Q3)

Apply a GLM model (with considering the HRF function) to test the effects of eyes open, eyes close, and (eyes open - eyes close) on the same voxel signal in part (b). Interpret and discuss your results.

In [ ]:
```
# Import the requred Packages
import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import nibabel as nib

fmri_file = '../../datasets/fMRI/HW4/sub-001_ses-001_task-eoec_bold.nii.gz' # Get the nifti file
img = nib.load(fmri_file) # Load in the nifti file
```

The Predictors

The numbers used in the predictiors came from the content in the tsv file.

In [2]:
```
# TSV File content

# onset duration        trial_type
# 0     20      EC
# 20    20      EO
# 40    20      EC
# 60    20      EO
# 80    20      EC
# 100   20      EO
# 120   20      EC
# 140   20      EO
# 160   20      EC
# 180   20      EO
# 200   20      EC
# 220   20      EO

# Timepoints (assuming 1-second TR, adjust as necessary)
n_timepoints = 240  # Total timepoints
time = np.arange(n_timepoints)

# Create binary regressors
design_EC = np.zeros(n_timepoints)
design_EO = np.zeros(n_timepoints)

# Populate ranges based on TSV file
design_EC[0:20] = -1 # Set the value of first occurance of Eye Closed
design_EC[40:60] = -1 # Set the value of second occurance of Eye Closed
design_EC[80:100] = -1 # Set the value of third occurance of Eye Closed
design_EC[120:140] = -10 # Set the value of fourth occurance of Eye Closed
design_EC[160:180] = -15 # Set the value of fifth occurance of Eye Closed
design_EC[200:220] = -10 # Set the value of sixth occurance of Eye Closed

design_EO[20:40] = 20 # Set the value of first occurance of Eye Open
design_EO[60:80] = 15 # Set the value of second occurance of Eye Open
design_EO[100:120] = 20 # Set the value of third occurance of Eye Open
design_EO[140:160] = 10 # Set the value of fourth occurance of Eye Open
design_EO[180:200] = 10 # Set the value of fifth occurance of Eye Open
design_EO[220:240] = 10 # Set the value of sixth occurance of Eye Open
```

HRF GLM Details

All Previous information from "GLM (no HRF) Detail" from Q2 applies here.

However the clear thing to note here is the inclusion of the HRF (hemodynamic response function) which as in the name tries to mimic hemodynamic responses from the brain. This application of this function to the model could lead to a higher R^2 value which is a more accurate representation of the data and lead to a better fit of brain waves vs random noise. However this is not always the case due to the general differences between the HRF a normal regressor in some cases.

