Numerical Design Optimization

Report 3: Tilt-A-Whirl Optimization

Unique ID: 1059

Professor Hicken

1. Executive summary

1.1 Problem Statement

The Tilt-a-Whirl is a common theme park attraction, shown in Figure 1.1.1. The Tilt-a-Whirl is a ride that's motion is quite unpredictable, this is the aspect that people seem to find most enjoyable about this ride. The hypothesis is that increasing the standard deviation of angular velocity will increase unpredictable nature and thus the enjoyment of this ride. Figure 1.1.2 shows a more indepth schematic of the Tilt-a-Whirl.

Figure 1.1.1: The Tilt-A-Whirl

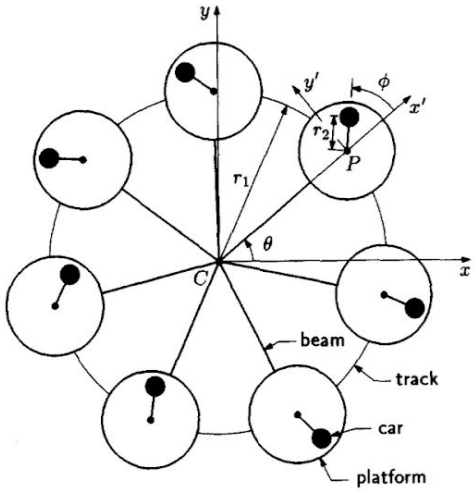

Figure 1.1.2: Tilt-a-Whirl schematic

1.2 Overview of methods

Methods for analysing the motion, finding the standard deviation, and optimization will be needed.

The motion is analysed using an ODE obtained from a paper on the dynamics of the tilt-a-whirl by Kautz and Huggard. This ODE is shown in Equation 1.2.1. This equation is dependant on values for ε, γ, α, β, and Q_0 which are dependent on the design variables: r2, α_1, and ω, a more in depth explanation of these values is in Analysis Methods. The ODE is solved using the Ode45 function in MatLab. From the results of the ODE solver the standard deviation of $\frac{d\varphi}{d\tau}$ can be found, which is the objective function.

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

Equation 1.2.1: ODE to find the values for $\frac{d\varphi}{d\tau}$ and τ

The optimization method chosen is surrogate modeling due to a very expensive to compute objective function. In order to sample the design space Latin Hypercube Sampling is used to generate values for the design variables: r2, α_1, and ω, then it is plugged into objective function and the output is stored in a separate matrix. These are then used to find the surrogate function using methods from the GPML library.

For the Tilt-a-Whirl to be able to be manufactured and to be able to fit in the allotted space, constraints on the design are placed. The optimization engine used is "fmincon" found in MatLab's Optimization Toolbox.

1.3 Overview of results

The optimal standard deviation of 4.9102 radians/sec which is an improvement of 222.5% over nominal.

2. Analysis method

2.1 Assumptions

In order to simplify this problem some assumptions are made, many assumptions are taken from the paper by Kautz and Huggard. These include: assuming the mass of the car acts as a point mass; setting the parameter Q0, the equation for which is shown in Equation 2.1.1, to 20, this may not be valid as Q0 is dependant on a design variable, r2, however it is a good rough estimate; assuming the path the cars follow is sinusoidal, this is a good estimate because as design engineers the shape is ours to control. R1 is also fixed at 4.3 meters.

$$Q_0 = (m/\rho)\sqrt{gr_2^3}$$

Equation 2.1.1: Parameter Q_0

2.2 Tilt-a-Whirl Analysis

The ODE from the paper by Kautz and Huggard, shown again for convenience, is dependant on values for ε, γ, α, β, and Q_0 and is in respect to non-dimensional time, τ. The equations for each of these parameters and Tau are shown in *Equations 2.2.1 thru 2.2.5* (Q_0 has already been shown in Assumptions). ε is a ratio of the r_1 and r_2; α *and* β are parameters that deal with the sinusoidal path; γ deals with scaling the angular velocity. Schematics describing these values are shown in *Figures 2.2.1 - 2.2.4.*

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

Equation 1.2.1: ODE to find the values for $\frac{d\phi}{d\tau}$ *and* τ

$$\alpha=\alpha_0-\alpha_1\cos\tau$$

Equation 2.2.1: Alpha

$$\epsilon=r_1/9r_2$$

Equation 2.2.2:Epsilon

$$\beta=3\alpha_1\sin\tau.$$

Equation 2.2.3: Beta

$$\tau=3\omega t,$$

Equation 2.2.4: Tau, non-dimensional time

$$\gamma=(1/3\omega)\sqrt{g/r_2},$$

Equation 2.2.5: Gamma

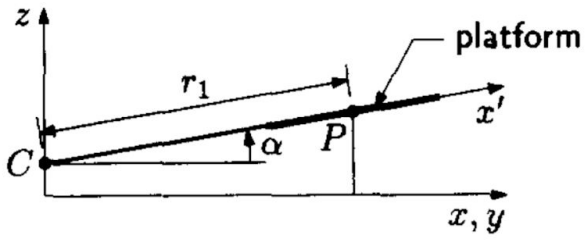

Figure 2.2.1: Representation of Alpha and r_1

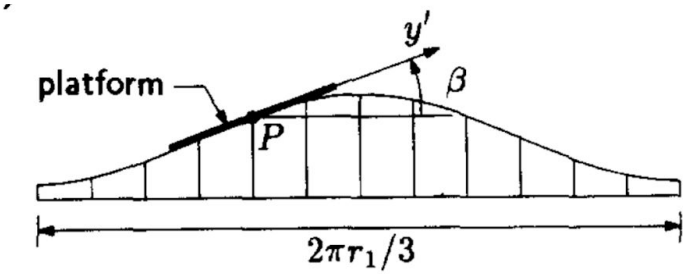

Figure 2.2.2: Representation of Beta

This ODE is solved using the ODE 45 solver in MatLab, the time span is set for it to solve from $\tau_{spin-up}$ to τ_{end} which are .3 * ω * 10,000 and 3 * ω * 10,000 and the initial conditions are [0,0]. This ensures that the simulation runs long enough as if it is running multiple trials, and that the influence of the initial conditions of the ODE are minimized. In order for the ODE to be in the form ODE45 takes, it first has to be in the form of 2 first order ODEs. The general form is shown in *Equation 2.2.1,*

$$\frac{d^2\phi}{d\tau^2} = F\left[\phi, \frac{d\phi}{d\tau}\right]$$

Equation 2.2.6: General form of turning second order ODE into two first order ODEs

To find the standard deviation, the general form is shown in Equation 2.2.1. Here T is non-dimensional time, y_1 is $\frac{d\phi}{d\tau}$ and ω is the average angular velocity.

$$f(x) = 3\omega\sqrt{\frac{1}{T}\int_0^T (y_1 - \bar{y}_1)^2 \, d\tau}$$

Equation 2.2.7: Standard Deviation of $\frac{d\phi}{d\tau}$

To ensure the ODE is working properly, values are compared against literature for a nominal design of: ω=6.5×(2π/60)rad/s, r_2 = 0.8m and α_1 = 0.058 rad. The result obtained from analysis is a standard deviation of 1.5227 rad/sec and literature values range from [1.49,1.61] for the same problem. Next a sweep of ω is done, keeping r_2 and α_1 constant at nominal values. The results are shown in *Figure 2.2.1,* compared to nominal shown in *Figure 2.2.2.* The paper by Kautz and Huggard reports that at very high values of ω the car is "locked at the outer edge of the platform" and thus there is very little change in φ

and very low values of ω the car reaches steady state quickly. *Figure 2.2.3 and Figure 2.2.4* show this phenomenon.

