Jesse Wynn
ME EN 575 Optimization
Homework 3
Unconstrained Optimization
February 5, 2018

Report

I. Description of Optimization Program

This program is a simple MATLAB script that conducts unconstrained optimization using three common methods: steepest descent, conjugate gradient, and the quasi-Newton method with a Broyden Fletcher Goldfarb Shanno (BFGS) update. This report contains results from each of these three methods for a quadratic function of three variables, and for Rosenbrock's function (two variables). For each of these methods, an exact line search routine (fitting a quadratic) was used to determine alpha* in order to take a 'minimizing step'.

Through several tests with the given functions, all three methods were successful at arriving at the optimum value for both the quadratic function of three variables and for Rosenbrock's function. In attempting to solve Rosenbrock's function using steepest descent, a solution could be found after around 1000 iterations but computation is normally halted after 500. It was noted that both the Conjugate Gradient and Steepest Descent methods were sensitive to step size (alpha) when attempting to solve Rosenbrock's function. With a value of alpha = 0.5, Conjugate Gradient and Steepest Descent both solved the quadratic function with relative ease. When tested on Rosenbrock's function, both methods failed. By adjusting alpha to alpha = 0.005, both the Steepest Descent and Conjugate Gradient methods were able to solve Rosenbrock's function at the cost of slightly more objective function evaluations. This particular value for alpha could likely be 'tuned in' to get slightly better performance, but this was not explored. The quasi-Newton BFGS method seemed to be much more robust and worked well with alpha = 0.5 for both the quadratic function and for Rosenbrock's.

As was to be expected, for both functions the quasi-Newton BFGS performed the best, the Conjugate Gradient was in the middle, and the Steepest Descent method performed the most poorly. For these tests performance was quantified by the number of minimizing steps taken (iterations), number of objective function evaluations, and number of function gradient evaluations. For more details on these results, please see section II.

To run this program, simply open fminunDriv.m, set the initial starting point, define the objective and gradient for the function you would like to minimize, choose which method you would like to use to solve (1 = Steepest Descent, 2 = Conjugate Gradient, 3 = BFGS quasi-Newton), set the stop tolerance, and finally hit 'Run'. The function fminunDriv() will call the function fminum() which has been written to solve the optimization problem. Once finished, the optimum value, and location of the optimum will be displayed in the output window along with the total number of iterations, number of objective evaluations, and number of gradient evaluations. If the function has not found an optimum after 500 iterations, it will return and indicate that it failed to reach an optimum.

II. Test Results

Problem 1: Quadratic function of three variables

Table 1: Iteration Data for Conjugate Gradient Method

Iteration	Starting Point	Obj. Value	Search Dir.	Step Length	# Obj
0	[10, 10, 10]'	520	[-93, 6, -8]'	0	1
1	[2.22, 10.50, 9.33]'	154.34	[-2.99, -10.31, -15.42]'	7.82	11
2	[-0.75, 0.26, -5.98]'	-14.40	[0.29, 3.27, -3.85]'	18.66	25
3	[-0.56, 2.44, -8.56]'	-22.39	[0.00, 0.00, 0.00]'	3.39	39

Table 2: Iteration Data for Quasi-Newton Method

Iteration	Starting Point	Obj. Value	Search Dir.	Step Length	# Obj
0	[10, 10, 10]'	520	[-0.99, 0.06, -0.09]'	0	1
1	[2.23, 10.50, 9.33]'	154.34	[-2.99, -10.32, -15.42]'	7.82	12
2	[-0.75, 0.26, -5.98]'	-14.40	[0.29, 3.27, -3.85]'	18.66	20
3	[-0.56, 2.44, -8.56]'	-22.39	[0.00, 0.00, 0.00]'	3.39	25

