```
Blocks:   0%|          | 0/1 [00:00<?, ?it/s]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~~~setigen/jupyter-notebooks/voltage/03_custom_signals.ipynb Cell 20 line 1
----> 1 rvb.record(output_file_stem='example_custom_signal',
      2           num_blocks=1,
      3           length_mode='num_blocks',
      4           header_dict={'HELLO': 'test_value',
      5                        'TELESCOP': 'GBT'},
      6           verbose=False)

File ~~jupyter-notebooks/voltage/../../setigen/voltage/backend.py:676, in RawVoltageBackend.record(self, output_file_stem, obs_length, num_blocks, length_mode, header_dict, digitize, load_template, verbose)
    674     tqdm.write(f'Creating block {j}...')
    675 self._make_header(f, header_dict)
--> 676 v = self.collect_data_block(digitize=digitize,
    677                             requantize=True,
    678                             verbose=verbose)
    680 f.write(xp.array(v, dtype=xp.int8).tobytes())
    681 if verbose:

File ~~jupyter-notebooks/voltage/../../setigen/voltage/backend.py:483, in RawVoltageBackend.collect_data_block(self, digitize, requantize, verbose)
    481 # Calculate the real voltage samples from each antenna
    482 t = time.time()
--> 483 antennas_v = self.antenna_source.get_samples(num_samples)
    484 self.sample_stage_t += time.time() - t
    486 for antenna in range(self.num_antennas):

File ~~jupyter-notebooks/voltage/../../setigen/voltage/antenna.py:124, in Antenna.get_samples(self, num_samples)
    121     samples = [[self.x.get_samples(num_samples),
    122                 self.y.get_samples(num_samples)]]
    123 else:
--> 124     samples = [[self.x.get_samples(num_samples)]]
    126 self.t_start += num_samples * self.dt
    127 self.start_obs = False

File ~~jupyter-notebooks/voltage/../../setigen/voltage/data_stream.py:225, in DataStream.get_samples(self, num_samples)
    221     self.v += noise_func(self.ts)
    223 for signal_func in self.signal_sources:
    224     # Ensure that the array is of the correct type
--> 225     signal_v = xp.array(signal_func(self.ts))
    226     # If there are complex voltages, make sure to cast self.v to complex
    227     if not xp.iscomplexobj(self.v) and xp.iscomplexobj(signal_v):

~~~/setigen/jupyter-notebooks/voltage/03_custom_signals.ipynb Cell 20 line 8
```

```
      7 def my_signal(ts):
----> 8     return 0.004 * chirp(t=ts,
      9                          f0=6002.2e6,
     10                          t1=rvb.time_per_block,
     11                          f1=6002.3e6,
     12                          method='quadratic')

File ~/.local/lib/python3.10/site-packages/scipy/signal/_waveforms.py:416, in chirp(t, f0, t1, f1, method, phi, vertex_zero)
    265 """Frequency-swept cosine generator.
    266
    267 In the following, 'Hz' should be interpreted as 'cycles per unit';
   (...)
    413
    414 """
    415 # 'phase' is computed in _chirp_phase, to make testing easier.
--> 416 phase = _chirp_phase(t, f0, t1, f1, method, vertex_zero)
    417 # Convert  phi to radians.
    418 phi *= pi / 180

File ~/.local/lib/python3.10/site-packages/scipy/signal/_waveforms.py:429, in _chirp_phase(t, f0, t1, f1, method, vertex_zero)
    422 def _chirp_phase(t, f0, t1, f1, method='linear', vertex_zero=True):
    423     """
    424     Calculate the phase used by `chirp` to generate its output.
    425
    426     See `chirp` for a description of the arguments.
    427
    428     """
--> 429     t = asarray(t)
    430     f0 = float(f0)
    431     t1 = float(t1)

File cupy/_core/core.pyx:1475, in cupy._core.core._ndarray_base.__array__()

TypeError: Implicit conversion to a NumPy array is not allowed. Please use `.get()` to construct a NumPy array explicitly.
```