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

DEFENCE R&D DÉFENSE

Modelling RADARSAT-2 Doppler centroids for marine applications

Chuck Livingstone and Pierre Beaulne

Defence R&D Canada – Ottawa

Technical Memorandum
DRDC Ottawa TM 2012-097
August 2012

Canada

Modelling RADARSAT-2 Doppler centroids for marine applications

Chuck Livingstone and Pierre Beaulne
DRDC Ottawa

Defence R&D Canada – Ottawa

Technical Memorandum
DRDC Ottawa TM 2012-097
August 2012

Principal Author

Original signed by Chuck Livingstone

Chuck Livingstone

Defence Scientist

Approved by

Original signed by Anthony Damini

Anthony Damini

A/H. Radar Systems

Approved for release by

Original signed by Chris McMillan

Chris McMillan

Chief Scientist DRDC Ottawa

Abstract

The Doppler centroid of acquired SAR (Synthetic Aperture Radar) and SAR-GMTI (Ground Moving Target Indication) signal data is required to refine theoretically-defined processing filters for SAR processing of stationary terrain and to minimize GMTI artefacts for moving targets. When precision orbit and attitude data are used in conjunction with an ellipsoidal earth model, SAR processing filters can be adjusted for the observation geometry to generate SAR imagery that is acceptable for most applications. Residual earth rotation artefacts in this imagery have no significant impact on its use.

The residual earth rotation effects after matched filter compensation are significant for GMTI measurements. For land scenes, the background terrain can be assumed to be static and radar returns from scene elements that do not contain moving targets can be used to provide the required reference. On the sea surface, the background terrain is moving and radar returns are often too weak to capture reliable Doppler centroid information from the acquired signal data. Although RADARSAT-2 has been designed to minimize earth rotation Doppler shifts by controlling the satellite attitude over the orbit to point the satellite beam near the zero Doppler azimuth angle, the attitude control law that is implemented in the satellite uses satellite position information and does not account for orbit and earth ellipticity or radar range. In addition there are perturbations in the reported attitude data due to azimuth pointing control lags and small misalignments between the attitude sensor and antenna coordinate systems. The composite effect on the Doppler centroid of the radar returns is not significant for most imaging operations but is large enough to impact GMTI measurements in cases where reference signals from static terrain are not available.

This Technical Memorandum presents a model-based analysis of the Doppler frequency properties of RADARSAT-2 data and presents a RADARSAT-2 Doppler centroid estimation model that is sufficiently accurate for GMTI measurements of marine targets in the open ocean. Model verification using metadata from ten months of RADARSAT-2 data acquisitions is described.

A RADARSAT-2 Doppler centroid estimation tool suitable for use with individual ocean data sets is presented.

Résumé

Le centroïde Doppler des données de signal acquises de SAR (radar à synthèse d'ouverture) et de SAR-GMTI (indication de cible terrestre mobile) doit être calculé pour améliorer des filtres de traitement définis théoriquement en vue du traitement SAR des éléments stationnaires du relief ainsi que pour réduire le plus possible les artéfacts GMTI des cibles mobiles. Lorsque des données de précision sur l'orbite et l'attitude sont utilisées avec un modèle ellipsoïdal de la Terre, les filtres de traitement SAR peuvent être corrigés en fonction de la géométrie d'observation afin de fournir des images SAR acceptables pour la plupart des applications. Les artéfacts résiduels dus à la rotation de la Terre n'ont pas d'effet important sur leur utilisation.

Les effets résiduels de la rotation de la Terre après l'application des corrections au filtre adapté sont importants pour les mesures GMTI. Pour les images terrestres, on peut considérer que le relief d'arrière-plan est statique; les échos radar des éléments de l'image qui ne contiennent pas de cibles mobiles peuvent fournir la référence nécessaire. À la surface de la mer, les centres de diffusion radar d'arrière-plan (les sources de fouillis radar) se déplacent et les échos radar sont souvent trop faibles pour obtenir une information fiable sur le centroïde Doppler à partir des données de signal acquises. Même si RADARSAT-2 a été conçu pour réduire les décalages de Doppler produits par la rotation de la Terre en pilotant l'attitude du satellite sur son orbite de façon à orienter le faisceau près de l'angle de variation Doppler nulle, la loi de commande de l'attitude utilisée dans le satellite fait appel à l'information de position du satellite et ne tient pas compte de l'ellipticité de l'orbite et de la Terre ni de la distance radar. De plus, les données sur l'attitude obtenues comportent des perturbations causées par les retards de la commande de pointage en azimut et de petits défauts d'alignement entre le capteur d'assiette et les systèmes de coordonnées d'antenne. Pour la plupart des applications d'imagerie, l'effet combiné sur le centroïde Doppler des échos radar peut être négligé, mais il est assez important pour perturber les mesures GMTI si des signaux de référence provenant d'éléments statiques ne sont pas disponibles.

Le présent rapport technique donne une analyse fondée sur un modèle des propriétés en fréquences Doppler des données RADARSAT-2 ainsi qu'un modèle d'évaluation du centroïde Doppler de RADARSAT-2 suffisamment précis pour permettre les mesures GMTI de cibles maritimes au grand large. La vérification du modèle au moyen de métadonnées obtenues à partir de données acquises par RADARSAT-2 sur une période de dix mois est décrite.

Un outil d'évaluation du centroïde Doppler de RADARSAT-2 pouvant être utilisé avec des jeux de données océaniques individuels est présenté.

Executive summary

Modelling RADARSAT-2 Doppler centroids for marine applications:

Chuck Livingstone; Pierre Beaulne; DRDC Ottawa TM 2012-097; Defence R&D Canada – Ottawa; August 2012.

Introduction or background: The Doppler centroid of acquired SAR (Synthetic Aperture Radar) and SAR-GMTI (Ground Moving Target Indication) signal data is required to refine theoretically defined processing filters for SAR processing of stationary terrain and to minimize GMTI artefacts for moving targets. For land scenes, the background terrain can be assumed to be static and radar returns from scene elements that do not contain moving targets can be used to provide the required reference. On the sea surface, the background radar scattering centers (radar clutter sources) are moving and radar returns are often too weak to capture reliable Doppler centroid information from the acquired signal data.

RADARSAT-2 minimizes Doppler shifts due to earth rotation and imaging geometry by controlling the satellite attitude to point the radar beam center close to the zero Doppler point on the earth's surface. RADARSAT-2 design information states that the satellite yaw angle is steered by rotating the satellite body, about the satellite nadir axis, through the dynamic yaw angle $-3.92^{\circ}\cos(\alpha)$, where α is defined as the orbit central angle, measured from the ascending equator crossing. Satellite design information is silent on dynamic pitch control but communications with MDA (MacDonald, Detwiller and Associates Ltd.), the RADARSAT-2 owner-operator, revealed that the satellite pitch is controlled to place the radar antenna in the plane that is normal to the satellite position-vector. The satellite attitude is servo-controlled using the satellite momentum wheels and using star-tracker pointing-error estimation to provide control feed-back.

When the nominal attitude control laws for RADARSAT-2 are interpreted in terms of the expected scene Doppler centroids, the residual earth rotation effects correspond to a maximum residual earth surface speed of 6.5 m/s which presents no issue for standard SAR imaging when the SAR processing filters are compensated by a precision observation geometry model (the errors fall within the RADARSAT-2 design specifications.), but is significant for GMTI measurements. This Technical Memorandum develops a theoretical model that describes the systematic part of RADARSAT-2 Doppler centroid measurements. The model is validated by using RADARSAT-2 metadata parameters and precision orbit data for ten months of SAR data acquisitions to test the modeled residual Doppler centroids. It is shown that the Doppler centroid model is valid to within the combined measurement errors in the metadata. The principal error contributors are servo errors of the RADARSAT-2 attitude control system and Doppler centroid measurement errors in metadata generation.

The residual Doppler shift model presented here provides Doppler centroid corrections that are suitable for use in marine GMTI applications.

Results: A model for RADARSAT-2 attitude control and measurement geometry has been developed and has been demonstrated to reduce systematic measurement geometry and earth

rotation effects to levels that are well below the associated measurement noise. A model-based Doppler centroid estimation tool has been developed and is presented.

Significance: When the model is used for marine GMTI measurements, earth rotation terms can be reduced below measurement noise levels so that they do not compromise target and sea surface motion estimates from complex SAR data.

Future plans: The model will be used in routine SAR-GMTI maritime measurements and is available for use in the analysis of complex SAR data from RADARSAT-2.

Sommaire

Modelling RADARSAT-2 Doppler centroids for marine applications:

Chuck Livingstone; Pierre Beaulne; DRDC Ottawa TM 2012-097; R & D pour la defence Canada – Ottawa; août 2012.

Introduction ou contexte: Le centroïde Doppler des données de signal acquises de SAR (radar à synthèse d'ouverture) et de SAR-GMTI (indication de cible terrestre mobile) doit être calculé pour améliorer des filtres de traitement définis théoriquement en vue du traitement SAR des éléments stationnaires du relief ainsi que pour réduire le plus possible les artéfacts GMTI des cibles mobiles. Pour les images terrestres, on peut considérer que le relief d'arrière-plan est statique; les échos radar des éléments de l'image qui ne contiennent pas de cibles mobiles peuvent fournir la référence nécessaire. À la surface de la mer, les centres de diffusion radar d'arrière-plan (les sources de fouillis radar) se déplacent et les échos radar sont souvent trop faibles pour obtenir une information fiable sur le centroïde Doppler à partir des données de signal acquises.

RADARSAT-2 minimise les décalages de Doppler produits par la rotation de la Terre et la géométrie de prise de vue en réglant l'attitude du satellite afin d'orienter le centre du faisceau près du point de variation Doppler nulle à la surface de la Terre. L'information de conception de RADARSAT-2 indique que l'angle de lacet du satellite est modifié en faisant tourner le corps du satellite autour de l'axe de nadir du satellite par l'angle de lacet dynamique $-3{,}92^{\circ}\cos(\alpha)$, avec α l'angle central de l'orbite mesuré à partir du passage au nœud ascendant. L'information de conception du satellite ne mentionne pas le pilotage dynamique en tangage, mais l'exploitant propriétaire de RADARSAT-2, MDA (MacDonald, Detwiller and Associates Ltd.), a divulgué dans une communication que le tangage du satellite est piloté de façon a placer l'antenne radar dans le plan normal au vecteur position du satellite. L'attitude du satellite est asservie au moyen de volants d'inertie et d'une réaction correspondant à l'erreur de pointage fournie par le suiveur stellaire.

Si on utilise les lois de commande de l'assiette nominales de RADARSAT-2 pour calculer les centroïdes Doppler attendus dans une image, l'effet résiduel de la rotation de la Terre correspond à une vitesse résiduelle maximale de la surface de la Terre de 6,5 m/s, ce qui ne présente aucun problème pour l'imagerie SAR ordinaire lorsque les filtres de traitement SAR sont corrigés au moyen d'un modèle de géométrie d'observation de précision (les erreurs se trouvent dans les bornes des spécifications de conception de RADARSAT-2). Ces erreurs sont cependant assez importantes pour perturber les mesures GMTI. Le présent rapport technique donne un modèle théorique qui décrit la portion systématique des mesures de centroïde Doppler de RADARSAT-2. Le modèle est validé au moyen de paramètres de métadonnées de RADARSAT-2 et de données d'orbite de précision acquises sur une période de dix mois pour contrôler les centroïdes Doppler résiduels fournis par le modèle. Le modèle de centroïde Doppler s'est avéré valide dans les bornes de l'erreur de mesure combinée des métadonnées. Les principaux facteurs contribuant à l'erreur sont les erreurs d'asservissement du système de commande de l'attitude de RADARSAT-2 et les erreurs de mesure du centroïde Doppler produites lors de la génération des métadonnées.

Le modèle du décalage de Doppler résiduel présenté ici permet d'obtenir des corrections du centroïde Doppler adaptées aux applications de GMTI maritime.

Résultats: Un modèle de la commande d'attitude et de la géométrie de mesure de RADARSAT-2 a été mis au point et on a démontré qu'il réduit les effets systématiques de la géométrie de mesure et de la rotation de la Terre à des niveaux bien en deçà du bruit de mesure connexe. Un outil d'évaluation du centroïde Doppler mis au point à l'aide du modèle est présenté.

Importance: Lorsque le modèle est utilisé pour prendre des mesures GMTI maritimes, les effets de la rotation de la Terre peuvent être réduits à des niveaux en deçà du bruit de mesure, de sorte qu'ils ne nuisent pas à l'évaluation du mouvement des cibles et de la surface de la mer à partir de données SAR complexes.

Perspectives: Le modèle servira à des mesures SAR-GMTI maritimes courantes; il peut également servir à l'analyse de données SAR complexes produites par RADARSAT-2.

Table of contents

List of figures

List of tables

Acknowledgements

The authors wish to thank MacDonald, Dettwiler and Associates Ltd. (MDA) scientist, Alan Thompson, for providing a set of RADARSAT-2 metadata to use for model tuning and model validation. The assistance provided by Alan and by Dan Williams (MDA) in clarifying details of the MDA data set, the underlying RADARSAT-2 orbit parameters and the attitude control assumptions was of great value. A review of the draft version of this report by MDA personnel was critical to the development of this final document.

This page intentionally left blank.

1 Introduction

The Doppler centroid of the acquired SAR (Synthetic Aperture Radar) and SAR-GMTI (Ground Moving Target Indication) signal data is required to refine theoretically-defined processing filters for SAR imaging of stationary (not moving) terrain and to minimize GMTI artefacts for moving targets. For land scenes, the background terrain can be assumed to be static over a radar observation interval and radar returns from scene elements that do not contain moving targets can be used to provide the required reference. On the sea surface, the background radar scattering centers (radar clutter sources) are moving and radar returns can be weak, so Doppler centroid information derived from the acquired signal data does not often provide a suitable stationary reference. Although RADARSAT-2 has been designed to minimize earth rotation Doppler shifts by controlling the satellite attitude over the orbit to point the satellite beam near the zero Doppler azimuth angle, the attitude control law that is implemented in the satellite uses satellite position information and does not account for orbit and earth ellipticity or radar range. In addition there are perturbations to the reported attitude data due to azimuth pointing control lags and there are small misalignments between the attitude sensor and effective antenna beam coordinate systems. The composite effect on the Doppler centroid of the radar returns is not significant for most imaging applications but is large enough to impact GMTI measurements in cases where true stationary reference signals are not available.

RADARSAT-2 design information states that the satellite yaw angle is steered by rotating the satellite body, about the satellite nadir axis, through the dynamic yaw angle -3.92ºcos(α) [1], where α is defined as the orbit central angle, measured from the ascending (north-bound) satellite equator crossing. Information obtained from the satellite owner-operator, MDA (MacDonald, Dettwiler and Associates Ltd.) indicates that the satellite pitch angle is defined with respect to the plane normal to the satellite position vector. The on-orbit pitch control function tries to constrain the long axis of the radar antenna to this plane.

RADARSAT-2 attitude control is accomplished by continuously adjusting the satellite angular momentum vectors about the earth-centered satellite position vector (yaw control) and about the satellite position vector normal (pitch control) using feed-back from an attitude measurement system. This implies that any radar measurement made by the satellite will contain low-level servo-error terms that have random (measurement error noise) and systematic (control lag) errors as well as constant (or seasonally variable) biases due to coordinate system misalignments between the star-tracker attitude sensors and the radar antenna. The systematic effects need to be incorporated into attitude control models and the noise terms will determine the ultimate limit of model accuracy.

An examination of RADARSAT-2 GMTI land data from 53 GMTI Doppler centroid measurement sites between latitudes N 35º and N 53º [3] revealed consistent Doppler centroid offsets that varied in sign with the orbit phase and had mean absolute values on the vicinity of 220 Hz. This corresponds to an earth surface velocity component of ~6 m/s that is projected along the radar range vector. The observed, residual earth rotation artefact is not significant for many standard SAR applications but is important for GMTI measurements. At issue for the global use of RADARSAT-2 GMTI modes is the ability to employ the modes for ocean surface measurements where Doppler centroid compensation from radar return measurements is not available or is not reliable.

The limited set of GMTI observations was expanded by global Doppler centroid data for the period September 2008 to November 2009 provided by Tony Luscombe (MDA) and displayed in Figure 1. Figure 1 shows that the measured Doppler centroids have multiple values at each geodetic latitude where data acquisitions have occurred. The distribution of Doppler centroid measurements can be described in terms of a cluster of values at each latitude and contains a random component superimposed on a systematic function that varies with latitude.

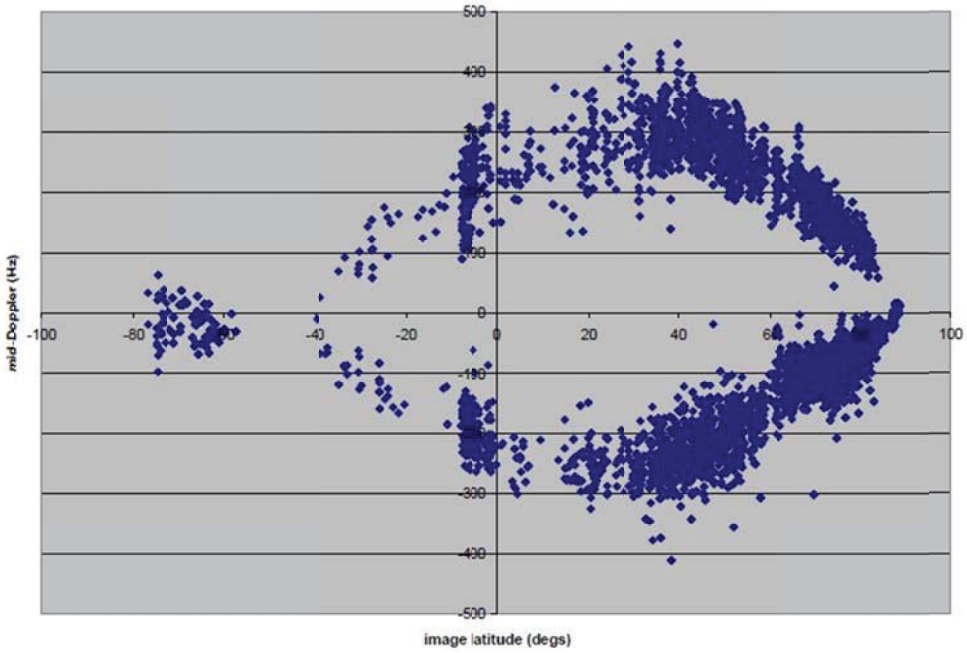

Figure 1: MDA whole earth residual Doppler shift observations Sept 2008 to November 2009

Image credit: Tony Luscombe, MDA

The observed Doppler centroids for the RADARSAT-2 GMTI scenes fall within the point spread shown in Figure 1.

This document presents an analysis of the effect of RADARSAT-2 orbital motion, attitude control and observation geometry to create a Doppler centroid model that accounts for relevant systematic effects and is valid for use with global maritime GMTI measurements. The theoretical model is validated by its fit to metadata obtained from ten months of RADARSAT-2 SAR acquisitions and the model fit is used to estimate the random part of the measured Doppler centroid distribution.

In the body of the report:

- Section 2 develops a RADARSAT-2 measurement geometry model for theoretical calculations. This section captures RADARSAT-2 orbit parameter data and earth

ellipsoid model data for use in subsequent calculations and then develops a satellite orbit model, a satellite attitude model and a range-angle geometry model to support theoretical estimates.

- Section 3 develops a theoretical scene Doppler centroid model and uses this model to predict residual earth rotation effects for RADARSAT-2 measurements based on defined assumptions about the RADARSAT-2 attitude control system.
- Section 4 tests the Doppler centroid model against historical RADARSAT-2 measurements that have been captured from RADARSAT-2 SAR processor signal processing operations. The model uses precision orbit state vectors retrieved from the RADARSAT-2 archive to provide radar motion information. Comparative results are presented and the magnitude of the combined RADARSAT-2 attitude control system noise and Doppler centroid measurement noise is estimated.
- Section 5 describes a Doppler centroid estimation tool that has been developed for use with individual RADARSAT-2 ocean data acquisitions. This section is written as a user manual for the tool.
- Section 6 summarizes conclusions drawn from the work reported in this document.
- Annex A contains the Matlab functions and scripts used to prepare the orbit file, and Doppler coefficient data for model analysis and to execute the model on a set of metadata. This Annex has been included to allow the analysis to be replicated for other metadata sets.
- Annex B contains the Matlab scripts and functions that form a Doppler centroid modelling tool for use with RADARSAT-2 ocean image data. The user manual for this tool is contained in Section 5.
- Annex C contains the library of user-defined Matlab functions needed to run the code in Annexes A and B.
- The Doppler centroid prediction tool can be run from the Matlab software contained in the attached CD.

2 RADARSAT-2 measurement geometry model

The orbit and observation geometry used by RADARSAT-2 form the basis of the mathematical model development. The model assumes that the satellite is measuring the surface of an ellipsoidal earth that is represented by the WGS 84 (World Geodetic System, 1984) ellipsoid. The earth symmetry allows model calculations to be based on the geodetic latitude and the orbit central angle and thus a simplified orbit plane model can be used.

2.1 RADARSAT-2 orbit parameters

The satellite orbit data for all data acquisitions can be extracted from archived precision state vectors. For simple, theoretical, calculations the reported orbit parameters, presented in Table 1 provide an adequate representation of the satellite orbit.

Table 1: RADARSAT-2 orbit parameters

Period	100.70045015 minutes
Orbits per day	14.299838
Mean altitude	797.7 km (CSA {Canadian Space Agency})
Altitude at perigee	792 km (CSA)
Altitude at apogee	799 km (CSA)
Inclination	98.58º
Orbit average speed	$V_S = \frac{631.34816}{\sqrt{R_S}}$ km/s = 7.45552589 km/s
Orbit eccentricity	0.001155 (Dan Williams, MDA (MacDonald Dettwiler and Associates))
Argument of perigee	89.72º(Dan Williams, MDA)
Semi-major axis	7167.07 km (Dan Williams, MDA)

2.2 Earth surface model

For the purposes of this analysis the earth surface can be modelled as an ellipsoid of revolution centered on the earth-rotation axis. The key parameters are presented in Table 2.

Table 2: WGS 84 earth geometry parameters

Equatorial radius	6378.1370 km
Polar radius	6356.7523 km
Mean radius (Volume match to sphere)	6371.0008 km
Earth ellipsoid flattening factor, f	1/298.257223563
Earth ellipticity, $e = \sqrt{2f - f^2}$	0.0818191908426
Sidereal day	23.9345 h
Earth angular velocity, Ω_E	7.292115x 10^{-5} radians per second

The analysis in this document uses the sidereal day for earth rotation rate estimates and represents the earth's surface by the WGS 84 ellipsoid,

$$R_E = \frac{R_{Eq}\left(1-e_E^2\right)^{\frac{1}{2}}}{\left(1-e_E^2 cos(\lambda_E)^2\right)^{\frac{1}{2}}}, \tag{1}$$

where λ_E is the geocentric latitude at the imaged point, R_{Eq} is the equatorial earth radius, e_E is the earth ellipticity and R_E is the local earth radius.

2.3 Satellite orbit model

The following conventions will be used in this technical memorandum: $\hat{A}$ is a unit vector, $\bar{A}$ is a vector. $\begin{bmatrix} A_X \\ A_Y \\ A_Z \end{bmatrix}$ is the matrix representation of a vector and A is a scalar quantity.

2.3.1 RADARSAT-2 elliptic orbit parameters

The RADARSAT-2 orbit is described as an ellipse with ellipticity $e_S = 0.001155$ and argument of perigee (the orbit-plane central angle of the orbit perigee), $\theta_P = 89.72^o$ using the convention that θ_p is measured from the ascending node. For model calculations we will characterize the satellite parameters in terms of an orbit internal angle, α, which is the angle between the satellite position vector at any point on the orbit and the satellite position vector at the north-bound equator crossing (the ascending node of the orbit). Since this angle increases in the satellite direction of satellite travel, we will assume that counter-clockwise angles are positive.

An elliptic orbit has the perigee focus of the orbit ellipse at the gravitational center of the earth. For RADARSAT-2, the orbit semi-major axis length is $R_{SM} = 7167.07$ km, the separation of the perigee and apogee ellipse foci is 14.2459 km, the orbit radius at perigee is 7159.95 km and the orbit radius at apogee is 7171.19 km.

For model calculations, three elliptic orbit parameters are of interest:

- the orbit radius,
- the true anomaly of the satellite,
- the angle between the orbit radius vector and the satellite velocity vector, and
- the satellite orbital speed.

If θ_P is the argument of perigee and α is the orbit internal angle, the true anomaly of the satellite, ν, is the earth-center angle between the satellite position at perigee and its current position, or:

$$\nu = \alpha - \theta_P. \tag{2}$$

The elliptic orbit radius is given by

$$R_S = \frac{R_{SM}\left(1-e_S^2\right)}{1+e_S cos(\nu)} \tag{3}$$

km where R_{SM} is the semi-major axis of the orbit and e_S is the orbit ellipticity.

The angle between the plane normal to the satellite position vector and the satellite velocity vector (the flight-path angle) is given by:

$$\phi = atan2(e_S sin(v), 1 + e_S \cos(v)). \quad (4)$$

The satellite speed at the true anomaly, v, is then:

$$V_S = \frac{631.34816\sqrt{R_{SM}(1-e_S^2)}}{R_S cos(\phi)} \quad (5)$$

km/s.

2.3.2 The inertial orbit plane model

For convenience, we will define the RADARSAT-2 orbit in a reference plane using the following basis vector selections:

Choose $\bar{Z}$ = north along the earth rotation axis
Choose $\bar{X}$ = the vector from the earth center through the orbit ascending node at the Equator.
Choose $\bar{Y} = \bar{Z} \times \bar{X}$ to form a right handed coordinate basis.

Figure 2 shows the relationship between the orbit plane and this coordinate system choice.

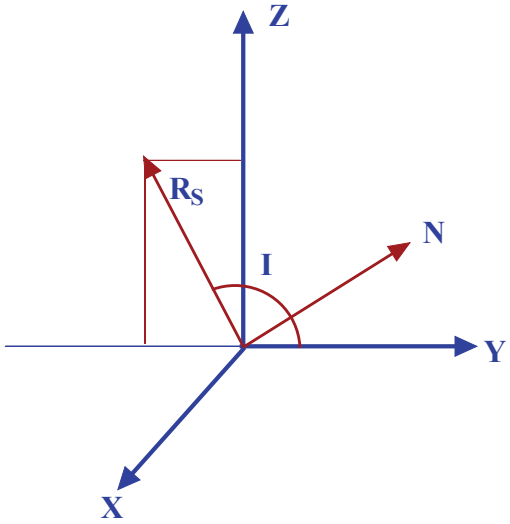

Figure 2: Selected inertial coordinate system. The orbit is in the X, R_S plane

In Figure 2, I is the orbit inclination angle measured from the earth rotation direction (98.58º for RADARSAT-2) and $\overline{R}_S$ is the satellite position vector.

