GSE 760–S01

Advanced Methods in Geospatial Modeling:

Computation for Remote Sensing Analysis and Product Generation

xiaoyang Zhang, 3/26/2021

Class Schedule

Date	Lecture (Friday)	Date	Exercises (Friday)
Jan 15	**Lecture 1**: Course overview and introduction to remote sensing processing on Linux system	Jan 15	Lab assignment 1- Linux system setup
Jan 22	**Lecture 2**: Getting start with Linux system	Jan 25	Lab assignment 2 - Command line syntax
Jan 29	**Lecture 3:** Linux files and file utilities	Jan 29	Lab assignment 3 – File utilities
Feb 5	**Lecture 4:** File system and processes	Feb 5	Lab assignment 4 – File system and processes
Feb 12	**Lecture 5:** Shell scripting	Feb 12	Lab assignment 5- Shell scripting
Feb 19	**Lecture 6:** Perl scripting (1)	Feb 19	Lab assignment 6- Perl scripting
Feb 26	**Lecture 7:** Perl scripting (2)	Feb 26	Lab assignment 7- Perl scripting
March 5	**Lecture 8:** Python scripting	March 5	Midterm Exam
March 12	**Spring Break**		**Spring Break**
March 19	**Lecture 9:** Satellite data and file format	March 19	Lab assignment 8- Python scripting
March 26	**Lecture 10:** Satellite data processing	March 26	Lab assignment 9- Satellite data processing
April 2	**No class/Easter Recess**	April 2	**No class/Easter Recess**
April 9	**Lecture 11:** Operational product generation	April 9	Lab assignment 10- Programing for product generation
April 16	**Lecture 12:** Software and product documentation	April 16	Project work overview
April 23	Work on projects	April 23	Work on projects
April 30	Work on projects	April 30	Work on projects
May 7	**Lecture 13:** Final presentation		

Final Exam: We will schedule final presentations during the final exam period.

Note: Recommended to readings to accompany each chapter will be assigned on the class D2L site.

Satellite Data Processing

Digital Image Processing

Digital remotely sensed image processing including :

image restoration

Refers to the correction and calibration of images

Image enhancement

predominantly concern with the modification of images to optimize their appearance to the visual system.

Image classification

refers to the computer-assisted interpretation of images

Image transformation

refers to the derivation of new imagery as a result of some mathematical treatment of the raw image bands

Satellite time series analysis

establish temporal variation of vegetation properties

Image restoration

Image Rectification and Registration

Image restoration: **includes *radiometric restoration* and *geometric restoration* in order to achieve as faithful a representation of the earth surface as possible—a fundamental consideration for all applications.**

Radiometric restoration refers to the removal or diminishment of distortions in the degree of electromagnetic energy registered by each detector. A variety of agents can cause distortion in the values recorded for image cells, including:

uniformly elevated values, due to atmospheric haze, which preferentially scatters short wavelength bands;

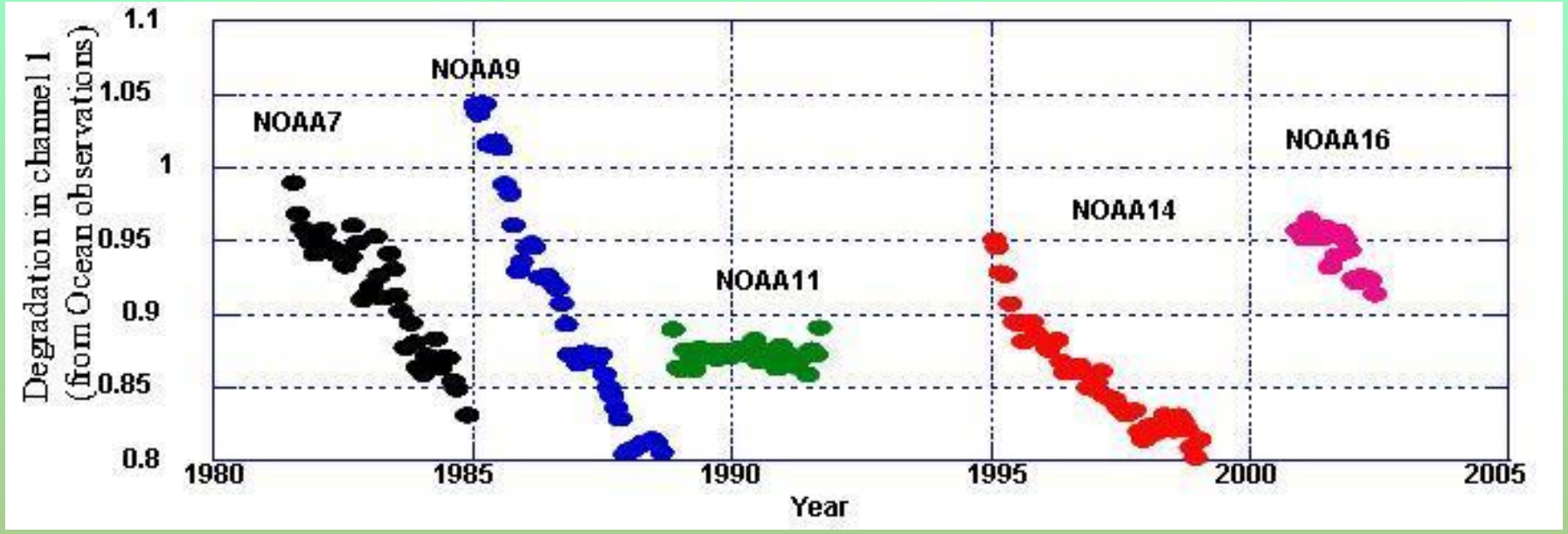
striping, due to detectors going out of calibration;

random noise, due to unpredictable and unsystematic performance of the sensor or transmission of the data; and

scan line drop out, due to signal loss from specific detectors.

Calibration, due to instrument degradation and instrument difference in a long-term data

