Next Steps

Resources for learning the Wolfram Language

- Wolfram Research
- Beginner/Intermediate Textbooks
- Subject Guides
- Online Resources

Wolfram Research

- Online help in Mathematica / Wolfram Language Documentation

- Wolfram, ‘Elementary Introduction to the Wolfram Language’

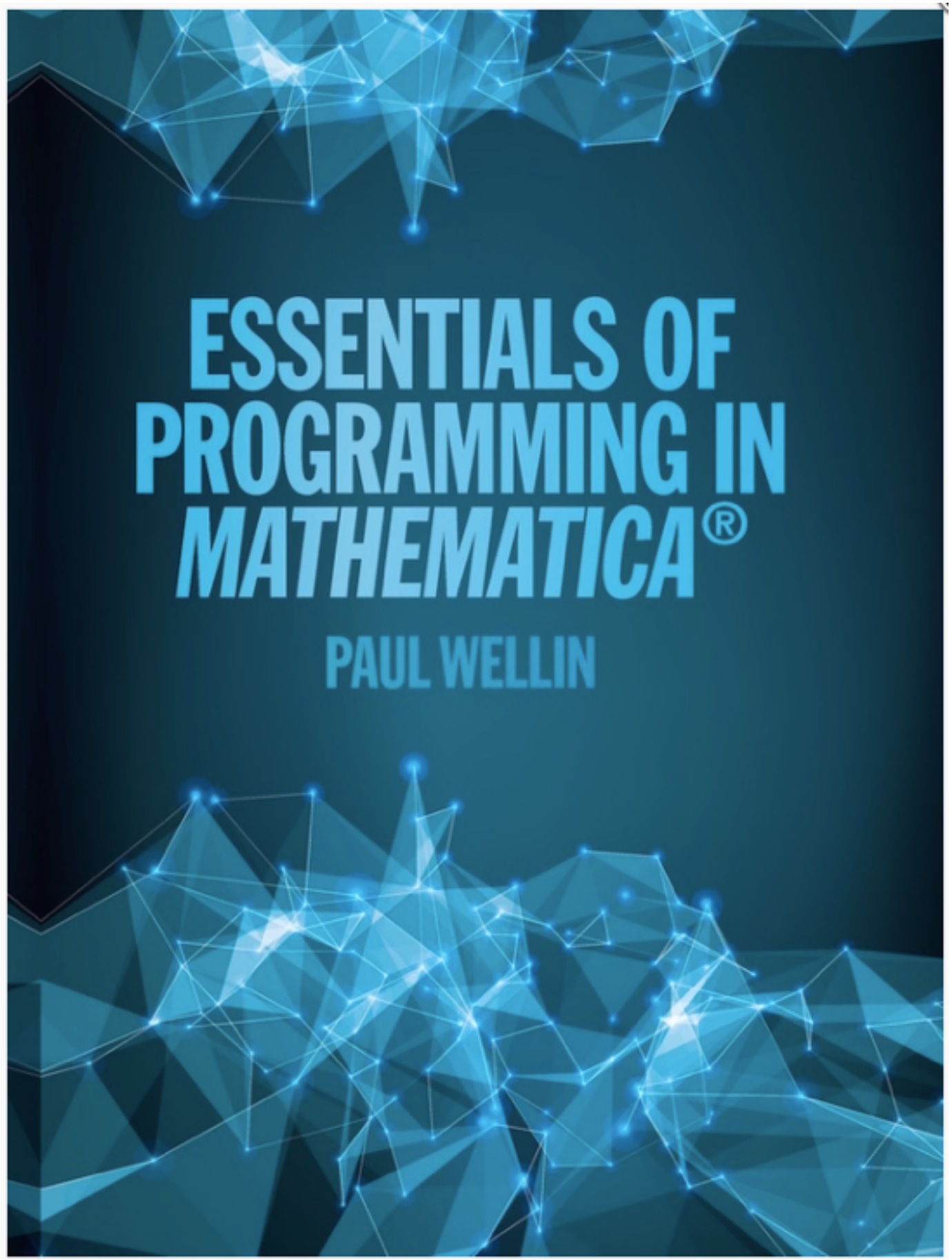
- Wolfram, ‘Fast Introduction for Programmers’

