УДК 004.9+501:372.8

Гаев Е.А., Мартич М., Тарак Г.
Национальный авиационный университет, Київ, Украина

ПРОГРАММЫ МОДЕЛИРОВАНИЯ СЛУЧАЙНЫХ ЯВЛЕНИЙ ДЛЯ ИЗУЧЕНИЯ ПРОГРАММИРОВАНИЯ И МАТЕМАТИКИ

DOI: 10.14308/ite000533

Разработаны MATLAB–программы моделирования некоторых дискретных случайных событий, которые предназначены (1) как упражнения при изучении курса алгоритмизации и программирования, и (2) для проведения теоретико-вероятностных "экспериментов" лектором во время преподавания курса теории вероятностей и статистики или студентами при его самостоятельном изучении. Программа позволяет проделать тот или иной вероятностный эксперимент в необходимом количестве M, используя генератор случайных чисел, подсчитать частоту появления "благоприятных событий" и сравнить ее с теоретической вероятностью. Тем самым иллюстрируется проявление Закона больших чисел – сближение теории и эксперимента при неограниченном увеличении M. Работа, однако, не только в этом прагматическом результате. Она призывает учащихся изучать вопросы теории вероятности созданием аналогичных компьютерных кодов. Наиболее легко и быстро это делать в MATLAB-среде. Потому, работа раскрывает принципы программирования в ней и создания графического интерфейса (GUI).

Ключевые слова: *программирование, MATLAB, дискретные случайные процессы*.

Введение. При изучении современных методов алгоритмизации и программирования одной из интереснейших задач – одновременно и достаточно сложных, и увлекательных для студентов, а также привлекательных своим практическим выходом – могут быть задачи программирования основных случайных явлений, изучаемых как в средней школе, так и в соответствующем курсе высшей школы [1-3]. С одной стороны, целью этих задач программирования может быть углубленное изучение самого программирования – использование и освоение операторов ***for . . . end***, ***while . . . end***, ***if . . . else***, ***switch…case...end*** с элементами алгебры логики, создание современного графического интерфейса (GUI). С другой стороны, эти же задачи программирования позволяют самим студентам-разработчикам лучше осмыслить соответствующий теоретико-вероятностный материал и овладеть им, а также, в результате, предоставить возможность преподавателю – продемонстрировать, а студентам – опробовать подходящие „численные эксперименты” в соответствующем стандартном курсе обучения “Теория вероятностей и математическая статистика”.

Поэтому целью настоящей разработки является создание современной и красивой компьютерной программы, выполняющей такой „эксперимент” в сопоставлении с теорией.

1. Состояние вопроса

Разработка программного обеспечения, помогающего преподавателю лучше преподнести учебный материал, и студенту – эффективнее его освоить, признано одним из актуальнейших направлений компьютеризации народного хозяйства. Можно назвать несколько интересных публикаций по ряду тем [3-8]. Нам не известны, однако, какие-либо обучающие программы именно по теории вероятности, кроме описанной в [9].

Переход от математического анализа детерминированных задач к изучению студентами теории вероятностей требует от них совершенно другой *парадигмы* мышления [10]. Чтобы облегчить этот процесс, преподаватели теории вероятностей начинают с интуитивной теории, основываются на „очевидных” экспериментах с

дискретными случайными явлениями. Но физический эксперимент с, например, бросанием монеты или кубика, не может быть проведен большое количество раз, достаточное для действия закона больших чисел, подтверждающего теоретико-вероятностные выводы. Математический „эксперимент" не имеет таких ограничений. Кроме того, креативный студент получит только дальнейший стимул к личному творчеству от работы и экспериментирования с такой программой.

Результирующая MATLAB-программа оформлена в виде графического интерфейса, показанного на рис. 1. В нем несколько кнопок, запускающих ту или иную задачу. Прежде чем выдать результат в графическое окно, программа симулирует и анимирует на экране компьютера вращение шестигранной игральной кости, рис. 2, после чего выдает результаты заказанного количества экспериментов E в графическое окно. Возникающие при разработке программы вопросы разберем в разделах 4 и 5, а сейчас уточним постановку задач.

2. Элементарные вероятностные задачи

В данной работе мы рассмотрим три проблемы, касающиеся теории вероятности, обе о подбрасывании игрального кубика с количеством граней P, на которых отмечены одна, две, …, P точек (очков). В "обычном" кубике P=6; однако задача может быть обобщена по этому параметру, т.е. – рассмотрена для кубиков с произвольным количеством граней, [11].

2.1. Простейшая задача о "бросании кубика" ставится так: «Какова ***вероятность*** выпадения какой-то выделенной грани кубика (например, с одним очком)?». Эта задача исторически возникла одной из первых и ответ на нее интуитивно очевиден:

$$p = p_0 = 1/P, \qquad (1)$$

где P – количество граней кубика. И если кубик правильно центрирован, каждая грань равновероятна. Известно, однако, что реальный эксперимент с бросанием кубика дает отличающуюся ***частоту*** f появления данного "благоприятного события", равную отношению количества этих событий к количеству проведенных таких экспериментов E. И только при $E \to \infty$ имеет место $f \to p$ (знак "стремится" $\to$ понимается, таким образом, в вероятностном смысле [1, 2]).

