Technology A Relaxation

August 17, 2021

1 RRAM Relaxation Data Notebook

This notebook contains the analysis on empirical RRAM relaxation data across three technologies (A, B, C). It loads and processes the measurements taken for each technology.

[1]:
```
# Imports
import gzip
import json
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.signal
import scipy.stats
# from matplotlib.offsetbox import AnchoredText

%config InlineBackend.figure_format = 'svg'
```

1.1 Load the technology and its settings

Below, choose which technology to load data and settings for:

[2]:
```
# Choose technology here
TECH = 'A'

# Load settings for technology
with open(f"data/tech{TECH}/settings.json") as sfile:
    settings = json.load(sfile)
```

1.2 Time series analysis

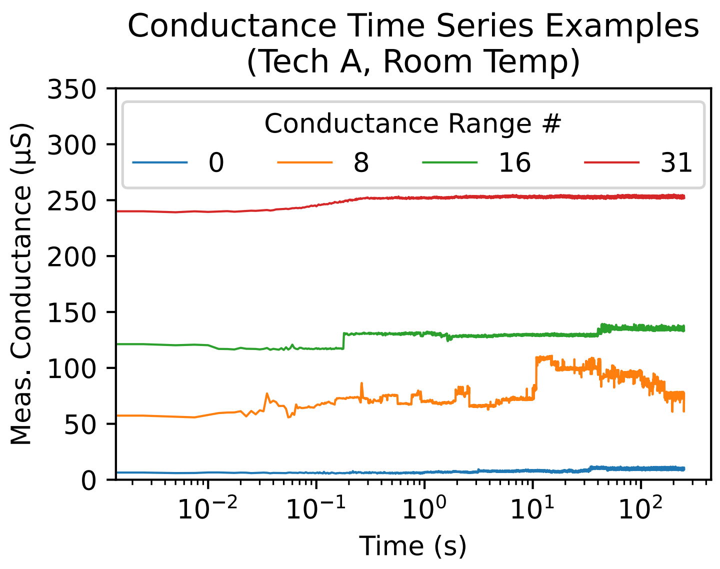
In this section, we will look at example time series data on a log scale and also examine the power spectral density (PSD). First, let us load the time series data:

1.2.1 Example Time Series Data

Below, we can look at the time series data for the ranges chosen above:

[3]:
```
# Load data for technology
with gzip.open(f"data/tech{TECH}/tsdata.min.tsv.gz", "rt") as datafile:
    # Plot example time series data
    fig = plt.figure(figsize=(4,2.7))
    ax = fig.add_subplot(111)
    ax.set_title(f"Conductance Time Series Examples\n(Tech {TECH}, Room Temp)")
    for r in settings["ts_ranges"]:
        # Select data
        d = datafile.readline()
        print(r, "step 1")
        d = np.fromstring(d, dtype=float, sep='\t')[2:]
        print(r, "step 2")

        # Plot time series data
        plt.plot(np.arange(len(d))/settings["fs"], d*1e6, label=r, linewidth=0.
↪8)

    # Format and display
    ax.legend(title="Conductance Range #", ncol=4, loc=9)
    ax.set_ylim(*settings["ts_ylim"])
    ax.set_xlabel("Time (s)")
    ax.set_ylabel("Meas. Conductance (µS)")
    ax.set_xscale("log")
    plt.savefig(f"figs/tech{TECH}/time-series.png", dpi=300,␣
↪bbox_inches="tight")
    plt.show()
```

```
0 step 1
0 step 2
8 step 1
8 step 2
16 step 1
16 step 2
31 step 1
31 step 2
```

