Assignment 1

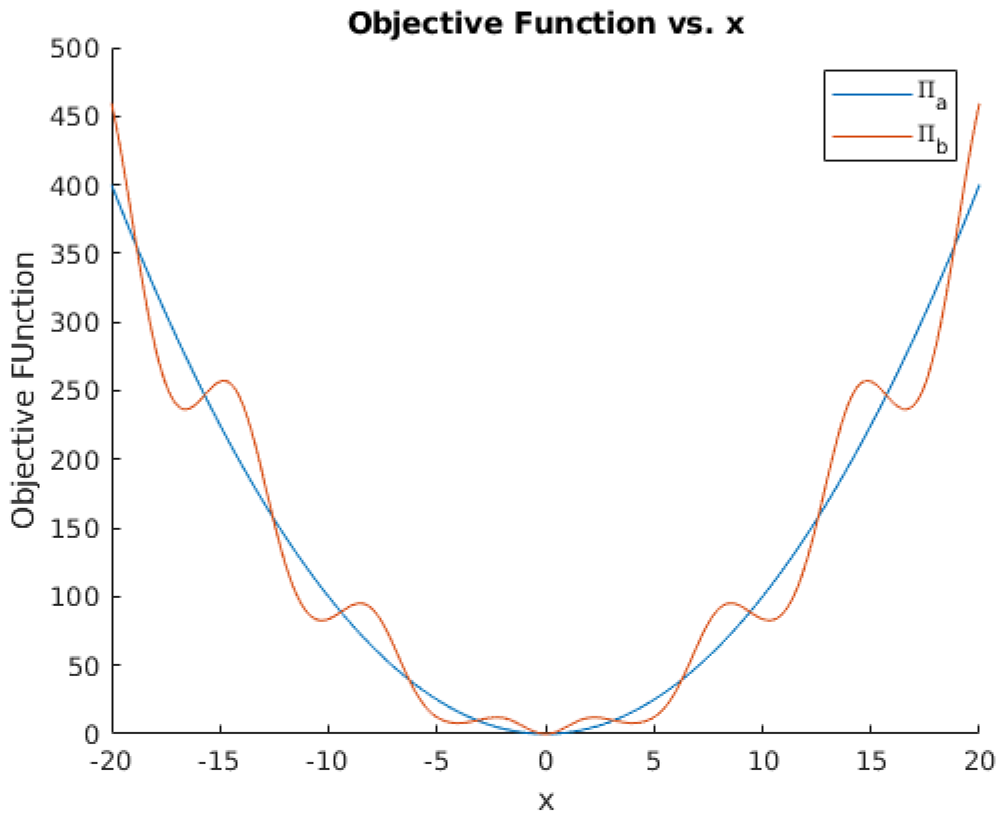
Table of Contents

Dylan Callaway Fall 2019 E150

Netwon's Method

1

```
clear; clc;

x = linspace(-20, 20, 1000);
pi_a = x.^2;
pi_b = (x + (pi./2).*sin(x)).^2;

figure;
hold on
plot(x, pi_a)
plot(x, pi_b)
title('Objective Function vs. x')
xlabel('x')
ylabel('Objective FUnction')
legend('\Pi_a', '\Pi_b')
hold off
```



2

3

4

5

```
% In file called myNewton.m
```

6

```
close all
clear
clc

syms x

pi_a = x^2;
pi_b = (x + (pi/2)*sin(x))^2;
```

```
f_a = gradient(pi_a)
f_b = gradient(pi_b)
df_a = hessian(pi_a)
df_b = hessian(pi_b)

pi_a = matlabFunction(pi_a, 'Vars', x);
pi_b = matlabFunction(pi_b, 'Vars', x);
f_a = matlabFunction(f_a, 'Vars', x);
f_b = matlabFunction(f_b, 'Vars', x);
df_a = matlabFunction(df_a, 'Vars', x);
df_b = matlabFunction(df_b, 'Vars', x);
```

