3.4 Spektralanalyse

Lösungen

Aufgabe 1

Das GNU Octave-Skript lädt zunächst das Package `signal`, das die Funktion **`resample`** enthält, und berechnet anschließend die Zeitreihe.

```
# Übungsblatt 3.4, Aufgabe 1: Fourier-Transformation
# Einfluss der Abtastrate, Alias-Effekt
# ------------------------------------------------------------

  set(0, "defaultaxesfontsize", 10);

  pkg load signal

  file = mfilename();

# Daten

  T = 0.001;      % Impulsdauer
  n =     5;      % Messdauer = n * T

# Zeitreihe

  fs  = 6 / T; dt1 = 1 / fs;    % Abtastrate und Zeitschritt
  t1  = 0 : dt1 : n * T;        % Zeitpunkte
  N1  = length(t1);

  x1    = zeros(1, N1);
  j     = find(t1 < T);
  x1(j) = 4 * t1(j) .* (T - t1(j)) / T^2;
```

Die Berechnung der Fourier-Transformation erfolgt mithilfe der Funktion **`fft`** für die diskrete Fourier-Transformation. Für die Werte der Fourier-Transformierten gilt

$$\hat{X}_k = \hat{X}\left(f_S \frac{k}{N_1} \right) = \Delta t_1 X_k .$$

Die Fourier-Transformierte wird zusätzlich mit der Impulsdauer **`T`** skaliert. Die Werte zu den positiven Frequenzen sind in den ersten **`N1/2`** Elementen des Felds **`X1`** gespeichert.

```
# Fourier-Transformation

  X1d = fft(x1);
  X1  = dt1 * X1d(1 : N1/2 + 1) / T;
```

```
  f1  = fs * (0 : N1/2) / N1;
```

Nun wird die Abtastrate ohne Filtern verringert. Dazu wird einfach jedes zweite Element der ursprünglichen Zeitreihe extrahiert. Anschließend wird wieder die Fourier-Transformierte berechnet und mit der Impulsdauer skaliert.

```
# Abtastrate verringern ohne Filtern

  x2  = x1(1 : 2 : N1); t2  = t1(1 : 2 : N1);
  N2  = length(x2);
  X2d = fft(x2);
  X2  = 2 * dt1 * X2d(1 : N2/2 + 1) / T;
  f2  = 0.5 * fs * (0 : N2/2) / N2;
```

Bei Verwendung der Funktion **`resample`** wird die Zeitreihe vor Verringern der Abtastrate so gefiltert, dass die Nyquist-Bedingung erfüllt ist.

```
# Abtastrate verringern mit Filtern

  x3  = resample(x1, 1, 2); N3  = length(x3);
  t3  = 2 * dt1 * (0 : N3 - 1);
  X3d = fft(x3);
  X3  = 2 * dt1 * X3d(1 : N3/2 + 1) / T;
  f3  = 0.5 * fs * (0 : N3/2) / N3;
```

Schließlich werden zwei Diagramme erzeugt. Das erste vergleicht die ungefilterte mit der gefilterten Zeitreihe, und das zweite zeigt die Fourier-Transformierten.

```
# Ausgabe

  figure(1, "position", [100, 500, 1000, 500],
            "paperposition", [0, 0, 15, 7.5]);
  plot(t1, x1, "color", "red", t3, x3, "color", "green");
  grid;
```

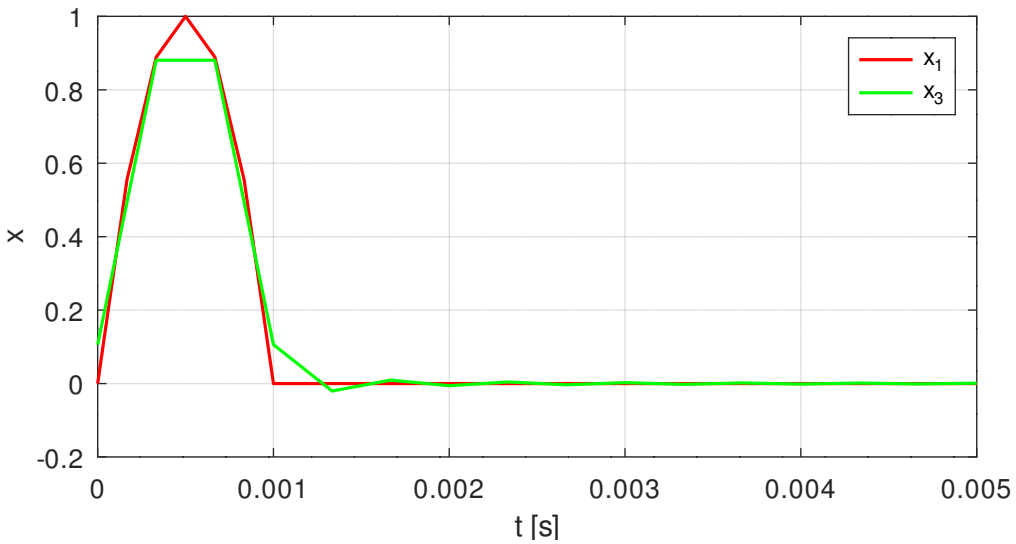

Abbildung 1.1: Zeitreihen

```
legend("x_1", "x_3");
xlabel("t [s]"); ylabel("x");
print([file, "x.svg"], "-dsvg");

figure(2, "position", [200, 400, 1000, 800],
          "paperposition", [0, 0, 15, 10]);
subplot(2, 1, 1, "position", [0.14, 0.59, 0.77, 0.34]);
   plot(f1, real(X1), "color", "red",
        f2, real(X2), "color", "blue",
        f3, real(X3), "color", "green");
   grid;
   legend("X_1", "X_2", "X_3");
   ylabel("Re(X) / T");
subplot(2, 1, 2, "position", [0.14, 0.13, 0.77, 0.33]);
   plot(f1, imag(X1), "color", "red",
        f2, imag(X2), "color", "blue",
        f3, imag(X3), "color", "green");
   grid;
   legend("X_1", "X_2", "X_3", "location", "east");
   ylabel("Im(X) / T"); xlabel("f [Hz]");
print([file, "X.svg"], "-dsvg");
```

Abbildung 1.1 zeigt die Zeitreihen und Abbildung 1.2 ihre Fourier-Transformierten. Wegen der Filterung weicht die Zeitreihe x_3 von der Zeitreihe x_1 ab. Dafür stimmen die Fourier-Transformierten der Zeitreihen x_1 und x_3 in dem gemeinsam abgedeckten Frequenzbereich überein, während die Fourier-Transformierte der ohne Filterung erzeugten Zeitreihe x_2 deutlich abweicht.

