NTNU
Norwegian University of
Science and Technology

Fatigue damage in an offshore wind turbine using probability density evolution

Odd Eiken

Civil and Environmental Engineering
Submission date: June 2017
Supervisor: Ole Andre Øiseth, KT
Co-supervisor: Michael Muskulus, IBM

Norwegian University of Science and Technology
Department of Structural Engineering

Department of Structural Engineering
Faculty of Engineering Science and Technology
NTNU- Norwegian University of Science and Technology

ACCESSIBILITY

OPEN

MASTER THESIS <2017>

SUBJECT AREA: Stochastic Dynamics	DATE:11/06/2017	NO. OF PAGES:127(incl. appendix)

TITLE:

Fatigue damage in an offshore wind turbine using probability density evolution

BY:

Odd Eiken

SUMMARY:
Due to large variability of the offshore environment, the load analysis of an offshore wind turbine is a complex task. It is normally performed in the time domain, by running Monte Carlo simulations. However, this is usually very time consuming due to the necessity of having long time series in order to sample events with low probability of happening. An alternative is to perform the analysis in the frequency domain. However, this method is only valid for linear systems, which makes it rather uncertain for fatigue damage results. This project aims to investigate an alternative approach based on probability evolution methods, which can obtain accurate results for linear, as well as non-linear systems. Nevertheless the methods is not very well known, and a drawback is that it can be numerically and computation challenging especially for high-dimensional problems. The benefits of the method is that it simulates all possible occurrences without generating a long signal in time domain.

This thesis describes the system as a probability density evolution (PDEM) with a cell mapping technique. The system is described as a Markov chain where the state transitions are described with a stochastic matrix. In order to keep all probability in the predefined states, a boundary is created at the edges of the displacement and velocity states. Different load cases have been tested for the method: a totally correlated harmonic load, completely uncorrelated white noise, and partial correlated autoregressive load. Furthermore the Markov chain approach is extended to peak to trough evolution describing a half cycle in a stochastic manner. This means that the damage calculations are based on probability density evolution rather than time domain simulation, and this new approach is therefore compared against already known theory.

The results show that the response probability distributions can be well estimated using the cell mapping technique for the investigated loads, and the Markov chain approach. The added boundaries do not disturb the final distribution, as long as the limits are not set too narrow. The damage intensity is found for a white noise spectrum, and the fatigue damage result seem to fit well with previously developed methods in time domain. This result is promising in terms of PDEMs ability of describing the stochastic system in a more precise and reliable manner.

RESPONSIBLE TEACHER: Professor Michael Muskulus

SUPERVISOR(S)

CARRIED OUT AT: Norwegian University of Science and Technology, Department of Structural Engineering

Summary

Due to large variability of the offshore environment, the load analysis of an offshore wind turbine is a complex task. It is normally performed in the time domain, by running Monte Carlo simulations. However, this is usually very time consuming due to the necessity of having long time series in order to sample events with low probability of happening. An alternative is to perform the analysis in the frequency domain. However, this method is only valid for linear systems, which makes it rather uncertain for fatigue damage results. This project aims to investigate an alternative approach based on probability evolution methods, which can obtain accurate results for linear, as well as non-linear systems. Nevertheless the methods is not very well known, and a drawback is that it can be numerically and computation challenging especially for high-dimensional problems. The benefits of the method is that it simulates all possible occurrences without generating a long signal in time domain.

This thesis describes the system as a probability density evolution (PDEM) with a cell mapping technique. The system is described as a Markov chain where the state transitions are described with a stochastic matrix. In order to keep all probability in the predefined states, a boundary is created at the edges of the displacement and velocity states. Different load cases have been tested for the method: a totally correlated harmonic load, completely uncorrelated white noise, and partial correlated autoregressive load. Furthermore the Markov chain approach is extended to peak to trough evolution describing a half cycle in a stochastic manner. This means that the damage calculations are based on probability density evolution rather than time domain simulation, and this new approach is therefore compared against already known theory.

The results show that the response probability distributions can be well estimated using the cell mapping technique for the investigated loads, and the Markov chain approach. The added boundaries do not disturb the final distribution, as long as the limits are not set too narrow. The damage intensity is found for a white noise spectrum, and the fatigue damage result seem to fit well with previously developed methods in time domain. This result is promising in terms of PDEMs ability of describing the stochastic system in a more precise and reliable manner.

Sammendrag

På grunn av stor variasjon i marine miljø er lastanalysen av en offshore vindturbin en kompleks oppgave. Vanligvis er analysene utført i tidsdomenet, ved å kjøre Monte Carlo simuleringer. Dette er ofte veldig tidkrevende på grunn av at det er nødvendig med en lang tidsserie for å fange opp de hendelsene som har lav sannsynlighet for å skje. Alternativt kan analysen utføres i frekvensdomenet, men er da kun gyldig for lineære system. Dette gjør metoden svært usikker for å regne på utmatting. Denne masteroppgaven tar sikte på en alternativ tilnærming basert på sannsynlighetstetthet evolusjonsmetoden. Denne metoden har vist seg å gi gode resultater for både lineære- og ulineære system. Likevel er denne metoden lite kjent i dag, og en ulempe er at den kan bli utfordrende for systemer med mange dimensjoner og frihetsgrader. Fordelen med metoden er at den beskriver alle mulige hendelser uten å måtte generere et langt signal i tidsdomenet.

Masteroppgaven beskriver systemet med sannsynlighetstetthets evolusjon og cell-mapping metoden. Systemet beskrives som en Markov kjede der overgangen fra en tilstand til en annen beskrives med en stokastisk matrise. For å hindre sannsynligheten fra å forsvinne fra systemet er det satt opp en grense rundt forskyvnings og hastighets domenet. Ulike lastmodeller er blitt testet for å metoden: en fullstendig korrelert harmonisk last, fullstendig ukorrelert hvit støy, og delvis korrelert autoregressiv last. Videre er Markov kjeden utvidet til å beskrive overgangen mellom vendepunktene i forskyvningene (Topp- til bunnpunkt), og beskriver alle mulige halv-sykluser stokastisk. Det betyr at beregningsmodellen er basert på sannsynlighetstetthets evolusjon i stedet for en analyse i tidsdomenet, og disse resultatene sammenlignes derfor med eksisterende modeller.

Resultatene viser at responsfordelingen kan estimeres med cell-mapping for de nevnte lasttilfellene, og Markov kjede antagelsen. Grensene som er satt rundt domenet forstyrrer ikke den endelige fordelingen så lenge grensene ikke er satt for smalt. Halv-syklusene er funnet for en hvit støy last, og den estimerte utmattingsskaden stemmer godt overens med tidligere utviklede modeller i tidsdomenet. Dette resultatet er lovende med tanke på om sannsynlighets tetthets evolusjon kan beskrive det stokastiske systemet på en mer presis og pålitelig måte.

Acknowledgement

I would like to express my gratitude to my thesis supervisor Prof. Michael Muskulus, for providing suggestions, advise, and inspiration during this thesis. Additionally I want to thank PhD candidate Sebastian Schafhirt for help with fatigue calculations, and advise.

I want to thank Quad Geometrics AS for the letting me use their computer in order to test the method developed in this thesis.

Finally, I want to thank Linn T. Selbekk Bjørvig for being patient, and understanding this whole time.

Table of Contents

Summary **i**

Sammendrag **ii**

Acknowledgement **iii**

Table of Contents **vii**

List of Tables **ix**

List of Figures **xiii**

Abbreviations **xiv**

1 Introduction **1**

2 Theory **3**

2.1 Loads 3

2.1.1 White Noise 4

2.1.2 AR - process 4

2.1.3 Harmonic Load 8

2.2 Response 8

2.3 Monte Carlo Simulations 10

2.4 Discrete Fourier transform - Welch Spectrum 11

2.5 Markov Process 12

2.6 Fatigue 12

2.6.1 Narrow Banded process 13

2.6.2 Markov Method 15

3 Methodology **17**

3.1 Numerical solution of the equation of motion 17

3.2 Cell-to-Cell Mapping 19

3.3 General Cell-Mapping 19
3.4 The Transition Probability Matrix 20
3.4.1 Speeding up Python 21
3.5 Restricting Boundaries 22
3.5.1 Steady State Solution 23
3.6 Autoregressive (AR) load model 24
3.6.1 Truncation and Discretization of white noise 25
3.6.2 Modified Discrete Noise 26
3.6.3 AR - force mapping 29
3.7 Harmonic Load 30
3.7.1 Probability distribution of response 30
3.7.2 Harmonic load-mapping 33
3.8 Cycle counting 34
3.8.1 Markov chain aproximation 34
3.8.2 Damage Intensity 35
3.8.3 Damage intensity for Narrow banded process 37

4 Steady State Calculations **39**
4.1 AR(0) load - White Noise 40
4.1.1 Number of Random Mapping Points 41
4.1.2 Time step 43
4.1.3 Boundary Condition 46
4.2 Single Harmonic load 49
4.2.1 Variable time step 50
4.2.2 Different response mesh 54
4.2.3 Different phase size 55
4.3 AR(1) Load 57
4.3.1 Monte Carlo Simulation of response 58
4.3.2 Force Evolution 59
4.3.3 Cell Mapping results 60
4.4 Harmonic load with withe noise 61
4.4.1 Monte Carlo Simulation of response 61
4.4.2 Cell Mapping 62
4.5 AR(2) Load with spectral peak 63
4.5.1 Monte Carlo Simulation of response 65
4.5.2 Force Evolution 65
4.5.3 Cell Mapping Results 66
4.6 Two Harmonic loads 68
4.6.1 Results 69

5 Fatigue Calculations **71**
5.1 White noise Load, narrow banded response 71
5.1.1 Half-cycle iteration 72
5.1.2 Peak Transition Matrix 75
5.1.3 Rainflow Intensities and Damage intensity 76
5.1.4 Comparing with WAFO and Narrow banded approximation . . . 76

6 Discussion **79**
6.1 General Cell Mapping Accuracy and Discretion 79
6.2 Fatigue 81
6.3 Remarks on Computational speed 82

7 Conclusion **83**

Bibliography **84**

Appendix **89**
A Modified noise Force Mapping results 89
B Code: Cell Mapping 90
C Code: Cycle counting 100
D Code: Other 108

List of Tables

4.1 System properties of first modal shape of NREL 5-MW Baseline Wind Turbine . . . 40
4.2 Computer Properties used in analysis . . . 40
4.3 AR(0) Load Model Parameters . . . 41
4.4 Standard Deviation of displacement with Monte-Carlo-Simulations . . . 43
4.5 Standard Deviation of displacement with Cell-Mapping . . . 43
4.6 Range specification . . . 48
4.7 Single Harmonic Load Parameters . . . 49
4.8 time step used, relation to load frequency, and number of phase cells . . . 51
4.9 AR(1) Load Model Parameters . . . 58
4.10 Excact Standard deviation of force and response of described AR(1) load 58
4.11 Standard deviation of AR(1) Load response . . . 61
4.12 Harmonic load with white noise Model Parameters . . . 61
4.13 Standard deviation of harmonic load with white noiseresponse . . . 62
4.14 AR(2) Load Model Parameters . . . 64
4.15 Standard deviation of AR(2) Load response . . . 67
4.16 Two Harmonic Parameters . . . 69

5.1 Fatigue test Model Parameters . . . 71

List of Figures

2.1 Power spectral Density and autocorrelation functions for a white noise process. 4
2.2 Power spectral Density and autocorrelation function for AR(1) load with $\phi = 0.5$, $\sigma_W^2 = 1$, and $\Delta t = 0.1s$. 6
2.3 Power spectral Density and autocorrelation function for AR(2) load with $\phi_1 = 0.75$, $\phi_2 = -0.5$, $\sigma_W^2 = 1$, and $\Delta t = 0.1s$. 7
2.4 Power spectral Density and autocorrelation function for Harmonic load with $\omega_f = 4$ 8
2.5 Sketch of Single Degree of Freedom system 9
2.6 Typical S-N fatigue curve 12
2.7 Rayleigh distribution of peaks for a Gaussian narrow Banded process ($\sigma_x = 1$) 13

3.1 Flow chart of Algorithm used to determine the Probability Transition Matrix 20
3.2 Illustration of a Point mapped outside Range of Cells 21
3.3 Example of how a normal distribution is discretized into 10 discrete values 26
3.4 Probability distribution of random variable T 27
3.5 Different AR(1) force model distributions with standard deviation $\sigma_F = 1$ 28
3.6 Probability distribution of random variable θ 30
3.7 Illustrating of time for which t^* for which $x \leqslant x(t) \leqslant x + dx$ 31
3.8 Probability density function of harmonic steady state response 32
3.9 Relation between velocity and displacement described with an ellipse for harmonic response 32
3.10 Illustrating that a certain displacement will result in 2 possible velocity states 33
3.11 Deterministic and stochastic description of a path 34

4.1 Sketch of a shallow water wind turbine with natural loads 39
4.2 Joint Probability steady state with different number of mapping points in each cell 41

4.3 Probability density plot of displacement and velocity with different number of mapping points in each cell 42
4.4 Probability density plot of displacement and velocity for different time steps 44
4.5 Probability of Image cell mapped to the same state as the starting state . . 45
4.6 Evolution of response for a white noise load starting with a random vector 46
4.7 Evolution of mean and variance of displacement for white noise load . . . 47
4.8 Error vs. number of iterations 47
4.9 Displacement distribution for different boundary ranges 48
4.10 Joint probability plots for different boundary ranges 49
4.11 3D Illustration of Steady State Distribution from harmonic load response . 50
4.12 Harmonic load response distribution for different time steps Δt 52
4.13 Displacement-, and velocity harmonic load response distribution for different time steps compared to exact solution 53
4.14 Harmonic load response distribution for different response discretizations 54
4.15 Displacement-, and velocity harmonic load response distribution for different response discretizations compared to exact solution 55
4.16 Harmonic load response distribution for different phase discretizations . . 56
4.17 Displacement-, and velocity harmonic load response distribution for different phase discretizations compared to exact solution 57
4.18 Exact Spectral Distributions of AR(1) load and response 58
4.19 Spectrum and Distribution from Monte Carlo Simulation of AR(1) load . 59
4.20 AR(1) Force evolution distribution 59
4.21 Displacement, and velocity response of AR(1) laod from Cell Mapping . 60
4.22 Spectrum and Distribution from Monte Carlo Simulation of Harmonic Load with white noise 61
4.23 Join probability distribution for Harmonic Load with White Noise 62
4.24 Displacement and velocity distribution for Harmonic Load with White Noise 63
4.25 Exact Spectral Distributions of AR(2) load and response 64
4.26 Results from 28h Monte Carlo Simulation of Response from AR(2) load . 65
4.27 Steady state AR(2) Force distribution 66
4.28 Joint Distr. of Steady State AR(2) Cell Mapping Response with modified noise 66
4.29 Steady State of AR(2) Cell Mapping Response with modified noise 67
4.30 3D illustration of probability distribution from response from 2 harmonic loads using cell mapping 68
4.31 Steady state joint probability distribution of 2 harmonic loads response . . 69
4.32 Displacement and Velocity probability distribution for 2 harmonic loads response 70

5.1 Response and force signal in time domain from load described in Table 5.1 72
5.2 Result from half cycle iteration with white noise load spectrum, (starting from $x_0 = -2.8m$) 73
5.3 Result from half cycle iteration with white noise load spectrum, (starting from $x_0 = -2.0m$) 74
5.4 *Peak Transition Matrix*: Probability that peak u_i at time k, is followed by peak u_j, at time $k + 1$ 75

5.5 Steady state of Peak Transition Matrix, peak distribution (blue), and Rayleigh Distribution of peaks (red) 75
5.6 Rainflow and Damage Intensity for White Noise Load response 76
5.7 *Damage Intensity* curves for various methods 76

1 Results after 3 iteration of force parameters starting width $\psi = 1.0$ and $\xi = 1.0$ and 9 force states. 89
2 Results after 3 iteration of force parameters starting width $\psi = 1.0$ and $\xi = 1.0$ and 10 force states. 89

Abbreviations

Symbol	=	definition
i	=	variable integer
j	=	variable integer
k	=	(i) Stiffness
		(ii) variable integer
m	=	Mass
$m(x)$	=	mapping of x
n	=	number of discrete values
$\mathbf{p}$	=	Probability vector
u	=	peak level
v	=	trough level
$\|x\|$	=	denotes absolute value of x
$\dot{x}$	=	$\frac{dx}{dt}$
$\ddot{x}$	=	$\frac{d^2x}{dt^2}$
$\{x, y\}$	=	cycle with peak x and trough y
$C(\tau)$	=	Autocovariance function
D(T)	=	Damage function
E[]	=	Expected value of value in square brackets
F	=	Force
$H(\omega)$	=	Frequency Response Function
$\mathbf{I}$	=	Identity Matrix
$N_T(u, v)$	=	Number of crossings with peak larger than u,
		and trough smaller than v for a time length T
P[]	=	Probability of statement in square brackets
$\mathbf{P}$	=	Transition Probability Matrix
$\mathbf{P^p}$	=	Peak Transition Matrix
$S(\omega)$	=	Spectral Density
T	=	Discrete random variable
$\mathbf{Y}$	=	State vector
Var[]	=	Variance of values in square brackets
W	=	Gaussian distributed random variable
α	=	Modified noise constant
β	=	Modified noise constant
$\delta(x)$	=	Dirac delta function
ζ	=	Damping ratio
θ	=	phase variable
λ	=	length / σ
μ_x	=	Mean Value
$\mu^{PT}(u, v)$	=	Intensity of peak and following trough count
$\mu^{RFC}(u, v)$	=	Intensity of rainflow count

ξ	=	modified noise variance parameter
σ_x	=	Standard Deviation
τ	=	Time lag
ψ	=	Modified noise width parameter
ω_n	=	Natural Frequency
ω	=	Load frequency
Γ	=	Gamma function
Δ	=	Sampling interval for discrete variables
Ω	=	Sample space
$\sum$	=	Denotes the summation of

Chapter 1

Introduction

In order to reduce CO_2 emission from fossil fuels, a lot of research and studies have been published on how to meet the energy demand with renewable sources. A study by Ernst & Young [32] expect the growth in offshore wind to triple the capacity (2015) by 2020. However, a few challenges needs to be addressed to make that happen, like the high cost of installation and service.

In Norway, a lot of research and development preceded the offshore structures built for extracting petroleum in the North sea. These platforms were design with high safety factors because high profit margins made it better to be on the safe side with respect to the staff on board and environmental risks. In order to make a cost effective wind farm, more detailed and precise calculation should be carried out in order to reduce overdimensions. A main aspect in offshore engineering is the cyclic stress variations due to environmental loading. With an operating expectancy of 40 years, over 10^8 stress cycles will increase the probability of fatigue cracking and failure. [6]

Today, calculations are done in time domain with Monte Carlo simulations of the response. From these results cycle count methods like the rainflow counting algorithm are applied in order to calculate the expected life time of the structure, and fatigue. The drawback with time simulations, is that in order to have an occurrence with low probability a long time simulation is needed which makes these calculation computational heavy, and time demanding for many load cases. [28]

An alternative is to solve the response in a stochastic manner. One of them is the Probability Density Evolution Method (PDEM) [19] where the time evolution of the probability density function is described as a Partial Differential Equation. This is often used for complicated time simulations with nonlinear stochastic structures [20]. Another technique is the cell-mapping technique which is mainly developed for nonlinear systems [15]. The advantage of the cell mapping method is that the problem is presented as a stochastic matrix instead of a partial differential equation. However, dividing the system into cells makes the system more vulnerable to discretization errors, and one has to understand how to discretize the system.

This thesis has two main focuses. The first is to develop a method for the cell mapping technique with stochastic loads. This is done with a *Transition Probability Matrix* describing the transition from one state to another. Setting a boundary around the displacement and velocity states is a new approach preventing any leakage of probability from the domain of the system. The steady state solution will then be compared against either Monte Carlo simulations or the theoretical distribution in Chapter 4. The second focus is to use the *Transition Probability Matrix* to find the transition between the turning points in displacements, or half-cycles. The *Peak Transition Matrix* describes then this transition between turning points, and with equations derived in Section 3.8 the *Damage intensity* can be obtained by utilising the Palmgren-Miner summation. [7] This is the first time (to my knowledge) that the probability density evolution method is used to calculate fatigue damage, and the result is therefore compared against Monte Carlo simulation with rainflow counting algorithm [5] [26], and the theoretical solution for a narrow banded process derived in Section 3.8.

Chapter 2

Theory

2.1 Loads

For response calculations a representation of the natural random loads have to be stated. Two extremes are totally uncorrelated white noise, and totally correlated harmonic loads. Most realistically the load characteristics are between these two extreme cases. In this thesis an retrogressive AR(0) process is used to represent totally uncorrelated loads, a random phase sine-load is used to model totally correlated loads, while an AR(1) process is used to model partial correlated loads. The autocorrelation is the measure of a signals correlation with it self,

$$R_{FF}(t,s) = E[F(t)F(s)] \tag{2.1}$$

where t and s are two instances in time. For an ergodic time-invariant process, the expression for the autocorrelation simplifies to

$$R_{FF}(\tau) = E[F(t)F(t+\tau)] \tag{2.2}$$

where $\tau = |s - t|$ is the difference between the two points evaluated. The Wiener-Khintchin theorem [23] states that the Fourier transform of the autocorrelation function is the power spectral distribution function.

$$R_{FF}(\tau) \xrightarrow{\mathscr{F}} S_{FF}(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} R_{FF}(\tau)e^{-i\omega\tau}d\tau \tag{2.3}$$

$$S_{FF}(\omega) \xrightarrow{\mathscr{F}^{-1}} R_{FF}(\tau) = \int_{-\infty}^{\infty} S_{FF}(\omega)e^{i\omega\tau}d\omega \tag{2.4}$$

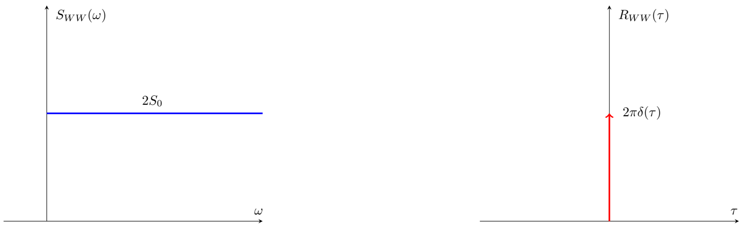
the variance of the variable is integrated over all frequencies,

$$Var[F] = \sigma_F^2 = R_{FF}(0) = \int_{-\infty}^{\infty} S_{FF}(\omega)d\omega \tag{2.5}$$

which means that the power spectral distribution function could be evaluated as the frequency distribution of the random variable. $S_{FF}(\omega)$ is defined for all frequencies, both positive and negative. However, since the spectral distribution is symmetrical for a real time signal, on could therefore only consider the one sided spectrum. The symmetry property of $S_{FF}(\omega)$ is shown in [23], and defines the one sided spectrum:

$$S_{FF}^{+}(\omega) = \begin{cases} 2S_{FF}(\omega), & \omega \geq 0 \\ 0, & \omega < 0 \end{cases} \tag{2.6}$$

2.1.1 White Noise



(a) One sided Spectral Density **(b)** Autocorrelation

Figure 2.1: Power spectral Density and autocorrelation functions for a white noise process.

A white noise process is often defined with a constant spectral density over all frequencies between $\omega = 0 \rightarrow \omega = \infty$. The variance of such a process is infinite large, and the autocorrelation function is derived as a dirac delta function [24].

$$\begin{aligned} S_{FF}^{+}(\omega) &= 2S_0 \\ R_{FF}(\tau) &= 2\pi\delta(\tau) \end{aligned} \tag{2.7}$$

The mathematical concept of such a process is purely theoretical, and in practical sense there is no process defined with a constant power for an infinite number of frequencies. In engineering a white noise process is often constructed with a spectrum with a bandwidth which extends over all frequencies of interest. In structural dynamics and for offshore wind turbines with low eigenfrequencies these frequencies often range between zero and some upper frequency limit. Newland [24] derives the autocorrelation for such a constructed process which defines the spectral density and the autocorrelation as follows,

$$\begin{aligned} S_{FF}^{+}(\omega) &= \begin{cases} 2S_{FF}(\omega), & 0 < \omega < \omega_1 \\ 0, & \text{otherwise} \end{cases} \\ R_{FF}(\tau) &= 2S_0 \frac{\sin(\omega_1 \tau)}{\tau} \end{aligned} \tag{2.8}$$

2.1.2 AR - process

The autoregressive model is defined as a discrete time stationary random process with linear dependency on its own previous values, and a stochastic noise term. Despite its simple definition, it is a powerful class of stochastic models, and is often used to model sea states, and forecast random processes [25] [22]. The model is said to be of order p,

$$F_i = \sum_{n=1}^{p} \phi_n F_{i-n} + W_i \tag{2.9}$$

where ϕ are the parameters describing the linear dependency with the previous values. It is shown in [3] how the values for the parameters must be chosen in order to have a stationary process. The stationary condition is that all roots of the equation

$$m^p - \phi_1 m^{p-1} - ... - \phi_p = 0 \tag{2.10}$$

must lie inside the unit circle. W s a white noise term added to the process as a stochastic term.

AR(1) - process

An autoregressive model of first order ($p = 1$) is also a Markov process, meaning that the forecasting of the next state is only dependent on the current state,

$$\begin{aligned} F_{i+1} &= \phi F_i + W_i \\ &= \phi^n F_0 + \sum_{i=1}^{n} \phi^{i-1} W_{i-1}. \end{aligned} \tag{2.11}$$

The mean value for such a process can be found,

$$\begin{aligned} \mu_F = E[F_{i+1}] &= \phi E[F_i] + E[W_i] \\ &= \phi^n E[F_0] + E[W] \sum_{i=0}^{n} \phi^{i-1} \\ &= \phi^n E[F_0] \underset{n\to\infty}{=} 0, \end{aligned} \tag{2.12}$$

since the stationary condition states that $|\phi| \leq 1$, while the variance,

$$\begin{aligned} \sigma_F^2 = Var[F_{i+1}] &= \phi^2 Var[F_i] + Var[W_i] \\ &= \phi^{2n} Var[F_0] + Var[W] \sum_{i=0}^{n} \phi^{2(i-1)} \\ &\underset{n\to\infty}{=} \frac{\sigma_W^2}{1-\phi^2} \end{aligned} \tag{2.13}$$

The autocorrelation function and spectral density is derived in [13] and is just shown here,

$$\begin{aligned} R_{FF}(\tau) &= \frac{\sigma_W^2}{1-\phi^2} \phi^\tau \\ S_{FF}^{+}(\omega) &= \frac{\sigma_W^2 \Delta t}{\pi[1 - 2\phi\cos(\omega\Delta t) + \phi^2]} \quad \text{for } 0 < \omega < \frac{\pi}{\Delta t} \end{aligned} \tag{2.14}$$

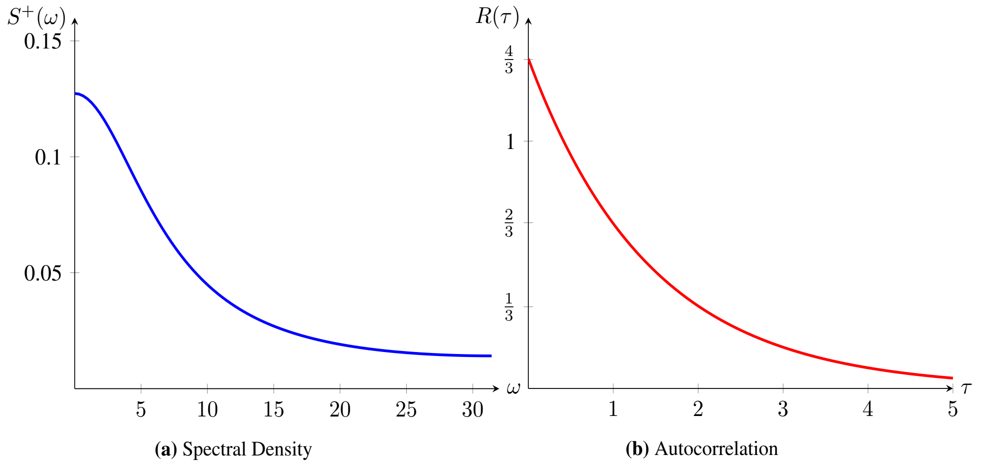


Figure 2.2: Power spectral Density and autocorrelation function for AR(1) load with $\phi = 0.5$, $\sigma_W^2 = 1$, and $\Delta t = 0.1s$.

