Oletetaan, että meillä on kolmen trigonometrisen funktion 4 sin(6t), 3 sin(15t) ja 5 cos(20t) summana muodostuva signaali jota näytteistetään kymmenen hertsin taajuudella (0.1s välein). Pythonilla voidaan aika ja signaali tällöin määritellä seuraavasti:

```
t = np.arange(0, 100, 0.1)
f = 4 * np.sin(6 * t) + 3 * np.sin(15 * t) + 5 * np.cos(20 * t)
```

a: Piirrä signaalin kuvaaja, voisitko sen perusteella arvata, miten signaali on muodostettu? Muuta akseleita (zoomaa) tarvitaessa

b: Laske signaalin Fourier-muunnos FFT-algoritmilla

c: Laske signaalin tehospektri

d: Tarkastele tehospektriä, mitkä ovat dominoivat taajuudet? Miten ne vastaavat alunperin määriteltyä signaalia?

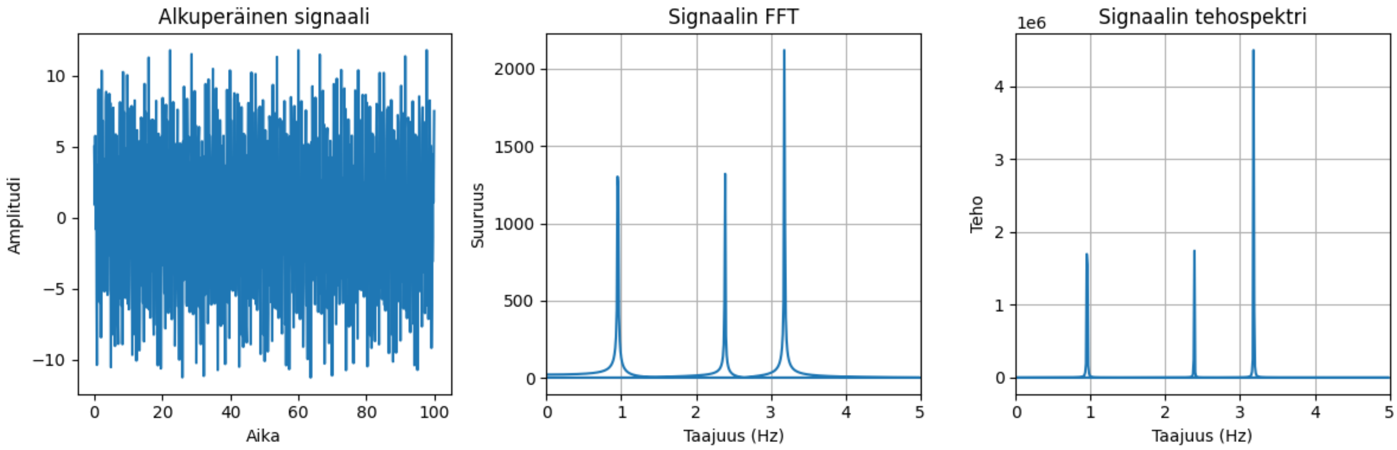
e: Laske Fourier-muunnoksen käänteismuunnos ja varmista, että saat sen tuloksena alkuperäisen signaalin.

BONUS: Pura FFT-algoritmin antama tulos sini- ja kosinimuotoisiksi signaaleiksi. Varmista, että niiden summana saat alkuperäisen signaalin.