Example: Data Inconsistence



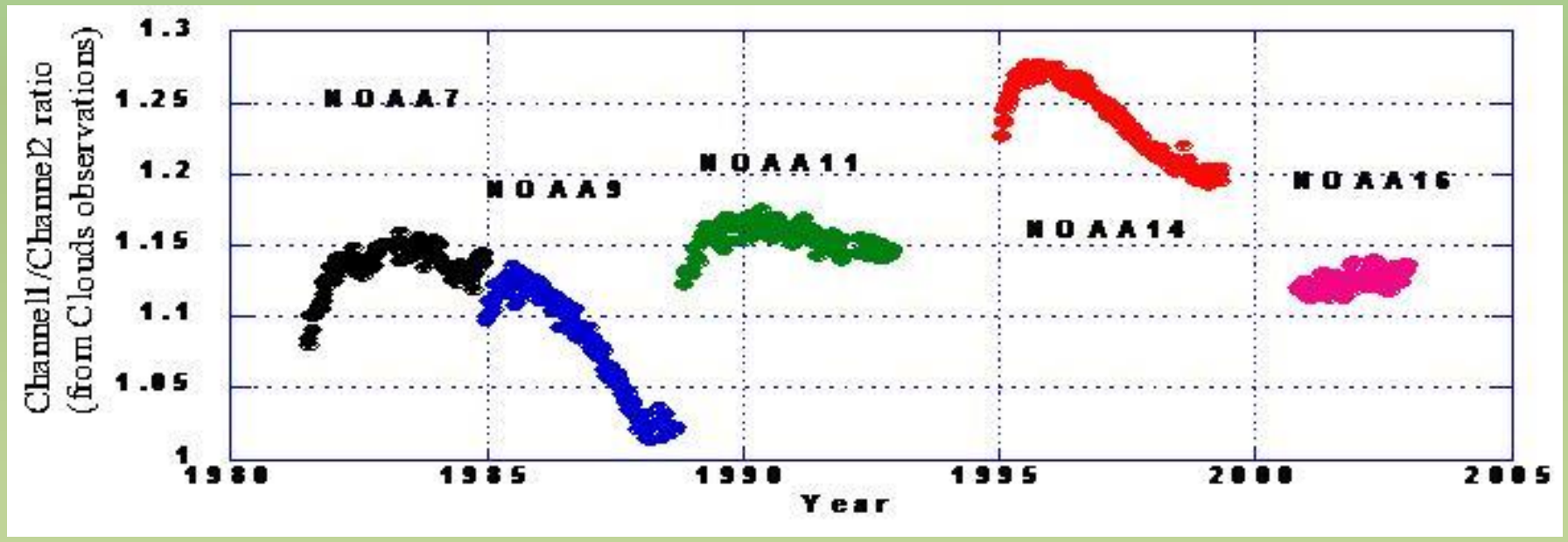


Image restoration

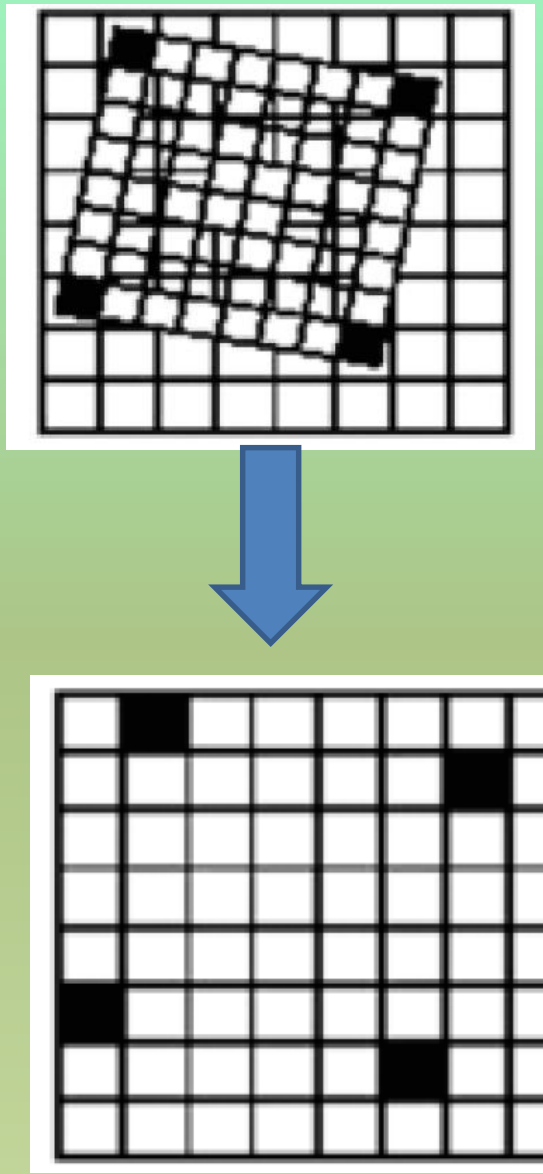
Image Rectification and Registration

Geometric Restoration: Remotely sensed imagery should be accurately registered to the proposed map base through:

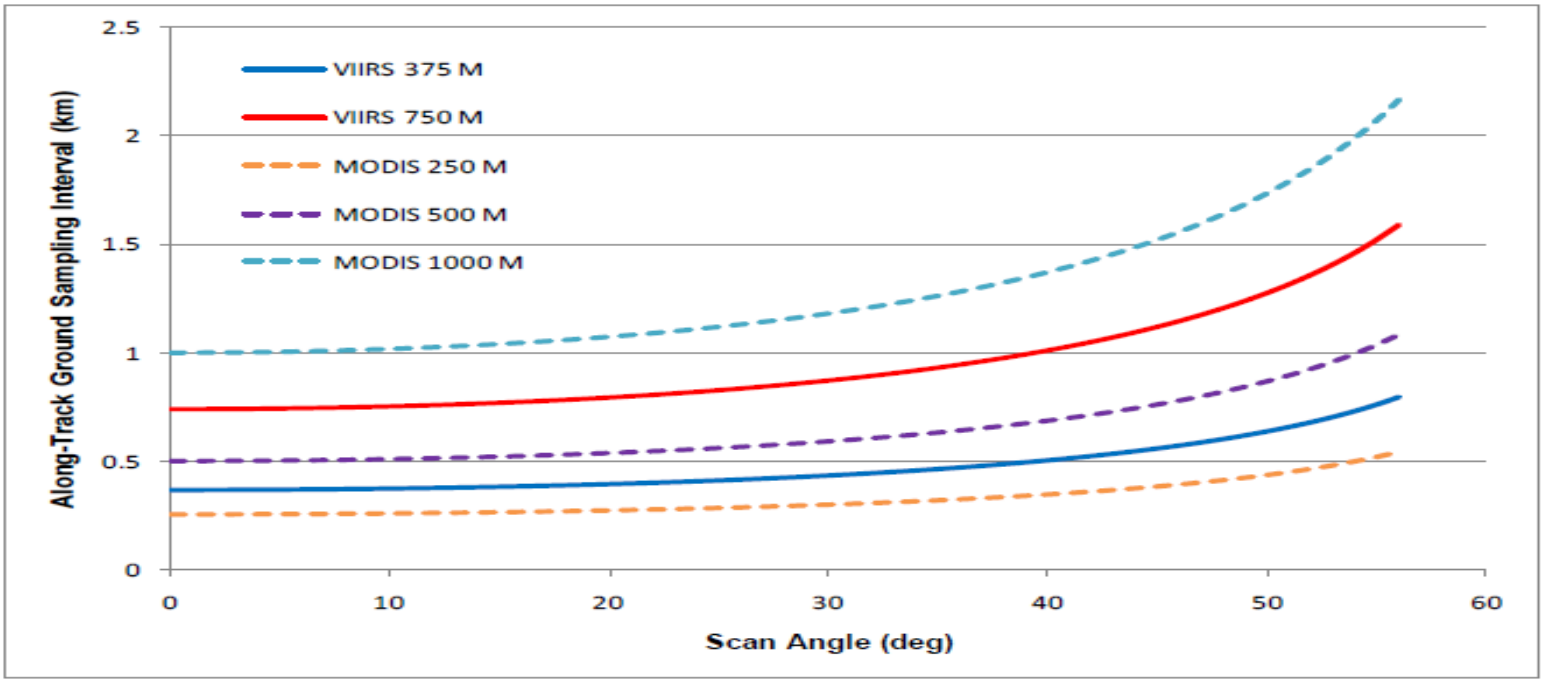
(1) use a systematic *rubber sheet* transformation process that gently warps an image (through the use of polynomial equations) based on the known positions of a set of widely dispersed control points.

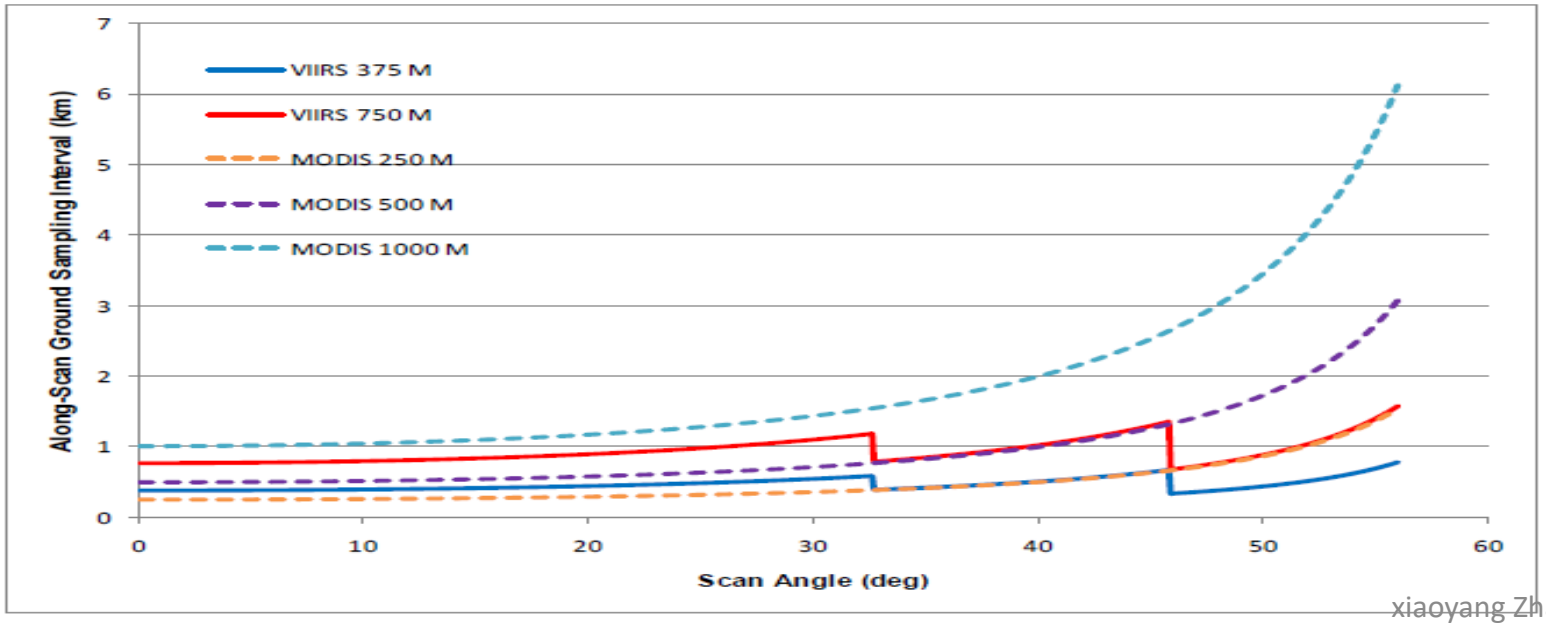
(2) use photogrammetric rectification to remove irregular distortions (such as variable topographic relief) and provide accurate map measurements.

