A 5.0 Spectral Mapping

1. Lewis and M. van Leeuwen

UCL/NCEO

31st May 2016

# A5.1 Introduction

The QA4ECV 'A' approach aims to provide an optimal estimate of the parameters of a linear model (and their uncertainty) that allow estimation of albedo are related reflectance terms at over set of spectral basis functions. In the original work, we defined three broad waveband responses for albedo, being Visible, Near Infrared , and total shortwave. In this section, we demonstrate how this can be extended to other spectral bases.

The algorithm requires that datasets from the set of heterogeneous sensors are mapped to a common spectral basis. This is permissible because we are using linear models throughout, i.e. for spectral and angular modelling. In this section, we derive an initial estimate of appropriate spectral mapping functions using a spectral database provided by DLR.

This involves:

1. Applying bandpass functions to the spectra to simulate spectra for particular sensors

# A5.2 The database spectra

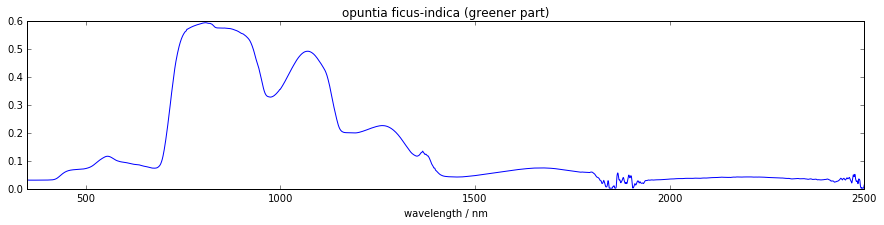
The spectral database used here is the same as that used in a precious ESA ADAM study.

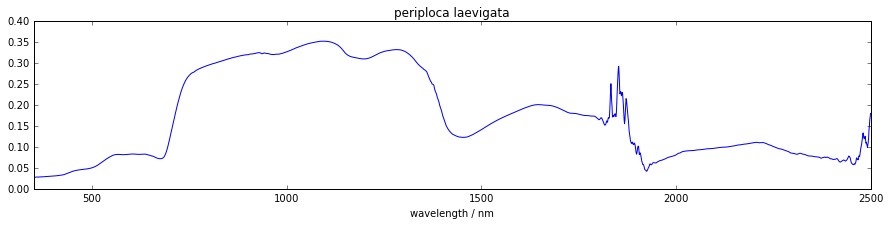
This is intended to provide a proof of concept of the approach, and will ideally be replaced by a wider dataset at a later point.

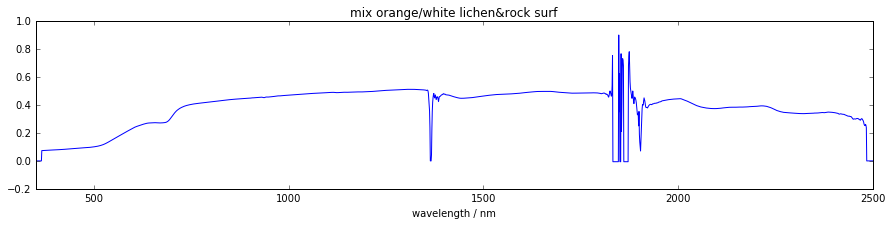
We can examine the types of data contained in the database:

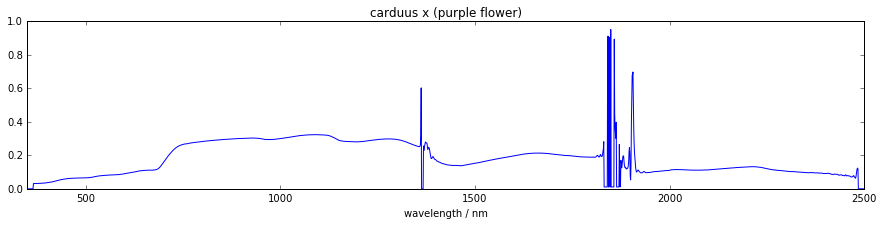
# read the vegetation spectra from DLR database  
spectra\_fn = './../dump/spectra/Spectres.sav'  
spectra = scipy.io.readsav(spectra\_fn)  
swl = spectra.wl  
spect = spectra.spectres  
types = spectra.type  
  
print types

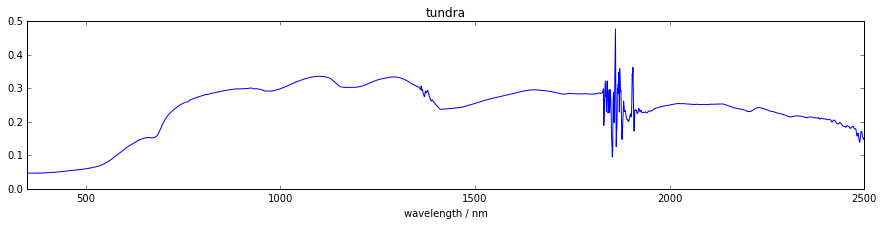
for i in [10,15,33,45,-18]:  
 plt.figure(figsize=(15,3))  
 plt.plot(swl,spect[i])  
 plt.xlim(swl[0],swl[-1])  
 plt.xlabel('wavelength / nm')  
 plt.title(types[i])











# A5.3 Applying bandpass functions to the database spectra.

We develop an integration function that takes two dictionaries where the first holds a spectrum, e.g. of vegetation, and the other holds the bandpass

import numpy as np  
from scipy.interpolate import interp1d  
import scipy, scipy.io  
import matplotlib.pyplot as plt  
import os, glob, pickle  
  
import numpy as np  
import scipy  
import matplotlib.pyplot as plt  
from mpl\_toolkits.mplot3d import Axes3D  
import datetime  
import spectral.io.envi as envi  
  
import pickle  
%matplotlib inline

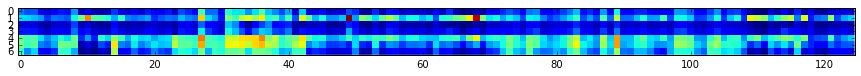
def integrate(spectrum, bandpass, minimum, maximum, dlambda):  
 r = interp1d(spectrum['wavelength'], spectrum['reflectance'], bounds\_error=False, fill\_value=0)  
 b = interp1d(bandpass['wavelength'], bandpass['rsr'], bounds\_error=False, fill\_value=0)  
  
 d = np.arange(minimum, maximum, dlambda)  
 integral = np.sum(r(d) \* b(d) \* dlambda)  
 bsum = np.sum(b(d) \* dlambda)  
 norm = integral / bsum  
 return norm

We will make a dictionary that holds for every sensor (i.e. MERIS, Terra, Aqua, Sentinel 2) an array of m x n where m is the number of vegetation spectra (from DLR) and n is the number of wavebands associated with the sensor. Each element of this array holds the vegetation reflectance integrated over the bandpass, i.e., the relative spectral response curve. In addition, for each sensor we also create a list of centre wavelengths corresponding to the array columns.