Данная компьютерная программа заменяет физический эксперимент на математический с генератором случайных целых чисел, равномерно распределенных от 1 до P, и разработка позволяет в этом убедиться (кнопки «Start» на рис. 1 и 4). Генерируя сразу E таких случайных чисел (E экспериментов, п. 4.2), надо подсчитать количество благоприятных исходов в полученной выборке.

2.2. Следующая элементарная задача звучит таким образом: «Какова вероятность выпадения одного или другого из двух выделенных очков, т.е. ***или*** одного из них, ***или*** другого?». На основании теоремы о сумме вероятностей [1, 2], ответ очевиден:

$$p_0 = 2/P.$$

Или в более общем виде: «Какова вероятность выпадения заданного количества очков от нуля до k, $k \le N$?» Имеем ответ в следующем общем виде:

$$p_0 = k/P. \qquad (2)$$

(индекс "0" подчеркивает отношение к данным "элементарным" задачам). Проверка (2) на частных случаях: (i) если $k = 0$, т.е. никакая грань не выпадает, что невозможно, то естественно, вероятность такого события $p_0 = 0$; (ii) если $k = N$, т.е. любая грань означает "успех", то естественно, $p_0 = 1$ (достоверное событие).

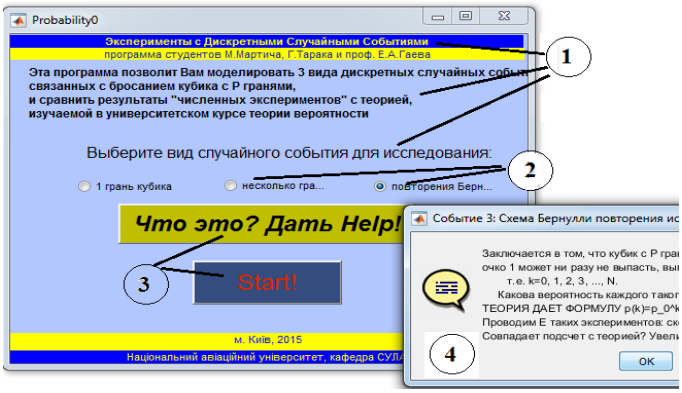

Рис.1. Начальный интерфейс программы: 1-статический текст; 2-радиокнопки, отвечающие задачам 4.2 и 4.3; 3-нажимаемые клавиши; 4-последующее, после нажатия на Help, диалоговое GUI `helpdlg`.

Программная реализация “математического аналога” соответствующего физического эксперименты несколько сложнее, и описана в п. 4.3.1.

3. Схема повторения испытаний

Следующая, уже не столь простая задача, известна как “схема Бернулли повторения испытаний” [1-3,12]: P-гранный кубик бросаем N раз; результаты и составляют теперь “единичный эксперимент”. “Удачей” называем выпадение одной определенной грани кубика, например – выпадение 1. Она может случиться 0 раз, 1 раз, …, N раз. Ставим вопрос «Какова вероятность, что “Удача” имеет место k раз, где $k = 0,1,...,N$?». Теория, как известно [1–3,12], дает формулу “биномиального распределения”

$$p(k) = \frac{N!}{k!(N-k)!} p_0^k q_0^{N-k}, \qquad (3)$$

где p_0 – вероятность элементарного события (1) и $q_0 = 1 - p_0$ вероятность того, что это элементарное событие не случится. Для каждого k из множества возможных исходов $k = 0,...,N$ получаем свою вероятность соответствующего события согласно (3). График $p(k)$ кривой (3) имеет, как правило, колоколообразный вид в зависимости от количества бросаний N, входящих в “эксперимент”, и вероятности p_0 элементарного события. Очевидно, что сумма вероятностей всех возможных событий дает 1,

$$p(0) + p(1) + ... p(N) = 1, \qquad (4)$$

поскольку [1]

$$\sum_{k=0}^{N} \frac{N!}{k!(N-k)!} p^k q^{N-k} = 1$$

(бином Ньютона).

И эту теорию (3) можно проверить, выполнив описанный эксперимент из N бросков, повторяя эксперименты много раз, записывая выпавшие очки и подсчитывая *частоту* их выпадения f. Пример: эксперимент, состоящий из $N = 2$ бросаний кубика с $P = 6$ гранями, проводим $E = 3$ раз; пусть, получили выпадение следующих граней в каждом из экспериментов: {1,5}, {6,2} и {4,1}. Значит, частота появления события «Выпала грань 1» в этой серии экспериментов была $f = \frac{2}{3}$. А частота события «Выпала грань 3» есть $f = 0$. Как эти утверждения согласуются с теоретической формулой (3)? Ведь, согласно ей, осуществление каждого из этих событий 0 раз, 1 и 2 раза есть $p(0) = 25\left(\frac{1}{6}\right)^2$, $p(1) = 10\left(\frac{1}{6}\right)^2$ и $p(2) = \left(\frac{1}{6}\right)^2$ соответственно (ниспадающий характер распределения).

Очевидно, что в таких экспериментах соответствующие подсчеты выполнить физически еще труднее, чем в предыдущих задачах. Поручим это компьютерной программе. В ней вместо физического бросания кубика снова используем генератор случайных чисел. Цель программы – сравнить выводы теории (1) – (3) в соответствующих задачах с результатами компьютерной симуляции эксперимента с бросанием игрального кубика.

4. Создание MATLAB- программы

Для разработки выбрана среда MATLAB, поскольку в ней имеется в готовом виде большое количество математических программ (команд), решающих те или иные вспомогательные задачи, например – генерацию случайных чисел. В языках C, Java и др., в отличие от MATLAB, соответствующие библиотеки надо еще подключать, и потому путь до результата дольше и труднее.