- Wolfram Programming Lab

- Wolfram Virtual Workshops & Conferences

Beginner/Intermediate Textbooks

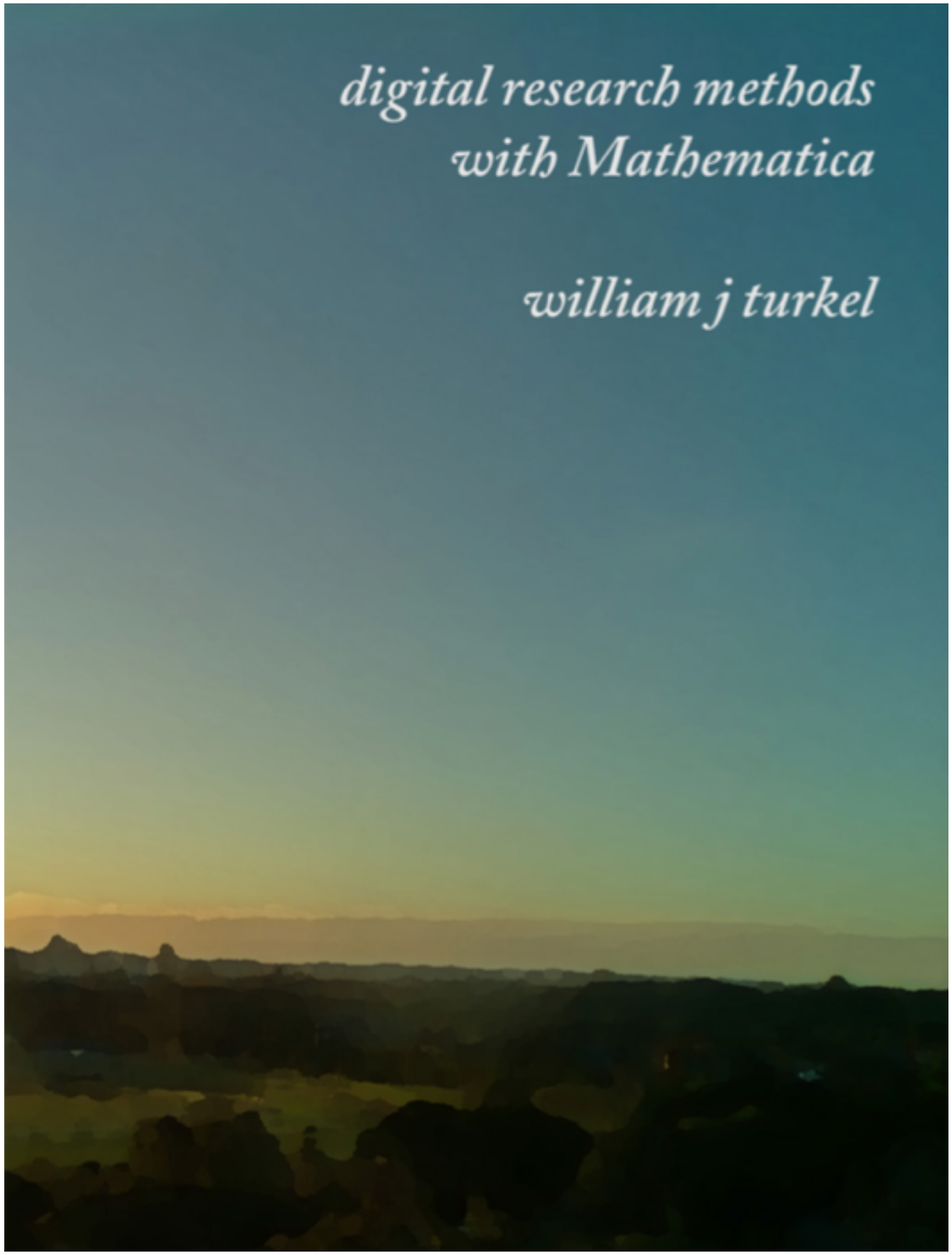
- Wolfram, ‘Elementary Introduction to the Wolfram Language’
- Wellin, ‘Essentials of Programming in Mathematica’
- Turkel, ‘Digital Research Methods with Mathematica’ & syllabus/course material



