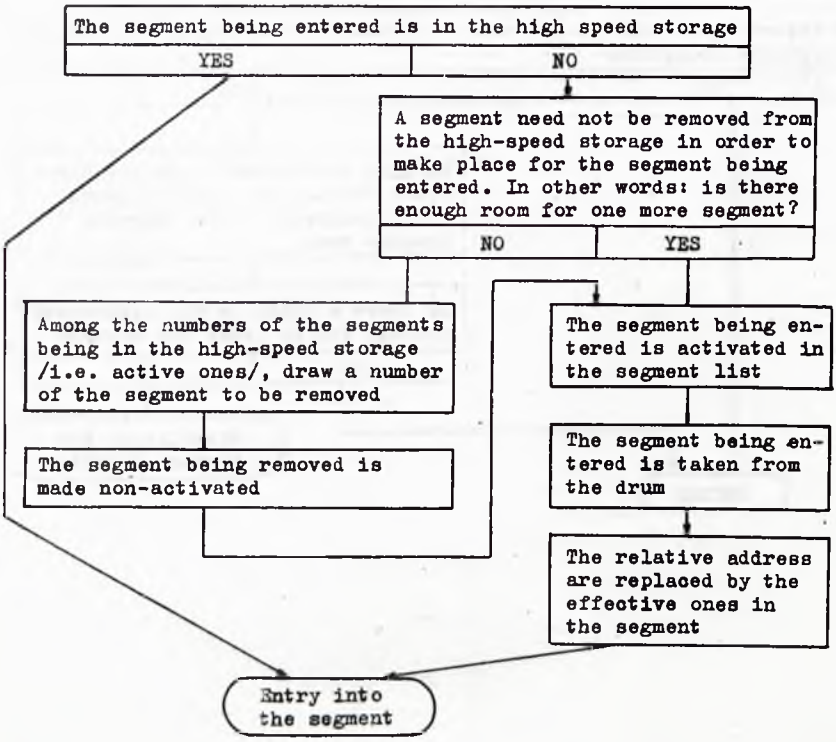
SEGMENT JUMP /SEGMENTS EXCHANGE SUBROUTINE/

SEG – P08

The sequence: P08(n, m)

n – the segment number
m – the address towards the beginning of the segment of the place being entered

The subroutine:



Bibliography

[1] NAUR P.: Revised ALGOL 60 Report ACM, Vol. 6.
[2] NAUR P.: The design of the GIER ALGOL Computer BIT 3/1/63.
[3] ŁUKASZEWICZ L.: Rodzina maszyn ZAM.
[4] CZAJA L., SZORC P.: Storage Allocation for ALGOL, Algorytmy No 7, 1967.

Instytut Maszyn Matematycznych
Algorytmy N° 7

681.3.06

THE STRUCTURE OF PROCEDURES ADJOINED TO THE ALGOL SYSTEM FOR ZAM COMPUTERS

by Krzysztof MOSZYŃSKI
Ryszard POGORZELSKI

Received June 30th, 1966

The paper gives a description of structures of procedures, adjoined to the ALGOL system for ZAM computers. Given: The organization of adjoined procedures. The cooperation of adjoined procedures with the ALGOL system and the proposed equipment of the translator system with declarations facilitating this cooperation. Principles of using the adjoined procedures.

1. INTRODUCTORY NOTES

Adjoined procedures are those which are not written in the autocode, but adapted to be adjoined by means of a translator to a program written in ALGOL. The adjoined procedures should satisfy the following conditions:

- The use of the adjoined program does not necessarily require its recording in the main program, i.e. tapes comprising adjoined procedures must not be written every time in the program.
- The use of the adjoined procedure ought to be possible after getting acquainted with the formal description of parameters and results, its internal structure being of no interest.
- Adjoined procedures should be more optimal, as regards the operation time and the number of places in the storage than analogous procedures written in autocode.
- The time of program translation comprising adjoined procedures should be shorter that the time of program translation fully written in ALGOL.

Two possibilities are foreseen:

For computers with magnetic tapes the library of adjoined procedures should be recorded on magnetic tapes.

For computers not equipped with magnetic tapes separate procedures should be prepared in the form of separate paper tapes with catalogue numbers.

