Laboratorium MATLA

Ćwiczenie 4.

Debugowanie. Efektywności kodu. Wektoryzacja.

Opracowali:

- dr inż. Beata Leśniak-Plewińska

Zakład Inżynierii Biomedycznej,
Instytut Metrologii i Inżynierii Biomedycznej,
Wydział Mechatroniki Politechniki Warszawskiej.

Warszawa, 2019

Cel ćwiczenia

Celem ćwiczenia jest nabycie przez studentów umiejętności w zakresie debuggowania kodu oraz poprawy jego efektywności poprzez wykorzystanie wektoryzacji.

Referencje

1. Andrew P. King Paul Aljabar *MATLAB Programming for Biomedical Engineers and Scientists* 1st Ed. Academic Press, 2017
2. Stuart McGarrity, MathWorks *Programming Patterns: Maximizing Code Performance by Optimizing Memory Access*

   https://www.mathworks.com/company/newsletters/articles/programming-patterns-maximizing-code-performance-by-optimizing-memory-access.html
3. Dokumentacja MATLAB'a
   a) Metody poprawy efektywności kodu w MATLAB'ie

      https://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html
   b) Jak zmierzyć efektywność kodu w MATLAB'ie?

      https://www.mathworks.com/help/matlab/matlab_prog/measure-performance-of-your-program.html

1. Napisz skrypt realizujący następujące zadania

   a) Utworzenie macierzy `A` o rozmiarze 100×100 zawierającej losowe liczby rzeczywiste z przedziału <0, 1>.
   b) Utworzenie macierzy `B` jako kopii macierzy `A`.
   c) Zmiana wartości każdego elementu macierzy `B`, którego początkowa wartość była większa niż 0.5 na 1 z użyciem zagnieżdżonych pętli `for`.
   d) Utworzenie macierzy `C` jako kopii macierzy `A`.
   e) Zmiana wartości każdego elementu macierzy `C`, którego początkowa wartość była większa niż 0.5 na 1 z użyciem zagnieżdżonych pętli `for` dla odmiennego sposobu indeksowania elementów w zagnieżdżonych pętlach `for`. Jeśli w pp. c) stosowany był porządek wierszowy, zmień go na kolumnowy lub odwrotnie.
   f) Utworzenie macierzy `D` jako kopii macierzy `A`.
   g) Zmiana wartości każdego elementu macierzy `D`, którego początkowa wartość była większa niż 0.5 na 1 z użyciem indeksowania logicznego (nie korzystaj z funkcji `find` oraz instrukcji sterujących).

   Zastosuj wbudowane funkcje timera do oceny czasu realizacji kodu w pp. c), e) i g). Jeśli to konieczne dodaj dodatkowe instrukcje umożliwiające poprawny pomiar czasu (patrz WSKAZÓWKA 2).

   Wyznacz współczynniki poprawy efektywności kodu jako stosunek czasu realizacji obliczeń w pp. e) i g) do czasu realizacji obliczeń w pp. c).

   Kod oraz uzyskane wartości czasu realizacji obliczeń dla pp. c), e) i g) i współczynników poprawy efektywności kodu wpisz w odpowiednich rubrykach *Sprawozdania*. Czy czasy obliczeń są jednakowe? Dlaczego?

   Uruchom każdą implementację dla macierzy `A` o rozmiarze NxN i N zmieniającego się od 100 do 1000 z krokiem co 100 oraz wyznacz czas realizacji każdej implementacji dla każdej wartości N. Zilustruj przebieg czasu realizacji każdej implementacji w funkcji liczby elementów macierzy `A`. Wyjaśnij przebiegi uzyskanych wykresów?

   WSKAZÓWKA 1: W celu zapoznania się z różnicami w porządku indeksowania elementów macierzy zapoznaj się z https://en.wikipedia.org/wiki/Row-_and_column-major_order.

   WSKAZÓWKA 2: W celu wyznaczenia czasu realizacji obliczeń możesz użyć funkcji timera (`tic`, `toc`). Zadbaj o zapewnienie odpowiednich warunków pomiaru czasu (np. w przypadku funkcji timera za wiarygodne przyjmuje się pomiary czasu realizacji kodu wynoszące min. 0.01 s).

2. Napisz funkcję MATLAB'a, która będzie pobierać jeden parametr wejściowy – wartość całkowitą N, i będzie zwracać macierz reprezentującą tabliczkę mnożenia dla liczb całkowitych z przedziału od 1 do N. Np. dla N=5 macierz powinna mieć postać

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

   W tym celu kolejno:

   a) Użyj pętli `for`, skup się na napisaniu przejrzystego i zrozumiałego kodu, który będzie

działać poprawnie; kod zapisz w pliku *tabmnoz_a.m* (PAMIĘTAJ, że nazwa funkcji musi być zgodna z nazwą m-pliku, w którym jest ona zapisana.).

b) Napisz drugą implementacje tej funkcji, tym razem nie korzystaj z instrukcji sterujących; kod zapisz w pliku *tabmnoz_b.m* (PAMIĘTAJ, że nazwa funkcji musi być zgodna z nazwą m-pliku, w którym jest ona zapisana.).

