Luminosity Distance - PHYS417 Project 2

Ryan Cox

2022-10-16

1 The Friedmann Equation

The Hubble parameter is given by

$$H(t)^2 = \frac{8\pi G}{3}\rho - \frac{k}{a^2} + \frac{\Lambda}{3}$$

We can define

$$\rho_c = \frac{3H(t)^2}{8\pi G}$$

$$\Omega = \frac{\rho}{\rho_c} = \frac{8\pi G}{3H(t)^2}\rho$$

$$\Omega_k = -\frac{k}{H(t)^2 a^2}$$

$$\Omega_\Lambda = \frac{\Lambda}{3H(t)^2}$$

as seen in Lecture 2. We see we can rewrite our first equation as

$$\frac{H^2}{H_0^2} = \frac{8\pi G}{3H_0^2}\rho - \frac{k}{H_0^2 a^2} + \frac{\Lambda}{H_0^2}$$

where we have dropped the t in our notation for $H(t)$ and used a subscript zero to indicate quantities at the present time If we assume

$$\rho = \frac{\rho_0}{a^3}$$

and use $a_0 = 1$ as holds for matter density, in a matter dominated universe or mixture, this transforms into

$$\frac{H^2}{H_0^2} = \frac{\Omega_0}{a^3} + \frac{\Omega_{k,0}}{a^2} + \Omega_{\Lambda,0}$$

Then it is just a matter of using

$$a = \frac{1}{1+z}$$

to find

$$\frac{H^2}{H_0^2} = \Omega_0(1+z)^3 + \Omega_{k,0}(1+z)^2 + \Omega_{\Lambda,0}$$

or

$$H = H_0\sqrt{\Omega_0(1+z)^3 + \Omega_{k,0}(1+z)^2 + \Omega_{\Lambda,0}}$$

1.1 Luminosity Distance Behaviour

Luminosity distance is given by

$$d_L = d_p(1+z)$$

where the physical (comoving) distance is given by

$$d_p = \frac{c}{H_0}\int_0^z \frac{dz'}{E(z')}$$

In a universe dominated by dark energy

$$\Omega_{\Lambda,0} = 1$$
$$E(z) = \sqrt{\Omega_{\Lambda,0}} = 1$$
$$d_p = \frac{cz}{H_0}$$

Then luminosity distance is simply

$$d_L = \frac{cz}{H_0}(1+z)$$

For large redshifts we get

$$d_L \propto z^2$$

In a universe dominated by matter

$$\Omega_0 = 1$$
$$E(z) = (1+z)^{3/2}$$
$$d_p = \frac{2c}{H_0}\left(1 - \frac{1}{\sqrt{1+z}}\right)$$

