Supporting information
"The electrocatalytic activity for the hydrogen evolution reaction on alloys is determined by element-specific adsorption sites rather than *d*-band properties"

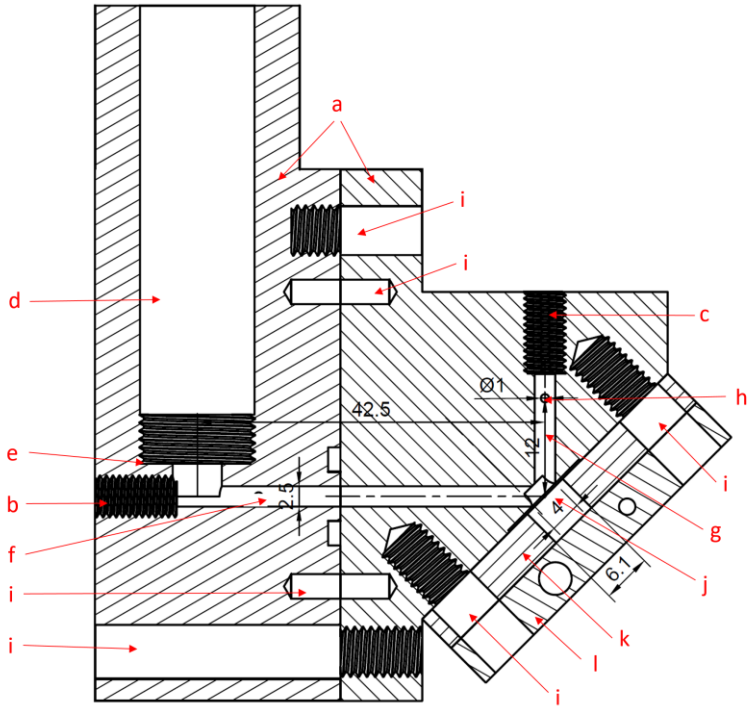
Maximilian Schalenbach*[a], Rebekka Tesch*[b,c,d], Piotr M. Kowalski[b,d], R-A. Eichel[a, e]

[a] Fundamental Electrochemistry (IEK-9), Institute of Energy and Climate Research, Forschungszentrum Jülich, Wilhelm-Johnen-Straße, 52425 Jülich, Germany
[b] Theory and Computation of Energy Materials (IEK-13), Institute of Energy and Climate Research, Forschungszentrum Jülich, Wilhelm-Johnen-Straße, 52425 Jülich, Germany
[c] Chair of Theory and Computation of Energy Materials, Faculty of Georesources and Materials Engineering, RWTH Aachen University, 52062 Aachen, Germany
[d] Jülich Aachen Research Alliance JARA Energy & Center for Simulation and Data Science (CSD), 52425 Jülich, Germany
[e] Institute of Physical Chemistry, RWTH Aachen University, 52062 Aachen, Germany
* *Corresponding authors: m.schalenbach@fz-juelich.de, r.tesch@fz-juelich.de*

Contents

1 Blueprint of the flow cell

Figure S1 shows a blueprint of the flow cell that was used for the electrochemical measurements.



a) Polypropylene main bodies of the cell
b) Electrolyte inlet
c) Electrolyte outlet
d) Compartment for the reference electrode
e) M14 fine-pitch thread to clamp a Nafion membrane
f) Electrolyte channel towards the specimen
g) Electrolyte channel towards the cell outlet
h) Hole for the Pt wire counter electrode
i) Mechanical connectors between different cell parts
j) Place for the metal specimen
k) Polypropylene plate to hold the metal specimen in place
l) Stainless steel contacting plate with 4 and 6 mm holes for banana jacks

Figure S1: Blueprint of the plastic body of the V-cell used for the electrochemical experiments. The numbers represent distances in mm. The letters refer to the description of the cell parts at the right side.

2 Manufacturing of the alloys

Figure S2 shows a schematic sketch of the in-house made apparatus used in the induction furnace to manufacture the alloys. A graphite crucible was used to meld the weighted Au and Pt powders. The quartz glass vessel that contained the graphite crucible was evacuated three times and in between flooded with Argon to reduce the oxygen content. The pure metal and alloy specimen were produced with the following procedure: The crucibles were heated to 1650°C for one minute to melt the metal powders. Au significantly evaporates at these temperatures, for which the graphite crucibles were covered with a graphite lid while an Ar pressure of 0.1 bar was applied. Afterwards, the temperature was lowered to 1580°C to reduce the Au evaporation. By resting at this temperature for 20 minutes, diffusion processes mix the liquid metals so that uniformly distributed compositions of the alloys were achieved. After cooling, the crucibles were heated once again to a temperature of 1500°C for one second and cooled as fast as possible under an Ar stream to reduce the recrystallization during the phase transition from the liquid to the solid phase. In the case of the Pt specimen, the crucible was heated to 1750°C for ten minutes without any following treatment.

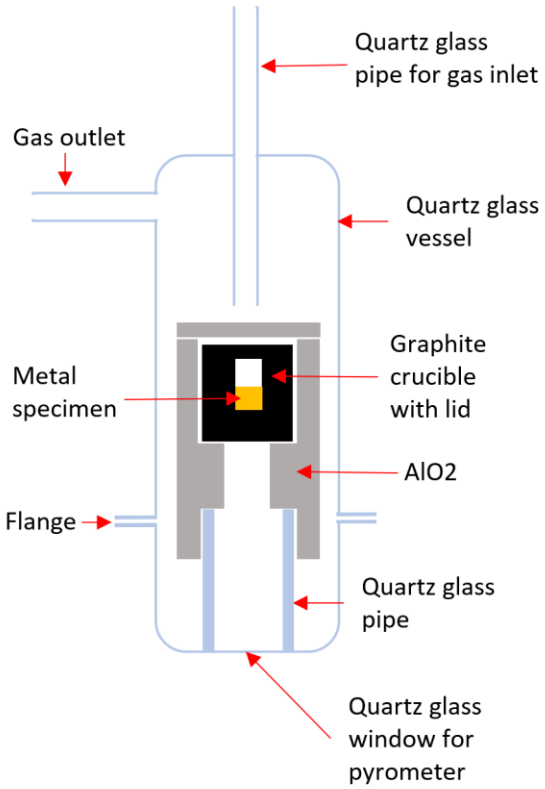


Figure S2: Schematic sketch of the apparatus to produce the alloy specimens from Au and Pt powders with an induction furnace. The quartz glass vessel was immersed in the coil of the induction furnace.

3 Composition analysis of the alloys

XPS is usually the method of choice to characterize the surface composition of alloys. However, Au and Pt are neighbors in the periodic table and thus their electron binding energies show little differences. Figure S3 shows EDS spectra around the two main peaks of Au and Pt of five of the alloys obtained with an Elite Super Detector (EDAX). The Pt peaks mingle with the Au peaks. A trend is visible, however, a quantitative analysis is error prone. For the case of the 11.2 % specimen, the "Apex Advanced v. 2.5.1001.0001" software for the EDS analysis showed a relative error of more than 100% for the Pt content.

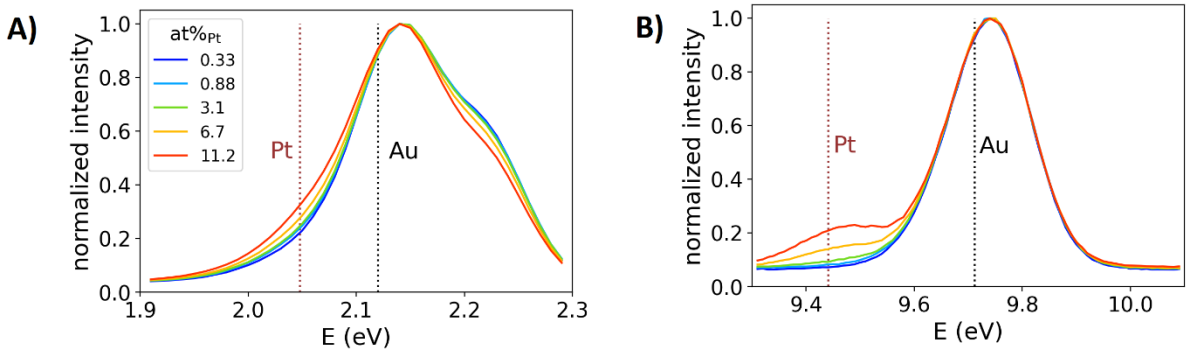


Figure S3: EDS data of the alloy specimens. A) Spectrum for lower electronic binding energies. B) Spectrum for higher electron bindings.

As an alternative, the near-surface composition was determined with a power probe extracted from the surface preparation via inductively coupled plasma optical emission spectroscopy (ICP-OES).

Hereto, the specimens' surfaces were filed with a fine and clean iron file to produce powders. The powders were dissolved in aqua regia on a heating plate with a temperature of 80°C. Table S1 shows the weight percentages of the Au and Pt contents of the sample determined with ICP-OES and the thereon calculated atomic weight percentages.

Table S1: The weight percent [wt%] of the Au and Pt content measured with the ICP-OES analysis. The Pt content in [at%] was calculated based on the weight composition and the error was detmerined with the propagation of error.

Specimen name	**Au**		**Pt**		**Pt**	
	mean [wt%]	**sdw [wt%]**	**mean [wt%]**	**std [wt%]**	**mean [at%]**	**err [at%]**
Au	100	3	<0,002		0	
0.02%$_{Pt}$	98.5	1	0.0207	0.0006	0.021	0.0006
0.33 %$_{Pt}$	99.1	1	0.325	0.005	0.33	0.0061
0.88 %$_{Pt}$	98.9	1.1	0.867	0.008	0.88	0.013
3.1 %$_{Pt}$	98.09	0.99	3.12	0.07	3.11	0.074
6.7 %$_{Pt}$	91.9	0.9	6.51	0.05	6.67	0.078
11.2 %$_{Pt}$	89.3	0.4	11.13	0.04	11.18	0.057

4 XRD characterization

Figure S4 shows the XRD data of powders from six of the eight different specimens that were examined in the article. The powders were filed off the metal specimens with a fine file. This file was thoroughly cleaned between preparing powders of different specimens. For the XRD measurements, the powders were placed between two plastic foils. The diffraction patterns were measured with a Panalytical Empyrean X-ray diffractometer using a Mo X-ray source. The highest angular resolution of the device was employed with a step size 0,002°. At each step, a time 350 s was used to measure the diffraction pattern.

