Homework #6: Misc. and Advanced Concepts — ME 575

due 4/8/2015 before class via Learning Suite — **50 possible points**

Complete **2 of the following 3 tasks**.

6.1 Robust Optimization. Complete the following problem from Dr. Parkinson. We solved most of this problem by hand in class. For the homework I would like you to implement it on a computer in a more general way. By that I mean ignore the fact that this is a QP problem, but solve it as a general nonlinear problem so that you could reuse the approach (i.e., use fmincon and a method for estimating gradients).

For the following problem:

$$\begin{aligned} \text{min} \quad & f = x_1^2 + 2x_2^2 + 3x_3^2 \\ \text{s.t.} \quad & c_1 = 2x_1 + x_2 + 2x_3 \geq 6 \end{aligned}$$

where $\Delta x_1 = \Delta x_2 = \pm 0.1$, $\Delta x_3 = \pm 0.05$

(a) Find the nominal optimum.

(b) Find the worst-case, robust optimum.

(c) Instead of the worst case tolerances, assume the variables are normally distributed with $\sigma_{x_1} = \sigma_{x_2} = \pm 0.033$, $\sigma_{x_3} = \pm 0.0167$. Find the statistical robust optimum where we wish the the optimum to be feasible 99.865% of the time.

(d) Suppose we add another constraint: $g_2 = -5x_1 + x_2 + 3x_3 \leq -10$ Find the statistical robust optimum with this constraint added. What is the overall estimated feasibility?

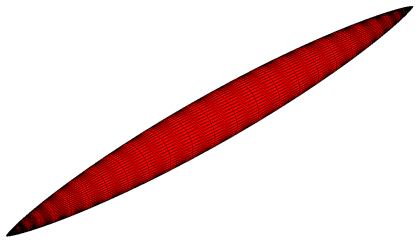
(e) Compare your estimated feasibility with a Monte Carlo simulation of feasibility.

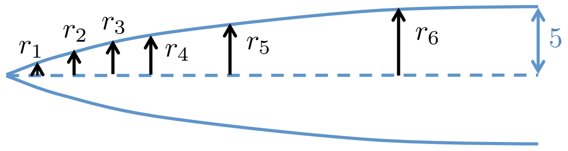
6.2 Surrogate-Based Optimization. Minimize the drag of a supersonic body of revolution using a global polynomial surrogate model. The details of this problem are on the following pages. If you select this option you will need to implement it in Matlab. (You could do it in Python but I've written the walk-through in Matlab and it would require two separate third-party Python libraries. It's probably not worth setting those up for this one-off analysis this late in the semester. You can access Matlab on any of the CAEDM machines.)

6.3 Dynamic Optimization. Complete the model predictive control homework from Dr. Hedengren. The write-up is attached to the back of this document, and the corresponding Excel spreadsheet is posted on Learning Suite.

Drag Minimization of a Supersonic Body of Revolution

In this problem we want to find the three-dimensional shape with the lowest supersonic wave drag for a fixed length and fixed maximum diameter. This problem has a known solution called the Sears-Haack body (Fig. 1a). This solution is useful, for example, in supersonic aircraft design. The goal there is to design the aircraft with area distributions close to that of the Sears-Haack body.


(a) "Sears-Haack" by Wolfkeeper at en.wikipedia.



(b) Six radii will be optimized along the body of revolution.

Figure 1: Sears Haack body and a corresponding parameterization of its shape for optimization.

For this exercise, let's pretend that the solution is unknown and use surrogate-based optimization to try to find the best design we can. We parameterize the body of revolution as shown in Fig. 1b, with six radii to optimize (the body has fore-aft symmetry, and the aft half is not shown). Points are clustered towards the nose where the shape changes rapidly. The interior 6 points are bookended by a point at the nose with 0 radius so that the body is closed, and a point with a fixed radius of 5 so that the diameter of the body is constant (all dimensions are unitless in this problem). An Akima spline is fit between the 7 points to define the full shape. The length of the body is 100. My colleague Dev Rajnarayan has provided the code that will be used to analyze the wave drag of the supersonic body as a function of the six radii. You will not have access to the source code, and the output has some "discretization noise" as well. This is to simulate real-life. Think of this code as a computational fluid dynamics (CFD) simulation. CFD is typically very timing consuming to run (the code in this exercise is quick) and unless you have access to the source to add analytic gradients, or gradients are already provided, then optimizing with CFD in the loop is usually too costly. Because of a limited time budget we want to find a good solution with as few function calls as possible. We may also want to explore variations in our design after finding the optimal solution. These requirements make the problem a good candidate for a surrogate model. We will construct a global polynomial surrogate model of the simulation and will optimize the surrogate rather than the actual function.