Then luminosity distance is

$$d_L = \frac{2c}{H_0}\left(1 + z - \sqrt{1+z}\right)$$

For large redshifts we get

$$d_L \propto z$$

1.2 Luminosity Distance Plots

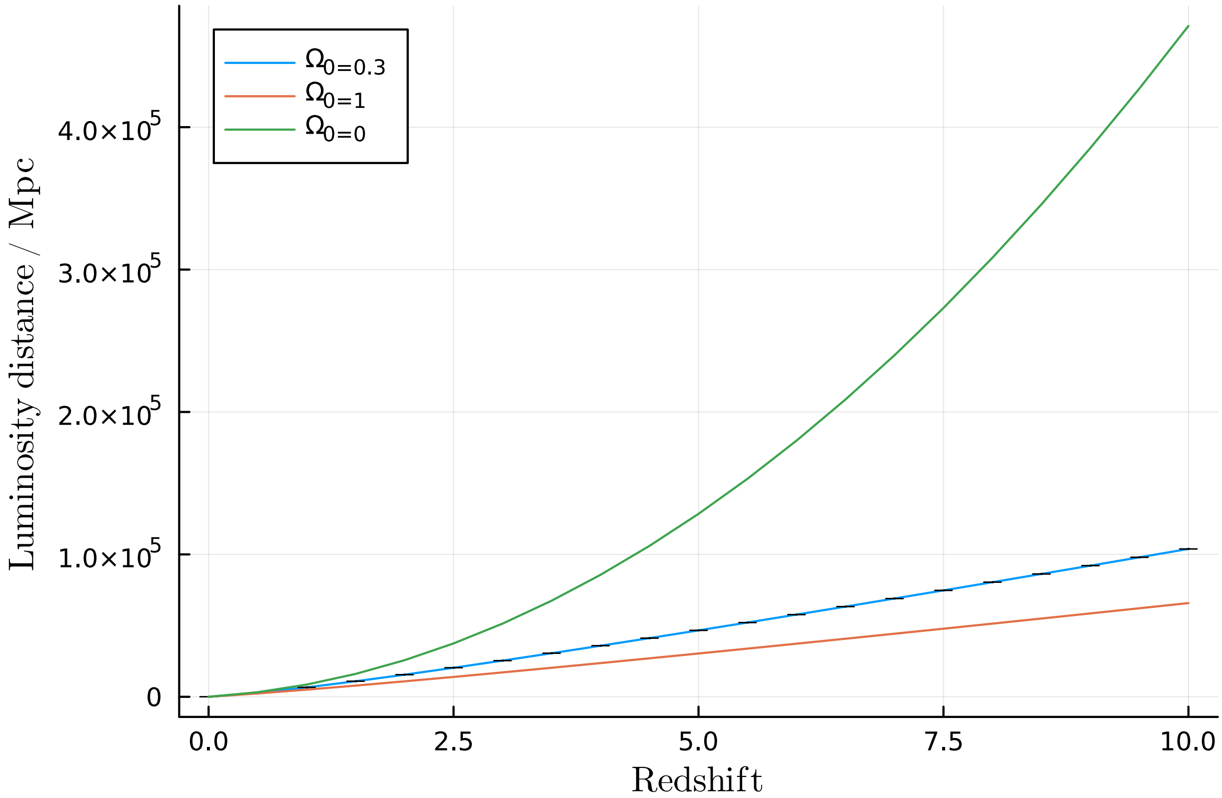
In a universe with both matter and dark energy we need to find d_p by numerical integration of

$$d_p(z) = \frac{c}{H_0} \int_0^z \frac{dz'}{E(z')}$$

From here on out we'll assume a flat universe, unless stated otherwise.

```
using Unitful # Unit handling
using UnitfulAstro # Astronomical units
using PhysicalConstants.CODATA2018: c_0 # Speed of light from CODATA2018, with units
using QuadGK # Numerical integration
using Plots, Latexify, UnitfulLatexify, LaTeXStrings
using Measurements # Uncertainly handling

# Unitful doesn't export preferunits so we have to reference by package
Unitful.preferunits(u"Mpc",u"Msun")

# Define our cosmological parameters
Ω0::Real = 0.3
Ωk::Real = 0 # flat universe
ΩΛ::Real = 1 - Ω0
H0 = 70.0u"km/s/Mpc"

# This is a one line function definition
E(z::Real)::Number = sqrt(Ω0*(1+z)^3 + Ωk*(1+z)^2 + ΩΛ)

# Input type must be real and the output must be a length
# Unitful will determine and check the dimensions of the output
function dp(z::Real)::Unitful.Length
    """Calculate proper distance from redshift."""
    integral, err = quadgk(zz -> 1/E(zz), 0, z, rtol=1e-8)
    return c_0/H0 .* (integral ± err)
end

dl(z::Real) = dp(z) * (1+z)

z = 0:0.5:10
# dl.(z) vectorises dl so it acts elementwise on z
plot(z,
    upreferred.(dl.(z)),
    unitformat=latexify,
    label="\\Omega_0=$(Ω0)",
    legend=:topleft,
    xlabel=L"\mathrm{Redshift}",
    ylabel="\\mathrm{Luminosity\\ distance}"
)

# Matter dominated -> no dark energy
Ω0::Real = 1
ΩΛ::Real = 1 - Ω0
plot!(z, 2 * c_0 / H0 * (1 .+ z - sqrt.(1 .+ z)), label="\\Omega_0=$(Ω0)") # plot!()
updates last plot
# Testing using the numerical solution agrees with the exact solution, but I've
# left it off the final plot since you can't see both lines at once.
# plot!(z, dl.(z), label="\\Omega_0=$(Ω0)")
```

```julia
# Dark energy dominated -> no matter
Ω0::Real = 0
ΩΛ::Real = 1 - Ω0
plot!(z, c_0 / H0 * z .* (1 .+ z), label="\\Omega_0=$(Ω0)")
# plot!(z, dl.(z), label="\\Omega_0=$(Ω0)")
```