amazon.com

Kindle $45.44
Paperback $54.18

programmingmathematica.com

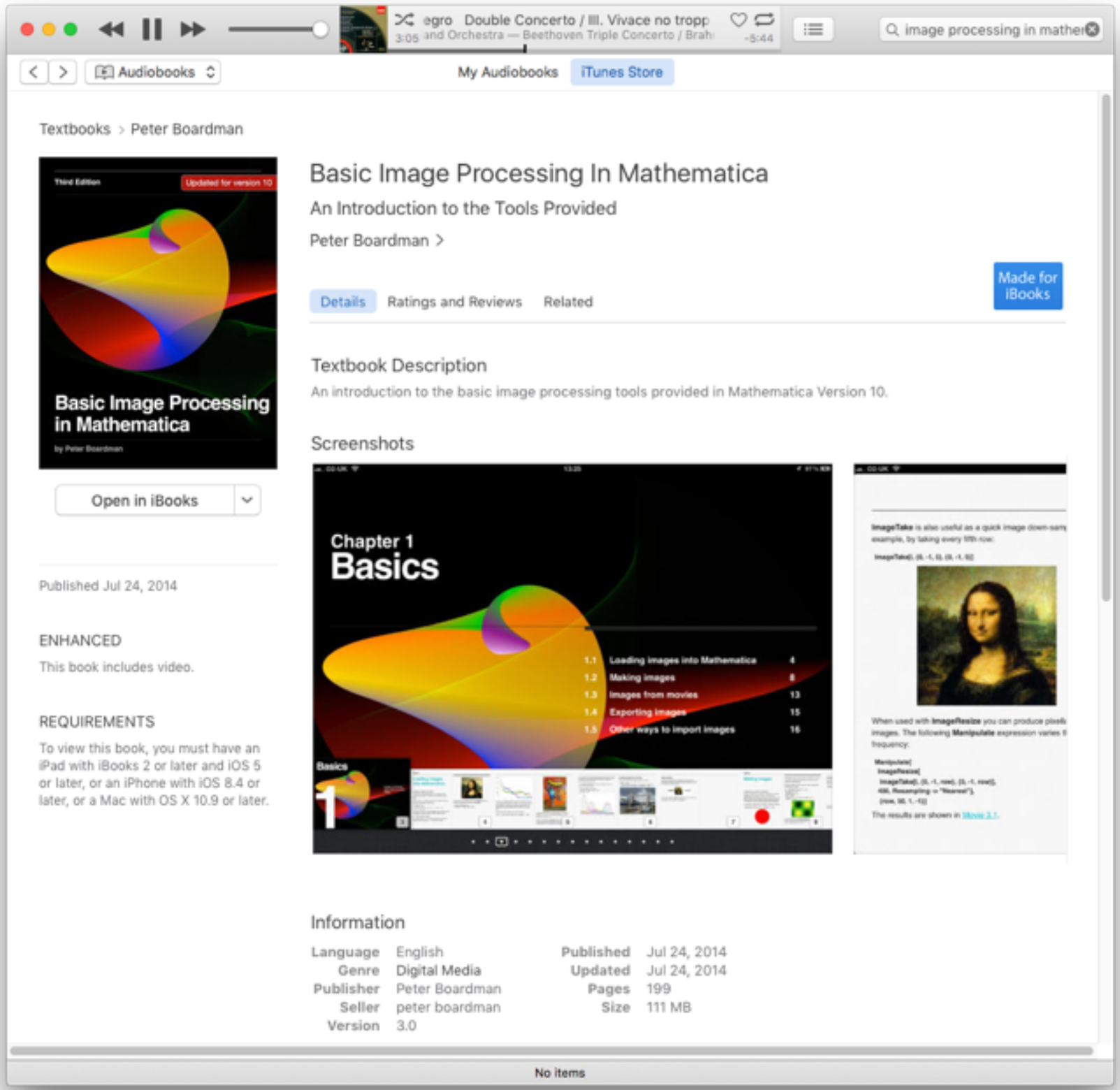


free!

williamjturkel.net

Subject Guides

- Boardman, ‘Basic Image Processing in Mathematica’
- Saquib, ‘Mathematica Data Visualization’
- Mangano, ‘Mathematica Cookbook’
- Introductory Image Processing textbooks



Apple iBooks

free!

Chapter 1
Basics

Section 1

Loading images into Mathematica

Loading images from disk

Let's start by pointing *Mathematica* to a directory which contains some images – a folder with a few JPEG and PNG images is ideal. Here's an example on my computer:

```
SetDirectory["~/projects/mathematica/imageprocessing"];
```

To list the JPEG files in the current directory:

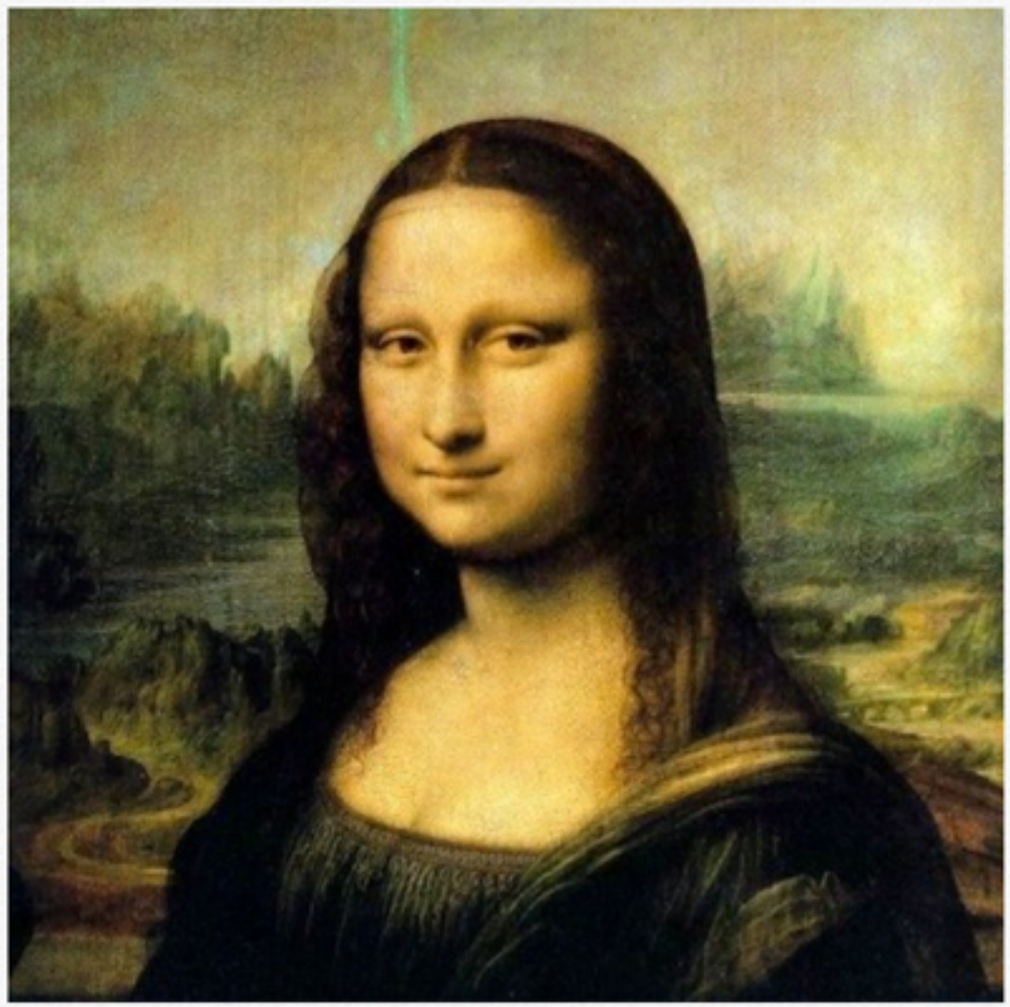
```
FileNames["*jpg"]
```

```
{"London_2010_Tower_Bridge.jpg", "monalisa.jpg",
"Robot_banana.jpg"}
```

