Naval Research Laboratory

Washington, DC 20375-5320

NRL/MR/5753--17-9744

Application of a Statistical Linear Time-Varying System Model of High Grazing Angle Sea Clutter for Computing Interference Power

Corey D. Cooke

Applied Technology, Inc.
King George, Virginia

Donald E. Jarvis

Advanced Techniques Branch
Tactical Electronic Warfare Division

December 08, 2017

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
08-12-2017	Memorandum Report	May 2017 - August 2017

4. TITLE AND SUBTITLE

Application of a Statistical Linear Time-Varying System Model of High Grazing Angle Sea Clutter for Computing Interference Power

5a. CONTRACT NUMBER
N00173-17-D-2003/N0024-5-C-5346

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
NFE 2007-004

6. AUTHOR(S)

Corey D. Cooke,* and Donald E. Jarvis

5d. PROJECT NUMBER

5e. TASK NUMBER
N00173-17-F-2010

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5753--17-9744

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Raytheon Company
1847 West Main Road
Portsmouth, RI 02871

Naval Sea Systems Command
1333 Isaac Hull Avenue SE
Washington Navy Yard, DC 20376

10. SPONSOR / MONITOR'S ACRONYM(S)

RTN/NAVSEA

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

*Applied Technology, Inc., 5200 Potomac Dr., King George, VA 22485

14. ABSTRACT

In this work we further develop a linear time-varying system model of sea clutter returns begun in a previous NRL report. We further refine the model to account for the periodic nature of the pulse ambiguity function and refine the computation of the Doppler spectrum to account for wave motion. We also present analytic formulae for the clutter autocorrelation with the hope that these mathematical tools can be useful in the design of clutter mitigation signal processing techniques.

15. SUBJECT TERMS

Time-frequency analysis; Sea clutter modeling; Linear time-varying systems.

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Corey D. Cooke
Unclassified Unlimited	Unclassified Unlimited	Unclassified Unlimited	Unclassified Unlimited	37	**19b. TELEPHONE NUMBER** *(include area code)* (202) 767-6351

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

CONTENTS

FIGURES

TABLES

EXECUTIVE SUMMARY

In this work we further develop a linear time-varying system model of sea clutter returns begun in a previous NRL report [1]. We present analytic formulae for the clutter autocorrelation function, which can be used in the design and analysis of clutter mitigation signal processing techniques. We further refine the model to account for the periodic nature of the pulse ambiguity function and refine the computation of the Doppler spectrum to account for wave motion as well as improve its numerical integrability.

We apply these tools to demonstrate how the effectiveness of a candidate waveform in the presence of clutter may be evaluated.

APPLICATION OF A STATISTICAL LINEAR TIME-VARYING SYSTEM MODEL OF HIGH GRAZING ANGLE SEA CLUTTER FOR COMPUTING INTERFERENCE POWER

1. INTRODUCTION

Statistical linear time-varying (LTV) system theory, particularly in the case of wide-sense stationary uncorrelated scattering (WSSUS) was first put on firm mathematical footing by Bello [2], but was hinted at by other authors that preceded him, such as Price and Green [3, 4]. WSSUS channel models have been widely used in communications [5–7], but have only occasionally seen use in radar circles [8–10] despite originating largely in this community. In this work we further refine a WSSUS model for a sea clutter scattering channel developed by the authors [1] and demonstrate how it can be used to evaluate waveform performance in the presence of clutter.

Let the single-polarization response $y(t)$ of a monostatic radar scattering channel to input signal $x(t)$ be given by the superposition integral:

$$y(t) = \int_{-\infty}^{\infty} h(\tau, t)x(t-\tau)d\tau \tag{1}$$

where $h(\tau, t)$ is the time-varying impulse response. Note that if there is no relative motion in the channel between the radar and the scatterers, $h(\tau, t) = h(\tau)$, and the system becomes a linear time-invariant (LTI) system whose output is given by the convolution of x and h.

In applying this model to a radar system, we use the following assumptions:

- The LTV impulse response $h(\tau, t)$ is only valid over a single coherent processing interval (CPI)
- Over a single CPI, the range to target(s) is roughly constant
- The lag variable τ represents downrange delay
- The time variable t represents the change in the channel over time

Variation in h with respect to t represents fluctuation in the signal amplitude due to relative motion between the radar and the scatterers and, subsequently, produces a Doppler shift in the received signal that is not present if the system is LTI.

The other key assumption of a WSSUS model is that the channel is random, and it is usually assumed to have Gaussian statistics in both the in-phase and quadrature channels. If the channel consists only of clutter, this corresponds to the Rayleigh amplitude model commonly used for high-grazing angle clutter,

Manuscript approved November 6, 2017.

which result derives from the assumption that a large number of independent scatterers are contained in a single resolution cell [11].

Note that due to the linearity of (1), if multiple scatterers are present, the overall impulse response is given by:

$$h(\tau, t) = h_1(\tau, t) + h_2(\tau, t) + \ldots, \tag{2}$$

where h_i represent the impulse responses of individual scatterers.

2. AUTOCORRELATION FUNCTION

Knowledge of the clutter autocorrelation function allows one to predict the effectiveness of signal processing techniques such as space-time array processing (STAP) [12], therefore being able to model it effectively will be invaluable in the analysis and design of future signal processing algorithms. The autocorrelation function of the random channel $h(\tau, t)$ is given by [5]:

$$\mathbb{E}\left[h^*(\tau, t)h(\tau', t + \Delta t)\right] = A(\tau, \Delta t)\delta(\tau - \tau'), \tag{3}$$

which implicitly states that the channel is wide-sense stationary (WSS) in t, and it undergoes uncorrelated scattering (US) in τ, hence the name WSSUS. The function $A(\tau, \Delta t)$ is known simply as the WSSUS autocorrelation function.

The scattering function, $S(\tau, \rho)$, which is a representation of the channel gain as a function of delay and Doppler shift, is simply the Fourier transform of $A(\tau, \Delta t)$ along the Δt axis:

$$S(\tau, \rho) = \int_{-\infty}^{\infty} A(\tau, \Delta t)e^{-j2\pi\rho\Delta t}d\Delta t. \tag{4}$$

Note that if multiple scatterers are present, as in the case of (2), and if those scatterers are statistically independent, it can be shown that the overall scattering function is given by:

$$S(\tau, \rho) = S_1(\tau, \rho) + S_2(\tau, \rho) + \ldots, \tag{5}$$

where S_i represent the scattering functions of the individual scatterers. This superposition property also applies to the autocorrelation functions (3), provided the target responses are independent.

In the author's previous work [1], it was shown that the WSSUS autocorrelation function $A(\tau, \Delta t)$ for a sea clutter scattering channel with no other targets is given by:

$$A(\tau, \Delta t) = \frac{\lambda^2\sigma^0 c}{(4\pi)^3(\frac{1}{2}c\tau)^3} \cdot \frac{1}{2}\int_{\langle 2\pi\rangle} G^2(\phi, \theta)e^{j2\pi\rho'(\phi,\theta)\Delta t}d\phi, \tag{6}$$

where

$$\rho'(\phi,\theta) = \rho_x \cos\theta\cos\phi + \rho_z \sin\theta \tag{7}$$

is the observed Doppler shift as a function of azimuth angle ϕ and depression angle θ for an airborne radar moving at velocity $\mathbf{v} = [v_x, 0, v_z]^T$, λ is the carrier wavelength, c is the speed of light, $\rho_x = 2v_x/\lambda$, $\rho_z = 2v_z/\lambda$, and $\sigma^0 = \sigma^0(\alpha(\tau))$ is the normalized radar cross section (NRCS) of the clutter as a function of grazing angle α, which itself is a function of τ. We will assume an antenna power pattern $G(\phi,\theta)$ given by:

$$G(\phi,\theta) = D\left|\mathrm{sinc}\left(\frac{W}{B}(\phi-\phi_a)\right)\right|\left|\mathrm{sinc}\left(\frac{W}{B}(\theta-\theta_a)\right)\right| \tag{8}$$

where D is the antenna directivity, $W = 1.20671$ rad is a normalization constant, B is the 3 dB beamwidth in radians, and (ϕ_a, θ_a) are the orientation angles of the main beam. We can approximate one of the sinc factors using the Dirichlet kernel to facilitate computation of the integral in (6) as follows:

$$\left|\mathrm{sinc}\left(\frac{W}{B}(\phi-\phi_a)\right)\right| \approx \left|\frac{\sin(\pi W(\phi-\phi_a)/B)}{(2\pi W/B)\sin((\phi-\phi_a)/2)}\right|. \tag{9}$$

If we define the quantity N:

$$N = 2\pi\frac{W}{B} \tag{10}$$

then we can express (9) as follows:

$$\left|\mathrm{sinc}\left(\frac{W}{B}(\phi-\phi_a)\right)\right| \approx \left|\frac{\sin(N(\phi-\phi_a)/2)}{N\sin((\phi-\phi_a)/2)}\right|, \tag{11}$$

which, if $N \gg 1$, then to good approximation we can round N to the nearest integer and express (11) by:

$$\left|\frac{\sin(N(\phi-\phi_a)/2)}{N\sin((\phi-\phi_a)/2)}\right| = \left|\frac{1}{N}\sum_{m=0}^{N-1} e^{jm(\phi-\phi_a)}\right|. \tag{12}$$

