

Pyradi: an open-source toolkit for infrared calculation and data processing

Cornelius J. Willers^a, Maria S. Willers^b, Ricardo Augusto T. Santos^c, Petrus J. van der Merwe^b, Johannes J. Calitz^a, Alta de Waal^a, Azwitamisi E. Mudau^a

^a CSIR, P.O. Box 395, 0001 Pretoria, South Africa

^b Denel Dynamics, P.O. Box 7412, 0046 Centurion, South Africa

^c Instituto Tecnológico de Aeronáutica, Laboratório de Guerra Eletrônica, Pca. Mal. Eduardo Gomes, 50 São José Dos Campos - SP - Brazil 12228-900

ABSTRACT

Electro-optical system design, data analysis and modelling involve a significant amount of calculation and processing. Many of these calculations are of a repetitive and general nature, suitable for including in a generic toolkit. The availability of such a toolkit facilitates and increases productivity during subsequent tool development: “develop once and use many times”. The concept of an extendible toolkit lends itself naturally to the open-source philosophy, where the toolkit user-base develops the capability cooperatively, for mutual benefit. This paper covers the underlying philosophy to the toolkit development, brief descriptions and examples of the various tools and an overview of the electro-optical toolkit.

The toolkit is an extendable, integrated collection of basic functions, code modules, documentation, example templates, tests and resources, that can be applied towards diverse calculations in the electro-optics domain. The toolkit covers (1) models of physical concepts (e.g. Planck’s Law), (2) mathematical operations (e.g. spectral integrals, spatial integrals, convolution, 3-D noise calculation), (3) data manipulation (e.g. file input/output, interpolation, normalisation), and (4) graphical visualisation (2-D and 3-D graphs).

Toolkits are often written in scriptable languages, such as Python and Matlab. This specific toolkit is implemented in Python and its associated modules Numpy, SciPy, Matplotlib, Mayavi, and PyQt/PySide. In recent years these tools have stabilised and matured sufficiently to support mainstream tool development. Collectively, these tools provide a very powerful capability, even beyond the confines of this toolkit alone. Furthermore, these tools are freely available.

Rudimentary radiometric theory is given in the paper to support the examples given. Examples of the toolkit use, as described in the paper, include (1) spectral radiometric calculations of arbitrary source-medium-sensor configurations, (2) spectral convolution processing, (3) 3-D noise analysis, (4) loading of ASCII text files, binary files, MODTRAN `tape7` and FLIR Inc `*.ptw` files, (5) data visualisation in 2-D and 3-D graphs and plots, (6) detector modelling from detail design parameters (bulk material detectors), (7) colour coordinate calculations, and (8) various utility functions.

The toolkit is developed as a cooperative effort between the CSIR, Denel SOC and DCTA. The project, available on Google Code at <http://code.google.com/p/pyradi>, is managed in accordance with general practice in the open source community.

Keywords: Pyradi, radiometry, infrared calculation, MODTRAN, visualisation

Further author information: (Send correspondence to C.J.W.)

C.J.W.: E-mail: nwillers@csir.co.za, Telephone: +27-12-841-4261

M.S.W.: E-mail: riana.willers@deneldynamics.co.za, Telephone: +27-12-671-1901

R.A.T.S.: E-mail: Ricardo.tavares@ita.br, Telephone: +551239476899

1. INTRODUCTION

Electro-optical system design involves the integration of a diverse set of technologies and concepts, covering areas such as real world object signatures, atmospheric effects, optics, optical detectors, electronics and signal processing. The design process involves the optimisation of several (often interrelated) parameters in this electro-optical system. Such optimisation requires modelling of the relevant components in the system and performing trade-off calculations and data analysis with these models.

Many of these calculations are of a repetitive and general nature, suitable for adding to a generic toolkit. The availability of a well-designed toolkit facilitates and increases productivity during subsequent tool development, where new tools are added to an ever-increasing set of tools. The concept of an extendible toolkit lends itself naturally to the open-source philosophy, where the toolkit user-base develops the capability cooperatively, for mutual benefit.

The development of this toolkit is following the Unix philosophy for software development, summarised in the words of Doug McIlroy: “Write programs that do one thing and do it well. Write programs to work together.” In broader terms the philosophy was stated by Eric Raymond¹ (only selected items shown here): (1) Rule of Modularity: Write simple parts connected by clean interfaces. (2) Rule of Clarity: Clarity is better than cleverness. (3) Rule of Composition: Design programs to be connected to other programs. (4) Rule of Simplicity: Design for simplicity; add complexity only where you must. (5) Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do. (6) Rule of Transparency: Design for visibility to make inspection and debugging easier. (7) Rule of Robustness: Robustness is the child of transparency and simplicity. (8) Rule of Representation: *Fold knowledge into data* so program logic can be stupid and robust. (9) Rule of Economy: Programmer time is expensive; conserve it in preference to machine time. (10) Rule of Generation: Avoid hand-hacking; write programs to write programs when you can. (11) Rule of Optimisation: Prototype before polishing. Get it working before you optimise it. (12) Rule of Extensibility: Design for the future, because it will be here sooner than you think.

The pyradi* toolkit is an extendable, integrated and coherent collection of basic functions, code modules, documentation, example templates, tests and resources, that can be applied towards diverse calculations in the electro-optics domain. The fundamental principle in constructing this toolkit is therefore to write a number of cooperating specialised modules, where each module focusses on one task only, and perform this task with minimal workload on the user.

While several toolkits for Matlab and Python exist in other scientific domains, there is no radiometry toolkit readily available.

This paper covers the domain requirements for the toolkit, the language selection considerations, the toolkit design considerations and the structure of the toolkit. Several worked examples of the toolkit are presented. Finally, the instructions for downloading the toolkit are given.

2. REQUIREMENTS FOR TOOLKIT

2.1 Electro-Optical Domain Requirements

A typical radiometry toolkit requirement (very much simplified) is the calculation of the detector current of an electro-optical sensor viewing a target object. The system can be conceptually modelled as shown in Figure 1, comprising a radiating source with spectral radiance, an intervening medium (e.g. the atmosphere), a spectral filter, optics, a detector and an amplifier. The amplifier output signal can be calculated² by integrating Equation 1 over all wavelengths, over the full source area A_0 and over the optical aperture area A_1 ,

$$v = Z_t \int_{A_0} \int_{A_1} \frac{1}{r_{01}^2} \int_0^\infty \epsilon_\lambda L_\lambda(T, A_0) \tau_{a\lambda} \tau_{s\lambda}(A_1) \mathcal{R}_\lambda d\lambda d(\cos \theta_0 A_0) d(\cos \theta_1 A_1). \quad (1)$$

*The name pyradi is derived from the combination of the two words ‘Python’ and ‘Radiometry’. In accordance with Python practice it is spelt with all lowercase letters. For grammatical purposes the name pyradi is capitalised according to English grammar rules, where necessary.

