

# Simulation of Radar Profiles for Satellites Using Mercury Method of Moments

Daniel Topa  
daniel.topa@hii-tds.com

*Mission Technologies  
Huntington Ingalls Industries  
Kirtland AFB, NM*

October 16, 2024

## Abstract

A brief survey of characterizing the three dimensional radar cross section of satellites.

## Contents

<b>1</b>	<b>Precís</b>	<b>2</b>
1.1	Running the Code . . . . .	2
1.2	Radar . . . . .	2
<b>2</b>	<b>Overview: Modeling Radar Cross Section</b>	<b>2</b>
2.1	Radar . . . . .	2
2.2	About . . . . .	3
<b>3</b>	<b>Additional Information</b>	<b>3</b>
3.1	YouTube Videos . . . . .	3
3.2	Further Reading . . . . .	3
<b>A</b>	<b>Mercury Method of Moments: Distribution and Rights</b>	<b>4</b>
A.1	Distribution Letter for Software . . . . .	4
A.2	Copyright Statement by the Author . . . . .	6
A.3	Legal Statement . . . . .	6
A.4	Obtaining Software and Documentation . . . . .	8
A.5	Distribution Contents . . . . .	8
A.5.1	Executables . . . . .	8
A.5.2	Documentation . . . . .	8
<b>B</b>	<b>Mercury Method of Moments: Data Formats</b>	<b>8</b>
B.1	Numeric Results . . . . .	8

<b>C Mercury Method of Moments: Software Toolkit</b>	<b>9</b>
C.1 <code>rcsharvester.f08</code> . . . . .	9
C.1.1 Class Electric Fields: <code>m-class-electric-fields.f08</code> . . . . .	9
C.1.2 <code>m-class-electric-fields.f08</code> . . . . .	9
C.1.3 Class Data File: <code>m-class-data-file.f08</code> . . . . .	11

## 1 Precís

### 1.1 Running the Code

`./MMoM\4.1.12 geofilename`

### 1.2 Radar

[Topa 2020c] [Topa 2020c] Working with CAF files, producing output, compressing data. Topa 2020d Topa (2020d)

## 2 Overview: Modeling Radar Cross Section

### 2.1 Radar

Wave speed equation

$$\lambda\nu = c \quad (1)$$

band	$\nu$	$\lambda$
HF	3 – 30 MHz	10 – 1 m
UHF	30 – 300 MHz	0.1 – 0.01 m
VHF	300 – 1000 MHz	0.01 – 0.03 m
L	1 – 2 GHz	30 – 15 mm
S	2 – 4 GHz	15 – 7.5 mm
C	4 – 8 GHz	7.5 – 3.7 mm
X	8 – 12 GHz	3.7 – 2.5 mm
Ku	12 – 18 GHz	2.5 – 1.7 mm
K	18 – 27 GHz	1.7 – 1.1 mm
Ka	27 – 40 GHz	1.1 – 0.75 mm
V	40 – 75 GHz	0.75 – 0.4 mm
W	75 – 110 GHz	0.4 – 0.27 mm
mm	110 – 300 GHz	0.27 – 0.1 mm

Table 1: IEEE Standard Designations for Radar Bands (Bruder et al. 2003).

- (A) Build a CAD model of the satellite (\*.cad)
- (B) Seal the CAD mesh
- (C) Create geometry file (\*.geo)
- (D) Irradiate object with Mercury MoM
- (E) Harvest backscatter
- (F) Construct RCS
- (G) Resolve RCS measurements into spherical harmonics

## 2.2 About

- (A) Build a CAD model of the satellite (\*.cad)
- (B) Seal the CAD mesh
- (C) Create geometry file (\*.geo)
- (D) Irradiate object with Mercury MoM
- (E) Harvest backscatter
- (F) Construct RCS
- (G) Resolve RCS measurements into spherical harmonics

## 3 Additional Information

### 3.1 YouTube Videos

YouTube offers useful didactic presentations and simulations.

1. The Radar cross-section of backscattering objects
2. Basic Concepts of Radar Cross Section (RCS)
3. Mie scattering
4. Mie theory (BME51 Lecture 5)
5. Mie Scattering

### 3.2 Further Reading

Radar rudiments

1. D. K. Barton and H.R. Ward (1969). *Handbook of Radar Measurement*. New York, NY: Penguin Random House
2. Andrei A. Kolosov (1987). *Over the Horizon Radar*. Artech House. ISBN: 9780890062333. URL: <https://us.artechhouse.com/Over-the-Horizon-Radar-P254.aspx>
3. Peyton Z Peebles (2007). *Radar principles*. John Wiley & Sons

Radar cross section

1. JW Jr Crispin (2013). *Methods of radar cross-section analysis*. Elsevier
2. Allen E Fuhs (1982). *Radar cross section lectures*. Monterey, California, Naval Postgraduate School. URL: <https://calhoun.nps.edu/server/api/core/bitstreams/9e69ec48-4628-4243-9f9b-7e879521f7f8/content>
3. Eugene F Knott, John F Schaeffer, and Michael T Tulley (2004). *Radar cross section*. SciTech Publishing
4. M Madheswaran and P Suresh Kumar (2012). “Estimation of wide band radar cross section (RCS) of regular shaped objects using method of moments (MOM)”. in: *Ictact Journal on Communication Tech-nology* 3.2, pp. 536–541

Method of Moments

1. Walton C Gibson (2021). *The method of moments in electromagnetics*. Chapman and Hall/CRC
2. Roger F Harrington (1987). “The method of moments in electromagnetics”. In: *Journal of Electromagnetic waves and Applications* 1.3, pp. 181–200
3. Cai-Cheng Lu and Chong Luo (2003). “Comparison of iteration convergences of SIE and VSIE for solving electromagnetic scattering problems for coated objects”. In: *Radio Science* 38.2, pp. 11–1
4. Jiade Yuan, Changqing Gu, and Guodong Han (2009). “Efficient generation of method of moments matrices using equivalent dipole-moment method”. In: *IEEE Antennas and Wireless Propagation Letters* 8, pp. 716–719