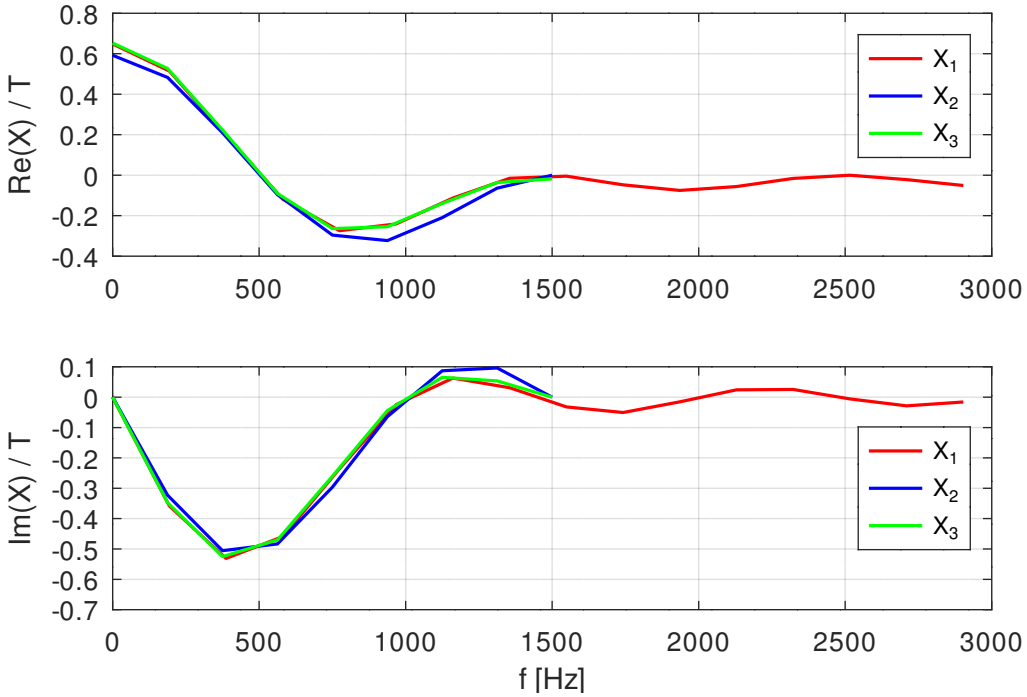

Abbildung 1.2: Fourier-Transformierte

Aufgabe 2

Zur Berechnung der Koeffizienten der Fourier-Reihe wird zunächst eine Zeitreihe erzeugt, deren Länge einem ganzzahligen Vielfachen der Periode entspricht.

```
# Übungsblatt 3.4, Aufgabe 2: Fourier-Reihe
# ------------------------------------------------------------

  set(0, "defaultaxesfontsize", 12);

  file = mfilename();

# Daten

  T = 0.01;                     % Periode
  n =    2;                     % Anzahl Perioden (frei gewählt)

# Zeitreihe

  fs = 12 / T;                  % Abtastrate
  dt = 1 / fs;                  % Zeitschritt
  N  = n * T * fs;              % Anzahl Zeitpunkte
  t  = (0 : N - 1) * dt;        % Zeitpunkte
  x  = sin(pi * t / T).^2;      % Zeitreihe
```

Die Koeffizienten der Fourier-Reihe werden mithilfe der diskreten Fourier-Transformation berechnet. Es gilt:

$$c_0 = X_0, \quad c_k = \frac{2}{N}\Re(X_k), s_k = -\frac{2}{N}\Im(X_k), \quad k = 1,\ldots,\frac{N}{2}$$

Die Koeffizienten für die positiven Frequenzen sind in den ersten **1+N/2** Elementen des Felds **C** gespeichert. Für die zugehörigen Frequenzen gilt

$$f_k = \frac{k}{nT}, \quad k = 0,\ldots,\frac{N}{2} \quad ,$$

wobei n die Anzahl der für die Berechnung gewählten Perioden ist.

```
# Fourier-Transformation

  C = 2 * fft(x) / N;  C(1) = 0.5 * C(1);
  f = (0 : N/2) / (n * T);
  c =  real(C(1 : N/2 + 1));
  s = -imag(C(1 : N/2 + 1));
```

Die Koeffizienten der Fourier-Reihe werden in Form eines Balkendiagramms ausgegeben.

```
# Ausgabe

  figure(1, "position", [100, 500, 800, 500],
            "paperposition", [0, 0, 14, 7]);
```

```
bar(f, c, "barwidth", 0.2);
xticks(0 : 100 : f(end)); yticks(-1 : 0.25 : 1);
grid;
xlabel("f [Hz]"); ylabel("c");
print([file, "a.svg"], "-dsvg");
```

Zur Probe werden die Koeffizienten der Fourier-Reihe analytisch ermittelt. Aus

$$\sin^2\left(\pi\frac{t}{T}\right)=\frac{1}{2}\left(1-\cos\left(2\pi\frac{t}{T}\right)\right)$$

folgt:

$$c_0=\frac{1}{2},\quad c_1=-\frac{1}{2},\quad f_1=\frac{1}{T}=100\,\mathrm{Hz}$$

Außerdem wird die berechnete Fourier-Reihe mit dem ursprünglichen Zeitsignal verglichen.

```
# Probe

  wt = 2 * pi * f' * t;
  y  = c * cos(wt) + s * sin(wt);

  figure(2, "position", [300, 200, 800, 500],
            "paperposition", [0, 0, 14, 6]);
  plot(t, x, "color", "red", t, y, "color", "green");
  grid;
  legend("original", "Fourier-Reihe", "location", "north");
  xlabel("t [s]"); ylabel("x");
  print([file, "b.svg"], "-dsvg");
```