(3) Use data resample to relocate the digital satellite observations.



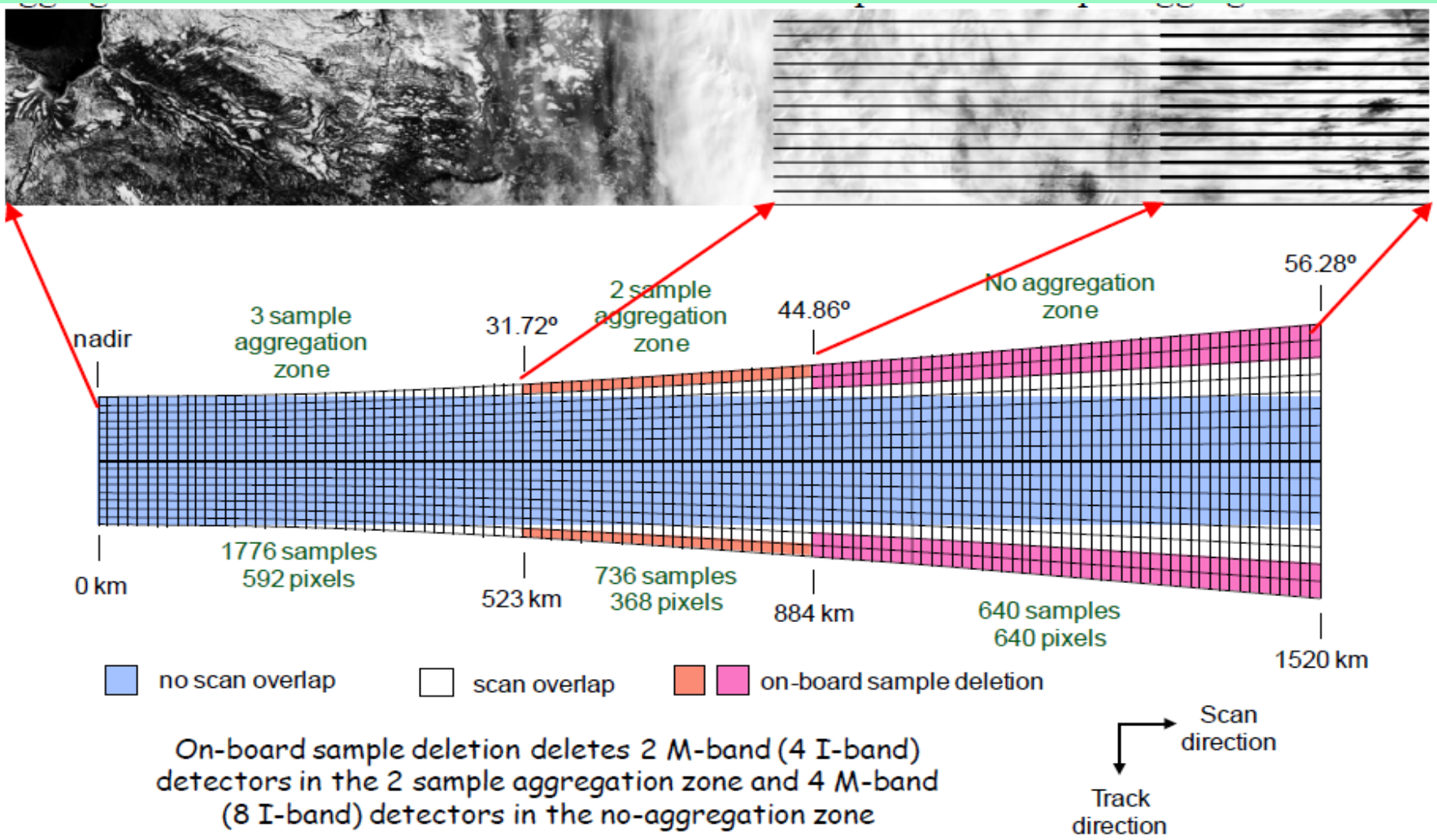
Spatial Resolution between MODIS and VIIRS





Bruce Guenther et al., 2011

VIRRS Pixel Size



Lower: schematics of bow-tie effect, bow-tie deletion and aggregation scheme for single-gain M-bands (scale is exaggerated in the track direction). Upper: example of bow-tie deletion effect when the raw data is displayed in sample space.

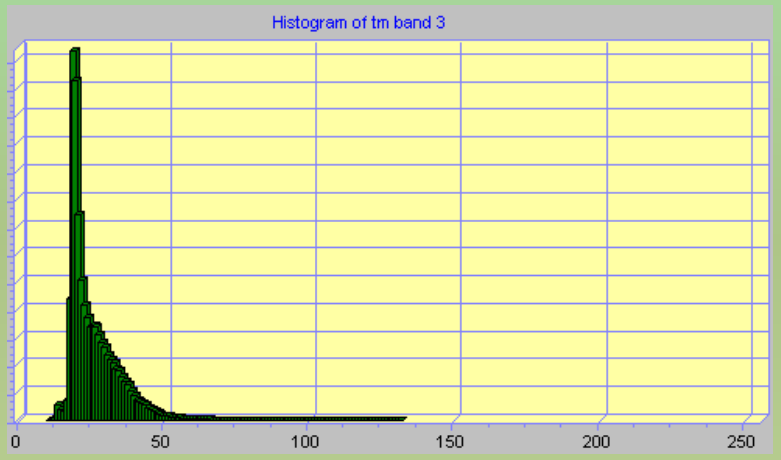
http://www.star.nesdis.noaa.gov/smcd/spb/nsun/snpp/VIIRS/VIIRS_SDR_Users_guide.pdf

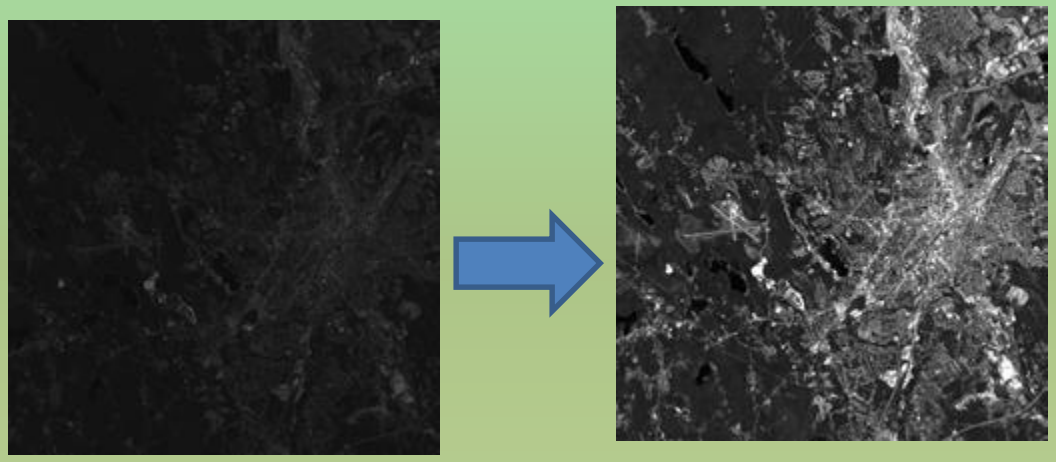
Image Enhancement

Modification of images to make them more suited to the capabilities of human vision.

