# Exercise

**AR:** Auto-Regressive, **MA:** Moving Average

In [30]:
```
# Imports
import warnings
from statsmodels.tools.sm_exceptions import ConvergenceWarning
warnings.simplefilter('ignore', ConvergenceWarning)
import requests
import numpy as np
from io import BytesIO
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.stattools import ccf, adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.arima_process import arma_generate_sample
from statsmodels.tsa.holtwinters import ExponentialSmoothing
np.random.seed(0) # For reproducibility
```

In [9]:
```
def ts_plot(x,y,title,xlabel="Time",ylabel="Value"):
    plt.figure(figsize=(10,5))
    plt.plot(x, y)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.grid(alpha=0.3);
```

In [10]:
```
def acf_pacf_plot(ts,lags=30):
    fig, ax = plt.subplots(1,2, figsize=(15, 5))
    plot_acf(ts, lags=lags, ax=ax[0])
    plot_pacf(ts, lags=lags, ax=ax[1])
    plt.show()
```

In [11]:
```
def mse(observations, estimates):
    difference = observations - estimates
    sq_diff = difference ** 2
    mse = np.mean(sq_diff)
    return mse
```

# Exercise

- Load the two time series `arma_ts1` by running the code below.

In [12]:
```
# Load the first time series
response = requests.get("https://zenodo.org/records/10951538/files/arma_ts3.npz?download=1")
response.raise_for_status()
ts = np.load(BytesIO(response.content))['signal']
print(len(ts))
```

