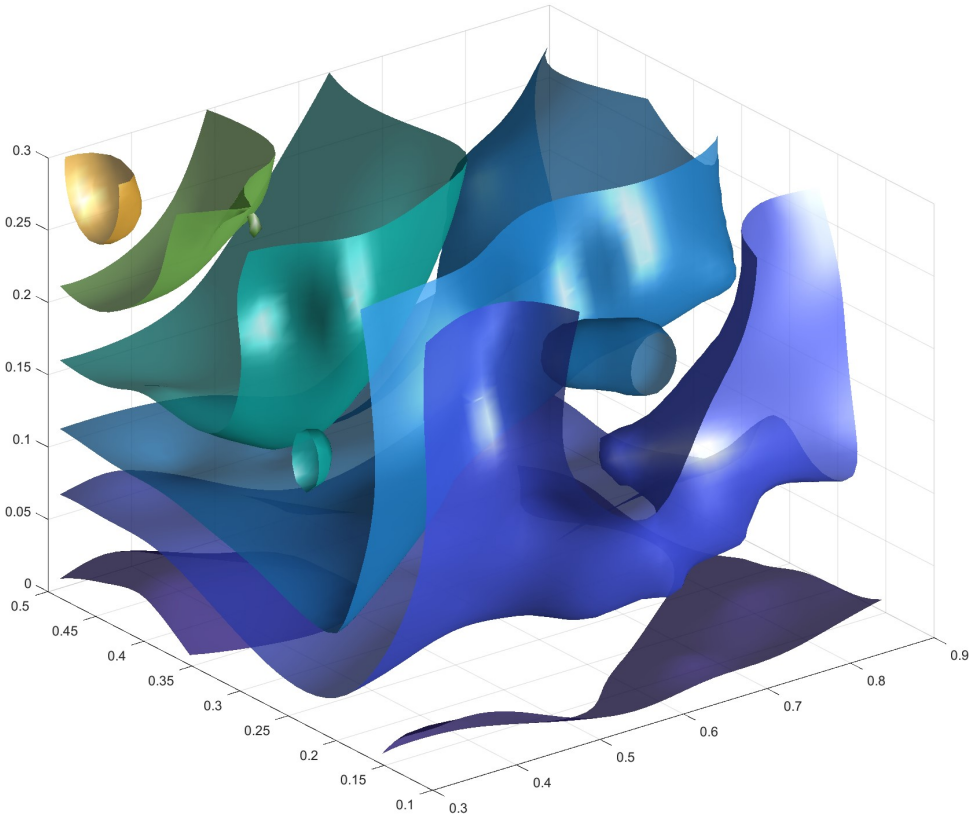
OPTIMIZING FUN:
CHAOS AT THE AMUSEMENT PARK



Contours of 4-D Surrogate Function in Full, 3-Dimensional Design Space

ID: 2204
Numerical Design Optimization
Prof. Hicken
Fall, 2021

ABSTRACT

In *Chaos at the Amusement Part: Dynamics of the Tilt-a-Whirl*, by Kautz and Huggard, it is proposed that the most enjoyment from a ride can be gained by increasing its unpredictability. “Unpredictability” in this case, refers to the standard deviation of the angular velocity components over the duration of the ride. Maximizing this standard deviation, in turn, leads to the rider failing to guess the ride’s next move, and thus being constantly surprised by the ride’s movements. Assuming the ride operates at a pace which does not induce excessive fear into the rider, this constant state of surprise in theory maximizes enjoyment, and will in turn cause riders to buy more tickets when they return to the amusement park.

INTRODUCTION

The Tilt-A-Whirl in this optimization problem consists of a rotating shaft with arms spanning outward in the radial direction. The speed at which the shaft rotates is the first design variable in this optimization, ω. At the ends of the arms are cars that ride along a track which varies in height sinusoidally, undergoing a full period of oscillation three times throughout one full rotation of the ride. The maximum angle of the in the vertical/radial plane is the second design parameter, α_1. The seats on each car (located at the end of each arm) rotate eccentrically on the car’s platform. The distance from the center of rotation to the center of the seats’ mass is r_2 , and the velocity at which the platform rotates is φ', or the derivative of the angle of rotation with respect to time. All other values are given. This configuration is depicted below in Figure 1. Given Values and design constraints are tabulated below in Table 1.

