# Optimizing Wind Farm Layout for Maximum Energy Production Using Direct Methods

Jacob R. West

*Stanford University, Department of Mechanical Engineering*

**Pattern search and adaptive simulated annealing are applied to the problem of wind farm optimization. The optimization objective is to maximize the energy production of a wind farm with a set number of turbines, given constraints on land area and inter-turbine spacing. An inverse relationship between wind farm density and farm efficiency is shown. An inverse relationship is also shown between the standard deviation of the wind direction distribution and the farm efficiency.**

## Nomenclature

| | |
|---|---|
| $x$ | streamwise position of turbines |
| $y$ | lateral position of turbines |
| $D$ | turbine diameter |
| $d$ | distance between turbine centers |
| $u$ | wind speed |
| $p$ | probability of wind being from a given direction |
| $L$ | size of domain |
| $N$ | number of wind turbines |
| $M$ | number of wind direction bins |
| $\beta$ | constraint violation penalty |
| $\sigma$ | standard deviation of wind direction distribution |
| $\eta$ | wind farm efficiency |

*Subscript*

| | |
|---|---|
| $i, j$ | turbine index |
| $k$ | wind direction index |
| $\infty$ | freestream wind |

## I. Introduction

Wind turbines in a wind farm typically perform worse than an isolated turbine would under the freestream wind conditions. Turbines influence each other through the wakes they create, which can reduce energy available to trailing turbines, as well as lead to more unsteady structural loads. As a result, if a wind farm is designed in a naive way, there may be significant degradation of power generation capacity and turbine life. Reduced power generation of 5%-40% has been reported depending on the wind conditions and turbine separation studied.[1]

Wind farm layouts can be optimized by predicting the wake interactions of a candidate layout. I focus on maximizing the power output from a set number of turbines in a defined land area. I use the results of the optimization to study how wind farm efficiency and optimal arrangement are affected by 1) wind farm density (turbines per land area) and 2) wind direction distribution. For a very sparse wind farm, it is trivial to position the turbines in a way that their wakes do not interfere with each other. But as the turbine density increases or the wind distribution broadens, maximizing the farm's power output may require wake interference. I will demonstrate the extent to which this is true. Section II details the formal optimization

problem, constraints, and optimization algorithm. Section III shows the results of the optimization and the trends in wind farm efficiency.

## II. Optimization Approach

The optimization problem is formally stated as

$$\max_{\mathbf{x},\mathbf{y}} \quad \eta \tag{1}$$

*subject to*

$$0 \leq x_i \leq L_x \tag{2}$$

$$0 \leq y_i \leq L_y \tag{3}$$

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq 2D, \quad \forall \quad i \neq j \tag{4}$$

It may appear that there are only 2 design variables, x and y, but both x and y are vectors with N elements, corresponding to an x and y position for each turbine. So for a a wind farm with 50 turbines, there are 100 design variables!

### A. Cost Function

Instead of optimizing annual energy production (AEP) directly, it is practical to optimize an efficiency metric, $\eta$, which represents the percentage of freestream wind power capture by the farm. $\eta$ is proportional to AEP and facilitates comparison between wind farms of different size. The definition of AEP given by Herbert-Acero et al.[2] is

$$AEP = 8766 \sum_{i=1}^{N} \int_{0}^{360} \int_{0}^{U_{max}} \frac{1}{2} \rho A C_p(U, \rho) p_{m,H}(U, \theta) dU d\theta \tag{5}$$

which represents a sum over the number of hours in a year and N turbines, each sum containing integrals over all wind angles and power-speed curves for the turbine design(s) used. Because I am considering a farm at steady state, with only one type of turbine, and a discrete probability distribution of wind directions, the integrals become to sums. Because I am not concerned with an accurate prediction of energy production, but only with comparing the efficiency of different layouts, the factors of air density ($\rho$), area (A), and power coefficient ($C_p$) are irrelevant. Finally, because the power in the wind is proportional to the cube of velocity, I can simplify to $\eta$, which by construction, lies between 0 and 1.