The orbit vectors defined in this coordinate system are:

- The orbit plane normal unit vector defined in the direction of the satellite orbit angular momentum vector is

$$\widehat{N} = \begin{bmatrix} 0 \\ \sin(I) \\ -cos(I) \end{bmatrix}. \tag{6}$$

- The satellite position vector is

$$\bar{R}_S = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = R_S \begin{bmatrix} cos(\alpha) \\ cos(I)sin(\alpha) \\ sin(I)sin(\alpha) \end{bmatrix} = R_S \hat{R}_S \,. \tag{7}$$

- The Satellite velocity vector is

$$\bar{V}_S = \begin{bmatrix} \frac{dX}{dt} \\ \frac{dY}{dt} \\ \frac{dZ}{dt} \end{bmatrix} = V_S \left\{ \begin{bmatrix} -sin(\alpha) \\ cos(I)cos(\alpha) \\ sin(I)cos(\alpha) \end{bmatrix} \cos(\phi) + \begin{bmatrix} cos(\alpha) \\ cos(I)sin(\alpha) \\ sin(I)sin(\alpha) \end{bmatrix} \sin(\phi) \right\} \tag{8}$$

$$= V_S \{ \widehat{V'}_S \cos(\phi) + \hat{R}_S \sin(\phi) \} = V_S \hat{V}_S \,,$$

where $\widehat{V'}_S$. is the velocity unit vector that is normal to the satellite position vector.

The geocentric latitude of the satellite in this coordinate system is

$$\lambda_S = tan^{-1}\left(\frac{Z}{\sqrt{X^2+Y^2}}\right) = tan^{-1}\left(\frac{R_{SZ}}{\sqrt{R_{SX}^2+R_{SY}^2}}\right), \tag{9}$$

for satellite position vector $\bar{R}_S$.

2.3.3 Satellite attitude control model

The unit vectors $\hat{R}_S, \widehat{V'}_S, \widehat{N}$ form a right-handed coordinate system centered at the satellite and are referred to as the Orbit Reference Frame (ORF) in [1]. This is the coordinate frame used for satellite attitude control. The attitude control operation consists of a rotation of the satellite body about the yaw axis, $\hat{R}_S$, by the yaw angle, θ_Y, so that the long axis of the antenna lies along the unit vector

$$\hat{S} = \hat{V}_S \sin(\theta_Y) + \widehat{N}\cos(\theta_Y) \tag{10}$$

followed by a rotation by the pitch angle, θ_P, of the long axis of the antenna about the joint normal to $\hat{R}_S$ and $\hat{S}$ so that it lies along the unit vector

$$\widehat{M} = \hat{S}cos(\theta_P) - \hat{R}_S si\,n(\theta_P), \tag{11}$$

where both θ_Y and θ_P contain both control law terms and servo control error terms.

A residual Doppler shift (Doppler centroid) model needs to account for:

- azimuth and pitch biases due to misalignments between the star-tracker coordinate system and the effective RADARSAT-2 antenna coordinate system. The term “effective” has been

used as mechanical and electronic steering biases cannot be separated from measurements made on the radar signal data.
- RADARSAT-2 attitude control system response lags, and
- the yaw and pitch control laws.

RADARSAT-2 attitude control has been designed to minimize the effect of earth rotation by steering the radar antenna yaw angle according to the control law [1]

$$\Phi_Y = Y_a \cos(\alpha) \tag{12}$$

where, α, is the orbit internal angle (measured from the ascending node of the orbit) and Y_a is the constant coefficient, -3.919515º [1].

When the yaw steering control lag and the star-tracker coordinate misalignment with the antenna coordinates system are taken into account, the yaw steering angle for the antenna is

$$\theta_Y = Y_a \cos\left(\alpha - 2\pi \frac{V_S \cos(\phi)}{R_S} \tau_Y\right) + \delta_Y \tag{13}$$

where τ_Y is the yaw control servo lag in seconds and δ_Y is the antenna coordinate system yaw misalignment in radians. For single aperture configurations of the RADARSAT-2 antenna, $\delta_Y$is the yaw bias between the star-tracker and antenna coordinate references. For multiple-aperture configurations of the RADARSAT-2 antenna, δ_Y contains squint contributions [6] that are caused by electromagnetic boundary condition mismatches at the ends of the (two-way) apertures.

As the satellite travels around its orbit, its pitch angle is dynamically controlled to compensate the orbit ellipticity and to keep the radar pointed towards the earth. Information obtained from MDA indicates that the pitch angle of the satellite is controlled to keep the radar antenna oriented normal to the satellite position vector so that it lies in the $\widehat{V}'_S, \widehat{N}$ plane. The unit vector, $\widehat{V}'_S$ from equation (8) defines the desired pitch-compensated antenna attitude. For model calculations that compensate for the control lags of the pitch servo process and misalignments between the star-tracker and antenna coordinates, we have a model pitch angle of

$$\theta_P \approx \phi - 2\pi \frac{e_S \cos(\alpha)}{1+e_S \cos(\alpha)} \frac{V_S \cos(\phi)}{R_S} \tau_P + \delta_P \,, \tag{14}$$

where τ_P is the pitch control servo lag in seconds and δ_P is the coordinate system misalignment (pitch bias) in radians. Noting that e_S in equation (14) is the orbit eccentricity, the RADARSAT-2 pitch bias term is significantly larger than the pitch servo lag term for all reasonable values of τ_P.

Since the pitch and yaw control functions are servo responses, the angles expressed in equations (13) and (14) will have associated random variables that arise from attitude measurement errors and servo response errors. These random errors will determine the accuracy limit of model predictions. They must be estimated from model comparisons to a set of satellite radar measurements.

2.4 Range-angle geometry model

The satellite range observation-plane geometry is shown in Figure 3. The angle, A, is defined at the center of the earth and the angle, i, is defined with respect to the earth ellipsoid normal at the observed point on the ellipsoid surface. The angle, B, is the radar illumination angle for the observed point.

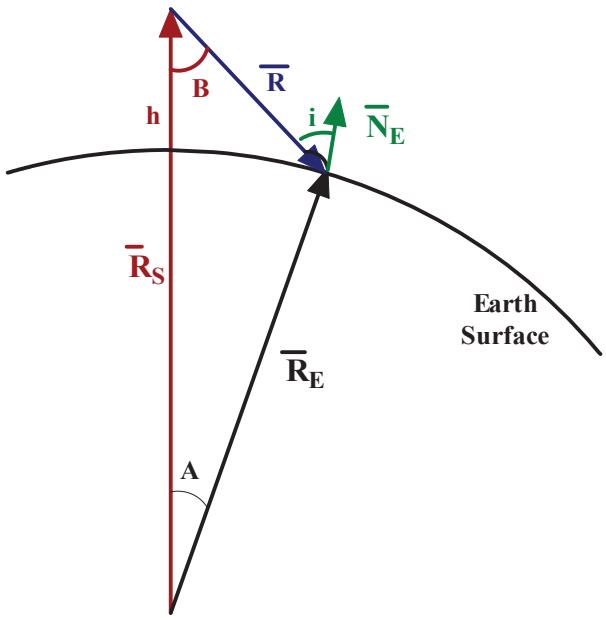

Figure 3: Observation plane geometry

In Figure 3:

$\bar{R}_E$ is the earth radius vector at the observation point.
$\bar{R}_S$ is the satellite position vector where $R_S=R_E+h$ and R_E is the sub-satellite earth radius.
h is the satellite altitude above the earth ellipsoid.
$\bar{R}$ is the radar range vector.
$\bar{N}_E$ is the local earth surface normal vector.
i is the satellite incidence angle at the observation point.
B is the satellite illumination angle measured from nadir at the satellite position

Given that $\bar{R}_S$ is known, to fully exploit the information in Figure 3, we need to know or derive either $\bar{R}$ or $\bar{R}_E$. The following relationships will be useful.

The radar illumination angle, B, can be calculated in two ways:

- If only the vector lengths are known, the cosine law can be used,

$$B = acos\left(\frac{R_S^2+R^2-R_E^2}{2RR_S}\right) \tag{15}$$

- If the vectors are known, B can be computed from the dot product between the unit vectors describing the satellite position and the radar range,

$$B = \text{acos}\left(-\hat{R}_S \cdot \hat{R}\right) \tag{16}$$

The local smooth-earth incidence angle on the reference ellipsoid is given by,

$$i = \text{acos}\left(-\hat{R} \cdot \hat{N}_E\right), \tag{17}$$

where $\hat{N}_E$ is the unit vector that is normal to the local, earth-ellipsoid at the target point.

The local earth radius vector can be computed from the satellite position and radar range vectors,

$$\bar{R}_E = \bar{R}_S + \bar{R}. \tag{18}$$

The local earth radius magnitude, R_E, can be computed from equation (1) if the geocentric latitude, $\lambda_{E,}$ is known at the target point.

Once the target point is known, its geodetic latitude on the earth ellipsoid can be estimated as

$$\lambda_G = atan2(R_{EZ}, (1 - e_E^2)\sqrt{R_{EX}^2 + R_{EY}^2}\,). \tag{19}$$

2.5 The model evaluation problem

Given the satellite state vector as a function of the orbit internal angle and given the radar range at some known point on the orbit, we need to estimate the earth radius vector at the target points as a function of orbit internal angle for constant radar range or for constant local incidence angle.

2.5.1 Known parameters

Since the variables are interdependent, use an iterative approach. Start from the constant range assumption. The known variables are:

- the orbit plane normal unit vector, $\hat{N}$, from equation (6),
- the satellite position unit vector, $\hat{R}_S$, from equation (7),
- the satellite distance from the earth center, R_S, from equation (3),
- the satellite velocity unit vector, $\hat{V}_S$, from equation (8),
- the satellite velocity unit vector that is normal to the satellite position vector, $\hat{V}'_S$, from equation (8),
- the satellite orbital speed, V_S, from equation (5),
- the satellite flight path angle, ϕ, from equation (4),
- the satellite look direction (right or left) of the satellite track,
- the radar range, R,
- the geocentric latitude of the satellite, λ_S, from equation (9) and the sub-satellite earth radius, R_{ES}, from equation (1), and

- the azimuth control law from equation (12).

We assume that the satellite pitch control law is given by, $\phi(\alpha)$ from equation (4), and we assume that angles are positive counter-clockwise.

2.5.2 Unknown parameter estimation

2.5.2.1 Constant range calculation option

This section steps through the model calculation sequence. Some equations that have been presented earlier in this document have been repeated here for clarity. It is assumed that the model calculations start from the ascending node of the orbit.

At the start of the estimation process, we do not know:

- the satellite illumination angle, *B* (equation (15)), at the radar target point,
- the target point coordinates on the earth,
- the earth radius vector at the target point,
- the earth-normal unit vector at the target point,
- either of the yaw or pitch coordinate system misalignments between the star-tracker used to measure satellite attitude, and
- the servo control lags of the satellite attitude control system.

The iterative estimation process uses satellite parameters to make an initial set of observation parameter estimates and then updates these estimates using the calculated values.

Set the coordinate misalignments and the servo control errors to zero for initial model calculations and use a locally-spherical earth approximation and the sub-satellite latitude from equation (9) to make the initial estimates.

The satellite geocentric latitude is

$$\lambda_S = atan2\left(\hat{R}_{SZ}, \sqrt{\hat{R}_{SX}^2 + \hat{R}_{SY}^2}\right). \tag{20}$$

The earth radius at the sub-satellite point from equation 1 provides an initial earth-radius estimate

$$R_{E1} = \frac{R_{Eq}\left(1-e^2\right)^{\frac{1}{2}}}{\left(1-e^2 cos(\lambda_S)^2\right)^{\frac{1}{2}}}. \tag{21}$$

The first estimate of the illumination angle to the target point uses the initial parameters and equation 15 to yield

$$B1 = acos\left(\frac{R_S^2+R^2-R_{E1}^2}{2RR_S}\right). \tag{22}$$

The first estimate of the radar range unit vector for a right-looking radar is

$$\widehat{R1} = \hat{R}_S cos(B1) + \hat{V'}_S sin(B1) sin(\Phi_Y) + \hat{N} sin(B1) \cos(\Phi_Y). \quad (23)$$

The first estimate of the earth radius vector at the target point is given by equation (18) using the initial variables,

$$\bar{R}_{E1} = R_s \hat{R}_S + R\widehat{R1} = R_{E1} \hat{R}_{E1}, \quad (24)$$

and the radar target point geocentric latitude, λ_E, is given by equation (19) with the components of $\hat{R}_S$ replaced by the components of $\bar{R}_{E1}$,

$$\lambda_E = atan2\left(\hat{R}_{EZ}, \sqrt{\hat{R}_{EX}^2 + \hat{R}_{EY}^2}\right). \quad (25)$$

The target point geodetic latitude, λ_G, is given by equation (20) using the components of $\hat{R}_{E1}$.

We can relate the inertial orbit-plane coordinates to ECEF (Earth-Centered, Earth-Fixed) coordinates by relating the ECEF reference longitude to the orbit ascending node longitude as the satellite progresses around its orbit. In our inertial orbit plane coordinate system, the relative longitude of the radar target point, defined by the target point position on a non-rotating ellipsoid is given by

$$\gamma_T = atan2(R_{EY}, R_{EX}) \quad (26)$$

and the local earth normal unit vector is

$$\hat{N}_E = \begin{bmatrix} \cos(\lambda_G) \cos(\gamma_T) \\ \cos(\lambda_G) \sin(\gamma_T) \\ \sin(\lambda_G) \end{bmatrix}. \quad (27)$$

The local incidence angle, i_G, is now given by equation (11).

Start from the latest illumination angle, B given by equation (22) with variables replaced by the latest values and iterate through the equation sequence (23) to (25), (20), (26), (27), (11), (21), (22) replacing the variables sequentially. Full convergence is achieved in two iterations. The final variables are $\lambda_E,\ \lambda_G, R_E, B, \hat{R}, \hat{N}_E, i$ and $\bar{R}_E$.

Under the constant range assumption, the RADARSAT-2 target point incidence angle at the reference ellipsoid varies around the orbit as illustrated in Figure 4 (for a constant radar range of 1000 km).

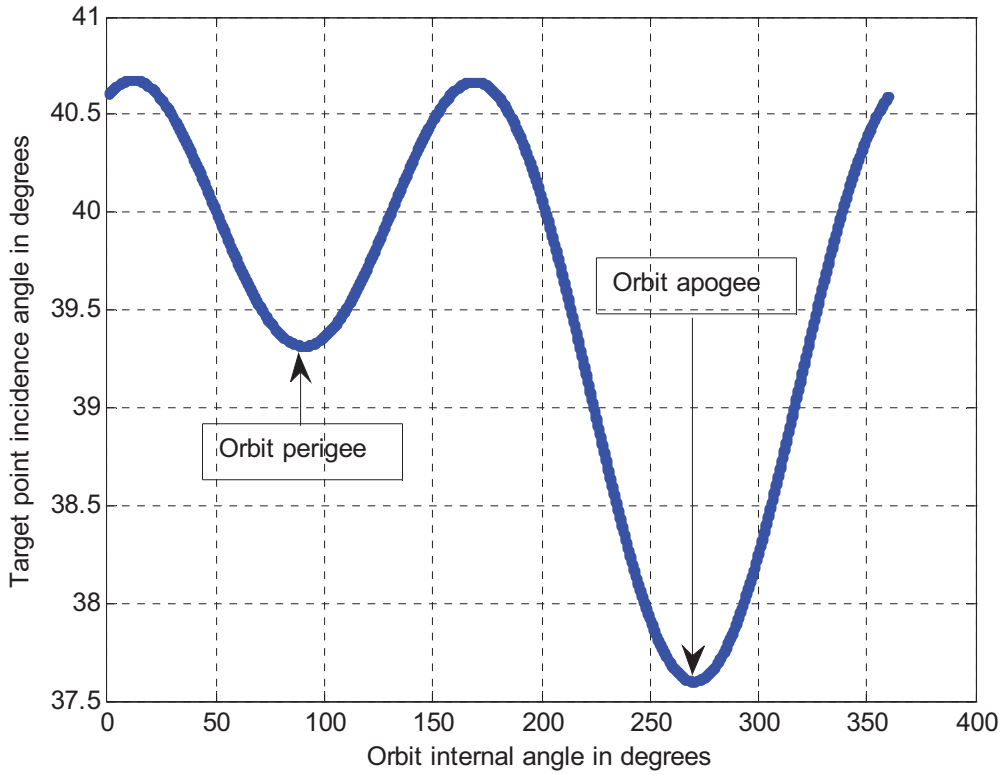

Figure 4: Target point incidence angle for 1000 km constant radar range

2.5.2.2 Constant incidence angle computation option

To perform the model calculations for a constant incidence angle at the target point, perform the calculations described in equations (20) to (27) and equation (11) to make initial estimates for constant range. Arbitrarily choose the incidence angle reference point at the ascending node of the orbit. The relationships between the radar observation parameters and the target-point incidence angle on an ellipsoidal earth are transcendental functions composed of inter-dependent parameters. Noting that the radar range for constant incidence angle is strongly coupled to the satellite height above the observed terrain, a first approximation to the radar range variation with orbit central angle that yields an approximately constant incidence angle at the observation points is given by

$$R \approx R_a \frac{R_S(\alpha) - R_E(\alpha)}{R_S(0) - R_E(0)} \tag{28}$$

where:

- R_a is the radar range at the ascending node.
- $R_S(0)$ and $R_E(0)$ are the satellite orbit radius and the corresponding earth radius when the satellite is at its ascending node.

- R_S and R_E are the satellite orbit radius and the target point earth radius at any orbit central angle.

The model radar range is estimated by equation (28) and then equation (22) (using current parameter values for R and R_E) is used to re-estimate the satellite illumination angle, B. An iteration through equations (22) to (27) and equation (11) provides parameter values for λ_E, $\lambda_G, R_E, B, \hat{R},\ R, \widehat{N}_E, i$ and $\bar{R}_E$ that are consistent with a nearly constant incidence angle as the satellite traverses around its orbit.

Figure 5 shows that the target point incidence angle is constant to within 0.075º when R is 1000 km at the ascending node. The incidence angle variation around the orbit is sufficiently small that further refinement of the radar range variability around the orbit is not necessary for this study.

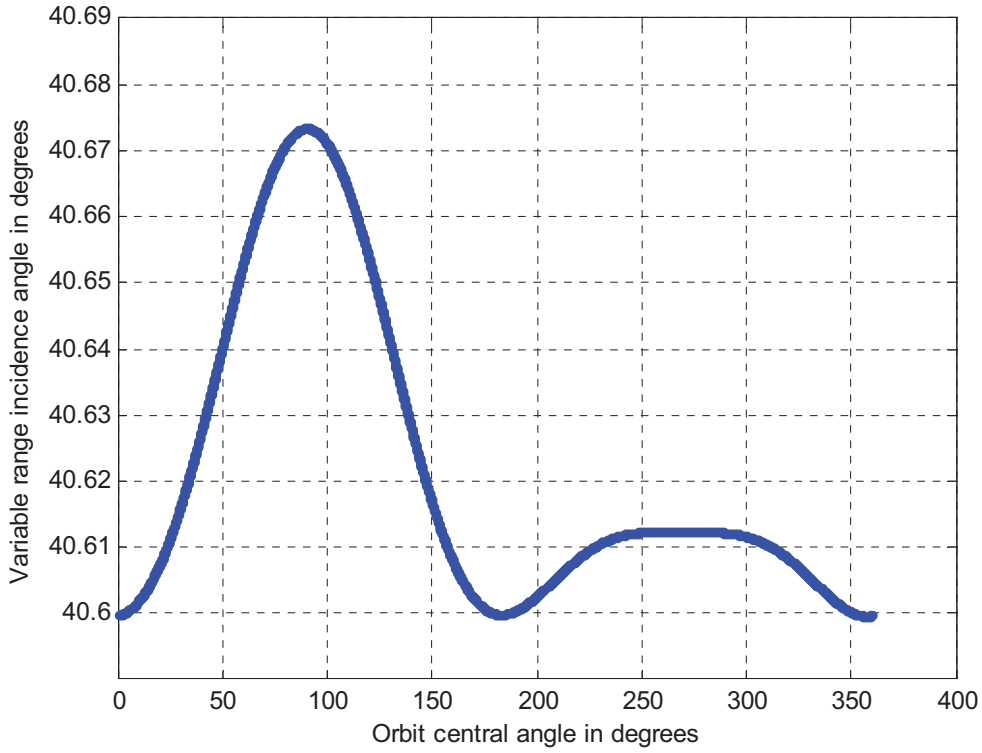

Figure 5: Target point incidence angle in degrees for one complete RADARSAT-2 orbit when the ascending node range is 1000 km.

3 RADARSAT-2 observation geometry for earth rotation compensation

To minimize earth rotation effects on space-based SAR measurements, velocities associated with the satellite motion and the imaging geometry that are projected along the radar range vector must be cancelled by a component of the earth rotation velocity vector projected along the radar range vector. The cancellation process is necessarily dependent on the radar range and, true cancellation can only be obtained at a single range for any realizable combination of the satellite orbit state vector, $[\bar{R}_S, \bar{V}_S]$, and satellite attitude state vector, $[\theta_Y, \theta_R, \theta_P]$, where θ_Y, θ_R, and θ_P are the satellite yaw, roll and pitch angles respectively.

The projected satellite and imaging geometry velocities are:

- The satellite orbital velocity compensated for the flight path angle,

$$\bar{V}_{Sm} = V_S \cos(\phi)\, \hat{V}'_S; \tag{29}$$

- the rate of change in satellite position vector length along the satellite position vector,

$$\bar{V}_{Rs} = \frac{dR_s}{dt} \hat{R}_S; \tag{30}$$

- and the rate of change of target point earth radius along the earth radius vector,

$$\bar{V}_{Re} = \frac{dR_e}{dt} \hat{R}_E. \tag{31}$$

The time increment that is used to compute the derivatives is

$$dt = \frac{2\pi R_S}{N V_S \cos(\phi)}, \tag{32}$$

where N is the number of satellite position samples (evenly spaced, orbit central-angle samples) used to compute the model for one complete orbit.

The satellite velocity projection component of the earth rotation compensation process can be generated by controlling the pitch and yaw attitudes of the radar antenna.

The earth rotation vector is defined at the target point using the vector from the earth's axis and the earth rotation rate,

$$\bar{V}_E = \Omega_E \sqrt{R_{EX}^2 + R_{EY}^2}\, \hat{V}_E\,, \tag{33}$$

where

$$\hat{V}_E = \begin{bmatrix} -\frac{R_{EY}}{\sqrt{R_{EX}^2+R_{EY}^2}} \\ \frac{R_{EX}}{\sqrt{R_{EX}^2+R_{EY}^2}} \\ 0 \end{bmatrix} \tag{34}$$

and R_{EX} and R_{EY} are components of the target point earth radius vector $\bar{R}_E$ using the coordinate definition shown in Figure 2.

The residual earth rotation velocity is computed along the radar range vector using

$$V_{Res} = (\bar{V}_{Sm} + \bar{V}_{Re} + \bar{V}_{Rs} + \bar{V}_E) \cdot \hat{R} \tag{35}$$

and the modeled Doppler centroid at the target point is given by

$$f_D = \frac{2V_{Res}}{\lambda} \tag{36}$$

for radar wavelength, λ.

A model calculation result for RADARSAT-2, right looking SAR with a constant slant range of 1000 km is shown in Figure 5. The model parameters used to generate Figure 6 are displayed in Table 3

Table 3: Model settings for Figure 6

Yaw coefficient	3.92º
Yaw lag	0 seconds
Yaw bias	0º
Pitch lag	0 seconds
Pitch bias	0º
Ascending node target range	1000 km
Constant range flag	0

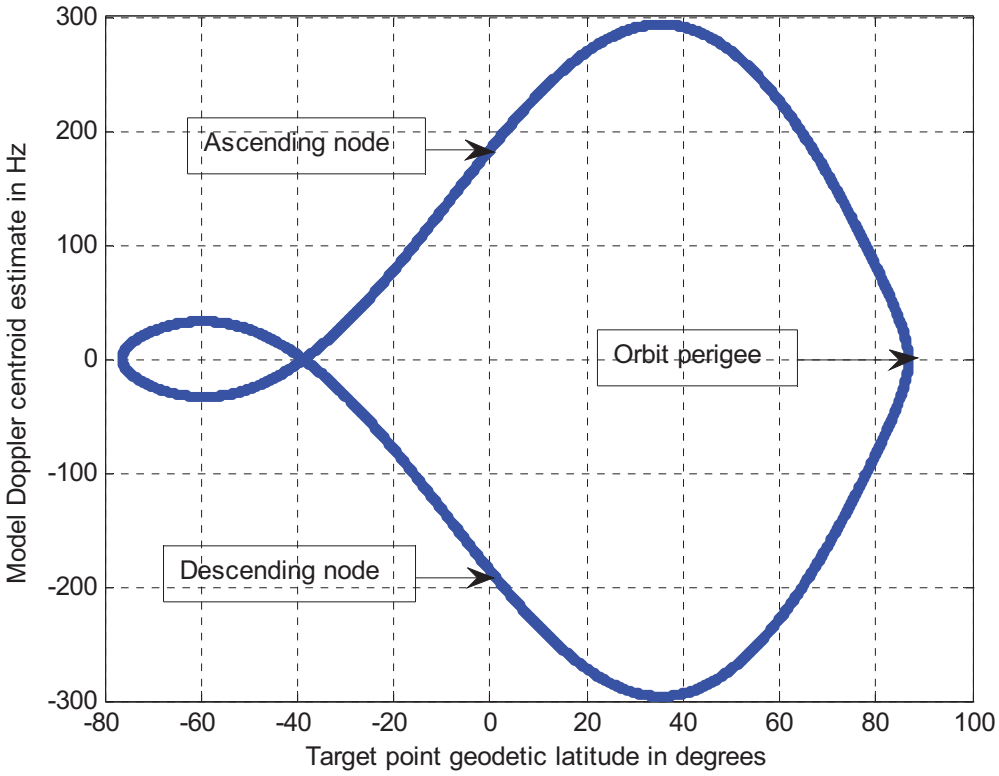

Figure 6: Model calculation results for right looking RADARSAT-2 with scene centers at 1000 km slant range.

When Figure 6 is compared to Figure 1, the Doppler centroid model is qualitatively similar to the measured data and represents a systematic Doppler frequency component that is present in single-look complex SAR data from RADARSAT-2.

4 Model validation study

The model validation study was based on RADARSAT-2 metadata from ten months of SAR data acquisitions and on the precision orbit state vectors that correspond to each of the data acquisitions.

Validation proceeded in three stages:

1. A preliminary data reduction stage acquired the precision orbit data and used these in conjunction with some of the metadata to generate model input parameters and measured scene-center Doppler centroid estimates.

2. Meta data provided by MDA were analyzed to verify model consistency.

3. Model calculations were performed for each metadata set to generate a theoretical Doppler centroid for each set, tune the model to the ensemble of metadata vectors and evaluate the degree of correspondence between the measured and predicted Doppler centroids.

4.1 RADARSAT-2 data acquisition metadata

Initial yaw steering model results were discussed with Alan Thompson (MDA) and he agreed to provide a metadata set from MDA data acquisitions between January 1 2010 and November 10, 2010 for model testing. An xml spreadsheet containing records from 65535 RADARSAT-2 data acquisitions was received and was translated into Matlab format to generate the measurement data-space, shown in Table 4, used for model tuning and evaluation.

