Locust tutorial workshop

P. L. Slocum

Sept. 8, 2020

Overview of tutorial

- Brief intro to Locust
- Compiled examples:
  - Test signal.
  - Plane wave.
  - Transmitting antenna.
  - Electron radiating in free space.
  - Generating a Kassiopeia magnetic field map.
  - Free space detection with Project 8 hexbug config files.
  - Reconstructing electron signal with beam forming in Katydid.
  - Cluster jobs as simple example scripts.
- Extra slides: Parameter definitions and troubleshooting.

What is Locust?

- Originally it was an RF receiver and digitizer simulation*.
- Next it was expanded to calculate radiative fields from moving electrons**.
- Now it can also model the response of an antenna to incident fields**.
- It is modular and flexible, and is compiled with the Kassiopeia*** particle tracking package.

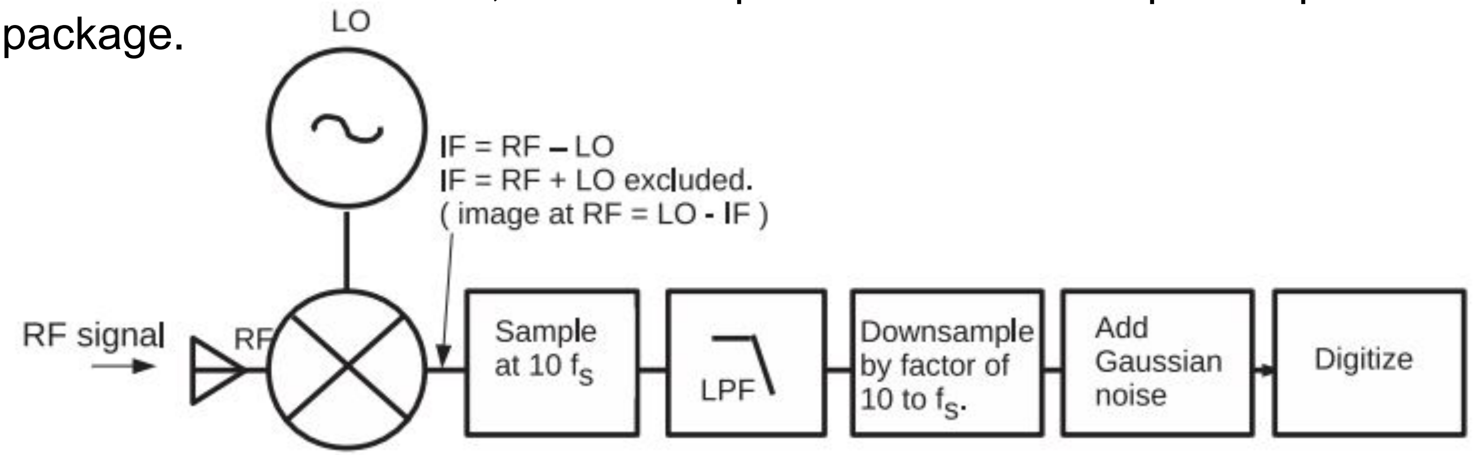
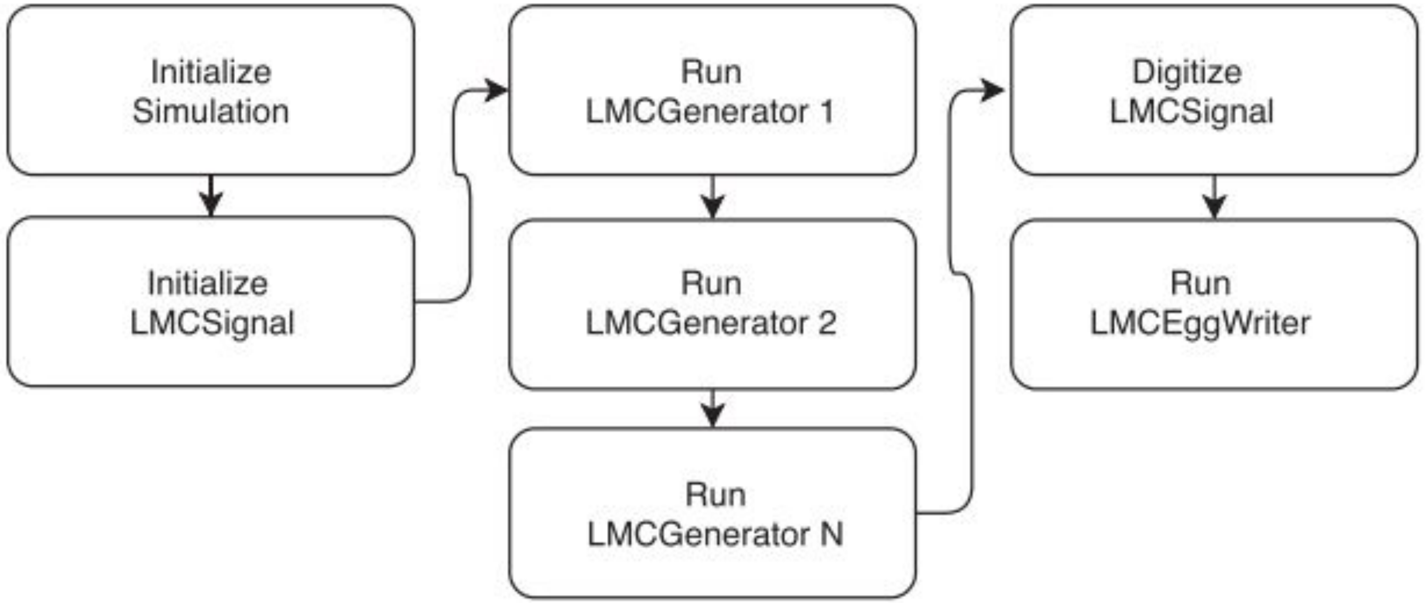


Figure 2. Block diagram of a receiver implemented algorithmically in Locust. Each of the square blocks represents one generator as in figure 1.

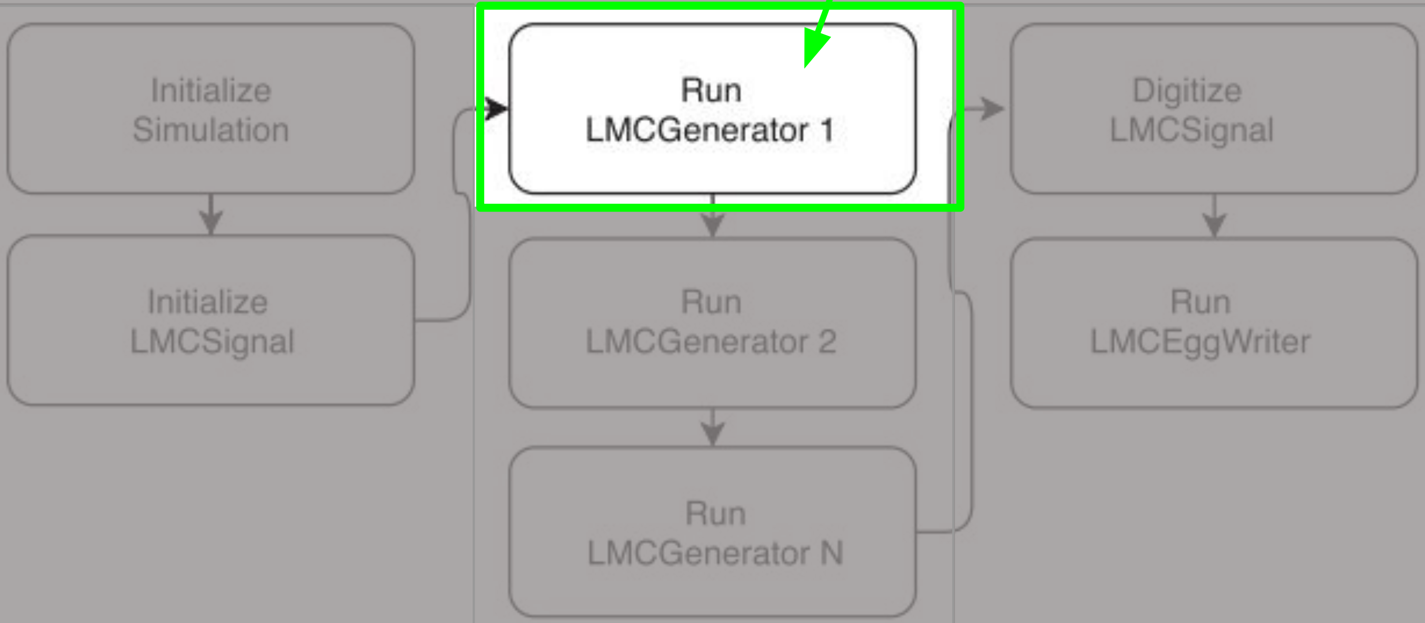
*Project 8 collaboration, New J. Phys. 21 (2019) 113051
**https://github.com/project8/locust_mc.git
***Furse D et al 2017 New J. Phys. 19 053012