In Equation 1 v is the output signal voltage, r_{01} is the distance between elemental areas $d(\cos\theta_0 A_0)$ and $d(\cos\theta_1 A_1)$, ϵ_λ is the source spectral emissivity, $L_\lambda(T, A_0)$ is the Planck Law radiation at temperature T at location A_0 , $\tau_{a\lambda}$ is the atmospheric spectral transmittance, $\tau_{s\lambda}(A_1)$ is the sensor spectral transmittance at location A_1 , \mathcal{R}_λ is the spectral detector responsivity in [A/W], Z_t is the amplifier transimpedance gain in [V/A]. The spectral integral $\int_0^\infty d\lambda$ accounts for the total flux for all wavelengths, the spatial integral $\int_{A_0} d(\cos\theta_0 A_0)$ accounts for flux over the total area of the source, and the spatial integral $\int_{A_1} d(\cos\theta_1 A_1)$ accounts for the total area of the receiving area.

A spectral variable can be considered a function of wavelength. Consider two sets A and B . The set A , called the *domain*, is a set of numbers which represents the wavelengths, wavenumbers or frequencies at which the spectral variable is defined. The set B , called the *codomain* or spectral quantity, is the set of values of the spectral variable at the specific points defined in the domain A . We define the spectral variable (a function) f from A to B such that for each $a \in A$, there is a unique $f(a) = b \in B$. Examples of B are spectral emissivity, spectral transmittance or spectral detector responsivity. The top graphic in Figure 1 illustrates the reasoning behind the spectral integral as a product, followed by an integral (summation),

$$\int_0^\infty \epsilon_\lambda L_\lambda(T) \tau_{a\lambda} \tau_{s\lambda} \mathcal{R}_\lambda d\lambda, \quad (2)$$

where the spectral variability of the source, medium and sensor parameters are multiplied as spectral variables and afterwards integrated over all wavelengths to yield the total in-band signal. The domain of spectral quantities can be stated in terms of a wavelength, wavenumber, or less often, temporal frequency. The toolkit must be able to support all three domain types, as well as the conversion between spectral densities, such as [W/(m²·μm)] to [W/(m²·cm⁻¹)].

Likewise, the source radiance is integrated over the two respective areas of the target A_0 , and the sensor aperture A_1 . Note that if the sensor field of view footprint at the source is smaller than the physical source area, only the flux emanating from the footprint area is integrated.

Fundamental to almost all electro-optics calculations are the use of Planck's Law for thermal radiation. According to Planck's Law the maximum spectral radiant emittance and the spectral radiant emittance temperature derivative, for a given temperature T , are given by

$$M_{e\lambda}(T) = \frac{2\pi hc^2}{\lambda^5 \left(e^{\frac{hc}{\lambda kT}} - 1 \right)}, \quad \frac{dM_{e\lambda}(T)}{dT} = \frac{2\pi hc^2 x e^x}{T \lambda^5 \left(e^{\frac{hc}{\lambda kT}} - 1 \right)^2}, \quad (3)$$

where $x = \frac{hc}{\lambda kT}$, c is the speed of light, λ is wavelength, h is Planck's constant, k is the Boltzman constant, yielding emittance in units of [W/m³] and the derivative in units of [W/(m³·K)]. The toolkit must support the calculation of Planck's Law in any of the three spectral domain variables: wavelength [μm], wavenumber [cm⁻¹] and frequency [Hz].

Spectral convolution is required when working with spectral data filtered to different spectral widths. Under condition that a narrow-band optics filter is used in the system in Figure 1, the spectral integral, Equation 2, can be written, by change of variable $\lambda = \lambda_c - x$, as follows

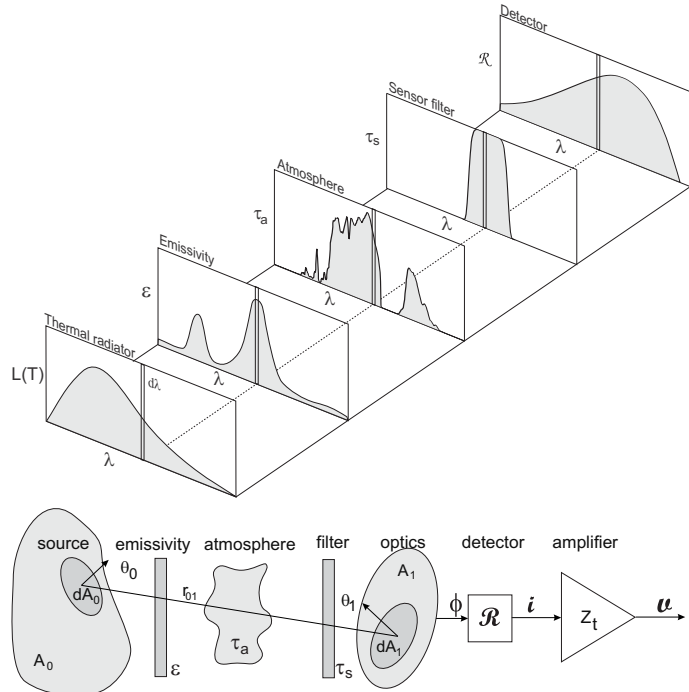


Figure 1: Simple model of a sensor.

$$\int_{\lambda_c - \frac{\Delta\lambda}{2}}^{\lambda_c + \frac{\Delta\lambda}{2}} \epsilon_\lambda L_\lambda(T) \tau_{a\lambda} \tau_{s\lambda} \mathcal{R}_\lambda d\lambda = \int_{-\frac{\Delta\lambda}{2}}^{+\frac{\Delta\lambda}{2}} \epsilon_x L_x(T) \tau_{ax} \tau_{s(\lambda_c-x)} \mathcal{R}_x dx. \quad (4)$$

These equations illustrate very clearly that the irradiance measured with the filter centered around wavelength λ_c includes source energy from $\lambda_c - \frac{\Delta\lambda}{2}$ to $\lambda_c + \frac{\Delta\lambda}{2}$. Apart from the spectral selection, the filter has an additional effect of smoothing the spectrum being observed, since the filter has a non-zero spectral width. Equation 4 is called a convolution integral since it describes the convolution between the product $\epsilon_\lambda L_\lambda \tau_{a\lambda} \mathcal{R}_\lambda$ and $\tau_{s\lambda}$. Supporting the convolution integral is an important requirement for the toolkit.

There is an abundance of colour coordinate definitions, each optimised for different applications. Essentially, the calculation of colour coordinates is a radiometric calculation involving normalisation with given spectral weights. Commonly used colour spaces include the CIE 1931 tristimulus values XYZ, or the xyY chromaticity colour space.^{3,4} The toolbox must provide at least rudimentary capabilities for colour space calculation.

Non-trivial real-world problems require integration over spatial surfaces, i.e. over the surface of extended objects and large solid angles. One common example is the solid angle of simple Euclidian shapes, such as a large rectangular plate, shown in Figure 2. The geometric solid angle, ω_s , and projected solid angle, Ω_s , of the rectangular flat surface, as seen from a reference point centred above the plate, are given by the following two equations:

$$\omega_s = \int_W \int_D \frac{dw dd}{H^2} \left(\frac{H}{\sqrt{w^2 + d^2 + H^2}} \right)^3 \quad (5)$$

$$\Omega_s = \int_W \int_D \frac{dw dd}{H^2} \left(\frac{H}{\sqrt{w^2 + d^2 + H^2}} \right)^4 \quad (6)$$

where W and D are the dimensions of the rectangle and H is the reference point height above the plate. The integral is performed along the W and D dimensions with increments of dw and dd . The slant range between the reference point and the elemental area $dw \times dd$ is $r = H / \cos \theta$. Even though each problem formulation is different, the toolkit must be able to support the calculation of such spatial integrals.