Use the **Import** function to load the image and store it in a symbol, such as **i** (which is easy to type):

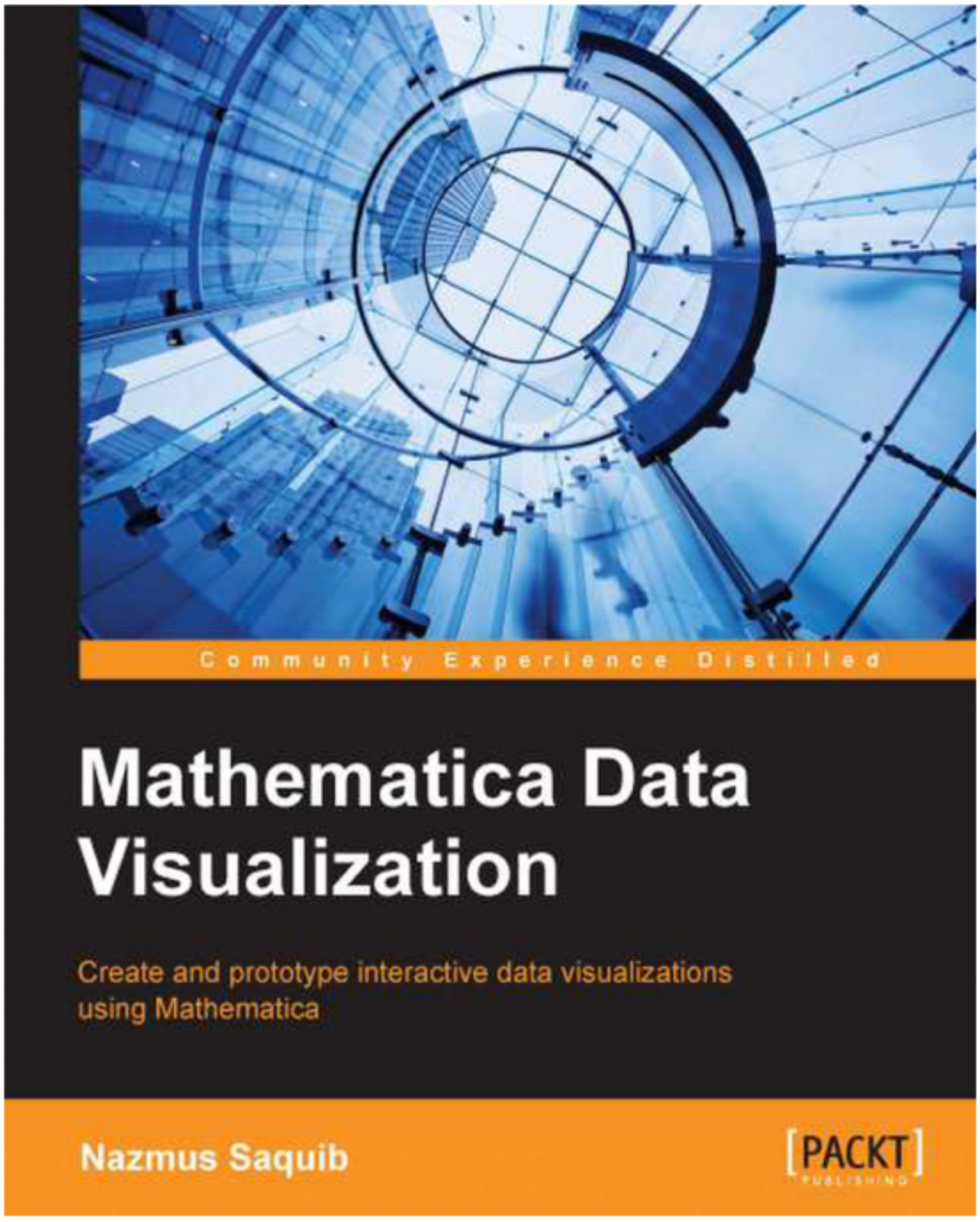
```
i = Import["monalisa.jpg"]
```

The image appears in your notebook (unless you finish the expression with a semicolon — this imports the image but doesn't display it).