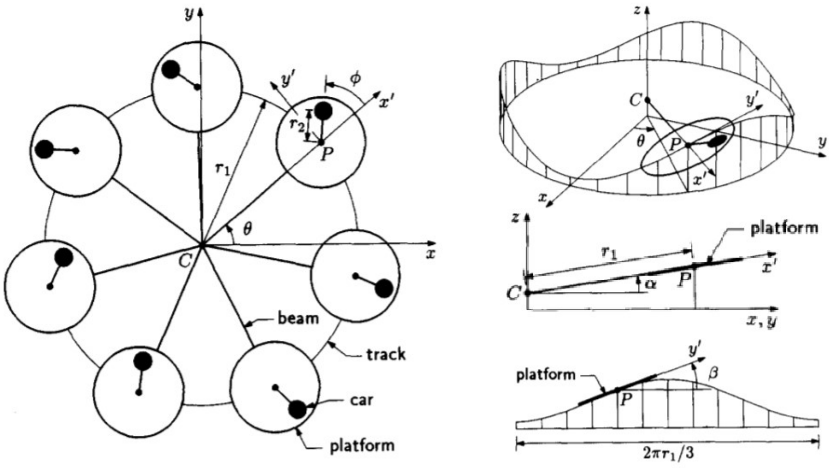


Figure 1: Tilt-a-Whirl System Diagram

Table 1: Design Constrains and Fixed Values

Parameter	Value or Constraint	Description
ω	3 rpm ≤ ω 60 2π rpm ≤ 8 rpm	Total Angular Velocity
r2	0.1 m ≤ r2 ≤ 1.5 m	Secondary Arm Length
α1	0 rad ≤ α1 ≤ 0.3 rad	Primary Arm Angle from Ground
r1	4.3 m	Primary Arm Angle
α0	0.036 rad	Primary Arm Incline Offset
Q0	20	Non-Dimensional Parameter

This optimization problem posed is deceivingly difficult. The governing equation for the system is a second order ordinary differential equation, with three design variables, and is shown below in Equation 1.

$$\frac{d^2\phi}{d\tau^2}+(\gamma/Q_0)\frac{d\phi}{d\tau}+(\epsilon-\gamma^2\alpha)\sin\phi+\gamma^2\beta\cos\phi=0,$$

$$\alpha=\alpha_0-\alpha_1\cos\tau, \qquad \tau=3\omega t,$$

$$\beta=3\alpha_1\sin\tau. \qquad \gamma=(1/3\omega)\sqrt{g/r_2},$$

$$Q_0=(m/\rho)\sqrt{gr_2^3}. \qquad \epsilon=r_1/9r_2,$$

Equation 1: Governing Equation of Motion

Second order ODE's are difficult for computational solvers to solve, and require lots and lots of computing power. For this reason, this ODE will be reduced to first order so that MATLAB's "ode45" solver can find solutions based on given inputs. This operation still requires immense computing power, and the entire design space cannot be feasibly explored. For this reason, a surrogate function will be constructed in order to more simply represent the trends in the design space, and find a solution without iteratively solving the ODE. Lattice Hypercube Meshing was used to sample points, due to its evenly-distributed randomness, when compared to Monte-Carlo, which can have uneven sample density in the design space. An example of these sampling methods are compared in Figure 2.

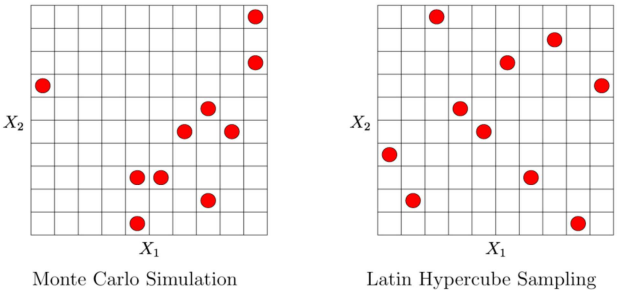


Figure 2: Monte Carlo vs Latin Hypercube Sampling

The reduced ODE was examined at each of the aforementioned sample points using MATLAB's ode45 function, and then the resulting angular velocity, φ', for each value in non-dimensional time and angle, φ, for each value in non-dimensional time. The standard deviation of all of the velocity components was taken, thus creating a scalar output with three scalar inputs. These inputs and their corresponding output were then given to a Gaussian Process Machine Learning toolbox in order to construct a surrogate function for optimization. This surrogate became the new objective function, which takes significantly less time to analyze than the original differential equation. MATLAB's fmincon function was used to find the minimum value of the objective function, which was the negative of the surrogate. Since fmincon defaults to finding a minimizer, it is best practice to use a negative version of the objective function if you wish to maximize it. After optimization, the results are negated again.

