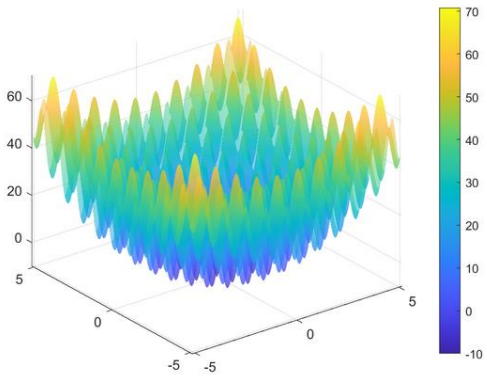
Banzhaf, Deb, Goodman Spring 2021

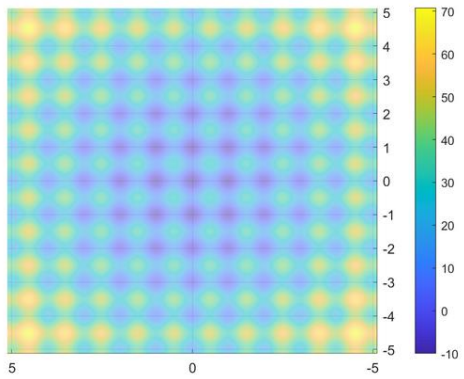
CSE848: Evolutionary Computation
Michigan State University
Assignment HA1: Home Assignment 1

Q1 – The minimum solution of the five-variable Rastrigin function solved using MATLAB's fminsearch(). **The main code and results are provided below:**

To better understand the search space of the multivariable objective function, we plot surfaces of the Rastrigin function in 2-dimensional and 3-dimensional for each α = 0.25, 1.0, and 2 .0. From the surface of a Rastrigin function with α = 2.00, we found that there are several local minima. This is multimodal optimization, where we try to find the optimum of a single objective function that has multiple local optima.

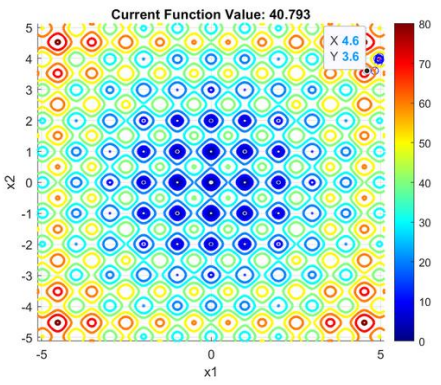


3D surface of a Rastrigin function when α = 2.00

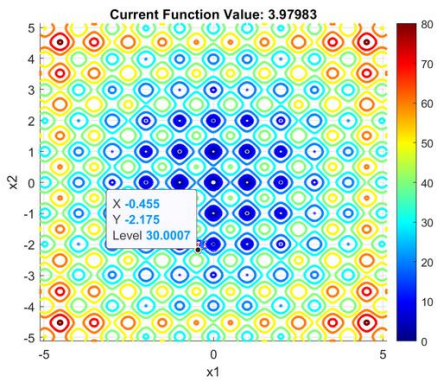


2D surface of a Rastrigin function when α = 2.00

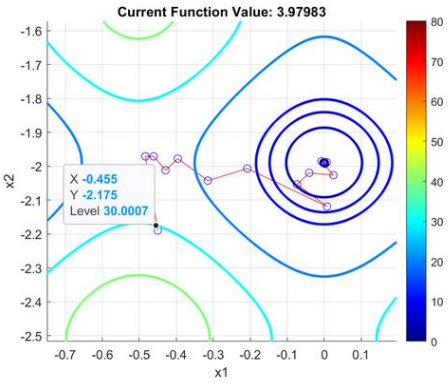
All code and high-resolution images available at https://github.com/JaturongKongmanee/evo-com



The history of intermediate solutions of α = 2 starting at x0=[4.6, 3.6]



The history of intermediate solutions of α = 2 starting at x0=[-0.455, -2.175]

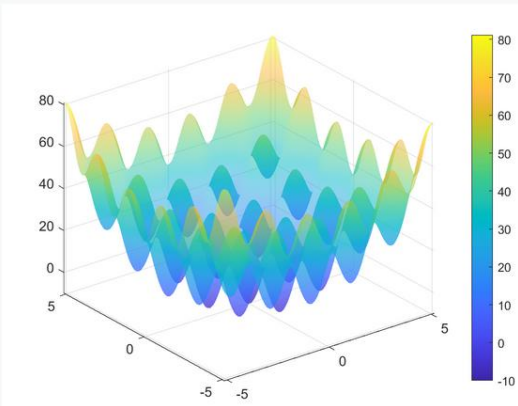


The history of intermediate solutions of α = 2 starting at x0=[-0.455, -2.175]

