1. a) Let us first write out the Taylor expansions for $f(x \pm \delta)$ and $f(x \pm 2\delta)$:

$$f(x+\delta) = f(x) + \delta f'(x) + \frac{\delta^2}{2} f''(x)..., \tag{1}$$

$$f(x-\delta) = f(x) - \delta f'(x) + \frac{\delta^2}{2} f''(x)..., \tag{2}$$

$$f(x+2\delta) = f(x) + 2\delta f'(x) + 2\delta^2 f''(x)..., \tag{3}$$

$$f(x-2\delta) = f(x) - 2\delta f'(x) + 2\delta^2 f''(x).... \tag{4}$$

In theory what we want is to get $f'(x)$ in terms of the 4 points that we know and no other terms. Since we have 4 equations, we have 4 coefficients and hence, can only consider up to $f'''(x)$. If we multiply each equation by its respective coefficient, and sum them together such that $f'(x)\delta$ has a coefficient of 1 and everything else 0, we end up with a matrix. The matrix is as follows (in order of the equations above):

$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 1 & -1 & 2 & -2 & 1 \\ 1 & 1 & 4 & 4 & 0 \\ 1 & -1 & 8 & -8 & 0 \end{array}\right]$$

Using an online solver we find that:

$$\delta f'(x) = \frac{2}{3} f(x+\delta) - \frac{2}{3} f(x-\delta) - \frac{1}{12} f(x+2\delta) + \frac{1}{12} f(x-2\delta), \tag{5}$$

$$\therefore f'(x) = \frac{\frac{2}{3} f(x+\delta) - \frac{2}{3} f(x-\delta) - \frac{1}{12} f(x+2\delta) + \frac{1}{12} f(x-2\delta)}{\delta}. \tag{6}$$

b) The total error is a combination of the fact that we cut off the Taylor series after $f'''(x)$ and the machine precision. The round off error is simple to determine as just the next highest order term. Using (6) and just taking the Taylor series to the next leading order, even the 4th derivative cancels (since due to the nature of our derivative all the even order terms disappear) leaving the 5th derivative term.

$$\text{RoundError} = \frac{f'''''(x)\delta^4}{30}. \tag{7}$$

Now the machine precision can be determined by multiplying equation (6) by ϵ and using the approximation that at very small δ, $f(x \pm \delta) \approx f(x)$ as can be seen from equations (1)-(4). The maximum error is when each term is summed up so the minuses are turned to plus signs. Therefore, we get:

$$\text{Machine} = \epsilon \frac{\frac{2}{3} f(x+\delta) + \frac{2}{3} f(x-\delta) + \frac{1}{12} f(x+2\delta) + \frac{1}{12} f(x+2\delta)}{\delta} = \epsilon \frac{3f(x)}{2\delta}. \tag{8}$$

This gives a total error of:

$$\text{TotalError} = \text{RoundError} + \text{Machine} = \frac{f'''''(x)\delta^4}{30} + \epsilon \frac{3f(x)}{2\delta}. \tag{9}$$

Minimizing with respect to δ to find the best δ:

$$0 = 4\frac{f'''''(x)\delta^3}{30} - \epsilon \frac{3f(x)}{2\delta^2}, \tag{10}$$

$$\delta = \left(\epsilon \frac{45 f(x)}{4 f'''''(x)}\right)^{1/5}. \tag{11}$$

Using 64-bit, this gives $\delta \approx 0.00102, 0.102$ for $f(x) = e^x, e^{0.01x}$ by plugging into formula (11) and taking the derivatives (since $\epsilon = 10^{-16}$ for 64-bit). Note, the derivatives are fairly simply for e^x hence, δ can be found fairly easily. The python script uploaded to GitHub was used to show how accurate this estimate for δ was. The most accurate δ was found to be 0.00123 and 0.109 for e^x and $e^{0.01x}$, respectively. This is quite similar to the predictions, but not the exact same due to the approximations that were made. Furthermore, the optimum δ fluctuates drastically with a change in the number of δ's tested in the interval and hence, this result might not be the most accurate.