Locust work flow diagram



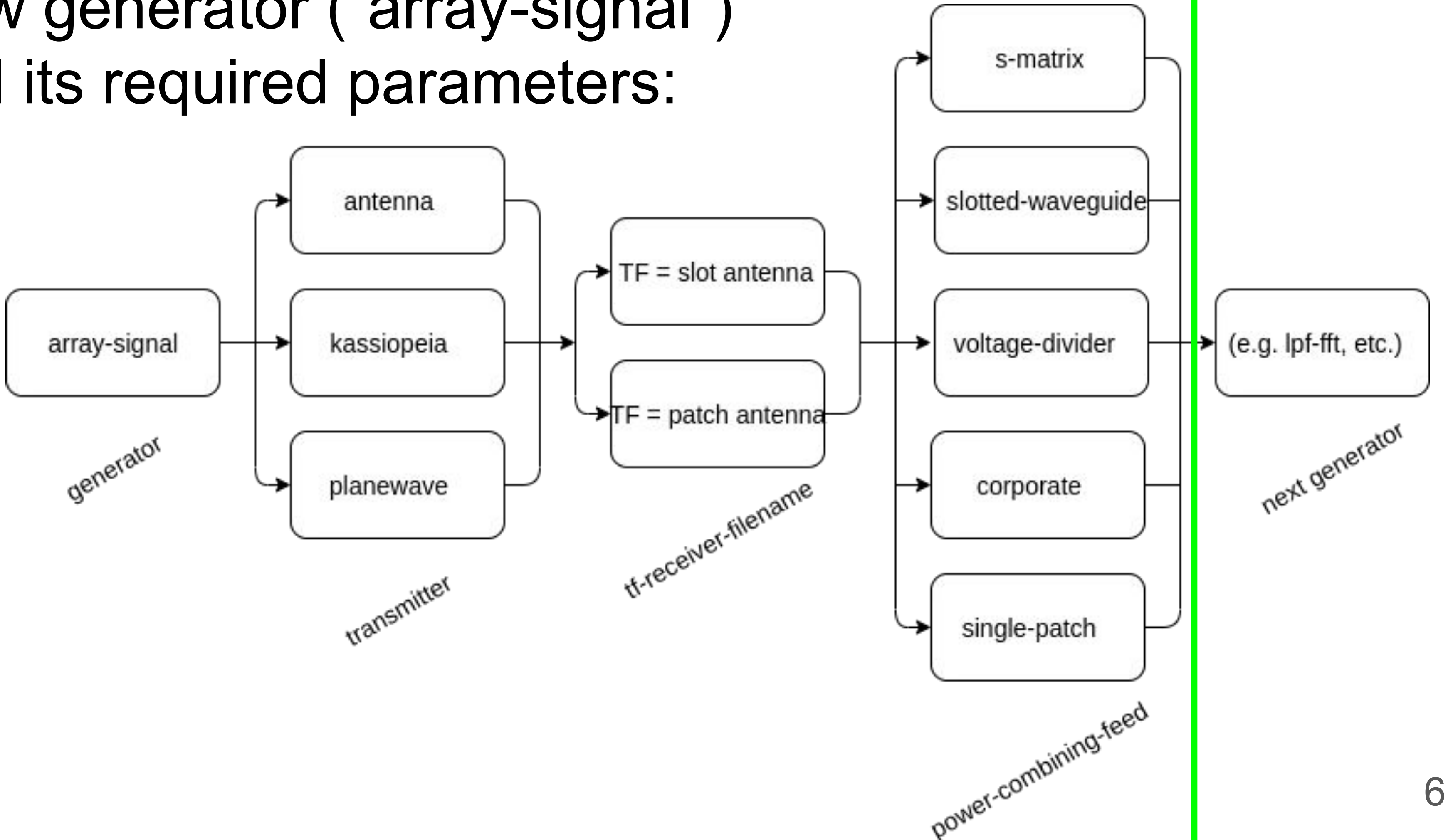
Locust work flow diagram

New functionality has been added here for antenna modeling



Project 8 collaboration, New J. Phys. 21 (2019) 113051

New generator ("array-signal") and its required parameters:



Basic Locust json file: List of generators to be run, followed by parameter definitions

```
{
   "generators":
   [
    "test-signal",
    "lpf-fft",
    "decimate-signal",
    "digitizer"
   ],

   "test-signal":
   {
   "rf-frequency": 20.7e9,
   "lo-frequency": 20.65e9,
   "amplitude": 5.0e-8
   },
```

Generator 1: Selected by application.

Generators 2+: Receiver chain

```
"simulation":
  {
     "egg-filename": "/usr/local/p8/locust/v2.1.1/output/locust_mc.egg",
     "n-records": 1,
     "n-channels": 1,
     "record-size": 8192
  },

  "gaussian-noise":
  {
  "noise-floor-psd": 4.0e-22,
  "domain": "time"
  },
```

```
    "digitizer":
    {
    "v-range": 8.0e-6,
    "v-offset": -4.0e-6
    }

}
```

Examples

First, some steps for setting up

Before we start the examples:

1. Install docker as in https://docs.docker.com/get-docker.
2. sudo docker pull project8/p8compute
3. sudo docker pull project8/p8compute-jupyter
4. Create a directory in your home directory, called ~/p8tutorial:

   mkdir ~/p8tutorial

5. Start p8compute-jupyter and leave it open for the duration of the workshop:

   ```
   docker run -p 8888:8888 -v ~/p8tutorial:/tmp
   project8/p8compute-jupyter
   ```

6. Open a browser tab using one of the links provided in the resulting terminal output.

First, some steps for setting up (cont.)

7. In the ~/p8tutorial directory, clone the hexbug and locust-tutorial repos:

```
cd ~/p8tutorial
git clone git@github.com:project8/hexbug
git clone git@github.com:project8/locust-tutorial
```

8. cd into to ~/p8tutorial/locust-tutorial/scripts:

```
cd ~/p8tutorial/locust-tutorial/scripts
```

