minuit_example

September 6, 2017

1 Minuit Tutorial: Fit Cosmological Models to Type 1a Supernova Data

Created: Summer 2017 Harrison B. Prosper

1.1 Introduction

In this tutorial, I show how to use the CERN function minimization package Minuit, created by CERN scientist Dr. Fred James and released with the CERN data analysis package **ROOT**, to perform simple fits of cosmological models to Type 1a supernova data. For completeness, I start with a bit of background to this problem.

Prerequisite: some familiarity with **Python** and **ROOT** is helpful.

A **Type Ia supernova** is thought to be the thermonuclear detonation of a carbon-oxygen white dwarf whose mass has reached the **Chandrasekhar limit** of about 1.4 times the mass of the Sun. Beyond that mass limit, the "quantum pressure" of the electrons due to the Pauli exclusion principle is insufficient to keep the white dwarf stable. The favored model is a binary system in which a white dwarf accretes hydrogen from its red giant partner until the white dwarf reaches the point of thermonuclear instability. The fact that roughly the same mass explodes each time, namely 1.4 solar masses, which yields an immensely luminous event, makes Type 1a supernovae excellent markers for measuring cosmological distances. While there is some variation in the luminosity of these explosions, it turns out that through a simple empirical procedure it is possible to convert these explosions into standard candles. Given a standard candle, that is, a system of known luminosity and therefore known intrinsic brightness, and given the system's apparent brightness, the inverse square law can be used to infer the distance to the system. If we can determine the distance and redshift $z = (\lambda_o - \lambda_e)/\lambda_e$ for many Type 1a supernovae, we can use these data to infer the parameters of cosmological models. The observed wavelength λ_o is readily measured, while the emitted wavelength λ_e, that is, the wavelength of the light emitted by the supernova in its rest frame, can be inferred by identifying the known spectral lines of the excited atoms and molecules.

Current cosmological models of the universe are based on the **1st Friedmann equation**,

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G}{3}\rho(a) - \frac{Kc^2}{a^2} + \frac{\Lambda c^2}{3}, \tag{1}$$

and the **Friedmann-Lemaitre-Robertson-Walker** (FLRW) metric

$$ds^2 = (cdt)^2 - a^2(t)\left[\frac{dr^2}{1 - Kr^2} + r^2(d\theta^2 + \sin^2\theta\, d\phi^2)\right], \tag{2}$$

where $a(t)$ is a dimensionless function, called the **scale factor**, that models how **proper distances** change with cosmic time t. The proper distance is the spacetime separation between *simultaneous* events. By convention, the scale factor is normalized so that $a(t_0) = 1$ at the present epoch t_0, which is the elapsed time since the Big Bang, defined by $a(0) = 0$. G is the gravitational constant, c the speed of light in vacuum, $\dot{a} \equiv da/dt$, and ρc^2 is the density of all forms of energy excluding that due to the cosmological constant Λ. The constant K, which has units of inverse area, is the global curvature of space.

Care must be exercised in interpreting the symbols of any metric. For example, the radial coordinate r is not the proper distance between the center of the sphere at $r = 0$ and the sphere! The operational meaning of the radial coordinate r is simply $r = \sqrt{A\ /\ (4\pi)}$, where A is the proper area of the sphere centered at $r = 0$, *today*. The proper distance between any two nearby galaxies today, when by choice $a(t_0) = 1$, is the square root of the term in square brackets in the FLRW metric. That distance, $d\chi$, is called the **comoving distance**. It is comoving because the coordinate grid that defines it expands with the universe. Therefore, if a galaxy is stationary with respect to this expanding grid, its comoving coordinates (relative to some origin) do not change. By choice, comoving distances match proper distances today; at any other time t, the proper distance between any two galaxies that are not necessarily nearby is given by

$$d(t) = a(t)\,\chi. \tag{3}$$

The motion of galaxies relative to the expanding comoving grid, the so-called perculiar motion, is small (roughly on the order of 200 km/s) compared with the speed of light. Therefore, on the scale of millions of years, it is a very good approximation to presume that the same time t since the Big Bang can be assigned to all galaxies, that is, that all galaxies share the same surfaces of simultaneity. This is just as well, because it makes it possible to describe the evolution of the universe in a way that is not specific to our particular circumstance. Ironically, however, this is a highly non-relativistic way to conceptual spacetime. In principle, spacetime is to be regarded as a completed 4-dimensional "thing" that doesn't evolve; spacetime just is!

Without loss of generality, the comoving distance between any two galaxies can be oriented to lie along the radial direction, for which $d\theta = d\phi = 0$. Since, by definition, the comoving distance is the proper distance between galaxies today, it follows that

$$\chi = \int_0^r \frac{dy}{\sqrt{1 - Ky^2}} = \sin^{-1}(\sqrt{K}\,r)/\sqrt{K},$$

where, in order to avoid confusion, y is used as the integration variable to distinguish it from its lower and upper bounds. The result can be inverted to give

$$r = \sin(\sqrt{K}\chi)/\sqrt{K}. \tag{4}$$

1.2 First Friedmann Equation

Consider the scaling law $d(t) = a(t)\,\chi$ and its derivative $\dot{d} = \dot{a}\,\chi$ with respect to the universal time t. If we interpret $v(t) = \dot{d}$ as the proper velocity of the expansion, then we arrive at the general form of Hubble's law

$$v(t) = H(t)\,d(t),$$

where $H(t) = \dot{a}/a$ is called the **Hubble parameter** and **Hubble's constant** is $H_0 = H(t_0)$. Therefore, the Friedmann equation at time t_0 is

$$H_0^2 = \frac{8\pi G}{3}\rho(1) - Kc^2 + \frac{\Lambda c^2}{3},$$

or, equivalently,

$$1 = \frac{8\pi G}{3H_0^2}\rho(1) - \frac{Kc^2}{H_0^2} + \frac{\Lambda c^2}{3H_0^2}.$$

