Basic data reduction

Observational Astronomy 2025

Lab 1

Prof. S.C. Trager

university of groningen
faculty of science and engineering
kapteyn astronomical institute

Why “reduce” our data?

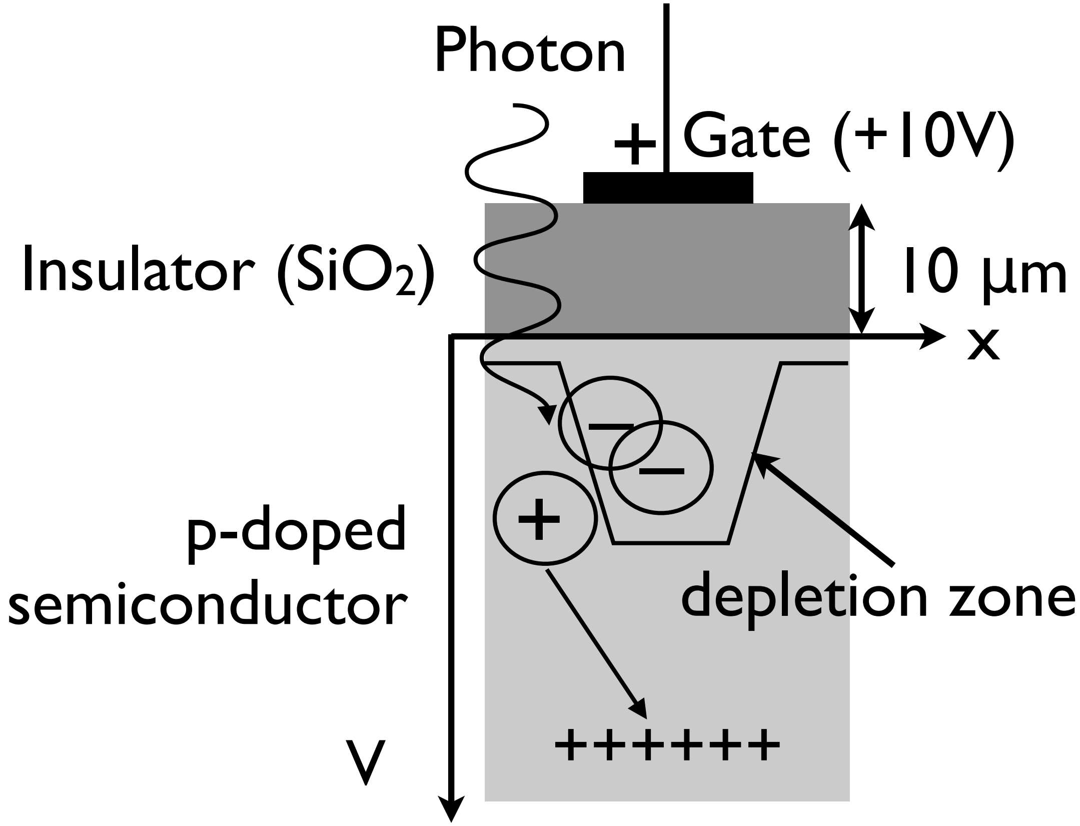
- Overall, we want to “reduce” huge numbers of data points (e.g., millions of pixels!) to a reasonable number that can be analyzed
- But first, we want to remove the **instrumental signatures** from our data
  - These “get in the way” of our science goals!
  - They are (usually) easily correctable, or at least *identifiable*
    - So let’s identify them and (if possible) correct them!

CCD data reduction

- An example that we'll need for this class: CCD imaging data reduction
  - What do we need to do?
    - (1) remove the *bias* ("zero") level imposed at the ADC
    - (2) remove the *dark current* created by the CCD itself
    - (3) remove the *pixel-to-pixel gain variations* of the CCD ("flat field")
    - (4) remove the *illumination pattern* of the camera ("illumination")
    - (5) identify and correct *bad pixels* (esp. *dead* pixels)