The following write-up will walk you through the steps of the analysis. I've given full implementation details, but you shouldn't just blindly copy the code. Pause between steps, try implementing some portions on your own, and print results to the screen. The goal is to understand what is happening at each step.

The external code is written in Java, but we can easily call Java from Matlab. We just need to load the jar files. Let's also clear everything.

Listing 1: Load Java files.

```matlab
clear; close all; clc; clear java;

% Add jar files to java classpath
javaaddpath({strcat(pwd,'/me575hw6.jar'), strcat(pwd,'/colt.jar')});
wavedrag = @WaveDragAS.compute;  % rename function call for convenience
```

Next let's set bounds on the design variables.

Listing 2: Set bounds.

```matlab
% Lower bound
lb = [0.05, 0.15, 0.65, 1.75, 3.00, 4.25];
```

```
% Upper bound
ub = [0.25, 0.40, 0.85, 2.25, 3.25, 4.75];
```

We need to sample our function to construct the initial surrogate. We will use Latin Hypercube Sampling (LHS). Matlab has a function called lhsdesign which can do this for us. We are going to start with a really small number of function calls (20) in our 6-dimensional space, but you can adjust that number as you like. The lhsdesign call will return numbers between 0 and 1. We will want to scale those so that our numbers span lb to ub.

$$x = lb + (ub - lb)x_{lhs} \tag{1}$$

While we are at it we will evaluate our wave drag coefficient function at each of the sample points. Note that LHS is based on random number generation so you will get different sample points every time.

Listing 3: Latin Hypercube Sampling.

```
% LHS sample
N = length(lb);  % number of design variables
M = 20;  % number of samples
X = lhsdesign(M, N);
F = zeros(M, 1);

for i = 1:M
    X(i, :) = lb + (ub - lb).*X(i, :);
    F(i) = wavedrag(X(i, :));
end
```

We now need to decide what type of surrogate to construct. Let's choose a polynomial model, and more specifically a quadratic model. (To do a better job we should do parameter studies where we vary the order of the model, try other surrogate models, and use cross-validation to select the model, but we will skip all of that to keep things simpler in this exercise.) There are 6 dimensions so our quadratic model will have 28 unknowns (the w's below) that we need to estimate. Note that there are only 28 terms because we don't want to double count terms like $x_1 x_2$ and $x_2 x_1$, which are actually the same.

$$\begin{aligned}\hat{f} &= w_1 + w_2 x_1 + w_3 x_2 + \ldots + w_8 x_1^2 + w_9 x_1 x_2 + w_{10} x_1 x_3 + \ldots + w_{28} x_6^2 \\ \hat{f} &= x_{exp}^T w\end{aligned} \tag{2}$$

The expanded x vector x_{exp} is something we are going to need a few times so let's write a function for it

Listing 4: Expand out quadratic terms.

```
function xexp = expandQuad(x)
xexp = [1, x, x(1)*x, x(2)*x(2:end), x(3)*x(3:end), x(4)*x(4:end), x(5)*x(5:end), x(6)*x(6)
    ];
end
```

We will now use our samples to perform a maximum likelihood estimate of w. We will form a linear system

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} [1 & x_1 & x_2 & \ldots & x_1^2 & x_1 x_2 & x_1 x_3 & \ldots & x_6^2]_{\text{at sample 1}} \\ [1 & x_1 & x_2 & \ldots & x_1^2 & x_1 x_2 & x_1 x_3 & \ldots & x_6^2]_{\text{at sample 2}} \\ \vdots \\ [1 & x_1 & x_2 & \ldots & x_1^2 & x_1 x_2 & x_1 x_3 & \ldots & x_6^2]_{\text{at sample M}} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{28} \end{bmatrix} \tag{3}$$

Which can be expressed as $y = \Phi w$ where Φ is

$$\Phi = \begin{bmatrix} x_{exp_1} \\ x_{exp_2} \\ \vdots \\ x_{exp_M} \end{bmatrix} \tag{4}$$

We decided on the M x_{exp} points from our sampling, evaluated our function to get the y's, and the unknowns are the w's. In code:

Listing 5: Assemble quadratic terms in matrix.

```
PHI = zeros(M, 28);
for i = 1:M
    PHI(i, :) = expandQuad(X(i, :));
end
```

We can now estimate the w terms through a separate optimization process. In this case the optimization is easy because our parametrization is linear in w. The equation $y = \Phi w$ does not have a unique solution, instead we are interested in a least-squares approximate solution (minimizing the sum of the squared errors between prediction and actual function values).

