dimensionality reduction

PCA of Snow Depth in MA

```
In [1]: state='MA'
        meas='SNWD'
```

Compute sequence of residuals

Residuals are the remainder left after successive approximations:

1) Original vector = $\vec{v}$

2) $\overrightarrow{r_0} = \vec{v} - \vec{\mu}$

3) $\overrightarrow{r_1} = \overrightarrow{r_0} - (\vec{v} \cdot \overrightarrow{u_1})\, u_1$

4) $\overrightarrow{r_2} = \overrightarrow{r_1} - (\vec{v} \cdot \overrightarrow{u_2})\, u_2$

5) $\overrightarrow{r_3} = \overrightarrow{r_0} - (\vec{v} \cdot \overrightarrow{u_3})\, u_3$

6) .....

For each residual $\vec{r_i}$ we compute its square norm, which we will refer to as **residual norm**:

$$\| \vec{r_i} \|_2^2 = \sum_{j=1}^{n} (r_{i,j})^2$$

The smaller the norm, the better the approximation.

A few things we know from linear algebra:

1) The zero'th residual norm is the square distance of $\vec{v}$ from the mean $\vec{\mu}$

2) The k'th residual norm is the minimal square between $\vec{v}$ and a point that can be expressed as

$$\overrightarrow{w_k} = \vec{\mu} + \sum_{i=1}^{k} c_i \overrightarrow{u_i}$$

Where $c_1, \ldots, c_k$ are arbitrarily real numbers. We call $\overrightarrow{w_k}$ the k'th approximation of the reconstruction of $\vec{v}$

3) The residual norms are non-increasing

4) The residual vector $\overrightarrow{r_n}$ is the zero vector. In other words, $\overrightarrow{w_n} = \vec{v}$

`decompose_dataframe` extracts the series from the row, computers the *k* to decomposition coefficients and the square norm of the residuals and constructs a new row that is reassembled into a new dataframe.

For more details, use `%load lib/decomposer.py`

Let’s do some decompositions!

In [9]:
```
m='SNWD'
state='MA'
```

In [10]:
```
%%time
#get mean and eigenvectors for measurement m
EigVec=STAT[m]['eigvec']
Mean=STAT[m]['Mean']
EigVec.shape

Query="""
select *
from weather
WHERE state='%s' and measurement='%s'
"""%(state,m)
df=sqlContext.sql(Query)
print('number of records=',df.count())
df.show(3)


k=5
df2=decompose_dataframe(sqlContext,df,EigVec[:,:k],Mean).cache() # Make it possible to generate only first k coefficients.
```

```
number of records= 8750
+-----------+-----------+----+--------------------+--------+---------+---------+-------------+-----------------+-----+-------------+
|    Station|Measurement|Year|              Values|latitude|longitude|elevation|    dist2coast|             name|state|      country|
+-----------+-----------+----+--------------------+--------+---------+---------+-------------+-----------------+-----+-------------+
|US1MABA0018|       SNWD|2022|[00 00 00 00 19 F...| 41.5818| -70.5257|      9.8|0.94091796875|  WAQUOIT 0.6 SSW|   MA|United States|
|US1MABA0060|       SNWD|2022|[00 00 00 00 00 0...| 41.6581| -70.3077|     14.0| 2.849609375|  HYANNIS 0.7 WNW|   MA|United States|
|US1MAES0004|       SNWD|2022|[00 00 00 00 00 0...| 42.7471| -71.0426|      8.2|          NaN|GROVELAND 0.5 WSW|   MA|United States|
+-----------+-----------+----+--------------------+--------+---------+---------+-------------+-----------------+-----+-------------+
only showing top 3 rows

CPU times: user 11.1 ms, sys: 2.55 ms, total: 13.6 ms
Wall time: 1.95 s
```

In [10]:
```
%%time
#get mean and eigenvectors for measurement m
EigVec=STAT[m]['eigvec']
Mean=STAT[m]['Mean']
EigVec.shape

Query="""
select *
from weather
WHERE state='%s' and measurement='%s'
"""%(state,m)
df=sqlContext.sql(Query)
print('number of records=',df.count())
df.show(3)


k=5
df2=decompose_dataframe(sqlContext,df,EigVec[:,:k],Mean).cache() # Make it possible to generate only first k coefficients.
```

```
number of records= 8750
+-----------+-----------+----+--------------------+--------+---------+---------+-------------+------------------+-----+-------------+
|    Station|Measurement|Year|              Values|latitude|longitude|elevation|    dist2coast|              name|state|      country|
+-----------+-----------+----+--------------------+--------+---------+---------+-------------+------------------+-----+-------------+
|US1MABA0018|       SNWD|2022|[00 00 00 00 19 F...| 41.5818| -70.5257|      9.8|0.94091796875|  WAQUOIT 0.6 SSW|   NA|United States|
|US1MABA0060|       SNWD|2022|[00 00 00 00 00 0...| 41.6581| -70.3077|     14.0|  2.849609375|  HYANNIS 0.7 WNW|   NA|United States|
|US1MAES0004|       SNWD|2022|[00 00 00 00 00 0...| 42.7471| -71.0426|      8.2|          NaN|GROVELAND 0.5 WSW|   NA|United States|
+-----------+-----------+----+--------------------+--------+---------+---------+-------------+------------------+-----+-------------+
only showing top 3 rows

CPU times: user 11.1 ms, sys: 2.55 ms, total: 13.6 ms
Wall time: 1.95 s
```

In [11]:
```
%%time
print(df2.count())
```

```
8750
CPU times: user 24.9 ms, sys: 12.8 ms, total: 37.7 ms
Wall time: 1min 40s
```

Join decomposition information with station information

In [12]:
```
col=df2.columns

Dcol={}
Dcol['station,year,measurement info']=col[:4]+col[9:14]+[col[21]]
Dcol['coeff']=col[4:9]
Dcol['residuals']=[col[22]]+[col[20]]+col[15:20]
for key in Dcol:
    print('==',key,'==')
    print(','.join(Dcol[key]))
```

```
-- station,year,measurement info --
Station,Measurement,Year,Values,state,country,total_var,res_mean,res_1,res_5
-- coeff --
latitude,longitude,elevation,dist2coast,name
== residuals ==
coeff_5,coeff_4,res_2,coeff_2,res_3,coeff_3,res_4
```

Removing years with little snow

In some locations and in some years, there is almost no snow accumulation. We want to treat these separately.

