LATEX BASED TOOL FOR MANAGING MULTIPLE CHOICE QUESTION EXAMS

C. Fernández, M.A. Vicente, R. Puerto, R.P. Ñeco]

[1]Miguel Hernández University, Elche (SPAIN)
{c.fernandez, suni, r.puerto, ramon.neco}@umh.es

Abstract

Evaluation through exams based on multiple choice questions has an important advantage: the results are completely objective. However, this kind of evaluation is seldom used due to its main drawback: cheating, or copying the answers from a classmate, is much easier for the students.

We present a simple software tool that manages multiple choice exams. The main idea is automating the generation of different exam versions by shuffling all question versions.

First, a Latex document must be created, with the main structure of the exam and the different versions of each question delimited with markers. Then, a simple Matlab script (easily portable to other programming languages) produces a new Latex file where the different versions of each question are combined, resulting in a huge number of exam versions. A separate Latex file is also created, in order to be used by the docent, helping him/her in the correction process.

Finally, the appropriate Latex commands are run in order to compile the files and produce two pdf outputs: one for the students (all the version of the exam to be solved) and one for the docent (which helps in the correction process).

The Matlab script, as well as usage instructions and examples are freely available from our website (http://lcsi.umh.es).

Keywords: Evaluation; Multiple choice exam; Latex; Matlab; Engineering.

1 INTRODUCTION

Evaluation is one of the most important aspects in education, and particularly in higher education. Among the different requirements that an evaluation methodology must fulfil, we can consider:

- Variety of contents that are evaluated. Ideally, an evaluation methodology should cover all skills and contents of the subject under evaluation (theoretical foundations, practical skills, etc.). Such ideal methodology is not easy to implement, due to different limitations.
- Objectiveness. The evaluation must be as objective as possible, leaving the docents subjective opinion apart. Multiple choice exams are clearly the best option concerning this requirement, although some skills are difficult to evaluate with this kind of exams.
- Reliability or robustness to cheating. Depending on the countries and the culture, cheating (usually in the form of copying the others results) is common among students. An evaluation methodology should take this into account.
- Feasibility in terms of workload for the students. Continuous evaluation is one of the best options for measuring the skills acquired by the students, but such a methodology is not always feasible due to time requirements: the students are usually enrolled in a large number of different subjects per semester, and the accumulation of partial tests and assignments may result in an excessive workload for them.
- Feasibility in terms of workload for the docent. Complex evaluation strategies may result in an excessive workload for the docent, who may get overwhelmed, particularly in Universities where docents are responsible of several subjects per semester and the number of students enrolled is high.
- Feasibility in terms of space requirements. Ideally, during the exam, the students should be placed far enough from each other in order to avoid copying the others results. Depending on the number

of students enrolled in a certain subject, the required space requirements may not be available in certain cases.

Taking into account all the aspects above, multiple choice exams can be considered as a good tool in an evaluation methodology: basically, they are highly objective, they are feasible in terms of workload for students and docent, and they can cover most contents of the subject. Ideally, they should be complemented with other methodologies in order to evaluate practical skills.

However, multiple choice exams have an important drawback: either they are prone to cheating (students can copy the others result) or they are not feasible in terms of space requirements (if students are placed enough apart from each other so that they cannot copy and the number of students is high, the space required is usually not available).

2 EXAMS WITH MULTIPLE VERSIONS

In order to avoid cheating, a common strategy is to produce different versions of the exam, so that students sitting close to each other during the test cannot share the results. Such a strategy, though effective, is not easy to manage: every different version of the exam requires a new document (in whatever the text processing software used) and increases the workload of the docent.

A common scenario may be an exam with e.g. three different questions and four different versions for each question. The number of different exams that could be generated equals 4x4x4 = 64. However, no docent will generate 64 different Word or Latex documents. Usually, only four different documents are generated: document #1 with version 1 of all questions; document #2 with version 2 of all questions; and so on. The result is that, even though copying from a classmate becomes more difficult, there is a high probability of one student sitting close to another student with exactly the same exam.

In order to assure that students sitting close to each other do not have the same exam, more documents must be produced. Manually generating so many documents is extremely time-consuming and can easily lead to errors; we propose the automation of such task.

3 PROPOSED STRATEGY

We present a simple software tool that manages multiple choice exams. The main idea is automating the generation of different exam versions by shuffling all question versions. Our tool is based on Latex text processing software [1], so the first step is to create an initial version of the exam in Latex. Once this initial version is created, the docent may introduce different versions of each question by including markers in the Latex file (in scientific or engineering degrees, the simplest versioning strategy is just to modify the data of the problem, leaving the problem structure untouched).