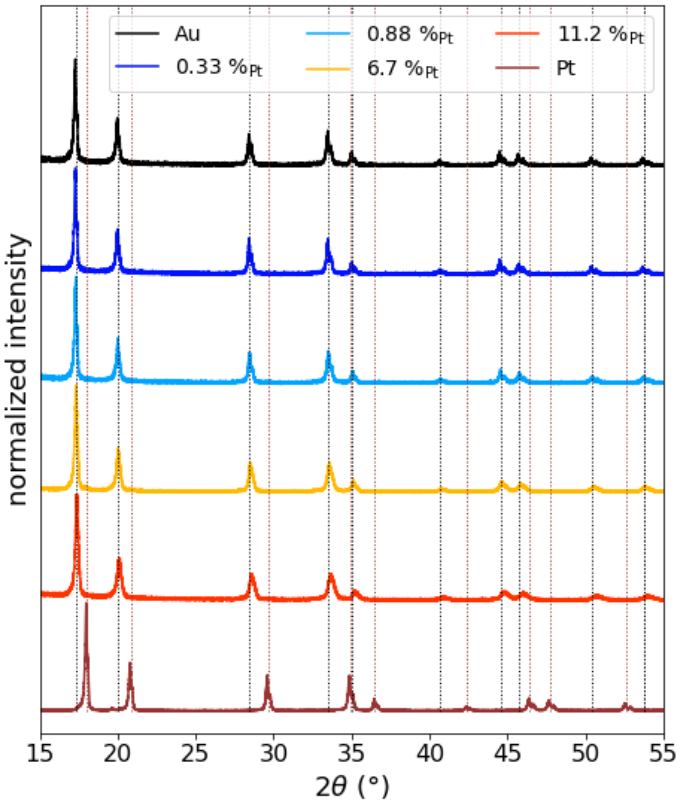


Figure S4: Measured XRD diffraction pattern of powders of the Au, Pt, and four alloys specimen. The black dotted line and the brown dotted line indicate reference peak positions for Au and Pt, respectively.

Au and Pt both have a fcc lattice. The 2θ values of Pt are approximately 4% higher than those of Au. In the alloys, the Au peak positions are slightly shifted towards those of the Pt peaks. However, distinct peaks at the Pt positions do not appear in the diffraction pattern of the alloys, for which most of the Pt is dissolved in the same fcc matrix as the Au. Hence, the thermally manufactured alloys are homogenous without significant formation of non-dissolved pure Pt phases, in agreement to the phase diagram.

5 Tafel analysis of the Au and Pt specimen

Figure S5 shows a detailed graph on the experimental data of the Au and Pt specimens with Tafel fits.

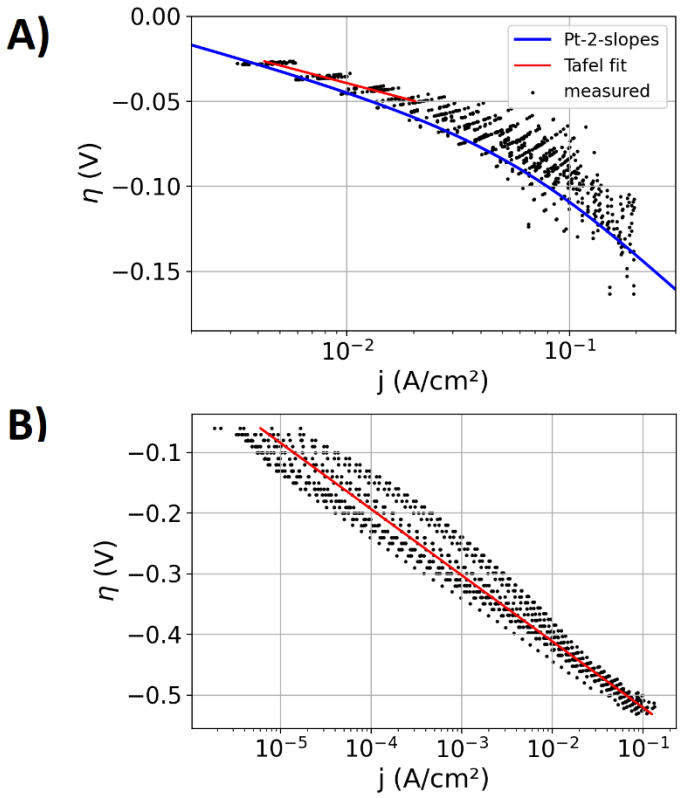


Figure S5: Tafel plots and fits of the pure Pt (A) and Au (B) specimens, respectively. Blue dots: Measured data. Solid red lines: Linear fits. Black solid line: Results of the Pt-2-slope description

Table S2 shows the parameters of these Tafel fits, referring to Tafel equation stated in the main text.

Table S2:Tafel fit parameters for the Au and Pt specimen, respectively.

	j_0 (µA/cm²)	a (1/V)
Au	1.69	21.11
Pt	734.8	66.48

The current at the Pt component $j_{\mathrm{Pt}}(\eta)$ with two Tafel slopes was calculated by

$$j_{\mathrm{Pt}}(\eta) = B\,\frac{j_0^{\mathrm{Pt}} e^{-a^{\mathrm{Pt}}\eta}}{j_0^{\mathrm{Au}} e^{-a^{\mathrm{Au}}\eta\, C} + B}\,, \tag{S1}$$

where the factor $B = 30\ \mu\mathrm{A/cm^2}$ and $C = 0.45$ denote empirical parameters (the other variables are defined in the main article). Both parameters were chosen so that $j_{\mathrm{Pt}}(\eta)$ agrees with the measured data (see main text).

6 Codes for the interpolation procedure and data treatment

The following python code is extracted from a jupyter notebook including the output.

Import of Python libraries and define graph style¶

In [1]:

```
# load python libraries
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import matplotlib
from scipy.optimize import curve_fit

# Edit graph style
matplotlib.rc('xtick', labelsize=16)
matplotlib.rc('ytick', labelsize=16)

font1 = {'family' : 'normal',
        'weight' : 'normal',
        'size'   : 18}
matplotlib.rc('font', **font1)
matplotlib.rcParams["figure.figsize"] = (6,4)
props_legend = {'size': 12}
txt_box_props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
```

Notations¶

- j: current density
- eta: overpotential
- df: DataFrame
- *_array: An Array data type
- *_interpol: Notates interpolated data
- *_TE: Notates data obtained by the Tafel equation
- *_linear: Refers to the linear interpolation procedure
- *_limited: Refers tot eh interpolation procedure that includes transport limitations
- *_err: Notates error data for the considered value

Data management¶

In [2]:

```
# Experimental Parameters used
offset = 0.197 # Offset of the Ag/AgCl reference electrode at pH=0
Rs = 6 # Resistance of the V-cell (determined by the high frequency resistance of a
n impedance spectra)
area = np.pi*0.2**2 # area of the electrode that is exposed to the electrolyte in t
he V-cell

# The basepath is the folder, in which the experimental data is saved
basepath = r"C:\science\Publikaitonen\AuPt\Daten\flow_cell_data_merged\000_merged"
basepath_CO = r"C:\science\Publikaitonen\AuPt\Daten\2022-05-16-AuPt_flow_cell_CO"

# Individual folders include the measured data of the alloys
list_folders = ['Au', 'Au999Pt1','Au997Pt3','Au99Pt1','Au35Pt1','Au34Pt2','Au9Pt1',
'Pt']
list_specimens = ["Au","Pt0_02","Pt0_3","Pt0_88","Pt3","Pt6","Pt11","Pt"]

dict_folders = dict(zip(list_folders,list_specimens))
dict_folders
```

Out[2]:

```
{'Au': 'Au',

 'Au999Pt1': 'Pt0_02',

 'Au997Pt3': 'Pt0_3',

 'Au99Pt1': 'Pt0_88',
```

```
 'Au35Pt1': 'Pt3',
 'Au34Pt2': 'Pt6',
 'Au9Pt1': 'Pt11',
 'Pt': 'Pt'}
```

In [3]:

```
# define the specimens' labels
label_dict = {'Au': "Au",
 'Au999Pt1': "0.021 %$\mathregular{_{Pt}}$",
 'Au997Pt3': "0.33 %$\mathregular{_{Pt}}$",
 'Au99Pt1': "0.88 %$\mathregular{_{Pt}}$",
 'Au35Pt1': "3.1 %$\mathregular{_{Pt}}$",
 'Au34Pt2': "6.7 %$\mathregular{_{Pt}}$",
 'Au9Pt1': "11.2 %$\mathregular{_{Pt}}$",
 'Pt': "Pt",}

# define the specimens' colors for the plots
color_dict = {'Au': (0.0, 0.0, 0.0, 1.0),
 'Au999Pt1': (160/256, 70/256, 160/256, 1.0),
 'Au997Pt3': (0.0, 0.07142857142857142, 1.0, 1.0),
 'Au99Pt1': (0.0, 0.6428571428571429, 1.0, 1.0),
 'Au35Pt1': (114/256, 220/256, 30/256, 1.0),
 'Au34Pt2': (1.0, 0.7248677248677255, 0.0, 1.0),
 'Au9Pt1': (1.0, 0.1957671957671958, 0.0, 1.0),
 'Pt': (0.6, 0.2, 0.2, 1.0)}

# List with the Pt content of all specimen
#list_composition = np.asarray([0,1e-3,3e-3,1e-2,1/36,2/36,1/10,1])
list_composition = np.asarray([0,0.021,0.33,0.88,3.11,6.67,11.18,100])/100
list_composition_error = np.asarray([0,0.0006,0.013,0.074,0.078,0.057,0])/100
dict_composition = dict(zip(list_folders,list_composition))

# colors and linestyles for the evaluation
color_Au = color_dict["Au"]
color_alloys = "r"
color_Pt = color_dict["Pt"]
color_interpol = "dodgerblue"
color_Tafel_ip = "green"
color_empirical_ip1 = "blue"
#color_empirical_ip2 = "indigo"
linesttyle_Tafel_ip = "dashed"
linesttyle_empirical_ip1 = "solid"
#linesttyle_empirical_ip2 = "dotted"

# names
name_IP_Tafel = "IPP-Pt-1-slope"
name_IP_two_slopes = "IPP-Pt-2-slopes"
name_IP_data_points = "IP-measured"
```