AR(2) - process

With a second order (p=2) autoregressive model, the forecasting of the next state is linearly dependent on the current, and the previous step. The process represent a discretized second order stochastic linear differential equation, and has two degrees of freedom. That means for instance that the process can oscillate in one frequency.

$$F_i = \phi_1 F_{i-1} + \phi_2 F_{i-2} + W. \tag{2.15}$$

The properties of higher order Autoregressive processes are usually estimated with the *Yule-Walker* equations, [3] and in this thesis the outcome of these parameters are just shown. The referenced literature will show how these parameters are derived. The stationary condition must be satisfied in order to have a stationary, zero mean process.

$$\begin{aligned} &\phi_2 + \phi_1 < 1 \\ &\phi_2 - \phi_1 < 1 \\ &-1 < \phi_2 < 1 \end{aligned} \tag{2.16}$$

The characteristic equation for such a process is given,

$$\phi_2 r^2 + \phi_1 r + 1 = 0 \tag{2.17}$$

if the parameters are chosen in the region $\phi_1^2 + 4\phi_2 < 0$, which leads to complex roots of the characteristic equation, the process will oscillate at one specific frequency. The variance of the process is then given by, [3]

$$\sigma_F^2 = \frac{1-\phi_2}{1+\phi_2}\frac{\sigma_W^2}{(1-\phi_2)^2 - \phi_1^2} \tag{2.18}$$

While, the spectrum is,

$$S^+_{FF}(\omega) = \frac{\sigma_W^2 \Delta t}{\pi[1 + \phi_1^2 + \phi_2^2 - 2\phi_1(1-\phi_2)\cos(\omega\Delta t) - 2\phi_2\cos(2\omega\Delta)]} \qquad (2.19)$$
$$\text{for } 0 < \omega < \frac{\pi}{\Delta t}$$

The autocorrelation function is dependent whether the parameters ϕ_1 and ϕ_2 contains real or complex roots inserted in the characteristic equation 2.17. With real roots the autocorrelation function consists of a mixture of damped exponential, while for complex roots the autocorrelation displays a damped sine wave: [3]

$$R_{FF}(\tau) = \frac{\sigma^2 D^{\left(\frac{\tau}{\Delta t}\right)} \sin(\omega_0 \left(\frac{\tau}{\Delta t}\right) + F)}{\sin(F)} \qquad (2.20)$$

where,

$$D = \sqrt{-\phi_2} \qquad (2.21)$$

$$\omega_0 = \arccos(\frac{\phi_1}{2\sqrt{-\phi_2}}) \qquad (2.22)$$

$$F = \arctan\left(\frac{1+D^2}{1-D^2} tan(\omega_0)\right) \qquad (2.23)$$

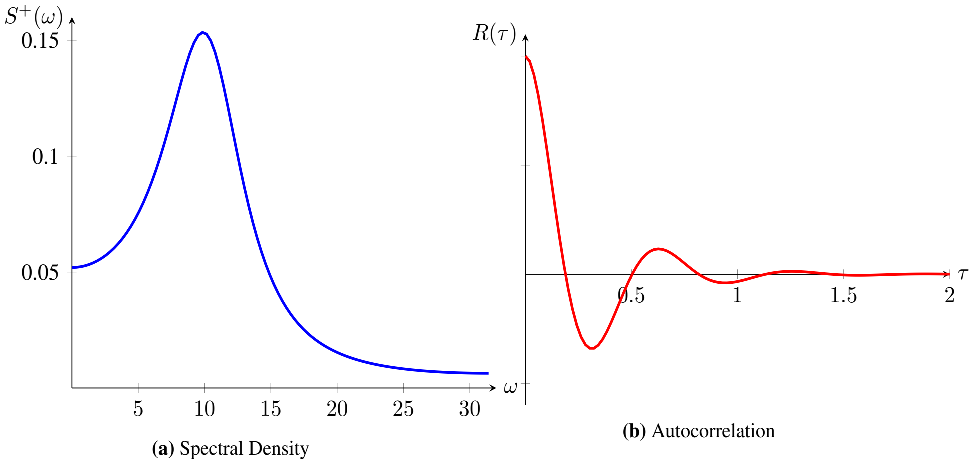


Figure 2.3: Power spectral Density and autocorrelation function for AR(2) load with $\phi_1 = 0.75$, $\phi_2 = -0.5$, $\sigma_W^2 = 1$, and $\Delta t = 0.1s$.

2.1.3 Harmonic Load

A harmonic load can be described with a sin function,

$$F(t) = F_0 \sin(\omega_f t + \theta), \tag{2.24}$$

where θ is a random phase variable with uniform distribution between $0 < \theta \leq 2\pi$. Since the load is repeating itself every $t = \frac{2\pi}{\omega}$, the proces is stationary, and ergodic. The autocorrelation of such a process is found by evaluating,

$$\begin{aligned} E[F(t)F(t+\tau)] &= E[F_0^2 \sin(\omega_f t + \theta)\sin(\omega_f(t+\tau)+\theta)] \\ &= \frac{F_0^2}{2\pi}\int_0^{2\pi} \sin(\omega_f t + \theta)\sin(\omega_f t 0\theta + \omega_f \tau) d\theta, \end{aligned} \tag{2.25}$$

we might rewrite $\sin(\omega_F t + \theta + \omega_f \tau) = \sin(\omega_f t + \theta)\cos(\omega_f \tau) + \cos(\omega_f t + \theta)\sin(\omega_f \tau)$. Evaluation the integral gives,

$$R_{FF}(\tau) = \frac{1}{2}F_0^2 \cos(\omega_f \tau) \tag{2.26}$$

The fourier transform of the autocorrelation function gives the spectral density,

$$S_{FF}^{+}(\omega) = \frac{1}{2}F_0^2 \delta(\omega - \omega_f) \tag{2.27}$$

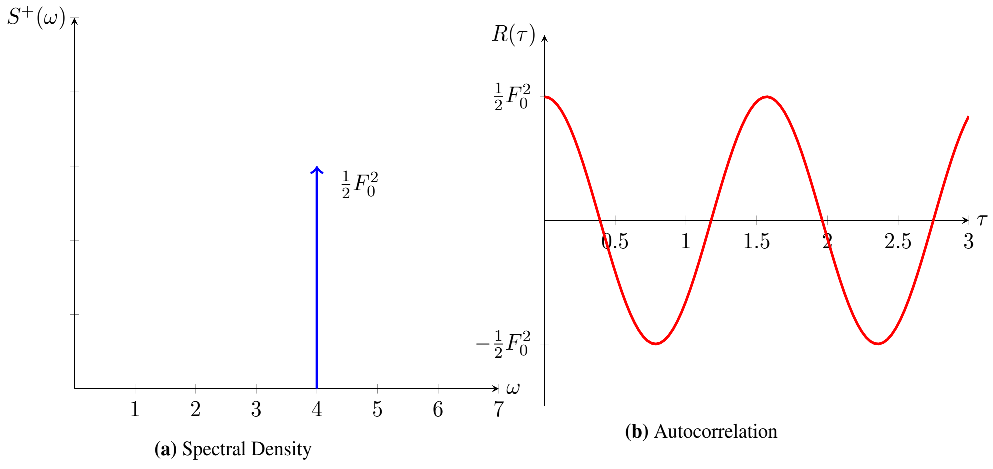


Figure 2.4: Power spectral Density and autocorrelation function for Harmonic load with $\omega_f = 4$

2.2 Response

The Force equilibrium of the system is the described with the forces[9]:

$$R^{ine} + R^{dmp} + R^{int} = R^{ext} \tag{2.28}$$

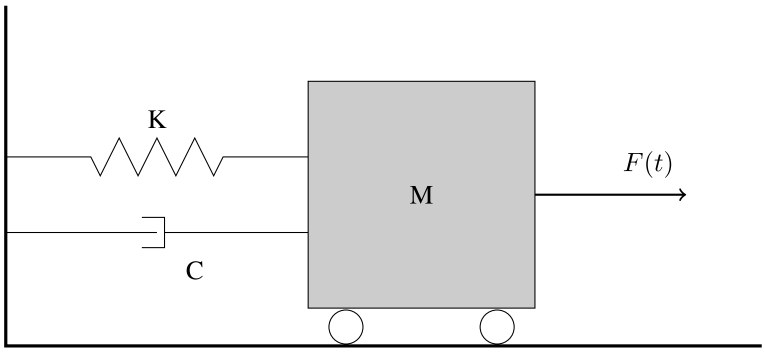


Figure 2.5: Sketch of Single Degree of Freedom system

where *ine, dmp, int,* and *ext* stands for inertial forces, damping forces, internal forces, and external forces. For a linear single degree of freedom system the equation simplifies to this linear second order ordinary differential equation:

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = m^{-1}F(t) \tag{2.29}$$

where the damping ratio ζ, and natural frequeny ω_n has been introduced.

$$\begin{aligned} \zeta &= \frac{c}{2m\omega_n} \\ \omega_n^2 &= \frac{k}{m} \end{aligned} \tag{2.30}$$

introducing a particular solution to the equation of form:

$$x_p(t) = \eta e^{i\omega t} \tag{2.31}$$

inserted in the equation of motion:

$$\begin{aligned} \eta e^{i\omega t}[(i\omega)^2 + 2\zeta\omega_n(i\omega) + \omega_n^2] &= m^{-1}F(t) \\ x(t) &= \frac{m^{-1}}{\omega_n^2 - \omega^2 + 2i\zeta\omega\omega_n}F(t) = H(\omega)F(t) \end{aligned} \tag{2.32}$$

with the frequency response function $H(\omega)$ describing the linear transform between the load and the response. The absolute value of the frequency response function gives the amplification of an amplitude, and the phase shift is given by the relation between the complex part and the real part.

$$\begin{aligned} |H(\omega)| &= \frac{m^{-1}}{\sqrt{(\omega_n^2 - \omega^2)^2 + (2\zeta\omega\omega_n)^2}} \\ \phi &= arctan(\frac{2\zeta\omega_n\omega}{\omega_n^2\omega^2}) \end{aligned} \tag{2.33}$$

Which decouples the complex valued function into a amplitude part, and phase part,

$$H(\omega) = |H(\omega)|e^{i\phi} \tag{2.34}$$

The autocorrelation and the power spectral distribution for the response could therefore be found,

$$\begin{aligned} R_{xx}(\tau) &= E[x(t)x(t+\tau)] \\ &= E[|H(\omega)|F(t)|H(\omega)|F(t+\tau)] \\ &= |H(\omega)|^2 R_{FF}(\tau) \end{aligned} \tag{2.35}$$

$$\begin{aligned} S_{xx}(\omega) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} R_{xx}(\tau)e^{-i\omega\tau}d\tau \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} |H(\omega)|^2 R_{FF}(\tau)e^{-i\omega\tau}d\tau \\ &= |H(\omega)|^2 S_{FF} \end{aligned} \tag{2.36}$$

The first two statistical moments of the response could therefore be expressed with the load mean and spectral distribution, and the frequency response function.

$$\begin{aligned} \mu_x &= E[x(t)] = E[|H(\omega)|F(t)] = |H(\omega)|\mu_F \\ \sigma_x^2 &= R_{xx}(0) = |H(\omega)|^2 R_{FF}(0) = \int_{-\infty}^{\infty} S_{FF}|H(\omega)|^2 d\omega \end{aligned} \tag{2.37}$$

The relation between the velocity and displacement is known, where the velocity is the first derivative of the displacement, and for a harmonic motion, $x(t) = e^{i\omega t}$,

$$\dot{x} = \frac{dx}{dt} = i\omega x. \tag{2.38}$$

The response spectrum for the velocity might therefore be stated,

$$S_{\dot{x}\dot{x}}(\omega) = \omega^2 |H(\omega)|^2 S_{xx} \tag{2.39}$$

The variance of the response can for a white noise Gaussian process be obtained analytically [24].

$$\sigma_x^2 = \int_{-\infty}^{\infty} S_{FF}(\omega)|H_{FX}(\omega)|^2 d\omega = \frac{S_0\omega_n}{8k^2\zeta} \tag{2.40}$$

$$\sigma_{\dot{x}}^2 = \int_{-\infty}^{\infty} S_{FF}(\omega)|i\omega H_{FX}(\omega)|^2 d\omega = \frac{S_0\omega_n^3}{8k^2\zeta} \tag{2.41}$$

2.3 Monte Carlo Simulations

Monte-Carlo-Simulations are widely used when calculating stochastic loads on a structure. To run a Monte carlo analysis, one has to build a mathematical system that simulates a real system. Then a large number of random samples is put in to the model which generates a large number of output results. [28] This is often done in dynamic simulation, where the system is described with a differential equation (Eq.2.29), and where the load is considered a random process. There are different ways to simulate random loads. All of the most common computer languages today have built in algorithm for drawing random numbers from different distributions (Python, Matlab, C++, FORTRAN, etc). One way is with a autoregressive model, where the randomness is added with a random number generator. It is also possible to create a time sample when the spectral density is known,

$$X(t) = \sum_{k=1}^{N/2} A_k \sin(\frac{2\pi}{N}t + \theta) \tag{2.42}$$

where,

$$A_k = \sqrt{\int_{\Delta f(k-\frac{1}{2})}^{\Delta f(k+\frac{1}{2})} S_{FF}(f)df} \tag{2.43}$$

N is the length of time series, t is the time vector, $\Delta t = \frac{1}{N}$, and θ is a random phase between zero and 2π generated from a random number generator. The longer the time sample, more harmonic motion is generated, which is key for a good description of the process.

2.4 Discrete Fourier transform - Welch Spectrum

Generated monte carlo simulations, and response signals are defined in a discrete manner. In order to estimate the spectral density of a discrete signal the Fourier transform has to be established for a discrete case. The Welch spectrum is a common method used for this, and is defined by averaging over multiple samples of the time sample, [29]

$$\hat{P}_{XX}^{(i)}(f) = \frac{1}{NU}\left|\sum_{m=0}^{N-1} w(m)x_i(m)e^{-j2\pi fm}\right|^2 \tag{2.44}$$

where w is a window function used for filtering, N is the number of samples and,

$$U = \frac{1}{U}\sum_{m=0}^{N-1} w^2(m) \tag{2.45}$$

the Welch spectrum is the average of all spectrums,

$$\hat{P}_{XX}^{W}(f) = \frac{1}{K}\sum_{i=0}^{K-1} \hat{P}_{XX}^{(i)}(f) \tag{2.46}$$

In this thesis the hanning window is used for smoothing in all spectral distribution estimation.

2.5 Markov Process

A Markov process has the properties that future values can be predicted based only on the present state. This is often used in probability theory. When the state space is discrete, one often refers to a Markov chain and the transition between states is represented with a transition matrix describing the transition between one state to another.

$$\mathbf{P}_{Transition} = (\mathbf{p}_{ij}) = P[Y_{n+1} = x_i | Y_n = x_j] \tag{2.47}$$

Such a matrix has the properties of a stochastic matrix since the probability of transition from a state x_i to all other states is equal to one. A stationary solution to a stochastic matrix is when a state vector $\mathbf{p} = [P(Y_1), pP(Y_2), ..., P(Y_n)]$ does not change when multiplied with the transition matrix. [4]

$$\mathbf{p} \cdot \mathbf{P} = \mathbf{p} \tag{2.48}$$

which also is the eigenvector associated with an eigenvalue of 1, $\mathbf{pP} = 1\mathbf{p}$.

2.6 Fatigue

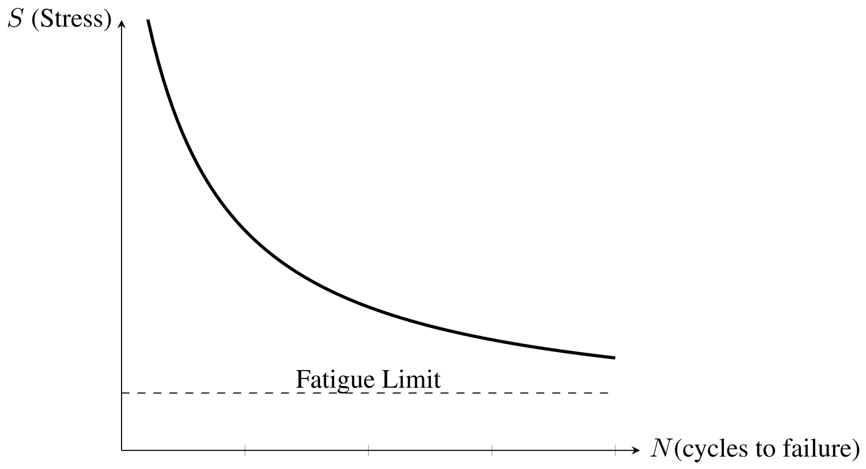


Figure 2.6: Typical S-N fatigue curve

A common way of mechanical failure due to vibrations is fatigue which is caused by an initial crack that gradually propagates under cyclic stresses in the material. This crack grows faster during high range alternating stresses than under, low range alternating stresses. The stresses in a structures are found with the stress strain relation which connects the occurring strains to stresses in the structure with a material law often refereed to as the stress-strain curve. [9] A simplification is to consider a linear stress-strain curve which means that the stresses in the structure are proportional with the displacement. In a steel structure this is an often introduced simplification as long as the stresses are smaller than the yield stress. [9] Larger deformations will cause plastic deformation which decreases the lifetime considerably and could therefore mean that the fatigue failure is never reached. The relation between number of cycles until failure and stress range of a cycle is often presented in a S-N-curve. [24] The S-N curve is found experimentally and describes the number of cycles N, with a cyclic stress range with fixed amplitude S before material failure. However, under random loading with various amplitudes of the stress range S, the mechanism is yet not fully understood [24]. A commonly used damage rule is the Palmgren and Miner hypothesis where the number of cycles at a stress range divided by the number of cycles to failure for that stress range (n_i/N_i) is summed up,

$$D(T) = \sum_i \left(\frac{n_i(T)}{N_i} \right), \tag{2.49}$$

where failure is expected when the damage $D(T)$ exceeds 1. If there is a simple harmonic stress range, the Palmgeen-Miner rule is easy to apply, while for a spectrum load containing more frequencies one would need a counting algorithm of the stress evolution.

2.6.1 Narrow Banded process

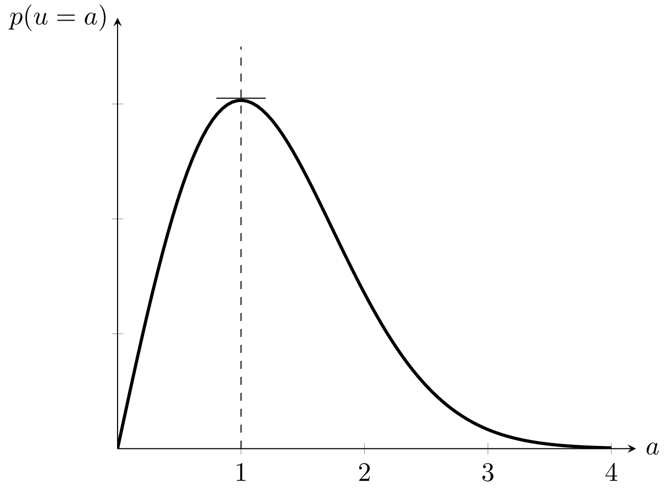


Figure 2.7: Rayleigh distribution of peaks for a Gaussian narrow Banded process ($\sigma_x = 1$)

A narrow banded process defined as a process with only one peak for each zero upcrossing $\nu_x^+(0)$. The probability of a peak with height higher than a level a is then defined as, [23]

$$P[u > a] = \frac{\nu_x^+(a)}{\nu_x^+(0)} \tag{2.50}$$

The cumulative distribution function is then $F_u(a) = 1 - \frac{\nu_x^+(a)}{\nu_x^+(0)}$, so if $\nu_c^+(a)$ can be differentiated with respect to a, the probability distribution of peaks becomes,

$$f_u(a) = -\frac{1}{\nu_x^+(0)} \frac{d\nu_x^+(a)}{da} \quad a \geq 0 \tag{2.51}$$

Where the expected a-crossing rate can be obtained with Rice's formula: [23]

$$\nu_x^+(a) = \int_0^\infty \dot{x} f_{x\dot{x}}(a,\dot{x}) d\dot{x} \tag{2.52}$$

Further on can assume that the process is Gaussian, meaning that the displacement and velocity is normal distributed. Hence the displacement and velocity $(x, \dot{x})$ are independent variables, (proof [13]) one can state the jointly normal distributed variable as,[14]

$$\begin{aligned} f_{x\dot{x}}(x,\dot{x}) &= f_x(x) \cdot f_{\dot{x}}(\dot{x}) \\ &= \frac{1}{2\pi\sigma_x\sigma_{\dot{x}}} e^{-\frac{1}{2}\left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 + \left(\frac{\dot{x}}{\sigma_{\dot{x}}}\right)^2\right]} \end{aligned} \tag{2.53}$$

Combining Eq 2.52 and Eq. 2.53 leads to the Raileigh distribution of peaks, (Fig. 2.7) [24]

$$f_u(a) = \frac{a}{\sigma_x^2} e^{-\frac{a^2}{2\sigma_x^2}} \quad \text{for} \quad a < 0 \tag{2.54}$$

The damage is than the integral evaluated over the probability distribution of the peak amplitudes of the stress and might be evaluated if the S-N curve is available.

$$D(T) = \nu_0^+ T \int_0^\infty \frac{1}{N(S)} p(S_p) dS_p \tag{2.55}$$

For more general loading the rainflow counting is known to yield the most accurate results. However, the procedure of defining a rainflow cycle is not straightforward as in the originally stated form, where the whole load story is required known before the counting process is started [11]. A mathematical stated definition given by Rychlic [26] has made the rainflow counting method convenient for statistical analysis.

In order to use the method a counting distribution has to be defined for a load $F(t), 0 \leq t \leq T$, and the cycles with peak an trough $\{(x,y)_i\}$. The counting distribution is then defined as follows:

$$N_T(u,v) = \#\{(x,y)_i; t_i \leq T \text{ and } x > u \geq v > y\} \tag{2.56}$$

Since the failure is defined when the total damage is 1, so a simple estimator fir the time to failure $\hat{T}$ is defined,

$$\hat{T} = \frac{1}{E[D(1)]} \tag{2.57}$$

and $E[D(1)]$ is defined as the *damage intensity*. While the *counting intensities* of the rainflow count is then defined for a stationary process,

$$\mu^{RFC}(u,v) = E[N_T^{RFC}(u,v)]/T \tag{2.58}$$

The counting intensity is estimated with the counting distribution,

$$\hat{\mu}^{RFC}(u,v) \approx N_T^{RFC}(u,v)/T \tag{2.59}$$

The *Damage intensity* can the be estimated with the following equation, (proof given in [12])

$$\begin{aligned} E[D(1)] = &- \int\int_{u\geq v} \hat{\mu}^{RFC}(u,v)\frac{\partial^2 f(u,v)}{\partial u \partial v} dudv \\ &- \int_{-\infty}^{\infty} \hat{\mu}^{RFC}(u,u)\frac{\partial f(u,u)}{\partial v}|_{u=v} du \end{aligned} \tag{2.60}$$

and $f(u,v)$ is the damage caused by a cycle with peak an trough $\{u,v\}$. A simplification of a real S-N curve is given with this function, [12]

$$f(u,v) = (u-v)^{\beta}, \quad u \geq v, \quad 2 \leq \beta \leq 5. \tag{2.61}$$

and β depends on the material. However, since the goal of this thesis is to show that the method is working, this simplification is introduced.

2.6.2 Markov Method

The peak-trough count could be considered a Markovian process, where the following trough is predicted only based on the current peak, and where the peak is predicted based only on the previous trough. If the intensity, $\mu^{PT}(u,v)$ for a peak trough count is known the rainflow intensity could be obtained through matrix manipulations. The following method is presented in [12],

Let $u_i, u_i > u_{i+1}, i = 1,2,...n$ be discrete levels where a half cycle is formed with a peak and trough $\{x_i, y_i\}$. $\mathbf{P}$ is the transition matrix from peak to the following trough, while $\hat{\mathbf{P}}$ is the transition matrix from the trough to the following peak,

$$\begin{aligned} \mathbf{P} &= (p_{ij}) = P[y_k = u_j | x_k = u_i] \\ \hat{\mathbf{P}} &= (\hat{p}_{ij}) = P[x_k = u_j | y_{k-1} = u_i] \end{aligned} \tag{2.62}$$

The rainflow intensity is then given by,

$$\mu^{RFC}(u,v) = \mu^{PT}(u,v) + \mathbf{qB(I-AB)^{-1}e} \tag{2.63}$$

Where $\mathbf{A}$ and $\mathbf{B}$ are submatrices of $\mathbf{P}$ and $\hat{\mathbf{P}}$ for fixed indices $(i,j), i<j$,

$$\begin{aligned} \mathbf{A} &= (p_{kl}), \quad i \leq k \leq j-1, \quad i+1 \leq l \leq j \\ \mathbf{B} &= (\hat{p}_{kl}), \quad i+1 \leq k \leq j, \quad i \leq l \leq j-1 \end{aligned} \tag{2.64}$$

Where, $\mathbf{A}$ are the probabilities that peaks at $u_i, u_{i+1}, ..., u_{j-1}$ are followed by troughs at $u_i{+}1, ui+2, .., u_j$, while $\mathbf{B}$ are the probabilities that troughs at $u_i{+}1, ui+2, .., u_j$ are followed by peaks at $u_i, u_{i+1}, ..., u_{j-1}$. The vector $\mathbf{p}$ is defined,

$$\mathbf{p} = [p_k] = \sum_{l=j+1}^{n} p_{kl}, \quad k = 1,2,...,n \tag{2.65}$$

and is the conditional probability that a peak with height u_k is followed by a trough smaller than u_j. $\mathbf{e}$ and $\mathbf{q}$ are defined:

$$\mathbf{e} = [p_i, p_{i+1}, ..., p_{j-1}]^T \tag{2.66}$$

$$\mathbf{q} = [\mu^{PT}(u_i, u_{l-1}) - \mu^{PT}(u_i, u_l)], \quad i + 1 \leq l \leq j. \tag{2.67}$$

,

Chapter 3

Methodology

3.1 Numerical solution of the equation of motion

There has been a lot of development in order to solve second order linear differential equations numerically. For instance the central difference method, and the family of Newmark beta methods are widely used in the field of structural dynamics. These methods take advantage of a known time history in order to compute the acceleration and velocities. This is often efficient when the load series is known, but since it relies on the properties at the previous state $\mathbf{Y_{i-1}}$, it should not be necessary to take this into account if the process is Markovian. Instead the Runge-Kutta-Nyström method [18], or other explicit integration schemes should be used when predicting the next state. The benefit of using this method is that it is more memory-effective, while the drawback often is that it is computational more expensive than the Newmark methods. Rewriting the linear equation of motion (Eq.2.29):

$$\ddot{x} = f(p(t), x, \dot{x}) = m^{-1}(F(t) - c\dot{x} - kx) = \omega_n^2 \left(\frac{F(t)}{k} - \frac{2\zeta}{\omega_n}\dot{x}_i - x_i \right) \tag{3.1}$$

where:

$$\omega_n^2 = \frac{k}{m} \qquad \text{and} \qquad \zeta = \frac{c}{c_{crit}} = \frac{c}{2m\omega_n} \tag{3.2}$$

For a second order differential equation the state of the system is described by the displacement x, and the first derivative $\dot{x}$, velocity.

$$\mathbf{Y}(t) = \begin{bmatrix} \dot{x}(t) & x(t) \end{bmatrix}^T \tag{3.3}$$

the state derivative:

$$\dot{\mathbf{Y}}(t) = \mathbf{A}(Y) + \mathbf{B}(t) \tag{3.4}$$

where,

$$\mathbf{A} = \begin{bmatrix} -\omega_n^2(\frac{2\zeta}{\omega_n}\dot{x} + x) \\ \dot{x} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} m^{-1}F(t) \\ 0 \end{bmatrix} \tag{3.5}$$

In a basic euler scheme,[18] the next state is predicted only by evaluation the state derivative at the current state.

$$\mathbf{Y_{i+1}} = \mathbf{Y_i} + \mathbf{\dot{Y}_i}\Delta t \tag{3.6}$$

The Runge-kutta methods however evaluates the equation at different states along the path. The fourth order runge kutta is widely used, and evaluates the differential equation at 4 states along the path, and one has to calculate four auxillary quantities that are used in the Runge-Kutta equation. Higher order Runge-Kutta methods are often neglected since one need more evaluation points than order of convergence.[18] The Runge-Kutta-Nyström method is developed for second order differential equations. The method is an extention of the classical Runge-Kutta method, which reduces the set of evaluation points in half. The Runge-Kutta-Nystom method makes it therefore possible to evaluate second order differential equations with only 4 auxillary quantities instead of eight.
The Runge-Kutta-Nyström method gives the next state:

$$\mathbf{Y_{i+1}} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \mathbf{Y_i} + \frac{1}{3} \begin{bmatrix} \Delta t & \Delta t & \Delta t & 0 \\ 1 & 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{bmatrix} \tag{3.7}$$

the auxiliary quantities k_1, k_2, k_3, k_4 are as follows when introduced int the function of $\ddot{x}$:

$$\begin{aligned} k_1 &= \frac{\Delta t \omega_n^2}{2}\left(\frac{F_i}{k} - \frac{2\zeta}{\omega_n}\dot{x}_i - x_i\right) \\ k_2 &= \frac{\Delta t \omega_n^2}{2}\left(\frac{F_{i+\frac{\Delta t}{2}}}{k} - \frac{2\zeta}{\omega_n}(\dot{x}_i + k_1) - (x_i + K)\right), \quad \text{where} K = \frac{1}{2}\Delta t(\dot{x} + \frac{1}{2}k_1) \\ k_3 &= \frac{\Delta t \omega_n^2}{2}\left(\frac{F_{i+\frac{\Delta t}{2}}}{k} - \frac{2\zeta}{\omega_n}(\dot{x}_i + k_2) - (x_i + K)\right) \\ k_4 &= \frac{\Delta t \omega_n^2}{2}\left(\frac{F_{i+\Delta t}}{k} - \frac{2\zeta}{\omega_n}(\dot{x}_i + 2k_3) - (x_i + L)\right), \quad \text{where} L = \Delta t(\dot{x} + k_3) \end{aligned} \tag{3.8}$$

The load which is a function of time has to be evaluated at 3 different time steps. At the beginning of the integration, at the end, and at the midpoint. For a load case in defined in discrete time, the first and last step would be known. Either explicit in the beginning, or a chosen value from a random distribution at the end. The mid value should then be evaluated by averaging these two forces. For a load case where the force function do not contain any stochastics the mid value could be evaluated with this function. As an example this would be the harmonic load (section 2.1.3) where the stochastic properties are contained in the phase with no evolution in time. A continuous stochastic force evolution is also possible [16]. This would however mean that the Runge Kutta integration scheme has to be reevaluated to get a correct evaluation of this continuous process.