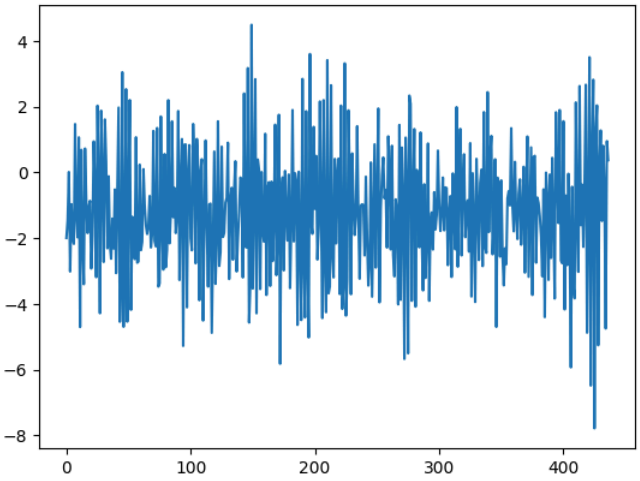
```
479
```

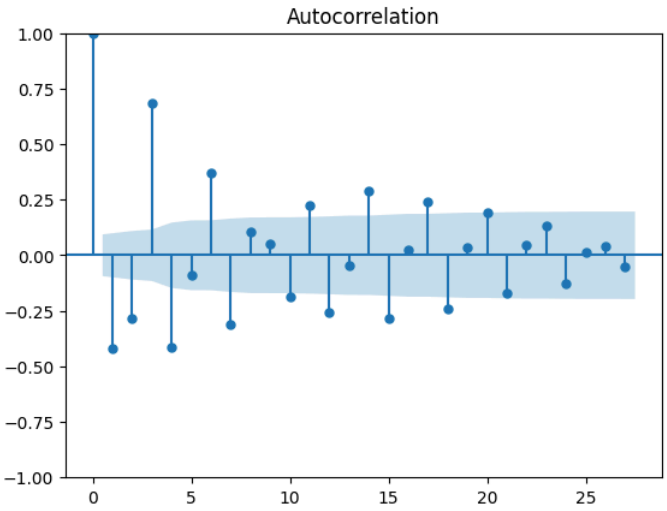
For each time series

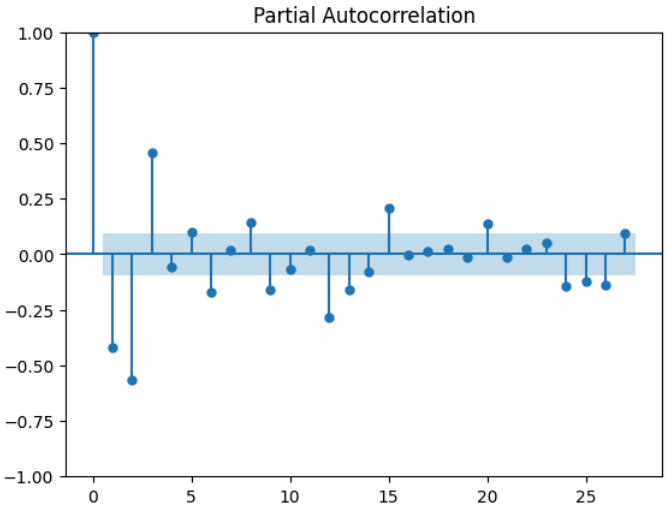
1. Split the time series in train and test.
   - Use the last 30 values as test for the first time series
   - Use the last 100 as test for the second time series.
2. Make the time series stationary.
3. Determine the order $p$ of an AR model.
4. Compute the prediction of the test data with the AR($p$) model.
5. Determine the order $q$ of a MA model.
6. Compute the prediction of the test data with the MA($q$) model.

In [13]:
```
time = range(len(ts))
```

In [14]:
```
train = ts[:-30]
test = ts[-30:]
s = 12
ts_diff = np.diff(ts)
ts_sdiff = ts[s:] - ts[:-s]
train_diff = ts_diff[:-30] #np.diff(train)
train_sdiff = ts_sdiff[:-30] #train[s:] - train[:-s]
test_diff = ts_diff[-30:]
test_sdiff = ts_sdiff[-30:]
# import pandas as pd
# train_sdiff = pd.DataFrame(train).diff(12).dropna()
# fig,ax=plt.subplots((1,3))
plt.plot(train_sdiff);
plot_acf(train_sdiff);
plot_pacf(train_sdiff);
```




Implementing ADF test

In [15]:
```
adfuller(train_sdiff)
```

Out[15]:
```
(-6.63876415493461,
 5.477142531888239e-09,
 14,
 422,
 {'1%': -3.44594128742536,
  '5%': -2.868413360220551,
  '10%': -2.570431271085555},
 1343.7496673604292)
```

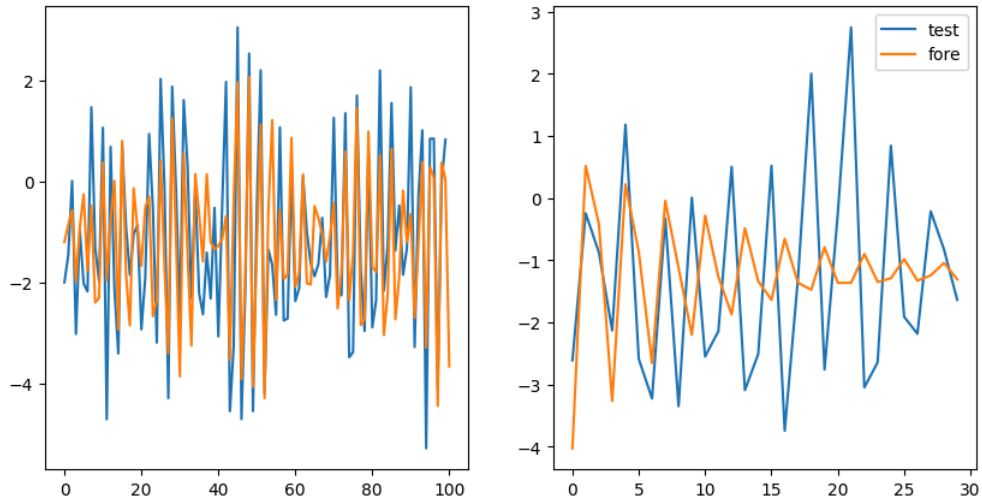
we see that `sdiff` is stationary

In [16]:
```
model_ARMA = ARIMA(train_sdiff, order=(3,0,0))
model_ARMA_fit = model_ARMA.fit() # Fit the model

fig, axs = plt.subplots(1,2,figsize=(10,5))
axs[0].plot(train_sdiff[:100],label='train')
axs[0].plot(model_ARMA_fit.predict(0,100),label='pred')
axs[1].plot(test_sdiff,label='test')
```

```
axs[1].plot(model_ARMA_fit.forecast(len(test_sdiff)),label='fore')
plt.legend()
```