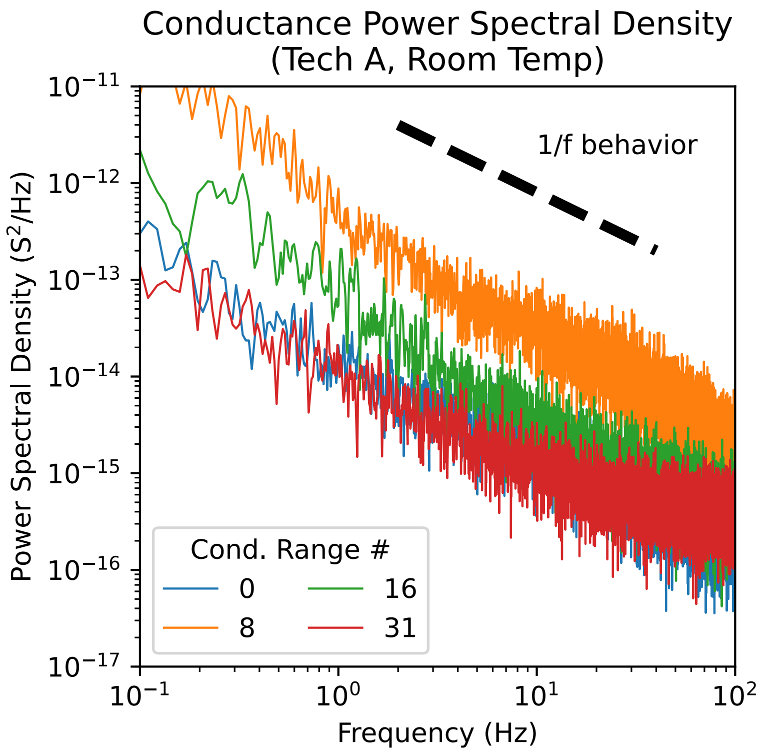


1.2.2 Power Spectral Density (PSD)

In this section, we will look at the PSDs to understand the relaxation behavior better:

```
[4]: # Load data for technology
with gzip.open(f"data/tech{TECH}/tsdata.min.tsv.gz", "rt") as datafile:
    # Plot power spectral density (PSD)
    fig = plt.figure(figsize=(4,4))
    ax = fig.add_subplot(111)
    ax.set_title(f"Conductance Power Spectral Density\n(Tech {TECH}, Room␣
 ↪Temp)")
    slopes = []
    for i, r in enumerate(settings["ts_ranges"]):
        # Select data
        d = datafile.readline()
        print(r, "step 1")
        d = np.fromstring(d, dtype=float, sep='\t')[2:]
        print(r, "step 2")

        # # Lomb-Scargle PSD
        # f = np.logspace(np.log10(1/600), np.log10(2), 500)
        # f = np.logspace(np.log10(2), np.log10(400/2), 1000)
        # p = scipy.signal.lombscargle(d["time"], d["g"], f)

        # Welch PSD
        f, p = scipy.signal.welch(d, fs=settings["fs"],␣
 ↪nperseg=settings["psd_nperseg"])
        plt.plot(f, p, label=r, linewidth=0.8)
        # # Power law fit
        # a, b = np.polyfit(np.log(f[[30,-30]]), np.log(p[[30,-30]]), 1)
        # print(f"Range {r} slope: {a}")
        # slopes.append(a)
        # plt.plot(f[1:], np.exp(a*np.log(f[1:]) + b), 'k--', zorder=10,␣
↪linewidth=2)

        # # 1/f fit
        # fitfn = lambda logf, A: A - logf
        # params, _ = scipy.optimize.curve_fit(fitfn, np.log(f), np.log(p))
        # A = params[0]
        # print(A)
        # plt.plot(f, np.exp(fitfn(np.log(f), A)), 'k--', zorder=10,␣
↪linewidth=2)

    # Format and display
    # ax.add_artist(AnchoredText("Slopes: [%.2f, %.2f]" % (max(slopes),␣
↪min(slopes)), loc=1, frameon=False))
    plt.plot(*settings["psd_fpts"], 'k--', zorder=10, linewidth=4)
    if "psd_f2pts" in settings:
        plt.plot(*settings["psd_f2pts"], 'k--', zorder=10, linewidth=4)
    ax.legend(title="Cond. Range #", ncol=2, loc=3)
    ax.set_xlabel("Frequency (Hz)")
    ax.set_ylabel("Power Spectral Density (S$^2$/Hz)")
    ax.set_xscale("log")
    ax.set_xlim(*settings["psd_xlim"])
    ax.set_ylim(*settings["psd_ylim"])
    ax.set_yscale("log")
    plt.text(*settings["psd_ftextloc"], "1/f behavior")
    if "psd_f2textloc" in settings:
        plt.text(*settings["psd_f2textloc"], "1/f$^2$")
    plt.savefig(f"figs/tech{TECH}/psd.png", dpi=300, bbox_inches="tight")
    plt.show()
```

```
0 step 1
0 step 2
8 step 1
8 step 2
16 step 1
16 step 2
31 step 1
31 step 2
```



1.3 Relaxation data analysis

Here, we will be analyzing the large dataset relaxation behavior. We will examine: (1) examples of conductance distribution broadening behavior over time, (2) scatterplot of conductance deviation vs. conductance, (3) standard deviation vs. time for different starting conductance values. First let us load the data:

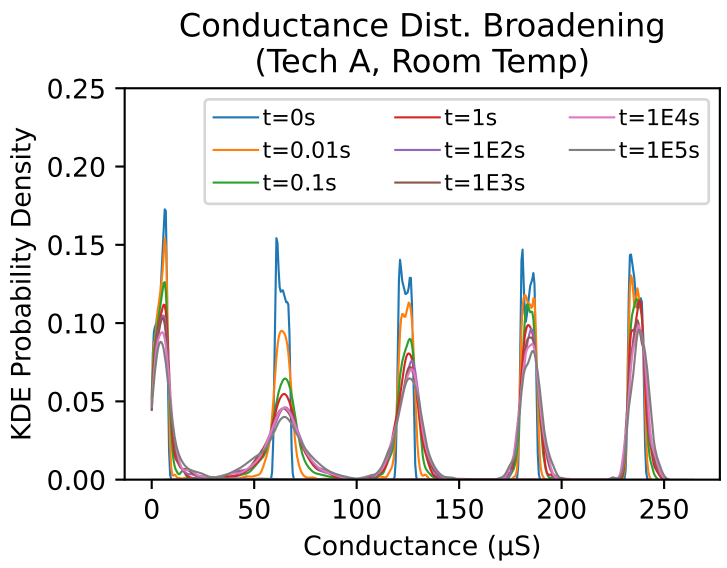
[5]:
```
# Load data for technology
colnames = ["addr", "time", "r", "g", "gi", "range", "timept"]
data = pd.read_csv(f"data/tech{TECH}/relaxdata.min.tsv.gz", names=colnames,␣
 ↪sep='\t')
data.head()
```

[5]:
```
    addr  time              r         g        gi  range  timept
0  80000   0.0  337532.079408  0.000003  0.000003      0     0.0
1  80001   0.0   76320.104403  0.000013  0.000013      1     0.0
2  80002   0.0   48034.594014  0.000021  0.000021      2     0.0
3  80003   0.0   21027.150891  0.000048  0.000048      6     0.0
4  80004   0.0   33193.229603  0.000030  0.000030      4     0.0
```

1.3.1 Conductance distribution broadening behavior

Below are examples of conductance broadening behavior over time:

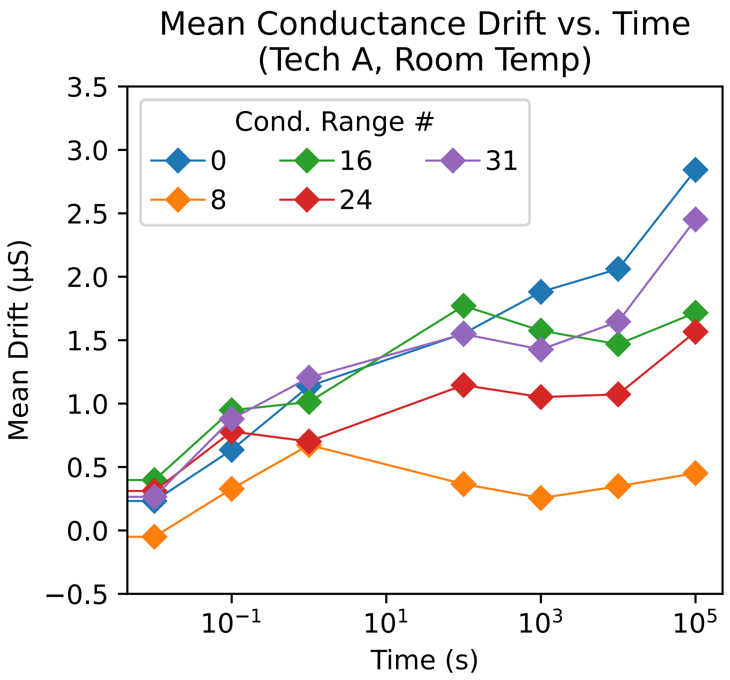
```
[6]: # Select ranges to study
     ranges = [0, 8, 16, 24, 31]

     # Conductance broadening behavior
     fig = plt.figure(figsize=(4, 2.7))
     ax = fig.add_subplot(111)
     ax.set_title(f"Conductance Dist. Broadening\n(Tech {TECH}, Room Temp)")
     colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]
     for time, color in zip(settings["times"], colors):
         for r in ranges:
             gx = np.linspace(0, settings["gmax"] * 1.1e6, 500)
             gvals = data[(data["range"] == r) & (data["timept"] == time)]["g"]
             pdf = scipy.stats.gaussian_kde(gvals * 1e6).pdf(gx)
             label = (f"t={time}s" if time < 100 else f"t=1E{int(np.log10(time))}s")␣
      ↪if r == 0 else None
             plt.plot(gx, pdf, color=color, label=label, linewidth=0.8)
     ax.legend(ncol=3, handletextpad=0.2, fontsize="small")
     ax.set_ylim(*settings["gbroad_ylim"])
     ax.set_xlabel("Conductance (µS)")
     ax.set_ylabel("KDE Probability Density")
     plt.savefig(f"figs/tech{TECH}/broadening-time.pdf", bbox_inches="tight")
     plt.show()
```



1.3.2 Mean Drift and Variance Growth (Time Dependence)

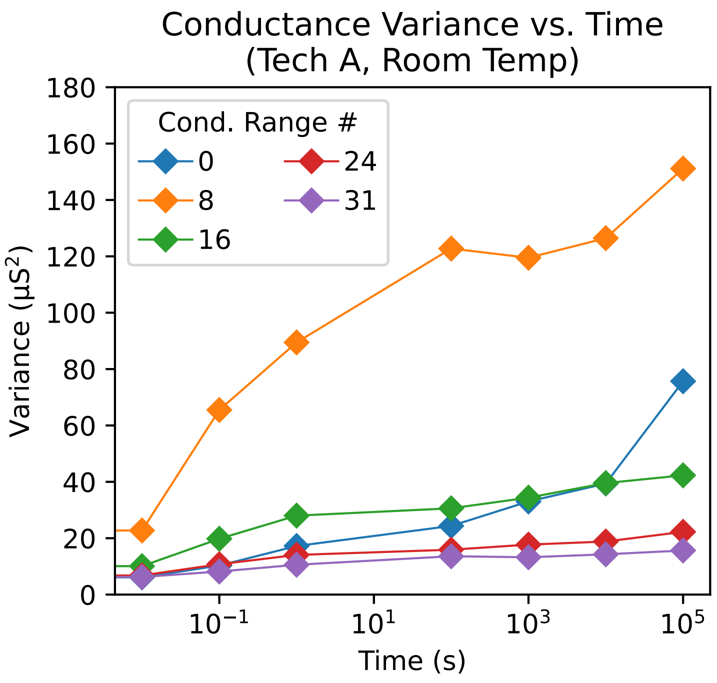
Here, we examine the drift of the distribution mean and variance growth as a function of time.

```
[7]: # Mean drift behavior (time dependence)
     fig = plt.figure(figsize=(4, 3.5))
     ax = fig.add_subplot(111)
     ax.set_title(f"Mean Conductance Drift vs. Time\n(Tech {TECH}, Room Temp)")
     colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]
     for r, color in zip(ranges, colors):
         d = data[(data["range"] == r) & (data["gi"] <= settings["gmax"])]
         gi = d.groupby("timept").mean()["gi"] * 1e6
         gf = d.groupby("timept").mean()["g"] * 1e6
         deltag = gf - gi
         plt.plot(deltag.index, deltag, '-D', color=color, label=r, linewidth=0.8)
     ax.legend(title="Cond. Range #", ncol=2 if TECH == 'C' else 3, handletextpad=0.
      ↪2)
     if "gmeandrift_t_ylim" in settings:
         ax.set_ylim(*settings["gmeandrift_t_ylim"])
     ax.set_xscale("log")
     ax.set_xlabel("Time (s)")
     ax.set_ylabel("Mean Drift (µS)")
     plt.savefig(f"figs/tech{TECH}/mean-drift-vs-time.pdf", bbox_inches="tight")
     plt.show()
```