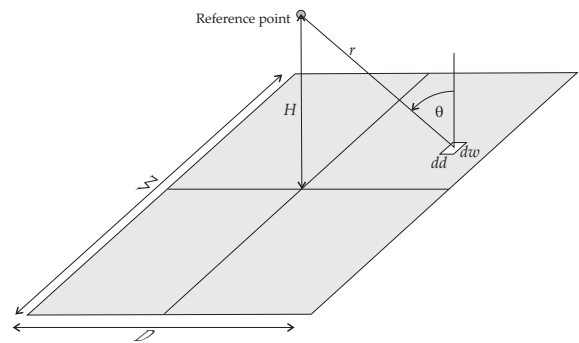


Figure 2: Solid angle of a centred flat plate.

2.2 General Software Requirements

In addition to the domain-specific requirements, there are also requirements to read data files, interpolate data and visualise results. Data visualisation is a very important element in the understanding and validation of the calculated results—errors are more readily recognised in graphical visualisation than in tabular data. There is also a need for mathematical tools for interpolation, normalisation and so forth. Scipy provides a substantial number of general purpose functions, including the values of physical constants, interpolation and integration.

Data visualisation is required in the form of two-dimensional (x, y) and three-dimensional (x, y, z) data graph plots. Two-dimensional data graphs must include combinations of linear and log scale plots, as well as polar plots. Three dimensional data graphs must include line plots and three-dimensional iso-surface plots.

One popular normalisation method, calculating the spectrally weighted effective value of a spectral variable, is given by

$$\mathcal{F}_{\text{eff}} = \frac{\int_0^\infty \mathcal{F}_\lambda \mathcal{G}_\lambda d\lambda}{\int_0^\infty \mathcal{G}_\lambda d\lambda}, \quad (7)$$

where \mathcal{F}_λ is the spectral variable in question and \mathcal{G}_λ is some common spectral variable. Note that the effective value of \mathcal{F} depends on the spectral shapes of both \mathcal{F} and \mathcal{G} ; the effective value of \mathcal{F} thus calculated therefore applies only to the specific \mathcal{G} used in the calculation.

A common need is for the reading of instrument data files, such as from imaging radiometers. Unfortunately each instrument has its own, often proprietary data format, but it would be convenient to be able to read such files in the toolkit. The toolkit currently provides the capability to read FLIR Inc.⁵ *.ptw files.

Modern software users seem to require Graphical User Interfaces (GUI) before accepting new software. Sadly so, since sometimes the GUI often comes in the way, or limits the software useability. In particular, the GUI can limit the automation of repetitive tasks. However, some tasks do lend themselves better to GUI interaction, such as a structured walk through a decision tree, where future decisions depend on the outcome of the current decision. For example, writing a MODTRAN tape5 data file is indeed an arduous task in a text editor and easier done in a GUI! The construction of selected GUI applications in pyradi may be addressed in future, but not in the short term.

3. DESIGN CONSIDERATIONS

3.1 Language Considerations

The requirements for an electro-optical calculation environment can be met by most modern computer languages, such as FORTRAN, Java, C/C++ or similar. However these languages often do not provide a native graphics toolkit and fail on the ease of implementing operations such as spectral multiplication and integration.

There are a number of languages that provide built-in operations on scalars, vectors and arrays/matrices with single operators. Examples of such languages are Matlab⁶ (or its open source equivalents SciLab⁷ and Octave⁸) and Python⁹ together with Scipy and Numpy.¹⁰ These languages offer the following benefits: (1) A spectral variable is easily modelled as a vector or column in an array or matrix. All spectral vectors are easily converted and interpolated to the same spectral values or converted between wavelength, wavenumber and frequency. (2) Spatial (area) integrals are readily computed by expressing the shape as a two-dimensional array. (3) Vector or array variables can be loaded from ASCII files with simple commands. (4) Graphical visualisation tools are available and easy to use. (5) An interactive environment supports very fast development time, as well as the capability to write script files for more complex problems. (6) The capability to write your own often-used functions or subroutines into re-usable toolboxes or modules.

The two primary candidates for the toolkit are Matlab and Python. For spectral calculations and plotting, such as shown here, the two products are practically equal in ease of use and capability. Both languages require equal effort in becoming a fluent and effective user. Python is a better (and constantly re-) engineered language, while Matlab grew out of a linear algebra background with some flawed design decisions and work-arounds for general programming applications. Python has a large number of modules providing a considerable capability as a general purpose language, while Matlab has a large number of specialised and powerful scientific and engineering toolboxes. The associated Python data visualisation tools, Matplotlib¹¹ and Mayavi,¹² provide considerably stronger capability than their Matlab equivalent. Python is also well supported for scientific processing with Numpy and SciPy. Python applications can employ high quality GUI interfaces with PyQt¹³ or PySide.¹⁴ Matlab is proprietary source and carries a hefty price tag (especially for the toolboxes), while Python and friends are free and open source.

After extensive use in industrial and academic environments, of both Matlab and Python, specifically for modelling radiometry systems, we decided to continue only with Python. This decision was not taken on emotional or ideological grounds, but is based purely on our perception of Python as a better all-round scripting language with better data visualisation tools. In recent years Python and friends have been well tested, and have stabilised and matured sufficiently to support mainstream tool development. PyQt carries some licence restrictions, but all the other packages are not encumbered by restrictive licence constraints.

3.2 Toolkit structure

The toolkit is a loose collection of files, containing classes and functions of similar nature. Within the toolkit, the tools are somewhat interdependent, borrowing shared functionality. The current set of module files are:

- ryplanck.py** This module provides functions for Planck Law emittance calculations, as well as Planck Law temperature derivative calculations. The functions provide spectral emittance in $[W/(m^2 \cdot *)]$ or $[q/(s \cdot m^2 \cdot *)]$, given the temperature and a spectral domain vector; with wavelength, wavenumbers and frequency supported. The total emittance can also be calculated by using the Stefan-Boltzman equation, in $[W/m^2]$ or $[q/(s \cdot m^2)]$.
- ryfiles.py** This module provides functions for file input/output. These are mostly wrapper functions, based on existing functions in other Python classes. Functions are provided to save a two-dimensional array to a text file, load selected columns of data from a text file, load a column header line, read & write raw binary files, process strings to include only legal filename characters, and a function from the Python Cookbook¹⁵ to recursively match filename patterns in a directory tree.
- ryplot.py** This module provides functions for cartesian plots, polar plots and three-dimensional plots. This class provides a basic plotting capability with a minimum number of lines. These are all wrapper functions, based on existing functions in other Python classes. Provision is made for combinations of linear and log scales, as well as polar plots for two-dimensional graphs, and image plotting. The module can also plot three-dimensional line data (e.g. (x, y, z) trajectories) and iso-surface data in three-dimensional graphics. The Plotter class can save files to disk in a number of formats for subsequent use in reports.
- ryutils.py** This module provides various utility functions for radiometry calculations. Functions are provided for modelling a maximally flat spectral filter, a simple photon detector spectral response, effective value calculation, spectral convolution, and conversion of spectral domain variables between $[\mu m]$, $[cm^{-1}]$ and $[Hz]$, conversion of spectral density quantities between $[/\mu m]$, $[/cm^{-1}]$ and $[/Hz]$.
- rychroma.py** This module provides rudimentary colour coordinate processing. A function is provided to calculate the CIE 1931 red-green-blue chromaticity coordinates for an arbitrary illuminance spectrum.
- ryptw.py** This module provides the capability to read FLIR Inc *.ptw files. All information in the file header is read and made available with the image frames.
- ry3dnoise.py** This module provides utilities to calculate three-dimensional noise parameters¹⁶ of an image sequence.
- rydetector.py** This module provides a bulk material detector model (e.g. Si, Ge, InSb), based on academic and well known parameters and models found in the classical literature. The model is built to be able to calculate and predict the main figures of merit for infrared detectors such as its I×V characteristics, detectivity, responsivity and noise equivalent power (NEP).
- rymodtran.py** This module provides the functionality to read in MODTRAN `tape7` files, given the column header names. More functionality will be added later.