$$AEP \quad \sim \quad \eta = \frac{1}{N} \sum_{k=1}^{M} p_k \sum_{i=1}^{N} \left(\frac{u_i}{U_\infty}\right)^3 \tag{6}$$

The most accurate way to evaluate the energy production of a proposed wind farm design would be to run a fully-resolved direct numerical simulation of the Navier-Stokes equations in order to resolve all scales of the turbulent flow in a wind farm. However, even a large eddy simulation of such a flow is too computationally intensive to be a practical optimization tool. As a result, many wake models have been developed that approximate how turbines influence the flow field around therm, thereby affecting other turbines. For my purposes, I use the simple 2D "Jensen Model", upon which many other more sophisticated models are based.[3] This model treats each wake as a linearly expanding region of velocity deficit which recovers according to conservation of momentum. The equation for the wind speed in the wake is given below.

$$u = U_\infty \left(1 - \frac{2}{3} \left(\frac{D/2}{D/2 + \alpha x}\right)^2\right) \tag{7}$$

$\alpha$ is a wake expansion coefficient, which is set to 0.1, per the recommendation of Jensen. This equation results in a "top-hat" profile, because the velocity drops sharply from the freestream value to the wake value. With this model, $u_i$ at each turbine location can be computed quickly through an iterative process, where each turbine affects the value of $U_\infty$ in equation 7.

Since I am considering the same wind farm under a variety of wind conditions specified by a discrete probability distribution, I compute $\eta$ for each wind direction, and then perform a weighted sum according to the probability distribution. A simple probability distribution can be parametrized by a mean wind direction and a standard deviation, $\sigma$. Without loss of generality, the mean can be aligned with the x-axis. For my purposes, I consider discrete distributions of wind angle with a Gaussian profile, in 6 bins, ranging from 0° to 180°. A high value of sigma corresponds to a site where the wind blows from all directions with equal likelihood, and a low value of sigma corresponds to a site with a strongly dominant wind direction.

### B. Constraints

The domain I have chosen for the wind farm is a $\pi$ x $\pi$ square, with turbines of $D = 0.1$. This is a common size and shape for a dimensionless domain studied by large eddy simulation investigations of very large wind farms.[4] Furthermore, I have restricted the turbines so that their centers must be at least 2 diameters apart. This gives them clearance to rotate in the wind, as well as extra clearance for safety.

It is always possible to improve the wind farm's performance by moving the turbines farther apart, so I have enforced the constraints using penalty functions as shown below.

$$\beta_1 = \sum_{j}^{N} \sum_{i \neq j}^{N} d_{ij} \quad \forall \quad d_{ij} > 2D \tag{8}$$

$$\beta_2 = \sum_{i}^{N} |x_i| + |y_i| \quad \forall \quad x_i, y_i \leq 0 \tag{9}$$

$$\beta_3 = \sum_{i}^{N} |x_i - L_x| + |y_i - L_y| \quad \forall \quad x_i \geq L_x, \quad y_i \geq L_y \tag{10}$$

$$\beta_{total} = (\rho\beta_1)^2 + (\rho(\beta_2 + \beta_3))^2 \tag{11}$$

$$\tag{12}$$

$\beta_{total}$ is subtracted from the value of $\eta$ computed during optimization, and the value of $\rho$ is multiplied by 1.1 whenever a more optimal point is found by the algorithm.

### C. Optimization Algorithm

The wind farm design optimization problem is known to have a high number of local optima.[2] This occurs because of the large number of design variables, but also because there are many design points that are equivalent or near-equivalent. For example, the design point $\mathbf{x} = [1, 2, 3], \mathbf{y} = [4, 5, 6]$ is exactly equivalent to $\mathbf{x} = [3, 1, 2], \mathbf{y} = [6, 4, 5]$. The layouts are the same, but the ordering is shuffled. In sparse wind farms, there are many global optima, where it is possible to position all turbines in the freestream wind. In denser wind farms, this possibility becomes less likely, and knowing with certainty whether the optima is global or local becomes more difficult. Population methods are widely used in the wind farm optimization literature[2] to deal with the problem of local optima, but care must be taken when the "individuals" in the population interact. For example, with a genetic algorithm, the crossover operation becomes murky when all the design variables represent points in a single x-y plane.

