CS 542G: Assigned Work, Part 1

Robert Bridson

Fall, 2009

1 Explanation

This is the assigned work for the full course. Complete it at your pace, but make sure to hand in everything by the end of lectures (December 5). You may hand in parts at any time in any order, and I will mark them as soon as I can.

All questions require some write-up, which may be electronic (emailed to me) or hardcopy (given to me or left in my mailbox).

Some questions also require programming: for these you should email me a `tar.gz` or `zip` archive of the code. Some questions will indicate a language which compiles down to machine code should be used, such as C, C++ or assembly (but not Java!)—typically because measuring performance is an important part of the question or external libraries such as BLAS are required. Note that C++ makes it fairly easy to incur a huge abstraction overhead if you use objects naively; it would be a good idea to adopt a more C-like (or maybe template-oriented) style for these cases. For most questions it doesn't matter what language you use, and higher level languages such as MATLAB, Octave, Python, Java, etc. may prove to be much more convenient (and indeed, may be useful for prototyping the other questions too).

The set of all assigned questions will be worth 40% of your final mark (and the final exam worth 60%).

2 The Questions

[Q1] For the 2D Thin Plate Spline RBF kernel $\|x\|^2 \log \|x\|$, what should be done at $\|x\| = 0$?

[Q2] Implement 2D RBF interpolation, with different kernels and a degree one polynomial component. Implement the Thin Plate Spline kernel, the Gaussian kernel $e^{-\|x\|^2/(2*\sigma^2)}$, and the multiquadric kernel $\sqrt{\|x\|^2+\sigma^2}$ (for the latter two, you will also have to choose a parameter σ).

[Q3] For the 2D RBF code, construct a test data set, sampled from some known smooth 2d function. Compare the three kernels in terms of quality. You need to first describe how you are doing this comparison, and also how you choose the parameter σ for the Gaussian and the multiquadric.

[Q4] Work out how well conditioned the RBF interpolation is. To make this specific, fix your test data set (the input) and a set of arbitrarily chosen point within the domain at which the interpolant is evaluated (the output). How much does the output change (in a relative sense) for a perturbation in the input (measured in a relative sense as well)?

[Q5] An *M-matrix* is an invertible diagonally-dominant square matrix with positive diagonal entries and non-positive off-diagonal entries. (These regularly arise in solving partial differential equations, for example.) Prove that LU factorization of an M-matrix doesn't require pivoting.

[Q6] Prove that the inverse of an M-matrix has all non-negative entries. (This is known as the Perron-Frobenius theorem: as a hint, look at an arbitrary single column of the inverse and examine the linear system it solves.)

[Q7] A *Stieltjes matrix* is a symmetric M-matrix. Prove that they must be positive definite, and thus have a Cholesky factorization.

[Q8] Write your own in-place LU factorization code, without pivoting, in a compiled language such as C. Implement the rank-1 update algorithm with the matrices stored row-by-row or column-by-column as you choose. Implement both loop orderings for the rank-1 update (i.e. updating the rows one at a time, or the columns one at a time). Time how fast the two versions run for large-ish matrices (e.g. 1000×1000) and report on the difference.

[Q9] Figure out how to use BLAS and LAPACK in your code, and compare the performance of LU factorization (subroutine `dgetrf`) against your own rank-1 update code.

[Q10] Implement a blocked version of LU without pivoting, in a compiled language like C: store matrices in fixed size $k \times k$ blocks. Implement the rank-k update version. Fix k as a compile-time constant; you may assume the matrix size will always be a multiple of k to make life easier. For the $k \times k$ block operations you may use the BLAS instead of your own code if you prefer. Find a good value of k, and compare performance to LAPACK and to the rank-1 update code from the previous two questions.

[Q11] Moving Least Squares (MLS) can break down for some arrangements of input points: (1) if the weight kernel has finite support and the points are too far apart, or (2) if the points are aligned in special ways. Explain exactly what goes wrong in these two cases. Suggest a way of dealing with these cases if they arise, but make sure to discuss disadvantages of your proposal.

[Q12] Implement Orthogonal Iteration for finding the eigenvalues of a symmetric matrix. Test it on random matrices, as well as matrices of the form $[0A; A^T 0]$ for arbitrary A. For the latter case, what might go wrong, and figure out to modify the algorithm to handle these cases.

[Q13] An interesting optimization algorithm intermediate between plain Steepest Descent and Newton is sometimes called Scaled Steepest Descent (SSD). For minimizing $f(x)$, a step in direction $d = -\hat{H}^{-1}g$ is used, where $\hat{H}$ has just the diagonal part of the Hessian H and g is the gradient $\nabla f(x)$ of f. This has the advantage that $\alpha = 1$ is a natural step length to try, as opposed to plain Steepest Descent, but can be cheaper per iteration compared to Newton. How does SSD compare to each in convergence? (Try it out on appropriate examples.)

More questions will be added for the second half of the course.