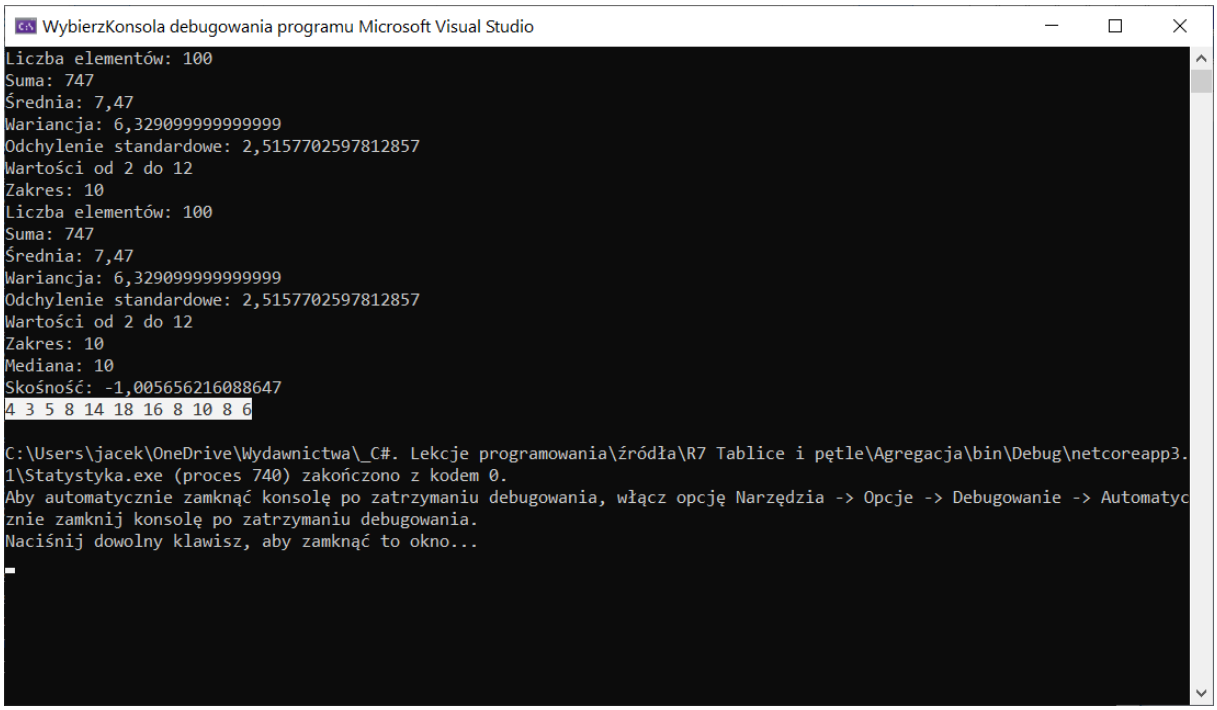
Jacek Matulewski, e-mail: *jacek@fizyka.umk.pl*
Strona wykładu: *http://www.fizyka.umk.pl/~jacek/dydaktyka/programowanie*

