Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 1 (2012) 913–919

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

# Introductory computational science using MATLAB and image processing

D. Brian Larkins[1], William Harvey

*Dept. of Computer Science and Engineering*
*The Ohio State University*
*Columbus, OH 43210*
{*larkins,harveywi*}@*cse.ohio-state.edu*

**Abstract**

We describe a new course designed to introduce engineering students to computational thinking. One of the most significant challenges in teaching an introductory-level applied computing course is that students are not expected to have substantial math and science experience. To address this, we have developed exercises which use edge detection and basic image processing to motivate the use of programming MATLAB in a non-trivial scientific application. MATLAB is a popular high-level programming language and environment which supports a wide range of computational science applications. MATLAB has strong support for operating on image data, which allows us to balance solving practical engineering problems with basic core concepts of computer science such as functional abstraction, conditional execution, and iteration.

## 1. Introduction

The use of computers for simulation and data analysis has become widespread throughout many engineering disciplines. As a result, engineering departments are requiring students to take courses which provide an introduction to computational thinking and applied computer programming. The need for students to have familiarity with algorithmic thinking, programming concepts and their application to science and engineering problems spans many disciplines and departments. As a result, an introductory course has been created and offered through the Department of Computer Science and Engineering at Ohio State. Designing a general, introductory class which effectively delivers a mix of basic programming concepts and scientific applications is a challenge. Such a course must strike a balance between covering the core concepts of computation with the need for practical skills to solve real engineering problems.

The students in this course come from a variety of science and engineering backgrounds: from electrical, civil, and systems engineering, as well as materials science and other fields. To allow a wide range of students from a diverse range of disciplines to take this course early in their programs, students are only required to have had at least one calculus course, with no other prerequisites. As such, it is difficult to develop exercises and other experiential course components which reflect actual applications related to a student's chosen field.

[1]corresponding author

Regardless of the math and science background of a student or their intended major, an introductory course in computational science must teach practical skills that will be useful to every student. While it is possible to solve many science and engineering problems using low-level programming languages, it is much more common in practice to use high-level tools which support problem-solving within a specific domain. Ideally, students should be exposed to applied problem solving with computational environments that they would see later in their education or vocation.

To address the tension between providing basic computer science knowledge and the need for practical experience with applied scientific computing, this course is taught using the MATLAB programming language environment. MATLAB is a powerful tool which has seen widespread adoption throughout industry and academia. Since MATLAB provides a complete high-level programming language, it can be used as a basis for teaching core CS concepts. In this paper, we describe the outline of an introductory course suitable for teaching science and engineering students computational thinking. Our approach is based on two key insights. First, MATLAB is a tool which provides a rich environment for both exploring general computing concepts and solving domain-specific applied problems. Second, that some applied image processing problems can be intuitively understood with a basic calculus background and are similar in spirit to other applied science and engineering problems.

We make the following contributions: First, we describe a course offered at Ohio State which covers both general computer science concepts and applied scientific computing using MATLAB. Second, we demonstrate how image processing can be used as a model scientific application. We provide specific exercises based around detecting the edges in an image which give students experience solving realistic problems, but do not require substantial math background. Lastly, we demonstrate the effectiveness of these exercises with results from a student survey.

### *1.1. MATLAB as a Computational Tool*

MATLAB is software environment which supports high-level programming, simulation, and visualization of science and engineering applications. MATLAB is a useful introductory tool for teaching computational science for several reasons: MATLAB enjoys popularity in many different science and engineering disciplines in both industry and academia [1]. It is a multi-platform system, which is scalable from laptops all the way up to high-performance computing clusters [2]. There is a broad support base, with an active user community, sample code repositories, and the availability of many textbooks. MATLAB also supports a large number of specialized toolboxes, such as support for advanced statistics, partial differential equations, signal processing, etc.

A key benefit of using MATLAB is that while much of the programming can be done at a high-level, it also supports general programming constructs such as conditionals, loops, and functions which correspond to other popular programming languages such as C++ or Python. MATLAB is a commercial product, which can be prohibitively expensive for students to acquire, however the College of Engineering at Ohio State provides all engineering students access to MATLAB under a site license.