Out[16]: <matplotlib.legend.Legend at 0x1bfdd19ce90>




## Auto-ARIMA

In [17]:
```
from pmdarima import auto_arima
```

In [18]:
```
auto_arima(train_sdiff).summary()#,seasonal=True,m=7)
```

Out[18]:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 437 |
| **Model:** | SARIMAX(3, 0, 5) | **Log Likelihood** | -724.221 |
| **Date:** | Wed, 17 Dec 2025 | **AIC** | 1468.442 |
| **Time:** | 13:24:06 | **BIC** | 1509.241 |
| **Sample:** | 0 | **HQIC** | 1484.542 |
| | - 437 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **intercept** | -0.4493 | 0.141 | -3.193 | 0.001 | -0.725 | -0.174 |
| **ar.L1** | -0.3248 | 0.051 | -6.317 | 0.000 | -0.426 | -0.224 |
| **ar.L2** | 0.2419 | 0.054 | 4.454 | 0.000 | 0.135 | 0.348 |
| **ar.L3** | 0.7090 | 0.046 | 15.276 | 0.000 | 0.618 | 0.800 |
| **ma.L1** | -0.0166 | 0.062 | -0.267 | 0.790 | -0.138 | 0.105 |
| **ma.L2** | -0.5543 | 0.061 | -9.054 | 0.000 | -0.674 | -0.434 |
| **ma.L3** | -0.0984 | 0.067 | -1.474 | 0.141 | -0.229 | 0.032 |
| **ma.L4** | 0.1608 | 0.052 | 3.093 | 0.002 | 0.059 | 0.263 |
| **ma.L5** | -0.4500 | 0.048 | -9.302 | 0.000 | -0.545 | -0.355 |
| **sigma2** | 1.6336 | 0.125 | 13.058 | 0.000 | 1.388 | 1.879 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.34 | **Jarque-Bera (JB):** | 0.85 |
| **Prob(Q):** | 0.56 | **Prob(JB):** | 0.65 |
| **Heteroskedasticity (H):** | 1.23 | **Skew:** | -0.06 |
| **Prob(H) (two-sided):** | 0.21 | **Kurtosis:** | 2.83 |

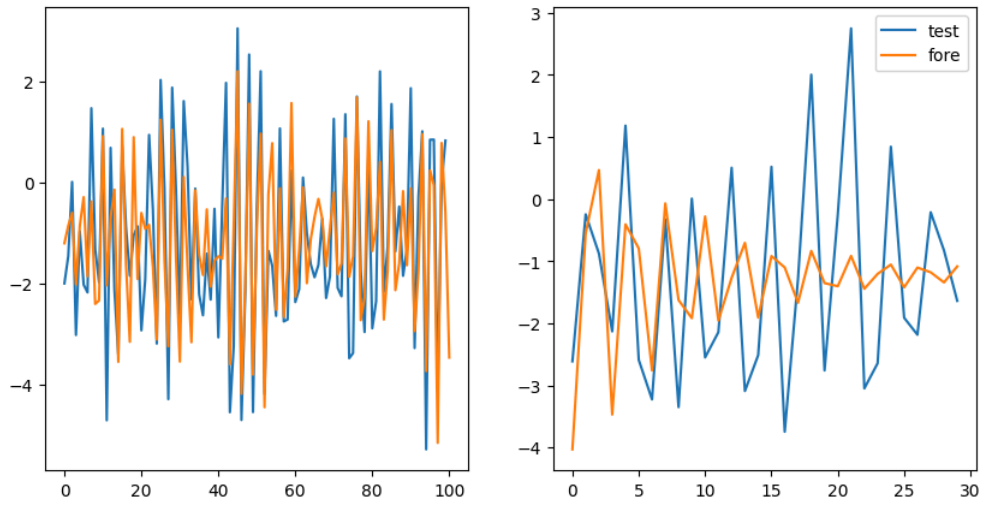
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Note that auto-arima considers in-sample performance