In both cases a certain system of checking should be foreseen, operating during the procedure input to the storage e.g. by means of control sums.

2. THE USE OF ADJOINED PROCEDURES

At the beginning of the block which uses the procedure, a declaration should be placed as follows:

. prooedure name of procedure /parameters/;
. value part specification of arguments; library /"catalogue number"/.

The name is every time given by the programmer who uses the procedure. The catalogue number of the adjoined procedure is being established. The value part, specification and the numbers of parameters as well as their successiveness, are imposed by the description. The call to a so-declared procedure is the same as to the ALGOL procedure.

3. GENERAL CONSTRUCTION OF ADJOINED PROCEDURES

The adjoined procedure consists of:

- content of the procedure,
- field of the procedure.

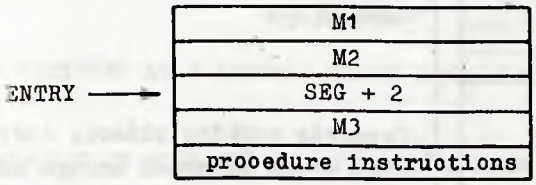
The content of the procedure comprises instructions and it must agree with segmentation rules.

The field of the procedure serves for storing:

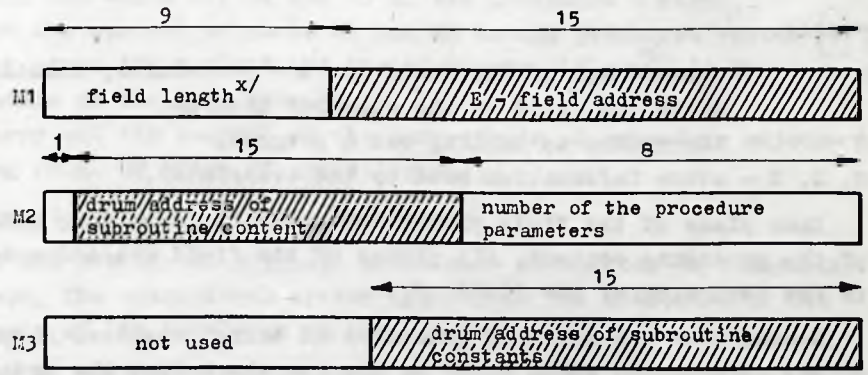
- parameters needed for cooperating with the ALGOL system,
- information about the procedure parameters,
- separate working places,
- information about working arrays,
- procedure constants.

The content of the procedure is being written on the drum by the operational system. If constant procedures appear, they must by placed directly behind the content /on the drum/.

The content of the procedure is the following:



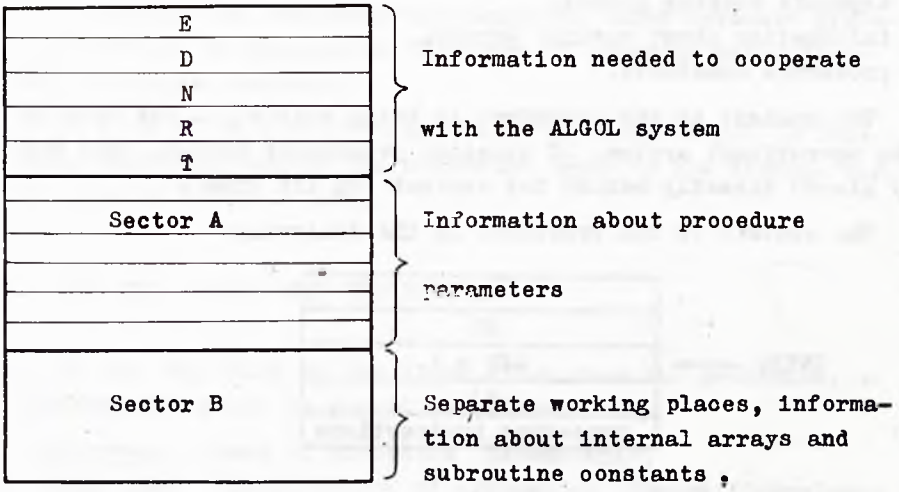
Words M1, M2 and M3 comprise the following information:



The hatched places are filled up by the translator or by the operational system, and those that are not hatched – by the one who writes the procedure. The zero-bit of the word M2 is destined for information about the procedure activation [1].

