Set-Up

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import statsmodels.tsa.api as tsa
from scipy.linalg import cholesky
from scipy.optimize import minimize, LinearConstraint, NonlinearConstraint
from scipy.linalg import orth

# Plot Setting
plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rc('font', serif='Computer Modern')
```

Transform data

In [2]:
```
mydata = pd.read_csv("mydata.csv")
mydata['time'] = pd.to_datetime(mydata['time']).dt.to_period('Q')
mydata.set_index('time', inplace=True)
mydata.head()
```

```
C:\Users\Yutao\AppData\Local\Temp\ipykernel_5432\2964630960.py:2: UserWarning: Could
not infer format, so each element will be parsed individually, falling back to `date
util`. To ensure parsing is consistent and as-expected, please specify a format.
  mydata['time'] = pd.to_datetime(mydata['time']).dt.to_period('Q')
```

Out[2]:

	Interest Rate	Real Wages	Adjusted Reserves	PPIC	GDP	private_consumption	gover
time							
1954Q3	1.03	42.832	8.191	31.4	390996000000	240303000000	
1954Q4	0.99	43.388	8.362	31.1	399734000000	245093000000	
1955Q1	1.34	43.629	8.339	30.9	413073000000	251398000000	
1955Q2	1.50	44.054	8.358	30.6	421532000000	256466000000	
1955Q3	1.94	44.802	8.320	30.3	430221000000	260651000000	

In [3]:
```
denominator = mydata["gdp_deflator"] * mydata["Population"]
print(denominator.head())
names = [
    "GDP",
    "private_consumption",
    "government_expenditure",
    "government_revenue",
    "private_non_residential_investment",
]
```

```
for name in names:
    mydata[f"real_{name}"] = mydata[name] / denominator
mydata.iloc[:, 1:] = mydata.iloc[:, 1:].map(lambda x: np.log(x))
mydata.head()
```

```
time
1954Q3    2.208911e+09
1954Q4    2.225725e+09
1955Q1    2.245490e+09
1955Q2    2.263748e+09
1955Q3    2.289917e+09
Freq: Q-DEC, dtype: float64
```

```
C:\Users\Yutao\AppData\Local\Temp\ipykernel_5432\1810958852.py:12: FutureWarning: Se
tting an item of incompatible dtype is deprecated and will raise in a future error o
f pandas. Value 'time
1954Q3    26.691963
1954Q4    26.714065
1955Q1    26.746890
1955Q2    26.767162
1955Q3    26.787565
            ...
2014Q4    30.516496
2015Q1    30.524916
2015Q2    30.536817
2015Q3    30.543460
2015Q4    30.545280
Freq: Q-DEC, Name: GDP, Length: 246, dtype: float64' has dtype incompatible with int
64, please explicitly cast to a compatible dtype first.
  mydata.iloc[:, 1:] = mydata.iloc[:, 1:].map(lambda x: np.log(x))
C:\Users\Yutao\AppData\Local\Temp\ipykernel_5432\1810958852.py:12: FutureWarning: Se
tting an item of incompatible dtype is deprecated and will raise in a future error o
f pandas. Value 'time
1954Q3    26.205166
1954Q4    26.224904
1955Q1    26.250303
1955Q2    26.270262
1955Q3    26.286448
            ...
2014Q4    30.122702
2015Q1    30.125859
2015Q2    30.137701
2015Q3    30.147305
2015Q4    30.150599
Freq: Q-DEC, Name: private_consumption, Length: 246, dtype: float64' has dtype incom
patible with int64, please explicitly cast to a compatible dtype first.
  mydata.iloc[:, 1:] = mydata.iloc[:, 1:].map(lambda x: np.log(x))
C:\Users\Yutao\AppData\Local\Temp\ipykernel_5432\1810958852.py:12: FutureWarning: Se
tting an item of incompatible dtype is deprecated and will raise in a future error o
f pandas. Value 'time
1954Q3    25.235750
1954Q4    25.238030
1955Q1    25.243629
1955Q2    25.250831
1955Q3    25.267613
            ...
2014Q4    28.790808
2015Q1    28.790388
2015Q2    28.804762
2015Q3    28.810769
2015Q4    28.812178
Freq: Q-DEC, Name: government_expenditure, Length: 246, dtype: float64' has dtype in
compatible with int64, please explicitly cast to a compatible dtype first.
  mydata.iloc[:, 1:] = mydata.iloc[:, 1:].map(lambda x: np.log(x))
C:\Users\Yutao\AppData\Local\Temp\ipykernel_5432\1810958852.py:12: FutureWarning: Se
tting an item of incompatible dtype is deprecated and will raise in a future error o
f pandas. Value 'time
1954Q3    24.328438
1954Q4    24.375844
```

```
1955Q1    24.491026
1955Q2    24.566646
1955Q3    24.617552
            ...
2014Q4    28.577965
2015Q1    28.614003
2015Q2    28.610021
2015Q3    28.605101
2015Q4    28.589439
Freq: Q-DEC, Name: private_non_residential_investment, Length: 246, dtype: float64'
has dtype incompatible with int64, please explicitly cast to a compatible dtype firs
t.
  mydata.iloc[:, 1:] = mydata.iloc[:, 1:].map(lambda x: np.log(x))
```

Out[3]:

time	Interest Rate	Real Wages	Adjusted Reserves	PPIC	GDP	private_consumption	gov
1954Q3	1.03	3.757285	2.103036	3.446808	26.691963	26.205166	
1954Q4	0.99	3.770183	2.123698	3.437208	26.714065	26.224904	
1955Q1	1.34	3.775722	2.120943	3.430756	26.746890	26.250303	
1955Q2	1.50	3.785416	2.123219	3.421000	26.767162	26.270262	
1955Q3	1.94	3.802253	2.118662	3.411148	26.787565	26.286448	

In [4]:

```
filter_data = mydata[(mydata.index >= "1955Q1") & (mydata.index <= "2000Q4")]
filter_data.head()
```

Out[4]:

time	Interest Rate	Real Wages	Adjusted Reserves	PPIC	GDP	private_consumption	gov
1955Q1	1.34	3.775722	2.120943	3.430756	26.746890	26.250303	
1955Q2	1.50	3.785416	2.123219	3.421000	26.767162	26.270262	
1955Q3	1.94	3.802253	2.118662	3.411148	26.787565	26.286448	
1955Q4	2.36	3.808261	2.119743	3.394508	26.803410	26.301632	
1956Q1	2.48	3.823585	2.124893	3.391147	26.809463	26.307348	

VAR Estimation

In [5]:

```
names = ["real_GDP", "real_private_consumption", "real_government_expenditure", "re
X = filter_data[names]
X.to_csv(r"D:\Program Files (x86)\Project\PythonProject\Python Time Series\Replicat
```

In [6]:

```
var_res = tsa.VAR(X).fit(6, trend="n")
```

```
print(var_res.is_stable())
```

False

Get Parameter

In [7]:
```
sigma = var_res.sigma_u
params = var_res.params
B0 = cholesky(sigma, lower=True)
dim = B0.shape[0]
```

Define f function

In [8]:
```
def f(x):
    if x>=0:
        res = 100 * x
    else:
        res = x
    return res
```

Define get_q

In [9]:
```
def get_q(m):

    random_vector = np.random.randn(m)
    q = random_vector / np.linalg.norm(random_vector)

    return q
```

Define get_r_ja

In [10]:
```
def get_r_ja(var_res, q, horizon=10):

    q = np.asarray(q)
    n = var_res.neqs

    irf = var_res.irf(horizon)
    orth_irfs = irf.orth_irfs

    r_a = np.zeros((horizon+1, n))

    for k in range(horizon + 1):
        r_a[k, :] = orth_irfs[k] @ q

    return r_a
```

Test function

In [11]:
```
q = get_q(10)
irf = var_res.irf(10)
orth_irfs = irf.orth_irfs
```

```
r_a_by_func = get_r_ja(var_res=var_res, q=q)
print(orth_irfs[1,:,:]@q)
```

```
[-0.0021544  -0.00223172  0.00033237 -0.00348191 -0.00279819 -0.00381137
  0.28617107 -0.0111038   0.0177741   0.00042011]
```

In [12]:
```
print(r_a_by_func[1,])
```

```
[-0.0021544  -0.00223172  0.00033237 -0.00348191 -0.00279819 -0.00381137
  0.28617107 -0.0111038   0.0177741   0.00042011]
```

Define psi function

In [13]:
```
def psi(q, positive, negative, periods):
    positive_part = 0
    negative_part = 0
    if len(positive) > 0:
        for positive_index in positive:
            for period in range(periods + 1):
                std = np.sqrt(np.diag(sigma)[positive_index])
                inner_part = get_r_ja(var_res, q=q, horizon=10)[period, positive_in
                positive_part += f(-1 * inner_part)

    if len(negative) > 0:
        for negative_index in negative:
            for period in range(periods + 1):
                std = np.sqrt(np.diag(sigma)[negative_index])
                inner_part = get_r_ja(var_res, q=q, horizon=10)[period, negative_in
                negative_part += f(inner_part)
    res = positive_part + negative_part
    return res
```

Business Cycle shocks

In [14]:
```
from scipy.optimize import minimize
positive = [0, 1, 3, 5]
negative = []
periods = 3



q0 = get_q(10)

norm_constraint = NonlinearConstraint(lambda q: np.sum(q**2), lb=1.0, ub=1.0)

res = minimize(
    fun=lambda q: psi(q, positive, negative, periods),
    x0=q0,
    method='trust-constr',
    constraints=[norm_constraint],
    options={'verbose': 2, 'maxiter': 1000, 'xtol': 1e-12}
)

q_business_shocks = res.x
```

```
| niter |f evals|CG iter|  obj func   |tr radius |   opt    |  c viol  |
|-------|-------|-------|-------------|----------|----------|----------|
|   1   |  11   |   0   | -4.1430e+00 | 1.00e+00 | 1.23e+01 | 2.22e-16 |
|   2   |  22   |   1   | -1.7090e+01 | 7.00e+00 | 3.77e+00 | 1.00e+00 |
|   3   |  33   |   2   | -1.9690e+01 | 8.11e+00 | 3.19e+00 | 1.34e+00 |
|   4   |  44   |   3   | -1.6301e+01 | 8.11e+00 | 1.06e+00 | 4.55e-01 |
|   5   |  55   |   4   | -1.3979e+01 | 8.11e+00 | 4.12e-01 | 5.94e-02 |
|   6   |  66   |   5   | -1.3616e+01 | 8.11e+00 | 9.52e-02 | 3.42e-03 |
|   7   |  77   |   6   | -1.3594e+01 | 8.11e+00 | 1.60e-03 | 9.77e-05 |
|   8   |  88   |   7   | -1.3594e+01 | 8.11e+00 | 4.51e-06 | 2.84e-08 |
|   9   |  99   |   8   | -1.3594e+01 | 8.11e+00 | 3.47e-07 | 2.19e-13 |
|  10   |  110  |  10   | -1.3594e+01 | 8.11e+00 | 5.79e-07 | 8.22e-15 |
|  11   |  121  |  11   | -1.3594e+01 | 8.11e+00 | 1.81e-07 | 5.11e-15 |
|  12   |  143  |  13   | -1.3594e+01 | 8.11e+00 | 5.16e-07 | 2.22e-16 |
|  13   |  165  |  16   | -1.3594e+01 | 8.11e+00 | 4.68e-07 | 1.11e-16 |
|  14   |  176  |  19   | -1.3594e+01 | 8.11e+00 | 1.77e-07 | 4.44e-16 |
|  15   |  198  |  24   | -1.3594e+01 | 8.11e-01 | 1.77e-07 | 4.44e-16 |
|  16   |  220  |  29   | -1.3594e+01 | 8.11e-02 | 1.77e-07 | 4.44e-16 |
|  17   |  242  |  35   | -1.3594e+01 | 8.11e-03 | 1.77e-07 | 4.44e-16 |
|  18   |  264  |  43   | -1.3594e+01 | 8.11e-04 | 1.77e-07 | 4.44e-16 |
|  19   |  286  |  50   | -1.3594e+01 | 8.11e-05 | 1.77e-07 | 4.44e-16 |
|  20   |  308  |  55   | -1.3594e+01 | 8.11e-06 | 1.77e-07 | 4.44e-16 |
|  21   |  330  |  60   | -1.3594e+01 | 8.11e-07 | 1.77e-07 | 4.44e-16 |
|  22   |  352  |  65   | -1.3594e+01 | 8.11e-08 | 1.77e-07 | 4.44e-16 |
|  23   |  374  |  70   | -1.3594e+01 | 8.11e-09 | 1.77e-07 | 4.44e-16 |
```

```
d:\Anaconda\envs\TimeSeriesCode\Lib\site-packages\scipy\optimize\_differentiable_fun
ctions.py:728: UserWarning: delta_grad == 0.0. Check if the approximated function is
linear. If the function is linear better results can be obtained by defining the Hes
sian as zero instead of using quasi-Newton approximations.
  self.H.update(delta_x, delta_g)
```

```
|  24   |  396  |  73   | -1.3594e+01 | 4.06e-09 | 1.77e-07 | 4.44e-16 |
|  25   |  418  |  76   | -1.3594e+01 | 1.50e-09 | 1.77e-07 | 4.44e-16 |
|  26   |  429  |  79   | -1.3594e+01 | 7.19e-10 | 1.77e-07 | 4.44e-16 |
|  27   |  451  |  82   | -1.3594e+01 | 2.09e-10 | 1.77e-07 | 4.44e-16 |
|  28   |  462  |  85   | -1.3594e+01 | 4.19e-11 | 1.77e-07 | 4.44e-16 |
|  29   |  473  |  87   | -1.3594e+01 | 1.77e-11 | 1.77e-07 | 4.44e-16 |
|  30   |  484  |  89   | -1.3594e+01 | 4.74e-12 | 1.77e-07 | 4.44e-16 |
|  31   |  495  |  91   | -1.3594e+01 | 1.27e-12 | 1.77e-07 | 4.44e-16 |
|  32   |  517  |  93   | -1.3594e+01 | 2.47e-13 | 1.77e-07 | 4.44e-16 |

`xtol` termination condition is satisfied.
Number of iterations: 32, function evaluations: 517, CG iterations: 93, optimality:
1.77e-07, constraint violation: 4.44e-16, execution time: 1.6e+01 s.
```

In [15]:

```python
r_business = get_r_ja(var_res,q_business_shocks, 25)
print(r_business.shape)
```

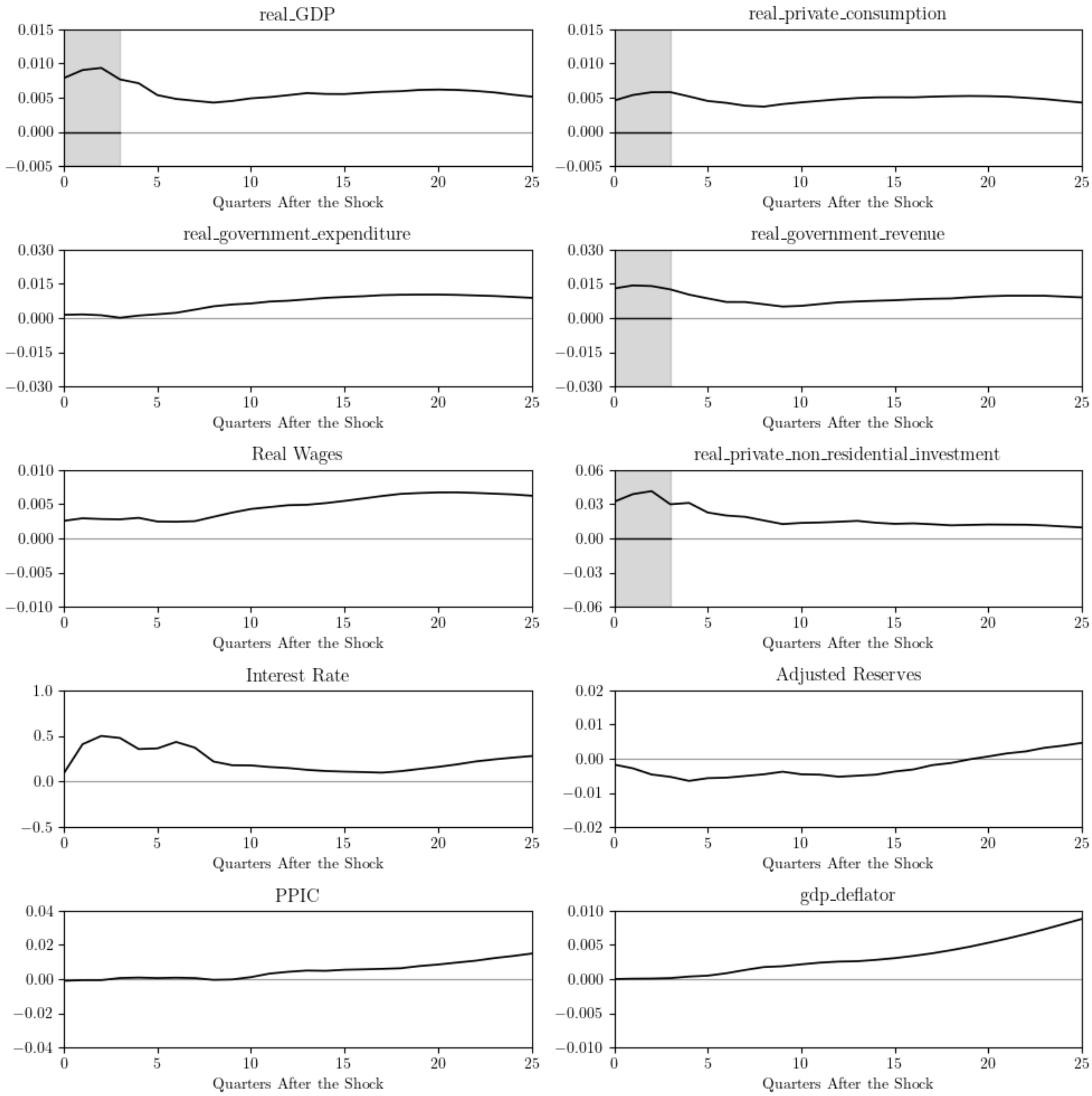
```
(26, 10)
```

Plot Business shocks

In [16]:

```python
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
    for j in range(2):
        axes[i,j].plot(r_business[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
```

```
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[0,0].axvspan(0, 3, color='gray', alpha=0.3)
axes[0,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[1,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[2,1].axvspan(0, 3, color='gray', alpha=0.3)

axes[0,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[0,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[1,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[2,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.005, 0.015)
axes[0,1].set_ylim(-0.005, 0.015)
axes[1,0].set_ylim(-0.03, 0.03)
axes[1,1].set_ylim(-0.03, 0.03)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.06, 0.06)
axes[3,0].set_ylim(-0.5, 1)
axes[3,1].set_ylim(-0.02, 0.02)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)

axes[1,0].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[1,1].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[2,1].set_yticks([-0.06,-0.03,0,0.03,0.06])

plt.tight_layout()
plt.savefig("business.pdf", format="PDF")
plt.show()
```



Monetary shocks

In [17]:
```
positive = [6]
negative = [7, 8, 9]
periods = 3

q0 = get_q(10)

q_business_shocks = q_business_shocks / np.linalg.norm(q_business_shocks)

norm_constraint = NonlinearConstraint(lambda q: np.sum(q**2), lb=1.0, ub=1.0)
orth_constraint = LinearConstraint(q_business_shocks.reshape(1, -1), lb=0.0, ub=0.0

res = minimize(
    fun=lambda q: psi(q, positive, negative, periods),
    x0=q0,
    method='trust-constr',
    constraints=[norm_constraint, orth_constraint],
```

```
    options={'verbose': 2, 'maxiter': 1000, 'xtol': 1e-12}
)

q_monetary_shocks = res.x
```

niter	f evals	CG iter	obj func	tr radius	opt	c viol
1	11	0	+4.5168e+02	1.00e+00	5.26e+02	5.34e-02
2	22	1	+6.5099e-01	2.00e+00	2.03e+02	1.00e+00
3	33	3	-5.7562e+00	2.64e+00	4.92e+00	1.42e-01
4	44	5	-5.5061e+00	2.64e+00	4.89e+00	4.79e-03
5	55	7	-5.4858e+00	2.64e+00	5.41e+01	2.17e-05
6	66	10	-5.7037e+00	2.64e+00	4.71e+00	8.85e-04
7	88	13	-6.4159e+00	2.64e+00	3.97e+00	6.81e-05
8	110	16	-6.9469e+00	2.64e+00	3.47e+00	3.28e-05
9	132	20	-6.9469e+00	2.64e-01	3.47e+00	3.28e-05
10	154	23	-7.9303e+00	1.85e+00	2.58e+00	1.22e-03
11	176	27	-8.3496e+00	1.85e+00	2.11e+00	4.86e-03
12	187	31	-8.3642e+00	1.85e+00	2.23e+00	4.18e-03
13	209	35	-8.3642e+00	1.85e-01	2.23e+00	4.18e-03
14	220	38	-8.3565e+00	1.85e-01	2.23e+00	2.45e-05
15	231	41	-8.3655e+00	1.85e-01	2.23e+00	1.68e-05
16	253	43	-8.3655e+00	1.85e-02	2.23e+00	1.68e-05
17	275	44	-8.3655e+00	1.85e-03	2.23e+00	1.68e-05
18	286	45	-8.3729e+00	1.29e-02	2.22e+00	3.42e-06
19	308	48	-8.3789e+00	9.06e-02	2.22e+00	7.30e-09
20	330	51	-8.3839e+00	1.64e-01	2.21e+00	7.57e-08
21	352	52	-8.3839e+00	1.64e-02	2.21e+00	7.57e-08
22	374	56	-8.3839e+00	1.64e-03	2.21e+00	7.57e-08
23	385	57	-8.3905e+00	1.15e-02	2.21e+00	2.70e-06
24	396	58	-8.3910e+00	1.15e-02	2.21e+00	1.91e-08
25	407	59	-8.3916e+00	1.15e-02	2.21e+00	1.91e-08
26	418	60	-8.3921e+00	1.15e-02	2.21e+00	1.91e-08
27	429	61	-8.3927e+00	1.15e-02	2.21e+00	1.91e-08
28	451	62	-8.3927e+00	1.15e-03	2.21e+00	1.91e-08
29	473	63	-8.3927e+00	1.15e-04	2.21e+00	1.91e-08
30	495	64	-8.3927e+00	1.21e-05	2.21e+00	1.91e-08
31	506	65	-8.3927e+00	8.45e-05	2.21e+00	1.46e-10
32	517	66	-8.3927e+00	8.45e-05	2.21e+00	1.67e-11
33	528	67	-8.3928e+00	8.45e-05	2.21e+00	1.67e-11
34	539	68	-8.3928e+00	8.45e-05	2.21e+00	1.67e-11
35	550	69	-8.3928e+00	8.45e-05	2.20e+00	1.67e-11
36	572	70	-8.3928e+00	8.45e-06	2.20e+00	1.67e-11
37	583	71	-8.3928e+00	2.35e-05	2.20e+00	1.13e-11
38	594	72	-8.3928e+00	2.35e-05	2.20e+00	1.04e-11
39	605	73	-8.3928e+00	2.35e-05	2.20e+00	1.04e-11
40	616	74	-8.3928e+00	2.35e-05	2.20e+00	1.04e-11
41	627	75	-8.3929e+00	2.35e-05	2.20e+00	1.04e-11
42	638	76	-8.3929e+00	1.64e-04	2.20e+00	5.52e-10
43	660	77	-8.3929e+00	1.64e-05	2.20e+00	5.52e-10
44	671	78	-8.3930e+00	4.70e-05	2.20e+00	4.51e-11
45	693	79	-8.3930e+00	4.70e-06	2.20e+00	4.51e-11
46	704	80	-8.3930e+00	4.70e-06	2.20e+00	6.37e-14
47	715	81	-8.3930e+00	4.70e-06	2.20e+00	5.88e-14
48	726	82	-8.3930e+00	4.70e-06	2.20e+00	5.97e-14
49	737	83	-8.3930e+00	4.70e-06	2.20e+00	5.86e-14
50	759	84	-8.3930e+00	4.70e-07	2.20e+00	5.86e-14
51	770	85	-8.3930e+00	1.30e-06	2.20e+00	3.82e-14
52	781	86	-8.3930e+00	1.30e-06	2.18e+01	3.33e-14
53	803	88	-8.3930e+00	1.30e-07	2.18e+01	3.33e-14
54	825	89	-8.3930e+00	6.52e-08	2.18e+01	3.33e-14

```
d:\Anaconda\envs\TimeSeriesCode\Lib\site-packages\scipy\optimize\_differentiable_fun
ctions.py:728: UserWarning: delta_grad == 0.0. Check if the approximated function is
linear. If the function is linear better results can be obtained by defining the Hes
sian as zero instead of using quasi-Newton approximations.
  self.H.update(delta_x, delta_g)
|   55    |  847  |  90   | -8.3930e+00 | 3.26e-08 | 2.18e+01 | 3.33e-14 |
|   56    |  869  |  91   | -8.3930e+00 | 1.63e-08 | 2.18e+01 | 3.33e-14 |
|   57    |  891  |  92   | -8.3930e+00 | 8.15e-09 | 2.18e+01 | 3.33e-14 |
|   58    |  913  |  93   | -8.3930e+00 | 4.08e-09 | 2.18e+01 | 3.33e-14 |
d:\Anaconda\envs\TimeSeriesCode\Lib\site-packages\scipy\optimize\_differentiable_fun
ctions.py:376: UserWarning: delta_grad == 0.0. Check if the approximated function is
linear. If the function is linear better results can be obtained by defining the Hes
sian as zero instead of using quasi-Newton approximations.
  self.H.update(self.x - self.x_prev, self.g - self.g_prev)
|   59    |  935  |  94   | -8.3930e+00 | 2.04e-09 | 2.18e+01 | 3.33e-14 |
|   60    |  957  |  95   | -8.3930e+00 | 1.02e-09 | 2.18e+01 | 3.33e-14 |
|   61    |  979  |  96   | -8.3930e+00 | 5.10e-10 | 2.18e+01 | 3.33e-14 |
|   62    | 1001  |  97   | -8.3930e+00 | 2.55e-10 | 2.18e+01 | 3.33e-14 |
|   63    | 1023  |  98   | -8.3930e+00 | 1.27e-10 | 2.18e+01 | 3.33e-14 |
|   64    | 1045  |  99   | -8.3930e+00 | 6.37e-11 | 2.18e+01 | 3.33e-14 |
|   65    | 1067  |  100  | -8.3930e+00 | 3.18e-11 | 2.18e+01 | 3.33e-14 |
|   66    | 1089  |  101  | -8.3930e+00 | 1.59e-11 | 2.18e+01 | 3.33e-14 |
|   67    | 1111  |  102  | -8.3930e+00 | 7.96e-12 | 2.18e+01 | 3.33e-14 |
|   68    | 1133  |  103  | -8.3930e+00 | 3.98e-12 | 2.18e+01 | 3.33e-14 |
|   69    | 1144  |  104  | -8.3930e+00 | 1.99e-12 | 2.18e+01 | 3.33e-14 |
|   70    | 1166  |  105  | -8.3930e+00 | 9.95e-13 | 2.18e+01 | 3.33e-14 |

`xtol` termination condition is satisfied.
Number of iterations: 70, function evaluations: 1166, CG iterations: 105, optimalit
y: 2.18e+01, constraint violation: 3.33e-14, execution time: 3.7e+01 s.
```

In [18]:

```python
r_monetary = get_r_ja(var_res, q_monetary_shocks, 25)
print(r_monetary.shape)
```

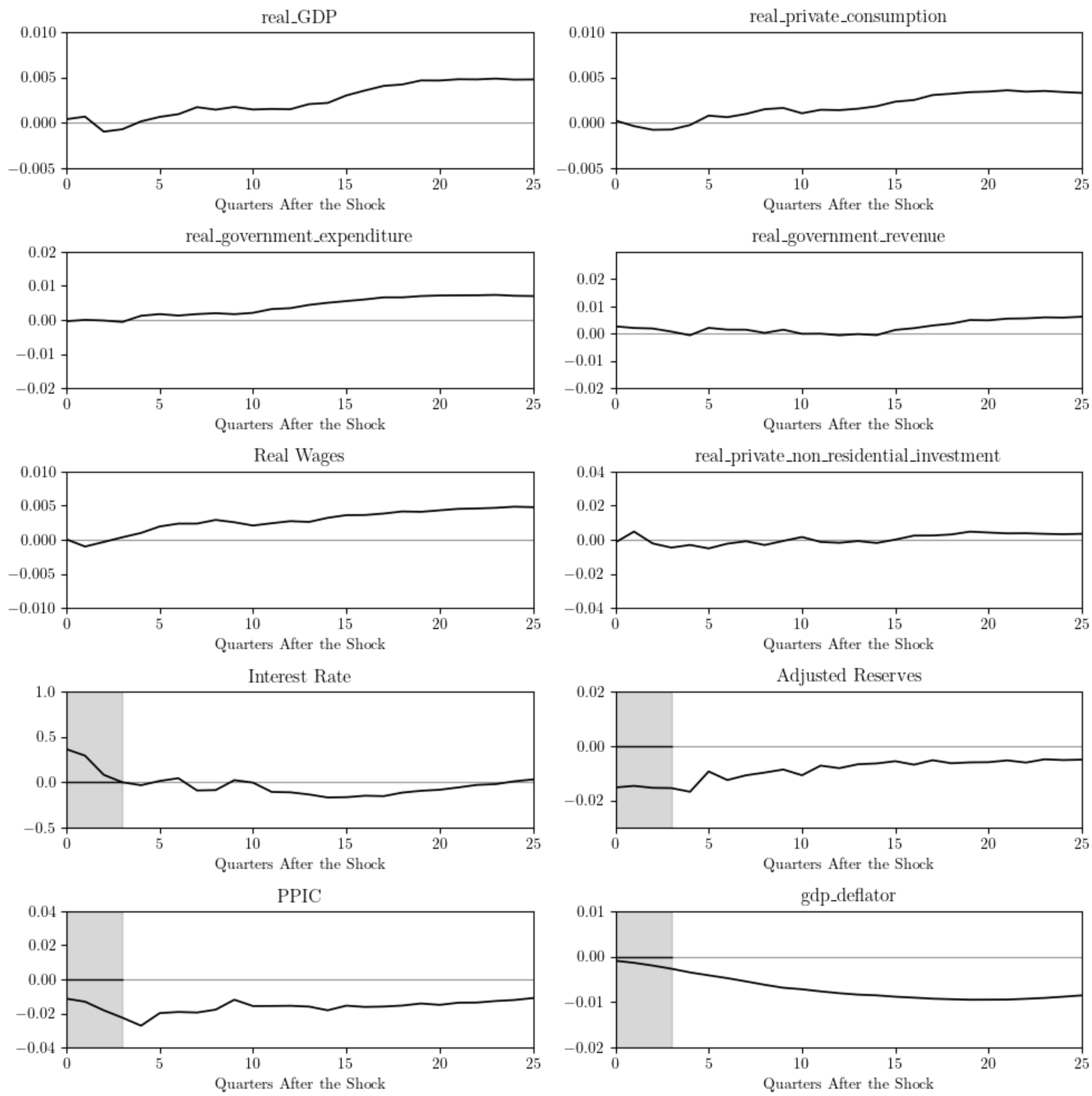
```
(26, 10)
```

Plot Monetary Shocks

In [19]:

```python
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
    for j in range(2):
        axes[i,j].plot(r_monetary[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[3,0].axvspan(0, 3, color='gray', alpha=0.3)
axes[3,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[4,0].axvspan(0, 3, color='gray', alpha=0.3)
axes[4,1].axvspan(0, 3, color='gray', alpha=0.3)

axes[3,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[3,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[4,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
```

```
axes[4,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.005, 0.01)
axes[0,1].set_ylim(-0.005, 0.01)
axes[1,0].set_ylim(-0.02, 0.02)
axes[1,1].set_ylim(-0.02, 0.03)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 1)
axes[3,1].set_ylim(-0.03, 0.02)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.02, 0.01)

axes[1,1].set_yticks([-0.02,-0.01,0,0.01,0.02])
plt.tight_layout()
plt.savefig("monetary.pdf", format="PDF")
plt.show()
```



Fiscal revenue shocks

In [20]:
```
positive = [3]
negative = []
periods = 3

q0 = get_q(10)

q_business_shocks = q_business_shocks / np.linalg.norm(q_business_shocks)
q_monetary_shocks = q_monetary_shocks / np.linalg.norm(q_monetary_shocks)

norm_constraint = NonlinearConstraint(lambda q: np.sum(q**2), lb=1.0, ub=1.0)
orth_constraint1 = LinearConstraint(q_business_shocks.reshape(1, -1), lb=0.0, ub=0.
orth_constraint2 = LinearConstraint(q_monetary_shocks.reshape(1, -1), lb=0.0, ub=0.


