DSNB Decay Notes 7/05

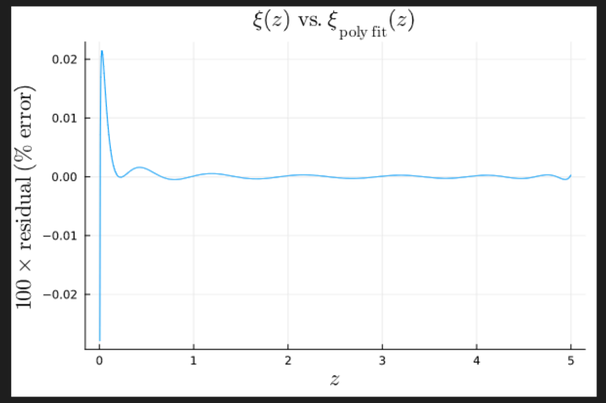
Miller MacDonald

First optimization attack: the $\xi(z)$ effective length function

Fit with a 14th order polynomial, gets us down from 8 allocations per leff call to 0

```
plot(
    xseval,
    (leff.(xseval) .- lefftest.(xseval)) ./ leff.(xseval)*100,
    xlabel=L"z",
    ylabel=L"100\times\mathrm{residual}~(\mathrm{\%~error})",
    xguidefontsize=15,
    yguidefontsize=15,
    label=false,
    title=L"\xi(z)~\mathrm{vs.}\xi_\mathrm{poly~fit}(z)",
    titlefontsize=15
)
```

[329] ✓ 0.4s




```
@time decay(1, 10^(-26), 0, 0.01)
```

[296] ✓ 0.2s

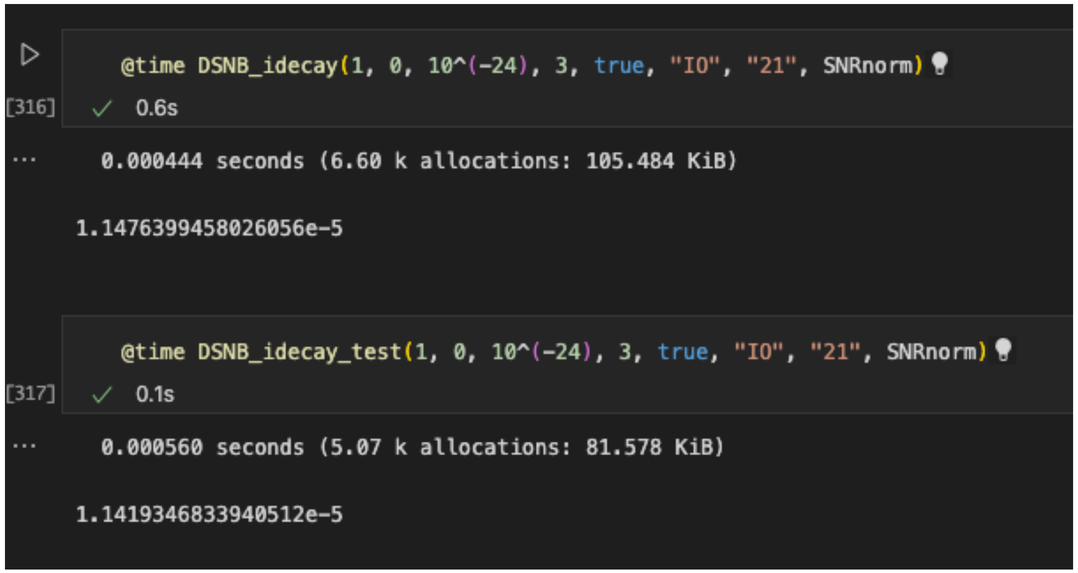
```
  0.000022 seconds (8 allocations: 128 bytes)

0.9360877758914534
```

```
@time decaytest(1, 10^(-26), 0, 0.01)
```