Direct methods are a simple way to approach the high number of design variables in this problem, and can be implemented in a straightforward way. I found both Hooke-Jeeves (pattern search) and Adaptive Simulated Annealing to be effective for this problem, albeit with different strengths.[5] For both algorithms, multiple random starts were needed to explore local optima and find the best one. In practice, for this problem, the pattern search approach amounts to moving each turbine in x, and then y, until an improvement to the total performance is found. When an improvement is found, that candidate point is accepted as the new design point, and algorithm continues through the coordinate directions, looking for improved points. If the algorithm proceeds through an entire cycle of 4N moves without finding any improvement, then the step size is halved and the process continues. The algorithm is stopped when the step size is less than 0.001. For the non-dimensional domain typically studied in the literature, and assuming a turbine diameter of 100m, this corresponds to a location adjustment of 1m. Any finer would be an impractical and unachievable design

tolerance for the construction of a wind farm. The initial step size is set to 0.512, which makes the ending step size 0.001, but is also large enough for the turbines to move around significantly early on.

Some limitations of the pattern search approach are that only certain step sizes are allowed, turbines cannot be moved in x and y simultaneously, and 2 turbines cannot be moved at once. All of these slow it down and limit the search space. My implementation of adaptive simulated annealing addresses the first 2 issues by using random steps, and by pairing the x and y design variables for each turbine. It adjusts the step size in x and y for each turbine based on how frequently the random steps are accepted according to the Metropolis criterion. The idea behind pairing is to speed up adaptive simulated annealing by reducing the number of random steps per cycle by half, the insight being that the x and y positions of a turbine should not really be treated as independent variables since they describe the same turbine. In my implementation, 4 random moves are performed for each turbine before adjusting the step size and reducing the temperature. The temperature is initialized to $T = 1$ and decayed by a factor of 0.9. The algorithm is stopped when $T < 10^{-6}$.

