ME201 Project 1

Tristan Schwab

February 2022

1 Graphing Objective Functions

Shown below are two objective functions: $\Pi_a = x^2$ and $\Pi_b = (x + \frac{\pi}{2} sin(x))^2$

To locate minima in these functions, we use the Newton-Raphson method– which is able to find minima for low-order functions, such as Π_a. This method is limited in finding absolute minima for higher order functions such as Π_b.

When we initialize the code (see appendix), we notice that for each initial guess, x_0, Π_a is able to locate the absolute minimum within two iterations for all values of x_0. In contrast, a minimum using the Newton-Raphson method on Π_b is largely dependent on the initial conditions. For $x_0 = 0.2$, we can locate the absolute minimum within two iterations, which is to be expected from (1). For other values of x_0, the function fails at locating an absolute minimum due to concavity/convexity of Π_b (2). Another method should be utilized.

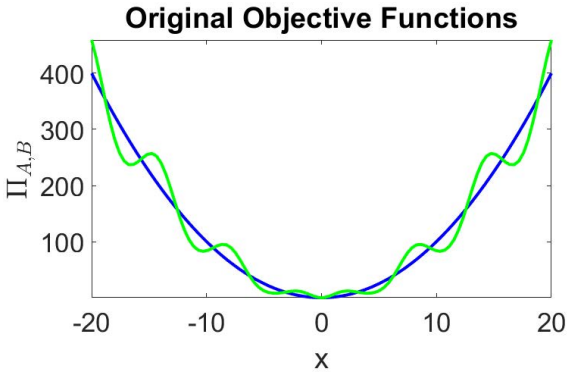


Figure 1: Plotting both objective functions on the domain -20 $\leq$ x $\leq$ 20

2 Genetic Algorithm Implementation

To locate an absolute minimum regardless of the function behavior, a genetic algorithm is used. Four runs were performed on Π_b (and two runs on Pi_b. The

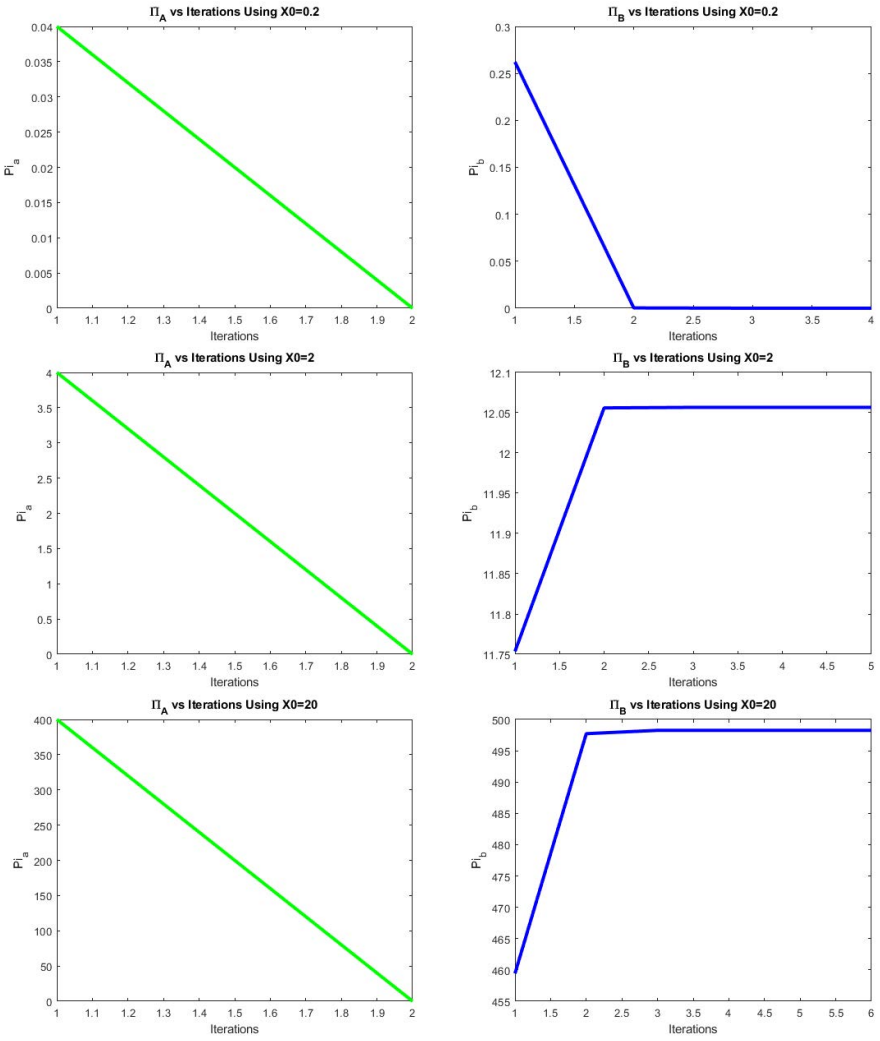


Figure 2: Plot of $\Pi(hist)$ for Π_a Π_b

cost of the best design strings for each generation is shown (3). The variety of genetic fitness decreases (that is, with each generation, variation approaches zero). The average cost remains relatively level throughout the iterations, though the minimum cost decreases with each generation.