[297] ✓ 0.1s

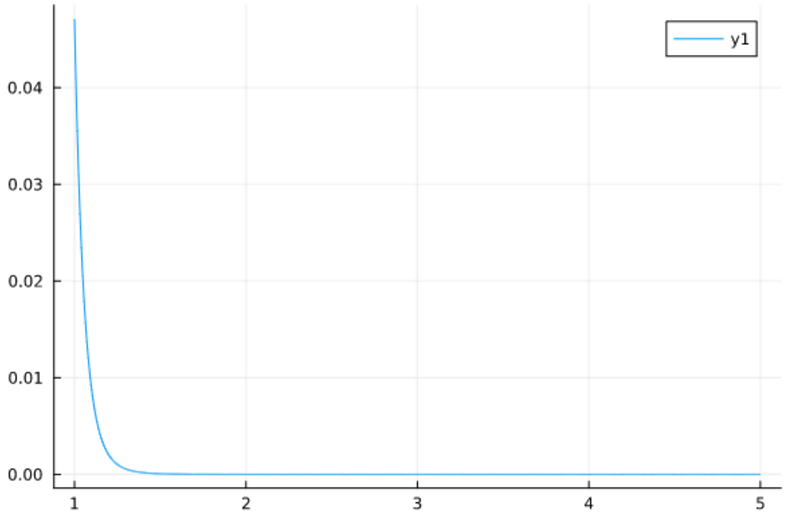
```
  0.000001 seconds

0.9360918489105652
```

At most 0.02% error



A pretty significant reduction in the number of allocations, but not a time reduction…

Another idea: apparently QuadGK.jl likes when we split the integral bounds such that large spikes in the integrand are isolated. For short lifetimes, we get integrands that are very spiked at low z



We want to find a suitable $z_{\text{cutoff}}(E, \alpha, z_0)$ to split the integral evaluation

Let's solve the problem $e^{\beta\alpha(\xi(z_0)-\xi(z))/E} = 0.01$ ($\beta = 4.68 \times 10^{28}$ is the conversion scaling factor to make the units work out