9. Open tutorialLocustscript.sh and tutorialKatydidscript.sh with a text editor.

Example #1: Locust test signal

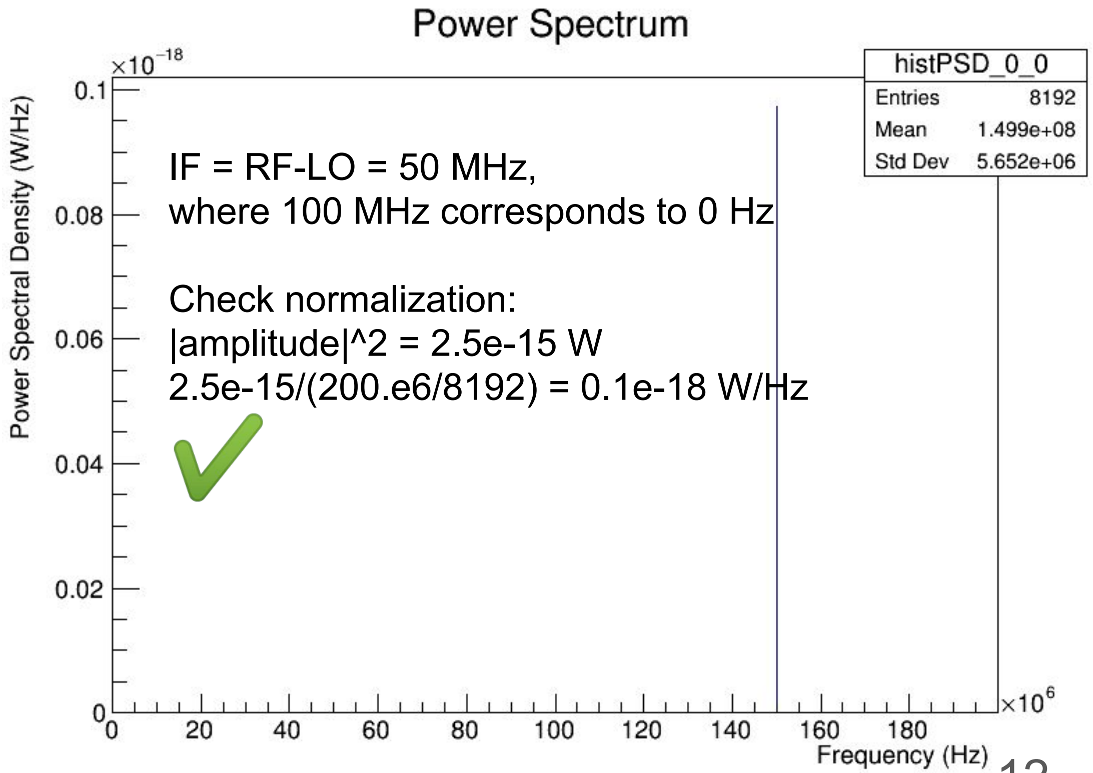
- Drive 50 ohm antenna with a sinusoid in LMCTestSignalGenerator:
  - (uncomment the command below #Example 1 in tutorialLocustscript.sh, then):
    `./tutorialLocustscript.sh`
  - (or interactively, in container):
    `LocustSim config=/path/to/config/LocustTestSignal.json`
  - Process egg file with `./tutorialKatydidscript.sh`

```
 "test-signal":
{
"rf-frequency": 20.7e9,
"lo-frequency": 20.65e9,
"amplitude": 5.0e-8
}
```

This is in LocustTestSignal.json

Example #1: Locust test signal, cont.

- In the jupyter browser tab, navigate with single clicks to /tmp/locust-tutorial/scripts/plotting/PlotPSD.ipynb
- Click the ⏩, then click “Restart and run all cells”.
- This plot should appear -> .



IF = RF-LO = 50 MHz,
where 100 MHz corresponds to 0 Hz

Check normalization:
|amplitude|^2 = 2.5e-15 W
2.5e-15/(200.e6/8192) = 0.1e-18 W/Hz
✓

Example #2: Locust plane wave

- Drive 6 patches combined in a passive voltage combiner in LMCPlaneWaveSignalGenerator:
    - (uncomment the command below #Example 2 in tutorialLocustscript.sh, then):
      `./tutorialLocustscript.sh`
    - (or interactively, in container):
      `LocustSim config=/path/to/config/LocustPlaneWaveTemplate.json`
    - Process egg file with `./tutorialKatydidscript.sh`

```
"array-signal":
  {
        "transmitter": "planewave",
        "transmitter-frequency": 25.9281e9,
        "planewave-amplitude": 1.0e-8,
        "AOI": 0.0,
        "voltage-check":  true,
        "lo-frequency": 25.8781e9,
        "nelements-per-strip": 6,
        "element-spacing": 0.007753,
        "power-combining-feed": "voltage-divider",
        "tf-receiver-filename": "/usr/local/p8/locust/v2.1.1/data/PatchTFLocust.txt",
        "tf-receiver-bin-width": 0.01e9
  }
```

verbose

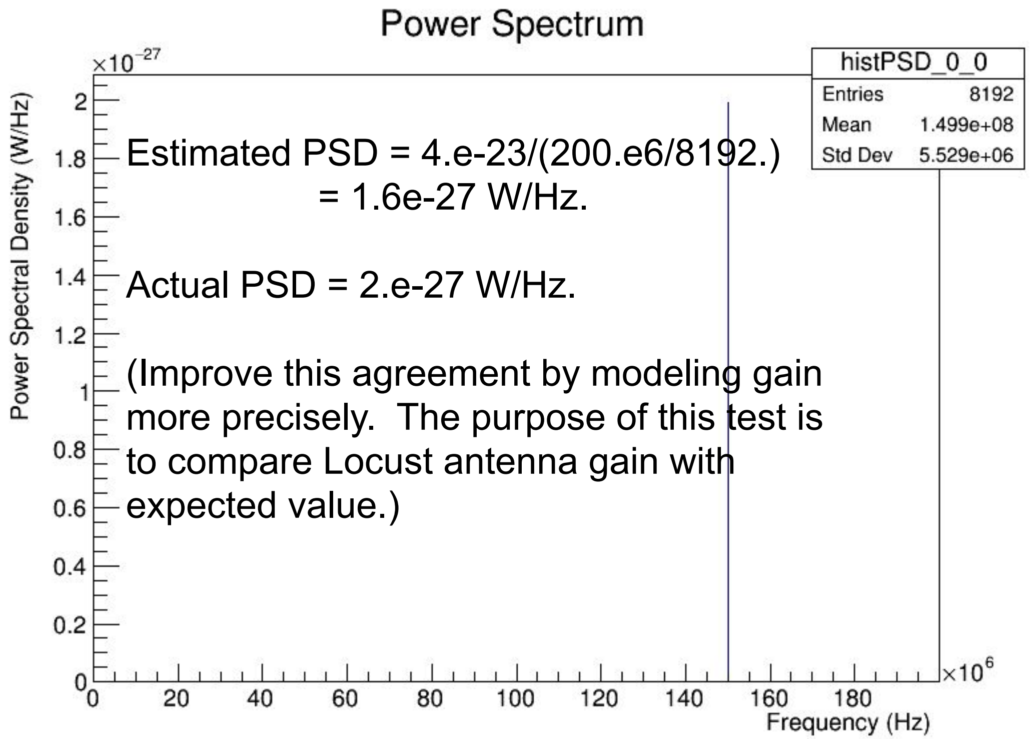
transfer function

This is in LocustPlaneWaveTemplate.json

Example #2: Locust plane wave, cont.

- In the jupyter browser tab, navigate to /tmp/locust-tutorial/scripts/plotting/PlotPSD.ipynb
- Click the ▶▶, then click "Restart and run all cells". See this plot. ->
- Check calibration:
  - Estimate G = 10 dB for 6-patch voltage divider.
  - Effective aperture A_e for 6-patch voltage divider = 1.5e-4 m^2.
  - Power S in plane wave: $c\varepsilon_0|E|^2$ = 2.7e-19 W/m^2; power incident on antenna strip S \dot A = 4.e-23 W

$$A_e = \frac{\lambda^2}{4\pi} G$$



Estimated PSD = 4.e-23/(200.e6/8192.)
= 1.6e-27 W/Hz.

Actual PSD = 2.e-27 W/Hz.

(Improve this agreement by modeling gain more precisely. The purpose of this test is to compare Locust antenna gain with expected value.)

Example #3: Dipole antenna transmitter

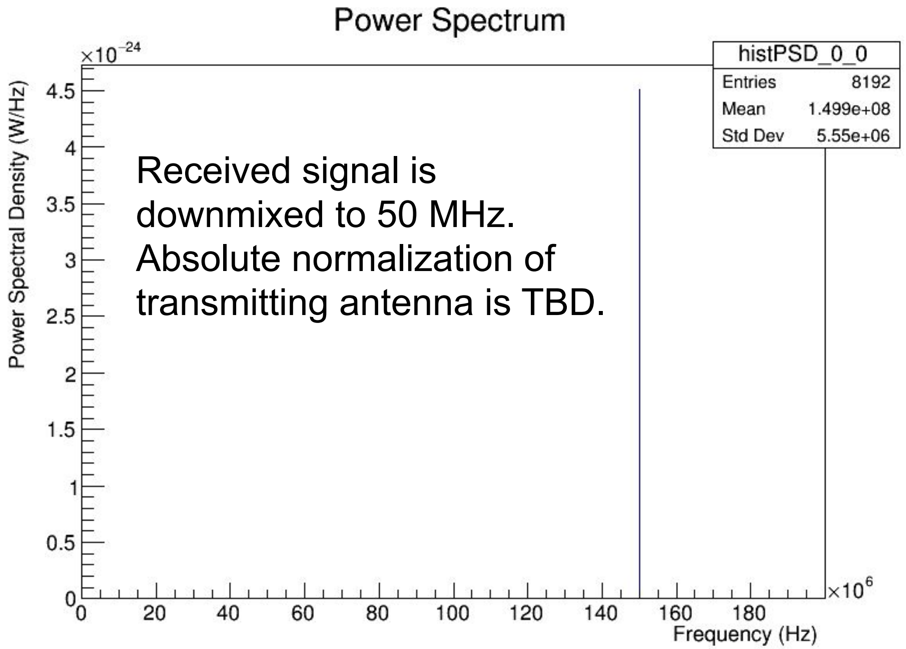
- Drive single patch in LMCAntennaSignalGenerator:
    - (uncomment the command below #Example 3 in tutorialLocustscript.sh, then):
      `./tutorialLocustscript.sh`
    - (or interactively, in container):
      `LocustSim config=/path/to/config/LocustMagDipoleAntennaTemplate.json`
    - Process egg file with `./tutorialKatydidscript.sh`

```
  "array-signal":
    {
        "transmitter": "antenna",
        "transmitter-frequency": 25.9281e9,
        "antenna-voltage-amplitude": 1.0,
        "tf-transmitter-filename":
"/usr/local/p8/locust/v2.1.2/data/UncoupledHalfWaveeDipoleTF.txt",
        "voltage-check": true,
        "lo-frequency": 25.8781e9,
        "array-radius": 0.05,
        "nelements-per-strip": 1,
        "power-combining-feed": "single-patch",
        "tf-receiver-filename":
```