res = minimize(
    fun=lambda q: psi(q, positive, negative, periods),
    x0=q0,
    method='trust-constr',
    constraints=[norm_constraint, orth_constraint1, orth_constraint2],
    options={'verbose': 2, 'maxiter': 1000, 'xtol': 1e-12}
)

q_revenue_shocks = res.x
```

```
| niter |f evals|CG iter|  obj func   |tr radius |   opt    |  c viol  |
|-------|-------|-------|-------------|----------|----------|----------|
|   1   |  11   |   0   | +3.9525e+01 | 1.00e+00 | 1.64e+02 | 7.91e-02 |
|   2   |  22   |   1   | -1.9198e+00 | 1.00e+00 | 1.04e+00 | 1.00e+00 |
|   3   |  33   |   2   | -1.2287e+00 | 2.69e+00 | 1.30e+00 | 1.48e-01 |
|   4   |  44   |   3   | -1.1056e+00 | 2.69e+00 | 1.34e+00 | 5.55e-03 |
|   5   |  55   |   4   | -1.1099e+00 | 2.69e+00 | 1.34e+00 | 2.83e-05 |
|   6   |  66   |   5   | -1.1194e+00 | 2.69e+00 | 1.33e+00 | 3.53e-05 |
|   7   |  77   |   6   | -1.1287e+00 | 2.69e+00 | 1.32e+00 | 3.46e-05 |
|   8   |  88   |   7   | -1.1380e+00 | 2.69e+00 | 1.31e+00 | 3.39e-05 |
|   9   |  99   |   8   | -1.1471e+00 | 2.69e+00 | 1.30e+00 | 3.33e-05 |
|  10   |  121  |   9   | -1.1471e+00 | 2.69e-01 | 1.30e+00 | 3.33e-05 |
|  11   |  143  |  10   | -1.5298e+00 | 1.89e+00 | 8.08e-01 | 1.32e-03 |
|  12   |  154  |  11   | -1.6022e+00 | 1.89e+00 | 6.99e-01 | 3.70e-03 |
|  13   |  165  |  13   | -1.6379e+00 | 1.89e+00 | 6.38e-01 | 1.20e-03 |
|  14   |  176  |  14   | -1.6686e+00 | 1.89e+00 | 6.20e-01 | 8.98e-04 |
|  15   |  187  |  16   | -1.6930e+00 | 1.89e+00 | 6.07e-01 | 6.26e-04 |
|  16   |  209  |  18   | -1.6930e+00 | 1.89e-01 | 6.07e-01 | 6.26e-04 |
|  17   |  231  |  19   | -1.8446e+00 | 1.32e+00 | 4.53e-01 | 3.27e-04 |
|  18   |  253  |  21   | -1.9509e+00 | 1.32e+00 | 7.97e-02 | 4.75e-03 |
|  19   |  264  |  23   | -1.9515e+00 | 1.32e+00 | 1.11e-02 | 2.77e-03 |
|  20   |  275  |  24   | -1.9488e+00 | 1.32e+00 | 1.08e-02 | 1.97e-06 |
|  21   |  286  |  25   | -1.9488e+00 | 1.32e+00 | 1.04e-02 | 7.64e-08 |
|  22   |  297  |  26   | -1.9488e+00 | 1.32e+00 | 1.00e-02 | 1.12e-07 |
|  23   |  308  |  27   | -1.9488e+00 | 1.32e+00 | 9.62e-03 | 1.03e-07 |
|  24   |  330  |  28   | -1.9488e+00 | 1.32e+00 | 1.31e-07 | 8.92e-10 |
|  25   |  341  |  30   | -1.9488e+00 | 1.32e+00 | 1.78e-08 | 1.51e-14 |
|  26   |  352  |  33   | -1.9488e+00 | 1.32e+00 | 2.41e-08 | 6.94e-18 |
|  27   |  374  |  37   | -1.9488e+00 | 1.32e-01 | 2.41e-08 | 6.94e-18 |
|  28   |  396  |  44   | -1.9488e+00 | 1.32e-02 | 2.41e-08 | 6.94e-18 |
```

```
d:\Anaconda\envs\TimeSeriesCode\Lib\site-packages\scipy\optimize\_differentiable_fun
ctions.py:376: UserWarning: delta_grad == 0.0. Check if the approximated function is
linear. If the function is linear better results can be obtained by defining the Hes
sian as zero instead of using quasi-Newton approximations.
  self.H.update(self.x - self.x_prev, self.g - self.g_prev)
d:\Anaconda\envs\TimeSeriesCode\Lib\site-packages\scipy\optimize\_differentiable_fun
ctions.py:728: UserWarning: delta_grad == 0.0. Check if the approximated function is
linear. If the function is linear better results can be obtained by defining the Hes
sian as zero instead of using quasi-Newton approximations.
  self.H.update(delta_x, delta_g)
```

```
|   29   |   418   |   50    | -1.9488e+00 | 1.32e-03 | 2.41e-08 | 6.94e-18 |
|   30   |   440   |   55    | -1.9488e+00 | 1.32e-03 | 3.16e-08 | 0.00e+00 |
|   31   |   462   |   60    | -1.9488e+00 | 1.32e-04 | 3.16e-08 | 0.00e+00 |
|   32   |   484   |   65    | -1.9488e+00 | 1.32e-05 | 3.16e-08 | 0.00e+00 |
|   33   |   506   |   70    | -1.9488e+00 | 1.32e-06 | 3.16e-08 | 0.00e+00 |
|   34   |   528   |   75    | -1.9488e+00 | 1.32e-07 | 3.16e-08 | 0.00e+00 |
|   35   |   550   |   80    | -1.9488e+00 | 1.32e-08 | 3.16e-08 | 0.00e+00 |
|   36   |   572   |   85    | -1.9488e+00 | 1.74e-09 | 3.16e-08 | 0.00e+00 |
|   37   |   594   |   87    | -1.9488e+00 | 7.98e-10 | 3.16e-08 | 0.00e+00 |
|   38   |   616   |   90    | -1.9488e+00 | 3.99e-10 | 3.16e-08 | 0.00e+00 |
|   39   |   627   |   92    | -1.9488e+00 | 2.00e-10 | 3.16e-08 | 0.00e+00 |
|   40   |   649   |   94    | -1.9488e+00 | 9.98e-11 | 3.16e-08 | 0.00e+00 |
|   41   |   671   |   96    | -1.9488e+00 | 4.55e-11 | 3.16e-08 | 0.00e+00 |
|   42   |   693   |   98    | -1.9488e+00 | 2.18e-11 | 3.16e-08 | 0.00e+00 |
|   43   |   715   |   99    | -1.9488e+00 | 1.09e-11 | 3.16e-08 | 0.00e+00 |
|   44   |   737   |   100   | -1.9488e+00 | 5.45e-12 | 3.16e-08 | 0.00e+00 |
|   45   |   759   |   101   | -1.9488e+00 | 2.73e-12 | 3.16e-08 | 0.00e+00 |
|   46   |   781   |   104   | -1.9488e+00 | 1.36e-12 | 3.16e-08 | 0.00e+00 |
|   47   |   792   |   107   | -1.9488e+00 | 6.81e-13 | 3.16e-08 | 0.00e+00 |

`xtol` termination condition is satisfied.
Number of iterations: 47, function evaluations: 792, CG iterations: 107, optimality: 3.16e-08, constraint violation: 0.00e+00, execution time:  6.3 s.
```

In [21]:

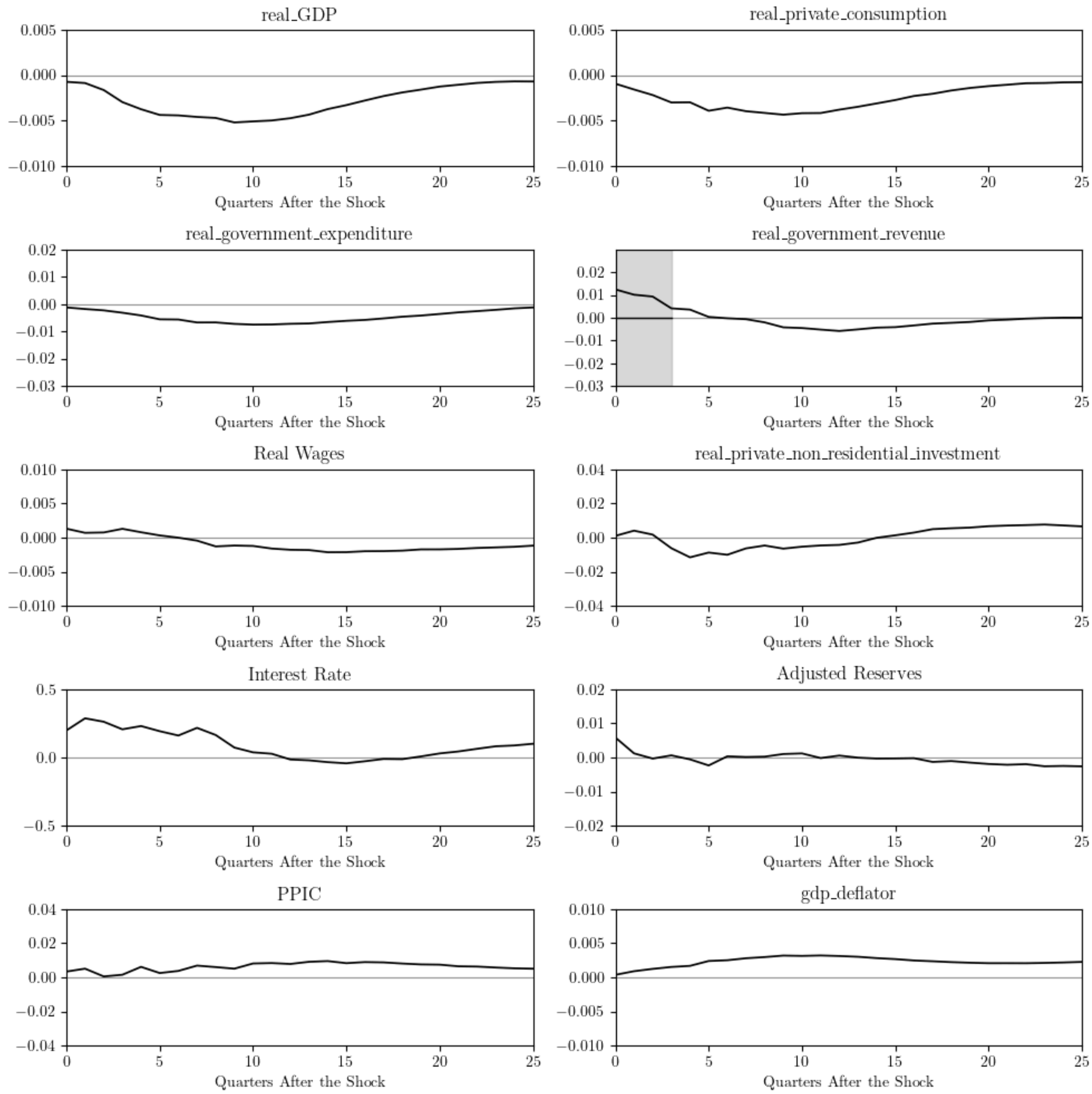
```python
r_revenue = get_r_ja(var_res, q_revenue_shocks, 25)
print(r_revenue.shape)
```

```
(26, 10)
```

In [22]:

```python
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
    for j in range(2):
        axes[i,j].plot(r_revenue[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[1,1].axvspan(0, 3, color='gray', alpha=0.3)

axes[1,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.01, 0.005)
axes[0,1].set_ylim(-0.01, 0.005)
axes[1,0].set_ylim(-0.03, 0.02)
axes[1,1].set_ylim(-0.03, 0.03)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 0.5)
axes[3,1].set_ylim(-0.02, 0.02)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)


axes[1,0].set_yticks([-0.03,-0.02,-0.01,0,0.01,0.02])
axes[1,1].set_yticks([-0.03,-0.02,-0.01,0,0.01,0.02])
```

```
axes[3,0].set_yticks([-0.5,0,0.5])

plt.tight_layout()
plt.savefig("revenue.pdf", format="PDF")
plt.show()
```



Fiscal spending shocks

In [23]:
```
positive = [2]
negative = []
periods = 3



q_business_shocks = q_business_shocks / np.linalg.norm(q_business_shocks)
q_monetary_shocks = q_monetary_shocks / np.linalg.norm(q_monetary_shocks)

norm_constraint = NonlinearConstraint(lambda q: np.sum(q**2), lb=1.0, ub=1.0)
orth_constraint1 = LinearConstraint(q_business_shocks.reshape(1, -1), lb=0.0, ub=0.
orth_constraint2 = LinearConstraint(q_monetary_shocks.reshape(1, -1), lb=0.0, ub=0.
```

```python
q0 = get_q(10)

res = minimize(
    fun=lambda q: psi(q, positive, negative, periods),
    x0=q0,
    method='trust-constr',
    constraints=[norm_constraint, orth_constraint1, orth_constraint2],
    options={'verbose': 2, 'maxiter': 1000, 'xtol': 1e-12}
)

q_spending_shocks = res.x
```

```
| niter |f evals|CG iter|  obj func   |tr radius |   opt    |  c viol  |
|-------|-------|-------|-------------|----------|----------|----------|
|   1   |  11   |   0   | -1.3921e+00 | 1.00e+00 | 2.28e+00 | 4.55e-01 |
|   2   |  22   |   1   | -3.6936e+00 | 7.00e+00 | 9.77e-01 | 1.00e+00 |
|   3   |  33   |   2   | -4.9084e+00 | 7.77e+00 | 2.35e-01 | 1.23e+00 |
|   4   |  44   |   3   | -3.6163e+00 | 7.77e+00 | 5.43e-02 | 1.96e-01 |
|   5   |  55   |   4   | -3.3251e+00 | 7.77e+00 | 5.87e-02 | 1.14e-02 |
|   6   |  66   |   5   | -3.3098e+00 | 7.77e+00 | 8.68e-03 | 1.20e-03 |
|   7   |  77   |   6   | -3.3079e+00 | 7.77e+00 | 1.85e-05 | 1.99e-05 |
|   8   |  88   |   7   | -3.3079e+00 | 7.77e+00 | 1.43e-07 | 1.89e-10 |
|   9   |  99   |   8   | -3.3079e+00 | 7.77e+00 | 3.67e-08 | 6.44e-15 |
|  10   |  110  |   9   | -3.3079e+00 | 7.77e+00 | 5.59e-08 | 6.66e-16 |
|  11   |  121  |  11   | -3.3079e+00 | 7.77e+00 | 7.00e-08 | 1.33e-15 |
|  12   |  132  |  13   | -3.3079e+00 | 7.77e+00 | 4.05e-08 | 1.11e-15 |
|  13   |  154  |  15   | -3.3079e+00 | 7.77e-01 | 4.05e-08 | 1.11e-15 |
|  14   |  176  |  17   | -3.3079e+00 | 7.77e-01 | 5.70e-08 | 2.22e-16 |
|  15   |  198  |  20   | -3.3079e+00 | 7.77e-02 | 5.70e-08 | 2.22e-16 |
|  16   |  220  |  23   | -3.3079e+00 | 7.77e-03 | 5.70e-08 | 2.22e-16 |
|  17   |  242  |  27   | -3.3079e+00 | 7.77e-04 | 5.70e-08 | 2.22e-16 |
|  18   |  264  |  30   | -3.3079e+00 | 7.77e-05 | 5.70e-08 | 2.22e-16 |
```

```
d:\Anaconda\envs\TimeSeriesCode\Lib\site-packages\scipy\optimize\_differentiable_fun
ctions.py:728: UserWarning: delta_grad == 0.0. Check if the approximated function is
linear. If the function is linear better results can be obtained by defining the Hes
sian as zero instead of using quasi-Newton approximations.
  self.H.update(delta_x, delta_g)
```

