Cell Reports Methods, Volume *3*

Supplemental information

High-throughput combined voltage-clamp/current-clamp analysis of freshly isolated neurons

Mohammad-Reza Ghovanloo, Sidharth Tyagi, Peng Zhao, Emre Kiziltug, Mark Estacion, Sulayman D. Dib-Hajj, and Stephen G. Waxman

1 Methods S1. Supplementary Algorithms, related to STAR Methods.

Description of functions:

The python script includes a set of 5 functions, 3 of which process data output from the automated patch clamp device, and 2 of which concatenate this processed data into a master file containing activation, inactivation, and recovery from inactivation data sorted by cell identifier.

Descriptions of the functions are as follows:

"Conductance" – Given a file with data for peaks of activation vs. voltage, normalizes to maximal conductance. Excludes cells where voltage was inadequately controlled. Fits both a single and double Boltzmann function, and plots data and saves as .png image. Bins data based on superior fit, and saves all parameters and normalized data.

"Inactivation" - Given a file with normalized data for peaks of fast inactivation vs. voltage, fits both a single and double Boltzmann function, and plots data and saves as .png image. Bins data based on superior fit, and saves all parameters and normalized data.

"Recovery from Inactivation" – Given a file with recovery from inactivation data (i.e. ratio of peak amplitude vs. time), fits a double exponential function. Saves images as .png, then saves all parameters and data.

"Process" – Concatenates results from files generated by other functions. Creates a master sheet of data, mapped by identifier.

"AddendRecInactivation" – If fit data for recovery from inactivation data is generated by an outside program (e.g. Prism), this function allows the data to be appended to the sheet of master data created by the "Process" function. Identifiers are preserved.

System Requirements:

This software was developed on the Anaconda distribution of Python 3.9.12.

Required packages are: pandas, numpy, os, glob, matplotlib, scikit.learn

Analysis Workflow:

The program is modular, meaning functions can be run independently from one another. For example, if one was interested in only fast inactivation data, they could ignore the other modules and the program will still work. The process function concatenates the outputs of these modular functions. You can select which data to include in the master file by commenting out lines corresponding to undesired functions.

Step-by-step instructions

The global path can be set at the top of the script. This path will be the location of all outputs in the functions.

<u>Conductance</u>

1. Create two input files, and save as .csv:

a. "Activation data" – Headers must be the unique cell identifier. The first column should be a list of test potentials (mV). Successive columns should be a list of current amplitudes corresponding to test potentials. Blank columns are okay.

b. "Activation reversal potentials" – Headers must be the unique cell identifier. They can be in a different order than in the Activation data and extras are okay. Cell A2 should be labeled "Vrev", and successive values in Row 2 should indicate the reversal potential of each cell. Blank columns are okay.

2. Replace labeled lines of code with file paths to the appropriate files as created above.

3. Uncomment the "conductance()" function at the bottom of the script (line 639) and execute

<u>Fast inactivation</u>

1. Create one input file, and save as .csv:

"Inactivation data" – headers must be the unique cell identifier. The first column should be a list of test potentials (mV). Successive columns should be a list of *normalized* current amplitudes corresponding to peak currents following an inactivating prepulse of the given test potential. Blank columns are okay.

2. Replace labeled lines of code with file paths to the appropriate file as created above.

3. Uncomment the "inactivation()" function at the bottom of the script (line 638) and execute

<u>Recovery from Inactivation</u>

1. Create one input file, and save as .csv:

"Recovery from Inactivation data" – headers must be the unique cell identifier. The first column should be a list of times between pulse 1 and pulse 2 (ms), and have the header "Time (ms)". Successive columns should be a list of ratios that compare amplitudes of pulse 1 and pulse 2. Blank columns are okay.

2. Replace labeled lines of code with file paths to the appropriate file as created above.

3. Uncomment the RecInactivation() function at the bottom of the script (line 642) and execute.

<u>Process</u>

1. Collect input files. These files are the ones generated by the above modules. Collect both Data files as well as fit parameter files.

2. Create a capacitance file for each parameter. This data would be taken from the automated patch clamp machine, saved as a .csv, and formatted as follows:
Headers must be the unique cell identifier. They can be in a different order than in the Data and extras are okay. Cell A2 should be labeled “Cap”, and successive values in Row 2 should indicate the capacitance of each cell. Blank columns are okay.
3. Comment out unused modules. For example, if one did not have Fast Inactivation data, all lines corresponding to “FIN” or “Inactivation” within the process function should be commented.
4. Replace labeled lines of code with file paths to the appropriate file as created above. File paths must be given for Data files, Parameter files, Capacitance files, and, for activation, the Reversal potential file.
5. Uncomment the “process()” function at the bottom of the script (line 641) and execute.

