9/10

Computational Physics
Homework 1

Matthew Drnevich

1. (a)

```python
import numpy as np

def forward_difference(f: Callable[[np.single], np.single], x: np.single, h:
    np.single):
    return (f(x + h) - f(x)) / h

def central_difference(f: Callable[[np.single], np.single], x: np.single, h:
    np.single):
    two = np.single(2)
    return (f(x + h/two) - f(x - h/two)) / h

def extrapolated_difference(f: Callable[[np.single], np.single], x: np.single,
     h: np.single):
    two = np.single(2)
    eight = np.single(8)
    twelve = np.single(12)
    return (eight*f(x+h) - eight*f(x-h) + f(x - two*h) - f(x + two*h)) / (
        twelve*h)
```

(b) Here are the figures obtained for the error of the derivative methods.

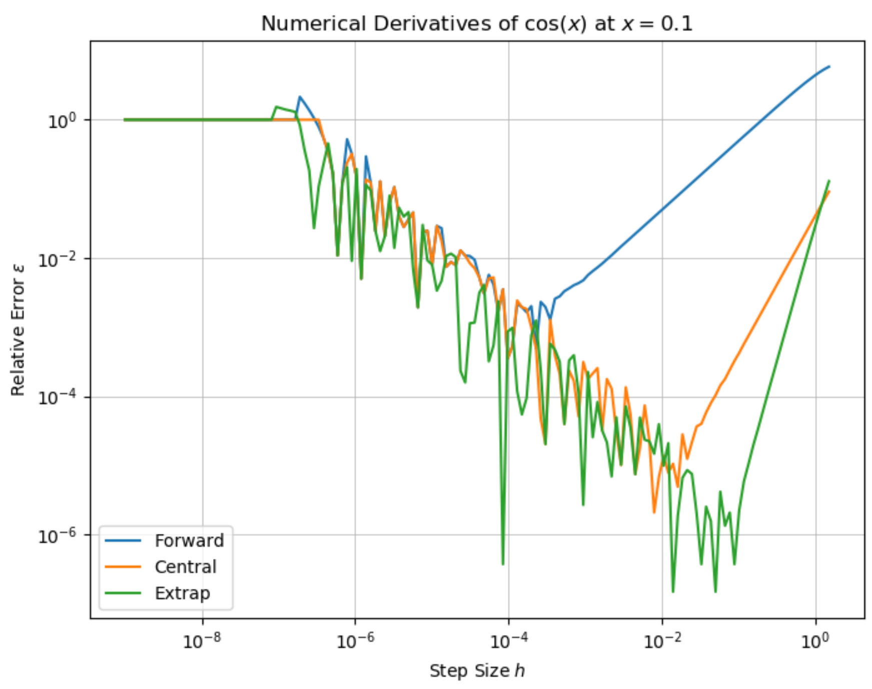


Figure 1: Numerical derivatives of $\cos(x)$ at $x = 0.1$

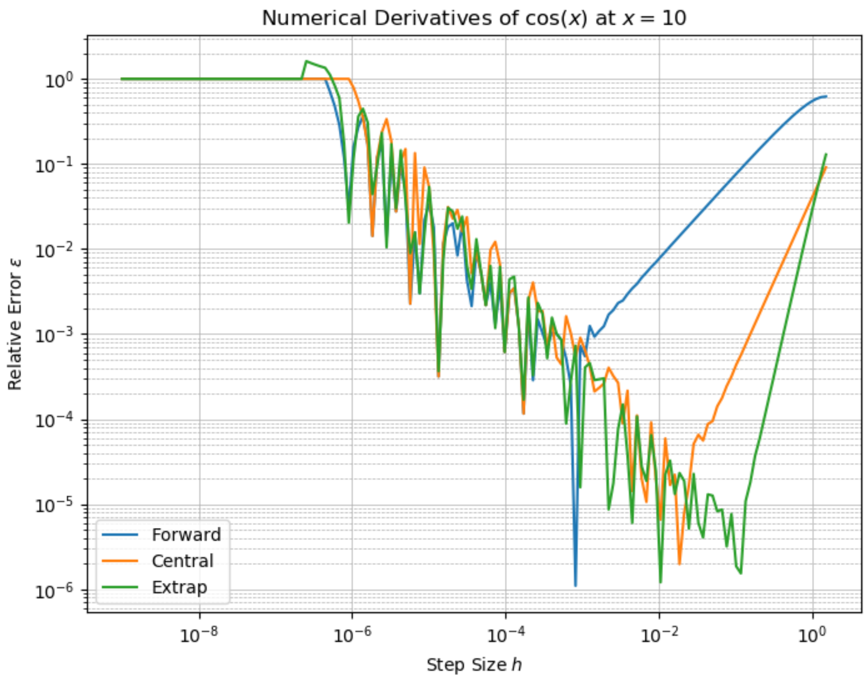


Figure 2: Numerical derivatives of $\cos(x)$ at $x = 10$

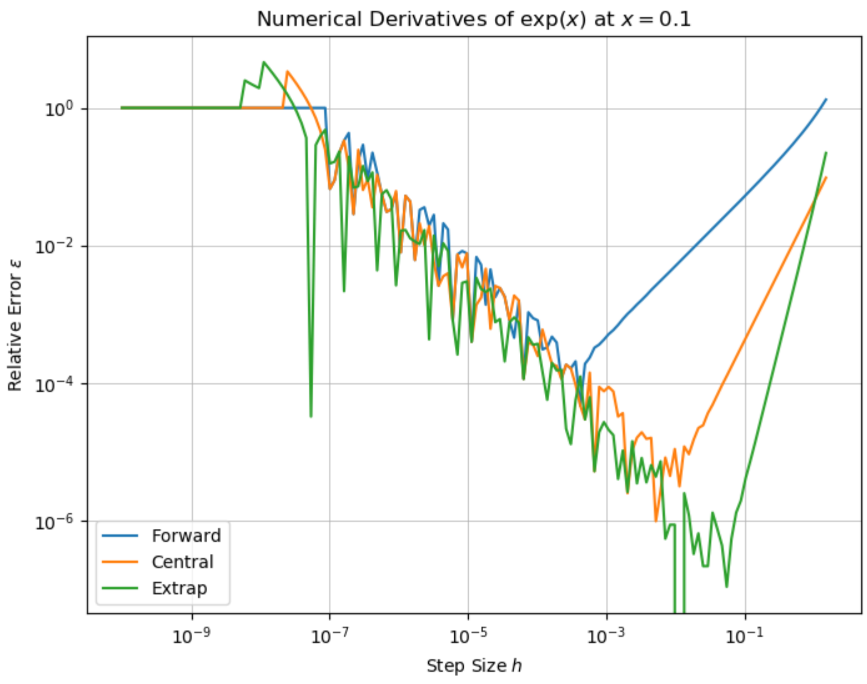
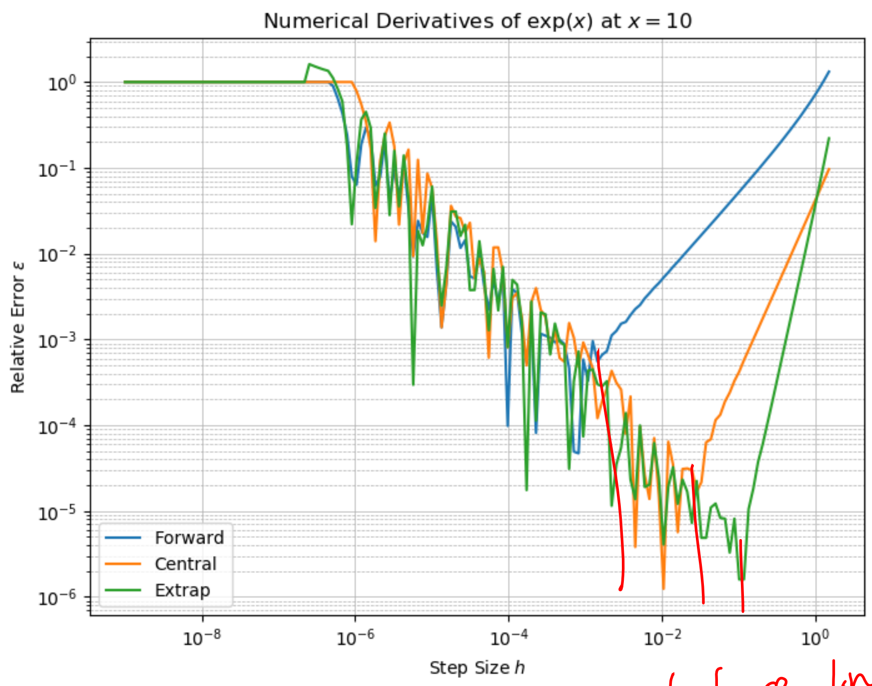


Figure 3: Numerical derivatives of $\exp(x)$ at $x = 0.1$



do these line up w/ expectations?