Внешний вид исходной графической программы показан на рис. 1. Запуск той или иной вероятностной задачи осуществляется выбором мышкой одной из радиокнопок 2. При этом сначала симулируется вращение “бросаемого” кубика, рис. 2, потом в виде графика выдается решение задачи. Последнее состоит, собственно, из двух частей – из представления теоретического результата (1), (2) или (3), и “экспериментальных точек” на основании подсчета частот f по окончании “эксперимента”.

Программирование состояло из нескольких этапов, которые опишем далее.

4.1. Желательно было создать на экране компьютера имитацию вращения игрального кубика перед выдачей результатов компьютерного эксперимента, рис. 2. Это было сделано с помощью трехмерной графики MATLAB. Изложим кратко это построение.

В MATLAB имеется команда ***patch***( ), ответственная за построение 3*d*-объектов с учетом затенения одной поверхностью другой. В документации ее работа и формат подготовки данных описаны не очень ясно. Дадим наше построение. Сначала в командном окне MATLAB (знак **>>** здесь и далее будет отмечать действия в командном окне; когда же алгоритм готов – переносим эти команды в *m*–файл, исполняемую программу MATLAB) указываем координаты всех 8 вершин кубика, нумерация которых представлена на рис. 3, и грани, на которых они находятся:

```
>>%Грань с вершинами 1-2-3-4:
>>P1=[0,0,0]; P2=[1,0,0];
>>P3=[1,1,0]; P4=[0,1,0];
>>%Грань 1-4-5-6:
>>P1;P4;P5=[0,1,1];P6=[0,0,1];
>>%Грань 6-7-8-5:
>>P6;P7=[1,0,1];P8=[1,1,1];P5;
>>%Грань 8-3-4-5:
>>P8;P3;P4;P5;
>>%Грань 1-2-7-6:
>>P1;P2;P7;P6;
>>%Грань 2-3-8-7:
>>P2;P3;P8;P7;
```

(знак % служит комментарием).

Теперь составляем матрицы, описывающие эти грани:

```
>>S1234=[P1;P2;P3;P4];
>>S1456=[P1;P4; P5;P6];
>>S6785=[P6;P7;P8;P5];
>>S8345=[P8;P3; P4;P5];
>>S1276=[P1;P2;P7;P6];
>>S2783=[P2;P7;P8;P3];
```

(индексация здесь и далее самоочевидна, согласно рис. 3). “Матричная арифметика” MATLAB [13] позволяет собрать в отдельный вектор-столбец *x*-, *y*- и *z*-координаты этих граней:

```
>>S1234x=S1234(:,1);        S1234y=S1234(:,2);
S1234z=S1234(:,3);
```

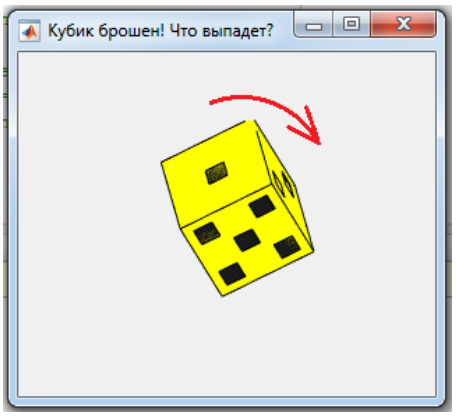

Рис. 2. Вращающийся кубик

```
>>S1456x=S1456(:,1); S1456y=S1456(:,2); S1456z=S1456(:,3);
>>S6785x=S6785(:,1); S6785y=S6785(:,2); S6785z=S6785(:,3);
>>S8345x=S8345(:,1); S8345y=S8345(:,2); S8345z=S8345(:,3);
>>S1276x=S1276(:,1); S1276y=S1276(:,2); S1276z=S1276(:,3);
>>S2783x=S2783(:,1); S2783y=S2783(:,2); S2783z=S2783(:,3);
```

Остается, согласно спецификации команды *patch*, собрать *X*-, *Y*- и *Z*–координаты всех граней кубика:

```
>>X=[S1234x, S1456x, S6785x, S8345x, S1276x, S2783x];
>>Y=[S1234y, S1456y, S6785y, S8345y, S1276y, S2783y];
>>Z=[S1234z, S1456z, S6785z, S8345z, S1276z, S2783z];
```

Каждая из этих матриц размером 4*6 представляет значение соответствующей координаты всех четырех вершин (по столбцам) и всех шести граней (строки).

Наконец, группа команд

```
>>Hp=patch(X,Y,Z,1:6), axis equal, axis square, view(3), axis off
```

Изобразит на экране трехмерный кубик с гранями разного цвета. При этом становится возможным “ручное поворачивание” кубика и просмотр его со всех сторон. Вариант команды *patch*(*X*,*Y*,*Z*,'*yellow*') раскрасит все грани в желтый цвет. Переменная `Hp` является “*хэндлом*” (указателем, *handle*) на созданный объект.

