Time series analysis:
Unit root test and Hurst Exponent

ΕΠΛ 428: IOT PROGRAMMING

Dr. Panayiotis Kolios
Assistant Professor, Dept. Computer Science,
KIOS CoE for Intelligent Systems and Networks
Office: FST 01, 116
Telephone: +357 22893450 / 22892695
Web: https://www.kios.ucy.ac.cy/pkolios/

Πανεπιστήμιο Κύπρου

RECAP

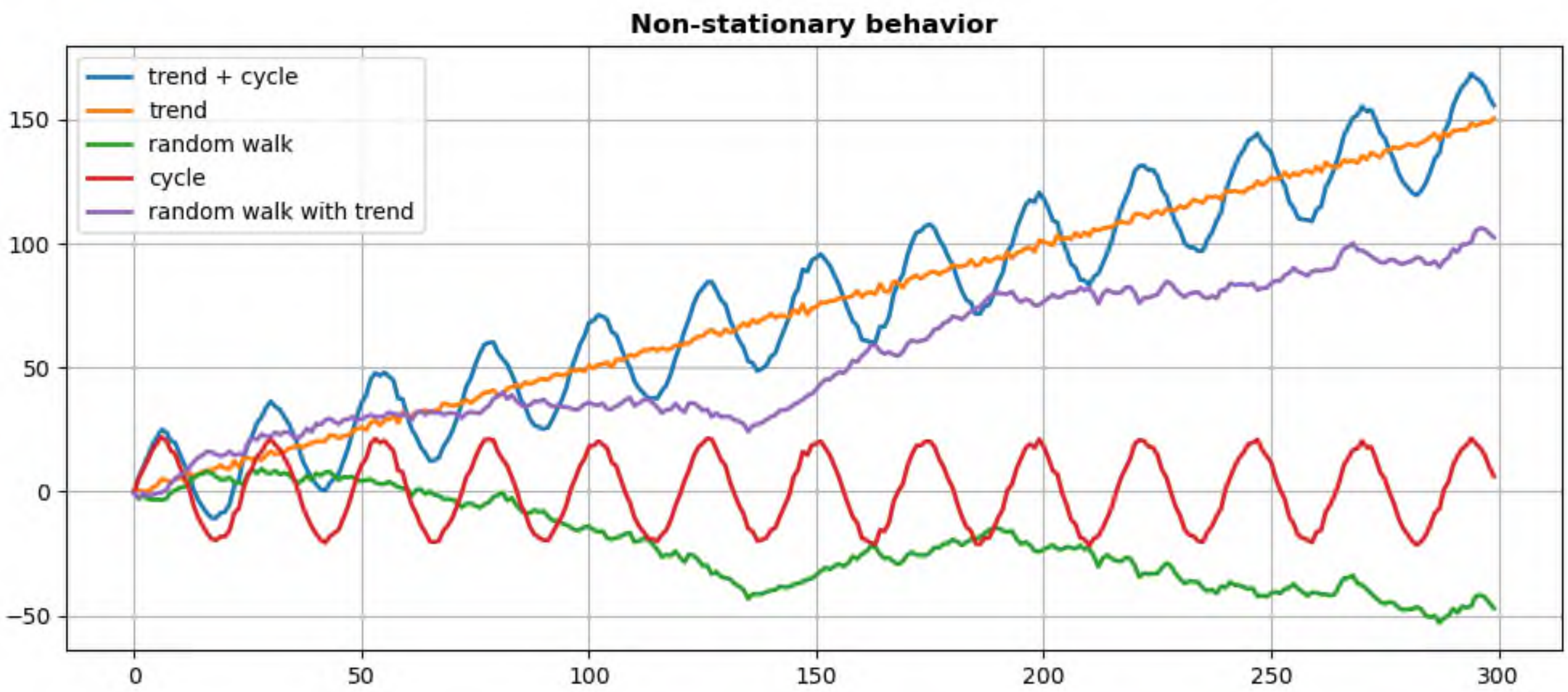
- Decomposition
  - trend, seasonal, and random fluctuation components
- Trends
  - Increasing / Decreasing / Flat
  - Larger trends can be made up of smaller trends
  - No defined timeframe for what constitutes a trend: it depends on your data and task at hand
- Seasonal effects
  - Weekend retail sales spikes
  - Holiday shopping.
  - Energy requirement changes with annual weather patterns.
- Random Fluctuations
  - Observation errors
  - Uncertainty / noise / faults
  - The smaller this is in relation to trend and seasonal components, the better we can predict the future