Using Mercury MoM and post-processing

1. Daniel Topa (Mar. 2020c). *Radar Cross Section Models for AFCAP Dashboard: Rapid Report 2020-02: Corrected*. Briefing
2. Daniel Topa (Apr. 2020a). *Mercury Method of Moments Adjunct Visualization Tool: Trials and Tribulations*. Tech. rep. ARFL/RVB
3. Daniel Topa (Apr. 2020d). *Radar Cross Section: Phase 1 Summary Report*. Tech. rep. ARFL/RVB
4. Daniel Topa (2020b). *Mercury Method of Moments: AFRL Quick Start Guide*. Tech. rep. AFRL

## A Mercury Method of Moments: Distribution and Rights

### A.1 Distribution Letter for Software

The subsequent distribution letter was signed by Randy J. Petyak of the NASA Software Release Authority and describes terms for distribution, Government rights, and the ITAR status of the software.

December 11, 2019

Air Force Research Laboratory  
RVB  
3550 Aberdeen Ave SE  
Kirtland Air Force Base, NM 87117-5776  
Attn: Mr. Nelson Bonito

Subject: Transmittal of Mercury MoM version 4.1.12, MM\_Viz Code.

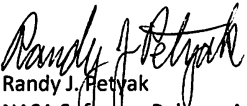
This distribution letter details the terms for distribution, the Government rights in the software, and the ITAR status of the software. The software usage agreement you signed covers Mercury MoM and MMViz executable codes on both Linux 64 bit and Windows 64 bit. The Mercury MoM software is copyrighted by Matrix Compression, LLC. of which the Government retains certain rights to the software, and must be controlled as outlined in the signed Software Usage Agreement.

NASA furnishes this software under the condition that no further dissemination of the software shall be made without prior written permission of the NASA Langley Research Center. Additionally, this software has been designated as ITAR and needs appropriate protection while on the DVD or on an installed machine.

Note: The software falls under the purview of the U.S. Munitions List (USML), as defined in the International Traffic in Arms Regulations (ITAR), 22 CFR 120-130, and is export controlled. It shall not be taken out of the U.S. nor transferred to foreign nationals in the U.S. or abroad, without specific approval of a knowledgeable export control official, and/or unless an export license/license exemption is obtained/available from the United States Department of State. Violation of these regulations is punishable by fine, imprisonment, or both.

We are interested in your use of this software and the results you obtain. Please include us on your mailing list for any publications that may result from your use of this code.

If you have any additional questions related to your request, please contact me.

  
Randy J. Petyak  
NASA Software Release Authority  
(202) 358-4387

## A.2 Copyright Statement by the Author

=====

MERCURY MOM(TM) ( Copyrighted and Patents Issued)  
MATRIX COMPRESSION TECHNOLOGIES, LLC

For licensing information contact:

John Shaeffer

3278 Hunterdon Way

Marietta, Georgia 30067

770.952.3678

Copyright 2006 Matrix Compression Technologies, LLC.

This software was developed under NASA Contracts NAS1-02057, NAS1-02117, NNL08AA00B, and NNL13AA08B, and the U.S. Government retains certain rights.

The Government, and others acting on its behalf, retain a paid-up, nonexclusive, irrevocable, worldwide license to reproduce, prepare derivative works, and perform publicly and display publicly (but not to distribute copies to the public) by or on behalf of the Government, without any obligation of confidentiality on the part of the U.S. Government. Such license extends to use by NASA contractors, and others working under agreements with the U.S. Government; provided that use of the software shall not be allowed to any person or entity where such use is not in direct performance of a contract with the United States; and provided that such use is not for internal research and development by the contractor or others that is not directly funded by the United States.

=====

## A.3 Legal Statement

MERCURY MOM™

Copyrighted

US Patents: 7,742,886; 7,844,407; 8,209,138; 8,725,464

Copyright 2006 Matrix Compression Technologies, LLC.

This software was developed under NASA Contracts NAS1-02057, NAS1-02117, NNL08AA00B, and NNL13AA08B, and the U.S. Government retains certain rights.

The Government, and others acting on its behalf, retain a paid-up, nonexclusive, irrevocable, worldwide license to reproduce, prepare derivative works, and perform publicly and display publicly (but not to distribute copies to the public) by or on behalf of the Government, without any obligation of confidentiality on the part of the U.S. Government. Such license extends to use by NASA contractors, and others working under agreements with the U.S. Government; provided that use of the software shall not be allowed to any person or entity where such use is not in direct performance of a contract with the United States; and provided that such use is not for internal research and development by the contractor or others that is not directly funded by the United States.

Matrix Compression Technologies, L.L.C. expressly disclaims any and all warranties, including the warranty of non-infringement, the warranty of merchantability, and the warranty of fitness for a particular purpose. Matrix Compression Technologies, L.L.C. shall not be obligated to indemnify or pay any party for consequential damages or any other damages arising from the use of the MERCURY MOM™ software. Non-U.S. Government entities shall not distribute the MERCURY MOM™ software to any third party without the express written permission of Matrix Compression Technologies, L.L.C.

MATRIX COMPRESSION TECHNOLOGIES, LLC

John Shaeffer  
3278 Hunterdon Way  
Marietta, Georgia 30067  
john@shaeffer.com  
770.952.3678

=====

NASA ITAR notice:

Note: The enclosed software falls under the purview of the U.S. Munitions List (USML), as defined in the International Traffic in Arms Regulations (ITAR), 22 CFR 120-130, and is export controlled. It shall not be taken out of the U.S. nor transferred to foreign nationals in the U.S. or abroad, without specific approval of a knowledgeable export control official, and/or unless an export license/license exemption is obtained/available from the United States Department of State. Violation of these regulations is punishable by fine, imprisonment, or both.