Now we can solve the integral in (6) as follows:

$$\begin{aligned}&\int_{\langle 2\pi\rangle} G^2(\phi,\theta) e^{j2\pi\rho'(\phi,\theta)\Delta t} d\phi \\ &= D^2\,\mathrm{sinc}^2\left(\frac{W}{B}(\theta-\theta_a)\right) e^{j2\pi\rho_z\sin\theta\Delta t}\left[\int_{\langle 2\pi\rangle}\left|\frac{1}{N}\sum_{m=0}^{N-1} e^{jm(\phi-\phi_a)}\right|^2 e^{j2\pi\rho_x\cos\theta\cos\phi\Delta t} d\phi\right].\end{aligned} \tag{13}$$

Note that if we expand the first factor in the integrand in a Fourier series:

$$\left|\frac{1}{N}\sum_{m=0}^{N-1} e^{jm(\phi-\phi_a)}\right|^2 = \sum_{k=-\infty}^{\infty} c_k e^{jk\phi}, \tag{14}$$

where

$$c_k = \begin{cases} \left(\frac{1}{N} - \frac{|k|}{N^2}\right) e^{-jk\phi_a}, & |k| \leq N \\ 0, & \text{else} \end{cases}, \tag{15}$$

and then substitute the Jacobi-Anger expansion [13] for the second factor of the integrand in (13):

$$e^{j(2\pi\rho_x \cos\theta\Delta t)\cos\phi} = \sum_{n=-\infty}^{\infty} j^n J_n(2\pi\rho_x \cos\theta\Delta t) e^{jn\phi}, \tag{16}$$

where $J_n(x)$ is the n^{th}-order Bessel function of the first kind, we can solve the integral (13) as follows:

$$\begin{aligned}
\int_{\langle 2\pi\rangle} \left|\frac{1}{N}\sum_{m=0}^{N-1} e^{jm(\phi-\phi_a)}\right|^2 e^{j2\pi\rho_x \cos\theta\cos\phi\Delta t} d\phi &= \int_{\langle 2\pi\rangle} \sum_k c_k e^{jk\phi} \sum_n j^n J_n(2\pi\rho_x\cos\theta\Delta t) e^{jn\phi} d\phi \\
&= \sum_k \sum_n c_k j^n J_n(2\pi\rho_x\cos\theta\Delta t) \int_{\langle 2\pi\rangle} e^{j(k+n)\phi} d\phi \\
&= \sum_k \sum_n c_k j^n J_n(2\pi\rho_x\cos\theta\Delta t) 2\pi\delta[k+n] \\
&= 2\pi \sum_n c_{-n} j^n J_n(2\pi\rho_x\cos\theta\Delta t) \\
&= 2\pi \sum_{n=-N}^{N} \left(\frac{1}{N} - \frac{|n|}{N^2}\right) e^{jn\phi_a} j^n J_n(2\pi\rho_x\cos\theta\Delta t) \\
&= \frac{2\pi}{N} \sum_{n=-N}^{N} \left(1 - \frac{|n|}{N}\right) e^{j(\phi_a+\pi/2)n} J_n(2\pi\rho_x\cos\theta\Delta t)
\end{aligned} \tag{17}$$

The correlation function (6) is then given by:

$$\begin{aligned}
A(\tau,\Delta t) &= \frac{\lambda^2\sigma^0 c}{(4\pi)^3(\frac{1}{2}c\tau)^3} \cdot \frac{1}{2}\int_{\langle 2\pi\rangle} G^2(\phi,\theta) e^{j2\pi\rho'(\phi,\theta)\Delta t} d\phi \\
&= \frac{\lambda^2\sigma^0 c\pi D^2}{(4\pi)^3(\frac{1}{2}c\tau)^3 N} \cdot \text{sinc}^2\left(\frac{W}{B}(\theta-\theta_a)\right) \cdot e^{j2\pi\rho_z \sin\theta\Delta t} \cdot \sum_{n=-N}^{N} \left(1 - \frac{|n|}{N}\right) e^{j(\phi_a+\pi/2)n} J_n(2\pi\rho_x\cos\theta\Delta t).
\end{aligned} \tag{18}$$

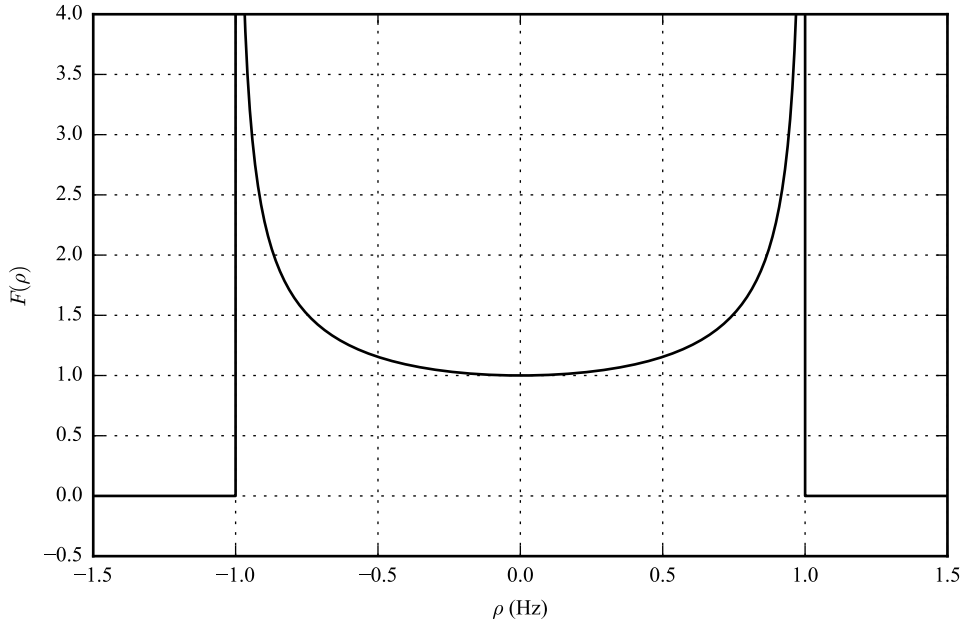

Fig. 1: Unsmoothed Clarke spectrum with $G = 1$, $\rho_x \cos\theta = 1$ Hz, and $\rho_z \sin\theta = 0$ Hz.

3. DOPPLER SMOOTHING

3.1 Frequency Domain

From the author's previous report [1], it was determined that the clutter scattering function $S(\tau, \rho)$ was given by:

$$S(\tau,\rho) = \begin{cases} \frac{\lambda^2 c}{(4\pi)^3} \cdot \frac{\sigma^0(\alpha(\tau))}{(\frac{1}{2}c\tau)^3} \cdot \frac{G^2(\phi(\tau,\rho),\theta(\tau))}{\sqrt{(\rho_x \cos(\theta(\tau)))^2 - (\rho - \rho_z \sin(\theta(\tau)))^2}}, & |\rho - \rho_z \sin(\theta(\tau))| < \rho_x \cos(\theta(\tau)) \\ 0, & \text{else} \end{cases} . \quad (19)$$

This is a generalization of the famous result given by Clarke in his landmark paper [14]. However, this result was derived assuming the antenna pattern $G(\phi, \theta)$ was an even function of ϕ. For the most general case, i.e. when $\phi_a \neq 0$, the scattering function is given by:

$$S(\tau,\rho) = \begin{cases} \frac{\lambda^2 c\, \sigma^0(\alpha(\tau))}{(4\pi)^3(\frac{1}{2}c\tau)^3} \cdot \frac{1}{2} \frac{G^2(\phi(\tau,\rho),\theta(\tau)) + G^2(-\phi(\tau,\rho),\theta(\tau))}{\sqrt{(\rho_x \cos(\theta(\tau)))^2 - (\rho - \rho_z \sin(\theta(\tau)))^2}}, & |\rho - \rho_z \sin(\theta(\tau))| < \rho_x \cos(\theta(\tau)) \\ 0, & \text{else} \end{cases} . \quad (20)$$

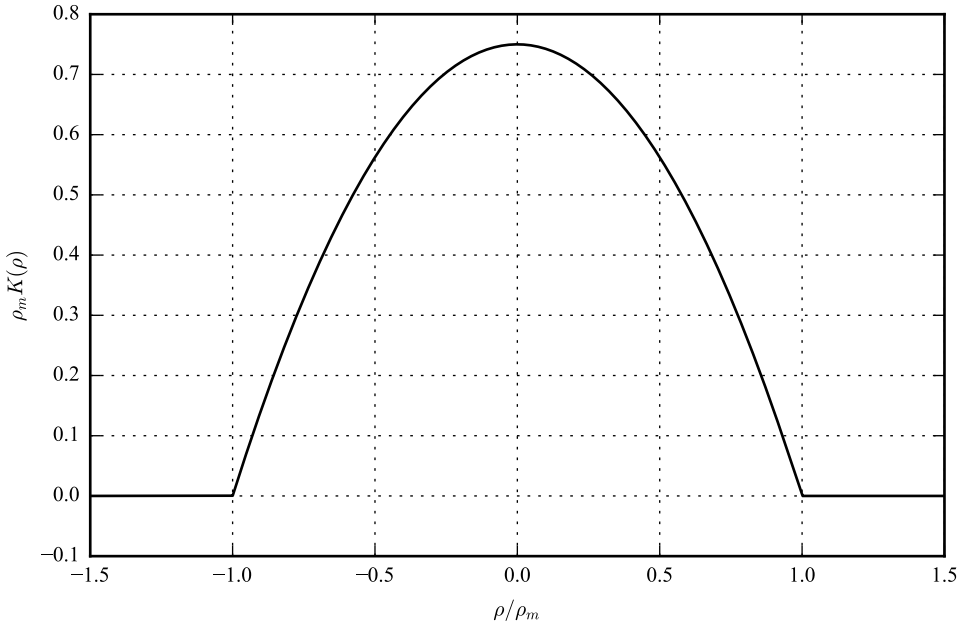

Fig. 2: Doppler spectrum smoothing kernel.