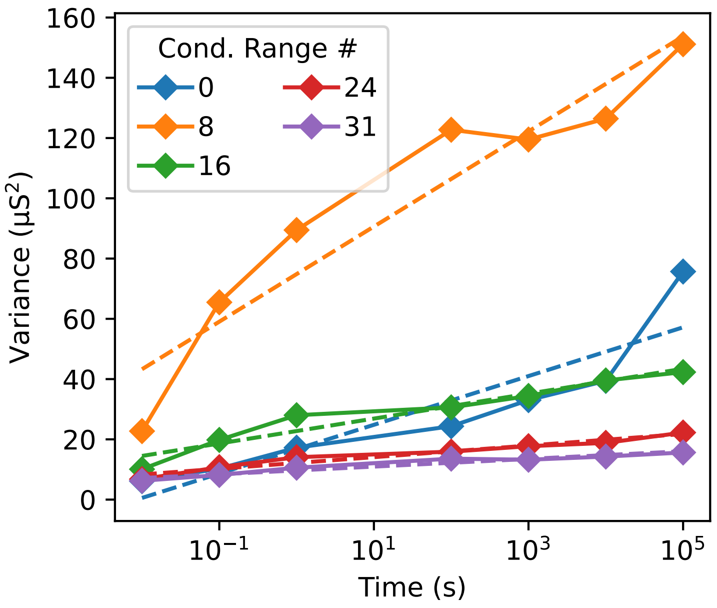
```
[8]: # Variance behavior (time dependence)
     fig = plt.figure(figsize=(4, 3.5))
     ax = fig.add_subplot(111)
```

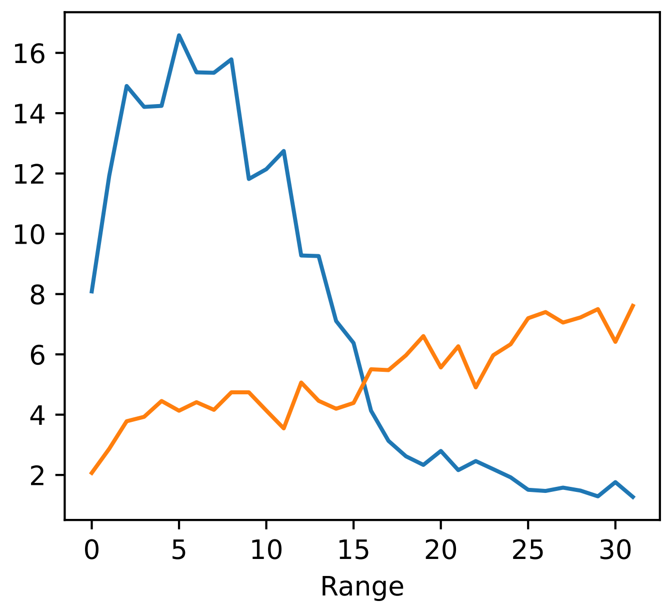
```
ax.set_title(f"Conductance Variance vs. Time\n(Tech {TECH}, Room Temp)")
colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]
for r, color in zip(ranges, colors):
    d = data[(data["range"] == r) & (data["gi"] <= settings["gmax"])]
    gi = d.groupby("timept").mean()["gi"] * 1e6
    gfvar = d.groupby("timept").var()["g"] * (1e6 ** 2)
    plt.plot(gfvar.index, gfvar, '-D', color=color, label=r, linewidth=0.8)
ax.legend(title="Cond. Range #", ncol=2, handletextpad=0.2)
if "gvar_t_ylim" in settings:
    ax.set_ylim(*settings["gvar_t_ylim"])
ax.set_xscale("log")
ax.set_xlabel("Time (s)")
ax.set_ylabel("Variance (μS$^2$)")
plt.savefig(f"figs/tech{TECH}/var-vs-time.pdf", bbox_inches="tight")
plt.show()
```



[9]:
```
# Variance behavior fitting on time dependence (var(g, t) = C1(g) * log(t) +␣
↪C2(g))
fig = plt.figure(figsize=(4, 3.5))
ax = fig.add_subplot(111)
C_fit = np.zeros((32, 2))
for r in range(32):
    d = data[(data["range"] == r) & (data["gi"] <= settings["gmax"]) &␣
↪(data["timept"] > 0)]
```

```
    gfvar = d.groupby("timept").var()["g"] * (1e6 ** 2)
    fit = np.polyfit(np.log10(gfvar.index), gfvar.values, 1)
    C_fit[r, :] = fit
    p = np.poly1d(fit)
    if r in ranges:
        color = colors[ranges.index(r)]
        ax.plot(gfvar.index, gfvar, '-D', color=color, label=r)
        ax.plot(gfvar.index, p(np.log10(gfvar.index)), '--', color=color)
ax.set_xscale("log")
ax.legend(title="Cond. Range #", ncol=2, handletextpad=0.2)
ax.set_xlabel("Time (s)")
ax.set_ylabel("Variance (µS$^2$)")
plt.show()

fig = plt.figure(figsize=(4, 3.5))
ax = fig.add_subplot(111)
ax.plot(range(32), C_fit[:, 0], range(32), C_fit[:, 1] / C_fit[:, 0]) # C_fit[:
 ↪, 1] / C_fit[:, 0] should be constant log(fs/2)
ax.set_xlabel("Range")
plt.show()
```