Although the surrogate function is orders of magnitude faster than solving the ODE for many points, it is not without error. These two operations were placed inside a "while" loop that continued to iterate until error in the optimal standard deviation falls below 0.05 rad per second. The error was calculated by subtracting the true function's $f(\omega^*,r2^*,\alpha1^*)$ from the surrogate's, using the same inputs for each and comparing outputs. If the absolute value of this difference was greater than 0.05 rad/s, the optimal values from the previous iteration would be used as the initial values in the current iteration.

RESULTS

All results in this paper will detail the case in which 500 sample points were used to formulate the surrogate function. Less is sufficient, but more tends to have less error and thus fewer iterations and better results. A noise value of 0.1 was also used, though analysis with other values were not explored. Additionally, hyper-parameters of log(1/4) and log(1.0) were used in tandem with the Matern Covariance function, due to its ability to capture non-smooth functions

The maximum standard deviation of angular velocities observed in this analysis was 6.5524 rad/s, with local maximizers of $\omega^* = 0.79409$ rad/s, $r2^* = 0.1$ m, and $\alpha1^* = 0.26537$ rad. The error for this calculation was as low as 0.0269. When running with 1000 sample points, a standard deviation above 7 was often observed, but due to computing time it is not feasible to use this for every iteration.

Since this system is highly chaotic, and Latin Hypercube Sampling always chooses different points, the number of iterations varied greatly between runs. Error generally appeared to decrease drastically for the first few iterations, and then remained relatively volatile for remaining iterations. Figure 3 shows the error, surrogate result, and true solution over iterations in order to show mesh convergence. In this case, convergence was reached at iteration 5. Increasing the number of sample points tended to decrease the number of iterations to reach convergence, but had little to no effect on decreasing error on any given iteration.

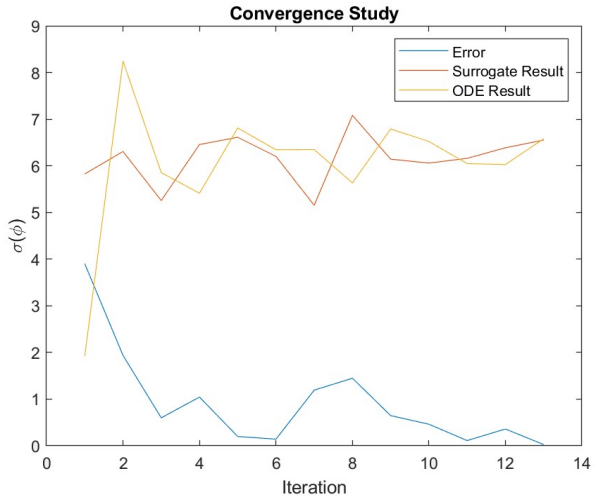


Figure 3: Error, Surrogate, and ODE Standard Deviation vs Loop Iteration

In this particular analysis, the the optimal standard deviation with respect to w was plotted using both the surrogate and the real function. In this plot, r2* and α1*are held constant in order to show the pure change with respect to w in this 1-d slice of the design space.

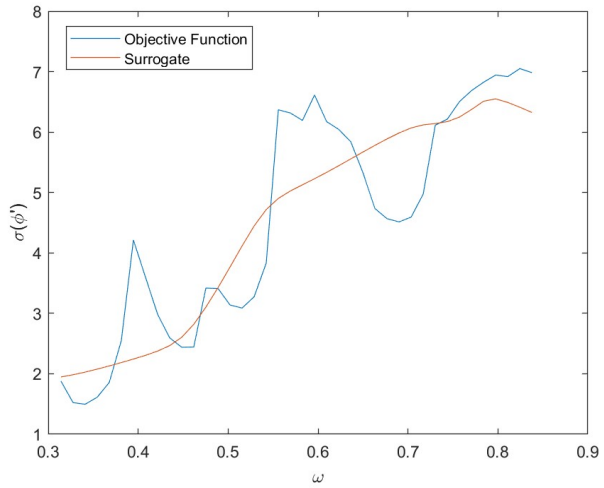


Figure 4: Standard Deviation of Local Angular Velocity vs Total Angular Velocity (r2 = 0.1 m, α1* = 0.26537 rad)*

Viewing Figure 4, it is clear the surrogate captures the major trends of the function, but neglects the nuanced curves. When viewing the contours of the solution in this design space in three dimensions, this fact remains true. Lack of computing power prevented the formulation of 3-D slices for the true solution, but Figure 5 below shows the design space of the surrogate function and the corresponding output, with one value held at its optimal range and the other two varying across their whole design space.