```
|  19   |  286  |  33   | -3.3079e+00 | 7.77e-06 | 5.70e-08 | 2.22e-16 |
|  20   |  308  |  38   | -3.3079e+00 | 7.77e-07 | 5.70e-08 | 2.22e-16 |
|  21   |  330  |  43   | -3.3079e+00 | 7.77e-08 | 5.70e-08 | 2.22e-16 |
|  22   |  352  |  48   | -3.3079e+00 | 1.57e-08 | 5.70e-08 | 2.22e-16 |
|  23   |  374  |  50   | -3.3079e+00 | 7.62e-09 | 5.70e-08 | 2.22e-16 |
|  24   |  385  |  52   | -3.3079e+00 | 1.52e-08 | 8.82e-08 | 2.22e-16 |
|  25   |  407  |  54   | -3.3079e+00 | 4.23e-09 | 8.82e-08 | 2.22e-16 |
|  26   |  429  |  55   | -3.3079e+00 | 2.12e-09 | 8.82e-08 | 2.22e-16 |
```

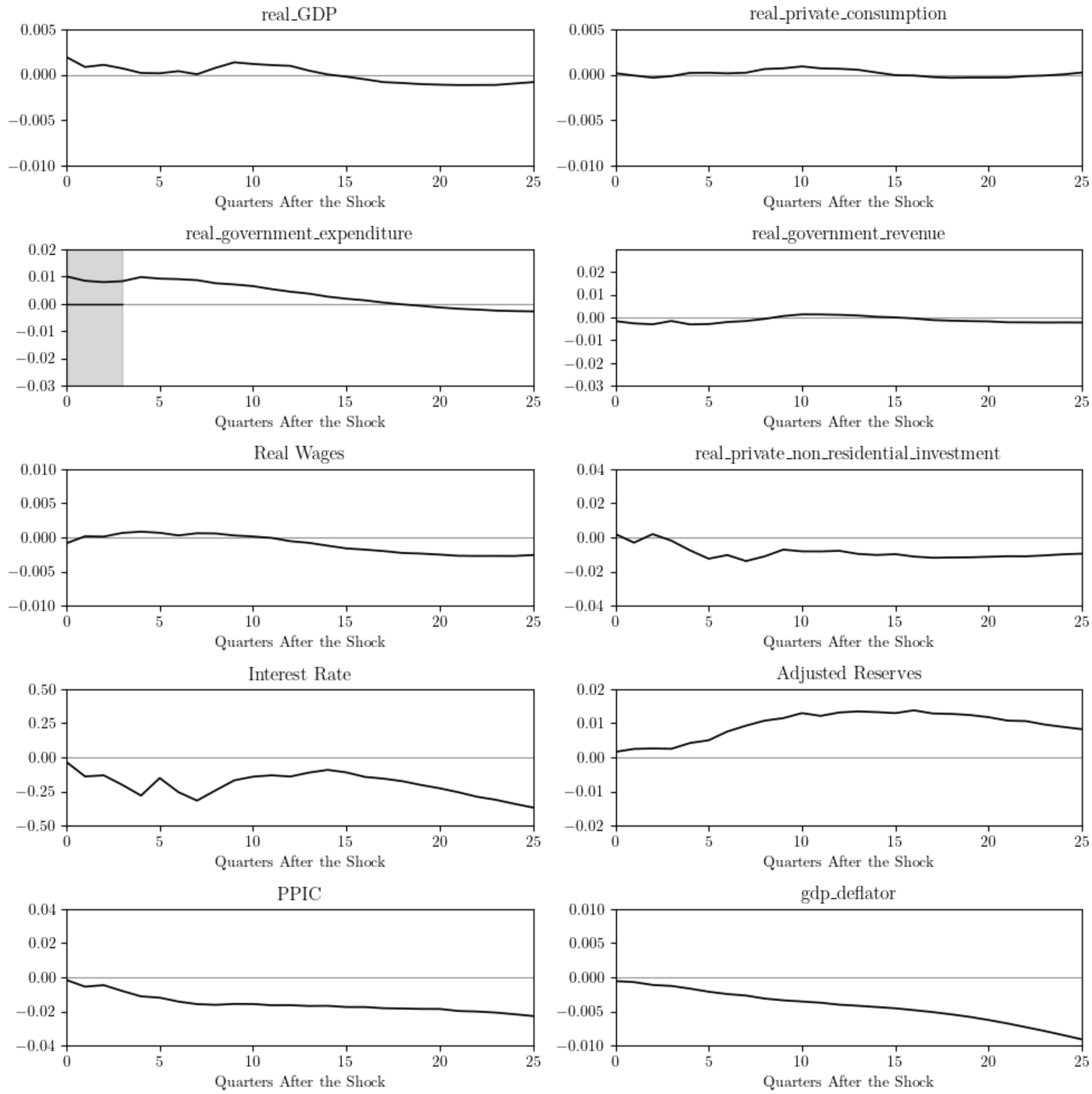
```
d:\Anaconda\envs\TimeSeriesCode\Lib\site-packages\scipy\optimize\_differentiable_fun
ctions.py:376: UserWarning: delta_grad == 0.0. Check if the approximated function is
linear. If the function is linear better results can be obtained by defining the Hes
sian as zero instead of using quasi-Newton approximations.
  self.H.update(self.x - self.x_prev, self.g - self.g_prev)
```

```
|   27   |  451   |   57   | -3.3079e+00 | 6.05e-10 | 8.82e-08 | 2.22e-16 |
|   28   |  473   |   59   | -3.3079e+00 | 1.22e-10 | 8.82e-08 | 2.22e-16 |
|   29   |  495   |   61   | -3.3079e+00 | 1.97e-11 | 8.82e-08 | 2.22e-16 |
|   30   |  517   |   63   | -3.3079e+00 | 5.21e-12 | 8.82e-08 | 2.22e-16 |
|   31   |  539   |   65   | -3.3079e+00 | 5.52e-13 | 8.82e-08 | 2.22e-16 |

`xtol` termination condition is satisfied.
Number of iterations: 31, function evaluations: 539, CG iterations: 65, optimality:
8.82e-08, constraint violation: 2.22e-16, execution time:  4.3 s.
```

In [24]:
```
r_spending = get_r_ja(var_res, q_spending_shocks, 25)
print(r_spending.shape)
```

```
(26, 10)
```

In [25]:
```
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
    for j in range(2):
        axes[i,j].plot(r_spending[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[1,0].axvspan(0, 3, color='gray', alpha=0.3)

axes[1,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.01, 0.005)
axes[0,1].set_ylim(-0.01, 0.005)
axes[1,0].set_ylim(-0.03, 0.02)
axes[1,1].set_ylim(-0.03, 0.03)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 0.5)
axes[3,1].set_ylim(-0.02, 0.02)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)

axes[1,0].set_yticks([-0.03,-0.02,-0.01,0, 0.01 ,0.02])
axes[1,1].set_yticks([-0.03,-0.02,-0.01,0, 0.01 ,0.02])


plt.tight_layout()
plt.savefig("spending.pdf", format="PDF")
plt.show()
```



Extension-1 Zero-restriction

Define Zero-restriction psi function

In [26]:
```
def zero_psi(q, positive, negative, periods_start, periods_end):
    positive_part = 0
    negative_part = 0
    if len(positive) > 0:
        for positive_index in positive:
            for period in range(periods_start, periods_end + 1):
                std = np.sqrt(np.diag(sigma)[positive_index])
                inner_part = get_r_ja(var_res, q=q, horizon=10)[period, positive_in
                positive_part += f(-1 * inner_part)

    if len(negative) > 0:
        for negative_index in negative:
            for period in range(periods_start, periods_end + 1):
```

```
                std = np.sqrt(np.diag(sigma)[negative_index])
                inner_part = get_r_ja(var_res, q=q, horizon=10)[period, negative_in
                negative_part += f(inner_part)
    res = positive_part + negative_part
    return res
```

Zero-restriction for revenue shocks

In [27]:
```
j = 3
h_zero = 4
R = np.vstack([var_res.irf(10).orth_irfs[k][j, :] for k in range(h_zero+1)])


norm_constraint = NonlinearConstraint(lambda q: np.sum(q**2), lb=1.0, ub=1.0)

orth_constraint1 = LinearConstraint(q_business_shocks.reshape(1, -1), lb=0.0, ub=0.
orth_constraint2 = LinearConstraint(q_monetary_shocks.reshape(1, -1), lb=0.0, ub=0.

zero_constraints = [LinearConstraint(R[i].reshape(1, -1), lb=0.0, ub=0.0) for i in

q0 = get_q(10)


positive = [3]
negative = []
periods_start = 4
periods_end = 7


res = minimize(
    fun=lambda q: zero_psi(q, positive, negative, periods_start, periods_end),
    x0=q0,
    method='trust-constr',
    constraints=[
        norm_constraint,
        orth_constraint1,
        orth_constraint2,
        *zero_constraints
    ],
    options={'verbose': 2, 'maxiter': 1000, 'xtol': 1e-12}
)

q_anticipated_revenue_shocks = res.x
```

niter	f evals	CG iter	obj func	tr radius	opt	c viol
1	11	0	-5.1372e-01	1.00e+00	4.91e-01	3.09e-01
2	22	1	-5.1372e-01	2.53e-01	4.91e-01	3.09e-01
3	33	2	-4.5711e-01	1.77e+00	4.06e-01	1.49e-01
4	44	3	-4.5711e-01	2.16e-01	4.06e-01	1.49e-01
5	55	4	-3.6510e-01	1.51e+00	4.65e-01	9.97e-02
6	66	5	-3.6510e-01	1.51e-01	4.65e-01	9.97e-02
7	77	6	-4.2366e-01	1.03e+00	4.28e-01	4.63e-02
8	88	7	-4.2366e-01	1.79e-01	4.28e-01	4.63e-02
9	99	8	-5.0566e-01	1.25e+00	3.81e-01	1.48e-02
10	110	9	-5.0566e-01	1.25e-01	3.81e-01	1.48e-02
11	121	11	-5.1731e-01	2.51e-01	3.86e-01	1.46e-02
12	132	13	-5.1731e-01	3.77e-02	3.86e-01	1.46e-02
13	143	14	-5.3637e-01	2.64e-01	3.74e-01	3.44e-03
14	154	16	-5.3637e-01	2.64e-02	3.74e-01	3.44e-03
15	165	17	-5.4999e-01	1.85e-01	3.66e-01	1.53e-03
16	176	19	-5.4999e-01	1.85e-02	3.66e-01	1.53e-03
17	187	20	-5.6161e-01	1.29e-01	3.60e-01	1.48e-03
18	198	21	-5.6161e-01	5.39e-02	3.60e-01	1.48e-03
19	209	22	-5.9502e-01	1.08e-01	3.44e-01	2.75e-03
20	220	23	-5.9502e-01	5.39e-02	3.44e-01	2.75e-03
21	231	24	-6.2651e-01	1.08e-01	3.27e-01	2.89e-03
22	242	25	-6.2651e-01	5.39e-02	3.27e-01	2.89e-03
23	253	26	-6.5639e-01	1.08e-01	3.14e-01	2.90e-03
24	264	27	-6.5639e-01	5.39e-02	3.14e-01	2.90e-03
25	275	28	-6.8456e-01	1.08e-01	3.04e-01	2.91e-03
26	286	29	-6.8456e-01	5.39e-02	3.04e-01	2.91e-03
27	297	30	-7.1090e-01	1.08e-01	2.92e-01	2.91e-03
28	308	31	-7.1090e-01	5.39e-02	2.92e-01	2.91e-03
29	319	32	-7.3531e-01	1.08e-01	2.80e-01	2.91e-03
30	330	33	-7.3531e-01	5.39e-02	2.80e-01	2.91e-03
31	341	34	-7.5766e-01	1.08e-01	2.67e-01	2.91e-03
32	352	35	-7.5766e-01	5.39e-02	2.67e-01	2.91e-03
33	363	36	-7.5766e-01	2.70e-02	2.67e-01	2.91e-03
34	374	38	-7.5315e-01	5.39e-02	2.71e-01	7.27e-04
35	385	39	-7.5315e-01	2.70e-02	2.71e-01	7.27e-04
36	396	40	-7.6333e-01	5.39e-02	2.65e-01	7.27e-04
37	407	41	-7.6333e-01	2.70e-02	2.65e-01	7.27e-04
38	418	42	-7.6333e-01	1.01e-02	2.65e-01	7.27e-04
39	429	44	-7.6210e-01	7.04e-02	2.66e-01	1.01e-04
40	440	45	-7.6210e-01	7.04e-03	2.66e-01	1.01e-04
41	451	46	-7.6462e-01	4.93e-02	2.64e-01	6.82e-05
42	462	47	-7.7277e-01	4.93e-02	4.03e+00	4.81e-04
43	484	49	-7.9129e-01	9.86e-02	4.83e+00	2.06e-06
44	506	51	-7.9129e-01	4.93e-02	4.83e+00	2.06e-06
45	528	52	-7.9129e-01	2.47e-02	4.83e+00	2.06e-06
46	550	53	-7.9129e-01	1.23e-02	4.83e+00	2.06e-06
47	572	54	-7.9129e-01	6.16e-03	4.83e+00	2.06e-06
48	594	55	-7.9129e-01	3.08e-03	4.83e+00	2.06e-06
49	616	56	-7.9129e-01	1.54e-03	4.83e+00	2.06e-06
50	638	57	-7.9129e-01	7.70e-04	4.83e+00	2.06e-06
51	660	58	-7.9129e-01	3.85e-04	4.83e+00	2.06e-06
52	682	59	-7.9129e-01	1.93e-04	4.83e+00	2.06e-06
53	704	60	-7.9129e-01	9.63e-05	4.83e+00	2.06e-06
54	726	61	-7.9129e-01	4.81e-05	4.83e+00	2.06e-06

```
|  55   |  737  |  62   | -7.9127e-01 | 4.81e-05 | 4.83e+00 | 2.32e-09 |
|  56   |  759  |  63   | -7.9127e-01 | 2.41e-05 | 4.83e+00 | 2.32e-09 |
|  57   |  781  |  64   | -7.9127e-01 | 1.20e-05 | 4.83e+00 | 2.32e-09 |
|  58   |  803  |  65   | -7.9127e-01 | 6.02e-06 | 4.83e+00 | 2.32e-09 |
|  59   |  825  |  66   | -7.9127e-01 | 3.01e-06 | 4.83e+00 | 2.32e-09 |
|  60   |  847  |  67   | -7.9127e-01 | 1.50e-06 | 4.83e+00 | 2.32e-09 |
|  61   |  869  |  68   | -7.9127e-01 | 7.52e-07 | 4.83e+00 | 2.32e-09 |
|  62   |  891  |  69   | -7.9127e-01 | 3.76e-07 | 4.83e+00 | 2.32e-09 |
|  63   |  913  |  70   | -7.9127e-01 | 1.88e-07 | 4.83e+00 | 2.32e-09 |
|  64   |  935  |  71   | -7.9127e-01 | 9.40e-08 | 4.83e+00 | 2.32e-09 |
|  65   |  957  |  72   | -7.9127e-01 | 4.70e-08 | 4.83e+00 | 2.32e-09 |
|  66   |  968  |  73   | -7.9127e-01 | 4.70e-08 | 4.83e+00 | 4.00e-15 |
|  67   |  990  |  74   | -7.9127e-01 | 2.35e-08 | 4.83e+00 | 4.00e-15 |
|  68   | 1012  |  75   | -7.9127e-01 | 1.18e-08 | 4.83e+00 | 4.00e-15 |
```

```
d:\Anaconda\envs\TimeSeriesCode\Lib\site-packages\scipy\optimize\_differentiable_fun
ctions.py:728: UserWarning: delta_grad == 0.0. Check if the approximated function is
linear. If the function is linear better results can be obtained by defining the Hes
sian as zero instead of using quasi-Newton approximations.
  self.H.update(delta_x, delta_g)
```

```
|  69   | 1034  |  76   | -7.9127e-01 | 5.88e-09 | 4.83e+00 | 4.00e-15 |
|  70   | 1056  |  77   | -7.9127e-01 | 2.94e-09 | 4.83e+00 | 4.00e-15 |
|  71   | 1078  |  78   | -7.9127e-01 | 1.47e-09 | 4.83e+00 | 4.00e-15 |
|  72   | 1100  |  79   | -7.9127e-01 | 7.35e-10 | 4.83e+00 | 4.00e-15 |
|  73   | 1122  |  80   | -7.9127e-01 | 3.67e-10 | 4.83e+00 | 4.00e-15 |
|  74   | 1144  |  81   | -7.9127e-01 | 1.84e-10 | 4.83e+00 | 4.00e-15 |
|  75   | 1166  |  82   | -7.9127e-01 | 9.18e-11 | 4.83e+00 | 4.00e-15 |
|  76   | 1188  |  83   | -7.9127e-01 | 4.59e-11 | 4.83e+00 | 4.00e-15 |
|  77   | 1210  |  84   | -7.9127e-01 | 2.30e-11 | 4.83e+00 | 4.00e-15 |
|  78   | 1232  |  85   | -7.9127e-01 | 1.15e-11 | 4.83e+00 | 4.00e-15 |
|  79   | 1254  |  86   | -7.9127e-01 | 5.74e-12 | 4.83e+00 | 4.00e-15 |
|  80   | 1276  |  87   | -7.9127e-01 | 2.87e-12 | 4.83e+00 | 4.00e-15 |
|  81   | 1298  |  88   | -7.9127e-01 | 1.43e-12 | 4.83e+00 | 4.00e-15 |
|  82   | 1320  |  89   | -7.9127e-01 | 7.17e-13 | 4.83e+00 | 4.00e-15 |