Listing 6: Maximum likelihood parameter estimation.

```
w = PHI\F;
```

We now have a surrogate model to compute the wave drag coefficient at any point (accuracy at any point is another question). The estimate of the drag at any point x would simply be

$$\widehat{C_D} = x_{exp}^T w \tag{5}$$

or in code:

Listing 7: Surrogate model evaluation.

```
CDhat = expandQuad(x)*w;
```

With our surrogate model constructed we can now optimize the model directly to find a minimum drag design (according to the surrogate). This is easily done

Listing 8: Optimization using surrogate model.

```
% starting design
x0 = [0.100,  0.198,  0.7,  1.800,  3.200,  4.550];

J = @(x) expandQuad(x)*w;
[xopt, fopt] = fmincon(J, x0, [], [], [], [], lb, ub);
```

The resulting point will probably not be particularly accurate. This is because we used an extremely small number of samples (20) to construct our initial surrogate in a 6D space, so we can't hope for high accuracy everywhere except near the sample points. One strategy is to sample more points initially to create a better global model. This would be a good approach if wanted to explore global changes to the designs. However, in this case we don't necessarily care about having an accurate model everywhere, but instead care about having an accurate model near the optimum. We intentionally kept the initial sampling small because we are going to add our "optimum" solution to our sample, recreate the surrogate with the new point added, and then reoptimize using the new surrogate. This should improve our surrogate near the actual optimum, and if we iterate a few times we should get a pretty good solution. (Note: If we are not careful we could add points to our surrogate that are really close to points that our already there, which can cause numerical issues with our least squares solutions—but we won't worry about that in this example.)

Fortunately, we also know exactly how far our surrogate is off by evaluating the original function. Let's run our optimization until the optimal point has a predicted drag coefficient within 5% (arbitrarily chosen) of the actual drag coefficient. We will see that even though the difference between our predicted function value $\hat{f}(\hat{x}^*)$ and the actual function value $f(\hat{x}^*)$ may be off by as much as 5%, the difference between the actual function value $f(\hat{x}^*)$ and the actual optimum $f^* = f(x^*)$ (which is what we actually care about) is much smaller—usually less than 1%. In addition to our error stopping criteria, we will limit the loop to 20 function calls at the most (generally the loop breaks after about 10). The loop is as follows:

Listing 9: Repeated optimization using surrogate model.

```
% starting design
x0 = [0.100,  0.198,  0.7,  1.800,  3.200,  4.550];

for i = 1:20

    % perform optimization using surrogate
    J = @(x) expandQuad(x)*w;
    [xopt, fopt] = fmincon(J, x0, [], [], [], [], lb, ub);

    % compute the error in our surrogate prediction
    factual = wavedrag(xopt);
    error = abs(factual - fopt)/factual;
    fprintf('surrogate error = %f\n', error);
    if error < .05
        fprintf('iter = %d\n', i);
        break
    end

    % otherwise, add another row to sample data
    PHI = [PHI; expandQuad(xopt)];
    F = [F; factual];  % be sure to use the actual function call not fopt the estimate

    % re-estimate the surrogate parameters
    w = PHI\F;
end
```

We could have used the optimal point of the previous solution as the starting point of the next iteration, but evaluating the surrogate is ridiculously cheap (just a vector-vector multiply) so it really doesn't matter.