In [45]:
```
model_ARMA_auto = ARIMA(train_sdiff, order=(3,0,5))
model_ARMA_auto_fit = model_ARMA_auto.fit() # Fit the model

fig, axs = plt.subplots(1,2,figsize=(10,5))
axs[0].plot(train_sdiff[:100],label='train')
axs[0].plot(model_ARMA_auto_fit.predict(0,100),label='pred')
axs[1].plot(test_sdiff,label='test')
axs[1].plot(model_ARMA_auto_fit.forecast(len(test_sdiff)),label='fore')
plt.legend()
```

Out[45]: <matplotlib.legend.Legend at 0x1bfdcb0bef0>




## Grid Search

In [20]:
```
%%time
best_RMSE = np.inf
for p in range(1,4):
    for d in [0]:
        for q in range(1,4):
            try:
                model = ARIMA(train_sdiff, order=(p,d,q))  # ARIMA with d=0 is equivalent to ARMA
                model_ARMA_grid_fit = model.fit()
                forecast_sdiff = model_ARMA_grid_fit.forecast(len(test_sdiff))
            except:
                pass
            else:
                MAE = abs((test_sdiff-forecast_sdiff)).mean() # not suitable for data containing 0s
                RMSE = mse(test_sdiff, forecast_sdiff)**0.5
                AIC = model_ARMA_grid_fit.aic
                BIC = model_ARMA_grid_fit.bic
                if RMSE < best_RMSE:
                    best_model = (p,d,q)
                    best_RMSE = RMSE
                print('ARIMA(%.0f,%.0f,%.0f) -> MAE:%.3f, RMSE:%.3f, AIC:%.1f, BIC:%.1f' % (p,d,q, MAE, RMSE, AIC, BIC))
print('Best model by RMSE (Forecast):',best_model)
```

```
ARIMA(1,0,1) -> MAE:1.412, RMSE:1.655, AIC:1719.6, BIC:1735.9
ARIMA(1,0,2) -> MAE:1.414, RMSE:1.660, AIC:1656.3, BIC:1676.7
C:\Users\serhan.aydin\Documents\my_docs\venv\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using zeros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
ARIMA(1,0,3) -> MAE:1.400, RMSE:1.651, AIC:1574.0, BIC:1598.5
ARIMA(2,0,1) -> MAE:1.338, RMSE:1.601, AIC:1511.4, BIC:1531.8
ARIMA(2,0,2) -> MAE:1.347, RMSE:1.612, AIC:1505.4, BIC:1529.9
ARIMA(2,0,3) -> MAE:1.495, RMSE:1.750, AIC:1492.0, BIC:1520.6
ARIMA(3,0,1) -> MAE:1.437, RMSE:1.719, AIC:1499.8, BIC:1524.3
ARIMA(3,0,2) -> MAE:1.408, RMSE:1.677, AIC:1497.8, BIC:1526.4
ARIMA(3,0,3) -> MAE:1.484, RMSE:1.743, AIC:1493.1, BIC:1525.7
Best model by RMSE (Forecast): (2, 0, 1)
CPU times: total: 1.17 s
Wall time: 1.25 s
```

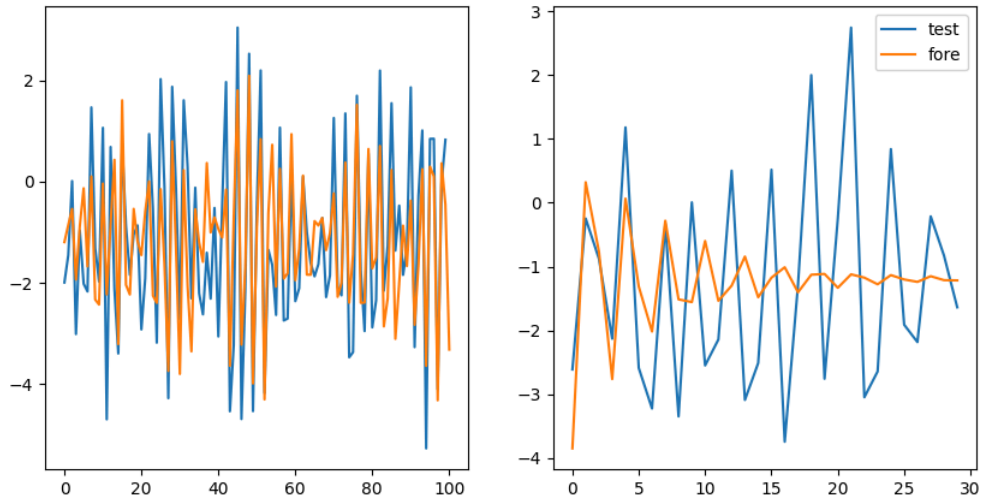
Do not use MAPE on:

- Differenced data
- Seasonally differenced data
- Log-returns
- Residual-like series

In [21]:
```
model_ARMA_grid = ARIMA(train_sdiff, order=best_model)
model_ARMA_grid_fit = model_ARMA_grid.fit() # Fit the model

fig, axs = plt.subplots(1,2,figsize=(10,5))
axs[0].plot(train_sdiff[:100],label='train')
axs[0].plot(model_ARMA_grid_fit.predict(0,100),label='pred')
axs[1].plot(test_sdiff,label='test')
axs[1].plot(model_ARMA_grid_fit.forecast(len(test_sdiff)),label='fore')
plt.legend()
```

Out[21]:
```
<matplotlib.legend.Legend at 0x1bfe047e210>
```




In [46]:
```
from statsmodels.tools.eval_measures import mse
print('MSE train (model_ARMA_fit) %s:' % str(model_ARMA.order), mse(model_ARMA_fit.predict(),train_sdiff))
print('MSE train (model_ARMA_auto_fit) %s:' % str(model_ARMA_auto.order), mse(model_ARMA_auto_fit.predict(),train_sdiff))
print('MSE train (model_ARMA_grid_fit) %s:' % str(model_ARMA_grid.order), mse(model_ARMA_grid_fit.predict(),train_sdiff))
print('\n')
print('MSE test (model_ARMA_fit) %s:' % str(model_ARMA.order), mse(model_ARMA_fit.forecast(len(test_sdiff)),test_sdiff))
print('MSE test (model_ARMA_auto_fit) %s:' % str(model_ARMA_auto.order), mse(model_ARMA_auto_fit.forecast(len(test_sdiff)),test_sdiff))
print('MSE test (model_ARMA_grid_fit) %s:' % str(model_ARMA_grid.order), mse(model_ARMA_grid_fit.forecast(len(test_sdiff)),test_sdiff))
```

```
MSE train (model_ARMA_fit) (3, 0, 0): 1.7685451058780566
MSE train (model_ARMA_auto_fit) (3, 0, 5): 1.5948204662820704
MSE train (model_ARMA_grid_fit) (2, 0, 1): 1.8128786923346902


MSE test (model_ARMA_fit) (3, 0, 0): 3.3326137214316542
MSE test (model_ARMA_auto_fit) (3, 0, 5): 2.5988643812573473
MSE test (model_ARMA_grid_fit) (2, 0, 1): 2.563046523485501
```

Discussion on the results:

- Auto-arima is based on in-sample
- This can explain why it is best in-sample but 2nd best in out-of-sample
- Visual check resulted in a relatively poorer performance

