ASTR121 Lab 6 Hubbles Law

Zach Burnett* Kevin Lehr†

May 22, 2017

Abstract

This lab investigates Hubble's law $\nu = H_0 \times d$ through comparing observed spectral lines in the galactic data of 15 galaxies with the rest wavelengths of Ca II K (3934 Å [2]) and Ca II H (3969 Å [2])m as well as the H_α-line (6563 Å [3]) obtained from the NASA/IPAC Extragalactic Database (NED) [5]. We observe the shift of each spectral line for each spectrum, and use this to determine its redshift – hence determining its recessional velocity ν. We use the mean redshift-independent distance modulus and mean metric distance, values also reported by NED that averages available galactic distance data found using standard candle techniques, to determine the distance to each galaxy d in a way independent to our determination of the recessional velocity. By plotting ν vs d and creating a linear fit with an intercept of 0, we can determine Hubble's constant H_0, which we found to be $(78.34 \pm 5.18)\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$. Taking the inverse of this constant, we determine the Hubble time to be $(1.248 \times 10^{10} \pm 2.675 \times 10^{-13})$ yr. This is close to the accepted values of the Hubble constant, $H_0 = (67.8 \pm 0.9)\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$ [1] or $H_0 = 70\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$ [4]. More data would improve the results of this calculation.

*University of Maryland Department of Astronomy, College Park, Maryland 20740, zrb@umd.edu
†University of Maryland Department of Astronomy, College Park, Maryland 20740, klehr43@umd.edu

1 Introduction

1.1 Galactic recession

Measurements of Doppler shifts of spectra of nearby and distant galaxies implies all galaxies outside our Local Group are moving away from us at increasing velocity. These measurements were first done by Edwin Hubble in the 1920s, who was not only the first to discover and utilize a standard candle called Cepheid variable stars, but also the first to prove that many objects in the night sky thought to be nebulae were actually distant galaxies. Measuring the redshift of distant galaxies is achieved through studying the location of common, precisely known emission or absorption lines on the galaxy's spectrum that are emitted or absorbed by gas found in its atmosphere. The faster the galaxy is moving away from us, the more stretched out the light from this gas is, and the more redshifted it will be when it reaches our eyes. On the spectrum of the galaxies, this will appear as a translation of these absorption/emission lines to a location in the galactic spectrum that is deeper into the red end of the spectrum. However, these lines will maintain their spacing in reference to each other, making them identifiable even when the reach near-IR wavelengths.

Quantitatively, the recessional speed can be determined by

$$\nu = \frac{(c \times (\lambda_0 - \lambda_s))}{\lambda_s} \tag{1}$$

where ν is the recessional velocity, c is the speed of light equal to 2.998×10^8m/s, λ_s is the stationary wavelength of the emission/absorption line, and λ_0 is the measured wavelength of the spectral line we observe from the galaxy in question.

We will use three absorption/emission lines to determine the redshift of 15 galaxies in this study. The first is the lowest Energy transition in the Balmer series, the H_α line which comes from hot clouds of Hydrogen gas. This line is a transition from the n=3 to the n=2 state in Hydrogen, and has a wavelength of λ_s =6563 Å [3]. The other two come from the absorption by calcium gas, which caused the calcium to become singly ionized. These are the K and H lines of CaII, at wavelengths of 3934 Å [2]and 3969 Å [2], respectively.

1.2 Redshift-Independent Distance Measurements

To investigate Hubble's Law, we need not only the recessional velocity of each galaxy, but also the distance to each. To get a true value of Hubble's constant, we needed to measure this distance in a way that did not depend on redshift of the galaxy, which would be affected from the Doppler shift due to its recession. Astronomers do this by using Standard Candle techniques, techniques which take advantage of objects in space whose luminosity and apparent brightness can be determined without knowing the distance to the star. Usually, these are events such as Type Ia supernovae, which always have a specific and well-documented lightcurve, or some kind of stellar object who brightness varies periodically over a time that is dependent on its maximum luminosity. For many of the distance measurements used, several Standard Candle techniques were used by several various groups and the average was taken for all of these values.

1.3 Hubble's Law

Once we know both the distance and recessional velocity of each galaxy, we can use this to find Hubble's Constant H_0 using Hubble's Law

$$\nu = H_0 \times d \tag{2}$$

which can be plotted as a linear relationship between d and ν where H_0 is the slope of ν vs d with no intercept. Hubble's constant is a value with units of velocity divided by distance, generally in $\mathrm{km\,s^{-1}\,Mpc^{-1}}$. Under the assumption that Hubble's constant has remained the same throughout the history of universe (which is a good enough approximation for this purpose) we can also determine the age of the universe $t = 1/H_0$.

2 Data and Methodology

All of the galaxy data needed for this calculation was taken from the NASA/IPAC Extragalactic Database (NED) [5]. For each galaxy, we pulled from this database spectral data in the Flexible Image Transport System (FITS) format, the format of choice for observational astronomy which carries both a wealth of observational data, and any relevant secondary information (date, time, etc.) stored in the header of the file. To retrieve all of the necessary information from these FITS files, we used a Matlab script, rfits.m, created by Dr. Bolatto from the University of Maryland, College Park. The script can be found in the appendix, and a sample of one of these spectra can be found below in Figure 1. Each of the spectra were taken from $3650.0 - 7100.0\mathring{A}$ with a step interval of $2.0\mathring{A}$. We also took from the database two other values, the redshift-independent mean metric distance and the standard deviation of this value for each galaxy. The standard deviation provided by NED is a standard deviation of all reported distance values using various Standard Candle methods known to the database. Only one galaxy, NGC4750, has no associated standard deviation, which we have conservatively decided to make 2 Mpc. Galaxies that are similar distances away seemed to have a standard deviation between 1.5 and 1.8 Mpc. We reasoned that a possible explanation for the lack of data on this galaxy is because of its difficulty to measure, which would lead to a higher standard deviation if more measurements were performed. Table 1 shows each galaxy and the distances we gathered from NED.