The toolkit does not employ software exceptions for error handling, but rather signal errors by return value.

3.3 Toolkit file structure

The Python files all have a common structure, in keeping with standard Python module practice. The same file that is imported by the user, also contains the summary documentation and test/demonstration code. Each file has three sections: a header, the module classes and functions and the test/demonstration code.

The header is relatively short and contains the licence statement, the copyright notice and module version.

The second section of the file contains the code that will be executed when the user imports the module. The import process makes the code part of the user's library for the current session. Hence this is the code that the user will use in everyday application of the toolkit. The code in this section are normally collections of classes and functions. It should not contain demonstration or testing code.

The last section of the file is enclosed in an “if `__name__ == '__main__':`” block. The `__name__` variable is only equal to `__main__` if the file is executed as a script (i.e. not imported). The code in this if-block is not

imported/executed or visible when the file it normally *imported* as a module. This section should contain the test and demonstration code: both the developer and user may on occasion read this code for guidance on how to use the modules.

3.4 Numerical Approximations

The spectral integral in Equation 2 and spatial integrals can be calculated very simply by virtue of the theory of the Riemann integral as sums over ever decreasing intervals:

$$\int_0^\infty \epsilon_\lambda L_\lambda(T) \tau_{a\lambda} \tau_{s\lambda} \mathcal{R}_\lambda d\lambda \approx \sum_{i=0}^\infty \epsilon_{\lambda_i} L_{\lambda_i}(T) \tau_{a\lambda_i} \tau_{s\lambda_i} \mathcal{R}_{\lambda_i} \Delta\lambda. \quad (8)$$

Likewise can spatial integrals be done as the sum of small elements over the surface of the object.

In the pyradi toolkit integrals are calculated by using narrow spectral intervals or spatial sizes and using Numpy's trapezium summation function `numpy.trapz`. Provided that the intervals are made small enough, the summation yields reasonable results.

4. EXAMPLE APPLICATIONS

The examples of the toolkit application shown here are for illustration purposes only, the code itself is available in the module files on the pyradi web site (see Section 5).

4.1 Sensor irradiance calculation

The first example is a relatively complete worked example. The objective is to calculate the signal of a simple sensor, detecting the presence or absence of a flame in the sensor field of view. The sensor is pointed to an area just outside a furnace smokestack, against a clear sky background. The sensor must detect a change in signal, to indicate the presence or absence of a flame.

The sensor has an aperture area of $7.8 \times 10^{-3} \text{ m}^2$ and a field of view of $1 \times 10^{-4} \text{ sr}$. The sensor filter spectral transmittance is shown in Figure 3. The InSb detector has a peak responsivity of 2.5 A/W and normalised spectral response shown in Figure 3. The preamplifier transimpedance is 10000 V/A.

The flame area is 1 m^2 , the flame temperature is 1000 °C, and the emissivity is shown in Figure 3. The emissivity is 0.1 over most of the spectral band, due to carbon particles in the flame. At $4.3 \mu\text{m}$ there is a strong emissivity rise due to the hot carbon dioxide (CO_2) in the flame.

The distance between the flame and the sensor is 1000 m. The atmospheric properties are calculated with the MODTRAN Tropical climatic model. The path is oriented such that the sensor stares out to space, at a zenith angle of 88° . The spectral transmittance and path radiance along this path is shown in Figure 3.

The peak in the flame emissivity and the dip in atmospheric transmittance are both centered around the $4.3 \mu\text{m}$ CO_2 band. The calculation of flux radiative transfer through the atmosphere must account for the strong spectral variation, by using a spectral integral.

The signal caused by the flame is given by Equation 1, where the integrals over the surfaces of the flame and sensor are just their respective areas. The signal caused by the atmospheric path radiance is given by

$$v = Z_t \omega_{\text{optics}} A_{\text{optics}} \int_0^\infty L_{\text{path}\lambda} \tau_{s\lambda} \mathcal{R}_\lambda d\lambda, \quad (9)$$

where ω_{optics} is the sensor field of view, A_{optics} is the optical aperture area, $L_{\text{path}\lambda}$ is the spectral path radiance and the rest of the symbols are as defined for Equation 1.

An extract of the code to perform this calculation, and the resultant output, are as follows (the complete and well commented file is available on the web site):