In this section, we will define the function $F(\rho)$ to represent the factors of (20) that only depend on ρ:

$$F(\rho) = \frac{G^2(\phi(\rho), \theta)}{\sqrt{\rho_x^2 \cos^2\theta - (\rho - \rho_z \sin\theta)^2}}. \tag{21}$$

This function is plotted in Figure 1. It should be noted that (21) assumes the clutter is not moving. As well, it should be noted that (21) possesses singularities when $|\rho - \rho_z \sin\theta| = \rho_x \cos\theta$, which presents problems during numerical integration because it makes the result very sensitive to the sample spacing.

One way of alleviating both of these problems is to smooth the Doppler spectrum by convolving the scattering function (20) with a smoothing kernel $K(\rho)$ to remove the singularities and broaden the Doppler spectrum by a modest amount to attempt to model wave motion. In this work, we choose a parabolic smoothing kernel given by:

$$K(\rho) = \frac{3}{4} \cdot \frac{1}{\rho_m} \operatorname{rect}\left(\frac{1}{2} \cdot \frac{\rho}{\rho_m}\right)\left(1 - \left(\frac{\rho}{\rho_m}\right)^2\right). \tag{22}$$

This function was chosen because it has a mostly "round" shape and (more importantly) the convolution of K with F possesses an analytic solution. Note that $K(\rho)$ is normalized so that it integrates to one so that the total power is conserved. The Doppler-domain convolution (of the factors that depend on ρ) is expressed

as follows:

$$\{F * K\}(\rho) = \left(\frac{G^2(\phi(\rho), \theta)}{\sqrt{\rho_x^2 \cos^2\theta - (\rho - \rho_z \sin\theta)^2}} \right) * K(\rho). \tag{23}$$

To simplify this convolution to a form for which an analytic solution is available, we will assume that the Doppler spread of the main antenna lobe is greater than ρ_m, implying that the antenna gain is constant over the smoothing interval:

$$\{F * K\}(\rho) \approx G^2(\phi, \theta) \left[\left(\frac{1}{\sqrt{\rho_x^2 \cos^2\theta - (\rho - \rho_z \sin\theta)^2}} \right) * K(\rho) \right], \tag{24}$$

for which the solution is given by:

$$\{F * K\}(\rho) = G^2(\phi, \theta) \begin{cases} 0, & |\rho - \rho_z \sin\theta| \geq \rho_x \cos\theta + \rho_m \\ h(\rho; \rho + \rho_m) - h(\rho; \rho - \rho_m), & |\rho - \rho_z \sin\theta| \leq \rho_x \cos\theta - \rho_m \\ h(\rho; \rho + \rho_m) - h(\rho; -\rho_x \cos\theta + \rho_z \sin\theta), & |\rho - \rho_z \sin\theta + \rho_x \cos\theta| < \rho_m \\ h(\rho; \rho_x \cos\theta + \rho_z \sin\theta) - h(\rho; \rho - \rho_m), & |\rho - \rho_z \sin\theta - \rho_x \cos\theta| < \rho_m \end{cases}, \tag{25}$$

where

$$h(\rho; L) = \frac{3}{8\rho_m^3} \Bigg[\left(\rho_x^2 \cos^2\theta + 2\rho_z^2 \sin^2\theta - 4\rho_z \rho \sin\theta + 2\rho^2 - 2\rho_m^2\right) \tan^{-1}\left(\frac{\rho_z \sin\theta - L}{\sqrt{\rho_x^2 \cos^2\theta - (\rho_z \sin\theta - L)^2}} \right) \ldots$$
$$\ldots + (3\rho_z \sin\theta - 4\rho + L) \sqrt{\rho_x^2 \cos^2\theta - (\rho_z \sin\theta - L)^2} \Bigg] \tag{26}$$

This result is plotted in Figure 3. Note that now the singularities have been removed and the Doppler spectrum has been broadeded, as desired.

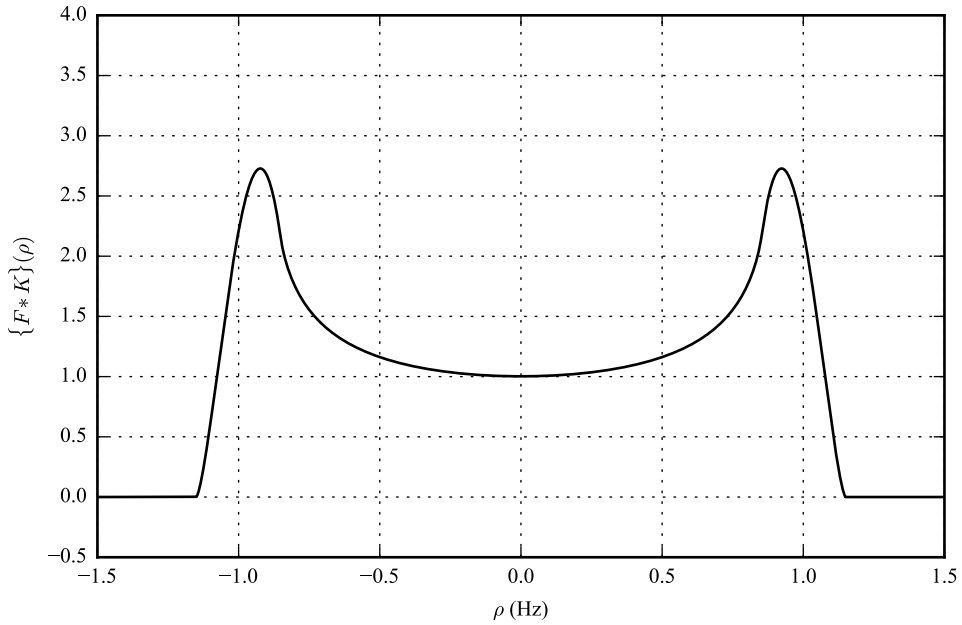

Fig. 3: Smoothed Clarke spectrum with $G = 1$, $\rho_m = 0.15$ Hz, $\rho_x \cos\theta = 1$ Hz, and $\rho_z \sin\theta = 0$ Hz.

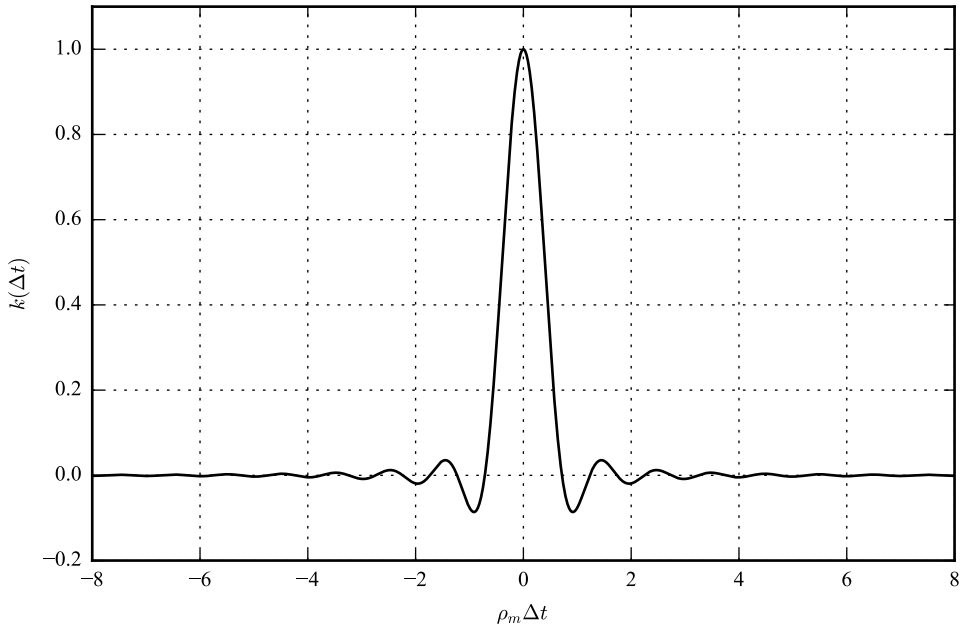

Fig. 4: Time-domain autocorrelation window function used to account for wave motion.

