Luminosity Distance - PHYS417 Project 2

Ryan Cox

2022-10-14

1 The Friedmann Equation

Show it can be written Predict behaviour (take limits)

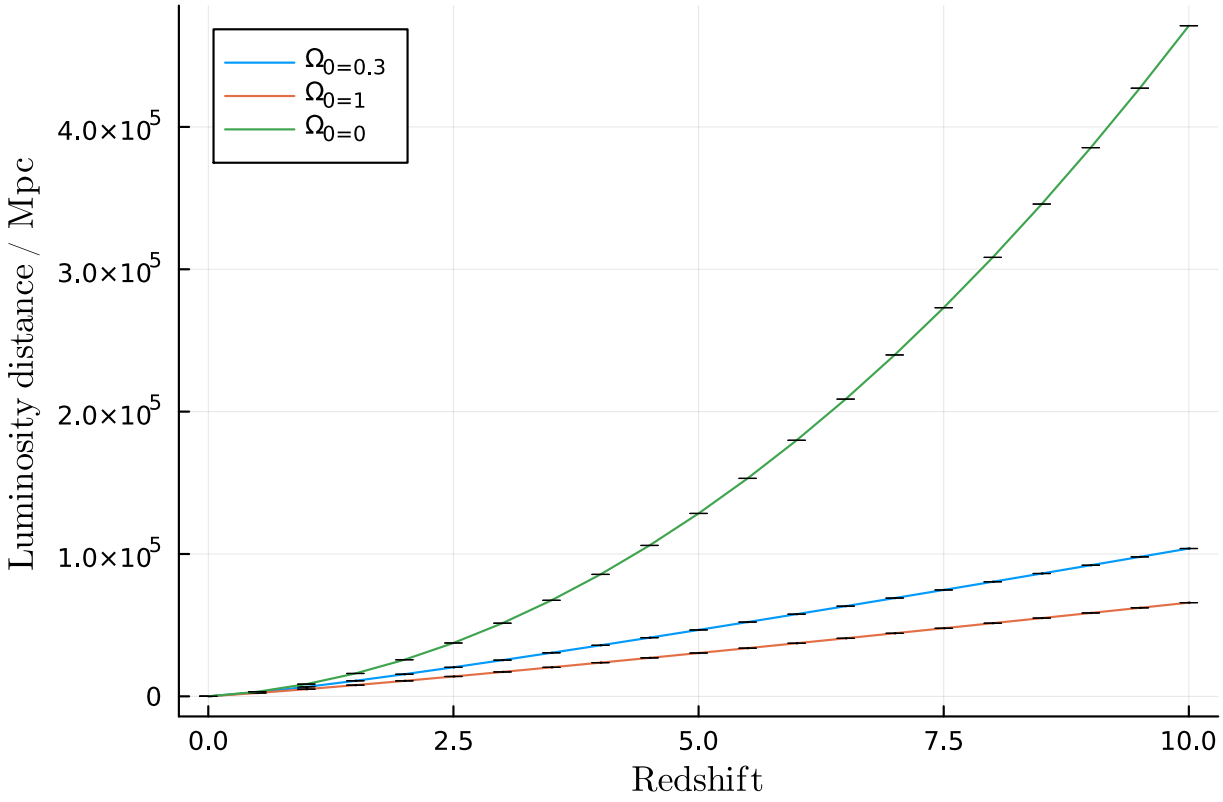
In a universe with both matter and dark energy we need to find d_p by numerical integration of

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

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, dl.(z), label="\\Omega_0=$(Ω0)") # plot!() updates last plot

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



Candle time.