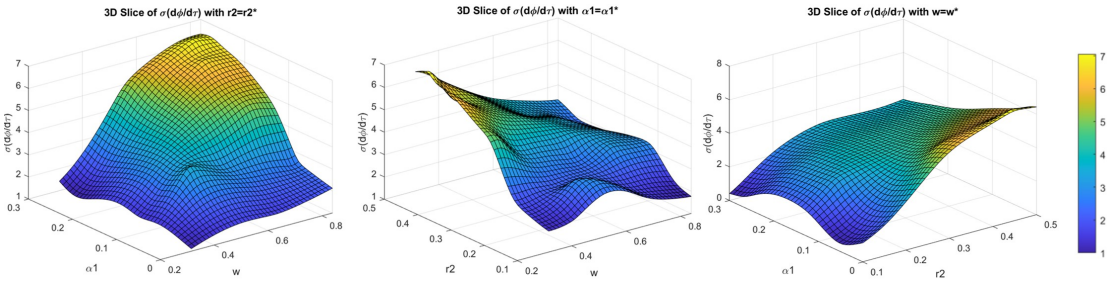


Figure 5: 3D Slice of Objective Function Output with r2=r2 (left), a1=a1* (middle) and w=w* (right)*

Figure 6 depicts the contours of the solution in the design space. The surrogate is on the left, and is extremely smooth and easy to understand which direction to move the value to achieve an optimal result, while the true function appears very complex with interlocking contours which would be extremely difficult to optimize. The overall trend remains the same for each, though. Minimize the total angular velocity, and maximize the length of the secondary arm and the primary arm's incline angle, and the standard deviation will be pretty close to the maximum possible value.

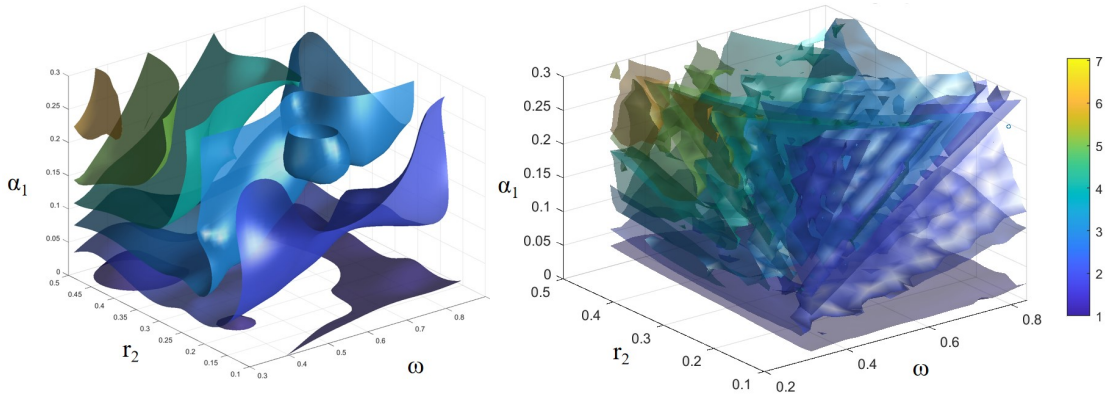


Figure 6: 3-D Contours of Standard Deviation for Surrogate Function (left) and True ODE Function (right) Spanning Full Design Space.

The true function was analyzed at 8000 evenly-spaced points in the design space, and took over 30 minutes to solve all of them. Meanwhile, the surrogate was

analyzed at 64,000 points, and was finished processing in 12.8 minutes. This is a testament to the decreased computing power required for the surrogate function.

This function appears to be quite chaotic, and it may be, as small changes in input parameters can yield large changes in outputs. Indeed, the 3D contour plot of the surrogate was never exactly the same from run to run, despite identical inputs (aside from sample locations). This is why a larger number of sample points yield a more confident result, because the program has a higher-resolution snapshot of how the function behaves across the design space.

Viewing the instantaneous velocity across the dimensionless time span, it was easy to see that (what appears to be) randomness is occurring. A rider would not be able to predict such behavior, and this constant state of being surprised would contribute to optimal enjoyment. Figure 7 depicts the Angular Velocity vs Dimensionless time.

