scaled_dl

July 14, 2022

[1]:
```
import pybamm
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from functools import partial
```

1 Rescaled double layer model definition

We first generate two sub-class definitions, namely: - SclDiffForm, which introduces the 1/a scaling in the delta_phi RHS - DFNScl, which simply overrides the set_electrolyte_submodel to leverage the new SclDiffForm submodel

[2]:
```
class SclDiffForm(pybamm.electrolyte_conductivity.surface_potential_form.
↪FullDifferential):
    def set_rhs(self, variables):
        if self.domain == "Separator":
            return

        C_dl = self.domain_param.C_dl

        delta_phi = variables[self.domain + " electrode surface potential␣
↪difference"]
        i_e = variables[self.domain + " electrolyte current density"]

        # Get surface area to volume ratio (could be a distribution in x to
        # account for graded electrodes)
        a = variables[self.domain + " electrode surface area to volume ratio"]

        # Variable summing all of the interfacial current densities
        sum_j = variables[
            "Sum of " + self.domain.lower() + " electrode interfacial current␣
↪densities"
        ]

        self.rhs[delta_phi] = 1 / (a * C_dl) * (pybamm.div(i_e) - a * sum_j)
```

```
class DFNScl(pybamm.lithium_ion.DFN):
    def set_electrolyte_submodel(self):

        surf_form = pybamm.electrolyte_conductivity.surface_potential_form

        self.submodels["electrolyte diffusion"] = pybamm.electrolyte_diffusion.
 ↪Full(
            self.param, self.options
        )

        if self.options["electrolyte conductivity"] not in ["default", "full"]:
            raise pybamm.OptionError(
                "electrolyte conductivity '{}' not suitable for DFN".format(
                    self.options["electrolyte conductivity"]
                )
            )

        if self.options["surface form"] == "false":
            self.submodels[
                "electrolyte conductivity"
            ] = pybamm.electrolyte_conductivity.Full(self.param, self.options)
        if self.options["surface form"] == "false":
            surf_model = surf_form.Explicit
        elif self.options["surface form"] == "differential":
            surf_model = SclDiffForm
#             surf_model = surf_form.FullDifferential
        elif self.options["surface form"] == "algebraic":
            surf_model = surf_form.FullAlgebraic

        for domain in ["Negative", "Separator", "Positive"]:
            self.submodels[
                domain.lower() + " surface potential difference"
            ] = surf_model(self.param, domain, self.options)
```

2 EIS simulation and measurement extraction

Next we define functions to - generate sinusoidal driving currents - simulate EIS experiments in PyBaMM - extract impedance measurements from voltage/current measurements via the FFT

[3]:
```
def curr(A, freq):
    def Iapp(t):
        return A*pybamm.sin(2*np.pi*freq*t)
    return Iapp

def measure_impedance(t, I, V, phase_shift=np.pi):
    I_centered = I - I.mean()
    V_centered = V - V.mean()
    N = len(I_centered)
    nyq_bin = N//2 + 1
    dt = t[1] - t[0]
    fs = np.array(range(0, nyq_bin))*(1/dt/N)

    Ihat = np.fft.fft(I_centered)[0:nyq_bin]
    Vhat = np.fft.fft(V_centered)[0:nyq_bin]
    idxI = np.argmax(np.abs(Ihat))
    idxV = np.argmax(np.abs(Vhat))
    Z = Vhat[idxV] / Ihat[idxI] * np.exp(-1j*phase_shift)
    return Z

def get_eis_data(model, params, freqs,
                 num_periods=1, samples_per_period=2048, amp=1e-3,
                 phase_shift=np.pi, soc=0.6):
    solver = pybamm.CasadiSolver(mode='fast')
    Z = []
    cycle_data = pd.DataFrame()
    for freq in freqs:
        params['Current function [A]'] = curr(amp, freq)
        dt = 1/freq/samples_per_period
        N = samples_per_period*num_periods
        tspan = np.array(range(0, N))*dt
        sol = pybamm.Simulation(model, parameter_values=params).solve(tspan,␣
 ↪solver=solver,
                                                                               ␣
 ↪initial_soc=soc)
        t, I, V = [sol[val].entries for val in ['Time [s]', 'Current [A]',␣
 ↪'Terminal voltage [V]']]
        cycle_data = pd.concat([cycle_data, pd.DataFrame({'t': t, 'I': I, 'V':␣
 ↪V, 'Fs': freq})]).\
            reset_index(drop=True)
        Z.append(measure_impedance(t, I, V, phase_shift=phase_shift))
    return np.array(Z), cycle_data
```

