```julia
analytical_soln(x::Real, y::Real) = cos(x) * sin(y);

force(x::Real, y::Real) = -2.0 * cos(x) * sin(y);

type Grid
  nx::Int
  ny::Int
  us::Matrix{Float64}
  xs::Matrix{Float64}
  ys::Matrix{Float64}

  function Grid(h::Real)
    n = Int(round(1/h) + 1);
    xs = Matrix{Float64}(n, n);
    ys = Matrix{Float64}(n, n);
    for (j, y) in zip(1:n, 0.0:h:1.0), (i, x) in zip(1:n, 0.0:h:1.0)
      xs[i, j] = x;
      ys[i, j] = y;
    end
    return new(n, n, zeros(n, n), xs, ys);
  end

  function Grid(h::Real, analytical_soln::Function)
    n = Int(round(1/h) + 1);
    xs = Matrix{Float64}(n, n);
    ys = Matrix{Float64}(n, n);
    us = Matrix{Float64}(n, n);
    for (j, y) in zip(1:n, 0.0:h:1.0), (i, x) in zip(1:n, 0.0:h:1.0)
      xs[i, j] = x;
      ys[i, j] = y;
      us[i, j] = analytical_soln(x, y);
    end
    return new(n, n, us, xs, ys);
  end

  Grid(grid::Grid) = new(grid.nx, grid.ny, copy(grid.us), copy(grid.xs),
                         copy(grid.ys));
end

function copy_grid_values!(dest::Grid, src::Grid)
  copy!(dest.us, src.us);
end

function enforce_bcs!(grid::Grid)
  for j=1:grid.ny
    grid.us[1, j] = analytical_soln(grid.xs[1, j], grid.ys[1, j]);
    grid.us[grid.nx, j] = analytical_soln(grid.xs[grid.nx, j],
                                          grid.ys[grid.nx, j]);
  end

  for i=2:grid.nx-1
    grid.us[i, 1] = analytical_soln(grid.xs[i, 1], grid.ys[i, 1]);
    grid.us[i, grid.ny] = analytical_soln(grid.xs[i, grid.ny],
                                          grid.ys[i, grid.ny]);
  end
end
```