To see what we've loaded, we can use **ImageLevels** on the symbol **i**, and display the results using **ListPlot**:

```
ListPlot[ImageLevels[i],
 PlotStyle -> {Red, Green, Blue}]
```



amazon.com

Kindle $13.38
Paperback $34.99

PairedHistogram

The two sets of numbers can be compared using PairedHistogram too. The syntax for `PairedHistogram` is as follows:

`PairedHistogram[`**`data`**$_1$**`, data`**$_2$**`, bspec, options`**`…]`

Let's compare sets of numbers that use `PairedHistogram` using the following code snippet:

```
(* paired histograms *)
PairedHistogram[ data[[;;,4]],
data[[;;,5]], 20 ]
```

The number of bins is set to 20 again. This output can be seen in the following screenshot:

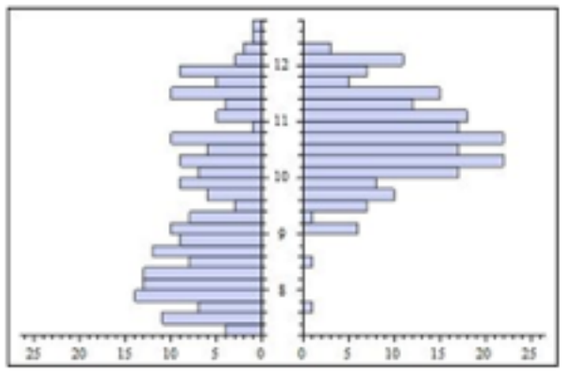


Figure 4.3 A PairedHistogram of bottom and top margins

The paired histogram plots both the histograms on two sides for bin-specific comparison.

Histogram3D

This function can be used to visualize the distribution of random variables that have two dimensions. The syntax for this function is as follows:

`Histogram3D[{ {`x_1`,`y_1`}, {`x_2`,`y_2`}, … }, bspec, options…]`

Here, each element of the list is now a list of two numbers. Let's demonstrate the function using some randomly generated data using the following code:

```
(* 3d histogram example *)
data1 = RandomVariate[
NormalDistribution[0,1],
{1000,2}];
data2 = RandomVariate[
NormalDistribution[3,0.7],
{1000,2}];
Histogram3D[{data1,data2}]
```

Here, `RandomVariate` is a function that creates a set of random numbers based on the distribution entered in its first argument. The second argument seeks the dimension of the randomly generated list. For both the data series, we have chosen the normal distribution, one with a mean of 0 and standard deviation of 1, the other with a mean of 3 and standard deviation of 0.7 (thus, narrower and taller). Both datasets are of dimension 1,000 by 2, which means there are 1,000 rows, each containing 2 random numbers. Next, we pass both the datasets to `Histogram3D`. Figure 4.4 shows the result as follows:

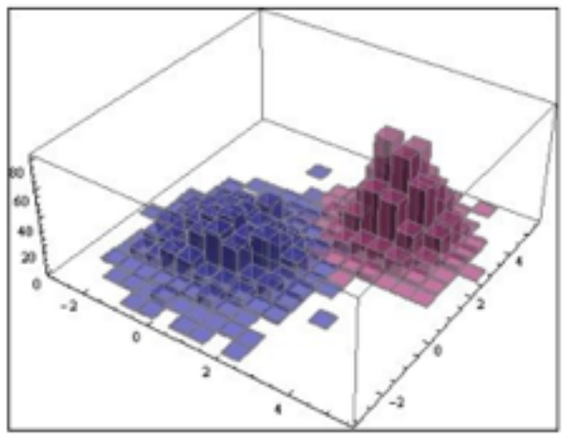
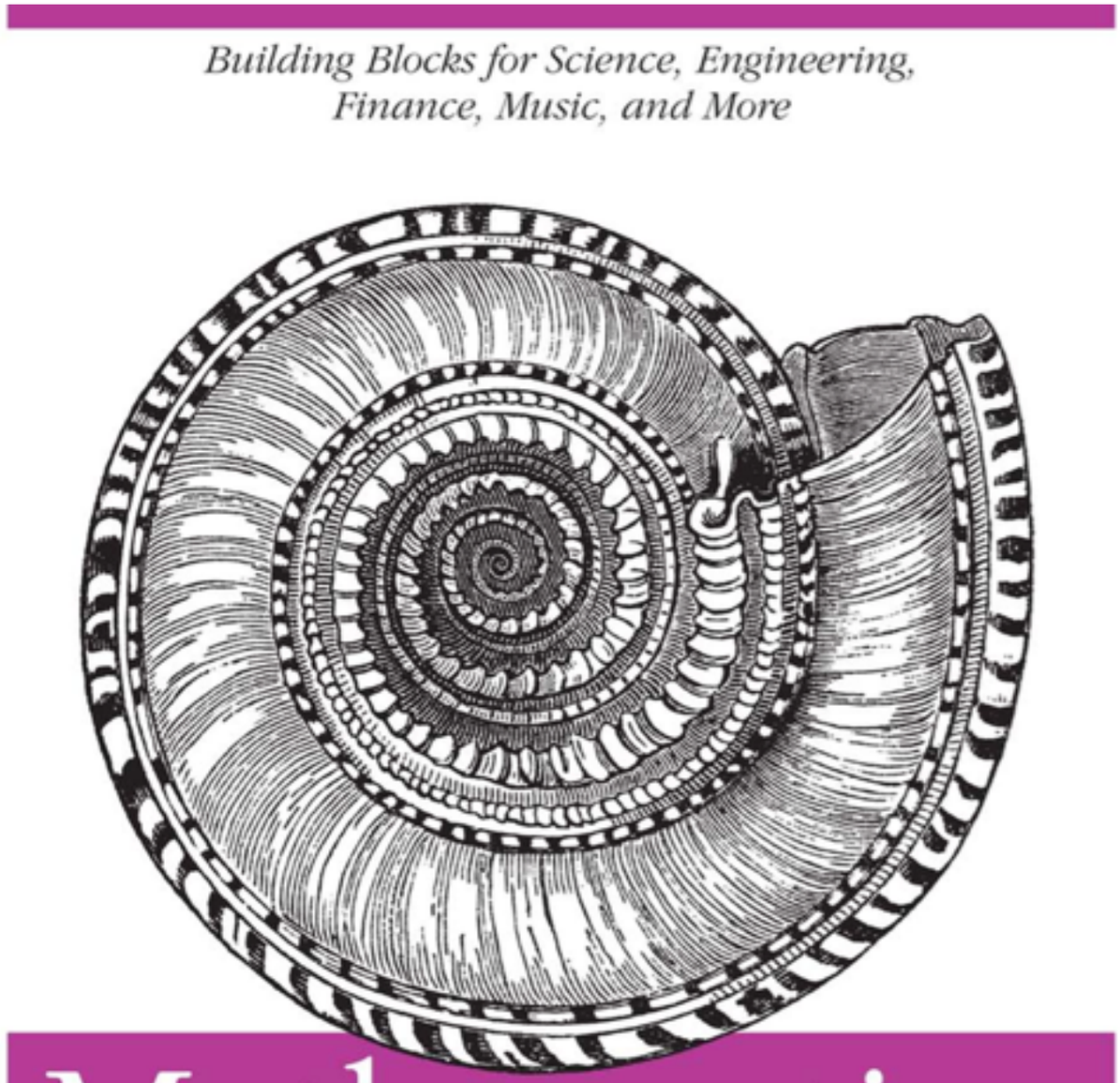


Figure 4.4 A 3D histogram of two randomly generated datasets

The resulting figure can be tilted and rotated in Mathematica to explore the shape of the distributions.



amazon.com

Kindle $13.38
Paperback $34.99

See Also

`Nest` is discussed in Recipe 2.11.

5.5 Matching and Searching Text

Problem

You want to determine if a string contains a pattern and at what positions.

Solution

Use `StringMatchQ[string,pattern]` to determine if a string matches a pattern.

```
In[92]:= StringMatchQ["1234", NumberString]
Out[92]= True
```

Here I show a match on multiple strings with a pattern that is predicated.

```
In[93]:= StringMatchQ[{"1234", "1237"}, p : NumberString /; OddQ[FromDigits[p]]]
Out[93]= {False, True}
```

Use `StringFreeQ[string,pattern]` to determine if a string does not match a pattern.

```
In[94]:= StringFreeQ[{"1234", "abcde"}, p : NumberString]
Out[94]= {False, True}
```

Use `StringPosition[string,pattern]` to find the integer offsets of matches. The default behavior is to search for all occurrences of the pattern (i.e., `Overlaps → True`).

```
In[95]:= StringPosition["1234abcd54321", NumberString]
Out[95]= {{1, 4}, {2, 4}, {3, 4}, {4, 4},
          {9, 13}, {10, 13}, {11, 13}, {12, 13}, {13, 13}}
```

With `Overlaps → False`, you only get matches on substrings that don't share characters with prior matches.

```
In[96]:= StringPosition["1234abcd54321", NumberString, Overlaps → False]
Out[96]= {{1, 4}, {9, 13}}
```

Discussion

There are a lot of neat applications for an integrated dictionary.

Crossword puzzles

Here is how you might cheat at a crossword puzzle. Say you have three letters of a six-letter word and the clue is "51 down: unkeyed."