In [44]:

```python
import matplotlib.pyplot as plt
import numpy as np

# Aikavektori ja signaali
t = np.arange(0, 100, 0.1)
f = 4 * np.sin(6 * t) + 3 * np.sin(15 * t) + 5 * np.cos(20 * t)

# Piirretään kuvaaja alkuperäisestä signaalista
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.plot(t, f)
plt.title('Alkuperäinen signaali')
plt.xlabel('Aika')
plt.ylabel('Amplitudi')

# Lasketaan signaalin FFT
fft_values = np.fft.fft(f)
n = len(f)  # Datapisteiden lukumäärä signaalissa
fs = 1 / (t[1] - t[0])  # Näytteenottoväli, datapisteiden ajallinen välimatka
fft_freq = np.fft.fftfreq(n, d=1/fs)

# Lasketaan tehospektri
psd = fft_values*np.conj(fft_values)

# Piirretään kuvaaja FFT suuruudesta
plt.subplot(1, 3, 2)
plt.plot(fft_freq, np.abs(fft_values))
plt.title('Signaalin FFT')
plt.xlabel('Taajuus (Hz)')
plt.ylabel('Suuruus')
plt.grid(True)
plt.xlim(0, fs/2)  # Rajataan X-akseli positiivisiin taajuuksiin

# Piirretään tehospektri
plt.subplot(1, 3, 3)
plt.plot(fft_freq, psd)
plt.title('Signaalin tehospektri')
plt.xlabel('Taajuus (Hz)')
plt.ylabel('Teho')
plt.grid(True)
plt.xlim(0, fs/2)  # Rajataan X-akseli positiivisiin taajuuksiin

plt.tight_layout()
plt.show()
```

```
c:\Users\henri\AppData\Local\Programs\Python\Python312\Lib\site-packages\matplotlib\cbook.py:1762: ComplexWarning: Casting complex values to real discards t
he imaginary part
  return math.isfinite(val)
c:\Users\henri\AppData\Local\Programs\Python\Python312\Lib\site-packages\matplotlib\cbook.py:1398: ComplexWarning: Casting complex values to real discards t
he imaginary part
  return np.asarray(x, float)
```