Table 3: Iteration Data for Steepest Descent Method

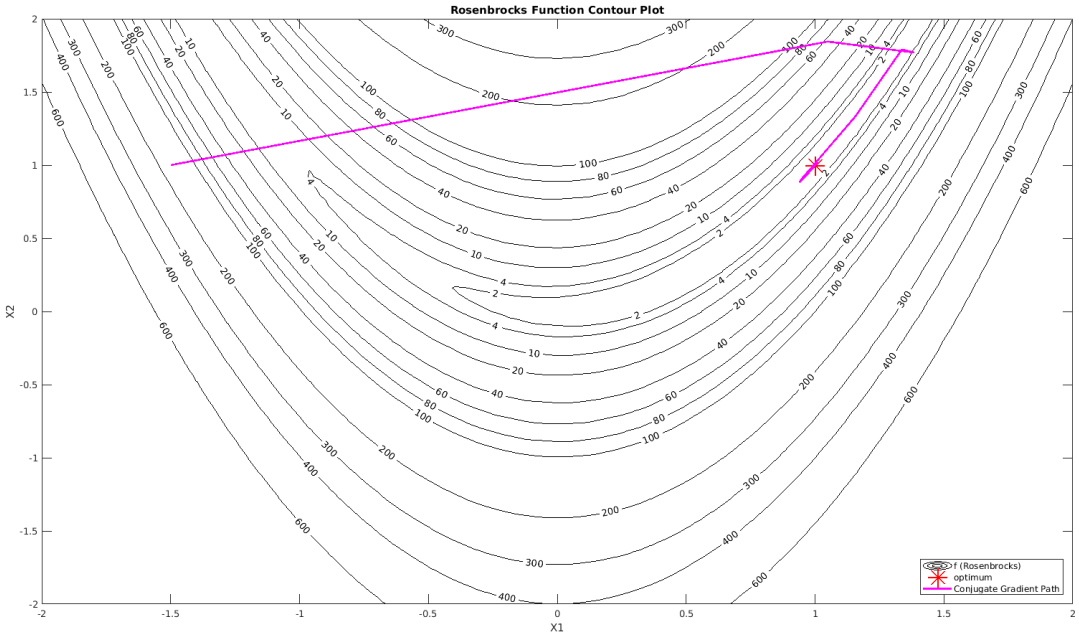
Iteration	Starting Point	Obj. Value	Search Dir.	Step Length	# Obj
0	[10, 10, 10]'	520	[-0.99, 0.06, -0.09]'	0	1
1	[2.23, 10.50, 9.33]'	154.34	[0.03, -0.57, -0.82]'	7.82	18
2	[2.65, 3.33, -0.93]'	38.88	[-0.98, 0.16, -0.15]'	12.53	36
3	[0.19, 3.72, -1.31]'	1.55	[0.12, -0.16, -0.98]'	2.51	51
4	[0.73, 3.02, -5.59]'	-12.99	[-0.98, 0.12, -0.14]'	4.38	68
5	[-0.23, 3.14, -5.74]'	-18.67	[0.10, -0.28, -0.95]'	0.98	83
33	[-0.56, 2.44, -8.56]'	-22.39	[0.11, -0.27, -0.96]	0.00	298

Total Objective and Gradient Evaluations for Each Method:

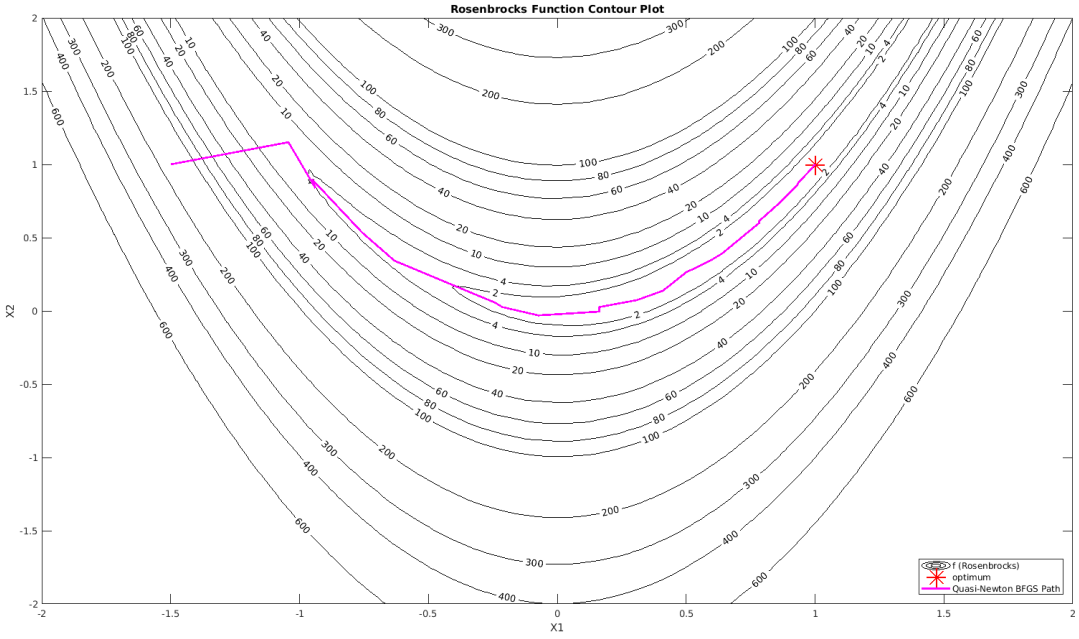
- Steepest Descent: 298 Objective Evaluations and 34 Gradient Evaluations
- Conjugate Gradient: 39 Objective Evaluations and 4 Gradient Evaluations
- BFGS quasi-Newton: 25 Objective Evaluations and 4 Gradient Evaluations