Notice that each term on the right-hand side is dimensionless, suggesting that it might be useful to define the dimensionless functions

$$\begin{aligned} \Omega_M(a) &= \frac{8\pi G}{3H_0^2}\rho(a), \\ \Omega_K(a) &= -\frac{Kc^2}{H_0^2}\frac{1}{a^2}, \quad \text{and} \\ \Omega_\Lambda &= \frac{\Lambda c^2}{3H_0^2}, \end{aligned} \tag{5}$$

for arbitrary values of a. It is also useful to define $\Omega(a)$ as the sum of these functions. Then, we can write 1st Friedmann equation as

$$\dot{a} = H_0 a\sqrt{\Omega(a)}. \tag{6}$$

Notice also that the quantities $\Omega_M(1)$, $\Omega_K(1)$, and Ω_Λ satisfy the sum rule

$$\Omega_M(1) + \Omega_K(1) + \Omega_\Lambda = 1, \tag{7}$$

which implies the condition $\Omega(1) = 1$, irrespective of the cosmological model.

For simplicity, we shall take the dimensionless functions written *without* the dependence on a to be the values of the functions evaluated at $a = 1$; for example, Ω_M is a synonym for $\Omega_M(1)$. The alternative convention is to append the subscript 0 to each symbol to denote its value today, e.g., Ω_{M0}.

1.3 Cosmological Models

For our purposes, a cosmological model is a mathematical description of how the dimensionless density $\Omega(a)$ in the model universe evolves with the scale factor a together with dependence of the scale factor on the universal time t, obtained by solving the (1st) Friedmann equation $\dot{a} = H_0 a\sqrt{\Omega(a)}$.

In this tutorial, we consider two cosmological models, the standard model of cosmology ΛCDM in which on every surface of simultanaeity (aka 3D space!) the model universe is filled with a homogeneous distribution of massless particles, a pressureless dust of galaxies, and a cosmological constant Λ. The second model (which I cooked up during an introductory class I taught on modern physics) is a phantom energy model in which the validity of the Friedmann equation is assumed.

1.3.1 ΛCDM Model

During most of the history of the universe, the energy density due to massless particles is negligible. Therefore, in a universe in which matter is conserved, we can write

$$\Omega_M(a) = \frac{\Omega_M}{a^3}.$$

This makes sense because if we double proper distances, we expect the matter density to go down by 2^3. The ΛCDM model is therefore defined by

$$\Omega(a) = \frac{\Omega_M}{a^3} + \frac{1 - \Omega_M - \Omega_\Lambda}{a^2} + \Omega_\Lambda, \tag{8}$$

where Ω_M, Ω_Λ, and H_0 are the free parameters of the model.

1.3.2 A Phantom Energy Model

This model is defined by

$$\Omega(a) = \frac{e^{a^n - 1}}{a^3}, \tag{9}$$

and the parameters n and H_0. Given the degeneracy inherent in the Friedmann equation, any model $\Omega(a)$ is consistent with infinitely many universes, each differing in content! For example, it is possible to regard the phantom energy model as one in which the phantom energy is coupled to matter in such a way that

$$\Omega(a) = \frac{\Omega_M}{a^3} + \frac{e^{a^n - 1} - \Omega_M}{a^3}.$$

But, since neither the curvature parameter Ω_K nor the mass parameter Ω_M are identifiable in this model, we can choose their values at will. In particular, in order to be consistent with observations, we can choose $\Omega_K = 0$ and $\Omega_M \approx 0.30$!

Interestingly, this model can be integrated exactly. We find that

$$H_0 t = \sqrt{e}\, 2^{3/(2n)}\, \Gamma(3/(2n),\, a^n \,/\, 2) \,/\, n, \tag{10}$$

with a future singularity (dubbed the **Big Rip**) characterized by the condition $a \to \infty$ at a *finite* time. In this model, the Big Rip occurs at

$$t_{\text{rip}} = \frac{1}{H_0} \sqrt{e}\, 2^{3/(2n)}\, \Gamma(3/(2n))\, n, \tag{11}$$

where $\Gamma(s, x) = \int_0^x t^{s-1}\, e^{-t}\, dt$ is the **incomplete gamma function**.