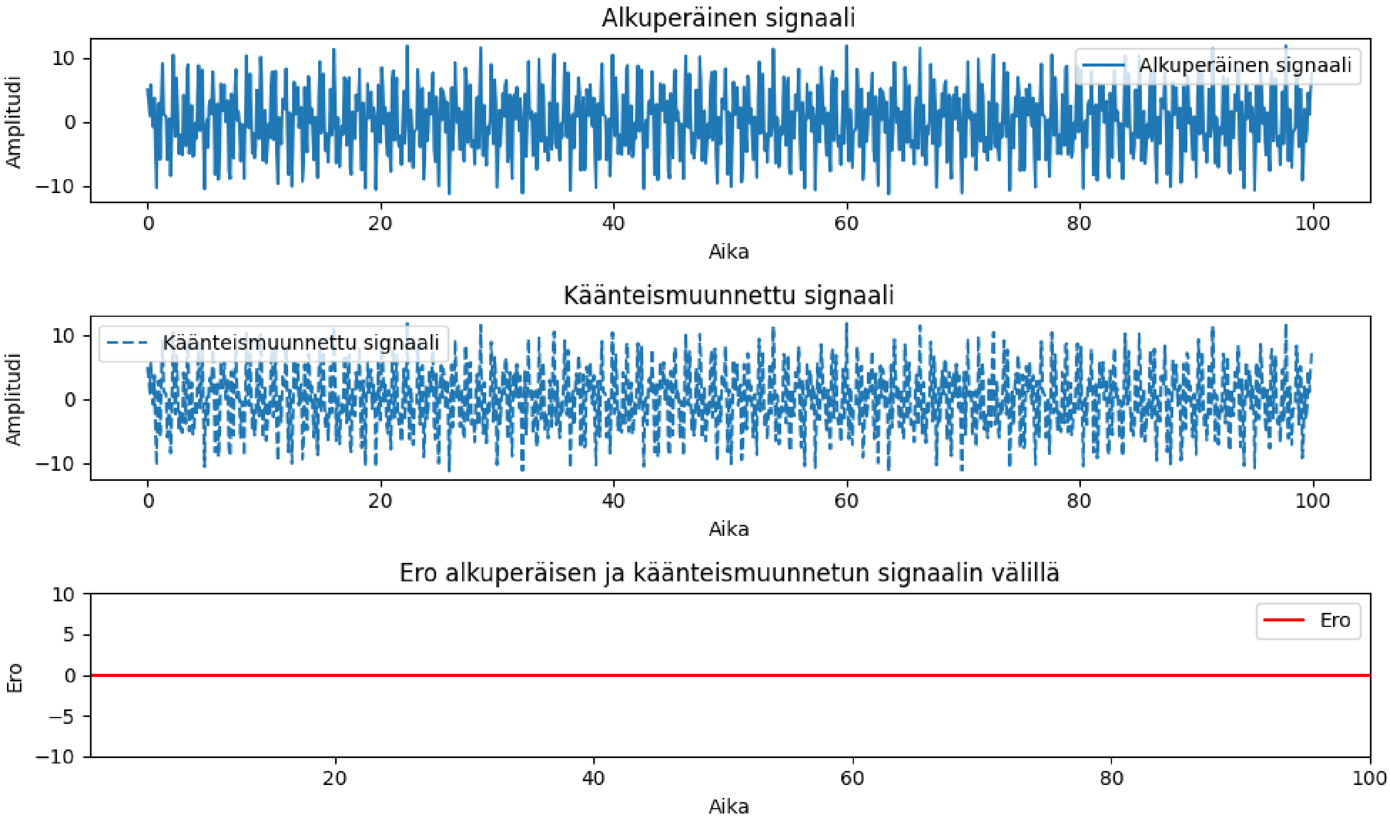


a: Näyttää siltä, että signaali on yhdistelmä kahta sinisignaalia johon on lisätty kohinaa. b: Kuvaaja yllä c: Kuvaaja yllä d: 0.9Hz, 2,3Hz, sekä 3,2Hz. Ne vastaavat alkuperäistä signaalia täydellisesti.

In [45]:

```python
import matplotlib.pyplot as plt
import numpy as np

# Aikavektori ja signaali
t = np.arange(0, 100, 0.1)
f = 4 * np.sin(6 * t) + 3 * np.sin(15 * t) + 5 * np.cos(20 * t)

# Lasketaan signaalin FFT
fft_values = np.fft.fft(f)

# Lasketaan käänteismuunnos
ifft_values = np.fft.ifft(fft_values)

# Numeerinen vertailu, rajataan desimaalit, koska muuten vertailun tulos on häviävän pieni lukema
difference = np.abs(f - ifft_values.real)
max_difference = np.max(difference)
print(f'FFT arvojen ja FFT-käänteisarvojen välinen ero on maksimissaan: {max_difference:.10f}')

# Piirretään alkuperäinen ja käänteismuunnettu signaali
plt.figure(figsize=(10, 6))

# Alkuperäinen signaali
plt.subplot(3, 1, 1)
plt.plot(t, f, label='Alkuperäinen signaali')
plt.title('Alkuperäinen signaali')
plt.xlabel('Aika')
plt.ylabel('Amplitudi')
plt.legend()

# Käänteismuunnettu signaali
plt.subplot(3, 1, 2)
plt.plot(t, ifft_values.real, label='Käänteismuunnettu signaali', linestyle='--')
plt.title('Käänteismuunnettu signaali')
plt.xlabel('Aika')
plt.ylabel('Amplitudi')
plt.legend()

# Ero signaalien välillä
plt.subplot(3, 1, 3)
plt.plot(t, difference, label='Ero', color='r')
plt.axis([1,100,-10,10])
plt.title('Ero alkuperäisen ja käänteismuunnetun signaalin välillä')
plt.xlabel('Aika')
plt.ylabel('Ero')
plt.legend()

plt.tight_layout()
plt.show()
```

FFT arvojen ja FFT-käänteisarvojen välinen ero on maksimissaan: 0.0000000000



e: Käänteis-signaali laskettuna ja tuotua kuvaajaan yllä. Signaalit näyttäisivät kuvaajia vertaamalla vastaavan toisiaan täydellisesti.