`xtol` termination condition is satisfied.
Number of iterations: 82, function evaluations: 1320, CG iterations: 89, optimality:
4.83e+00, constraint violation: 4.00e-15, execution time: 1e+01 s.
```

In [28]:

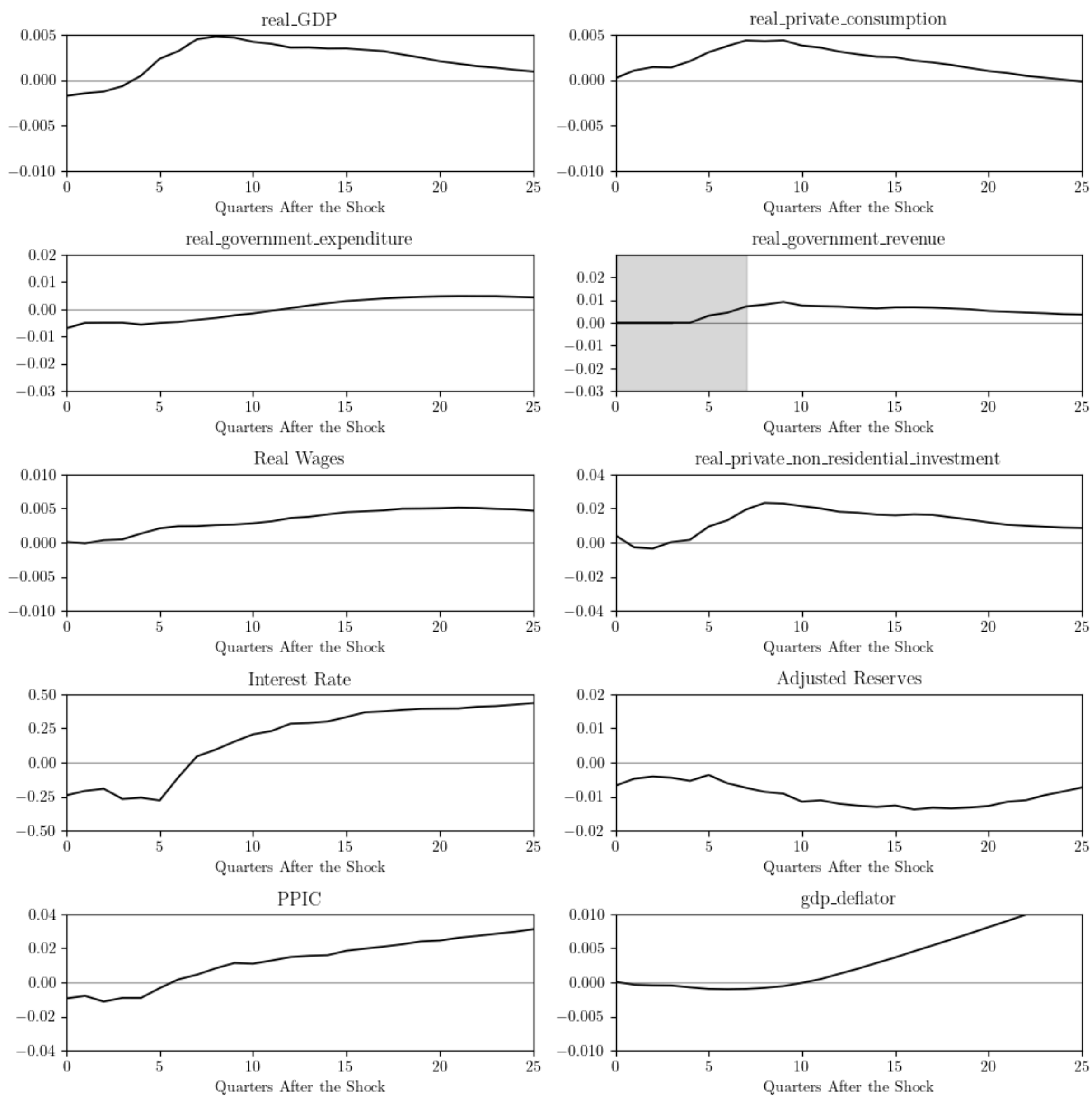
```
r_anticipated_revenue = get_r_ja(var_res, q_anticipated_revenue_shocks, 25)
print(r_anticipated_revenue.shape)
```

```
(26, 10)
```

In [29]:

```
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
    for j in range(2):
        axes[i,j].plot(r_anticipated_revenue[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[1,1].axvspan(0, 7, color='gray', alpha=0.3)

axes[1,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
```

```
axes[0,0].set_ylim(-0.01, 0.005)
axes[0,1].set_ylim(-0.01, 0.005)
axes[1,0].set_ylim(-0.03, 0.02)
axes[1,1].set_ylim(-0.03, 0.03)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 0.5)
axes[3,1].set_ylim(-0.02, 0.02)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)

axes[1,0].set_yticks([-0.03,-0.02,-0.01,0, 0.01 ,0.02])
axes[1,1].set_yticks([-0.03,-0.02,-0.01,0, 0.01 ,0.02])


plt.tight_layout()
plt.savefig("anticipated_revenue.pdf", format="PDF")
plt.show()
```



Zero-restriction for spending shocks

In [30]:
```
j = 2
h_zero = 4
R = np.vstack([var_res.irf(10).orth_irfs[k][j, :] for k in range(h_zero+1)])

norm_constraint = NonlinearConstraint(lambda q: np.sum(q**2), lb=1.0, ub=1.0)

orth_constraint1 = LinearConstraint(q_business_shocks.reshape(1, -1), lb=0.0, ub=0.
orth_constraint2 = LinearConstraint(q_monetary_shocks.reshape(1, -1), lb=0.0, ub=0.

zero_constraints = [LinearConstraint(R[i].reshape(1, -1), lb=0.0, ub=0.0) for i in

q0 = get_q(10)


positive = [2]
negative = []
periods_start = 5
periods_end = 8


res = minimize(
    fun=lambda q: zero_psi(q, positive, negative, periods_start, periods_end),
    x0=q0,
    method='trust-constr',
    constraints=[
        norm_constraint,
        orth_constraint1,
        orth_constraint2,
        *zero_constraints
    ],
    options={'verbose': 2, 'maxiter': 1000, 'xtol': 1e-12, 'gtol': 1e-12}
)

q_anticipated_spending_shocks = res.x
```

```
| niter |f evals|CG iter|  obj func   |tr radius |   opt    |  c viol  |
|-------|-------|-------|-------------|----------|----------|----------|
|   1   |  11   |   0   | +1.5483e+02 | 1.00e+00 | 5.65e+01 | 4.73e-01 |
|   2   |  22   |   1   | +1.3207e+02 | 7.00e+00 | 6.62e+01 | 7.77e-01 |
|   3   |  33   |   2   | +2.3134e+00 | 7.00e+00 | 2.39e+01 | 5.21e-01 |
|   4   |  44   |   4   | -1.6783e-01 | 7.00e+00 | 8.63e-01 | 5.43e-02 |
|   5   |  55   |   5   | -1.7630e-01 | 7.00e+00 | 8.65e-01 | 7.87e-04 |
|   6   |  66   |   6   | -1.8173e-01 | 7.00e+00 | 8.66e-01 | 1.63e-05 |
|   7   |  77   |   8   | -1.8757e-01 | 7.00e+00 | 8.68e-01 | 2.01e-05 |
|   8   |  99   |  10   | -2.0192e-01 | 7.00e+00 | 8.76e-01 | 8.10e-06 |
|   9   |  121  |  11   | -2.0192e-01 | 7.00e-01 | 8.76e-01 | 8.10e-06 |
|  10   |  143  |  12   | -1.1093e+00 | 7.00e-01 | 5.81e-01 | 6.00e-02 |
|  11   |  165  |  13   | -1.4265e+00 | 7.00e-01 | 4.90e-02 | 7.05e-02 |
|  12   |  176  |  14   | -1.3869e+00 | 7.00e-01 | 1.09e-02 | 7.80e-03 |
|  13   |  187  |  15   | -1.3819e+00 | 7.00e-01 | 1.60e-04 | 2.61e-04 |
|  14   |  198  |  16   | -1.3817e+00 | 7.00e-01 | 1.41e-06 | 6.79e-08 |
|  15   |  209  |  17   | -1.3817e+00 | 7.00e-01 | 7.37e-09 | 3.96e-12 |
|  16   |  220  |  18   | -1.3817e+00 | 7.00e-01 | 1.07e-08 | 2.22e-16 |
|  17   |  231  |  19   | -1.3817e+00 | 7.00e-01 | 2.08e-09 | 2.78e-17 |
|  18   |  253  |  21   | -1.3817e+00 | 7.00e-02 | 2.08e-09 | 2.78e-17 |
|  19   |  264  |  23   | -1.3817e+00 | 7.00e-02 | 9.50e-09 | 2.78e-17 |
|  20   |  275  |  25   | -1.3817e+00 | 7.00e-02 | 5.79e-10 | 2.78e-17 |
|  21   |  297  |  27   | -1.3817e+00 | 7.00e-03 | 5.79e-10 | 2.78e-17 |
|  22   |  308  |  28   | -1.3817e+00 | 7.00e-04 | 5.79e-10 | 2.78e-17 |
|  23   |  330  |  30   | -1.3817e+00 | 7.00e-05 | 5.79e-10 | 2.78e-17 |
|  24   |  352  |  32   | -1.3817e+00 | 7.00e-06 | 5.79e-10 | 2.78e-17 |
|  25   |  374  |  33   | -1.3817e+00 | 7.00e-07 | 5.79e-10 | 2.78e-17 |
|  26   |  396  |  34   | -1.3817e+00 | 7.00e-08 | 5.79e-10 | 2.78e-17 |
|  27   |  418  |  35   | -1.3817e+00 | 7.00e-09 | 5.79e-10 | 2.78e-17 |
|  28   |  440  |  36   | -1.3817e+00 | 7.00e-10 | 5.79e-10 | 2.78e-17 |
|  29   |  462  |  37   | -1.3817e+00 | 7.00e-11 | 5.79e-10 | 2.78e-17 |
|  30   |  484  |  38   | -1.3817e+00 | 7.00e-12 | 5.79e-10 | 2.78e-17 |
|  31   |  506  |  39   | -1.3817e+00 | 7.00e-13 | 5.79e-10 | 2.78e-17 |

`xtol` termination condition is satisfied.
Number of iterations: 31, function evaluations: 506, CG iterations: 39, optimality:
5.79e-10, constraint violation: 2.78e-17, execution time:  4.0 s.
```

In [31]:

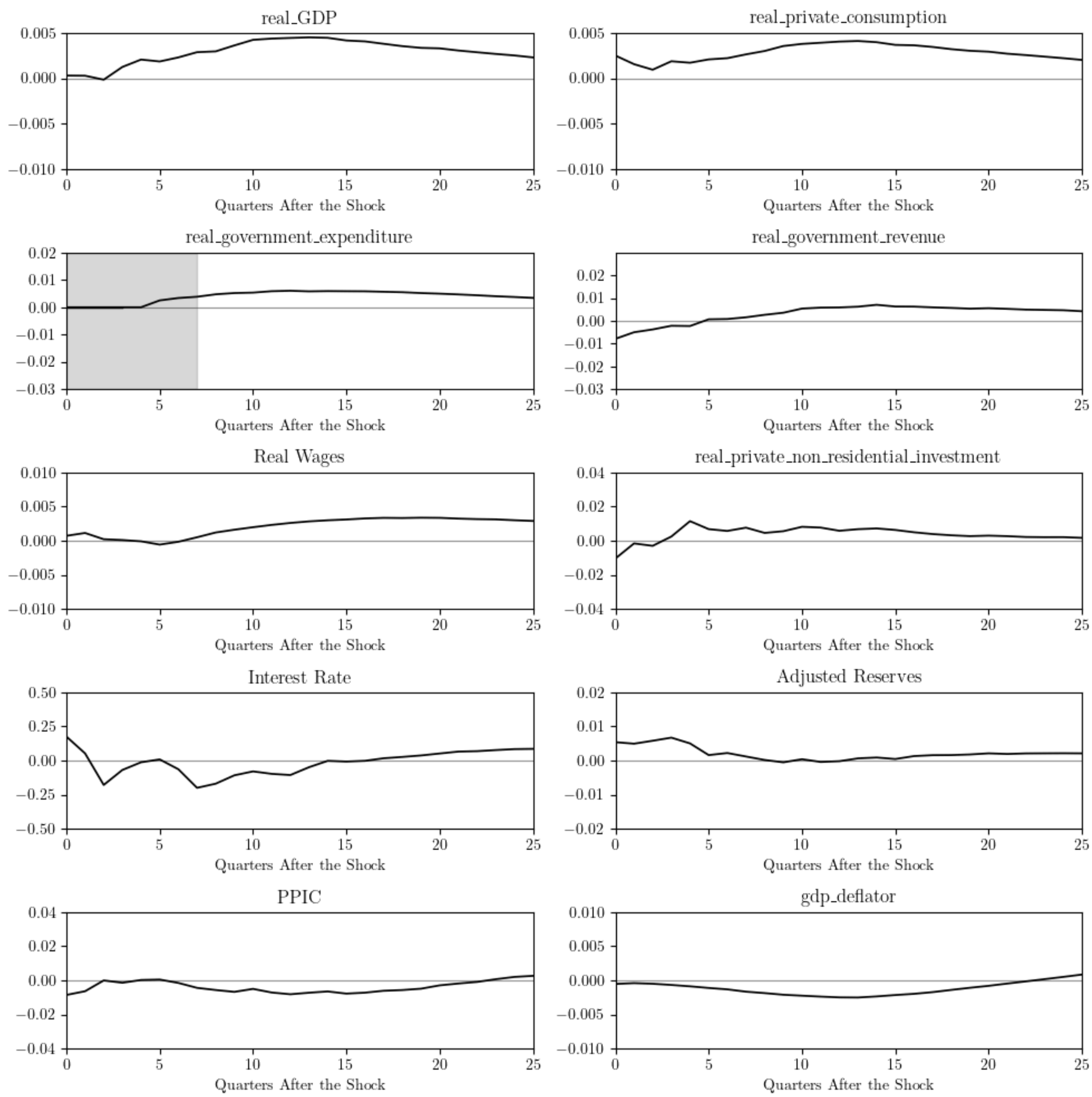
```
r_anticipated_spending = get_r_ja(var_res, q_anticipated_spending_shocks, 25)
print(r_anticipated_spending.shape)
```

```
(26, 10)
```

In [32]:

```
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
    for j in range(2):
        axes[i,j].plot(r_anticipated_spending[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[1,0].axvspan(0, 7, color='gray', alpha=0.3)

axes[1,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
```

```
axes[0,0].set_ylim(-0.01, 0.005)
axes[0,1].set_ylim(-0.01, 0.005)
axes[1,0].set_ylim(-0.03, 0.02)
axes[1,1].set_ylim(-0.03, 0.03)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 0.5)
axes[3,1].set_ylim(-0.02, 0.02)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)

axes[1,0].set_yticks([-0.03,-0.02,-0.01,0, 0.01 ,0.02])
axes[1,1].set_yticks([-0.03,-0.02,-0.01,0, 0.01 ,0.02])


plt.tight_layout()
plt.savefig("anticipated_spending.pdf", format="PDF")
plt.show()
```



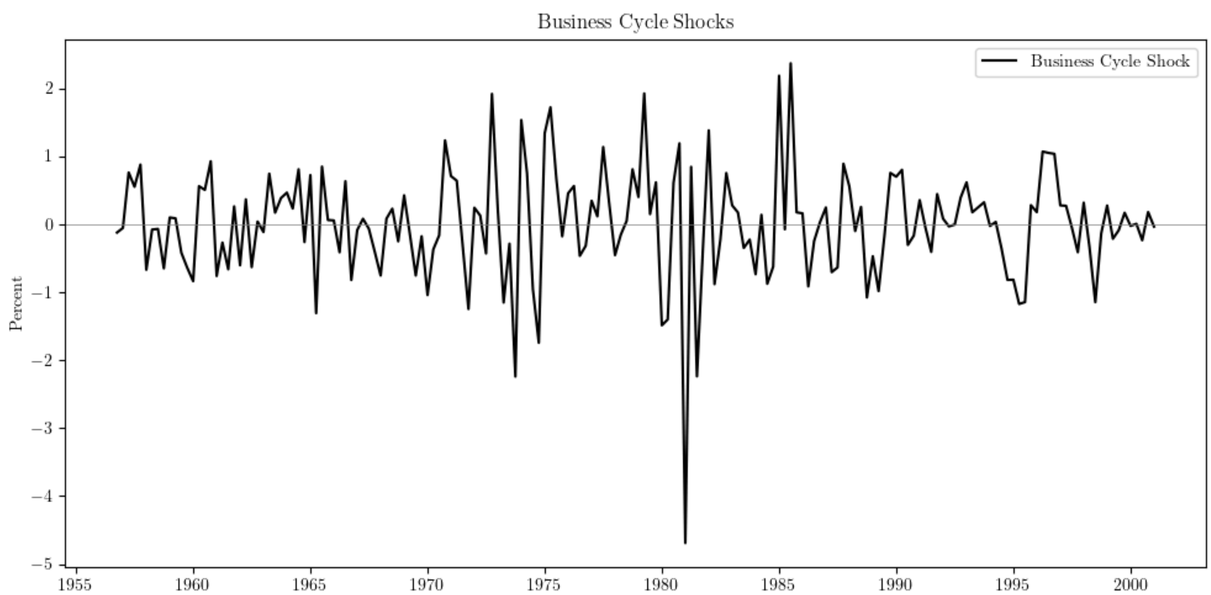
Extension 2 - Time trend of Shocks

In [33]:
```
resid = var_res.resid
print(resid.shape)
```

