Task1: Signal Processing I

Name: Nils Wildt **Degree:** MA **ID:** 20807870

Problem 1: Sampling (10 points)

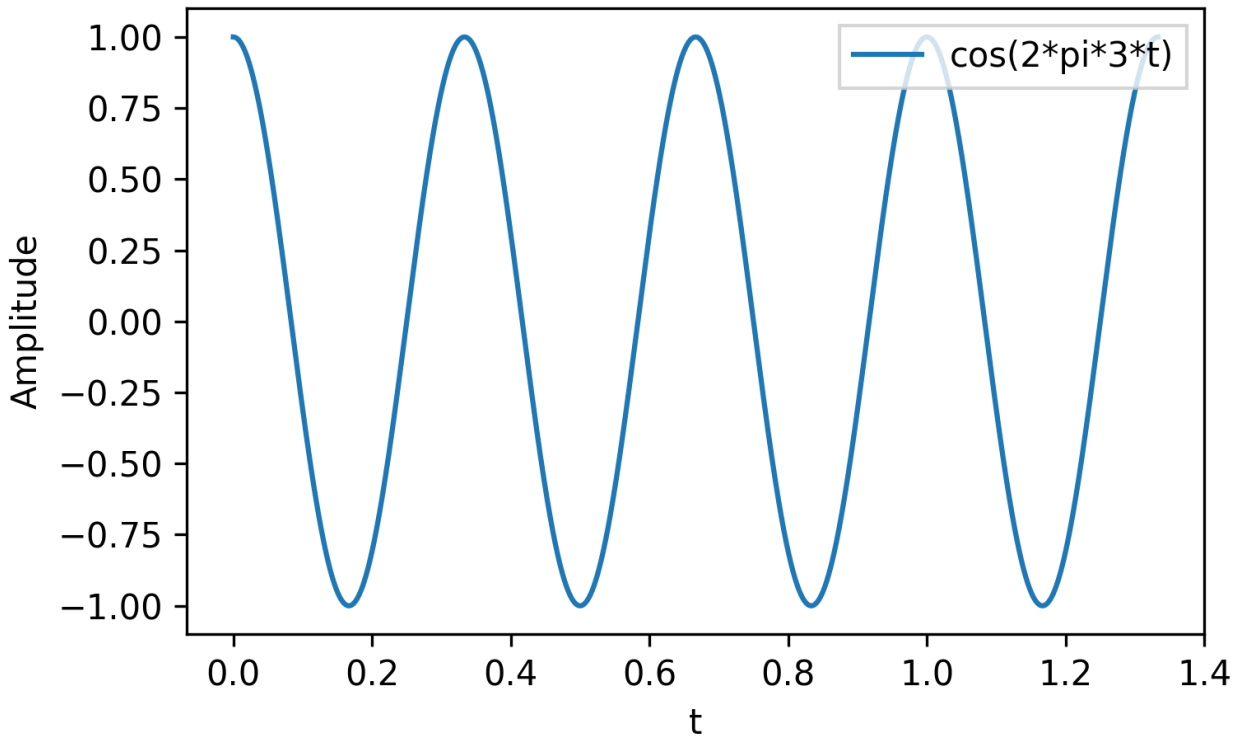
(a) What is the difference between a continuous (or analogue) and discrete (or digital) signals?

- A continuous signal has the property, that for every chosen time point data is available -- and may also change for each timechange, no matter how small/big. In mathematical terms that means, that the time can be drawn from the positive real axis, and as the real numbers are complete, for every chosen point, a value exists.

(b) Plot a 3 Hz cosine wave with high sampling rate (nearly analog signal). Please connect sampled points and plot only four cycles of the wave.

```
%run initscript.ipy # For ipython notebook. Otherwise it needs to be preloaded

## Please note, that in the following speaking variables are used to improve readability.

# Symbolic function is a "true" continuous function for t \in \mathbb{R}.
t_sy,f_sy = sy.symbols(r't,f')
cos_sy = sy.cos(2.0*sy.pi*f_sy*t_sy)

## Variables setup
f = 3.0 # 3 Hz
T = 1.0/f # Periode time

f_sample = 1.0E4
n_cycles = 4.0 # Number of cycles to be plottet

t_start = 0.0
t_end = n_cycles*T
t_sample =  np.arange(t_start,t_end,1/f_sample)

# Substitute and lambdify to evaluate
cos = cos_sy.subs({f_sy:f})
cos = lambdify(t_sy, cos, modules=["numpy"],dummify=False)
y = cos(t_sample)

# Plotting
fig, ax = newfig(0.9)
ax.plot(t_sample,y,label=u'cos(2*pi*3*t)');
legend = ax.legend(loc="upper right", frameon=True, facecolor='white', shadow=False,
fancybox=False);
ax.set_xlabel(r't');
```

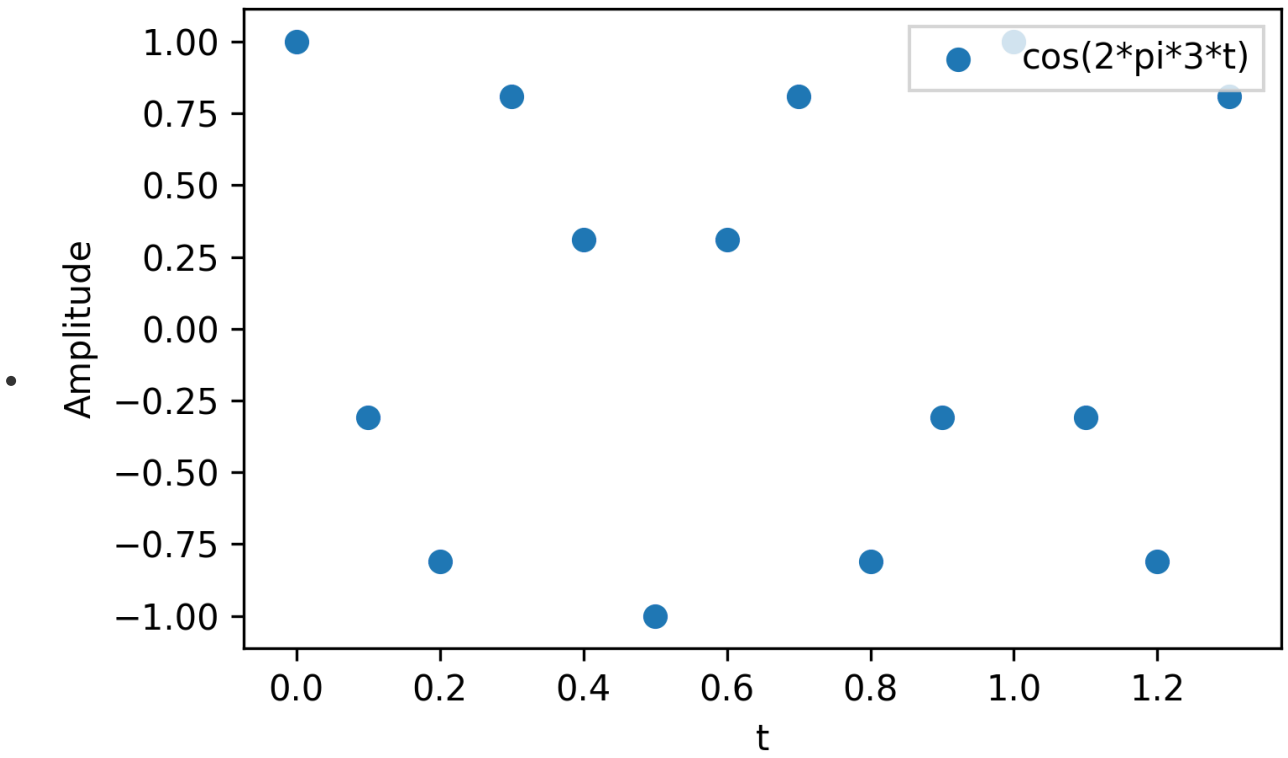
```
ax.set_ylabel("Amplitude");
# ax.set_aspect('equal')
# ax.grid(True, which='both');
# ax.ticklabel_format(style='sci', scilimits=(-3, 4), axis='both');
savefig("1_b")
# plt.show()
```



(c) Plot the 3 Hz cosine wave after sampling with 10 Hz. Please do not connect sampled points and plot the sampled data for four cycles of the wave.

```
## Please note, that in the following speaking variables are used to improve readability.

# Symblic function is a "true" continuous function for t \in \mathbb{R}.
t_sy,f_sy = sy.symbols(r't,f')
cos_sy = sy.cos(2.0*sy.pi*f_sy*t_sy)

## Variables setup
f = 3.0 # 3 Hz
T = 1.0/f # Periode time

f_sample = 10.0
n_cycles = 4.0 # Number of cycles to be plottet

t_start = 0.0
t_end = n_cycles*T
t_sample =  np.arange(t_start,t_end,1/f_sample)

# Substitute and lambdify to evaluate
cos = cos_sy.subs({f_sy:f})
cos = lambdify(t_sy, cos, modules=["numpy"],dummify=False)
```

```
y = cos(t_sample)

fig, ax = newfig(0.9)
ax.scatter(t_sample,y,label=u'cos(2*pi*3*t)');
legend = ax.legend(loc="upper right", frameon=True, facecolor='white', shadow=False,
fancybox=False);
ax.set_xlabel(r't');
ax.set_ylabel("Amplitude");
# ax.set_aspect('equal')
# ax.grid(True, which='both');
# ax.ticklabel_format(style='sci', scilimits=(-3, 4), axis='both');
savefig("1_c")
plt.show()
```



(d) Plot the 6 Hz sine wave after sampling with 12 Hz. Please do not connect sampled points and plot only four cycles of the wave. Do you think that you can measure this wave if you add a phase angle (φ) on this sine wave? for example, the wave is sin(2πft + φ).