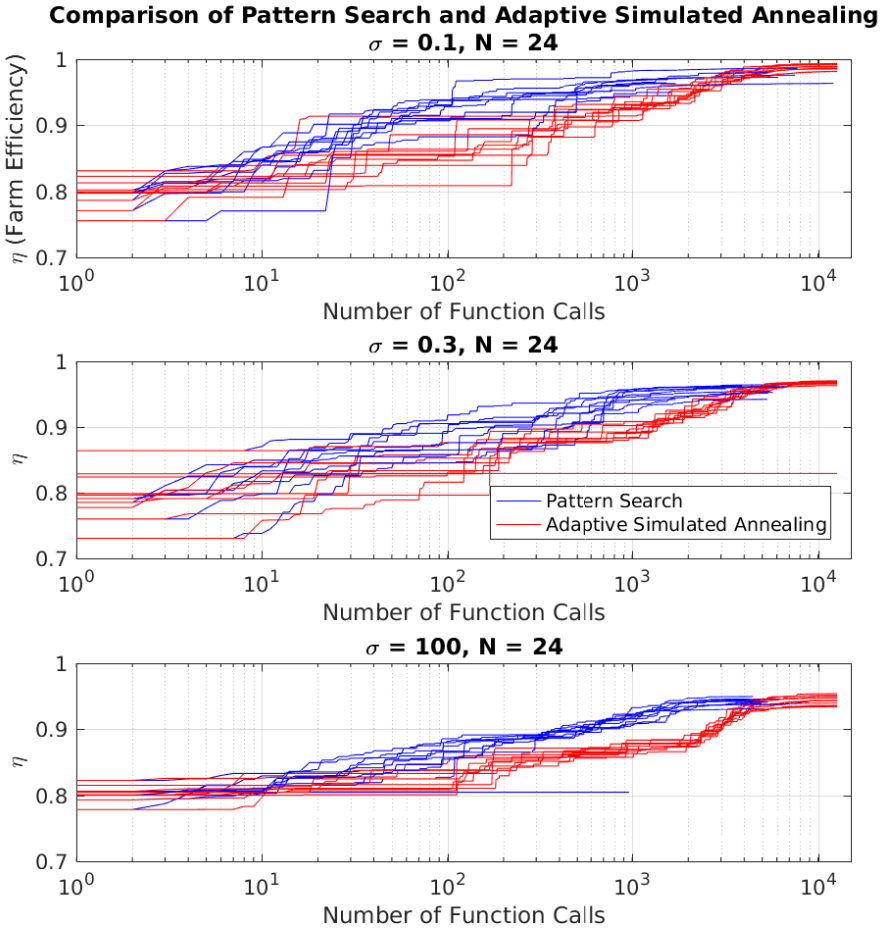


**Figure 1. Optimized wind farm efficiency as a function of number of cost function calls, for both pattern search and adaptive simulated annealing, plotted for N=24 and three values of $\sigma$.**

Figure 1 shows the performance of pattern search and adaptive simulated annealing as a function of number of cost function evaluations. 10 random starting conditions were used to generate each plot. In general, adaptive simulated annealing is able to find slightly better optima than pattern search, but also tends to converge more slowly. For this reason, I used pattern search to study the effects of wind farm density and wind direction distribution.

# III. Results and Discussion

## A. Optimized Layouts

Figure 2 shows 4 optimized layouts using a simple offset grid arrangement (often used in practice) as a starting point. For a single wind direction, the baseline layout has $\eta = 0.840$, and for a uniform distribution, it has $\eta = 0.716$. The efficiency for a single wind direction ($\sigma = 0.01$) is reasonably good, but the turbines in the back of the wind farm are stuck in slower wind, which is sub-optimal. For a single wind direction, pattern search improves on the layout while preserving a grid-like structure, due to the way in cycles through the coordinates in an ordered fashion. The optimal layout from adaptive simulated annealing is less structured but better able to position turbines in the freestream wind. When multiple wind directions are considered, the optimization becomes more difficult, and the optimized layouts are less likely to contain easily identifiable alignments. Pattern Search and adaptive simulated annealing achieve quantitatively and qualitatively similar results in this case.