To do so we compare the error of using the average to the error of using a zero vector. We keep only those yearXstation where the mean is a better approximation than the zero Vector.

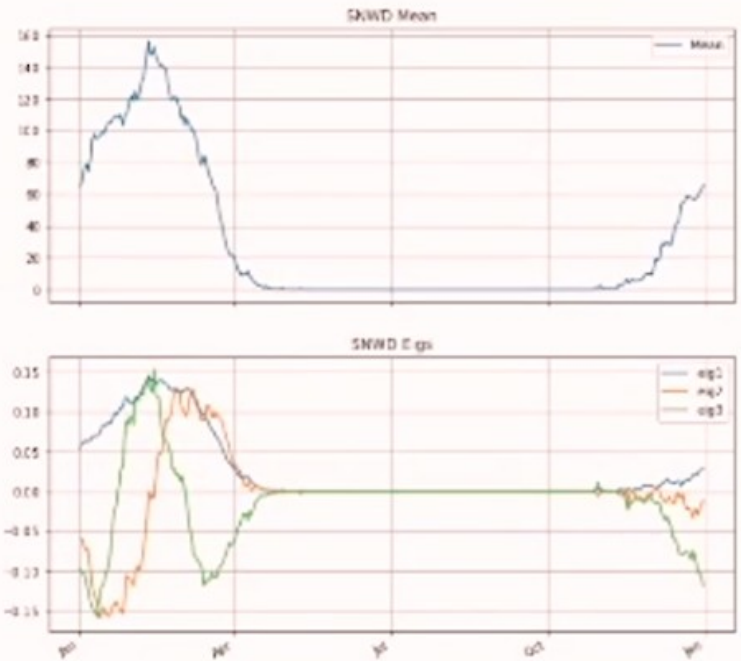
```
In [13]: %%time
         #filter out vectors for which the mean is a worse approximation than zero.
         print('all Rows',df2.count())
         df3=df2.filter(df2.res_mean<1)
         print('Rows where mean is better approx than zero',df3.count())

         all Rows 8750
         Rows where mean is better approx than zero 4172
         CPU times: user 2.82 ms, sys: 3.41 ms, total: 6.23 ms
         Wall time: 358 ms
```

Plot mean and top eigenvectors

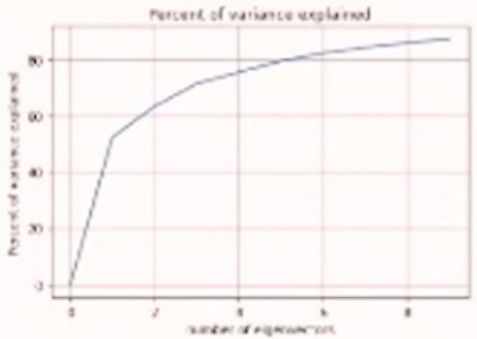
Construct approximation of a time series using the mean and the *k* top-eigen-vectors. First, we plot the mean and the top *k* eigenvectors.

In [14]:
```
import pylab as plt
fig,axes=plt.subplots(2,1, sharex='col', sharey='row',figsize=(10,10));
k=3
EigVec=np.array(STAT[m]['eigvec'][:,:k])
Mean=STAT[m]['Mean']
YearPlotter().plot(Mean,fig,axes[0],label='Mean',title=m+' Mean')
YearPlotter().plot(EigVec,fig,axes[1],title=m+' Eigs',labels=['eig'+str(i+1) for i in range(k)])
fig.savefig('r_figures/SNWD_mean_eigs')
```



plot Percentage of variance explained

In [15]:
```
#  x=0 in the graphs below correspond to the fraction of the variance explained by the mean alone
#  x=1,2,3,... are the residuals for eig1, eig1+eig2, eig1+eig2+eig3 ...
fig,ax=plt.subplots(1,1);
eigvals=STAT[m]['eigval']; eigvals/=sum(eigvals); cumvar=np.cumsum(eigvals); cumvar=100*np.insert(cumvar,0,0)
ax.plot(cumvar[:10]);
ax.grid();
ax.set_ylabel('Percent of variance explained')
ax.set_xlabel('number of eigenvectors')
ax.set_title('Percent of variance explained');
```



Exploring the decomposition

Intuitive analysis

- **Eig1** is very similar to the Mean --- Indicates heavy/light snow
- If **coef_1** is large: snow accumulation is higher.

- **Eig2** is a negative January, positive march. Indicates early vs. late season
- If **coef_2** is high: snow season is late.

- **Eig3** is positive Feb, negative Jan, March -- Indicates a short or long season.
- If **Coef_3** is high: Season is short.