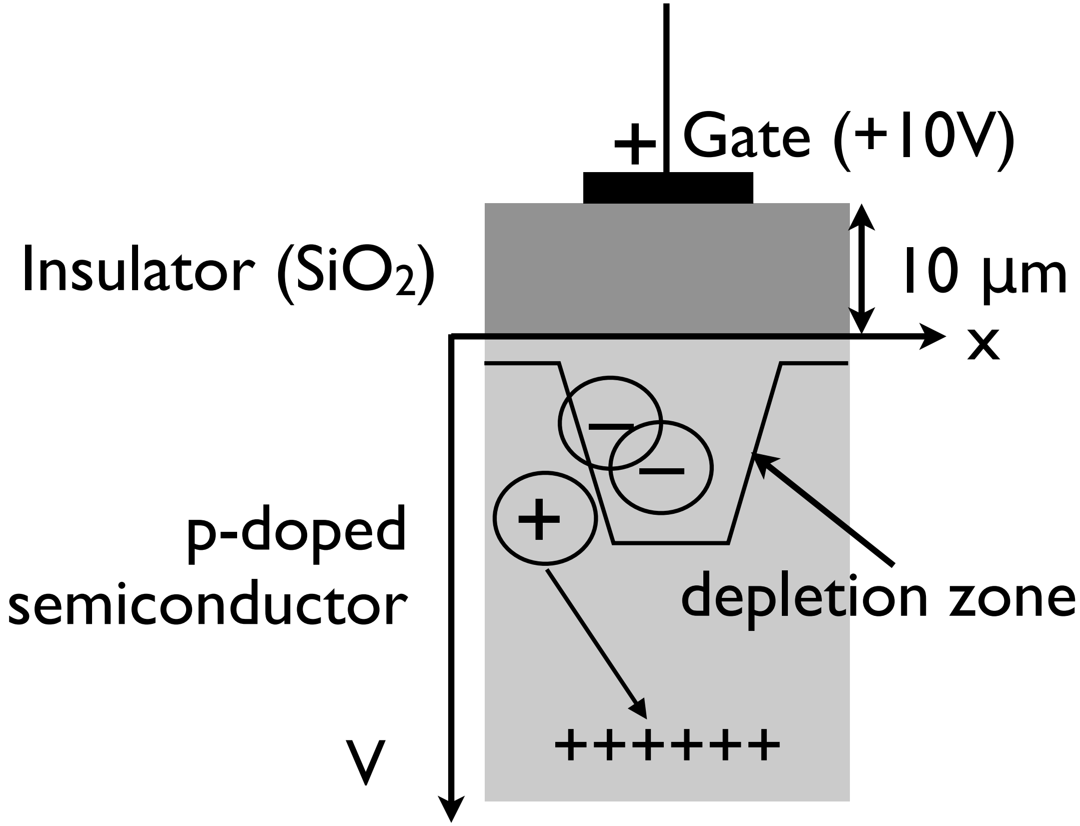
CCD pixel

- A CCD pixel is composed of a **metal-oxide semiconductor** (***MOS***) that “converts” photons to electrons that can be **stored** by applying a positive voltage to an aluminum ***gate***
    - An incoming photon hits a silicon atom in the semiconductor lattice and ejects an electron which is (hopefully!) trapped in the ***depletion zone*** and a ***hole*** that “migrates” away from the gate
    - The chance that this actually happens is called the ***quantum efficiency*** of the pixel



CCD pixel

- *Quantum mechanical tunneling* of an electron means that <u>extra</u> electrons might be collected in the depletion zone, especially at higher temperatures
    - This gives rise to an extra signal called ***<u>dark current</u>***
        - Dark current is not usually a problem for professional observatories with LN2-cooled detectors, but it's a significant issue for warm CCDs like the one on the Gratama Telescope



CCD readout

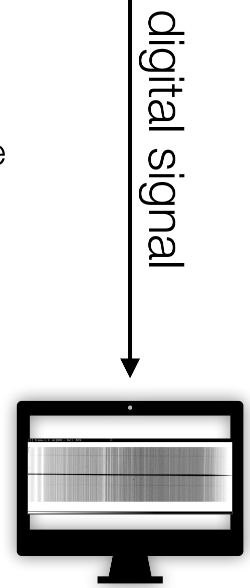
We readout a CCD with the following algorithm:

1) shift all rows by one pixel into the ***MUX***

2) read out **all** MUX pixels in order by shifting charges along the MUX to the amplifier — a ***field effect transistor*** (FET)

