



---

**EUFAR**

**Standards and Protocols**

# **EGADS Algorithm Handbook**

**Version 0.8.7**

Last Updated: January 3, 2018



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>I</b>	<b>General Algorithms</b>	<b>2</b>
<b>2</b>	<b>Mathematics</b>	<b>3</b>
2.1	Time Derivative . . . . .	4
<b>3</b>	<b>Corrections</b>	<b>5</b>
3.1	Simple correction of spikes . . . . .	6
<b>4</b>	<b>Transforms</b>	<b>7</b>
4.1	Linear Interpolation . . . . .	8
4.2	Linear Interpolation (old) . . . . .	9
4.3	Convert ISO 8601 time to date/time elements . . . . .	10
4.4	Convert ISO 8601 time string to seconds . . . . .	11
4.5	Convert elapsed seconds to ISO 8601 time string . . . . .	12
4.6	Converts a time or a time vector to decimal year. . . . .	13
<b>II</b>	<b>Atmospheric Algorithms</b>	<b>14</b>
<b>5</b>	<b>Thermodynamics</b>	<b>15</b>
5.1	Incremental pressure altitude . . . . .	16
5.2	Pressure altitude . . . . .	17
5.3	Density of dry air . . . . .	18
5.4	Relative humidity from capacitive probe . . . . .	19
5.5	Pressure and angle of incidence (CNRM) . . . . .	20
5.6	Dynamic pressure and angle of incidence . . . . .	21
5.7	Potential Temperature . . . . .	23
5.8	Static Temperature . . . . .	24
5.9	Virtual Temperature . . . . .	25
5.10	Mach number . . . . .	26
5.11	True air speed (CNRM) . . . . .	27
5.12	True air speed (RAF) . . . . .	28
5.13	Longitudinal true airspeed . . . . .	29

5.14	3D Wind Vectors . . . . .	30
<b>6</b>	<b>Microphysics</b>	<b>31</b>
6.1	Effective diameter . . . . .	32
6.2	Mean diameter . . . . .	33
6.3	Median Volume Diameter . . . . .	34
6.4	Extinction Coefficient . . . . .	35
6.5	Mass Concentration . . . . .	36
6.6	Total Number Concentration (DMT) . . . . .	37
6.7	Total Number Concentration . . . . .	38
6.8	Sample area for imaging probes (All in) . . . . .	39
6.9	Sample area for imaging probes (Center In) . . . . .	40
6.10	Sample area for scattering probes . . . . .	41
6.11	Sample Volume . . . . .	42
6.12	Surface Area Concentration . . . . .	43
<b>7</b>	<b>Radiation</b>	<b>44</b>
7.1	Camera Viewing Angles . . . . .	45
7.2	Planck Emission . . . . .	46
7.3	Rotate solar vector to aircraft frame . . . . .	47
7.4	Scattering Angles . . . . .	49
7.5	Solar Vector Calculation (Blanco) . . . . .	50
7.6	Solar Vector Calculation (Reda-Andreas) . . . . .	52
7.7	Blackbody Temperature . . . . .	58
<b>8</b>	<b>Quality Control</b>	<b>59</b>
8.1	Check navigation data for inconsistencies . . . . .	60
8.2	Additional consistency check & QA for navigation data (no correction!) . .	62
	<b>Index</b>	<b>64</b>
	<b>Bibliography and references</b>	<b>64</b>

# Chapter 1

## Introduction

This document contains descriptions of algorithms contained in the EGADS toolbox. Within each algorithm description is the following:

- **Algorithm Name** – name of algorithm as implemented in EGADS .
- **Category** – general category of algorithm. Algorithm can be found in this subdirectory in EGADS .
- **Summary** – short description of what the algorithm does.
- **Inputs** – expected inputs to algorithm. This field includes expected units, and data type of input.
- **Outputs** – outputs produced by algorithm.
- **Formula** – description of formulas or methods behind the algorithm.
- **Source** – person, institution or entity who provided the algorithm.
- **References** – any references to literature, journals or documents with more information on the current algorithm

To aid in algorithm usage and discovery, there is a general naming scheme for EGADS algorithms. Generally, algorithm names are composed as follows:

`{measurement}_{context/detail/instrument}_{source}`

For example, an algorithm provided by CNRM to calculate the density of dry air would be named `density_dry_air_cnm`.

For more information about using these algorithms within EGADS , or using EGADS itself, please refer to the EGADS documentation which can be found at <https://github.com/eufarn7sp/egads>

**Part I**

**General Algorithms**

## Chapter 2

# Mathematics

## 2.1 Time Derivative

**Algorithm name:** derivative\_wrt\_time

**Category:** Mathematics

**Summary:** Calculation of the first time derivative of a generic parameter. Calculations of time derivatives are centered for all except the first and last values in the vector. Returns **None** value for scalar parameters.

### Inputs:

$x$	Vector	Parameter to calculate first derivative
$t$	Vector	Time signal [sec]

### Outputs:

$\dot{x}$	Vector	First derivative of $x$ [units of $x$ / sec]
-----------	--------	--

**Formula:**

$$\dot{x}_i = \frac{x_{i+1} - x_{i-1}}{t_{i+1} - t_{i-1}}$$

**Source:**

**References:**



## Chapter 3

## Corrections

### 3.1 Simple correction of spikes

**Algorithm name:** correction\_spike\_simple\_cnrm

**Category:** Corrections

**Summary:** Detection of spikes which exceed a specified threshold. The detected value is replaced with the mean of the surrounding values.

This algorithm does not apply well to variables that are naturally discontinuous.

**Inputs:**

$X$	Vector	Parameter for analysis
$S_0$	Coeff	Spike detection threshold (same units as $X$ , and must be positive)

**Outputs:**

$X_c$	Vector	Parameter with corrections applied
-------	--------	------------------------------------

**Formula:** The  $i$ th term is considered a spike if the following are all true:

$$\|X[i] - X[i - 1]\| > S_0 \quad (3.1)$$

$$\|X[i] - X[i + 1]\| > S_0 \quad (3.2)$$

$$(X[i] - X[i - 1])(X[i] - X[i + 1]) > 0 \quad (3.3)$$

with

$$X_c[i] = \frac{X[i + 1] + X[i - 1]}{2}$$

Otherwise,  $X_c[i] = X[i]$

**Source:** CNRM/GMEI/TRAMM

**References:**

## Chapter 4

# Transforms

## 4.1 Linear Interpolation

**Algorithm name:** interpolate\_linear

**Category:** Transforms

**Summary:** This algorithm linearly interpolates a variable piecewise from one coordinate system to another. It is mostly used to fill gaps.

### Inputs:

$x$	Vector	x-coordinates of the data points (must be increasing).
$f$	Vector	Data points to interpolate.
$x_{interp}$	Vector	New set of x-coordinates to use in interpolation.
$f_{left}$	Coeff, optional	Value to return when $x_{interp} < x_0$ . Default is $f_0$ .
$f_{right}$	Coeff, optional	Value to return when $x_{interp} > x_n$ . Default is $f_n$ .

### Outputs:

$f_{interp}$	Vector	Interpolated values of $f$ .
--------------	--------	------------------------------

**Formula:** For each value of  $x_{interp}$  the two surrounding points are found and designated  $x_a$  and  $x_b$ , with corresponding values  $f_a$  and  $f_b$ . Then  $f_{interp}$  is calculated piecewise as follows:

$$f_{interp}[i] = f_a + (x_{interp}[i] - x_a) \frac{f_b - f_a}{x_b - x_a}$$

Values where  $x_{interp}$  is less than  $x_0$  are replaced with  $f_{left}$ , if provided, or  $f_0$ . Likewise,  $f_{right}$  if given, or  $f_n$  are substituted where  $x_{interp}$  is greater than  $x_n$ .

Important: in the current version of the algorithm, the corresponding  $i^{th}$  value is interpolated only if:

- $x_{interp}[i]$  doesn't exist in  $x$
- $f(x) = NaN$  if  $x_{interp}[i]$  exists in  $x$

**Source:**

**References:**

## 4.2 Linear Interpolation (old)

**Algorithm name:** `interpolate_linear_old`

**Category:** Transforms

**Summary:** This algorithm linearly interpolates a variable piecewise from one coordinate system to another. All values are interpolated, even if they exist in the new coordinate system.

### Inputs:

$x$	Vector	x-coordinates of the data points (must be increasing).
$f$	Vector	Data points to interpolate.
$x_{interp}$	Vector	New set of x-coordinates to use in interpolation.
$f_{left}$	Coeff, optional	Value to return when $x_{interp} < x_0$ . Default is $f_0$ .
$f_{right}$	Coeff, optional	Value to return when $x_{interp} > x_n$ . Default is $f_n$ .

### Outputs:

$f_{interp}$	Vector	Interpolated values of $f$ .
--------------	--------	------------------------------

**Formula:** For each value of  $x_{interp}$  the two surrounding points are found and designated  $x_a$  and  $x_b$ , with corresponding values  $f_a$  and  $f_b$ . Then  $f_{interp}$  is calculated piecewise as follows:

$$f_{interp}[i] = f_a + (x_{interp}[i] - x_a) \frac{f_b - f_a}{x_b - x_a}$$

Values where  $x_{interp}$  is less than  $x_0$  are replaced with  $f_{left}$ , if provided, or  $f_0$ . Likewise,  $f_{right}$  if given, or  $f_n$  are substituted where  $x_{interp}$  is greater than  $x_n$ .

