MODELOWANIE I SYMULACJA SYSTEMÓW KOLEJKOWYCH W ŚRODOWISKU MATLAB – STUDIUM PRZYPADKU

KRZYSZTOF JURCZYK, MONIKA KLAŚ, WOJCIECH WOŹNIAK

Streszczenie

W pracy zaproponowano model symulacyjny systemu kolejkowego zbudowanego z pięciu obiektów, którymi są dwa bufory wejściowe, dwie stacje przetwarzające oraz realizator zadań w postaci podajnika transportującego elementy przepływu dwóch rodzajów. Model został utworzony w postaci pliku funkcyjnego zapisanego w oprogramowaniu Matlab. W kolejnych akapitach artykułu szczegółowo omówiono zasadę działania utworzonego programu oraz przedstawiono wyniki testów numerycznych, jakie przeprowadzono w celu zbadania jego stabilności oraz szybkości działania.

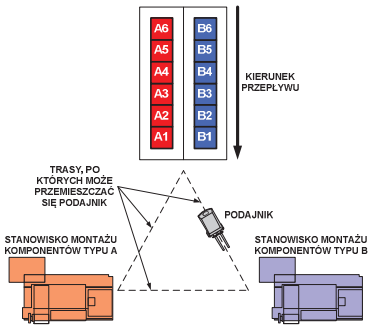
Słowa kluczowe: języki programowania, modelowanie i symulacja, specjalistyczne pakiety obliczeniowe, systemy kolejkowe

Wprowadzenie

Symulacja komputerowa stanowi doskonałe narzędzie służące analizie złożonych procesów czy systemów logistycznych. Znajduje zastosowanie w sytuacjach, gdy zawodzą dostępne metody analityczne jednocześnie dostarczając jak największej liczby informacji o analizowanym procesie. Koniecznym jest jednak wybór odpowiednich narzędzi wykorzystywanych do symulacji procesów, wśród których można wymienić języki programowania ogólnego przeznaczenia, specjalistyczne języki symulacyjne, specjalistyczne pakiety symulacyjne oraz arkusze kalkulacyjne [4, 6]. Szerokiej analizy wymienionych klas narzędzi stosowanych w symulacji procesów dokonali w swoich pracach m.in. B. Noche i S. Wenzel [5] oraz J. Banks [1]. W niniejszym artykule wykazano korzyści, jakie można uzyskać stosując specjalistyczne pakiety obliczeniowe takie jak np. oprogramowanie Matlab w symulacji systemów i procesów logistycznych [2].

1. Założenia do budowy modelu symulacyjnego

Rozważmy system kolejkowy zbudowany z pięciu elementów, tj. dwóch buforów, dwóch stacji montażowych oraz podajnika realizującego transport elementów przepływu pomiędzy buforami i/lub stacjami montażowymi. Jeden podajnik obsługuje dwa stanowiska montażowe dedykowane dla dwóch różnych komponentów, oznaczonych jako A oraz B. Czasy między przybyciami, czasy montażu każdego z komponentów, czasy transportu komponentów przez podajnik z wejścia systemu na stanowisko montażowe oraz ze stanowiska montażowego do są wielkościami znanymi. Ponadto występują osobne strumienie wejściowe dla każdego z komponentów A oraz B. Podobny model był wcześniej analizowany przez autorów ze względu na sposób ustalania liczby powtórzeń eksperymentu symulacyjnego [3]. Na rysunku 1 zaprezentowano schemat realizacji przepływów w omawianym systemie.



Rysunek 1. Schemat analizowanego systemu kolejkowego

Źródło: opracowanie własne.

Celem prezentowanego w niniejszej pracy eksperymentu symulacyjnego jest ustalenie sposobu generowania zmiennych wynikowych takich jak: średni czas oczekiwania komponentu na podajnik $\overline{T_{ocz}}$, średni czas przebywania komponentu w systemie $\overline{T_{prz}}$, całkowitą długość harmonogramu produkcyjnego T_{sum} oraz sumaryczny czas bezczynności podajnika T_{bcz}.

2. Model symulacyjny utworzony w środowisku Matlab

W tabeli 1 zamieszczono plik funkcyjny utworzony w środowisku Matlab służący analizie przedstawionego w poprzednim podrozdziale systemu kolejkowego.

Tabela 1. Model symulacyjny w postaci pliku funkcyjnego zapisanego w środowisku Matlab