Uwaga! To jest kontynuacja filmu z ćwiczeniem na temat tablic i pętli

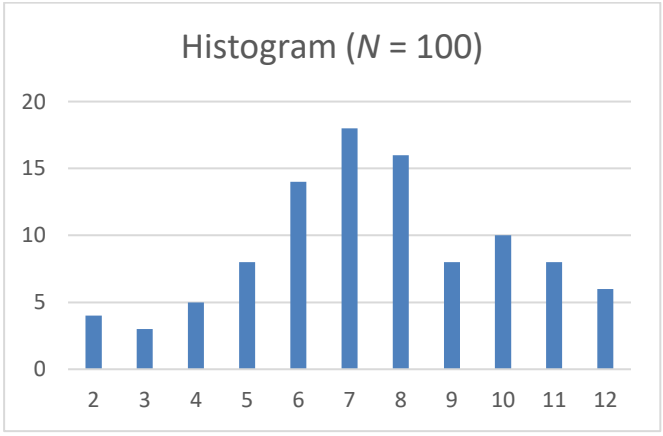
Histogram

Policzmy histogram zbioru tj. podliczmy, ile razy w tym zbiorze występują poszczególne wartości. W analizowanym przypadku, w którym możliwe wartości mają całkowite wartości z zakresu od 2 do 12, obliczanie histogramu mogłoby być tak proste, jak pokazałem w podrozdziale „Liczby losowe". My jednak postanowiliśmy skomplikować sobie życie i skopiowaliśmy te wartości do tablicy liczb rzeczywistych `double[]` i dla takiej tablicy, z założeniem że elementy zbioru mogą mieć dowolne rzeczywiste wartości, przygotowane zostały wszystkie powyższe metody. W takim przypadku zanim przystąpimy do obliczania histogramu, musimy wyznaczyć przedziały, do których będą „wpadać" wartości elementów tablicy. Dopiero wówczas przebiegamy całą tablicę i sprawdzamy, do którego przedziału należą poszczególne wartości. Liczbę przedziałów możemy arbitralnie narzucić – przekazywana jest jako argument metody `histogram` widocznej na listingu poniżej wraz z metodą `Main`, w której ją testujemy.

```
static int[] histogram(double[] wartości, int liczbaPrzedziałów)
{
    double rozmiarPrzedziału = zakres(wartości) / liczbaPrzedziałów;
    if (rozmiarPrzedziału == 0) throw new Exception("Niepoprawne dane");

    int indeksMinimum, indeksMaksimum;
    ekstrema(wartości, out indeksMinimum, out indeksMaksimum);
    double minimum = wartości[indeksMinimum];

    int[] histogram = new int[liczbaPrzedziałów];
    foreach (double wartość in wartości)
    {
        int i = (int)((wartość - minimum) / rozmiarPrzedziału);
        //skrajny punkt zakresu wkładamy do ostatniego przedziału
        if (i == liczbaPrzedziałów) i = liczbaPrzedziałów - 1;
        histogram[i]++;
    }
    return histogram;
}

static void Main(string[] args)
{
    int[] wartości = zbiórSumOczekZDwóchKostek(100);
    double[] tablica = Array.ConvertAll<int, double>(wartości, i => (double)i);

    int[] histogram = Program.histogram(tablica, 11);
    foreach (int liczbaWartości in histogram)
        Console.Write(liczbaWartości.ToString() + " ");
    Console.WriteLine();
}
```



Rysunek 7.3. Wynik obliczania histogramu dla 100 wylosowanych par liczb.