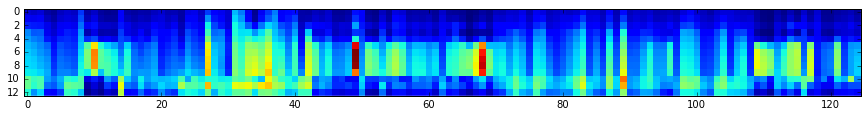
# read the bandpasses  
bandpass\_fn = './../dump/rsr/rsr.dump'  
bandpass\_f = open(bandpass\_fn, 'r')  
rsr = pickle.load(bandpass\_f)  
bandpass\_f.close()  
  
# prepare a dictionary holding arrays of m x n (no. of spectra x no. of bands) for each sensor  
integrated\_spectra = {}  
  
m = np.shape(spect)[0]  
n\_meris = len(rsr['meris']['c1'].keys())  
n\_terra = len(rsr['terra']['c1'].keys())  
n\_aqua = len(rsr['aqua']['c1'].keys())  
n\_sentinel2 = len(rsr['sentinel2']['c1'].keys())  
n\_vgt = len(rsr['vgt']['c1'].keys())  
  
integrated\_spectra['meris'] = np.zeros((m, n\_meris), dtype=float)  
integrated\_spectra['terra'] = np.zeros((m, n\_terra), dtype=float)  
integrated\_spectra['aqua'] = np.zeros((m, n\_aqua), dtype=float)  
integrated\_spectra['sentinel2'] = np.zeros((m, n\_sentinel2), dtype=float)  
integrated\_spectra['vgt'] = np.zeros((m, n\_vgt), dtype=float)  
  
# populate the dictionary with vegetation spectra integrated over the sensor bandpasses  
c = 'c1' # fix the camera/channel  
for sensor in ['meris', 'terra', 'aqua', 'sentinel2', 'vgt']:  
 bands = rsr[sensor][c].keys()  
 # create a new/empty array and centre wavelength list  
 integrefl = np.zeros((m, len(bands)), dtype=float)  
 cwls = []  
 for band in range(1, len(bands)+1 ):  
 wl = rsr[sensor][c]['band%s' % band]['wavelength']  
 relresp = rsr[sensor][c]['band%s' % band]['rsr']  
 cwls.append( rsr[sensor][c]['band%s' % band]['cwl'] )  
 bandpass = {'wavelength': wl, 'rsr': relresp}  
 for i in range(m):  
 refl = spect[i, :]  
 spectrum = {'wavelength': swl, 'reflectance': refl}  
 integrefl[i,band-1] = integrate(spectrum, bandpass, 300, 2500, 1.0)  
  
 cwls = np.asarray(cwls)  
 integrated\_spectra[sensor] = {'integrefl': integrefl, 'cwls': cwls}

And we visualize the results...

sensor = 'terra'  
plt.figure(figsize=(15,3))  
plt.imshow( integrated\_spectra[sensor]['integrefl'].T, interpolation='none' )  
print integrated\_spectra[sensor]['cwls']  
plt.show()

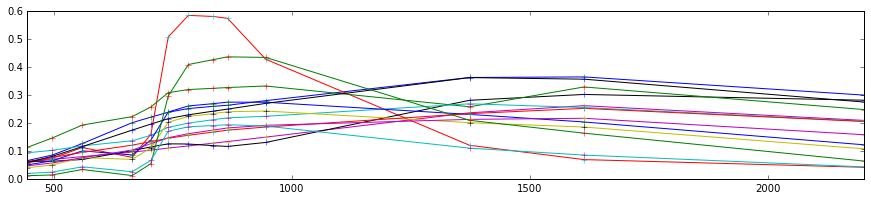


sensor = 'sentinel2'  
# plot in order of band numbers, not necessarily in order of their centre wavelengths...  
plt.figure(figsize=(15,3))  
plt.imshow(integrated\_spectra[sensor]['integrefl'].T, interpolation='none')  
plt.show()



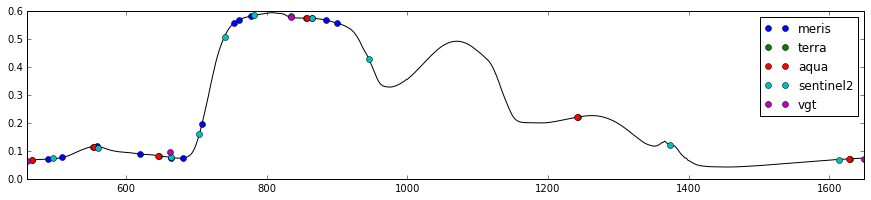
For one sensor, plot the integrated reflectance (i.e. vegetation spectrum integrated over the sensor bandpasses).

sensor = 'sentinel2'  
plt.figure(figsize=(15,3))  
for i in range(0, m, 10): # pick some vegetation spectra  
 # sort the second dimenions of our integrefl array w.r.t. cwls  
 cwls = integrated\_spectra[sensor]['cwls']  
 e = np.argsort(cwls)  
 integrefl = integrated\_spectra[sensor]['integrefl'][i,:]  
 plt.plot( cwls[e], integrefl[e])  
 plt.plot( cwls[e], integrefl[e], '+' )  
plt.xlim(cwls[e][0],cwls[e][-1])  
plt.show()



For one specific vegetation spectrum, plot the reflectance integrated over the sensor bandpasses for all sensor-band combinations.

spect\_no = 10  
plt.figure(figsize=(15,3))  
plt.plot(swl, spect[spect\_no,:], 'k-')  
  
for sensor in ['meris', 'terra', 'aqua', 'sentinel2', 'vgt']:  
 # do some sorting of the second dimension of our integrefl array  
 cwls = integrated\_spectra[sensor]['cwls']  
 e = np.argsort(cwls)  
 integrefl = integrated\_spectra[sensor]['integrefl'][spect\_no,:]  
 cwls = cwls[e]  
 integrefl = integrefl[e]   
 plt.plot(cwls, integrefl, 'o', label=sensor)  
plt.xlim(cwls[e][0],cwls[e][-1])  
plt.legend()  
plt.show()



# save  
fl = open('./../dump/integrated\_spectra/integrated\_spectra.dump', 'w')  
pickle.dump(integrated\_spectra, fl)  
fl.close()

# A5.4 Finding linear mappings for the different sensors

In this section, we develop the set of linear mapping functions.