3.2 Time-Domain

To obtain an expression for the time-domain windowed autocorrelation function $A_w(\tau, \Delta t)$, we will perform Fourier inversion on the smoothed clutter scattering function $S_c(\tau, \rho)$ to yield the time-domain autocorrelation:

$$S_c(\tau, \rho) = S(\tau, \rho) * K(\rho) \tag{27}$$

$$\implies A_w(\tau, \Delta t) = \mathcal{F}_\rho^{-1}\left[S_c(\tau, \rho)\right]$$

$$= \mathcal{F}_\rho^{-1}\left[S(\tau, \rho) * K(\rho)\right] = \mathcal{F}_\rho^{-1}\left[S(\tau, \rho)\right]\mathcal{F}_\rho^{-1}\left[K(\rho)\right]$$

$$= A(\tau, \Delta t)k(\Delta t), \tag{28}$$

where

$$k(\Delta t) = \mathcal{F}_\rho^{-1}\left[K(\rho)\right] = 3\left(\frac{\sin(2\pi\rho_m\Delta t)}{(2\pi\rho_m\Delta t)^3} - \frac{\cos(2\pi\rho_m\Delta t)}{(2\pi\rho_m\Delta t)^2}\right) \tag{29}$$

Note that ρ_m can be modified to include range-dependence, i.e. $\rho_m = \rho_m(\tau)$ without loss of generality. The window function $k(\Delta t)$ is plotted in Figure 4. The resultant autocorrelation can then be found by substituting (18) into (28).

The Python code used to generate Figures 1-4 is found in Appendix A.

4. OUTPUT TIME-FREQUENCY POWER DISTRIBUTION

We will assume the matched-filter output is the correlation of N pulses against an infinite pulse train input, represented by the periodic ambiguity function (PAF) $|\chi_{NT}|$, given by [15]:

$$|\chi_{NT}(\tau,\rho)| = \left| \frac{1}{NT_r} \int_0^{NT_r} u(t)u^*(t+\tau)e^{j2\pi\rho t}dt \right|, \tag{30}$$

where $u(t)$ is the normalized unit-energy input pulse train and T_r is the pulse repetition interval (PRI). The pulse repetition frequency (PRF) is given by $F_r = 1/T_r$. The PAF has the following properties that are relevant to our application:

$$|\chi_{NT}(\tau,\rho)| = |\chi_{NT}(\tau+T_r,\rho)| \tag{31}$$

$$|\chi_{NT}(\tau,\rho)| = |\chi_{1T}(\tau,\rho)| \left| \frac{\sin N\pi\rho T_r}{N\sin\pi\rho T_r} \right| \tag{32}$$

The time-frequency power distribution at the channel output is given by:

$$\begin{aligned} P(\tau,\rho) &= E_x S(\tau,\rho) * *|\chi_{NT}(\tau,\rho)|^2. \\ &= E_x \int\int S(\tau',\rho')|\chi_{NT}(\tau-\tau',\rho-\rho')|^2 d\tau' d\rho', \end{aligned} \tag{33}$$

where E_x is the energy of the pulse train. Note that since $|\chi_{NT}|^2$ is periodic along the τ axis, the result of the convolution integral (33) is periodic along this axis as well. If we assume the support of S is limited to $[0, LT_r]$ for some positive integer L along the τ axis, then the output power distribution can be written as a circular convolution:

$$\begin{aligned} P(\tau,\rho) &= E_x \int_0^{LT_r} \int_{-\infty}^{\infty} S(\tau',\rho')|\chi_{NT}(\tau-\tau',\rho-\rho')|^2 d\rho' d\tau' \\ &= E_x \, S(\tau,\rho) \underset{\tau}{\circledast} \underset{\rho}{*} |\chi_{NT}(\tau,\rho)|^2, \end{aligned} \tag{34}$$

where $\underset{\tau}{\circledast}$ represents circular convolution along the τ axis and $\underset{\rho}{*}$ represents ordinary convolution along the ρ axis. The problem was cast into this form so that discrete implementations can use the Fast Fourier Transform (FFT) to efficiently compute the periodic output power distribution along the τ axis without zero-padding.

5. SIGNAL-TO-CLUTTER RATIO

We will now use the results from the previous sections to compute target signal and clutter signal power distributions. We will use as our transmitted signal an unmodulated train of $N = 8$ rectangular pulses with pulse width $T = 10$ μs, PRI $T_r = 100$ μs, and $E_x = N$ J. A plot of the PAF for this pulse train is shown in Figure 5 for the principal PRI and PRF interval. A sample rate $T_s = T/10 = 1$ μs and a frequency bin spacing of $\Delta\rho = F_r/256 = 39.0625$ Hz was used for all plots.

A smoothed clutter scattering function $S_c(\tau, \rho)$ was generated from (20) and (27) with the following parameters:

- $f = 10$ GHz
- $\mathbf{v} = [v_x, v_y, v_z]^T = [300, 0, 0]^T$ m/s
- $\rho_m = 66.713$ Hz
- $G(\phi, \theta)$ given by (8) with:
  - $D = 100$
  - $B = 10^\circ$
  - $\phi_a = 0^\circ$
  - $\theta_a = 25^\circ$

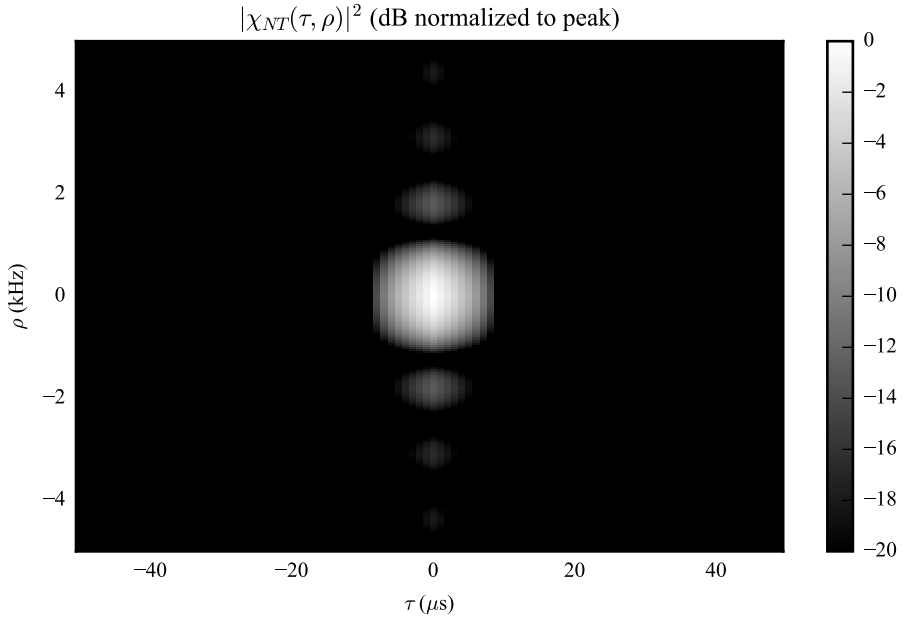

Fig. 5: Periodic ambiguity function (PAF) of a train of $N = 8$ rectangular pulses.

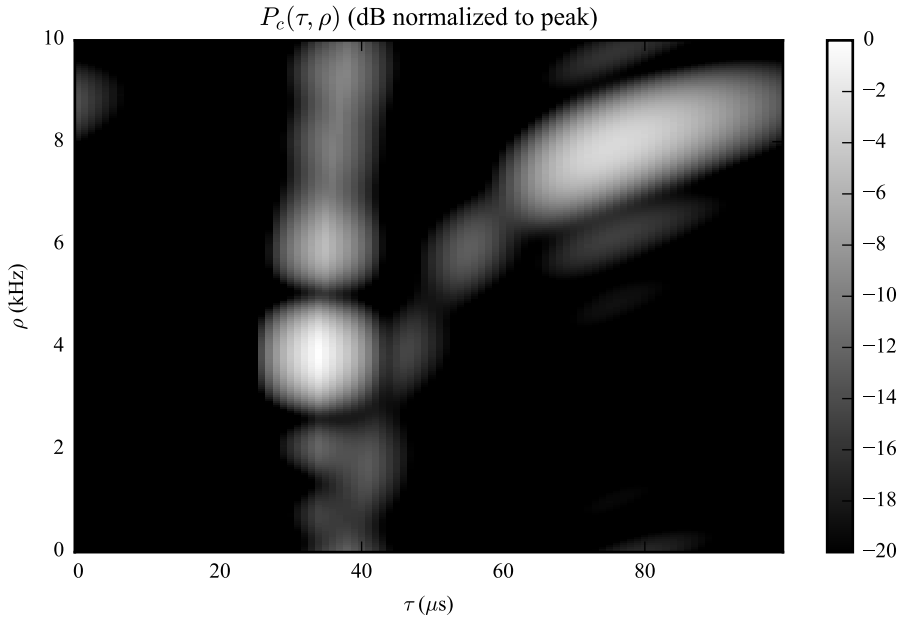

Fig. 6: Time-frequency distribution of clutter signal power.