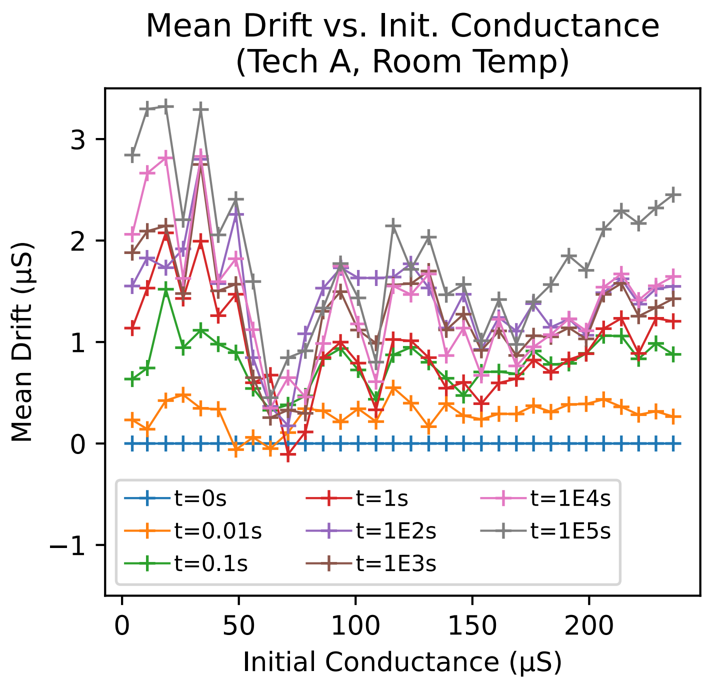




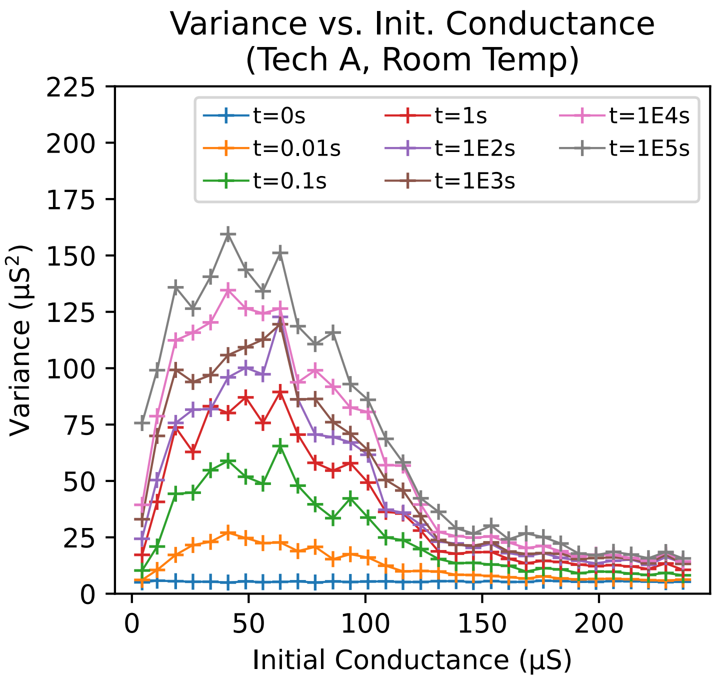
1.3.3 Mean Drift and Variance Growth (Conductance Dependence)

Here, we examine the drift of the distribution mean and variance growth as a function of conductance.

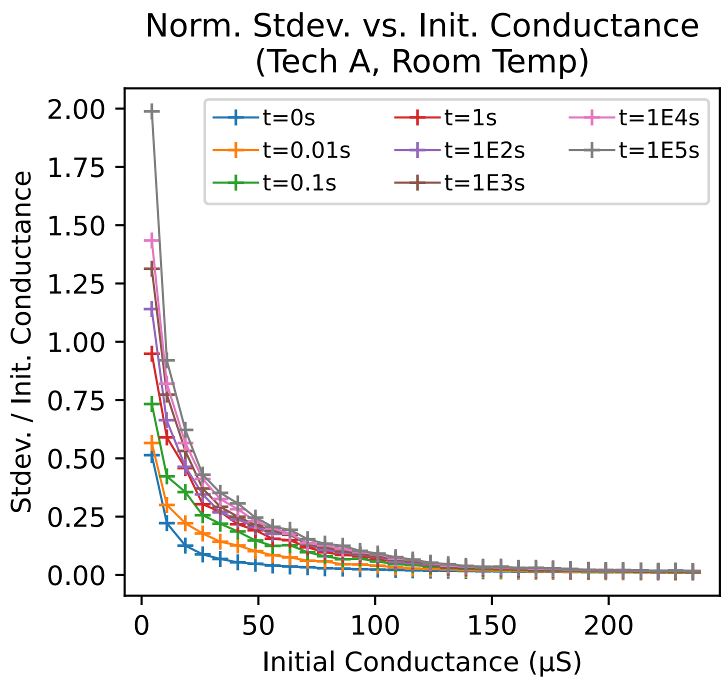
[10]:
```python
# Mean drift behavior (conductance dependence)
fig = plt.figure(figsize=(4, 3.5))
ax = fig.add_subplot(111)
ax.set_title(f"Mean Drift vs. Init. Conductance\n(Tech {TECH}, Room Temp)")
colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]
for time, color in zip(settings["times"], colors):
    d = data[(data["timept"] == time) & (data["gi"] <= settings["gmax"])]
    gi = d.groupby("range").mean()["gi"] * 1e6
    gf = d.groupby("range").mean()["g"] * 1e6
    deltag = gf - gi
    label = (f"t={time}s" if time < 100 else f"t=1E{int(np.log10(time))}s")
    plt.plot(gi, deltag, '-+', color=color, label=label, linewidth=0.8)
ax.legend(ncol=3, handletextpad=0.2, fontsize="small")
ax.set_ylim(*settings["gmeandrift_gi_ylim"])
ax.set_xlabel("Initial Conductance (µS)")
ax.set_ylabel("Mean Drift (µS)")
plt.savefig(f"figs/tech{TECH}/mean-drift-vs-g.pdf", bbox_inches="tight")
plt.show()
```



