

PHY604: homework 1

2025-09-22

1 *understanding round-off error (no program required)*

Consider a quadratic equation of the form $ax^2 + bx + c = 0$. The two solutions of this are:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

- 1(a)** Explain how this expression may be problematic with respect to roundoff errors if b is much larger than a and c . Recall that such errors often occur when subtracting close large numbers.

If $b^2 \gg 4ac$, then $\sqrt{b^2 - 4ac} \approx b$, in which case the $+$ solution will end up with something asymptotic to $-b + b$ in the numerator, which is prone to roundoff error. \square

- 1(b)** Provide an alternative expression that will have smaller errors in the situation you describe in (a).

The $-$ solution is not a risk, so we ignore it for now. For the $+$ solution, we multiply by one:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \left(\frac{b + \sqrt{b^2 - 4ac}}{b + \sqrt{b^2 - 4ac}} \right) = \frac{4ac}{2a(b + \sqrt{b^2 - 4ac})}.$$

2 *round-off error and accurate calculation of the exponential series*

Consider the series expansion for an exponential function:

$$e^x \approx S_n(x) := 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!}.$$

- 2(a)** Write a program that computes the exponential function using this series expansion for a given number of terms n .

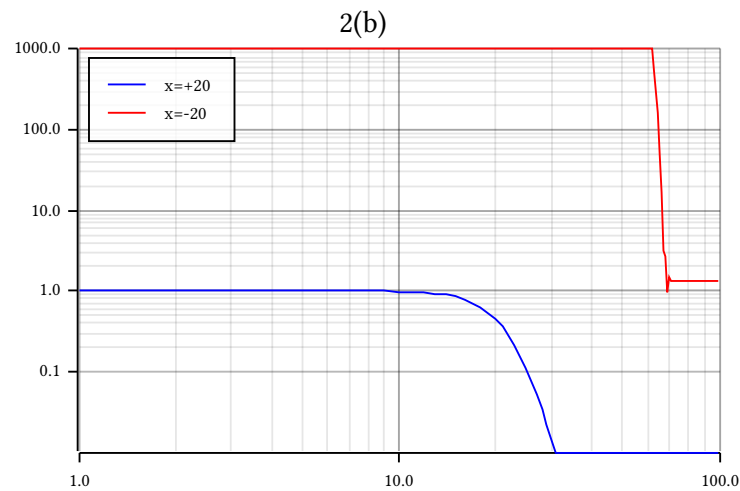
Done. See the `exponential_series` function in `hw01.rs`.

- 2(b)** For n ranging between 0 and 100, compare the result with the exponent calculated with a built-in function or function from a numerical library (e.g. `numpy.exp`) in the following way. Plot the error defined by

$$\epsilon_n := \frac{|e^x - S_n(x)|}{e^x}$$

on a log-log plot for a large positive and large negative exponent (e.g., $x = 20$ and $x = -20$). Describe what you see.

The plot:

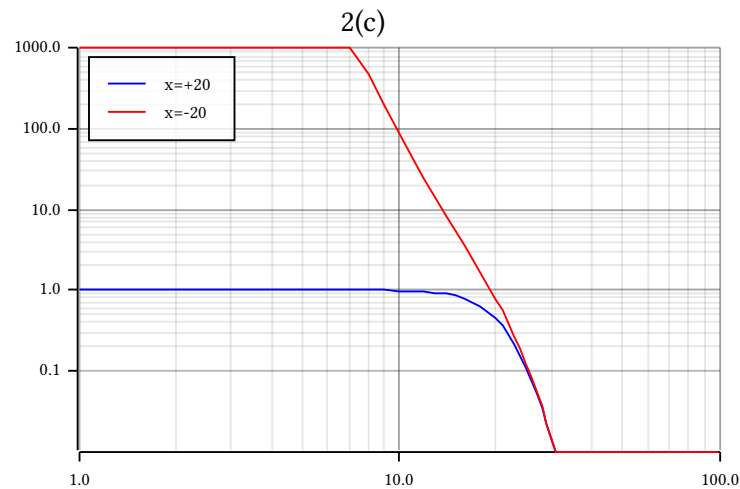


You can see that we actually never get to the correct answer for $x = -20$, because somewhere along the process of going to $n = 100$, factorial on a u64 overflowed.