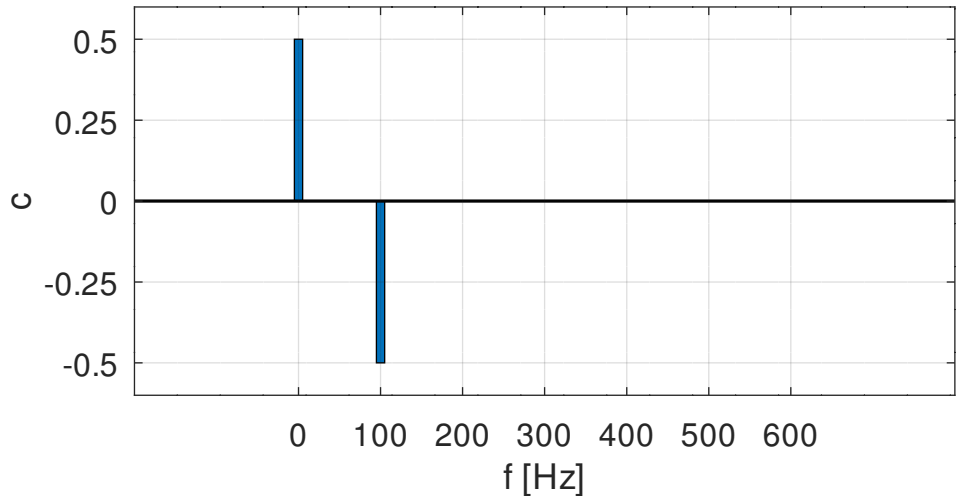

Abbildung 2.1: Fourier-Koeffizienten

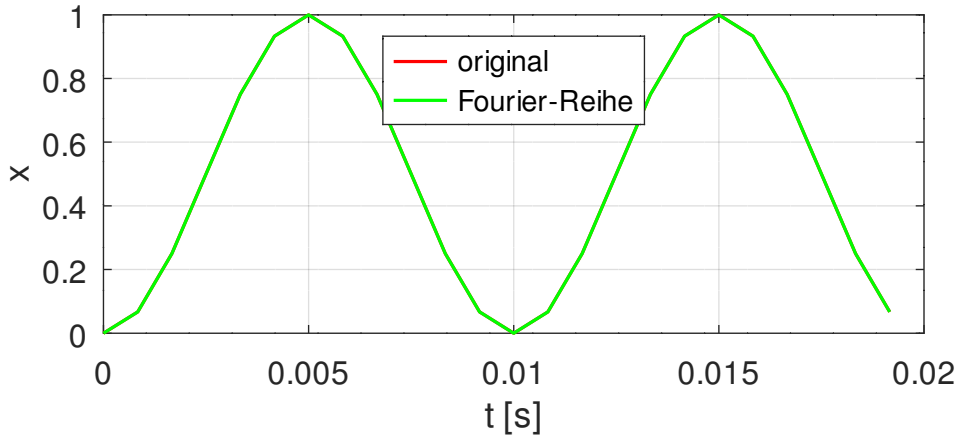

Abbildung 2.2: Vergleich der Zeitsignale

Die Ergebnisse sind in den Abbildungen 2.1 und 2.2 dargestellt. Abbildung 2.1 zeigt, dass die numerisch ermittelten Koeffizienten mit den analytisch ermittelten Werten übereinstimmen. Abbildung 2.2 zeigt, dass die Fourier-Reihe das Zeitsignal exakt wiedergibt.

Aufgabe 3

Zur Berechnung der Koeffizienten der Fourier-Reihe wird zunächst eine Zeitreihe erzeugt, deren Länge einem ganzzahligen Vielfachen der Periode entspricht.

```
# Übungsblatt 3.4, Aufgabe 3: Fourier-Reihe
# ------------------------------------------------------------

  set(0, "defaultaxesfontsize", 12);

  file = mfilename();

# Daten

  T =  0.1;       % Periode
  r = 0.75;
  n =    2;       % Anzahl Perioden

# Zeitreihe

  fs = 20 / T;                % Abtastrate
  dt = 1 / fs;                % Zeitschritt
  N  = n * T * fs;            % Anzahl Zeitpunkte
  t  = (0 : N - 1) * dt;      % Zeitpunkte

  wt = 2 * pi * t / T;
  x  = r * sin(wt);
  x  = cos(wt) ./ sqrt(1 - x.^2);
  x  = sin(wt) .* (1 + r * x);
```

Die Koeffizienten der Fourier-Reihe werden aus den Werten der diskreten Fourier-Transformation berechnet. Die verwendeten Formeln sind die gleichen wie bei Aufgabe 2.

```
# Fourier-Transformation

  C = 2 * fft(x) / N; C(1) = 0.5 * C(1);
  f = (0 : N/2) / (n * T);
  c =  real(C(1 : N/2 + 1));
  s = -imag(C(1 : N/2 + 1));
```

Die Koeffizienten werden in tabellarischer Form in einer Datei ausgegeben sowie graphisch als Balkendiagramm.

```
# Ausgabe

  fid = fopen([file, ".res"], "wt");

  fprintf(fid, "  k      f         c         s\n");
  fprintf(fid, " ---------------------------------\n");
  for k = 0 : N / (2 * n) - 1
      l = 2 * k + 1;
      fprintf(fid, "  %2d  %8.4f  %8.4f  %8.4f\n",
              k, f(l), c(l), s(l));
  end

  fclose(fid);

  figure(1, "position", [100, 500, 800, 500],
            "paperposition", [0, 0, 14, 7]);
  bar(f, s, "barwidth", 0.2);
  xticks(0 : 10 : f(end));
  grid;
  xlabel("f [Hz]"); ylabel("s");
  print([file, "a.svg"], "-dsvg");
```

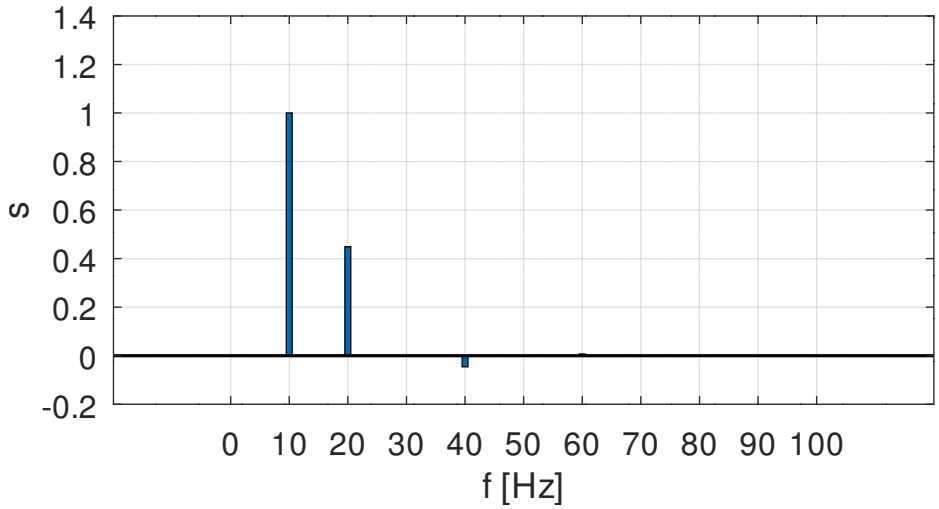

Abbildung 3.1: Fourier-Koeffizienten

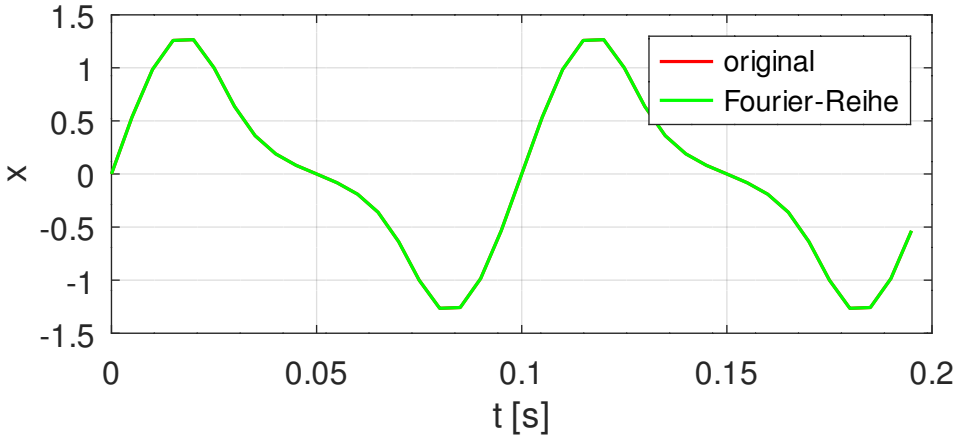

Abbildung 3.2: Vergleich der Zeitreihen

Zur Kontrolle wird die Fourier-Reihe mit dem ursprünglichen Zeitsignal verglichen.

```
# Probe

  wt = 2 * pi * f' * t;
  y  = c * cos(wt) + s * sin(wt);

  figure(2, "position", [300, 200, 800, 500],
            "paperposition", [0, 0, 14, 6]);
  plot(t, x, "color", "red", t, y, "color", "green");
  grid;
  legend("original", "Fourier-Reihe");
  xlabel("t [s]"); ylabel("x");
  print([file, "b.svg"], "-dsvg");
```

Abbildung 3.1 zeigt die Fourier-Koeffizienten und Abbildung 3.2 einen Vergleich der Zeitreihen. Die Fourier-Reihe gibt das Zeitsignal sehr gut wieder.

