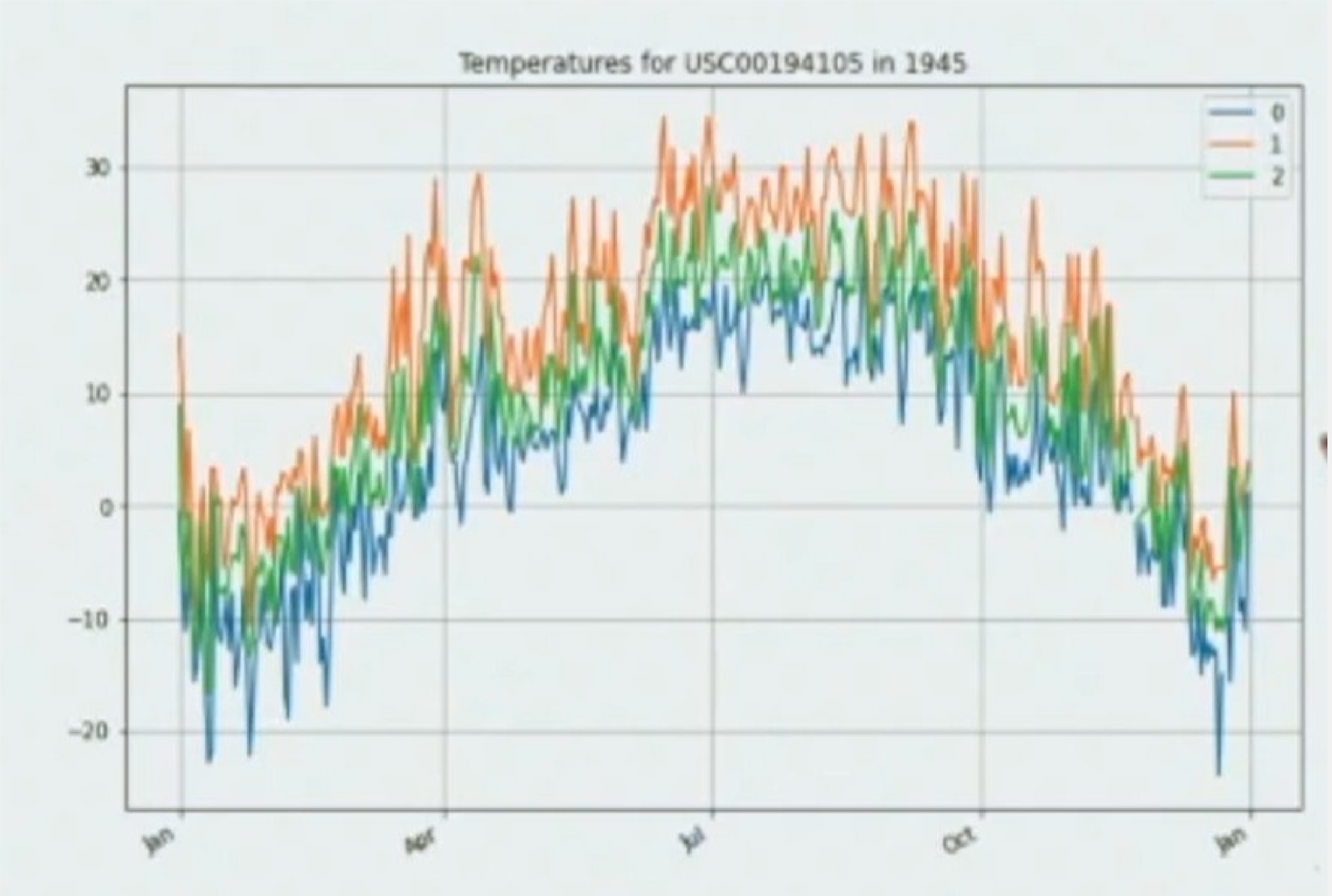
9.1 Functions As Vectors

DSC 232R, Class 9: PCA for Weather Data

Temperature Per Day as a Function

The records of temperatures for each day can be represented as a function from 1, 2, 3, … , 365 to the temperature on that day (0=Tmin, 1=TMAX, 2=TOBS)