Equation 3.7 is now casted as a point map in discrete time, and continuous space,

$$m(Y_i) = Y_{i+1} \tag{3.9}$$

3.2 Cell-to-Cell Mapping

The easiest and most convenient way to build blocks or cells in an Euclidean N-space is with rectangular hyperrectangle. In R^1 it is a line, in R^2 it is a rectangle, and in R^3 it is a cuboid. When we have defined our sample space $\mathbf{\Omega}$ in $R^N(x_1, x_2, x_3, ..., x_N)$ we can divide it into cells with a size of $\Delta\Omega = \Delta x_1 \cdot \Delta x_2 \cdot \Delta x_3 \cdot ... \cdot \Delta x_N$. Each cell now represents a state in the multidimensional state space. Let $p_i(n)$ denote the probability in state cell i at time $t = n\Delta t$. The vector $\mathbf{p}(x)$ with components $p_i, i = 1, 2, 3, ..., N_{cells}$ such that $\mathbf{p} = [P(y = Y_1), P(y = Y_2), ..., P(y = Y_n)]$. $\mathbf{p}$ is then the cell probability vector and describes the probability of the system being in a given state. From Kolmogorovs second axiom in probability theory the sum of all elements in the probability vector $\sum_{i=0}^{N_cells} p_i = 1$. [17]

When the system is considered a Markov chain [4], the evolution of *probability vector* $\mathbf{p}$ can be found through the *Transition Probability Matrix* $\mathbf{P}$, with the transition probability p_{ij} = *Prob*$\{j$ at n+1$|i$ at n$\}$. The cell mapping can now be described through the evolution with the Transition Probability Matrix.

$$\mathbf{p}(n+1) = \mathbf{p}(n)\mathbf{P} \tag{3.10}$$

Where $\mathbf{p}$ is the row vector containing the state probabilities. Since the Transition Probability Matrix controls the whole evolution:

$$\mathbf{p}(m) = \mathbf{p}(0)\mathbf{P^m} \tag{3.11}$$

As for $m = 2$ is the same as solving equation (Eq 3.10) two times, while for large values of m ($m \to \infty$), the transition matrix $\mathbf{P^m}$ describes the mapping from each transient state to a recurrent state. (steady state).

There are different methods developed in order to calculate the *Transition Probability Matrix*. *Simple Cell Mapping* is described by Hsu [15] and tested on a stochastic offshore structure in a previous master thesis [13]. In simple cell mapping, each cell has only one domain cell where all the probability is mapped to. *General cell mapping* is a more complete method to describe the transition. Each cell might have several domain cells with different probabilities among the cells in the domain. The benefits of simple cell mapping is that it is easier to calculate the transition probability matrix, and storing the matrix does not require a lot of memory. The benefits of using the general cell mapping method is that it describes the mapping in a more complete sense. However, the drawbacks are that it is a more costly calculation, and the storing the transition probability matrix requires more memory.

3.3 General Cell-Mapping

In this thesis the general cell to cell mapping technique is used. For each cell several points are selected and mapped to new cells as suggested by Hsu [15]. There are also other algorithms suggested which could be used for the general cell mapping. The adaptive method [10] divides each cell into sub spaces and maps each subspace into a new cell state depending of the size of the subspace. This is shown to be a more effective method, jet in this thesis I will try to show that the simple method of selecting a number of random points in each cell will yield accurate results.

With the Runge Kutta integration, the state space in order to find the next state of displacement, and velocity is given by the parameters $\mathbf{Y_i} = [x_i, \dot{x}_i, p_i, p_{i+\frac{1}{2}}, p_{i+1}, k, \zeta, \omega_n]$. The first load case is described as an autoregressive model of zero or first order (Section 3.6) where the next load is described with a linear parameter of the previous time step, and a normal distributed random variable. In order to determine the cell mapping, different realisations of the distributed random variable with a certain probability is selected, and the image point of the displacement and velocity is found with the Runge-Kutta-Nyström method. The other load case investigated is a harmonic load with a random phase. In this load case the cell mapping is performed by selecting several random points for the random phase cells.

3.4 The Transition Probability Matrix

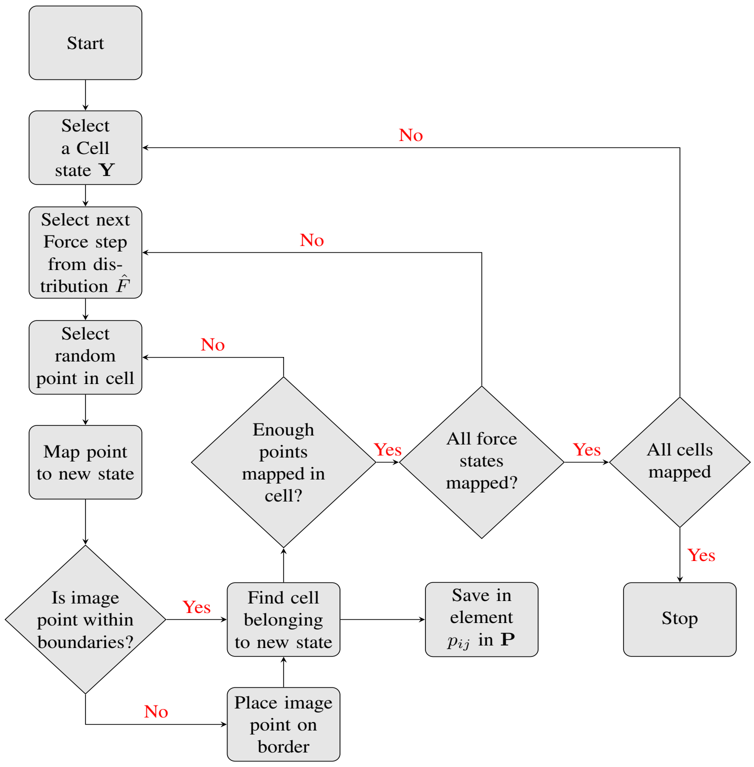

Start
Select a Cell state $\mathbf{Y}$
Select next Force step from distribution $\hat{F}$
Select random point in cell
Map point to new state
Is image point within boundaries?
Yes
No
Find cell belonging to new state
Place image point on border
Save in element p_{ij} in $\mathbf{P}$
Enough points mapped in cell?
No
Yes
All force states mapped?
No
Yes
All cells mapped
No
Yes
Stop

Figure 3.1: Flow chart of Algorithm used to determine the Probability Transition Matrix

The *Transition Probability Matrix* describes how the system transitions from one probability state to the next. By considering the system as a Markov chain, the next state is only dependent by its previous state and the mapping is then described on this matrix with an element p_{ij} is the mapping from state j to state i,

$$P = (p_{ij}) = P[\mathbf{Y^{k+1}} = y_j | \mathbf{Y^k} = y_i] \tag{3.12}$$

This makes it possible to construct the matrix with column vectors where each vector represents which states the cell is mapped to. Usually most of the values in these vectors are zeros, so in order to save the matrix memory efficient it could be stored as a *Row Sparse Matrix* format. That means that the matrix is stored as 3 vectors, one containing all the data, the Column index, and a vector containing location where the data vector start a new Row. This is a compressed way of describing the matrix, and techniques are developed in the scipy.sparce module in order to deal with large matrices. [27] Some of the most essential code developed in this thesis is found in the Appendix.

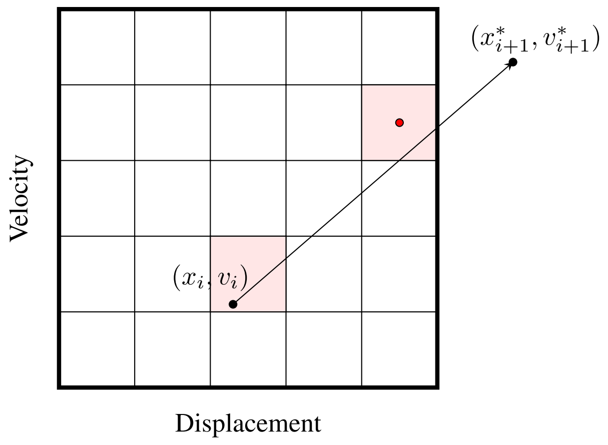


Figure 3.2: Illustration of a Point mapped outside Range of Cells

3.4.1 Speeding up Python

Considering the Flow chart, Figure 3.1, the calculation will in most cases consist of a large number of iterations and cells. Writing the code in Python, which is done in this thesis makes the code easy to read, but in most cases the speed will not be as good as if it was programmed in languages like C++ or Fortran. In order to make the code in python more time efficient the Numba library is implemented to vectorize some of the functions like the Runge-Kutta-Nyström point mapping algorithm. Listing 1 shows how the next states are calculated. The 3 first lines is Numba code which specifies the input as floating numbers in an array of either (n), or (m) length. Prescribing the size of the arrays and data type lets Numba compile the computer code in a more efficient way than regular python code. The time reduction is about 100 times faster when using Numba to compile the code.

```python
@guvectorize(
    'void(f8[:], f8[:], f8[:], f8[:], f8[:], f8[:], f8[:], f8[:])',
    '(m),(n),(n),(n),(n),(n)->(n),(n)',
    target='parallel')
def RKN(par, x, v, F0, F1, F2, xn, vn):
    omega_n, zeta, k, h = par
    K = 0.5 * h * omega_n**2
    for i in xrange(len(x)):
        k1 = K * (F0[i] / k - 2 * zeta / omega_n * v[i] - x[i])
        k2 = K * (
            F1[i] / k -
            2 * zeta / omega_n * (v[i] + k1) -
            (x[i] + 0.5 * h * (v[i] + .5 * k1)))
        k3 = K * (
            F1[i] / k -
            2 * zeta / omega_n * (v[i] + k2) -
            (x[i] + 0.5 * h * (v[i] + .5 * k1)))
        k4 = K * (
            F2[i] / k -
            2 * zeta / omega_n * (v[i] + 2 * k3) -
            (x[i] + h * (v[i] + k3)))
        xn[i] = x[i] + h * (v[i] + (k1 + k2 + k3) / 3.)
        vn[i] = v[i] + (k1 + 2 * k2 + 2 * k3 + k4) / 3.
```

Listing 1: Runge Kutta Nyström code accelerated with numba

3.5 Restricting Boundaries

For a structure with dynamic loads, the steady state solution is often of interest for long term considerations of the system. By discretizing the state space, a problem occurs when the image point from a cell is mapped outside the region, which does not correspond to a cell. If one simply remove these points, the sum of probability in all cells will get smaller and smaller for each iteration, and the steady state solution will be zero for all cells. Often this problem is overcome by selecting a wide range of cells such that the probability of mapping outside the discrete state space is small. Also it is possible to renormalize the probability vector $\mathbf{p}$ such that the sum of probability in all states is equal to 1 after each iteration. However, it is not quite clear if this method is sound to use. The probability leakage from a point mapped outside the range will more often skew the probability plot towards the more extreme values in either displacement or velocity. A renormalization over all states might therefore underestimate the distribution in the extreme values. In this thesis, the problem with probability leakage is overcome by setting boundaries around the outermost cells. Image points which would be mapped outside the discretized space will be placed in a cell along the border which is determined by a linear path between the starting point and image point. Figure 3.2 shows the cell mapped from, and image cell in red when a point is mapped outside the range of cells. This will introduce truncation errors at the boundaries, which will be investigated in this thesis. The boundary condition applied when a image point falls outside the discretized domain is stated as with (x^*_{i+1}, v^*_{i+1}) as the point mapped outside, and (x_{i+1}, v_{i+1}) as the spurious image points mapped inside the sample space $\mathbf{\Omega}$.

$$L = max \begin{cases} H(|x^*_{i+1} - x_{lim}|)\frac{|x^*_{i+1}-x_{lim}|}{|x^*_{i+1}-x_i|} \\ H(|v^*_{i+1} - v_{lim}|)\frac{|v^*_{i+1}-v_{lim}|}{|v^*_{i+1}-v_i|} \end{cases} \tag{3.13}$$

$$\begin{aligned} x_{i+1} &= x_i + (x^*_{i+1} - x_i) \cdot (1 - L) \\ v_{i+1} &= v_i + (v^*_{i+1} - v_i) \cdot (1 - L) \end{aligned} \tag{3.14}$$

where H is the heavside step function defined,

$$H(x) = \begin{cases} 1 & \text{for } x \geq 1 \\ 0 & \text{for } x < 0 \end{cases} \tag{3.15}$$

Algorithm 1 Simple iteration Algorithm

```
 1: Inputs:
        $-\mathbf{P}$
        $-\mathbf{p}_0$
        – tolerance
        – max iterations
 2: Initialize:
        $i = 0$
 3: for max iterations do
 4:     $\mathbf{p}_{i+1} = \mathbf{P} \cdot \mathbf{p}_i$
 5:     err = $\sum_{n=1}^{N} |p_{i+1,n} - p_{i,n}|$
 6:     $i$+=1
 7:     if err $\leq$ tolerance then
 8:         break
 9:     end if
10: end for
```

3.5.1 Steady State Solution

The steady state solution occurs when the transient behaviour vanishes. For a dynamical system with initial condition this happens when the homogeneous solution to the differential equation (Eq. 2.29) is zero and happens only as $t \to \infty$, since the homogeneous solution is defined [8],

$$x_h(t) = e^{-\zeta\omega_n t}(A\sin(\omega_D t) + B\cos(\omega_D t)) \tag{3.16}$$

where $\sqrt{A+B}e^{-\zeta\omega_n t}$ is the maximum amplitude for a time t. Specifying a tolerance value ϵ measuring the amplitude of the signal at a time t

$$\frac{x_h(t)}{x_h(0)} = e^{-\zeta\omega_n t_\epsilon} \leq \epsilon \tag{3.17}$$

$$t_\epsilon = \frac{-\ln(\epsilon)}{\zeta\omega_n} \tag{3.18}$$

The time t_ϵ is then the time until the transient behaviour reaches a level ϵ times the initial amplitude, and for small values of ϵ this would give an indication on how long time before reaching steady state. This is used when Monte Carlo Simulations are carried out for the response in order to compare with the cell mapping results.

Algorithm 1 shows how a steady state solution can be iterated from an initial state vector $\mathbf{p}_0$. When the system reaches a steady state the difference between state $\mathbf{p}_i$ and the next state $\mathbf{p}_{i+1}$ should go towards zero.

Another method of finding the steady state solution to the mapping problem is by considering the eigenvalue problem,

$$\mathbf{pP} = \lambda\mathbf{p}, \tag{3.19}$$

When solving for the eigenvector, an algorithm for solving only the largest eigenvalue is used, since this would usually lead to the case where the eigenvalue is 1. The largest eigenvalue for a stochastic matrix is always 1 according to the Perron-Frobenius theorem.[2] In this thesis this is done by implementing the ARPACK software [31] which uses the Arnoldi Method of finding the larges eigenvalue which is the preferred method of solving large sparse matrix eigenvalues [1].

3.6 Autoregressive (AR) load model

The white noise process is defined as a continuous stochastic process. In order to use the Runge Kutta Nyström method and obtain fourth order accuracy in the time stepping of the equation of motion (Eq: 2.29), the load has to be known at the current time step, the next time step, and a time step in between the two. In order to take discrete samples in time a new variable F is defined as the average over an integrated path of the white noise process W.

$$F\Delta t = \int_{t-\frac{\Delta t}{2}}^{t+\frac{\Delta t}{2}} W(t)dt \tag{3.20}$$

the mean:

$$\mu_F = E[F] = E\left(\frac{1}{\Delta t}\int_{t-\frac{\Delta t}{2}}^{t+\frac{\Delta t}{2}} W(t)dt\right) = \frac{1}{\Delta t}\int_{t-\frac{\Delta t}{2}}^{t+\frac{\Delta t}{2}} E[W(t)]dt = \mu_W = 0 \tag{3.21}$$

and variance:

$$\begin{aligned}\sigma_F^2 = E[F^2] - E^2[F] &= E\left(\frac{1}{\Delta t}\int_{t-\frac{\Delta t}{2}}^{t+\frac{\Delta t}{2}} W(t_1)dt_1 \frac{1}{\Delta t}\int_{t-\frac{\Delta t}{2}}^{t+\frac{\Delta t}{2}} W(t_2)dt_2\right) \\ &= \frac{1}{\Delta t^2}\int_{t-\frac{\Delta t}{2}}^{t+\frac{\Delta t}{2}}\int_{t-\frac{\Delta t}{2}}^{t+\frac{\Delta t}{2}} E[W(t_1)W(t_2)]dt_1 dt_2\end{aligned} \tag{3.22}$$

Since the variable W is time-invariant, the expected value of the two variables can be expressed with the autocorrelation: $E[W(t_1)W(t_2)] = R_{WW}(\tau)$. Since the process also is a zero-mean process, the variance could be obtained by the following expression, [21]

$$\begin{aligned}\sigma_F^2 &= \frac{1}{\Delta t^2}\int_{-\Delta t}^{\Delta t} [\Delta t - |\tau|]R_{WW}(\tau)d\tau \\ &= \frac{1}{\Delta t^2}\int_{-\Delta t}^{\Delta t} [\Delta t - |\tau|]2\pi S_0 \delta(\tau)d\tau \\ &= \frac{2\pi S_0}{\Delta t}\end{aligned} \tag{3.23}$$

The continuous White Noise process can now be modelled as a discrete Gaussian Random Process where the force realisations are independent samples characterised by their mean and variance. However, the variance of the signal and response would not be equal as for the expressions shown in (Eq 2.40) and (Eq 2.41). Due to anti aliasing in a discrete signal, the power spectra would only be defined for frequencies smaller than the nyquist frequency f_n [29]. In order to find the response variance the integral in (Eq 2.40) and (Eq 2.41) should be evaluated with the followin limits,

$$\sigma_Y^2 = \int_0^{\omega_{Ny}} S_0|H_{FX}(\omega)|^2 d\omega \tag{3.24}$$

$$\sigma_{\dot{Y}}^2 = \int_0^{\omega_{Ny}} S_0|i\omega H_{FX}(\omega)|^2 d\omega \tag{3.25}$$

and could be evaluated numerically.

3.6.1 Truncation and Discretization of white noise

Truncation

The noise term associated with the AR models is a normal distributed noise term. In monte carlo simulations, this noise is simulated by picking a random value weighed by the normal distribution curve. Since the cell mapping method divides the force range into a finite number of states, the added noise must be truncated at the force limits,

$$F \in \left[-\frac{n_f \Delta F}{2}, \frac{n_f \Delta F}{2}\right], \tag{3.26}$$

where n_f is the number of cells, and ΔF the cell size. Values are then only considered inside these limits, and neglected outside. In order to preserve the probability of the random variable F, the variable is renormalised. This will distort the normal distribution, so the limits of the truncation should be large enough to contain most of the distribution.

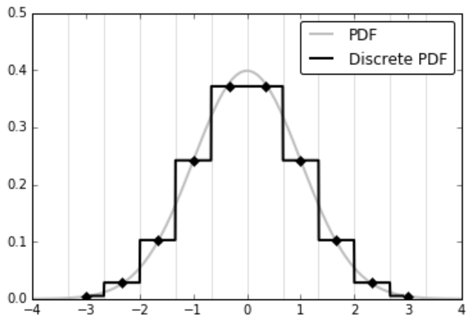


Figure 3.3: Example of how a normal distribution is discretized into 10 discrete values

Discretization

The truncated discrete continuous variable should be discretized into discrete values. Each discrete value creates a image point from a cell state, so fewer values would mean faster computation time. The size should however be large enough to represent a normal distribution. Let $\hat{F}$ be the discrete variable with n_f states. The probability distribution of the discrete variable is then given by the cumulative distribution of the continuous variable,

The probability of X being between the upper and lower limit of the cell (F_U and F_L) is given by the cumulative distribution of a normal random variable:

$$\begin{aligned} P(f = \hat{F_i}) &= P(f \leq F(\hat{F_i} + \frac{\Delta F}{2})) - P(f \leq F(\hat{F_i} - \frac{\Delta F}{2})) \\ &= \mathbf{F}(\hat{F_i} + \frac{\Delta F}{2})) - \mathbf{F}(\hat{F_i} - \frac{\Delta F}{2}) \end{aligned} \tag{3.27}$$

3.6.2 Modified Discrete Noise

The difference between the AR(1) model, and the AR(0) model used to model white noise, is the correlation with the previous force step $\phi \neq 0$. The correlation changes the statistical properties of the load, $F \neq W$. The discretization of the load parameter F could therefore differ from the random variable W, as opposed to the AR(0) process. The Central Limit Theorem [30] establishes that when statistical independent random variables with finite variance are added, the sum tends toward a normal distribution. For $\phi > 0$ the force F could represent such a summation, though it may not be significant for small values of ϕ. For larger values of ϕ it might be tempting to change the random variable W to a defined discrete variable containing only a small number of values since this will decrease the computation time.

let the random variable T be defined for 3 values,

$$\mathbf{T} = \beta[-1, 0, 1] \tag{3.28}$$

where β is the width of the variable,

$$\mathbf{P}(T) = \frac{1}{2+\alpha}[1, \alpha, 1] \tag{3.29}$$

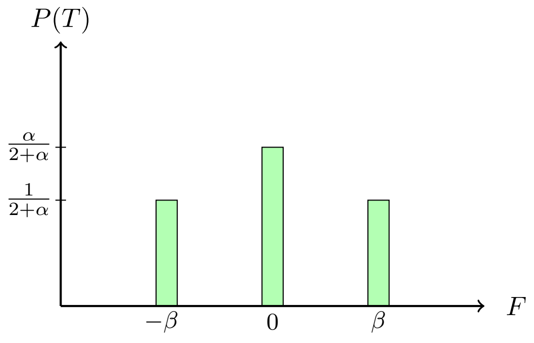


Figure 3.4: Probability distribution of random variable T

The first two moments are found as follows,

$$\begin{aligned} E[T] &= \mathbf{T} \cdot \mathbf{P}(T)^{\mathbf{T}} = 0 \\ Var[T] &= \mathbf{T^2} \cdot \mathbf{P}(T)^{\mathbf{T}} = \frac{2\beta^2}{2+\alpha} \end{aligned} \tag{3.30}$$

The constants α, and β must be determined. The magnitude of the force is described by the width β, and is now defined as a function of the force cell size,

$$\beta = \psi \Delta F \tag{3.31}$$

where ψ is the parameter describing the with in terms of force cell size. This value should of course not be less than one which would result in the force mapping back to the same cell. Large values compared to the force discretization size should also be avoided. In order to get a correct distribution of the force, the statistical properties must be kept in the new variable T. Which is zero for the mean, while the variance,

$$\begin{aligned} \frac{2\beta^2}{2+\alpha} &= \sigma_W^2 \\ \frac{2(\psi\Delta F)^2}{2+\alpha} &= \sigma_W^2 \\ \alpha = 2\left(\frac{\psi\Delta F}{\sigma_W}\right)^2 &- 2 \end{aligned} \tag{3.32}$$

The Force stated is discretized into n bins, with a range between $[-\lambda\sigma_F, \lambda\sigma_F]$, so the cell size is expressed in terms of n and λ,

$$\Delta F = \frac{2\lambda\sigma_F}{n} \tag{3.33}$$

For a AR(1) force, the relation between the variance of the force, and the variance of the added white noise is given in Eq. 2.13. The cell size ΔF can therefore be expressed in terms of the added noise variance,

$$\Delta F = \frac{2\lambda\sigma_W}{n\sqrt{1-\phi^2}} \tag{3.34}$$

Which lets us define our constants α and β in such way,

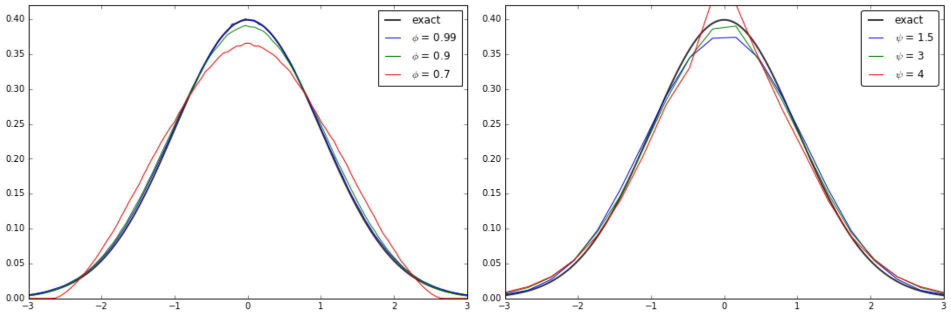
$$\begin{aligned}\alpha &= \frac{8\psi^2\lambda^2}{n^2(1-\phi^2)} - 2 \\ \beta &= \psi\Delta F = \frac{2\psi\lambda\sigma_W}{n\sqrt{1-\phi^2}} = \sigma_W\sqrt{\alpha/2+1}\end{aligned} \tag{3.35}$$

$\alpha > 0$ is also required to have the probability positive defined, so the relation in the Force state discretization,

$$\frac{\lambda}{n} > \frac{1}{2}\frac{\sqrt{1-\phi^2}}{\psi} \tag{3.36}$$

The AR(1) process could now be realised with the change of random noise,

$$F_{i+1} = \phi F_i + T_i \tag{3.37}$$



(a) MC sim. in continuous Force space **(b)** Cell Mapping in discrete force space, $\phi = 0.9$

Figure 3.5: Different AR(1) force model distributions with standard deviation $\sigma_F = 1$

Which value ψ should take is not obvious and Figure 3.5 shows that the solution is highly dependent on the value chosen. The optimal value is therefore found through testing, where the cell space only consist of force cells. For an autoregressive load case this would be creating a cell space of $n-2$ dimensions, where n being the number of dimensions including displacement and velocity. This is a small problem compared with the whole model which makes it possible to test for several realisations of ψ withing reasonable time.

There is often a goal to discretize the force into a coarse mesh. A detailed representation of the displacement and velocity is in general more important than the details in the force representation. A coarse mesh will more often however result in an error in the final force distribution. The variance of the random noise added to the AR model could then be modified to accommodate these discretization errors in the final force distribution. A possibility is to modify the probability distribution parameter,

$$\hat{\alpha} = \alpha\xi, \tag{3.38}$$

where ξ is another scalar found empirical through force mapping. Algorithm 2 shows how these parameters are estimated in this thesis. The first set of parameters is chosen by guessing. ψ is then chosen after iterating through different values, and chosen based on how well the force distribution fits with the expected distribution. When a suitable value for ψ is chosen, different realisations for the parameter ξ is tried. The best fit is here chosen based on how well the variation fits compared to the known variation. These parameters are therefore chosen based on different criteria. ψ is chosen based on how well the shape fits, while ξ is chosen based on how well the variance of the distribution fits.

The modified distribution parameter $\hat{\alpha}$ changes the variance of the added noise. For ξ larger that 1, the added noise decreases, while for values smaller than 1 it decreases. This value is chosen to accommodate for the changes in variance due to the errors introduced when the force is discretized into cells.

Algorithm 2 Find force parameters

```
 1: **Inputs:**
        $\lambda$, $\phi$, $\sigma_F$
 2: **Initialize:**
        $\psi_0 = \min\{1, \frac{n}{2\lambda}\sqrt{1-\phi^2}\}$,        $\xi = 1$, or user defined
 3: **for** range of $\psi$ **do**
 4:     determine $\alpha$ and $\beta$ (EQ 3.35)
 5:     determine prob. transition matrix P
 6:     **p** = steady state probability vector
 7:     $p \rightarrow f_F(x)$ linear interpolation
 8:     $R_{\psi_i} = \int (f_F(x) - f_W(x|0,\sigma_F))^2 dx$
 9: **end for**
10: Select $\psi$ based on lowest residual in $R_\psi$
11: **Initialize:**
        $\beta = \psi\Delta F$
12: **for** range of $\xi$ **do**
13:     determine $\hat{\alpha} = \alpha\xi$
14:     determine prob. transition matrix P
15:     p(F) = steady state vector
16:     $p \rightarrow f_F(x)$ linear interpolation
17:     $R_{\xi_i} = (\int x^2 f_F(x) dx - \sigma_F^2)^2$
18: **end for**
19: Select $\xi$ based on lowest residual in $R_\xi$
        **return** $\psi, \xi$
```

3.6.3 AR - force mapping

In the last sections the force model is established. The discrete force model is a discretized force space where the added noise is a discrete random variable. The mapping function should describe the evolution of force from a state,

$$m(F_i) = F_i \rightarrow F_{i+1} \tag{3.39}$$

and for the white noise process this evolution is described with the probability vector $P(W)$.
For a AR(1) load the mapping is described through the noise variable T defined in Section 3.6.2 and with Eq. 3.29,

$$m(F_k) = \phi_1 F_k + \mathbf{P}(T) \tag{3.40}$$

and for an AR(2) process both the current and the previous force step has to be mapped,

$$m(F_k, F_{k-1}) = \begin{cases} \phi_1 F_k + \phi_2 F_{k-1} + \mathbf{P}(T), & \text{for } F_k \\ F_k, & \text{for } F_{k-1} \end{cases} \tag{3.41}$$

3.7 Harmonic Load

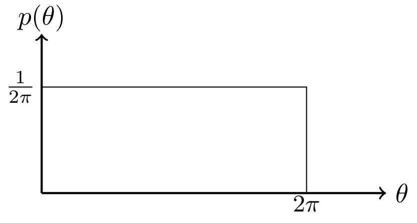


Figure 3.6: Probability distribution of random variable θ

The force model is described in section 2.1.3, and the response is found with the frequency response function,[24]

$$x(t) = H(\omega)F_0 \sin(\omega t + \theta) = G \sin(\omega t + \theta - \phi) \tag{3.42}$$

$$G = |H(\omega)|F_0 \tag{3.43}$$

3.7.1 Probability distribution of response

$H(\omega)$ is defined in EQ 2.33, which means that $H(\omega)$ is purely deterministic for all realisations of θ. the phase ϕ is the phase difference between the load and the response, and makes it possible to deal with the frequency response function as a real number.