1.4 Distance Modulus

In a non-expanding universe, the energy flux f from a supernova of luminosity L (in watts), is given by the inverse square law,

$$f = \frac{L}{4\pi\, r^2}. \tag{12}$$

However, in an expanding universe, the luminosity crossing a sphere of proper area $A = 4\pi\, r^2$ is reduced by the factor $(1 + z)^2$; one factor of $(1 + z)$ arises from the reduction in a photon's energy by the time it reaches the sphere due to the expansion of the universe, and another factor arises from the lower rate at which photons arrive, again because of the expansion. Therefore, in an expanding universe the flux through the sphere today is given by

$$f = \frac{L}{4\pi\, d_L^2}, \tag{13}$$

where

$$\begin{aligned} d_L &= (1+z)\, r, \\ &= (1+z)\ \sin(\sqrt{K}\chi)/\sqrt{K}, \end{aligned} \tag{14}$$

is called the luminosity distance.

Astronomers are fond of odd units. Rather than work with flux, they use apparent magnitude m, defined by $f = q10^{-2m/5} = L/(4\pi d_L^2)$, where q is the flux of objects of zero magnitude. In addition, astronomers define an absolute magnitude M through $f_M = q10^{-2M/5} = L/(4\pi d_M^2)$.

The absolute magnitude of an object is its apparent magnitude if it were placed at a distance of $d_M = 10\,$parsecs, that is, 10^{-5} mega-parsecs (Mpc). The standard measure of distance used in observational cosmology is the distance modulus $\mu = m - M$, which, noting that $f_M/f = 10^{2(m-M)/5} = (d_L/10^{-5})^2$, is given by

$$\mu = 5\log_{10}[(1+z)d_L] + 25. \tag{15}$$

1.5 Comoving Distance

We need to express the comoving distance χ in terms of the parameters of the cosmological model. To that end, consider the worldline of a photon in spacetime. Massless particles travel on null geodesics, defined by $ds = 0$, for which $cdt = a(t)d\chi$. The latter expression is deceptively simple. The left-hand side states that a photon travels a distance cdt from some event $A(t)$ to a *non-simultaneous* event $B(t + dt)$. But on the right-hand side, the comoving distance $d\chi$ between *simultaneous* events $A(t)$ and $B(t)$ is scaled by the factor $a(t)$ to give the proper distance $a(t)d\chi$ between these events. The correspondence between the distance traveled by light and the proper distance permits the use of the worldline of a photon as a standard ruler whose measure, namely the distance traveled by light in the time interval $(t, t + dt)$, can be scaled by $1/a(t)$ to yield an expression for the comoving distance $d\chi = cdt\ /\ a(t)$ that depends on the cosmological model. In order to compute the comoving distance χ between a supernova explosion at time t_1 whose light is detected, now, at time t_0, we need merely compute the integral

$$\begin{aligned} \chi &= \int_{t_1}^{t_0} \frac{cdt}{a}, \\ &= c\int_{1/(1+z)}^{1} \frac{da}{a\dot{a}}, \end{aligned} \tag{16}$$

where $a(t_1) = 1/(1+z)$ is the scale factor at the time of the supernova explosion and $a(t_0) = 1$ is the scale factor when the light is detected. After replacing $\dot{a}$ with the right-hand side of the Friedmann equation $\dot{a} = H_0 a\sqrt{\Omega(a)}$, and defining

$$\boxed{u(z) \equiv \int_{1/(1+z)}^{1} \frac{da}{a^2\sqrt{\Omega(a)}}}, \tag{17}$$

we find

$$\chi = \frac{c}{H_0}\,u(z). \tag{18}$$

In the luminosity distance, $d_L = (1+z)\,\sin(\sqrt{K}\chi)/\sqrt{K}$, the product $\sqrt{K}\chi$ is necessarily dimensionless. Recall that $\Omega_K = -Kc^2/H_0^2$; therefore, $\sqrt{K} = \sqrt{-\Omega_K}\,H_0/\,c$. Consequently, $\sqrt{K}\chi = \sqrt{-\Omega_K}\,u$. Therefore, we can rewrite the luminosity distance as the product

$$d_L = \frac{c}{H_0}\,(1+z)\,\sin(\sqrt{-\Omega_K}\,u)\;/\;\sqrt{-\Omega_K}, \tag{19}$$

of the **Hubble distance** $c\,/\,H_0$ and a dimensionless function of the cosmological parameters, which leads to the final form of the distance modulus, namely,

$$\boxed{\mu = 5\log_{10}[(1+z)\,\sin(\sqrt{-\Omega_K}\,u)\;/\;\sqrt{-\Omega_K}] - 5\log_{10}(H_0) + 5\log_{10}(c) + 25}. \tag{20}$$

Note that $\sin(\sqrt{-\Omega_K}\,u)\;/\;\sqrt{-\Omega_K} \to u$ as $\Omega_K \to 0$, that is, in the limit of a globally flat spatial geometry.

1.6 Lifetime of the Universe

Through a slight rearrangement of the Friedmann equation, $\dot{a} = H_0 a\sqrt{\Omega(a)}$, we can find t as a function of the scale factor, a,

$$t = \frac{1}{H_0}\int_0^a \frac{dy}{y\sqrt{\Omega(y)}}. \tag{21}$$

By construction, the elapsed time since the Big Bang, t_0, is obtained by setting $a = 1$ in the function $t(a)$.