- 2(c) Consider the following (trivial) equality: $e^{-x} = (e^{-1})^x$. Write a program that utilizes this equality to get a more accurate series expansion for large negative exponents. Plot ϵ_n on a log-log plot to demonstrate that you have achieved this.

This one's easy; we just rerun with

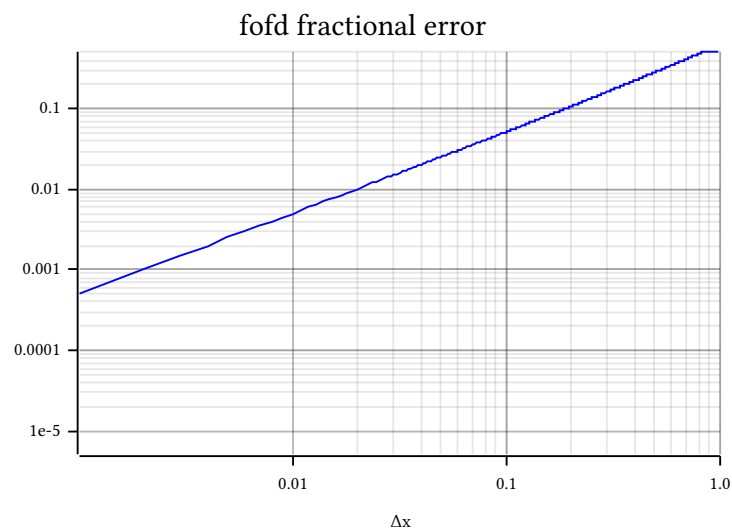
```
fn exponential_series_alt(n: u64, x: f64) → f64 {
    return 1.0 / exponential_series(n, -x);
}
```



3 errors in numerical differentiation

Calculate the derivative of the function $f(x) = \sin x$ at the point $x = \pi/4$ using the first-order forward difference. Plot on a log-log plot the error with respect to the analytical derivative for a wide range of Δx . Describe the behavior you see (especially for very small Δx) and the reason for the trends. How does it change if you use a second-order central difference? How about a fourth-order central difference?

First order forward:

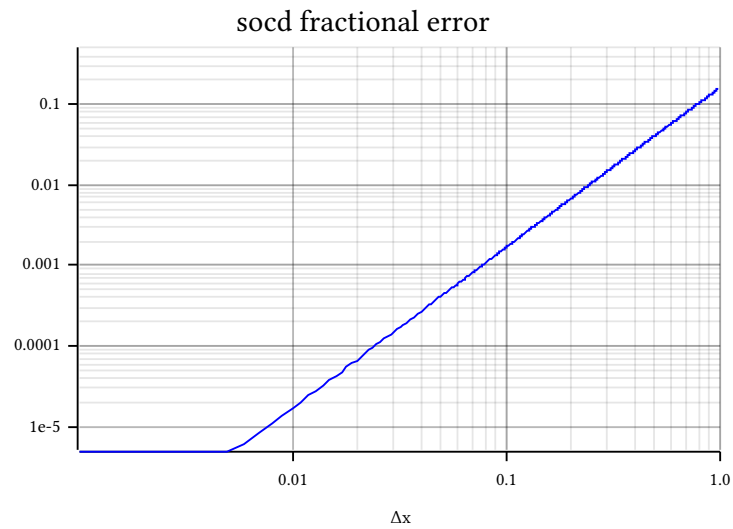


It's pretty much a straight line on a log-log plot. I think this makes sense, because the fractional error is (the magnitude of)

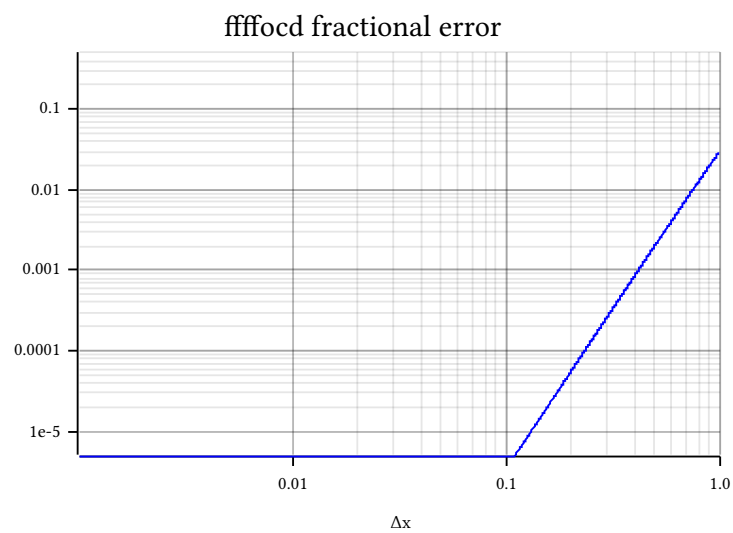
$$\frac{\frac{\sin(x+\Delta x) - \sin x}{\Delta x} - \cos x}{\cos x} = \frac{(\sin x)(\cos \Delta x - 1)}{\Delta x \cos x} - 1 \approx \frac{(\sin x)(-\Delta x^2)}{2 \Delta x \cos x} - 1$$