These are texsted with a subset of the data (27 samples), with the resul used for training.

def fit(X, y):  
 # solve normal equations  
 beta\_hat = (X.T \* X).I \* X.T \* y  
  
 # estimate y given X  
 y\_hat = X \* beta\_hat  
  
 # compute the variance of y given X, var(y|X) = var(epsilon|X)  
 e = y - y\_hat  
 var\_y\_given\_X = np.std(e)\*\*2 # this value is the sum (product?) of the  
 # conditional variance and the variance that results from not knowing  
 # the true values of beta  
 return beta\_hat, var\_y\_given\_X  
  
def evaluate(Xnew, X, beta\_hat, var\_y\_given\_X):  
 # evaluate the model at values in Xnew  
 y\_pred = Xnew \* beta\_hat  
  
 # compute variance of   
 u = var\_y\_given\_X \* (1.0 + Xnew \* (X.T \* X).I \* Xnew.T)  
 var\_pred = np.diag(u)  
  
 return y\_pred, var\_pred  
  
# copied from: http://adorio-research.org/wordpress/?p=1932  
def AIC(RSS, k, n):  
 """  
 Computes the Akaike Information Criterion.  
  
 RSS-residual sum of squares of the fitting errors.  
 k - number of fitted parameters.  
 n - number of observations.  
 """  
 AIC = 2 \* k + n \* (np.log(2 \* np.pi \* RSS/n) + 1)  
 return AIC

fn = './../dump/integrated\_spectra/integrated\_spectra.dump'  
fl = open(fn, 'r')  
integrated\_spectra = pickle.load(fl)  
fl.close()

# Create sets of test and training data  
test\_size = 27  
  
n = 125 # number of observations or spectra in the DLR database  
training\_size = n - test\_size  
spectrum\_numbers = np.arange(n)  
training\_idxs = np.sort( np.random.choice(n, size=training\_size, replace=False ) ).astype(int)  
test\_idxs = np.delete(spectrum\_numbers, training\_idxs)  
  
print 'training', training\_idxs  
print ''  
print 'testing', test\_idxs

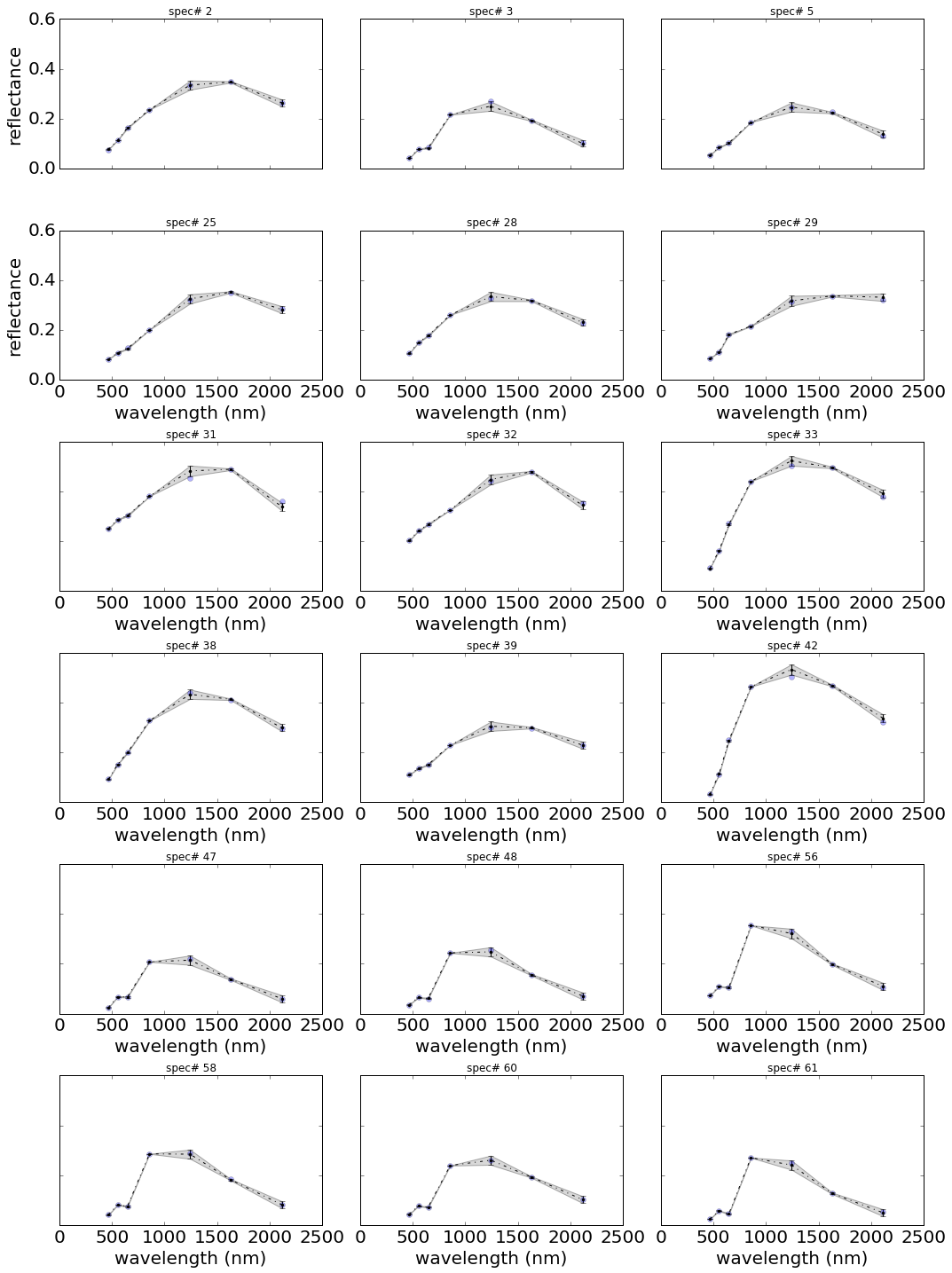
# for a given sensor A  
# ... for a given band  
# ... ... for another given sensor B  
# ... ... ... compute distance (wavelength) between bands of B to the selected band of A  
# ... ... ... build linear models that map bands of B to the band of A, always include bands nearer to the selected band of A  
# ... ... ... compute an information criterion, e.g. Akaike's index  
# ... ... ... store everything in a dictionary format  
  