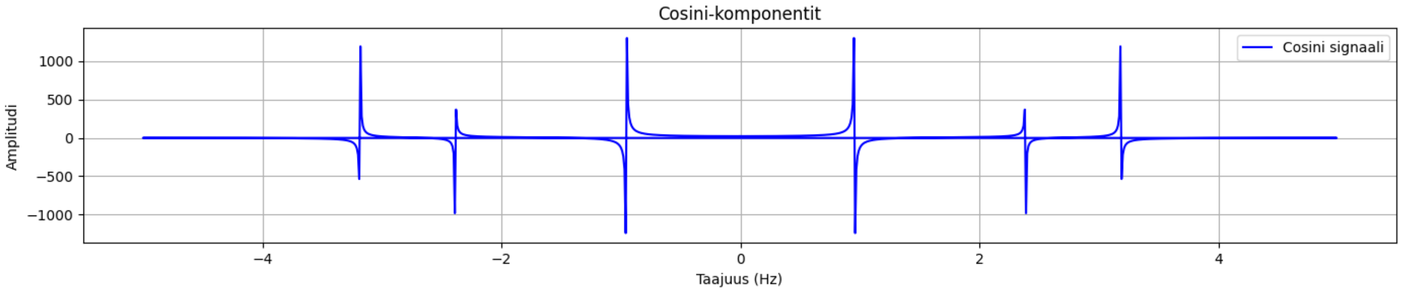
Signaaleja vertaamalla ja niistä kuvaajan piirtämällä saadaan aikaiseksi suora viiva. Mitä lähemäksi zoomataan, niin sitä enemmän eroavaisuuksia kuitenkin löytyy.

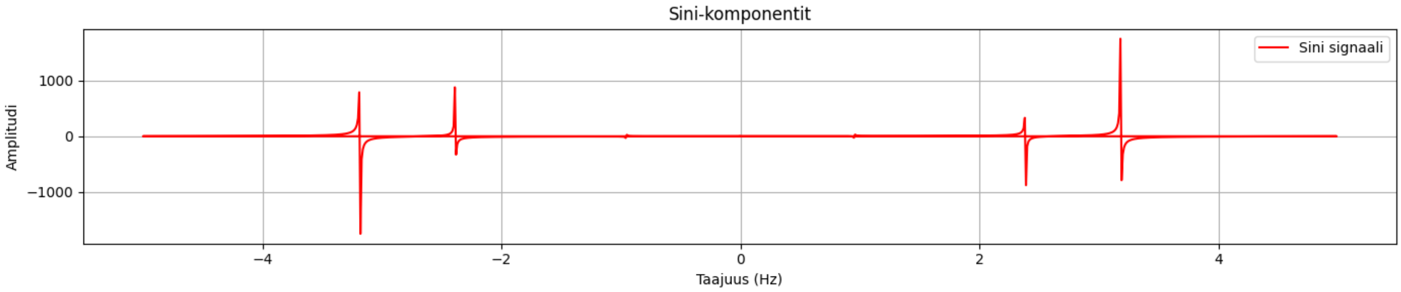
Numeerisen vertailun perusteella ero on maksimissaan 0.0000000000, eli olematon. Ilman desimaalirajausta (.10f) tulokseksi saatiin erittäin pieni luku (5.329070518200751e-15), mutta se ei ole millään tavalla merkittävä.