AddendRecInactivation

1. Create a file for the externally generated fit parameters for Recovery from Inactivation data. The file should be saved as .csv and formatted as follows:
Headers must be the unique cell identifier. They can be in a different order than in the data and extras are okay. The first column should be the names of the fit parameters. Subsequent columns should include the numerical values of each parameter.
2. Replace labeled lines of code with file paths to the appropriate file as created above. Structures exist to handle two sets of recovery from inactivation data (for example, data with pulses of different duration). If only one set of data exists, comment out the other structure (lines containing code marked by prefixes “short” or “long).
3. Uncomment the “AddendRecInactivation()” function at the bottom of the script (line 643) and execute.

Free text of the code is included below. The script can also be accessed at:
https://github.com/emrekiziltug3/AutomatedClampAnalysis
DOI: 10.5281/zenodo.7411103

```
'''
@author: Sidharth Tyagi
v1.2 - 6-27-2022
'''
# -*- coding: utf-8 -*-

from dask.array.routines import transpose
import os

#Set directory for outputs
path = 'XXXXXXXXX'
os.chdir(path)

#Process function concatenates results from files generated by other functions
def process():
    import pandas as pd
    import numpy as np
    import os
    import glob
    import matplotlib.pyplot as plt

    FIN_data = pd.read_csv('INSERT PATH FOR FIN DATA HERE')
    ACT_data = pd.read_csv('INSERT PATH FOR ACTIVATION DATA HERE')
    ACT_data.dropna(how='all', axis = 1, inplace = True)
    ACT_data = ACT_data.iloc[: , 1:]
    FIN_data = FIN_data.iloc[: , 1:]

    ActivationCap = pd.read_csv('INSERT PATH TO FILE WITH ACTIVATION CAPACITANCES HERE')
    ActivationCap.rename(columns = {list(ActivationCap)[0]:'Variable'}, inplace=True)
```

```
   ActivationParam = pd.read_csv('INSERT PATH TO FILE WITH ACTIVATION FIT PARAMETERS HERE')
   ActivationParam.rename(columns = {list(ActivationParam)[0]:'Variable'}, inplace=True)
   ActivationVrev = pd.read_csv('INSERT PATH TO FILE WITH ACTIVATION REVERSAL POTENTIALS
HERE')
   ActivationVrev.rename(columns = {list(ActivationVrev)[0]:'Variable'}, inplace=True)

   InactivationCap = pd.read_csv('INSERT PATH TO FILE WITH FAST INACTIVATION CAPACITANCES
HERE')
   InactivationCap.rename(columns = {list(InactivationCap)[0]:'Variable'}, inplace=True)
   InactivationParam = pd.read_csv('INSERT PATH TO FILE WITH FAST INACTIVATION FIT PARAMETERS
HERE')
   InactivationParam.rename(columns = {list(InactivationParam)[0]:'Variable'}, inplace=True)

   nan_value = float("NaN")
   ActivationVrev.replace("", nan_value, inplace = True)
   ActivationVrev.dropna(how='all', axis = 1, inplace = True)
   ActivationVrev.dropna(how='all', axis = 0, inplace = True)
   ActivationVrevNames = ActivationVrev.columns.values.tolist()
   NewCol = []
   for string in ActivationVrevNames:
      new_string = string.replace(":", "_")
      NewCol.append(new_string)
   ActivationVrev.columns = NewCol

   ActivationCap.replace("", nan_value, inplace = True)
   ActivationCap.dropna(how='all', axis = 1, inplace = True)
   ActivationCap.dropna(how='all', axis = 0, inplace = True)
   ActivationCapNames = ActivationCap.columns.values.tolist()
   NewCol = []
   for string in ActivationCapNames:
      new_string = string.replace(":", "_")
      NewCol.append(new_string)
   ActivationCap.columns = NewCol
   ActivationCap.at[0,"Variable"] = "Activation cap"

   ActivationCap.replace(regex=True, inplace=True, to_replace=r'[^0-9.\-]', value=r'')
   ActivationCap = ActivationCap.astype('float64')

   InactivationCap.replace("", nan_value, inplace = True)
   InactivationCap.dropna(how='all', axis = 1, inplace = True)
   InactivationCap.dropna(how='all', axis = 0, inplace = True)
   InactivationCapNames = InactivationCap.columns.values.tolist()
   NewCol = []
   for string in InactivationCapNames:
      new_string = string.replace(":", "_")
      NewCol.append(new_string)
   InactivationCap.columns = NewCol
   InactivationCap.at[0,"Variable"] = "Inactivation cap"

