A Basic Digitising Software

SAMIR KUMAR PAL

In many scientific experiments, it becomes necessary to measure the area under the experimentally obtained curve (spectrum). For instance, for measurement of fluorescence quantum yield and differential scanning calorimetry etc, it is often required to store the experimentally obtained hardcopy (spectrum) in the form of a data file for future use so as to regenerate the curve using some curve-tracing software such as Grapher or Microcal Origin, etc.

Typically, in such an experiment, one of the physical quantities is obtained as a function y(x) of another physical quantity x, which is treated as the independent variable. The quantity of interest, however, is not y(x) but rather I(x, y) where $I(x, y) = \int_a^b y(x)dx$, $(a \le x \le b)$, i.e. the area under the curve.

However, if the experimental output is in the form of a spectrum and the spectrometer is not computer interfaced, we need a scanner to digitise the spectrum, determine the area under the spectrum, and store the corresponding data. This technique, though effective, is at the same time expensive. An alternative cheaper technique has been developed by the author using a GWBASIC program which does away with the scanner.

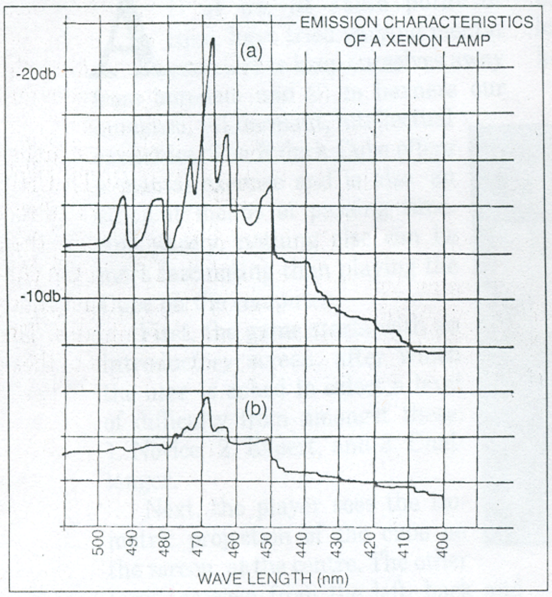


Fig. 1: Actual emission spectrum of xenon lamp obtained using flourimeter

The experimental spectrum is first traced on a transparent sheet. This sheet is then attached on to the screen of the computer monitor such that the base line of the spectrum coincides with the green coloured baseline of the program-generated screen. The spectral curve is then traced pixel by pixel (- - - -) on the screen beneath the attached transparent sheet by using the four arrow keys. When the entire curve has been thus redrawn on the monitor screen, the paper is taken off and <Esc> key is pressed. If the data file corresponding to the spectrum is required, then the start wavelength, end wavelength, and name of the data file can be specified interactively. The program computes the area under the curve in arbitrary (arb.) units and displays the result as the output.

The actual emission characteristic (spectrum) of a xenon lamp obtained using a Perkin Elmer made fluorimeter, which is not computer interfaced, is shown in Fig. 1(a). Fig. 1(b) is the attenuated version of the same. Figs 2(a) and 2(b) are the program-generated replicas of Figs 1(a) and 1(b) respectively.

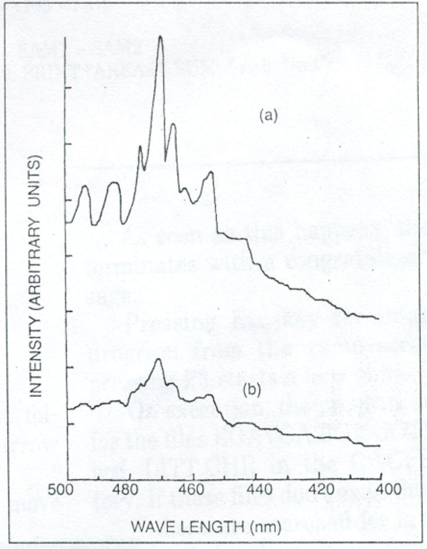


Fig. 2: Program generated replicas of spectrum obtained using software

It is obvious that the program-generated spectra are true reproduction of the actual spectra. The area under the spectra Figs 2(a) and 2(b) have been calculated to be 23,445 arb. units and 6,142.5 arb. units using the program. From the ratio of the two areas we can easily infer that, within the wavelength range of 400 nm to 500 nm, the total number of photons emitted in the attenuated version is 3.8 times less than that of previous one.