Die Ausgabedatei enthält die folgenden Werte:

```
  k      f          c          s
 ------------------------------
   0     0.0000    -0.0000    -0.0000
   1    10.0000    -0.0000     1.0000
   2    20.0000    -0.0000     0.4491
   3    30.0000    -0.0000     0.0000
   4    40.0000    -0.0000    -0.0455
   5    50.0000     0.0000     0.0000
   6    60.0000     0.0000     0.0069
   7    70.0000     0.0000     0.0000
   8    80.0000     0.0000    -0.0011
   9    90.0000     0.0000    -0.0000
```

Die Werte zeigen, dass die Fourier-Reihe nur Sinus-Terme enthält. Die wesentlichen Beiträge kommen von den ersten vier Reihengliedern.

Aufgabe 4

Wenn die Periode nicht bekannt ist, werden zur Bestimmung der Koeffizienten der Fourier-Reihe Fensterfunktionen benötigt. Die Fensterfunktionen sind im Package `signal` enthalten, das daher geladen werden muss. Anschließend werden die Daten eingelesen und in den Feldern **`t`** (Zeitpunkte) und **`x`** (Werte der Zeitreihe) gespeichert. Die Abtastrate wird aus dem Zeitschritt berechnet, der als Mittelwert der Zeitdifferenzen bestimmt wird.

```
# Übungsblatt 3.4, Aufgabe 4: Periodische Zeitreihe
# ------------------------------------------------------------

  set(0, "defaultaxesfontsize", 12);

  pkg load signal

  file = mfilename();

# Daten einlesen

  data = dlmread([file, ".csv"]);
  t    = data(:, 1);
  x    = data(:, 2);
  N    = length(x);
  fs   = 1 / mean(diff(t));
```

Nun werden die Koeffizienten der Fourier-Reihe und die Frequenzen berechnet. Die erste Rechnung erfolgt mit einem Rechteck-Fenster, d. h. ohne Verwendung einer Fensterfunktion. Die anderen beiden Rechnungen verwenden ein Hanning-Fenster bzw. ein Flattop-Fenster.

```
# Fourier-Transformation ohne Fenster

  C = 2 * fft(x) / N; C(1) = 0.5 * C(1);
  a =  abs(C(1 : N/2 + 1));
  f = (0 : N/2) * fs / N;

# Fourier-Transformation mit Hanning-Fenster

  wh = hanning(N);
  Ch = 2 * fft(wh .* x) / sum(wh); Ch(1) = 0.5 * Ch(1);
  ah =  abs(Ch(1 : N/2 + 1));

# Fourier-Transformation mit Flattop-Fenster

  wf = flattopwin(N);
  Cf = 2 * fft(wf .* x) / sum(wf); Cf(1) = 0.5 * Cf(1);
  af =  abs(Cf(1 : N/2 + 1));
```

Die berechneten Werte für das Hanning- und das Flattop-Fenster werden in tabellarischer Form in einer Datei ausgegeben. Die Tabelle enthält nur Amplituden, die größer als $0{,}01$ sind. Außerdem werden die Ergebnisse für alle drei

Fenster zum Vergleich graphisch ausgegeben.

```
# Ausgabe

  fid = fopen([file, ".res"], "wt");
  fprintf(fid, "     f     Hanning      Flattop\n");
  fprintf(fid, "  ------------------------------\n");

  for k = 1 : N/2 + 1
      if (max(ah(k), af(k)) > 1.e-2)
         fprintf(fid, "  %6.2f  %10.5e  %10.5e\n",
                 f(k), ah(k), af(k));
      end
  end

  fclose(fid);

# Graphischer Vergleich

  figure(1, "position", [100, 500, 1000, 500],
            "paperposition", [0, 0, 15, 7]);
  plot(f, a, "color", "red", f, ah, "color", "green",
       f, af, "color", "blue");
  legend("kein Fenster", "Hanning-Fenster", "Flattop-Fenster");
  grid;
  xlabel("f [Hz]");
  ylabel("Amplitude");
  print([file, ".svg"], "-dsvg");
```

Die Ausgabedatei enthält die folgenden Ergebnisse:

```
     f      Hanning       Flattop
  ------------------------------
    8.55  7.13442e-04  9.10927e-02
    9.03  2.92257e-03  3.13496e-01
    9.50  2.35198e-01  4.79706e-01
    9.98  4.99440e-01  4.99999e-01
   10.46  2.66373e-01  4.85248e-01
   10.93  4.11751e-03  3.33238e-01
   11.41  1.03878e-03  1.05208e-01
   18.53  6.33534e-04  3.64761e-02
   19.01  2.44954e-03  1.31025e-01
   19.49  9.49103e-02  2.05739e-01
   19.96  2.14935e-01  2.15904e-01
   20.44  1.21778e-01  2.10536e-01
   20.91  3.65215e-03  1.48053e-01
   21.39  8.72815e-04  4.86623e-02
   38.97  9.04637e-04  2.62045e-02
   39.45  1.75626e-02  4.34269e-02
   39.92  4.54703e-02  4.62933e-02
   40.40  2.89896e-02  4.55027e-02
   40.87  1.74913e-03  3.34731e-02
   41.35  3.84063e-04  1.19006e-02
   58.93  5.45654e-04  1.17216e-02
   59.41  7.18515e-03  2.05586e-02
   59.88  2.14480e-02  2.23312e-02
```

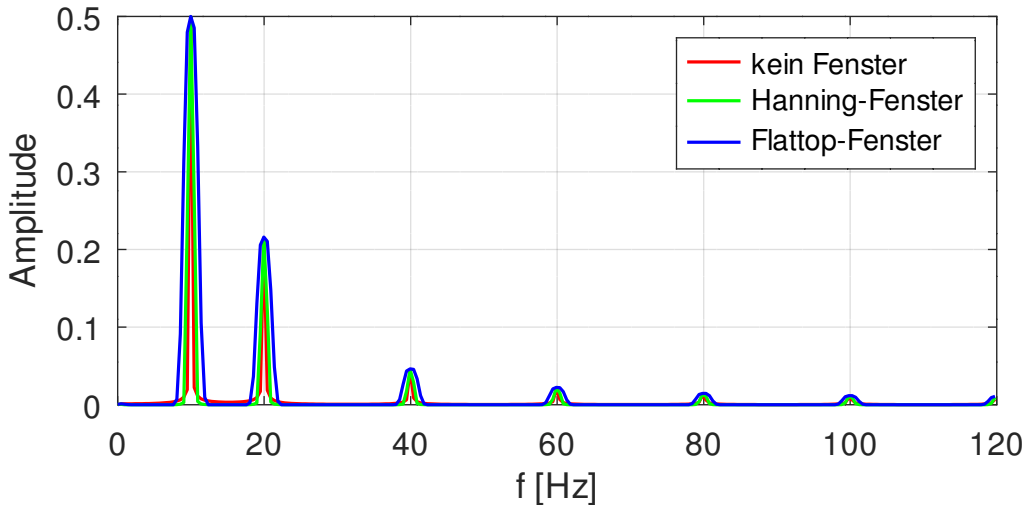

Abbildung 4.1: Fourier-Koeffizienten

```
 60.36  1.53410e-02  2.20828e-02
 60.83  1.41885e-03  1.69406e-02
 79.37  3.93544e-03  1.32484e-02
 79.84  1.36928e-02  1.47111e-02
 80.32  1.09642e-02  1.46141e-02
 80.79  1.38838e-03  1.16538e-02
 99.33  2.55675e-03  1.03305e-02
 99.80  1.05182e-02  1.17656e-02
100.28  9.41521e-03  1.17297e-02
119.76  9.31777e-03  1.09499e-02
```

Die Fourier-Koeffizienten sind in Abbildung 4.1 graphisch dargestellt.

