

# Supernova Cosmology and Chi-square Fitting

## Type Ia Supernovae as Standard Candles

A white dwarf is a star that has exhausted its supply of hydrogen fuel, blows off its outer envelope, and collapses into a dense core of free electrons and nuclei that is supported against further gravitational collapse by the Pauli exclusion principle, which forbids the electrons from getting too close to one another. A Type Ia supernova is a catastrophic thermonuclear explosion of this core. This can happen when the white dwarf accretes so much matter from a neighboring star or the surrounding gas that it becomes unstable. The explosion takes place in seconds and the mostly carbon-oxygen core is converted into a gas of heavier elements that is blown off into interstellar space. This debris remains visible as a supernova remnant.

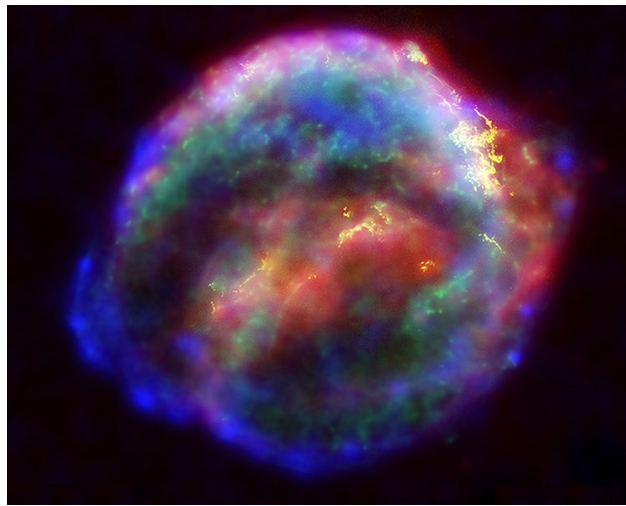


Figure 1: Remnant of the Type Ia supernova SN 1604 [http://en.wikipedia.org/wiki/SN\\_1604](http://en.wikipedia.org/wiki/SN_1604) observed by Johannes Kepler.