1.7 Fitting Models to Supernova Data

For each supernova, i, of which there are $N = 580$ in the **Union 2.1 compilation**, the data comprises the redshift z_i, which is measured with negligible error, the measured distance modulus x_i and the associated uncertainty σ_i, which is taken to be the standard deviation of a Gaussian likelihood,

$$p(x_i\,|\,z_i,\sigma_i,\theta) = \text{Gauss}(x_i,\mu(z_i,\theta),\sigma_i),$$

where θ denote the cosmological parameters. The supernova data are **heteroscadastic**, which means that, in general, the standard deviations σ_i vary from one supernova to the next. Neglecting correlations between the measurements, we can write the overall likelihood of the supernova data as

$$p(x\,|\,z,\sigma,\theta) = \prod_{i=1}^{N} p(x_i\,|\,z_i,\sigma_i,\theta).$$

The best fit values are obtained via maximum likelihood, or equivalently, by minimizing the negative log-likelihood, which for data with Gaussian errors is the same as minimizing the χ^2,

$$\chi^2(\theta) = \sum_{i=1}^{N} \left[\frac{x_i - \mu(z_i, \theta)}{\sigma_i} \right]^2.$$

For a good fit, we expect $\min[\chi^2]/\text{NDF} \approx 1$, where the number of degrees of freedom (NDF) $= N - P$, where P is the number of free parameters.

1.8 Other Dependencies

This tutorial uses the github package **histutil**, which contains some simple ROOT-based utilities. To install this package do

git clone https://github.com/hbprosper/histutil.git

and source the setup.sh script.

```
In [1]: import os, sys
        import ROOT
        from histutil import setStyle, mkgraph, mkgraphErrors, mkhist1, Scribe
        %jsroot off
```

```
Welcome to JupyROOT 6.10/02
```

1.8.1 Model parameters

- ID: model identifier
- free: specifies whether parameter is free
- name: name of parameter
- guess: starting (or fixed) value of parameter
- step: step size during minimization
- min, max: parameter range

```
In [2]: #                         ID    free,  name,  guess,    step,    min,  max
        PARAMS = {'LCDM' :      [0, [(True,  'OM',      1,    1.e-3,      0,   10),
                                     (True,  'OL',      0,    1.e-3,    -10,  10),
                                     (True,  'H0',     70,    1.e-2,      0,  200)]],

                  'phantom':    [1, [(False, 'OM',      1,    1.e-3,      0,   10),
                                     (False, 'OL',      0,    1.e-3,    -10,  10),
                                     (True,  'H0',     70,    1.e-2,      0,  200),
                                     (True,  'n',       2,    1.e-3,      0,   10)]]
                 }
        # define ranges for redshifts and distance moduli
        ZMIN   = 0.0
        ZMAX   = 1.5
        MUMIN  = 32.0
        MUMAX  = 48.0
```

1.8.2 Choose model

- MODEL = 'LCDM' or 'phantom'

```
In [3]: MODEL = 'LCDM'
```

1.8.3 Compile C++ classes Model and CosmicCode

- **Model** defines $\Omega(a)$ for the cosmological models
- **CosmicCode** computes the distance modulus
- import codes into Python global namespace

```
In [4]: def compileCode(modelname, modelparams):
            ROOT.gROOT.ProcessLine(open('../CosmicCode.cc').read())
            from ROOT import CosmicCode, Model
            # make sure model name is valid
            if not modelparams.has_key(modelname):
                sys.exit("** unknown model %s" % modelname)

            # get model id and parameters
            modelid, params = modelparams[modelname]
            model = Model(modelid)
            code  = CosmicCode(model)
            return (code, params)
```

```
In [5]: code, params = compileCode(MODEL, PARAMS)
```

```
LCDM model
```

1.8.4 Read Type 1a supernova data

```
In [6]: # ---------------------------------------------------------
        # read Type Ia data
        # format: name, z, x, dx
        # name:     name of supernova
        # z         measured redshift of supernova
        # x +/- dx: measured distance modulus
        # ---------------------------------------------------------
        def readData(filename):
            import os
            from array import array
            from string import split, atof
            if not os.path.exists(filename):
                sys.exit("** can't open file %s" % filename)

            # skip first 5 lines and convert 2nd through 4th
```

```
        # columns to floats
        data = map(lambda x: map(atof, x[1:-1]),
                      map(split,
                          open(filename).readlines()[5:]))

        z  = array('d')
        x  = array('d')
        dz = array('d')
        dx = array('d')

        ndata = len(data)
        print "number of observations: %d" % ndata
        print "%5s\t%10s\t%10s +/- %-10s" % ('', 'z', 'x', 'dx')
        for ii, d in enumerate(data):
            z.append(d[0])
            x.append(d[1])
            dz.append(0)
            dx.append(d[2])
            if ii % 100 == 0:
                print "%5d\t%10.3f\t%10.4f +/- %-10.4f"%\
                  (ii, z[-1], x[-1], dx[-1])
        return (z, x, dz, dx)
```

```
In [7]: data = readData('../SCPUnion2.1_mu_vs_z.txt')
```