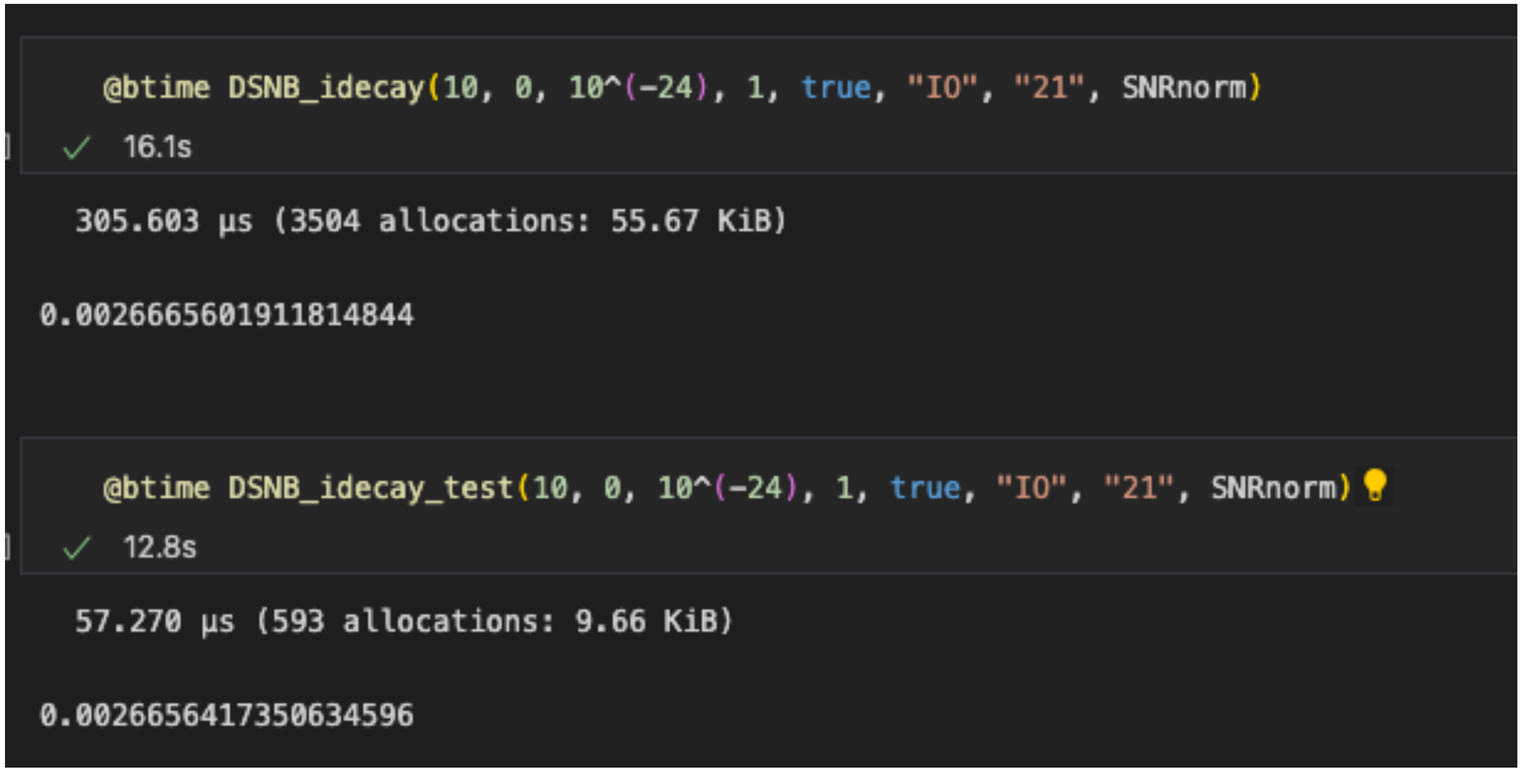
We get that $\xi(z) = -\frac{E}{\beta\alpha}\ln\left(0.01e^{-\beta\alpha\xi(z_0)/E}\right)$

To get the rough shape of the effective length function, let's approximate it as

$$\xi(z) \approx -\frac{0.2}{70}e^{-1.8x+0.9} + \frac{0.5}{70}$$

We end up with the solution

$$z_{\text{cutoff}} \approx -0.555\left(\ln\left(350\left(0.007 + \frac{E}{\beta\alpha}\ln\left(0.01e^{-\beta\alpha\xi(z_0)/E}\right)\right)\right) - 0.9\right)$$



Ohohohoho we've got some major time saved!

This only saves time for some situations but it's definitely nice to have, and the time it saves is precisely in the scenarios where the previous integral was having trouble because of the spiky integrand

Can we do the same thing with the integrals over energy in the q_{ji}'s?

We have that the integrand goes like $\text{integrand}(E) \sim \phi_j^{\text{decay}}(E, \alpha' s)\frac{1}{E}\psi_{j\to i}(E, E_{\text{rs}})$,

where $E_{\text{rs}} = E_0\frac{1+z}{1+z_0}$ is the redshifted energy and

$$\psi_{j\to i}(E_h, E_l) = \begin{cases} 2\frac{E_l}{E_h^2} & \text{h.c.} \\ \frac{2}{E_h}\left(1 - \frac{E_l}{E_h}\right) & \text{h.f.} \end{cases}$$

For h.c., the integrand's behavior in energy is definitely dominated by the $\frac{1}{E}\psi \sim \frac{1}{E^3}$ component

So we can ask a similar question to last time: what is the energy E_{cutoff} such that

$$\frac{\text{integrand}(E_{\text{cutoff}})}{\text{integrand}(E_{\text{rs}})} \approx 0.01?$$