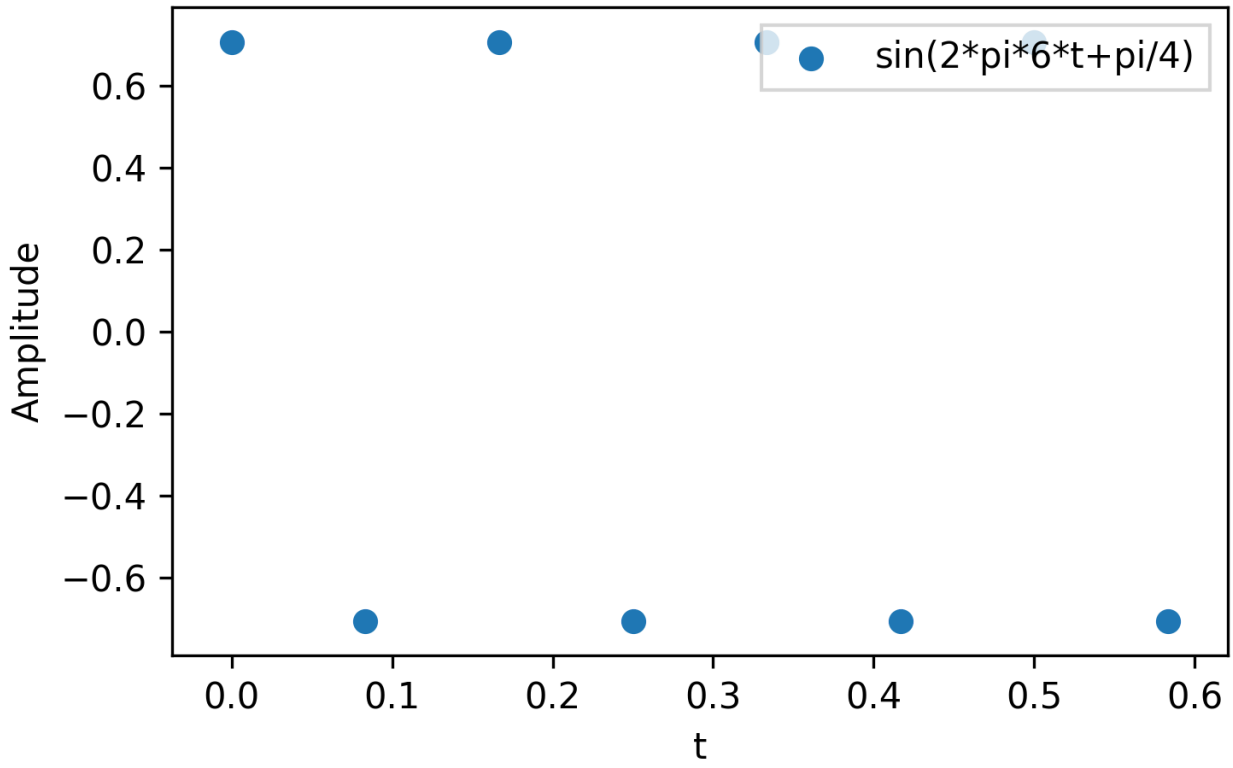
```
## Please note, that in the following speaking variables are used to improve readability.

# Symblic function is a "true" continuous function for t \in \mathbb{R}.
t_sy,f_sy, phi_sy = sy.symbols(r't,f,\phi')
sin_sy = sy.sin(2.0*sy.pi*f_sy*t_sy+phi_sy)

## Variables setup
f = 6.0
T = 1.0/f # Periode time
phi = 1*np.pi/4
```

```
f_sample = 12.0
n_cycles = 4.0 # Number of cycles to be plottet

t_start = 0.0
t_end = n_cycles*T
t_sample =  np.arange(t_start,t_end,1/f_sample)
# np.linspace(t_start,t_end,int(t_end*f_sample))

# Substitute and lambdify to evaluate
sin = sin_sy.subs({f_sy:f,phi_sy:phi})
sin = lambdify(t_sy, sin, modules=["numpy"],dummify=False)
y = sin(t_sample)

fig, ax = newfig(0.9)
ax.scatter(t_sample,y,label=u'sin(2*pi*6*t+0)');
legend = ax.legend(loc="upper right", frameon=True, facecolor='white', shadow=False,
fancybox=False);
ax.set_xlabel(r't');
ax.set_ylabel("Amplitude");
# ax.set_aspect('equal')
# ax.grid(True, which='both');
# ax.ticklabel_format(style='sci', scilimits=(-3, 4), axis='both');
# savefig("1_d_1")
plt.show()
```

- No, as the Nyquist–Shannon sampling theorem is not fulfilled. However the reproducibility of frequency seems to change if plotted... Why though? Probably dependent on s.th. like if the last point is included and thus the curve is perfectly aligned to the sampling frequency.
- That's how I tested:

  ```
  fig, ax = newfig(0.9)
  f, Pxx_den = signal.periodogram(y, f_sample,nfft=2**10)
  plt.plot(f, Pxx_den)
  plt.xlabel('frequency [Hz]')
  plt.ylabel('PSD [V**2/Hz]')
  plt.show()
  ```

-



Problem 2: Aliasing (10 points)

We learned when aliasing occurs and how it is related to the sampling frequency of the data acquisition system.

(a) A 10 Hz sine wave is sampled at 12 Hz. Compute the alias frequency that can be represented in the resulting sampled signal. Plot the wave and sampled points.

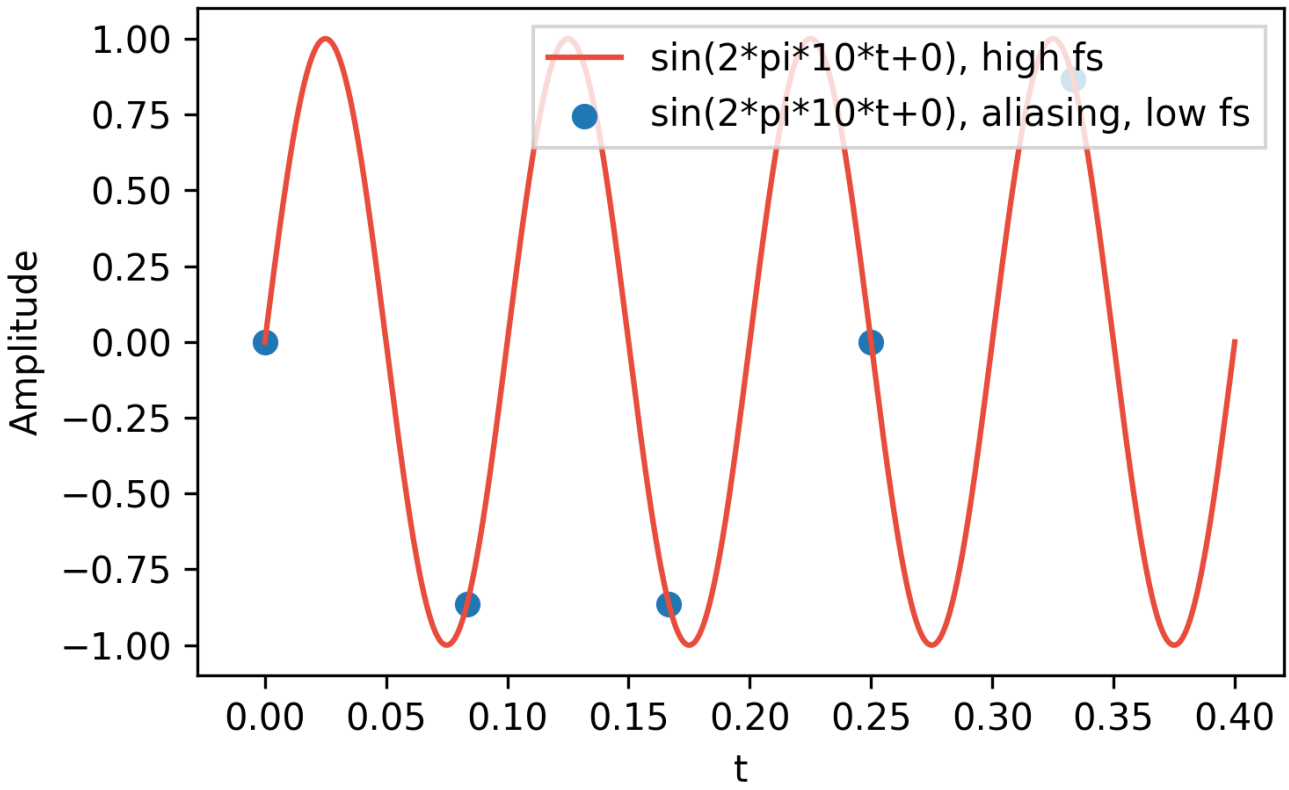
```
def get_samplefreq(f,fs):
    """Calculates the theoretical frequency of a signal, while undersampling.
    @Input: f = frequency of the signal, fs = sampling frequency
    @Returns: real value of the resulting frequency that will be measured.
    """
    return np.abs(fs/2 - np.abs(f-fs/2))
```

- The nyquist frequency lies at fs/2 = 6Hz.
- The frequency of the digitized signal can be found at 2Hz.