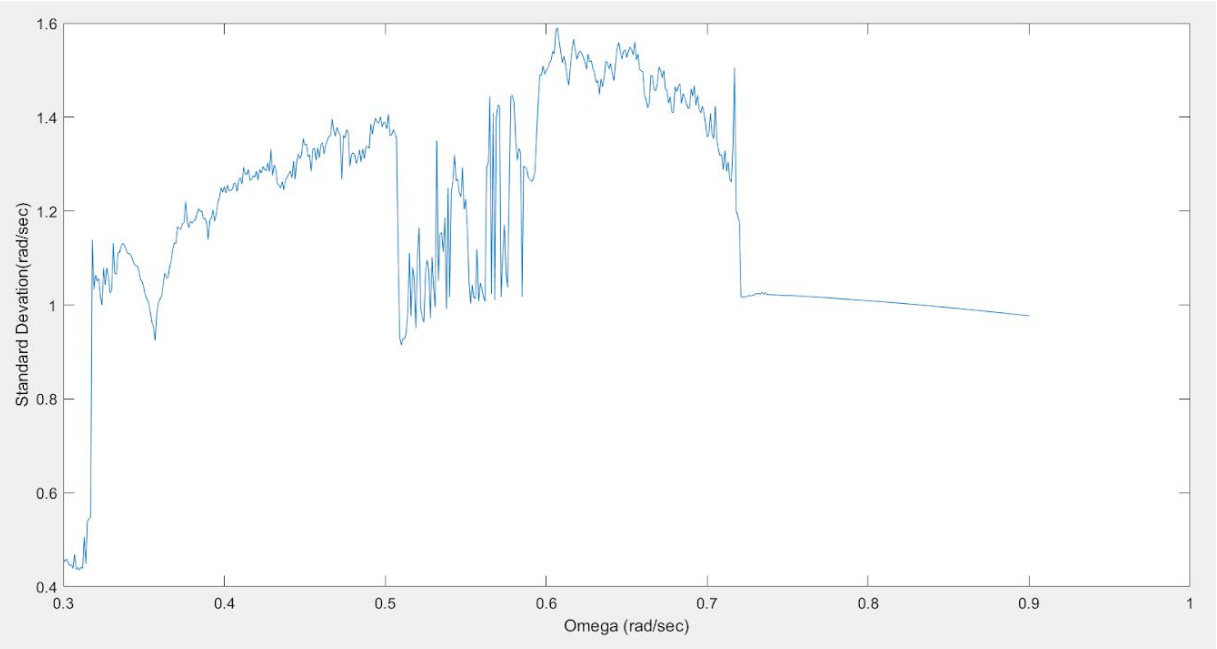

Figure 2.2.3: Standard Deviation Vs ω , from Analysis

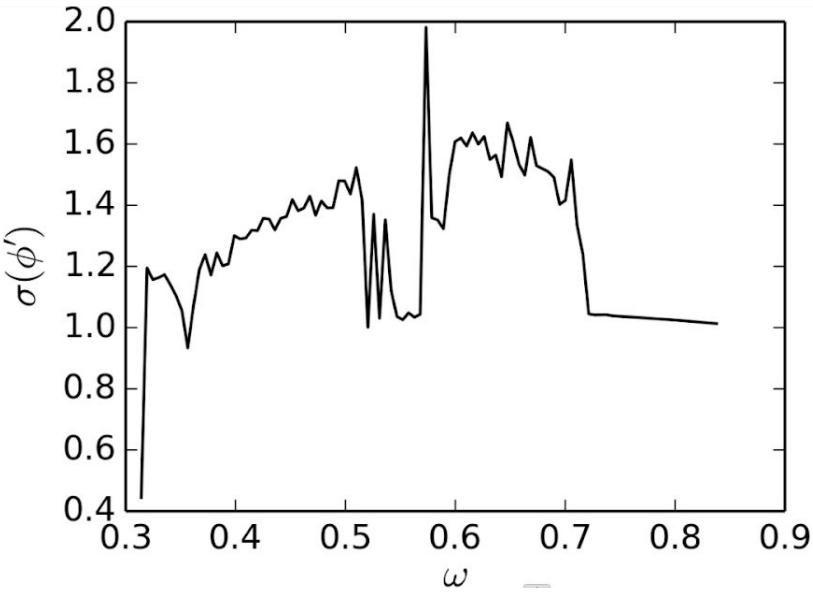

Figure 2.2.4: Standard Deviation Vs ω , from Literature

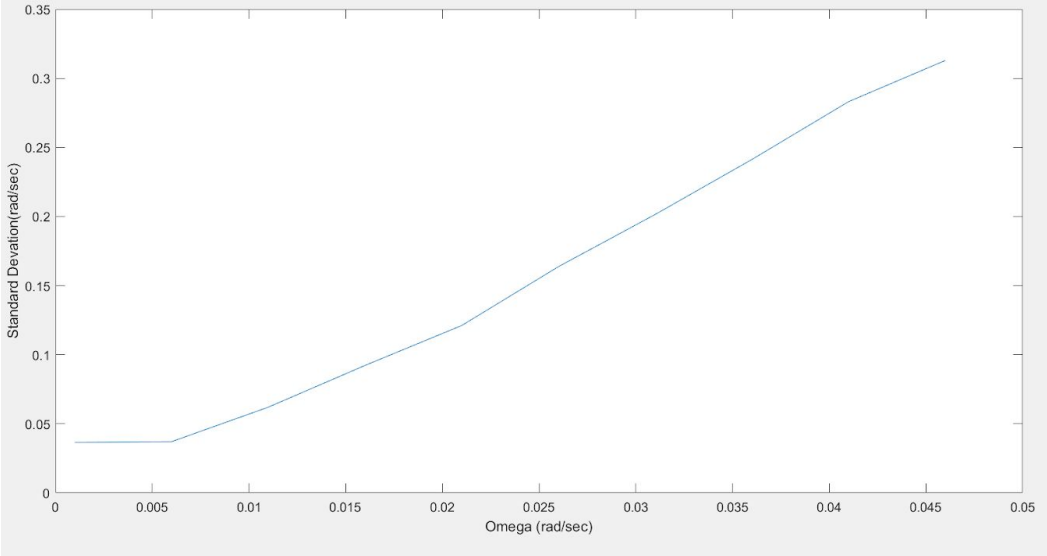

Figure 2.2.5: Standard Deviation Vs ω, for very low ωs

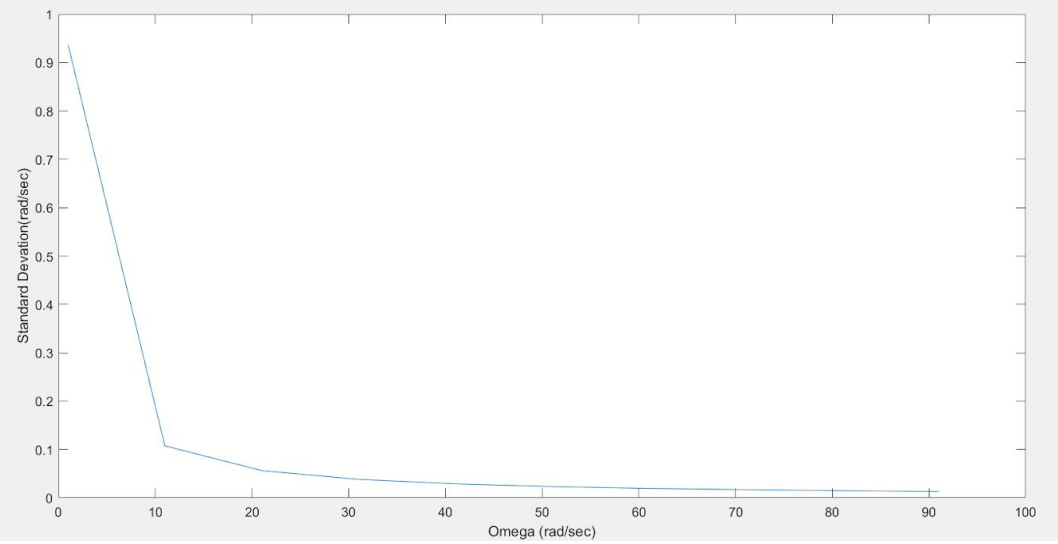

Figure 2.2.6: Standard Deviation Vs ω, for very high ωs

This confirms that the ODE and analysis is working as expected.

3. Optimization

3.1 Surrogate

To develop the surrogate the design space is sampled using Latin Hypercube Sampling for each design variable and plugging them into the objective function. From the convergence study, see section 3.3 Convergence for more information, the optimal number of samples is 2850. To ensure that the whole design space is being explored the samples are shifted, by first

adding each element to its lowest possible value, determined by constraints, and multiplying it by its maximum range of values, the upper bound minus the lower bound. This is quite costly to compute, however it represents the "peaks" and "valleys" the best. The surrogate model is then found using functions in the GPML Matlab library. The covariance function chosen is of the Matérn form and called "covMaterniso", the likelihood function is the Gaussian function "likGauss", initial guesses for the hyperparameters are [log(1); log(.5)] as found by convergence studies, the noise level set is .005 also found via convergence. From there, the hyperparameters are minimized using the GPML function "minimize" and the surrogate function is created using the "gp" function in the GPML library. Since the goal is to maximize the objective function, the inverse of this function is passed into the optimization algorithm.

3.2 Constraints

The constraints are simple upper and lower bound on each design variable. These bounds are shown in *Table 3.2.1.*

Design Variable	*Lower Bound*	*Upper Bound*
α_1 (rad)	0	.3
r_2(m)	.1	1.5
ω (rad/ sec)	$\frac{3 * 2\pi}{60}$	$\frac{8 * 2\pi}{60}$

3.3 Convergence

To determine the sample size, noise level, and initial guess for hyperparameters, convergence studies are conducted. For the sample size study, the hyperparameters are set to [log(1); log(.5)] and the noise is set to 5. The next study looked at the effect of the noise level, 1000 samples are used for computational time and hyperparameters are set to [log(1); log(.5)]. The last study looks at the effect of varying hyperparameter starting points, the noise is set to .005 and 1000 samples are used. For all studies, r_2 and α_1 are set to nominal values.