Aufgabe 5

Das GNU Octave-Skript lädt zunächst das Package `signal`, das die für die Berechnung des Leistungsdichtespektrums benötigte Funktion **`pwelch`** enthält. Anschließend werden die Parameter für die Berechnung des Leistungsdichtespektrums definiert und die Daten eingelesen. Es handelt sich um denselben stochastischen Prozess wie in Aufgabe 3 von Übungsblatt 3.2. Die Zeitpunkte werden in dem Feld **`t`** und die Realisierungen in dem Feld **`x`** gespeichert.

```
# Übungsblatt 3.4, Aufgabe 5: Leistungsdichtespektren
# ---------------------------------------------------------------

  colors = [1, 0, 0; 0, 1, 0; 0, 0, 1; 1, 0, 1; 1, 0.5, 0;
            0, 0, 0];
  set(0, "defaultaxescolororder", colors)
  set(0, "defaultaxesfontsize", 12);

  pkg load signal

  file = mfilename();
```

```
# Parameter für Berechnung der Leistungsdichtespektren

  lenw    =  512;  % Fensterlänge
  overlap =  0.5;  % Überlappungsfaktor

# Daten einlesen

  data = dlmread("u3_2_3.csv");

  t         = data(:, 1);
  x         = data(:, 2 : end);
  [nt, nr] = size(x);
  dt        = mean(diff(t));
  fs        = 1 / dt;
```

In einer Schleife über die Realisierungen werden die Leistungsdichtespektren der einzelnen Realisierungen berechnet. Anschließend wird über die Realisierungen gemittelt.

```
# Leistungsdichtespektren

  for k = 1 : nr
      [G(:, k), f] = pwelch(x(:, k), lenw, overlap, [], fs);
  end
  Gm = mean(G, 2);
```

Zur Bewertung wird der quadratische Mittelwert aus den Zeitreihen der Realisierungen berechnet und gemittelt und zusammen mit dem aus dem gemittelten Leistungsdichtespektrum berechneten quadratischen Mittelwert ausgegeben. Für die numerische Integration des Leistungsdichtespektrums wird die Funktion **`trapz`** verwendet.

```
# Quadratischer Mittelwert

  s2x = mean(meansq(x));
  s2G = trapz(f, Gm);

  fid  = fopen([file, ".res"], "wt");
  fprintf(fid, "Quad. Mittelwert aus Zeitreihe = %8.5f\n", s2x);
  fprintf(fid, "Quad. Mittelwert aus PSD       = %8.5f\n", s2G);
  fclose(fid);
```

Zum Schluss werden alle Leistungsdichtespektren zusammen mit dem gemittelten Leistungsdichtespektrum graphisch dargestellt.

```
# Graphische Darstellung

  lgtxt{1} = "gemittelt"; k = 2;
  for n = 2 : 2 : nr
      lgtxt{k++} = sprintf("%2.0d. Realisierung", n);
  end

  figure(1, "position", [10, 10, 1000, 500],
```

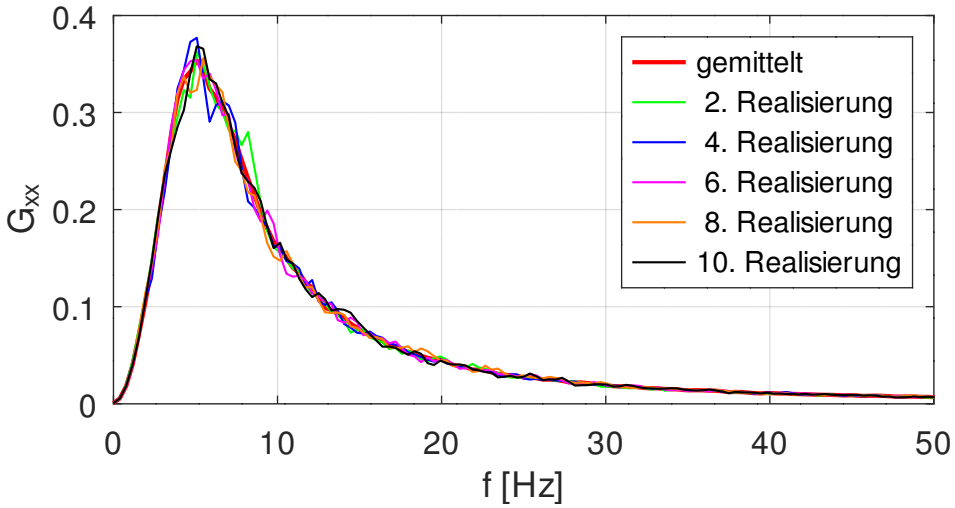

Abbildung 5.1: Leistungsdichtespektren

```
                "paperposition", [0, 0, 14, 7]);

  plot(f, Gm, "color", "red", "linewidth", 1.6,
       f, G(:, 2 : 2 : nr)', "linewidth", 0.8);
  set(gca(), "xlim", [0, 50]);
  legend(lgtxt);
  grid;
  xlabel('f [Hz]'); ylabel('G_{xx}');
  print([file, ".svg"], "-dsvg");
```

Für die quadratischen Mittelwerte werden folgende Werte ausgegeben:

```
Quad. Mittelwert aus Zeitreihe =  3.71245
Quad. Mittelwert aus PSD       =  3.71308
```

Die Werte stimmen gut miteinander überein. Abbildung 5.1 zeigt die Leistungsdichtespektren. Die Mittelung über die Realisierungen führt zu einer deutlichen Glättung des Leistungsdichtespektrums. Der gleiche Effekt lässt sich mit einer entsprechend längeren Zeitreihe erzielen.

Aufgabe 6

Das GNU Octave-Skript lädt zunächst das Package `signal`, das die Funktionen **`pwelch`** und **`cpsd`** zur Berechnung von Leistungs- und Kreuzleistungsdichtespektrum enthält. Anschließend werden die Parameter für die Berechnung definiert und die Daten eingelesen. Es handelt sich um dieselben stochastischen Prozesse wie in Aufgabe 4 von Übungsblatt 3.2. Die Zeitpunkte werden in dem Feld **`t`** und die Realisierungen der beiden Prozesse in den Feldern **`x`** und **`y`** gespeichert.

```
# Übungsblatt 3.4, Aufgabe 6:
# Leistungs- und Kreuzleistungsdichtespektrum
# ------------------------------------------------------------
```

```
  set(0, "defaultaxesfontsize", 12);

  pkg load signal

  file = mfilename();

# Parameter für Berechnung der Leistungsdichtespektren

  lenw    =  512;  % Fensterlänge
  overlap =  0.5;  % Überlappungsfaktor

# Daten einlesen

  data = dlmread("u3_2_4.csv");

  t  = data(:, 1); x  = data(:, 2); y  = data(:, 3);
  nt = rows(t);
  dt = mean(diff(t));
  fs = 1 / dt;
```