Πανεπιστήμιο Κύπρου

Recap Additive vs Multiplicative

- Additive
    - Data = Trend + Seasonal + Random.
    - If our seasonality and fluctuations are stable, we likely have an additive model
- Multiplicative
    - Data = Trend
    - Seasonal
    - Random
    - Similar to additive in log scale:
      log(Data) = log(Trend + Seasonal + Random).
    - Use multiplicative models if:
        - the amplitude in seasonal and random fluctuations grow with the trend
        - the percentage change of our data is more important than the absolute value change (e.g. stocks, commodities)

- A time series is stationary if:
  - The mean of the series is constant
  - The variance does not change over time (homoscedasticity)
  - The covariance is not a function of time



Checking stationarity

- Make a run-sequence plot
- Rolling statistics:
    - Compute and plot rolling statistics such as moving average/variance
    - Check if the statistics change over time
    - This technique can be done on different windows (small windows are noisy, large windows too conservative)
- Histogram of time series:
    - Does it look normal? -> stationary
    - Does it look non-normal (e.g., uniform)? -> non-stationary
- Augmented Dickey-Fuller (ADF) test:
    - Statistical tests for checking stationarity
    - The null hypothesis H_0 is that the time series is non-stationary
    - If the test statistic is small enough and the p-value below the target a, we can reject H_0, i.e., series is stationary

Achieving stationarity

- Take the log of the data
- Difference (multiple times if needed) to remove trends and seasonality OR
- Subtract estimated trend and seasonal components

- Random Walk

$$X(t) = X(t-1) + \varepsilon_t$$

- where ε_t are called innovations and are iid, e.g. $\varepsilon_t \sim N(0, \sigma^2)$
- Can also expressed as

$$X(t) = X(0) + W(t)$$

- with $W(t)$ being the Wiener process, or:

$$X(t) = X(0) + B(t)$$

- with $B(t)$ being the Brownian Motion
- $W(t)$ and B(t) are cumulative sums of normality distributed $\varepsilon_1 + \varepsilon_2 + \cdots + \varepsilon_t$
- Variance of Brownian Motion at time lag τ

$$Var(X(t+\tau) - X(t))$$

- increment $B(t+\tau) - B(t)$ is normally distributed with 0 mean and variance τ

Πανεπιστήμιο Κύπρου

- ADF test is one of the most popular unit root tests
- The presence of a unit root suggests that the time series is generated by a stochastic process with some level of persistence
- This means that shocks to the system will have permanent effects
- This is opposed to stationary processes where shocks have only temporary effects

- Consider a simple autoregressive process of order 1, denoted as AR(1):

$$Y(t) = \varphi Y(t-1) + \varepsilon_t$$

- where φ is a coefficient, and ε_t a white noise error term
- To analyze the properties of this process, we can rewrite the equation in terms of lag operator L

$$LY(t) = Y(t-1)$$

$$(1-\varphi L)Y(t) = e_t$$

- $(1-\varphi L)$known as the characteristic equation of AR(1)
- The roots of this equation are found by setting $1-\varphi L = 0$ and solving for L
- Giving $L = 1/\varphi$ which is the root of the characteristic eq.
- For $\varphi = 1$, $L = 1$, given a root with "unit" value
- AR(1) with unit root is not stationary since it becomes random walk: $Y(t) = \varphi Y(t-1) + \varepsilon_t$

Formulation of the test

- ADF test assesses whether lagged values of the time series are useful in predicting current values
- The test starts with a model that includes the time series lagged by one period (lag-1)
- Then, other lagged terms are added to control for higher-order correlation (this is the "augmented" part of the ADF test)

ADF test equation

- Models the time series as follows:

$$\Delta Y(t) = \alpha + \beta t + \gamma Y(t-1) + \sum_{i=1}^{p} \delta_i \Delta_{t-i} + \varepsilon_t$$

- with $\Delta_t = Y(t) - Y(t-1)$, difference at time t
- α being a constant, β trend, γ coefficient of lagged values
- δ_i coefficients for lagged differences (account for higher order correlations)

Πανεπιστήμιο Κύπρου

ADF connection to unit root

- Assume $\alpha = \beta = 0$ (zero mean and no trend) and do not consider higher-order terms ($\delta_i = 0$)

- Let $\gamma = (\varphi - 1)$

  - $\Delta_t = (\varphi - 1)Y(t) + \varepsilon_t$

- If $\gamma = 0$ then $\varphi = 1$ (unit root)

  - $\Delta_t = \varepsilon_t$ -> $Y(t) = Y(t-1) + \varepsilon_t$

Null and alternative hypotheses

- $H_0 : \gamma = 0$, the time series has unit root, i.e., it is not stationary
- $H_1 : \gamma < 0$, time series does not have unit root

Test statistics

- The ADF test statistic is calculated based on the estimated coefficient $\hat{\gamma}$
- This statistic is then compared to critical values for the ADF distribution
- If the test statistic is more negative than the critical value, H_0 is rejected
- If the test statistic is less negative than the critical value, H_0 cannot be rejected.