The results of these studies are shown in *Figure 3.3.1 - 3.3.3.*

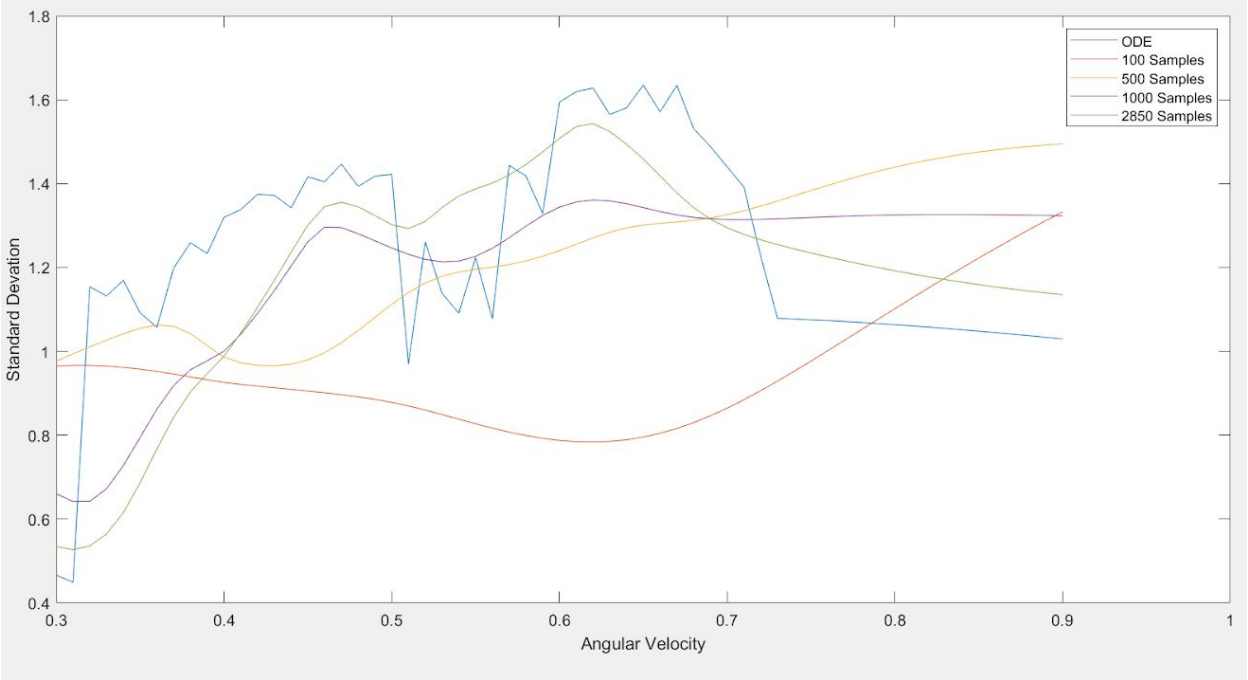

Figure 3.3.1: Convergence study for Sample size

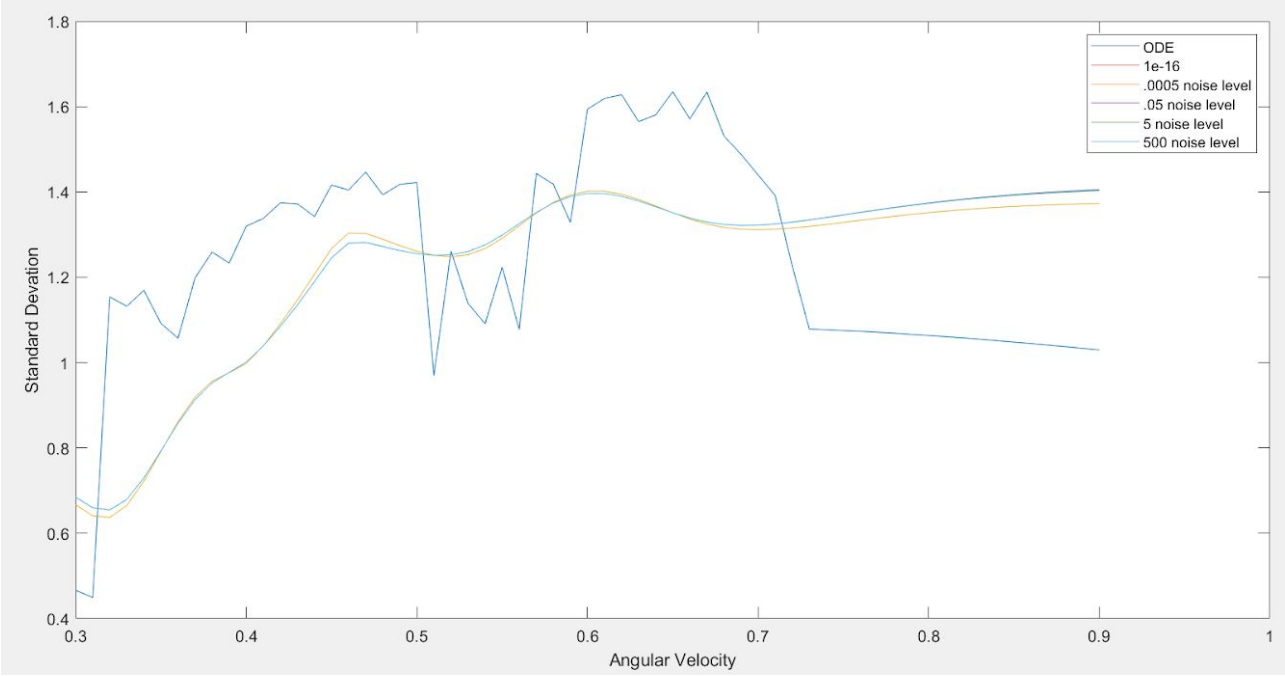

Figure 3.3.1: Convergence study for Noise Level

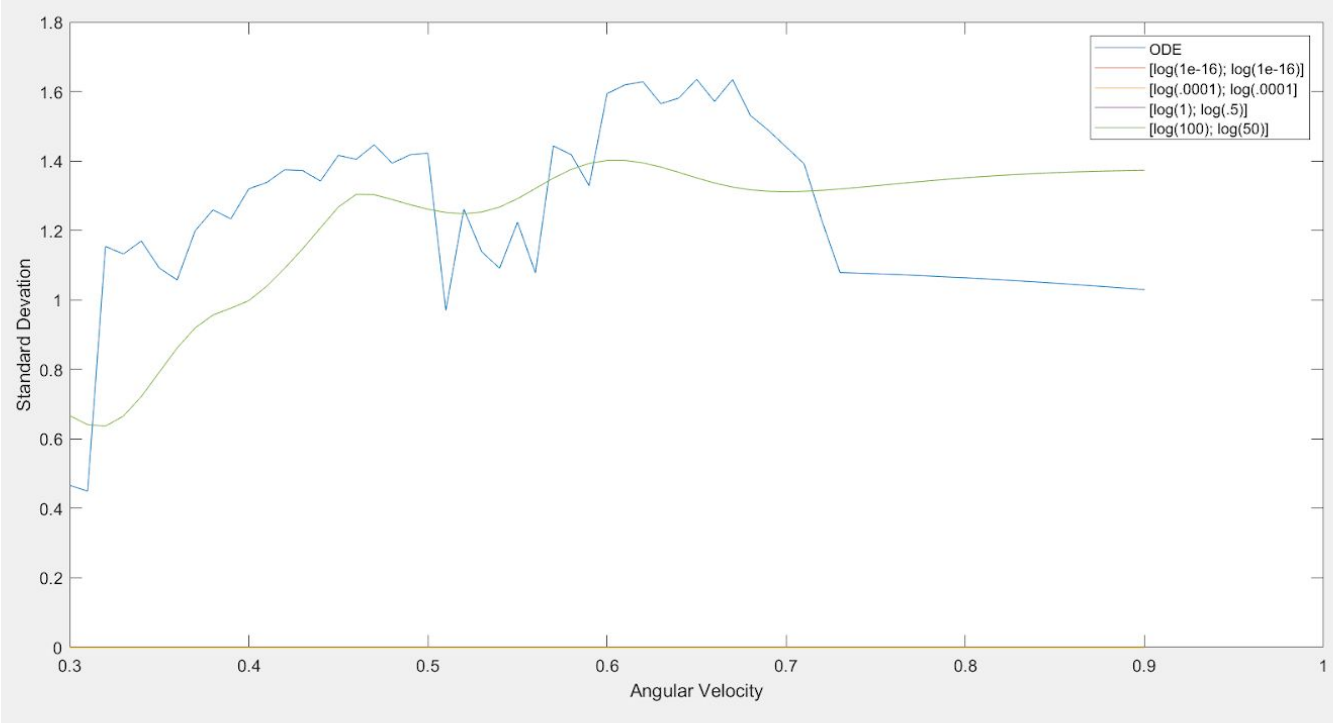

Figure 3.3.1: Convergence study for Hyperparameter Initial Guess

From these studies, it is clear that 1000 samples gives a good estimate for the general trends seen in the but 2850 samples shows them much more clearly and does a better job representing the peaks. It seemed to be fairly inelastic to both noise and initial guess of hyperparameters, so the noise level is set at .005 and the initial guess for hyperparameters is [log(1); log(.5)].

3.4 Optimization Algorithm

The optimization method used is fmincon in MatLab's Optimization Toolbox. This method is used because it can handle constrained optimization problems like the one presented. Within fmincon the algorithms considered were the default, "sqp" and "active-set". The algorithm chosen is "active-set" because it performs much faster than the default and minimal differences are spotted between "sqp" and "active-set" in practice.

There are multiple ways to go about optimizing this problem, due to the noisey and multimodal nature of the objective function and thus the surrogate. The first way to run the optimization many times with random initial guess to increase the likelihood it will converge to a new local maximum, this method will be referred to as random initial condition and is run at 2850 samples. The next method is to run the optimization and set the optimized result as the initial condition, this method will be called the recirculation method and is also run at 2850 samples. A hybrid approach attempts to blend the best aspects of both approaches, both codes are run in the same loop, however the recirculation only happens if the random result is less than the recirculated result, if it's not then the random result becomes the

recirculated and is gradually improved upon. Due to the nature of this algorithm, many iterations are needed, so the sample size is brought down to 1000 to compensate. Finally, a more blunt force approach can be taken, where the number of samples is brought down even further to 800 but the loop iterations are brought up to 30.

4. Results

Best results, found computing the ODE at the optimized parameters from each method are shown in Table 4.1.