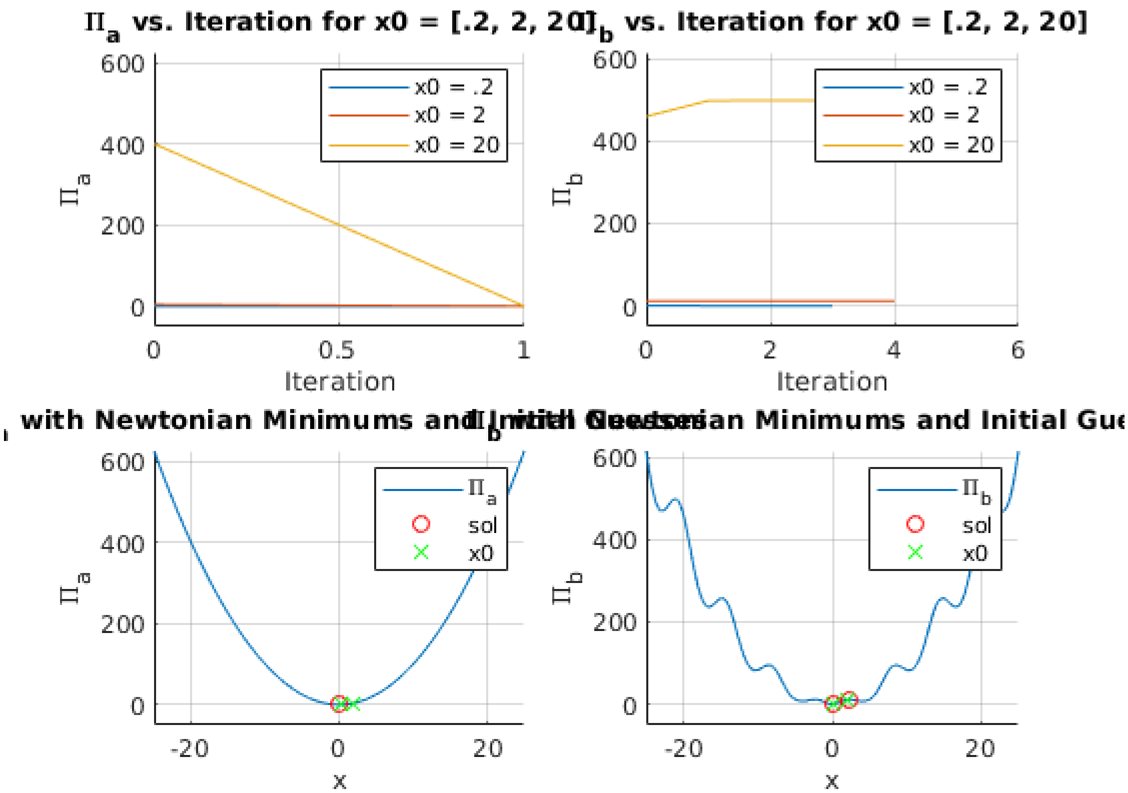
```
f_a =

2*x


f_b =

2*(x + (pi*sin(x))/2)*((pi*cos(x))/2 + 1)


df_a =

2


df_b =

2*((pi*cos(x))/2 + 1)^2 - pi*sin(x)*(x + (pi*sin(x))/2)
```

