In [1]:
```
import os

# Specify the directory path you want to set as the working directory
directory_path = "/Users/macbookpro/Desktop"

# Change the working directory
os.chdir(directory_path)

# Verify that the working directory has been changed
print("Working directory has been set to:", directory_path)
```

```
Working directory has been set to: /Users/macbookpro/Desktop
```

## Preliminaries

In [2]:
```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(123)
```

In [3]:
```
# Parameters
T = 100  # Number of observations
K = 2    # Number of variables
max_lag = 5  # Maximum lag length to consider
```

In [4]:
```
# Define different combinations of AR forms
ar_forms = [
    {"intercept": False, "trend": False},
    {"intercept": True, "trend": False},
    {"intercept": False, "trend": True},
    {"intercept": True, "trend": True}
]

# Define different levels of cross-correlation
cross_correlation_values = [0, 0.3, 0.6, 0.9]
```

In [5]:
```
# Coefficients for the VAR equations (AR(1))
coefficients = np.array([[0.6, -0.4],   # Coefficients for lagged y1
                         [0.3, 0.8]])   # Coefficients for lagged y2
```

## DATA GENERATING PROCESS for Reduced Form VAR(1) (for differing levels of cross-correlation (or covariance) and AR forms.)

In [6]:
```
# Loop over different cross-correlation values
for correlation in cross_correlation_values:
    print(f"Cross-correlation: {correlation}")

    # Generate covariance matrix with the specified cross-correlation
    covariance_matrix = np.array([[1.0, correlation],
                                  [correlation, 1.0]])

    # Generate VAR errors with cross-correlation
    mean = np.zeros(K)
    errors = np.random.multivariate_normal(mean, covariance_matrix, size=T)

    # Compute correlation between the two error series in 'errors'
    correlation_errors = np.corrcoef(errors[:, 0], errors[:, 1])[0, 1]
    print("Correlation between error series in 'errors':", correlation_errors)

    # Loop over different AR forms
    for ar_form in ar_forms:
        intercept = ar_form["intercept"]
        trend = ar_form["trend"]
        print(f"AR Form: Intercept={intercept}, Trend={trend}")

        # Simulate SVAR data for p=1
        data = np.zeros((T, K))
        for t in range(1, T):
            lagged_values = data[t-1:t, :]  # Lagged values of the variables

            # Compute the SVAR equation for each variable
            for i in range(K):
                lagged_product = np.dot(lagged_values.flatten(), coefficients[:, i])  # Compute lagg
                intercept_term = 21 if intercept else 0
                trend_term = 0.01 * t if trend else 0
                data[t, i] = lagged_product + intercept_term + trend_term + errors[t, i]

        # Compute AIC for p = 1, 2, 3, 4, 5
        aic_values = []
        for p in range(1, max_lag + 1):
            model = sm.tsa.VAR(data)
            results = model.fit(p)
            aic_values.append(results.aic)

        # Print AIC values for different lag lengths
        print("AIC values:", aic_values)
```

```
AR Form: Intercept=True, Trend=True
AIC values: [-0.17300815846891499, -0.12772568601342174, -0.1716367597420776, -0.19107567520379
143, -0.14842064006369105]
Cross-correlation: 0.6
Correlation between error series in 'errors': 0.6756636791594098
AR Form: Intercept=False, Trend=False
AIC values: [-0.36591334270468434, -0.3011680957907281, -0.24728412125014365, -0.21462414760633
775, -0.17874568606314079]
AR Form: Intercept=True, Trend=False
AIC values: [-0.3317295315591752, -0.2830108161443323, -0.23900664507921932, -0.209506496677946
74, -0.19902298415765785]
AR Form: Intercept=False, Trend=True
AIC values: [-0.36187665965151716, -0.2929597737765248, -0.2382951920009987, -0.183350664303359
1, -0.1531277391683442]
AR Form: Intercept=True, Trend=True
AIC values: [-0.30960667773363765, -0.2753297900432685, -0.2507846526841164, -0.216999090367671
5, -0.20835181837390948]
Cross-correlation: 0.9
Correlation between error series in 'errors': 0.905911291700551
AR Form: Intercept=False, Trend=False
```