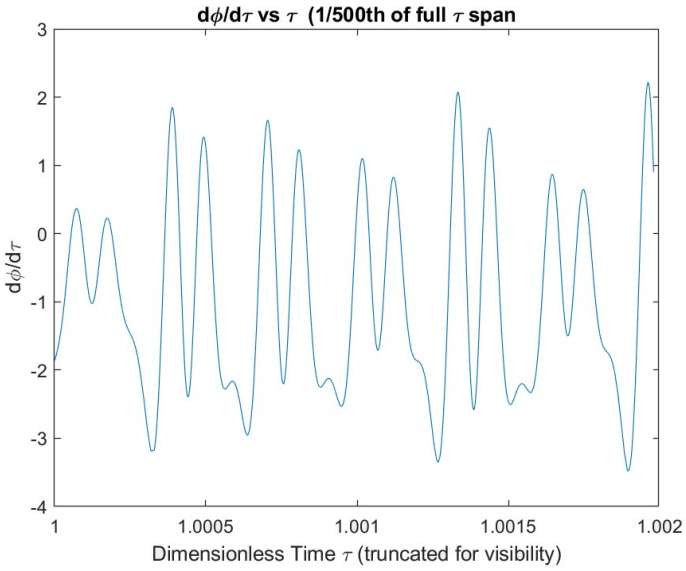


Figure 7: Angular Velocity over Dimensionless Time with the Standard Deviation Maximized

Given that the solution using on the nominal input values yielded a standard deviation of 1.4141, the optimized result of 6.5524 is a 480% improvement, which almost absolutely directly correlates to a 480% improvement in overall fun and enjoyment.