[11]:
```python
# Variance behavior (conductance dependence)
fig = plt.figure(figsize=(4, 3.5))
ax = fig.add_subplot(111)
ax.set_title(f"Variance vs. Init. Conductance\n(Tech {TECH}, Room Temp)")
colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]
for time, color in zip(settings["times"], colors):
    d = data[(data["timept"] == time) & (data["gi"] <= settings["gmax"])]
    gi = d.groupby("range").mean()["gi"] * 1e6
    gfvar = (d.groupby("range")).var()["g"] * (1e6 ** 2)
    label = (f"t={time}s" if time < 100 else f"t=1E{int(np.log10(time))}s")
    plt.plot(gi, gfvar, '-+', color=color, label=label, linewidth=0.8)
ax.legend(ncol=3, handletextpad=0.2, fontsize="small")
ax.set_ylim(*settings["gvar_gi_ylim"])
ax.set_xlabel("Initial Conductance (µS)")
ax.set_ylabel("Variance (µS$^2$)")
plt.savefig(f"figs/tech{TECH}/var-vs-g.pdf", bbox_inches="tight")
plt.show()
```



[12]:
```python
# Norm. stdev. behavior (conductance dependence)
fig = plt.figure(figsize=(4, 3.5))
ax = fig.add_subplot(111)
ax.set_title(f"Norm. Stdev. vs. Init. Conductance\n(Tech {TECH}, Room Temp)")
colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]
for time, color in zip(settings["times"], colors):
    d = data[(data["timept"] == time) & (data["gi"] <= settings["gmax"])]
    gi = d.groupby("range").mean()["gi"]*1e6
    gfnormdev = d.groupby("range").std()["g"] / d.groupby("range").mean()["gi"]
    label = (f"t={time}s" if time < 100 else f"t=1E{int(np.log10(time))}s")
    plt.plot(gi, gfnormdev, '-+', color=color, label=label, linewidth=0.8)
ax.legend(ncol=3, handletextpad=0.2, fontsize="small")
#ax.set_ylim(*settings["gvar_gi_ylim"])
ax.set_xlabel("Initial Conductance (µS)")
ax.set_ylabel("Stdev. / Init. Conductance")
plt.savefig(f"figs/tech{TECH}/norm-stdev-vs-g.pdf", bbox_inches="tight")
plt.show()
```



1.4 Temperature dependence analysis via 1hr bake

Here, we analyze the effect of baking on the distribution broadening. In particular, we will examine examples of the conductance distributions broadening for different temperatures and then analyze the temperature-dependent mean drift and variance. We can first load the data and preprocess it a little bit:

[13]:
```
# Load data for technology
colnames = ["addr", "time", "r", "g", "temp"]
data = pd.read_csv(f"data/tech{TECH}/bake.tsv.gz", names=colnames, sep='\t')

# Get conductance range
data["gi"] = data.groupby(["addr", "temp"])["g"].transform("first")
data["range"] = np.int32(data["gi"] / settings["gmax"] * 32)

# Filter out the rest of the data
data = data[data["time"] > 100].groupby(["addr", "temp"]).first().reset_index()

# Show data
data.head()
```

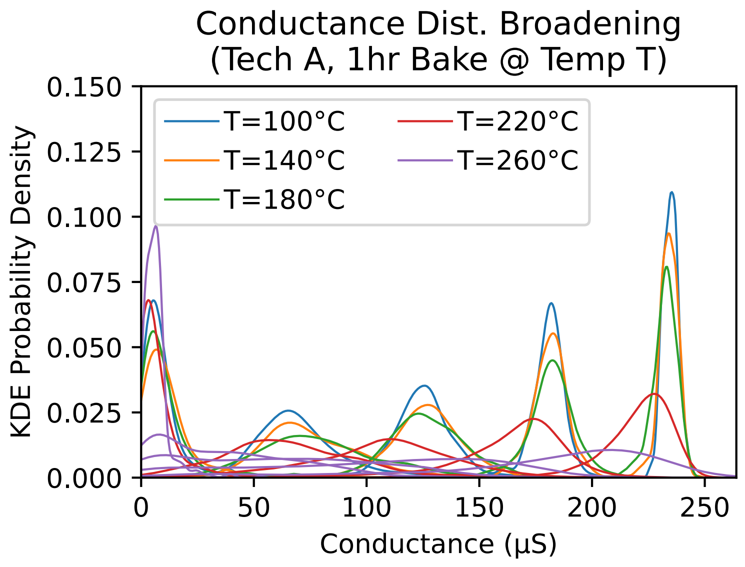
[13]:
```
    addr  temp         time             r         g        gi  range
0  80000   100  7164.671020   6317.765383  0.000158  0.000112     14
1  80000   140  4493.587238   7598.918623  0.000132  0.000068      9
2  80000   180  5263.012372  16033.495632  0.000062  0.000015      1
```

```
3  80000   220  4566.413692  43554.782008  0.000023  0.000006       0
4  80000   260  5199.774635  82084.987230  0.000012  0.000016       2
```

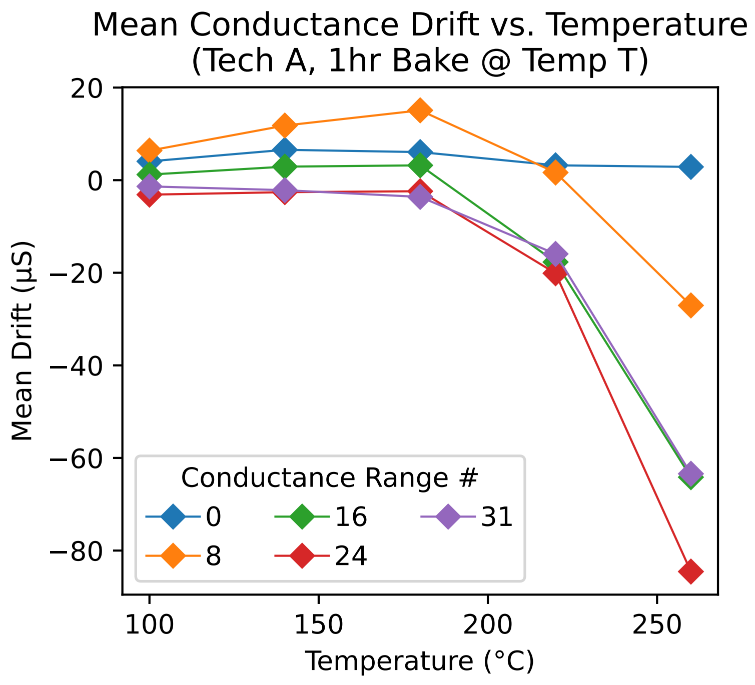
1.4.1 Conductance distribution broadening behavior