## DATA GENERATING PROCESS for ARFVAR(1) (for differing levels of cross-correlation (or covariance) and AR forms.)

In [7]:
```python
# Loop over different cross-correlation values
for correlation in cross_correlation_values:
    print(f"Cross-correlation: {correlation}")

    # Generate covariance matrix with the specified cross-correlation
    covariance_matrix = np.array([[1.0, correlation],
                                   [correlation, 1.0]])

    # Generate VAR errors with cross-correlation
    mean = np.zeros(K)
    errors = np.random.multivariate_normal(mean, covariance_matrix, size=T)

    # Perform Cholesky decomposition
    chol_decomp = np.linalg.cholesky(covariance_matrix)

    # Print the Lower Triangular Matrix B
    print("Lower Triangular Matrix B:")
    print(np.array2string(chol_decomp, separator=', ', formatter={'float_kind':lambda x: "%.2f" % x}

    # Transform errors to make them orthogonal
    orthogonal_errors = np.dot(errors, np.linalg.inv(chol_decomp.T))

    # Compute correlation between the two error series in 'orthogonal_errors'
    correlation_orthogonal_errors = np.corrcoef(orthogonal_errors[:, 0], orthogonal_errors[:, 1])[0
    print("Correlation between error series in 'orthogonal_errors':", correlation_orthogonal_errors

    # Loop over different AR forms
    for ar_form in ar_forms:
        intercept = ar_form["intercept"]
        trend = ar_form["trend"]
        print(f"AR Form: Intercept={intercept}, Trend={trend}")

        # Simulate SVAR data for p=1
        data = np.zeros((T, K))
        for t in range(1, T):
            lagged_values = data[t-1:t, :]  # Lagged values of the variables

            # Compute the SVAR equation for each variable
            for i in range(K):
                lagged_product = np.dot(lagged_values.flatten(), coefficients[:, i])  # Compute lag
                intercept_term = 21 if intercept else 0
                trend_term = 0.01 * t if trend else 0
                data[t, i] = lagged_product + intercept_term + trend_term + orthogonal_errors[t, i]

        # Compute AIC for p = 1, 2, 3, 4, 5
        aic_values = []
        for p in range(1, max_lag + 1):
            model = sm.tsa.VAR(data)
            results = model.fit(p)
            aic_values.append(results.aic)

        # Print AIC values for different lag lengths
        print("AIC values:", aic_values)
```

```
6, 0.1841622262105414]
AR Form: Intercept=True, Trend=True
AIC values: [-0.008146151723012474, 0.06567177289503887, 0.13453692632443937, 0.162431849346599
65, 0.17790347406191553]
Cross-correlation: 0.3
Lower Triangular Matrix B:
[[1.00, 0.00],
 [0.30, 0.95]]
Correlation between error series in 'orthogonal_errors': -0.014257675944483206
AR Form: Intercept=False, Trend=False
AIC values: [-0.04116358364177314, 0.022418412558089656, 0.06144349808267657, 0.076358113129092
13, 0.0744039569397652]
AR Form: Intercept=True, Trend=False
AIC values: [-0.06230186907658289, 0.005563750505523601, 0.02230143646660593, 0.072982481123259
89, 0.08441226916336653]
AR Form: Intercept=False, Trend=True
AIC values: [0.12940287418834798, 0.12695397446292275, 0.16083181950675265, 0.163983462884756 8,
0.19247397828632618]
AR Form: Intercept=True, Trend=True
```

In [8]:
```
# Print the Lower Triangular Matrix B
print("Lower Triangular Matrix B:")
print(np.array2string(chol_decomp, separator=', ', formatter={'float_kind':lambda x: "%.2f" % x}))
```

```
Lower Triangular Matrix B:
[[1.00, 0.00],
 [0.90, 0.44]]
```

## IRFs of RFVAR (Impulse Responses for each series y1t and y2t)

In [11]:
```
from matplotlib.backends.backend_pdf import PdfPages

# Create a PDF file to store all plots
pdf_filename = "RFVAR_IRF_plots.pdf"
pdf_pages = PdfPages(pdf_filename)

# Loop over different cross-correlation values
for correlation in cross_correlation_values:
    print(f"Cross-correlation: {correlation}")

    # Loop over different AR forms
    for ar_form in ar_forms:
        intercept = ar_form["intercept"]
        trend = ar_form["trend"]
        print(f"AR Form: Intercept={intercept}, Trend={trend}")

        # Generate covariance matrix with the specified cross-correlation
        covariance_matrix = np.array([[1.0, correlation],
                                      [correlation, 1.0]])

        # Generate VAR errors with cross-correlation
        mean = np.zeros(K)
        errors = np.random.multivariate_normal(mean, covariance_matrix, size=T)

        # Simulate VAR data for p=1
        data = np.zeros((T, K))
        for t in range(1, T):
            lagged_values = data[t-1:t, :]  # Lagged values of the variables

            # Compute the VAR equation for each variable
            for i in range(K):
                lagged_product = np.dot(lagged_values.flatten(), coefficients[:, i])  # Compute lagg
                intercept_term = 21 if intercept else 0
                trend_term = 0.01 * t if trend else 0
                data[t, i] = lagged_product + intercept_term + trend_term + errors[t, i]

        # Fit VAR model for p = 1
        model = sm.tsa.VAR(data)
        results = model.fit(1)

        # Compute impulse response functions
        irf = results.irf(10)  # Compute IRFs for 10 periods
        fig = irf.plot(orth=False)   # Plot IRFs without orthogonalization

        # Set x-axis ticks for all subplots
        for i in range(len(fig.axes)):
            fig.axes[i].set_xticks(np.arange(0, 11, 2))

        # Set a customized title for all plots
        fig.suptitle(f"Impulse Response for Cross-correlation: {correlation}, Intercept: {intercept}

        # Update plot titles to denote the shock/error term
        for ax in fig.get_axes():
            title = ax.get_title()
            if "->" in title:
                title_parts = title.split("->")
                title_parts[0] = "ε" + title_parts[0].strip()
                ax.set_title(" -> ".join(title_parts))

        # Export the figure to the PDF file
        pdf_pages.savefig(fig)

        # Show the plots
        plt.close(fig)  # Close the current figure

# Close the PDF file
pdf_pages.close()

print("Plots exported to PDF:", pdf_filename)
```

```
Cross-correlation: 0
AR Form: Intercept=False, Trend=False
AR Form: Intercept=True, Trend=False
AR Form: Intercept=False, Trend=True
AR Form: Intercept=True, Trend=True
Cross-correlation: 0.3
AR Form: Intercept=False, Trend=False
AR Form: Intercept=True, Trend=False
AR Form: Intercept=False, Trend=True
AR Form: Intercept=True, Trend=True
Cross-correlation: 0.6
AR Form: Intercept=False, Trend=False
AR Form: Intercept=True, Trend=False
AR Form: Intercept=False, Trend=True
AR Form: Intercept=True, Trend=True
Cross-correlation: 0.9
AR Form: Intercept=False, Trend=False
AR Form: Intercept=True, Trend=False
AR Form: Intercept=False, Trend=True
AR Form: Intercept=True, Trend=True
Plots exported to PDF: RFVAR_IRF_plots.pdf
```