## A.4 Obtaining Software and Documentation

For more information regarding this document contact the following:

Kam W. Hom  
NASA  
Langley Research Center  
Mail Stop 207  
Hampton, Virginia 23681-2199  
757-864-9608  
kam.w.hom@nasa.gov

or

Jeffrey A. Miller, PhD  
NASA  
Langley Research Center  
Mail Stop 207  
Hampton, Virginia 23681-2199  
757-864-9611  
jeffrey.allen.miller@nasa.gov

Figure 1: Contact information to request Mercury MoM Software and Documentations

## A.5 Distribution Contents

### A.5.1 Executables

1. Linux 64-bit
2. Windows 64-bit

### A.5.2 Documentation

The distribution includes four documents in PDF which are marked as CUI:

1. User's Guide
2. Pill Tutorial
3. Code Validation Report
4. Benchmark Tests

## B Mercury Method of Moments: Data Formats

### B.1 Numeric Results

The MoM RCS data is delivered in a matrix with  $m$  rows and  $n$  columns (standard matrix addressing).



$1 \leq m \leq 28$  MHz (integer steps)  
 $1 \leq n \leq 90$  degrees (integer steps)

The matrix is WIDE (more columns than rows)

Frequency partition: row 1: 3 MHz row 2: 4 MHz . . . row 28: 30 Mhz

Let  $r$  index the rows. Then frequency  $\nu$  is in row  $= \nu - 2$

Angular partition col 1: 0 col 2: 1 . . . col 181: 180

col 1 col 2 col 3 col 181 0 1 2 . . . 180

Let  $c$  be the column index. The measurement for angle alpha is in column  $c = \alpha + 1$

The test asset is symmetric:  $\sigma(\alpha) = \sigma(-\alpha)$

But the matrix can easily be delivered in other forms, such as the transpose (interchange rows and columns), or packed into a linear array.

Sample:

4.16411, 4.14247, 4.07319, 3.95637, 3.79263, 3.58287, 3.32827, 3.0303, . . .  
 18.2776, 18.2369, 18.1199, 17.9248, 17.6523, 17.3041, 16.8817, 16.3876, . . .  
 25.6306, 25.5886, 25.463, 25.2538, 24.9618, 24.5882, 24.1346, 23.6028, . . .  
 . . .

## C Mercury Method of Moments: Software Toolkit

Mercury MoM produces thousands of lines of output to a `*.4112.txt` file, a mix of numbers and strings. Once the data portions are located, they can be harvested straightaway. However, the text messages include debug information and the text patterns are varied.

Data analysis on data sets with a large number of facets can take several hours.

### C.1 rcsharvester.f08

```
! harvest the electric field values from the ASCII file *.4112.txt mixed text and numeric
lines
! compute the mean total RCS and write these values
```

#### C.1.1 Class Electric Fields: m-class-electric-fields.f08

##### C.1.2 m-class-electric-fields.f08

The primary output of the simulation are the electric fields. The input electric field is resolved into two polarization axes: horizontal and vertical. Each of these fields are resolved into horizontal and vertical components creating four complex vectors (line 21) whose length matches the angular sample size.

The class `m-class-electric-fields.f08` reads the text file and harvests the electric fields eventually passing back a composite value (lines 65-66) for all four components of the scattering return.

```
1 ! Parses alphanumeric line from MoM *.4112.txt and extracts electric field values
2 module mClassElectricFields
3
4   use mFormatDescriptors,          only : fmt_stat, fmt_iomsg
5   use mLibraryOfConstants,        only : cZero, MoMlineLength, messageLength
6   use mPrecisionDefinitions,      only : ip, rp
7
8   implicit none
```

```

9
10 integer ( ip ) :: left = 0, right = 0
11 integer :: io_stat = 0
12 character ( len = messageLength ) :: io_msg = ""
13 character ( len = 15 ) :: number = ""
14
15 ! theta = azimuth
16 ! phi = elevation (North Pole = 0, equator = 90)
17 type :: electricFields
18     real ( rp ) :: meanTotalRCS = 0.0_rp
19     real ( rp ) :: dBsm = 0.0_rp
20     real ( rp ) :: theta = 0.0_rp, phi = 0.0_rp
21     complex ( rp ) :: thetaTheta = cZero, thetaPhi = cZero, phiTheta = cZero, phiPhi = cZero
22 contains
23     procedure, public :: gather_mean_total_rcs => gather_mean_total_rcs_sub
24 end type electricFields
25
26 private :: gather_mean_total_rcs_sub
27 private :: compute_mean_total_rcs_sub, compute_dbsm_sub, extract_electric_fields_sub
28 private :: gather_complex_field_sub, gather_real_field_sub
29
30 ! parameters
31 integer ( ip ), parameter :: mll = MomlineLength
32 ! finger print of data line: start and stop positions for each numerical field
33 ! load matrix as columns
34 ! sample data line:
35 ! 90.0000, 0.0000,(-0.4572920E+05, 0.8350829E+05),( 0.2034567E+06,-0.9493007E+05),( 0.2034813E+06,-0.9492184E+05),( -0.1727375E+06, 0.3787291E+05)
36 integer, parameter :: endpoints ( 1 : 10, 1 : 2 ) = &
37     reshape ( [ [ 1, 14, 28, 44, 62, 78, 96, 112, 130, 146 ], &
38     [ 12, 26, 42, 58, 76, 92, 110, 126, 144, 160 ] ], [ 10, 2 ] )
39 ! constructor
40 type ( electricFields ), parameter :: electricFields0 = &
41     electricFields ( meanTotalRCS = 0.0, theta = 0.0, phi = 0.0, &
42     thetaTheta = cZero, thetaPhi = cZero, phiTheta = cZero, phiPhi = cZero )
43 contains
44
45 ! master routine: only exposed procedure
46 subroutine gather_mean_total_rcs_sub ( me, textLine )
47     class ( electricFields ), target :: me
48     character ( len = mll ), intent ( in ) :: textLine
49     call extract_electric_fields_sub ( me, textLine )
50     call compute_mean_total_rcs_sub ( me )
51     call compute_dbsm_sub ( me )
52     return
53 end subroutine gather_mean_total_rcs_sub
54
55 ! Sciacca prescription
56 subroutine compute_dbsm_sub ( me )
57     class ( electricFields ), target :: me
58     me % dBsm = 10.0_rp * log10 ( me % meanTotalRCS )
59     return
60 end subroutine compute_dbsm_sub
61
62 ! Sciacca prescription
63 subroutine compute_mean_total_rcs_sub ( me )
64     class ( electricFields ), target :: me
65     me % meanTotalRCS = abs ( me % thetaTheta ) + abs ( me % thetaPhi ) &
66     + abs ( me % phiTheta ) + abs ( me % phiPhi )
67     me % meanTotalRCS = me % meanTotalRCS / real ( 2, kind = rp )
68     return
69 end subroutine compute_mean_total_rcs_sub
70
71 subroutine extract_electric_fields_sub ( me, textLine )
72     class ( electricFields ), target :: me
73     character ( len = mll ), intent ( in ) :: textLine
74     integer ( ip ) :: position = 0
75     ! move across text line gathering numeric values
76     position = 1
77     call gather_real_field_sub &
78     ( position = position, real_value = me % theta, textLine = textLine, fmt = "( f12.4 )" )
79     call gather_real_field_sub &
80     ( position = position, real_value = me % phi, textLine = textLine, fmt = "( f12.4 )" )
81     call gather_complex_field_sub &
82     ( position = position, complex_value = me % thetaTheta, textLine = textLine )
83     call gather_complex_field_sub &
84     ( position = position, complex_value = me % thetaPhi, textLine = textLine )
85     call gather_complex_field_sub &
86     ( position = position, complex_value = me % phiTheta, textLine = textLine )
87     call gather_complex_field_sub &
88     ( position = position, complex_value = me % phiPhi, textLine = textLine )
89     return
90 end subroutine extract_electric_fields_sub
91
92 subroutine gather_real_field_sub ( position, real_value, textLine, fmt )
93     real ( rp ), intent ( out ) :: real_value
94     integer ( ip ), intent ( inout ) :: position
95     character ( len = mll ), intent ( in ) :: textLine
96     character ( len = 9 ), intent ( in ) :: fmt
97     left = endpoints ( position, 1 )
98     right = endpoints ( position, 2 )
99     write ( number, fmt = 100 ) textLine ( left : right )
100     if ( io_stat /= 0 ) then