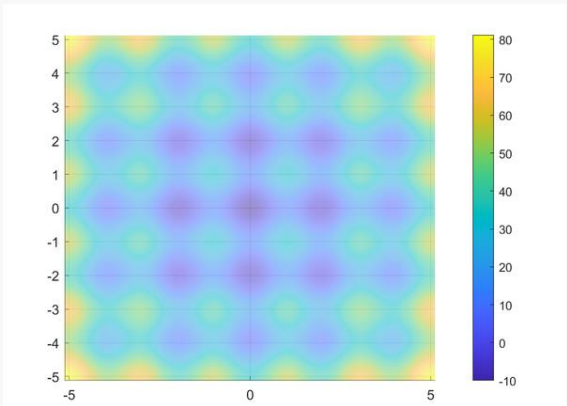
Before moving to n = 5 variables optimization, we select two initial points and plot their history of intermediate solutions to see how initial points affect their convergence. The initial two points are (4.6, 3.6) and (-0.455, -2.175), where the first one is at the edge of the region and the latter is close to a global minimum at point (0, 0). We can see the latter point's zoom plot stuck in the local minima leading to the suboptimal solution. Although it is initially set near a global minimum point.

All code and high-resolution images available at https://github.com/JaturongKongmanee/evo-com

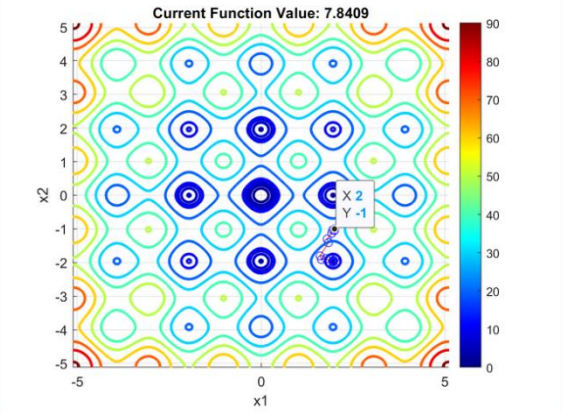
Now let us plot the search space of a given function with α = 1.00. We firstly visualize the surface, as shown in the plot below. We found that there are several local minima but less than α = 2.00. This time we select an initial point near a global minimum, hoping that it can find the way to reach a global point where the value of the function is 0. The plot below shows that algorithms still fail to find a global minimum because of its many local minima.



3D surface of a Rastrigin function when α = 1.00



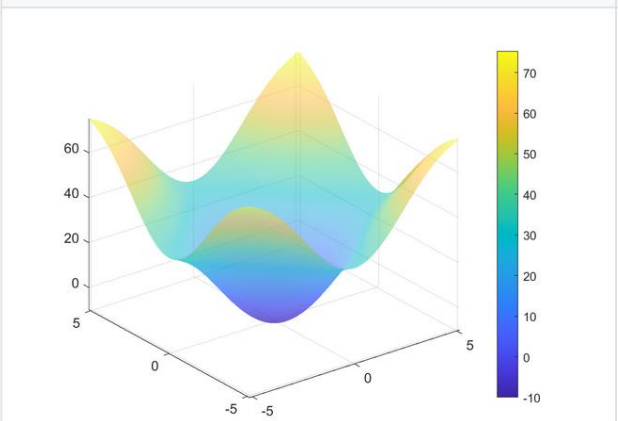
2D surface of a Rastrigin function when α = 1.00



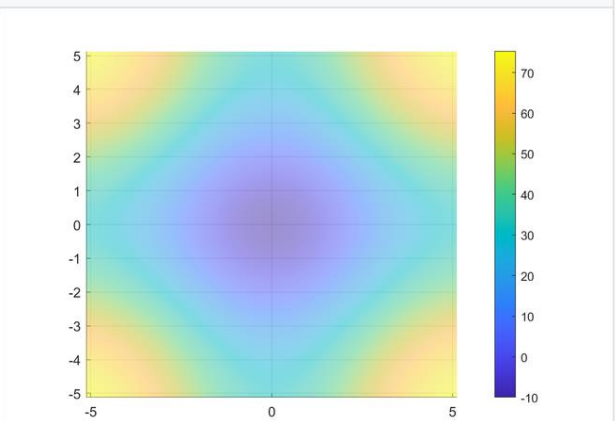
The history of intermediate solutions of α = 1.00 starting at x0=[2, -1]