Теперь следует нанести “очки” на стороны кубика. Пусть “очком” будет черный квадратик со стороной $a=\frac{1}{9}$. Необходимо скрупулезно выписать координаты вершин всех 1+2+. . .+6=21 квадратиков на всех шести гранях. Нанесем одно очко в центре грани S5678. Тогда координаты четырех вершин квадратика будут задаваться строчками массива

```
>> d1=(1-a)/2;
R1=[d1,  d1,   1
   d1+a, d1,   1
   d1+a, d1+a, 1
   d1,   d1+a, 1];
```

(колонки – соответственно *X*-, *Y*- и *Z*-координаты). В дальнейшем координаты потребуется выделить в отдельные массивы,

```
>>R1x=R1(:,1);R1y=R1(:,2);R1z=R1(:,3);
```

Три очка на грани S1456, например, строятся несколько сложнее, но тем же алгоритмом:

```
>>d3=(1-3*a)/4;
R31=[0, d3,   d3
     0, d3+a, d3
     0, d3+a, d3+a
     0, d3,   d3+a];
R32=[0,2*d3+a,  2*d3+a
     0,2*d3+2*a,2*d3+a
     0,2*d3+2*a,2*d3+2*a
     0,2*d3+a,  2*d3+2*a];
R33=[0, 3*d3+2*a,3*d3+2*a
     0, 3*d3+3*a,3*d3+2*a
     0, 3*d3+3*a,3*d3+3*a
     0, 3*d3+2*a,3*d3+3*a];
R31x=R31(:,1);R31y=R31(:,2);
R31z=R31(:,3);
R32x=R32(:,1);R32y=R32(:,2);
R32z=R32(:,3);
R33x=R33(:,1);R33y=R33(:,2);
R33z=R33(:,3);
```

В итоге координаты всех 21 квадратиков-очков объединяем в три массива размерностью 4*21,

```
>>Xsp=[R1x,R21x,R22x,
       R31x,R32x,R33x,
       R41x,R42x,R43x,R44x,
       R51x,R52x,R53x,R54x,R55x,
       R61x,R62x,R63x,R64x,R65x,R66x];
>>Ysp=[R1y,R21y,R22y, . . .];
>>Zsp=[R1z,R21z,R22z, . . .];
```

Наконец, команда

```
>>Hsp=patch(Xsp, Ysp, Zsp,'k')
```

наносит очки на грани кубика черным цветом. Как и ранее, `Hsp` – хэндл этого объекта. Есть команда, чтобы узнать о значениях его свойств; командой

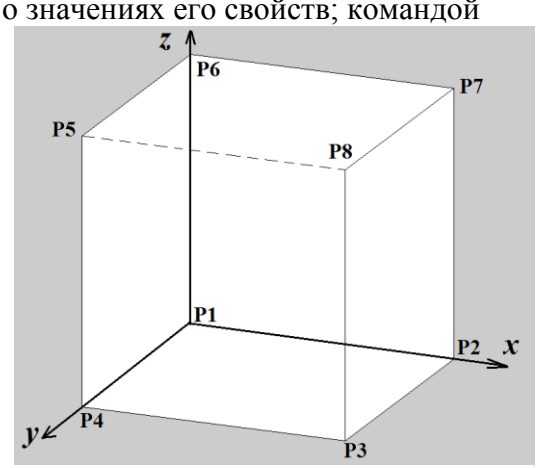

Рис. 3. Нумерация вершин кубика

```
>> set(Hsp,'FaceColor','r')
```

можно изменить, например, окраску очков на красный цвет.

Далее, MATLAB-команда `rotate(H,[ψ,φ],α)` позволяет повернуть любой объект с хендлом H на угол α вокруг оси, характеризуемой углами ψ и φ в сферической системе координат. Остается лишь организовать цикл по углу поворота α обеих объектов по образцу:

```
>>axis([-.2 1.4, -.2 1.4, -.2 1.4])
for i=1:1000
    rotate(Hp,[90,0],1)
    rotate(Hsp,[90,0],1)
    pause(.005)
end
```

Действия в командном окне, выписанные здесь, необходимо собрать в едином *m*-файле, получив, таким образом, исполняемую программу MATLAB `CubeRotation.m`.

4.2. По окончании “вращения” начинаем статистический „эксперимент”. Выберем сначала кнопку GUI №1, отвечающую задаче 2.1. Алгоритм включает в себя стандартную программу (команду) генерации случайных целых чисел равномерно распределенных от 1 до *P*, по числу граней бросаемого кубика:

```
>>P=6;E=10; Exp=randi(P,E,1)
```

Такая команда генерирует столбик `Exp` таких чисел, который и интерпретируем как результат E экспериментов, “бросаний” P–гранного кубика; в данном случае $E=10$, и отдельный эксперимент состоит из одного “бросания” этого кубика. Остается подсчитать, сколько раз выпала “***Удача***”; таковой считаем, для примера, выпадение одного очка. Операция сравнения

```
>> X=(Exp==1)
```

даст вектор из нулей и единиц; количество последних и есть количество“***Удач***” в данных E экспериментах,

```
>> S=sum(X)
```

Отсюда и получаем частоту их выпадения в проведенном эксперименте,

```
f=S/E;
```
(5)

Чтобы подсчитать частоту появления каждой грани, следует использовать цикл:

```
>> for i=1:P
       X(i)=sum(Exp==i);
   end
      f=X/E
```

P элементов результирующего вектора f и дает частоту появления каждой грани в экспериментах `Exp`. Можно проверить, что `sum(f)=1`.

Теперь сравним графически теорию (1) и „эксперимент” (5) для выпадения всех возможных очков, см. п. 7. В MATLAB несложно задать вид линии для теоретического распределения вероятности, вид значка для экспериментальных частот, а также их цвет.