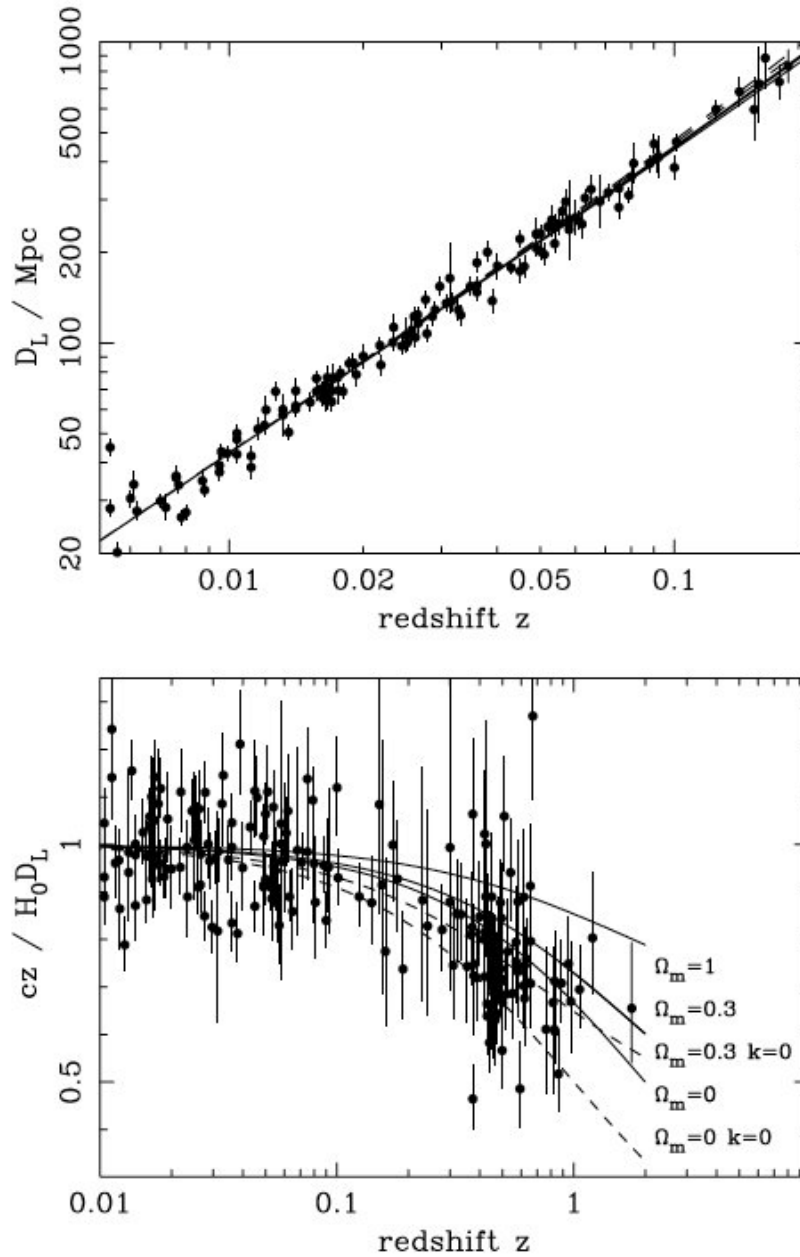
## Supernovae and Hubble's constant

A plot of light intensity emitted as a function of time is called the light curve of the supernova explosion. Type Ia supernovae have characteristic light curves from which their absolute luminosities can be inferred. Their distances from Earth can then be calculated from the observed apparent luminosities.

Thus Type Ia supernovae can be used as standard candles, just like Cepheid variable stars. Because supernova explosions are billions of times brighter than typical stars they can be observed at much greater distances from Earth. Their distances and redshifts can be used to measure the Hubble parameter, see Fig. 2.

Astronomers measure the brightness of an object in the sky using a unit called the magnitude. The absolute magnitude of the object is denoted by  $M$  and the apparent magnitude by  $m$ . The absolute magnitude is defined to be the apparent magnitude observed at a distance of 10 parsecs from the object. The distance modulus is defined as the difference between the apparent and absolute magnitudes

$$\mu \equiv m - M = 5 \log_{10} r - 5, \quad (1)$$



**Figure 19.1:** The type Ia supernova Hubble diagram [23–25]. The first panel shows that for  $z \ll 1$  the large-scale Hubble flow is indeed linear and uniform; the second panel shows an expanded scale, with the linear trend divided out, and with the redshift range extended to show how the Hubble law becomes nonlinear. ( $\Omega_r = 0$  is assumed.) Comparison with the prediction of Friedmann-Lemaître models appears to favor a vacuum-dominated Universe.

Figure 2: From reference [2].

where  $r$  is the distance in parsecs of the object from Earth.

Davis et al.[1] study redshift and distance modulus data using a set of 192 Type Ia supernovae with large redshifts. This data set is available as a file giving redshift  $z$ , distance modulus  $\mu$  and the error in  $\mu$  for each object, as shown by the following snippet:

```

; Columns
; SN= supernova identifier
; z = redshift
; mu= distance modulus
; mu_err = error in distance modulus
;
;      SN      z      mu      mu_err
b013  0.4260  41.98   0.23
d033  0.5310  42.96   0.17
d083  0.3330  40.71   0.14
d084  0.5190  42.95   0.29

```

We will write a program to fit this data to a straight line and estimate Hubble's constant. According to Eq. 1, Supernova b013 is 248.9 Mpc from Earth, and the relativistic Doppler formula

$$1 + z = \frac{1 + \frac{v}{c}}{\sqrt{1 - \frac{v^2}{c^2}}} \quad (2)$$

gives its speed to be  $v = 0.341 c$ . At these cosmological distances and relativistic velocities the equations of general relativity must be used to relate the general relativistic redshift

$$z = \frac{R(t_0)}{R(t)} - 1, \quad (3)$$

to the distance modulus. For distant supernovae the relation can be approximated as follows:

$$\mu = 25 + 5 \log_{10} \left( \frac{cz}{H_0} \right) + 1.086(1 - q_0)z + \dots \quad (4)$$

where  $c$  is measured in km/s and  $H_0$  in km/s/Mpc. This equation follows from a Taylor expansion of the cosmic scale factor

$$R(t) = R(t_0) \left[ 1 + (t - t_0)H_0 - \frac{1}{2}(t - t_0)^2 q_0 H_0^2 + \dots \right], \quad (5)$$

where  $t$  is the time of emission of light from the supernova and

$$H_0 \equiv \frac{\dot{R}(t_0)}{R(t_0)}, \quad \text{and} \quad q_0 \equiv -\frac{R(t_0)\ddot{R}(t_0)}{\dot{R}^2(t_0)}, \quad (6)$$

are Hubble's constant and the deceleration parameter at the present cosmological time  $t_0$ . See Chapter 14 §6 of Weinberg[4] for a detailed derivation of these formulas.