Contrast Stretch

Composite Generation





Linear Stretch

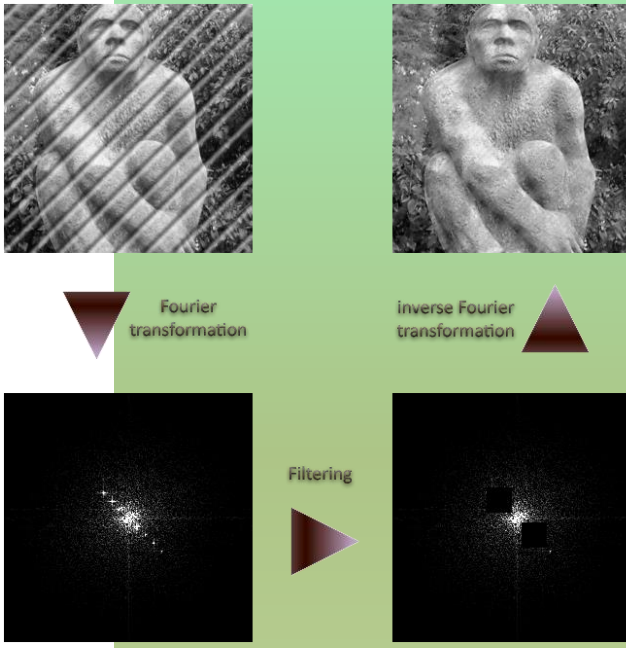
Spatial Digital Filtering

Low-Frequency Filtering in the Spatial Domain
High-Frequency Filtering in the Spatial Domain

- Convolution

z1	z2	z3
z4	z5	z6
z7	z8	z9

w1	w2	w3
w4	w5	w6
w7	w8	w9



Response of a linear mask, R,

$$R = (w1z1 + w2z2 + w3z3 \ldots + w9z9) = \sum_{1}^{9} wizi$$

f(x,y) is centered around "z5" so g(x,y) = R =filter_about*f(x,y)

Band Ratio –Vegetation index

Index	Function
Normalized difference vegetation index	$NDVI = (\rho NIR - \rho R)/(\rho NIR + \rho R)$
Simple ratio	$SR = \rho NIR/\rho R$
MIR index	$MIRI = (\rho MIR - \rho R)/(\rho MIR + \rho R)$
Difference vegetation index	$DVI = \rho NIR - \rho R$
Modified soil adjusted vegetation index 1	$MSAVI1 = (\rho NIR - \rho R)(1 + L)/(\rho NIR + \rho R + L))$; where L is a soil adjustment factor
Modified soil adjusted vegetation index 2	$MSAVI2 = \rho NIR + 0.5 - ((\rho NIR + 0.5)2 - 2(\rho NIR - \rho R))1/2$
Enhanced vegetation index	$EVI = G(\rho NIR - \rho R)/(\rho NIR + C1*\rho R - C2\rho B + L)$; where $L = 1$, $C1 = 6$, $C2 = 7.5$, $G = 2.5$
Chlorophyll based difference index	$CI = (\rho 850 - \rho 710)/(\rho 850 - \rho 680)$
MSI	$MSI = \rho SWIR/\rho NIR$
Tasseled cap transformation	Greenness (TCg), brightness (TCb) and wetness indices (TCw)
NDVIc index	$NDVIc = ((\rho NIR - \rho R)/(\rho NIR + \rho R))(1 - (\rho SWIR - \rho SWIRmin)/(\rho SWIRmax - \rho SWIRmin))$; where $\rho SWIRmax$ and $\rho SWIRmin$ are the minimum and mavximum reflectance observed in field plots

Image Classification

Supervised Classification

Training sites extraction:

identify examples of the information classes (i.e., land cover types) of interest in the image.

Signature analysis (Feature Selection):

develop a statistical characterization of the reflectances for each information class, such as analyses of the mean, variances and covariances over all bands.

Classification:

examine the reflectances for each pixel and making a decision about which of the signatures it resembles most based on the statistical characterization in training sites.

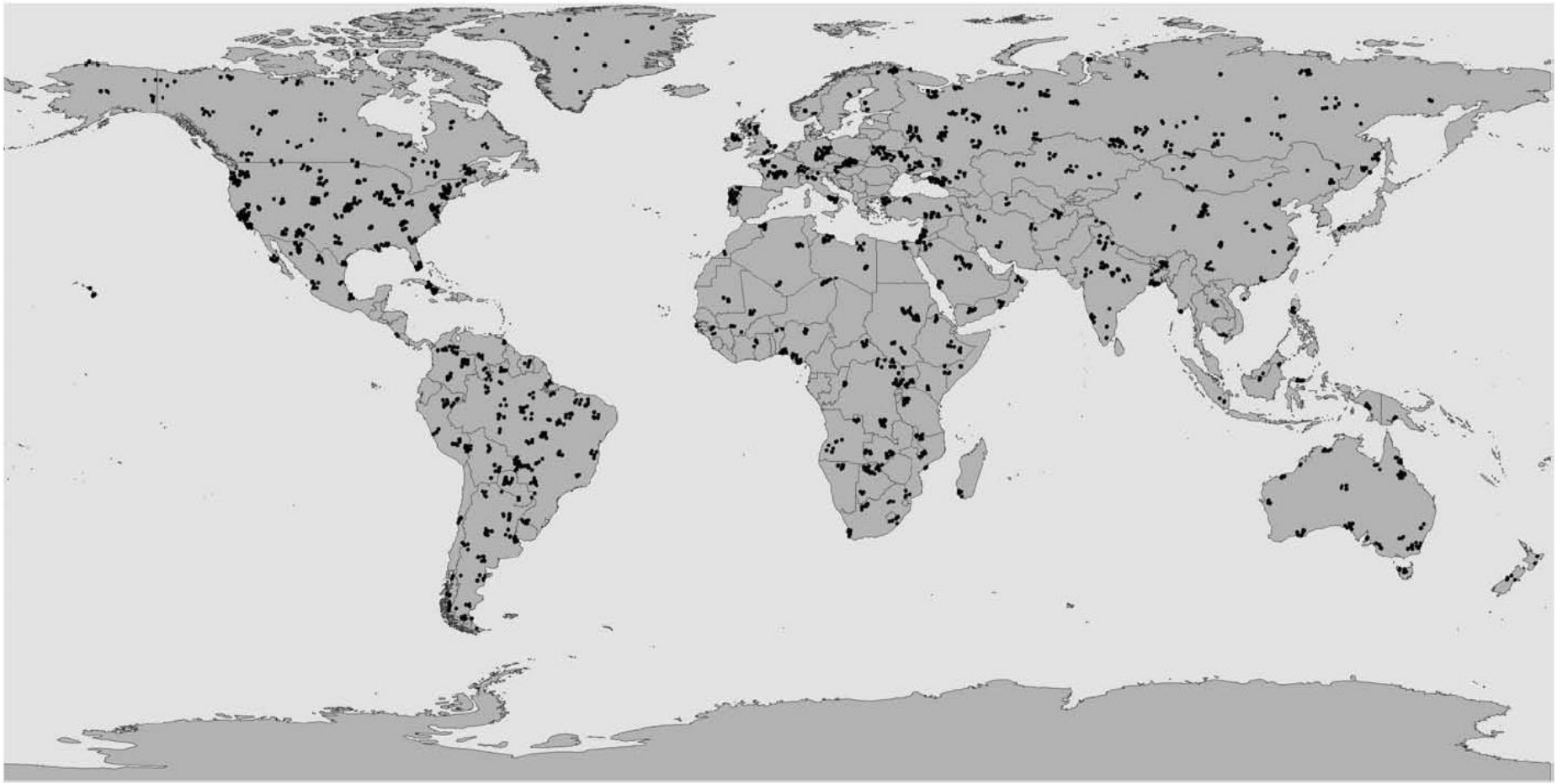


Fig. 2. Map of training sites (identified by dots) used to create the MODIS land cover type product.

287 land tiles, columns: 4800, Row: 4800 in each 500m tile.
Inputs: 12 Month x 7 band references

