Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Numerically Computing the KummerU Function and a Special Case of the Gauss Hypergeometric Function

With Application to the Computation of CFAR Thresholds for Dual Aperture SAR GMTI

Ishuwa C. Sikaneta

Defence R&D Canada – Ottawa
TECHNICAL REPORT
DRDC Ottawa TR 2006-294
December 2006

Canada

Numerically Computing the KummerU Function and a Special Case of the Gauss Hypergeometric Function

With Application to the Computation of CFAR Thresholds for Dual Aperture SAR GMTI

Ishuwa C. Sikaneta

Defence R&D Canada – Ottawa

Technical Report

DRDC Ottawa TR 2006-294

December 2006

Principal Author

Ishuwa C. Sikaneta

Approved by

Doreen Dyck
Head/RS

Approved for release by

Cam Boulet
Head/Document Review Panel

Abstract

This report proposes new methods to compute the Kummer hypergeometric function of the second kind and to approximate a special case of the Gauss hypergeometric function. The Kummer hypergeometric function, $\Psi(a, c; z)$, is computed for $a, c \in \mathbb{R}$ and $x \in \mathbb{R}^+$ although the proposed method can be extended to $a, c, z \in \mathbb{C}$. The special case of the Gauss hypergeometric function is given by ${}_2\mathrm{F}_1(\mu+\nu-1, \nu-1/2; \mu+1/2; x)$ where $\mu > |\nu - 1|$, and where $x \in \mathbb{R} < 1$. Numerical results of the proposed algorithms are compared with the capabilities of GSL™, Mathematica™and Maple™.

The evaluation of these functions is required for the dual aperture Synthetic Aperture Radar (SAR), Ground Moving Target Indication (GMTI), Constant False Alarm Rate (CFAR) problem. It is shown that the proposed methods provide a practical means for computing the CFAR thresholds over a wide range of the SAR GMTI statistical parameters and false alarm thresholds, even in very heterogeneous terrain. Additionally, this report shows how to handle infinite integrals that arise when the statistics of heterogeneous terrain are described by the product model.

A C++ implementation of the developed algorithms is provided.

Résumé

Ce rapport présente de nouvelles techniques numériques permettant de calculer la fonction hypergéométrique de Kummer de la deuxième espèce et d'estimer un cas spécial de la fonction hypergéométrique de Gauss. La méthode proposée calcule la fonction hypergéométrique de Kummer, $\Psi(a, c; z)$, pour $a, c \in \mathbb{R}$ et $x \in \mathbb{R}^+$, bien qu'elle puisse être étendue pour $a, c, z \in \mathbb{C}$. Le cas spécial de la fonction hypergéométrique de Gauss est donné par ${}_2\mathrm{F}_1(\mu + \nu - 1, \nu - 1/2; \mu + 1/2; x)$, pour $\mu > |\nu - 1|$ et $x \in \mathbb{R} < 1$. Nous avons comparé les résultats numériques obtenus avec l'algorithme proposé avec les capacités de GSL™, Mathematica™et Maple™.

Nous avons évalué ces fonctions en les appliquant au problème du taux de fausses alarmes constant pour l'indication de cibles mobiles au sol par un radar ouverture synthètique (ROS) double. Nous montrons que les méthodes proposées sont pratiques pour calculer des seuils de taux de fausses alarmes constant (TFAC), pour une vaste gamme de paramètres statistiques d'indication de cibles mobiles au sol par un ROS et de seuils de fausse alarme, même pour un terrain très hétérogène. En outre, ce rapport indique comment traiter les intégrales infinies qui apparaissent lorsque les statistiques de terrain hétérogène sont décrites par le modèle du produit.

Nous fournissons un programme C++ des algorithmes mis au point.

Executive summary

Numerically Computing the KummerU Function and a Special Case of the Gauss Hypergeometric Function

Ishuwa C. Sikaneta; DRDC Ottawa TR 2006-294; Defence R&D Canada – Ottawa; December 2006.

This report develops new numerical techniques to compute the Kummer hypergeometric function of the second kind, and to approximate a special case of the Gauss hypergeometric function. These two special functions are widely used in the physical sciences, from quantum mechanics to multivariate statistical analysis to computational chemistry, but are here motivated by the desire to implement a Constant False Alarm Rate (CFAR) scheme for Synthetic Aperture Radar (SAR) Ground Moving Target Indication (GMTI). Computation of the Kummer hypergeometric function is achieved by using path integration on its defining differential equation, and approximation to the special case of the Gauss hypergeometric function is achieved by using the method of steepest descents on an equivalence with an infinite integral.

A CFAR detector is probably the only means to implement an automatic ground moving target indication (GMTI) processor such as that envisioned for Radarsat 2. The desire to use a wide range of false alarm rates motivates the construction of a processor where the false alarm rate is run-time-defined. The mathematical theory for implementing the CFAR detector is known, however, the implementation of the theory quickly encounters practical limitations because of the aforementioned special functions. Additionally, problems arise when numerically computing infinite integrals that arise in the case of terrain non-stationarity. This report provides the algorithms to compute the special functions and a method to numerically handle the infinite integrals.

A C++ implementation of the developed algorithms is provided.

Sommaire

Numerically Computing the KummerU Function and a Special Case of the Gauss Hypergeometric Function

Ishuwa C. Sikaneta; DRDC Ottawa TR 2006-294; R & D pour la défense Canada – Ottawa; décembre 2006.

Ce rapport présente de nouvelles techniques numériques permettant de calculer la fonction hypergéométrique de Kummer de la deuxième espèce et d'estimer un cas spécial de la fonction hypergéométrique de Gauss. Ces deux fonctions spéciales sont largement utilisées en sciences physiques, notamment en mécanique quantique, en analyse statistique multivariée et en passant chimie informatique. Notre travail était motivé par le souhait de créer un mécanisme de TFAC pour l'indication de cible mobiles au sol (ICMS) l'aide d'un ROS. Le calcul de la fonction hypergéométrique de Kummer est réalisée par l'intégration de chemin des équations différentielles qui la définissent, tandis que l'approximation du cas spécial de la fonction hypergéométrique de Gauss est réalisée l'aide de la méthode de la plus grande pente sur une fonction équivalente avec une intégrale infinie.

Un détecteur avec un taux de fausses alarmes constant est probablement le seul moyen de mettre en oeuvre un processeur d'indication de sources mobiles au sol comme celui considéré pour Radarsat2. Le souhait d'utiliser une vaste gamme de taux de fausses alertes entrane la construction d'un processeur dont le taux de fausses alertes serait défini lors de l'exécution. La théorie mathématique visant la mise en oeuvre du détecteur TFAC est connue mais, en pratique, l'application de celle-ci est rapidement limitée par les fonctions spéciales susmentionnées. En outre, l'évaluation numérique des intégrales infinies qui apparaissent lorsque le terrain n'est pas stationnaire cause des difficultés. Ce rapport contient les algorithmes de calcul de ces fonctions spéciales, ainsi qu'une méthode de traitement numérique des intégrales infinies.

Nous fournissons un programme C++ des algorithmes mis au point.

Table of contents

List of tables

List of figures

1 Introduction

This report discusses a new numerical method to compute the confluent hypergeometric function $\Psi(a, c; z)$ for the case of $a, c, z \in \mathbb{R}^+$. This computation of $\Psi(a, c; z)$ has no issues with integer values of a and c. The motivation for computing this function arises from the practical problem of computing CFAR thresholds in SAR GMTI. However, as stated in [1], this function has application in virtually every branch of physical science, from statistics, to computational chemistry to quantum physics. This report presents a method based on integration of the defining ordinary differential equation (ODE), and demonstrates that the function can be evaluated over a large domain in its defining parameters. In fact, the presented routine is able to compute the function values in domains where current routines that are implemented by commercial software fail.

A second objective of this report is to provide a numerical algorithm for computing a special case of the Gauss hypergeometric function. This special case, again, arises from the SAR GMTI CFAR problem. The presented algorithm computes this special case of the Gauss hypergeometric function using an approximation derived by using the method of steepest descents. Specifically, an approximation to ${}_2\mathrm{F}_1(\mu + \nu - 1, \nu - 1/2; \mu + 1/2; x)$ with $\mu > |\nu - 1|$ $x < 1$ is provided. Unlike direct application of the summation formula, this approximation does not suffer as x approaches the singularity at $|x| = 1$.

This report applies the developed numerical techniques to the problem of determining the CFAR thresholds for SAR GMTI.

1.1 Report outline

Section 2 discusses computation the Kummer hypergeometric function, $\Psi(a, c; z)$.

In section 3 we provide an algorithm for approximating ${}_2\mathrm{F}_1(\mu+\nu-1, \nu-1/2; \mu+1/2; x)$ with $\mu > |\nu - 1|$ $x < 1$. This function appears in the distribution of one of the CFAR variables.

Section 4 introduces the SAR GMTI CFAR problem. Since this problem motivated the research, the loop is closed by demonstrating how the proposed algorithms can be used to solve the problem of computing CFAR thresholds for the Displaced Phase Centre Antenna (DPCA), the Along-Track Interferometric Phase (ATI) and the hy-

perbolic detector. To complete the SAR GMTI CFAR analysis, Section 4.3 introduces a model of terrain heterogeneity and presents the heterogeneous GMTI metric probability distributions. Computation of these modified probability distributions is further complicated because of their representation through an infinite integral. For DPCA, for a special case, this infinite integral can be solved in closed form, but no closed form exists for the hyperbolic detector; thus, we discuss how to account for the infinite limit for a general, semi-definite metric. The ATI metric probability distribution is not affected by terrain heterogeneity.

Results using the proposed methods are compared to those obtained using commercial software such as Maple™, Mathematica™and Matlab™, as well as open source libraries such as GSL™. We demonstrate that the proposed methods provide a greater domain of accurate computation for the distribution functions when compared to Maple™, Mathematica™, Matlab™and GSL™.

Finally, C++ code that implements the developed algorithms is presented in the appendix. The code has been used to compute all tabulated values in this report which are compared, in various places, to results computed using Maple™, GSL™and Mathematica™.

2 Numerically computing $\Psi(a, c; z)$

This section outlines a numerical method for evaluating the Kummer hypergeometric function of the second kind.

2.1 Background on $\Psi(a, c; z)$

$\Psi(a, c; z)$ is sometimes called the Tricomi function, the Kummer U function, or a degenerate hypergeometric function of the second kind. The usual notation is $U(a, c; z)$, or $\Psi(a, c; z)$. This report adopts the latter notation. The report also uses the notations of $z \in \mathbb{C}$ and $x \in \mathbb{R}$.

$\Phi(a, c; z)$ and $\Psi(a, c; z)$ are solutions of the second-order differential equation

$$z\frac{\mathrm{d}^2 y}{\mathrm{d}z^2} + (c - z)\frac{\mathrm{d}y}{\mathrm{d}z} - ay = 0. \tag{1}$$

From (1) one recognises regular singular points at $z = 0, \infty$. All other values of z are

"ordinary" as defined in [2] and thus two solutions of the form

$$y = \sum_{k=0}^{\infty} a_k (z-a)^k \tag{2}$$

exist for $z \neq 0$.

$\Phi(a, c; z)$ is known as the Kummer M function, or the degenerate hypergeometric function of the first kind and also has alternative notations $M(a, c; z)$ and ${}_1F_1(a, c; z)$. It can be written as

$$\Phi(a, c; z) = \sum_{n=0}^{\infty} \frac{(a)_n}{(c)_n} \frac{z^n}{n!}, \tag{3}$$

where, as in [3; 1; 4], the Pochhammer symbol is defined as $(a)_n = a(a+1)(a+2)\ldots(a+n-1)$, $(a)_0 = 1$.

The second solution, and object of our interest is given by

$$\Psi(a, c; z) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} \Phi(a, c; z) + \frac{\Gamma(c-1)}{\Gamma(a)} z^{1-c} \Phi(a-c+1, 2-c; z), \quad c \notin \mathbb{Z}, \tag{4}$$

and is defined by the limit as c approaches an integer for $c \in \mathbb{Z}$. It has an integral representation, see [3; 1; 4], given by

$$\Psi(a, c; z) = \frac{1}{\Gamma(a)} \int_0^{\infty} e^{-zt} t^{a-1} (1+t)^{c-a-1} \mathrm{d}t \tag{5}$$

It is apparent from (5) that $\lim_{z \to 0} \Psi(a, c; z) = \infty$ for $c > 1$, because then the integrand is of order t^{-1} or larger.

For special values of a and c, $\Psi(a, c; z)$ is related to many functions including the modified Bessel function, the Hankel functions, the spherical Bessel function, the Airy function, the parabolic cylinder function and others. The function $\Psi(a, c; z)$ is related to the incomplete gamma function by, see [3],

$$x^{\alpha} e^{-x} \Psi(1, 1+\alpha; x) = \Gamma(\alpha, x), \tag{6}$$

where

$$\Gamma(\alpha, x) = \int_x^{\infty} e^{-t} t^{\alpha-1} \mathrm{d}t, \tag{7}$$

and $\Psi(a, c; z)$ is related to the modified Bessel function through

$$\Psi(\nu + \frac{1}{2}, 2\nu + 1; 2z) = \pi^{-\frac{1}{2}} e^z (2z)^{-\nu} K_{\nu}(z). \tag{8}$$

2.2 Path integration of the differential equation

One method for numerically evaluating $\Psi(a, c; z)$ uses path integration of the differential equation from a known point to a desired point, see, for instance, [5]. In this method, an n^{th} order linear ordinary differential equation (ODE) is transformed into an $n-$dimensional first order system. This first order system is advanced from a known point to a desired point using any number of stepping routines that are all, essentially, based upon the Euler method as described in [5; 6]. In this report, integration of the differential equation is achieved using an embedded fourth order Runge-Kutta-Fehlberg stepping routine, see [6]. The Runge-Kutta-Fehlberg routine proves more robust than the 4^{th} order (classical) Runge-Kutta method, the Bulirsch-Stoer method, Gear methods and other Implicit fourth order Runge-Kutta methods.

Advantages of path integration of the differential equation include the possibility to evaluate the functions over a large domain in $\mathbb{C}$, although we restrict our analysis to $\mathbb{R}$, and the flexibility to evaluate both solutions with the same algorithm, though with different initial values.

Integration of (1) for $\Psi(a, c; z)$, using routines outlined in [5], suffers from the following problems (illustrated in figure 1):

- the method fails when integrating outwards from small values with both explicit routines and implicit routines that are designed integrate stiff differential equations.
- path integration method starts to fail as $x \to 0$.

In figure 1, we see divergence when using an outward directed embedded Runge-Kutta-Fehlberg method. The function $\Psi(3, 14.5; x)$ is initially evaluated at $x_0 = 2.0$. Furthermore, the method fails to compute values lower than $\approx 10^{-4.2}$. We first consider the problem of divergence when integrating outwards from the origin.

2.2.1 Integration path orientation error analysis

In order to numerically integrate a second order linear differential equation, such as (1), one first constructs a first-order linear differential equation by considering a new variable

$$u = \frac{\mathrm{d}y}{\mathrm{d}z}. \tag{9}$$

The coupled first-order differential equation can then be written as

$$\frac{\mathrm{d}}{\mathrm{d}z}\begin{bmatrix} u \\ y \end{bmatrix} = \begin{bmatrix} -P(z) & -Q(z) \\ 1 & 0 \end{bmatrix}\begin{bmatrix} u \\ y \end{bmatrix}, \tag{10}$$

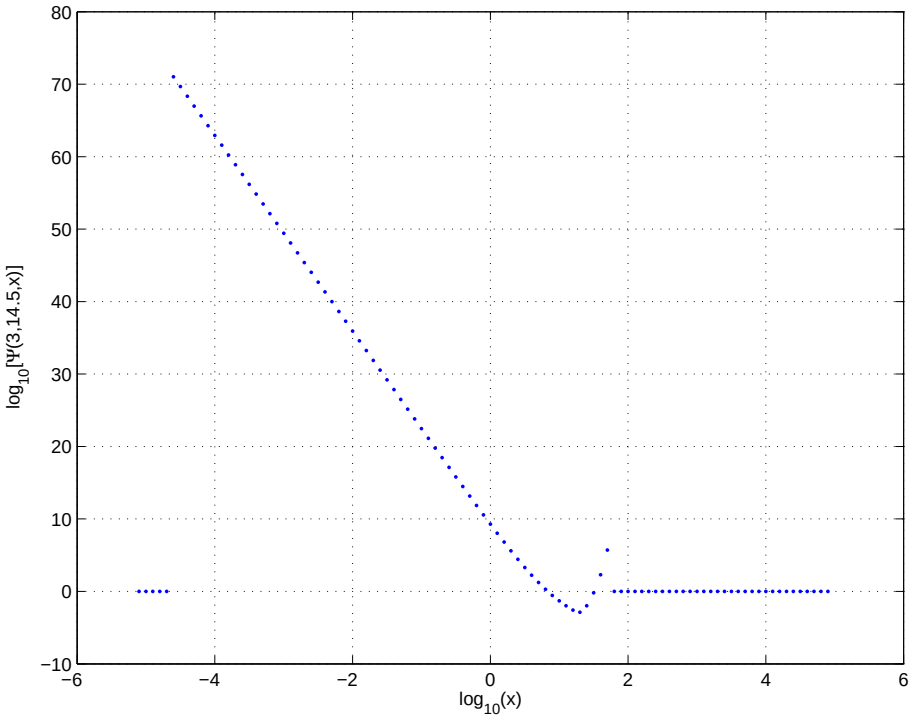

Figure 1: $\log_{10}(\Psi(a,c;x))$ *as a function of* $\log_{10}(x)$ *for* $a = 3$ *and* $c = 14.5$ *for the linear (classical) method.*

which we write as

$$\frac{\mathrm{d}}{\mathrm{d}z}\vec{y} = \boldsymbol{C}(\boldsymbol{z})\vec{y}, \tag{11}$$

where

$$\vec{y} = \begin{bmatrix} u \\ y \end{bmatrix}, \tag{12}$$

$$P(z) = \frac{c-z}{z}, \tag{13}$$

$$Q(z) = -\frac{a}{z}. \tag{14}$$

The basic Euler algorithm, as outlined in [6; 5], advances the solution at a known point to a neighbouring point through

$$\begin{aligned} \vec{y}_{n+1} &= \vec{y}_n + h\frac{\mathrm{d}}{\mathrm{d}z}\vec{y}_n, \\ &= \vec{y}_n + h\boldsymbol{C}(\boldsymbol{z})\vec{y}_n, \\ &= (\mathbf{1} + h\boldsymbol{C}(\boldsymbol{z}))\vec{y}_n. \end{aligned} \tag{15}$$