Below are examples of conductance broadening behavior at different temperatures:

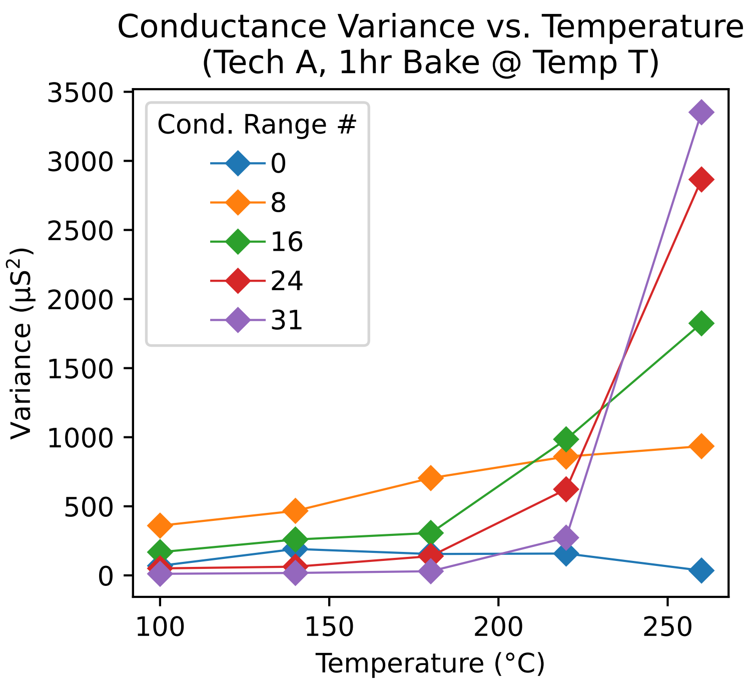
[14]:
```
# Select ranges to study
ranges = [0, 8, 16, 24, 31]

# Conductance broadening behavior
fig = plt.figure(figsize=(4, 2.7))
ax = fig.add_subplot(111)
ax.set_title(f"Conductance Dist. Broadening\n(Tech {TECH}, 1hr Bake @ Temp T)")
colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]
for temp, color in zip(settings["temps"], colors):
    for r in ranges:
        gx = np.linspace(0, settings["gmax"]*1.1e6, 500)
        gvals = data[(data["range"] == r) & (data["temp"] == temp)]["g"]
        pdf = scipy.stats.gaussian_kde(gvals*1e6).pdf(gx)
        plt.plot(gx, pdf, color=color, label=f"T={temp}°C" if r==0 else None,␣
↪linewidth=0.8)
ax.legend(ncol=2, handletextpad=0.2)
ax.set_xlim(0, settings["gmax"]*1.1e6)
ax.set_ylim(*settings["gbroad_temp_ylim"])
ax.set_xlabel("Conductance (µS)")
ax.set_ylabel("KDE Probability Density")
plt.savefig(f"figs/tech{TECH}/broadening-temp.pdf", bbox_inches="tight")
plt.show()
```



```
[15]: # Mean drift behavior (temperature dependence)
fig = plt.figure(figsize=(4, 3.5))
ax = fig.add_subplot(111)
ax.set_title(f"Mean Conductance Drift vs. Temperature\n(Tech {TECH}, 1hr Bake @␣
 ↪Temp T)")
colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]
for r, color in zip(ranges, colors):
    d = data[data["range"] == r]
    gi = d.groupby("temp").mean()["gi"]*1e6
    gf = d.groupby("temp").mean()["g"]*1e6
    deltag = gf - gi
    plt.plot(deltag.index, deltag, '-D', color=color, label=r, linewidth=0.8)
ax.legend(title="Conductance Range #", ncol=2 if TECH == 'C' else 3,␣
 ↪handletextpad=0.2)
if "gmeandrift_temp_ylim" in settings:
    ax.set_ylim(*settings["gmeandrift_temp_ylim"])
ax.set_xlabel("Temperature (°C)")
ax.set_ylabel("Mean Drift (µS)")
plt.savefig(f"figs/tech{TECH}/mean-drift-vs-temp.pdf", bbox_inches="tight")
plt.show()
```



```
[16]: # Variance behavior (temperature dependence)
fig = plt.figure(figsize=(4, 3.5))
ax = fig.add_subplot(111)
ax.set_title(f"Conductance Variance vs. Temperature\n(Tech {TECH}, 1hr Bake @␣
 ↪Temp T)")
colors = plt.rcParams["axes.prop_cycle"].by_key()["color"]
for r, color in zip(ranges, colors):
    d = data[data["range"] == r]
    gfvar = d.groupby("temp").var()["g"]*(1e6**2)
    plt.plot(gfvar.index, gfvar, '-D', color=color, label=r, linewidth=0.8)
ax.legend(title="Cond. Range #", ncol=1, handletextpad=0.2)
if "gvar_temp_ylim" in settings:
    ax.set_ylim(*settings["gvar_temp_ylim"])
ax.set_xlabel("Temperature (°C)")
ax.set_ylabel("Variance (μS$^2$)")
plt.savefig(f"figs/tech{TECH}/var-vs-temp.pdf", bbox_inches="tight")
plt.show()
```

