LAC 2026 · LINUX AUDIO CONFERENCE

In [1]: from measurement import Measurement

A Python-Based Interactive Acoustic Impulse Response Measurement Tool

Measure a speaker, room, or effect — from an IPython prompt, on anything JACK or PipeWire can see.

Martin Pollow • built on zita-jacktools (Fons Adriaensen)

Why another measurement tool?

- Existing tools are **GUI-bound**, or batch scripts with **hardcoded ALSA or JACK device strings**.
- The painful part is the **plumbing**: which devices, ports, channels? Did the signal actually arrive?
- **Interactive-first** (IPython / Jupyter), FLOSS, native **JACK and PipeWire**, multiple sound cards in one measurement.
- Use **JACK native zita-jacktools**, add flexibility for **PipeWire** — and experiment with an **LLM as a coding agent** for the implementation.

The tool in one slide

```
m = Measurement(…)  →  m.run()  →  m.ir   # NumPy array
```

LOG-SWEEP EXCITATION

Configurable frequency range, duration, level.

LATENCY CALIBRATION

Sample-accurate, via electrical loopback.

PER-CHANNEL CALIBRATION

Sensitivity-normalized, one sweep, many mics.

DEVICE BROWSER

TAB-completable `output_to` / `input_from`.

PLOTS & OVERLAYS

Time + frequency, compare measurements.

PERSISTENCE

Audio as WAVEX · CAF · W64 · raw NumPy,
meta data as a JSON file.

Acoustic measurements in a nutshell

x(t)
measurement signal

play →

SYSTEM
room · speaker · effect

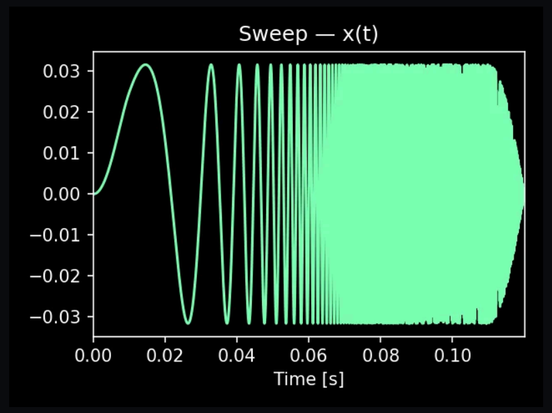
record →

y(t)
recording

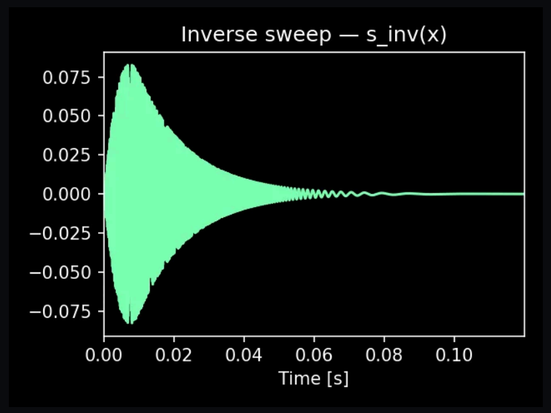
⊛ $x_{inv}(x)$ →
deconvolution

h(t)
impulse response

SWEEP – x(t)



INVERSE FILTER – $x_{inv}(x)$



DEMO

$ switch to the terminal

① ROOM	laptop speaker	—air→	built-in mic array	
② SOUND CARDS	UCA222 out	—cable→	UCA222 in	latency · cross-talk
	laptop headphone out	—cable→	UCA222 in	measurement over different sound cards
③ PLUGIN · REAPER	sweep	→	REAPER plugin	frequency response

06 / PROCESS

Built with an LLM in the loop

HUMAN

DSP core · deconvolution
JACK backend & routing
Acoustic-domain decisions
Fons' sweep generation code and low-level audio IO

LLM (Claude Code)

TAB-completable device namespace
Multi-port selection
Plot overlays · file formats
Display formatting & repr

transcript excerpt — claude · CLI

```
›
make m.output_to tab-complete the live JACK ports

I'll expose detected ports as attributes on a
namespace object, so m.output_to.<TAB> resolves
against what's connected right now, with glob
and prefix matching for multi-port selection…
```

$ Runtime needs **no** LLM — it's plain Python.

