Grading Rubric for Math 5610

Software Manual

Joe Koebbe
Department of Mathematics and Statistics
Utah State University
Logan, Utah 84322-3900

Fall Semester 2016[1]

[1]Last Revised: November 2, 2016

Contents

1 Instructions for the Title Page

Your software manual should have a cover page much like the cover page on this document. The following provides the information you should include on any cover page. You can use the cover page on this document as a template if you like.

---

1. A descriptive title for the document.
2. Your name.
3. The date the software manual is complete (due date in the class).
4. The course you are producing the document for.
5. The last revision date of the document.

2 Table of Contents

Your software manual should include a thorough table of contents. Each section or chapter of the document should have an entry in the table of contents. It is fairly easy to edit a LaTeX document to include the entries in a table of contents. I think I will have an additional lecture in the very near future on how to use latex to format mathematics and scientific documents using LaTeXfor those that are interested. Your table of contents should go down to the level of the individual subroutines/functions/methods for your document. For a template, see the table of contents in this document.

3 Instructions for the Introduction

Instructions for creating an introduction for the software manual will be given in this section. You will need to have an overview of the computer codes written for this class. You should not discuss details and comparisons of the codes writteen during the semester. Instead, you should summarize the content covered in the course and provide general relationships between major content areas presented in class and the sections in your software manual.

Introduction of Software Manual

The following will be used in assessing the introduction to your software manual. You should take these as general guidelines and if you have questions about the introduction to software manual ask your instructor.

1. The student demonsstrates a clear understanding of the content in the software manual.

   (a) **Not Addressed:** There is no discussion of the organization of the software in the manual. There is no discussion of the purpose of the project or the software included in the manual.

   (b) **Novice:** There is a basic summary of the content of the course in terms of various codes and routines that have been developed during the semester. Maybe something like a chapter by chapter summary of what is included in the chapter. No overall puspose for the project is included in the introduction.

   (c) **Intermediate:** Discussion of relationships between software in the various chapters is included in the introduction along with a brief summary of the purpose of the project is included.

   (d) **Expert:** Examples defining relationships bewtween chapters and software routines is included. A thorough discussion of the purpose of the project as it relates this and other classes and possible work is included in the introduction.

To get the highest marks on this project you should aim for the expert level. Please note that the introduction should be no more than a few pages. Be concise in your comments and overview.

4 Instructions for Software Documentation

Each and every algorithm should be included in this software manual. Divide the sections for the different competing methods for solving general problems. For example, there should be a section for the Bisection method and Newton's method in a section that addresses the general problem of finding roots for nonlinear fnuctions of a single variable.

Software Entries

The following will be used in assessing the sections that document the various codes written during the semester. You should take these as general guidelines and if you have questions about the introduction to software manual ask your instructor.

1. The student demonsstrates a clear understanding of the general problem being solved by the algorithm represented by the code.

   (a) **Not Addressed:** There is no discussion of the general problem or relationship to the algorithm that is represented by the computer code.

   (b) **Novice:** There is a basic summary of the code and and a small number of comment lines embedded in the code.

   (c) **Intermediate:** The code representing the algorithm is well documented with the input and output variables clearly defined. There is little or no comparison to other methods or discussion about the limitations of the code you have written. The student does not include examples of how to use the code and verification of how how the code works on test problems.

   (d) **Expert:** The code is well documented with clear explanations of inputs and outputs. There is a discussion of how the algorithm works and verification that it works on examples. There is a discussion on how this code fits into the discussion of how to approximately solve the general problem with limitations on the code and algorithm.

To get the highest possible grade aim for the expert level defined above. Here again, I will grade on on how concise the descriptions are. Do not include programs that call the software documented in these sections. Reserve these until an appendix at the end of the software manual.

Also note that you will need to have a section for the basic methods you have written for the class. For example, the routines that compute the vector norms of arrays and such. These do not solve any general problems. However, they are useful in many of the codes that you have written for the approximate solution of a number of general problems. The basic methods as just described can be in a section at the beginning or at the end or in an appendix.

5 Instructions for Citations in the Document

It is likely that you will need to cite other books, articles, etc. For example, you may want to refer to a certain page in the textbook to describe where the algorithm is defined. Make sure you provide a thorough set of citations. As a concrete example consider the algorithm for Newton's method on page 116 of [1]. Make sure you cite any algorithms or mathematics that you use in explanations.