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

$$v_r \approx zc$$

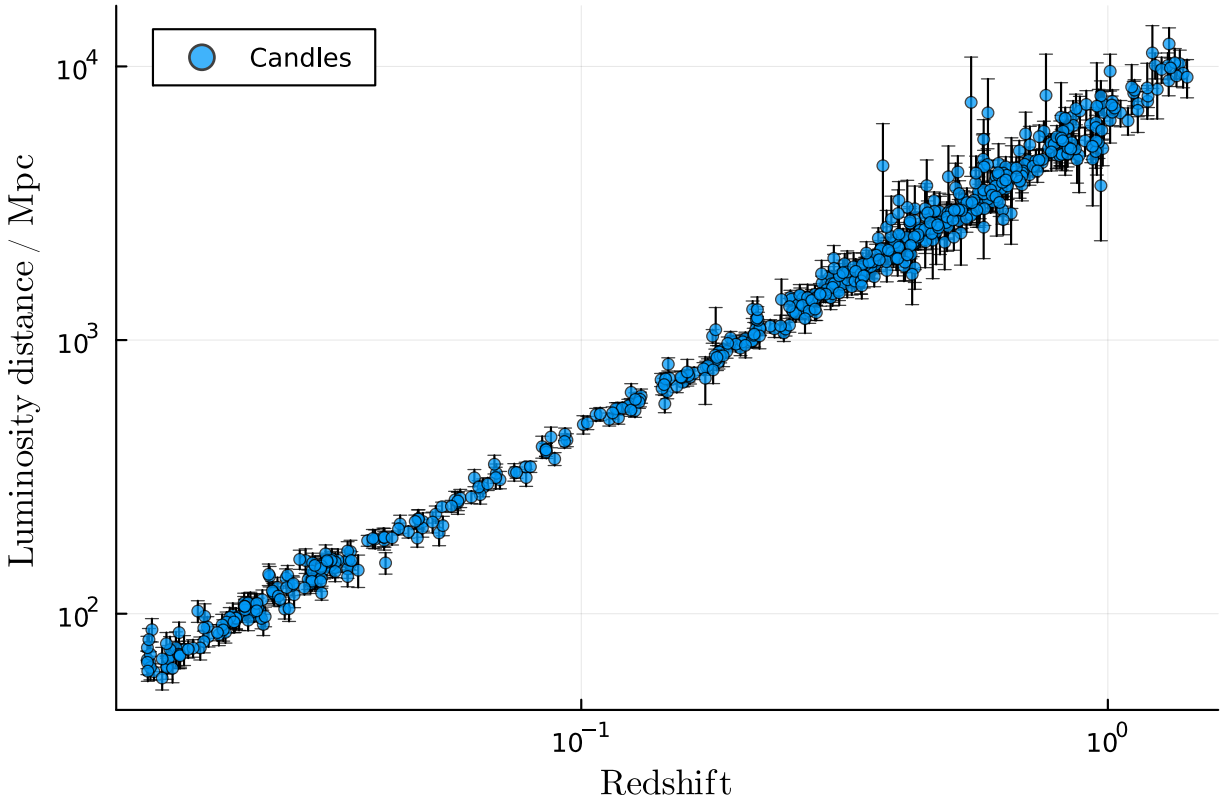
so we can conclude

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

Combining this with our luminosity distance equation and gives

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

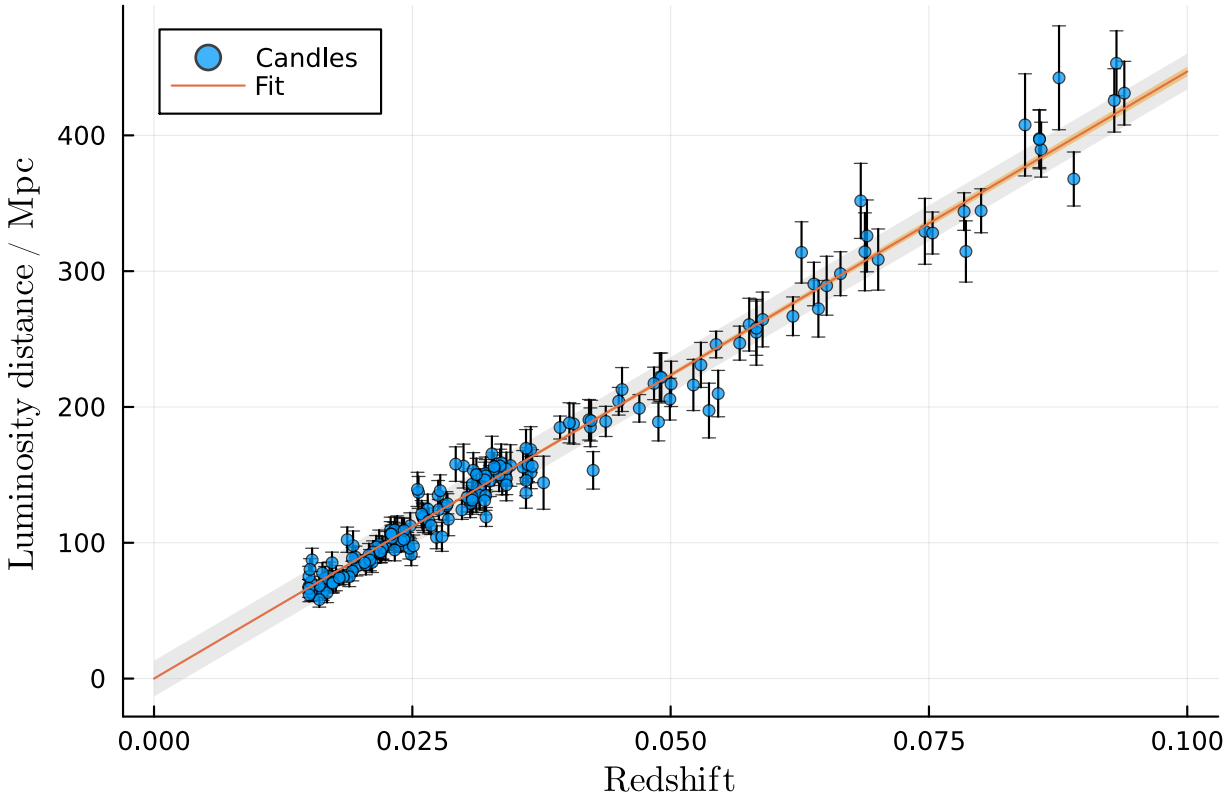
```
using CSV, DataFrames

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
)
```



We can perform basic least squares in Julia with remarkable ease. Remarkably it handles uncertaintities. 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 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

function H0LeastSquares(kernel, z_data, dl_data)
    reciprocal_H0 = kernel \ ustrip.(u"Mpc", dl_data) # This performs least squares
    est_H0 = 1u"km/s/Mpc" / reciprocal_H0

    println("Estimated H0 = $(est_H0)")

    # sum of squares error
    fit_err = sum(abs2, Measurements.value.(dl_data) -
z_data*c_0/Measurements.value(est_H0))
    dl_mean = mean(dl_data)
    # sum of variance
    var = sum(abs2, Measurements.value.(dl_data .- Measurements.value.(dl_mean)))
    println("R^2 = $(1 - fit_err/var)") # Coefficent of determination
    dl_fit_err = sqrt(1/(length(z_data)-1) * fit_err) # Std of dl from fit error

    return est_H0, dl_fit_err
end

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

lowz_H0, lowz_err = H0LeastSquares(G, lowz_data.z, lowz_data.dl)

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!(z, dlz, linewidth=1, label="Fit")