Image Classification

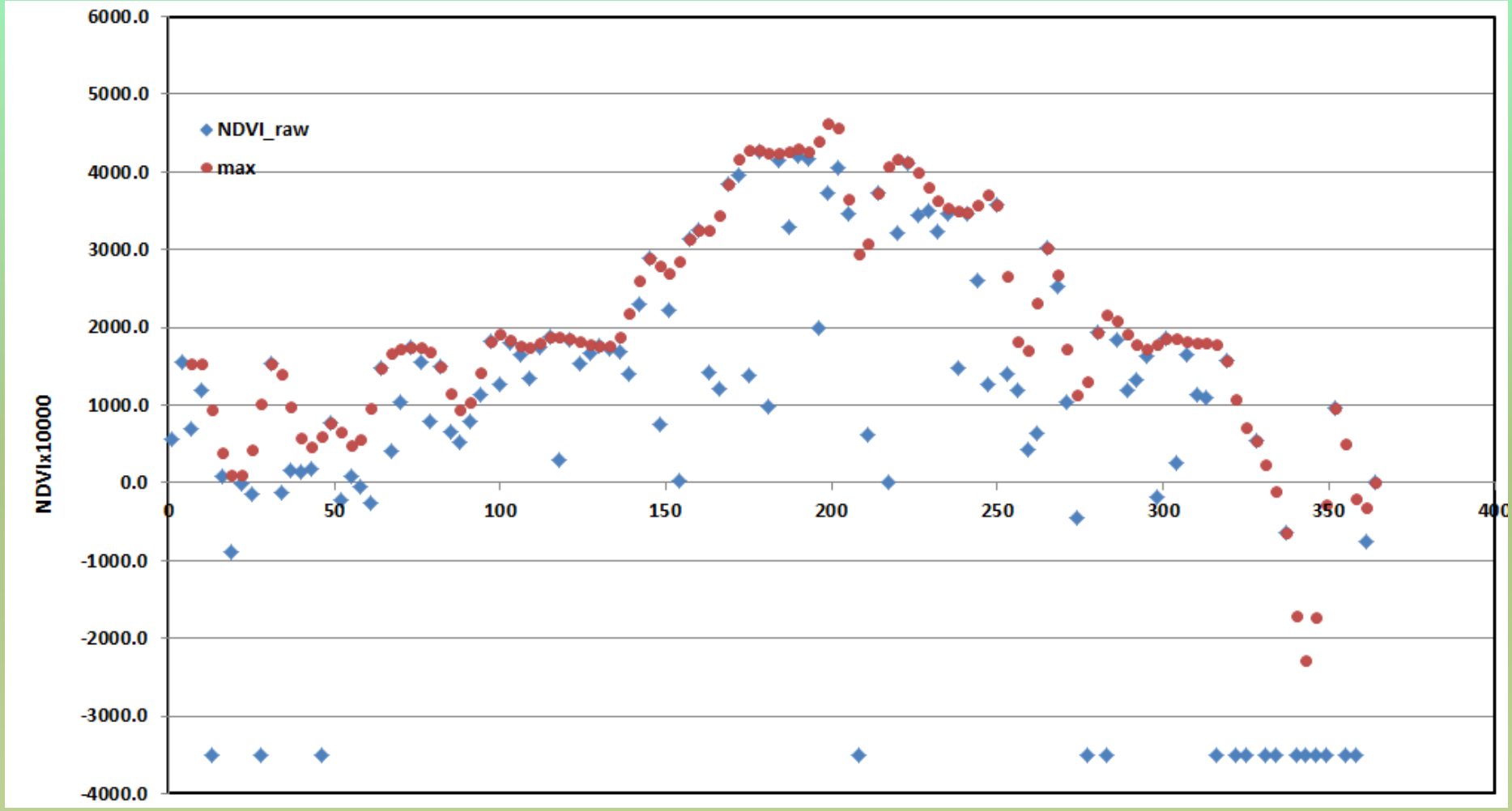
Unsupervised Classification

- unsupervised classification requires no advance information about the classes of interest.

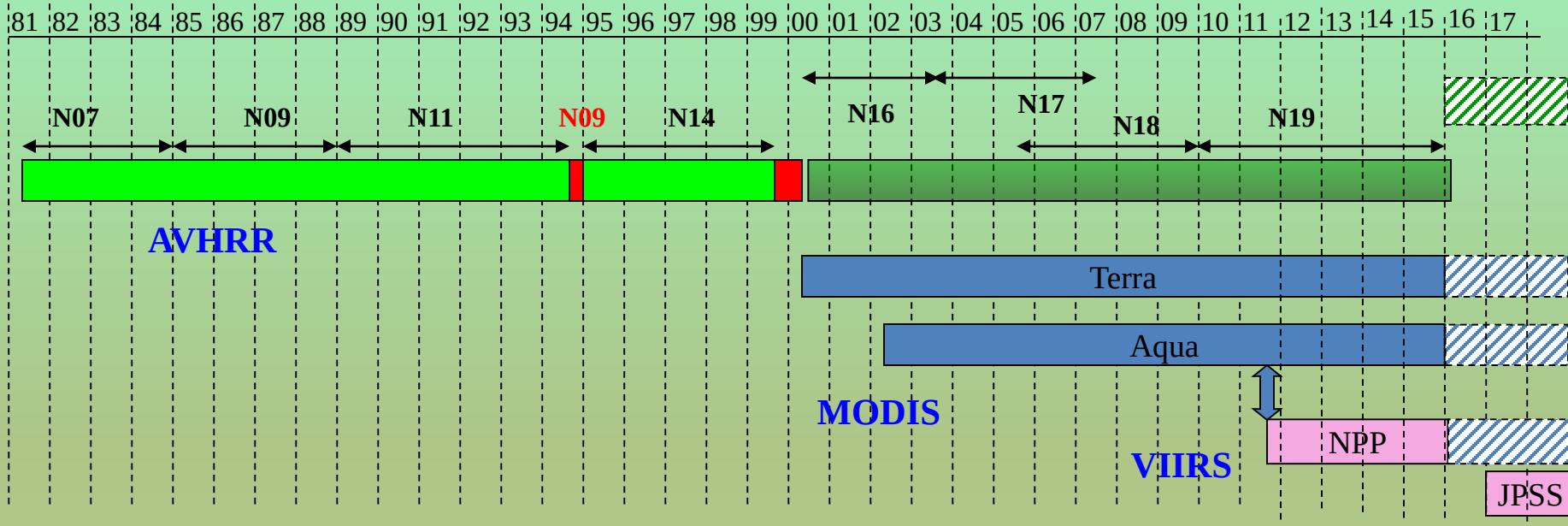
- Examine the data and breaks it into the most prevalent natural spectral groupings, or clusters, present in the data.

- Identify these clusters as land cover classes through a combination of familiarity with the region and ground truth visits.

Time Series Analysis:
Smoothing time series of vegetation index

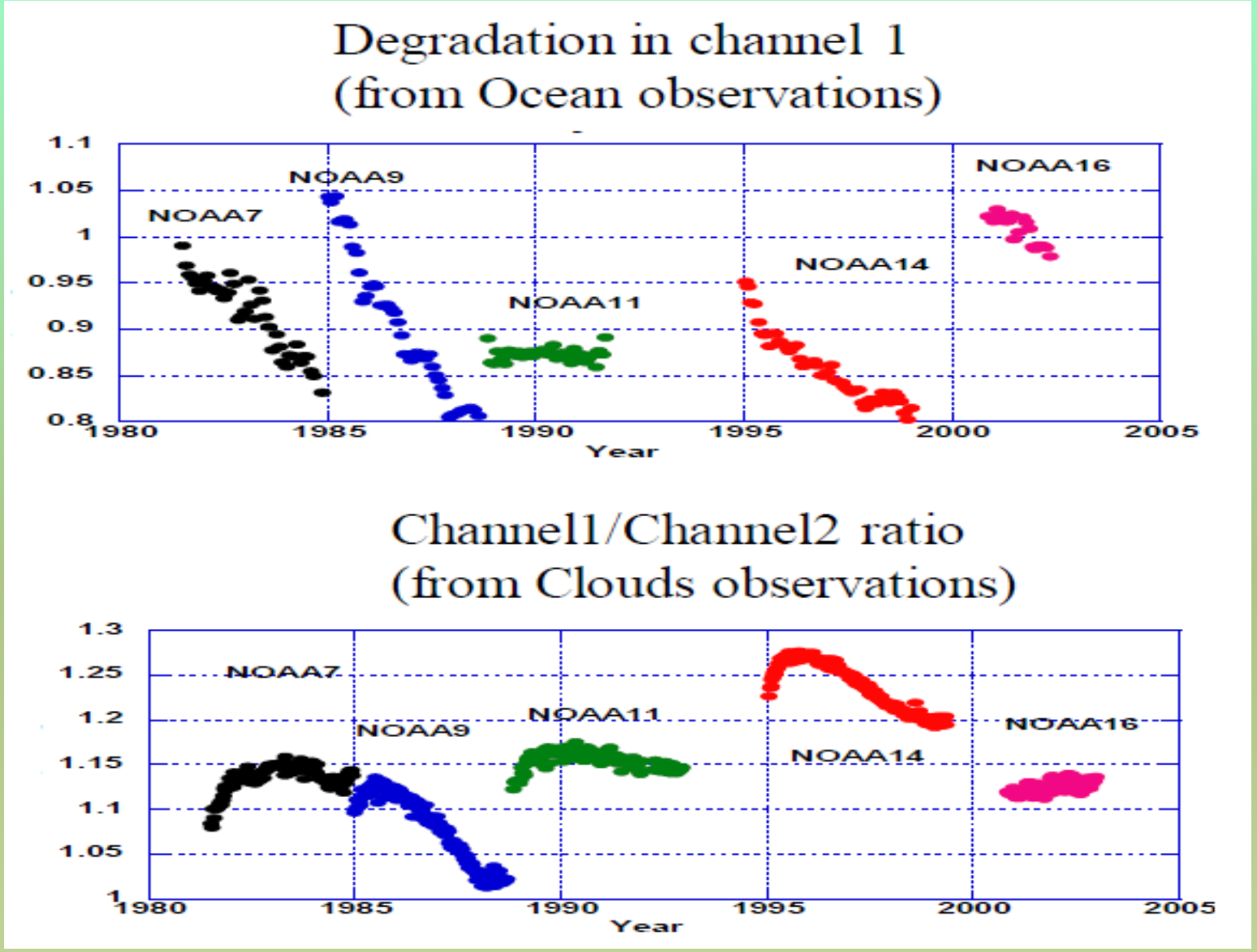


Long Term Data Record from Different Sensors



Modified from Eric Vermont

Calibration: Satellite data calibration



Calculation of Time Series NDVI from AVHRR Radiance

Reflectance of visible (VIS, channel1) and near infrared (NIR, channel 2) channels were calculated from the *radiance counts* by using the equation below:

Reflectance1= S1* (ch1_count -D1)

Reflectance2= S2* (ch2_count -D2)

Where S1, S2 are the calibration slope, they are estimated as a function of days since satellite launched.