Studying the effect of Coefficient 1

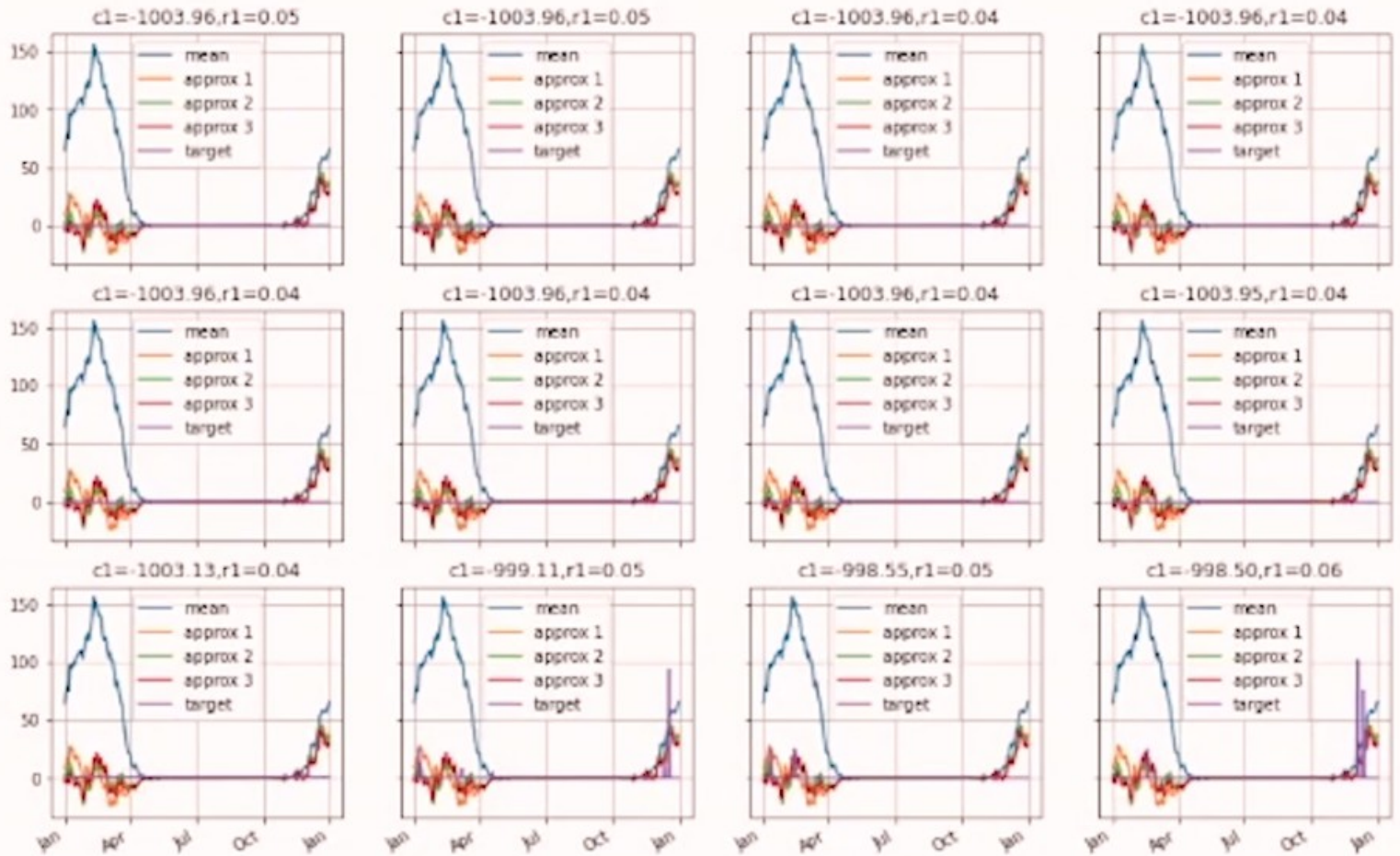
In [18]:
```
# Checking that res_2 is smaller than 0.1 and that rows are sorted based on coeff_2
df4=df2.filter(df2.res_1<0.1).sort(df2.coeff_1)
print(df4.count())
coeff1_rows=df4.collect()
rows=coeff1_rows[:12]
```

```
241
```

