Разбор задач по Matlab

В задании было указано, что при решении задач циклами пользоваться запрещено, поэтому все решения с использованием циклов не засчитывались. За использование в решениях информации о виде входных данных, приведенных в примерах, баллы снимались.

Ниже приводятся рекомендуемые варианты решения. Во всех задачах более громоздкие, но приводящие к верному результату варианты ответов тоже засчитывались.

1. Даны две строки `x` и `y` одинаковой длины. Определить количество индексов, для которых элементы в обеих строках ненулевые. Определить количество индексов, для которых хотя бы один из двух элементов ненулевой.

   ```
   x = [1, 0, 2, 7, 0, 6, 1], y = [0, 8, 0, 3, 0, 5, 0],
   output: 2, 6
   ```

   Варианты решения первой части задания:

   ```
   % Равенство нулю проверяется  с помощью x == 0, функция isNull здесь не причем
   sum(x ~= 0 & y ~= 0) % & - поэлементное И, не путать с all()
   sum(x.*y ~= 0)
   sum(~~(x.*y))        % сравнить с sum(~~x.*y)
   ```

   Второй:

   ```
   sum(x ~= 0 | y ~= 0) % | - поэлементное ИЛИ, не путать с any()
   sum(abs(x) + abs(y) ~= 0)
   ```

   Распространенные неверные варианты:

   ```
   length(find(x + y > 0)) % не учтен вариант x + y < 0
   %find(x) находит индексы ненулевых элементов, find(x > 0) - положительных
   length(find(x+y))       % контрпример: x = -y
   sum(find(x.*y))         % либо sum(x.*y ~= 0), либо length(find(x.*y))
   ```

   Также были предложены варианты с `any()`, `all()`. Здесь они ничем помочь не могут, так как `any/all` вычисляют логическое ИЛИ/И для эдементов вектора, но не находят индексов или количества нулей в нем:

   ```
   any([0, 1, 0]) = 1, all([0, 1, 0]) = 0
   % предложенный вариант:
   all(x.*y > 0) + all(x.*y < 0) % --> 0 + 0 = 0
   ```

2. Объединить две матрицы `A` и `B` с одинаковым количеством столбцов, расположив значения первого столбца получившейся матрицы в порядке возрастания:

   ```
   A = [[1, 2, 3]; [2, 5, 6]], B = [[4, 0, 0]; [1, 3, 4]; [3, 7, 8]]
   output = [[1, 2, 3]; [1, 3, 4]; [2, 5, 6]; [3, 7, 8]; [4, 0, 0]]
   ```

   Решение:

```
C = [A; B];              % объединяем матрицы
% Сортируем первый столбец получившейся матрицы
% подавляем результат сортировки, оставляем только индексы
[~, idx] = sort(C(:,1)) % idx = [1; 4; 2; 5; 3]
C(idx,:)                % переставляем строки в нужном порядке
```

Были попытки объединить матрицы с помощью `union`. В данном случае `union` использовать нельзя, так как `union(A, B)` возвращает уникальные элементы из объединения значений `A` и `B`, а не "склеивает" матрицы, как показано в примере.

3. В вектор-строке `x` повторить каждый элемент `N` раз подряд.

```
x = [1, 7, 4], N = 3
output = [1, 1, 1, 7, 7, 7, 4, 4, 4]
```

Решение:

```
reshape(repmat(x, N, 1), 1, N*length(x))
```

либо

```
y = repmat(x, N, 1)   % y = [1, 7, 4; 1, 7, 4; 1, 7, 4]
y(:)'                 % транспонируем, чтобы получить строку
```

Решение этой задачи с использованием констант (например, 3, 9 и даже 4) оценивалось в полбалла.

4. В вектор-строке `x` найти максимальный элемент среди тех, перед которыми стоит нулевой.

```
x = [0, 1, 0, 0, 7, 8, 0, 4, 0]
output = 7
```

Распространенный, но неверный вариант решения (частично засчитывался):

```
idx = find(x == 0) + 1 % idx = [2 4 5 8 10]
max(x(idx))            % --> Index exceeds matrix dimensions: length(x) = 9
```

Решение:

```
% проверяем условие x(i) = x(i-1) для всех элементов x, начиная со второго.
% Первый элемент нас не интересует (так как перед первым ничего не стоит),
% поэтому idx(1) = false:

idx = [false, diff(x) == x(2:end)] % idx = [0 1 0 1 1 0 0 0 1 0]
max(x(idx))
```

5. В вектор-строке `x` заполнить каждый нулевой элемент предыдущим ненулевым значением.

```
x = [1, 0, 0, 7, 0, 0, 0, 4, 0]
output = [1, 1, 1, 7, 7, 7, 7, 4, 4]
```

Решение:

```
idx = find(x) % idx = [1 4 8]
y = zeros(size(x));
y(idx(2:end)) = x(idx(1:end-1)) % y = [0, 0, 0, 1, 0, 0, 0, 7, 0]
cumsum(x) - cumsum(y) % [1, 1, 1, 8, 8, 8, 8, 12, 12] - [0, 0, 0, 1, 1, 1, 1, 8, 8]
```

6. К строке '`sin(alpha) = `' добавить вычисленное для заданного `alpha` значение синуса:

```
alpha = pi/4
output = ['sin(alpha) = 0.7071']
```

Решение:

```
['sin(alpha) = ', num2str(sin(alpha))]
```

Текстовые строки – это вектор-строки символов, их можно объединять так же как и числовые векторы. Для объединения строки с числом достаточно преобразовать число в строку. Функции `printf` в матлабе нет.

7. Вычислить приближенное значение функции $f(x) = \ln(1 + x)$ в точке `x0` с помощью разложения в ряд Фурье `N`-го порядка в окрестности нуля:

$$\ln(1 + x) \approx \sum_{n=1}^{N} \frac{(-1)^{n+1} x^n}{n}.$$

Решения с циклами засчитаны не были, необходимо было воспользоваться операциями поэлементного деления и возведения в степень:

```
sum( -(repmat(-x, 1, N).^(1:N)) ./ (1:N) )
```

8. В чем разница между `x = [0:0.01*2*pi:2*pi]` и `x = linspace(0, 2*pi, 100)`?

`linspace(a, b, n)` автоматически подбирает длину интервала, разбивая отрезок `[a, b]` на `n` частей. При использовании `[a:step:b]` длина интервала `step` задается в явном виде, частота разбиения равна примерно `(b-a)/step + 1`. "Примерно", потому что `(b-a)` не обязательно делится на `step`. Например:

```
[0:0.3:1] --> [0 0.3 0.6 0.9]
```

9. Дана выборка объектов — матрица X «объекты-признаки» размера $m \times n$ (m — количество объектов, n —количество признаков). Найти выборочное среднее и ковариационную матрицу этой выборки, не пользуясь функциями `mean` и `cov`.

Решение: воспользовавшись формулой среднего значения для столбца $\mathbf{x}_j$ матрицы X

$$\bar{x}_j = \frac{1}{m} \sum_{i=1}^{m} X_{ij},$$

найдем выборочное среднее для всех столбцов матрицы X:

```
meanX = sum(X)/size(X,1); % size(X, 1) вместо length(X) на случай m < n
```

Ковариационную матрицу по выборке X можно приблизить как $X^{\circ T}X^{\circ}$, где

$$X^{\circ}_{ij} = X_{ij} - \bar{x}_j.$$

```
circX = X - repmat(meanX, size(X,1), 1);
covX = circX'*circX;
```