Lina	Kod programu
1	`function [Harmonogram, Tocz_sum, Bezc_sum, Harm_dl, Tprz_sr] = podajnik(TmpA,`
2	`TmpB,...`
3	`...TobA, TobB, THIA, THIB, THOA, THOB);`
4	`a = length(TmpA);`
5	`b = length(TmpB);`
6	`n = a + b;`
7	
8	`Twe = [cumsum(TmpA); cumsum(TmpB)];`
9	`Typ = [ones(a,1); zeros(b,1)];`
10	
11	`[Twe_sort, I] = sort(Twe);`
12	`Typ_sort = Typ(I);`
13	
14	`Tob = [TobA; TobB];`
15	`Tob_sort = Tob(I);`
16	
17	`ostatni = zeros(n,1);`
18	
19	`for i = 2:n`
20	`if Typ_sort(i) == Typ_sort(1)`
21	`ostatni(i) = find(Typ_sort(1:i-1)==Typ_sort(1),1,'last');`
22	`end`
23	`end`
24	`pomoc = find(Typ_sort~=Typ_sort(1),2,'first');`
25	`for i = pomoc(2):n`
26	`if Typ_sort(i) == Typ_sort(pomoc(1))`
27	`ostatni(i) = find(Typ_sort(1:i-1)==Typ_sort(pomoc(1)),1,'last');`

```
28          end
29      end
30
31      THI = zeros(n,1);
32      THO = zeros(n,1);
33      for i = 1:n
34          if Typ_sort(i) == 1
35              THI(i) = THIA;
36              THO(i) = THOA;
37          else
38              THI(i) = THIB;
39              THO(i) = THOB;
40          end
41      end
42
43      Tocz1b = zeros(n,1);
44      Tocz2 = zeros(n,1);
45
46      Bezczynnosc = [0; -ones(2*n-1,1)];
47
48      Trw = zeros(n,1);
49      Tzw = zeros(n,1);
50      Twy = zeros(n,1);
51
52      Trw(1) = Twe_sort(1) + THI(1);
53      Tzw(1) = Trw(1) + Tob_sort(1);
54
55      while min(Bezczynnosc)<0
56
57          Twy(1) = Tzw(1) + Tocz2(1) + THO(1);
58
59          if min(pomoc)>2
60              for i = 2:pomoc(1)-1
61                  Trw(i) = max(Twy(i-1),Twe_sort(i)) + THI(i);
62                  Tzw(i) = Trw(i) + Tob_sort(i);
63                  Twy(i) = Tzw(i) + Tocz2(i) + THO(i);
64              end
65          end
66
67          Trw(pomoc(1)) = Twe_sort(pomoc(1)) + THI(pomoc(1)) + Tocz1b(pomoc(1));
68          Tzw(pomoc(1)) = Trw(pomoc(1)) + Tob_sort(pomoc(1));
69          Twy(pomoc(1)) = Tzw(pomoc(1)) + Tocz2(pomoc(1)) + THO(pomoc(1));
70
71          for i = (pomoc(1)+1):n
72              Trw(i) = max([Twe_sort(i), Twy(ostatni(i))]) + THI(i) + Tocz1b(i);
73              Tzw(i) = Trw(i) + Tob_sort(i);
74              Twy(i) = Tzw(i) + Tocz2(i) + THO(i);
75          end
76
77          ID_komponentu = [(1:n)' ; (1:n)'];
78          H_type = [ones(n,1); zeros(n,1)];
79          THall = [Trw - THI , Trw ; Twy - THO, Twy];
80
81          [THall_sort_od, K] = sort(THall(:,1));
82          THall_sort_do = THall(K,2);
83          ID_komponentu_sort = ID_komponentu(K);
84          H_type_sort = H_type(K);
85
86          Bezczynnosc = THall_sort_od - [0; THall_sort_do(1:2*n-1)];
87          kolizja = find(Bezczynnosc<0,1,'first');
88
89          if H_type_sort(kolizja)==0
90      Tocz2(ID_komponentu_sort(kolizja))=Tocz2(ID_komponentu_sort(kolizja))-Bezczyn-
91  nosc(kolizja);
92          elseif H_type_sort(kolizja)==1
```

```
93          Tocz1b(ID_komponentu_sort(kolizja))=Tocz1b(ID_komponentu_sort(kolizja))-Bezczynnosc(ko-
94      lizja);
95              end
96
97          end
98
99          Tocz1 = Trw - Twe_sort - THI;
100         Tprz = Twy - Twe_sort;
101
102         Tocz_sum = sum(Tocz1) + sum(Tocz2);
103         Bezc_sum = sum(Bezczynnosc);
104         Harm_dl = THall_sort_do(2*n);
            Tprz_sr = mean(Tprz);

            Harmonogram = [Twe_sort, Tocz1, THI, Tob_sort, Tocz2, THO];
```

Źródło: opracowanie własne.

Kod programu rozpoczyna się od deklaracji, że utworzony program będzie plikiem funkcyjnym (linia kodu 001). Po słowie kluczowym `function` następuje deklaracja zmiennych wejściowych oraz wyjściowych utworzonej funkcji. Jako zmienne wejściowe określono wektor czasów między kolejnymi pojawieniami się komponentów typu A na wejściu systemu `TmpA`, wektor czasów między kolejnymi pojawieniami się komponentów typu B na wejściu systemu `TmpB`, wektor czasów obróbki kolejnych komponentów typu A na dedykowanym stanowisku montażu `TobA`, wektor czasów obróbki kolejnych komponentów typu B na dedykowanym stanowisku montażu `TobB`, czas transportu komponentu typu A z bufora do stanowiska montażu `THIA`, czas transportu komponentu typu B z bufora do stanowiska montażu `THIB`, czas transportu komponentu typu A ze stanowiska montażu do wyjścia z systemu `THOA` oraz czas transportu komponentu typu B ze stanowiska montażu do wyjścia z systemu `THOB`. Jako zmienne wyjściowe funkcji przyjęto macierz obrazującą poszczególne składowe czasowe kolejno pojawiających się komponentów `Harmonogram`, sumę czasów oczekiwania wszystkich komponentów `Tocz_sum`, całkowity czas bezczynności podajnika `Bezc_sum`, długość harmonogramu `Harm_dl` oraz średni czas przebywania komponentów w systemie `Tprz_sr`.