Now assume that at some iteration we have an estimate $\hat{\vec{y}}_n = \vec{y}_n + \vec{\epsilon}_n$ where the vector $\vec{\epsilon}$ is the error in the estimate. When this estimate is substituted into (15), one finds that

$$\begin{aligned} \hat{\vec{y}}_{n+1} &= (\mathbf{1} + h\boldsymbol{C}(\boldsymbol{z}))(\vec{y}_n + \vec{\epsilon}_n), \\ &= \vec{y}_{n+1} + (\mathbf{1} + h\boldsymbol{C}(\boldsymbol{z}))\vec{\epsilon}_n, \\ &= \vec{y}_{n+1} + \vec{\epsilon}_{n+1}, \end{aligned} \tag{16}$$

which yields a recurrence relation for the error of the $(n+1)^{\text{th}}$ estimate given by

$$\vec{\epsilon}_{n+1} = \boldsymbol{A}(\boldsymbol{z})\vec{\epsilon}_n, \quad \text{where} \tag{17}$$

$$\boldsymbol{A}(\boldsymbol{z}) = \begin{bmatrix} 1 - hP(z) & -hQ(z) \\ h & 1 \end{bmatrix}. \tag{18}$$

In the limit $|z| \to \infty$, we find that

$$\lim_{|z|\to\infty} \boldsymbol{A}(\boldsymbol{z}) = \begin{bmatrix} 1+h & 0 \\ h & 1 \end{bmatrix}, \tag{19}$$

and the error vector becomes

$$\lim_{|z|\to\infty} \vec{\epsilon}_{n+1} = \begin{bmatrix} (1+h)\epsilon_0 \\ \epsilon_1 + h\epsilon_0 \end{bmatrix}, \tag{20}$$

where

$$\vec{\epsilon}_n = \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \end{bmatrix}. \tag{21}$$

Repeated application of the recurrence relation yields

$$\lim_{|z|\to\infty} \vec{\epsilon}_{n+k+1} = \begin{bmatrix} (1+h)^k\epsilon_0 \\ \epsilon_1 - \epsilon_0(1-(1+h)^k) \end{bmatrix}. \tag{22}$$

Clearly, if $h > 0$, i.e. integrating outwards, then the error vector magnitude diverges as k increases. On the other hand, if one integrates inwards from a large $|z|$, with $h < 0$, $|h| < 1$ and for increasing k, the first component of the error vector decreases, while the second component tends to the constant value of $\epsilon_1 - \epsilon_0$. Thus, it behoves the computation to orient the path of integration inwards towards the origin. Fortunately, an asymptotic expansion[1] for $\Psi(a, c; z)$ and the associated error as $z \to \infty$ is known. We propose using a large initial value of z for the path integration, chosen such that the error in the approximation satisfies some provided tolerance. The solution at this large value, and its derivative, defined by

$$\frac{\mathrm{d}}{\mathrm{d}z}\Psi(a, c; z) = -a\Psi(a+1, c+1; z), \tag{23}$$

are used as the initial values in integration along a path directed towards the origin.

[1] asymptotic expansions need not be unique [7]

2.3 An asymptotic expansion of $\Psi(a, c; z)$

An asymptotic expansion of $\Psi(a, c; z)$ can be derived using Watson's lemma as in [7] to give,

$$\Psi(a, c; z) = z^{-a} \left\{ \sum_{n=0}^{R-1} \frac{(a)_n (1 + a - c)_n}{n!} (-z)^{-n} + O(|z|^{-R}) \right\}, \tag{24}$$

where

$$O(|z|^{-R}) = \frac{(a)_R (1 + a - c)_R}{R!} (-z)^{-R} \left[\frac{1}{2} + \frac{\frac{1}{8} + \frac{1}{4}c - \frac{1}{2}a + \frac{1}{4}z - \frac{1}{4}R}{z} + O(|z|^{-2}) \right], \tag{25}$$

and R is selected such that, of all the terms in (24), the absolute value of the R^{th} term is a minimum.

One peculiarity of asymptotic expansions that should be stressed is that for a given value of z, increasing the number of terms in the asymptotic expansion does not increase the accuracy of the estimate. In fact, generally, the expansion diverges. There is an optimum number of terms to use in the expansion that depends on $|z|$. Figure 2 illustrates the error in the asymptotic expansion, (24), as a function of $|z|$ versus the number of terms in the expansion. The C++ code provided in appendix E devotes special attention to the selection of an appropriate large value for x as well as the appropriate number of terms for the asymptotic expansion.

2.4 Computation of $\Psi(a, c; z)$ as $z \longrightarrow 0$

An aid to computing $\Psi(a, c; z)$ as $z \to 0$ is provided by first transforming the differential equation. We hypothesize that the solution to (1) takes the form $y(z) = e^{f(z)}$. Direct substitution of this expression into differential equations of the form

$$\frac{\mathrm{d}^2 y}{\mathrm{d}z^2} + P(z) \frac{\mathrm{d}y}{\mathrm{d}z} + Q(z) y = 0 \tag{26}$$

yields a new differential equation in $f(z)$,

$$\frac{\mathrm{d}^2 f}{\mathrm{d}z^2} + \left(\frac{\mathrm{d}f}{\mathrm{d}z} \right)^2 + P(z) \frac{\mathrm{d}f}{\mathrm{d}z} + Q(z) = 0. \tag{27}$$

By using path integration on (27), we can thus compute a solution to $\ln[y(z)]$. This simple transformation improves the convergence (speed of convergence) and accuracy of the evaluation of $\Psi(a, c; z)$ for small z when $c > 1$, since $\Psi(a, c; z)$ grows more rapidly than $\ln(\Psi(a, c; z))$.

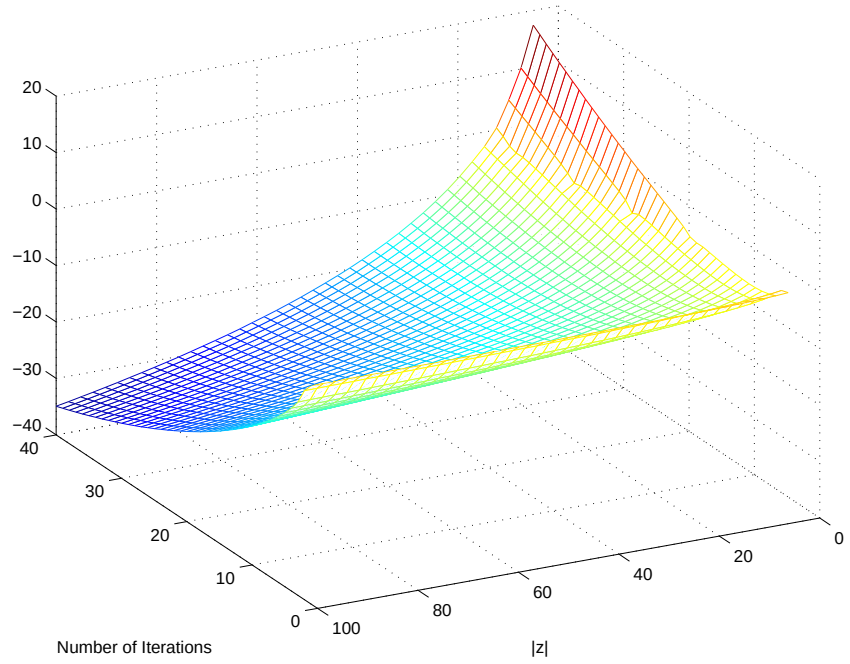

Figure 2: $\log_{10}(error)$ *in the asymptotic expansion of* $\Psi(a, c; z)$ *as a function of* $|z|$ *and the number of iterations*

The improvement in convergence as a function of x is illustrated in figures 3 and 4. These two plots demonstrate the number of iterations required for convergence with a desired error of 10^{-10} in the differential equation integrator for two different values of the c parameters, namely, $\{6.0, 14.5\}$, respectively. In figure 4, a maximum number of iterations (16384) is reached before the integration converges to the required error for the linear method.

Additionally, and more importantly, for small values of z, and $c > 1$, the path integration benefits greatly from implementation of the Kummer transformation (equation 13.1.29 in [4])

$$\Psi(a, c; z) = z^{1-c}\Psi(1 + a - c, 2 - c; z) \tag{28}$$

because $c > 1$ means $2 - c < 1$, and, therefore $\Psi(1 + a - c, 2 - c; z)$ has a finite value at $z = 0$. The differential equation integrator has a much easier time computing the finite value at zero.

We illustrate the domain of computation of $\Psi(3, n+2; x)$ in figure 5. For large values of x, the asymptotic expansion is used, while for small values, the differential equation

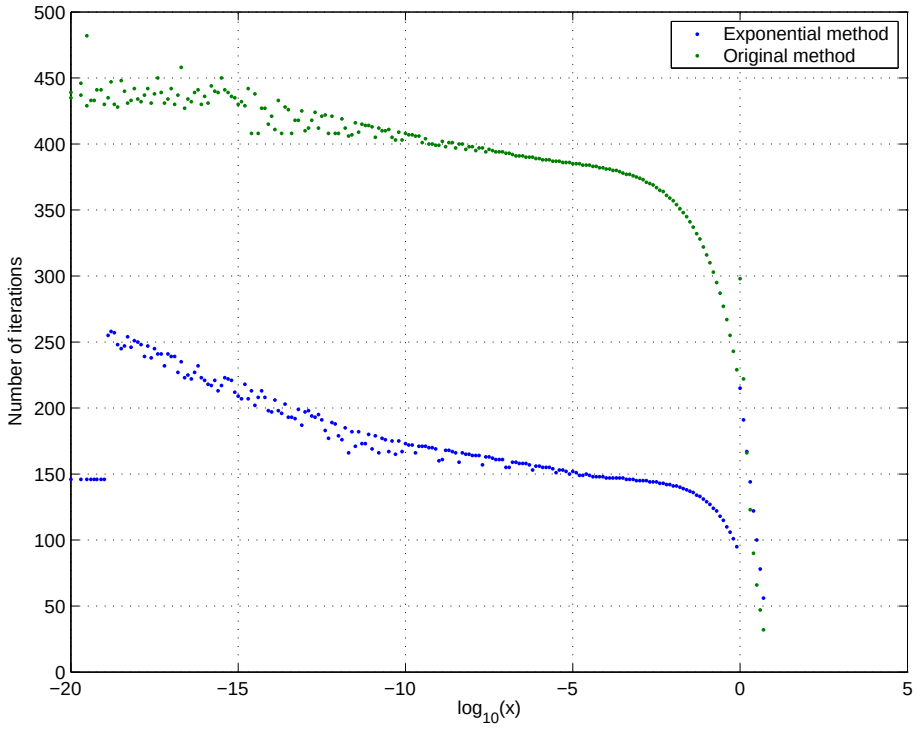

Figure 3: *Number of iterations required for Runge-Kutta-Felhberg stepper to reach accuracy of* 10^{-10} *for exponential and linear methods. Iterations plotted as a function of* $\log_{10}(x)$, $a = 3$, $c = 6.0$.

integrator of the logarithm of $\Psi(3, n+2; x)$ is used. If $x < 1$, then (28) is first used.

2.5 Summary

We numerically compute the Kummer hypergeomtric function by using path integration of the transformed differential equation of $\Psi(3, n+2; x)$, (27). The integration path links the initial value computed using an asymptotic expansion to the desired smaller value. This choice of orientation of the integration path is such that error does not unduly accumulate. For small values of x, the Kummer transformation is first applied.

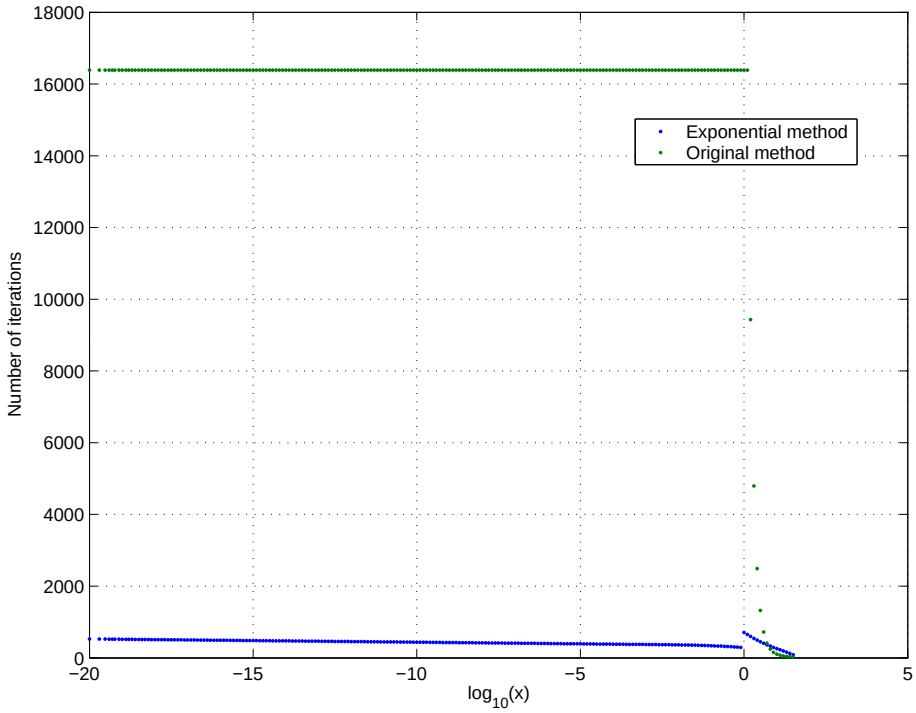

Figure 4: *Number of iterations required for Runge-Kutta-Fehlberg stepper to reach accuracy of* 10^{-10} *for exponential and linear methods. Iterations plotted as a function of* $\log_{10}(x)$, $a = 3$, $c = 14.5$. **Note: 16384 iterations means the desired accuracy was not achieved.**

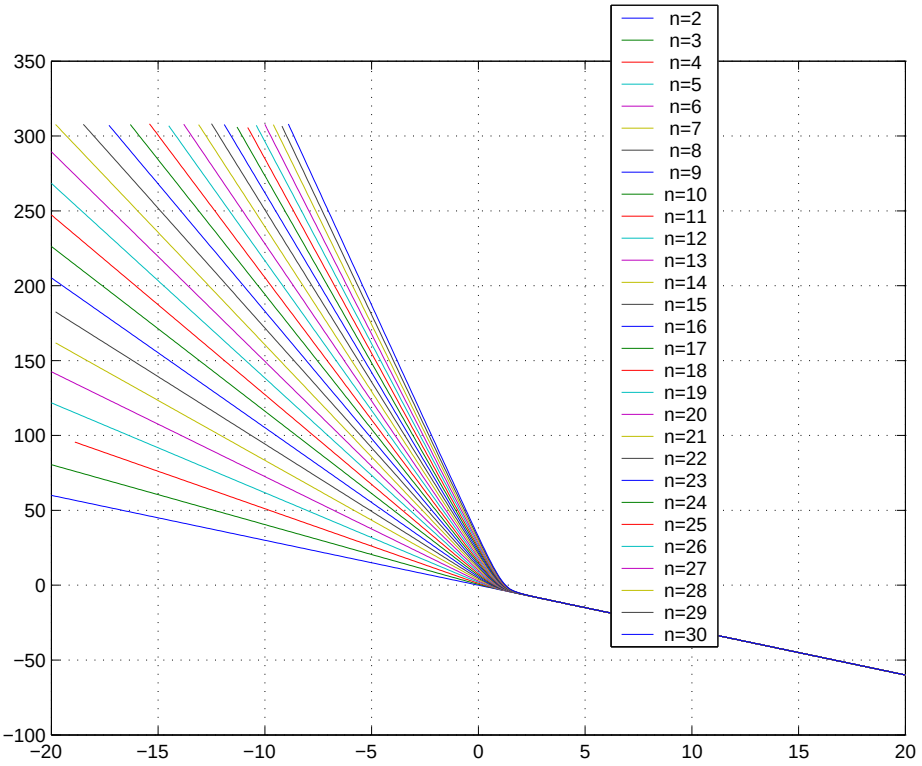

Figure 5: *Domain of numerical computation of* $\Psi(3, n+2; z)$ *as a function of* $\log_{10}(x)$ *for* $n \in \{2, 30\}$.

3 Numerical approximation of a special case of the Gauss hypergeometric function

This section demonstrates how to approximate a special case of the Gauss hypergeometric function by using the method of steepest descents.

3.1 Background on ${}_2\mathrm{F}_1(a, b; c; z)$

The Gauss hypergeometric function solves the differential equation

$$z(1-z)\frac{\mathrm{d}^2y}{\mathrm{d}z^2} + [c-(a+b+1)z]\frac{\mathrm{d}y}{\mathrm{d}z} - aby = 0. \tag{29}$$

From the differential equation one recognises three regular singular points at $z = 0$, $z = 1$ and $z = \infty$. Incidentally, the confluent hypergeometric function discussed in the previous section is related to the Gauss hypergeometric function by formally allowing the singular point at $z = 1$ to merge with the singular point at $z = \infty$, see [7]. The solution to (29) is denoted by ${}_2\mathrm{F}_1(a, b; c; z)$, and is given by the infinite series

$${}_2\mathrm{F}_1(a, b; c; z) = \sum_{k=0}^{\infty} \frac{(a)_k (b)_k}{(c)_k} \frac{z^k}{k!} \tag{30}$$

The Gauss hypergeoemetric function arises in many branches of physical science, from quantum field theory to computational chemistry to probability and statistics.