**Source:**

**References:**

### 4.3 Convert ISO 8601 time to date/time elements

**Algorithm name:** `isotime_to_elements`

**Category:** Transforms

**Summary:** This algorithm takes a series of ISO 8601 strings and splits them into their component values (year, month, day, hour, minute, second) using the Python `dateutil` module. This module is format agnostic, and will recognize any ISO 8601 format.

**Inputs:**

---

<i>t<sub>ISO</sub></i>	Vector	ISO 8601 date-time string
------------------------	--------	---------------------------

**Outputs:**

---

<i>year</i>	Vector	year
<i>month</i>	Vector	month
<i>day</i>	Vector	day
<i>hour</i>	Vector	hour
<i>minute</i>	Vector	minute
<i>second</i>	Vector	second

**Formula:** This algorithm applies the Python `dateutil.parser` module to decompose an ISO date-time string into its component values.

**Source:**

**References:**

## 4.4 Convert ISO 8601 time string to seconds

**Algorithm name:** `isotime_to_seconds`

**Category:** Transforms

**Summary:** This algorithm converts a series of ISO 8601 date-time strings to delta time in seconds. It takes an optional format string for the conversion and an optional reference time. If no reference time is provided, then Jan 1, 1970, 00:00:00 is used as the reference.

**Inputs:**

$t_{ISO}$	Vector	ISO 8601 strings
$t_{ISOref}$	String, Optional	Reference time [ISO 8601 string] - default is '19700101T000000'
$format$	String, Optional	ISO 8601 string format - if none provided, alg will attempt to deconstruct time string.

**Outputs:**

$\Delta t$	Vector	Seconds since reference
------------	--------	-------------------------

**Formula:** This algorithm uses the Python `dateutil` and `datetime` modules to parse and process ISO 8601 date strings into seconds elapsed. The basic steps of the algorithms are:

1. Convert from ISO 8601 string into datetime tuple. If no format string is used, the Python function `dateutil.parser.parse` is used to deconstruct the string, since it can automatically recognize nearly any date string format. If a format string is provided, then `datetime.datetime.strptime(string, format)` is used to deconstruct the string.
2. datetime tuple objects are subtracted from the reference time to get a `datetime.timedelta` object.
3. Number of seconds and microseconds are calculated from the `datetime.timedelta` object and stored as numeric objects and passed out of the algorithm.

**Source:**

**References:**

## 4.5 Convert elapsed seconds to ISO 8601 time string

**Algorithm name:** `seconds_to_isotime`

**Category:** Transforms

**Summary:** Given a vector of elapsed seconds and a reference time, this algorithm calculates a series of ISO 8601 formatted time strings using the Python datetime module. The format of the returned ISO 8601 strings can be controlled by the optional *format* parameter. The default format is `yyyymmddTHH-MMss`.

### Inputs:

$t_{secs}$	Vector	Elapsed seconds [s]
$t_{ref}$	String	ISO 8601 reference time
<i>format</i>	String, optional	ISO 8601 format string, default is <code>yyyymmddTHH-MMss</code>

### Outputs:

$t_{ISO}$	Vector	ISO 8601 date-time strings
-----------	--------	----------------------------

**Formula:** The ISO 8601 time strings are generated from the inputs using the Python datetime module using these steps for each item in the  $t_{secs}$  vector:

1. Create a datetime object using the input reference time ( $t_{ref}$ ) representing the start time.
2. Calculate a timedelta object from the input elapsed seconds parameter.
3. Add the timedelta object to the reference datetime object to calculate an absolute time.
4. Convert the resulting datetime object to an ISO 8601 string following the given *format*, if any.

**Source:**

**References:**



## 4.6 Converts a time or a time vector to decimal year.

**Algorithm name:** `time_to_decimal_year`

**Category:** Transforms

**Summary:** Given a vector of time ( $ms/s/mm/h/d/m$ ) and an optional reference year, this algorithm converts the data to a format in decimal year. Ex: 1995.0125

### Inputs:

$t$	Vector	Time [s]
$t_{ref}$	String, optional	Time reference, default is 19500101T000000

### Outputs:

$t_y$	Vector	Time in decimal year [year]
-------	--------	-----------------------------

**Formula:** The decimal year vector  $t_y$  is generated from the inputs using the Python datetime module using these steps for each item in the  $t$  vector:

1. Regardless of the time format (second, minute, hour, day, month, ...),  $t$  is converted to year automatically by the instance EgadsData.
2. The user time reference,  $t_{ref}$ , if provided by the user, is converted to seconds using the algorithm ISOtimeToSeconds, based on the reference 1950-01-01 at 00h00mm00s.  $t_{ref}$  can be positive if the user time reference is after 1950-01-01, or negative if the user time reference is before 1950-01-01.
3. The time reference is then rescaled to year.
4. The final  $t_y$  vector is computed by adding  $t_{ref} + 1950$  to  $t$ .

**Source:**

**References:**

**Part II**

**Atmospheric Algorithms**

## Chapter 5

# Thermodynamics

## 5.1 Incremental pressure altitude

**Algorithm name:** altitude\_pressure\_incremental\_cnrm

**Category:** Thermodynamics

**Summary:** Calculate a pressure altitude incrementally along the trajectory of an aircraft from the Laplace formula ( $Z_2 = Z_1 + R_a/g < T_v > \log(P_1/P_2)$ ).

**Inputs:**

$Ps$	Vector[t]	Static pressure [hPa]
$Tv$	Vector[t]	Virtual temperature [K or °C]
$t$	Vector[t]	Measurement period [s]
$Z_0$	Coeff	Reference altitude at $S_0$ if $S_0$ is provided, can be airport altitude (m) if $S_0$ is not provided and measurements start in airport [m]
$S_0$	Coeff, optional	Reference time, if not provided $S_0 = t[0]$ [s]

**Outputs:**

$alt\_p$	Vector[t]	Pressure altitude [m]
----------	-----------	-----------------------

**Formula:**  $Tv$  is converted to Kelvin if needed, then:

$$Z_{i0} = Z_0 \text{ with } i_0 \text{ such as } ref\_time_{i_0} = S_0$$

$$Z_j = Z_i + \frac{R_a}{g} \cdot \left( \frac{T_{v_j} + T_{v_i}}{2} \right) \cdot \log \left( \frac{P_{s_i}}{P_{s_j}} \right) \text{ with } \begin{cases} i = j + 1 \text{ for } j < i_0 \\ i = j - 1 \text{ for } j > i_0 \end{cases}$$

**Source:** CNRM/GMEI/TRAMM

**References:** Equation of state for a perfect gas, Triplet-Roche [10], page 36.

## 5.2 Pressure altitude

**Algorithm name:** altitude\_pressure\_raf

**Category:** Thermodynamics

**Summary:** Calculates pressure altitude given static pressure using US Standard Atmosphere definitions. Sea level conditions in the US Standard Atmosphere are defined as having a pressure of 1013.25 hPa and a temperature of 15 degC at an altitude of 0m.

**Inputs:**

---

$P_s$	Vector	Static pressure [hPa]
-------	--------	-----------------------

---

**Outputs:**

---

$H$	Vector	Pressure altitude [m]
-----	--------	-----------------------

---

**Formula:** For pressures greater than or equal to 226.3206:

$$H = \frac{T_0}{L} \left[ 1 - \left( \frac{P_s}{P_0} \right)^{\frac{R_a L}{g}} \right]$$

where the lapse rate  $L$  is 0.0065 K/m. For pressures less than 226.3206:

$$H = H_1 + \frac{R_a T_1}{g} \ln \left( \frac{P_1}{P_s} \right)$$

where  $H_1$  is 11000m,  $T_1$  is 216.65 K and  $P_1$  is 226.3206.

**Source:** NCAR EOL-RAF

**References:** US Standard Atmosphere 1976 (NASA-TM-X-74335), 241 pages. <http://ntrs.nasa.gov/archive>

### 5.3 Density of dry air

**Algorithm name:** density\_dry\_air\_cnrm

**Category:** Thermodynamics

**Summary:** Calculates density of dry air given static temperature and pressure.

**Inputs:**

$P_s$	Vector	Static pressure [hPa]
$T_s$	Vector	Static temperature [K or °C]

**Outputs:**

$\rho$	Vector	Density of dry air [kg/m <sup>3</sup> ]
--------	--------	---

**Formula:**

$$\rho = \frac{100P_s}{R_a T_s}$$

with  $R_a = 287.05 \text{ J kg}^{-1} \text{ K}^{-1}$

Density of humid air can be calculated using this same algorithm by using virtual temperature instead of static temperature.

**Source:** CNRM/GMEI/TRAMM

**References:** Equation of state for a perfect gas, Triplet-Roche [10], page 34.

## 5.4 Relative humidity from capacitive probe

**Algorithm name:** hum\_rel\_capacitive\_cnrn

**Category:** Thermodynamics

**Summary:** Calculates relative humidity using the measured frequency from a capacitive probe.

### Inputs:

$Ucapf$	Vector	Output frequency of the capacitive probe [Hz]
$T_s$	Vector	Static temperature [K]
$P_s$	Vector	Static pressure [hPa]
$\Delta P$	Vector	Dynamic pressure [hPa]
$C_t$	Coeff.	Temperature correction coefficient [%°C]
$F_{min}$	Coeff.	Minimal acceptable frequency [Hz]
$C_0$	Coeff.	0th degree calibration coefficient
$C_1$	Coeff.	1st degree calibration coefficient
$C_2$	Coeff.	2nd degree calibration coefficient

### Outputs:

$H_u$	Vector	Relative humidity [%]
-------	--------	-----------------------

**Formula:** If  $Ucapf \leq F_{min}$  then  $Ucapf = F_{min}$

$$H_u = \frac{P_s}{P_s + \Delta P} [C_0 + C_1 Ucapf + C_2 Ucapf^2 + C_t (T_s - 20)]$$

with  $T_s$  in °C and 20 in °C.