W liniach kodu 003 oraz 004 obliczana jest długość wektorów `TmpA` oraz `TmpB`. Pod zmienną `a` zostanie w ten sposób przypisana liczba komponentów typu A, natomiast pod zmienną `b` liczba komponentów typu B, które pojawiły się na wejściu systemu. W linii kodu 005 obliczana jest suma wszystkich komponentów, a następnie obliczona wartość zostaje przypisana pod zmienną `n`.

W linii kodu 007 przy wykorzystaniu funkcji `cumsum` obliczane są sumy skumulowane wartości z obu wektorów obrazujących interwały czasowe między pojawieniem się kolejnych komponentów danego typu, a następnie uzyskane wartości zapisywane są do jednego wektora kolumnowego `Twe` obrazującego czasy pojawienia się komponentów w systemie. Uzyskany wektor `Twe` ma długość odpowiadającą sumie długości wektorów `TmpA` oraz `TmpB`, czyli `n`. Pierwsze `a`-wartości odpowiada kolejnym czasom pojawiania się komponentów typu A, natomiast kolejne `b`-wartości odpowiada czasom pojawiania się komponentów typu B na wejściu do systemu. W linii kodu 008 deklarowany jest typ komponentu. Wektor kolumnowy `Typ` jest wypełniany w taki sposób, że pierwsze `a`-wartości przyjmuje wartość równą 1 (funkcja `ones`), co będzie oznaczać komponent typu A, natomiast kolejne `b`-wartości przyjmuje wartość równą 0 (funkcja `zeros`), co będzie oznaczać komponent typu B.

W linii kodu 010 czasy wejścia `Twe`, przy wykorzystaniu funkcji `sort` są sortowane rosnąco, czego wynikiem są: wektor posortowanych czasów pojawiania się komponentów `Twe_sort` oraz wektor kolejnych indeksów z wektora `Twe` im odpowiadających `I`. Następnie w linii kodu 011 tworzony jest wektor obrazujący rodzaje komponentów w kolejności, w jakiej pojawiały się na wejściu systemu. Kolejne wartości utworzonego wektora `Typ_sort` odpowiadają wartościom z wektora `Typ` na kolejnej pozycji zapisanej w wektorze `I`.

W liniach kodu 013 oraz 014 podobne operacje przeprowadzane są na zmiennych opisujących czasy obróbki kolejnych komponentów. W linii 013 czasy obróbki komponentów typu A oraz czasy obróbki komponentów typu B są zapisywane do jednego wektora kolumnowego `Tob`, a następnie w linii 014 czasy te są zapisywane do wektora `Tob_sort` w kolejności, odpowiadającej kolejnym wartościom zapisanym w wektorze `I`.

W linii kodu 016 deklarowana jest przestrzeń robocza dla zmiennej `ostatni` obrazującej poprzedni okres, w którym pojawił się komponent tego samego typu, co w okresie analizowanym. Wypełnienie tej zmiennej dla przykładowej sekwencji komponentów postaci ABAABAABBB będzie miało ostateczną postać wektora [0, 0, 1, 3, 2, 4, 6, 5, 8, 9]. Wypełnianie zmiennej `ostatni` ma miejsce w liniach kodu od 018 do 028. W liniach kodu od 018 do 022 przy użyciu pętli `for` wypełniane są te komórki wektora `ostatni`, które odpowiadają pojawianiu się komponentów tego samego typu co pierwszy komponent w systemie. Wypełnianie następuje od komórki o indeksie `i = 2`. Od tego momentu sprawdzane jest, kiedy pojawił się poprzedni komponent tego samego typu co pierwszy komponent w systemie. Wykorzystywana jest instrukcja warunkowa `if`. Jeżeli w wektorze `Typ_sort` na `i`-tej pozycji znajduje się wartość odpowiadająca wartości pierwszej pozycji z tego wektora czyli `Typ_sort(1)`, to na `i`-tą pozycję do wektora `ostatni` zostanie przypisana wartość odpowiadająca poprzedniemu wystąpieniu tej szukanej wartości w wektorze `Typ_sort`. Wykorzystywana jest w tym celu funkcja `find`, która wypisuje ostatni element z `i-1`-pierwszych komórek wektora `Typ_sort` spełniający zadany warunek. Dla wskazanej na wstępie akapitu przykładowej sekwencji postaci ABAABAABBB kolejne trzy iteracje będą wyglądały następująco:

```
Typ_sort(1) = 0
ostatni = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
i = 2
Typ_sort(2) = 1
```

Ponieważ `Typ_sort(2)` ≠ `Typ_sort(1)` nie jest podejmowana żadna akcja.

```
i = 3
Typ_sort(3) = 0
```

Ponieważ `Typ_sort(3) = Typ_sort(1)` przy użyciu funkcji find szukamy poprzedniego wystąpienia wartości 0 wśród pierwszych dwóch wartości wektora `Typ_sort`:

```
find(Typ_sort(1:2)==Typ_sort(1))
```

Jako wynik z tak zapisanej funkcji find otrzymamy wartość 1 i to ją przypiszemy do zmiennej ostatni na pozycję `i = 3`:

```
ostatni(3) = 1
ostatni = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
i = 4
Typ_sort(4) = 0
```