D1, D2 are constant, dark count for VIS and NIR bands

Calculation of Time Series NDVI from AVHRR Radiance

For NOAA 16-19, Metop A and Metop B:

Step 1. pre-launch calibration:

Calculate pre-launch calibrated reflectance (R_prelaunch):

if (ch_count < threshold)

R_prelaunch= (low_gain_slope * *ch_count* + low_gain_intercept);

else

R_prelaunch=(high_gain_slope * *ch_count* + high_gain_intercept);

where, slop and intercept for low and high gain can be found in pre-launch calibration file (Appendix A)

see: https://www.star.nesdis.noaa.gov/smcd/emb/vci/VH/vh_avhrr_calibration.php

Calculation of Time Series NDVI from AVHRR Radiance

Pre-launch calibration Parameter - Appendix A

#sat, launch_date, see http://en.wikipedia.org/wiki/Advanced_Very_High_Resolution_Radiometer

NN=NOAA-18 (NOAA-NN): 05/20/2005

NP=NOAA-19 (NOAA-NP): 02/06/2009

pre-launched calibration, version

#Sat	CH	Slope_lo	Int_lo	Slope_hi	Int_hi	BreakPoint
NN	CH1:	0.0545	-2.179	0.1613	-55.63	500.54
NN	CH2:	0.0526	-2.084	0.1584	-55.07	500.4
NP	CH1:	0.0551	-2.1415	0.1625	-55.863	496.43
NP	CH2:	0.0549	-2.1288	0.1635	-56.445	500.37

Calculation of Time Series NDVI from AVHRR Radiance

Step 2. post-launch calibration:

Reflectance = R_prelaunch * Slope

where,

'Slope' is the post-launch calibration ratio:

Slope= Reference_Reflectance/(reflectance_constant + degradation_rate * daysSinceLaunch)

Reference_Reflectance for VIS and NIR bands are 37.80 and 42.60 respectively;

reflectance_constant and degradation_rate are read from monthly updated calibration records.

reflectance_constant= 41.0745

degradation_rate= -0.0599 /100

AVHRR NDVI Calculation

NDVI= (Reflectance2-Reflectance1)/(Reflectance2+Reflectance1)

AVHRR NN (**NOAA-18**) data

(1)Check the variables and metadata
(2)Extract variable
(3)Calculate daily NDVI
(4)Composite to 3-day NDVI using maximum approach

(1) Check the variables and metadata using "hdp" or "hdfview"

[zhangx@hunter]$ hdp dumpsds -h GVIX_NN_G04_C07_Y2013_P50_D344.hdf |more

Checking Metadata

Attr8: Name = GRID_ROWS
Type = 32-bit signed integer
Count= 1
Value = 3616
Attr9: Name = GRID_COLUMNS
Type = 32-bit signed integer
Count= 1
Value = 10000
Attr10: Name = START_LATITUDE_RANGE
Type = 32-bit floating point
Count= 1
Value = 75.024002
Attr11: Name = END_LATITUDE_RANGE
Type = 32-bit floating point
Count= 1
Value = -55.152000

Attr12: Name = START_LONGITUDE_RANGE
Type = 32-bit floating point
Count= 1
Value = -180.000000
Attr13: Name = END_LONGITUDE_RANGE
Type = 32-bit floating point
Count= 1
Value = 180.000000

Checking Metadata

Variable Name = ch1_count
Index = 8
Type= 16-bit signed integer
Ref. = 18
Compression method = DEFLATE
Deflate level = 6
Compression ratio (original:compressed) = 5.95:1
Rank = 2
Number of attributes = 8
Dim0: Name=fakeDim14
Size = 3616
Scale Type = number-type not set
Number of attributes = 0
Dim1: Name=fakeDim15
Size = 10000
Scale Type = number-type not set
Number of attributes = 0

Attr0: Name = UNITS
Type = 8-bit signed char
Count= 5
Value = count
Attr1: Name = MISSING
Type = 32-bit floating point
Count= 1
Value = -999.000000
Attr2: Name = SCALED
Type = 8-bit signed integer
Count= 1
Value = 0
Attr3: Name = units
Type = 8-bit signed char
Count= 5
Value = count

Processing in ASCII Text data

Extract ASCII data:

[zhangx@hunter]$ hdp dumpsds -n ch1_count -d -o Y2013_P50_D344_ch1.txt -x GVIX_NN_G04_C07_Y2013_P50_D344.hdf

Check the ASCII data:

[zhangx@hunter]$ wc Y2013_P50_D344_ch1.txt
2455372 36160000 172905171 Y2013_P50_D344_ch1.txt

Where total line = 2455372,
total value = 36160000, which is the same as the size (row=3616 and column=10000) in hdf file

Perl Script in Producing NDVI Time Series

Lect10_1.pl

```
#!/usr/bin/perl
$ENV{OUTD}="/disk1/pub/XYZ/JPSS/DATA4km/DATA/";
$dir4km="/data/data044/DATA/VIIRS/4km/daily/";
$yyyy=2013;
$startday=210;
$endday=240;
$fill_value=-999;
for($i=$startday;$i<$endday;$i++)
{
&READ_DATA;
}
print "end\n";
```

GVIX_NN_G04_C07_Y2013_P30_D224.hdf

```
sub READ_DATA{
   $dd=$i;
   if($i<10) {$dd=join('',"00",$i);}
   if(($i>9)&&($i<100)) {$dd=join('',"0",$i);}
  $input_file=join('',$dir4km,"GVIX_NN_G04_C07_Y",$yyyy, "_P30_D", $dd,".hdf");

 $out_ch1_file=join('',$ENV{OUTD},$yyyy,$dd, "ch1_", $yyyy,$dd,".txt");
 $out_ch2_file=join('',$ENV{OUTD},$yyyy,$dd, "ch2_", $yyyy,$dd,".txt");
$ndvi_file=join('',$ENV{OUTD},$yyyy,$dd, "NDVI_", $yyyy,$dd,".txt");
system("hdp dumpsds -n ch1_count -d -o $out_ch1_file -x $input_file");
system("hdp dumpsds -n ch2_count -d -o $out_ch2_file -x $input_file");
&Calculate_CH1;
&Calculate_CH2;
&calculate_NDVI;
}
```

Perl Script in Producing NDVI Time Series

```
sub Calculate_CH1{
}

sub Calculate_CH2{
}

Sub Calculate_NDVI{

}
```

Perl Script in Producing NDVI Time Series

Lect10_2.pl

```
#!/usr/bin/perl
$ENV{OUTD}="/disk1/pub/XYZ/JPSS/DATA4km/DATA/";
$dir4km="/data/data044/DATA/VIIRS/4km/daily/";
$yyyy=2013;
$startday=210;
$endday=240;
$fill_value=-999;
for($i=$startday;$i<$endday;$i++)
{
&READ_DATA;
}
print "end\n";
```

```
sub READ_DATA{
   $dd=$i;
   if($i<10) {$dd=join('',"00",$i);}
   if(($i>9)&&($i<100)) {$dd=join('',"0",$i);}
 $input_file=join('',$dir4km,"GVIX_NN_G04_C07_Y",$yyyy, "_P50_D", $dd,".hdf");
$out_ch1_file=join('',$ENV{OUTD},$yyyy,$dd, "ch1_", $yyyy,$dd,".txt");
$out_ch2_file=join('',$ENV{OUTD},$yyyy,$dd, "ch2_", $yyyy,$dd,".txt");
 $ndvi_file=join('',$ENV{OUTD},$yyyy,$dd, "NDVI_", $yyyy,$dd,".txt");
 system("hdp dumpsds -n ch1_count -d -o $out_ch1_file -b $input_file");
 system("hdp dumpsds -n ch2_count -d -o $out_ch2_file -b $input_file");
 #calculate reflectance in ch1
#calcualte relectance in Ch2
#calculate NDVI
System("./cal_AVHRR_NDVI.exe  $out_ch1_file $out_ch2_file   $ndvi_file")
### in C or FORTRAN codes
}
```