Choosing lag length

- The number of lags (p) included in the test equation is important
- Too few lags might leave out necessary corrections for autocorrelation
- Too many lags can reduce the power of the test
- The appropriate lag length is often chosen based on information criteria such as the Akaike Information Criterion (AIC) or the Schwarz Information Criterion (BIC)

Choosing lag length

```
# Generating a synthetic time series (replace this with your dataset)
data = pd.Series(100 + np.random.normal(0, 1, 100).cumsum())

# Perform Augmented Dickey-Fuller test
# the lag can be set manually with 'maxlag' or inferred automatically with
autolag
result = adfuller(data, autolag='AIC')  # You can change to 'BIC' for Schwarz
Information Criterion

adf_statistic, p_value, usedlag, nobs, critical_values, icbest = result
print(f'ADF Statistic: {adf_statistic :.2f}')
print(f'p-value: {p_value :.2f}')
print(f'Used Lag: {usedlag}')
print(f'Number of Observations: {nobs}')
print(f"Critical Values: {[f'{k}: {r:.2f}' for r,k in
zip(critical_values.values(), critical_values.keys())]}\n")
```

ADF Statistic: -1.13

p-value: 0.70

Used Lag: 0

Number of Observations: 99

Critical Values: ['1%: -3.50', '5%: -2.89', '10%: -2.58']