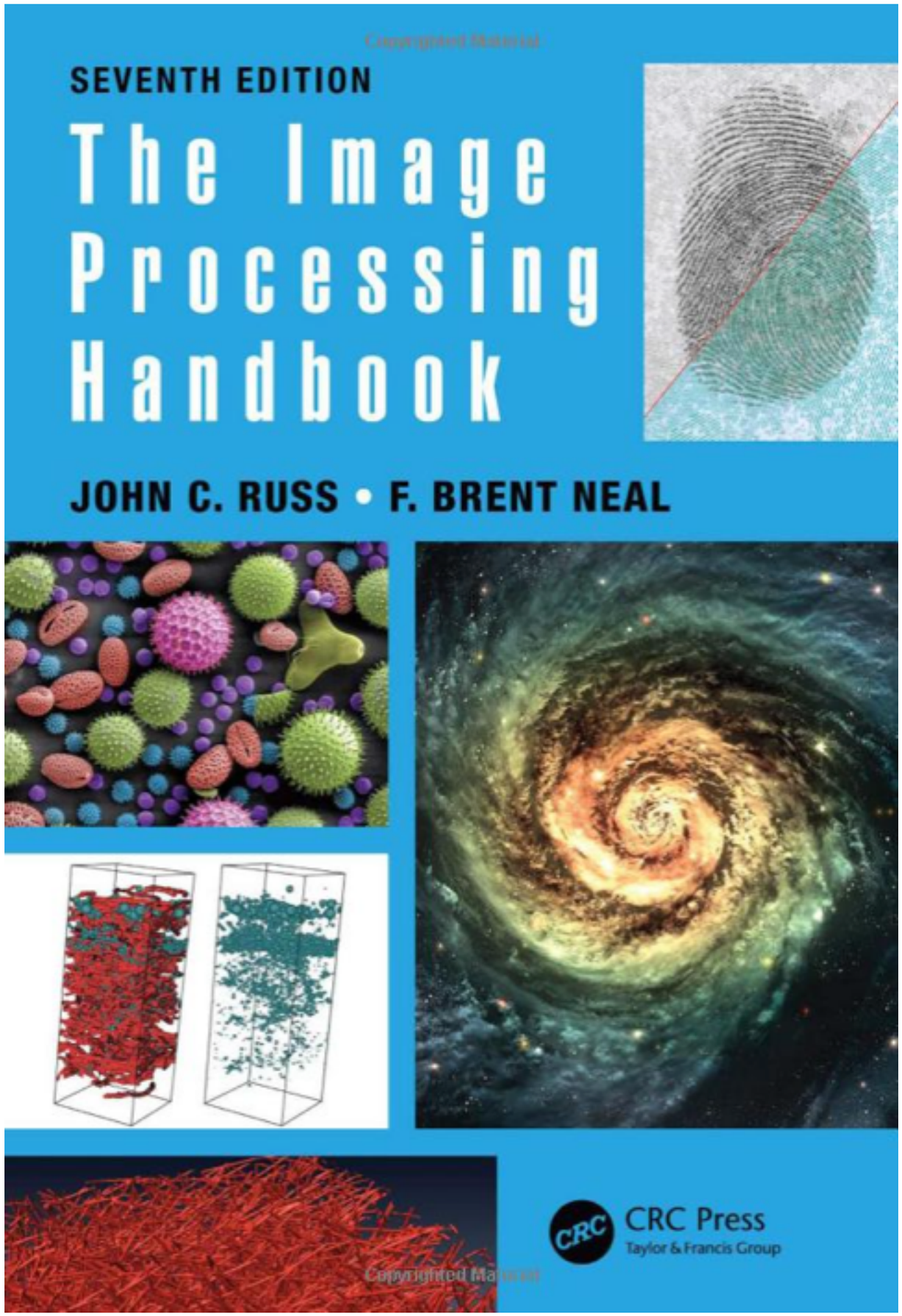
```
In[118]:= DictionaryLookup["a" ~~ _ ~~ "o" ~~ _ ~~ _ ~~ "l"]
Out[118]= {amoral, atonal, avowal}
```

Ah, *atonal* sounds right (pun intended)!

Anagrams

You can also help your second grader impress the teacher on that November worksheet for finding all the words you can make out of the letters in "Thanksgiving" (i.e., anagrams). Here we use a pattern containing all combinations of the letters in "thanksgiving" and an extra constraint function to ensure letters are contained by their availability (count). Strictly speaking, an anagram must use all the letters of the input, but I ignore that here.