This is in LocustMagDipoleAntennaTemplate.json

Example #3: Dipole antenna transmitter, cont.

- In the jupyter browser tab, navigate to /tmp/locust-tutorial/scripts/plotting/PlotPSD.ipynb
- Click the ▶▶, then click "Restart and run all cells". See this plot. ->



Example #4: Locust-Kass electrons in free space

- Drive voltage divider with 3 sequential Kassiopeia electrons in LMCArraySignalGenerator:
  - (uncomment the command below #Example 4 in tutorialLocustscript.sh, then):
    `./tutorialLocustscript.sh`
  - (or interactively, in container):
    `LocustSim config=/path/to/config/LocustFreeSpaceTemplate.json`
  - Process egg file with `./tutorialKatydidscript.sh`

```
"array-signal":
    {
        "transmitter": "kassiopeia",
        "event-spacing-samples": 15000,
        "lo-frequency": 25.8781e9,
        "array-radius": 0.05,
        "nelements-per-strip": 6,
        "element-spacing": 0.007753,
        "power-combining-feed": "voltage-divider",
        "tf-receiver-filename": "/usr/local/p8/locust/v2.1.1//data/PatchTFLocust.txt",
        "tf-receiver-bin-width": 0.01e9,
        "Xml-filename":
"/usr/local/p8/locust/v2.1.1/config/ LocustKass_FreeSpace_Template.xml "
    },
```

(fast, 10x) samples between Kass events.

This is in LMCFreeSpaceTemplate.json.

Kassiopeia config file, next slide

Example #4: Locust-Kass free space electrons, cont.

```
<ks_simulation
    name="project8_simulation"
    run="1"
    seed="[seed]"
    events="3"
    magnetic_field="field_electromagnet"
    magnetic_field="field_magnetic_main"
    space="space_world"
    generator="[generator]"
    trajectory="[trajectory]"
    space_navigator="nav_space"
    surface_navigator="nav_surface"
    writer="write_root"
    add_static_run_modifier="run_pause"
    add_static_event_modifier="event_hold"
    add_static_step_modifier="rad_extr"
/>
```

In LocustKass_FreeSpace_Template.xml, **<ks_simulation/>** defines simulation with parameters defined elsewhere in file.

3 sequential e-'s will be generated

"generator" defines e- starting kinematics.
"trajectory" solves time-dependent motion.

Locust modifications

Example #4: Locust-Kass free space electrons, cont.

Also in LocustKass_FreeSpace_Template.xml:

```
<ksterm_max_time name="term_max_time" time="1.0e-6"/>
```

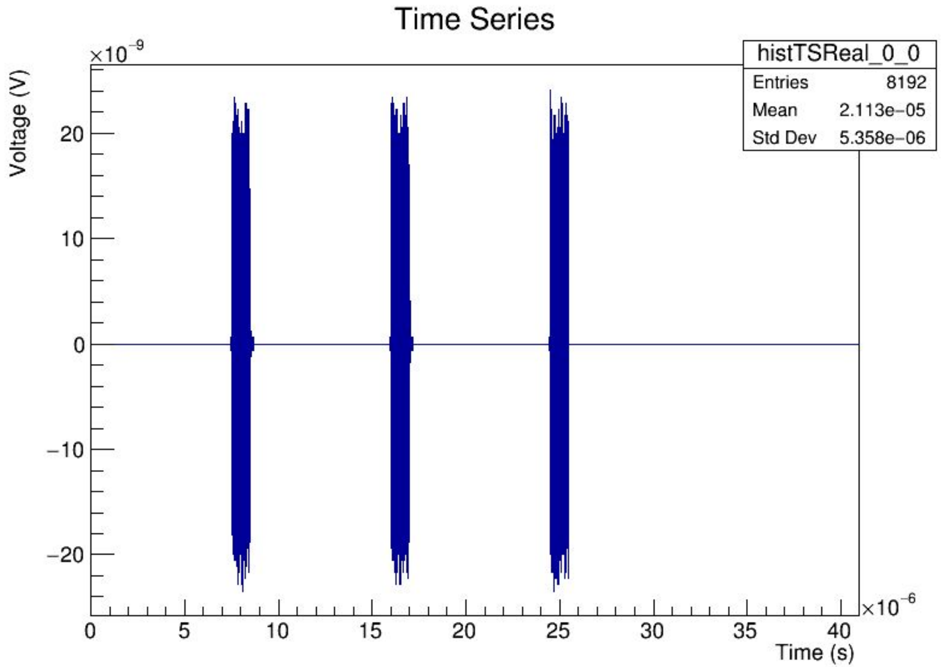
electron max duration (sec)

```
<geometry>
    <include name="[config_path]/FreeSpaceGeometry.xml"/>
</geometry>
```

Trap geometry file

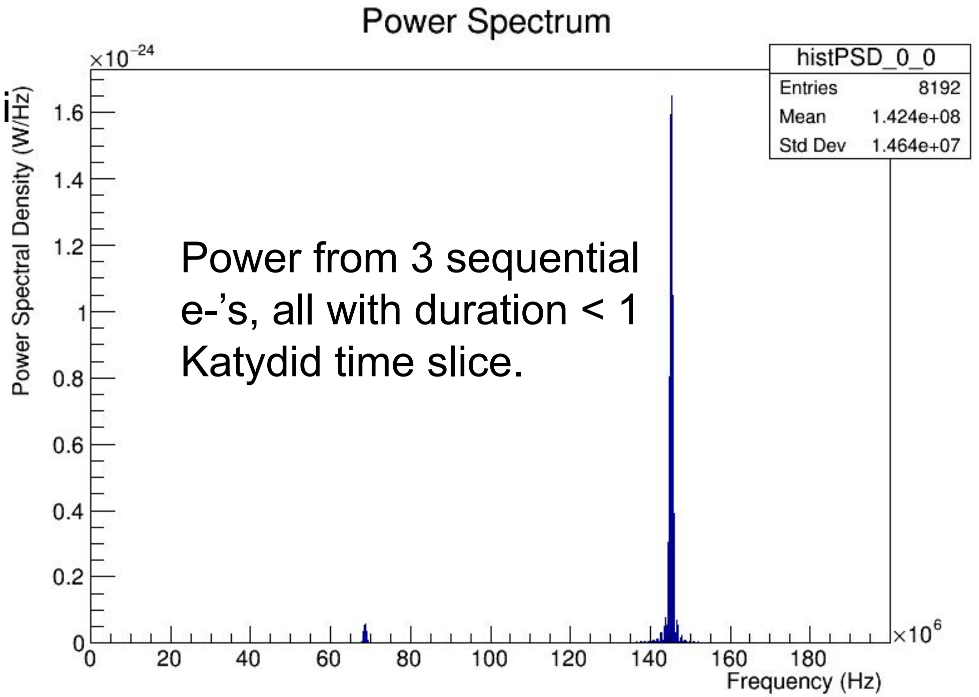
Example #4: Locust-Kass free space electrons, cont.