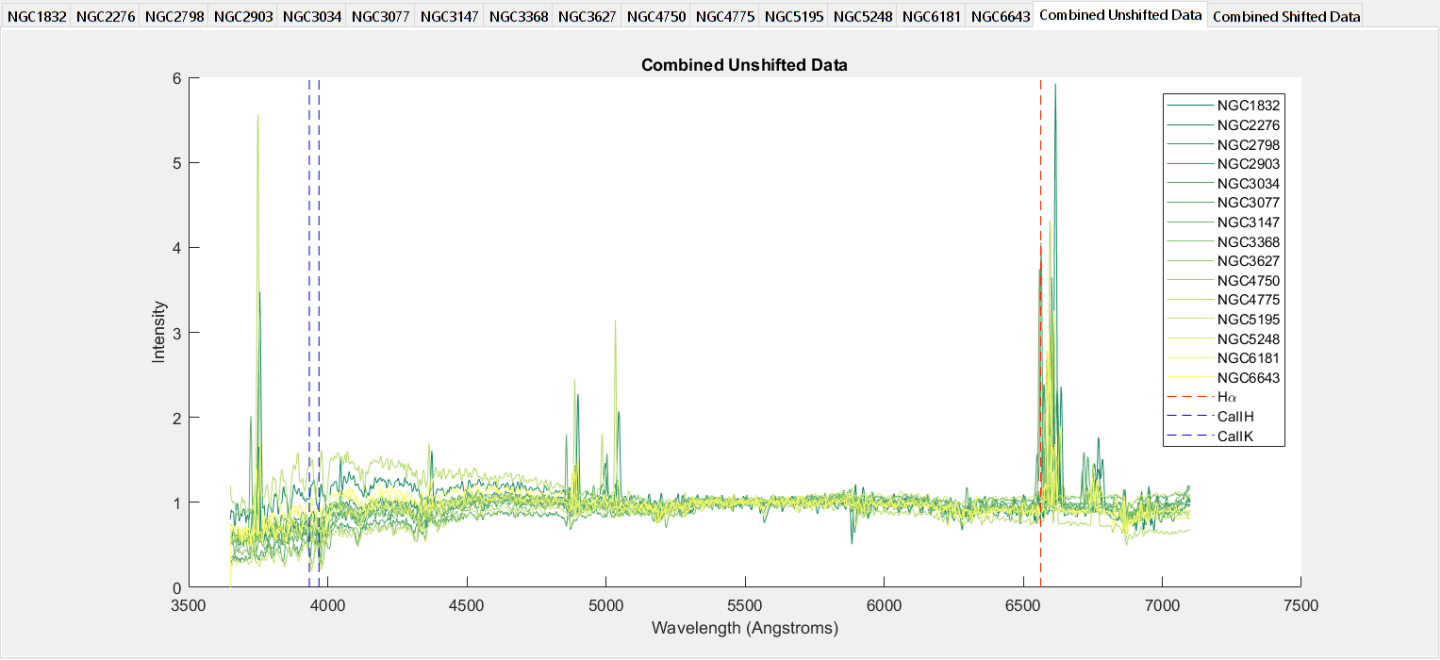

Figure 1: Spectral data from NED with H-alpha and CaII spectral lines overlayed.

The emission line, H_α, and the two absorption lines, CaIIK and CaIIH, were determined in spectrum using an algorithm we developed. For this algorithm we made two important assumptions: one is that all the redshifts would be positive (i.e. every Galaxy we observed would be moving away from us), and two, that the shift was uniform throughout all wavelengths in one galaxy's spectrum. The algorithm takes indexes through possible integer values of the redshift, starting, of course, with no shift and increasing until z = 536. For each potential redshift, it compares the stationary wavelengths of each of the emission/absorption lines to the nearest local wavelength extrema (nearest minimum for CaIIK and H lines, and nearest maximum for the H_α line), and calculates the sum of squares for each. The redshift with the smallest sum of squares gets selected, and weight is put on the peaks that are more prominent and sharp.

3 Analysis

Once the redshift is calculated, it is a simple task to find the recessional velocity using the following equation from the lab handout [2]:

Galaxy	Shift (A)	Unweighted σ (A)	Weighted σ (A)
NGC1832	33.00	1.67	7.00
NGC2276	40.67	4.00	8.67
NGC2798	27.67	2.34	7.67
NGC2903	6.67	2.67	2.67
NGC3034	2.86	0.86	0.86
NGC3077	4.33	4.33	4.33
NGC3147	42.67	1.20	8.67
NGC3368	15.34	2.53	3.34
NGC3627	11.00	2.20	3.00
NGC4750	25.47	3.47	3.47
NGC4775	28.00	2.53	4.00
NGC5195	16.00	0.53	8.00
NGC5248	17.80	2.33	3.80
NGC6181	37.34	0.67	5.34
NGC6643	22.67	0.53	4.67

Table 1: Measured wavelength shift for each galaxy, with weighted and unweighted residuals

Galaxy	Distance (Mpc)	σ (Mpc)	Velocity (m/s)	σ (m/s)
NGC1832	24.86	5.76	2 019 672.55	428 576.11
NGC2276	20.39	6.56	2 488 842.02	530 569.47
NGC2798	24.83	5.21	1 693 293.79	469 373.45
NGC2903	8.07	1.96	408 177.43	163 393.37
NGC3034	3.91	0.69	175 224.60	52 832.56
NGC3077	3.65	0.56	264 978.75	264 978.75
NGC3147	39.61	8.95	2 611 234.05	530 569.47
NGC3368	10.78	1.57	938 542.92	204 190.71
NGC3627	9.46	1.97	673 360.18	183 792.04
NGC4750	26.10	2.00	1 558 662.56	212 350.18
NGC4775	16.28	7.32	1 713 692.47	244 988.06
NGC5195	7.20	2.11	979 340.26	489 772.12
NGC5248	13.25	4.27	1 089 493.09	232 748.85
NGC6181	31.64	5.31	2 284 855.29	326 582.74
NGC6643	19.40	3.27	1 387 313.71	285 785.40

Table 2: The redshift-independent mean metric distance and standard deviation of all reported values for 15 galaxies taken from the NASA/IPAC Extragalactic Database, as well as calculated recessional velocities and their uncertainties

$$\nu = \frac{(c \times (\lambda_0 - \lambda_s))}{\lambda_s} \tag{3}$$

which is discussed above in the introduction. To determine an uncertainty in each wavelength, we used propagation of errors and obtained the formula

$$\sigma_\nu = \frac{c \times \sigma_{\lambda_0}}{\lambda_s} \tag{4}$$

where σ_{λ_0} is the uncertainty in the determination of our shifted wavelength, which can be found in Table 2. These unweighted residuals is the distance between each determined wavelength and the next nearest extrema, whereas the the weighted uncertainty once again takes into accounts the prominence, width, and intensity of that wavelength. The final calculated recessional velocities with their uncertainties are shown in Table 1.

Using the data from Table 1, we can now determine the Hubble constant by doing a linear fit of Hubble's Law. Setting the y-intercept to 0, the slope of this fit will be the Hubble Constant H_0. This fit is shown in Figure 2. The uncertainty in the value of the Hubble constant is the propagation of error for both the uncertainty in the distance and the recessional velocity. This is shown in the following equation,

$$\sqrt{\frac{{\sigma_\nu}^2}{d} + \frac{\nu {\sigma_d}^2}{d^2}} \tag{5}$$

where d and σ_d is the distance and its uncertainty, and ν and $sigma_\nu$ is the recessional velocity and its uncertainty. By taking the inverse of this constant, we get Hubble time, or the age of the universe. The uncertainty for this value is simply using the error propagation on

$$\tau = \frac{1}{H_0} \tag{6}$$

to obtain the equation

$$\tau = \frac{\sigma_{H_0}}{H_0^2} \tag{7}$$

Using these equations, we obtain the result $H_0 = (78.34 \pm 5.18)\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$and $\tau = (1.248 \times 10^{10} \pm 2.675 \times 10^{-13})$ yr.