which is linear.

Second order central:



Fourth order central:



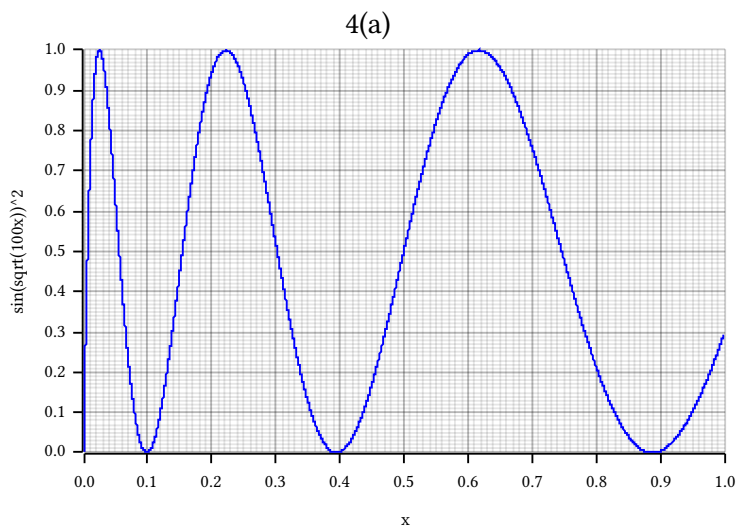
4 *comparing methods of integration*

Consider the variable

$$I = \int_0^1 (\sin \sqrt{100x})^2 dx$$

4(a) Plot the integrand over the range of the integral.

As instructed:



- 4(b) Write a program that uses the *adaptive trapezoid rule* to calculate the integral to an approximate accuracy of $\epsilon = 10^{-6}$, using the following procedure. Start with the trapezoid rule using a single subinterval. Double the number of subintervals and recalculate the integral. Continue to double the number of subintervals until the error is less than 10^{-6} . Recall that the error is given by $\epsilon_i = \frac{1}{3}(I_i - I_{i-1})$ where the number of subintervals N_i used to calculate I_i is twice that used to calculate I_{i-1} . To make your implementation more efficient, use the fact that

$$I_i = \frac{1}{2}I_{i-1} + h_i \sum_k f(a + kh_i)$$

where h_i is the width of the subinterval for the i th iteration, and k runs over *odd numbers* from 1 to $N_i - 1$.

Done in hw01.rs (see trapezoid_integrate):

```
4b: subintervals = 4096, ret = 0.455832058278271
```

- 4(c) Write a separate program that uses *Romberg integration* to solve the integral, also to an accuracy of 10^{-6} using the following procedure. First calculate the integral with the trapezoid rule for 1 subinterval (as you did in part (b)); we will refer to this as step $i = 1$, and the result as $I_1 = R_{1,1}$. Then calculate $I_2 = R_{2,1}$ using 2 subintervals. Using these two results, we can construct an improved estimate of the integral as: $R_{2,2} = R_{2,1} + \frac{1}{3}(R_{2,1} - R_{1,1})$. In general

$$R_{i,m+1} = R_{i,m} + \frac{1}{4^m - 1}(R_{i,m} - R_{i-1,m}).$$

Therefore, for each iteration i (where we double the number of subintervals), we can obtain improved approximations up to $m = i - 1$ with very minor extra work. For each i and m , we can calculate the error at previous steps as

$$\epsilon_{i,m} = \frac{1}{4^m - 1} (R_{i,m} - R_{i-1,m}).$$

Use these two equations to iterate until the error in $R_{i,i}$ is less than 10^{-6} . How significant is the improvement with respect to number of subintervals necessary compared to the approach of part (b)?