7

```
k = [-1, 0, 1];
x0s = 2.*10.^k;
TOL = 10^-8;
maxit = 20;
sol_a = [];
sol_b = [];

subplot(2, 2, 1)
for x0 = x0s
    [sol, its, hist] = myNewton(f_a, df_a, x0, TOL, maxit);
    sol_a = [sol_a, sol];
    hold on
    plot(0:length(hist)-1,pi_a(hist))
end
title('\Pi_a vs. Iteration for x0 = [.2, 2, 20]')
xlabel('Iteration')
ylabel('\Pi_a')
legend('x0 = .2', 'x0 = 2', 'x0 = 20')
```

```
grid on
ylim([-50, pi_a(25)])
hold off

subplot(2, 2, 2)
for x0 = x0s
    [sol, its, hist] = myNewton(f_b, df_b, x0, TOL, maxit);
    sol_b = [sol_b, sol];
    hold on
    plot(0:length(hist)-1,pi_b(hist))
end
title('\Pi_b vs. Iteration for x0 = [.2, 2, 20]')
xlabel('Iteration')
ylabel('\Pi_b')
legend('x0 = .2', 'x0 = 2', 'x0 = 20')
grid on
ylim([-50, pi_b(25)])
hold off

vals = linspace(-25, 25, 1000);

subplot(2, 2, 3)
hold on
plot(vals, pi_a(vals))
plot(sol_a, pi_a(sol_a), 'ro')
plot(x0s, pi_a(x0s), 'gx')
ylim([-50, pi_a(25)])
grid on
legend('\Pi_a', 'sol', 'x0')
title("\Pi_a with Newtonian Minimums and Initial Guesses")
xlabel("x")
ylabel("\Pi_a")
hold off

subplot(2, 2, 4)
hold on
plot(vals, pi_b(vals))
plot(sol_b, pi_b(sol_b), 'ro')
plot(x0s, pi_b(x0s), 'gx')
ylim([-50, pi_b(25)])
grid on
legend('\Pi_b', 'sol', 'x0')
title("\Pi_b with Newtonian Minimums and Initial Guesses")
xlabel("x")
ylabel("\Pi_b")
hold off

% From graph 1 (Pi_a) you can see that Newton's method converged to 0
% for all x0. Graph 2 (Pi_b) shows that the solution changes depending
 on
% x0. This is due to there being local extrema throughout Pi_b.
% This can further be seen in graphs 3 and 4, where the solution for
% Pi_a is always 0, even as x0 moves away rom the global extrema,
 while Pi_b's
```

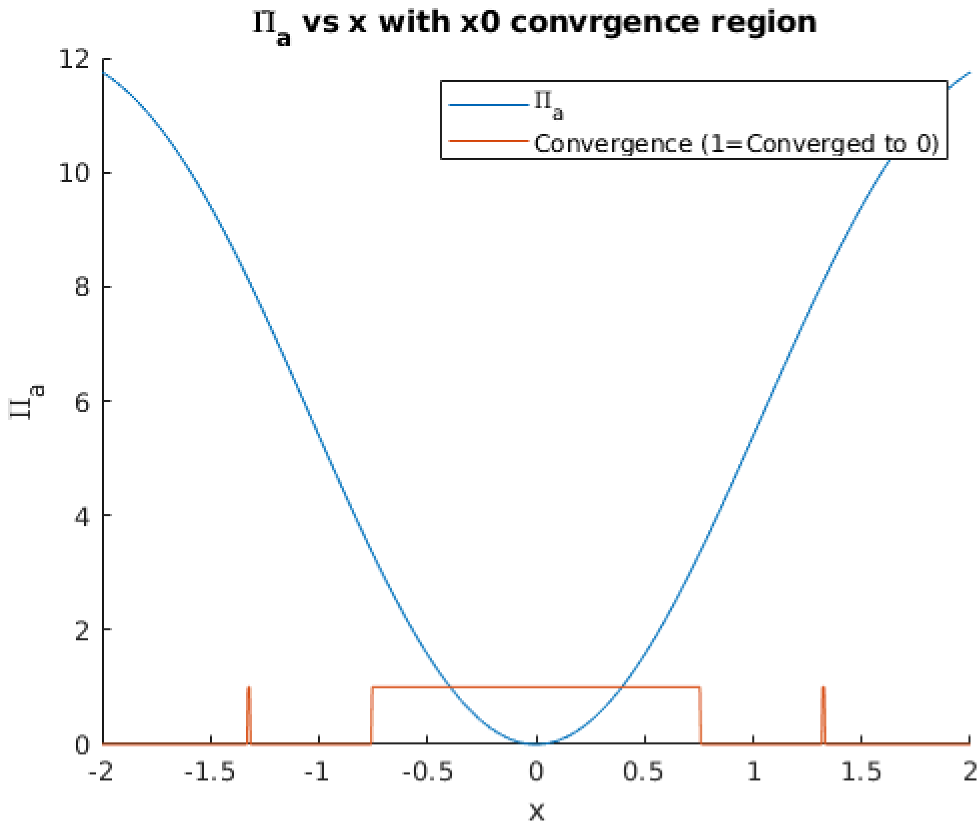
```
% solution is always the extrema nearest the initial guess. Pi_a also
 converges in 2
% iterations regardless of x0, while Pi_b takes 4 or 5 iterations to
% converge to the nearest local extrema.
```