Πανεπιστήμιο Κύπρου

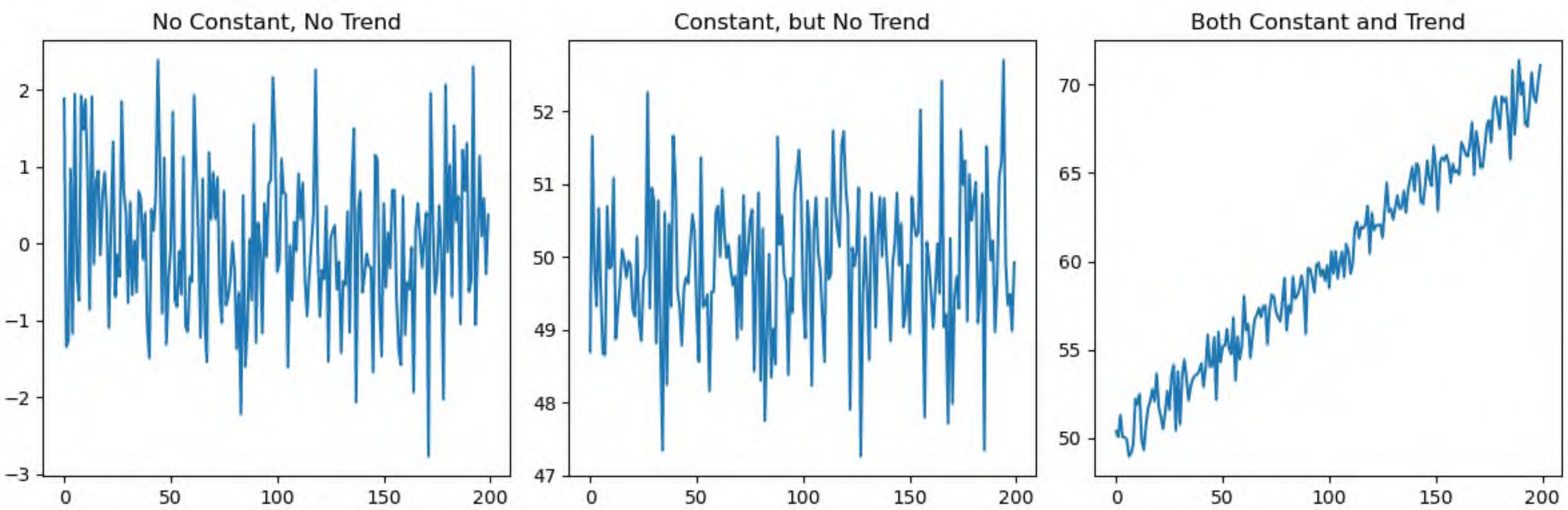
ADF Test types

- 3 versions depending on whether the equation includes none, both, or one of the terms α (constant) and βt (trend):
    - No constant or trend ('n').
    - Constant, but no trend ('c').
    - Both constant and trend ('ct').

```
# Function to perform ADF test
def perform_adf_test(series, title, regression_type):
    out = adfuller(series, regression=regression_type)
    print(f"Results for {title}:")
    print(f'ADF Statistic: {out[0]:.2f}')
    print(f'p-value: {out[1]:.3f}')
    print(f"Critical Values: {[f'{k}: {r:.2f}' for r,k in zip(out[4].values(),
out[4].keys())]}\n")

# 1. No Constant or Trend
series_no_const_no_trend = pd.Series(np.random.normal(0, 1, 200))

# 2. Constant, but No Trend
series_const_no_trend = pd.Series(50 + np.random.normal(0, 1, 200))

# 3. Both Constant and Trend
series_const_trend = pd.Series(50 + np.linspace(0, 20, 200) + np.random.normal(0,
1, 200))
```

ADF Test types

```
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
series_no_const_no_trend.plot(title='No Constant, No Trend')
plt.subplot(1, 3, 2)
series_const_no_trend.plot(title='Constant, but No Trend')
plt.subplot(1, 3, 3)
series_const_trend.plot(title='Both Constant and Trend')
plt.tight_layout();
```



ADF Test types

```
# 1. No Constant or Trend
perform_adf_test(series_no_const_no_trend, "No Constant, No Trend", 'n')

# 2. Constant, but No Trend
perform_adf_test(series_const_no_trend, "Constant, No Trend", 'c')

# 3. Both Constant and Trend
perform_adf_test(series_const_trend, "Constant and Trend", 'ct')
```

Results for No Constant, No Trend:

ADF Statistic: -15.47

p-value: 0.000

Critical Values: ['1%: -2.58', '5%: -1.94', '10%: -1.62']

Results for Constant, No Trend:

ADF Statistic: -13.95

p-value: 0.000

Critical Values: ['1%: -3.46', '5%: -2.88', '10%: -2.57']

Results for Constant and Trend:

ADF Statistic: -14.68

p-value: 0.000

Critical Values: ['1%: -4.00', '5%: -3.43', '10%: -3.14']

- Mean reversion refers to the property of a time series to revert to its historical mean
- This concept is particularly popular in financial economics, where it is often assumed that asset prices and revert to their historical average over the long term
- Application examples
    - Portfolio Management: Investors use mean reversion as a strategy to buy assets that have underperformed and sell assets that have overperformed, expecting that they will revert to their historical mean
    - Risk Management: Understanding mean reversion helps in assessing the long-term risk of assets. If an asset is highly mean-reverting, it might be considered less risky over the long term, as it tends to move back to its average
    - Pricing Models: In option pricing, certain models assume mean reversion in the underlying asset's volatility. This affects the pricing and strategy for options trading
    - Economic Forecasting: Economic variables (like GDP growth rates, interest rates) often exhibit mean-reverting behavior. This assumption is used in macroeconomic models and forecasts.

Mean reversion test

- Determines whether, after a deviation from its mean, a time series will eventually revert back to that mean
- This can be done using unit root tests such as ADF
- If a time series has a unit root, it implies that it does not revert to a mean

```
def get_data(tickerSymbol, period, start, end):

    # Get data on the ticker from Yahoo Finance
    tickerData = yf.Ticker(tickerSymbol)

    # Get the historical prices for this ticker
    tickerDf = tickerData.history(period=period, start=start, end=end)

    return tickerDf

data = get_data('GOOG', period='1d', start='2004-09-01', end='2020-08-31')
```

```
def get_data(tickerSymbol, period, start, end):

    # Get data on the ticker from Yahoo Finance
    tickerData = yf.Ticker(tickerSymbol)

    # Get the historical prices for this ticker
    tickerDf = tickerData.history(period=period, start=start, end=end)

    return tickerDf

data = get_data('GOOG', period='1d', start='2004-09-01', end='2020-08-31')
```

```
# Plotting the Closing Prices
plt.figure(figsize=(14, 5))
plt.plot(data['Close'], label='GOOG Closing Price')
plt.title('Google Stock Closing Prices (2004-2020)')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend();
```

```
# Perform the ADF test
perform_adf_test(data['Close'],"Google Stock Closing Prices", 'ct')
```

Results for Google Stock Closing Prices:

ADF Statistic: -0.78

p-value: 0.968

Critical Values: ['1%: -3.96', '5%: -3.41', '10%: -3.13’]

- H_0 cannot be rejected

- Testing for mean reversion and testing for stationarity are related but distinct concepts in time series analysis

Key Differences:

- Mean reversion testing is focused on whether a time series will return to a specific level (the mean)
- Stationarity testing checks if the overall statistical properties of the series remain consistent over time
- A stationary time series may or may not be mean reverting
- A stationary series with a constant mean and variance over time might still not revert to its mean after a shock
- Conversely, a mean-reverting series must have some stationarity, particularly in its mean, but it might still have changing variance or other properties over time

- The Hurst exponent (H) is a measure used to characterize the long-term memory of time series
- It helps to determine the presence of autocorrelation or persistence in the data
- The goal of the Hurst Exponent is to provide us with a scalar value that will help us to identify whether a series is:
    - random walk
    - trending
    - mean reverting
- The key insight is that, if any autocorrelation exists, then

$$Var(X(t+\tau) - X(t)) \propto \tau^{2H}$$

- with H being the Hurst exponent

Πανεπιστήμιο Κύπρου

- Time series can be characterized using Hurst exponent (H):
    - If $H = 0.5$, the time series is similar to a random walk (Brownian motion). In this case, the variance increases linearly with τ
    - If $H < 0.5$, the time series exhibits anti-persistence, i.e. mean reversal. The variance increases more slowly than linearly with τ
    - If $H > 0.5$, the time series exhibits persistent long-range dependence, i.e. is trending. The variance increases more rapidly than linearly with τ

```
def hurst(ts):
    # Create the range of lag values
    lags = range(2, 100)
    # Calculate the array of the variances of the lagged differences
    tau = [np.sqrt(np.std(np.subtract(ts[lag:], ts[:-lag]))) for lag in lags]
    # Use a linear fit to estimate the Hurst Exponent
    poly = np.polyfit(np.log(lags), np.log(tau), 1)
    # Return the Hurst exponent from the polyfit output
    return poly[0]*2.0
```

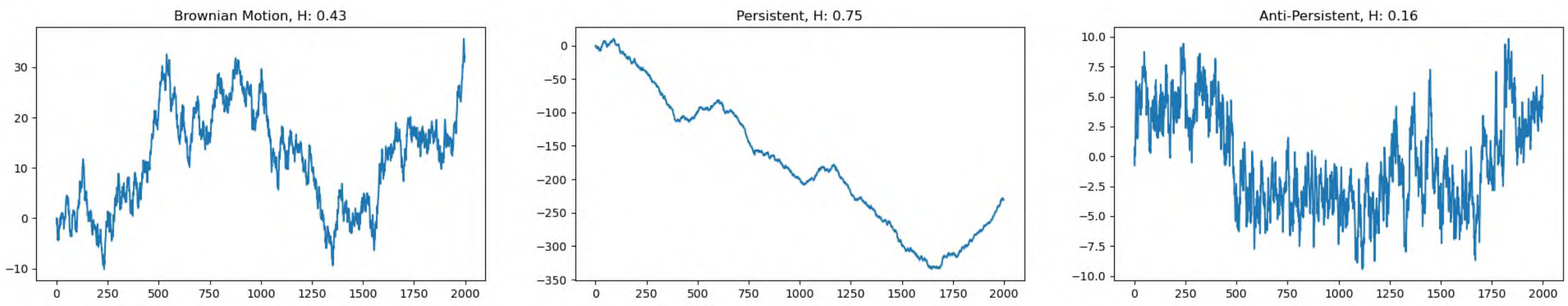
```
def random_walk_memory(length, proba, min_lookback, max_lookback)
```

- `proba` is the probability that the next increment will follow the trend
  - proba > 0.5 persistent random walk
  - proba < 0.5 antipersistent one
  - min_lookback and max_lookback are the minimum and maximum window sizes to calculate trend direction

```
def random_walk_memory(length, proba=0.5, min_lookback=1, max_lookback=100):
    series = [0.] * length
    for i in range(1, length):
        # If the series has not yet reached the min_lookback threshold
        # the direction of the step is random (-1 or 1)
        if i < min_lookback + 1:
            direction = np.sign(np.random.randn())
        # consider the historical values to determine the direction
        else:
            # randomly choose between min_lookback and the minimum of
            # i-1 (to ensure not exceeding the current length) and max_lookback.
            lookback = np.random.randint(min_lookback, min(i-1, max_lookback)+1)
            # Decides whether to follow the recent trend or move against it,
            # based on a comparison between proba and a random number between 0 and 1.
            recent_trend = np.sign(series[i-1] - series[i-1-lookback])
            change = np.sign(proba - np.random.uniform())
            direction = recent_trend * change
        series[i] = series[i-1] + np.fabs(np.random.randn()) * direction
    return series
```