Неограниченно увеличивая E, количество “экспериментов” с кубиком, убеждаемся, что $f \to p$, т.е. эксперимент сближается с теорией. Разумеется, сказанное верно лишь для кубика с равной вероятностью выпадения каждой грани. Аналогично, требуем хорошее качество внутренней MATLAB-программы псевдо-равномерной генерации случайных целых чисел.

Отметим, что использованные команды MATLAB существенно опираются на „матричную философию” последнего [13]. Перенося их в *m*–файл, получаем исполняемую программу *Problem1.m*.

4.3. Задача 2.2, “привязанная” к средней радиокнопке 2, рис. 1, даже проще в отношении программирования. Однако, здесь требуется строить более сложные логические конструкции. Как и ранее, командой

```
P=6;E=10; Exp=randi(P,E,1);
```

образуется массив *Exp* с результатами *E* экспериментов. “Успехом” будем считать, для примера, любой из трех результатов: выпадение или очка 1, или 2, или 3. Чтобы, аналогично предыдущему, получить вектор из нулей и единиц, отвечающих, соответственно, “Успеху” или “Неудаче”, выполняем команду

```
>> X=(Exp==1)|(Exp==1)|(Exp==3);
```

После этого, как в (5), имеем частоту “***Удач***” в *E* экспериментах,

```
>> f= sum(X)/E;
```

Сравнивая с $p_0 = 3/P$, согласно (2), и увеличивая E, наблюдаем $f \to p$. Распределение вероятностей здесь имеет линейный характер.

Однако, построение составного логического выражения, использованное здесь, неудобно тем, что его сложно обобщить на произвольное количество граней K.

4.4. Задача 2.2 становится более сложной, если 4.3.1 рассматривать не для фиксированного $K = 3$, а строить программу для произвольных P-гранных кубиков и всех возможных значений $K = 1,2,...,P$. Без особых разъяснений, аналогичных предыдущим, приведем окончательный листинг *m*–файла, исполнимой программы в MATLAB с комментариями:

```
function [f,p]=Problem2(P,E,Color)
%Пояснения к программе (Help)
%Генерация E случайных чисел:
Exp=randi(P,E,1);
%Подготовка массива с нулями:
X=zeros(1,P);
%Анализ "эксперимента" Exp,
%цикл в цикле:
for K=1:P
    for i=1:K
        X(K)=X(K)+sum(Exp==i);
    end
end
f=X/E; f=[0,f];%Частоты
%Теоретические вероятности:
p=(0:P)/P;
%График теор. Распределения:
figure(1)
plot(0:P,p,'r--o'), hold on
%График "эксперимент." частот:
plot(0:P,f,'Color',Color)
%Надписывание осей:
xlabel('Выпадение очков')
ylabel('Вероятность')
```

5. Обработка схемы Бернулли повторения испытаний

Отдельно рассмотрим самую сложную здесь вероятностную задачу об испытаниях Бернулли, п. 3. Команда

```
P=6;E=50;N=3; Exp=randi(P,E,N);
```

создает уже не колонку случайных чисел от 1 до *P*, а двумерный массив *Exp* с *E* строками и *N* столбцами с результатами, соответственно, *E* "экспериментов". Каждую строку можно интерпретировать как единичный "эксперимент" из *N* повторений. Логическое сравнение с единицей дает, согласно "матричной философии" MATLAB [13], матрицу такой же размерности из нулей и единиц; ее транспонирование и суммирование

```
>>Successes=sum((Exp==1)');
```

дает вектор размерности 1 на E, означающий, сколько раз "Успех", выпадение грани с одним очком, случилось в каждом эксперименте. Теперь цикл

```
>>HowMuch=zeros(N+1,1);
>>for i=0:N
HowMuch(i+1)=sum(Successes==i);
 end
```

дает вектор *HowMuch* размерности 1**N*+1 с информацией, в скольких экспериментах "Успех" имел место соответственно 0 раз, …, *N*+1 раз. Наконец, искомая частота вычисляется по п. 2.1. На результирующем графике эти данные показываем цветными значками того или иного вида и сравниваем с теоретическими значениями в виде красной штриховой кривой со значками. Для проверки $f \to p$ увеличиваем количество испытаний *E*. Проблем с вычислениями нет до $E=10^6$.

6. GUI программы

Графический интерфейс пользователя (GUI) – современный стандарт окончательного представления компьютерных программ [14]. MATLAB, как и многие другие современные средства программирования, предоставляет для этого некоторые инструменты, в данном случае – утилиту *guide*, позволяющие весьма легко создавать GUI. Сначала в специальном "разлинованном" окне разработки создается дизайн GUI, как, например, на рис. 1. Он содержит такие элементы GUI, как несколько неизменяемых (статичных) текстов (1), три радиокнопки (2) и нажимаемые клавиши (3). Затем в *m*-файле, ассоциированном с GUI, каждому GUI-элементу следует сопоставить *Callback*-программу. В данном случае для радиокнопок сделано так, что если одна из них отмечается – обе других деактивируются. Например, для третьей кнопки:

```
  global Val1 Val2 Val3
  Val3=get(hObject,'Value');
  Val2=0; Val1=0;
if (Val3==1)
  set(handles.radiobutton1,'Value',0)
  set(handles.radiobutton2,'Value',0)
end
```

Обмен данными между подпрограммами осуществляется, как видим, посредством хендлов или оператором *global*.