4 Discussion

Hubble's Law requires that galaxies at 0 distance have 0 recessional velocity, which makes physical sense if we consider that our own galaxy is not constantly expanding around us. Before fixing the value of the intercept to zero, we obtained a value for Hubble's constant that was much closer to the accepted value, $(77.17 \pm 8.26)\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$, but with an intercept of $(102.78 \pm 85.54)\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$. Essentially, having an intercept allows the slope of the line to flatten a little and still create a good fit. Our value for Hubble's constant, $H_0 = (78.34 \pm 5.18)\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$, is already much larger than that of other accepted values for Hubble's constant – $H_0 = (67.8 \pm 0.9)\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$ [1]according to the Planck Mission, or $H_0 = 70\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$ [4], which was used in class. One reason our data may have yielded a larger constant than these other sources is the quantity of the data. For instance, the data featured in the Astrobites article [4] contains measurements from well over 40 galaxies, while ours contains points for only 15. Furthermore, all of the galaxies we studied were relatively close to our own – closer galaxies are moving away from us a slower speed, and therefore a measurement of their recessional velocities would be more difficult to make. Additionally, a value of Hubble's constant that is greater than the accepted value points to an overestimation of the recessional velocities, and therefore an over estimation of the Doppler shift. This could be due to redshifts from extraneous factors, such as gas and other galaxies in between us that could cause gravitational redshift. The time that we determine from our determination of the constant, $(1.248 \times 10^{10} \pm 2.675 \times 10^{-13})$ yr, is a younger estimate of the age of the universe than the accepted value of about 14.8 billion years. Since Hubble's constant and Hubble's time are inversely related, a larger constant value will yield a younger time.

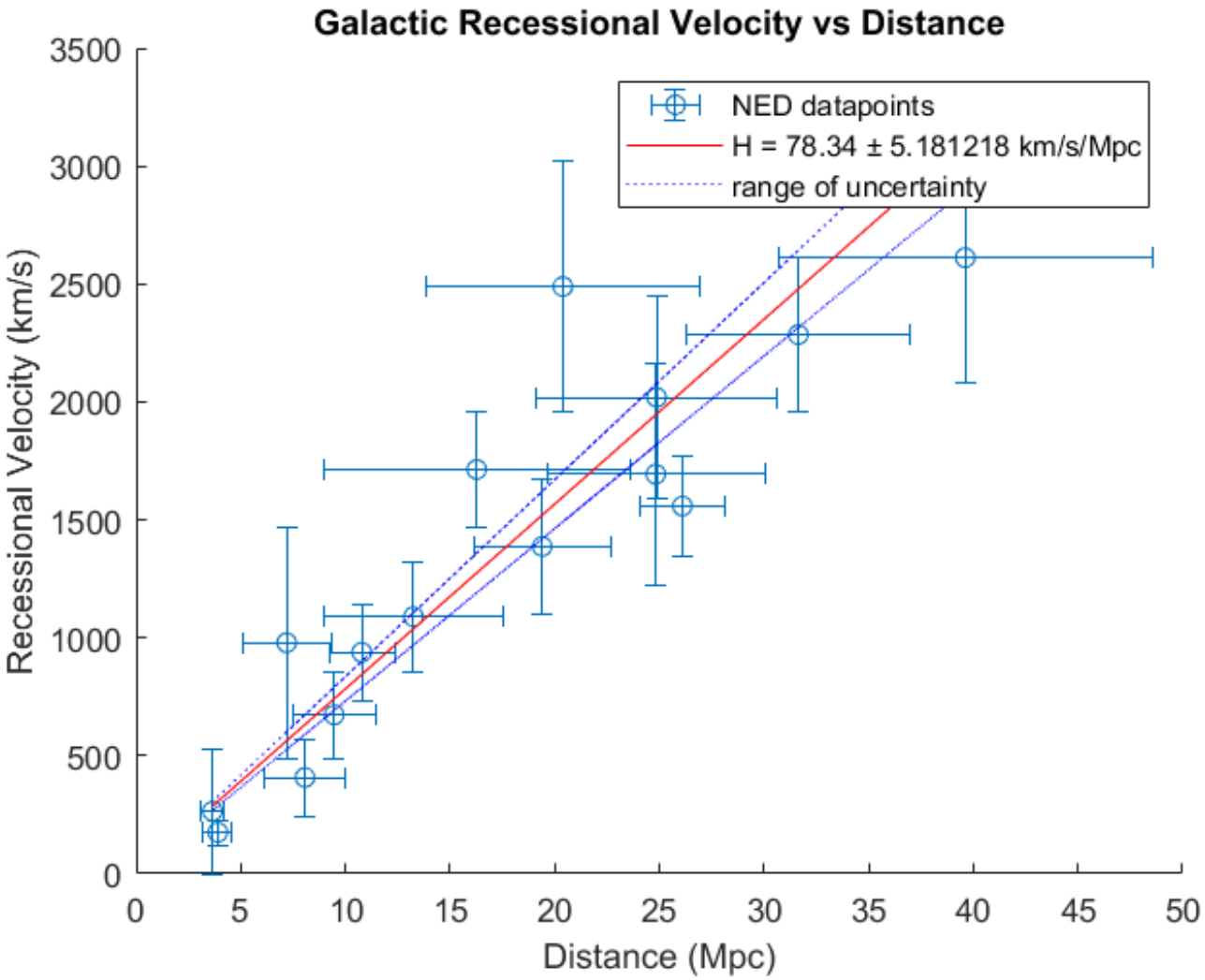

Figure 2: Calculated recessional velocities plotted against NED redshift-independent distances.

5 References

[1] P. A. R. Ade et al. Planck 2015 results. XIII. Cosmological parameters. *Astron. Astrophys.*, 594:A13, 2016.

[2] Joe DeMartini. Lab 6 handout: Hubbles law.

[3] R.M. Eisberg and R. Resnick. *Quantum physics of atoms, molecules, solids, nuclei, and particles.* Quantum Physics of Atoms, Molecules, Solids, Nuclei and Particles. Wiley, 1985.

[4] Caroline Morely. A Brand New Way to Determine Hubbles Constant. *Astrobites*, 2012.

[5] NASA/IPAC. The Nasa/ipac Extragalactic Database NED. Technical report, California Technical Institute, 1990.

