Vol 7. No. 2 - June, 2014

African Journal of Computing & ICT

www.ajocict.net

Computationally Efficient Algorithm for computing Cumulative Grade Point Average of a Large Number of Students

R.O. Akinola
Department of Mathematics
Faculty of Natural Sciences
University of Jos
Jos, Plateau State, Nigeria

Computational Biology Group
Faculty of Health Science
University of Cape Town, Cape Town, South Africa.
Corresponding email: *roakinola@yahoo.com*

ABSTRACT
One major problem faced by Level Coordinators at the University of Jos is the computation of their students' Cumulative Grade Point Average (CGPA) in senate format at the end of each session. To many a lecturer, it is laborious, time wasting and a frustrating exercise. This paper proposes a python based algorithm for computing each students CGPA for an entire level in a fast and less tasking manner. It surpasses an earlier algorithm proposed by the same author which uses octave and has the capability to compute the TCE, TGP, TGR, GPA, their cumulatives as well as state which courses are to be repeated or if a student passed (i.e., no carry over) by default. The algorithm can also state each student's class of degree which makes it more powerful.

Keywords: Computation, Algorithms, Cumulative Grade Point Average, Students & Results.

African Journal of Computing & ICT Reference Format:
R.O. Akinola (2014). Computationally Efficient Algorithm for computing Cumulative Grade Point Average of a Large Number of Students. Afr J. of Comp & ICTs. Vol 7, No. 2. Pp27-32.

1. INTRODUCTION

Level coordinators at the University of Jos have this seemingly daunting task of computing the CGPA of their students. At the University of Jos, there is an ICT directorate which 'organises' workshops on how to generate this senate format but due to heavy workload, many a lecturer do not attend such workshops. Coupled with the fact that it demands that each lecturer enters the individual scores of each student in all courses into an already prepared template under 'pressure' or undue supervision. Most lecturers find this herculean. In addition, the problem of power outages and possibility of systems crashing makes that option a no-go-area; and that is why lecturers who are assigned as level coordinators prefer to prepare their senate format themselves. Of course, this is at a cost, discouraging and painful.

Often times mistakes are made in computing students CGPA and it takes a lot of effort and administrative bureaucracies to get them corrected once results have been presented to senate. When we talk about the senate format, we mean a summary table showing the number of courses registered by each student in a particular level, the mark, corresponding grade point obtained in each course, the Total Credit Earned (TCE), Total Credit Registered (TCR), the Total Grade Point (TGP), Gross Point Average (GPA).

Their respective cumulatives for example : Cumulative Total Credit Earned (CTCE), Cumulative Grade Point Average (CGPA) as well as a remark column which states if a student passed (without carry overs) or failed (with carry over and which course(s) to be repeated). This table is normally drawn by each level coordinator and presented to the university senate at the end of each successive academic session. However, one of the inherent problems is that to compile the senate format comprises of getting the list of all the students in a particular level of a department, knowing the number of courses each student registered, ability to collate the results from the various departments and recording them accordingly in a table as shown below (Table 1):

Table 1: Table containing all the known quantities for each student.

UNIVERSITY OF JOS
FACULTY OF NATURAL SCIENCES
DEPARTMENT OF MATHEMATICS
100 LEVEL B.Sc. MATHEMATICS SENATE FORMAT RESULTS 2010/2011 SESSION