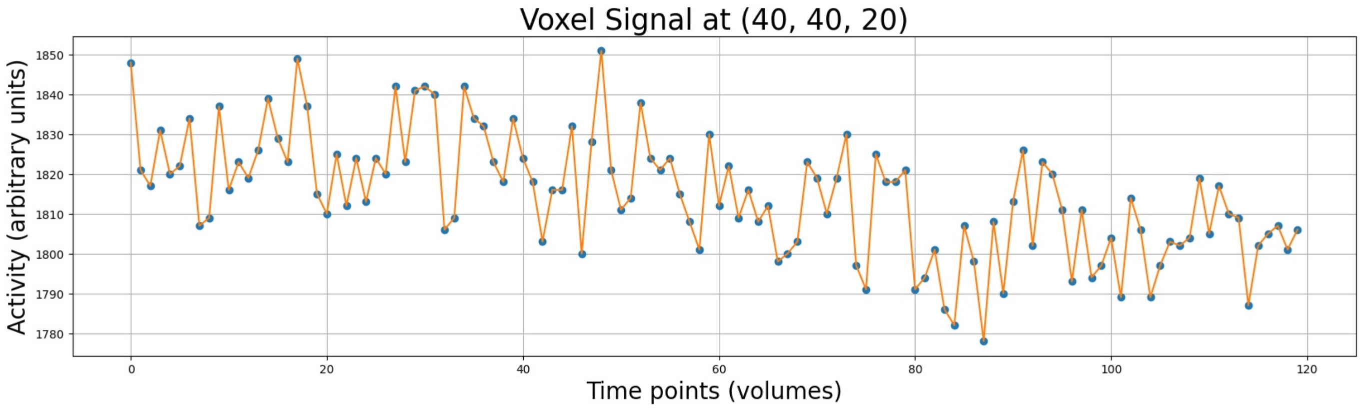
Get Voxel Signal used in Q2

In [3]:
```
data = img.get_fdata() # Get the 4 dimentional data from the fMRI
voxel_signal = data[40, 40, 20, :] # Look at a random voxel's time signal
print(voxel_signal) # Look at the random voxel's time signal
```

```
[1848. 1821. 1817. 1831. 1820. 1822. 1834. 1807. 1809. 1837. 1816. 1823.
 1819. 1826. 1839. 1829. 1823. 1849. 1837. 1815. 1810. 1825. 1812. 1824.
 1813. 1824. 1820. 1842. 1823. 1841. 1842. 1840. 1806. 1809. 1842. 1834.
 1832. 1823. 1818. 1834. 1824. 1818. 1803. 1816. 1816. 1832. 1800. 1828.
 1851. 1821. 1811. 1814. 1838. 1824. 1821. 1824. 1815. 1808. 1801. 1830.
 1812. 1822. 1809. 1816. 1808. 1812. 1798. 1800. 1803. 1823. 1819. 1810.
 1819. 1830. 1797. 1791. 1825. 1818. 1818. 1821. 1791. 1794. 1801. 1786.
 1782. 1807. 1798. 1778. 1808. 1790. 1813. 1826. 1802. 1823. 1820. 1811.
 1793. 1811. 1794. 1797. 1804. 1789. 1814. 1806. 1789. 1797. 1803. 1802.
 1804. 1819. 1805. 1817. 1810. 1809. 1787. 1802. 1805. 1807. 1801. 1806.]
```

See Content of Voxel Signal at (40, 40, 20)

In [4]:
```python
# Generates a plot of a voxel's signal.

plt.figure(figsize=(20, 5)) # Make the figure size look presentable
plt.plot(voxel_signal, 'o') # Plot the voxel signal's numerical values with an 'o'
plt.plot(voxel_signal) # Plot the whole voxel signal
plt.xlabel('Time points (volumes)', fontsize=20) # Provide an understandable x label
plt.ylabel('Activity (arbitrary units)', fontsize=20) # Provide an understandable y label
plt.title('Voxel Signal at (40, 40, 20)', fontsize=25) # Provide an understandable title
plt.grid() # Display grid lines
plt.show() # Show the plot in the output cell
```



Generate HRF Function

In [5]:
```python
# Create a canonical HRF function
from nilearn.glm.first_level.hemodynamic_models import glover_hrf # Import glover_hrf function
TR = 2 # Repetition Time
osf = 2 # Oversampling Factor
length_hrf = 32 # Length of HRF in seconds
canonical_hrf = glover_hrf(tr=TR, oversampling=osf, time_length=length_hrf,
onset=0) # Assign the resulting glover_hrf with the specified input above to a variable
canonical_hrf /= canonical_hrf.max() # Normalize HRF to have a maximum of one.
print("Size of canonical hrf variable: %i" % canonical_hrf.size) # Print hrf size
```

```
Size of canonical hrf variable: 32
```

Fitting the HRF Predictor Model (eyes open)