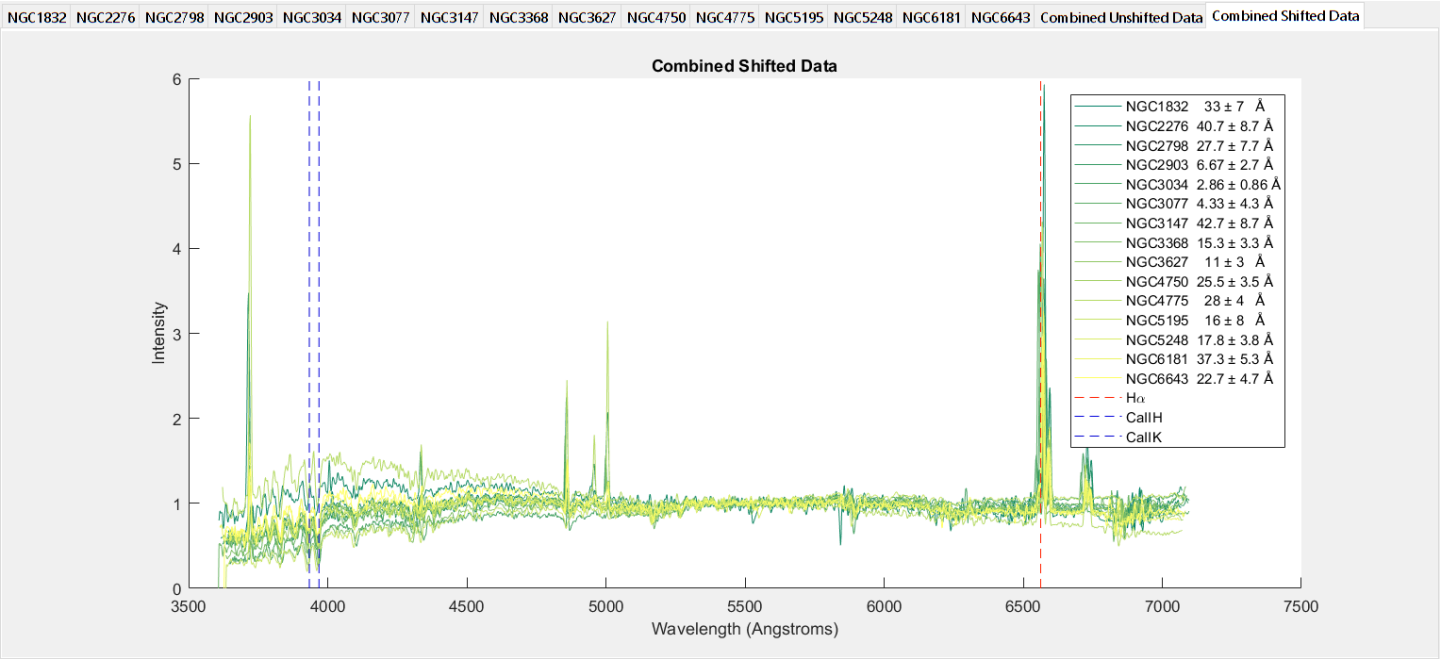

Figure 3: Spectral data from NED with calculated shifts applied and H-alpha and CaII spectral lines overlayed.

6 Appendix

Listings

6.1 Data import using rfits.m

```
% import data into single combined struct
galaxy_data_struct = struct(...
    'NGC1832', rfits('NGC_1832-S-UBVR-k1992.fits.gz'), ...
    'NGC2276', rfits('NGC_2276-S-UBVR-k1992.fits.gz'), ...
    'NGC2798', rfits('NGC_2798-S-UBVR-k1992.fits.gz'), ...
    'NGC2903', rfits('NGC_2903-S-UBVR-k1992.fits.gz'), ...
    'NGC3034', rfits('NGC_3034-S-UBVR-k1992.fits.gz'), ...
    'NGC3077', rfits('NGC_3077-S-UBVR-k1992.fits.gz'), ...
    'NGC3147', rfits('NGC_3147-S-UBVR-k1992.fits.gz'), ...
    'NGC3368', rfits('NGC_3368-S-UBVR-k1992.fits.gz'), ...
    'NGC3627', rfits('NGC_3627-S-UBVR-k1992.fits.gz'), ...
    'NGC4750', rfits('NGC_4750-S-UBVR-k1992.fits.gz'), ...
    'NGC4775', rfits('NGC_4775-S-UBVR-k1992.fits.gz'), ...
    'NGC5195', rfits('NGC_5195-S-UBVR-k1992.fits.gz'), ...
    'NGC5248', rfits('NGC_5248-S-UBVR-k1992.fits.gz'), ...
    'NGC6181', rfits('NGC_6181-S-UBVR-k1992.fits.gz'), ...
    'NGC6643', rfits('NGC_6643-S-UBVR-k1992.fits.gz'));
```