The plots:

```
## Please note, that in the following speaking variables are used to improve readability.

# Symblic function is a "true" continuous function for t \in \mathbb{R}.
t_sy,f_sy, phi_sy = sy.symbols(r't,f,\phi')
sin_sy = sy.sin(2.0*sy.pi*f_sy*t_sy+phi_sy)
```

```
## Variables setup
f = 10.0
T = 1.0/f # Periode time
phi = 0*np.pi/4

f_sample = 12.0
n_cycles = 4.0 # Number of cycles to be plottet

t_start = 0.0
t_end = n_cycles*T
t_sample =  np.arange(t_start,t_end,1.0/f_sample)
t_fine = np.linspace(t_start,t_end,1E5)

# Substitute and lambdify to evaluate
sin = sin_sy.subs({f_sy:f,phi_sy:phi})
sin = lambdify(t_sy, sin, modules=["numpy"],dummify=False)
y = sin(t_sample)
N = len(y)

fig, ax = newfig(0.9)
ax.scatter(t_sample,y,label=u'sin(2*pi*10*t+0), aliasing, low fs');
ax.plot(t_fine,sin(t_fine),label=u'sin(2*pi*10*t+0), high fs',color=colors[1],alpha=1);
legend = ax.legend(loc="upper right", frameon=True, facecolor='white', shadow=False,
fancybox=False);
ax.set_xlabel(r't');
ax.set_ylabel("Amplitude");
# ax.set_aspect('equal')
# ax.grid(True, which='both');
# ax.ticklabel_format(style='sci', scilimits=(-3, 4), axis='both');
savefig("2_a")
plt.show()
```

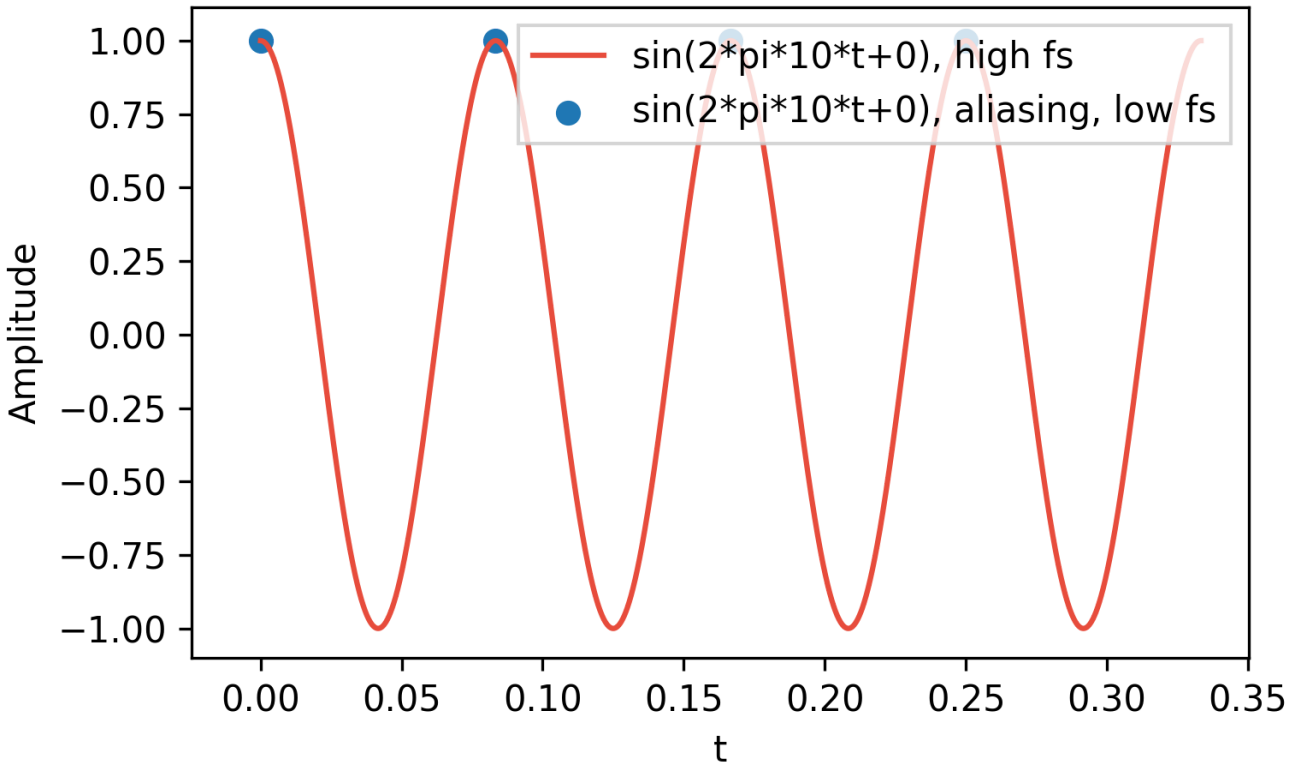


(b) A 12 Hz cosine wave is sampled at 12 Hz. Compute the alias frequency that can be represented in the resulting sampled signal. Plot the wave and sampled points.

- The nyquist frequency is at 6Hz
- The digitized Frequency is at 0 Hz.

```
## Please note, that in the following speaking variables are used to improve readability.

# Symblic function is a "true" continuous function for t \in \mathbb{R}.
t_sy,f_sy, phi_sy = sy.symbols(r't,f,\phi')
sin_sy = sy.sin(2.0*sy.pi*f_sy*t_sy+phi_sy)

## Variables setup
f = 12.0
T = 1.0/f # Periode time
phi = 2*np.pi/4

f_sample = 12.0
n_cycles = 4.0 # Number of cycles to be plottet

t_start = 0.0
t_end = n_cycles*T
t_sample =  np.arange(t_start,t_end,1.0/f_sample)
t_fine = np.linspace(t_start,t_end,1E5)

# Substitute and lambdify to evaluate
sin = sin_sy.subs({f_sy:f,phi_sy:phi})
sin = lambdify(t_sy, sin, modules=["numpy"],dummify=False)
```

```
y = sin(t_sample)

fig, ax = newfig(0.9)
ax.scatter(t_sample,y,label=u'sin(2*pi*10*t+0), aliasing, low fs');
ax.plot(t_fine,sin(t_fine),label=u'sin(2*pi*10*t+0), high fs',color=colors[1],alpha=1);
legend = ax.legend(loc="upper right", frameon=True, facecolor='white', shadow=False,
fancybox=False);
ax.set_xlabel(r't');
ax.set_ylabel("Amplitude");
# ax.set_aspect('equal')
# ax.grid(True, which='both');
# ax.ticklabel_format(style='sci', scilimits=(-3, 4), axis='both');
savefig("2_b")
plt.show()
```