2. Most of this problem has been written in detail and explained in the script. The plot of the interpolation has been shown below to provide a gauge of accuracy.

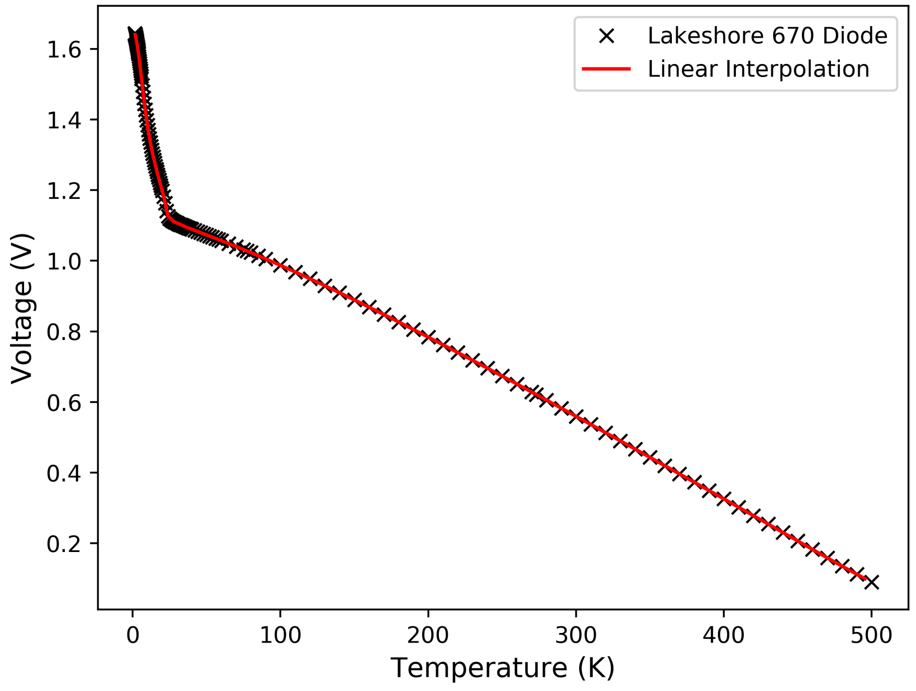

Figure 1: Lakeshore 670 diode data fit with our linear interpolation model.

3. Running the interpolation code for $\cos x$ with 6 points, and calculating a standard deviation to determine the difference between the interpolation and the actual values of the function. Polynomial, spline and rational fitting had standard deviations of approximately 0.00904, 0.00124, 0.000857 respectively. The plot of the difference between $\cos x$ and each fit has been shown below. As expected, the rational fit is the best and the polynomial is the worst.