The performance of the algorithm is dependent upon the parents passing information to the new generation. With each generation, a fitness (in this case, assessing the minimum value of the string from the parents is made). If genetic information does not make it to the new generation, the algorithm falls apart, as a new generation will have to start with it's minimum rather than the entire tree's minimum values. Run-time would increase, because the sequencing would end when the maximum iterations is reached (see Appendix).

The performance of the algorithm is dependent on the use-case. In a perfect-world of continuous functions, the Newton-Raphson method will generally perform faster than the Genetic Algorithm. Unfortunately, most real-world functions will not be as perfect, and using the N-R method would not yield correct results. To maintain accuracy, the Genetic Algorithm should be used for optimization of complicated functions as long as the experimenter has an expectation of the final result. Run-time of the GA will ultimately depend on the number of sequences generated and the complexity of the function.

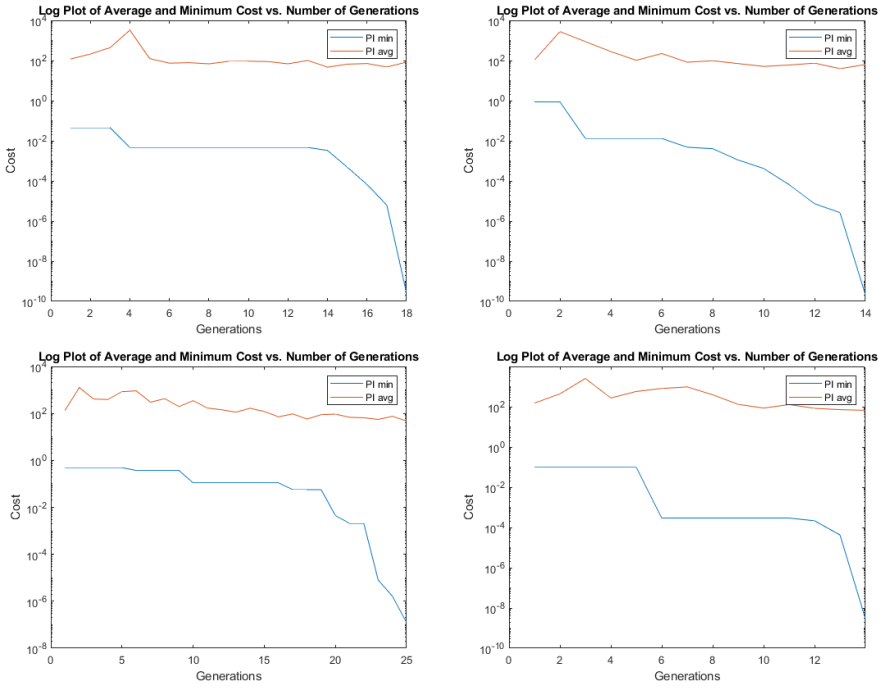


Figure 3: Cost of the best design and mean cost of all of the design strings for each generation Π_b