С нажимаемой кнопкой (3) "Дать Help!" связываем *Callback*-программу, использующую готовую МАТЛАБ-команду *helpdlg*, которая вызывает окно 4 рис.1:

```
global Val1 Val2 Val3
if Val1==1
    helpdlg('Текст','Заголовок')
end
```

С нажатием кнопки “Start!” вызывается та или иная новая (в зависимости от отмеченной радиокнопки) GUI-программа, решающая одну из названных выше задач теории вероятностей 2.1, 2.2 или 3, рис. 4. Как и предыдущая графическая программа, они создаются утилитой *guide* и сохраняются в виде двух файлов – бинарного “рисунка” с расширением ***.fig*** и программы с расширением ***.m***. (Приемом *Save as* можно вторую и третью сделать подобной первой, что обеспечивает единый стиль программного комплекса). Затем *m*-файлы новых GUI-окон дополняют *Callback*-программами. Остановимся на самых главных.

На рис. 4 показана первая из названных трех. Она содержит некоторые новые GUI-элементы, пронумерованные от 1 до 5, рис. 4. Элементы 1 получены, подобно рис. 1, как Static Text. Их свойство *String* задано через Property Inspector в виде того текста, что мы видим на интерфейсе. Нажимаемая GUI-кнопка “Помощь” вызывает тот или иной поясняющий текст с помощью `helpdlg`, как и ранее.

Два GUI-окна (2) (тип *Editable Text*) являються новыми, они позволяют вводить числовые данные. Стоящие там по умолчанию или введенные пользователем рассматриваются как *Event* (событие), для которого в *m*-файле прописывают те или иные действия. Например, первой `Callback`-функции поручаем просто прочесть содержимое как число и поместить его в *global*–память:

```
global P
P=str2double(get(hObject,'String'));
```

Два нижележащих GUI-окна (3) имеют уже тип *ListBox* – левое предлагает список значков для последующих графических точек, правое – выбор цвета для них. Содержащиеся в них списки были введены через *guide* и последующий вызов (двойным щелчком) соответствующего Property Inspector. Числовые и текстовые значения, введенные через окна (2) и (3), также обрабатываются в *m*-файле, ассоциированном с данной графической программой, и передаются в функцию, описывающие реакцию на события нажимаемой кнопки “*Start*!” (тип *PushButton*). Последняя выглядит примерно таким образом:

```
function Start_Callback
global P E Sign Color
figure(1)
CubeRotation
Problem1(P,E,Sign,Color)
```

то есть, она запускает сначала программу вращения кубика п. 4.1, и затем – программу решения той или иной вероятностной задачи, описанной в 4.2, 4.4 или 5), для значений `P` и `E`, и в графическом окне (5) строит полученные статистические результаты значками `Sign` и в цвете `Color`.

Полученный программный комплекс содержит порядка 1300 строчек МАТЛАБ-команд (включая короткие комментарии). Мы видели, что в этом коде обычным для программирования способом использовались блоки управления вычислительным процессом *if-else-end, for-end*, иногда *case-otherwise-end*.

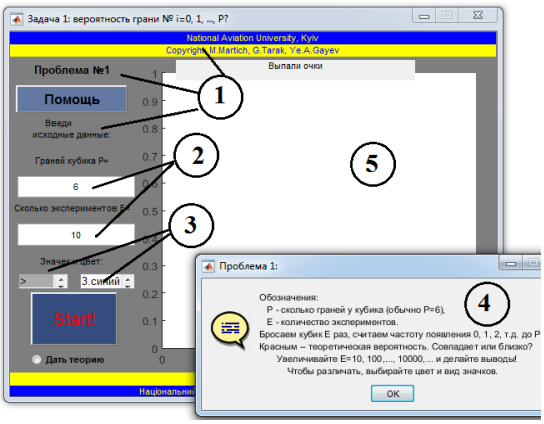

Рис. 4. Одно из очередных графических окон (GUI), запускающее соответствующую вероятностную задачу, с элементами GUI: 1 – статический текст; 2 – редактируемый текст; 3 – списки выбираемых опций, 4 – GUI-окно `helpdlg`*; 5 – подготовленное графическое окно для результатов.*

7. Исследование случайных явлений разработанной программой

Программа закончена. Рассмотрим, как она работает. Первая программа, изображенная на рис. 1, запущена из командного окна:

*>> Probability*0

В ней радиокнопки (2) позволяют выбрать одну из трех задач для последующего рассмотрения. Для выбранной задачи нажимаемая кнопка «Что это? Дать Help!» вызывает соответствующую задаче информацию (4) (для задачи № 3 в данном случае). Нажимаем кнопку «Start!» – возникает отвечающая ей новая графическая программа (для проблемы № 1 на рис. 4). Далее работаем с ними.

В каждой есть кнопка «Помощь» – она выдает в графическом окне вида (4) смысл обозначений и краткое описание сущности рассматриваемой вероятностной проблемы (№ 1 в данном случае). Вводим конкретные числовые значения количества граней кубика *P*, количества экспериментов *E* (и количество бросаний *N*, входящих в один эксперимент, задача № 3), выбираем вид и цвет значков для последующего изображения результата на графиках окна (5). Далее кнопка «Start!» запускает на короткое время вращающийся кубик рис. 2 (это должно, нам представляется, усилить впечатления пользователя), и, наконец, запускается генератор случайных чисел в форме, отвечающей задаче 4.2, 4.3 или 5, и проводится соответствующая статистическая обработка. В результате – видим частоты *f* для всех возможных значений дискретной случайной величины *k*. Хотим видеть ход теоретических кривых (1), (2) или (3) для данной задачи – отмечаем радиокнопку «Дать теорию» и получаем последнюю в виде штриховой линии красного цвета. Рассмотрим все 3 вероятностные задачи.