3.2 A special case of the Gauss hypergeometric function

The special case of the Gauss hypergeometric function considered in this report is given in formula 6.6.21.3 in [3] where it is therein related to an infinite integral through

$${}_2\mathrm{F}_1\left(\mu+\nu-1, \nu-1/2; \mu+1/2; \frac{A+B}{A-B}\right) = C\int_0^{\infty} x^{\mu-1}e^{Ax}\mathrm{K}_{\nu-1}(Bx)\mathrm{d}x, \tag{31}$$

where

$$C = \frac{(B-A)^{\mu+\nu-1}}{\sqrt{\pi}(2B)^{\nu-1}} \frac{\Gamma(\mu+1/2)}{\Gamma(\mu+\nu-1)\Gamma(\mu-\nu+1)}, \tag{32}$$

and where $\mathrm{Re}(\mu) > |\mathrm{Re}(\nu-1)|$, and $\mathrm{Re}(B-A) > 0$. On the right side, in the integral, one can make the substitution $u = Bx$ to get

$$\begin{aligned} I(D) &= \frac{C}{B^{\mu}}\int_0^{\infty} u^{\mu-1}e^{Du}\mathrm{K}_{\nu-1}(u)\mathrm{d}u, \\ &= \frac{(1-D)^{\nu+\mu-1}\Gamma(\mu+1/2)}{\sqrt{\pi}2^{\nu-1}\Gamma(\mu+\nu-1)\Gamma(\mu-\nu+1)}\int_0^{\infty} u^{\mu-1}e^{Du}\mathrm{K}_{\nu-1}(u)\mathrm{d}u, \end{aligned} \tag{33}$$

where $D = A/B$. In the following, we shall use the method of steepest descents to approximate the infinite integral on the right side.

3.3 Method of steepest descents

The method of steepest descents, as presented in [2], benefits from a change in variables that extends the lower limit of integration out to $-\infty$. Thus, we make the substitution[2] $u = e^v$. Let us write the integral as

$$I(D) = \frac{C}{B^{n+1}} \int_{-\infty}^{\infty} \exp(v\mu + De^v + \ln \mathrm{K}_{\nu-1}(e^v))\mathrm{d}v. \tag{34}$$

Consider the exponent function given by

$$g(v) = v\mu + De^v + \ln \mathrm{K}_{\nu-1}(e^v). \tag{35}$$

To implement the method, we must find the local maxima of $g(v)$ and the contours through these points for which $g(v)$ has a complex component that remains, mainly, constant. Since we only consider $g(v) \in \mathbb{R}$, we can ignore the orientation of the integration path. From formula 8.486.11 and 8.486.10 in [3], the first derivative of g is given by

$$\begin{aligned} \frac{\mathrm{d}g(v)}{\mathrm{d}v} &= De^v - e^v \frac{\mathrm{K}_{\nu-2}(e^v) + \mathrm{K}_{\nu}(e^v)}{2\mathrm{K}_{\nu-1}(e^v)} + \mu \\ &= De^v - \frac{e^v \mathrm{K}_{\nu}(e^v)}{\mathrm{K}_{\nu-1}(e^v)} + \mu + \nu - 1 \\ &= De^v + \mu + \nu - 1 - q(v). \end{aligned} \tag{36}$$

Further differentiation reveals

$$\frac{\mathrm{d}^{m+1}g(v)}{\mathrm{d}v^{m+1}} = De^v - \frac{\mathrm{d}^m q(v)}{\mathrm{d}v^m}, \quad m \in \{1, 2, \ldots\}. \tag{37}$$

Conveniently, the remarkable recurrence relation

$$\frac{\mathrm{d}^{m+1}q(v)}{\mathrm{d}v^{m+1}} = \sum_{k=0}^{m} \binom{m}{k} \frac{\mathrm{d}^{m-k}q(v)}{\mathrm{d}v^{m-k}} \frac{\mathrm{d}^k q(v)}{\mathrm{d}v^k} - (2\nu - 2)\frac{\mathrm{d}^m q(v)}{\mathrm{d}v^m} - 2^m e^{2v}, \tag{38}$$

[2] The method of steepest descents applied to the integral representation of $\Gamma(n)$ leads to Stirling's formula which increases in accuracy as n increases. If the transformation $u = e^x$ is used to transform the integral representation of $\Gamma(n)$, one again obtains Stirling's formula; however, it is the Stirling approximation to $\Gamma(n+1)/n$ which, as we have said, is more accurate.

permits numerical computation of all derivatives simply from $q(v)$. A proof of the recurrence relation is provided in appendix B.

Appendix C shows that $\exists$ only one local maximum of $g(v)$ when (31) is finite. This is important because if there were a second local maximum of $g(v)$, we would need to add its contribution to the overall integral. Let the position of this maximum be denoted as v_0 and expand $g(v)$ around v_0 to get

$$g(v) = g(v_0) + 0 + \frac{g^2(v_0)(v-v_0)^2}{2!} + \frac{g^3(v_0)(v-v_0)^3}{3!} + \dots \tag{39}$$

The integral in (34) can now be written as

$$\begin{aligned} I(D) &= \frac{C}{B^{n+1}} e^{g(v_0)} \int_{-\infty}^{\infty} \exp(\frac{g^2(v_0)(v-v_0)^2}{2!} + \frac{g^3(v_0)(v-v_0)^3}{3!} + \dots)\mathrm{d}v, \\ &= \frac{C}{B^{n+1}} e^{g(v_0)} \int_{-\infty}^{\infty} \exp(\frac{g^2(v_0)v^2}{2!} + \frac{g^3(v_0)v^3}{3!} + \dots)\mathrm{d}v, \\ &= \frac{C}{B^{n+1}} e^{g(v_0)} \int_{-\infty}^{\infty} e^{-a_2v^2+a_3v^3+a_4v^4+\cdots}\mathrm{d}v, \\ &= \frac{C}{B^{n+1}} e^{g(v_0)} \int_{-\infty}^{\infty} e^{-u^2}\frac{\mathrm{d}v}{\mathrm{d}u}\mathrm{d}u, \end{aligned} \tag{40}$$

where we have made the substitution $u^2 = a_2v^2 - a_3v^3 - a_4v^4 - \dots$, and $a_2 = -g^2(v_0)/2!$, $a_k = g^k(v_0)/v!$ for $k > 2$. To compute the derivative of v with respect to u, we make use of series reversion. This technique is reviewed and executed in appendix D to yield the following first five terms in the expansion $v = b_1u + b_2u^2 + b_3u^3 + \dots$

$$\begin{aligned} b_1 &= \frac{1}{\sqrt{a_2}}, \\ b_2 &= \frac{1}{2}\frac{a_3}{{a_2}^2}, \\ b_3 &= \frac{5}{8}\frac{{a_3}^2}{{a_2}^{7/2}} + \frac{1}{2}\frac{a_4}{{a_2}^{5/2}}, \\ b_4 &= \frac{{a_3}^3}{{a_2}^5} + \frac{3}{2}\frac{a_4\,a_3}{{a_2}^4} + \frac{1}{2}\frac{a_5}{{a_2}^3}, \\ b_5 &= \frac{231}{128}\frac{{a_3}^4}{{a_2}^{13/2}} + \frac{63}{16}\frac{a_4\,{a_3}^2}{{a_2}^{11/2}} + \frac{7}{4}\frac{a_5\,a_3}{{a_2}^{9/2}} + \frac{7}{8}\frac{{a_4}^2}{{a_2}^{9/2}} + \frac{1}{2}\frac{a_6}{{a_2}^{7/2}}. \end{aligned} \tag{41}$$

After substituting our known values for the a_i in terms of the derivatives $g^m(v_0)$,

written as g_m, we find that

$$b_1 = 2\,\frac{1}{\sqrt{-2\,g_2}}, \tag{42}$$

$$b_2 = 1/3\,\frac{g_3}{{g_2}^2}, \tag{43}$$

$$b_3 = -\frac{5}{36}\,\frac{\sqrt{2}{g_3}^2}{{g_2}^3\sqrt{-g_2}} + 1/12\,\frac{\sqrt{2}g_4}{{g_2}^2\sqrt{-g_2}}, \tag{44}$$

$$b_4 = -\frac{4}{27}\,\frac{{g_3}^3}{{g_2}^5} + 1/6\,\frac{g_3\,g_4}{{g_2}^4} - 1/30\,\frac{g_5}{{g_2}^3}, \tag{45}$$

$$b_5 = \frac{77}{864}\,\frac{\sqrt{2}{g_3}^4}{{g_2}^6\sqrt{-g_2}} - \frac{7}{48}\,\frac{\sqrt{2}g_4\,{g_3}^2}{{g_2}^5\sqrt{-g_2}} + \frac{7}{180}\,\frac{\sqrt{2}g_5\,g_3}{{g_2}^4\sqrt{-g_2}} + \frac{7}{288}\,\frac{\sqrt{2}{g_4}^2}{{g_2}^4\sqrt{-g_2}} - \frac{1}{180}\,\frac{\sqrt{2}g_6}{{g_2}^3\sqrt{-g_2}}. \tag{46}$$

Since the exponential in the integral in (40) is an even function, the only terms that survive integration are the b_{2k+1}. In these cases, we must evaluate

$$I(D) = \frac{C}{B^{n+1}} e^{g(v_0)} \int_{-\infty}^{\infty} \sum_{k=0}^{\infty} b_{2k+1}(2k+1) u^{2k} e^{-u^2} \mathrm{d}u. \tag{47}$$

The even moments over the Gaussian measure are tabulated in [3] and given by

$$\int_{-\infty}^{\infty} u^{2k} e^{-u^2} \mathrm{d}u = \begin{cases} \sqrt{\pi} & \text{if } k = 0, \\ \frac{\sqrt{\pi}}{2} & \text{if } k = 1, \\ \frac{\sqrt{\pi}}{2^{k+1}} \frac{\Gamma(2k)}{\Gamma(k)} & \text{otherwise,} \end{cases} \tag{48}$$

thus, after switching the order of integration and summation

$$I(D) \approx \hat{I}(D) = \frac{C}{B^{n+1}} e^{g(v_0)} \left[\sqrt{\pi} b_1 + \frac{3\sqrt{\pi}}{2} b_3 + \frac{15\sqrt{\pi}}{4} b_5 + \ldots \right]. \tag{49}$$

3.3.1 Normalising the approximation

To improve the approximation, we can normalise $\hat{I}(D)$ to $I(D)$ by comparing the two functions at an easily computed value. When $D = 0$, on can use formula 6.56.16 in [3],

$$\int_0^{\infty} x^{\mu-1} \mathrm{K}_{\nu}(x) \mathrm{d}x = 2^{\mu-2} \Gamma\left(\frac{\mu+\nu-1}{2}\right) \Gamma\left(\frac{\mu-\nu+1}{2}\right), \tag{50}$$

to perform such a normalisation. The approximation is then transformed by

$$\hat{I}(D) \to \hat{I}(D) \frac{I(0)}{\hat{I}(0)}. \tag{51}$$

3.4 Some Results

Table table 1 compares the numerical approximation to results obtained using Maple™. The accuracy of the approximation degrades slightly as $D \rightarrow 1$ However, the approximation yields good results even for large values of μ and ν.

Table 1: *Gauss hypergeometric function: approximation versus Maple*

μ	ν	**D**	**Approximation**	**Maple**
17	16	0.999	3.696e-59	3.715 e-59
17	16	0.9	7.739e-28	7.738e-28
31	30	0.9	9.961e-54	9.965e-54
31	30	0.999	2.155e-113	2.161e-54
5	4	0.9	3.826e-6	3.350e-6
5	4	0.999	3.319e-13	3.195e-13
61	60	-19	2.841e+117	2.840e+117

3.5 Summary

The method of steepest descents is used to numerically approximate a special case of the Gauss hypergeometric function. The expansion using series reversion provides a very good and rapid estimation of the function value. The root find in the method of steepest descents depends on the function $\mathrm{K}_\nu(x)$ which can be computed by using the method of section 2 and (8). Software that can compute the series reversion coefficients at run-time could possibly be used to improve the accuracy of the approximation[3]. The convergence of the approximation and the associated error as the number of terms increases poses an open question for future investigation.

[3]The approximation may diverge if the series reversion terms diverge as in some asymptotic expansions.

4 Constant false alarm thresholds for SAR GMTI

A multi-antenna SAR can be used for GMTI. A SAR-GMTI system measures the superposition of clutter and moving target echoes and passes the digitised measurements to a GMTI processor. The processor analyses the data and automatically tests for and extracts moving target signals.

Ideally, given profiles of measurements from a moving target and measurements from clutter, a GMTI processor automatically determines whether a sample most likely represents a moving target or clutter. Unfortunately, although one can construct a clutter profile, the complexity of creating a moving target profile prohibits such a processor. An alternative processor determines whether a given sample fits the clutter profile or not. Those samples not declared to be clutter are, by exclusion, moving targets. These two automatic tests are called Baysian and Neyman-Pearson tests, respectively.

This section solves the numerical challenges of implementing a Neyman-Pearson test using a CFAR scheme for the SAR-GMTI problem. In a CFAR scheme the profile of the clutter is provided through a probability distribution, $F_X(x)$, and a threshold, γ, is chosen such that if the sample under test exceeds the threshold, then it is unlikely that the sample represents clutter. The probability of a clutter sample exceeding the threshold is called the false alarm probability, p_F. The mechanics of the test are illustrated in figure 6.

The sampled clutter signals are complex-valued random variables. The SAR-GMTI processor first transforms these signals into real-valued GMTI metrics that are tested using the Neyman-Pearson test. These metrics are functions of the collected two-antenna data and are designed such that the probability distribution of the moving targets, though not known, should have significant mass above the clutter CFAR threshold, thus enabling their detection. This report fully describes how to numerically compute the CFAR thresholds for three GMTI metrics, namely, DPCA, the ATI phase and the hyperbolic detector as defined in [8]. These three metrics are chosen for the tradeoffs in their strengths and weaknesses which depend on the statistical nature of the data under test. For instance, DPCA is an optimal detector when the data is homogeneous, which one rarely encounters, but might find over distributed terrain such as grassland and forest. The ATI phase detector outperforms DPCA when the terrain is highly heterogeneous as occurs in the urban environment and, depending on weather conditions, the marine environment. The hyperbolic detector has almost

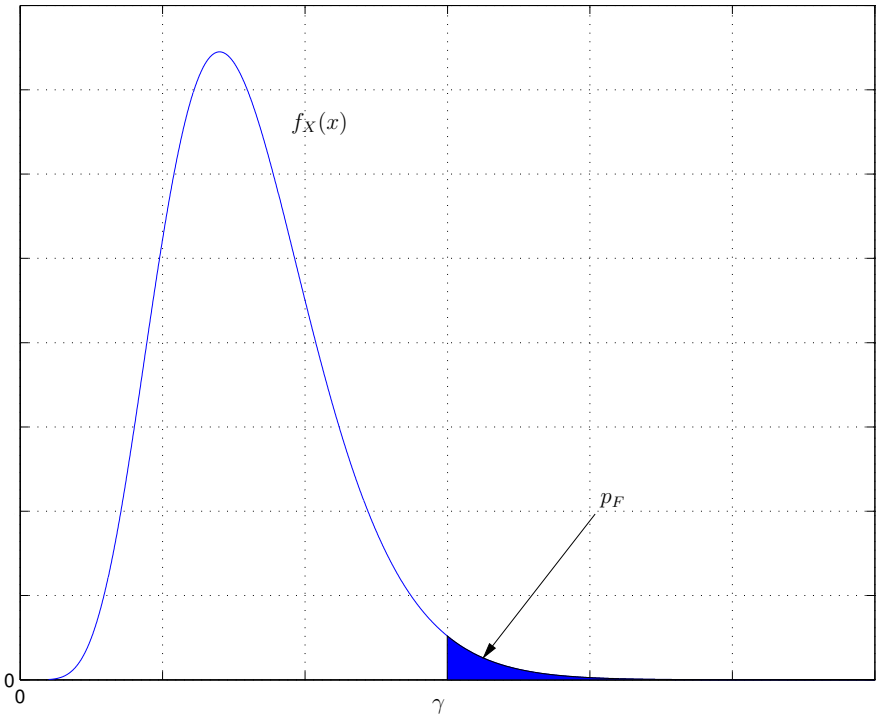

Figure 6: *CFAR test.*

the same performance characteristics in homogeneous terrain as DPCA, and outperforms the ATI phase in heterogeneous terrain. Over most types of terrain, which are moderately to extremely heterogeneous, the hyperbolic detector, which blends the characteristics of DPCA and ATI, will provide the best performance, see [8]. The SAR-GMTI processor for Radarsat-2 will use the computed thresholds and the metrics to automatically detect moving targets given a desired, arbitrary probability of false alarm.

4.1 Distributions for the GMTI metrics in homogeneous terrain

This section introduces the GMTI metrics and their probability distributions in homogeneous terrain.