## Chi-square Fitting to a Straight Line

Hubble's 1929 paper did not quote error bars on the data values, although we can fairly safely assume that he quoted values with an appropriate number of significant digits. To fit Hubble's data we used a simple

least-squares fit and estimated the error bar in the data set from the deviations of the data points from the fitted straight line. It is not possible to estimate the reliability of the least-squares fit in the absence of error bars on the data.

If the error bars  $\sigma_i$  are available on the  $y$  values of the set, then it is possible to take them into account by minimizing the chi-square sum, which is defined as

$$\chi^2(a, b) \equiv \sum_{i=0}^{n-1} \left( \frac{y_i - a - bx_i}{\sigma_i} \right)^2. \quad (7)$$

In this expression, data values with small error bars are given more weight than data points with large error bars.

The parameters  $a, b$  are determined by minimizing this function. The following formulas are discussed in detail in Numerical Recipes[3]:

$$b = \frac{1}{S_{tt}} \sum_{i=0}^{n-1} \frac{t_i y_i}{\sigma_i}, \quad a = \frac{S_y - S_x b}{S}, \quad \sigma_a^2 = \frac{1}{S} \left( 1 + \frac{S_x^2}{S S_{tt}} \right), \quad \sigma_b^2 = \frac{1}{S_{tt}}. \quad (8)$$

Here

$$t_i = \frac{1}{\sigma_i} \left( x_i - \frac{S_x}{S} \right), \quad S_{tt} = \sum_{i=0}^{n-1} t_i^2, \quad (9)$$

and

$$S = \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2}, \quad S_x = \sum_{i=0}^{n-1} \frac{x_i}{\sigma_i^2}, \quad S_y = \sum_{i=0}^{n-1} \frac{y_i}{\sigma_i^2}. \quad (10)$$

The goodness of fit can be computed as the probability  $Q$  that the value of  $\chi^2$  should be greater than or equal to its computed value. Because this involves computing an incomplete Gamma function, we will use the simpler criterion that the  $\chi^2$  per degree of freedom is close to unity:

$$\chi^2/\text{d.o.f} \equiv \frac{1}{n-2} \sum_{i=0}^{n-1} \left( \frac{y_i - a - bx_i}{\sigma_i} \right)^2 \approx 1. \quad (11)$$

Because the fit has two parameters  $a, b$ , the number of degrees of freedom is  $\nu = n - 2$ . Two data points can be fit exactly with two parameters. If  $\chi^2 \approx n - 2$ , then the terms in the sum have average magnitude close to one. This implies that the deviations from the straight line are consistent with the error bars. If  $\chi^2/\text{d.o.f} \ll 1$  the fit is too good to be true; and if it is much larger than unity, data cannot be approximated by a straight line.

## C++ strings, files and Gnuplot

A good way of visualizing a data set is to plot it. If you do not have a better plotting program available you should download and install Gnuplot from <http://gnuplot.info/> and learn how to use it.

The following simple program shows how to simulate some data points with error bars and plot them using Gnuplot from a C++ program.