Ponieważ `Typ_sort(4) = Typ_sort(1)` przy użyciu funkcji find szukamy poprzedniego wystąpienia wartości 0 wśród pierwszych trzech wartości wektora `Typ_sort`:

```
find(Typ_sort(1:3)==Typ_sort(1))
```

Jako wynik z tak zapisanej funkcji `find` otrzymamy wartości 1 oraz 3. Do zmiennej `ostatni` na pozycję `i = 4` przypisujemy ostatnią z tych wartości (mówią o tym argumenty 1 oraz `'last'` funkcji `find`).

```
ostatni(4) = 3
ostatni = [0, 0, 1, 3, 0, 0, 0, 0, 0, 0]
```

Po ostatnim uruchomieniu pętli `for` wektor `ostatni` będzie w tym wypadku przyjmował postać:

```
ostatni = [0, 0, 1, 3, 0, 4, 6, 0, 0, 0]
```

Za wypełnienie pozostałych pozycji wektora `ostatni`, odpowiadających wystąpieniu komponentów typu innego niż pierwszy, który pojawił się na wejściu systemu odpowiadają linie kodu od 023 do 028. W linii kodu 023 przy użyciu funkcji wbudowanej `find` wypisywane są pozycje jakie w wektorze `Typ_sort` zajmują dwa pierwsze komponenty innego typu niż typ komponentu, który pierwszy pojawił się na wejściu systemu. Wynik zapisywany jest do wektora dwuelementowego `pomoc`. W liniach kodu od 024 do 028 przy użyciu pętli `for` wypełniane są te komórki wektora `ostatni`, które odpowiadają pojawianiu się komponentów tego samego typu co komponent inny aniżeli pierwszy jaki pojawił się w systemie. Wypełnianie następuje od komórki o indeksie `i = pomoc(2)`. Od tego momentu sprawdzane jest, kiedy pojawił się poprzedni komponent tego samego typu co pierwszy komponent innego typu aniżeli pierwszy komponent w systemie. Wykorzystywana jest instrukcja warunkowa `if`. Jeżeli w wektorze `Typ_sort` na `i`-tej pozycji znajduje się wartość odpowiadająca wartości tego wektora na pierwszej pozycji wektora `pomoc` czyli `Typ_sort(pomoc(1))`, to na `i`-tą pozycję do wektora `ostatni` zostanie przypisana wartość odpowiadająca poprzedniemu wystąpieniu tej szukanej wartości w wektorze `Typ_sort`. Wykorzystywana jest w tym celu funkcja `find`, która wypisuje ostatni element z `i-1`-pierwszych komórek wektora `Typ_sort` spełniający zadany warunek. Dla wskazanej na wstępie akapitu przykładowej sekwencji postaci ABAABAABBB kolejne cztery iteracje będą wyglądały następująco:

```
pomoc = [2, 5]
Typ_sort(1) = 0
ostatni = [0, 0, 1, 3, 0, 4, 6, 0, 0, 0]
i = 5
Typ_sort(5) = 1
Typ_sort(pomoc(1)) = Typ_sort(2) = 1
```

Ponieważ `Typ_sort(5) = Typ_sort(2)` przy użyciu funkcji `find` szukamy poprzedniego wystąpienia wartości 1 wśród pierwszych czterech wartości wektora `Typ_sort`:

```
find(Typ_sort(1:4)==Typ_sort(2))
```

Jako wynik z tak zapisanej funkcji `find` otrzymamy wartość 2 i to ona zostanie przypisana do zmiennej `ostatni` na pozycję `i = 5`:

```
ostatni(5) = 2
ostatni = [0, 0, 1, 3, 2, 4, 6, 0, 0, 0]
i = 6
Typ_sort(6) = 0
```

Ponieważ `Typ_sort(6) ≠ Typ_sort(2)` nie jest podejmowana żadna akcja.

```
i = 7
Typ_sort(7) = 0
```

Ponieważ `Typ_sort(7) ≠ Typ_sort(2)` nie jest podejmowana żadna akcja.

```
i = 8
Typ_sort(8) = 1
```

Ponieważ `Typ_sort(8) = Typ_sort(2)` przy użyciu funkcji `find` szukamy poprzedniego wystąpienia wartości 1 wśród pierwszych siedmiu wartości wektora `Typ_sort`:

```
find(Typ_sort(1:7)==Typ_sort(2))
```

Jako wynik z tak zapisanej funkcji `find` otrzymamy wartości 2 oraz 5. Do zmiennej `ostatni` na pozycję `i = 8` przypisujemy ostatnią z tych wartości (mówią o tym argumenty `1` oraz `'last'` funkcji `find`).

```
ostatni(8) = 5
ostatni = [0, 0, 1, 3, 2, 4, 6, 5, 0, 0]
```

Po ostatnim uruchomieniu pętli `for` wektor `ostatni` będzie w tym wypadku przyjmował postać:

```
ostatni = [0, 0, 1, 3, 2, 4, 6, 5, 8, 9]
```