4.2 The multi-looked sample covariance matrix

Let the sampled data from two antennas on a multi-aperture radar that are aligned in the along-track direction be denoted by

$$\begin{aligned} z_1(m_r/f_s, n_r/f_p) &\equiv z_1(m_r, n_r), \\ z_2(m_r/f_s, n_r/f_p) &\equiv z_2(m_r, n_r), \end{aligned} \tag{52}$$

where f_s is the complex pulse sampling frequency and f_p is the pulse repetition frequency. One usually says that the variables m_r and n_r denote raw samples in "fast-time" and "slow-time", respectively. It is assumed that measurements are made simultaneously by both antennas on the system. Hence, relative to the leading antenna, the trailing antenna provides a slow-time delayed version of the clutter signal, i.e., $z_2(m_r, n_r) \approx z_1(m_r, n_r + \Delta)$, where $\Delta = d/v_p$ with d denoting the distance between the antenna phase centres and v_p denoting the radar platform velocity in the direction of alignment of the antennas. From hereon, we assume that $z_2(m_r, n_r - \Delta)$ has been estimated or interpolated from $z_2(m_r, n_r)$, and we further write $z_2(m_r, n_r) \equiv z_2(m_r, n_r - \Delta)$. That is, for each n_r, $z_1(m_r, n_r)$ and $z_2(m_r, n_r)$ represent time displaced measuremets of the scene made from the same point in space. Now consider the matrix

$$\bar{\boldsymbol{R}}(m_r, n_r) = \begin{bmatrix} z_1(m_r, n_r)z_1^*(m_r, n_r) & z_1(m_r, n_r)z_2^*(m_r, n_r) \\ z_2(m_r, n_r)z_1^*(m_r, n_r) & z_2(m_r, n_r)z_2^*(m_r, n_r) \end{bmatrix}. \tag{53}$$

From $\bar{\boldsymbol{R}}(m_r, n_r)$ we can construct a sample correlation matrix given by

$$\hat{\boldsymbol{R}}(m_d, n_d) = \sum_{m_r, n_r = -\infty}^{\infty} h(m_r, n_r)\bar{R}\left([m_d + 1/2]D_m - m_r, [n_d + 1/2]D_n - n_r\right), \tag{54}$$

where D_m and D_n are downsample factors in the fast-time and slow-time directions respectively[4]. In practise, we choose $h(m_r, n_r)$ as a low-pass FIR filter in both m_r and n_r. For example, we might choose $h(m_r, n_r) = \mathrm{Rect}(m_r/D_m - 1/2)\mathrm{Rect}(n_r/D_n - 1/2)$, where

$$\mathrm{Rect}(x) = \begin{cases} 1 & \text{if } |x| < \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}. \tag{55}$$

The operation of low-pass filtering can be considered as a weighted spatial averaging. This introduces the concept of multi-looking. In this report, we say that **the sample correlation matrix is multi-looked by** n, where n denotes the number of statistically independent $\bar{\boldsymbol{R}}(m_r, n_r)$, that are included in the weighted average for each $\hat{\boldsymbol{R}}(m_d, n_d)$. The number n can be estimated using techniques described in [9].

[4] By downsampling, we mean that only every D^{th} sample is retained in the downsampled signal.

If it is assumed that the transfer functions of both antennas are the same, then the expected value of $\hat{\boldsymbol{R}}(m_d, n_d)$ can be written as

$$\boldsymbol{R}(m_d, n_d) = \sigma^2(m_d, n_d) \begin{bmatrix} 1 & \rho(m_d, n_d) \exp\left(-j\Phi(m_d, n_d)\right) \\ \rho(m_d, n_d) \exp\left(j\Phi(m_d, n_d)\right) & 1 \end{bmatrix}, \tag{56}$$

where $\rho(m_d, n_d) \exp\left(-j\Phi(m_d, n_d)\right)$ denotes the complex correlation coefficient between the two measurements, and $\sigma^2(m_d, n_d)$ denotes the power of each measurement. Figure 7 illustrates the procedure for computing the sample correlation matrix $\hat{R}(m_d, n_d)$. The assumption of terrain homogeneity translates into saying that

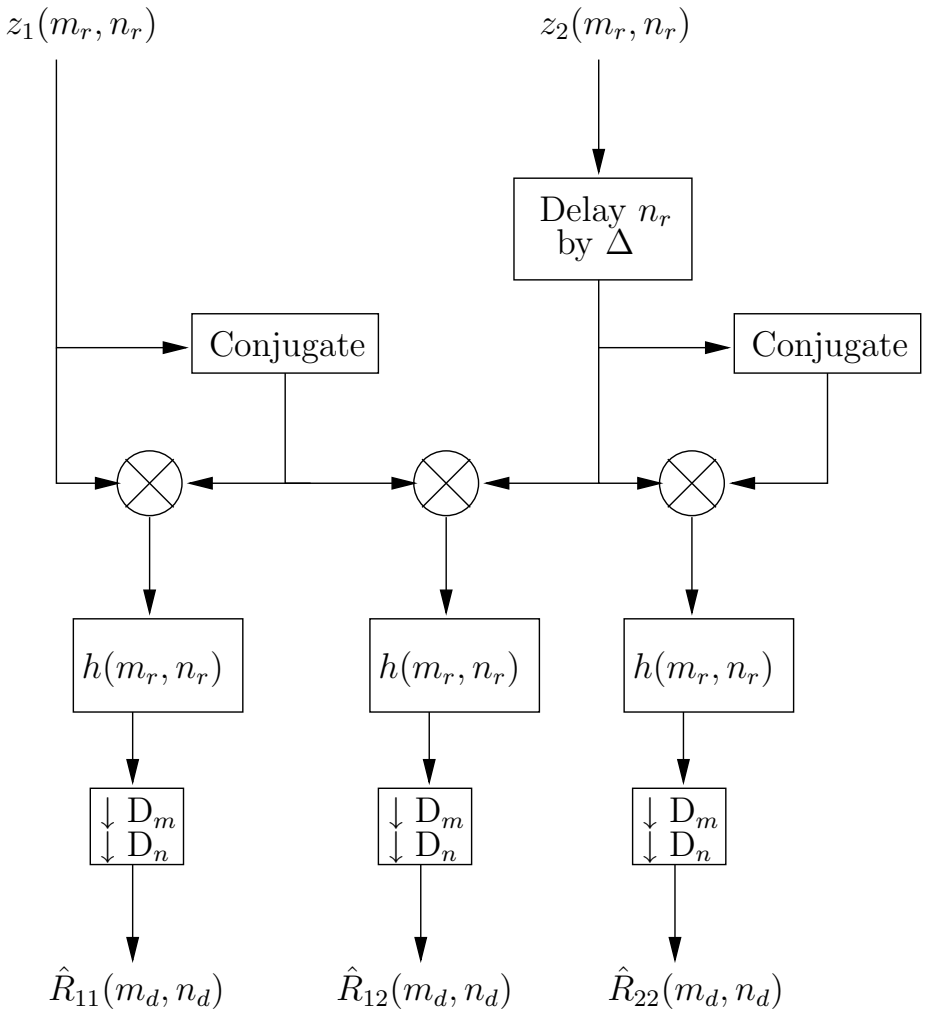

Figure 7: *Flow diagram for correlation matrix estimate.*

$\boldsymbol{R}(m_d, n_d) \equiv \boldsymbol{R}$, or that the terrain is stochastically stationary. It is assumed from here onwards that the measured data $z_i(m_r, n_r)$, $i \in \{1, 2\}$ are **stationary,**

zero-mean, circular Gaussian complex random variables as also assumed in [10; 8; 11]. Under this assumption, the correlation matrix is the same as the covariance matrix and can be written as

$$\boldsymbol{R} = \begin{bmatrix} \sigma^2 & \sigma^2\rho\exp(-j\Phi) \\ \sigma^2\rho\exp(j\Phi) & \sigma^2 \end{bmatrix}. \tag{57}$$

The larger of $\boldsymbol{R}$'s eigenvalues is $s_1 = \sigma^2(1+\rho)$, the smaller is $s_2 = \sigma^2(1-\rho)$. The sample covariance matrix and the true covariance matrix, $\hat{\boldsymbol{R}}$ and $\boldsymbol{R}$, respectively, are used to construct the GMTI metrics as outlined in the next three subsections.

4.2.1 ATI phase

The sample ATI phase random variable is defined as $\delta = \text{Arg}[\hat{R}_{12}(m_d, n_d)]$. The probability distribution for the along-track interferometric phase is given in [10] as

$$f_\Delta(\delta) = \frac{\Gamma(n+1/2)(1-\rho^2)^n\beta}{2\sqrt{\pi}\Gamma(n)(1-\beta^2)^{(n+1/2)}} + \frac{(1-\rho^2)^n}{2\pi}{}_2F_1(n,1;1/2;\beta^2), \tag{58}$$

for $\pi < \delta \leq \pi$, where $\beta = |\rho|\cos(\delta - \Phi)$, ρ and Φ are defined in (57), and ${}_2F_1(a,b;c;z)$ is the Gauss hypergeometric function, see [3] for a definition.

In addition to the phase distribution, the ATI magnitude distribution, defined as $\eta = |\hat{R}_{12}(m_d, n_d)|/\sigma^2$is also derived in [10] as

$$f_\eta(\eta) = \frac{4n^{n+1}\eta^n}{\Gamma(n)(1-\rho^2)} I_0\left(\frac{2n\eta\rho}{1-\rho^2}\right) K_{n-1}\left(\frac{2n\eta}{1-\rho^2}\right), \tag{59}$$

where I_0 and K_{n-1} are modified Bessel functions of the first and second kind, respectively, and of order zero and $n-1$, also respectively, as defined in [3]. Both the phase and amplitude distributions are derived by marginalising their joint distribution function given by

$$f_{\eta,\delta}(\eta,\delta) = \frac{2n^{n+1}\eta^n}{\pi\Gamma(n)(1-\rho^2)} e^{\left(\frac{2n\eta\rho\cos\delta}{1-\rho^2}\right)} K_{n-1}\left(\frac{2n\eta}{1-\rho^2}\right). \tag{60}$$

4.2.2 DPCA

DPCA is defined as a multi-looked quantity through

$$\begin{aligned}
y = & \sum_{m_r,n_r=-\infty}^{\infty} h(m_r,n_r)|z_1([m_d+1/2]D_m - m_r, [n_d+1/2]D_n - n_r) \\
& - z_2([m_d+1/2]D_m - m_r, [n_d+1/2]D_n - n_r)|^2, \\
= & \sum_{m_r,n_r=-\infty}^{\infty} h(m_r,n_r)\Bigg\{\bar{R}_{11}\Big([m_d+1/2]D_m - m_r, [n_d+1/2]D_n - n_r\Big) \\
& + \bar{R}_{22}\Big([m_d+1/2]D_m - m_r, [n_d+1/2]D_n - n_r\Big) \\
& - \bar{R}_{12}\Big([m_d+1/2]D_m - m_r, [n_d+1/2]D_n - n_r\Big) \\
& - \bar{R}_{21}\Big([m_d+1/2]D_m - m_r, [n_d+1/2]D_n - n_r\Big)\Bigg\}, \\
= & \hat{R}_{11}(m_d,n_d) + \hat{R}_{22}(m_d,n_d) - 2\text{Re}\{\hat{R}_{12}(m_d,n_d)\}.
\end{aligned} \tag{61}$$

The probability distribution for DPCA is computed in [12] as

$$f_Y(y) = \frac{y^{n-1}}{\Gamma(n)\sigma_\Delta^{2n}} \exp\left(-y/\sigma_\Delta^2\right), \tag{62}$$

where $\sigma_\Delta^2 = 2\sigma^2(1-\rho\cos\Phi)$.

4.2.3 Hyperbolic detector

To define the hyperbolic detector, we first define variables that represent an eigendecomposition of the sample covariance matrix

$$\Lambda_1(m_d,n_d) = \frac{1}{2}\left(\mathrm{Tr}\hat{\boldsymbol{R}}(m_d,n_d) + \sqrt{\mathrm{Tr}\hat{\boldsymbol{R}}^2(m_d,n_d) - 2|\hat{\boldsymbol{R}}(m_d,n_d)|}\right), \tag{63}$$

$$\Lambda_2(m_d,n_d) = \frac{1}{2}\left(\mathrm{Tr}\hat{\boldsymbol{R}}(m_d,n_d) - \sqrt{\mathrm{Tr}\hat{\boldsymbol{R}}^2(m_d,n_d) - 2|\hat{\boldsymbol{R}}(m_d,n_d)|}\right), \tag{64}$$

$$\Delta(m_d,n_d) = \arg(\hat{R}_{12}(m_d,n_d)), \tag{65}$$

$$\Theta(m_d,n_d) = \frac{1}{2}\arctan\left(\frac{2|\hat{R}_{12}(m_d,n_d)|}{\hat{R}_{11}(m_d,n_d) - \hat{R}_{22}(m_d,n_d)}\right). \tag{66}$$

The eigendecomposition is defined by

$$\begin{aligned}\hat{\boldsymbol{R}}(m_d,n_d) = &\begin{bmatrix}\cos\Theta(m_d,n_d) & e^{j\Delta(m_d,n_d)}\sin\Theta(m_d,n_d)\\ e^{-j\Delta(m_d,n_d)}\sin\Theta(m_d,n_d) & -\cos\Theta(m_d,n_d)\end{bmatrix}\\ &\cdot\begin{bmatrix}\Lambda_1(m_d,n_d) & 0\\ 0 & \Lambda_2(m_d,n_d)\end{bmatrix}\\ &\cdot\begin{bmatrix}\cos\Theta(m_d,n_d) & e^{j\Delta(m_d,n_d)}\sin\Theta(m_d,n_d)\\ e^{-j\Delta(m_d,n_d)}\sin\Theta(m_d,n_d) & -\cos\Theta(m_d,n_d)\end{bmatrix}.\end{aligned} \tag{67}$$

The hyperbolic detector is constructed through

$$u = \Lambda_2\left(\frac{1+\cos\Delta(m_d,n_d)\sin 2\Theta(m_d,n_d)}{2s_1} + \frac{1-\cos\Delta(m_d,n_d)\sin 2\Theta(m_d,n_d)}{2s_2}\right), \tag{68}$$

and its probability distribution is computed in [8] as

$$\begin{aligned}f_U(u) = \frac{2u^{2n-1}\Psi(3,n+2;u)}{\Gamma(n)\Gamma(n-1)(s_1-s_2)}\Big[&s_1 e^{-u\mathrm{Tr}\boldsymbol{R}/s_1}(s_2/s_1)^n\Psi(1,2n;u\mathrm{Tr}\boldsymbol{R}/s_1)\\ &- s_2 e^{-u\mathrm{Tr}\boldsymbol{R}/s_2}(s_1/s_2)^n\Psi(1,2n;u\mathrm{Tr}\boldsymbol{R}/s_2)\Big],\end{aligned} \tag{69}$$

where $\Psi(a,b;z)$ is a degenerate hypergeometric function of the second kind (formula 9.210 in [3]).

4.3 Distributions of the GMTI metrics in inhomogeneous terrain

In practice one invariably encounters limitations in the model of a stationary zero-mean Gaussian process. Although the stationary Gaussian model is still locally suitable, the process is not stationary when the background reflectivity (variance) changes from multilooked sample to multilooked sample. While it is still valid to assume that the mean of the process is zero and stationary, the variance must be modelled as globally non-stationary. For instance, the variance of the SAR data may be quite different for grassy terrain as compared to forested terrain. In a scene containing both types (and more) of terrain, a statistical description of the random variance must be included to validate the model.

4.4 The product model using a $\chi^{-2\kappa}$ distribution

To account for different variances from different terrain elements, one can introduce a random variable $A \in [0, \infty)$ such that $\hat{\boldsymbol{R}} \rightarrow A\hat{\boldsymbol{R}}$, where $\hat{\boldsymbol{R}}$ denotes the sample covariance matrix. This leads to the so-called product model. Many authors have studied appropriate statistical models for A including [13; 14; 11]. In [13], application of the product model leads the so-called K-distribution of clutter. This paper adopts the inverse chi-square model, χ^{-2}, of [11], with a generalisation to arbitrary powers, $\chi^{-2\kappa}$, $\kappa > 0$. The probability density function for χ^{-2} is given by [11]

$$f_A(a) = \frac{\Theta^\nu e^{-\Theta/a}}{\Gamma(\nu) a^{\nu+1}}, \quad a \in [0, \infty), \tag{70}$$

where Θ and ν are the shape and degree of freedom parameters, respectively. It also has a cumulative distribution function given by

$$F_A(a) = \frac{1}{\Gamma(\nu)} \Gamma(\nu, \Theta/a), \tag{71}$$

which can readily be computed from the characteristic function for the gamma function presented in [15]. Imposing the constraint $\mathcal{E}_A\{A\} = 1$[5] introduces a dependence between the two parameters of the distribution thereby, effectively, leaving only a single describing parameter. In the case of (70), the constraint yields $\Theta = \nu - 1$.

In our proposed generalisation of the above random variable, we make the transformation $A \rightarrow W = A^\kappa$ where $\kappa > 0$. The Jacobian introduced by this transformation is $\mathrm{d}w = \kappa a^{\kappa-1} \mathrm{d}a$ thus leading to a probability distribution for W given by

$$f_W(w) = \frac{(\nu-1)^\nu e^{-\frac{\nu-1}{w^{1/\kappa}}}}{\kappa \Gamma(\nu) w^{(\nu/\kappa)+1}}. \tag{72}$$

[5] $\mathcal{E}_A\{\cdot\}$ denotes expectation over the measure $\mathrm{d}F_A(a)$.

Generally, the theoretical distribution generated assuming homogeneous conditions predicts fewer high valued samples than are actually measured. The purpose of the random variable, W, is thus to elongate the theoretical probability distribution.

The introduction of the product model complicates the evaluation of closed form solutions of the theoretical probability distributions. However, closed form solutions can be specified for the classical techniques of ATI and DPCA.

4.4.1 ATI phase distribution under the product model

For the ATI phase it is easy to verify that the common multiplier W for the real and imaginary components cancels out when the argument is computed, see [16; 14; 10; 13]. In other words, the ATI phase for the clutter is invariant against the multiplicative texture random variable and hence, determination of the CFAR threshold remains unchanged from the homogeneous case.

4.4.2 DPCA distribution under the product model

It is possible to compute a closed form special case solution to the modified DPCA distribution. We start with

$$Z = W \cdot Y = W \sum_{m=0}^{n-1} |Z_1(m) - Z_2(m)|^2. \tag{73}$$

For the special case of $\kappa = 1$, (73) can be calculated as

$$\begin{aligned} f_Z(z) &= \int_0^\infty \frac{1}{w} f_W(w) f_Y(\frac{z}{w})\, dw, \\ &= \frac{[(\nu-1)\sigma_\Delta^2]^\nu}{B(n,\nu)} \frac{z^{n-1}}{[(\nu-1)\sigma_\Delta^2 + z]^{n+\nu}}, \end{aligned} \tag{74}$$

where the Beta function $B(n,\nu) = \Gamma(n)\Gamma(\nu)/\Gamma(n+\nu)$. The r^{th} moment of Z is

$$\mathcal{E}_Z\{Z^r\} = [(\nu-1)\sigma_\Delta^2]^r \frac{B(n+r,\nu-r)}{B(n,\nu)} \qquad \nu > r, \tag{75}$$

which can be exploited to estimate the parameter ν within a given SAR data set, see [16]. The cumulative probability function can be derived as

$$F_Z(z) = \frac{z^n}{n[(\nu-1)\sigma_\Delta^2]^n} {}_2\mathrm{F}_1(n+\nu, n; n+1; -\frac{z}{(\nu-1)\sigma_\Delta^2}), \tag{76}$$

where ${}_2\mathrm{F}_1(a,b;c;z)$ is the Gauss hypergeometric function as defined in [3]. Using [17], the characteristic function can be expressed as

$$\begin{aligned}\phi_Z(s) = {}&{}_1\mathrm{F}_1(n, 1-\nu; -js(\nu-1)\sigma_\Delta^2) \\ &+ \frac{\Gamma(-\nu)(-js(\nu-1)\sigma_\Delta^2)^\nu}{\Gamma(n+\nu)} {}_1\mathrm{F}_1(n+\nu, 1+\nu; -js(\nu-1)\sigma_\Delta^2), \quad (77)\end{aligned}$$

which is valid for $\nu \in \mathbb{R}^+$ by analytic continuation.