All code and high-resolution images available at https://github.com/JaturongKongmanee/evo-com

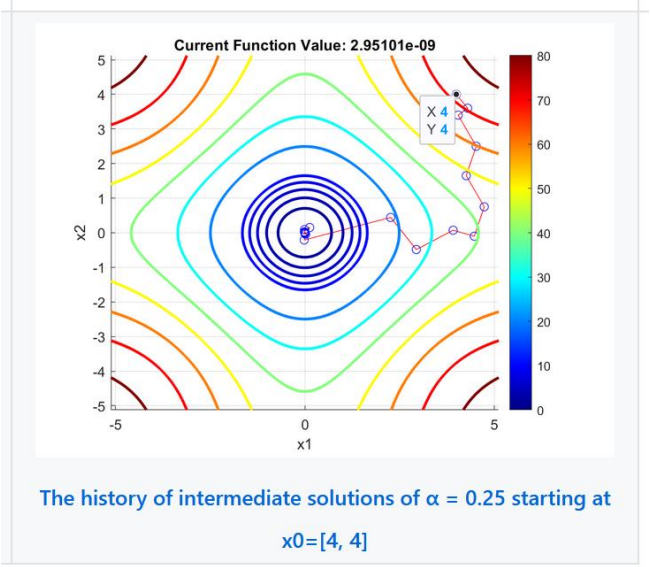
We now focus on the same function with α = 0.25. This function now has only one optimum point, as shown in the plot below. This property makes it easy for algorithms to find a minimum solution. We follow the same procedure as done previously. We arbitrarily select an initial point (4, 4) and plot the history of intermediate solutions. We found that it converges quickly with a small number of iterations.



3D surface of a Rastrigin function when α = 0.25



2D surface of a Rastrigin function when α = 0.25



The history of intermediate solutions of α = 0.25 starting at x0=[4, 4]

All code and high-resolution images available at https://github.com/JaturongKongmanee/evo-com

The section below shows the results of 100, 500, and 1000 runs for each α = 0.25, 1.00, 2.00. The best results for each α are provided in the tables below.

- The best result of `100` runs for each `α = (0.25, 1.00, 2.00)`

α	Best Solution	Objective Func Val (at the solution)	No. of Runs	No. of Iterations
0.25	[-2.097571028020289e-05;-6.514110354774446e-06;1.803443550319991e-05;9.684530414252273e-06;1.988684478441167e-05]	5.296996619108541e-09	100	264
1.00	[1.065192899121709e-05;-1.070792115707770e-06;-8.886268651530956e-06;1.960156693653340;-1.960247859930061]	7.8409	100	208
2.00	[-0.994895753536609;2.415873477669292e-05;2.906895671367380e-05;-1.877636814686927e-05;-2.984863450013099]	9.949561447944830	100	300

The results here suggest that, with α = 0.25, the algorithms easily converge to the global minimum. However, it is not the case with α = 1.00 and 2.00. We then increase the number of runs from 100 to 500. As shown in the table below, we see that the objective function values with α = 1.00 and 2.00 decrease rationally due to the higher number of variabilities of initial solutions.

- The best result of `500` runs for each `α = (0.25, 1.00, 2.00)`

α	Best Solution	Objective Func Val (at the solution)	No. of Runs	No. of Iterations
0.25	[-4.586718898415657e-06;9.627403619522660e-06;-7.658245703302216e-07;2.912130929769804e-05;8.577821374631565e-06]	4.231040406921238e-09	500	281
1.00	[-2.091614528800770e-05;2.570894475885090e-05;2.596739941619614e-05;1.981245334745658e-05;-1.960177785976345]	3.920451570920001	500	252
2.00	[-0.995000750245414;-0.994975122619967;-0.994961824003926;-4.314852432230637e-05;-0.994990369191254]	3.979837204966565	500	182

Having the higher number of variabilities of initial solutions tends to give a better solution. We then increase the number of runs to 1000. We found the objective function value is slightly decreased with higher number of iterations for α = 2.00. It implies that the intermediate solutions are at a slightly better point. However, this is not the case for α = 1.00 with 1000 runs. This probably is the case where the algorithms fall into the worse local minima.