In order to find a probability density plot of the displacement one has to look at the amount of the displacement at a certain time $x(t^*)$ lies in the band $x \leqslant x(t^*) \leqslant x + dx$ for $0 < \theta \leqslant 2\pi$. The first order probability density function p(x) is defined: (See Fig. 3.7). [24]

$$Prob(x \leqslant x(t^*) \leqslant x + dx) = p(x)dx = 2d\theta p(\theta)dx \tag{3.44}$$

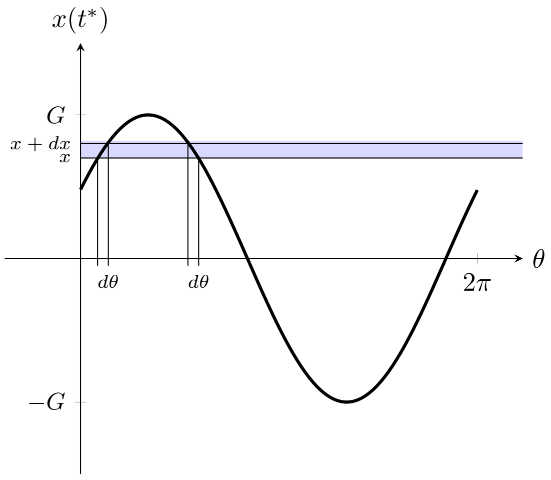


Figure 3.7: Illustrating of time for which t^* for which $x \leqslant x(t) \leqslant x + dx$

Taking the derivative of $x(t)$ with respect to θ gives,

$$d\theta = \frac{dx}{G\cos(\omega t + \theta - \phi)} \tag{3.45}$$

Substituting for:

$$\cos(\omega t + \theta) = \sqrt{1 - \sin^2(\omega t + \theta - \phi)} \tag{3.46}$$

Combining eq: (3.42) & eq: (3.45):

$$d\theta = \frac{dx}{G\sqrt{1 - \frac{x^2}{G^2}}} \tag{3.47}$$

Leads to result when combining eq: (3.44) & (3.47):

$$p(x) = \frac{1}{\pi\sqrt{G^2 - x^2}} \tag{3.48}$$

Figure 3.8 shows that the probability density is lowest at the mean value and increases towards the extremes. It can also be shown by integration that:

$$\int_{-\infty}^{\infty} p(x)dx = \int_{-G}^{G} \frac{1}{\pi\sqrt{G^2 - x^2}}dx = 1 \tag{3.49}$$

Which makes it certain that $x(t) \leqslant |G|$.

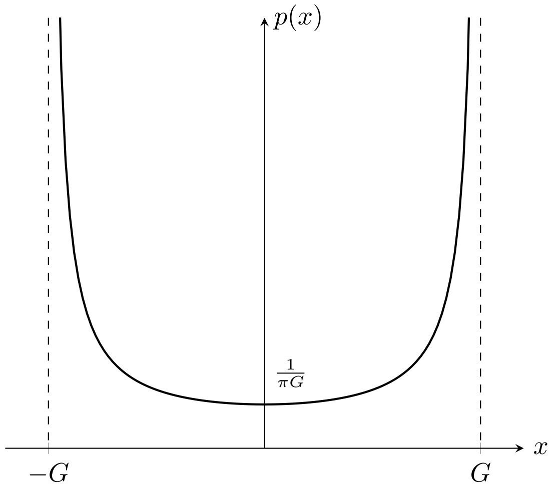


Figure 3.8: Probability density function of harmonic steady state response

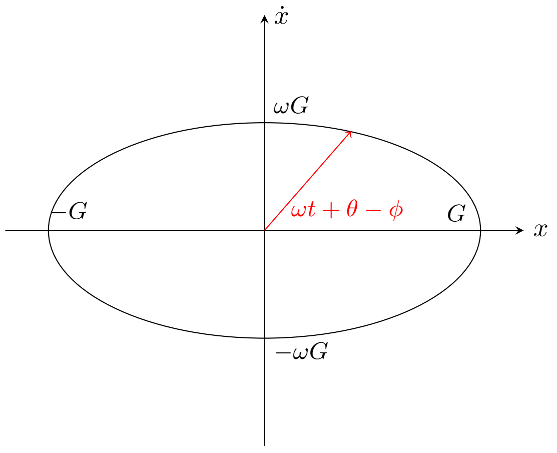


Figure 3.9: Relation between velocity and displacement described with an ellipse for harmonic response

Joint probability density

The joint probability density of displacement and velocity is also of interest. The derivative of the displacement (Eq 3.42) gives the equation for the velocity state,

$$\frac{\dot{x}(t)}{\omega G} = \cos(\omega t + \theta - \phi) = \sqrt{1 - \sin^2(\omega t + \theta - \phi)} = \sqrt{1 - \left(\frac{x(t)}{G}\right)^2} \tag{3.50}$$

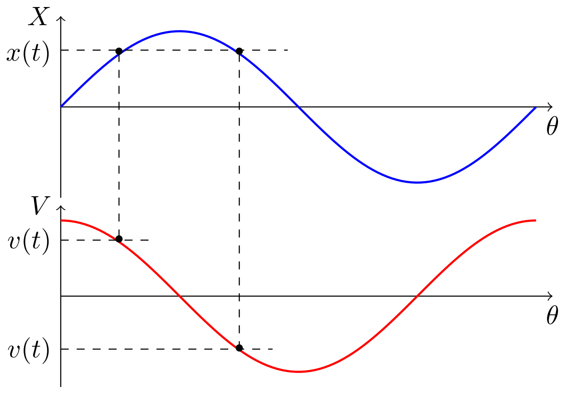


Figure 3.10: Illustrating that a certain displacement will result in 2 possible velocity states

Which states that the solution always will be on the ellipse,

$$\left(\frac{x}{G}\right)^2 + \left(\frac{\dot{x}}{\omega G}\right)^2 = 1 \tag{3.51}$$

The joint probability distribution for independent variables is defined [30]:

$$P(X,Y) = P(X|Y) \cdot P(Y) \tag{3.52}$$

so for a given displacement, there is at most two velocity states possible. This is illustrated in Figure 3.10, and could also be seen by the relation $\cos(\arcsin(x)) = \sqrt{1-x^2}$. This means that the conditional probability could be described with the dirac-delta function δ,

$$P(V|X) = \frac{1}{2}\delta(\dot{x} + \omega\sqrt{G^2 - x^2}) + \frac{1}{2}\delta(\dot{x} - \omega\sqrt{G^2 - x^2}) \tag{3.53}$$

and ultimately the joint probability,

$$P(X,V) = \frac{1}{2\pi\sqrt{G^2 - x^2}}\left[\delta(\dot{x} + \omega\sqrt{G^2 - x^2}) + \delta(\dot{x} - \omega\sqrt{G^2 - x^2})\right] \tag{3.54}$$

3.7.2 Harmonic load-mapping

We have defined our probability space for a harmonic load in section 3.7. In order to implement this in the cell-mapping alporithm the random phase state has to be discretized, and the time evolution, or mapping function has to be described. The mapping function of the phase θ is here defined,

$$m(\theta) = \theta + \omega_f \Delta t. \tag{3.55}$$

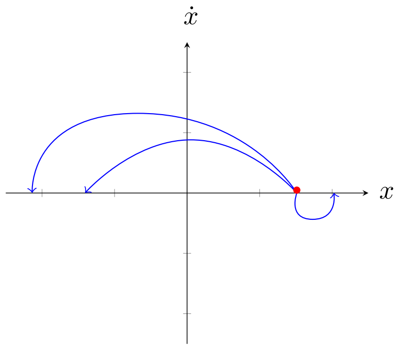
The mapping distance should be equal to the cell size of θ in order to insure that every point from a certain cell is mapped to the same image cell.

$$m(\theta) - \theta = \Delta\theta j \qquad \text{for } j = 1, 2, 3, ... \tag{3.56}$$

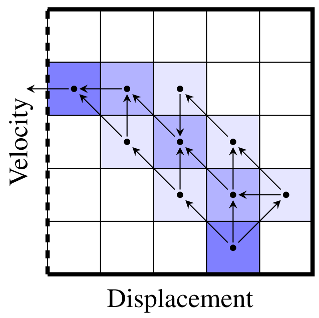
since the cell size $\Delta\theta = \frac{2\pi}{n_\theta}$, the following criteria should be satisfied,

$$\omega_f \Delta t n_\theta = 2\pi j \qquad \text{for j=1,2,3,...} \tag{3.57}$$

3.8 Cycle counting



(a) Monte Carlo sim.



(b) Cell mapping

Figure 3.11: Deterministic and stochastic description of a path

In order to do any fatigue calculations one has to know how many cycles and of which magnitude they are. One of the most common way to do this today is the rainflow counting method [11]. This has to be carried out as a time domain analysis, and the fatigue cycles are counted according to the rainflow method. A drawback with this method is that in order to capture events in the response with low probability of occurring, on has to deal with an extensively long sample. It is therefore of interest to see if one can use the cell mapping transition matrix in order to find the cycles for fatigue calculations.

3.8.1 Markov chain aproximation

A time signal could be described as a series of turning points, where the derivative is zero. One half-cycle is defined for a process going from one turning point to the next at some time. For a dynamic system the turning points would be each time the first derivative of the displacement, the velocity reaches zero. The signal then forms a series of peaks and troughs each defining one half cycle. Frendahl and Rychlik shows that these turning points can be approximated as a Markov chain for Gaussian, as well as for some non-Gaussian loads [12]. This means that a trough is only dependent on the previous peak and vice versa. The half cycle forms a peak through count$\{M_i, m_i\}$. This makes it possible to state the transition with another stochastic matrix,

$$\mathbf{P}^{\mathbf{p}} = (p_{ij}) = P[peak_{k+1} = x_i | peak_k = x_j] \tag{3.58}$$

Finding this *Transition Probability Matrix* between one extrema to another extrema however must be found. On could simulate different half cycle paths with a Monte Carlo simulation and store the results in a matrix. (see Fig. 3.11). In this thesis the *Transition probability matrix* derived in section 3.4 describing the transition from one state to the next state is used to describe the probabilistic half cycle path with initial condition,

$$P(X, V) = \delta(X = x_0) \cdot \delta(V), \tag{3.59}$$

describing the system starting with an amplitude x_0 and zero velocity. With cell mapping this could be achieved by finding the cell which satisfies the initial condition (transient state), and iterate with the transition matrix until it reaches the final extrema state (recurrent state). The recurrent state is the final state of a path when the velocity reaches zero. In order to describe such a path, the *Transition Probability Matrix* has to be modified in some way to satisfy the following: 1. If mapped to a recurrent state, the probability should stay in this state for every following iterations. 2. If an image cell of a state is defined with a change of velocity the path should be transitioned to a recurrent state.

The first criterion is dealt with by finding all the recurrent cells which describes a zero velocity state, c_r.

$$P^{mod_1} = P \cdot I_r + \hat{I}_r \tag{3.60}$$

where I_r is the identity matrix with zero columns at the recurrent states, and $\hat{I}_r$ is the zero matrix with ones at the recurrent states.

The second criterion is dealt with by finding all the cells where the mapping crosses the zero velocity state, and map these cells again as described with the algorithm presented in section 3.4.

$$P^{mod_2} = \hat{P} = P \cdot I_c + P_c \tag{3.61}$$

where I_c is the Identity matrix with zeros at the states crossing zero velocity, and P_c is the transition matrix only describing the mapping from this states. The transition matrix P_c has to be computed using the algorithm in section 3.4 with similar limitations used in (Eq. 3.13 & Eq. 3.14), where the mapping is prevented from crossing the zero velocity line. This will result in an accumulation of points mapped to the states along the zero velocity axis. There are methods developed for manipulating the matrix and finding the recurrent states with an initial transition state [4]. If one wants to extract the time information, how long it takes for a half cycle the only way is through iterations [4]. For large problems iteration might also be the fastest way to extract the cycle results. However, one has to ensure that a predominant number of image points has reached a recurrent state. Algorithm 3 shows an algorithm developed in order to iterate a solution to get the Extrema to extrema transition matrix.

3.8.2 Damage Intensity

The *Damage Intensity* is usually found by first creating a time signal and then find the *counting intensities* from this time signal. (Eq. 2.59) [12] In this thesis a new approach is considered, by finding the *counting intensity* through the extrema to extrema transition matrix. Since this stochastic matrix describes the transition from peak to trough and vice versa, the steady state vector $\mathbf{p^P} = [P(u = x_1), P(u = x_2), ..., P(u = x_i)]$ of this system will describe the peak distribution,

$$\mathbf{p^P P^P} = \mathbf{p^P} \tag{3.62}$$

Now consider two submatrices in $\mathbf{P^P}$ where $\mathbf{P^{P \to T}}$ is the transition from a certain peak to the following trough, and $\mathbf{P^{T \to P}}$ is the transition from from a trough to the following peak.

$$\begin{aligned} \mathbf{P^{P \to T}} &= (p_{ij}) = P[v = x_j | u = x_i] \quad \text{for } i \geq j \\ \mathbf{P^{T \to P}} &= (p_{ij}) = P[u = x_j | v = x_i] \quad \text{for } i \leq j \end{aligned} \tag{3.63}$$

Algorithm 3 Create Extrema Matrix

```
 1: Inputs:
        Cell Mapping Transition Matrix: $\mathbf{P}$,
        recurrent cells: $c_r$,
        crossing cells: $c_c$
 2: $\hat{\mathbf{P}} = \mathbf{P} \cdot I_r + \hat{I}_r$
 3: Cell Mapping$\{c_c\} \rightarrow \mathbf{P_c}$
 4: $\hat{\mathbf{P}} = \hat{\mathbf{P}} \cdot I_c + \mathbf{P_c}$
 5: Initialize:
        set tolerance,
        $\mathbf{P^p} = \{zeros, shape(c_r.size, c_r.size)\}$
 6: for i in every displacement state do
 7:     $\mathbf{p_0}$ =$\{$1 for displacement state, 0 all other$\}$
 8:     $\mathbf{p} = \mathbf{P} \cdot \mathbf{p_0}$
 9:     while $1 - p(c_r) > tolerance$ do
10:         $\mathbf{p} = \hat{\mathbf{P}} \cdot \mathbf{p}$
11:     end while
12:     $\mathbf{P^p}(j) = p(c_r)$
13: end forreturn $\mathbf{P^p}$
```

The probability of having a cycle with a peak of height u followed by a trough with the depth v can the be expressed with the joint probability function (Eq. 3.52),

$$\begin{aligned} P[u = x_i, v = x_j] &= P[v = x_j | u = x_i] \cdot P(u = x_i) \\ &= \mathbf{p^P}(u_i) \cdot \mathbf{P^{P\rightarrow T}}(u_i, v_j) \end{aligned} \tag{3.64}$$

the *counting intensity* for the peak trough cycles can then be obtained,

$$\begin{aligned} P[x > u, y < v] &= \int_{x=u}^{\infty} \int_{-\infty}^{y=v} P[x = u, y = v] dy dx \\ &\approx \sum_{x_i = v_i}^{n} \sum_{x_j = 1}^{v_j} \mathbf{p^P(x_i)} \mathbf{P^{P\rightarrow T}}(x_i, x_j) \\ &= \mu^{PT}(u, v) \end{aligned} \tag{3.65}$$

Having the *counting intensity* for the peak trough count the next step is to acquire the rainflow counting intensity. The procedure is defined in section 2.6, and since the intensity is defined with discrete values and represented in a matrix, the *Damage intensity* (Eq. 2.60) can be calculated as follows:

$$E[D(1)] = \sum_{j=1}^{n} \sum_{i=j}^{n} \mu^{RFC}(x_i, x_j) \beta(\beta - 1)(u - v)^{\beta - 2} (\Delta x)^2 \tag{3.66}$$

3.8.3 Damage intensity for Narrow banded process

The damage function is described with equation 2.55. The damage caused by a cyclic peak is for a narrow banded process $N(u) = (2u)^{-\beta}$ (since the stress range varies over 2 times the peak height, (see Eq. 2.61). The damage intensity can then be written,

$$\begin{aligned} E[D(1)] &= E[\nu_0^+ \int_0^\infty (2u)^\beta \frac{u}{\sigma_x^2} e^{-\frac{u^2}{2\sigma_x^2}} du] \\ &= \nu_0^+ \left[2^{3\beta/2} \left(\frac{1}{\sigma_x^2} \right)^\beta \Gamma\left(\frac{\beta}{2} + 1 \right) \right] \end{aligned} \tag{3.67}$$

Chapter 4

Steady State Calculations

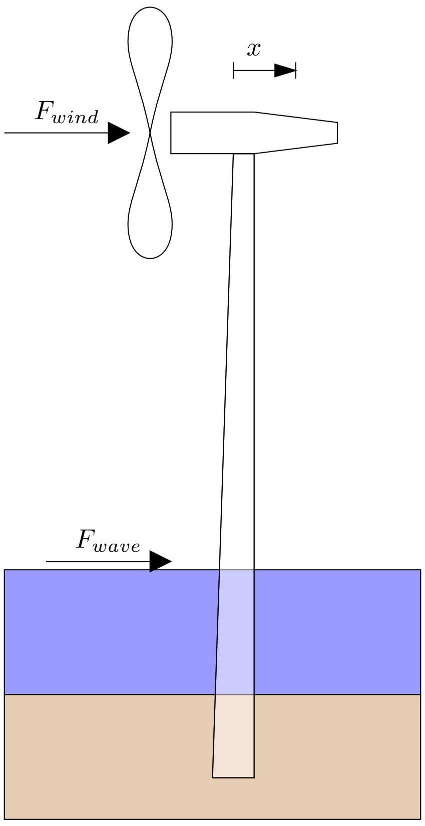


Figure 4.1: Sketch of a shallow water wind turbine with natural loads

The structural properties of the wind turbine in Figure 4.1 could be found by analysing the modal shapes [8], or with any finite element anslysis software which is done in a previous master thesis by Hembre [13]. In this thesis the properties for the fundamental mode of a NREL 5-MW Baseline Wind Turbine will be used in the testing of the method. [13].

Table 4.1: System properties of first modal shape of NREL 5-MW Baseline Wind Turbine

m	=	4.1 10^5 kg
k	=	1.8 MN/m
c	=	1.7 10^4 kg/s
ω_n	=	2.1 rad s^{-1}
f_n	=	0.33 Hz
ζ	=	0.01

Computer Properties

Table 4.2: Computer Properties used in analysis

OS	=	Windows 10 64-bit operating system
CPU	=	Intel(R) Xeon(R) CPU E5-2630 v3 @2.4GHz
Cores	=	6
Logical processors	=	12
Memory	=	64 Gb DIMM

4.1 AR(0) load - White Noise

The state $\mathbf{Y}$ is described by the displacement, velocity and force acting on the system, which makes the sample space a 3 dimensional space described with the vectors $\mathbf{\Omega} = [\mathbf{x}, \dot{\mathbf{x}}, \mathbf{F}]$

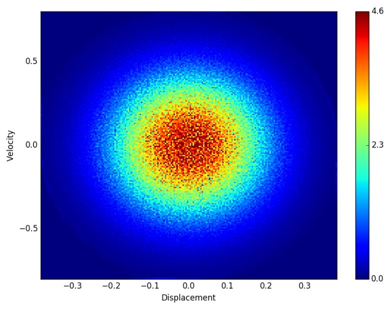
Discretization

Let the white noise spectra be defined with an intensity $S_0 = 1.59 \cdot 10^8 \ N^2/Hz$. The standard deviation in the force and response can be found with the equations in Chapter 3.6. For this test the cell space is meshed within 3 times the standard deviation in both displacement, velocity and force which would capture 99.7% of all events. The total number of cells are 404 010. Both displacement and velocities are discretized into 201 states, while the force is discretized into 10 states. The fine mesh in the displacement and velocity state is chosen to accurate map how the system response. While for the force, a few states is necessary to represent the statistical properties, and the round-off error due to a coarse mesh is expected to be smeared out through the mapping function (Runge Kutta Integration).

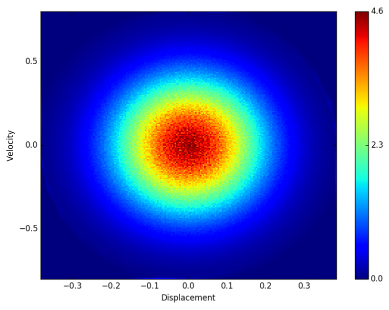
Table 4.3: AR(0) Load Model Parameters

ϕ	=	0
σ_W	=	100 kN
Δt	=	0.1 s
n_x	=	201
n_v	=	201
λ	=	3
n_F	=	10

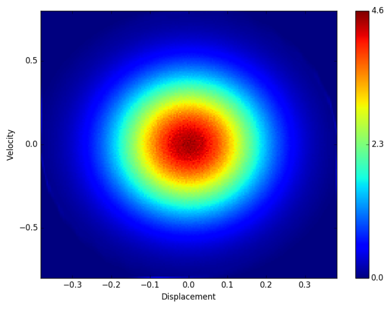
4.1.1 Number of Random Mapping Points



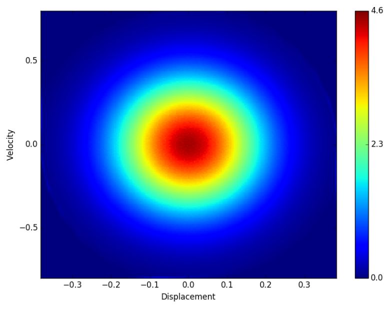
(a) 1 Random Point



(b) 10 Random Points



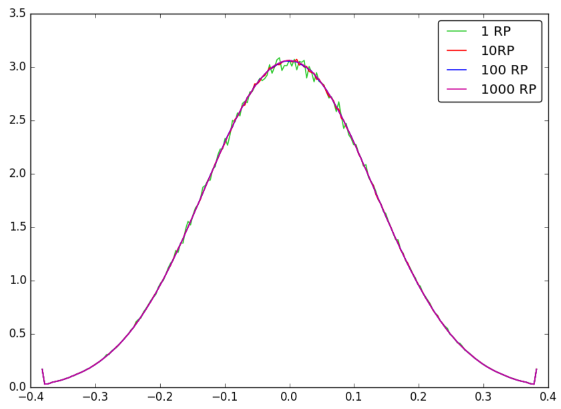
(c) 100 Random Points



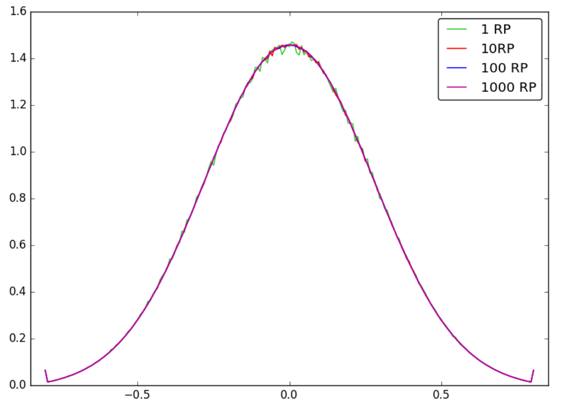
(d) 1 000 Random Points

Figure 4.2: Joint Probability steady state with different number of mapping points in each cell

One interesting aspect is to see how many sampling points is needed in each cell in order to get a sufficiently accurate plot of the steady state probability distributions. The joint probability plot (Figure 4.2) shows the distribution of displacement and velocity. The state of the system is described as in 3 dimensions, which means that there are 10 cells which



(a) Displacements



(b) Velocities

Figure 4.3: Probability density plot of displacement and velocity with different number of mapping points in each cell

corresponds to a given displacement and velocity. (The same number as length of force vector). The joint probability plot is therefore the sum of probability over all 10 force states. Where the probability density plot of displacement and velocity is summed up over 10 x 201 states. A single mapping point in the 3 dimensional cell space is then in fact 10 mapping points in the displacement-velocity space, and 2100 mapping points in either displacement, or velocity.

For an analysis where the force mapping, and resolution of the joint probability of the displacement, and velocity is required a higher number of random mapping points should be carried out. If however only the one dimensional distributions along either the displacement, or velocity axis is important a much smaller number of random mapping points is required. Figure 4.3 shows almost the same accuracy for 10 random mapping points as for 1000 random mapping points.

4.1.2 Time step

Monte Carlo Simulations

Table 4.4: Standard Deviation of displacement with Monte-Carlo-Simulations

Time Step	Measured STD.	Exact STD	error
0.01s	0.1277	0.1273	0.310 %
0.05s	0.1272	0.1273	-0.08 %
0.1s	0.1257	0.1273	-1.24 %
0.2s	0.1243	0.1273	-2.36 %
0.3s	0.1218	0.1273	-4.29 %
0.4s	0.1153	0.1273	-9.45 %
0.5s	0.1024	0.1273	-19.5 %

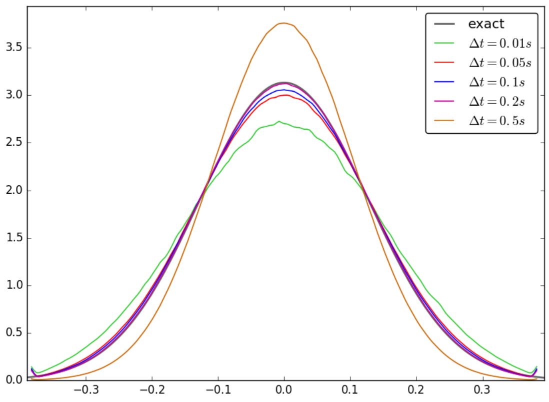
The time step used in the integration could be crucial for having the correct estimation of the probability density plot. The first analysis carried out is a Monte Carlo Simulation of the load, where probability density distribution of the response is estimated. For each step a force vector is generated with random normal distributed entities with zero mean, and standard deviation found in (Eq 3.23). Each simulation is about 100h long after deleting the transient behaviour at the beginning of the signal. The transient threshold is set to 10^{-10} meaning that the amplitude of the transient signal has decayed 10^{-10} times. The analysis shows it is crucial to choose a small time step in order to estimate the distribution with a decent accuracy.

Cell-mapping

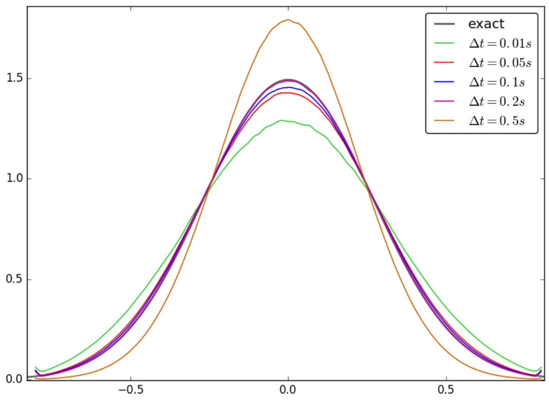
Table 4.5: Standard Deviation of displacement with Cell-Mapping

Time Step	Measured STD.	Exact STD	error
0.01s	0.1419	0.1273	11.5 %
0.05s	0.1302	0.1273	2.34 %
0.1s	0.1283	0.1273	0.841 %
0.2s	0.1262	0.1273	-0.829 %
0.5s	0.1058	0.1273	-16.9 %

For each cell 100 random points are selected in order to determine the image of the cell, and determine the *Transition Probability Matrix*. The steady state solution is found by solving the largest eigenvalue of the Transition probability Matrix. The results show that for the smallest time step, the cell mapping method will not give accurate results.



(a) Displacements



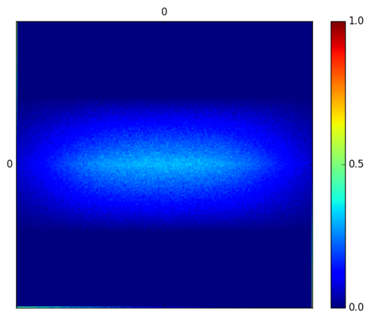
(b) Velocities

Figure 4.4: Probability density plot of displacement and velocity for different time steps

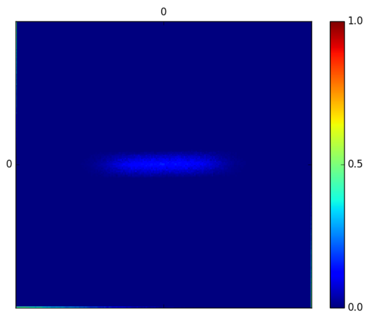
Ammount Mapped in Starting Cell

The Monte Carlo Simulations shows that in order to get a satisfactory estimation of the probability density distribution of either displacement, or velocity a small step size is necessary. Even while all the step sizes in the analysis above are below the critical time step, and the Nyquist frequency, in order to get a good estimation of the probability density distribution, an even smaller time step is necessary in order to caption all the distribution away from the mean.

