```
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

```

```
65  for (method_str, approx_method) in [("Jacobi iterative solution", jacobi);
66                                      ("Gauss-Seidel iterative solution",
    gauss_seidel);
67                                      ("SOR iterative solution", SOR)]
68    for h in [0.1; 0.05; 0.025]
69      println(method_str, ", h = ", h);
70      grid_approx_soln, iterations, time_elapsed, did_converge = approx_method(h);
71      println("   time elapsed:     ", time_elapsed);
72      println("   iterations:       ", iterations);
73      println("   sec/iteration:    ", time_elapsed / iterations);
74      println("   converged:        ", (did_converge) ? "true" : "false");
75      grid_analytical_soln = Grid(h, analytical_soln);
76      println("   rel. L2 error:    ",
77              norm(grid_analytical_soln.us - grid_approx_soln.us, 2) /
78              norm(grid_analytical_soln.us, 2));
79      println();
80    end
81  end
```