Appendix 1: Main Code

```
close all
clear all
clc
tic
% %Import GPML Toolbox
functionality
% mydir =
'~\Documents\School\Design Op\gpml-
matlab-v3.6-2015-07-07';
% addpath(mydir(1:end-1))
% addpath([mydir,'cov'])
% addpath([mydir,'doc'])
% addpath([mydir,'inf'])
% addpath([mydir,'lik'])
% addpath([mydir,'mean'])
% addpath([mydir,'prior'])
% addpath([mydir,'util'])

%% User Inputs
Max_Iter=25; %Maximum Number of
Iterations when looping to solve
ODE
tolerance=0.05;
n=500; %Number of Latin Hypercube
Sample Points
tspan = 10000; %time-fidelity
%Noise
noise = 0.1 ;
hyp.cov= [log(1/4); log(1.0)];

%% Given and Nominal Values
%Given Vals
r1 = 4.3; %(m)
Q0 = 20; %kg*m^2/s
alpha0 = 0.036; %rad
%Nominal Vals for Design Vals
nom_w = 6.5*2*pi/60; %rad/s
nom_r2 = 0.8; %m
nom_alpha1 = 0.058; %rad

%% Iniitialize Parameters
Iter= 1;

%initialize design variable's
arrays
w_star = nom_w; %design variable 1
r2_star = nom_r2; %design variable
2
alpha1_star = nom_alpha1; %design
variable 3

fx_star = zeros(1,Iter);
err = zeros(1,Iter);

%% ---<<< SURROGATE >>>---

%initialize iteration status to
continue. Will update in while loop.
status=0; %status=0: keep iterating.
If /=0, optimization is complete.

while Iter < Max_Iter && status ==
0
tic
disp(['Iteration #',num2str(Iter)])
% Latin Hypercube Sampling
%Latin Hypercube Sampling using
lhsdesign(n,3)
%n Number of Samples (User Input) &
3 Design Variables
P = lhsdesign(n,3); %Defines n
points in a nondimensional space of
3 dimensions

%Rescale LHS Samples to entire
feasiblie design space
% RESCALING |MIN |MAX
P(:,1) = rescale(P(:,1), 3*2*pi/60,
8*2*pi/60); % w (rad/sec)
P(:,2) = rescale(P(:,2), 0.1,
1.5 ); % r2 (m)
P(:,3) = rescale(P(:,3), 0.0,
0.3 ); % alpha1 (rad)
% Calculating Spin Up velocity
% Non Dimensional Spin-Up (SU) Time
(Total)
T = 100000;
% Initial Condition for spin-up of
y= [y1; y2] = [ dPHI/dt; PHI ]
y0_SU=[0 0];

%Equation 27, reduced to first
order for ODE45 Solver.
%The final tau and y values found
here are the initial conditions of
%the steady state
solutionCalcODEFunc(r1,alpha0,Q0,va
rs,y,tau)
[tau_SU, y_SU] = ode45(@(tau,y)
SolveODE(r1,alpha0,Q0,[w_star;
r2_star; alpha1_star],y,tau), [0
tspan], y0_SU);

% Post Spin-Up
% Initial Condition AFTER SPIN UP
of y= [y1; y2] = [ dPHI/dt; PHI ]
y0 =[ y_SU(end,1) ; y_SU(end,2) ];
%Initialize Standard Deviation
vector
STDEV=zeros(n,1);

%Iterate ODE for each Sample Point
for i = 1:n
```

```
%cluster design variables into one
array
w_r2_a1 = [P(i,1); P(i,2); P(i,3)];
%Solve Equation 27 using the final
conditions of the spin up period
%as the initial conditions
[tau,y] = ode45(@(tau,y)
SolveODE(r1,alpha0,
Q0,w_r2_a1,y,tau), [tau_SU(end)
tau_SU(end)+tspan], y0);
%Calculate the standard deviation
of the velocity with respect to
%time
STDEV(i,1) =
CalcStdDev(T,y(:,1),w_r2_a1(1),tau);
end
% Squared Expansion covariance
covfunc= {@covMaterniso,1};

% Likelihood Function using gauss
process
likfunc = @likGauss ;


%Hyperparameters
hyp.lik = log(noise);
hyp = minimize(hyp, @gp, -100,
@infExact, [], covfunc,likfunc, P,
STDEV);

% OPTIMIZATION
%create objective function
obj = @(z) objGP(hyp,covfunc,
likfunc, P, STDEV, z);

%Constraints for design variables
% w r2 alpha1
lwr_cnstrnt = [ 3*2*pi/60, 0.1,
0.0 ] ;
upr_cnstrnt = [ 8*2*pi/60, 1.5,
3.0 ] ;
%fmincon options (display iteration
information)
options =
optimoptions('fmincon','Display','i
ter','Algorithm','active-set');
%Run Fmincon using objective
function
[x,f] = fmincon(obj,[w_star r2_star
alpha1_star],[],[],[],[],lwr_cnstrn
t,upr_cnstrnt,[],options);
%invert output to max instead of
min
vals=-f;
%find error size
test =
CalcStdDev(T,y_SU,x(1),tau_SU);
err = abs(vals-test(1));
disp(vals)
disp(test(1))
disp(err(1))
Err(Iter)=err(1);

Vals(Iter)=vals;
bestSTDDEV(Iter)=max(Vals);
%deconstruct design var array into
separate vectors
w_star = x(1);
r2_star = x(2);
alpha1_star = x(3);
%Store Values for use outside of
While loop to see convergence
fx_star(Iter) = vals;
err_mem(Iter) = err;
testmem(Iter) = test(1);
bestErr(Iter)=min(err_mem);

% While Loop Iteration Criteria
%If error is below tolerance, break
%If error is above tolerance,
iterate
fprintf('Iteration Clock\n')
toc
if err < tolerance
fprintf('Standard Deviation of
Angular Velocity Maximized')
disp(fx_star(end))
itertmp=[1:Iter];
disp(['Iter ',num2str(itertmp)])
disp(['f(x*) ',num2str(fx_star)])
disp(['Error ',num2str(err_mem)])
status = 1; %changes status to 1 if
error is low, ending the iterations
else
%prepare for next iteration
n = n +1;
Iter = Iter +1;
P = [P; x];
%show error
fprintf('error report: ')
disp(err)
fprintf('\nOptimal Value Found.
Error above Tolerance. Iterating
Again..\n')
itertmp=[1:Iter];
disp(['iter ' ,num2str(itertmp)])
disp(['f(x*) ' ,num2str(fx_star)])
disp(['Error ' ,num2str(err_mem)])
fprintf('\n\n')
disp(['Best f(x*)
',num2str(bestSTDDEV)])
disp(['Lowest
Error',num2str(bestErr)])
continue %proceed with iteration
end

end
if err > tolerance
```

```
disp(['Optimization failed\nCould
not find values within tolerance