Method	ω (rad/sec)	r_2 (m)	α_1 (rad)	Standard deviation(rad/s)
Random Initial condition	0.6846	0.1613	0.2990	4.9031
Recirculation	0.6846	0.1613	0.2990	4.8753
Hybrid	0.6846	0.1613	0.2990	4.9728
Blunt Force	0.6846	0.1613	0.2990	4.8897

Table 4.1: Comparing different optimization methods

5. Conclusions and Discussion of the Results

Each method gave a strikingly similar result, however there is a slight variation in the standard deviation found from the ODE. This is due to the digits after the 4 digits reported. However, since it is highly unlikely and probably cost prohibitive to be able to manufacture to beyond that precision they are not reported. The average from these results is a standard deviation of 4.9102 radians/sec which is an improvement of 222.5% over nominal.

Appendix

A. Works Cited

Fmincon. (2006). Retrieved November 10, 2020, from https://www.mathworks.com/help/optim/ug/fmincon.html

Hicken, J. (2020). Numerical Design Optimization Lecture Slides. Retrieved November 10, 2020

Kautz, R. L., & Huggard, B. M. (1994). Chaos at the amusement park: Dynamics of the Tilt-A-Whirl. American Journal of Physics, 62(1), 59-66. doi:10.1119/1.17742

B. Source code

B.1 Tilt-a-Whirl ODE code

```
function [dx] = ode_(w, r2, a1, y, t, r1)
% is a representation of the equation found in the paper
% inputs:
% w: average angular velocity
% r2
% a1
% y = [y1, y2]
% y1 = phi
% y2 = dphi / dtau
% t: simulation time
% r1
% outputs:
% dx = [d_phi/d_tau, d^2_phi/d_tau^2]

g = 9.81;
Q0 = 20;
a0 = 0.036;
tau = t;
gam = (1/(3*w)) * (g / r2)^.5;
alpha = a0 - a1* cos(tau);
beta = 3 * a1 * sin(tau);
eps = r1 / (9 * r2);
A = gam / Q0;
B = (eps - gam^2 * alpha);
C = gam^2 * beta;
dx = [0,0]';
% create a state vector here
dx(1)= -A * y(1) - B * sin(y(2)) - C*cos(y(2));
dx(2)= y(1);
```

B.2 Test code:

```
% This code is created to test the ODE to ensure it is working
% as expected

clearvars;
close all;
%part one run with nominal values
w = 6.5*(2 * pi /60);
r2 = 0.8;
a1 = 0.058;
T = 10000;
r1 = 4.3;
fun = @(t,y) ode_(w, r2, a1, y, t, r1);
Tau = T;
[t,y_2] = ode45(fun, [0,Tau],[0,0]);
y = y_2(:,1);
y_m = 1/Tau * trapz(t,y);
st = 3* w * (1/Tau * trapz(t,(y - y_m).^2)).^(1/2);
% Plot the ODE to ensure it looks chaotic
xlim([100, 200])
plot(t,y)

% part 2: run a sweep of omega
r2 = 0.8;
a1 = 0.058;
T = 10000;
r1 = 4.3;
% change w_lst to change what range to sweep
w_lst = 1:10:100; % .3:.001:.9
st_ = zeros(length(w_lst),1);
i =1;
for w = w_lst
    fun = @(t,y) ode_(w, r2, a1, y, t, r1);
    Tau = 3 * w * T;
    tspin = .1 * Tau;
    [t,y_2] = ode45(fun, [tspin,Tau],[0,0]);
    y = y_2(:,1);
    y_m = 1/Tau * trapz(t,y);
    st_(i) = 3* w * (1/Tau * trapz(t,(y - y_m).^2)).^(1/2);
    i = i+1;
end
plot(w_lst, st_)
xlabel('Omega (rad/sec)')
ylabel('Standard Deviation(rad/sec)')
```

B.3 st_d:

```
function dev = st_d(t,y_2, Tau, w)
% This function finds the standard deviation of d_phi/d_tau
% inputs:
% t: list of tau values
% y_2: list of  d_phi/d_tau and phi values
% Tau: total non-dimensional time to run for
% w: average angular velocity
% output:
% dev: standard deviation

y = y_2(:,1);
y_m = 1/Tau * trapz(t,y); % this finds the mean of d_phi/d_tau
dev = 3* w * (1/Tau * trapz(t,(y - y_m).^2)).^(1/2);
```

B.4 Samples

```
% this code generates samples and finds the value of the ODE at those
% sample locations

clearvars
close all

samp = 2850;
% use Latin Hypercube to find samples
w_lhs = lhsdesign(samp,1);
% need to scale each set of samples to fit within its bounds and to get the
% full range of bounds
w_lhs = (w_lhs + 3 * 2 *pi/60)*5*2*pi/60;
r2_lhs = lhsdesign(samp,1);
r2_lhs = (r2_lhs + .1) * 1.4;
a1_lhs = lhsdesign(samp,1);
a1_lhs = a1_lhs * .3;
st_dev = zeros(samp,1);
T = 10000;

r1 = 4.3;

for i = 1:samp
    % solve the ode at each set of samples
    fun = @(t,y) ode_(w_lhs(i), r2_lhs(i), a1_lhs(i), y, t, r1);
    Tau = 3 * w_lhs(i) * T;
    Tspin = .1 * Tau;
    [t,y_2] = ode45(fun, [Tspin, Tau],[0,0]);
    y = y_2(:,1);
    y_m = 1/Tau * trapz(t,y);
    st_dev(i) = 3* w_lhs(i) * (1/Tau * trapz(t,(y - y_m).^2)).^(1/2);
end
x = zeros(samp,4);
x(:,1) = w_lhs; x(:,2) = r2_lhs; x(:,3) = a1_lhs; x(:,4) = st_dev;
% save to an excel file
```

```
xlswrite('latin_samples_p3.xlsx',x)
```

