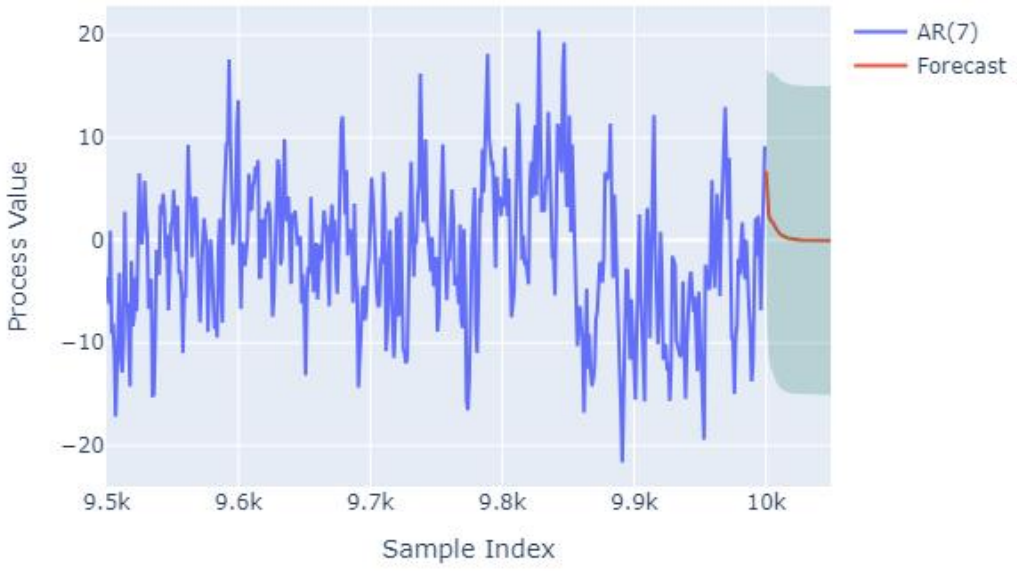
1.

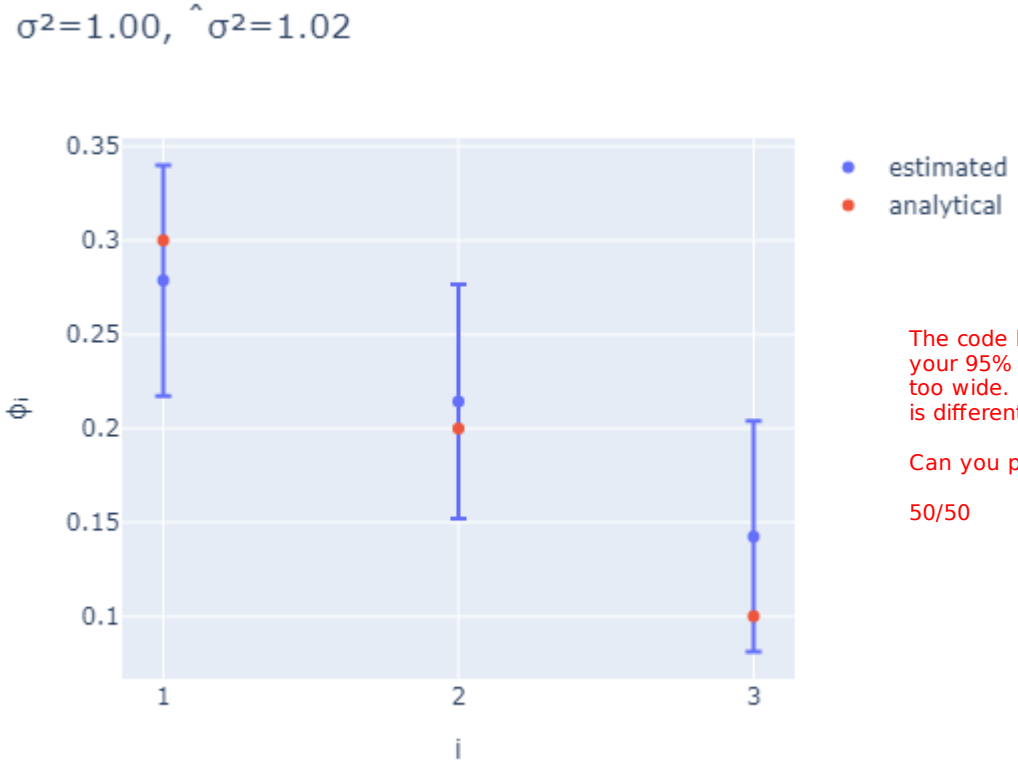


Code for the `forecast` function:

```
def forecast(x, acov, mu, m, max_h):
    Gamma_m = buildGamma(acov=acov, m=m)
    forecasts_means = np.empty(max_h, dtype=np.double)
    forecasts_vars = np.empty(max_h, dtype=np.double)
    xMinusMu_mR = (x-mu)[::-1][:m]
    for h in range(1, max_h+1):
        #compute gamma_mh, i.e., covariance between observed series and future
step h)
1       gamma_mh = np.array([acov[h + k] for k in range(m)])
        #find the vector a_m for the linear equation Gamma_m*a_m=gamma_mh
        a_m = np.linalg.solve(Gamma_m, gamma_mh)
        #forecast mean: mean (mu) + weighted sum of past deviations
        forecasts_means[h - 1] = mu + np.dot(a_m, xMinusMu_mR)
        #forecast variance: variance of white noise minus explained variance
        forecasts_vars[h - 1] = acov[0] - np.dot(gamma_mh, a_m)
    return forecasts_means, forecasts_vars
```

Nerdy point: a more efficient line 1 is gamma_mh = acov[h:h+m]

2.



The code below looks right, but your 95% confidence intervals are too wide. Maybe our data length T is different? I am using T=10000.

Can you push to the repo all your code?

50/50

Code for the `estimateCoefsAndNoiseVarARpYW` function:

```
def estimateCoefsAndNoiseVarARpYW(acov, p, N):
    Gammap = buildGamma(acov=acov, m=p)
    gammaph = acov[1:]
    #find the vector of phi values for the linear equation Gammap*phi=gamma_mh
    phiHat = np.linalg.solve(Gammap,gammaph)
    #calculate the sigma^2=autocov(0)-phi_hat_transposed*gamma_mh
    sigma2Hat = acov[0] - np.inner(gammaph, phiHat) # complete
    #calculate the covariance of the estimated AR coefficients:
    phiCovHat = sigma2Hat * np.linalg.inv(Gammap)/N
    return phiHat, phiCovHat, sigma2Hat
```