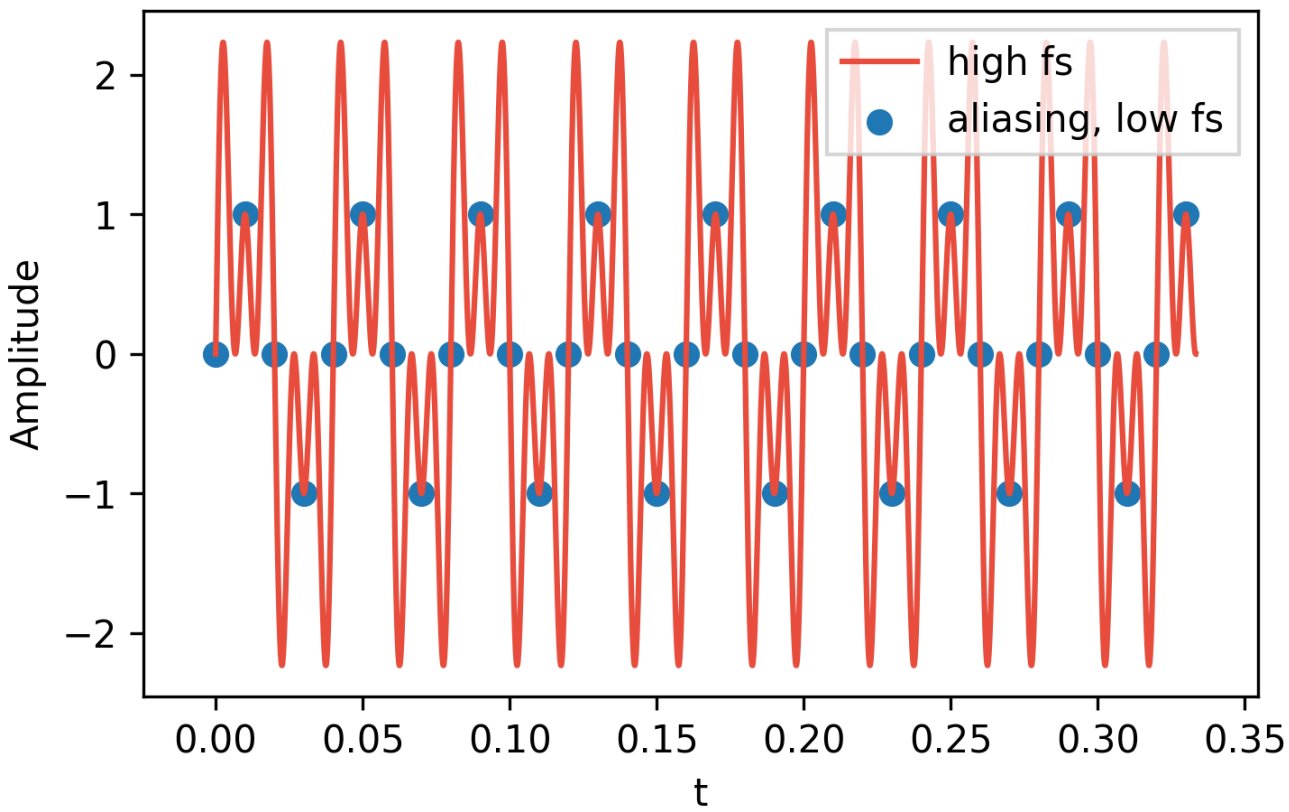
(c) Assume that the measured signal has a complex periodic signal of the form:

If the signal is sampled at 100 Hz, determine the frequency content of the resulting discrete response signal.

- Only 25 can be detected, as f_nyquist = 100/2 = 50Hz.
- The other two frequency will cause an alias at 25Hz.

```
plt.close("all")
## Please note, that in the following speaking variables are used to improve readability.


## Variables setup
f = 12.0
```

```
T = 1.0/f # Periode time
phi = 0*np.pi/4

f_sample = 100.0
n_cycles = 4.0 # Number of cycles to be plottet

t_start = 0.0
t_end = n_cycles*T
t_sample =  np.arange(t_start,t_end,1/f_sample)
t_fine = np.linspace(t_start,t_end,int(1E6))

# Substitute and lambdify to evaluate
fun = lambda t: np.sin(2*np.pi*25*t) + np.sin(2*np.pi*75*t)+np.sin(2*np.pi*125*t)
y = fun(t_sample)

fig, ax = newfig(0.9)
ax.scatter(t_sample,y,label=u'aliasing, low fs');
ax.plot(t_fine,fun(t_fine),label=u'high fs',color=colors[1],alpha=1);
legend = ax.legend(loc="upper right", frameon=True, facecolor='white', shadow=False,
fancybox=False);
ax.set_xlabel(r't');
ax.set_ylabel("Amplitude");
# ax.set_aspect('equal')
# ax.grid(True, which='both');
# ax.ticklabel_format(style='sci', scilimits=(-3, 4), axis='both');
savefig("2_c")
plt.show()
```



All Aliases were calculated by:

```
testpairs = [(10,12),(12,12),(25,100),(75,100),(125,100)]
[get_samplefreq(*i) for i in testpairs ]
```

with

```
get_samplefreq(f,fs)
```

from above.

Problem 3: Issues in Sampling (20 points)

(a) Please explain a quantization error. When do they occur? How to avoid?

- A signal that is quantizized can't have a arbitrary fine value. For example reducing a continuous signal at one time to the normal numbers would for example round a lot of numbers to fit into that format. If only a small number of colors are for example stored, like 256, a lot of finer color variations have to be neglected.
- A resolution that is high enough, to preserve important features can minimize the impact of the quantization error.
- Dithering can also help to reduce the negative impact of quantization error, by hiding the distortion by adding noise which makes the error more uncorrelated and less noticeable.

(b) Please explain a clipping error. When do they occur? How to avoid?

- If a signal exceeds for example a sensors threshold, clipping can occur. That means, that the signal is cut as for example a higher voltage of an analog signal is not available. Also in terms of visual content clipping can become a problem. It results in desaturation, when rgb values are not accordingly transformed to 1-255 but exceed this range. Also a over exposured image shows the effect of clipping -- the no contrasts can be seen anymore between neighboring pixels.
- For audio equipment the pre-gain could be adjusted to control, that the signal does not exceed the dynamic range.
- A limiter filter can prevent hard clipping, by damping to high signals.

(c) Please explain an oversampling issue. When do they occur? What are the consequence of the oversampling?

- Oversampling means to sample with a sampling frequency that is (significantly) higher than the required nyquist_frequency+eps.
- The amount of data is increased --> Vectorlength increased.
- The SNR is increased (see https://en.wikipedia.org/wiki/Oversampling).

(d) Assume that a building vibrates with a 10 Hz sine wave:

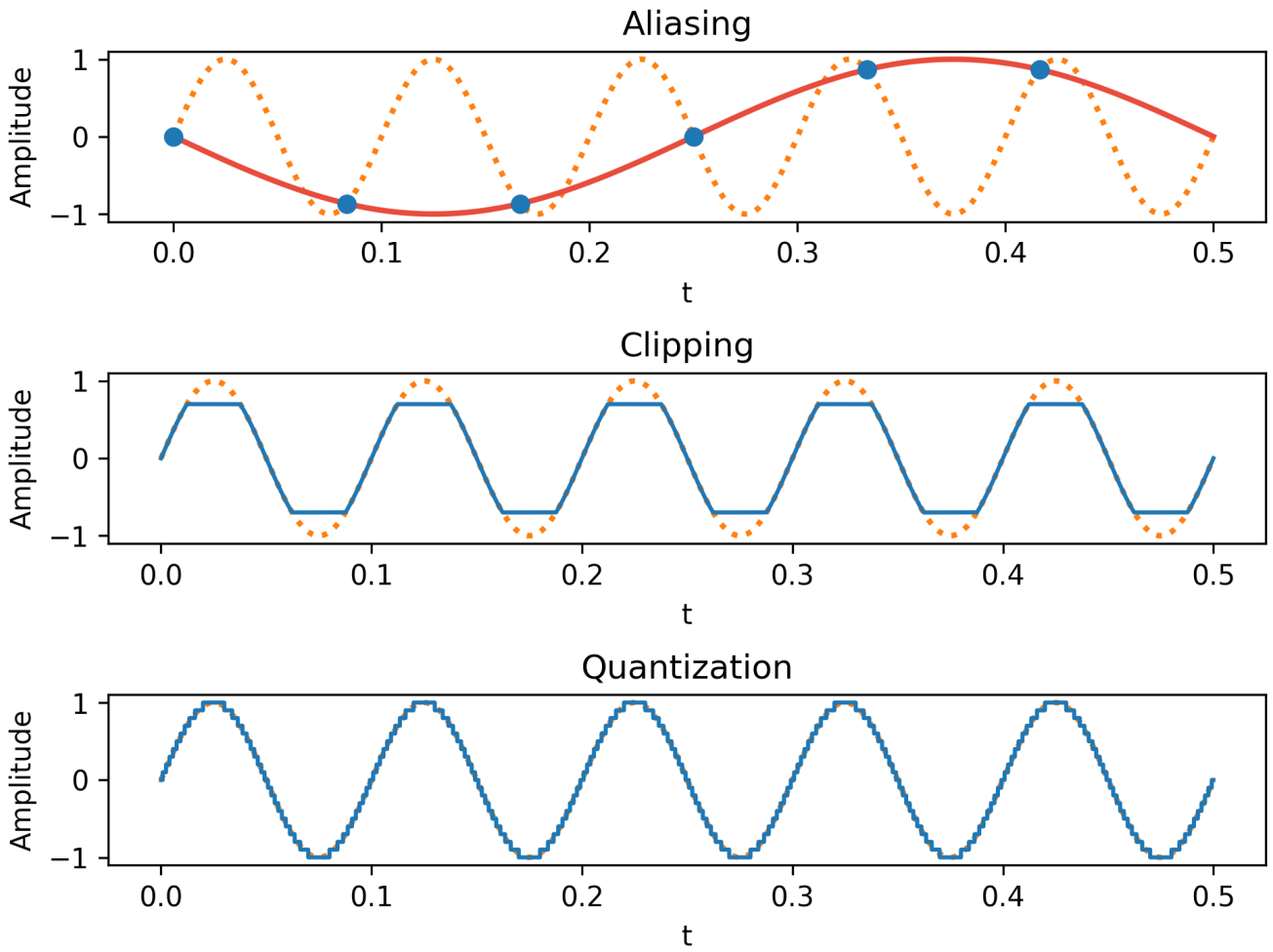
and you measured this vibration using your accelerometer and DAQ system. Please write a code to create three different sampled signals that are damaged by aliasing, quantization error and clipping error, respectively. Also, generate a signal having an oversampling issue. You should understand the topics of *Aliasing*, *Quantization*, *Clipping*, and *Oversampling* in `data_cquisition_V3.mlx` to solve this problem. You need

to explain why your sampled signals contain each of these errors/issue. You can assume any sampling rate, output range, or ADC resolution to generate these signals. Your code should plot these three signals.