Table 4: Matlab representation of the MDA metadata table that was used for model evaluation

Parameter name	Size	Data type
DC_minrange	65535 x 1	double
DopplerEstimateTime	65535 x 1	structure
StartTime	65535 x 1	structure
antennaPointing	65535 x 1	cell
dopplerCentroidCoefficients	65535 x 5	double
dopplerCentroidConfidence	65535 x 1	double
dopplerCentroidPolynomialPeriod	65535 x 1	double
dopplerCentroidReferenceTime	65535 x 1	double
satelliteHeight	65535 x 1	double
geodeticTerrainHeight	65535 x 1	double
imageCornerLat1	65535 x 1	double
imageCornerLat2	65535 x 1	double
min_range_est	65535 x 1	double
yawMeasurement	65535 x 1	double
pitchMeasurement	65535 x 1	double

orbfiles	65535 x 1	cell
productID	65535 x 1	cell
passDirection	65535 x 1	cell
pulseRepetitionFrequency	65535 x 1	double
rawDataStartTime	65535 x 1	cell
timeOfDopplerCentroidEstimate	65535 x 1	cell

All data vectors preserve the original sequence of the MDA table. The parameter name column in Table 4 describes the useful metadata vector components for each of the 65535 samples.

4.2 Orbit state-vector data and analysis

The StartTime and scene identification information in Table 3 were used to identify and acquire the RADARSAT-2 precision state vector files for all orbits represented in the data set. A small number, 187, of precision state vectors could not be retrieved and an examination of the acquired data revealed an additional small number, 592, metadata / precision-orbit-data combinations had various problems. Analysis was constrained to the remaining 64756 data points and a pointer vector, GoodPoint_ID, was generated to extract valid data from the total set.

The scene center for each data acquisition file (metadata vector) was computed by averaging the imageCornerLat and imageCornerLong parameters (the international date line must be accounted for when averaging longitude) and then transforming these into ECEF (Earth-Centered, Earth Fixed) Cartesian coordinates using a WGS 84 ellipsoid model to generate $\bar{R}_E$ for each scene center point in the data set. The parameter, sceneCenterLatitude, is the average image corner latitude.

The scene center time was estimated from the StartTime, imageCornerLat1, the sceneCenterLatitude and the pulseRepetitionFrequency parameters.

The state vector data in the vicinity of (within 5 minutes) the estimated scene center time were interpolated at the PRI (1/pulseRepetitionFrequency parameter) [2] and the state vector, $\bar{R}_{PCA}$, corresponding to the Point-of-Closest-Approach, PCA, (minimum slant range to the scene center point) was found. The Doppler centroid polynomial described by "dopplerCentroidCoefficients" was evaluated for time, t_{poly}, which is the difference between the PCA range time and the "dopplerCentroidReferenceTime", t_{ref}, to generate the parameter DC_{PCA}

$$t_{poly} = \frac{2|\bar{R}_{PCA}|}{c} - t_{ref} \tag{37}$$

where c is the speed of light and the symbol, | |, denotes the vector length.

The acquisition time (i.e. zero Doppler time accounting for squint due to DC_{PCA}) was then determined from the interpolated state vector (in the ECEF frame, where the earth surface velocity vector, $\bar{V}_E = 0$) for which:

(38)

$$f_{Dop} = \frac{-2}{\lambda}\left(\frac{(\bar{V}_S - \bar{V}_E)\cdot(\bar{R}_{PCA} - \bar{R}_E}{|\bar{R}_{PCA} - \bar{R}_E|}\right) - DC_{PCA} = 0$$

where $\bar{V}_S$ is the satellite velocity state vector at the time of closest approach, $\bar{R}_E$ is the earth radius vector at the scene center and λ is the radar wavelength.

The interpolated orbit state vector at the acquisition time, $\bar{R}_S$ was then used to calculate the radar range vector, $\bar{R} = \bar{R}_S - \bar{R}_E$, its length, R, and the radar illumination angle at each scene center, $B = acos(\bar{R}_S \cdot \bar{R})$.

The calculations were done in ECEF coordinates, as the slant range vector magnitude and the illumination angle are the same in the inertial (ECI) and ECEF frames.

The Doppler centroid polynomial was then re-evaluated with the acquisition time slant range, $\overline{R}$, and DopplerCentroid, was generated as an output parameter.

To properly calculate the orbit central angle in the inertial orbit plane (adjusted for earth-orbit position), the orbit state vector ($\overline{R}_S, \overline{V}_S$) and the slant range vector ($\overline{R}$) for each acquisition point were expressed in the Earth Centered Inertial (ECI) reference frame (Figure 2) by rotation of the ECEF basis about the $\hat{Z}$ (earth polar) axis by the Greenwich hour angle (GMST) [2].

The Ascending Node Vector expressed in the ECI system is:

$$\bar{N}_{asc} = \hat{Z} \times \bar{h} \tag{39}$$

where $\bar{h} = \bar{R}_S \times \bar{V}_S$ is the satellite specific angular momentum [5].

The central angle in the orbit plane, α, is given by the dot product of the Ascending Node Vector and the state vector position at the signal acquisition time,

$$\alpha = \text{acos}(\hat{N}_{ASC} \cdot \hat{R}_S) \tag{40}$$

where $\alpha = 2\pi - \alpha$ for satellite positions in the southern hemisphere.

The results of the calculations for each data acquisition point are captured in the Matlab data vectors. shown in Table 5. The Matlab scripts used for pre-processing are presented in Annex A.1.

Table 5: Model parameters obtained from precision state-vector analysis

Symbol	Parameter	Size	Type
Kv	GoodPoint_ID	65535x1	double
$\bar{R}_S$	SatPosn (ECEF vector)	65535x3	double
$\bar{V}_S$	SatVel (ECEF vector)	65535x3	double
$\bar{R}_E$	SceneEarthRadius (ECEF vector)	65535x3	double
λ_G	SceneCenterLatitude	65535x1	double
R	Range	65535x1	double
α	OrbitCentralAngle	65535x1	double
B	RadarIlluminationAngle	65535x1	double
DC	DopplerCentroid	65535x1	double

The parameter list in column 2 of Table 5 forms the model input data vector for each sample point. The parameter sample sequence in Table 5 is that same as the sequence in Table 4.

4.3 Model validation

Model calculations were performed in the satellite orbit plane coordinates described in Figure 2. The model calculations described in Section 3.3 were reorganized to use the data input from the MDA spreadsheet, as outlined in Table 4 and the data generated from the precision state vector files as outlined in Table 5. The model estimates the residual Doppler shift (theoretical Doppler centroid) at each scene center point in the input data set.

Since the number of left-looking data points is small, eliminate these and perform the validation calculations on the right looking data only. 63631 data points fit the selection criteria.

The left looking case can be accommodated in the model by rotating the range vector by 180º using the satellite position vector as the rotation axis and adjusting related parameters. Since the RADARSAT-2 beam pointing tables are different for the right and left looking cases and since the number of left looking cases are small, it was decided not to use these data for verification calculations.

The model calculations were repeated with different choices of model tuning parameters to get a best fit to the data starting from the values estimated for the calculations in section 3.3.

4.3.1 Model calculation algorithm description

The model implementation is described as a set of preparatory steps followed by an algorithm that is evaluated for each point in the data set. The internally documented Matlab script used to implement the model is presented in Annex A. A detailed description of the computation steps used for the model validation algorithm follows.

1. Load the data described in Tables 4 and 5.
2. Modify the imported pointer-vector, *Kv* (GoodPoint_ID), to select valid, right-looking data vectors using the, antennaPointing parameter from Table 4 to generate the pointer vector, *K*.

K retains the data order in Tables 4 and 5. The current data set contains 63631 valid, measured points.

3. Extract and rename valid right-looking parameters. Transposed vectors to order samples as columns. Convert all angles to radians.
    a. Scene center geodetic latitude λ_V = SceneCenterLatitude(*K*),
    b. Scene center earth radius *REv* = SceneEarthRadius(:,*K*),
    c. Orbit central angle α = OrbitCentralAngle(*K*),
    d. Radar illumination angle B = RadarIlluminationAngle(*K*),
    e. Scene center Doppler centroid *DC*=DopplerCentroid(*K*),
    f. Range to the scene center (km) R = Range(*K*)/1000,
    g. ECEF satellite position vector RSv = SatPosn(:,*K*),
    h. ECEF satellite velocity vector *VSv* = SatelliteVelocity(:,*K*).
4. Order the imported data in ascending orbit central angle. This is not necessary but it simplifies some subsequent analysis steps.
5. Define the model tuning parameters to be explored. Example values are shown with angles in degrees.
    a. Yaw angle constant, Ya=-3.919515,
    b. Yaw angle bias in degrees Yb=0.0,
    c. Yaw angle servo lag in seconds ylag=10,
    d. Pitch angle bias in degrees Pb=-0.02,
    e. Pitch angle servo lag in seconds plag=0.0,
    f. Model scale factor factor=2.92,
    g. Orbit central angle increments n=3600.
6. Define the model constants for RADARSAT-2. Convert all angles to radians for computation.
    a. Orbit semi-major axis (km) *Rsm* = 7167.07,
    b. Orbit inclination angle *I*=98.58º,
    c. Orbit eccentricity *E* = 0.001155,
    d. Argument of perigee *Per* = 89.72º,
    e. Orbital velocity constant *Kgm*=631.34816,
    f. Equatorial earth radius (km) *Req*=6378.1379,
    g. Equatorial earth angular speed *O1*=2π/86164.2,
    h. Earth eccentricity e = 0.0818191908.
7. Extract the model calculation variables from the metadata.
    a. Satellite orbit radius in km
$$Rss = \frac{\sqrt{\overline{RSv'} \cdot \overline{RSv}}}{1000},$$
    b. Target point earth radius in km
$$ree = \frac{\sqrt{\overline{REv'} \cdot \overline{REv}}}{1000}.$$
8. Define the calculation variables for an elliptical satellite orbit.
    a. True anomaly, *v*, equation (2),

b. Flight-path angle, ϕ, equation (4),
c. Theoretical orbit speed, *Vs,* equation (5),
d. Time increment for lag calculation, $dT = 2\pi Rss/(Vss \cdot \cos(\phi))$.

9. Define the satellite attitude.
   a. Yaw angle equation (13),
   b. Pitch angle equation (14).
10. Define reference vectors and parameters for each metadata point at index *i*.
    a. Orbit plane unit normal vector, $\widehat{N}(i)$, equation (6),
    b. Satellite position unit vector, $\hat{R}_S(:,i)$, equation (7),
    c. Satellite velocity unit vector, $\hat{V}_S(:,i)$, equation (8),
    d. Define the satellite pitch angle, Pit(i), equation (14),
    e. Define the satellite yaw angle, Yaw(i), equation (13).
11. Loop through the data set with index *i* to perform the model estimates at each metadata point. Calculations are performed in the inertial orbit plane.
    a. Estimate the radar range unit vector, $\hat{R}(i)$, using equation (23).
    b. Estimate the pitch-adjusted satellite position unit vector, $\hat{R}_P(i) = \hat{R}_S(i)\cos(\theta_P(i)) + \hat{V}_S\sin(\theta_P(i))$, for velocity calculations.
    c. Compute the scene-center earth-radius vector, $\bar{R}_E(i)$, using an adjusted form of equation (24), $\bar{R}_E(:,i) = Rss(i)\hat{R}_P(:,i) + R(i)\hat{R}(:,i)$.
    d. Compute the earth-radius unit vector,
    $$\hat{R}_E(:,i) = \frac{\bar{R}_E(:,i)}{\sqrt{\bar{R}_E(:,i)'\bar{R}_E(:,i)}}.$$
    e. Estimate the target point earth-center latitude, $\lambda_E(i)$, using equation (25).
    f. Estimate the target point geodetic latitude, $\lambda_G(i)$, using equation (20) with the variables replaced by the components of $\bar{R}_E(:,i)$.
    g. Estimate the relative longitude, $\gamma_T(i)$ of the target point in the coordinate system shown in Figure 2 using equation (26).
    h. Compute the WGS 84 earth-normal vector at the target point, $\widehat{N}_E(:,i)$, using equation (27).
    i. Estimate the target point incidence angle, $i_G(i)$, using equation (11).
12. The modeled Doppler centroid at the target point is estimated by projecting the satellite velocity, the earth rotation velocity, the orbit vertical velocity and the vertical velocity of the target point onto the radar range vector and summing them to compute the relative velocity of the target point with respect to the radar. Since each point in the metadata set corresponds to a unique set of observations, theoretically derived parameters need to be generated to compute the vertical velocity derivatives in equations (30) and (31).
    a. Define the angle increment, $d\alpha$, to be used for vertical velocity estimates to be
    $$d\alpha = \frac{2\pi}{n}.$$
    b. Define the time increment, $dt(i)$, that spans two angle increments and is used for vertical velocity estimates, to be

$$dt(i) = \frac{2\pi Rss(i)}{nVs(i)\cos(\phi(i))}.$$

c. The vertical velocity component of the satellite motion, $\bar{V}_{Rs}(:,i)$ is given by equation (30) where the vertical displacement of the satellite, $dR_S(i)$, over the two orbit central angle increments, $\alpha(i) + d\alpha$ and $\alpha(i) - d\alpha$, is estimated using equation (3).

d. The effective vertical velocity of the earth surface is approximated by computing the change in the sub-satellite surface over two increments of the orbit central angle as follows:

   i. Estimate the satellite position unit vector at $\alpha(i) + d\alpha$ and $\alpha(i) - d\alpha$ using the unit vector defined by equation (7).

   ii. Compute the sub-satellite latitude for each unit vector using equation (20).

   iii. The change in earth radius over the two orbit central angle increments, $dR_E(i)$, is the difference between the earth radii at the two latitudes as estimated by equation 1.

e. The effective velocity of the earth surface, $\bar{V}_{Re}(:,i)$ is given by equation (31).

f. The satellite velocity to be projected, $\bar{V}_{Sm}(:,i)$, is given by equation (29).

g. The earth rotation velocity vector, $\bar{V}_E(:,i)$ to be projected is given by equations (33) and (34).

h. The range-projected relative velocity sum, $V_{Res}(i)$, is given by equation (35).

13. The model Doppler centroid, $f_D(i)$, is given by equation (36).

4.4 Analysis results

When the model is tuned, the best model representation of the data, including scene center range effects, is shown in Figure 7 for the full set of right-looking test cases. The point spread in Figure 7 illustrates the radar-range dependence of the model.

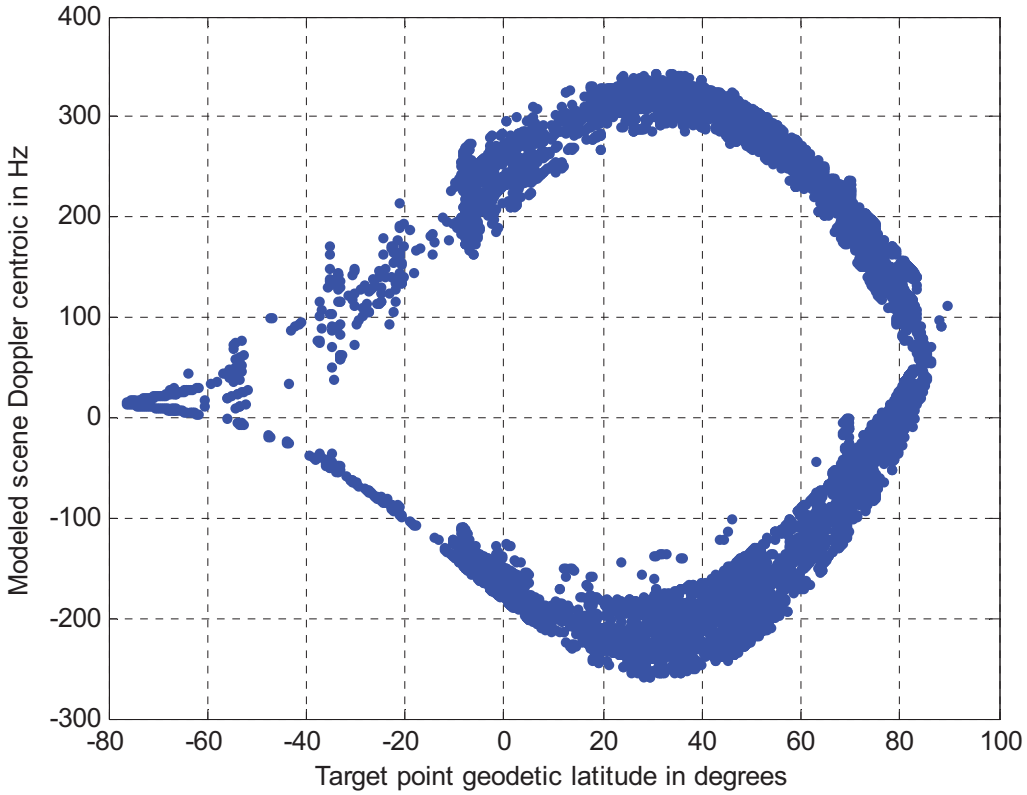

Figure 7: RADARSAT-2 Doppler centroid model outputs for data-base scene centers including radar range effects.

The superposition of the Doppler centroid model on the Doppler centroid metadata for the MDA test data set is displayed in Figure 8 for the model parameters shown in Table 6.

Table 6: Model parameters for Figures 7 and 8

Symbol	**Parameter**	**Value**	**Unit**
Ya	Yaw steering coefficient	-3.92	degrees
Yb	Yaw bias	0.01	degrees
ylag	Yaw servo lag	10	seconds
Pb	Pitch bias	-0.02	degrees
plag	Pitch servo lag	0.0	seconds
n	Orbit earth center angle increments for velocity estimates	3600	
factor	Model scale factor	2.92	

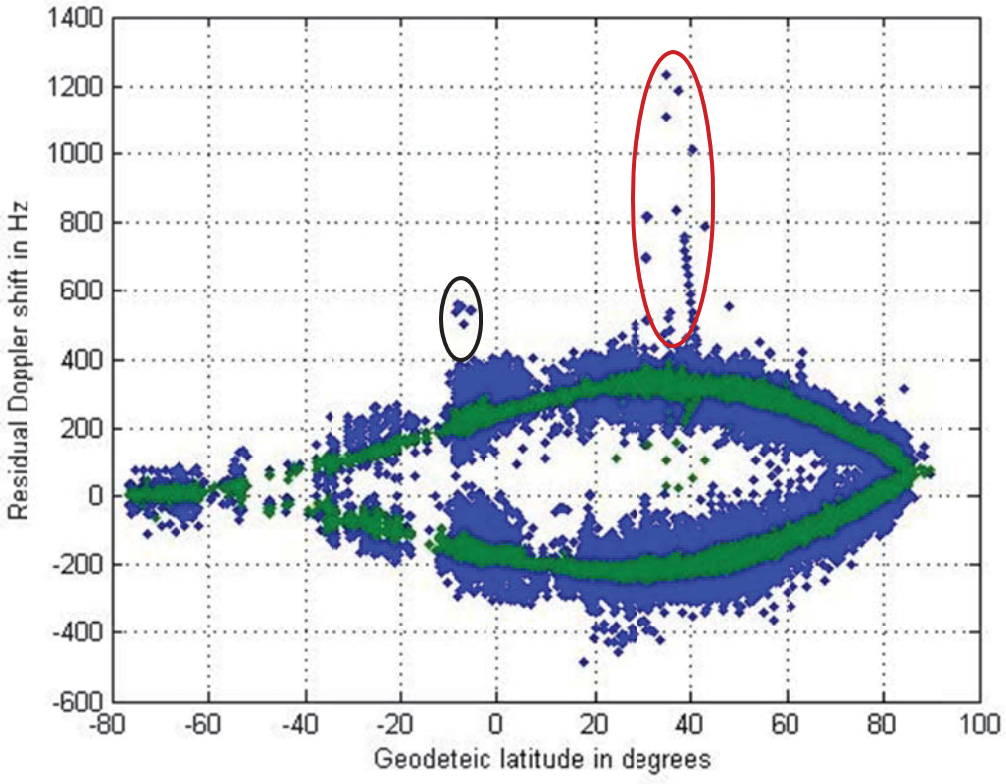

Figure 8: Measured (blue) and modeled (green) Doppler centroids for RADARSAT-2 data collections from January 1 to November 10, 2010

The ovals in Figure 8 highlight data anomalies. The points enclosed by the black oval have been identified as belonging to a radar calibration test over the Amazon and are believed to correspond to the process of stepping through sub-sets of TR modules on the antenna. The points enclosed in the red oval are situated near the eastern coast of North America and may be an intentional azimuth beam pointing effect.

Analysis of the sample difference between the measured and modeled Doppler centroid data shows that the mean difference between the measured and modeled Doppler centroids is -0.0296 Hz and that the standard deviation of these data is 46.7341 Hz.

A 200 bin histogram was computed for the difference between the measured and modelled Doppler centroids and is shown in Figure 9 with a superimposed Gaussian function defined by

$$N_{Samp} = 3406e^{\frac{-(f+0.0296)^2}{3650}} \tag{41}$$

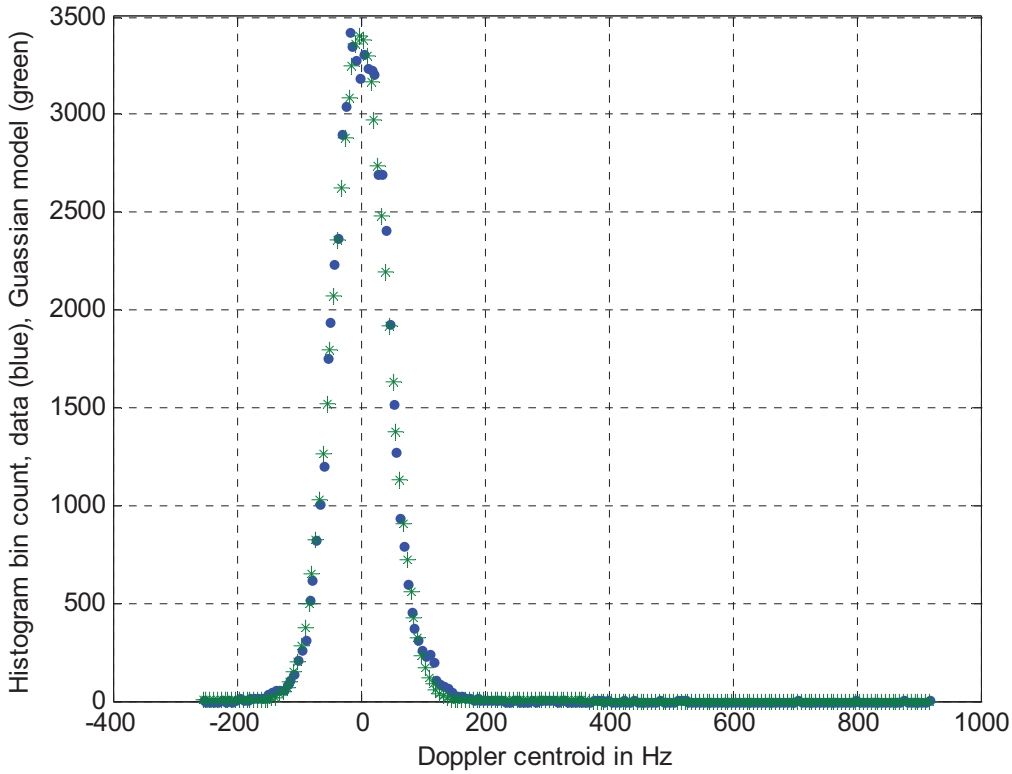

Figure 9: 200 bin histogram of measured Doppler centroids (blue dots), Gaussian model (green asterisks)

The difference of the measured and modelled Doppler centroid parameters is nearly Gaussian as is expected for a combination of random servo errors and random Doppler centroid measurement errors. The difference between the Doppler centroid estimation error and the Gaussian model, Figure 10, shows evidence of a weak (≤ 10% contribution to the distribution function describing the difference between the model and the measured Doppler centroids in the MDA metadata set), odd order component in the error distribution function.

The modeled Doppler centroids estimated by the algorithms presented in this document use precision orbit data to provide validated estimates of the systematic components of Doppler centroid frequencies in RADARSAT-2 SAR data.

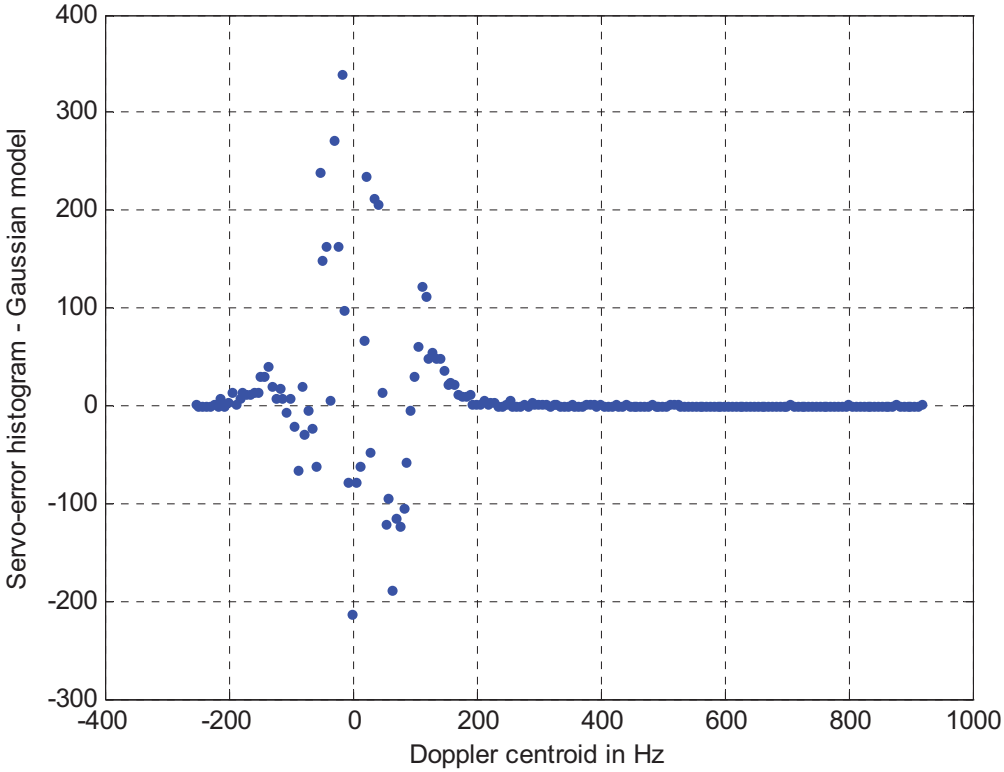

Figure 10: Servo error bin count minus its Gaussian model

5 Doppler centroid model for an ocean data set: user manual

The validated and tuned Doppler centroid model has been implemented in Matlab code for use in estimating the systematic part of Doppler centroids for any maritime SAR or GMTI data acquisition that is processed to a SLC (Single-Look Complex) data product. This code makes use of:

1. the .xml file, Product.xml, which is delivered with every RADARSAT-2 processed radar image;
2. target-point slant-range image coordinates extracted from a processed, strip-map, SAR image;
3. RADARSAT-2 state vector data extracted either from Product.xml or from a precision orbit file; and
4. a table (file) containing UT1 to UTC correction angles for the year containing the acquired data set.