Perl Script in Producing NDVI Time Series

```
#!/usr/bin/perl
$ENV{OUTD}="/disk1/pub/XYZ/JPSS/DATA4km/DATA/";
$dir4km="/data/data044/DATA/VIIRS/4km/daily/";
$yyyy=2013;
$startday=210;
$endday=240;
$fill_value=-999;
for($i=$startday;$i<$endday;$i++)
{
&READ_DATA;
}
print "end\n";
```

```
sub READ_DATA{
   $dd=$i;
   if($i<10) {$dd=join('',"00",$i);}
   if(($i>9)&&($i<100)) {$dd=join('',"0",$i);}
 $input_file=join('',$dir4km,"GVIX_NN_G04_C07_Y",$yyyy, "_P50_D", $dd,".hdf");
$ndvi_file=join('',$ENV{OUTD},$yyyy,$dd, "NDVI_", $yyyy,$dd,".txt");

System("./cal_AVHRR_NDVI_full.exe  $input_file $ndvi_file")
### in C or FORTRAN codes
}
```

Perl Script in Producing NDVI Time Series

```
#!/usr/bin/perl
$ENV{OUTD}="/disk1/pub/XYZ/JPSS/DATA4km/DATA/";
$dir4km="/data/data044/DATA/VIIRS/4km/daily/";

open(INPUT, "<fileList.txt");    # Open new file to write
@alllines = <INPUT>;        # Import all from a file into an array
close(INPUT);

foreach input_file (@alllines){
&READ_DATA;
}
print "end\n";
```

```
sub READ_DATA{

$input_file=join('',$dir4km,"GVIX_NN_G04_C07_Y",$yyyy, "_P50_D", $dd,".hdf");
$ndvi_file=join('',$ENV{OUTD},$yyyy,$dd, "NDVI_", $yyyy,$dd,".txt");

System("./cal_AVHRR_NDVI_full.exe  $input_file $ndvi_file")
### in C or FORTRAN codes
}
```

Reading Binary Data from a File in Perl

- The read stream must be set to binary mode using the **'binmode'** operator.
- Reading is performed by successive calls to the 'read' function Specifying the maximum number of bytes to read
- The 'read' function returns the **number of bytes** read
- The end of file is detected when the 'read' function returns zero.

Writing Binary Data to a File

The output stream must be set to binary mode

The 'print' function is used to write data

Perl pack/unpack Function

Using the pack function to assign a binary literal to a variable

The built-in perl function pack returns a string of bytes from the decimal/**hexadecimal** representation received as argument.

unpack function unpacks the binary string **STRING** using the format specified in **TEMPLATE**. Basically reverses the operation of pack, returning the list of packed values according to the supplied format.

unpack TEMPLATE, STRING

pack/unpack TEMPLATE

Character	Description
a	ASCII character string padded with null characters
A	ASCII character string padded with spaces
b	String of bits, lowest first
B	String of bits, highest first
c	A signed character (range usually -128 to 127)
C	An unsigned character (usually 8 bits)
d	A double-precision floating-point number
f	A single-precision floating-point number
h	Hexadecimal string, lowest digit first
H	Hexadecimal string, highest digit first
i	A signed integer
I	An unsigned integer
l	A signed long integer
L	An unsigned long integer
n	A short integer in network order
N	A long integer in network order
p	A pointer to a string

(0-255)

pack/unpack TEMPLATE (2)

Character	Description	
s	A signed short integer	(-32,768 to 32,767)
S	An unsigned short integer	(0 to 65,535)
u	Convert to uuencode format	
v	A short integer in VAX (little-endian) order	
V	A long integer in VAX order	
x	A null byte	
X	Indicates "go back one byte"	
@	Fill with nulls (ASCII 0)	

pack/unpack TEMPLATE (3)

Character	Description
a	ASCII character string padded with null characters
A	ASCII character string padded with spaces
b	String of bits, lowest first
B	String of bits, highest first
c	A signed character (range usually -128 to 127)
C	An unsigned character (usually 8 bits) (0-255)
d	A double-precision floating-point number
f	A single-precision floating-point number
h	Hexadecimal string, lowest digit first
H	Hexadecimal string, highest digit first

Most Significant Bit (MSB) and least significant bit (LSB)

Binary (Decimal: 149)	1	0	0	1	0	1	0	**1**
Bit weight for given bit position n (2^n)	2^7	2^6	2^5	2^4	2^3	2^2	2^1	**2^0**
Bit position label	MSB	___	___	___	___	___	___	**LSB**

Using the pack/unpack function to convert between a binary literal and a variable

The built-in perl function **pack** returns **a string of bytes** from the **decimal/hexadecimal** representation received as argument.

```
my $data;
$data = pack(“s”, 655);
$data = pack(”C”, 41);
$data = pack(”H”,”414243”)
```

The inverse of pack is ‘**unpack**’ returns **a text string** with the **decimal/hexadecimal** representation of binary data received as argument:

```
print "hexadecimal representation"  unpack("H",$data) “\n";
print “unsigned character representation: "  unpack("C",$data) “\n";
print "decimal representation: "  unpack(“s",$data) “\n";
```

Using the pack/unpack function to convert between a binary literal and a variable

A format character followed by a * repeats the format character enough times to swallow up the rest of the list or the rest of the binary image string.

```
my $data;
$data = pack(“s*”, 655, 664, 673);
$data = pack(”C*”, 41, 42, 43);
$data = pack(”H*”,”414243”)
```

The inverse of pack is ‘**unpack**’ returns **a text string** with the **decimal/hexadecimal** representation of binary data received as argument:

```
print "hexadecimal representation"  unpack("H*",$data) "\n";
print " unsigned character representation: "  join(",",unpack("C*",$data)) "\n";
print "decimal representation: "  join(",",unpack(“s*",$data)) "\n";
```

Using the pack/unpack function to convert between a binary literal and a variable

Two short integers followed by "as many unsigned chars as possible,"
$buf = pack(″s2C*″, 3141, 5926, 5, 3, 5, 8, 9, 7, 9, 3, 2);

unpack **with an asterisk specification** can generate a list of elements of un-predetermined length.

@values = unpack(″C*″, ″hello, world!\n″);
yields a list of 14 elements, one for each of the characters of the string.

```
$da=pack(“s*”, 3141, 5926, 125, 223, 125, 28);
@data = unpack(″s*″, $da);
```

Example

Lect10_3.pl

```
#!/usr/bin/perl
# Open the file
my $filename = "data.dat";
open DATAIN, "<$filename"
   or die "Error opening file $filename: $!\n";
# set stream to binary mode
binmode DATAIN;

my $data;        # read buffer
my $nbytes;
while ($nbytes = read DATAIN, $data, 2) {
   # Process data read
   $oneV=unpack(“s", $data);
## $oneV=join(" ", unpack(“s*", $data));
   print "$nbytes bytes read, data==$oneV\n";
}

close DATAIN
   or die "Error closing $filename: $!\n";
```

Example

Lect10_4.pl

```
#!/usr/bin/perl

# Open the file
my $filename 1= "data.dat1";
open DATAIN1, "<$filename1"
or die "Error opening file $filename: $!\n";
# set stream to binary mode
binmode DATAIN1;
my $filename2= "data.dat2";
open DATAIN2, "<$filename2"
 or die "Error opening file $filename: $!\n";
# set stream to binary mode
binmode DATAIN2;
```

```
my $data1;       # read buffer
my $nbytes1;
my $data2;       # read buffer
my $nbytes2;

while ($nbytes1=read DATAIN1, $data1, 2) {
 $oneV1=unpack(“s", $data1);
$nbytes2=read (DATAIN2, $data2, 2);
 $oneV2=unpack(“s", $data2);
$x=($oneV2-$oneV1)/($oneV2+$oneV1);
   print "$oneV1, $oneV2, $x\n";
}
close DATAIN1
   or die "Error closing $filename: $!\n";
close DATAIN2
   or die "Error closing $filename: $!\n";
```

Example

Lect10_5.pl

```
#!/usr/bin/perl

# Open the input file
my $filename = "data.dat";
open DATAIN, "<$filename"
  or die "Error opening file $filename: $!\n";
# set stream to binary mode
binmode DATAIN;

$filename = "data.out";
open DATAOUT,">$filename"
 or die "Error opening file $filename: $!\n";
# set the stream to binary mode
binmode DATAOUT;

my $data; # read buffer
my $nbytes=2;
my $xout;
```

```
# Read data in chunks of 2 bytes
while (read DATAIN, $data, $nbytes) {
   # Process data read
$oneV=join(" ", unpack(“s*", $data));

##print DATAOUT pack(‘s*',@arr_datA);
## $outd=pack('S*',@arr_datA);

$outd=$outd+10;
$outd=pack(‘s*',$oneV);
print DATAOUT $outd;
}

close DATAIN
   or die "Error closing $filename: $!\n";
close DATAOUT
or die "Error closing file $fielname: $!\n";
```