2 Candles

We know that radial velocity is related to proper distance by

$$v_r = H_0 d_p$$

In the low redshift (and thus non-relativistic) limit

$$v_r \approx zc$$

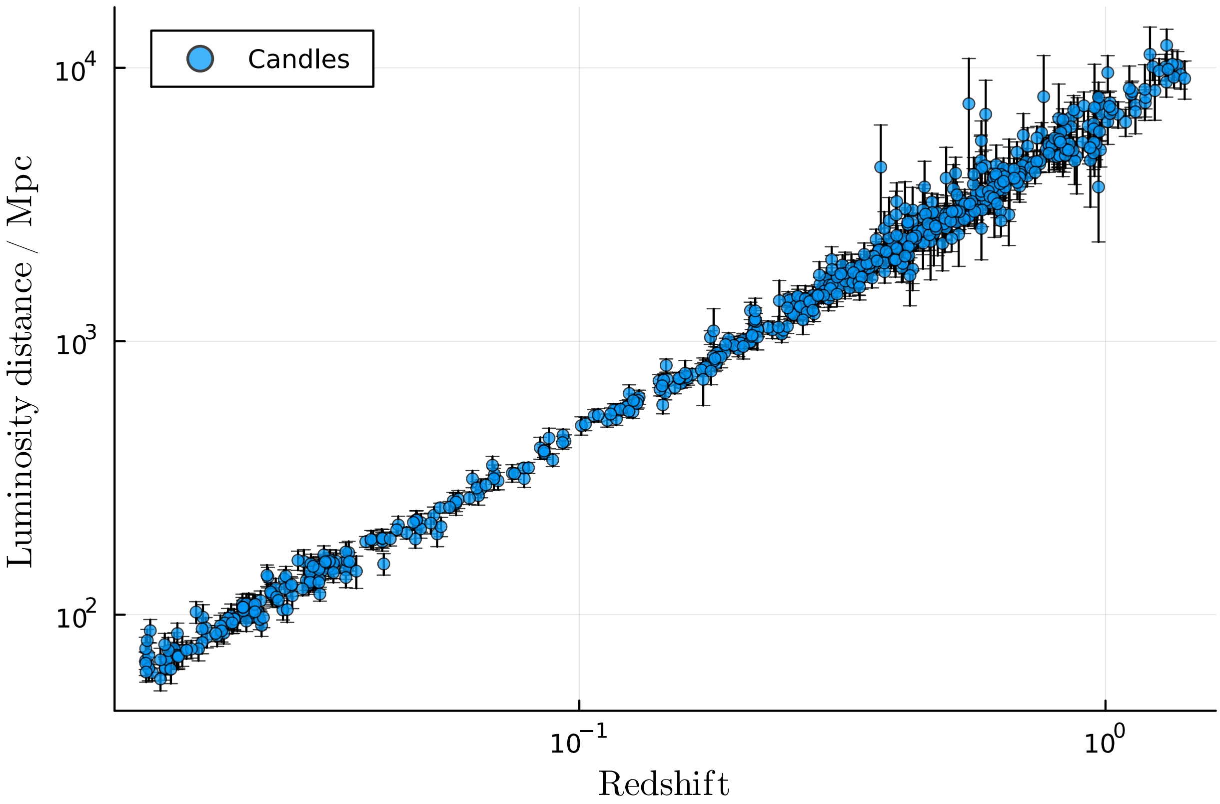
so we can conclude

$$d_p \approx \frac{zc}{H_0}$$

Combining this with our luminosity distance equation and gives

$$d_l \approx \frac{zc}{H_0}(1+z) \approx \frac{zc}{H_0} \text{as } z \to 0$$

```julia
using CSV, DataFrames

dl_from_m(dm::Number)::Unitful.Length = 10u"pc" * 10^(dm/5)

data = CSV.read("SCPUnion2.1_mu_vs_z.txt",
                DataFrame,
                header=["id", "z", "DMVal", "DMErr", "Prob"],
                skipto=6
        )

data.DM = data.DMVal .± data.DMErr

select!(data, :id, :z, :DM)  # Get rid of the other columns

data.dl = upreferred.(dl_from_m.(data.DM))

scatter(data.z,
        data.dl,
        unitformat=latexify,
        label="Candles",
        legend=:topleft,
        xlabel=L"\mathrm{Redshift}",
        ylabel="\\mathrm{Luminosity\\ distance}",
        markersize=3,
        markeralpha=0.75,
        xscale=:log10,
        yscale=:log10
)
```