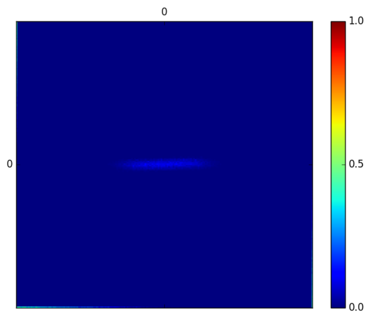
The plots also reveals the problem that occurs with domain discretization and small time steps. For the smallest time step $\Delta t = 0.01s$ the probability is diffused over a



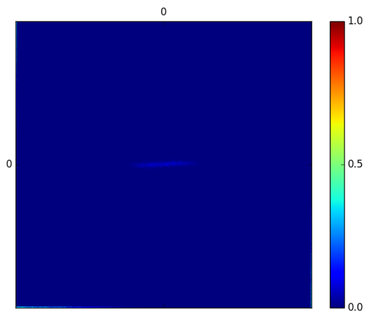
(a) $\Delta t = 0.01s$



(b) $\Delta t = 0.05s$



(c) $\Delta t = 0.1s$



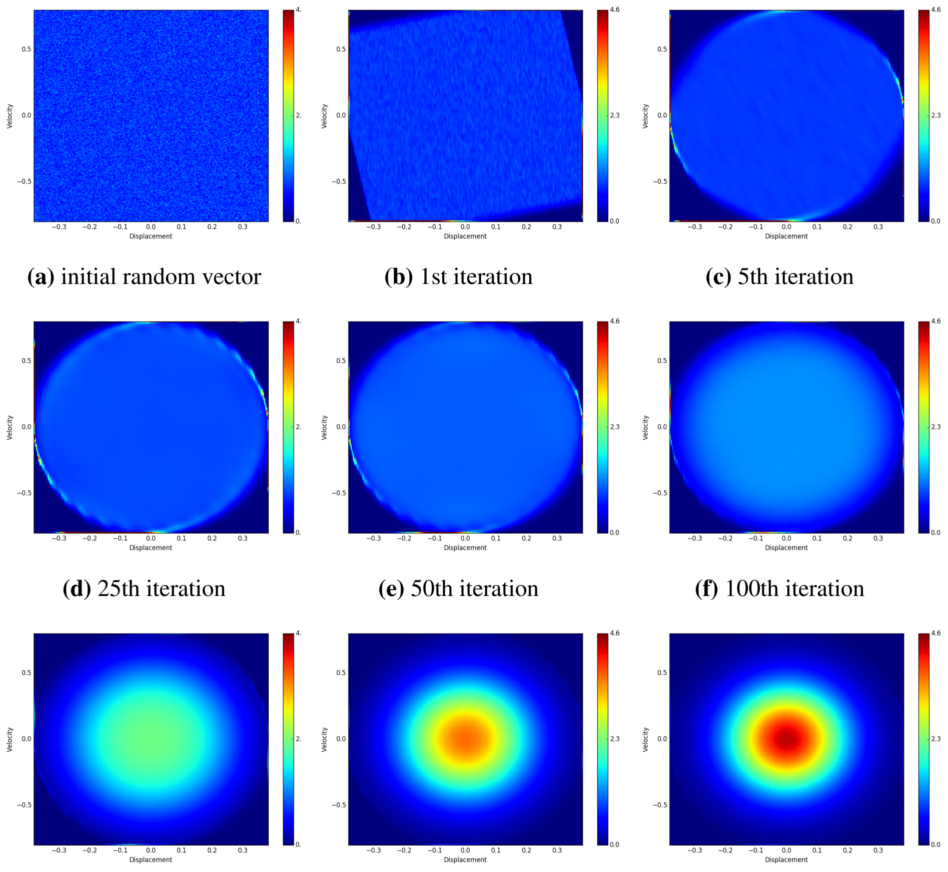
(d) $\Delta t = .2s$

Figure 4.5: Probability of Image cell mapped to the same state as the starting state

large domain. This result could be explained when a cell size is large relative to the time integration step a lot of image point will not be able to reach outside the starting cell. Which means that the Transition Probability Matrix will not be able to describe the dynamic behaviour in a sufficient manner. The stiffness caused by the cell discretization is seen in all the time steps tested where Table 4.5 shows a larger standard deviation for all time steps in the cell-mapping method than for Monte-Carlo simulations.

To check whether the cell size is too large compared to the time step, the amount of probability mapped in to the same cell could be checked considering only the *Transition Probability Matrix*. The amount mapped into the same state are the diagonal entities: (p_{ij}), where $i = j$. However, when only considering the response displacement, and velocity mapped into the same state this is solved with a basic script. This can be seen in the Appendix. Figure 4.5 shows the the probability of mapping back to the same response state for different time step. It is clear that for a time step $\Delta t = 0.01s$ a lot of the probability is mapped back to the same displacement-velocity state, and could therefore explain why the distribution is so far off.

4.1.3 Boundary Condition



(a) initial random vector

(b) 1st iteration

(c) 5th iteration

(d) 25th iteration

(e) 50th iteration

(f) 100th iteration

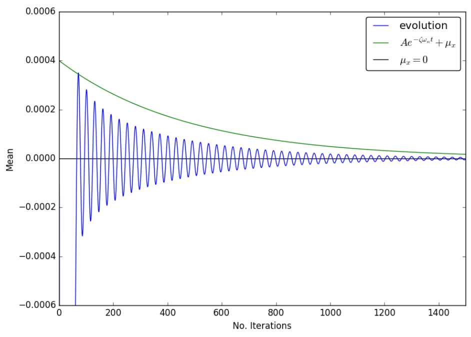
(g) 250th iteration

(h) 500th iteration

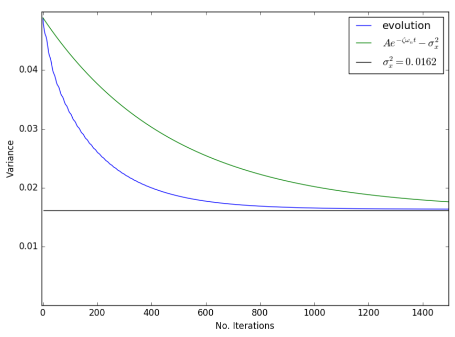
(i) 1000th iteration

Figure 4.6: Evolution of response for a white noise load starting with a random vector

The boundaries which limit the displacement, velocity from escaping the discretized domain add a distortion to the system. The image point is mapped to a cell in the domain while the system wants to put the image point outside. The mapping of these points would therefore be spurious Image points. Figure 4.6 shows how the system behaves at the boundaries after different number of iterations with the Transition Matrix. The effect form the boundaries are primarily seen at the first iterations where the probability is spread randomly. For a higher number of iterations, the probability moves towards the centre of the domain, and the effect from the boundary is therefore marginal.
Figure 4.7 shows how the mean and variance evolves with number of iteration. It seems that the statistical properties goes toward the steady state solution faster than the transient behaviour decays (green curve). This shows that the statistical properties also could



(a) Mean



(b) Variance

Figure 4.7: Evolution of mean and variance of displacement for white noise load

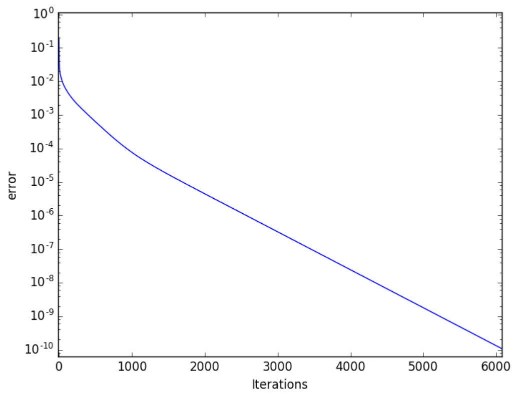


Figure 4.8: Error vs. number of iterations

be implemented in an iteration scheme in order to satisfy convergence of the statistical properties. Looking at the error (difference between current and previous probability state vector) Figure 4.8 shows a linear convergence rate.

Steady state for different range specifications

A goal for this thesis is to demonstrate how different ranges of the domain influence the steady state distribution. Table 4.6 shows how the displacement, and velocity mesh is discretized for the different setups. The force is discretized into 15 states, and there are 100 of random points in each cell to determine the cell mapping. The steady state solution is found by solving the largest eigenvalue of the transition matrix.

The result shows that for an accurate estimation of the probability distribution one should at least look at a range 3 times the standard deviation of the response. For smaller range the accumulation of probability can be seen at the boundaries, which would be a clear indication that the domain range should be widened. Figure 4.9 also shows that a standard deviation of at least 4 is need in order to remove any significant accumulation of probability at the boundaries.

Table 4.6: Range specification

Range	Mesh	$\Delta x/\Delta v[m]$
4σ	160x160	0.00625/0.0125
3σ	120x120	0.00625/0.0125
2σ	80x80	0.00625/0.0125
1.5σ	60x60	0.00625/0.0125
1σ	40x40	0.00625/0.0125

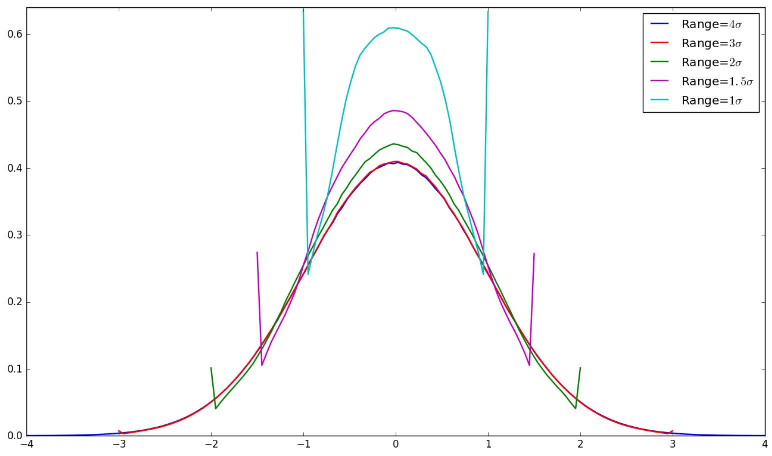
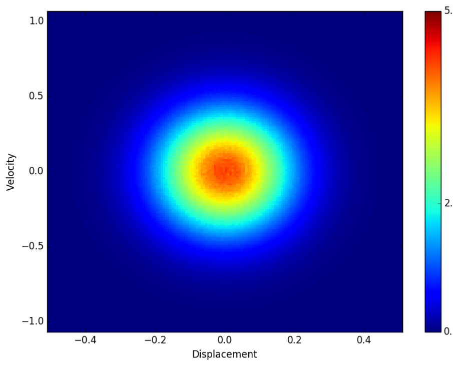
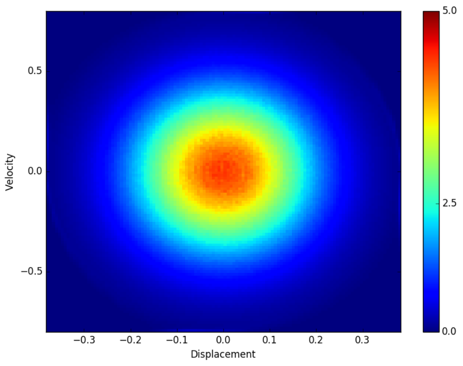


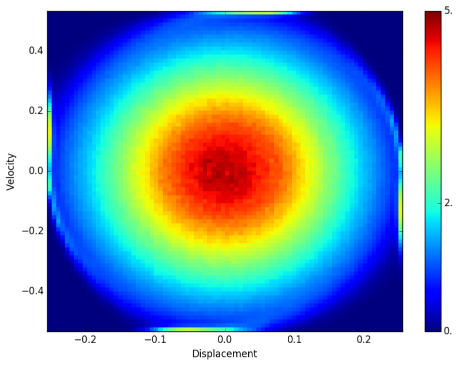
Figure 4.9: Displacement distribution for different boundary ranges



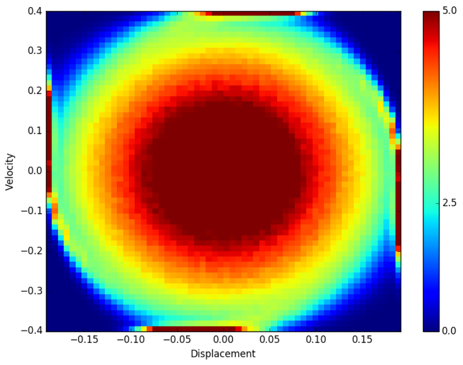
(a) $Range = 4\sigma$



(b) $Range = 3\sigma$



(c) $Range = 2\sigma$



(d) $Range = 1.5\sigma$

Figure 4.10: Joint probability plots for different boundary ranges

4.2 Single Harmonic load

Table 4.7: Single Harmonic Load Parameters

F_0	=	100 kN
ω_f	=	3 rads^{-1}
n_θ	=	100
n_r	=	100
Δt	=	0.67

The single harmonic load case is described in section 3.7. In this section the steady state of this load case will be investigated. This means that the discretisation in time, response

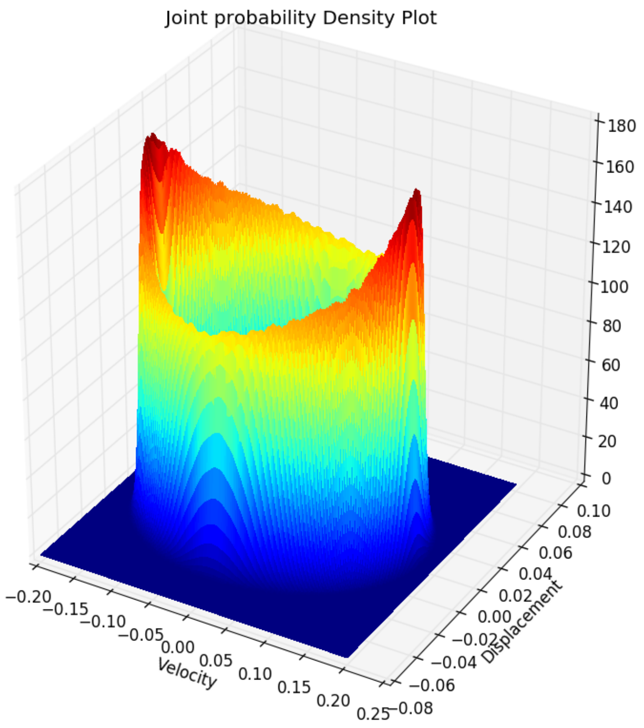


Figure 4.11: 3D Illustration of Steady State Distribution from harmonic load response

(displacement and velocity) and phase is changed to see how these changes influences the final distribution. Figure 4.11 is a 3D projection of the contour plot in Figure 4.14d as a spacial illustration of the final distribution.

4.2.1 Variable time step

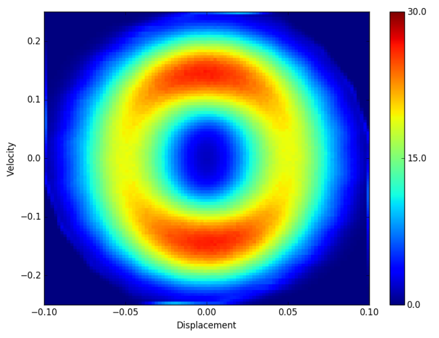
In order to find out how a different time step influences the steady state response, a relatively fine mesh is chosen in displacement, velocity and phase. The probability space is 3 dimensional $\mathbf{\Omega} = [x, \dot{x}, \theta]$, and the displacement and velocity is discretized into 100 states each. The phase discretization is described in Table 4.8 where the number of states is chosen to satisfy eq 3.57. For calculation the probability transition matrix, 1000 random points are selected in each cell, and the steady state is found by solving the largest eigenvector of the transition matrix.

The final results show a lot of probability diffusion for a small time step. Figure 4.12 shows how the probability distribution gets narrower for larger time steps. The largest time step shows a skewed joint distribution and indicates that the solution could grow unstable. There migth therefore seem that a time step $\Delta t = \frac{2}{\omega_f}$ yields the most accurate results.

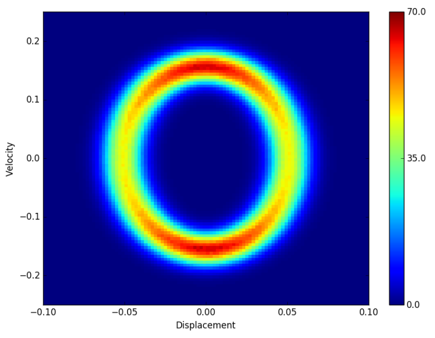
Figure 4.13 shows the distribution along the displacement- and velocity axis compared with the theoretical result provided in Section 3.7. There seem to be a tendency for all time steps to underestimate the peak probability of the displacement distribution. Both distributions seem to get closer to the theoretical solution for increasing time steps except for the largest time step.

Table 4.8: time step used, relation to load frequency, and number of phase cells

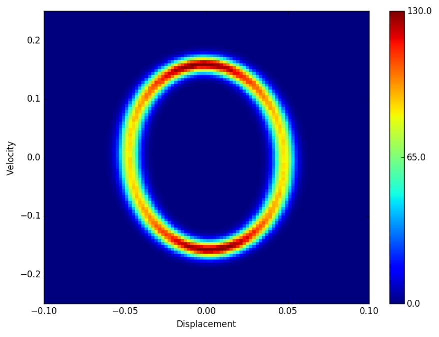
Δt	$\Delta t \omega_f$	n_θ
0.1197 s	0.3591	100
0.2394 s	0.7182	100
0.4787 s	1.4361	100
0.5984 s	1.7952	100
0.6667 s	2.0	110
0.8976 s	2.6928	100



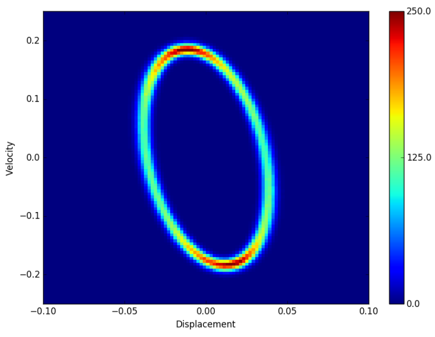
(a) $\Delta t = 0.12$



(b) $\Delta t = 0.48$

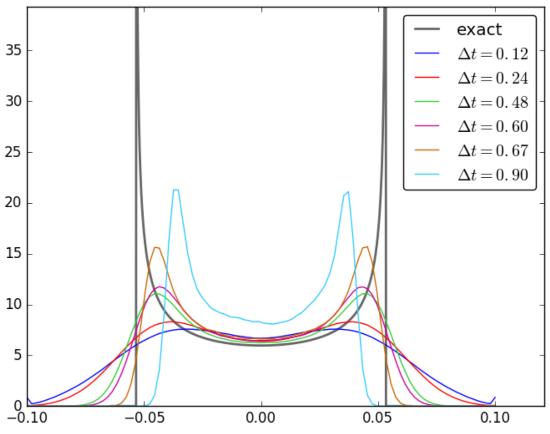


(c) $\Delta t = 0.67$

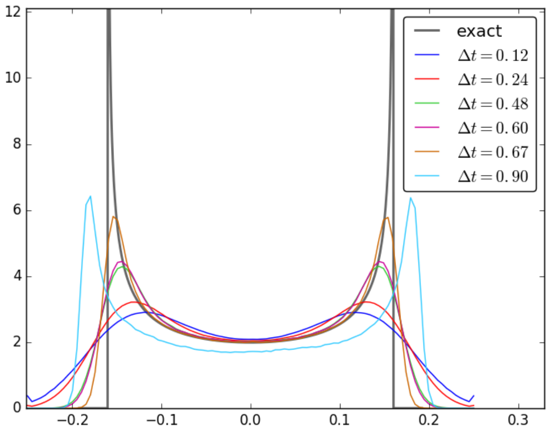


(d) $\Delta t = 0.90$

Figure 4.12: Harmonic load response distribution for different time steps Δt



(a) Displacement



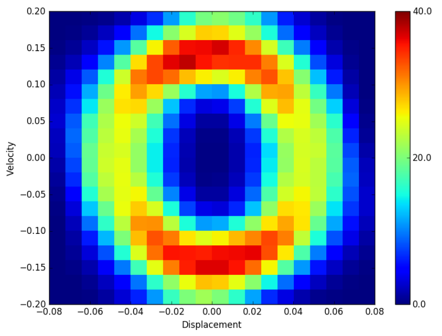
(b) Velocity

Figure 4.13: Displacement-, and velocity harmonic load response distribution for different time steps compared to exact solution

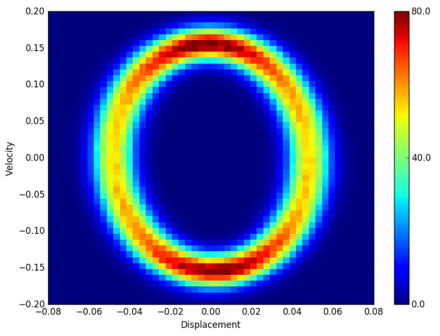
4.2.2 Different response mesh

The objective is to demonstrate how a fine mesh versus a coarse mesh affects the results. The phase is discretized into 100 states, while a time step of 0.6 seconds is chosen. 1000 Random points is chosen in each cell in order to calculate the probability transition matrix, and the steady state is solved with the eigenvector of the transition matrix.

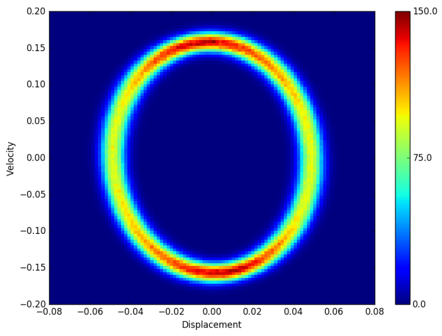
The result show larger probability diffusion for a coarse mesh than for finer mesh. The probability distribution along the displacement, and velocity axis gets closer to the theoretical distribution for each refinement of the mesh. How accurate the result has to be can be how fine the response mesh should be discretized. As opposed to the different time steps, there does not seem to be any unstable solutions for different response discretizations, but the solution improves when the mesh is refined.



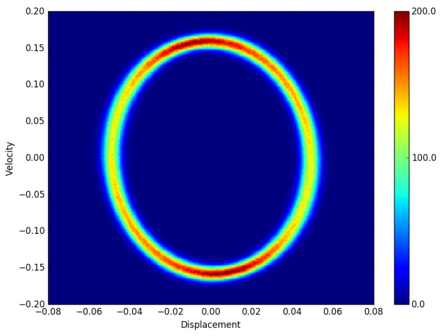
(a) $n_x = n_{\dot{x}} = 20$



(b) $n_x = n_{\dot{x}} = 50$

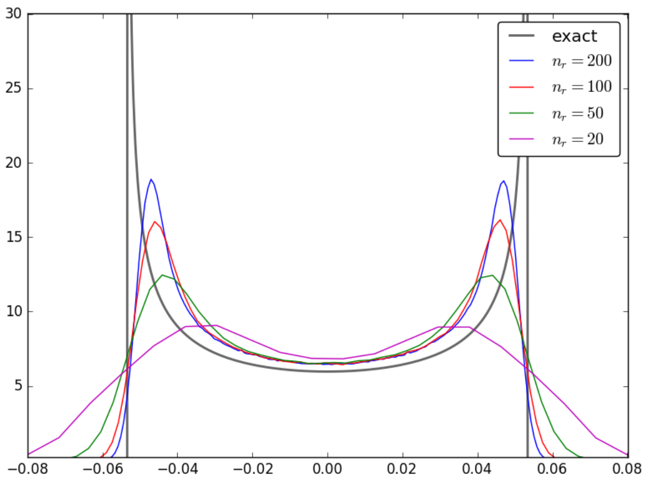


(c) $n_x = n_{\dot{x}} = 100$

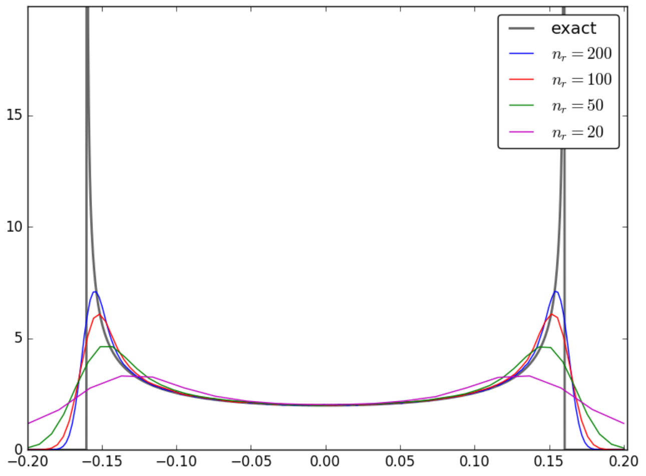


(d) $n_x = n_{\dot{x}} = 200$

Figure 4.14: Harmonic load response distribution for different response discretizations



(a) Displacement distribution



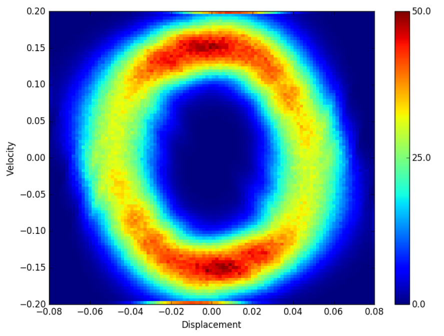
(b) Velocity distribution

Figure 4.15: Displacement-, and velocity harmonic load response distribution for different response discretizations compared to exact solution

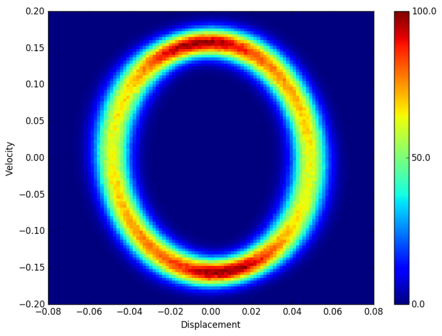
4.2.3 Different phase size

In this section a different size for the phase state is varied while the response mesh is kept to 100 states in both displacement, and velocity direction. The time step chosen is 0.6 seconds. The transition probability matrix is calculated with 1000 random points, and the steady state is found by finding the eigenvector.

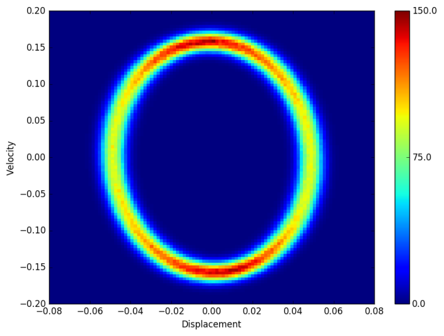
The results are shown in Figure 4.16 and Figure 4.17. It is shown that the solution is improved for finer discretization. There is also nothing that implies that the solution is unstable for any kind of discretization. However, the solution is gradually improved for a finer mesh.



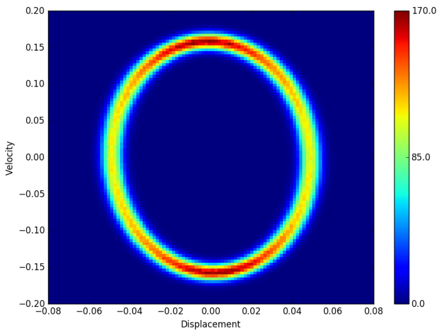
(a) $n_\theta = 20$



(b) $n_\theta = 50$

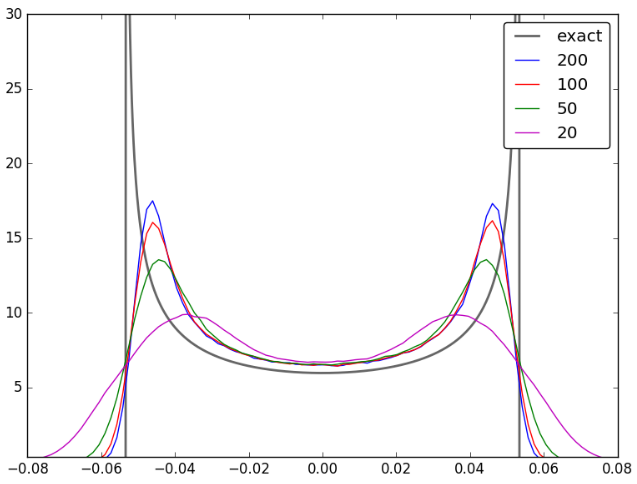


(c) $n_\theta = 100$

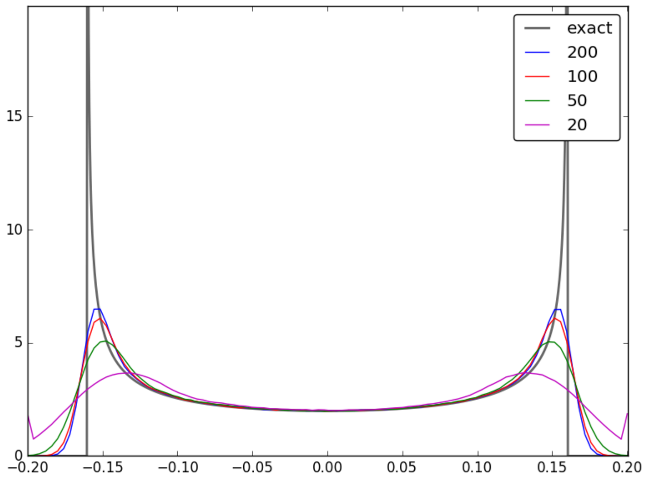


(d) $n_\theta = 200$

Figure 4.16: Harmonic load response distribution for different phase discretizations



(a) Displacement distribution



(b) Velocity distribution

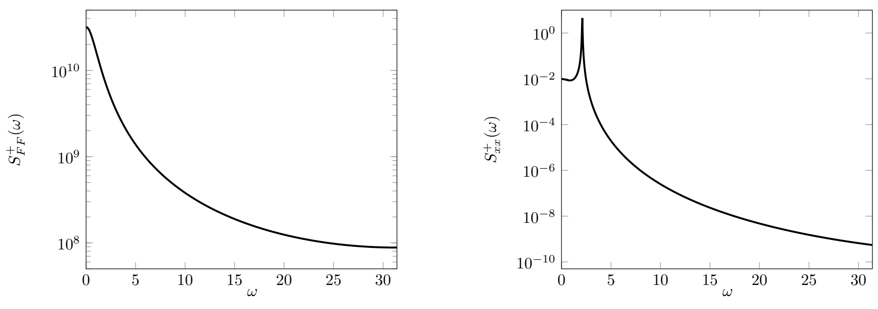
Figure 4.17: Displacement-, and velocity harmonic load response distribution for different phase discretizations compared to exact solution

4.3 AR(1) Load

The structural properties are defined in Table 4.1, while the parameters for the AR(1) Force model has to be chosen. Usually this is done by comparing the Spectral Density of the signal generated by the model with a wanted spectral density, however for this test arbitrary values are chosen in Table 4.9. Figure 4.18 shows the spectral densities fro the load and response for the chosen values, and the standard deviation of the force and response is presented in Table 4.10 using Eq. 2.13, and Eq. 2.37.