Limitations

EXPERIMENTAL	Measurement and latency compensation are **stable on a (good) single sound card**. Using **multiple cards** is experimental and may introduce artefacts (PipeWire resamples adaptively).
FEATURES	Not a full measurement suite, but a **flexible toolkit**. No explicit linearity check, no smoothing, no averaging, no further analysis. Just the raw measurements with discoverable JACK ports.

08 / GET IT

Get it on github, if you find it useful...

Thank you!
Any questions?

clone from github

connect on LinkedIn

REPOSITORY

```
$ git clone https://github.com/mpollow/measurement.git
```

LICENSE

GPLv3

CONTACT

Martin Pollow

linkedin.com/in/martin-pollow

APPENDIX

Audio-failure parachute

Rehearsal screenshots at presentation font size — one slide per demo step, grouped into the three setups.

① ROOM	A1 repr · A2 tab · no cards · A3 pt · pf
② SOUND CARD	A4 hotplug · A5 set out · in · A6 latency · A7 cross-talk · A8 cross-card
③ PLUGIN	A9 insert

Cold open — `repr(m)`

```
In [1]: from measurement import Measurement

In [2]: m = Measurement()

In [3]: m
Out[3]:
Measurement(
  not measured
  output: not configured
  input: not configured
  samplingrate: 48000 Hz
  f_min: 20.0 Hz
  f_max: 20000.0 Hz
  t_sweep: 1.0 s
  output_amplification: -30 dBFS
  latency: 0 samples (0.0 ms)
  ir_length: 10000 samples
)

In [4]:
```

The measurement object prints its full config — devices, sweep parameters, calibration — before anything runs.

TAB completion — no external cards

`m.output_to`

```
In [1]: from measurement import Measurement

In [2]: m = Measurement()

In [3]: m.output = m.output_to.
                               Laptop_Speakers
                               REAPER
```

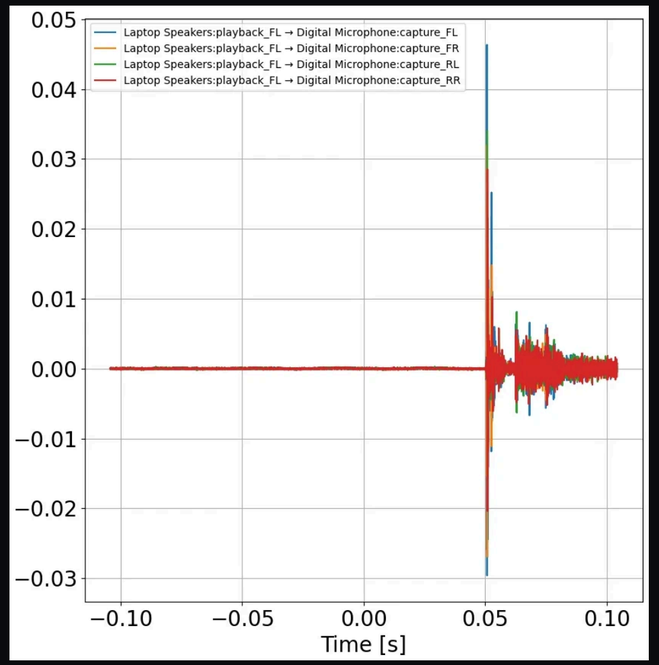
`m.input_from`

```
In [1]: from measurement import Measurement

In [2]: m = Measurement()

In [3]: m.output = m.output_to.Laptop_Speakers.playback_FL

In [4]: m.input = m.input_from.
                               Digital_Microphone           Laptop_Speakers
                               Headphones_Stereo_Microphone REAPER
```