```

```

101         write ( *, fmt = ' ( 3g0 ) ' ) "Failure to WRITE string value '", trim ( textLine ( left : right ) ) , "'."
102         write ( *, fmt = fmt_stat ) io_stat
103         write ( *, fmt = fmt_iomsg ) trim ( io_msg )
104         stop "Error occured in module 'mClassElectricFields', subroutine 'gather_real_field_sub'."
105     end if
106     read ( number, fmt = fmt ) real_value
107     if ( io_stat /= 0 ) then
108         write ( *, fmt = ' ( 5g0 ) ' ) "Failure to READ string value '", trim ( textLine ( left : right ) ) , &
109             " as a REAL number using format descriptor ", fmt, "."
110         write ( *, fmt = fmt_stat ) io_stat
111         write ( *, fmt = fmt_iomsg ) trim ( io_msg )
112         stop "Error occured in module 'mClassElectricFields', subroutine 'gather_real_field_sub'."
113     end if
114     position = position + 1
115     return
116     100 format ( g0 )
117 end subroutine gather_real_field_sub
118
119 subroutine gather_complex_field_sub ( position, complex_value, textLine )
120     complex ( rp ),          intent ( out ) :: complex_value
121     integer ( ip ),          intent ( inout ) :: position
122     character ( len = mll ), intent ( in ) :: textLine
123     real ( rp ) :: x = 0.0_rp, y = 0.0_rp
124     call gather_real_field_sub ( position = position, real_value = x, textLine = textLine, fmt = " ( e15.7 )" )
125     call gather_real_field_sub ( position = position, real_value = y, textLine = textLine, fmt = " ( e15.7 )" )
126     complex_value = cmplx ( x, y )
127     return
128 end subroutine gather_complex_field_sub
129
130 end module mClassElectricFields

```

### C.1.3 Class Data File: m-class-data-file.f08

```

1 module mClassDataFile
2
3     use, intrinsic :: iso_fortran_env, only : iostat_end
4     ! classes
5     use mClassAverages,          only : average, average0
6     use mClassElectricFields,    only : electricFields, electricFields0
7     use mClassMesh,             only : meshReal
8     use mAllocations,           only : allocationToolKit, allocationToolKit0
9     use mAllocationsSpecial,     only : allocate_rank_one_averages_sub
10    ! utilities
11    use mLibraryOfConstants,      only : fileNameLength, messageLength, MoMlineLength
12    ! use mBulkRCS,              only : BulkRCS, BulkRCS0
13    use mFileHandling,           only : safeopen_readonly, safeopen_writereplace
14    use mFormatDescriptors,       only : fmt_one, fmt_stat, fmt_iomsg, fmt_shape2
15    use mPrecisionDefinitions,    only : ip, rp
16    use mTextFileUtilities,       only : count_lines_sub, mark_frequencies_sub, read_text_lines_sub, file_closer_sub, &
17    !                             iostat_check_sub
18    ! use mTextFileUtilities,     only : count_lines_sub, file_closer_sub, iostat_check_sub, mark_frequencies_sub, &
19    !                             parse_name_sub, read_text_lines_sub
20    implicit none
21
22    ! parameters
23    integer ( ip ), parameter :: fnl = fileNameLength, msgl = messageLength, mll = MoMlineLength
24    character ( len = 9 ), parameter :: strAzimuth = "azimuth "
25    character ( len = 9 ), parameter :: strElevation = "elevation"
26    character ( len = * ), parameter :: moduleCrash = "Program crashed in module 'mClassDataFile', "
27
28    integer :: io_stat = 0
29    character ( len = msgl ) :: io_msg = ""
30
31    type :: dataFile4112
32        ! rank 2
33        real ( rp ),          allocatable :: rcs_table ( : , : ) ! angle mesh length x nu mesh length
34        real ( rp ),          allocatable :: dbsm_table ( : , : ) ! angle mesh length x nu mesh length
35        ! ! rank 1
36        integer ( ip ),        allocatable :: lineNumbersFrequency ( : )!, &
37        !                             lineNumbersFinished ( : )
38        type ( average ),      allocatable :: perFrequencyAverage ( : ) ! nu mesh length
39        type ( average )      :: globalAverage
40        character ( len = mll ), allocatable :: lines4112Text ( : ) ! length numlines4112Text
41        ! rank 0
42        type ( electricFields ) :: eFields = electricFields0
43        type ( meshReal ) :: meshFrequency, &
44        meshFreeAngle
45        integer ( ip ) :: numFrequencies = 0, &
46        numFixedAngles = 0, &
47        numFreeAngles = 0, &
48        numMeasurements = 0, &
49        numLines4112Text = 0
50        integer ( ip ) :: io_unit = 0
51        character ( len = 9 ) :: angleFixedType = "", angleFreeType = ""
52        character ( len = fnl ) :: file4112Name = "", fileRCStxtName = "", fileRCSbinaryName = "", &
53        fileBsmTxtName = "", fileBsmBinaryName = ""
54        ! allocation tools
55        type ( allocationToolKit ) :: myKit = allocationToolKit0