6 An Example of How to Extract Codes:

During the course I wrote a code that has been posted on formatting and documenting code written during the semester. I will give you an idea of how to extract a couple of methods for inclusion in your software manual. So, here goes. One of the first test codes you wrote was to compute the machine precision of for your computational environment. The code I wrote was the following:

```
c
c coding language:    Fortran 77
c
c ------------------------------------------------------------------------------
c
c written by:         Joe Koebbe
c date written:       Sept 28, 2014
c written for:        Problem Set 1
c course:             Math 5610
c
c purpose:            Determine a machine epsilon for the computers I would like
c                     work on computationally. The code contains 2 subroutines.
c
c                     smacsps - returns the single precision value for machine
c                               precision
c                     dmacsps - returns the double precision value for machine
c                               precision
c
c ------------------------------------------------------------------------------
c
      program main
c
c do the work in double precision
c -------------------------------
c
      real sval
      real*8 dval
c
c single precision test
c ---------------------
c
      call smaceps(sval, ipow)
      print *, ipow, sval
      call dmaceps(dval, ipow)
      print *, ipow, dval
c
c done
c ----
c
```

```
      stop
      end
c
c single precision computation of machine precision
c -------------------------------------------------
c
      subroutine smaceps(seps, ipow)
c
c set up storage for the algorithm
c --------------------------------
c
      real seps, one, appone
c
c initialize variables to compute the machine value near 1.0
c ----------------------------------------------------------
c
      one = 1.0
      seps = 1.0
      appone = one + seps
c
c loop, dividing by 2 each time to determine when the difference between one and
c the approximation is zero in single precision
c ---------------------------------------------
c
      ipow = 0
      do 1 i=1,1000
         ipow = ipow + 1
c
c update the perturbation and compute the approximation to one
c ------------------------------------------------------------
c
        seps = seps / 2
        appone = one + seps
c
c do the comparison and if small enough, break out of the loop and return
c control to the calling code
c ---------------------------
c
        if(abs(appone-one) .eq. 0.0) return
c
    1 continue
c
c if the code gets to this point, there is a bit of trouble
c ---------------------------------------------------------
c
      print *,"The loop limit has been exceeded"
c
c done
```

```
c ----
c
      return
      end
c
c double precision computation of machine precision
c -------------------------------------------------
c
      subroutine dmaceps(deps, ipow)
c
c set up the calculation so that the comparison is done at 1.0
c ------------------------------------------------------------
c
      real*8 deps, one, appone
c
c set some constants for work near 1.d0
c -------------------------------------
c
      one = 1.d0
      deps = 1.d0
      appone = one + deps
c
c loop over, dividing by 2 each time to determine when the difference between
c one and the approximation is zero to finite precision
c -----------------------------------------------------
c
      ipow = 0
      do 1 i=1,1000
         ipow = ipow + 1
c
c update the perturbation and compute the approximation
c -----------------------------------------------------
c
        deps = deps / 2
        appone = one + deps
c
c do the comparison
c -----------------
c
        if(abs(appone-one) .eq. 0.d0) return
c
    1 continue
c
c done
c ----
c
      return
      end
```

There are two routines that can be extracted from the code that I would want to include in my software manual. These are:

```
subroutine smaceps(seps, ipow)

subroutine dmaceps(deps, ipow)
```

Each of these can be documented in a separate section. The following two sections will document these codes. Note that the two routines will appear in their own subsection of the software manual.

7 Basic Computational Routines

In this section of the software manual a number of basic computational routines that support the rest of the computational coding for approximately solving the problems discussed in the course. For example, routines that compute the norms of vectors, machine precision, and a number of other basic ccmputations that can be used as building blocks for the algorithms discussed in the course.

7.1 Machine Precision Routines

In this subsection routines that compute the machine epsilon are documented. The routines are typically used prior to doing calculations to allow the user to determine the smallest number available in single and double precision that will determine the best possible error in approximations that algorithms produce.

7.1.1 smaceps

Description: This routine is used to compute the machine epsilon in sincle precision for a computer. The code divides a remainder by two until the result added produces no change in the value of the variable one containing the the number 1.

Input:

1. float *seps* - variable passed in to hold the smallest number computed
2. integer *ipow* - the power of two that gives the number of iterations through computation. this is computed internally and the value is returned.