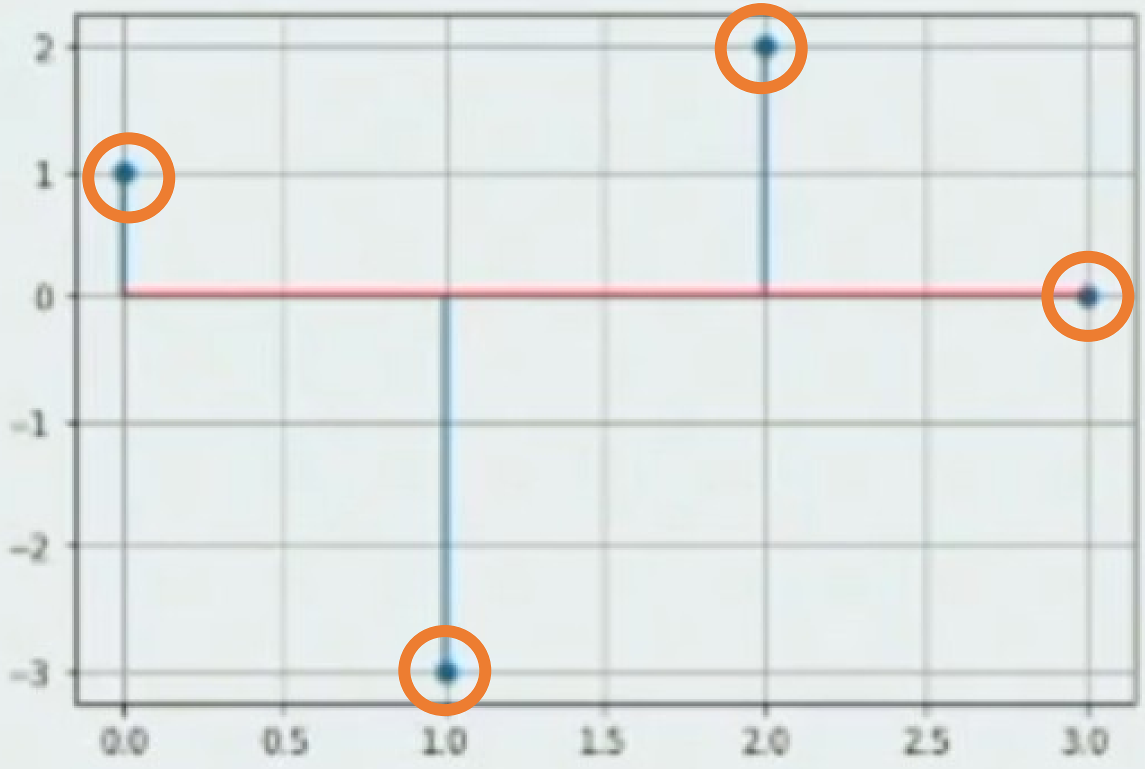
We want to find a way to approximate these functions using a set of basis function



How can We Visualize Vectors that are in Dimension Higher than 3?

- One good way to visualize a d-dimensional vector is to draw it as a function from 1, 2, ..., d to the reals

In [14]:
```
d=4
plt.stem([1,-3,2,0])
grid()
```



All of the Vector Operations are well defined, including approximating a function using an othernormal set of functions

Function Approximation

For simplicity, consider the vectors that are defined by sinusoidal functions.

Define an Orthonormal Set

The dimension of the space is 365 (arbitrary choice: the number of days in a year).