S/NO	MAT. NO	NAME	ME	MNSA	NSS	MTH	GP	MTH	GP	MTH	GP	CS	GP	CS	GP	STA	GP	STA	GP	STA	GP	PHY	GP	GST	GP	GST	GP	GST	GP	GST	GP
						101		102		103		101		102		111		131		203		101		101		102		103		104	
						3		3		3		2		3		4		4		4		3		4		2		2		2	
1	UJ/20Y/NS/X1	SI	UME	12	2	74	5	69	4	63	4	71	5	66	4	71	5			70	5			59	3	76	5	62	4	63	4
2	UJ/20Y/NS/X2	YMY	UME	12	2	75	5	68	4	70	5	80	5	72	5	94	5			67	4			71	5	81	5	76	5	65	4
3	UJ/20Y/NS/X3	IAP	UME	12	2	46	2	72	5	60	4	73	5	77	5	76	5	63	4	68	4			80	5	77	5	62	4	60	4
4	UJ/20Y/NS/X4	RSA	UME	12	2	44	1	50	3	44	2	73	5	63	4	60	4			40	1			75	5	68	4	72	5	61	4
5	UJ/20Y/NS/X5	KAP	UME	12	2	45	2	40	1	43	1	71	5	43	1			29	0	35	0			45	2	60	4	43	1	42	1
6	UJ/20Y/NS/X6	GJB	UME	12	2	46	2	65	4	40	1	76	5	55	3					41	1	41	1	61	4	78	5	62	4	69	4
7	UJ/20Y/NS/X7	GJR	UME	12	2	6	0	1	0	2	0	46	2	19	0					13	0			65	4	78	5	58	3	53	3
8	UJ/20Y/NS/X8	DYA	UME	12	2	43	1	22	0	40	1	47	2	22	0					22	0			66	4	45	2	46	2	51	3
9	UJ/20Y/NS/X9	JSA	UME	12	2	8	0	2	0	14	0	17	0	11	0					8	0	40	1	47	2	31	0	40	1	47	2
10	UJ/20Y/NS/X11	HMI	UME	12	2	41	1	41	1	46	2	57	3	66	4	43	1			41	1			70	5	83	5	50	3	56	3
11	UJ/20Y/NS/X12	DE	UME	12	2	41	1	45	2	18	0	40	1	56	3			56	3	42	1			61	4	66	4	60	4	57	3
13	UJ/20Y/NS/X13	OOC	UME	12	2	5	0	3	0	5	0	49	2	43	1					10	0	9	0	58	3	43	1	ABS	0	51	3
14	UJ/20Y/NS/X14	IAT	UME	12	2	9	0	42	1	41	1	46	2	45	2					35	0	18	0	71	5	42	1	52	3	52	3
15	UJ/20Y/NS/X15	GJO	REM	12	2	6	0	1	0	3	0	80	5	21	0			15	0	8	0			59	3	52	3	45	2	48	2
16	UJ/20Y/NS/X16	EG	REM	12	2	43	1	44	1	43	1	70	5	62	4	21	0	56	3	33	0			ABS	0	72	5	50	3	54	3

The approach for collating the data in this format can be direct vis-vis, recording the mark and grade point from the result collected from a certain department or indirectly, by entering each student courses in sheets of paper before transferring them into this final format. However, for the purpose of this article, we will not attempt to describe the various means of collating the senate format. We assume that the various data have been collected and the table drawn, what then remains is how to compute the respective CGPA. Before we continue, lets define the terms GPA and CGPA. GPA is the ratio of TGP to TCR and it is given by the formula

GPA =TGP/TCR ----------(1)

while CGPA is defined as the ratio

CGPA=CTGP/CTCR. ----------(2)

An interaction with some colleagues at the university revealed how they compute their students CGPA's. One way a staff member computed the CGPA is to use excel. This is done by writing adequate formula in excel to compute the TGP and TCR. However, their technique still relies on using a calculator to take the ratio of the TGP and the TCR afterwards. The second approach adopted at the departmental levels, is manual. This technique is herculean and can be very frustrating. Both methods used by fellow lecturers suffers the inability to state which course(s) a student is to repeat and students' class of degree has to be manually entered.

All these are circumvented by the proposed algorithm. In addition, as mentioned in the abstract, an earlier algorithm by the same author suffers the disadvantage that the user has to type in each students grade point as a result is subject to error, the user still has to type the computed CGPA and fill in the remark section by brute force. In this paper, we propose a Python-based algorithm for computing a students CGPA. It is fast, reliable, efficient, does not require the user to supply any input except the file containing the student names, score, grade point and courses. In the next paragraph, we give a brief introduction of Python.