Figure 4: Numerical derivatives of $\exp(x)$ at $x = 10$

(c) At small h, the difference in the function values will be quite small, leading to cancellation and roundoff error, which grows with $\mathcal{O}(h^{-1/2})$, until it reaches machine precision size and gives us a derivative of zero due to roundoff error, causing the flatline. This all agrees with our graph. ✓

At large h, the approximation will be poor, leading to truncation error, which scales as

- Forward: $\mathcal{O}(h)$
- Central: $\mathcal{O}(h^2)$ ✓
- Extrapolated: $\mathcal{O}(h^4)$

The transition between these two errors is characterizes by the point of minimum error, i.e. the base of the "v" shape. ✓

2. (a)

```
import numpy as np

def midpoint_rule(f: Callable[[np.single], np.single], a: np.single, b: np.
    single, N: np.single):
    n = int(N)
    h = (b - a) / N
    s = np.single(0)
    one = np.single(1)
    two = np.single(2)
    for k in range(n):
        s += f(a + (two*np.single(k) + one)*h/two)
    return h*s

def trapz_rule(f: Callable[[np.single], np.single], a: np.single, b: np.single
    , N: np.single):
    n = int(N)
    h = (b - a) / N
    s = np.single(0.5)*f(a) + np.single(0.5)*f(b)
    for k in range(1, n):
        s += f(a + np.single(k)*h)
    return h*s

def simpsons_rule(f: Callable[[np.single], np.single], a: np.single, b: np.
    single, N: np.single):
    n = int(N)
    h = (b - a) / N
    s = f(a) + f(b)
    two = np.single(2)
    four = np.single(4)
    for k in range(1,n,2):
        s += four*f(a + np.single(k)*h)
    for k in range(2,n,2):
        s += two*f(a + np.single(k)*h)
    return np.single(1/3)*h*s
```

(b) Here is the figure obtained for the error of the integrals.

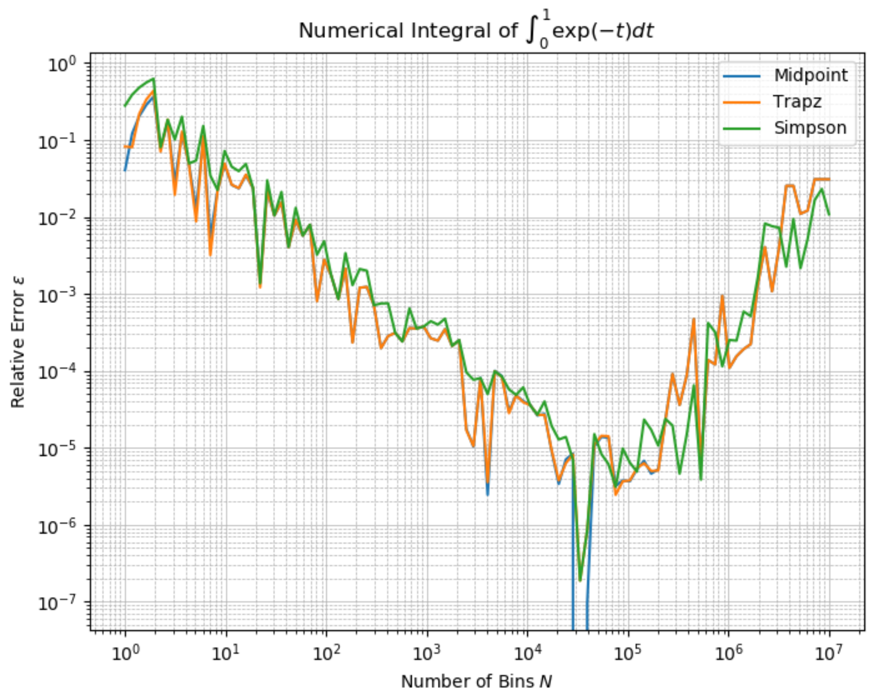


Figure 5: Numerical integral of $\int_0^1 \exp(-t)dt$

(c) At high N, the bins will be quite small, leading to small integrals, leading to roundoff error, which grows with $\mathcal{O}(\sqrt{N})$, agreeing with our graph.