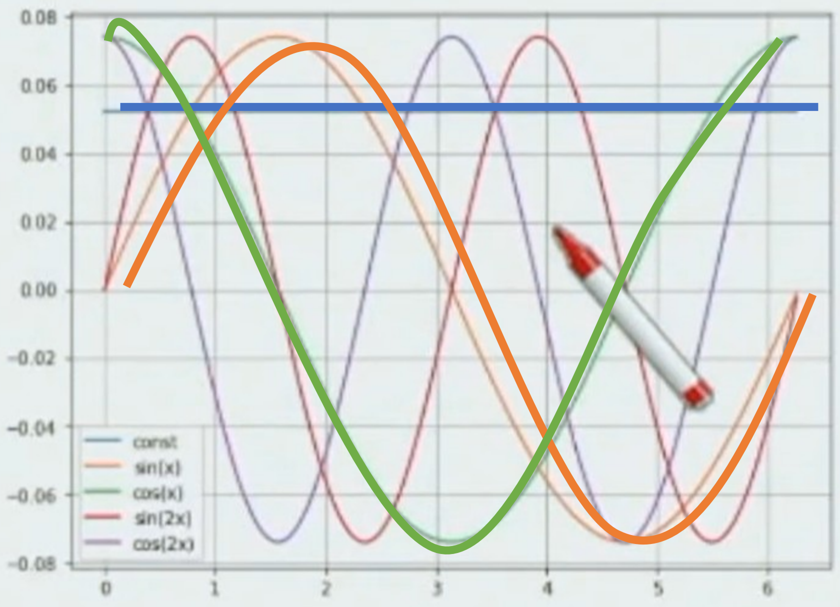
We define some functions based on sin() and cos()

```
In [4]: c=sqrt(step/(pi))
        v=[]
        v.append(np.array(cos(0*x))*c/sqrt(2))
        v.append(np.array(sin(x))*c)
        v.append(np.array(cos(x))*c)
        v.append(np.array(sin(2*x))*c)
        v.append(np.array(cos(2*x))*c)
        v.append(np.array(sin(3*x))*c)
        v.append(np.array(cos(3*x))*c)
        v.append(np.array(sin(4*x))*c)
        v.append(np.array(cos(4*x))*c)

        print("v contains %d vectors"%(len(v)))

        v contains 9 vectors
```

Const

Sin

In [5]:
```
# plot some of the functions (plotting all of them results in a figure that is hard to read.
figure(figsize=_figsize)
for i in range(5):
    plot(x,v[i])
grid()
legend(['const','sin(x)','cos(x)','sin(2x)','cos(2x)']);
```



Check that it is an Orthonormal Basis

- This basis is not **complete** it does not span the space of all functions. It spans a 9 dimensional sub-space

- We will now check that this is an **orthonormal** basis. In other words, the length of each vector is 1 and every pair of vectors are orthogonal.

```
In [6]: for i in range(len(v)):
            print()
            for j in range(len(v)):
                a=dot(v[i],v[j])
                a=round(1000*a+0.1)/1000
                print('%1.0f'%a, end=' ')
```

```
1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1
```

Rewriting the Set of Vectors as a Matrix

- Combining the vectors as rows in a matrix allows us use very succinct (and very fast) matrix multiplications instead of for loops with vector products

```
In [7]: U=vstack(v).transpose()
        shape(U)

Out[7]: (365, 9)
```

Approximating an Arbitrary Function

We now take an unrelated function $f = |x - 4|$ and see how we can use the basis matrix U to approximate it