\_\_\_\_\_ Program 1: <http://www.physics.buffalo.edu/phy410-505/topic1/gnuplot.cpp> \_\_\_\_\_

```
#include <cmath>          /* defines abs() */
```

```

#include <cstdlib>      /* defines rand() and system() */
#include <fstream>     /* defines ofstream object for writing to a file */
#include <iostream>
#include <string>      /* defines string objects */

using namespace std;

// path to gnuplot executable - change if located somewhere else

#ifdef _MSC_VER        /* this variable is defined by Visual C++ */
    string gnuplot("C:\\gnuplot\\bin\\wgnuplot.exe");
#else
    /* Linux or Macintosh */
    string gnuplot("/usr/bin/gnuplot");
#endif

int main() {

    string plot_file_name("plot.data");
    ofstream plot_file(plot_file_name.c_str());

    for (int i = 0; i < 50; i++) {
        double x = 0.2 * i;
        double random_error = 0.2 * (-1 + 2.0 * rand() / (RAND_MAX + 1.0));
        double y = sin(x) + random_error;
        double error_bar = abs(random_error);
        plot_file << x << '\t' << y << '\t' << error_bar << '\n';
    }
    plot_file.close();

    ofstream script_file("script.gnu");
    script_file << "set title \'gnuplot.cpp\'\n"
        << "set xlabel \'x\'\n"
        << "set ylabel \'y(x)\'\n"
        << "set yrange [-1.5:1.5]\n"
        << "set grid\n";
    script_file << "plot \'plot.data\' with errorbars, sin(x)\n";
    script_file << "pause mouse\n";
    script_file.close();

    string command(gnuplot + " script.gnu");
    system(command.c_str());
}

```

---

This program uses C++ `string` objects to hold and manipulate character strings. The member function `c_str()` converts a string object to a character string.

A file can be opened for writing by creating an `ofstream` object. The `<<` operator can then be used to write objects to the stream. The member function `close()` flushes the stream and closes the connection to the file.

The function `rand()` returns an integer between 0 and `RAND_MAX`. The function `abs()` returns the absolute

value of its argument. The function `system()` issues its argument as a command to the operating system.

## C++ Program to fit supernova data

The following program estimates the Hubble constant from a data set<sup>[1]</sup> that includes y-error bars. The data is fitted to a linear function using the chi-square merit function. The goodness of the fit is estimated by computing the chi-square per degree of freedom.

The program `hubble.cpp` used static arrays to store the data, which were coded by hand in the program file. The supernova data set is much larger, so the program will read the data directly from the data file. The program uses template `std::vector` objects from the C++ standard template library to hold the data. A `std::vector` is very convenient to use because the number of elements need not be constant. Starting with empty vectors, the program adds one data point to the end of the vector for each line in the data file using the `push_back` member function. The number of elements can be determined by calling the `size` member function.

The program uses `std::string` and `std::stringstream` objects to read and parse each line in the data file.

\_\_\_\_\_ Program 2: <http://www.physics.buffalo.edu/phy410-505/topic1/supernova.cpp> \_\_\_\_\_

```
#include <cmath>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
using namespace std;

// ----- declare global variables -----

string url("http://dark.dark-cosmology.dk/~tamarad/SN/");
string data_file_name("Davis07_R07_WV07.dat");

vector<double>          // C++ std template vector type
  z_data,              // redshift - column 2 in data file
  mu_data,             // distance modulus - column 3
  mu_err_data;        // error in distance modulus - column 4

// ----- function declarations -----

void read_data();      // opens and reads the data file

void chi_square_fit(   // makes a linear chi-square fit
  const vector<double>& x, // vector of x values - input
  const vector<double>& y, // vector of y values - input
  const vector<double>& err, // vector of y error values - input
  double& a,             // fitted intercept - output
  double& b,             // fitted slope - output
  double& sigma_a,      // estimated error in intercept - output
  double& sigma_b,      // estimated error in slope - output
```

```

    double& chi_square          // minimized value of chi-square sum - output
);

// ----- function definitions -----

int main() {

    cout << " Chi-square fit of supernova data to a straight line\n"
         << " Reference: " << url << endl;

    read_data();

    int n = z_data.size();
    vector<double> logz_data(n);
    for (int i = 0; i < n; i++)
        logz_data[i] = log10(z_data[i]);    // to use  $\mu = a + b \log_{10}(z)$ 

    double intercept, slope, intercept_err, slope_err, chisqr;
    chi_square_fit(logz_data, mu_data, mu_err_data,
                  intercept, slope, intercept_err, slope_err, chisqr);

    cout.precision(4);
    cout << " slope = " << slope << " +- " << slope_err << "\n"
         << " intercept = " << intercept << " +- " << intercept_err << "\n"
         << " chi-square/d.o.f = " << chisqr / (n - 2) << endl;
}

void read_data() {

    // create an input file stream object and open the data file
    ifstream data_file(data_file_name.c_str());
    if (data_file.fail())
        cerr << "sorry, cannot open " << data_file_name << endl;

    // read the data file one line at a time
    string line;          // string object to hold current line
    while (getline(data_file, line)) {    // std::getline defined in <string>

        if (line[0] == ';' )          // skip lines starting with semicolon
            continue;

        string SN;          // name of supernova in column 1
        double z, mu, mu_err;    // columns 2, 3, 4
        istringstream is(line);    // string stream object to read line
        is >> SN >> z >> mu >> mu_err;    // read successive column entries

        if (is.fail()) {          // if a read error occurs
            cerr << "error reading line: "
                 << line << endl;    // print an error message
            continue;
        }
    }
}

