Phys 512 : PS2

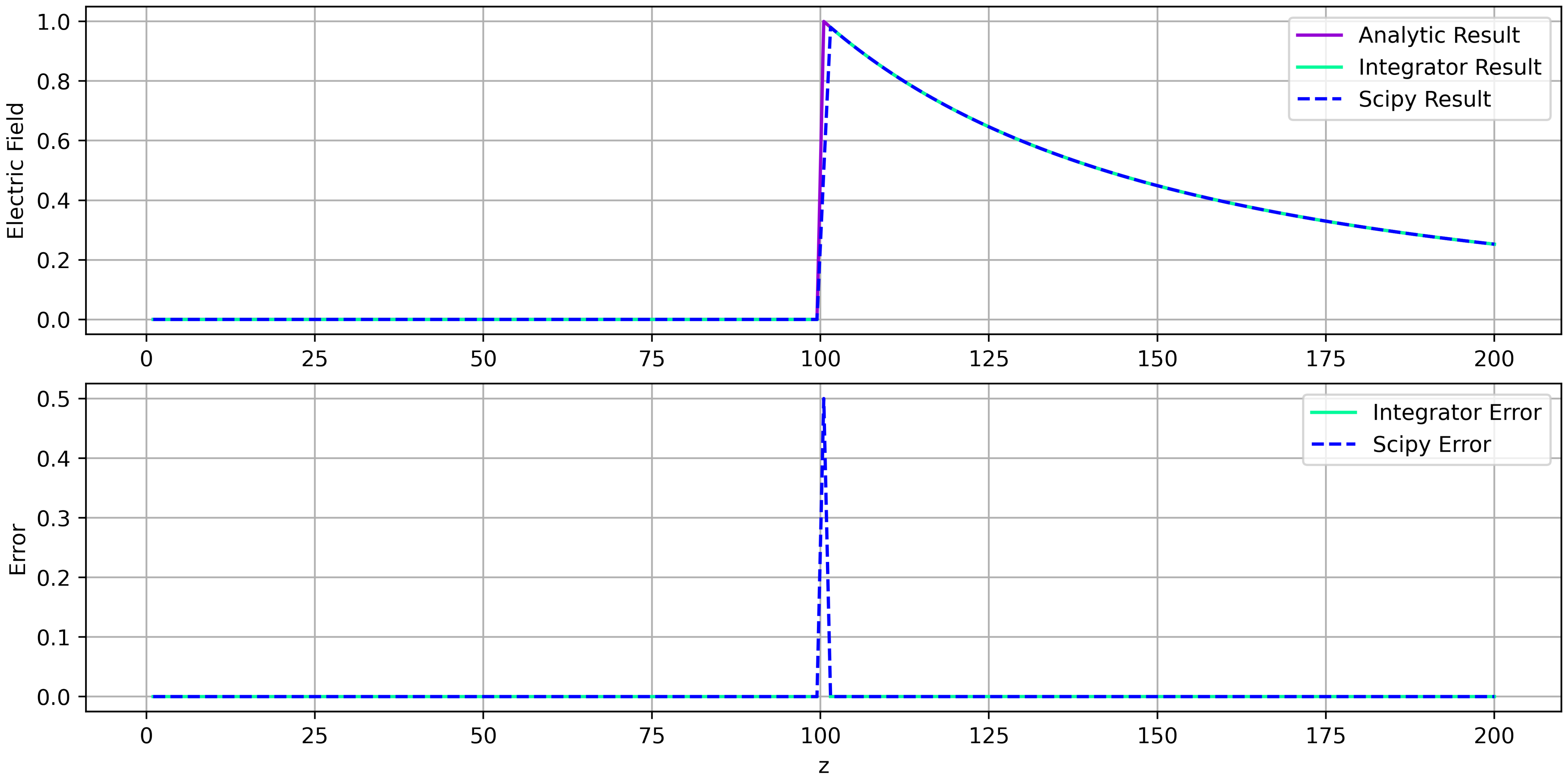
1) Quoting the solutions to Griffiths problem 2.7, the integral we want to compute is:

$$E(z) = \frac{1}{4\pi\varepsilon_0}\int d\theta\, d\phi \, \frac{\sigma R^2 \sin\theta\,(z - R\cos\theta)}{(R^2+z^2-2Rz\cos\theta)^{3/2}} \quad , \quad \int d\phi = 2\pi$$

$$= \frac{2\cancel{\pi}\sigma R^2}{2\cancel{4\pi}\varepsilon_0}\int_0^{\pi} d\theta \, \frac{\sin\theta\,(z-R\cos\theta)}{(R^2+z^2-2Rz\cos\theta)^{3/2}} = \frac{\sigma R^2}{2\varepsilon_0}\int_0^{\pi} d\theta\,\frac{\sin\theta\,(z-R\cos\theta)}{(R^2+z^2-2Rz\cos\theta)^{3/2}}$$