Python is an object-oriented, high-level programming language [4]. Other examples of high-level programming languages are C, C++, Java and Perl. Low-level programming languages are often-times termed machine or assembly languages. Thus, programs written in high level programming languages needs to be processed into low-level ones before they can run successfully. Hence, the extra time spent in processing is a 'small disadvantage' of high-level languages. However, high-level languages are portable, easier to code, shorter, more likely to be correct and easier to read. Programs written in Python are executed by an interpreter and that is why it is often regarded as an interpreted language [4]. The interpreter is used in two ways viz.: script and interactive mode. In the later mode, Python programs are typed by the user while the interpreter displays the result. In the script mode, you save the code in a script file before using the interpreter to execute or run the contents of the file.

Python scripts are usually saved with names that end with .py. This means if you are working in an environment such as a UNIX command window and have a script named

filename.py, you type python filename.py. For more information on Python and how to download it, visit http://python.org *[1]*.

2. METHODOLOGY

In this section, we present the algorithm and explain some lines of the code. We assume that the various entries been entered into their various columns as shown in Table 1. The problem is simple and formulated as given the following known quantities: names, matriculation number, Mode of Entry (ME), Maximum Number of Semester Spent (MNSS), the Number of Semesters Spent (NSS), course codes, course GP, each students course mark and corresponding course grade point; for each student, find/solve/compute for the unknown quantities: TCE, TCR, TGP, GPA, CTCE, CTCR, CTGP, CGPA, remarks and outputting the results in excel format. Each row of the table represents all the knowns for each student and each empty cell means that the student did not register the course. To do this, the lecturer supplies the file containing the known quantities in .txt format by just copying the content of an excel format file and saving it into .txt format.

The main part of the program is the function gpa(line, courses, z) which depends on three parameters line, courses and z. The first dependent variable line consists of the content of each cell in each row. Of course, this depends solely on splitting each row and its tab delimited columns into individual entities or cells, where each cell is numbered in python format from zero unto the last column n = 31 (The total number of columns n in Table 1 is 32). The second variable courses consists of concatenating rows 6 and 7 beginning from column 7; for example, part of the main program concatenates into MTH101, MTH102, · · · , GST104. Observe that in the concatenation, the columns 'GP' are all skipped. The third variable z consists of all the credit loads starting from row 8, column 7. Mathematically, courses and z can be thought of as vectors.

We start by initializing Total_GP, Total_Credit_Registered and Total_Credit_Earned to zero in lines 2-4 of the function gpa(line, courses, z) and j=7 represents the column in which the score starts. Line 6 shows an initialization of Remark to "RPT". Lines 7 to 18 is the main loop for computing the GPA. The *if line[j]* means, if the line corresponding to column j is non zero, mind you because we are starting with j=7 and python starts counting from zero, this means that the code starts looping from the 8th column and above. The line *grade=int(line[j])* means, assign each grade point of each student in each course to the variable named grade, for example, for student with matriculation number UJ/20Y/NS/X1, in the first course MTH101, the student scored 74 and the grade point is 5, hence grade= 5.

Since the corresponding credit load is 3, this means credit_earned=3∗5=15 (since credit_earned=grade*credit load). The next line increments the Total_GP from zero to 15, Total_Credit_Registered is also incremented from zero to 3 because Total_Credit_Registered+=float(credit_load). The next line ascertains if the grade is not zero or if the candidate passed a particular course, increment the Total_Credit_Earned by the credit load and since the first student passed the course, the Total_Credit_Earned becomes 3. The next line of the program checks if the student actually failed the course, if yes, then the variable Remark which already had been initialized to 'RPT' is concatenated by the course code, otherwise, the loop is skipped. The j+=2 means subsequently increment j by 2 to 9, 11, 13 etc. The next course is now in line to be considered and so on and so forth until all the different variables have been computed, incremented or concatenated as the case maybe. After all the nonzero columns for each particular student have been considered, the code then computes the GPA for a particular student by the already given ratio

GPA =Total_GP/Total_Credit_Registered ---------(3)

