# DeerLab
## Basics

## 1 Initialize

```python
import numpy as np
import matplotlib.pyplot as plt
import deerlab as dl
```

## 2 Load data

```python
t,V = dl.deerload("dataset.DTA")
Vexp = dl.correctphase(V)
Vexp = Vexp/np.max(Vexp)
```

## 3 Time axis correction

```python
tmin = 0.3  # µs
t = t - t[0]
t = t + tmin
```

## 4 Experiment

```python
r = np.linspace(2,5,50)
tau1 = 0.4  # µs
tau2 = 3.5  # µs

my4pdeer = dl.ex_4pdeer(
          tau1, tau2, pathways=[1,2,3])
```

## 5a Model (non-parametric)

```python
Vmodel = dl.dipolarmodel(
          t, r, experiment=my4pdeer)
```

## 5b Model (parametric)

```python
Pmodel = dl.dd_gauss2

Vmodel = dl.dipolarmodel(t, r, pmodel=Pmodel,
                experiment=my4pdeer)
```

## 6a Fitting

```python
results = dl.fit(Vmodel, Vexp)
print(results)
results.plot(xlabel="Time ($\mu s$)")
```

## 6b Fitting with compactness

```python
compactness = dl.dipolarpenalty(
        Pmodel=None, r=r, type='compactness')
results = dl.fit(
        Vmodel, Vexp, penalties=compactness)
print(results)
results.plot(xlabel="Time ($\mu s$)")
```

## 7a Extracting distance (non-parametric)

```python
P = results.P
Puq = results.PUncert
Pci95 = Puq.ci(95)
```
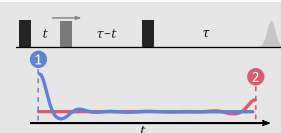
## 7b Extracting distance (parametric)

```python
Pfit = results.evaluate(Pmodel, r)
scale = np.trapz(Pfit, r)
Puncert = results.propagate(
                Pmodel, r, lb=np.zeros_like(r))
Pfit = Pfit / scale
Pci95 = Puncert.ci(95) / scale
```
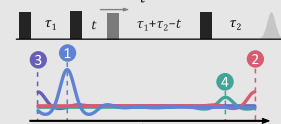
## Experiment

**3 pulse DEER**
```python
dl.ex_4pdeer(tau, pathways,
        pulselength)
```
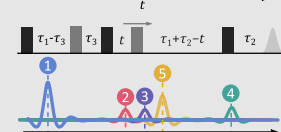
**4 pulse DEER:**
```python
dl.ex_4pdeer(tau1, tau2,
        pathways, pulselength)
```
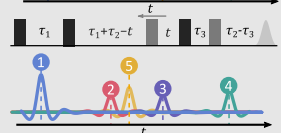
**5 pulse DEER**
```python
dl.ex_fwd5pdeer(tau1, tau2,
tau3,  pathways,pulselength)
```
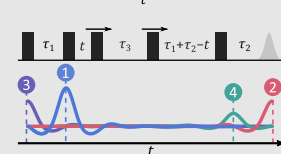
```python
dl.ex_rev5pdeer(tau1, tau2,
tau3,  pathways, pulselength)
```

**RIDME**
```python
dl.ex_ridme(tau1, tau2,
        pathways, pulselength)
```
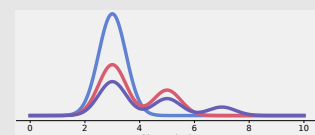
Other Experiments Models are available…

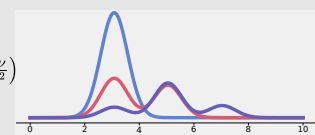## Distance Models

**Gaussian**

$$P(r) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(r - \langle r \rangle)^2}{2\sigma^2}\right)$$

```python
dl.dd_gauss(r, mean, std)
dl.dd_gauss2(r, mean1, std1, amp1, mean2, std2, amp2)
dl.dd_gauss3(r, mean1, std1, amp1, mean2, std2, amp2, …)
```
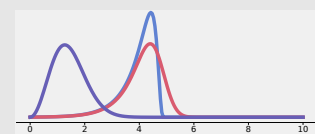
**Rice**

$$P(r) = \frac{\nu^{n/2-1}}{\sigma^2}r^{n/2}\exp\left(-\frac{(r^2 + \nu^2)}{2\sigma^2}\right)I_{n/2-1}\left(\frac{r\nu}{\sigma^2}\right)$$

```python
dl.dd_rice(r, location, spread)
dl.dd_rice2(r, location1, spread1, amp1, location2, spread2, amp2)
dl.dd_rice3(r, location1, spread1,  amp1, location2, spread2, amp2, …)
```