   vertical_stack = pd.concat([ActivationCap, ActivationParam, ActivationVrev],axis = 0)
   vertical_stack = vertical_stack.append(pd.Series("", index=vertical_stack.columns), ignore_index=True)
   vertical_stack = vertical_stack.append(pd.Series("", index=vertical_stack.columns), ignore_index=True)
   vertical_stack = pd.concat([vertical_stack,InactivationCap,InactivationParam],axis = 0)
   vertical_stack = pd.concat([InactivationCap,InactivationParam],axis = 0)
   vertical_stack = vertical_stack.append(pd.Series("", index=vertical_stack.columns), ignore_index=True)
   vertical_stack = vertical_stack.append(pd.Series("", index=vertical_stack.columns), ignore_index=True)
```

```
    vertical_stack = pd.concat([vertical_stack,ACT_data],axis = 0)
    vertical_stack = vertical_stack.append(pd.Series("", index=vertical_stack.columns), ignore_index=True)
    vertical_stack = vertical_stack.append(pd.Series("", index=vertical_stack.columns), ignore_index=True)
    vertical_stack = pd.concat([vertical_stack,FIN_data],axis = 0)

    vertical_stack.to_csv('INSERT DESIRED FILE NAME FOR MASTERDATA HERE')
    #File will output to previously designated path
    return 0

#Fast inactivation
def inactivation ():
    import pandas as pd
    import numpy as np
    import os
    import glob
    import matplotlib.pyplot as plt

    InactivationDf = pd.read_csv('INSERT PATH TO FILE WITH FAST INACTIVATION DATA HERE')
    nan_value = float("NaN")
    InactivationDf.replace("", nan_value, inplace = True)
    InactivationDf.dropna(how='all', axis = 1, inplace = True)
    InactivationColNames = InactivationDf.columns.values.tolist()
    NewCol = []
    for string in InactivationColNames:
        new_string = string.replace(":", "_")
        NewCol.append(new_string)
    InactivationDf.columns = NewCol
    InactivationDf.replace(regex=True, inplace=True, to_replace=r'[^0-9.\-]', value=r'')
    InactivationDf = InactivationDf.astype('float64')

    MembPotent = InactivationDf['Membrane Potential (mV)']
    InactivationDf['Membrane Potential (mV)'] = MembPotent
    InactivationDfColumns = InactivationDf.columns.values.tolist()
    InactivationDf = InactivationDf[InactivationDfColumns[1:]]

    Initial = InactivationDf.head(4)
    End = InactivationDf.tail(4)
    InitialMean = pd.DataFrame(Initial.mean())
    EndMean = pd.DataFrame(End.mean())
    Diff = InitialMean - EndMean
    Range = InactivationDf.max(axis=0) - InactivationDf.min(axis=0)
    Range = pd.DataFrame(Range)
    nordiff = Diff/Range
    nordiff = nordiff.transpose()
    output1 = InactivationDf
    output1.insert(0,'Membrane Potential (mV)',MembPotent)
    output1['Membrane Potential (mV)'] = 'FIN_' +  output1['Membrane Potential (mV)'].astype(str)
    output1.to_csv('Raw_normalized_FIN.csv')
    InactivationDf = InactivationDf[InactivationDfColumns[1:]]
    faulty = pd.DataFrame()
    for column in nordiff:
        if nordiff.at[0,column] <0.8:
            faulty = pd.concat((faulty, InactivationDf[column]), axis = 1)
            InactivationDf.drop(column, inplace=True, axis=1)
    output2 = InactivationDf
    output2.insert(0,'Membrane Potential (mV)',MembPotent)
```

```
    output2.to_csv('Excluded_normalized_FIN.csv')

    InactivationDf = InactivationDf.iloc[: , 1:]
    CurrStd = pd.DataFrame(InactivationDf.std(axis = 0, ddof=0))
    CurrStd = CurrStd.transpose()
    Volt = MembPotent
    from scipy import optimize
    def Boltzmann(x, v, k):
        return 1/(1+np.exp((-v+x)/k))

    v_0 = -70
    k_0 = 9

    def DoubleBoltzmann(x, y0, A, z1, z2, k1, k2, p):
        B = np.exp((x-z1)/k1)
        C = np.exp((x-z2)/k2)
        return y0+ A*(p/(1+B)+(1-p)/(1+C))

    y0_0 = 1
    A_0 = 33
    z1_0 = -62
    z2_0 = -50
    k1_0 = -8.3
    k2_0 = -3.2
    p_0 = 0.9
    floatlist = []
    floatlist2 = []
    SingleBoltFitParams = pd.DataFrame()
    DoubleBoltFitParams = pd.DataFrame()
    print('xx')
    for column in InactivationDf:
        try:
            Curr = InactivationDf[column].values
            params, params_covariance = optimize.curve_fit(Boltzmann, Volt, Curr, p0 = ([v_0, k_0]))
            plt.figure(figsize = (6,4))