## 2. Course Overview

The course is designed as a 10-week class (1 quarter) which meets four days a week: three lecture classes and one lab meeting. The textbook covers both introductory programming skills and techniques specific to MATLAB for vector and matrix manipulation [3].

### *2.1. Topics*

Students are first given an informal treatment of algorithms and how to view problem solving as a set of steps.

Students are then exposed to the following concepts in order:

**Expressions:** Basic arithmetic, logical, and boolean expressions. Constructing complex expressions from smaller, simpler ones.

**Variables:** Using variables in expressions and changing their values with assignment.

**Vectors and Matrices:** Vectors as a first-class datatype, vector creation, vector operations (e.g. `sum()`, `max()` etc.), and expressions with vectors (e.g. vector-vector operations, vector-scalar operations). Element-wise operations: updating an element, slicing, using simple and complex expressions to index into vectors. Extend same concepts to matrices.

**Visualization:** Displaying data, input/output, 2D and 3D plotting.

**Functions:** Functional abstraction, input parameters, return values, returning multiple values.
**Conditionals:** Introduce `if` statements, work with control-flow diagrams to differentiate between `if`, `if...else`, and `if...elseif` conditional blocks.
**Iteration:** Bounded definite loops using `for`, work examples demonstrating iteration space, variable loop bounds, etc.
**Composition:** Whole program construction, composition of various program elements (conditionals, loops, functions, etc.)

### 2.2. Course Structure

Students are assigned homework every week to assess their level of understanding the lecture material. The dedicated lab session is used as a closed laboratory at the beginning of the course, with assignments due at the end of class. The closed lab sessions focus on giving students hands-on experience with the lecture topics described above. The closed labs are designed to develop basic programming skills and the direct application of MATLAB-specific syntax and constructs.

As students gain more sophisticated programming knowledge and are able to compose rudimentary programs, the lab session is used to work on four in-depth laboratory assignments. These labs are more focused on problem definition and specification, leaving students to develop their own algorithms and compose programming elements to generate the correct output. The first two laboratory assignments introduce students to the problem solving process by asking them to develop solutions to conceptually straightforward problems. The first focuses on manually computing basic statistical parameters of a predefined data set and the second requires the use of conditionals to implement a decision tree.

The last two laboratories introduce students to basic image processing theory and assign students the task of using this knowledge to detect the edges in an image. These assignments are discussed in detail below.

## 3. Understanding Computational Science through Image Processing

### 3.1. Why Image Processing?

Since this course is an introductory course, students are assumed to understand basic calculus concepts, but may not have had linear algebra, signal processing, partial differential equations, etc. Any assignments designed to apply programming knowledge and experience to a computational problem must either be readily understood with only a rudimentary calculus background, or the underlying theory must be introduced in class. Furthermore, given that student demographics range over different science and engineering disciplines, no one application will be relevant for all students.

Image processing is a good choice for understanding computational science for several reasons. First, it is a challenging subject that is itself the subject of active research and based on mathematical and statistics theory common to many fields [4]. Edge detection, in particular, is a useful pedagogical tool because although the underlying theory can be complex, it can be understood intuitively with an understanding of basic calculus. This intuitive approach to edge detection is bolstered by the rich support for image processing provided by MATLAB, allowing a great deal of flexibility in deciding which aspects of the problem are exposed to students and which can be handled by the computational environment. Lastly, image processing is inherently visual. Students are working with images which can be viewed at every step of the program, strengthening the intuitive sense of *process* and algorithmic thinking.