within ',Iter,' Iterations.\n'])

end

%% Post Processing

% close all
%% Plot Results
hold off
%Plot Mesh Convergence & Error
figure(1)
plot(itertmp,err_mem)
hold on
plot(itertmp,fx_star)
plot(itertmp,testmem)
title('Convergence Study')
xlabel('Iteration')
ylabel('\sigma(\phi)')
legend('Error','Surrogate
Result','ODE Result')


%Clip area in order to view some
angular velocities
ntrunc=500;
for i = 1:size(tau)/ntrunc
periodcalc_tau(i)=tau(i)/10000;
periodcalc_iter(i)=i;
end

%Full Tau Range
figure(2)
plot(tau,y(:,1));
hold on
title(['d\phi/d\tau vs \tau ' ])
ylabel('d\phi/d\tau')
xlabel('Dimensionless Time \tau')
%Clipped Tau Range
figure(3)
step=linspace(1,periodcalc_iter(end)
,periodcalc_iter(end));
plot(periodcalc_tau,y(step,1))
title('d\phi/d\tau vs \tau(1/500th
of full \tau span')
hold on
ylabel('d\phi/d\tau')
xlabel('Dimensionless Time \tau
(truncated for visibility)')

%Number of ODE bins for each design
variable
N_ode = 20;
%Number of bins in surrogate
function for each design variable
N=40;
%Create points for surrogate
plotting
wvec = linspace(3,8,N)*2*pi/60;
r2vec = linspace(0.1,0.5,N);
a1vec = linspace(0.0,0.3,N);
stddev_vec=zeros(N,1);

%plotting in one slice of r2 & a1
for i = 1:N
vars = [wvec(i) r2_star
alpha1_star];
y0 = [y_SU(end,1) y_SU(end,2)];
tau0 = [tau_SU(end)
tau_SU(end)+tspan];
[ tau, y] = ode45(@(tau,y)
SolveODE(r1, alpha0,Q0,vars,y,tau),
tau0, y0);
stddev_vec(i) =
CalcStdDev(T,y(:,1),vars(1),tau);
end



%Solve ODE at all N_ODE^3 points
%Create Points
wvec_CORSE =
linspace(3,8,N_ode)*2*pi/60;
r2vec_CORSE =
linspace(0.1,0.5,N_ode);
a1vec_CORSE =
linspace(0.0,0.3,N_ode);
stddev_vec3d=zeros(N_ode,N_ode,N_od
e);
disp(['Commencing ODE Solution
process at
',num2str(N_ode*N_ode*N_ode),'
Points'])
y0_3d = [y_SU(end,1) y_SU(end,2)];
tau0_3d = [tau_SU(end)
tau_SU(end)+tspan];
tic
%Loop over all dimensions of design
space
for i = 1:N_ode
disp(['running... ',num2str(i)])
for j = 1:N_ode
for k=1:N_ode
%Solve ODE at i,j,k point
[tau3d, y3d] = ode45(@(tau,y)
SolveODE(r1,
alpha0,Q0,[wvec_CORSE(i)
r2vec_CORSE(j)
a1vec_CORSE(k)],y,tau), tau0_3d,
y0_3d);
stddev_vec3d(i,j,k) =
CalcStdDev(T,y3d(:,1),wvec_CORSE(i),
tau3d);
end
end
end
```

```
disp(['ODE Solved! Solve Time:
',num2str(toc)])

fprintf('\n')

%Repeating process for Surrogate
Function

s2 = zeros(1,N);
covfunc = {@covMaterniso, 1};
likfunc = @likGauss;
hyp.lik = log(noise);
hyp = minimize(hyp, @gp, -100,
@infExact, [], covfunc, likfunc, P,
STDEV);
disp(['Commencing Surrogate
Function Creation at
',num2str(N*N*N),' Points'])
for i = 1:N
disp(['running... ',num2str(i)])
windep(i) =
gp(hyp,@infExact,[],covfunc,
likfunc, P, STDEV,[wvec(i), r2_star,
alpha1_star]);
r2indep(i)=
gp(hyp,@infExact,[],covfunc,
likfunc, P, STDEV,[w_star, r2vec(i),
alpha1_star]);
a1indep(i)=
gp(hyp,@infExact,[],covfunc,
likfunc, P, STDEV,[w_star, r2_star,
a1vec(i)]);
for j = 1:N
contourslice_wconst(i,j) =
gp(hyp,@infExact,[],covfunc,
likfunc, P, STDEV,[w_star, r2vec(i),
a1vec(j)]);
contourslice_r2const(i,j) =
gp(hyp,@infExact,[],covfunc,
likfunc, P, STDEV,[wvec(i), r2_star,
a1vec(j)]);
contourslice_a1const(i,j) =
gp(hyp,@infExact,[],covfunc,
likfunc, P, STDEV,[wvec(i),
r2vec(j), alpha1_star]);
for k = 1:N
contour3d(i,j,k) =
gp(hyp,@infExact,[],covfunc,
likfunc, P, STDEV,[wvec(i),
r2vec(j), a1vec(k)]);
end
end
end

figure(4)
plot(wvec, stddev_vec)
xlabel("\omega")
ylabel("\sigma(\phi')")
hold on
plot(wvec,windep)
legend("Objective Function",
"Surrogate")


figure(5)
surf(r2vec,a1vec,contourslice_wcons
t)
xlabel("r2")
ylabel("\alpha1')")
zlabel('\sigma(d\phi/d\tau)')
title('3D Slice of
\sigma(d\phi/d\tau) with w=w*')
figure(6)
surf(wvec,a1vec,contourslice_r2cons
t)
xlabel("w")
ylabel("\alpha1")
zlabel('\sigma(d\phi/d\tau)')
title('3D Slice of
\sigma(d\phi/d\tau) with r2=r2*')
figure(7)
surf(wvec,r2vec,contourslice_a1cons
t)
xlabel("w")
ylabel("r2")
zlabel('\sigma(d\phi/d\tau)')
title('3D Slice of
\sigma(d\phi/d\tau) with
\alpha1=\alpha1*')

wslice=linspace(3*2*pi/60,
8*2*pi/60,3);
r2slice=linspace(0.1, 1.5,2);
a1slice=linspace( 0.0, 0.3,2);


numclip1=7;

clip=linspace(1,fx_star(end)+0.5,nu
mclip1);
for i =1:numclip1
figure(8)
isosurface(wvec,r2vec,a1vec,contour
3d,clip(i))
hold on
end
figure(8)
scatter3(w_star,r2_star,alpha1_star,
fx_star(end))
alpha(0.8)
view(3)
grid on
title('Contours of Surrogate in
Full Design Space')
xlabel('\omega')
ylabel('r2')
zlabel('\alpha1')
```

```
numclip2=7;
clip=linspace(1,fx_star(end)+0.5,nu
mclip2);
figure(9)
for i =1:numclip2
isosurface(wvec_CORSE,r2vec_CORSE,a
1vec_CORSE,stddev_vec3d,clip(i))
hold on
end
alpha(0.5)
scatter3(w_star,r2_star,alpha1_star,
fx_star(end))
view(3)
grid on
title('Contours of ODE for Full
Design Space')
xlabel('\omega')
ylabel('r2')
zlabel('\alpha1')

disp('Optimal Inputs:')
disp(['w* = ',num2str(w_star)])
disp(['r2* = ',num2str(r2_star)])
disp(['alpha1*
=',num2str(alpha1_star)])
fprintf('\n')
disp('Optimal Output:')
disp(['f(w*,r2*,a1*) =
',num2str(fx_star(end))])
fprintf('\n')
fprintf('Total Time Report: \n')

toc
```