- $\sigma^0(\alpha)$ data taken from Figure 7.13 of [16].

with the following earth geometry assumptions:

- $h = 5$ km
- $\alpha(\tau) = \theta(\tau) = \sin^{-1}(h/(\frac{1}{2}c\tau)) \implies$ flat earth model.
- $\phi(\tau, \rho) = \cos^{-1}\left(\frac{\rho - \rho_z \sin\theta(\tau)}{\rho_x \cos\theta(\tau)}\right)$, assuming $0 \leq \phi < \pi$.

The support of $S_c(\tau, \rho)$ along the τ axis was assumed to be $[0, 500 \times 10^{-6}]$ s, implying that $L = 5$. The relation derived in (34) was used to compute the clutter distribution, and the result $P_c(\tau, \rho)$ is plotted in Figure 6. From this figure the altitude return at $\tau \approx 33.4$ μs is clearly visible as well as the mainlobe clutter at $\tau \approx 78.9$ μs and $\rho \approx 8.14$ kHz.

The time-frequency distribution of a point target with RCS $\sigma = 40$ dBsm located at $(\bar{\tau}, \bar{\rho})$ is given by [1]:

$$P_t(\tau, \rho) = E_x \frac{G^2(\phi(\bar{\tau}, \bar{\rho}), \theta(\bar{\tau}))\lambda^2\sigma}{(4\pi)^3(\frac{1}{2}c\bar{\tau})^4} |\chi_{NT}(\tau - \bar{\tau}, \rho - \bar{\rho})|^2, \tag{35}$$

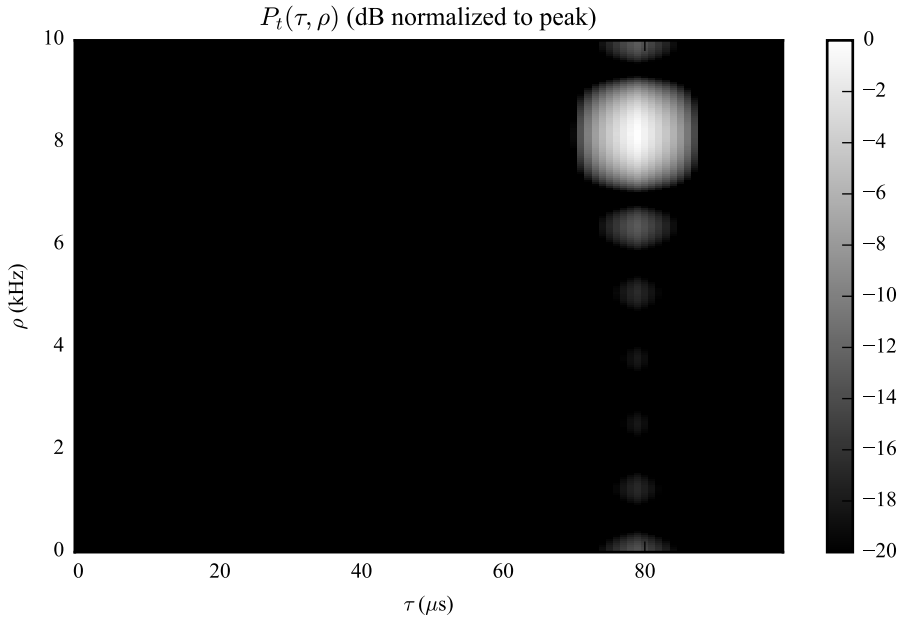

Fig. 7: Time-frequency distribution of target signal power.

which is simply a shifted and scaled version of the PAF. If the main beam is pointed at the target then $\bar{\tau} \approx 78.93$ μs and $\bar{\rho} = 18.14$ kHz. However, due to the fact that the digital receiver can only observe spectrum from 0 to F_r, the main lobe of the power distribution will appear at $\bar{\rho} - F_r \approx 8.14$ kHz. The result is plotted in Figure 7. The signal-to-clutter ratio (SCR) P_t/P_c as a function of (τ, ρ) is calculated and plotted in Figure 8. The observed SCR in the direction of the main lobe was computed as a function of N and is shown in Table 1.

N	SCR (dB)
1	7.228
2	7.331
4	7.463
8	7.730
16	8.414

Table 1: Signal-to-clutter ratio as a function of number of pulses coherently integrated.

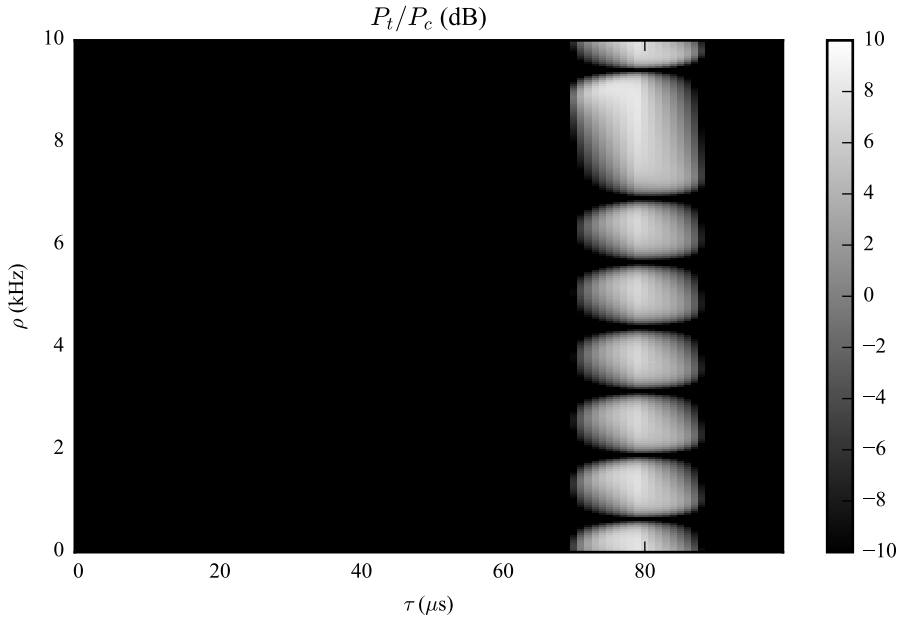

Fig. 8: Signal-to-clutter ratio as a function of delay and Doppler shift.

It can be seen from Table 1 that increasing the number of pulses integrated does improve the SCR, albeit modestly. If the clutter return from each pulse was completely independent, the SCR would be expected to increase linearly with N. However, because the pulses are highly correlated, integrating extra pulses does not improve the SCR very much.

The Python code used to generate Figures 5-8 is found in Appendix B.

6. CONCLUSION AND FUTURE WORK

In this report, further derivations related to the implementation of a WSSUS clutter model have been furnished, and output time-frequency power distributions have been produced to simulate what an actual radar would "see". Python code for said simulations has also been provided.

Future work on this topic could focus on verifying the assumptions behind the WSSUS model against experimental data and applying this model to the analysis and design of practical clutter mitigation techniques.

ACKNOWLEDGMENTS

The authors would like to acknowledge Applied Technology, Inc. and the U.S. Naval Research Lab for supporting this work.

REFERENCES

1. C. D. Cooke and D. E. Jarvis, "Application of statistical linear time-varying system theory to modeling of high grazing angle sea clutter," rept. NRL/MR/5750–17-9747, U.S. Naval Research Laboratory (July 2017).

2. P. Bello, "Characterization of randomly time-variant linear channels," *IEEE Transactions on Communications Systems* **11**(4), 360–393 (December 1963), ISSN 0096-1965, doi: 10.1109/TCOM.1963.1088793.

3. R. Price and P. E. Green, Jr., "Signal processing in radar astronomy– communication via fluctuating multipath media," rept. 234, MIT Lincoln Laboratory (October 1960).

4. P. E. Green, Jr., "Radar astronomy measurement techniques," rept. 282, MIT Lincoln Laboratory (December 1962).

5. A. Goldsmith, *Wireless Communications* (Cambridge University Press, 2005).

6. J. Proakis and M. Salehi, *Digital Communications* (McGraw-Hill, 5th edition, 2008).

7. G. Matz, H. Bolcskei, and F. Hlawatsch, "Time-frequency foundations of communications: Concepts and tools," *IEEE Signal Processing Magazine* **30**(6), 87–96 (Nov 2013), ISSN 1053-5888, doi: 10.1109/MSP.2013.2269702.

8. H. L. Van Trees, *Detection, Estimation, and Modulation Theory*, volume 3 (John Wiley & Sons, 1971).

9. S. M. Kay and S. B. Doyle, "Rapid estimation of the range-doppler scattering function," *IEEE Transactions on Signal Processing* **51**(1), 255–268 (Jan 2003), ISSN 1053-587X, doi: 10.1109/TSP.2002.806579.