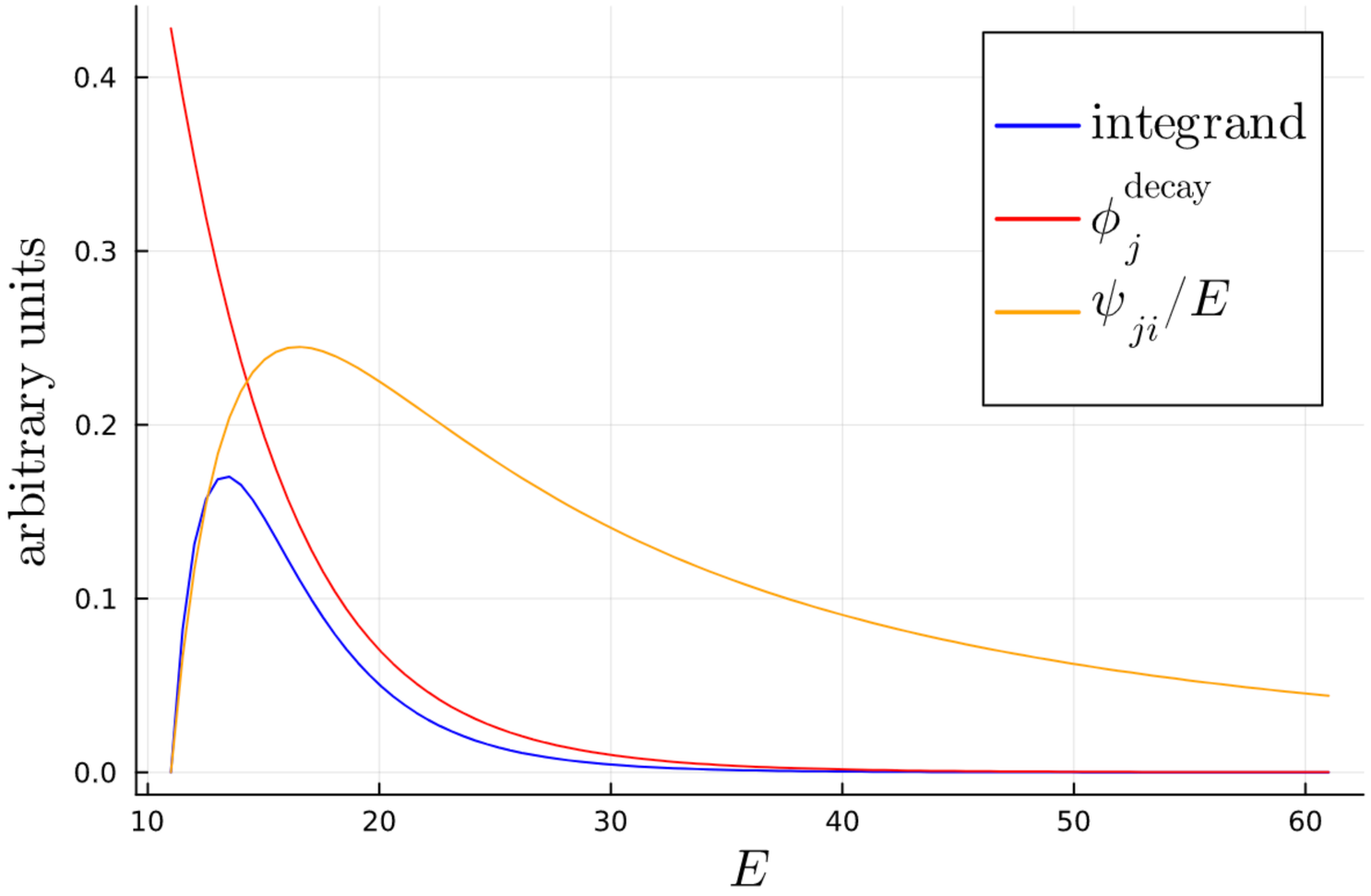
This reduces for h.c. to $\frac{2E_{\text{rs}}/E^3}{2/E_{\text{rs}}^2} \approx 0.01 \quad \Rightarrow \quad E_{\text{cutoff}} \approx 100^{1/3} E_{\text{rs}}$

```
@time q21contrib_NO_test(1, 0.1, 0.2, 10^(-26), 10^(-26), true, true, "21", SNRnorm)
✓ 1.5s

  1.371834 seconds (22.69 M allocations: 357.309 MiB, 10.14% gc time)

1.3940086051981697
```

```
@time q21contrib_NO(1, 0.1, 0.2, 10^(-26), 10^(-26), true, true, "21", SNRnorm)
✓ 8.4s

  7.625838 seconds (100.48 M allocations: 1.541 GiB, 10.06% gc time)

1.3940128163430077
```