W liniach kodu 030 oraz 031 deklarowana jest przestrzeń robocza dla zmiennych obrazujących czas transportu komponentu od bufora do stacji montażu `THI` oraz czas transportu komponentu ze stacji montażu do wyjścia z systemu `THO`. Obie ze zmiennych to wektory kolumnowe o liczbie elementów równej `n`. W liniach kodu od 032 do 040 przy użyciu pętli `for` oraz instrukcji warunkowej `if` odpowiednie komórki zmiennych `THI` oraz `THO` są nadpisywane poprzez odpowiednie dla danego komponentu czasy transportu. Przeglądany jest na tym etapie wektor `Typ_sort`. Jeżeli wartość tego wektora na `i`-tej pozycji odpowiada wystąpieniu komponentu typu A, tj. przyjmuje wartość równą 1, to do wektora `THI` na i-tą pozycję przypisywany jest czas transportu komponentu A do stacji obróbki `THIA`, natomiast do wektora `THO` na `i`-tą pozycję przypisywany jest czas transportu komponentu A od stacji obróbki do wyjścia z systemu `THOA`. Jeżeli natomiast na `i`-tej pozycji wektora `Typ_sort` pojawi się wartość równa 0 do wektorów `THI` oraz `THO` na i-te pozycje przypisywane są wartości zmiennych `THIB` oraz `THOB`.

W linii kodu 042 rezerwowana jest przestrzeń robocza na zmienną `Tocz1b`. Jest to zmienna w postaci wektora kolumnowego, o długości `n`, którego kolejne wartości oznaczają czas o jaki będzie korygowany czas oczekiwania na podajnik po pojawieniu się na buforze nowego komponentu. W linii kodu 043 rezerwowana jest z kolei przestrzeń na zmienną oznaczającą czas oczekiwania na transport ze stacji obróbki do wyjścia z systemu `Tocz2`. Wstępnie zakładamy, że komponent nie czeka na transport ze stacji obróbki do wyjścia z systemu, dlatego też zmienna ta na początku przyjmuje postać wektora kolumnowego o długości `n`, wypełnionego jedynie zerami.

W linii kodu 045 deklarowana jest zmienna `Bezczynnosc`, która będzie odzwierciedlać przerwy w pracy podajnika. Zmienna ta deklarowana jest w postaci wektora kolumnowego o długości `2n` – każdy komponent zajmuje podajnik dwukrotnie, tj. podczas transportu z bufora do stacji obróbki, oraz ze stacji obróbki do wyjścia z systemu. Pierwszym elementem wektora `Bezczynnosc` jest wartość równa zero – pierwszy komponent jaki pojawi się na wejściu systemu nigdy nie będzie czekał na zajęcie podajnika. Pozostałe `2n-1` elementów wektora `Bezczynnosc` zostanie zapełnionych wartościami równymi `-1` (`-ones`) – wartości te w kolejnych liniach kodu będą modyfikowane aż do momentu usunięcia ostatniej ujemnej wartości.

W liniach kodu 047, 048 oraz 049 rezerwowana jest przestrzeń robocza na zmienne `Trw`, `Tzw` oraz `Twy`. Zmienna `Trw` będzie odzwierciedlać czas rozpoczęcia obróbki komponentu na stacji montażu, zmienna `Tzw` czas zakończenia tej obróbki, natomiast zmienna `Twy` czas wyjścia komponentu z systemu.

W linii kodu 051 obliczany jest czas rozpoczęcia obróbki dla pierwszego komponentu, który pojawił się na wejściu systemu. W związku z tym, że podajnik nie może być w tej sytuacji zajęty, czas ten obliczany jest jako suma pierwszej wartości z wektora posortowanych czasów pojawiania się komponentów `Twe_sort(1)` oraz czasu transportu tego komponentu od bufora do stacji montażu `THI(1)`. W linii kodu 052 obliczany jest czas zakończenia obróbki pierwszego komponentu na stacji montażu. Czas rozpoczęcia obróbki tego komponentu – `Trw(1)` – jest powiększany o pierwszą z posortowanych wartości czasów obróbki – `Tob_sort(1)`.

W linii kodu 054 rozpoczyna się wykonywanie pętli `while`. Zapisany w kolejnych liniach kodu algorytm jest powtarzany tak długo, jak w wektorze `Bezczynnosc` występują wartości ujemne.

W linii kodu 056 obliczany jest moment opuszczenia systemu przez pierwszy z komponentów – `Twy(1)`. Wartość ta stanowi sumę czasu zakończenia obróbki – `Tzw(1)` – obliczonej w linii kodu 052, czasu oczekiwania na podajnik po zakończeniu obróbki `Tocz2(1)` oraz czasu transportu komponentu przez podajnik ze stacji montażu do wyjścia z systemu `THO(1)`.