## IRFs of ARFVAR (Impulse Responses for each series y1t and y2t)

In [12]:
```
from matplotlib.backends.backend_pdf import PdfPages

# Create a PDF file to store all plots
pdf_filename = "ARFVAR_IRF_plots.pdf"
pdf_pages = PdfPages(pdf_filename)

# Loop over different cross-correlation values
for correlation in cross_correlation_values:
    print(f"Cross-correlation: {correlation}")

    # Loop over different AR forms
    for ar_form in ar_forms:
        intercept = ar_form["intercept"]
        trend = ar_form["trend"]
        print(f"AR Form: Intercept={intercept}, Trend={trend}")

        # Generate covariance matrix with the specified cross-correlation
        covariance_matrix = np.array([[1.0, correlation],
                                      [correlation, 1.0]])

        # Generate VAR errors with cross-correlation
        mean = np.zeros(K)
        errors = np.random.multivariate_normal(mean, covariance_matrix, size=T)

        # Perform Cholesky decomposition
        chol_decomp = np.linalg.cholesky(covariance_matrix)

        # Transform errors to make them orthogonal
        orthogonal_errors = np.dot(errors, np.linalg.inv(chol_decomp.T))

        # Simulate SVAR data for p=1
        data = np.zeros((T, K))
        for t in range(1, T):
            lagged_values = data[t-1:t, :]  # Lagged values of the variables

            # Compute the SVAR equation for each variable
            for i in range(K):
                lagged_product = np.dot(lagged_values.flatten(), coefficients[:, i])  # Compute lagg
                intercept_term = 21 if intercept else 0
                trend_term = 0.01 * t if trend else 0
                data[t, i] = lagged_product + intercept_term + trend_term + orthogonal_errors[t, i]

        # Fit SVAR model for p = 1
        model = sm.tsa.VAR(data)
        results = model.fit(1)

        # Compute impulse response functions
        irf = results.irf(10)  # Compute IRFs for 10 periods
        fig = irf.plot(orth=False)   # Plot IRFs without orthogonalization

        # Set x-axis ticks for all subplots
        for i in range(len(fig.axes)):
            fig.axes[i].set_xticks(np.arange(0, 11, 2))

        # Set a customized title for all plots
        fig.suptitle(f"Impulse Response for Cross-correlation: {correlation}, Intercept: {intercept}

        # Update plot titles to denote the shock/error term
        for ax in fig.get_axes():
            title = ax.get_title()
            if "->" in title:
                title_parts = title.split("->")
                title_parts[0] = "ε" + title_parts[0].strip()
                ax.set_title(" -> ".join(title_parts))

        # Export the figure to the PDF file
        pdf_pages.savefig(fig)

        # Show the plots
        plt.close(fig)  # Close the current figure

# Close the PDF file
pdf_pages.close()

print("Plots exported to PDF:", pdf_filename)
```

```
AR Form: Intercept=False, Trend=False
AR Form: Intercept=True, Trend=False
AR Form: Intercept=False, Trend=True
AR Form: Intercept=True, Trend=True
Cross-correlation: 0.3
AR Form: Intercept=False, Trend=False
AR Form: Intercept=True, Trend=False
AR Form: Intercept=False, Trend=True
AR Form: Intercept=True, Trend=True
Cross-correlation: 0.6
AR Form: Intercept=False, Trend=False
AR Form: Intercept=True, Trend=False
AR Form: Intercept=False, Trend=True
AR Form: Intercept=True, Trend=True
Cross-correlation: 0.9
AR Form: Intercept=False, Trend=False
AR Form: Intercept=True, Trend=False
AR Form: Intercept=False, Trend=True
AR Form: Intercept=True, Trend=True
Plots exported to PDF: ARFVAR_IRF_plots.pdf
```