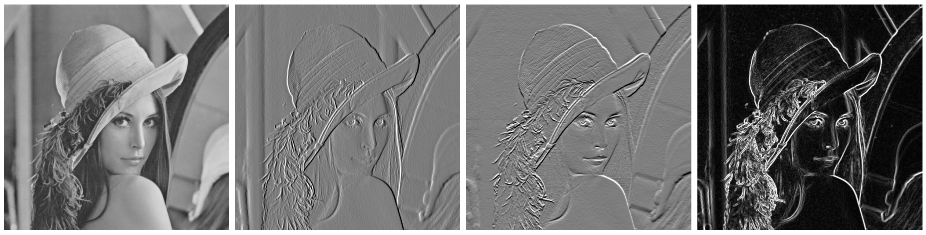
### 3.2. Overview of Edge Detection

Given a digital representation of a grayscale image, the objective is to identify pixels that correspond to edges in the image. For example, Figure 1 illustrates a sample input image and detected edge pixels.

Mathematically, a grayscale image can be represented via the map

$$I : \mathbb{R}^2 \to \mathbb{R} \tag{1}$$

This representation provides an intuitive interpretation of an image as a landscape where the intensity at a point is equivalent to its height. Thus, edge-like structures in the image become barriers and cliffs where the height of the



(a) Original image (b) Gradient in $x$ direction (c) Gradient in $y$ direction (d) Gradient magnitude

Figure 1: Edge detection framework in which the gradient magnitude is used to identify pixels that correspond to edge features in the input image.

landscape changes abruptly. These features can be characterized as collections of points where (locally) the change in height in one direction is large, yet the change in height in an orthogonal direction is small.

For every point $\vec{x} \in \mathbb{R}^2$, the rate of change of $I$ is determined by the *gradient*, defined as follows:

$$\begin{aligned}(\nabla I) &: \mathbb{R}^2 \to \mathbb{R}^2 \\ \vec{x} &\mapsto (I_x(\vec{x}),\ I_y(\vec{x}))\end{aligned} \tag{2}$$

$\nabla I(\vec{x})$ captures the direction in which the rate of increase of $I$ at $\vec{x}$ is maximized, with $||\nabla I(\vec{x})||$ representing the magnitude of change. Thus, applying a threshold to $||\nabla I(\vec{x})||$ provides a simple classifier that identifies regions belonging to the edge features in the image.

While this mathematical framework is sound, its implementation requires adaptation to work reliably with digital grayscale images. An 8-bit grayscale digital image of dimensions $(w \times h)$ can be represented using the map

$$L : [1, w] \times [1, h] \to [0, 255] \tag{3}$$

The partial derivatives of L along the $x$ and $y$ directions can be computed as follows:

$$L_x(x, y) = L(x + 1, y) - L(x, y) \tag{4}$$

$$L_y(x, y) = L(x, y + 1) - L(x, y) \tag{5}$$

However, due to discretization artifacts and presence of noise, it is beneficial to locally smooth $L$ and to use a small neighborhood around each point to estimate the partial derivatives. This leads to the Sobel operator formulation for estimating the partial derivates:

$$L_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * L \qquad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * L \tag{6}$$

where $*$ denotes discrete convolution. The result of using the Sobel operator [4] to compute the partial derivatives of an input image is shown in Figures 1(b) and 1(c). When computing the partial derivatives along the boundaries of $L$, the image intensities beyond the borders are considered to be zero, although other heuristics can be used. It is possible that the images $L_x^{\to}$ and $L_y^{\to}$ of $L_x$ and $L_y$ contain values beyond the range of valid 8-bit grayscale quantities. In this case, they can be corrected using the following affine transformation:

$$\hat{L_x} = \frac{255(L_x - \min L_x^{\to})}{\max L_x^{\to} - \min L_x^{\to}} \qquad \hat{L_y} = \frac{255(L_y - \min L_y^{\to})}{\max L_y^{\to} - \min L_y^{\to}} \tag{7}$$

With $\hat{L_x}$ and $\hat{L_y}$ in hand, the gradient magnitude $G(\vec{x})$ is computed using the $L_2$-norm, and edge points are detected by applying a threshold $\tau \in [0, 255]$ to $G$.