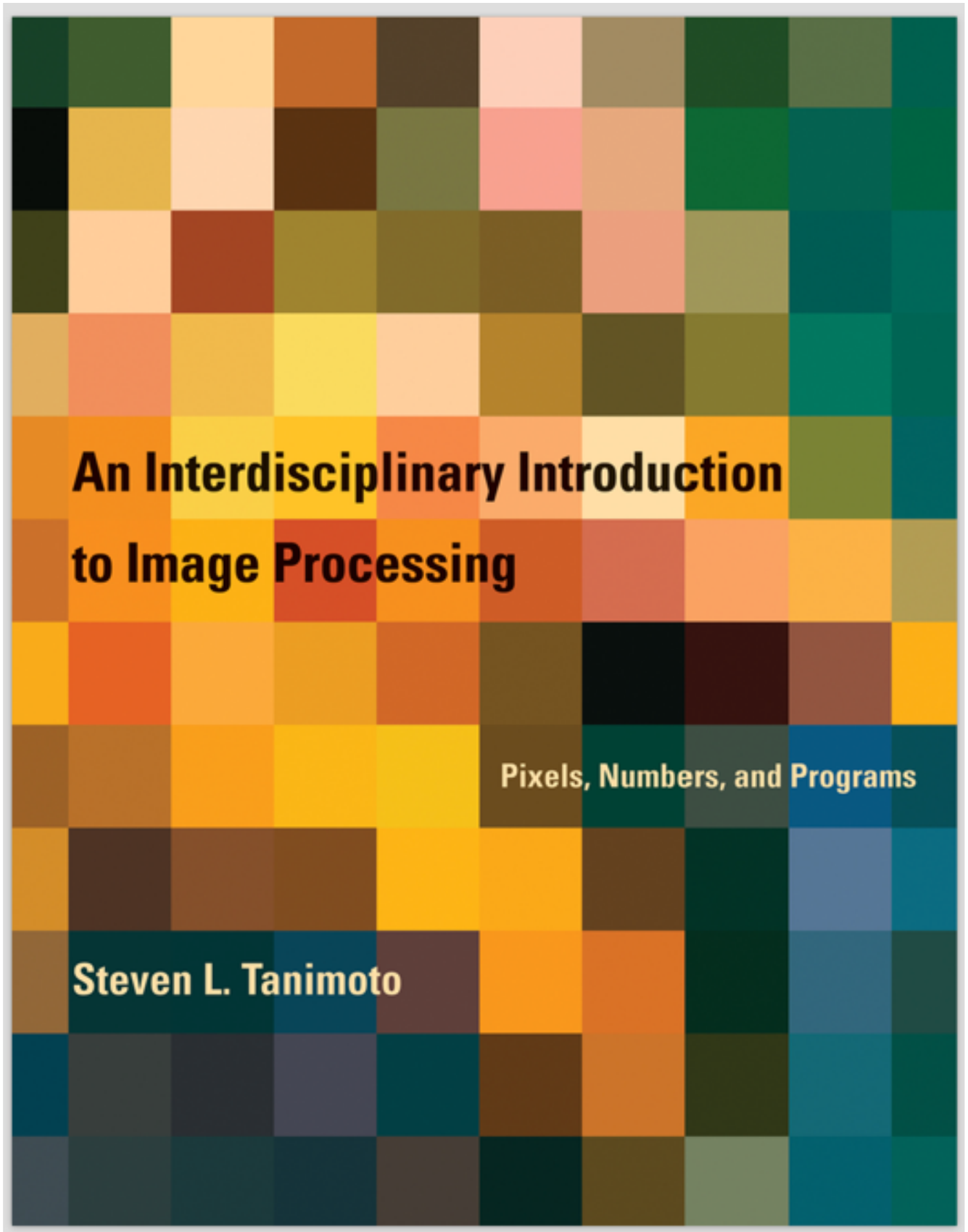
```
In[119]:= thanksgivingQ[word_] := StringCount[word, "t"] < 2 &&
            StringCount[word, "h"] < 2 && StringCount[word, "a"] < 2 &&
            StringCount[word, "n"] < 3  && StringCount[word, "k"] < 2 &&
            StringCount[word, "s"] < 2 && StringCount[word, "g"] < 3 &&
            StringCount[word, "i"] < 3 && StringCount[word, "v"] < 2 ;

In[120]:= DictionaryLookup[
           word : ("t" | "h" | "a" | "n" | "k" | "s" | "g" | "i" | "v") .. /;
            thanksgivingQ[word], IgnoreCase → True ]
```



amazon.com

Kindle $160.33
Hardcover $190.07

(google for
cheaper editions..)



amazon.com

Kindle $32.78
Hardcover $44.50

Online Resources

- Our workshop's Slack forum
- Mathematica Stack Exchange
- Wolfram Community
- Wolfram YouTube channel

In short..

- **For review in the next 2 weeks?** Wolfram's 'Elementary Introduction' or 'Fast Introduction for Programmers' (free)

- **Over the summer?** Turkel's 'Digital Research Methods' (free)

- **Help & inspiration?** Wolfram Community forum (free)

- **In-depth treatment?** Wellin's 'Essentials of Programming in Mathematica' ($$ but worth it..)

- **For Mathematica Newbies?** Wolfram Programming Lab (free)

Stay in touch!

Twitter: @analyse_humdata
Website: http://shift-enter.org
Slack: http://shift-enter-2016.slack.com