THESIS

DEVELOPMENT OF A WEATHER RADAR SIGNAL SIMULATOR TO EXAMINE SAMPLING RATES AND SCANNING SCHEMES.

by

Ulf P. Schroder

September 2005

Thesis Advisor:	Jeffrey B. Knorr
Second Reader:	Phillip E. Pace

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Development of a Weather Radar Signal Simulator to Examine Sampling Rates and Scanning Schemes.			**5. FUNDING NUMBERS**
6. AUTHOR(S) Ulf Schroder			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

A weather radar signal simulator that produces an output consisting of a vector of I and Q values representing the radar return permits investigation of the performance of different estimators for the weather signal parameters and their sensitivity when varying radar parameters and precipitation models. Although several empirical statistical models are available to describe precipitation behavior, the creation of a physical model enables adaptation to actual data (e.g. rain rate, wind shears) thereby making it possible to apply and examine different scanning schemes, especially rapid scanning schemes. A physical model allows gradual improvements to realism to study the effects on the radar return for different phenomena. A Weather Radar Signal Simulator has been developed in MATLAB. Several different functionalities have been implemented allowing for stepped frequency, multiple PRFs, pulse compression using a chirp, and variation of both weather and radar input parameters. Post processing capabilities include autocorrelation and FFT (for single PRF only); estimation of weather parameters such as reflectivity factor, Z; average doppler, radial velocity, and velocity spread; pedagogical plots including a Phasor plot of phase change over time and a velocity histogram, instantaneous observed reflectivity and power for each pulse over time.

14. SUBJECT TERMS Weather Radar Signal Simulator			**15. NUMBER OF PAGES** 175
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

DEVELOPMENT OF A WEATHER RADAR SIGNAL SIMULATOR TO EXAMINE SAMPLING RATES AND SCANNING SCHEMES

Ulf P. Schroder
Lieutenant Colonel, Swedish Armed Forces
BSSE, Swedish National Defense College, 2003

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2005

Author:	Ulf P. Schroder
Approved by:	Jeffrey B. Knorr Thesis Advisor
	Phillip E. Pace Second Reader
	Dan C. Boger Chairman, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A weather radar signal simulator that produces an output consisting of a vector of I and Q values representing the radar return permits investigation of the performance of different estimators for the weather signal parameters and their sensitivity when varying radar parameters and precipitation models. Although several empirical statistical models are available to describe precipitation behavior, the creation of a physical model enables adaptation to actual data (e.g. rain rate, wind shears) thereby making it possible to apply and examine different scanning schemes, especially rapid scanning schemes. A physical model allows gradual improvements to realism to study the effects on the radar return for different phenomena. A Weather Radar Signal Simulator has been developed in MATLAB. Several different functionalities have been implemented allowing for stepped frequency, multiple PRFs, pulse compression using a chirp, and variation of both weather and radar input parameters. Post processing capabilities include autocorrelation and FFT (for single PRF only); estimation of weather parameters such as reflectivity factor, *Z*; average doppler, radial velocity, and velocity spread; pedagogical plots including a Phasor plot of phase change over time and a velocity histogram, instantaneous observed reflectivity and power for each pulse over time.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First of all I want to express my deepest gratitude to Prof. Jeff Knorr taking upon himself the responsibility of guiding me through a new field of radar science. His thoroughness and expertise has been comforting in times of despair and the trust he has shown in my work and abilities has been a tremendous source of inspiration and motivation.

I would like to direct a special thank you to Prof. Phillip Pace, as the second reader of this thesis but mostly as a provider of safe haven and enormous moral and administrative support during my stay at the Naval Postgraduate School. He has shown great personal interest in me and my family's wellbeing starting the first day of school.

Also, I would like to thank PhD. Marielle Gosset of the Institute de Recherche pour le Developpement, LTHE, Grenoble, France, that contributed by providing the code of a simulator in progress and assistance in developing a final product.

Apart from all knowledge gained meeting fascinating professionals teaching at NPS, I have had the pleasure of getting to know Fabio D. P. Alves, truly the best friend one could ever ask for. Thank you for allowing me to be your friend.

I must mention and thank Mr. Paul Buczynski for all support and inspiration during my time at NPS. Working together with a person with the kind of expertise Paul holds has been very gratifying and I have a lot to thank him for especially concerning the outcome of this thesis. Paul, you are the greatest!

Finally, I would like to thank my beloved wife Marie for running the home and taking care of our family and for accepting the school workload I have been carrying, not leaving much time for the family. Thank you for taking on this adventure with me. My kids, Sofia and Victor, have due to my job been forced to adapt to a new environment learning a whole new language, a task they have mastered splendidly. Kids, you are my inspiration and I thank you for all your support!

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Electromagnetic emission is increasing in all surroundings not only in the military arena making it necessary to understand, interpret, and sometimes filter information created by man or nature. In an attempt to learn more in one research area, studying other areas might provide the insight needed to break new ground. Electromagnetic returns from weather are in one area of radar science regarded as clutter while in another it as a way of gaining information and making estimates and predictions. Exploring the weather side of radar science provides an insight into weather returns as source of information and also improves knowledge about effects that can lead to a better understanding of how to overcome unwanted weather returns.

Creating a weather radar signal simulator that produces an output consisting of a vector of I and Q values representing the radar return, permits investigation of the performance of different estimators for the weather signal parameters and their sensitivity when varying radar parameters and precipitation models. In weather radar applications the return signal samples can be used to estimate reflectivity, velocity and velocity spread. These are the zeroth, first and second moments of the Doppler spectrum. Reflectivity can be used to estimate the rain rate of the specific volume return. A number of mathematical models relating rain rate to reflectivity exist and can be used when an estimated power return is available. There are also a number of measures and statistics for precipitation that have resulted in functions describing rain drop size distribution for a certain rain rate, terminal velocities of rain drops, etc.

Limiting the amount of samples required to accomplish small variance when estimating average power calls for independent samples meaning that the return samples are de-correlated. On the other hand, measuring Doppler calls for coherent, correlated samples, this typically requires a higher pulse repetition frequency (PRF). The velocity spread effects the weather signal correlation time and therefore drives sample time required to arrive at an estimate with specific variance. Although several empirical statistical models are available to describe precipitation behavior, the creation of a physical model enables adaptation to actual data (e.g. rain rate, wind shears) thereby making it possible to apply and examine different scanning schemes, especially rapid scanning schemes, as well as to create output graphical representations that can serve a pedagogical purpose.

This research was undertaken to build a weather radar signal simulator delivering I and Q values from a physical representation of rain drops, which has been accomplished. A previously developed simulator was initially evaluated and verified serving as the basic building block when developing the final version. The Radiation Integrals were used to derive the electromagnetic scattering from rain drops approximated as dielectric spheres. Without accounting for multiple scattering, attenuation, or coupling effects, the scattered electric field from each drop is summed for every pulse providing the RCS of the rain in the radar resolution cell as well as a complex voltage return representation. Using the Marshall-Palmer drop size distribution, an initial position for every drop is determined within an extended radar resolution volume enabling the drops to fall through the radar beam for all samples. Between pulses the drops are moved according to their respective terminal velocities based on the Atlas-Ulbrich approximation and wind. To allow for width in the velocity spectrum, a Gaussian spread is applied to drop velocities that determine how the average velocity is spread. Also a Gaussian spread, that is size dependent, is applied to the velocity and it represents the spread of velocity due to difference in size. The simulator power return is weighted using the Radar Range Equation for point targets, considering all drops as point targets within the resolution cell. The zeroth moment estimate is derived by averaging N power samples.

Several different functionalities have been implemented allowing for stepped frequencies, multiple PRFs, pulse compression using a chirp, and varying input parameters. Post processing capabilities include autocorrelation and FFT (only for single PRF); evaluation of weather parameter estimators such as average reflectivity factor, *Z*; average doppler, radial velocity, and velocity spread. Pedagogical plots including a Phasor plot of phase change over time and a velocity histogram, instantaneous observed reflectivity and power for each pulse over time, have also been implemented.

I. INTRODUCTION

A. BACKGROUND

Electromagnetic emission is increasing in all surroundings not only in the military arena making it necessary to understand, interpret, and sometimes filter information created by man or nature. In an attempt to learn more in one research area, studying other areas might provide the insight needed to break new ground. Electromagnetic returns from weather are in one area of radar science regarded as clutter while others use it as a way of gaining information and make estimates and predictions. Exploring the weather side of radar science will not only provide an insight into weather returns as source of information but also improve knowledge about effects that can lead to a better understanding of how to overcome unwanted weather returns.

Creating a weather radar signal simulator that produces an output consisting of a vector of I and Q values representing the radar return, will permit investigation of the performance of different estimators for the weather signal parameters and their sensitivity when varying radar parameters and precipitation models. In weather radar applications the return signal samples can be used to estimate reflectivity, velocity and velocity spread. These are the zeroth, first and second moments of the Doppler spectrum. Reflectivity can be used to estimate the rain rate of the specific volume return. A number of mathematical models relating rain rate to reflectivity exist and can be used when an estimated power return is available. There are also a number of measures and statistics for precipitation that have resulted in functions describing rain drop size distribution for a certain rain rate, terminal velocities of rain drops, etc.

To minimize the number of samples required to realize small variance when estimating average power independent samples are required meaning that the return samples are de-correlated. On the other hand, measuring Doppler calls for coherent, correlated samples, this typically requires a higher pulse repetition frequency (PRF). The velocity spread effects the weather signal correlation time and therefore drives sample rate. Although several empirical statistical models are available to describe precipitation behavior, the creation of a physical model enables adaptation to actual data (e.g. rain rate, wind shears) thereby making it possible to apply and examine different scanning schemes, especially rapid scanning schemes, as well as to create output graphical representations that can serve a pedagogical purpose. The output signal should therefore correspond to the values expected using the empirical models. The simulator should also enable input of data to create variation in reflectivity, velocity spread and Doppler.

The simulator is based on the reflection of each drop creating a scattered electric field. A Marshall-Palmer drop size distribution is used. The sum of the fields scattered by all drops at any particular instant of time gives the instantaneous scattered field, total radar cross-section and instantaneous received power. Motion is imparted to each drop using a terminal velocity based on drop size and wind field. The motion leads to a rearrangement of drops in the sample volume and thus to a new value of instantaneous received weather signal power at the next sample time. The effect of electromagnetic coupling among drops is investigated.

B. OBJECTIVE

The main objective was to build a weather radar signal simulator that produces an output consisting of a vector of I and Q values of the radar return. From this, average power, Doppler and Doppler spread can be derived. To guide the research and provide tools for verification and validation of the simulator the following questions were addressed:

1. Is it possible to build a weather radar signal simulator based on a physical model of a spatial region containing raindrops?
2. Can realistic motion that depends upon the type of weather system be imparted to the raindrops?
3. Can the output of the weather radar simulator be used to study spatial and temporal sampling schemes for constant or stepped frequency sampling pulses?

4. Can the weather radar simulator be used to study the use of pulse compression and range averaging as a means of rapidly obtaining independent samples?
5. Can the output of the weather radar signal simulator be used to produce displays of pedagogical interest?
6. Can the output of the weather radar signal simulator be used to study the performance of estimators for the first three Doppler moments?
7. Can the output of the weather radar signal simulator be used to study the utility of estimators for higher order Doppler moments?

C. APPROACH

The simulator was developed by building modules that represented a functionality or physical behavior, testing and verifying the module, and then placing the module into the final version of the simulator. Having a prototype from France provided valuable ideas and experience for developing the simulator. Initially this version was translated, evaluated, and respective function or module output was verified against empirical models based on statistics of real life measurements. When all improvements were implemented to the French version, new modules were created aiming at addressing the research questions posed. The final product was optimized to give accurate results structured and presented in a pedagogical manner. During the development Graphical User Interfaces (GUIs) were of secondary concern.

When the simulator was working, i.e. it has passed the necessary testing for verification; the application phase began. Research questions three to seven were then addressed. Results were again compared with empirical models built on statistics of real life measurements. Questions not addressed were formulated for future investigation by others.

Although the subject title implies that meteorology will be extensively covered, the goal of this work was to provide a tool to evaluate the effects on the radar output of different weather systems. Precipitation physics was used to build a physical model, but initially weather system input was included only as a conceptual mathematical vector model to provide flexibility to later implement real models of weather.

D. RELATED WORK

M. Gosset, J. Nicol, and A. Sanchez submitted an abstract for The 6th Conference on Hydrological Applications of Weather Radar describing a simulator with similar focus [1]. After personal contact with M. Gosset we learned that the simulator was unfinished and later the MATLAB code was sent to us to develop at NPS in cooperation with M. Gosset.

J.S. Marshall and Walter Hitschfeld [2] thoroughly investigated theoretical effects such as fluctuation echoes from randomly distributed scatterers providing mathematical proofs and suggestions for pulsing and scanning schemes to improve estimate accuracy. The paper defines probability distributions useful to interpret fluctuating returns and describes different methods to achieve independent data using difference in time, range, and frequency, to mention a few.

Richard J. Doviak and Dušan S. Zrnić [3] provide extensive guidance in the weather radar area presenting tools and explanations of all aspects investigated in this thesis. Most background data has originated from thoughts, theories, and descriptions provided by their fantastic book. Building a simulator calls for in depth interpretation of radar hardware solutions, which in large part can be found in their work.

Finally, John B Sandifer [4] developed scanning strategies for research and operational applications. Interesting temporal schemes were presented aiming to provide methods to rapidly scan and update large volumes using Phased Array Doppler radars. The use of stepped frequency to acquire independent samples without having to wait for the weather to reshuffle enough for de-correlation is explored and presented as one successful way of achieving a fast scanning capability.

E. THESIS OUTLINE

The thesis is organized as follows:

Chapter II, titled Weather Radar and Weather/Rain theory, introduces the basic principles of radar and the specific functions in weather applications. The physics of precipitation is summarized with emphasis on electromagnetic effects (scatter), motion behavior, and drop size distribution. The chapter also describes Radar Cross Section (RCS) of rain. The basis of the simulator is the radar signal return from the rain, which is why the electromagnetic effects will be examined to provide the foundation for the simulator design.

Chapter III covers simulator design considerations and the building of the simulator. In this chapter the scope and the limitations of the simulator are presented based upon the research questions that need to be addressed and the abilities of the simulator building tool of choice (MATLAB). The chapter further describes the building of the simulator. This chapter covers both the development and testing of the simulator. MATLAB is the computer language of choice and the simulator is initially developed in modules, to simplify development and testing. Ending the chapter the simulator parts are described.

Chapter IV provides analysis of applications. The final test or validation of the simulator where the research questions 3-7 are addressed.

Chapter V presents the results from both the development and the applications covered in the thesis.

Chapter VI contains conclusions and recommendations for future work

Appendix A Derivation of RCS for single drop

Appendix B Microwave Studio simulator setup

Appendix C Simulator setup for the Weather Radar Simulator

Appendix D MATLAB code

THIS PAGE INTENTIONALLY LEFT BLANK

II. WEATHER RADAR AND WEATHER/RAIN THEORY

Laying the foundation for the simulator, the basic theory is covered in this chapter ending with a statement of ground truth for later comparison and verification.

A. RADAR PRINCIPLES

Radar or RAdio Detecting And Ranging uses the return of transmitted electromagnetic energy from a transmitter to detect and locate objects of interest. Although the basic principles of radar have been known for decades new applications are developed constantly as a consequence of the exploration of different areas of interest. The basic function of a radar is provided by a transmitter that generates an electromagnetic signal into space by the means of an antenna. Objects in the signal path scatter or reflect part of the power projected by the transmitter back towards the receiver. By measuring the return in terms of angle of incidence and time to the return, power levels, etc. different factors can be estimated depending on the application. [5]

To be able to detect movement or to calculate average power, more than one return pulse must be measured leading to the idea of Pulse Repetition Frequency (PRF). By selecting different PRFs some quantities will be become ambiguous while others become the unambiguous.

1. Estimation of Range

Finding the range to the object of interest is merely a question of measuring the time it takes for the signal to travel from the transmitter to the target and back to the receiver, or more precisely from and to the antenna. Knowing that the electromagnetic energy travels at the speed of light, *c*, which in free space is approximately 3×10^8 m/s, the distance *R* is given by

$$R = \frac{cT}{2} \tag{2.1}$$

where *T* is the time from the antenna to the object and back to the antenna.

To ensure unambiguous measurements of range the time difference between two consecutive pulses must be at least the time it takes for the signal to travel to the range of interest and back. The maximum ambiguous range, R_u is thereby

$$R_u = \frac{cT_{rep}}{2} = \frac{c}{2f_p} \tag{2.2}$$

where T_{rep} is the pulse repetition period and f_p the PRF.

2 Estimation of Power Density

a. Antenna Gain and Beam Shapes

An important part of a radar system is the antenna. The antenna is the impedance matcher making it possible to transmit and direct energy into space with minimal losses. Some antennas provide the ability to direct the energy, making the power density higher in a limited solid angle while being lower elsewhere. This ability, called directivity, is converted to a measure of power per unit solid angle radiated in a particular direction and is compared to the same measure but without directivity. The ratio is called the directive gain of an antenna and is a function of direction and it does not include losses in the antenna.

Apart from resulting in a higher Gain, the ability to shape the beam can be helpful to enhance angular resolution, since it will limit the volume of illumination. An isotropic antenna has a power density at a range *R* of

$$P = \frac{P_t}{4\pi R^2} \tag{2.3}$$

where P_t is the transmitted power, while an antenna with gain has the power density of

$$P = \frac{P_t G}{4\pi R^2}. \tag{2.4}$$

b. Radar Range Equation

To be able to estimate the return power from an object, a parameter called Radar Cross Section (RCS) must be defined. The RCS can be defined as [6]

$$\sigma = \frac{\text{Power reflected to receiver per unit solid angle}}{\text{Incident power density}/4\pi}$$

The scattered density power at the receiving antenna is given by

$$P_{ant} = \frac{P_t G}{4\pi R^2} \cdot \frac{\sigma}{4\pi R^2} \text{ [W/m}^2\text{].} \tag{2.5}$$

Accounting for the capture area of the radar antenna, A_{eff} , results in received power

$$P_{rec} = \frac{P_t G}{4\pi R^2} \cdot \frac{\sigma}{4\pi R^2} \cdot A_{eff} \text{ [W].} \tag{2.6}$$

By using the relationship between transmit gain and receive effective area,

$$G = \frac{4\pi A_{eff}}{\lambda^2} \tag{2.7}$$

where λ is the radar wavelength, received power can be expressed as

$$P_{rec} = \frac{P_t G_t G_r \sigma \lambda^2}{(4\pi)^3 R^4} \tag{2.8}$$

ignoring all system and propagation losses. Since losses are expected let us for now add a one-way propagation loss term L rendering

$$P_{rec} = \frac{P_t G_t G_r \sigma \lambda^2}{(4\pi)^3 R^4 L^2}. \tag{2.9}$$

3. Estimation of Velocity

Pulsed radar can, by measuring the phase change between consecutive pulses, estimate radial velocity. The effect, called Doppler, is the frequency change that occurs due to the relative movement of a target with respect to the radar. Considering a two-way propagation path renders a total phase change of

$$\phi = 2\pi \cdot \frac{2R}{\lambda}. \tag{2.10}$$

In terms of angular frequency the result is

$$\varpi = \frac{d\phi}{dt} = \frac{4\pi}{\lambda} \cdot \frac{dR}{dt} = \frac{4\pi v_r}{\lambda} = 2\pi f_d \tag{2.11}$$

where v_r is the radial velocity and f_d is the doppler frequency [4].

From a pulse pair, using a coherent detector, the phase change can be extracted using

$$\begin{gathered} V_n = I_n + jQ_n = A_n e^{j\phi_n} \\ \arg\{V_n\} = \phi_n \\ V_{n+1}V_n^* = A_{n+1}A_n e^{j(\phi_{n+1}-\phi_n)} \\ \arg\{V_{n+1}V_n^*\} = \phi_{n+1} - \phi_n = \delta\phi \end{gathered} \tag{2.12}$$

leading to the single pulse pair velocity estimate

$$\hat{v}_r = \frac{\lambda\delta\phi}{4\pi\delta t} = \frac{\lambda}{4\pi T_p}\arg\{V_{n+1}V_n^*\} = \frac{\lambda f_p}{4\pi}\arg\{V_{n+1}V_n^*\} \tag{2.13}$$

where T_p is time between pulses and f_p is pulse repetition frequency.

B. WEATHER RADAR

The radar application that will be explored in this research is the ability to use the radar to estimate weather parameters such as rain reflectivity and wind speed. Most radar applications consider rain as clutter and find ways to filter out those effects. In weather applications the scatter from precipitation is used to compute the estimates mentioned earlier.

1. Sample Correlation

The return signal power level from weather will vary over time. To compute an estimate of reflectivity with small variance a number of independent samples are summed and averaged which means that the target volume must reshuffle enough to ensure de-correlation during the observation time. The power spectral density of a meteorological signal is approximately Gaussian [7] and can be written as

$$S(f) = S_0 e^{-\left\{\frac{(f-\bar{f})^2}{2\sigma_f^2}\right\}} \tag{2.14}$$

where f is the doppler frequency and σ_f^2 is the doppler spectrum variance. Taking the Fourier transform of (2.14) and normalizing results in the correlation coefficient

$$\rho(\tau) = e^{-\left(\frac{\tau^2}{2\sigma_\tau^2}\right)} \tag{2.15}$$

where σ_τ^2 is the time spectrum variance and is related to σ_f^2 by

$$\sigma_\tau = \frac{1}{2\pi\sigma_f} \tag{2.16}$$

and

$$\sigma_f = \frac{2\sigma_v}{\lambda}. \tag{2.17}$$

By using the correlation coefficient a measure of minimum difference in time can be estimated ensuring dependent or independent samples. Nathanson claims that for independent sampling $\rho(\tau) < 0.02$ is required whereas $\rho(\tau) > 0.15$ will insure dependence [7]. Doviak and Zrnić [3] set a correlation threshold for coherence at $\rho^2(\tau) \geq e^{-1}$ which corresponds to a value of $\rho(\tau) \geq 0.6$. Using this threshold renders

$$T_s \leq 0.25\frac{\lambda}{\pi\sigma_v} \tag{2.18}$$

for highly correlated samples. For independent samples using [7] $T_s > 0.70\frac{\lambda}{\pi\sigma_v}$ is required.

2. Depth of Volume

Estimating range to a volume target, compared to a point target, still involves measuring travel time. The range to every raindrop will not be determined since the return signal will be the sum of several scatterers. Measuring a volume takes pulsewidth into account as one of the parameters determining its size. A pulsewidth of τ renders a depth of the return volume of

$$d_V = \frac{c\tau}{2}. \tag{2.19}$$

3. Estimation of Power Density and Reflectivity *Z*

a. Antenna Gain and Beam Effective Solid Angle

The directivity of an antenna is a function of both azimuth and elevation angles. The radar beam has a width that has to be specified to enable an estimation of the magnitude of the volume return. Estimates of a point target involve the angle off bore sight to get the accurate gain level, while a volume target, like rain, takes into account the total gain over the volume of interest. Estimation of the gain for a volume target uses the antenna pattern function $f(\theta,\phi)$ integrated over the solid angle. Since the radar equation is a power relation the pattern function needs to be squared, also, due to the two way propagation, the power pattern function is squared again leading to

$$\int_{4\pi} \left| f(\theta,\phi) \right|^4 d\Omega. \tag{2.20}$$

Using a Gaussian Pattern Model provides a good approximation of the main beam between the 3 dB points. Extending it to both azimuth and elevation renders

$$\left| f(\theta,\phi) \right|^2 = e^{-\left(\frac{\theta^2}{\gamma^2}+\frac{\phi^2}{\delta^2}\right)}, \quad |\theta| \le \frac{1}{2}\theta_1, \ |\phi| \le \frac{1}{2}\phi_1 \tag{2.21}$$

where $\gamma^2 = \frac{\theta_1^2}{4\ln 2}$ and $\delta^2 = \frac{\phi_1^2}{4\ln 2}$, and θ_1, ϕ_1 are the half power beamwiths in the orthogonal planes. Performing the integration over the solid angle renders [8]

$$\int_{4\pi} \left| f(\theta,\phi) \right|^4 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-\left(\frac{2\theta^2}{\gamma^2}+\frac{2\phi^2}{\delta^2}\right)} d\theta d\phi = \frac{\pi\theta_1\phi_1}{8\ln 2}(1-\xi) \tag{2.22}$$

where $\xi < 0.034$ and represents the contribution from the sidelobes. It corresponds to 0.15 dB error allowing for an approximation ignoring the term $(1-\xi)$ rendering

$$\int_{4\pi} \left| f(\theta,\phi) \right|^4 = \frac{\pi\theta_1\phi_1}{8\ln 2}. \tag{2.23}$$

b. *Weather Radar Range Equation and Estimation of Reflectivity*

To define the weather radar range equation we return to the original formula

$$P_{rec} = \frac{P_t G_t G_r \sigma \lambda^2}{(4\pi)^3 R^4}. \tag{2.24}$$

In the weather radar equation RCS (σ) is replaced by RCS density $\overline{\eta}$, which is the expected radar cross section density per unit volume. Since the target is a volume target, the expected return needs to be integrated over the whole volume, which can be divided into the solid angle and depth of the return. Letting the power level at the target be $P_{\text{target}} = \frac{P_t G_t}{4\pi R^2} dA = \frac{P_t G_t}{4\pi} d\Omega$ and integrating the return power over the return volume renders

$$\overline{P}_{rec} = \int dP_{return} = \frac{P_t \lambda^2 G_t G_r}{(4\pi)^3} \int_R^{R+\frac{c\tau}{2}} \frac{dR}{R^2} \cdot \overline{\eta} \int_\Omega \left| f(\theta,\phi) \right|^4 d\Omega = \frac{P_t \lambda^2 G_t G_r c\tau\theta_1\phi_1}{1024\pi^2 R^2 \ln 2} \overline{\eta} \tag{2.25}$$

recognizing that $R \gg \frac{c\tau}{2}$.

Radar meteorologists use reflectivity factor, *Z*, instead of instead of RCS density where

$$\overline{\eta} = \frac{\pi^5}{\lambda^4} \left| K_w \right|^2 Z, \tag{2.26}$$

and

$$Z \equiv \frac{1}{\Delta V} \sum_{i \in V} D_i^6. \tag{2.27}$$

Substituting (2.26) in (2.25) and solving for *Z* renders the reflectivity estimator

$$\hat{Z} = \overline{P}_{rec} \cdot \frac{1024\pi^2 R^2 \ln 2}{P_t \lambda^2 G_t G_r c\tau\theta_1\phi_1} \cdot \frac{\lambda^4}{\pi^5 \left| K_w \right|^2} = \overline{P}_{rec} \cdot \frac{1024 R^2 \ln 2 \lambda^2}{P_t G_t G_r c\tau\theta_1\phi_1\pi^3 \left| K_w \right|^2}. \tag{2.28}$$

For an assumed continuous distribution of drops sizes with diameter density $N(D)$ (2.27) becomes

$$Z = \int_0^{D_{\max}} N(D) D^6 dD. \tag{2.29}$$

Solving the integral form for an assumed Marshall-Palmer Drop Size Distribution, DSD (Equation (2.35)), yields

$$Z = \int_0^{\infty} D^6 N(D) dD = \frac{6! N_0}{\left(4.1R^{-0.21}\right)^7}. \tag{2.30}$$

Drops do not exist in sizes ranging up to infinite diameter (see later sections for details) but are likely to be limited by a maximum diameter. Doviak and Zrnić [3] state that Equation (2.30) overestimates *Z* and suggest a truncated DSD with $D = D_{\max}$. In this case

$$Z = \frac{N_0}{\left(4.1R^{-0.21}\right)^7} \gamma(7, a) \tag{2.31}$$

where $a = \Lambda D_{\max}$ and $\gamma(7, a)$ is the incomplete Gamma function. Equations (2.30) and (2.31) provide an analytical means for computing *Z* and providing a reference as ground truth reflectivity against which simulator estimates can be compared.

4. Estimation of Velocity

Measuring phase-change over time for a volume target demands coherence between pulse pairs. Phase change will also occur due to other effects rather than radial movement. To obtain an estimate of radial velocity with small variance, a number of sample pairs need to be summed and averaged. Building on (2.13) we obtain the multiple pulse pair estimator

$$\hat{\bar{v}} = \frac{1}{N-1} \sum_{n=1}^{N-1} v_{n+1} = \frac{1}{N-1} \frac{\lambda f_p}{4\pi} \sum_{n=1}^{N-1} \arg\left\{V_{n+1} V_n^*\right\}. \tag{2.32}$$

5. Estimation of Velocity Spread

An estimate of the velocity spread can be computed using the velocity samples obtained from a pulse pair comparison (Equation (2.13)). The estimator for sample variance is given by

$$\hat{\sigma}_v^2 = \frac{\sum_{i=1}^{N}\left(\hat{v}_i - \hat{\bar{v}}\right)^2}{N} \tag{2.33}$$

and the estimator for standard deviation is

$$\hat{\sigma}_v = \sqrt{\hat{\sigma}_v^2}. \tag{2.34}$$

C. PHYSICS OF PRECIPITATION

Why and how raindrops form and what constitutes their shape and behavior is beyond the scope of this thesis, however, some basics in precipitation physics are needed to develop a physical representation of rain for the simulator.

1. Size and Shape

All drops examined are assumed to be falling at or close to their terminal velocity. Small drops, with a diameter $D < 0.35$ mm, are spherical in shape. Drops ranging from $0.35 < D < 4$ mm have progressively flattened bases, and can be approximated by spheroids [2]. Larger raindrops tend to break up after collision and drops larger than about 10 mm in diameter are unstable and break up even without collision [9]. Also drops can cluster which will affect the drop size distribution. This phenomenon will not be covered in this thesis. For further reading [10] is suggested.

2. Drop Size Distribution, DSD

RCS density is the average radar cross section density per unit volume, as mentioned previously, and for independent scattering is the sum of the RCS of all scatterers in the return volume divided by the volume. The drop size distribution represents the diameter density (mm/m^3) of drops for all diameters. The Marshall-Palmer drop size distribution [11], derived empirically, provides a general model,

$$\begin{aligned} N(D) &= N_0 \exp(-\Lambda D), \\ \Lambda &= 4.1 R^{-0.21} mm^{-1}, \\ N_0 &= 8 \times 10^3 m^{-3} mm^{-1}. \end{aligned} \tag{2.35}$$

Applying truncation as suggested in [3] with a maximum drop diameter of $D_{\max}$ yields

$$N(D) = \begin{cases} N_0 \exp(-\Lambda D), & D < D_{\max} \\ 0, & D_{\max} < D \end{cases}. \tag{2.36}$$

3. Terminal Velocities

The terminal velocity of a water drop is dependent on the size of the drop. Atlas and Ulbrich [12] derived an expression that relates terminal velocity to the diameter of the water drop as

$$v(D) = 1767 D_{cm}^{0.67} [cm/s] \approx 386.6 D_m^{0.67} [m/s] \tag{2.37}$$

which is valid for $5 \times 10^{-4} < D < 5 \times 10^{-3}$, [m]. The result has been verified against measured data [3]. Fall speed is also dependent on air pressure but that is a second order effect not considered here.

4. Velocity Spectrum Width

There are several mechanisms contributing to the spread of the velocity spectrum both from nature and the radar itself. The radar antenna motion, turbulence, differences in fall speed due to drop size and wind shears all contribute to the spread of the radial velocity of precipitation. Physically modeling all contributors demands great insight into both behavior and effects, and is beyond the scope of this thesis. However, to control de-correlation time a spread must be introduced in the simulator. This is accomplished by imparting a random Gaussian distributed velocity component to drops.

5. Rainfall Rate

Rainfall rate determines depth of accumulated water per unit time, and can be derived from water content and fall speeds. It can be shown that the a cloud's water density is

$$M = \frac{\pi}{6}\rho_w \int_0^\infty D^3 N(D)\, dD \tag{2.38}$$

where ρ_w is the water density of a drop.

Turning this into rainfall rate renders

$$R = \frac{\pi}{6}\int_0^\infty D^3 N(D) v(D)\, dD. \tag{2.39}$$

D. RADAR CROSS SECTION (RCS) OF RAIN

1. Cross Section

A coherent measure of radar return can be found by summing the electric field contribution from each point scatterer in the return volume of interest. Raindrops can be approximated as small dielectric spheres. It can be shown (see appendix A) that the backscatter field of a small sphere, when the radius is much smaller than a wavelength (λ), is given by

$$\sigma_b = \frac{\pi^5}{\lambda^4}|K_w|^2 D^6 , \tag{2.40}$$

where D is the drop diameter, $K_w = \frac{(\varepsilon_r - 1)}{(\varepsilon_r + 2)}$ and ε_r is the relative permittivity.

This approximation is valid as long as the wavelength is much larger than the drop circumference and the drop is of spherical shape and is referred to as the Rayleigh approximation [3]. As described under Physics of Precipitation, drops tend to flatten when between 280 and 1000 μm which render a polarization dependent return. This can be used to separate different sizes of precipitation when a dual-polarization radar is used to obtain the ratio of two orthogonally polarized returns [3]. This will not be included in the initial simulator work.

To verify the return predicted by the simulator, the RCS model needs to be accurate. Not only does the model need to be accurate in terms of magnitude, the fluctuation over time must also be accurate to assure the model will produce the proper variation matching the theoretical density function. For the magnitude test for accuracy, Microwave Studio has been used to predict the scatter from spheres with properties of raindrops. Microwave Studio uses the Finite Integration Technique to calculate RCS. In Microwave Studio a sphere was created with a diameter of 4 mm. Conductivity was set to 0.0001 S/m (pure water) [13]. The set up for all simulations was in most aspects the same and is described in Figure 1 and in detail in appendix B.

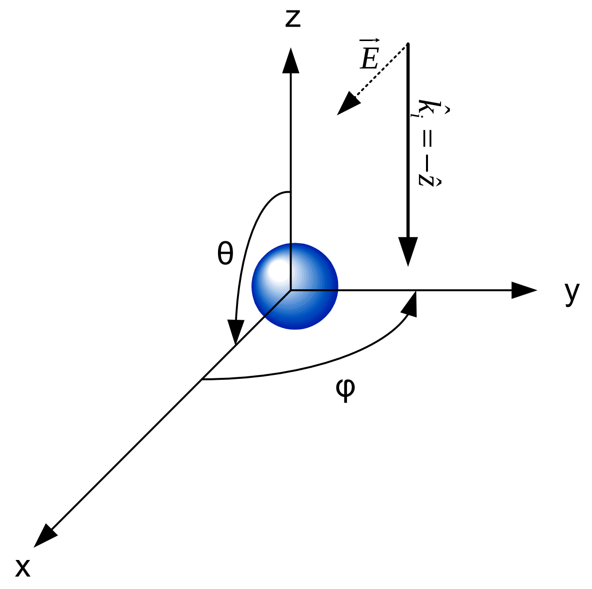


Figure 1. Simulation setup using Microwave Studio, single drop. $\hat{k}_i$ is the incident field unit vector and $\vec{E}$ is the direction of the electric field.

The calculations were made assuming an X-band radar. Although drops with $D \geq 4\,mm$ flatten out when falling, a spherical approximation was used allowing the Rayleigh approximation for the backscatter. For an X-band radar with frequency 9.4 GHz the theoretical backscatter of a single drop with $|K_m|^2 = 0.93$, and a diameter of 4 *mm* is

$$\sigma_b = \pi^5 \frac{D^6}{\lambda^4} |K_m|^2 = \pi^5 \frac{\left(4\times10^{-3}\right)^6}{c^4} \left(9.4\times10^9\right)^4 0.93 m^2$$
$$\sigma_b = 1.124\times10^{-6} m^2 \rightarrow -59.5 dBsm.$$

The result using Microwave Studio with the same input values used in the above is shown in Figure 2 and Figure 3.

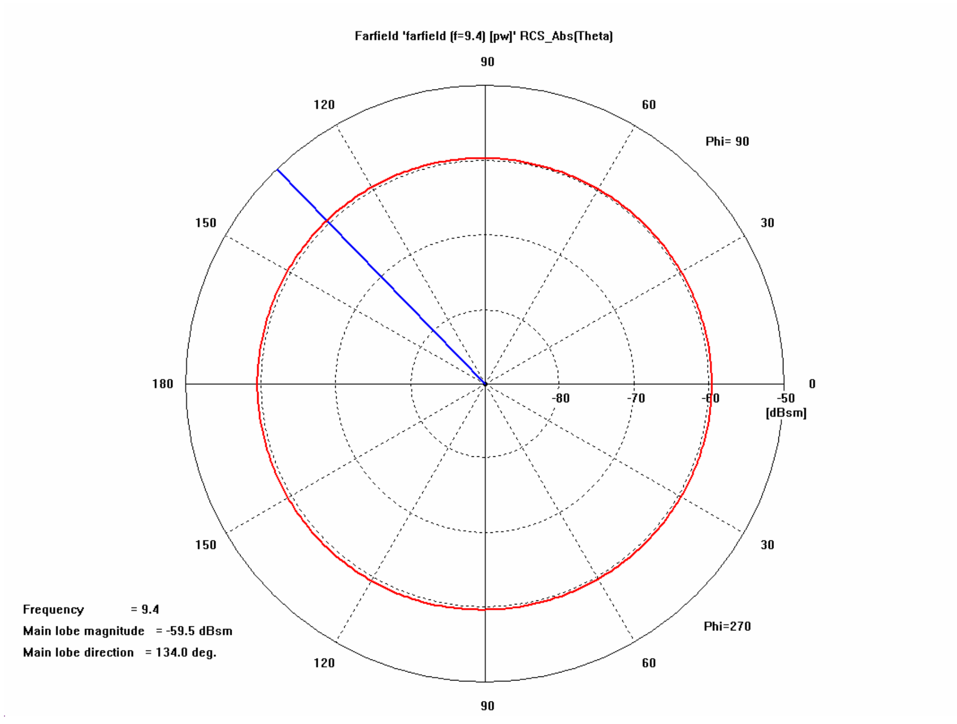


Figure 2. **Farfield Scatter from a sphere of 4 *mm* in diameter in the θ-plane for $\phi = 90^\circ$ and $\phi = 270^\circ$.**

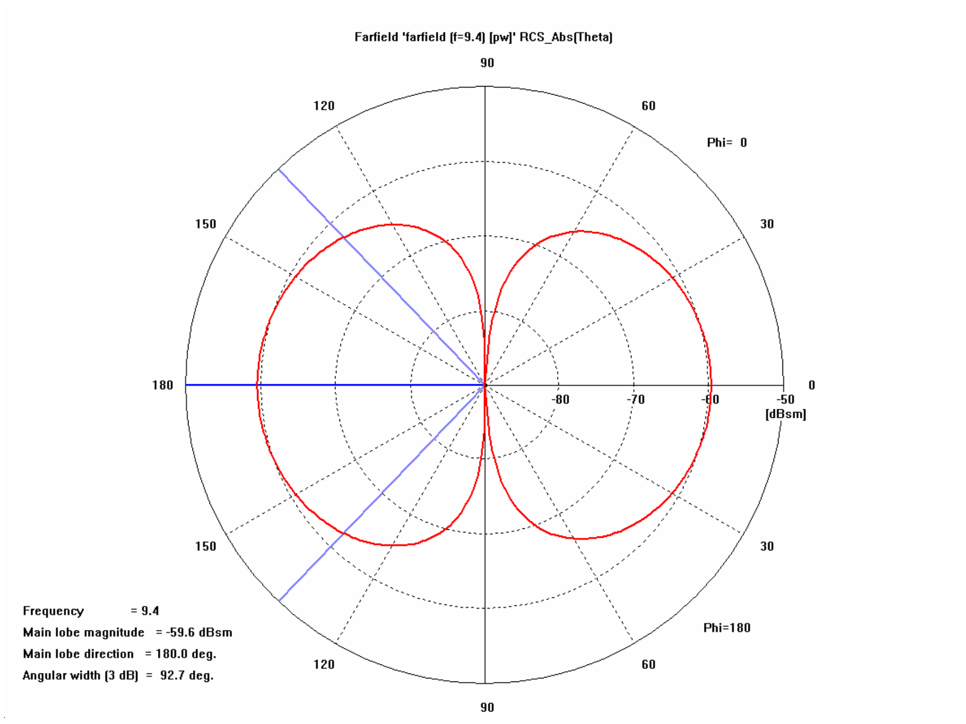


Figure 3. **Farfield Scatter from a sphere of 4 *mm* in diameter in the θ-plane for $\phi = 0$ and $\phi = 180^\circ$.**

In Figure 2 the level of the farfield scatter is equal in all directions, which is expected considering the electric field direction (x-direction) and the view of observation. The pattern is similar to a dipole where the farfield scatter has its maximum in the orthogonal direction from the electric field [15]. In Figure 3 the zero directions are inline with the polarization of the incident filed, which occurs when $\phi = 0$ and $\phi = 180^\circ$ with $\theta = 90^\circ$.