Warning: The data extraction code reads a .xml file using a Java library. Java must be enabled in Matlab to run this code. Check the setting of the Matlab "no JVM" switch.

Warning: How Matlab accesses Java depends on the Matlab configuration used in the analyst's computer and where Java is located in the computer directory. Use an absolute path description to point to the directory that holds product.xml.

The software accepts instructions from the data analyst and input files and generates the Matlab data structure:

modelDataOut
- .rangeCoordinate
- .azimuthCoordinate
- .Glat
- .Lat
- .Re
- .Yaw
- .Pit
- .Doppler

For each target point, the data structure fields are defined as:

1. rangeCoordinate is the target-point range sample number measured from the first slant range in the image.
2. azimuthCoordinate is the target-point azimuth sample number measured from the first azimuth sample in the image.
3. Glat is the computed geodetic latitude (on the WGS 84 ellipsoid) of the target-point in degrees.
4. Lat is computed geocentric latitude (on the WGS 84 ellipsoid) of the target-point in degrees.
5. Re is the WGS 84 earth radius at the target-point in km.

6. Yaw is the modelled satellite yaw angle at the time of the target-point data acquisition (center of the azimuth matched filter) in degrees.
7. Pit is the modelled satellite antenna pitch angle at the time of the target-point data acquisition (center of the azimuth matched filter) in degrees.
8. Doppler is the modelled Doppler centroid at the target-point in Hz.

The data structure, modelDataOut, contains one set of fields for each input target point.

The software used to generate the model-based Doppler centroid data for selected target points consists of two major blocks:

1. The data preparation block, described in Section 5.1 transforms input information into the data structure needed to run the Doppler centroid model algorithm.
2. The Doppler centroid model block, described in Section 5.2, generates model outputs for each target point.

All Matlab code needed to set up and run the Doppler centroid model is described in Annexes B and C. The required Matlab functions and scripts are contained in the CD attached to this report. These can also be entered manually from Annexes B and C. All functions are internally documented by comments and are designed to provide user guidance in response to the Matlab command: help function_name.

The model code and the associated function library should be placed in a user directory whose path is defined for Matlab either by including this directory into the default Matlab path definition or by using the command path (path,'literal path string')

The model is run by the Matlab script, runDopplerCentroidModel.m, which calls the user-edited specification script, model_specifications.m, calls the data preparation function, model_dat_prep.m and then calls the function model_Doppler.m to generate the output data structure.

5.1 Data sources

5.1.1 Image and product.xml

All RADARSAT-2 data sets received from the archive or from data acquisitions, for example PDS_0036896_20090119, have processed image file directories that contain several files. Two of these files, imagery_HH.tif (the last two letters can be HH, VV, HV) and product.xml are data sources for the Doppler centroid model calculations.

5.1.2 Precision orbit data

Precision orbit data can be retrieved from the MDA orbit data repository. The process needed to access these data can be found in the ATP user manual [9], Annex H. The analyst will need to set up an account and a password to retrieve these data (E-mail clientservices@mdacorporation.com for help in setting up an account).

The precision orbit data file can be placed in the data product directory or in a separate directory. The software deals with both cases.

5.1.3 UT1 to UTC correction

The UT1toUTC correction file, UT1corr_file='./UT1-UTC_2007to2011.csv'; is an ASCII .csv (comma separated values) file containing 2 columns. The first column is the Modified Julian Date (fraction of the day is not necessary) and the second column is the UT1-UTC difference in seconds for that day. The file can have from 1 to N lines, where N is the number of days spanning the date range to be loaded.

The data can be obtained from the US Naval Observatory EO database search page at http://maia.usno.navy.mil/search/search.html. This database contains Earth orientation data from 1-2-1973, to present. Just enter a date range and check the box "Bull. A UT1-UTC (sec. of time)".

The directory used to store this file is the analyst's choice. The path to the directory must be included in the data retrieval script.

5.2 Data preparation

The data preparation operation uses the contents of a Matlab script, model_specification.m, which has been edited by the user to define the current model calculation problem.

During the editing operation the user needs to provide the following parameters:

1. input_targets
   input_targets is an nx2 matrix that contains the image range and azimuth sample numbers of a user-selected target point in each row. If no target points have been defined for Doppler centriod estimation this variable should be an empty matrix, []. The software will then compute the Doppler centroid for the center of the SLC image. The target points in image_data can be in random order with respect to the image display.

   For example, input_targets=[2320,8772;1123,11000;5700,5123] defines three target points for Doppler centroid estimation.

2. Tgt
   Tgt is a switch which can have values 0 or 1 (Since the zero condition is tested, 1 can be replaced by any other number). The value Tgt = 1 tells the software that the target points have been extracted from the SLC image that was delivered from the SAR processor. The value Tgt = 0 tells the software that the image data were in data-acquisition order (1,1 is near-range and first azimuth line). MODEX data can be in either order.

3. data_path
   data_path is a literal variable that gives the absolute path to the directory that holds the files product.xml and orbit_data or just product.xml.
   For example:
   data_path='C:/media/DATAPART1_/R2-MODEX/PDS_0051229_20090518/'.

4. Orb

Orb is a switch that tells the software where the precision orbit file, orbit_file, is located. The setting, Orb = 1, says that orbit_file is in the same directory as product.xml. the setting, Orb = 0, says that orbit_file is located in a different directory and that the path to this directory in included in the file name.

5. orbit_file
   orbit_file is a literal variable that contains the name of the file containing precision orbit data. If the precision orbit file does not exist, enter the literal '0' and the state vector data will be extracted from the file product.xml. For example: orbit_file='07444_def.orb'.

   Alternatively, the orbit file name can contain the path to the orbit file to form a string of the form: 'C:/Work/Test_data/Definitive_Orbit_Files_08_10_acquisitions/09578_def.orb'.

   The orbit file name must correspond to the data acquisition that generated the product.xml file.

6. UT1corr_file
   The UT1corr_file is a literal variable that contains the path and the name of the UTC-UT1 correction file for ECEF to ECI longitude adjustment. If the UT1corr_file is not available, set the literal path and file name to `'0'`

   For example, UT1corr_file='C:/data/reference/UT1-UTC_2007to2011.csv' , could be a valid string.

7. UTCend
   UTCend contains the integer year following the last valid year in the UTCcorr_file. If the file contents lie outside of the valid date interval, the software sets the longitude correction to zero.

   For example, UTCend= `2012;`

8. sel
   sel is a switch that tells the software whether to use the satellite state vector velocity or a theoretical model of this velocity in running model_Doppler.m. The value sel=1 tells the model to use the state vector velocity data and is recommended if precision orbit data is used. sel=0 tells the model to use theoretical satellite velocity estimates and is recommended if state vector data is to be extracted from product.xml.

After model_specification.m has been edited and saved in the work space, the function model_data_prep.m calls the function model_Init.m to open the specified files, read them and extract parameters for use in analysis. model_dat_prep.m then performs intermediate calculations and interpolations to generate the indexed data-structure, modelDataIn which has the form:

modelDataIn(k) (data-structure with fields)

.rangeCoordinate	(the range sample number measured from the left edge of the image)
.azumuthCoordinate	(the sample number measured from the top of the image);
.range	(the radar slant range at the target point in m).

.radarPosition	(a column data vector, [X; Y; Z], for the target point, containing the ECI position of the satellite in m in the model coordinate system)
.radarVelocity	(the ECI velocity column vector,[Vx; Vy; Vz]:) in in m/s for the target point radar position)
.orbitCentralAngle	(the angle in degrees from the ascending node of the radar to the measurement point),
.PRF	(the radar pulse repetition frequency in Hz),
.pitch	(the reported pitch angle in degrees),
.Loo	(the radar look direction 'right' or 'left),
.ECEFVel	(the satellite velocity column vector in m/s that corresponds to the target point, k),
.ECEFPos	(the satellite position column vector that corresponds to the target point, k).

The data-structure index, k, is the row containing the target data in the matrix, input_targets.

Since the model is run from a Matlab script, the structure modelDataIn remains in the workspace for further use after the model execution is completed.

Internally-documented Matlab code for the scripts model_specification.m, and runDopplerCentroid Model.m, and for the functions model_data_prep.m and model_Init.m is described in Annexes B1.1, B1.2, B2.1 and B2.2 respectively.

5.3 Model execution

Once the data preparation calculations are complete, the control function, runDopplerCentroidModel.m executes the command modelDataOut = model_Doppler(inputData,sel).

The function model_Doppler.m generates the output data structure modelDataOut as an indexed data structure as described previously in Section 5

The index number is the target point row in input_targets.

The function, model_Doppler.m, executes the algorithms described in Sections 2.5.2.1, Section 3 and Section 4.3.1 of this Technical Memorandum for the constant-range condition (a different range is defined for each point) and a single orbit-internal-angle value for each target point, with the following changes:

1. If the command variable, sel > 0, the interpolated, measured, satellite orbit position and velocity vectors (derived from precision orbit data) are used in the model calculations and the definitions of the satellite velocity unit vectors are modified to reflect this change. If sel = 0, the satellite velocity vector is replaced by its theoretical value for a RADARSAT-2 elliptical orbit as discussed in Section 2.3.1.
2. Vertical velocity estimations use orbit central angle and time increments computed from the satellite speed, local orbit radius and the radar pulse repetition frequency. $dt = \frac{2}{f_{Samp}}$, and $d\alpha = \frac{V_S}{f_{Samp} R_S}$.
3. The measured satellite pitch angle is included in the pitch angle definition.

4. The radar range definition has been modified to accommodate left-looking radar cases.

The internally documented Matlab code for the function, model_Doppler(inputData,sel).m, is described in Annex B.3.

6 Conclusions

The model validation work described in this document is based on approximately ten months of RADARSAT-2 SAR acquisitions between January 1, 2010 and November 10, 2010. All conclusions of this study are based on this data set as a representative ensemble describing RADARSAT-2 imaging performance. Some properties of the satellite are known to have weak seasonal influences (solar illumination angle effects) and parameters will slowly change over time as components age (Observations reported by MDA [10] confirm the presence of a small, seasonal dependence in pitch and yaw estimates that has shown consistent trends in recent years). Neither seasonal effects nor component aging effects were captured in this study, however, sufficient detail has been included in this document to allow the work to be replicated for other data sets as part of a more detailed study of satellite properties.

The attitude steering model implemented on RADARSAT-2 has systematic behaviour that reasonably follows the nominal steering assumptions based on satellite design data.
Yaw angle = $-3.92^{o}\cos(\alpha)$ (for orbit central angle, α) and
Pitch angle = $\phi(\upsilon)$ (from equation (4)).

When the observation geometry is accounted for, there is a servo control lag between the observed yaw steering angle and the yaw-steering angle expected from the yaw control law of approximately10 seconds.

The pitch angle servo errors are too small to be observable with the available data and the nominal servo lag is set to zero seconds.

The yaw bias angle between the star-tracker coordinate system and the antenna coordinate system is best described as 0.01 degrees for normal SAR operation. There is independent evidence that there is a weak yaw bias effect between two-way GMTI apertures [6].

The pitch bias angle between the star-tracker coordinate system and the antenna coordinate system has a nominal value of -0.02º from model fitting analysis.

Analysis of the model performance shows that:

1. The model can estimate the mean, measured Doppler centroids to -0.0296 Hz.
2. The standard deviation of the difference between the measured and modeled Doppler centroids is 46.73 Hz. This is interpreted as a measurement of the sum of:
    a. the random component of the satellite attitude control (servo error),
    b. beam pointing errors due to other mechanisms such as:
        i. coordinate system misalignments and
        ii. digital beam-forming artefacts,
    c. Doppler shifts due to motion of radar clutter sources and
    d. Doppler centroid measurement errors embedded in the input metadata.

If the standard deviation is attributed to the servo control error in the yaw control system, the yaw pointing error is ±0.0098º which is well within the design specifications of the radar [2].

The attributed servo error effects have a nearly Gaussian distribution with a weak (<10%) odd-order-dominant, additional term.

The Doppler centroid model developed and tested in this report is adequate for use in marine GMTI measurements when scene Doppler centroid data are unsuitable for use as stationary-target references.

The Matlab software used to evaluate the model is documented in Annex A to allow the model validation analysis to be replicated for other metadata sets.

The model has been adapted for use by data analysts who need to estimate Doppler centroids for ocean data scenes. Section 5 is written as a user manual for the adapted Matlab code and Annexes B and C contain all of the Matlab software needed to operate the model for ocean data analysis.

References

[1] RADARSAT-2 reference frames and attitude parameters, RSATMDA-148, Dec 16, 2002.

[2] Ali, Z., G. Kroupnik, G. Matharu, J. Graham, I. Barnard, P. Fox, and G. Raimondo, RADARSAT-2 space segment design and its enhanced capabilities with respect to RADARSAT-1*, Can. J. Remote Sensing* V 30, No. 3, P. 235-245, 2004.

[3] Beaulne, P.D. and C.E. Livingstone, Evaluation of RADARSAT-2 Yaw Steering for SMTI Applications, Proc. EUSAR 2010, Aachen, June 7-10, 2010.

[4] Beaulne, P.D. and I.C. Sikaneta, A simple and precise approach to position and velocity estimation of low earth orbit satellites, DRDC Ottawa TM 2005.250, December 2005.

[5] Wertz, J.R. and W.J. Larson (Ed), Space mission analysis and design third edition, Microcosm Press (Hawthorn, CA) and Springer (New York, NY), ISBN 978-1881883-10-4, 2008.

[6] Chiu, S. and M.V. Dragosevic., Moving target indication via RADARSAT-2 multi-channel.synthetic aperture radar processing., *Advances in Signal Processing*, V. 2010, 19 P., 2010.

[7] Vallado, D.A., Fundamentals of Astrodynamics and Applications, Space technology Library, Microcosm Press 2001, ISBN 1-881883-12-4.

[8] "Department of Defense World Geodetic System 1984", National Imagery and Mapping Agency, NIMA TR8350.2, Last updated June, 2004.

[9] RADARSAT-2 Acquisition Planning Tool user manual, Rev. 1/3, MDA client services, RK-MA-52-1492, March 21, 2011.

[10] Williams, D., MDA Image Quality Group, reviewers comments, March 2012.

This page intentionally left blank.

Annex A Matlab functions and scripts used for model validation calculations

This Annex contains the Matlab code that was used to generate the data points used for the RADARSAT-2 Doppler centroid model validation study. The code modules presented here are working research software and have not been optimized or refined. They are provided so that the calculations describe in this report can be easily replicated on other data sets.

The code modules described in annexes A.1, A.2, A.3 and A.4 are intended to be run in the sequence presented as each program generates data required by the following program.

A.1 Source data preparation modules

The data sources that were used to generate model input data were:

1. an MDA Excel spreadsheet that contained 65535 metadata samples from all RADARSAT-2 acquisition between January 1, 2010 and November 10, 2010,
2. a table that gives UT1 to UTC time corrections for each day of 2010,
3. Precision orbit files for every orbit contained in the MDA spreadsheet, and
4. The vector orbfile(:) which contains the paths to the orbit files.

This section contains the Matlab script that was used for initial data preparation. It generates the parameters found in Table 5 of Section 4.2

Author: Pete Beaulne, DRDC Ottawa October 2011.

```
% User defined functions called directly by the script
        % lla2ecef.m
        % read_MDA_orbit.m
        % absVec.m
        % interState.m
        % evaluate_DC.m
        % convtime.m
        % ecef2eci.m
% This script calculates the satellite state vector at acquisition time, the slant range, the DC and
%the elevation and orbit central angles from the MDA scene metadata and the definitive orbit
%files. Outputs are fed to the DC model.
%
% constants …
% convert radians to degrees
r2d=180/pi;
% Convert degrees to radians
d2r=pi/180;
% Speed of light in free space in m/s
c = 299792458;
%Radar wavelength in m
 lambda=5.5465772433E-02;
```

```
%
% Required conditions
%This script requires that the following files and variables are in the Matlab work space:
% MDA_DC_spreadsheet0.mat
%ut1utcdiff_2010.csv
%porbfile, path to the directory containing ASCII precision orbit files whose name prefix is
'2010-'
%If different metadata are used, the script must be edited appropriately to access the data
%
% Import spreadsheet parameters that were used to generate Table 4 in Section 4.1
% The Excel data have been previously converted to Matlab format.
%
load MDA_DC_spreadsheet0.mat
%
% Import a 365 element array of UT1-UTC corrections for each day of 2010. Needed to convert
from ECEF to ECI coordinates.
%
load ut1utcdiff_2010;
%
% number of scenes to process …
Nscenes = length(geodeticTerrainHeight);
% Initialize successful processing flag
success = ones(Nscenes,1);
%
% Define output quantities …
GoodPoint_ID=[];
SatPosn = -999*ones(Nscenes,3);
SatVel = -999*ones(Nscenes,3)
Range= -999*ones(Nscenes,1);
OrbitCentralAngle = -999*ones(Nscenes,1);
RadarIlluminationAngle = -999*ones(Nscenes,1);
DopplerCentroid = -999*ones(Nscenes,1);
%
% calc. geodetic (lat, long, alt) at scene centre, in radians from input MDA spreadsheet entries in
degrees
%
ctr_lla=-999*ones(Nscenes,3); % initialize
ctr_lla(:,1) = d2r*(imageCornerLat1+imageCornerLat2)/2;
ctr_lla(:,2) = d2r*(imageCornerLong1+imageCornerLong2)/2;
ctr_lla(:,3) = geodeticTerrainHeight;
%
% correct longitudes around antimeridian by doing average using angles in [0,360] range instead
of [-180,180]
jj1=find(imageCornerLong1<-90 && imageCornerLong2>90);
jj2=find(imageCornerLong2<-90 && imageCornerLong1>90);
imageCornerLong1(jj1) = 360+ imageCornerLong1(jj1);
imageCornerLong2(jj2) = 360+ imageCornerLong2(jj2);
% corrected longitude back to radians
```

```
ctr_lla(jj1,2) = d2r*(imageCornerLong1(jj1)+imageCornerLong2(jj1))/2;
ctr_lla(jj2,2)= d2r*(imageCornerLong1(jj2)+ imageCornerLong2(jj2))/2;
Clear jj1 jj2;
%
% convert antimeridian area longitudes back to [-180,180], then convert to radians
jj=find(ctr_lla(:,2) > pi);
ctr_lla(jj,2) = ctr_lla(jj,2) – 2*pi;
clear jj;
%
% Latitude, longitude, terrain height converted to ECEF xyz
ctr_xyz = lla2ecef(ctr_lla);
%
% calculations on a scene by scene basis …
%
for ii=1:Nscenes
        % read in definitive orbit file
        % orbit file path and name, The path and filenames are pre-stored in the vector orbfile.
        %File names start with ‘2010-‘For data sets in other years change this
        porbfile=char(orbfiles(ii));
        DefState=read_MDA_orbit(porbfile,'2010-');
        %
        % skip if orbit file missing
        if(isempty(DefState)),
                success(ii)=0;
                continue;
        end
%
        state_timestruct = [DefState(:).time];
        state_times = [state_timestruct(:).MJD];
        clear state_timestruct;
        %
        % find state vectors within 5 minutes of start time calculations using Modified Julian
        %Date to avoid errors due to date change. 86400 is the number of seconds per Julian Day
        subsetID=find(state_times > (StartTime(ii).MJD-900/86400) & …
        state_times < (StartTime(ii).MJD+900/86400));
        subsetL=length(subsetID);
%
        % use this subset of the orbit file to find PCA vicinity by minimizing the
        % ECEF slant range (Rvec_ctr(x,y,z) – Rvec_sat(x,y,z))
         %
         tmpSATPOS=[DefState(subsetID).position]';
         tmpRS=repmat(ctr_xyz(ii,:),[subsetL,1]) - tmpSATPOS;
        [tmpR0, tmpR0id] = min(absVec(tmpRS));
%
        % extract 2 state vectors on either side of min for    %interpolation
        State = DefState(subsetID(tmpR0id-2):subsetID(tmpR0id+2));
         clear tmp* DefState;
%
```

```
        % set up interpolation at PRI for 5 second
        % period around PCA state vector
        N = ceil(5*pulseRepetitionFrequency(ii); %  20000;
         if(mod(N,2));
                N = N+1;
         end
        % N even
        %
         pulse_times = State(3).time.utcsec + (-1*floor(N/2):floor(N/2)-
1)/pulseRepetitionFrequency(ii);
        %
        % build pulse time structure to pass to interState
        for ipulse = 1:N
                pulsetime(ipulse).year = State(1).time.year;
                pulsetime(ipulse).doy = State(1).time.doy;
                pulsetime(ipulse).month = State(1).time.month;
                pulsetime(ipulse).day = State(1).time.day;
                [h,m,s] = sec2hms(pulse_times(ipulse));
                pulsetime(ipulse).hour = h;
                clear h;
                pulsetime(ipulse).minute = m; clear m;
                pulsetime(ipulse).second = floor(s);
                pulsetime(ipulse).usec = 1e6*(s-floor(s)); clear s;
                pulsetime(ipulse).utcsec = pulse_times(ipulse);
                        pulsetime(ipulse).JD = jday(pulsetime(ipulse).year,
                        pulsetime(ipulse).month, pulsetime(ipulse).day, ...
                        pulsetime(ipulse).hour, pulsetime(ipulse).minute,
                        pulsetime(ipulse).second+pulsetime(ipulse).usec/1e6);
                pulsetime(ipulse).MJD = pulsetime(ipulse).JD - 2400000.5;
        end
        clear pulse_times;
        %
        % do the interpolation …
        interp_statevecs=interState(State,pulsetime(:));
        %
        % calculate range and range rate at each pulse
        % loop over pulses
        for kk=1:N
                %
                Rs_xyz(kk,:) = ctr_xyz(ii,:) -interp_statevecs.position(:,kk)';
                rs_calc(kk) = absVec(Rs_xyz(kk,:));
                rs_dot(kk) = dot(-1* interp_statevecs.velocity(:,kk)',Rs_xyz(kk,:)/rs_calc(kk));
        end
        % end of loop over pulses
        %
        clear interp_statevecs;
        %
        % slant range to scene centre at PCA
```

```
[Rs_PCA,R0id]=min(rs_calc);
%
% evaluate doppler centroid at PCA
polytime = 2*Rs_PCA/c - dopplerCentroidReferenceTime(ii);
DC_PCA = evaluate_DC(dopplerCentroidCoefficients(ii,:),polytime)
clear polytime;
% range rate due to PCA centroid
rs_dotDC = -lambda*DC_PCA/2;
% loop over pulses and
% calculate doppler frequency accounting for centroid
for kk=1:N
        fdc(kk) = -2*(rs_dot(kk)-rs_dotDC)/lambda;
        end
        %
        % calculate true acquisition time interpolated pulse index
         % ie true zero Doppler crossing in presence of squint due to DC_PCA.
        [tmp,DCid]=min(abs(fdc));
        %
        % time, sat pos and velocity and slant range at acquisition
        TIME_ACQ = pulsetime(DCid);
        SatPosn(ii,:) = interp_statevecs.position(:,DCid)';
        SatVel(ii,:) = interp_statevecs.velocity(:,DCid)';
        Range(ii) = rs_calc(DCid);
        %
        % evaluate doppler centroid at acquisition time
        polytime = 2*Range(ii)/c - dopplerCentroidReferenceTime(ii);
        DopplerCentroid =.evaluate_DC(dopplerCentroidCoefficients(ii,:),polytime)
        clear polytime;
        %
        % elevation angle from range vector - sat vector dot product
        RadarIlluminationAngle(ii) = acos(dot(Rs_xyz(kk,:),-SatPosn(kk,:)) …
        /(Range(ii)*absVec(SatPosn(kk,:)));
        %
        % for conversion to inertial coordinates ... get UT1 times (correct for delta from
        %UTC), calculate GMST – needed to calc orbit central angle in inertial frame
        %
        Dut1utc = ut1utcdiff_2010(TIME_ACQ.doy,2);
        [ut1, tut1, jdut1, utc, tai, tt, ttt, jdtt, tdb, ttdb, jdtdb ] ...
        = convtime ( TIME_ACQ.year, TIME_ACQ.month, TIME_ACQ.day, ...
        TIME_ACQ.hour, TIME_ACQ.minute, …
        (TIME_ACQ.second + TIME_ACQ.usec/1e6),0,Dut1utc, 0);
        [gmst, dgmst] = gstime(jdut1);
        %
        GMST = gmst;
        GMSTdot = dgmst;
        clear ut1 tut1 jdut1 utc tai tt ttt jdtt tdb ttdb jdtdb gmst dgmst;
        %
        % acquisition time state vector transform to ECI
```

```
                %
                [pos_eci,vel_eci,A,Ap]=ecef2eci(SATPOS_ACQ(ii,:)',SATVEL_ACQ(ii,:)',…
                GMST,GMSTdot);
                %
                SATPOS_ECI=pos_eci';
                SATVEL_ECI=vel_eci';
                clear pos_eci vel_eci A Ap;
                %
                 % scene centre ECEF vector transform to ECI …
                [pos_eci,vel_eci,A,Ap]=ecef2eci(ctr_xyz(ii,:)',[0;0;0],GMST,GMSTdot);
                CTR_ECI = pos_eci';
                clear pos_eci vel_eci A Ap;
                %
                % satellite specific angular momentum vector
                AngMomentumVec=cross(SATPOS_ECI,SATVEL_ECI);
                %
                % line of nodes vector … points to ascending node
                Khat=[0,0,1];
                NodeVec_ECI=cross(Khat,AngMomentumVec);
                %
                % central orbit angle is dot product of node vector with sat. pos'n vector
                 cosAngle = dot(NodeVec_ECI,SATPOS_ECI) …
                /(absVec(NodeVec_ECI)*absVec(SATPOS_ECI));
                %
                OrbitCentralAngle(ii) = acos(cosAngle);
                %
                % correct southern hemisphere scenes for the fact that
                % acos is from 0 to pi
                if(SatPosn(ii,3)<0)
                OrbitCentralAngle(ii) = 2*pi-OrbitCentralAngle(ii)
        end
% end of loop over scenes
end
%
% identify good points …
GoodPoint_ID = find(success==1);
```

A.2 Metadata and orbit data preparation

The Matlab script that executes the data preparation works on the assumption that the data described in Section 4.1 Table 4 and Section 4.2 table 5 have been loaded into Matlab for analysis. The script described in Annex A.1 performs the required operations. The code in this annex executes three functions:

1. It selects metadata that have been deemed suitable for analysis in previous preparation sets and that correspond to a right-looking radar geometry. Although left looking cases are equally valid, they are sparse enough that they were not included here.
2. Transposes input vectors to place individual samples in separate columns.

3. It sequences the data in ascending order of the orbit central angle. This function is not necessary for the modeling activity but it does put the data into a form suitable for some subsequent analysis. The code can be easily adapted to order the data according to any other monotonic variable in the metadata set.