PERSISTENT AND ANTI-PERSISTENT TIME SERIES

```
bm = random_walk_memory(2000, proba=0.5)
persistent = random_walk_memory(2000, proba=0.7)
antipersistent = random_walk_memory(2000, proba=0.3)

_, axes = plt.subplots(1,3, figsize=(24, 4))
axes[0].plot(bm)
axes[0].set_title(f"Brownian Motion, H: {hurst(bm):.2f}")
axes[1].plot(persistent)
axes[1].set_title(f"Persistent, H: {hurst(persistent):.2f}")
axes[2].plot(antipersistent)
axes[2].set_title(f"Anti-Persistent, H: {hurst(antipersistent):.2f}");
```




```
print(f"GOOG closing price, H: {hurst(data['Close'].values):.2f}")
```

GOOG closing price, H: 0.41

Πανεπιστήμιο Κύπρου

- The Hurst exponent (H) is a critical metric in the analysis of time series (e.g., financial data)
- Offers insights into the behavior of data, such as stocks
- Here's how to interpret H in the context of closing stock prices and its influence on investment decisions:

Case 1: $H = 0.5$

- Data follows a geometric Brownian motion, i.e., a completely random walk
- Implies that future price movements are independent of past movements
- There are no autocorrelations in price movements to exploit; past data cannot predict future prices

Πανεπιστήμιο Κύπρου

Case 2: $H < 0.5$

- Indicates a mean-reverting series, i.e., data tends to revert to its historical average
- This suggests that the asset is less risky over the long term
- Investors might interpret a low H as an opportunity to buy stocks after a significant drop, expecting a reversion to the mean, or to sell after a substantial rise

Case 3: $H > 0.5$

- Suggests a trending series, where increases or decreases in data are likely to be followed by further increases or decreases, respectively
- This persistence indicates potential momentum in data, which can be exploited by momentum strategies:
- Buying stocks that have been going up in the hope that they will continue to do so, and selling those in a downtrend

Hurst exponent Practical considerations

- The observed value of H can vary over different time frames: analyze H over the period relevant to the prediction horizon
- External factors (such as geopolitical events, environment) can influence data and should be considered alongside H
- If the time series is too short, the value of H might not be reliable

Πανεπιστήμιο Κύπρου

Geometric Brownian Motion

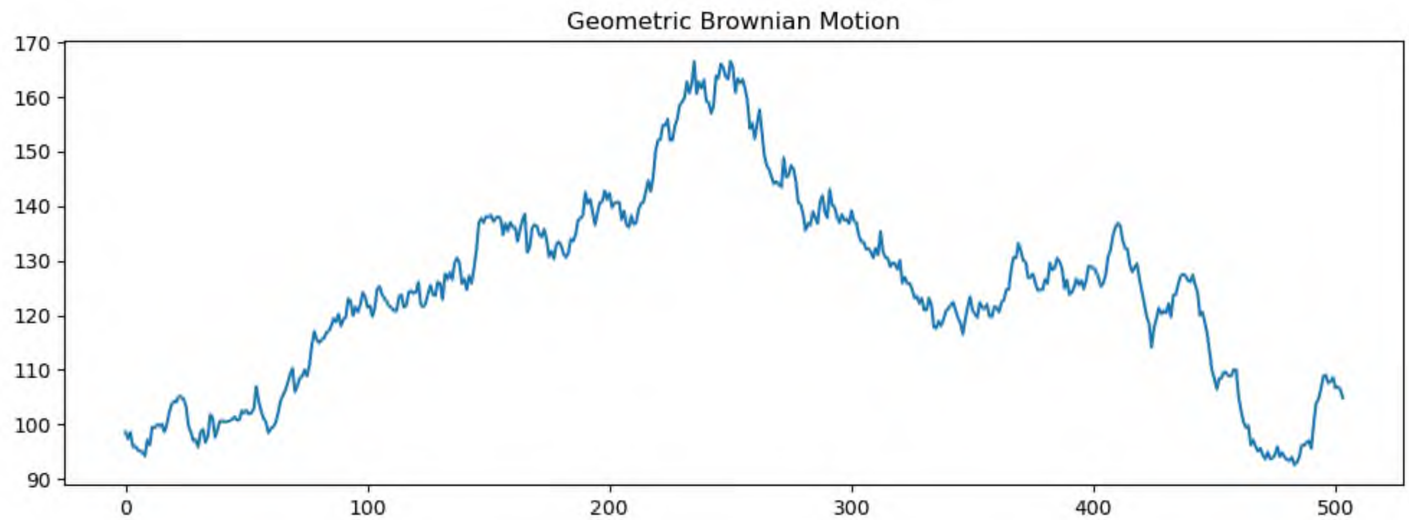
- The Geometric Brownian Motion (GBM) is a stochastic process
- It is often used to model stock prices and other financial variables that do not revert to a mean but rather exhibit trends with a drift μ and volatility σ
- The GBM is defined as:

$$S(t) = S_0 \exp\left(\left(\mu - \frac{1}{2}\sigma^2\right) + \sigma W(t)\right)$$

