# CMPUT 340—Introduction to Numerical Methods

## Assignment 1

Winter 2007
Department of Computing Science
University of Alberta

---

**Due**: *Friday, February 2 at 23:59:59 local time*
**Worth**: 15% of final grade

**Instructor**: Dale Schuurmans, Ath409, x2-4806, dale@cs.ualberta.ca

---

**Note**: The programming part of this assignment is to be submitted electronically by using the Unix "astep" command which is available on the undergraduate machines. You need to submit the answers to all written questions in hard copy to the course drop box. Matlab functions need to be submitted in .m files, such as "SolveDiag.m" (all of the Matlab functions required are discussed below). You can submit your files by typing "`astep -c c340 -p a1` *file1 file2 ...*". If you need help with "astep" see: http://ugweb.cs.ualberta.ca/~astep

---

## **Part 1** (Approximation and error analysis—6%)

Consider the problem of evaluating the function $\arctan(x)$. This is a very useful function that, in fact, for many decades provided the basis for computing $\pi$, as you will see.

(a) (1%) Estimate the absolute *and* relative error in evaluating $\arctan(x)$ due to a small perturbation $h$ in the argument $x$. What is the condition number for this problem? For what values of the argument $x$ is this problem highly sensitive?

(b) (1%) The arctan function can also be written as the infinite series

$$\arctan(x) \;=\; x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

(This series is called "Gregory's Series", which was actually developed 40 years before Taylor' theorem was first published in the 1715.)

What are the forward and backward errors if we approximate the arctan function by using only the first term in the series, i.e., $\arctan(x) \approx x$, for $x = 0.1$, 0.5, and 1.0?

What are the forward and backward errors if we approximate the arctan function by using the first two terms in the series, i.e., $\arctan(x) \approx x - x^3/3$, for $x = 0.1$, 0.5, and 1.0?

(c) (1%) Gregory actually developed the following exact formula based on expanding the series out to $n+1$ terms, plus a correction

$$\arctan(x) \;=\; \left(\sum_{i=0}^{n}(-1)^i\frac{x^{2i+1}}{2i+1}\right) \;+\; \left((-1)^{n+1}\int_0^x \frac{z^{2n+2}}{1+z^2}dz\right)$$

The size of the correction term can be bounded by the simpler form

$$\left|\int_0^x \frac{z^{2n+2}}{1+z^2}dz\right| \;\leq\; \left|\int_0^x z^{2n+2}dz\right| \;=\; \tfrac{1}{2n+3}\,|x^{2n+3}|$$

*Key observation*:

$$\arctan(1) \;=\; \frac{\pi}{4}$$

Given this observation, one can easily estimate $\pi$ by estimating arctan(1).

How may terms are needed in the Gregory series of arctan(1) to approximate $\pi$ to 23 bits of accuracy (the floating point standard)?

How may terms are needed in the Gregory series of arctan(1) to approximate $\pi$ to 52 bits of accuracy (the double precision standard)?

(d) (1%) Explain how you would arrange the terms in the series expansion to obtain as accurate a double precision answer as you can.

(e) (2%) Write your own Matlab function `[y] = myarctan(x)` that uses your answer to the previous questions to return an accurate double precision answer. Submit your function in a file called "myarctan.m". (Make your function as efficient and accurate as you can.)

How close is your `4*myarctan(1)` result to Matlab's built in `pi`?

How close are your `myarctan` results to Matlab's built in `atan`? Over what range of values is your implementation accurate? Over what range of values is your impementation *inaccurate*?

Remember to submit all written answers in the course drop box.

---

**A note on Matlab programming style**

Matlab is a vector and matrix based programming environment. Matrix and vector operations are built in and heavily optimized. As much as possible you must avoid the use of `for` loops. Most `for` loops can be replaced by vectorizing the programs you write. You will lose marks by simply implementing the nested loops given in the textbook.

Also, functions in `help elmat` can be used below, but not those in `help matfun`.

## **Part 2** (Solving simple linear constraint systems—3%)

(a) (1%) Write a Matlab function `[x] = SolveDiag(d,b)` which computes a vector $\mathbf{x}$ that solves the linear system $D\mathbf{x} = \mathbf{b}$, where $D$ is a diagonal matrix whose diagonal vector is $\mathbf{d}$. Submit the function in a file called "SolveDiag.m".

(b) (1%) Write a Matlab function `[x] = SolveUpperTri(U,b)` which computes a vector $\mathbf{x}$ that solves the linear system $U\mathbf{x} = \mathbf{b}$, where $U$ is an upper triangular matrix. (See Page 66 in the text.) Submit the function in a file called "SolveUpperTri.m".

(c) (1%) Write a Matlab function `[x] = SolveLowerTri(L,b)` which computes a vector $\mathbf{x}$ that solves the linear system $L\mathbf{x} = \mathbf{b}$, where $L$ is a lower triangular matrix. (See Page 65 in the text.) Submit the function in a file called "SolveLowerTri.m".

## **Part 3** (Solving general linear constraint systems—4%)

(a) (2%) Write a Matlab function `[L,U] = LUfactor(A)` which computes a lower triangular matrix $L$ and an upper triangular matrix $U$ such that $A = LU$. (See Page 69 in the text.) You do not need to employ any sort of pivoting. Submit the function in a file called "LUfactor.m".

(b) (1%) Write a Matlab function `[x] = SolveGeneralLU(A,b)` which uses LU factorization, forward subtitution, and back substitution to compute a solution vector $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}$. Submit the function in a file called "SolveGeneralLU.m".

(c) (1%) Write a Matlab function `[x] = SolveGeneralGS(A,b,x0,TOL)` which uses Gauss-Siedel iteration, starting at $\mathbf{x}^{(0)}$, to compute a solution vector $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}$. Gauss-Siedel iteration is described on Page 470 of the text. It is an iterative solution technique where the $k$th iterate, $\mathbf{x}^{(k)}$, is given by the solution to

$$(D + L)\mathbf{x}^{(k)} = \mathbf{b} - U\mathbf{x}^{(k-1)}$$

where `D = diag(diag(A))`, `L=tril(A)-D`, and `U=triu(A)-D`. (See `help elmat` for a description of these functions.) Stop the iteration when the norm of the residual vector $\mathbf{r} = b - A\mathbf{x}^{(k)}$ is less than $TOL$. Submit the function in a file called "SolveGeneralGS.m".

## **Part 4** (Comparing solution techniques—2%)

You now have three different functions available for solving a system of linear equality constraints: `SolveGeneralLU`, `SolveGeneralGS`, and `slash` (Matlab's built in solver). Compare the speed and accuracy of each of these solvers. Give an example of a linear system that makes each solver look good and bad relative to the rest (if possible). Be sure to submit your answers in the course drop box.

## **Bonus!** (Alternative solution techniques—2%)

Implement the following solution methods and include them in the comparison of Part 4.

(a) (1%) Write a Matlab function `[L,U,P] = LUfactorPPivot(A)` which computes a lower triangular matrix $L$, an upper triangular matrix $U$, and a permutation matrix $P$ such that $PA = LU$ using partial pivoting described on Page 73 of the text. Use this procedure to implement a Matlab function `[x] = SolveGeneralLUPP(A,b)`.

(b) (1%) Write a Matlab function `[x] = SolvePosSemiDefCG(A,b,x0,TOL)` which uses conjugate gradient iteration, starting at $\mathbf{x}^{(0)}$, to compute a solution vector $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}$, where $A$ is positive semi-definite. Conjugate gradient iteration is described on Page 473 of the text.