**Source:** CNRM/GMEI/TRAMM

**References:** CAM note on humidity instrument measurements. [1]

## 5.5 Pressure and angle of incidence (CNRM)

**Algorithm name:** pressure\_angle\_incidence\_cnrm

**Category:** Thermodynamics

**Summary:** Calculates static pressure and dynamic pressure by correction of static error. Angle of attack and sideslip are calculated from the horizontal and vertical differential pressures.

### Inputs:

$P_{sr}$	Vector	Raw static pressure [hPa]
$\Delta P_r$	Vector	Raw dynamic pressure [hPa]
$\Delta P_h$	Vector	Horizontal differential pressure [hPa]
$\Delta P_v$	Vector	Vertical differential pressure [hPa]
$C_\alpha$	Coeff.[2]	Angle of attack calibration coefficients
$C_\beta$	Coeff.[2]	Slip calibration coefficients
$C_{errstat}$	Coeff.[4]	Static error coefficients

### Outputs:

$P_s$	Vector	Static Pressure [hPa]
$\Delta P$	Vector	Dynamic pressure corrected with static error [hPa]
$\alpha$	Vector	Angle of attack [rad]
$\beta$	Vector	Sideslip [rad]

**Formula:** If  $\Delta P_r > 25\text{hPa}$ :

$$Errstat = C_{errstat}[0] + C_{errstat}[1]\Delta P_r + C_{errstat}[2]\Delta P_r^2 + C_{errstat}[3]\Delta P_r^3$$

otherwise:

$$\begin{aligned}
 Errstat &= \frac{\Delta P_r}{25} \text{ Errstat @ 25 hPa} \\
 P_s &= P_{sr} - Errstat \\
 \Delta P &= \Delta P_r + Errstat \\
 \alpha &= C_\alpha[0] + C_\alpha[1] \frac{\Delta P_v}{\Delta P} \\
 \beta &= C_\beta[0] + C_\beta[1] \frac{\Delta P_h}{\Delta P}
 \end{aligned} \tag{5.1}$$

**Source:** CNRM/GMEI/TRAMM

**References:**



## 5.6 Dynamic pressure and angle of incidence

**Algorithm name:** pressure\_dynamic\_angle\_incidence\_vdk

**Category:** Thermodynamics

**Summary:** This algorithm calculates dynamic pressure and angles of incidence from a 5-hole probe using differences in pressure between the ports. The algorithm requires calibration coefficients which are obtained by a calibration procedure of the probe at predefined airflow angles. See van den Kroonenberg, 2008 [11] for more details on the calibration procedure.

### Inputs:

$\Delta P_t$	Vector	Pressure difference between top port and center port [hPa]
$\Delta P_b$	Vector	Pressure difference between bottom port and center port [hPa]
$\Delta P_l$	Vector	Pressure difference between left port and center port [hPa]
$\Delta P_r$	Vector	Pressure difference between right port and center port [hPa]
$\Delta P_{0s}$	Vector	Pressure difference between center port and static pressure [hPa]
$a_{ij}$	Coeff[11,11]	Angle of attack calibration coefficients
$b_{ij}$	Coeff[11,11]	Sideslip calibration coefficients
$q_{ij}$	Coeff[11,11]	Dynamic pressure calibration coefficients

### Outputs:

$q$	Vector	Dynamic pressure [hPa]
$\alpha$	Vector	Angle of attack [deg]
$\beta$	Vector	Sideslip angle [deg]

**Formula:** Total pressure difference is calculated using pressure differentials from the 5 ports.

$$\Delta P = \left( \frac{1}{125} [(\Delta P_t + \Delta P_r + \Delta P_b + \Delta P_l)^2 + (-4\Delta P_t + \Delta P_r + \Delta P_b + \Delta P_l)^2 + (\Delta P_t - 4\Delta P_r + \Delta P_b + \Delta P_l)^2 + (\Delta P_t + \Delta P_r - 4\Delta P_b + \Delta P_l)^2 + (\Delta P_t + \Delta P_r + \Delta P_b - 4\Delta P_l)^2] \right)^{1/2} + \frac{1}{4}(\Delta P_t + \Delta P_r + \Delta P_b + \Delta P_l)$$

The dimensionless pressure coefficients  $k_\alpha$  and  $k_\beta$  are defined using  $\Delta P$  and the measured differential pressures.

$$k_\alpha = \frac{\Delta P_t - \Delta P_b}{\Delta P}$$

$$k_\beta = \frac{\Delta P_r - \Delta P_l}{\Delta P}$$

These are applied to general calibration polynomial form (11th order) from Bohn and Simon, 1975 [3], where  $m = n = 11$ .

$$\tilde{\alpha} = \sum_{i=0}^m (k_\alpha)^i \left[ \sum_{j=0}^n a_{ij} (k_\beta)^j \right]$$

$$\tilde{\beta} = \sum_{i=0}^m (k_\alpha)^i \left[ \sum_{j=0}^n b_{ij} (k_\beta)^j \right]$$

$$k_q = \sum_{i=0}^m (k_\alpha)^i \left[ \sum_{j=0}^n q_{ij} (k_\beta)^j \right]$$

Finally, the dynamic pressure, angle of attack and sideslip angle can be calculated using these coefficients.

$$q = \Delta P_{0s} + \Delta P k_q$$

$$\alpha = \tilde{\alpha}$$

$$\beta = \arctan \left( \frac{\tan \tilde{\beta}}{\cos \tilde{\alpha}} \right)$$

**Source:**

**References:**

A.C. van der Kroonenberg, et al., “Measuring the Wind Vector Using the Autonomous Mini Aerial Vehicle M<sup>2</sup>AV,” *J. Atmos. Oceanic Technol.*, 25 (2008): 1969-1982. [11]

## 5.7 Potential Temperature

**Algorithm name:** temp\_potential.cnrm

**Category:** Thermodynamics

**Summary:** Calculates potential temperature.

### Inputs:

$T_s$	Vector	Static temperature [K or °C]
$P_s$	Vector	Static pressure [hPa]
$R_a/c_{pa}$	Coeff.	Gas constant of air divided by specific heat of air at constant pressure

### Outputs:

$\theta$	Vector	Potential temperature [same unit as $T_s$ ]
----------	--------	---

**Formula:**

$$\theta = T_s \left( \frac{1000}{P_s} \right)^{R_a/c_{pa}}$$

**Source:** CNRM/GMEI/TRAMM

**References:** Triplet-Roche [10].

## 5.8 Static Temperature

**Algorithm name:** temp\_static\_cnrm

**Category:** Thermodynamics

**Summary:** Calculates static temperature of the air from total temperature. This method applies to probe types such as the Rosemount.

### Inputs:

$T_t$	Vector	Measured total temperature [K]
$\Delta P$	Vector	Dynamic pressure [hPa]
$P_s$	Vector	Static pressure [hPa]
$r_f$	Coeff.	Probe recovery coefficient
$R_a/c_{pa}$	Coeff.	Gas constant of air divided by specific heat of air at constant pressure

### Outputs:

$T_s$	Vector	Static temperature [K]
-------	--------	------------------------

### Formula:

$$T_s = \frac{T_t}{1 + r_f \left( \left( 1 + \frac{\Delta P}{P_s} \right)^{R_a/c_{pa}} - 1 \right)}$$

**Source:** CNRM/GMEI/TRAMM

**References:**

## 5.9 Virtual Temperature

**Algorithm name:** temp\_virtual\_cnm

**Category:** Thermodynamics

**Summary:** Calculates the virtual temperature of air.

**Inputs:**

---

$T_s$	Vector	Static temperature [K or °C]
$r$	Vector	Water vapor mixing ratio [g/kg]

---

**Outputs:**

---

$T_v$	Vector	Virtual temperature [same units as $T_s$ ]
-------	--------	--

---

**Formula:**

$$T_v = T_s \frac{1 + (R_v/R_a)r}{1 + r}$$

where  $R_v/R_a = 1.608$

**Source:** CNRM/GMEI/TRAMM

**References:** Triplet-Roche [10], page 56.

## 5.10 Mach number

**Algorithm name:** velocity\_mach\_raf

**Category:** Thermodynamics

**Summary:** Calculates the mach number based on dynamic and static pressure.

**Inputs:**

---

$\Delta P$	Vector	Dynamic pressure [hPa]
$P_s$	Vector	Static pressure [hPa]

---

**Outputs:**

---

$M$	Vector	Mach number
-----	--------	-------------

---

**Formula:**

$$M = \sqrt{\frac{2}{\gamma - 1} \left[ \left( \frac{\Delta P}{P_s} + 1 \right)^{\frac{\gamma - 1}{\gamma}} - 1 \right]}$$

**Source:** NCAR-EOL

**References:** NCAR-RAF Bulletin #23 [7]

## 5.11 True air speed (CNRM)

**Algorithm name:** velocity\_tas\_cnrm

**Category:** Thermodynamics

**Summary:** Calculates true air speed based on static pressure, static temperature and dynamic pressure using the Barré-St Venant formula.