- Where
  - S(t) is the data at time t
  - S_0 initial value at t=0
  - μ is the expected value (drift coefficient)
  - σ volatility (standard deviation)
  - W(t) is a Wiener process (standard Brownian montion)

Πανεπιστήμιο Κύπρου

```
S0 = 100                    # Initial stock price
mu = 0.09                   # Expected annual return (9%)
sigma = 0.25                # Annual volatility (25%)
T = 2                       # Time horizon in years
dt = 1/252                  # Time step in years, assuming 252 trading days per year
N = int(T/dt)               # Number of time steps
t = np.linspace(0, T, N)    # Time vector

# Brownian Motion
dW = np.random.normal(0, np.sqrt(dt), N)
W = np.cumsum(dW)
# Geometric Brownian Motion
S = S0 * np.exp((mu - 0.5 * sigma**2) * t + sigma * W)

# Plotting the Geometric Brownian Motion
plt.figure(figsize=(12, 4))
plt.plot(S)
plt.title('Geometric Brownian Motion');
```



- $S(t)$ is *log-normally distributed* because it's an exponential function of a normally distributed process $\mathrm{B}(t)$
- Variance of $S(t)$ can be found from the properties of the log-normal distribution:

$$Var\big(S(t)\big) = \left(e^{\sigma^2 t} - 1\right) e^{2\mu t+\sigma^2 t} S_0^2$$

- Variance of GBM is not linear in t like the BM
- Instead, it grows exponentially with time due to the exponential term $e^{\sigma^2 t}$
- This, and the possibility of modelling drift (expected annual return) are the main additions of GBM over BM

- GBM can be used to model real stock prices and simulate their future behavior.

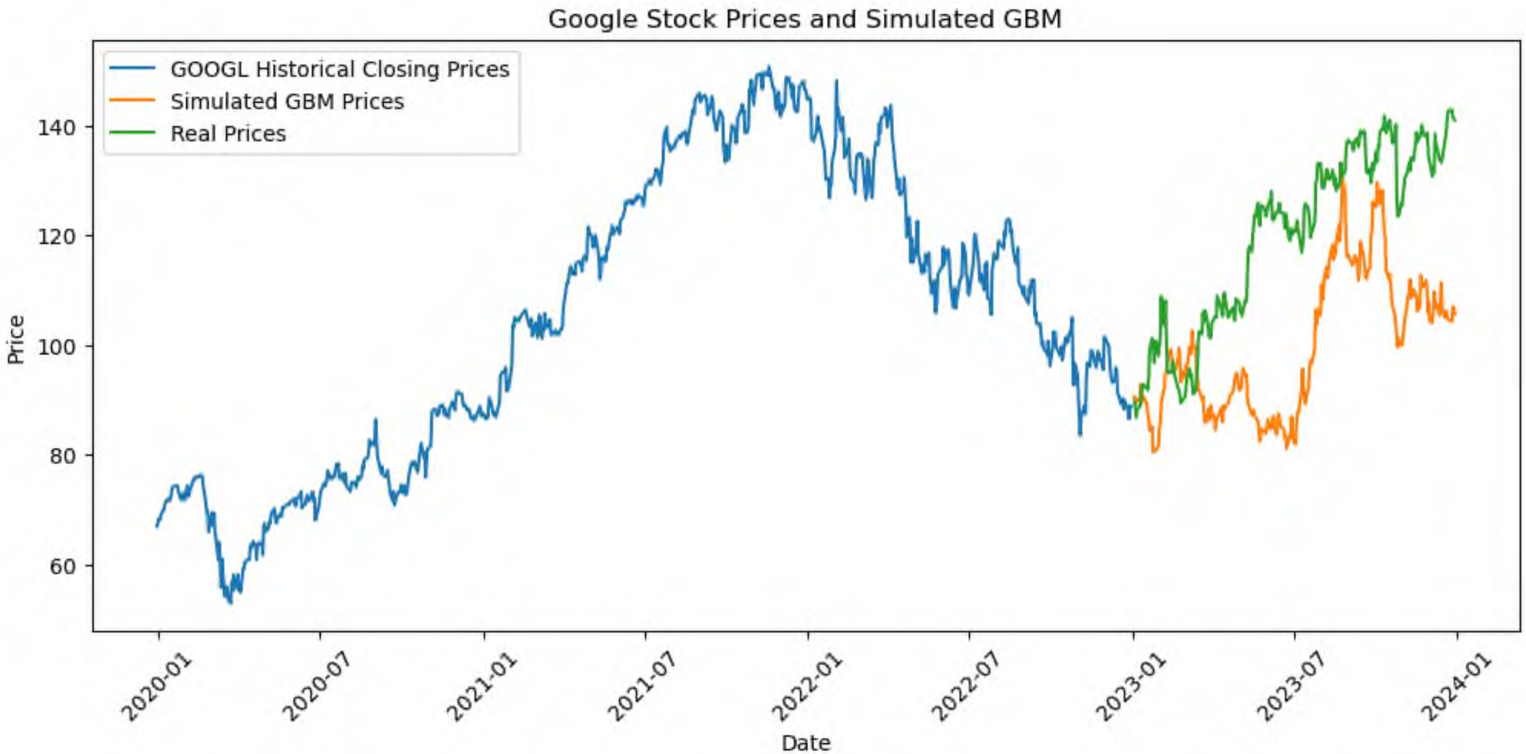
1. First, we estimate μ and σ from historical stock price data.
   - μ could be the historical average of the stock's logarithmic returns
   - σ could be the standard deviation of those returns
2. Then, we use these estimates in the GBM formula to simulate future price paths.