7.1. Задача № 1 (п. 2.1). Возьмем стандартный кубик с *P*=6 гранями (хотя, напомним, бывают и другие, [11]) и “бросим” его *E*=10 раз; подсчитаем, какова была частота появления граней 0, 1, …, *P* в этих экспериментах *E*. Такое проделано дважды, результаты помечены значками * и ◀ на графике рис. 5а. Не удивительно, что результаты этих двух реализаций не совпадают – ведь они случайны! Кроме того, эти “экспериментальные” точки, “взятые в совокупности”, весьма далеки от теоретической линии (штрих-пунктир). Проведем теперь *E*=100 “экспериментов” с тем же *P*; результат – значком ■. Можно видеть, однако, что эти точки уже ближе к теории. Наконец, еще один “эксперимент” с *E*=1000 (значек +) – результаты практически легли на теоретическую линию по формуле (1). Последняя являет собой горизонтальную линию $p(k) = 1/6 \approx 0{,}17$ (все грани равновероятны, $p_0 = 1/P = 1/6$) за исключением случая $k = 0$, т.к. грань «без очков» отсутствует.

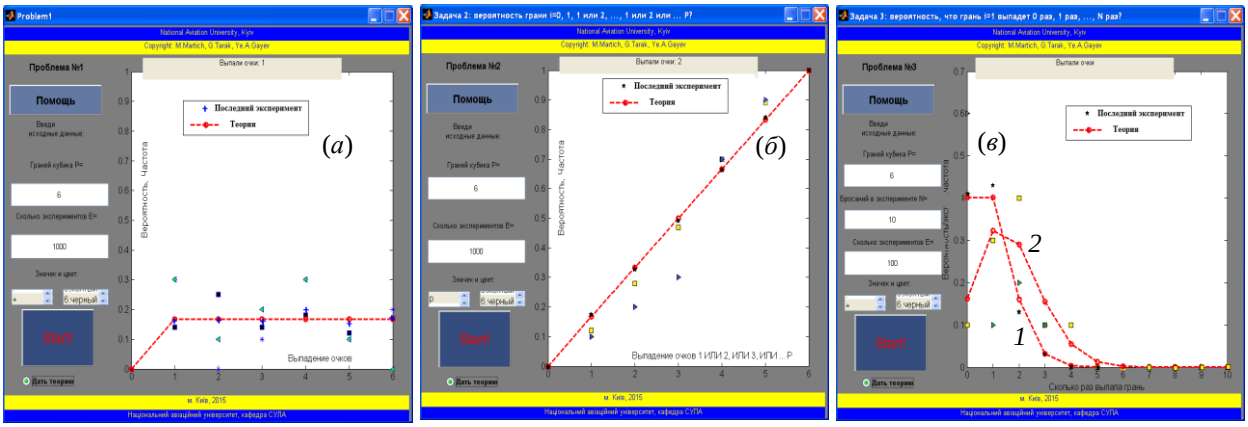

Рис. 5. Примеры решения задач программой: (А) задача № 1, (Б) задача №2 и (В) №3.

7.2. Задача № 2 (п. 2.2). Выполняем опыты с P=6 в количестве 1) E=10 (значки), 2) E=100 (значки) и 3) E=1000 ( ) – результаты на рис. 5.б. Последние точки практически легли на теоретическую линию по формуле (3). Последняя являет собой, согласно (3), диагональ.

7.3. Задача № 3 (п. 3). Здесь исследуем два случая. (*1*) Выполним повторения Бернулли с N=5 при бросании кубика с P=6 гранями. Теория дает монотонную теоретическую кривую согласно (5). Выполнили E=10 экспериментов – получали точки ; выполнили E=1000 экспериментов – точки . Они уже много ближе к теоретической кривой *1*.

(*2*) Теперь выполним с тем же кубиком повторения Бернулли с N=10 – теоретическая кривая согласно (3) стала колоколообразной. Десяти экспериментам E=10 отвечают точки ; тысяче E=1000 – точки .

“Эксперименты” по всем трем задачам демонстрируют действие закона больших чисел: при $E \to \infty$ имеет место $f \to p$.

Выводы. Таким образом, MATLAB оказался удобной средой для разработки учебного приложения, связанного с моделированием и исследованием дискретных случайных процессов. При этом были использовано несколько стандартных конструкций программирования вычислительного процесса и алгебры логики. Подпрограммы, решающие отдельные задачи моделирования, объединены в единый графический интерфейс (graphical user interface, GUI).

Запуск отдельных задач программы (через отдельные кнопки GUI) позволяет исследовать три вида случайных дискретных явлений (п. 2,3): при разных параметрах случайных событий P, p_0, N варьировать количество экспериментов E, наблюдать разброс результатов численных экспериментов вокруг теоретических значений, и тем самым – “почувствовать” проявления случайности. В то же время, такая “игра в кубики” позволяет убедиться в действии закона больших чисел: частота появления случайного события приближается к теоретическому значению, к теоретической кривой $p(k)$, при количестве “экспериментов” $E \to \infty$. Программа, описанная в данной работе, может быть использована как учителями для преподавания, так и студентами для более легкого, увлекательного и глубокого освоения материала по теории вероятности путем личного “экспериментирования” с рассмотренными случайными явлениями.