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!(
    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
```

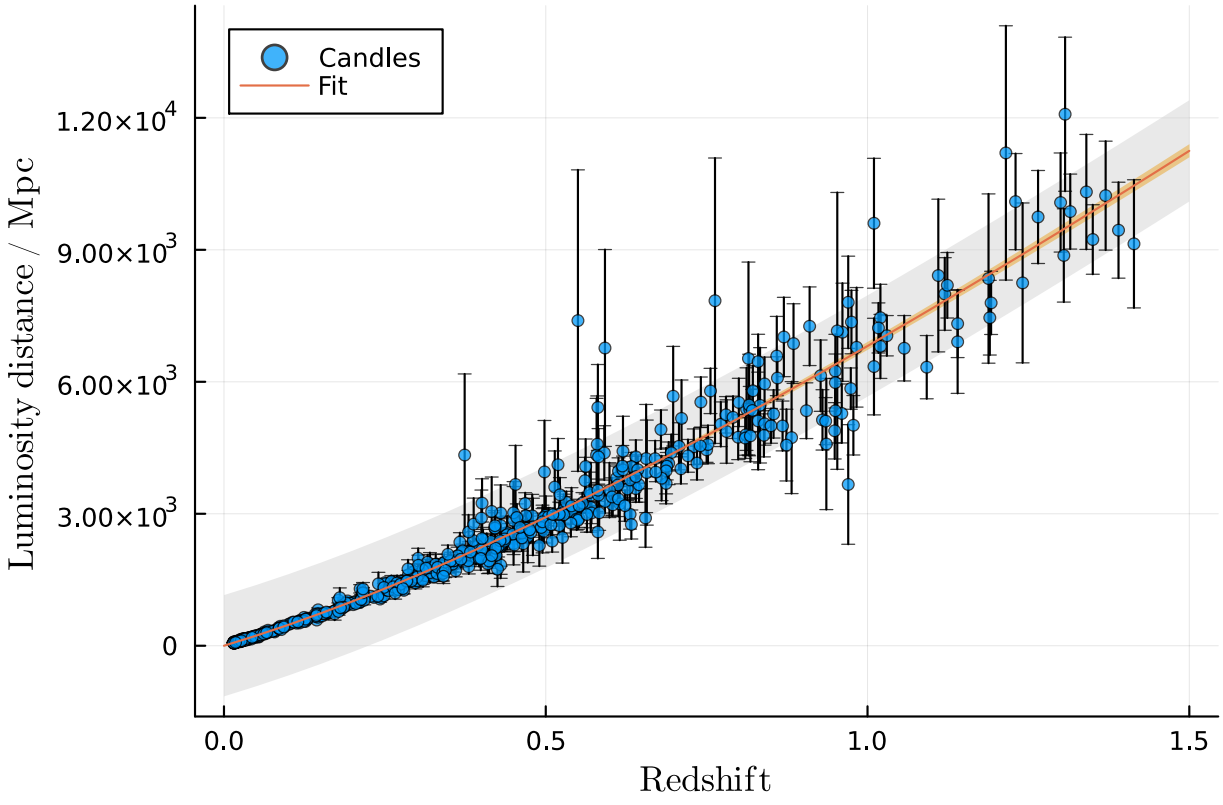


Now we'll fit the full data, using the full equation for luminosity distance and assuming $\Omega_0 = 0.3$.

```
Ω0::Real = 0.3
ΩΛ::Real = 1 - Ω0

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

G = ustrip(u"km/s", c_0) * E_integral.(data.z) .* (1 .+ data.z)

allz_H0, allz_err = H0LeastSquares(G, data.z, data.dl)

# I should probably put all this repeated code into a function
# But I'm not going to

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

z = 0:0.05:1.5
dlz1 = c_0  * Measurements.value.(E_integral.(z) .* (1 .+ z))
dlz = dlz1 / Measurements.value(allz_H0)
```

```
plot!(z, dlz, linewidth=1, label="Fit")

plot!(
    z,
    dlz .+ allz_err,
    fillrange = dlz .- allz_err,
    fillcolor = :lightgray,
    fillalpha = 0.5,
    linecolor = nothing,
    primary = false, # no legend entry
    z_order = :back
)