- In the jupyter browser tab, navigate to /tmp/locust-tutorial/scripts/plotting/PlotTimeSeries.ipynb
- Click the ▶▶, then click “Restart and run all cells”. See this plot. ->
- We see signals from 3 electrons spaced at 15000 fast samples (e.g. 2 GHz or 10x acq-rate), each with duration 1.e-6 s.



Example #4: Locust-Kass free space electrons, cont.

- In the jupyter browser tab, navigate to /tmp/locust-tutorial/scripts/plotting/PlotPSD.ipynb
- Click the ⏩, then click "Restart and run all cells". See this plot. ->
- Power from 3 identical θ=90° electrons in one Katydid time slice. Carrier at 45 MHz; sideband at -32 MHz due to low-amplitude trap-dependent axial motion.



Example #5: Generate Kassiopeia field map

- Without Locust, use Kassiopeia to measure B field values along field lines.
    - (uncomment the command below #Example 5 in tutorialLocustscript.sh, then):
      `./tutorialLocustscript.sh`
    - (or interactively, in container):
      `/path/to/LMCKassiopeia /path/to/config/JustKassFieldMap.xml`
    - Output is Root TTree file ~/p8tutorial/locust-tutorial/output/FieldLineSeed*.root . Plot TTree variables e.g. `position_z` vs. `magnetic_field_z` for field map.

```
<external_define name="generator" value="gen_bfieldlines" />
```

specialized generator

```
<geometry>
    <include name="[config_path]/FreeSpaceGeometry.xml"/>
</geometry>
```

Trap geometry

Lines in JustKassFieldMap.xml

Example #6A: hexbug config files

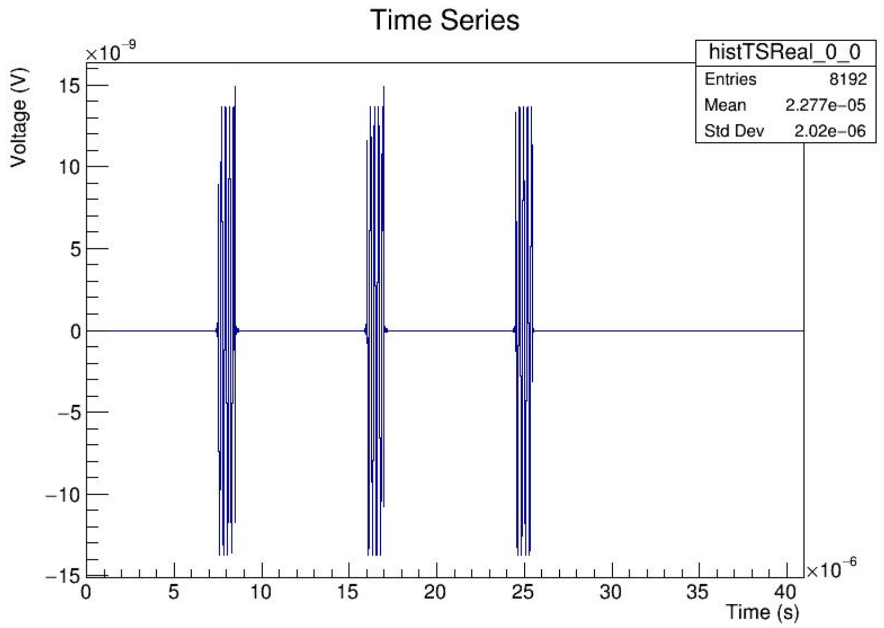
- Drive 5-slot slotted waveguide antenna with 3 sequential Kassiopeia electrons in LMCArraySignalGenerator:
  - (uncomment the command below #Example 6 in tutorialLocustscript.sh, then):
    `./tutorialLocustscript.sh`
  - (or interactively, in container):
    `LocustSim config=/tmp/hexbug/Phase3/LocustPhase3Template.json`
  - Process egg file with `./tutorialKatydidscript.sh`

```
 "array-signal":
    {
        "transmitter": "kassiopeia",
         "event-spacing-samples": 15000,
        "lo-frequency": 25.8781e9,
        "array-radius": 0.1,
        "nelements-per-strip": 5,
        "element-spacing": 0.007753,
        "power-combining-feed": "slotted-waveguide",
        "tf-receiver-filename":
"/tmp/hexbug/Phase3/TransferFunctions/FiveSlotTF.txt",
        "tf-receiver-bin-width": 0.01e9,
        "xml-filename": "/tmp/hexbug/Phase3/LocustKassElectrons.xml"
```

Trap and transfer function are P8-specific.

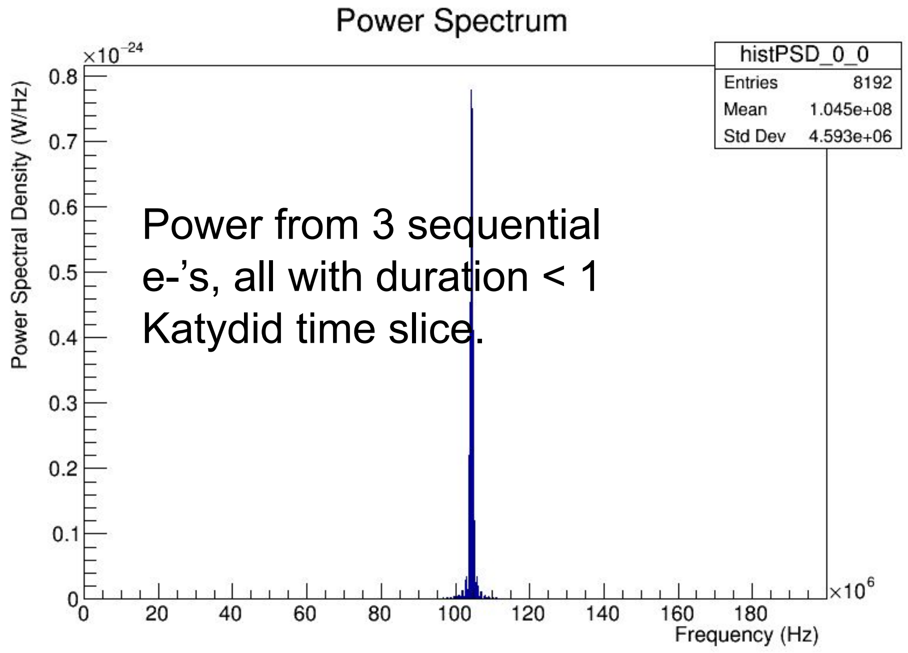
Example #6A: hexbug config files, cont.

- In the jupyter browser tab, navigate to /tmp/locust-tutorial/scripts/plotting/PlotTimeSeries.ipynb
- Click the ▶️▶️, then click “Restart and run all cells”. See this plot. ->
- We see signals from 3 electrons spaced at 15000 fast samples (e.g. 2 GHz or 10x acq-rate), each with duration 1.e-6 s.



Example #6A: hexbug config files, cont.

- In the jupyter browser tab, navigate to /tmp/locust-tutorial/scripts/plotting/PlotPSD.ipynb
- Click the ⏩, then click “Restart and run all cells”. See this plot. ->
- Power from 3 identical θ=90° electrons in one Katydid time slice. Carrier at 5 MHz.



Power from 3 sequential e-’s, all with duration < 1 Katydid time slice.

