Guidelines for Numerical Codes

Hans Petter Langtangen[1,2]

[1]Simula Research Laboratory
[2]Univiversity of Oslo

May 28, 2010

Abstract

A common conception is that simple scientific computing scripts implemented in Matlab and Python are very similiar - almost identical. However, practice observed by this author shows that students (and teachers) tend to make software with bad design in Matlab, while the design improves significantly when they use Python. This note points to the importance of a software design that acknowledges the generic nature of mathematical algorithms.

This document is also available in HTML format.

Motivation

Many science courses now have examples and exercises involving implementation and application of numerical methods. How to structure such numerical programs has, unfortunately, received little attention. Students and teachers occasionally write programs that are too tailored to the problem at hand instead of being a good starting point for future extensions. A key issue is to split the program into functions and to implement general mathematics in general functions applicable to many problems. We shall illustrate this point through a case study and briefly discuss the merits of different types of programming styles.

Exercise

Integrate the function $g(t) = \exp(-t^4)$ from -2 to 2 using the Trapezoidal rule, defined by

$$\int_a^b f(x)dx \approx h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+ih)\right), \quad h = (b-a)/n \qquad (1)$$

Solution 1: Minimalistic Matlab

The simplest possible program may look as follows in Matlab:

```
a = -2; b = 2;
n = 1000;
h = (b-a)/n;
s = 0.5*(exp(-a^4) + exp(-b^4));
for i = 1:n-1
    s = s + exp(-(a+i*h)^4);
end
r = h*s;
r
```

The solution is minimalistic and correct. Nevertheless, this solution has a pedagogical and software engineering flaw: a special function $\exp(-t^4)$ is merged into a general algorithm (1) for integrating an arbitrary function $f(x)$.

Solution 2: Matlab with functions

A successful software engineering practice is to use functions for splitting a program into natural pieces, and if possible, make these functions sufficiently general to be reused in other problems. In the present problem we should strive for the following principles:

- The Trapezoidal rule is implemented in a separate Python function taking a general mathematical function $f(x)$ as argument, together with the input data for the problem: the integration limits a and b and the numerical resolution parameter n.
- The special $g(t)$ formula is implemented in a separate Python function.
- A main program solves the specific problem in question by calling the general algorithm (the Trapezoidal integration function) with the special data of the given problem ($g(t)$, $a = -2$, $b = 2$, $n = 1000$).

Let us apply these desirable principles in a Matlab context. User-defined Matlab functions must be placed in separate files. This is sometimes found annoying, and therefore many programmers tend to avoid functions. In the present case, we should implement the Trapezoidal method in a file `Trapezoidal.m` containing

```
function r = Trapezoidal(f, a, b, n)
% TRAPEZOIDAL Numerical integration from a to b
% with n intervals by the Trapezoidal rule
f = fcnchk(f);
h = (b-a)/n;
s = 0.5*(f(a) + f(b));
for i = 1:n-1
    s = s + f(a+i*h);
end
r = h*s;
```

The special $g(t)$ function is implemented in a separate file `g.m`:

```
function v = g(t)
v = exp(-t^4)
end
```

Finally, we create a main program `main.m`:

```
a = -2; b = 2;
n = 1000;
result = Trapezoidal(@g, a, b, n);
disp(result);
exit
```

Solution 3: Standard Python

Both Solution 1 and Solution 2 are readily implemented in Python. However, functions in Python do *not* need to be located in separate files to be reusable, and therefore there is no psychological barrier to put a piece of code inside a function. The consequence is that a Python programmer is more likely to go for Solution 2. The relevant code can be placed in a single file, say `main.py`, looking as follows:

```
def Trapezoidal(f, a, b, n):
    h = (b-a)/float(n)
    s = 0.5*(f(a) + f(b))
    for i in range(1,n,1):
        s = s + f(a + i*h)
    return h*s

from math import exp  # or from math import *
def g(t):
    return exp(-t**4)

a = -2;  b = 2
n = 1000
result = Trapezoidal(g, a, b, n)
print result
```

Discussion

Looking at the simple exercise isolated, all three solutions produce the same correct mathematical result and are hence equivalent mathematically. However, the nature of this exercise is that we want to solve a special problem by a general mathematical method. This is often the case when mathematics is applied to practical problems. The software should reflect this division between the general part and the special part of the given problem of two reasons.

First, the division is important for the understanding the general nature of mathematical methods and how general methods can be used to solve a special problems. Second, the implementation of the general part, here the Trapezoidal rule, can be reused in many other problems.

We may say that the the first reason is comes from the philosophy of mathematics and science, while the second reason is motivated by the practical aspect of reducing future coding efforts by relying on a resuable, general, and working

function. This aspect is the basis of a fundamental software engineering practice: *programs should consist of general pieces (functions) that can be reused without modifications to solve other problems.* The importance of this philosopy becomes obvious when we extend the problem as described below.

Another point worth mentioning is the way we can transfer functions to functions as an argument. The argument `f` in the Python function `Trapezoidal` is treated as a standard variable, and `f` is called by simply writing `f(x)`. In Matlab and other languages, functions that are argument to other functions require special, often somewhat "ugly", syntax. This aspect, together with the ease of writing functions, make the Python solution slightly preferable in the present case.

We also emphasize that the $g(t)$ formula is implemented in a separate Python function such that the formula can be reused in other occations, for instance, when integrating $g(t)$ by an alternative numerical integration rule. In many problems, the formula is much more complicated than the one used here, and it is important to have a single, well-tested implementation of the formula.