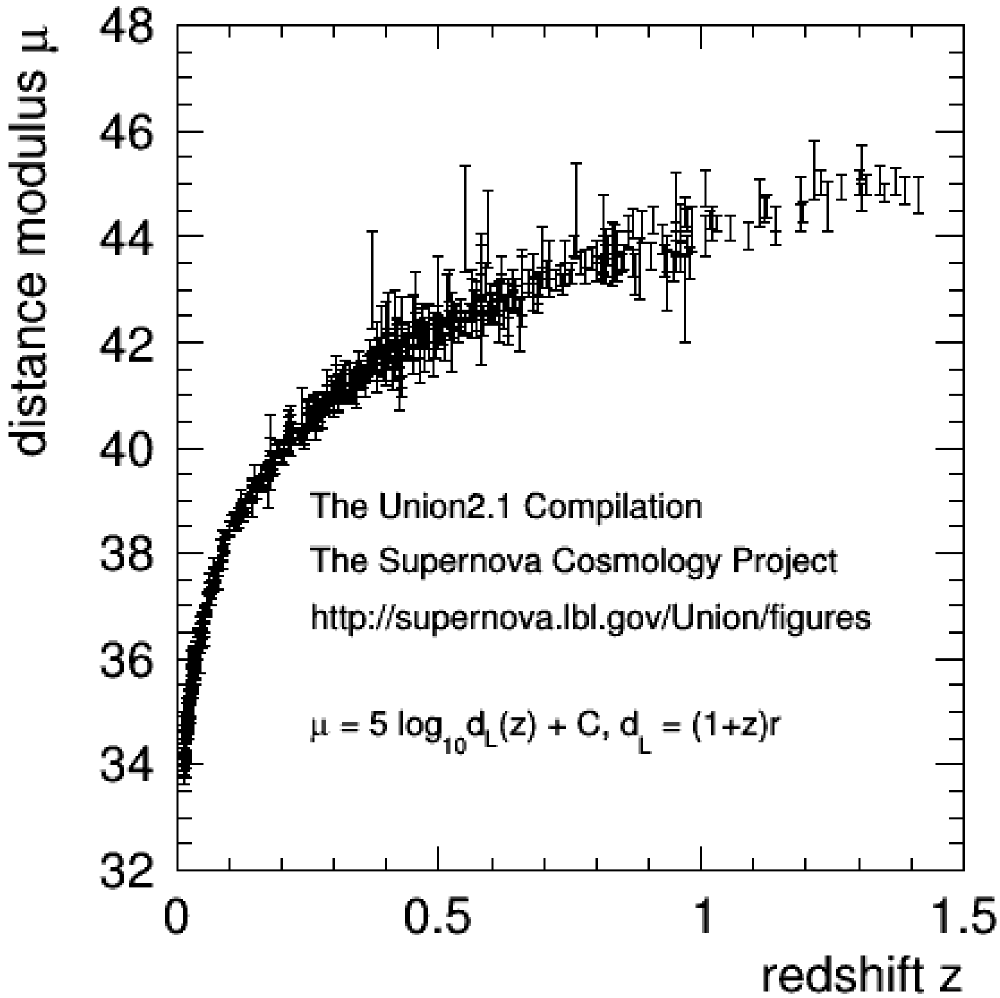
```
number of observations: 580
                     z                  x +/- dx
    0            0.028            35.3466 +/- 0.2239
  100            0.065            37.3067 +/- 0.1628
  200            0.194            39.9615 +/- 0.1264
  300            0.620            43.2280 +/- 0.3903
  400            0.710            43.0220 +/- 0.1843
  500            0.564            42.3729 +/- 0.2920
```

```
In [8]: setStyle()
        def plotData(data, code, zmin=0.0, zmax=1.5, mumin=32.0, mumax=48.0):
            from array import array

            z, x, dz, dx = data
            ndata = len(z)

            g = mkgraphErrors(z, x, dz, dx,
                              "redshift z",
                              "distance modulus #mu",
                              zmin, zmax,
                              ymin=mumin,
                              ymax=mumax,
```

```
                          color=ROOT.kBlack)
    ROOT.SetOwnership(g, 0)
    g.SetName('dataplot')
    g.SetTitle('')
    g.SetMarkerSize(0.2)

    c = ROOT.TCanvas("fig_data", "SN1a data", 500, 500)
    ROOT.SetOwnership(c, 0)

    c.cd()
    g.Draw("ap")

    xpos = 0.32
    ypos = 0.50
    textsize = 0.035

    scribe = Scribe(xpos, ypos, textsize)
    ROOT.SetOwnership(scribe, 0)
    scribe.write("The Union2.1 Compilation")
    scribe.write("The Supernova Cosmology Project")
    scribe.write("http://supernova.lbl.gov/Union/figures")
    scribe.write("")
    scribe.write("#mu = 5 log_{10}d_{L}(z) + C, d_{L} = (1+z)r")

    c.Draw()
    c.SaveAs(".pdf")
    return g
```

```
In [9]: gd = plotData(data, code)
```

```
Info in <TCanvas::Print>: pdf file ./fig_data.pdf has been created
```



1.8.5 χ^2 function to be minimized

```
In [10]: def chisq(npar, grad, fval, xval, flag):
             fval[0] = 0.0
             ndata = len(data[0])
             for i in xrange(ndata):
                 z  = data[0][i]
                 x  = data[1][i]
                 dx = data[3][i]
                 mu = code.distanceModulus(z, xval)
                 c  = (x - mu)/dx
                 fval[0] += c*c
```