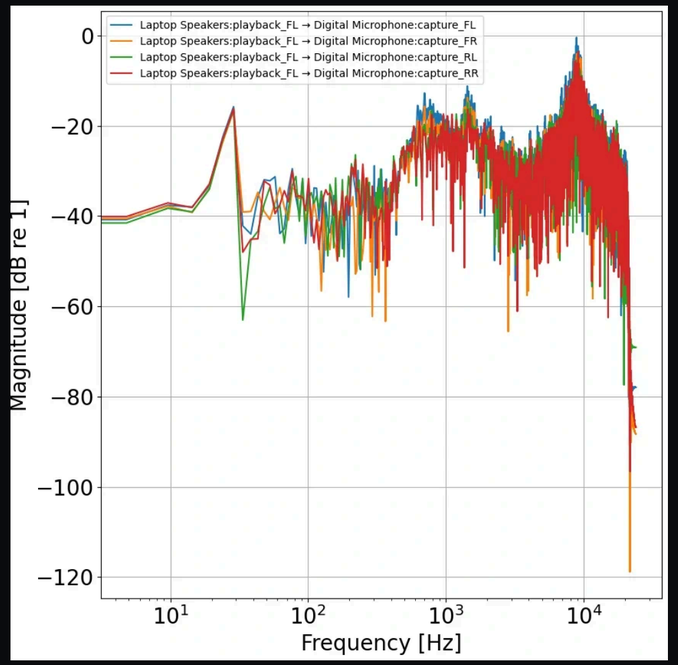
Give only the **node** for the input and it selects **all its channels** — selecting **Digital Microphone** captures all four channels.

Laptop measurement — pt + pf, annotated

pt — TIME DOMAIN



pf — FREQUENCY DOMAIN



Annotate: direct sound · reflections · noise floor · LF rolloff · comb ripple.

m.output_to — before / after UCA222 hotplug

BEFORE

```
In [1]: from measurement import Measurement

In [2]: m = Measurement()

In [3]: m.output = m.output_to.
                               Laptop_Speakers
                               REAPER
```

AFTER

```
In [1]: from measurement import Measurement

In [2]: m = Measurement()

In [3]: m.output = m.output_to.
                               Laptop_Speakers
                               REAPER
                               UCA222
```

TAB-completion picks up the UCA222 the moment it is plugged in — no restart, no rescan.

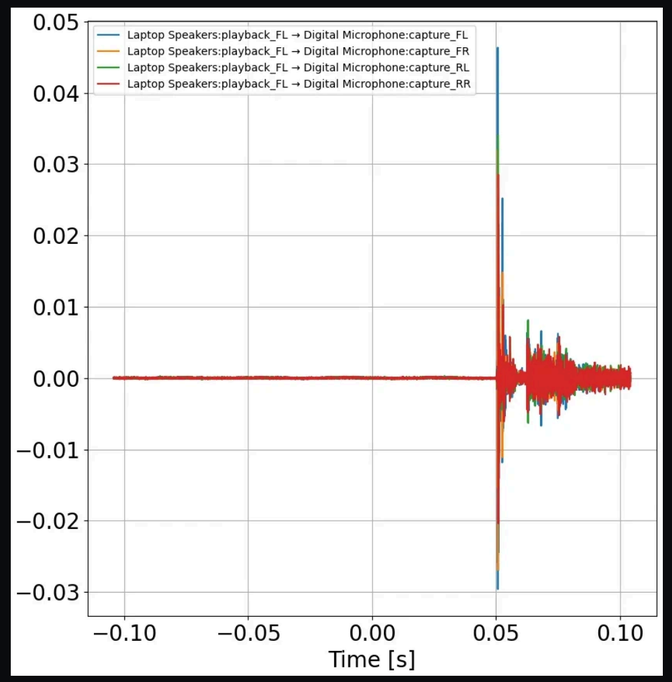
Set output & input — `run()` with per-channel dBFS

```
In [1]: from measurement import Measurement

In [2]: m = Measurement()

In [3]: m.output = m.output_to.Laptop_Speakers.playback_FL

In [4]: m.input = m.input_from.Digital_Microphone

In [5]: m.output_amplification = -10

In [6]: m.run()
Sweep: 20.0-20000.0 Hz, 1.00 s, -10 dBFS
Route: Laptop Speakers:playback_FL → Digital Microphone:capture_FL, Digital Microphone:capture_FR, Digital Microphone:c
apture_RL, Digital Microphone:capture_RR
  Digital Microphone:capture_FL: -12.0 dBFS
  Digital Microphone:capture_FR: -15.0 dBFS
  Digital Microphone:capture_RL: -16.9 dBFS
  Digital Microphone:capture_RR: -16.1 dBFS

In [7]:
```