All code and high-resolution images available at https://github.com/JaturongKongmanee/evo-com

α	Best Solution	Objective Func Val (at the solution)	No. of Runs	No. of Iteratio
0.25	[6.770030277358018e-06;-1.003060955703033e-05;-4.930712999280834e-06;1.339601204699992e-07;-8.442152185983215e-06]	9.885710028356698e-10	1000	273
1.00	[-1.824574994612021e-05;-1.111014499543094e-05;4.371511388608755e-05;1.960146894474212;1.683800537591302e-05]	3.920451633627039	1000	229
2.00	[0.994972856232765;0.994950409354347;0.994963964297747;6.099518411731260e-05;0.994950458030700]	3.979837038880760	1000	239

The main code to generate the results presented above is provided here.

```
%HA1Q1
%rng default %ensure to get the same results from random
%rand % returns the same value as at startup
%====|====|====|experiment setting|====|====|====|
min_number = -5.12;
max_number = 5.12;
alpha_list = [0.25, 1.0, 2.0];
n_run = 100;
n_var = 5;
%====|====|====|experiment setting|====|====|====|
r = (max_number - min_number).*rand(100, 1) + min_number;
r_range = [min(r), max(r)]

global best_solutions;
best_solutions = struct();

options = optimset('MaxIter', 2000, 'MaxFunEvals', 5000, 'Display','iter',
'PlotFcns',@optimplotfval);
rastrigin_func = @(x, alpha)10*length(x) + sum(x .^2 - 10 * cos(alpha * pi * x));

for alpha = alpha_list
    min_fval = Inf;
    best_solution = 0;
    n_iterations = 0;
    run_th = 0;
    fun = @(x)rastrigin_func(x, alpha);
    for c = 1:n_run
        x0 = randsample(r,n_var, true);
        [x,fval,exitflag,output] = fminsearch(fun,x0,options);
        if (fval < min_fval)
            min_fval = fval;
            best_solution = x;
            n_iterations = output.iterations;
            run_th = c;
        end
    end
    logData(run_th, alpha, min_fval, best_solution, n_iterations);
end

best_solutions

function logData(run_th, alpha, min_fval, best_solution, n_iterations)
    global best_solutions;
    field = "a" + num2str(alpha*100);
    best_solutions.(field) = struct('run_th', run_th, 'alpha', alpha, 'min_fval',
min_fval, 'solutions', best_solution, 'n_iterations', n_iterations);
end
```

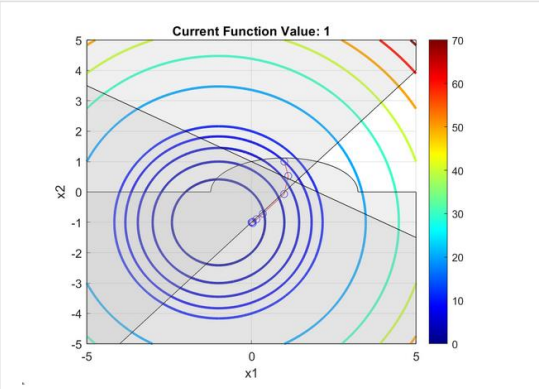
All code and high-resolution images available at https://github.com/JaturongKongmanee/evo-com

Q2 – The minimum solutions of the constrained minimization problem solved using MATLAB's fmincon(). **The main code, plots, and table showing minimum solutions of different initial points are presented below:**

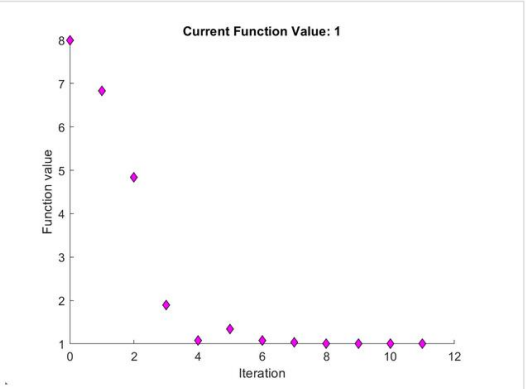
Initial Solution	Best Solution	Objective Func Val (at the solution)	No. of Iterations
[1, 1]	[0.0000, -1.0000]	1.0000	11
[0, 0]	[0.0000, -1.0000]	1.0000	11
[4, 2]	[0.0000, -1.0000]	1.0000	13
[-1, 4]	[0.0000, -1.0000]	1.0000	12
[-1, -4]	[0.0000, -1.0000]	1.0000	9