10. G. E. Pfander and P. Zheltov, "Estimation of overspread scattering functions," *IEEE Transactions on Signal Processing* **63**(10), 2451–2463 (May 2015), ISSN 1053-587X, doi: 10.1109/TSP.2015.2403309.

11. K. Ward, R. Tough, and S. Watts, *Sea Clutter: Scattering, the K Distribution and Radar Performance*, Radar, Sonar &; Navigation (Institution of Engineering and Technology, 2013), URL `http://digital-library.theiet.org/content/books/ra/pbra025e`.

12. M. A. Richards, *Fundamentals of Radar Signal Processing* (IET/McGraw-Hill Education, 2nd edition, 2014).

13. E. W. Weisstein, "Jacobi-Anger expansion" (July 2017), URL `http://mathworld.wolfram.com/Jacobi-AngerExpansion.html`.

14. R. H. Clarke, "A statistical theory of mobile-radio reception," *The Bell System Technical Journal* **47**(6), 957–1000 (July 1968), ISSN 0005-8580, doi: 10.1002/j.1538-7305.1968.tb00069.x.

15. N. Levanon and E. Mozeson, *Radar Signals* (IEEE Press/Wiley-Interscience, 2004).

16. M. I. Skolnik, *Introduction to Radar Systems* (McGraw-Hill, 3rd edition, 2001).

Appendix A

CODE LISTING: FIGS1.PY

```
from pylab import *
close('all')
from figs2 import doppler_slice

def main():
    N = 2**14
    dt = linspace(-8., 8., N)
    rho_m = 1.
    figsize = (5.25,3.5)


    def k(tt):
        x = 2*pi*rho_m*tt
        rval = 3.*(sin(x)/x**3. - cos(x)/x**2.)
        rval[x == 0.] = 1.
        return rval

    figure(figsize=figsize)
    plot(dt, k(dt), 'k')
    xlabel(r'$\rho_m\Delta_t$')
    ylabel(r'$k(\Delta_t)$')
    ylim(-0.2, 1.1)
    grid()
    tight_layout()
    savefig('fig_kdelta.pdf')


    def K(ff):
        rval = 3./(4.*rho_m)*(1.-(ff/rho_m)**2.)
        rval[abs(ff) >= rho_m] = 0.
        return rval

    rho = linspace(-1.5, 1.5, N)

    figure(figsize=figsize)
    plot(rho, rho_m*K(rho), 'k')
    xlabel(r'$\rho/\rho_m$')
    ylabel(r'$\rho_m K(\rho)$')
    xlim(min(rho), max(rho))
    ylim(-0.1, 0.8)
```

```
    grid()
    tight_layout()
    savefig('fig_krho.pdf')


    rho_x = 1.
    rho_m = 0.15
    theta = zeros(rho.shape)
    rho_z = 0.

    def unsmoothed_slice(rr):
        i = abs(rho-rho_z*sin(theta)) < rho_x*cos(theta)
        returnval = zeros(rr.shape)
        returnval[i] = 1./sqrt((rho_x*cos(theta[i]))**2. - \
            (rr[i]-rho_z*sin(theta[i]))**2.)
        return returnval

    figure(figsize=figsize)
    plot(rho, unsmoothed_slice(rho), 'k')
    xlim(min(rho), max(rho))
    ylim(-0.5, 4)
    xlabel(r'$\rho$ (Hz)')
    ylabel(r'$F(\rho)$')
    grid()
    tight_layout()
    savefig('fig_f_unsmoothed.pdf')

    F = doppler_slice(rho, rho_x, rho_z, rho_m, theta=theta)
    figure(figsize=figsize)
    plot(rho, F, 'k')
    xlim(min(rho), max(rho))
    ylim(-0.5, 4)
    xlabel(r'$\rho$ (Hz)')
    ylabel(r'$\{F*K\}(\rho)$')
    grid()
    tight_layout()
    savefig('fig_f_smoothed.pdf')

    show()


if __name__ == '__main__':
    main()
```