In [55]:

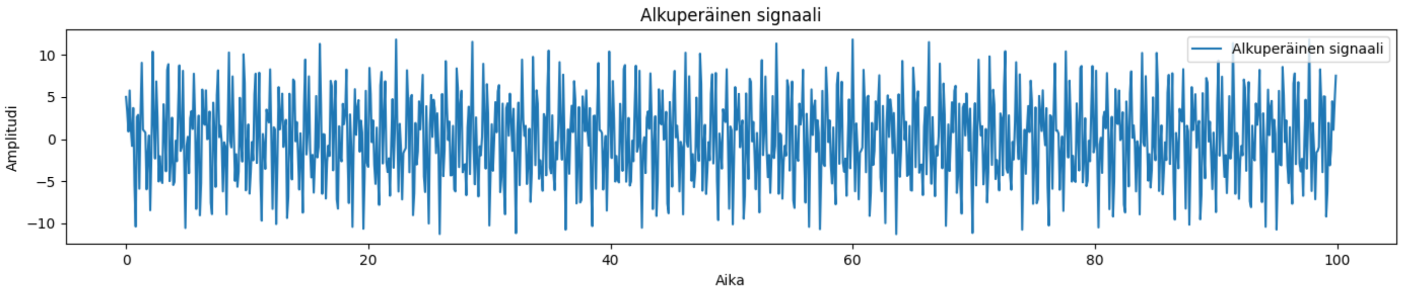
```
import numpy as np
import matplotlib.pyplot as plt

# Erota reaaliosa (cosini-komponentit) ja imaginaariosa (sini-komponentit)
cos_components = np.real(fft_values)
sin_components = np.imag(fft_values)

# Piirretään cosini- ja sinikomponentit
plt.figure(figsize=(14, 6))

# Cosini-komponentit
plt.subplot(2, 1, 1)
plt.plot(fft_freq, cos_components, label='Cosini signaali', color='b')
plt.title('Cosini-komponentit')
plt.xlabel('Taajuus (Hz)')
plt.ylabel('Amplitudi')
plt.grid(True)
plt.legend()

# Sini-komponentit
plt.subplot(2, 1, 2)
plt.plot(fft_freq, sin_components, label='Sini signaali', color='r')
plt.title('Sini-komponentit')
plt.xlabel('Taajuus (Hz)')
plt.ylabel('Amplitudi')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```

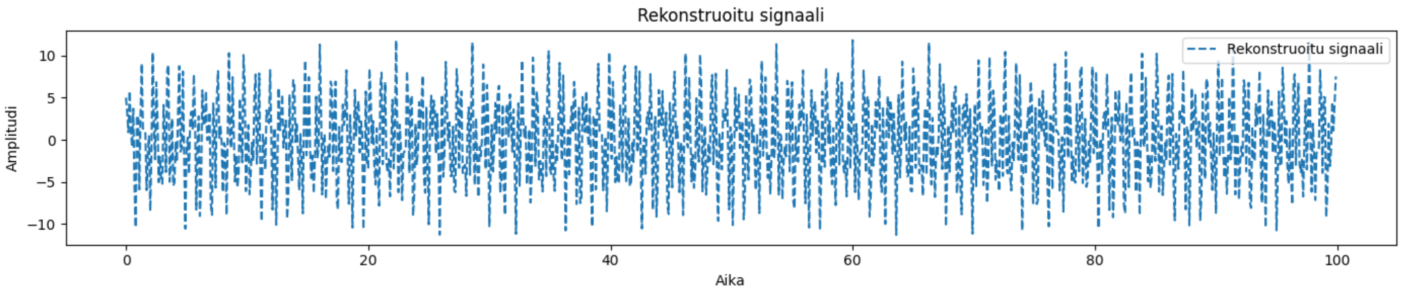
```
plt.show()

# Luo sini- ja cosini-signaalit taajuuskomponenteista
reconstructed_signal = np.zeros_like(f)
for i in range(n):
    reconstructed_signal += (cos_components[i] * np.cos(2 * np.pi * fft_freq[i] * t) -
                             sin_components[i] * np.sin(2 * np.pi * fft_freq[i] * t)) / n