As noted in the comment in the body of the program, there can never be a case of division by zero in the above formula because the Total_Credit_Registered add up. The last line in the function shows the output or desired result that we hope to use in the main body of the program. Therefore, the program returns Total_Credit_Registered, Total_Credit_Earned, Total_GP, GPA, Remark as integers, integers, integers, float and string respectively. We will not bother to explain the nitty-gritty of the main body of the program because it is cosmetics, the last line of the main body of the program calls the main function and assigns each of the variables to each of the ouput of the function in that order.

```
def gpa(line, courses, z) :
        Total_GP = 0.0
        Total_Credit_Registered = 0.0
        Total_Credit_Earned = 0.0
        j = 7                                   #where
the scores begin
        Remark = "RPT"
        for course, credit_load in zip(courses, z) :
                if line[j] :
                        if 0 <= float(line[j]) <= 5 :
#This ensures that no GP is greater than five
                                grade = int(line[j])
                                credit_earned =
grade*credit_load
                                Total_GP +=
float(credit_earned)

        Total_Credit_Registered += float(credit_load)
                                if grade != 0 :

        Total_Credit_Earned += float(credit_load)
                                if grade == 0 :
                                        Remark +=
" "+course+","
                        else :

        sys.stderr.write("\nPlease check your file to see if
there is no error\n")
```

```
            sys.stderr.write("\n!!!!!!!One of the credit loadS is above five!!!!!!\n")
                                    print "!Ooops, the following student has an incorrect field: ", line[2]
                                    print "One of the courses has a GP greater than five. That is", line[j]
                                    sys.exit(0)
                        #else :
                        #       print 'This/these students have VOLUNTARILY WITHDRAWN', line[2]

                        j += 2
                        if j == m+1 : break

            GPA = Total_GP/Total_Credit_Registered #???division by zero is not possible because courses registered add up
            return int(Total_Credit_Registered), int(Total_Credit_Earned), int(Total_GP), round(GPA, 2), Remark

#Part of the Main Program
x = []
y = []
z = []

w = "\t"+'TCR'+"\t"+ 'TCE' +"\t"+ 'TGP' +"\t"+ 'CGPA' + "\t" + 'REMARK'

tab5 = ""
for m in range(0,5) :
            tab5 += "\t"

tab6 = ""
for m in range(0, 6) :
            tab6 += "\t"

mm = 6      #Start from the seventh column but python starts counting from zero
for line in file :
            line = line.rstrip()
            line = line.split("\t")
            n = len( line ) - mm            #Total columns containing grades and gp minus redundant columns
            m = len( line )                 #Total number of columns in senate format
            print line
            if len(line) > 1 :
                        if line[0] == "S/NO" or line[0] == "S/N" :
                                    ww = functools.reduce((lambda x, y : str(x) +"\t"+ str(y)), line[0:6]) + w + "\t" + \
                                    functools.reduce((lambda x, y : str(x)+"\t"+str(y)), line[6:]) + w
                                    out.write(ww +"\n")

                                    for i in range(mm, m, 2) :
                                                x.append(line[i])

                        if line[0] == '' :      #Picking the empty cells and columns involving 101, 102, 103 and grade points
                                    ww = tab5 + functools.reduce((lambda x, y : str(x) + "\t"+ str(y)), line[0 : ]) + tab6
                                    out.write( ww + "\n")

                                    for i in range(mm, m, 2) :
                                                if int(line[i]) >= 100 :

            y.append(line[i])

                                                if int(line[i]) <= 5 :

            z.append(float(line[i]))

                        if len(x) == 0 or len(y) == 0 or len(z) == 0 :continue  #to avoid instances of length zero
                        if len(x) == len(y) == len(z) :
                                    courses = [x+y for x, y in zip(x, y)]

                        if line[0] == '' : continue
                        Total_Credit_Registered, Total_Credit_Earned, Total_GP, GPA, Remark = gpa(line, courses, z)
```

We bring this section to a close by presenting a glossary of terms used.

1. TCE is the summation of all Credit units of courses passed by a student. Its cumulative (CTCE) is TCE taken over more than one session.
2. TCR is the sum of all Credit units registered by a student. Its cumulative (CTCR) is TCR taken over more than one session.
3. TGP, first take the product of Grade Point and Credit unit for each course and sum over all courses. Its cumulative (CTGP) is TGP taken over more than one session.
4. GPA and CGPA already defined.

