In [13]:
```
#importing necessary packages
from sympy.interactive import printing
printing.init_printing(use_latex=True)
from sympy import *
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
```

Q1 (a)

Here we find the general solution for for second order linear differencial equation that is homogeneous with constant coefficents

in class we solved y'' +y=0 and we realized that y can take two forms either exponetial 'e^kx'or sin

This is because when we plag either to the equation and solve then we end up with 0 as expected

so we expect our solution-general solution- to be in the form of:

y=Asin()+Bsin() where A and B are coeffcients or

y=Ae^ωk+Be^ωk

Either of the above forms is correct

In [14]:
```
#espressing ω,t as symbols that we will use later where omega where ω=sqrt(k/m)
ω,t=sp.symbols(' ω t')
#expressing x as a fuction
x=sp.Function('x')
```

In [15]:
```
#here our fuction is dependant on t
diff_2=Eq(sp.Derivative(x(t), t, 2) + ω**2 * x(t),0)
#displaying our fuction
display(diff_2)
```

$$\omega^2 x(t) + \frac{d^2}{dt^2} x(t) = 0$$

The General solution is as below where **C1** and **C2** are coeffcients

In [16]:
```
# here we dislove the fuction which means we differenciate with respect to t , solving the equation gives us
#C1 and C2 on the display are place holders
ans= sp.dsolve(sp.Derivative(x(t), t, 2) + ω**2 * x(t))
print("\n General solution\n" )
display(ans)
```

General solution

$$x(t) = C_1 e^{-it\omega} + C_2 e^{it\omega}$$

--

We solve for our two unkowns c1 and c2
We also substitute relevant variables with intial condition values where
ω0 = 2,
x(0) =1,
x'(0) = 0

In [17]:
```
initial_con = [sp.Eq(ans.args[1].subs(t, 0), 1), sp.Eq(ans.args[1].diff(t).subs(t, 0), 0)]
initial_con
```

Out[17]: $[C_1 + C_2 = 1, \ -iC_1\omega + iC_2\omega = 0]$

In [18]:
```
initial_con_ans_1=sp.solve(initial_con)
initial_con_ans_1
```

Out[18]: $\left[\left\{C_1 : \frac{1}{2}, C_2 : \frac{1}{2}\right\}\right]$

The below equation is as a result of solving for the cieffcients and substitutin with there values:

we then simplified the equation to obtain a simpler version of x(t)

The complex exponentials are not a surprise as we expected an oscillatory fuction from the natural behaviour of springs # **statistical Mechanic NS162**

In [19]:
```
initial_con_full_ans = ans.subs(initial_con_ans_1[0])
initial_con_full_ans
```

Out[19]: $x(t) = \frac{e^{it\omega}}{2} + \frac{e^{-it\omega}}{2}$

We substitute the values

ω0 = 2,

x(0) =1,

x'(0) = 0

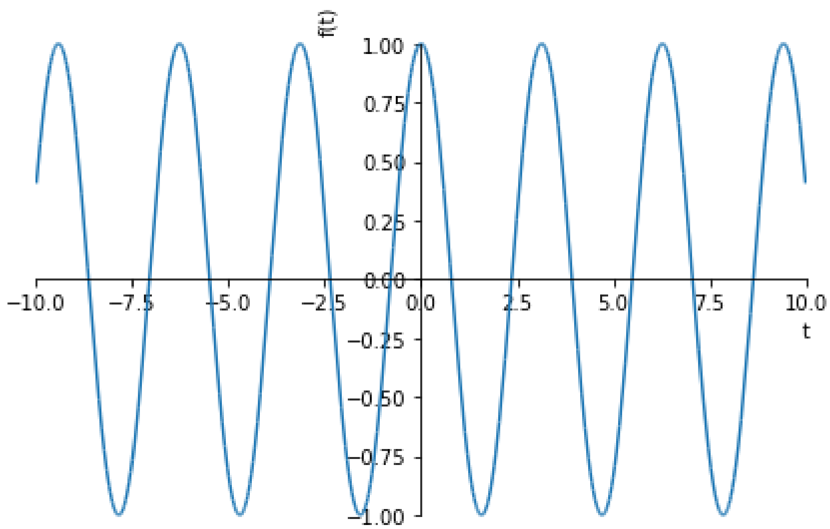
Given in the problem and simplify to obtaing the below fuction

In [20]:
```
ẋ, x_0=sp.symbols(' ẋ  x_0')
given_scenario = sp.simplify(initial_con_full_ans.subs({x_0:1, ẋ:0, ω:2}))
given_scenario
```

Out[20]: $x(t) = \cos(2t)$

We then plot the simplified **x(t)** over range

In [21]:
```
sp.plot(given_scenario.rhs)
```



Out[21]: <sympy.plotting.plot.Plot at 0x7fdd0eee2cd0>

The plot above follows the naturla behaviour of harmonic oscillators

---

Q1 (b)

Here we implement ous solution using the **complexification technique**

That is : u=(x+yi)^n where yi is the imaginary part and x is the real part

In [22]:
```
ω_1=sp.symbols('ω_1')
```

In [23]:

```
diff_1=Eq(sp.Derivative(x(t), t, 2) + ω**2 * x(t),cos(ω_1*t))
display(diff_1)
```

$$\omega^2 x(t) + \frac{d^2}{dt^2} x(t) = \cos(t\omega_1)$$

We already know the genral solution from part a here we just need to include the real part

In [24]:

```
complification=dsolve(Eq(sp.Derivative(x(t), t, 2) + ω**2 * x(t),cos(ω_1*t)))
print("\n General solution\n" )
display(complification)
```

General solution

$$x(t) = C_1 e^{-it\omega} + C_2 e^{it\omega} + \frac{\cos(t\omega_1)}{\omega^2 - \omega_1^2}$$

We solve for our two unkowns c1 and c2
We also substitute relevant variables with intial condition values where
ω0 = 2,
x(0) =1,
x'(0) = 0

In [25]:

```
complification_sol= [sp.Eq(complification.args[1].subs(t, 0), 1), sp.Eq(complification.args[1].diff(t).subs(t
display(complification_sol)
```

$$\left[C_1 + C_2 + \frac{1}{\omega^2 - \omega_1^2} = 1, \quad -iC_1\omega + iC_2\omega = 0 \right]$$

We substitute the values

ω0 = 2,

x(0) =1,

x'(0) = 0

Given in the problem and simplify to obtaing the below fuction

In [26]:

```
complification_sol_1=sp.solve(complification_sol)
display(complification_sol_1)
```

$$\left[\left\{ C_1 : \frac{\omega^2 - \omega_1^2 - 1}{2(\omega - \omega_1)(\omega + \omega_1)}, \; C_2 : \frac{\omega^2 - \omega_1^2 - 1}{2(\omega - \omega_1)(\omega + \omega_1)} \right\} \right]$$

The below equation is as a result of solving for the cieffcients and substitutin with there values:

we then simplified the equation to obtain a simpler version of x(t)

The complex exponentials are not a surprise as we expected an oscillatory fuction from the natural behaviour of springs # **statistical Mechanic NS162**

In [27]:
```
complification_sol_full = ans.subs(complification_sol_1[0])
display(complification_sol_full )
```

$$x(t) = \frac{\left(\omega^2 - \omega_1^2 - 1\right) e^{it\omega}}{2\left(\omega - \omega_1\right)\left(\omega + \omega_1\right)} + \frac{\left(\omega^2 - \omega_1^2 - 1\right) e^{-it\omega}}{2\left(\omega - \omega_1\right)\left(\omega + \omega_1\right)}$$

ω1=! ω0 because the fuction will be undefined, the denominator will be 0 and this we can not solve the problem

---

We substitute the values

ω0 = 2,

x(0) =1,

x'(0) = 0

Given in the problem and simplify to obtaing the below fuction

In [28]:
```
given_scenario_2 = sp.simplify(complification_sol_full.subs({x_0:1, ẋ:0, ω:2,ω_1:3}))
given_scenario_2
```

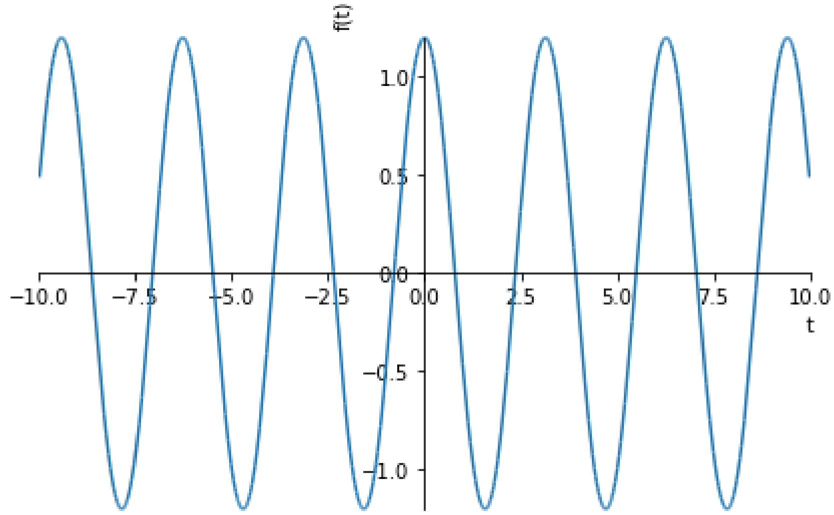
Out[28]:

$$x(t) = \frac{6\cos(2t)}{5}$$

Q1 (c)

We then plot the simplified **x(t)** over range

In [29]:
```
sp.plot(given_scenario_2.rhs)
```



Out[29]: <sympy.plotting.plot.Plot at 0x7fdd0fa10bd0>

The plot above follows the naturla behaviour of harmonic oscillators

---

We repeat Q1 b and c using sin instead, all fuctions are the same just replace cos with sin, and the expanantions are same too.

In [30]:
```
diff_1_sin=Eq(sp.Derivative(x(t), t, 2) + ω**2 * x(t),sin(ω_1*t))
display(diff_1_sin)
```

$$\omega^2 x(t) + \frac{d^2}{dt^2} x(t) = \sin(t\omega_1)$$

In [31]:
```
complification_sin=dsolve(Eq(sp.Derivative(x(t), t, 2) + ω**2 * x(t),sin(ω_1*t)))
display(complification_sin)
```

$$x(t) = C_1 e^{-it\omega} + C_2 e^{it\omega} + \frac{\sin(t\omega_1)}{\omega^2 - \omega_1^2}$$

In [32]:
```
complification_sol_sin= [sp.Eq(complification_sin.args[1].subs(t, 0), 1), sp.Eq(complification_sin.args[1].
display(complification_sol_sin)
```

$$\left[C_1 + C_2 = 1, \ -iC_1\omega + iC_2\omega + \frac{\omega_1}{\omega^2 - \omega_1^2} = 0 \right]$$

In [33]:
```
complification_sol_1_sin=sp.solve(complification_sol_sin)
display(complification_sol_1_sin)
```

$$\left[\left\{ C_1 : \frac{\omega^3 - \omega\omega_1^2 - i\omega_1}{2\omega\left(\omega^2 - \omega_1^2\right)}, \ C_2 : \frac{\omega^3 - \omega\omega_1^2 + i\omega_1}{2\omega\left(\omega^2 - \omega_1^2\right)} \right\} \right]$$

In [34]:
```
complification_sol_full_sin = ans.subs(complification_sol_1_sin[0])
display(complification_sol_full_sin )
```

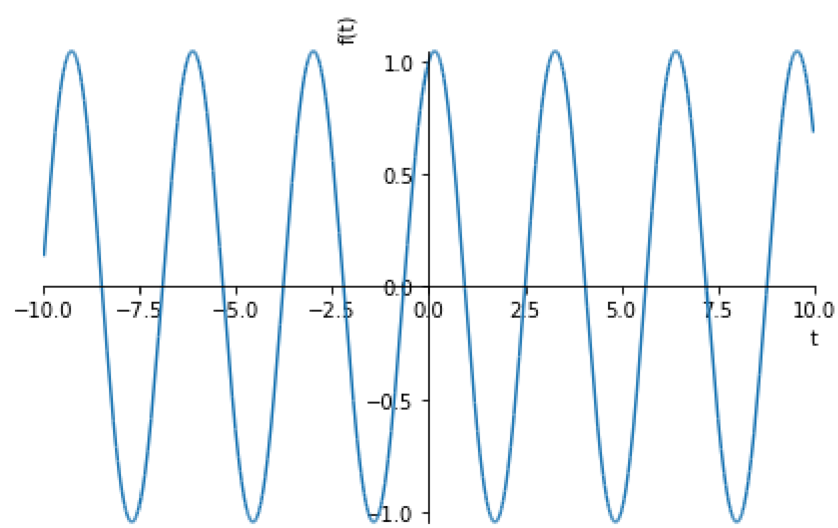
$$x(t) = \frac{\left(\omega^3 - \omega\omega_1^2 - i\omega_1\right) e^{-it\omega}}{2\omega\left(\omega^2 - \omega_1^2\right)} + \frac{\left(\omega^3 - \omega\omega_1^2 + i\omega_1\right) e^{it\omega}}{2\omega\left(\omega^2 - \omega_1^2\right)}$$

In [35]:
```
given_scenario_2_sin = sp.simplify(complification_sol_full_sin.subs({x_0:1, ẋ:0, ω:2,ω_1:3}))
given_scenario_2_sin
```

Out[35]:

$$x(t) = \frac{\left((10 - 3i)\, e^{4it} + 10 + 3i\right) e^{-2it}}{20}$$

In [36]:
```
sin_plot=sp.plot(given_scenario_2_sin.rhs)
sin_plot
```



Out[36]: <sympy.plotting.plot.Plot at 0x7fdd0feb3090>

Q1 (d)

we set x=f(t)e^(iωt): we do this because solving it directly leads to undefined solution as we get 0 at the denominator

x=f(t)e^(iωt)

x'=f'(t)e^(iωt)+iωtf'(t)e^(iωt)

x''=f''(t)e^(iωt)+iωtf'(t)e^(iωt)+iωtf'(t)e^(iωt)-ω^2 * f(t)e^(iωt)

x''+ω^2x=*f"(t)e^(iωt)+2iωf'(t)e^(iωt)-ω^2 * f(t)e^(iωt)+ω^2 * f(t)e^(iωt)*

The last terms disapear

f"(t)e^(iωt)+2iω f(t)e^(iωt)

dislvoing the fuction further yields e^iω

e^iω=cosω+isinω

general solution

e^iω or cosω+isinω

In [ ]:

```
import cmath
import math
from math import e
i =1j
```

In [ ]:

```
#set x=f(t)e^(iωt)
x=sp.Function('f')(t)*e**(i*ω*t)
display(x)
```

In [39]:

```
fuc=Eq(sp.Derivative(x, t, 2) + ω**2 * x,sin(ω*t))
display(fuc)
```

$$2.71828182845905^{1.0it\omega}\omega^2 f(t) + \frac{\partial^2}{\partial t^2}2.71828182845905^{1.0it\omega} f(t) = \sin(t\omega)$$

In [41]:

```
complification_ii=dsolve(Eq(sp.Derivative(x, t, 2) + ω**2 * x,sin(ω*t)))
display(complification_ii)
```

Q1 (e)

The general solution differ in the coefficients as well as the placement of the imaginary part in the first one we had general solution in the form Ae^ωk+Be^ωk while in the d part we have cosω+isinω:

Q1 (f) Vedio explanation based on Numeroical analysis knowledge

We know that ω=sqrt(k/m) where k is the spring constant and m is the mass. If k is big it means the spring is strong that the oscillation will be small but faster , while if m is big the ossicilation would be slower due to change of intertia. Applying the same knowledge on the brige from the 0.21 second the oscillation are slow but big which means the mass is greater than what the spring can hold thus may tend towards breaking point . Some sections of the brige are vibrating faster but the amplitude is smaller which means the bridge has kess weight there that is less than the spring constant. The wind acts as an external agents that acts as a catalyst, here it increased m thus the bridge reached its breaking point. Notice that it breaks fast where the vibration were slow but big (m>>k).. This show how omega which is a fuction of sqrt(k/m) contributes in the harmonic oscilation

Question 2

I recorded two audio from different location

and used https://www.online-convert.com/result#j=5eaeec88-dddd-426d-a83b-4012d440a97c (https://www.online-convert.com/result#j=5eaeec88-dddd-426d-a83b-4012d440a97c)

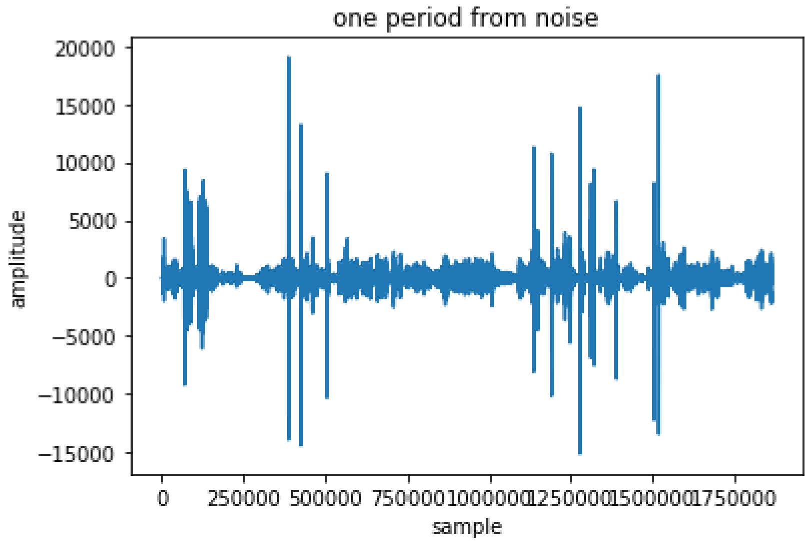
to convert the audi into wave. I then read it throw wave.open from scipy

In [50]:
```
from scipy.io import wavfile
from scipy import *
import wave
import matplotlib.pyplot as plt
```

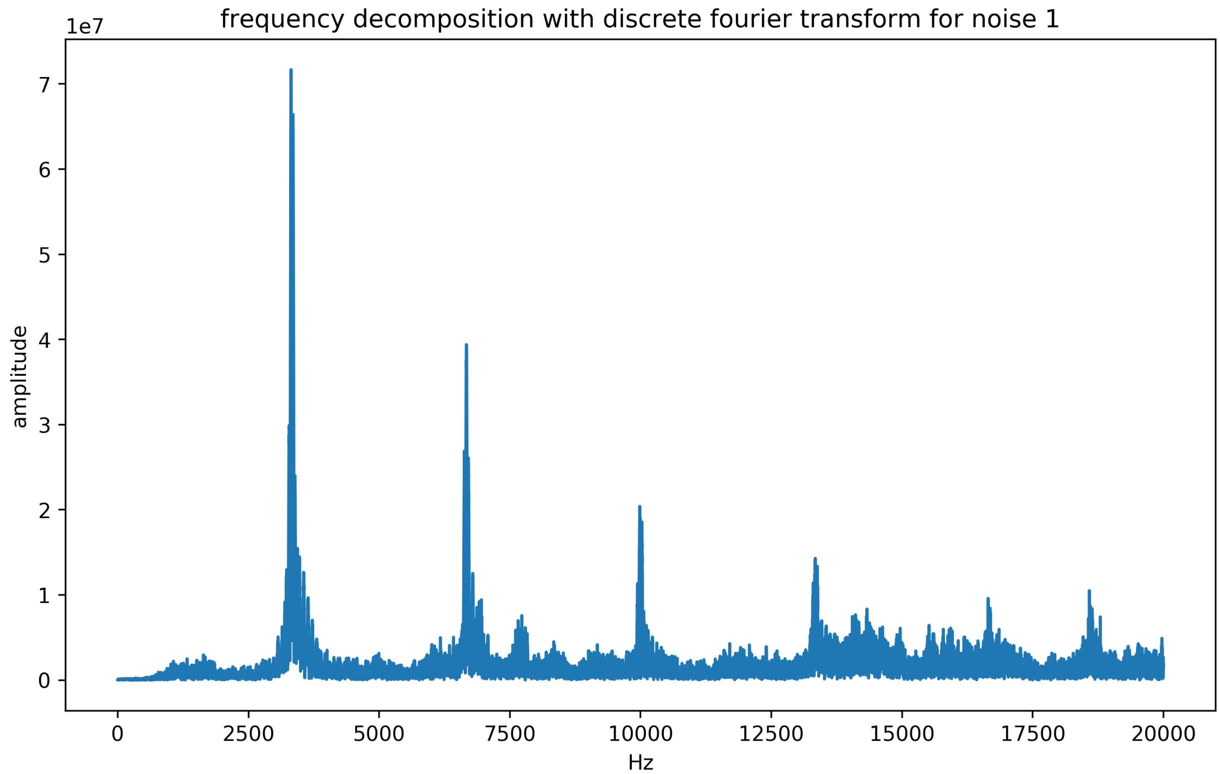
In [51]:
```
noise = wave.open("noise1.wav",'r')
noise_rate=np.inf
```

In [52]:
```
sig = np.frombuffer(noise.readframes(noise_rate), dtype=np.int16)
plt.plot(sig)
plt.title("one period from noise")
plt.xlabel("sample")
plt.ylabel("amplitude")
```

Out[52]: Text(0, 0.5, 'amplitude')



In [53]:
```
trans_1 = fft(sig)
plt.figure(figsize=(10, 6), dpi=330)
plt.plot(abs(trans_1)[:20000])
plt.ylabel("amplitude")
plt.xlabel("Hz")
plt.title("frequency decomposition with discrete fourier transform for noise 1")
plt.show()
```



In [72]:
```
peak_1=2500+2500/2
peak_1
```

Out[72]: 3750.0

The dominant peak is at around 3750.0HZ

This is the highest frequency after we decompose the wave into finite number of sinusoidal, From the graph we other smaller peaks that reduce over time and flattens towards the 20000HZ

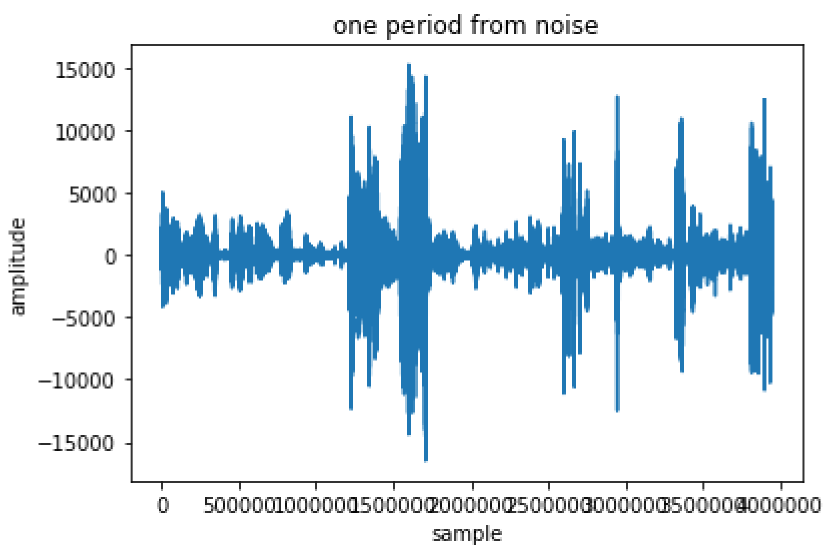
In [54]:

```
noise_2 = wave.open("noise2.wav",'r')
noise_rate=np.inf
```

In [55]:

```
sig_2 = np.frombuffer(noise_2.readframes(noise_rate), dtype=np.int16)
plt.plot(sig_2)
plt.title("one period from noise")
plt.xlabel("sample")
plt.ylabel("amplitude")
```

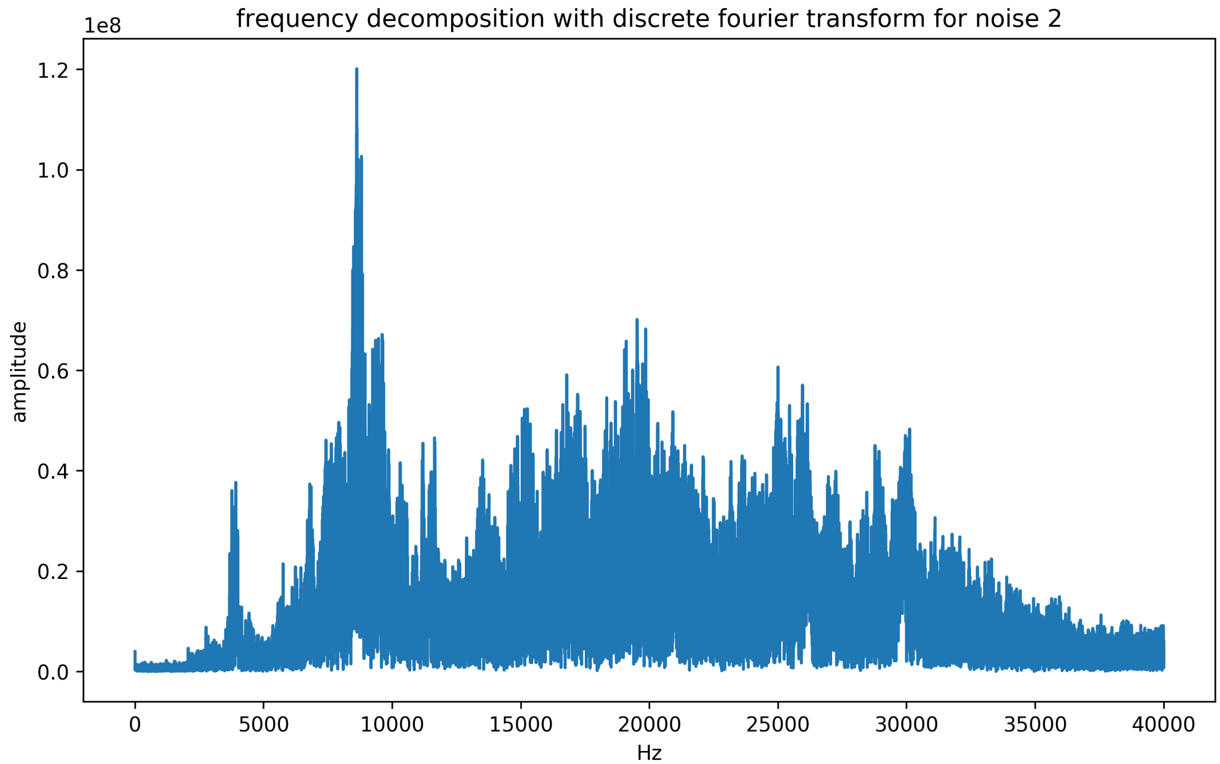
Out[55]: Text(0, 0.5, 'amplitude')



In [73]:

```
from scipy.signal import find_peaks
trans_2 = fft(sig_2)
plt.figure(figsize=(10, 6), dpi=330)
peaks, _=  find_peaks(trans_2, height=0)
plt.plot(abs(trans_2)[:40000])
plt.ylabel("amplitude")
plt.xlabel("Hz")
plt.title("frequency decomposition with discrete fourier transform for noise 2")
plt.show()
```

```
/opt/anaconda3/lib/python3.7/site-packages/scipy/signal/_peak_finding.py:264: ComplexWarning: Casting co
mplex values to real discards the imaginary part
  value = np.asarray(value, order='C', dtype=np.float64)
```



In [71]:
```
peak=7500+2500/2
peak
```

Out[71]: 8750.0

The dominat frequncy is at around 8750.0HZ
From there the frequency hight reduces over time and decreases more between 35000 and 40000. This means that if I was studying and got rid of 8750.0 then noise would reducee significantly . It is at the dorminat frequency where the highest energy is found E = hv = hc/λ, where E = energy, h = Planck's constant, v = frequency, c = the speed of light, and λ = wavelength. # **statistical Mechanic NS162 .** Removing the highest frequency means that noise measured in decibels would be lower

From the second gragh I would need earbud that cancels frequncy btn 2500Hz to 35000Hz as its the region with more fluctuation and high average enegy thus noise

From the 1st gragh the noise cancellation should be able to cancel frequencies between 2500Hz to 17500Hz