Math 660: Problem Set 6

Matthew Grasinger

May 3, 2017

1 C1: Linear Iterative Methods

1.1 Source Code

```julia
include("hw6_helpers.jl");

function jacobi(h::Real; tol::Real=1e-7, max_iterations::Int=10000)
  grid = Grid(h);
  enforce_bcs!(grid);
  next_grid = Grid(grid);
  const start_time = time();
  for iteration = 1:max_iterations
    for j=2:grid.ny-1, i=2:grid.nx-1
      next_grid.us[i, j] = (grid.us[i-1, j] + grid.us[i+1, j] +
                            grid.us[i, j-1] + grid.us[i, j+1] -
                            h*h*force(grid.xs[i, j], grid.ys[i, j])) / 4.0;
    end
    if norm(next_grid.us - grid.us, 2) < tol
      return next_grid, iteration, time() - start_time, true;
    end
    copy_grid_values!(grid, next_grid);
  end
  return grid, max_iterations, time() - start_time, false;
end

function gauss_seidel(h::Real; tol::Real=1e-7, max_iterations::Int=10000)
  grid = Grid(h);
  enforce_bcs!(grid);
  next_grid = Grid(grid);
  const start_time = time();
  for iteration = 1:max_iterations
    for j=2:grid.ny-1, i=2:grid.nx-1
      # use updated neighboring values
      next_grid.us[i, j] = (next_grid.us[i-1, j] + next_grid.us[i+1, j] +
                            next_grid.us[i, j-1] + next_grid.us[i, j+1] -
                            h*h*force(grid.xs[i, j], grid.ys[i, j])) / 4.0;
    end
    if norm(next_grid.us - grid.us, 2) < tol
      return next_grid, iteration, time() - start_time, true;
    end
    copy_grid_values!(grid, next_grid);
  end
  return grid, max_iterations, time() - start_time, false;
end

function SOR(h::Real; tol::Real=1e-7, max_iterations::Int=10000)
  const omega = 2.0 / (1.0 + pi * h);
  grid = Grid(h);
  enforce_bcs!(grid);
  next_grid = Grid(grid);
  const start_time = time();
  for iteration = 1:max_iterations
    for j=2:grid.ny-1, i=2:grid.nx-1
      # use updated neighboring values
      next_grid.us[i, j] += omega * (
                           (next_grid.us[i-1, j] + next_grid.us[i+1, j] +
                            next_grid.us[i, j-1] + next_grid.us[i, j+1] -
                            h*h*force(grid.xs[i, j], grid.ys[i, j])) / 4.0 -
                           next_grid.us[i, j]);
    end
    if norm(next_grid.us - grid.us, 2) < tol
      return next_grid, iteration, time() - start_time, true;
    end
    copy_grid_values!(grid, next_grid);
  end
  return grid, max_iterations, time() - start_time, false;
end
```

```julia
for (method_str, approx_method) in [("Jacobi iterative solution", jacobi);
                                    ("Gauss-Seidel iterative solution", gauss_seidel);
                                    ("SOR iterative solution", SOR)]
  for h in [0.1; 0.05; 0.025]
    println(method_str, ", h = ", h);
    grid_approx_soln, iterations, time_elapsed, did_converge = approx_method(h);
    println("   time elapsed:     ", time_elapsed);
    println("   iterations:       ", iterations);
    println("   sec/iteration:    ", time_elapsed / iterations);
    println("   converged:        ", (did_converge) ? "true" : "false");
    grid_analytical_soln = Grid(h, analytical_soln);
    println("   rel. L2 error:    ",
            norm(grid_analytical_soln.us - grid_approx_soln.us, 2) /
            norm(grid_analytical_soln.us, 2));
    println();
  end
end
```

1.2 Results

```
Jacobi iterative solution, h = 0.1
   time elapsed:     0.004197835922241211
   iterations:       286
   sec/iteration:    1.4677747979864373e-5
   converged:        true
   rel. L2 error:    5.604084855203742e-5

Jacobi iterative solution, h = 0.05
   time elapsed:     0.08460402488708496
   iterations:       1098
   sec/iteration:    7.70528459809517e-5
   converged:        true
   rel. L2 error:    1.4088944068211665e-5

Jacobi iterative solution, h = 0.025
   time elapsed:     0.682297945022583
   iterations:       4180
   sec/iteration:    0.0001632291734503787
   converged:        true
   rel. L2 error:    2.14377570070329e-6

Gauss-Seidel iterative solution, h = 0.1
   time elapsed:     0.0020170211791992188
   iterations:       152
   sec/iteration:    1.3269876178942228e-5
   converged:        true
   rel. L2 error:    5.624218515105956e-5

Gauss-Seidel iterative solution, h = 0.05
   time elapsed:     0.0244901180267334
   iterations:       580
   sec/iteration:    4.222434142540241e-5
   converged:        true
   rel. L2 error:    1.451514372688249e-5

Gauss-Seidel iterative solution, h = 0.025
   time elapsed:     0.3040142059326172
   iterations:       2207
   sec/iteration:    0.0001377499800329031
   converged:        true
   rel. L2 error:    2.985097232356857e-6

SOR iterative solution, h = 0.1
   time elapsed:     0.0005340576171875
   iterations:       37
   sec/iteration:    1.4433989653716216e-5
   converged:        true
   rel. L2 error:    5.639472054373996e-5
```

```
SOR iterative solution, h = 0.05
   time elapsed:     0.0026369094848632812
   iterations:       71
   sec/iteration:    3.713957020934199e-5
   converged:        true
   rel. L2 error:    1.49024396571672e-5

SOR iterative solution, h = 0.025
   time elapsed:     0.017843961715698242
   iterations:       139
   sec/iteration:    0.0001283738252927931
   converged:        true
   rel. L2 error:    3.8099858389904924e-6
```