6.2 Calculate and plot Doppler shifts

```
% Find Doppler shifts of each spectra based on fit to rest wavelengths

% define range of wavelengths in angstroms
wavelengths = 3650:2:7100;

% wavelengths from http://physics.nist.gov/PhysRefData/ASD/lines_form.html
% sulfur and nitrogen found at https://en.wikipedia.org/wiki/Forbidden_mechanism

% define rest wavelengths of the first nine entries in the Balmer (Hydrogen)
    ↪ series, as well as of Calcium II (K and H) and Sulfur
spectral_lines = table([6730.815; 6716.726; 6584; 6562.79; 6548; 4861.35;
    ↪ 4340.472; 4101.734; 3970.075; 3968.5; 3933.7; 3889.064], ...
     {'emission'; 'emission'; 'emission'; 'emission'; 'emission'; 'emission'; '
        ↪ emission'; 'emission'; 'emission'; 'absorption'; 'absorption'; 'emission
        ↪ '}, ...
     'VariableNames', {'Wavelength_A', 'Type'}, ...
     'RowNames', {'SII1'; 'SII2'; 'NII1'; 'H\alpha'; 'NII2'; 'H\beta'; 'H\gamma'; '
        ↪ H\delta'; 'H\epsilon'; 'CaIIH'; 'CaIIK'; 'H\zeta'});

% specify selected spectral lines.
selected_spectral_lines = spectral_lines([4,10,11], :);

% define weights assigned to priminence, intensity, and width of peaks and valleys
peak_prominence_weight = 1;
valley_prominence_weight = 3;
peak_intensity_weight = 0;
valley_intensity_weight = 1;
peak_width_weight = 0;
valley_width_weight = 1;

% define range of potential Doppler shifts
potential_shifts = 0:2:round((max(wavelengths) - max(selected_spectral_lines.
    ↪ Wavelength_A)));

% get fieldnames (galaxy names)
galaxy_names = fieldnames(galaxy_data_struct)';

% define plotting colors
data_colors = summer(length(galaxy_names));
spectral_line_colors = jet(length(wavelengths));

% create new tab group to contain plots as tabs
tab_group = uitabgroup;

% create table to hold calculated redshifts
doppler_shifts = array2table(zeros(length(galaxy_names), 3), ...
     'VariableNames', {'Shift_A'; 'Unweighted_Residual_A'; 'Weighted_Residual_A'},
        ↪ ...
     'RowNames', galaxy_names');

% create table to hold weighted and unweighted residuals
```

```
spectral_line_weighted_residuals = array2table(zeros(height(doppler_shifts),
    ↪ height(selected_spectral_lines)), ...
    'VariableNames', strrep(selected_spectral_lines.Properties.RowNames, '\', '_')
        ↪ , ...
    'RowNames', galaxy_names');

spectral_line_unweighted_residuals = array2table(zeros(height(doppler_shifts),
    ↪ height(selected_spectral_lines)), ...
    'VariableNames', strrep(selected_spectral_lines.Properties.RowNames, '\', '_')
        ↪ , ...
    'RowNames', galaxy_names');

% plot separately
for current_galaxy_name_index = 1:length(galaxy_names)
    current_galaxy_name = galaxy_names{current_galaxy_name_index};

    current_intensity_data = galaxy_data_struct.(current_galaxy_name).data;

    % find local minima and maxima
    [local_maxima_intensities, local_maxima_wavelengths, local_maxima_widths,
        ↪ local_maxima_prominences] = findpeaks(current_intensity_data,
        ↪ wavelengths);
    [local_minima_intensities, local_minima_wavelengths, local_minima_widths,
        ↪ local_minima_prominences] = findpeaks(current_intensity_data * -1,
        ↪ wavelengths);

    % create arrays for weighted and unweighted residuals, as well as for applied
        ↪ weights
    weighted_residuals = zeros(length(potential_shifts), height(
        ↪ selected_spectral_lines));
    unweighted_residuals = zeros(length(potential_shifts), height(
        ↪ selected_spectral_lines));
    weights = zeros(length(potential_shifts), height(selected_spectral_lines));

    % iterate over potential wavelength shifts
    for current_potential_shift_index = 1:length(potential_shifts)
        current_potential_shift = potential_shifts(current_potential_shift_index);

        % iterate over spectral lines
        for current_spectral_line_index = 1:height(selected_spectral_lines)
            current_wavelength = selected_spectral_lines.Wavelength_A(
                ↪ current_spectral_line_index);

            % check if emission line
            if strcmp(selected_spectral_lines.Type(current_spectral_line_index), '
                ↪ emission')
                current_weights = (local_maxima_prominences').^
                    ↪ peak_prominence_weight ...
                    ./ (local_maxima_widths).^peak_width_weight ...
                    ./ abs(local_maxima_intensities' - mean(current_intensity_data
                        ↪ )).^peak_intensity_weight;
```

```
        % get the index of the nearest local maximum (weighted and
            ↪ unweighted)
        [~, weighted_nearest_extrema_index] = min(abs((current_wavelength
            ↪ + current_potential_shift) - local_maxima_wavelengths) ./
            ↪ current_weights);
        [~, unweighted_nearest_extrema_index] = min(abs((
            ↪ current_wavelength + current_potential_shift) -
            ↪ local_maxima_wavelengths));

        % populate weights
        weights(current_potential_shift_index, current_spectral_line_index
            ↪ ) = current_weights(weighted_nearest_extrema_index);

        % get residuals (weighted and unweighted) between the shifted
            ↪ spectral line and the nearest local maximum, for uncertainty
            ↪  purposes
        weighted_residuals(current_potential_shift_index,
            ↪ current_spectral_line_index) = ((current_wavelength +
            ↪ current_potential_shift) - local_maxima_wavelengths(
            ↪ weighted_nearest_extrema_index));
        unweighted_residuals(current_potential_shift_index,
            ↪ current_spectral_line_index) = ((current_wavelength +
            ↪ current_potential_shift) - local_maxima_wavelengths(
            ↪ unweighted_nearest_extrema_index));
    else % else assume absorption
        current_weights = (local_minima_prominences').^
            ↪ valley_prominence_weight ...
            ./ (local_minima_widths).^valley_width_weight ...
            ./ abs(local_minima_intensities' - mean(current_intensity_data
                ↪ )).^valley_intensity_weight;

        % get the index of the nearest local minimum (weighted and
            ↪ unweighted)
        [~, weighted_nearest_extrema_index] = min(abs((current_wavelength
            ↪ + current_potential_shift) - local_minima_wavelengths) ./
            ↪ current_weights);
        [~, unweighted_nearest_extrema_index] = min(abs((