The next step is using a simple Matlab [2] script (easily portable to other programming languages) which produces a new Latex file where the different versions of each question are combined, resulting in a huge number of exam versions. As an example, a 4 questions test where every question has 4 different versions may offer 256 different exams. All exam versions are pages of a single document, so there is no extra management workload for the docent. Besides, a separate Latex file is also created, in order to be used by the docent, helping him/her in the correction process.

Finally, the appropriate Latex commands are run in order to compile the files and produce two pdf outputs: one for the students (all the version of the exam to be solved) and one for the docent (which helps in the correction process).

4 DETAILS OF USAGE

4.1 Generation of the initial Latex document

The first step for creating the exam is the generation of an initial Latex document. Most of the document is common for all exam versions; no extra care is needed in this case. For those parts of the document we want to vary between versions, a tag must be added, showing the number version. Though the tag is customizable, Figure 1 shows an example where %v1, %v2, %v3 and %v4 have been used as tags for versions 1 to 4 of the exam question. In order to improve clarity, a bold font has been used for all the Latex lines with tags.

```
\centerline{\textbf{PROBLEMA 1 (4 puntos).}}

\vspace{0.25cm}

%v1 En la ecuación siguiente, x(t) es la entrada; y(t) es la salida y el punto de funcionamiento está definido por x(0-) = 2:

%v2 En la ecuación siguiente, x(t) es la entrada; y(t) es la salida y el punto de funcionamiento está definido por x(0-) = 3:

%v3 En la ecuación siguiente, x(t) es la entrada; y(t) es la salida y el punto de funcionamiento está definido por x(0-) = 1:

%v4 En la ecuación siguiente, x(t) es la entrada; y(t) es la salida y el punto de funcionamiento está definido por x(0-) = 4:


\begin{displaymath}

%v1 5 \frac{dy(t)}{dt} y^{2}(t) + 6y(t) = 7 \frac{dx(t)}{dt} + 3x^{3}(t) + 12

%v2 7 \frac{dy(t)}{dt} y^{2}(t) + 4y(t) = 5 \frac{dx(t)}{dt} + 2x^{2}(t) + 2

%v3 4 \frac{dy(t)}{dt} y^{2}(t) + 3y(t) = 3 \frac{dx(t)}{dt} + 2x^{3}(t) + 7

%v4 3 \frac{dy(t)}{dt} y^{2}(t) + 5y(t) = 4 \frac{dx(t)}{dt} + 2x^{2}(t) + 3

\end{displaymath}


Se pide:

\begin{enumerate}

  \item Calcular el valor de la señal y(t) en el punto de funcionamiento.

  \item Obtener la función de transferencia que relaciona la salida con la entrada.

%v1  \item Calcular el valor de la salida y(t) en el instante t=5 si la entrada es un escalón de amplitud 5.

%v2  \item Calcular el valor de la salida y(t) en el instante t=2 si la entrada es un escalón de amplitud 4.

%v3  \item Calcular el valor de la salida y(t) en el instante t=3 si la entrada es un escalón de amplitud 6.

%v4  \item Calcular el valor de la salida y(t) en el instante t=6 si la entrada es un escalón de amplitud 2.

  \item Ante la misma entrada, calcular el valor de y(t) en régimen permanente.

\end{enumerate}


Marca una de las 4 soluciones propuestas o N.A. (ninguna de las anteriores):

\begin{table}[h]

…

\end{table}
```

Figure 1: Example of exam question with multiple versions (table of answers is omitted due to space limitations).

The example of Figure 1 corresponds to one exam question of a systems theory test (obtaining a transfer function and a step response from a set of differential equations and a set-point). Each exam version has a specific set of differential equations, a specific data for the set-point, a specific value for the amplitude of step input and a specific time instant when the output value is asked for. Thus, versions are completely different and copying is almost impossible. Figure 2 shows the pdf result obtained for the first version of the exam (tags marked as %v1).

PROBLEMA 1 (4 puntos).

En la ecuación siguiente, x(t) es la entrada; y(t) es la salida y el punto de funcionamiento está definido por x(0-) = 2:

$$5\frac{dy(t)}{dt}y^2(t) + 6y(t) = 7\frac{dx(t)}{dt} + 3x^3(t) + 12$$

Se pide:

1. Calcular el valor de la señal y(t) en el punto de funcionamiento.
2. Obtener la función de transferencia que relaciona la salida con la entrada.
3. Calcular el valor de la salida y(t) en el instante t=5 si la entrada es un escalón de amplitud 5.
4. Ante la misma entrada, calcular el valor de y(t) en régimen permanente.