Functions for the experimental data treatment and analysis¶

In [4]:

```
def import_HER_file_with_correction(path):
    """
    - Reads the measured data file
    - Ohmic drop correction
    - Converts potentials to overpotentials
    - Converts currents to current densities
    """
    df = pd.read_csv(path)
    df["eta"] = (df.E + offset - df.I*Rs)
```

```
    df["j"] = -df.I/area
    df = df[df["j"]<0.2]
    return df

def import_HER_file(path):
    """
    - import_HER_file_with_correction was previously applied
    """
    df = pd.read_csv(path)
    df = df[df["j"]<0.2]
    return df

def make_mean_slices(df,eta_start,eta_stop,slices):
    """
    Slices the dataframe regarding the overpotential eta
    In the different slices of eta, the mean current density and
    its error are calculated
    """
    df_new = pd.DataFrame(columns = list(df) + ["j_err"])
    eta_array = np.linspace(eta_start,eta_stop,int(abs(eta_start-eta_stop)/slices))
    for i in range(0,len(eta_array)-1):
        row_mean = df[(df.eta >= eta_array[i]) & (df.eta < eta_array[i+1])].mean().
to_list()
        j_err = [df[(df.eta >= eta_array[i]) & (df.eta < eta_array[i+1])]["j"].std(
)]
        df_new.loc[i] = row_mean + j_err
    return df_new.dropna()
```

Functions for the linear interpolation procedure¶

In [5]:

```
# an array with a variation of overpotetnisl
eta_test_array = np.linspace(-1e-7,-0.54,10000) # an array with a variation of over
potetnisl etas
j_TE_Au_test_array = 0 # current density array, with the Tafel equation data on the
eta_test_array
j_TE_Pt_test_array = 0 # current density array, with the Tafel equation data on the
eta_test_array

def Tafel_eq(j_0,exponent):
    """
    Tafel equation, to calculate the current density for a given overpotential eta
    """
    return j_0*np.exp(-exponent)

def point_to_point_linear_combination_eta(eta_Au,eta_Pt):
    """
    finds numerical solution for the current density for an array of overpotentials
    """
    # make an array with the interpolated data
    x_inter = np.linspace(0,4,100)
    y_inter = np.linspace(0,1,100)
    j_interpol_array = 1*10**(-x_inter)
    eta_interpol_array = ((y_inter)*(-eta_Au) + (1-y_inter)*(-eta_Pt))
    return j_interpol_array, eta_interpol_array

def df_window(df,column,low, high):
    """
    filters the dataframe within the low and high boundaries
    of a specific column
    """
    return df[(df[column]<high) & (df[column]>low)]

def get_exp_etas(df,j,j_pm):
    """
```

```
    returns the mean measured overpotential and its standard variation
    for a given current density range between j-j_pm and j+j_pm
    """
    df_test = df_window(df,"j",j-j_pm, j+j_pm)
    return df_test.eta.mean(), df_test.eta.std()
```

Functions for local current densities and their interpolation procedure¶

In [6]:

```
fit_Au = [0,0]
fit_Pt = [0,0]
def current_Pt_Tafel(eta):
    global fit_Pt
    return Tafel_eq(fit_Pt[0],fit_Pt[1]*eta)

def current_Au_Tafel(eta):
    global fit_Au
    return Tafel_eq(fit_Au[0],fit_Au[1]*eta)

def current_Pt_two_slopes(eta):
    exp_divide = 0.45
    prefactor = 3e-5
    return prefactor*(Tafel_eq(fit_Pt[0],fit_Pt[1]*eta)/(Tafel_eq(fit_Au[0],fit_Au[
1]*eta/exp_divide)+prefactor))

def calc_linear_combination_eta(Pt_kinetic_function, eta, mole_frac_Pt):
    mole_frac_Au = 1-mole_frac_Pt
    j = mole_frac_Au*current_Au_Tafel(eta) + mole_frac_Pt*Pt_kinetic_function(eta)
    return j

def solve_linear_combination_j(Pt_kinetic_function,j, mole_frac_Pt):
    j_test_array = calc_linear_combination_eta(Pt_kinetic_function,eta_test_array,
mole_frac_Pt)
    argmin = np.argmin(abs(j_test_array - j))
    return eta_test_array[argmin]

def closest_index(index_list, search_value):
    """
    finds the index of the df, that is closest to the the search_value
    """
    np_index = np.asarray(index_list)
    return np_index[np.argmin(abs(np_index-search_value))]
```

In [7]:

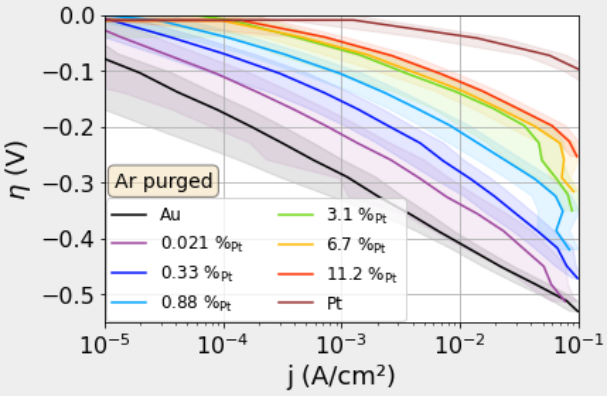
```
def Tafel_slope(j_array, eta_array):
    return -(eta_array.diff()/np.log10(j_array).diff())*1e3
```

Show experimental HER Data¶

In [8]:

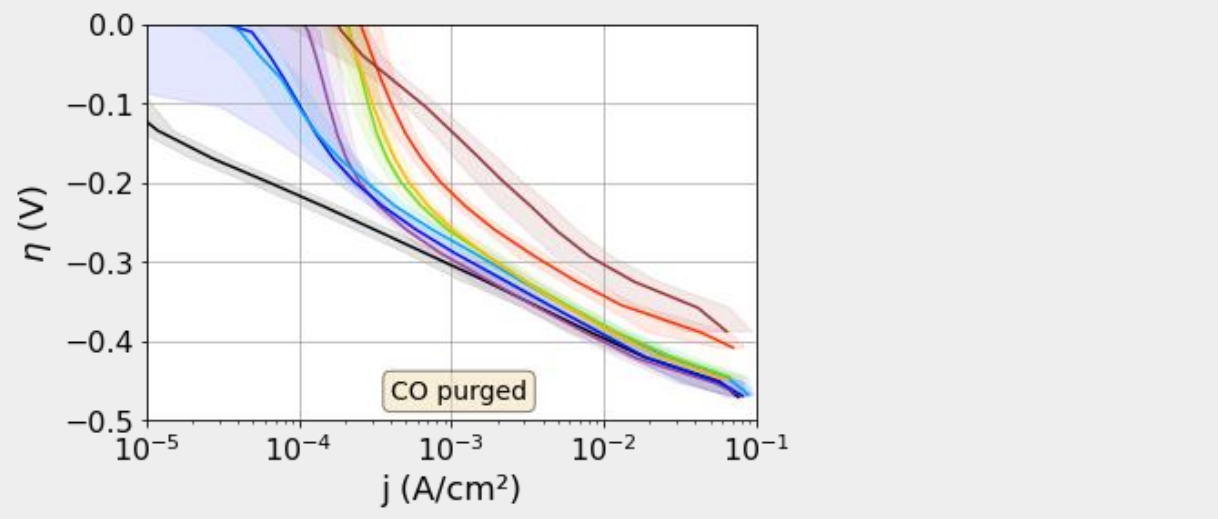
```
#plot mean HER data with error bars
for folder in list_folders:
    df = import_HER_file(basepath+"/"+dict_folders[folder]+"/HER.csv")
    vars()["df_HER_"+folder] = df
    # plot forward
    #df = df[df.j <= 0.2]
    df = df[df.eta <= 0.02]
    df = make_mean_slices(df,-1,0.1,0.03)
    plt.plot(df["j"],df["eta"], color = color_dict[folder], linestyle = "solid",lab
el = label_dict[folder])
```

```
    plt.fill(np.append(df["j"]-df["j_err"],(df["j"]+df["j_err"])[::-1]),np.append(d
f["eta"],df["eta"][::-1]),color = color_dict[folder],alpha=0.1)

plt.grid()
plt.xlabel("j (A/cm²)")
plt.xscale("log")
plt.ylabel("$\eta$ (V)")
plt.ylim(-0.55,0)
plt.xlim(1e-5,0.1)
plt.legend(prop=props_legend,ncol=2,loc=(0.002,0.005))
plt.text(1.2e-5, -0.315, "Ar purged", fontsize=14, verticalalignment='bottom', bbox
=txt_box_props)
plt.savefig("pics/UI_with_error_shading.png", dpi = 200, bbox_inches = "tight")
```



In [9]:

```
#plot mean HER-CO data with error bars
for folder in list_folders:
    df = import_HER_file_with_correction(basepath_CO+"/"+folder+"/HER.csv")
    vars()["df_HER_CO"+folder] = df
    df = df[df.j <= 0.12]
    df = df[df.eta <= 0.02]
    df = make_mean_slices(df,-1,0.1,0.03)
    plt.plot(df["j"],df["eta"], color = color_dict[folder], linestyle = "solid",lab
el = label_dict[folder])
    plt.fill(np.append(df["j"]-df["j_err"],(df["j"]+df["j_err"])[::-1]),np.append(d
f["eta"],df["eta"][::-1]),color = color_dict[folder],alpha=0.1)
plt.grid()
plt.xlabel("j (A/cm²)")
plt.xscale("log")
plt.ylabel("$\eta$ (V)")
plt.ylim(-0.5,0)
plt.xlim(1e-5,0.1)
plt.text(4e-4, -0.48, "CO purged", fontsize=14, verticalalignment='bottom', bbox=tx
t_box_props)
plt.savefig("pics/UI_CO.png", dpi = 200, bbox_inches = "tight")
```