            ↪ current_wavelength + current_potential_shift) -
            ↪ local_minima_wavelengths));

        % populate weights
        weights(current_potential_shift_index, current_spectral_line_index
            ↪ ) = current_weights(weighted_nearest_extrema_index);

        % get residuals (weighted and unweighted) between the shifted
            ↪ spectral line and the nearest local minimum, for uncertainty
            ↪  purposes
        weighted_residuals(current_potential_shift_index,
            ↪ current_spectral_line_index) = ((current_wavelength +
            ↪ current_potential_shift) - local_minima_wavelengths(
            ↪ weighted_nearest_extrema_index));
        unweighted_residuals(current_potential_shift_index,
            ↪ current_spectral_line_index) = ((current_wavelength +
```

```
                ↪ current_potential_shift) - local_minima_wavelengths(
                ↪ unweighted_nearest_extrema_index));
        end
    end
end

% find sum of squares of every residual row and sort
weighted_residual_sumsquares = sum((weighted_residuals ./ weights).^2, 2);
[~, sorted_weighted_residual_sumsquares_indices] = sort(
    ↪ weighted_residual_sumsquares);
least_sum_of_squares_index = sorted_weighted_residual_sumsquares_indices(1);

% populate current row of Doppler shift table
doppler_shifts.Shift_A(current_galaxy_name) = potential_shifts(
    ↪ least_sum_of_squares_index);
doppler_shifts.Unweighted_Residual_A(current_galaxy_name) = mean(abs(
    ↪ unweighted_residuals(least_sum_of_squares_index, :)));
doppler_shifts.Weighted_Residual_A(current_galaxy_name) = mean(abs(
    ↪ weighted_residuals(least_sum_of_squares_index, :)));

% populate spectral line tables
spectral_line_unweighted_residuals{current_galaxy_name_index, :} =
    ↪ unweighted_residuals(least_sum_of_squares_index, :);
spectral_line_weighted_residuals{current_galaxy_name_index, :} =
    ↪ weighted_residuals(least_sum_of_squares_index, :);

doppler_shifts.Shift_A(current_galaxy_name) = doppler_shifts.Shift_A(
    ↪ current_galaxy_name) + mean(doppler_shifts.Weighted_Residual_A(
    ↪ current_galaxy_name));

% add new tab to figure window
current_tab = uitab(tab_group, 'Title', current_galaxy_name);
axes('Parent', current_tab);

% start plotting
hold on

% plot shifted intensity data
plot(wavelengths - doppler_shifts.Shift_A(current_galaxy_name),
    ↪ current_intensity_data, 'color', 'k');

% create array for spectral line legend entries
spectral_line_legend_entries = strings(1, height(selected_spectral_lines));

% draw selected spectral lines
for current_spectral_line_index = 1:height(selected_spectral_lines)
    current_wavelength = selected_spectral_lines.Wavelength_A(
        ↪ current_spectral_line_index);
    current_spectral_line_name = selected_spectral_lines.Properties.RowNames{
        ↪ current_spectral_line_index};
    line([current_wavelength current_wavelength], get(gca, 'YLim'), 'color',
        ↪ spectral_line_colors(round((current_wavelength - min(wavelengths)) /
        ↪  range(wavelengths) * length(spectral_line_colors)), :), 'LineStyle'
```

```
            ↪ , '--');

        spectral_line_legend_entries(current_spectral_line_index) = sprintf('%-5s␣
            ↪ %+10.3f␣\x212B', ...
            current_spectral_line_name, ...
            spectral_line_unweighted_residuals.(strrep(current_spectral_line_name,
                ↪ '\', '_'))(current_galaxy_name));
    end


    % add title and legend
    title(current_galaxy_name);
    legend([strcat({'best␣fit␣shift:␣'}, string(doppler_shifts.Shift_A(
        ↪ current_galaxy_name)), {'␣'}, char(197)), spectral_line_legend_entries])
        ↪ ;
    xlabel('Wavelength␣(Angstroms)');
    ylabel('Intensity');

    % end plotting
    hold off
end

% plot combined
current_tab = uitab(tab_group, 'Title', 'Combined␣Unshifted␣Data');
axes('Parent', current_tab);

hold on

for current_galaxy_name_index = 1:length(galaxy_names)
    current_galaxy_name = galaxy_names{current_galaxy_name_index};
    plot(wavelengths, galaxy_data_struct.(current_galaxy_name).data, 'color',
        ↪ data_colors(current_galaxy_name_index, :));
end

for current_spectral_line_index = 1:height(selected_spectral_lines)
    current_wavelength = selected_spectral_lines.Wavelength_A(
        ↪ current_spectral_line_index);
    line([current_wavelength current_wavelength], get(gca, 'YLim'), 'color',
        ↪ spectral_line_colors(round((current_wavelength - min(wavelengths)) /
        ↪ range(wavelengths) * length(spectral_line_colors)), :), 'LineStyle', '--
        ↪ ');
end

title('Combined␣Unshifted␣Data');
legend([galaxy_names, selected_spectral_lines.Properties.RowNames']);
xlabel('Wavelength␣(Angstroms)');
ylabel('Intensity');

hold off

% add new tab to figure window
current_tab = uitab(tab_group, 'Title', 'Combined␣Shifted␣Data');
axes('Parent', current_tab);
```

```
% start plotting combined data
hold on

% plot each galaxy's shifted intensity data
for current_galaxy_name_index = 1:length(galaxy_names)
    current_galaxy_name = galaxy_names{current_galaxy_name_index};
    plot(wavelengths - doppler_shifts.Shift_A(current_galaxy_name),
        ↪ galaxy_data_struct.(current_galaxy_name).data, 'color', data_colors(
        ↪ current_galaxy_name_index, :));
end

% draw selected spectral lines
for current_wavelength_index = 1:height(selected_spectral_lines)
    current_wavelength = selected_spectral_lines.Wavelength_A(
        ↪ current_wavelength_index);
    line([current_wavelength current_wavelength], get(gca, 'YLim'), 'color',
        ↪ spectral_line_colors(round((current_wavelength - min(wavelengths)) /
        ↪ range(wavelengths) * length(spectral_line_colors)), :), 'LineStyle', '--
        ↪ ');
end

% create array for galaxy legend entries
galaxy_legend_entries = strings(1, length(galaxy_names));

% populate galaxy legend entries
for galaxy_name_index = 1:length(galaxy_names)
    galaxy_legend_entries(galaxy_name_index) = sprintf('%7s \t %4.3g \x00B1 %-3.2g
        ↪ \x212B', ...
        galaxy_names{galaxy_name_index}, ...
        doppler_shifts.Shift_A(galaxy_name_index), ...
        abs(doppler_shifts.Weighted_Residual_A(galaxy_name_index)));
end

% add title and legend
title('Combined Shifted Data');
legend([galaxy_legend_entries, selected_spectral_lines.Properties.RowNames']);
xlabel('Wavelength (Angstroms)');
ylabel('Intensity');

% end plotting combined data
hold off
```