B.5 samples convergence

```
% samples convergence
clearvars
close all

mydir = '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'])

w_lhs = xlsread('latin_samples_p3.xlsx',"A1:A2850");
r2_lhs = xlsread('latin_samples_p3.xlsx',"B1:B2850");
a1_lhs = xlsread('latin_samples_p3.xlsx',"C1:C2850");
st_dev = xlsread('latin_samples_p3.xlsx',"D1:D2850");
x = zeros(size(a1_lhs,1),3);
x(:,1) = w_lhs; x(:,2) = r2_lhs; x(:,3) = a1_lhs;
r2 = 0.8;
a1 = 0.058;
r1 = 4.3;
T = 10000;
% set the squared exponential covariance function
covfunc =  {@covMaterniso, 1}; % @covSEiso;
% set the likelihood function to Gaussian
likfunc = @likGauss;
sn = 5; %1e-16; % this is the noise level

hyp_1.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)
hyp_2.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)
hyp_3.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)
hyp_4.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)

hyp_1.lik = log(sn);
hyp_2.lik = log(sn);
hyp_3.lik = log(sn);
hyp_4.lik = log(sn);


% maximize the likelihood function to find the hyperparameters
hyp_1 = minimize(hyp_1, @gp, -100, @infExact, [], covfunc,...
   likfunc, x(1:100,:), st_dev(1:100));
hyp_2 = minimize(hyp_2, @gp, -100, @infExact, [], covfunc,...
   likfunc, x(1:500,:), st_dev(1:500));
hyp_3 = minimize(hyp_3, @gp, -100, @infExact, [], covfunc,...
   likfunc, x(1:1000,:), st_dev(1:1000));
hyp_4 = minimize(hyp_4, @gp, -100, @infExact, [], covfunc,...
```

```
    likfunc, x(1:2850,:), st_dev(1:2850));
% create a different surrogate for each, extract only the samples appropriate
[f_100] = @(z) gp(hyp_1, @infExact, [], covfunc, likfunc, x(1:100,:), st_dev(1:100), z);
[f_500] = @(z) gp(hyp_2, @infExact, [], covfunc, likfunc, x(1:500,:), st_dev(1:500), z);
[f_1000] = @(z) gp(hyp_3, @infExact, [], covfunc, likfunc, x(1:1000,:), st_dev(1:1000), z);
[f_2850] = @(z) gp(hyp_4, @infExact, [], covfunc, likfunc, x(1:2850,:), st_dev(1:2850), z);

iter = .01;
sd_ode = zeros(length(.3:iter:.9),1);
sd_surr_1 = zeros(length(.3:iter:.9),1);
sd_surr_2 = zeros(length(.3:iter:.9),1);
sd_surr_3 = zeros(length(.3:iter:.9),1);
sd_surr_4 = zeros(length(.3:iter:.9),1);

i =1;
for w = .3:iter:.9
    % perform a sweep of the angular velocities
    fun = @(t,y) ode_(w, r2, a1, y, t, r1);
    Tau = 3 * w * T;
    [t,y_2] = ode45(fun, [0,Tau],[0,0]);
    y = y_2(:,1);
    y_m = 1/Tau * trapz(t,y);
    sd_ode(i) = 3* w * (1/Tau * trapz(t,(y - y_m).^2)).^(1/2);
    sd_surr_1(i) = f_100([w, r2, a1]);
    sd_surr_2(i) = f_500([w, r2, a1]);
    sd_surr_3(i) = f_1000([w, r2, a1]);
    sd_surr_4(i) = f_2850([w, r2, a1]);
    i = i+1;
end
% plot
plot(.3:iter:.9, sd_ode, 'DisplayName', 'ODE')
hold on
plot(.3:iter:.9, sd_surr_1, 'DisplayName', '100 Samples')
hold on
plot(.3:iter:.9, sd_surr_2, 'DisplayName', '500 Samples')
hold on
plot(.3:iter:.9, sd_surr_3, 'DisplayName', '1000 Samples')
hold on
plot(.3:iter:.9, sd_surr_4, 'DisplayName', '2850 Samples')
legend
xlabel('Angular Velocity')
ylabel('Standard Deviation')
```

B.6 Noise convergence

```
% This script runs convergence for noise

clearvars
close all
% get the GMPL library
mydir = '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'])
% read samples from sheet
w_lhs = xlsread('latin_samples_p3.xlsx',"A1:A800");
r2_lhs = xlsread('latin_samples_p3.xlsx',"B1:B800");
a1_lhs = xlsread('latin_samples_p3.xlsx',"C1:C800");
st_dev = xlsread('latin_samples_p3.xlsx',"D1:D800");
sd_ode = xlsread('latin_samples_p3.xlsx',"E1:E61");
x = zeros(size(a1_lhs,1),3);
x(:,1) = w_lhs; x(:,2) = r2_lhs; x(:,3) = a1_lhs;
r2 = 0.8;
a1 = 0.058;
r1 = 4.3;
T = 10000;
% set the squared exponential covariance function
covfunc =  {@covMaterniso, 1}; % @covSEiso;
% set the likelihood function to Gaussian
likfunc = @likGauss;
hyp_1.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)
hyp_2.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)
hyp_3.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)
hyp_4.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)
hyp_5.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)

hyp_1.lik = log(1e-16); % this is the noise level
hyp_2.lik = log(.00005);
hyp_3.lik = log(.05);
hyp_4.lik = log(5);
hyp_5.lik = log(50);


% maximize the likelihood function to find the hyperparameters
hyp_1 = minimize(hyp_1, @gp, -100, @infExact, [], covfunc,...
   likfunc, x, st_dev);
hyp_2 = minimize(hyp_2, @gp, -100, @infExact, [], covfunc,...
   likfunc, x, st_dev);
hyp_3 = minimize(hyp_3, @gp, -100, @infExact, [], covfunc,...
   likfunc, x, st_dev);
hyp_4 = minimize(hyp_4, @gp, -100, @infExact, [], covfunc,...
   likfunc, x, st_dev);
hyp_5 = minimize(hyp_5, @gp, -100, @infExact, [], covfunc,...
   likfunc, x, st_dev);
% Find the surrogate models for each
```

```
[f_1] = @(z) gp(hyp_1, @infExact, [], covfunc, likfunc, x, st_dev, z);
[f_2] = @(z) gp(hyp_2, @infExact, [], covfunc, likfunc, x, st_dev, z);
[f_3] = @(z) gp(hyp_3, @infExact, [], covfunc, likfunc, x, st_dev, z);
[f_4] = @(z) gp(hyp_4, @infExact, [], covfunc, likfunc, x, st_dev, z);
[f_5] = @(z) gp(hyp_5, @infExact, [], covfunc, likfunc, x, st_dev, z);


iter = .01;
sd_surr_1 = zeros(length(.3:iter:.9),1);
sd_surr_2 = zeros(length(.3:iter:.9),1);
sd_surr_3 = zeros(length(.3:iter:.9),1);
sd_surr_4 = zeros(length(.3:iter:.9),1);
sd_surr_5 = zeros(length(.3:iter:.9),1);

i =1;
for w = .3:iter:.9
% run a sweep of omega values
   sd_surr_1(i) = f_1([w, r2, a1]);
   sd_surr_2(i) = f_2([w, r2, a1]);
   sd_surr_3(i) = f_3([w, r2, a1]);
   sd_surr_4(i) = f_4([w, r2, a1]);
   sd_surr_5(i) = f_5([w, r2, a1]);
   i = i+1;
end
plot(.3:iter:.9, sd_ode, 'DisplayName', 'ODE')
hold on
plot(.3:iter:.9, sd_surr_1, 'DisplayName', '1e-16')
hold on
plot(.3:iter:.9, sd_surr_2, 'DisplayName', '.0005 noise level')
hold on
plot(.3:iter:.9, sd_surr_3, 'DisplayName', '.05 noise level')
hold on
plot(.3:iter:.9, sd_surr_4, 'DisplayName', '5 noise level')
hold on
plot(.3:iter:.9, sd_surr_5, 'DisplayName', '500 noise level')
legend
xlabel('Angular Velocity')
ylabel('Standard Deviation')
```