The code that is presented in this section is described as an internally commented Matlab script. If it is available in .txt or .wrd formats, it can be run as is by pasting it into Matlab. It can be easily modified to create Matlab.m function files if desired.

Author: Chuck Livingstone DRDC Ottawa, November 2011.

```
% The software contained in this section consists of three Matlab scripts that execute data
%preparation calculations needed to run Doppler model calculations for MDA data acquisition
%metadata records.
% This analysis assumes that the data preparation algorithm described in Annex A.1 has been run.
%
% Author: Chuck Livingstone, January 2012
%
% Inputs
        % Input data vectors from Table 4 of Section 4.1 and Table 5 of Section 4.2
                %GoodPoint_ID,
                %antennaPointing,
                %imageCornerLat1,
                %imageCornerLat2,
                %OrbitAngleNode_ECI,
                %ElevAngle,
                %DopplerCentroid,
                %SlantRange,
                %geodeticTerrainHeight,
                %SATPOS(K,:),
                %SATVEK(K,:),
                %TARGPOS(K,:),
                %yawMeasurement,
                %pitchMeasurement.
% Outputs
        %Sorted and scaled
                % Orbit internal angle, mAlph,
                % Scene center latitude, mLat,
                % Radar illumination angle, mB,
                % Measured scene Doppler centroid, mDC,
                % Radar range to the scene center, mR,
                % Scene center height above geoid, mH,
                % Measured yaw angle, mY,
                % Measured pitch angle, mP,
                % Satellite position vector, mRS,
                % ECEF satellite velocity vector, mVS,
                % Scene center position vector, mRe.
%User-defined functions called
        % rad.m
```

%Data selection script

```
%
%Define a pointer to select valid, right-looking data from the metadata set.
% Set the pointer vector for data points that have valid orbit state vectors
KV=GoodPoint_ID;
% Select the right-looking valid data. Set a pointer vector
%Set a counter
j=0;
for i=1:length(KV)
        P=char(antennaPointing(KV(i)));
        % test for Right looking
        if length(P)==5
                j=j+1;
                % Define the pointer vector
                K(j)=KV(i);
        end
end
%
% The output of this script is the pointer data vector K(j)
```

% Parameter extraction and scaling script

```
% Extract and rename valid right looking parameters. Scale their units.
% Rename the parameters to fit the calculations but retain them as column vectors
%
        %Scene center geodetic latitude in radians
        Latm=rad((imageCornerLat1(K)+imageCornerLat2(K))/2);
        %orbit internal angle in radians
        Alphm=OrbitAngle_Node_ECI(K);
        % Radar illumination angle in radians
        Bm=ElevAngle(K);
        %MDA Doppler centroid estimate
        DC=DopplerCentroid(K);
        % radar range to the scene center in km
        Rm=SlantRange(K)/1000;
        % Terrain elevation in km
        H=geodeticTerrainHeight(K)/1000;
        %Satellite position vector in ECEF in km
        RSm=SATPOS(K,:);
        %Satellite velocity vec tor in ECEF in km
        VSm=SATVEL(K,:);
        % Target Position vector (X,Y,Z)
        Rem=TARGPOS(K,:);
        % Measured yaw angle in degrees
        Ym=yawMeasurement(K);
        % Measured pitch angle in degrees
        Pm=pitchMeasurement(K);
%
```

```
% The data vectors that are generated in this script retain the same order as was present in the
input data set
```

% Parameter re-ordering script

```
% This script reorders that data and transforms all outputs into row vectors where each data point
%is in a separate column. For multi-component data vectors, each component occupies a separate
row..
%
% Reorder the parameters by increasing earth center angle
%
% Create a matrix of the scalar parameters to be sorted
TMat=zeros(length(Alphm),8);
%
TMat=[Alphm,Latm,Bm,DC,Rm,H,Ym,Pm];
 Set up a matrix for TMat when it is sorted by ascending Alphm
SMat=zeros(length(Alphm),8);
%
% Sort the matrix in increasing order of Alphm
SMat=sortrows(TMat,1);
%
% Extract the contents of the sorted matrix as row vectors to be compatible with model
calculations and clear the working matrices.
        mAlph=SMat(:,1)';
        mLat=SMat(:,2)';
        mB=SMat(:,3)';
        mDC=SMat(:,4)';
        mR=SMat(:,5)';
        mH=SMat(:,6)';
        mY=SMat(:,7)';
        mP=SMat(:,8)';
%
clear TMat SMat
%
% Create a matrix of vector quantities to be sorted
TMat=zeros(length(Alphm),10);
TMat=[Alphm,RSm(:,1),RSm(:,2),RSm(:,3),VSm(:,1),VSm(:,2),VSm(:,3),Rem(:,1),Rem(:,2),Re
m(:,3)];
%
% Set up a matrix for TMat when it is sorted by ascending Alphm
SMat=zeros(length(Alphm),10);
%
% Sort the matrix in increasing order of Alphm
SMat=sortrows(TMat,1);
%
% Extract the contents of the sorted matrix as row vectors to be compatible with model
calculations and clear the working matrices.
        mRS=SMat(:,2:4)';
```

```
        mVS=SMat(:,5:7)';
        mRe=SMat(:,8:10)';
%
clear TMat SMat
% The output of the script is a set of data vectors that are sorted in order of increasing orbit
central angle and are compatible with the Doppler centroid model in Annex A.3
```

A.3 Doppler Centroid model validation code

The Matlab script presented in this section is the implementation of the algorithm presented in Section 4.3.1 5. to 12.h as it was used to compute results for Doppler centroid model validation studies. The data preparation steps that are included in Section 4.3.1 are described in Annex A.1.

The analysis that is executed in this script expects input data in the form that is generated by Metadata and orbit data preparation script. These data are row-oriented and have independent samples in separate columns.

The model analysis is presented as a Matlab script, which if it is available in .txt or .wrd formats can be copied and pasted into Matlab for execution. No attempt has been made to streamline to code for fast operation. If desired, the code can be restructured as one or more Matlab.m files. It can be readily adapted into a utility for generating Doppler centroid predictions for use in marine data analysis. The following code has been internally documented to simplify modification.

Author: Chuck Livingstone, DRDC Ottawa, January 2012

```
%Model calculations for MDA metadata inputs
%
% Author: Chuck Livingstone, DRDC Ottawa, January 2012
%
% Inputs
        % Orbit internal angle, mAlph
        % Scene center latitude, mLat
        % Radar illumination angle, mB
        % Measured scene Doppler centroid, mDC
        % Radar range to the scene center, mR
        % Scene center height above geoid, mH
        % Measured yaw angle, mY
        % Measured pitch angle, mP
        % Satellite position vector, mRS
        % ECEF satellite velocity vector, mVS
        % Scene center position vector, mRe
% Outputs
        % Model Doppler centroid, fD
        % Target point geocentric latitude, Lat
        % Target point geodetic latitude, Glat
        % Target point incidence angle, Ginc
% User defined functions called
        % deg.m
```

```
        % rad.m
%

% Define model tuning parameters
%
% Yaw steering coefficient in degrees
        Ya=-3.92;
        % Yaw bias in degrees
        Yb=0.01;
        % Pitch bias in degrees
        Pb=-0.02;
        % Yaw control servo lag in seconds
        ylag=10;
        % Pitch control servo lag in seconds
        plag=0.0;
        % Earth center angle increment for velocity calculations
        % number of earth center angle increments
        n=3600;
        % velocity scaling parameter
        factor=2.96;
%
% Define Constants
%
        % Satellite orbit constants
        %
        % Orbit semi-major axis in km
        Rsm=7167.07;
        % Orbit inclination in degrees
        I=98.58;
        % Orbit eccentricity
        E=0.001155;
        %Argument of perigee in degrees
        Per=89.72;
        % Velocity constant, sqrt(GM)
        Kgm=631.34816;
%
        % Earth model constants
        %
        % equatorial earth radius in km
        Req=6378.1379;
        % eccentricity
        e=0.0818191908;
        % Sidereal day in h
        Td=23.9345;
        % Earth rotation angular velocity in radians/s
        Om=2*pi/(Td*3600);
%
% Prepare metadata for model calculations
```

```
%
% calculate the orbit radius and the target-point earth radius
for i=1:length(mAlph)
        % Orbit radius in km
        Rss(i)=sqrt(mRS(:,i)'*mRS(:,i))/1000;
        % earth radius in km
        ree(i)=sqrt(mRe(:,i)'*mRe(:,i))/1000;
end
%
% Illumination angle
B=mB;
% Radar range
R=mR;
%
% Define the calculation variables for an elliptical satellite orbit
% Assume a time-of-year adjusted inertial orbit.
        %Orbit central angle (earth center) from the ascending node at the equator in radians
        alph=mAlph;
        % Orbit central angle measured from perigee (true anomaly)
        Nu=alph-rad(Per);
        % Estimate the angle between the satellite velocity vector and the satellite position vector
        normal (Flight-path angle)
        phi1=atan2(E.*sin(Nu),1+E.*cos(Nu));
        % Estimate the satellite orbital speed in km/s
        Vs=Kgm*sqrt(Rsm*(1-E^2))./(Rss.*cos(phi1));
        % Time unit as a function of satellite motion (seconds)
        dT=2*pi*Rss./(Vs.*cos(phi1));
%
% Define the model yaw steering function including control lag and bias
%
Yaw=rad(Ya).*cos(alph+2*pi.*(ylag./dT))+rad(Yb);
%
% Define the model pitch steering function with respect to the satellite position vector normal
%
Pit=2*pi.*E.*cos(Nu)./(1+E.*cos(Nu)).*plag./dT+rad(Pb)+rad(mP);
%
% Define the inertial orbit plane parameters for earth center coordinates
%
        %Orbit plane normal unit vector (column vector)
        UN=[0,sin(rad(I)),-cos(rad(I))]';
        % Earth centered satellite position unit vector
        URs=[cos(alph)',cos(rad(I)).*sin(alph)',sin(rad(I)).*sin(alph)']';
        %Satellite velocity unit vector normal to Urs (compensated for the flight path angle pitch,
        phi1)
        UVs=[-sin(alph)',cos(rad(I)).*cos(alph)',sin(rad(I)).*cos(alph)']';
        % Satellite velocity unit vector without flight-path angle pitch compensation (true unit
        vector)
        for i=1:length(alph)
```

```
                UVsp(:,i)=UVs(:,i).*cos(phi1(i))+URs(:,i).*sin(phi1(i));
                UVsp(:,i)=UVsp(:,i)./absVec(UVsp(:,i));
                %Adjust the satellite position unit vector by the pitch angle for velocity
                projection calculations
                URp(:,i)=URs(:,i).*cos(Pit(i))+UVs(:,i).*sin(Pit(i));
        end
%
% Run the model calculations for the metadata points
% Loop through the orbit central angle for an ellipsoidal earth
for i=1:length(alph)
        % Estimate the radar range unit vector
        UR(:,i)=-
        URp(:,i).*cos(B(i))+UVs(:,i).*sin(B(i)).*sin(Yaw(i))+UN(:).*sin(B(i)).*cos(Yaw(i));
        %
        % Vector normalization confirm
        UR(:,i)=UR(:,i)./absVec(UR(:,i));
        % Observation point earth radius vector
        Re(:,i)=Rss(i).*URp(:,i)+R(i).*UR(:,i);
        % Earth radius unit vector
        URe(:,i)=Re(:,i)./sqrt(Re(:,i)'*Re(:,i));
        %Target point  earth center latitude
         Lat(i)=atan2(Re(3,i),sqrt(Re(1,i).^2+Re(2,i).^2));
        %Target point  geodetic latitude
        Glat(i)=atan2(Re(3,i),sqrt(Re(1,i).^2+Re(2,i).^2)*(1-e^2));
        % Target point relative longitude
        gam(i)=atan2(Re(2,i),Re(1,i));
        % calculate the ellipsoid normal at the target point
        UNe(:,i)=[cos(Glat(i)).*cos(gam(i)),cos(Glat(i)).*sin(gam(i)),sin(Glat(i))]';
        %Geodetic incidence angle
        Ginc(i)=acos(-UR(:,i)'*UNe(:,i));
        %Model earth radius
        re(i)=sqrt(Re(:,i)'*Re(:,i));
end
%
% Calculate the residual earth surface velocity contributions and satellite vertical velocity
contributions to the Doppler centroid
%
% Estimate all velocity projections and compute the model Doppler centroid
%
% Calculate the residual earth surface velocity contributions and satellite vertical velocity
contributions to the Doppler centroid
%
for i=1:length(alph)
        % Compute the time required to move two increments of orbit central angle at constant
        range
        dt(i)=4*pi.*Rss(i)./(n.*Vs(i).*cos(phi1(i)));
        %Angle increment dalph
        dalph=pi/n;
```

```
        % Compute the orbit radius change
        dRs1(i)= Rsm*(1-E^2)./(1+E.*cos(Nu(i)+dalph))-Rsm*(1-E^2)./(1+E.*cos(Nu(i)-
        dalph));
        %
        % Satellite height variation velocity along the satellite position vector
        VRs1(i)=dRs1(i)./dt(i);
        %Satellite position unit vectors before and after the observation point
        URs1=[cos(alph(i)+dalph)',cos(rad(I)).*sin(alph(i)+dalph)',sin(rad(I)).*sin(alph(i)+dalph)
        ']';
        %
        URs2=[cos(alph(i)-dalph)',cos(rad(I)).*sin(alph(i)-dalph)',sin(rad(I)).*sin(alph(i)-
        dalph)']';
        % Latitudes before and after the observation point
        lat1= atan2(URs1(3),sqrt(URs1(1).^2+URs1(2).^2));
        lat2= atan2(URs2(3),sqrt(URs2(1).^2+URs2(2).^2));
        % Target point earth radius before and after the observation point
        re1=Req*(1-e^2)^0.5./(1-e^2.*cos(lat1).^2).^0.5;
        re2=Req*(1-e^2)^0.5./(1-e^2.*cos(lat2).^2).^0.5;
        % Earth surface height variation velocity along the earth radius vector
        VRe1(i)=(re1-re2)./dt(i);
        % Earth surface rotation velocity unit vector at the target point
        UVe(:,i)=[-Re(2,i)./sqrt(Re(1,i).^2+Re(2,i).^2),Re(1,i)./sqrt(Re(1,i).^2+Re(2,i).^2),0]';
        %
        % Estimate the range-projected satellite velocity terms in km/s
        % Satellite velocity projection along the radar range vector
        Vsr(i)=Vs(i).*cos(phi1(i)).*(UVsp(:,i)'*UR(:,i));
        % Vertical velocity projections along the radar range vector
        VH(i)=VRs1(i).*(UR(:,i)'*URp(:,i))+VRe1(i).*(UR(:,i)'*URe(:,i));
        %satellite pitch projection along the radar range vector
        %Vp(i)=Vs(i).*(URp(:,i))'*UR(:,i);
        %Vp(i)=Vs(i).*sin(Pit(i));
        Vp(i)=0;
        % Estimate the projection of the target point earth rotation velocity onto the range vector
        in km/s.
        Vre(i)= Om*sqrt(Re(1,i)^2+Re(2,i)^2).*UR(:,i)'*UVe(:,i);
end
%
% Sum the velocity components that have been projected along the range vector
% I have no idea where the bug is that requires "factor" to be ~ 2.99.
Vres=(Vsr+VH+Vp+Vre)/factor;
%
% Estimate the residual Doppler frequency (scene Doppler centroid) in Hz
% The factor of 1000 converts km/s to m/s.
        fD=2000*Vres/0.055;
%
% End of script
```

Some of the parameters that are generated in this script are not used internally but have been retained to simplify other analyses.

A.4 Data display and histogram analysis

The Matlab code in this section is not necessary for analysis but can be useful if anyone wants to replicate the results that are reported in this document for other data sets and compare the new analysis to the results in this document.

Author: Chuck Livingstone, DRDC Ottawa, January 2012

```
%
% User-defined function called
%deg.m
%
%Generate a plot of measured and modeled Doppler centroid data that corresponds to %Figure 7.
        plot(deg(Glat),mDC,'.',deg(Glat),fD,'.')
%
%Generate a plot of the difference between the measured and modeled Doppler centroids that
corresponds to Figure 8
%
% Compute servo-error statistics and use them to generate the error distribution in Figure 9
% Histogram of the servo error data
[n,x]=hist(mDC-fD,200);
%Histogram peak
maxn=max(n);
%
%Mean servo error
errmn=mean(mDC-fD);
% Servo error standard deviation
errstd=std(mDC-fD);
% Find a Gaussian function that most closely approximates the histogram using the form shown
in equation 41. For a first approximation:
y=maxn*exp(-(x-errmn).^2/(2*errstd^2));
%Adjust the constants in y to get the best fit.
% Generate the plot
figure
plot(x,n,'.',x,y,'*')
%
%Examine the fit between the histogram and the Gaussian model to generate the discrepancy plot
in Figure 11
plot(x,n-y,'.')
```

This page intentionally left blank.

Annex B Doppler centroid model for RADARSAT-2 ocean data analysis: Matlab code

The Matlab code presented in this Annex has been developed as a user package that will allow a scientist to model the expected RADARSAT-2 Doppler centroid for oceanographic measurements using any mode of RADARSAT-2 that will generate single-look complex or raw data. The algorithms implemented in the following software are described earlier in this document and have been shown to provide Doppler centroid estimates for RADARSAT-2 imaging operations that are accurate to within the control-noise limitations of the satellite's radar beam pointing hardware and software.

The Doppler centriod computed by this software is a best estimate of the systematic correction to Doppler centroids measured from RADARSAT-2 single-look complex SAR data.

The actions required to use this software and the data files that must be accessed for its operation are explained in Section 5 of this technical memorandum.

The master control function, runDopplerModel.m, executes the calculations described in Annexes B.1 and B.2.

B.1 Run the Doppler centroid model

For an analyst to run the Doppler centroid model calculations there are a number of preferred and required preparation steps.

1. The signal data files for the GMTI raw data scene or SLC SAR scene to be modelled must be loaded into a directory. For the model to work, the file, product.xml, for the scene of interest must be present.
2. It is preferred (but not required) that the precision orbit file, file_name.orb be retrieved and loaded into the same directory as product.xml. If this file is missing, product .xml will be used as the state-vector data source.
3. It is preferred (but not required) that UT1 to UTC correction file UT1-UTC_'year'to'year'.csv that includes the data acquisition date be loaded into a known directory. If this file is missing the UTC offset correction will be set to zero.
4. It is preferred (but not required) that data-acquisition-oriented image points (matrix input_targets) to be used for Doppler centroid estimation be specified in the script model_specification.m used to run the model. If this matrix is not present, the scene center will be used to generate the estimates.
5. It is required that the script model_specification.m be edited to specify parameters required to run the model.

When the script model_specification.m has been edited the command runDopplerModel will execute the model command script and will generate data structures modelDataIn and modelDataOut for use in further analysis.

B.1.1 model_specification.m

This script must be edited for each new data set to be modelled

```
%model_specification.m
%
%Specify the image points for Doppler centroid estimation. The coordinates are a range-azimuth
%pair of integers representing the data acquisition sequence (1,1 is near range, first acquired
%line). the data are input as a two column matirix with each row representing a different point.
%The line sequence can be random with respect to the image.
%

% The default condition assumes that the target coordinates have been extracted from a SLC
%image file that is delivered with the data set. If no target points have been extracted set
%Input_targets to the empty matrix [] and the software will give the Doppler centroid for the
%scene center.
input_targets=[2320,8772;1123,11000;5700,5123];
%
%Set the target point source. If Tgt ~= 0, it is assumed that the target point coordinates have been
extracted from the SLC image file. If Tgt = 0, it is assumed that the target point coordinates have
been extracted from a processed image file that is in data acquisition order.
Tgt=1;
%
%Set the data path to the product.xml and orbit_file.orb files. The data path is required. To avoid
%confusing the Matlab-Java links for reading xml files, use the full data path.
data_path='C:/media/DATAPART1_/R2-MODEX/PDS_0051229_20090518/';
%
%Set a switch, Orb, to tell the software whether the orbit data file is contained in the directory
%that holds product.xml, Orb = 1, or is in a separate directory Orb = 0. If a separate directory
%(Orb = 0) is used, the orbit data file name must contain the full path to the file. A full-path
%example is: orbit_file =
'C:/Work/Test_data/Definitive_Orbit_Files_08_10_acquisitions/07444_def.orb'
%
Orb=1;
%
%Specify the orbit data file. If the precision orbit file does not exist, enter the literal '0' and the
%state vector data will be extracted from the file product.xml.
orbit_file = '07444_def.orb';
%
%Specify the UTC correction file If there is no UTC correction file, enter the literal '0' and the
%UTC offset will be set to 0..
UT1corr_file='/path/UT1-UTC_2007to2011.csv';
%
%UTCcorr_file end date. The UTC correction file must be in a year between 2007 and this year.
UTCend= 2012;
%
```

```
%Set the satellite velocity source selection switch. sel=0 selects a theoretical satellite velocity
%and sel=1 selects an ECI value derived from satellite measurements. Use sel=0 when the state
%vectors must be extracted from the file product.xml.
sel=1;
%
```

B.1.2 runDopplerCentroidModel.m

The script runDopplerCentroidModel.m executes the model calculations

```
runDopplerCentroidModel
%runDopplerCentroidModel.m
%
%Load the model specifications into the work space
model_specification;
%
%Prepare the data for model calculations
modelDataIn=model_data_prep(data_path,orbit_file,UT1corr_file,UTCend,input_targets,Tgt);
%
%Run the model
modelDataOut=model_Doppler(modelDataIn,sel);
%
```

B.2 Source data preparation

B.2.1 Prepare model inputs: model_data_prep.m

The function, model_data_prep.m: calls the function model_Init.m to

1. access and read the RADARSAT-2 metra-data file product.xml which is associated with each RADARSAT-2 data acquisition and

2. access and read the ASCII definitive orbit file, (orbit number)_def.orb,

to provide radar and satellite orbit information for analysis. The function interpolates orbit state-vector and satellite attitude information to the times corresponding to each target data point in the input matrix, input_targets and maps these data from the ECEF (Earth-Centered, Earth_Fixed) reference frame used for RADARSAT-2 orbit information to the the orbit plane reference frame described in Section 2.3.2.

The function generates the Matlab data structure, modelDataIn, described in Section 5.2, which is used by the Doppler centroid modeling function, model_Doppler.m, to perform the output estimates.

The function is internally documented by comments. The Matlab command, <<help model_data_prep, will display the first block of comments as a help file. User-defined functions called by model_data_prep.m can be found in Annex C.

The function model_data_prep.m is called from the master control function, runDopplerModel.m.