Readers may also realize that the nature of programming (combined with sound programming habits) helps to increase the understanding of mathematics through the clear distinction between general methods and and a specialized problem. Understanding the generality of methods also requires an understanding of abstractions in mathematics. Programming exercises therefore enforce a stronger focus on abstractions in general. All these arguments boil down to Kristen Nygaard's famous three words: "Programming is understanding"!

Extended Exercise

Compute the following integrals with the Midpoint rule, the Trapezoidal rule, and Simpson's rule:

$$\begin{aligned}
\int_0^{\pi} \sin x\, dx &= 2, \\
\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2} dx &= 1, \\
\int_0^1 3x^2 dx &= 1, \\
\int_0^{\ln 11} e^x dx &= 10, \\
\int_0^1 \frac{3}{2}\sqrt{x} dx &= 1.
\end{aligned}$$

For each integral, write out a table of the numerical error for the three methods using a n function evaluations, where n varies as $n = 2^k + 1$, $k = 1, 2, ..., 12$.

Discussion

In the extended problem, Solution 1 is obviously inferior because we need to apply, e.g., the Trapezoidal rule to five different integrand functions for 12 different n values. Then it makes sense to implement the rule in a separate function that can be called 60 times.

Similarly, a mathematical function to be integrated is needed in three different rules, so it makes sense to isolate the mathematical formula for the integrand in a function in the language we are using.

We can briefly sketch a compact and smart Python code, in a single file, that solves the extended problem:

```
def f1(x):
    return sin(x)

def f2(x):
    return 1/sqrt(2)*exp(-x**2)

...

def f5(x):
    return 3/2.0*sqrt(x)

def Midpoint(f, a, b, n):
    ...

def Trapezoidal(f, a, b, n):
    ...

def Simpson(f, a, b, n):
    ...

problems = [(f1, 0, pi),  # list of (function, a, b)
            (f2, -5, 5),
            ...
            (f3, 0, 1)]

methods = (Midpoint, Trapezoidal, Simpson)
result = []
for method in methods:
    for func, a, b in problems:
        for k in range(1,13):
            n = 2**k + 1
            I = method(func, a, b, n)
            result.append((I, method.__name__, func.__name__, n))

# write out results, nicely formatted:
for I, method, integrand, n in result:
    print '%-20s, %-3s, n=%5d, I=%g' % (I, method, integrand, n)
```

Note that since everything in Python is an object that can be referred to by a variable, it is easy to make a list `methods` (list of Python functions), and a list `problems` where each element is a list of a function and its two integration limits. A nice feature is that the name of a function can be extracted as a string in the function object (`name` with double leading and trailing underscores).

To summarize, Solution 2 or 3 can readily be used to solve the extended problem, while Solution 1 is not worth much. In courses with many very simple

exercises, solutions of type 1 will appear naturally. However, published solutions should employ approach 2 or 3 of the mentioned reasons, just to train students to think that *this is a general mathematical method that I should make reusable through a function.*

Solution 4: A Java OO Program

Introductory courses in computer programming usually employ the Java language and emphasize object-oriented programming. Many computer scientists argue that it is better to start with Java than Python or (especially) Matlab. But how well is Java suited for introductory numerical programming?

Let us look at our first integration example, now to be solved in Java. Solution 1 is implemented as a simple `main` method in a class, with a code that follows closely the displayed Matlab code. However, students are in a Java course trained in splitting the code between classes and methods. Therefore, Solution 2 should be an obvious choice for a Java programmer. However, it is not possible to have stand-alone functions in Java, functions must be methods belonging to a class. This implies that one cannot transfer a function to another function as an argument. Instead one must apply the principles of object-oriented programming and implement the function argument as a reference to a superclass. To call the "function argument", one calls a method via the superclass reference. The code below provides the details of the implementation:

```
import java.lang.*;

interface Func {  // superclass for functions f(x)
    public double f (double x);  // default empty implementation
}

class f1 implements Func {
    public double f (double t)
    { return Math.exp(-Math.pow(t, 4)); }
}

class Trapezoidal {
    public static double integrate
           (Func f, double a, double b, int n)
    {
        double h = (b-a)/((double)n);
        double s = 0.5*(f.f(a) + f.f(b));
        int i;
        for (i = 1; i <= n-1; i++) {
            s = s + f.f(a+i*h);
        }
        return h*s;
    }
}

class MainProgram {
    public static void main (String argv[])
    {
        double a = -2;
        double b = 2;
        int n = 1000;
```

```
        double result = Trapezoidal.integrate(f, a, b, n);
        System.out.println(result);
    }
}
```

From a computer science point of view, this is a quite advanced solution since it relies on inheritance and true object-oriented programming. From a mathematical point of view, at least when compared to the Matlab and Python versions, the code looks unnecessarily complicated. Many introductory Java courses do not cover inheritance and true object-oriented programming, and without mastering these concepts, the students end up with Solution 1. On this background, one may argue that Java is not very suitable for implementing this type of numerical algorithms.

Conclusions and Recommendations

Simple exercises have pedagogical advantages, but some disadvantages with respect to programming, because the programs easily become too specialized. In such cases, the exercise may explicitly ask the student to divide the program into functions. This requirement can be motivated by an extended exercise where a piece of code are needed many times, typically that several methods are applied to several problems.

Especially when using Matlab, students may be too lazy to use functions when this is not explicitly required.

Although Java is very well suited for making large programs systems, Java code for simpler numerical problems, where one wants to transfer functions to other functions, looks as an overkill compared with Matlab, Python, C++, and Fortran implementations.

Recommendations.

1. Identify general and special parts of the mathematical problem to be solved.
2. Implement the general parts in functions that can be reused to solve similar problems. Consider carefully the argument lists of the functions.
3. Isolate the special features of the problem in a separate "main" function or a main program. The best solution is to isolate the special features in a file separate from the general functions.