```
(178, 10)
```

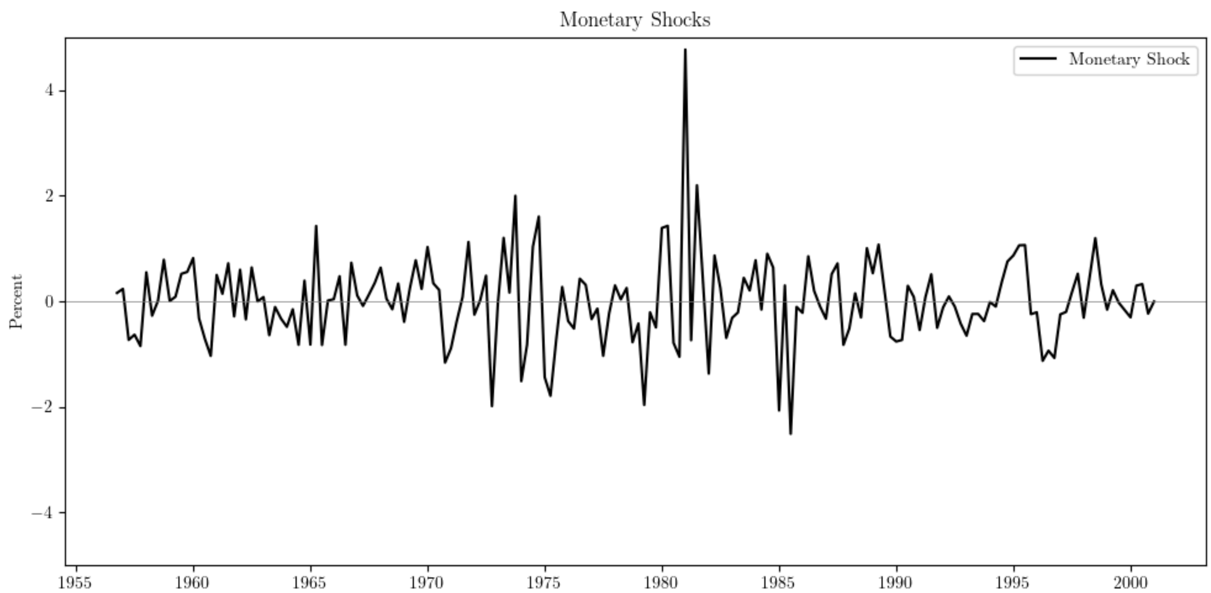
In [34]:
```
q = q_business_shocks / np.sqrt(q_business_shocks.T @ sigma @ q_business_shocks)
eps_biz = resid @ q
```

In [35]:
```
T = resid.shape[0]
dates = pd.date_range(start="1956-10-01", periods=T, freq="QS")
plt.figure(figsize=(10, 5))
plt.plot(dates, eps_biz, label="Business Cycle Shock", color='black')
plt.axhline(0, color='gray', lw=0.5)
#plt.ylim(-0.5,0.5)
plt.ylabel("Percent")
plt.title("Business Cycle Shocks")
plt.legend()
plt.tight_layout()
plt.show()
```



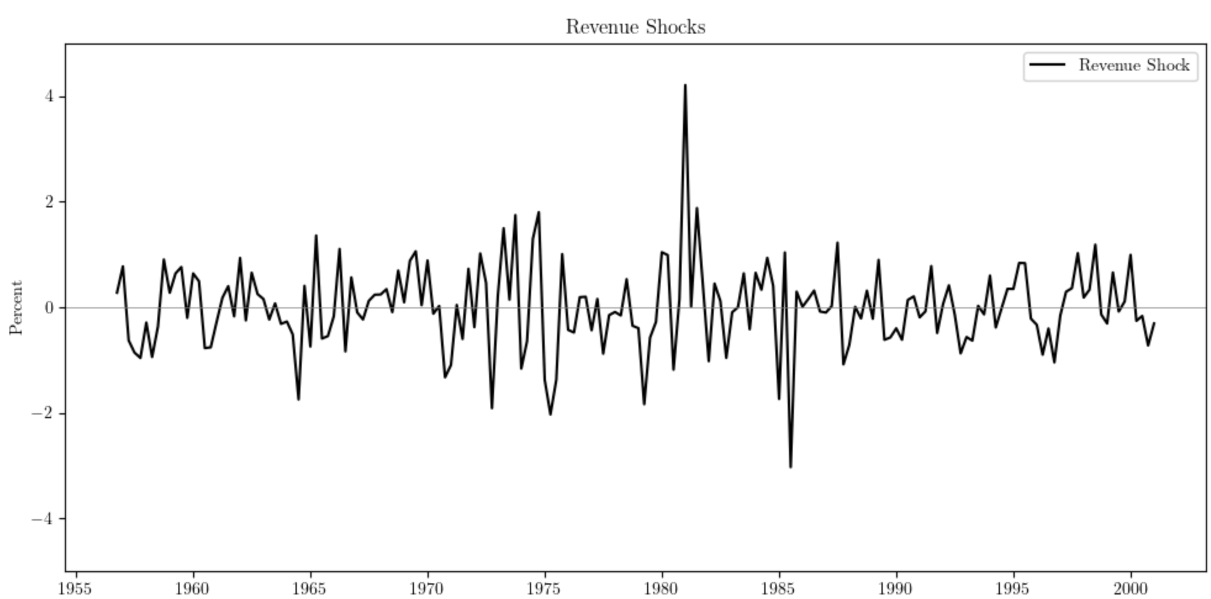
In [36]:
```
q = q_monetary_shocks / np.sqrt(q_monetary_shocks.T @ sigma @ q_monetary_shocks)
eps_monetary= resid @ q
```

In [37]:
```
T = resid.shape[0]
dates = pd.date_range(start="1956-10-01", periods=T, freq="QS")
plt.figure(figsize=(10, 5))
plt.plot(dates, eps_monetary, label="Monetary Shock", color='black')
plt.axhline(0, color='gray', lw=0.5)
plt.ylim(-5,5)
plt.ylabel("Percent")
plt.title("Monetary Shocks")
plt.legend()
plt.tight_layout()
plt.show()
```



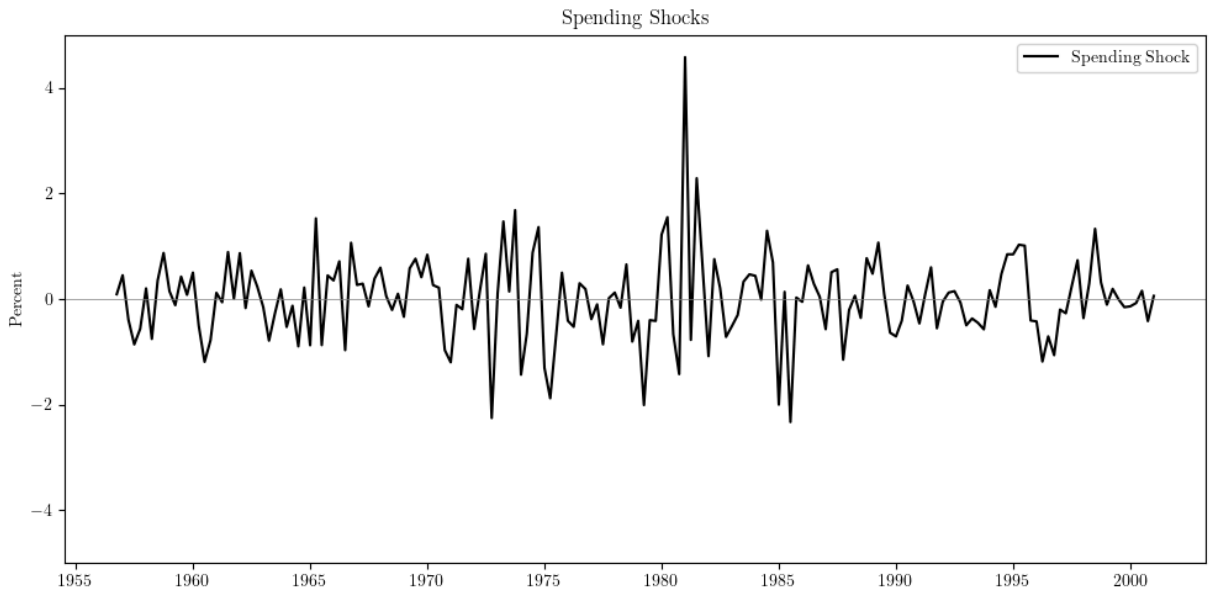
In [38]:
```
q = q_revenue_shocks / np.sqrt(q_revenue_shocks.T @ sigma @ q_revenue_shocks)
eps_revenue = resid @ q
```

In [39]:
```
T = resid.shape[0]
dates = pd.date_range(start="1956-10-01", periods=T, freq="QS")
plt.figure(figsize=(10, 5))
plt.plot(dates, eps_revenue, label="Revenue Shock", color='black')
plt.axhline(0, color='gray', lw=0.5)
plt.ylim(-5,5)
plt.ylabel("Percent")
plt.title("Revenue Shocks")
plt.legend()
plt.tight_layout()
plt.show()
```



In [40]:
```
q = q_spending_shocks / np.sqrt(q_spending_shocks.T @ sigma @ q_spending_shocks)
eps_spending = resid @ q
```

In [41]:
```
T = resid.shape[0]
dates = pd.date_range(start="1956-10-01", periods=T, freq="QS")
plt.figure(figsize=(10, 5))
plt.plot(dates, eps_spending, label="Spending Shock", color='black')
plt.axhline(0, color='gray', lw=0.5)
plt.ylim(-5,5)
plt.ylabel("Percent")
plt.title("Spending Shocks")
plt.legend()
plt.tight_layout()
plt.show()
```



Extension 3 - Confidence Interval

In [42]:
```
lower, upper = var_res.irf_errband_mc(orth=True, steps=30, repl=3000, signif=0.32,
```

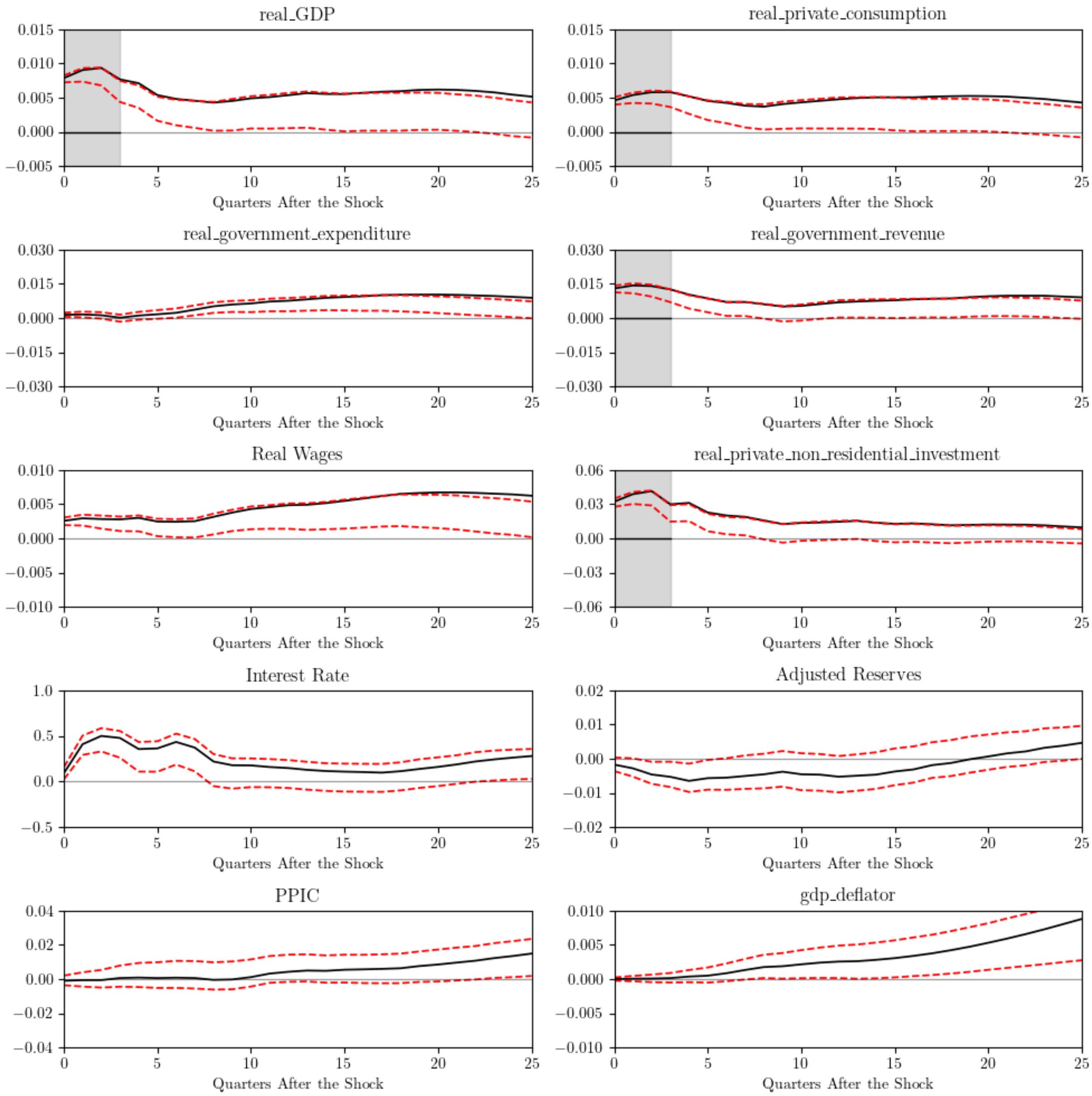
Business Cycle Shocks

In [43]:
```
irf_lower = np.zeros((25 + 1, 10))
irf_upper = np.zeros((25 + 1, 10))

for k in range(25 + 1):
    for i in range(10):
        cholesky_irf_lower = lower[k, i, :]
        cholesky_irf_upper = upper[k, i, :]


        irf_lower[k, i] = np.dot(cholesky_irf_lower, q_business_shocks)
        irf_upper[k, i] = np.dot(cholesky_irf_upper, q_business_shocks)
print(irf_lower.shape, irf_upper.shape)
```

```
(26, 10) (26, 10)
```

In [44]:
```
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
    for j in range(2):
```

```
        axes[i,j].plot(r_business[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].plot(irf_lower[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].plot(irf_upper[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[0,0].axvspan(0, 3, color='gray', alpha=0.3)
axes[0,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[1,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[2,1].axvspan(0, 3, color='gray', alpha=0.3)

axes[0,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[0,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[1,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[2,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.005, 0.015)
axes[0,1].set_ylim(-0.005, 0.015)
axes[1,0].set_ylim(-0.03, 0.03)
axes[1,1].set_ylim(-0.03, 0.03)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.06, 0.06)
axes[3,0].set_ylim(-0.5, 1)
axes[3,1].set_ylim(-0.02, 0.02)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)

axes[1,0].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[1,1].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[2,1].set_yticks([-0.06,-0.03,0,0.03,0.06])

plt.tight_layout()
plt.savefig("business_mc.pdf", format="PDF")
plt.show()
```



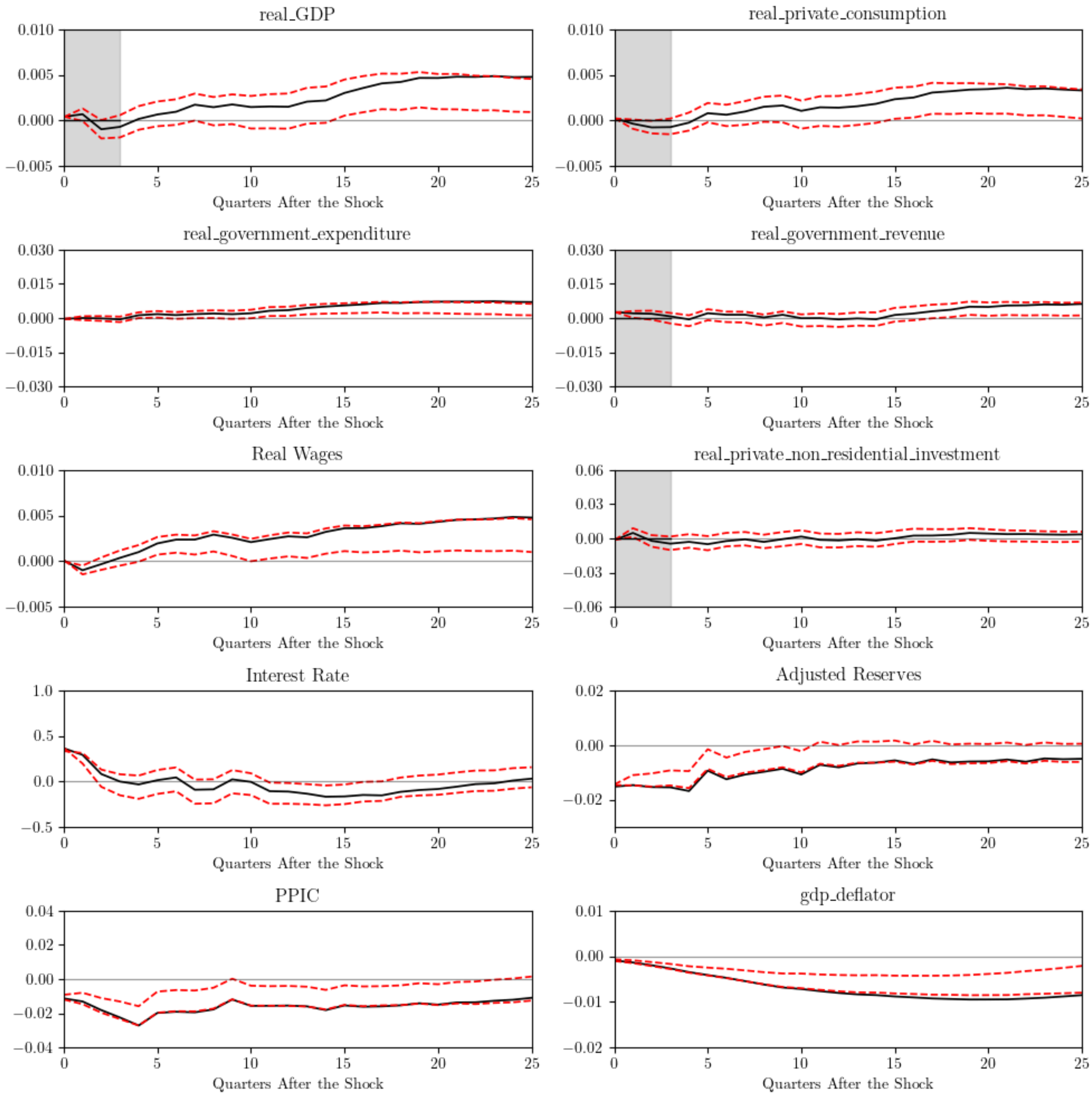
Monetary Shocks

In [45]:
```
irf_lower = np.zeros((25 + 1, 10))
irf_upper = np.zeros((25 + 1, 10))

for k in range(25 + 1):
    for i in range(10):
        cholesky_irf_lower = lower[k, i, :]
        cholesky_irf_upper = upper[k, i, :]


        irf_lower[k, i] = np.dot(cholesky_irf_lower, q_monetary_shocks)
        irf_upper[k, i] = np.dot(cholesky_irf_upper, q_monetary_shocks)
print(irf_lower.shape, irf_upper.shape)
```