mappings = {}  
  
sensors = ['terra', 'aqua', 'meris', 'sentinel2', 'vgt']  
for sensorA in sensors:  
 print sensorA  
  
 cwlsA = integrated\_spectra[sensorA]['cwls']  
 mappings[sensorA] = {}  
  
 for bandA, cwlA in enumerate(cwlsA):  
 # get observations y that we want to map to  
 integreflA = integrated\_spectra[sensorA]['integrefl']  
 y = np.matrix( integreflA[training\_idxs,:][:,bandA] ).T  
 mappings[sensorA][bandA] = {}  
  
 for sensorB in sensors:  
 if sensorA == sensorB:  
 continue  
  
 mappings[sensorA][bandA][sensorB] = {}  
  
 # compute distance between band of sensorA and bands of sensorB  
 cwlsB = integrated\_spectra[sensorB]['cwls']  
 dist = cwlsB - cwlA  
 e = np.argsort(dist)  
  
 # build design matrices X with band observation of sensorB  
 integreflB = integrated\_spectra[sensorB]['integrefl']  
 aic\_previous = np.inf  
 for i in range(1, len(e)+1): # test the first five bands  
 bandsB = e[0:i]  
 k = len(bandsB) # number of fitted parameters  
 assert np.shape(integreflB)[0] == np.shape(integreflA)[0]  
 X\_ = integreflB[training\_idxs,:][:,bandsB] # of shape number of training samples x number of bands considered in linear regression  
  
 X = np.matrix( np.hstack(( np.ones(training\_size).reshape((training\_size, 1)), X\_ )) )  
 # should we apply some weighting? and how?  
 beta\_hat, var\_y\_given\_X = fit(X, y)  
 y\_pred, var\_pred = evaluate(X, X, beta\_hat, var\_y\_given\_X)  
 RSS = (y\_pred - y).T \* (y\_pred - y)  
 aic = float(AIC(RSS, k, n))  
  
 if aic < aic\_previous:  
 mappings[sensorA][bandA][sensorB] = {'bandsB':bandsB, 'cwls':cwlsB[bandsB], \  
 'beta\_hat':beta\_hat, 'AIC':aic, 'var\_y\_given\_X': var\_y\_given\_X}  
 aic\_previous = aic

Part of the quality assurance here is that the test samples should mostly lie within the confidence interval.

sensorA = 'sentinel2'  
bandA = 9  
sensorB = 'vgt'  
  
cwlA = integrated\_spectra[sensorA]['cwls'][bandA]  
y = integrated\_spectra[sensorA]['integrefl'][:,bandA]  
  
integreflB = integrated\_spectra[sensorB]['integrefl']  
bandsB = mappings[sensorA][bandA][sensorB]['bandsB']  
var\_y\_given\_X = mappings[sensorA][bandA][sensorB]['var\_y\_given\_X']  
beta\_hat = mappings[sensorA][bandA][sensorB]['beta\_hat']  
  
X\_ = integreflB[:, bandsB]  
X = np.matrix( np.hstack(( np.ones( np.shape(X\_)[0] ).reshape(( np.shape(X\_)[0], 1)), X\_ )) ) # create design matrix  
  
y\_pred, var\_y = evaluate(X, X, beta\_hat, var\_y\_given\_X)  
  
c = 0  
for i in test\_idxs:  
 yp = float(y\_pred[i,0])  
 sigma = np.sqrt(float(var\_y[i]))  
 t = 1.96  
 bln = (y[i] > (yp - (t \* sigma))) and (y[i] < (yp + (t \* sigma)))  
 #print i, y[i], yp - yvar, yp + yvar, bln  
 if bln:  
 c+=1  
  
print 'There were', c, 'positives out of', test\_size, 'or %.2f' % (float(c)/float(test\_size)\*100.0), \  
 '% of observations fell within confidence interval'