```

```

56 contains
57     procedure, public :: allocate_rcs_tables          => allocate_rcs_tables_sub, &
58         allocate_rcsAverages                        => allocate_rcsAverages_sub, &
59         characterize_rcs_by_frequency                => characterize_rcs_by_frequency_sub, &
60         check_rcs_table_structure                   => check_rcs_table_structure_sub, &
61         establish_free_angle_mesh                    => establish_free_angle_mesh_sub, &
62         establish_frequency_mesh                     => establish_frequency_mesh_sub, &
63         extract_rcs_from_4112_file                   => extract_rcs_from_4112_file_sub, &
64         harvest_frequencies                          => harvest_frequencies_sub, &
65         set_file_names                              => set_file_names_sub, &
66         set_free_angle_azimuth                      => set_free_angle_azimuth_sub, &
67         set_free_angle_elevation                    => set_free_angle_elevation_sub, &
68         write_rcs_file_set                           => write_rcs_file_set_sub, &
69         write_rcs_binary                             => write_rcs_binary_sub, &
70         write_rcs_csv                                => write_rcs_csv_sub, &
71         write_dBsm_binary                           => write_dBsm_binary_sub, &
72         write_dBsm_csv                               => write_dBsm_csv_sub, &
73         write_summary_by_frequency                  => write_summary_by_frequency_sub, &
74         write_summary_for_all_frequencies            => write_summary_for_all_frequencies_sub
75
76 end type dataFile4112
77
78 private :: allocate_rcs_tables_sub, allocate_rcsAverages_sub, &
79     establish_free_angle_mesh_sub, establish_frequency_mesh_sub, extract_rcs_from_4112_file_sub, &
80     harvest_frequencies_sub, &
81     set_file_names_sub, set_free_angle_azimuth_sub, set_free_angle_elevation_sub, &
82     write_summary_by_frequency_sub, write_summary_for_all_frequencies_sub
83
84 contains
85
86 subroutine characterize_rcs_by_frequency_sub ( me )
87     class ( dataFile4112 ), target :: me
88     type ( average ), pointer :: p => null ( )
89     integer ( ip ) :: kFrequency = 0
90
91     sweep_frequencies: do kFrequency = 1, me % numFrequencies
92         p => me % perFrequencyAverage ( kFrequency )
93         call p % find_max_and_min ( vector = me % rcs_table ( 1 : me % numFreeAngles, kFrequency ) )
94         call p % compute_mean_and_variance ( vector = me % rcs_table ( 1 : me % numFreeAngles, kFrequency ), &
95             one = me % meshFreeAngle % one )
96     end do sweep_frequencies
97
98     return
99 end subroutine characterize_rcs_by_frequency_sub
100
101 module subroutine write_summary_for_all_frequencies_sub ( me )
102     class ( dataFile4112 ), target :: me
103     integer ( ip ) :: kFrequency = 0, first = 0, last = 0, numConvolution = 0
104     real ( rp ), allocatable :: global_rcs ( : ), one ( : )
105     ! allocate memory for all RCS measurements
106     numConvolution = me % numFrequencies * me % numFreeAngles
107     call me % myKit % allocate_rank_one_reals ( real_array = global_rcs, index_min = 1, index_max = numConvolution )
108     call me % myKit % allocate_rank_one_reals ( real_array = one, index_min = 1, index_max = numConvolution )
109     ! load data vector
110     sweep_frequencies: do kFrequency = 1, me % meshFrequency % numMeshElements
111         first = ( kFrequency - 1 ) * me % numFreeAngles + 1
112         last = first + me % numFreeAngles - 1
113         global_rcs ( first : last ) = me % rcs_table ( 1 : me % numFreeAngles, kFrequency )
114     end do sweep_frequencies
115     ! compute extrema
116     one ( : ) = global_rcs ( : ) - global_rcs ( : ) + 1.0_rp
117     call me % globalAverage % find_max_and_min ( vector = global_rcs ( 1 : numConvolution ) )
118     call me % globalAverage % compute_mean_and_variance ( vector = global_rcs ( 1 : numConvolution ), one = one )
119     write ( *, * )
120     write ( *, fmt = 100 ) me % globalAverage % mean, &
121         me % globalAverage % standardDeviation, &
122         me % globalAverage % extrema % minVal, &
123         me % globalAverage % extrema % maxVal
124
125     return
126 100 format ( "Aggregate for all RCS measurements: mean = ", g0, " +/- ", g0, ", min = ", g0, ", max = ", g0 )
127 end subroutine write_summary_for_all_frequencies_sub
128
129 module subroutine write_summary_by_frequency_sub ( me )
130     class ( dataFile4112 ), target :: me
131     integer ( ip ) :: kFrequency = 0
132     write ( *, * )
133     sweep_frequencies: do kFrequency = 1, me % meshFrequency % numMeshElements
134         write ( *, fmt = 100 ) kFrequency, me % meshFrequency % meshValues ( kFrequency ), &
135             me % perFrequencyAverage ( kFrequency ) % mean, &
136             me % perFrequencyAverage ( kFrequency ) % standardDeviation, &
137             me % perFrequencyAverage ( kFrequency ) % extrema % minVal, &
138             me % perFrequencyAverage ( kFrequency ) % extrema % maxVal
139     end do sweep_frequencies
140
141     return
142 100 format ( "I3.3, ". nu = ", g0, ", mean RCS = ", g0, " +/- ", g0, ", min = ", g0, ", max = ", g0 )
143 end subroutine write_summary_by_frequency_sub
144
145 module subroutine write_rcs_file_set_sub ( me )
146     class ( dataFile4112 ), target :: me
147     call me % write_rcs_csv ( )
148     call me % write_rcs_binary ( )
149     call me % write_dBsm_csv ( )