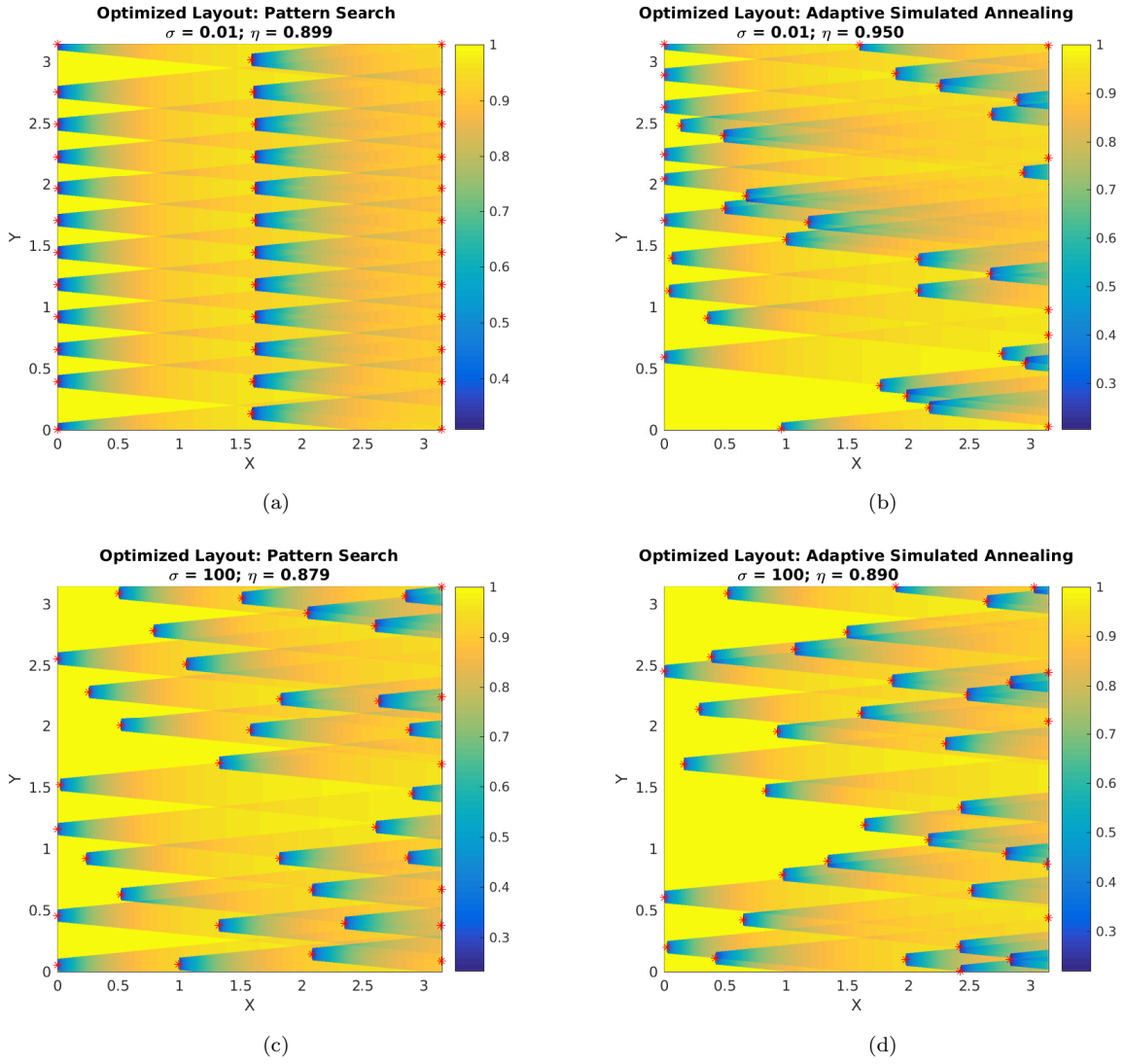


**Figure 2. Streamwise velocity profiles in the mean wind direction for optimized Layouts for a 36 turbine farm, starting from a simple grid layout. Left column is optimized with Pattern Search, right column with Adaptive Simulated Annealing. Top row is optimized for a single wind direction. Bottom row is optimized for a uniform distribution of wind directions**

### B. Effect of Farm Density and Wind Distribution

As the number of wind turbines in a wind farm increases, it becomes increasingly difficult to arrange them in an optimal way, especially as wind distributions are taken into account. Figure 3 shows how increasing the number of turbines in the same area affects $\eta$ for several values of $\sigma$. The optimal efficiency decreases as N is increased, and as $\sigma$ is increased. For lower values of $\sigma$, globally optimal ($\eta = 1$) arrangements can be found for higher values of N.