            plt.scatter(Volt, Curr, label = InactivationDf[column].name)
            plt.plot(Volt, Boltzmann(Volt, params[0], params[1]), label = 'Single Boltzmann')
            plt.legend(loc='best')
            plt.savefig(column +'_single_FIN.png')
            diff = Curr - Boltzmann(Volt, params[0], params[1])
            rms = (np.sqrt(np.sum(np.square(diff)/len(diff))))
            normrms = rms / CurrStd.at[0,column]
            floatlist.append(normrms)
            params = pd.DataFrame(params)
            SingleBoltFitParams = pd.concat((SingleBoltFitParams, params), axis = 1)
            params2, params_covariance = optimize.curve_fit(DoubleBoltzmann, Volt, Curr, p0 = ([y0_0, A_0, z1_0,
z2_0, k1_0, k2_0, p_0]), maxfev = 300)
            plt.figure(figsize = (6,4))
            plt.scatter(Volt, Curr, label = InactivationDf[column].name)
            plt.plot(Volt, DoubleBoltzmann(Volt, params2[0], params2[1], params2[2], params2[3], params2[4],
params2[5], params2[6]), label = 'Double Boltzmann')
            plt.legend(loc='best')
            plt.savefig(column +'_double_FIN.png')
            diff2 = Curr - DoubleBoltzmann(Volt, params2[0], params2[1], params2[2], params2[3], params2[4],
params2[5], params2[6])
            rms2 = (np.sqrt(np.sum(np.square(diff2)/len(diff2))))
```

```
            normrms2 = rms2 / CurrStd.at[0,column]
            floatlist2.append(normrms2)
            params2 = pd.DataFrame(params2)
            DoubleBoltFitParams = pd.concat((DoubleBoltFitParams, params2), axis = 1)

        except:
            string = 'Failed'
            faileddf = pd.DataFrame([string])
            floatlist2.append('Failed')
            DoubleBoltFitParams = pd.concat((DoubleBoltFitParams, faileddf), axis = 1)
            pass

    print(floatlist)
    print(floatlist2)
    bin = []
    for i in range(len(floatlist)):
        if floatlist2[i] == "Failed":
            bin.append('single')
        elif floatlist2[i] < floatlist[i]:
            bin.append('double')
        else:
            bin.append('single')
    ID = pd.DataFrame({"FIN_Single Boltzmann":floatlist,"FIN_Double Boltzmann":floatlist2,"FIN_Bin":bin})
    ID = ID.transpose()
    ID.columns = InactivationDf.columns.values.tolist()
    DoubleBoltFitParams.columns = InactivationDf.columns.values.tolist()
    DoubleBoltFitParams = DoubleBoltFitParams.rename(index={0:'FIN_y0', 1:'FIN_A', 2:'FIN_z1', 3:'FIN_z2',
4:'FIN_k1', 5:'FIN_k2', 6:'FIN_p'})
    SingleBoltFitParams.columns = InactivationDf.columns.values.tolist()
    SingleBoltFitParams = SingleBoltFitParams.rename(index={0:'FIN_v',1:'FIN_k'})
    ID_final = pd.concat([ID, SingleBoltFitParams,DoubleBoltFitParams], axis=0)
    ID_final.to_csv('Fit_parameters_FIN.csv')
    return 0

# Activation
def conductance():
    import pandas as pd
    import numpy as np
    import os
    import glob
    import matplotlib.pyplot as plt
    import chardet

    with open('INSERT PATH TO FILE WITH ACTIVATION DATA HERE', 'rb') as rawdata:
        result = chardet.detect(rawdata.read(100000))
    PeakDf = pd.read_csv('INSERT PATH TO FILE WITH ACTIVATION DATA HERE')
    PeakColNames = PeakDf.columns.values.tolist()
    NewCol = []
    for string in PeakColNames:
        new_string = string.replace(":", "_")
        NewCol.append(new_string)
    PeakDf.columns = NewCol
    # Optional - number of rows to drop if desired
    # nrows = 4
    # PeakDf = PeakDf.iloc[:-nrows,:]
    PeakDf.replace(regex=True, inplace=True, to_replace=r'[^0-9.\-]', value=r'')
```

```
    PeakDf = PeakDf.astype('float64')
    ParamDf = pd.read_csv('INSERT PATH TO FILE WITH ACTIVATION REVERSAL POTENTIALS
HERE.csv')
    NewCol[0] = 'IDENTIFIER'
    ParamDf.columns = NewCol
    ParamDf.replace(to_replace = ["Failed"], value = 0.00002, inplace = True)
    PeakDf_Copy = PeakDf.copy()
    PeakDfColumns = PeakDf_Copy.columns.values.tolist()
    #Creates membrane potential dataframe
    MembPotent = PeakDf.iloc[:, 0]
    MembDf = pd.DataFrame(MembPotent)

    for i in range(2,len(PeakDfColumns)):
        MembDf['Membrane Potential'+str(i)] = MembDf.iloc[:, 0]

    #Renames columns
    MembDf.columns = PeakDfColumns[1:]