Next, we briefly describe the cost of the algorithm in terms of floating point point operations. Basically, we base our assumption of the efficiency of the algorithm on the fact that it is just a matrix vector multiplication and no matter how large the number of students are, it is sure to deliver in good time.

Floating Point Operations (Flops) of the Algorithm
For the inner product of two n-vectors x and y,

$$x^T y = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n \quad \text{---------(4)}$$

there are n multiplications and n-1 additions which equals 2n-1 flops or ~o(n) for n large. In the present algorithm, if we refer to the grade point of each student as an element of a matrix, this means the entire students grade points can be represented by a matrix say A. In the same vein, the credit units of a particular class can be thought of as an n-vector whose elements consists of individual course credit unit and a zero if a student did not offer the course. If m (total number of students in context) is the number of rows of the matrix A and n the number of columns, then this connotes a matrix vector operation. Hence, each element requires an inner product of length n at a cost of 2n-1 flops. The cost of the matrix-vector multiplication is then (2n-1)m floating point operations (flops). For m large, this results into 2mn flops [2] and [3].

3. EXAMPLES

In this section, we show the expected output of running the program applied on the Table 1. The format of the original file is .xls or .xlsx excel format or any tab delimited format.

The user needs to copy, paste and save it into a .txt format. If for example, the above table has been saved as filename.txt, then to run the program, you just type ***python senate_100.py filename.txt***. The expected output will be in ***filename_Senate_Format.xls***. The results are as shown in Table 2 and 3. Note that in both tables, we have decided to collapse/add 'necessary' columns/rows for easy viewing because the whole spreadsheet is big.

1. Ensure that the Python program is in the same directory/folder as your working directory.
2. In your working directory, open the excel file containing the known rows and columns.
3. In the menu bar, click Edit and Select All.
4. While all the rows and columns are selected, right click and choose Copy.
5. Open gedit or any text editor of your choice and paste.
6. Save the file as filename.txt, where filename is any name of choice.
7. On a Python shell/window type : ***python senate 100.py filename.txt***.
8. If everything is in order, you will see: !!!!!!!!!!Hurray, Program was successful and you should see output as filename_Senate_ Format.xls.

Table 2: While Table 1 shows the knowns, this table shows the result of applying the our algorithm to the inputs in Table 1. The unknown quantities TCR, TCE, TGP, CGPA and Remark are now automatically outputted in their correct columns. Note that we 'deleted' unwanted columns to show the results since the spreadsheet is quite big and cannot fit in adequately.

UNIVERSITY OF JOS										
FACULTY OF NATURAL SCIENCES										
DEPARTMENT OF MATHEMATICS										
100 LEVEL B.Sc. MATHEMATICS SENATE FORMAT RESULTS 2010/2011 SESSION										
S/NO	MAT. NO	NAME	ME	MNSA	NSS	TCR	TCE	TGP	CGPA	REMARK
1	UJ/20Y/NS/X1	SI	UME	12	2	32	32	139	4.34	PASS
2	UJ/20Y/NS/X2	YSM	UME	12	2	32	32	151	4.72	PASS
3	UJ/20Y/NS/X3	IAP	UME	12	2	36	36	156	4.33	PASS
4	UJ/20Y/NS/X4	RSA	UME	12	2	32	32	106	3.31	PASS
5	UJ/20Y/NS/X5	KAP	UME	12	2	32	24	45	1.41	RPT STA131, STA203
6	UJ/20Y/NS/X6	GJB	UME	12	2	31	31	89	2.87	PASS
7	UJ/20Y/NS/X7	GJR	UME	12	2	28	12	42	1.5	RPT MTH101, MTH102, MTH103, CS102, STA203
8	UJ/20Y/NS/X8	DYA	UME	12	2	28	18	40	1.43	RPT MTH102, CS102, STA203
9	UJ/20Y/NS/X9	JSA	UME	12	2	31	11	17	0.55	ON PROBATION: RPT MTH101, MTH102, MTH103, CS101, CS102, STA203, GST102
10	UJ/20Y/NS/X10	AZ	UME	12	2	31	31	135	4.35	PASS
11	UJ/20Y/NS/X11	HMI	UME	12	2	32	32	80	2.5	PASS
12	UJ/20Y/NS/X12	DE	UME	12	2	32	29	74	2.31	RPT MTH103
13	UJ/20Y/NS/X13	OOC	UME	12	2	31	13	27	0.87	ON PROBATION: RPT MTH101, MTH102, MTH103, STA203, PHY101, GST103
14	UJ/20Y/NS/X14	IAT	UME	12	2	31	21	50	1.61	RPT MTH101, STA203, PHY101
15	UJ/20Y/NS/X15	GJO	REM	12	2	32	12	36	1.13	RPT MTH101, MTH102, MTH103, CS102, STA131, STA203
16	UJ/20Y/NS/X16	EG	REM	12	2	36	24	65	1.81	RPT STA111, STA203, GST101