Marca una de las 4 soluciones propuestas o N.A. (ninguna de las anteriores):

1	y(0-)=6	y(0-)=5	y(0-)=3	y(0-)=7	*N.A.*
2	$\frac{5s+12}{175s+4}$	$\frac{3s+6}{36s+3}$	$\frac{4s+16}{147s+5}$	$\frac{7s+36}{180s+6}$	*N.A.*
3	y(t)=6.04	y(t)=8.23	y(t)=10.77	y(t)=5.65	*N.A.*
4	$y(\infty) = 36$	$y(\infty) = 17$	$y(\infty) = 15$	$y(\infty) = 13{,}4$	*N.A.*

Figure 2: PDF output of the Latex code of figure 1 (version #1 of a total of 4 versions).

4.2 Generating all exam versions

Once the initial Latex document has been created, two Matlab functions must be run in order to generate all exam versions:

- The first function (named *versiones*) creates a new Latex file with all possible version combination (e.g. 256 different exams in a test with 4 questions and 4 versions per question). Every exam is delimited with and end of page marker, so they can be printed in different pages and given to different students.
- The second function (named *corregir*) creates another Latex file where every page corresponds to a question version. Such document is useful for the docent in order to make correction easier.

Both functions take as arguments the filenames of the initial and final Latex documents; and return as output matrixes showing the version ordering used (more details will be given in next sections).

In order to make our code easier to use, we have created a main script which calls both functions. Such script is shown in figure 3.

```
% genera versiones de examen a partir de un documento latex inicial
% genera por una parte el fichero para corregir (una pregunta por página,
% pregunta por pregunta, con todas las versiones de cada pregunta)
% y por otra el fichero para los alumnos (todas las preguntas en cada
% versión, tantas versiones como el producto de las versiones de cada
% pregunta).

% ejemplo de uso:

% creamos el fichero para los alumnos
[v1, v2] = versiones('examen.tex', 'documento_alumnos.tex');

% creamos el fichero para corregir
[v3, v4] = corregir('examen.tex', 'documento_profesor.tex');

% En las variables v1, v2, v3, v4 quedan los indices utilizados para las
% versiones (se pueden consultar para comprobar que todo ha ido bien)

% Una vez generados los ficheros, hay que compilarlos con Latex para
% obtener el pdf definitivo.
```

Figure 3: Main Matlab script which generates both the document for the docent and the document with all exam versions for the students.

In order to make our code extensible to different exam styles, the user can configure the kind of marker desired for the beginning of each problem and the kind of marker used for the version tags. Should any of these markers be changed, they must be specified in the configuration part of the code. Such configuration part is shown in Figure 4. The data to introduce includes:

- The marker used at the beginning of each problem. This information is needed in order to detect which lines correspond to each question and thus how many versions should be produced using these lines.
- The number of versions per question.
- The label used for version tags. This information is also strictly necessary. Every line starting with the tag will be considered different for each version. All remaining lines will be copied without any selection (they are considered to be equal for all exam versions).

```
% etiqueta para cada pregunta:
etiq{1} = '\centerline{\textbf{PROBLEMA 1';
etiq{2} = '\centerline{\textbf{PROBLEMA 2';
etiq{3} = '\centerline{\textbf{PROBLEMA 3';

% numero de versiones para cada pregunta:
num(1) = 4;
num(2) = 8;
num(3) = 4;
numpreg = length(num);

% etiqueta para las versiones:
ver = '%v';
```

Figure 4: Configurable part of the code: labels used and number of versions per question.

4.3 Version ordering

As it has been said in the previous section, the document generated for the students is a single pdf file with the different exam versions separated with end of page markers. Usually, exams are given to the students in order (depending on the structure of the classroom the order may start with the rows or with the columns), so that two consecutive students are given consecutive documents. Our code tries to maximize the difference between consecutive exams: the goal is that two consecutive exams do not share any question.