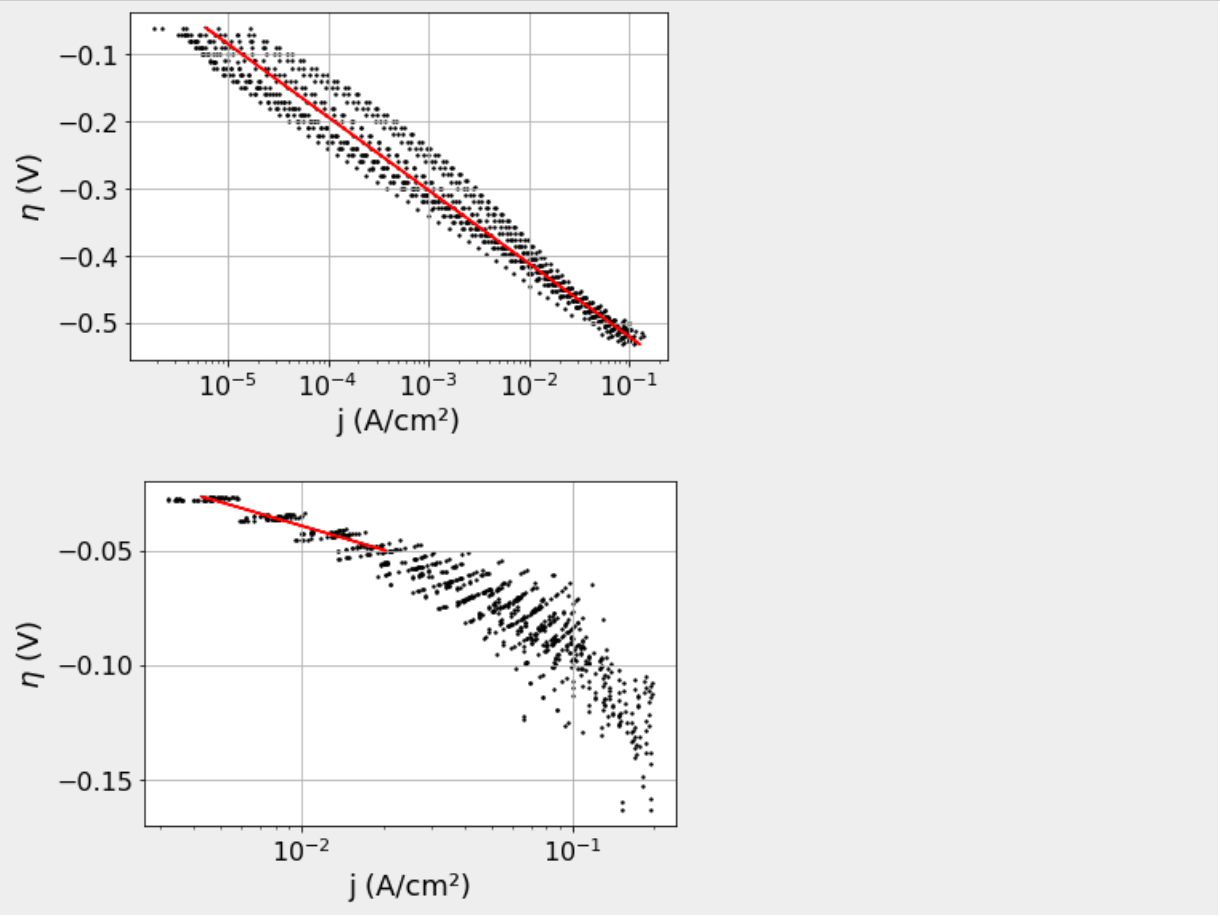


Fit Tafel equation to data of the pure metals¶

In [10]:

```
# fit platinum and gold bx exponential dependence
def func(x,j0, tafel_slope):
    return j0*np.exp(-tafel_slope*x)

def linear(x,ln_j0,tafel_slope):
    return ln_j0 - tafel_slope*x

# fit gold
df_test = df_HER_Au[df_HER_Au["eta"]<-0.06]
eta_data = df_test.eta
j_data = df_test.j
popt, pcov = curve_fit(linear, eta_data, np.log(j_data))
popt = [np.exp(popt[0]),popt[1]]
fit = func(eta_data, *popt)
fit_Au = popt
plt.grid()
plt.scatter(j_data,eta_data,s=2,color="k")
plt.plot(fit,eta_data,"r")
plt.xscale("log")
plt.ylabel("$\eta$ (V)")
plt.xlabel("j (A/cm²)")
plt.savefig("pics/fit_Au.png", dpi = 200, bbox_inches = "tight")
plt.show()

# fit platinum
df_test = df_HER_Pt[df_HER_Pt["eta"]<-0.02]
df_test = df_test[df_test["eta"]>-0.05]
eta_data = df_test.eta
j_data = df_test.j
popt, pcov = curve_fit(linear, eta_data, np.log(j_data))
popt = [np.exp(popt[0]),popt[1]]
fit = func(eta_data, *popt)
fit_Pt = popt
df_test = df_HER_Pt[df_HER_Pt["eta"]<-0.02]
plt.grid()
plt.scatter(df_test.j,df_test.eta,s=2,color="k")
plt.plot(fit,eta_data,"r")
plt.xscale("log")
plt.ylabel("$\eta$ (V)")
plt.xlabel("j (A/cm²)")
plt.xlim()
plt.savefig("pics/fit_Pt.png", dpi = 200, bbox_inches = "tight")
```

```
plt.show()
```



In [11]:

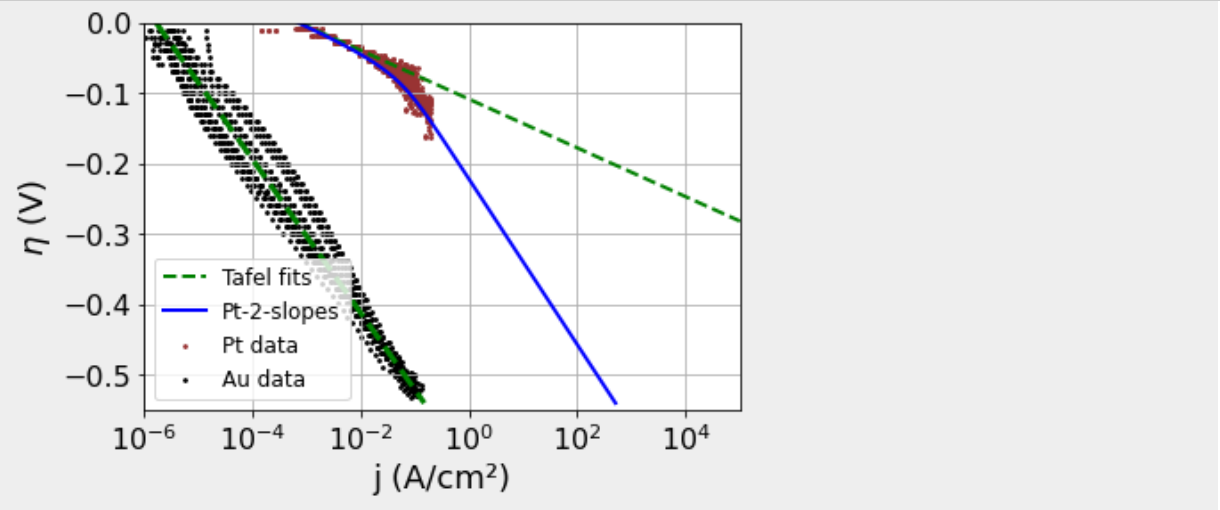
```
print(fit_Au)
print(fit_Pt)
```

```
[1.6923191926643396e-06, 21.106052318541753]

[0.0007348237759699302, 66.47895170863465]
```

In [12]:

```
linewidth = 2

j_Au = []
j_Pt_linear = []
j_Pt_two_slopes = []

for eta in eta_test_array:
    j_Au.append(current_Au_Tafel(eta))
    j_Pt_linear.append(current_Pt_Tafel(eta))
    j_Pt_two_slopes.append(current_Pt_two_slopes(eta))

plt.grid()

df_Au = df_HER_Au[df_HER_Au["eta"]<-0.005]
df_Pt = df_HER_Pt[df_HER_Pt["eta"]<-0.005]
plt.scatter(df_Pt.j,df_Pt.eta, color = color_dict["Pt"], label = "Pt data",s=3)
plt.scatter(df_Au.j,df_Au.eta, color = color_dict["Au"], label = "Au data",s=3)

df_Tafel = pd.DataFrame()
df_Tafel["eta"] = eta_test_array
```

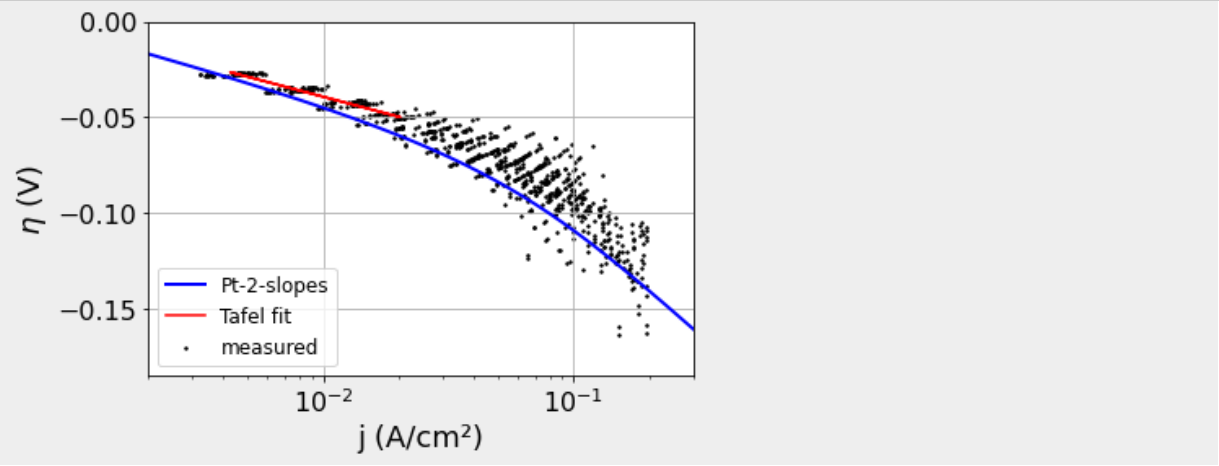
```
df_Tafel["Au_linear"] = j_Au
df_Tafel["Pt_linear"] = j_Pt_linear
df_Tafel["Pt_two_slopes"] = j_Pt_two_slopes

plt.plot(df_Tafel["Au_linear"],df_Tafel["eta"], linewidth=3,  linestyle = linesttyl
e_Tafel_ip, color = color_Tafel_ip)
plt.plot(df_Tafel["Pt_linear"],df_Tafel["eta"], label = "Tafel fits", linewidth=lin
ewidth, linestyle = linesttyle_Tafel_ip, color = color_Tafel_ip)
plt.plot(df_Tafel["Pt_two_slopes"],df_Tafel["eta"], linewidth=linewidth, linestyle
= linesttyle_empirical_ip1, label = "Pt-2-slopes", color = color_empirical_ip1)

plt.xscale("log")
plt.xlabel("j (A/cm²)")
plt.ylabel("$\eta$ (V)")
plt.ylim(-0.55,0)
plt.xlim(1e-6,1e5)
plt.legend(prop=props_legend, loc = "lower left")
plt.savefig("pics/Tafel Pt mass transport.png", dpi = 200, bbox_inches = "tight")

plt.show()
```