8

```
close all
clc

% Newton's method will find the global minimum of Pi_a for all x0.

% My intuition tells me that x0 would need to be between the zeros of
 H
% around the global min of Pi_b...but this does not prove to be true
 when I
% use myNetwon on Pi_b.

test_x0 = linspace(-2, 2, 1000);
solns = []
for x0 = test_x0
    [sol, its, hist] = myNewton(f_b, df_b, x0, TOL, maxit);
    solns = [solns, sol<10^-8 && sol>-10^-8];
end

figure
```

```
hold on
plot(test_x0, pi_b(test_x0))
plot(test_x0,solns)
title("\Pi_a vs x with x0 convrgence region")
xlabel("x")
ylabel("\Pi_a")
legend("\Pi_a", "Convergence (1=Converged to 0)")
hold off

% From this plot you can see that sol converges to zero for x0 =
 [-.75, .75].

% This means x would need at least 1 point in [-.75, .75]. This is
% accomplished with
ng = 32


solns =

     []


ng =

    32
```



9

```
close all
clc

syms x y z

pi_bx = (x + (pi/2)*sin(x))^2;
pi_by = (y + (pi/2)*sin(y))^2;
pi_bz = (z + (pi/2)*sin(z))^2;

pi_b2 = pi_bx + pi_by;
pi_b3 = pi_bx + pi_by + pi_bz;


f_b2 = gradient(pi_b2, [x, y])
df_b2 = hessian(pi_b2, [x, y])
f_b3 = gradient(pi_b3, [x, y, z])
df_b3 = hessian(pi_b3, [x, y, z])

pi_b2 = matlabFunction(pi_b2, 'Vars', [x, y]);
pi_b3 = matlabFunction(pi_b3, 'Vars', [x, y, z]);
f_b2 = matlabFunction(f_b2, 'Vars', [x, y]);
df_b2 = matlabFunction(df_b2, 'Vars', [x, y]);
f_b3 = matlabFunction(f_b3, 'Vars', [x, y, z]);
df_b3 = matlabFunction(df_b3, 'Vars', [x, y, z]);
```

```
f_b2 =

 2*(x + (pi*sin(x))/2)*((pi*cos(x))/2 + 1)
 2*(y + (pi*sin(y))/2)*((pi*cos(y))/2 + 1)


df_b2 =

[ 2*((pi*cos(x))/2 + 1)^2 - pi*sin(x)*(x + (pi*sin(x))/2),
                                                     0]
[                                                            0,
 2*((pi*cos(y))/2 + 1)^2 - pi*sin(y)*(y + (pi*sin(y))/2)]


f_b3 =

 2*(x + (pi*sin(x))/2)*((pi*cos(x))/2 + 1)
 2*(y + (pi*sin(y))/2)*((pi*cos(y))/2 + 1)
 2*(z + (pi*sin(z))/2)*((pi*cos(z))/2 + 1)


df_b3 =
```

```
[ 2*((pi*cos(x))/2 + 1)^2 - pi*sin(x)*(x + (pi*sin(x))/2),
                                                          0,
                              0]
[                                                         0,
 2*((pi*cos(y))/2 + 1)^2 - pi*sin(y)*(y + (pi*sin(y))/2),
                                                          0]
[                                                         0,
                                                          0, 2*((pi*cos(z))/2 + 1)^2
 - pi*sin(z)*(z + (pi*sin(z))/2)]
```