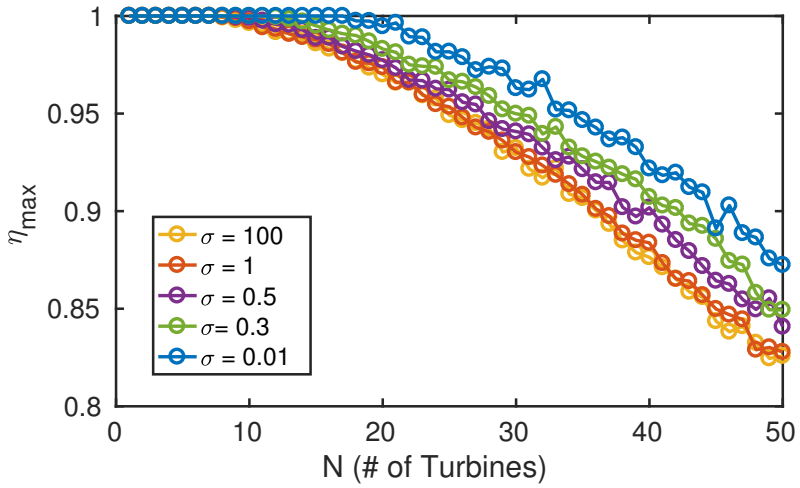


**Figure 3. Optimized wind farm efficiency as a function of the number of turbines in the farm, for several values of standard deviation of the wind distribution. For this purpose, the wind distribution was binned into 6 directions from 0 to 180°.**

It should be noted that $\eta$ should be a monotonically decreasing function of N. This implies that the optima from Figure 3 may not all be global, but the trends with respect to $\sigma$ and N are evident nonetheless.

## IV. Conclusion

The relationships found between farm density and efficiency, and wind spread and efficiency are expected. They highlight the practical difficulties associated with wind energy in the real world, where wind does not always blow from one direction, and denser farms are more practical from a capital investment perspective. A key result of this work is that a simple, direct method approach can be used to design better arrangements than simple grids that can be practically implemented. Furthermore, the optimization with multiple wind directions can be generalized to situations where there is some uncertainty about the wind direction, in order to design a layout that is more robust to changes in the wind distribution. In the future, my goal is expand this work to consider more realistic wind farm physics during the optimization, and to validate the optimized layouts with large eddy simulation.

## References

[1]Sanderse, B., "Aerodynamics of wind turbine wakes - Literature review", Technical Report ECN-E09-016, 2009.

[2]Herbert-Acero et al., "A Review of Methodological Approaches for the Design and Optimization of Wind Farms", Energies, Vol. 7, No. 11, 2014, pp. 6930-7016.

[3]Jensen, N. O., *A note on wind generator interaction*, Ris-M, No. 2411, 1983.

[4]Calaf, M., Meveveau, C., and Meyers, J., "Large eddy simulation study of fully developed wind-turbine array boundary layers", Physics of Fluids, Vol. 22, 015110, 2010.

[5]Kochenderfer, M. and Wheeler, T., *Algorithms for Optimization*, "Direct Methods".

# Appendix: MATLAB Code

## A. Pattern Search Algorithm

```
function res = pattern_search(xvec,step_break,wake_model,cost_function,
    wind_rose,Lx,Ly)
%%% takes in an initial layout, step size break condition, and wake model
%%% returns optimized layout and cost
    step0 = 0.512;
    cost0 = cost_function(xvec,wind_rose,wake_model);
    step = step0;
    bestcost = cost0;
    iter = 1;
    bestcost_vec(iter) = bestcost;
    N = length(xvec)/2; %number of turbines
    rho1 = 10; %penalty weight 1
    rho2 = 10; %penalty weight 2

    while step>=step_break
        dir= zeros(size(xvec));
        dir(1) = 1;
        last_best = bestcost;
            for ind = 1:N*2 %iterate over each direction
                search_point = xvec + step*dir;
                cost = cost_function(search_point,wind_rose,wake_model);
                iter = iter+1;
                bestcost_vec(iter) = bestcost;
                penalty = constraints(search_point,rho1,rho2);
                cost = cost+penalty;
                if cost<bestcost
                    %move to new point, search in next direction
                    xvec = search_point;
                    bestcost = cost;
                    rho1 = 1.1*rho1;
                    rho2 = 1.1*rho2;
                else
                    %try opposite direction
                    dir(ind) = -1;
                    search_point = xvec + step*dir;
                    cost = cost_function(search_point,wind_rose,wake_model
                        );
                    penalty = constraints(search_point,rho1,rho2);
                    cost = cost+penalty;
                    iter = iter+1;
                    bestcost_vec(iter) = bestcost;
                    if cost<bestcost
                        xvec = search_point; %reassign xvec
                        bestcost = cost;
                        rho1 = 1.1*rho1;
                        rho2 = 1.1*rho2;
                    end

                end
                %change to next coorindate
                dir(ind) = 0;
                dir(ind+1) = 1;
```

```
            end
        if last_best == bestcost
            step = step/2;
        end
    end

    res = [bestcost; xvec; bestcost_vec'; iter];

end
```