In [13]:

```
# fit platinum details with Pt-2-slopes
df_test = df_HER_Pt[df_HER_Pt["eta"]<-0.02]
df_test = df_test[df_test["eta"]>-0.05]
eta_data = df_test.eta
j_data = df_test.j
popt, pcov = curve_fit(linear, eta_data, np.log(j_data))
popt = [np.exp(popt[0]),popt[1]]
fit = func(eta_data, *popt)
fit_Pt = popt
df_test = df_HER_Pt[df_HER_Pt["eta"]<-0.02]
plt.grid()
plt.plot(df_Tafel["Pt_two_slopes"],df_Tafel["eta"], linewidth=linewidth, linestyle
= linesttyle_empirical_ip1, label = "Pt-2-slopes", color = color_empirical_ip1)

plt.scatter(df_test.j,df_test.eta,s=2,color="k",label = "measured")
plt.plot(fit,eta_data,"r", label = "Tafel fit")
plt.xscale("log")
plt.ylabel("$\eta$ (V)")
plt.xlabel("j (A/cm²)")
plt.xlim(2e-3,0.3)
plt.ylim(-0.185,0)
plt.yticks(-np.arange(0,0.2,0.05))
plt.legend(prop = props_legend)
plt.savefig("pics/fit_Pt.png", dpi = 200, bbox_inches = "tight")
```

```
plt.show()
```

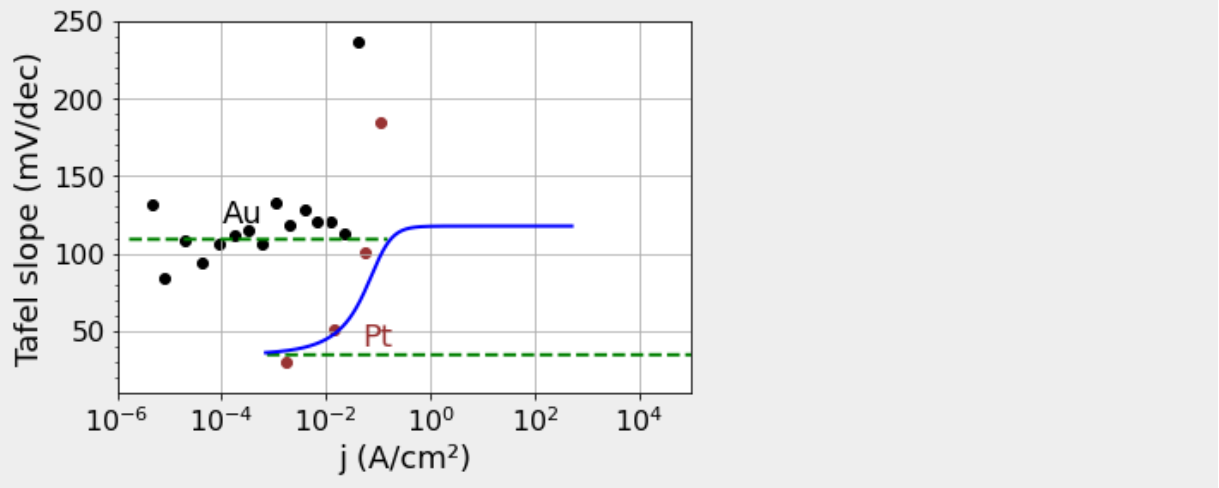


In [14]:

```
# arrays used for further data evaluation
j_TE_Au_test_array = Tafel_eq(fit_Au[0],fit_Au[1]*eta_test_array)
j_TE_Pt_test_array = Tafel_eq(fit_Pt[0],fit_Pt[1]*eta_test_array)
mole_frac_Pt_array = np.linspace(0,1,10000)[1:] # An array with linear spaced value
s to resemble a composition variation
x_inter = np.linspace(-4,6.1,1000)
j_array = 1*10**(-x_inter)
```

In [15]:

```
plt.plot(df_Tafel["Au_linear"],Tafel_slope(df_Tafel["Au_linear"],df_Tafel["eta"]),
linewidth=2,  linestyle = linesttyle_Tafel_ip, color = color_Tafel_ip)
plt.plot(df_Tafel["Pt_linear"],Tafel_slope(df_Tafel["Pt_linear"],df_Tafel["eta"]),
label = "Tafel fits", linewidth=linewidth, linestyle = linesttyle_Tafel_ip, color =
color_Tafel_ip)
plt.plot(df_Tafel["Pt_two_slopes"],Tafel_slope(df_Tafel["Pt_two_slopes"],df_Tafel["
eta"]), linewidth=linewidth, linestyle = linesttyle_empirical_ip1, label = name_IP_
two_slopes, color = color_empirical_ip1)

df = df_HER_Au
df = df[df.j <= 0.06]
df = df[df.eta <= -0.02]
df = make_mean_slices(df,-1,0.1,0.03)
df_Au_m = df

plt.scatter(df_Au_m["j"],Tafel_slope(df_Au_m["j"],df_Au_m["eta"]), color = color_di
ct["Au"])
df_Pt_m = make_mean_slices(df_Pt,-1,0.1,0.03)
df_Pt_m["TafelSlope"] = Tafel_slope(df_Pt_m["j"],df_Pt_m["eta"])
plt.scatter(df_Pt_m["j"],df_Pt_m["TafelSlope"], color = color_dict["Pt"])

plt.ylim(10,250)
plt.xlim(1e-6,1e5)
plt.minorticks_on()
plt.xscale("log")
plt.xlabel("j (A/cm²)")
plt.ylabel("Tafel slope (mV/dec)")
plt.text(1e-4,120,"Au")
plt.text(5e-2,40,"Pt",color = color_dict["Pt"])
plt.grid()
plt.savefig("pics/Tafel slope fit and IP.png", dpi = 200, bbox_inches = "tight")
```

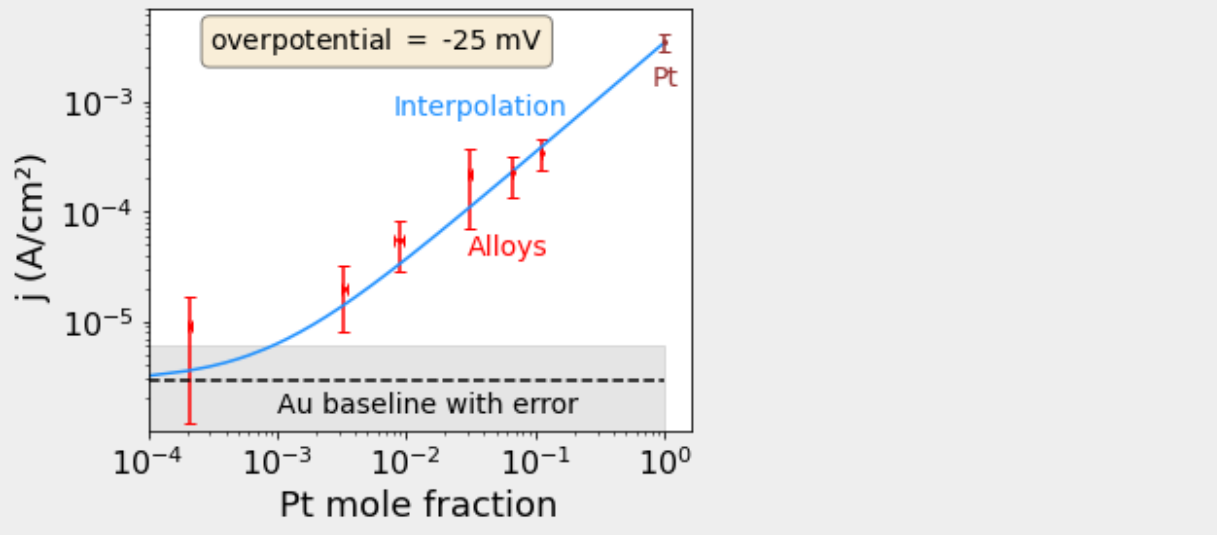


Interpolation for given overpotentials¶

In [42]:

```
matplotlib.rcParams["figure.figsize"] = (5,4)

#graphical abstract
etas = [-0.025]
iteration = 0
for eta in etas:
    # errors from statistics: True or False
    eta_low = eta-max(0.01, eta*0.3)
    eta_high = eta+max(0.01, eta*0.3)

    # get data
    list_j = []
    list_j_err = []
    for alloy in list_folders:
        df_test = make_mean_slices(vars()["df_HER_"+alloy],-1,0,0.01)
        df_test = df_window(df_test,"eta",eta_low,eta_high)
        list_j_err.append(df_test.j_err.mean())
        list_j.append(df_test.j.mean())

    # determine arrays and errors
    #Au_baseline = list_j[0]
    Au_baseline = current_Au_Tafel(eta)
    Pt_baseline = list_j[len(list_j)-1]
    Au_bl_array = np.asarray([Au_baseline,Au_baseline])
    j_interpol = np.linspace(Au_baseline,Pt_baseline,10000)[1:] # interpolation arr
ay between Au and Pt baseline
    j_interpol_err = np.linspace(list_j_err[0],list_j_err[-1],10000)[1:]

    # errors
    error_array = list_j_err #errors of the measurements
    length = len(list_j)-1
    mean_Au_Pt_error = (list_j_err[0]/list_j[0] + list_j_err[length]/list_j[length]
)/2

    #plt.grid()
    # Au baseline
    plt.plot([1e-4,1],Au_bl_array, color = color_Au,linestyle="dashed",label = "Au
baseline")
    plt.fill_between([1e-4,1], Au_bl_array-error_array[0], Au_bl_array+error_array[
0],color = color_Au,alpha=0.1)
    # Pt
```