Die Berechnung der Leistungsdichtespektren erfolgt mit der Funktion **pwelch**. Das Kreuzleistungsdichtespektrum wird mit der Funktion **cpsd** berechnet.

```
# Leistungs- und Kreuzleistungsdichtespektren

  [Gxx, f] = pwelch(x, lenw, overlap, [], fs);
  Gyy      = pwelch(y, lenw, overlap, [], fs);
  Gxy      = cpsd(x, y, lenw, overlap, [], fs);
```

Leistungs- und Kreuzleistungsdichtespektren werden in einem Bild dargestellt, das zwei Diagramme enthält. Das linke Bild zeigt die reellen Leistungsdichtespektren der beiden stochastischen Prozesse und das rechte Real- und Imaginärteil des komplexen Kreuzleistungsdichtespektrums.

```
# Graphische Darstellung

  figure(1, "position", [10, 500, 1000, 500],
            "paperposition", [0, 0, 15, 8]);
  subplot(1, 2, 1, "position", [0.13, 0.17, 0.34, 0.75]);
     plot(f, Gxx, "color", "red", f, Gyy, "color", "green");
     legend("G_{xx}", "G_{yy}");
     xlim([0, 50]);
     grid;
     xlabel("f [Hz]");
  subplot(1, 2, 2, "position", [0.57, 0.17, 0.34, 0.75]);
     plot(f, real(Gxy), "color", "red",
          f, imag(Gxy), "color", "green");
     legend("Re(G_{xy})", "Im(G_{xy})");
     xlim([0, 50]);
     grid;
     xlabel("f [Hz]");
  print([file, ".svg"], "-dsvg");
```

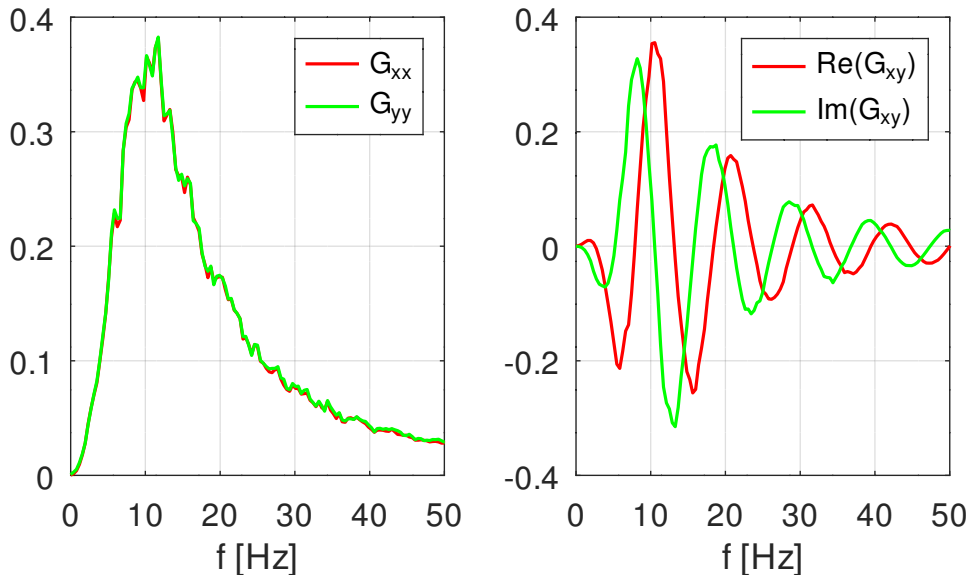

Abbildung 6.1: Leistungs- und Kreuzleistungsdichtespektren

Die Leistungs- und Kreuzleistungsdichtespektren sind in Abbildung 6.1 graphisch dargestellt. Die Leistungsdichtespektren sind praktisch identisch.

Aufgabe 7

Das GNU Octave-Skript beginnt mit der Definition einiger Parameter für die graphische Darstellung. Anschließend werden die Packages `signal` und `statistics` geladen, die die benötigten Funktionen **`resample`** und **`pwelch`** sowie **`normpdf`** enthalten, und die Ausgabedatei geöffnet.

```
# Übungsblatt 3.4, Aufgabe 7: Staubsauger
#
# ------------------------------------------------------------

  set(0, "defaultaxesfontsize", 12);

  pkg load signal
  pkg load statistics

  file = mfilename();
  fid  = fopen([file, ".res"], "wt");
```

Dann werden eine Reihe von Konstanten definiert. Die Konstante **`nbin`** wird für die Berechnung der Wahrscheinlichkeitsdichte benötigt. Die Konstante **`lenw`** legt die Fensterlänge und die Konstante **`overlap`** den Überlappungsfaktor für die Berechnung des Leistungsdichtespektrums fest. Die Konstante **`taumax`** gibt an, für welches Intervall die Autokorrelation berechnet werden soll. Die Konstante **`nsplit`** legt fest, in wie viele Intervalle ein Zeitschritt bei der Ermittlung der Nulldurchgänge unterteilt werden soll.

```
# Konstanten

  nbin    =  100;  % Intervalle für die Häufigkeitsverteilung
  lenw    = 2048;  % Fensterlänge
  overlap =  0.5;  % Überlappungsfaktor
  taumax  =  0.1;  % Zeit für Autokorrelation
  nsplit  =    4;  % Faktor für Intervallunterteilung

  fprintf(fid, "Parameter:\n\n");
  fprintf(fid, "nbin    = %4.0f, lenw    = %4.0f, ", nbin, lenw);
  fprintf(fid, "overlap = %4.1f\n", overlap);
  fprintf(fid, "taumax  = %4.1f, nsplit  = %4.0f\n\n",
          taumax, nsplit);
```

Die Funktion **audioread** liest die Zeitreihe **x** und die Abtastrate **fs** aus der gegebenen Datei.

```
# Daten einlesen

  [x, fs] = audioread([file, ".wav"]);

  nx = length(x); T  = nx / fs;

  fprintf(fid, "Anzahl Messwerte = %8.0f\n", nx);
  fprintf(fid, "Messdauer        = %8.5f s\n", T);
```

a) Statistische Kennwerte

Mittelwert, Varianz, Standardabweichung, Kurtosis und Schiefe werden mit den in GNU Octave zur Verfügung stehenden Funktionen berechnet und ausgegeben.

```
# Mittelwerte

  xm = mean(x); xv = var(x); xs = std(x);

  fprintf(fid, "\nStatistische Kennwerte der Zeitreihe:\n\n");
  fprintf(fid, "Mittelwert          = %8.5f Pa\n", xm);
  fprintf(fid, "Varianz             = %8.5f Pa²\n", xv);
  fprintf(fid, "Standardabweichung  = %8.5f Pa\n", xs);

# Kurtosis und Schiefe

  b2 = kurtosis(x); g1 = skewness(x);

  fprintf(fid, "Kurtosis            = %8.5f\n", b2);
  fprintf(fid, "Schiefe             = %8.5f\n", g1);
```

Die Ausgabedatei enthält folgende Ergebnisse:

```
Parameter:

nbin    =  100, lenw    = 2048, overlap =  0.5
taumax  =  0.1, nsplit  =    4
```

```
Anzahl Messwerte =  1332989
Messdauer        = 30.22651 s

Statistische Kennwerte der Zeitreihe:

Mittelwert          = -0.00033 Pa
Varianz             =  0.00857 Pa²
Standardabweichung  =  0.09258 Pa
Kurtosis            =  3.00254
Schiefe             =  0.00505
```

Aus diesen Werten kann abgelesen werden, dass es sich mit sehr guter Näherung um einen Gaußschen stochastischen Prozess handelt. Der Mittelwert ist nahezu null.