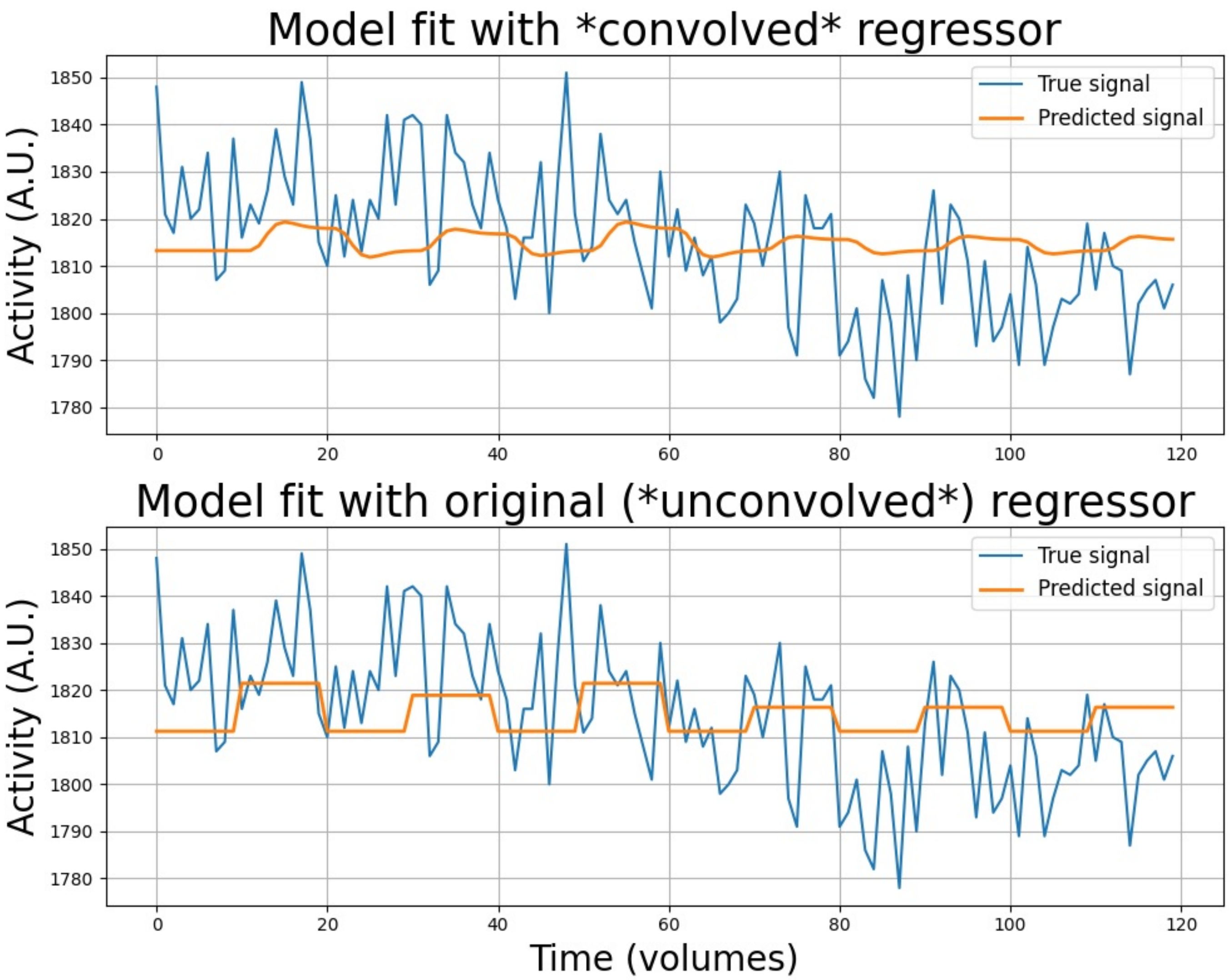
In [6]:
```python
predictor_all = design_EO # Assign the eyes open vector to predictor_all
predictor_conv = np.convolve(predictor_all.squeeze(), canonical_hrf) # Convolve the predictor_all vector with th
# After convolution, we also neem to "trim" off some excess values from the convolved signal
predictor_conv = predictor_conv[:predictor_all.size]
# And we have to add a new axis again to go from shape (N,) to (N, 1), which is important for stacking the inter
predictor_conv = predictor_conv[:, np.newaxis]

# Import needed funcitons from libraries
from scipy.interpolate import interp1d
from numpy.linalg import inv


# NO HRF (Q2) MODEL GENERATION
original_scale = np.arange(0, 240, 1) # Sample the original time values (seconds)
resampler = interp1d(original_scale, predictor_all) # Resample the time values with the regular predictor_all va
desired_scale = np.arange(0, 240, 2) # Half the original time to get time volumes
predictor_all_ds = resampler(desired_scale) # Apply the resample to get desired scale

predictor_all_ds = predictor_all_ds[:, np.newaxis] # Copy predictor and add an extra axis to it
icept = np.ones((predictor_all_ds.size, 1)) # Generate template for intercepts
X_simple = np.hstack((icept, predictor_all_ds)) # Generate Simple Model with hstack b_0 and b_1
betas_simple = np.linalg.inv(X_simple.T @ X_simple) @ X_simple.T @ voxel_signal # Generate Optimal beta values
```

```python
# HRF MODEL GENERATION
original_scale = np.arange(0, 240) # Sample the original time values (seconds)
resampler = interp1d(original_scale, np.squeeze(predictor_conv)) # Resample the time values with the convolved
desired_scale = np.arange(0, 240, 2) # Half the original time to get time volumes
predictor_conv_ds = resampler(desired_scale) # Apply the resample to get desired scale

predictor_conv_ds = predictor_conv_ds[:, np.newaxis] # Copy predictor and add an extra axis to it
intercept = np.ones((predictor_conv_ds.size, 1)) # Generate template for intercepts
X_conv = np.hstack((intercept, predictor_conv_ds)) # Generate HRF Convolutional Model with hstack b_0 and b_1
betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal # Generate Optimal beta values for b_0
```

See HRF Predictor Model (eyes open) with Voxel Signal & compare it to the Q2 Version with no HRF