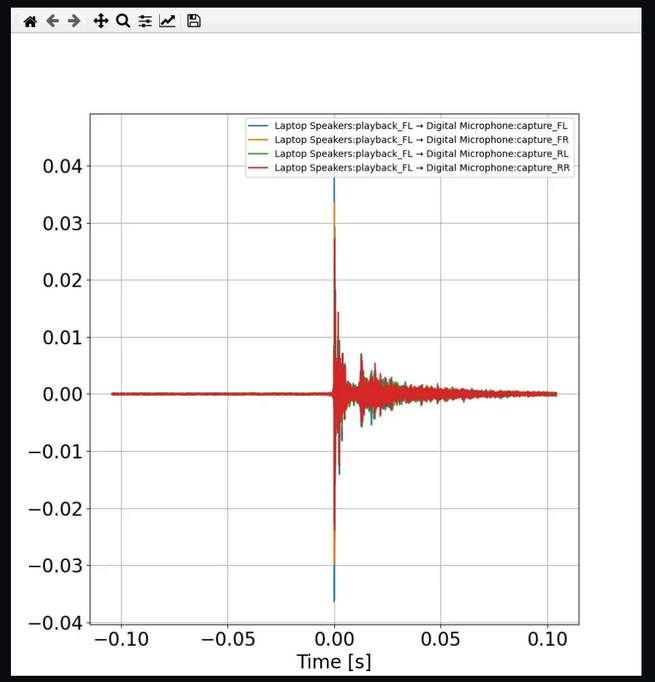
`m.input` resolves by implicit **prefix matching**; per-channel peak dBFS confirms signal arrived.

Latency — `pt` before / after `run_latency()`

BEFORE — peak offset



AFTER — peak at t = 0



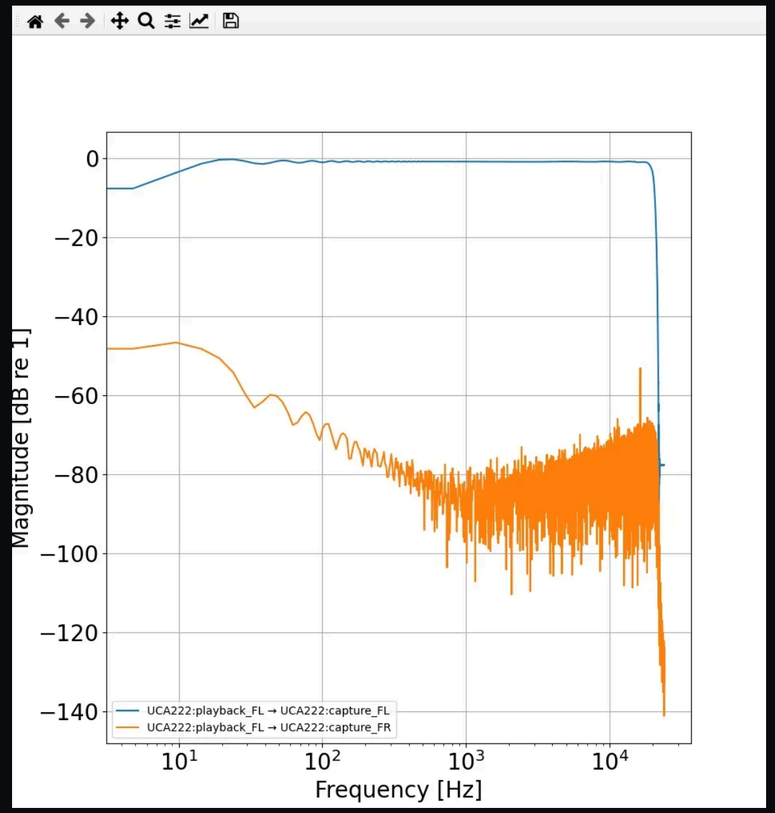
Run the loopback calibration and the peak snaps to t = 0 — round-trip delay compensated to the sample.