Zur Kontrolle wird die Wahrscheinlichkeitsdichte aus der Häufigkeitsverteilung ermittelt und zusammen mit der Gaußschen Normalverteilung dargestellt.

```
# Häufigkeitsverteilung

  xmax  = max(abs(x));
  edges = linspace(-xmax, xmax, nbin + 1);
  xc    = 0.5 * (edges(1 : nbin) + edges(2 : end));

  N = hist(x, xc);

# Wahrscheinlichkeitsdichtefunktion

  dx   = mean(diff(edges));
  pdfx = N / (nx * dx);
  pdfg = normpdf(xc, xm, xs);

  figure(1, "position", [100, 400, 600, 600],
            "paperposition", [0, 0, 12.5, 8.5]);
  plot(xc, pdfx, "color", "green", "marker", "+",
       xc, pdfg, "color", "red");
  legend("Daten", "Gauss");
  grid;
  xlim([-3 * xs, 3 * xs]);
  xlabel("x [Pa]"); ylabel("p(x) [1/Pa]");
  print([file, "pdf.svg"], "-dsvg");
```

Abbildung 7.1 zeigt, dass die Wahrscheinlichkeitsdichte sehr gut mit der Gaußschen Normalverteilung übereinstimmt.

b) Anzahl der Nulldurchgänge pro Zeiteinheit

Eine hinreichende Bedingung für einen Nulldurchgang in einem Zeitintervall ist, dass zwei aufeinander folgende Werte der Zeitreihe unterschiedliche Vorzeichen haben, d. h. dass gilt:

$$x_n x_{n+1} < 0$$

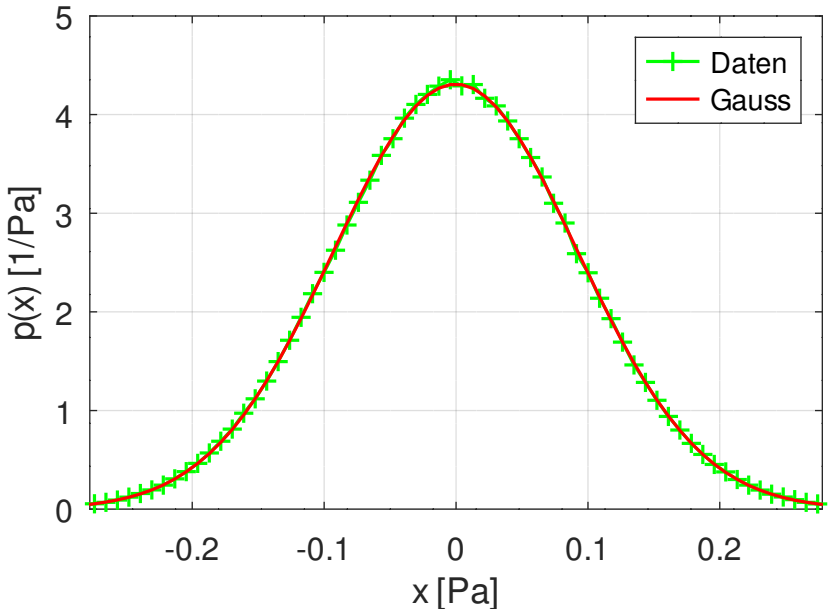

Abbildung 7.1: Wahrscheinlichkeitsdichte

Diese Bedingung ist jedoch nicht notwendig, da auch zwischen Werten mit gleichem Vorzeichen ein Nulldurchgang liegen kann, wenn beide Werte nahe bei null liegen. Daher wird zunächst die Abtastrate um den Faktor **`nsplit`** erhöht, d. h. jedes Zeitintervall wird in **`nsplit`** Intervalle unterteilt.

```
# Nulldurchgänge pro Zeit

  y = resample(x, nsplit, 1);
  p  = y(1 : end-1) .* y(2 : end);
  N0 = nnz(p < 0) / T;

  fprintf(fid, "Nulldurchgänge      = %8.2f 1/s\n", N0);

# Graphische Darstellung eines typischen Intervalls

  t1 = 12.0005; t2 = 12.001;
  tx = (0 : nx - 1) / fs;
  ty = (0 : nsplit * nx - 1) / (nsplit * fs);
  ix = find(tx >= t1 & tx <= t2);
  iy = find(ty >= t1 & ty <= t2);

  figure(2, "position", [150, 450, 800, 500],
            "paperposition", [0, 0, 14, 10]);
  plot(tx(ix), x(ix), "marker", "o", "linestyle", "none",
       ty(iy), y(iy));
  legend('Messwerte', 'interpoliert',"location", "southeast");
  grid;
  xticks(linspace(t1, t2, 3));
  xlabel('t [s]'); ylabel('x [Pa]');
  print([file, "x.svg"], "-dsvg");
```

Die Ausgabedatei enthält folgendes Ergebnis für die Anzahl der Nulldurch-

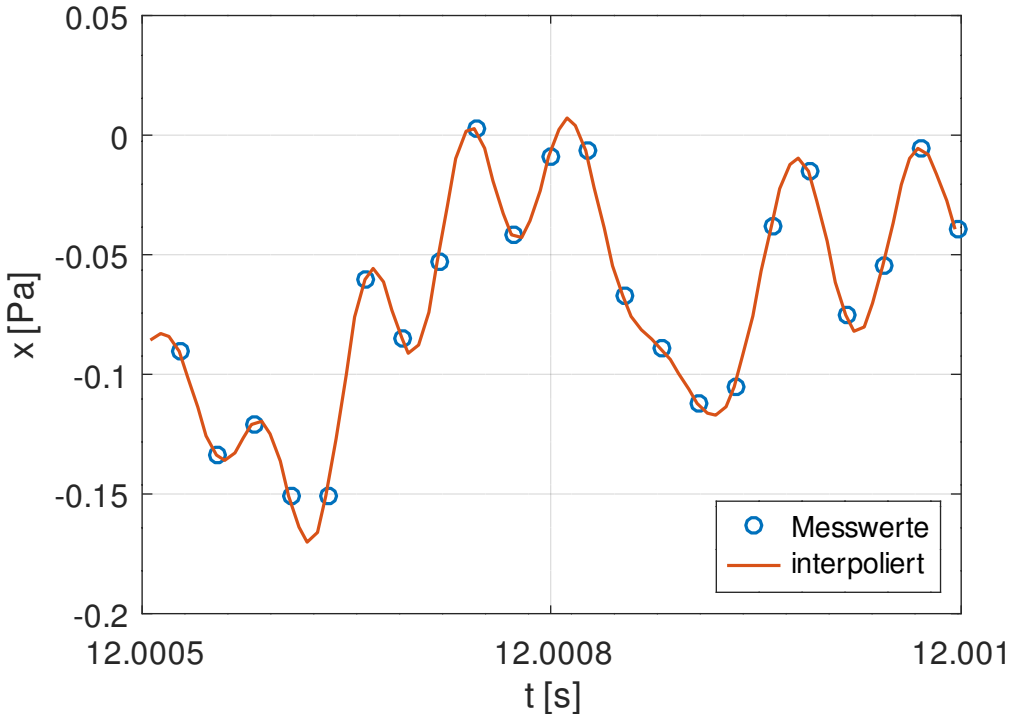

Abbildung 7.2: Ausschnitt der Zeitreihe

gänge pro Zeiteinheit:

```
Nulldurchgänge      =  5164.18 1/s
```

Abbildung 7.2 zeigt einen Ausschnitt der Zeitreihe. Dargestellt sind die Messwerte zusammen mit der von der Funktione **`resample`** berechneten Interpolation durch die Kardinalreihe. Der Abschnitt zeigt kurz nach $12.0008\ \mathrm{s}$ zwei Nulldurchgänge, die aus den gemessenen Werten mit dem verwendeten Kriterium nicht erkannt werden.