Thanks are due to author's colleages A. Datta and D. Mandal for their very useful comments and suggestions.

Digitising Software Program in Basic

```
10 DIM X(1000), Y(1000), F(2000), G(2000), A(1000), B(1000)
20 I = 0
30 CLS : SCREEN 8: KEY OFF: X = 1: Y = 195
40 GOSUB 530
50 PSET (X, Y), 15
60 I = I + 1
70 R = POINT(0)
80 C = POINT(1)
90 X$ = “”
100 WHILE X$ = “”: X$ = INKEY$: WEND
110 IF X$ = CHR$(27) THEN GOTO 230
120 IF X$ = CHR$(0) + CHR$(72) THEN Y = Y - 1: N = Y + 1
130 IF X$ = CHR$(0) + CHR$(80) THEN Y = Y + 1: N = Y - 1
140 IF X$ = CHR$(0) + CHR$(75) THEN X = X - 1: M = X + 1
```

```
150 IF X$ = CHR$(0) + CHR$(77) THEN X = X + 1: M = X - 1
160 F(I) = Y: G(I) = X
170 IF X < 0 THEN GOTO 230
180 IF X > 637 THEN GOTO 230
190 IF Y > 195 THEN GOTO 230: BEEP
200 IF Y < 0 THEN GOTO 230
210 REM PRESET(M,Y)
220 GOTO 50
230 BEEP: T1 = X: LOCATE 5, 10: PRINT "DO YOU WANT TO
    CONTINUE(Y=1/N=2)"
240 LOCATE 6, 10: INPUT A10
250 IF A10 = 1 GOTO 50
260 IF A10 = 2 GOTO 270
270 CLS : GOSUB 530
280 FOR I = 1 TO 1500: PSET (G(I), F(I)), 15: NEXT I
290 PRESET (0, 0)
300 FOR C = 1 TO T1
310 FOR R = 1 TO 195
320 IF POINT(C, R) <> 0 THEN GOTO 360
330 NEXT R
340 NEXT C
350 GOTO 370
360 X(C) = C: Y(C) = R: GOTO 340
370 FOR I = 1 TO T1: LOCATE 6,10: PRINT "X="; X(I), "Y=", Y(I): NEXT I
380 LOCATE 6, 10: INPUT "Do you want to digitise spectra(y=1/n=2)?",A20
385 LOCATE 6, 10: PRINT "                    "
390 IF A20 = 2 GOTO 470
400 LOCATE 6, 10: INPUT "START WAVELENGTH(nm)=>", S
405 LOCATE 6, 10: PRINT "                    "
410 LOCATE 6, 10: INPUT "END WAVELENGTH (nm)=>", E
415 LOCATE 6, 10: PRINT "                    "
420 NM = (E - S) / T1: A(1) = S: B(1) = (195 - Y(1)) * 10
430 FOR I = 2 TO T1: A(I) = A(I - 1) + NM: B(I) = (195 - Y(I)) * 10: NEXT I
440 LOCATE 6, 10: INPUT "ENTER FILE NAME=>", A$
445 LOCATE 6, 10: PRINT "                              "
450 OPEN "o", #1, A$
460 FOR I = 1 TO T1: PRINT #1, A(I), B(I): NEXT I
470 CLS
480 GOSUB 530
490 FOR I = 1 TO T1: LINE (X(I), 195)-(X(I), Y(I)), 15: NEXT I
500 GOSUB 630
510 LOCATE 1, 1
520 END
530 CLS : KEY OFF: SCREEN 8
540 COLOR 13, 0
550 LOCATE 1, 6: PRINT "A BASIC DIGITISING SOFTWARE
    BY SAMIR K. PAL"
560 LINE (0, 10)-(639, 10), 14
570 LINE (0, 10)-(0, 199), 14
580 LINE (639, 10)-(639, 199), 14
590 LINE (0, 199)-(639, 199), 14
600 COLOR 10, 0
610 LINE (0, 195)-(639, 195), 10
620 RETURN
630 SUM = 0
640 SAM1 = (195 - Y(I)) / 2: SAM2 = (195 - Y(T1)) / 2
650 FOR I = 2 TO (T1 - 1)
660 SUM = SUM + (195 - Y(I))
670 NEXT I
680 SUM = SUM + SAM1 + SAM2
690 LOCATE 6, 10: PRINT "AREA="; SUM; " Arb. Unit"
700 RETURN
```