Flame sensor

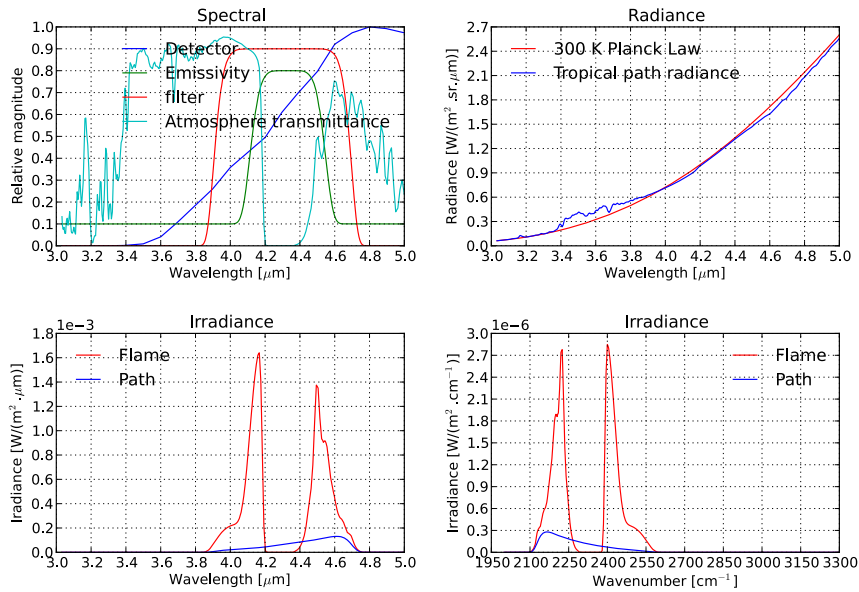


Figure 3: Flame sensor example spectral data.

```
# the transmittance is specified in the wavenumber domain with
# 5 cm-1 intervals, but we want to work in wavelength with 2.5 cm-1
waven = numpy.arange(2000.0, 3300.0, 2.5).reshape(-1, 1)
wavel= ryutils.convertSpectralDomain(waven, type='nw')

#remove comment lines, and scale path radiance from W/cm2.sr.cm-1 to W/m2.sr.cm-1
tauA = ryfiles.loadColumnTextFile('data/path1kmflamesensor.txt', [1],\
    abscissaOut=waven, comment='%')
lpathwn = ryfiles.loadColumnTextFile('data/pathspaceflamesensor.txt', [9],\
    abscissaOut=waven, ordinateScale=1.0e4, comment='%')
#convert path radiance spectral density from 1/cm^-1 to 1/um
(dum, lpathwl) = ryutils.convertSpectralDensity(waven, lpathwn, type='nw')

#load the detector file in wavelengths, and interpolate on required values
detR = ryfiles.loadColumnTextFile('data/detectorflamesensor.txt', [1],\
    abscissaOut=wavel, comment='%')

#construct the flame emissivity from parameters
emis = ryutils.sfilter(wavel,center=4.33, width=0.45, exponent=6, taupass=0.8, \
    taustop=0.1 )

#plot the data
plot1= ryplot.Plotter(1, 2, 2,'Flame sensor',figsize=(12,8))
#it seems that all attempts to plot in same subplot space must use same ptile.
plot1.plot(1, "Spectral","Wavelength [μm]", "Relative magnitude", wavel, detR,\
    plotCol=['b'], label=['Detector'])

# define sensor scalar parameters
opticsArea=7.8e-3 # optical aperture area [m2]
opticsFOV=1.0e-4 # sensor field of view [sr]
transZ=1.0e4 # amplifier transimpedance gain [V/A]
responsivity=2.5 # detector peak responsivity =A/W

# define the flame properties
flameTemperature = 1000+273.16 # temperature in [K]
flameArea = 1 # in [m2]
distance = 1000 # [m]
fill = (flameArea /distance**2) / opticsFOV # how much of FOV is filled
fill = 1 if fill > 1 else fill # limit target solid angle to sensor FOV

# first do for flame
# get spectral radiance in W/m^2.sr.cm-1
radianceFlame=ryplanck.planck(waven,flameTemperature,type='en').reshape(-1,1)/numpy.pi
```



```

inbandirradianceFlame = radianceFlame * detR * tauA * emis * filter * fill * opticsFOV
totalirradianceFlame=npumpy.trapz(inbandirradianceFlame.reshape((-1,1)),waven, axis=0)[0]
signalFlame = totalirradianceFlame *transZ*responsivity *opticsArea

plot1.saveFig('flamesensor01.eps')

```

```

Optics : area=0.0078 m^2 FOV=0.0001 [sr]
Amplifier: gain=10000.0 [V/A]
Detector : peak responsivity=2.5 [A/W]
Flame : temperature=1273.16 [K] area=1 [m^2] distance=1000 [m] fill=0.01 [-]
Flame : irradiance= 3.29e-04 [W/m^2] signal= 0.0641 [V]
Path : irradiance= 5.45e-05 [W/m^2] signal= 0.0106 [V]

```

It is clear that the flame signal is six times larger than the path radiance signal, even though the flame fills only 1% of the sensor field of view.

4.2 Solid angle calculation

Array processing in Python and Matlab condenses and simplifies most two-dimensional calculations significantly. The code to solve the integral in Equation 5, for $W=104$, $D=90$, $H=60$ is as follows:

Matlab code:

```

delta = 0.5;
x = [-45:delta:45];
y = [-52:delta:52];
a = ones(size(y))' * x ;
b = (ones(size(x))' * y)' ;
gv=(1 ./ ((a/60).^2 + (b/60).^2 + 1)).^(3/2);
solidAngle = delta.^2*trapz(trapz(gv))/(60*60)

```

Python code:

```

import numpy as np
x,y = np.mgrid[-45:45:181j, -52:52:209j]
gv = (1 / ( (x/60)**2 + (y/60)**2 + 1 )) ** (3./2)
a = np.trapz(gv, dx=0.5)
solidAngle = np.trapz(a, dx=0.5)/(60*60)

```

4.3 Spectral convolution

Spectral resolution matching is required when different data sources provide spectral data with different spectral resolution. One such example is shown where the signal from a Bunsen burner is analysed at 4 cm^{-1} , while the atmospheric transmittance data are calculated at 1 cm^{-1} . The left-hand side of Figure 4 shows the inappropriate combination of spectra at different spectral resolutions. On the right-hand side the same operation is performed but with resolution matched data. Note that this 'correction' is not accurate, since the atmospheric CO_2 concentration during the measurement was not measured.

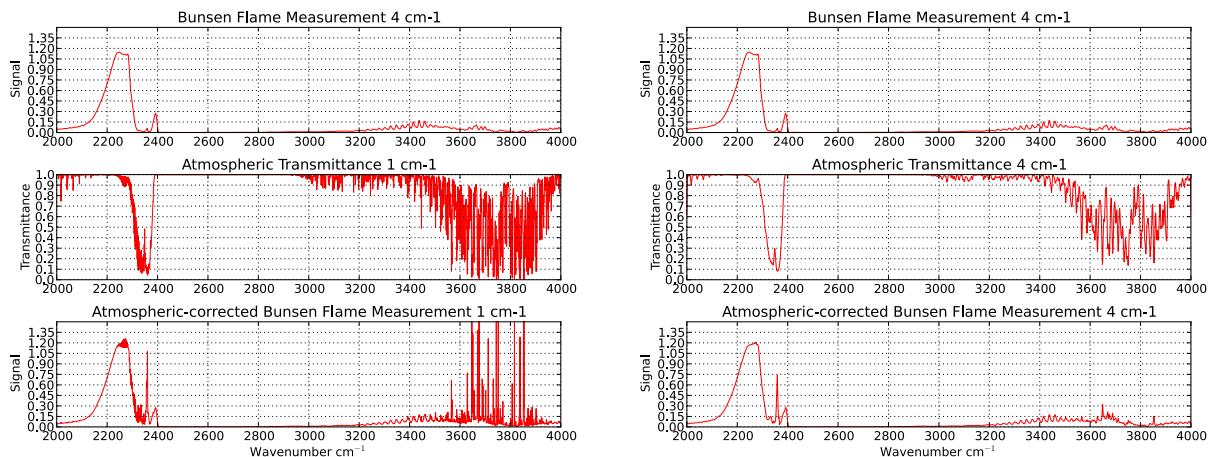


Figure 4: Example of spectral convolution calculations.

4.4 Reading FLIR Inc PTW files

Certain infrared FLIR Inc.⁵ imagers store data in a propriety format tagged with a *.ptw extension. This format is graphically summarised in Figure 5. The main header section describes the database itself, as well as all acquisition parameters. A stream of image frames follows the main header. These frames each starts with a frame header, containing the parameters specific to the frame, e.g. frame time, subwindowing and integration time.

The aim of the python code `ryptw.py` is to read all the header parameters and the image data. The code has no graphical user interface, as it is not a stand-alone analysis tool, but merely a way to read the acquisition parameters and the data frame(s). The current version of the code reads the file as digital levels, but it is planned to include camera calibration later, such as to yield radiance or temperature values.

The call to the code is a two step process, where the header information is read first, and using that header information, a specific frame can be extracted. A complete example of reading the *.ptw file is given on the web site. The typical code required to read a few frames is as follows:

```
header = ryptw.readPTWHeader('data/PyradiSampleMWIR.ptw')
ryptw.showHeader(header)
rows = header.h_Rows
cols = header.h_Cols

#loading sequence of frames
framesToLoad = [3,4,10]
data = getPTWFrames(header, framesToLoad)
print(data.shape)
```



Figure 5: FLIR Inc. *.ptw file format outline.

