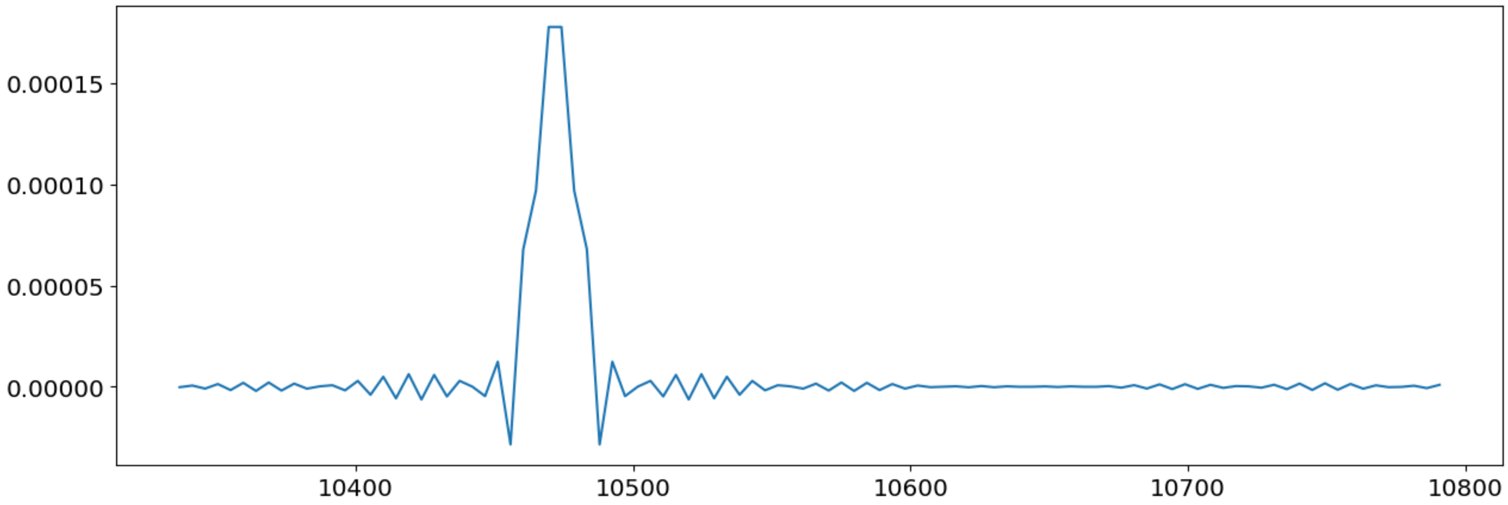
In [59]:
```
# HW6 1-d solution
# read in the data:
E_cr_sim_1d_cpx=np.fromfile("alossim_1d.dat",dtype=np.complex64)

# in this problem, some point targets were arranged along a single range rho_f, so parallel
# where rho_f can be calculated from the provided height h_sc and look angle, theta_f
#rho_f = h / np.cos(theta_f*np.pi/180.)
# the extent of the synthetic aperture at this range is given through the beamwidth theta_L
# where L is the antenna length in azimuth
theta_L_a = 0.88 * Lambda/L  # 0.88 gives a better approximation to the 3-dB beamwidth
# from this we can calculate the extent of the synthetic aperture the extent of the beam on
s_s_ref = - rho_f * theta_L_a / 2.  # start of synthetic aperture
s_e_ref = + rho_f * theta_L_a / 2.  # end of synthetic aperture
# the number of points in an array to hold the reference function will depend on the data s
# the spacing Delta_s = v_sc/prf, where both v_sc and prf are given in the problem statemen
n_s_ref = int(np.round((s_e_ref-s_s_ref)* Delta_s))  # number of points in synthetic apertu
# now can define a reference function extent for the synthetic aperture
s_sa_ref = np.linspace(s_s_ref,s_e_ref,n_s_ref)
# with this we can calculate the range history over this extent
rho_sa = np.sqrt(Rho_cr[Ind_cr[0]]**2+(s_sa_ref)**2)
# and the phase history over this extent
phi_sa = 4.*np.pi*rho_sa/Lambda
# and the matched filter history over this extent
ref_sa = np.exp(1j*phi_sa)

# note that since the flight path is parallel to the reflectors in this problem, we can cal
# and use it for all iterations of back projection.  However, if the track deviated from a
# parallel to the ground path of interest, we would need to recalculate this matched filter
# in the loop below.  As such in this problem under these simple assumptions, the backproje
# is no different from the correlation done in conventional range doppler processing

E_bp = np.zeros(s_sim.shape,dtype=np.complex128)
# loop over the output points, which for convenience here are the same as the input simulat
# You could specify a denser output grid to better resolve the point targets
for i,s in enumerate(s_sim):
    # compute the limits of the data needed for this output point to apply the matched filt
    # Since the problem has zero squint, the data needed is +/- half a beamwidth. Clip it i
    # the end of the array
    s_s_im = np.clip(s - rho_f * theta_L_a / 2.,s_sim[0],s_sim[-1])
    s_e_im = np.clip(s + rho_f * theta_L_a / 2.,s_sim[0],s_sim[-1])
    # compute the number of samples.  This should be the same as the matched filter length
    n_s_im = int(np.round((s_e_im-s_s_im)* Delta_s))
    # compute where in the array the data will be in pixels rather than in meters.
    stind = int((s_s_im-s_sim[0])*Delta_s)
    enind = stind+n_s_im
    #d o the matched filter operation (cross-multiply and sum) for each point only if the s
    # does not run off the end of the array
    if (len(E_cr_sim_1d[stind:enind]) == len(ref_sa)):
        E_bp[i] = np.sum(E_cr_sim_1d_cpx[stind:enind]*ref_sa)
```

In [62]:
```
# here is a plot of the entire compressed "imageplt.plot(s_sim,E_bp)
plt.show()
```

In [61]:
```
# there are three targets.  the small blip at the left is an ambiguity
# (aliased energy in the sidelobe of the antenna pattern).  The target to the right are act
# closely spaced.  This can be seen by expanding the plot around that target.  It looks wid
# if you were to oversample the output, you'd see two distinct peaks.  Try it!
plt.plot(s_sim,E_bp)
plt.plot(s_sim[9100:9200],E_bp[9100:9200])
plt.show()
```



Now let's look at these echoes in the presence of the thermal noise signature, with a field with noise power kTB_r.