### *3.3. Exercises*

Students work with the edge detection problem in two lab assignments over three weeks. The first lab assignment is primarily focused on introducing students to working with image data, creating functions, and using basic loop constructs. The second lab focuses more on the implementation of basic image processing algorithms.

**First Lab:** This project consists of two functions and a main program which perform edge detection on an image.

1. *Create a function to display a matrix as an image.*
   This function will be used to display progress throughout the lab, and also can aid in debugging. Functions from the MATLAB Image Processing Toolbox are called to do the actual display work, having students focus on function creation and parameter passing.
2. *Create a function to convert a grayscale image to a monochrome image at some threshold.*
   This function takes a grayscale image matrix and a threshold value, $t$, and produces a new monochrome image matrix. Students implement a function which iterates over each point, $p$, in the grayscale image and sets the corresponding point, $p'$, in the monochrome image to either black or white depending on whether or not the intensity value of $p$ is greater than the threshold, $t$. Students are required to implement this with an explicit loop structure which requires the composition of loop and conditional constructs.

The main program uses these two functions to implement the edge detection process:

1. *Read an image file into a matrix variable.*
   This functionality is provided by the Image Processing Toolbox.
2. *Create gradient filter masks in the x and y directions.*
   This step creates the two $3 \times 3$ Sobel operator filters which are used to detect edges in each of the $x$ and $y$ directions. These filters are created from Image Processing Toolbox functions but could be defined in the exercise or derived from theory. There are a number of filters and techniques used to detect edges in an image which could be used, such as Sobel, Canny, Laplacian, or Prewitt filters[5, 6, 4].
3. *Apply the gradient filters to the image.*
   In this exercise, students call the MATLAB function, `imfilter()`, which performs the discrete convolution of the filter and the image matrix. The resulting matrix consists of intensity values which represent the rate of change of the intensity values in the source image. The second edge detection laboratory has the students implement the filtering convolution, without relying on the `imfilter()` function.
4. *Use MATLAB operations to compute gradient magnitude image.*
   The magnitude is computed by taking the $x$ and $y$ filtered gradient images and computing a new magnitude image, which is defined by: $m = \sqrt{x^2 + y^2}$.
5. *Call threshold function to convert grayscale magnitude image to a monochrome image.*
   The grayscale magnitude image is converted to a monochrome image by the student-authored threshold function. Students iterate over a range of threshold values and determine which ones eliminate artifacts from noise without sacrificing too much edge detail.

**Second Lab:** For the second image processing assignment, students implement the discrete convolution operator which performs the filtering operation. Students can assume that the filter mask is of a fixed size, but extra credit was offered for implementations which operate on arbitrarily-sized masks.

The basic structure of this exercise is similar to the first, with the primary difference being the student-developed filter function. Students also use this filtering operation for an explicit smoothing step using a Gaussian smoothing mask. The filtering operation can be thought of as a stencil operation on each element in the input image.

The image filtering function performs three key tasks:

1. *Output image padding:* Since the filtering operation is similar to a stencil computation, the original image must be zero-padded in order to ensure well-defined output in the filtered image. Students must determine the appropriate amount of padding based on the filter size and add a corresponding border of zero-valued elements.
2. *Discrete convolution operation:* The convolution operation involves iterating over each element in the source image and taking the sum of the products of neighboring elements with the filter values. This requires students to write a nested loop statement with the computation using fixed offset indices of the loop index variables.

3. *Normalization:* The convolved image matrix may contain values which are outside the 8-bit intensity value range from 0 to 255, including negative values. Students must scale the values in the filtered image to the legal range, again giving them experience with problem-motivated matrix transformations.