Appendix B

CODE LISTING: FIGS2.PY

```
from pylab import *
close('all')

SPEED_OF_LIGHT_M_PER_S = 299792458.0

# Helper functions
def db(x):
    return 10*log10(x + 1e-99)

def cosd(x):
    return cos(x*pi/180.)

def sind(x):
    return sin(x*pi/180.)

def tand(x):
    return sind(x)/cosd(x)

def csc(x):
    return 1./sin(x)

def cscd(x):
    return csc(x*pi/180.)

def nnz(x):
    # returns number of nonzero elements
    return sum((x!=0).flatten())

def circconv2_tauaxis(x, y):
    # circular convolution along second dimension
    # normal zero-padded linear convolution along first dimension
    n1x, n2x = x.shape
    n1y, n2y = y.shape
    if n2x != n2y: raise Exception('second dimensions must be equal')
    n1 = n1x + n1y - 1
    npadx = n1 - n1x
    npady = n1 - n1y
    xpad = concatenate((
        complex128(x),
        zeros((npadx, n2x),
```

```
        dtype=complex,
        )), axis=0)
    ypad = concatenate((
        complex128(y),
        zeros((npady, n2y),
        dtype=complex,
        )), axis=0)
    return ifft2(fft2(xpad)*fft2(ypad))

def sinc2pattern(phi_deg, theta_deg, gain_db=20, beamwidth_deg=10):
    # sinc^2 power pattern in radial direction
    phi = phi_deg*pi/180.
    theta = theta_deg*pi/180.
    gain = 10.**(gain_db/10.)
    beamwidth = beamwidth_deg*pi/180.
    W = 0.44294647068945234030836999*2  # sinc half-power width
    return gain*sinc(sqrt(phi**2. + theta**2.)*W/beamwidth)**2

def sincpattern(phi_deg, theta_deg, gain_db=20, beamwidth_deg=10):
    # sinc power pattern in each direction
    phi = phi_deg*pi/180.
    theta = theta_deg*pi/180.
    gain = 10.**(gain_db/10.)
    beamwidth = beamwidth_deg*pi/180.
    W = 0.60335456440161419764575384*2  # sinc half-power width
    return gain*abs(sinc(phi*W/beamwidth)*sinc(theta*W/beamwidth))


def plot_image(x,y,z, figsize=None):
    dx1 = x[1,0] - x[0,0]
    dx2 = x[0,1] - x[0,0]
    dx = max(dx1,dx2)

    dy1 = y[1,0] - y[0,0]
    dy2 = y[0,1] - y[0,0]
    dy = max(dy1,dy2)

    if figsize != None:
        figure(figsize=figsize)
    else:
        figure()
    imshow(
        z,
        aspect='auto',
        origin='lower',
        interpolation='none',
```

```
        extent=(
            x.min() - dx/2.,
            x.max() + dx/2.,
            y.min() - dy/2.,
            y.max() + dy/2.,
            ),
        cmap=cm.gray,
        )
    colorbar()

def pulse_train_factor(rho, N, p, T_r):
    return_val = zeros(rho.shape)
    i = (rho == 0)
    return_val[~i] = abs(sin(pi*rho[~i]*(N-abs(p))*T_r)/sin(
        pi*rho[~i]*T_r))
    return_val[i] = N - abs(p)
    return return_val

def doppler_integral(rho, L, rhox, rhoz, rhom, theta):
    rhox += 1e-8
    sqrtarg = rhox**2.*cos(theta)**2. - (rhoz*sin(theta)-L)**2.
    badinds = sqrtarg < 0
    sqrtarg[badinds] = 0.
    sqrtpart = sqrt(sqrtarg)
    arctanpartargnum = rhoz*sin(theta)-L
    arctanpart = zeros(badinds.shape)
    arctanpart[badinds] = sign(arctanpartargnum[badinds])*pi/2.
    arctanpart[~badinds] = arctan(
        arctanpartargnum[~badinds]/sqrtpart[~badinds])
    return 3./(8.*rhom**3.)*(
        (rhox**2.*cos(theta)**2. + 2.*rhoz**2.*sin(theta)**2. - \
            4*rhoz*rho*sin(theta) + 2.*rho**2. - 2.*rhom**2. \
            )*arctanpart + sqrtpart*(3.*rhoz*sin(theta)-4.*rho + L))

def doppler_slice(rho, rhox, rhoz, rhom, theta):

    return_val = zeros(rho.shape)

    i1 = abs(rho - rhoz*sin(theta)) <= rhox*cos(theta) - rhom
    i2 = abs(rho - rhoz*sin(theta) + rhox*cos(theta)) < rhom
    i3 = abs(rho - rhoz*sin(theta) - rhox*cos(theta)) < rhom

    L2 = rho[i1] + rhom
    L1 = rho[i1] - rhom
    I2 = doppler_integral(rho[i1], L2, rhox, rhoz, rhom, theta[i1])
    I1 = doppler_integral(rho[i1], L1, rhox, rhoz, rhom, theta[i1])
```

```
    return_val[i1] =  I2 - I1

    L2 = rho[i2] + rhom
    L1 = -rhox*cos(theta[i2]) + rhoz*sin(theta[i2])
    return_val[i2] = doppler_integral(rho[i2], L2, rhox, rhoz, rhom,
        theta[i2])\
        - doppler_integral(rho[i2], L1, rhox, rhoz, rhom, theta[i2])

    L2 = rhox*cos(theta[i3]) + rhoz*sin(theta[i3])
    L1 = rho[i3] - rhom
    return_val[i3] = doppler_integral(rho[i3], L2, rhox, rhoz, rhom,
        theta[i3])\
        - doppler_integral(rho[i3], L1, rhox, rhoz, rhom, theta[i3])

    return return_val

# classes for implementing functors to simplify some computations
class AntennaPowerPattern(object):
    def __init__(self, gain_db, antenna_depression_angle_deg,
        beamwidth_deg, patterntype='rect'):

        self.gain_db = gain_db
        self.antenna_depression_angle_deg = \
            antenna_depression_angle_deg
        self.beamwidth_deg = beamwidth_deg
        self.patterntype = patterntype

    def __call__(self, phi, theta):
        if self.patterntype == 'rect':
            return sincpattern(
                phi*180./pi,
                theta*180./pi-self.antenna_depression_angle_deg,
                self.gain_db,
                self.beamwidth_deg,
                )
        elif self.patterntype == 'radial':
            theta_a = pi/180.*self.antenna_depression_angle_deg
            x = cos(theta_a)*cos(theta)*cos(phi) + \
                sin(theta_a)*sin(theta)
            y = cos(theta)*sin(phi)
            z = -sin(theta_a)*cos(theta)*cos(phi) + \
                cos(theta_a)*sin(theta)
            phi_prime = arctan2(y, x)
            theta_prime = arctan2(z, sqrt(x**2. + y**2.))
            phi_prime_deg = 180./pi*phi_prime
            theta_prime_deg = 180./pi*theta_prime
```

```
            return sinc2pattern(
                phi_prime_deg,
                theta_prime_deg,
                self.gain_db,
                self.beamwidth_deg,
                )

class DepressionAngle(object):
    def __init__(self, altitude):
        self.altitude = altitude

    def __call__(self, tau):
        output = zeros(len(tau))
        zeroind = 0.5*SPEED_OF_LIGHT_M_PER_S*tau < self.altitude
        output[zeroind] = pi/2.
        output[~zeroind] = arcsin(
            self.altitude/(0.5*SPEED_OF_LIGHT_M_PER_S*tau[~zeroind]))
        return output

class NrcsLookupTable(object):
    def __init__(self, f=10e9, polarization='vv'):
        if polarization == 'vv' and 8e9 <= f <= 12e9:
            data = genfromtxt('./data/sigma0vv_xband.csv',
                delimiter=',')
        else:
            raise Exception(
                'invalid polarization or frequency choice')
        self.x = data[:,0]
        self.y = 10.**(data[:,1]/10.)

    def __call__(self, grazing_angle_rad):
        grazing_angle_deg = grazing_angle_rad*180./pi
        output = zeros(len(grazing_angle_deg))
        badind = (grazing_angle_deg < 0) | (grazing_angle_deg >= 90)
        output[badind] = 0
        output[~badind] = interp(grazing_angle_deg[~badind],
            self.x, self.y)
        return output

class FlatEarthDepressionAngle(object):
    # also equal to the grazing angle
    def __init__(self, altitude):
        self.altitude = altitude

    def __call__(self, tau):
        r = 0.5*SPEED_OF_LIGHT_M_PER_S*tau
```

```
        return_val = zeros(tau.shape)

        i = r >= self.altitude
        return_val[i] = arcsin(self.altitude/r[i])
        return_val[~i] = pi/2.
        return return_val

class AzimuthAngle(object):
    def __init__(self, v, f, theta):
        if v[1] != 0: raise Exception('v_y must be zero')
        self.wavelength = SPEED_OF_LIGHT_M_PER_S/f
        self.rho_x = 2*v[0]/self.wavelength
        self.rho_z = 2*v[2]/self.wavelength
        self.theta = theta

    def __call__(self, tau, rho):
        arccosarg = (rho-self.rho_z*sin(self.theta(tau)))/(
            self.rho_x*cos(self.theta(tau)))
        badinds = abs(arccosarg) > 1.
        if nnz(badinds) > 0:
            arccosarg[badinds] = sign(arccosarg[badinds])
        return arccos(arccosarg)

class ClutterScatteringFunction(object):
    def __init__(self, f, sigma_0, v, phi, theta, alpha, G, h, rho_m):
        self.wavelength = SPEED_OF_LIGHT_M_PER_S/f
        self.sigma_0 = sigma_0
        if v[1] != 0:
            raise Exception('v_y must be zero')
        if v[0] == 0 and v[2] == 0:
            raise Exception('v must be nonzero')
        self.rho_x = 2*v[0]/self.wavelength
        self.rho_z = 2*v[2]/self.wavelength
        self.phi = phi
        self.theta = theta
        self.alpha = alpha
        self.G = G
        self.h = h
        self.rho_m = rho_m

    def __call__(self, tau, rho):
        thetatau = self.theta(tau)
        costheta = cos(thetatau)
        sintheta = sin(thetatau)

        i1 = abs(rho - self.rho_z*sintheta) < self.rho_x*costheta + \
```

```
            self.rho_m
        i2 = 0.5*SPEED_OF_LIGHT_M_PER_S*tau > self.h
        i = i1 & i2

        return_val = zeros(tau.shape)
        return_val[i] = self.wavelength**2*SPEED_OF_LIGHT_M_PER_S/(
            4.*pi)**3.