### Inputs:

$T_s$	Vector	Static temperature [K]
$\Delta P$	Vector	Dynamic pressure [hPa]
$P_s$	Vector	Static pressure [hPa]
$c_{pa}$	Coeff.	Specific heat of air at constant pressure (for dry air 1004 J K <sup>-1</sup> kg <sup>-1</sup> )
$R_a/c_{pa}$	Coeff.	Gas constant of air divided by specific heat of air at constant pressure

### Outputs:

$V_t$	Vector	True air speed [m/s]
-------	--------	----------------------

### Formula:

$$V_t = \sqrt{2c_{pa}T_s \left[ \left( 1 + \frac{\Delta P}{P_s} \right)^{R_a/c_{pa}} - 1 \right]}$$

**Source:** CNRM/GMEI/TRAMM

**References:** NCAR-RAF Bulletin #23 [7], *Mécanique des fluides*, Candel [4]

## 5.12 True air speed (RAF)

**Algorithm name:** velocity\_tas\_raf

**Category:** Thermodynamics

**Summary:** Calculates true air speed based on Mach number, measured temperature and thermometer recovery factor. Typical values of the thermometer recovery factor range from 0.75-0.9 for platinum wire ratiometer (flush bulb type) thermometers, and around 1.0 for TAT type thermometers.

**Inputs:**

$T_r$	Vector	Measured temperature [K]
$M$	Vector	Mach number
$e$	Coeff.	thermometer recovery factor

**Outputs:**

$V_t$	Vector	True air speed [m/s]
-------	--------	----------------------

**Formula:**

$$V_t = \sqrt{\frac{R\gamma T_r M^2}{1 + 0.5(\gamma - 1)eM^2}}$$

where the recovery factor  $e$  can be determined for a thermometer by comparing its measured temperature with the actual total and static temperature.

$$e \equiv \frac{T_r - T_s}{T_t - T_s}$$

**Source:** NCAR-EOL

**References:** NCAR-RAF Bulletin #23 [7]



## 5.13 Longitudinal true airspeed

**Algorithm name:** velocity\_tas\_longitudinal\_cnrm

**Category:** Thermodynamics

**Summary:** Calculates the true air speed along the longitudinal axis of the aircraft.

**Inputs:**

---

$V_t$	Vector	True air speed [m/s]
$\alpha$	Vector	Angle of attack [rad]
$\beta$	Vector	Sideslip angle [rad]

---

**Outputs:**

---

$V_{tx}$	Vector	Longitudinal true air speed [m/s]
----------	--------	-----------------------------------

---

**Formula:**

$$V_{tx} = \frac{V_t}{\sqrt{1 + \tan^2 \alpha + \tan^2 \beta}}$$

**Source:** CNRM/GMEI/TRAMM

**References:** NCAR-RAF Bulletin #23 [7]

## 5.14 3D Wind Vectors

**Algorithm name:** wind\_vector\_3d\_raf

**Category:** Thermodynamics

**Summary:** This algorithm applies vector transformations using aircraft speed, angle of attack and sideslip to calculate the three-dimensional wind vector components.

### Inputs:

$U_a$	Vector	Corrected true air speed [m/s]
$\alpha$	Vector	Aircraft angle of attack [rad]
$\beta$	Vector	Aircraft sideslip [rad]
$u_p$	Vector	Easterly aircraft velocity from INS [m/s]
$v_p$	Vector	Northerly aircraft velocity from INS [m/s]
$w_p$	Vector	Upward aircraft velocity from INS [m/s]
$\phi$	Vector	Roll [rad]
$\theta$	Vector	Pitch [rad]
$\psi$	Vector	True Heading [rad]
$\dot{\theta}$	Vector	Pitch rate [rad/sec]
$\dot{\psi}$	Vector	Yaw rate [rad/sec]
$L$	Vector	Distance separating INS and gust probe along aircraft center line [m]

### Outputs:

$u$	Vector	Easterly wind velocity component [m/s]
$v$	Vector	Northerly wind velocity component [m/s]
$w$	Vector	Upwards wind velocity component (positive up) [m/s]

### Formula:

$$D = \sqrt{(1 + \tan^2 \alpha + \tan^2 \beta)}$$

$$u = -U_a D^{-1} [\sin \psi \cos \theta + \tan \beta (\cos \psi \cos \phi + \sin \psi \sin \theta \sin \phi) + \tan \alpha (\sin \psi \sin \theta \cos \phi - \cos \psi \sin \phi)] + u_p - L(\dot{\theta} \sin \theta \sin \psi - \dot{\psi} \cos \psi \cos \theta)$$

$$v = -U_a D^{-1} [\cos \psi \cos \theta - \tan \beta (\sin \psi \cos \phi - \cos \psi \sin \theta \sin \phi) + \tan \alpha (\cos \psi \sin \theta \cos \phi + \sin \psi \sin \phi)] + v_p - L(\dot{\psi} \sin \psi \cos \theta + \dot{\theta} \cos \psi \sin \theta)$$

$$w = -U_a D^{-1} (\sin \theta - \tan \beta \cos \theta \sin \phi - \tan \alpha \cos \theta \cos \phi) + w_p + L \dot{\theta} \cos \theta$$

**Source:**

**References:** NCAR-RAF Bulletin #23 [7]

## Chapter 6

# Microphysics

## 6.1 Effective diameter

**Algorithm name:** diameter\_effective\_dmt

**Category:** Microphysics

**Summary:** Calculates effective diameter of a size distribution. In general, this definition is only meaningful for water clouds, and another form must be used when in ice clouds.

**Inputs:**

$c_i$	Array[time, bins]	Number concentration of hydrometeors in size category $i$ [ $\text{cm}^{-3}$ ]
$d_i$	Vector[bins]	Average diameter in size category $i$ [ $\mu\text{m}$ ]

**Outputs:**

$D_e$	Vector[time]	Effective diameter [ $\mu\text{m}$ ]
-------	--------------	--------------------------------------

**Formula:**

$$D_e = \frac{3 \sum_{i=1}^m c_i d_i^3}{4 \sum_{i=1}^m c_i d_i^2}$$

**Source:**

**References:** “Data Analysis User’s Guide Chapter I: Single Particle Light Scattering,” Droplet Measurement Technologies, 30. [5]

## 6.2 Mean diameter

**Algorithm name:** diameter\_mean\_raf

**Category:** Microphysics

**Summary:** Calculates the arithmetic average of all particle diameters given in a particle size distribution.

**Inputs:**

---

$n_i$	Array[time, bins]	Number of particles in each channel $i$
$d_i$	Vector[bins]	Channel $i$ size [ $\mu\text{m}$ ]

---

**Outputs:**

---

$\bar{D}$	Vector[time]	Mean diameter [ $\mu\text{m}$ ]
-----------	--------------	---------------------------------

---

**Formula:**

$$\bar{D} = \frac{\sum_i n_i d_i}{N_t}$$

where  $N_t$  is the total number of particles.

**Source:** NCAR-RAF

**References:** NCAR-RAF Bulletin No. 24. [8]

## 6.3 Median Volume Diameter

**Algorithm name:** diameter\_median\_volume\_dmt

**Category:** Microphysics

**Summary:** Calculates the median volume diameter given a size distribution. The median volume diameter is the size of droplet below which 50% of the total water volume resides.

**Inputs:**

$c_i$	Array[time, bins]	Number concentration of hydrometeors in size category $i$ [ $\text{cm}^{-3}$ ]
$d_i$	Vector[bins]	Average diameter of size category $i$ [ $\mu\text{m}$ ]
$s_i$	Array[time, bins], Optional	Shape factor of the hydrometeor of size category $i$ to account for asphericity
$\rho_i$	Vector[bins], Optional	Density of hydrometeor in size category $i$ [ $\text{g cm}^{-3}$ ]. Default is $\rho_w = 1.0 \text{ g cm}^{-3}$

**Outputs:**

$D_{mvd}$	Vector[time]	Median volume diameter [ $\mu\text{m}$ ]
-----------	--------------	--

**Formula:** Step 1: Compute liquid water content

$$W = \frac{\pi}{6} \sum_{i=1}^m c_i d_i^3 \rho_i s_i$$

Step 2: Beginning at the first size channel, calculate the accumulated mass  $S_n = w_1 + w_2 + \dots w_n$  where  $w_1$  is the mass of water in channel 1, and  $w_n$  is the channel where the accumulated mass is greater than or equal to  $0.5W$ , i.e. greater than or equal to 50% of the total LWC.

Step 3: Compute the median volume diameter,  $D_{mvd}$  by interpolating linearly between the channels that bracket where the accumulated mass exceeded the total LWC:

$$D_{mvd} = d_{n-1} + (0.5 - S_{n-1}/S_n)(d_n - d_{n-1})$$

**Source:**

**References:** “Data Analysis User’s Guide Chapter I: Single Particle Light Scattering,” Droplet Measurement Technologies, 33. [5]

## 6.4 Extinction Coefficient

**Algorithm name:** extinction\_coeff\_dmt

**Category:** Microphysics

**Summary:** Calculates extinction coefficient given a particle size distribution.

**Inputs:**

$c_i$	Array[time, bins]	Number concentration of hydrometeors in size category $i$ [ $\text{cm}^{-3}$ ]
$d_i$	Vector[bins]	Average diameter of size category $i$ [ $\mu\text{m}$ ]
$Q_e$	Vector[bins], Optional	Extinction efficiency; default is $Q_e = 2$

**Outputs:**

$B_e$	Vector[time]	Extinction coefficient [ $\text{km}^{-1}$ ]
-------	--------------	---

**Formula:**

$$B_e = \frac{\pi}{4} \sum_{i=1}^m Q_e c_i d_i^2$$

**Source:**

**References:** “Data Analysis User’s Guide Chapter I: Single Particle Light Scattering,” Droplet Measurement Technologies, 30. [5]

## 6.5 Mass Concentration

**Algorithm name:** mass\_conc\_dmt

**Category:** Microphysics

**Summary:** Calculates mass concentration given a size distribution. Can be used to calculate liquid or ice water content depending on the types of hydrometeors being sampled.