# Numeerinen vertailu
difference = np.abs(f - reconstructed_signal)
max_difference = np.max(difference)
mean_difference = np.mean(difference)
print(f'Alkuperäisen ja rekonstruoidun signaalin maksimieroavaisuus: {max_difference:.10f}')
print(f'Alkuperäisen ja rekonstruoidun signaalin keskimääräinen eroavaisuus: {mean_difference:.10f}')

# Piirretään alkuperäinen ja rekonstruoitu signaali
plt.figure(figsize=(14, 6))

# Alkuperäinen signaali
plt.subplot(2, 1, 1)
plt.plot(t, f, label='Alkuperäinen signaali')
plt.title('Alkuperäinen signaali')
plt.xlabel('Aika')
plt.ylabel('Amplitudi')
plt.legend()

# Rekonstruoitu signaali
plt.subplot(2, 1, 2)
plt.plot(t, reconstructed_signal, label='Rekonstruoitu signaali', linestyle='--')
plt.title('Rekonstruoitu signaali')
plt.xlabel('Aika')
plt.ylabel('Amplitudi')
plt.legend()

plt.tight_layout()
plt.show()
```





Alkuperäisen ja rekonstruoidun signaalin maksimieroavaisuus: 0.0000000000
Alkuperäisen ja rekonstruoidun signaalin keskimääräinen eroavaisuus: 0.0000000000





f. Ylemmässä kuvaajassa näemme, että Sini ja Cosini -signaalit osuvat yksiin ja ovat itsensä kanssa hyvin symmetrisiä. Signaalien muutokset osuvat yhteen aiemmin nähdyn tehospektrin piikkien kanssa.

Sini ja cosini signaalit purkamalla ja rekonstruoimalla saadaan aikaiseksi täymälleen alkuperäistä vastaava kuvaaja. Numeerisesti varmistettuna signaalit ovat yhteneväiset.

Tulostetaan muistikirja pdf:ksi

In [70]:

```python
NB_name='FourierTransform'

!jupyter nbconvert --to html {NB_name}.ipynb

# Add custom CSS to the HTML file
html_file = f'{NB_name}.html'
with open(html_file, 'r', encoding='utf-8') as file:
    html_content = file.read()

custom_css = """
<style>
pre {
    background-color: #f5f5f5;
    border: 1px solid #ccc;
    padding: 10px;
    border-radius: 5px;
    overflow: auto;
}
code {
    background-color: #f5f5f5;
    border: 1px solid #ccc;
    padding: 2px 4px;
    border-radius: 3px;
}
</style>
"""

# Insert the custom CSS into the <head> section of the HTML file
html_content = html_content.replace('<head>', '<head>' + custom_css)

# Write the modified HTML content back to the file
with open(html_file, 'w', encoding='utf-8') as file:
    file.write(html_content)

# Convert HTML to PDF using wkhtmltopdf with --enable-local-file-access
!wkhtmltopdf --enable-local-file-access {NB_name}.html {NB_name}.pdf
```

```
[NbConvertApp] Converting notebook FourierTransform.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 4 image(s).
[NbConvertApp] Writing 1096501 bytes to FourierTransform.html
Loading pages (1/6)
[>                                                           ] 0%
[======>                                                     ] 10%
[=============================>                              ] 49%
[=============================>                              ] 49%
[=====================================================>      ] 90%
[=====================================================>      ] 90%
[============================================================] 100%
Counting pages (2/6)
[============================================================] Object 1 of 1
Resolving links (4/6)
[============================================================] Object 1 of 1
Loading headers and footers (5/6)
Printing pages (6/6)
[>                                                           ] Preparing
[=======>                                                    ] Page 1 of 8
[===============>                                            ] Page 2 of 8
[======================>                                     ] Page 3 of 8
[==============================>                             ] Page 4 of 8
[=====================================>                      ] Page 5 of 8
[=============================================>              ] Page 6 of 8
[====================================================>       ] Page 7 of 8
[============================================================] Page 8 of 8
Done
```