

%copyright Charles Davi 2022

```
function [final_problem_domain weight_matrix] =  
Mem_Interf_Rand_Optimization_PrePopMatrix_GB(problem_domain, num_iterations,  
search_depth, weight_matrix)
```

```
best_goal = Inf; %minimum difference from goal initial value  
best_quantity = 0;
```

```
num_dimensions = size(problem_domain,1);  
num_rows = size(problem_domain,2);
```

```
counter = 1;
```

```
%iterates over number of interpolation attempts  
for k = 1 : num_iterations
```

```
    %iterates over depth  
    for i = 1 : search_depth
```

```
        %iterates over all dimensions  
        for D = 1 : num_dimensions
```

```
            %if true, then this is the first column, random guess  
            if(i == 1)
```

```
                current_pos(D) = randi(num_rows);
```

```
            %if true, we're at the top boundary  
            elseif(current_pos(D) == 1)
```

```
                weight_vector = weight_matrix(1:2,i,D);  
                [IGNORE next_pos(D)] = min(weight_vector);  
                next_pos(D) = next_pos(D) + 1; %we either go straight or down
```

```
            %if true, we're at the bottom boundary  
            elseif(current_pos(D) == num_rows)
```

```
                weight_vector = weight_matrix(num_rows-1:num_rows,i,D);  
                [IGNORE next_pos(D)] = min(weight_vector); %either straight or down
```

```
            %otherwise, it's not the first column or either boundary  
            else
```

```
                weight_vector = weight_matrix(current_pos(D) - 1 : current_pos(D) + 1, i, D);  
                [IGNORE next_pos(D)] = min(weight_vector);
```

```

endif %end of special case if test

prior_pos(D) = current_pos(D);

%updates current position-----

%if true, we calculate the next position
if(i != 1)

    %if true, move up
    if(next_pos(D) == 1)

        current_pos(D) = current_pos(D) - 1;

    %if true, move straight
    elseif(next_pos(D) == 2)

        %no change

    %if true, move down
    else

        current_pos(D) = current_pos(D) + 1;

    endif

endif

endfor %end of D-loop

%evaluates polynomial and tests difference from goal
[approx_goal approx_quantity] = eval_goal_function(problem_domain, prior_pos);

%updates the weights for each dimension
for D = 1 : num_dimensions

    weight_matrix(prior_pos(D),i,D) = approx_goal;

endfor

%if true, this is the best answer
if(approx_goal <= best_goal && approx_quantity > best_quantity)

    best_quantity = approx_quantity
    best_goal = approx_goal
    final_problem_domain(counter,:) = prior_pos;
    counter = counter + 1;

```

endif

endfor %end of i-loop

endfor %end of outer k-loop

endfunction