Problem 2: Rosenbrock's Function

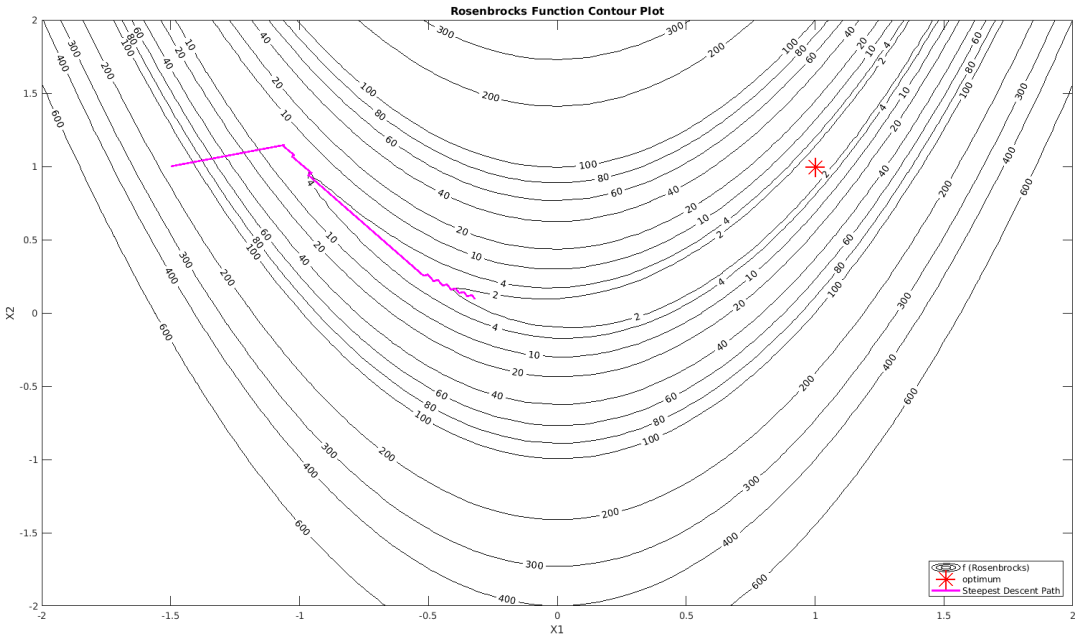
Plot 1: Conjugate Gradient Path to the optimum of Rosenbrock's function