3 Model setup and EIS options

We set up two models to compare: - `dfn`, which is a standard DFN model as-is from PyBaMM - `dfn_scl`, which uses the (1/a)-scaled version of the double layer equation

Additionally we pull basic LCO parameters for simulation, and specify options for our EIS experiment simulations

[4]:
```
options = {'surface form': 'differential'}
# freqs = np.logspace(-1, 1, 3)
freqs = np.logspace(-3, 4, 70)
```

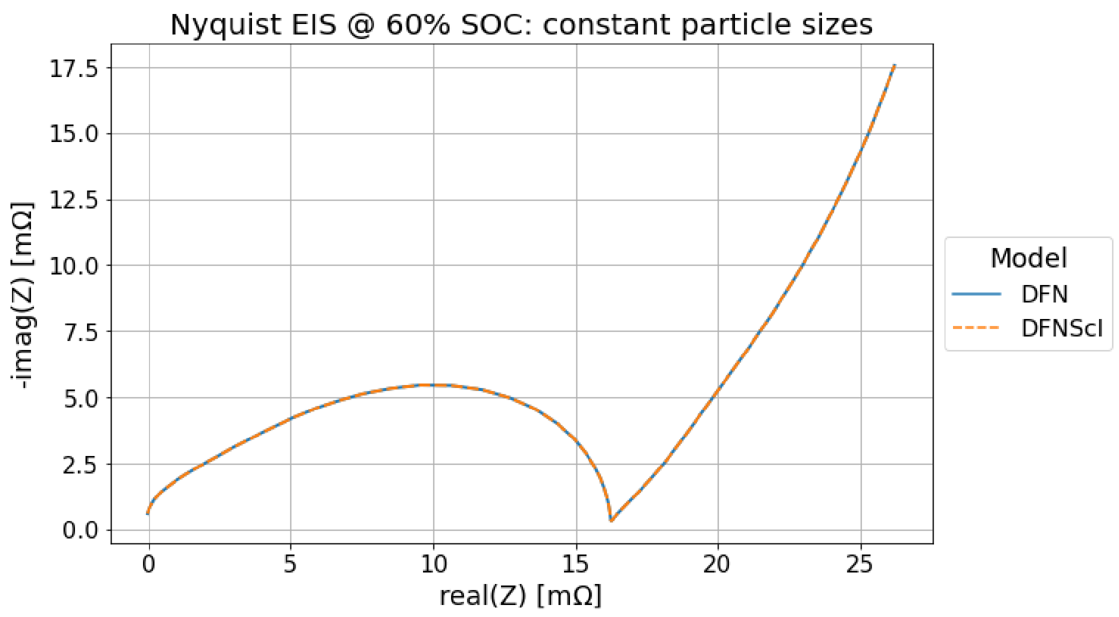
```
dfn = pybamm.lithium_ion.DFN(options=options)
dfn_scl = DFNScl(options=options)
params = pybamm.ParameterValues('Ramadass2004')
eis_opts = {'num_periods': 3, 'amp': params['Typical current [A]']/20}
```

4 Constant particle sizes

For the first comparison, we do not modify the LCO parameters, and thus use a **constant particle radius** profile for each electrode. Since there is no spatial dependence on R, we expect $a = 3\epsilon/R \equiv \frac{1}{|\mathrm{Vol}|}\int_{\mathrm{Vol}} a dV = \bar{a}$, and thus the outcome of the EIS experiment should be (and is) identical between the `dfn` and `dfn_scl` models

[5]:
```
Z_dfn, cycle_data_dfn = get_eis_data(dfn, params, freqs, **eis_opts)
Z_scl, cycle_data_scl = get_eis_data(dfn_scl, params, freqs, **eis_opts)

plot_dfs = [pd.DataFrame({'realZ': np.real(Z)*1e3, 'neg_imagZ': -np.
 ↪imag(Z)*1e3, 'model': mod_name})
            for Z, mod_name in zip([Z_dfn, Z_scl], ['DFN', 'DFNScl'])]
plot_df = pd.concat(plot_dfs).reset_index(drop=True)
```

[6]:
```
fig, ax = plt.subplots(figsize=(10, 10/1.6))
sns.lineplot(ax=ax, data=plot_df, x='realZ', y='neg_imagZ', hue='model',␣
 ↪style='model', lw=2)
ax.legend(bbox_to_anchor=(1, 0.5), loc='center left', fontsize=16,␣
 ↪title='Model',
          title_fontsize=18)
ax.set_xlabel('real(Z) [m$\Omega$]', fontsize=18)
ax.set_ylabel('-imag(Z) [m$\Omega$]', fontsize=18)
ax.tick_params(labelsize=16)
ax.grid('on')
ax.set_title('Nyquist EIS @ 60% SOC: constant particle sizes', fontsize=20);
```



5 Variable particle sizes

We now compare the `dfn` and `dfn_scl` models using a variable particle radii profile in each electrode. We set up a simple Gaussian profile for each electrode, with maximum values occurring in the center of the respective electrode and quickly tapering towards the boundaries. In this case, there is spatial dependence in R, thus the outcome of the EIS experiment should be (and is) slightly different between the two models.