Author: Pete Beaulne, DRDC Ottawa, February 2012

Modified by: Chuck Livingstone, DRDC Ottawa, March 2012.

```
function
modelDataIn=model_data_prep(data_path,orbit_file,UT1corr_file,UTCend,input_targets,Tgt,Orb
)
%
%The function model_data_prep.m uses the outputs of the user-edited script
%model_specification.m to generate the data structure modelDataIn that is required by the
%Doppler centroid model. model_Doppler.m to estimate the systematic component of scene
%Doppler centroids.
%
% Multiple target points in the same data acquisition are supported.
%
%Usage
        % modelDataIn=model_data_prep(data_path, orbit_file, UT1corr_file, UTCend,
        %input_targets, Tgt)
%
%Inputs
        %data_path is a literal variable that defines the path from the current Matlab work-space
        %to the directory that contains product.xml and the precision orbit data in orbit_file.
        %
        %orbit_file is a literal variable that contains the name of the definitive orbit file to be
        used.
        %UT1corr_file is a literal variable that contains the path and name of the UT1 to UTC
        correction file.
        %UTC end is an integer that defines the year following the last year in the UT1corr_file.
        %input_targets is an n x 2 matrix whose rows are the selected target points in an SLC
        %SAR image and whose columns are [range coordinate, azimuth coordinate] where the
        %range and azimuth coordinates are the pixel and line number for the target point.
        %
        %Tgt is an integer that tells the program whether the data used to generate the target
        %points was in the range-azimuth time orientation defined by product.xml (Tgt = 1) or
        %whether the source image was in data-acquisition orientation (Tgt = 0). Depending on
        %the analyst's choice, MODEX data could be in either orientation.
        %Orb is a switch that ells the software whether the precision orbit file is in the same
        %directory as product.xml, Orb = 1, or in a separate directory, Orb = 0.
%
%Outputs:
        %The function output, modelDataIn, is an indexed data structure with fields:
                % .rangeCoordinate, (the slant-range image range coordinate measured from the
                % reference corner  of the image);
                %
                % .azumuthCoordinate, (the slant-range image azimuth coordinate measured
                %from the reference corner of the image);
                %
```

```
                % .range, (the radar range in m to the target point).
                %
                % .radarPosition, (the ECI position of the radar in m translated to the origin of
                %the model coordinate system, which is the column vector [X; Y; Z]:)
                %
                % .radarVelocity, (the ECI velocity in the radar in m/s), as the column vector
                %[Vx; Vy; Vz]:)
                %
                %orbitCentralAngle, (the angle in degrees from the ascending node of the radar
                %to the measurement point),
                %
                % .PRF, (the radar pulse repetition frequency in Hz),
                %
                % .pitch, (the reported pitch angle in degrees),
                %
                % Look, (the radar look direction 'Right' or 'Left)'
%Note
        % If the target points are selected from the standard image file generated for
        %RADARSAT-2, the image reference point, (1,1) is the north west corner of the image.
        %If the target points are extracted from user-processed image data the reference point can
        %be the near slant range and the first acquired azimuth line.
%User-defined functions called
        % model_Init.m
        % interState.m
        % ecef2eci.m
        % convtime.m
        % gstime.m
%
%Author: Pete Beaulne, DRDC Ottawa, February 2012, modified by Chuck Livingstone, DRDC
Ottawa, March 2012
%

% Get needed quantities from product and orbit file
[AzTime, Range, StateVecs, AttVecs, AncData] = model_Init(data_path,orbit_file,Orb);
%
%Test the input_target matrix and use the scene center if it is empty
xx=size(input_targets);
if xx(1) == 0
        input_targets = [round(length(Range)/2),round(length(AzTime)/2)];
end

% Load the UT1 - UTC correction for the acquisition date. If the file is not named, or the time is
%outside the data boundaries, set the correction to zero.
if ~strcmp(UT1corr_file,'0') & (AzTime(1).year > 2006) & (AzTime(1).year < UTCend)
        fid = fopen(UT1corr_file, 'r');
        for kk=1:1826
                line=fgetl(fid);
                tmpxstr=regexp(line,'\,','split');
```

```
                if( str2double(char(tmpxstr(1))) == floor(AzTime(1).MJD) )
                        ut1utcdiff=str2double(char(tmpxstr(2)));
                        break;
                end
        end
        fclose(fid);
        clear fid kk line tmpxstr;
else
        ut1utcdiff = 0;
end
%
%Get the size of the input_targets matrix and scroll through the targets
 [N_targets,n] = size(input_targets);
%
% Initialize the output data structure, modelDataIn
modelDataIn=struct('rangeCoordinate',{},'azimuthCoordinate',{},'radarPosition',{},'radarVelocity
',{},'orbitCentralAngle',{},'PRF',{},'pitch',{},'Look',{},'ECEFVel',{},'ECEFPos',{});

%
% Perform the calculations for each target point.
%
for kk=1:N_targets
        target = input_targets(kk,:);
        %Reset the target point coordinates for processed signal data that are in the data
        %acquisition-time sequence as opposed to the MDA image time sequence reported by
        %product.xml.
        %
        if Tgt == 1

                % Set the product line and pixel numbers to correspond to the acquisition
                %geometry, i.e. Az sample 1 == earliest in time, Rng sample 1 == shortest range
                %Test the range sample orientation
                if(strcmp(AncData.RangeTimeOrder,'Decreasing'))
                        target(1) = length(Range) - input_targets(kk,1) + 1;
                else
                        target(1) = input_targets(kk,1);
                end
                %
                %Test the azimuth sample orientation
                if(strcmp(AncData.AzimuthTimeOrder,'Decreasing'))
                        target(2) = length(AzTime) - input_targets(kk,2) + 1;
                else
                        target(2) = input_targets(kk,2);
                end
        end
        %
        % Load range and azimuth target coordinates into the modelDataIn data structure. These
        %coordinates are the user's inputs as extracted from the image data.
```

```
        modelDataIn (kk).rangeCoordinate = input_targets(kk,1);
        modelDataIn (kk).azimuthCoordinate  = input_targets(kk,2);
        %
        %Load the slant range to each target into the data structure from the Range data vector.
        modelDataIn (kk).range = Range(target(1));
        %
%Interpolate pitch data in the AttVecs file
        %Get the attitude time data for each target
        att_time = [AttVecs(:).time];
        tatt = 86400*([att_time(:).MJD] - att_time(1).MJD);
        tint = 86400*(AzTime(target(2)).MJD - att_time(1).MJD);
        %
        %Interpolate pitch data
        int_type='spline';
        % default linear. use also nearest, linear, spline, cubic=pchip
        pitch_int = interp1(tatt,[AttVecs.pitch],tint,int_type);
        %Clear working variables
        clear att_time tatt tpint;
        %
%Interpolate state vector data
        % If there are enough state vectors, find 2 state vectors on either side of target azimuth
        %for interpolation. Otherwise, use the available state vector data. It is assumed that the
        %state vector data in product.xml spans the companion image.
        stt = [StateVecs(:).time];
        %Find the number of available state vectors
        n=length(StateVecs);
        %Find the number of state vectors preceding the target point.
        jj=find([stt(:).MJD] < AzTime(target(2)).MJD,1,'last');
        %
        %Define the data-structure State for use in interpolation. Test for the position of a target
        %point in the state vector time array in case the state vectors were extracted from
        %product.xml
        if n >= 4
                if jj == 1
                        State=StateVecs(1:4);
                elseif n-jj<2
                        State=StateVecs (n-3:n);
                else
                        State=StateVecs(jj-1:jj+2);
                end
        else
                State=StateVecs(1:n);
        end
        %
        clear stt jj n;
        %
        % Interpolate the state-vector data to the target point and make the data structure
        %interp_statevec.
```

```
        interp_statevec=interState(State,AzTime(target(2)));
        %
%Convert ECEF position and velocity data to ECI coordinates for model %calculations.
        % For conversion to inertial coordinates get UT1 times (correct %for delta from UTC)
        and calculate GMST
        [ut1, tut1, jdut1, utc, tai, tt, ttt, jdtt, tdb, ttdb, jdtdb ] ...
        = convtime ( AzTime(target(2)).year, AzTime(target(2)).month, AzTime(target(2)).day,
        AzTime(target(2)).hour, AzTime(target(2)).minute, (AzTime(target(2)).second +
        AzTime(target(2)).usec/1e6),0, ut1utcdiff, 0);
        %
        [gmst, dgmst] = gstime(jdut1);
        %
        clear ut1 tut1 jdut1 utc tai tt ttt jdtt tdb ttdb jdtdb;
        %
        % Transform ECEF position and velocity to ECI coordinates
        [pos_eci,vel_eci,A,Ap]=ecef2eci([interp_statevec.position],[interp_statevec.velocity],gm
        st,dgmst);
        %
        clear interp_statevec gmst dgmst;
        %
%Compute the orbit internal angle from the ascending node equator crossing.
        % compute the normalized satellite angular momentum vector.
        AngMomentumVec = cross(pos_eci,vel_eci);
        %
        %Compute the line of nodes vector.
        Khat=[0;0;1];
        NodeVec=cross(Khat,AngMomentumVec);
        %
        % Compute the orbit internal angle from dot product of node vector %and sat position
        vector at the data acquisition point
        cos_OrbitAngle = dot(NodeVec,pos_eci) / (absVec(NodeVec)*absVec(pos_eci));
        %
        OrbitAngle = acos(cos_OrbitAngle);
        % Correct southern hemisphere scenes for the fact that acos is %defined from 0 to pi
        if(pos_eci(3)<0)
                OrbitAngle = 2*pi-OrbitAngle;
        end
        % calculate rotation angle about Z to bring ECI X axis to ascending Node
        Ihat=[1;0;0];
        cos_NodeAngle = dot(Ihat,NodeVec) / absVec(NodeVec);
        NodeAngle = acos(cos_NodeAngle);
        % correct NodeAngle for the fact that acos is %defined from 0 to pi
        if(cos_NodeAngle < 0);
                NodeAngle = 2*pi-NodeAngle;
        end

        % rotation matrix for rotation by +NodeAngle
```

```
        Node_rotmat = [cos(NodeAngle), sin(NodeAngle) 0; -sin(NodeAngle), cos(NodeAngle),
        0; 0, 0, 1];
        %
% Finish loading the model_inputs data structure.
        %
        modelDataIn (kk).radarPosition =Node_rotmat* pos_eci;
        modelDataIn (kk).radarVelocity = Node_rotmat*vel_eci;
        modelDataIn (kk).orbitCentralAngle = (180/pi)*OrbitAngle;
        modelDataIn (kk).PRF = AncData.prf;
        %model_inputs(kk).pitch = mean([AttVecs(:).pitch]); %%% mean for now
        modelDataIn (kk).pitch = pitch_int;
        modelDataIn (kk).Look = AncData.antennaPointing;
        %
        clear Khat pos_eci vel_eci AngMomentumVec NodeVec cos_OrbitAngle OrbitAngle;
        %
% End the target-point for loop
end
%
return
%
```

B.2.2 Extract data for model calculations: model_Init.m

The Matlab function model_Init.m retrieves RADARSAT-2 parameters from the file product.xml and, if it is available, from the ASCII precision orbit file, orbit_file. The function converts data formats to Matlab-readable forms and returns them to the work space in the array [AzimuthTime,SlantRange,StateVectors, AttitudeVectors, Anc_data] where:

1. AzimuthTime is a time data-structure indexed by the transmitted radar pulses.
2. SlantRange is a data-vector with one value per radar slant range.
3. StateVectors is a data_structure with five indexed values.
4. AttitudeVectors is a data_structure containing the number of values in Product.xml
5. Anc_data is an un-indexed data-structure containing parameters extracted from product.xml.

The function allows the files product.xml and orbit_file to be in the same directory or in separate directories (if the path to the orbit file is specified with the file name. The function also assumes that the RADARSAT-2 SAR data have been processed to single-look, complex (SLC) form.

The function is internally documented by comments. The Matlab command, <<help model_Init, will display the first block of comments as a help file. User-defined functions called by model_Init.m can be found in Annex C.

The function model_Init.m is called from the function model_data_prep.m.

Author: Pete Beaulne, DRDC Ottawa, February 2012.

```
function [AzimuthTime, SlantRange, StateVectors, AttitudeVectors, anc_data] =
model_Init(data_path,orbit_file,Orb)
%
% The Matlab function model_Init.m retrieves RADARSAT-2 parameters from the file
%product.xml and, if it is available, from the ASCII precision orbit file, orbit_file. The function
%converts data formats to Matlab readable forms and returns them to the work space in the array
%[AzimuthTime,SlantRange,StateVectors, AttitudeVectors, Anc_data]
%
%Assumptions
        % The SAR data for product.xml has been processed to single-look complex format.
        % Both orbit_file andproduct.xml are in the same directory
        %Leap years are hard-wired to 2016
%
%Inputs
        %dat_path, (the path needed to reach the directory containing product.xml and Orbit_file)
        %orbit_file. (the name of the MDA precision orbit ASCII file)
        %Orb, (a switch that ells the software where the file orbit_file is. Orb = 1 says that
        orbit_file is in the same directory as product,xml. Orb = 0 says that orbit_file is in another
        directory and that the full path to that directory is included in the file name.)
%
%Outputs
        %Azimuth time. an indexed data-structure with fields:
                %.year
                %.month
                %.day
                %.doy
                %.hour
                %.minute
                %.second
                %.usec
                %.utcsec
                %.JD
                %.MJD
        %
        %SlantRange (This is a data vector with one value per range sample)
        %
        %StateVectors, an indexed data structure with fields:
                %.time (data-structure of the form shown above)
                %.position, (satelltie position column vector [X; Y; Z]
                %.velocity, (satellite velocity vector [Vx;Vy;Vz] in ECEF coordinates)
        %
        %AttitudeVectors, an indexed data structure with fields
                %.time, a data-structure with fields
                        %.year
                        %.month
                        %.day
                        %.doy
                        %.hour
```

```
                %.minute
                %.second
                %.usec
        %.attitude, a data-structure with fields
                %.roll
                %.pitch
                %.yaw
    %
    %anc_data, a data-structure with fields
        %.Nchannels, (the number of SAR data channels processed)
        %.Npulses, (the number of azimuth lines in the image)
        %.Nsamples, (the number of range samples in the image)
        %.LineSpacing, (azimuth sample spacing in m)
        %.SampleSpacing, (range sample spacing in m)
        %.AzimuthTimeOrder, (Increasing or Decreasing
        %.RangeTimeOrder, (Increasing or Decreasing)
        %.AntennaPointing, (Right or Left)
        %.passDirection, (Ascending or Descending)
        %.adcSamplingRate,
        %.prf, (pulse repetition rate per channel)
        %.MinRangeTime, (radar propagation time to the first range sample)
        %.MinRange, (slant range in m to the near image sample)
        %.Incidence_Near, (incidence angle at the near slat range)
        %.Incidence_Far, (incidence angle at the fra slat range)
        %.Line1_ZeroDop_time, (Zero Doppler time of the first line as a string variable)
        %.LineN_ZeroDop_time,, (Zero Doppler time of the last line as a string variable)
        %.ZeroDop_timeOffset, (microseconds from raw data start)
%
%Usage
    %[AzimuthTime, SlantRange, StateVectors, AttitudeVectors, anc_data] =
    %model_Init(data_path,orbit_file,Orb)
%
%User-defined functions called
    %time_str2struct.m,
    %jday.m,
    %sec2hms.m,
    %load_product_state_vectors.m,
    %load_product_attitude.m,
    %read_definitiveOrbit.m.
%
%Author: Pete Beaulne, DRDC Ottawa, February 2012

% Define constants ...
    c = 299792458;
    %Radar wavelength
    lambda = 0.055465772433;
    %Radar carrier frequency
    fc = c/lambda;
```

```
        %Month length for leap years
        dpmleap=[31,29,31,30,31,30,31,31,30,31,30,31];
        %Month length for regular years
        dpmreg=[31,28,31,30,31,30,31,31,30,31,30,31];
%
%Input data paths
product_file = [data_path,'product.xml'];
%Test for the presence of an orbit file and point to the right directory

if ~strcmp(orbit_file,'0')
        if Orb ==1
                orbit_file = [data_path,orbit_file];
        end
end
%
% open product.xml file ...
product = xmlread(product_file);
%
% Get Raw data Start time
myNode = product.getElementsByTagName('rawDataStartTime');
if(myNode.getLength > 0)
        timestr = char(myNode.item(0).getFirstChild.getData);
        anc_data.rawDataStartTime = timestr;
end
% convert time string to time structure ...
raw_start_time = time_str2struct(timestr);
%
% define days in each month for this year
if (raw_start_time.year == 2008 || raw_start_time.year == 2012 || raw_start_time.year == 2016)
        dpm=dpmleap;
else
        dpm=dpmreg;
end
%
%Build the data-structure anc_data by reading product.xml
        % use 'settableGain' parameter as proxy  for MODEX or
        % polarimetric data. Indicates number of virtual
        % channels used in the acquisition. MODEX-1: 2,
        % MODEX-2: 4, dual-pol: 2, quad pol (4);
        %
        myNode = product.getElementsByTagName('settableGain');
        anc_data.Nchannels = myNode.getLength;
        %
        % total number of lines (pulses)
        myNode = product.getElementsByTagName('numberOfLines');
        if(myNode.getLength > 0)
                anc_data.Npulses = str2double(myNode.item(0).getFirstChild.getData);
        end
```

```
%
% total number of slant range samples per line
myNode = product.getElementsByTagName('numberOfSamplesPerLine');
if(myNode.getLength > 0)
        anc_data.Nsamples = str2double(myNode.item(0).getFirstChild.getData);
end
%
% Line (Azimuth) spacing in m (== Vg / PRF for SLC)
myNode = product.getElementsByTagName('sampledLineSpacing');
if(myNode.getLength > 0)
        anc_data.LineSpacing = str2double(myNode.item(0).getFirstChild.getData);
end
%
% slant range sample spacing in metres (== (c/2)*adcSamplingRate )
myNode = product.getElementsByTagName('sampledPixelSpacing');
if(myNode.getLength > 0)
        anc_data.SampleSpacing = str2double(myNode.item(0).getFirstChild.getData);
end
%
% line (azimuth) time ordering: 'Increasing' or 'Decreasing'
% needed to convert product image to acquisition coordinates
myNode = product.getElementsByTagName('lineTimeOrdering');
if(myNode.getLength > 0)
        anc_data.AzimuthTimeOrder = char(myNode.item(0).getFirstChild.getData);
end
%
% pixel (rangesample) time ordering: 'Increasing' or 'Decreasing'
% needed to convert product image to acquisition coordinates
myNode = product.getElementsByTagName('pixelTimeOrdering');
if(myNode.getLength > 0)
        anc_data.RangeTimeOrder = char(myNode.item(0).getFirstChild.getData);
end
%
% Extract the antenna pointing direction ('Right' or 'Left')
myNode = product.getElementsByTagName('antennaPointing');
if(myNode.getLength > 0)
        anc_data.antennaPointing = char(myNode.item(0).getFirstChild.getData);
end
%
% Extract the pass direction ('Ascending' or 'Descending')
myNode = product.getElementsByTagName('passDirection');
if(myNode.getLength > 0)
        anc_data.passDirection = char(myNode.item(0).getFirstChild.getData);
end
%
% Extract range the sampling frequency
myNode = product.getElementsByTagName('adcSamplingRate');
if(myNode.getLength > 0)
```

```
            anc_data.adcSamplingRate = str2double(myNode.item(0).getFirstChild.getData);
        end

        % get the PRF - the reported transmit PRF needs to be scaled to the receive PRF per
        %channel. For 4 channels, divide the prf by 2
        %
        myNode = product.getElementsByTagName('pulseRepetitionFrequency');
        if(myNode.getLength > 0)
            if anc_data.Nchannels == 4
                anc_data.prf = str2double(myNode.item(0).getFirstChild.getData)/2;
            else
                anc_data.prf = str2double(myNode.item(0).getFirstChild.getData);
            end
        end
        %
        % calculate slant range vector to each sample to
        % convert pixel (range) image coordinate to slant range
        %
        % Extract the near range time  (or near range in m)
        myNode = product.getElementsByTagName('slantRangeTimeToFirstRangeSample');
        if(myNode.getLength > 0)
            anc_data.MinRangeTime = str2double(myNode.item(0).getFirstChild.getData);
        end
        %
        myNode = product.getElementsByTagName('slantRangeNearEdge');
        if(myNode.getLength > 0)
            anc_data.MinRange = str2double(myNode.item(0).getFirstChild.getData);
        end

% Build a slant-rangedata-vector containing each range sample
        SlantRange = (anc_data.MinRangeTime + (0:anc_data.Nsamples-
        1)/anc_data.adcSamplingRate)*c/2;
        %
        myNode = product.getElementsByTagName('incidenceAngleNearRange');
        if(myNode.getLength > 0)
            anc_data.Incidence_Near = str2double(myNode.item(0).getFirstChild.getData);
        end
        %
        myNode = product.getElementsByTagName('incidenceAngleFarRange');
        if(myNode.getLength > 0)
            anc_data.Incidence_Far = str2double(myNode.item(0).getFirstChild.getData);
        end
        %
        % Image line 1 zero Doppler time
        myNode = product.getElementsByTagName('zeroDopplerTimeFirstLine');
        if(myNode.getLength > 0)
            timestr = char(myNode.item(0).getFirstChild.getData);
            anc_data.Line1_ZeroDop_time = timestr;
```

```
        end
        Line1_ZeroDop_time = time_str2struct(timestr);
        %
        % Image line N zero Doppler time
        myNode = product.getElementsByTagName('zeroDopplerTimeLastLine');
        if(myNode.getLength > 0)
                timestr = char(myNode.item(0).getFirstChild.getData);
                anc_data.LineN_ZeroDop_time = timestr;
        end
        %LineN_ZeroDop_time = time_str2struct(timestr);
        %
        % zero doppler time offset
        anc_data.ZeroDop_timeOffset = Line1_ZeroDop_time.utcsec - raw_start_time.utcsec;
%
% Build a data-structure of SLC azimuth times indexed by radar pulse number
% *** NEEDS MODS FOR OTHER SAR formats ***
%
        aztimes=raw_start_time.utcsec+(0:anc_data.Npulses-1)/anc_data.prf;
        %
        for ii = 1:anc_data.Npulses
                AzimuthTime(ii).year = raw_start_time.year;
                AzimuthTime(ii).month = raw_start_time.month;
                AzimuthTime(ii).day = raw_start_time.day;
                AzimuthTime(ii).doy = raw_start_time.doy;
                %
                [h,m,s] = sec2hms(aztimes(ii));
                %
                AzimuthTime(ii).hour = h; clear h;
                AzimuthTime(ii).minute = m; clear m;
                 AzimuthTime(ii).second = floor(s);
                AzimuthTime(ii).usec = 1e6*(s-floor(s)); clear s;
                AzimuthTime(ii).utcsec = aztimes(ii);
        end
        %
        % fix AzimuthTime structure for date change: sec2hms
        % does not change date if input seconds > 86400

        idd=find([AzimuthTime(:).hour]>=24);
        %
        if(~isempty(idd))
                for kk=1:length(idd)
                        % Fix hour and UTC seconds in the day
                        AzimuthTime(idd(kk)).hour = AzimuthTime(idd(kk)).hour - 24;
                        AzimuthTime(idd(kk)).utcsec = AzimuthTime(idd(kk)).utcsec - 86400;

                        if(AzimuthTime(idd(kk)).day == dpm(AzimuthTime(idd(kk)).month))
                                % if last day of the month, set new day to 1,
                                % and increment day of year
```

```
                                AzimuthTime(idd(kk)).day = 1;
                                AzimuthTime(idd(kk)).doy = AzimuthTime(idd(kk)).doy + 1;
                                %
                                 if(AzimuthTime(idd(kk)).month == 12)
                                        % if last month of year, set new month
                                        % and new doy to 1
                                        AzimuthTime(idd(kk)).month = 1;
                                        AzimuthTime(idd(kk)).doy = 1;
                                        %
                                        AzimuthTime(idd(kk)).year =
                                        AzimuthTime(idd(kk)).year + 1;
                                        %
                                else
                                % if not last month of the year, increment month
                                        %
                                        AzimuthTime(idd(kk)).month =
                                        AzimuthTime(idd(kk)).month + 1;
                                        %
                                end
                                %
                        else
                                %if not last day of month, increment day and day of year
                                AzimuthTime(idd(kk)).day = AzimuthTime(idd(kk)).day + 1;
                                AzimuthTime(idd(kk)).doy = AzimuthTime(idd(kk)).doy + 1;
                        end
                end
        end
        %
        % add regular and modified julian day to Aztime structure
        for ii = 1:anc_data.Npulses
                %
                AzimuthTime(ii).JD = jday(AzimuthTime(ii).year, AzimuthTime(ii).month,
                AzimuthTime(ii).day, AzimuthTime(ii).hour, AzimuthTime(ii).minute,
                AzimuthTime(ii).second+AzimuthTime(ii).usec/1e6); AzimuthTime(ii).MJD =
                AzimuthTime(ii).JD - 2400000.5;
        end
%
% If it is specified, open the definitive orbit file, and trim state vector set to the scene.
% Keep two vectors before start of the acquisition and two vectors after the end of  the
acquisition
% if the orbit_file is not specified, read the data from the product.xml file.
%
%Read the precision orbit file if it exists as a non-zero value
if ~strcmp(orbit_file,'0')
        StateVectors=read_DefinitiveOrbit(orbit_file);
        stt=[StateVectors(:).time];
        %
```

```
        subid = find([stt(:).MJD] > AzimuthTime(1).MJD & [stt.MJD] <
        AzimuthTime(end).MJD);
        %
        StateVectors = StateVectors(subid(1)-2:subid(end)+2);
else
        StateVectors=load_product_state_vectors(product_file);
end
%
% load attitude from product file
AttitudeVectors = load_product_attitude(product_file);
%
% The commented code refers to a currently unidentified file and a currently undeveloped
function
% if(exist('attitude_file','var'))
        %Att=load_r2_attitude(attitude_file);
% else
        %Att = load_product_attitude(product_file);
% end
%
% end of function model_Init.m
end
```

B.3 Doppler centroid model: model_Doppler.m

The Matlab function model_Doppler.m accepts data from the data source preparation modules discussed in Annexes B.1 and B.2 and uses the model described in this document to estimate the Doppler centroid for each image point defined in its input data structure.

The function is internally documented by comments. The Matlab command, <<help model_Doppler, will display the first block of comments as a help file. User-defined functions called by model_Doppler.m can be found in Annex C.