3) After the FET, the amplified signal goes through the ***analogue-to-digital converter*** (ADC)

4) when the MUX is completely empty, repeat from 1)



Bias, read noise, and flat fields

- Three major effects from the readout need to be understood to reduce and analyze the data:
  - ***Read noise***: the FET amplifier adds noise to the signal while amplifying it (this is always true with amplifiers – just ask any musician!). This noise is called **read noise** and cannot be removed but can be *accounted for* in the reduction
  - The factor that the FET amplifies the signal by is called the (inverse) ***gain*** and is usually given in electrons per "ADU" (analogue-to-digital unit) or "counts", which is the number of photons per count
    - note that you should **always compute your statistics in photons** (we'll see why in a few lectures)

Bias, read noise, and flat fields

- ***Bias*** level: the ADC converts the analogue signal output from the FET amplifier into a digital signal that can be ingested by a computer.
  - The ADC has a fixed number of ***bits*** that can be used to convert analogue signals to digital signals: typical CCD controllers have **16 bits = 2^{16} = 65536** possible values
  - However, because of the readout noise, the analogue signal can be *negative* for pixels with low or zero signal. How to deal with this?
  - Two possibilities:
    - Use one bit as a +/- sign – this reduces the possible dynamic range (for positive counts) by a factor of 2 (i.e., 32768 maximum counts)
    - *Add a small **bias level** to the counts* – usually around 1000–2000 counts, therefore using only a small amount of the dynamic range
      - This added bias level is what is done for **every CCD controller** I've ever used

Bias, read noise, and flat fields

- ***<u>Flat fields</u>***: the quantum efficiency (QE) of each pixel in a CCD is slightly (or sometimes strongly) different, so the same amount of light hitting different parts of the CCD give different numbers of counts in different pixels
  - To calibrate this QE variation, we take pictures of a uniform luminosity source
    - We can use this flat field to determine the *relative response* of each pixel: this normalized framed is called a (normalized) **flat field**

Bias, read noise, and flat fields

- The twilight sky makes a very good flat field
  - although watch out for clouds, which can cause irregularities in the flat field, and stars at the end of twilight!
- Sometimes “internal” (from inside the instrument) or “dome” (from light reflected off the inside of the dome) flat fields are used, but these are rarely <u>illuminated</u> in the same way at the real sky
  - In this case, you’ll also need to use ***<u>illumination frames</u>*** to correct for the residual pattern: these are constructed from (dome or internal) “flat fielded” twilight frames
    - heavily used in spectroscopic observations, where obtaining good flat fields from twilights can be challenging

One last thing…

- Sometimes there's a **dead pixel** (one with, e.g., a broken or flawed gate) in the chip
    - this can cause a ***bad column***, as any pixels behind that pixel cannot be readout as the readout proceeds
- What do with bad columns?
    - *Either* make sure you took a few image of the same source, shifting the telescope each time (best!) *or* interpolate over the bad column(s) (not best, but sometimes the best you can do)

A data reduction cookbook

1. Subtract the <u>bias level</u>
    a. First, combine an <u>odd</u> number of bias frames together using a median combine: if you have N bias frames (where N is odd), then at each pixel, rank the values from lowest to highest and take the value at the middle rank (note: if you have an <u>even</u> number of bias frames, you can take the average of the middle two values – this is what NumPy's "median" does)
    b. Call this new frame "<u>master bias</u>", *B* (and look at it before you do anything else!)
    c. Subtract the master bias from all of your other (non-bias) images *I*: $I_B = I - B$
    d. Optional but handy: you may want to <u>trim</u> your images to remove unnecessary "overscan" regions that are a representation of the bias – the "TRIMSEC" keyword in the FITS image header specifies the "good" region (in count-1 numbering, *à la* FORTRAN, not count-0, *à la* Python)

A data reduction cookbook

2. Subtract the dark current

   a. First combine the dark frames together to make a "master dark" frame D (make sure that they all have the same exposure time, otherwise you have to scale, which can be annoying)