This more or less works! Again, not for all cases, but it specifically targets the cases that are the slowest due to QuadGK having a hard time evaluating the spiky integrand

For h.f., harder because the attenuation is caused not by the psi_{ji}/E but by the decayed heavier mass state flux ϕ_j… haven't thought about if there's an easy way to approximate this



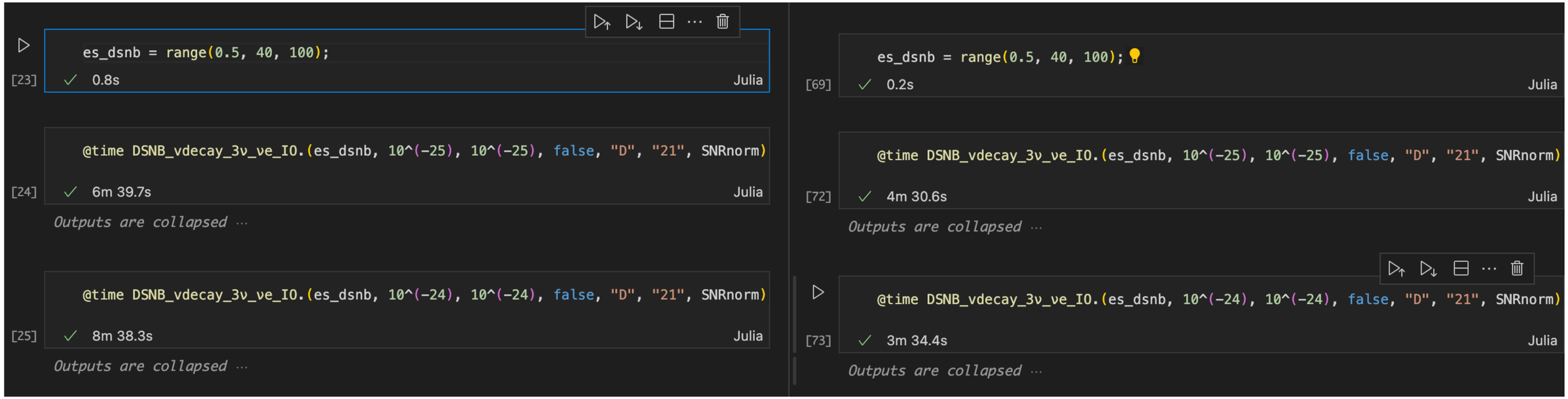
```
[83]  @time q21contrib_NO(1, 0.1, 0.2, 10^(-24), 10^(-24), true, true, "21", SNRnor
      ✓ 24.3s                                                                  Julia
...   23.498237 seconds (444.97 M allocations: 6.776 GiB, 14.25% gc time)

      0.36047986830227596
```

```
[206] @time q21contrib_NO_test(1, 0.1, 0.2, 10^(-24), 10^(-24), true, true, "21", SNRnorm)
      ✓ 2.7s                                                                  Juli
...   2.621049 seconds (37.72 M allocations: 594.166 MiB, 12.46% gc time)

      0.3602381119512785
```