In [7]:

```python
plt.figure(figsize=(10, 8)) # Make the figure size look presentable
plt.subplot(2, 1, 1) # Create the first subplot
plt.plot(voxel_signal) # Plot the whole voxel signal
plt.plot(X_conv @ betas_conv, lw=2) # Plot the HRF Predicted Model (eyes open)
plt.ylabel("Activity (A.U.)", fontsize=20) # Provide an understandable y label
plt.title("Model fit with *convolved* regressor", fontsize=25) # Provide an understandable title
plt.legend(['True signal', 'Predicted signal'], fontsize=12, loc='upper right') # Display legend to discern pre
plt.grid() # Display grid lines
plt.subplot(2, 1, 2) # Create the second subplot
plt.plot(voxel_signal) # Plot the whole voxel signal
# betas_simple = np.array([voxel_signal.mean(), 1.02307437])
plt.plot(X_simple @ betas_simple, lw=2) # Plot the no HRF Predicted Model (eyes open) from Q2
plt.ylabel("Activity (A.U.)", fontsize=20) # Provide an understandable y label
plt.title("Model fit with original (*unconvolved*) regressor", fontsize=25) # Provide an understandable title
plt.legend(['True signal', 'Predicted signal'], fontsize=12, loc='upper right') # Display legend to discern pre
plt.xlabel("Time (volumes)", fontsize=20) # Provide an understandable x label
plt.grid() # Display grid lines
plt.tight_layout() # Tighten up the layout for display
plt.show() # Show the plot in the output cell
```



From plot above we can see that the HRF has been taken into affect and looks like it maps out the general direction more accurately. However area above and below the curve seem to be smaller as well.

Test the HRF Predictor Model (eyes open) with R^2

In [8]:
```
# Fit a regresison model to the predictor and target signal.

y_hat_conv = X_conv[:, 0] * betas_conv[0] + X_conv[:, 1] * betas_conv[1] # Generate the estimated y values for 
print(betas_conv) # Print the Optimal HRF beta values
numerator = np.sum((voxel_signal - y_hat_conv) ** 2) # Get the MSE of y_hat
denominator = np.sum((voxel_signal - np.mean(voxel_signal)) ** 2) # Get the MSE of the voxel_signal mean
r_squared = 1 - numerator / denominator # Get R^2 value from y_hat
print('The R² value is: %.3f' % r_squared) # Print R^2 value
```

```
[1.81323465e+03 6.74257043e-02]
The R² value is: 0.019
```

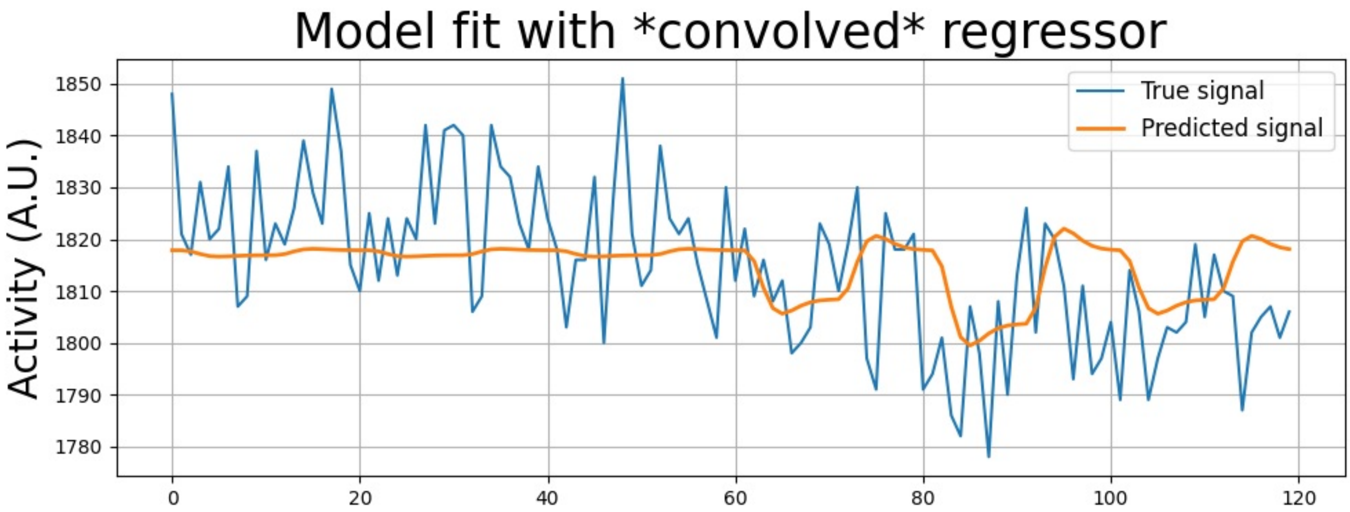
Our R^2 value is now 0.019 which explains 1.9% of the variance and to be honest sucks vs the 0.070 we had before. Further analysis will be conducted after seeing all results.