      - a median combine like you used for bias correction is ok, if you have an odd number of dark frames (but can be biased by extreme pixels like cosmic ray hits)
      - otherwise a "sigma clipping" (which ignores values more than a set number of standard deviations away from the mean) or "minmax clipping" (which ignores the lowest and highest values) algorithm can be used

   b. Then *normalize* (i.e., divide) the master dark by the exposure time so that your master dark is in counts per second: $D_n = D/t_{\mathrm{exp}}$

   c. Now subtract this normalized master dark scaled to the exposure time from all your other (non-bias, non-dark) images: $I_{BD} = I_B - D_n \times t_{\mathrm{exp}}(I_B)$

A data reduction cookbook

3. Flat field the images

   a. For each band, scale and combine the flat field images in that band

      - Your goal is have a normalized master flat field for each band: one where the average value is 1
      - To do this, measure the median counts in each individual flat field and scale the counts in that frame so that the average (ok, median) value is 1 before combining
        - for example, if the median value of the flat field image F_i is $\langle F_i \rangle$, then scale the image by $1/\langle F_i \rangle$: $F_{i,n}=F_i/\langle F_i \rangle$
      - Then combine the normalized individual flat field images in a single band together with, say, a median combine (as explained above) to make a normalized master flat field for that band: $F_N(X)$ for band X

   b. Now flat field your (non-bias, non-dark, non-flat field) images by the appropriate normalized master flat field for that band — e.g., for imaging data in the V-band, divide all your V-band images by the V-band normalized flat field: $I_{BDF}(V)=I_{BD}(V)/F_N(V)$

A data reduction cookbook

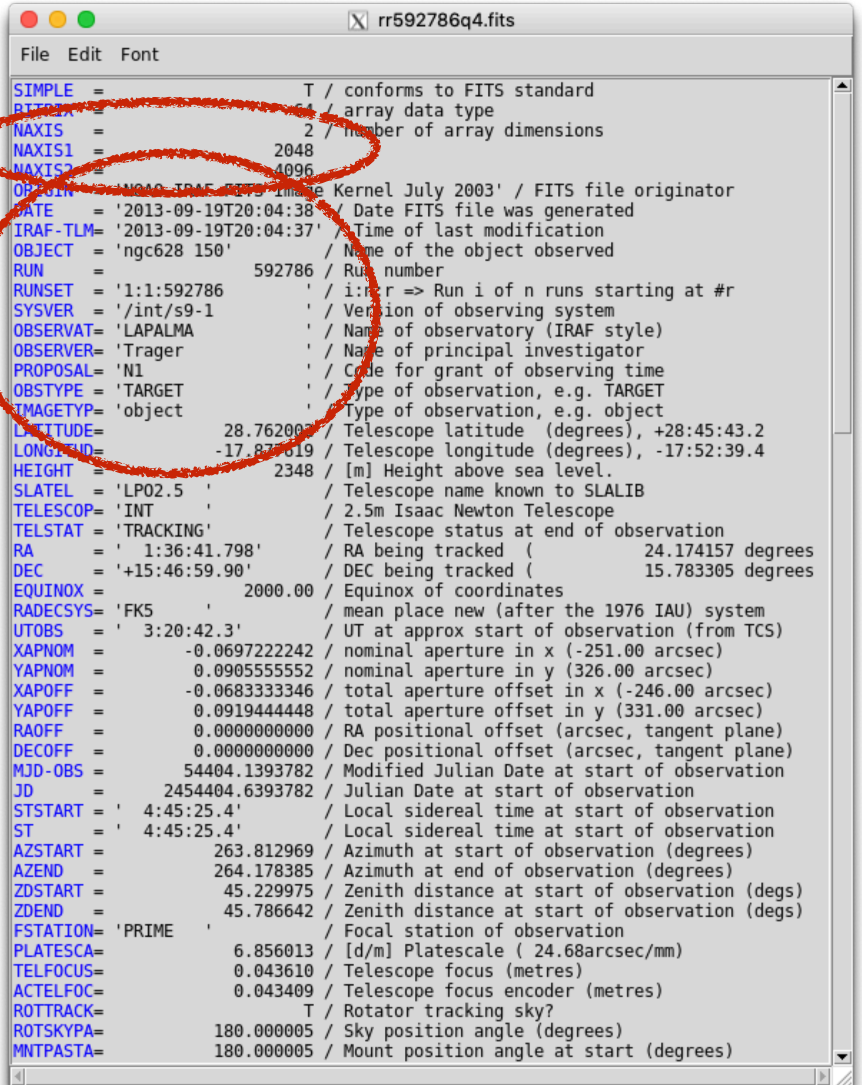
4. If <u>illumination correction</u> is required (because, e.g., the flat field images weren't taken against the twilight sky), then follow the same steps as for creating a normalized master flat field for each band, but use (bias-corrected, dark-corrected, flat-fielded) twilight images in place of the flat field images, and apply them in the same way to the (non-bias, non-dark, non-flat field) images in each band.