Approximations of Increasing Accuracy

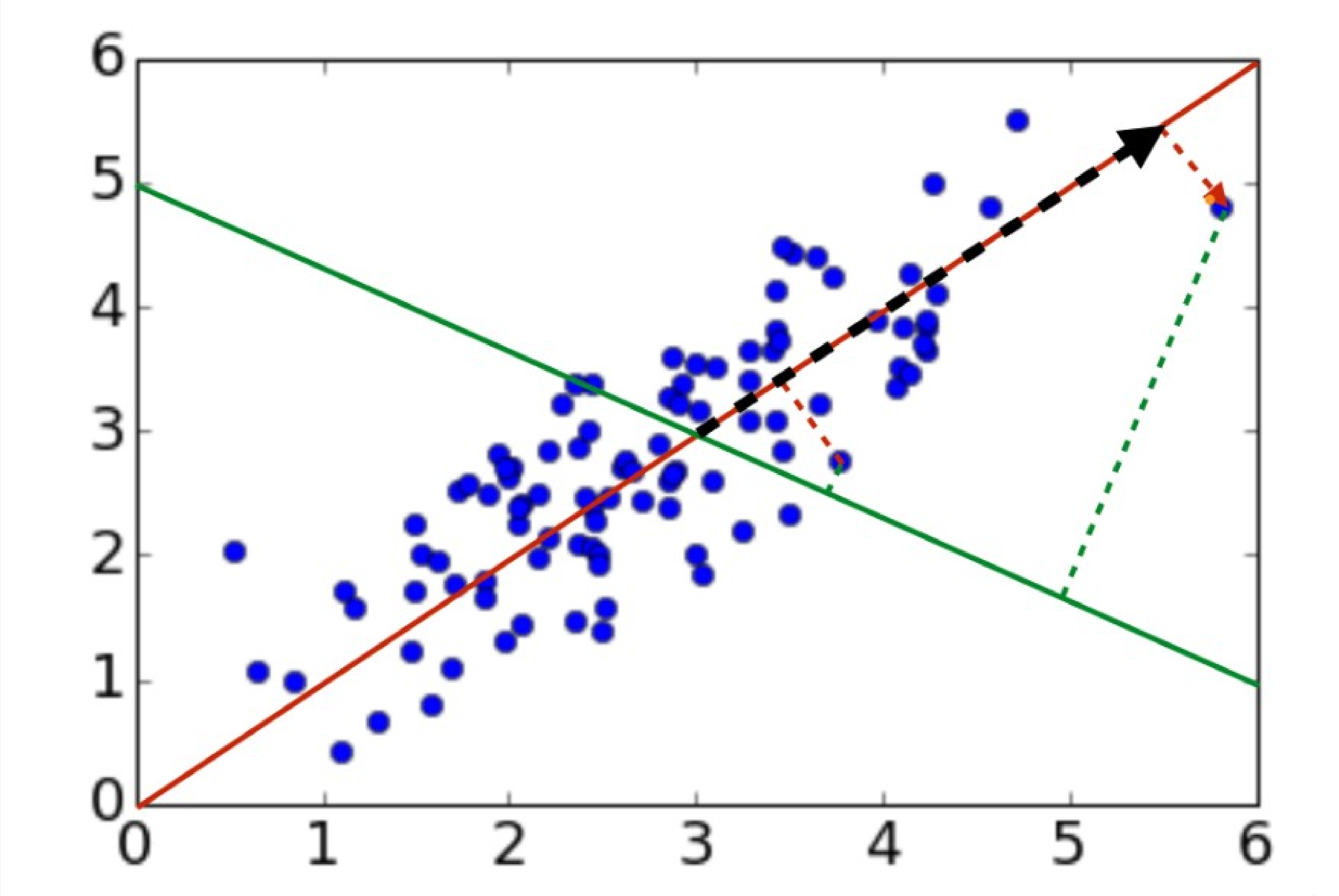
- To understand how we can use a basis to approximate functions, we create a sequence of approximations $g(i)$ such that $g(i)$ is an approximation that used the first i vectors in the basis

$$g(i) = \sum_{j=0}^{i} (f \cdot u_i) u_i$$

The larger is i, the closer $g(i)$ is to f. Where the distance between f and $g(i)$ is defined by the eculidean norm:

$$\|g(i) - f\|_2$$

Reconstruction from 1D Projection



Approximations of Increasing Accuracy

- We are given a function $f(x) = |x - 4|$, we create a sequence of approximations $g_i(x)$ such that g_i is an approximation that used the first i vectors in the basis

$$g_i = \sum_{j=0}^{i} (f \cdot u_i) u_i$$

The larger is i, the closer $g_i(x)$is to $f(\mathrm{x})$. Where the distance between f and g_i is defined by the eculidean norm:

$$\|g_i - f\|_2$$

Reconstruction in 2D using u_1

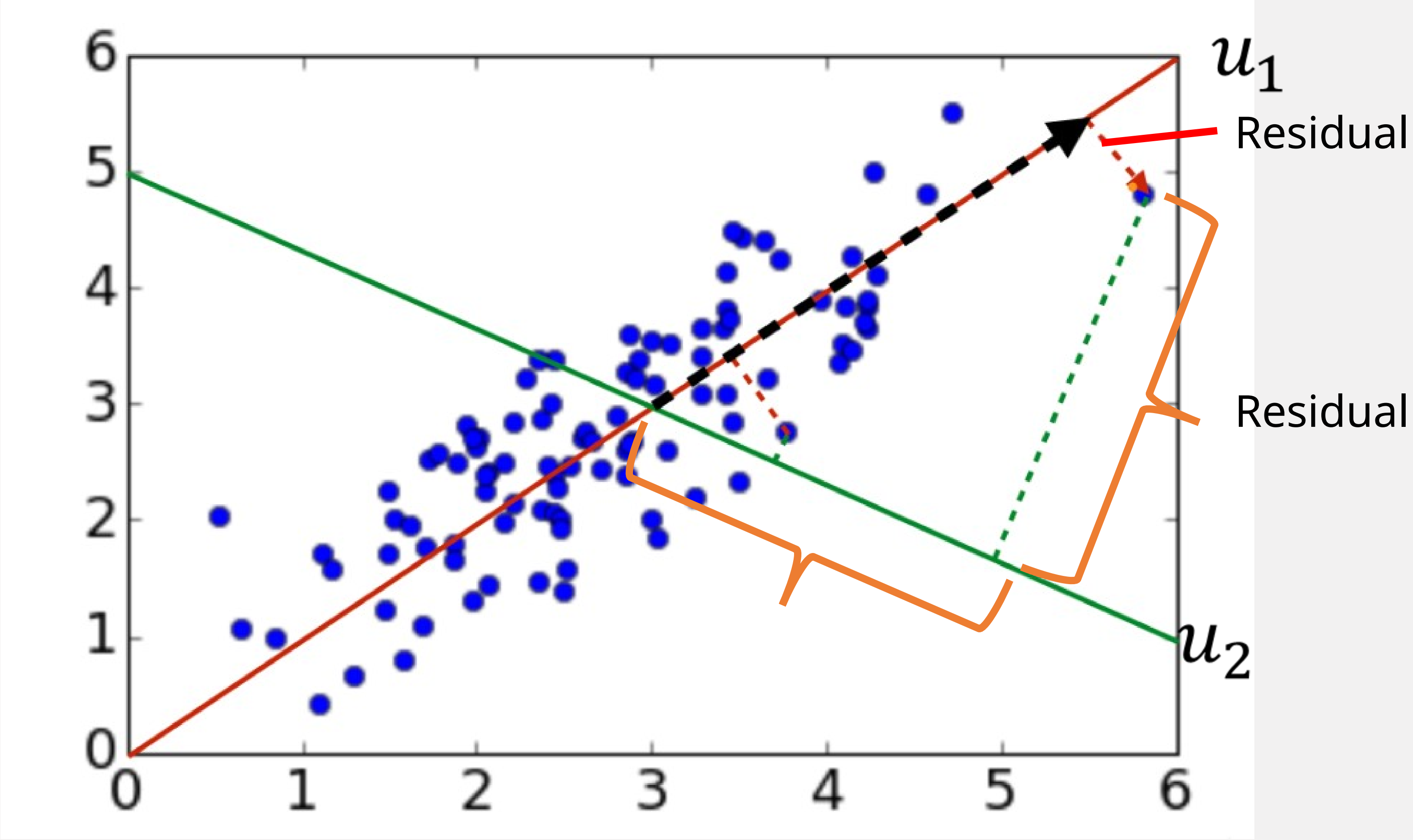
Reconstruction

$$g(1) = (p \cdot u_1)u_1$$

Residual

$$r(1) = p - g(1)$$

Reconstruction from 1D Projection



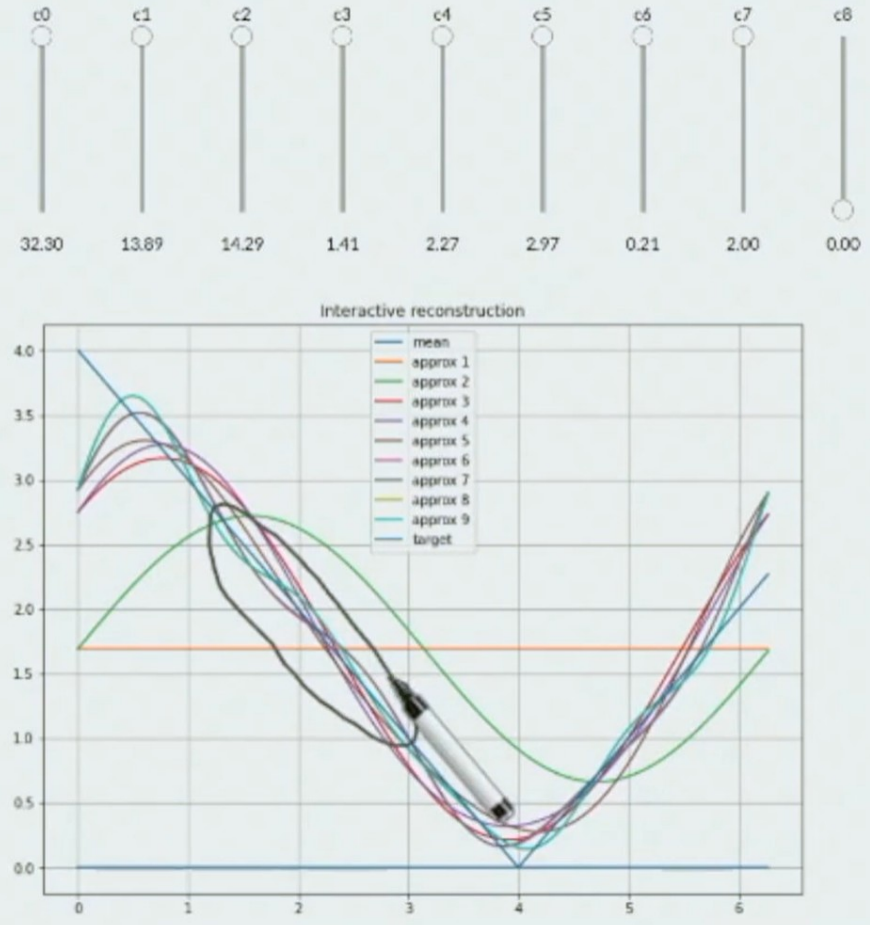
Plotting the Approximations

Below we show how increasing the number of vectors in the basis improves the approximation of f

In [10]:
```
eigen_decomp=Eigen_decomp(x,f,np.zeros(len(x)),U)