At low N, the approximation will be poor, leading to truncation error, which scales as

- Midoint: $\mathcal{O}(N^{-1})$
- Trapezoidal: $\mathcal{O}(N^{-2})$
- Simpson's: $\mathcal{O}(N^{-4})$

-1

?

For this specific function we actually expect that each of these integration methods will roughly agree. This is because e^{-x} is a monotonically decreasing function, so midpoint is a very good approximation as it's underestimates and overestimates cancel out at each step in the iteration, leaving only the endpoints for error. The same argument can be made for the trapezoidal rule. Because of this large increase in accuracies for these methods, it puts them on par with Simpson's rule for this scenario, which is why midpoint and trapezoidal rules are essentially identical in error and are very close to Simpson's.

But you saw my results in class. You could prove this by using a different function.

3. For this problem, first I fit a cubic spline to the given data for the function $P(k)$. Then, I used trapezoidal rule (with double precision) to evaluate $\xi(r) = \frac{1}{2\pi^2}\int_{0.0001}^{1000} dk\, k^2 P(k) \frac{\sin(kr)}{kr}$. I chose the limits to be 0.0001 and 1000 because these were the minimum and maximum values of k, respectively, that were given in the datafile. If I extrapolated $P(k)$ beyond these values then my integral would be vastly incorrect, especially given the cubic spline approximation, which rapidly reaches very large negative numbers outside of this range. Therefore, the best approach given the data is to only integrate within this range. With this approach, the graph on the left shows the values of the cubic spline evaluated at the k values given in the datafile, which perfectly agrees with the dataset and shows no pathological behavior. On the right, we see the resultant graph of $r^2\xi(r)$ in the range $[0, 120]$ Mpc/h, which follows the shape we expected and demonstrates a BAO peak near $r = 106$ Mpc/h.

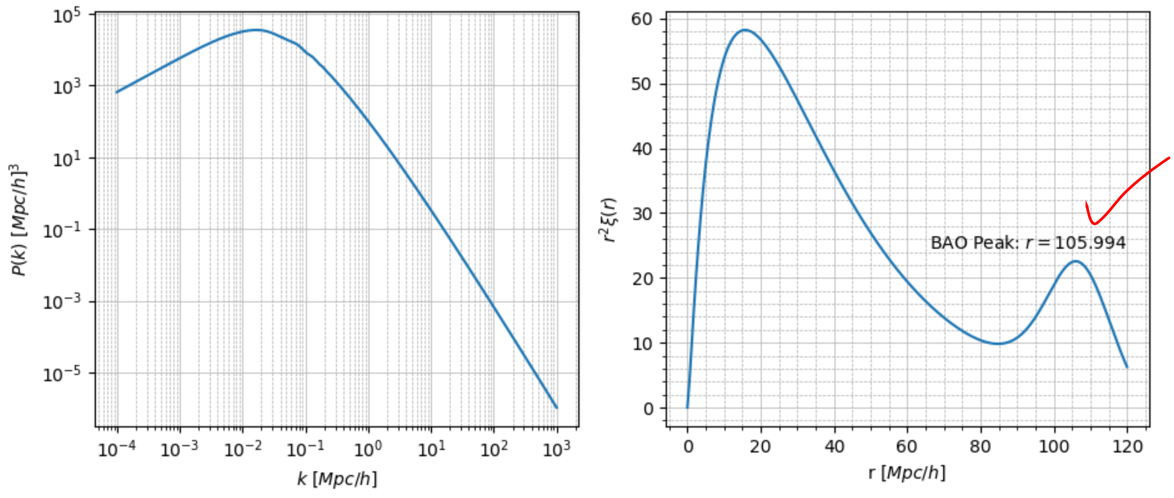


Figure 6: Graphs demonstrating the fit of the power spectrum and the correlation function.