2. Array Factor

Multiple scatterers can be modeled as an array of scatterers even though they are not identical. One must realize that calculating the total coherent RCS of multiple scatterers is not a matter of summing the RCSs since RCS is acquired by summing the scattered electric field which then is squared to get the RCS. A simple approach is to calculate the electric field return from every scatterer and then add the contributions vectorally [6]. The phase change due to difference in location using a far field approximation is given by

$$e^{2jkg_{0n}}, \tag{2.41}$$

where g_{0n} represents the path difference for every drop with reference to x_{0n}, y_{0n}, z_{0n} and is given by (see Figure 4)

$$g_{0n} = x_{0n} \sin\theta \cos\phi + y_{0n} \sin\theta \sin\phi + z_{0n} \cos\theta \,. \tag{2.42}$$

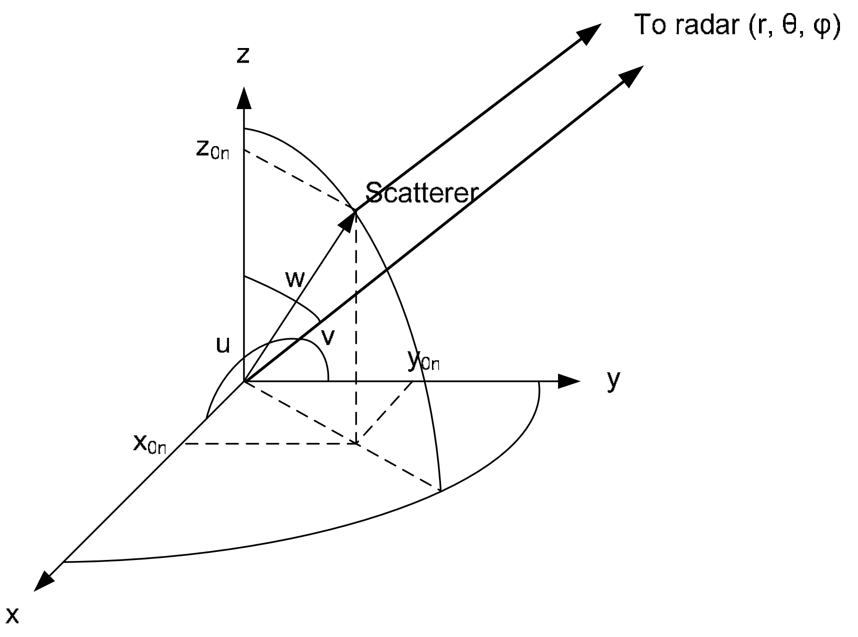


Figure 4. Far-field path difference due to shift from origin.

For equal sized scatterers the array factor which determines the total RCS is

$$AF = \sum_{n=1}^{N} e^{2jkg_{on}} \tag{2.43}$$

where $g_{on} = \vec{r} \cdot \hat{k}_i$ and $\hat{k}_i$ the incident field unit vector (Figure 5). Letting $\phi = 0$ and the distance between the drops be d gives $g_{on} = (n-1)d\sin\theta$, where n is the drop number. Recognizing the geometric series and writing the sum in closed form yields

$$AF = \sum_{n=1}^{N} e^{2jkg_{on}} = \sum_{n=1}^{N} e^{2jk(n-1)d\sin\theta} = \frac{\sin(Nkd\sin\theta)}{\sin(kd\sin\theta)}. \tag{2.44}$$

For $N = 2$, and $D = 4$ mm,

$$\sigma_{\max} = \pi^5 \frac{D^6}{\lambda^4} |K_m|^2 \left| \frac{\sin(Nkd\sin\theta)}{\sin(kd\sin\theta)} \right|^2 = 1.124 \times 10^{-6} |2|^2 \, m^2$$
$$\sigma_{\max} = 4.94 \times 10^{-6} m^2 \rightarrow -53.5 dBsm. \tag{2.45}$$

The result is independent of *d*, since $\max\left(\frac{\sin(Nkd\sin\theta)}{\sin(kd\sin\theta)} \right) = N$.

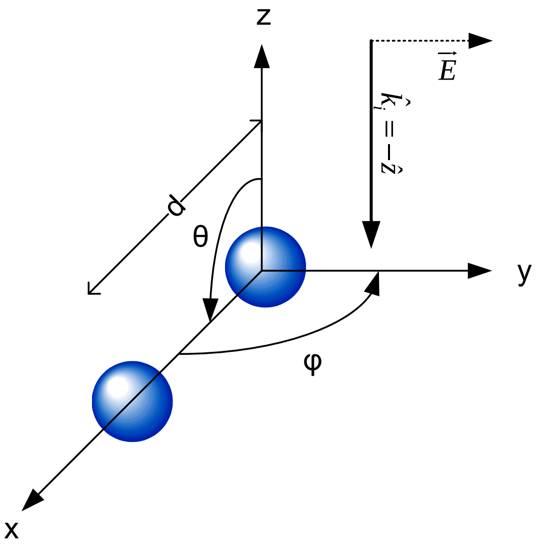


Figure 5. Simulation setup using Microwave Studio, two drops. $\hat{k}_i$ is the incident field unit vector and $\vec{E}$ is the direction of the electric field.

The value is calculated only for a direct reflection not taking into account the $\cos^2\theta$ dependence due to angle of incidence. The two drop setup using $D = 4\ mm$ and $d = \lambda$ rendered results presented in Figure 6 and Figure 7. In Figure 6 the farfield scatter pattern show a maximum in the orthogonal direction from the electric field, as expected since the electric field was along the y-axis ($\phi = 90^\circ$ and $\phi = 270^\circ$ with $\theta = 90^\circ$). In Figure 7 zero directions show for $\phi = 0$ and $\phi = 180^\circ$ with $\theta = 30^\circ$ and $\theta = 150^\circ$, which is expected since

$$\left|\frac{\sin(Nkd\sin\theta)}{\sin(kd\sin\theta)}\right| = \left|\frac{\sin\left(2\cdot\frac{2\pi}{\lambda}\lambda\sin\theta\right)}{\sin\left(\frac{2\pi}{\lambda}\lambda\sin\theta\right)}\right| = \left|\frac{\sin(4\pi\sin\theta)}{\sin(2\pi\sin\theta)}\right|. \qquad (2.46)$$

So for $\theta = 30^\circ$ and $\theta = 150^\circ$ the expression yields zero. Comparing the calculations, Equation (2.45), with the results from the Microwave Studio simulations reveals a difference of only 0.1 dB, which can be the effects of mutual coupling. The max values are called Bragg lobes and their locations are dependent on the ratio of d/λ, their location with respect to each other, and the measuring point.

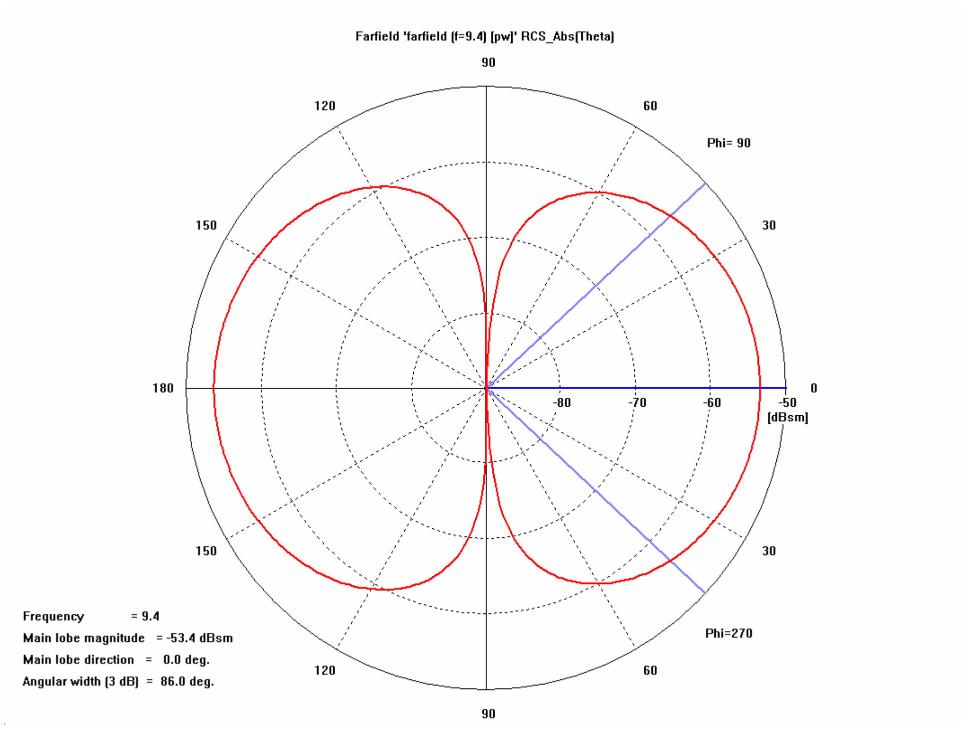


Figure 6. Farfield Scatter from two spheres of 4 *mm* in diameter in the θ-plane for $\phi = 90^{\circ}$ and $\phi = 270^{\circ}$.

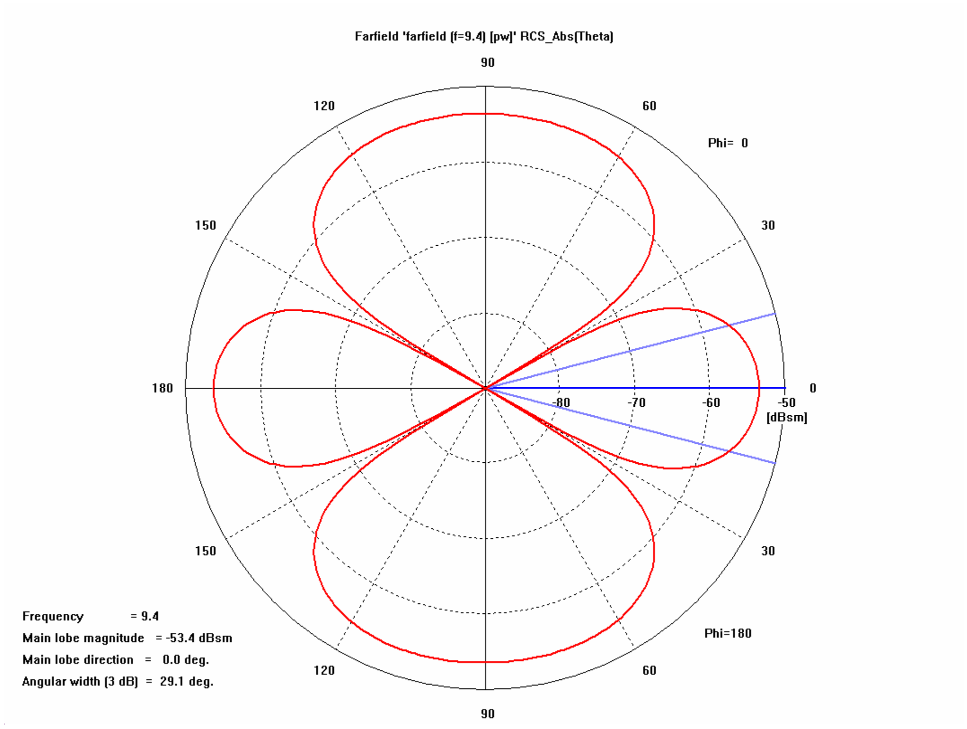


Figure 7. Farfield Scatter from two spheres of 4 *mm* in diameter in the θ-plane for $\phi = 0$ and $\phi = 180^{\circ}$.

The question of mutual coupling was addressed using Microwave Studio. The scattering pattern from the illumination of first two and then three spheres was examined, this time for different separation distances (Figure 8). The values plotted are the results using the same drop setup as in Figure 5, i.e. $D = 4\ mm$, $\theta = 0$ and $\phi = 0$. The measures indicate coupling effects. However, the software is not optimized to calculate low conductivity targets, as raindrops are. When the space between the drops increases the program adds noise as the numerical grid becomes significant (see Appendix B). Although mutual coupling is likely to appear, available tools could not provide guidance how to provide realistic implementation.

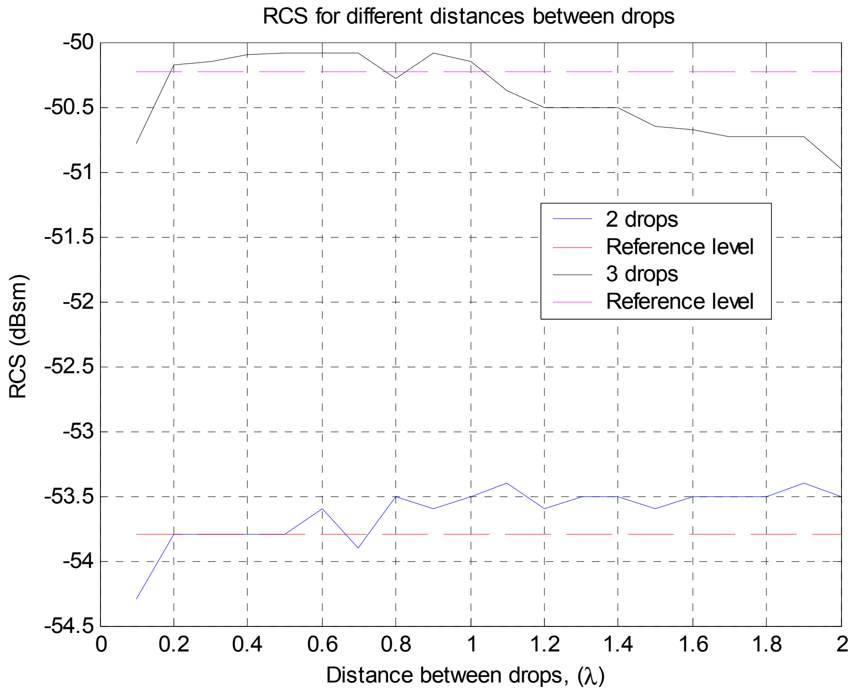


Figure 8. Monostatic Back Scatter of two and three drops of 4 mm diameter when spacing is varied from 0.1 λ to 2 λ.

3. Averaging over Time

The weather radar return signal is composed of the scattered electric field from all drops in the resolution cell (see Figure 9). The total scattered field of the drops in the resolution cell is given by

$$\vec{E}_s(O) = \sum_{n=1}^{N} \vec{E}_n(O) \tag{2.47}$$

where N is the number of drops and O is the observation point.

Rearranging, $\sigma = 4\pi r^2 \left|\frac{\vec{E}_s}{\vec{E}_i}\right|^2$ renders $\left|\frac{\vec{E}_s}{\vec{E}_i}\right| = \frac{\sqrt{\sigma}}{\sqrt{4\pi r^2}}$, so summing the electric field from all scatterers yields

$$E(r,\theta,\phi)\cdot\vec{a} = \sum_{n=1}^{N} \sqrt{\sigma_n} e^{j\psi_n} \vec{a} \tag{2.48}$$

where ψ is given by $\psi_n = 2kg_{0n}$ and is a function of r, θ and ϕ (Equation (2.43)) [6] and $\vec{a}$ is a unit vector giving the direction of the scattered field.

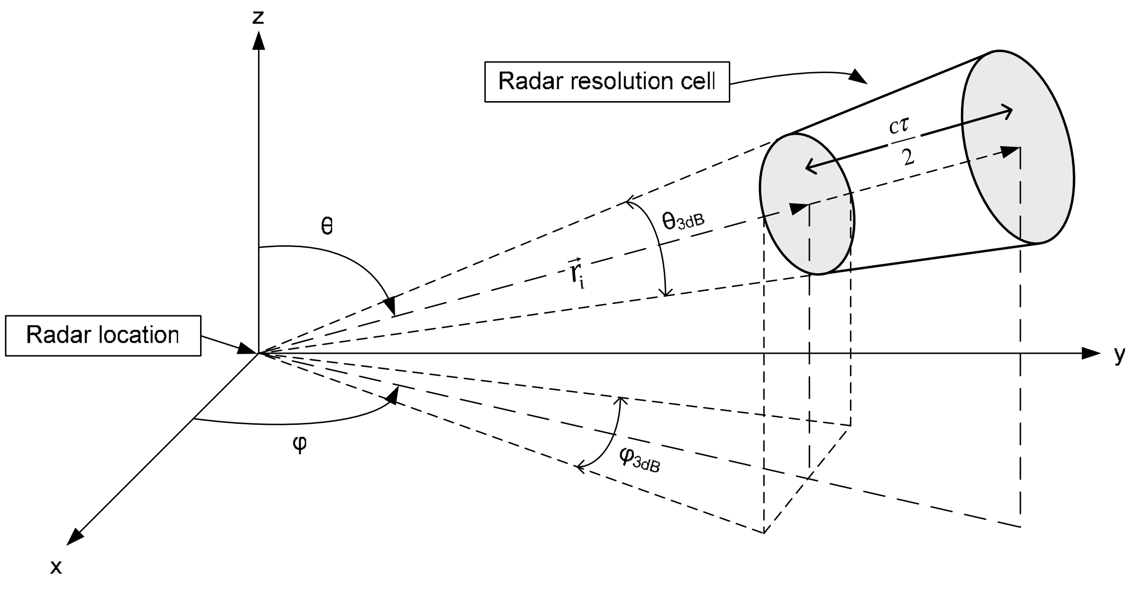


Figure 9. Radar resolution cell used in Simulator.[1]

It can be argued that ψ is a random variable uniformly distributed on the interval $[0, 2\pi]$ thus, expected RCS will be

[1] Azimuth and Elevation angles are reversed compared to the mathematical conventions.

$$\langle \sigma_{total} \rangle = \sum_{n=1}^{N} \sigma_n \tag{2.49}$$

where $\langle \ \rangle$ denotes the expected value. So to get the average power estimate a number of independent samples need to be gathered and averaged.

4. Absorption and Attenuation

Apart from the fraction of the incident electric field that rain scatters, rain also absorbs energy which causes attenuation. This has greater effect for higher frequencies, which imposes limits on the possibilities of getting higher angular resolution by using short-wavelength radars. The effects of attenuation can be used in bi-static setups estimating the weather parameters based on the level of attenuation. Bi-static setups and effects of attenuation will not be taken into account in the initial simulator work.

E. MATHEMATICAL MODELS

This section describes the basic formulas that will be used to model the parameters required in the simulator to generate the return signal. The models for each parameter are defined in Table 1. Although choosing Equation (2.30) as ground truth will probably create too high estimates of reflectivity [3], this will be the initial reference as the simulator development starts.

Table 1. Mathematical models for Weather Estimators.

Parameter	***Input or estimator***	***Ground Truth***
Drop Size Distribution, DSD	$N(D) = N_0 \exp(-\Lambda D)$, $\Lambda = 4.1R^{-0.21} mm^{-1}$, $N_0 = 8\times10^3 m^{-3} mm^{-1}$	
Reflectivity Factor, Z	$\hat{Z} = \hat{\bar{P}}_{rec} \cdot \dfrac{1024 R^2 \ln 2 \lambda^2}{P_t G_t G_r c\tau\theta_1\phi_1\pi^3 \lvert K_w \rvert^2}$	$Z = \int_0^\infty D^6 N(D)\, dD = \dfrac{6!N_0}{\left(4.1R^{-0.21}\right)^7}$
Average Radial Velocity	$\hat{\bar{v}} = \dfrac{1}{N-1}\dfrac{\lambda f_p}{4\pi}\sum_{n=1}^{N-1} \arg\{V_{n+1}V_n^*\}$	*Compare with input value*
Doppler Spectrum Width	$\hat{\sigma}_v^2 = \dfrac{\sum_{i=1}^{N}\left(\hat{v}_i - \hat{\bar{v}}\right)^2}{N}$, $\hat{\sigma}_v = \sqrt{\hat{\sigma}^2}$.	*Compare with input value*

THIS PAGE INTENTIONALLY LEFT BLANK

III. SIMULATOR DESIGN CONSIDERATIONS AND DEVELOPMENT OF SIMULATOR

Having described the required parameter models in the previous chapter the design and development of the simulator will commence. As stated, the simulator should allow for inputs of rain rate, standard radar values (e.g. frequency, azimuth and elevation angles, and pulsewidth), wind speeds and angles, and also velocity spread. Output signal parameter estimators of interest are average power, average Doppler velocity and Doppler spread. The output should preferably be in form of I and Q return voltage making it possible to post process the data to obtain the estimate of average power, Doppler velocity and Doppler spread.

A. SCOPE

To provide a tool to answer the research questions specified in the introduction chapter, the following is required:

1. A physical model of a spatial region of raindrops must be created and later tested and verified against real life measures or existing models,
2. A model for realistic motion of raindrops that are weather system dependent must be created where at least an interface between weather models and the simulator must be specified,
3. A method of implementing spatial and temporal sampling schemes to study the output for constant or stepped frequency sampling,
4. A method to study the use of pulse compression and range averaging as a means of rapidly obtaining independent samples,
5. Displays of pedagogical interest to facilitate the presentation and testing of research issues mentioned,
6. A study of the performance of estimators for the first three Doppler moments,
7. A study of the utility of estimators for higher order Doppler moments.

B. LIMITATIONS

The initial design will not include:

- Dual polarization
- Absorption/Attenuation
- Losses and noise
- Mutual coupling and multiple reflections

C. DEVELOPMENT

Having a prototype from M. Gosset [1] provided valuable ideas and experience developing the simulator. The method used to develop the final version is described in Figure 10.

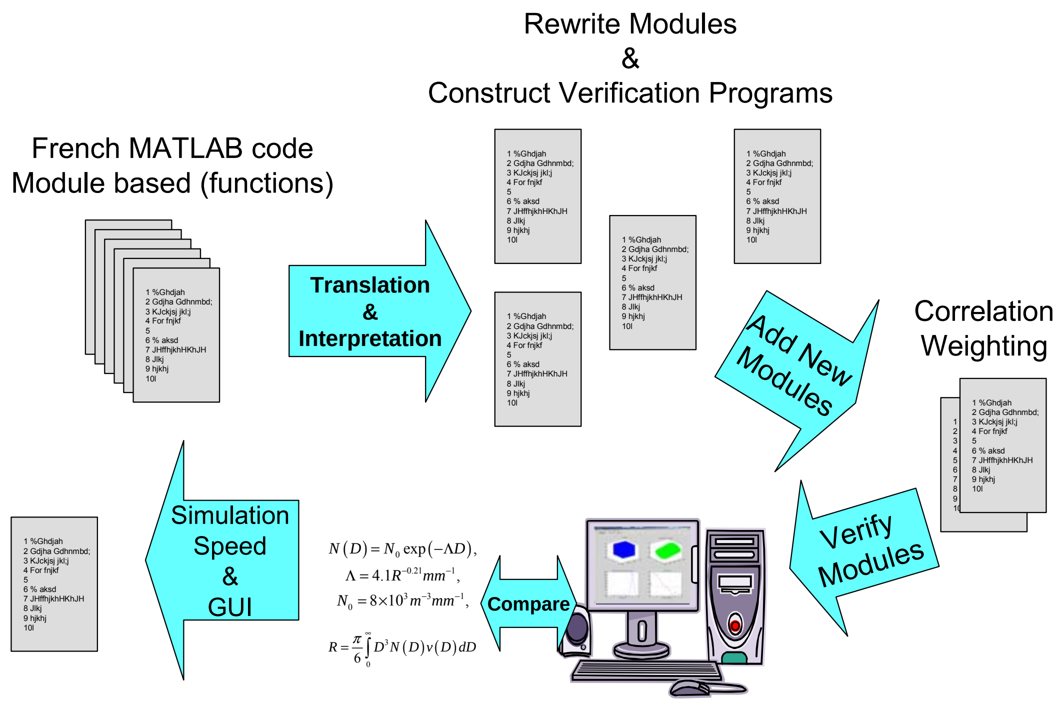

Rewrite Modules & Construct Verification Programs
French MATLAB code Module based (functions)
Translation & Interpretation
Add New Modules
Correlation Weighting
Verify Modules
Compare
Simulation Speed & GUI
$N(D) = N_0 \exp(-\Lambda D)$, $\Lambda = 4.1 R^{-0.21} mm^{-1}$, $N_0 = 8 \times 10^3 m^{-3} mm^{-1}$, $R = \frac{\pi}{6}\int_0^\infty D^3 N(D) v(D) dD$

Figure 10. Plan for development of Simulator using the French version.

1. Translation and Interpretation

The French code consists of a main program that calls for functions that performs different tasks to support the main program. The functions are[2]:

- Size
- Approximation
- Marshall-Palmer
- Resolution Volume
- Number of drops
- Initial Position
- Summation
- Movement
- Atlas Ulbrich

The input parameters are shown in Table 2.

Table 2. Simulator input parameters

Radar Data	***Weather Parameters***
Transmit Power	Drop size diameter limits
Max Antenna Gain	Drop Diameter resolution
Range to resolution cell	Rain rate
One-way 3 dB Beamwidth	Rain angles
Pulsewidth	Wind, speed and direction
Frequency	
Pulse Repetition Frequency (PRF)	
Elevation and Azimuth	
Number of pulses for integration	

[2] The code was translated by Professor Monique Fargues of NPS.

a. Function Description

The main program starts by calling the Size-function that generates a discrete Diameter vector based on the diameter interval and resolution of interest. The function calls the Approximation-function that calculates the RCS for each diameter using Equation (2.40). The Marshall Palmer-function is called to calculate the number of drops for each diameter using Equation (2.35). The resolution is multiplied with the number of drops per diameter and stored in a matrix together with the Diameters and RCSs. The Size-function also has a part that uses a pre-set threshold to take away the smaller drops that account for 10 % of the total RCS. The matrix created in the Size-function consists of Diameter, RCS, Number of drops, total RCS per diameter, and Reflectivity Factor structured as

$$T = \begin{pmatrix} D_1 & \cdots & D_N \\ \sigma_1 & \cdots & \sigma_N \\ N(D_1) & \cdots & N(D_N) \\ \sigma_1 \cdot N(D_1) & \cdots & \sigma_N \cdot N(D_N) \\ Z_1 & \cdots & Z_N \end{pmatrix} \tag{3.1}$$

where the first values are set by the threshold and *N* by the upper Diameter limit.

The main program then calls the Resolution Volume-function to generate the resolution volume using the radar data. The function adds a margin to the resolution volume so that the drops can pass through the resolution volume as time passes between pulses. Both the borders for the increased resolution volume, henceforth called the Box, and the resolution volume are defined for later use. The main program uses the data to feed the Number of drops-function that calculates the number of drops considering the volume of the Box and changing the values in the Matrix.

Using the Initial Position-function, the simulator uniformly distributes every drop inside the Box using the data from the Matrix. The positions are stored in files unique for each diameter using spherical coordinates to simplify comparison when determining whether a drop is inside the Resolution Volume or not.

The main program then starts the simulation for the number of pulses specified. For all iterations the Summation-function calculates the return voltage and the return power taking into account the position and RCS of each drop. The radar cross section is calculated summing the scattered electric field from each rain drop using

$$E_{total}\left(t_k\right)=\sum_s\sum_{i=1}^{N_i}\sqrt{\sigma_{\mathrm{D_s}}}\cdot e^{j2k\bar{r}_i\left(t_k\right)} \tag{3.2}$$

where t_k is the sample time and s is the Diameter vector.

The results are stored in different files depending on what data it is. Before next iteration the main program calls the Movement-Function to move the drops according to wind and fall velocities. To get the estimated terminal velocities for each diameter the Atlas Ulbrich-function [12] is used, which is based on Equation (2.37).

The output of the simulator consists of four plots that show the Box with all initial drops plotted, the Resolution Volume with all positions the drops have during the simulation plotted, power return levels for every pulse, and the phase of every drop and every pulse.

b. Verification

To evaluate the accuracy of the simulator, verification programs were created to estimate errors of the modules of the simulator. The modules to examine were

- Number of drops
- Average power estimation
- Doppler estimation
- Doppler spread estimation

(1) Number of drops. The original French program used a diameter interval between 0.05 mm and 5 mm with a resolution varying from 0.08 mm for the smaller drops (up to 1 mm) to 0.2 mm. Using the original rain rate set to 50 mm renders a histogram as seen in Figure 11. The threshold, that removes the smaller drops that account for 10 % of the total RCS, is also indicated in the same figure.

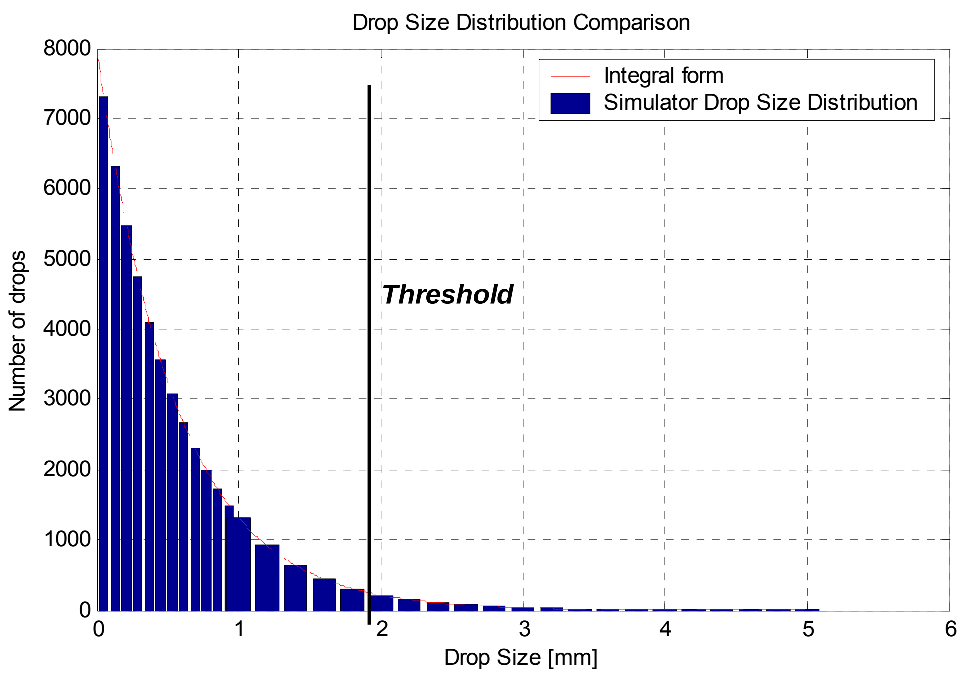


Figure 11. Drop Size Distribution comparison for French simulator. The volume is set to 1 m^3 and the rain rate to 50 mm.

Eliminating the smallest drops will effect the fluctuation of the return signal as well as the peak values, which in turn affects the average value. This is an effect of the array factor previously described that can make several small drops generate high return values when in phase. Also the choice of upper limits will affect the level of power return especially for higher rain rates, which will be examined in a later section. These values are plotted without making the number of drops to integer values. The simulator uses CEIL to round off the number of drops making the integer equal to or higher than the original value. Simulations show that the effect becomes significant for rain rates below 40 mm/hr, creating too high return power values (Figure 12, RCS using original French simulator). To produce values closer to what the integral form generates using Equation (2.30), the span of diameters needs to be widened and the round off tool exchanged.

(2) Average Power Estimation. To verify the average power return, a simplified simulator was built that didn't include the movement of the drops. The movement was simulated by randomly re-plotting the drops for all iterations. Using the knowledge about the effects of choosing limits for the drops size, a simulation was

performed verifying the levels for average return power. Also the statistical distribution of the return signal was examined to assure it agreed with basic theory [1], [2]. The simulator was run for 1000 samples and for rain rates between 10 and 120 mm/h and a wavelength $\lambda = 0.1$ m.

The results (Figure 12) show that using the original French simulator makes the relation between RCS and rain rate almost linear while the new generator follows the integral form (Equation (2.30)). The new generator uses a wider span of diameters (up to 15 mm), does not apply a threshold and uses ROUND instead of CEIL making the value closer to the real value.

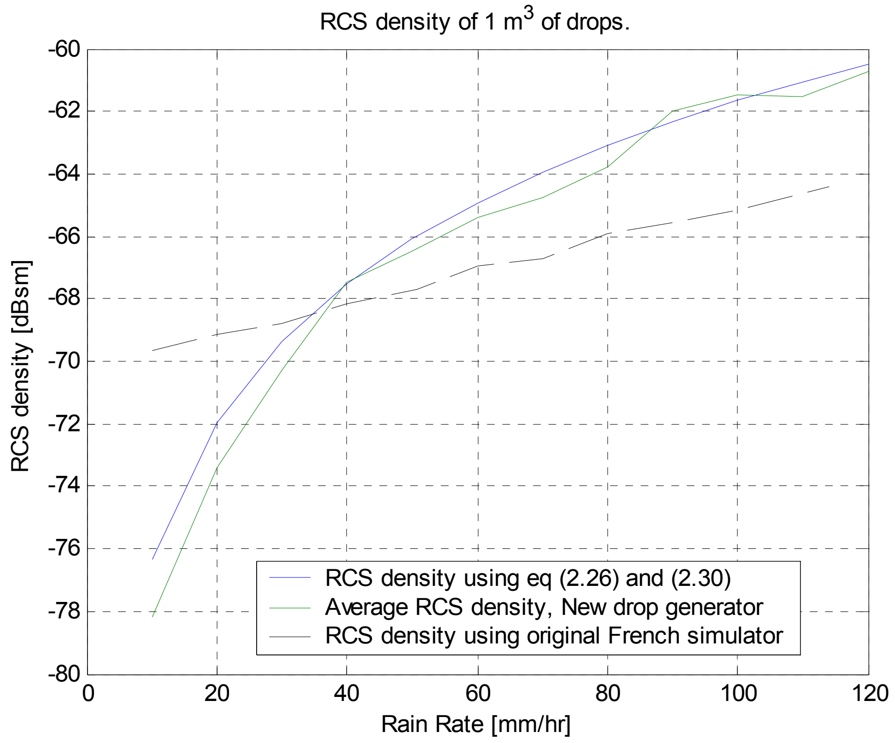


Figure 12. Average RCS density comparison using different drop generators. 1000 samples were used and compared with the result using the integral form of reflectivity to calculate RCS density. $\lambda = 0.1$ m.

The new generator still produces values with errors up to 2 dB even with sample sizes as large as 1000 samples. To examine the cause of this error another version of a drop generator was constructed taking into account the error caused by the rounding off. The generator keeps track of the error and adds it at the end of the simulation. This also provided an idea of how to be able to improve the simulator speed by reducing the number of drops while still retrieving accurate RCS and statistical behavior. Reducing the number of drops by dividing by a constant and after the simulations multiply it back into the calculations to retrieve reflectivity, number of drops, and rain rate is valid since all these operations are performed by integrating over the diameter. Treating the error like a constant results in a max error of about 0.4 dB, an error that seems to increase as the rain rate increases (see Figure 13). Since the simulator replaces the integration by a summation, errors will arise. Using Equation (2.30) renders a *Z*-value of 54.1 dBZ, for a rain rate of 100 mm/h. When using the same form of summation the simulator applies, the same estimation is 54.5 dBZ, which explains the error of 0.4 dB.

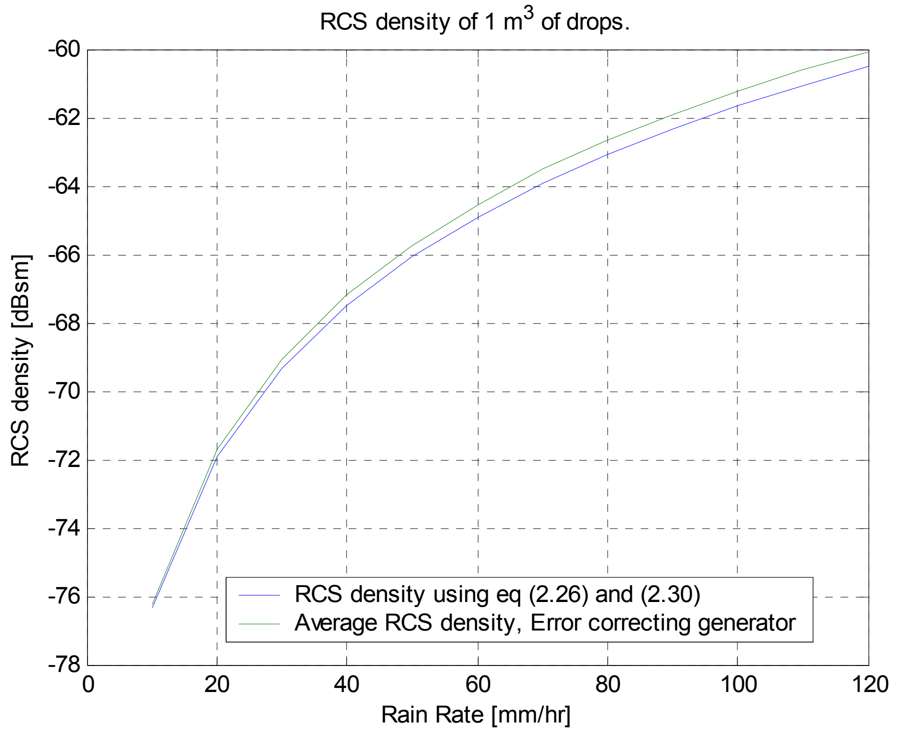


Figure 13. RCS density comparison using new drop generator with error correction and reduction factor. $\lambda = 0.1$ m.

To get the average value of the return power, several independent samples are used. The RCS of a raindrop was derived using the radiation integral, which can be used when the return complex voltage is calculated. Since I and Q are random variables, independent and none of them dominant, the central limit theorem states that their sum tends to be Gaussian, as long as the numbers of samples are large enough. The amplitude of the return voltage i.e. the absolute value of the I and Q voltage return,

$$|V| = \left(I^2 + Q^2\right)^{1/2} \tag{3.3}$$

is therefore expected to have a Rayleigh distribution [3]. Running the drop generator for 10,000 samples renders results shown in Figure 14 and Figure 15.

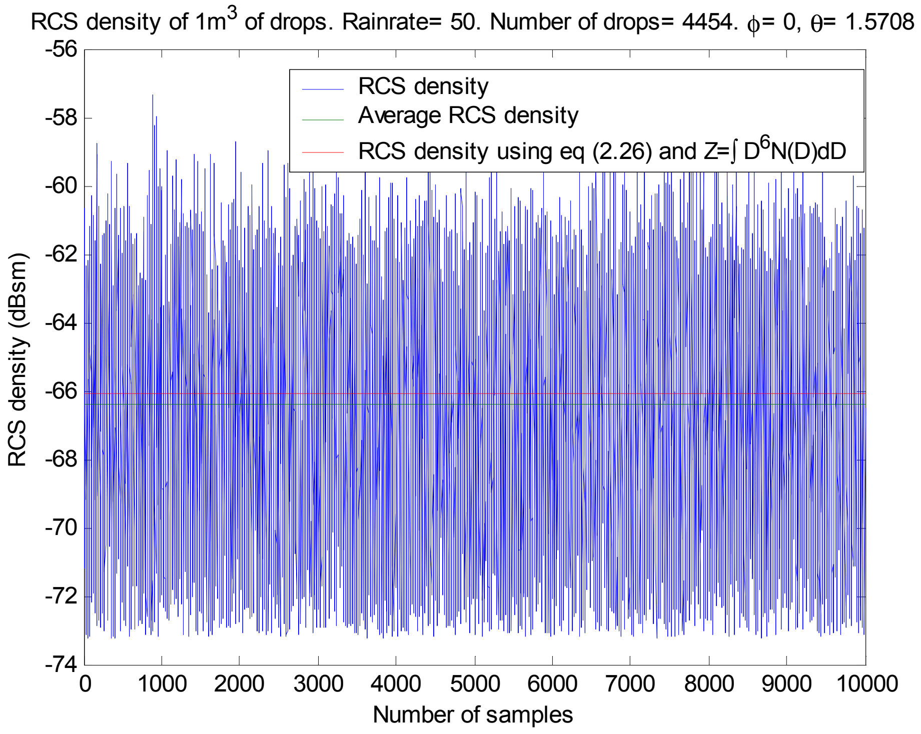


Figure 14. Plot of 10,000-sample-run of drop generator.

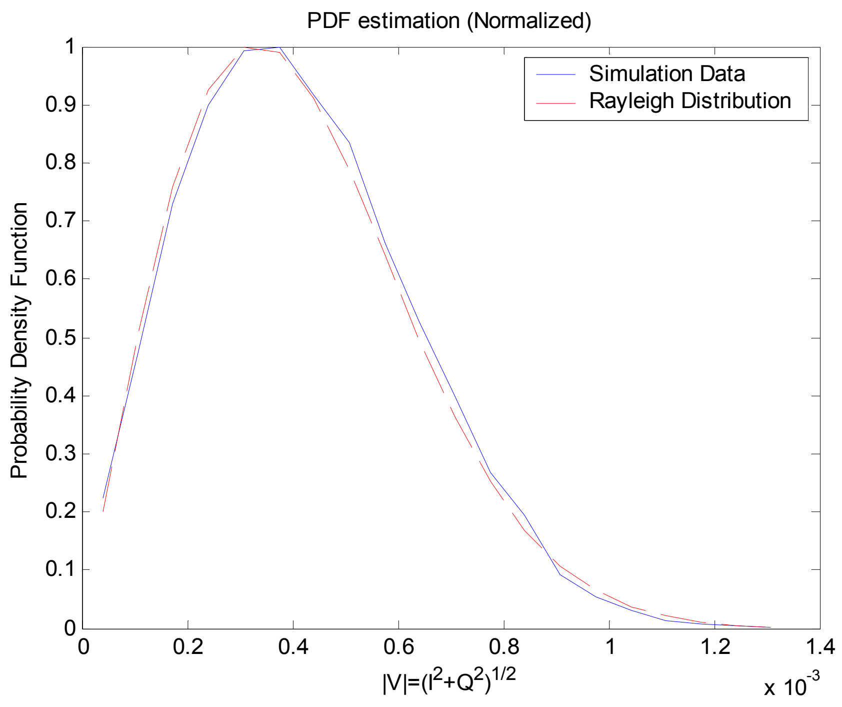


Figure 15. Rayleigh fit of data from 10,000-sample run of drop generator.

In Figure 14 the average value of RCS is lower than the result using the integral form. This was explained earlier when the results from summing over all diameters was compared with the integration from zero to infinity. The Rayleigh fit in Figure 15 follows the expected distribution. The results were retrieved by plotting a histogram of the 10,000-sample-run. Both the histogram and the fit were normalized to allow for comparison.

(3) Doppler estimation. The module that moves the drops takes into account the terminal velocities, determined by Equation (2.37) with an associated fall angle, and wind speeds in x, y, and z-dimension. The spread of terminal velocities together with the fall angle will contribute to the spread of the Doppler spectrum. To evaluate the mean Doppler frequency a pulse pair algorithm is used to record the phase change over time using Equation (2.32) and comparing the phase extracted from two consecutive returns pulses. Applying this on the drop generator and the movement module enables an estimate of the mean frequency. A simulation was run for 30 pulses (29 pulse pairs) with a PRF of 2000 Hz. Wind speed was set to zero and the rain fall angle to 30° with respect to the z-axis (for detailed specifications see appendix C). The

simulation showed an average Doppler frequency $\hat{\bar{f}}_d = -57.3$ Hz or $\hat{\bar{v}}_r = -2.85$ m/s in x-direction (radial direction) and standard deviation of $\hat{\sigma}_f = 26.8$ Hz, which represents $\hat{\sigma}_v = 1.34$ m/s (see Figure 16). The spread was induced by the differences in terminal velocities for different drops sizes. To estimate the accuracy, calculations were made using a 2 mm drop for which

$$v(D) \approx 386.6 D_m^{0.67} = 386.6 \cdot 0.002^{0.67} = 7.89[m/s].$$

Taking the fall angle into account renders

$$v_{x-dir} = v \cdot \sin\left(30^\circ\right)\cos\left(0\right) = -3.01\left[m/s\right].$$

To further verify that the simulator generates accurate results, the fall angle was removed and only wind in the x-direction was used. The same inputs were used adding wind of -5 m/s rendering $\hat{\bar{f}}_d = -105.1$ Hz, which corresponds to $\hat{v}_r = -5.3$ m/s as seen in Figure 17. The Doppler spread was $\hat{\sigma}_f = 7.9$ Hz ($\hat{\sigma}_v = 0.4$ m/s). Note that the first spectrum (Figure 16) is skewed towards zero indicating that the smaller drops, constituting the majority of the drops, bias the result. Also the limited number of samples (29 pulse pairs) can have effects on the shape of the histogram.

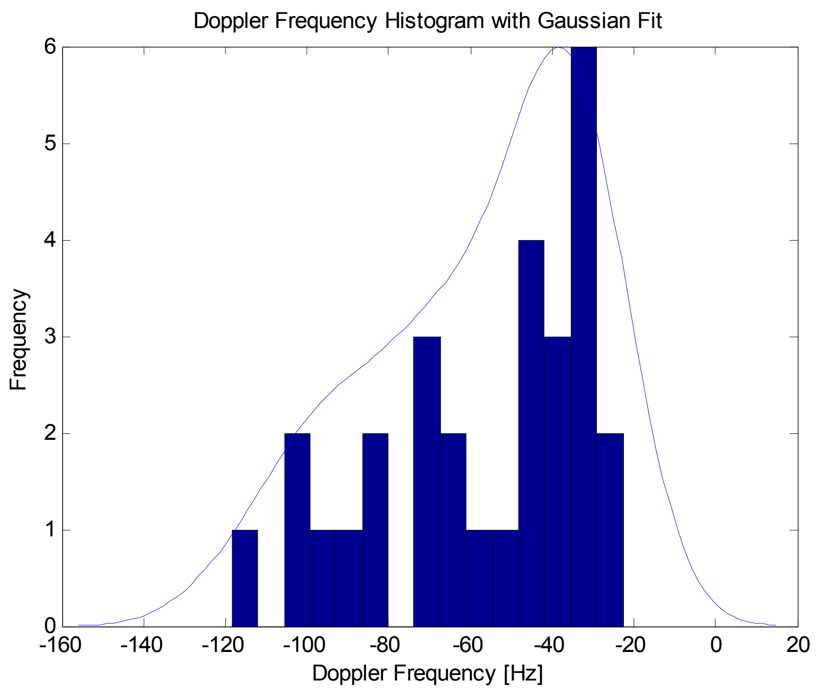


Figure 16. Doppler Frequency Histogram applying terminal velocities and 30° fall angle. 30 pulses and 29 pulse pairs.

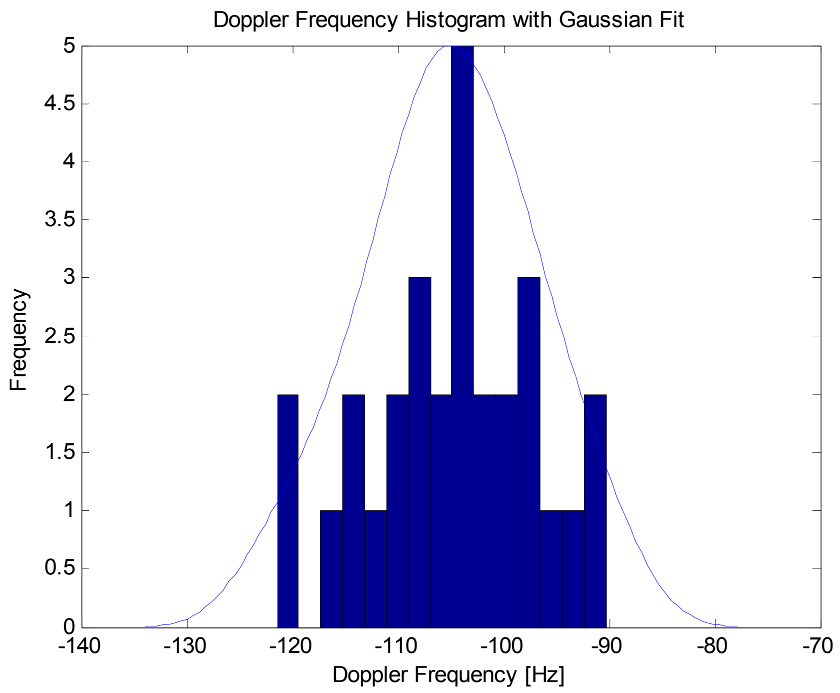


Figure 17. Doppler Frequency Histogram applying -5 m/s wind in –x-direction. 30 pulses and 29 pulse pairs.

(4) Doppler spread. The simulation run above showed a standard deviation of 1.18 m/s, which is a measure of how much the average radial velocity of all scatterers of each pulse return spread over all samples. The data was fitted to a normal density function, which is expected for a power spectrum [3], to explore if the data could be considered Gaussian. The second run fits better with the Gaussian plot, which can be explained by the randomness of the process different from the first simulation where the spread is distorted by the differences in drop sizes and therefore terminal velocities. To further examine the power spectrum new algorithms were implemented as described in the upcoming section.

2. Adding New Modules

The verification programs have been used to validate the simulator but there are still improvements that can be made to achieve better accuracy and realism. First to recapitulate what has been accomplished up until this point:

- A drop generator that provides the number of drops given by the distribution of interest based on rain rate.
- An algorithm that calculates the complex scattered voltage return based on the position of each drop within the radar resolution volume.
- A module that introduces movement to the drops concerning fall speed with an associated angle, and wind speed.
- An algorithm that calculates the doppler frequency of the moving drops based on phase change between pulse pairs.
- An error correcting algorithm that also can be used to reduce the number of drops used in a simulation to make it faster.

a. Range Weighting

To provide a more realistic level of the return signal value, a range weighting function, based on the radar range equation, was built and implemented. As mentioned earlier, effects of absorption and other losses will not be considered, leaving only the effects of difference in range. To account for range effects, Equations (2.21), (2.24), and (2.48) were used rendering

$$\vec{E}_{weigthed} = \sqrt{\frac{P_t G_t(\theta,\phi) G_r(\theta,\phi)\lambda^2}{(4\pi)^3 (r(\theta,\phi))^4}} \cdot \vec{E}(r,\theta,\phi)$$

$$\vec{E}_{weigthed} = \sqrt{\frac{P_t}{(4\pi)^3}} \cdot G\lambda \cdot |f(\theta,\phi)|^2 \cdot \frac{\vec{E}(r,\theta,\phi)}{r^2} \tag{3.4}$$

$$E_{weigthed} \cdot \vec{a} = \sqrt{\frac{P_t}{(4\pi)^3}} \cdot G\lambda \cdot \frac{\sqrt{\sigma_d} e^{j\psi} \cdot e^{\left(-\frac{\theta^2}{\gamma^2} - \frac{\phi^2}{\delta^2}\right)}}{r^2} \cdot \vec{a}$$

where $\vec{a}$ is a unit vector giving the direction of the scattered field. For the total Resolution Cell

$$\sum E_{weighted} \cdot \vec{a} = \sqrt{\frac{P_t}{(4\pi)^3}} \cdot G\lambda \cdot \sum_{n=1}^{N} \left(\frac{\sqrt{\sigma_n} e^{j\psi_n} \cdot e^{\left(-\frac{\theta_n^2}{\gamma^2} - \frac{\phi_n^2}{\delta^2}\right)}}{r_n^2} \right) \cdot \vec{a} \tag{3.5}$$

where $\psi_n = 2kg_{0n}$ and g_{0n} is given by Equation (2.42).

b. Correlation

Another estimate of the doppler from frequency the return electric field can be obtained by taking the discrete Fourier transform (here FFT) of the signal samples using the complex voltage return. The Doppler Power Spectrum is given by

$$\hat{S}(f) = |Z(f)|^2 T_s / M$$
$$Z(f) = \sum_{m=0}^{M-1} V(mT_s) e^{-j2\pi f T_s m} \tag{3.6}$$

where M is the number of samples. An estimate can be retrieved by taking the Fourier transform of the autocorrelation function

$$\hat{R}(l) = \frac{1}{M} \sum_{m=0}^{M-|l|-1} V^*(m) V(m+l). \tag{3.7}$$

Computing the autocorrelation function and the FFT for the output in the example on pg. 39 produces the same average frequency estimate, $\hat{\bar{f}} = -105.1$ Hz as before (see Figure 18).

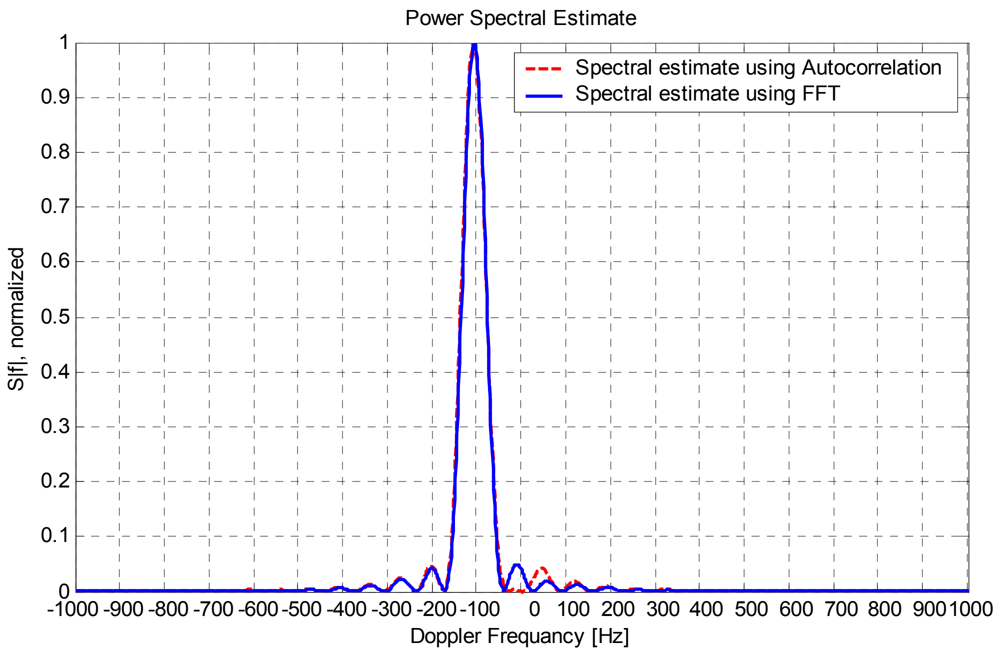


Figure 18. Doppler spectrum using FFT and the Fourier transform of the autocorrelation function. (They overlap each other). Peak gives average frequency estimate $\hat{\bar{f}} = -105.1$ Hz.

Both autocorrelation and FFT preserve amplitude information, which has consequences interpreting average Doppler using these functions. Larger drops will affect the power spectrum to a greater extent than smaller since drop RCS is proportional to D^6. Thus, any difference in the radial velocity of large and small drops will result in the power spectrum being skewed towards the Doppler associated with the larger drops. Using the pulse pair algorithm (Equation (2.13)) removes the amplitude information producing a more “true” average doppler frequency or radial velocity.

c. Frequency Stepping

To accomplish rapid volumetric scanning there are several methods proposed. One method to achieve de-correlation is by shifting in frequency. A fast scanning scheme to speed up the acquisition of independent samples was proposed by Doviak and Zrnić [3]. They suggested a method where a pulse pair is sent with one frequency assuring dependent doppler measures, then the frequency is changed a minimum of $1/\tau$, for de-correlation to send the next pulse pair. Implementing this scheme into the simulator requires modules stepping between more than one PRF also taking into account the differences in wavelengths when computing the meteorological signal parameter estimates. In Figure 19, a three frequency stepping scheme is introduced that transmits a pulse pair at frequency 1, and listens for the time set by the Doppler Pulse Repetition Time, PRT_1, before changing to frequency 2. Before the second frequency can be transmitted the frequency oscillator needs a settling time, here called delay. After all three frequencies have been transmitted the radar returns to frequency 1. The waiting time will be dependent on the de-correlation time for frequency 1, which in this case is PRT_2 rendering a true time delay before resending frequency 1 of

$$t_{delay} = PRT_2 - N_{fq}\left(PRT_1 + Delay\right)$$

where N_{fq} equals the number of frequencies used.

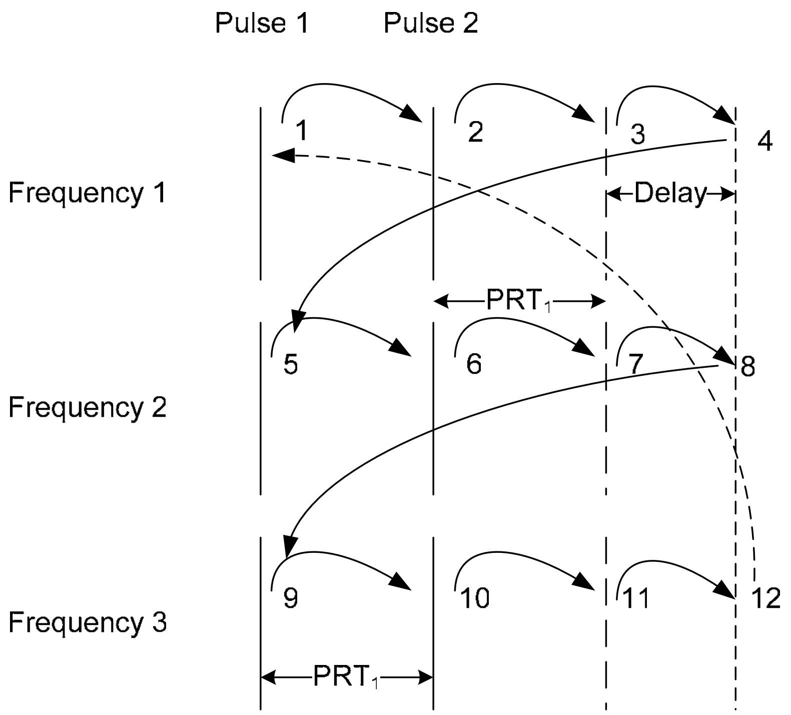


Figure 19. Frequency scheme using stepped frequencies to obtain de-correlation.

d. Pulse Compression

A benefit of Pulse Compression lies in the capability of averaging over wider range where the gain in range resolution provides independent samples that can be used to estimate the average power for the volume of the wider pulse. Implementing chirped pulse compression in the simulator calls for the ability to step in frequency as well as range. A scheme was developed enabling stepped frequency and range (see Figure 20), where the Resolution Cell is divided into smaller range bins defined by the sub pulse width and the number of frequency steps.

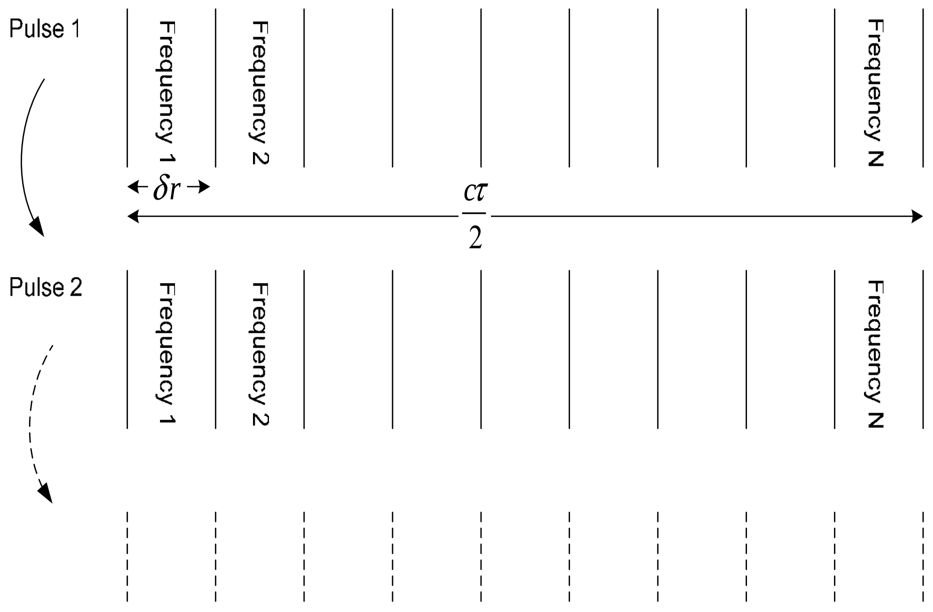


Figure 20. Pulse Compression implementation.

When the frequency has been stepped through the whole pulse, the drops are moved in accordance to the chosen PRT. The process then starts the frequency and range stepping again, collecting returns from the next pulse. The total power from each pulse is summed over all frequencies and then averaged for the number of range bins used.

e. Plot of Phasors for Phase Change Comparison

To visually illustrate the phase change for pulse pairs over time, 30 pulse pairs are presented in polar plots as phasors (see example Figure 21). The current version of the simulator can only plot 30 pulse pairs but uses all simulation pulse pairs to calculate the average radial velocity estimate. The estimated radial velocities retrieved from each pulse pair are plotted in a Histogram with a Gaussian fit for comparison, as presented in Figure 22. A three frequency stepped scheme simulation (pg. 44) produced an average radial velocity estimate of $\hat{\bar{v}}_r = -7.38$ m/s and a standard deviation of $\hat{\sigma}_v = 4.68$ m/s for a single run of 51 pulses calculated using Equations (2.32), (2.33) and (2.34) (see Figure 22).

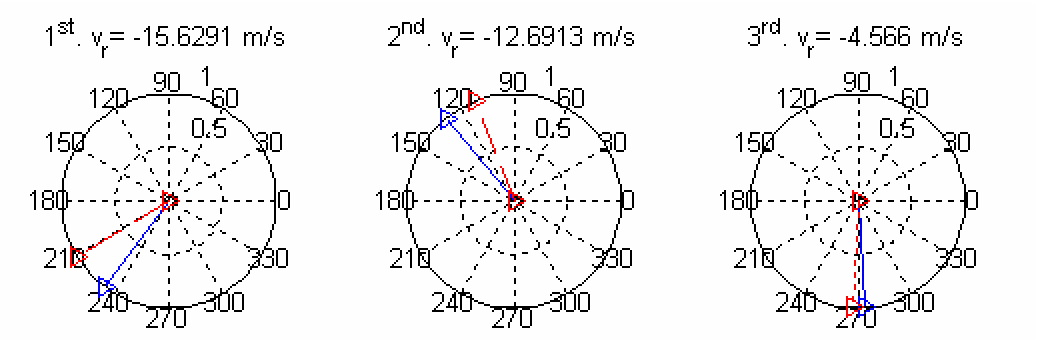


Figure 21. The three first pulse pairs plotted in a Phasor plot. The input radial velocity was -10 m/s with a velocity spread of 4 m/s.

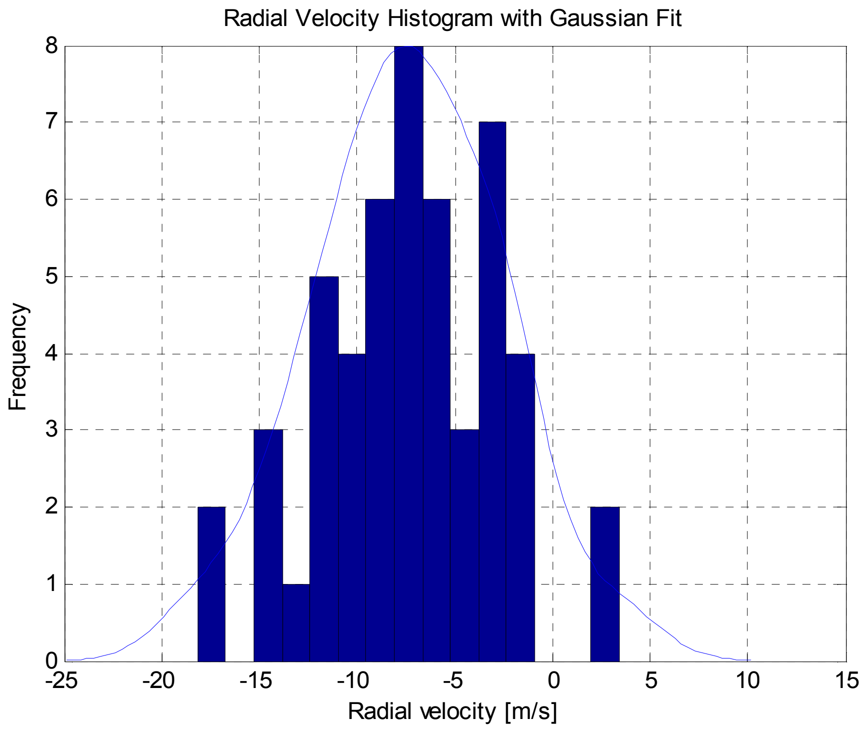


Figure 22. Radial Velocity Histogram with Gaussian Fit for 51 pulse pairs. Input velocity was -10 m/s with a velocity spread of 4 m/s.

f. Adding Spread to the Doppler Spectrum

As described in a previous section, a spread of the velocity spectrum needs to be implemented to make differences in correlation possible. Applying a Gaussian random component to the wind velocity to introduce a controllable value of velocity spread was tested. This enables verification of the theory of the spread contributing to de-correlation. Results from running simulations, again using a PRF of 2000 Hz and no fall angle, for different spreads applied can be observed in Figure 23. The spread applied was, from left to right, $\sigma_v = 4$ m/s, $\sigma_v = 3$ m/s, and $\sigma_v = 2$ m/s, and the average simulator velocity spread estimates were $\hat{\sigma}_v = 4.05$ m/s, $\hat{\sigma}_v = 2.94$ m/s, and $\hat{\sigma}_v = 2.00$ m/s again calculated using Equations (2.32), (2.33) and (2.34).

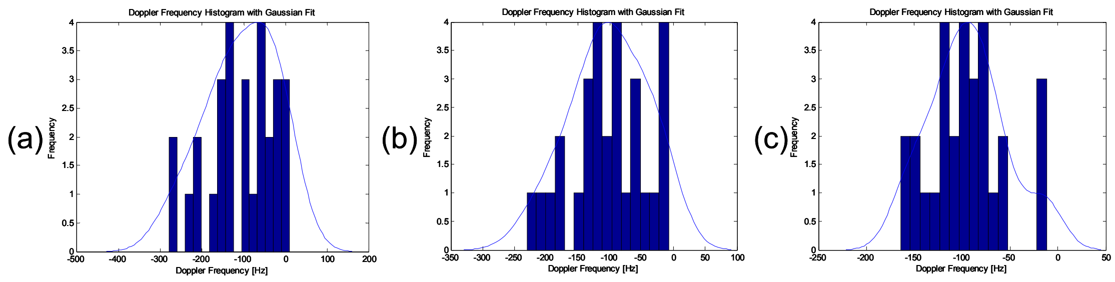


Figure 23. Test of Doppler spread. (a) input spread was 4 m/s, output 4.05 m/s. (b) input spread was 3 m/s, output 2.94 m/s. (c) input spread was 2 m/s, output 2.00 m/s.

The best de-correlation results were accomplished when the velocity spread was applied in two steps; first, a spread representing the spread of wind velocity over time, second, a spread representing the spread due to differences in drop sizes and other effects that will make the velocity of each drop vary over time. The fact that drops fall out of the resolution volume and new drops join adds an uncontrollable spread mechanism that can be examined by running several consecutive runs with the same input data. As the number of samples increases, the spread of the average velocity is expected to become smaller [3], which can be used for comparison.

D. SIMULATOR DESCRIPTION

The simulator was built in modules developed to enable desired functionalities. Initially the modules were separate functions that a main program called for execution, but MATLAB works faster when only one long script is used rather than functions. The inputs are separated into Radar data and Rain parameters and are specified in previous sections. Part I of the simulator uses input Radar data to generate a Beam Resolution cell approximation and a Cube with an added margin to the Beam Resolution cell to allow for the rain drops to fall the through beam over time. Part II uses the Rain Parameters to generate a fundamental T Matrix consisting of:

- Diameter Vector and Number of drops in each diameter interval using the Marshall-Palmer exponential approximation for drop size distribution.
- The backscatter RCSs for each Diameter size
- The total number of drops of each diameter per m^3

When the frequency stepped version is used the RCSs are placed in a separate matrix, since RCS is frequency dependent. In Part III all drops a placed randomly in the Cube developed in Part I and in Part IV the coherent Electric field return is calculated for every pulse. Between pulses all drops are moved in accordance with their velocities where velocity has a random component determined by σ_v. In Part V the final calculations are completed and the results are plotted.

1. Part I

To create the radar resolution cell based on the Radar data the volume of interest is retrieved by calculating the volume of the cone segment that represents the resolution cell (see Figure 9). Assuming an elliptical beam, the resolution volume can be approximated as [5]

$$V_{res} = \frac{\pi}{4} \cdot \left(r_i \theta_{3dB}\right)\left(r_i \phi_{3dB}\right)\left(\frac{c\tau}{2}\right). \tag{3.8}$$

To allow for the water drops to fall through the resolution cell over all pulses, a margin is added to the resolution cell creating a box where all drops are initially placed.

Both volumes are defined in spherical coordinates to allow for placement, plots, and later comparison. Also the volumes are registered.

2. Part II

This part produces a matrix of all drops that contribute to the total reflectivity of the resolution cell. Using the diameter limits defined in Rain parameters, a drop size vector is created. The limits can be chosen to create results close to the integral form by pushing the upper limit of diameter to 15 mm (see results Figure 12), or to resemble more realistic values setting the max diameter to around 6 mm. Depending on what limits are chosen, the actual *Z* must be computed using the appropriate model, i.e. chose between Equation (2.30) and (2.31). The simulator allows for differences in resolution for different diameter ranges depending on what one wants to examine. The diameter vector is then used to create an RCS vector applying Equation (2.40), and a Drop Size Distribution (DSD) vector applying Equation (2.35) (Marshall-Palmer). To improve the speed of the simulator the number of drops is scaled down by a factor (*Norm*), which is used later to rescale the output values. The number of drops is rounded off to nearest integer using the MATLAB function ROUND. Finally a matrix is created to store all values computed in Part II

$$matrix\_T\_round = \begin{pmatrix} D_1 & \cdots & D_N \\ \sigma_1 & \cdots & \sigma_N \\ N(D_1) & \cdots & N(D_N) \end{pmatrix} \tag{3.9}$$

and errors due to rounding off are calculated. The errors accounted for are RCS, *Z*, Number of Drops, and Rain Rate. In the stepped frequency and pulse compression versions the RCSs are stored in a separate matrix

$$rcs = \begin{pmatrix} \sigma_1^{fq1} & \cdots & \sigma_N^{fq1} \\ \sigma_1^{fq2} & \cdots & \sigma_N^{fq2} \\ \sigma_1^{fq3} & \cdots & \sigma_N^{fq3} \end{pmatrix}. \tag{3.10}$$

3. Part III

The drops created in Part II are placed inside the Box created in Part I. For each diameter every drop is randomly placed inside the Box using a uniform distribution function and its position is stored in spherical coordinates in a position matrix saved in a *.mat*-file

$$matrixpos = \begin{pmatrix} r_1 & \cdots & r_N \\ \phi_1 & \cdots & \phi_N \\ \theta_1 & \cdots & \theta_N \\ angle_1 & \cdots & angle_N \end{pmatrix} \tag{3.11}$$

where *angle* is the angle off boresight.

4. Part IV

In Part IV the Coherent Electric Field return is calculated for every pulse specified in the Radar Data. For every diameter, drops that are positioned inside the Resolution Cell are selected and based on matrices (3.9) and (3.11) the weighted electric field return is calculated using Equation (3.5) and summed for all drop diameters. Also the rain rate, for the specific Resolution Cell is calculated using Equation (2.39) approximated by a summation over all drop sizes. After the calculations, all drops are moved to a new position by applying the wind vector and terminal fall velocities specified in Rain parameters using Equation (2.37). All drops are assumed to move with the velocity of the wind vector with a Gaussian spread applied. The spread can be applied equally to all drops, so all drops move with the same velocity between pulses and the spread is applied over all pulses, or applied per diameter size so equally sized drops move at the same velocity between two specific pulses and the spread is applied over sizes. A third possibility is to apply the velocity spread on all drops individually creating a true spread although not fully realistic. The new drop positions are stored in *matrixpos*.

In the stepped frequency version, two different PRTs are used together with a delay described in previous sections. After the first pulse has been registered in terms of I and Q return, using frequency 1, the drops are moved according to the Doppler PRT (*PRT_1*). The phase is then registered completing the pulse pair before moving the drops again, this time using the Doppler PRT and the delay marking the change of frequency. Frequency 2 and frequency 3 are then applied in the same way as frequency 1. When changing back to frequency 1 the simulator uses the second PRT (*PRT_2*) and subtracts the time elapsed due to the other frequencies and their transmitted pulse pairs. The return values are stored in a matrix making sure the errors and the final estimates account for the differences in wavelengths and range between the returns.

The pulse compression version steps through the number of stretched pulses specified in the Radar data. For every stretched pulse the chirp is approximated by discrete frequency steps using the frequency step size and the number of steps defined. Every frequency is associated with a specific range bin defined by the Resolution Cell depth and the number of frequency steps. After the return values for all frequencies and range bins have been collected, all drops are moved and a new pulse initiated. The time between pulses is defined by the PRT.

5. Part V

The final part of the simulation produces the weather signal parameter estimates and plots the results. The estimated parameters are

- Average Power estimate, $\hat{\bar{P}}$, retrieved using the weighted electric field returns.
- Reflectivity estimate, $\hat{Z}$, retrieved from the average power estimate using (2.28) and compared with Z based on the simulator inputs using (2.27), evaluated by a summation and Equation (2.30), evaluated using the parameters N_0 and Λ from the Marshall-Palmer DSD.
- Average doppler frequency estimate, $\hat{\bar{f}}_d$, retrieved using a pulse pair algorithm that extracts the phase-change over time (see Equation (2.32)).
- Doppler spectrum width estimate, $\hat{\sigma}_f$, retrieved taking the standard deviation of the doppler frequency data (see Equation (2.33) and (2.34)).

Here the errors and the reduction factor are put back in the calculations. As a comparison, the ground truth for the respective parameter is derived using the integral form for calculation and plotted in the same graphs for clarity. Examples of output plots are presented in Figure 24 where a simulation using three frequency steps was evaluated. The input parameters include frequency steps from 3 GHz to 3.002 GHz with a 1 MHz frequency step, a Doppler PRF of 5,000 Hz, and the second PRF of 300 Hz. Further, the radial velocity was -10 m/s with a velocity spread of 4 m/s, which should produce a $\rho \approx 0.246$, using $T_s = \frac{1}{300}$ s. The average Doppler frequency estimate $\hat{\bar{f}}_d = -150$ Hz (Figure 24 (a)) is about 25 % from the expected Doppler frequency of $f_d = -200$ Hz. Only 51 sample pairs were used to estimate the average Doppler, which could explain the difference between the estimate and the true value. The simulation is a point estimate, which means that running consecutive trials would better show the behavior of the velocity estimate. The offset can also be the effects of adding new drops and loosing others in the resolution cell as explained in previous sections. The plot of Estimation of *Z*, (c) presents results close to the outcome expected using the mathematical formulas for comparison. The correlation value of 0.246 can explain why the value of the estimate $\hat{Z}$ is so close to the integral form, as de-correlated samples will yield an estimate with a smaller variance. The example output plot of correlation coefficient, (d), although close to the expected value from the above described simulation, is from another simulation running only one PRF. The input parameters were:

• Wavelength, λ	0.1 m
• PRF	300 Hz
• Wind	-10 m/s in x-direction (=radial velocity)
• Velocity spread	4 m/s

The expected correlation, using Equation (2.15), is $\rho \approx 0.246$, which is very close to the simulator output value.

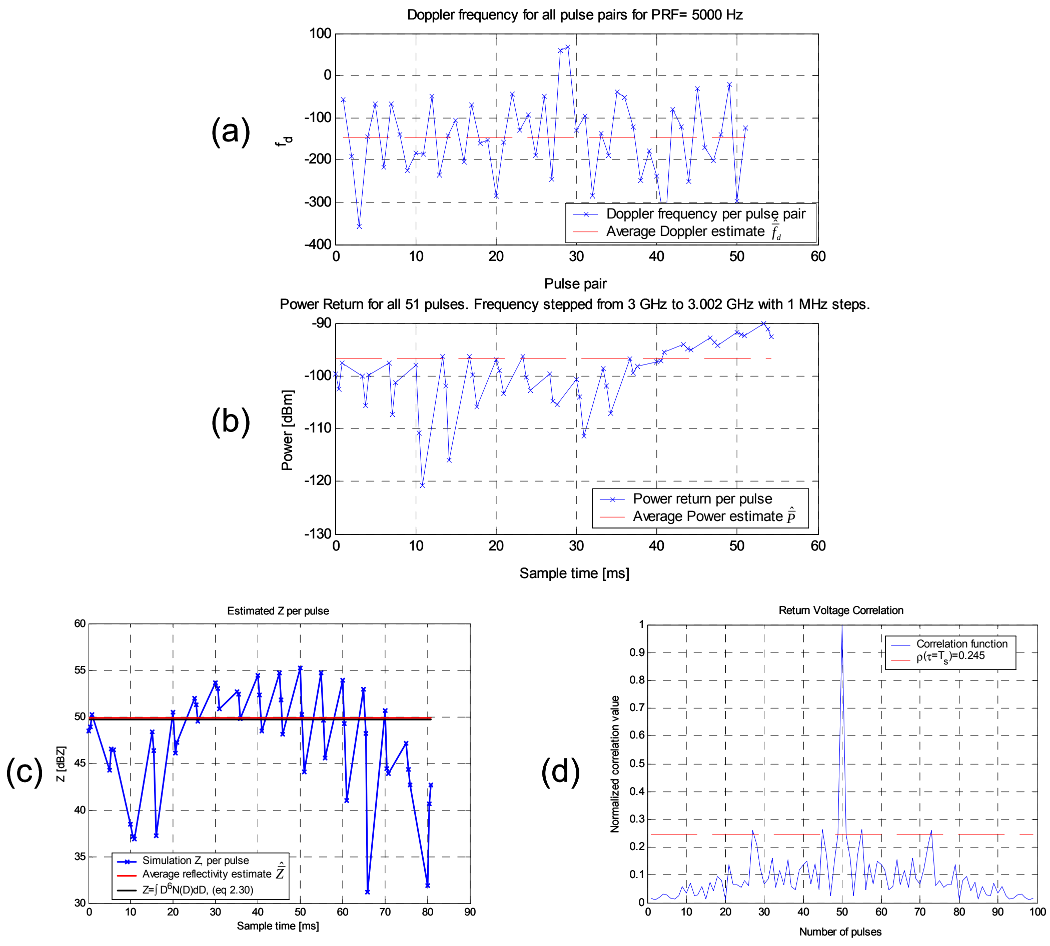


Figure 24. Example output plots from simulator. (a) Doppler, (b) power, (c) reflectivity, and (d) correlation coefficient. Note that (d) is from a different simulation than (a) – (c).

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ANALYSIS OF APPLICATIONS AND DISCUSSION OF RESULTS

A. IS IT POSSIBLE TO BUILD A WEATHER RADAR SIGNAL SIMULATOR BASED ON A PHYSICAL MODEL OF A SPATIAL REGION CONTAINING RAINDROPS?

The physical model built is based on the Rayleigh scattering approximation and the number of scatterers and their sizes are specified using the Marshal-Palmer drop size distribution. The drops are approximated by spheres and are uniformly distributed in a volume defined by the radar data and the return signal is calculated by summing the electric field. The model accounts for varying terminal velocities for different drops sizes and allow input for wind, which is represented by a 3-dimesional vector applied to every drop. The wind vector is implemented with a Gaussian spread among drops enabling investigation of correlation effects, which can be applied within or between pulses.

The results from running the simulator show that using the model developed will generate data following expected theoretical distributions. The simulator treats the rain drops as point targets adding the electric fields to generate the RCS. The results verify the assumption that the average return from rain can be approximated by adding the RCSs of all drops. The simulator can be adapted to explore effects of changes in DSD. Since the simulator distributes the drops randomly within the whole volume, including the resolution volume and its margins, the average distribution for sub-cells does not fully follow that of Marshal-Palmer. The effects of this should be further explored.

The current version of the simulator does not take into account attenuation, mutual coupling, and multiple scattering. Neither losses nor noise are introduced and dual polarization is left for future work. Although simplified, the weather radar signal simulator based on a physical model of a spatial region containing raindrops produced from this research work will allow for further development improving realism.

B. CAN REALISTIC MOTION THAT DEPENDS UPON THE TYPE OF WEATHER SYSTEM BE IMPARTED TO THE RAINDROPS?

This work has prioritized the electromagnetic and radar aspects and the meteorology portions have been kept to a minimum. The simulator version presented uses a vector model to introduce motion to the raindrops. This can be adapted to impart realistic motion. To do that, realistic models need to be acquired in vector form allowing implementation in the simulator. The current version of the simulator measures only one resolution cell exploring effects of de-correlation using different schemes. However, the simulator can be extended to cover a larger volume including several range bins. Using velocity spread as an input parameter makes it possible to simulate spectrum widening effects without having to create a detailed physical model of each spreading factor. There is no doubt that using a vector model would make it possible to implement realistic motion to the raindrops although this has not been done in this version of the simulator. It was beyond the scope of the thesis to fully implement realistic motion models.

C. CAN THE OUTPUT OF THE WEATHER RADAR SIMULATOR BE USED TO STUDY SPATIAL AND TEMPORAL SAMPLING SCHEMES FOR CONSTANT OR STEPPED FREQUENCY SAMPLING PULSES?

Two methods have been implemented focusing on temporal sampling; namely adapting Pulse Repetition Frequency so the drops have time to reshuffle between pulses, and Frequency shifting between sample pairs. A simulation was run with the following data (for details see appendix C):

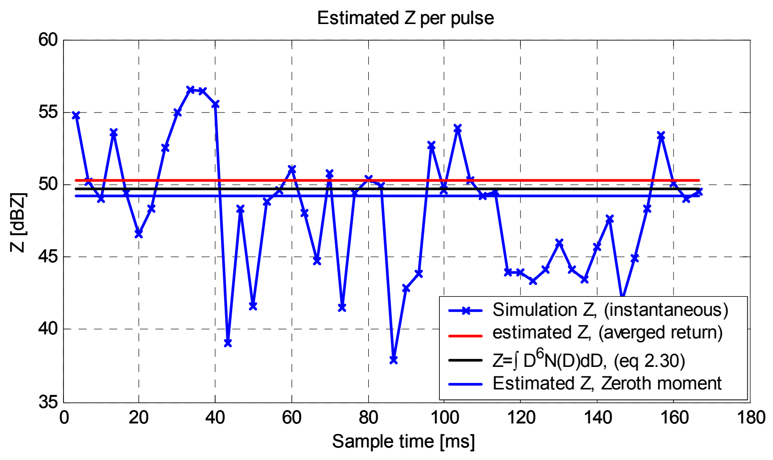
• Wavelength, λ	0.1 m
• PRF	300 Hz
• Wind	-5 m/s in x-direction (=radial velocity)
• Velocity spread	2 m/s

To determine a suitable PRF the sample time threshold provided in Equation (2.18) was used. The average $\hat{Z}$ of the simulator data lies within 0.5 dB of the integral form which is also true for the value retrieved using the power samples (see Figure 25). The reflectivity estimate based on sample power lies approximately 1 dB below the estimated average reflectivity, $\hat{\bar{Z}}$ and 0.5 dB below the ground truth, which can be explained by the approximation of the radar beam where only drops inside the half power beamwidth are considered. The drop position within the radar beam will also affect the gain level, which makes the position of the drops vital. A constant gain approximation might result in a more accurate approximation. The chosen PRF led to a level of correlation of $\hat{\rho} \approx 0.543$, which is just below the threshold of 0.6 but still enough to make it difficult to estimate radial velocity with reasonable accuracy. The sample pairs are not strongly correlated which makes an estimate possible but the variance will cause errors partly due to the lack of correlation, and partly because there are too few samples.

A new simulation was run this time using a rapid scanning scheme with stepped frequencies as described in a previous section. The new simulation was run using the same inputs but adding another PRF of 5000 Hz to improve the Doppler estimation capability.

The result were very similar to the first run using only time to acquire de-correlation, however, comparing the total time to produce 50 samples differ significantly, still accomplishing a *Z*-value close to the integral form (Figure 26). The reason for this is that the 300 Hz PRF only needs to be used to separate equal frequencies while stepping between frequencies can be done very quickly. Three de-correlated samples can now be collected in almost the same amount of time as previously only one pulse could be. The pulse pair scheme also enables an estimation of radial velocity seen in Figure 27. The spread was larger than the input of 2 m/s which can be explained by the fact that the spread was applied both between pulses and within pulses between different drop sizes.

(a)



(b)

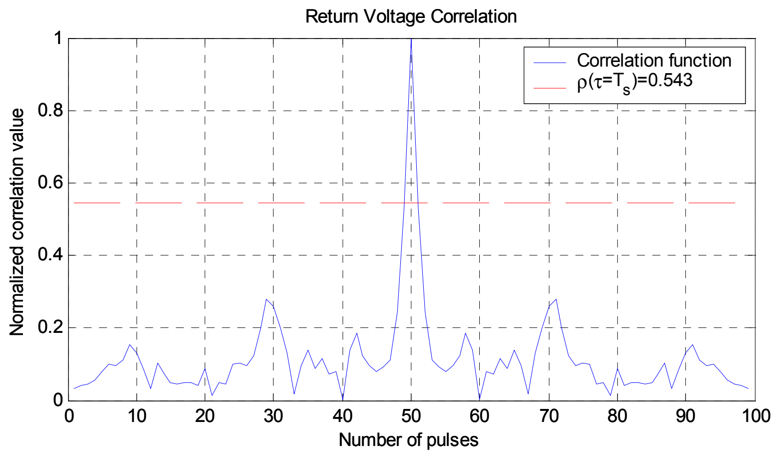


Figure 25. Temporal Sampling, PRF variation. PRF 300 Hz. Velocity Spread 2 m/s between pulses and within pulses for drops of different sizes.

The output from the latest simulation presented an average radial velocity of -5.25 m/s and a velocity spread of 4.6 m/s (using Equations (2.33) and (2.34)) which is close to the expected values.

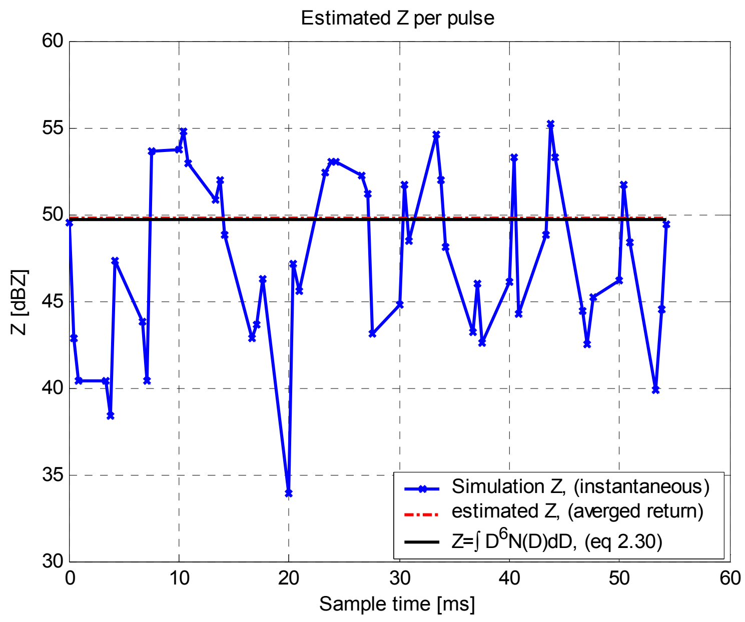


Figure 26. Temporal Sampling, Stepped Frequency. PRF$_1$ 5000 Hz, PRF$_2$ 300 Hz. Velocity Spread 2 m/s between pulses and within pulses for drops of different sizes.

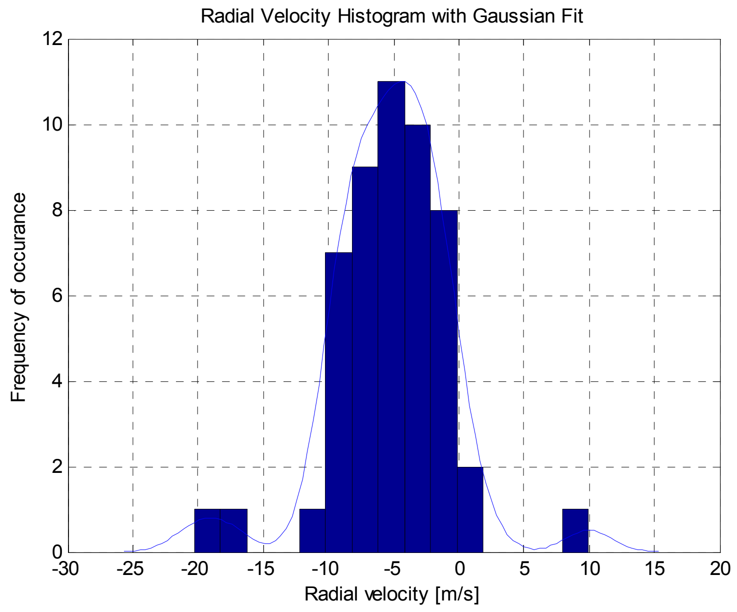


Figure 27. Velocity histogram with Gaussian Fit, Temporal Sampling using stepped frequency. The input velocity was -5 m/s. $\hat{\sigma}_v = 4.6$ is estimated using Equations (2.33) and (2.34).

To analyze the simulator behavior over time for several consecutive runs, a simulation series was performed using 50 runs and the results were analyzed for different number of pulses. In [2] the intensity probability distribution for the averaged return for different numbers of pulses was plotted as in Figure 28, where the dependence of the number of pulses can be seen.

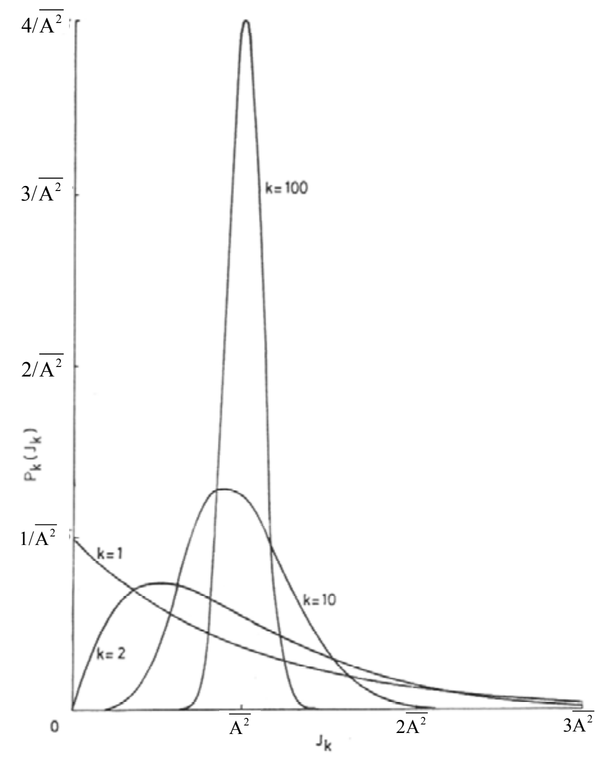


Figure 28. Probability distribution of J_k, which is the return intensity. The intensity average is $\overline{A^2}$ (From [2]).

Running the simulator using a $\lambda = 0.1$ m, and a PRF of 200 Hz, which represents a correlation coefficient of about 0.04 (Equation (2.15)) considering a velocity spread of 4 m/s, renders results seen in Figure 29. The figures resemble each other in shape but for $N = 100$ samples, the peak for the simulator distribution is not as high as the result in Figure 28. As the number of pulses goes up the distribution become more Gaussian, which is expected for any distribution if the number of samples increases. Also, the spread decreases as the number of pulses increases. Having completely independent samples will change the average velocity spread as $\sigma_{\overline{x}} = \frac{\sigma}{\sqrt{k}}$, where *k* is the number of pulses or samples. The results are somewhat expected since the distribution of the power return is expected to be exponential [3]. The standard deviation of an exponential distribution is expected to be equal to the mean of the same, which the results verify. However, if samples are independent the spread should decrease with a factor of $1/\sqrt{k}$ as k increases. For 2 pulses the expected value equals $1/\sqrt{2} = 0.71$, which matches the simulator result. For 10 pulses the value is 0.316, while the simulator shows 0.41. Finally for 100 pulses the value is 0.10. The simulator shows 0.21, which is more than the double what is expected. This suggests that not all samples can be considered independent. Why this occurs when a sufficient spread is applied cannot be explained at this point but is of relevance for future investigations.

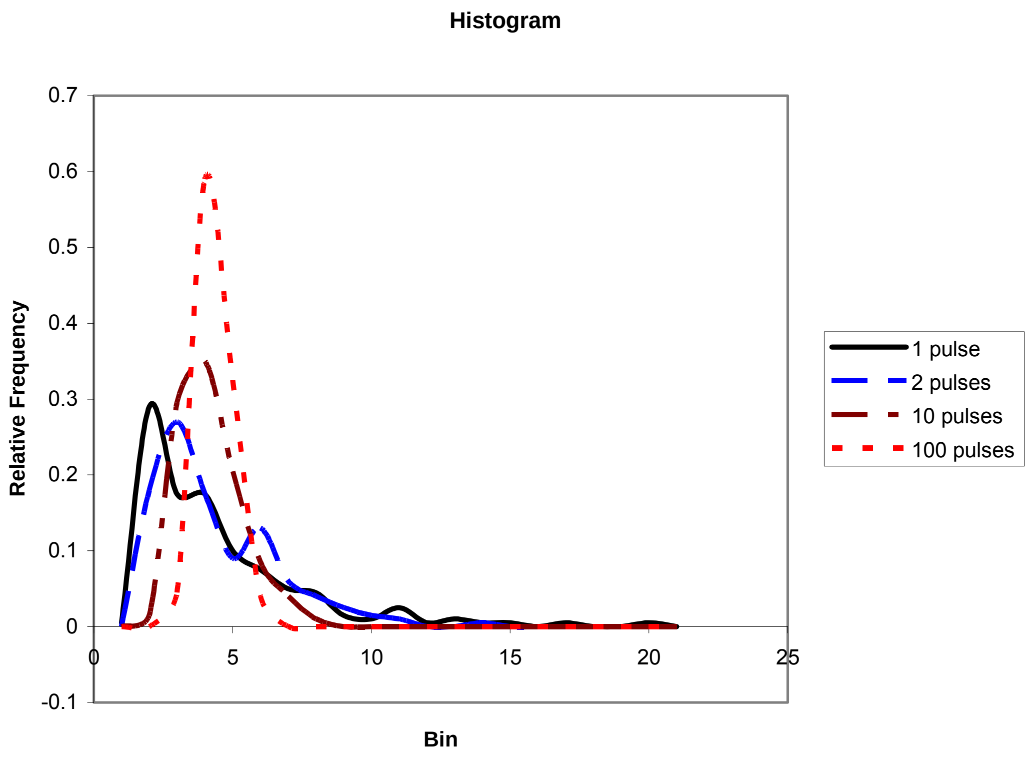


Figure 29. **Histogram showing distribution of intensity (power) from 200 consecutive runs. Number of pulses averaged is 1, 2, 10 and 100. $\hat{\rho} \approx 0.04$ so independent samples are expected. The relative standard deviation ($\hat{\sigma}_{power} / \hat{\bar{P}}$) in each case is, 1 pulse=0.99, 2 pulses=0.74, 10 pulses=0.41, and 100 pulses=0.21.**

D. CAN THE WEATHER RADAR SIMULATOR BE USED TO STUDY THE USE OF PULSE COMPRESSION AND RANGE AVERAGING AS A MEANS OF RAPIDLY OBTAINING INDEPENDENT SAMPLES?

A simulation was performed using the simulator version adapted to pulse compression as described in a previous section. The resolution volume differs from the previous simulations because of the stretched pulse making the range dimension of the resolution cell 10 times larger. All other data was set as before, i.e. PRF of 300 Hz, wind speed of 5 m/s and a velocity spread of 2 m/s. The number of pulses is still 50 but they are used differently letting every pulse cover a range bin equal to previous runs. The new stretched pulse consists of 10 sub-pulses of different frequencies creating the chirp. Each new pulse is summed over all frequencies stepped over the whole resolution cell and then averaged. Effectively only 5 pulses are used, but since each pulse consists of 10 frequency steps, the average value is based on 50 samples.

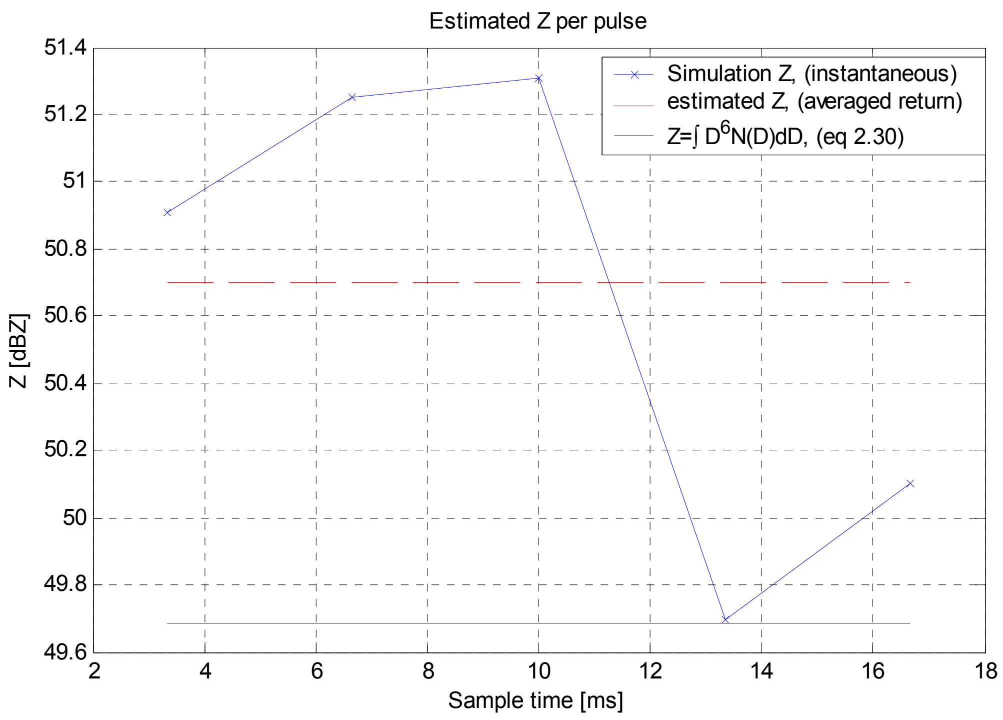


Figure 30. Pulse Compression simulation. 5 pulses, Frequency stepped from 3 GHz to 3.0018 GHz with 0.2 MHz steps for each pulse.

The estimate of *Z* based on averaging lies 1.0 dB above the result using the integral form, which indicates good accuracy. The smaller deviation between samples compared with previous simulations can be explained by the fact that each value really represents an average of 10 range bins. The higher level can be explained by the method used to average. The volume of the different resolution bins increases moving away from the radar so if this is not taken into account when averaging, the values will end up deviated from the expected value. Another explanation is the fact that the spread in frequency is not enough to ensure sufficient de-correlation to get independent samples. Running the simulator for 8 stretched pulses generated a result very close to the integral form (0.4 dB) something that needs further investigation for verification.

The results show the possibility to use the simulator to study the use of pulse compression and range averaging as a means of rapidly obtaining independent samples.

E. CAN THE OUTPUT OF THE WEATHER RADAR SIGNAL SIMULATOR BE USED TO PRODUCE DISPLAYS OF PEDAGOGICAL INTEREST?

During the progress of building the simulator, several displays have been developed and used to assist with the evaluation of each module. Once the simulator output data has been produced it is all numbers making it possible to post process in many different ways, displays being one. To be able to use the simulator in teaching, visual aids presenting the signal pulse-to-pulse were developed as well a phasor representation displaying the phase change over time between pulse pairs. Adding a Graphical User Interface, GUI, would probably enhance the utility the simulator as a pedagogical tool for investigating the effects of different weather phenomena and variation of input values for both radar and weather parameters. Nevertheless, the output of the existing simulator can be used to produce displays of pedagogical interest.

F. CAN THE OUTPUT OF THE WEATHER RADAR SIGNAL SIMULATOR BE USED TO STUDY THE PERFORMANCE OF ESTIMATORS FOR THE FIRST THREE DOPPLER MOMENTS?

The first versions of the simulator only allowed for changes in a single PRF making it possible to vary time between pulses effecting de-correlations. The simple form of return signal produced samples equally spaced in time making it simple to perform post processing in the form of auto correlation and FFT. The spectrum was plotted and it was possible to derive an estimated radial velocity of the drops from the average Doppler frequency. When the complexity of the return signal increased and the simulator allowed for multiple PRFs and delays, the signal needed manipulation to enable FFT or autocorrelation. Initial trials padding the signal with zeros making the samples equally spaced in time were performed but adding time to a signal demands more signal processing to remove effects of folding and aliases. Digital signal processing is beyond the original scope of the thesis which is why this interesting trail was abandoned. There are no test runs verifying that the simulator can be used to study the performance of estimators for the first three Doppler moments but that does not imply that the simulator cannot be used for this application. Having a complex return signal allows for post processing of many sorts, as mentioned earlier.

G. CAN THE OUTPUT OF THE WEATHER RADAR SIGNAL SIMULATOR BE USED TO STUDY THE UTILITY OF ESTIMATORS FOR HIGHER ORDER DOPPLER MOMENTS?

Since the estimation of the lower order of estimators was not performed, the higher order estimators have not been investigated.

V. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE DEVELOPMENT AND WORK

A. CONCLUSIONS

The objective of the research was to build a weather radar signal simulator delivering I and Q channel outputs from a physical representation of rain drops. A previously developed simulator was initially evaluated and verified serving as the basic building block when developing the final version. The Radiation Integrals were used to derive the electromagnetic scattering from rain drops approximated as dielectric spheres.

Without accounting for multiple scattering, attenuation, or coupling effects, the scattered electric field from each drop is summed for every pulse providing the total RCS of the rain in the radar resolution cell as well as a complex voltage return representation. Using the Marshall-Palmer drop size distribution, an initial position for every drop is determined within an extended volume enabling the drops to fall through the radar beam for all samples. Between pulses the drops are moved according to their respective terminal velocities based on the Atlas-Ulbrich approximation and wind. To allow for width in the velocity spectrum, a Gaussian velocity spread is applied to the drops between and within pulses. The simulator power return is weighted using the Radar Range Equation for point targets, considering all drops as point targets, and evaluated as a volume target using the Weather Radar Range Equation.

Several different functionalities have been implemented allowing for stepped frequencies, multiple PRFs, pulse compression using a chirp, and variation of radar and weather input parameters. Post processing capabilities include autocorrelation and FFT (only for single PRF version); computation of weather parameter estimates such as reflectivity factor, *Z*; average doppler, radial velocity, and velocity spread; pedagogical plots including a Phasor plot of phase change over time, a velocity histogram, power and reflectivity for each pulse over time.

This research has shown that building a weather radar signal simulator based on a physical model of a spatial region containing raindrops is possible. The benefit of using a physical model is improved realism and the ability to study effects of scientific interest.

Also, the model permits isolation of effects making it possible to explore the results when varying some parameters while keeping others fixed. Using a vector model to input wind and other effects on the rain drops makes it possible to impart realistic motion to the drops although this has not been done fully in this version of the simulator. This was beyond the scope of this thesis but is a recommended area for further study. Evaluating the result from averaging radial velocity rendered an interesting finding namely the difference between using FFT, autocorrelation, and pulse pair processing to estimate radial velocity. Using the FFT or autocorrelation skewed the result towards the bigger drops because the amplitude information is kept in the process. On the other hand with the pulse pair algorithm, only the phase information is used, resulting in a "true" average velocity for all drops regardless of size. Intuitively smaller drops would be more susceptive to wind than bigger drops which would emphasize the need for pulse pair algorithms over the FFT or autocorrelation for better wind velocity estimates. No theoretical investigation or look for research in this area has been performed but is suggested for future studies. Different implementations of velocity spread have been tested leading to variations in results. The final version of the simulator allows for spread both between pulses, which represents how the average velocity is spread between samples; and within pulses with a size dependent spread, which represents the spread within the resolution cell.

The title of this research indicates where the priority lies, namely developing the simulator to examine sampling rates and scanning schemes. The results show that it is possible to use the weather radar simulator to study spatial and temporal sampling schemes for constant or stepped frequency sampling pulses. The research aimed at building the radar and not to fully explore the effects of different rate and schemes, but the results from testing and verification indicate that using methods to more rapidly obtain independent samples decreases the time to obtain accurate estimates of weather signal parameters. By introducing three frequency steps, the time to acquire accurate data was reduced by a third, still being able to acquire Doppler information as well as reflectivity. However, more simulations need to be performed comparing results when parameters are varied to fully explore the benefits of frequency stepping and its limitations.

By implementing a pulse compression capability, using frequency stepping and range averaging, results were provided showing the ability to rapidly obtain independent samples using pulse compression. Increasing the radar resolution cell slows down the simulator, which can be compensated by increasing the *Norm*-factor (see previous chapters for details). Again further testing is suggested exploring the effects of varying parameters.

Post processing the signal samples provides wide-ranging opportunities for different displays of pedagogical interest. Most plots have been developed to compare and verify results but some plots have been developed only to display pedagogically interesting phenomena. Being able to follow phase change over time is an example of the latter. Implementing a GUI will probably be the greatest improvement in this area.

Having an accurate signal allows for more than just plots of results, it opens up possibilities for post processing schemes beyond the scope of this work. Previously indicated processing includes filtering, FFT and autocorrelation. In the introduction, cross scientific benefits were mentioned signifying the use of knowledge from one area in another. Having a weather signal consisting of rain echoes can be used to show or investigate filtering to remove weather clutter.

The initial intentions included ideas of improving the simulator speed by optimizing algorithms and scripts. This has not been accomplished. However, the simulator runs a simulation locally on a PC with an Intel Pentium 4 processor 550 (3.4GHz, 1M, 800MHz FSB, NTFS Memory 4.0GB DDR2 Non-ECC SDRAM, 400Hz (4DIMM)), in just a few minutes, depending on complexity and input parameters.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

The simulator is ready to do its job, setting more than one path for future work. The focus in this research has been to develop a tool for research, which leaves the weather research parts for future work. Two main areas of development and research are proposed:

- Examine sampling rates and scanning schemes for fast volumetric scanning and update

- Simulator development

1. Examine Sampling Rates and Scanning Schemes for Fast Volumetric Scanning and Update

Weather Radar Signal Simulator should be run to examine weather signal parameter estimation sensitivity when varying radar parameters and precipitation estimator models where changing the drop size distribution would be of interest. Comparison should be made to real life measures as well as theoretical models. To complement this work weather models should be explored to find the physics behind different weather phenomena enabling implementation in the simulator. Alterations in the simulator might be necessary to facilitate this implantation.

2. Simulator Development

Even though the final version of the simulator has several interesting functionalities many interesting effects and functionalities can be added.

a. Adding a GUI

The main priority developing the simulator should be to create a GUI making the tool more user-friendly.

b. Modeling Non-Spherical Drops

The current simulator approximates each drop, regardless of size, as a dielectric sphere. Research show that drops flatten [9] while falling so an oblate spheroid approximation would be more appropriate. Work done by Bringi and Chandrasekar [14] provide good mathematical tools enabling implementation in the simulator.

c. Adding Dual Polarization

Implementing a spheroid approximation makes dual polarization measurements possible and new estimates can be computed such as differential reflectivity. The complexity of this can be overwhelming considering effects like cross talk and multiple scattering. This is a problem for scientists or students who are searching for a real challenge.

d. Post Processing for Higher Order Moments

The output signal can be post processed to produce the power spectrum, which was done in the early versions that had uniform time spacing between samples. Applying digital signal processing to the output signal, retrieving the power spectrum would make it possible to explore higher order moments of the spectrum and to evaluate their utility as estimators.

e. Linking Simulator Input to Storm Physics

As mentioned, storm models should be explored to determine the physics associated with different weather phenomena. The physics can be used to determine the wind velocity and velocity spread that should be used as simulator input parameters. This will add to the realism of the simulator and will make it possible to explore effects on the radar signal caused by different weather phenomenon.

f. Adding Noise to the Weather Signal

Real weather signals will always be contaminated by some noise. Noise from all sources should therefore be added as a set of inputs. The benefit of this would be the ability to study the degradation caused by noise.

g. Adding Mutual Coupling

Mutual coupling was examined but needs more attention and different techniques to enable implementation into the simulator. For large rainfall rates, raindrops may be spaced at distances that result in mutual coupling. The total RCS in this case will be less than that computed assuming no coupling. This effect needs further investigation to determine what storm types and rain rates result in coupling that renders the independents scattering assumption invalid.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. DERIVATION OF RCS FOR SINGLE DROP

A water drop can be approximated as a small dielectric sphere. Let the sphere radius be *a*, and the permittivity ε, and permeability μ_0, also let the sphere be centered at the origin. An incident plane wave is present

$$\vec{E}_i = \hat{z}E_0 e^{-jkx}$$

The electric field at a point in space can be expressed as

$$\vec{E} = \vec{E}_i + \vec{E}_s$$

Using the volume equivalent principal, the current $\vec{J}$ can be written as

$$\vec{J} = j\omega(\varepsilon - \varepsilon_0)\vec{E}$$

Letting $a \gg \lambda$, $\vec{E}$ is then approximately $\vec{E} = \hat{z}E_1$ (constant inside). Using $\vec{E}_s = \dfrac{j\vec{J}}{3\omega\varepsilon_0}$ E_1 can be derived as

$$\hat{z}\vec{E}_1 = \vec{E} = \vec{E}_i + \vec{E}_s = \vec{E}_i + \frac{j\vec{J}}{3\omega\varepsilon_0}$$

$$\hat{z}\vec{E}_1 = \vec{E}_i + j\frac{j\omega(\varepsilon_r\varepsilon_0 - \varepsilon_0)\vec{E}}{3\omega\varepsilon_0} = \vec{E}_i - \frac{(\varepsilon_r - 1)\hat{z}\vec{E}_1}{3}$$

$$\hat{z}\vec{E}_1 = \frac{\vec{E}_i}{1 + \dfrac{(\varepsilon_r - 1)}{3}} = \frac{3\vec{E}_i}{(\varepsilon_r + 2)}$$

Now solving for $\vec{J}$

$$\hat{z}\vec{E}_1 = \frac{3\vec{E}_i}{(\varepsilon_r + 2)}$$

$$\vec{E} = \frac{\vec{J}}{j\omega(\varepsilon - \varepsilon_0)}$$

$$\vec{J} = \frac{j3\omega\varepsilon_0(\varepsilon_r - 1)\vec{E}_i}{(\varepsilon_r + 2)}$$

Finally putting the current in the radiation integral

$$\vec{E}(r,\theta,\phi) = \frac{-jk\eta}{4\pi r} e^{-jkr} \iiint \vec{J} e^{jkg} dv', g = 0$$

$$\vec{E}(r,\theta,\phi) = \frac{-jk\eta}{4\pi r} e^{-jkr} \frac{j3\omega\varepsilon_0(\varepsilon_r - 1)\vec{E}_i}{(\varepsilon_r + 2)} \int_0^a dx' \int_0^a dy' \int_0^a dz'$$

$$\vec{E}(r,\theta,\phi) = \frac{-jk\eta}{4\pi r} e^{-jkr} \frac{j3\omega\varepsilon_0(\varepsilon_r - 1)\vec{E}_i}{(\varepsilon_r + 2)} \frac{4\pi a^3}{3}$$

$$\vec{E}(r,\theta,\phi) = \frac{k}{r}\sqrt{\frac{\mu_0}{\varepsilon_0}} e^{-jkr} \frac{2\pi c\varepsilon_0(\varepsilon_r - 1)\vec{E}_i}{\lambda(\varepsilon_r + 2)} a^3$$

$$\vec{E}(r,\theta,\phi) = \frac{k^2}{r} e^{-jkr} \frac{(\varepsilon_r - 1)}{(\varepsilon_r + 2)} \vec{E}_i a^3$$

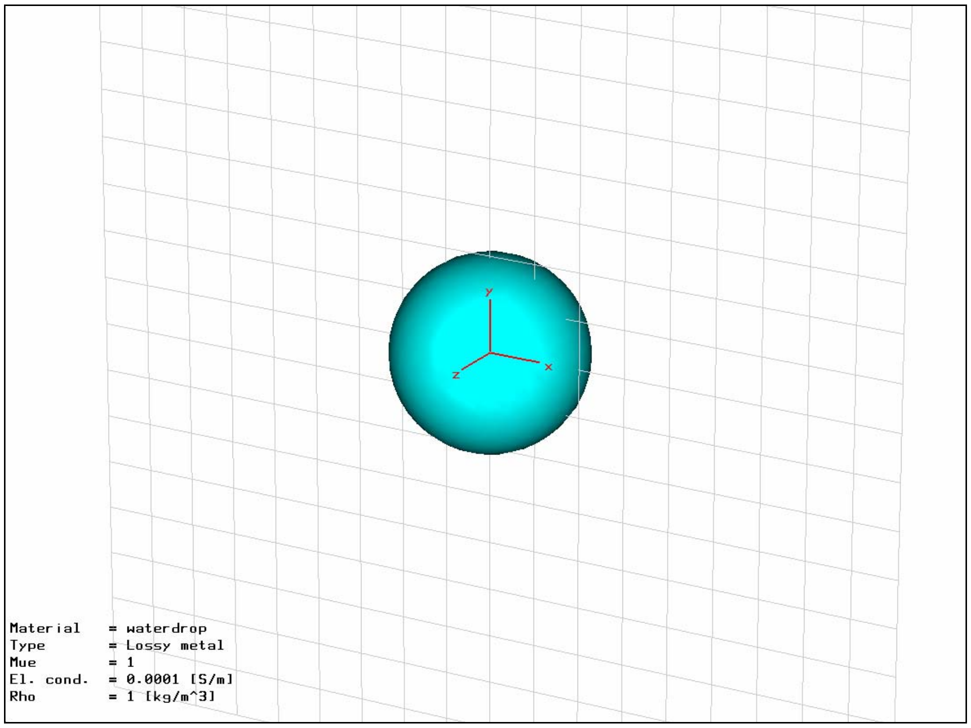
Using this in the formula for RCS renders, (letting $a = D/2$)

$$\sigma_b = 4\pi r^2 \left|\frac{\vec{E}_s}{\vec{E}_i}\right|^2 = 4\pi r^2 \left|\frac{\frac{k^2}{r} e^{-jkr} \frac{(\varepsilon_r - 1)}{(\varepsilon_r + 2)} \vec{E}_i a^3}{\vec{E}_i}\right|^2 = \frac{4\pi k^4 a^6 (\varepsilon_r - 1)^2}{(\varepsilon_r + 2)^2}$$

$$\sigma_b = \frac{\pi^5 (\varepsilon_r - 1)^2}{\lambda^4 (\varepsilon_r + 2)^2} D^6$$

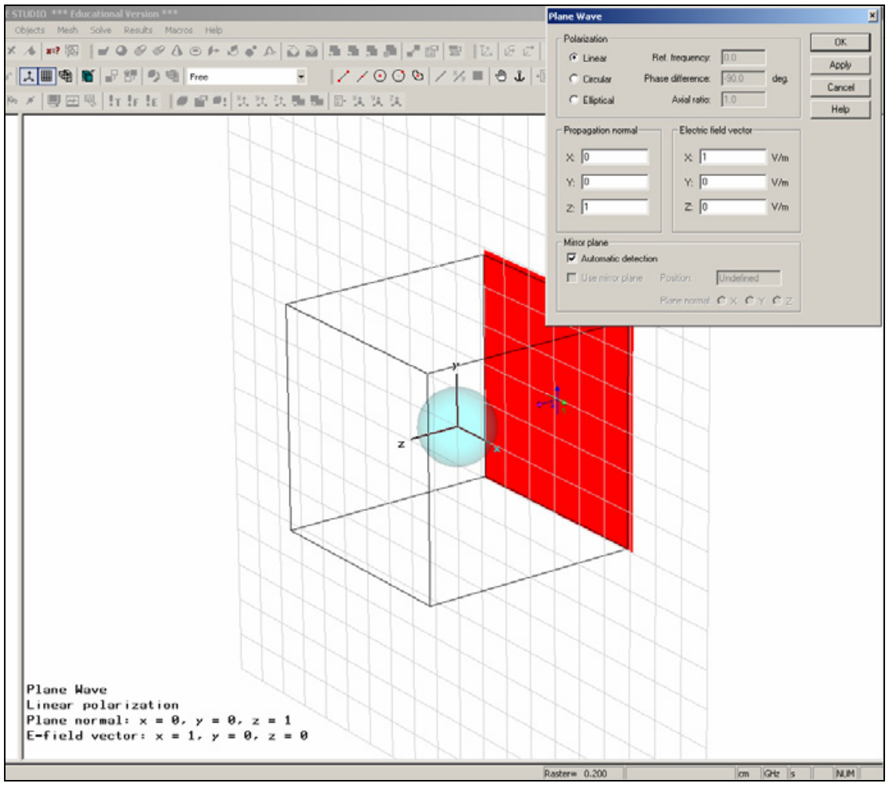
APPENDIX B. MICROWAVE STUDIO SIMULATION SETUP

First the target(s) was created. All targets were spheres of 4 mm in diameter with the specifications of a water drop. The incident ray was a plane wave in the z-direction with polarization in the x-direction or in the y-direction. When adding the second drop, the drop was placed side by side with the first drop, i.e. in the x-direction. The plots are farfield plots in spherical coordinates. The mesh resolution was set to 20 lines per wavelength.



```
Material    = waterdrop
Type        = Lossy metal
Mue         = 1
El. cond.   = 0.0001 [S/m]
Rho         = 1 [kg/m^3]
```

Figure 1. Target. Water drop.


STUDIO *** Educational Version ***
Objects Mesh Solve Results Macros Help
Free
Raster= 0.200
cm
GHz
s
NUM
y
z
x

Plane Wave

Polarization
- (●) Linear — Ref. frequency: 0.0
- ( ) Circular — Phase difference: -90.0 deg.
- ( ) Elliptical — Axial ratio: 1.0

Propagation normal
- X: 0
- Y: 0
- Z: 1

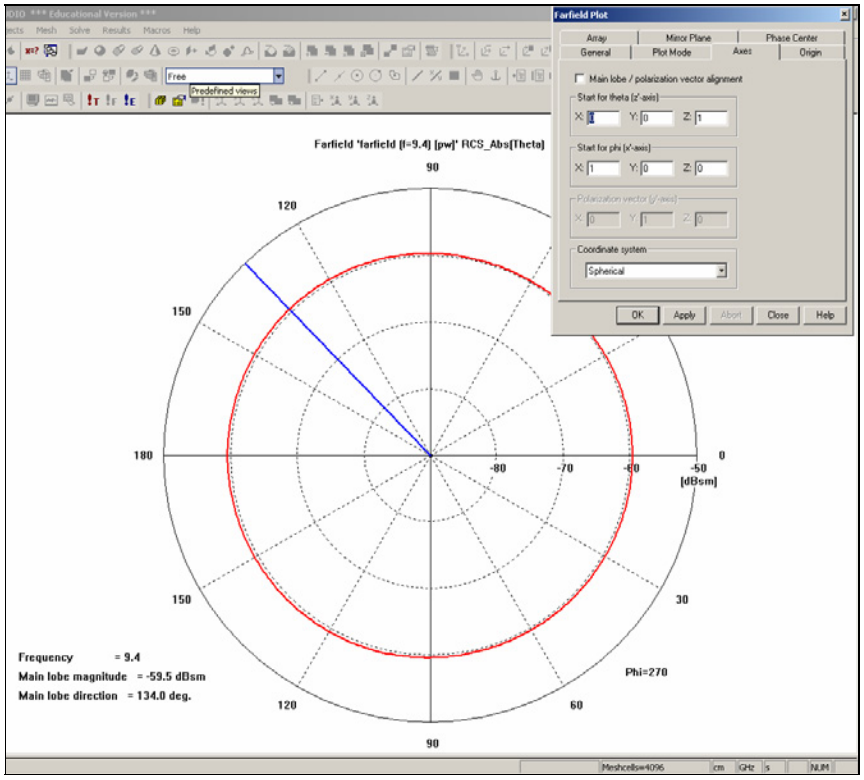
Electric field vector
- X: 1 V/m
- Y: 0 V/m
- Z: 0 V/m

Mirror plane
- [✓] Automatic detection
- [ ] Use mirror plane — Position: Undefined
- Plane normal: ( ) X ( ) Y ( ) Z

OK | Apply | Cancel | Help

```
Plane Wave
Linear polarization
Plane normal: x = 0, y = 0, z = 1
E-field vector: x = 1, y = 0, z = 0
```

Figure 2. Incident Plane for the one-drop simulation


IDIO *** Educational Version ***
ects Mesh Solve Results Macros Help
Free
Predefined views
Farfield 'farfield (f=9.4) [pw]' RCS_Abs(Theta)
90
120
150
180
150
120
90
60
30
0
-80
-70
-60
-50
[dBsm]
Phi=270
Meshcells=4096
cm
GHz
s
NUM

Farfield Plot

Array | Mirror Plane | Phase Center
General | Plot Mode | Axes | Origin

- [ ] Main lobe / polarization vector alignment

Start for theta (z'-axis)
- X: 1 Y: 0 Z: 1

Start for phi (x'-axis)
- X: 1 Y: 0 Z: 0

Polarization vector (y'-axis)
- X: 0 Y: 1 Z: 0

Coordinate system
- Spherical

OK | Apply | Abort | Close | Help

```
Frequency            = 9.4
Main lobe magnitude  = -59.5 dBsm
Main lobe direction  = 134.0 deg.
```

Figure 3. Type of plot and definition of angles.

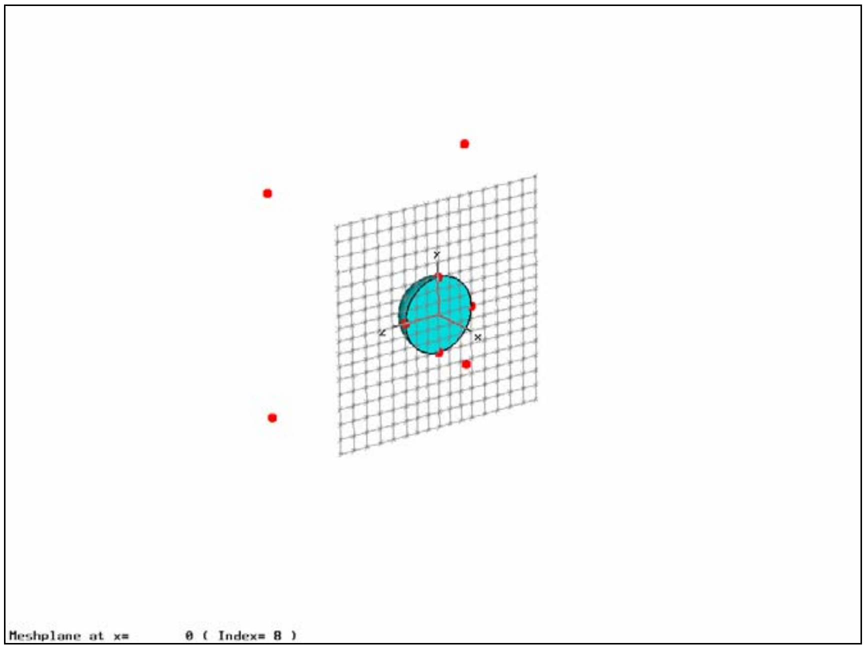


Figure 4. Resolution of the Meshed grid.

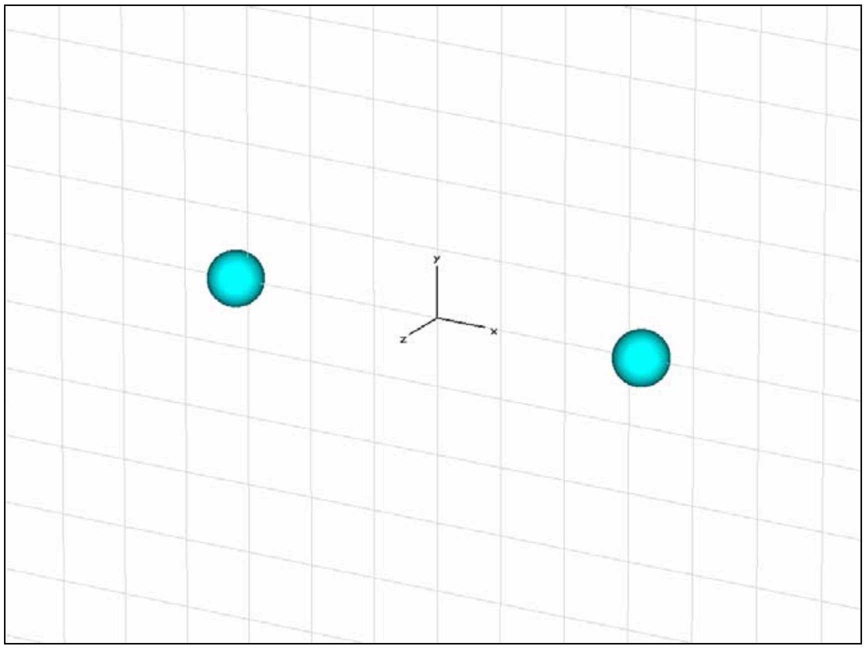


Figure 5. Two drop set-up.

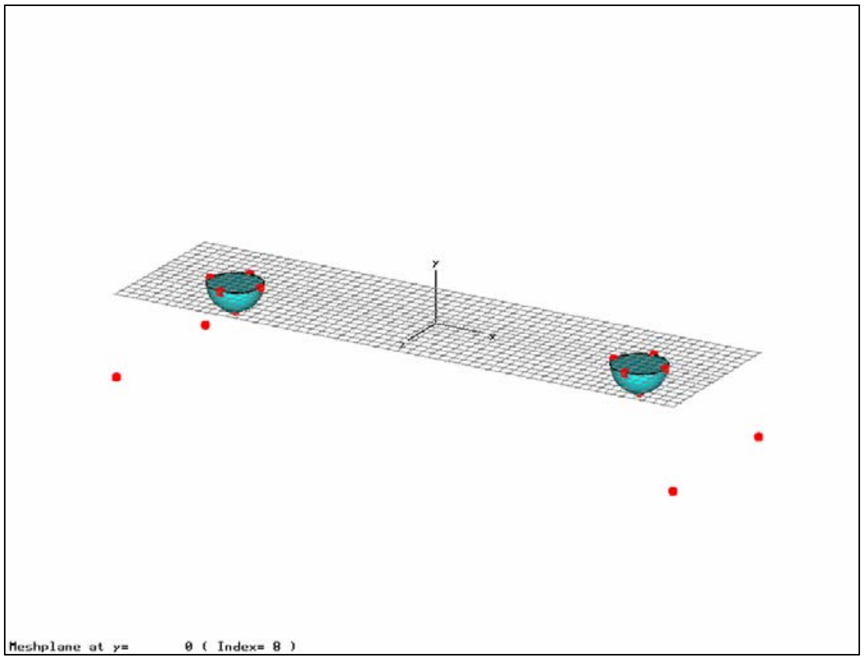
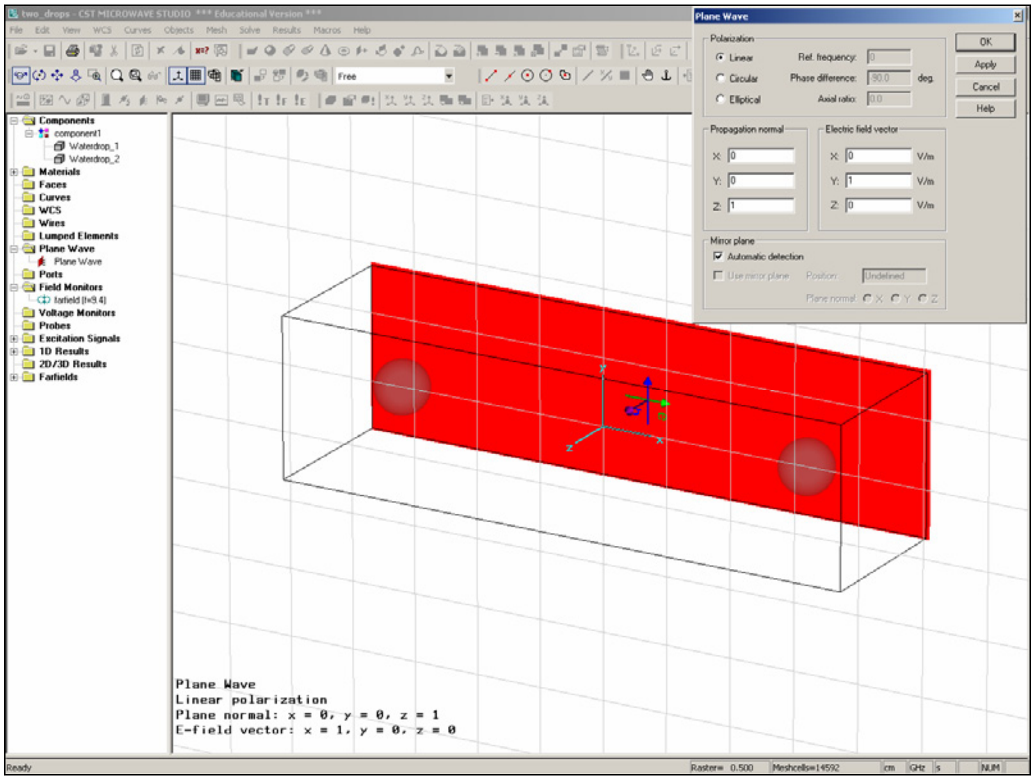


Figure 6. Resolution of the Meshed grid for two drops.



Plane Wave

Polarization: [✓ Linear] Circular Elliptical

Ref. frequency: 0

Phase difference: -90.0 deg.

Axial ratio: 0.0

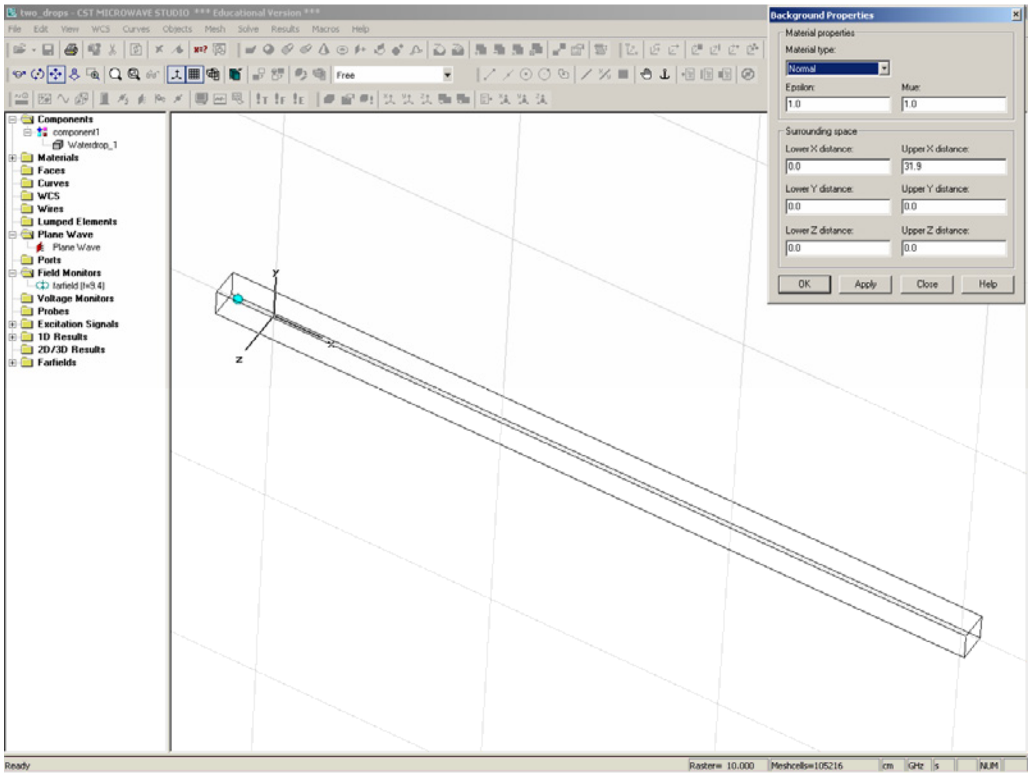
Propagation normal: X: 0 Y: 0 Z: 1

Electric field vector: X: 0 V/m Y: 1 V/m Z: 0 V/m

Mirror plane: [✓ Automatic detection] Use mirror plane Position: Undefined Plane normal: X Y Z

```
Plane Wave
Linear polarization
Plane normal: x = 0, y = 0, z = 1
E-field vector: x = 1, y = 0, z = 0
```

Figure 7. Incident Plane for the two-drop simulation



Background Properties

Material properties

Material type: Normal

Epsilon: 1.0 Mue: 1.0

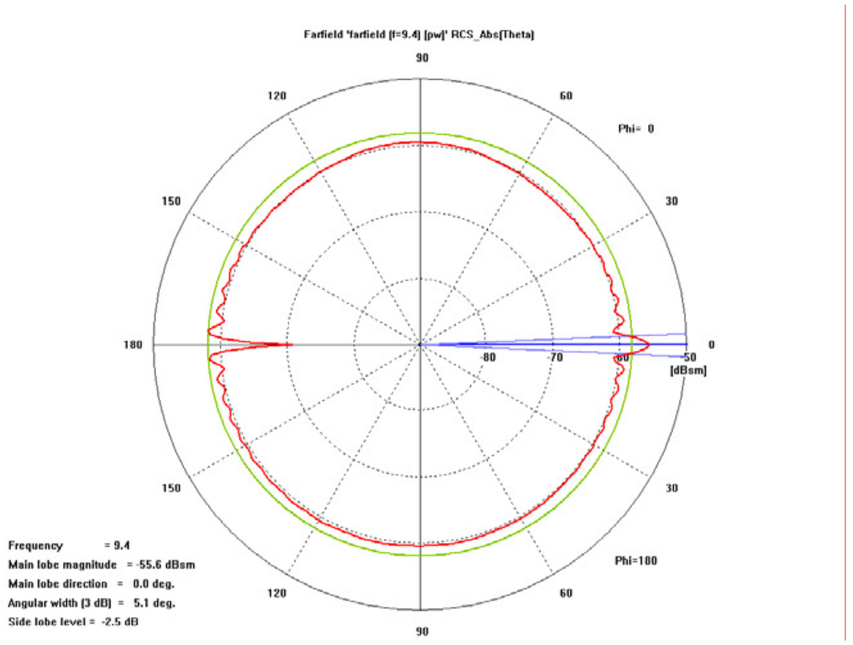
Surrounding space

Lower X distance: 0.0 Upper X distance: 31.9

Lower Y distance: 0.0 Upper Y distance: 0.0

Lower Z distance: 0.0 Upper Z distance: 0.0

Figure 8. Test of noise by adding extra volume when calculate RCS of one drop.



Frequency = 9.4
Main lobe magnitude = -55.6 dBsm
Main lobe direction = 0.0 deg.
Angular width (3 dB) = 5.1 deg.
Side lobe level = -2.5 dB

Figure 9. Plot of results adding extra volume when calculating RCS for one drop in the θ-plane for $\phi = 0$.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. SIMULATION SETUP FOR THE WEATHER RADAR SIMULATOR

Radar Data

```
P_t=10e3;                % Transmitted power
G_dB=20;                 % Maximum gain in dB
G=10^(G_dB/10);          % Maximum gain
angle_3db=0.20;          % 3 dB Beam angle [degrees], (half half power beamwidth)
pulsewidth=0.20;         % Pulsewidth [microsecond]
elevation=3;             % Elevation angle [degrees]
margine=5;               % margin for resolution cell[m]
azimuth=0;               % Azimuth angle [rad]
number_pulses=50;        % Number of pulses
PRT=1/300;               % Pulse Repetition Time [seconds]
distance=0.5;            % Range from the radar to the resolution cell [Km]
wavelength=0.1;          % Wavelength [m]
k=2*pi/wavelength;
```

Rain parameters

```
diam_low=0.05;           % Lowest diameter of Precipitation [mm]
diam_lim=1;              % Divider between low and mid [mm]
diam_mid=5;              % Divider between mid and high [mm]
diam_high=15;            % Highest diameter of Precipitation [mm]
resolution1=0.08;        % Lower diameter resolution [mm]
resolution2=0.3;         % Higher diameter resolution [mm]
resolution3=0.8;         % Highest diameter resolution [mm]
sort_approx=1;           % RCS approximation model; [1]=Rayleigh
kw=0.93;                 % abs(K)^2 approx for water
rain_rate=50;            % Rain Rate [mm/h]
rain_angle1=0;           % degrees Rain Angle (Phi)
rain_angle2=0;           % degrees Rain Angle (Theta)
wind_vector=[-5,0,0];    % Wind speed vector (x,y,z) [m/s]
spread=2;                % Wind speed spread [m/s], (all directions)
Norm=100;                % Normalization coefficient
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. CODE DEVELOPED IN MATLAB.

There are currently four versions of the simulator:

1) The basic version, WR_basic, that will run with one PRF. The plots include

- Doppler per pulse pair with average Doppler,
- Radial velocity histogram with Gaussian fit
- Return power with average Return power
- Reflectivity estimations with Ground Truth for comparison
- Voltage Correlation function with estimation of $\rho(\tau = T_s)$
- Power spectrum of return signal

2) Dual PRF version, WR_dual_prf, that will run with two PRFs. The plots include

- Doppler per pulse pair with average Doppler,
- Radial velocity histogram with Gaussian fit
- Return power with average Return power
- Reflectivity estimations with Ground Truth for comparison
- Phasor plots for the 30 first pulse pairs.

3) Frequency step version, WR_fq_step, that will step through several frequencies and transmit a pulse pair on every frequency. One PRF controls the time between pulses in the pulse pair and one PRF controls the time between equal frequencies. The plots include

- Doppler per pulse pair with average Doppler,

- Radial velocity histogram with Gaussian fit
- Return power with average Return power
- Reflectivity estimations with Ground Truth for comparison
- Phasor plots for the 30 first pulse pairs.

4) Pulse compression version, WR_pulse_com, that divide the resolution volume into range bins and averages over bins and pulses. The chirped pulse is approximated by discrete stepped frequencies. The plots include

- Return power with average Return power
- Reflectivity estimations with Ground Truth for comparison

1) WR_basic

```
% Weather Radar Signal Simulator, developed version
% Ulf Schroder
% July, 2005
%-----------------------------------------------------------------------

% Main program for The NPS/Marielle Gosset Weather Radar Signal Simulator.

%-----------------------------------------------------------------------
% DESCRIPTION
% Part I
% From Radar Data:
% - Generate Beam Resolution cell approximation
% - Generate Cube adding a margine to the Beam Resolution cell to allow for
% the rain drops to fall trough the through beam over time.
%
% Part II
% From Rain Parameters:
% Generating the fundamental T Matrix concisting of
% - Diameter Vector and Number of drops in each diameter interval
%   using the Marshall-Palmer exponential approximation.
% - the backscatter RCSs for each Diameter size
% - the total number of drops for each size per m^3

% Part III
% Randomly place all drops in cube

% Part IV
% For every pulse the coherent Electric field return is calculatet. Between
% pulses all drops are moved.

% Part V
% Calculate and plot results
%------------------------------------------------------------------------

clear all
close all
clc

% Input Data
c=3e8;                % Speed of light [m/s]

% Radar Data
P_t=10e3;             % Transmitted power
G_dB=20;              % Maximum gain in dB
G=10^(G_dB/10);       % Maximum gain
angle_3db=0.20;       % 3 dB Beam angle [degrees], (half half power beamwidth)
pulsewidth=0.20;      % Pulsewidth [microsecond]
elevation=3;          % Elevation angle [degrees]
margine=5;            % margin for resolution cell[m]
azimuth=0;            % Azimuth angle [rad]
number_pulses=40;     % Number of pulses
PRT=1/200;            % Pulse Repetition Time [seconds]
distance=0.5;         % Range from the radar to the resolution cell [Km]
wavelength=0.1;       % Wavelength [m]
k=2*pi/wavelength;

% Rain parameters
diam_low=0.05;        % Lowest diameter of Precipitation [mm]
diam_lim=1;           % Divider between low and mid [mm]
diam_mid=5;           % Divider between mid and high [mm]
diam_high=15;         % Highest diameter of Precipitation [mm]
resolution1=0.08;     % Lower diameter resolution [mm]
resolution2=0.3;      % Higher diameter resolution [mm]
resolution3=0.8;      % Highest diameter resolution [mm]
sort_approx=1;        % RCS approximation model; [1]=Rayleigh
kw=0.93;              % abs(K)^2 approx for water
rain_rate=50;         % Rain Rate [mm/h]
rain_anglel=0;        % degrees Rain Angle (Phi)
rain_angle2=0;        % degrees Rain Angle (Theta)
wind_vector=[-10;0;0];  % Wind speed vector (x,y,z) [m/s]
spread=1;             % Wind velocity spread [m/s], (all directions)
spread_2=4;           % Drop velocity spread [m/s],
Norm=100;             % Normalization coefficient

% Directional cosines
u=sin(pi/2-elevation*pi/180)*cos(azimuth*pi/180);
v=sin(pi/2-elevation*pi/180)*sin(azimuth*pi/180);
w=cos(pi/2-elevation*pi/180);

%------------------------------------------------------------------------
% Part I, Generate the resolution volume
%------------------------------------------------------------------------
% Approximation of the resolution cell of the pulse circular box,
% for any azimuth angle
%------------------------------------------------------------------------
% INPUT
% - distance
% - angle_3dB
% - pulsewidth
% - elevation
% - margine
% - azimuth
% -----------------------------------------------------------------------
% OUTPUT
% - volume_resolution: computed volume of the resolution cell
% - volume_box: volume of the box where the drops are put
% - box: the 8 rectangular coordinates of the box
% - coord_vol_res: the coordinates of the resolution cell where the drops
% are taken.
%------------------------------------------------------------------------

%------------------------------------------------------------------------
% Creating the circular beam resolution cell
range_res=(c*pulsewidth*1e-6)/2;                  ...
   % Range resolution (ct/2) [m]

% small angle approximation allows for following
angle_rad=(angle_3db*pi)/180;                     ...
   % 3 dB beamwidth [rad]
radius1=(tan(angle_rad)*(distance*1000));              ...
   % Radius of beam at R=distance
radius2=(tan(angle_rad)*((distance*1000)+range_res));   ...
   % Radius of beam at R=distance + range resolution

surface1=pi*radius1^2;
surface2=pi*radius2^2;

% volume of cone = base surface times height for all three
volume_cone1=(surface1*(distance*1000))/3;
volume_cone2=(surface2*((distance*1000)+range_res))/3;
volume_resolution=(volume_cone2-volume_cone1);         % Beam Cone Volume

% computation of the resolution cell coordinates, spherical coordinates
% elevation in radians
elevation_rad=(elevation*pi)/180;                  % Elevation [rad]
azimuth_rad=(azimuth*pi)/180;                      % Azimuth [rad]

% Matrix used when comparing wheather a drop is inside or outside
% beamwidth
coord_vol_res=[distance*1000,(distance*1000)+range_res;elevation_rad-angle_rad,...
      elevation_rad+angle_rad;azimuth_rad-angle_rad,azimuth_rad+angle_rad];
%------------------------------------------------------------------------

%------------------------------------------------------------------------
% Creating the box
% computation of the box and its coordinates
coin1=[azimuth_rad+angle_rad,elevation_rad+angle_rad,distance*1000];
coin2=[azimuth_rad-angle_rad,elevation_rad+angle_rad,distance*1000];
coin3=[azimuth_rad+angle_rad,elevation_rad-angle_rad,distance*1000];
coin4=[azimuth_rad-angle_rad,elevation_rad-angle_rad,distance*1000];
coin5=[azimuth_rad+angle_rad,elevation_rad+angle_rad,distance*1000+range_res];
coin6=[azimuth_rad-angle_rad,elevation_rad+angle_rad,distance*1000+range_res];
coin7=[azimuth_rad+angle_rad,elevation_rad-angle_rad,distance*1000+range_res];
```

```
coin8=[azimuth_rad-angle_rad,elevation_rad-angle_rad,distance*1000+range_res];

[x1,y1,z1]=sph2cart(coin1(1),coin1(2),coin1(3));
[x2,y2,z2]=sph2cart(coin2(1),coin2(2),coin2(3));
[x3,y3,z3]=sph2cart(coin3(1),coin3(2),coin3(3));
[x4,y4,z4]=sph2cart(coin4(1),coin4(2),coin4(3));
[x5,y5,z5]=sph2cart(coin5(1),coin5(2),coin5(3));
[x6,y6,z6]=sph2cart(coin6(1),coin6(2),coin6(3));
[x7,y7,z7]=sph2cart(coin7(1),coin7(2),coin7(3));
[x8,y8,z8]=sph2cart(coin8(1),coin8(2),coin8(3));

xmax=max([x1,x2,x3,x4,x5,x6,x7,x8]);
xmin=min([x1,x2,x3,x4,x5,x6,x7,x8]);
ymax=max([y1,y2,y3,y4,y5,y6,y7,y8]);
ymin=min([y1,y2,y3,y4,y5,y6,y7,y8]);
zmax=max([z1,z2,z3,z4,z5,z6,z7,z8]);
zmin=min([z1,z2,z3,z4,z5,z6,z7,z8]);

% the box with its margin
coin_1=[xmin-margine,ymin-margine,zmin-margine];
coin_2=[xmin-margine,ymax+margine,zmin-margine];
coin_3=[xmax+margine,ymax+margine,zmin-margine];
coin_4=[xmax+margine,ymin-margine,zmin-margine];
coin_5=[xmin-margine,ymin-margine,zmax+margine];
coin_6=[xmin-margine,ymax+margine,zmax+margine];
coin_7=[xmax+margine,ymax+margine,zmax+margine];
coin_8=[xmax+margine,ymin-margine,zmax+margine];


box=[coin_1',coin_2',coin_3',coin_4',coin_5',coin_6',coin_7',coin_8'];

% calculate the volume of the box
volume_box=abs((xmax+margine-(xmin-margine))*(ymax+8-(ymin-margine))*...
   (zmax+margine-(zmin-margine)));


cube=[coin_1',coin_2',coin_3',coin_4',coin_1',coin_5',coin_6',coin_2',...
       coin_6',coin_7',coin_3',coin_7',coin_8',coin_4',coin_8',coin_5'];
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% Part II, Generate the fundamental T Matrix
%-----------------------------------------------------------------------
% This part produces a matrix of all drops that contribute to the total
% reflectivity of the resolution cell
%-----------------------------------------------------------------------
% First all sizes are created, with two vectors of different size resolution.
% The size vectors represents all diameters. Next the backscatter RCS for each
% size is computed.
% Using the Marshall-Palmer exponential approximation, the number of drops of
% each diameter is computed.
%-----------------------------------------------------------------------
% INPUT
% - diam_low
% - diam_lim
% - diam_high
% - resolution1
% - resolution2
% - sort_approx
% - wavelength
% - Kw
% - rain_rate
%------------------------------------------------------------------------
% OUTPUT
% - number_drops
% Matrix T:
% - first row all sizes
% - second row all the backscatter RCSs
% - third row the total number of drops for each size per m^3
% the columns starting from the smallest size
%------------------------------------------------------------------------
% Create dropsize vector

% Precipitaion Diameter vector (lowest<D<divider)
vector_size_1=diam_low:resolution1:diam_lim;

% Precipitaion Diameter vector (divider<D<mid)
vector_size_2=diam_lim:resolution2:diam_mid;

% Precipitaion Diameter vector (mid<D<highest)
vector_size_3=diam_mid:resolution3:diam_high;

% Precipitaion Diameter Vector
vector_size=[vector_size_1,vector_size_2,vector_size_3];

%------------------------------------------------------------------------
% Create RCS vector for all Rain drops of Precipitation Vector.
rcs=((kw^2)*(pi^5)/(wavelength^4))*((vector_size*1e-3).^6);
%------------------------------------------------------------------------

%------------------------------------------------------------------------
% The Marshall_Palmer approximation is used to create Precipitaion Diameter
% Distribution Vector (Drop Size Distribution N(D) Vector)
% The number of drops are scaled down by a factor (Norm) to improve the
% speed of the simulator
%------------------------------------------------------------------------
lambda=4.1*(rain_rate^(-0.21));        %[mm-1]
number=8000*exp(-lambda*vector_size)/Norm;

vector_res1=ones(1,length(vector_size_1))*resolution1;
vector_res2=ones(1,length(vector_size_2))*resolution2;
vector_res3=ones(1,length(vector_size_3))*resolution3;

% delta(Diameter) dD
vector_res=[vector_res1,vector_res2,vector_res3];

% the approximate "true" number of drops N(D)dD vector
number=number.*vector_res;
%------------------------------------------------------------------------

%------------------------------------------------------------------------
% create matrix T
```

```
T=[vector_size;rcs;number];

%-----------------------------------------------------------------------
% Change matrix 'T' to matrix 'matrix_T_Round'
%-----------------------------------------------------------------------
% This part computes the total number of drops by utilizing the resolution
% cell and the matrix_T_Round
% The only difference between the matrix_T_Round and T, is the fact that
% matrix_T_Round has the total number of drops on the third row and they
% are rounded of to nearest integer value.
%-----------------------------------------------------------------------

matrix_T_Round=T;
matrix_T_Round(3,:)=round(matrix_T_Round(3,:)*volume_box);

% Keep track of errors due to round off
diff=T(3,:)*volume_box-matrix_T_Round(3,:);
error_rcs=sum(diff.*matrix_T_Round(2,:))*Norm/volume_box*volume_resolution;
error_Z=error_rcs/volume_resolution*wavelength^4/pi^5/abs(kw)^2/1e-18;

%-----------------------------------------------------------------------
% Part III, Initial positioning of the drops within the volume
%-----------------------------------------------------------------------
% This part generates the initial position of the drops in the volume of
% the box.
% The positions are recorded in the file "sizexx.mat' taking into account
% the azimuth angle
%-----------------------------------------------------------------------
% INPUT
% - box
% - matrix_T_Round
% - elevation
% - azimuth
%-----------------------------------------------------------------------
% OUTPUT
% All files including the sizes.  Each contains the positions of all drops
% for a given diameter.  The data recoreded as follows:
% - radial distance r
% - elevation phi
% - azimuth theta
% - angle with respect to beam center.  This parameter will be
% used to identify drops which are in the beam.
%-----------------------------------------------------------------------

% Randomly put the calculated number of drops of each diameter in the box
number_size=length(matrix_T_Round(1,:));

for i=1:number_size

   % select number of drops of certain Diameter
           drops=matrix_T_Round(3,i);

   % create as many positions on the x axis as # drops per size
           elements_x=rand(1,drops);

   % put them in the range of interest
```

```
    % Random placement x for every drop
            position_x=box(1,1)+(box(1,3)-box(1,1))*elements_x;

    % create as many positionson the y axis as # drops per size
            elements_y=rand(1,drops);

    % put them in the range of interest
    % Random placement y for every drop
            position_y=box(2,1)+(box(2,2)-box(2,1))*elements_y;

    % create as many positions on the z axis as # drops per size
            elements_z=rand(1,drops);

    % put them in the range of interest
    % Random placement z for every drop
            position_z=box(3,1)+(box(3,5)-box(3,1))*elements_z;

    % matrix of results
    % Creating a drop Matrix using spherical coordinates
            [theta,phi,r]=cart2sph(position_x,position_y,position_z);

    % Calculating the range, phi, theta and angle to every drop with
    % reference to phi_0 and theta_0. Used to estimate weighted power return.
    vector_azimuth=ones(1,drops)*((azimuth*pi)/180);
    vector_elevation=ones(1,drops)*((elevation*pi)/180);
    [xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);
    range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);

    % approximation
    angle=atan(range./r);

    % Saving the drop positions in size%d.mat
    % matrixpos
    matrixpos=[r;phi;theta;angle];

    save(sprintf('size%d.mat',i),'matrixpos');

end
%--------------------------------------------------------------------------
%--------------------------------------------------------------------------
% Part IV, Calculate Coherent Electric field return.
%--------------------------------------------------------------------------
% For every pulse the coherent Electric field return is calculatet. Between
% pulses all drops are moved.
%--------------------------------------------------------------------------

% Run the simulation for specified number of pulses

% Zero all vectors and matrices that are going to be used
phase_vector=[];
sum_E=[];
weighted_E=[];

for p=1:number_pulses

    disp(sprintf('calculate the power-return of pulse %d\n',p));
```

```
% compute the (weighted) I and Q return
%-------------------------------------------------------------------
        % computes the total number of drops
        %-------------------------------------------------------------------
        % INPUT
        % - Matrix T
        % - wavelength
        % - angle_3dB
        % - distance
        % - coord_vol_res
        % - number_pulses
        %-------------------------------------------------------------------
        % OUTPUT
        % - total_E: the received complex Electric field)
        % - power: the power
        % - voltage: the voltage
        % - drops_beam: the number of drops in the beam
        %-------------------------------------------------------------------
        % Only the drops which are in the resolution cell are selected for the
        % computation
        %-------------------------------------------------------------------

number_size=length(matrix_T_Round(1,:));
total_E=0;
total_phase=0;
new_E=0;

for i=1:number_size

   % opening the file of registered data matrixpos.
   load(sprintf('size%d.mat',i));

   % removing drops that are outside the resolution cell
   number_drops_per_size=length(matrixpos(1,:));
   dropmatrix=[];

   % Filling the drop matrix with all drops inside the beam
   % resolution volume
   for d=1:number_drops_per_size
      if ((matrixpos(4,d)<=(angle_3db*pi/180))&(matrixpos(1,d)>=...
            coord_vol_res(1,1))&(matrixpos(1,d)<=coord_vol_res(1,2)));
         dropmatrix=[dropmatrix,matrixpos(:,d)];
      end
   end

   %-------------------------------------------------------------
   % Convert to cartesian coordinates
   if isempty(dropmatrix)==0
      [position_x,position_y,position_z]=sph2cart(dropmatrix(3,:),...
         dropmatrix(2,:),dropmatrix(1,:));

      % Create comparison vectors
      vector_azimuth=ones(1,length(dropmatrix(1,:)))*((azimuth*pi)/180);
      vector_elevation=ones(1,length(dropmatrix(1,:)))*((elevation*pi)/180);
```

```
        % Create a phase vector for the drops of current Diameter
        r_phase=position_x.*u+position_y.*v+position_z.*w;
        phase=exp((2*j*k).*r_phase);

        % Sum the contributions to the E-field
        total_phase=total_phase+sum(phase);
        total_E=total_E+sum(sqrt(matrix_T_Round(2,i)).*phase);

        %------------------------------------------------------------
        % ceate weighted power return, concidering range (r), theta, phi
        % with reference to boresight and radar parameters
        E_weighted=sqrt(P_t)*G*wavelength/(sqrt(4*pi)^3)./dropmatrix(1,:)...
           .^2.*(exp(-4*log(2).*((dropmatrix(3,:)-vector_azimuth).^2/...
           (2*angle_3db*pi/180)^2+((dropmatrix(2,:)-vector_elevation)...
           .^2/(2*angle_3db*pi/180)^2))));
        new_E=new_E+sum(E_weighted.*(sqrt(matrix_T_Round(2,i)).*phase));

    end
end
%-------------------------------------------------------------------

         %-------------------------------------------------------------------
         % This part simulates the movement of the drops and gives new positions.
         %-------------------------------------------------------------------
         % INPUT
         % - matrix_T_Round
         % - PRT
         % - azimuth
         % - elevation
         % - rain_angle1
         % - rain_angle2
         % - diam_lim
% - resolution1
% - resolution2
         % - wind_vector
         %-------------------------------------------------------------------
         % OUTPUTS
         % Output files containing sizes with new positions
         % structured the same way as in previous
         % there is also an output file for speed (one for each size) for the pulse
         % pair.
         %-------------------------------------------------------------------

number_size=length(matrix_T_Round(1,:));

% adding a Gaussian random term to make the wind speed spread
if wind_vector(1)~=0
   new_wind_vector(1)=wind_vector(1)*normrnd(1,abs(spread/max(wind_vector(1))));
else
   new_wind_vector(1)=0;
end
if wind_vector(2)~=0
   new_wind_vector(2)=wind_vector(2)*normrnd(1,abs(spread/max(wind_vector(2))));
else
   new_wind_vector(2)=0;
end
```

```
if wind_vector(3)~=0
   new_wind_vector(3)=wind_vector(3)*normrnd(1,abs(spread/max(wind_vector(3))));
else
   new_wind_vector(3)=0;
end

for i=1:number_size

   % opening of data file registered in matrixpos.
   load(sprintf('size%d.mat',i));

   % total number of drops
   drops=matrix_T_Round(3,i);

   % transforming to rectangular coordinates
   [x,y,z]=sph2cart(matrixpos(3,:),matrixpos(2,:),matrixpos(1,:));

   diameter=matrix_T_Round(1,i);

   %-----------------------------------------------------------------
   % Drop fall speed depends on drop diameter in the z axis,
   % Atlas-Ulbrich approximation is used
                  % Approximation is valid in the diameter range 5*e-4, 5*e-3
                  %-----------------------------------------------------------------

   wtmax=386.6*((matrix_T_Round(1,i)*0.001)^0.67);

   velocityz=(wtmax*(ones(1,drops)))*cos((rain_anglel*pi)/180);
   velocityx=(wtmax*(ones(1,drops)))*sin((rain_anglel*pi)/180)*...
      cos((rain_angle2*pi)/180);
   velocityy=(wtmax*(ones(1,drops)))*sin((rain_anglel*pi)/180)*...
      sin((rain_angle2*pi)/180);


   newest_wind_vector=[];


    % adding Gaussian random term to make the wind speed
    % spread between drops
   if new_wind_vector(1)~=0
      newest_wind_vector(1,:)=new_wind_vector(1)*...
         normrnd(1,abs(spread_2/max(new_wind_vector(1))),1,drops);
   else
      newest_wind_vector(1,:)=0;
   end
   if new_wind_vector(2)~=0
      newest_wind_vector(2,:)=new_wind_vector(2)*...
         normrnd(1,abs(spread_2/max(new_wind_vector(2))),1,drops);
   else
      newest_wind_vector(2,:)=0;
   end
   if new_wind_vector(3)~=0
      newest_wind_vector(3,:)=new_wind_vector(3)*...
         normrnd(1,abs(spread_2/max(new_wind_vector(3))),1,drops);
   else
      newest_wind_vector(3,:)=0;
```

```
        end

        % If spread within pulse
        velocityxt=newest_wind_vector(1,:)-velocityx;
        velocityyt=newest_wind_vector(2,:)-velocityy;
        velocityzt=newest_wind_vector(3,:)-velocityz;


        % speed registration
        m_velocity=[velocityxt;velocityyt;velocityzt];
        save(sprintf('velocity%d.mat',i),'m_velocity');

        deplacementx=velocityxt*PRT;
        deplacementy=velocityyt*PRT;
        deplacementz=velocityzt*PRT;

        position_x=x+deplacementx;
        position_y=y+deplacementy;
        position_z=z+deplacementz;

        % transformation into spherical coordinates
        [theta,phi,r]=cart2sph(position_x,position_y,position_z);

        % computation of new angles
        vector_azimuth=ones(1,drops)*(azimuth*pi)/180;
        vector_elevation=ones(1,drops)*(elevation*pi)/180;
        [xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);

        range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);
        % approximation
        angle=atan(range./r);

        % matrixpos
        matrixpos=[r;phi;theta;angle];

        % save to file, matrixpos
                    s=sprintf('size%d.mat',i);
            save(s, 'matrixpos');

    end
          %-----------------------------------------------------------------

    % add to E-field and phase vector
    sum_E=[sum_E,total_E];
    weighted_E=[weighted_E,new_E];
    phase_vector=[phase_vector,ANGLE(total_phase)];

end

%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% Power return for every pulse including error

running_Z=(Norm*abs(sum_E).^2)*wavelength^4/pi^5/abs(kw)^2/volume_resolution...
    /1e-18+error_Z;
```

```
power_return=Norm*abs(weighted_E).^2;                    % [W]

% calculate average Power and Z
Z_avg=sum(running_Z)/p;
Z_avg_plot=ones(1,p)*Z_avg;
power_return_avg=sum(Norm*abs(weighted_E).^2)/p;           % [W]
power_return_avg_plot=ones(1,p)*power_return_avg;

% calculate doppler frequency
phase_vector_u=unwrap(phase_vector);
doppler_fq=([phase_vector_u(2:p)-phase_vector_u(1:p-1)])/2/pi/PRT;

number_vector=1:p;

%-------------------------------------------------------------------------
% Calculate references
% Integral form of Z
ii=0:100/1000:100;
zz=ii.^6.*8000.*exp(-4.1.*rain_rate.^(-.21).*ii);
real_Z=ones(1,number_pulses)*10*log10(trapz(ii,zz));
% reflectivity_real=ones(1,number_pulses)*10*log10(pi^5/wavelength^4*...
%     volume_resolution*abs(kw)^2*real_Z*1e-18);

% Adding the error to the weighted_E estimations
error_power=error_rcs*P_t*G^2*wavelength^2/pi^2*(2*angle_3db*pi/180)^2*...
   (c*pulsewidth*1e-6)/1024/log(2)/pi^2/(distance*1e3)^2/volume_resolution;

% Evaluate average Power return to make Z estimation including error
Z_estimation=power_return_avg/P_t/G^2/wavelength^2*...
   pi^2/(2*angle_3db*pi/180)^2/(c*pulsewidth*1e-6)*1024*log(2)*(distance*1e3)^2/...
   ((kw^2)*(pi^5)/(wavelength^4))/1e-18;
Z_estimation_plot=ones(1,number_pulses)*10*log10(Z_estimation+error_Z);

% calculate the FFT of the weighted electric field return (1024 points) to
% get the power spectal estimate
NN=1024;            % number of FFT points
spectral_estimate=abs(fft(weighted_E-mean(weighted_E),NN)).^2;
spectral_estimate_shifted=fftshift(spectral_estimate);
%-------------------------------------------------------------------------
% Part V, Plot of results
%-------------------------------------------------------------------------

figure(1);
subplot(2,1,1);
plot(number_vector(1:p-1),doppler_fq(1:p-1),'-x',number_vector(1:p),...
   ones(1,p)*mean(doppler_fq),'--r');
title(['Doppler frequency for all pulse pairs for PRF= ',num2str(1/PRT),' Hz']);
xlabel('Pulse pair');
ylabel('f_d');
legend('Doppler frequency per pulse pair','Average Doppler estimation')
grid on


subplot(2,1,2);
v_r=doppler_fq*wavelength/2;
```

```
hist(v_r,15)
hold on

[ff,xx] = ksdensity(v_r);
plot(xx,ff/max(ff)*max(hist(v_r,15)))
title('Radial Velocity Histogram with Gaussian Fit');
xlabel('Radial velocity [m/s]');
ylabel('Frequency');
grid on

figure (2)
subplot(2,1,1)
% also converting to dBm
plot(number_vector,10*log10(power_return+error_power)+30,'-x',...
   number_vector,10*log10(power_return_avg_plot+error_power)+30,'--r');
title('Power Return for all pulses');
xlabel('Sample time');
ylabel('Power [dBm]');
legend('Power return', 'Average Power return',0)
grid on

subplot(2,1,2);
plot(number_vector*1000*PRT,10*log10(abs(running_Z)),'-xb',number_vector...
   *1000*PRT,10*log10(abs(Z_avg_plot)),'-.r',number_vector*1000*...
   PRT,real_Z,'-k',number_vector*1000*PRT,Z_estimation_plot,':b');
title('Estimated Z per pulse');
xlabel('Sample time');
ylabel('Z [dBZ]');
legend('Simulation Z, actual', 'Average Z','Z=\int D^6N(D)dD',...
   'Estimated Z, Zeroth moment',0)
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',2)
grid on


figure (3)
subplot(2,1,1)
f=abs(xcorr((weighted_E),'coeff'));
tau_ts=ones(1,length(f))*f((length(f)+1)/2-1);
plot(1:length(f),abs(xcorr((weighted_E),'coeff')),'b',1:length(f),tau_ts,'--r');
title('Return Voltage Correlation');
xlabel('Number of pulses');
ylabel('Normalized correlation value');
legend(['Correlation function'],['\rho(\tau=T_s)=',num2str(tau_ts(1))])
grid on

subplot(2,1,2)
% Prepare for ticks so every 100 Hz is marked
n=0:1/PRT/100;
N_i=100*(NN-1)*n*PRT;
alfa=zeros(1,1/PRT/100+1);
vic=(length(alfa)-1)/2;
for beta=1:length(alfa)
   alfa(1,beta)=-vic*100;
   vic=vic-1;
end
```

```
avg_fq=mean(doppler_fq)
std_fq=std(doppler_fq)
gauss_vector=min(alfa):10:max(alfa);
uu=length(gauss_vector);
[value1,index1]=max(abs(spectral_estimate_shifted));
avg_fq_fft=(index1-NN/2)/PRT/NN;

plot_gauss_vector=linspace(0,NN,uu);
plot_norm_fq=normpdf(gauss_vector,avg_fq,std_fq/2);
plot_norm_fft=normpdf(gauss_vector,avg_fq_fft,std_fq/2);

plot(spectral_estimate_shifted/max(spectral_estimate_shifted))
hold on
plot(fftshift(abs(fft(xcorr((weighted_E),'coeff'),1024))/...
   max(abs(fft(xcorr((weighted_E),'coeff'),1024)))),'-k')
title('Power Spectral Estimate');
xlabel('Doppler Frequancy [Hz]');
ylabel('S|f|, normalized');
xlim([0 NN]);
set(gca,'xtick', N_i)
set(gca,'xticklabel', num2str(alfa'));
grid on
```

2) WR_dual_prf

```
% Weather Radar Signal Simulator, developed version
% Two PRFs enabling pulse pair comparison
% Ulf Schroder
% August, 2005
%------------------------------------------------------------------------

% Main program for The NPS/Marielle Gosset Weather Radar Signal Simulator.

%------------------------------------------------------------------------
% DESCRIPTION
% Part I
% From Radar Data:
% - Generate Beam Resolution cell approximation
% - Generate Cube adding a margine to the Beam Resolution cell to allow for
% the rain drops to fall trough the through beam over time.
%
% Part II
% From Rain Parameters:
% Generating the fundamental T Matrix concisting of
% - Diameter Vector and Number of drops in each diameter interval
%   using the Marshall-Palmer exponential approximation.
% - the backscatter RCSs for each Diameter size
% - the total number of drops for each size per m^3

% Part III
% Randomly place all drops in cube

% Part IV
% For every pulse the coherent Electric field return is calculated. Between
% pulses all drops are moved.

% Part V
% Calculate and plot results
%------------------------------------------------------------------------

clear all
close all
clc

% Input Data
c=3e8;                % Speed of light [m/s]

% Radar Data
P_t=10e3;              % Transmitted power
G_dB=20;               % Maximum gain in dB
G=10^(G_dB/10);        % Maximum gain
angle_3db=0.20;        % 3 dB Beam angle [degrees], (half half power beamwidth)
pulsewidth=0.20;       % Pulsewidth [microsecond]
elevation=3;           % Elevation angle [degrees]
margine=5;             % margin for resolution cell[m]
azimuth=0;             % Azimuth angle [rad]
number_pulses=40;      % Number of pulses
PRT_1=1/2000;          % Doppler Pulse Repetition Time [seconds]
PRT_2=1/200;           % Avg Power Pulse Repetition Time [seconds]
distance=0.5;          % Range from the radar to the resolution cell [Km]
wavelength=0.1;        % Wavelength [m]
k=2*pi/wavelength;

% Rain parameters
diam_low=0.05;         % Lowest diameter of Precipitation [mm]
diam_lim=1;            % Divider between low and mid [mm]
diam_mid=5;            % Divider between mid and high [mm]
diam_high=15;          % Highest diameter of Precipitation [mm]
resolution1=0.08;      % Lower diameter resolution [mm]
resolution2=0.3;       % Higher diameter resolution [mm]
resolution3=0.8;       % Highest diameter resolution [mm]
sort_approx=1;         % RCS approximation model; [1]=Rayleigh
kw=0.93;               % abs(K)^2 approx for water
rain_rate=50;          % Rain Rate [mm/h]
rain_anglel=0;         % degrees Rain Angle (Phi)
rain_angle2=30;        % degrees Rain Angle (Theta)
wind_vector=[-10;0;0];   % Wind speed vector (x,y,z) [m/s]
spread=1;              % Wind speed spread [m/s], (all directions)
spread_2=4;            % Drop velocity spread [m/s],
Norm=100;              % Normalization coefficient

% Directional cosines
u=sin(pi/2-elevation*pi/180)*cos(azimuth*pi/180);
v=sin(pi/2-elevation*pi/180)*sin(azimuth*pi/180);
w=cos(pi/2-elevation*pi/180);

%------------------------------------------------------------------------
% Part I, Generate the resolution volume
%------------------------------------------------------------------------
% Approximation of the resolution cell of the pulse circular box,
% for any azimuth angle
%-----------------------------------------------------------------------
% INPUT
% - distance
% - angle_3dB
% - pulsewidth
% - elevation
% - margine
% - azimuth
% -----------------------------------------------------------------------
% OUTPUT
% - volume_resolution: computed volume of the resolution cell
% - volume_box: volume of the box where the drops are put
% - box: the 8 rectangular coordinates of the box
% - coord_vol_res: the coordinates of the resolution cell where the drops
% are taken.
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% Creating the circular beam resolution cell
range_res=(c*pulsewidth*1e-6)/2;                    ...
   % Range resolution (ct/2) [m]

% small angle approximation allows for following
angle_rad=(angle_3db*pi)/180;                       ...
   % 3 dB beamwidth [rad]
radius1=(tan(angle_rad)*(distance*1000));           ...
   % Radius of beam at R=distance
radius2=(tan(angle_rad)*((distance*1000)+range_res));   ...
   % Radius of beam at R=distance + range resolution

surface1=pi*radius1^2;
surface2=pi*radius2^2;

% volume of cone = base surface times height for all three
volume_cone1=(surface1*(distance*1000))/3;
volume_cone2=(surface2*((distance*1000)+range_res))/3;
volume_resolution=(volume_cone2-volume_cone1);         % Beam Cone Volume

% computation of the resolution cell coordinates, spherical coordinates
% elevation in radians
elevation_rad=(elevation*pi)/180;                  % Elevation [rad]
azimuth_rad=(azimuth*pi)/180;                      % Azimuth [rad]

% Matrix used when comparing wheather a drop is inside or outside
% beamwidth
coord_vol_res=[distance*1000,(distance*1000)+range_res;elevation_rad-angle_rad,...
      elevation_rad+angle_rad;azimuth_rad-angle_rad,azimuth_rad+angle_rad];
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% Creating the box
% computation of the box and its coordinates
coin1=[azimuth_rad+angle_rad,elevation_rad+angle_rad,distance*1000];
coin2=[azimuth_rad-angle_rad,elevation_rad+angle_rad,distance*1000];
coin3=[azimuth_rad+angle_rad,elevation_rad-angle_rad,distance*1000];
```

```
coin4=[azimuth_rad-angle_rad,elevation_rad-angle_rad,distance*1000];
coin5=[azimuth_rad+angle_rad,elevation_rad+angle_rad,distance*1000+range_res];
coin6=[azimuth_rad-angle_rad,elevation_rad+angle_rad,distance*1000+range_res];
coin7=[azimuth_rad+angle_rad,elevation_rad-angle_rad,distance*1000+range_res];
coin8=[azimuth_rad-angle_rad,elevation_rad-angle_rad,distance*1000+range_res];

[x1,y1,z1]=sph2cart(coin1(1),coin1(2),coin1(3));
[x2,y2,z2]=sph2cart(coin2(1),coin2(2),coin2(3));
[x3,y3,z3]=sph2cart(coin3(1),coin3(2),coin3(3));
[x4,y4,z4]=sph2cart(coin4(1),coin4(2),coin4(3));
[x5,y5,z5]=sph2cart(coin5(1),coin5(2),coin5(3));
[x6,y6,z6]=sph2cart(coin6(1),coin6(2),coin6(3));
[x7,y7,z7]=sph2cart(coin7(1),coin7(2),coin7(3));
[x8,y8,z8]=sph2cart(coin8(1),coin8(2),coin8(3));

xmax=max([x1,x2,x3,x4,x5,x6,x7,x8]);
xmin=min([x1,x2,x3,x4,x5,x6,x7,x8]);
ymax=max([y1,y2,y3,y4,y5,y6,y7,y8]);
ymin=min([y1,y2,y3,y4,y5,y6,y7,y8]);
zmax=max([z1,z2,z3,z4,z5,z6,z7,z8]);
zmin=min([z1,z2,z3,z4,z5,z6,z7,z8]);

% the box with its margin
coin_1=[xmin-margine,ymin-margine,zmin-margine];
coin_2=[xmin-margine,ymax+margine,zmin-margine];
coin_3=[xmax+margine,ymax+margine,zmin-margine];
coin_4=[xmax+margine,ymin-margine,zmin-margine];
coin_5=[xmin-margine,ymin-margine,zmax+margine];
coin_6=[xmin-margine,ymax+margine,zmax+margine];
coin_7=[xmax+margine,ymax+margine,zmax+margine];
coin_8=[xmax+margine,ymin-margine,zmax+margine];


box=[coin_1',coin_2',coin_3',coin_4',coin_5',coin_6',coin_7',coin_8'];

% calculate the volume of the box
volume_box=abs((xmax+margine-(xmin-margine))*(ymax+margine-(ymin-margine))*(...
    zmax+margine-(zmin-margine)));


cube=[coin_1',coin_2',coin_3',coin_4',coin_1',coin_5',coin_6',coin_2',...
       coin_6',coin_7',coin_3',coin_7',coin_8',coin_4',coin_8',coin_5'];
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% Part II, Generate the fundamental T Matrix
%-----------------------------------------------------------------------
% This part produces a matrix of all drops that contribute to the total
% reflectivity of the resolution cell
%-----------------------------------------------------------------------
% First all sizes are created, with two vectors of different size resolution.
% The size vectors represents all diameters. Next the backscatter RCS for each
% size is computed.
% Using the Marshall-Palmer exponential approximation, the number of drops of
% each diameter is computed.
%-----------------------------------------------------------------------
```

```
% INPUT
% - diam_low
% - diam_lim
% - diam_high
% - resolution1
% - resolution2
% - sort_approx
% - wavelength
% - Kw
% - rain_rate
%-----------------------------------------------------------------------
% OUTPUT
% - number_drops
% Matrix T:
% - first row all sizes
% - second row all the backscatter RCSs
% - third row the total number of drops for each size per m^3
% the columns start with the smallest size
%-----------------------------------------------------------------------
% Create dropsize vector

% Precipitaion Diameter vector (lowest<D<divider)
vector_size_1=diam_low:resolution1:diam_lim;

% Precipitaion Diameter vector (divider<D<mid)
vector_size_2=diam_lim:resolution2:diam_mid;

% Precipitaion Diameter vector (mid<D<highest)
vector_size_3=diam_mid:resolution3:diam_high;

% Precipitaion Diameter Vector
vector_size=[vector_size_1,vector_size_2,vector_size_3];

%-----------------------------------------------------------------------
% Create RCS vector for all Rain drops of Precipitation Vector.
rcs=((kw^2)*(pi^5)/(wavelength^4))*((vector_size*1e-3).^6);
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% The Marshall_Palmer approximation is used to create Precipitaion Diameter
% Distribution Vector (Drop Size Distribution N(D) Vector)
% The number of drops are scaled down by a factor (Norm) to improve the
% speed of the simulator
%-----------------------------------------------------------------------
lambda=4.1*(rain_rate^(-0.21));        %[mm-1]
number=8000*exp(-lambda*vector_size)/Norm;

vector_res1=ones(1,length(vector_size_1))*resolution1;
vector_res2=ones(1,length(vector_size_2))*resolution2;
vector_res3=ones(1,length(vector_size_3))*resolution3;

% delta(Diameter) dD
vector_res=[vector_res1,vector_res2,vector_res3];

% the approximate "true" number of drops N(D)dD vector
number=number.*vector_res;
```

```
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% create matrix T
T=[vector_size;rcs;number];

%-----------------------------------------------------------------------
% Change matrix 'T' to matrix 'matrix_T_Round'
%-----------------------------------------------------------------------
% This part computes the total number of drops by utilizing the resolution
% cell and the matrix_T_Round
% The only difference between the matrix_T_Round and T, is the fact that
% matrix_T_Round has the total number of drops on the third row and they
% are rounded off to nearest integer value.
%-----------------------------------------------------------------------

matrix_T_Round=T;
matrix_T_Round(3,:)=round(matrix_T_Round(3,:)*volume_box);

% Keep track of errors due to round off
diff=T(3,:)*volume_box-matrix_T_Round(3,:);
error_rcs=sum(diff.*matrix_T_Round(2,:))*Norm/volume_box*volume_resolution;
error_Z=error_rcs/volume_resolution*wavelength^4/pi^5/abs(kw)^2/1e-18;

%-----------------------------------------------------------------------
% Part III, Initial positioning of the drops within the volume
%-----------------------------------------------------------------------
% This part generates the initial position of the drops in the volume of
% the box.
% The positions are recorded in the file "sizexx.mat' taking into account
% the azimuth angle
%-----------------------------------------------------------------------
% INPUT
% - box
% - matrix_T_Round
% - elevation
% - azimuth
%-----------------------------------------------------------------------
% OUTPUT
% All files including the sizes.  Each contains the positions of all drops
% for a given diameter.  The data recoreded as follows:
% - radial distance r
% - elevation phi
% - azimuth theta
% - angle with respect to beam center.  This parameter will be
% used to identify drops which are in the beam.
%-----------------------------------------------------------------------

% Randomly put the calculated number of drops of each diameter in the box
number_size=length(matrix_T_Round(1,:));

for i=1:number_size

  % select number of drops of certain Diameter
        drops=matrix_T_Round(3,i);
```

```
    % create as many positions on the x axis as # drops per size
            elements_x=rand(1,drops);

    % put them in the range of interest
    % Random placement x for every drop
            position_x=box(1,1)+(box(1,3)-box(1,1))*elements_x;

    % create as many positionson the y axis as # drops per size
            elements_y=rand(1,drops);

    % put them in the range of interest
    % Random placement y for every drop
            position_y=box(2,1)+(box(2,2)-box(2,1))*elements_y;

    % create as many positions on the z axis as # drops per size
            elements_z=rand(1,drops);

    % put them in the range of interest
    % Random placement z for every drop
            position_z=box(3,1)+(box(3,5)-box(3,1))*elements_z;

    % matrix of results
    % Creating a drop Matrix using spherical coordinates
            [theta,phi,r]=cart2sph(position_x,position_y,position_z);

    % Calculating the range, phi, theta and angle to every drop with
    % reference to phi_0 and theta_0. Used to estimate weighted power return.
    vector_azimuth=ones(1,drops)*((azimuth*pi)/180);
    vector_elevation=ones(1,drops)*((elevation*pi)/180);
    [xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);
    range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);

    % approximation
    angle=atan(range./r);

    % Saving the drop positions in size%d.mat
    % matrixpos
    matrixpos=[r;phi;theta;angle];

    save(sprintf('size%d.mat',i),'matrixpos');

end
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% Part IV, Calculate Coherent Electric field return.
%-------------------------------------------------------------------------
% For every pulse the coherent Electric field return is calculatet. Between
% pulses all drops are moved.
%-------------------------------------------------------------------------

% Run the simulation for specified number of pulses

% Zero all vectors and matrices that are going to be used
powerT=0;
vector_power=[];
```

```
phase_vector=zeros(1,2*number_pulses);
imaginary_return=zeros(1,2*number_pulses);
sum_E=[];
weighted_E=[];

for p=1:number_pulses

  disp(sprintf('calculate the power-return of pulse %d\n',p));

  % compute the (weighted) I and Q return
  %--------------------------------------------------------------------
       % computes the total number of drops
       %--------------------------------------------------------------------
       % INPUT
       % - Matrix T
       % - wavelength
       % - angle_3dB
       % - distance
       % - coord_vol_res
       % - number_pulses
       %--------------------------------------------------------------------
       % OUTPUT
       % - total_E: the received complex Electric field)
       % - power: the power
       % - voltage: the voltage
       % - drops_beam: the number of drops in the beam
       %--------------------------------------------------------------------
       % Only the drops which are in the resolution cell are selected for the
       % computation
       %--------------------------------------------------------------------

  number_size=length(matrix_T_Round(1,:));
  total_E=0;
  total_phase=0;
  new_E=0;

  for i=1:number_size

    % opening the file of registered data matrixpos.
    load(sprintf('size%d.mat',i));

    % removing drops that are outside the resolution cell
    number_drops_per_size=length(matrixpos(1,:));
    dropmatrix=[];

    % Filling the drop matrix with all drops inside the beam
    % resolution volume
    for d=1:number_drops_per_size
      if ((matrixpos(4,d)<=(angle_3db*pi/180))&(matrixpos(1,d)>=...
            coord_vol_res(1,1))&(matrixpos(1,d)<=coord_vol_res(1,2)));
        dropmatrix=[dropmatrix,matrixpos(:,d)];
      end
    end

    %---------------------------------------------------------------
    % Convert to cartesian coordinates
    if isempty(dropmatrix)==0
      [position_x,position_y,position_z]=sph2cart(dropmatrix(3,:),...
        dropmatrix(2,:),dropmatrix(1,:));

      % Create comparison vectors
      vector_azimuth=ones(1,length(dropmatrix(1,:)))*((azimuth*pi)/180);
      vector_elevation=ones(1,length(dropmatrix(1,:)))*((elevation*pi)/180);

      % Create a phase vector for the drops of current Diameter
      r_phase=position_x.*u+position_y.*v+position_z.*w;
      phase=exp((2*j*k).*r_phase);

      % Sum the contributions to the E-field
      total_phase=total_phase+sum(phase);
      total_E=total_E+sum(sqrt(matrix_T_Round(2,i)).*phase);

      %------------------------------------------------------------
      % ceate weighted power return, concidering range (r), theta, phi
      % with reference to boresight and radar parameters
      E_weighted=sqrt(P_t)*G*wavelength/(sqrt(4*pi)^3)./dropmatrix(1,:).^2.*...
        (exp(-4*log(2).*((dropmatrix(3,:)-vector_azimuth).^2/...
        (2*angle_3db*pi/180)^2+((dropmatrix(2,:)-vector_elevation).^2/...
        (2*angle_3db*pi/180)^2))));
      new_E=new_E+sum(E_weighted.*(sqrt(matrix_T_Round(2,i)).*phase));

    end
  end

  phase_vector(2*p-1)=ANGLE(total_phase);
  imaginary_return(2*p-1)=total_phase;

  %--------------------------------------------------------------------

       %--------------------------------------------------------------------
       % This part simulates the movement of the drops and gives new positions.
  % First using the Doppler PRT
       %--------------------------------------------------------------------
       % INPUT
       % - matrix_T_Round
       % - PRT
       % - azimuth
       % - elevation
       % - rain_angle1
       % - rain_angle2
       % - diam_lim
  % - resolution1
  % - resolution2
       % - wind_vector
       %--------------------------------------------------------------------
       % OUTPUTS
       % Output files containing sizes with new positions
       % structured the same way as in previous
       % there is also an output file for speed (one for each size) for the pulse
       % pair.
       %--------------------------------------------------------------------
```

```
number_size=length(matrix_T_Round(1,:));

   % adding a Gaussian random term to make the wind speed spread
if wind_vector(1)~=0
   new_wind_vector(1)=wind_vector(1)*normrnd(1,abs(spread/max(wind_vector(1))));
else
   new_wind_vector(1)=0;
end
if wind_vector(2)~=0
   new_wind_vector(2)=wind_vector(2)*normrnd(1,abs(spread/max(wind_vector(2))));
else
   new_wind_vector(2)=0;
end
if wind_vector(3)~=0
   new_wind_vector(3)=wind_vector(3)*normrnd(1,abs(spread/max(wind_vector(3))));
else
   new_wind_vector(3)=0;
end


for i=1:number_size

   % opening of data file registered in matrixpos.
   load(sprintf('size%d.mat',i));

   % total number of drops
   drops=matrix_T_Round(3,i);

   % transforming to rectangular coordinates
   [x,y,z]=sph2cart(matrixpos(3,:),matrixpos(2,:),matrixpos(1,:));

   diameter=matrix_T_Round(1,i);

   %-----------------------------------------------------------------
   % Drop fall speed depends on drop diameter in the z axis,
   % Atlas-Ulbrich approximation is used
                  % Approximation is valid in the diameter range 5*e-4, 5*e-3
                  %-----------------------------------------------------------------

   wtmax=386.6*((matrix_T_Round(1,i)*0.001)^0.67);

   velocityz=(wtmax*(ones(1,drops)))*cos((rain_anglel*pi)/180);
   velocityx=(wtmax*(ones(1,drops)))*sin((rain_anglel*pi)/180)*...
      cos((rain_angle2*pi)/180);
   velocityy=(wtmax*(ones(1,drops)))*sin((rain_anglel*pi)/180)*...
      sin((rain_angle2*pi)/180);

   newest_wind_vector=[];


    % adding Gaussian random term to make the wind speed
    % spread between different drops sizes
   if new_wind_vector(1)~=0
      newest_wind_vector(1,:)=new_wind_vector(1)*normrnd(1,...
         abs(spread_2/max(new_wind_vector(1))),1,drops);
   else
```

```
      newest_wind_vector(1,:)=0;
   end
   if new_wind_vector(2)~=0
      newest_wind_vector(2,:)=new_wind_vector(2)*normrnd(1,...
         abs(spread_2/max(new_wind_vector(2))),1,drops);
   else
      newest_wind_vector(2,:)=0;
   end
   if new_wind_vector(3)~=0
      newest_wind_vector(3,:)=new_wind_vector(3)*normrnd(1,...
         abs(spread_2/max(new_wind_vector(3))),1,drops);
   else
      newest_wind_vector(3,:)=0;
   end

   velocityxt=newest_wind_vector(1,:)-velocityx;
   velocityyt=newest_wind_vector(2,:)-velocityy;
   velocityzt=newest_wind_vector(3,:)-velocityz;

   % speed registration
   m_velocity=[velocityxt;velocityyt;velocityzt];
   save(sprintf('velocity%d.mat',i),'m_velocity');

   deplacementx=velocityxt*PRT_1;
   deplacementy=velocityyt*PRT_1;
   deplacementz=velocityzt*PRT_1;

   position_x=x+deplacementx;
   position_y=y+deplacementy;
   position_z=z+deplacementz;

   % transformation into spherical coordinates
   [theta,phi,r]=cart2sph(position_x,position_y,position_z);

   % computation of new angles
   vector_azimuth=ones(1,drops)*(azimuth*pi)/180;
   vector_elevation=ones(1,drops)*(elevation*pi)/180;
   [xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);

   range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);
   % approximation
   angle=atan(range./r);

   % matrixpos
   matrixpos=[r;phi;theta;angle];

   % save to file, matrixpos
                  s=sprintf('size%d.mat',i);
          save(s, 'matrixpos');

end

        %------------------------------------------------------------------
% Calculate phase for pulse pair
total_phase=0;
```

```
for i=1:number_size

   % opening the file of registered data matrixpos.
   load(sprintf('size%d.mat',i));

   % removing drops that are outside the resolution cell
   number_drops_per_size=length(matrixpos(1,:));
   dropmatrix=[];

   % Filling the drop matrix with all drops inside the beam
   % resolution volume
   for d=1:number_drops_per_size
      if ((matrixpos(4,d)<=(angle_3db*pi/180))&(matrixpos(1,d)>=...
            coord_vol_res(1,1))&(matrixpos(1,d)<=coord_vol_res(1,2)));
         dropmatrix=[dropmatrix,matrixpos(:,d)];
      end
   end

   %------------------------------------------------------------------
   % Convert to cartesian coordinates
   if isempty(dropmatrix)==0
      [position_x,position_y,position_z]=sph2cart(dropmatrix(3,:),...
         dropmatrix(2,:),dropmatrix(1,:));

      % Create comparison vectors
      vector_azimuth=ones(1,length(dropmatrix(1,:)))*((azimuth*pi)/180);
      vector_elevation=ones(1,length(dropmatrix(1,:)))*((elevation*pi)/180);


      % Create a phase vector for the drops of current Diameter
      r_phase=position_x.*u+position_y.*v+position_z.*w;
      phase=exp((2*j*k).*r_phase);

      % Sum the contributions to the E-field
      total_phase=total_phase+sum(phase);
   end
end

phase_vector(2*p)=ANGLE(total_phase);
imaginary_return(2*p)=total_phase;


%---------------------------------------------------------------------
        % This part simulates the movement of the drops and gives new positions
% using the Avg Power PRT
        %---------------------------------------------------------------------

number_size=length(matrix_T_Round(1,:));


% adding a Gaussian random term to make the wind speed spread
if wind_vector(1)~=0
   new_wind_vector(1)=wind_vector(1)*normrnd(1,abs(spread/max(wind_vector(1))));
else
   new_wind_vector(1)=0;
end
```

```
if wind_vector(2)~=0
   new_wind_vector(2)=wind_vector(2)*normrnd(1,abs(spread/max(wind_vector(2))));
else
   new_wind_vector(2)=0;
end
if wind_vector(3)~=0
   new_wind_vector(3)=wind_vector(3)*normrnd(1,abs(spread/max(wind_vector(3))));
else
   new_wind_vector(3)=0;
end



for i=1:number_size

   % opening of data file registered in matrixpos.
   load(sprintf('size%d.mat',i));

   % total number of drops
   drops=matrix_T_Round(3,i);

   % transforming to rectangular coordinates
   [x,y,z]=sph2cart(matrixpos(3,:),matrixpos(2,:),matrixpos(1,:));

   diameter=matrix_T_Round(1,i);

   %------------------------------------------------------------------
   % Drop fall speed depends on drop diameter in the z axis,
   % Atlas-Ulbrich approximation is used
                        % Approximation is valid in the diameter range 5*e-4, 5*e-3
                    %------------------------------------------------------------------

   wtmax=386.6*((matrix_T_Round(1,i)*0.001)^0.67);

   velocityz=(wtmax*(ones(1,drops)))*cos((rain_angle1*pi)/180);
   velocityx=(wtmax*(ones(1,drops)))*sin((rain_angle1*pi)/180)*...
      cos((rain_angle2*pi)/180);
   velocityy=(wtmax*(ones(1,drops)))*sin((rain_angle1*pi)/180)*...
      sin((rain_angle2*pi)/180);

   newest_wind_vector=[];


     % adding Gaussian random term to make the wind speed
     % spread between different drops sizes
   if new_wind_vector(1)~=0
      newest_wind_vector(1,:)=new_wind_vector(1)*normrnd(1,...
         abs(spread_2/max(new_wind_vector(1))),1,drops);
   else
      newest_wind_vector(1,:)=0;
   end
   if new_wind_vector(2)~=0
      newest_wind_vector(2,:)=new_wind_vector(2)*normrnd(1,...
         abs(spread_2/max(new_wind_vector(2))),1,drops);
   else
      newest_wind_vector(2,:)=0;
```

```
        end
        if new_wind_vector(3)~=0
           newest_wind_vector(3,:)=new_wind_vector(3)*normrnd(1,...
              abs(spread_2/max(new_wind_vector(3))),1,drops);
        else
           newest_wind_vector(3,:)=0;
        end

        velocityxt=newest_wind_vector(1,:)+velocityx;
        velocityyt=newest_wind_vector(2,:)+velocityy;
        velocityzt=newest_wind_vector(3,:)+velocityz;

        % speed registration
        m_velocity=[velocityxt;velocityyt;velocityzt];
        save(sprintf('velocity%d.mat',i),'m_velocity');

        deplacementx=velocityxt*(PRT_2-PRT_1);
        deplacementy=velocityyt*(PRT_2-PRT_1);
        deplacementz=velocityzt*(PRT_2-PRT_1);

        position_x=x+deplacementx;
        position_y=y+deplacementy;
        position_z=z+deplacementz;

        % transformation into spherical coordinates
        [theta,phi,r]=cart2sph(position_x,position_y,position_z);

        % computation of new angles
        vector_azimuth=ones(1,drops)*(azimuth*pi)/180;
        vector_elevation=ones(1,drops)*(elevation*pi)/180;
        [xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);

        range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);
        % approximation
        angle=atan(range./r);

        % matrixpos
        matrixpos=[r;phi;theta;angle];

        % save to file, matrixpos
                    s=sprintf('size%d.mat',i);
             save(s, 'matrixpos');

     end


           %-------------------------------------------------------------------

     % add to E-field, drop and rain rate comparison
     sum_E=[sum_E,total_E];
     weighted_E=[weighted_E,new_E];

  end

  %--------------------------------------------------------------------
```

```
%-------------------------------------------------------------------------
% Calculate RCS and doppler

% calculate RCS, Z and Power return for every pulse including error
running_Z=(Norm*abs(sum_E).^2)*wavelength^4/pi^5/abs(kw)^2/volume_resolution/...
    1e-18+error_Z;
power_return=Norm*abs(weighted_E).^2;                    % [W]

% calculate average Power and Z
Z_avg=sum(running_Z)/p;
Z_avg_plot=ones(1,p)*Z_avg;
power_return_avg=sum(Norm*abs(weighted_E).^2)/p;          % [W]
power_return_avg_plot=ones(1,p)*power_return_avg;

% calculate doppler frequency
phase_vector_u=unwrap(phase_vector);
doppler_fq=([phase_vector_u(2:2:2*p)-phase_vector_u(1:2:2*p-1)])/2/pi/PRT_1;

number_vector=1:p;

%-------------------------------------------------------------------------
% Calculate references
% Integral form of Z
ii=0:100/1000:100;
zz=ii.^6.*8000.*exp(-4.1.*rain_rate.^(-.21).*ii);
real_Z=ones(1,number_pulses)*10*log10(trapz(ii,zz));

% Adding the error to the weighted_E estimations
error_power=error_rcs*P_t*G^2*wavelength^2/pi^2*(2*angle_3db*pi/180)^2*...
    (c*pulsewidth*1e-6)/1024/log(2)/pi^2/(distance*1e3)^2/volume_resolution;

% Evaluate average Power return to make Z estimation including error
Z_estimation=power_return_avg/P_t/G^2/wavelength^2*...
    pi^2/(2*angle_3db*pi/180)^2/(c*pulsewidth*1e-6)*1024*log(2)*(distance*1e3)^2/...
    ((kw^2)*(pi^5)/(wavelength^4))/1e-18;
Z_estimation_plot=ones(1,number_pulses)*10*log10(Z_estimation+error_Z);


%-------------------------------------------------------------------------
% Part V, Plot of results
%-------------------------------------------------------------------------

figure(1);
subplot(2,1,1);
plot(number_vector(1:p),doppler_fq(1:p),'-x',number_vector(1:p),ones(1,p)*...
    mean(doppler_fq),'--r');
title(['Doppler frequency for all pulse pairs for PRF= ',num2str(1/PRT_1),' Hz']);
xlabel('Pulse pair');
ylabel('f_d');
legend('Doppler frequency per pulse pair','Average Doppler estimation')
grid on

subplot(2,1,2)
hist(doppler_fq,15)
hold on
[ff,xx] = ksdensity(doppler_fq);
```

```
plot(xx,ff/max(ff)*max(hist(doppler_fq,15)))
title('Doppler Frequency Histogram with Gaussian Fit');
ylabel('Frequency');
xlabel('Doppler Frequency [Hz]');


figure (2)
subplot(2,1,1);
% also converting to dBm
plot(number_vector*PRT_2*1000,10*log10(power_return+error_power)+30,'-xb',...
   number_vector*PRT_2*1000,10*log10(power_return_avg_plot+error_power)+30,'--r');
title(['Power Return for all ',num2str(p),' pulses. PRF= ',num2str(1/PRT_2),' Hz']);
xlabel('Sample time [ms]');
ylabel('Power [dBm]');
legend('Power return', 'Average Power return',0)
grid on


subplot(2,1,2);
plot(number_vector,10*log10(abs(running_Z)),'-xb',number_vector,10*...
   log10(abs(Z_avg_plot)),'--r',number_vector,real_Z,'--k',...
   number_vector,Z_estimation_plot,'--b');
title('Estimated Z per pulse');
xlabel('Sample time');
ylabel('Z [dBZ]');
legend('Simulation Z, (instantaneous)', 'Estimated Z, (average return)',...
   'Z=\int D^6N(D)dD','Estimated Z, Zeroth moment',0)
grid on

figure(3)
polarmatrix(1:2:119)=zeros(1,60);
polarmatrix(2:2:120)=phase_vector_u(1:60);
lengthmatrix(1:2:119)=zeros(1,60);
lengthmatrix(2:2:120)=ones(1,60);
subplot(5,6,1)
polar(polarmatrix(1:2),lengthmatrix(1:2),'-*')
hold on
polar(polarmatrix(3:4),lengthmatrix(3:4),'--*r')
title(['1^{st}. v_r= ',num2str(doppler_fq(1)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,2)
polar(polarmatrix(5:6),lengthmatrix(5:6),'-*')
hold on
polar(polarmatrix(7:8),lengthmatrix(7:8),'--*r')
title(['2^{nd}. v_r= ',num2str(doppler_fq(2)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,3)
polar(polarmatrix(9:10),lengthmatrix(9:10),'-*')
hold on
polar(polarmatrix(11:12),lengthmatrix(11:12),':*r')
title(['3^{rd}. v_r= ',num2str(doppler_fq(3)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,4)
```

```
polar(polarmatrix(13:14),lengthmatrix(13:14),'-*')
hold on
polar(polarmatrix(15:16),lengthmatrix(15:16),':*r')
title(['4^{th}. v_r= ',num2str(doppler_fq(4)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,5)
polar(polarmatrix(17:18),lengthmatrix(17:18),'-*')
hold on
polar(polarmatrix(19:20),lengthmatrix(19:20),':*r')
title(['5^{th}. v_r= ',num2str(doppler_fq(5)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,6)
polar(polarmatrix(21:22),lengthmatrix(21:22),'-*')
hold on
polar(polarmatrix(23:24),lengthmatrix(23:24),':*r')
title(['6^{th}. v_r= ',num2str(doppler_fq(6)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,7)
polar(polarmatrix(25:26),lengthmatrix(25:26),'-*')
hold on
polar(polarmatrix(27:28),lengthmatrix(27:28),':*r')
title(['7^{th}. v_r= ',num2str(doppler_fq(7)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,8)
polar(polarmatrix(29:30),lengthmatrix(29:30),'-*')
hold on
polar(polarmatrix(31:32),lengthmatrix(31:32),':*r')
title(['8^{th}. v_r= ',num2str(doppler_fq(8)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,9)
polar(polarmatrix(33:34),lengthmatrix(33:34),'-*')
hold on
polar(polarmatrix(35:36),lengthmatrix(35:36),':*r')
title(['9^{th}. v_r= ',num2str(doppler_fq(9)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,10)
polar(polarmatrix(37:38),lengthmatrix(37:38),'-*')
hold on
polar(polarmatrix(39:40),lengthmatrix(39:40),':*r')
title(['10^{th}. v_r= ',num2str(doppler_fq(10)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,11)
polar(polarmatrix(41:42),lengthmatrix(41:42),'-*')
hold on
polar(polarmatrix(43:44),lengthmatrix(43:44),':*r')
title(['11^{th}. v_r= ',num2str(doppler_fq(11)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,12)
```

```
polar(polarmatrix(45:46),lengthmatrix(45:46),'-*')
hold on
polar(polarmatrix(47:48),lengthmatrix(47:48),':*r')
title(['12^{th}. v_r= ',num2str(doppler_fq(12)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,13)
polar(polarmatrix(49:50),lengthmatrix(49:50),'-*')
hold on
polar(polarmatrix(51:52),lengthmatrix(51:52),':*r')
title(['13^{th}. v_r= ',num2str(doppler_fq(13)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,14)
polar(polarmatrix(53:54),lengthmatrix(53:54),'-*')
hold on
polar(polarmatrix(55:56),lengthmatrix(55:56),':*r')
title(['14^{th}. v_r= ',num2str(doppler_fq(14)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,15)
polar(polarmatrix(57:58),lengthmatrix(57:58),'-*')
hold on
polar(polarmatrix(59:60),lengthmatrix(59:60),':*r')
title(['15^{th}. v_r= ',num2str(doppler_fq(15)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,16)
polar(polarmatrix(61:62),lengthmatrix(61:62),'-*')
hold on
polar(polarmatrix(63:64),lengthmatrix(63:64),':*r')
title(['16^{th}. v_r= ',num2str(doppler_fq(16)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,17)
polar(polarmatrix(65:66),lengthmatrix(65:66),'-*')
hold on
polar(polarmatrix(67:68),lengthmatrix(67:68),':*r')
title(['17^{th}. v_r= ',num2str(doppler_fq(17)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,18)
polar(polarmatrix(69:70),lengthmatrix(69:70),'-*')
hold on
polar(polarmatrix(71:72),lengthmatrix(71:72),':*r')
title(['18^{th}. v_r= ',num2str(doppler_fq(18)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,19)
polar(polarmatrix(73:74),lengthmatrix(73:74),'-*')
hold on
polar(polarmatrix(75:76),lengthmatrix(75:76),':*r')
title(['19^{th}. v_r= ',num2str(doppler_fq(19)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,20)
```

```
polar(polarmatrix(77:78),lengthmatrix(77:78),'-*')
hold on
polar(polarmatrix(79:80),lengthmatrix(79:80),':*r')
title(['20^{th}. v_r= ',num2str(doppler_fq(20)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,21)
polar(polarmatrix(81:82),lengthmatrix(81:82),'-*')
hold on
polar(polarmatrix(83:84),lengthmatrix(83:84),':*r')
title(['21^{th}. v_r= ',num2str(doppler_fq(21)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,22)
polar(polarmatrix(85:86),lengthmatrix(85:86),'-*')
hold on
polar(polarmatrix(87:88),lengthmatrix(87:88),':*r')
title(['22^{th}. v_r= ',num2str(doppler_fq(22)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,23)
polar(polarmatrix(89:90),lengthmatrix(89:90),'-*')
hold on
polar(polarmatrix(91:92),lengthmatrix(91:92),':*r')
title(['23^{th}. v_r= ',num2str(doppler_fq(23)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,24)
polar(polarmatrix(93:94),lengthmatrix(93:94),'-*')
hold on
polar(polarmatrix(95:96),lengthmatrix(95:96),':*r')
title(['24^{th}. v_r= ',num2str(doppler_fq(24)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,25)
polar(polarmatrix(97:98),lengthmatrix(97:98),'-*')
hold on
polar(polarmatrix(99:100),lengthmatrix(99:100),':*r')
title(['25^{th}. v_r= ',num2str(doppler_fq(25)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,26)
polar(polarmatrix(101:102),lengthmatrix(101:102),'-*')
hold on
polar(polarmatrix(103:104),lengthmatrix(103:104),':*r')
title(['26^{th}. v_r= ',num2str(doppler_fq(26)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,27)
polar(polarmatrix(105:106),lengthmatrix(105:106),'-*')
hold on
polar(polarmatrix(107:108),lengthmatrix(107:108),':*r')
title(['27^{th}. v_r= ',num2str(doppler_fq(27)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,28)
```

```
polar(polarmatrix(109:110),lengthmatrix(109:110),'-*')
hold on
polar(polarmatrix(111:112),lengthmatrix(111:112),':*r')
title(['28^{th}. v_r= ',num2str(doppler_fq(28)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,29)
polar(polarmatrix(113:114),lengthmatrix(113:114),'-*')
hold on
polar(polarmatrix(115:116),lengthmatrix(115:116),':*r')
title(['29^{th}. v_r= ',num2str(doppler_fq(29)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,30)
polar(polarmatrix(117:118),lengthmatrix(117:118),'-*')
hold on
polar(polarmatrix(119:120),lengthmatrix(119:120),':*r')
title(['30^{th}. v_r= ',num2str(doppler_fq(30)*wavelength/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)

mean(doppler_fq)
std(doppler_fq)
```

3) WR_fq_step

```
% Weather Radar Signal Simulator, developed version
% Frequency step
% Ulf Schroder
% August, 2005
%------------------------------------------------------------------------

% Main program for The NPS/Marielle Gosset Weather Radar Signal Simulator.

%------------------------------------------------------------------------
% DESCRIPTION
% Part I
% From Radar Data:
% - Generate Beam Resolution cell approximation
% - Generate Cube adding a margine to the Beam Resolution cell to allow for
% the rain drops to fall trough the through beam over time.
%
% Part II
% From Rain Parameters:
% Generating the fundamental T Matrix concisting of
% - Diameter Vector and Number of drops in each diameter interval
%   using the Marshall-Palmer exponential approximation.
% - the total number of drops for each size per m^3
% The backscatter RCSs for each Diameter size is put in a separate RCS
% Matrix to account for the differences due to frequency

% Part III
% Randomly place all drops in cube

% Part IV
% For every pulse the coherent Electric field return is calculated. Between
% pulses all drops are moved.

% Part V
% Calculate and plot results
%-----------------------------------------------------------------------

clear all
close all
clc

% Input Data
c=3e8;                % Speed of light [m/s]

% Radar Data
P_t=10e3;             % Transmitted power
G_dB=20;              % Maximum gain in dB
G=10^(G_dB/10);       % Maximum gain
angle_3db=0.20;       % 3 dB Beam angle [degrees], (half half power beamwidth)
pulsewidth=0.20;      % Pulsewidth [microsecond]
elevation=3;          % Elevation angle [degrees]
margine=5;            % margin for resolution cell[m]
azimuth=0;            % Azimuth angle [rad]
number_pulses=40;     % Number of pulses
PRT_1=1/5000;         % Doppler Pulse Repetition Time [seconds]
PRT_2=1/200;          % Avg Power Pulse Repetition Time [seconds]
distance=0.5;         % Range from the radar to the resolution cell [Km]
wavelength=0.1;       % Wavelength [m]
frequency=c/wavelength; % Frequency [Hz]

%-----------------------------------------------------------------------
%Introduced in stepped version
fq_step=1e6;          % Frequency step in [Hz]
delay=50e-6;          % Delay for changing fq in [s]
n_steps=3;            % Number of Frequency steps
%-----------------------------------------------------------------------

% Rain parameters
diam_low=0.05;        % Lowest diameter of Precipitation [mm]
diam_lim=1;           % Divider between low and mid [mm]
diam_mid=5;           % Divider between mid and high [mm]
diam_high=15;         % Highest diameter of Precipitation [mm]
resolution1=0.08;     % Lower diameter resolution [mm]
resolution2=0.3;      % Higher diameter resolution [mm]
resolution3=0.8;      % Highest diameter resolution [mm]
sort_approx=1;        % RCS approximation model; [1]=Rayleigh
kw=0.93;              % abs(K)^2 approx for water
rain_rate=50;         % Rain Rate [mm/h]
rain_anglel=0;        % degrees Rain Angle (Phi)
rain_angle2=0;        % degrees Rain Angle (Theta)
wind_vector=[-10;0;0];  % Wind speed vector (x,y,z) [m/s]
spread=0;             % Wind speed spread [m/s], (all directions)
spread_2=4;           % Drop velocity spread [m/s],
Norm=100;             % Normalization coefficient

% Directional cosines
```

```
u=sin(pi/2-elevation*pi/180)*cos(azimuth*pi/180);
v=sin(pi/2-elevation*pi/180)*sin(azimuth*pi/180);
w=cos(pi/2-elevation*pi/180);

%-------------------------------------------------------------------------
% Part I, Generate the resolution volume
%-------------------------------------------------------------------------
% Approximation of the resolution cell of the pulse circular box,
% for any azimuth angle
%-------------------------------------------------------------------------
% INPUT
% - distance
% - angle_3dB
% - pulsewidth
% - elevation
% - margine
% - azimuth
% ------------------------------------------------------------------------
% OUTPUT
% - volume_resolution: computed volume of the resolution cell
% - volume_box: volume of the box where the drops are put
% - box: the 8 rectangular coordinates of the box
% - coord_vol_res: the coordinates of the resolution cell where the drops
% are taken.
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% Creating the circular beam resolution cell
range_res=(c*pulsewidth*1e-6)/2;                    ...
   % Range resolution (ct/2) [m]

% small angle approximation allows for following
angle_rad=(angle_3db*pi)/180;                       ...
   % 3 dB beamwidth [rad]
radius1=(tan(angle_rad)*(distance*1000));             ...
   % Radius of beam at R=distance
radius2=(tan(angle_rad)*((distance*1000)+range_res));   ...
   % Radius of beam at R=distance + range resolution

surface1=pi*radius1^2;
surface2=pi*radius2^2;

% volume of cone = base surface times height for all three
volume_cone1=(surface1*(distance*1000))/3;
volume_cone2=(surface2*((distance*1000)+range_res))/3;
volume_resolution=(volume_cone2-volume_cone1);         % Beam Cone Volume

% computation of the resolution cell coordinates, spherical coordinates
% elevation in radians
elevation_rad=(elevation*pi)/180;                   % Elevation [rad]
azimuth_rad=(azimuth*pi)/180;                       % Azimuth [rad]

% Matrix used when comparing wheather a drop is inside or outside
% beamwidth
coord_vol_res=[distance*1000,(distance*1000)+range_res;elevation_rad-angle_rad,...
       elevation_rad+angle_rad;azimuth_rad-angle_rad,azimuth_rad+angle_rad];
```

```
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% Creating the box
% computation of the box and its coordinates
coin1=[azimuth_rad+angle_rad,elevation_rad+angle_rad,distance*1000];
coin2=[azimuth_rad-angle_rad,elevation_rad+angle_rad,distance*1000];
coin3=[azimuth_rad+angle_rad,elevation_rad-angle_rad,distance*1000];
coin4=[azimuth_rad-angle_rad,elevation_rad-angle_rad,distance*1000];
coin5=[azimuth_rad+angle_rad,elevation_rad+angle_rad,distance*1000+range_res];
coin6=[azimuth_rad-angle_rad,elevation_rad+angle_rad,distance*1000+range_res];
coin7=[azimuth_rad+angle_rad,elevation_rad-angle_rad,distance*1000+range_res];
coin8=[azimuth_rad-angle_rad,elevation_rad-angle_rad,distance*1000+range_res];

[x1,y1,z1]=sph2cart(coin1(1),coin1(2),coin1(3));
[x2,y2,z2]=sph2cart(coin2(1),coin2(2),coin2(3));
[x3,y3,z3]=sph2cart(coin3(1),coin3(2),coin3(3));
[x4,y4,z4]=sph2cart(coin4(1),coin4(2),coin4(3));
[x5,y5,z5]=sph2cart(coin5(1),coin5(2),coin5(3));
[x6,y6,z6]=sph2cart(coin6(1),coin6(2),coin6(3));
[x7,y7,z7]=sph2cart(coin7(1),coin7(2),coin7(3));
[x8,y8,z8]=sph2cart(coin8(1),coin8(2),coin8(3));

xmax=max([x1,x2,x3,x4,x5,x6,x7,x8]);
xmin=min([x1,x2,x3,x4,x5,x6,x7,x8]);
ymax=max([y1,y2,y3,y4,y5,y6,y7,y8]);
ymin=min([y1,y2,y3,y4,y5,y6,y7,y8]);
zmax=max([z1,z2,z3,z4,z5,z6,z7,z8]);
zmin=min([z1,z2,z3,z4,z5,z6,z7,z8]);

% the box with its margin
coin_1=[xmin-margine,ymin-margine,zmin-margine];
coin_2=[xmin-margine,ymax+margine,zmin-margine];
coin_3=[xmax+margine,ymax+margine,zmin-margine];
coin_4=[xmax+margine,ymin-margine,zmin-margine];
coin_5=[xmin-margine,ymin-margine,zmax+margine];
coin_6=[xmin-margine,ymax+margine,zmax+margine];
coin_7=[xmax+margine,ymax+margine,zmax+margine];
coin_8=[xmax+margine,ymin-margine,zmax+margine];


box=[coin_1',coin_2',coin_3',coin_4',coin_5',coin_6',coin_7',coin_8'];

% calculate the volume of the box
volume_box=abs((xmax+margine-(xmin-margine))*(ymax+margine-(ymin-margine))...
   *(zmax+margine-(zmin-margine)));


cube=[coin_1',coin_2',coin_3',coin_4',coin_1',coin_5',coin_6',coin_2',...
      coin_6',coin_7',coin_3',coin_7',coin_8',coin_4',coin_8',coin_5'];
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% Part II, Generate the fundamental T Matrix
%-----------------------------------------------------------------------
% This part produces a matrix of all drops that contribute to the total
% reflectivity of the resolution cell
%------------------------------------------------------------------------
% First all sizes are created, with two vectors of different size resolution.
% The size vectors represents all diameters. Next the backscatter RCS for each
% size is computed.
% Using the Marshall-Palmer exponential approximation, the number of drops of
% each diameter is computed.
%------------------------------------------------------------------------
% INPUT
% - diam_low
% - diam_lim
% - diam_high
% - resolution1
% - resolution2
% - sort_approx
% - wavelength
% - Kw
% - rain_rate
%------------------------------------------------------------------------
% OUTPUT
% - number_drops
% Matrix T:
% - first row all sizes
% - second row all the backscatter RCSs
% - third row the total number of drops for each size per m^3
% the columns starting from the smallest size
%------------------------------------------------------------------------
% Create dropsize vector

% Precipitaion Diameter vector (lowest<D<divider)
vector_size_1=diam_low:resolution1:diam_lim;

% Precipitaion Diameter vector (divider<D<mid)
vector_size_2=diam_lim:resolution2:diam_mid;

% Precipitaion Diameter vector (mid<D<highest)
vector_size_3=diam_mid:resolution3:diam_high;

% Precipitaion Diameter Vector
vector_size=[vector_size_1,vector_size_2,vector_size_3];

%------------------------------------------------------------------------
% Create RCS matrix for all Rain drops of Precipitation Vector and for all
% frequencies.
fq_rr=frequency:fq_step:frequency+(n_steps-1)*fq_step;
rcs(1:n_steps,:)=((kw^2)*(pi^5)/(c^4)*fq_rr'.^4)*((vector_size*1e-3).^6);
%------------------------------------------------------------------------

%------------------------------------------------------------------------
% The Marshall_Palmer approximation is used to create Precipitaion Diameter
% Distribution Vector (Drop Size Distribution N(D) Vector)
% The number of drops are scaled down by a factor (Norm) to improve the
% speed of the simulator
%------------------------------------------------------------------------
lambda=4.1*(rain_rate^(-0.21));         %[mm-1]
number=8000*exp(-lambda*vector_size)/Norm;
```

```
vector_res1=ones(1,length(vector_size_1))*resolution1;
vector_res2=ones(1,length(vector_size_2))*resolution2;
vector_res3=ones(1,length(vector_size_3))*resolution3;

% delta(Diameter) dD
vector_res=[vector_res1,vector_res2,vector_res3];

% the approximate "true" number of drops N(D)dD vector
number=number.*vector_res;
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% create matrix T without rcs
T=[vector_size;number];
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% Change matrix 'T' to matrix 'matrix_T_Round'
%-----------------------------------------------------------------------
% This part computes the total number of drops by utilizing the resolution
% cell and the matrix_T_Round
% The only difference between the matrix_T_Round and T, is the fact that
% matrix_T_Round has the total number of drops on the third row and they
% are rounded of to nearest integer value.
%-----------------------------------------------------------------------

matrix_T_Round=T;
matrix_T_Round(2,:)=round(matrix_T_Round(2,:)*volume_box);

% Keep track of errors due to round off
diff=T(2,:)*volume_box-matrix_T_Round(2,:);
error_rcs=diff*rcs'*Norm/volume_box*volume_resolution;
error_Z=error_rcs/volume_resolution*c^4./fq_rr.^4/pi^5/abs(kw)^2/1e-18;

%-----------------------------------------------------------------------
% Part III, Initial positioning of the drops within the volume
%-----------------------------------------------------------------------
% This part generates the initial position of the drops in the volume of
% the box.
% The positions are recorded in the file "sizexx.mat' taking into account
% the azimuth angle
%-----------------------------------------------------------------------
% INPUT
% - box
% - matrix_T_Round
% - elevation
% - azimuth
%-----------------------------------------------------------------------
% OUTPUT
% All files including the sizes.  Each contains the positions of all drops
% for a given diameter.  The data recoreded as follows:
% - radial distance r
% - elevation phi
% - azimuth theta
% - angle with respect to beam center.  This parameter will be
% used to identify drops which are in the beam.
%-----------------------------------------------------------------------

% Randomly put the calculated number of drops of each diameter in the box
number_size=length(matrix_T_Round(1,:));

for i=1:number_size

   % select number of drops of certain Diameter
          drops=matrix_T_Round(2,i);

   % create as many positions on the x axis as # drops per size
          elements_x=rand(1,drops);

   % put them in the range of interest
   % Random placement x for every drop
          position_x=box(1,1)+(box(1,3)-box(1,1))*elements_x;

   % create as many positionson the y axis as # drops per size
          elements_y=rand(1,drops);

   % put them in the range of interest
   % Random placement y for every drop
          position_y=box(2,1)+(box(2,2)-box(2,1))*elements_y;

   % create as many positions on the z axis as # drops per size
          elements_z=rand(1,drops);

   % put them in the range of interest
   % Random placement z for every drop
          position_z=box(3,1)+(box(3,5)-box(3,1))*elements_z;

   % matrix of results
   % Creating a drop Matrix using spherical coordinates
          [theta,phi,r]=cart2sph(position_x,position_y,position_z);

   % Calculating the range, phi, theta and angle to every drop with
   % reference to phi_0 and theta_0. Used to estimate weighted power return.
   vector_azimuth=ones(1,drops)*((azimuth*pi)/180);
   vector_elevation=ones(1,drops)*((elevation*pi)/180);
   [xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);
   range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);

   % approximation
   angle=atan(range./r);

   % Saving the drop positions in size%d.mat
   % matrixpos
   matrixpos=[r;phi;theta;angle];

   save(sprintf('size%d.mat',i),'matrixpos');

end
%-----------------------------------------------------------------------
```

```
%------------------------------------------------------------------------
% Part IV, Calculate Coherent Electric field return.
%------------------------------------------------------------------------
% For every pulse the coherent Electric field return is calculatet. Between
% pulses all drops are moved.
%
% Run the simulation for specified number of pulses
%------------------------------------------------------------------------

% Zero all vectors and matrices that are going to be used
phase_vector=zeros(1,2*number_pulses);
imaginary_return=zeros(1,2*number_pulses);
sum_E=zeros(n_steps,ceil(number_pulses/n_steps));
weighted_E=zeros(n_steps,ceil(number_pulses/n_steps));
running_Z=[];

% Initiate reference variables
p=0;                        % Used to separate pulses
hh=0;                       % Used to separate avg power samples

while p<number_pulses

   step_counter=0;          % Used to separate frequencies
   hh=hh+1;

   for fq_new=frequency:fq_step:frequency+(n_steps-1)*fq_step

      k=2*pi*fq_new/c;
      p=p+1;
      step_counter=step_counter+1;

      disp(sprintf('calculate the power-return of pulse %d\n',p));

      %------------------------------------------------------------------
      % computes the (weighted) I and Q return
      %------------------------------------------------------------------
                 % computes the total number of drops
                 %------------------------------------------------------------------
                 % INPUT
                 % - Matrix T
                 % - wavelength
                 % - angle_3dB
                 % - distance
                 % - coord_vol_res
                 % - number_pulses
                 %------------------------------------------------------------------
                 % OUTPUT
                 % - total_E: the received complex Electric field)
                 % - power: the power
                 % - voltage: the voltage
                 % - drops_beam: the number of drops in the beam
                 %------------------------------------------------------------------
                 % Only the drops which are in the resolution cell are selected for the
                 % computation
                 %------------------------------------------------------------------
```

```
number_size=length(matrix_T_Round(1,:));
total_E=0;
total_phase=0;
new_E=0;

for i=1:number_size

   % opening the file of registered data matrixpos.
   load(sprintf('size%d.mat',i));

   % removing drops that are outside the resolution cell
   number_drops_per_size=length(matrixpos(1,:));
   dropmatrix=[];

   % Filling the drop matrix with all drops inside the beam
   % resolution volume
   for d=1:number_drops_per_size
      if ((matrixpos(4,d)<=(angle_3db*pi/180))&(matrixpos(1,d)>=...
            coord_vol_res(1,1))&(matrixpos(1,d)<=coord_vol_res(1,2)));
         dropmatrix=[dropmatrix,matrixpos(:,d)];
      end
   end

   %-----------------------------------------------------------------
   % Convert to cartesian coordinates
   if isempty(dropmatrix)==0
      [position_x,position_y,position_z]=sph2cart(dropmatrix(3,:),...
         dropmatrix(2,:),dropmatrix(1,:));

      % Create comparison vectors
      vector_azimuth=ones(1,length(dropmatrix(1,:)))*((azimuth*pi)/180);
      vector_elevation=ones(1,length(dropmatrix(1,:)))*((elevation*pi)/180);


      % Create a phase vector for the drops of current Diameter
      r_phase=position_x.*u+position_y.*v+position_z.*w;
      phase=exp((2*j*k).*r_phase);

      % Sum the contributions to the E-field
      total_phase=total_phase+sum(phase);
      total_E=total_E+sum(sqrt(rcs(step_counter,i)).*phase);

      %--------------------------------------------------------------
      % ceate weighted power return, concidering range (r), theta, phi
      % with reference to boresight and radar parameters
      E_weighted=sqrt(P_t)*G*c/fq_rr(step_counter)/(sqrt(4*pi)^3)./...
         dropmatrix(1,:).^2.*(exp(-4*log(2).*((dropmatrix(3,:)-...
         vector_azimuth).^2/(2*angle_3db*pi/180)^2+((dropmatrix(2,:)...
         -vector_elevation).^2/(2*angle_3db*pi/180)^2))));
      new_E=new_E+sum(E_weighted.*(sqrt(rcs(step_counter,i)).*phase));

   end

end

%-----------------------------------------------------------------
```

```
phase_vector(2*p-1)=ANGLE(total_phase);
imaginary_return(2*p-1)=total_phase;

% add to E-field and drop- and rate-vector
sum_E(step_counter,hh)=total_E;
weighted_E(step_counter,hh)=new_E;

%-------------------------------------------------------------------

                %-------------------------------------------------------------------
                % This part simulates the movement of the drops and gives new positions.
% First using the Doppler PRT (PRT_1)
                %-------------------------------------------------------------------
                % INPUT
                % - matrix_T_Round
                % - PRT
                % - azimuth
                % - elevation
                % - rain_angle1
                % - rain_angle2
                % - diam_lim
% - resolution1
% - resolution2
                % - wind_vector
                %-------------------------------------------------------------------
                % OUTPUTS
                % Output files containing sizes with new positions
                % structured the same way as in previous
                % there is also an output file for speed (one for each size) for the pulse
                % pair.
                %-------------------------------------------------------------------

number_size=length(matrix_T_Round(1,:));

% adding a Gaussian random term to make the wind speed spread
if wind_vector(1)~=0
   new_wind_vector(1)=wind_vector(1)*normrnd(1,abs(spread/max(wind_vector(1))));
else
   new_wind_vector(1)=0;
end
if wind_vector(2)~=0
   new_wind_vector(2)=wind_vector(2)*normrnd(1,abs(spread/max(wind_vector(2))));
else
   new_wind_vector(2)=0;
end
if wind_vector(3)~=0
   new_wind_vector(3)=wind_vector(3)*normrnd(1,abs(spread/max(wind_vector(3))));
else
   new_wind_vector(3)=0;
end

for i=1:number_size

   % opening of data file registered in matrixpos.
   load(sprintf('size%d.mat',i));
```

```
          % total number of drops
          drops=matrix_T_Round(2,i);

          % transforming to rectangular coordinates
          [x,y,z]=sph2cart(matrixpos(3,:),matrixpos(2,:),matrixpos(1,:));

          diameter=matrix_T_Round(1,i);

          %---------------------------------------------------------------
          % Drop fall speed depends on drop diameter in the z axis,
          % Atlas-Ulbrich approximation is used
                          % Approximation is valid in the diameter range 5*e-4, 5*e-3
                          %--------------------------------------------------------------

          wtmax=386.6*((matrix_T_Round(1,i)*0.001)^0.67);

          velocityz=(wtmax*(ones(1,drops)))*cos((rain_anglel*pi)/180);
          velocityx=(wtmax*(ones(1,drops)))*sin((rain_anglel*pi)/180)*...
             cos((rain_angle2*pi)/180);
          velocityy=(wtmax*(ones(1,drops)))*sin((rain_anglel*pi)/180)*...
             sin((rain_angle2*pi)/180);

          %---------------------------------------------------------------
          %---------------------------------------------------------------
%            % adding another Gaussian random term to make the wind speed
%            % spread between different drops sizes
%            if new_wind_vector(1)~=0
%               newest_wind_vector(1)=new_wind_vector(1)*normrnd(1,abs(spread_2/...
%                  max(wind_vector(1))));
%            else
%               newest_wind_vector(1)=0;
%            end
%            if new_wind_vector(2)~=0
%               newest_wind_vector(2)=new_wind_vector(2)*normrnd(1,abs(spread_2/...
%                  max(wind_vector(2))));
%            else
%               newest_wind_vector(2)=0;
%            end
%            if new_wind_vector(3)~=0
%               newest_wind_vector(3)=new_wind_vector(3)*normrnd(1,abs(spread_2/...
%                  max(wind_vector(3))));
%            else
%               newest_wind_vector(3)=0;
%            end
%
%            % If spread within pulse
%            velocityxt=newest_wind_vector(1)-velocityx;
%            velocityyt=newest_wind_vector(2)-velocityy;
%            velocityzt=newest_wind_vector(3)-velocityz;
%
%            % speed registration
%            m_velocity=[velocityxt;velocityyt;velocityzt];
%            save(sprintf('velocity%d.mat',i),'m_velocity');

          %---------------------------------------------------------------
```

```
%-----------------------------------------------------------------
% If spread for all drops regardless of size
newest_wind_vector=[];


  % adding Gaussian random term to make the wind speed
  % spread between drops
if new_wind_vector(1)~=0
    newest_wind_vector(1,:)=new_wind_vector(1)*...
        normrnd(1,abs(spread_2/max(new_wind_vector(1))),1,drops);
else
    newest_wind_vector(1,:)=0;
end
if new_wind_vector(2)~=0
    newest_wind_vector(2,:)=new_wind_vector(2)*...
        normrnd(1,abs(spread_2/max(new_new_wind_vector(2))),1,drops);
else
    newest_wind_vector(2,:)=0;
end
if new_wind_vector(3)~=0
    newest_wind_vector(3,:)=new_wind_vector(3)*...
        normrnd(1,abs(spread_2/max(new_wind_vector(3))),1,drops);
else
    newest_wind_vector(3,:)=0;
end

% If spread within pulse
velocityxt=newest_wind_vector(1,:)-velocityx;
velocityyt=newest_wind_vector(2,:)-velocityy;
velocityzt=newest_wind_vector(3,:)-velocityz;

%-----------------------------------------------------------------
%-----------------------------------------------------------------

deplacementx=velocityxt*PRT_1;
deplacementy=velocityyt*PRT_1;
deplacementz=velocityzt*PRT_1;

position_x=x+deplacementx;
position_y=y+deplacementy;
position_z=z+deplacementz;

% transformation into spherical coordinates
[theta,phi,r]=cart2sph(position_x,position_y,position_z);

% computation of new angles
vector_azimuth=ones(1,drops)*(azimuth*pi)/180;
vector_elevation=ones(1,drops)*(elevation*pi)/180;
[xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);

range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);
% approximation
angle=atan(range./r);

% matrixpos
matrixpos=[r;phi;theta;angle];
```

```
   % save to file, matrixpos
                              s=sprintf('size%d.mat',i);
                    save(s, 'matrixpos');

end

                %-------------------------------------------------------------------
% Calculate phase for pulse pair
total_phase=0;

for i=1:number_size

   % opening the file of registered data matrixpos.
   load(sprintf('size%d.mat',i));

   % removing drops that are outside the resolution cell
   number_drops_per_size=length(matrixpos(1,:));
   dropmatrix=[];

   % Filling the drop matrix with all drops inside the beam
   % resolution volume
   for d=1:number_drops_per_size
      if ((matrixpos(4,d)<=(angle_3db*pi/180))&(matrixpos(1,d)>=...
            coord_vol_res(1,1))&(matrixpos(1,d)<=coord_vol_res(1,2)));
         dropmatrix=[dropmatrix,matrixpos(:,d)];
      end
   end

   %-----------------------------------------------------------------
   % Convert to cartesian coordinates
   if isempty(dropmatrix)==0
      [position_x,position_y,position_z]=sph2cart(dropmatrix(3,:),...
         dropmatrix(2,:),dropmatrix(1,:));

      % Create comparison vectors
      vector_azimuth=ones(1,length(dropmatrix(1,:)))*((azimuth*pi)/180);
      vector_elevation=ones(1,length(dropmatrix(1,:)))*((elevation*pi)/180);


      % Create a phase vector for the drops of current Diameter
      r_phase=position_x.*u+position_y.*v+position_z.*w;
      phase=exp((2*j*k).*r_phase);

      % Sum the contributions to the E-field
      total_phase=total_phase+sum(phase);
   end
end

phase_vector(2*p)=ANGLE(total_phase);
imaginary_return(2*p)=total_phase;


%-------------------------------------------------------------------
                % This part simulates the movement of the drops and gives new positions.
% Using the doppler PRT and the delay (PRT_2, delay)
```

```
               %------------------------------------------------------------------

               %------------------------------------------------------------------

number_size=length(matrix_T_Round(1,:));

   % adding a Gaussian random term to make the wind speed spread
   if wind_vector(1)~=0
      new_wind_vector(1)=wind_vector(1)*normrnd(1,abs(spread/...
         max(wind_vector(1))));
   else
      new_wind_vector(1)=0;
   end
   if wind_vector(2)~=0
      new_wind_vector(2)=wind_vector(2)*normrnd(1,abs(spread/...
         max(wind_vector(2))));
   else
      new_wind_vector(2)=0;
   end
   if wind_vector(3)~=0
      new_wind_vector(3)=wind_vector(3)*normrnd(1,abs(spread/...
         max(wind_vector(3))));
   else
      new_wind_vector(3)=0;
   end

   for i=1:number_size

      % opening of data file registered in matrixpos.
      load(sprintf('size%d.mat',i));

      % total number of drops
      drops=matrix_T_Round(2,i);

      % transforming to rectangular coordinates
      [x,y,z]=sph2cart(matrixpos(3,:),matrixpos(2,:),matrixpos(1,:));

      diameter=matrix_T_Round(1,i);

      %----------------------------------------------------------------
      % Drop fall speed depends on drop diameter in the z axis,
      % Atlas-Ulbrich approximation is used
                          % Approximation is valid in the diameter range 5*e-4, 5*e-3
                          %--------------------------------------------------------------

      wtmax=386.6*((matrix_T_Round(1,i)*0.001)^0.67);

      velocityz=(wtmax*(ones(1,drops)))*cos((rain_angle1*pi)/180);
      velocityx=(wtmax*(ones(1,drops)))*sin((rain_angle1*pi)/180)*...
         cos((rain_angle2*pi)/180);
      velocityy=(wtmax*(ones(1,drops)))*sin((rain_angle1*pi)/180)*...
         sin((rain_angle2*pi)/180);

      %------------------------------------------------------
      %------------------------------------------------------
```

```
%           % adding another Gaussian random term to make the wind speed
%           % spread between different drops sizes
%           if new_wind_vector(1)~=0
%              newest_wind_vector(1)=new_wind_vector(1)*normrnd(1,abs(spread_2/...
%                 max(wind_vector(1))));
%           else
%              newest_wind_vector(1)=0;
%           end
%           if new_wind_vector(2)~=0
%              newest_wind_vector(2)=new_wind_vector(2)*normrnd(1,abs(spread_2/...
%                 max(wind_vector(2))));
%           else
%              newest_wind_vector(2)=0;
%           end
%           if new_wind_vector(3)~=0
%              newest_wind_vector(3)=new_wind_vector(3)*normrnd(1,abs(spread_2/...
%                 max(wind_vector(3))));
%           else
%              newest_wind_vector(3)=0;
%           end
%
%           % If spread within pulse
%           velocityxt=newest_wind_vector(1)-velocityx;
%           velocityyt=newest_wind_vector(2)-velocityy;
%           velocityzt=newest_wind_vector(3)-velocityz;
%
        %-----------------------------------------------------------------
        %-----------------------------------------------------------------
        % If spread for all drops regardless of size
        newest_wind_vector=[];


         % adding Gaussian random term to make the wind speed
         % spread between drops
        if new_wind_vector(1)~=0
           newest_wind_vector(1,:)=new_wind_vector(1)*...
              normrnd(1,abs(spread_2/max(new_wind_vector(1))),1,drops);
        else
           newest_wind_vector(1,:)=0;
        end
        if new_wind_vector(2)~=0
           newest_wind_vector(2,:)=new_wind_vector(2)*...
              normrnd(1,abs(spread_2/max(new_new_wind_vector(2))),1,drops);
        else
           newest_wind_vector(2,:)=0;
        end
        if new_wind_vector(3)~=0
           newest_wind_vector(3,:)=new_wind_vector(3)*...
              normrnd(1,abs(spread_2/max(new_wind_vector(3))),1,drops);
        else
           newest_wind_vector(3,:)=0;
        end
        % If spread within pulse
        velocityxt=newest_wind_vector(1,:)-velocityx;
        velocityyt=newest_wind_vector(2,:)-velocityy;
        velocityzt=newest_wind_vector(3,:)-velocityz;

        %----------------------------------------------------------------
        %----------------------------------------------------------------


        % speed registration
        m_velocity=[velocityxt;velocityyt;velocityzt];
        save(sprintf('velocity%d.mat',i),'m_velocity');

        deplacementx=velocityxt*(PRT_1+delay);
        deplacementy=velocityyt*(PRT_1+delay);
        deplacementz=velocityzt*(PRT_1+delay);

        position_x=x+deplacementx;
        position_y=y+deplacementy;
        position_z=z+deplacementz;

        % transformation into spherical coordinates
        [theta,phi,r]=cart2sph(position_x,position_y,position_z);

        % computation of new angles
        vector_azimuth=ones(1,drops)*(azimuth*pi)/180;
        vector_elevation=ones(1,drops)*(elevation*pi)/180;
        [xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);

        range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);
        % approximation
        angle=atan(range./r);

        % matrixpos
        matrixpos=[r;phi;theta;angle];

        % save to file, matrixpos
                                s=sprintf('size%d.mat',i);
                        save(s, 'matrixpos');

    end


                %-------------------------------------------------------------------

%-------------------------------------------------------------------
                % This part simulates the movement of the drops and gives new positions.
% Using the Avg Power PRT reduced by the sample times for the previous
% samples
                %-------------------------------------------------------------------

                %-------------------------------------------------------------------
number_size=length(matrix_T_Round(1,:));

    % adding a Gaussian random term to make the wind speed spread
    if wind_vector(1)~=0
```

```
        new_wind_vector(1)=wind_vector(1)*normrnd(1,abs(spread/...
            max(wind_vector(1))));
    else
        new_wind_vector(1)=0;
    end
    if wind_vector(2)~=0
        new_wind_vector(2)=wind_vector(2)*normrnd(1,abs(spread/...
            max(wind_vector(2))));
    else
        new_wind_vector(2)=0;
    end
    if wind_vector(3)~=0
        new_wind_vector(3)=wind_vector(3)*normrnd(1,abs(spread/...
            max(wind_vector(3))));
    else
        new_wind_vector(3)=0;
    end


for i=1:number_size

    % opening of data file registered in matrixpos.
    load(sprintf('size%d.mat',i));

    % total number of drops
    drops=matrix_T_Round(2,i);

    % transforming to rectangular coordinates
    [x,y,z]=sph2cart(matrixpos(3,:),matrixpos(2,:),matrixpos(1,:));

    diameter=matrix_T_Round(1,i);

    %-----------------------------------------------------------------
    % Drop fall speed depends on drop diameter in the z axis,
    % Atlas-Ulbrich approximation is used
                        % Approximation is valid in the diameter range 5*e-4, 5*e-3
                        %-----------------------------------------------------------------

    wtmax=386.6*((matrix_T_Round(1,i)*0.001)^0.67);

    velocityz=(wtmax*(ones(1,drops)))*cos((rain_anglel*pi)/180);
    velocityx=(wtmax*(ones(1,drops)))*sin((rain_anglel*pi)/180)*...
        cos((rain_angle2*pi)/180);
    velocityy=(wtmax*(ones(1,drops)))*sin((rain_anglel*pi)/180)*...
        sin((rain_angle2*pi)/180);

    %-----------------------------------------------------------------
    %-----------------------------------------------------------------
%       % adding another Gaussian random term to make the wind speed
%       % spread between different drops sizes
%       if new_wind_vector(1)~=0
%           newest_wind_vector(1)=new_wind_vector(1)*normrnd(1,abs(spread_2/...
%               max(wind_vector(1))));
%       else
%           newest_wind_vector(1)=0;
%       end
%           if new_wind_vector(2)~=0
%              newest_wind_vector(2)=new_wind_vector(2)*normrnd(1,abs(spread_2/...
%                 max(wind_vector(2))));
%           else
%              newest_wind_vector(2)=0;
%           end
%           if new_wind_vector(3)~=0
%              newest_wind_vector(3)=new_wind_vector(3)*normrnd(1,abs(spread_2/...
%                 max(wind_vector(3))));
%           else
%              newest_wind_vector(3)=0;
%           end
%
%           % If spread within pulse
%           velocityxt=newest_wind_vector(1)-velocityx;
%           velocityyt=newest_wind_vector(2)-velocityy;
%           velocityzt=newest_wind_vector(3)-velocityz;

        %------------------------------------------------------------------
        %------------------------------------------------------------------
        % If spread for all drops regardless of size
        newest_wind_vector=[];


          % adding Gaussian random term to make the wind speed
          % spread between drops
        if new_wind_vector(1)~=0
           newest_wind_vector(1,:)=new_wind_vector(1)*...
              normrnd(1,abs(spread_2/max(new_wind_vector(1))),1,drops);
        else
           newest_wind_vector(1,:)=0;
        end
        if new_wind_vector(2)~=0
           newest_wind_vector(2,:)=new_wind_vector(2)*...
              normrnd(1,abs(spread_2/max(new_new_wind_vector(2))),1,drops);
        else
           newest_wind_vector(2,:)=0;
        end
        if new_wind_vector(3)~=0
           newest_wind_vector(3,:)=new_wind_vector(3)*...
              normrnd(1,abs(spread_2/max(new_wind_vector(3))),1,drops);
        else
           newest_wind_vector(3,:)=0;
        end

        % If spread within pulse
        velocityxt=newest_wind_vector(1,:)-velocityx;
        velocityyt=newest_wind_vector(2,:)-velocityy;
        velocityzt=newest_wind_vector(3,:)-velocityz;

        %------------------------------------------------------------------
        %------------------------------------------------------------------


        % speed registration
        m_velocity=[velocityxt;velocityyt;velocityzt];
```

```
            save(sprintf('velocity%d.mat',i),'m_velocity');

            deplacementx=velocityxt*(PRT_2-n_steps*(PRT_1+delay));
            deplacementy=velocityyt*(PRT_2-n_steps*(PRT_1+delay));
            deplacementz=velocityzt*(PRT_2-n_steps*(PRT_1+delay));

            position_x=x+deplacementx;
            position_y=y+deplacementy;
            position_z=z+deplacementz;

            % transformation into spherical coordinates
            [theta,phi,r]=cart2sph(position_x,position_y,position_z);

            % computation of new angles
            vector_azimuth=ones(1,drops)*(azimuth*pi)/180;
            vector_elevation=ones(1,drops)*(elevation*pi)/180;
            [xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);

            range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);
            % approximation
            angle=atan(range./r);

            % matrixpos
            matrixpos=[r;phi;theta;angle];

            % save to file, matrixpos
                                s=sprintf('size%d.mat',i);
                        save(s, 'matrixpos');
        end

    end
end

%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% Calculate RCS and doppler
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% calculate RCS, Z and Power return for every pulse including error
%-------------------------------------------------------------------------
for o=1:n_steps
    running_Z(o,:)=(Norm*abs(sum_E(o,:)).^2)*c^4/fq_rr(o)^4/pi^5/...
        abs(kw)^2/volume_resolution/1e-18+error_Z(o);
end

running_Z=reshape(running_Z,1,[]);      % Transform into a vector

%-------------------------------------------------------------------------
% Adding the error to the weighted_E estimations
%-------------------------------------------------------------------------

error_power=error_rcs*P_t*G^2*c^2./fq_rr.^2/pi^2*(2*angle_3db*pi/180)^2*...
    (c*pulsewidth*1e-6)...
    /1024/log(2)/pi^2/(distance*1e3)^2;
```

```
for o=1:n_steps
   power_return_matrix(o,:)=Norm*abs(weighted_E(o,:)).^2+error_power(o);
end

power_return=reshape(power_return_matrix,1,[]);% Transform into a vector

%------------------------------------------------------------------------
% calculate average Power and Z
%------------------------------------------------------------------------
Z_avg=sum(running_Z)/p;
Z_avg_plot=ones(1,p)*Z_avg;

power_return_avg=sum(sum(Norm*abs(weighted_E).^2,2)+error_power')/p;  % [W]
power_return_avg_plot=ones(1,p)*power_return_avg;

%------------------------------------------------------------------------
% calculate doppler frequency
% use unwrap to make sure no alias is presented
%------------------------------------------------------------------------
phase_vector_u=unwrap(phase_vector);
doppler_fq=([phase_vector_u(2:2:2*p)-phase_vector_u(1:2:2*p-1)])/2/pi/PRT_1;

number_vector=1:p;

%------------------------------------------------------------------------
% Calculate references
% Integral form of Z
%------------------------------------------------------------------------
ii=0:100/1000:100;
zz=ii.^6.*8000.*exp(-4.1.*rain_rate.^(-.21).*ii);
real_Z=ones(1,p)*10*log10(trapz(ii,zz));

%------------------------------------------------------------------------
% Evaluate average Power return to make Z estimation including error
%------------------------------------------------------------------------
% To make the estimate with the correct frequency, the power return Matrix
% is used. For the plot the result must me summed and divided by the number
% of samples used to average

Z_estimation=power_return_matrix'*(fq_rr.^2)'/P_t/G^2/c^2*...
   pi^2/(2*angle_3db*pi/180)^2/(c*pulsewidth*1e-6)*1024*log(2)*(distance*1e3)^2/...
   ((kw^2)*(pi^5)/(wavelength^4))/1e-18;
Z_estimation_plot=ones(1,p)*10*log10((sum(Z_estimation)+sum(error_Z))/p);

%------------------------------------------------------------------------
% Part V, Plot of results
%------------------------------------------------------------------------

figure(1);
subplot(2,1,1);
plot(number_vector(1:p),doppler_fq(1:p),'-x',number_vector(1:p),ones(1,p)*...
   mean(doppler_fq),'--r');
title(['Doppler frequency for all pulse pairs for PRF= ',num2str(1/PRT_1),' Hz']);
xlabel('Pulse pair');
ylabel('f_d');
```

```
legend('Doppler frequency per pulse pair','Average Doppler estimation')
grid on


subplot(2,1,2)
% To account for the differences in wavelengths a correct wavelength vector
% is created and multiplied with the doppler frequency vector
u=ones(1,p/n_steps);
lambida=[u*c/fq_rr(1);u*c/fq_rr(2);u*c/fq_rr(3)];
wave_l=reshape(lambida,1,[]);
v_r=doppler_fq.*wave_l/2;

hist(v_r,15)
hold on

[ff,xx] = ksdensity(v_r);
plot(xx,ff/max(ff)*max(hist(v_r,15)))
title('Radial Velocity Histogram with Gaussian Fit');
xlabel('Radial velocity [m/s]');
ylabel('Frequency');
grid on


figure(2)
subplot(2,1,1);
kl=0:p/n_steps-1;
power_number_vector=[kl*PRT_2; kl*PRT_2+PRT_1+PRT_1+delay; kl*PRT_2+2*...
        (PRT_1+PRT_1+delay)];
power_number_vector=reshape(power_number_vector,1,[]);

text_string=['Power Return for all ',num2str(p),' pulses.',' Frequency stepped from ',...
        num2str(fq_rr(1)/1e9),' GHz to ',num2str(fq_rr(n_steps)/1e9),' GHz with ',...
        num2str(fq_step/1e6),' MHz steps.'];
% also converting to dBm
plot(power_number_vector*1000,10*log10(power_return)+30,'-xb',...
    power_number_vector*1000,10*log10(power_return_avg_plot)+30,'--r');
title(text_string);
xlabel('Sample time [ms]');
ylabel('Power [dBm]');
legend('Power return', 'Average Power return',0)
grid on

subplot(2,1,2);
plot(power_number_vector*1000,10*log10(abs(running_Z)),'-xb',power_number_vector*...
    1000,10*log10(abs(Z_avg_plot)),'-.r',power_number_vector*1000,real_Z,'-k',...
    power_number_vector*1000,Z_estimation_plot,'--b');
title('Estimated Z per pulse');
xlabel('Sample time');
ylabel('Z [dBZ]');
legend('Simulation Z, (instantaneous)', 'Estimated Z, (average return)',...
    'Z=\int D^6N(D)dD','Estimated Z, Zeroth moment',0)
grid on

figure (3)
% This plot only works if the number of pulse pairs exceed 30
polarmatrix(1:2:119)=zeros(1,60);
polarmatrix(2:2:120)=phase_vector_u(1:60);
lengthmatrix(1:2:119)=zeros(1,60);
lengthmatrix(2:2:120)=ones(1,60);
subplot(5,6,1)
polar(polarmatrix(1:2),lengthmatrix(1:2),'-*')
hold on
polar(polarmatrix(3:4),lengthmatrix(3:4),'--*r')
title(['1^{st}. v_r= ',num2str(doppler_fq(1)*c/fq_rr(1)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,2)
polar(polarmatrix(5:6),lengthmatrix(5:6),'-*')
hold on
polar(polarmatrix(7:8),lengthmatrix(7:8),'--*r')
title(['2^{nd}. v_r= ',num2str(doppler_fq(2)*c/fq_rr(2)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,3)
polar(polarmatrix(9:10),lengthmatrix(9:10),'-*')
hold on
polar(polarmatrix(11:12),lengthmatrix(11:12),':*r')
title(['3^{rd}. v_r= ',num2str(doppler_fq(3)*c/fq_rr(3)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,4)
polar(polarmatrix(13:14),lengthmatrix(13:14),'-*')
hold on
polar(polarmatrix(15:16),lengthmatrix(15:16),':*r')
title(['4^{th}. v_r= ',num2str(doppler_fq(4)*c/fq_rr(1)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,5)
polar(polarmatrix(17:18),lengthmatrix(17:18),'-*')
hold on
polar(polarmatrix(19:20),lengthmatrix(19:20),':*r')
title(['5^{th}. v_r= ',num2str(doppler_fq(5)*c/fq_rr(2)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,6)
polar(polarmatrix(21:22),lengthmatrix(21:22),'-*')
hold on
polar(polarmatrix(23:24),lengthmatrix(23:24),':*r')
title(['6^{th}. v_r= ',num2str(doppler_fq(6)*c/fq_rr(3)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,7)
polar(polarmatrix(25:26),lengthmatrix(25:26),'-*')
hold on
polar(polarmatrix(27:28),lengthmatrix(27:28),':*r')
title(['7^{th}. v_r= ',num2str(doppler_fq(7)*c/fq_rr(1)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,8)
polar(polarmatrix(29:30),lengthmatrix(29:30),'-*')
hold on
polar(polarmatrix(31:32),lengthmatrix(31:32),':*r')
```

```
title(['8^{th}. v_r= ',num2str(doppler_fq(8)*c/fq_rr(2)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,9)
polar(polarmatrix(33:34),lengthmatrix(33:34),'-*')
hold on
polar(polarmatrix(35:36),lengthmatrix(35:36),':*r')
title(['9^{th}. v_r= ',num2str(doppler_fq(9)*c/fq_rr(3)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,10)
polar(polarmatrix(37:38),lengthmatrix(37:38),'-*')
hold on
polar(polarmatrix(39:40),lengthmatrix(39:40),':*r')
title(['10^{th}. v_r= ',num2str(doppler_fq(10)*c/fq_rr(1)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,11)
polar(polarmatrix(41:42),lengthmatrix(41:42),'-*')
hold on
polar(polarmatrix(43:44),lengthmatrix(43:44),':*r')
title(['11^{th}. v_r= ',num2str(doppler_fq(11)*c/fq_rr(2)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,12)
polar(polarmatrix(45:46),lengthmatrix(45:46),'-*')
hold on
polar(polarmatrix(47:48),lengthmatrix(47:48),':*r')
title(['12^{th}. v_r= ',num2str(doppler_fq(12)*c/fq_rr(3)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,13)
polar(polarmatrix(49:50),lengthmatrix(49:50),'-*')
hold on
polar(polarmatrix(51:52),lengthmatrix(51:52),':*r')
title(['13^{th}. v_r= ',num2str(doppler_fq(13)*c/fq_rr(1)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,14)
polar(polarmatrix(53:54),lengthmatrix(53:54),'-*')
hold on
polar(polarmatrix(55:56),lengthmatrix(55:56),':*r')
title(['14^{th}. v_r= ',num2str(doppler_fq(14)*c/fq_rr(2)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,15)
polar(polarmatrix(57:58),lengthmatrix(57:58),'-*')
hold on
polar(polarmatrix(59:60),lengthmatrix(59:60),':*r')
title(['15^{th}. v_r= ',num2str(doppler_fq(15)*c/fq_rr(3)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,16)
polar(polarmatrix(61:62),lengthmatrix(61:62),'-*')
hold on
polar(polarmatrix(63:64),lengthmatrix(63:64),':*r')
```

```
title(['16^{th}. v_r= ',num2str(doppler_fq(16)*c/fq_rr(1)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,17)
polar(polarmatrix(65:66),lengthmatrix(65:66),'-*')
hold on
polar(polarmatrix(67:68),lengthmatrix(67:68),':*r')
title(['17^{th}. v_r= ',num2str(doppler_fq(17)*c/fq_rr(2)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,18)
polar(polarmatrix(69:70),lengthmatrix(69:70),'-*')
hold on
polar(polarmatrix(71:72),lengthmatrix(71:72),':*r')
title(['18^{th}. v_r= ',num2str(doppler_fq(18)*c/fq_rr(3)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,19)
polar(polarmatrix(73:74),lengthmatrix(73:74),'-*')
hold on
polar(polarmatrix(75:76),lengthmatrix(75:76),':*r')
title(['19^{th}. v_r= ',num2str(doppler_fq(19)*c/fq_rr(1)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,20)
polar(polarmatrix(77:78),lengthmatrix(77:78),'-*')
hold on
polar(polarmatrix(79:80),lengthmatrix(79:80),':*r')
title(['20^{th}. v_r= ',num2str(doppler_fq(20)*c/fq_rr(2)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,21)
polar(polarmatrix(81:82),lengthmatrix(81:82),'-*')
hold on
polar(polarmatrix(83:84),lengthmatrix(83:84),':*r')
title(['21^{th}. v_r= ',num2str(doppler_fq(21)*c/fq_rr(3)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,22)
polar(polarmatrix(85:86),lengthmatrix(85:86),'-*')
hold on
polar(polarmatrix(87:88),lengthmatrix(87:88),':*r')
title(['22^{th}. v_r= ',num2str(doppler_fq(22)*c/fq_rr(1)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,23)
polar(polarmatrix(89:90),lengthmatrix(89:90),'-*')
hold on
polar(polarmatrix(91:92),lengthmatrix(91:92),':*r')
title(['23^{th}. v_r= ',num2str(doppler_fq(23)*c/fq_rr(2)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,24)
polar(polarmatrix(93:94),lengthmatrix(93:94),'-*')
hold on
polar(polarmatrix(95:96),lengthmatrix(95:96),':*r')
```

```
title(['24^{th}. v_r= ',num2str(doppler_fq(24)*c/fq_rr(3)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,25)
polar(polarmatrix(97:98),lengthmatrix(97:98),'-*')
hold on
polar(polarmatrix(99:100),lengthmatrix(99:100),':*r')
title(['25^{th}. v_r= ',num2str(doppler_fq(25)*c/fq_rr(1)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,26)
polar(polarmatrix(101:102),lengthmatrix(101:102),'-*')
hold on
polar(polarmatrix(103:104),lengthmatrix(103:104),':*r')
title(['26^{th}. v_r= ',num2str(doppler_fq(26)*c/fq_rr(2)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,27)
polar(polarmatrix(105:106),lengthmatrix(105:106),'-*')
hold on
polar(polarmatrix(107:108),lengthmatrix(107:108),':*r')
title(['27^{th}. v_r= ',num2str(doppler_fq(27)*c/fq_rr(3)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,28)
polar(polarmatrix(109:110),lengthmatrix(109:110),'-*')
hold on
polar(polarmatrix(111:112),lengthmatrix(111:112),':*r')
title(['28^{th}. v_r= ',num2str(doppler_fq(28)*c/fq_rr(1)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,29)
polar(polarmatrix(113:114),lengthmatrix(113:114),'-*')
hold on
polar(polarmatrix(115:116),lengthmatrix(115:116),':*r')
title(['29^{th}. v_r= ',num2str(doppler_fq(29)*c/fq_rr(2)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)
subplot(5,6,30)
polar(polarmatrix(117:118),lengthmatrix(117:118),'-*')
hold on
polar(polarmatrix(119:120),lengthmatrix(119:120),':*r')
title(['30^{th}. v_r= ',num2str(doppler_fq(30)*c/fq_rr(3)/2),' m/s'])
hline = findobj(gca,'Type','line');
set(hline,'LineWidth',3)



mean(doppler_fq)
std(doppler_fq)
```

4) WR_pulse_com

```
% Weather Radar Signal Simulator, developed version
% Pulse compression
% Ulf Schroder
% August, 2005
%-----------------------------------------------------------------------

% Main program for The NPS/Marielle Gosset Weather Radar Signal Simulator.

%-----------------------------------------------------------------------
% DESCRIPTION
% Part I
% From Radar Data:
% - Generate Beam Resolution cell approximation
% - Generate Cube adding a margine to the Beam Resolution cell to allow for
% the rain drops to fall trough the through beam over time. The stretched
% pulse will make the resolution cell larger and divided into range bins
% according to specified input data. To get a resonable average the middle
% range bin will serve as the "average" range bin
%
% Part II
% From Rain Parameters:
% Generating the fundamental T Matrix concisting of
% - Diameter Vector and Number of drops in each diameter interval
%   using the Marshall-Palmer exponential approximation.
% - the backscatter RCSs for each Diameter size
% - the total number of drops for each size per m^3

% Part III
% Randomly place all drops in cube

% Part IV
% For every pulse the coherent Electric field return is calculated. Between
% pulses all drops are moved.

% Part V
% Calculate and plot results
%-----------------------------------------------------------------------

clear all
close all
clc

% Input Data
c=3e8;              % Speed of light [m/s]

% Radar Data
P_t=10e3;           % Transmitted power
G_dB=20;            % Maximum gain in dB
G=10^(G_dB/10);     % Maximum gain
angle_3db=0.20;     % 3 dB Beam angle [degrees], (half half power beamwidth)
pulsewidth=0.20;    % Pulsewidth [microsecond]
elevation=3;        % Elevation angle [degrees]
margine=5;          % margin for resolution cell[m]
azimuth=0;          % Azimuth angle [rad]
PRT_1=1/200;        % Pulse Repetition Time [seconds]
distance=0.5;       % Range from the radar to the resolution cell [Km]
wavelength=0.1;     % Wavelength [m]
frequency=c/wavelength; % Frequency [Hz]
```

```
%------------------------------------------------------------------------
%Introduced in Pulse Compression version
chirp_width=2e6;                        % Frequency step in [Hz]
n_steps=10;                             % Number of Frequency steps
fq_step=chirp_width/n_steps;            % Frequency step in [Hz]
d_range=c*pulsewidth*n_steps*1e-6/2/n_steps; % range bin due to pulse compression
num_runs=10;                            % Number of runs (pulse compressed)
range_bin=(distance*1000):d_range:(distance*1000+(n_steps-1)*d_range);
%------------------------------------------------------------------------

% Rain parameters
diam_low=0.05;        % Lowest diameter of Precipitation [mm]
diam_lim=1;           % Divider between low and mid [mm]
diam_mid=5;           % Divider between mid and high [mm]
diam_high=15;         % Highest diameter of Precipitation [mm]
resolution1=0.08;     % Lower diameter resolution [mm]
resolution2=0.3;      % Higher diameter resolution [mm]
resolution3=0.8;      % Highest diameter resolution [mm]
sort_approx=1;        % RCS approximation model; [1]=Rayleigh
kw=0.93;              % abs(K)^2 approx for water
rain_rate=50;         % Rain Rate [mm/h]
rain_anglel=0;        % degrees Rain Angle (Phi)
rain_angle2=0;        % degrees Rain Angle (Theta)
wind_vector=[-10;0;0];   % Wind speed vector (x,y,z) [m/s]
spread=1;             % Wind speed spread [m/s], (all directions)
spread_2=4;           % Drop velocity spread [m/s]
Norm=200;             % Normalization coefficient

% Directional cosines
u=sin(pi/2-elevation*pi/180)*cos(azimuth*pi/180);
v=sin(pi/2-elevation*pi/180)*sin(azimuth*pi/180);
w=cos(pi/2-elevation*pi/180);

%------------------------------------------------------------------------
% Part I, Generate the resolution volume
%------------------------------------------------------------------------
% Approximation of the resolution cell of the pulse circular box,
% for any azimuth angle
%------------------------------------------------------------------------
% INPUT
% - distance
% - angle_3dB
% - pulsewidth
% - elevation
% - margine
% - azimuth
% ------------------------------------------------------------------------
% OUTPUT
% - volume_resolution: computed volume of the resolution cell
% - volume_box: volume of the box where the drops are put
% - box: the 8 rectangular coordinates of the box
% - coord_vol_res: the coordinates of the resolution cell where the drops
% are taken.
%------------------------------------------------------------------------
```

```
        %-------------------------------------------------------------------------
        % Creating the circular beam resolution cell
        range_res=(c*pulsewidth*n_steps*1e-6)/2;                    ...


        % small angle approximation allows for following
        angle_rad=(angle_3db*pi)/180;                        ...
           % 3 dB beamwidth [rad]
        radius1=(tan(angle_rad)*(distance*1000));                ...
           % Radius of beam at R=distance
        radius2=(tan(angle_rad)*((distance*1000)+range_res));   ...
           % Radius of beam at R=distance + range resolution

        surface1=pi*radius1^2;
        surface2=pi*radius2^2;

        % volume of cone = base surface times height for all three
        volume_cone1=(surface1*(distance*1000))/3;
        volume_cone2=(surface2*((distance*1000)+range_res))/3;
        volume_resolution=(volume_cone2-volume_cone1);          % Beam Cone Volume

        % To assure right volume for averaging
        DR=1:10;
        range_res_ref=(c*pulsewidth*DR*1e-6)/2;                   ...
        radius2_ref=(tan(angle_rad)*((distance*1000)+range_res_ref));   ...
        surface2_ref=pi*radius2_ref.^2;
        volume_cone2_ref=(surface2_ref.*((distance*1000)+range_res_ref))/3;
        volume_resolution_ref=(volume_cone2_ref-[volume_cone1             volume_cone2_ref(1:end-1)]);
% Beam Cone Volume


        % computation of the resolution cell coordinates, spherical coordinates
        % elevation in radians
        elevation_rad=(elevation*pi)/180;                    % Elevation [rad]
        azimuth_rad=(azimuth*pi)/180;                        % Azimuth [rad]

        % Matrix used when comparing wheather a drop is inside or outside
        % beamwidth
        coord_vol_res=[distance*1000,(distance*1000)+range_res;elevation_rad-angle_rad,...
              elevation_rad+angle_rad;azimuth_rad-angle_rad,azimuth_rad+angle_rad];
        %-------------------------------------------------------------------------

        %-------------------------------------------------------------------------
        % Creating the box
        % computation of the box and its coordinates
        coin1=[azimuth_rad+angle_rad,elevation_rad+angle_rad,distance*1000];
        coin2=[azimuth_rad-angle_rad,elevation_rad+angle_rad,distance*1000];
        coin3=[azimuth_rad+angle_rad,elevation_rad-angle_rad,distance*1000];
        coin4=[azimuth_rad-angle_rad,elevation_rad-angle_rad,distance*1000];
        coin5=[azimuth_rad+angle_rad,elevation_rad+angle_rad,distance*1000+range_res];
        coin6=[azimuth_rad-angle_rad,elevation_rad+angle_rad,distance*1000+range_res];
        coin7=[azimuth_rad+angle_rad,elevation_rad-angle_rad,distance*1000+range_res];
        coin8=[azimuth_rad-angle_rad,elevation_rad-angle_rad,distance*1000+range_res];

        [x1,y1,z1]=sph2cart(coin1(1),coin1(2),coin1(3));
        [x2,y2,z2]=sph2cart(coin2(1),coin2(2),coin2(3));
```

```
[x3,y3,z3]=sph2cart(coin3(1),coin3(2),coin3(3));
[x4,y4,z4]=sph2cart(coin4(1),coin4(2),coin4(3));
[x5,y5,z5]=sph2cart(coin5(1),coin5(2),coin5(3));
[x6,y6,z6]=sph2cart(coin6(1),coin6(2),coin6(3));
[x7,y7,z7]=sph2cart(coin7(1),coin7(2),coin7(3));
[x8,y8,z8]=sph2cart(coin8(1),coin8(2),coin8(3));

xmax=max([x1,x2,x3,x4,x5,x6,x7,x8]);
xmin=min([x1,x2,x3,x4,x5,x6,x7,x8]);
ymax=max([y1,y2,y3,y4,y5,y6,y7,y8]);
ymin=min([y1,y2,y3,y4,y5,y6,y7,y8]);
zmax=max([z1,z2,z3,z4,z5,z6,z7,z8]);
zmin=min([z1,z2,z3,z4,z5,z6,z7,z8]);

% the box with its margin
coin_1=[xmin-margine,ymin-margine,zmin-margine];
coin_2=[xmin-margine,ymax+margine,zmin-margine];
coin_3=[xmax+margine,ymax+margine,zmin-margine];
coin_4=[xmax+margine,ymin-margine,zmin-margine];
coin_5=[xmin-margine,ymin-margine,zmax+margine];
coin_6=[xmin-margine,ymax+margine,zmax+margine];
coin_7=[xmax+margine,ymax+margine,zmax+margine];
coin_8=[xmax+margine,ymin-margine,zmax+margine];


box=[coin_1',coin_2',coin_3',coin_4',coin_5',coin_6',coin_7',coin_8'];

% calculate the volume of the box
volume_box=abs((xmax+margine-(xmin-margine))*(ymax+margine-(ymin-margine))*...
   (zmax+margine-(zmin-margine)));


cube=[coin_1',coin_2',coin_3',coin_4',coin_1',coin_5',coin_6',coin_2',...
      coin_6',coin_7',coin_3',coin_7',coin_8',coin_4',coin_8',coin_5'];
%------------------------------------------------------------------------

%------------------------------------------------------------------------
% Part II, Generate the fundamental T Matrix
%------------------------------------------------------------------------
% This part produces a matrix of all drops that contribute to the total
% reflectivity of the resolution cell
%------------------------------------------------------------------------
% First all sizes are created, with two vectors of different size resolution.
% The size vectors represents all diameters. Next the backscatter RCS for each
% size is computed.
% Using the Marshall-Palmer exponential approximation, the number of drops of
% each diameter is computed.
%------------------------------------------------------------------------
% INPUT
% - diam_low
% - diam_lim
% - diam_high
% - resolution1
% - resolution2
% - sort_approx
% - wavelength
% - Kw
% - rain_rate
%-----------------------------------------------------------------------
% OUTPUT
% - number_drops
% Matrix T:
% - first row all sizes
% - second row all the backscatter RCSs
% - third row the total number of drops for each size per m^3
% the columns start with the smallest size
%-----------------------------------------------------------------------
% Create dropsize vector

% Precipitaion Diameter vector (lowest<D<divider)
vector_size_1=diam_low:resolution1:diam_lim;

% Precipitaion Diameter vector (divider<D<mid)
vector_size_2=diam_lim:resolution2:diam_mid;

% Precipitaion Diameter vector (mid<D<highest)
vector_size_3=diam_mid:resolution3:diam_high;

% Precipitaion Diameter Vector
vector_size=[vector_size_1,vector_size_2,vector_size_3];

%-----------------------------------------------------------------------
% Create RCS matrix for all Rain drops of Precipitation Vector and for all
% frequencies.
fq_rr=frequency:fq_step:frequency+(n_steps-1)*fq_step;
rcs(1:n_steps,:)=((kw^2)*(pi^5)/(c^4)*fq_rr'.^4)*((vector_size*1e-3).^6);
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% The Marshall_Palmer approximation is used to create Precipitaion Diameter
% Distribution Vector (Drop Size Distribution N(D) Vector)
% The number of drops are scaled down by a factor (Norm) to improve the
% speed of the simulator
%-----------------------------------------------------------------------
lambda=4.1*(rain_rate^(-0.21));        %[mm-1]
number=8000*exp(-lambda*vector_size)/Norm;

vector_res1=ones(1,length(vector_size_1))*resolution1;
vector_res2=ones(1,length(vector_size_2))*resolution2;
vector_res3=ones(1,length(vector_size_3))*resolution3;

% delta(Diameter) dD
vector_res=[vector_res1,vector_res2,vector_res3];

% the approximate "true" number of drops N(D)dD vector
number=number.*vector_res;
%-----------------------------------------------------------------------

%-----------------------------------------------------------------------
% create matrix T without rcs
T=[vector_size;number];
%-----------------------------------------------------------------------
```

```
%------------------------------------------------------------------------
% Change matrix 'T' to matrix 'matrix_T_Round'
%------------------------------------------------------------------------
% This part computes the total number of drops by utilizing the resolution
% cell and the matrix_T_Round
% The only difference between the matrix_T_Round and T, is the fact that
% matrix_T_Round has the total number of drops on the third row and they
% are rounded of to nearest integer value.
%------------------------------------------------------------------------

matrix_T_Round=T;
matrix_T_Round(2,:)=round(matrix_T_Round(2,:)*volume_box);

% Keep track of errors due to round off
diff=T(2,:)*volume_box-matrix_T_Round(2,:);
error_rcs=diff*rcs'*Norm/volume_box*sum(volume_resolution_ref)/n_steps;
error_Z=error_rcs/sum(volume_resolution_ref)/n_steps*c^4./fq_rr.^4/pi^5/...
   abs(kw)^2/1e-18;

%------------------------------------------------------------------------
% Part III, Initial positioning of the drops within the volume
%------------------------------------------------------------------------
% This part generates the initial position of the drops in the volume of
% the box.
% The positions are recorded in the file "sizexx.mat' taking into account
% the azimuth angle
%------------------------------------------------------------------------
% INPUT
% - box
% - matrix_T_Round
% - elevation
% - azimuth
%------------------------------------------------------------------------
% OUTPUT
% All files including the sizes.  Each contains the positions of all drops
% for a given diameter.  The data recoreded as follows:
% - radial distance r
% - elevation phi
% - azimuth theta
% - angle with respect to beam center.  This parameter will be
% used to identify drops which are in the beam.
%------------------------------------------------------------------------

% Randomly put the calculated number of drops of each diameter in the box
number_size=length(matrix_T_Round(1,:));

for i=1:number_size

   % select number of drops of certain Diameter
         drops=matrix_T_Round(2,i);

   % create as many positions on the x axis as # drops per size
         elements_x=rand(1,drops);

   % put them in the range of interest
```

```
    % Random placement x for every drop
          position_x=box(1,1)+(box(1,3)-box(1,1))*elements_x;

    % create as many positionson the y axis as # drops per size
          elements_y=rand(1,drops);

    % put them in the range of interest
    % Random placement y for every drop
          position_y=box(2,1)+(box(2,2)-box(2,1))*elements_y;

    % create as many positions on the z axis as # drops per size
          elements_z=rand(1,drops);

    % put them in the range of interest
    % Random placement z for every drop
          position_z=box(3,1)+(box(3,5)-box(3,1))*elements_z;

    % matrix of results
    % Creating a drop Matrix using spherical coordinates
          [theta,phi,r]=cart2sph(position_x,position_y,position_z);

    % Calculating the range, phi, theta and angle to every drop with
    % reference to phi_0 and theta_0. Used to estimate weighted power return.
    vector_azimuth=ones(1,drops)*((azimuth*pi)/180);
    vector_elevation=ones(1,drops)*((elevation*pi)/180);
    [xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);
    range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);

    % approximation
    angle=atan(range./r);

    % Saving the drop positions in size%d.mat
    % matrixpos
    matrixpos=[r;phi;theta;angle];

    save(sprintf('size%d.mat',i),'matrixpos');

end
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% Part IV, Calculate Coherent Electric field return.
%-------------------------------------------------------------------------
% For every pulse the coherent Electric field return is calculatet. Between
% pulses all drops are moved.
%
% Run the simulation for specified number of pulses
%-------------------------------------------------------------------------

% Zero all vectors and matrices that are going to be used
phase_vector=[];
sum_E=zeros(num_runs,n_steps);
weighted_E=zeros(num_runs,n_steps);
running_Z=[];

% Initiate reference variables
```

```
p=0;                     % Used to separate runs

for runs=1:num_runs

  p=p+1;
  hh=0;                    % Used to separate frequencies

  for fq_new=frequency:fq_step:frequency+(n_steps-1)*fq_step

    hh=hh+1;
    k=2*pi*fq_new/c;

    disp(sprintf('calculate the power-return of pulse %d\n',p));

    %-------------------------------------------------------------------
    % computes the (weighted) I and Q return
    %-------------------------------------------------------------------
              % computes the total number of drops
              %-------------------------------------------------------------------
              % INPUT
              % - Matrix T
              % - wavelength
              % - angle_3dB
              % - distance
              % - coord_vol_res
              % - number_pulses
              %-------------------------------------------------------------------
              % OUTPUT
              % - total_E: the received complex Electric field)
              % - power: the power
              % - voltage: the voltage
              % - drops_beam: the number of drops in the beam
              %-------------------------------------------------------------------
              % Only the drops which are in the resolution cell are selected for the
              % computation
              %-------------------------------------------------------------------

    number_size=length(matrix_T_Round(1,:));
    total_E=0;
    total_phase=0;
    new_E=0;

    for i=1:number_size

      % opening the file of registered data matrixpos.
      load(sprintf('size%d.mat',i));

      % removing drops that are outside the resolution cell
      number_drops_per_size=length(matrixpos(1,:));
      dropmatrix=[];

      % Filling the drop matrix with all drops inside the beam
      % resolution volume
      for d=1:number_drops_per_size
        if ((matrixpos(4,d)<=(angle_3db*pi/180))&(matrixpos(1,d)>=...
            range_bin(hh))&(matrixpos(1,d)<=range_bin(hh)+d_range));
```

```
                  dropmatrix=[dropmatrix,matrixpos(:,d)];
               end
            end

            %-------------------------------------------------------------------
            % Convert to cartesian coordinates
            if isempty(dropmatrix)==0
               [position_x,position_y,position_z]=sph2cart(dropmatrix(3,:),...
                  dropmatrix(2,:),dropmatrix(1,:));

               % Create comparison vectors
               vector_azimuth=ones(1,length(dropmatrix(1,:)))*((azimuth*pi)/180);
               vector_elevation=ones(1,length(dropmatrix(1,:)))*((elevation*pi)/180);


               % Create a phase vector for the drops of current Diameter
               r_phase=position_x.*u+position_y.*v+position_z.*w;
               phase=exp((2*j*k).*r_phase);

               % Sum the contributions to the E-field
               total_phase=total_phase+sum(phase);
               total_E=total_E+sum(sqrt(rcs(hh,i)).*phase);

               %---------------------------------------------------------------
               % ceate weighted power return, concidering range (r), theta, phi
               % with reference to boresight and radar parameters
               E_weighted=sqrt(P_t)*G*c/fq_rr(hh)/(sqrt(4*pi)^3)./dropmatrix(1,:).^2.*...
                  (exp(-4*log(2).*((dropmatrix(3,:)-vector_azimuth).^2/...
                  (2*angle_3db*pi/180)^2+((dropmatrix(2,:)-
vector_elevation).^2/(2*angle_3db*pi/180)^2))));
               new_E=new_E+sum(E_weighted.*(sqrt(rcs(hh,i)).*phase));

            end
         end

         %-------------------------------------------------------------------
         % add to E-field and phase vector
         sum_E(p,hh)=total_E;
         weighted_E(p,hh)=new_E;
         phase_vector(2*p-1)=ANGLE(total_phase);
         %-------------------------------------------------------------------

      end
      %-------------------------------------------------------------------
            % This part simulates the movement of the drops and gives new positions.
      % First using the Doppler PRT (PRT_1)
            %-------------------------------------------------------------------
            % INPUT
            % - matrix_T_Round
            % - PRT
            % - azimuth
            % - elevation
            % - rain_angle1
            % - rain_angle2
            % - diam_lim
      % - resolution1
% - resolution2
        % - wind_vector
        %------------------------------------------------------------------
        % OUTPUTS
        % Output files containing sizes with new positions
        % structured the same way as in previous
        % there is also an output file for speed (one for each size) for the pulse
        % pair.
        %------------------------------------------------------------------

number_size=length(matrix_T_Round(1,:));

% adding a Gaussian random term to make the wind speed spread
if wind_vector(1)~=0
   new_wind_vector(1)=wind_vector(1)*normrnd(1,abs(spread/max(wind_vector(1))));
else
   new_wind_vector(1)=0;
end
if wind_vector(2)~=0
   new_wind_vector(2)=wind_vector(2)*normrnd(1,abs(spread/max(wind_vector(2))));
else
   new_wind_vector(2)=0;
end
if wind_vector(3)~=0
   new_wind_vector(3)=wind_vector(3)*normrnd(1,abs(spread/max(wind_vector(3))));
else
   new_wind_vector(3)=0;
end

for i=1:number_size

   % opening of data file registered in matrixpos.
   load(sprintf('size%d.mat',i));

   % total number of drops
   drops=matrix_T_Round(2,i);

   % transforming to rectangular coordinates
   [x,y,z]=sph2cart(matrixpos(3,:),matrixpos(2,:),matrixpos(1,:));

   diameter=matrix_T_Round(1,i);

   %---------------------------------------------------------------
   % Drop fall speed depends on drop diameter in the z axis,
   % Atlas-Ulbrich approximation is used
                % Approximation is valid in the diameter range 5*e-4, 5*e-3
                %---------------------------------------------------------------

   wtmax=386.6*((matrix_T_Round(1,i)*0.001)^0.67);

   velocityz=(wtmax*(ones(1,drops)))*cos((rain_anglel*pi)/180);
   velocityx=(wtmax*(ones(1,drops)))*sin((rain_anglel*pi)/180)*...
      cos((rain_angle2*pi)/180);
   velocityy=(wtmax*(ones(1,drops)))*sin((rain_anglel*pi)/180)*...
      sin((rain_angle2*pi)/180);
```

```
    %-----------------------------------------------------------------
    %-----------------------------------------------------------------
%
%         % adding another Gaussian random term to make the wind speed
%         % spread between different drops sizes
%       if new_wind_vector(1)~=0
%           newest_wind_vector(1)=new_wind_vector(1)*normrnd(1,abs(spread...
%               /max(wind_vector(1))));
%       else
%           newest_wind_vector(1)=0;
%       end
%       if new_wind_vector(2)~=0
%           newest_wind_vector(2)=new_wind_vector(2)*normrnd(1,abs(spread...
%               /max(wind_vector(2))));
%       else
%           newest_wind_vector(2)=0;
%       end
%       if new_wind_vector(3)~=0
%           newest_wind_vector(3)=new_wind_vector(3)*normrnd(1,abs(spread...
%               /max(wind_vector(3))));
%       else
%           newest_wind_vector(3)=0;
%       end
%
%       % If spread within pulse
%       velocityxt=newest_wind_vector(1)-velocityx;
%       velocityyt=newest_wind_vector(2)-velocityy;
%       velocityzt=newest_wind_vector(3)-velocityz;
%
%
    %-----------------------------------------------------------------
    %-----------------------------------------------------------------
    % If spread for all drops regardless of size
    newest_wind_vector=[];


      % adding Gaussian random term to make the wind speed
      % spread between drops
    if new_wind_vector(1)~=0
        newest_wind_vector(1,:)=new_wind_vector(1)*...
            normrnd(1,abs(spread_2/max(new_wind_vector(1))),1,drops);
    else
        newest_wind_vector(1,:)=0;
    end
    if new_wind_vector(2)~=0
        newest_wind_vector(2,:)=new_wind_vector(2)*...
            normrnd(1,abs(spread_2/max(new_new_wind_vector(2))),1,drops);
    else
        newest_wind_vector(2,:)=0;
    end
    if new_wind_vector(3)~=0
        newest_wind_vector(3,:)=new_wind_vector(3)*...
            normrnd(1,abs(spread_2/max(new_wind_vector(3))),1,drops);
    else
        newest_wind_vector(3,:)=0;
        end

        % If spread within pulse
        velocityxt=newest_wind_vector(1,:)-velocityx;
        velocityyt=newest_wind_vector(2,:)-velocityy;
        velocityzt=newest_wind_vector(3,:)-velocityz;

        %-----------------------------------------------------------------
        %-----------------------------------------------------------------

        % speed registration
        m_velocity=[velocityxt;velocityyt;velocityzt];
        save(sprintf('velocity%d.mat',i),'m_velocity');

        deplacementx=velocityxt*PRT_1;
        deplacementy=velocityyt*PRT_1;
        deplacementz=velocityzt*PRT_1;

        position_x=x+deplacementx;
        position_y=y+deplacementy;
        position_z=z+deplacementz;

        % transformation into spherical coordinates
        [theta,phi,r]=cart2sph(position_x,position_y,position_z);

        % computation of new angles
        vector_azimuth=ones(1,drops)*(azimuth*pi)/180;
        vector_elevation=ones(1,drops)*(elevation*pi)/180;
        [xf,yf,zf]=sph2cart(vector_azimuth,vector_elevation,r);

        range=sqrt((position_x-xf).^2.+(position_y-yf).^2.+(position_z-zf).^2);
        % approximation
        angle=atan(range./r);

        % matrixpos
        matrixpos=[r;phi;theta;angle];

        % save to file, matrixpos
                    s=sprintf('size%d.mat',i);
              save(s, 'matrixpos');

   end

end

%----------------------------------------------------------------------

%----------------------------------------------------------------------
% calculate RCS, Z and Power return for every pulse including error. The
% reflectivity is caculated for each volume bin and then summed and
% averaged.
%----------------------------------------------------------------------
for o=1:num_runs
   running_Z(o,:)=(Norm*abs(sum_E(o,:)).^2)*c^4./fq_rr(o)^4/pi^5/...
       abs(kw)^2/volume_resolution_ref(o)/1e-18+error_Z(o);
end
```

```
running_Z=sum(running_Z,2)'/n_steps;        % Transform into a vector

%------------------------------------------------------------------------
% Adding the error to the weighted_E estimations
%------------------------------------------------------------------------
error_power=error_rcs*P_t*G^2*c^2./fq_rr.^2/pi^2*(2*angle_3db*pi/180)^2*...
    (c*pulsewidth*1e-6)/1024/log(2)/pi^2/(distance*1e3)^2;

for o=1:num_runs
    power_return(o,:)=Norm*abs(weighted_E(o,:)).^2+error_power(o);
end

power_return=sum(power_return,2)'/n_steps;   % Transform into a vector

% to use to estimate zeroth moment
for o=1:n_steps
    power_return_ref_matrix(:,o)=Norm*abs(weighted_E(:,o)).^2+error_power(o);
end

power_return_ref=sum(power_return_ref_matrix,1)/num_runs;   % Transform into a vector

%------------------------------------------------------------------------
% calculate average Power and Z
%------------------------------------------------------------------------
Z_avg=sum(running_Z)/p;
Z_avg_plot=ones(1,p)*Z_avg;

power_return_avg=sum(power_return)/p;            % [W]
power_return_avg_plot=ones(1,p)*power_return_avg;

number_vector=1:p;

%------------------------------------------------------------------------
% Calculate references
% Integral form of Z
%------------------------------------------------------------------------
ii=0:100/1000:100;
zz=ii.^6.*8000.*exp(-4.1.*rain_rate.^(-.21).*ii);
real_Z=ones(1,p)*10*log10(trapz(ii,zz));

%------------------------------------------------------------------------
% Evaluate average Power return to make Z estimation including error
%------------------------------------------------------------------------
% To make the estimate with the correct frequency, the power return Matrix
% is used. For the plot the result must me summed and divided by the number
% of samples used to average (which means over both frequency and pulse).

Z_estimation=power_return_ref_matrix*((fq_rr.^2).*((c./fq_rr).^4))'/P_t/G^2/c^2*...
    pi^2/(2*angle_3db*pi/180)^2/(c*pulsewidth*1e-
6)*1024*log(2)*(distance*1e3+range_res/2)^2/...
    ((kw^2)*(pi^5))/1e-18;
Z_estimation_plot=ones(1,p)*10*log10(sum(Z_estimation)/n_steps/num_runs);


%------------------------------------------------------------------------
```

```
% Part V, Plot of results
%-----------------------------------------------------------------------

figure(1);
kl=1:p;
number_vector=kl.*PRT_1*1000;

text_string=['Power Return for all ',num2str(p),' pulse trains.'...
        ,' Frequency stepped from ',num2str(fq_rr(1)/1e9),' GHz to ',...
        num2str(fq_rr(n_steps)/1e9),' GHz with ',num2str(fq_step/1e6)...
        ,' MHz steps.'];
% also converting to dBm
plot(number_vector,10*log10(power_return)+30,'-xb', number_vector,10*log10...
    (power_return_avg_plot)+30,'--r');
title(text_string);
xlabel('Sample time [ms]');
ylabel('Power [dBm]');
legend('Power return', 'Average Power return',0)
grid on

figure (2)
plot(number_vector,10*log10(abs(running_Z)),'-xb',number_vector,10*...
    log10(abs(Z_avg_plot)),'--r',number_vector,real_Z,'--k',...
    number_vector,Z_estimation_plot,'--b');
title('Estimated Z per pulse');
xlabel('Sample time');
ylabel('Z [dBZ]');
legend('Simulation Z, (instantaneous)', 'Estimated Z, (average return)','Z=\int D^6N(D)dD',...
    'Estimated Z, Zeroth moment',0)
grid on
```

LIST OF REFERENCES

[1] M. Gosset, J. Nicol and A. Sanchez, "Development of a weather radar signal simulator to revisit the problem of signal statistics and measurement errors," retrieved 2005-08-22 from http://www.bom.gov.au/bmrc/basic/old_events/hawr6/errors_radar_measurements/gosset1_develop_signal.pdf

[2] J.S. Marshall and W. Hitschfeld, "Interpretation of the fluctuating echo from randomly distributed scatterers. Part I" *Can J Phys,* vol. Vol. 31, pp. 962-994, 1953.

[3] R.J. Doviak and D.S. Zrnić, *Doppler radar and weather observations,* San Diego: Academic Press, 1993.

[4] J.B. Sandifer, "Meteorological measurements with a MWR-05XP Phased Array Radar," *Naval Postgraduate School,* March 2005.

[5] M.I. Skolnik, *Introduction to radar systems,* Boston: McGraw Hill, 2001.

[6] D.C. Jenn, *Radar and laser cross section engineering,* Reston, Va.: American Institute of Aeronautics and Astronautics, 2005.

[7] F.E. Nathanson, *Radar design principles; signal processing and the environment,* New York: McGraw-Hill Book Co, 1969.

[8] J.R. Probert-Jones, "The radar equation in meteorology," *Q. J. R. Meteorol. Soc.,* vol. Vol. 88, pp. 485-495, July 1962.

[9] H.R. Pruppacher, "A wind tunnel investigation of the circulation and shape of water drops falling at terminal velocity in air," *Q. J. R. Meteorol. Soc.,* vol. Vol. 96, pp. 247-256, December 1970.

[10] G.M. McFarquhar, "The effect of raindrop clustering on collision-induced break-up of raindrops," *Q. J. R. Meteorol. Soc.,* pp. 2169-2190, 2004.

[11] J.S. Marshall and W.M. Palmer, "The Distribution of Raindrops with Size," *J. Meteorol.,* vol. Vol. 5, pp. 165-166, August 1948.

[12] D. Atlas and C.W. Ulbrich, "Path- and Area-Integrated Rainfall Measurement by Microwave Attenuation in the 1-3 cm Band." *J. Applied Meteorology,* vol. Vol. 16, pp. 1322-1331, December 1977.

[13] F.T. Ulaby, *Fundamentals of applied electromagnetics,* Upper Saddle River, NJ: Pearson, 2004.

[14] V.N. Bringi and V. Chandrasekar, *Polarimetric Doppler weather radar : principles and applications,* Cambridge; New York: Cambridge University Press, 2001.

[15] W.L. Stutzman and G.A. Thiele, *Antenna theory and design,* New York: J. Wiley, 1998.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Chairman, Code IW
   Department of Information Sciences
   Naval Postgraduate School
   Monterey, California

4. Prof. Jeffrey B. Knorr
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

5. Prof. Phillip E. Pace
   Center for Joint Services Electronic Warfare
   Naval Postgraduate School
   Monterey, California

6. Prof. David C. Jenn
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California