### Inputs:

$c_i$	Array[time, bins]	Number concentration of hydrometeors in size category $i$ [ $\text{cm}^{-3}$ ]
$d_i$	Vector[bins]	Average diameter of size category $i$ [ $\mu\text{m}$ ]
$s_i$	Array[time, bins]	Shape factor of the hydrometeor of size category $i$ to account for asphericity
$\rho_i$	Vector[time, bins]	Density of the hydrometeor in size category $i$ [ $\text{g cm}^{-3}$ ]

### Outputs:

$M$	Vector[time]	Mass concentration [ $\text{g cm}^{-3}$ ]
-----	--------------	---

### Formula:

$$M = \frac{\pi}{6} \sum_{i=1}^m s_i \rho_i c_i d_i^3$$

### Source:

**References:** “Data Analysis User’s Guide Chapter I: Single Particle Light Scattering,” Droplet Measurement Technologies, 30. [5]



## 6.6 Total Number Concentration (DMT)

**Algorithm name:** number\_conc\_total\_dmt

**Category:** Microphysics

**Summary:** Calculation of total number concentration given distribution of particle counts from a particle sampling probe.

**Inputs:**

$c_i$	Array[time, bins]	Number concentration of hydrometeors in size category $i$ [ $\text{cm}^{-3}$ ]
-------	-------------------	--

**Outputs:**

$N$	Vector[time]	Total number concentration [ $\text{cm}^{-3}$ ]
-----	--------------	---

**Formula:**

$$N = \sum_{i=1}^m c_i$$

**Source:**

**References:** “Data Analysis User’s Guide Chapter I: Single Particle Light Scattering,” Droplet Measurement Technologies, 30. [5]

## 6.7 Total Number Concentration

**Algorithm name:** number\_conc\_total\_raf

**Category:** Microphysics

**Summary:** Calculation of total number concentration for a particle probe.

**Inputs:**

$n_i$	Array	Number of particles in each channel $i$
$SV$	Array	Sample volume [m <sup>3</sup> ]

**Outputs:**

$N_t$	Vector	Total number concentration [m <sup>-3</sup> ]
-------	--------	---

**Formula:**

$$N_t = \sum_i \frac{n_i}{SV_i}$$

**Source:** NCAR-RAF

**References:** NCAR-RAF Bulletin No. 24. [8]

## 6.8 Sample area for imaging probes (All in)

**Algorithm name:** sample\_area\_oap\_all\_in\_raf

**Category:** Microphysics

**Summary:** Calculation of 'all in' sample area size for OAP probes such as the 2DC, 2DP, CIP, etc. This sample area varies by number of shadowed diodes. This routine calculates a sample area per bin.

### Inputs:

$\lambda$	Coeff.	Laser wavelength [nm]
$D_{arms}$	Coeff.	Distance between probe arm tips [mm]
dD	Coeff.	Diode diameter [ $\mu\text{m}$ ]
M	Coeff.	Probe magnification factor
N	Coeff.	Number of diodes in array

### Outputs:

SA	Vector	Sample area [ $\text{m}^2$ ]
----	--------	------------------------------

### Formula:

$$DOF_i = \frac{6R_i^2}{\lambda} \quad (6.1)$$

$$R_i = i \frac{dD}{2}$$

$$X = 1 \dots N - 1$$

where  $DOF$  must be less than  $D_{arms}$ . The parameter  $i$  ranges from 1 to  $N - 1$ , since particles touching either edge are rejected as they are not considered 'all-in'.

$$ESW_i = \frac{dD(N - X_i - 1)}{M}$$

A value for  $ESW_i$  (effective sample width) is calculated for each  $X$ .

$$SA_i = (DOF_i)(ESW_i)$$

**Source:** NCAR-RAF

**References:** NCAR-RAF Bulletin No. 24. [8]

## 6.9 Sample area for imaging probes (Center In)

**Algorithm name:** sample\_area\_oap\_center\_in\_raf

**Category:** Microphysics

**Summary:** Calculation of 'center in' sample area size for OAP probes such as the 2DC, 2DP, CIP, etc. This sample area varies by number of shadowed diodes. This routine is intended to calculate a sample area per bin.

### Inputs:

$\lambda$	Coeff.	Laser wavelength [nm]
$D_{arms}$	Coeff.	Distance between probe arm tips [mm]
dD	Coeff.	Diode diameter [ $\mu\text{m}$ ]
M	Coeff.	Probe magnification factor
N	Coeff.	Number of diodes in array

### Outputs:

SA	Vector	Sample area [ $\text{m}^2$ ]
----	--------	------------------------------

### Formula:

$$\begin{aligned}
 DOF_i &= \frac{6R_i^2}{\lambda} \\
 R_i &= X \frac{dD}{2} \\
 X &= 1 \dots N
 \end{aligned} \tag{6.2}$$

where  $DOF$  must be less than  $D_{arms}$ . The parameter  $i$  ranges from 1 to  $N$ .

$$ESW = \frac{NdD}{M}$$

$$SA_i = (DOF_i)(ESW)$$

**Source:** NCAR-RAF

**References:** NCAR-RAF Bulletin No. 24. [8]

## 6.10 Sample area for scattering probes

**Algorithm name:** sample\_area\_scattering\_raf

**Category:** Microphysics

**Summary:** Calculation of sample area for scattering probes such as the FSSP, CAS, etc.

**Inputs:**

---

DOF	Coeff.	Depth of field [m]
BD	Coeff.	Beam diameter [m]

**Outputs:**

---

SA	Coeff.	Sample area [m <sup>2</sup> ]
----	--------	-------------------------------

**Formula:**

$$SA = (DOF)(BD)$$

**Source:** NCAR-RAF

**References:** NCAR-RAF Bulletin No. 24. [8]

## 6.11 Sample Volume

**Algorithm name:** sample\_volume\_general\_raf

**Category:** Microphysics

**Summary:** Calculates sample volume for microphysics probes (1D, 2D, FSSP, etc).

**Inputs:**

$V_t$	Vector	True air speed [m/s]
SA	Coeff.	Sample area of probe [m <sup>2</sup> ]
$t_s$	Coeff.	Sample rate [s]

**Outputs:**

SV	Vector	Sample volume [m <sup>3</sup> ]
----	--------	---------------------------------

**Formula:**

$$SV = V_t t_s SA$$

**Source:** NCAR-RAF

**References:** NCAR-RAF Bulletin No. 24. [8]

## 6.12 Surface Area Concentration

**Algorithm name:** surface\_area\_conc\_dmt

**Category:** Microphysics

**Summary:** Calculation of surface area concentration given size distribution from particle probe.

**Inputs:**

$c_i$	Array[time, bins]	Number concentration of hydrometeors in size category $i$ [ $\text{cm}^{-3}$ ]
$d_i$	Vector[bins]	Average diameter of size category $i$ [ $\mu\text{m}$ ]
$s_i$	Array[time, bins]	Shape factor of hydrometeor in size category $i$ , to account for asphericity

**Outputs:**

$S$	Vector[time]	Surface area concentration [ $\mu\text{m}^2 \text{cm}^{-3}$ ]
-----	--------------	---

**Formula:**

$$S = \pi \sum_{i=1}^m s_i c_i d_i^2$$

**Source:**

**References:** “Data Analysis User’s Guide Chapter I: Single Particle Light Scattering,” Droplet Measurement Technologies, 30. [5]

## Chapter 7

# Radiation



## 7.1 Camera Viewing Angles

**Algorithm name:** camera\_viewing\_angles

**Category:** Radiation

**Summary:** Calculates per-pixel camera viewing angles of a digital camera given its sensor dimension and focal length. x-y coordinates are defined as having the left side of the image (x=0) aligned with the flight direction and y=0 to the top of the image.

### Inputs:

$n_x$	Coeff	Number of pixels in x direction
$n_y$	Coeff	Number of pixels in y direction
$l_x$	Coeff	Length of the camera sensor in x direction [mm]
$l_y$	Coeff	Length of the cameras sensor in y direction [mm]
$f$	Coeff	Focal length of the camera lens [mm]

### Outputs:

$\theta_c$	Array[ $n_x, n_y$ ]	Camera viewing zenith angle [deg]
$\Phi_c$	Array[ $n_x, n_y$ ]	Camera viewing azimuth angle [deg], mathematic negative system with 0° into flight direction, clockwise

### Formula:

For each  $i, j$  where  $0 < i < n_x$  and  $0 < j < n_y$ :

$$x = l_x \frac{(i - n_x/2)}{n_x}$$

$$y = l_y \frac{(j - n_y/2)}{n_y}$$

$$d = \sqrt{x^2 + y^2}$$

$$\theta_c(i, j) = 2 \tan^{-1} \frac{d}{2f}$$

$$\Phi_c(i, j) = 2\pi - \tan^{-1} \frac{y}{x}$$

**Source:** Andre Ehrlich, Leipzig Institute for Meteorology (a.ehrlich@uni-leipzig.de)

**References:**

## 7.2 Planck Emission

**Algorithm name:** planck\_emission

**Category:** Radiation

**Summary:** Calculates the Planck emission of a surface at a given wavelength given its temperature.

**Inputs:**

$T$	Vector	Temperature [K]
$\lambda$	Coeff	Wavelength [nm]

**Outputs:**

$rad$	Vector	Black body radiance [W m <sup>-2</sup> sr <sup>-1</sup> nm <sup>-1</sup> ]
-------	--------	--

**Formula:** After converting  $\lambda$  to meters, the radiance is calculated by:

$$rad = \frac{2hc^2}{\lambda^5 (\exp(\frac{hc}{k_B \lambda T}) - 1)} * 10^{-9}$$

where  $c$  is the speed of light in m/s,  $h$  is the Planck constant in J s and  $k_B$  is the Boltzmann constant in J/K.