### B. Adaptive Simulated Annealing Algorithm

```
function res = simu_anneal(xvec,temp_break,wake_model,cost_function,
    wind_rose,Lx,Ly)
%%% takes in an initial layout, temperature break condition, and wake
    model
%%% returns optimized layout and cost

    %% Initialize algorithm
    T0 = 1; %initial temperature
    gamma = 0.9; %decay factor
    rho1 = 10; %constraint penalty multiplier
    rho2 = 10; %constraint penalty multiplier
    cost0 = cost_function(xvec,wind_rose,wake_model);
    cost = cost0;
    xbest = xvec;
    bestcost = cost0;
    iter = 1;
    bestcost_vec(iter) = bestcost;
    N = length(xvec)/2;
    T=T0;
    Ns = 4;
    Nt = 1;
    C = 2;
    xvec= [xvec(1:N),xvec(N+1:end)];
    v = Lx/10*ones(N,1); %initial max step size
    a = zeros(size(v));

        %% Optimization Loop
    while T>temp_break
        T = T*gamma;
        for t_ind = 1:Nt
            for s_ind = 1:Ns
                for ind = 1:N
                    dir = zeros(N,1);
                    dir(ind,:) = 1;
                    %choose a next point to evaluate
                    x_prime = xvec + v.*dir.*(2*rand(1,2)-1);
                    cost_prime = cost_function([x_prime(:,1);x_prime(:,2)
                        ],wind_rose,wake_model);
                    iter = iter+1;
                    bestcost_vec(iter) = bestcost;
                    penalty = constraints([x_prime(:,1);x_prime(:,2)],rho1
                        ,rho2);
                    cost_prime = cost_prime+penalty;
                    delta_cost = cost_prime-cost;
```

```
                    if delta_cost <= 0
                        % accept new point
                        xvec = x_prime;
                        cost = cost_prime;
                        rho1 = 1.1*rho1;
                        rho2 = 1.1*rho2;
                        a(ind) = a(ind) +1; %records number of accepted
                            points
                    else
                        metropolis = exp(-delta_cost/T);
                        p = min(metropolis,1);
                        if p>rand
                            %accept based on probability p
                            xvec = x_prime;
                            cost = cost_prime;
                            a(ind) = a(ind) +1; %records number of
                                accepted points
                        end
                    end

                    if cost<bestcost
                        xbest = [xvec(:,1);xvec(:,2)];
                        bestcost = cost;
                    end

                end
            end
            %adjust step sizes
            mask_high = a>0.6*Ns;
            mask_low = a<0.4*Ns;
            mask_mid = 1-(mask_high+mask_low);
            v = v.*((1+C*(a/Ns-0.6)/0.4).*mask_high + (1+C*(.4-a/Ns)/0.4)
                .^(-1).*mask_low + mask_mid);
            a = zeros(size(v));
        end
    end
    xvec = [xvec(:,1);xvec(:,2)];
    res = [bestcost; xvec; bestcost_vec'; iter];

end
```