```

```

148         call me % write_dBsm_binary ( )
149     return
150 end subroutine write_rcs_file_set_sub
151
152 module subroutine write_rcs_binary_sub ( me )
153     class ( dataFile4112 ), target :: me
154     integer ( ip ) :: io_rcs = 0
155     character ( len = msg1 ) :: crashChain = ""
156
157     crashChain = moduleCrash // "subroutine 'write_rcs_binary_sub'."
158
159     open ( newunit = io_rcs, file = me % fileRCSbinaryName, action = 'WRITE', status = 'REPLACE', form = 'UNFORMATTED', &
160           iostat = io_stat, iomsg = io_msg )
161     call iostat_check_sub ( action = "UNFORMATTED OPENING", fileName = me % fileRCSbinaryName, crashChain = crashChain, &
162           iostat = io_stat, iomsg = io_msg )
163
164     write ( io_rcs, iostat = io_stat, iomsg = io_msg ) me % rcs_table ( 1 : me % meshFreeAngle % numMeshElements, &
165           1 : me % meshFrequency % numMeshElements )
166     call iostat_check_sub ( action = "UNFORMATTED WRITE to", fileName = me % fileRCSbinaryName, crashChain = crashChain, &
167           iostat = io_stat, iomsg = io_msg )
168     call file_closer_sub ( io_unit = io_rcs, fileName = me % fileRCSbinaryName, crashChain = crashChain )
169
170     return
171 end subroutine write_rcs_binary_sub
172
173 module subroutine write_rcs_csv_sub ( me )
174     class ( dataFile4112 ), target :: me
175     integer ( ip ) :: kFrequency = 0, kFreeAngle = 0, &
176           io_out = 0
177     character ( len = msg1 ) :: crashChain = ""
178
179     crashChain = moduleCrash // "subroutine 'write_rcs_csv_sub'."
180     io_out = safeopen_writereplace ( me % fileRCStxtName )
181     ! write RCS values one row (frequency) at a time
182     sweep_frequencies: do kFrequency = 1, me % meshFrequency % numMeshElements
183         write ( io_out, fmt = me % meshFreeAngle % valuesFormatDescriptor ) ( me % rcs_table ( kFreeAngle, kFrequency ), &
184               kFreeAngle = 1, me % meshFreeAngle % numMeshElements )
185         call iostat_check_sub ( action = "WRITE to", fileName = me % fileRCStxtName, crashChain = crashChain, &
186               iostat = io_stat, iomsg = io_msg )
187     end do sweep_frequencies
188     ! close io handle
189     call file_closer_sub ( io_unit = io_out, fileName = me % fileRCStxtName, crashChain = crashChain )
190
191     return
192 end subroutine write_rcs_csv_sub
193
194 module subroutine write_dBsm_binary_sub ( me )
195     class ( dataFile4112 ), target :: me
196     integer ( ip ) :: io_rcs = 0
197     character ( len = msg1 ) :: crashChain = ""
198
199     crashChain = moduleCrash // "subroutine 'write_dBsm_binary_sub'."
200
201     open ( newunit = io_rcs, file = me % filedBsmBinaryName, action = 'WRITE', status = 'REPLACE', form = 'UNFORMATTED', &
202           iostat = io_stat, iomsg = io_msg )
203     call iostat_check_sub ( action = "UNFORMATTED OPENING", fileName = me % fileRCSbinaryName, crashChain = crashChain, &
204           iostat = io_stat, iomsg = io_msg )
205
206     write ( io_rcs, iostat = io_stat, iomsg = io_msg ) me % dBsm_table ( 1 : me % meshFreeAngle % numMeshElements, &
207           1 : me % meshFrequency % numMeshElements )
208     call iostat_check_sub ( action = "UNFORMATTED WRITE to", fileName = me % fileRCSbinaryName, crashChain = crashChain, &
209           iostat = io_stat, iomsg = io_msg )
210     call file_closer_sub ( io_unit = io_rcs, fileName = me % filedBsmBinaryName, crashChain = crashChain )
211
212     return
213 end subroutine write_dBsm_binary_sub
214
215 module subroutine write_dBsm_csv_sub ( me )
216     class ( dataFile4112 ), target :: me
217     integer ( ip ) :: kFrequency = 0, kFreeAngle = 0, &
218           io_out = 0
219     character ( len = msg1 ) :: crashChain = ""
220
221     crashChain = moduleCrash // "subroutine 'write_dBsm_csv_sub'."
222     io_out = safeopen_writereplace ( me % filedBsmTxtName )
223     ! write RCS values one row (frequency) at a time
224     sweep_frequencies: do kFrequency = 1, me % meshFrequency % numMeshElements
225         write ( io_out, fmt = me % meshFreeAngle % valuesFormatDescriptor ) ( me % dBsm_table ( kFreeAngle, kFrequency ), &
226               kFreeAngle = 1, me % meshFreeAngle % numMeshElements )
227         call iostat_check_sub ( action = "WRITE to", fileName = me % filedBsmTxtName, crashChain = crashChain, &
228               iostat = io_stat, iomsg = io_msg )
229     end do sweep_frequencies
230     ! close io handle
231     call file_closer_sub ( io_unit = io_out, fileName = me % fileRCStxtName, crashChain = crashChain )
232
233     return
234 end subroutine write_dBsm_csv_sub
235
236 subroutine set_file_names_sub ( me, file4112Name )
237     class ( dataFile4112 ), target :: me
238     character ( len = fnl ), intent ( in ) :: file4112Name
239     integer ( ip ) :: nameLength = 0