Table 4.9: AR(1) Load Model Parameters

ϕ	=	0.9
σ_W	=	100 kN
Δt	=	0.1 s
n_x	=	100
n_v	=	100
λ	=	3
n_F	=	9 or 10
ψ	=	1.16 or 1.22
ξ	=	1.98 or 1.73
Random points	=	10 000



(a) Load Spectrum **(b)** Response Spectrum

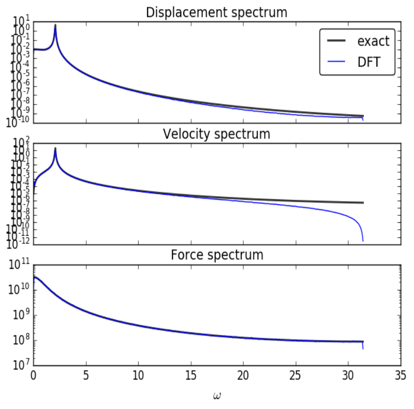
Figure 4.18: Exact Spectral Distributions of AR(1) load and response

Table 4.10: Excact Standard deviation of force and response of described AR(1) load

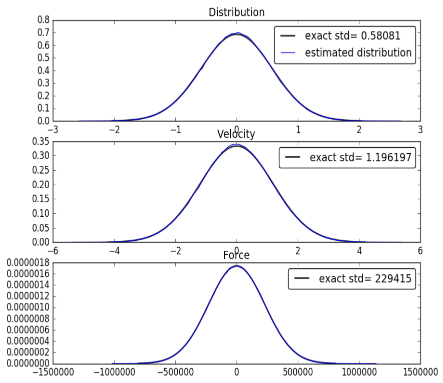
σ_F	=	229 kN
σ_x	=	0.5808 m
$\sigma_{\dot{x}}$	=	1.1962 m

4.3.1 Monte Carlo Simulation of response

Monte Carlo simulation is performed in order to see whether the time integration of the equation holds a sufficient accuracy. The simulation is performed by generating a 15 hour Force Vector with 1.08 10^6 samples, which are integrated stepwise in the Runge Kutta Method described in Section 3.1. Figure 4.19 shows the spectral density and, probability



(a) Spectral Density distribution

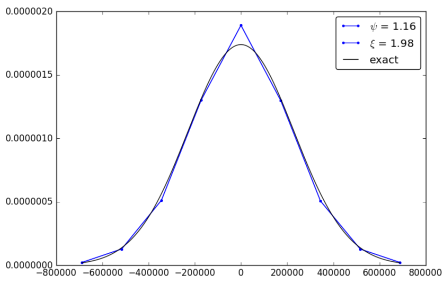


(b) Probability Density distribution

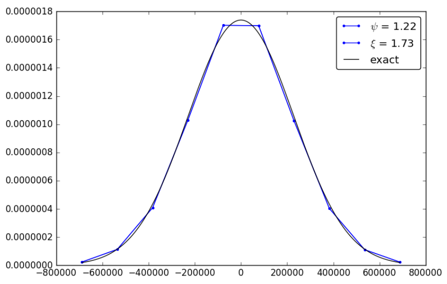
Figure 4.19: Spectrum and Distribution from Monte Carlo Simulation of AR(1) load

distribution of the response signal with Monte Carlo simulation. Both plots seem too be in accordance with the theoretical solution and would therefore expect to give accurate results in the cell mapping algorithm.

4.3.2 Force Evolution



(a) Force distribution for $n_F = 9$



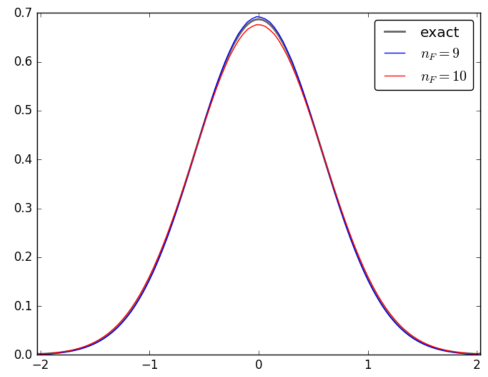
(b) Force distribution for $n_F = 10$

Figure 4.20: AR(1) Force evolution distribution

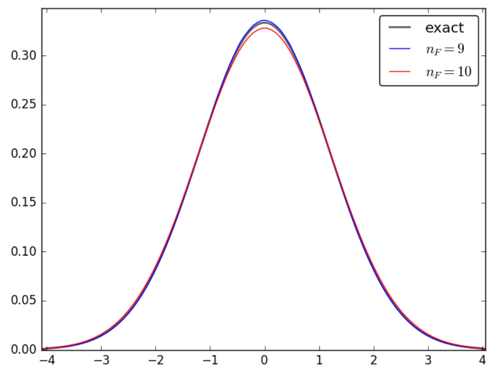
Before analysing the response of the system, a cell-to-cell mapping analysis is done with the load only. For this load, two force discretizations will be investigated. First when the load space is discretized into an odd number of cells, and secondly when discretized into an even set of cells. The one dimensional cell space $\mathbf{\Omega} = [\mathbf{F}]$ is therefore divided into $n_F = 9$ cells, and then into $n_F = 10$ cells. Even if this is a coarse discretization it will be interesting whether this could produce accurate results. The empirical parameters ψ and ξ

are found by iterating through different values. These results can be found in the appendix. Figure 4.20 shows the final distribution and the values for the modified noise paramters.

4.3.3 Cell Mapping results



(a) Displacement distribution



(b) Velocity distribution

Figure 4.21: Displacement, and velocity response of AR(1) laod from Cell Mapping

The cell mapping is performed with either 9 or 10 force cells, while the displacement and velocity response is discretized into 100 states each. That means the system consist of either 90 000 cells or 100 000 cells. The steady state result for the two cases are quite similar. However, it seems that the even discretized force overestimates the response, while the odd discretized load case underestimates the response. (Fig 4.21 and Table 4.11)

Table 4.11: Standard deviation of AR(1) Load response

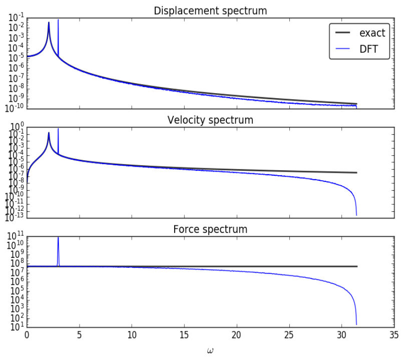
	σ_x	σ_v	error
n_F=9	0.5754	1.1844	-0.9588%
n_F=10	0.5881	1.2111	1.250%

4.4 Harmonic load with withe noise

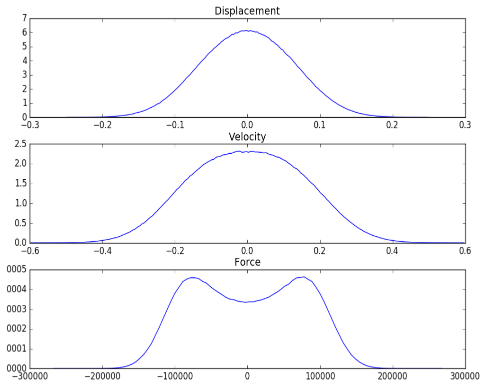
Table 4.12: Harmonic load with white noise Model Parameters

σ_W	=	40 kN
F_0	=	100 kN
ω_f	=	3 rads^{-1}
Δt	=	0.1 s
λ	=	3
n_f	=	20
n_x	=	201
n_v	=	201
n_θ	=	90

4.4.1 Monte Carlo Simulation of response



(a) Spectral Density distribution

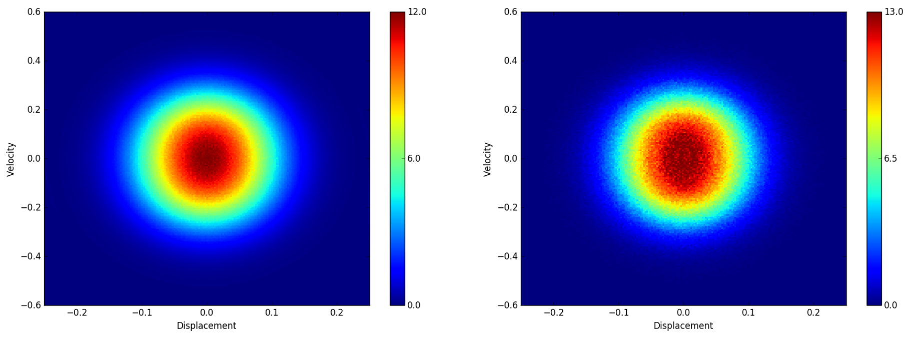


(b) Probability Density distribution

Figure 4.22: Spectrum and Distribution from Monte Carlo Simulation of Harmonic Load with white noise

The Monte Carlo simulation shows that the response is not a Gaussian process, due to the shape of the velocity which does not have the characteristic ”bell shape” of the Gaussian normal distribution. It is therefore interesting to wee whether the cell mapping method is capable of such a non-Gaussian probability distribution.

4.4.2 Cell Mapping

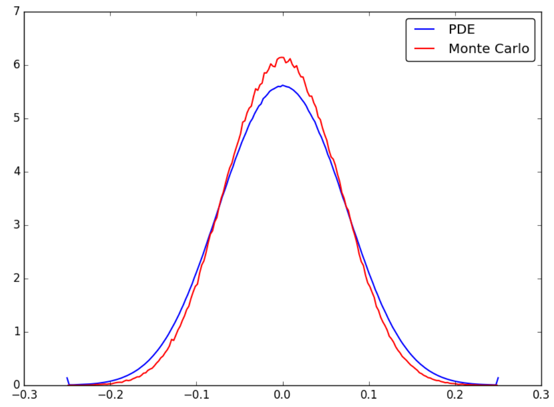


(a) Cell-Mapping **(b)** Monte-Carlo simulation

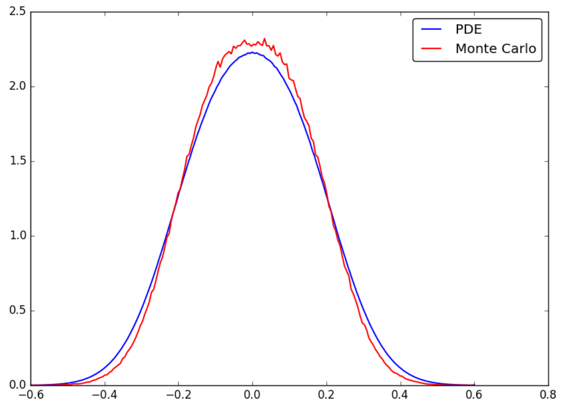
Figure 4.23: Join probability distribution for Harmonic Load with White Noise

Table 4.13: Standard deviation of harmonic load with white noiseresponse

	σ_x	σ_v
Monte Carlo	0.0631	0.155
Cell Mapping	0.0696	0.167
Difference	9.38 %	6.88 %



(a) Displacements

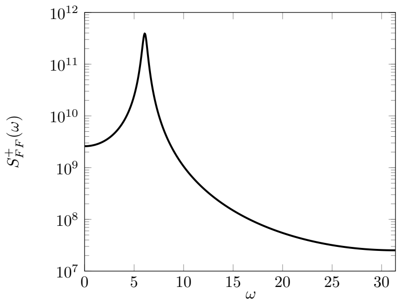


(b) Velocities

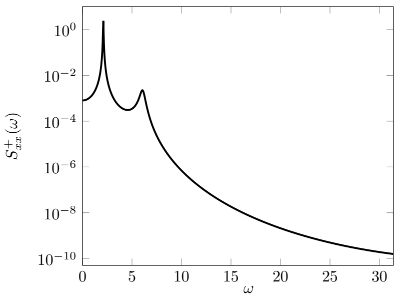
Figure 4.24: Displacement and velocity distribution for Harmonic Load with White Noise

4.5 AR(2) Load with spectral peak

With the autoregressive parameters ϕ_1, and ϕ_2 chosen in Table 4.14, the process will oscillate, with a frequency $\omega_f = 6.1\text{rads}^{-1}$ (Eq 2.20). This is above the natural frequency of the system, meaning that the response will contain two dominating frequencies. (see Figure 4.25) The force will in this section be mapped with a the modified noise term, and with a truncated discretized white noise term with 25 discrete values. Section 4.3 shows that it is possible to estimate the response with the modified noise term derived in Section 3.6.2. For the second order autoregressive load, the modified noise term will be compared with a normal distributed noise term, to see the difference in performance. The modified



(a) Load Spectrum



(b) Response Spectrum

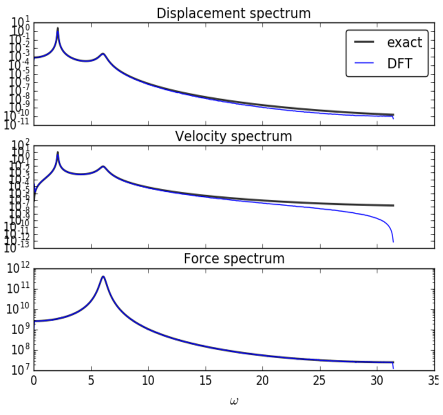
Figure 4.25: Exact Spectral Distributions of AR(2) load and response

Table 4.14: AR(2) Load Model Parameters

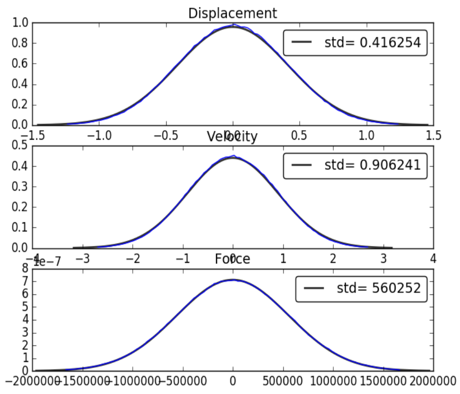
ϕ_1	=	1.6
ϕ_2	=	-0.95
σ_W	=	100 kN
Δt	=	0.1 s
λ	=	3
n_f	=	25
n_x	=	200
n_v	=	200
ψ	=	3.83
ξ	=	3.13

random noise is either mapped with 1 random point, or 1000 random points.

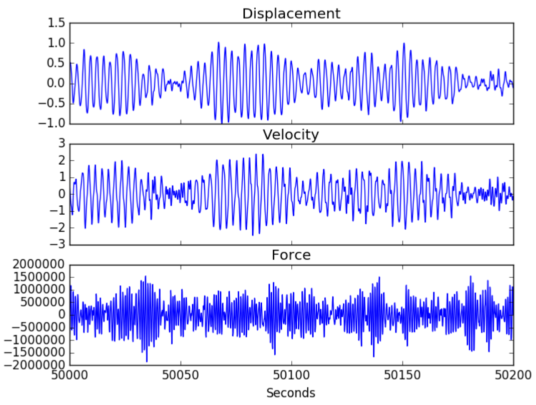
4.5.1 Monte Carlo Simulation of response



(a) Spectral Density Distribution



(b) Probability Density Distribution



(c) Time domain response and force sample

Figure 4.26: Results from 28h Monte Carlo Simulation of Response from AR(2) load

The Monte Carlo simulation shows that for the chosen time step, the spectral densities, and the variances are according to theory. In the following the assumption that the autoregressive process presented in Table 4.14 is a Gaussian process is introduced, meaning that all the final distributions goes towards a Gaussian normal distributed variable when time increases. The assumption is based on Figure 4.26b after the Monte Carlo simulation of the load.

4.5.2 Force Evolution

The parameters ψ and ξ for the force are found minimizing the residual in shape, and variance according to Algorithm 2. Since the assumption that the process is Gaussian is introduced, the discrete force model is compared to a Gaussian normal variable with variance according to Eq. 2.18. The other force evolution is done by a truncated discretized white noise term with no modification.

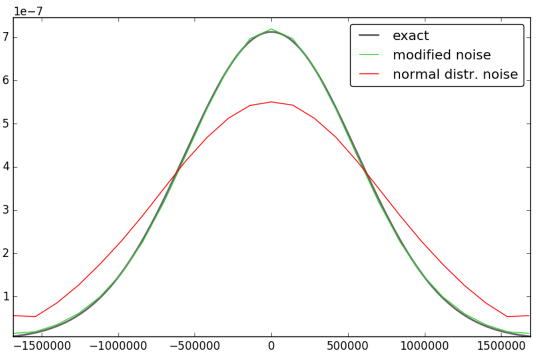
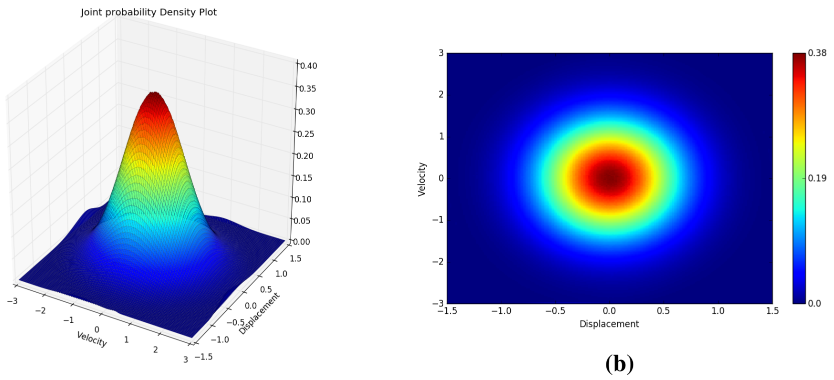


Figure 4.27: Steady state AR(2) Force distribution

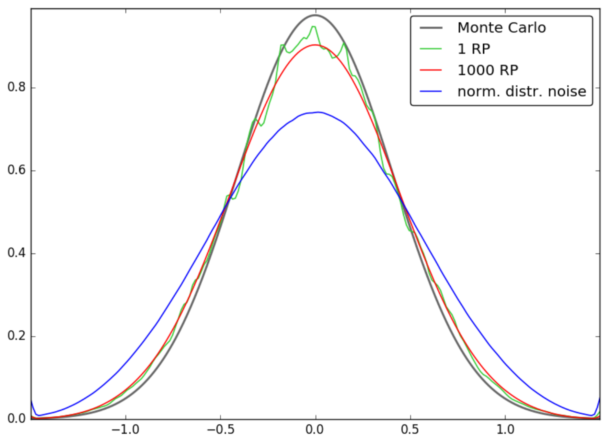
4.5.3 Cell Mapping Results



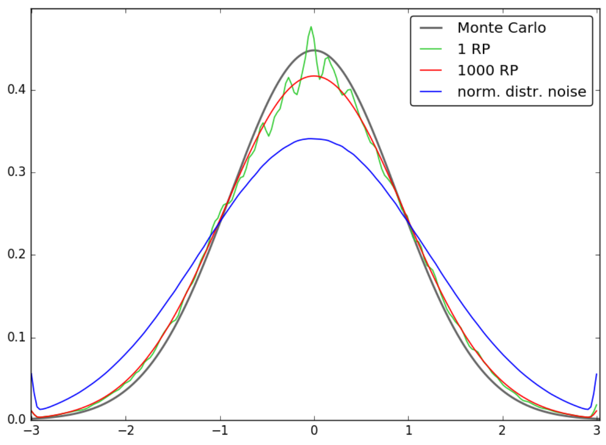
(a) **(b)**

Figure 4.28: Joint Distr. of Steady State AR(2) Cell Mapping Response with modified noise

Figure 4.29 shows the different steady state displacements. The distribution of the modified added noise comes closest to reproduce the expected steady state response distribution, while the truncated discrete normal distributed noise is much more diffusive.



(a) Displacement



(b) Velocity

Figure 4.29: Steady State of AR(2) Cell Mapping Response with modified noise

Table 4.15: Standard deviation of AR(2) Load response

	σ_x**[m]**	σ_v**[m]**
Monte Carlo	0.4094	0.8904
Mod.Noise(1RP)	0.4321	0.9383
Mod.Noise(1000RP)	0.4374	0.9484
Norm.Noise	0.5204	1.1313

4.6 Two Harmonic loads

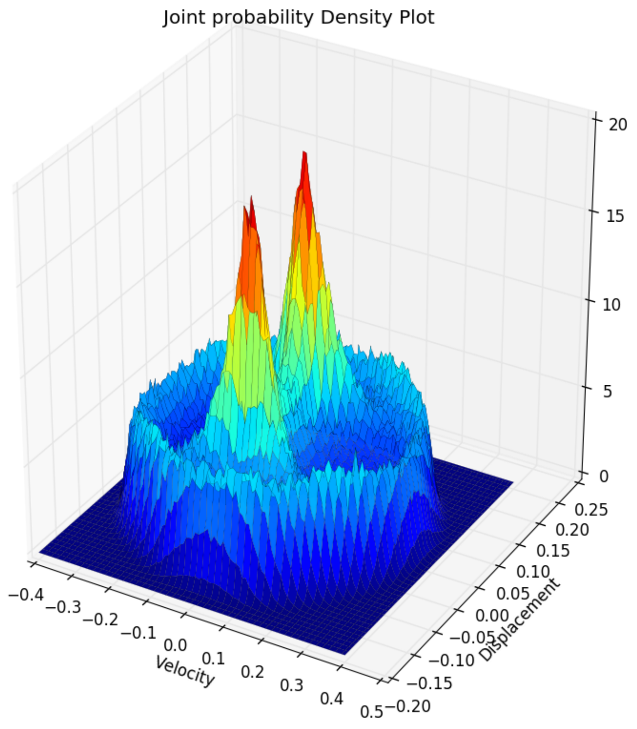


Figure 4.30: 3D illustration of probability distribution from response from 2 harmonic loads using cell mapping

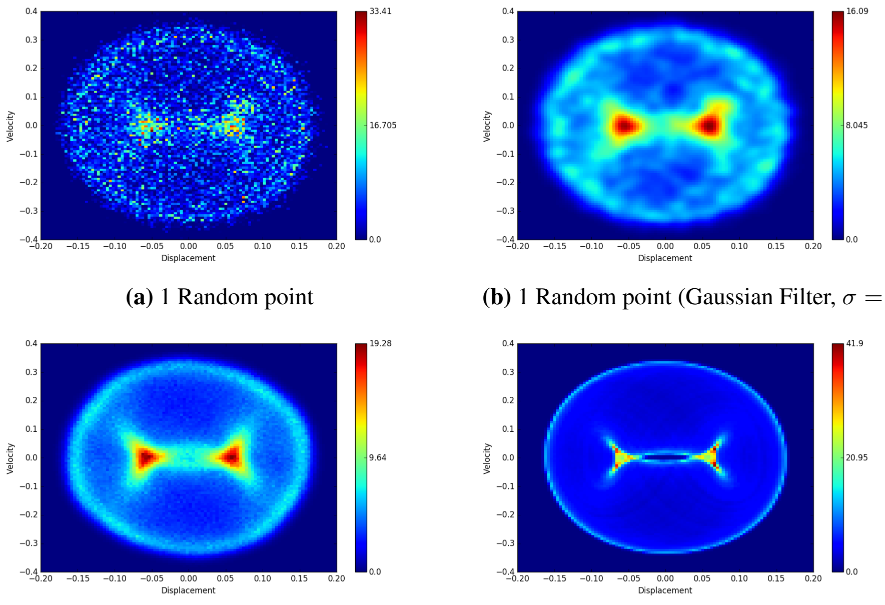
Two harmonic loads acting on the structure could for instance be multiple frequencies of the rotor blade propagating in the structure. Investigating the Single harmonic cell mapping in Section 4.2 shows that a fairly fine discretization of both the displacement and velocities, and the phase is needed to obtain accurate results. With yet another harmonic load case added added to the problem, a new dimension for the phase state is added to the probability space. With the parameters in Table 4.16 the probability space is discretized to 118 Million cells. For such a large probability space, the calculation of the *Transition probability Matrix* will be time consuming. Since the time used to calculate the matrix is expected to grow proportional with number of random points selected in each cell, this

Table 4.16: Two Harmonic Parameters

F_1	=	100 kN
F_2	=	100 kN
ω_{f_1}	=	1.5 rad s^{-1}
ω_{f_2}	=	3 rad s^{-1}
n_{θ_1}	=	107
n_{θ_2}	=	110
n_x	=	100
n_v	=	100
Δt	=	0.67s
Random points	=	1 or 500

test is carried out with either 1 random point, or 500 random points to see what accuracy could be expected with only 1 random mapping point in each cell.

4.6.1 Results



(a) 1 Random point

(b) 1 Random point (Gaussian Filter, $\sigma = 3$)

(c) 500 Random points

(d) 1000 Monte Carlo simulations

Figure 4.31: Steady state joint probability distribution of 2 harmonic loads response

Figure 4.31 shows the joint probability distribution for the cell mapping result as well as

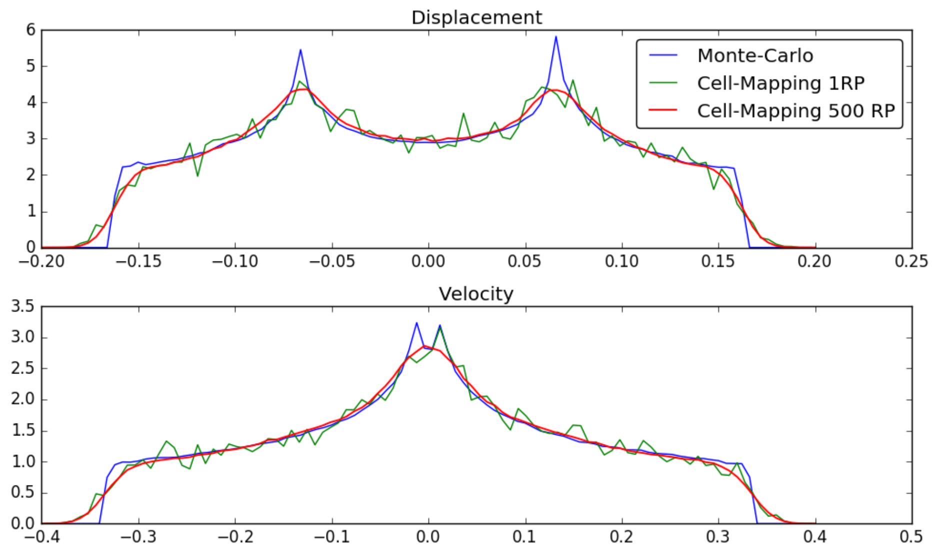


Figure 4.32: Displacement and Velocity probability distribution for 2 harmonic loads response

the Monte Carlo simulations. The steady state is found by iterating the *Transition Probability Matrix*. Since the solution with only 1 random point seem quite noisy. (Fig.4.31a), the result image is convolved with a Gaussian distribution (Gaussian Filtered) in order to smooth the image. While both results acknowledges the two peaks in the distribution, the final distribution is a lot more diffusive than the Monte Carlo simulation. Figure 4.32 shows the displacement and velocity distributions for Monte Carlo simulations and cell mapping with different number of random points.

Chapter 5

Fatigue Calculations

The method described in Section 3.8 will in this chapter be tested. The object is first to observe how a half cycle behaves during iteration with the *Transition Probability Matrix*. The results from the half-cycle mapping will be extracted and used to create the peak-to-trough transition matrix, and the peak-trough intensity $\mu^{PT}(u, v)$ will be found. The last step is to obtain the *Damage Intensity* and see how the results obtained with the peak-to-trough matrix are compared with results from the WAFO toolbox [5] which extracts the damage calculations from a generated time series with the theory presented in Section 2.6 .