B.7 Hyperparamters convergence

```
% Hyperparameters convergance
clearvars
close all

mydir = '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'])
% read samples from sheet

w_lhs = xlsread('latin_samples_p3.xlsx',"A1:A800");
r2_lhs = xlsread('latin_samples_p3.xlsx',"B1:B800");
a1_lhs = xlsread('latin_samples_p3.xlsx',"C1:C800");
st_dev = xlsread('latin_samples_p3.xlsx',"D1:D800");
sd_ode = xlsread('latin_samples_p3.xlsx',"E1:E61");
x = zeros(size(a1_lhs,1),3);
x(:,1) = w_lhs; x(:,2) = r2_lhs; x(:,3) = a1_lhs;
r2 = 0.8;
a1 = 0.058;
r1 = 4.3;
T = 10000;
% set the squared exponential covariance function
covfunc =  {@covMaterniso, 1}; % @covSEiso;
% set the likelihood function to Gaussian
likfunc = @likGauss;
hyp_1.cov = [log(1e-16); log(1e-16)]; % first component is log(l) and second is log(sigma)
hyp_2.cov = [log(.0001); log(.0001)]; % first component is log(l) and second is log(sigma)
hyp_3.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)
hyp_4.cov = [log(100); log(50)]; % first component is log(l) and second is log(sigma)

hyp_1.lik = log(.005); % this is the noise level
hyp_2.lik = log(.005);
hyp_3.lik = log(.005);
hyp_4.lik = log(.005);


% maximize the likelihood function to find the hyperparameters
hyp_1 = minimize(hyp_1, @gp, -100, @infExact, [], covfunc,...
   likfunc, x, st_dev);
hyp_2 = minimize(hyp_2, @gp, -100, @infExact, [], covfunc,...
   likfunc, x, st_dev);
hyp_3 = minimize(hyp_3, @gp, -100, @infExact, [], covfunc,...
   likfunc, x, st_dev);
hyp_4 = minimize(hyp_4, @gp, -100, @infExact, [], covfunc,...
   likfunc, x, st_dev);
% Find the surrogate models for each
[f_1] = @(z) gp(hyp_1, @infExact, [], covfunc, likfunc, x, st_dev, z);
[f_2] = @(z) gp(hyp_2, @infExact, [], covfunc, likfunc, x, st_dev, z);
[f_3] = @(z) gp(hyp_3, @infExact, [], covfunc, likfunc, x, st_dev, z);
[f_4] = @(z) gp(hyp_4, @infExact, [], covfunc, likfunc, x, st_dev, z);


iter = .01;
sd_surr_1 = zeros(length(.3:iter:.9),1);
sd_surr_2 = zeros(length(.3:iter:.9),1);
sd_surr_3 = zeros(length(.3:iter:.9),1);
sd_surr_4 = zeros(length(.3:iter:.9),1);
```

```
sd_surr_5 = zeros(length(.3:iter:.9),1);

i =1;
for w = .3:iter:.9
   % perform a sweep of the angular velocities
   sd_surr_1(i) = f_1([w, r2, a1]);
   sd_surr_2(i) = f_2([w, r2, a1]);
   sd_surr_3(i) = f_3([w, r2, a1]);
   sd_surr_4(i) = f_4([w, r2, a1]);
   i = i+1;
end
%plot
plot(.3:iter:.9, sd_ode, 'DisplayName', 'ODE')
hold on
plot(.3:iter:.9, sd_surr_1, 'DisplayName', '[log(1e-16); log(1e-16)]')
hold on
plot(.3:iter:.9, sd_surr_2, 'DisplayName', '[log(.0001); log(.0001]')
hold on
plot(.3:iter:.9, sd_surr_3, 'DisplayName', '[log(1); log(.5)]')
hold on
plot(.3:iter:.9, sd_surr_4, 'DisplayName', '[log(100); log(50)]')
hold on

xlabel('Angular Velocity')
ylabel('Standard Deviation')
legend
```

B.8 Recirculation and random initial guess

```
% runs the recirculation and the random sample optimization method
clearvars;
close all;
% get the GPML Library
mydir = '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'])
r1 = 4.3;
% dx = ode_(w, r2, a1, y, t, rho, m);
% read the sample data
samp = 2850;
w_lhs = xlsread('latin_samples_p3.xlsx',"A1:A2850");
r2_lhs = xlsread('latin_samples_p3.xlsx',"B1:B2850");
a1_lhs = xlsread('latin_samples_p3.xlsx',"C1:C2850");
st_dev = xlsread('latin_samples_p3.xlsx',"D1:D2850");
T = 10000;
x(:,1) = w_lhs; x(:,2) = r2_lhs; x(:,3) = a1_lhs;
% set the squared exponential covariance function
covfunc = {@covMaterniso, 1};
```

```
hyp.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)

% set the likelihood function to Gaussian
likfunc = @likGauss;
sn = .005; %1e-16; % this is the noise level
hyp.lik = log(sn);

% maximize the likelihood function to find the hyperparameters
hyp = minimize(hyp, @gp, -100, @infExact, [], covfunc,...
    likfunc, x, st_dev);

hyp1 = hyp;
% create two surrogates, one for each method
[f] = @(z) 1/ gp(hyp, @infExact, [], covfunc, likfunc, x, st_dev, z);
[f1] = @(z) 1/ gp(hyp1, @infExact, [], covfunc, likfunc, x, st_dev, z);
% using active- set
options = optimset("algorithm", "active-set"); %,'Display', 'iter');%, "PlotFcn","optimplotfirstorderopt" );
% define bounds
w_l = 3 * 2 * pi / 60; r2_l = .1; a1_l= 0;
w_u = 8 * 2 * pi / 60; r2_u = 1.5; a1_u= .3;
ub = [w_u, r2_u, a1_u];
lb = [w_l, r2_l, a1_l];
s = samp + 1;
max_sd = 0;
max_sd1 = 0;
%random initital condition
z1 = [(rand(1)+ 3 * 2 *pi/60)*5*2*pi/60, (rand(1) + .1) * 1.4, rand(1)*.3];
x1 = x;
st_dev1= st_dev;
for m = 1:6
z0 = [(rand(1)+ 3 * 2 *pi/60)*5*2*pi/60, (rand(1) + .1) * 1.4, rand(1)*.3];
% x = fmincon      (fun,x0,A,b, Aeq,beq,lb,ub, nonlcon,options)
[opt, sd] = fmincon(f, z0,[],[], [], [], lb, ub, [], options);
[opt1, sd1] = fmincon(f1, z1,[],[], [], [], lb, ub, [], options);
% opt = [w, r2, a1]
Tau = 3 * opt(2) * T;
Tspin = .1 * Tau;
% create an annonomus function with only (t,y) as inputs
fun = @(t,y) ode_(opt(1), opt(2), opt(3), y, t, r1);
[t,y_2] = ode45(fun, [Tspin,Tau],[0,0]);

fun1 = @(t,y) ode_(opt1(1), opt1(2), opt1(3), y, t, r1);
[t1,y_21] = ode45(fun1, [Tspin,Tau],[0,0]);

st_ = st_d(t,y_2, Tau, opt(1));
st_1 = st_d(t1,y_21, Tau, opt1(1));

if st_ > max_sd
    % check if theres a new max
    max_sd = st_;
end

if st_1 > max_sd1
    % check if theres a new max
    max_sd1 = st_1;
```

```
end
% append optized parameters and result to the list of samples
x(s,:) =  [opt(1), opt(2), opt(3)];
st_dev(s) = st_;
% recreate the surrogate model
hyp = minimize(hyp, @gp, -100, @infExact, [], covfunc,...
   likfunc, x, st_dev);
[f] = @(z) 1/ gp(hyp, @infExact, [], covfunc, likfunc, x, st_dev, z);
x1(s,:) =  [opt1(1), opt1(2), opt1(3)];
st_dev1(s) = st_1;
hyp1 = minimize(hyp1, @gp, -100, @infExact, [], covfunc,...
   likfunc, x1, st_dev1);
[f1] = @(z) 1/ gp(hyp1, @infExact, [], covfunc, likfunc, x1, st_dev1, z);
% only for recirculation part
z1 = opt;
s = s+1;
end
```