*) The field length is counted in long words /48 bits/.

The field of the procedure is the following:



where:

R – the address of the field of λ – procedures, allocated in the field of the superior program,
T – the return to calling out a program,
N, D, E – other information used by the translator.

Each place of the field should be accessible from each place of the procedure content. All places of the field are addressed to the beginning of the field P_0.

Sector B of the field of procedure is being organized /appropriate information being sent/ by the one who writes the procedure but the remaining parts of the field are filled up by the translator. The translator sends information to sector A about actually given parameters successively as they appear in the procedure declaration. Two words l_i and l_i' are destined for every parameter. If the procedure uses its own contants, the content of the

procedure is being called every time, using information comprised in M3 and M1, and it should rewrite the constants from the drum to sector B of the subroutine field.

The cooperation with the drum storage, treated as an auxiliary storage, is organized internally by the adjoined procedure. Therefore, an information is needed, accessible from the procedure content, about the not occupied part of the drum.

The return from the adjoined procedure to the program which calls out the above procedure is made by means of the instruction

SKO "RETURN"

where "RETURN" is a certain fixed absolute address.

4. COOPERATION WITH THE OPERATIONAL SYSTEM

The operational system should:

- fill out words M1, M2 and M3 of the procedure content,
- put the content of words M1 and M2 to the procedure vector [1],
- introduce the content and the constants /if any*)/ of the procedure to the drum storage,
- carry out the control of the correctness of procedure adjoining /e.g. by means of control sums/,

whereas:

if the library of adjoined procedures is written on a magnetic tape, the operational system identifies the procedure on the basis of its catalogue number,

but in the case of procedures on paper tapes, separately adjoined to every program, the operational system checks whether the

*) In connection with this it seems to be necessary to complete the operational system with declarations signalizing the content and the constants of the adjoined procedures.

number on the tape agrees with the number in the declaration*) and it possibly signalizes the error.

5. THE USE OF PARAMETERS IN AN ADJOINED PROCEDURE

The following may appear as an actual adjoined procedure:

1. expression,
2. array /by name/,
3. procedure.

At the moment of calling out the procedure, the translator sends information about actually given parameters to places l_i and l_i' of sector A in the procedure field. This information is being recorded in a successiveness concordant with the successiveness of appearing of the parameters in the procedure declaration.

Let us consider the form of the information and its use to an adjoined procedure.

ad 1. The use of a parameter depends on its appearance in a set of values. According to this let us consider two cases:

- When the expression is substituted by value then the value of this expression is in l_i and l_i' places. The way of using is evident.
- When the expression is substituted by name then the address of λ- procedure**) is sent to place l_i corresponding to the given parameter, but the information about the kind of parameter is sent to place l_i', according to the table.

---

*) The successiveness of the entered tapes should agree, with the successiveness of declarations of procedures adjoined in the program.

**) See point 6.

Table of the kind of parameters

0 - simple variable,
1 - indioed variable,
2 - expression, number, procedure,
3 - array.

In the given case number 0 or number 2 is being sent to l_i'.

The value of the expression is obtained by means of the following sequence of instructions

ARG l_i
SEG + 1

The results of these instructions are the following:
In registers A and M - the expression value is written in a floating point*). In register B - the address of the above given expression value**) in the field of a superior program /calling out the given procedure/.

ad 2. The address of the beginning of the array is being sent to place l_i', and the address of the so-called dope vector $[1]$ - to place l_i'. The element $a[i_1, \ldots, i_n]$ is taken according to the following instructions:

HOR l_i
r_1
r_2
$\vdots$
r_n

where $r_1, \ldots, r_n$ are addresses of indices $i_1, \ldots, i_n$.

*) A - accumulator, M - multiplier, B - modification register.
**) The variable is treated as a special case of an expression.

As a result we obtain:

- the value $a[i_1, \ldots, i_n]$ in registers A and M,
- the address $a[i_1, \ldots, i_n]$ in register B.