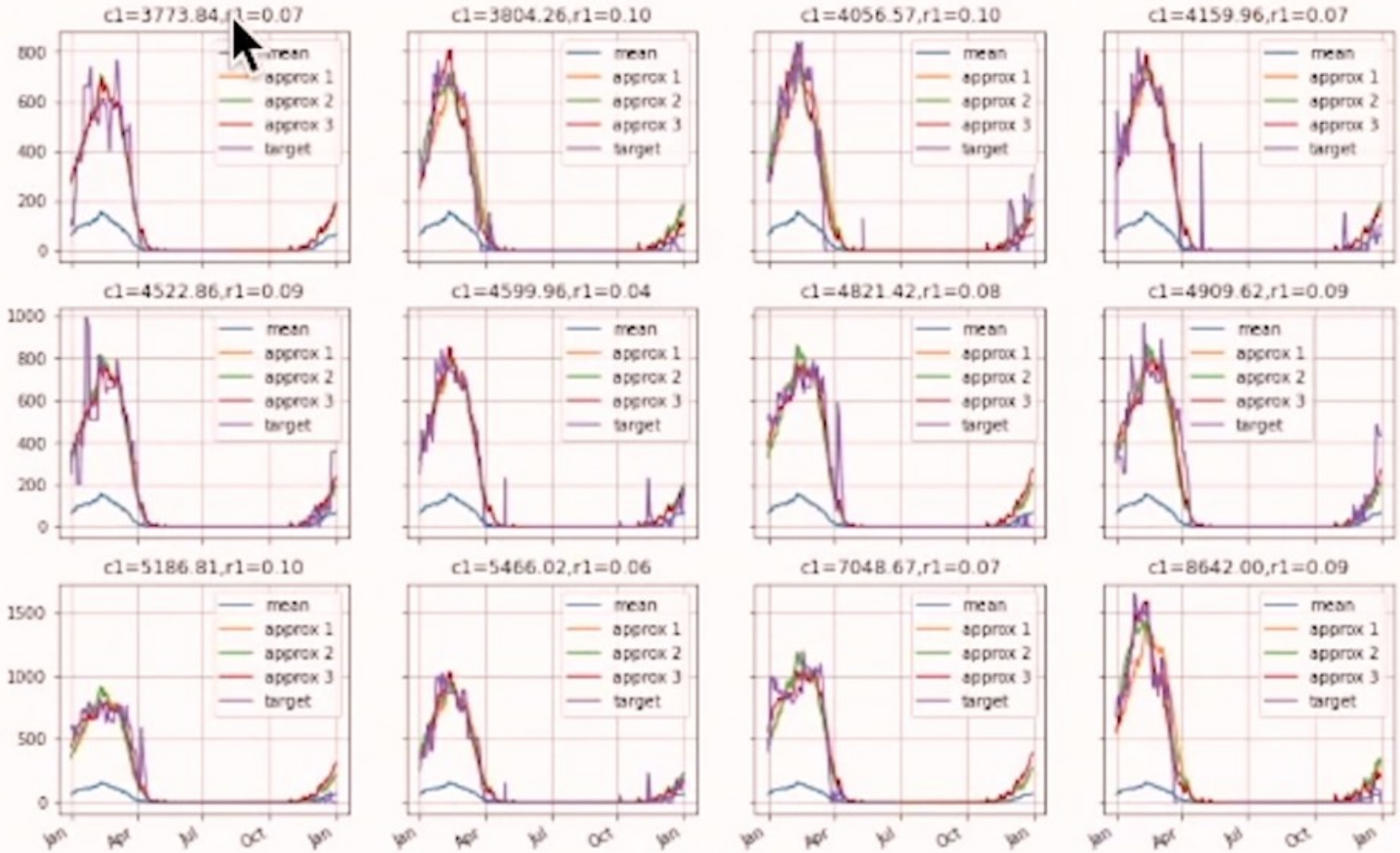
In [19]:
```
df4.select('coeff_1','coeff_2','coeff_3','res_1','res_2','res_3',).show(n=4,truncate=14)
```

```
+--------------+--------------+--------------+--------------+--------------+--------------+
|       coeff_1|       coeff_2|       coeff_3|         res_1|         res_2|         res_3|
+--------------+--------------+--------------+--------------+--------------+--------------+
|-1003.96473...|122.1904414...|59.06081741...|0.045953264...|0.031749198...|0.028274253...|
|-1003.96473...|122.1904414...|59.06081741...|0.045953264...|0.031749198...|0.028274253...|
|-1003.96152...|121.3460782...|54.50636963...|0.044944415...|0.030897039...|0.027799273...|
|-1003.96152...|121.3460782...|54.50636963...|0.044944415...|0.030897039...|0.027799273...|
+--------------+--------------+--------------+--------------+--------------+--------------+
only showing top 4 rows
```

In [20]:
```
plot_recon_grid(coeff1_rows[0:12],Mean,EigVec,header='c1=%3.2f,r1=%3.2f',params=('coeff_1','res_1'))
```



In [21]:
```
plot_recon_grid(coeff1_rows[-12:],Mean,EigVec,header='c1=%3.2f,r1=%3.2f',params=('coeff_1','res_1'))
```



Studying the effect of Coefficient 2

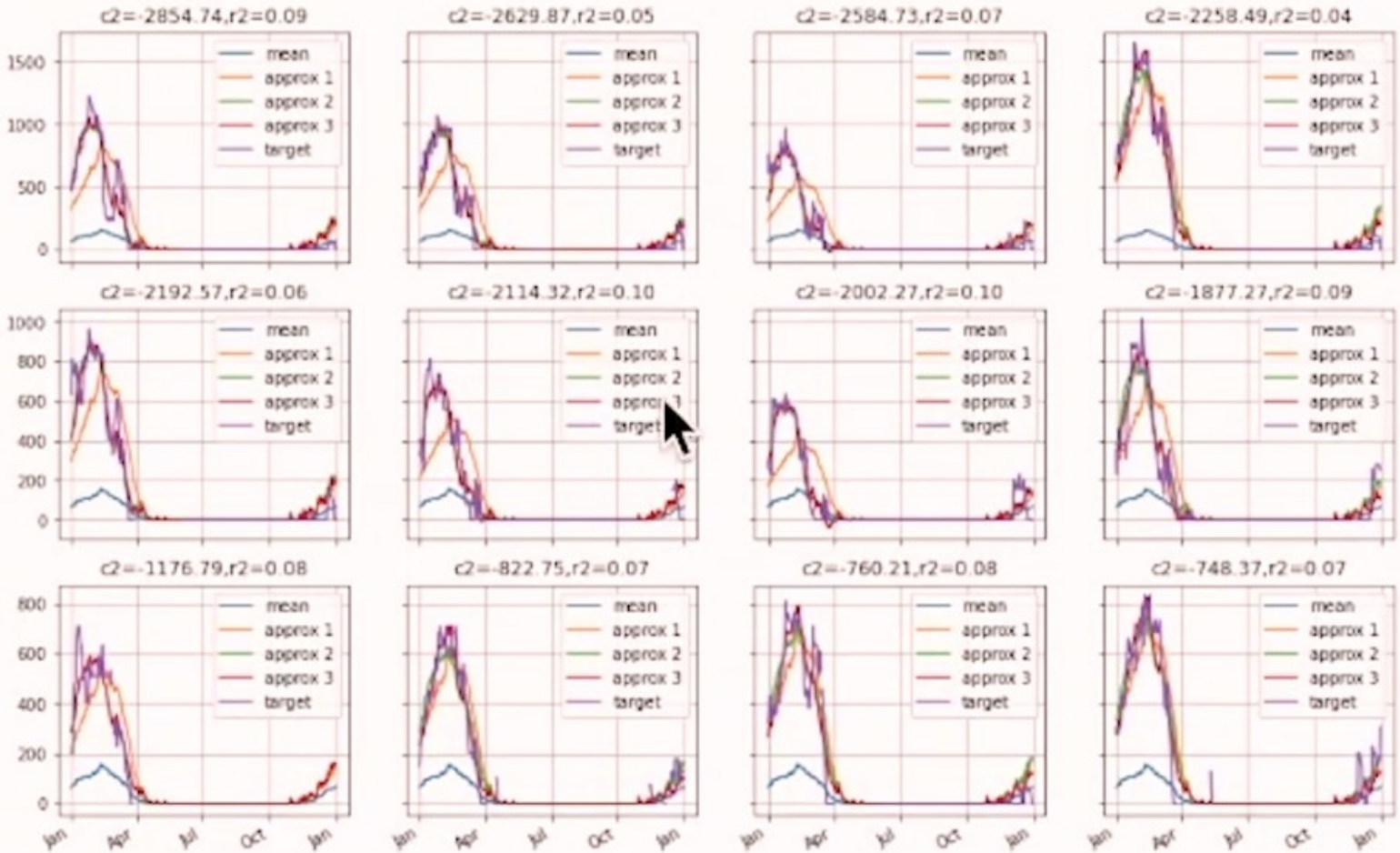
In [22]:
```
# Checking that res_2 is smaller than 0.1 and that rows are sorted based on coeff_2
df4=df2.filter(df2.res_2<0.1).sort(df2.coeff_2)
print(df4.count())
coeff2_rows=df4.collect()
rows=coeff2_rows[:12]
```

```
354
```