B.9 Hybrid Model

```
% runs the hybrid optimization model
clearvars;
close all;
% get the GPML Library

mydir = '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'])
r1 = 4.3;
% dx = ode_(w, r2, a1, y, t, rho, m);
% read the sample data

samp = 1000;
w_lhs = xlsread('latin_samples_p3.xlsx',"A1:A1000");
r2_lhs = xlsread('latin_samples_p3.xlsx',"B1:B1000");
a1_lhs = xlsread('latin_samples_p3.xlsx',"C1:C1000");
st_dev = xlsread('latin_samples_p3.xlsx',"D1:D1000");
T = 10000;
x(:,1) = w_lhs; x(:,2) = r2_lhs; x(:,3) = a1_lhs;
% set the squared exponential covariance function
covfunc = {@covMaterniso, 1};
hyp.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)

% set the likelihood function to Gaussian
likfunc = @likGauss;
sn = .005; %1e-16; % this is the noise level
hyp.lik = log(sn);
```

```
% maximize the likelihood function to find the hyperparameters
hyp = minimize(hyp, @gp, -100, @infExact, [], covfunc,...
    likfunc, x, st_dev);
hyp1 = hyp;
% create two surrogates, one for each method
[f] = @(z) 1/ gp(hyp, @infExact, [], covfunc, likfunc, x, st_dev, z);
[f1] = @(z) 1/ gp(hyp1, @infExact, [], covfunc, likfunc, x, st_dev, z);
% using active- set
options = optimset("algorithm", "active-set"); %,'Display', 'iter');%,
"PlotFcn","optimplotfirstorderopt" );
% define bounds
w_l = 3 * 2 * pi / 60; r2_l = .1; a1_l= 0;
w_u = 8 * 2 * pi / 60; r2_u = 1.5; a1_u= .3;
ub = [w_u, r2_u, a1_u];
lb = [w_l, r2_l, a1_l];
s = samp + 1;
max_sd = 0;
max_sd1 = 0;
%random inititial condition
z1 = [(rand(1)+ 3 * 2 *pi/60)*5*2*pi/60, (rand(1) + .1) * 1.4, rand(1)*.3];
x1 = x;
st_dev1= st_dev;
for m = 1:10
z0 = [(rand(1)+ 3 * 2 *pi/60)*5*2*pi/60, (rand(1) + .1) * 1.4, rand(1)*.3];
% x = fmincon      (fun,x0,A,b, Aeq,beq,lb,ub, nonlcon,options)
[opt, sd] = fmincon(f, z0,[],[], [], [], lb, ub, [], options);
[opt1, sd1] = fmincon(f1, z1,[],[], [], [], lb, ub, [], options);
% opt = [w, r2, a1]
Tau = 3 * opt(2) * T;
Tspin = .1 * Tau;
fun = @(t,y) ode_(opt(1), opt(2), opt(3), y, t, r1);
[t,y_2] = ode45(fun, [Tspin,Tau],[0,0]);

fun1 = @(t,y) ode_(opt1(1), opt1(2), opt1(3), y, t, r1);
[t1,y_21] = ode45(fun1, [Tspin,Tau],[0,0]);

st_ = st_d(t,y_2, Tau, opt(1));
st_1 = st_d(t1,y_21, Tau, opt1(1));

if st_ > max_sd
    % check if theres a new max
    max_sd = st_;

end

if st_1 > max_sd1
    % check if theres a new max
    max_sd1 = st_1;
end
x(s,:) =  [opt(1), opt(2), opt(3)];
st_dev(s) = st_;
```

```
hyp = minimize(hyp, @gp, -100, @infExact, [], covfunc,...
    likfunc, x, st_dev);
[f] = @(z) 1/ gp(hyp, @infExact, [], covfunc, likfunc, x, st_dev, z);
%
x1(s,:) =  [opt1(1), opt1(2), opt1(3)];
st_dev1(s) = st_1;
hyp1 = minimize(hyp1, @gp, -100, @infExact, [], covfunc,...
    likfunc, x1, st_dev1);
[f1] = @(z) 1/ gp(hyp1, @infExact, [], covfunc, likfunc, x1, st_dev1, z);
if st_ > st_1
    % if the random guess is greater than the gradual increase, set that to
    % new inital guess
    z1 = opt;
else
    %otherwise keep recirculating
    z1 = opt1;
end
s = s+1;
end
```

B.10 Brute force

```
% runs brute force method
clearvars;
close all;

mydir = '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'])
r1 = 4.3;
% dx = ode_(w, r2, a1, y, t, rho, m);
samp = 800;
w_lhs = xlsread('latin_samples_p3.xlsx',"A1:A800");
r2_lhs = xlsread('latin_samples_p3.xlsx',"B1:B800");
a1_lhs = xlsread('latin_samples_p3.xlsx',"C1:C800");
st_dev = xlsread('latin_samples_p3.xlsx',"D1:D800");
T = 10000;
x(:,1) = w_lhs; x(:,2) = r2_lhs; x(:,3) = a1_lhs;
% set the squared exponential covariance function
covfunc = {@covMaterniso, 1};
hyp.cov = [log(1); log(.5)]; % first component is log(l) and second is log(sigma)

% set the likelihood function to Gaussian
likfunc = @likGauss;
sn = .005; %1e-16; % this is the noise level
hyp.lik = log(sn);
```

```
% maximize the likelihood function to find the hyperparameters
hyp = minimize(hyp, @gp, -100, @infExact, [], covfunc,...
    likfunc, x, st_dev);
[f] = @(z) 1/ gp(hyp, @infExact, [], covfunc, likfunc, x, st_dev, z);

options = optimset("algorithm", "active-set"); %,'Display', 'iter');%,
"PlotFcn","optimplotfirstorderopt" );
w_l = 3 * 2 * pi / 60; r2_l = .1; a1_l= 0;
w_u = 8 * 2 * pi / 60; r2_u = 1.5; a1_u= .3;
ub = [w_u, r2_u, a1_u];
lb = [w_l, r2_l, a1_l];
s = samp + 1;
max_sd = 0;
for m = 1:30
z0 = [(rand(1)+ 3 * 2 *pi/60)*5*2*pi/60, (rand(1) + .1) * 1.4, rand(1)*.3];
% x = fmincon      (fun,x0,A,b, Aeq,beq,lb,ub, nonlcon,options)
[opt, sd] = fmincon(f, z0,[],[], [], [], lb, ub, [], options);
% opt = [w, r2, a1]
Tau = 3 * opt(2) * T;
Tspin = .1 * Tau;
fun = @(t,y) ode_(opt(1), opt(2), opt(3), y, t, r1);
[t,y_2] = ode45(fun, [Tspin,Tau],[0,0]);

st_ = st_d(t,y_2, Tau, opt(1));

if st_ > max_sd
    max_sd = st_;
end

x(s,:) =  [opt(1), opt(2), opt(3)];
st_dev(s) = st_;
hyp = minimize(hyp, @gp, -100, @infExact, [], covfunc,...
    likfunc, x, st_dev);
[f] = @(z) 1/ gp(hyp, @infExact, [], covfunc, likfunc, x, st_dev, z);
s = s+1;
end
```