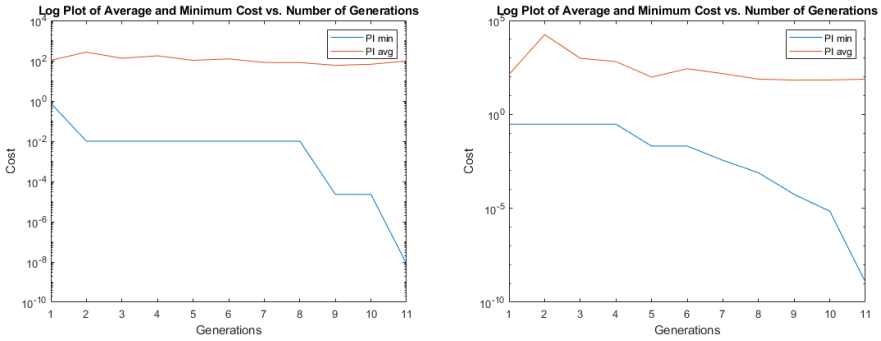


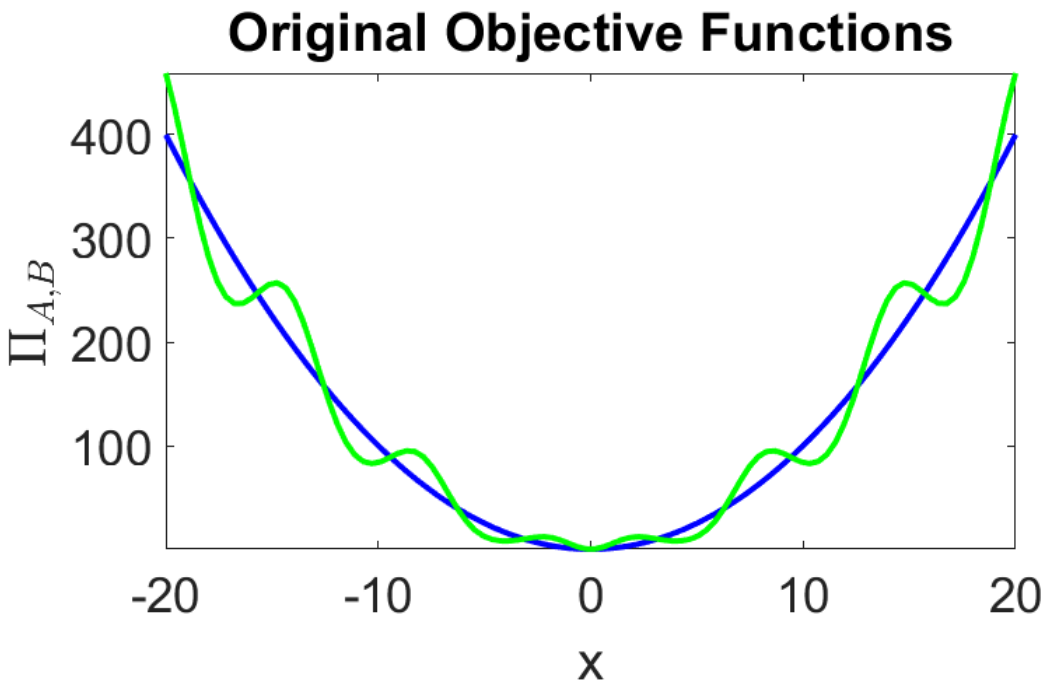
Figure 4: Cost of the best design and mean cost of all of the design strings for each generation Π_a

Appendix


```
% Tristan Schwab, UC BERKELEY ME, tristanschwab@berkeley.edu
% Prepared for ME201, Project One

save_images = 1; % set this to 1 to save outputs
output_folder = "./output_images";
if save_images &&  ~(isfolder(output_folder))
    mkdir(output_folder);
end
```

Graphing Objective Functions

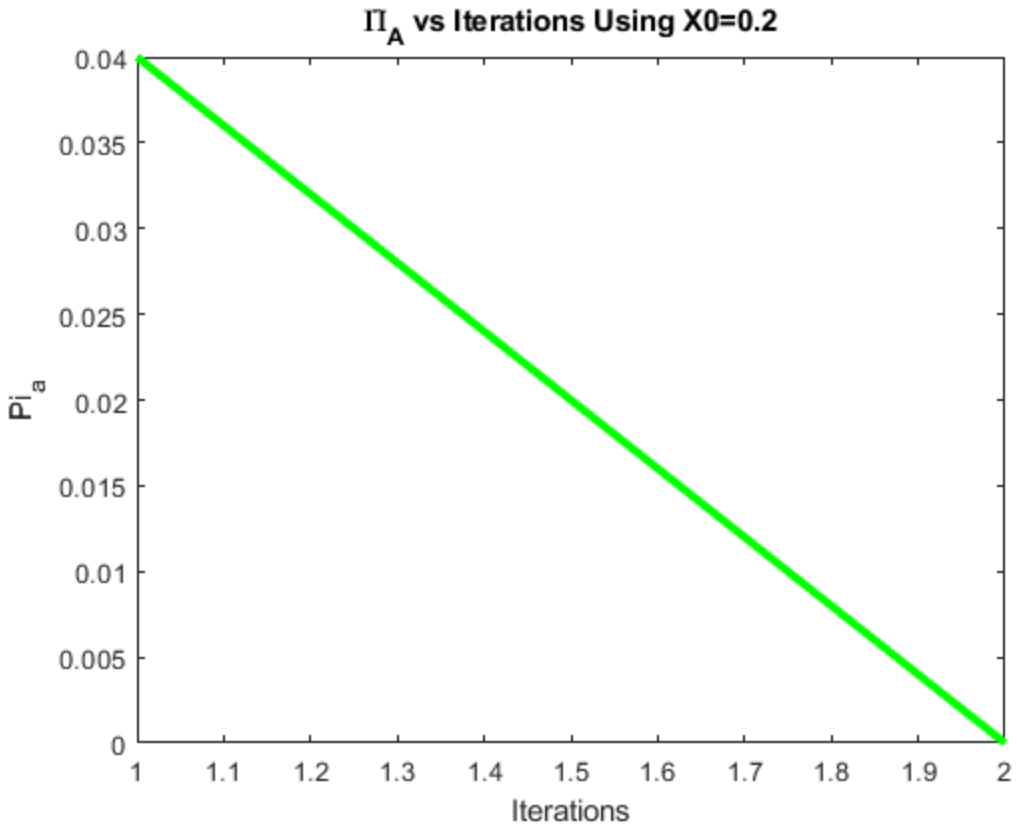
```
% House-Cleaning
close all % close all open figures
clc % clear command window
clear

% Definitions
x = linspace(-20,20);
P1=x.^2;
P2=(x+(pi/2)*sin(x)).^2;
g=diff(P1);
h=diff(P2);

%Plotting
figure("name", "Objective Functions")  % Generate new named figure for
 plot
plot(x, P1,'b', 'LineWidth', 3) % use 'LineWidth' attribute to
 increase line
% thickness. Makes plot lines a lot easier to see
hold on
plot(x,P2,'g', 'LineWidth',3)
ylabel('$\Pi_{A,B}$','Interpreter','latex')
xlabel("x")
title('Original Objective Functions') % Always include title, axis
 labels with
set(gca,'FontSize', 26); % Use this command to increase text size on
 plot
axis tight % make axis crop out any dead space
x0 = 0; y0 = 0; width = 800; height = 500;
set(gcf,'units','pixels','position',[x0,y0,width,height])
```