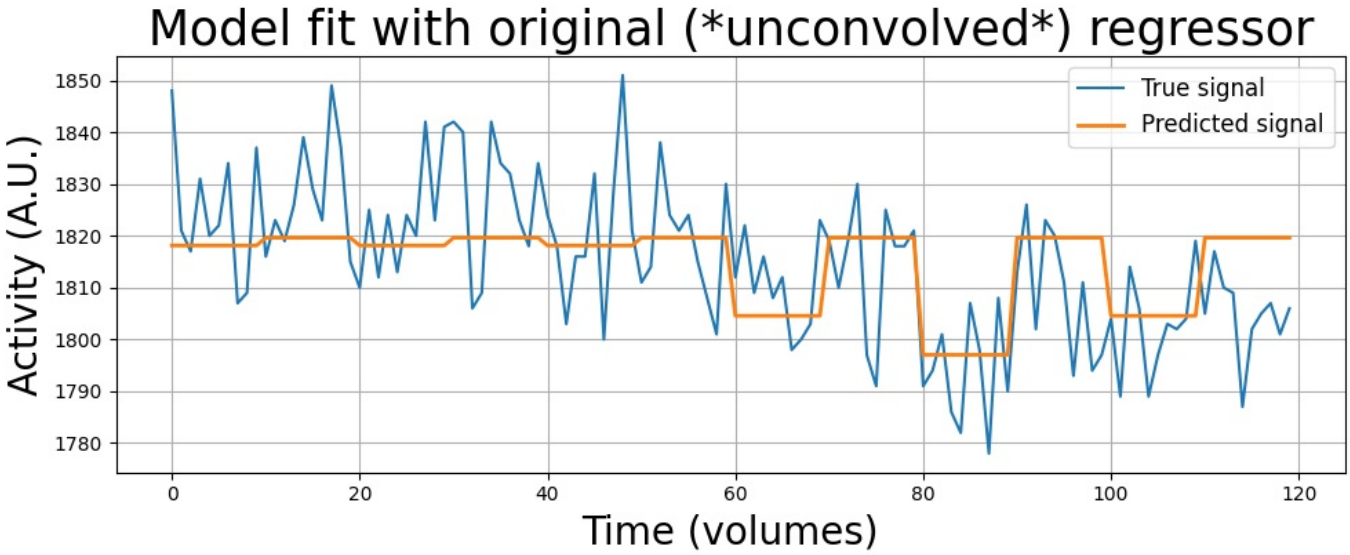
Fitting the HRF Predictor Model (eyes closed)

In [9]:
```
predictor_all = design_EC # Assign the eyes closed vector to predictor_all
predictor_conv = np.convolve(predictor_all.squeeze(), canonical_hrf) # Convolve the predictor_all vector with t
# After convolution, we also neem to "trim" off some excess values from the convolved signal
predictor_conv = predictor_conv[:predictor_all.size]
# And we have to add a new axis again to go from shape (N,) to (N, 1), which is important for stacking the inte
predictor_conv = predictor_conv[:, np.newaxis]

# Import needed funcitons from libraries
from scipy.interpolate import interp1d
from numpy.linalg import inv


# NO HRF (Q2) MODEL GENERATION
original_scale = np.arange(0, 240, 1) # Sample the original time values (seconds)
resampler = interp1d(original_scale, predictor_all) # Resample the time values with the regular predictor_all v
desired_scale = np.arange(0, 240, 2) # Half the original time to get time volumes
predictor_all_ds = resampler(desired_scale) # Apply the resample to get desired scale

predictor_all_ds = predictor_all_ds[:, np.newaxis] # Copy predictor and add an extra axis to it
icept = np.ones((predictor_all_ds.size, 1)) # Generate template for intercepts
X_simple = np.hstack((icept, predictor_all_ds)) # Generate Simple Model with hstack b_0 and b_1
betas_simple = np.linalg.inv(X_simple.T @ X_simple) @ X_simple.T @ voxel_signal # Generate Optimal beta values 


# HRF MODEL GENERATION
original_scale = np.arange(0, 240) # Sample the original time values (seconds)
resampler = interp1d(original_scale, np.squeeze(predictor_conv)) # Resample the time values with the convolved 
desired_scale = np.arange(0, 240, 2) # Half the original time to get time volumes
predictor_conv_ds = resampler(desired_scale) # Apply the resample to get desired scale

predictor_conv_ds = predictor_conv_ds[:, np.newaxis] # Copy predictor and add an extra axis to it
intercept = np.ones((predictor_conv_ds.size, 1)) # Generate template for intercepts
X_conv = np.hstack((intercept, predictor_conv_ds)) # Generate HRF Convolutional Model with hstack b_0 and b_1
betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal # Generate Optimal beta values for b_0
```