    #Creates dataframes containing reversal potential and capacitance parameters while preserving column headers
    Vrev = pd.DataFrame(columns=ParamDf.columns)
    Cap = pd.DataFrame(columns=ParamDf.columns)
    RevRows = ParamDf.loc[0, :]
    Vrev = Vrev.append(RevRows, ignore_index = True)
    CapRows = ParamDf.loc[1, :]
    Cap = Cap.append(CapRows, ignore_index = True)
    # PeakDf_Copy = PeakDf[PeakDfColumns[1:]].div(PeakDf['Membrane Potential'],axis=0)
    PeakDf_Copy = PeakDf[PeakDfColumns[1:]]
    RevDf = pd.DataFrame(columns = PeakDfColumns[1:]).reindex_like(PeakDf_Copy).dropna()
    RevDf = RevDf.append(Vrev, ignore_index = False)
    del RevDf['IDENTIFIER']
    for i in range (1, len(MembPotent)):
        RevDf.loc[i] = RevRows
    DenomDf = MembDf - RevDf
    CondDf = PeakDf_Copy/DenomDf
    NormCond = (CondDf-CondDf.min())/(CondDf.max()-CondDf.min())
    output1 = NormCond
    output1.insert(0,'Membrane Potential (mV)',MembPotent)
    output1['Membrane Potential (mV)'] = 'ACT_' +  output1['Membrane Potential (mV)'].astype(str)
    output1.to_csv('INSERT FILE NAME FOR ALL NORMALIZED CONDUCTANCE DATA')
    NormCond = NormCond.iloc[: , 1:]
    NormCond = NormCond.dropna(axis=1, how='all')
    VoltError = pd.DataFrame()
    #Correction for voltage error
    for column in NormCond:
        if ((NormCond[column].diff() > 0.4).any()):
            VoltError = pd.concat((VoltError, NormCond[column]), axis = 1)
            NormCond.drop(column, inplace=True, axis=1)
    for column in NormCond:
        if ((NormCond[column].diff() < -0.4).any()):
            VoltError = pd.concat((VoltError, NormCond[column]), axis = 1)
            NormCond.drop(column, inplace=True, axis=1)
    output2 = NormCond
    output2.insert(0,'Membrane Potential (mV)',MembPotent)
    output2.to_csv('INSERT FILE NAME FOR ALL NORMALIZED CONDUCTANCE DATA AFTER
VOLTAGE ERROR FILTERING')
    NormCond = NormCond.iloc[: , 1:]
    from scipy import optimize
```

```
    def Boltzmann(x, v, k):
        return 1/(1+np.exp((v-x)/k))

    v_0 = -30
    k_0 = 9

    def DoubleBoltzmann(x, y0, A, z1, z2, k1, k2, p):
        B = np.exp((x-z1)/k1)
        C = np.exp((x-z2)/k2)
        return y0+ A*(p/(1+B)+(1-p)/(1+C))

    y0_0 = 0.009
    A_0 = 33
    z1_0 = -42
    z2_0 = -30
    k1_0 = -8.3
    k2_0 = -3.2
    p_0 = 0.9

    #Creating empty structures for RMSD calculations - Single Boltzmann
    print(NormCond)
    CurrStd = pd.DataFrame(NormCond.std(axis = 0, ddof=0))
    CurrStd = CurrStd.transpose()
    CurrMean = pd.DataFrame(NormCond.mean())
    CurrMean = CurrMean.transpose()
    SingleBoltFitParams = pd.DataFrame()
    floatlist = []
    Volt = MembPotent
    #Creating empty structures for RMSD calculations - Double Boltzmann
    DoubleBoltFitParams = pd.DataFrame()
    floatlist2 = []
    print (NormCond)
    for column in NormCond:
        try:
            Curr = NormCond[column].values
            params, params_covariance = optimize.curve_fit(Boltzmann, Volt, Curr, p0 = ([v_0, k_0]))
            plt.figure(figsize = (6,4))
            plt.scatter(Volt, Curr, label = NormCond[column].name)
            plt.plot(Volt, Boltzmann(Volt, params[0], params[1]), label = 'Single Boltzmann')
            plt.legend(loc='best')
            plt.savefig(column +'_single_conductance2.png')
            diff = Curr - Boltzmann(Volt, params[0], params[1])
            rms = (np.sqrt(np.sum(np.square(diff)/len(diff))))
            normrms = rms / CurrStd.at[0,column]
            floatlist.append(normrms)
            params = pd.DataFrame(params)
            SingleBoltFitParams = pd.concat((SingleBoltFitParams, params), axis = 1)
            params2, params_covariance = optimize.curve_fit(DoubleBoltzmann, Volt, Curr, p0 = ([y0_0, A_0, z1_0,
z2_0, k1_0, k2_0, p_0]), maxfev = 300)
            plt.figure(figsize = (6,4))
            plt.scatter(Volt, Curr, label = NormCond[column].name)
            plt.plot(Volt, DoubleBoltzmann(Volt, params2[0], params2[1], params2[2], params2[3], params2[4],
params2[5], params2[6]), label = 'Double Boltzmann')
            plt.legend(loc='best')
            plt.savefig(column +'_double_conductance2.png')
```