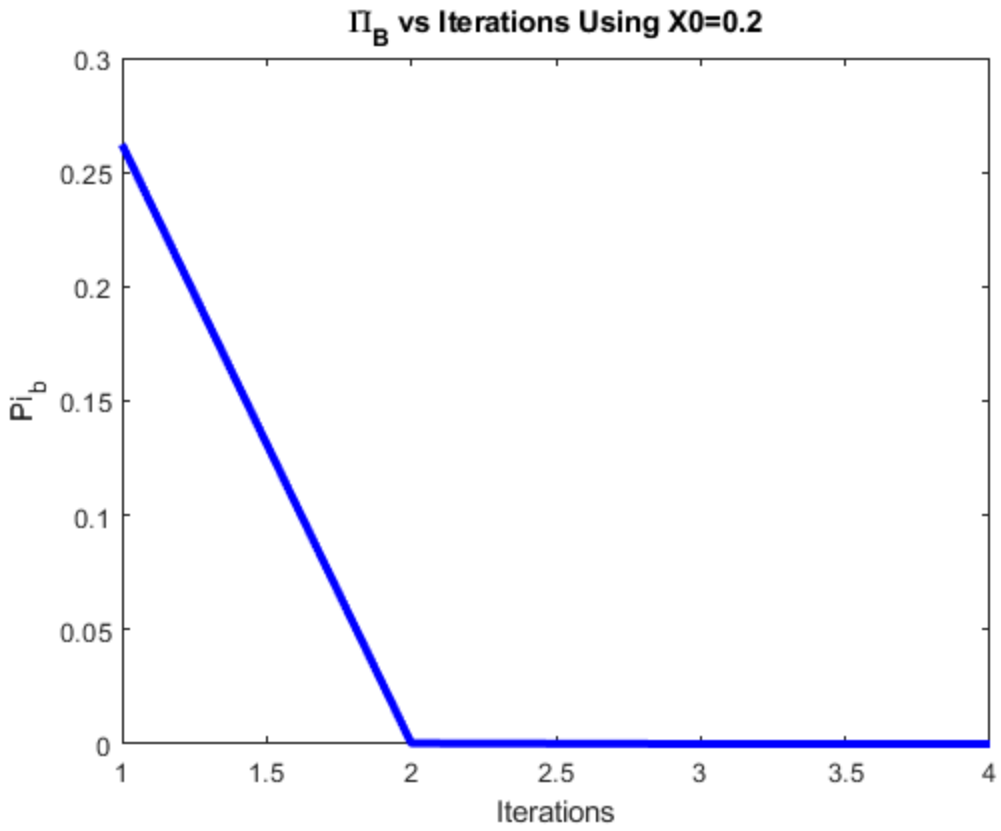


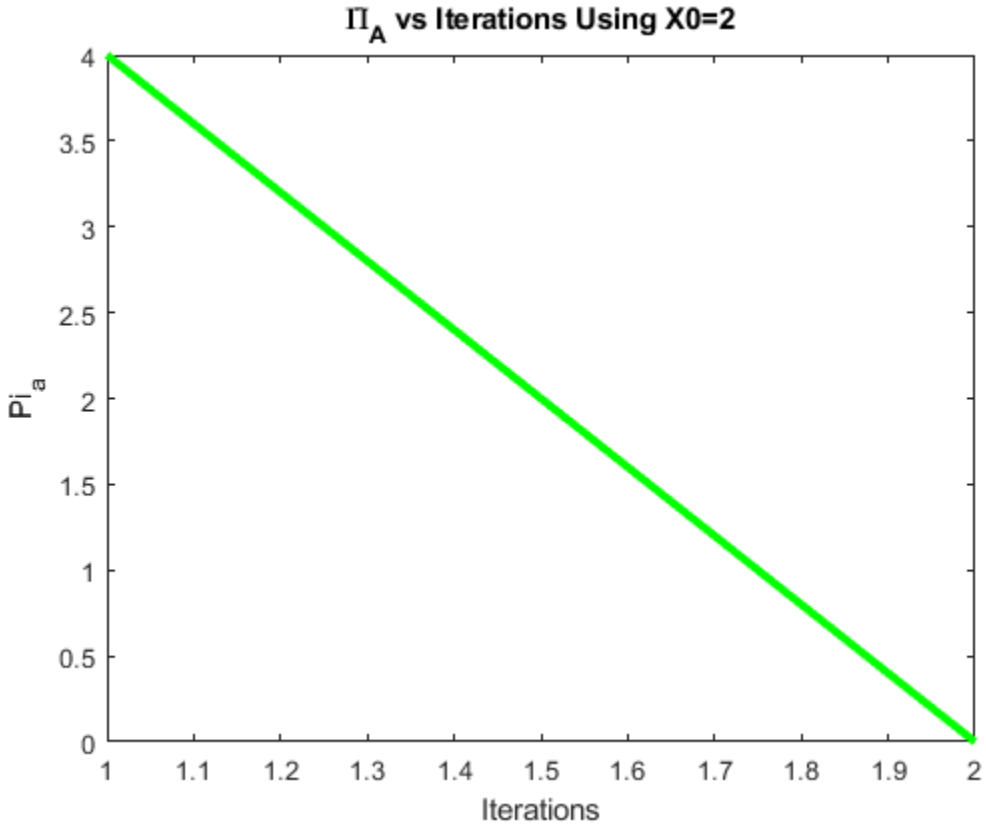
Algorithm for Newton-Raphson

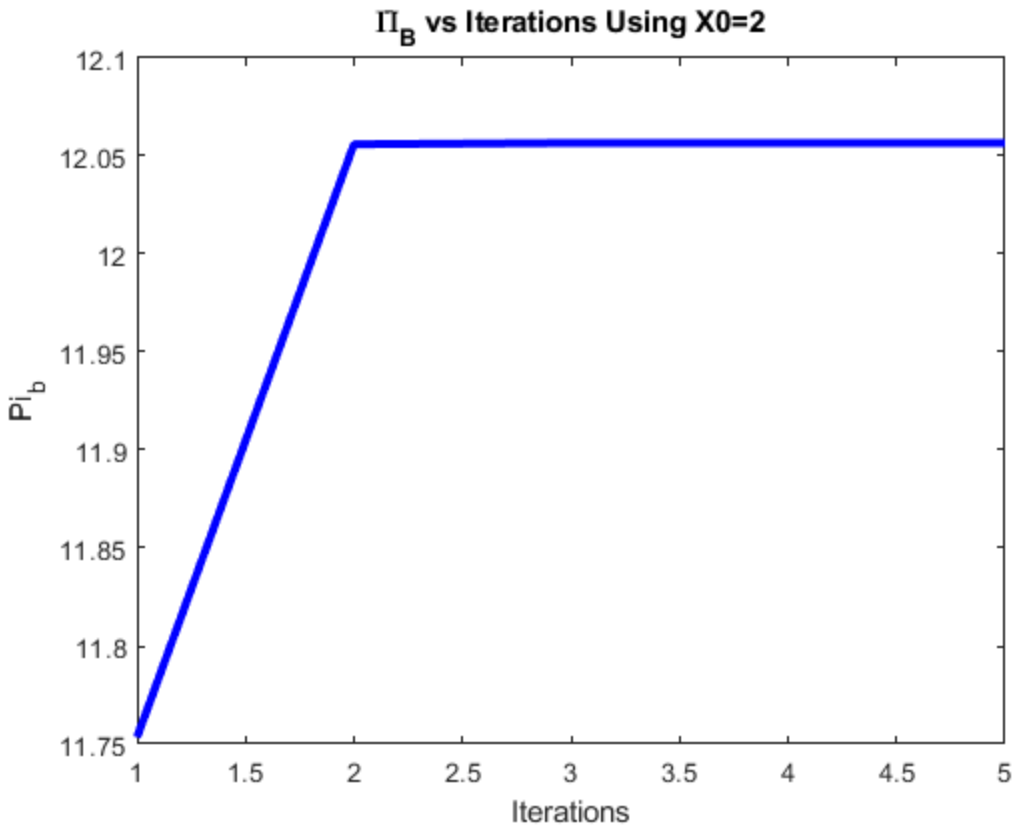
```
%House-Cleaning
close all % close all open figures
clc % clear command window
clear

%Definitions
syms x Pi_a(x) Pi_b(x)
Pi_a(x)=x.^2; Pi_b(x)=(x+(pi/2)*sin(x)).^2;
f_a = diff(Pi_a,x); f_b = diff(Pi_b,x);
df_a = matlabFunction(hessian(Pi_a,x));
df_b = matlabFunction(hessian(Pi_b,x));

%Initial Conditions
TOL = 10^(-8);
maxit = 20;
g=[-1,0,1];
hist_a1=zeros(1,length(g));
hist_b1=zeros(1,length(g));
for k = g
    x0 = 2*10^(k);
    %Evaluate Pi_A
    [sol_a, its_a, hist_a] = myNewton(matlabFunction(f_a), df_a, x0,
 TOL, maxit);
    %Pi_a=[Pi_a,sol_a];

    %Evaluate Pi_B
    [sol_b, its_b, hist_b] = myNewton(matlabFunction(f_b), df_b, x0,
 TOL, maxit);
```