The construction of the dope vector of the array in the form e.g.

$$a\left[\alpha_1 : \beta_1, \alpha_2 : \beta_2\right]$$

is the following:

number of indices
$(\beta_1 - \alpha_1 + 1)$
$\beta_2 - \alpha_2 + 1$
φ_0

where:

$$\varphi_0 = (\beta_2 - \alpha_2 + 1)\,\alpha_1 + \alpha_2$$

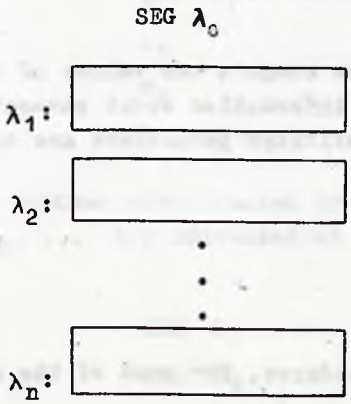
The address x of the element $a[i, j]$ counting from the beginning of the array, is obtained from the formula:

$$x = (\beta_2 - \alpha_2 + 1)\,i + j - \varphi_0$$

When using the above given information about the construction of the dope vector the programmer is not obliged to use standard instruction order for element $a[i_1, i_2]$. Thus, computations in separate procedures can be significantly accelerated.

ad 3. The address in the procedure vector, corresponding to the substituted procedure, is sent to place l_i, and number 3 /see table/ - to place l_1'.

In order to call out the substituted procedure, the following order of instructions should be used:




```
λ0: AKT  l1 ℜ
    STS  Rλ
    SEG  + 2n + 1
    SEG  λ1
    STS  kind(λ1)
    ..............
    SEG  λn
    STS  kind(λn)
```

where $kind(\lambda_1)$ is the information about the kind of parameter λ_1 /see table/.

Fragments of the above sequence denoted by labels λ_1, λ_2, ..., λ_n are λ - procedures. These procedures serve to compute the parameters substituted to the called out procedure. The construction of λ - procedures is described below. Instructions starting with label λ_0 activate the called out procedure, and they simultaneously transfer to it the information about λ- procedures, R_λ denotes the address of λ - procedure working places. Two subsequent short words be reserved for these working places in sector B of the field of the adjoined procedure. l_1 denotes, as usual, the address of the first word in sector A, corresponding to the considered parameter /which is, in this case, a procedure/.

6. CONSTRUCTION OF λ-PROCEDURES

λ-procedures serve to compute the values of substituted parameters, or to transfer information about parameters to the activated procedure. Substituted parameters are divided into four groups:

1. Simple variables
2. Expressions
3. Arrays /by name/
4. Procedures.

Appropriate λ-procedures, for each of the above group are the following:

ad 1. UMB – the name of the procedure to which belongs X
UMB 1 +
UAA X +
WRO 1.

where X is the address of the variable.

ad 2.

Instructions calculating the expression value

WAR R_λ

In the case when the expression contains a parameter which was previously substituted by name the sequence preceding the instruction WAR R_λ must include the instructions:

ARG l_i
SEG + 1

where l_i is the address, corresponding to this argument in sector A. These instructions cause a call to the proper λ-procedure, which is beyond the adjoined procedure, in order to calculate values of the substituted argument. Similarly, if the substituted expression contains the array element, the sequence preceding the instruction WAR R_λ should comprise

```
HOR  l_i
r_1
r_2
 :
```

where l_i is the address subordinated to the above array in sector A, and r_1, r_2, ... are addresses of indices.

ad 3.

```
UAM  l_i
WAR  R_λ
```

where l_i - appropriate address in sector A,

R_λ - working place for λ -procedure in sector A.

ad 4.

```
AKT  l_i
SEG + 2
STS  50.
WAR  R_λ
```

where l_i - appropriate address in sector A,

R_λ - working place for λ -procedures in sector B.

7. OWN ARRAYS

An area in the internal storage is reserved for arrays. The size of this area changes dynamically. The highest level of arrays, actually stored, is in the register /a fixed storage place/ and is denoted by symbol L*) [2]. In the case of calling from an adjoined procedure to other ones, the content of register L may be augmented. Therefore, before calling out another procedure,

*) The address of register L is commonly accessible.

but after having built up all arrays, the content of register L should be stored in the working place L′ in sector B of the field, and after having returned from the called out procedure, the content of register L should be restored, by means of transforming L′ to L.

The adjoined procedure can form its own arrays, so-called internal arrays. They do not appear as parameters. The address comprised in L should be accepted as the beginning of the formed internal arrays. Let us consider two cases:

- when the formed array is not be substituted to another procedure,
- when the internal array is the argument of another procedure.