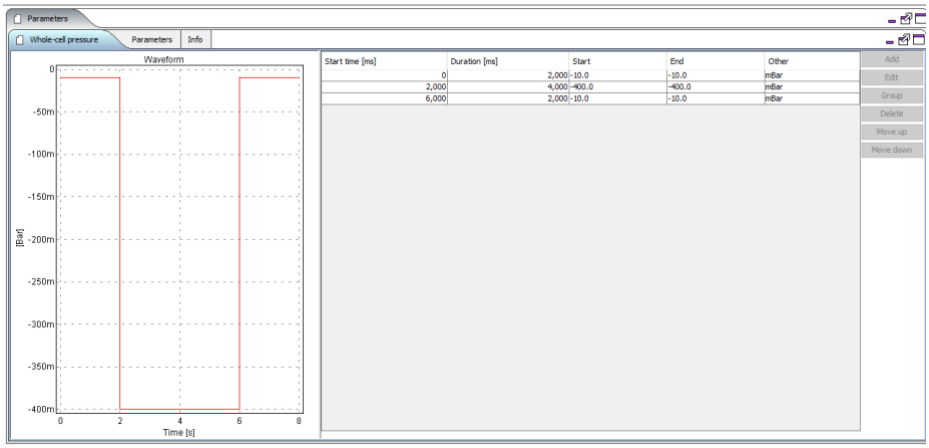
```
            diff2 = Curr - DoubleBoltzmann(Volt, params2[0], params2[1], params2[2], params2[3], params2[4],
params2[5], params2[6])
            rms2 = (np.sqrt(np.sum(np.square(diff2)/len(diff2))))
            normrms2 = rms2 / CurrStd.at[0,column]
            floatlist2.append(normrms2)
            params2 = pd.DataFrame(params2)
            DoubleBoltFitParams = pd.concat((DoubleBoltFitParams, params2), axis = 1)

        except:
            string = 'Failed'
            faileddf = pd.DataFrame([string])
            floatlist2.append('Failed')
            DoubleBoltFitParams = pd.concat((DoubleBoltFitParams, faileddf), axis = 1)
            pass
    ColNames = NormCond.columns.values.tolist()
    print(ColNames)
    bin = []
    for i in range(len(floatlist)):
        if floatlist2[i] == "Failed":
            bin.append('single')
        elif floatlist2[i] < floatlist[i]:
            bin.append('double')
        else:
            bin.append('single')
    ID2 = pd.DataFrame({"ACT_Single Boltzmann":floatlist,"ACT_Double Boltzmann":floatlist2,"ACT_Bin":bin})
    ID2 = ID2 .transpose()
    ID2.columns = ColNames[:]
    DoubleBoltFitParams.columns = ColNames[:]
    DoubleBoltFitParams = DoubleBoltFitParams.rename(index={0:'ACT_y0', 1:'ACT_A', 2:'ACT_Va', 3:'ACT_Vb',
4:'ACT_k1', 5:'ACT_k2', 6:'ACT_p'})
    SingleBoltFitParams.columns = ColNames[:]
    SingleBoltFitParams = SingleBoltFitParams.rename(index={0:'ACT_V0.5',1:'ACT_k'})
    ID_final = pd.concat([ID2, SingleBoltFitParams,DoubleBoltFitParams], axis=0)
    ID_final.to_csv('INSERT FILE NAME FOR PROCESSED ACTIVATION FIT PARAMETERS')

#Recovery from Inactivation
def RecInactivation ():
    import pandas as pd
    import numpy as np
    import os
    import glob
    import matplotlib.pyplot as plt
    RecInactivationDf = pd.read_csv('INSERT PATH TO FILE WITH RECOVERY FROM INACTIVATION
DATA')
    nan_value = float("NaN")
    RecInactivationDf.replace("", nan_value, inplace = True)
    RecInactivationDf.dropna(how='all', axis = 1, inplace = True)
    RecInactivationColNames = RecInactivationDf.columns.values.tolist()
    NewCol = []
    for string in RecInactivationColNames:
        new_string = string.replace(":", "_")
        NewCol.append(new_string)
    RecInactivationDf.columns = NewCol
    RecInactivationDf.replace(regex=True, inplace=True, to_replace=r'[^0-9.\-]', value=r'')
    RecInactivationDf = RecInactivationDf.astype('float64')
    Times = RecInactivationDf['Time (ms)']
```

```
    RecInactivationDfColumns = RecInactivationDf.columns.values.tolist()
    RecInactivationDf = RecInactivationDf[RecInactivationDfColumns[1:]]

    from scipy import optimize
    def DoubleExp(x, a, b, c, d):
        return (a*np.exp(b*x)+c*np.exp(d*x))

    # Starting parameters
    # a_0 = -70
    # b_0 = 9
    # c_0 = 0
    # d_0 = 0

    floatlist = []
    DoubleExpFitParams = pd.DataFrame()

    for column in RecInactivationDf:
        try:
            Curr = RecInactivationDf[column].values
            params, params_covariance = optimize.curve_fit(DoubleExp, Times, Curr)
            plt.figure(figsize = (6,4))
            plt.scatter(Volt, Curr, label = RecInactivationDf[column].name)
            plt.plot(Volt, Boltzmann(Volt, params[0], params[1], params[2], params[3]), label = 'Double Exponential')
            plt.legend(loc='best')
            plt.savefig(column +'_REC.png')
            plt.show()
            diff = Curr - DoubleExp(Times, params[0], params[1], params[2], params[3])
            rms = (np.sqrt(np.sum(np.square(diff)/len(diff))))
            normrms = rms / CurrStd.at[0,column]
            floatlist.append(normrms)
            params = pd.DataFrame(params)
            DoubleExpFitParams = pd.concat((SingleBoltFitParams, params), axis = 1)
        except:
            pass

    return 0

# If recovery from inactivation data is fit in another program, this function allows for appending those results to the
master data
def AddendRecInactivation ():
    import pandas as pd
    import numpy as np
    import os
    import glob
    import matplotlib.pyplot as plt

    MasterFile = pd.read_csv('INSERT PATH TO FILE WITH MASTER DATA, GENERATED BY PROCESS
FUNCTION')
    ShortRecParam = pd.read_csv('INSERT PATH TO FILE WITH SHORT RECOVERY FROM INACTIVATION
FITS')
    ShortRecParam.rename(columns = {list(ShortRecParam)[0]:'Variable'}, inplace=True)
    nan_value = float("NaN")
    ShortRecParam.replace("", nan_value, inplace = True)
    ShortRecParam.dropna(how='all', axis = 1, inplace = True)
    ShortRecParam.dropna(how='all', axis = 0, inplace = True)
    ShortRecParamNames =  ShortRecParam.columns.values.tolist()
```

```
    NewCol = []
    for string in  ShortRecParamNames:
        new_string = string.replace(":", "_")
        NewCol.append(new_string)
    ShortRecParam.columns = NewCol
    ShortRecParam.at[0,"Variable"] = "XX ms Recovery from Inactivation"

    LongRecParam = pd.read_csv('INSERT PATH TO FILE WITH LONG RECOVERY FROM INACTIVATION
FITS')
    LongRecParam.rename(columns = {list(LongRecParam)[0]:'Variable'}, inplace=True)
    nan_value = float("NaN")
    LongRecParam.replace("", nan_value, inplace = True)
    LongRecParam.dropna(how='all', axis = 1, inplace = True)
    LongRecParam.dropna(how='all', axis = 0, inplace = True)
    LongRecParamNames =  LongRecParam.columns.values.tolist()
    NewCol = []
    for string in  LongRecParamNames:
        new_string = string.replace(":", "_")
        NewCol.append(new_string)
    LongRecParam.columns = NewCol
    LongRecParam.at[0,"Variable"] = "XXX ms Recovery from Inactivation"

    vertical_stack = MasterFile.append(pd.Series("", index=MasterFile.columns), ignore_index=True)
    vertical_stack = vertical_stack.append(pd.Series("", index=vertical_stack.columns), ignore_index=True)
    vertical_stack = pd.concat([vertical_stack,ShortRecParam],axis = 0)
    vertical_stack = vertical_stack.append(pd.Series("", index=vertical_stack.columns), ignore_index=True)
    vertical_stack = pd.concat([vertical_stack,LongRecParam],axis = 0)
    vertical_stack.to_csv('INSERT FILE NAME FOR MASTER DATA WITH RECOVERY FROM
INACTIVATION')

# Uncomment name of function to run
if __name__ == '__main__':
    # inactivation()
    # conductance()
    # process()
    # RecInactivation()
    # AddendRecInactivation()
```