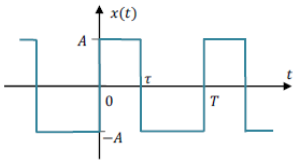
Dla N=200 wyznacz współczynnik poprawy efektywności kodu jako stosunek czasu realizacji kodu uzyskany w pp. b) dla kodu w wersji oryginalnej do czasu realizacji kodu w wersji a).

Obie wersje kodu oraz uzyskane wartości czasu realizacji obu wersji kodu i współczynnika poprawy efektywności kodu wpisz w odpowiednich rubrykach *Sprawozdania*. Czy czasy obliczeń są jednakowe? Dlaczego?

Uruchom obie implementacje dla N równego kolejno od 1 do 200 i wyznacz czasy realizacji obu implementacji dla każdej wartości N. Zilustruj przebieg czasów realizacji obu implementacji w funkcji liczby elementów macierzy reprezentującej tabliczkę mnożenia. Wyjaśnij przebiegi uzyskanych wykresów?

WSKAZÓWKA: W celu wyznaczenia czasu realizacji obliczeń, zależnie od sytuacji, możesz użyć funkcji timera (`tic`, `toc`) lub funkcji `timeit`. W przypadku wykorzystania funkcji timera zadbaj o zapewnienie odpowiednich warunków pomiaru czasu (np. w przypadku funkcji timera za wiarygodne przyjmuje się pomiary czasu realizacji kodu wynoszące min. 0.01 s).

3. Funkcja prostokątna bipolarna o okresie T może być zdefiniowana jako:



$$f(t)=\begin{cases} A & dla\ t\in(n\cdot T,\ n\cdot T+\tau) \\ -A & dla\ t\in(n\cdot T+\tau,\ n\cdot T+T) \\ 0 & dla\ t=n\cdot T,\ n\cdot T+\tau \end{cases}$$

Jeśli $\tau=T/2$, mówimy wówczas o bipolarnej fali prostokątnej o wypełnieniu ½. Jej aproksymacja za pomocą szeregu Fouriera jest określona wzorem:

$$S(t)=\frac{4\cdot A}{\pi}\sum_{k=0}^{\infty}\frac{1}{2\cdot k+1}\cdot\sin\left(\frac{2\cdot(2\cdot k+1)\cdot\pi\cdot t}{T}\right)$$

Stworzono skrypt `szereg` (plik *szereg.m*). Jego zadaniem jest iteracyjne wyznaczenie minimalnej liczby wyrazów szeregu Fouriera `nmin` niezbędnych do aproksymacji bipolarnej funkcji prostokątnej o następujących parametrach: $A = 1$, $T = 1$ wyznaczonej w przedziale czasu t = <-1,1 1,1>, dla której wartość błędu średniokwadratowego (MSE – ang. *Mean Square Error*) nie będzie większa niż 0.02.

Nawet jeśli kryterium wartości MSE nie zostanie osiągnięte, skrypt powinien przerwać swoje działanie po 100-ej iteracji. Ponadto, skrypt powinien zakończyć swoje działanie wówczas gdyby miało dojść do sumowania wyrazu odpowiadającego składowej sinusoidalnej o częstotliwości niespełniającej twierdzenia o próbkowaniu.

Po pomyślnym wyznaczeniu wartości `nmin` skrypt powinien utworzyć animację, w której we wspólnym układzie współrzędnych nałożone będą na siebie: wykres funkcji prostokątnej w funkcji czasu oraz kolejne jej aproksymacje dla *n* zmieniającego się od 1 do wyznaczonej wartości minimalnej `nmin`.

W celu realizacji obliczeń w skrypcie `szereg` zdefiniowano trzy funkcje lokalne:

- `mseErr = mseError(wartosc, estymator)`, która wyznacza wartość błędu średniokwadratowego `mseErr` – wartość oczekiwana kwadratu „błędu" czyli różnicy pomiędzy estymatorem (`estymator`), a wartością estymowaną (`wartosc`) (plik *mseError.m*) określonego wzorem

$$MSE(\theta) = E((\hat{\theta} - \theta)^2)$$

  gdzie $\hat{\theta}$ - estymator, θ - wartość estymowana.

- `prostsyg = prostokat([t1, t2], delta_t)`, która wyznacza wartości bipolarnej funkcji prostokątnej `prostsyg` o wypełnieniu ½ dla *t* z przedziału < `t1`, `t2`> z krokiem iteracji `delta_t` dla *T=1* i *A=1* (plik *prostokat.m*)

- `szeregF_new(t, delta_t, n)`, która wyznacza aproksymację bipolarnej funkcji prostokątnej o wypełnieniu ½ jako skończoną sumę wyrazów szeregu Fouriera dla *t* z przedziału < `t1`, `t2`> i *k* zmieniającego się od 0 do `n`.

Niestety kod skryptu `szereg`, w tym funkcji lokalnych, zawiera błędy.