```

```

240
241     nameLength = len ( trim ( file4112Name ) )
242     me % file4112Name = trim ( file4112Name )
243     me % fileRCStxtName = trim ( file4112Name ( 1 : nameLength - 4 ) ) // ".rcs.txt"
244     me % fileRCBsmBinaryName = trim ( file4112Name ( 1 : nameLength - 4 ) ) // ".rcs.r32"
245     me % filedBsmTxtName = trim ( file4112Name ( 1 : nameLength - 4 ) ) // ".dBsm.txt"
246     me % filedBsmBinaryName = trim ( file4112Name ( 1 : nameLength - 4 ) ) // ".dBsm.r32"
247
248     return
249 end subroutine set_file_names_sub
250
251 subroutine allocate_rcsAverages_sub ( me )
252     class ( dataFile4112 ), target :: me
253     call allocate_rank_one_averages_sub ( rank_1_average = me % perFrequencyAverage, &
254                                         index_min = 1, index_max = me % numFrequencies )
255     return
256 end subroutine allocate_rcsAverages_sub
257
258 subroutine allocate_rcs_tables_sub ( me )
259     class ( dataFile4112 ), target :: me
260     call me % myKit % allocate_rank_two_reals ( rank_2_real_array = me % rcs_table, &
261                                               dim1_index_min = 1, dim1_index_max = me % numFreeAngles, &
262                                               dim2_index_min = 1, dim2_index_max = me % numFrequencies )
263     call me % myKit % allocate_rank_two_reals ( rank_2_real_array = me % dBsm_table, &
264                                               dim1_index_min = 1, dim1_index_max = me % numFreeAngles, &
265                                               dim2_index_min = 1, dim2_index_max = me % numFrequencies )
266     return
267 end subroutine allocate_rcs_tables_sub
268
269 subroutine establish_frequency_mesh_sub ( me )
270     class ( dataFile4112 ), target :: me
271     ! count lines in MoM file (e.g. 14844)
272     call count_lines_sub ( fullFileName = me % file4112Name, numLines = me % numLines4112Text )
273     ! allocate object to hold text of MoM file as a collection of text lines
274     call me % myKit % allocate_rank_one_characters ( character_array = me % lines4112Text, &
275                                                    index_min = 1, index_max = me % numLines4112Text )
276     ! load MoM text into memory to count frequencies and angles
277     call read_text_lines_sub ( fileName = me % file4112Name, linesText = me % Lines4112Text )
278     ! sift through text lines for " Freq ="
279     call me % harvest_frequencies ( )
280     return
281 end subroutine establish_frequency_mesh_sub
282
283 ! sweep through character array looking for " Freq"
284 ! store these values in a temporary array until numMesh is allocated
285 subroutine harvest_frequencies_sub ( me )
286     class ( dataFile4112 ), target :: me
287     ! pointers
288     ! character ( len = m11 ), pointer :: p => null ( )
289     type ( meshReal ), pointer :: q => null ( )
290     type ( allocationToolkit ), pointer :: s => null ( )
291     ! temp arrays
292     real ( ip ) :: tempFrequencyValues ( 1 : 500 )
293     integer ( ip ) :: tempLineNumsFrequency ( 1 : 500 )
294     ! scalars
295     integer ( ip ) :: numFrequencies = 0, kFrequency = 0
296
297     ! find lines containing " Freq ="
298     call mark_frequencies_sub ( lines4112Text = me % lines4112Text, &
299                               numLines4112Text = me % numLines4112Text, &
300                               tempFrequencyValues = tempFrequencyValues, &
301                               tempLineNumsFrequency = tempLineNumsFrequency, &
302                               numFrequencies = numFrequencies )
303
304     ! record what we have learned about the mesh
305     q => me % meshFrequency
306     q % numMeshElements = numFrequencies
307     ! allocate data objects
308     call q % allocate_mesh_real ( )
309     s => me % myKit
310     call s % allocate_rank_one_integers ( integer_array = me % lineNumbersFrequency, index_min = 1, &
311                                          index_max = q % numMeshElements )
312     s => null ( )
313     ! move temporary array data into data object
314     do kFrequency = 1, q % numMeshElements
315         q % meshValues ( kFrequency ) = tempFrequencyValues ( kFrequency )
316         me % lineNumbersFrequency ( kFrequency ) = tempLineNumsFrequency ( kFrequency )
317     end do
318     me % numFrequencies = q % numMeshElements
319     call q % analyze_mesh_values ( )
320     q => null ( )
321     return
322 end subroutine harvest_frequencies_sub
323
324 subroutine extract_rcs_from_4112_file_sub ( me )
325     class ( dataFile4112 ), target :: me
326     ! locals
327     lreal ( rp ) :: sigma = 0.0_rp
328     integer ( ip ) :: kFrequency = 0, kFreeAngle = 0, linePosition = 0
329     character ( len = m11 ) :: textLine
330
331     ! open *.4112.txt file, read text lines into memory