Table 3: We try to fit in the knowns and computed unknown quantities into a table and because of how big the spreadsheet is, the font sizes appear very tiny.

UNIVERSITY OF JOS
FACULTY OF NATURAL SCIENCES
DEPARTMENT OF MATHEMATICS
100 LEVEL B.Sc. MATHEMATICS SENATE FORMAT RESULTS 2010/2011 SESSION

S/N	MAT. NO	NAME	MTH 101	GP	MTH 102	GP	MTH 103	GP	CS 101	GP	CS 102	GP	STA 111	GP	STA 131	GP	STA 203	GP	PHY 101	GP	GST 101	GP	GST 102	GP	GST 103	GP	GST 104	GP	TCR	TCE	TGP	CGPA	REMARK
			3		3		3		2		3		4		4		4		3		4		2		2		2						
1	UJ/20Y/NS/X1	SI	74	5	69	4	63	4	71	5	66	4	71	5			70	5			59	3	76	5	62	4	63	4	32	32	139	4.34	PASS
2	UJ/20Y/NS/X2	YSM	75	5	68	4	70	5	80	5	72	5	94	5			67	4			71	5	81	5	76	5	65	4	32	32	151	4.72	PASS
3	UJ/20Y/NS/X3	IAP	46	2	72	5	60	4	73	5	77	5	76	5	63	4	68	4			80	5	77	5	62	4	60	4	36	36	156	4.33	PASS
4	UJ/20Y/NS/X4	RSA	44	1	50	3	44	2	73	5	63	4	60	4			40	1			75	5	68	4	72	5	61	4	32	32	106	3.31	PASS
5	UJ/20Y/NS/X5	KAP	45	2	40	1	43	1	71	5	43	1			29	0	35	0			45	2	60	4	43	1	42	1	32	24	45	1.41	RPT STA131, STA203
6	UJ/20Y/NS/X6	GJB	46	2	65	4	40	1	76	5	55	3					41	1	41	1	61	4	78	5	62	4	69	4	31	31	89	2.87	PASS
7	UJ/20Y/NS/X7	GJR	6	0	1	0	2	0	46	2	19	0					13	0			65	4	78	5	58	3	53	3	28	12	42	1.5	RPT MTH101, MTH102, MTH103, CS102, STA203
8	UJ/20Y/NS/X8	DYA	43	1	22	0	40	1	47	2	22	0					22	0			66	4	45	2	46	2	51	3	28	18	40	1.43	RPT MTH102, CS102, STA203
9	UJ/20Y/NS/X9	JSA	8	0	2	0	14	0	17	0	11	0					8	0	40	1	47	2	31	0	40	1	47	2	31	11	17	0.55	ON PROBATION:RPT MTH101, MTH102, MTH103, CS101, CS102, STA203, GST102
10	UJ/20Y/NS/X10	AZ	63	4	82	5	72	5	89	5	66	4					52	3	57	3	75	5	92	5	77	5	79	5	31	31	135	4.35	PASS
11	UJ/20Y/NS/X11	HMI	41	1	41	1	46	2	57	3	66	4	43	1			41	1			70	5	83	5	50	3	56	3	32	32	80	2.5	PASS
12	UJ/20Y/NS/X12	DE	41	1	45	2	18	0	40	1	56	3			56	3	42	1			61	4	66	4	60	4	57	3	32	29	74	2.31	RPT MTH103
13	UJ/20Y/NS/X13	OOC	5	0	3	0	5	0	49	2	43	1					10	0	9	0	58	3	43	1	ABS	0	51	3	31	13	27	0.87	ON PROBATION:RPT MTH101, MTH102, MTH103, STA203, PHY101, GST103
14	UJ/20Y/NS/X14	IAT	9	0	42	1	41	1	46	2	45	2					35	0	18	0	71	5	42	1	52	3	52	3	31	21	50	1.61	RPT MTH101, STA203, PHY101
15	UJ/20Y/NS/X15	GJO	6	0	1	0	3	0	80	5	21	0			15	0	8	0			59	3	52	3	45	2	48	2	32	12	36	1.13	RPT MTH101, MTH102, MTH103, CS102, STA131, STA203
16	UJ/20Y/NS/X16	EG	43	1	44	1	43	1	70	5	62	4	21	0	56	3	33	0			ABS	0	72	5	50	3	54	3	36	24	65	1.81	RPT STA111, STA203, GST101