FITS files: how most astronomical data are stored

- FITS (“Flexible Image Transport System”) files are the cornerstone of astronomical data storage
- Nearly *all* optical CCD data are stored using FITS
- The <u>simplest</u> FITS file is composed of two parts:
  - A **header** that contains critical (and not-so-critical) information about the data in ASCII **keywords**
  - **Image data** (in binary format)
- Sometimes FITS files have multiple sets of these header+data pairs, each called an HDU (header-data unit)
  - the data doesn’t necessarily need to be an image – it can also be a **table**

FITS files: how most astronomical data are stored

- A FITS header contains keywords to help understand the content of the data
  - For example, when the data was taken, what kind of target it was, etc.
  - Critical information includes how many dimensions the data array has (NAXIS) and how big these dimensions are (NAXIS1, NAXIS2, ...)

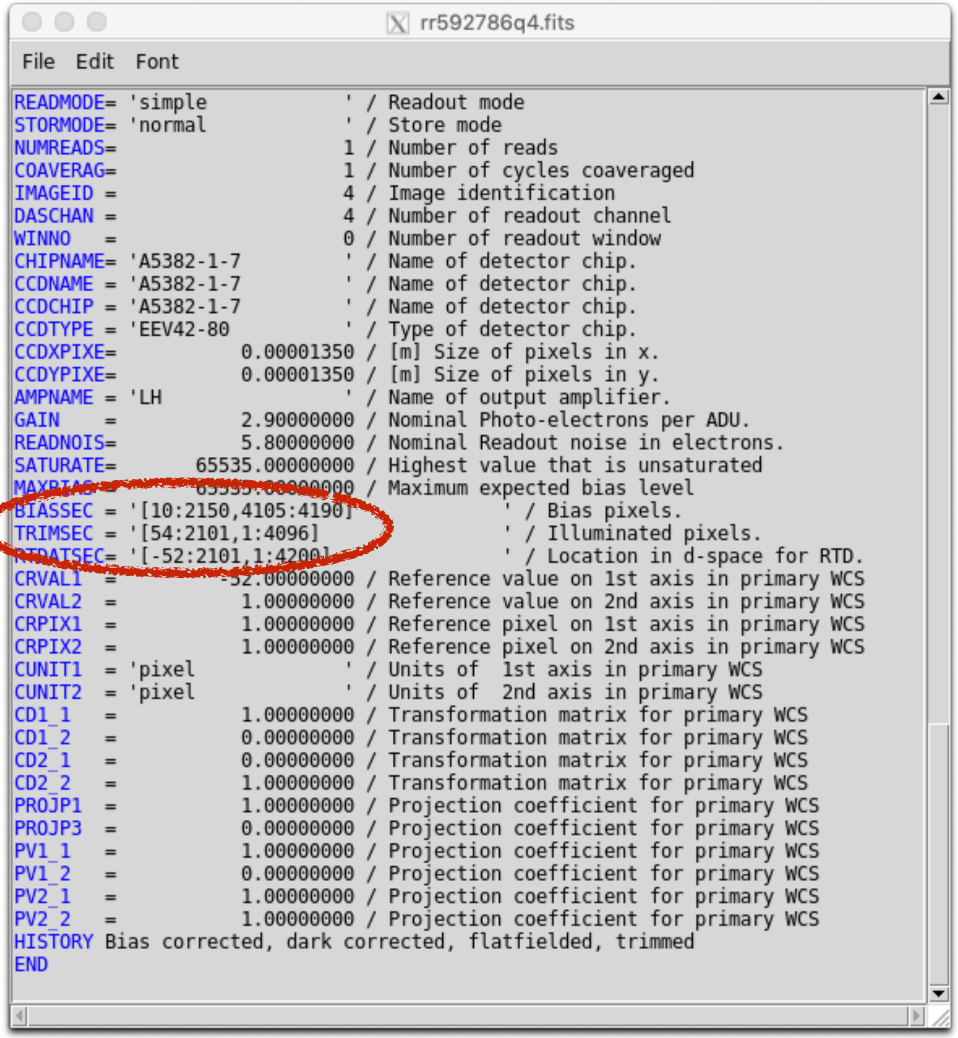


FITS files: how most astronomical data are stored

- The data part contains, well, the data: a two-dimensional FITS (primary) data section contains an image

FITS files: how most astronomical data are stored

- It is important to note that FITS files are based on **FORTRAN**!
  - all counting begins at **1** and not 0
    - keywords that specify data sections (like `TRIMSEC`) start with 1!
  - array counting goes faster in the first dimension than the others, unlike in Python or C
    - so in Python, the dimensions for this file are [4096,2048]



rr592786q4.fits

File Edit Font

```
READMODE= 'simple             ' / Readout mode
STORMODE= 'normal             ' / Store mode
NUMREADS=                   1 / Number of reads
COAVERAG=                   1 / Number of cycles coaveraged
IMAGEID =                   4 / Image identification
DASCHAN =                   4 / Number of readout channel
WINNO   =                   0 / Number of readout window
CHIPNAME= 'A5382-1-7          ' / Name of detector chip.
CCDNAME = 'A5382-1-7          ' / Name of detector chip.
CCDCHIP = 'A5382-1-7          ' / Name of detector chip.
CCDTYPE = 'EEV42-80           ' / Type of detector chip.