This table provides the minimum solutions of different initial points. To see how algorithms handle the multivariable constrained optimization problem and how they converge to the minimum point, we select different initial points based on the number of constraint functions of the given problem they satisfy. For example, as shown in a region below, an initial point (1, 1) satisfies only two constraint functions initially but moves toward the feasible region after two iterations, as shown in the output terminal below. We see that these initial points have the same minimum solution (0.0000, -1.0000) with the objective function value of 1.0000 and slightly different numbers of iterations. Although fminsearch()'s algorithms is the local search algorithm, but there is a global optima, as shown in the contour plot of an objective function below. This implicitly tells that any initial points in this region can converge to the global minima.

x0 = [1, 1]



The history of intermediate solutions



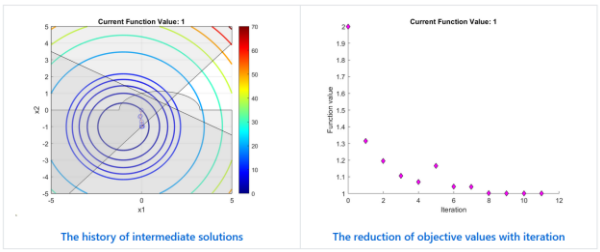
The reduction of objective values with iteration

All code and high-resolution images available at https://github.com/JaturongKongmanee/evo-com

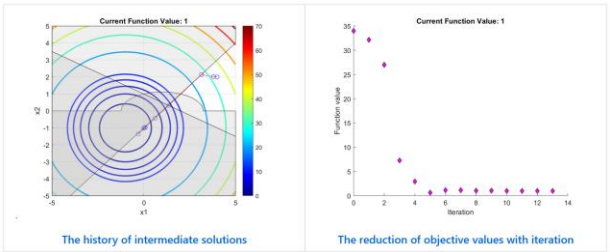
```
                                                First-order      Norm of
Iter F-count            f(x)  Feasibility   optimality         step
   0       3    8.000000e+00    1.000e+00    5.405e+00
   1       6    6.825179e+00    4.088e-01    4.461e+00    4.869e-01
   2       9    4.839429e+00    0.000e+00    3.266e+00    6.103e-01
   3      13    1.896039e+00    0.000e+00    1.978e+00    9.126e-01
   4      17    1.080451e+00    1.422e-01    5.984e-01    4.393e-01
   5      20    1.338408e+00    0.000e+00    2.650e-01    1.720e-01
   6      23    1.077172e+00    0.000e+00    7.156e-02    1.397e-01
   7      26    1.040264e+00    0.000e+00    2.082e-02    2.333e-02
   8      29    1.001424e+00    0.000e+00    8.860e-03    2.048e-02
   9      32    1.000401e+00    0.000e+00    2.000e-04    6.078e-04
  10      35    1.000004e+00    0.000e+00    5.579e-05    2.120e-04
  11      38    1.000004e+00    0.000e+00    2.000e-06    7.068e-08

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.
```

For all initial points in a table, the plots of the history of intermediate solutions, and the reduction of objective function values with iterations are provided below. The contour of an objective function and the lines with their shadow of the feasible region of constraint functions are drawn.

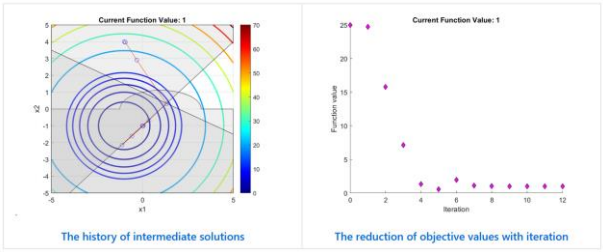
x0 = [0, 0]



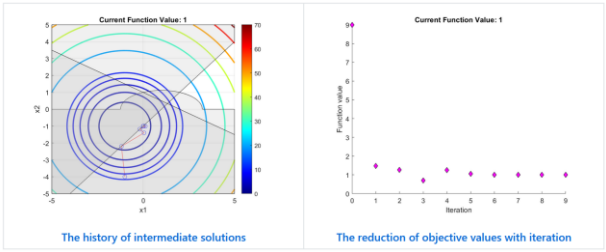
x0 = [4, 2]



x0 = [-1, 4]



x0 = [-1, -4]