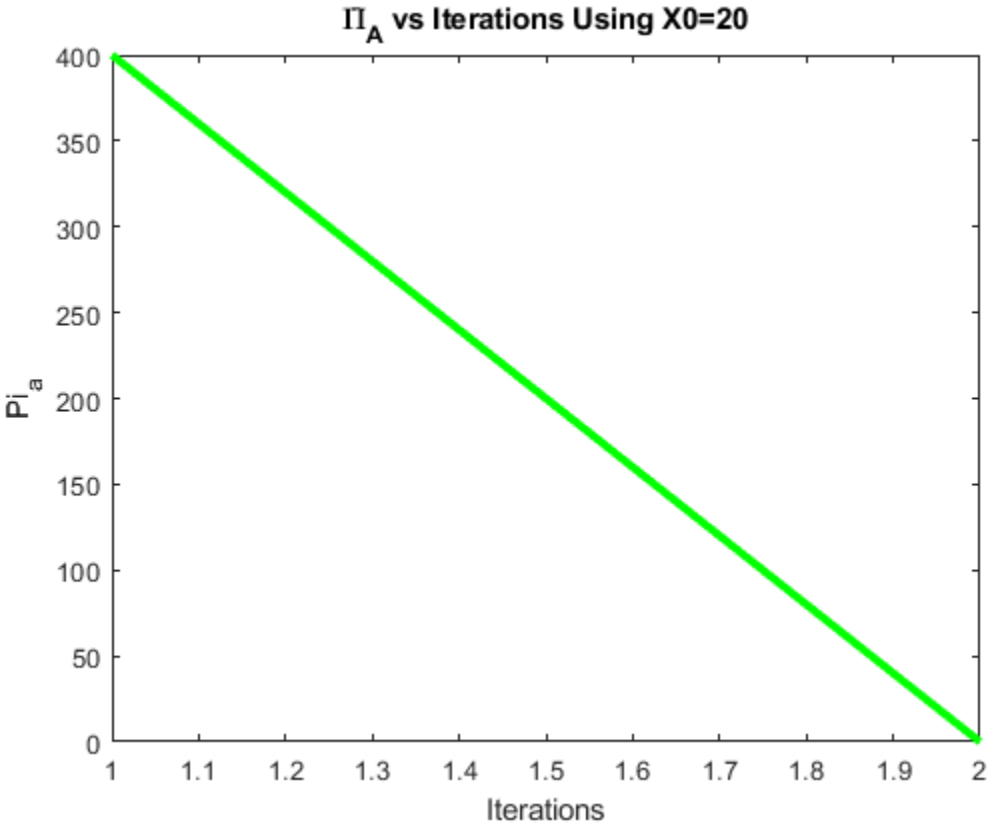
```
    %Pi_b=[Pi_b,sol_b];

    % Plotting
    figure("name","Pi A");
    plot(1:1:its_a+1, subs(Pi_a,hist_a),'g','LineWidth',3);
    title(['\Pi_A vs Iterations Using X0=' num2str(x0)])
    xlabel('Iterations')
    ylabel('Pi_a')
    hold on
    figure("name","Pi B");
    plot(1:1:its_b+1, subs(Pi_b,hist_b),'b','LineWidth',3);
    title(['\Pi_B vs Iterations Using X0=' num2str(x0)])
    xlabel('Iterations')
    ylabel('Pi_b')
end
```