CCDXPIXE=          0.00001350 / [m] Size of pixels in x.
CCDYPIXE=          0.00001350 / [m] Size of pixels in y.
AMPNAME = 'LH                 ' / Name of output amplifier.
GAIN    =          2.90000000 / Nominal Photo-electrons per ADU.
READNOIS=          5.80000000 / Nominal Readout noise in electrons.
SATURATE=      65535.00000000 / Highest value that is unsaturated
MAXBIAS =      65535.00000000 / Maximum expected bias level
BIASSEC = '[10:2150,4105:4190]              ' / Bias pixels.
TRIMSEC = '[54:2101,1:4096]                 ' / Illuminated pixels.
RTDATSEC= '[-52:2101,1:4200]                ' / Location in d-space for RTD.
CRVAL1  =        -52.00000000 / Reference value on 1st axis in primary WCS
CRVAL2  =          1.00000000 / Reference value on 2nd axis in primary WCS
CRPIX1  =          1.00000000 / Reference pixel on 1st axis in primary WCS
CRPIX2  =          1.00000000 / Reference pixel on 2nd axis in primary WCS
CUNIT1  = 'pixel              ' / Units of  1st axis in primary WCS
CUNIT2  = 'pixel              ' / Units of  2nd axis in primary WCS
CD1_1   =          1.00000000 / Transformation matrix for primary WCS
CD1_2   =          0.00000000 / Transformation matrix for primary WCS
CD2_1   =          0.00000000 / Transformation matrix for primary WCS
CD2_2   =          1.00000000 / Transformation matrix for primary WCS
PROJP1  =          1.00000000 / Projection coefficient for primary WCS
PROJP3  =          0.00000000 / Projection coefficient for primary WCS
PV1_1   =          1.00000000 / Projection coefficient for primary WCS
PV1_2   =          0.00000000 / Projection coefficient for primary WCS
PV2_1   =          1.00000000 / Projection coefficient for primary WCS
PV2_2   =          1.00000000 / Projection coefficient for primary WCS
HISTORY Bias corrected, dark corrected, flatfielded, trimmed
END
```

A useful program for inspecting FITS files: DS9

- A *very* useful application for viewing FITS files is SAOimage DS9: https://sites.google.com/cfa.harvard.edu/saoimageds9/home?authuser=0
  - You can display data from two- and three-dimensional FITS files and examine their headers easily