```python
import numpy as np
from multitaper import MTSpec
from scipy.fft import fft, ifft
from scipy.linalg import solve_toeplitz

def prewhiten(xt, order=2, nw = 4, kspec = 7, nfft_add=2,
return_components=False):
    """
    NOTE: Claude Sonnet 4.0 was _initially_ used to convert the
prewhiten.R file to a set of Python functions, however it did such a
terrible job that I wrote it myself.

    Prewhitens a univariate time series using an autoregressive (AR)
model.

    The model coefficients are estimated by:
      1. Estimating the spectrum of the time series using a low-
variance, low-bias estimator (this amounts to using larger nw and kspec
values in the aMTM spectrum).
      2. Obtaining the autocovariance sequence (ACVS) estimate by taking
the inverse FFT of the spectrum from 1.
      3. Using the Yule-Walker equations and Levinson recursion
(solve_toeplitz()). to solve an equation of the form in Percival and
Walden (2020) just under Equation (450b); i.e., solving for p in the
equation g = Gp where g is the vector of ACVS values starting with lag-
1, G is a toeplitz matrix with first row and column the ACVS values
(lag-0 to lag-(p-1)), and p is the vector containing the AR(P)
coefficients.
      4. Obtaining the first-step predictions of the input time series
using the AR coefficients from 3.
      5.

    Parameters
    ----------
    xt : array-like
        Input time series xt (1D np.array)
    order : int, default=2
        Order of the AR model to fit
    nw : float, default=4
      Time bandwidth parameter for estimating the autocovariance
sequence (ACVS). Typically an integer or of the form X.5 where X is an
integer.
    kspec : int, default=7
      Number of tapers to use for the spectrum that will be inverted to
    nfft_add : int, default=2
        Zero-padding factor for FFT.
    return_components : bool, default=False
        If True, returns dict with prewhitened xt, AR coefficients, and
autocovariances
        If False, returns only the prewhitened x

    Returns
    -------
```

```
    numpy.ndarray or dict
        If return_components=False: prewhitened time series
        If return_components=True: dict with keys 'xt_pw', 'ar_coef',
'acvs'
    """

    N = len(xt)

    # determine the zero-padded length of the series
    nfft = int(np.power(2, np.ceil(np.log2(N)) + nfft_add))

    # remove the mean
    xt_mean = xt - np.mean(xt)

    # calculate the aMTM spectrum
    xt_mtspec = MTSpec(x = xt_mean, nw = nw, kspec = kspec, dt = 1, nfft
= nfft, iadapt = 0)

    # Inverse FFT to obtain aMTM estimates of the ACVS
    xt_acv = ifft(xt_mtspec.spec, axis=0).real

    # Yule-Walker equations and Levinson recursion used to obtain the
    # AR coefficients
    phi = solve_toeplitz(xt_acv[0:order, 0].ravel(), xt_acv[1:(order+1),
0].ravel())

    # Calculate the fitted values (one-step predictions)
    xt_ar_fit = np.convolve(xt_mean, phi, mode='valid')[:-1]

    # Below we have "fitted minus observed", which is not what you would
think
    # should be the approach, however if you use "observed minus
fitted", you end up
    # with a blue spectrum rather than a white spectrum.
    # I have to look at this more closely as the rationale is escaping
me at the
    # moment (other than, "it works using this approach," which is
    # highly unsatisfying).
    xt_prewhitened = xt_ar_fit - xt_mean[order:]

    # Drop the first order number values as these would not be using all
of the
    # AR coefficients.
    # Also remove the mean again, _just_ in case.
    xt_pw = xt_prewhitened[order:] - np.mean(xt_prewhitened[order:])

    if return_components:
      return({"xt_pw": xt_pw, "ar_coef": phi, "acvs": xt_acv});
    else:
      return(xt_pw);

def correct_spec_for_pw(spec_pw, ar_coef, dt=1):
    """
    Corrects a prewhitened spectrum by removing the effect of AR model
```

```
prewhitening.

    Parameters
    ----------
    spec_pw : multitaper.MTSpec object
        The prewhitened spectrum object corresponding to a series that
was prewhitened using an AR model.
        Assumes the full frequency array as returned by the multitaper
package.
    ar_coef : array_like
        The AR coefficients used in prewhitening (phi_1, ..., phi_p),
assuming the model:
        X_t = phi_1 * X_{t-1} + ... + phi_p * X_{t-p} + Îµ_t
    nfft : int, optional
        Number of points in the FFT used for frequency-domain
correction. If not specified, it defaults to the next power of 2 above
the length of `spec_pw`, plus 2.
    dt : float64
        The sampling rate of the series. This should be left as 1 in all
cases that I can think of, but I provided the option just in case.

    Returns
    -------
    spec_corrected : np.ndarray
        The spectrum corrected for the prewhitening filter, ideally
recovering the
        original (unwhitened) spectral shape.

    Notes
    -----
    The AR transfer function is computed as:
        H_AR(Ï‰) = 1 - Ï†â‚ e^{-iÏ‰} - ... - Ï†_p e^{-i p Ï‰}
    The correction divides `spec_pw` by |H_AR(Ï‰)|Â².
    """

    nfft = len(spec_pw)
    N_phi = len(ar_coef)

    ar_padded = np.pad(np.append(-1, ar_coef), pad_width=(0, nfft -
(N_phi+1)))

    ar_H = fft(ar_padded)
    ar_H2 = dt * (np.abs(ar_H)**2)
    spec_num = spec_pw.ravel()

    return(spec_num / ar_H2)
```