**Source:** Andre Ehrlich, Leipzig Institute for Meteorology (a.ehrlich@uni-leipzig.de)

**References:**

### 7.3 Rotate solar vector to aircraft frame

**Algorithm name:** rotate\_solar\_vector\_to\_aircraft\_frame

**Category:** Radiation

**Summary:** Rotates solar vector to aircraft coordinates given roll, pitch and yaw. All rotations are defined with a mathematically positive spherical coordinate system.

**Inputs:**

$\theta_{\odot}$	Vector	Solar Zenith [degrees]
$\Phi_{\odot}$	Vector	Solar Azimuth [degrees] (mathematic negative, North=0°, clockwise)
$\phi_a$	Vector	Aircraft roll angle [degrees] (mathematic positive, left wing up=positive)
$\theta_a$	Vector	Aircraft pitch angle [degrees] (mathematic positive, nose down=positive)
$\psi_a$	Vector	Aircraft yaw angle [degrees] (mathematic negative, North=0°, clockwise)

**Outputs:**

$\theta_{\odot a}$	Vector	Solar Zenith, AC coordinates [degrees]
$\Phi_{\odot a}$	Vector	Solar Azimuth, AC coordinates [degrees] (mathematic negative, North=0°, clockwise)

**Formula:** First,  $\Phi_{\odot}$  and  $\psi_a$  must be transformed into mathematically positive coordinate systems by subtracting them from 360.

Next, the cartesian coordinates are calculated from the solar vector:

$$\begin{aligned} x &= \sin \theta_{\odot} \cos \Phi_{\odot} \\ y &= \sin \theta_{\odot} \sin \Phi_{\odot} \\ z &= \cos \theta_{\odot} \end{aligned}$$

Then, the cartesian coordinates are rotated using three rotation matrixes using yaw, pitch and roll:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos \theta_a \cos \psi_a & \cos \theta_a \sin \psi_a & -\sin \theta_a \\ \sin \phi_a \sin \theta_a \cos \psi_a - \cos \phi_a \sin \psi_a & \sin \phi_a \sin \theta_a \sin \psi_a + \cos \phi_a \cos \psi_a & \sin \phi_a \cos \theta_a \\ \cos \phi_a \sin \theta_a \cos \psi_a + \sin \phi_a \sin \psi_a & \cos \phi_a \sin \theta_a \sin \psi_a - \sin \phi_a \cos \psi_a & \cos \phi_a \cos \theta_a \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Finally, convert back to spherical coordinates:

$$\begin{aligned} \theta_{\odot a} &= \cos^{-1} \frac{z'}{\sqrt{x'^2 + y'^2 + z'^2}} \\ \Phi_{\odot a} &= \tan^{-1} \frac{y'}{x'} \end{aligned}$$

$\Phi_{\odot a}$  must be between 0 and 360 and then converted back to mathematic negative coordinate system (i.e. subtract it from 360).

**Source:** Andre Ehrlich, Leipzig Institute for Meteorology (a.ehrlich@uni-leipzig.de)

**References:**

## 7.4 Scattering Angles

**Algorithm name:** scattering\_angles

**Category:** Radiation

**Summary:** Calculates the scattering angle for each pixel of an image given the camera viewing angle and solar vector.

### Inputs:

$n_x$	Coeff	Number of pixels in x dimension
$n_y$	Coeff	Number of pixels in y dimension
$\theta_c$	Array[ $n_x, n_y$ ]	Camera viewing zenith angle [degrees]
$\Phi_c$	Array[ $n_x, n_y$ ]	Camera viewing azimuth angle [degrees] ( $0^\circ$ = flight direction)
$\theta_\odot$	Coeff	Solar zenith angle [degrees]
$\Phi_\odot$	Coeff	Solar azimuth angle [degrees] ( $0^\circ$ = North)

### Outputs:

$\theta_{scat}$	Array[ $n_x, n_y$ ]	Scattering angles of each pixel [degrees]
-----------------	---------------------	---

### Formula:

$$\theta_{scat} = \cos^{-1}(-\sin \theta_\odot \cos \Phi_\odot \sin \theta_c \cos \Phi_c - \sin \theta_\odot \sin \Phi_\odot \sin \theta_c \sin \Phi_c + \cos \theta_\odot \cos \theta_c)$$

**Source:** Andre Ehrlich, Leipzig Institute for Meteorology (a.ehrlich@uni-leipzig.de)

### References:

## 7.5 Solar Vector Calculation (Blanco)

**Algorithm name:** solar\_vector\_blanco

**Category:** Radiation

**Summary:** This algorithm computes the current solar vector, given current date, time, latitude and longitude. Algorithm is most accurate between 1999-2005, but calculations out to 2015 show the solar vector can be determined with an error of less than 0.5 minutes of arc.

### Inputs:

Date_time	Vector	ISO String of current date/time in UTC [yyyymmddThhmmss]
lat	Vector	Latitude [degrees]
long	Vector	Longitude [degrees]

### Outputs:

ra	Vector	Right ascension [radians]
$\delta$	Vector	Declination [radians]
$\theta_z$	Vector	Solar Zenith [radians]
$\gamma$	Vector	Solar Azimuth [radians]

### Formula:

$$jd = \frac{1461}{4}(y + 4800 + (m - 14)/12) + \frac{367}{12}(m - 2 - 12((m - 14)/12))$$

$$- \frac{3}{4}(y + 4900 + (m - 14)/12)/100 + d - 32075 - 0.5 + hour/24.0$$

$$jd = (1461(y + 4800 + (m - 14)/12))/4 + (367(m - 2 - 12((m - 14)/12)))/12$$

$$- (3((y + 4900 + (m - 14)/12)/100))/4 + d + 32075 - 0.5 + hour/24.0$$

where  $y$  is the year,  $m$  is the month,  $d$  is the day of the month and  $hour$  is the current hour in decimal format, i.e. with minutes and seconds as fractions of an hour. Note that all divisions in this calculation are integer divisions except the last.

The ecliptic coordinates of the sun are computed from the Julian Day by:

$$n = jd - 2451545.0$$

$$\Omega = 2.1429 - 0.0010394594n$$

$$L \text{ (mean longitude)} = 4.8950630 + 0.017202791698n$$

$$g \text{ (mean anomaly)} = 6.2400600 + 0.0172019699n$$

$$l \text{ (ecliptic longitude)} = L + 0.03341607 \sin g + 0.00034894 \sin 2g - 0.0001134 - 0.0000203 \sin \Omega$$

$$ep \text{ (obliquity of the ecliptic)} = 0.4090928 - 6.2140 \times 10^{-9}n + 0.0000396 \cos \Omega$$

The conversion from ecliptic coordinates to celestial coordinates is computed by:

$$\begin{aligned} ra \text{ (right ascension)} &= \tan^{-1} \left[ \frac{\cos ep \sin l}{\cos l} \right] \\ \delta \text{ (declination)} &= \sin^{-1} [\sin ep \sin l] \end{aligned}$$

where  $ra$  must be between 0 and  $2\pi$ .

The conversion between celestial coordinates to horizontal coordinates is then computed by the following equations:

$$\begin{aligned} gmst &= 6.6974243242 + 0.0657098283n + hour \\ lmst &= \frac{\pi}{180} (15gmst + long) \\ \omega \text{ (hour angle)} &= lmst - ra \\ \theta_z &= \cos^{-1} [\cos lat \cos \omega \cos \delta + \sin \delta \sin lat] \\ \gamma &= \tan^{-1} \left[ \frac{-\sin \omega}{\tan \delta \cos lat - \sin lat \cos \omega} \right] \\ Parallax &= \frac{EarthMeanRadius}{AstronomicalUnit} \sin \theta_z \\ \theta_z &= \theta_z + Parallax \end{aligned}$$

where:  $EarthMeanRadius = 6371.01$  km and  $AstronomicalUnit = 149597890$  km

**Source:**

**References:** Manuel Blanco-Muriel, et al., “Computing the Solar Vector,” *Solar Energy* 70 (2001): 436-38. [2]

## 7.6 Solar Vector Calculation (Reda-Andreas)

**Algorithm name:** solar\_vector\_reda

**Category:** Radiation

**Summary:** This algorithm calculates the current solar vector based on time, latitude and longitude inputs. It accepts optional pressure and temperature arguments to correct for atmospheric refraction effects. The zenith and azimuth angle calculated by this algorithm have uncertainties equal to  $\pm 0.0003^\circ$  in the period from the year -2000 to 6000.