plotter=recon_plot(eigen_decomp,year_axis=False,interactive=True,figsize=_figsize);
display(plotter.get_Interactive())
```

Cont.

In [10]:
```
eigen_decomp=Eigen_decomp(x,f,np.zeros(len(x)),U)
plotter=recon_plot(eigen_decomp,year_axis=False,interactive=True,figsize=_figsize);
display(plotter.get_Interactive())
```



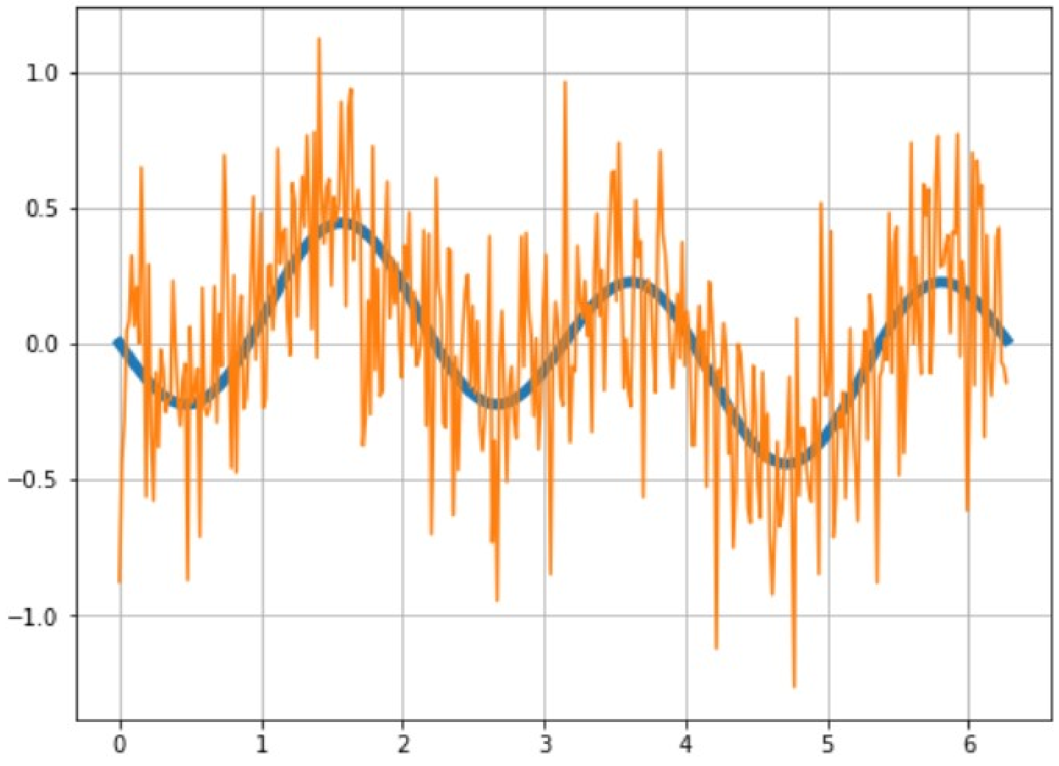
Recovering from Noise

In [11]:

```
noise=np.random.normal(size=x.shape)
f1=2*v[1]-4*v[5]
f2=f1+0.3*noise
```