4.4.3 The hyperbolic and semi-definite metric distributions under the product model

The model for semi-definite power dependent metrics, such as the hyperbolic metric, assumes statistical independence in the texture statistic. Let us designate an arbitrary semi-definite power dependent random variable as X. In the previous section we had $X = Y$, the random variable describing DPCA. We seek the probability distribution for $Z = WX$. Since $W > 0$ and $X > 0$,

$$F_Z(z) = \mathcal{E}_X\{F_W(z/X)\}. \quad (78)$$

By differentiation with respect to z, one finds that

$$f_Z(z) = \mathcal{E}_X\{f_W(z/X)\frac{1}{X}\}. \quad (79)$$

Computation of the threshold, γ_z, for a CFAR detector with false alarm rate P_F, requires solving $1 - F_Z(\gamma_z) = \mathrm{P}_F$.

We now focus on the chosen probability distribution for W, namely,

$$\begin{aligned}F_W(w) &= F_A(w^{1/\kappa}) \\ &= \frac{1}{\Gamma(\nu)}\Gamma(\nu, \Theta/(w^{1/\kappa})).\end{aligned} \quad (80)$$

Using (78), one finds that

$$F_Z(z) = \int_0^\infty \frac{\Gamma(\nu, \Theta(x/z)^{1/\kappa})}{\Gamma(\nu)} \mathrm{d}F_X(x), \quad (81)$$

where, for the cases that we generally consider, $\mathrm{d}F_X(x) = f_X(x)\mathrm{d}x$. Evaluation of the CFAR threshold requires solving

$$\mathrm{P}_F = 1 - \int_0^\infty \frac{\Gamma(\nu, \Theta(x/\gamma_z)^{1/\kappa})}{\Gamma(\nu)} \mathrm{d}F_X(x) \quad (82)$$

for γ_z. With the condition that, [8],

$$\mathcal{E}_W\{W\} \equiv 1 \Rightarrow \Theta \equiv \left(\frac{\Gamma(\nu)}{\Gamma(\nu-\kappa)}\right)^{1/\kappa}, \tag{83}$$

(82) becomes

$$\mathrm{P}_F = 1 - \int_0^\infty \frac{\Gamma\left(\nu, \left[\frac{x\Gamma(\nu)}{\gamma_z\Gamma(\nu-\kappa)}\right]^{1/\kappa}\right)}{\Gamma(\nu)} \mathrm{d}F_X(x) \tag{84}$$

To solve (84) one requires an estimate of ν and κ. Estimation of these parameters is discussed in [8]. Once these parameters have been determined, numerical integration can be performed on (84) using the distribution for X where appropriate. For example, if X corresponds to the hyperbolic metric, we substitute (69) into (84) and solve for γ_z [8].

4.4.4 Numerically evaluating the infinite integral arising from the product model

This section outlines how to set a limit on the infinite integration domain required for computation of the CFAR threshold in (84). The idea stems from consideration of two positive functions, $f(x)$ and $g(x)$, both decreasing for $x > \zeta$ and $f(x) \leq x^{-p}$, for some p for $x > \zeta$

$$\begin{aligned}\int_0^\infty f(x)g(x)\mathrm{d}x &= \int_0^\zeta f(x)g(x)\mathrm{d}x + \int_\zeta^\infty f(x)g(x)\mathrm{d}x, \\ &\leq \int_0^\zeta f(x)g(x)\mathrm{d}x + \int_\zeta^\infty \frac{g(x)}{x^p}\mathrm{d}x, \\ &= \int_0^\zeta f(x)g(x)\mathrm{d}x + \frac{g(\zeta)}{\zeta^{p-1}(p-1)} \quad \text{if } p > 1.\end{aligned} \tag{85}$$

Given an error tolerance, ε, one can thus solve

$$\frac{g(\zeta)}{\zeta^{p-1}(p-1)} = \varepsilon \tag{86}$$

for ζ.

Now consider the expression in (84). The incomplete gamma function in the integrand can be written as

$$\Gamma\left(\nu, Cx^{1/\kappa}\right) = \int_{Cx^{1/\kappa}}^\infty \frac{t^{\nu-1}}{e^t}\mathrm{d}t, \tag{87}$$

where

$$C = \left[\frac{\Gamma(\nu)}{\zeta\Gamma(\nu-\kappa)}\right]^{1/\kappa}. \tag{88}$$

The incomplete gamma function is a monotonically decreasing function on the real line. By definition, the probability measure $\mathrm{d}F_X(x)$ also becomes monotonically decreasing for x large enough. With the substitution $t = uCx^{1/\kappa}$, (87) becomes

$$\begin{aligned}\Gamma\left(\nu, Cx^{1/\kappa}\right) &= (Cx^{1/\kappa})^\nu \int_1^\infty \frac{u^{\nu-1}}{e^{uCx^{1/\kappa}}}\mathrm{d}u, \\ &= (Cx^{1/\kappa})^\nu \int_1^\infty \frac{u^{\nu-1}}{1 + [uCx^{1/\kappa}]/1! + [uCx^{1/\kappa}]^2/2! + \ldots}\mathrm{d}u, \\ &< (Cx^{1/\kappa})^\nu \int_1^\infty \frac{u^{\nu-1}}{[uCx^{1/\kappa}]^p/p!}\mathrm{d}u, \\ &= C^{\nu-p}x^{\frac{\nu-p}{\kappa}}p!(p-\nu) \quad \text{if } \nu < p.\end{aligned} \tag{89}$$

One can thus see that

$$\begin{aligned}
&\int_{\zeta}^{\infty} \frac{\Gamma\left(\nu, Cx^{1/\kappa}\right)}{\Gamma(\nu)} \mathrm{d}F_X(x), \\
&< \frac{C^{\nu-p} p!(p-\nu) f_X(\zeta)}{\Gamma(\nu)} \int_{\zeta}^{\infty} x^{\frac{\nu-p}{\kappa}} \mathrm{d}x, \\
&= \frac{C^{\nu-p} p!(p-\nu)(p-\nu-\kappa) f_X(\zeta)}{\kappa\Gamma(\nu)} \zeta^{\frac{\kappa+\nu-p}{\kappa}} \quad \text{if } p > \nu+\kappa.
\end{aligned} \tag{90}$$

The above states that if one desires to compute the integral in (84) to an accuracy given by ε, then the infinite limit in (84) can be replaced by a finite integral with the upper integration limit, ζ, where ζ is given by the solution of

$$\varepsilon = \frac{p!(p-\nu)(p-\nu-\kappa) f_X(\zeta)}{\kappa\Gamma(\nu)} \zeta \left[\frac{\Gamma(\nu)\zeta}{\delta_z \Gamma(\nu-\kappa)}\right]^{\frac{\nu-p}{\kappa}}, \quad \text{where } p > \nu+\kappa. \tag{91}$$

4.5 ATI phase distribution evaluation

Although the ATI phase distribution does not change under heterogeneity, it is, nevertheless, difficult to compute because of the ${}_2\mathrm{F}_1(a,b;c;z)$ function in its definition, particularly when $\rho \to 1$, the case best suited to GMTI. That ${}_2\mathrm{F}_1(a,b;c;z)$ is challenging to numerically evaluate over a large domain is evidenced by the fact that the function is not implemented in Matlab™and that it only works sporadically in GSL™.

Rather than attempt to integrate the differential equation of ${}_2\mathrm{F}_1(a,b;c;z)$, which is possible, we suggest using the approximation of section 3. The aforementioned method can be used directly on the joint distribution of magnitude and phase given by (60). Marginalisation over η yields

$$f_\Delta(\Delta) = \frac{2n^{n+1}}{\pi\Gamma(n)(1-\rho^2)} \int_0^\infty \eta^n e^{\eta \frac{2n\rho\cos\delta}{1-\rho^2}} \mathrm{K}_{n-1}(\eta \frac{2n}{1-\rho^2}) \mathrm{d}\eta, \tag{92}$$

which, by using (31), yields

$$f_\Delta(\Delta) = \frac{2n^{n+1}}{\pi\Gamma(n)(1-\rho^2)} \frac{{}_2\mathrm{F}_1(\mu+\nu-1, \nu-1/2; \mu+1/2; \frac{A+B}{A-B})}{C}, \tag{93}$$

where $\mu = n+1$, $\nu = n$, $A = 2n\rho\cos\delta/(1-\rho^2)$, $B = 2n/(1-\rho^2)$, and $D = A/B = \rho\cos\delta$.

The accuracy and speed afforded by the approximation proves sufficient for all practical purposes. Figure 8 illustrates how even the first term of (49) provides an extremely good approximation to the phase distribution function. For the purpose of evaluating numerical thresholds, the figure shows that even with only one term, the estimated probability of distribution would be, at worst, 94% correct.

Figure 9 illustrates how the above method offers improvement over currently available software. Of the software currently available, Mathematica™can compute the Gauss hypergeometric function using the method of [18]. Currently, Matlab™is unable to compute the Gauss hypergeometric function, and the GSL™library fails utterly in the considered case. This computation along with the above method for $n = 16$ and $\rho = 0.999$ produces figure 9. We clearly see that the Mathematica™routine fails, while the above method, using the first and second terms provides, at least, an approximation to the phase probability density function. By using a Matlab™quadrature integration routine, the approximate distribution integrates to 0.9949. Table 2 captures the estimated probability of false alarm for a number of thresholds using the presented method and compares these thresholds to those obtained by using GSL™in a domain where the GSL™Gauss hypergeometric function is valid. It is assumed that the GSL™results are exact.

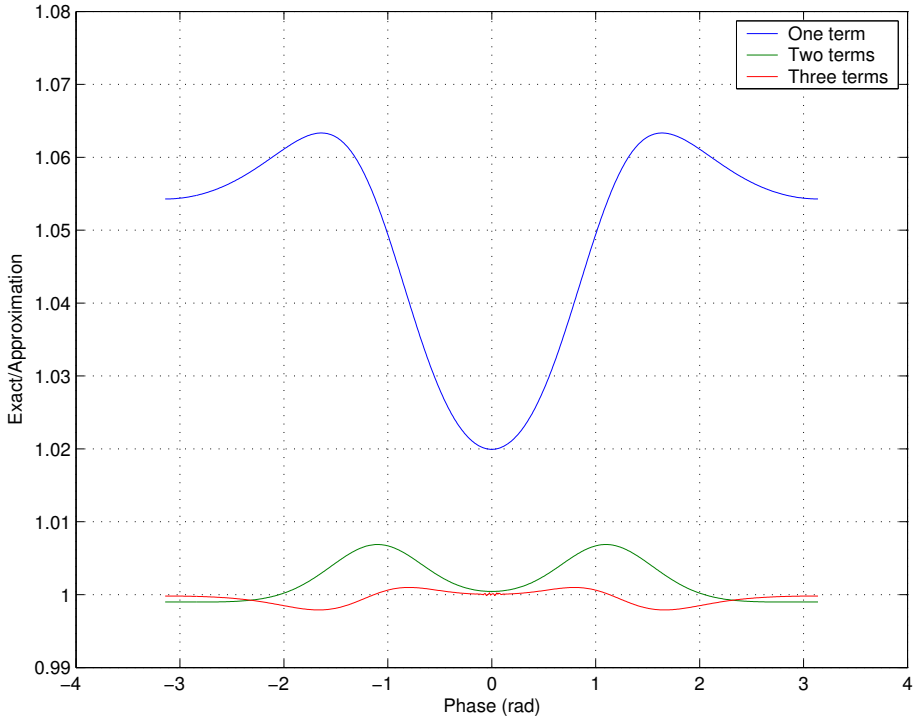

Figure 8: *Approximation to the ATI phase distribution for* $n = 4$, $\rho = 0.98$.

Table 2: *Estimated and exact probabilities of false alarm for* $n = 4$, $\rho = 0.98$ *for various thresholds.*

Threshold	**Exact**	**Estimated**
5.364100000000001e-01	1.000940000000000e-04	1.000200000000000e-04
7.774870000000000e-01	1.000550000000000e-05	9.999400000000000e-06
1.202250000000000e+00	1.000510000000000e-06	1.001700000000000e-06
2.243710000000000e+00	1.000200000000000e-07	1.000633395689900e-07
3.030270000000000e+00	9.999649999999999e-09	1.000147144229164e-08
3.130430000000000e+00	9.999969999999999e-10	9.999144361695851e-10
3.140480000000000e+00	1.000110000000000e-10	9.966495278034449e-11

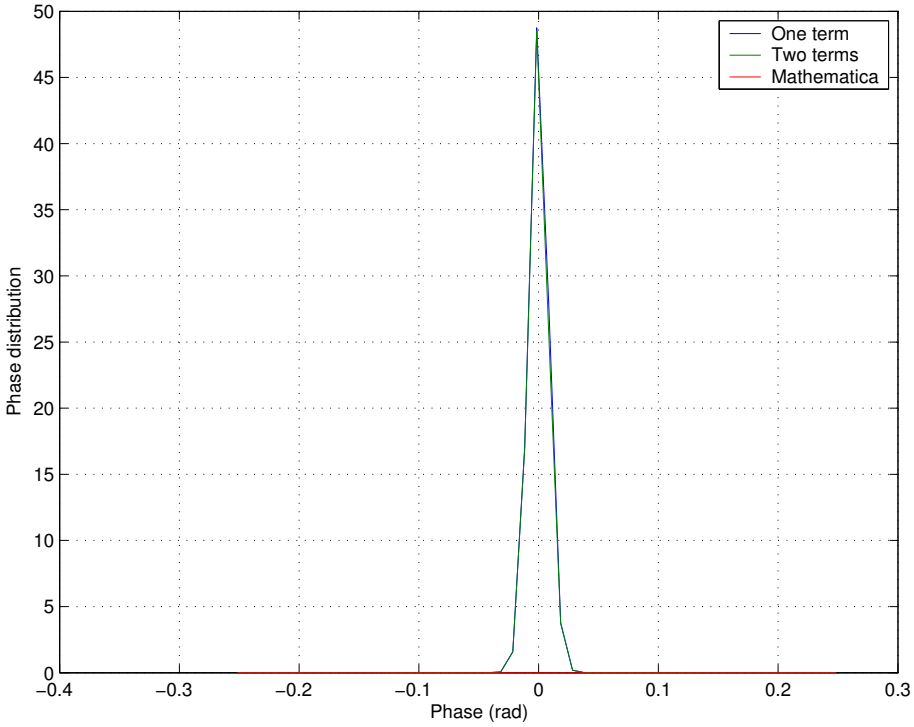

Figure 9: *ATI phase distribution for* $n = 16$, $\rho = 0.999$.

4.6 Numerical computation of thresholds for the hyperbolic detector

The main challenge of numerically computing the hyperbolic probability density function is in evaluating the Kummer hypergeometric function $\Psi(a, c; z)$. At the time of writing, the author is unaware of an open source library able to compute $\Psi(a, c; z)$ over a large domain in c and z, although the GSL™library has an implementation of severely limited utility. This fact is compounded by the observation that even commercially available software, such as Maple 9.1™and Matlab 7™fail to compute the function over a large enough domain. Specifically, a greatly conservative estimate of the domain of interest is $a \in \{1, 3\}$, $2 \leq n \leq 30 \rightarrow c \in (2, 32]$ and $Re(z) \in [10^{-20}, 10^{20}]$.

4.6.1 Quadrature integration for $\Psi(1, 2n; x_1) - \Psi(1, 2n; x_2)$

Using (6), we can rewrite (69) as

$$f_U(u) = \frac{2|\boldsymbol{R}|^n \Psi(3, n+2; u)}{\Gamma(n)\Gamma(n-1)(s_1 - s_2)\mathrm{Tr}(\boldsymbol{R})^{2n-1}} \left[\Gamma(2n-1, u\mathrm{Tr}(\boldsymbol{R})/s_1) - \Gamma(2n-1, u\mathrm{Tr}(\boldsymbol{R})/s_2)\right]. \quad (94)$$

Round-off error can accumulate in the computation of the difference of incomplete gamma functions, especially for small values of u and for large values of n. We thus use the definition of the incomplete gamma function, see [3], given as

$$\Gamma(\alpha, x) = \int_x^\infty e^{-t} t^{\alpha-1} \mathrm{d}t, \quad (95)$$

to rewrite (94) as

$$f_U(u) = \frac{2|\boldsymbol{R}|^n \Psi(3, n+2; u)}{\Gamma(n)\Gamma(n-1)(s_1 - s_2)\mathrm{Tr}(\boldsymbol{R})^{2n-1}} \int_{\mathrm{Tr}(\boldsymbol{R})u/s_2}^{\mathrm{Tr}(\boldsymbol{R})u/s_1} e^{-t} t^{2n-2} \mathrm{d}t. \quad (96)$$

The integration can be numerically achieved using standard quadrature integration techniques such as described in [5].

4.7 Summary

In summary, the computation of the Gauss hypergeometric function in the ATI phase distribution is replaced by the computation of a modified Bessel function when finding the root of (36). Once the root is found, the recurrence relation, (38), is used to

quickly compute higher order derivatives of $g(v)$ at the root, and the derivatives are used to estimate the ATI phase distribution. The computed ATI phase distribution very closely approximates the true ATI phase distribution. We note that formula (8) can be used to compute $\mathrm{K}_\nu(x)$.