1.8.6 Setup Minuit

- instantiate a Minuit object
- specify parameters

```
In [11]: def setupMinuit(modelname, params):
             from array import array

             print 'setup Minuit'

             PRINT_LEVEL = -1   # -1 => quiet, 1 => loud
             UP = 1.0 # appropriate for 68% CL using chisq
             npar = len(params)
             minuit = ROOT.TMinuit(npar)
             minuit.SetFCN(chisq)
             minuit.SetErrorDef(UP)
             minuit.SetPrintLevel(PRINT_LEVEL)

             status = ROOT.Long() # needed for integers passed by refence (int& ii)
             print "%10s\t%-10s %10s %10s %10s %10s" % \
             ('free', 'param', 'guess', 'step', 'min', 'max')

             for ii, t in enumerate(params):
                 print "%10s\t%-10s %10.2e %10.3e %10.3e %10.3e" % t
                 free, name, guess, step, pmin, pmax = t
                 minuit.mnparm(ii, name, guess, step, pmin, pmax, status)
                 if status != 0:
                     sys.exit("** mnparm(%s) status = %d" % (name, status))

                 if not free:
                     stat = minuit.FixParameter(ii)
                     if stat != 0:
                         sys.exit("** FixParameter(%s) status = %d" % (name, stat))
             return minuit
```

```
In [12]: minuit = setupMinuit(MODEL, params)
```

```
setup Minuit
      free          param             guess         step          min          max
      True          OM             1.00e+00   1.000e-03   0.000e+00   1.000e+01
      True          OL             0.00e+00   1.000e-03  -1.000e+01   1.000e+01
      True          H0             7.00e+01   1.000e-02   0.000e+00   2.000e+02
 **********
 **     1 **SET ERRDEF            1
 **********
```

1.8.7 Perform fit

```
In [13]: def performFit(modelname, minuit, params):
             from array import array

             LINE    = "="*80
             print LINE
             print "running Minuit"

             # define parameters
             MAXITER    = 1000
             TOLERANCE = 1.e-4
             args = array('d')
             args.append(MAXITER)
             args.append(TOLERANCE)

             swatch = ROOT.TStopwatch()
             swatch.Start()

             status = ROOT.Long()
             minuit.mnexcm("MIGRAD", args, 2, status)
             if status != 0: sys.exit("** mnexcm status = %d" % status)
             print "real time: %10.3f s" % swatch.RealTime()

             # print results
             print
             filename = '%s_fit.txt' % modelname
             out    = open(filename, 'w')
             value = ROOT.Double() # needed for passing doubles by reference
             error = ROOT.Double()
             results = []
             print "%10s\t%11s\t%11s" % ('name', 'value', 'uncertainty')
             for ii, t in enumerate(params):
                 name = t[1]
                 minuit.GetParameter(ii, value, error)
                 record = "%10s\t%11.3f\t%11.3f" % (name, value, error)
                 out.write('%s\n' % record)
                 print record
                 results.append((float(value), float(error)))
             out.close()
             print LINE
             return results
```

```
In [14]: results = performFit(MODEL, minuit, params)
```

```
================================================================================
running Minuit
real time:      0.228 s
```

```
      name                value        uncertainty
        OM                0.279              0.070
        OL                0.725              0.117
        H0               69.824              0.438
================================================================================
```