10

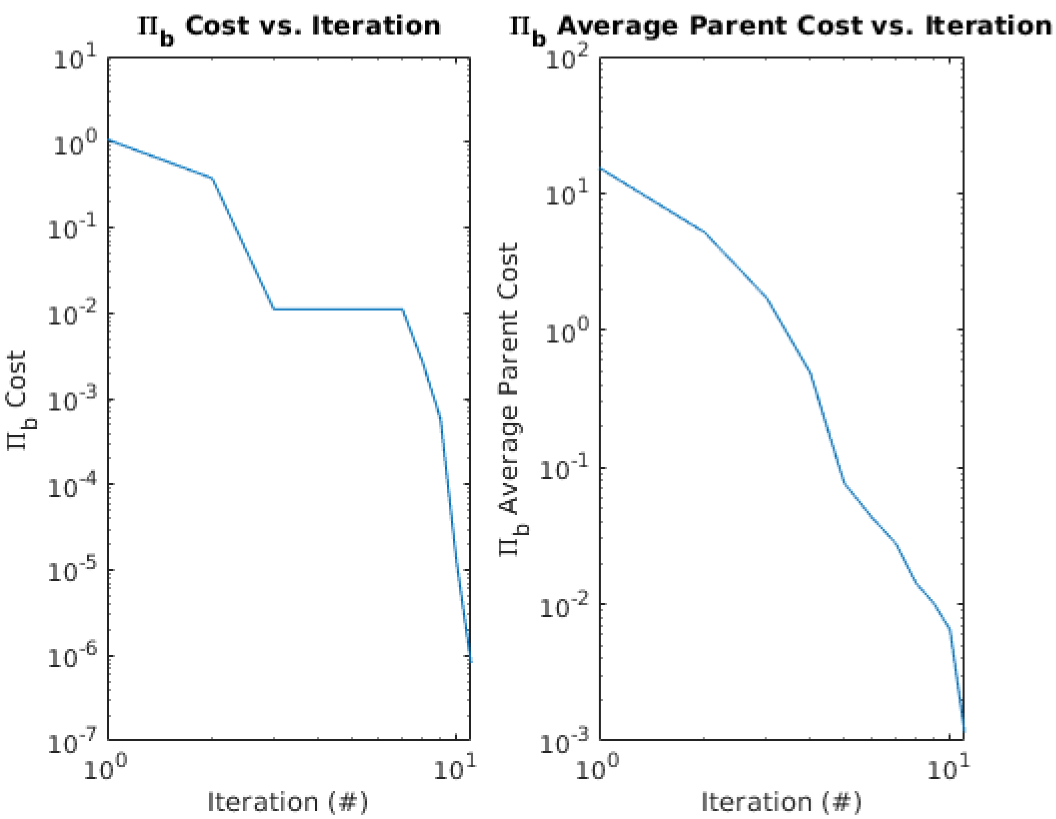
```
clc

x = linspace(-20, 20, ng);
y = x; z = x;

% 1D
time_sum = 0;
run_nums = 100;
min_val = Inf;

for run = 1:run_nums
    min_val = Inf;
    tic
    for i = x
        val = pi_b(i);
        if val<min_val
            min_val = val;
        end
    end
    time_sum = toc + time_sum;
end
time_avg1 = time_sum/run_nums

% 2D
time_sum = 0;
min_val = Inf;

for run = 1:run_nums
    min_val = Inf;
    tic
    for i = x
        for j = y
            val = pi_b2(i, j);
            if val<min_val
                min_val = val;
            end
        end
    end
    time_sum = toc + time_sum;
end
```

```
time_avg2 = time_sum/run_nums

% 3D
time_sum = 0;
min_val = Inf;

for run = 1:run_nums
    min_val = Inf;
    tic
    for i = x
        for j = y
            for k = z
                val = pi_b3(i, j, k);
                if val<min_val
                    min_val = val;
                end
            end
        end
    end
    time_sum = toc + time_sum;
end
time_avg3 = time_sum/run_nums

num_vars = [1, 2, 3];
times = [time_avg1, time_avg2, time_avg3];

quad_fit = polyfit(num_vars, times, 2);

% Based on 1D, 2D, and 3D optimization, 15D optimization would take
% approximately this long:
time_avg15 = polyval(quad_fit, 15)

% Changing ng from 32 to 100 increased the 3D run time to .1505s, and
 the 15D run time to 13.5s.


time_avg1 =

   2.3240e-05


time_avg2 =

   3.9561e-04


time_avg3 =

    0.0059


time_avg15 =

    0.4742
```