### Inputs:

Date_time	Vector	ISO String of current date/time in UTC [yyyymmddThhmmss]
lat	Vector	Latitude [degrees]
long	Vector	Longitude [degrees]
E	Vector	Elevation [m]
P	Vector, Optional	Local pressure [hPa]
T	Vector, Optional	Local temperature [ $^\circ\text{C}$ ]

### Outputs:

$\theta$	Vector	Solar Zenith [degrees]
$\Phi$	Vector	Solar Azimuth [degrees]

### Formula:

1. Calculate Julian and Julian Ephemeris Day, Century and Millennium:

(a) Calculate Julian Day ( $JD$ ):

$$JD = \text{INT}(365.25(Y + 4716)) + \text{INT}(30.6001(M + 1)) + D + B - 1524.5$$

where:

- INT is the integer of the calculated terms (e.g.  $8.7 = 8$ ,  $8.2 = 8$ , etc)
- $Y$  is the year
- $M$  is the month of the year. If  $M \leq 2$  then  $Y = Y - 1$  and  $M = M + 12$
- $D$  is the day of the month with decimal time (i.e. with fractions of the day being represented after the decimal point.)
- $B$  is equal to 0 for the Julian Calendar, and equal to  $(2 - A + \text{INT}(A/4))$  for the Gregorian calendar, where  $A = \text{INT}(Y/100)$

(b) Calculate Julian Ephemeris Day ( $JDE$ ):

$$JDE = JD + \frac{\Delta T}{86400}$$



Where  $\Delta T$  is the difference between the Earth rotation time and the Terrestrial Time. It can be calculated following the NASA “Polynomial expressions for  $\delta T$  ( $\Delta T$ )” [12].

- (c) Calculate Julian Century ( $JC$ ) and the Julian Ephemeris Century ( $JCE$ ) for the 2000 standard epoch:

$$JC = \frac{JD - 2451545}{36525}$$

$$JCE = \frac{JDE - 2451545}{36525}$$

- (d) Calculate the Julian Ephemeris Millennium ( $JME$ ) for the 2000 standard epoch:

$$JME = \frac{JCE}{10}$$

2. Calculate Earth heliocentric longitude, latitude and radius vector ( $L$ ,  $B$ , and  $R$ ):

- (a) Calculate  $L0_i$  and  $L0$ :

$$L0_i = A_i \cos(B_i + C_i \times JME)$$

$$L0 = \sum_{i=0}^n L0_i$$

Where the terms  $A_i$ ,  $B_i$  and  $C_i$  are based on values found in table A4.2 of the algorithm literature [9].

- (b) Calculate the terms  $L1$ ,  $L2$ ,  $L3$ ,  $L4$  and  $L5$  by using these same equations, but using the appropriate terms from the table.  
(c) Calculate the Earth heliocentric longitude (in radians):

$$L = 10^{-8}(L0 + L1 \times JME + L2 \times JME^2 + L3 \times JME^3 + L4 \times JME^4 + L5 \times JME^5)$$

- (d) Convert  $L$  to degrees and limit between  $0^\circ$  and  $360^\circ$ .  
(e) Calculate the Earth heliocentric latitude  $B$  by using table A4.2 and repeating steps (a)-(c) using the appropriate values. Then convert  $B$  to degrees. Note that there are no  $B2$  through  $B5$ .  
(f) Calculate the Earth radius vector  $R$  (in AU) in a similar manner by repeating steps (a)-(c) and using the appropriate values from table A4.2.

3. Calculate the geocentric longitude and latitude ( $\Theta$  and  $\beta$ ):

$$\Theta = L + 180$$

$$\beta = -B$$

Where  $\Theta$  must be limited between  $0^\circ$  and  $360^\circ$ .

4. Calculate the nutation in longitude and obliquity ( $\Delta\psi$  and  $\Delta\epsilon$ ):

- (a) Calculate the mean elongation of the moon from the sun (in degrees):

$$X_0 = 297.85036 + 445267.11480JCE - 0.0019142JCE^2 + \frac{JCE^3}{189474}$$

- (b) Calculate the mean anomaly of the sun (in degrees):

$$X_1 = 357.52772 + 35999.050340JCE - 0.0001603JCE^2 - \frac{JCE^3}{300000}$$

- (c) Calculate the mean anomaly of the moon (in degrees):

$$X_2 = 134.96298 + 477198.867398JCE + 0.0086972JCE^2 + \frac{JCE^3}{56250}$$

- (d) Calculate the moon's argument of latitude (in degrees):

$$X_3 = 93.27191 + 483202.017538JCE - 0.0036825JCE^2 + \frac{JCE^3}{327270}$$

- (e) Calculate the longitude of the ascending node of the moon's mean orbit on the ecliptic, measured from the mean equinox of the date (in degrees):

$$X_4 = 125.04452 - 1934.136261JCE + 0.0020708JCE^2 + \frac{JCE^3}{450000}$$

- (f) For each row in table A4.3, calculate the terms  $\Delta\psi$  and  $\Delta\epsilon$  (in 0.0001 of arc seconds):

$$\Delta\psi_i = (a_i + b_iJCE) \sin \left( \sum_{j=0}^4 X_j Y_{i,j} \right)$$

$$\Delta\epsilon_i = (c_i + d_iJCE) \cos \left( \sum_{j=0}^4 X_j Y_{i,j} \right)$$

where:

- $a_i$ ,  $b_i$ ,  $c_i$  and  $d_i$  are the values listed in the  $i$ th row and columns a, b c and d in Table A4.3.
- $X_j$  are the  $X$  values calculated above
- $Y_{i,j}$  are the values in row  $i$  and  $j$ th Y column in table A4.3.

- (g) Calculate the nutation in longitude and obliquity (in degrees):

$$\Delta\psi = \frac{\sum_{i=0}^{63} \Delta\psi_i}{36000000}$$

$$\Delta\epsilon = \frac{\sum_{i=0}^{63} \Delta\epsilon_i}{36000000}$$

5. Calculate the true obliquity of the ecliptic (in degrees):

$$\begin{aligned}
 U &= JME/10 \\
 \epsilon_0 &= 84381.448 - 4680.93U - 1.55U^2 + 1999.25U^3 - 51.38U^4 \\
 &\quad - 249.67U^5 - 39.05U^6 + 7.12U^7 + 27.87U^8 + 5.79U^9 + 2.45U^{10} \\
 \epsilon &= \epsilon_0/3600 + \Delta\epsilon
 \end{aligned}$$

6. Calculate the aberration correction (in degrees):

$$\Delta\tau = -\frac{20.4898}{3600R}$$

7. Calculate the apparent sun longitude (in degrees):

$$\lambda = \Theta + \Delta\psi + \Delta\tau$$

8. Calculate the apparent sidereal time at Greenwich at any given time (in degrees):

$$\begin{aligned}
 \nu_0 &= 280.46061837 + 360.98564736629(JD - 2451545) + 0.000387933JC^2 - \frac{JC^3}{38710000} \\
 \nu &= \nu_0 + \Delta\psi \cos \epsilon
 \end{aligned}$$

where  $\nu_0$  must be limited to the range from  $0^\circ$  to  $360^\circ$ .

9. Calculate the geocentric sun right ascension (in degrees):

$$\alpha = \frac{180}{\pi} \tan^{-1} \left( \frac{\sin \lambda \cos \epsilon - \tan \beta \sin \epsilon}{\cos \lambda} \right)$$

where, as before,  $\alpha$  must be limited to the range from  $0^\circ$  to  $360^\circ$ .

10. Calculate the geocentric sun declination  $\delta$  (in degrees):

$$\delta = \frac{180}{\pi} \sin^{-1}(\sin \beta \cos \epsilon + \cos \beta \sin \epsilon \sin \lambda)$$

11. Calculate the observer local hour angle (in degrees):

$$H = \nu + long - \alpha$$

Limit  $H$  from  $0^\circ$  to  $360^\circ$ , and note that in this algorithm  $H$  is measured westward from south.

12. Calculate the topocentric sun right ascension and declination (in degrees):

- (a) Calculate the equatorial horizontal parallax of the sun (in degrees):

$$\xi = \frac{8.794}{3600R}$$

- (b) Calculate the terms  $u$  (in radians),  $x$  and  $y$ :

$$u = \tan^{-1}(0.99664719 \tan lat)$$

$$x = \cos u + \frac{E}{6378140} \cos lat$$

$$y = 0.99664719 \sin u + \frac{E}{6378140} \sin lat$$

- (c) Calculate the parallax in the sun right ascension (in degrees):

$$\Delta\alpha = \frac{180}{\pi} \tan^{-1} \left( \frac{-x \sin \xi \sin H}{\cos \delta - x \sin \xi \cos H} \right)$$

- (d) Calculate the topocentric sun right ascension and declination (in degrees):

$$\alpha' = \alpha + \Delta\alpha$$

$$\delta' = \tan^{-1} \left( \frac{(\sin \delta - y \sin \xi) \cos \Delta\alpha}{\cos \delta - x \sin \xi \cos H} \right)$$

13. Calculate the topocentric local hour angle (in degrees):

$$H' = H - \Delta\alpha$$

14. Calculate the topocentric zenith angle (in degrees):

- (a) Calculate the topocentric elevation angle without atmospheric correction (in degrees):

$$e_0 = \frac{180}{\pi} \sin^{-1}(\sin lat \sin \delta' + \cos lat \cos \delta' \cos H')$$

- (b) Calculate the atmospheric refraction correction (in degrees):

$$\Delta e = \frac{P}{1010} \frac{283}{(T + 273)} \frac{1.02}{60 \tan \left( e_0 + \frac{10.3}{e_0 + 5.11} \right)}$$

Note that this step is skipped if temperature and pressure are not provided by the user. Also note that the argument for the tangent is computed in degrees. A conversion to radians may be needed if required by your computer or calculator.