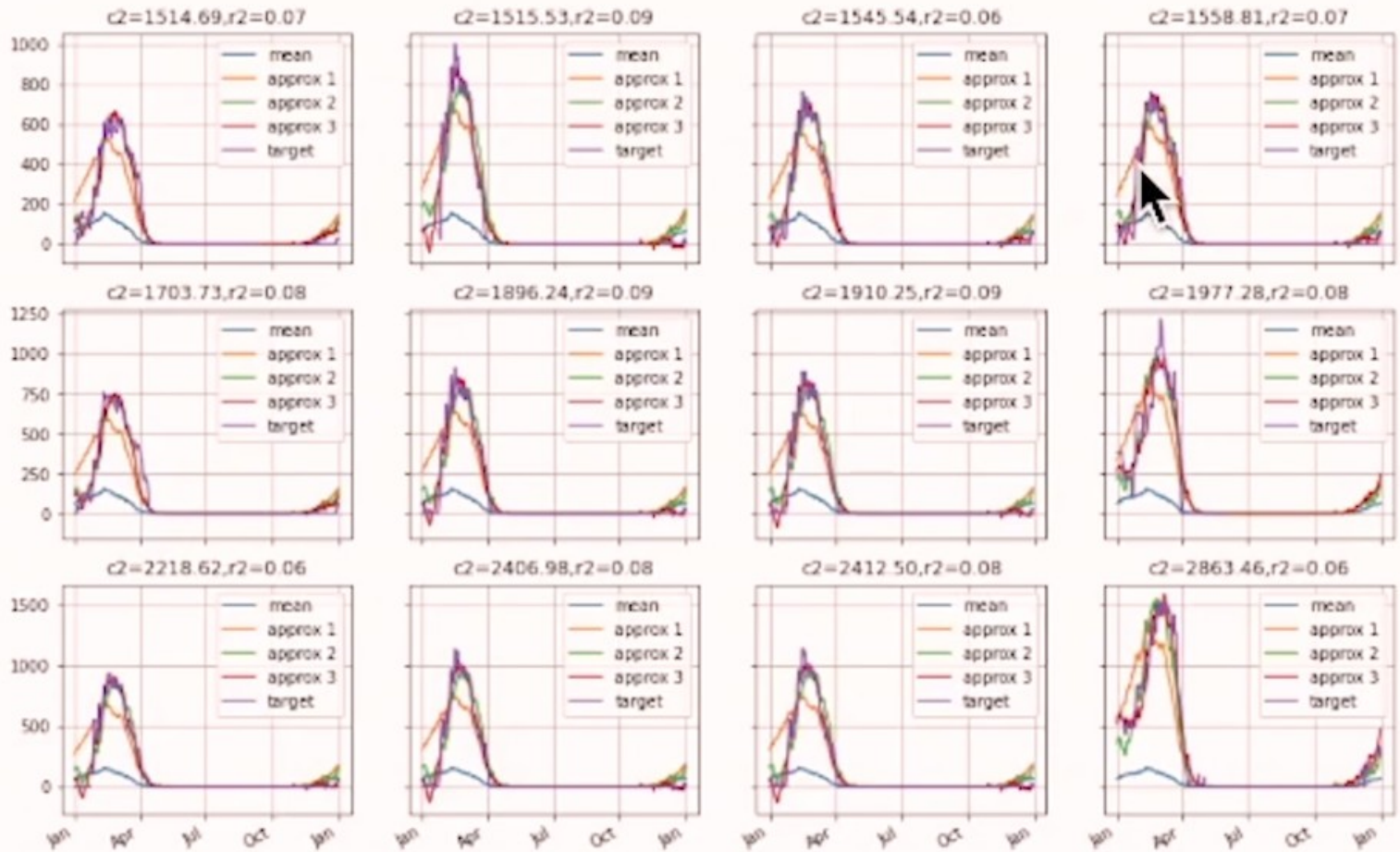
In [23]:
```
df4.select('coeff_1','coeff_2','coeff_3','res_1','res_2','res_3',).show(n=4,truncate=14)
```

```
+--------------+--------------+--------------+--------------+--------------+--------------+
|       coeff_1|       coeff_2|       coeff_3|         res_1|         res_2|         res_3|
+--------------+--------------+--------------+--------------+--------------+--------------+
|4669.911393...|-2854.73871...|2127.3300801...|0.337034996...|0.090424964...|0.089186776...|
|4414.914018...|-2629.86548...|354.4145977...|0.278723925...|0.048887022...|0.044874400...|
|3044.966024...|-2584.72806...|-86.5652329...|0.460027113...|0.070949028...|0.070512619...|
|8642.004188...|-2258.48545...|1122.222425...|0.091368165...|0.037353296...|0.025544162...|
+--------------+--------------+--------------+--------------+--------------+--------------+
only showing top 4 rows
```