5.1 White noise Load, narrow banded response

Table 5.1: Fatigue test Model Parameters

S_0	=	5 GN^2/Hz
ϕ	=	0
σ_W	=	800 kN
Δt	=	0.05 s
n_x	=	301
n_v	=	301
n_F	=	20
λ_x	=	4
λ_v	=	4
λ_F	=	3
no. random points	=	10

The Model parameters used in this section is shown in Tab 5.1. This is a flat load spectrum similar to the one used in Section 4.1. The difference is basically that the intensity of the load is higher, and the response state discretization is different. The range is also wider for response, meaning that the model describes more states at the tails of this distribution.

The response of a low damped structure with a flat load spectrum is often considered to be a narrow banded process, due to the dominant response around the eigenfrequency. Figure 5.1 suggests that this could be considered as a good estimation for the current system, and seems to be generally true for most excitations.

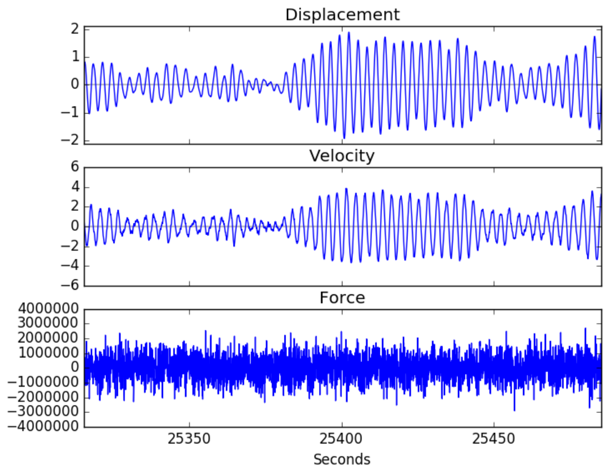
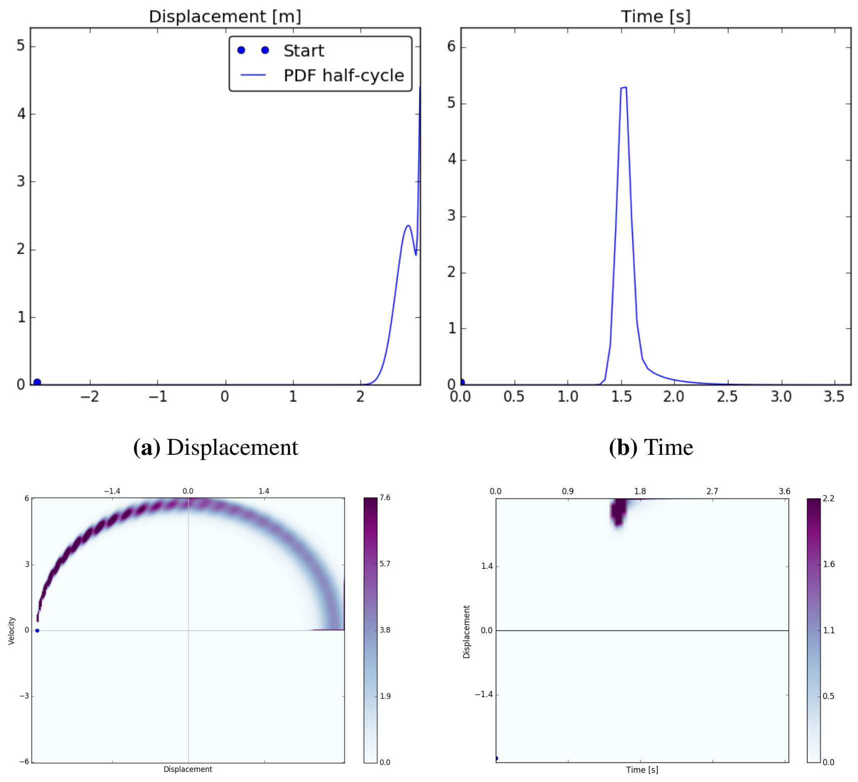


Figure 5.1: Response and force signal in time domain from load described in Table 5.1

5.1.1 Half-cycle iteration

Figure 5.2 and Figure 5.3 shows how a half cycle is presented with the transition probability matrix. The difference with starting the iteration at $x_0 = -2.8m$ and at $x_0 = -2.0m$ is that a large amount of probability hits the boundary when an iteration is started close to the boundary of the displacement state. While the undisturbed probability distribution can be seen in figure 5.3 which looks close to a Gaussian distribution in both displacement and time, the boundary is disturbing the distribution in Figure 5.2. Both in displacement where a large portion of probability is absorbed in the outer cell, but also the time distribution get a trailing tail which differs from the undisturbed cycle. This is further seen in the *Peak Transition Matrix* visualised in Figure 5.4 where there is a higher intensity at the top left, and bottom right corner.



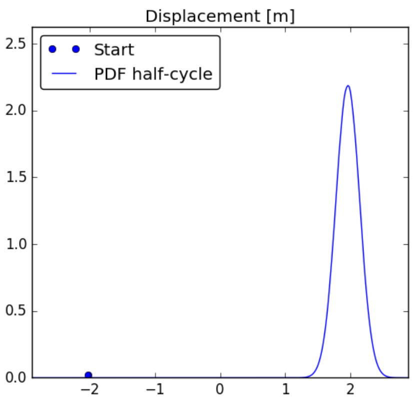
(a) Displacement

(b) Time

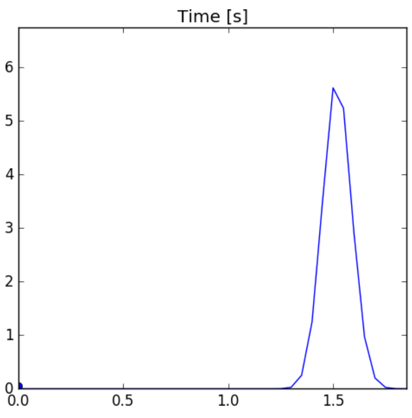
(c) Half-cycle path iteration

(d) joint PDF for peak displ, and time

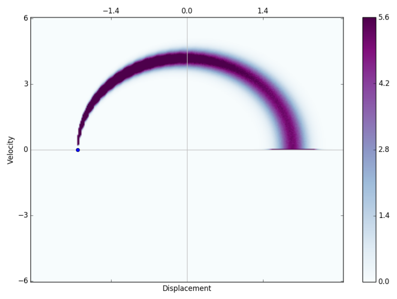
Figure 5.2: Result from half cycle iteration with white noise load spectrum, (starting from $x_0 = -2.8m$)



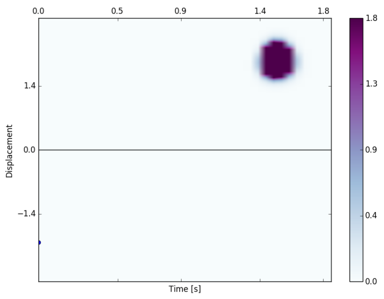
(a) Displacement



(b) Time



(c) Half-cycle path iteration



(d) joint PDF for peak displ, and time

Figure 5.3: Result from half cycle iteration with white noise load spectrum, (starting from $x_0 = -2.0m$)

5.1.2 Peak Transition Matrix

The results from the half cycle iteration is stored in the *Peak Transition Matrix*, where each row represent the displacement distribution from a peak state u_i.

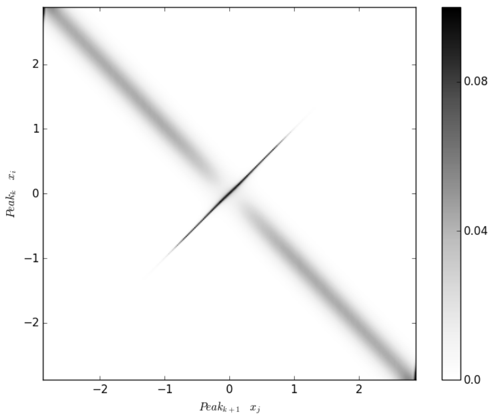


Figure 5.4: *Peak Transition Matrix*: Probability that peak u_i at time k, is followed by peak u_j, at time $k + 1$

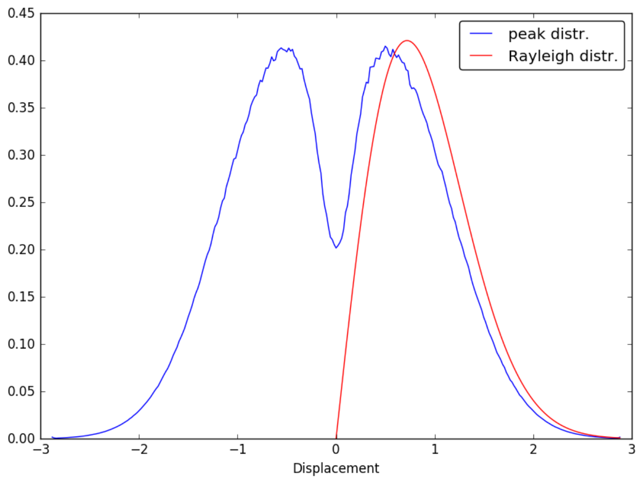
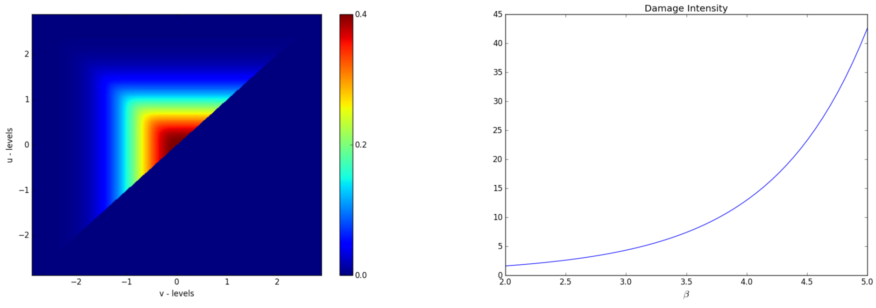


Figure 5.5: Steady state of Peak Transition Matrix, peak distribution (blue), and Rayleigh Distribution of peaks (red)

5.1.3 Rainflow Intensities and Damage intensity



(a) Ranflow intensity **(b)** Damage Intensity

Figure 5.6: Rainflow and Damage Intensity for White Noise Load response

5.1.4 Comparing with WAFO and Narrow banded approximation

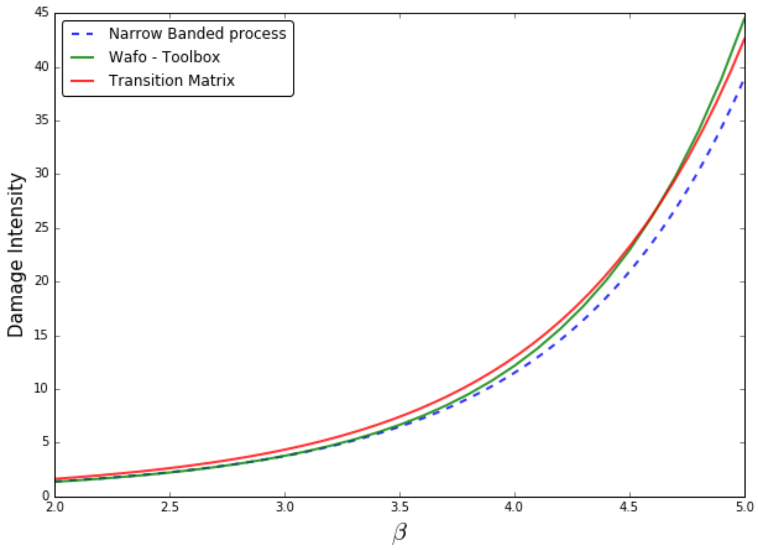


Figure 5.7: *Damage Intensity* curves for various methods

In Figure 5.7 the Damage intensity curves are shown for the different methods. The WAFO-toolbox curve is found by simulating a 10h long time sample, where a rainflow count is performed and the damage is calculated according to the result of this rainflow count. The Narrow banded process curve is a realisation of Eq. 3.67 where the damage is estimated from the Rayleigh distribution of peaks. The Transition Matrix curve shows the results from the method in Sec 3.8. Which estimates the damage intensity considering the process as a Markov chain and therefore finds the rainflow intensity without any time domain simulation.

Chapter 6

Discussion

6.1 General Cell Mapping Accuracy and Discretion

All of the results presented in Chapter 4 show that it is possible to obtain precise estimates of the response probability distribution with the general cell mapping algorithm provided.

Timestep

The Runge Kutta Nyström scheme is provided for the linear equation of motion, and different time steps show different accuracy of the integration. While the critical time step is about $t_{cr} = 2/\omega_n$ before the solution becomes unstable, a time step of $t = 1/(2\omega_n)$ is needed to obtain a standard deviation error of $< 1\%$ for a flat load spectrum. For smaller timesteps the final distribution gets more accurate, and the results imply that the theoretical distribution is approached as $t \to 0$. The results show that a time step smaller than t_{cr} should be used when applying a spectrum load.

Cell Discretization

The mapping distance is proportional to the time step used in the Runge Kutta Nyström integration. Since the response displacement, and velocity is discretized into cells, a spurious amount of image points are mapped back to the same starting cell. The results show that the discretization of displacement and velocity into cell states causes the distribution to be wider than the exact distribution. This should be as expected since different states are mapped from a cell with a volume rather than a specific point, and the diffusion is caused by the size of this cell space. The results provided show that when a large amount of probability is mapped back to the same displacement/ velocity state, an artificially wide distribution is acquired.

Since the state mapping is a function of the time step used, and while lengthen the time step causes the final distribution to tighten, and the discretized response space causes the distribution to widen, one could speculate if there is a optimal relation between a chosen time step and the cell discretization. This has not been investigated further in this thesis, where the time step, and response discretization has been made appropriate small.

Domain Boundaries

How large effect the boundaries have on the cell-to-cell mapping solution is interesting, since image points who would leave the domain is placed back on the border again. The steady state solution shows that for a Gaussian process, accurate probability density estimates can be obtained as long as the domain is at least three times the standard deviation wide. When the domain is narrower, an accumulation of probability reaches the edges and disturb the overall probability density function. It seems therefore that as long as the border cells represents state which has a low probability of happening, the spurious image points from these states will not have a large impact on the dynamics of the system.

The effects of the border can also be seen on the half-cycle iteration. Starting the system near an edge causes a larger amount of probability to hit the boundary on the opposite side. Again, as long as these starting states are extremas with low probability of happening, this will not have a large influence in the peak trough,- nor rainflow intensities. Eq. 3.65 shows that the counting intensities are found by multiplying with the peak distribution vector.

Steady State

Two methods has been used to find the steady state of a *Transition Probability Matrix*. While the eigenvector with a eigenvalue of 1 is mathematically correct and the most elegant solution to the problem, for large matrices it computationally heavy to find. The iteration scheme presented in this thesis on the other hand is faster, but requires an idea of how many iterations is needed to reach steady state. The results in this thesis show that there is a linear convergence rate for the iteration scheme, and the statistical properties converges faster than the transient decay. This means that the steady state is obtained efficiently, and reveals one of the differences between Monte Carlo simulations. While for a Monte Carlo simulation one can start sampling after the transient behaviour vanishes, the probability density evolution provides the steady state at once the transient behaviour has vanished. Of course each iteration with the *Probaility Transition Matrix* is more demanding than forecasting the next values in a Monte Carlo simulation, but this illustrates one of the advantages with the probability density evolution method compared to time simulations.

Load Model Description

Two different types of load have been tested. There is the autoregressive representation of a stochastic load, and the harmonic load represented with a random phase. For the autoregressive load, the force space is represented with a relatively coarse mesh. For a flat spectral load density, the force distribution is equal to the added noise. The force distribution is therefore only dependent on the truncation and discretization of the added noise.

The first and second order autoregressive load model is a linear combination of previous step(s), and a diffusive added noise. The results might indicate that discretizing the force space leads to larger diffusion of the probability distribution. As a remedy the added normal distributed noise is changed to a modified noise, where the variance of this noise is changed to fit the finial force distribution. The results in this thesis shows that when the distribution of the discretized force model is close to the continuous force distribution an accurate result of the response displacement and velocity is obtained. This result is surprising since changing the variance of the added noise would change the intensity of the load and alter the stochastic process. This means that the process has to be transformed when changing the state space from a continuous space to a discrete space, and this transform has to be made in order to get the same statistical properties. However, going back to a continuous space, the process has to be transformed back to its original stated form since the modified added noise has different statistical properties as the normal distributed added noise. The results show that a discrete force model must be modelled with lower intensity in order to get the same statistical properties of the final load distribution. However, more investigation should be put in to understand how the transform between continuous space to discrete space is related.

The harmonic load has a more complicated probability distribution, and needs a finer discretization of the phase in order to have good accuracy of the response distribution. The single harmonic load results have shown that the Runge Kutta Nystrm method has a critical time step around $t_{cr} = 2/\omega$, and that the most accurate results are obtained for a time step close to the critical time step. This contradicts the results for a optimal time step for a flat load spectrum, and seems therefor only valid for a load with a single frequency component. For finer response and random load phase discretization, the results seem to improve gradually,but since the distribution goes towards infinity at the peaks, it will be impossible to recreate the exact solution with discretized displacement, and velocities.

6.2 Fatigue

The results from the cycle count method will be discussed in this section. In Chapter 3, the method is developed in order to acquire results considering the peak to trough process as a Markov chain, and the *Peak Transition Matrix* is found by iterating the *Probability Transition Matrix* from one extrema to the next. The result are then compared to a theoretical damage curve considering the signal as a narrow banded process, and by generating a time signal from the response spectrum and doing the rainflow counting with a MATLAB toolbox [5] (Appendix C). The Rainflow intensity shown in Figure 5.6a is found from Markov transition between states, and therefore without any simulation in time. The method is described in Section 3.8.

Peak Transition Matrix

An image of the peak transition matrix is shown in Figure 5.4 and is a result of the half cycle iteration performed in Section 5.1.1. The image shows that the narrow banded approximation is quite precise, since the transition mainly changes the sign of the peak. The long diagonal starting at the upper left corner describes the mapping from a peak to an opposite trough, while the diagonal at the middle of the figure describes the transition back to the starting base point, which contradicts the narrow banded approximation. This is then also seen in the peak distribution (Fig 5.5). where the Rayleigh distribution of narrow banded peaks estimates larger densities for higher peaks, while the cell mapping peak distribution acknowledges the fact that an extrema close to zero does not always change sign, which leads to a distribution with higher densities close to zero. The difference between the narrow banded Reyleigh distribution of peaks, and the steady state of the *Peak Transition Matrix* is therefor as could be expected.

Damage Intensity

The *Damage Intensity* curves for the different methods are shown in Figure 5.7. The method of finding the peak to transition matrix seem to agree with the results acquired with the WAFO - toolbox, while the *Damage Intensity* curve for a narrow banded process seem to somewhat underestimate the damage. The differences between the WAFO generated Damage Intensity and the Transition Matrix Damage Intensity curve could be due to several factors. One factor might be the Markov approximation of a peak to trough process, which is assumed. However, in Frandahl and Rychilcs paper [12], they acknowledges that this assumption is generally accurate for a Gaussian process. Another difference is that the peaks represent discrete states in the *Peak Transition Matrix*, and that the *Damage Intensity* is the sum of the rainflow intensity matrix with the damage function, while the rainflow count with the WAFO - toolbox is done in continuous space. Even while the *Peak Transition Matrix* is defined in discrete space, the advantage is that the time information is not needed, and the damage calculations can be done without generating large time series. Due to these differences between the methods, small differences in the results should also be expected.

6.3 Remarks on Computational speed

This thesis has not provided any run time test for the code developed, and the speed of calculation is of course dependent of the hardware available. The computer used in this thesis with the properties described in Table 4.2 manages to map about 2-3 million points per second with Runge Kutta Nyström function (see Listing 1). This could however be done faster by running the code in parallel. For the cases studied in this thesis all cells are mapped within minutes even for the largest problems, while adding more random points increases the time linearly.

The steady state solution is found either by solving the largest eigenvalue of the matrix with Arnoldi method, or iterating with a probability vector until the probability vector reaches a steady state. The statistical properties of the distribution converges faster than the transient behaviour of the system vanishes, and iterating to a steady state is therefore usually faster than solving the eigenvalue problem with the Arnoldi method.

Chapter 7

Conclusion

The results in Chapter 4 show that the structural response of a single degree of freedom system can be described as a Markov chain for several load cases. The load cases tested are autoregressive 1st and 2nd order Gaussian processes, and a harmonic load case with one or two frequency component. The results show that a finer mesh is required for an accurate description of the harmonic load case than for the autoregressive processes. Since the state space is discretized in the force models. The statistical properties of the noise term should be changed in order to keep the statistical properties of the steady state distribution. The Autoregressive formulation of a process is often considered a continuous space - discrete time process, while in this thesis the space is discretized as well. Changing the variance of the noise term acts therefore as a remedy for the errors introduced by the discretization. When adjusting the added noise term to preserve the statistical properties of the load, this thesis shows that the process can be modelled with only a few force states, and 3 discrete noise values. Doing this reduces the number of calculations for the problem, which again makes it possible to look at larger problems in the future.

A restricting factor is the computer capacity in order to solve large matrix operations, and compute large problems. This thesis show that problems with discrete probability space of at least 100 Million states (which is the largest problem solved in this thesis) are solvable with the computer capacity available today. Since it also is possible to reduce the load space with the modified added noise, this opens up the possibility to investigate more complex load models without exceeding the computer capacity. This is due to the modified random noise added to the autoregressive load formulation. While a large set of calculations requires a high end CPU, saving a large system (with many cell states) increases the size of the *Transition Probability Matrix* and requires a large memory (RAM). This becomes even more important for multidegree of freedom systems.

The boundary conditions introduced when calculating the *Probability Transition Matrix* keeping the probability from escaping any defined state are introduces so that the matrix can be considered a stochastic matrix with the properties associated with such a matrix. The results show that the response space should be at least 3 times the standard deviation for a Gaussian response process to acquire results which are not affected by the boundaries in any considerable manner. For the half cycle iterations, it is seen that the boundaries disturb the probability distribution when an iteration is started close to a boundary. However, for damage calculations this does not seem to affect the final damage intensity, since the peak probability close to the border is small.

In Chapter 5 The Damage intensity is found with rainflow counting through a *Transition Matrix* using the probability density evolution method. The method provides accurate results compared with time domain simulation for a low damped structure with a white noise load spectrum. While for other structures and load cases the method has yet to be tested. This result encourages to further investigating random processes where the peak trough process can be considered Markovian with the probability density evolution cell-mapping technique.

Further work

The method for fatigue calculation described in Section 3.8 could be tested for more complex load cases than in this thesis in order to understand the method better.

One extension of this work should be to implement a moving average load in order to describe weakly stationary loads, and fluctuations in the average load applied on a structure. Also, investigating what happens when the state space of an autoregressive model is discretized is not yet very well understood, however in order to do the cell mapping, and the Markov approximation of the system this is essential in order to acquire good results.

When the dynamics of the cell mapping method for stochastic loads is well understood, an extension to a multidegree of freedom system would be natural in order to simulate more complex structures with several eigenfrequencies.

The discretization in the cell mapping method is done in Cartesian coordinates. Another way is to define the response mesh for the displacement and velocities in polar coordinates. This is actually a more natural way of describing the system, since the solution of an damped vibration without an external load is a logarithmic spiral in the velocity-displacement space.

Bibliography

[1] W. E: Arnoldi. Principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9(1):17–29, 1951.

[2] R: B: Bapat and T. E. S Raghavan. *Nonnegative Matrices and Applications*. Cambridge University Press, 1997.

[3] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forcasting and Control (fourth ed.)*. Wiley, 2008.

[4] P. Bremaud. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer, 2013.

[5] P.A. Brodtkorb, P. Johannesson, G. Lindgren, I. Rychlik, J. Rydén, and E. Sjö. WAFO - a Matlab toolbox for the analysis of random waves and loads. In *Proc. 10'th Int. Offshore and Polar Eng. Conf., ISOPE, Seattle, USA*, volume 3, pages 343–350, 2000.

[6] T. Burton, N. Jenkins, D. Sharpe, and E. Bossanyi. *Wind Energy Handbook (2nd. Ed)*. Wiley, 2011.

[7] R.L. Carlson, G.A. Kardomateas, and J.I. Craig. *Mechanics of Failure Mechanisms in Structures*. Springer, 2012.

[8] A. K. Chopra. *Dynamics of structures: Theory and applications to earthquake engineering*. Prentice Hall, 2012.

[9] R. D Cook, D. S. Malkus, M. E Plesha, and R. J. Witt. *Concepts and applications of finite element analysis 4th ed.* John Wiley & sons, 2002.

[10] M. Dellnitz and E. Kreuger. An adaptive method for the approximation of the generalized cell mapping. *Pergamon press*, 8(4):525–534, 1997.

[11] S. D. Downing and D. F. Socie. Simple rainflow counting algorithms. *International Journal of Fatigue*, 4(1):31–40, 1982.

[12] M. Frendahl and I. Rychlik. Rainflow analysis: Markov method. *International Journal of Fatigue*, 15(4):265–272, 1993.

[13] J. M. Hembre. Stochastic analysis of an offshore wind turbine using a simplified dynamic model. Masters thesis, Norwegian University of Science and Technology, Norway, 2014.

[14] J. Holmes. *Wind Loading of Structures (3rd ed.)*. CRC Press, 2015.

[15] C. S. Hsu. *Cell-to-Cell Mapping, A Method of Global Analysis for Nonlinear Systems*. Springer-Verlag, 1987.

[16] P. E. Kloden and E. Platen. *Nummerical Solutions of Stochastic Differential Equations*. Springer, 1992.

[17] A. N. Kolmogorov. *Foundations of the theory of probability*. Dover, 1950.

[18] E. Kreyszig. *Advanced Engineering Mathematics (10th ed.)*. Wiley, 2011.

[19] J. Li and J. Chen. *Stochastic Dynamics of Structures*. John Wiley & sons, 2009.

[20] J. Li, J. Chen, W. Sun, and Y. Peng. Advances of the probability density evolution method for nonlinear stochastic systems. *Probabilistic Engineering Mechanics*, 28:132–142, 2012.

[21] L. C Ludeman. *Random Processes. Filtering, Estimation, and Detection*. Wiley, 2003.

[22] V. Monbeta, P. Ailliota, and M. Prevostob. Survey of stochastic models for wind and sea state time series. *Probabilistic Engineering Mechanics*, 22:113–126, 2007.

[23] A Naess. *An Introduction to random vibrations*. NTNU, 2011.

[24] D. E. Newland. *An Introduction to Random Vibrations, Spectral & Wavelet Analysis*. Dover Publications, 2005.

[25] P. Pinson and H. Madsen. Adaptive modelling and forecasting of offshore wind power fluctuations with markov-switching autoregressive models. *Journal of forecasting*, 31(4):281–313, 2012.

[26] I. Rychlik. A new definition of the rainflow cycle counting. *International Journal of Fatigue*, 9(2):119–121, 1987.

[27] SciPy community. *Scipy Reference Guide*, 0.18.1 edition, 2016.

[28] N. T. Thomopoulos. *Essentials of Monte Carlo Simulation*. Springer, 2013.

[29] S. V. Vaseghi. *Advanced Digital Signal Processing and Noise Reduction*. Wiley, 2008.

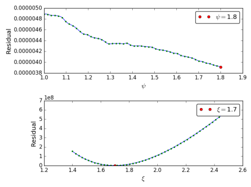
[30] R. E. Walpole, R. H. Meyers, S. L Meyers, and E. Y. Keying. *Probability and Statistics for engineers (9th ed.)*. Prentice-Hall, 2012.

[31] C. Yang. *ARPACK Users' Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. University of Leeds, 1997.

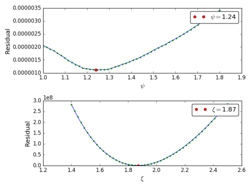
[32] Erns & Young. Offshore wind in europe, walking the tightrope to success, 2015.

Appendix

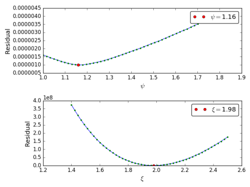
A Modified noise Force Mapping results



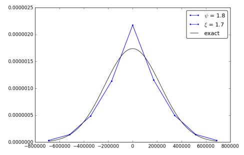
(a) 1st.iteration Residuals



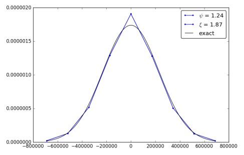
(b) 2nd. iteration residuals



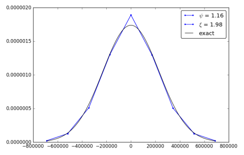
(c) 3rd. iteration residuals



(d) 1st. iteration distribution

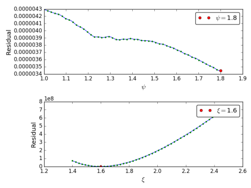


(e) 2nd. iteration distribution

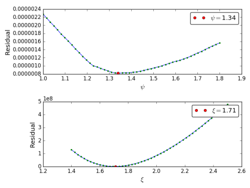


(f) 3rd. iteration distribution

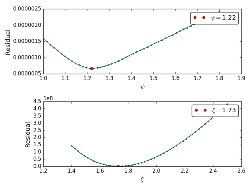
Figure 1: Results after 3 iteration of force parameters starting width $\psi = 1.0$ and $\xi = 1.0$ and 9 force states.