### C. Wake Model

```
function total_power = jensen_wakes(xvec, wind, plot_flag)
    %% Jensen Wakes
    N = length(xvec)/2;
    r0 =  .05;
    alpha = .1;
    theta = atan2(wind(2),wind(1));
    u_inf = norm(wind);

    %first pack xvec into x and y
    x0 = (xvec(1:N))';
    y0 = (xvec(N+1:end))';
```

```
    %then rotate theta
    R = [cos(theta), -sin(theta);
         sin(theta),   cos(theta)];
    x_rot = R*[x0;y0];

    %and repack into xvec
    xvec = [(x_rot(1,:))';(x_rot(2,:))'];
    X = xvec(1:N);
    Y = xvec(N+1:end);
    V = u_inf;
    for turb_ind = 1:N
        X_prime = X-xvec(turb_ind);
        Y_prime = Y-xvec(turb_ind+N);
        mask = (X_prime>0) & (abs(Y_prime) < (alpha*(X_prime)+r0));
        V = V.*(1-2/3*mask*(r0^2)./(r0+alpha*X_prime).^2); %this is the
            wake velocity
    end
    vel = V;

%% plotting
    if plot_flag == 1
        grid = 1000;
        dx = pi/grid;

        xmin = 0;
        xmax = pi;
        ymin = 0;
        ymax = pi;
        xgrid_rot = linspace(xmin,xmax,grid);
        ygrid_rot = linspace(ymin,ymax,grid);

        [X,Y] = meshgrid(xgrid_rot,ygrid_rot); %use only for plotting

        V = u_inf;
       for turb_ind = 1:N
        X_prime = X-xvec(turb_ind);
        Y_prime = Y-xvec(turb_ind+N);
        mask = (X_prime>0) & (abs(Y_prime) < (alpha*(X_prime)+r0));
        V = V.*(1-2/3*mask*(r0^2)./(r0+alpha*X_prime).^2); %this is the
            wake velocity
       end

        h = pcolor(X,Y,V);
        set(h,'EdgeColor','none'), colorbar, hold on,
        plot(xvec(1:N),xvec(N+1:end),'r*')
        axis equal
        xlim([0,pi]), ylim([0,pi])
        clear h
        xlabel('X/H')
        ylabel('Y/H')
    end

    %calculate power
    total_power = sum(vel.^3);
```

```
end
```

### D. Functions Used

```
function res = wind_rose_design(sigma)
    bins = 6;
    a=0;
    b=pi;
    d_theta = (b-a)/(bins);
    mu = (a+b)/2;
    theta = (a:d_theta:b-d_theta)';
    p = 1/(sigma*sqrt(2*pi))*exp(-(theta-mu).^2/(2*sigma^2));
    int = d_theta*sum(p);
    p = d_theta*p/int; %correct p so it integrates to 1
    res = [p,sin(theta),cos(theta)];
end

function res = turbine_distances(xvec)
    N = length(xvec)/2;
    x = xvec(1:N);
    y = xvec(N+1:end);
    res = sqrt((x-x').^2+(y-y').^2);
end

function res = farm_cost(xvec,wind_rose,wake_model)
    [dir,~] = size(wind_rose);
    cost = 0;

    for ind = 1:dir
        p = wind_rose(ind,1);
        wind = [wind_rose(ind,2),wind_rose(ind,3)];
        temp = wake_model(xvec,wind,0);
        cost = cost-p*temp; %weight power from that direction with
           probability
    end

    res = cost;
end

function res = constraints(xvec,rho1,rho2)
    % constraint 1: distance between turbines
    D = 0.1;
    max_dist = 2*D;
    d = turbine_distances(xvec);
    mask = (d<max_dist)-eye(N);
    violation = mask.*d;
    penalty_1 = 0.5*sum(violation(:)); %sum of all the distance violations
        between turbines

    % constraint 2: within boundaries
    mask1 = xvec<0;
    mask2 = xvec>Lx;
    penalty_2 = sum(abs(mask1.*xvec) + abs(mask2.*(xvec-pi)));

    res= (rho1*penalty_1).^2+(rho2*penalty_2).^2;
end
```