Inter-channel cross-talk

SETUP

```
In [2]: m = Measurement()

In [3]: m.output = m.output_to.UCA222.playback_FL

In [4]: m.input = m.input_from.UCA222.capture

In [5]: m.output_amplification = -10

In [6]: m.f_min = 1

In [7]: m
Out[7]:
Measurement(
  not measured
  output: ['UCA222:playback_FL']
  input: ['UCA222:capture_FL', 'UCA222:capture_FR']
  samplingrate: 48000 Hz
  f_min: 1 Hz
  f_max: 20000.0 Hz
  t_sweep: 1.0 s
  output_amplification: -10 dBFS
  latency: 0 samples (0.0 ms)
  ir_length: 10000 samples
)

In [8]:
```

RESULT



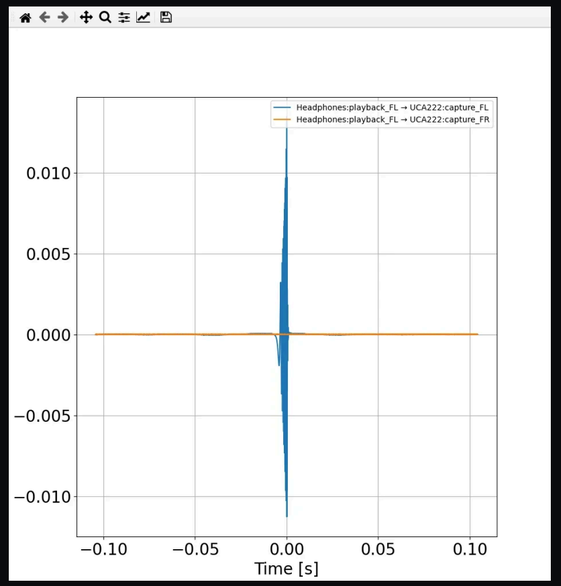
Measure across channels and the leakage floor between them shows the converter and routing cross-talk.

Measurement across different sound cards

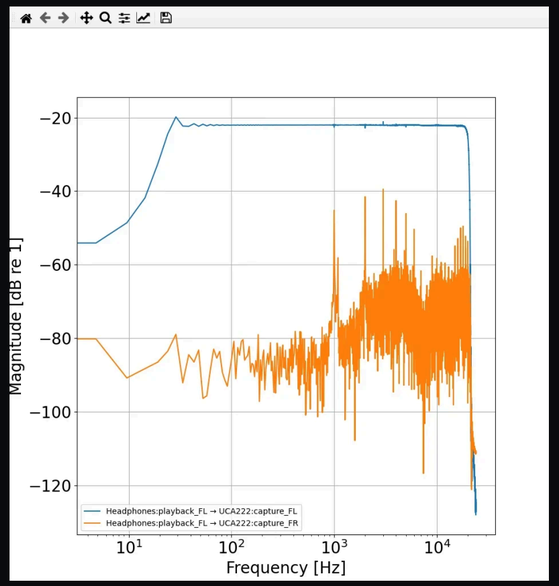
SIGNAL PATH | 'Laptop Headphone' —cable→ 'UCA222 input'