```
(26, 10) (26, 10)
```

In [46]:
```
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
```

```
    for j in range(2):
        axes[i,j].plot(r_monetary[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].plot(irf_lower[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].plot(irf_upper[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[0,0].axvspan(0, 3, color='gray', alpha=0.3)
axes[0,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[1,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[2,1].axvspan(0, 3, color='gray', alpha=0.3)

axes[0,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[0,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[1,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[2,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.005, 0.01)
axes[0,1].set_ylim(-0.005, 0.01)
axes[1,0].set_ylim(-0.02, 0.02)
axes[1,1].set_ylim(-0.02, 0.03)
axes[2,0].set_ylim(-0.005, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 1)
axes[3,1].set_ylim(-0.03, 0.02)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.02, 0.01)

axes[1,0].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[1,1].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[2,1].set_yticks([-0.06,-0.03,0,0.03,0.06])

plt.tight_layout()
plt.savefig("monetary_mc.pdf", format="PDF")
plt.show()
```



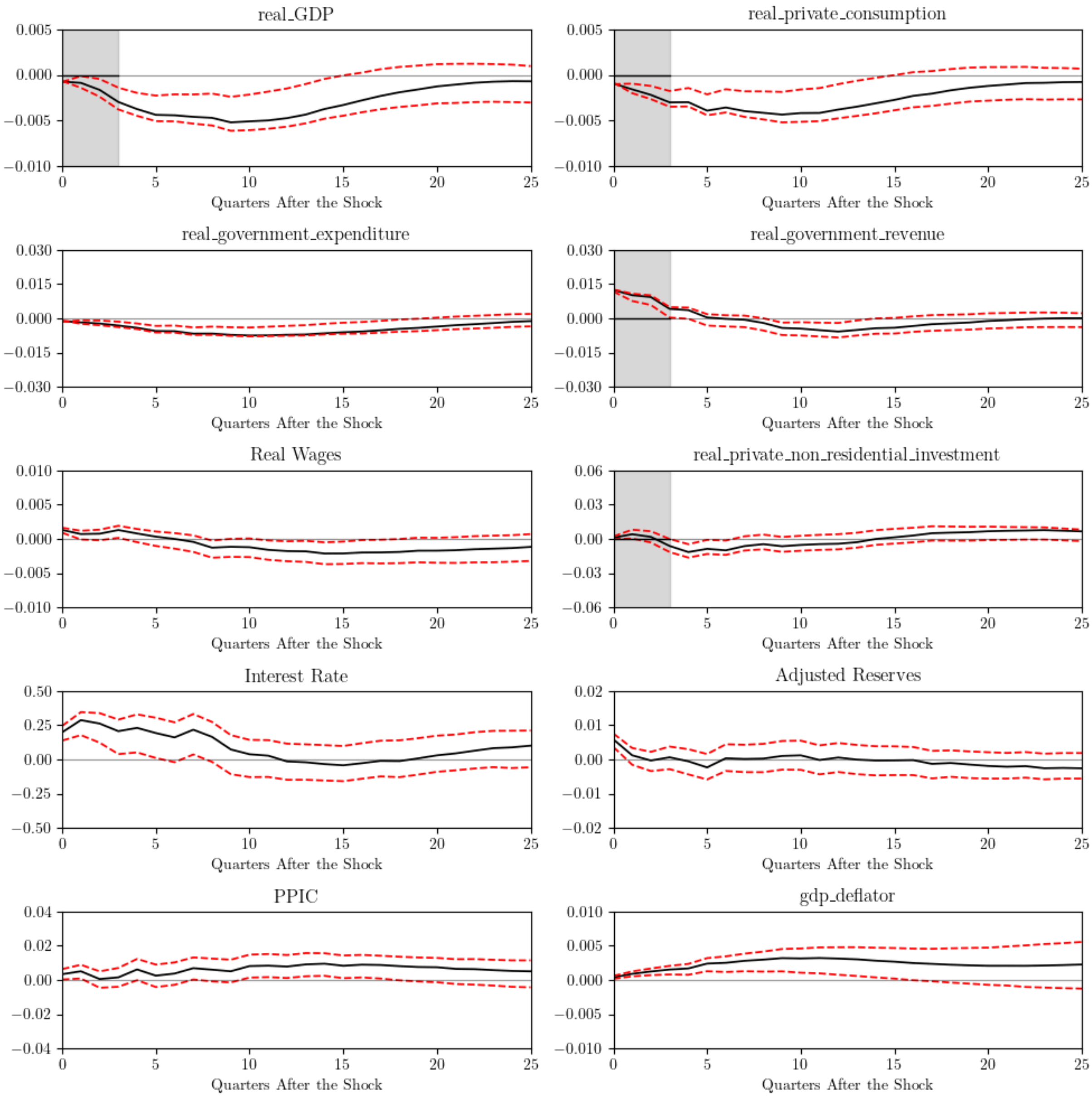
Revenue Shocks

In [47]:
```
irf_lower = np.zeros((25 + 1, 10))
irf_upper = np.zeros((25 + 1, 10))

for k in range(25 + 1):
    for i in range(10):
        cholesky_irf_lower = lower[k, i, :]
        cholesky_irf_upper = upper[k, i, :]


        irf_lower[k, i] = np.dot(cholesky_irf_lower, q_revenue_shocks)
        irf_upper[k, i] = np.dot(cholesky_irf_upper, q_revenue_shocks)
print(irf_lower.shape, irf_upper.shape)
```

```
(26, 10) (26, 10)
```

In [48]:
```
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
```

```
    for j in range(2):
        axes[i,j].plot(r_revenue[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].plot(irf_lower[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].plot(irf_upper[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[0,0].axvspan(0, 3, color='gray', alpha=0.3)
axes[0,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[1,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[2,1].axvspan(0, 3, color='gray', alpha=0.3)

axes[0,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[0,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[1,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[2,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.01, 0.005)
axes[0,1].set_ylim(-0.01, 0.005)
axes[1,0].set_ylim(-0.03, 0.02)
axes[1,1].set_ylim(-0.03, 0.02)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 0.5)
axes[3,1].set_ylim(-0.02, 0.02)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)

axes[1,0].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[1,1].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[2,1].set_yticks([-0.06,-0.03,0,0.03,0.06])

plt.tight_layout()
plt.savefig("revenue_mc.pdf", format="PDF")
plt.show()
```



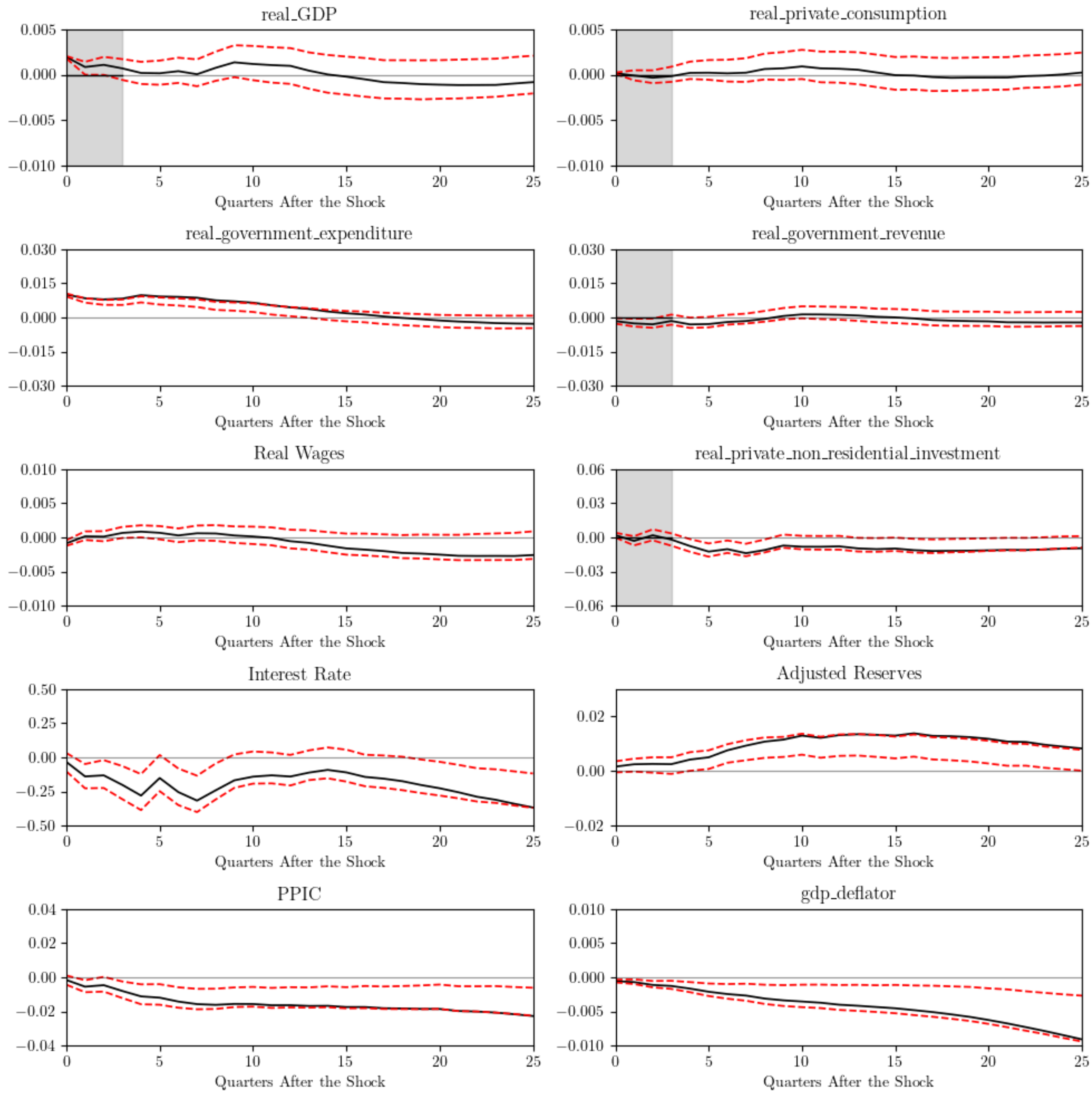
Spending Shocks

In [49]:
```
irf_lower = np.zeros((25 + 1, 10))
irf_upper = np.zeros((25 + 1, 10))

for k in range(25 + 1):
    for i in range(10):
        cholesky_irf_lower = lower[k, i, :]
        cholesky_irf_upper = upper[k, i, :]


        irf_lower[k, i] = np.dot(cholesky_irf_lower, q_spending_shocks)
        irf_upper[k, i] = np.dot(cholesky_irf_upper, q_spending_shocks)
print(irf_lower.shape, irf_upper.shape)
```

```
(26, 10) (26, 10)
```

In [50]:
```
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
```

```
    for j in range(2):
        axes[i,j].plot(r_spending[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].plot(irf_lower[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].plot(irf_upper[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[0,0].axvspan(0, 3, color='gray', alpha=0.3)
axes[0,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[1,1].axvspan(0, 3, color='gray', alpha=0.3)
axes[2,1].axvspan(0, 3, color='gray', alpha=0.3)

axes[0,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[0,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[1,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[2,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.01, 0.005)
axes[0,1].set_ylim(-0.01, 0.005)
axes[1,0].set_ylim(-0.03, 0.02)
axes[1,1].set_ylim(-0.03, 0.02)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 0.5)
axes[3,1].set_ylim(-0.02, 0.03)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)

axes[1,0].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[1,1].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[2,1].set_yticks([-0.06,-0.03,0,0.03,0.06])

plt.tight_layout()
plt.savefig("spending_mc.pdf", format="PDF")
plt.show()
```



Zero-Restriction Revenue Shocks

In [51]:

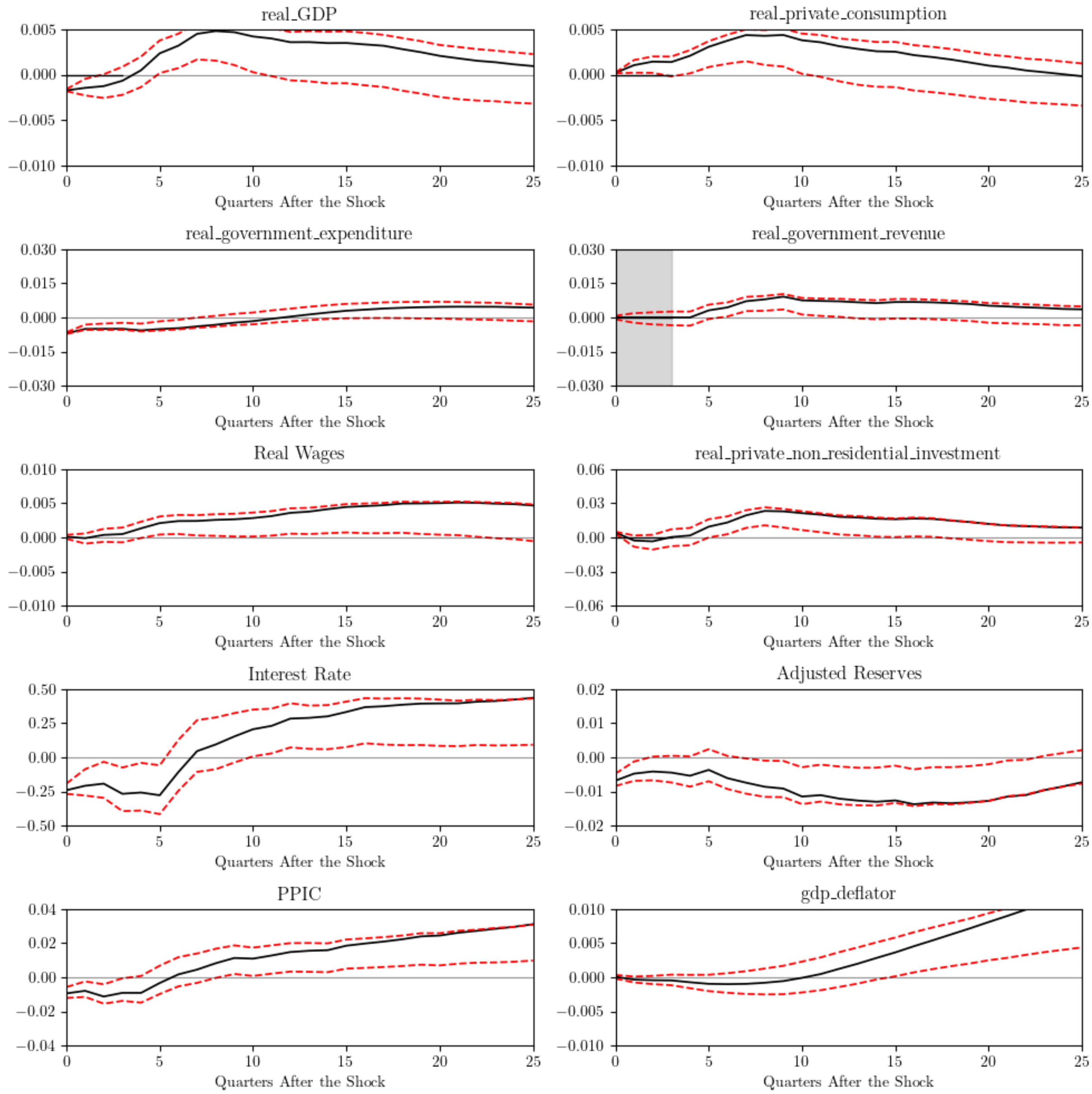
```
irf_lower = np.zeros((25 + 1, 10))
irf_upper = np.zeros((25 + 1, 10))

for k in range(25 + 1):
    for i in range(10):
        cholesky_irf_lower = lower[k, i, :]
        cholesky_irf_upper = upper[k, i, :]


        irf_lower[k, i] = np.dot(cholesky_irf_lower, q_anticipated_revenue_shocks)
        irf_upper[k, i] = np.dot(cholesky_irf_upper, q_anticipated_revenue_shocks)
print(irf_lower.shape, irf_upper.shape)
```

```
(26, 10) (26, 10)
```

In [52]:

```
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
```

```
    for j in range(2):
        axes[i,j].plot(r_anticipated_revenue[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].plot(irf_lower[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].plot(irf_upper[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[1,1].axvspan(0, 3, color='gray', alpha=0.3)

axes[0,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[0,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[1,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[2,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.01, 0.005)
axes[0,1].set_ylim(-0.01, 0.005)
axes[1,0].set_ylim(-0.03, 0.02)
axes[1,1].set_ylim(-0.03, 0.02)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 0.5)
axes[3,1].set_ylim(-0.02, 0.02)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)

axes[1,0].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[1,1].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[2,1].set_yticks([-0.06,-0.03,0,0.03,0.06])

plt.tight_layout()
plt.savefig("anticipated_spending_mc.pdf", format="PDF")
plt.show()
```



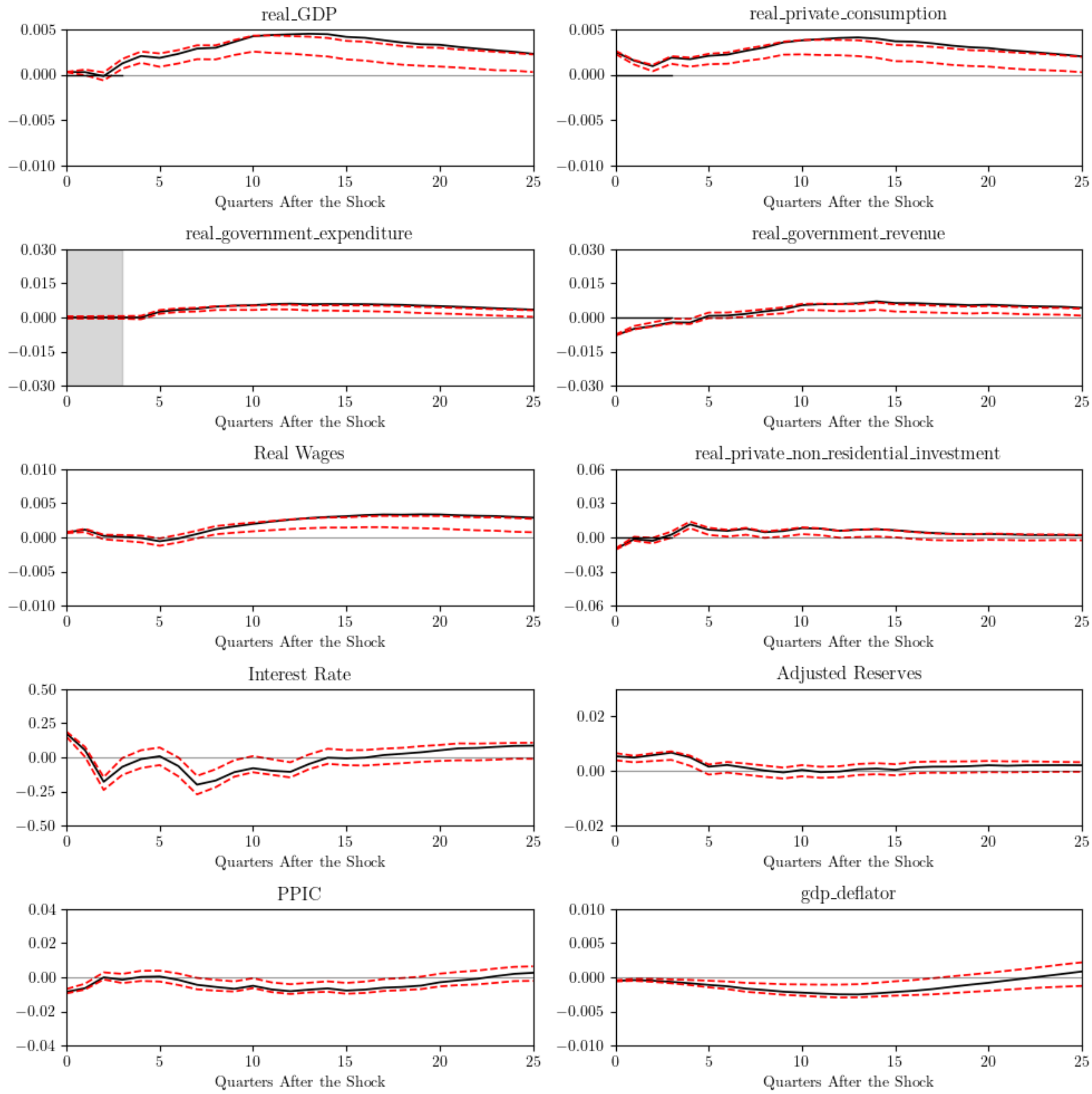
Zero-Restriction Spending Shocks

In [53]:
```
irf_lower = np.zeros((25 + 1, 10))
irf_upper = np.zeros((25 + 1, 10))

for k in range(25 + 1):
    for i in range(10):
        cholesky_irf_lower = lower[k, i, :]
        cholesky_irf_upper = upper[k, i, :]


        irf_lower[k, i] = np.dot(cholesky_irf_lower, q_anticipated_spending_shocks)
        irf_upper[k, i] = np.dot(cholesky_irf_upper, q_anticipated_spending_shocks)
print(irf_lower.shape, irf_upper.shape)
```

```
(26, 10) (26, 10)
```

In [54]:
```
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
```

```
    for j in range(2):
        axes[i,j].plot(r_anticipated_spending[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].plot(irf_lower[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].plot(irf_upper[:, i*2+j], color = "red", ls="dashed", lw=1.25)
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[1,0].axvspan(0, 3, color='gray', alpha=0.3)


axes[0,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[0,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[1,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[2,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.01, 0.005)
axes[0,1].set_ylim(-0.01, 0.005)
axes[1,0].set_ylim(-0.03, 0.02)
axes[1,1].set_ylim(-0.03, 0.02)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 0.5)
axes[3,1].set_ylim(-0.02, 0.03)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)

axes[1,0].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[1,1].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[2,1].set_yticks([-0.06,-0.03,0,0.03,0.06])

plt.tight_layout()
plt.savefig("anticipated_spending_mc.pdf", format="PDF")
plt.show()
```



Extention 4 - Policy Analysis

In [55]:
```
cashe = get_r_ja(var_res,q_revenue_shocks,4)
gov_spending_by_revenue = cashe[:4, 2]
gov_revenue_by_revenue = cashe[:4, 3]
print(gov_spending_by_revenue)
```

```
[-0.00121186 -0.00178498 -0.00228191 -0.00314356]
```

In [56]:
```
cashe = get_r_ja(var_res,q_spending_shocks,4)
gov_spending_by_spending = cashe[:4, 2]
gov_revenue_by_spending = cashe[:4, 3]
print(gov_spending_by_spending)
```

```
[0.01004194 0.00843614 0.0079416  0.00828326]
```

In [57]:
```
A_spending_gs = np.array([
    [gov_spending_by_spending[0], 0, 0, 0],
    [gov_spending_by_spending[1], gov_spending_by_spending[0], 0, 0],
```

```
    [gov_spending_by_spending[2], gov_spending_by_spending[1], gov_spending_by_spen
    [gov_spending_by_spending[3], gov_spending_by_spending[2], gov_spending_by_spen
])

A_spending_gr = np.array([
    [gov_spending_by_revenue[0], 0, 0, 0],
    [gov_spending_by_revenue[1], gov_spending_by_revenue[0], 0, 0],
    [gov_spending_by_revenue[2], gov_spending_by_revenue[1], gov_spending_by_revenu
    [gov_spending_by_revenue[3], gov_spending_by_revenue[2], gov_spending_by_revenu
])

A_revenue_gs = np.array([
    [gov_revenue_by_spending[0], 0, 0, 0],
    [gov_revenue_by_spending[1], gov_revenue_by_spending[0], 0, 0],
    [gov_revenue_by_spending[2], gov_revenue_by_spending[1], gov_revenue_by_spendin
    [gov_revenue_by_spending[3], gov_revenue_by_spending[2], gov_revenue_by_spendin
])

A_revenue_gr = np.array([
    [gov_revenue_by_revenue[0], 0, 0, 0],
    [gov_revenue_by_revenue[1], gov_revenue_by_revenue[0], 0, 0],
    [gov_revenue_by_revenue[2], gov_revenue_by_revenue[1], gov_revenue_by_revenue[0
    [gov_revenue_by_revenue[3], gov_revenue_by_revenue[2], gov_revenue_by_revenue[1
])


A_full_8d = np.block([
    [A_spending_gs, A_spending_gr],
    [A_revenue_gs, A_revenue_gr]
])

b_8d = np.concatenate([
    np.full(4, 0.01),
    np.zeros(4)
])

solution_8d = np.linalg.solve(A_full_8d, b_8d)

shock_labels_8d = [f"BGS_{i}" for i in range(4)] + [f"BGR_{i}" for i in range(4)]
solution_8d_series = pd.Series(solution_8d, index=shock_labels_8d)
solution_8d_series
```

Out[57]:

```
BGS_0    1.011525
BGS_1    0.184709
BGS_2    0.104465
BGS_3    0.011562
BGR_0    0.130111
BGR_1    0.128688
BGR_2    0.093888
BGR_3   -0.022494
dtype: float64
```

In [58]:

```
irfs_revenue = get_r_ja(var_res, q_revenue_shocks, 30)
irfs_spending = get_r_ja(var_res, q_spending_shocks, 30)

BGS_j = solution_8d_series.iloc[:4].values
```

```
BGR_j = solution_8d_series.iloc[4:].values

BGS_extended = np.zeros(30)
BGS_extended[:4] = BGS_j

BGR_extended = np.zeros(30)
BGR_extended[:4] = BGR_j

response_spending = np.zeros((30, 10))
response_revenue = np.zeros((30, 10))

for k in range(30):
    for j in range(k + 1):
        response_spending[k] += BGS_extended[j] * irfs_spending[k - j]
        response_revenue[k] += BGR_extended[j] * irfs_revenue[k - j]

response_total = response_spending + response_revenue
```

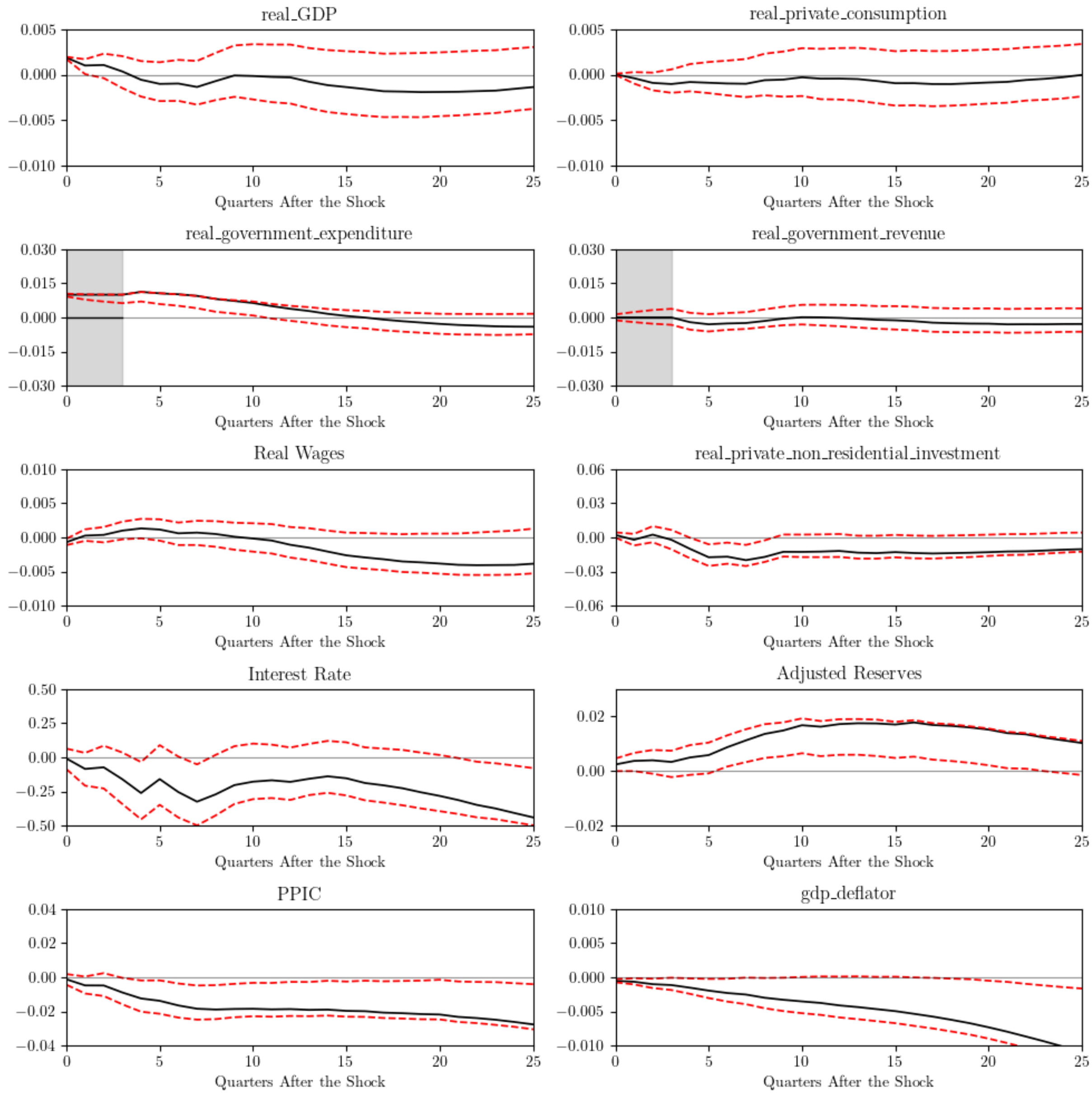
In [59]:
```
irf_revenue_lower = np.zeros((30, 10))
irf_revenue_upper = np.zeros((30, 10))
irf_spending_lower = np.zeros((30, 10))
irf_spending_upper = np.zeros((30, 10))


for k in range(30):
    for i in range(10):
        cholesky_irf_lower = lower[k, i, :]
        cholesky_irf_upper = upper[k, i, :]


        irf_revenue_lower[k, i] = np.dot(cholesky_irf_lower, q_revenue_shocks)
        irf_revenue_upper[k, i] = np.dot(cholesky_irf_upper, q_revenue_shocks)

        irf_spending_lower[k, i] = np.dot(cholesky_irf_lower, q_spending_shocks)
        irf_spending_upper[k, i] = np.dot(cholesky_irf_upper, q_spending_shocks)
print(irf_revenue_lower.shape)
```

```
(30, 10)
```

In [60]:
```
response_lower = np.zeros((30, 10))
response_upper = np.zeros((30, 10))

for k in range(30):
    for j in range(k + 1):
        response_lower[k] += (
            BGS_extended[j] * irf_spending_lower[k - j]
            + BGR_extended[j] * irf_revenue_lower[k - j]
        )
        response_upper[k] += (
            BGS_extended[j] * irf_spending_upper[k - j]
            + BGR_extended[j] * irf_revenue_upper[k - j]
        )
```

In [61]:
```
fig, axes = plt.subplots(5,2, figsize=(10, 10))
for i in range(5):
    for j in range(2):
```

```
        axes[i,j].plot(response_total[:, i*2+j], color = "black", lw=1.25)
        axes[i,j].plot(response_lower[:, i*2+j], color = "red", ls="dashed", lw=1.2
        axes[i,j].plot(response_upper[:, i*2+j], color = "red", ls="dashed", lw=1.2
        axes[i,j].axhline(y=0, color = "black", alpha=0.4, lw=0.8)
        axes[i,j].set_xlim(0,25)
        axes[i,j].set_title(names[i*2+j], fontsize=12)
        axes[i,j].set_xlabel("Quarters After the Shock", fontsize=10)

axes[1,0].axvspan(0, 3, color='gray', alpha=0.3)
axes[1,1].axvspan(0, 3, color='gray', alpha=0.3)

axes[1,1].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)
axes[1,0].axhline(y=0, xmin=0, xmax=0.12, color = "black",  lw=1)

axes[0,0].set_ylim(-0.01, 0.005)
axes[0,1].set_ylim(-0.01, 0.005)
axes[1,0].set_ylim(-0.03, 0.02)
axes[1,1].set_ylim(-0.03, 0.02)
axes[2,0].set_ylim(-0.01, 0.01)
axes[2,1].set_ylim(-0.04, 0.04)
axes[3,0].set_ylim(-0.5, 0.5)
axes[3,1].set_ylim(-0.02, 0.03)
axes[4,0].set_ylim(-0.04, 0.04)
axes[4,1].set_ylim(-0.01, 0.01)

axes[1,0].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[1,1].set_yticks([-0.03,-0.015,0,0.015,0.03])
axes[2,1].set_yticks([-0.06,-0.03,0,0.03,0.06])

plt.tight_layout()
plt.savefig("deficit_spending_mc.pdf", format="PDF")
plt.show()
```


real_GDP
Quarters After the Shock
real_private_consumption
Quarters After the Shock
real_government_expenditure
Quarters After the Shock
real_government_revenue
Quarters After the Shock
Real Wages
Quarters After the Shock
real_private_non_residential_investment
Quarters After the Shock
Interest Rate
Quarters After the Shock
Adjusted Reserves
Quarters After the Shock
PPIC
Quarters After the Shock
gdp_deflator
Quarters After the Shock