See HRF Predictor Model (eyes closed) with Voxel Signal & compare it to the Q2 Version with no HRF

In [10]:
```
plt.figure(figsize=(10, 8)) # Make the figure size look presentable
plt.subplot(2, 1, 1) # Create the first subplot
plt.plot(voxel_signal) # Plot the whole voxel signal
plt.plot(X_conv @ betas_conv, lw=2) # Plot the HRF Predicted Model (eyes closed)
plt.ylabel("Activity (A.U.)", fontsize=20) # Provide an understandable y label
plt.title("Model fit with *convolved* regressor", fontsize=25) # Provide an understandable title
plt.legend(['True signal', 'Predicted signal'], fontsize=12, loc='upper right') # Display legend to discern pre
plt.grid() # Display grid lines
plt.subplot(2, 1, 2) # Create the second subplot
plt.plot(voxel_signal) # Plot the whole voxel signal
# betas_simple = np.array([voxel_signal.mean(), 1.02307437])
plt.plot(X_simple @ betas_simple, lw=2) # Plot the no HRF Predicted Model (eyes closed) from Q2
plt.ylabel("Activity (A.U.)", fontsize=20) # Provide an understandable y label
plt.title("Model fit with original (*unconvolved*) regressor", fontsize=25) # Provide an understandable title
plt.legend(['True signal', 'Predicted signal'], fontsize=12, loc='upper right') # Display legend to discern pre
plt.xlabel("Time (volumes)", fontsize=20) # Provide an understandable x label
plt.grid() # Display grid lines
plt.tight_layout() # Tighten up the layout for display
plt.show() # Show the plot in the output cell
```





From plot above we can see that the HRF has been taken into affect and looks like it maps out the general direction more accurately. However area above and below the curve seem to be smaller as well.

Test the HRF Predictor Model (eyes closed) with R^2

In [11]:
```
# Fit a regresison model to the predictor and target signal.

y_hat_conv = X_conv[:, 0] * betas_conv[0] + X_conv[:, 1] * betas_conv[1] # Generate the estimated y values for 
print(betas_conv) # Print the Optimal HRF beta values
numerator = np.sum((voxel_signal - y_hat_conv) ** 2) # Get the MSE of y_hat
denominator = np.sum((voxel_signal - np.mean(voxel_signal)) ** 2) # Get the MSE of the voxel_signal mean
r_squared = 1 - numerator / denominator # Get R^2 value from y_hat
print('The R² value is: %.3f' % r_squared) # Print R^2 value
```

```
[1.81785363e+03 2.71311752e-01]
The R² value is: 0.124
```

Our R^2 value is now 0.124 which explains 12.4% of the variance and to be honest is worse vs the 0.259 that we had before, but still not completely bad. Further analysis will be conducted after seeing all results.

Fitting the HRF Predictor Model (eyes open - eyes closed)