plot!(
    z,
    dlz1/(Measurements.value(allz_H0)-Measurements.uncertainty(allz_H0)),
    fillrange = dlz1/(Measurements.value(allz_H0)+Measurements.uncertainty(allz_H0)),
    fillcolor = :orange,
    fillalpha = 0.75,
    linecolor = nothing,
    primary = false, # no legend entry
    z_order = :back
)
```

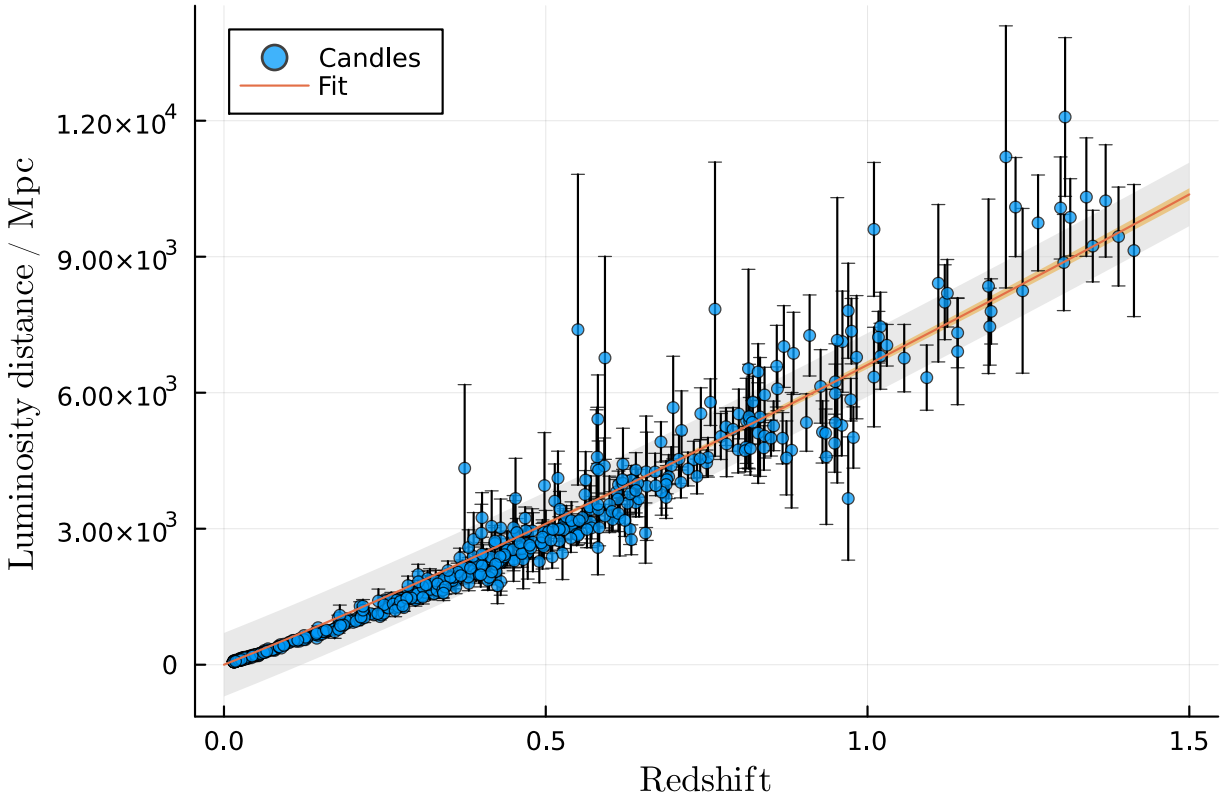
```
Estimated H0 = 67.89 ± 0.9 km Mpc^-1 s^-1
R^2 = 0.7533095159272138
```



Lets try a universe without dark energy.