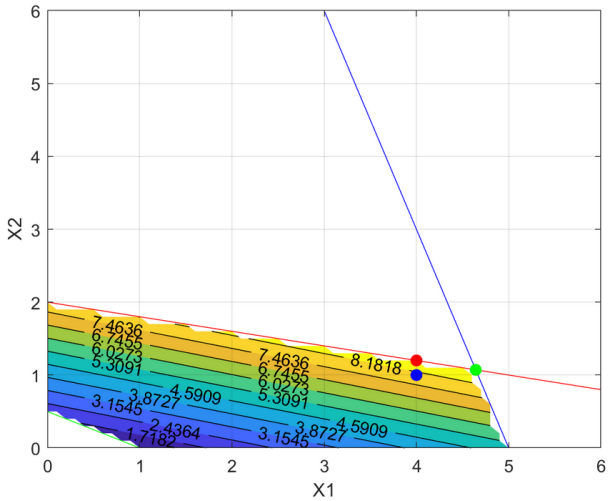


All code and high-resolution images available at https://github.com/JaturongKongmanee/evo-com

```
% constraint.m
% three constraint functions
function [c, ceq] = constraint(x)
    c(1) = (x(1)-1)^2+4*x(2)^2-5;
    c(2) = -1*x(1)+x(2)+1;
    c(3) = x(1)+2*x(2)-2;
    ceq = [];
```

```
% myproblem_fmincon.m
function [x, fval, history] = myproblem_fmincon(x0)
    history = x0;
    options = optimset('OutputFcn', @myoutput, 'Display', 'iter', 'PlotFcns',
@optimplotfval);
    objfunc = @(x)(x(1)+1)^2+(x(2)+1)^2;
    lb = [-5, -5];
    ub = [5, 5];
    [x,fval,exitflag,output] = fmincon(objfunc, x0, [], [], [], [], lb, ub, @constraint,
options);
    x, fval, exitflag, output

    function stop = myoutput(x, optimvalues, state)
        stop = false;
        if isequal(state, 'iter')
            history = [history; x];
        end
    end
end
```

Q3 – The optimal solutions solved using MATLAB's linprog() **and** intlinprog(). **The main code and a plot are presented below:**



The optimal solutions on ax1-x2 plot of feasible region

All code and high-resolution images available at https://github.com/JaturongKongmanee/evo-com

Best Solution	Objective Func Val (at the solution)	MATLAB's function used	Point Color (graph above)	*Integer or Real-valued
x = [4.6429, 1.0714]	fval = 8.9286	`linprog()`	Green	Both are real-valued
x = [4.0000, 1.2000]	fval = 8.8000	`intlinprog()`	Red	intcon = [1]
x = [4.0000, 1.0000]	fval = 8.0000	`intlinprog()`	Blue	intcon = [1, 2]

The solutions are verified with the 2D plane above. Drawing three constraints on a 2D plane, where the constraints form the feasible region (a set of points satisfying the constraints) in gradient color of a convex polygon with objective function values. We see that there are many solutions satisfying the constraints. Among all these feasible solutions, the optimal solutions must be at the intersections of constraints, where some intersections that are outside the feasible region need not to be considered.

We obtain an optimal solution from linprog(), where it is found at the intersection point of constraints `(4.6429, 1.0714) with fval=8.9286`, represented by the green dot. If the optimizer increases the point any further, there would be no part of the intersection line that would remain the feasible region . Now let us add two restrictions on the input variable types to see how the results change. First, we only allow the first variable to take only an integer value. Second, we force two variables to take only integer values. We then solve them using intlinprog(). The red dot `(4.0000, 1.2000) with faval=8.8000` lies on the boundary and is a suboptimal solution obtained from the first restriction. We see that only x1 has an integer value. The blue dot `(4.0000, 1.0000) with fval=8.000` lies on the feasible region and is a basic feasible solution of the second restriction, where both variables are integers. We found that if the intersection points of constraints do not lie on the integer coordinate, linprog() always provides a better solution than intlinprog() because of the higher precision of the number represented.

```
% HA1Q3
c = [1, 4];
A = [1, 5; 3, 1; -1, -2];
b = [10; 15; -1];

[x,fval,exitflag,output,lambda] = linprog(-c, A, b)

% HA1Q3
c = [1, 4];
A = [1, 5; 3, 1; -1, -2];
b = [10; 15; -1];

% declare x1 as integers, but x2 as real-valued
intcon = [1];
[x,fval,exitflag,output] = intlinprog(-c, intcon, A, b)

% declare both x1 and x2 as integers
intcon = [1, 2];
[x,fval,exitflag,output] = intlinprog(-c, intcon, A, b)
```

All code and high-resolution images available at https://github.com/JaturongKongmanee/evo-com