1.3 Discussion

The time per iteration was consistently on the order of 10^{-4}–10^{-5} seconds per iteration for each method, with longer times corresponding to finer grids. The fact that the time per iteration is consistent across methods agrees with theory. The accuracy, with respect to the analytical solution, is also consistent across methods and appears to depend primarily on the fineness of the grid. Therefore, the most apparent difference between methods is the number of iterations required to reach convergence. The Jacobi method required the largest number of iterations to converge. Gauss-Seidel converged in about half as many iterations on average. SOR converged consistently converged in the least number of steps, requiring an order of magnitude less steps than either the Jacobi or Gauss-Seidel methods.

2 C2: Conjugate Gradient

2.1 Source Code

```julia
include("hw6_helpers.jl");

function conjugate_gradient(h::Real; tol::Real=1e-7)

  # initialize data
  grid = Grid(h);
  enforce_bcs!(grid);
  us = grid.us;
  rs = zeros(grid.nx, grid.ny);
  qs = zeros(grid.nx, grid.ny);
  const start_time = time();

  for j = 2:grid.ny-1, i = 2:grid.nx-1
    x, y = grid.xs[i, j], grid.ys[i, j];
    rs[i, j] = (-h*h * force(x, y) + us[i+1, j] + us[i-1, j] + us[i, j+1] +
                us[i, j-1] - 4.0 * us[i, j]);
  end
  r2_prev = dot(rs, rs);
  ps = copy(rs);
  for j = 2:grid.ny-1, i = 2:grid.nx-1
    qs[i, j] = 4*rs[i, j] - rs[i+1, j] - rs[i-1, j] - rs[i, j+1] - rs[i, j-1];
  end

  pdotq = dot(ps, qs);
  for iteration = 1:(grid.nx*grid.ny)
    alpha = r2_prev / pdotq;
    for j = 2:grid.ny-1, i = 2:grid.nx-1
      us[i, j] += alpha * ps[i, j];
    end

    if norm(alpha * ps, 2) < tol
      return us, iteration, time() - start_time, true;
    end

    for j = 2:grid.ny-1, i = 2:grid.nx-1
      rs[i, j] -= alpha * qs[i, j];
    end
    r2_new = dot(rs, rs);
    beta = r2_new / r2_prev;
    for j = 2:grid.ny-1, i = 2:grid.nx-1
      ps[i, j] = rs[i, j] + beta * ps[i, j];
      qs[i, j] = (4*rs[i, j] - rs[i+1, j] - rs[i-1, j] - rs[i, j+1] -
                  rs[i, j-1] + beta * qs[i, j]);
    end
    r2_prev = r2_new;
    pdotq = dot(ps, qs);
  end

  return us, grid.nx*grid.ny, time() - start_time, false;
end

for h in [0.1; 0.05; 0.025]
  println("Conjugate gradient, h = ", h);
  approx_soln, iterations, time_elapsed, did_converge = conjugate_gradient(h);
  println("   time elapsed:      ", time_elapsed);
  println("   iterations:        ", iterations);
  println("   sec/iteration:     ", time_elapsed / iterations);
  println("   converged:         ", (did_converge) ? "true" : "false");

  const n = size(approx_soln, 1);
  num = 0.0;
  den = 0.0;
  for (j, y) in zip(1:n, 0.0:h:1.0), (i, x) in zip(1:n, 0.0:h:1.0)
    num += (approx_soln[i, j] - analytical_soln(x, y))^2;
```

```
    den += (analytical_soln(x, y))^2;
  end

  println("   rel. L2 error:    ", sqrt(num / den));
  println();
end
```

2.2 Results

```
Conjugate gradient, h = 0.1
   time elapsed:     0.0004241466522216797
   iterations:       28
   sec/iteration:    1.5148094722202846e-5
   converged:        true
   rel. L2 error:    5.6416793358370105e-5

Conjugate gradient, h = 0.05
   time elapsed:     0.0019061565399169922
   iterations:       58
   sec/iteration:    3.286476792960331e-5
   converged:        true
   rel. L2 error:    1.4923993244701214e-5

Conjugate gradient, h = 0.025
   time elapsed:     0.013283014297485352
   iterations:       118
   sec/iteration:    0.00011256791777529959
   converged:        true
   rel. L2 error:    3.835920588689894e-6
```

2.3 Discussion

The time per iteration was on the order of 10^{-4}–10^{-5} seconds per iteration for the conjugate gradient method, which is consistent with the methods used in Section 1. The accuracy, with respect to the analytical solution, was also consistent with the methods used in Section 1. However, the conjugate gradient required less iterations to converge for all cases.