4.5 3-D noise analysis

D'Agostino and Webb¹⁶ defined directional averaging operators that allow the mathematical derivation of eight noise components from a sensor noise data set. The model defines a temporal dimension (t) representing the framing sequence, and vertical (v) and horizontal (h) dimensions which provide the spatial information. The operators average the data in a specified direction, aiding in the calculation of the noise components. The noise descriptors describe temporal and spatial noise with respect to pixel, row, column and frame variations.

A three-dimensional numpy array with shape (frames,rows,columns) facilitates the 3-D noise calculation, using numpy array manipulation and statistics routines. Sample output from a *.ptw file from the worked example (web site) in this module is given in Figure 6.

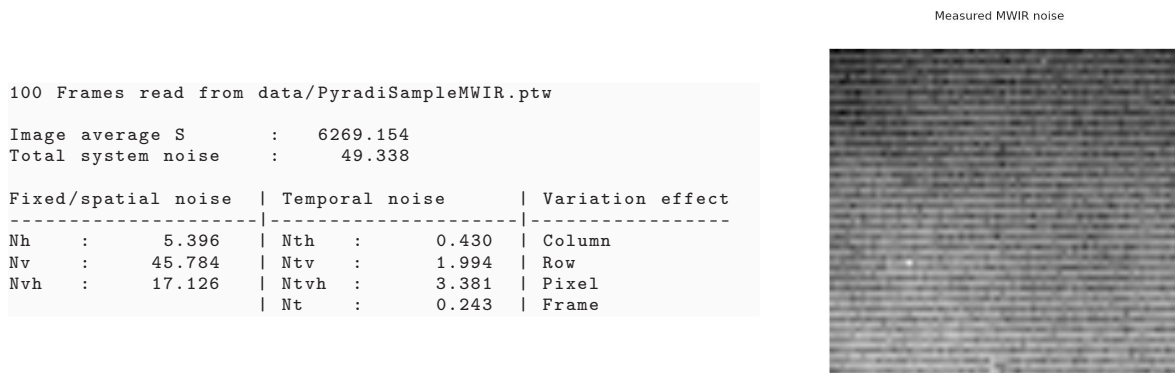


Figure 6: 3-D noise calculation on measured MWIR noise images.

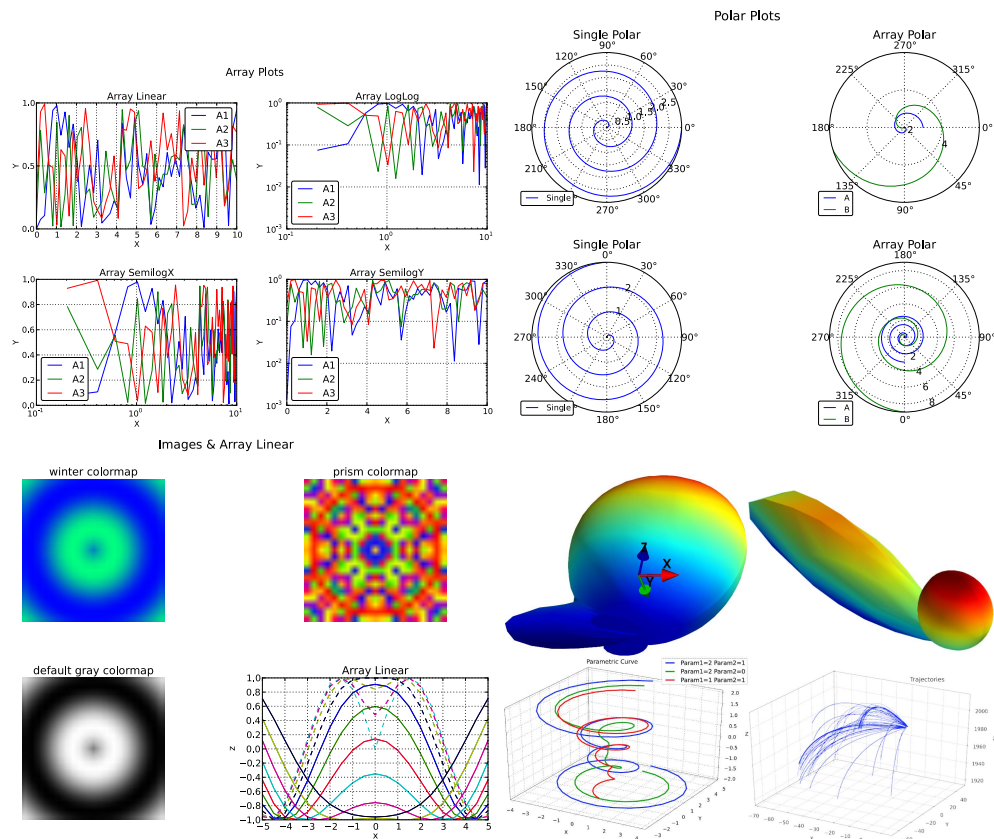


Figure 7: Cartesian, polar, image and 3-D plots.

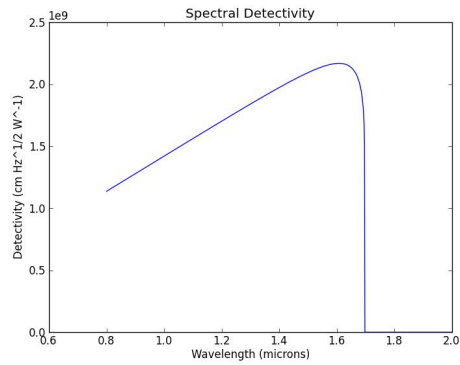
4.6 Data plotting

Figure 7 shows a variety of cartesian, polar, 3-D and image plots. Note the polar plot rotational direction and zero offset variations. All 2-D plotting routines support the plotting of an arbitrary number of multiple lines, for array data. Image data can be calculated in code or read in from data files. 3-D plots are done with Matplotlib (line graphs) and Mayavi (iso-curves).

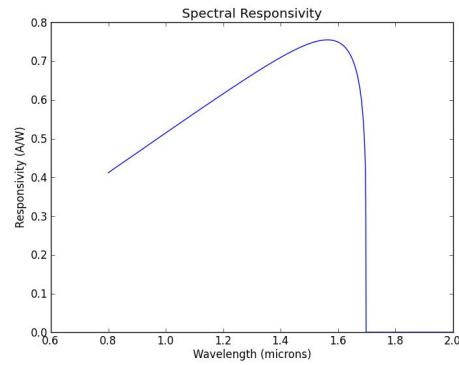
4.7 Detector Modelling

The detector modelling module provides a comprehensive detector model, based on physical design parameters, giving the user accurate control over key model behaviour elements. Infrared detectors can be designed to be photodetectors or thermal detectors. In this case, the work focuses on bulk material photovoltaic infrared photodetectors. This kind of detector is often called photodiode because it is a semiconductor diode (p-n junction) which is light sensitive.¹⁷ In this model, a single element infrared detector is simulated using classical parameters as inputs for the calculation, such as detector area, bandgap energy, doping, and detector temperature among others. In the source code, all the equations and resulting functions are described and referenced in order to allow the user to understand the procedure step by step. It is important to note that all the equations and parameters used are from classical models found in the literature, giving generality and reliability to the procedure. The model provides a spectral response and several figures of merit, such as the detectivity and I×V characteristics.