In [24]:
```
plot_recon_grid(coeff2_rows[:12],Mean,EigVec,header='c2=%3.2f,r2=%3.2f',params=('coeff_2','res_2'))
```



In [25]:
```
plot_recon_grid(coeff2_rows[-12:],Mean,EigVec,header='c2=%3.2f,r2=%3.2f',params=('coeff_2','res_2'))
```



Studying the effect of Coefficient 3

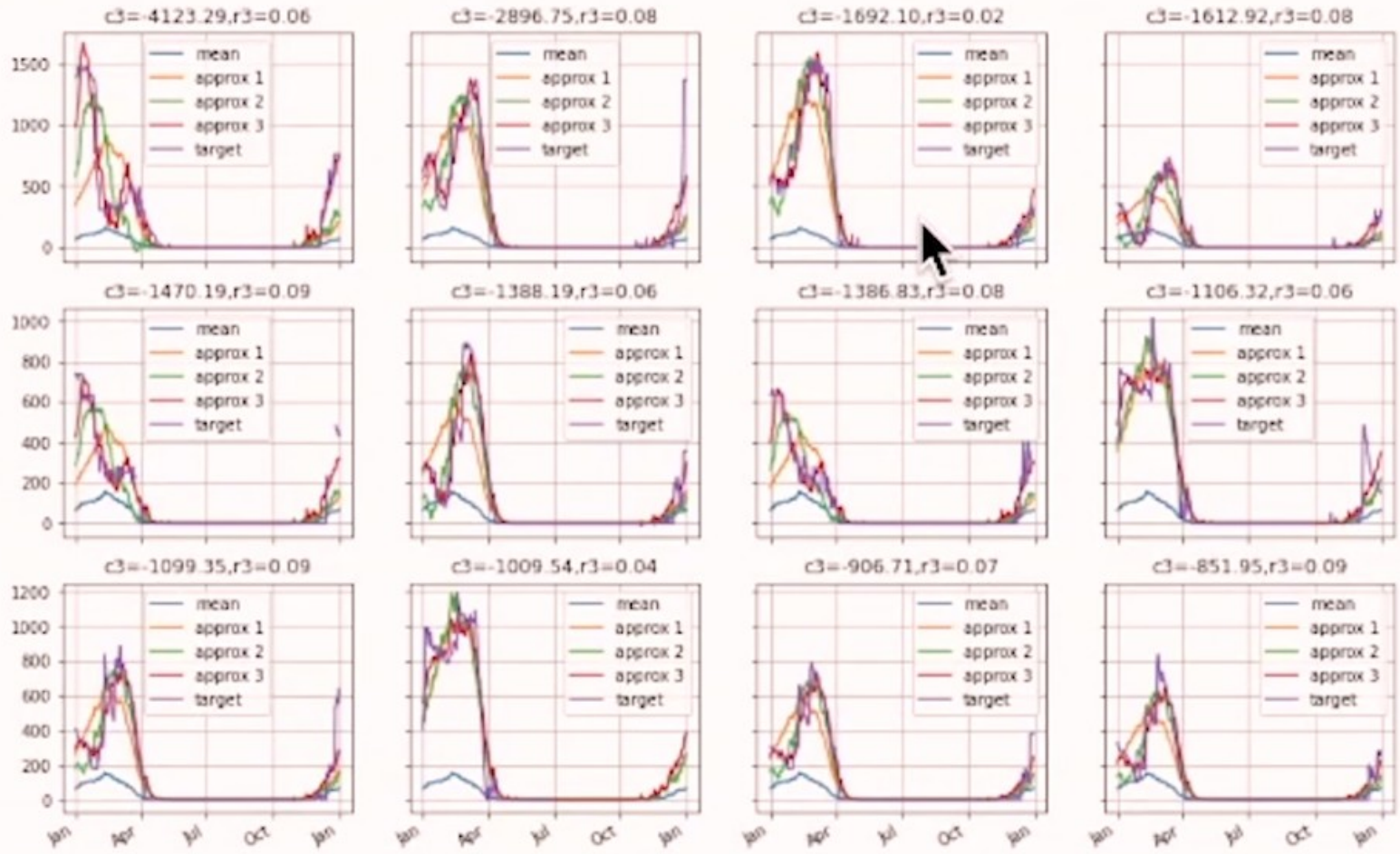
In [26]:
```
df4=df2.filter(df2.res_3<0.1).sort(df2.coeff_3)
print(df4.count())
coeff3_rows=df4.collect()
rows=coeff3_rows[:12]
```

```
536
```