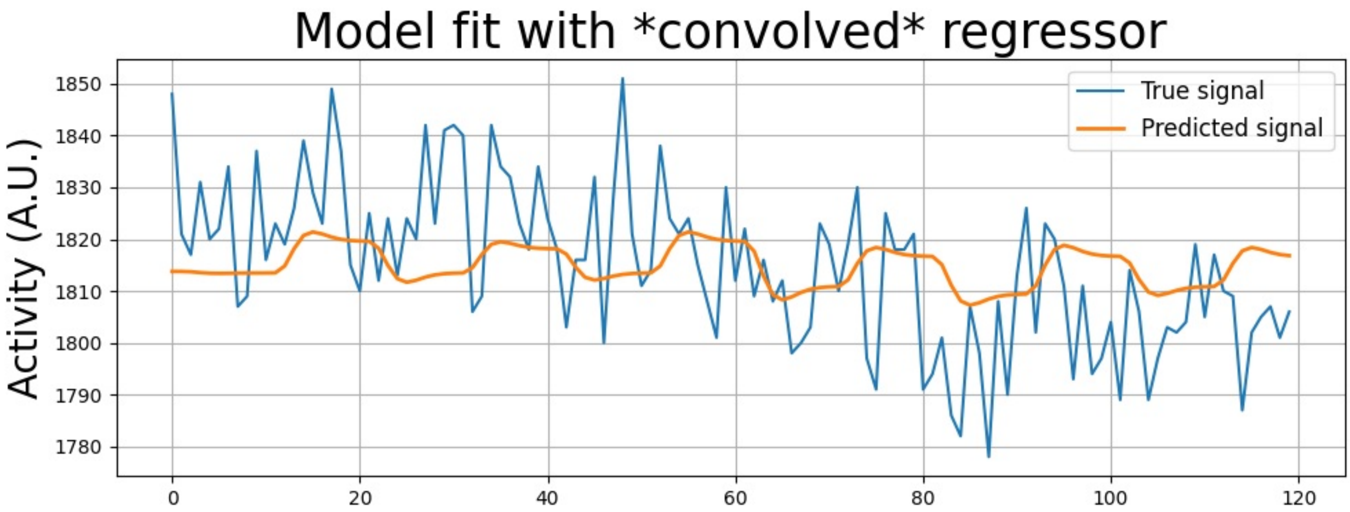
In [ ]:
```
predictor_all = (design_EO + design_EC) # Assign the (eyes open - eyes closed) vector to predictor_all
predictor_conv = np.convolve(predictor_all.squeeze(), canonical_hrf) # Convolve the predictor_all vector with th
# After convolution, we also neem to "trim" off some excess values from the convolved signal
predictor_conv = predictor_conv[:predictor_all.size]
# And we have to add a new axis again to go from shape (N,) to (N, 1), which is important for stacking the inte
predictor_conv = predictor_conv[:, np.newaxis]

# Import needed funcitons from libraries
from scipy.interpolate import interp1d
from numpy.linalg import inv


# NO HRF (Q2) MODEL GENERATION
```

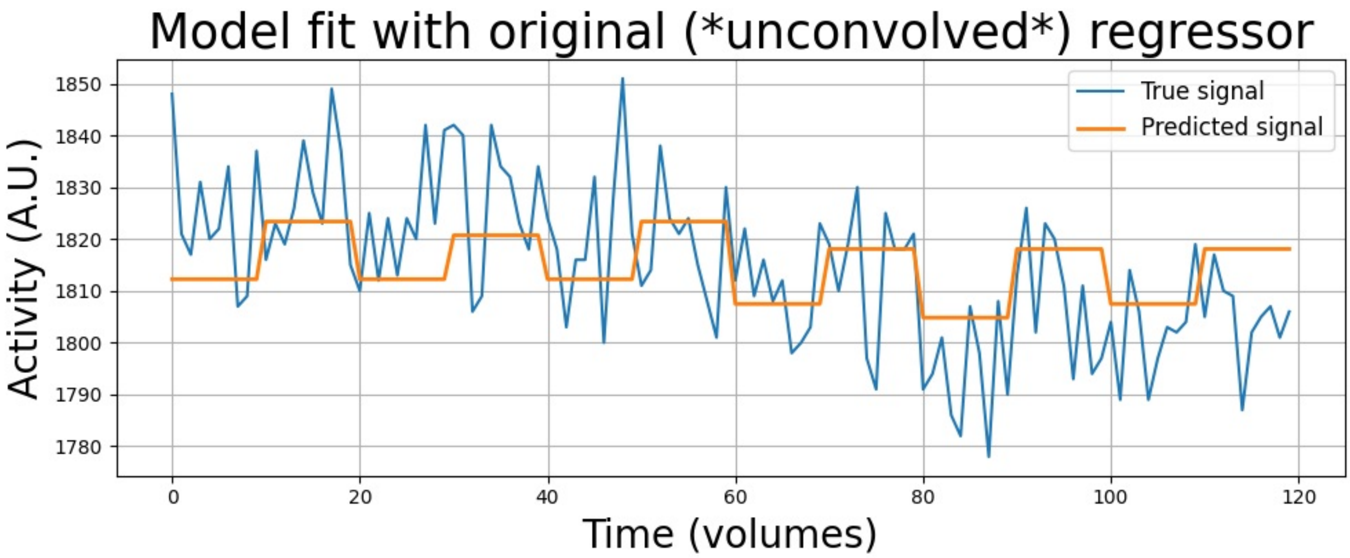
```
original_scale = np.arange(0, 240, 1) # Sample the original time values (seconds)
resampler = interp1d(original_scale, predictor_all) # Resample the time values with the regular predictor_all va
desired_scale = np.arange(0, 240, 2) # Half the original time to get time volumes
predictor_all_ds = resampler(desired_scale) # Apply the resample to get desired scale

predictor_all_ds = predictor_all_ds[:, np.newaxis] # Copy predictor and add an extra axis to it
icept = np.ones((predictor_all_ds.size, 1)) # Generate template for intercepts
X_simple = np.hstack((icept, predictor_all_ds)) # Generate Simple Model with hstack b_0 and b_1
betas_simple = np.linalg.inv(X_simple.T @ X_simple) @ X_simple.T @ voxel_signal # Generate Optimal beta values 


# HRF MODEL GENERATION
original_scale = np.arange(0, 240) # Sample the original time values (seconds)
resampler = interp1d(original_scale, np.squeeze(predictor_conv)) # Resample the time values with the convolved 
desired_scale = np.arange(0, 240, 2) # Half the original time to get time volumes
predictor_conv_ds = resampler(desired_scale) # Apply the resample to get desired scale

predictor_conv_ds = predictor_conv_ds[:, np.newaxis] # Copy predictor and add an extra axis to it
intercept = np.ones((predictor_conv_ds.size, 1)) # Generate template for intercepts
X_conv = np.hstack((intercept, predictor_conv_ds)) # Generate HRF Convolutional Model with hstack b_0 and b_1
betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal # Generate Optimal beta values for b_0
```