1.9 Plot Results

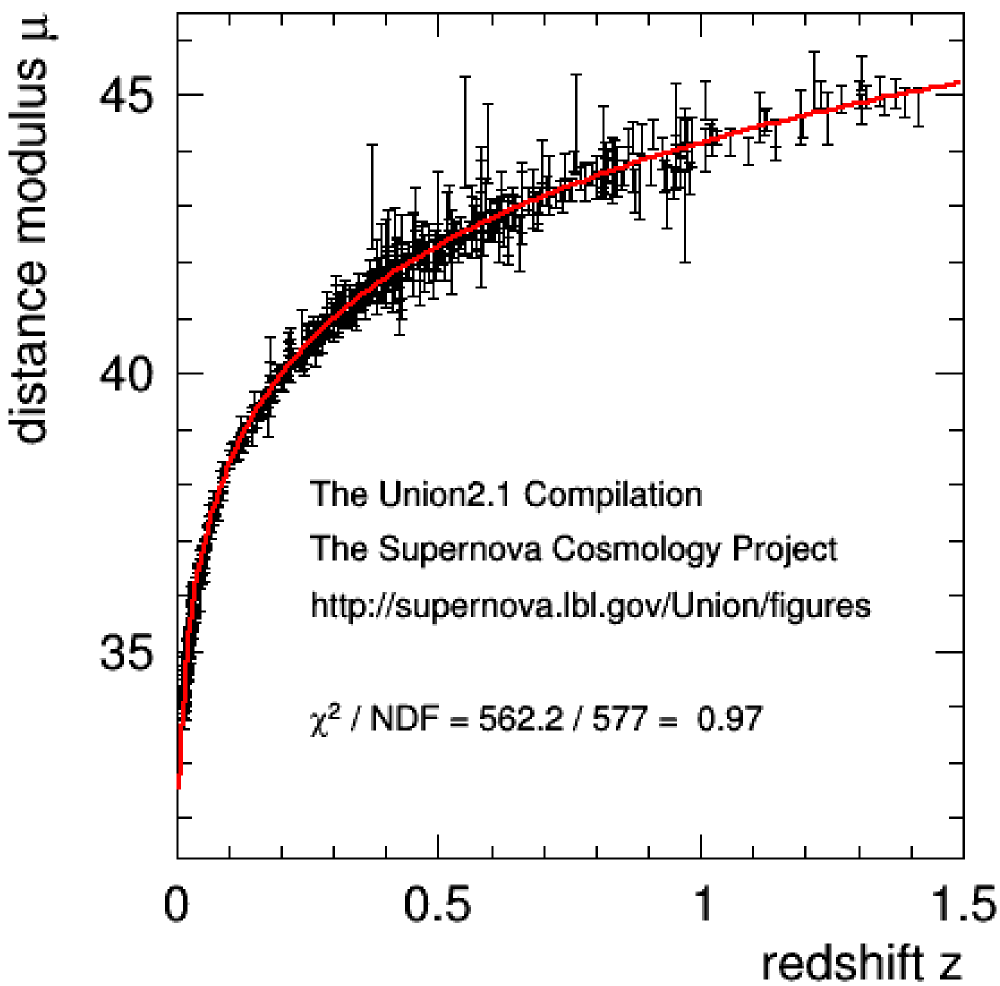
```
In [15]: def annotate(modelname, scribe, results, offset=0.01):
             if modelname == 'LCDM':
                 OM = tuple(results[0])
                 OL = tuple(results[1])
                 scribe.write("#LambdaCDM model")
                 scribe.write("")
                 scribe.write("#Omega(a) = #frac{#Omega_{M}}{a^{3}} + "
                       "#frac{(1 - #Omega_{M} - #Omega_{#Lambda})}{a^{2}}"
                               " + #Omega_{#Lambda}",
                               offset)
                 scribe.write("")
                 scribe.write("#Omega_{M} = %5.2f #pm %-5.2f" % OM,
                               offset)
                 scribe.write("#Omega_{#Lambda} = %5.2f #pm %-5.2f " % OL,
                               offset)
             else:
                 n, dn = results[3]
                 x = 3.0/(2*n)
                 G = ROOT.TMath.Gamma(x)
                 T = G*ROOT.TMath.Sqrt(2.718)*2**x/n
                 scribe.write("phantom model")
                 scribe.write("")
                 scribe.write("#Omega(a) = #frac{#Omega_{M}}{a^{3}} + "
                           "#frac{e^{a^{n}-1} - #Omega_{M}}{a^{3}}", offset)
                 scribe.write("")
                 scribe.write("H_{0}t = #sqrt{e} 2^{3/(2n)} #Gamma(3/(2n), a^{n}/2)/n",
                              offset)
                 scribe.write("")
                 scribe.write("H_{0}t_{rip} = #sqrt{e} 2^{3/(2n)} #Gamma(3/(2n))/n = %4.2f"%\
                              T, offset)
                 scribe.write("")
                 scribe.write("where #Gamma(s, x) = #int_{0}^{x} t^{s-1} e^{-t} dt", offset)
                 scribe.write("and n = %4.2f" % n, offset)

In [16]: def plotModel(modelname, code, results, npar,
                       gd, data,
                       zmin=ZMIN, zmax=ZMAX, mumin=MUMIN, mumax=MUMAX):
             from array import array
             p = array('d')
```

```
for value, error in results:
    p.append(value)

z, x, dz, dx = data
ndata = len(z)

# compute chisq
chi2 = 0.0
for i in xrange(ndata):
    mu = code.distanceModulus(z[i], p)
    c = (x[i] - mu)/dx[i]
    chi2 += c*c
NDF = ndata - npar # number of degrees of freedom

# compute curve
nz = 100
zstep = (zmax - zmin) / nz
zz = array('d')
mu = array('d')
for ii in xrange(nz):
    zz.append( (ii+0.5)*zstep )
    mu.append( code.distanceModulus(zz[-1], p) )

g = mkgraph(zz, mu,
            "redshift z",
            "distance modulus #mu",
            zmin, zmax, color=ROOT.kRed, lwidth=2)
ROOT.SetOwnership(g, 0)
g.SetName('model')
g.SetTitle('')

c = ROOT.TCanvas("fig_%s_fit" % modelname, "SN1a model fit", 500, 500)
ROOT.SetOwnership(c, 0)

c.cd()
g.Draw('ac')
gd.Draw('psame')
g.Draw('csame')

xpos = 0.32
ypos = 0.50
textsize = 0.035
scribe = Scribe(xpos, ypos, textsize)
ROOT.SetOwnership(scribe, 0)
scribe.write("The Union2.1 Compilation")
scribe.write("The Supernova Cosmology Project")
scribe.write("http://supernova.lbl.gov/Union/figures")
scribe.write("")
```

```
        scribe.write("#chi^{2} / NDF = %5.1f / %d = %5.2f" % (chi2, NDF, chi2/NDF))

        c.Draw()
        c.SaveAs(".pdf")
```

```
In [17]: plotModel(MODEL, code, results, minuit.GetNumFreePars(), gd, data)
```

```
Info in <TCanvas::Print>: pdf file ./fig_LCDM_fit.pdf has been created
```