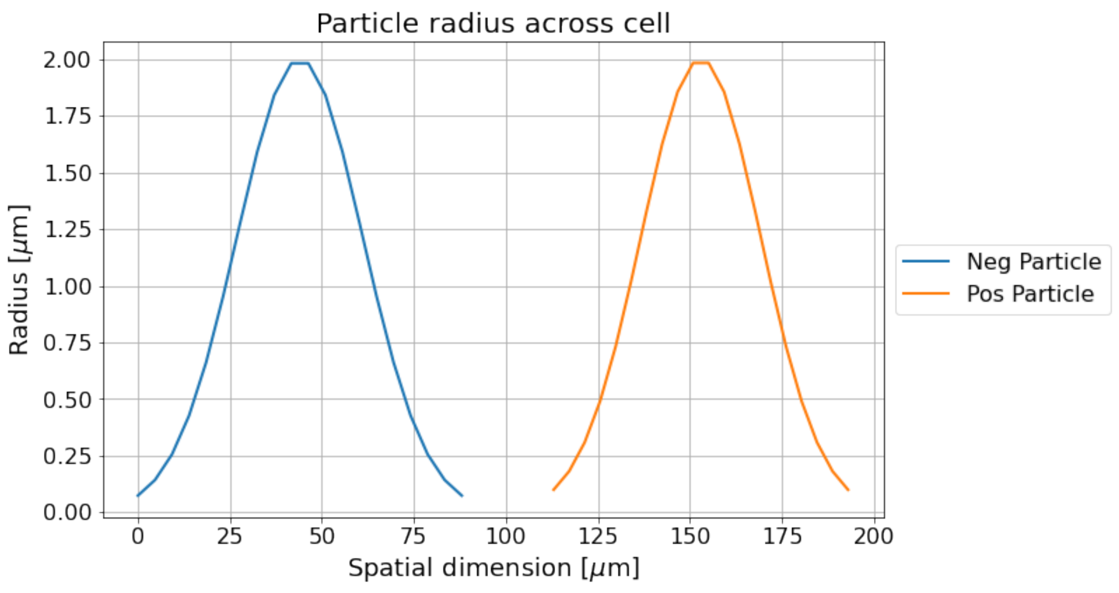
[7]:
```
params = pybamm.ParameterValues('Ramadass2004')
```

[8]:
```
def gauss(x, mu=0, sig=1, scl=1):
    return scl*pybamm.exp(-(x-mu)**2 / 2 / sig)
def particle_size(x, xl=0, xr=1, dilate=1, scl=1):
    mu = (xl + xr) / 2
    sig = (xr - xl) / 3 / dilate
    return gauss(x, mu=mu, sig=sig, scl=scl)
x_neg, x_sep, x_pos, scl_neg, scl_pos = map(lambda k: params[k], ['Negative␣
 ↪electrode thickness [m]',
                                                                  'Separator␣

 ↪thickness [m]',
                                                                  'Positive␣

 ↪electrode thickness [m]',
                                                                  'Negative␣

 ↪particle radius [m]',
                                                                  'Positive␣

 ↪particle radius [m]'])
```

```
params['Negative particle radius [m]'] = partial(particle_size,
                                                xl=0,
                                                xr=x_neg,
                                                dilate=1e5,
                                                scl=scl_neg)
params['Positive particle radius [m]'] = partial(particle_size,
                                                xl=x_neg + x_sep,
                                                xr=x_neg + x_sep + x_pos,
                                                dilate=1e5,
                                                scl=scl_pos)
```