Figure 8 presents the detectivity and responsivity calculated by the model for a Ge detector with a detector area of $100 \times 100 \mu\text{m}^2$, a detector temperature equal to 80 K, background temperature equals to 280 K and a positive doping of $1 \times 10^{22} \text{ cm}^{-3}$. The model is in good agreement with Ge detector measurements done at LabGE/ITA. The estimated error is around 10% when compared with the peak values for responsivity and detectivity.



(a) Ge detector detectivity.



(b) Ge detector responsivity.

Figure 8: Physics-based detector parameter calculations.

4.8 Colour coordinate calculations

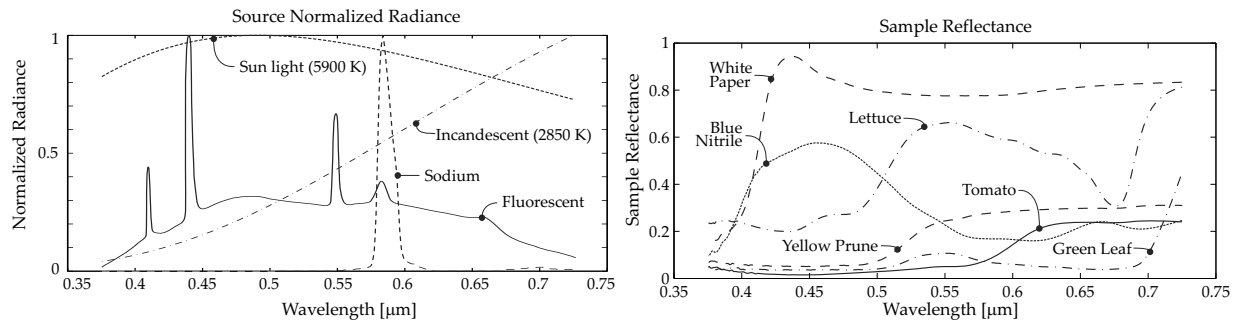


Figure 9: Normalised source radiance and sample reflection.

The example code in the `rychroma.py` file calculates colour coordinates as an application of radiometric normalising, rather than the human perception of colour. Four sources are considered, with normalised spectra shown in Figure 9. The first light source is a ‘daylight’ fluorescent phosphor, the second source is the sun, modelled as a thermal radiator at 5900 K, the third source is an incandescent light globe at a temperature of 2850 K and the fourth source is a low pressure sodium lamp. The samples illuminated by the sources are a red tomato, lettuce, a yellow prune, a dark green leaf, a blue Nitrile (latex-like) surgical glove and standard white printing paper. Figure 9 shows the spectral reflectance of the samples. These diffuse reflection spectra were measured with an ASD spectroradiometer, illuminating the sample with a bright light at short distance. The fruit samples all demonstrated considerable light propagation deeper into the fruit. The blue glove was located on top of a Spectralon white reference (note the considerable ‘white’ reflectance beyond 0.55 μm). The colour coordinates of the samples, in the different source spectra are shown in Figure 10. Note how, under the near-monochromatic sodium illumination, all sample colour coordinates converge to the same colour, that of the source.

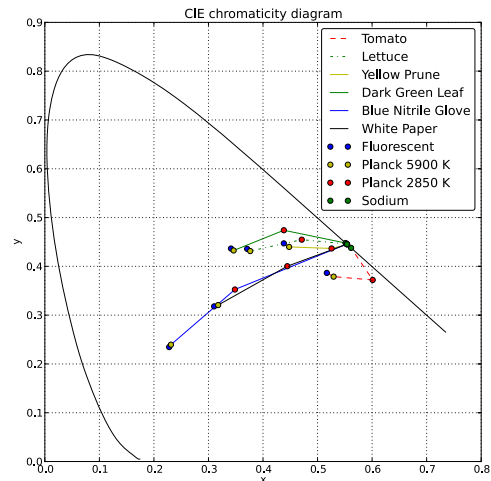
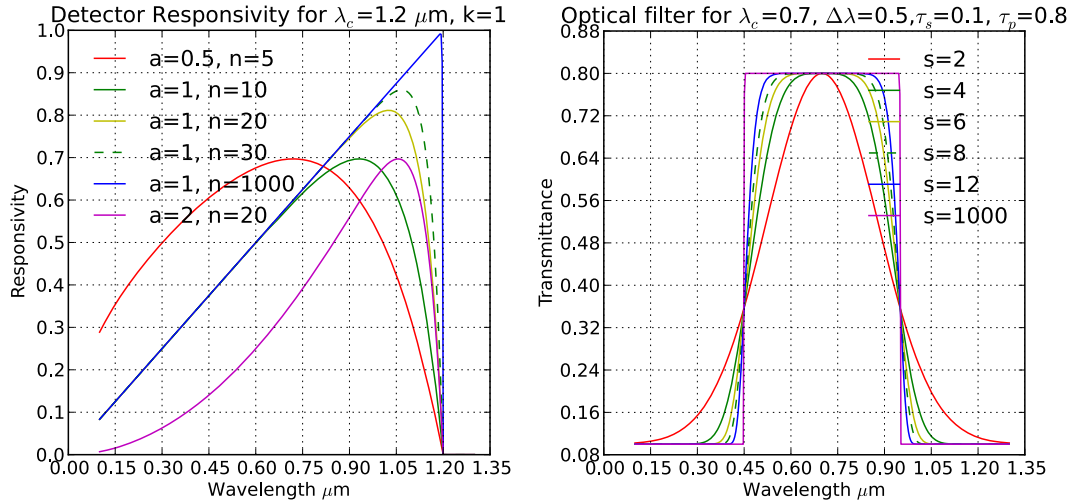


Figure 10: Example of colour coordinate calculations.

4.9 Utility Functions



(a) Photon detector spectral response curves.

(b) Filter spectral transmittance curves.

Figure 11: Example of spectral filter and detector spectral shapes.

A number of utility functions are provided. The utility functions include a simple mathematical model for the spectral responsivity of photon detectors and spectral transmittance of optical filters. A function is also provided to calculate the effective value according to Equation 7. Functions are provided for conversion between spectral density values (i.e. between $[W/(m^2 \cdot \mu m)]$, $[W/(m^2 \cdot cm^{-1})]$, and $[W/(m^2 \cdot Hz)]$). Similar functions are also provided for the conversion between spectral values (i.e. between $[\mu m]$, $[cm^{-1}]$, and $[Hz]$).

Generic detector spectral responsivity curves are calculated with Equation 10:

$$\mathcal{R}_\lambda = k \left[\left(\frac{\lambda}{\lambda_c} \right)^a - \left(\frac{\lambda}{\lambda_c} \right)^n \right] \quad \text{for } \mathcal{R}_\lambda > 0, \quad 0 \quad \text{otherwise,} \quad (10)$$

where $0 \leq a \leq 5$, $0 \leq \lambda \leq \lambda_c$ and $5 \leq n \leq 50$, and where k is a scaling constant. Figure 11a shows a few typical curves. Clearly, these are only approximations, measured detector spectral responsivity curves should be used if available.