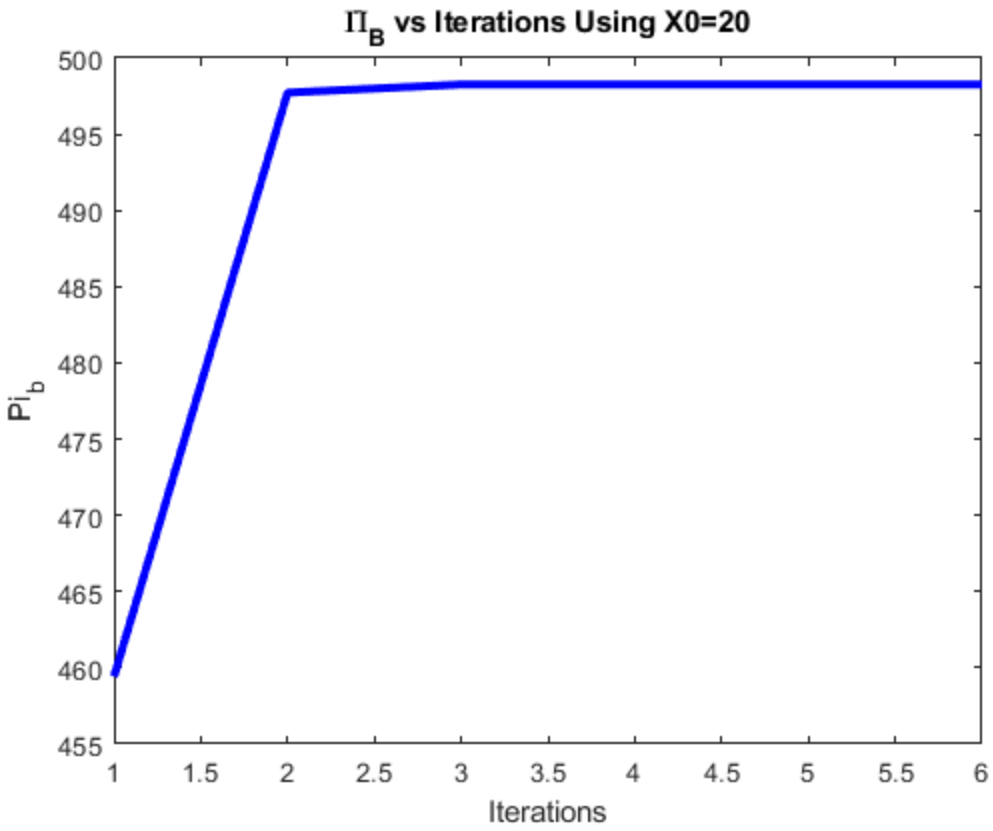



Π_B vs Iterations Using X0=0.2
0.3
0.25
0.2
0.15
0.1
0.05
0
Pi_b
1
1.5
2
2.5
3
3.5
4
Iterations


Π_A vs Iterations Using X0=2
4
3.5
3
2.5
2
1.5
1
0.5
0
Pi_a
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2
Iterations


Π_B vs Iterations Using X0=2
12.1
12.05
12
11.95
11.9
11.85
11.8
11.75
Pi_b
1
1.5
2
2.5
3
3.5
4
4.5
5
Iterations

