# Система контроля версий, тестирование

Срок сдачи – 5 мая 2013 г. 23:59

Данное задание направлено на освоение студентами навыков использования систем контроля версий, тестирования, оформления кода.

Задание:

1) Научиться работать с системой SVN. Проделать операции, описанные в разделе ниже.
2) Решить на MATLAB три задачи из список ниже. Распределение задач по студентам приведено на сайте курса. Весь написанный код должен подчиняться популярному стайл-гайду по matlab'у. Соответствие будет тщательно проверяться. Стайл-гайд можно найти здесь http://www.datatool.com/downloads/matlab_style_guidelines.pdf.
3) Провести юнит-тестирование решенных задач. Инструкции по юнит-тестированию приведены ниже.
4) Написать отчет о проделанной работе (формат PDF). Отчет должен включать в себя краткое описание решения (алгоритма) каждого из трех заданий, аргументация выбора используемых исключений и обработки различных вариантов входных данных, описание написанных юнит-тестов, описание работы с системой контроля версий.
5) Выслать отчет и заархивированный репозиторий (не рабочую папку!) преподавателю. Для экономии места в почте рекомендуется выложить архив куда-либо и прислать ссылку.

# Работа с системой SVN

1) Установить на компьютер клиент SVN. Под ОС Windows рекомендуется TortoiseSVN (http://tortoisesvn.net/).
2) Создать на компьютере локальный репозиторий. Если клиент требует выбрать тип репозитория, выбирайте fsfs. Обратите внимание, что файлы репозитория являются служебными и Вы не должны их править вручную!
3) Сделать checkout из локального репозитория в рабочую папку.
4) Задание по MATLAB и юнит-тестированию предлагается выполнять в рабочей папке.
5) Изменения в решениях разных задач должны находиться в разных коммитах. Таким образом, у Вас должно быть не менее 3 коммитов. Большее число коммитов приветствуется ☺
6) Все коммиты должны содержать краткое описание (commit message).

Краткое руководство по созданию и работе с локальным репозиторием для TortoiseSVN: http://thinkinging.com/2007/04/12/creating-a-local-subversion-repository-with-tortoisesvn/

# Задачи

В рамках данного задания требуется решить задачи, часть из которых студенты уже решали в осеннем семестре. Однако теперь акцент ставится на проверку входных данных на корректность, а также на юнит-тестирование кода.

В рамках задания необходимо выполнить по одной задаче из каждой части согласно распределению вариантов.

К решениям предъявляются следующие требования:

1) В случае если входные данные не позволяют корректно решить задачу, необходимо выдать информативное исключение. Пример проверки и вызова исключения (`repeatAllElements` – название функции, в которой вызвано исключение):

```
if ~isrow(x)
    error('repeatAllElements:xIsNotRow', 'x is not a row');
end
```

2) Обратите внимание на тип аргументов. Например, аргументы могут потенциально быть не численного типа, а, например, cell array. Полезные функции: `isnumeric, isscalar`.
3) Каждая реализованная функция должна быть покрыта юнит-тестами, см. раздел «Юнит-тестирование».
4) Каждая реализованная функция должна быть кратко задокументирована. Документация должна выводиться командой `help <functionName>`.

## Часть I. Операции без погрешностей

1) Заполнить в векторе-строке x все нулевые значения предыдущими ненулевыми значениями. Для x = [7 0 0 1 0 0 0 3 0] должен получиться ответ x = [7 7 7 1 1 1 1 3 3].
2) В векторе-строке x повторить все элементы N раз. Для x = [7 1 3] и N = 3 должен получиться ответ x = [7 7 7 1 1 1 3 3 3].
3) Найти максимальный элемент в вектор-строке среди элементов, перед которым стоит нулевой. Для A = [6,2,0,3,0,0,5,7,0] ответ 5.
4) Дана двумерная матрица A и cell array указателей на функции S. Вернуть трехмерный массив размера [size(A, 1), size(A, 2), numel(S)], содержащий вычисленные значения всех функций на всех элементах матрицы A.
5) Дана матрица A и два массива индексов I и J длины N. Построить вектор [A(I(1), J(1)),…, A(I(N), J(N))].