6.3 Calculate and plot velocities

```
speed_of_light = 299792458; % meters per second

% create table for radial velocities
galactic_radial_velocities = array2table(zeros(length(galaxy_names), 6), ...
    'VariableNames', {'Distance_Mpc', 'Uncertainty_Mpc', 'Velocity_m_s', '
        ↪ Uncertainty_m_s', 'Velocity_c', 'Uncertainty_c'}, ...
    'RowNames', galaxy_names);
```

```
galactic_radial_velocities.Distance_Mpc = [24.864, 20.390, 24.830, 8.070, 3.907,
    ↪ 3.651, 39.613, 10.781, 9.464, 26.100, 16.275, 7.196, 13.247, 31.638,
    ↪ 19.400]';
galactic_radial_velocities.Uncertainty_Mpc = [5.757, 6.562, 5.214, 1.956, 0.687,
    ↪ 0.555, 8.947, 1.566, 1.971, 2.000, 7.316, 2.114, 4.271, 5.306, 3.267]';

% iterate over galaxies to populate radial velocities
for galaxy_name_index = 1:length(galaxy_names)
    current_galaxy_name = galaxy_names{galaxy_name_index};

    galactic_radial_velocities.Velocity_c(current_galaxy_name) = mean(
        ↪ doppler_shifts.Shift_A(current_galaxy_name) ./ spectral_lines.
        ↪ Wavelength_A);
    galactic_radial_velocities.Uncertainty_c(current_galaxy_name) = mean(sqrt((1
        ↪ ./ spectral_lines.Wavelength_A).^2 .* (doppler_shifts.
        ↪ Weighted_Residual_A(current_galaxy_name)).^2));

    galactic_radial_velocities.Velocity_m_s(current_galaxy_name) =
        ↪ galactic_radial_velocities.Velocity_c(current_galaxy_name) *
        ↪ speed_of_light;
    galactic_radial_velocities.Uncertainty_m_s(current_galaxy_name) =
        ↪ galactic_radial_velocities.Uncertainty_c(current_galaxy_name) *
        ↪ speed_of_light;
end

% define x, y, and error metric
x = galactic_radial_velocities.Distance_Mpc;
y = galactic_radial_velocities.Velocity_m_s;
err = (galactic_radial_velocities.Uncertainty_m_s ./ max(
    ↪ galactic_radial_velocities.Uncertainty_m_s)) ./ ...
    (galactic_radial_velocities.Uncertainty_Mpc / max(galactic_radial_velocities.
        ↪ Uncertainty_Mpc));

hubble_constant_best_fit = sum((x .* y) ./ err.^2) / sum(x.^2 ./ err.^2);

root_sum_squared_deviation_per_degree_freedom_naught = sqrt(sum((y -
    ↪ hubble_constant_best_fit * x).^2) / (height(galactic_radial_velocities) - 1)
    ↪ );

hubble_constant_best_fit_uncertainty =
    ↪ root_sum_squared_deviation_per_degree_freedom_naught / sqrt(sum(x.^2 ./ err
    ↪ .^2));

hubble_time = 1 / (hubble_constant_best_fit / 1000 / 30856776000000000000);
hubble_time_uncertainty = sqrt((-(1 / hubble_constant_best_fit^2))^2 *
    ↪ hubble_constant_best_fit_uncertainty^2);

hold on

% plot data with errorbars
errorbar(galactic_radial_velocities.Distance_Mpc, galactic_radial_velocities.
    ↪ Velocity_m_s / 1000, galactic_radial_velocities.Uncertainty_m_s / 1000,
    ↪ galactic_radial_velocities.Uncertainty_m_s / 1000,
```

```
    ↪ galactic_radial_velocities.Uncertainty_Mpc, galactic_radial_velocities.
    ↪ Uncertainty_Mpc, 'o');

% plot Hubble's constant from Planck
%line(galactic_radial_velocities.Distance_Mpc, (hubble_constant * 1000) *
    ↪ galactic_radial_velocities.Distance_Mpc, 'color', 'r', 'LineStyle', '-');

% plot constant found through data
plot(galactic_radial_velocities.Distance_Mpc, hubble_constant_best_fit *
    ↪ galactic_radial_velocities.Distance_Mpc / 1000, '-r');
plot(galactic_radial_velocities.Distance_Mpc, (hubble_constant_best_fit +
    ↪ hubble_constant_best_fit_uncertainty) * galactic_radial_velocities.
    ↪ Distance_Mpc / 1000, ':b');
plot(galactic_radial_velocities.Distance_Mpc, (hubble_constant_best_fit -
    ↪ hubble_constant_best_fit_uncertainty) * galactic_radial_velocities.
    ↪ Distance_Mpc / 1000, ':b');

% plot zeroline
%line(get(gca, 'XLim'), [0 0], 'color', 'k', 'LineStyle', '--');

title('Galactic Recessional Velocity vs Distance');
xlabel('Distance (Mpc)');
ylabel('Recessional Velocity (km/s)');

legend('NED datapoints', sprintf('H = %.4g \x00B1 %f km/s/Mpc',
    ↪ hubble_constant_best_fit / 1000, hubble_constant_best_fit_uncertainty /
    ↪ 1000), ...
     'range of uncertainty');
hold off
```