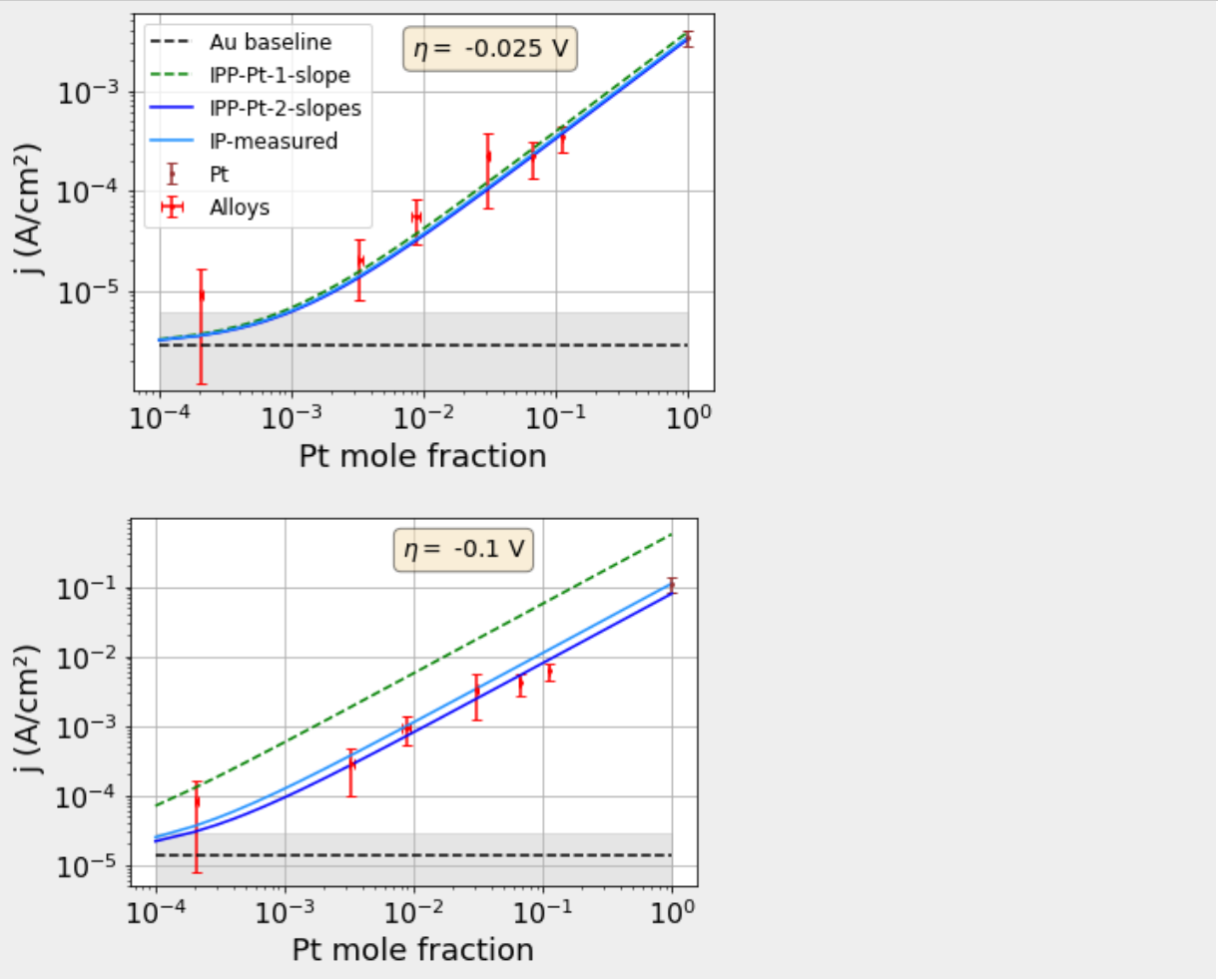
```
    plt.errorbar(list_composition[length:length+1], list_j[length:length+1], yerr=n
p.asarray(list_j_err[length:length+1]), fmt='o', markersize=2, capsize=3, color = c
olor_Pt,label="Pt")
    # Alloys
    length = len(list_composition)-1
    plt.errorbar(list_composition[1:length], list_j[1:length], yerr=np.asarray(list
_j_err[1:length]), xerr=np.asarray(list_composition_error[1:length]), fmt='o', mark
ersize=2, capsize=3, color = color_alloys,label="Alloys")

    # interpolated activity
    j_linear = []
    j_two_slopes = []
    for mole_frac_Pt in mole_frac_Pt_array:
        # solve linear interpolation
        j_linear.append(calc_linear_combination_eta(current_Pt_Tafel, eta, mole_fra
c_Pt))
        j_two_slopes.append(calc_linear_combination_eta(current_Pt_two_slopes, eta,
mole_frac_Pt))

    #plt.plot(mole_frac_Pt_array,j_linear, color = color_Tafel_ip,linestyle = lines
ttyle_Tafel_ip, label = name_IP_Tafel)
    #plt.plot(mole_frac_Pt_array,j_two_slopes, color = color_empirical_ip1,linestyl
e = linesttyle_empirical_ip1, label = name_IP_two_slopes)

    # interpolated data
    plt.plot(mole_frac_Pt_array,j_interpol, color = color_interpol,linestyle = "sol
id", label = name_IP_data_points)
    plt.xscale("log")
    plt.yscale("log")
    plt.xlabel("Pt mole fraction")
    plt.ylabel("j (A/cm²) ")
    plt.ylim(1.01e-6,7e-3)
    #plt.xlim(7e-4,)
    plt.xlim(1e-4,)
    #if iteration == 0 : plt.legend(prop=props_legend)
    plt.text(0.0003, 5e-3, "overpotential $=$ -25 mV", fontsize=14, verticalalignme
nt='top', bbox=txt_box_props)
    plt.text(0.001, 1.5e-6, "Au baseline with error", color = "k", fontsize=14)
    plt.text(8e-3, 7.5e-4, "Interpolation", color = color_interpol, fontsize=14)
    plt.text(0.03, 4e-5, "Alloys", color = "r", fontsize=14)
    plt.text(0.8, 1.4e-3, "Pt", color = (0.6, 0.2, 0.2, 1.0), fontsize=14)
    plt.savefig("pics/activity_at_eta_TOC"+str(int(eta*1000))+"mV.png", dpi = 200,
bbox_inches = "tight")
    plt.show()
matplotlib.rcParams["figure.figsize"] = (6,4)
```



In [19]:

```
etas = [-0.025, -0.1]
ylim = [1.001e-6,5e-6]
iteration = 0
for eta in etas:
    # errors from statistics: True or False
    eta_low = eta-max(0.01, eta*0.3)
    eta_high = eta+max(0.01, eta*0.3)

    # get data
    list_j = []
    list_j_err = []
    for alloy in list_folders:
        df_test = make_mean_slices(vars()["df_HER_"+alloy],-1,0,0.01)
        df_test = df_window(df_test,"eta",eta_low,eta_high)
        list_j_err.append(df_test.j_err.mean())
        list_j.append(df_test.j.mean())

    # determine arrays and errors
    #Au_baseline = list_j[0]
    Au_baseline = current_Au_Tafel(eta)
    Pt_baseline = list_j[len(list_j)-1]
    Au_bl_array = np.asarray([Au_baseline,Au_baseline])
    j_interpol = np.linspace(Au_baseline,Pt_baseline,10000)[1:] # interpolation arr
ay between Au and Pt baseline
    j_interpol_err = np.linspace(list_j_err[0],list_j_err[-1],10000)[1:]

    # errors
    error_array = list_j_err #errors of the measurements
    length = len(list_j)-1
    mean_Au_Pt_error = (list_j_err[0]/list_j[0] + list_j_err[length]/list_j[length]
)/2

    plt.grid()
    # Au baseline
    plt.plot([1e-4,1],Au_bl_array, color = color_Au,linestyle="dashed",label = "Au
baseline")
    plt.fill_between([1e-4,1], Au_bl_array-error_array[0], Au_bl_array+error_array[
0],color = color_Au,alpha=0.1)
    # Pt
    plt.errorbar(list_composition[length:length+1], list_j[length:length+1], yerr=n
p.asarray(list_j_err[length:length+1]), fmt='o', markersize=2, capsize=3, color = c
olor_Pt,label="Pt")
    # Alloys
    length = len(list_composition)-1
    plt.errorbar(list_composition[1:length], list_j[1:length], yerr=np.asarray(list
_j_err[1:length]), xerr=np.asarray(list_composition_error[1:length]), fmt='o', mark
ersize=2, capsize=3, color = color_alloys,label="Alloys")

    # interpolated activity
    j_linear = []
    j_two_slopes = []
    for mole_frac_Pt in mole_frac_Pt_array:
        # solve linear interpolation
        j_linear.append(calc_linear_combination_eta(current_Pt_Tafel, eta, mole_fra
c_Pt))
        j_two_slopes.append(calc_linear_combination_eta(current_Pt_two_slopes, eta,
mole_frac_Pt))

    plt.plot(mole_frac_Pt_array,j_linear, color = color_Tafel_ip,linestyle = linest
tyle_Tafel_ip, label = name_IP_Tafel)
    plt.plot(mole_frac_Pt_array,j_two_slopes, color = color_empirical_ip1,linestyle
= linesttyle_empirical_ip1, label = name_IP_two_slopes)

    # interpolated data
    plt.plot(mole_frac_Pt_array,j_interpol, color = color_interpol,linestyle = "sol
id", label = name_IP_data_points)
    plt.xscale("log")
```

```
    plt.yscale("log")
    plt.xlabel("Pt mole fraction")
    plt.ylabel("j (A/cm²) ")
    plt.ylim(ylim[iteration],)
    if iteration == 0 : plt.legend(prop=props_legend)
    plt.text(0.008, np.asarray(j_linear).max()*0.95, "$\eta =$ "+ str(eta) + " V",
fontsize=14, verticalalignment='top', bbox=txt_box_props)
    plt.savefig("pics/activity_at_eta"+str(int(eta*1000))+"mV.png", dpi = 200, bbox
_inches = "tight")
    plt.show()
    iteration += 1
```