The function, model_Doppler.m, is called from the master control function, runDopplerModel.m.

```
function modelDataOut=model_Doppler(modelDataIn,sel)
%
% The function model_Doppler(modelDataIn,sel).m accepts the data structure “modelDataIn”
%that contains the radar parameter and observation geometry information needed to run the
%Doppler centroid prediction model described in “Modelling.RADARSAT-2 Doppler centroids
%for marine applications”, Chuck Livingstone and Pierre Beaulne, DRDC TM 2012_XXX. The
%function outputs the indexed data structure modelDataOut which contains model estimates of
%the RADARSAT-2 Doppler centroid and useful, associated, information at each input point.
%
% Assumptions:
        %The earth is modeled as a WGS 84 ellipsoid.
        %The target surface is assumed to be the ocean surface.
        %The target point height above ellipsoid is assumed to be zero m.
        %RADARSAT-2 is assumed to be the data source.
```

```
%
% Inputs
        % modelDataIn, is an indexed data structure with fields:
                % .targetPoint, which has sub-fields
                        % .rangeCoordinate, (the slant-range image range coordinate measured
                        from the image reference point);
                        % .azumuthCoordinate, (the slant-range image azimuth coordinate
                        %measured from the image reference point);
                %
                % .range, (the radar range in m to the target point).
                %
                % .radarPosition, (the ECI position of the radar in m translated to the origin of
                %the model coordinate system, which is the column vector [X; Y; Z]:)
                %
                % .radarVelocity, (the ECI velocity in the radar in m/s), as the column vector
                %[Vx; Vy; Vz]:)
                %
                %orbitCentralAngle, (the angle in degrees from the ascending node of the radar
                %to the measurement point),
                %
                % .PRF, (the radar pulse repetition frequency in Hz),
                %
                % .pitch, (the reported pitch angle in degrees),
                %
                % Look, (the radar look direction ‘right’ or ‘left)’
        %
        %sel is a selection parameter which has default value zero (use theoretical satellite
        %velocity estimates). A choice of sel = 1 (or any positive value) selects the input velocity
        %state vector. If precision orbit data is used to generate state vectors, sel=1 is
        %recommended.
%
% Multiple target points in the same data acquisition are supported.
%
%.Outputs
        % modelDataOut, is a data structure with fields:
                % .rangeCoordinate, (the slant-range image range coordinate measured from the
                near slant range of the image);
                % .azumuthCoordinate, (the slant-range image azimuth coordinate measured
                from the start of the acquired data set);
                % .Glat, (the geodetic latitude of the target point),
                % .Lat, (the geocentric latitude of the target point),
                % .Re, (the target point earth radius in m),
                % .Ginc, (the ellipsoid incidence angle at the target point in degrees),
                % Yaw, (the model yaw angle in degrees),
                % .Pit, (the model pitch angle in degrees),
                % .Doppler, (the model Doppler centroid in Hz).
%
%Usage:
```

```
        % modelDataOut=model_Doppler(modelDataIn,sel)
%
% User-defined functions called are:
        % rad.m,
        % deg.m,
        % absVec.m.
%
%Model tuning parameters are hard-wired in the code.
%
% Chuck Livingstone, DRDC Ottawa, January 2012
%

% Extract input parameters for model calculations from each modelDataIn field
% Get the length of the data structure modelDatadataIn
Mt=length(modelDataIn);
        for j=1:Mt
                % radar range in km
                R(j)=modelDataIn(j).range/1000;
                % Satellite position vectors in km
                RS(j,:)= modelDataIn (j).radarPosition'./1000;
                % Satellite velocity vectors in km/s
                VS(j,:)= modelDataIn (j).radarVelocity'./1000;
                % Orbit central angle to the target point in degrees
                alph(j)= modelDataIn (j).orbitCentralAngle;
                % Pulse repetition frequency in Hz
                prf(j)= modelDataIn (j).PRF;
                % Reported pitch angle in degrees
                pit(j)= modelDataIn (j).pitch;
                % Radar look direction (litteral)
                look(j,:)= modelDataIn (j).Look;
                %
% Convert parameters to working variables
                % Satellite height
                Rss(j)=absVec(RS(j,:));
                % Satellite speed
                Vsm(j)=absVec(VS(j,:));
%
%Look direction switch
                if size(look(j,2))==4
                        dir=-1;
                else
                        dir=1;
                end
        end
%
% Calculation index
M=length(R);
%
```

```
% Define model tuning parameters
%
        % Yaw steering coefficient in degrees
        Ya=-3.92;
        % Yaw bias in degrees
        Yb=0.01;
        % Pitch bias in degrees
        Pb=-0.02;
        % Target range in km
        Ra=1000;
        % Yaw control servo lag in seconds
        ylag=10;
        % Pitch control servo lag in seconds
        plag=0.0;
        % Velocity scale factor
        factor=2.96;
%
% Define Constants
%
        % Satellite orbit constants
        %
                % Orbit semi-major axis in km
                Rsm=7167.07;
                % Orbit inclination in degrees
                I=98.58;
                % Orbit eccentricity
                E=0.001155;
                %Argument of perigee in degrees
                Per=89.72;
                % Velocity constant, sqrt(GM)
                Kgm=631.34816;
        %
        % Earth model constants
        %
                % equatorial earth radius in km
                Req=6378.1379;
                % eccentricity
                e=0.0818191908;
                % Sidereal day in h
                Td=23.9345;
                % Earth rotation angular velocity in radians/s
                Om=2*pi/(Td*3600);
        %
% Define the calculation variables for an elliptical satellite orbit. (Theoretical orbit model)
%
        %Orbit central angle (earth center) from the ascending node at the equator in radians
        alph=rad(alph);
        % Orbit central angle measured from perigee (true anomaly)
```

```
        Nu=alph-rad(Per);
        % Estimate the angle between the satellite velocity vector and the satellite position vector
        normal (Flight-path angle)
        % Eliptical orbit radius as a function of the true anomaly
        for j=1:length(alph)
                Rss(j)=Rsm*(1-E^2)/(1+E*cos(Nu(j)));
        end
        phi1=atan2(E.*sin(Nu),1+E.*cos(Nu));
        % Estimate the satellite orbital speed in km/s
        Vs=Kgm*sqrt(Rsm*(1-E^2))./(Rss.*cos(phi1));
        % Time unit as a function of satellite motion (seconds) for lag calculation
        if sel==1
                Vs=Vsm;
        end
        dT=2*pi*Rss./(Vs.*cos(phi1));
%
% Define the model yaw steering function including control lag and bias
%
Yaw=rad(Ya).*cos(alph+2*pi.*(ylag./dT))+rad(Yb);
%
% Define the model pitch steering function with respect to the satellite position vector normal
%
Pit=2*pi.*E.*cos(Nu)./(1+E.*cos(Nu)).*plag./dT+rad(Pb+pit);
%
% Define the inertial orbit plane parameters for earth center coordinates
        %
        %Orbit plane normal unit vector (column vector)
        UN=[0,sin(rad(I)),-cos(rad(I))]';
        % Earth centered satellite position unit vector
        URs=[cos(alph)',cos(rad(I)).*sin(alph)',sin(rad(I)).*sin(alph)']';
        %Satellite velocity unit vector normal to Urs (compensated for the flight path angle pitch,
        phi1)
        UVs=[-sin(alph)',cos(rad(I)).*cos(alph)',sin(rad(I)).*cos(alph)']';
        % Satellite velocity unit vector without flight-path angle pitch compensation (true unit
        vector)
        for i=1:length(alph)
                UVsp(:,i)=UVs(:,i)./cos(phi1(i))+URs(:,i).*sin(phi1(i));
                %Adjust the satellite position unit vector by the pitch angle for velocity
                projection calculations
                URp(:,i)=URs(:,i).*cos(Pit(i))+UVs(:,i).*sin(Pit(i));
                %Normalize the vectors to unit magnitude
                UVs(:,i)=UVs(:,i)/absVec(UVs(:,i));
                URp(:,i)=URp(:,i)/absVec(URp(:,i));

        end
        %
% Extract satellite motion from measured parameters
%
```

```
% Select modeled or measured velocity
if sel>0
        sel=1;
        % Measured range unit vector
        URs=[RS(1,:)./Rss;RS(2,:)./Rss;RS(3,:)./Rss]';
        %Measured velocity unit vector without flight angle compensation
        UVsp=[VS(1,:)./Vsm;VS(2,:)./Vsm;VS(3,:)./Vsm]';
        % Flight-angle compensated satellite velocity unit vector
        for i=1:M
                UVs(:,i)=UVsp(:,i).*cos(phi1(j)+rad(pit(j)))-URs(:,i).*sin(phi1(j)+rad(pit(j)));
                %Adjust the satellite position unit vector by the pitch angle for velocity
                projection calculations
                URp(:,i)=URs(:,i).*cos(Pit(i))+UVs(:,i).*sin(Pit(i));
                %Normalize the vectors to unit magnitude
                UVs(:,i)=UVs(:,i)./absVec(UVs(:,i));
                URp(:,i)=URp(:,i)./absVec(URp(:,i));
        end
end
%
% Set initial parameter values for geometry calculations
for i=1:M
        % Use the sub-satellite latitude for an initial earth radius estimate
        Lat(i)=atan2(RS(3,i),sqrt(RS(1,i).^2+RS(2,i).^2));
        % Estimate the sub-satellite earth radius
        re(i)=Req*(1-E^2)^0.5./(1-E^2.*cos(Lat(i))^2).^.5;
        % Make a first estimate of the satellite illumination angle
        B(i)=acos((Rss(i)^2+R(i)^2-re(i)^2)./(2*R(i).*Rss(i)));
end
%
%Iterate geometry calculations to converge on the target points.
for j=1:2
        % Compute values for the target points
        for i=1:M
                % Estimate the radar range unit vector, compensating for look direction
                UR(:,i)=-
                URp(:,i).*cos(B(i))+UVs(:,i).*sin(B(i)).*sin(dir*Yaw(i))+UN(:).*sin(B(i)).*cos(
                Yaw(i))*dir;
                %Normalize UR
                UR(:,i)=UR(:,i)./absVec(UR(:,i));
                % Observation point earth radius vector from satellite position and range vectors
                Re(:,i)=Rss(i).*URp(:,i)+R(i).*UR(:,i);
                % Earth radius unit vector
                URe(:,i)=Re(:,i)./sqrt(Re(:,i)'*Re(:,i));
                %Target point  earth center latitude
                 Lat(i)=atan2(Re(3,i),sqrt(Re(1,i).^2+Re(2,i).^2));
                %Target point geodetic latitude
                Glat(i)=atan2(Re(3,i),sqrt(Re(1,i).^2+Re(2,i).^2)*(1-e^2));
                % Target point relative longitude
```

```
                gam(i)=atan2(Re(2,i),Re(1,i));
                % calculate the ellipsoid normal at the target point
                UNe(:,i)=[cos(Glat(i)).*cos(gam(i)),cos(Glat(i)).*sin(gam(i)),sin(Glat(i))]';
                %Geodetic incidence angle
                Ginc(i)=acos(-UR(:,i)'*UNe(:,i));
                %Model earth radius
                re(i)=sqrt(Re(:,i)'*Re(:,i));
                %Target illumination angle
                B(i)=acos(abs(dot(URp(:,i),UR(:,i))));
        end
end
%
% Calculate the residual earth surface velocity contributions and satellite vertical velocity
contributions to the Doppler centroid
%
% Set up calculation parameters
% time increment
dt=2./prf;
% Orbit central angle increment
dalph=Vs./(prf.*Rss);
%
% Estimate all velocity projections and compute the model Doppler centroid for each target point
for i=1:M
        % Compute the orbit radius change
        dRs1(i)= Rsm*(1-E^2)./(1+E.*cos(Nu(i)+dalph(i)))-Rsm*(1-E^2)./(1+E.*cos(Nu(i)-
        dalph(i)));
        % Satellite height variation velocity along the satellite position vector
        VRs1(i)=dRs1(i)./dt(i);
        %Theoretical satellite position unit vectors after and before the observation point
        URs1=[cos(alph(i)+dalph)',cos(rad(I)).*sin(alph(i)+dalph)',sin(rad(I)).*sin(alph(i)+dalph)
        ']';
        URs2=[cos(alph(i)-dalph)',cos(rad(I)).*sin(alph(i)-dalph)',sin(rad(I)).*sin(alph(i)-
        dalph)']';
        % Sub-satellite latitudes after and before the observation point
        lat1= atan2(URs1(3),sqrt(URs1(1).^2+URs1(2).^2));
        lat2= atan2(URs2(3),sqrt(URs2(1).^2+URs2(2).^2));
        % Theoetical sub-satellite point earth radius after and before the observation point
        re1=Req*(1-e^2)^0.5./(1-e^2.*cos(lat1).^2).^0.5;
        re2=Req*(1-e^2)^0.5./(1-e^2.*cos(lat2).^2).^0.5;
        % Earth surface height variation velocity along the earth radius vector
        VRe1(i)=(re1-re2)./dt(i);
        % Earth surface rotation velocity unit vector at the target point
        UVe(:,i)=[-Re(2,i)./sqrt(Re(1,i).^2+Re(2,i).^2),Re(1,i)./sqrt(Re(1,i).^2+Re(2,i).^2),0]';
        %
        % Estimate the range-projected satellite velocity terms in km/s
        %
        % Satellite velocity projection along the radar range vector
        Vsr(i)=Vs(i).*cos(phi1(i)).*(UVsp(:,i)'*UR(:,i));
```

```
        % Vertical velocity projections along the radar range vector
        VH(i)=VRs1(i).*(UR(:,i)'*URp(:,i))+VRe1(i).*(UR(:,i)'*URe(:,i));
        %satellite pitch projection along the radar range vector
        Vp(i)=Vs(i).*sin(Pit(i));
        % Estimate the projection of the target point earth rotation velocity onto the range vector
        in km/s.
        Vre(i)= Om*sqrt(Re(1,i)^2+Re(2,i)^2).*UR(:,i)'*UVe(:,i).*dir;
end
%
% Sum the velocity components that have been projected along the range vector, convert to m/s
Vres=(Vsr+VH+Vp+Vre)*1000/factor;
%
% Estimate the residual Doppler frequency (scene Doppler centroid) in Hz
fD=2*Vres/0.055;
%
% Load the output data structure. Retain the target point coordinates to simplify data registration
for i=1:M
        modelDataOut(i).rangeCoordinate= modelDataIn (i).rangeCoordinate;
        modelDataOut(i).azimuthCoordinate= modelDataIn (i).azimuthCoordinate;
        modelDataOut(i).Glat=deg(Glat(i));
        modelDataOut(i).Lat=deg(Lat(i));
        modelDataOut(i).Re=re(i);
        modelDataOut(i).Ginc=deg(Ginc(i));
        modelDataOut(i).Yaw=deg(Yaw(i));
        modelDataOut(i).Pit=deg(Pit(i));
        modelDataOut(i).Doppler=fD(i);
end
%
% End of function
return
```

Annex C Utility function library

This Annex contains all user-defined Matlab utility functions that are called by the Matlab code in Annexes A and B. This function library must be within or linked to the active Matlab workspace to run the model calculations in this report.

The functions in this library have been developed by various authors and span several generations of Matlab. All functions run on Matlab version 7.8.0.

The functions are internally documented and have help blocks that can be accessed from Matlab by entering the command line, help function_name. The help request will output the first contiguous block of comments in the function code..

C.1 absVec.m

The function absVec.m computes the magnitude of a arbitrary, n dimensional vector, a, that can be in either row or column format.

Author: Shen Chiu, DRDC Ottawa, modified by Pete Beaulne, DRDC Ottawa September 2011

```
function b = absVec(a)
%
% The function absVec.m computes the magnitude of a arbitrary, n dimensional vector, a, that
%can be in either row or column format.
%
%Author
        %Shen Chiu and Pete Beaulne, DRDC Ottawa
%
%Input
        % a row or column vector A
%
%Output
        %the scalar magnitude of A
%
%User-defined functions called
        %None
%
%Usage
        %a = absVec(A);
%

% Determine the vector orientation and convert to a column vector
sz=size(a);
% if row vectors
if( sz(2) < sz(1) )
        a = a';
end
```

```
% calculate the vector magnitude
b = sqrt(dot(a,a));
% Return the magnitude in the same orientation as the source vector
if( sz(2) < sz(1) )
        b = b';
end
%
```

C.2 convtime.m

This function, convtime.m, converts universal time data in: UTC, UT1 or local-time form into a large number of time format conventions.

Author: David Vallado, 4 June 2002, Revisions: David Vallado, 8, October 2002

Reference: Vallado, D.A., Fundamentals of Astrodynamics and Applications, Space technology Library, Microcosm Press 2001, ISBN 1-881883-12-4

```
function [ut1, tut1, jdut1, utc, tai, tt, ttt, jdtt, tdb, ttdb, jdtdb ] ... = convtime ( year, mon, day, hr,
min, sec, timezone, dut1, dat );
%
% This function finds the time parameters and Julian century values for inputs in UTC or UT1
form.
% Numerous outputs are found as shown in the local variables because calculations are in UTC.
%  You must include timezone if you enter a local time, otherwise it should be zero.
%
%  author        : david vallado                719-573-2600    4 jun 2002
%
%  revisions
%        vallado     -      fix documentation for dut1                          8 oct 2002
%        Livingstone –     replace modulo with mod function                    29 Feb, 2012
%
% Input parameters
%
        %inputs          description                     range / units
        %year            year                            1900 .. 2100
        %mon         -    month                           1 .. 12
        %day         -    day                             1 .. 28,29,30,31
        %hr          -    universal time hour             0 .. 23
        %min         -    universal time min              0 .. 59
        %sec         -    universal time sec (utc)        0.0  .. 59.999
        %timezone        offset to utc from local site   0 .. 23 hr
        %dut1        -    delta of ut1 - utc                      sec
        %dat         -    delta of utc - tai              sec
%
%  Output parameters      :
        %ut1         -    universal time                  sec
        %tut1        -    julian centuries of ut1
```

```
        %jdut1      -    julian date of ut1                 days from 4713 bc
        %utc        -    coordinated universal time         sec
        %tai        -    atomic time                        sec
        %tdt        -    terrestrial dynamical time         sec
        %ttdt       -    julian centuries of tdt
        %jdtdt      -    julian date of tdt                 days from 4713 bc
        %tdb        -    terrestrial barycentric time       sec
        %ttdb       -    julian centuries of tdb
        %jdtdb      -    julian     date of tdb             days from 4713 bc
%
%Local Parameters       :
        %hrtemp     -    temporary hours                    hr
        %mintemp         temporary minutes                  min
        %sectemp    -    temporary seconds                  sec
        %localhr    -    difference to local time           hr
        %jd         -    julian date of request             days from 4713 bc
        %me         -    mean anomaly of the earth          rad
%
%  coupling     :
%    hms_2_sec   -        conversion between hr-min-sec .and. seconds
%    jday        -        find the julian date
%


      deg2rad = pi/180.0;

      % ------------------------  implementation   ------------------
      jd =  jday( year,mon,day,0,0,0.0 );
      mjd  = jd - 2400000.5;
      mfme = hr*60.0 + min + sec/60.0;

      % ------------------ start if ( ut1 is known ------------------
      localhr= timezone + hr;

      utc = hms2sec( localhr,min,sec );

      ut1= utc + dut1;
      [hrtemp,mintemp,sectemp] = sec2hms(  ut1 );
      jdut1 = jday( year,mon,day, hrtemp, mintemp, sectemp );
      tut1= (jdut1 - 2451545.0  )/ 36525.0;

      tai= utc + dat;

      tt= tai + 32.184;   % sec
      [hrtemp,mintemp,sectemp] = sec2hms( tt );
      jdtt = jday( year,mon,day, hrtemp, mintemp, sectemp);
      ttt= (jdtt - 2451545.0  )/ 36525.0;
```

```
    me= 357.5277233  + 35999.05034 *ttt;
    me= rad(mod( me,360.0  ));
    tdb= tt + 0.001658  * sin(me) + 0.00001385 *sin(2.0 *me);
    [hrtemp,mintemp,sectemp] = sec2hms( tdb );
    jdtdb = jday( year,mon,day, hrtemp, mintemp, sectemp );
    ttdb= (jdtdb - 2451545.0  )/ 36525.0;

%       fprintf( 'time %14.6f%14.6f%9.6f%18.5f\n',utc,ut1,ttt,jdut1 );

return
```

C.3 deg.m

The function deg.m accepts an angle or an array of angles in radians and outputs the angles in degrees.

Author: Chuck Livingstone, CCRS, 1990

```
function d=deg(t)
% deg returns angles in degrees for (vector) inputs in radians
%Author Chuck Livingstone, CCRS, 1990
%
%Input:
        %angle, t, in radians (t can be an array of angles
%
%Output
        % angle (or array of angles), d, in degrees
%
%Usage:
        % d=deg(t)
%
%User-defined functions called :
        %None
%
d=180./pi.*t;
```

C.4 doy2date.m

The function doy2date.m generates month and day as integers from inputs year and day of year. This function is valid up to 2024 before the hard-wired leap year condition needs to be changed.

Author: Pete Beaulne, DRDC Ottawa, Jamuary 2012

```
function [month,day] = doy2date(doy,year)
%
%  doy2date.m
%  Author: Pete Beaulne, Jan 2012
%
```

```
%  Usage-
        %[month,day] = doy2date(doy,year)
%
%  Input- year:
        %integer year
        %doy:   integer day of the year
%
%  Output-
        %month: integer month
        %day:   integer day
%
%  User functions called –
        %NONE
%
% doy of 1st of every month for a leap year
dleap=[1,32,61,92,122,153,183,214,245,275,306,336];
% doy of 1st of every month for a normal year
dreg=[1,32,60,91,121,152,182,213,244,274,305,335];
%
% convert day of the year to date
if (year == 2008 || year == 2012 || year == 2016 || year == 2020)
        d=dleap;
else
        d=dreg;
end
%
k=find(d<=doy);
month=max(k);
day=doy-d(month)+1;
%
end
```

C.5 ecef2eci.m

This function ecef2eci.m accepts satellite state-vector data in ECEF (Earth-Centered Earth-Fixed) coordinates and transforms them to ECI (Earth-Centered Inertial) coordinates with respect to the vernal equinox.

Author: Pete Beaulne DRDC Ottawa, 2005

```
function [pos_eci,vel_eci,rotmat,drotmat]=ecef2eci(pos_ecef,vel_ecef,gmst,dgmst)
%
%Function to compute eci coordinates from ecef coordinates and UTC time
%
%Usage
```

```
        %[pos_eci,vel_eci,rotmat,drotmat]=ecef2eci(pos_ecef,vel_ecef,gmst,dgmst)
% Inputs
        % ECEF position state vector in earth centered Cartesian coordinates [X,Y,Z]
        %ECEF velocity state vector in earth centered Cartesian coordinates [Vx,Vy,Vz]
        %Rotation angle, gmst, from the Greenwich meridian to the local UTC time
        %The rate of change of Greenwich mean sidereal time (earth rotation rate)
%
 %Outputs
        % ECI position state vector in earth centered Cartesian coordinates [X,Y,Z]
        % ECI velocity state vector in earth centered ?Cartesian coordinates [Vx,Vy,Vz]
        %Rotation matrix, rotmat, from ECEF to ECI coordinates
        %Rotation rate matrix. drotmat
%
%User-defined functions called\
        % None
%

pos_ecef=pos_ecef(:);
vel_ecef=vel_ecef(:);
th = gmst;
dth = dgmst;
vel_eci = [];
%
% Compute transformation matrix: rotation angle is -gmst
A=[cos(th),-sin(th),0;sin(th),cos(th),0;0,0,1];
%
% Compute derivative of transformation matrix
Ap=[-sin(th)*dth,-cos(th)*dth,0; cos(th)*dth,-sin(th)*dth,0;0,0,0];
%
% Compute the transformed position coordinates
pos_eci=A*pos_ecef;
%
% Compute the velocity vector using product rule for differentiation
veci=A*vel_ecef + Ap*pos_ecef;
vel_eci = [vel_eci, veci];
%
rotmat = A;
drotmat = Ap;
%
return
```

C.6 evaluate_DC.m

The function evaluate_DC.m computes the Doppler centroid at an image point using the Doppler dentroid polynomial coefficients provided with the radar metadata extracted from product.xml.

Author: Pete Beaulne, DRDC Ottawa, 2011
%

```
% evaluate_DC.m
%
function DC_out = evaluate_DC(DC_coeff,FastTime,FastTimeRef)
%
% This function evaluates the Doppler centroid polynomial in DC_coeff at FastTime (range time)
% OPTIONAL FastTimeRef is DC polynomial reference time
%
%Usage:
        % DC_out = evaluate_DC(DC_coeff,FastTime,FastTimeRef)
% Inputs
        % Doppler centroid coefficient vector, DC_coeff
        %The time from the first image range to the target point being evaluated, FastTime
        %The signal propagation time to the first radar range, FastTimeRef
% Outputs
        % The target point Doppler centroid in Hz
% User-defined functions called
        %None
%
if(nargin <3);
        FastTimeRef=0;
end
% set ref to zero if none passed
DC_out = 0;
%
polytime = FastTime - FastTimeRef;
for kk=1:length(DC_coeff)
        DC_out = DC_out + DC_coeff(kk)*polytime^(kk-1);
end
%
return
%
```

C.7 hms2sec.m

The function hms2sec converts hours, minutes and seconds to seconds

Author: David Vallado

Reference: Vallado, D.A., Fundamentals of Astrodynamics and Applications, Space technology Library, Microcosm Press 2001, ISBN 1-881883-12-4

```
function [utsec ] = hms2sec( hr,min,sec );
%
%  this function converts hours, minutes and seconds into seconds from the
%    beginning of the day.
%
```

```
%  author        : david vallado                  719-573-2600   27 may 2002
%
%  revisions
%                -
%
%  inputs          description                    range / units
%    hr             - hours                        0 .. 24
%    min            - minutes                      0 .. 59
%    sec            - seconds                      0.0 .. 59.99
%
%   outputs       :
%    utsec        - seconds                      0.0 .. 86400.0
%
%  locals        :
%    temp         - temporary variable
%
%  coupling      :
%    none.
%Usage:
%         function [utsec ] = hms2sec( hr,min,sec );
%
       utsec  = hr * 3600.0 + min * 60.0 + sec;
%
```

C.8 interstate.m

The function interstate.m interpolates satellite state vectors to estimate values at intermediate times. This function has been created from a combination of two source functions.

Author: Ishuwa Sikaneta, DRDC Ottawa, modified by Pete Beaulne, DRDC Ottawa, 2005

Reference: Beaulne, P.D. and Sikaneta, I.C., A simple and precise approach to position and velocity eastimation of low earth orbit satellites, DRDC Technical Memorandum, DRDC Ottawa TM 2005.250, December 2005.

```
function istate = interState(state_vectors,time)
%
%interState.m
%
% This function interpolates a state vector
%Usage
        % istate = interState(state_vectors,time)
%
% Inputs
        % Data structure, state_vectors(:), with fields:
                %position
                %velocity
                %time
```

```
        % Data structure, time(:), containing interpolation times
%
% Outputs
        %Data structure, istate, with fields
                % position
                % velocity
%
% User-defined functions called
        %time2secondof day.m
        %diff.m
%

% Unpack the position and velocity state vectors from the input data structure
pos = [state_vectors(:).position];
vel = [state_vectors(:).velocity];
%
%[pos(:).x;pos(:).y;pos(:).z]';
statepVec = pos';
%[vel(:).x;vel(:).y;vel(:).z]';
statevVec = vel';
%
% Calculate the sampling period
state_time_array = time2secofday([state_vectors(:).time]);
desired_time_array = time2secofday(time);

% Compute the sampling period of the state vectors
T = mean(diff(state_time_array));
%
% Compute the interpolation times in terms of state vector sampling
% period relative to first state_vector
interTime = (desired_time_array - state_time_array(1))/T;
%
% Interleave position and velocity state vector components
[m,n]=size(statepVec);
idx=1:m;
state=zeros(2*m,n);
state((2*idx-1),:)=statepVec(idx,:);
state((2*idx),:)=T*statevVec(idx,:);
%
% Scale time variable
interTime=interTime(:);
idx = find((interTime >= 0) & (interTime <= (length(state_vectors)-1)));
interTime = interTime(idx);
[p,q]=size(interTime);

% Create descending powers array
pk=(2*m-1):-1:0;
%
```

```
istate.index = idx;
istate.time = time(idx);
istate.position = zeros(3,p);
istate.velocity = zeros(3,p);
H = inv(qmatrix(length(state_vectors)));
%
% Loop through desired time points and interpolate
for idx=1:p,
        % Create polynomial powers in time for interpolation
        ptvec=(interTime(idx)).^pk;
        vtvec=[pk.*(interTime(idx)).^abs(pk-1)];
        % Compute the interpolated position and velocity state vectors
        istate.position(:,idx)=(ptvec*H*state).';
        istate.velocity(:,idx)=(vtvec*H*state/T).';
end
%
return
%
```

C.9 jday.m

The function jday.m computes the Jullian day from year, month, day and time.

Author: David Vallado, 4 June 2002,

Reference: Vallado, D.A., Fundamentals of Astrodynamics and Applications, Space technology Library, Microcosm Press 2001, ISBN 1-881883-12-4

```
function jd = jday(yr, mon, day, hr, min, sec)
%
%  this function finds the julian date given the year, month, day, and time.
%
%  author        : david vallado                719-573-2600   27 may 2002
%
%Usage
        % jd = jday(yr, mon, day, hr, min, sec)
%
% Inputs                description                    range / units
        %year       -      year                        1900 .. 2100
        %mon        -     month                       1 .. 12
        %day        -     day                         1 .. 28,29,30,31
        %hr         -     universal time hour         0 .. 23
        %min        -      universal time min         0 .. 59
        %sec        -      universal time sec          0.0d0 .. 59.999d0
        %whichtype  -  julian .or. gregorian calender  'j' .or. 'g'
%
%  Outputs       :
        %jd         -     julian date                 days from 4713 bc
```

```
%
%User-defined functions called
        %None
%
%  references    :
%    vallado       2001, 186-188, alg 14, ex 3-14
%

      % ------------------------  implementation   ------------------
      jd = 367.0 * yr  ...
          - floor( (7 * (yr + floor( (mon + 9) / 12) ) ) * 0.25 )   ...
          + floor( 275 * mon / 9 ) ...
          + day + 1721013.5  ...
          + ( (sec/60.0d0 + min ) / 60.0 + hr ) / 24.0;
      %  - 0.5 * sign(100.0 * yr + mon - 190002.5) + 0.5;
return
```

C.10 lla2ecf.m

The function lla3ecf.m converts geodetic latitude, longitude, and terrain height above the WGS84 ellipsoid to ECEF earth centered Cartesian coordinates [X,Y,Z].