Rysunek 7.4. Histogram sum oczek dla stu rzutów parą kostek

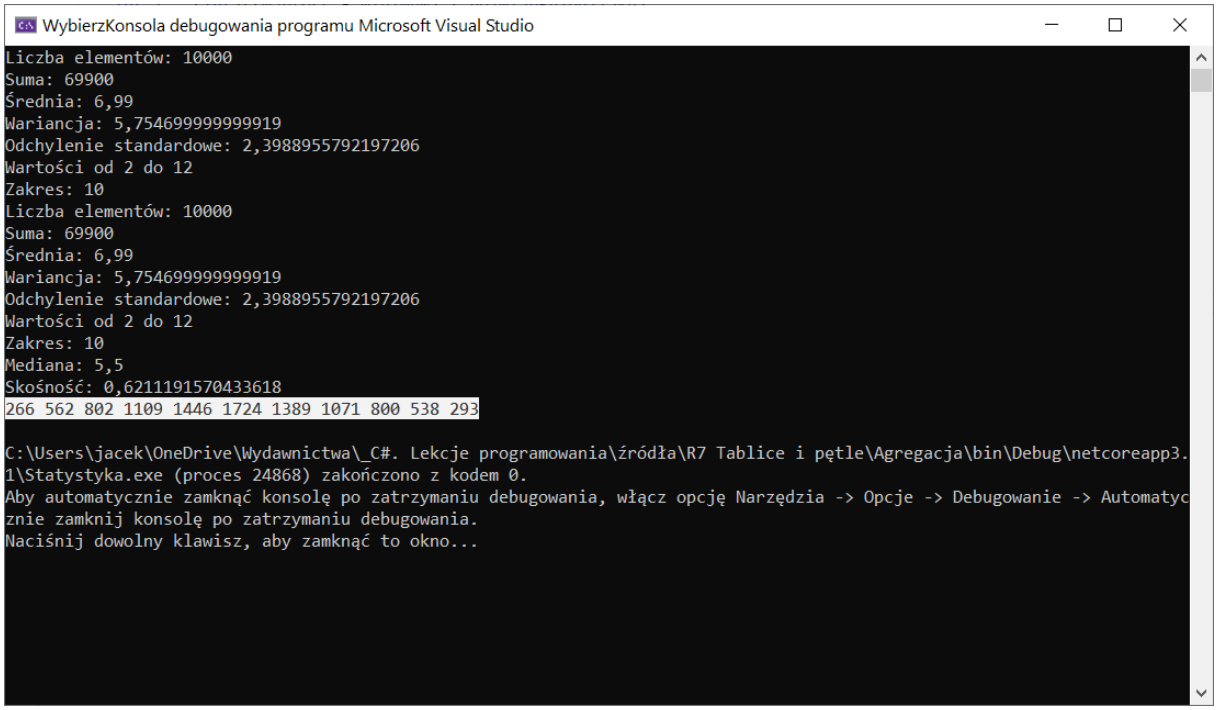
Wynik testów widoczny jest na rysunku 7.3, a wykres otrzymanych wartości na rysunku 7.4. Jak widać, wykres nie jest zbyt regularny – ewidentnie liczba rzutów jest jeszcze zbyt mała, żeby stał się gładki. Zanim zwiększymy tę liczbę, zastanówmy się najpierw jak ten wykres powinien w ogóle wyglądać. W tym celu zbierzmy wszystkie możliwe kombinacje rzutów kostką (por. tabela 7.1, na której w pierwszym rzędzie są wyniki rzutu jedną kostką, w pierwszej kolumnie – drugą, a pozostałe pola to sumy oczek z obu kostek). Widać wyraźnie, że do wartości 2 i 12 prowadzą tylko pojedyncze kombinacje oczek na obu kostkach – dwie jedynki i dwie szóstki. Do wartości 3 – dwie możliwości, a do wartości 7 – aż sześć różnych kombinacji. Liczby kombinacji dla poszczególnych wartości zebrane zostały w tabeli 7.2. Widać z niej, że skrajne wartości powinny występować rzadko, a dominantą powinna być wartość 7. Im bliżej tej wartości tym częstość występowania powinna być większa. I rzeczywiście, jeżeli zwiększymy liczbę rzutów np. do 10 000 (argument metody `zbiórSumOczekZDwóchKostek`), rozkład wylosowanych wartości wygładzi się i będzie odpowiadał przewidywaniom wynikającym z liczby kombinacji (rysunki 7.5 i 7.6).

Tab. 7.1. Możliwe kombinacje rzutu dwoma kostkami

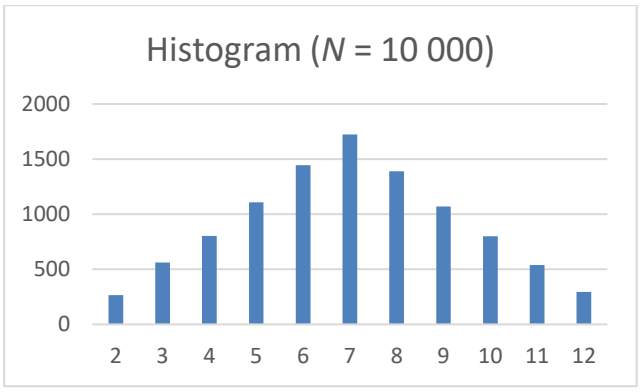
	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Tab. 7.2. Liczby kombinacji prowadzące do poszczególnych sum oczek na dwóch kostkach

Suma oczek	2	3	4	5	6	7	8	9	10	11	12
Liczba wystąpień	1	2	3	4	5	6	5	4	3	2	1



Rysunek 7.5. Histogram po zwiększeniu liczby rzutów



Rysunek 7.6. Histogram dla 10 000 rzutów dwoma kostkami