Linie kodu od 058 do 064 są wykonywane tylko w sytuacji, gdy drugi i kolejne komponenty jakie pojawiają się w systemie są tego samego typu co komponent pierwszy. W linii kodu 058 sprawdzane są wartości dwuelementowego wektora `pomoc` – jeżeli w tym wektorze wystąpi wartość równa 2 oznaczać to będzie, że drugi komponent jaki pojawił się w systemie jest innego typu niż komponent pierwszy, dlatego też kolejne linie kodu programu wykonywane będą tylko w sytuacji, gdy minimalna wartość w wektorze `pomoc` będzie większa niż 2. Jeżeli zatem spełniony jest omawiany warunek, w liniach kodu od 059 do 063 przy wykorzystaniu pętli `for` obliczane są wartości zmiennych `Trw`, `Tzw` oraz `Twy` dla kolejnych `i`-tych komponentów (pierwszym elementem, dla którego wykonywana jest pętla `for` jest element o indeksie `i = 2`, natomiast indeks ostatniego rozpatrywanego elementu odpowiada wartości minimalnej z wektora `pomoc` pomniejszonej o 1). Minimalna wartość wektora `pomoc` zapisywana jest w tym wektorze na pierwszej pozycji. W linii kodu 060 obliczany jest czas rozpoczęcia obróbki kolejnego `i`-tego komponentu `Trw(i)`. Czas transportu `i`-tego komponentu z bufora do stacji montażu – `THI(i)` – jest sumowany z większą spośród dwóch wartości: czasu opuszczenia systemu przez poprzedni komponent `Twy(i-1)` oraz posortowanego czasu wejścia analizowanego, `i`-tego komponentu `Twe_sort(i)`. W linii kodu 061 obliczany jest czas zakończenia obróbki `i`-tego komponentu `Tzw(i)`. podobnie jak w przypadku obliczeń dla pierwszego komponentu (linia kodu 052) stanowi on sumę czasu rozpoczęcia obróbki `i`-tego komponentu `Trw(i)` oraz `i`-tej posortowanej wartości czasów obróbki – `Tob_sort(i)`. W linii kodu 062 (wg schematu identycznego jak przedstawiony dla pierwszego komponentu schemat w linii kodu 056) obliczany jest czas opuszczenia systemu przez kolejny, `i`-ty komponent – `Twy(i)`.

W liniach kodu 066, 067 oraz 068 obliczane są wartości zmiennych `Trw`, `Tzw` oraz `Twy` dla pierwszego komponentu innego rodzaju niż komponent jaki pierwszy pojawił się w systemie. Kolejny rozpatrywany indeks tych zmiennych odpowiada wartości jaka znajduje się na pierwszej pozycji w wektorze `pomoc`. Wartość zmiennej obrazującej czas rozpoczęcia montażu dla tego indeksu – `Trw(pomoc(1))` – obliczana jest jako suma odpowiadających temu indeksowi: posortowanego czasu pojawiania się komponentów `Twe_sort(pomoc(1))`, czasu transportu tego komponentu od bufora do stacji montażu `THI(pomoc(1))` oraz czasu oczekiwania komponentu na podajnik po zakończeniu obróbki `Tocz1b(pomoc(1))`. Wartości zmiennych obrazujących czas zakończenia obróbki oraz czas wyjścia z systemu – `Tzw` oraz `Twy` – obliczane są wg schematu identycznego jak ten opisany wcześniej w liniach kodu 061 oraz 062.

W liniach kodu od 070 do 074 przy wykorzystaniu pętli `for` obliczane są wartości zmiennych `Trw`, `Tzw` oraz `Twy` dla pozostałych komponentów. Pierwszym rozpatrywanym komponentem jest ten, którego wartość indeksu jest równa wartości, jaka znajduje się w wektorze `pomoc` na pierwszej pozycji powiększona o 1, natomiast ostatnim komponent `n`-ty. Czas rozpoczęcia obróbki

kolejnego `i`-tego komponentu (linia kodu 071) stanowi w tym przypadku sumę: maksymalnej wartości spośród posortowanego `i`-tego czasu pojawienia się komponentu w systemie oraz czasu wyjścia z systemu poprzedniego komponentu tego samego rodzaju co komponent `i`-ty, czasu transportu `i`-tego komponentu z bufora do stacji montażu oraz czasu oczekiwania `i`-tego komponentu na podajnik po pojawieniu się w systemie. Wartości zmiennych obrazujących czas zakończenia obróbki oraz czas wyjścia z systemu – `Tzw` oraz `Twy` – obliczane są wg schematu identycznego jak ten opisany wcześniej w liniach kodu 061 oraz 062.

Kolejne linie kodu odpowiedzialne są za korektę rozwiązania i usunięcie ewentualnie nakładających się czasów transportu. W linii 076 generowana jest zmienna `ID_komponentu` w postaci wektora kolumnowego o długości `2n`, którego kolejne wartości odpowiadają dwukrotnemu powtórzeniu kolejnych `n`-liczb. Następnie w linii kodu 077 generowana jest zmienna `H_type`. Zmienna ta również przyjmuje postać wektora kolumnowego o długości `2n` i odzwierciedla rodzaj transportu – każdy komponent jest transportowany dwukrotnie, wartość 1 oznaczać będzie transport z bufora do stacji obróbki, natomiast wartość 0 oznaczać będzie transport ze stacji obróbki do wyjścia z systemu. W linii kodu 078 generowana jest zmienna `THall` w postaci macierzy o wymiarach: `2n` wierszy oraz 2 kolumny. Komórki od 1 do `n` w kolumnie pierwszej macierzy zawierają wartości oznaczające czas rozpoczęcia transportu kolejnego `i`-tego komponentu z bufora do stacji montażu (obliczany jako różnica między zmiennymi `Trw` oraz `THI`). W drugiej kolumnie w tym samym zakresie komórek zapisano czas zakończenia tego transportu – wartość ta odpowiada zmiennej `Trw` obrazującej czas rozpoczęcia montażu. Komórki od `n+1` do `2n` w kolumnie pierwszej macierzy zawierają wartości oznaczające czas rozpoczęcia transportu kolejnego `i`-tego komponentu ze stacji montażu do wyjścia z systemu (obliczany jako różnica między zmiennymi `Twy` oraz `THO`). W drugiej kolumnie w tym samym zakresie komórek zapisano z kolei czas zakończenia tego transportu – wartość ta odpowiada zmiennej `Twy` obrazującej czas opuszczenia systemu przez komponent.