**Disordered physical models**

```python
dl.dd_wormchain(r, contour,
        persistence)
dl. dd_wormgauss(r, contour,
        persistence, std)
dl.dd_randcoil(r, Nres, scaling,
        length)
```

# DeerLab
## Advanced

## Background Models

```
dl.fit(..., Bmodel=dl.bg_hom3d, ...)
```

**Physical Models**

```
dl.bg_hom3d(t, conc, lam)
dl.bg_hom3dex(t, conc, rex, lam)
dl.bg_homfractal(t, fconc, fdim, lam)
```

**Phenomenological Models**

```
dl.bg_exp(t, decay)
dl.bg_strexp(t, decay, stretch)
dl.bg_prodstrexp(t, decay1, stretch1, decay2,
                 stretch2)
```

## Other useful functions

**... bootstrap sampling?**
→ `dl.fit(..., bootsrap=200, bootcore=4)`
**... distribution statistics?**
→ `dl.diststats(r, P, Puq=Puq, verbose=True)`
**... fit statistics?**
→ `dl.goodness_of_fit(data, results.model, Ndof,`
`               noiselvl)`
**... residual analysis plots?**
→ `fit.plot(axis=t, xlabel='t ($\mu s$)',`
`           gof=True)`
**... save object?**
→ `dl.store_pickle(object, "filename")`
**... load object?**
→ `dl.load_pickle(object, "filename")`
**... profile likelihood analysis?**
→ `dl.profile_analysis(model, y,`
`               parameters=["lam1"])`
**... save array as txt?**
→ `np.savetxt('filename', array)`
**... save array as matfile?**
→ `from scipy.io import savemat`
→ `savemat('filename.mat', array1, array2, ...)`

## Basic Plotting

**Import Matplotlib**
→ `import matplotlib.pyplot as plt`

**Create plot**
→ `plt.figure(figsize=[10,3])`
→ `plt.subplot(1,2,1)`

**Plot data points**
→ `plt.plot(t, Vexp, '.', color='0.6',`
`               label='data')`

**Plot fit**
→ `plt.plot(t, results.model, lw=3, label='Fit')`

**Plot labels**
→ `plt.legend()`
→ `plt.xlabel('t ($\mu s$)')`
→ `plt.ylabel('V (arb.u.)')`

**Change sub-figure**
→ `plt.subplot(1,2,2)`

**Plot distance distribution**
→ `plt.plot(r, P, linewidth=3)`

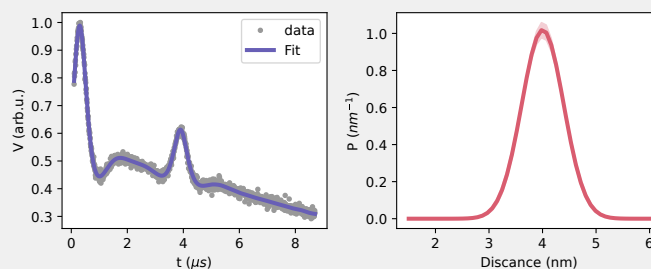**Plot distance uncertainties**
→ `plt.fill_between(r, P95[:,0], P95[:,1],`
`               alpha=0.3)`

**Plot labels**
→ `plt.xlabel('Discance (nm)')`
→ `plt.ylabel('P ($nm^{-1}$)')`

**Save plot?**
→ `plt.save_plot("filename", dpi)`



More infomation can be found on the matplotlib docs

## Global Fitting

**1. Build individual Models**
```
Vmodels = []
Vmodels.append(modelA)
...
```

**2. Put data into a list**
```
Vs = []
Vs.append(dataA)
...
```

**3. Make the global model by joining the individual models**
```
globalmodel = dl.merge(*Vmodels)
```

**4. Link the distance distribution into a global parameter**
```
globalmodel = dl.link(globalmodel,
P=['P_1','P_2'])
```

**5. Adjust weights in fit (optional)**
```
results = dl.fit(globalmodel, Vs, weights=[1,1])
```

## Model Manipulation

**... extract a parameter?**
→ `model.param`  # e.g. Model.conc, model.lam1
**... freeze a parameter?**
→ `param.freeze(value)`  # e.g. lam1.freeze(0.25)
**... unfreeze a parameter?**
→ `param.unfreeze()`
**... set the initial value?**
→ `param.set(par0=value)`
**... set the limits?**
→ `param.set(lb=0.1, ub=np.inf)`

**... link two parameter?**
→ `dl.link(model, newparam=['paramA','paramB'])`
**... merge two parameter?**
→ `dl.merge(model1, model2)`
**... relate two parameter?**
→ `dl.relate(model, paramA=lambda paramF:`
`fcn(paramF))`

**... linear combine two models?**
→ `dl.lincombine(model1, model2, model3)`
**... copy a model?**
→ `from copy import deepcopy`
→ `deepcopy(modelA)`