2 Supplementary Figures.

A Screenshot of the Suction Setting



Screenshot of Pressures

B

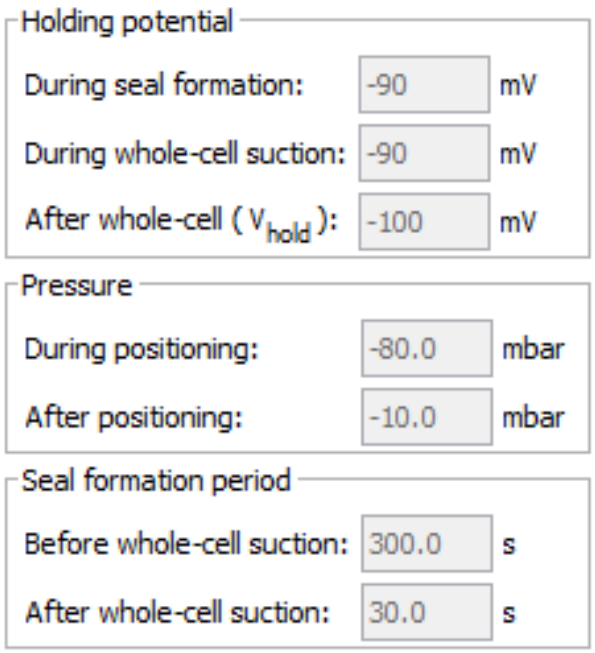
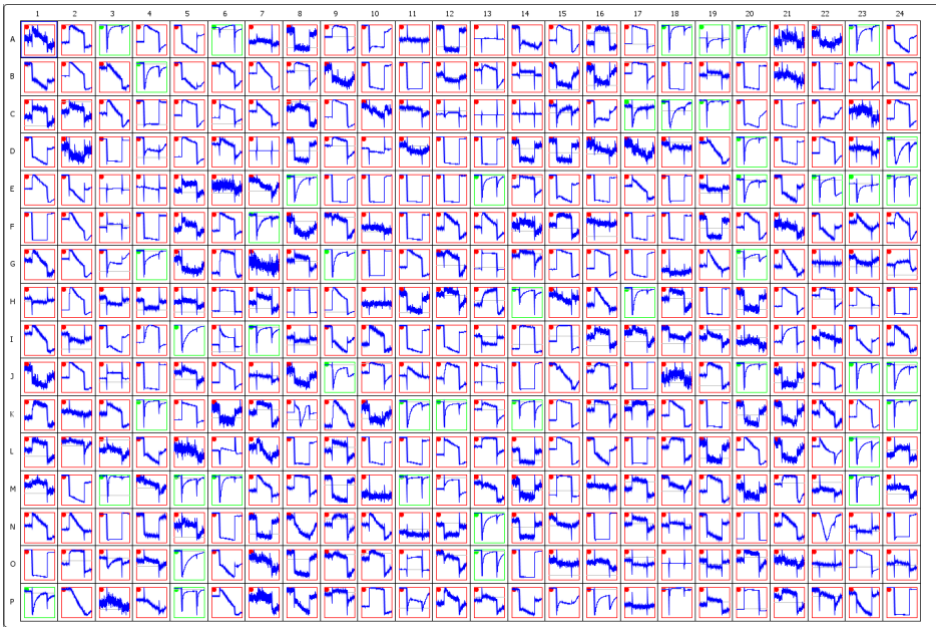