In the first case, the way of using these arrays is the same as the one of using simple working places, and it depends exclusively on the one who writes the procedure. In the second case, the arrays should be formed according to the ALGOL structure. One should then make the dope vector for the internal array in sector B, and subordinate to this array two subsequent words in sector B. These words will comprise:

- the address of the beginning of the array,
- the address of the dope vector.

In both cases, when the adjoined procedure makes its own arrays, the content L should be increased by the global number of places occupied by these arrays. This is made by means of the instruction

ZWL p

where p is the address of the number by which L should be augmented.

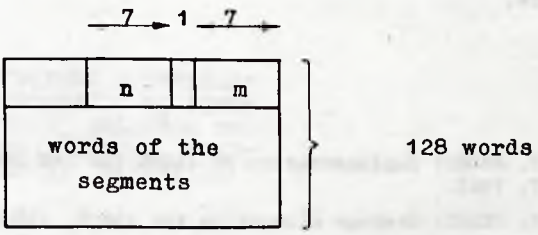
8. REMARKS ON THE LANGUAGE OF ADJOINED PROCEDURES

A possibility od preparing adjoined procedures would be desirable:

- in a language resembling the computer code /e.g. SAS/,
- in ALGOL, with a possibility of translating it to a form that meets the demands for adjoined subroutines /see point 1/.

Let us consider the first case only.

Procedures written in a language resembling the computer code must be subjected to the segmentation rule [2]. The procedure segmentation must be relative, i.e. the programmer subordinates subsequent numbers, starting with zero, to the segment, and if using such a procedure, the translator assigns appropriate numbers to its segments. The procedure segment is of the following structure which is imposed the translator of the ALGOL system for ZAM computers:



when n – the number of the first instruction in the segment to be readdressed

m – the number of the segment.

The communication among segments occurs only by "segment jump".

SEG (m, n)

where m – the relative number of segment, n – the place in the segment.

It would be desirable to equip the system with segment conditional jumps. Because of the necessity of using two reference points /the beginning of the segment and the beginning of the field/ it is necessary to distinguish the instructions addressed to the segment and those addressed to the beginning of the field.

At the same time only addresses to the beginning of the segment and to the beginning of the field are claimed, without giving information about the position of the next instruotion being readdressed.

The present paper is based on an actual stage of realization of the design of ALGOL translator for ZAM computers. Therefore, some changes are possible resulting from optional changes introduced to the ALGOL system.

The authors wish to express their great thankfulness to Mr L. Czaja for his contribution to the elaboration of the present paper. His valuable remarks and explanations of details of the ALGOL system realization for ZAM computers were of essential help to the authors.

References

[1] L. CZAJA, P. SZORC: Implementation of ALGOL for ZAM Computers, ALGORYTMY No 7, 1967.

[2] L. CZAJA, P. SZORC: Storage Allocation for ALGOL, ALGORYTMY No 7, 1967.

Instytut Maszyn Matematycznych
Algorytmy N° 7

681.3.06

СИСТЕМА АВТОМАТИЧЕСКОГО ПРОГРАММИРОВАНИЯ М И К О Д ДЛЯ МАШИНЫ М И Н С К - 2.

П.БЫРНЕВ
Г.ПЕНЧЕВ
Д.ШИШКОВ
Р.КАЛТИНСКА
М.АПОСТОЛОВА
(София)

Статья поступила в редакцию 14.05.1966

В статье подана основная информация по системе автоматического программирования электронной цифровой вычислительной машины Минск-2 (система близка к автокоду МАРК-II).

Система автоматического программирования МИКОД была создана в 1965 г. в Математическом Институте с Вычислительным центром БАН. Она предназначена для автоматизации программирования на машине МИНСК-2.

Система состоит из входного языка, транслятора, библиотеки СП и ряда рабочих программ.

Алгоритмы описываются символически на входном языке как последовательность операторов. В начале полученной таким образом программы ставятся описания границ изменений индексов индексированных переменных. Программа, написанная на входном языке, перфорируется во 2-ом международном телеграфном коде и вводится в машину. Транслятор просматривает программу и выводит на печать все замеченные формальные ошибки. Если нет ошибок, программа переводится на машинный язык, выводится из машины вместе с некоторыми таблицами чи-

сел и величин и по желанию программиста может быть сразу выполнена. Для этого, конечно, должны быть подготовлены на вводе все входные данные.