Done in hw01.rs (see romberg_integrate):

```
4c: subintervals = 64, ret = 0.45583249446137863
```

You can see that the number of subintervals reduced from 4096 to 64.

- 4(d) Use the Gauss–Legendre approach to calculate the integral. What order (i.e., how many points) do you need to obtain an accuracy below 10^{-6} ? You can find tabulated weights and points online.

Done in hw01.rs, quadgl.rs, and quadgl/quadgl_data.rs. We see:

```
4d: n = 2, ret = 0.6273262731105194
4d: n = 3, ret = 0.23603692720508807
4d: n = 4, ret = 0.4267093047956246
4d: n = 5, ret = 0.5631145323435273
4d: n = 6, ret = 0.470222775561266
4d: n = 7, ret = 0.45644586447958324
4d: n = 8, ret = 0.45584440641945256
4d: n = 9, ret = 0.455832655443542
4d: n = 10, ret = 0.455832533065431
4d: n = 11, ret = 0.4558325323120352
4d: n = 12, ret = 0.4558325323090928
```

Doing analytic integration in SageMath, we can see that the actual answer is 0.455832532309085:

```
sage: N(integrate(sin(sqrt(100*x))^2, x, 0, 1))
0.455832532309085
```

and we are within 10^{-6} of that with just $n = 9$.

5 integration to ∞

Consider the gamma function,

$$\Gamma(a) = \int_0^{\infty} x^{a-1} e^{-x} dx.$$

We want to evaluate this numerically, and we will focus on $a > 1$. Consider a variable transformation of the form:

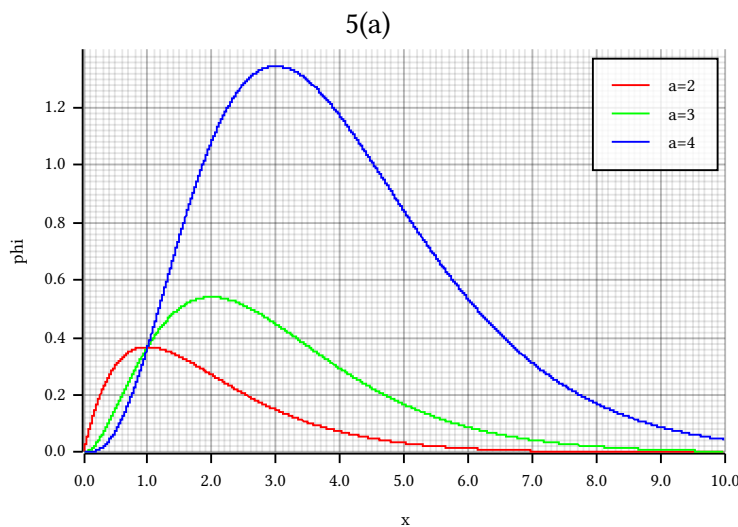
$$z = \frac{x}{x+c}.$$

This will map $0 \leq x < \infty$ to $0 \leq z \leq 1$, allowing us to do this integral numerically in terms of z . For convenience, we express the integrand as $\phi(x) = x^{a-1} e^{-x}$.

5(a)

Plot $\phi(x)$ for $a \in \{2, 3, 4\}$.

As instructed:



5(b) For what value of x is the integrand $\phi(x)$ maximum?

Setting the derivative equal to zero:

$$0 = \frac{d}{dx} \phi(x) = e^{-x}(a-1)x^{a-2} - x^{a-1}e^{-x} \Rightarrow (a-1)x^{a-2} = x^{a-1} \Rightarrow x = (a-1).$$

5(c) Choose the value c in our transformation such that the peak of the integrand occurs at $z = 1/2$. What value is c ?

This choice spreads the interesting regions of integrand over the domain $0 \leq z \leq 1$, making our numerical integration more accurate.

5(d) Find $\Gamma(a)$ for a few different values of $a > 1$ using any numerical integration method you wish, integrating from $z = 0$ to $z = 1$. Keep the number of points in your quadrature to a reasonable amount ($N \leq 50$).

Don't forget to include the factors you pick up when changing dx to dz .

Note that roundoff error may come into play in the integrand. Recognizing that you can write $x^{a-1} = e^{(a-1)\ln x}$ can help minimize this.