Figure S1 – Suction protocol, related to STAR Methods.

A) The suction setting, we designed on the Qube. B) The suction pressures that were used to form seals and obtain whole-cell configuration.

A

Screenshot of a Sample Qube Run



Distribution of Series Resistances

B

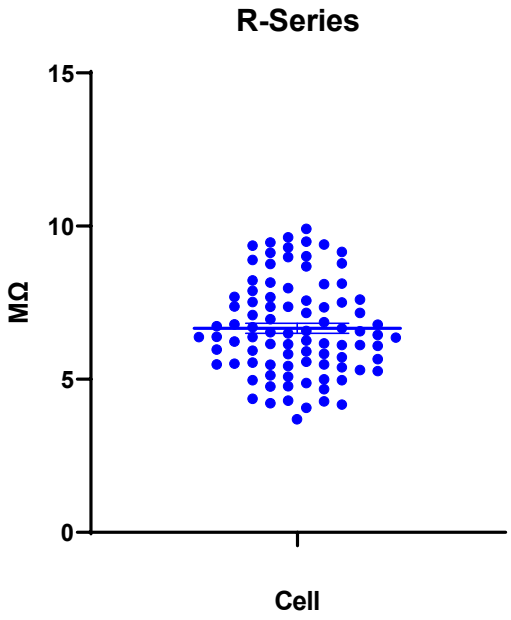


Figure S2 – Sample run and series resistance, related to STAR Methods.

A) Screenshot of a sample Qube run at one of the initial square pulse experimental periods (-120 to -20 mV). The wells that display sodium currents are highlighted in green. B) Distribution of the series resistances for each of the cells.

A

Name	Unit	Min Value	Max Value	Enabled
Liquid Period Filter Pass Count		1.00		☑
C-slow Exp. Average	pF	10.00		☐
1stLP[V½(IV-500ms_Inactivation)]	mV	-110.00	-5.00	☐
4thLP[Avg(It-120 mV Hold)]	pA		0.00	☐
1stLP[FitQual(IV-500ms_Inactivation)]			0.20	☐
4thLP[Avg(-120 mV Tau)]	ms		20.00	☐
4thLP[Avg(-120 mV Tau (FitQuality))]			0.20	☐
1stLP1[FitQual(IV-Activation)]			0.20	☐
1stLP1[V½(IV-Activation)]	mV	-60.00	-5.00	☐

B

Name	Unit	Min Value	Max Value
Sweep Filter Pass Count		1.00	
R-membrane Average for Period	MΩ	60.00	
R-series Maximum for Period	MΩ		10.00
C-slow Average for Period	pF	9.50	

Figure S3 – Analyzer filter screenshot, related to STAR Methods.

A) The experimental filters that were used to identify wells with quality recordings, as a first data filtration step. We used the “Enabled” tab to turn on or off a given filter with the specified ranges for a given experimental pulse protocol. B) The liquid period filters we used as a secondary filter to impose a quality check onto the cells. Each filter was enabled, when appropriate, for the analysis of a given parameter. These preceded the Python steps.

3 Supplementary Tables.

Number of adult mice per prep	3-4 mice
Number of preps tested	7: (5 for VC), (2 for CC + VC)
Number of cells patched at the beginning of experiment per prep	Up to ~80-100 cells
Activation – number of cells that passed filters	27 cells
Inactivation – number of cells that passed filters	87 cells
Recovery from 20 ms inactivation – number of cells that passed filters	83 cells
Recovery from 500 ms inactivation – number of cells that passed filters	82 cells
CC + VC – number of cells that passed filters	29 cells
Number of runs/prep	2-4
Viability of cells post dissection and dissociation for automated patch-clamp	Up to ~4 hours
Lifespan of cells while being patched	Up to ~40-50 minutes

Table S1. Characteristic features and output of the assay. The cell counts include all the cells that passed all our extensive quality and mathematical filtration steps, related to Table 1.