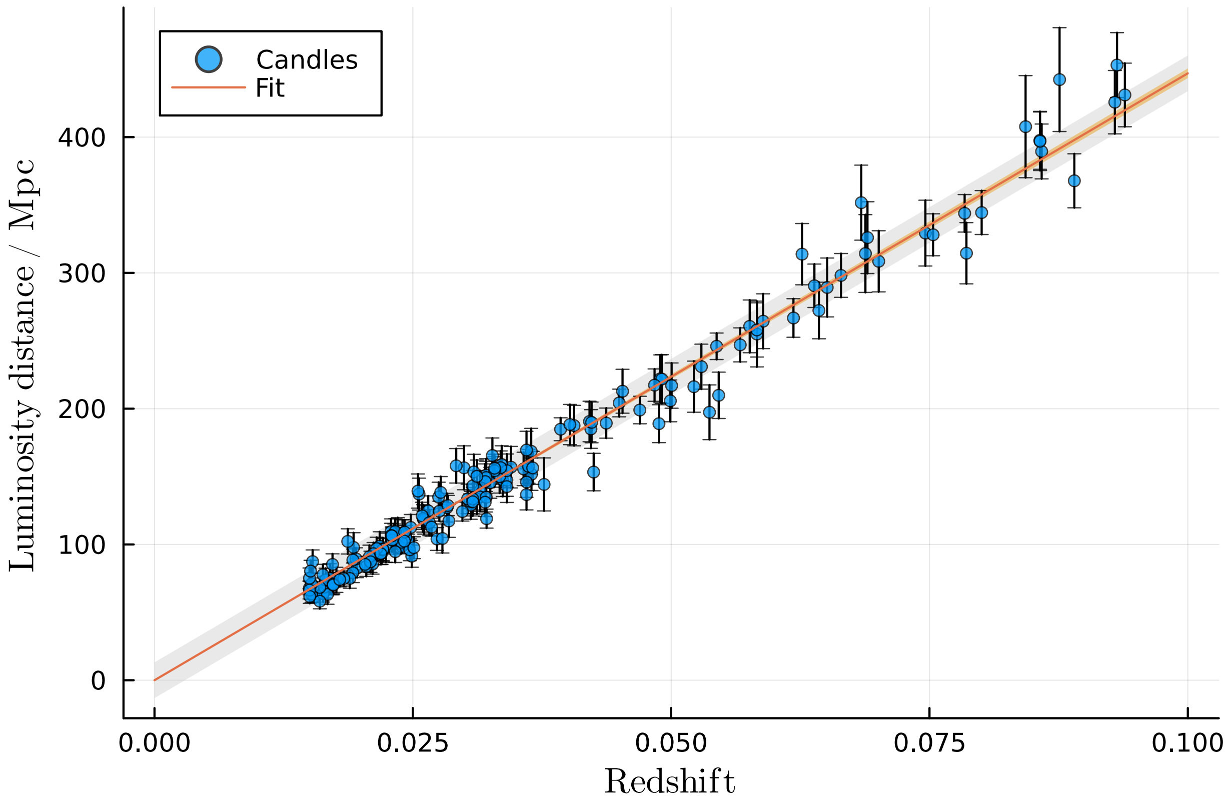
2.1 Low Redshift

We can perform basic least squares in Julia with remarkable ease. Remarkably it progagates the uncertaintities right through. However it doesn't like units so we'll have to strip them (carefully!). The line of best fit is underlaid by a light orange region showing variation from the propagated uncertainty in H_0 and a grey region showing the standard deviation of the data as estimated from the fitting. The first region is very narrow and thus hard to see.

```
using Statistics: mean

lowz_data = @view data[data.z .< 0.1, [:z, :dl]] # get z and DM for low redshift
kernel = lowz_data.z * ustrip(u"km/s", c_0) # construct kernel. dl = G*(1/H0)

reciprocal_H0 = kernel \ ustrip.(u"Mpc", lowz_data.dl) # This performs least squares

lowz_H0 = 1u"km/s/Mpc" / reciprocal_H0
println("Estimated H0 = $(lowz_H0)")

dl_mean = mean(lowz_data.dl)
# sum of squares error
fit_err = sum(abs2, Measurements.value.(lowz_data.dl) -
lowz_data.z*c_0/Measurements.value(lowz_H0))
# sum of variance
var = sum(abs2, Measurements.value.(lowz_data.dl .- Measurements.value.(dl_mean)))
# Coefficent of determination
println("R^2 = $(1 - fit_err/var)")
# Std of dl from fit error
lowz_err = sqrt(1/(length(lowz_data.z)-1) * fit_err)

# plot data
scatter(lowz_data.z,
        lowz_data.dl,
        unitformat=latexify,
        label="Candles",
        legend=:topleft,
        xlabel=L"\mathrm{Redshift}",
        ylabel="\\mathrm{Luminosity\\ distance}",
        markersize=3,
        markeralpha=0.75
)

z = 0:0.01:0.1
dlz = z*c_0/Measurements.value(lowz_H0)

# plot fitted line
plot!(z, dlz, linewidth=1, label="Fit")

# plot area around fitted line equal to ± standard deviation
plot!(
    z,
    dlz .+ lowz_err,
    fillrange = dlz .- lowz_err,
    fillcolor = :lightgray,
    fillalpha = 0.5,
    linecolor = nothing,
    primary = false, # no legend entry
    z_order = :back
)
```

```
# plot area around fitted line equal to minimum and maximum from uncertainities
plot!(
    z,
    z*c_0/(Measurements.value(lowz_H0)-Measurements.uncertainty(lowz_H0)),
    fillrange = z*c_0/(Measurements.value(lowz_H0)+Measurements.uncertainty(lowz_H0)),
    fillcolor = :orange,
    fillalpha = 0.75,
    linecolor = nothing,
    primary = false, # no legend entry
    z_order = :back
)
```

```
Estimated H0 = 67.07 ± 0.53 km Mpc^-1 s^-1
R^2 = 0.9791817338152677
```