```

```

332         call read_text_lines_sub ( fileName = me % file4112Name, linesText = me % lines4112Text )
333         ! sweep and harvest RCS value
334         sweep_frequencies: do kFrequency = 1, me % numFrequencies
335             linePosition = me % lineNumbersFrequency ( kFrequency ) + 8
336             sweep_free_angles: do kFreeAngle = 1, me % numFreeAngles
337                 textLine = me % lines4112Text ( linePosition )
338                 call me % eFields % gather_mean_total_rcs ( textLine = textLine )
339                 me % rcs_table ( kFreeAngle, kFrequency ) = me % eFields % meanTotalRCS
340                 me % dBsm_table ( kFreeAngle, kFrequency ) = me % eFields % dBsm
341                 linePosition = linePosition + 1
342             end do sweep_free_angles
343         end do sweep_frequencies
344
345         return
346     end subroutine extract_rcs_from_4112_file_sub
347
348     subroutine set_free_angle_elevation_sub ( me )
349         class ( dataFile4112 ), target :: me
350         me % angleFreeType = strElevation
351         me % angleFixedType = strAzimuth
352         return
353     end subroutine set_free_angle_elevation_sub
354
355     subroutine set_free_angle_azimuth_sub ( me )
356         class ( dataFile4112 ), target :: me
357         me % angleFreeType = strAzimuth
358         me % angleFixedType = strElevation
359         return
360     end subroutine set_free_angle_azimuth_sub
361
362     subroutine establish_free_angle_mesh_sub ( me, angle_min, angle_max, angle_count )
363         class ( dataFile4112 ), target :: me
364         real ( rp ), intent ( in ) :: angle_min, angle_max
365         integer ( ip ), intent ( in ) :: angle_count
366
367         me % meshFreeAngle % meshAverage % extrema % minVal = angle_min
368         me % meshFreeAngle % meshAverage % extrema % maxVal = angle_max
369         me % meshFreeAngle % numMeshElements = angle_count
370         me % numFreeAngles = angle_count
371
372         call me % meshFreeAngle % allocate_mesh_real ( )
373         call me % meshFreeAngle % compute_real_mesh_length ( )
374         call me % meshFreeAngle % compute_real_mesh_interval ( )
375         call me % meshFreeAngle % populate_real_mesh ( )
376         call me % meshFreeAngle % populate_integer_mesh ( )
377
378         return
379     end subroutine establish_free_angle_mesh_sub
380
381     subroutine check_rcs_table_structure_sub ( me )
382         class ( dataFile4112 ), target :: me
383         write ( *, * )
384         write ( *, fmt = '( g0 )' ) "# # Dimensions for RCS data container # #"
385         write ( *, * )
386         write ( *, fmt = '( g0 )' ) "# Expected dimensions:"
387         write ( *, fmt = '( 2g0 )' ) "# Number of radar frequencies scanned by MoM: ", me % numFrequencies
388         write ( *, fmt = '( 4g0 )' ) "# Number of ", me % angleFreeType, " angles scanned by MoM: ", me % numFreeAngles
389         write ( *, * )
390         write ( *, fmt = '( 2g0 )' ) "# Container MoM 4112.txt file: rcs_table"
391         write ( *, fmt = '( 6g0 )' ) "# Free angle dimension = ", size ( me % rcs_table, 1 ), &
392             " indices run from ", lbound ( me % rcs_table, 1 ), &
393             " to ", ubound ( me % rcs_table, 1 )
394         write ( *, fmt = '( 6g0 )' ) "# Frequency dimension = ", size ( me % rcs_table, 2 ), &
395             " indices run from ", lbound ( me % rcs_table, 2 ), &
396             " to ", ubound ( me % rcs_table, 2 )
397         write ( *, * )
398         return
399     end subroutine check_rcs_table_structure_sub
400
401 end module mClassDataFile

```

## References

1. Barton, D. K. and H.R. Ward (1969). *Handbook of Radar Measurement*. New York, NY: Penguin Random House.
2. Bruder, J et al. (2003). "IEEE standard for letter designations for radar-frequency bands". In: *IEEE Aerospace & Electronic Systems Society*, pp. 1–3.
3. Crispin, JW Jr (2013). *Methods of radar cross-section analysis*. Elsevier.

4. Fuhs, Allen E (1982). *Radar cross section lectures*. Monterey, California, Naval Postgraduate School. URL: <https://calhoun.nps.edu/server/api/core/bitstreams/9e69ec48-4628-4243-9f9b-7e879521f7f8/content>.
5. Gibson, Walton C (2021). *The method of moments in electromagnetics*. Chapman and Hall/CRC.
6. Harrington, Roger F (1987). “The method of moments in electromagnetics”. In: *Journal of Electromagnetic waves and Applications* 1.3, pp. 181–200.
7. Knott, Eugene F, John F Schaeffer, and Michael T Tulley (2004). *Radar cross section*. SciTech Publishing.
8. Kolosov, Andrei A. (1987). *Over the Horizon Radar*. Artech House. ISBN: 9780890062333. URL: <https://us.artechhouse.com/Over-the-Horizon-Radar-P254.aspx>.
9. Lu, Cai-Cheng and Chong Luo (2003). “Comparison of iteration convergences of SIE and VSIE for solving electromagnetic scattering problems for coated objects”. In: *Radio Science* 38.2, pp. 11–1.
10. Madheswaran, M and P Suresh Kumar (2012). “Estimation of wide band radar cross section (RCS) of regular shaped objects using method of moments (MOM)”. In: *Ictact Journal on Communication Technology* 3.2, pp. 536–541.
11. Peebles, Peyton Z (2007). *Radar principles*. John Wiley & Sons.
12. Topa, Daniel (Apr. 2020a). *Mercury Method of Moments Adjunct Visualization Tool: Trials and Tribulations*. Tech. rep. ARFL/RVB.
13. — (2020b). *Mercury Method of Moments: AFRL Quick Start Guide*. Tech. rep. AFRL.
14. — (Mar. 2020c). *Radar Cross Section Models for AFCAP Dashboard: Rapid Report 2020-02: Corrected*. Briefing.
15. — (Apr. 2020d). *Radar Cross Section: Phase 1 Summary Report*. Tech. rep. ARFL/RVB.
16. Yuan, Jiade, Changqing Gu, and Guodong Han (2009). “Efficient generation of method of moments matrices using equivalent dipole-moment method”. In: *IEEE Antennas and Wireless Propagation Letters* 8, pp. 716–719.