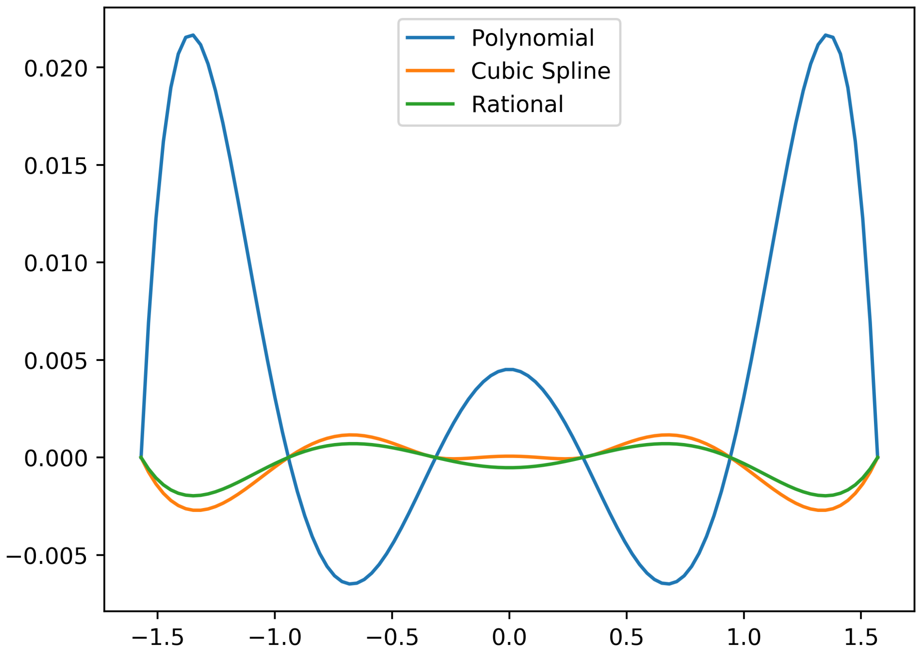

Figure 2: Difference between the actual function and $\cos x$ fittings for each type of interpolation.

Using an adapted version of the same code for the Lorentzian with 6 points as well and $n = 4$ and $m = 3$, the standard deviation is 0.0150, 0.00384 and 7.21×10^{-16}, for the polynomial, spline and rational fittings respectively.

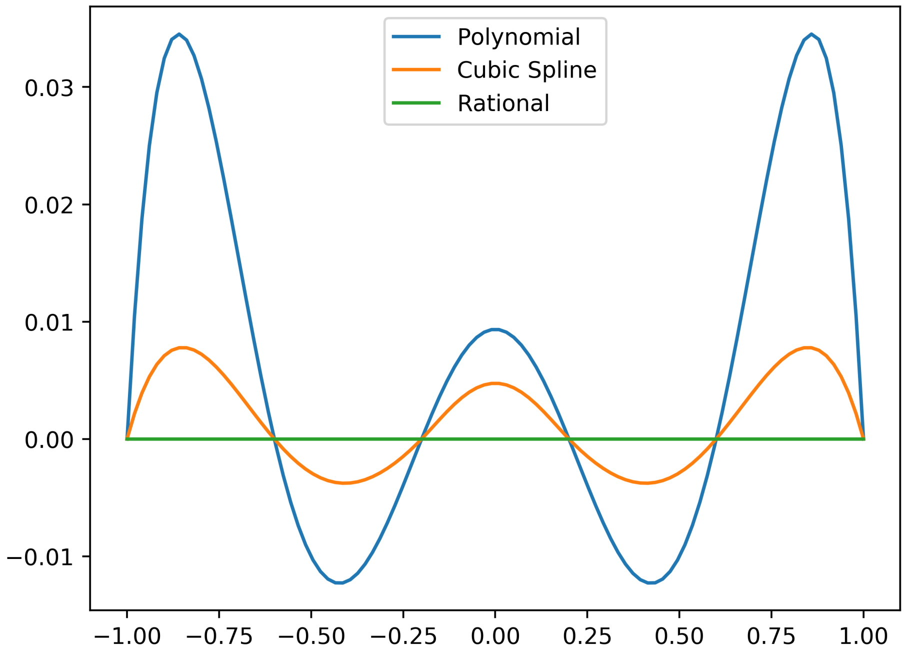

Figure 3: Difference between the actual function and Lorentzian fittings for each type of interpolation.

To get an estimate of the error from the rational function fit, the error will be the next highest order term in the Taylor series that has not been included. The Taylor series for the Lorentzian can be written as:

$$\frac{1}{1+x^2} = 1 - x^2 + x^4 - x^6 + x^8 - x^{10} + ... \tag{12}$$

The next highest order term depends on the number of points we use to calculate the rational fit as well as the values of n and m we choose. Hence, if we pick $n = m$ the only error is our machine precision which is what we see. Otherwise, we want n and m to be as close as possible such that $n >= m$ which allows for the next highest order term to be the smallest power of x. For example, if we use 8 points, $n = 4$ and $m = 5$ gives a standard deviation of 2.07 while $n = 5$ and $m = 4$ gives 0.773. Furthermore, $n = 6$ and $m = 3$ gives a standard deviation of 1.27×10^{-14}.