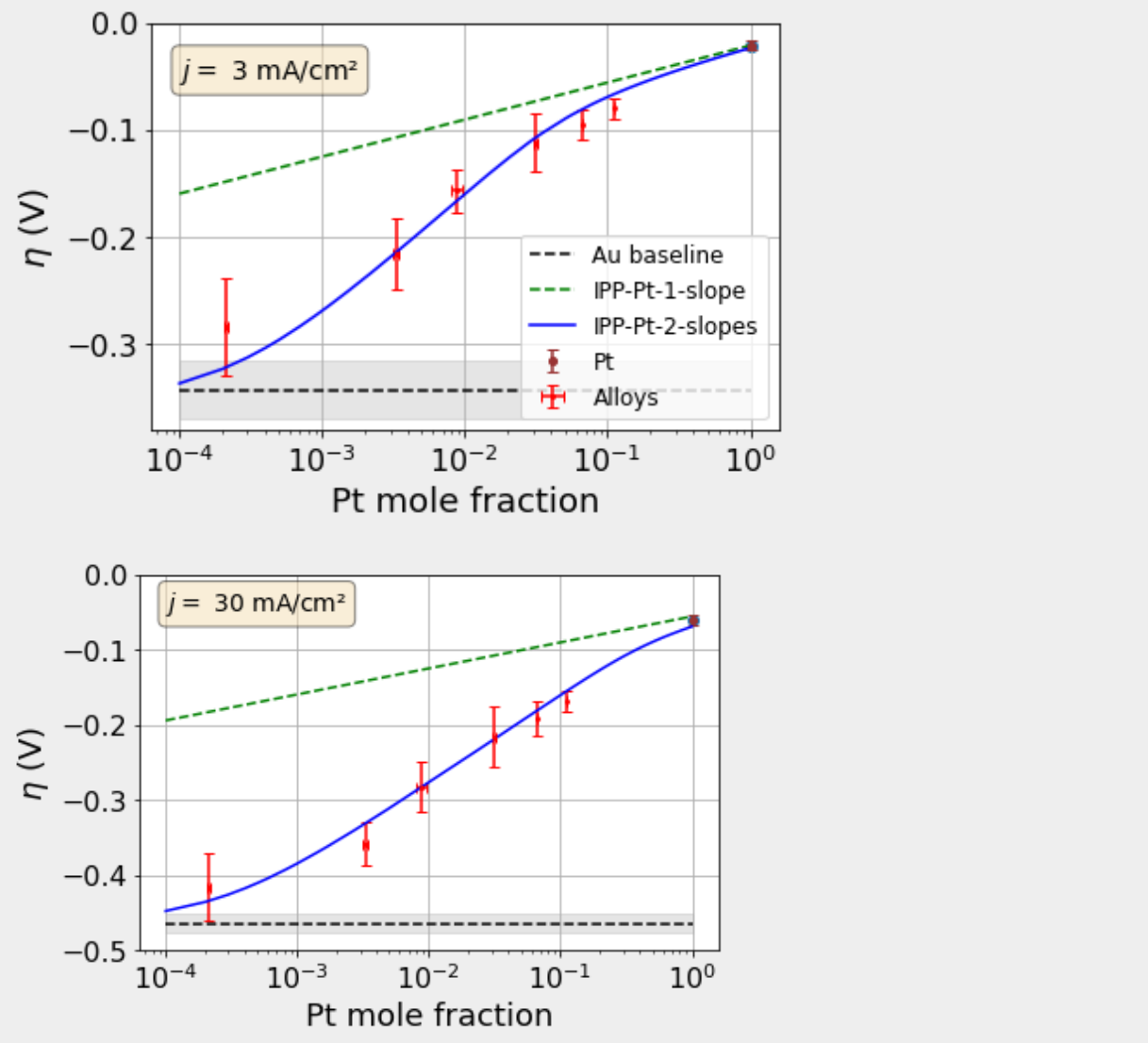


Interpolation for given current densities¶

In [20]:

```
# current test arrays for the numerical solution
j_TE_Au_test_array = Tafel_eq(fit_Au[0],fit_Au[1]*eta_test_array)
j_TE_Pt_test_array = Tafel_eq(fit_Pt[0],fit_Pt[1]*eta_test_array)

### start current interpolation ###
iteration = 0

ylim= [-0.38,-0.5]
for j in [0.003,0.03]:
    j_pm = j/5

    # parameteriztation of linear interpolation
    liste_eta_exp = []
```

```
    liste_eta_exp_err = []
    for i in list_folders:
        a, b = get_exp_etas(vars()["df_HER_"+i],j,j_pm)
        liste_eta_exp.append(a)
        liste_eta_exp_err.append(b)

    # determine arrays and errors
    Au_baseline = liste_eta_exp[0]
    Au_bl_array = np.asarray([Au_baseline,Au_baseline])
    length = len(list_composition)-1
    eta_linear = []
    eta_empirical1 = []
    eta_empirical2 = []

    for mole_frac_Pt in mole_frac_Pt_array:
        # solve linear interpolation
        eta_linear.append(solve_linear_combination_j(current_Pt_Tafel,j,mole_frac_P
t))
        eta_empirical1.append(solve_linear_combination_j(current_Pt_two_slopes,j,mo
le_frac_Pt))

    eta_empirical_err_l = np.asarray(eta_empirical1)-0.02
    eta_empirical_err_h = np.asarray(eta_empirical1)+0.02

    plt.grid()
    # Au baseline
    plt.plot([1e-4,1],Au_bl_array, color = color_Au,linestyle = "dashed", label = "
Au baseline")
    plt.fill_between([1e-4,1], Au_bl_array-liste_eta_exp_err[0], Au_bl_array+liste_
eta_exp_err[0],color = color_Au,alpha=0.1)

    # Pt data
    plt.errorbar(list_composition[length:length+1], liste_eta_exp[length:length+1],
yerr=liste_eta_exp_err[length:length+1], fmt='o', markersize=4, capsize=3, color =
color_Pt,label="Pt")
    plt.scatter(list_composition[length:length+1], liste_eta_exp[length:length+1])

    # Alloys
    plt.errorbar(list_composition[1:length], liste_eta_exp[1:length], yerr=liste_et
a_exp_err[1:length], xerr= list_composition_error[1:length], fmt='o', markersize=2,
capsize=3, color = color_alloys,label="Alloys")
    plt.plot(mole_frac_Pt_array,eta_linear,color = color_Tafel_ip, linestyle = line
sttyle_Tafel_ip, label = name_IP_Tafel)
    plt.plot(mole_frac_Pt_array,eta_empirical1,color = color_empirical_ip1, linesty
le = linesttyle_empirical_ip1, label = name_IP_two_slopes)

    plt.xscale("log")
    plt.xlabel("Pt mole fraction")
    plt.ylabel("$\eta$ (V)")
    if iteration == 0: plt.legend(prop=props_legend, loc = "lower right")
    plt.text(1e-4, -0.06, "$j =$ " + str(int(1000*j)) +" mA/cm²", fontsize=14, vert
icalalignment='bottom', bbox=txt_box_props)
    plt.ylim(ylim[iteration],0.0)
    plt.savefig("pics/activity_Current"+str(int(j*1000))+"mA.png", dpi = 200, bbox_
inches = "tight")
    plt.show()
    iteration += 1
```



Interpolated potential-current curves¶

In [21]:

```
# Data frames with total currents of the alloys excluding and
# including the transprot limitations. index = j
df_UI_model_Pt_1_slope = pd.DataFrame(columns = list_folders)
df_UI_model_Pt_2_slopes = pd.DataFrame(columns = list_folders)
# Data frames with current contributions of the Au or Pt surface
# sites including the transprot limitations. index = j
df_UI_model_Au = pd.DataFrame(columns = list_folders)
df_UI_model_Pt = pd.DataFrame(columns = list_folders)
```

In [22]:

```
# model overpotential vs current density
j_test_array = 10**(-np.linspace(1,5,1000)) # logarithmically spaced current densit
ies

for j in j_test_array:
    eta_Pt_1_slope = []
    eta_Pt_2_slopes = []
    mole_fraction_array = []
```

```
    for alloy in list_folders:
        # solve linear interpolation
        mole_frac_Pt = dict_composition[alloy]
        mole_fraction_array.append(mole_frac_Pt)
        eta_Pt_1_slope.append(solve_linear_combination_j(current_Pt_Tafel,j,mole_fr
ac_Pt))
        eta_Pt_2_slopes.append(solve_linear_combination_j(current_Pt_two_slopes,j,m
ole_frac_Pt))

    #put data into the DataFrame
    df_UI_model_Pt_1_slope.loc[j] = eta_Pt_1_slope
    df_UI_model_Pt_2_slopes.loc[j] = eta_Pt_2_slopes

    eta_Pt_2_slopes_np = np.asarray(eta_Pt_2_slopes)
    mole_frac_Pt_np = np.asarray(mole_fraction_array)
    df_UI_model_Au.loc[j] = (1-mole_frac_Pt_np)*current_Au_Tafel(eta_Pt_2_slopes_np
)
    df_UI_model_Pt.loc[j] = mole_frac_Pt_np*current_Pt_two_slopes(eta_Pt_2_slopes_n
p)
```

In [23]:

```
eta_test_array2 = np.linspace(-0.54,0,10000) # an array with a variation of overpot
etnisl etas
df_UI_model_Pt_2_slopes_eta =  pd.DataFrame()
df_UI_model_Pt_2_slopes_eta_Pt =  pd.DataFrame()
# model overpotential vs current density
df_UI_model_Pt_2_slopes_eta_Pt["eta"] = eta_test_array2
for alloy in list_folders:
    # solve linear interpolation
    mole_frac_Pt = dict_composition[alloy]
    df_UI_model_Pt_2_slopes_eta[mole_frac_Pt] = calc_linear_combination_eta(current
_Pt_two_slopes, eta_test_array2, mole_frac_Pt)
    df_UI_model_Pt_2_slopes_eta_Pt[mole_frac_Pt] = mole_frac_Pt*current_Pt_two_slop
es(eta_test_array2)
```

In [24]:

```
df_UI_model_Pt_2_slopes_eta_Pt.head()
```

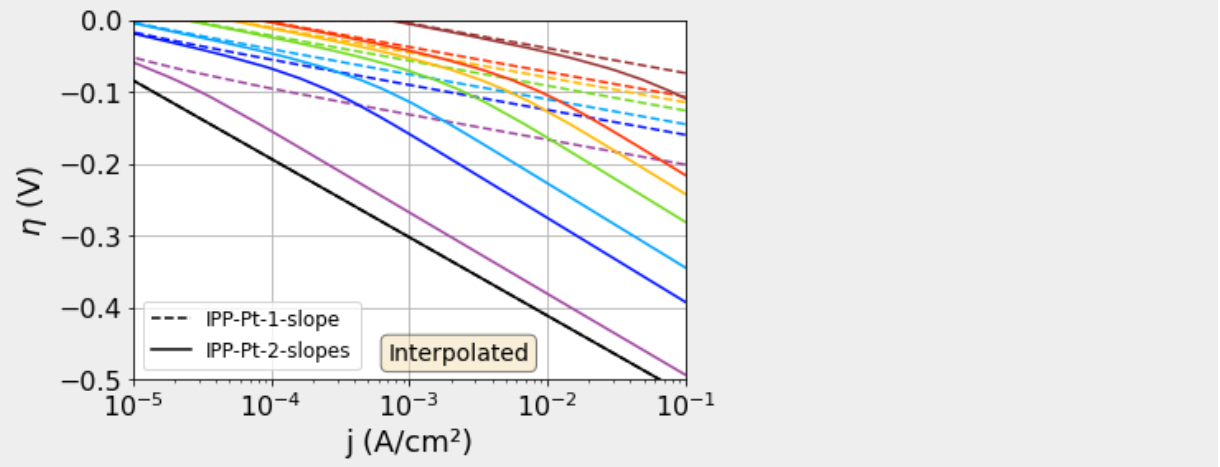
Out[24]:

	eta	0.0	0.00021	0.0033	0.0088	0.0311	0.0667	0.1118	1.0
0	-0.540000	0.0	0.106691	1.676580	4.470880	15.800496	33.887237	56.800496	508.054529
1	-0.539946	0.0	0.106579	1.674808	4.466156	15.783800	33.851429	56.740476	507.517676
2	-0.539892	0.0	0.106466	1.673039	4.461436	15.767121	33.815659	56.680519	506.981390

	eta	0.0	0.00021	0.0033	0.0088	0.0311	0.0667	0.1118	1.0
3	-0.539838	0.0	0.106354	1.671271	4.456722	15.750460	33.779926	56.620626	506.445671
4	-0.539784	0.0	0.106241	1.669505	4.452013	15.733817	33.744232	56.560796	505.910518

