MTH 351 – Lab 6

This lab explores solving various linear systems that arise from common numerical methods. The intent of the lab is to give you an appreciation for the relationship between the structure of a matrix and the efficiency of various linear solution approaches.

1. Use `lsq.m` from the website to find the degree m least squares polynomial approximation to a function via the solution of a linear system (Hilbert matrix) and explore how the error in the linear system solution method `GEpivot` affects the error in the approximation. In particular, test an easy function such as $exp(x)$ on the interval $[0, 1]$ by doing the following:

   (a) Try $m = 1$, 2, 3, and comment on the trend of maximum error on the interval.

   (b) Increment m, for what value of m does the maximum error stop decreasing?

   (c) Increment m, for what value of m are there visible errors in the approximation?

   (d) Increment m a couple more times, comment on the trend of maximum error on the interval.

   (e) Can we conclude that low degree polynomial approximations are better than high degree approximations? (Careful.)

   (f) Could Jacobi iteration be applied to the linear system? Why or why not? Hint: consider computing the following for various values of m

   ```
   A=hilb(m+1);
   N=diag(diag(A));
   P=N-A;
   norm(inv(N)*P)
   ```

   (g) Could Gauss-Seidel iteration be applied to the linear system? Why or why not? Hint: consider computing the following for various values of m

   ```
   A=hilb(m+1);
   N=tril(A);
   P=N-A;
   norm(inv(N)*P)
   ```

   (h) **Optional:** Apply `CG.m` to perform the linear solve. Is there any improvement in runtime or error? Although we did not discuss Conjugate-Gradient in detail, you can look at the code to get an idea of how it works. Also, try `pcg.m` built-in to MATLAB.

2. Use `gaussweights.m` from the website to compute the weights for Gaussian quadrature given n nodes on the interval $[-1, 1]$. The code uses the exact values of the nodes (from a table) to determine the linear system (Vandermonde matrix) to be solved. It applies `GEpivot` to solve the system, and compares the computed weights to the exact values graphically.

   (a) Try $n = 2$, 4, and 8, and compare the results with Table 5.7 in the book.

   (b) Try $n = 32$ and 64. Include only these plots in your work to turn in. Explain what goes wrong. It may be helpful to compute the condition number by adding `cond(A,inf)` to the code.

   (c) Could Jacobi iteration be applied to the linear system? Why or why not?

   (d) Could Gauss-Seidel iteration be applied to the linear system? Why or why not?

   (e) **Optional:** As the matrix is not symmetric, `CG.m` and `pcg.m` do not apply. But you may want to try other alternatives for the $m = 64$ case, such as

   ```
   x=inv(A)*b;
   x=A\b;
   x=gmres(A,b,[],eps,1000);
   ```

   Look closely at the scale of the vertical axis of the plot from each method to get an idea of which one performs the best.

3. Use `cubspline.m` from the website to solve the tridiagonal system corresponding to natural cubic spline interpolation. The code uses the example of approximating the Legendre polynomial of degree m using n (uniformly spaced) nodes. The linear system is solved using `GEpivot`. The polynomial and its approximation are plotted, and the residual and computation times are output. In this example, use $m = 15$. Try $n = 8$ and 16 to get an idea of what the code is doing.

   (a) Try $n = 64$, 128, and 256. Record the time required to solve the linear system. Notes: Computation time varies due to many factors. Consider running each example a couple times and averaging the runtimes.

   (b) Modify the code to call the file `Jacobi.m` in order to use Jacobi iteration for the linear solve. Record the time required, and the number of iterations, for $n = 64$, 128, and 256. Compare the runtimes to those of `GEpivot`. Notes: Use error tolerance `delta=eps` to get accuracy upto round-off. Use a sufficiently large maximum number of iterations.

   (c) Modify the code to call the file `GS.m` in order to use Gauss-Seidel iteration for the linear solve. Record the number of iterations required for $n = 64$, 128, and 256 and compare to those of `Jacobi`. Notes: Again, use `delta=eps`. In general one would write Gauss-Seidel in such a way that the time for each iteration is the same as a Jacobi iteration, but this has not been done here, so ignore the timings for Gauss-Seidel.

(d) Modify the code to call the file `CG.m` in order to use Conjugate-Gradient iteration for the linear solve. Record the runtimes and the number of iterations required for $n = 64$, 128, and 256, and compare to those of `Jacobi`. Notes: Use `delta=eps`. Although we did not discuss Conjugate-Gradient in detail, you can look at the code to get an idea of how it works.

(e) Modify the code to call the file `tridiag.m` in order to use the tridiagonal version of LU factorization and backsubstitution. Record the runtimes for $n = 64$, 128, and 256 and compare to those of all of the previous methods.

(f) **Optional:** Just for fun, try `tridiag.m` with $n = 2048$ and compare to the other methods using $n = 64$. Do not try `GEpivot` for $n = 2048$ unless you want to wait for a long time!

You do not need to include these plots in your work to turn in.

4. **Optional:** Use `poisson.m` from the website to solve the Poisson problem in 2D given by

$$\nabla^2 u = \sin(\pi x)\sinh(\pi y)$$

with $u = 0$ on the boundary of $[0, 1] \times [0, 1]$ (see Section 9.1). The code uses m grid points in x and y directions, and finite difference derivatives (see Section 5.4) for discretization. The resulting $m^2 \times m^2$ banded linear system (see Section 6.6, Problem 9) is solved with `GEpivot`. The solution is plotted, and the residual and computation times are output.

(a) Try $m = 4$, 8, and 16. Record the time required to solve the linear system. Notes: Computation time varies due to many factors. Consider running each example a couple times and averaging the runtimes.

(b) Follow instructions from (3b) here but use $m = 4$, 8, and 16.

(c) Follow instructions from (3c) here but use $m = 4$, 8, and 16.

(d) Follow instructions from (3d) here but use $m = 4$, 8, and 16. How are the number of iterations increasing with respect to increasing m?

(e) Note that `tridiag.m` does not apply. But you may want to try other alternatives, especially the FFT implementation at `http://www.siam.org/books/fa01/fish2d.m` by changing the `GEpivot` line to

```
uhsv = fish2d(b);
```

You do not need to include these plots in your work to turn in.