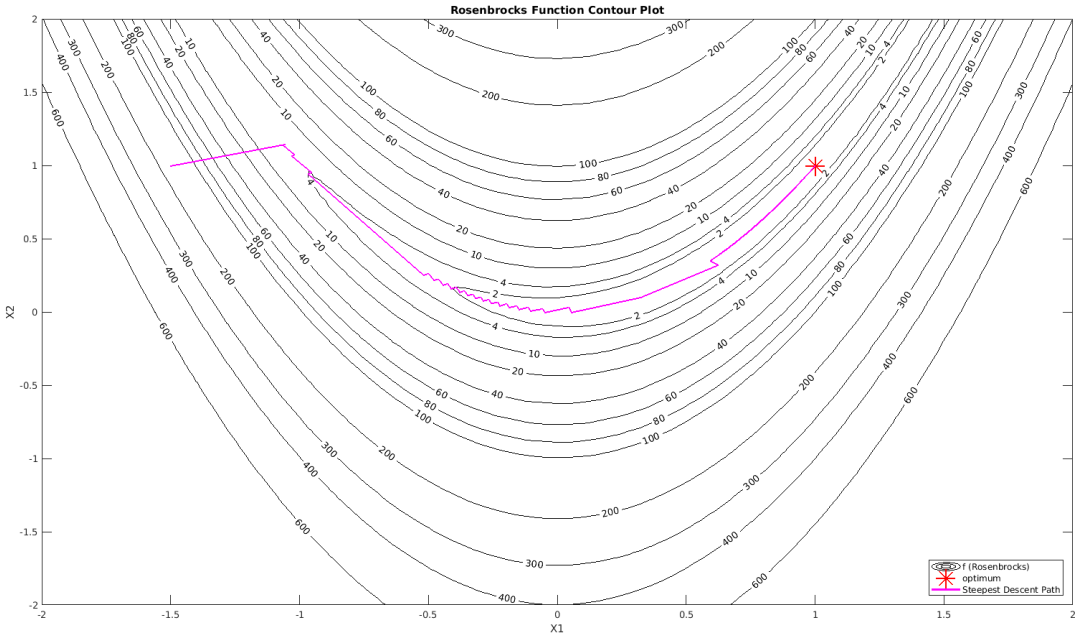
Plot 2: BFGS quasi-Newton Path to the optimum of Rosenbrock's function



Plot 3: Steepest Descent Path (first 20 steps) of Rosenbrock's function



Plot 4: Steepest Descent Path to the optimum of Rosenbrock's function



Total Objective and Gradient Evaluations for Each Method:

- Steepest Descent: 6997 Objective Evaluations and 1096 Gradient Evaluations
- Conjugate Gradient: 206 Objective Evaluations and 31 Gradient Evaluations
- BFGS quasi-Newton: 175 Objective Evaluations and 25 Gradient Evaluations