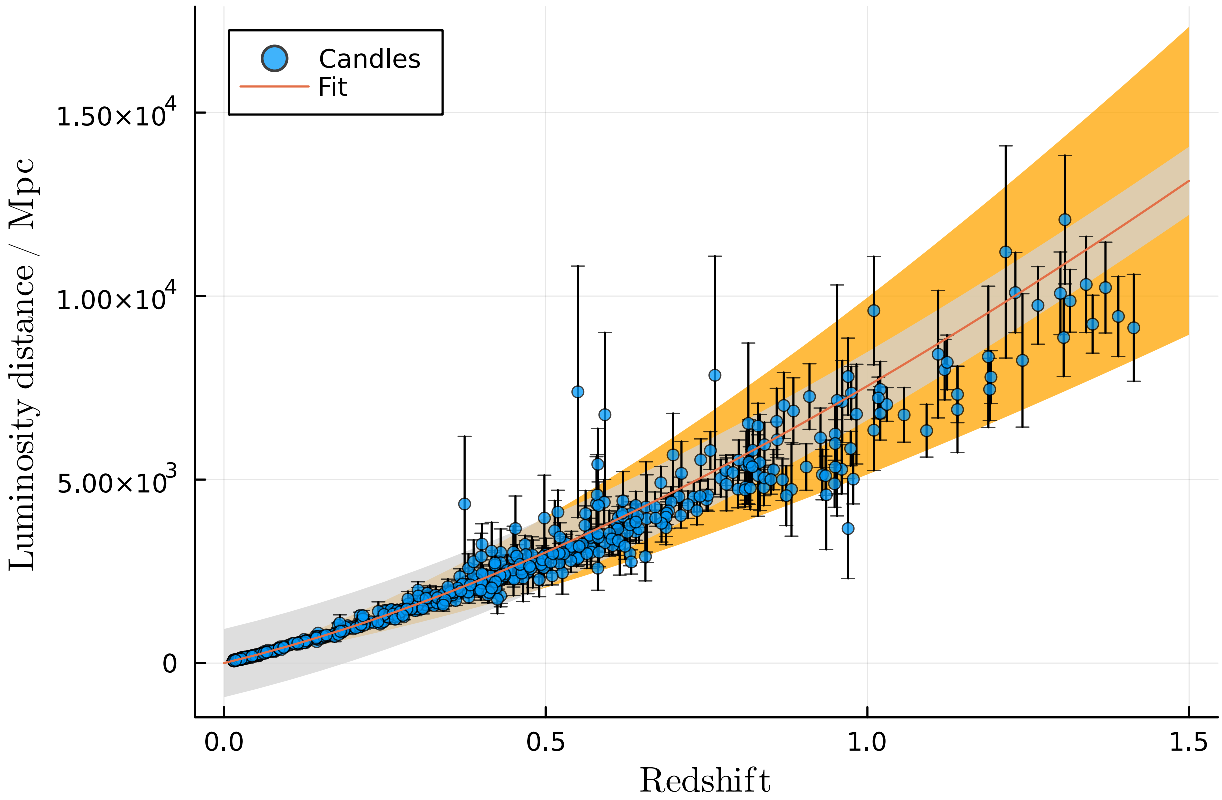
3 All Points

```
using LsqFit

# Implementing this curve fitting produced a succession of cryptic errors
# that I resolved by removing uncertainties and units, with some regret.

function dp(z, Ω, H)
    integral, err = quadgk(zz -> 1/sqrt(Ω*(1+z)^3 + (1-Ω)), 0, z, rtol=1e-8)
    return ustrip(u"km/s", c_0)/H * (integral ± err)
end

function dl_model(z, p)
    r =  dp.(z, p[1], p[2]) .* (1 .+ z)
    return Measurements.value.(r)
```

```julia
end

dl_in = ustrip.(u"Mpc", Measurements.value.(data.dl))
weights = 1 ./ ustrip.(u"Mpc", Measurements.uncertainty.(data.dl)).^2

fit = curve_fit(dl_model, data.z, dl_in, weights, [0.3, 70])

fit_err = stderror(fit) .* fit.param
# sum of squares error
fit_err2 = Measurements.value(sum(abs2, data.dl - dl_model(data.z, [Ω0,
ustrip(H0)])*1u"Mpc"))
var = Measurements.value.(sum(abs2, data.dl .- mean(data.dl)))
println("R^2 = $(1 - fit_err2/var)") # Coeffecent of determination
dl_fit_err = sqrt(1/(length(data.z)-1) * fit_err2) # Std of dl from fit error

params = fit.param .± fit_err
println("Ω0 = $(params[1])")
println("H0 = $(params[2])")

function E_integral(z::Real)::Number
    integral, err = quadgk(zz -> 1/E(zz), 0, z, rtol=1e-8)