Genetic Algorithm

1

```
close all
clc

[PI, Orig, Lambda] = myGenetic(pi_b, [-20, 20], 12, 10^-6, 100, 50,
 1);
subplot(1, 2, 1)
loglog(PI(:,1))
title('\Pi_b Cost vs. Iteration')
xlabel('Iteration (#)')
ylabel('\Pi_b Cost')
subplot(1, 2, 2)
parent_cost = PI(:,1:12);
avg_parent_cost = mean(parent_cost, 2);
loglog(avg_parent_cost)
title('\Pi_b Average Parent Cost vs. Iteration')
xlabel('Iteration (#)')
ylabel('\Pi_b Average Parent Cost')
```



2

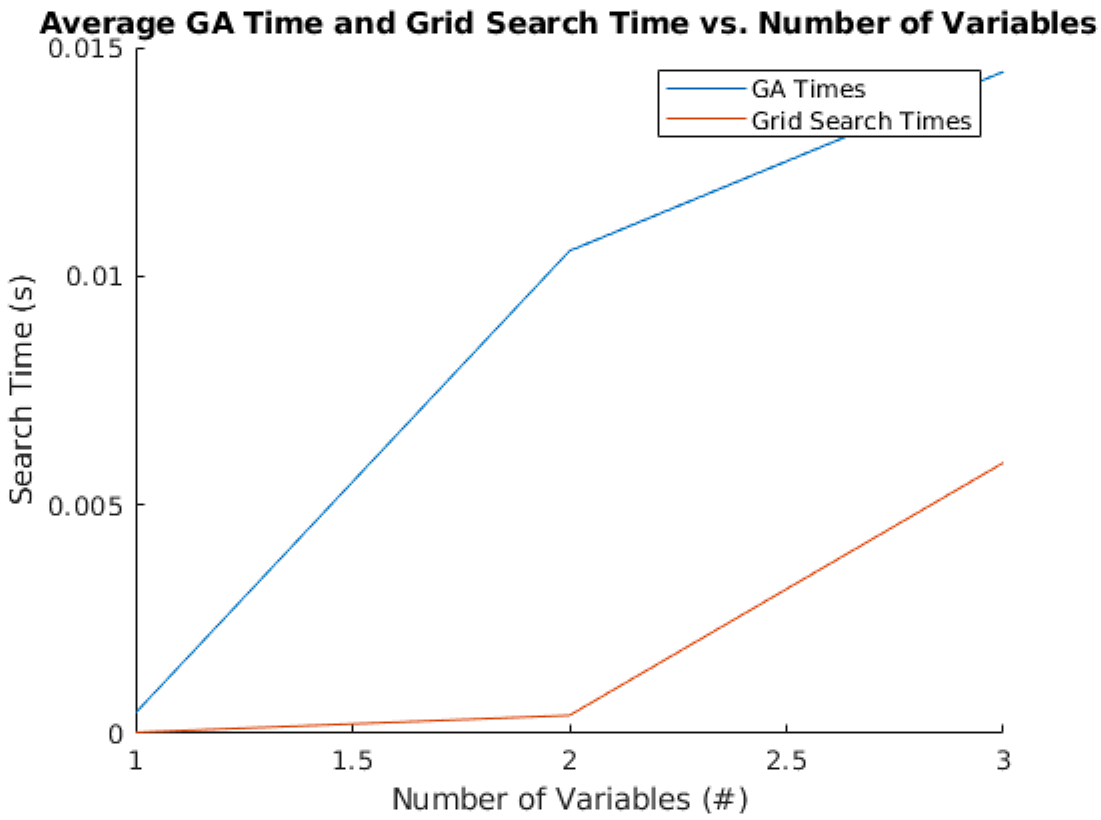
```
close all
clc

parents = [0, 6, 12];
GA_Times = [];
GA_Failures = [];

for parent = parents
    parent
    time_sum1 = 0;
    time_sum2 = 0;
    time_sum3 = 0;

    for i = 1:100
        tic;
        [PI, Orig, Lambda] = myGenetic(pi_b, [-20, 20], parent, 10^-1,
 100, 50, 1);
        [sol, its, hist] = myNewton(f_b, df_b, Lambda(1,:), 10^-6,
 20);
        time_sum1 = time_sum1 + toc;
        success1(i) = pi_b(sol) < 10^-6;

        tic;
        [PI, Orig, Lambda] = myGenetic(pi_b2, [-20, 20], parent,
 10^-1, 100, 50, 2);
        [sol, its, hist] = myNewton(f_b2, df_b2, Lambda(1,:), 10^-6,
 20);
        time_sum2 = time_sum2 + toc;
        func_vals = num2cell(sol);
        success2(i) = pi_b2(func_vals{:}) < 10^-6;

        tic;
        [PI, Orig, Lambda] = myGenetic(pi_b3, [-20, 20], parent,
 10^-1, 100, 50, 3);
        [sol, its, hist] = myNewton(f_b3, df_b3, Lambda(1,:), 10^-6,
 20);
        time_sum3 = time_sum3 + toc;
        func_vals = num2cell(sol);
        success3(i) = pi_b3(func_vals{:}) < 10^-6;
    end

    avg1 = time_sum1/100;
    avg2 = time_sum2/100;
    avg3 = time_sum3/100;

    GA_Times = [GA_Times; avg1, avg2, avg3]; % Rows are each number of
 parents, columns are pi_b, pi_b2, pi_b3
    fail1 = (100-sum(success1))/100;
    fail2 = (100-sum(success2))/100;
    fail3 = (100-sum(success3))/100;
    GA_Failures = [GA_Failures; fail1, fail2, fail3]; % Same as times
```

```
end
GA_Times
GA_Failures
```

```
parent =

     0


parent =

     6


parent =

    12


GA_Times =

    0.0006    0.0119    0.0140
    0.0004    0.0103    0.0140
    0.0003    0.0094    0.0154


GA_Failures =

    0.9500    1.0000    1.0000
         0         0    0.5200
         0         0    0.5200
```