- (c) Calculate the topocentric elevation angle (in degrees):

$$e = e_0 + \Delta e$$

- (d) Calculate the topocentric zenith angle (in degrees):

$$\theta = 90 - e$$

15. Calculate the topocentric azimuth angle (in degrees):

$$\Phi = \frac{180}{\pi} \tan^{-1} \left( \frac{\sin H'}{\cos H' \sin lat - \tan \delta' \cos lat} \right) + 180$$

Limit  $\Phi$  from  $0^\circ$  to  $360^\circ$ . Note that  $\Phi$  is measured eastward from north.

**Source:**

**References:** Reda and Andreas, “Solar Position Algorithm for Solar Radiation Applications,” National Renewable Energy Laboratory, Revised 2008, accessed February 14, 2012, <http://www.nrel.gov/docs/fy08osti/34302.pdf>. [9]

## 7.7 Blackbody Temperature

**Algorithm name:** temp\_blackbody

**Category:** Radiation

**Summary:** Calculates the blackbody temperature for a given radiance at a specific wavelength.

**Inputs:**

$rad$	Vector	Blackbody radiance [W m <sup>-2</sup> sr <sup>-1</sup> nm <sup>-1</sup> ]
$\lambda$	Coeff	Wavelength [nm]

**Outputs:**

$T$	Vector	Temperature [K]
-----	--------	-----------------

**Formula:** After converting  $\lambda$  to m and  $rad$  to W m<sup>-3</sup> sr<sup>-1</sup>, the blackbody temperature is calculated by:

$$T = \frac{hc}{k_B \lambda \ln\left(\frac{2hc^2}{\lambda^5 rad} + 1\right)}$$

where  $c$  is the speed of light in m/s,  $h$  is the Planck constant in J s and  $k_B$  is the Boltzmann constant in J/K.

**Source:** Andre Ehrlich, Leipzig Institute for Meteorology (a.ehrlich@uni-leipzig.de)

**References:**

## Chapter 8

# Quality Control

## 8.1 Check navigation data for inconsistencies

**Algorithm name:** nav\_chk

**Category:** Quality Control

**Summary:** Tests navigation file (position and attitude) for inconsistencies and corrects them. The code is based on a HyMap \*.gps File.

**Inputs:** \*.gps file plus the number of image lines according to the ENVI header of the related image data. The \*.gps file is a multi-column ASCII file derived by HyVista Corp. proprietary software, which synchronises times and generates an output which is indexed by scan line number. The table below shows the list of parameters.

Parameters	Example	Description
Line	1	Scan line number
UTC Time	48835.0462/20/5/2004	Time of day in seconds/day/month/year
VME Time	929386852.0	Internal computer tick time in microseconds
IMU Time	2048825953.1	Internal IMU time in microseconds
Latitude	48.03321015	Decimal degrees (positive = north, negative = south)
Longitude	11.28140200	Decimal degrees (positive = east, negative = west)
Altitude	2970.79892155	Meters above MSL
Pitch	0.22235917	Decimal degrees (positive = nose up)
Roll	0.54269902	Decimal degrees (positive = right wing up)
Heading	0.37774316	Decimal degrees (positive = N-E-S direction, negative = N-W-S direction)
True Track	1.00507651	Decimal degrees (0 to 360)
Ground Speed	72.90907700	Meters / second
Sat	5	Number of satellites being received
DGPS	1	DGPS status: 1 = DGPS being received 0 = no DGPS received

**Outputs:** status file → template+'\_status'

If applicable: corrected gps file

backup of original .gps → filename.gps\_original

**Formula:** test & correct the following

- point or colon - separator in .gps = ; error caught in hymap\_read\_gps.pro corrected when re-writing the .gps-file anyway



- #lines in image = #lines in gps
  - if too many gps-lines: truncate lines at beginning (like Hyvista does)
  - if too few gps-lines: adding extrapolated lines at end
- invalid start / end time: calculating average timestep & using last reliable line
- data gaps (indicated by identical time): interpolate info

**Source:** DLR-DFD

**References:** EUFAR FP7 - DJ2.2.2 - Quality Layers for VITO, DLR, INTA and PML

## 8.2 Additional consistency check & QA for navigation data (no correction!)

**Algorithm name:** nav\_const

**Category:** Quality Control

**Summary:** Tests navigation file (position and attitude) for consistency. The code is based on a HyMap \*.gps File.

This check can be performed after nav\_chk.pro.

**Inputs:** \*.gps file. The \*.gps file is a multi-column ASCII file derived by HyVista Corp. proprietary software, which synchronises times and generates an output which is indexed by scan line number. The table below shows the list of parameters.

Parameters	Example	Description
Line	1	Scan line number
UTC Time	48835.0462/20/5/2004	Time of day in seconds/day/month/year
VME Time	929386852.0	Internal computer tick time in microseconds
IMU Time	2048825953.1	Internal IMU time in microseconds
Latitude	48.03321015	Decimal degrees (positive = north, negative = south)
Longitude	11.28140200	Decimal degrees (positive = east, negative = west)
Altitude	2970.79892155	Meters above MSL
Pitch	0.22235917	Decimal degrees (positive = nose up)
Roll	0.54269902	Decimal degrees (positive = right wing up)
Heading	0.37774316	Decimal degrees (positive = N-E-S direction, negative = N-W-S direction)
True Track	1.00507651	Decimal degrees (0 to 360)
Ground Speed	72.90907700	Meters / second
Sat	5	Number of satellites being received
DGPS	1	DGPS status: 1 = DGPS being received 0 = no DGPS received

**Outputs:** if (KEYWORD\_SET(gps\_err\_array)) → QC array  
 otime, lat, lon, alt, pit, rol, heading, track, speed, sat, dgps  
 Values: 0:OK 1:minor problem 2:major problem  
 if (KEYWORD\_SET(gps\_data)) → gps data as array  
 otime, lat, lon, alt, pit, rol, heading, track, speed, sat, dgps

**Formula:** test & report the following

- if data range is not plausible

- if change between steps  $>$  threshold:  
latlon, alt, pit, rol, heading, track, speed
- uncorrectable errors in:  
time, latlon, alt, pit, rol, heading, track, speed, sat, dgps

**Source:** DLR-DFD

**References:** EUFAR FP7 - DJ2.2.2 - Quality Layers for VITO, DLR, INTA and PML

# Bibliography

- [1] Bellec, H. and G. Duverneuil, 1996: Appareils de mesure de l’hygrométrie sur le Merlin IV. Note de Centre 9, Météo-France CNRM/CAM, July 1996.
- [2] Blanco-Muriel, Manuel, Alarcón-Padilla, Diego C., López-Moratalla, Teodoro, and Lara-Coira, Martín, 2001: Computing the Solar Vector. *Solar Energy*, **70**, 431–441.
- [3] Bohn, D., and H. Simon, 1975: Mehrparametrische Approximation der Eichräume und Eichflächen von Unterschall- bzw. Überschall-5-Loch-Sonden. *Archiv für Technisches Messen und Meßtechnische Praxis*, Vol 470 (3) 31–37.
- [4] Candel, S., 1990. *Mécanique des fluides*. Dunod.
- [5] Droplet Measurement Technologies, Inc, 2009. Data Analysis User’s Guide Chapter I: Single Particle Light Scattering. DOC-0222, Rev A. Accessed February 2, 2012. [http://www.dropletmeasurement.com/sites/default/files/ManualsGuides/Data Analysis Guide/DOC-0222 Rev A Data Analysis Guide Ch 1.pdf](http://www.dropletmeasurement.com/sites/default/files/ManualsGuides/Data%20Analysis%20Guide/DOC-0222%20Rev%20A%20Data%20Analysis%20Guide%20Ch%201.pdf)
- [6] Droplet Measurement Technologies, Inc, 2009. Data Analysis User’s Guide Chapter II: Single Particle Imaging. DOC-0223, Rev A. Accessed February 2, 2012. [http://www.dropletmeasurement.com/sites/default/files/ManualsGuides/Data Analysis Guide/DOC-0223 Rev A Data Analysis Guide Ch 2.pdf](http://www.dropletmeasurement.com/sites/default/files/ManualsGuides/Data%20Analysis%20Guide/DOC-0223%20Rev%20A%20Data%20Analysis%20Guide%20Ch%202.pdf)
- [7] Lenschow, D.H. and P. Spyers-Duran, 1989: Measurement Techniques: Air Motion Sensing. NCAR Bulletin No. 23, 1989. Accessed June 23, 2010. <http://www.eol.ucar.edu/raf/Bulletins/bulletin23.html>
- [8] Barmgardner, Darrel, 1989. Airborne Measurements for Cloud Microphysics. NCAR Bulletin No. 24, 1989. Accessed June 23, 2010. <http://www.eol.ucar.edu/raf/Bulletins/bulletin24.html>
- [9] Reda, Ibrahim and Afshin Andreas, 2008: Solar Position Algorithm for Solar Radiation Applications. National Renewable Energy Laboratory. Revised 2008. Last accessed February 14, 2012. <http://www.nrel.gov/docs/fy08osti/34302.pdf>.
- [10] Triplet, J.P. and G. Roche, 1971. *Météorologie Générale*. Météo-France.
- [11] van den Kroonenberg, A.C., T. Martin, M. Buschmann, J. Bange, and P. Vörsmann, 2008: Measuring the Wind Vector Using the Autonomous Mini Aerial Vehicle M<sup>2</sup>AV. *J. Atmos. Oceanic Technol.*, **25**, 1969–1982.
- [12] NASA, GSFC Eclipse website, <https://eclipse.gsfc.nasa.gov/LEcat5/deltapoly.html>.