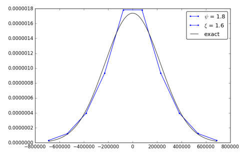
(a) 1st.iteration Residuals



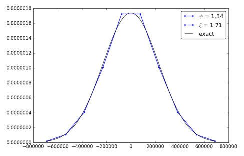
(b) 2nd. iteration residuals



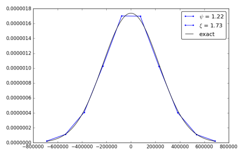
(c) 3rd. iteration residuals



(d) 1st. iteration distribution



(e) 2nd. iteration distribution



(f) 3rd. iteration distribution

Figure 2: Results after 3 iteration of force parameters starting width $\psi = 1.0$ and $\xi = 1.0$ and 10 force states.

B Code: Cell Mapping

```python
# -*- coding: utf-8 -*-
"""
@author: Odd Eiken
Created 2017
"""

@jit('i4(f8)', nopython=True)
def H(x):
    '''
    Heavside step function: returns 0 if x<0, and 1 if x>=0
    '''
    if x >= 0:
        return 1
    else:
        return 0


@guvectorize(
    'void(f8[:], f8[:], f8, f8, f8[:], f8[:])',
    '(n),(n),(),()->(n),(n)')
def Wall(x, v, xlim, vlim, xn, vn):
    '''
    Keeps displ. and vel. within domain
    '''
    for i in xrange(len(xn)):
        if xn[i]**2 > xlim**2 or vn[i]**2 > vlim**2:
            L = max(
                H(abs(xn[i])-xlim)*(abs(xn[i])-xlim)/abs(xn[i]-x[i]),
                H(abs(vn[i])-vlim)*(abs(vn[i])-vlim)/abs(vn[i]-v[i]))
            xn[i] = x[i] + (xn[i] - x[i]) * (1 - L)
            vn[i] = v[i] + (vn[i] - v[i]) * (1 - L)


@guvectorize(
    'void(f8[:], f8[:], f8[:], f8[:])',
    '(n),(n)->(n),(n)')
def Wall_zero_vel(x, v, xn, vn):
    '''
    Limits the velocity to change sign
    '''
    for i in xrange(len(x)):
        if vn[i] / v[i] < 0:
            xn[i] = (x[i] - xn[i]) / (v[i] - vn[i]) * v[i] + x[i]
            vn[i] = 0

```

```python

@jit('f8[:](f8, f8[:])')
def limit_F(F_lim, f):
    '''
    Limits the force from escaping domain
    '''
    for i in xrange(f.size):
        if abs(f[i]) > F_lim:
            f[i] = f[i] / abs(f[i]) * F_lim
    return f
```

Listing 2: Different Boundary functions

```python
# -*- coding: utf-8 -*-
"""
@author: Odd Eiken
Created 2017
"""

import numpy as np
import scipy.stats as ss
from scipy.sparse import csc_matrix
import time
import cell_discretization as cell


def Pmatrix(mesh, par, cells='all', Rand_P=10, zero_vel='no'):
    '''
    cells : which cells to map
    Rand_P : number of Random points in each cell
    zero_vel : 'yes', 'no': creates a boundary at v=0 if yes.
    mesh :dict that specify cell centrep. (see cell_discretization.py)
    par :dict containing defined parameters. (see cell_discretization.py)
            Default: get from cell_discretization

    '''
    LC = cell.load_case(par)
    print 'Load case: ', LC
    dt = par['dt']
    x = mesh['x']
    v = mesh['v']
    dx = x[1] - x[0]
    dv = v[1] - v[0]
    size = [x.size, v.size]
    ar_load = False
    harm_load = False
    flim = 0
    if LC in [1, 3, 4, 6, 7, 8]: # If load is AR(p)
            ar_load = True
            F = mesh['F']
            AR = par['AR']
            sigma_w = par['sigma_w']
            dF = F[1] - F[0]
            n_F = len(F)
            flim = F[-1] + (F[1] - F[0]) / 2. * .99
            for i in xrange(len(AR)):
                    size.append(F.size)
            if AR.all() == 0:

            size_w = n_F
            W = np.linspace(-3 * sigma_w, 3 * sigma_w, size_w)
            dW = W[1] - W[0]
            WU, WL = W + 0.5 * dW, W - 0.5 * dW
            pW = (ss.norm.cdf(WU, loc=0, scale=sigma_w) -
                 ss.norm.cdf(WL, loc=0, scale=sigma_w))
            pW = pW / np.sum(pW)
        else:
            W, pW = cell.AR_NOISE()
            size_w = 3
    if LC in [2, 4, 5, 7, 8, 9]: # If harmonic load
            harm_load = True
            phi = mesh['phi']
            F0 = par['F0']
            omega_f = par['omega_f']
            dphi = phi[1] - phi[0]
            size.append(phi.size)
            if LC in [5, 8, 9]:
                    phi2 = mesh['phi2']
                    size.append(phi2.size)
                    dphi2 = phi2[1] - phi2[0]
    ndim = len(size)  # number of dimentions
    RKNpar = par['omega_n'], par['zeta'], par['k'], dt
    ncells = 1
    for i in range(ndim):
        ncells = ncells * size[i]
    C = np.arange(ncells, dtype=np.uint32).reshape(size)
    n_it = 1e8  # no. it in each chunk (hardware dependend)
    if ar_load: # (chunks might be solved in parallel!)
        Rand_P = Rand_P * size_w
    if type(cells) is str:  # When all cells are mapped
        numthreads = int(ncells * Rand_P // n_it + 1)
        chunklen = ncells / numthreads
    else:  # When a list of cells are mapped
        numthreads = int(cells.size * Rand_P // n_it + 1)
        chunklen = cells.size / numthreads
    c = np.arange(ncells, dtype=np.uint32)
    if type(cells) is str:
        if cells == 'all':
            cells = c
    chunks = [cells[i * chunklen:min([cells[-1] + 1,
            (i + 1) * chunklen])] for i in range(numthreads)]
    print "Number of chunks: ", len(chunks)
    print 'Number of iterations to be solved in millions: ',
        round(ncells * Rand_P / 1e6, 2)
    rx = np.random.rand(chunklen * Rand_P) - .5
    rv = np.random.rand(chunklen * Rand_P) - .5
    rk = [np.random.rand(chunklen * Rand_P)-.5 for i in xrange(ndim - 2)]

    xlim = size[0] * dx / 2. - .1e-3 * dx
    vlim = size[1] * dv / 2. - .1e-3 * dv
    t0 = time.time()
    it = 0     # number of iterations
    i = 0      # numbers of loop while t < t0
    tt0 = time.time()
    P = csc_matrix((ncells, ncells))
    i = 0
    for chunk in chunks:
        print 'Calculating chunck no. %s of %s' %
              (i + 1, len(chunks))
        t0 = time.time()
        n = len(chunk) * Rand_P
        if ndim == 3:
            X, V, K1 = argwhere3d(size[1], size[2], chunk)
        elif ndim == 4:
            X, V, K1, K2=argwhere4d(size[1],size[2],size[3],chunk)
            X = np.repeat(X, Rand_P)
            V = np.repeat(V, Rand_P)
            X = index_to_value(X, x, rx[:n])
            V = index_to_value(V, v, rv[:n])


        if LC == 1:  # AR(0) or AR(1)
            f0 = np.repeat(K1, Rand_P)
            f0 = index_to_value(f0, F, rk[0][:n])
            f2 = np.tile(W, n / size_w) + f0 * AR[0]
            f1 = 0.5 * (f0 + f2)
            xn,vn,K1n=map(X,V,f0,f1,f2,RKNpar,xlim,vlim,flim,zero_vel)
            xn_idx=find_index(xn+dx/2*(size[0]%2),dx)+size[0]/2
            vn_idx=find_index(vn+dv/2*(size[1]%2),dv)+size[1]/2
            K1n_idx=find_index(K1n+dF/2*(n_F%2),dF)+n_F/2
            cn=C[xn_idx,vn_idx,K1n_idx]
            data_i=np.tile(pW,n/size_w)/Rand_P*size_w


        elif LC ==2: # single harmonic
            phi0 = np.repeat(K1, Rand_P)
            phi0 = index_to_value(phi0, phi, rk[0][:n])
            f0 = F0[0] * np.sin(phi0 )
            f1 = F0[0] * np.sin(phi0 + omega_f[0] * dt / 2 )
            f2 = F0[0] * np.sin(phi0 + omega_f[0] * dt )
            xn,vn,f2=map(X,V,f0,f1,f2,RKNpar,xlim,vlim,flim,zero_vel)
            xn_idx=find_index(xn+dx/2*(size[0]%2),dx)+size[0]/2
            vn_idx=find_index(vn+dv/2*(size[1]%2),dv)+size[1]/2
            K1n=(phi0+omega_f[0]*dt*N)%(2*np.pi)
            K1n_idx = np.int_(K1n / dphi)
            cn = C[xn_idx, vn_idx, K1n_idx]
            data_i = np.ones(n) / Rand_P


        elif LC == 3: # AR(2)
            f0 = np.repeat(K1, Rand_P)
            fneg1 = np.repeat(K2, Rand_P)
            f0 = index_to_value(f0, F, rk[0][:n])
            fneg1 = index_to_value(fneg1, F, rk[1][:n])
            f2 = np.tile(W, n / size_w) + f0 * AR[0] + fneg1 * AR[1]
            f1 = 0.5 * (f0 + f2)
            xn,vn,K1n=map(X,V,f0,f1,f2,RKNpar,xlim,vlim,flim,zero_vel)
            xn_idx=find_index(xn+dx/2*(size[0]%2),dx)+size[0]/2
            vn_idx=find_index(vn+dv/2*(size[1]%2),dv)+size[1]/2
            K2n = f0
            K1n_idx = find_index(K1n + dF / 2 * (n_F % 2), dF) + n_F / 2
            K2n_idx = find_index(K2n + dF / 2 * (n_F % 2), dF) + n_F / 2
            cn = C[xn_idx, vn_idx, K1n_idx, K2n_idx]
            data_i = np.tile(pW, n / size_w) / Rand_P * size_w


        elif LC == 4: # AR(1) or AR(0) + single harmonic
            f0 = np.repeat(K1, Rand_P)
            f0 = index_to_value(f0, F, rk[0][:n])
            f2 = np.tile(W, n / size_w) + f0 * AR[0]
            f1 = 0.5 * (f0 + f2)
            K1n = limit_F(flim, f2)
            K1n_idx = find_index(K1n + dF / 2 * (n_F % 2), dF) + n_F / 2
            phi0 = np.repeat(K2, Rand_P)
            phi0 = index_to_value(phi0, phi, rk[1][:n])
            f0 += F0[0] * np.sin(phi0)
            f1 += F0[0] * np.sin(phi0 + omega_f[0] * dt / 2)
            f2 += F0[0] * np.sin(phi0 + omega_f[0] * dt)
            xn,vn,fn=map(X,V,f0,f1,f2,RKNpar,xlim,vlim,flim,zero_vel)
            xn_idx=find_index(xn+dx/2*(size[0]%2),dx)+size[0]/2
            vn_idx=find_index(vn+dv/2*(size[1]%2),dv)+size[1]/2
            K2n = (phi0 + omega_f[0] * dt ) % (2 * np.pi)
            K2n_idx = np.int_(K2n / dphi)
            cn = C[xn_idx, vn_idx, K1n_idx, K2n_idx]
            data_i = np.tile(pW, n / size_w) / Rand_P * size_w


        elif LC == 5:  # two harmonic loads
            phi0 = np.repeat(K1, Rand_P)
            phi0 = index_to_value(phi0, phi, rk[0][:n])
            K1n = (phi0 + omega_f[0] * dt ) % (2 * np.pi)
```

```
                K1n_idx = np.int_(K1n / dphi)
                phi1 = np.repeat(K2, Rand_P)
                phi1 = index_to_value(phi1, phi2, rk[1][:n])
                K2n = (phi1 + omega_f[1] * dt ) % (2 * np.pi)
                K2n_idx = np.int_(K2n / dphi2)
                f0=(F0[0]*np.sin(phi0) +
                    F0[1] * np.sin(phi1))
                f1=(F0[0]*np.sin(phi0+omega_f[0]*dt/2) +
                    F0[1]*np.sin(phi1+omega_f[1]*dt/2))
                f2=(F0[0]*np.sin(phi0+omega_f[0]*dt) +
                    F0[1]*np.sin(phi1+omega_f[1]*dt))
                xn,vn,fn=map(X,V,f0,f1,f2,RKNpar,xlim,vlim,flim,zero_vel)
                xn_idx = find_index(xn+dx/2*(size[0]%2),dx)+size[0]/2
                vn_idx = find_index(vn+dv/2*(size[1]%2),dv)+size[1]/2
                cn = C[xn_idx, vn_idx, K1n_idx, K2n_idx]
                data_i = np.ones(n) / Rand_P


            col_i = np.repeat(chunk, Rand_P)  # column idx
            row_i = cn.flatten()  # row idx
            P_i = csc_matrix((data_i, (row_i, col_i)),
            shape=(ncells, ncells))
            P += P_i
            it += n
            t = time.time() - t0
            i += 1
            print 'Time used calculating chunk: %s seconds.' %(round(t, 2))
        print 'loop took: ', time.time() - tt0
        print 'Monte carlo points in each cell: ', int(Rand_P)
        print 'Step took %s seconds' % (time.time() - t0)
        print
        print 'Total iterations done in millions: ', it * 1e-6
        return P
```

Listing 3: Transition Matrix

```python
# -*- coding: utf-8 -*-
"""
@author: Odd Eiken
Created 2017
"""


import numpy  as np
from scipy.sparse import csc_matrix


def boundary_matrix(P, Rand_P, mesh, par):
    '''
    P : probability matrix
    Rand_P : number of simulations for boundary cells
    mesh : x, v, F : arrays of centerpoints in cells
    par : [omega_n, zeta, k, dt, AR1, sigma_w]
    '''
    def find_crossing_cells(P, mesh):
        ndim = len(mesh.keys())
        x = mesh['x']
        v = mesh['v']
        del mesh['x']
        del mesh['v']
        size = [x.size, v.size]
        for key in mesh.keys():
                size.append(mesh[key].size)
        mesh['x'] = x
        mesh['v'] = v
        ncells = np.prod(size)
        v_neg = np.arange(size[1] / 2 - 1 + size[1] % 2)
        v_pos = np.arange(size[1] / 2 + 1, size[1])

        C = np.arange(ncells).reshape(size)
        if ndim == 3:
                cneg = C[:, v_neg, :].flatten()
                cpos = C[:, v_pos, :].flatten()
        elif ndim == 4:
                cneg = C[:, v_neg, :, :].flatten()
                cpos = C[:, v_pos, :, :].flatten()

        c_sign = np.zeros(ncells, dtype=np.int8)
        c_sign[cneg] = -1
        c_sign[cpos] = 1
        row_ind = P.indices
        col_ptr = P.indptr

        boundry_cells = []    # Cells states that crosses zero vel.
        for i in cneg:  # looping through all negativ velocity states
            cn = row_ind[col_ptr[i]:col_ptr[i + 1]]
            if (c_sign[cn] == -1).all():
                pass
            else:
                boundry_cells.append(i)

        for i in cpos:  # looping through all positive velocity states
            cn = row_ind[col_ptr[i]:col_ptr[i + 1]]
            if (c_sign[cn] == 1).all():
                pass
            else:
                boundry_cells.append(i)
        print ('crossing cells in Matrix [percent]: ',
               round(len(boundry_cells) / float(ncells) * 100., 2))
        return np.array(boundry_cells)

    def find_zero_cells(mesh):
        '''
        returns cells where velocity is zero
        '''
        ndim = len(mesh.keys())
        x = mesh['x']
        v = mesh['v']
        del mesh['x']
        del mesh['v']
        size = [x.size, v.size]
        for key in mesh.keys():
                size.append(mesh[key].size)
        mesh['x'] = x
        mesh['v'] = v
        ncells = np.prod(size)
        v_zero = np.arange(size[1] / 2 - 1 + size[1] % 2, size[1] / 2 + 1)
        C = np.arange(ncells).reshape(size)
        if ndim == 3:
                c_zeros = C[:, v_zero, :].flatten()
        elif ndim == 4:
                c_zeros = C[:, v_zero, :, :].flatten()
        return c_zeros

    def Identity(n, c):
        '''
        Creates an identity matrix with n elements and ones placed at c
        '''
        cols = c
        rows = c
        data = np.ones(c.size)
        ID = csc_matrix((data,(rows, cols)),shape=(n, n), dtype=np.int8)
        return ID

    n, n = P.shape
    print 'finding crossing cells...'
    b = find_crossing_cells(P, mesh)
    Ib = Identity(n, np.delete(np.arange(n), b))
    z = find_zero_cells(mesh)
    Iz = Identity(n, np.delete(np.arange(n), z))
    Izb = Identity(n, z)
    Pb = Pmatrix(mesh, par, cells=b, Rand_P=Rand_P, zero_vel='yes')
    return (Pb + P * Ib) * Iz + Izb
```

Listing 4: Modified Transition Matrix $\hat{\mathbf{P}}$ (Algorithm 3)

C Code: Cycle counting

```python
# -*- coding: utf-8 -*-
"""
@author: Odd Eiken
Created 2017
"""


import numpy as np

def extrema_transition(file, b_iter=1000):
    '''
    returns extrema transition matrix for x values from X_max -> -X_min
    '''
    mesh, par, P_init = load_sparse_csc(file)
    ndim = len(mesh.keys())
    x = mesh['x']
    v = mesh['v']
    del mesh['x']
    del mesh['v']
    size = [x.size, v.size]
    for key in mesh.keys():
        size.append(mesh[key].size)
    mesh['x'] = x
    mesh['v'] = v
    ncells = np.prod(size)
    n = len(x)
    E = np.zeros((n, n))
    Pb = boundary_matrix(P_init, b_iter, mesh=mesh, par=par)
    v0_idx = np.argmin(abs(v))
    if ndim == 3:
                 # cells along x - axis
        cx = np.arange(ncells).reshape(size)[:, v0_idx, :]
        cx = cx.flatten()
    elif ndim == 4:
                # cells along x - axis
        cx = np.arange(ncells).reshape(size)[:, v0_idx, :, :]
        cx = cx.flatten()
    for j in np.arange(n):
        x0_idx = j
        if ndim == 3:
                 # cells at starting point
            c0 = np.arange(ncells).reshape(size)[x0_idx, v0_idx, :]
            c0 = c0.flatten()
        elif ndim == 4:
                 # cells at starting point
            c0 = np.arange(ncells).reshape(size)[x0_idx, v0_idx, :, :]
            c0 = c0.flatten()
        p = np.zeros(ncells)
        p[c0] = 1. / len(c0)
        p = P_init * p
        max_it = 1000
        tol = 5.e-5
        PXT = np.zeros((max_it, n))
        if ndim == 3:
                r_shape = size[0], size[2]
                axis = [1]
        elif ndim == 4:
                r_shape = size[0], size[2], size[3]
                axis = [2, 1]
        for i in xrange(max_it):
            p = Pb * p
            px = p[cx].reshape(r_shape)
            for a in axis:
                    px = np.sum(A, axis=a)
            PXT[i, :] = px
            p[cx] = 0
            if p.sum() < tol:
                PXT = PXT[:i + 1, :]
                break
        PX = np.sum(PXT[np.arange(i), :], axis=0)
        E[:, j] += PX / np.sum(PX)
        print 'iterations: ', i
        print 'cell number: ', j
        print
    fn = '../PROGRAM - Eddinburgh/saved/matrix/%s' % (file)
    np.savez(fn + '_extrema_full', mesh=mesh, par=par, Matrix=E)
    return E
```

Listing 5: Peak Transition Matrix $\mathbf{P}^{\mathbf{P}}$ (Algorithm 3

```python
# -*- coding: utf-8 -*-
"""
@author: Odd Eiken
Created 2017
"""


import numpy as np
from numba import jit

@jit
def PT_intensity(E):
        '''
        E : Extrema - to Extrema matrix
        '''
        n, n = E.shape
        p_p = power_method(E)
        p_p = gaussian_filter(p_p, 0)
        P = np.zeros(E.shape)
        INT = np.zeros(E.shape)
        for i in range(n):  # P[u=x_i, v=x_j]
            P[i, :i] = E[i, :i] * p_p[i]
        for i in range(n):  # p(x>u, y<v)
            for j in range(i + 1):
                    INT[i, j] = np.sum(P[i:, :j + 1])
        return INT




def rainflow_int(INT_PT, E):
    '''
    Algorith finding the rainflow intensity matrix
    from the transition matrix and peakt-trough intensity matrix
    '''
    n, n = E.shape
    I = np.zeros((n, n))
    for i in range(n):
        I[i, n - 1 - i] = 1
    I = np.matrix(I)
    M = np.matrix(E)
    E = (I * M * I)     #Ematrix maps with matrix x=[max_x->min_x]
    P = np.zeros(E.shape)
    P_hat = np.zeros(E.shape)
    for i in range(n):
        P[i, i:] = E[i, i:]
        P_hat[i, :i + 1] = E[i, :i + 1]
    INT_PT = I * np.matrix(INT_PT) * I
    INT_RFC = np.zeros(INT_PT.shape)

    @jit
    def A_matrix(i, j, P):
        return np.matrix(P[i:j, i + 1:j + 1])

    @jit
    def B_matrix(i, j, P_hat):
        return np.matrix(P_hat[i + 1:j + 1, i:j])

    @jit
    def p_vector(j, P):
        n, n = P.shape
        p = np.zeros(n)
        l = np.arange(j + 1, n)
        for k in range(n):
            p[k] = np.sum(P[k, l])
        return np.matrix(p).T

    @jit
    def e(i, j, P):
        return p_vector(j, P)[i:j]

    @jit
    def q_vector(i, j, int_pt):
        l = np.arange(i + 1, j + 1)
        return np.matrix(int_pt[i, l - 1] - int_pt[i, l])

    for i in xrange(n):
        print i
        for j in xrange(i, n):
            A = A_matrix(i, j, P)
            B = B_matrix(i, j, P_hat)
            q = q_vector(i, j, INT_PT)
            INT_RFC[i, j] = INT_PT[i, j] +
            q * B * np.linalg.inv(np.eye(j - i) - A * B) * e(i, j, P)
    return np.array(I * INT_RFC * I)
```

Listing 6: Peak Trough-, and Rainflow Intensity

```python
# -*- coding: utf-8 -*-
"""
@author: Odd Eiken
Created 2017
"""


import numpy as np


def damage(INT, file):
    INT = np.array(INT)
    mesh, par, E = load_extrema_matrix(file)
    n, n = E.shape
    x = mesh['x']
    dx = x[1] - x[0]

    def fdd(u, v, beta):  #second derivative of damage function
            return -beta * (beta - 1.) * (u - v)**(beta - 2.)

    steps = 50
    damage = np.zeros(steps)
    beta = np.linspace(2, 5, steps)
    i = 0
    for beta_i in beta:
        FDD = np.zeros((n, n))
        for vi in range(n):
            for ui in range(vi, n):
                u = x[ui]
                v = x[vi]
                FDD[ui, vi] = fdd(u, v, beta_i)
        I = 0  #integral sum
        M = FDD * INT
        for j in range(n):  # summing over all entities in matrix
            for k in range(j):
                I += M[j, k] * dx**2

        damage[i] = - I  # Damage for beta_i
        i += 1
    return beta, damage
```

Listing 7: Damage function

```
% Script written to find damage curve from rainflow counting
% using the WAFO - toolbox

% written by: Sebastian Shafhirt
% edited by: Odd Eiken



close all
clear all
clc
tic

%% INPUT
    % Time series
        T = 2000;                   % Simulation time
        dt = .05;                   % Time step
        Ns = T/dt+1;                % Number of samplings
        t = (0:dt:T);               % Time vector
        n =T / 2;                   % Number of wave components
        phase = 2*pi*rand(1,n);     % Random phase

    % Damage Calculation
        beta = (2:0.05:5);
        %beta = 5;
        nb = length(beta);

%% Generate time series from spectrum

    %  Frequency vector
        f1 = 0;
        f2 = 1/2/dt;
        df =(f2-f1)/(n-1);
        f = (f1:df:f2) + df.*rand(1,n);

    %   system
        b = f / (2.1 / 2 / pi);
        zeta = 0.01;
        k = 1.8e6;
        S0 = (8e5)^2*0.05/pi;
        AR1 = 0.;
        sigma_w = 5e8;
    % White Noise Response Spectrum
        S = S0 * 2 * pi * (k.^2*((1-b.^2).^2+(2*zeta*b).^2)).^(-1);
        A = sqrt(2*S_AR*df);    % Amplitude for each wave component
    % Wave elevation time series
        for i = 1:length(t)
```

```
            x(i) = sum(A .* cos(2 * pi * f*t(i) + phase));
        end
        disp('Signal Generated, log size:')
        disp(log10(size(t,2)))

%% Damage
        % t = gather(t);
        % x = gather(e);

    % RFC
        tp = dat2tp([t',x']);
        RFC = tp2rfc(tp);

    % Total damage - WAFO
        fb = power(ones(1,nb)*2,beta); % factor to match cc2dam func.
        DRFC = cc2dam(RFC,beta);
        dRFC = fb.*DRFC/T;

    % Total damage - Script
        nRFC = length(RFC);
        D = sum(power(repmat((RFC(:,2)-RFC(:,1)),1,nb),
            repmat(beta,nRFC,1)))/T;

    % Save to csv
        csvwrite('Damage.csv', D)
        disp('Damage Calculated')


%% Rainflow counting distribution
    minRFC = min(RFC(:,1));
    maxRFC = max(RFC(:,2));
    if minRFC < -4 || maxRFC > 4
        disp('u does not fit')
    end

    u = (4:-0.1:-4);
    nu = length(u);
    NRFC = zeros(nu);

    for ui = 1:nu
        for vi = ui:nu
            for i = 1:nRFC
                if RFC(i,2) > u(ui) && u(vi) > RFC(i,1)
                    NRFC(ui,vi) = NRFC(ui,vi) + 1;
                end
            end
        end
```

```matlab
    end
    NRFC = NRFC/T;


%% Plots

        figure
        plot(beta,D)
        title('Damage intensity as function of \beta')
        xlabel('\beta')
        ylabel('intensity')


toc
```

Listing 8: Damage function from WAFO - Toolbox (MATLAB)

D Code: Other

```python
# -*- coding: utf-8 -*-
"""
@author: Odd Eiken
Created 2017
"""


import numpy as np
import time

def iterate_steadystate(Matrix, p0=0, max_it=0, tol=0):
        '''
        Matrix : Probability matrix
        max_it : maximum number of iterations.
                 Default 10x vector lenght
        tol : how big change in next state is tolerated.
              Default = nummerical precision.
        '''
        n, n = P.shape
        print "size: ", n
        if max_it == 0:
                max_it = n * 10
        if tol == 0:
                tol = 1e-5
        t0 = time.time()
        if type(p0) == int:
                pi1 = np.random.rand(n)
                pi = pi1 / np.sum(pi1)
        else:
                pi = p0
                pi1 = np.ones(n)
        e = np.zeros(max_it / 2)
        n = norm(pi - pi1, 1); i = 0
        print "No. Iterations:"
        while n > tol and i < max_it / 2:
                pi1 = P.dot(pi)
                pi = P.dot(pi1)
                n = norm(pi - pi1, 1)
                e[i] = n
                i += 1
                if i%50 == 0:
                        print i*2
        t = time.time() - t0
        fig, ax = plt.subplots()
        ax.semilogy(np.arange(e.size) * 2, e)
        ax.set_xlabel('Iterations')
        ax.set_ylabel('error')
        print 'Number of iterations done: ', i*2
        print '%s Iterations took %s seconds' % (i, t)
        return pi
```

Listing 9: Steady state iteration (Algorithm 1)

```python
# -*- coding: utf-8 -*-
"""
@author: Odd Eiken
Created 2017
"""


import numpy as np


def map_to_same(file):
    '''
    Takes a Transition matrix and finds how much probability
    is mapped into the same (x, v) - state
    '''
    mesh, par, P = load_sparse_csc(file)
    x = mesh['x']
    v = mesh['v']
    F = mesh['F']
    indptr = P.indptr
    indices = P.indices
    data = P.data
    p = np.zeros(x.size * v.size)
    n_F = F.size
    for i in xrange(len(indptr) - 1):
        L = indptr[i]
        U = indptr[i + 1]
        same_state = np.arange(n_F) + i - i % n_F
        ind = indices[L: U]
        dat = data[L: U]
        for j in xrange(U - L):
            if ind[j] in same_state:
                p[i / n_F] += dat[j] / n_F
    return p.reshape(x.size, v.size)
```

Listing 10: Finding ammount mapped to same displ. Vel. state

```python
# -*- coding: utf-8 -*-
"""
@author: Odd Eiken
Created 2017
"""


import numpy as np
from scipy.ndimage import gaussian_filter

def data(p, mesh, smoothen=0):
    ndim = len(mesh.keys())
    x = mesh['x']
    v = mesh['v']
    del mesh['x']
    del mesh['v']
    size = [x.size, v.size]
    for key in mesh.keys():
        size.append(mesh[key].size)
    mesh['x'] = x
    mesh['v'] = v
    Q = p.reshape(size)
    if ndim == 4:
        Q = np.sum(Q[:, :, :, np.arange(size[3])], axis=3)
    PXV = np.sum(Q[:, :, np.arange(size[2])], axis=2)
    PXV = gaussian_filter(PXV, smoothen)
    PX = np.sum(PXV[:, np.arange(size[1])], axis=1)
    PV = np.sum(PXV[np.arange(size[0]), :], axis=0)
    return PX, PV, PXV.T
```

Listing 11: Get probability distribution from probability vector