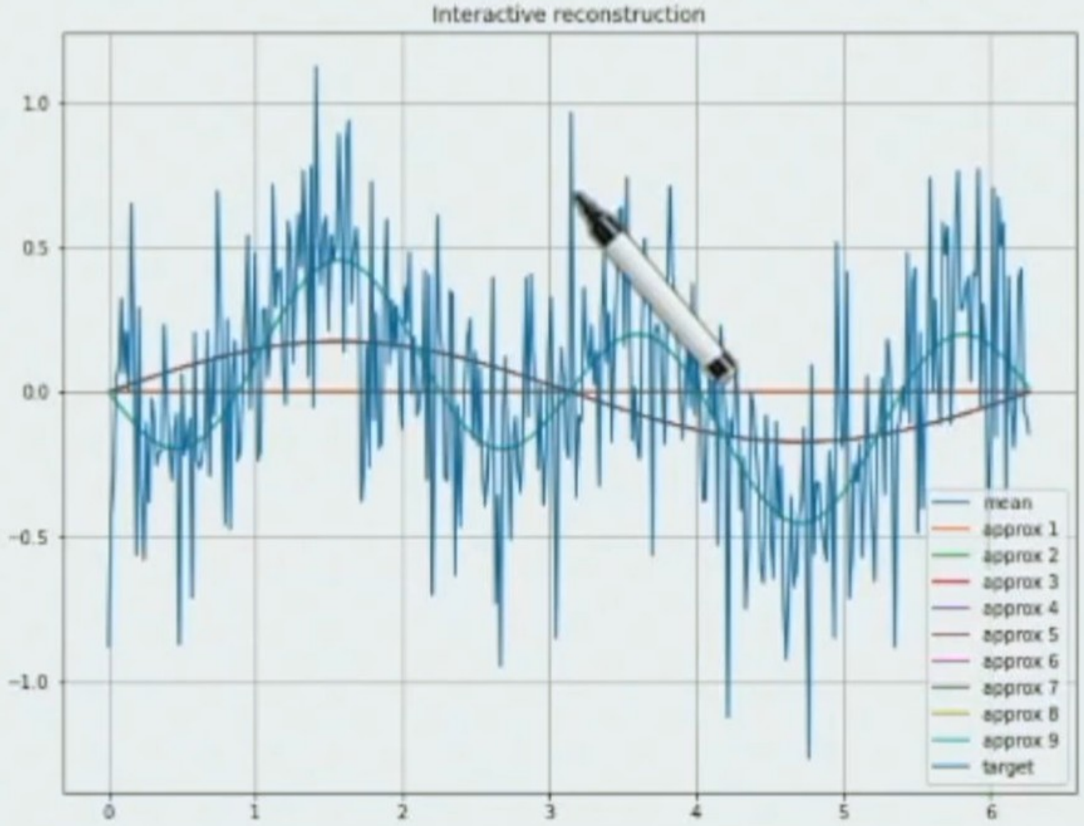
In [18]:

```
figure(figsize=_figsize)
plot(x,f1,linewidth=5);
plot(x,f2);
grid();
```



```
In [13]: eigen_decomp=Eigen_decomp(x,f2,np.zeros(len(x)),U)
         plotter=recon_plot(eigen_decomp,year_axis=False,interactive=True,figsize=_figsize);
         display(plotter.get_Interactive())
```

c0	c1	c2	c3	c4	c5	c6	c7	c8
0.00	2.35	0.00	0.00	0.00	-3.79	0.00	0.00	0.00



Summary

- Functions can be thought of as vectors and vice versa
- The **fourier** basis is a set of orthonormal functions made of $\sin s$ and $\text{cosine}\, s$
- Orthnormal functions can be used to remove the noise added to an underlying distribution