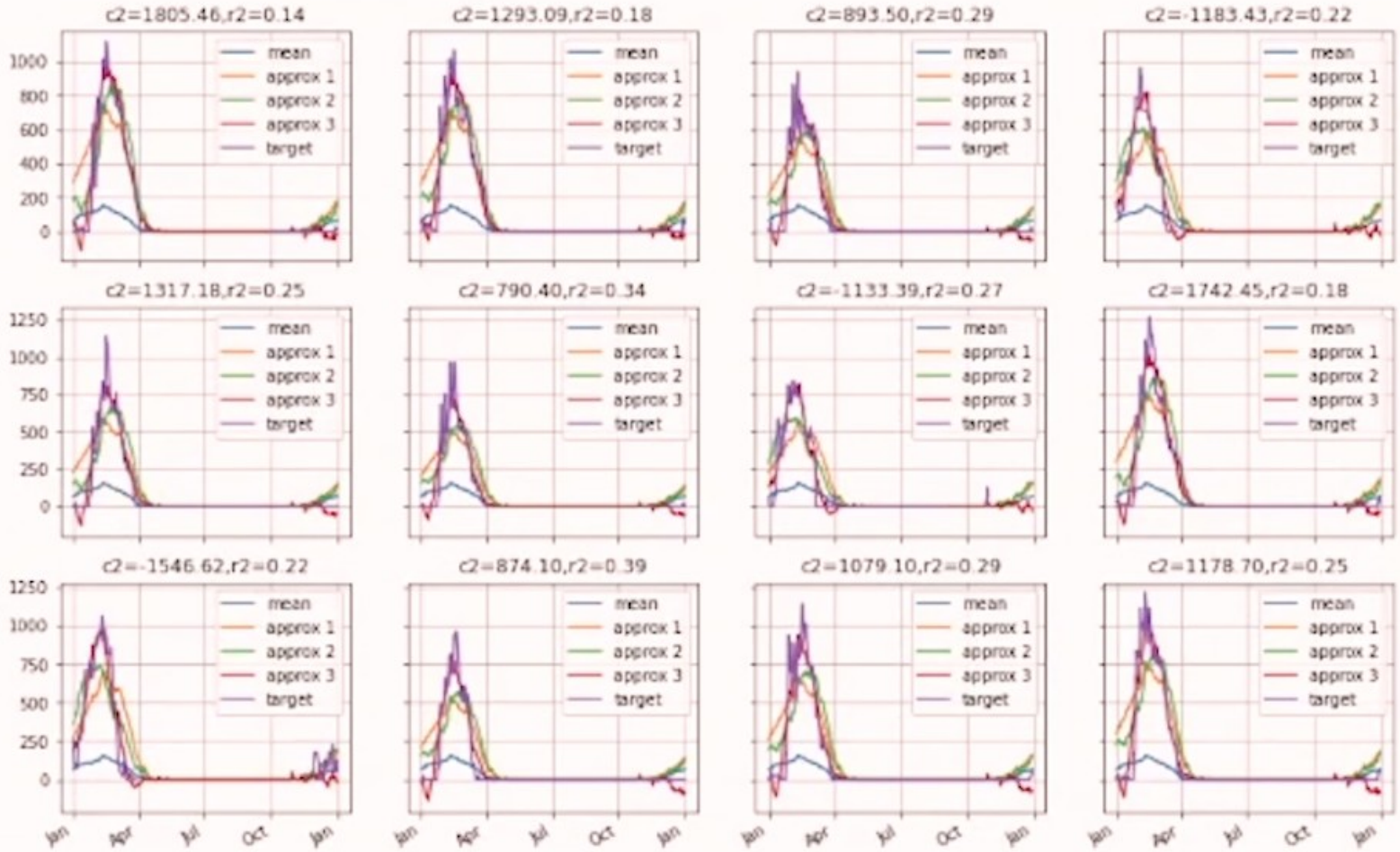
In [27]:
```
df4.select('coeff_1','coeff_2','coeff_3','res_1','res_2','res_3',).show(n=4,truncate=14)
```

```
+--------------+--------------+--------------+--------------+--------------+--------------+
|       coeff_1|       coeff_2|       coeff_3|         res_1|         res_2|         res_3|
+--------------+--------------+--------------+--------------+--------------+--------------+
|5102.441575...|-4033.76484...|-4123.28976...|0.592758773...|0.337794087...|0.063099711...|
|6874.374644...|1917.890071...|-2896.74815...|0.270694911...|0.214027348...|0.081676464...|
|8371.379532...|2863.455759...|-1692.09903...|0.156907317...|0.058389727...|0.023210721...|
|2181.687038...|1690.824113...|-1612.91537...|0.572272402...|0.315385046...|0.078044014...|
+--------------+--------------+--------------+--------------+--------------+--------------+
only showing top 4 rows
```

In [28]:
```
plot_recon_grid(coeff3_rows[:12],Mean,EigVec,header='c3=%3.2f,r3=%3.2f',params=('coeff_3','res_3'))
```



In [29]:
```
plot_recon_grid(coeff3_rows[-12:],Mean,EigVec)
```

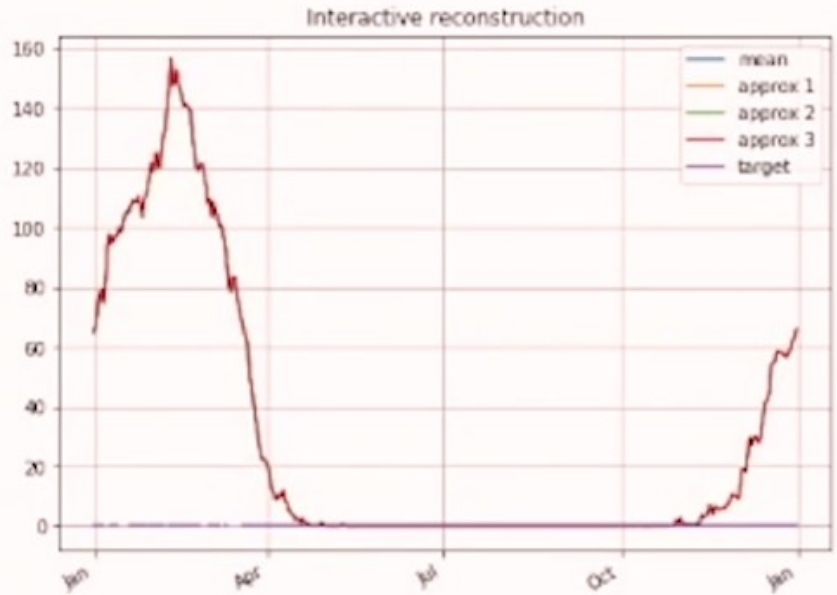


Interactive reconstruction

Following is an interactive widget which lets you change the coefficients of the eigenvectors to see the effect on the approximation. The initial state of the sliders (in the middle) corresponds to the optimal setting. You can zero a positive coefficient by moving the slider all the way down, zero a negative coefficient by moving it all the way up.