W linii kodu 080 do zmiennej `THall_sort_od` zapisywane są posortowane wartości z pierwszej kolumny macierzy `THall`, natomiast do zmiennej `K` wartości indeksów im odpowiadających. Uzyskane kolejne wartości zmiennej `K` są wykorzystywane w linii kodu 081 – zmienna `THall_sort_do` zawiera wartości z drugiej kolumny macierzy `THall`, ułożone w szeregu wg wartości indeksu `K`. W linii kodu 082 wg wartości indeksu `K` szeregowana jest zmienna `ID_komponentu` – wyniki są zapisywane do zmiennej `ID_komponentu_sort`. W linii kodu 083 wg identycznego schematu szeregowana jest zmienna `H_type` – wyniki są zapisywane do zmiennej `H_type_sort`. W linii kodu 085 obliczana jest aktualna bezczynność pracy podajnika. Do zmiennej `Bezczynnosc` nadpisywane są różnice między kolejnymi posortowanymi czasami rozpoczęcia transportu (zmienna `THall_sort_od`) a poprzednimi czasami zakończenia transportu (zmienna `THall_sort_do`). Jeżeli jakakolwiek wartość zmiennej `Bezczynnosc` będzie przyjmowała wartość ujemną oznaczać to będzie zaplanowanie kilku zadań dla podajnika w tym samym czasie – wartości te będą korygowane w kolejnych iteracjach pętli `while`. W linii kodu 086 do zmiennej `kolizja` przy użyciu funkcji `find` wypisywany jest indeks pierwszej ujemnej wartości jaka znajduje się w zmiennej wektorowej `Bezczynnosc`. Następnie w liniach kodu od 088 od 092 sprawdzane jest jakiemu rodzajowi transportu odpowiada indeks zapisany do zmiennej `kolizja`. Wykorzystywana jest instrukcja warunkowa `if`. Jeżeli wartość zmiennej

`H_type_sort` dla indeksu `kolizja` przyjmuje wartość równą 0 oznacza to, że korekty wymaga czas oczekiwania na transport ze stacji obróbki do wyjścia z systemu `Tocz2`. Indeks komponentu, którego dotyczy korekta jest równy wartości jaka znajduje się w wektorze `ID_komponentu_sort` na pozycji `kolizja`. Wartość zmiennej `Tocz2` dla wskazanego indeksu komponentu `ID_komponentu_sort(kolizja)` jest nadpisywana przez różnicę wartości jaka się pod tą zmienną znajduje na wskazanej pozycji a wartością zmiennej `Bezczynnosc` na pozycji `kolizja`. Identyczny schemat dotyczy sytuacji, w której wartość zmiennej `H_type_sort` dla indeksu `kolizja` przyjmuje wartość równą 1 – zwiększany jest wtedy czas oczekiwania na transport z bufora do stacji obróbki `Tocz1b`. wykonywanie pętli `while` kończy się w linii kodu 094.

W linii kodu 096 obliczana jest jedna ze składowych zmiennych wynikowych utworzonego pliku funkcyjnego – czas oczekiwania na transport z bufora do stacji obróbki `Tocz1`, obliczany jako różnica między czasem rozpoczęcia obróbki `Trw` a posortowanym czasem pojawiania się komponentów na wejściu systemu `Twe_sort` i czasem transportu z bufora do stacji obróbki `THI`. W linii kodu 097 obliczane są czasy przebywania komponentów w systemie `Tprz` – wynik stanowi różnicę między zmiennymi `Twy` oraz `Twe_sort`. W linii kodu 099 obliczany jest całkowity czas oczekiwania na transport `Tocz_sum` – wynik stanowi sumę wartości wszystkich elementów z wektorów `Tocz1` oraz `Tocz2`. W linii kodu 100 obliczana jest sumaryczna bezczynność podajnika – do zmiennej `Bezc_sum` zapisywana jest suma wszystkich wartości z wektora `Bezczynnosc`. W linii kodu 101 do zmiennej `Harm_dl` przypisywana jest ostatnia z wartości jakie znajdują się w wektorze `Thall_sort_do` – wartość ta wyznacza długość harmonogramu. W linii kodu 102 przy wykorzystaniu funkcji `mean` obliczany jest średni czas przebywania komponentu w systemie – wynik zapisywany jest do zmiennej `Tprz_sr`. W ostatniej linii kodu, tj. w linii 104 tworzona jest ostatnia zmienna wynikowa – `Harmonogram`. Zmienna ta przyjmuje postać macierzy, której kolejne kolumny odpowiadają wektorom `Twe_sort`, `Tocz1`, `THI`, `Tob_sort`, `Tocz2`, `THO`.

3. Testowanie utworzonego modelu

Utworzony plik funkcyjny został przetestowany na przykładowym zestawie danych. Przyjęto, że czasy między przybyciami oraz czasy montażu każdego z komponentów są liczbami losowymi o rozkładzie jednostajnym i parametrach: dla komponentu A odpowiednio 210 i 390 sekund oraz 120 i 360 sekund oraz dla komponentu B odpowiednio 126 i 234 sekund oraz 138 i 318 sekund. Kolejne założenie dotyczyło czasu transportu komponentów przez podajnik z wejścia systemu na stanowisko montażowe oraz ze stanowiska montażowego do wyjścia – ustalono, że wielkości te podlegają również rozkładowi jednostajnemu, którego parametry wynoszą odpowiednio 120 i 156 sekund oraz 84 i 96 sekund. W tabeli 2 zestawiono wyniki kolejnych dziesięciu powtórzeń analizowanego eksperymentu symulacyjnego.