```
Ω0::Real = 1
ΩΛ::Real = 1 - Ω0

G = ustrip(u"km/s", c_0) * E_integral.(data.z) .* (1 .+ data.z)
```

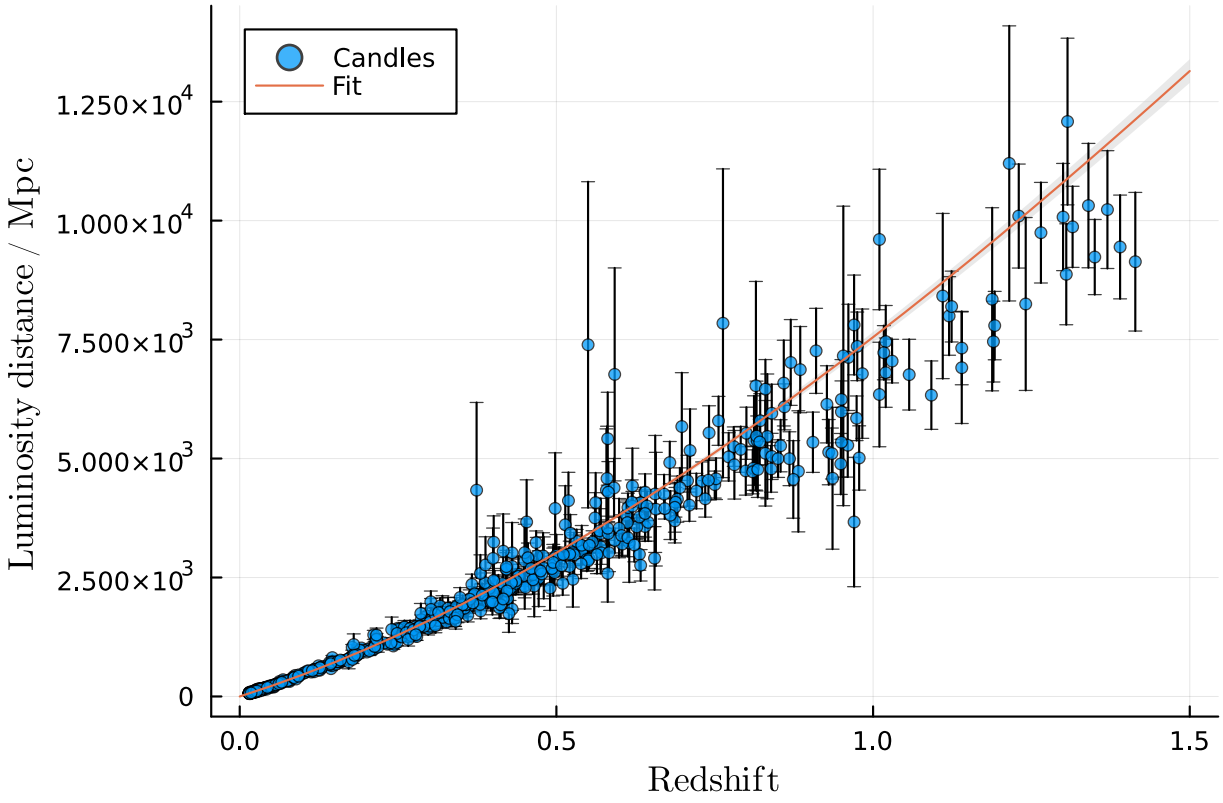
```
allz_H0, allz_err = H0LeastSquares(G, data.z, data.dl)

# I should probably put all this repeated code into a function
# But I'm not going to

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

z = 0:0.05:1.5
dlz1 = c_0  * Measurements.value.(E_integral.(z) .* (1 .+ z))
dlz = dlz1 / Measurements.value(allz_H0)

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

plot!(
    z,
    dlz .+ allz_err,
    fillrange = dlz .- allz_err,
    fillcolor = :lightgray,
    fillalpha = 0.5,
    linecolor = nothing,
    primary = false, # no legend entry
    z_order = :back
)

plot!(
    z,
    dlz1/(Measurements.value(allz_H0)-Measurements.uncertainty(allz_H0)),
    fillrange = dlz1/(Measurements.value(allz_H0)+Measurements.uncertainty(allz_H0)),
    fillcolor = :orange,
    fillalpha = 0.75,
    linecolor = nothing,
    primary = false, # no legend entry
    z_order = :back
)

Estimated H0 = 53.1 ± 0.68 km Mpc^-1 s^-1
R^2 = 0.908673608461059
```



Compare -> R^2 value, appearance of fit, match with other sources

```
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)
    s = Measurements.value.(r)
    return s
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 = margin_error(fit)
fit_std_err = stderror(fit)

display(fit.param)
display(fit_err)
display(fit_std_err)

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

dlz1 =  c_0  * E_integral.(z) .* (1 .+ z)
dlz = dlz1 / H0

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) + Measurements.uncertainty.(dlz),
    fillrange = Measurements.value.(dlz) - Measurements.uncertainty.(dlz),
    fillcolor = :lightgray,
    fillalpha = 0.5,
    linecolor = nothing,
    primary = false, # no legend entry
    z_order = :back
)

2-element Vector{Float64}:
  0.09930322376791202
 71.01740922100265
2-element Vector{Float64}:
 0.012018801020293067
 0.6269724699228766
2-element Vector{Float64}:
 0.006119313409312699
 0.31921994847830576
```


Candles
Fit
Luminosity distance / Mpc
1.250×10^4
1.000×10^4
7.500×10^3
5.000×10^3
2.500×10^3
0
0.0
0.5
1.0
1.5
Redshift