Generic filter spectral transmittance can typically be fit to curves of the following form

$$\tau_\lambda = \tau_s + \tau_p \exp \left[- \left(\frac{2(\lambda - \lambda_c)}{\Delta\lambda} \right)^s \right] \quad (11)$$

where τ_s is the filter transmittance in the suppression spectral range, τ_p is the peak transmittance, s is a factor defining the sharpness of the filter cutoff (if $s = 2$ the curve has a gaussian shape and if $s = \infty$ the curve is square), $\Delta\lambda$ defines the width of the pass-band and λ_c is the center wavelength of the pass-band. Figure 11b illustrates a number of filter curves.

5. AVAILABILITY, DOCUMENTATION, AND CONTRIBUTING

Pyradi is managed in accordance with general practice in the open source community. Pyradi is made available under the terms of the Mozilla Public License 1.1.¹⁸ Authors retain ownership of their respective contributions, but make it available for use by other users. User can benefit freely from the original authors' work, but any modifications such users make must be released under compatible terms.

Pyradi is stored in a subversion repository on Google Code, at <http://code.google.com/p/pyradi>. The toolkit files can be downloaded one-by-one using a web browser, or by using a download tool.¹⁹ It is recommended

however, that a subversion client²⁰ be used to ‘check out’ and ‘commit’ the code (instructions are given on the web site).

Potential users are encouraged to join in, and commit changes, updates and new contributions.

Pyradi documentation is available at http://pyradi.googlecode.com/svn/trunk/doc/_build/html/index.html. Documentation is extracted from the comments in the Python code files, using the Sphinx documentation generator. At the moment there is no User Guide, but complete commented example code is included at the end of each module file.

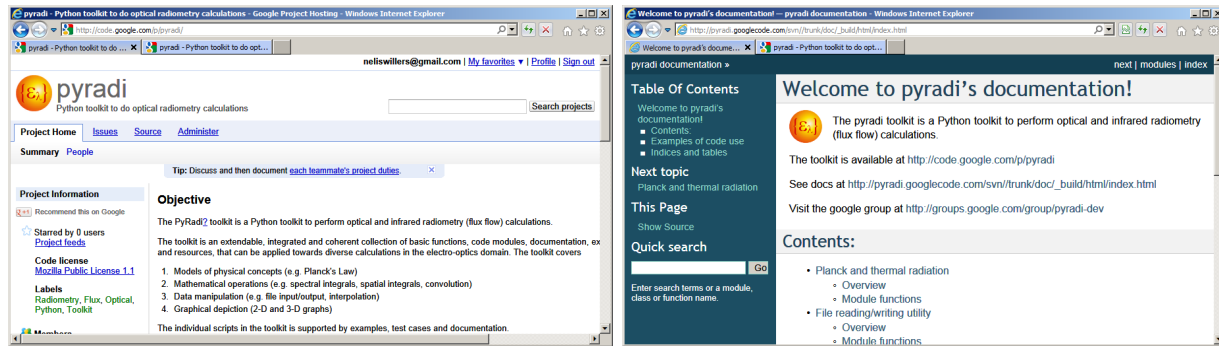


Figure 12: Pyradi repository and documentation web sites.

Pyradi has only moderate hardware requirements and was tested under the Windows and Linux operating systems. It should run in any environment where the Python tools are supported. At the time of writing the following software versions were used: Python 2.7.3, Numpy 1.6.1, Scipy 0.10.1, Matplotlib 1.1, Mayavi 4.1. Some users may find it a challenge to obtain and install some of these modules. The web site provides more information on preparing the Python environment for pyradi.

6. FUTURE WORK

The current focus in the toolkit is on providing a strong set of functions suitable for application in user’s scripts. Graphical user interfaces, using PyQt, are used extensively in our work, but these are mostly project specific. GUI tools may be added to pyradi later. Such tools may include a GUI for editing MODTRAN tape5 files, or viewing/processing *.ptw files.

The analysis of measured radiometric data constitutes a significant portion of the pyradi team’s work. The intent is to rewrite current Matlab code into pyradi modules, some of which, will be added to the repository.

At current pyradi is not yet available as an installable Python package. Users are recommended to checkout pyradi from the subversion repository at Google Code. This approach ensures that users can easily update to the latest version. If there is sufficient interest, pyradi may be released in the form of an installation package.

Pyradi is an ongoing project, supporting activities in the original development teams’ respective laboratories. Pyradi is also used to support the worked examples in an upcoming book.²

7. CONCLUSION

The pyradi toolkit is intended to provide researchers with a set of tools to simplify complex radiometry calculations and data visualisation. It is planned that the toolkit will grow, from its current modest beginning, with the addition of new functionality in future.

The application value of the toolkit is already evident in our respective laboratories, inviting and accelerating new tool development. Pyradi is offered as an open source product with the hope that others will also benefit from our work.

ACKNOWLEDGMENTS

The authors wish to thank FLIR Advanced Thermal Solutions for the permission to publicly release our Python version of the *.ptw file reader. Please note that the copyright to the proprietary *.ptw file format remains the property of FLIR Inc. Thanks also to Derek Griffith who performed the sample colour measurements with the ASD.

REFERENCES

- [1] “Unix Philosophy.” http://en.wikipedia.org/wiki/Unix_philosophy (August 2012).
- [2] Willers, C. J., [*Advanced Radiometry: Techniques for Real-World Applications*], SPIE Press (2013).
- [3] Planckian locus. http://en.wikipedia.org/wiki/Planckian_locus (2012).
- [4] CIE 1931 color space.
http://en.wikipedia.org/wiki/CIE_1931_color_space (2012).
- [5] FLIR Inc. <http://www.flir.com/cs/emea/en/view/?id=41702> (2012).
- [6] Matlab. The MathWorks Inc, <http://www.mathworks.com> (2011).
- [7] SciLab. <http://www.scilab.org> (2011).
- [8] Octave (2011). <http://www.gnu.org/software/octave>.
- [9] The Python computer language. <http://www.python.org/> (2011).
- [10] Numpy. <http://numpy.scipy.org/> (2011).
- [11] Matplotlib. <http://matplotlib.sourceforge.net/> (2011).
- [12] Mayavi, “The MayaVi data visualizer.” <http://mayavi.sourceforge.net/> (2012).
- [13] PyQt. <http://www.riverbankcomputing.com/software/pyqt/intro> (2012).
- [14] PySide. <http://www.pyside.org/> (2012).
- [15] Martelli, A. and Ascher, D., [*Python Cookbook*], O’Reilly (2002).
- [16] D’Agostino, J. A. and Webb, C. M., “Three-dimensional analysis framework and measurement methodology for imaging system noise,” in [*Infrared Imaging Systems: Design, Analysis, Modeling, and Testing II*], Holst, G. C., ed., **1488**, SPIE (1991).
- [17] Dereniak, E. L. and Boreman, G. D., [*Infrared Detectors and Systems*], John Wiley & Sons (1996).
- [18] Mozilla Public License. <http://www.mozilla.org/MPL/> (2012).
- [19] Downloadsvn. <http://downloadsvn.codeplex.com/> (2012).
- [20] Tortoise subversion client. <http://tortoisesvn.net/> (2012).