In order to achieve such behaviour, we use a matrix that represents, for every exam, the versions selected for each question. Initially, this matrix is created in increasing order, using the code of figure 5. The result is shown in figure 6 (in an example with 4 questions and 4 versions each). It is clear that two consecutive exams share most of the questions.

```
% numero de versiones de examen a generar
numver = prod(num);

% version de cada pregunta para cada version del examen
% buscamos el siguiente comportamiento: por ejemplo, si hay dos preguntas,
% la primera con 3 versiones y la segunda con dos:
% 1  1
% 1  2
% 2  1
% 2  2
% 3  1
% 3  2
for i=1:numpreg
    if i==numpreg
        repet = 1;
    else
        repet = prod(num(i+1:end));
    end
    indice = 1;
    for j=1:numver
        version_ini(i,j)=indice;
        if mod(j,repet)==0
            indice = indice+1;
        end
        if indice>num(i)
            indice = 1;
        end
    end
end

% ahora se reordenan para evitar que dos examenes
% contiguos se diferencien lo máximo posible
version = reordena(version_ini);
```

Figure 5: Generation of the different exam versions without reordering.

```
NON-REORDERED EXAMS
Exam: (001) (002) (003) (004) (005) (006) (007) (008) (009) (010) (011) (012) (013) (014) (015) ...
Q1:     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1   ...
Q2:     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1   ...
Q3:     1     1     1     1     2     2     2     2     3     3     3     3     4     4     4   ...
Q4:     1     2     3     4     1     2     3     4     1     2     3     4     1     2     3   ...
```

Figure 6: Non-reordered exams: consecutive exams share questions.

As this behaviour is not desired, a further reordering is carried out, trying to maximize the difference between two consecutive exams. The code in charge of this task is shown in Figure 7; while the resulting order is shown in Figure 8. The code simply selects, for the next exam, a combination of questions not used in the previous one. Ideally, all questions should be different; if this is not possible, the code searches for the maximum possible number of different questions.

```
function new_ver = reordena(vers)

% se traspone la matriz
vers = vers';

% x= numero versiones, y = numero preguntas
[x, y] = size(vers);

% versiones utilizadas
usado(1:x) = 0;

% primera pregunta
indice(1) = 1;
usado(1) = 1;

% resto de preguntas
for i=2:x

    max_dif = 0;
    for j=1:x
        diferencia = sum(vers(j,:)~=vers(indice(i-1),:));
        if usado(j)==0 && diferencia>max_dif
            max_dif = diferencia;
            max_ind = j;
        end
    end
    indice(i) = max_ind;
    usado(indice(i))=1;

end

for i=1:x
    new_ver(i,:) = vers(indice(i),:);
end

% se traspone el resultado
new_ver = new_ver';

return
```

Figure 7: Matlab code for version reordering.

```
REORDERED EXAMS
Exam: (001) (002) (003) (004) (005) (006) (007) (008) (009) (010) (011) (012) (013) (014) (015) ...
Q1:     1     2     1     2     1     2     1     2     1     2     1     2     1     2     1   ...
Q2:     1     2     1     2     1     2     1     2     1     2     1     2     1     2     1   ...
Q3:     1     2     1     2     1     2     1     3     2     1     2     1     2     1     2   ...
Q4:     1     2     3     1     2     3     4     1     2     1     3     2     1     3     4   ...
```

Figure 8: Results after reordering: consecutive exams do not share any questions.

4.4 Availability and portability

The Matlab scripts, as well as usage instructions and example exams are freely available from our website [3]. Porting the code to other programming languages is straightforward; in particular, the scripts run perfectly (without the need of any changes) under Octave [4], the most popular GNU alternative to Matlab.

5 CONCLUSIONS

Multiple choice exams have important advantages over other evaluation methodologies, the most obvious one being the objectiveness of the results. On the other side, copying the results form a classmate is easy for the students, so extra care is needed in order to avoid cheating.

When there are space limitations and a large number of students (so that they inevitably sit close to each other), the only way to ensure the privacy of the results of each student is the use of different exam versions. However, managing multiple versions is cumbersome and prone to errors, unless a specific tool is used.

We propose a tool based on Latex and Matlab that generates multiple versions of an exam in a single document, so that there is not extra workload for the docent and errors are avoided. Besides, the ordering of the questions is arranged so that two consecutive students have exams as different as possible.

The tool we have developed has proved to work reliably under Matlab and Octave platforms, and can be easily ported to other programming languages.

REFERENCES

[1] Latex text processing software. http://www.latex-project.org.

[2] Matlab, software for technical computing. http://www.mathworks.com/products/matlab/.

[3] LCSI website (*Laboratorio de Control y Sistemas Inteligentes*, Control and Intelligent Systems Lab). http://lcsi.umh.es/teaching/herramientas.

[4] Octave, GNU software for technical computing. http://www.gnu.org/software/octave.