```
%%%%%%%%%%%%%% Generalized Reduced Gradient Implementation %%%%%%%%%%%%%%
%%%%%%%%%%%%%% By Max Yi Ren and Emrah Bayrak %%%%%%%%%%%%%%%%%%%%%%%%%%%

function solution = gradient(f, g, h, delh, d0, d_id, s_id, opt)
    % Set initial conditions

    d = d0; % Set current solution to the initial guess
    s0 = [0.5;0.6]; % Set an arbitrary guess for s corresponding to your d0 (column
vector)

    x = zeros(numel(s_id)+numel(d_id),1); % Create a zero column vector for x

    x([d_id, s_id]) = [d;s0]; % Set initial guess to current solution.

    s = solveh(x, h, delh, s_id); % Find the corresponding s to d0 starting from s0

    x([d_id, s_id]) = [d;s]; % Save corrected s and d to current solution.

    % Initialize a structure to record search process
    solution = struct('x',[]);
    solution.x = [solution.x, x]; % save current solution to solution.x

    % Set the termination criterion:
    % Remember that in GRG reduced gradient is used to find stationary points.

    dfdv = g(x);

    dhds = delh(x);

    m = dhds(:,2:3); % current dh/ds

        % Modify dh/ds when it is singular
        %%% KEEP THIS %%%
    dhds_inv = correctH(m);

    delzdeld = dfdv(1) - dfdv(2:3)*dhds_inv*dhds(:,1); % reduced gradient


    gnorm = norm(delzdeld,2); % norm of reduced gradient

    while gnorm>opt.eps % if not terminated

        % opt.linesearch switches line search on or off.
        % You can first set the variable "a" to different constant values and see how
it
        % affects the convergence.
        if opt.linesearch
            a = lineSearch(f, g, delh, x, d_id, s_id);
        else
            a = 0.01;
        end

        % Gradient descent:
        dstep = a*delzdeld;      % step for decision variables (make sure it is column
vector)
```

```
        d = d - dstep;          % update d with dstep
        sstep = dhds_inv*dhds(:,1)*a*delzdeld;      % find approximate step for state↙
variables (column vector)
        s_approx = s - sstep;   % calculate approximate values for s using the the↙
approximate step

        x([d_id,s_id]) = [d; s_approx]; % save the decision and approximate state↙
variables to current solution

        % State variable correction
        s = solveh(x, h, delh, s_id); % Calculate the actual state variables using↙
linear approximation of h

        x([d_id,s_id]) = [d; s];      % Save the corrected variables to current↙
solution

        % Update termination criterion:
        dfdv = g(x);

        dhds = delh(x);

        m = dhds(:,2:3); % current dh/ds

            % Modify dh/ds when it is singular
            %%% KEEP THIS %%%
        dhds_inv = correctH(m);

        delzdeld = dfdv(1) - dfdv(2:3)*dhds_inv*dhds(:,1); % reduced gradient


        gnorm = norm(delzdeld,2); % norm of reduced gradient


        % save current solution to solution.x
        solution.x = [solution.x, x];
    end
    %disp(solution.x);
end
```