Система МИКОД предназначена для программирования научно-технических задач, для которых вычисления проводятся на числах с плавающей запятой. Использование системы для логических задач, в которых основными элементами являются части ячеек памяти, неудобно, хотя некоторая возможность предусмотрена для этого.

Переменные означаются посредством букв и цифр, числа записываются в десятичной системе в форме близкой к общепринятой: возможно использование целых чисел. Допускаются переменные лишь с одним индексом. Большое количество индексов можно реализовать только косвенным образом.

Арифметические формулы надо разлагать на одно - и двуаргументные операции.

Система допускает использование стандартных программ, написаных в машинном коде и включённых в библиотеку СП машины, точнее в систему МИС (Интерпретирующая система машины МИНСК-2).

Распределение оперативной памяти производится транслятором автоматически. Распределение внешней памяти предоставляется программисту. При переводе, транслятор проводит почти полную оптимизацию в смысле команд, индексных ячеек и распределения памяти.

Циклы осуществляются при помощи индексных ячеек. Для параллельных циклов одинаково уровня используются одни и те же индексные ячейки.

Для системы МИКОД создана специальная входная система для ввода данных с контролем и исправлением ошибок перфорации и программиста.

Транслятор состоит из трёх основных блоков: блок массивов, блок циклов и блок программирования. Они работают совместно, используя несколько вспомогательных блоков.

После ввода текста посредством входной системы, работа транслятора происходит в трёх этапах. На первом этапе блок массивов просматривает описание массивов, обнаруживает в них ошибки и рас-

пределяет массивы в оперативной памяти. На втором этапе блок циклов просматривает всю остальную (операторную) часть информации, выявляет структуру циклов и распределяет в памяти числа. Одновременно блок циклов выполняет функцию синтаксического контроля и обнаруживает все формальные ошибки. Если ошибки не обнаружены, тогда на третьем этапе блок программирования создаёт рабочую программу. Таким образом, трансляция происходит в течение двух просмотров информации.

Система МИКОД занимает примерно $4600_{(10)}$ ячеек памяти.

Instytut Maszyn Matematycznych
Algorytmy № 7

681.3.06

КОМПИЛИРУЮЩАЯ СИСТЕМА М И К С ДЛЯ ИСПОЛЬЗОВАНИЯ БИБЛИОТЕКИ СТАНДАРТНЫХ ПРОГРАММ ДЛЯ МАШИНЫ М И Н С К - 2.

П.Х.БЪРНЕВ
Д.И.ТОШКОВ
(София)

Статья поступила в редакцию 14.05.1966

Изложено описание компилирующе-интерпретирующей системы, предназначенной для использования библиотеки стандартных подпрограмм.

Существуют две системы использования библиотеки стандартных программ для машины МИНСК-2: Расстановочная программа (РП) [1] и МИС [2] , первая-компилирующего типа, а вторая-интерпретирующего. Обе системы в некотором смысле дополняют друг друга. Однако функции системы РП немного проще функции МИС, что создает ряд неудобств при работе с РП. В самом деле, работа РП сводится к вызову и настройке стандартных программ (СП) по внутренним адресам. Программы вызываются на определённые программистом места в оперативную память или последовательно одна за другой. Обращения к СП реализуются безусловной передачей управления с возвратом (команда - 31). Это приводит к некоторым вычислениям со стороны программиста в связи с размещением СП в ОП. Настройка СП согласно параметрам обращения и сохранение содержимого рабочих ячеек предоставляется СП. Имеются неудобства при использовании СП в других СП. Нерационально организованная работа с магнитными лентами приводит к лишним затратам машинного времени. Неудобством также является двукратное перерывание автоматической работы машины.

В предлагаемой системе МИКС сделана попытка избежать вышеприведённые недостатки и ввести некоторые дополнительные усовершенствования. С другой стороны, с целью упростить использование системы, вводятся некоторые незначительные ограничения.