We now have a pretty good solution to the minimum wave drag body of revolution and it only takes about 30 function calls. By comparison, optimizing with the actual function takes 88 function calls for this same starting point, and the result is generally not much better:

Listing 10: Optimize with actual (expensive) function.

```
J = @(x) wavedrag(x);
[xopt, fopt, exitflag, output] = fmincon(J, x0, [], [], [], [], lb, ub);
disp(output.funcCount);

% solution error
CD_sol = wavedrag(xopt);
fprintf('Optimization Error = %3.3g%%\n', (factual - CD_sol)/CD_sol*100);  % factual comes
    from the SBO
```

The error in using surrogate-based optimization (SBO) versus a full optimization in this problem is generally less than 1% (will vary randomly because of LHS) and takes about 1/3 as many function calls.

We can also make some comparisons to the theoretical minimum because it is a known solution for this problem. It is important to note that even optimizing with the actual function will not reach the theoretical minimum because of the discretization error (gets within 1.63%). The theoretical minimum wave drag for a fixed length and diameter is

$$C_{D min} = \frac{\pi d^2}{l} \tag{6}$$

We can print out our error as compared to the theoretical minimum

Listing 11: Theoretical Error

```
% theoretical error
d = 10; l = 100;
CD_min = (pi*d/l)^2;
fprintf('Theoretical Error = %3.3g%%\n', (factual - CD_min)/CD_min*100);
```

Finally, we can plot the difference in shapes between our SBO solution and the theoretical solution. The code is below and the resulting plot is shown in Fig. 2. Clearly the difference is extremely minimal.

Listing 12: Plot shape.

```
% ---------------- Plot geometry --------------------------
% SBO solution
x = l*[0.0, 0.005, 0.01, 0.025, 0.1, 0.2, 0.35, 0.5, 0.65, 0.8, 0.9, 0.975, 0.99, 0.995,
    1.0];
r = [0 xopt 5 fliplr(xopt) 0];

% theoretical optimal solution - Sears Haack
xSH = .01:.01:1;
rSH = d/2*sqrt(sqrt(1-xSH.^2)-xSH.^2.*log((1+sqrt(1-xSH.^2))./xSH));

xSH = l/2*[-fliplr(xSH) xSH] + l/2;
rSH = [fliplr(rSH) rSH];

figure; hold on;
h1 = plot(x,r,'b');
plot(x,-r,'b');
h2 = plot(xSH,rSH,'r--');
plot(xSH,-rSH,'r--');
axis equal;
legend([h1, h2], {'SBO','Sears-Haack'});
% ------------------------------------------------------------
```

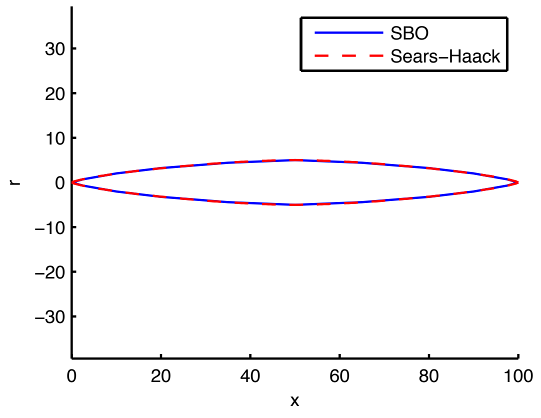
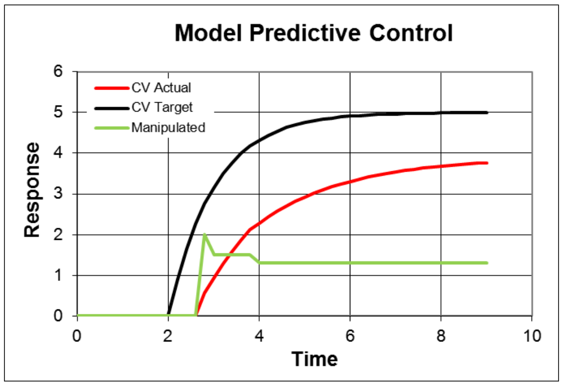


Figure 2: Surrogate based optimization (SBO) solution versus theoretical optimal shape (Sears-Haack).

Model Predictive Control - Homework

Model Predictive Control, or MPC, is an advanced method of process control that has been in use in the process industries such as chemical plants and oil refineries since the 1980s. Model predictive controllers rely on dynamic models of the process, most often linear empirical models obtained by system identification.



These models are typically in the finite impulse response form or linear state space form. Nonlinear elements can be added to avoid multiple model switching, gain scheduling, or other ad hoc measures commonly employed because of linear MPC restrictions.