- Answer:

```
plt.close("all")
## Please note, that in the following speaking variables are used to improve readability.


## Variables setup
f = 10.0
T = 1.0/f # Periode time
phi = 0*np.pi/4

f_sample = 12 # Aliasing!
n_cycles = 5.0 # Number of cycles to be plottet

t_start = 0.0
t_end = n_cycles*T
t_sample =  np.arange(t_start,t_end,1/f_sample)
t_fine = np.linspace(t_start,t_end,int(1E6))

# Substitute and lambdify to evaluate
fun = lambda t: np.sin(2*np.pi*f*t+phi)
y_aliasing = fun(t_sample)
fun_alias = lambda t: np.sin(2*np.pi*get_samplefreq(f,f_sample)*t)
# Plotting
fig, axarr = plt.subplots(3, 1,constrained_layout=True)
## Start mit aliasing
axarr[0].set_title('Aliasing')
axarr[0].scatter(t_sample, y_aliasing,zorder=3)
axarr[0].plot(t_fine, fun(t_fine),color=colors[4],linewidth=2,linestyle=':')
axarr[0].plot(t_fine, -fun_alias(t_fine),color=colors[1],linewidth=2)
axarr[0].set_xlabel('t')
axarr[0].set_ylabel('Amplitude')

## Now clipping is demonstrated:
y_clipping = fun(t_fine)
threshold = 0.7
y_clipping[y_clipping>threshold] = threshold
y_clipping[y_clipping<-threshold] = -threshold
axarr[1].set_title('Clipping')
axarr[1].plot(t_fine, y_clipping,zorder=3)
axarr[1].plot(t_fine, fun(t_fine),color=colors[4],linewidth=2,linestyle=':')
axarr[1].set_xlabel('t')
axarr[1].set_ylabel('Amplitude')

axarr[2].set_title('Quantization')
axarr[2].plot(t_fine, np.round(fun(t_fine),decimals=1),zorder=3)

axarr[2].plot(t_fine, fun(t_fine),color=colors[4],linewidth=2,linestyle=':')
axarr[2].set_xlabel('t')
axarr[2].set_ylabel('Amplitude')
```

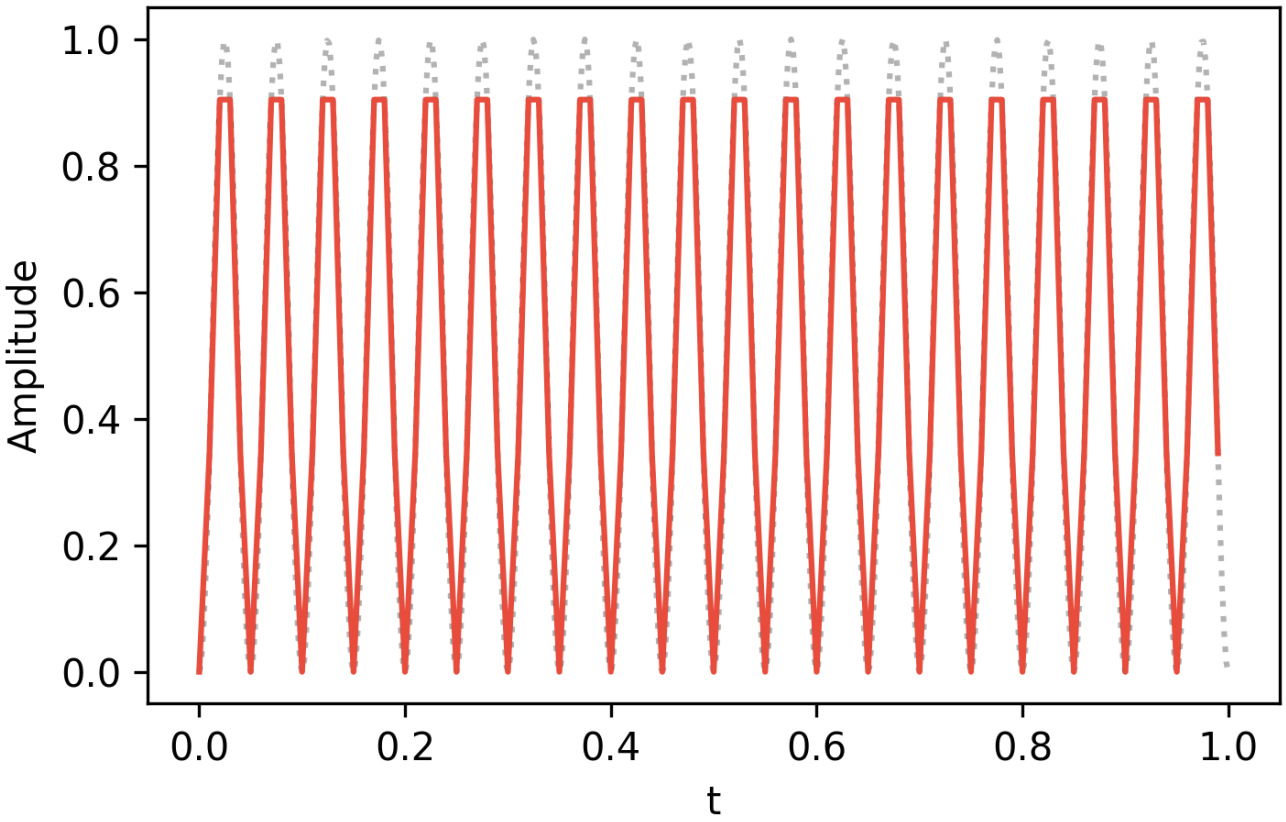
```
savefig("2_d")
plt.show()
```



Problem 4: Fourier Series 1 (15 points)

See Sampling in `data_cquisition_V3.mlx`

(a) Plot a **wave1** sampled with a 100 Hz sampling rate. The **wave1** is



where $f_0 = 10$. Please connect sampled points and plot only ten cycles of the wave.

(b) Derive a Fourier series (general form) of **wave1**. You should find analytic equations for coefficients of a_0, a_n, and b_n.

- The Fourier transforms states, that the function y(t) as states above, called wave1 can be written as $y(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{2\pi n t}{T_p}\right) + b_n \sin\left(\frac{2\pi n t}{T_p}\right)$
- It holds $\sin^2(2\pi f_0 t) = \sin^2(2\pi 10 \cdot t) = \frac{1}{2} - \frac{1}{2}\cos(2\pi \cdot 2 \cdot 10 \cdot t) = \frac{a_0}{2} + a_1 \cos(2\pi \cdot 1 \cdot 10 \cdot t) + a_2 \cos(2\pi \cdot 2 \cdot 10 \cdot t)$ with $a_0 = 1, a_1 = 0, a_2 = -\frac{1}{2}$. For the other coefficients hold $b_n = 0, \forall n \in \mathbb{N}$ and $a_n = 0, \text{for } \; n > 2$.

(c) Derive a Fourier series (complex form) of **wave1**. You should find an analytic equations for a coefficient of c_n.

- The complex coefficients can be derived from the real coefficients.
- The coefficients are derived using
$$c_0 = a_0/2$$
$$c_n = \frac{1}{2}(a_n - ib_n) \; (\text{ for } n \geq 1)$$
$$c_{-n} = \frac{1}{2}(a_n + ib_n) \; (\text{ for } n \geq 1)$$
- This gives: $c_0 = \frac{1}{2} + 0.0i$ and $c_1 = 0 + 0.0i$ and $c_2 = -\frac{1}{4} + 0.0i$.

(d) Derive a Fourier series (general form) of **wave2**:

You should find analytic equations for coefficients of a_0, a_n, and b_n.

- $a_0 = 11, a_1 = 0, a_2 = -0.5$ and all other variables for all other indices are zero.

(e) Please compare the results of (b) and (d) and explain their difference.

- The offset of the curve is only determined by the zeroth coefficient (c_0 or a_0). There is no b_0.