<u>Note</u>: This evaluates to: $E(z) = \frac{\sigma R^2}{2\varepsilon_0 z^2}\left[\frac{z-R}{|z-R|} + \frac{z+R}{|z+R|}\right]$ } <u>poles</u> at $z = \pm R$

We show the results of the $E(z)$ integral using a made integrator & scipy.integrate.quad; the analytic result & the error of the integrator & scipy (note I take $R=100$):



Electric Field
1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
200
Analytic Result
Integrator Result
Scipy Result
Error
0.5
0.4
0.3
0.2
0.1
0.0
Integrator Error
Scipy Error
z

It should be noted we work in units of $\sigma = \varepsilon_0 = 1$ as they are simply prefactors that only scale the result & don't change the behaviour of $E(z)$.

analytic - integral

↳ As we can see, the error is of order $O(10^{-1})$ for both the integrator & scipy

Scipy doesn't care about the singularity at $z = R$ $(=100)$. My integrator however did not like the division by zero at $E(z=100) \approx 0/0$, so to get around this I fixed $E(z=100) \approx 1$.

2) The way the integrator function I wrote works is: there is a call counter set to zero (integrator Calls = 0) & it counts w/in the function (by setting the calls as a global variable).

↳ Then, in the first iteration extra = None & the call updates by +5 as we select 5 points to integrate our

↳ In the next iteration when extra ≠ None, the function value & the extra values are extracted & put into an array

↳ Then we update the calls by +2. Once this is done we proceed with the usual Simpson rule integration

↳ Finally, if the integral error is larger than the set tolerance, we feed the yValue (y(x)) back into the integrator to compute it once more

We run the lazy integrator from class & my integrator & for the following functions we get the calls:

$f(x) = e^{-x^2/10}$: Lazy calls amount = 215, Integrator call amount = 89
$\Rightarrow$ Saved calls amount = 126

$f(x) = \sin(x/10 - 5)$: Lazy amount = 75, Integrator amount = 33
$\Rightarrow$ Saved amount = 42

$f(x) = x/(x^2-1000)$: Lazy amount = 15, Integrator amount = 9
$\Rightarrow$ Saved amount = 6