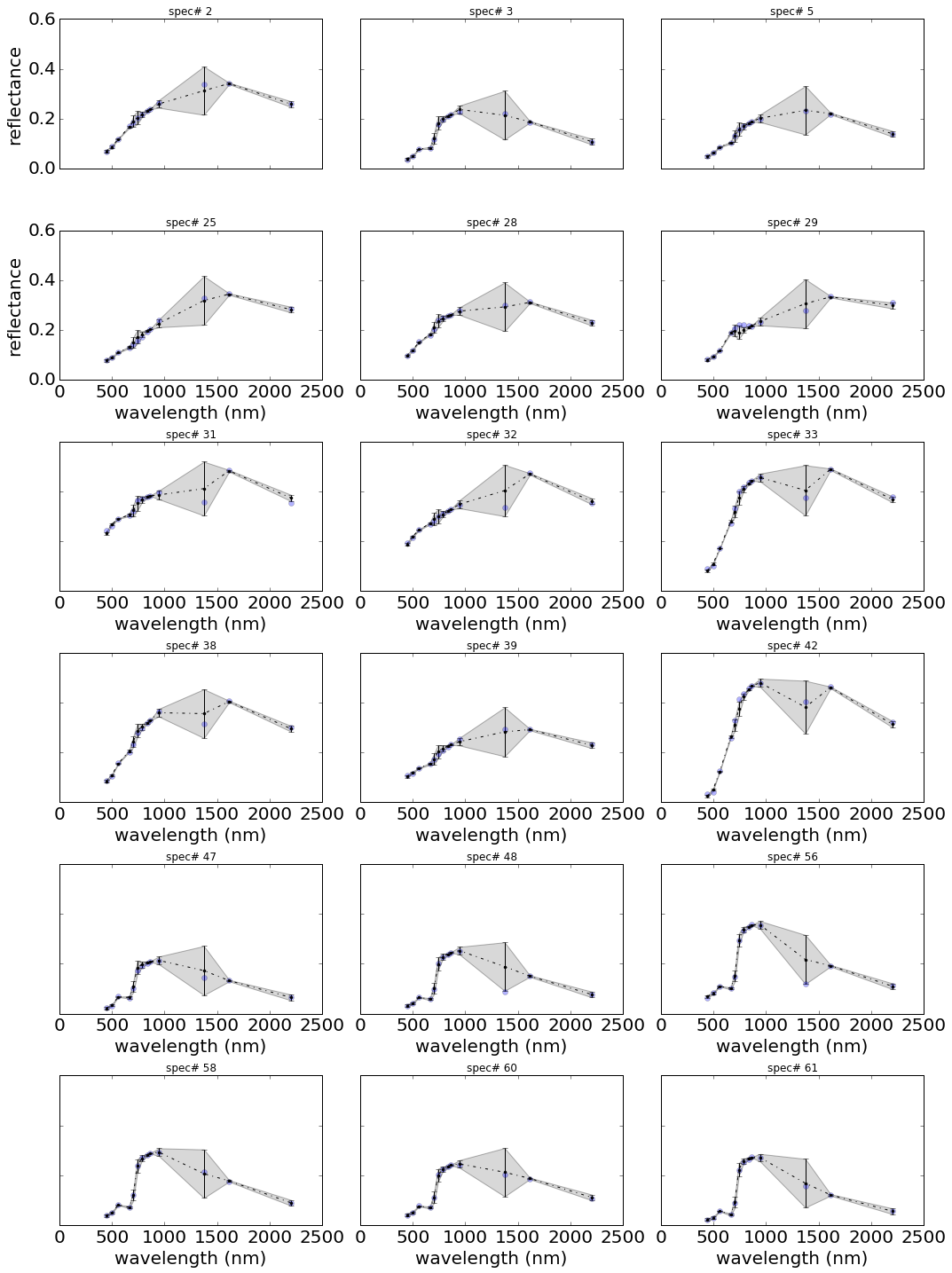
Here, we provide examples of the mapping from Sentinel-2 bands to MODIS:

# define here sensorA and sensorB  
# sensorA is the one to map to using band observations from sensorB  
sensorA = 'terra'  
sensorB = 'sentinel2'  
  
# for 27 randomly selected spectra from the DLR database...  
nrows = 6 # for image display  
ncols = 3 # for image display  
fig = plt.figure( figsize = (15, 20))  
num\_subplots = nrows \* ncols  
  
for i, spectrum\_number in enumerate(test\_idxs[:num\_subplots]):  
 cwlsA = integrated\_spectra[sensorA]['cwls']  
 m = len(cwlsA) # m number of bands in sensorA  
 bandsA = range(m)  
 spectrum = np.zeros((m, 4), dtype=float)  
 for bandA in bandsA:  
 cwlA = integrated\_spectra[sensorA]['cwls'][bandA]  
 y = integrated\_spectra[sensorA]['integrefl'][:,bandA]  
  
 integreflB = integrated\_spectra[sensorB]['integrefl']  
 bandsB = mappings[sensorA][bandA][sensorB]['bandsB']  
 var\_y\_given\_X = mappings[sensorA][bandA][sensorB]['var\_y\_given\_X']  
 beta\_hat = mappings[sensorA][bandA][sensorB]['beta\_hat']  
  
 X\_ = integreflB[:, bandsB]  
 X = np.matrix( np.hstack(( np.ones( np.shape(X\_)[0] ).reshape(( np.shape(X\_)[0], 1)), X\_ )) ) # create design matrix  
  
 y\_pred, var\_y = evaluate(X, X, beta\_hat, var\_y\_given\_X)  
  
 yp = float(y\_pred[spectrum\_number,0]) # y\_pred is a column vector, we still index in full 2 dimensions...  
 sigma = np.sqrt(float(var\_y[spectrum\_number]))  
 t = 1.96  
 y\_lower = yp - (t \* sigma)  
 y\_upper = yp + (t \* sigma)  
  
 spectrum[bandA, :] = y\_lower, yp, y\_upper, sigma  
  
 # plot in order of centre wavelengths  
 e = np.argsort(cwlsA)  
 ax = fig.add\_subplot(nrows, ncols, i+1)  
  
 ax.set\_ylim((0, 0.6))  
 ax.locator\_params(axis='y',nbins=4)  
 if i+1 > 3:  
 for tick in ax.xaxis.get\_major\_ticks():  
 tick.label.set\_fontsize(20)  
 ax.set\_xlabel("wavelength (nm)", fontsize=20)  
 else:  
 ax.set\_xticklabels( () )  
  
  
 if i+1 == 1 or i+1 == 4:  
 for tick in ax.yaxis.get\_major\_ticks():  
 tick.label.set\_fontsize(20)  
 ax.set\_ylabel("reflectance", fontsize=20)  
 else:  
 ax.set\_yticklabels(())  
  
 plt.plot(cwlsA[e], integrated\_spectra[sensorA]['integrefl'][spectrum\_number, e], 'o', alpha=0.3)  
 plt.errorbar(cwlsA[e], spectrum[e,1], yerr=1.96\*spectrum[e,3], fmt='.', color='black')  
 # in case you may want to interpolate between bands...  
 plt.plot(cwlsA[e], spectrum[e,1], 'k-.')  
 plt.fill\_between(cwlsA[e], spectrum[e,0], spectrum[e,2], facecolor='grey', alpha = 0.3)  
 plt.title('spec# %s' % spectrum\_number)  
 plt.locator\_params(nbins=5)  
  
plt.tight\_layout()  
plt.show()



Here, we provide examples of the mapping from MODIS bands to Senstinel 2:

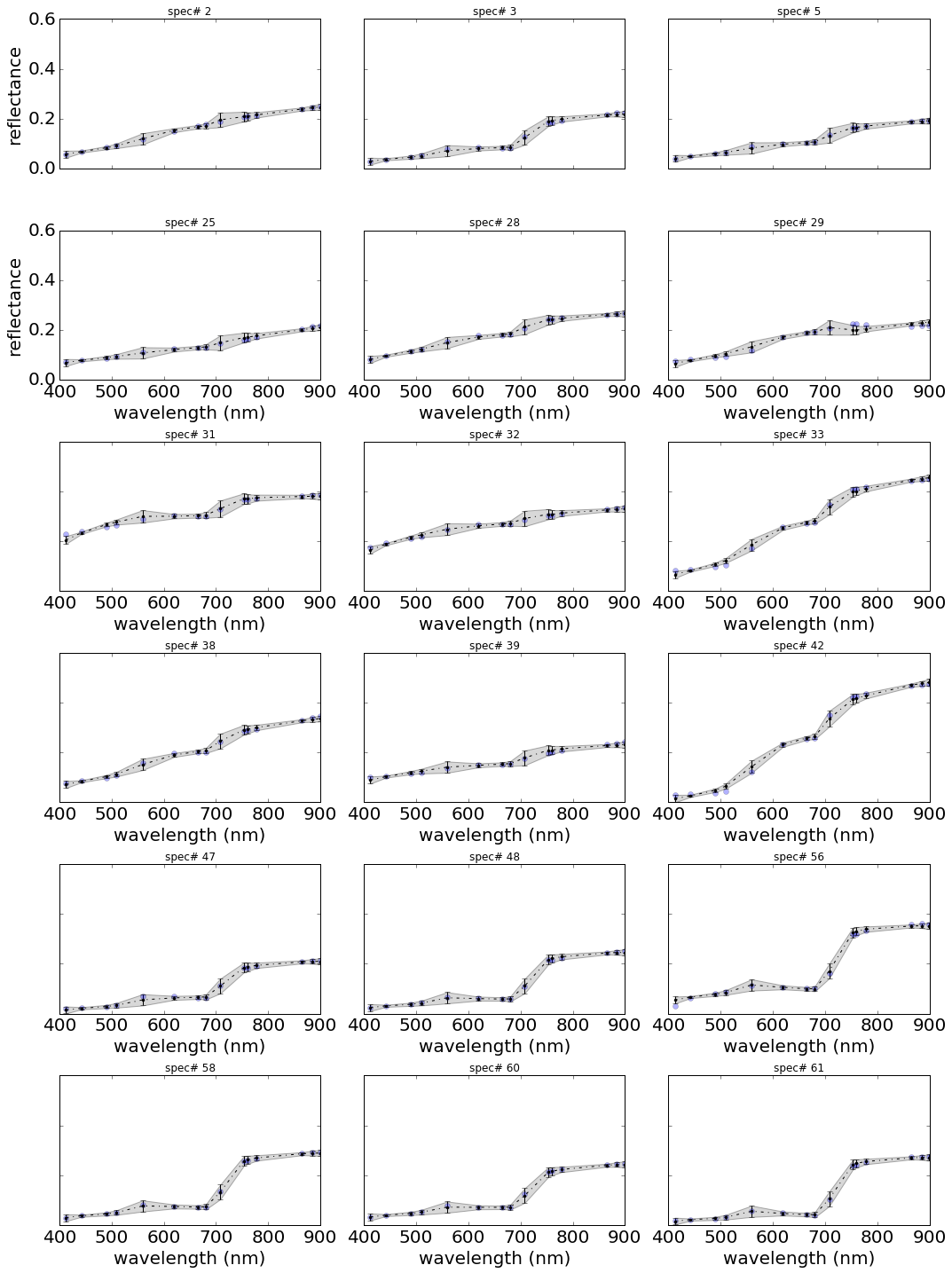
# define here sensorA and sensorB  
# sensorA is the one to map to using band observations from sensorB  
sensorB = 'terra'  
sensorA = 'sentinel2'  
  
# for 27 randomly selected spectra from the DLR database...  
nrows = 6 # for image display  
ncols = 3 # for image display  
fig = plt.figure( figsize = (15, 20))  
num\_subplots = nrows \* ncols  
  
for i, spectrum\_number in enumerate(test\_idxs[:num\_subplots]):  
 cwlsA = integrated\_spectra[sensorA]['cwls']  
 m = len(cwlsA) # m number of bands in sensorA  
 bandsA = range(m)  
 spectrum = np.zeros((m, 4), dtype=float)  
 for bandA in bandsA:  
 cwlA = integrated\_spectra[sensorA]['cwls'][bandA]  
 y = integrated\_spectra[sensorA]['integrefl'][:,bandA]  
  
 integreflB = integrated\_spectra[sensorB]['integrefl']  
 bandsB = mappings[sensorA][bandA][sensorB]['bandsB']  
 var\_y\_given\_X = mappings[sensorA][bandA][sensorB]['var\_y\_given\_X']  
 beta\_hat = mappings[sensorA][bandA][sensorB]['beta\_hat']  
  
 X\_ = integreflB[:, bandsB]  
 X = np.matrix( np.hstack(( np.ones( np.shape(X\_)[0] ).reshape(( np.shape(X\_)[0], 1)), X\_ )) ) # create design matrix  
  
 y\_pred, var\_y = evaluate(X, X, beta\_hat, var\_y\_given\_X)  
  
 yp = float(y\_pred[spectrum\_number,0]) # y\_pred is a column vector, we still index in full 2 dimensions...  
 sigma = np.sqrt(float(var\_y[spectrum\_number]))  
 t = 1.96  
 y\_lower = yp - (t \* sigma)  
 y\_upper = yp + (t \* sigma)  
  
 spectrum[bandA, :] = y\_lower, yp, y\_upper, sigma  
  
 # plot in order of centre wavelengths  
 e = np.argsort(cwlsA)  
 ax = fig.add\_subplot(nrows, ncols, i+1)  
  
 ax.set\_ylim((0, 0.6))  
 ax.locator\_params(axis='y',nbins=4)  
 if i+1 > 3:  
 for tick in ax.xaxis.get\_major\_ticks():  
 tick.label.set\_fontsize(20)  
 ax.set\_xlabel("wavelength (nm)", fontsize=20)  
 else:  
 ax.set\_xticklabels( () )  
  
  
 if i+1 == 1 or i+1 == 4:  
 for tick in ax.yaxis.get\_major\_ticks():  
 tick.label.set\_fontsize(20)  
 ax.set\_ylabel("reflectance", fontsize=20)  
 else:  
 ax.set\_yticklabels(())  
  
 plt.plot(cwlsA[e], integrated\_spectra[sensorA]['integrefl'][spectrum\_number, e], 'o', alpha=0.3)  
 plt.errorbar(cwlsA[e], spectrum[e,1], yerr=1.96\*spectrum[e,3], fmt='.', color='black')  
 # in case you may want to interpolate between bands...  
 plt.plot(cwlsA[e], spectrum[e,1], 'k-.')  
 plt.fill\_between(cwlsA[e], spectrum[e,0], spectrum[e,2], facecolor='grey', alpha = 0.3)  
 plt.title('spec# %s' % spectrum\_number)  
 plt.locator\_params(nbins=5)  
  
plt.tight\_layout()  
plt.show()