Appendix 2: objGP

```
function [negmean] =
objGP(hyp,covfunc,likfunc,X,STDEV,z)
[mean s2] =
gp(hyp,@infExact,[],covfunc,
likfunc, X, STDEV, z);
negmean = -mean;
end
```

Appendix 3: CalcStdDev

```
%Standard Deviation calculator
function [standarddeviation] =
CalcStdDev(T,y1,w,tau)
%average y value
meany = 1/T * trapz(tau,y1);
% elementwise deviation from mean,
squared
dev = (y1-meany).^2;
%create "timeline" with the same
number of elements as "dev"
t=linspace(0,T,size(dev,1));
%integrate y deviation with respect
to t
int = trapz(t,dev);
%Calculate the standard deviation
standarddeviation = 3*w*sqrt(int/T);
end
```

Appendix 4: SolveODE

```
%Function to calculate [ y(1) =
d(phi)/d(tau) ]
function [dphidt] =
SolveODE(r1,alpha0,Q0,w_r_a,y,tau)

%deconstruct design vars
w = w_r_a(1);
r2 = w_r_a(2);
alpha1 = w_r_a(3);
g=9.8067; %gravity
%secondary design vars
gam = 1/(3*w)*sqrt(g/r2);
alpha = alpha0-alpha1*cos(tau);
beta = 3*alpha1*sin(tau);
eps = r1/(9*r2);

%calculate dphidtau
dphidtau1 = -gam/Q0 .*y(1)-(eps-
gam.^2.*alpha).*sin(y(2))-
gam.^2*beta.*cos(y(2));
dphidtau2 = y(1);
dphidt = [dphidtau1; dphidtau2];
end
```

Appendix 5: CalcODEStdDev

```
function [STD_DEV] =
CalcOdeStdDev(vars,y,tau,T,tspan)
%Given Parameters
r1=4.3; %m
Q0=20; %kgm2/s
alpha0 = 0.036; %rad
% y= [y1; y2] = [ dPHI/dt; PHI ]
y0= [y(end,1) y(end,2)];

%Equation27 - Solving ODE with
ODE45
[tau,y] = ode45(@(tau,y)
SolveODE(r1,alpha0,Q0,vars,y,tau),[
tau(end) tau(end)+tspan],y0);
%calculate the standard deviation
using a tailor-made function
STD_DEV =
CalcStdDev(T,y(:,1),vars(1));
end
```