## 4. Experience and Observations

Reviewing the lab submissions for both image processing exercises illuminated some key insights as to how students were able to integrate the different program elements to construct a complete solution. For example, the minimal solution to implement the discrete convolution operator used in image filtering requires two nested loops, to iterate over each point in the source image. The filtering operation was most commonly implemented by explicitly summing the products of a point in the filter matrix with a point in the image matrix. However, several students implemented the filtering operation with the following form:

For a $3 \times 3$ filter, `f` and a source image, `im`:

```
for i = 1,rows
  for j = 1,cols
    out(i,j) = sum(f(1,1:3).*im(i-1,j-1:j+1)) + ...
  end
end
```

This approach blends both the benefits of using a high-level vector operations such as `sum()` with traditional computational structures like loops. Indeed, these skills are exactly what we want students to take away from the course: If high-level functions or language level support can solve the problem at hand they are preferred, otherwise, use lower-level techniques to implement the algorithm needed.

### 4.1. Student Survey Results

At the end of the course, students were surveyed and reported on their experience with different aspects of the course and on the use of the edge detection exercises in particular. The questions about the image processing components were whether they felt like edge detection was understandable given their math background, they felt like they learned from the experience, if it changed their opinion of computing and using MATLAB, and if so whether it was a positive or negative change. These results are summarized in Table 1.

Overall, over 81% of the students felt that the image processing concepts related to edge detection were understandable given their math and science background. This fits with our goal to pick a problem which is approachable to students with a basic calculus background. This also indicates that some students need some more context prior to undertaking the assignment. Future classes will spend more time on the math concepts involved and how they are related to the edge detection problem.

Over 74% of students felt like they learned from the edge detection laboratories. In the department course feedback forms, some students stated that they didn't feel that the edge detection labs were relevant to their chosen discipline. While this is certainly the case for many students, it will always be difficult to pick a problem of relevance for all students given their diverse intended fields. Just under 89% of students changed their opinion of computing and the use of MATLAB as a computational tool. Of these students, 83% came to view the practice of programming and using MATLAB to solve problems in a more favorable light.

## 5. Conclusion

Introducing engineering students to computational science is an important task early in the curriculum. Early exposure to computational tools allows students to use them throughout their education and into their vocation. Exposing students to computational science early, however, implies minimal math and science prerequisites which impacts the topics and applications covered in the course.

We have developed coursework and lab exercises which use image processing as a model application domain to give students experience solving scientific problems with modern high-level computational tools. This course has

| Question | Response | Count |
|---|---|---|
| 1. How understandable is the concept of edge detection in an image given your math/science background? | not at all | 2 |
| | difficult to understand | 3 |
| | understandable | 17 |
| | straightforward | 5 |
| 2. How much do you feel like you learned through the experience of solving the edge detection problem as a real-world application of MATLAB? | not at all | 1 |
| | not very much | 6 |
| | somewhat | 11 |
| | a lot | 9 |
| 3. Did your experience with the edge detection labs change your opinion of computing and the use of MATLAB to solve real-world problems? | yes, a lot | 8 |
| | yes, a little | 16 |
| | no, not at all | 3 |
| 4. If you answered yes, how did your opinion change? Do you now view programming and MATLAB more or less favorably? | more favorably | 20 |
| | less favorably | 4 |

Table 1: Summary of Student Survey Data. ($n$ = 27)

been successfully piloted at Ohio State and has been found to be a useful tool in engaging students in computational science and helping them learn how to solve scientific problems.

Much of the course materials (slides, laboratory assignments, homework) are available at `http://www.cse.ohio-state.edu/~larkins/cse294p`.

## References

[1] G. J. Borse, Numerical Methods with MATLAB: A Resource for Scientists and Engineers, International Thomson Publishing, 1996.

[2] J. Kepner, Parallel MATLAB for Multicore and Multinode Computers, Society for Industrial and Applied Mathematics (SIAM), 2009.

[3] D. T. Kaplan, Introduction to Scientific Computation and Programming, Brooks/Cole, 2004.

[4] D. A. Forsyth, J. Ponce, Computer Vision: A Modern Approach, Prentice Hall Professional Technical Reference, 2002.

[5] J. Canny, A computational approach to edge detection, IEEE Trans. Pattern Anal. Mach. Intell. 8 (6) (1986) 679–698.

[6] J. Canny, Finding edges and lines in images, in: MIT AI-TR, 1983.