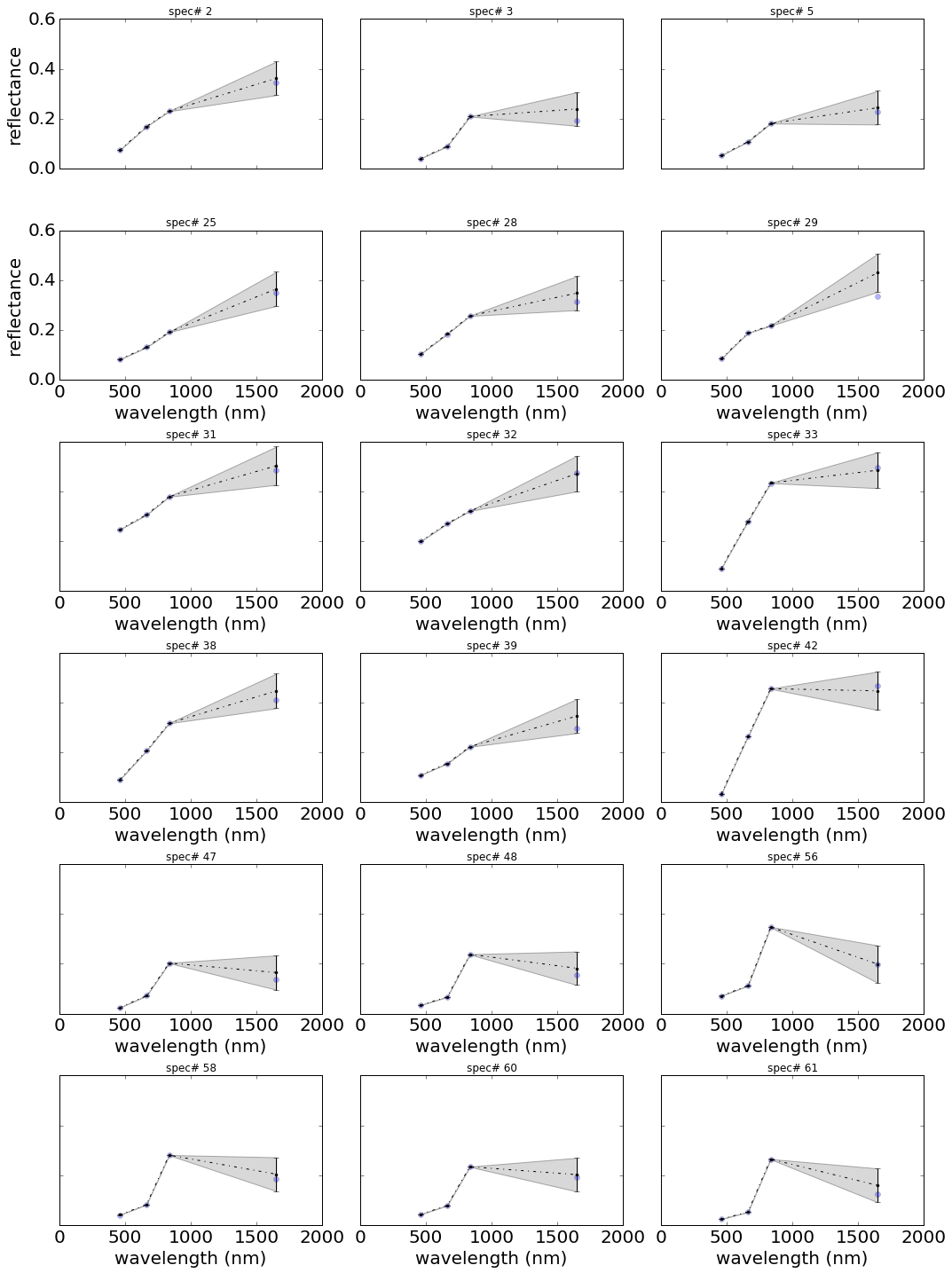
The illustrations show that with linear mapping functions derived from this DLR spectral database, we can derive suitable mappings and uncertainty quantification between any pairs of sensors. In the vase of Sentinel-2 (S2) to MODIS, we note the relatively large uncertainties in predicting MODIS bands 5 and 7 from S2 sampling, but that all uncertainties arising from this are really quite small.

In the case of MODIS to S2 mapping, it is (again) only in the SWIR that uncertainties become quite high.

# define here sensorA and sensorB  
# sensorA is the one to map to using band observations from sensorB  
sensorB = 'vgt'  
sensorA = 'meris'  
  
# for 27 randomly selected spectra from the DLR database...  
nrows = 6 # for image display  
ncols = 3 # for image display  
fig = plt.figure( figsize = (15, 20))  
num\_subplots = nrows \* ncols  
  
for i, spectrum\_number in enumerate(test\_idxs[:num\_subplots]):  
 cwlsA = integrated\_spectra[sensorA]['cwls']  
 m = len(cwlsA) # m number of bands in sensorA  
 bandsA = range(m)  
 spectrum = np.zeros((m, 4), dtype=float)  
 for bandA in bandsA:  
 cwlA = integrated\_spectra[sensorA]['cwls'][bandA]  
 y = integrated\_spectra[sensorA]['integrefl'][:,bandA]  
  
 integreflB = integrated\_spectra[sensorB]['integrefl']  
 bandsB = mappings[sensorA][bandA][sensorB]['bandsB']  
 var\_y\_given\_X = mappings[sensorA][bandA][sensorB]['var\_y\_given\_X']  
 beta\_hat = mappings[sensorA][bandA][sensorB]['beta\_hat']  
  