We numerically compute (96) by using path integration of the transformed differential equation of $\Psi(3, n+2; u)$, (27), and quadrature integration of $e^{-t}t^{2n-2}$. To numerically compute P_F for a given threshold for the hyperbolic metric, we use (91) and (96), with a desired error bound, to determine a suitable replacement for the infinite limit in (84). We then use quadrature integration to compute the predicted false alarm probability. A binary search is used to find the threshold that corresponds to the desired P_F.

5 Summary

This report applies the elegant method of path integration of the differential equation to the problem of evaluating the Kummer hypergeometric function of the second kind. As well, it applies the method of steepest descents to the approximation to a special case of the Gauss hypergeometric function. The capabilities of these numerical routines are compared with the capabilities of Maple™, Mathematica™and GSL™. It is seen that the Kummer hypergeometric function can be computed in domains where the other software fails, and that the approximation to the special case of the Gauss hypergeometric function yields very good results.

The report applies the developed numerical routines to the problem of implementing SAR GMTI CFAR detector using the metrics of DPCA, ATI and the hyperbolic detector. It is demonstrated that the algorithms are adequate to compute the CFAR thresholds for all practical purposes, from false alarm rates of 10^{-4} down to smaller than 10^{-12}. These thresholds can be computed over a wide range in the statistical parameters describing the data, namely, the correlation coefficient, the number of looks and the degree of terrain non-stationarity.

This report also demonstrates how to implement a product model designed to accommodate terrain non-stationarity. As an ensemble the proposed algorithms can be used to realise a practical CFAR GMTI processor.

The contributions to the open literature include the new method to evaluate the Kummer hypergeometric function $\Psi(a, c; z)$, and the new method to approximate a special case of the Gauss hypergeometric function. In particular the report demonstrates the improvement offered by the proposed method for evaluating $\Psi(a, c; z)$ over currently available software, both commercial and open source. The ability to evaluate $\Psi(a, c; z)$ over a wide domain has application in many other fields in the physical sciences.

The report provides a C++ implementation of the developed algorithms.

References

[1] Temme, N. M. (1983), The numerical computation of the hypergeometric function $\mathrm{U}(a, b, z)$, *Numer. Math.*, 41, 63–82.

[2] Spiegel, M. R. (1964), Theory and Problems of Complex Variables, New York: Schaums.

[3] Gradshteyn, I. S. and Ryzhik, I. M. (2000), Table of Integrals, Series and Products, sixth ed, New York: Academic Press.

[4] Abramowitz, M. and Stegun, A. (1970), Handbook of Mathematical Functions, 9th ed, New York: Dover.

[5] Press, William H., Teukolsky, Saul A., and Vetterling, William T. (1988), Numerical Recipes in C: The Art of Scientific Computing, second ed, New York: Cambridge University Press.

[6] Hairer, E., Norsett, S. P., and Wanner, G. (2000), Solving Ordinary Differential Equations 1, Second ed, Vol. 1 of *Springer Series in Computational Mathematics*, Berlin: Springer.

[7] Luke, Y. L. (1969), The Special Functions and Their Approximations, Vol. I, New York and London: Academic Publishers.

[8] Sikaneta, I. C. (2004), Detection of Ground Moving Objects with Synthetic Aperture Radar, Ph.D. thesis, University of Ottawa.

[9] Gierull, C.H. and Sikaneta, I.C. (2002), Estimating the effective number of looks in interferometric SAR data, *Geoscience and Remote Sensing, IEEE Transactions on*, 40(8), 1733–1742.

[10] Lee, J. S., Miller, A. R., and Hoppel, K. W. (1994), Statistics of phase difference and product magnitude of multi–look processed Gaussian signals, *Waves in Random Media. IOP Publishing Inc. London UK*, 4, 307–317.

[11] Gierull, C. H. (2001), Statistics of SAR Interferograms with Application to Moving Target Detection, (DREO TR 2001-045) Defence Research Establishment Ottawa.

[12] Sikaneta, I. C. and Gierull, C. H. (2005), Two-channel SAR ground moving target indication for traffic monitoring in urban terrain, *Proc. of IEEE Joint Workshop of ISPRS & German Association for Pattern Recognition, 2005.*

[13] Tough, R. A. J., Blacknell, D, and Quegan, S. (1995), A statistical description of polarimetric and interferometric synthetic aperture radar data, *Proc. R. Soc. Lond., A. Math. Phs. Sci*, A(449), 567–589.

[14] Joughin, I. R., Winebrenner, D. P., and Percival, D. B. (1994), Probablility Density Functions for Multilook Polarimetric Signatures, *IEEE Trans. Geosci and Rem. Sens.*, 32(3), 562–574.

[15] Spiegel, M. R. (1994), Theory and Problems of Probability and Statistics, New York: McGraw-Hill.

[16] Gierull, C. H. (2001), Unbiased Coherence Estimator for SAR Interferometry with Application to Moving Target Detection, *Electronics Letters*, 37(14), 913–915.

[17] Phillips, P. C. B. (1982), The true characteristic function of the F-distribution, *Biometrica*, 69(1), 261–264.

[18] Forrey, R. C. (1997), Computing the Hypergeometric Function, *Journal of Comp. Phys.*, 137(2), 79–100.

This page intentionally left blank.

Annex A: Table of values for $\Psi(a, c; z)$

Table A.1 and table A.2 compare the computed values for $\Psi(a, c; z)$ with $a = 3$, $b = 4.2$ and $a = 3$, $b = 10.2$, respectively against the benchmark of Maple 9.1™. The proposed algorithm can compute the required values to the default accuracy of Maple 9.1™, and can also compute the function over a wider domain (for larger values of x). The extended domain of computation is achieved using the asymptotic expansion of (24).

Table A.1: *Comparison of Maple computed values to the proposed algorithm for* $a = 3$, $b = 4.2$

$\log_{10}(x)$	**Proposed Method** $\Psi(3, 4.2; x)$	**Maple:** $\Psi(3, 4.2; x)$
-20	1.211982740e+64	1.211982740e+64
-19	7.647094112e+60	7.647094112e+60
-18	4.824990194e+57	4.824990194e+57
-17	3.044363000e+54	3.044363000e+54
-16	1.920863194e+51	1.920863194e+51
-15	1.211982740e+48	1.211982740e+48
-14	7.647094112e+44	7.647094112e+44
-13	4.824990194e+41	4.824990194e+41
-12	3.044363000e+38	3.044363000e+38
-11	1.920863194e+35	1.920863194e+35
-10	1.211982740e+32	1.211982740e+32
-9	7.647094113e+28	7.647094112e+28
-8	4.824990198e+25	4.824990198e+25
-7	3.044363028e+22	3.044363027e+22
-6	1.920863369e+19	1.920863369e+19
-5	1.211983842e+16	1.211983842e+16
-4	7.647163629e+12	7.647163628e+12
-3	4.825428683e+09	4.825428683e+09
-2	3.047121537e+06	3.047121537e+06
-1	1.937806914e+03	1.937806914e+03
0	1.301054615e+00	1.301054615e+00
1	1.052472199e-03	1.052472199e-03
2	1.005906765e-06	1.005906765e-06
3	1.000599043e-09	Inf
4	1.000059990e-12	NaN
5	1.000006000e-15	NaN
6	1.000000600e-18	NaN
7	1.000000060e-21	NaN
8	1.000000006e-24	NaN
9	1.000000001e-27	NaN
10	1.000000000e-30	NaN
⋮	⋮	⋮
18	1.000000000e-54	NaN
19	1.000000000e-57	NaN
20	1.000000000e-60	NaN

Table A.2: *Comparison of Maple computed values to the proposed algorithm for* $a = 3$, $b = 10.2$

$\log_{10}(x)$	**Proposed Method:** $\Psi(3, 10.2; x)$	**Maple:** $\Psi(3, 10.2; x)$
-20	3.100538195e+188	3.100538195e+188
-19	1.956307346e+179	1.956307346e+179
-18	1.234346488e+170	1.234346488e+170
-17	7.788199822e+160	7.788199822e+160
-16	4.914021878e+151	4.914021878e+151
-15	3.100538195e+142	3.100538195e+142
-14	1.956307346e+133	1.956307346e+133
-13	1.234346488e+124	1.234346488e+124
-12	7.788199822e+114	7.788199822e+114
-11	4.914021878e+105	4.914021878e+105
-10	3.100538195e+96	3.100538195e+96
-9	1.956307347e+87	1.956307347e+87
-8	1.234346497e+78	1.234346497e+78
-7	7.788200411e+68	7.788200411e+68
-6	4.914025593e+59	4.914025593e+59
-5	3.100561638e+50	3.100561638e+50
-4	1.956455267e+41	1.956455267e+41
-3	1.235280111e+32	1.235280111e+32
-2	7.847299337e+22	7.847299337e+22
-1	5.299294559e+13	5.299294559e+13
0	6.515246294e+04	6.515246294e+04
1	7.031410628e-03	7.031410627e-03
2	1.206765110e-06	1.206765110e-06
3	1.018794801e-09	Inf
4	1.001861936e-12	NaN
5	1.000186019e-15	NaN
6	1.000018600e-18	NaN
7	1.000001860e-21	NaN
8	1.000000186e-24	NaN
9	1.000000019e-27	NaN
10	1.000000002e-30	NaN
⋮	⋮	⋮
18	1.000000000e-54	NaN
19	1.000000000e-57	NaN
20	1.000000000e-60	NaN

This page intentionally left blank.

Annex B: A useful recurrence relation for the derivatives of $q(v)$ for the method of steepest descents

With

$$q(v) = \frac{e^v \mathrm{K}_n(e^v)}{\mathrm{K}_{n-1}(e^v)}, \tag{B.1}$$

and a temporary, sloppy, but useful change in notation, we can readily show that

$$q_0 = q(v), \tag{B.2}$$

$$q_1 = \frac{\mathrm{d}q(v)}{\mathrm{d}v} = q_0^2 - (2n-2)q_0 - e^{2v}. \tag{B.3}$$

The expression for q_1 arises from the differential relations in 8.486 in [3]. We see that the first derivative is simply a function of $q(v)$. Thus, so is the m^{th} derivative, written as $q_m(v)$, and we can immediately write

$$q_2 = 2(q_0 - n + 1)q_1 - 2e^{2v}, \tag{B.4}$$

$$q_3 = 2q_1^2 + 2(q_0 - n + 1)q_2 - 4e^{2v}, \tag{B.5}$$

$$\vdots \tag{B.6}$$

In fact the above can be written as

$$q_{m+1} = \sum_{k=0}^{m} \binom{m}{k} q_{m-k}q_k - (2n-2)q_m - 2^m e^{2v}. \tag{B.7}$$

Proof. The formula is seen to be correct for $m = 0, 1$ by direct substitution. Now consider

$$q_{m+2} = \sum_{k=0}^{m+1} \binom{m+1}{k} q_{m+1-k}q_k - (2n-2)q_{m+1} - 2^{m+1} e^{2v}. \tag{B.8}$$

The sum part evaluates to

$$\begin{aligned}
\sum_{k=0}^{m+1}\binom{m+1}{k}q_{m+1-k}q_k &= \sum_{k=0}^{m}\binom{m+1}{k}q_{m+1-k}q_k + q_0q_{m+1},\\
&= \sum_{k=0}^{m}\frac{m+1}{m+1-k}\binom{m}{k}q_{m+1-k}q_k + q_0q_{m+1},\\
&= \sum_{k=0}^{m}\left(1+\frac{k}{m+1-k}\right)\binom{m}{k}q_{m+1-k}q_k + q_0q_{m+1},\\
&= \sum_{k=0}^{m}\binom{m}{k}q_{m+1-k}q_k + \sum_{k=1}^{m}\left(\frac{k}{m+1-k}\right)\binom{m}{k}q_{m+1-k}q_k + q_0q_{m+1},\\
&= \sum_{k=0}^{m}\binom{m}{k}q_{m+1-k}q_k + \sum_{k=1}^{m}\binom{m}{k-1}q_{m+1-k}q_k + q_0q_{m+1},\\
&= \sum_{k=0}^{m}\binom{m}{k}q_{m+1-k}q_k + \sum_{k=0}^{m-1}\binom{m}{k}q_{m-k}q_{k+1} + q_0q_{m+1},\\
&= \sum_{k=0}^{m}\binom{m}{k}q_{m+1-k}q_k + \sum_{k=0}^{m}\binom{m}{k}q_{m-k}q_{k+1}.
\end{aligned} \tag{B.9}$$

Substitution into (B.8) yields

$$q_{m+2} = \sum_{k=0}^{m}\binom{m}{k}(q_{m+1-k}q_k + q_{m-k}q_{k+1}) - (2n-2)q_{m+1} - 2^{m+1}e^{2v}, \tag{B.10}$$

and examination shows that this is the derivative of q_{m+1}. Alternatively, the formula can be proved by using Leibniz's derivative formula, see [4], for the product of functions. □

To numerically evaluate the m^{th} derivative at v_0, we need only compute $q(v_0)$ and then use (B.7) sequentially.

Annex C: Proof of only one local maximum of $g(v)$ for the method of steepest descents

This section shows that when (31) is finite, then $g(v)$ has only one root.

Proof. The first derivative of $g(v)$, defined in (36), is given by

$$\frac{\mathrm{d}g(v)}{\mathrm{d}v} = De^v - \frac{e^v \mathrm{K}_n(e^v)}{\mathrm{K}_{n-1}(e^v)} + 2n, \tag{C.1}$$

where $D < 1$. For convenience, let us write the right side as

$$f(x) = Dx + 2n - xA(x), \tag{C.2}$$

where $x = e^v$ and

$$A(x) = \mathrm{K}_n(x)/\mathrm{K}_{n-1}(x). \tag{C.3}$$

Abramowitz and Stegun, [4], (9.6.24) provide an integral formula

$$\mathrm{K}_\nu(z) = \int_0^\infty e^{-z\cosh t}\cosh(\nu t)\mathrm{d}t, \quad |\arg(z)| < \frac{1}{2}\pi, \tag{C.4}$$

which, together with the fact that $\cosh x$ is an increasing function for $x > 0$, shows that $\mathrm{K}_n(x) > \mathrm{K}_{n-1}(x)$; hence, $A(x) > 1$ for $x > 0$. To compute $\lim_{v\to-\infty} g(v) = \lim_{x\to 0} f(x)$, one can use formula 9.6.9 in [4]

$$\lim_{x\to 0} \mathrm{K}_\nu(x) = \frac{\Gamma(\nu)}{2}\left(\frac{x}{2}\right)^{-\nu}. \tag{C.5}$$

Indeed,

$$\lim_{x\to 0} f(x) = 2n - 2(n-1) = 2. \tag{C.6}$$

Since $g(v)$ is a function in an exponential that integrates to a finite value, (34), we must have $\lim_{v\to\infty} g(v) = \lim_{x\to\infty} f(x) = -\infty$. Thus, $\exists$ at least one turning point, v_0, in $g(v)$ since, at $v = -\infty$, the derivative is positive.

Now for uniqueness. The derivative of $f(x)$ can be computed using formulas 8.4.86, [3] to give

$$f'(x) = D + x + 2(n-1)A(x) - xA^2(x). \tag{C.7}$$

Equating $f'(x)$ to zero reveals that at the turning points

$$\begin{aligned} (xA(x) - n + 1)^2 &= x^2 + Dx + (n-1)^2, \\ xA(x) &= \sqrt{x^2 + Dx + (n-1)^2} + (n-1). \end{aligned} \tag{C.8}$$

By using (C.5), we note that $f'(0) = D$. If there is more than one root of $f(x)$, then, since $f(0) > 0$, there must exist x_1 such that $f(x_1) \leq 0$ and $f'(x_1) = 0$. If this is the case, then, since $f(\infty) < 0$, there must exist $x_2 > x_1$ such that $f(x_2) \geq 0$ and $f'(x_2) = 0$. If $f(x_1) \leq 0$, then, using (C.2), $Dx_1 + 2n \leq x_1 A(x_1)$. This, in turn, using (C.8), implies that

$$\begin{aligned} Dx_1 + 2n &\leq \sqrt{x_1^2 + Dx_1 + (n-1)^2} + (n-1), \\ Dx_1 + n + 1 &\leq \sqrt{x_1^2 + Dx_1 + (n-1)^2}. \end{aligned} \tag{C.9}$$

On the other hand, if $f(x_2) \geq 0$, then

$$Dx_2 + n + 1 \geq \sqrt{x_2^2 + Dx_2 + (n-1)^2}. \tag{C.10}$$

By examining $f_1(x) = Dx + n + 1$, $f_2(x) = \sqrt{x^2 + Dx + (n-1)^2}$, we will see that the above two inequalities cannot be satisfied. Indeed, $f_1(0) = n + 1 > f_2(0) = n - 1$, thus, $\exists x_m$ such that for $x \in \mathcal{A} = (0, x_m]$, $f_1(x) \geq f_2(x)$, and $f_1(x_m) = f_2(x_m)$. For $x \in \mathcal{B} = [x_m, \infty)$, $f_2(x) \geq f_1(x)$. To satisfy (C.9), $x_1 \in \mathcal{B}$. To satisfy (C.10), $x_2 \in \mathcal{A}$. It is not possible that $x_2 > x_1$, $x_1 \in \mathcal{B}$ and $x_2 \in \mathcal{A}$; thus, there can only be one root. □

Annex D: Series Reversion

According to Wikipedia, this technique is attributed to Lagrange (1736-1813) and is called either the Lagrange inversion theorem or the Lagrane-Bürmann theorem. Suppose that $u(v)$ defined by

$$u^2 = a_2 v^2 - a_3 v^3 - a_4 v^4 + \ldots, \tag{D.1}$$

is analytic around v_0. Then, the inverse of u^2 can be written as

$$v = b_0 + b_1 u + b_2 u^2 + b_3 u^3 + \ldots, \tag{D.2}$$

where $v(u)$ is analytic around $v = u(v_0)$. By directly substituting (D.2) into (D.1), we see that

$$u^2 = a_2(b_0 + b_1 u + b_2 u^2 + b_3 u^3 + \ldots)^2 - a_3(b_0 + b_1 u + b_2 u^2 + b_3 u^3 + \ldots)^3 - \ldots, \tag{D.3}$$

and by equating coefficients, we find that

$$\begin{aligned} b_0 &= 0, \\ a_2 b_1^2 &= 1, \\ 2a_2 b_1 b_2 - a_3 b_1^3 &= 0, \\ &\vdots \end{aligned} \tag{D.4}$$

These equations allow us to solve, recursively, for b_i. The first five b_i, $i \in 1, \ldots 5$ are listed in (41).