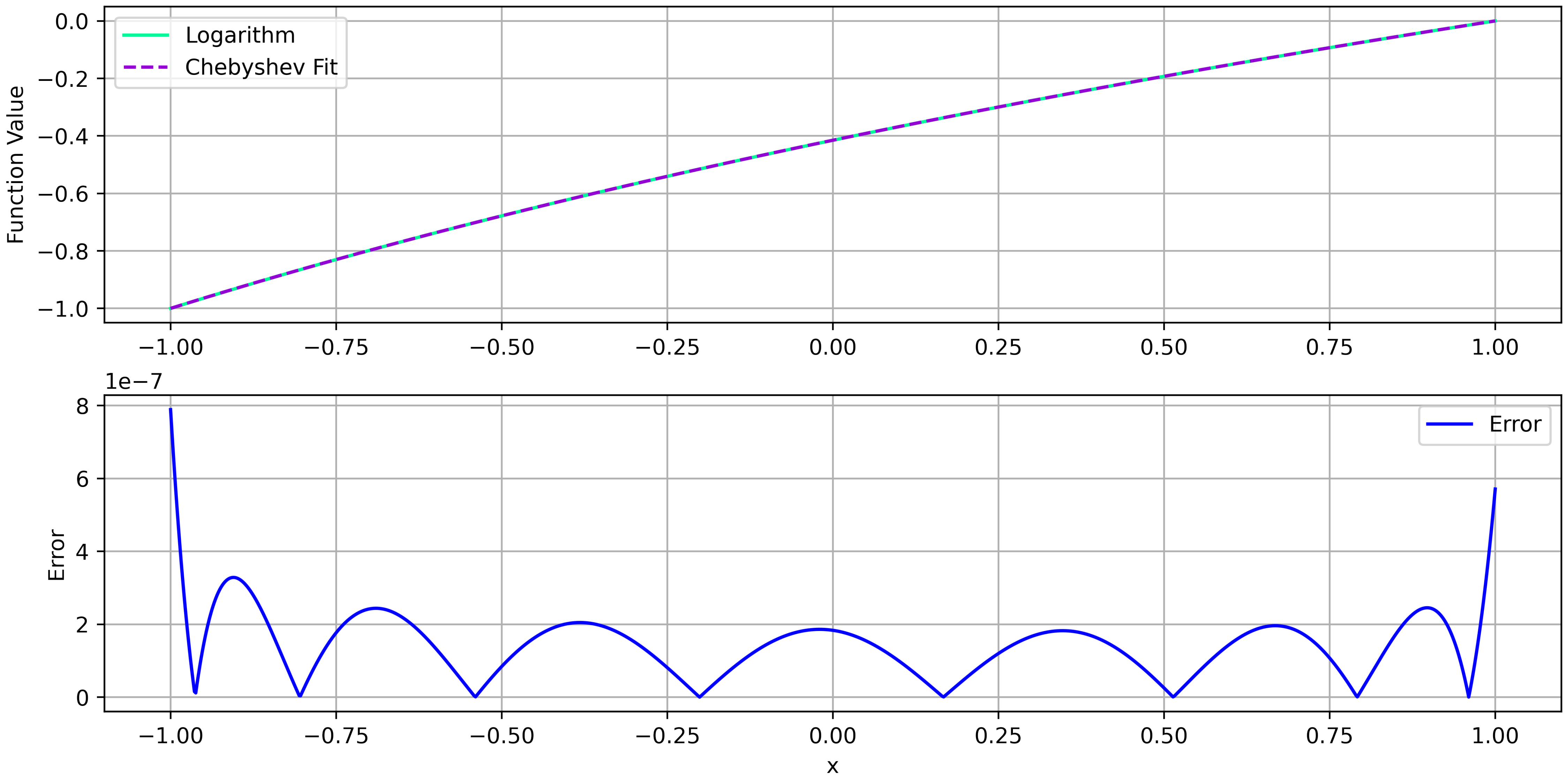
3) a) We want $f(x) = \log_2 x$ for $x \in [\frac{1}{2}, 1]$ w/ 10^{-6} accuracy or better.

↳ since we are using Chebyshev polynomials which are defined over $[-1,1]^2$, we perform the translation transformation:

$$x \longmapsto 4x - 3 \ : \ [\tfrac{1}{2}, 1] \longrightarrow [-1, 1]$$

$$\Rightarrow \log_2 x \longmapsto \log_2(4x - 3)$$

We plot $\log_2 x$, the chebyshev fit, & the error difference:



Logarithm
Chebyshev Fit
Function Value
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1e−7
8
6
4
2
0
Error
Error
x

As we can see, the error is of order $O(10^{-7})$ & thus fufills the requirements.

→ In this case I only chose 7 terms as past that the change in error is very small.

b) Now we want to split a number $x \in \mathbb{R}$ into its mantissa m & its exponent n, of the form:

$$x \longmapsto m \cdot 2^n$$

here it is a 2 being that we are considering $\log_2(x)$

Once we take $\log_2(x)$ under the decomposition, we get:

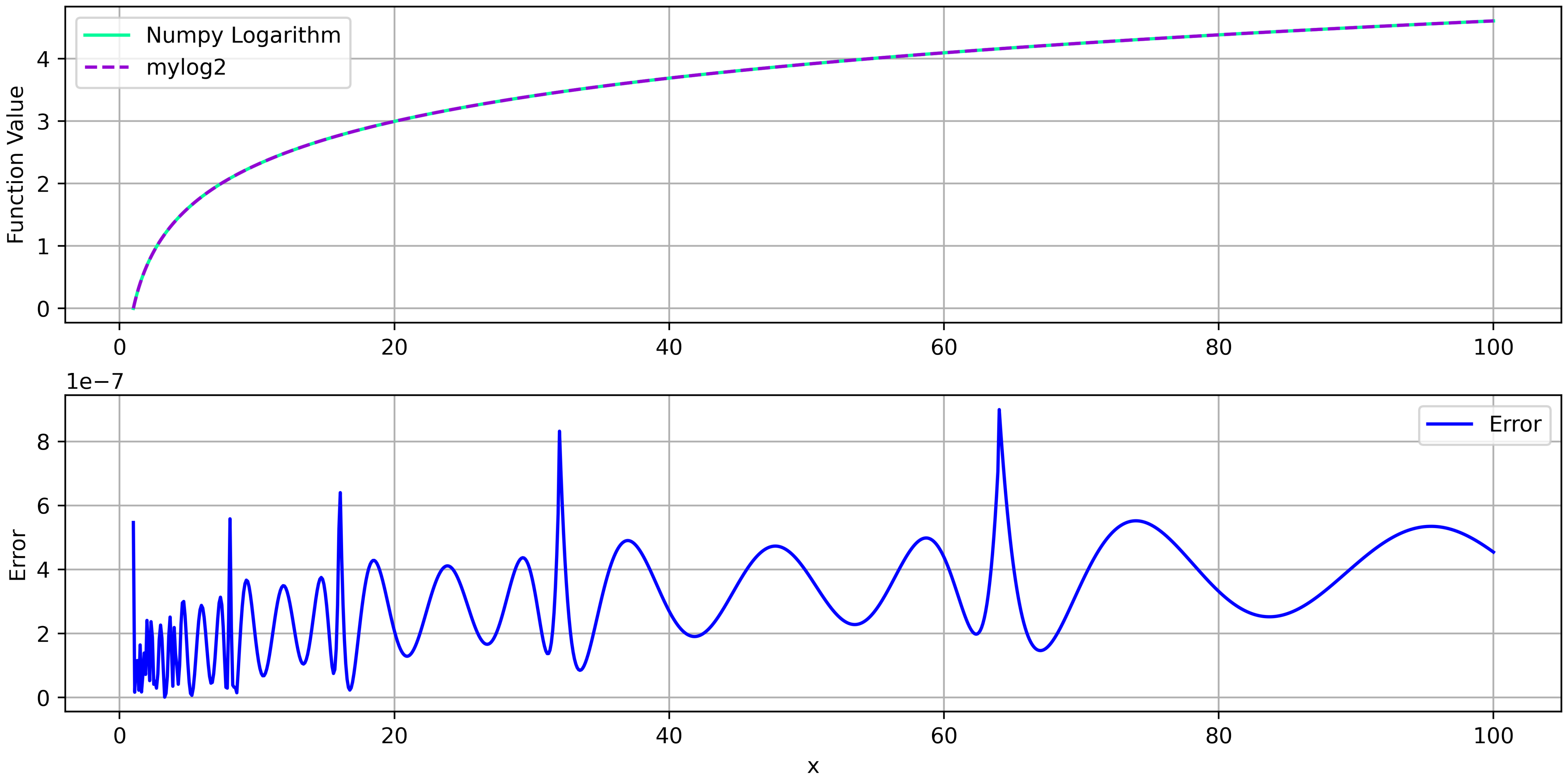
$$\log_2 x = \log_2(m 2^n) = \log_2 m + \log_2 2^n$$
$$= \log_2 m + n$$

Here $m \in [\frac{1}{2}, 1]$, so it suffices to use our previously defined Chebyshev fit.

Finally, to get the natural logarithm $\log_e x = \log x$, we divide by $\log_2 e$:

$$\log x = \log_2 x / \log_2 e$$

To see how well we did, we plot our natural logarithm, the numpy logarithm, & the difference:

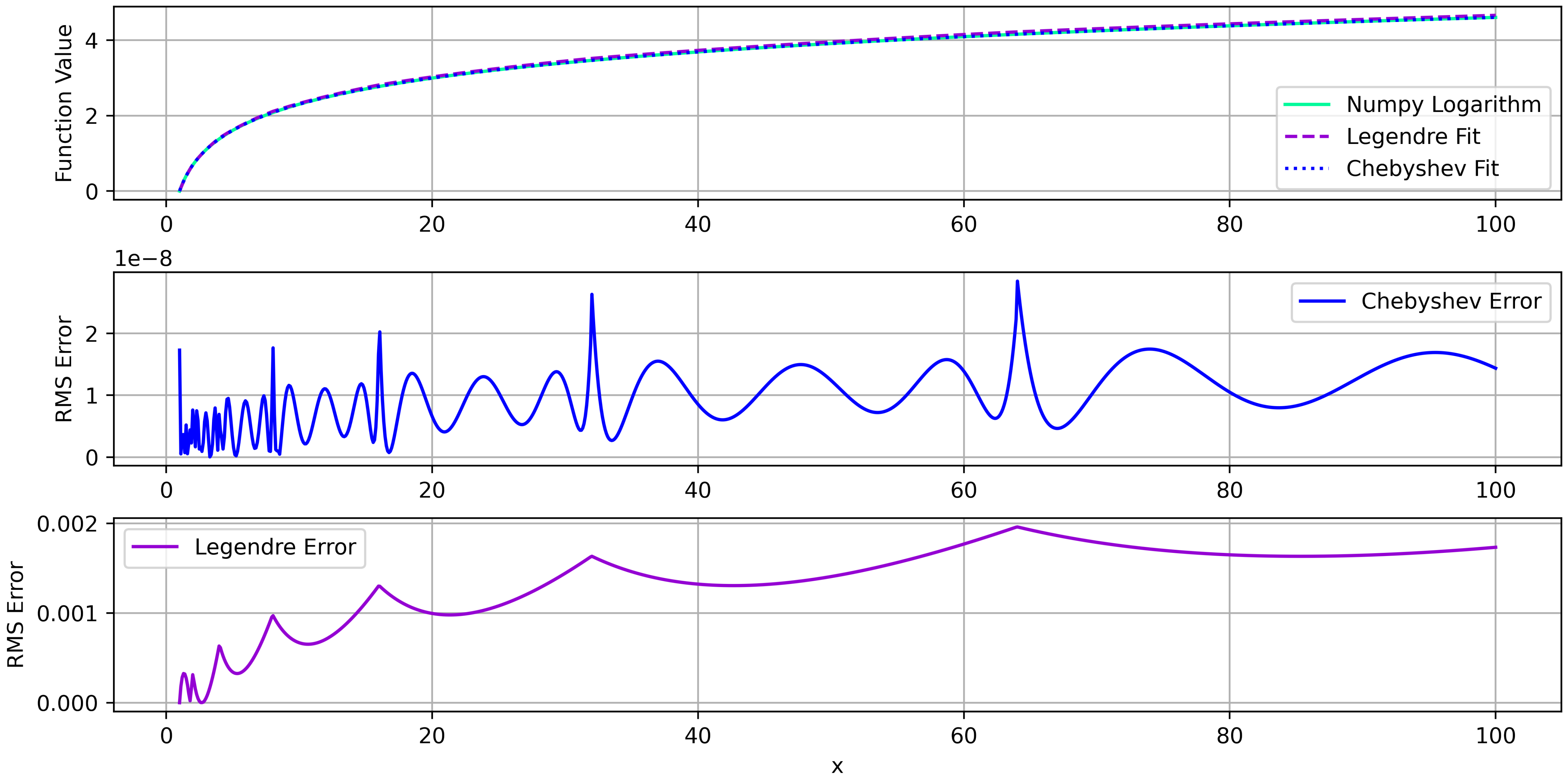


Numpy Logarithm
mylog2
Function Value
0
1
2
3
4
0
20
40
60
80
100
1e−7
Error
Error
0
2
4
6
8
0
20
40
60
80
100
x

As we can see, the difference is of order $O(10^{-7})$ which is decent.

bonus) Now we repeat the same exercise as done in a) & b) except w/ Legendre polynomials (for

↳ We plot the numpy logarithm, chebyshev/Legendre fits, and their RMS:



Function Value
Numpy Logarithm
Legendre Fit
Chebyshev Fit
1e−8
RMS Error
Chebyshev Error
RMS Error
Legendre Error
x

As we can see, the RMS error is much larger in the Legendre fit compared to the Chebyshev fit!

↳ The Chebyshev error is of order $O(10^{-8})$ while the Legendre error is of order $O(10^{-3})$, so Chebyshev is 5 order of magnitudes more precise than Legendre

The max Legendre RMS error is 0.0019596 while the max Chebyshev RMS error is 2.8443339×10^{-8}