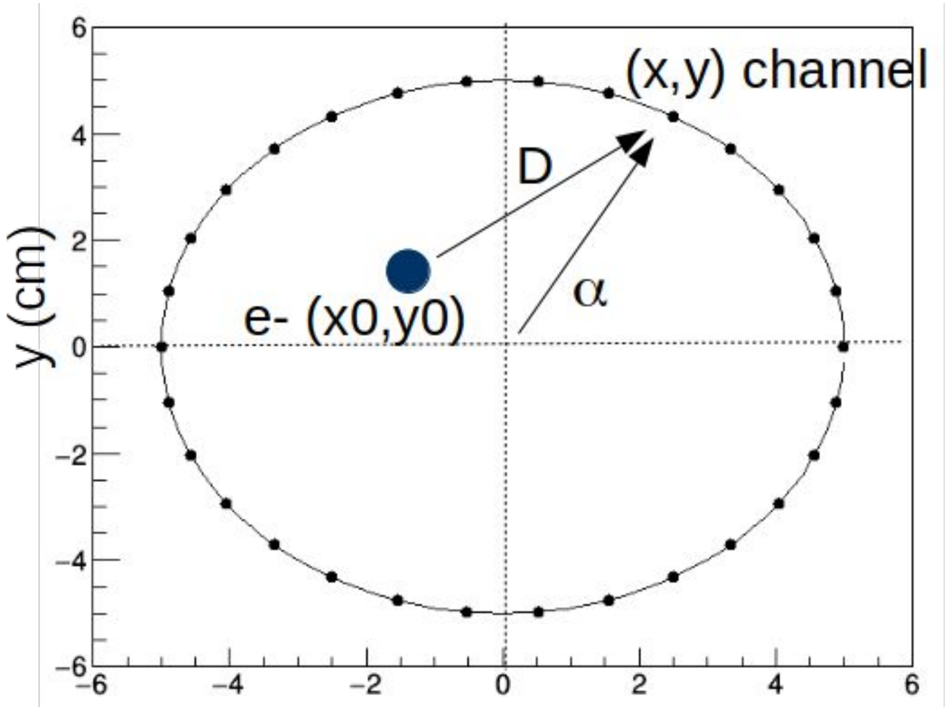
Example #6B: sim->analysis chain with hexbug configs

Let’s do a slightly longer run, with 4 channels, and then reconstruct the signal in Katydid. The Locust run should take 10 minutes.

1. Open ~/p8tutorial/hexbug/Phase3/LocustPhase3Template.json for editing.
   a. Delete this: `"event-spacing-samples": 15000,` (this defaults to 150000, which makes the event start at the end of time slice #1 if slice size = 8192.
   b. Change n-channels to 4, like this: `"n-channels": 4`
2. Open ~/p8tutorial/hexbug/Phase3/LocustKassElectrons.xml for editing.
   a. Increase max_time to 0.5e-4 s, like this: `"term_max_time" time="0.5e-4"/>`
   b. Change n-events to 1, like this: `events="1"`
   c. Change the pitch angle to 87°, also under “gen_uniform”:
      `<theta_uniform value_min="87.0" value_max="87.0"/>`
3. In tutorialKatydidscript.sh, uncomment the 2nd Katydid command that starts like this:
   `Katydid -c ${katydiddir}/config/ChannelAggregatorConfig.yaml …`
4. Run the scripts: `./tutorialLocustscript.sh && ./tutorialKatydidscript.sh`

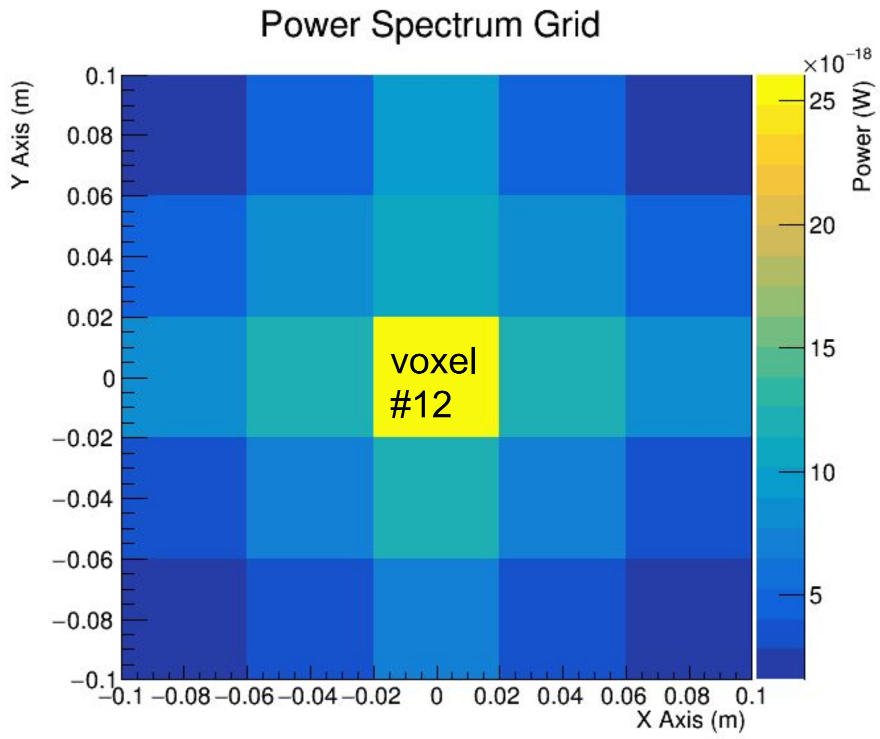
Beam forming signal extraction after Locust

- Steps to reconstruct signal:
  - Complex FFT in each channel.
  - Rotate voltage phase atan(Q/I) in each channel for each grid voxel in xy.
  - Sum complex phase-sensitive voltages over all channels.
  - Convert voltage sum to PSD.
  - Grid voxels associated with high array power may have an e-.
- A possible problem:
  - “High array power” can be caused by a powerful sideband at f != f_cyc.
  - This can complicate spectroscopy.



Example #6B: sim->analysis chain with hexbug, cont.

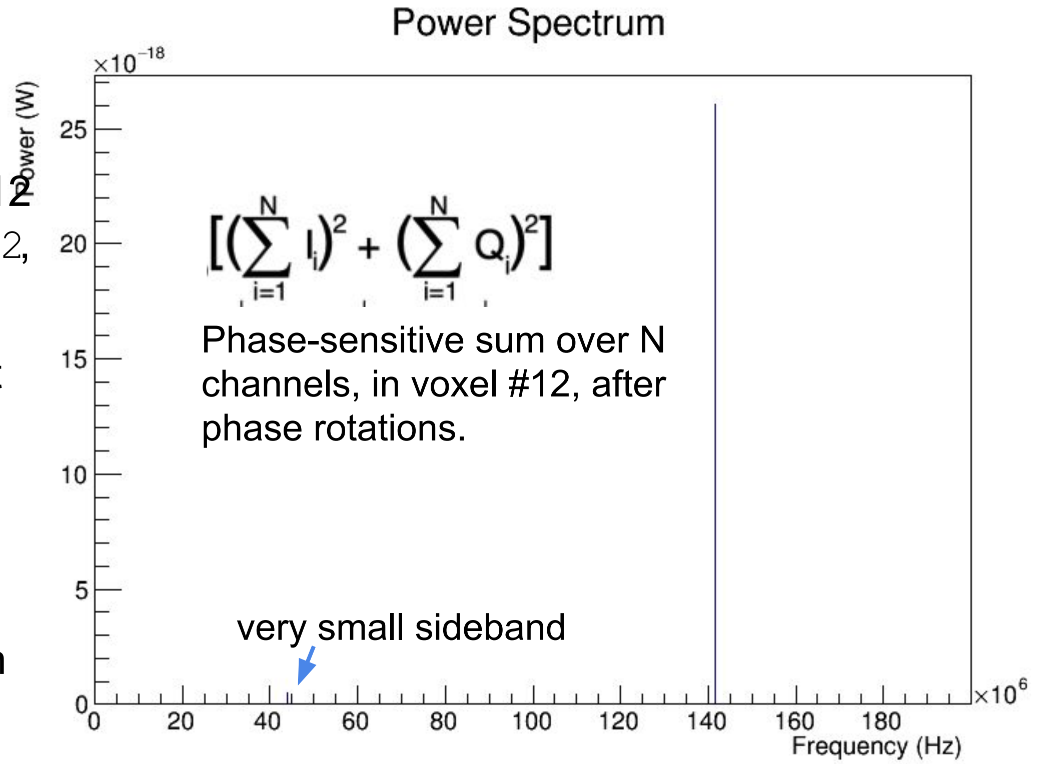
- In the jupyter browser tab, navigate to /tmp/locust-tutorial/scripts/plotting/PlotGrid.ipynb. Click the ▶▶, See this plot. ->
- Highest power feature: Power_max = 26.e-18 W.
- **To infer absolute detected power, we have to infer ideal power combining between the e.g. N=4 channels, and we should scale Power_max by 1/N=1/4. **.**
- (If simulating noise with signal, SNR will be correct with or without the above scaling.)



*M. Grando and M. Jones, https://3.basecamp.com/3700981/buckets/3107037/uploads/2956124063,
*N. Buzinsky, private conversations.

Example #6B: sim->analysis chain with hexbug cont.

- Each voxel in the grid corresponds to the maximum power in a frequency spectrum, after a set of N voltage phase rotations.
- In `PlotGrid.ipynb` we see pixel #12 in time slice 2 as `histAggChPS_2_12`, here ->
- Line (carrier or sideband) with highest power defines voxel power in grid. Here the carrier is highest; reconstruction straightforward (w/o θ correction) ->
- If sideband powers were highest, then grid voxels would represent sideband powers -> potentially hard to find carrier, but with θ constrained.



$$\left[\left(\sum_{i=1}^{N} I_i\right)^2 + \left(\sum_{i=1}^{N} Q_i\right)^2\right]$$

Phase-sensitive sum over N channels, in voxel #12, after phase rotations.

Example #6C: Systematic effects (line broadening)

Now try moving the electron off axis:

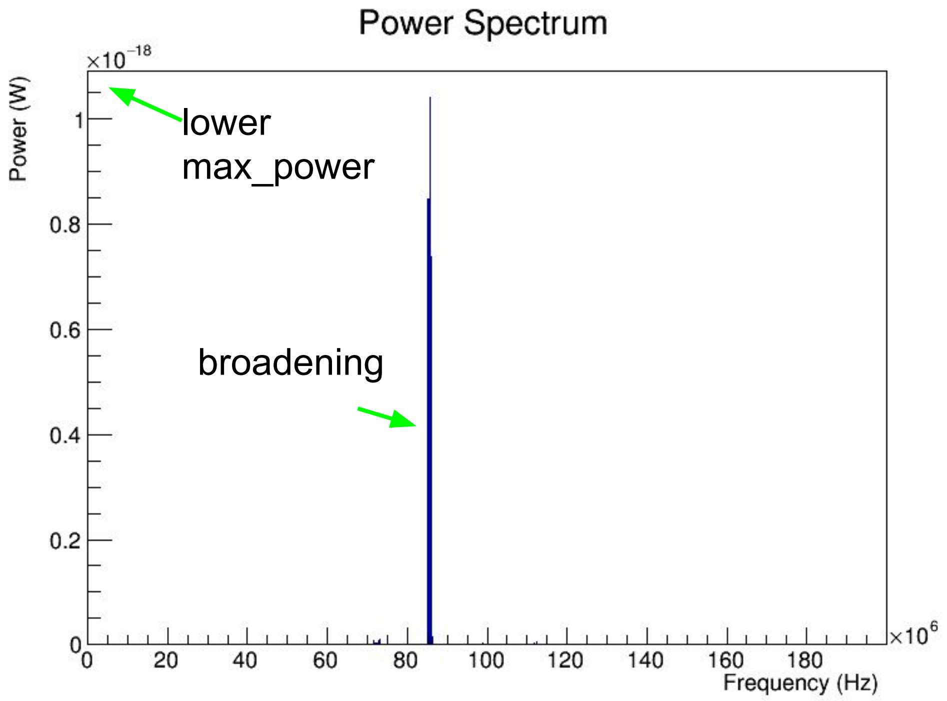
1. Open ~/p8tutorial/hexbug/Phase3/LocustKassElectrons.xml for editing. Move starting position to (x,y) = (0.01, 0.01) m:

```
<x_uniform value_min="0.01" value_max="0.01"/>
<y_uniform value_min="0.01" value_max="0.01"/>
```

2. Run the scripts: `./tutorialLocustscript.sh && ./tutorialKatydidscript.sh`

Example #6C: Systematic effects (line broadening, cont.)

- In the jupyter browser tab, navigate to /tmp/locust-tutorial/scripts/plotting/PlotGrid.ipynb. Click the ▶▶, See this plot. ->
- Notice the broadening over cyclotron orbit due to radial gradient in B.
- Also notice the peak power has decreased -> lost SNR. Try to avoid this problem.



Example #6D: Systematic effects (sidebands)

Now let’s change to a different (non-harmonic) trap. (For a list of trap names and properties, see the hexbug README):

1. Open ~/p8tutorial/hexbug/Phase3/LocustKassElectrons.xml for editing. Change starting position back to (x,y) = (0,0):

```
<x_uniform value_min="0.0" value_max="0.0"/>
<y_uniform value_min="0.0" value_max="0.0"/>
```

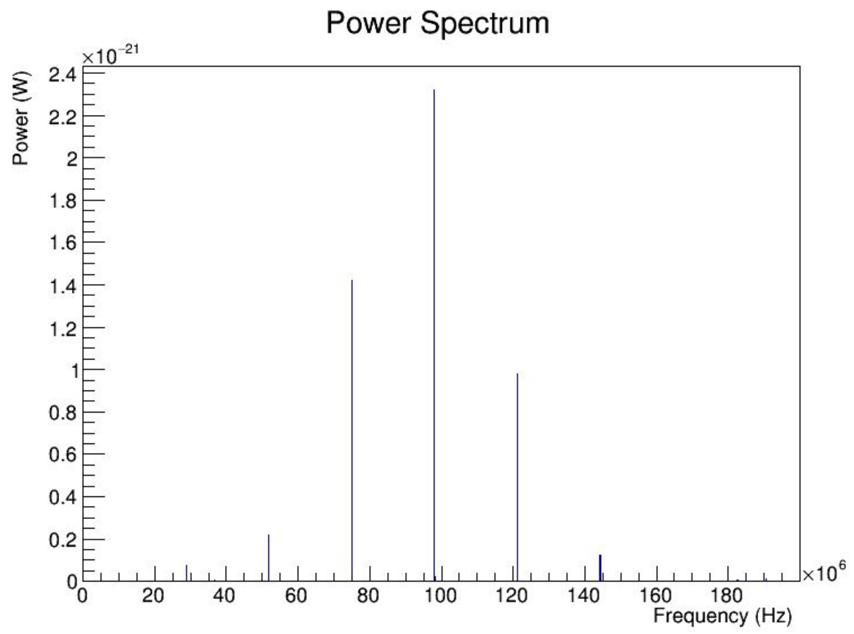
2. Edit the trap version to V00_00_09:

```
<include name="/tmp/hexbug/Phase3/Trap/FreeSpaceGeometry_V00_00_09.xml"/>
```

3. Run the scripts: `./tutorialLocustscript.sh && ./tutorialKatydidscript.sh`

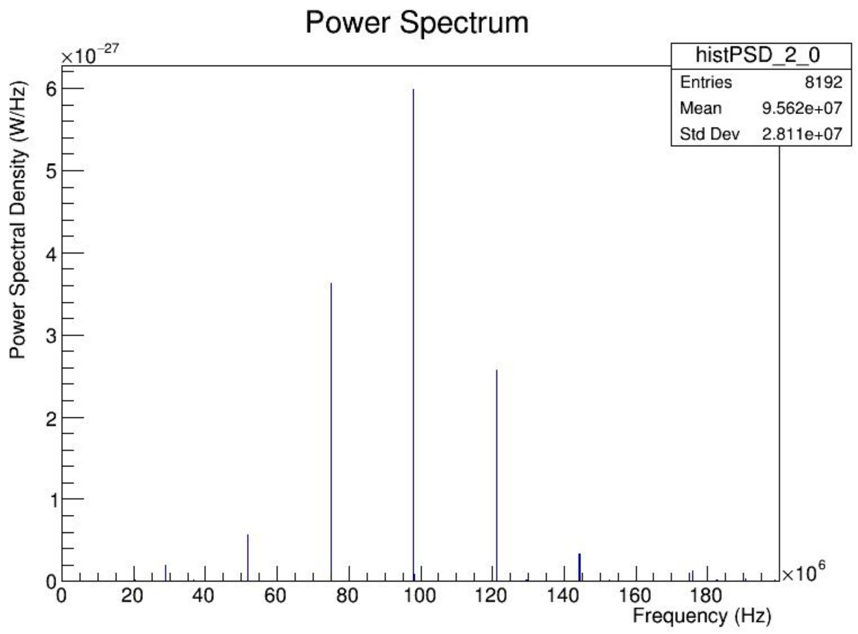
Example #6D: Systematic effects (sidebands, cont.)

- In the jupyter browser tab, navigate to /tmp/locust-tutorial/scripts/plotting/PlotGrid.ipynb. Click the ⏩, See this plot. ->
- Sidebands are now present. From the plot alone it is not always obvious which feature is the carrier.
- Grid search will select feature with highest power, but it might not be at f_cyc, and it may not be obvious what to report for e-energy. Reconstruction is more complicated.



Example #6D: Systematic effects (sidebands, cont.)

- Single-channel sideband structure is usually very similar to that in whole array. -> It is often faster to examine sidebands in just one channel.
- In the jupyter browser tab, navigate to /tmp/locust-tutorial/scripts/plotting/PlotPSD.ipynb. Choose time slice #2, channel #0 as in dataHisto = histFile.Get("histPSD_2_0") Click the ▶▶, See this plot. ->
- Notice similarity to sideband pattern on previous slide.



Example 7A: Running a Locust job on pnnl cluster

- After following instructions here:
  https://discourse.project8.org/t/using-dirac/89
  Start the dirac client like this:
  ```
  docker-compose run -v ~/p8tutorial:/tmp --rm p8-dirac-client
  ```
- In the container,
  ```
  cd /tmp/locust-tutorial/scripts/pnnl/locust
  dirac-wms-job-submit my_first_locust_job.jdl
  ```
- Follow instructions at
  https://discourse.project8.org/t/dirac-job-submission-basics/147
- The output *.egg file will be in the output sandbox.

Example 7B: Processing Locust output on pnnl cluster

- After following instructions here:
  https://discourse.project8.org/t/using-dirac/89
  Start the dirac client like this:
  ```
  docker-compose run -v ~/p8tutorial:/tmp --rm p8-dirac-client
  ```
- In the container,
  ```
  cd /tmp/locust-tutorial/scripts/pnnl/katydid
  dirac-wms-job-submit my_first_katydid_job.jdl
  ```
- Follow instructions at
  https://discourse.project8.org/t/dirac-job-submission-basics/147
- The output *.root file(s) will be in the output sandbox.

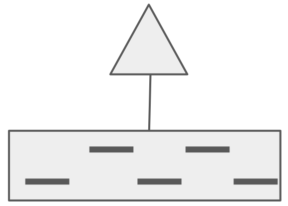
Example 8: Running p8compute in Singularity on Yale Grace cluster

- In ~/p8tutorial/locust-tutorial/scripts/yale/ are two scripts: `locustsingularity.batch` and `katydidsingularity.batch`.
  - Interactive running:
    - Log onto compute node with 2 cores, e.g. `srun -c2 --time=03:00:00 --pty -p interactive bash`
    - Configure local paths for your I/O in the *.batch file. The `sifdir` path does not need to be edited.
    - Run scripts interactively as `./locustsinglarity.batch` or `./katydidsingularity.batch` .
  - Batch job:
    - Configure paths in *.batch as above.
    - `sbatch locustsinglarity.batch` or `sbatch katydidsingularity.batch` .
  - Documentation on running jobs on Yale clusters: https://docs.ycrc.yale.edu/clusters-at-yale/job-scheduling/

Extra slides

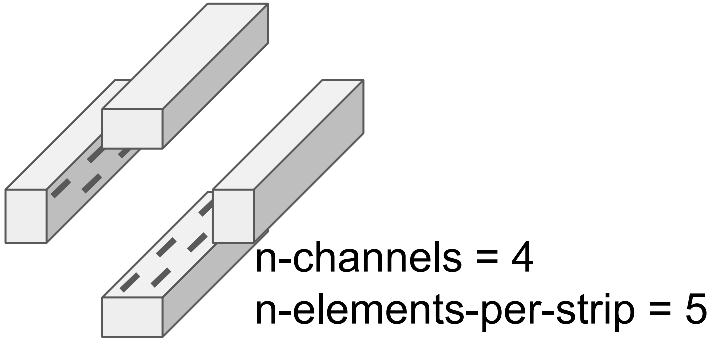
Parameters for positioning array in “array-signal” generator

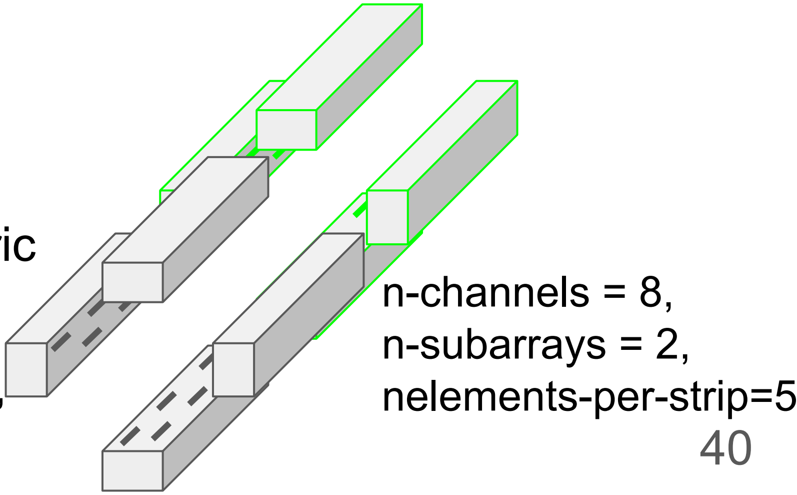
- “n-elements-per-strip”: Number of antenna elements (typically slots or patches) summed at the input to one amplifier.
- “array-radius”: Radius [m] of cylindrically-shaped antenna array.
- “element-spacing” : [m] spacing between elements. Determined for each slotted-waveguide transfer function.


n-elements-per-strip = 5

Parameters for positioning array in “array-signal” generator, cont.

- “n-channels”: Number of amplifiers in cylindrically-shaped array, spaced evenly around the radius of the cylinder.
- “n-subarrays”: Number of antenna rings.
- “zshift-array”: Longitudinal shift of entire antenna array relative to everything else (such as a trap, trapped electrons, or transmitting antenna). If not defined, then array is symmetric about z=0.
- Other parameters are listed in later slides here, and/or are defined in Locust classes on github.





Other parameters in “array-signal” generator

- “transmitter”: Select either kassiopeia, antenna, or planewave.
- “event-spacing-samples”: Number of fast-sampled (10x) digitizer samples preceding each electron event (if using kassiopeia). The default value is 150000, which allows an electron to start near the end of time slice #1 if Katydid slice-size = 8192 and Locust “acquisition-rate” is 200 MHz (default value).
- “voltage-check”: Print voltage values to terminal while the simulation is running. This is quite verbose and is used for quick debugging. A better way to check voltage values is to examine time series histograms in Katydid output.
- “lo-frequency”: Local oscillator frequency [GHz].

Other parameters in “array-signal” generator

- "tf-receiver-filename": Full path to transfer function file.
- "tf-receiver-bin-width": Resolution bandwidth [Hz] of transfer function.
- "xml-filename”: Full path to Kassiopeia xml file containing standard Locust modifications.

Troubleshooting

- Histograms and time series are empty.
  - Digitizer range is too large.
  - Record length is too short (e.g. record ended before electron started).
  - “event-spacing-samples” has delayed the event start time(s) past the end of the record.
  - LO is tuned such that the signal is out of the window.
  - Katydid n-slices is too small and so the Locust signal was not processed.
  - Katydid was not run at all.
  - B field is higher/lower than expected, moving signal out of window.
- Unexpected high-power artifacts
  - Digitizer range is too small (or possibly too large).
- Other
  - Switch on the verbose “voltage-check”: “true” flag to check for reasonable voltage values while simulation is running.
  - Look at the Katydid time series of voltages: check for clipping and quantization.

Other Root GUIs to try (1)

```
docker run -it --net=host --env="DISPLAY"
--volume="$HOME/.Xauthority:/root/.Xauthority:rw" -v ~/p8temp:/tmp
project8/p8compute

source /usr/local/p8/compute/v0.10.0/setup.shsource
/usr/local/p8/compute/v0.10.0/setup.sh

root

new TBrowser()
```

Other Root GUIs to try (2)

```
docker run -it --rm -e DISPLAY=host.docker.internal:0 -v
~/p8temp:/tmp project8/p8compute
source /usr/local/p8/compute/v0.10.0/setup.sh
 root
new TBrowser()
```

(The above required (on a Mac) first checking “Allow connections from network clients” in XQuartz security settings.)

Other Root GUIs to try (3)

Root’s official docker container (rootproject/root)