This page intentionally left blank.

Annex E: Example C++ Code

This section lists the example C++ code used to numerically evaluate the CFAR threshold for the ATI phase, DPCA and the hyperbolic detector. The code applies the product model to account for terrain heterogeneity as described in section 4.3. A special function object that can more generally use the algorithm for computing $\Psi(a, c; z)$ is presented.

The code makes use the the GSL™library for numerical integration and for integration of the differential equation. Otherwise there are no code dependencies.

The first listing shows the class structure for the pdf object. The ATI, DPCA and hyperbolic PDF objects derive from it. The base class defines the texture statistic as the generalized inverse χ^{-2} of (71).

```
// Object to represent a GMTI pdf
// Will use p,n,sigma^2, e.t.c to represent parameters
// Will use gsl library

#include "myfft.hpp"
#include "specialfunction.h"
#include <math.h>
#include <iostream>
#include <gsl/gsl_integration.h>

using namespace std;

#ifndef _PDF_INCLUDED_
#define _PDF_INCLUDED_
const bool verbose = false;
class pdf{
private:
protected:
  //! The following are the statistical parameters
  double n;
  double v;
  double e1;
  double e2;
  double kp;

  //! X represents the random variable
  double X;

  //! x1 and x2 are starting point values for computing the thresholds
  double x1,x2;
  bool testCount(gint32 count, gint32 MAXITERS)
```

```
  {
    return (count < MAXITERS)? true : false;
  };
  void handleCountError()
  {
    cerr << "Maximum iterations reached. Exiting!" << endl;
    exit(1);
  }
public:
  pdf(const double e1a=20.0,
      const double e2a=2.0,
      const double na=4.9,
      const double va=6.1,
      const double ka=1.0);
  specialFunction sf;
  double findUpperIntegrationLimit();
  double inverseChiCDF(const double x);
  virtual double getThreshold(const double pf);

  static double wrapperToHomogeneousPDF(double x, void *);
  static double wrapperToHeterogeneousKernel(double x, void *);
  virtual double homogeneousPDF(const double x);
  double heterogeneousCDF(const double x);
  double homogeneousCDF(const double x);
  double homogeneousCDF(const double x0, const double x1);

  double upperIntegrationLimit;
  double integrationTolerance;
};
#endif
```

This listing shows the implementation of the pdf object class functions. The wrapperToFunction call makes the appropriate call at run time. These appropriate PDF functions are defined in the derived objects.

```
#include "pdf.hpp"
#include <gsl/gsl_sf.h>
#include <iomanip>

pdf::pdf(const double e1a,
         const double e2a,
         const double na,
         const double va,
         const double ka):
n(na),
v(va),
e1(e1a),
e2(e2a),
kp(ka){
    upperIntegrationLimit=1e6;
    integrationTolerance=1e-12;
    x1=1.0;
    x2=1000.0;
}

double pdf::wrapperToHeterogeneousKernel(double x, void *g){
  pdf *pdfptr = (pdf *)g;
  return pdfptr->homogeneousPDF(x)*pdfptr->inverseChiCDF(x);
}

double pdf::inverseChiCDF(const double x){
  return sf.GammaInc(v,pow(x/X*sf.Gamma(v)/sf.Gamma(v-kp),1.0/kp))/sf.Gamma(v);
}

double pdf::wrapperToHomogeneousPDF(double x, void *g){
  pdf *pdfptr = (pdf *)g;
  return pdfptr->homogeneousPDF(x);
}

double pdf::homogeneousPDF(const double x){
  return x*x;
}

double pdf::findUpperIntegrationLimit(){
    // Determine the upper integration limit to use for this threshold
    double C=pow(sf.Gamma(v)/sf.Gamma(v-kp)/X,1.0/kp);
    double p=floor(kp+v)+1.0;
    double D=1.0/sf.Gamma(v)*sf.Gamma(p+1.0)*(p-v)/pow(C,(p-v))*(p-v-kp)/kp;
    double dup=upperIntegrationLimit;
    double ddown=upperIntegrationLimit;
    gint32 iters = 2048;
```

```
    gint32 count = 0;

    // Find values above and below the desired root
    while(integrationTolerance<
          D*pow(dup,−(p−v−kp)/kp)*homogeneousPDF(dup)
          & count++ < iters)
    {
        dup*=2.0;
    }
    if(!testCount(count,iters)) handleCountError();
    count = 0;
    while(integrationTolerance>
          D*pow(ddown,−(p−v−kp)/kp)*homogeneousPDF(ddown) & count++ < iters)
    {
        ddown/=2.0;
    }
    if(!testCount(count,iters)) handleCountError();

    // Use a bisection method to locate the root
    double dmid=(dup+ddown)/2.0;
    double integrationError=D*pow(dmid,−(p−v−kp)/kp)*homogeneousPDF(dmid);
    count = 0;
    while(fabs(integrationError−integrationTolerance)
          >integrationTolerance/1e2 & count++ < iters)
    {
        if((integrationError−integrationTolerance)>0.0){
            ddown=dmid;
        }else{
            dup=dmid;
        }
        dmid=(dup+ddown)/2.0;
        integrationError=D*pow(dmid,−(p−v−kp)/kp)*homogeneousPDF(dmid);
    }
    if(!testCount(count,iters)) handleCountError();

    // Return the root
    return dmid;
}

double pdf::heterogeneousCDF(const double x){
  /* Assign the values of the threshold */
  X=x;

  /* Create a workspace variable for the integration */
  gsl_integration_workspace *w = gsl_integration_workspace_alloc(10000);
  double result1, error;
  gsl_function F;
  F.function = &wrapperToHeterogeneousKernel;
  F.params = this;

  /* Integrate the function */
```

```
  double upperLimit = findUpperIntegrationLimit();
  gsl_integration_qags(&F,
                       0.0,                                          100
                       upperLimit,
                       0.0,
                       1.0e-6,
                       10000,
                       w,
                       &result1,
                       &error);

  gsl_integration_workspace_free(w);
  return result1;                                                    110
}

double pdf::homogeneousCDF(const double X0, const double X1)
{
  /* Create a workspace variable for the integration */
  gsl_integration_workspace *w = gsl_integration_workspace_alloc(10000);
  double result1, error;
  gsl_function F;
  F.function = &wrapperToHomogeneousPDF;
  F.params = this;                                                   120

  /* Integrate the function */
  gsl_integration_qags(&F,
                       X0,
                       X1,
                       1.0,
                       1.0e-2,
                       10000,
                       w,
                       &result1,                                     130
                       &error);

  gsl_integration_workspace_free(w);
  return result1;
}

double pdf::homogeneousCDF(const double x){
  /* Assign the values of the threshold */
  X=x;
                                                                     140
  /* Create a workspace variable for the integration */
  gsl_integration_workspace *w = gsl_integration_workspace_alloc(10000);
  double result1, error;
  gsl_function F;
  F.function = &wrapperToHomogeneousPDF;
  F.params = this;

  /* Integrate the function */
```

```
  gsl_integration_qags(&F,
                       0.0,
                       X,
                       1.0,
                       1.0e-2,
                       10000,
                       w,
                       &result1,
                       &error);

  gsl_integration_workspace_free(w);
  return result1;
}

double pdf::getThreshold(const double pf){
    double delta;
    double f,error;
    int MAXITER = 16384;
    integrationTolerance = pf/10.0;
    sf.setRelError(integrationTolerance);
    // Find value to the left of the real threshold
    f=heterogeneousCDF(x1);
    gint32 count = 0;
    while((1.0-f)<pf & count++ < MAXITER)
    {
        x1/=2.0;
        f=heterogeneousCDF(x1);
    }
    if(!testCount(count,MAXITER)) handleCountError();

    // Find value to the right of the real threshold
    f=heterogeneousCDF(x2);
    count = 0;
    while((1.0-f)>pf & count++ < MAXITER){
        x2*=2.0;
        f=heterogeneousCDF(x2);
    }
    if(!testCount(count,MAXITER)) handleCountError();

    // Do a binary search to find the correct threshold
    double xmid=(x1+x2)/2.0;
    f=heterogeneousCDF(xmid);
    count = 0;
    while(fabs(1.0-f-pf)>pf/1e2 & count++ < MAXITER){
        if((1.0-f-pf)>0.0){
            x1=xmid;
        }else{
            x2=xmid;
        }
        xmid=(x1+x2)/2.0;
        f=heterogeneousCDF(xmid);
```

```
    }                                                                    200
    if(!testCount(count,MAXITER)) handleCountError();
    return xmid;
}
```

The following lists the code for the ATI PDF object

```
#include "pdf.hpp"

#ifndef _ATIPDF_INCLUDED_
#define _ATIPDF_INCLUDED_
class atipdf : public pdf{
  private:
    double p;
  public:
    atipdf(const double e1a=20.0,
           const double e2a=2.0,                                    10
           const double na=4.9,
           const double va=6.9);
    virtual double homogeneousPDF(const double x);
    virtual double getThreshold(const double pf);
  protected:
    double root(const double D,
                const double n,
                const double eps);
    double qm(const double q0,
              const double n,                                       20
              const double ex0,
              const int m);
    double steepFunction(const double D,
                         const double n,
                         const double x);
};
#endif
```

The following lists the code for the ATI PDF implementation

```
#include "atipdf.hpp"
#include <gsl/gsl_sf.h>

atipdf::atipdf(const double e1a,
        const double e2a,
        const double na,
        const double va) :
  pdf(e1a,e2a,na,va){
  x1=1.0;
  x2=1.0;
  p = (e1a-e2a)/(e1a+e2a);
}

double atipdf::homogeneousPDF(const double x)
{
  double g[7];
  double D = p*cos(x);
  double ex0=root(D,n,1.0e-8);
  double q0=D*ex0+2.0*n;

  g[0] = log(ex0)*(n+1.0)+D*ex0+sf.lnBesselK(n-1.0,ex0);
  g[1] = 0.0;
  g[2] = D*ex0-qm(q0,n,ex0,1);
  g[3] = D*ex0-qm(q0,n,ex0,2);
  g[4] = D*ex0-qm(q0,n,ex0,3);
  g[5] = D*ex0-qm(q0,n,ex0,4);
  g[6] = D*ex0-qm(q0,n,ex0,5);

  double term1=
        sqrt(2.0*M_PI/(-g[2]));

  double term2=
        3.0/2.0*sqrt(2.0*M_PI/(-g[2]))*(
        5.0/36.0*pow(g[3],2)/pow(-g[2],3)
        +1.0/12.0*g[4]/pow(-g[2],2));

  double term3=
        15.0/4.0*sqrt(2.0*M_PI/(-g[2]))*(
        77.0/864.0*pow(g[3],4)/pow(g[2],6)
        -7.0/48.0*pow(g[3],2)*g[4]/pow(g[2],5)
        +7.0/180.0*g[5]*g[3]/pow(g[2],4)
        +7.0/288.0*pow(g[4],2)/pow(g[2],4)
        -1.0/180.0*g[6]/pow(g[2],3));

  // Make sure that the terms converge
  term3 = (fabs(term3)>fabs(term2))? 0.0 : term3;
  term3 = (fabs(term3)>fabs(term1))? 0.0 : term3;
  term2 = (fabs(term2)>fabs(term1))? 0.0 : term2;
```

```
  double pdf = exp(g[0])*pow(0.5-p*p/2.0,n)/M_PI/sf.Gamma(n)
       *(term1 + term2 + term3);
  return pdf;
}

double atipdf::qm(const double q0,
                  const double n,
                  const double ex0,
                  const int m)
{
  if(m==0) return q0;
  double q = 0.0;
  for(int k=0; k<m; k++)
  {
    q+=sf.poch((double)(m-k),k)/sf.poch(1.0,k)
       *qm(q0,n,ex0,m-k-1)
       *qm(q0,n,ex0,k);
  }
  return q-(2.0*n-2.0)*qm(q0,n,ex0,m-1)-pow(2.0,m-1)*ex0*ex0;
}

double atipdf::root(const double D,
                  const double n,
                  const double eps)
{
  // Find the root for the steepest descent method
  double xx1 = 1.0;
  int count = 0;
  int MAXITER = 16384;
  while(steepFunction(D,n,xx1)<0.0 & count++<MAXITER) xx1/=2.0;
  if(!testCount(count,MAXITER)) handleCountError();

  double xx2 = 1.0;
  count = 0;
  while(steepFunction(D,n,xx2)>0.0 & count++<MAXITER) xx2*=2.0;
  if(!testCount(count,MAXITER)) handleCountError();

  // Use the bisection method to find the root
  double xmid=(xx1+xx2)/2.0;
  double f=steepFunction(D,n,xmid);
  count = 0;
  while(fabs(xx2-xx1)>eps & count++ < MAXITER){
    if(f>0.0){
      xx1=xmid;
    }else{
      xx2=xmid;
    }
    xmid=(xx1+xx2)/2.0;
    f=steepFunction(D,n,xmid);
  }
```

```
  if(!testCount(count,MAXITER)) handleCountError();

  return xmid;
}

double atipdf::steepFunction(const double D,
                             const double n,
                             const double x)
{
  return D*x+2.0*n-x*exp(sf.lnBesselK(n,x) - sf.lnBesselK(n-1.0,x));
}

double atipdf::getThreshold(const double pf){
  double delta;
  double f,error;
  int count;
  int MAXITER = 16384;

  // Find value to the left of the real threshold
  count = 0;
  f=2.0*homogeneousCDF(x1,M_PI);
  while(f<pf & count++ < MAXITER){
    x1/=2.0;
    f=2.0*homogeneousCDF(x1,M_PI);
  }
  if(!testCount(count,MAXITER)) handleCountError();


  // Find value to the right of the real threshold
  count = 0;
  f=2.0*homogeneousCDF(x2,M_PI);
  while(f>pf & count++ < MAXITER){
    x2=(M_PI+x2)/2.0;
    f=2.0*homogeneousCDF(x2,M_PI);
  }
  if(!testCount(count,MAXITER)) handleCountError();

  // Do a binary search to find the correct threshold
  double xmid=(x1+x2)/2.0;
  f=2.0*homogeneousCDF(xmid,M_PI);
  count = 0;
  while(fabs(f-pf)>pf/1e2 & count++ < MAXITER){
    if((f-pf)>0.0){
      x1=xmid;
    }else{
      x2=xmid;
    }
    xmid=(x1+x2)/2.0;
    f=2.0*homogeneousCDF(xmid,M_PI);
  }
  if(!testCount(count,MAXITER)) handleCountError();
```

```
  return xmid;
}
```

The following lists the code for the DPCA PDF object

```
#include "pdf.hpp"

#ifndef _DPCAPDF_INCLUDED_
#define _DPCAPDF_INCLUDED_
class dpcapdf : public pdf{
private:
protected:
public:
  dpcapdf(const double e1a=20.0,
          const double e2a=2.0,
          const double na=4.9,
          const double va=6.9,
          const double ka=1.0);
  virtual double homogeneousPDF(const double x);
};
#endif
```

The following lists the code for the DPCA PDF implementation

```
#include "dpcapdf.hpp"
#include <gsl/gsl_sf.h>

dpcapdf::dpcapdf(const double e1a,
                 const double e2a,
                 const double na,
                 const double va,
                 const double ka) :
  pdf(e1a,e2a,na,va,ka){
  x1=x2=e2a*(na-1.0);                                                    10
}

double dpcapdf::homogeneousPDF(const double x){
  /* Calculate the pdf value */
  double s2=2.0*e2;
  return pow(x/s2,n-1.0)*exp(-x/s2)/s2/sf.Gamma(n);
}
```

The following lists the code for the Hyperbolic PDF object

```
#include "pdf.hpp"

#ifndef _HYPPDF_INCLUDED_
#define _HYPPDF_INCLUDED_
class hyperpdf : public pdf{
private:
protected:
public:
  hyperpdf(const double e1a=20.0,
           const double e2a=2.0,                                    10
           const double na=4.9,
           const double va=6.9,
           const double ka=1.0);
  virtual double homogeneousPDF(const double pf);
};
#endif
```

The following lists the code for the Hyperbolic PDF implementation. The implementation uses the GSL library for computing the quadrature integration and for some values of the hypergeometric function.

```
#include <stdio.h>
#include "hyperpdf.hpp"
#include <gsl/gsl_sf.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
#include <iomanip>

hyperpdf::hyperpdf(const double e1a,
                   const double e2a,
                   const double na,
                   const double va,
                   const double ka) :
  pdf(e1a,e2a,na,va,ka)
{
  x1=e2a;
  x2=e2a;
}

double hyperpdf::homogeneousPDF(const double x){
  if(x==0.0) return 0.0;

  // Compute the difference between the kummer functions
  double diffKummer = sf.diffGammaInc(2.0*n-1.0, x*(e1+e2)/e1,x*(e1+e2)/e2);

  // Compute the denominator gamma function term
  double GM = sf.Gamma(n-1.0);

  // Compute the Kummer(3,n+2,x) function
  double kummer1=sf.KummerU(3.0,n+2.0,x);

  // Return
  return 2.0/((e1-e2)*(n-1.0)*GM*GM*pow(e1+e2,2.0*n-1.0))*pow(e1*e2,n)
      *kummer1
      *diffKummer;
}
```