    err = Measurements.value(Measurements.uncertainty(integral) +
Measurements.value(err) + Measurements.uncertainty(err))
    return Measurements.value(integral) ± err
end

z = 0:0.05:1.5

Ω0 = fit.param[1] ± fit_err[1]
ΩΛ = 1 - Ω0
H0 = (fit.param[2] ± fit_err[2])*1u"km/s/Mpc"

dlz = c_0 / H0 * E_integral.(z) .* (1 .+ z)

scatter(data.z,
        data.dl,
        unitformat=latexify,
        label="Candles",
        legend=:topleft,
        xlabel=L"\mathrm{Redshift}",
        ylabel="\\mathrm{Luminosity\\ distance}",
        markersize=3,
        markeralpha=0.75
)

plot!(z, Measurements.value.(dlz), linewidth=1, label="Fit")

plot!(
    z,
    Measurements.value.(dlz) .+ dl_fit_err,
    fillrange = Measurements.value.(dlz) .- dl_fit_err,
    fillcolor = :lightgray,
    fillalpha = 0.75,
    linecolor = nothing,
    primary = false, # no legend entry
    z_order = :back
)

plot!(
```

```
    z,
    Measurements.value.(dlz) + Measurements.uncertainty.(dlz),
    fillrange = Measurements.value.(dlz) - Measurements.uncertainty.(dlz),
    fillcolor = :orange,
    fillalpha = 0.75,
    linecolor = nothing,
    primary = false, # no legend entry
    z_order = :back
)
```

```
R^2 = 0.8391220608855329
Ω0 = 0.0993 ± 0.00061
H0 = 71.0 ± 23.0
```



This result is unconvincing.