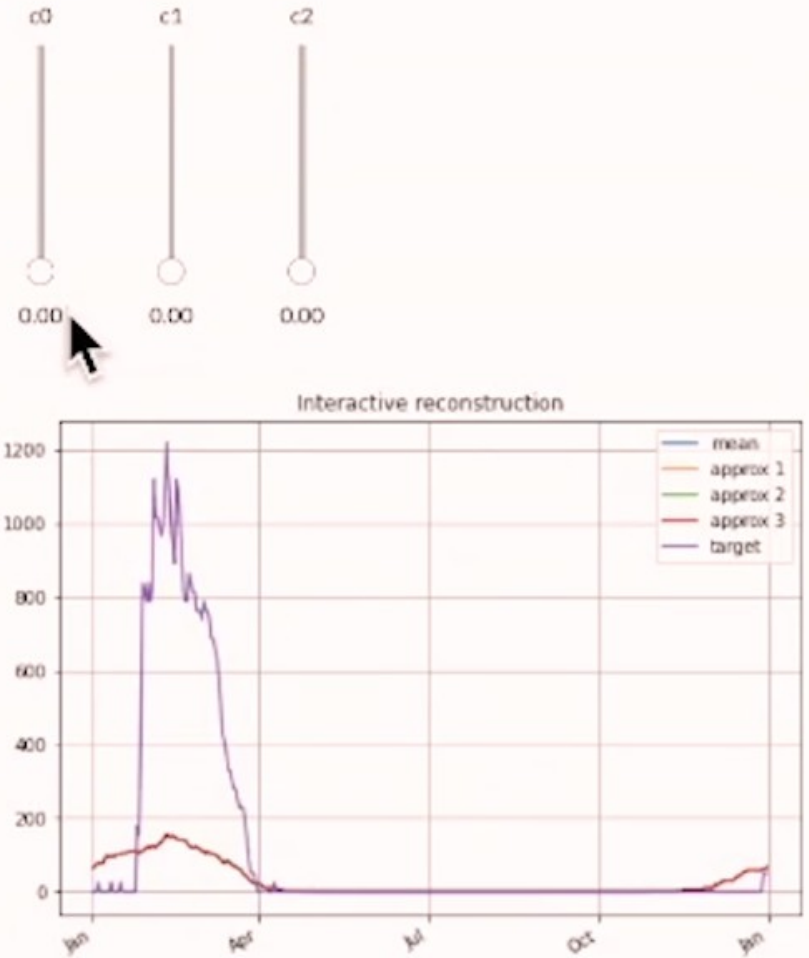
In [67]:
```
create_interactive(coeff3_rows[100])
```

```
residual normalized norm  after mean: inf
residual normalized norm  after mean + top eigs: [0.09999366 0.08752098 0.09270723]
```



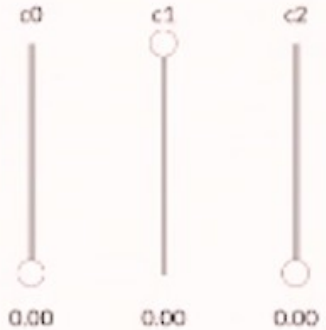
In [68]:
```
create_interactive(coeff3_rows[-1])
```

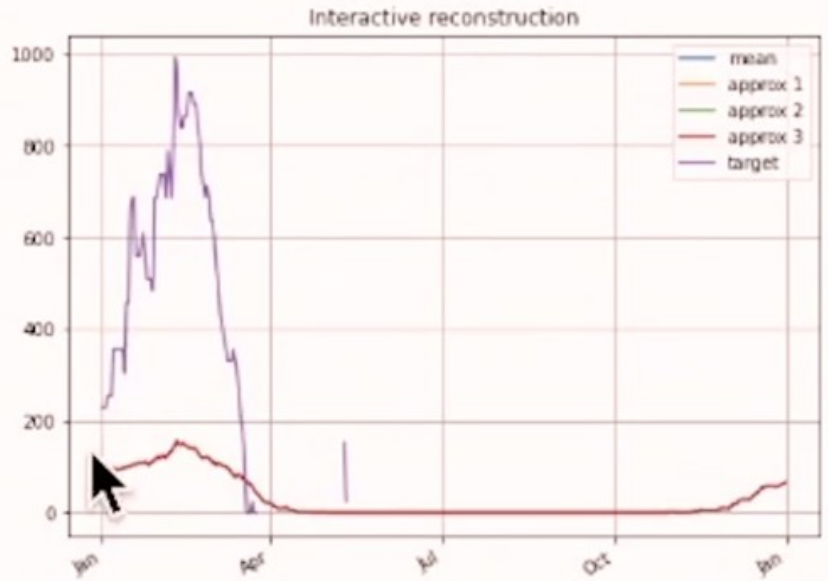
```
residual normalized norm  after mean: 0.7360576917711928
residual normalized norm  after mean + top eigs: [0.30434628 0.24886444 0.06216504]
```



In [69]:
```
create_interactive(coeff3_rows[-50])
```

```
residual normalized norm  after mean: 0.6735740247128583
residual normalized norm  after mean + top eigs: [0.12146204 0.10126608 0.05158754]
```





Studying the distribution of the coefficients

Coeff_1-3 capture most of the variance in the snow-depth distribution.

We can now look at how these coefficients vary from year to year.