6.4 rfits.m

```
function im=rfits(file,bval,bval2)

    % RFITS      Read in FITS
    %   A simple FITS image reader.
    %
    %   RFITS('file') reads a file written in FITS format and returns
    %   all the relevant information in a structure, including the header
    %   values, the data, and coordinate arrays for each axis.
    %
    %   RFITS('file',bval) fills replaces the blank pixels in the image with
    %   the value bval. If bval is not specified, it defaults to NaN.
    %
    %   RFITS('file','options'), where the second parameter is a string
    %   rather than a number, allows you to control the workings of RFITS.
    %   The currently implemented options are:
    %   - head or header: return just the header information, do not read
    %                     in the data (faster and less memory-intensive than
    %                     reading the whole thing).
    %   - nowcs: do not produce "World Coordinate System" matrices, which
    %            is very time consuming and only important if the image or
```

```
%            cube are big enough for the type of projection to be
%            important.
%   - xten#: specify which extension number to read (right now just one
%            digit
%
%   RFITS('file','options',bval) specifies a blanking value.


rlen=2880;  % FITS record length
if (nargin==1),
    bval=NaN;
end
if (ischar(bval)),
    head=~isempty(findstr(bval,'head'));
    nowcs=~isempty(findstr(bval,'nowcs'));
    silent=~isempty(findstr(bval,'silent'));
    silent=isempty(findstr(bval,'verbose'));
    ix=~isempty(findstr(bval,'xten'));
    if (ix>0),z
        xten=str2double(bval(ix+4));
    else
        xten=0;
    end
    if (nargin==1),
        bval=NaN;
    elseif (nargin>2),
        bval=bval2;
    end
else
    head=0;
    nowcs=0;
    xten=0;
    silent=1;
end
h=fopen(file,'r','b'); % Big endian IEEE format
if (h<0),
    error('File not found');
end
if (fscanf(h,'SIMPLE =%s')~='T'),
    error('File is not in FITS format');
end
xti=0;
st=0;
if (xten),
    nr=0;
    xt=[];
    im=rfits(file,'head'); % now read header for entire file even if
        ↪ soliciting xtension
    while (isempty(xt)&&(st==0)),
        nr=nr+1;
        st=fseek(h,nr*rlen,-1);
        xt=fscanf(h,'XTENSION=%s');
    end
```

```
    im.xtension=xt;
    xti=rlen*nr;
end
if (st<0),
    error(ferror(h));
end
fseek(h,xti+80,-1);
bitpix=fscanf(h,'BITPIX =%d');
if isempty(bitpix),
    error('BITPIX keyword not found');
end
fseek(h,xti+2*80,-1);
naxis=fscanf(h,'NAXIS =%d');
if isempty(naxis),
    error('NAXIS keyword not found');
end
im.naxis=naxis;
im.bitpix=bitpix;
im.numpt=zeros(1,naxis);
im.crval=zeros(naxis,1);
im.crpix=zeros(naxis,1);
im.cdelt=ones(naxis,1);
im.crota=zeros(naxis,1);
im.ctype=cell(naxis,1);
im.cunit=cell(naxis,1);
im.bscale=1;
im.bzero=0;
im.bunit='';
im.blank=NaN;
npt=1;
nh=1;
nc=1;
ln=0;
for ax=1:naxis,
    ln=2+ax;
    fseek(h,xti+ln*80,-1);
    fmt=['NAXIS' num2str(ax) ' =%d'];
    im.numpt(ax)=fscanf(h,fmt,1);
    npt=npt*im.numpt(ax);
    if isempty(im.numpt(ax)),
        error(['NAXIS' num2str(ax) ' keyword not found']);
    end
end
flg=1;
while (flg),                                    % Header parsing (new, AB
   ↪ 11/21/2007)
    ln=ln+1;
    fseek(h,xti+ln*80,-1);                      % Reposition file at start of new
       ↪ line
    lnstr=fscanf(h,'%c',80);                    % Read line in
    field=lower(deblank(lnstr(1:8)));
    ixf=findstr(field,'-');
    field(ixf)='_';
```

```
    fend=findstr(lnstr,'/');                    % Find scanning range (look for
       ↪ comment)
    if (isempty(fend)),
        fend=80;
    else
        fend=fend(end)-1;                       % Sometimes there can be more than
           ↪ one '/'
    end
    vstr=lnstr(10:fend);
    ixq=findstr(vstr,'''');                     % Is it a string or a number?
    if (length(ixq)>1),
        vstr=vstr((ixq(1)+1):(ixq(2)-1));
        value=deblank(vstr);                    % "value" always contains the
           ↪ field value
        if (~isempty(value)),
            if (value(end)=='&'),           % Some strings are continued on the
               ↪ next line using this convention
                if (iscontin==0), cfield=field; end
                iscontin=1;
                value=value(1:(end-1));  % drop trailing '&'
            else
                iscontin=0;
            end
        end
    else
        value=sscanf(vstr,'%f',1);
    end
    froot=field(1:(end-1));
    if (~isempty(strmatch(froot,{'crval','crpix','cdelt','crota'},'exact'))),
       ↪ % Special treatment for these fields
        ax=str2double(field(6));
        im.(field(1:5))(ax)=value;
    elseif (~isempty(strmatch(froot,{'ctype','cunit'},'exact'))), % CTYPE,
       ↪ CUNIT are cell arrays
        ax=str2double(field(6));
        im.(field(1:5)){ax}=value;
    elseif (strcmp(field,'history')),       % HISTORY also gets special
       ↪ treatment (can be repeated)
        im.history{nh}=deblank(lnstr(9:end));
        nh=nh+1;
    elseif (strcmp(field,'comment')),       % COMMENT is like history
        im.comment{nc}=deblank(lnstr(9:end));
        nc=nc+1;
    elseif (strcmp(field,'end')),           % END of course signals the end
        flg=0;
    elseif (strcmp(field,'continue')),  % CONTINUE statement, append to
       ↪ previous value
        im.(cfield)=[im.(cfield),value];
    elseif (~isempty(field)),               % otherwise just assign field
        im.(field)=value;
    end
end
```

```
if (isfield(im,'cd1_1')),
    if (~isfield(im,'cd1_2')), % force define cd1_2 for some funky headers
        im.cd1_2=0;
    end
    if (~isfield(im,'cd2_1')),
        im.cd2_1=0;
    end
end

if (head),                                  % Return now if only requesting
   ↪ header
    fclose(h);
    return;
end

im.x=cell(1,naxis);
if (isfield(im,'cd1_1')),                   % CD format for distortion coeffs.
    xv1=(1:im.numpt(1))-im.crpix(1);
    xv2=(1:im.numpt(2))-im.crpix(2);
    [xx,yy]=meshgrid(xv1,xv2);
    im.x(1)={im.cd1_1*xx+im.cd1_2*yy};
    im.x(2)={im.cd2_1*xx+im.cd2_2*yy};
    dcp=1;
elseif (isfield(im,'pc1_1')),               % PC format for distortion coeffs.
    xv1=(1:im.numpt(1))-im.crpix(1);
    xv2=(1:im.numpt(2))-im.crpix(2);
    [xx,yy]=meshgrid(xv1,xv2);
    im.x(1)={im.cdelt(1)*(im.pc1_1*xx+im.pc1_2*yy)};
    im.x(2)={im.cdelt(2)*(im.pc2_1*xx+im.pc2_2*yy)};
    dcp=1;
else                                        % usual axis definition
    for i=1:naxis,
        im.x(i)={((1:im.numpt(i))-im.crpix(i))*im.cdelt(i)};
    end
    dcp=0;
end

if ((~nowcs)&&(~isempty(im.ctype))),
    ctype=char(im.ctype{1});
    if (length(ctype)>=8),
        if (~silent), disp('Projection found, working on WCS...'); end
        if (dcp==0),
            x1=double(im.x{1});
            x2=double(im.x{2});
            % speed up calculations for big arrays
            % AB 3/24/2004
            [yy,xx]=meshgrid(x2,x1);
        else
            xx=double(im.x{1});
            yy=double(im.x{2});
        end
        switch (ctype(6:8)),
            case 'SIN',
```

```
                phi=atan2(yy,xx)+pi/2;
                theta=acos(pi/180*sqrt(xx.^2+yy.^2));
                wcs=1;
            case 'TAN',
                phi=atan2(yy,xx)+pi/2;
                theta=atan(180./(pi*sqrt(xx.^2+yy.^2)));
                wcs=1;
            otherwise,
                disp(['Unsupported projection ' ctype]);
                wcs=0;
        end
    else
        wcs=0;
    end
    if (wcs),
        % make up some space for big arrays
        % AB 3/24/2004
        clear xx yy
        dc=im.crval(2)*pi/180;
        ac=im.crval(1)*pi/180;
        sind=sin(theta).*sin(dc)-cos(theta).*cos(phi).*cos(dc);
        cosdsina=cos(theta).*sin(phi);
        cosdcosa=sin(theta).*cos(dc)+cos(theta).*cos(phi).*sin(dc);
        im.wcs1=180/pi*(ac+atan2(cosdsina,cosdcosa));
        im.wcs2=180/pi*atan2(sind,sqrt(cosdsina.^2+cosdcosa.^2));
    end
end
for i=1:naxis,
    im.x(i)={im.crval(i)+im.x{i}};
end

cp=ftell(h);
np=rlen*ceil(cp/rlen);
fseek(h,np,-1);                % Position at the begining of the data records

if (bitpix==32),
    prec='int32';
elseif (bitpix==16),
    prec='int16';
elseif (bitpix==-32),
    prec='float32';
elseif (bitpix==-64),
    prec='float64';
else
    prec='uint8';
end

nval=ceil(npt);
if (~silent), fprintf(1,'Header read. Getting %d data points...',nval); end
if (length(im.numpt)>2),
    [v,cnt]=fread(h,nval,prec);
    im.data=reshape(v,im.numpt);
    clear v
```

```
elseif (isempty(im.numpt)),
    return
else
    [im.data,cnt]=fread(h,im.numpt,prec);
end
if (~silent), disp('Data read...'); end
if (cnt<nval),
    disp(['File too short at pixel ' num2str(cnt)]);
else
    if (~ischar(bval)),
        if (~silent), disp('Finding blanks in cube...'); end
        ix=find(im.data==im.blank);
        if (~isnan(im.blank)), ix=[ix;find(isnan(im.data))]; end
    end
    if ((im.bscale~=1)||(im.bzero~=0)),
        if (~silent), disp('Rescaling data according to header...'); end
        im.data=im.data*im.bscale+im.bzero;
    end
    if (~ischar(bval)),
        if (~silent), disp('Reassigning blanking values...'); end
        im.data(ix)=bval;
        im.blank=bval;
    end
    %     im.data=reshape(v,im.numpt);
end

fclose(h);
return
```