Quick plot of the particle radii

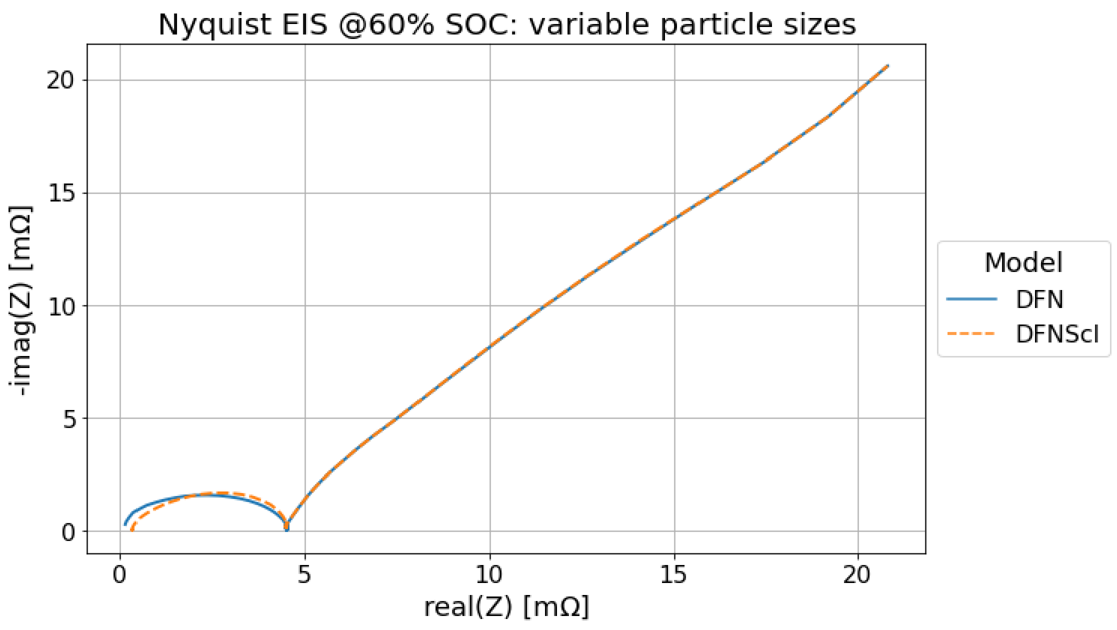
[9]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 10/1.6))
xs = np.linspace(0, x_neg, 20)
ys = np.linspace(x_neg + x_sep, x_neg + x_sep + x_pos, 20)
neg_rad = np.array([params['Negative particle radius [m]'](x).value for x in␣
 ↪xs])
pos_rad = np.array([params['Positive particle radius [m]'](y).value for y in␣
 ↪ys])
plt.plot(1e6*xs, 1e6*neg_rad, label='Neg Particle', lw=2)
plt.plot(1e6*ys, 1e6*pos_rad, label='Pos Particle', lw=2)
plt.legend(bbox_to_anchor=(1, 0.5), loc='center left', fontsize=16)
ax.set_xlabel('Spatial dimension [$\mu$m]', fontsize=18)
ax.set_ylabel('Radius [$\mu$m]', fontsize=18)
ax.tick_params(labelsize=16)
ax.grid('on')
ax.set_title('Particle radius across cell', fontsize=20);
```



```
[10]: Z_dfn, cycle_data_dfn = get_eis_data(dfn, params, freqs, **eis_opts)
      Z_scl, cycle_data_scl = get_eis_data(dfn_scl, params, freqs, **eis_opts)
      plot_dfs = [pd.DataFrame({'realZ': np.real(Z)*1e3, 'neg_imagZ': -np.
       ↪imag(Z)*1e3, 'model': mod_name})
                  for Z, mod_name in zip([Z_dfn, Z_scl], ['DFN', 'DFNScl'])]
      plot_df = pd.concat(plot_dfs).reset_index(drop=True)
```

```
[11]: fig, ax = plt.subplots(figsize=(10, 10/1.6))
      sns.lineplot(ax=ax, data=plot_df, x='realZ', y='neg_imagZ', hue='model',␣
       ↪style='model', lw=2)
      ax.legend(bbox_to_anchor=(1, 0.5), loc='center left', fontsize=16,␣
       ↪title='Model',
                title_fontsize=18)
      ax.set_xlabel('real(Z) [m$\Omega$]', fontsize=18)
      ax.set_ylabel('-imag(Z) [m$\Omega$]', fontsize=18)
      ax.tick_params(labelsize=16)
      ax.grid('on')
      ax.set_title('Nyquist EIS @60% SOC: variable particle sizes', fontsize=20)
```

```
[11]: Text(0.5, 1.0, 'Nyquist EIS @60% SOC: variable particle sizes')
```


Nyquist EIS @60% SOC: variable particle sizes
-imag(Z) [mΩ]
real(Z) [mΩ]
Model
DFN
DFNScl