Example

Lect10_6.pl

```
#!/usr/bin/perl

$filename = "data.out";
open DATAOUT,">$filename"
 or die "Error opening file $filename: $!\n";
# set the stream to binary mode
binmode DATAOUT;
# Open the file
my $filename = "data.dat";
##open DATAIN, $filenama
open DATAIN, "<$filename"
   or die "Error opening file $filename: $!\n";
# set stream to binary mode
binmode DATAIN;

my $data; # read buffer
my $nbytes=2*10;
```

```
while (read DATAIN, $data, $nbytes) {
   # Process data read
@inD=unpack("s*", $data);

foreach $x(@inD){
print "$x\n";
}
$outd=pack('s*',@inD);
print DATAOUT $outd;

# print "$nbytes bytes read\n";
}

close DATAIN
   or die "Error closing $filename: $!\n";
close DATAOUT
or die "Error closing file $fielname: $!\n";
```

Example

Lect10_7.pl

```
#!/usr/bin/perl

$filename = "data.out";
open DATAOUT,">$filename"
 or die "Error opening file $filename: $!\n";
# set the stream to binary mode
binmode DATAOUT;
# Open the file
my $filename = "data.dat";
##open DATAIN, $filenama
open DATAIN, "<$filename"
   or die "Error opening file $filename: $!\n";
# set stream to binary mode
binmode DATAIN;
my @Newdata=NULL;
my $data1; # read buffer
my $data2;
my $row=3616;
My $col=10000;
my $nbytes=2*$col;
```

```
for($i=0;$i<$row;$i++){
read(DATAIN1, $data1, $nbytes);
@inD=unpack("s*", $data);
$i=0;
foreach $x(@inD){
Chomp($x);
$Newdata[$i]=$x*$x;
}
$outd=pack('s*',@Newdata);
print DATAOUT $outd;

# print "$nbytes bytes read\n";
}

close DATAIN
   or die "Error closing $filename: $!\n";
close DATAOUT
or die "Error closing file $fielname: $!\n";
```

Perl Script in Producing NDVI Time Series

Lect10_8.pl

```
#!/usr/bin/perl

$ENV{OUTD}="/disk1/pub/XYZ/JPSS/DATA4km/DATA/";
$dir4km="/data/data044/DATA/VIIRS/4km/daily/";
$yyyy=2013;
$startday=210;
$endday=240;
$fill_value=-999;
for($i=$startday;$i<$endday;$i++)
{
&READ_DATA;
}
print "end\n";
```

```
sub READ_DATA{
   $dd=$i;
   if($i<10) {$dd=join("","00",$i);}
   if(($i>9)&&($i<100)) {$dd=join("","0",$i);}
$input_file=join('',$dir4km, "VGVI_21Bands.G04.C01.npp.P",$yyyy,$dd,".nc");
$out_ch1_file=join('',$ENV{OUTD},$yyyy,$dd, "ch1_", $yyyy,$dd,".txt");
$out_ch2_file=join('',$ENV{OUTD},$yyyy,$dd, "ch2_", $yyyy,$dd,".txt");
 $ndvi_file=join('',$ENV{OUTD},$yyyy,$dd, "NDVI_", $yyyy,$dd,".txt");
 system("hdp dumpsds -n ch1_count -d -o $out_ch1_file -x $input_file");
 system("hdp dumpsds -n ch2_count -d -o $out_ch2_file -x $input_file");
&calculate_NDVI;
}
```

Perl Script in Producing NDVI Time Series

```
Sub calculate_NDVI{
open DATAIN1,“< $out_ch1_file"
 or die "Error opening file $filename: $!\n";
binmode DATAIN1;
open DATAIN2,“< $out_ch2_file"
 or die "Error opening file $filename: $!\n";
binmode DATAIN2;

open DATAOUT, “>$ndvi_file"
   or die "Error opening file $filename: $!\n"
# set stream to binary mode
binmode DATAOUT;

my $data; # read buffer
my $row=3616;
My $col=10000;
```

```
my $data1;       # read buffer
my $data2;       # read buffer

while ($nbytes1=read DATAIN1, $data1, 2) {
 $oneV1=unpack(“s", $data1);
$nbytes2=read (DATAIN2, $data2, 2);
 $oneV2=unpack(“s", $data2);
If(($oneV1==$fill_value)||($oneV2==$fill_value)){
$NDVI=$fill_value;
}else{
$NDVI=($oneV2-$oneV1)/($oneV2+$oneV1);
}
  $outd=pack(‘s',$NDVI);
print DATAOUT $outd;
}
close DATAIN1
   or die "Error closing $filename: $!\n";
close DATAIN2
   or die "Error closing $filename: $!\n";
close DATAOUT
   or die "Error closing $filename: $!\n";
}
```

```
#!/usr/bin/perl
# get data VNP43IA4 from http

$uname='***';
$passwd='***';
##$ENV{work}='/hunter/data1/';
$ENV{work}='/hunter/data/XYZ/';
chdir("$ENV{work}");
###https://disc2.gesdisc.eosdis.nasa.gov/data/TRMM_L3/TRMM_3B42_Daily.7/1999/03/3B42_Daily.19990302.7.nc4
  $host="https://disc2.gesdisc.eosdis.nasa.gov";
  $dirc="data/TRMM_L3/TRMM_3B42_Daily.7";

for($yy=2000;$yy<2020;$yy++){
    if($yy%4 == 0){
    $days = 366;
    }else{
    $days = 365;
    }
}
for($yy=2000;$yy<2020;$yy++){
   for($mm=1;$mm<13;$mm++){
                    for($dd=1;$dd<32;$dd++){
                       $mm1=$mm;
                       $dd1=$dd;
                       if($mm<10){$mm1=join('', "0",$mm);}
                       if($dd<10){$dd1=join('', "0",$dd);}
                       $ymd=join('',$yy,$mm1,$dd1);
                       $fi=join('.',"3B42_Daily",$ymd,"7.nc4");
                       $url=join('/', $host, $dirc,$yy,$mm1,$fi);
##                     print "xx=$url\n";
  system("wget --user $uname --password $passwd $url");
                    }
   }
}
## System("wget --user user --password pass $URL");
```

xiaoyang Zhang, 3/26/2021

```
#!/usr/bin/perl


$ENV{work}='/hunter/***/';
chdir("$ENV{work}");
$outdirec="/hunter/**/";

 $host="*****";
 $dirc="***";
$uname="****";
$passwd="******";

$yy=2012;
for($yy=2016;$yy<=2018;$yy++){
if($yy%4 == 0){
 @md=(31,29,31,30,31,30,31,31,30,31,30,31);
   }else{
 @md=(31,28,31,30,31,30,31,31,30,31,30,31);
   }
```

```
$m=1;
$doy=1;  ##start DOY
foreach $dda(@md){
   chomp($dda);
                    $mm=$m;
                    if($m<10) {
                       $mm="0";
                       $mm.=$m;
                    }
   for($d=1;$d<=$dda;$d++){
                    $dd=$d;
                    if($d<10) {
                       $dd="0";
                       $dd.=$d;
                    }
                    print "ddd==$dd\n";

$locadir=join('.',$yy,$mm,$dd);

                    $directory=join('',$dirc,$locadir,"/");

                    $fileindex="index.html";
                    if(-e $fileindex){
                       unlink  $fileindex;
                    }

                    $URL=join('', "https://", $host,$directory);

system("wget $URL");  ##generate file of index.html
                    print "ttx\n";
                    if(-e $fileindex){

                    open(IN,"<$fileindex")|| die "cannot open file";
                    @alld=<IN>;
                    close IN;
```

```
$ind=0;
                        foreach $line(@alld){
                           chomp($line);
##                         print "x==$line\n";
##x==<img src="/icons/unknown.gif" alt="[   ]"> <a href="MCD43A4.A2012008.h35v09.006.2016092165411.hdf">MCD43A4.A2012008.h35v09.006.2016092165411.hdf</a>

                           $tile="h12v04";

if($line=~m/hdf">MCD43A4.*$tile/){
$fi1=(split('>',$line))[2];
print "file0==$fi1\n";
$fi1=(split('<',$fi1))[0];
print "file==$fi1\n";
$URL.=$fi1;

$ind2=system("wget -q -nc --user=$uname --password=$passwd $URL");        ##get the hdf file
if($ind2==0) {$ind=1;}
    else {$ind=2;}
                           }
                        }

system("rm index.html*");

                        }  ###file exists

                        $doy++;
   }
   $m++;
}
}
```

If hdfview, ncdump, hdf does not work, then

Add

```
export PATH=/opt/hdfview/bin:$PATH
export PATH=/opt/Toolkit-x86_64/hdf/bin:$PATH
```

To
~/.bashrc

```
~/source  .bashrc
```

END