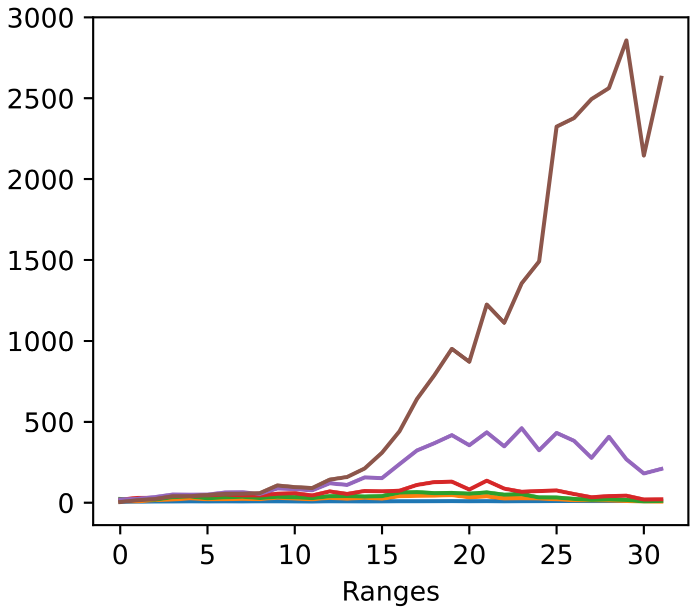


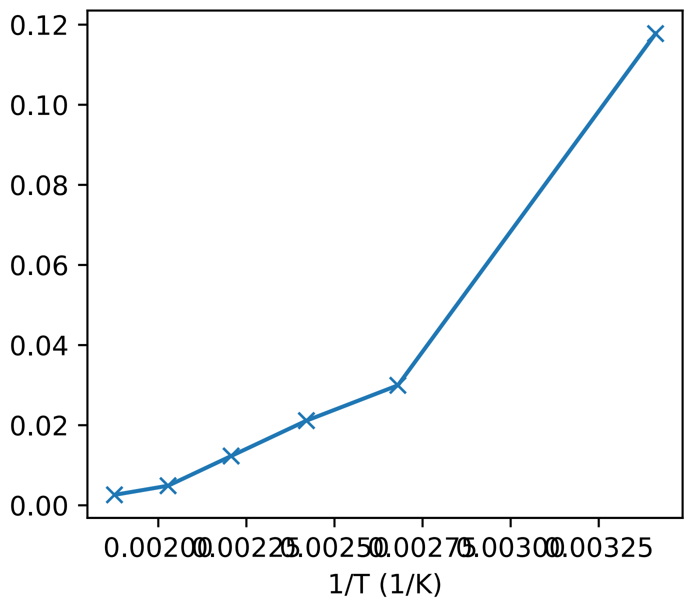
1.4.2 Temperature dependence fitting

Here we fit the temperature dependence parameters. The basic idea is to match the time dependence with the temperature dependence, and limit the number of assumptions made to derive the carrier distribution from temperature dependence. According to the equation $\sigma^2_{\Delta x}(t,T) = \frac{\pi(\rho_x w_0^3)^2 \ln(f_s t/2)}{(E_{a2}-E_{a1})/kT+\gamma(d_2-d_1)}$, for any temperature T_1 other than room temperature T_0, there exists some time $\tilde{t}$ such that $\sigma^2_{\Delta x}(\tilde{t}, T_1) = \sigma^2_{\Delta x}(t, T_0)$ holds on both width and length directions. Hence without approximating the final conductance variance on only one dimension, there should be $\mathrm{Var}(g_j, \tilde{t}, T_1) = \mathrm{Var}(g_j, t, T_0)$ for all j.

For any time dependence relationship $\mathrm{Var}(g_j, t, T_0) = f_j(t)$, we could extract $\tilde{t}$ from the temperature-dependent variance data, and get the ratio of the denominator $(E_{a2} - E_{a1})/kT + \gamma(d_2 - d_1)$ from $\ln(f_s t/2)$. In tech B example, the time dependence is shows a linear $\ln(f_s t/2)$ relationship, so the denominator ratio would be exactly the variance ratio, but such a linear $\ln t$ relationship is not needed for assumption here. We could derive the ratio between $E_{a2} - E_{a1}$ and $\gamma(d_2 - d_1)$ from the plot. Moreover, the denominator might not be linear with $1/kT$ if the carriers are not uniformly distributed, in which case the more complicated relationship could be reversely derived from the $\ln(f_s\tilde{t}/2)$-$1/kT$ relationship.

[17]:
```
fig = plt.figure(figsize=(4, 3.5))
ax = fig.add_subplot(111)
denom = np.zeros((32, 5))
for r in range(32):
    d = data[(data["range"] == r) & (data["gi"] <= settings["gmax"]) &␣
 ↪(data["g"] <= settings["gmax"])]
    gfvar = d.groupby("temp").var()["g"] * (1e6 ** 2)
    denom[r, :] = gfvar.values / C_fit[r, 0] # C_fit is from time dependence␣
 ↪fitting
room_denom = np.log10(3600) + C_fit[:, 1] / C_fit[:, 0] # Adding room␣
 ↪temperature
denom = np.hstack((room_denom.reshape(-1, 1), denom))
ax.plot(range(32), denom)
ax.set_xlabel("Ranges")
plt.show()

fig = plt.figure(figsize=(4, 3.5))
ax = fig.add_subplot(111)
x = np.mean(denom[10:20, :], axis=0) # Averaging among useful ranges
temp = np.array([20] + settings["temps"]) + 273.15
ax.plot(1 / temp, 1 / x, '-x')
ax.set_xlabel("1/T (1/K)")
plt.show()
fit = np.polyfit(1 / temp, 1 / x, 1)
print(fit)
```





```
[74.718783   -0.15064355]
```