The following lists the code for the special function object

```
//
// C++ Interface: specialfunction
//
// Description:
//
//
// Author: Ishuwa C. Sikaneta and Marielle Quinton
// ishuwa.sikaneta
//
// Copyright: See COPYING file that comes with this distribution
//
//
#ifndef SPECIALFUNCTION_H
#define SPECIALFUNCTION_H

#include <iostream>
#include <stdio.h>
#include <gsl/gsl_sf.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
#include <gsl/gsl_integration.h>
#include <math.h>

    using namespace std;
/**
    This class provides numerical routines to compute various special functions
    that are needed for the CFAR computation for SAR-GMTI. GSL is used for
    quadrature integration and for differential equation integration.

    author Ishuwa C. Sikaneta and Marielle Quinton
    <ishuwa.sikanetadrdc-rddc.gc.ca>
 */
    class specialFunction{
      public:
        specialFunction(const double relerrtol = 1e-6);
        ~specialFunction();
        double KummerU(double a, double c, double x);
        double lnKummerU(double a, double c, double x);
        double spGaussHyp(const double mu, const double nu, const double D);
        double BesselK(double a, double x);
        double lnBesselK(double a, double x);
        double GammaInc(double a, double x);
        double lnGammaInc(double a, double x);
        double diffGammaInc(double a, double x1, double x2);
        double Gamma(double x);
        double poch(double a, int k);
        void setRelError(double relerrtol)
        {
```

```
    errortol = relerrtol;
  };
  double getRelError()
  {
    return errortol;
  }
private:
  double errortol;
  static int func (double t,
                   const double y[],
                   double f[],
                   void *params);
  static int jac (double t,
                  const double y[],
                  double *dfdy,
                  double dfdt[],
                  void *params);

  double asymptoticU(double a,
                     double c,
                     double z,
                     double error,
                     double *zasym,
                     double *uasym,
                     int *k);
  double getAsymptoticUError(double a,
                             double c,
                             double z,
                             int R);
  double evaluateAsymptoticU(double a,
                             double c,
                             double z,
                             double *u,
                             int *k,
                             double error);

  static double kernel(double t, void *alpha);

  double qm(const double q0,
            const double n,
            const double ex0,
            const int m);

  double root(const double D,
              const double mu,
              const double nu,
              const double eps);

  double steepFunction(const double D,
                       const double mu,
                       const double nu,
```

```
                        const double x);
    bool testCount(int count, int MAXITERS)
    {
      return (count < MAXITERS)? true : false;
    };
    void handleCountError()
    {
      cerr << "Maximum iterations reached. Exiting!" << endl;
      exit(1);
    }                                                                    110

};

#endif
```

The following lists the code for the special function object implementation

```
//
// C++ Implementation: specialfunction
//
// Description:
//
//
// Author: Ishuwa C. Sikaneta and Marielle Quinton
// <ishuwa.sikaneta
//
// Copyright: See COPYING file that comes with this distribution
//
//
#include "specialfunction.h"

specialFunction::specialFunction(double relerrtol)
  : errortol(relerrtol)
{
}


specialFunction::~specialFunction()
{
}

double specialFunction::KummerU(double a, double c, double x)
{
  return exp(lnKummerU(a,c,x));
}

double specialFunction::lnKummerU(double a, double b, double x)
{
  double x0, dx0, ab[2], h, y[2], uasym, duasym, ferror, derror;
  int kf, kdf;

// Check if we're approaching a singluarity at zero
  if(b>1.0 & x<1.0) return log(pow(x,1.0-b)) + lnKummerU(1.0+a-b,2.0-b,x);

// See if we can use the asymptotic expansion
  if(fabs(evaluateAsymptoticU(a, b, x, &uasym, &kf, errortol))<errortol)
    return log(uasym);

/* Otherwise integrate the differential equation
  to the correct small value of x. */

/* Choose a suitable initial value for the differential equation
  integrator */
  ferror = asymptoticU(a, b, x, errortol, &x0, &uasym, &kf);
  derror = asymptoticU(a+1.0, b+1.0, x, errortol, &dx0, &duasym, &kdf);
```

```
  /* Select the larger of the function and derivative asymptotic values */
  x0=(dx0>x0)? dx0 : x0;

  /* Evaluate the function and derivative at the asymptotic value */
  ferror = evaluateAsymptoticU(a, b, x0, &uasym, &kf, errortol);
  derror = evaluateAsymptoticU(a+1.0, b+1.0, x0, &duasym, &kdf, errortol);

/* Set up the GSL library to integrate using the Embedded Runge Kutta Fehlberg
   stepping routine */
  const gsl_odeiv_step_type *T
      = gsl_odeiv_step_rkf45;

  gsl_odeiv_step * s
      = gsl_odeiv_step_alloc (T, 2);
  gsl_odeiv_control * c
      = gsl_odeiv_control_y_new (0.0, errortol);
  gsl_odeiv_evolve * e
      = gsl_odeiv_evolve_alloc (2);

  ab[0] = a;
  ab[1] = b;
  gsl_odeiv_system sys = {func, jac, 2, &ab};

// Integrate towards the origin
  h = -1e-6;

// Compute the initial values
  y[0]=log(uasym);
  y[1]=-a*duasym/uasym;

  int counter = 0;
  int MAXITER=32768;

  while (x0 > x)
  {
    int status = gsl_odeiv_evolve_apply (e, c, s,
                                         &sys,
                                         &x0, x,
                                         &h, y);

    if (status != GSL_SUCCESS)
      break;
    if(counter++>MAXITER)
    {
      cout << "Could not compute KummerU function" << endl;
      exit(1);
    }
  }


  gsl_odeiv_evolve_free (e);
```

```
  gsl_odeiv_control_free (c);
  gsl_odeiv_step_free (s);
  return y[0];
}

double specialFunction::BesselK(double a, double x)
{
  return exp(lnBesselK(a, x));
}
                                                                    110
double specialFunction::lnBesselK(double a, double x)
{
  return gsl_sf_bessel_lnKnu(a, x);
// Another possibility uses the KummerU function to evaluate BesselK
// This is slower than GSL's function
// return lnKummerU(a+0.5,2.0*a+1.0,2.0*x)
//      + a*log(2.0*x)
//      - x
//      + 0.5723649429247001;
}                                                                   120

double specialFunction::GammaInc(double a, double x)
{
  return gsl_sf_gamma_inc(a, x);
// Can also use the KummerU function here. Speed is similar to GSL
// return exp(lnGammaInc(a, x));
}

double specialFunction::lnGammaInc(double a, double x)
{                                                                   130
  return lnKummerU(1.0,1.0+a,x)
       +a*log(x)-x;
}

double specialFunction::Gamma(const double x){
  if(x>0.0) return gsl_sf_gamma(x);
  if((x<0.0) & (x!=floor(x))) return Gamma(x+1.0)/x;
  return 0.0;
}
                                                                    140
double specialFunction::diffGammaInc(double a, double x1, double x2)
{
  double integral,error;

// Compute the difference in the incomplete gamma functions
  gsl_integration_workspace *w = gsl_integration_workspace_alloc(1000);
  gsl_function F;
  F.function = &kernel;
  F.params = &a;
  gsl_integration_qags(&F,                                          150
                       x1,
```

```
                        x2,
                        0.0,
                        errortol,
                        1000,
                        w,
                        &integral,
                        &error);
  gsl_integration_workspace_free(w);
                                                                        160
  return integral;
}

double specialFunction::kernel(double t, void *alpha)
{
  return exp(-t)*pow(t,*((double *)alpha)-1.0);
}

int specialFunction::func(double t,
                          const double y[],                             170
                          double f[],
                          void *params)
{
  double *ab = (double *)params;
  double a = ab[0];
  double b = ab[1];
  f[0] = y[1];
  f[1] = a/t + (1.0-b/t)*y[1]-y[1]*y[1];
  return GSL_SUCCESS;
}                                                                       180

int specialFunction::jac(double t,
                         const double y[],
                         double *dfdy,
                         double dfdt[],
                         void *params)
{
  double *ab = (double *)params;
  double a = ab[0];
  double b = ab[1];                                                     190
  gsl_matrix_view dfdy_mat
      = gsl_matrix_view_array (dfdy, 2, 2);
  gsl_matrix * m = &dfdy_mat.matrix;
  gsl_matrix_set (m, 0, 0, 0.0);
  gsl_matrix_set (m, 0, 1, 1.0);
  gsl_matrix_set (m, 1, 0, 0.0);
  gsl_matrix_set (m, 1, 1, (1.0-b/t)-2.0*y[1]);
  dfdt[0] = 0.0;
  dfdt[1] = -a/t/t+b*y[1]/t/t;
  return GSL_SUCCESS;                                                   200
}
```

```
double specialFunction::asymptoticU(double a,
                                    double c,
                                    double z,
                                    double error,
                                    double *zasym,
                                    double *uasym,
                                    int *k)
{                                                                    210
/* The error in the asymptotic expansion depends on z
   so perform a binary search on the best value of z. First
   find an upper bound that satisfies the error criterion */
  double zlow;
  double zhigh;
  double zError;
  int R;

  *zasym=z;
                                                                     220
  zhigh=(*zasym);
  double aerror = evaluateAsymptoticU(a,c,zhigh,uasym,k,error);
  while(aerror>error)
  {
    zhigh*=2.0;
    aerror = evaluateAsymptoticU(a,c,zhigh,uasym,k,error);
  }
  *zasym=zhigh;

  zlow = fabs(a*(1.0+a-c));                                          230

  /* Do the binary search for the best z */
  zError = 1e-2;

  while(fabs((*zasym)-(zlow+zhigh)/2.0)>zError)
  {
    *zasym=(zlow+zhigh)/2.0;
    aerror = evaluateAsymptoticU(a,c,*zasym,uasym,k,error);
    if(aerror>error)
      zlow=(*zasym);                                                 240
    else
      zhigh=(*zasym);
  }
  return aerror;
}

double specialFunction::getAsymptoticUError(double a,
                                            double c,
                                            double z,
                                            int R)                   250
{
  return
```

```
      fabs(poch(a,R)*poch(1.0+a-c,R)/poch(1.0,R)
/pow(-z,R)*(0.5+(1.0/8.0+c/4.0-a/2.0+z
/4.0-(R)/4.0)/z+5.0/z/z));
}

double specialFunction::evaluateAsymptoticU(double a,
                                            double c,
                                            double z,
                                            double *u,
                                            int *k,
                                            double errtol)
{
  /* Evaluates the asymptotic expansion and returns the error */
  double term;
  *u=1.0;
  (*k)=1;
  term=1.0;
  while((fabs((a+(*k)-1.0)*(a-c+(*k))/(*k)/(-z))<=1.0)
        & (fabs(term)>errtol))
  {
    term*=(a+(*k)-1.0)*(a-c+(*k))/(*k)/(-z);
    (*u)+=term;
    (*k)++;
  }
  (*u)/=pow(z,a);
  (*k)--;
  return fabs(term); //getAsymptoticError(a,c,z,*k);
}

double specialFunction::poch(double a, int k)
{
  /* Function to compute the pochhammer symbol */
  int i;
  double y;
  y=1.0;

  for(i=0; i<k; i++)
    y*=(a+i);

  return y;
}

double specialFunction::spGaussHyp(const double mu, const double nu, const double D)
{
  double g[7];
  double ex0=root(D,mu,nu,1.0e-12);
  double q0=D*ex0+mu+nu-1.0;

  g[0] = log(ex0)*mu+D*ex0+lnBesselK(nu-1.0,ex0);
  g[1] = 0.0;
  g[2] = D*ex0-qm(q0,nu,ex0,1);
```

```
  g[3] = D*ex0-qm(q0,nu,ex0,2);
  g[4] = D*ex0-qm(q0,nu,ex0,3);
  g[5] = D*ex0-qm(q0,nu,ex0,4);
  g[6] = D*ex0-qm(q0,nu,ex0,5);

  double term1=                                                    310
        sqrt(2.0*M_PI/(-g[2]));

  double term2=
        3.0/2.0*sqrt(2.0*M_PI/(-g[2]))*(
        5.0/36.0*pow(g[3],2)/pow(-g[2],3)
        +1.0/12.0*g[4]/pow(-g[2],2));

  double term3=
        15.0/4.0*sqrt(2.0*M_PI/(-g[2]))*(
        77.0/864.0*pow(g[3],4)/pow(g[2],6)                         320
        -7.0/48.0*pow(g[3],2)*g[4]/pow(g[2],5)
        +7.0/180.0*g[5]*g[3]/pow(g[2],4)
        +7.0/288.0*pow(g[4],2)/pow(g[2],4)
        -1.0/180.0*g[6]/pow(g[2],3));

  // Make sure that the terms converge
  cout << -g[2] << " | "
       << g[3] << " | "
       << g[4] << " | "
       << g[5] << " | "                                            330
       << g[6] << " -> "
        << term1 << " | "
        << term2 << " | "
        << term3 << endl;
  term3 = (fabs(term3)>fabs(term2))? 0.0 : term3;
  term3 = (fabs(term3)>fabs(term1))? 0.0 : term3;
  term2 = (fabs(term2)>fabs(term1))? 0.0 : term2;


  double fn = exp(g[0])*(term1 + term2 + term3);                   340
  fn *= Gamma(mu+0.5)*pow(1.0-D,mu+nu-1.0)
        /sqrt(M_PI)/pow(2.0,nu-1.0)
        /Gamma(mu+nu-1.0)/Gamma(mu-nu+1.0);
  return fn;
}

double specialFunction::qm(const double q0,
                           const double nu,
                           const double ex0,
                           const int m)                            350
{
  if(m==0) return q0;
  double q = 0.0;
  for(int k=0; k<m; k++)
  {
```

```
    q+=poch((double)(m-k),k)/poch(1.0,k)
        *qm(q0,nu,ex0,m-k-1)
        *qm(q0,nu,ex0,k);
  }
  return q-(2.0*nu-2.0)*qm(q0,nu,ex0,m-1)-pow(2.0,m-1)*ex0*ex0;
}

double specialFunction::root(const double D,
                             const double mu,
                             const double nu,
                             const double eps)
{
// Find the root for the steepest descent method
  double xx1 = 1.0;
  int count = 0;
  int MAXITER = 32768;
  while(steepFunction(D,mu,nu,xx1)<0.0 & count++<MAXITER) xx1/=2.0;
  if(!testCount(count,MAXITER)) handleCountError();

  double xx2 = 1.0;
  count = 0;
  while(steepFunction(D,mu,nu,xx2)>0.0 & count++<MAXITER) xx2*=2.0;
  if(!testCount(count,MAXITER)) handleCountError();

// Use the bisection method to find the root
  double xmid=(xx1+xx2)/2.0;
  double f=steepFunction(D,mu,nu,xmid);
  count = 0;
  while(fabs((xx2-xx1)/xmid)>eps & count++ < MAXITER){
    if(f>0.0){
      xx1=xmid;
    }else{
      xx2=xmid;
    }
    xmid=(xx1+xx2)/2.0;
    f=steepFunction(D,mu,nu,xmid);
  }
  if(!testCount(count,MAXITER)) handleCountError();

  return xmid;
}

double specialFunction::steepFunction(const double D,
                                      const double mu,
                                      const double nu,
                                      const double x)
{
  return D*x+mu+nu-1.0-x*exp(lnBesselK(nu,x) - lnBesselK(nu-1.0,x));
}
```

This page intentionally left blank.

Distribution list

DRDC Ottawa TR 2006-294

Internal distribution

1 Ishuwa Sikaneta

1 Shen Chiu

1 Christoph Gierull

1 Chuck Livingstone

1 Pierre Beaulne

1 Martina Gabele

1 Bhashyam Balaji

1 Head/Radar Systems

4 Library

Total internal copies: 12

External distribution

1 ADM(S&T)

1 DRDKIM 3

1 Library & Archives Canada

1 CISTI

Total external copies: 4

Total copies: 16

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

   Defence R&D Canada – Ottawa
   3701 Carling Avenue, Ottawa, Ontario, Canada
   K1A 0Z4

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).

   UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).

   Numerically Computing the KummerU Function and a Special Case of the Gauss Hypergeometric Function

4. AUTHORS (last name, first name, middle initial)

   Ishuwa C. Sikaneta,

5. DATE OF PUBLICATION (month and year of publication of document)	6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc).	6b. NO. OF REFS (total cited in document)
December 2006	88	18

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).

   Technical Report

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).

   Defence R&D Canada – Ottawa
   3701 Carling Avenue, Ottawa, Ontario, Canada K1A 0Z4

9a. PROJECT NO. (the applicable research and development project number under which the document was written. Specify whether project).	9b. GRANT OR CONTRACT NO. (if appropriate, the applicable number under which the document was written).
15eg11	

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique.)	10b. OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Ottawa TR 2006-294	

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

    (X) Unlimited distribution
    ( ) Defence departments and defence contractors; further distribution only as approved
    ( ) Defence departments and Canadian defence contractors; further distribution only as approved
    ( ) Government departments and agencies; further distribution only as approved
    ( ) Defence departments; further distribution only as approved
    ( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution beyond the audience specified in (11) is possible, a wider announcement audience may be selected).

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

This report proposes new methods to compute the Kummer hypergeometric function of the second kind and to approximate a special case of the Gauss hypergeometric function. The Kummer hypergeometric function, $\Psi(a, c; z)$, is computed for $a, c \in \mathbb{R}$ and $x \in \mathbb{R}^+$ although the proposed method can be extended to $a, c, z \in \mathbb{C}$. The special case of the Gauss hypergeometric function is given by ${}_2\mathrm{F}_1(\mu+\nu-1, \nu-1/2; \mu+1/2; x)$ where $\mu > |\nu-1|$, and where $x \in \mathbb{R} < 1$. Numerical results of the proposed algorithms are compared with the capabilities of GSL™, Mathematica™and Maple™.

The evaluation of these functions is required for the dual aperture Synthetic Aperture Radar (SAR), Ground Moving Target Indication (GMTI), Constant False Alarm Rate (CFAR) problem. It is shown that the proposed methods provide a practical means for computing the CFAR thresholds over a wide range of the SAR GMTI statistical parameters and false alarm thresholds, even in very heterogeneous terrain. Additionally, this report shows how to handle infinite integrals that arise when the statistics of heterogeneous terrain are described by the product model.

A C++ implementation of the developed algorithms is provided.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

SAR; GMTI; CFAR; Gauss hypergeoemtric function; Kummer hypergeometric function; Steepest descents; Path integration; Series reversion; Asymptotic expansion; Heterogeneous; Homogeneous; ATI; DPCA; Hyperbolic detector

Defence R&D Canada

Canada's leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca