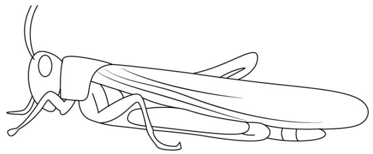
Locust v2.5.1 notes

P. L. Slocum

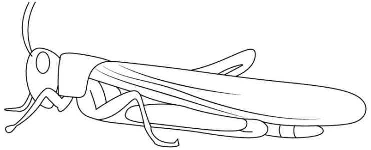
Aug. 26, 2023

Outline



- Brief introduction and instructions for setting up
- Tutorial examples
- Intermediate checks
- Development options
- Config files and cluster tools (presently partially internal).

Setting up



1. Install docker as in the instructions, https://docs.docker.com/engine/install/ .
2. Pull the Locust container: docker pull ghcr.io/project8/locust_mc:latest
3. Pull the p8compute container: docker pull project8/p8compute:latest
4. Create a directory in your home directory, e.g.

   mkdir ~/p8tutorial

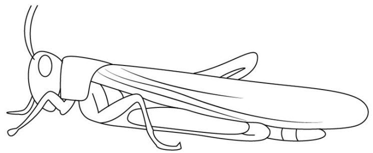
   (Steps 5-7 are optional for plotting Root histograms):

5. Pull the p8compute-jupyter container: docker pull project8/p8compute-jupyter
6. Start p8compute-jupyter and leave it open for the duration of the tutorial:

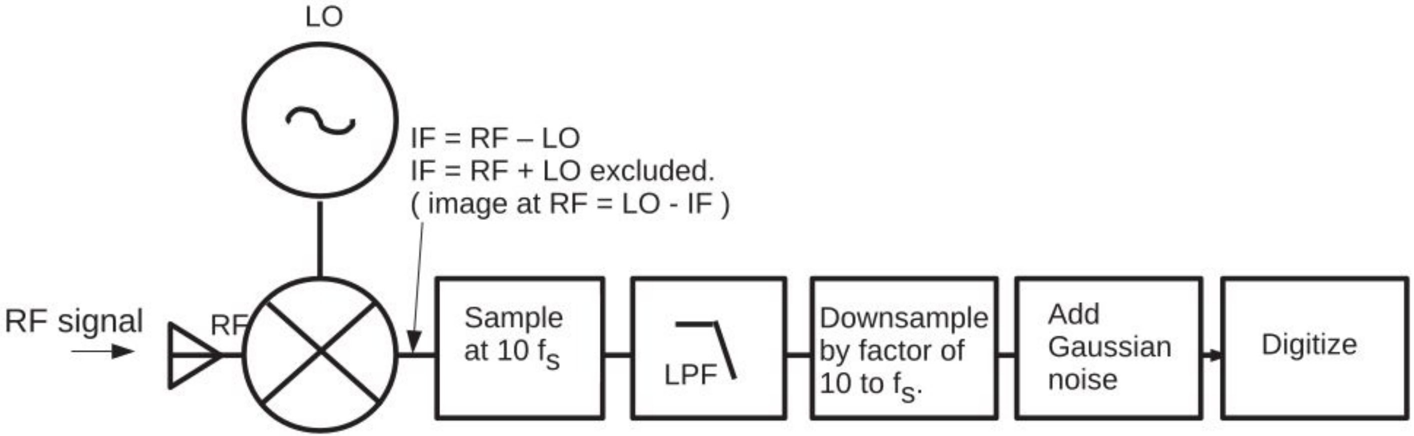
   ```
   docker run -p 8888:8888 -v ~/p8tutorial:/tmp project8/p8compute-jupyter
   ```

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

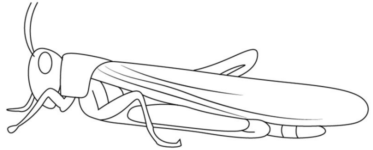
Locust* simulation software



- Developed in C++ within Project 8.
- Integrated with Project 8 DAQ libraries.
- Functionality models an RF receiver and digitizer.
- Interfaces are modular and flexible to allow for arbitrary input signals.

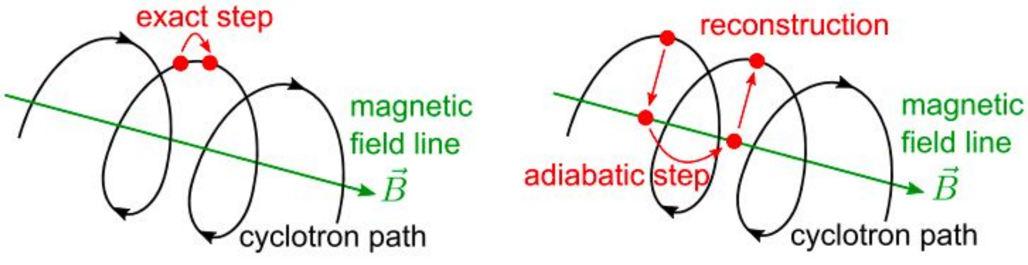


*Project 8, New J. Phys. 21 113051



Kassiopeia* simulation software

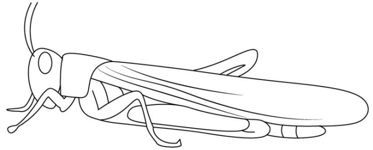
- Developed by the KATRIN collaboration.
- Highly advanced calculations of electron trajectories and energy losses in EM fields.
- Adaptable EM field solutions with configurable source geometries.
- Modular and flexible; C++ based code.



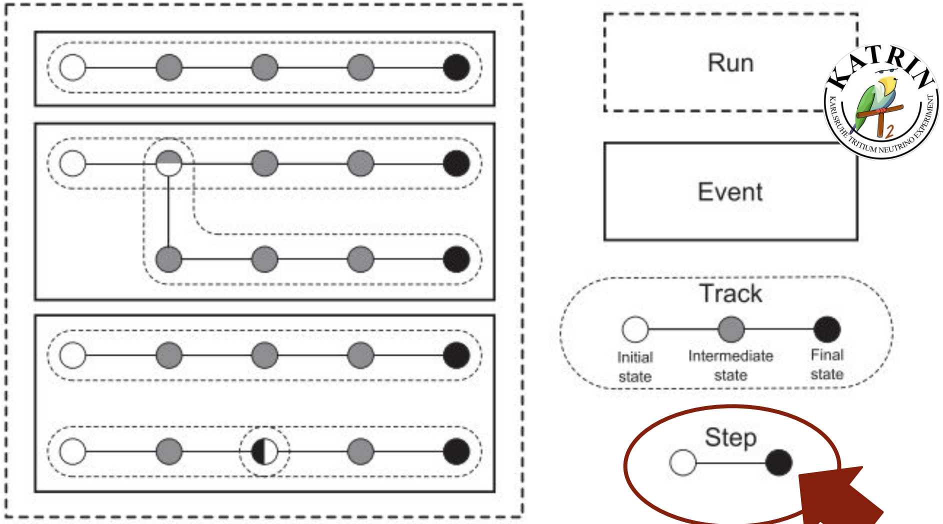
(a) Exact trajectory (b) Adiabatic trajectory

*Furse et al., 2017 New J. Phys. 19 053012

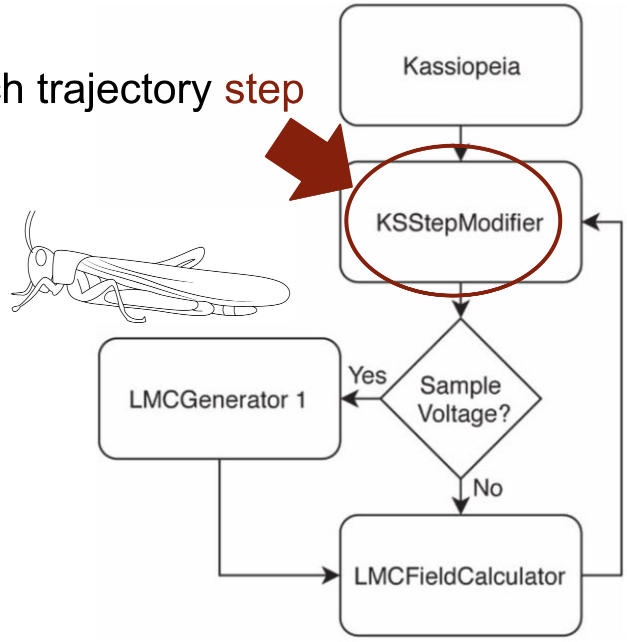
Locust-Kassiopeia interface



- Modularity in both Locust and Kassiopeia supports tight integration of Locust-Kassiopeia in the time domain.
- Cross-package communication happens after each trajectory step in Kassiopeia.



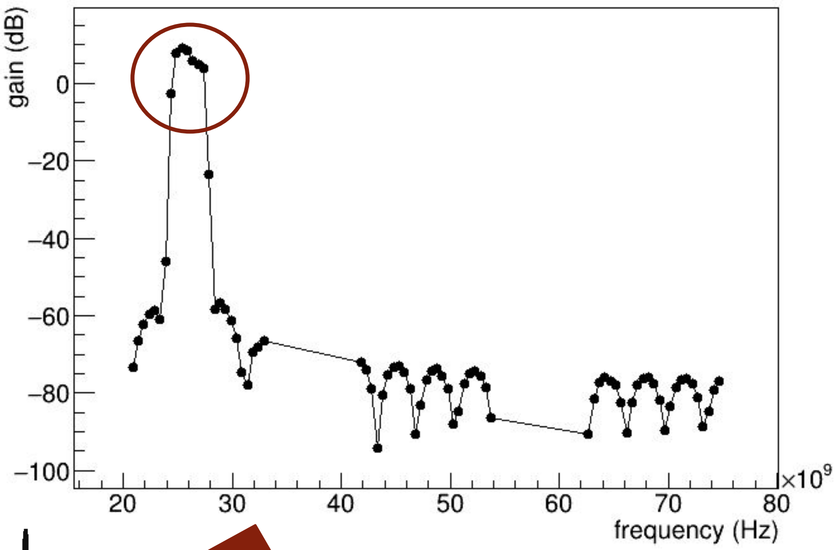
Furse et al., 2017 New J. Phys. 19 053012

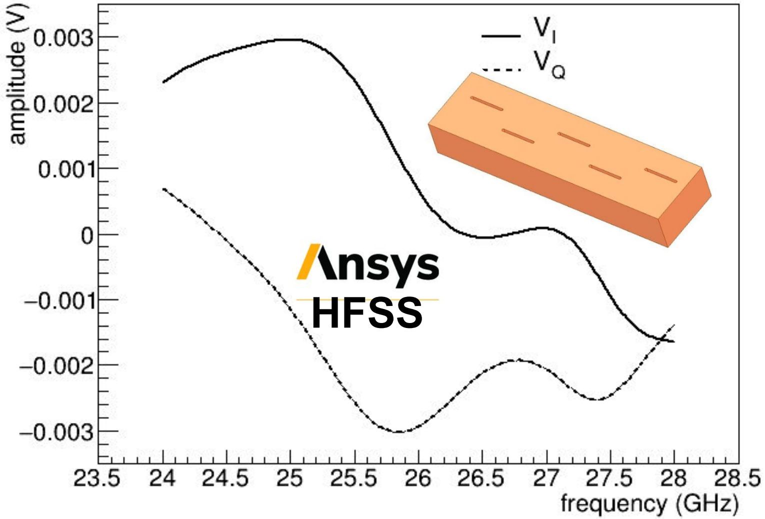


Project 8, 2019 New J. Phys. 21 113051

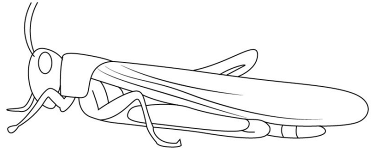
Locust-HFSS interface

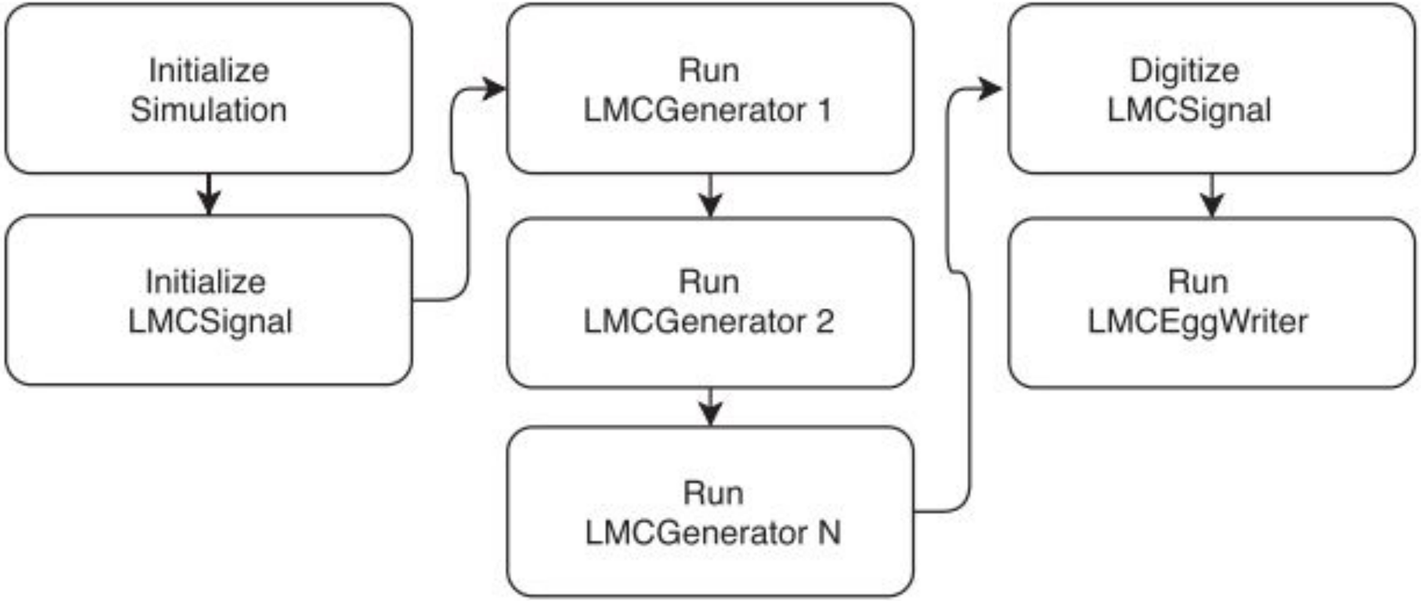
- Interface relies on Linear Time-Invariant (LTI) system theory.
- LTI antenna model is configured in HFSS.
- Locust EM fields drive the LTI antenna FIR.





Locust work flow diagram

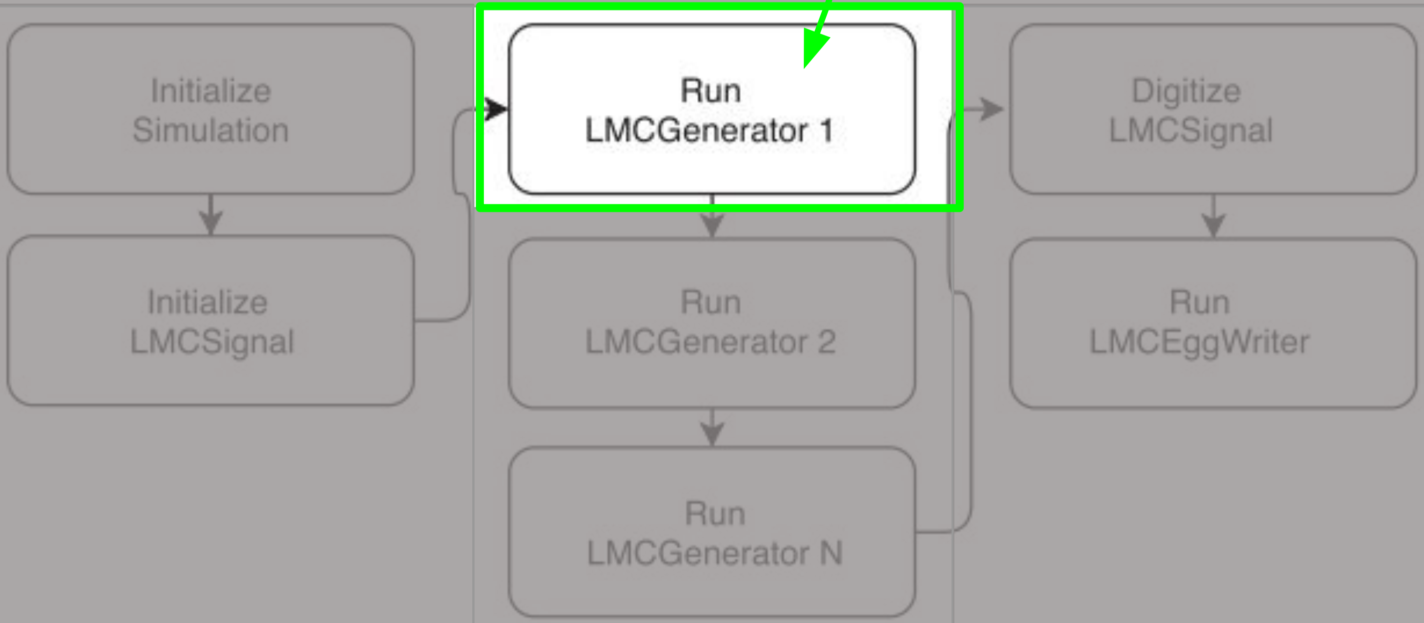




Project 8 collaboration, New J. Phys. 21 (2019) 113051

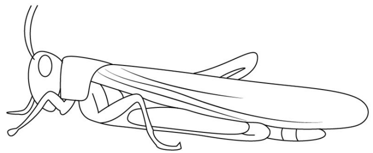
Locust work flow diagram

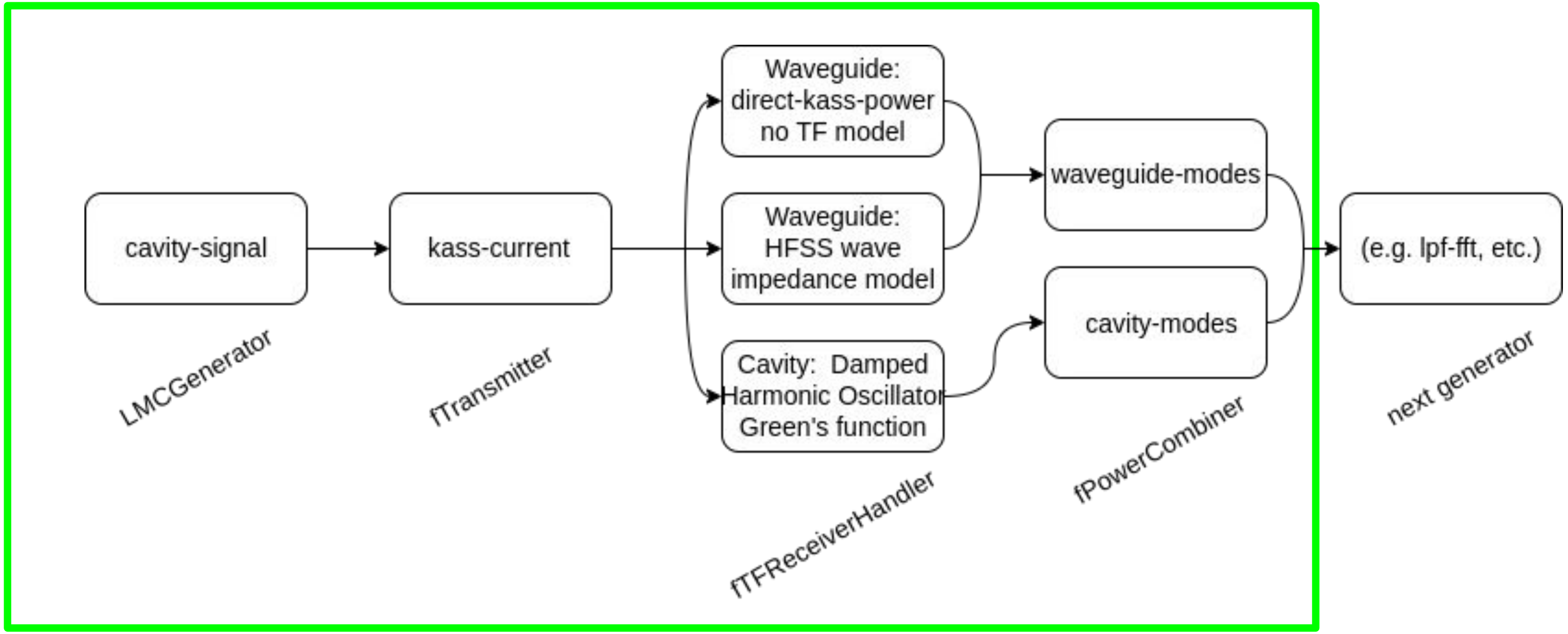
Cavity and waveguide calculations are handled here.



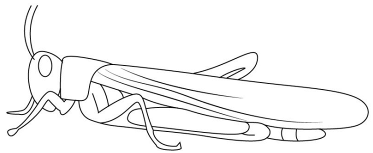
Project 8 collaboration, New J. Phys. 21 (2019) 113051

New generator ("cavity-signal") and its main configuration options:





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



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

Generator 1: Selected by application.

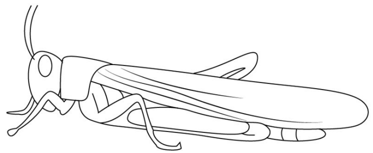
Generators 2+: Receiver chain

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

```
"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
      }

}
```

Running the tutorial examples



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

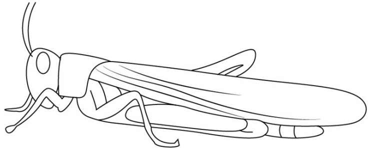
```
cd ~/p8tutorial
git clone git@github.com:project8/hexbug
git clone git@github.com:project8/locust-tutorial
```

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

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

3. Open tutorial_v2.5.1_Locustscript.sh and tutorialKatydidscript.sh with a text editor.

Example #1: Locust test signal plus noise

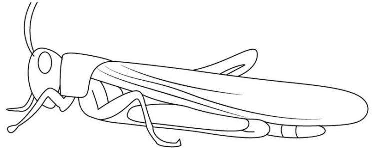


- Drive 50 ohm antenna with a sinusoid in LMCTestSignalGenerator:
    - (uncomment the command below #Example 1 in tutorialLocustscript_v2.5.1_.sh, then): `./tutorialLocustscript _v2.5.1_.sh`
    - (or interactively, in container): `LocustSim config=/path/to/config/LocustTestSignal.json`
    - Process egg file with `./tutorialKatydidscript.sh`

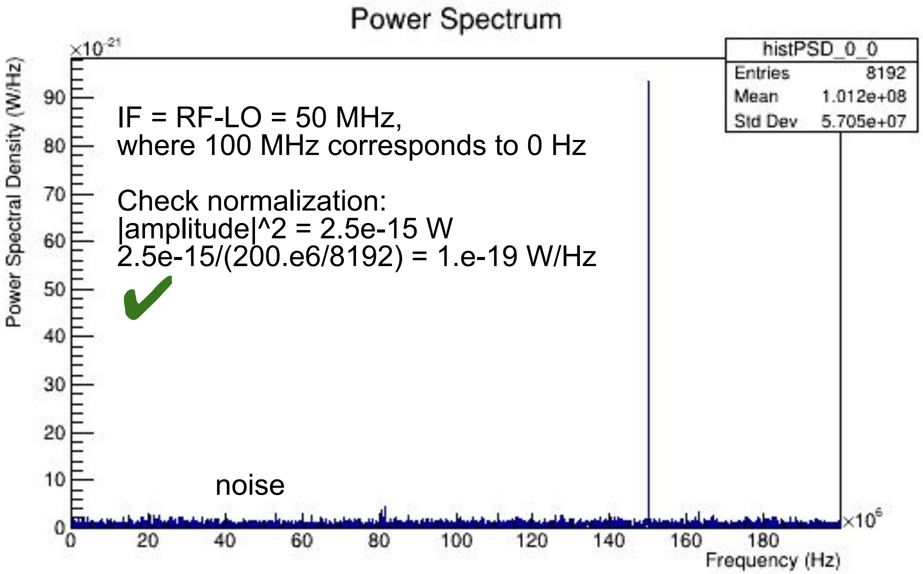
```
"test-signal":
{
"rf-frequency": 20.7e9,
"lo-frequency": 20.65e9,
"amplitude": 5.0e-8
}

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

This is in LocustSignalPlusNoise.json

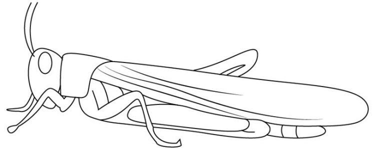
Example #1: Locust test signal plus noise, cont.



- Open the file output/basic.root and plot histPSD_0_0, as in ->
- Or, 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 -> .



Example #2: 25.9 GHz cavity example

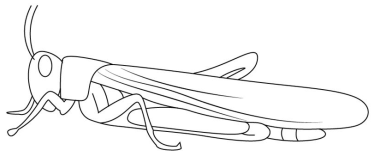


- Calculate an unnormalized signal from a single-mode resonant cavity, driven with a trapped Kassiopeia electron.
  - (uncomment the command below #Example 2 in tutorialLocustscript_v2.5.1_.sh, then):
    `./tutorialLocustscript _v2.5.1_.sh`
  - (or interactively, in container):
    `LocustSim config=/path/to/config/LocustCavityCCA.json`
  - Process egg file with `./tutorialKatydidscript.sh`

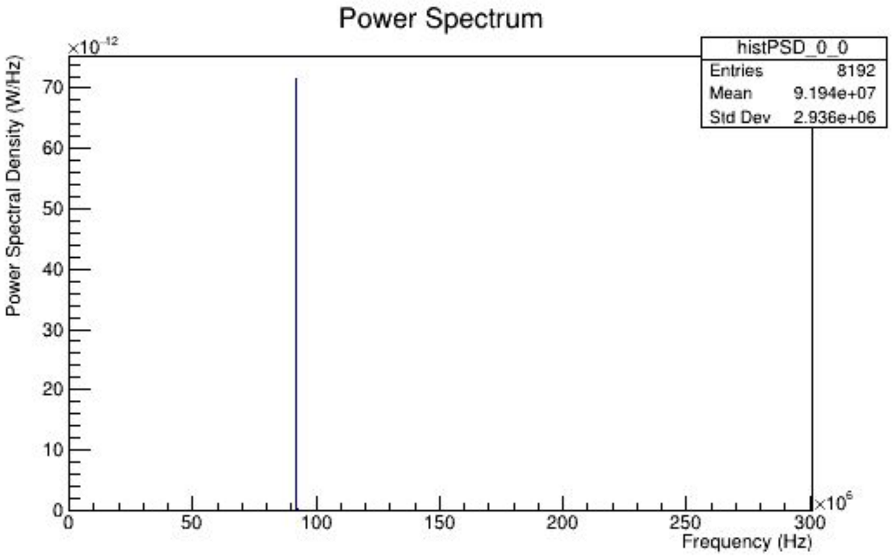
```
    "cavity-signal":
{
    "transmitter": "kass-current",
    "cavity-radius": 0.007,
    "cavity-length": 0.1,
    "back-reaction": "true",
    "dho-cavity-frequency": 25.9e9,
    "dho-time-resolution": 9.0e-11,
    "dho-threshold-factor": 0.01,
    "event-spacing-samples": 10,
    "rectangular-waveguide": false,
    "voltage-check": "false",
    "lo-frequency": 25.9602e9,
    "xml-filename": "/home/penny/locust_mc/cbuild/config/LocustKass_Cavity_CCA.xml"
},
```

This is in LocustCavityCCA.json

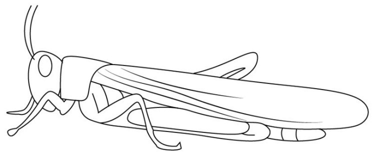
Example #2: 25.9 GHz cavity example, cont.



- Open the file output/basic.root and plot histPSD_0_0, as in ->
- Or, 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 -> .



Example #3: WR42 waveguide



- Calculate a normalized signal from a TE10 mode WR42 waveguide, driven with a trapped Kassiopeia electron.
  - (uncomment the command below #Example 3 in tutorialLocustscript_v2.5.1_.sh, then): `./tutorialLocustscript_v2.5.1_.sh`
  - (or interactively, in container): `LocustSim config=/path/to/config/LocustWaveguideTemplate.json`
  - Process egg file with `./tutorialKatydidscript.sh`

```
 "cavity-signal":
{
  "rectangular-waveguide": true,
  "direct-kass-power": false,
  "tf-receiver-filename": "/path/to/data/WEGA_Impedance_Center.txt",
  "tf-receiver-bin-width": 0.01e9,
  "transmitter": "kass-current",
  "waveguide-x": 0.010668,
  "waveguide-y": 0.004318,
  "waveguide-z": 10.0,
  "center-to-short": 0.05,
  "center-to-antenna": 0.05,
  "waveguide-central-frequency": 1.63e11,
  "back-reaction": "true",
  "event-spacing-samples": 10,
  "voltage-check": "false",
  "lo-frequency": 25.9602e9,
  "xml-filename": "/home/penny/locust_mc/cbuild/config/LocustKass_Waveguide_Template.xml"
},
```

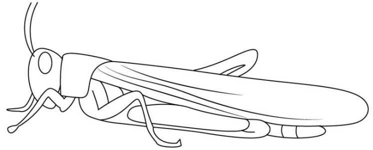
Set “direct-kass-power”=true to skip convolution with HFSS impedance.

Default: Convolve Kass current with complex impedance from HFSS

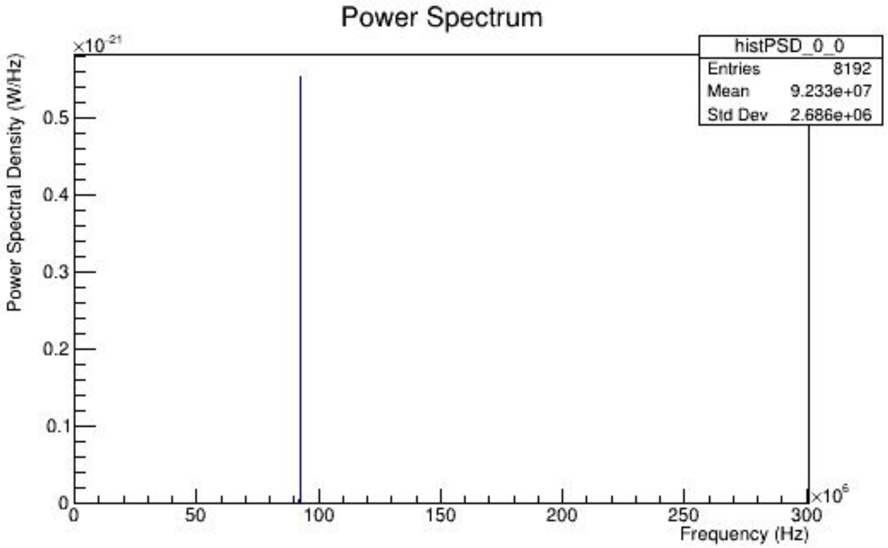
reflecting short

This is in LocustWaveguideTemplate.json

Example #3: WR42 waveguide, cont.



- Open the file output/basic.root and plot histPSD_0_0, as in ->
- Or, 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 -> .



Auxiliary plotting utilities (1)

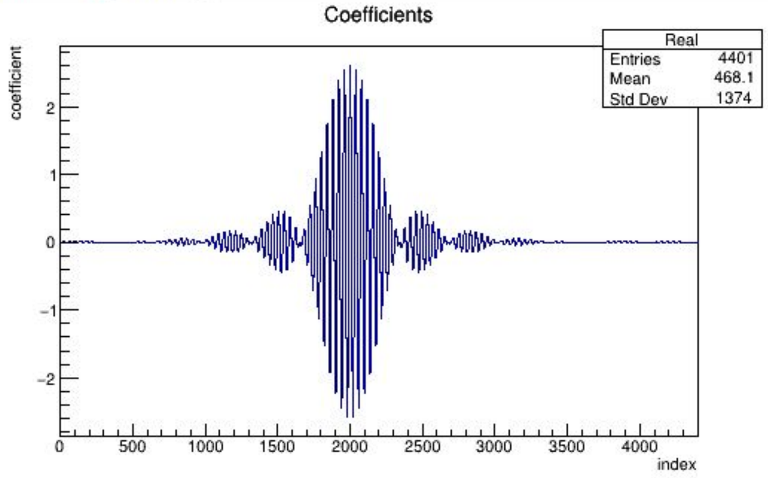
For checking the HFSS inputs and calculations, set the flag print-fir-debug=true, as in e.g.

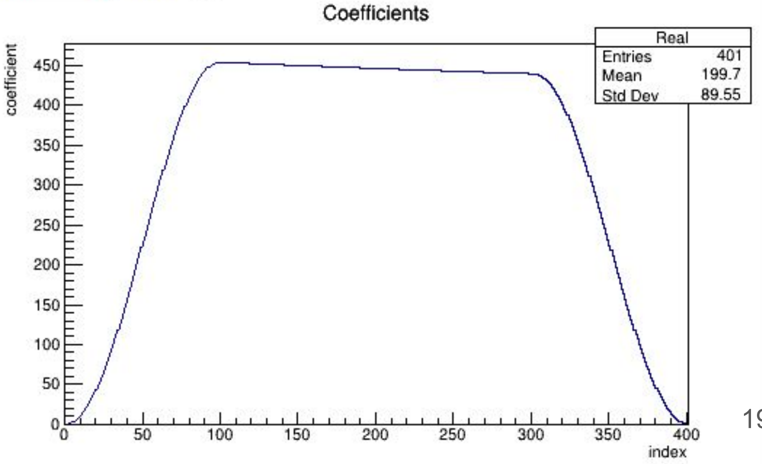
bin/LocustSim -c config/LocustWaveguideTemplate.json "cavity-signal.print-fir-debug"=true

Follow instructions in terminal:

2023-08-24 22:32:44 [PROG] (tid 140278570523584) seFileHandler.cc(191): Finished writing histos to output/FIRhisto.root and output/TFhisto.root

2023-08-24 22:32:44 [PROG] (tid 140278570523584) seFileHandler.cc(192): Press Return to continue, or Cntrl-C to quit.





Auxiliary plotting utilities (2)

Similarly, for checking the cavity resonant Green's function, print-fir-debug=true, as in e.g.

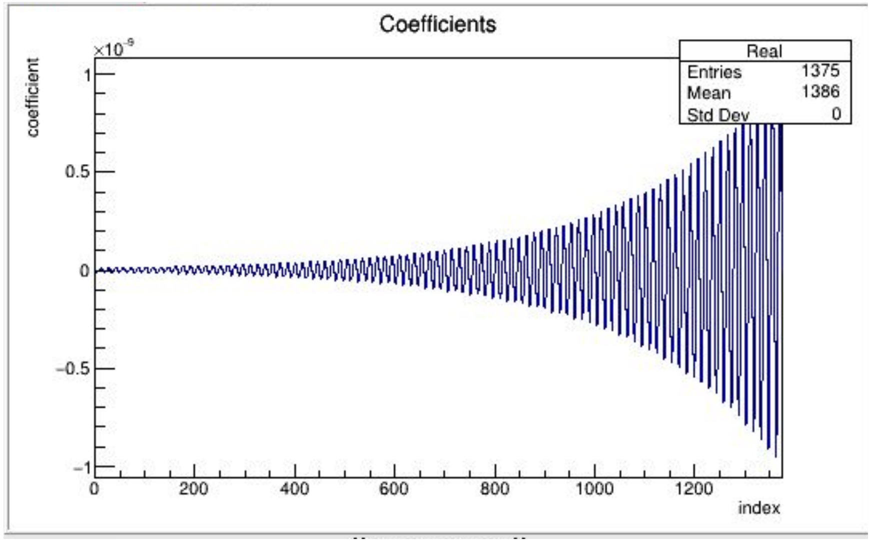
bin/LocustSim -c config/LocustCavity1GHz.json "cavity-signal.print-fir-debug"=true

Follow instructions in terminal:

2023-08-24 22:32:44 [PROG] (tid 140278570523584) seFileHandler.cc(191): Finished writing histos to output/FIRhisto.root

2023-08-24 22:32:44 [PROG] (tid 140278570523584) seFileHandler.cc(192): Press Return to continue, or Cntrl-C to quit.

For additional checks of the cavity resonance, use the compiled unit test bin/testLMCCavity.



Auxiliary plotting utilities (3)

Similarly, for plotting mode maps, use cavity-signal.plot-mode-maps=true, as in e.g.

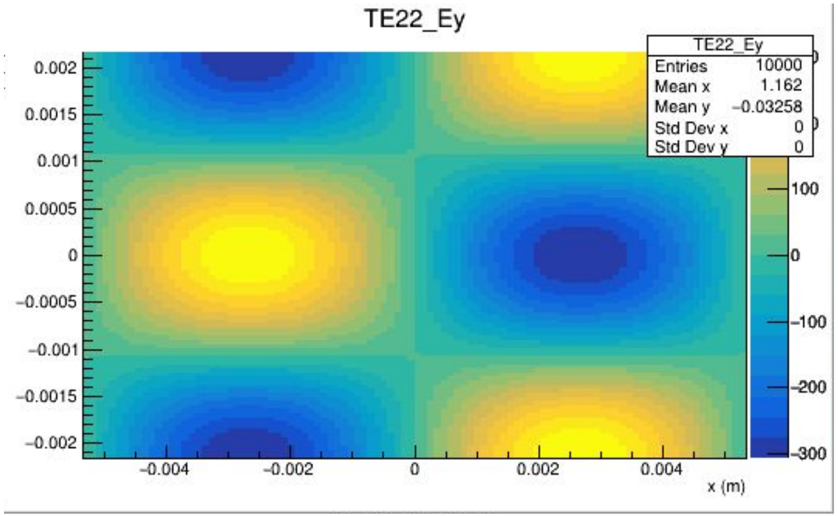
```
bin/LocustSim -c config/LocustWaveguideTemplate.json "cavity-signal.plot-mode-maps"=true
```

Or, to include more modes for plotting:

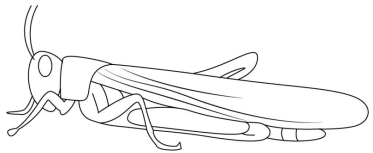
```
bin/LocustSim -c config/LocustWaveguideTemplate.json "cavity-signal.plot-mode-maps"=true “cavity-signal.n-modes”=3
```

Follow instructions in terminal:

```
2023-08-24 22:32:44 [ PROG] (tid 140278570523584) seFileHandler.cc(191): Finished writing histos to output/FIRhisto.root

2023-08-24 22:32:44 [ PROG] (tid 140278570523584) seFileHandler.cc(192): Press Return to continue, or Cntrl-C to quit.
```



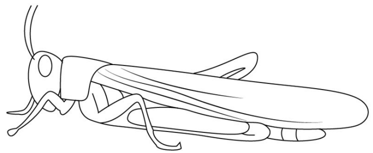
Some helpful unit tests



bin/testLMCCavity [-h]: Check the cavity resonance configuration, and optionally plot the output to a histogram in a Root file. This test is also run automatically by the LMCCavitySignalGenerator.

bin/testAliasing [-h]: Check the frequencies of RF harmonics including n=0,1,2, and determine whether they will appear in the measurement window. This test is also run automatically by the LMCCavitySignalGenerator.

How to obtain Locust and run some examples



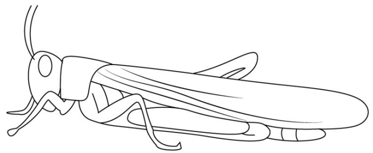
1. Start the container: docker run -it –rm ghcr.io/project8/locust_mc:latest /bin/bash
2. And/or, to mount a local directory to the docker output directory:
   (all one line): docker run -v /path/to/mydirectory:/usr/local/p8/locust/current/output -it –rm ghcr.io/project8/locust_mc:latest /bin/bash

Inside the container:

1. source /usr/local/p8/locust/current/setup.sh
2. source /usr/local/p8/kassiopeia/current/bin/kasperenv.sh
3. cd /usr/local/p8/locust/current/
4. LocustSim -c config/LocustCavity1GHz.json (1 GHz cavity example)
   or
   LocustSim -c config/LocustCavityCCA.json (25.9 GHz cavity example)
   or
   LocustSim -c config/LocustWaveguideTemplate.json (WR42 waveguide example)

Output *.egg files will be in the directory /usr/local/p8/locust/current/output .

Hexbug repo: Config files and cluster scripts



Hexbug is a private repository internal to Project 8.

On an hpc cluster that runs Slurm for job management, clone hexbug as git clone git@github.com:project8/hexbug.git .

cd hexbug

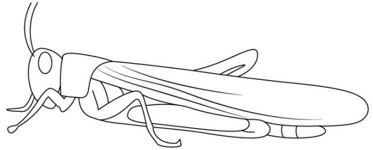
Edit clusterScripts/clusterCavity.cfg to configure outputdir

cd clusterScripts/[choiceOfDemonstrator]

python3 GenerateCavitySims.py or python3 GenerateCavitySims_dSQ.py (the latter is preferred).

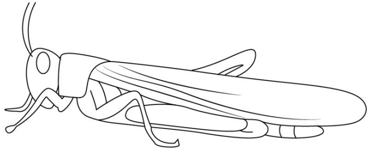
Follow terminal instructions. Edit scripts as needed to parametrize the job array.

Steps to extend Locust-Kass



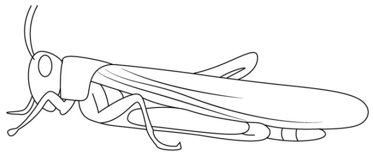
1. Develop a new unique LMCGenerator class and type its name in all files where other LMCGenerators are listed. For example, P8 cavity and waveguide signal simulations are run with the LMCCavitySignalGenerator.
2. Develop or select an LMCTransmitter to transmit Kass info to the Locust signal classes, e.g.:
    a. LMCKassCurrentTransmitter: Kass electron kinematics (Presently used in P8 cavity/waveguide simulations.
    b. LMCKassTransmitter: LW free-space field solutions.
3. After the 1st LMCGenerator as in #1, other classes are used for low-pass filtering, digitization, and optionally, broadband white noise. See the example json files.

How to develop (Option #1)



Option #1: (Re)build a local Docker container for each modification.

1. git clone git@github.com:project8/locust_mc.git
2. git submodule update –init –recursive
3. Make your code modifications etc.
4. docker build -t mylocustcontainer .
5. Run your new docker container as in previous slide.
6. Evaluate, return to Step #3.



How to develop (Option #2)

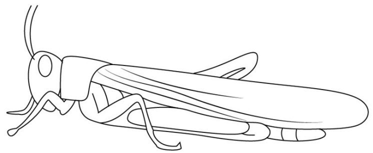
Option #2: Build from source in the kassiopeia_builder container.

First, clone locust and submodules as in Option #1.

1. docker pull ghcr.io/project8/kassiopeia_builder:v3.8.2-dev
2. docker run -it -v /path/to/your/locust_mc:/locust_mc --rm ghcr.io/project8/kassiopeia_builder:v3.8.2-dev /bin/bash
3. source /usr/local/p8/kassiopeia/current/setup.sh
4. source /usr/local/p8/kassiopeia/current/bin/kasperenv.sh
5. cd /locust_mc
6. mkdir cbuild
7. cmake -Dlocust_mc_PREBUILT_KASS_PREFIX=${KASS_PREFIX} -Dlocust_mc_BUILD_WITH_KASSIOPEIA=ON ../
8. make install
9. Make code modifications locally, then make install
10. Run LocustSim in the kassiopeia_builder container, evaluate, return to Step #8.

You can close the container and reopen as above, then continue working.

How to compile from source (not typically recommended for new users)



Check system requirements at https://github.com:project8/locust_mc.git

1. git clone git@github.com:project8/locust_mc.git
2. cd locust
3. git submodule update –init –recursive
4. mkdir build
5. cd build

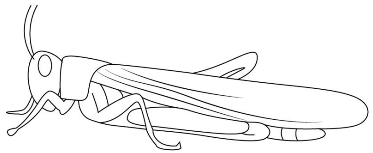
6. ccmake ../

   or

7. ccmake -Dlocust_mc_BUILD_WITH_KASSIOPEIA=ON ../

8. make install

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.