In [25]:

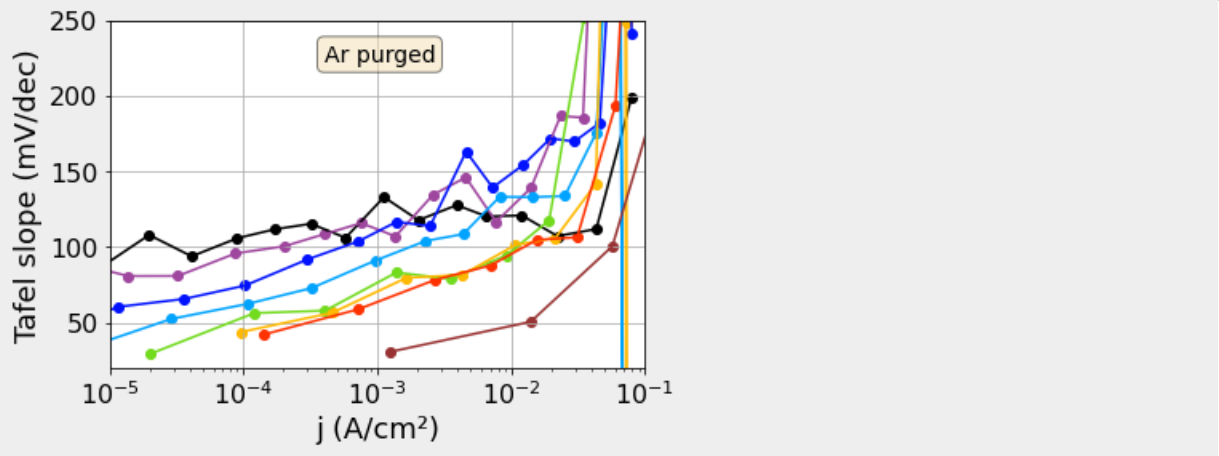
```
# plot modeled relations
for alloy in list_folders:
    col = alloy
    folder = alloy
    plt.plot(df_UI_model_Pt_2_slopes.index,df_UI_model_Pt_2_slopes[col], color = co
lor_dict[folder], linestyle = "solid")
    plt.plot(df_UI_model_Pt_1_slope.index,df_UI_model_Pt_1_slope[col], color = colo
r_dict[folder], linestyle = "dashed")

plt.grid()
plt.xlabel("j (A/cm²)")
plt.xscale("log")
plt.ylabel("$\eta$ (V)")
plt.ylim(-0.5,0)
plt.xlim(1e-5,0.1)
#plt.legend(prop=props_legend)
plt.text(7e-4, -0.48, "Interpolated", fontsize=14, verticalalignment='bottom', bbox
=txt_box_props)
plt.plot([-0.1,0],[-0.1,0],color = "k", linestyle = "dashed",label = name_IP_Tafel)
plt.plot([-0.1,0],[-0.1,0],color = "k", label = name_IP_two_slopes)
plt.legend(prop=props_legend)
plt.savefig("pics/UI_modeled.png", dpi = 200, bbox_inches = "tight")
```



In [26]:

```
#plot TAfel slopes
#plot mean HER data with error bars
for folder in list_folders:
    df = import_HER_file(basepath+"/"+dict_folders[folder]+"/HER.csv")
    vars()["df_HER_"+folder] = df
    # plot forward
```

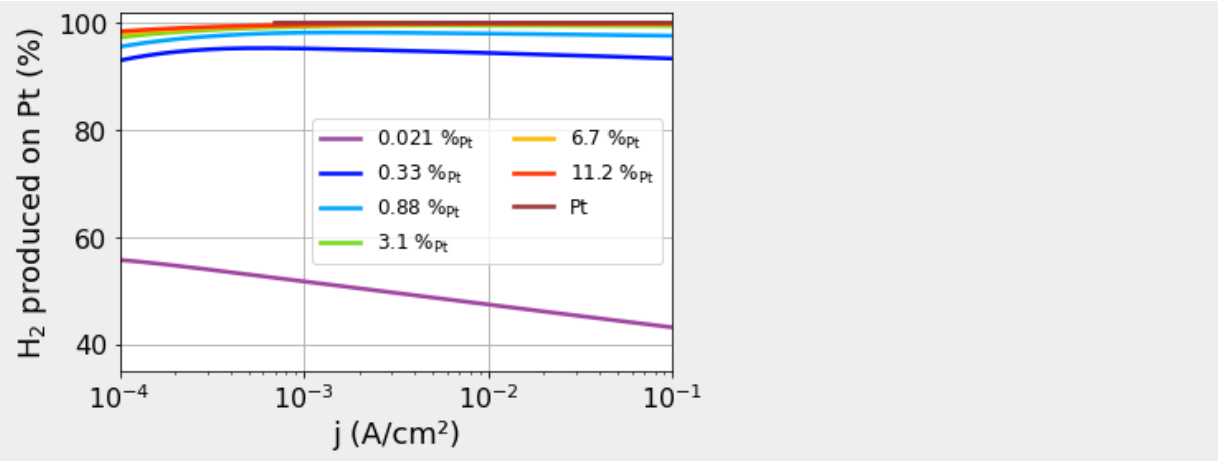
```
    #df = df[df.j <= 0.99]
    df = df[df.eta <= 0.02]
    df = make_mean_slices(df,-1,0.1,0.03)
    plt.scatter(df["j"],Tafel_slope(df["j"],df["eta"]), color = color_dict[folder])
    plt.plot(df["j"],Tafel_slope(df["j"],df["eta"]), color = color_dict[folder], li
nestyle = "solid")
plt.grid()
plt.xlabel("j (A/cm²)")
plt.xscale("log")
plt.ylabel("Tafel slope (mV/dec)")
plt.ylim(20,250)
plt.xlim(1e-5,0.1)
#plt.scatter(0,0,color = "k", label = "Measurements")
#plt.plot([-0.1,0],[-0.1,0],color = "k", label = "Interpolation")
#plt.legend(prop=props_legend)
plt.text(4e-4, 220, "Ar purged", fontsize=14, verticalalignment='bottom', bbox=txt_
box_props)
plt.savefig("pics/UI_tafel_slopes.png", dpi = 200, bbox_inches = "tight")
```



Plot current composition¶

In [44]:

```
matplotlib.rcParams["figure.figsize"] = (6,4)

#df_UI_model_limited_eta[mole_frac_Pt]

# plot current composition
for alloy in list_folders[1:]:
    mole_frac_Pt = dict_composition[alloy]
    col = alloy
    folder = alloy
    # find the index where the overpotential is 0 or larger:
    plt.plot(df_UI_model_Pt_2_slopes_eta[mole_frac_Pt],100*df_UI_model_Pt_2_slopes_
eta_Pt[mole_frac_Pt]/df_UI_model_Pt_2_slopes_eta[mole_frac_Pt], color = color_dict[
folder], linestyle = "solid",linewidth = 2.5,label = label_dict[folder])

plt.grid()
plt.xlabel("j (A/cm²)")
plt.xscale("log")
plt.ylabel(r"H$_2$ produced on Pt (%)")
plt.ylim(35,102)
plt.xlim(1e-4,0.1)
plt.legend(prop=props_legend,ncol=2)
plt.savefig("pics/Ratio_H2_production.png", dpi = 200, bbox_inches = "tight")
```



In [28]:

```
# plot Sabatier's principle with interacting activity descriptors
font1 = {'family' : 'normal',
        'weight' : 'normal',
        'size'   : 18}
matplotlib.rc('font', **font1)
matplotlib.rcParams["figure.figsize"] = (6,5)

# DFT calculated strongest adsorption strength at the Pt component
E_ads_Pt = {'Au': 0,
 'Au999Pt1': -0.35,
 'Au997Pt3': -0.35,
 'Au99Pt1': -0.35,
 'Au35Pt1': -0.35,
 'Au34Pt2': -0.35,
 'Au9Pt1': -0.35,
 'Pt': -0.43}

# DFT calculated strongest adsorption strength at the Au component
E_ads_Au = {'Au': 0.12,
 'Au999Pt1': 0.12,
 'Au997Pt3': 0.12,
 'Au99Pt1': 0.12,
 'Au35Pt1': 0.11,
 'Au34Pt2': 0.08,
 'Au9Pt1': 0.08,
 'Pt': 0}


j_test_array2 = [1e-4,1e-3,1e-2,1e-1]
linestyles = ["dotted","dashdot","dashed","solid"]

# plot vertical lines to show different regimes in the plot
plt.plot([-0.35,-0.35], [-0.7,0.205], color = "lightgrey",linestyle = "solid")
plt.plot([-0.3,-0.3], [-0.7,0.205], color = "lightgrey",linestyle = "solid")
plt.plot([0.07,0.07], [-0.7,0.205], color = "lightgrey",linestyle = "solid")

for i in range(0,len(j_test_array2)):
    list_eta = []
    list_ads = []
    j = j_test_array2[i]
    index = closest_index(list(df_UI_model_Pt_2_slopes.index),j)

    for alloy in list_folders:
        # find the index where the overpotential is 0 or larger:
        df_test_eta = df_UI_model_Pt_2_slopes[df_UI_model_Pt_2_slopes[alloy] <-0.00
3]
```

```
        # find the index where the overpotential is 0 or larger:
        try:
            eta = df_test_eta.at[index,alloy]
            ratio = df_UI_model_Pt.at[index,alloy]/index
        except:
            eta = 0
            ratio = 1

        adsorption_energy = ratio*E_ads_Pt[alloy]+(1-ratio)*E_ads_Au[alloy]
        list_eta.append(eta)
        list_ads.append(adsorption_energy)
        plt.scatter(adsorption_energy, eta, color = color_dict[alloy])
    plt.plot(list_ads,np.asarray(list_eta), color = "k", linestyle = linestyles[0],
linewidth = 1)#,label = str(j))

plt.yticks(np.arange(-0.6,0.1,0.2))
plt.tick_params(
    axis="y",
    which="minor",
    length=3,
color="k")

plt.ylabel("$\eta$ (V)")
#plt.yscale("log")
plt.xlabel(r"E$_\mathrm{ads,weighted}$ (eV)")
plt.ylim(-0.6,0.13)
plt.xlim(-0.45,0.17)
#plt.legend(prop=props_legend,ncol = 3,loc = "upper right")#, title="A/cm²")
plt.savefig("pics/Sabatier_4_dim.png", dpi = 200, bbox_inches = "tight")
#plt.grid()
```