III. Appendix

MATLAB Script

```
function [xopt, fopt, exitflag] = fminun(obj, gradobj, x0, stoptol, algoflag)

% get function and gradient at starting point
[n,~] = size(x0); % get number of variables
f = obj(x0);
grad = gradobj(x0);
x = x0;

%set starting step length
% alpha0 = 0.005;

% set the initial search direction and starting step length
if (algoflag == 1)     % steepest descent
   s = srchsd(grad, 1);
   alpha0 = 0.005;

elseif (algoflag == 2)    % conjugate gradient
   s = srchsd(grad, 0);  % don't normalize search dir.
   alpha0 = 0.005;

elseif (algoflag == 3)    % BFGS Quasi-Newton
   N = eye(n);
   s = srchsd(grad, 1);
   s = N*s;
   alpha0 = 0.5;

else
   disp('Error: Valid values for algoflag are 1, 2, or 3.')
   return;
end

% initialize variables
alpha = alpha0;
num_iterations = 0;
max_iterations = 1e4;
plot_func = 0;
max_data_rows = 100;

if plot_func == 1
   % initialize array to store x_values
   x_vals = zeros(max_data_rows,2);
   x_vals(1,:) = x0';
end


%% Steepest Descent
if algoflag == 1  % steepest descent
   while num_iterations < max_iterations

      % start going along the function in the direction of s
      xnew = x + alpha*s;
```

```
% evaluate the function at xnew
fnew = obj(xnew);

% if we're making progress going in the s direction...
if fnew < f

    % double alpha and then keep going
    alpha = alpha*2;

    % set f equal to fnew and go again
    f = fnew;

    % Here we know we've gone too far.  Now we can conduct a
    % line search, find the approprate alpha, and start working
    % in a new search direction.
elseif fnew > f

    a1 = alpha/4;
    a2 = alpha/2;
    a3 = alpha;
    a4 = (a3 + a2)/2;

    % Now we need to pick the 3 points we will use
    % to bracket the minimum and perform the quadratic
    % line search.

    % evaluate the function at each
    f1 = obj(x + a1*s);
    f2 = obj(x + a2*s);
    f3 = obj(x + a3*s);
    f4 = obj(x + a4*s);

    % find the minimum point
    [~, idx] = min([f1, f2, f3, f4]);

    if idx == 2
        a1 = a1;
        a2 = a2;
        a3 = a4;

        f1 = f1;
        f2 = f2;
        f3 = f4;

    elseif idx == 4
        a1 = a2;
        a2 = a4;
        a3 = a3;

        f1 = f2;
        f2 = f4;
        f3 = f3;

    elseif idx == 1

        % in this case we need to go one more step back in the
        % past. we also need to compute these in reverse order.
```

```
            a3 = a2;
            a2 = a1;
            a1 = a1/2;


            f3 = f2;
            f2 = f1;
            f1 = obj(x + a1*s);

        else
            % this case shouldn't happen
            % disp('Unexpected case: idx containing minimum value = 3')
        end

        % find the 'optimal' alpha using a quadratic line search
        a_star = qline_search(a1, f1, a2, f2, a3, f3);

        % set a new 'x' value using a_star and the current search dir
        x = x + a_star*s;

        % sl = norm(a_star*s);

        % evaluate the gradient at this new 'x' location
        grad = gradobj(x);

        % find our new search direction to start going in
        s = srchsd(grad, 1);

        % reset alpha down to a small number
        alpha = alpha0;

        % increment the iteration count
        num_iterations = num_iterations + 1;

        % store data if we're going to plot
        if plot_func == 1 && num_iterations < max_data_rows
            x_vals(num_iterations + 1,:) = x';
        end

        % check to see if the gradient is close enough to zero
        % i.e. we're at the optimum.
        if norm(grad) < stoptol
            break
        end

    else
        % f isn't changing so we've reached an optimum?
        break
    end


  end
end

%% Conjugate Gradient
if algoflag == 2  % conjugate gradient
  while num_iterations < max_iterations
```

```
% start going along the function in the direction of s
xnew = x + alpha*s;

% evaluate the function at xnew
fnew = obj(xnew);

% if we're making progress going in the s direction...
if fnew < f

    % double alpha and then keep going
    alpha = alpha*2;

    % set f equal to fnew and go again
    f = fnew;

    % Here we know we've gone too far.  Now we can conduct a
    % line search, find the approprate alpha, and start working
    % in a new search direction.
elseif fnew > f

    a1 = alpha/4;
    a2 = alpha/2;
    a3 = alpha;
    a4 = (a3 + a2)/2;

    % Now we need to pick the 3 points we will use
    % to bracket the minimum and perform the quadratic
    % line search.

    % evaluate the function at each
    f1 = obj(x + a1*s);
    f2 = obj(x + a2*s);
    f3 = obj(x + a3*s);
    f4 = obj(x + a4*s);

    % find the minimum point
    [~, idx] = min([f1, f2, f3, f4]);

    if idx == 2
        a1 = a1;
        a2 = a2;
        a3 = a4;

        f1 = f1;
        f2 = f2;
        f3 = f4;

    elseif idx == 4
        a1 = a2;
        a2 = a4;
        a3 = a3;

        f1 = f2;
        f2 = f4;
        f3 = f3;

    elseif idx == 1
```

```
            % in this case we need to go one more step back in the
            % past. we also need to compute these in reverse order.

            a3 = a2;
            a2 = a1;
            a1 = a1/2;


            f3 = f2;
            f2 = f1;
            f1 = obj(x + a1*s);

        else
            % this case shouldn't happen
            % disp('Unexpected case: idx containing minimum value = 3')