Дальнейшие вероятностные задачи, подобные [15], расширяют арсенал возможных учебных заданий при изучении программирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гнеденко Б.В., Хинчин А.Я. Элементарное введение в теорию вероятностей. М.: Наука, 1970. – 168 с.

2. Вентцель Е.С., Овчаров Л.А. Теория вероятностей и ее инженерные приложения. М.: Наука, 1988. – 480 с.
3. https://ru.wikipedia.org/wiki/. Статья "Схема Бернулли".
4. Зеленяк О.П. Стереометрія з комп'ютером? – Інформаційні технології в освіті. 2013. № 15. – С. 146 – 157.
5. Кудін А.П., Кархут В.Я. Мультимедійний навчально-методичний комплекс з вивчення теоретичної механіки. Інформаційні технології в освіті. 2013. № 15. – С. 52 – 59.
6. Бабенко М.І. Методи комп'ютерного моделювання при розв'язуванні фізичних задач в курсі фізики вищої та середньої школи. Інформаційні технології в освіті. 2013. № 17. – С. 77 -81.
7. Кулик А.С., Гавриленко Е.В. Разработка компьютерных обучающих программ на кафедре систем управления летательными аппаратами. Авиационно-космическая техника и технология, Харьков: ХАИ, № 4/111, 2014. – С. 97- 105.
8. Обруцький А.М. Фізика на Паскалі: Практикум. – Дніпропетровськ, 2006. – 224 с.
9. Азарсков В.М., Гаєв Є.О. Сучасне програмування. Модулі 1,2: "Програмування та математика із другом MATLABом". К.: НАУ, 2014. – 256 с.
10. Фройденталь Г. Математика как педагогическая задача. Ч. II. – М.: Просвещение, 1983. – 192 с.
11. https://ru.wikipedia.org/wiki/. Статья "Игральная кость"
12. http://ru.wikipedia.org/wiki/Распре-деление_Бернулли. "Распределение Бернулли".
13. Мельник И.В. Базовые принципы матричного программирования и их реализация в системе научно-технических расчетов MATLAB. – Вестник Херсонского н-т. ун-та, 2013, № 2 (47). – С. 220 – 224.
14. Патий Е. 19 ступеней вверх, или История графических пользовательских интерфейсов. "IT News", № 18/2005. http://smoking-room.ru/data/ pnp/gui_history
15. Статьи http://en.wikipedia.org/wiki/ Yahtzee, http://de.wikipedia.org/wiki/ Kniffel, https://ru.wikipedia.org/wiki. "Покер на костях".

Стаття надійшла до редакції 27.04.15

Yevgeny Gayev, Maxim Martich, Glib Tarak
National Aviation University, Kyiv, Ukraine

PROGRAMS FOR MODELLING RANDOM EVENTS FOR THE SAKE OF LEARNING BOTH PROGRAMMING AND MATHEMATICS.

MATLAB-programs of some discrete random event has been developed and intended (1) as an exercise at the study of Algorithmization and Programming Course, and (2) for carrying out some "experiments" by lecturing the Course of Probability and Statistics Theory, or at its self-study by students. The programs allows to do several probabilistic experiments in a necessary amount M, using the random number generator, to count up frequency of "favorable events" appearance and compare it to theoretical probability. This displays the Law of large numbers, i.e. approaching experimental results to theory with unlimited increase of M. The work, however, lies not only in this pragmatic result. It should encourage students to study problems of Probability Theory by means of creation appropriate computer codes. The most easy and quick way to this leads to MATLAB-environment. That is why the paper suggests principles of programming in it along with creation of graphical user interface (GUI).

Keywords: programming, MATLAB, discrete random processes.

Гаєв Є.О., Мартич М., Тарак Г.
Національний авіаційний університет, Київ, Україна

ПРОГРАМИ МОДЕЛЮВАННЯ ВИПАДКОВИХ ПОДІЙ ЗАДЛЯ ВИВЧЕННЯ ПРОГРАМУВАННЯ ТА МАТЕМАТИКИ

Розроблено MATLAB- програми моделювання деяких дискретних випадкових подій, які призначені (1) як вправи при вивченні курсу алгоритмізації і програмування, і (2) для проведення теоретико-ймовірнісних "експериментів" лектором під час викладання курсу теорії вірогідності і статистики або самими студентами шляхом самостійного вивчення

дисципліни. Програма дозволяє виконати той або інший ймовірнісний експеримент у необхідній кількості M, використовуючи генератор випадкових чисел, підрахувати частоту появи "сприятливих подій" і порівняти її з теоретичною вірогідністю. Тим самим ілюструється прояв Закону великих чисел – зближення теорії і експерименту при необмеженому збільшенні M. Робота, проте, не лише в цьому прагматичному результаті. Вона закликає учнів вивчати питання теорії вірогідності створенням аналогічних комп'ютерних кодів. Найлегше і швидко це робити в MATLAB-середовищі. Тому, стаття розкриває принципи програмування у ній і створення графічного інтерфейсу (GUI).

Ключові слова: програмування, MATLAB, дискретні випадкові процеси.