Tabela 2. Wyniki kolejnych dziesięciu powtórzeń analizowanego eksperymentu symulacyjnego

Zmienna wynikowa				
Kolejne powtórzenie	$\overline{T_{ocz}}$	$\overline{T_{prz}}$	T_{sum}	T_{bcz}
1	1478 s.	1921 s.	5392 s.	892 s.
2	1216 s.	1654 s.	5330 s.	788 s.
3	1423 s.	1906 s.	5798 s.	1196 s.
4	1250 s.	1714 s.	5514 s.	912 s.
5	1404 s.	1876 s.	5683 s.	1003 s.
6	1252 s.	1719 s.	5691 s.	1125 s.
7	1295 s.	1758 s.	5585 s.	1031 s.
8	1313 s.	1765 s.	5520 s.	972 s.
9	1397 s.	1859 s.	5677 s.	1105 s.
10	1429 s.	1891 s.	5837 s.	1199 s.

Źródło: [3, s. 82].

Utworzony plik funkcyjny okazał się wystarczająco stabilny oraz szybki w działaniu, co skłania do prób implementacji kolejnych modeli symulacyjnych w środowisku Matlab.

4. Podsumowanie

Wśród najczęściej stosowanych w praktyce narzędzi wspomagających procesy podejmowania decyzji związanych z identyfikacją, analizą oraz optymalizacją i planowaniem systemów logistycz-nych metody bazujące na modelowaniu symulacyjnym bez wątpienia stanowią największy udział. Symulacja komputerowa ma zastosowanie w sytuacjach, gdy dokładne zbadanie wybranego procesu czy systemu przy wykorzystaniu metod matematycznych czy narzędzi analitycznych z obszaru badań operacyjnych jest zbyt uciążliwe i skomplikowane [4]. W zależności od systemu, którego model symulacyjny ma być odwzorowany, modelujący musi dokonać wyboru odpowiedniego narzędzia wykorzystywanego do symulacji procesów. W niniejszej pracy przedstawiono model symulacyjny systemu kolejkowego utworzony w specjalistycznym pakiecie obliczeniowym Matlab. Jako główne zalety tak zaproponowanego rozwiązania należy wskazać jego wysoką elastyczność na wprowadzane przez modelującego zmiany, wysoką łatwość stosowania oraz łatwość walidacji i testowania. Rozwiązanie takie cechuje się ponadto dużą szybkością pracy oraz szerokim zakresem zastosowań. Jako wady należy wymienić długi czas potrzebny na budowę modelu oraz relatywnie długi czas, jaki musi przeznaczyć modelujący na naukę programowania. W stosunku do specjalistycznych pakietów symulacyjnych rozwiązanie takie jest jednak tańsze, co skłania do prowadzenia dalszych badań w tym zakresie.

Bibliografia

[1] Banks J.: *Software for Simulation*. Proceedings of the 1993 Winter Simulation Conference, 1993, s. 24–33.

[2] Jurczyk K., Krawczyk K., Woźniak W.: *Implementacja strategii uzupełniania zapasów w systemie przeglądu ciągłego w środowisku Matlab*. [w:] Feliks J. (red.) *Wybrane zagadnienia logistyki stosowanej*, tom 4, AGH, Kraków 2016, s. 358–376.

[3] Jurczyk K., Woźniak W.: *Metoda ustalania liczby powtórzeń eksperymentu symulacyjnego o skończonym horyzoncie czasowym*. Studies & Proceedings of Polish Association for Knowledge Management, tom 78, 2016, s. 78–87.
[4] Karkula M.: *Modelowanie i symulacja procesów logistycznych*. AGH, Kraków 2013.
[5] Noche B., Wenzel S.: *Marktspiegel Simulationstechnik in Produktion und Logistik.* Verlag TUV, 1991.
[6] Robinson S.: *Simulation: The practice of model development and use*. John Wiley & Sons Ltd, Chicester 2004.

MODELING AND SIMULATION OF QUEUING SYSTEMS IN MATLAB SOFTWARE – A CASE STUDY

Summary

The paper proposes a simulation model of a queuing system composed of five objects, which are two input buffers, two processing stations and a task executer represented as a transporting feeder for two flowitem types. The model was created as a function file written in the Matlab software. The following paragraphs of the article detail the principles of operation of the created program and the results of the numerical tests that have been carried out to investigate its stability and speediness of operation.

Keywords: programming languages, modelling and simulation, advanced technical computing environments, queuing systems

Praca realizowana w ramach grantu dziekańskiego nr 15.11.200.329.

Krzysztof Jurczyk
Wojciech Woźniak
Katedra Inżynierii Zarządzania
Wydział Zarządzania
AGH Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Ul. Gramatyka 10, 30-067 Kraków
e-mail: kjurczyk@zarz.agh.edu.pl
wojciech.wozniak.293@zarz.agh.edu.pl

Monika Klaś
Katedra Badań Operacyjnych
Wydział Zarządzania
AGH Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Ul. Gramatyka 10, 30-067 Kraków
e-mail: monika.klas@flexsim.pl