Output:

1. float *seps* - variable passed back that holds the smallest number computed
2. integer *ipow* - the power of two that holds the number of iterations through computation. this is computed internally and the value is returned.

Code Written:

```
c
c single precision computation of machine precision
c -------------------------------------------------
c
      subroutine smaceps(seps, ipow)
c
c set up storage for the algorithm
c --------------------------------
c
      real seps, one, appone
c
c initialize variables to compute the machine value near 1.0
c ----------------------------------------------------------
c
      one = 1.0
```

```
      seps = 1.0
      appone = one + seps
c
c loop, dividing by 2 each time to determine when the difference between one and
c the approximation is zero in single precision
c ---------------------------------------------
c
      ipow = 0
      do 1 i=1,1000
         ipow = ipow + 1
c
c update the perturbation and compute the approximation to one
c -------------------------------------------------------------
c
        seps = seps / 2
        appone = one + seps
c
c do the comparison and if small enough, break out of the loop and return
c control to the calling code
c ---------------------------
c
        if(abs(appone-one) .eq. 0.0) return
c
    1 continue
c
c if the code gets to this point, there is a bit of trouble
c ---------------------------------------------------------
c
      print *,"The loop limit has been exceeded"
c
c done
c ----
c
      return
      end
```

7.1.2 dmaceps

Description: This routine is used to compute the machine epsilon in double precision for a computer. The code divides a remainder by two until the result added produces no change in the value of the variable one containing the the number 1.

Input:

1. float *seps* - variable passed in to hold the smallest number computed
2. integer *ipow* - the power of two that gives the number of iterations through computation. this is computed internally and the value is returned.

Output:

1. float *seps* - variable passed back that holds the smallest number computed
2. integer *ipow* - the power of two that holds the number of iterations through computation. this is computed internally and the value is returned.

Code Written:

```
c
c double precision computation of machine precision
c -------------------------------------------------
c
      subroutine dmaceps(deps, ipow)
c
c set up the calculation so that the comparison is done at 1.0
c ------------------------------------------------------------
c
      real*8 deps, one, appone
c
c set some constants for work near 1.d0
c -------------------------------------
c
      one = 1.d0
      deps = 1.d0
      appone = one + deps
c
c loop over, dividing by 2 each time to determine when the difference between
c one and the approximation is zero to finite precision
c -----------------------------------------------------
c
      ipow = 0
```

```
      do 1 i=1,1000
         ipow = ipow + 1
c
c update the perturbation and compute the approximation
c -----------------------------------------------------
c
        deps = deps / 2
        appone = one + deps
c
c do the comparison
c -----------------
c
        if(abs(appone-one) .eq. 0.d0) return
c
    1 continue
c
c done
c ----
c
      return
      end
```

A Examples of How the Code Works

If you include examples in an appendix, you should include the code that is used to access the routines, but not the routines themselves. The following gives an example.

A.1 Computing Machine Epsilon

A code was written to test the computation of a machine epsilon in both single precision and double precision. The code is:

```
c
c coding language:    Fortran 77
c
c ------------------------------------------------------------------------------
c
c written by:         Joe Koebbe
c date written:       Sept 28, 2014
c written for:        Problem Set 1
c course:             Math 5610
c
c purpose:            Determine a machine epsilon for the computers I would like
c                     work on computationally. The code contains 2 subroutines.
c
c                     smacsps - returns the single precision value for machine
c                               precision
c                     dmacsps - returns the double precision value for machine
c                               precision
c
c ------------------------------------------------------------------------------
c
      program main
c
c do the work in double precision
c -------------------------------
c
      real sval
      real*8 dval
c
c single precision test
c ---------------------
c
      call smaceps(sval, ipow)
      print *, ipow, sval
      call dmaceps(dval, ipow)
      print *, ipow, dval
c
c done
```

```
c ----
c
      stop
      end
```

The results from running the code as shown are the following:

```
      24   5.96046448E-08
      53   1.1102230246251565E-016
```

Not that this indicates that in single precision, the maximum exponent in base two that can be used is 24 and the smallest value we can observe is about 10^{-8} or about 8 digits of precision. Also, this indicates that in double precision, the maximum exponent in base two that can be used is 53 and the smallest value we can observe is about 10^{-16} or about 16 digits of precision.

References

[1] U. Ascher and C. Grief, *A First Course in Numerical Methods*, SIAM, Philadelpha, PA, 2007.