CSE535 LAB BOOK

CORY COOK

Contents

1. Lab 1

1.1. **Introduction.** Stirlings Approximation:

$$n! \approx \sqrt{2\pi n}(\frac{n}{e})^n \tag{1}$$

Find the absolute and relative error and evaluate the error as n grows for Stirlings Approximation. We are given a function $n!$ and an approximation of the function $\sqrt{2\pi n}(\frac{n}{e})^n$ and we are tasked with nding out just how good the approximate function is by evaluating its absolute and relative error.

1.2. **Method.**

We can solve this rather rigorously by testing the functions at each value and comparing them. We would rst test the accurate function, then its approxi- mation, get the dierence of the values (absolute error) and, nally, display the absolute error as a percentage of the result (relative error).

1.3. **Solution.**

I tested the function in SciLab using the following bit of code:

```
for n=1:10
    p(1,n) = sqrt(2*%pi*n)*(n/%e)^n;
    p(2,n) = factorial(n);
    y(n) = p(2,n) - p(1,n);
    x(n) = y(n)/p(2,n);
end
// p(1,n) - approx.
// p(2,n) - actual
// y - absolute errror
// x - relative error
```

Listing 1. SciLab Solution

Which produced the following output:

p: column 1: 0.9221370, 1. column 2: 1.9190044, 2. column 3: 5.8362096, 6. column 4: 23.506175, 24. column 5: 118.01917, 120. column 6: 710.07818, 720. column 7: 4980.3958, 5040. column 8: 39902.395, 40320. column 9: 359536.87, 362880. column 10: 3598695.6, 3628800.

y: 0.0778630 0.0809956 0.1637904 0.4938249 1.980832 9.9218154 59.604168 417.60455 3343.1272 30104.381

x: 0.0778630 0.0404978 0.0272984 0.0205760 0.0165069 0.0137803 0.0118262 0.0103573 0.0092128 0.0082960

At this point we can input plot(x) into the console to get a graphical rep- representation of our relative error which, for larger n ranges appears to resemble the graph of 1 x. We can also input plot(y) into the console which returns a graph that increases exponentially or factorially as it were. Having it displayed graphically is nice; however, you can see that for each successive n the x value (relative error) is less than the previous x and each successive y value (absolute error) is greater than its preceding y value.

1.4. **Conclusion.**

2. Lab 2

2.1. **Introduction.**

In class we performed Gaussian Elimination on an image matrix to clarify the image; however, I still don't know how images are drawn using matrices and what the image matrix actually means. The goal of this lab is to create a gradient image using a matrix and scilab. This may sound excessively simple to some, but I really have no idea what I'm doing. I would really like to know how to generate matrix based images so I figured I could use this lab as a personal learning experience.

2.2. **Method.**

There are several approaches I can take to solve this problem, and I will likely end up using all of these methods before I arrive at my final answer. One approach is to simply look up the answer from some source and attempt to replicate the method on my own. I feel like my personal learning experience will be most rewarding if I leave this solution as a last resort. Another approach is to make educated guesses and attempt to implement them until I arrive at a solution. While this is how I usually approach projects that interest me, I think I will attempt my next approach. The approach I will likely attempt first is to use the image data from class, make changes to the data, and note changes in the result. The last solution seems like the best approach since I will be starting with a working set of data. Before I get started with my chosen method I am going to attempt to make an educated guess as to how an image is generated using a matrix.

Using what I know about matrices and what I know about a two-dimensional image I am going to design my own method for displaying graphical matrices. I am assuming that horizontally arranged elements in a matrix will be referred to as rows and vertically arranged elements will be referred to as columns with a respective order of reference. A two-dimensional image will be a construct of points referred to individually as a point in a Cartesian-like coordinate system. Elements from the matrix will map directly to points on the image where the row and column of the matrix value will determine its location in the image and matrix elements value will determine the color that is displayed. I am going to use HTML5 and the Canvas element to model my design. I will build a Cartesian-like coordinate system with boxes to represent 1x1 image elements (pixels) and use a matrix or matrix-like data element (multidimensional arrays) to assign colors to the individual elements. I will attempt to create a word image similar to the one in the class exercise. Now that I have my best guess as to how I would design a matrix-based image I can move on to how matrix-images are implemented in SciLab.

For the SciLab portion of this Lab I will take the matrix generated as a result of the class exercise and change the values of the individual elements one-by-one until I can determine the effect of changing a value. Once I have discovered the result of changing values I will add an additional column of values and note how it changes the output. As a test I will proceed to add an additional row to the matrix and note any changes. If my guess is correct I should notice that changes to the values of the original matrix change colors on the rendered image, adding additional columns should extend the image on the Cartesian "x" axis, and adding additional rows should extend the image on the "y" axis. Once I have determined how matrix images are generated in Scilab I can procede to the final stage.

During the last stage I will create a gradient matrix-based image in Scilab. The image will fade from green to blue, left to right as clearly as possible. The image generated on my operating system may not me what you see with the same code, but I will include images of what I see on my screen. My operating system does some additional blurring so the gradient will likely look fantastic for me and pixellated for everyone not running Windows 8.