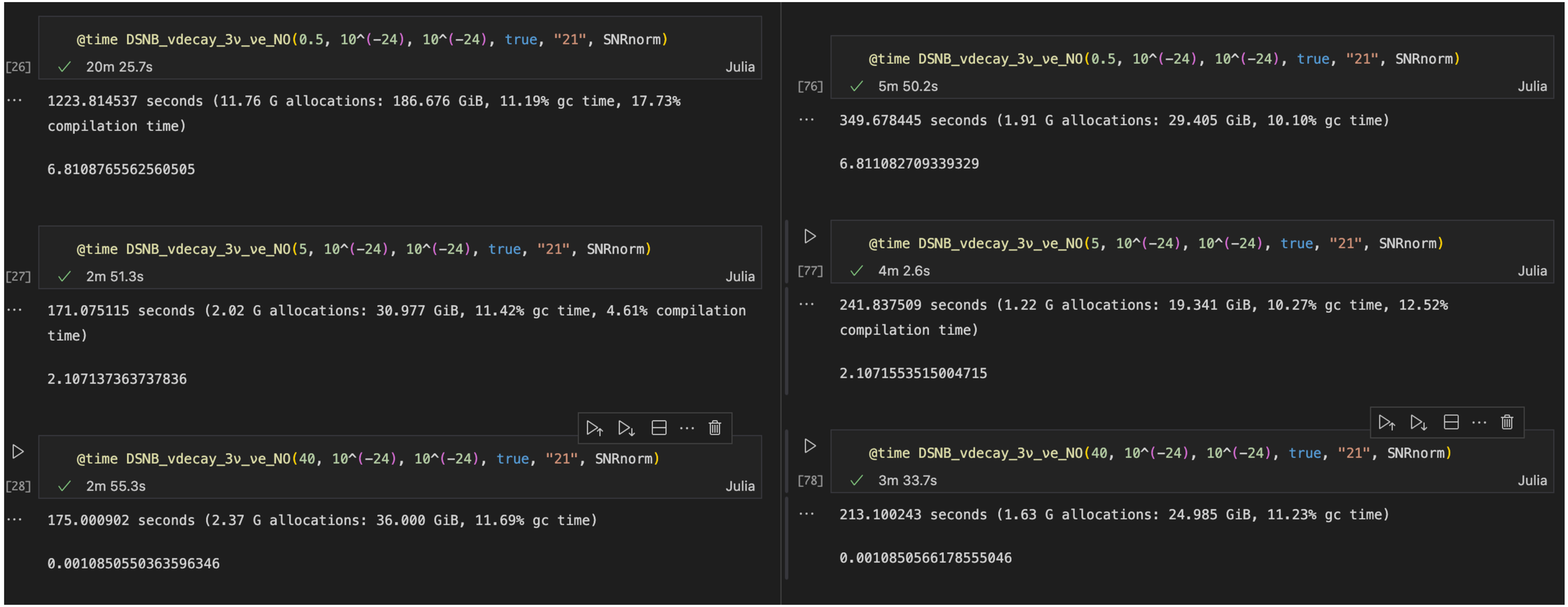
We've made a lot of progress though!

```
[84]  @time DSNB_vdecay_1_NO(1, 10^(-24), 10^(-24), true, "21", SNRnorm)
      ✓ 6m 41.1s                                                              Julia
...   400.822085 seconds (7.25 G allocations: 110.824 GiB, 8.47% gc time, 2.54%
      compilation time)

      8.44135630458028
```

```
[218] @time DSNB_vdecay_1_NO(1, 10^(-24), 10^(-24), true, "21", SNRnorm)
      ✓ 1m 49.7s
...   109.326251 seconds (1.52 G allocations: 23.340 GiB, 10.54% gc time)

      8.44148731401271
```



For the IO case, time definitely saved and I think we can start running these



For the NO case, things are a bit more mixed… saving allocations across the board, but for some reason in certain cases even when I'm allocating almost half the memory, it's still taking a lot longer… not sure why this is