        end

        % find the 'optimal' alpha using a quadratic line search
        a_star = qline_search(a1, f1, a2, f2, a3, f3);

        % set a new 'x' value using a_star and the current search dir
        x_plus = x + a_star*s;

        % sl = norm(a_star*s);

        % evaluate the gradient at this new 'xplus' location
        grad_plus = gradobj(x_plus);

        % compute beta
        beta = (grad_plus'*grad_plus)/(grad'*grad);

        % find our new conjugate gradient search direction to go in
        s_plus = -grad_plus + beta * s;

        % reset alpha down to a small number
        alpha = alpha0;

        % re-assign variable names for the next loop
        x = x_plus;
        s = s_plus;
        grad = grad_plus;

        % increment the iteration count
        num_iterations = num_iterations + 1;

        % store data if we're going to plot
        if plot_func == 1 && num_iterations < max_data_rows
            x_vals(num_iterations + 1,:) = x';
        end

        % check to see if the gradient is close enough to zero
        % i.e. we're at the optimum.
        if norm(grad) < stoptol
            break
        end

    else
        % f isn't changing so we've reached an optimum?
```

```
            break
        end


    end
end

%% BFGS quasi-Newton
if algoflag == 3  % bfgs quasi-Newton
    while num_iterations < max_iterations

        % start going along the function in the direction of s
        xnew = x + alpha*s;

        % evaluate the function at xnew
        fnew = obj(xnew);

        % if we're making progress going in the s direction...
        if fnew < f

            % double alpha and then keep going
            alpha = alpha*2;

            % set f equal to fnew and go again
            f = fnew;

            % Here we know we've gone too far.  Now we can conduct a
            % line search, find the approprate alpha, and start working
            % in a new search direction.
        elseif fnew > f

            a1 = alpha/4;
            a2 = alpha/2;
            a3 = alpha;
            a4 = (a3 + a2)/2;

            % Now we need to pick the 3 points we will use
            % to bracket the minimum and perform the quadratic
            % line search.

            % evaluate the function at each
            f1 = obj(x + a1*s);
            f2 = obj(x + a2*s);
            f3 = obj(x + a3*s);
            f4 = obj(x + a4*s);

            % find the minimum point
            [~, idx] = min([f1, f2, f3, f4]);

            if idx == 2
                a1 = a1;
                a2 = a2;
                a3 = a4;

                f1 = f1;
                f2 = f2;
                f3 = f4;
```

```
elseif idx == 4
   a1 = a2;
   a2 = a4;
   a3 = a3;

   f1 = f2;
   f2 = f4;
   f3 = f3;

elseif idx == 1

   % in this case we need to go one more step back in the
   % past. we also need to compute these in reverse order.

   a3 = a2;
   a2 = a1;
   a1 = a1/2;


   f3 = f2;
   f2 = f1;
   f1 = obj(x + a1*s);

else
   % this case shouldn't happen
   % disp('Unexpected case: idx containing minimum value = 3')
end

% find the 'optimal' alpha using a quadratic line search
a_star = qline_search(a1, f1, a2, f2, a3, f3);

% set a new 'x' value using a_star and the current search dir
x_plus = x + a_star*s;

% sl = norm(a_star*s);

% evaluate the gradient at this new 'xplus' location
grad_plus = gradobj(x_plus);

 % find delta x
delta_x = x_plus - x;

% find gamma
gamma = grad_plus - grad;

% find N_plus using BFGS Update
N_plus = N + ((1 + ((gamma'*N*gamma)/(delta_x'*gamma)))...
   *((delta_x*delta_x')/(delta_x'*gamma)))...
   - ((delta_x*gamma'*N + N*gamma*delta_x')/(delta_x'*gamma));

% compute new search direction
s_plus = -N_plus * grad_plus;

% reset alpha down to a small number
alpha = alpha0;

% re-assign variable names for the next loop
x = x_plus;
```

```
            s = s_plus;
            grad = grad_plus;
            N = N_plus;

            % increment the iteration count
            num_iterations = num_iterations + 1;

            % store data if we're going to plot
            if plot_func == 1 && num_iterations < max_data_rows
                x_vals(num_iterations + 1,:) = x';
            end

            % check to see if the gradient is close enough to zero
            % i.e. we're at the optimum.
            if norm(grad) < stoptol
                break
            end

        else
            % f isn't changing so we've reached an optimum?
            break
        end
    end
end

%%%%%%%%%%%%%%%%%
% function exit %
%%%%%%%%%%%%%%%%%

if num_iterations >= max_iterations
    max_iter_str = num2str(max_iterations);
    err_str = strcat('Failed to converge to an optimum after ', max_iter_str, ' iterations.');
    disp([newline, err_str])
    return;
end

if plot_func == 1
    plot_rosenbrock(algoflag, num_iterations, x_vals)
end



num_iterations
xopt = x;
fopt = obj(x);
exitflag = 0;

end

% get steepest descent search direction as a column vector
function [s] = srchsd(grad, normalized)
if normalized == 1
    mag = sqrt(grad'*grad);
    s = -grad/mag;
else
    s = -grad;
end
end
```

```
% quadratic line search
function a_star = qline_search(a1, f1, a2, f2, a3, f3)

num = f1*(a2^2 - a3^2) + f2*(a3^2 - a1^2) + f3*(a1^2 - a2^2);
den = 2*(f1*(a2 - a3) + f2*(a3 - a1) + f3*(a1 -a2));

a_star = num/den;
end
```