2.3. **Solution.** The first stage is implementation of my matrix-image design. To make life easy on myself I just used the standard RGB string as my matrix-element value.

```
var pixel = 20; //pixel size
var imagedata = [""];
imagedata[0] = ["#F00", "#FFF", "#F00", "#FFF", "#0F0", "#0F0", "#0F0"
    ];
imagedata[1] = ["#F00", "#FFF", "#F00", "#FFF", "#FFF", "#0F0", "#FFF"
    ];
imagedata[2] = ["#F00", "#F00", "#F00", "#FFF", "#FFF", "#0F0", "#FFF"
    ];
imagedata[3] = ["#F00", "#FFF", "#F00", "#FFF", "#FFF", "#0F0", "#FFF"
    ];
imagedata[4] = ["#F00", "#FFF", "#F00", "#FFF", "#0F0", "#0F0", "#0F0"
    ];
var b = document.getElementById("image");
var board = b.getContext("2d");
board.canvas.height = imagedata.length * pixel;
board.canvas.width = imagedata[0].length * pixel;
for (i = 0; i < imagedata.length; i++) {
    for (j = 0; j < imagedata[i].length; j++) {
        board.fillStyle = imagedata[i][j];
        board.fillRect(j * pixel, i * pixel, pixel, pixel);
    }
}
```

Listing 2. Define Image Data

The next stage requires me to load the data from the in-class assignment and test my theoretical model. I'm not going to list the values out nor will I display

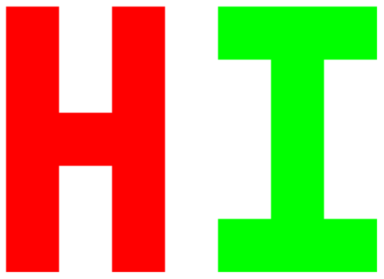
FIGURE 1. Output of imagedata

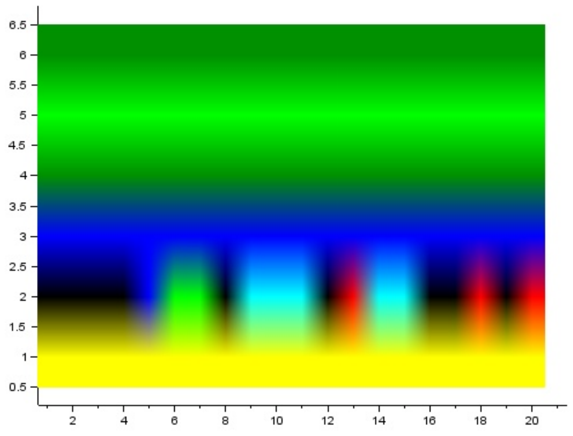

FIGURE 2. in-class image after my tampering

the result here since I don't want to ruin the surprise for anyone who hasn't had the opportunity to figure it out. Loading the data into scilab and getting the result using our methods from lab I saved the result in x and plotted the image with Matplot(round(x)). The round is necessary for me because of my operating system. I changed the value of x(1,1) and the element in the first row and first column changed color while the others stayed the same. Using x(:,20) = 5; sets all of the values in column 20 to the color associated with 5 (red) and using x(6,:) = 7 adds a row to the bottom of the image that is the color associated with 7 (yellow). Your color associations may be different as 1 is black when I believe that it should be white (zero is also black). Since I am just messing with the values here I may as well determine which colors I will need to generate my gradient image in the next stage. After changing some of the values I find that green is associated with 3 and blue is associated with 2; however, there are no partial values in-between that I can build a gradient with.

Since in-between values do not generate colors the way that I had imagined I am actually going to attempt to generate a faux gradient (figure 3). The Scilab only

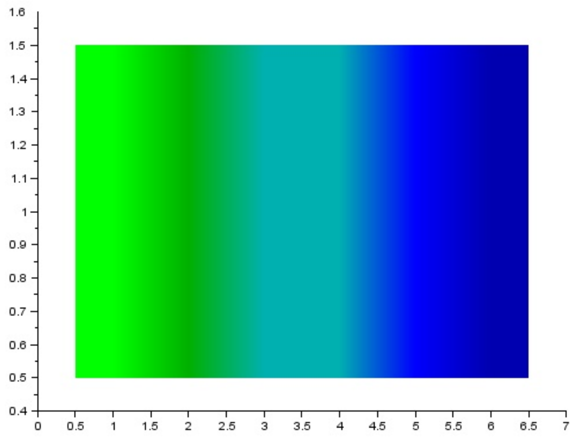

Figure 3. faux Scilab gradient

even looks close to a gradient because of my operating system. I'm sure that the color issue that I'm having is related to the operating system as well.

```
clear
x(1,1) = 3;
x(1,2) = 14;
x(1,3) = 17;
x(1,4) = 17;
x(1,5) = 2;
x(1,6) = 10;
Matplot(x);
```

Listing 3. faux gradient

2.4. **Conclusion.** My conclusion is that Scilab is not completely supported on Windows 8 yet, and that my guess for matrix-image implementation was fairly accurate with (possibly) some differences in the the way that the color value is implemented. Basically Matrix-based images get the location of the element in the image based on the location in the matrix (row,column) and the value of the matrix element is associated with the color that is displayed.