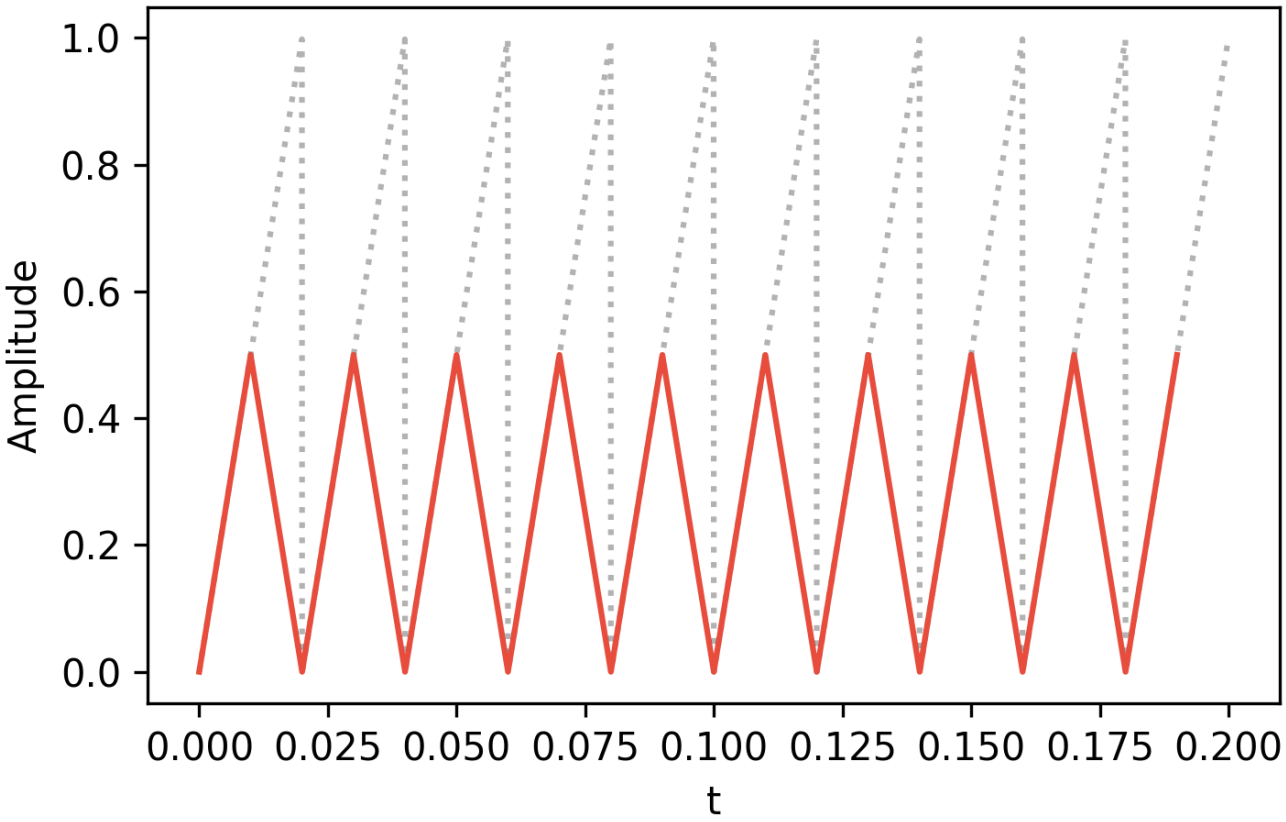
Problem 5: Fourier Series 2 (15 points)

Sawtooth wave: https://en.wikipedia.org/wiki/Sawtooth_wave

See Sampling in `data_cquisition_V3.mlx`

(a) Plot a 50Hz sawtooth wave sampled with a 100 Hz sampling rate. Please connect sampled points and plot only ten cycles of the wave.

```
##### plt.close("all")

## Variables setup
f0 = 50
T = 1.0/f0 # Periode time

def f(t):
    return t-np.floor(t)

f_sample = 100
n_cycles = 10.0 # Number of cycles to be plottet

t_start = 0.0
t_end = n_cycles*T
t_sample =  np.arange(t_start,t_end,1.0/f_sample)
t_fine = np.linspace(t_start,t_end,5000, endpoint=False)


y_sample = f(f0*t_sample)
y_fine = f(f0*t_fine)

fig, ax = newfig(0.9)
ax.plot(t_fine,y_fine,linestyle = ':',color = 'black',alpha = 0.3);
ax.plot(t_sample,y_sample,color = colors[1]);
# legend = ax.legend(loc="upper right", frameon=True, facecolor='white', shadow=False,
fancybox=False);
ax.set_xlabel(r't');
ax.set_ylabel("Amplitude");
# ax.set_aspect('equal')
# ax.grid(True, which='both');
# ax.ticklabel_format(style='sci', scilimits=(-3, 4), axis='both');
savefig("5_a")
plt.show()
```



(b) Derive a Fourier series (general form) for a sawtooth wave:

Please check the wikipedia link. You should find analytic equations for coefficients of a_0, a_n, and b_n.

- We derive $a_0 = 2/T_p \int_0^{T_p} T_p t \mathrm{d}t = T_p^2 = 1$, for the general case with T_p = 1.
- For $n > 0$: $a_n = 2/T_p \int_0^{T_p} t cos(2\pi f n t) \overbrace{=}^{\text{partial integration}} 0.$, as the sinus function is 2π periodic.
- $b_n = 2f \int_0^{T_p} t \sin(2\pi f m t) = 2f \left(-[\frac{cos(2\pi f m t)}{2\pi f m}]_0^{T_p} - \int_0^{T_P} sin(2\pi f m t) \mathrm{d}t \right) = -\frac{1}{\pi n}, n \neq 0...$

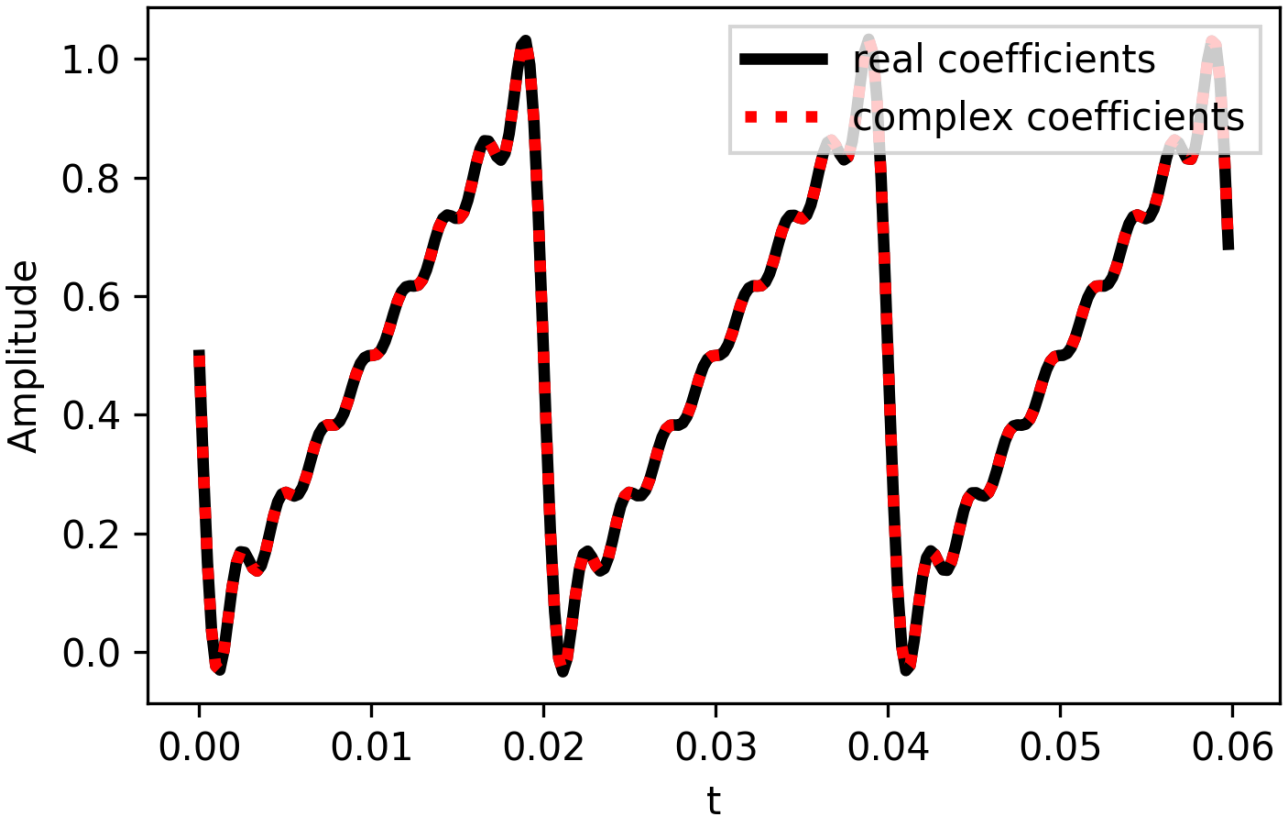
  The full series is then:
- $x(t) = 0.5 - \frac{1}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin(n\pi f t)$

(c) Derive a Fourier series (complex form) for the same sawtooth wave.You should find an analytic equation for a coefficient of c_n.