- This method is widely used for option pricing, risk management, and investment strategy simulations
- However, GBM has limitations, such as assuming a constant drift and volatility
- These assumptions may not hold true in real markets
- Therefore, it's often used as a component of a broader analysis or modeling strategy

```
# Step 1: Get the "training" data (e.g., 2020-2022)
data2 = get_data('GOOG', period='1d', start='2019-12-31', end='2022-12-31')

# Get "test" data, for comparison (e.g., 2023)
data3 = get_data('GOOG', period='1d', start='2022-12-31', end='2023-12-31')
test_days = len(data3)

# Step 2: Calculate Daily Returns
returns = data2['Close'].pct_change() # Interested in the returns, so we get the
changes in %

# Step 3: Estimate Parameters for GBM
mu = returns.mean() * 252  # Annualize the mean
sigma = returns.std() * np.sqrt(252)  # Annualize the std deviation

# Step 4: Set GBM parameters
T = 1  # Time horizon in years
dt = 1/test_days  # Time step in years, assuming 252 trading days per year
N = int(T/dt)  # Number of time steps
time_step = np.linspace(0, T, N)
S0 = data2['Close'].iloc[-1]  # Starting stock price (latest close price)

#  Step 5: Compute Simulation
W = np.random.standard_normal(size=N)
W = np.cumsum(W)*np.sqrt(dt)  # Cumulative sum for the Wiener process
X = (mu - 0.5 * sigma**2) * time_step + sigma * W
S = S0 * np.exp(X)  # GBM formula
```

Πανεπιστήμιο Κύπρου

```
# Plot the results
plt.figure(figsize=(12, 5))
plt.plot(data2['Close'], label='GOOGL Historical Closing Prices')
plt.plot(data3.index, S, label='Simulated GBM Prices')
plt.plot(data3['Close'], label='Real Prices')
plt.legend()
plt.title('Google Stock Prices and Simulated GBM')
plt.xlabel('Date')
plt.ylabel('Price')
plt.xticks(rotation=45);
```



There is stochastic component in GBM hence run monte carlo

```
# Simulate multiple paths
n_paths = 10
paths = []
for _ in range(n_paths):
    W = np.cumsum(np.random.standard_normal(size=N))*np.sqrt(dt)
    X = (mu - 0.5 * sigma**2) * time_step + sigma * W
    paths.append(S0 * np.exp(X))

path_mean = np.array(paths).mean(axis=0)
path_std = np.array(paths).std(axis=0)
```

```
plt.figure(figsize=(12, 5))
plt.plot(data2['Close'], label='GOOGL Historical Closing Prices')
plt.plot(data3.index, path_mean, label='Simulated GBM Prices')
plt.fill_between(data3.index, path_mean-1.96*path_std, path_mean+1.96*path_std,
color='tab:orange', alpha=0.2, label='95% CI')
plt.plot(data3['Close'], label='Real Prices')
plt.legend(loc='upper left')
plt.title('Google Stock Prices and Simulated GBM')
plt.xlabel('Date')
plt.ylabel('Price')
plt.xticks(rotation=45);
```