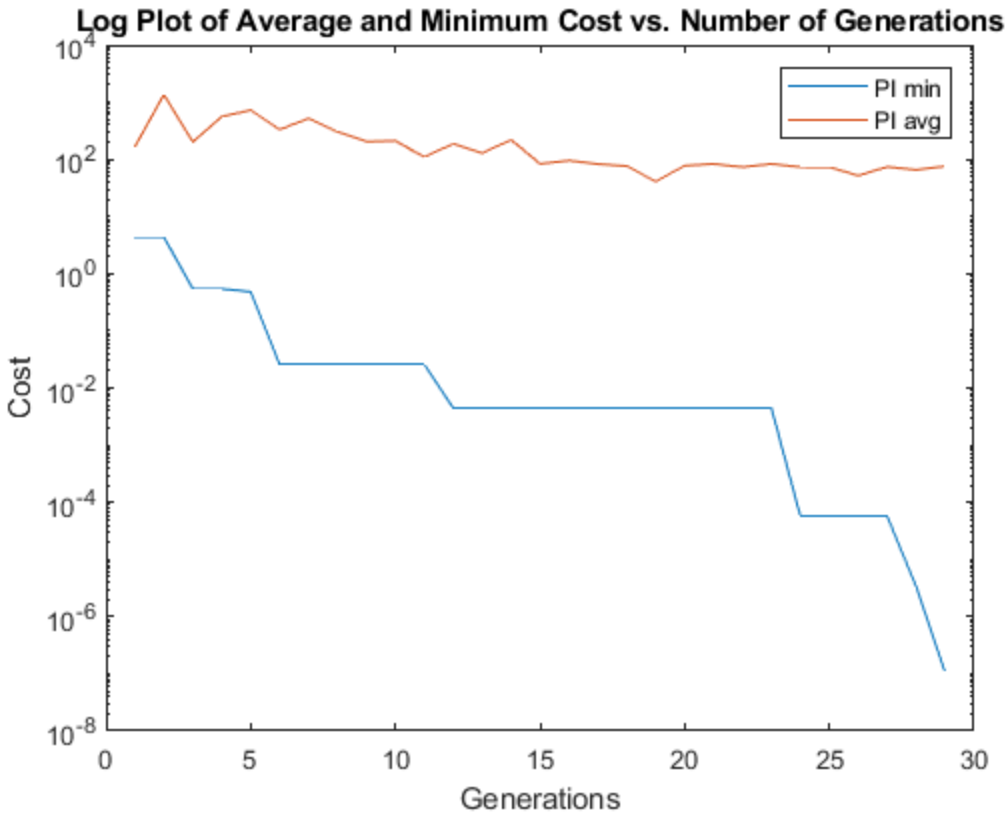

Π_A vs Iterations Using X0=20
400
350
300
250
200
150
100
50
0
Pi_a
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2
Iterations



Genetic Algorithm Implementation

```
% define variables

S = 50;
G = 100;
P = 12;
dv=1;
TOL_GA = 10^(-6);
PI = zeros(G,S);
Orig = zeros(G,S);
PI_min = [];
PI_avg = [];
new_pi = zeros(1,S);
g=1;
functio = Pi_b;
Lambda = -20+(20+20)*rand(S,1);

for s=1:S
    new_pi(1,s) =functio(Lambda(s,1));
end

[new_pi, ind] = sort(new_pi); %Sort new pi
PI(1,:) = new_pi;             %Store new pi
PI_min = [PI_min min(new_pi)];%Store cost of best performer
```

```
PI_avg = [PI_avg mean(new_pi)];%Store average cost of performers
Lambda = Lambda(ind);          %Store Lambda
Orig(1,:) = ind;               %Store indices of the first generation

while PI_min(end)>TOL_GA && g<=G %While the generation limit has not
 been exceeded and the cost is greater than TOL
    g=g+1;

    for u = 1:2:P %For Populating

        % Assign random numbers to Phi
        Phi_1 = randi([0 100]);
        Phi_2 = randi([-100 0]);

        %Generating New Children using Equation 5
        x_c1 = Phi_1*new_pi(u) + (1-Phi_1)*new_pi(u+1);
        x_c2 = Phi_2*new_pi(u) + (1-Phi_2)*new_pi(u+1);

        Lambda(P+u,dv) = x_c1;
        Lambda(P+u+1,dv) = x_c2;
        pi_c1 = functio(x_c1);
        pi_c2 = functio(x_c2);
        new_pi(dv,P+u) = pi_c1;
        new_pi(dv,P+u+1) = pi_c2;
    end
    for n=S-2*P:S
        Lambda = -20+(20+20)*rand(S,1);
        new_pi(1,n) = functio(Lambda(n,1));
    end
    [new_pi, ind] = sort(new_pi); %Sort new pi
    PI(1,:) = new_pi;             %Store new pi
    PI_min = [PI_min min(new_pi)];%Store cost of best performer
    PI_avg = [PI_avg mean(new_pi)];%Store average cost of performers
    Lambda = Lambda(ind);          %Store Lambda
    Orig(1,:) = ind;
end

figure("name","PI_min vs. PI_avg")
semilogy(1:g,PI_min, 1:g, PI_avg)
title('Log Plot of Average and Minimum Cost vs. Number of
 Generations')
xlabel('Generations')
ylabel('Cost')
legend('PI min','PI avg')
```



Published with MATLAB® R2020a