## Часть II. Операции с погрешностями

1) Даны две выборки объектом – X и Y. Число объектов в X – N, в Y – M. Вычислить матрицу евклидовых расстояний между объектами.
2) Даны вектор мат ожидания m и матрица ковариаций S многомерного нормального распределения, вектор-строка точек x. Вычислить значение логарифма плотности распределения во всех точках этого вектора.
3) Даны 4 точки на плоскости. Вычислить площадь четырехугольника.
4) Даны 2 пары точек на плоскости. Вычислить точку пересечения двух прямых, задаваемых парами точек.
5) Даны 3 точки на плоскости и 1 число. Вычислить точки пересечения прямой, задаваемой первой парой точек и окружности с центром в третьей точке и радиусом, заданным числом.

## Часть III. Алгоритмы машинного обучения

Пусть X - матрица объектов размера N на K, где K – количество признаков, N – количество объектов, y - вектор-столбец ответов длины N.

В каждом из вариантов требуется реализовать две функции. Первая функция по выборке строит модель машинного обучения. Модель должна быть одной переменной MATLAB. Вторая функция принимает модель и новую выборку X’ размера N’ на K и возвращает вектор-столбец предсказанных ответов y’ длины N’.

1) Матрица X и вектор y бинарные. Решается задача классификации с двумя классами. Реализовать наивный байесовский классификатор. Каждый признак считается случайной величиной Бернулли. Априорное распределение – параметр модели.
2) Матрица X вещественная, вектор y бинарный. Решается задача классификации с двумя классами. Реализовать метод k ближайших соседей по евклидовой метрике. Число соседей k – параметр модели.
3) Матрица X и вектор y вещественные. Решается задача одномерной регрессии. Реализовать линейную регрессию с регуляризацией (ridge regression). Коэффициент регуляризации t – параметр модели.
4) Матрица X вещественная, вектор y бинарный. Реализовать алгоритм классификации линейный дискриминант Фишера.
5) Матрица X вещественная, вектор y бинарный. Реализовать байесовский классификатор, восстанавливающий плотности обоих классов в виде многомерных нормальных распределений.

**Замечание.** Помимо тестов на функцию построения модели и функции предсказания ответов по модели, необходимо реализовать интеграционный тест, который проверяет, что две функции работают правильно в совокупности.

## Юнит-тестирование

К каждой из задач необходимо написать юнит-тесты с использованием фреймворка xUnit.

1) Скачать фреймворк xUnit с http://www.mathworks.com/matlabcentral/fileexchange/22846-matlab-xunit-test-framework и распаковать архив.
2) Добавить в MATLAB’е в Path папку `matlab_xunit\xunit`.
3) Если xUnit подключился, то команда runtests должна заработать, но выдать ошибку «No test cases found».
4) Название файлов с набором юнит-тестов должно начинаться с test.
5) Для запуска тестов можно либо запустить файл с тестами, либо выполнить в консоли команду runtests.
6) Каждой реализованной задаче должен соответствовать один набор тестов. В них должны быть протестированы все ветви кода, в том числе обработка ошибок.
7) Проверки разной функциональности желательно делать в разных функциях. Так вы сможете понять, что сломалось, по названию теста, который перестал проходить.
8) Перед отправкой задания проверьте, что все тесты проходят.

Пример набора тестов (файл testSin.m):

```
function test_suite = testSin
    initTestSuite;
end

function testSin45
    assertElementsAlmostEqual(sin(pi/4), sqrt(2)/2);
end
```

```
function testSinNoArgumentsFails
    f = @() sin();
    assertExceptionThrown(f, 'MATLAB:minrhs');
end
```

Полезные функции:

`assertEquals, assertElementsAlmostEqual, assertVectorsAlmostEqual, assertExceptionThrown`