3

```
close all
clc

GA_Times_trans = mean(GA_Times, 1)';
figure
hold on
plot(GA_Times_trans)
plot(times)
title('Average GA Time and Grid Search Time vs. Number of Variables')
xlabel('Number of Variables (#)')
ylabel('Search Time (s)')
legend('GA Times', 'Grid Search Times')
hold off
```



4

```
close all
clc

% The case with zero parents represents random guesses on the
 interval. It
% does not perform well because it is essentially a poorly organized
 grid
% search, with no way to keep memory of the good guesses.
```

5

```
% There are effectively 2 metrics by which these algorithms can be
% compared across. Speed (time to find a solution) and robustness (how
 likely that solution is to be the right one).

% Grid search can find a solution quickly at the cost of robustness,
 as
% shown in #10 of the Newton's method section. If ng were increased
 such
% that each point in the grid had only 0.0001 between the next, it
 would
% take a very long time to run, but would be able to find the minimum
% within 0.00005.
```

```
% Newton's method can be extremely fast and robust, converging quickly
 to
% the correct minimum as long as some information about the function
 is
% known beforehand (the approximate location of the global minimum).
% However, if this information is not known, Newton's method can be
% extremely unreliable due to it getting "stuck" on local extrema.

% The genetic algorithm is a good balance between speed and
 robustness. It
% can converge as quickly as Newton's, but does not get stuck on
% local extrema due to the randomness of its initial guesses (like in
 a
% grid search). However, it does not evaluate every guess, like a grid
% search does, as it maintains some memory/information about previous
% guesses, like Newton's method does by utilizing the Gradient and
 Hessian.

% Furtheremore, some of these methods have limitations on the types of
% functions that they can be used on. In particular, Newton's method
 requires that the
% function be twice differentiable for the variables of interest. The
 other
% two methods can be used on any function or set of dependent data.
```

Published with MATLAB® R2019a