- The complex coefficients are determined by
- $c_n = \frac{1}{T_p} \int_0^{T_p} \frac{1t}{T_p} e^{-in\omega_0 t} dt = \frac{1}{{T_p}^2} \int_0^{T_p} t e^{-in\omega_0 t} dt = \frac{T_p e^{\mathrm{in}\omega_0 T_p}}{(-\mathrm{in}w_0)} - \frac{1}{(\mathrm{in}\omega_0)^2} \left[e^{-in\omega_0 t} \right]_0^{T_p} = \frac{i}{2\pi n}.$
- It was used that $\omega_0 = 2\pi f$ and $e^{in2\pi} = 1$
- It holds, that $c_0 = a_0/2 = 0.5$.
- $x(t) = \sum_{n=-\infty}^{\infty} \frac{i}{2\pi n} \exp(i2\pi n t).$

(d) Write a code to create and plot approximated sawtooth waves (# of coeffcients (n) = 8) using the derived Fourier series in the general and complex forms. You should compare the waves from the general and complex forms.

```
f = 50.0 # Frequency

# Coefficients real
def a_0():
    return 1.0
def a_n():
    return 0.0
def b_n(i):
    return -1/np.pi/i

# (Real) fourier series
def x_real(t,n_max):
    return a_0()/2.0 + sum([b_n(i)*np.sin(2*i*np.pi*f*t) for i in range(1,n_max+1)])

# Complex fourier series:
def c_n(i):
    if i != 0:
        return 1.0j/(2*np.pi*i)
    else: # i==0
        return 0.5

def x_complex(t,n_max):
    return sum([c_n(i)*np.exp(1j*2*np.pi*f*i*t) for i in range(-n_max,n_max+1)])


t = np.linspace(0,10*1.0/f, 500, endpoint=False)
y_real = x_real(t,8)
y_complex = x_complex(t,8)

fig, ax = newfig(0.9)
ax.plot(t,y_real,linestyle = '-',color = 'black',alpha = 0.4,linewidth = 2);
ax.plot(t,y_complex,linestyle = ':',color = 'red',alpha = 0.4,linewidth = 2);

# legend = ax.legend(loc="upper right", frameon=True, facecolor='white', shadow=False,
fancybox=False);
ax.set_xlabel(r't');
ax.set_ylabel("Amplitude");
# ax.set_aspect('equal')
# ax.grid(True, which='both');
# ax.ticklabel_format(style='sci', scilimits=(-3, 4), axis='both');
savefig("5_d")
plt.show()

print("The pointwise difference measured in the 2 norm between both vectors is
{:2f}".format(np.linalg.norm(y_real-y_complex)))
```

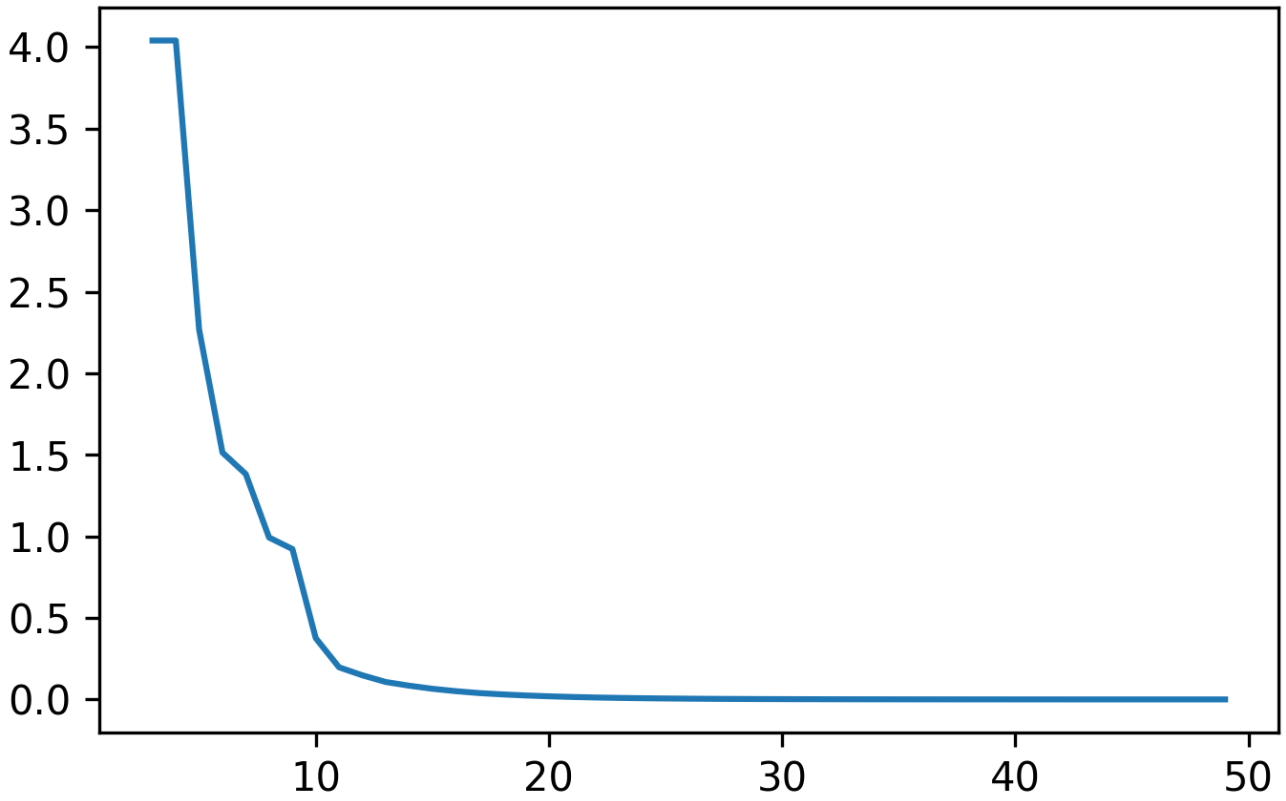


(e) Write a code to find numerical Fourier coefficients in the general and complex forms and compare them with the analytic Fourier coefficients found in (b) and (c).

First the complex coefficients are compared. Instead of having a high order integration scheme, the convergence of the simple rule is shown below, which shows, that indeed the numeric and the analytic coefficients converge pointwise and the difference depends mainly on the accuracy of the quadrature:

```
def c_n_complex(t,i):
    return t*np.exp(-1j*2*np.pi*1*t*i)

norm = []

for steps in list(np.logspace(0,5)):
    N = 8
    t,dt = np.linspace(0,1,np.int(steps),retstep=True)
    c_n_numeric = arr([sum(c_n_complex(t,i)*dt) for i in range(-N,N+1)])
    c_n_analytic = arr([c_n(i) for i in range(-N,N+1)])

#     np.set_printoptions(suppress=True)

#     print("Numeric complex")
#     print((c_n_numeric.flatten()))
#     print("Analytic complex")
#     print((c_n_analytic.flatten()))

    norm.append(np.linalg.norm(c_n_numeric.flatten() - c_n_analytic.flatten()))

fig, ax = newfig(0.9)
```

```
ax.plot(norm)
plt.show()
savefig('5_e')
```



Above figure shows the convergence of the euclidian error of the comlex values using rectangular quadrature. The x axis are the number of integration steps, a higher number means finer riemann of the ordinary riemann sum.

For the real part, the follwoing code shows that both methods yield comparable results:

```
print("Numeric real")
def f(t):
    return t

a0 = 2.0*sp.integrate.quad(f,0,1.0)[0]


def f(t,i):
    return 2*t*np.cos(2*np.pi*i*t*1)
an = arr([sp.integrate.quad(f,0,1.0,args=i)[0] for i in range(1,N+1)])


def f(t,i):
    return 2*t*np.sin(2*np.pi*i*t*1)
bn = arr([sp.integrate.quad(f,0,1.0,args=i)[0] for i in range(1,N+1)])
print(a0)
print(an)
print(bn)
```

```
print("Analytic real")
print(a_0())
print(arr([a_n() for i in range(1,N+1)]).flatten())
print(arr([b_n(i) for i in range(1,N+1)]).flatten())
```

Which outputs:

```
Numeric real
1.0
[ 0.  0. -0. -0. -0. -0. -0. -0.]
[-0.31830989 -0.15915494 -0.1061033  -0.07957747 -0.06366198 -0.05305165
 -0.04547284 -0.03978874]

Analytic real
1.0
[0. 0. 0. 0. 0. 0. 0. 0.]
[-0.31830989 -0.15915494 -0.1061033  -0.07957747 -0.06366198 -0.05305165
 -0.04547284 -0.03978874]
```

Problem 6: Fourier Transformation 1 (15 points)

Compute the Fourier transformation (integral) of the following functions and show the derivation process in detail.

(a) cosine wave

$$x(t) = y$$

$$X(f) = \int_{-\infty}^{\infty} \cos(2\pi p_0 t) \cdot \exp(-i2\pi p_0 t) \mathrm{d}t.$$

- $$X(f) = \int_{-\infty}^{\infty} \frac{1}{2}(\exp(i2\pi p_0 t) + \exp(-i2\pi p_0 t)) \cdot \exp(-i2\pi p_0 t) \mathrm{d}t$$

$$\text{See Table on slide 20 Signal Processing lecture}$$

$$X(f) = \frac{1}{2}(\delta(f - p_0) + \delta(f + p_0))$$

Assuming, that using the slides isn't enough for "show the derivation process in detail", in the following the integration is elaborated:

$$X(f) = 0.5 \int_{-\infty}^{\infty} \exp(i2\pi p0t - i2\pi ft) + \exp(i2\pi p0t + i2\pi ft)\mathrm{d}t$$

$$X(f) = 0.5 \int_{-\infty}^{\infty} \exp(i2\pi t(p0 - f)) + \exp(i2\pi t(p0 + f))\mathrm{d}t$$

$$X(f) = \frac{1}{2}\left([\frac{1}{2\pi i(p0 - f)}\exp(t2\pi i(p0 - f))]_{-\infty}^{\infty} + [\frac{1}{2\pi i(p0 + f)}\exp(t2\pi i(p0 + f))]_{-\infty}^{\infty} \right) \square$$

(b) cosine wave + dc (direct current) wave

Because of the linearity of the integral and the assumption, that the limit of the integral exists, we can just transform the "d" and add it to solution b):

- $$X(f) = \frac{1}{2}(\delta(f - p_0) + \delta(f + p_0)) + d\delta(f)$$

(c) Gaussian function

First: x(t) := y. The corresponding fourier transform is denoted as X(f).

$$X(f) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} \mathrm{e}^{-\frac{x^2}{2\sigma^2}} \mathrm{e}^{-2i\pi fx}\,\mathrm{d}x$$

$$X(f) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} \mathrm{e}^{-\frac{1}{2\sigma^2}\left(x^2 + 4i\pi\sigma^2 xf\right)}\,\mathrm{d}x$$

$$\text{Adding a } 0\text{, gives}$$

- $$X(f) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} \mathrm{e}^{-\frac{1}{2\sigma^2}\left(x^2 - 2\left(2i\pi\sigma^2 f\right)x + \left(2i\pi\sigma^2 f\right)^2 - \left(2i\pi\sigma^2 f\right)^2\right)}\,\mathrm{d}x$$

$$X(f) = \frac{1}{\sqrt{2\pi\sigma^2}} \left(\int_{-\infty}^{\infty} \mathrm{e}^{-\frac{1}{2\sigma^2}\left(x - 2i\pi\sigma^2 f\right)^2} dx \right) e^{\frac{1}{2\sigma^2}\left(2i\pi\sigma^2 f\right)^2}$$

$$X(f) = \frac{1}{\sqrt{2\pi\sigma^2}} \left(\sqrt{2\pi\sigma^2} \right) \mathrm{e}^{-2\pi^2\sigma^2 f^2}$$

Which is a nice as it can easily be seen, that the fourier transform of the gaussian, is again slightly changed gaussian.

Problem 7: Fourier Transformation 2 (15 points)

(a) Compute the Fourier transformation (integral) of the following function

- Technically the fourier transformation can be solved as follows, assuming again, that y =:x(t)

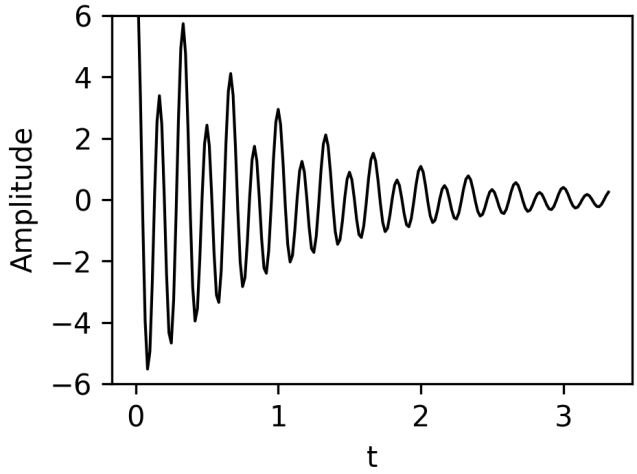
- $$X(f) = \int_0^\infty \exp(-at) b \cos(2\pi f_1 t) \cdot \exp(-i 2\pi f t) \mathrm{d}t + \int_{-\infty}^0 \exp(at) b \cos(2\pi f_1 t) \cdot \exp(-i 2\pi f t) \mathrm{d}t$$
  $$+ \int_0^\infty \exp(-at) c \cdot \cos(2\pi f_2 t) \cdot \exp(-i 2\pi f t) \mathrm{d}t + \int_{-\infty}^0 \exp(at) c \cdot \cos(2\pi f_2 t) \cdot \exp(-i 2\pi f t) \mathrm{d}t$$
- From the lecture slides we know that we get
  $X(f) = \frac{ab}{a^2+4\pi^2(f+f_1)^2} + \frac{ab}{a^2+4.0\pi^2(f-f_1)^2} + \frac{ac}{a^2+4\pi^2(f+f_2)^2} + \frac{ac}{a^2+4.0\pi^2(f-f_2)^2}$
- Another approach would have been to use the convolution theorem:
- $$\mathcal{F}\{f \cdot g\} = \mathcal{F}\{f\} * \mathcal{F}\{g\}.$$
- Note that here, the $*$ symbol denotes a convolution.

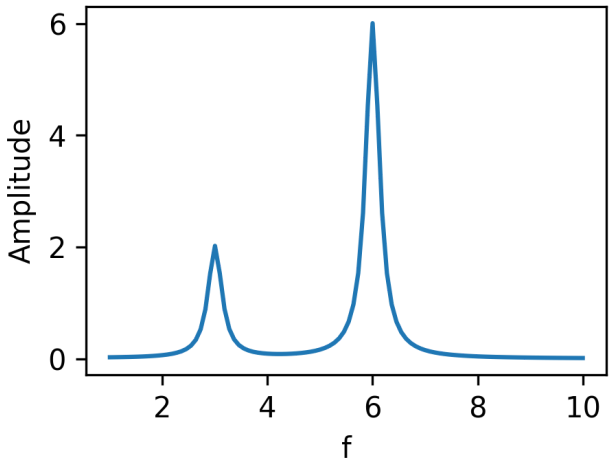
- This gives then with the given list of fourier transforms:
- $$X(f) = \left(\frac{2\alpha}{\alpha^2 + 4\pi^2 f_1^2}\right) * \left(\frac{2\alpha}{\alpha^2 + 4\pi^2 f_1^2}\right) * (b \cdot \delta(f_1))$$
  $$+ \left(\frac{2\alpha}{\alpha^2 + 4\pi^2 f_2^2}\right) * \left(\frac{2\alpha}{\alpha^2 + 4\pi^2 f_2^2}\right) * (c \cdot \delta(f_2))$$
- This equation is not used further, but it gives an idea, how the upper solution can be derived by that.

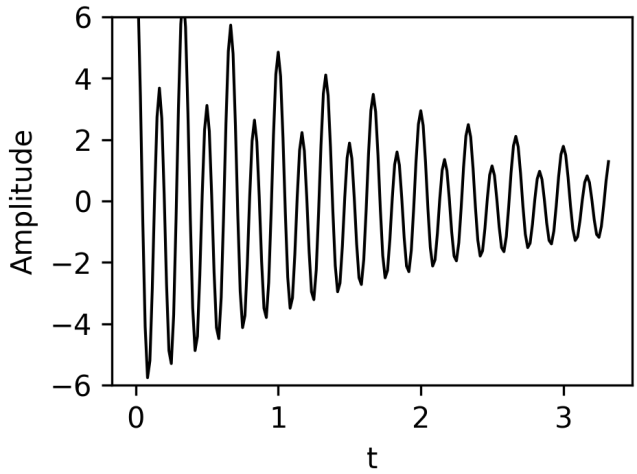
(b) Plot y in time domain and frequency domain, where a = 1, b = 2, c= 6, f_1 = 3, and f_2 = 6

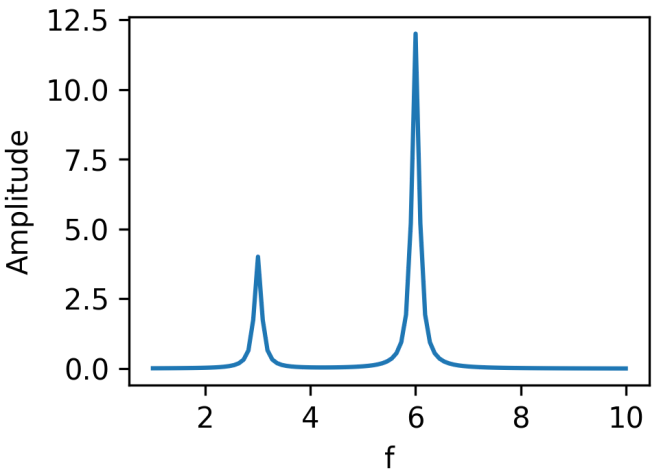
(c) Plot y in time domain and frequency domain, where a = 0.5, b = 2, c= 6, f_1 = 3, and f_2 = 6

Both tasks are shown below in correct order.


Amplitude
t
6
4
2
0
−2
−4
−6
0
1
2
3


Amplitude
f
6
4
2
0
2
4
6
8
10


Amplitude
t
6
4
2
0
−2
−4
−6
0
1
2
3


Amplitude
f
12.5
10.0
7.5
5.0
2.5
0.0
2
4
6
8
10