Author: Michael Kleder, July 2005

References: "Department of Defense World Geodetic System 1984", National Imagery and Mapping Agency, Last updated June, 2004, NIMA TR8350.2, Page 4-4

```
 function pos_ecef=lla2ecef(pos_lla)
%
% LLA2ECEF - converts latitude, longitude, and altitude to earth-centered, earth-fixed (ECEF)
cartesian coordinates.
%
%Inputs
        % lat = geodetic latitude (radians)
        % lon = longitude (radians)
        % alt = height above WGS84 ellipsoid (m)
%Outputs
        % [x,y,z]
        % x = ECEF X-coordinate (m)
        % y = ECEF Y-coordinate (m)
        % z = ECEF Z-coordinate (m)
% User defined functions called
        % None
% Usage:
        % [x,y,z] = lla2ecef(lat,lon,alt)
%
% Notes: This function assumes the WGS84 model.
%       Latitude is customary geodetic (not geocentric).
```

```
%
% Source: "Department of Defense World Geodetic System 1984"
%       Page 4-4
%       National Imagery and Mapping Agency
%       Last updated June, 2004
%       NIMA TR8350.2
%
% Michael Kleder, July 2005
%

% WGS84 ellipsoid constants:
a = 6378137;
e = 8.1819190842622e-2;
% Make sure triplets are col vectors
sz=size(pos_lla);
if( sz(2) == 3 )
          pos_lla = pos_lla';
end
%
lat = pos_lla(1,:);
lon = pos_lla(2,:);
alt = pos_lla(3,:);
%
% intermediate calculation
% (prime vertical radius of curvature)
N = a ./ sqrt(1 - e^2 .* sin(lat).^2);
%
% results:
x = (N+alt) .* cos(lat) .* cos(lon);
y = (N+alt) .* cos(lat) .* sin(lon);
z = ((1-e^2) .* N + alt) .* sin(lat);
%
pos_ecef = [x;y;z];
% return same dim as input
if(sz(2) == 3 )
          pos_ecef = pos_ecef';
end
%
return
%
```

C.11 load_product_attitude.m

The function load_product_attitude.m opens the file product.xml from the acquired data set and extracts the attitude time stamps and satellite attitude roll, yaw and pitch parameters to the indexed data structure, attitude. For this function to work, a java –based xml reading library must be enabled in Matlab.

```
function attitude = load_product_attitude(productFile)
%
% load_product_attitude.m extracts RADARSAT-2 attitude vector data from the file product.xml
%and creates the indexed data structure, attitude. There is one index step per attitude block in
%product.xml. The angle data in attitude are in degrees.
%
%Input:
        % product.xml, xml file transmitted with every RADARSAT-2 SAR data set.
%
%Output:
        % attitude, indexed data structure with fields:
                %.time, data structure with fields:
                        %.year
                        %.month
                        %.day
                        %.hour
                        %.minute
                        %.second
                        %.usec
                        %.doy  (day of year)
                %.attitude, data structure with fields
                        %.roll
                        %.pitch
                        %.yaw
%
%User-defined functions called
        %time_str2struct.m
        %ymd2doy
%
%Usage
        % attitude = load_product_attitude(productFile)
%
% Author Pete Beaulne, DRDC, January 2012
%

% Open the product and aux xml files
product = xmlread(productFile);
%
% Extract the state vector list
myNode = product.getElementsByTagName('attitudeInformation');
%
if(myNode.getLength > 0)
        % Get the timestamp string
        s_node = myNode.item(0).getElementsByTagName('timeStamp');
        %Loop through the product.xml attitude data
        for k=0:(s_node.getLength-1)
                 timestamp = char(s_node.item(k).getFirstChild.getData);
                attitude(k+1).time = time_str2struct(timestamp);
```

```
        end
        % add doy
        for l=1:(k+1)
                attitude(l).time.doy =
                ymd2doy(attitude(l).time.year,attitude(l).time.month,attitude(l).time.day);
        end
        %
        % Get the roll
        s_node = myNode.item(0).getElementsByTagName('roll');
        for k=0:(s_node.getLength-1)
                attitude(k+1).roll = str2num( s_node.item(k).getFirstChild.getData);
        end
        %
        % Get the pitch
        s_node = myNode.item(0).getElementsByTagName('pitch');
        for k=0:(s_node.getLength-1)
                attitude(k+1).pitch = str2num( s_node.item(k).getFirstChild.getData);
        end
        %
        % Get the yaw
        s_node = myNode.item(0).getElementsByTagName('yaw');
        for k=0:(s_node.getLength-1)
                attitude(k+1).yaw = str2num( s_node.item(k).getFirstChild.getData);
        end
%
end
%
return
```

C.12 load_product_state_vectors.m

The function load_product_state_vectors.m opens the file product.xml from the acquired data set and extracts the orbit state-vector time stamps and satellite position state-vector earth-centered Cartesian coordinate parameters (in m) to the indexed data-structure, state_vectors. For this function to work, a java –based xml reading library must be enabled in Matlab.

```
function state_vectors = load_product_state_vectors(productFile)
%
% load_product_state_vectors.m extracts RADARSAT-2 orbit state-vector data from the file
%product.xml and creates the indexed data structure, state_vectors. There is one index step per
%attitude block in product.xml. The angle data in attitude are in degrees.
%
%Input:
        % product.xml
%
%Output:
        % state_vectors, indexed data structure with fields:
                %.time, data structure with fields:
```

```
                        %.year
                        %.month
                        %.day
                        %.hour
                        %.minute
                        %.second
                        %.usecond
                        %.doy  (day of year)
                %.position, the three element column vector [xpos;ypos;zpos]
                %.velocity, the three element column vector [xvel;yvel;zvel]'
%
%User-defined functions called
        %time_str2struct.m
        %ymd2doy
%
%Usage
        % state_vectors = load_product_state_vectors(productFile)
%
% Author Pete Beaulne, DRDC, January 2012
%

% Open the product.xml file
product = xmlread(productFile);
%
% Extract the state vector list
myNode = product.getElementsByTagName('orbitInformation');
%
% Loop through the sate vector list in product.xml
if(myNode.getLength > 0)
        % Get the timestamp string
        s_node = myNode.item(0).getElementsByTagName('timeStamp');
        for k=0:(s_node.getLength-1)
                timestamp = char(s_node.item(k).getFirstChild.getData);
                state_vectors(k+1).time = time_str2struct(timestamp);
        end
        % add doy (day of year)
        for l=1:(k+1)
                state_vectors(l).time.doy =
                ymd2doy(state_vectors(l).time.year,state_vectors(l).time.month,state_vectors(l).ti
                me.day);
        end
        %
        % Get the xpos
        s_node = myNode.item(0).getElementsByTagName('xPosition');
        for k=0:(s_node.getLength-1)
                state_vectors(k+1).position(1,1) = str2double(
                s_node.item(k).getFirstChild.getData);
         end
```

```
        %
        % Get the ypos
        s_node = myNode.item(0).getElementsByTagName('yPosition');
        for k=0:(s_node.getLength-1)
                state_vectors(k+1).position(2,1) = str2double(
                s_node.item(k).getFirstChild.getData);
        end
        %
        % Get the zpos
        s_node = myNode.item(0).getElementsByTagName('zPosition');
        for k=0:(s_node.getLength-1)
                state_vectors(k+1).position(3,1) = str2double(
                s_node.item(k).getFirstChild.getData);
        end
        %
        % Get the xvel
        s_node = myNode.item(0).getElementsByTagName('xVelocity');
        for k=0:(s_node.getLength-1)
                state_vectors(k+1).velocity(1,1) = str2double(
                s_node.item(k).getFirstChild.getData);
        end
        %
        % Get the yvel
        s_node = myNode.item(0).getElementsByTagName('yVelocity');
        for k=0:(s_node.getLength-1)
                state_vectors(k+1).velocity(2,1) = str2double(
                s_node.item(k).getFirstChild.getData);
        end
        %
        % Get the zvel
        s_node = myNode.item(0).getElementsByTagName('zVelocity');
        for k=0:(s_node.getLength-1)
                state_vectors(k+1).velocity(3,1) = str2double(
                s_node.item(k).getFirstChild.getData);
        end
%
end
```

C.13 qmatrix.m

This function qmatrix.m generates a Hermite polynomial matrix for use in state-vector interpolation.

Author : Ishuwa Sikaneta, DRDC-Ottawa, November 22, 2002

Reference: Beaulne, P.D. and Sikaneta, I.C., A simple and precise approach to position and velocity eastimation of low earth orbit satellites, DRDC Technical Memorandum, DRDC Ottawa TM 2005.250, December 2005.

```
function Q=qmatrix(n)
% This function will generate the Hermite polynomial matrix
%
% Usage:
        %Q=qmatrix(n);
%
% Inputs
        % n         ----------> Number of tie points
%
% Outputs:
        % Q         ----------> Hermite interpolating matrix
%
% User defined functions called
        % None
%
% Caveats:
        % State vector samples have been sampled uniformly in time.
%
% Author: Ishuwa Sikaneta ARN/DRDC-Ottawa, November 22, 2002

% Create descending powers for time variable
k=(2*n-1):-1:0;

% Create first row
Q=[0.^k;0.^abs(k-1)];
%
% Create remaining rows
for l=1:(n-1);
        Q=[Q;l.^k;k.*l.^(k-1)];
end
%
return
%
```

C.14 rad.m

The simple function rad.m converts angles in degrees to angles in radians.

Author: Chuck Livingstone CCRS, 1990

```
function  rd=rad(t)
%
% rad returns angles in radians for (data vector) inputs in degrees
%
%Input
        % an angle or array of angles in degrees
%
```

```
%Output
        %an angle or an array of angles in radians
%
%Usage
        % rd=rad(t)
%
%User-defined functions called
        %None
%
%Author: Chuck Livingstone, CCRS, 1990
%
rd=(pi./180.0).*t;
%
```

C.15 read_DefinitiveOrbit.m

The function read_DefinitiveOrbit.m opens and reads data from MDA a definitive ASCII orbit file which has been retrieved from the RADARSAT-2 definitive orbit data base. The data are inserted into the indexed data structure Definitive_State(:) where the index number is the number of the state vector in the file. This function is defined for use with a control script when analyzing a specific RADARSAT-2 SAR data acquisition. Internal tests identify input file problems that cannot be dealt with automatically and return error messages.

The function read_definitiveOrbit.m was derived from read_MDA_orbit.m and there is a significant overlap in the code.

Author: Pete Beaulne, DRDC Ottawa January 2012

```
function Definitive_State = read_DefinitiveOrbit(orbit_file)
%
% This function loads the MDA definitive ASCII orbit file for a specific data acquisition and
%converts the data to a Matlab structure for further use.
%
%Input
        %orbit_file, path to the ASCII orbit file data from the work-space including the file name
%
%Output
        %Definitive_State, data structure with fields:
                %.orbit_num
                %.time, data structure with fields:
                        %.year
                        %.doy  (day of year)
                        %.month
                        %.day
                        %.hour
                        %.minute
                        %.second
```

```
                        %usec
                        %utcsec (second of day)
                        %JD  (Julian Day in seconds)
                        %MJD (Julian Day offset by 2400000.5)
                %.position, column vector [xposn;yposn;zposn]
                %.velocity, column vector [xvel;yvel;zvel]
%
% user-defined functions called
        %jday.m
        %doy2date.m
        %time2secofday.m
%
%Error messages:
        % “error opening: file not found ‘orbit_file’”  (empty matrix returned for
        Definitive_State)
        % ‘Bad format: non character line in orbit file”  (END_OF_FILE not found)
%
%Usage
        % Definitive_State = read_DefinitiveOrbit(orbit_file)
        %
%
% Author: Pete Beaulne, DRDC Ottawa, February 2012
 %

% Open orbit_file for reading
fid = fopen(orbit_file, 'r');
% Test for file existance
if(fid == -1)
        fprintf('error opening %s: file not found\n\n', orbit_file);
        Definitive_State=[];
        return;
end
%
% Set counters
state_num=0;
line_num=0;
start_extracting=0;
%
% Month boundary days for a leap year
dleap=[1,32,61,92,122,153,183,214,245,275,306,336];
% Month boundary days for a regular year
dreg=[1,32,60,91,121,152,182,213,244,274,305,335];
%
%Start file reading
while 1
        line=fgetl(fid);
        line_num = line_num +1;
        % break at end of file
```

```
if(strfind(line,'END_OF_FILE')), break, end
        % break if non character line
        if ~ischar(line)
        fprintf('Bad format: non character line in orbit file.\n\n Exiting \n');
        break;
end
%
% get orbit number
if(strfind(line,'ORBIT_NUMBER'))
        tmpxstr=regexp(line,'\=','split');
        orbit_num = str2double(char(tmpxstr(2)));
end
%
% find last line before state vector entries
if( strfind(line,'Position is in meters') )
        start_extracting = 1;
        continue;
end
%
% extract state vectors here
if( start_extracting )
        %Increment state number
        state_num = state_num+1;
        %
        xstr=regexp(line,'\-','split');
        tstr1=char(xstr(3));
        tstr=regexp(tstr1,'\:','split');
        %
        Definitive_State(state_num).orbit_num = orbit_num;
        Definitive_State(state_num).time.year=str2double(char(xstr(1)));
        Definitive_State(state_num).time.doy=str2double(char(xstr(2)));
        %
        % convert doy to month, day
        [month,day] =
        doy2date(Definitive_State(state_num).time.doy,Definitive_State(state_num).time
        .year);
        Definitive_State(state_num).time.month=month;
        Definitive_State(state_num).time.day = day;
        %
        Definitive_State(state_num).time.hour=str2double(char(tstr(1)));
        Definitive_State(state_num).time.minute=str2double(char(tstr(2)));
        Definitive_State(state_num).time.second=floor(str2double(char(tstr(3))));
        %
        Definitive_State(state_num).time.usec=round(1000000*(str2double(char(tstr(3)))
        -Definitive_State(state_num).time.second));
        %
        Definitive_State(state_num).time.utcsec =
        time2secofday([Definitive_State(state_num).time]);
```

```
                Definitive_State(state_num).time.JD =
                jday(Definitive_State(state_num).time.year,
                Definitive_State(state_num).time.month, ...
                Definitive_State(state_num).time.day, Definitive_State(state_num).time.hour, ...
                Definitive_State(state_num).time.minute, ...
                Definitive_State(state_num).time.second+Definitive_State(state_num).time.usec/
                1e6);
                %
                Definitive_State(state_num).time.MJD = Definitive_State(state_num).time.JD -
                2400000.5;
                %
                posline = fgetl(fid);
                velline = fgetl(fid);
                separator_line = fgetl(fid);
                %Increment line number
                line_num = line_num +3;
                pstr=regexp(posline,'\ ','split');
                vstr=regexp(velline,'\ ','split');
                %
                %Capture the position vector
                Definitive_State(state_num).position = [str2double(char(pstr(1)));
                str2double(char(pstr(2))); str2double(char(pstr(3)))];
                %
                %Capture the velocity vector
                Definitive_State(state_num).velocity = [str2double(char(vstr(1)));
                str2double(char(vstr(2))); str2double(char(vstr(3)))];
        %
        %Complete one extraction cycle
        end
%
%Complete while loop
end
%
%printf('Loaded %d state vectors\n',state_num);
fclose(fid);
%
%Return to calling program
end
```

C.16 read_MDA_orbit.m

The function read_MDA_orbit.m opens and reads data from MDA predicted and definitive ASCII orbit files which are accessed from a file list in a directory. The data are inserted into the indexed data structure MDAstate(:) where the index number is the number of the state vector in the file. This function is defined for interactive use when analyzing a large set of SAR acquisition metadata to allow the analyst to flag problem files and to decide how to deal with the problems encountered. A number of internal tests return control to the user.

Author: Pete Beaulne, DRDC Ottawa October 2011

```
function MDAstate = read_MDA_orbit(orbit_file,year_str)
%
% This function loads the MDA predicted and definitive ASCII orbit files from a list and extracts
%the data for analysis.
%
%Input
        %MDA ASCII orbit file with path from the Matlab workspace
        %year- as a string variable
%
%Output
        % data structure MDAstate(:) with fields
                %.orbit
                %.time (which has subfileds)
                        %.year
                        %.doy (day of year)
                        %.month
                        %.day
                        %.hour
                        %.minute
                        %.second
                        %.usec
                        %utcsec
                        %.JD
                        %.MJD
                %.position (column vector [X;Y;Z])
                %.velocity (column vector [Vx;Vy;Vz])
%
% User-defined functions called
        %jday.m
        %time2secofday.m
%
%Error messages:
        % “error opening: file not found ‘orbit_file’”  (empty matrix returned for
        Definitive_State)
%
%Usage
        % MDAstate = read_MDA_orbit(orbit_file,year_str)
%
% Author: Pete Beaulne, DRDC Ottawa, October 2011
%

% Open the orbit file for reading
fid = fopen(orbit_file, 'r');
%
%Test for file existance
if(fid == -1)
```

```
        fprintf('error opening %s: file not found\n\n', orbit_file);
        MDAstate=[];
        return;
end
%
% Set counters
state_num=0;
line_num=0;
%
%Month day boundaries for a leap year
dleap=[1,32,61,92,122,153,183,214,245,275,306,336];
Month day boundaries for a regular year
dreg=[1,32,60,91,121,152,182,213,244,274,305,335];
%
%Start the file reading operation
while 1
        line=fgetl(fid);
        %increment the line number
        line_num = line_num +1;
        %
        % Look for the end of the file and exit if there is an empty line.
        if ~ischar(line), break, end
        %
        %Find the orbit number and read it
        if(line_num == 1)
                tmpxstr=regexp(line,'\:','split');
                tmptstr=char(tmpxstr(2));
                file_orbit_num = str2num(tmptstr(1:5));
        end
        %
        %Read the parameters from this orbit
        if line_num > 10
                if( strfind(line, year_str) )
                        %Increment the state number index
                        state_num = state_num+1;
                        %
                        xstr=regexp(line,'\-','split');
                        tstr1=char(xstr(3));
                        tstr=regexp(tstr1,'\:','split');
                        %
                        MDAstate(state_num).orbit_num = file_orbit_num;
                        MDAstate(state_num).time.year=str2num(char(xstr(1)));
                        MDAstate(state_num).time.doy=str2num(char(xstr(2)));
                        %
                        % convert doy to month, day (year is hard wired)
                        if (MDAstate(state_num).time.year == 2008)
                                d=dleap;
                        else
```

```
                    d=dreg;
                end
                %
                %Read time data
                k=find(d<=MDAstate(state_num).time.doy);
                MDAstate(state_num).time.month=max(k);
                MDAstate(state_num).time.day=MDAstate(state_num).time.doy-
                d(max(k))+1;
                %
                MDAstate(state_num).time.hour=str2num(char(tstr(1)));
                MDAstate(state_num).time.minute=str2num(char(tstr(2)));
                MDAstate(state_num).time.second=floor(str2num(char(tstr(3))));
                MDAstate(state_num).time.usec=round(1000000*(str2num(char(tstr(3)))
                -MDAstate(state_num).time.second));
                %
                MDAstate(state_num).time.utcsec =
                time2secofday([MDAstate(state_num).time]);
                %
                MDAstate(state_num).time.JD = jday(MDAstate(state_num).time.year,
                MDAstate(state_num).time.month, ...
                MDAstate(state_num).time.day, MDAstate(state_num).time.hour, ...
                MDAstate(state_num).time.minute, ...
                MDAstate(state_num).time.second+MDAstate(state_num).time.usec/1e6
                );
                %
                MDAstate(state_num).time.MJD = MDAstate(state_num).time.JD -
                2400000.5;
                %
                %Read state-vector data
                posline = fgetl(fid);
                velline = fgetl(fid);
                %Set the line number
                line_num = line_num +2;
                pstr=regexp(posline,'\ ','split');
                vstr=regexp(velline,'\ ','split');
                %
                MDAstate(state_num).position = [str2num(char(pstr(1)));
                str2num(char(pstr(2))); str2num(char(pstr(3)))];
                %
                MDAstate(state_num).velocity = [str2num(char(vstr(1)));
                str2num(char(vstr(2))); str2num(char(vstr(3)))];
                %
            %endifstrfind(…
            end
        %end if line_num
        end
%end while(1) loop
end
```

```
%
fclose(fid);
%
return
%
```

C.17 sec2hms.m

The function sec2hms converst UTC seconds to hours, minutes and seconds

Author: David Valado

Reference: Vallado, D.A., Fundamentals of Astrodynamics and Applications, Space technology Library, Microcosm Press 2001, ISBN 1-881883-12-4

```
 function [hr,min,sec] = sec2hms( utsec );
%
%                       function sec2hms
%
%  this function converts seconds from the beginning of the day into hours,
%    minutes and seconds.
%
%  author        : david vallado                719-573-2600   25 jun 2002
%
%  revisions
%              -
%
%  inputs          description                    range / units
%    utsec       - seconds                      0.0 .. 86400.0
%
%  outputs       :
%    hr          - hours                      0 .. 24
%    min         - minutes                     0 .. 59
%    sec         - seconds                    0.0 .. 59.99
%
%  locals        :
%    temp        - temporary variable
%
%  coupling      :
%    none.
%
% [hr,min,sec] = sec2hms( utsec );
%

       % ------------------------  implementation   ------------------
       temp  = utsec / 3600.0;
```

```
hr    = fix( temp );
min   = fix( (temp - hr)* 60.0 );
sec   = (temp - hr - min/60.0 ) * 3600.0;
```

C.18 time2secondsofday.m

The function time2secondsofday.m converts a data structure which as fields .hour. .minute, .second, .usec into seconds from the start of a day.

Author**:** Pete Beaulne, DRDC Ottawa, September 2011

```
function secs_in_day = time2secofday(intime)
% time2secofday.m
%
% Input: indexed data structure, intime, with fields
        %.hour
        %.minute
        %.second
        %.usec
%
%Output
        %Array of times in seconds
%
%User-defined functions called
        %None
%
%Usage:
        % secs_in_day = time2secofday(intime);
%
n=length(intime);
%
for k=1:n
        secs_in_day(k,1) =
        3600*intime(k).hour+60*intime(k).minute+intime(k).second+intime(k).usec/1000000;
         %intime(k).sec_of_day = secs_in_day;
end
%
return
%
```

C.19 time_str2struct.m

The function time_str2structure.m reads a literal time string of the form 'YYYY-MM-DDThh:mm:ss.ssssssZ' and converts it to a Matlab data structure that has fields for each time variable.

This function is frequently used to convert the time contents of xml files to a Matlab-usable form.

Author: Pete Beaulne, DRDC Ottawa, January 2012

```
function time_structure = time_str2struct(time_string)
%
%  Author: Pete Beaulne, Jan 2012
%
%  Usage-
        %time_structure = time_str2struct(time_string)
%
%  Input-
        %time_string: a time character string of the format
        %'YYYY-MM-DDThh:mm:ss.ssssssZ' (trailing Z optional)
%
%  Output-
        %time_structure: a data-structure with fields
                %.year:         year
                %.month:        month
                %.day:          day of month
                %.doy:          day of year
                %.hour:         hour of the day
                %.minute:       minutes
                %.second:       seconds
                %.usec:         microseconds
                %.utcsec:       UTC time of day in seconds
                %.JD:           Julian Date (including fraction of day)
                %.MJD:          Modified Julian Date (including fraction of day)
%
%  User functions called –
        %ymd2doy.m
%

% day of month start for leap years
dleap=[1,32,61,92,122,153,183,214,245,275,306,336];
% doy of 1st of every month for regular years
dreg=[1,32,60,91,121,152,182,213,244,274,305,335];
%
tmpxstr=regexp(time_string,'\T','split');
%
date_substr=char(tmpxstr(1));
time_substr=char(tmpxstr(2));
%
date_xstr=regexp(date_substr,'\-','split');
%
time_structure.year = str2double(char(date_xstr(1)));
time_structure.month = str2double(char(date_xstr(2)));
time_structure.day = str2double(char(date_xstr(3)));
% day of the year
time_structure.doy = ymd2doy(time_structure.year,time_structure.month,time_structure.day);
```

```
%
time_structure.hour = str2double(time_substr(1:2));
time_structure.minute = str2double(time_substr(4:5));
time_structure.second = str2double(time_substr(7:8));
time_structure.usec = str2double(time_substr(10:15));
%
% time in UTC seconds since beginning of day
time_structure.utcsec = time2secofday(time_structure);
% Julian data (UTC)
%
time_structure.JD = jday(time_structure.year, time_structure.month, time_structure.day, ...
time_structure.hour, time_structure.minute, time_structure.second+time_structure.usec/1e6);
%
% modified Julian date
time_structure.MJD = time_structure.JD - 2400000.5;
%
% end of function time_str2struct
end
```

C.20 ymd2doy.m

The function ymd2doy.m accepts integer variables year, month, day and converts the date to the day of the year. The function is hard-wired for leap years up to 2024.

Author Pete Beaulne, January 2012

```
 function doy = ymd2doy(year,month,day)
%  ymd2doy.m
%  Author: Pete Beaulne, Jan 2012
%
%  Usage-
        %doy = ymd2doy(year,month,day)
%
%  Inputs:
        %year:          integer year
        %month:         integer month
        %day:           integer day
%
%  Output-
        %doy:           integer day of the year
%
%User functions called –
        %NONE
%

% doy of 1st of every month for leap years
dleap=[1,32,61,92,122,153,183,214,245,275,306,336];
% doy of 1st of every month for normal years
```

```
dreg=[1,32,60,91,121,152,182,213,244,274,305,335];
%
% day of the year
if (year == 2008 || year == 2012 || year == 2016 || year == 2020 || year ==2024)
        d=dleap;
else
        d=dreg;
end
%
doy = d(month)+day-1;
%
end
```

This page intentionally left blank.

List of symbols/abbreviations/acronyms/initialisms

[Enter list here, if applicable. If not, delete the page.]

CSA	Canadian Space Agency
DND	Department of National Defence
DRDC	Defence Research & Development Canada
DRDKIM	Director Research and Development Knowledge and Information Management
ECEF	Earth-Centred, Earth-Fixed
GMTI	Ground Moving Target Indication
MDA	MacDonald, Dettwiler and Associates
R&D	Research & Development
SAR	Synthetic Aperture Radar
WGS 84	World Geodetic System, 1984

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall 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 K1A 0Z4

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

UNCLASSIFIED
(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC JUNE 2010

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

Modelling RADARSAT-2 Doppler centroids for marine applications:

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Livingstone, C.E. and Beaulne, P.D.

5. DATE OF PUBLICATION (Month and year of publication of document.)

August 2012

6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)

126

6b. NO. OF REFS (Total cited in document.)

10

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 Memorandum

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 K1A 0Z4

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)

15eq

9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DRDC Ottawa TM 2012-097

10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

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

Unlimited

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.))

Unlimited

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.)

The Doppler centroid of acquired SAR (Synthetic Aperture Radar) and SAR-GMTI (Ground Moving Target Indication) signal data is required to refine theoretically-defined processing filters for SAR processing of stationary terrain and to minimize GMTI artefacts for moving targets. When precision orbit and attitude data are used in conjunction with an ellipsoidal earth model, SAR processing filters can be adjusted for the observation geometry to generate SAR imagery that is acceptable for most applications. Residual earth rotation artefacts in this imagery have no significant impact on its use.

The residual earth rotation effects after matched filter compensation are significant for GMTI measurements. For land scenes, the background terrain can be assumed to be static and radar returns from scene elements that do not contain moving targets can be used to provide the required reference. On the sea surface, the background terrain is moving and radar returns are often too weak to capture reliable Doppler centroid information from the acquired signal data. Although RADARSAT-2 has been designed to minimize earth rotation Doppler shifts by controlling the satellite attitude over the orbit to point the satellite beam near the zero Doppler azimuth angle, the attitude control law that is implemented in the satellite uses satellite position information and does not account for orbit and earth ellipticity or radar range. In addition there are perturbations in the reported attitude data due to azimuth pointing control lags and small misalignments between the attitude sensor and antenna coordinate systems. The composite effect on the Doppler centroid of the radar returns is not significant for most imaging operations but is large enough to impact GMTI measurements in cases where reference signals from static terrain are not available.

This Technical Memorandum presents a model-based analysis of the Doppler frequency properties of RADARSAT-2 data and presents a RADARSAT-2 Doppler centroid estimation model that is sufficiently accurate for GMTI measurements of marine targets in the open ocean. Model verification using metadata from ten months of RADARSAT-2 data acquisitions is described.

A RADARSAT-2 Doppler centroid estimation tool suitable for use with individual ocean data sets is presented.

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 is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

RADARSAT-2 attitude control;Doppler Centroid;attitude control model;residual earth rotation effects;control error

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