        return_val[i] *= self.sigma_0(self.alpha(tau[i]))/(
            0.5*SPEED_OF_LIGHT_M_PER_S*tau[i])**3.
        return_val[i] *= self.G(self.phi(tau[i],rho[i]),
            thetatau[i])**2.
        return_val[i] *= doppler_slice(
            rho[i], self.rho_x, self.rho_z, self.rho_m, thetatau[i])
        return_val[isinf(return_val)] = 0.
        return_val[isnan(return_val)] = 0.

        return return_val

class UnmodulatedPulseAmbiguityMag(object):
    def __init__(self, T):
        self.T = T

    def __call__(self, tau, rho):
        i = abs(tau) <= self.T
        return_val = zeros(tau.shape)
        return_val[i] = abs((1-abs(tau[i])/self.T)*sinc(self.T*rho[i]*(
            1-abs(tau[i])/self.T)))
        return return_val

class PulseTrainAmbiguityMag(object):
    def __init__(self, T, N, T_r, chi_pulse_mag):
        self.T = T
        self.N = N
        self.T_r = T_r
        self.chi_pulse_mag = chi_pulse_mag

    def __call__(self, tau, rho):
        return_val = zeros(tau.shape)
        i = abs(tau) <= self.N*self.T_r
        for p in xrange(-(self.N-1), self.N):
            return_val[i] += 1./self.N*self.chi_pulse_mag(
                tau[i]-p*self.T_r, rho[i])*pulse_train_factor(
                rho[i], self.N, p, self.T_r)
        return return_val

class PulseTrainPeriodicAmbiguityMag(PulseTrainAmbiguityMag):
```

```
    def __call__(self, tau, rho):
        tau_shift = copy(tau)%self.T_r
        i = tau_shift > self.T_r/2.
        tau_shift[i] = self.T_r - tau_shift[i]
        return 1./self.N*self.chi_pulse_mag(tau_shift, rho) * \
            pulse_train_factor(rho, self.N, p=0, T_r=self.T_r)


class PointTargetTimeFreqDistr(object):
    def __init__(self, E_x, G, phi, theta, f, sigma, tau_bar,
        rho_bar, chi_mag_sq):

        self.E_x = E_x
        self.G = G
        self.phi = phi
        self.theta = theta
        self.wavelength = SPEED_OF_LIGHT_M_PER_S/f
        self.sigma = sigma
        self.tau_bar = tau_bar
        self.rho_bar = rho_bar
        self.chi_mag_sq = chi_mag_sq

    def __call__(self, tau, rho):
        return self.E_x*self.G(self.phi(self.tau_bar,self.rho_bar),
            self.theta(self.tau_bar))**2. * \
            self.wavelength**2.*self.sigma/((4.*pi)**3. * \
            (0.5*SPEED_OF_LIGHT_M_PER_S*self.tau_bar)**4.) * \
            self.chi_mag_sq(tau-self.tau_bar, rho-self.rho_bar)


# main program code
def main():
    # inputs
    f = 10e9
    h = 5e3
    v = array([300, 0, 0])
    v_wave = 1.0
    phi_a_deg = 0.
    theta_a_deg = 25.
    beamwidth_deg = 10.
    gain_db = 20.
    T = 10e-6
    T_r = 100e-6
    N = 8
    E_x = N
    fft_pts_min_per_prf = 256
```

```
r_max = 50e3
figsize = (5.25,3.5)

# Calculated parameters
T_s = T/10.
rho_x = 2*v[0]*f/SPEED_OF_LIGHT_M_PER_S
rho_z = 2*v[2]*f/SPEED_OF_LIGHT_M_PER_S
rho_wave = 2*v_wave*f/SPEED_OF_LIGHT_M_PER_S
print 'rho_wave =', rho_wave
rho_h = rho_x*cosd(theta_a_deg)*cosd(phi_a_deg+beamwidth_deg/2.)+ \
    rho_z*sind(theta_a_deg)
deltarho = rho_x*cosd(theta_a_deg)*sind(phi_a_deg)*(
    beamwidth_deg/2.*pi/180.)
print 'differential doppler beamwidth (onesided) =', deltarho
r_a = h*cscd(theta_a_deg)
tau_a = 2*r_a/SPEED_OF_LIGHT_M_PER_S
rho_a = rho_x*cosd(theta_a_deg)*cosd(phi_a_deg) + rho_z*sind(
    theta_a_deg)
print 'tau_a = ', tau_a*1e6
print 'rho_a = ', rho_a
print 'doppler beamwidth (onesided) =', abs(rho_h - rho_a)
if abs(rho_h - rho_a) < rho_wave:
    raise Exception('antenna approximation not valid')
tau_max = 2*r_max/SPEED_OF_LIGHT_M_PER_S
if tau_a > tau_max:
    raise Exception('target is outside max simulation range')
B = rho_h + rho_wave + 1/T_r        # clutter bandwidth

# functors for doing computations
G = AntennaPowerPattern(
    gain_db=gain_db,
    antenna_depression_angle_deg=theta_a_deg,
    beamwidth_deg=beamwidth_deg,
    patterntype='rect'
    )
theta = FlatEarthDepressionAngle(
    altitude=h,
    )
alpha = theta
phi = AzimuthAngle(v, f, theta)
sigma_0 = NrcsLookupTable(f)
def G_rot(x,y): return G(x-abs(phi_a_deg*pi/180.), y)
C_eta = ClutterScatteringFunction(f, sigma_0, v, phi, theta,
    alpha, G_rot, h, rho_wave)

# time and frequency axis computations
```

```
if N < fft_pts_min_per_prf:
    fft_pad_factor = fft_pts_min_per_prf/N
else:
    fft_pad_factor = 1
drho = 1./(N*T_r)/fft_pad_factor
print 'drho:', drho
print 'PRI = %g us'%(T_r*1e6)
print 'PRF = %g kHz'%(1e-3/T_r)
if drho > rho_wave: raise Exception(
    'frequency resolution is too low')
print 'tau_max:', tau_max*1e6, 'us'
Npri = int(ceil(tau_max/T_r))
if (Npri%2) == 0: Npri += 1

# pulse train ambiguity computation
chi_pulse_mag = UnmodulatedPulseAmbiguityMag(T)
print 'computing pulse train ambiguity'
chi_train_mag = PulseTrainPeriodicAmbiguityMag(T, N, T_r,
    chi_pulse_mag)
def chi_train_mag_sq(ttt, rrr): return chi_train_mag(ttt, rrr)**2.
print 'done computing pulse train ambiguity'
Ntau = int(round(T_r/T_s))
tau_chi_lin = arange(Ntau)*T_s
rho_chi_lin = arange(-B, B, drho)
Nrho = len(rho_chi_lin)
tau_chi, rho_chi = meshgrid(tau_chi_lin, rho_chi_lin)

# plotting pulse train ambiguity
print 'plotting pulse train ambiguity'
tt = (tau_chi-T_r/2.)
rr = (rho_chi-0.5/T_r)
xx = db(chi_train_mag_sq(tt, rr))
plot_image(tt*1e6, rr/1e3, xx - xx.max(), figsize)
xlabel(r'$\tau$ ($\mu$s)')
ylabel(r'$\rho$ (kHz)')
title(r'$|\chi_{NT}(\tau,\rho)|^2$ (dB normalized to peak)')
ylim(-0.5/T_r/1e3, 0.5/T_r/1e3)
clim(-20,0)
tight_layout()
print 'done plotting'
savefig('fig_trainambig.pdf')

# repmatting ambiguity function
print 'pri ratio:', Npri
chi_train_mag_sq_repmat = tile(chi_train_mag_sq(tau_chi, rho_chi),
    (1, Npri))
```

```
tau = zeros(chi_train_mag_sq_repmat.shape)
rho = zeros(chi_train_mag_sq_repmat.shape)
for i in xrange(Npri):
    for j in xrange(1):
        tau[j*Nrho:(j+1)*Nrho, i*Ntau:(i+1)*Ntau] = tau_chi + i*T_r
        rho[j*Nrho:(j+1)*Nrho, i*Ntau:(i+1)*Ntau] = rho_chi + j/T_r
rho -= (1//2)/T_r    # centering Doppler axis on 0 Hz

# C_eta computation
print 'C_eta calc'
i = abs(rho_chi_lin) <= rho_h + rho_wave + 0.5/T_r
C_eta_mat = C_eta(tau[i,:],rho[i,:])
print 'done with C_eta'

# output power distribution computation
print 'convolving ambiguity'
cetarhomax = max(rho_chi_lin[i])
cetarhomin = min(rho_chi_lin[i])
chirhomax = max(rho_chi_lin)
chirhomin = min(rho_chi_lin)
rhomax = cetarhomax + chirhomax
rhomin = cetarhomin + chirhomin
minind = rhomin/drho
maxind = rhomax/drho
rhonew = arange(minind, maxind + 1)*drho
i = find((0 <= rhonew) & (rhonew < 1./T_r))
tau_P = arange(Ntau)*T_s
rho_P = arange(fft_pts_min_per_prf)*drho
tau_P, rho_P = meshgrid(tau_P, rho_P)
P = E_x*circconv2_tauaxis(C_eta_mat, chi_train_mag_sq_repmat
    ).real*T_s*drho
P = P[i,:Ntau]

print 'done convolving ambiguity'
plot_image(tau_P*1e6, rho_P/1e3, db(P/P.max()), figsize)
xlabel(r'$\tau$ ($\mu$s)')
ylabel(r'$\rho$ (kHz)')
title(r'$P_c(\tau,\rho)$ (dB normalized to peak)')
clim(-20, 0)
tight_layout()
savefig('fig_outputpdmap.pdf')

# point target return
sigma = 10000
P_prime = PointTargetTimeFreqDistr(E_x, G_rot, phi, theta, f,
    sigma, tau_a, rho_a, chi_train_mag_sq)
```

```
Pp = P_prime(tau_P, rho_P)
plot_image(tau_P*1e6, rho_P/1e3, db(Pp/Pp.max()), figsize)
xlabel(r'$\tau$ ($\mu$s)')
ylabel(r'$\rho$ (kHz)')
title(r"$P_t(\tau,\rho)$ (dB normalized to peak)")
clim(-20, 0)
tight_layout()
savefig('fig_pointtarget.pdf')

# calculating signal-to-clutter ratio
tau_a_aliased = tau_a
rho_a_aliased = rho_a
while tau_a_aliased > T_r: tau_a_aliased -= T_r
while rho_a_aliased > 1./T_r: rho_a_aliased -= 1./T_r
i = argmin(abs(tau_P[0,:]-tau_a_aliased))
j = argmin(abs(rho_P[:,0]-rho_a_aliased))
print 'tau_a_aliased =', tau_a_aliased
print 'rho_a_aliased =', rho_a_aliased
sigpwr = P_prime(array([tau_a_aliased]), array([rho_a_aliased]))
clutterpwr = P[j,i]
print 'db(sigpwr) =', db(sigpwr)
print 'db(clutterpwr) =', db(clutterpwr)
print 'SCR_discrete =', db(sigpwr/clutterpwr), 'dB'
# compare Richards "Fundamentals of Radar Signal Processing" p.48
if T/tau_a*tand(theta_a_deg) > beamwidth_deg*pi/180.:
    print 'beam limited according to Richards'
else:
    print 'pulse limited according to Richards'
R = 0.5*SPEED_OF_LIGHT_M_PER_S*tau_a
DeltaR = 0.5*SPEED_OF_LIGHT_M_PER_S*T
SCR_richards_blimited = sigma*sind(theta_a_deg)/(
    R**2.*(beamwidth_deg*pi/180.)**2.*sigma_0(
    array([theta_a_deg*pi/180.])))
SCR_richards_plimited = sigma*cosd(theta_a_deg)/(
    R*sigma_0(array([theta_a_deg*pi/180.])) * \
    DeltaR*beamwidth_deg*pi/180.)
print 'SCR_richards_blimited =', db(SCR_richards_blimited), 'dB'
print 'SCR_richards_plimited =', db(SCR_richards_plimited), 'dB'

# plotting SCR
SCR = Pp/P
plot_image(tau_P*1e6, rho_P/1e3, db(SCR), figsize)
xlabel(r'$\tau$ ($\mu$s)')
ylabel(r'$\rho$ (kHz)')
title(r"$P_t/P_c$ (dB)")
clim(-10, 10)
```

```
    tight_layout()
    savefig('fig_scr.pdf')



if __name__ == '__main__':
    main()
```