c) Autokorrelation

Die Autokorrelation wird mit der Funktion **`xcorr`** berechnet und anschließend graphisch dargestellt.

```
# Autokorrelation

  maxlag = ceil(taumax * fs);
  [Rxx, lag] = xcorr(x, maxlag, "unbiased");
  tau = lag / fs;

  figure(3, "position", [200, 500, 600, 400],
            "paperposition", [0, 0, 15, 8]);
  plot(tau, Rxx);
  grid;
  xlabel('\tau [s]'); ylabel('R_{xx} [Pa^2]');
  print([file, "Rxx.svg"], "-dsvg");
```

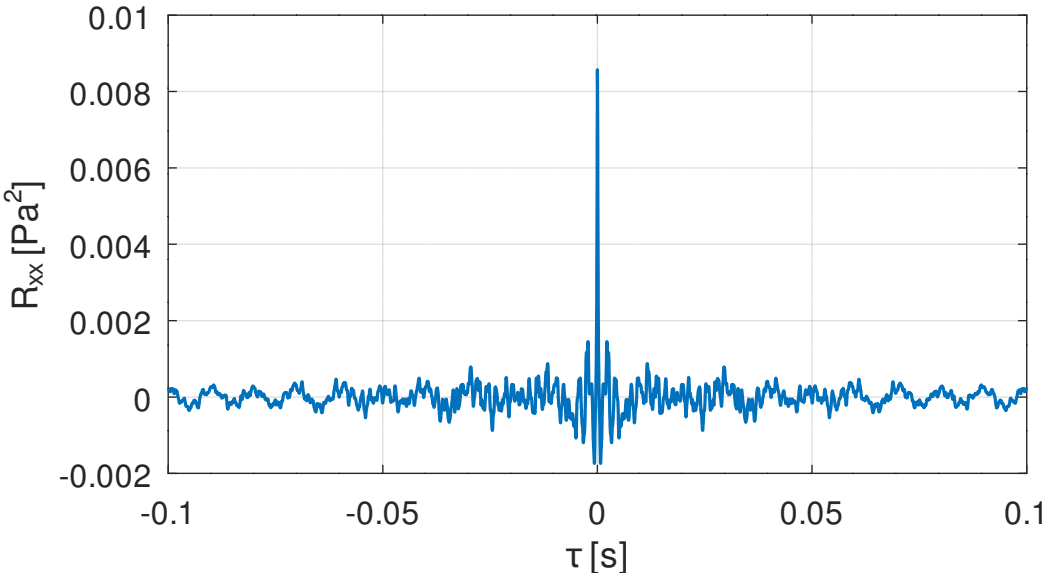

Abbildung 7.3: Autokorrelation

Abbildung 7.3 zeigt, dass die Autokorrelation sehr schnell abfällt.

d) Leistungsdichtespektrum

Das Leistungsdichtespektrum wird mit der Funktion **pwelch** aus der Zeitreihe berechnet. Durch numerische Integration mit der Funktion **trapz** kann daraus die Varianz sowie die Anzahl der Nulldurchgänge pro Zeiteinheit berechnet werden.

```
# Leistungsdichtespektrum

  [Gxx, f] = pwelch(x, lenw, overlap, [], fs);

  figure(4, "position", [300, 600, 600, 400],
            "paperposition", [0, 0, 15, 8]);
  loglog(f(2 : end), Gxx(2 : end), "color", "green");
  grid;
  xlabel('f [Hz]'); ylabel('G_{xx} [Pa^2/Hz]');
  print([file, "psd.svg"], "-dsvg");

  xv  = trapz(f, Gxx);
  xvf = trapz(f, f.^2 .* Gxx);
  N0  = 2 * sqrt(xvf / xv);

  fprintf(fid, "\nStatistische Kennwerte aus PSD:\n\n");
  fprintf(fid, "Varianz              = %8.5f Pa²\n", xv);
  fprintf(fid, "Nulldurchgänge       = %8.2f 1/s\n", N0);

  fclose(fid);
```

Das Leistungsdichtespektrum ist in Abbildung 7.4 dargestellt. Es fällt bei ca. 20 kHz sehr stark ab, was zeigt, dass die Messung mit einem Tiefpassfilter er-

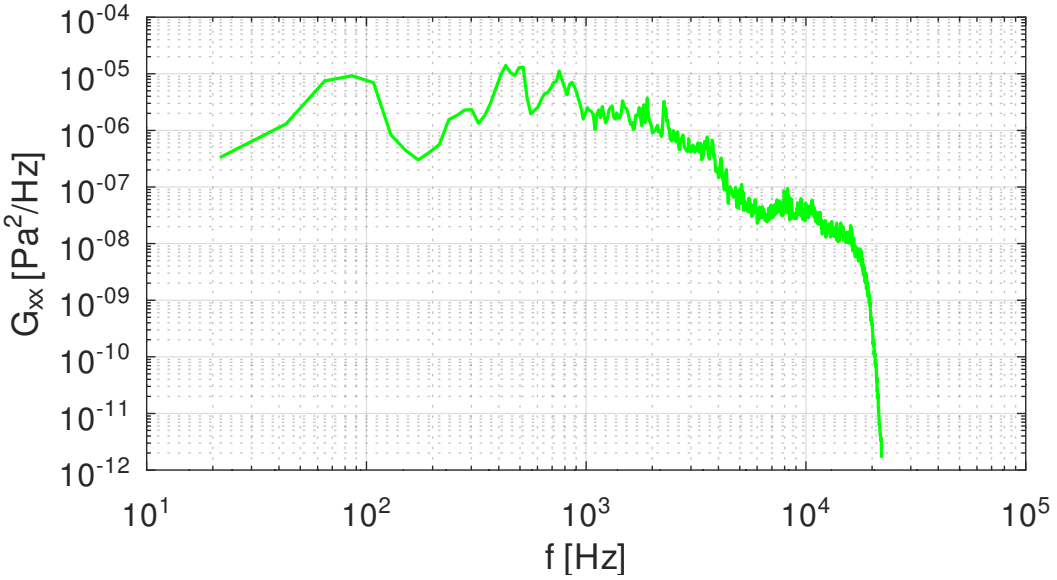

Abbildung 7.4: Leistungsdichtespektrum

folgte, um den Alias-Effekt zu vermeiden.

Die Ausgabedatei enthält folgende Ergebnisse:

```
Statistische Kennwerte aus PSD:

Varianz              =  0.00858 Pa²
Nulldurchgänge       =  5163.22 1/s
```

Diese Werte stimmen sehr gut mit den direkt aus der Zeitreihe ermittelten Werten überein.