```

```

    // store the data values in the data vectors
    z_data.push_back(z);
    mu_data.push_back(mu);
    mu_err_data.push_back(mu_err);
}

cout << " read " << z_data.size() << " data values" << endl;

data_file.close();
}

void chi_square_fit(           // makes a linear chi-square fit
    const vector<double>& x,    // vector of x values - input
    const vector<double>& y,    // vector of y values - input
    const vector<double>& err,  // vector of y error values - input
    double& a,                 // fitted intercept - output
    double& b,                 // fitted slope - output
    double& sigma_a,           // estimated error in intercept - output
    double& sigma_b,           // estimated error in slope - output
    double& chi_square)        // minimized value of chi-square sum - output
{
    int n = x.size();

    double S = 0, S_x = 0, S_y = 0;
    for (int i = 0; i < n; i++) {
        S += 1 / err[i] / err[i];
        S_x += x[i] / err[i] / err[i];
        S_y += y[i] / err[i] / err[i];
    }

    vector<double> t(n);
    for (int i = 0; i < n; i++)
        t[i] = (x[i] - S_x/S) / err[i];

    double S_tt = 0;
    for (int i = 0; i < n; i++)
        S_tt += t[i] * t[i];

    b = 0;
    for (int i = 0; i < n; i++)
        b += t[i] * y[i] / err[i];
    b /= S_tt;

    a = (S_y - S_x * b) / S;
    sigma_a = sqrt((1 + S_x * S_x / S / S_tt) / S);
    sigma_b = sqrt(1 / S_tt);

    chi_square = 0;
    for (int i = 0; i < n; i++) {
        double diff = (y[i] - a - b * x[i]) / err[i];

```

```
    chi_square += diff * diff;
}
}
```

---

The `chi_square_fit` function uses reference objects and variables for its arguments. A reference variable is declared by appending an ampersand `&` to its type. The function works on the original object or variable, so any changes it makes persist after it returns. If instead ordinary variables are used, then the function makes a copy. For a `const` input variable copying can waste time and memory, especially the object is very large. If the function modifies a non-`const` variable, the changes are made to the copy and are lost when the function returns.

## Homework Problem

Convert the slope output from the supernova program to km/s/Mpc units and compare with Hubble's value. Divide the supernova data set into two subsets, low redshift and high redshift. Compute the slope separately for each of the two subsets. Can you conclude from your results that the expansion of the Universe is constant, accelerating, or decelerating?

## References

- [1] T.M. Davis et al., "Scrutinizing Exotic Cosmological Models Using ESSENCE Supernova Data Combined with Other Cosmological Probes", *Ap. J.* **666** 716 (2007), <http://arxiv.org/abs/astro-ph/0701510>. Data can be downloaded from <http://dark.dark-cosmology.dk/~tamarad/SN/>.
- [2] K.A. Olive and J.A. Peacock, "Big-bang cosmology", in C. Amsler et al., *Phys. Lett.* **B667**, 1 (2008), <http://pdg.lbl.gov/2009/reviews/rpp2009-rev-bbang-cosmology.pdf>.
- [3] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, "Numerical Recipes in C" (Cambridge University Press 1992), §15.2 Fitting Data to a Straight Line, <http://www.nrbook.com/a/bookcpdf/c15-2.pdf>.
- [4] S. Weinberg, "Gravitation and Cosmology" (Wiley, 1972).