Przeanalizuj kod zawarty w m-pliku *szereg.m*. Wyodrębnij w nim poszczególne bloki kodu odpowiedzialne za określone zadania i zaznacz je dodając odpowiednie komentarze.

Następnie, korzystając z analizatora kodu oraz debuggera (jeśli zachodzi taka potrzeba użyj tzw. brekpointów) popraw WSZYSTKIE zauważone błędy, a następnie uruchom skrypt.

UWAGA. Nie nadpisuj kodu skryptu. Kod skryptu po zmianach zapisuj pod nowa nazwą: *szereg_new.m*.

WSKAZÓWKA 1: Liczba rubryk do opisu błędów nie musi być zgodna z faktyczną liczbą błędów.

WSKAZÓWKA 2: Fragment kodu służący do obsługi poprawności wartości wprowadzonych z klawiatury przez użytkownika za pomocą funkcji `input` nie zawiera błędów i służy ilustracji zalecanego sposobu obsługi błędów.

WSKAZÓWKA 3: Dobrym obyczajem jest dodawanie w kodzie adnotacji, które ułatwiają szybkie znajdowanie fragmentów wymagającego poprawy, uzupełnienia lub zaktualizowania. W tym celu zazwyczaj stosowany jest komentarz o treści TODO. Więcej informacji na ten temat znajduje się w dokumentacji MATLAB'a, np. https://www.mathworks.com/help/matlab/matlab_prog/add-reminders-to-files.html czy https://blogs.mathworks.com/community/2008/03/17/whats-on-my-todo-list/.

W *Sprawozdaniu* zanotuj wyznaczoną wartość `nmin` oraz wymień wszystkie zauważone błędy i podaj sposób ich naprawy.

Poza błędami w kodzie, implementacja aproksymacji bipolarnej funkcji prostokątnej jako skończonej sumy wyrazów szeregu Fouriera nie jest optymalna. Popraw kod funkcji `szeregF`, tak aby czas jej realizacji był o rząd wielkości mniejszy od czasu realizacji funkcji w pierwotnej postaci. Skrypt z poprawioną funkcją `szeregF` zapisz w pliku *szereg_opt.m*.

Wyznacz współczynnik poprawy efektywności kodu jako stosunek czasu realizacji kodu

w wersji czasu realizacji kodu w wersji poprawionej („zoptymalizowanej”) do czasu realizacji kodu oryginalnej (po poprawieniu ewentualnych błędów).

WSKAZÓWKA 4. Uprość kod unikając obu pętli `for`.

WSKAZÓWKA 5: W celu wyznaczenia czasu realizacji obliczeń, zależnie od sytuacji, możesz użyć funkcji timera czy funkcji `timeit`. W przypadku wykorzystania funkcji timera zadbaj o zapewnienie odpowiednich warunków pomiaru czasu (np. w przypadku funkcji timera za wiarygodne przyjmuje się pomiary czasu realizacji kodu wynoszące min. 0.01 s).

Sprawozdanie

Ćwiczenie nr 4. Debugowanie. Efektywności kodu. Wektoryzacja.

L.p.	Imię i nazwisko	Grupa	Data

Punkt cw./ L. punktów	Realizacja/wynik	Uwagi prowadzącego
1 / 1,5	a) ..... b) ..... c) .... d) ..... e) .... f) ..... g)	

<table>
<tr><td></td><td colspan="3">Czas realizacji kodu [s]</td><td colspan="2">Współczynnik poprawy efektywności kodu</td><td></td></tr>
<tr><td></td><td>pp. c)</td><td>pp. e)</td><td>pp. g)</td><td>pp. e)</td><td>pp. g)</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="4">2 / 1</td><td colspan="5">a) .....
.....
.....
.....
.....
.....
.....</td><td></td></tr>
<tr><td colspan="5">b) .....
.....
.....
.....
.....
.....
.....</td><td></td></tr>
<tr><td colspan="3">Czas realizacji kodu [s]</td><td colspan="2" rowspan="2">Współczynnik poprawy efektywności kodu</td><td></td></tr>
<tr><td>tabmnoz_a.m</td><td colspan="2">tabmnoz_b.m</td><td></td></tr>
<tr><td></td><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td rowspan="5">3 / 2,5</td><td colspan="5">Korekta błędów w skrypcie i funkcjach lokalnych w pliku szereg.m</td><td></td></tr>
<tr><td>Nr linii kodu z błędem</td><td colspan="4">Błędy i poprawny kod</td><td></td></tr>
<tr><td></td><td colspan="4"></td><td></td></tr>
<tr><td></td><td colspan="4"></td><td></td></tr>
<tr><td></td><td colspan="4"></td><td></td></tr>
</table>

Funkcja `prostokat`	
Funkcja `mseError`	
Funkcja `szeregF`	
Wartość `nmin`	
Poprawa czasu realizacji obliczeń dla funkcji `szeregF`. Poprawiony kod funkcji.	
.....	
.....	
.....	
.....	
.....	
.....	

Czas realizacji funkcji `szeregF`		Współczynnik poprawy efektywności kodu
Wersja oryginalna	Wersja poprawiona	