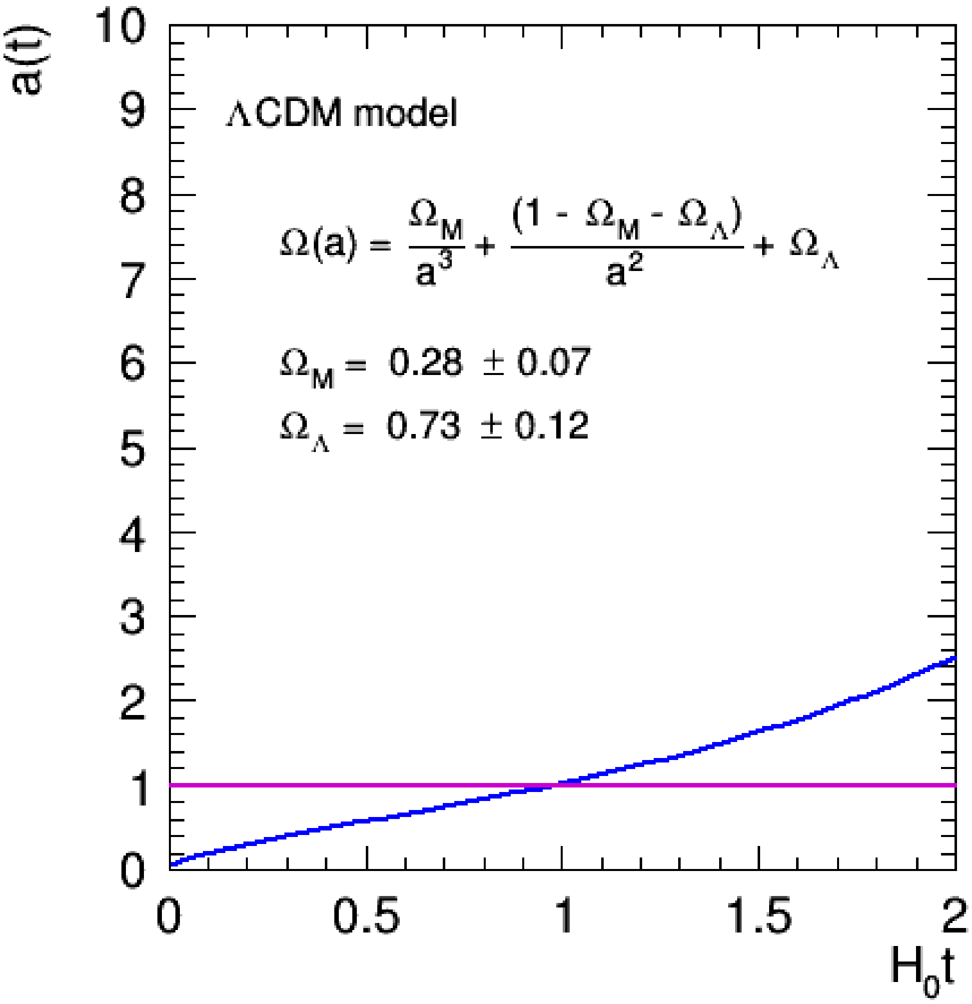



```
In [18]: def plotScaleFactor(modelname, code, results, amax=10, tmax=2):
             from array import array

             p = array('d')
             for value, error in results:
```

```
        p.append(value)

    # create a vs H0 t plot
    a = array('d'); a.fromlist(code.N*[0])
    t = array('d'); t.fromlist(code.N*[0])
    code.scaleFactor(amax, p, t, a)
    g = mkgraph(t, a,
                "H_{0}t}", "a(t)",
                0, tmax, color=ROOT.kBlue, lwidth=2)
    ROOT.SetOwnership(g, 0)
    g.SetName('scaleFactor')

    # create horizontal line at a = 1
    x = array('d'); x.append(0); x.append(tmax)
    y = array('d'); y.append(1); y.append(1)
    glineH = mkgraph(x, y, '', '', 0, tmax,
                     color=ROOT.kMagenta+1, lwidth=2)
    ROOT.SetOwnership(glineH, 0)
    glineH.SetName('line')

    c = ROOT.TCanvas("fig_%s_scaleFactor" % modelname,
                     "SN1a scalefactor",
                     500, 500)
    ROOT.SetOwnership(c, 0)
    c.cd()
    htmp = mkhist1('htmp', 'H_{0}t', 'a(t)',
                       50, 0, tmax, ymin=0, ymax=10)
    ROOT.SetOwnership(htmp, 0)
    htmp.Draw()
    g.Draw('csame')
    glineH.Draw('csame')

    offset = 0.05
    xpos = 0.25
    ypos = 0.87
    scribe = Scribe(xpos, ypos)
    ROOT.SetOwnership(scribe, 0)
    annotate(modelname, scribe, results, offset)
    c.Draw()
    c.SaveAs(".pdf")
```

```
In [19]: plotScaleFactor(MODEL, code, results)
```

```
Info in <TCanvas::Print>: pdf file ./fig_LCDM_scaleFactor.pdf has been created
```


a(t)
ΛCDM model
$\Omega(a) = \frac{\Omega_M}{a^3} + \frac{(1 - \Omega_M - \Omega_\Lambda)}{a^2} + \Omega_\Lambda$
$\Omega_M = 0.28 \pm 0.07$
$\Omega_\Lambda = 0.73 \pm 0.12$
$H_0 t$