For this homework assignment, you are requested to manually determine the set of inputs (MVs) that will achieve the desired target tracking. Use the associated Excel spreadsheet to complete this assignment. For the following 3 scenarios, complete parts a and b.

Scenario 1: Faster than the natural process time constant

Target Trajectory Parameters	
Final Target	5
Time Constant (tau)	1
Delay (theta)	2

Scenario 2: Equal to the natural process time constant

Target Trajectory Parameters	
Final Target	5
Time Constant (tau)	3
Delay (theta)	2

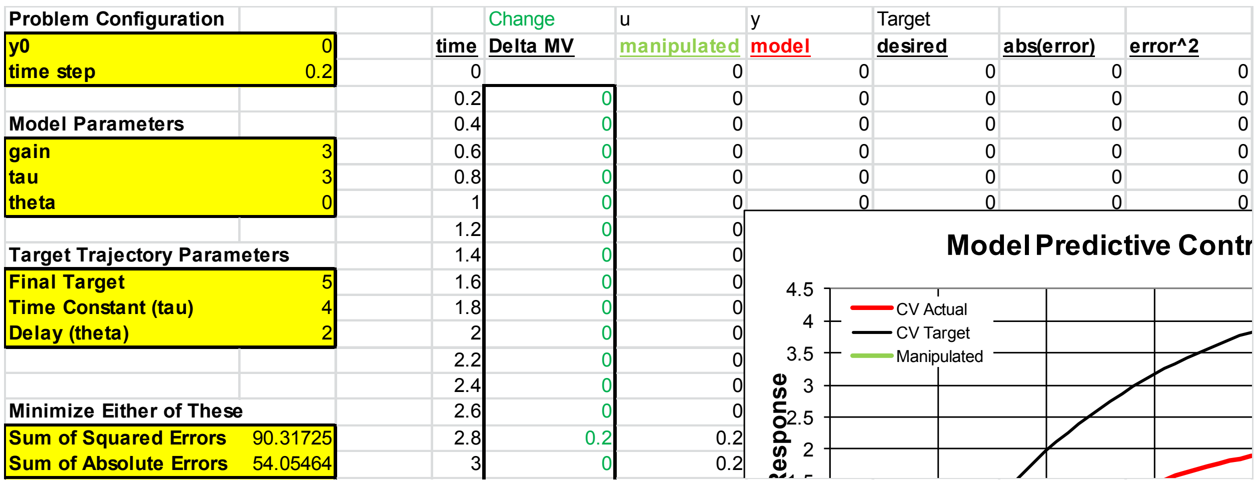
Scenario 3: Slower than the natural process time constant.

Target Trajectory Parameters	
Final Target	5
Time Constant (tau)	4
Delay (theta)	2

Part a) Use Excel solver to adjust the Delta MV moves to match the CV Actual with the CV Target. Minimize a sum of squared errors *OR* sum of absolute errors. Comment on the results and the ability of the solver to achieve the desired trajectory.

Part b) Manually adjust the solution that Excel solver provided to further improve the solution, if possible. Note that there are better solvers for this problem than the Nonlinear GRG solver that Excel provides. Manual adjustment of the solution is not required for commercial Model Predictive Control (MPC).

Part c) Create your own desired target trajectory (black line) and optimize the MV movements to best follow that path. Some potential paths include a saw-tooth design, step changes, ramps, etc.



Problem Configuration	
y0	0
time step	0.2
Model Parameters	
gain	3
tau	3
theta	0
Target Trajectory Parameters	
Final Target	5
Time Constant (tau)	4
Delay (theta)	2
Minimize Either of These	
Sum of Squared Errors	90.31725
Sum of Absolute Errors	54.05464

	Change	u	y	Target		
time	**Delta MV**	**manipulated**	**model**	**desired**	**abs(error)**	**error^2**
0		0	0	0	0	0
0.2	0	0	0	0	0	0
0.4	0	0	0	0	0	0
0.6	0	0	0	0	0	0
0.8	0	0	0	0	0	0
1	0	0	0	0	0	0
1.2	0	0				
1.4	0	0				
1.6	0	0				
1.8	0	0				
2	0	0				
2.2	0	0				
2.4	0	0				
2.6	0	0				
2.8	0.2	0.2				
3	0	0.2				

Fig 1. Screenshot of the Excel Spreadsheet for this exercise.