Система МИКС состоит из компилирующей программы (КП), каталога СП (КСП), библиотеки СП (БСП), двух рабочих программ (для включения новых СП в БСП и для контроля системы) и некоторых специальных СП, непосредственно связанных с использованием системы (СП для фиксации содержимого рабочих ячеек, СП для настройки по внутранним и условным адресам, СП для извлечения параметров из обращения в СП). Кроме того, предусмотрена группа констант, которая может быть записана в конце ОП по желанию программиста. КСП содержит по одной ячейке для каждой СП из БСП. В МИКС, как и в РП, считается, что БСП и все необходимые рабочие программы записаны на магнитной ленте.

Недостатком систем компилирующего типа, в которых настройка по параметрам выполняется КП, является необходимость размещать одну и ту же СП несколько раз в ОП, если параметры в разных обращениях неодинаковы. В связи с этим, в МИКС приняты некоторые меры для минимизации длины участка ОП, в котором будут помещаться необходимые СП.

В МИКС имеются три вида параметров в обращениях к СП, обозначаемые соответственно λ, μ и ν. Настройка СП по λ - параметрам выполняется КП, по μ - параметрам выполняется соответствующей СП, а настройка по всем ν - параметрам выполняется одновременно по желанию программиста или КП, или СП, в зависимости от признака π_0, данного в обращении. Поэтому, при составлении данной СП, в неё должен быть включён блок μ и ν - настройки (если в обращении имеются параметры μ и ν. При этом, блок ν - настройки вводится в ОП в случае, если имеются обращения к данной СП с признаком для ν - настройки. Таким образом можно сэкономить место в ОП за счёт блоков настройки отдельных СП, а с другой стороны, можно вызвать данную СП вместе с блоками μ и ν - настройки только один раз, если к этой СП будут обращения с различными μ и ν - параметрами. При различии в λ - параметрах приходится вызывать СП в ОП несколько раз. Это осуществляется специальным параметром κ в обращении. Тип параметров выбирается составителем СП.

Как было сказано выше, настройка СП по λ- (и ν-) параметрам производится КП. В данном адресе или в оперативной части каждой команды СП допускается формирование любой линейной функции $ap_i \pm b$ параметра p_i с некоторыми ограничениями для целых чисел a и b. Информация для этой функции кодируется как условный адрес на соответствующем месте в СП.

Рассматриваются два вида СП: простые СП (ПСП) и сложные СП (ССП). ПСП не имеют параметров и не используют других СП. Они выполняют преобразования одного аргумента, заданного в сумматоре, и посылают результат опять в сумматор. Для избежания, в большинстве случаев, необходимости фиксации содержимого рабочих ячеек, предусмотрены отдельные группы рабочих ячеек для ПСП и ССП. Таким образом приходится фиксировать содержимое рабочих ячеек только при использовании ССП в нестандартных блоках другой ССП, а это можно легко сделать при помощи специальной СП для фиксации.

Обращение к ПСП имеет вид:

-00 00 0000 $№_{СП}$

а к ССП:

-00 К 0000 $№_{СП}$

$Jo\,po$ p_1 p_2

.....

p_{3m} p_{3m+1} p_{3m+2}

Необходимая информация для работы МИКС задаётся в одной фиксированной ячейке. СП размещаются одна за другой начиная с конца ОП. КП вызывается вместе с КСП в начале ОП и два раза просматривает указанный программистом участок ОП. При обращении к данной СП, происходит вызов и настройка этой СП с заменой первой (условной) строки обращения в действительную. При первом просмотре вызываются только ССП и относящиеся к ним ПСП. При втором просмотре вызываются остальные необходимые ПСП, которые ещё не вызваны в ОП. После окончания работы КП передает управление в указанную программистом ячейку.

При работе КП число обращений к магнитным лентам примерно в два раза меньше чем в РП.

КП занимает $350_{(8)}$ ячеек ОП. Следующий за ней КСП занимает не более $177_{(8)}$ ячеек.

Литература

[1] Библиотека стандартных программ для ЦВМ МИНСК-2, Москва, 1963.

[2] Нефедьева, Л.С. и Ян-Фу-цин: Система интерпретации и библиотека стандартных программ для ЭВМ МИНСК-2. ОИЯИ, Дубна, 1965.

Druk. „Znak" Zl. 658/819 Nakl. 730 M-52