Using np.linalg.inv compared to np.linalg.pinv means that if the determinant of the matrix (mat in the code) is 0, then no inverse will be returned. However, pinv returns the pseudo inverse in this case and hence, will always return a value. In the case of the Lorentzian, this matrix does have a determinant that is 0 in certain cases. We can see this when looking at p and q. For example, with $n, m = 4, 5$, we get the following output for p and q using inv: $p = [2.308, 5, 7, -3.8944], q = [6, 9, -4.5, 8]$. However, using pinv we get: $p = [1, 1.78e-15, -3.33e-01, 0], q = [0, 6.67e-01, -1.78e-15, -3.33e-01]$. The key difference, is that p using pinv looks more like (12) where the first term is 1, the x term is practically 0, the x^2 term is negative etc. We do not see this with inv. Essentially, with these values of n and m, the determinant is essentially 0 and hence, pinv provides a better approximation of the inverse of this singular matrix allowing for better coefficients to be produced.

4. We start by quoting the integral from Griffith's in its simplified form, taking note to write E in terms of σ and z (distance) in terms of R. That way our integral is as general as possible.

$$\frac{E}{\sigma} = \frac{1}{2\epsilon_0} \int_{-1}^{1} \frac{z/R - u}{(1 + (z/R)^2 - 2(z/R)u)^{3/2}} du \tag{13}$$

There is a singularity in the integral at $z = R$. As a result, our integrator breaks at this point as it keeps getting recursively called infinitely. Scipy.quad on the other hand takes care of this issue for us. However, we can fix this by simply saying that at z=R, not to run the recursive script. Weirdly, a recursion error is also given at $0.7 < z/R < 1.4$, likely because of how quickly the function blows up. As a result the integrator is ignored between these points. The resulting plot is shown below,

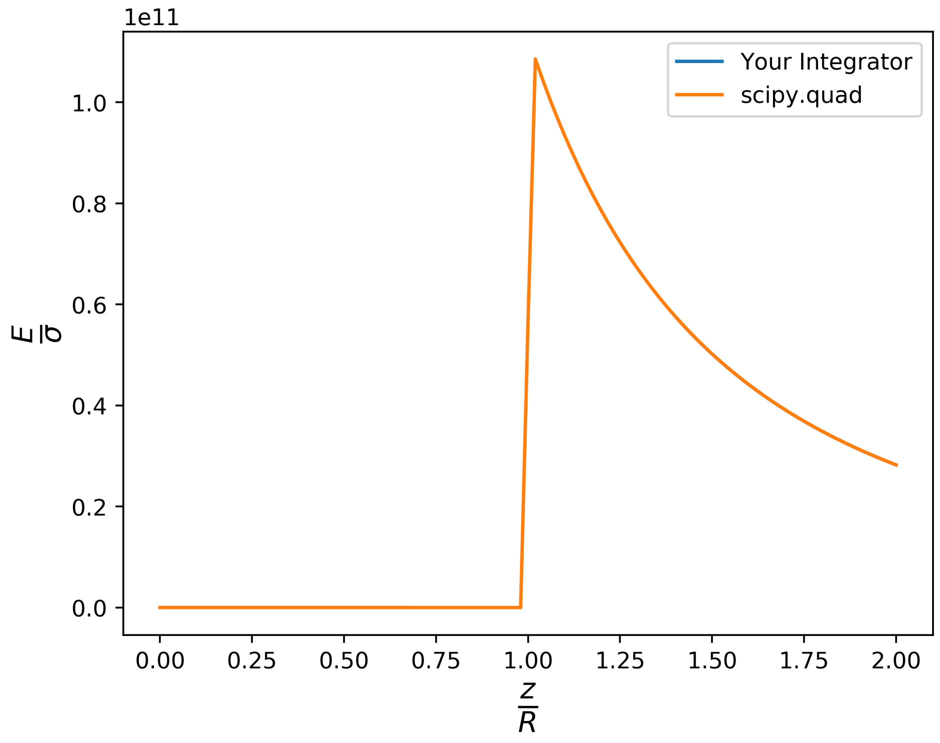

Figure 4: E/σ vs. z/R plotted as a result of scipy.quad integration and integration from our integrator. Both lines overlap and hence, almost no difference is seen.