Reintegrate original series

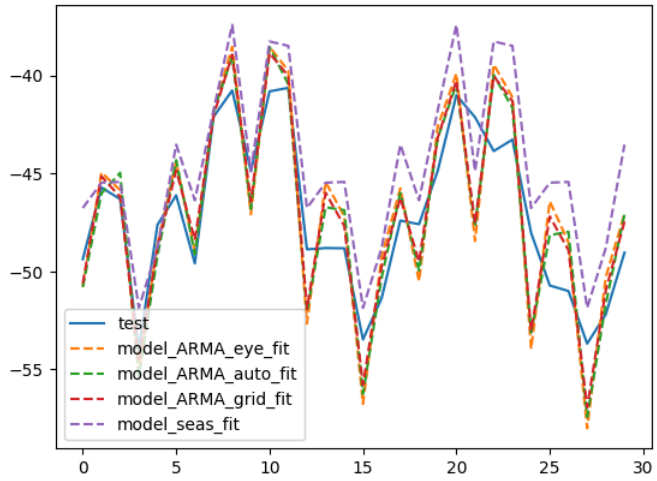
In [50]:
```
# Reversing the seasonal differencing
test_fore_ARMA = np.empty(len(test))
test_fore_ARMA_auto = np.empty(len(test))
test_fore_ARMA_grid = np.empty(len(test))
test_fore_seas = np.empty(len(test))

test_fore_ARMA[:12] = train[-12:] + model_ARMA_fit.forecast(len(test_sdiff))[:12]
test_fore_ARMA_auto[:12] = train[-12:] + model_ARMA_auto_fit.forecast(len(test_sdiff))[:12]
test_fore_ARMA_grid[:12] = train[-12:] + model_ARMA_grid_fit.forecast(len(test_sdiff))[:12]
test_fore_seas[:12] = train[-12:]

for i in range(12, len(test)):
    test_fore_seas[i] = test_fore_seas[i-12]
    test_fore_ARMA[i] = test_fore_ARMA[i-12] + model_ARMA_fit.forecast(len(test_sdiff))[i]
    test_fore_ARMA_auto[i] = test_fore_ARMA_auto[i-12] + model_ARMA_auto_fit.forecast(len(test_sdiff))[i]
    test_fore_ARMA_grid[i] = test_fore_ARMA_grid[i-12] + model_ARMA_grid_fit.forecast(len(test_sdiff))[i]

plt.plot(test, label='test')
plt.plot(test_fore_ARMA,'--', label='model_ARMA_eye_fit')
plt.plot(test_fore_ARMA_auto,'--', label='model_ARMA_auto_fit')
plt.plot(test_fore_ARMA_grid,'--', label='model_ARMA_grid_fit')
plt.plot(test_fore_seas,'--', label='model_seas_fit')
plt.legend()
```

Out[50]:
```
<matplotlib.legend.Legend at 0x1bfe64e67e0>
```




In [25]:
```
from statsmodels.tools.eval_measures import mse
print('MSE test (model_ARMA_fit) %s:' % str(model_ARMA.order), mse(test_fore_ARMA,test))
print('MSE test (model_ARMA_auto_fit) %s:' % str(model_ARMA_auto.order), mse(test_fore_ARMA_auto,test))
print('MSE test (model_ARMA_grid_fit) %s:' % str(model_ARMA_grid.order), mse(test_fore_ARMA_grid,test))
```

```
MSE test (model_ARMA_fit) (3, 0, 0): 7.853957943219815
MSE test (model_ARMA_autoarima_fit) (3, 0, 5): 5.672869974333564
MSE test (model_ARMA_grid_fit) (2, 0, 1): 5.211031373991924
```

**Alternatively we can use SARIMA fit/forecasting. But we have to tell the model how we achieved stationarity.**

In [52]:
```
model_SARIMA_grid_fit = SARIMAX(train, order=(2,0,1), seasonal_order=(0,1,0,12)).fit()
model_SARIMA_auto_fit = SARIMAX(train, order=(5,0,2), seasonal_order=(0,1,0,12)).fit()
model_SARIMA_eye_fit = SARIMAX(train, order=(3,0,0), seasonal_order=(0,1,0,12)).fit()

plt.plot(test, label='test')
plt.plot(model_SARIMA_grid_fit.forecast(len(test)),'--', label='model_SARIMA_grid_fore')
plt.plot(model_SARIMA_auto_fit.forecast(len(test)),'--', label='model_SARIMA_auto_fore')
plt.plot(model_SARIMA_eye_fit.forecast(len(test)),'--', label='model_SARIMA_eye_fore')
# plt.plot(test_fore_seas,'--', label='model_seas_fit')
plt.legend()
```

Out[52]: <matplotlib.legend.Legend at 0x1bfe65b6cc0>