 X\_ = integreflB[:, bandsB]  
 X = np.matrix( np.hstack(( np.ones( np.shape(X\_)[0] ).reshape(( np.shape(X\_)[0], 1)), X\_ )) ) # create design matrix  
  
 y\_pred, var\_y = evaluate(X, X, beta\_hat, var\_y\_given\_X)  
  
 yp = float(y\_pred[spectrum\_number,0]) # y\_pred is a column vector, we still index in full 2 dimensions...  
 sigma = np.sqrt(float(var\_y[spectrum\_number]))  
 t = 1.96  
 y\_lower = yp - (t \* sigma)  
 y\_upper = yp + (t \* sigma)  
  
 spectrum[bandA, :] = y\_lower, yp, y\_upper, sigma  
  
 # plot in order of centre wavelengths  
 e = np.argsort(cwlsA)  
 ax = fig.add\_subplot(nrows, ncols, i+1)  
  
 ax.set\_ylim((0, 0.6))  
 ax.locator\_params(axis='y',nbins=4)  
 if i+1 > 3:  
 for tick in ax.xaxis.get\_major\_ticks():  
 tick.label.set\_fontsize(20)  
 ax.set\_xlabel("wavelength (nm)", fontsize=20)  
 else:  
 ax.set\_xticklabels( () )  
  
  
 if i+1 == 1 or i+1 == 4:  
 for tick in ax.yaxis.get\_major\_ticks():  
 tick.label.set\_fontsize(20)  
 ax.set\_ylabel("reflectance", fontsize=20)  
 else:  
 ax.set\_yticklabels(())  
  
 plt.plot(cwlsA[e], integrated\_spectra[sensorA]['integrefl'][spectrum\_number, e], 'o', alpha=0.3)  
 plt.errorbar(cwlsA[e], spectrum[e,1], yerr=1.96\*spectrum[e,3], fmt='.', color='black')  
 # in case you may want to interpolate between bands...  
 plt.plot(cwlsA[e], spectrum[e,1], 'k-.')  
 plt.fill\_between(cwlsA[e], spectrum[e,0], spectrum[e,2], facecolor='grey', alpha = 0.3)  
 plt.title('spec# %s' % spectrum\_number)  
 plt.locator\_params(nbins=5)  
  
plt.tight\_layout()  
plt.show()



# define here sensorA and sensorB  
# sensorA is the one to map to using band observations from sensorB  
sensorA = 'vgt'  
sensorB = 'meris'  
  
# for 27 randomly selected spectra from the DLR database...  
nrows = 6 # for image display  
ncols = 3 # for image display  
fig = plt.figure( figsize = (15, 20))  
num\_subplots = nrows \* ncols  
  
for i, spectrum\_number in enumerate(test\_idxs[:num\_subplots]):  
 cwlsA = integrated\_spectra[sensorA]['cwls']  
 m = len(cwlsA) # m number of bands in sensorA  
 bandsA = range(m)  
 spectrum = np.zeros((m, 4), dtype=float)  
 for bandA in bandsA:  
 cwlA = integrated\_spectra[sensorA]['cwls'][bandA]  
 y = integrated\_spectra[sensorA]['integrefl'][:,bandA]  
  
 integreflB = integrated\_spectra[sensorB]['integrefl']  
 bandsB = mappings[sensorA][bandA][sensorB]['bandsB']  
 var\_y\_given\_X = mappings[sensorA][bandA][sensorB]['var\_y\_given\_X']  
 beta\_hat = mappings[sensorA][bandA][sensorB]['beta\_hat']  
  
 X\_ = integreflB[:, bandsB]  
 X = np.matrix( np.hstack(( np.ones( np.shape(X\_)[0] ).reshape(( np.shape(X\_)[0], 1)), X\_ )) ) # create design matrix  
  
 y\_pred, var\_y = evaluate(X, X, beta\_hat, var\_y\_given\_X)  
  
 yp = float(y\_pred[spectrum\_number,0]) # y\_pred is a column vector, we still index in full 2 dimensions...  
 sigma = np.sqrt(float(var\_y[spectrum\_number]))  
 t = 1.96  
 y\_lower = yp - (t \* sigma)  
 y\_upper = yp + (t \* sigma)  
  
 spectrum[bandA, :] = y\_lower, yp, y\_upper, sigma  
  
 # plot in order of centre wavelengths  
 e = np.argsort(cwlsA)  
 ax = fig.add\_subplot(nrows, ncols, i+1)  
  
 ax.set\_ylim((0, 0.6))  
 ax.locator\_params(axis='y',nbins=4)  
 if i+1 > 3:  
 for tick in ax.xaxis.get\_major\_ticks():  
 tick.label.set\_fontsize(20)  
 ax.set\_xlabel("wavelength (nm)", fontsize=20)  
 else:  
 ax.set\_xticklabels( () )  
  
  
 if i+1 == 1 or i+1 == 4:  
 for tick in ax.yaxis.get\_major\_ticks():  
 tick.label.set\_fontsize(20)  
 ax.set\_ylabel("reflectance", fontsize=20)  
 else:  
 ax.set\_yticklabels(())  
  
 plt.plot(cwlsA[e], integrated\_spectra[sensorA]['integrefl'][spectrum\_number, e], 'o', alpha=0.3)  
 plt.errorbar(cwlsA[e], spectrum[e,1], yerr=1.96\*spectrum[e,3], fmt='.', color='black')  
 # in case you may want to interpolate between bands...  
 plt.plot(cwlsA[e], spectrum[e,1], 'k-.')  
 plt.fill\_between(cwlsA[e], spectrum[e,0], spectrum[e,2], facecolor='grey', alpha = 0.3)  
 plt.title('spec# %s' % spectrum\_number)  
 plt.locator\_params(nbins=5)  
  
plt.tight\_layout()  
plt.show()



Similarly, looking at the mappings from VGT to MERIS, we see that the uncertainty inb mapping from VGT to MERIS is quite small, but when mapping from MERIS to VGT, the uncertainty in the SWIR becomes large. This is because MERIS has no spectral sampling in this region, and the mapping is enavbled simply by correlations in the data.

This is very positive for the approach developed here, as it demonstrates that we can achive sensible spectral mappings between datasets from sensors with different spectral sampling. Where the wavebands of the two sensors are close, we tend to get low uncertainties. In extrapolation (as in the MERIS to VGT case) the uncertainty can be quite high, but that is appropriate: pseudo-observations at such wavelengths will then have low weight in constraining albedo.