UC Berkeley Math 228B, Spring 2024: Problem Set 8

Prof. Per-Olof Persson (persson@berkeley.edu)

Optional, not graded

1. In this problem, you will write a multigrid solver for the linear system of equations generated by `fempoi` and `pmesh` from previous problem sets. Note that in 2-D it is very hard to be faster than the built-in backslash function, at least without using a compiled language. However, for large problems in 3-D, any multigrid solver should be superior to Gaussian elimination, so here we are more concerned about getting the right convergence behavior rather than a fast solver. To begin with, add two output arguments to the `fempoi` function to get access to the matrices A, b in the linear system:

```
function [u, A, b] = fempoi(p, t, e)
```

(a) Write a MATLAB function

```
function [u, res] = gauss_seidel(A, b, u, niter)
```

that makes `niter` iterations using the Gauss-Seidel method for $Au = b$:

$$u_{m+1} = u_m + (D - L)^{-1}(b - Au_m),$$

starting from the input `u` and returning the last iterate `u`. $D - L$ is the lower triangular part of A, including the diagonal (see the `tril` command). The output `res` is a vector of length `niter`+1 with the infinity norms of the residuals $b - Au_m$ at each iteration (including the initial and the final iterates). Try the function using the commands:

```
pv = [0,0; 2,0; 1.5,1; .5,1; 0,0];
[p, t, e] = pmesh(pv, 0.5, 3);
[u0, A, b] = fempoi(p, t, e);
[u, res] = gauss_seidel(A, b, 0*b, 1000);
semilogy(0:1000, res)
```

(b) Write a MATLAB function

```
function data = mginit(pv, hmax, nref)
```

that computes all the required arrays for a multigrid solution of the Poisson problem using the mesh parameters `pv,hmax,nref`. Start from the `pmesh` function, and make appropriate modifications and additions.

(a) `data(i).p`, `data(i).t`, `data(i).e` contain the mesh arrays p, t, e after $i-1$ refinements, for $i - 1 = 0, \ldots, n_{\text{ref}}$.

(b) `data(i).T` contains the interpolation matrix $T^{(i)}$ from grid i to grid $i+1$, for $i = 1 \ldots, n_{\text{ref}}$. Use linear interpolation for all the new midpoints (that is, averaging of the neighboring nodes). The second output argument of `unique` might be useful.

(c) `data(i).R` contains the restriction matrix $R^{(i)}$ from grid $i+1$ to grid i. Use the transpose of $T^{(i)}$, but with the rows scaled to have sums of 1.

(d) `data(nref+1).A`, `data(nref+1).b` contain A, b for the finest grid (the actual linear system)

(e) `data(i).A` contains the projected matrices $A^{(i)} = R^{(i)} A^{(i+1)} T^{(i)}$ for $i = 1, \ldots, n_{\text{ref}}$.

(c) Write a MATLAB function

```
function [u, res] = mgsolve(data, vdown, vup, tol)
```

that solves the problem precomputed in `data`, using multigrid V-cycles with `vdown/vup` pre/post-smoothing iterations using Gauss-Seidel, until the infinite norm of the residual is less than `tol`. The outputs are the solution `u` and the residuals `res` after each V-cycle (including the residual for the initial solution $u = 0$).

Test the function using the commands

```
pv = [0,0; 2,0; 1.5,1; .5,1; 0,0];
for iref = 1:5
    data = mginit(pv, 0.5, iref);
    [u, res] = mgsolve(data, 2, 2, 1e-10);
    semilogy(res), hold on
end
hold off
```

If everything works correctly, you should see a very fast convergence compared to pure Gauss-Seidel. More importantly, the number of iterations should not increase much when the grid is refined. This leads to the optimal $\mathcal{O}(n)$ computational cost of the algorithm.