two independent clocks · PipeWire resamples adaptively

pt — TIME DOMAIN



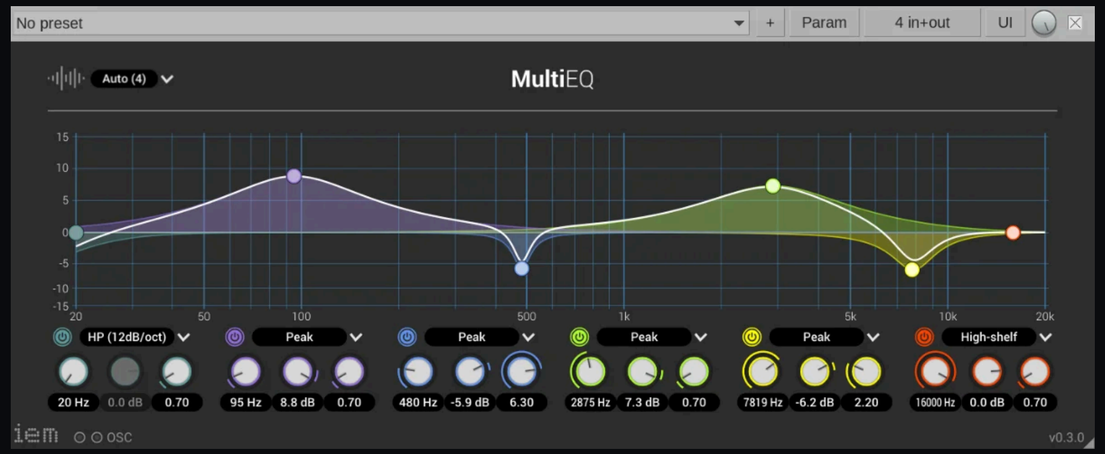
pf — FREQUENCY DOMAIN



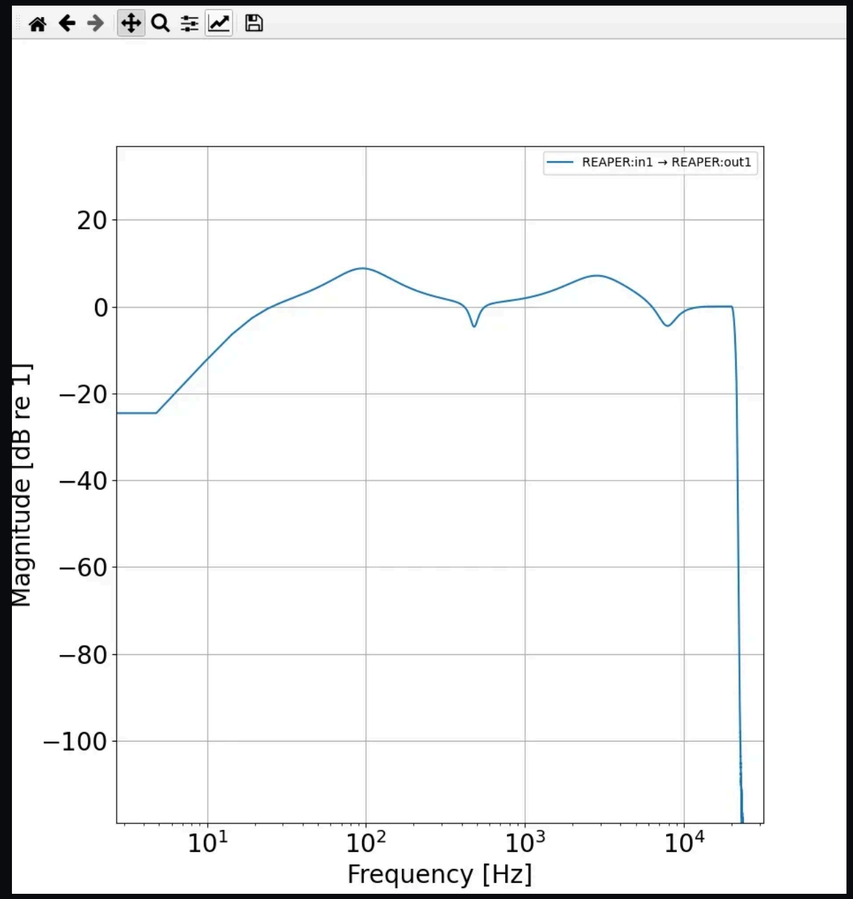
Bad result: smeared IR and resonant peaks in the frequency response.

REAPER — insert a plugin in the path

REAPER PLUGIN



MEASURED TRANSFER PATH



Open REAPER and insert a plugin — here an EQ — into the measured signal path.