See HRF Predictor Model (eyes open - eyes closed) with Voxel Signal & compare it to the Q2 Version with no HRF

In [13]:

```
plt.figure(figsize=(10, 8)) # Make the figure size look presentable
plt.subplot(2, 1, 1) # Create the first subplot
plt.plot(voxel_signal) # Plot the whole voxel signal
plt.plot(X_conv @ betas_conv, lw=2) # Plot the HRF Predicted Model (eyes closed)
plt.ylabel("Activity (A.U.)", fontsize=20) # Provide an understandable y label
plt.title("Model fit with *convolved* regressor", fontsize=25) # Provide an understandable title
plt.legend(['True signal', 'Predicted signal'], fontsize=12, loc='upper right') # Display legend to discern pre
plt.grid() # Display grid lines
plt.subplot(2, 1, 2) # Create the second subplot
plt.plot(voxel_signal) # Plot the whole voxel signal
# betas_simple = np.array([voxel_signal.mean(), 1.02307437])
plt.plot(X_simple @ betas_simple, lw=2) # Plot the no HRF Predicted Model (eyes closed) from Q2
plt.ylabel("Activity (A.U.)", fontsize=20) # Provide an understandable y label
plt.title("Model fit with original (*unconvolved*) regressor", fontsize=25) # Provide an understandable title
plt.legend(['True signal', 'Predicted signal'], fontsize=12, loc='upper right') # Display legend to discern pre
plt.xlabel("Time (volumes)", fontsize=20) # Provide an understandable x label
plt.grid() # Display grid lines
plt.tight_layout() # Tighten up the layout for display
plt.show() # Show the plot in the output cell
```





From plot above we can see that the HRF has been taken into affect and looks like it maps out the general direction more accurately. However area above and below the curve seem to be smaller as well.

Test the HRF Predictor Model (eyes open - eyes closed) with R^2

In [14]:
```
# Fit a regresison model to the predictor and target signal.

y_hat_conv = X_conv[:, 0] * betas_conv[0] + X_conv[:, 1] * betas_conv[1] # Generate the estimated y values for
print(betas_conv) # Print the Optimal HRF beta values
numerator = np.sum((voxel_signal - y_hat_conv) ** 2) # Get the MSE of y_hat
denominator = np.sum((voxel_signal - np.mean(voxel_signal)) ** 2) # Get the MSE of the voxel_signal mean
r_squared = 1 - numerator / denominator # Get R^2 value from y_hat
print('The R² value is: %.3f' % r_squared) # Print R^2 value
```

```
[1.81376976e+03 8.34730086e-02]
The R² value is: 0.062
```

Our R^2 value is now 0.062 which explains 6.2% of the variance and is bad vs the 0.164 we had before. Further analysis will be conducted after seeing all results.

Thoughts about all HRF GLM Models

Overall I think that this GLMs made here were not the best and all R^2 scores given here were worse than the ones given in the regression models in Q2. However This could just be due to the fact that the regression model was fit a little better into this voxel signal and therefore made the HRF worse in this one instance.

Compared to Q2, it's obvious that the **HRF didn't perform well** and is probably due to either the values set (not being good for HRF in this case) or other factors like the regression model potentially being overfit to this voxel signal (40,40,20) in particular.