Rubick Cube

SHREE KUMAR

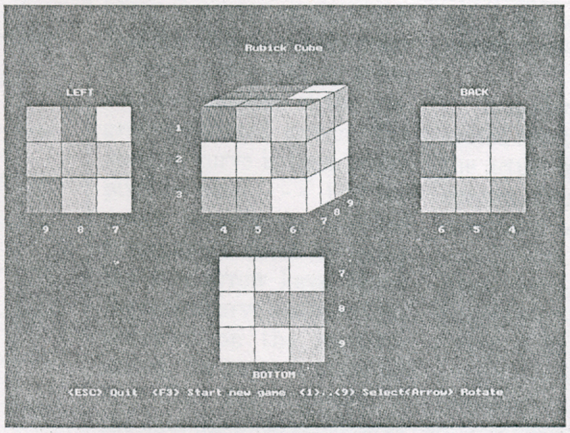
All of us, at some point of time, have tried varying mental exercises to keep ourselves away from boredom and (!) to harness our immense, yet dormant, intellectual capacities. The Rubick Cube offers a mental exercise and is also an efficient means of passing time. And surely, nothing else can be more fascinating than playing the Cube on the computer.

First, the game starts with an introductory screen, after which the user is asked to select a level of difficulty from amongst these: 1. Novice, 2. Expert, and 3. Challenger.

Next, the player sees the isometric projection of the cube on the screen, at the centre. The other faces, as seen from the left, back and bottom, are also shown to the left, right, and below the Cube respectively. Pressing any key jumbles the Cube.

The Cube is divided into nine rotatable parts. To move a part, press the number corresponding to the part followed by the direction (using arrow keys).

Suppose the player wants to move


Rubick Cube
LEFT
BACK
1
2
3
9 8 7
4 5 6
7 8 9
6 5 4
7
8
9
BOTTOM
<ESC> Quit <F3> Start new game <1>..<9> Select<Arrow> Rotate

the top row to the left, all he has to do is to press the corresponding number (in this case 1), followed by the left arrow key. Finally, the user has to end up with the same colour on each of the faces.

As soon as this happens, the game terminates with a congratulatory message.

Pressing Esc key terminates the program from the game screen and pressing F3 starts a new game.

On execution, the program searches for the files EGAVGA.BGI, GOTH.CHR, and LITT.CHR in the C:\CPP directory. If these files don't exist there, then these are searched for in the current directory. If not found there too, the program terminates. The user can specify the path to these files in the command line itself. For example, use

```
A:\>RUBICK C:\TC\BGI
```

if the above files are in the C:\TC\BGI directory. It's always preferable to have these files in the hard disk because loading these files from a floppy can take time. You can even create a standalone executable file using register bgi driver and register bgi font—the standard library functions of C.

This program was initially designed for a monochrome monitor; so the colour selection may not be to your liking. You can change the colours to suit your taste.

Further improvements that the user