4. CONCLUSION

In this work, we have presented an efficient algorithm for computing the CGPA of students in senate format at the University of Jos which surpasses an earlier algorithm by the same author [5]. The present algorithm is self starting, reliable, does not require the user to supply any information except those already in the pre-compiled senate format file, and can fill in the gaps with the required unknowns or information (TCE, TCR, TGP, GPA, Remarks etc.) in the desired format with great speed and accuracy. The algorithm will enhance early turnover rate of departmental senate formats and safe level coordinators the nightmare of doing the required computations manually or in sheets of papers. The program can be obtained directly from the author via email.

Future Work

The program presented in this paper requires the user to convert a .xls/.xlsx file to its .txt extension by simply cutting and pasting. However, this can be circumvented and it will be great to see how this program can be hosted on the web as a back-end to a php or html script. All these are achievable but more time is needed to make this a reality.

Acknowledgement

The author wishes to thank Professor L. S. O. Liverpool of the Department of Mathematics, University of Jos for his inspiration in writing the first article and the Vice-Chancellor of the University of Jos, Professor Hayward Mafuyai. My gratitude also goes to Mr Nimchak of the Appointment and Promotion Committee of the University of Jos for his role in making sure I came to Cape Town to pursue my post doctoral studies. My son TiOluwani (plus wife) too deserves commendation as he was born while I was away and to Prof. Nicola J. Mulder, my Principal Investigator at the Computational Biology Group, Faculty of Health Sciences, University of Cape Town, South Africa. Developers of open source software are hereby duely acknowledged.

REFERENCES

[1] Python: *http://python.org*

[2] 2.Complexity of matrix algorithms. http://www.seas.ucla.edu/~vandenbe/103/lectures/flops.pdf

[3] G. Golub and C. F. Van Loan: Matrix Computations, pages 2-4, third edition, John Hopkins University Press 1996

[4] Downey A, Think Python: How to Think Like a Computer Scientist Green Tea Press, Needham, Massachusetts, *http://www.thinkpython.com*, 2012.

[5] R. O. Akinola: An Octave Algorithm for computing a Student's Cumulative Grade Point Average. Publications of the ICMCS: Volume 5, Pages 67-76. (2012).

Author's Brief

Dr Richard Olatokunbo Akinola is a lecturer in the Department of Mathematics, University of Jos. He attended Government Secondary School Laranto, Jos-Plateau State (Can anything good come out of Nazareth?) and a graduate of Mathematics from the University of Jos. Holds a masters degree cum laude from Stellenbosch University, South Africa and a Ph.D in Applied Mathematics from the University of Bath, UK. He is currently doing postdoctoral research in Computational Biology at the Faculty of Health Sciences, University of Cape Town, South Afica. My research interests is in Numerical Analysis and Scientific Computing. To God be the glory for bringing me thus far because a man can receive nothing except it is given to him from above.