Electronic Supplementary Material (ESI) for Reaction Chemistry & Engineering.

Supporting Information for "Accelerated Optimization of Pure Metal and Ligand Compositions for Light-driven Hydrogen Production"

Maya Bhat, Eric M. Lopato, Zoe C. Simon, Jill E. Millstone, Stefan Bernhard, and John R. Kitchin

November 5, 2021

Contents

1 Model Contour and Raw Data Plots

The model contour and raw data plots for the noble and non-noble metals were similar. Here, we have included the plots for all four of the metals in Figure 1.

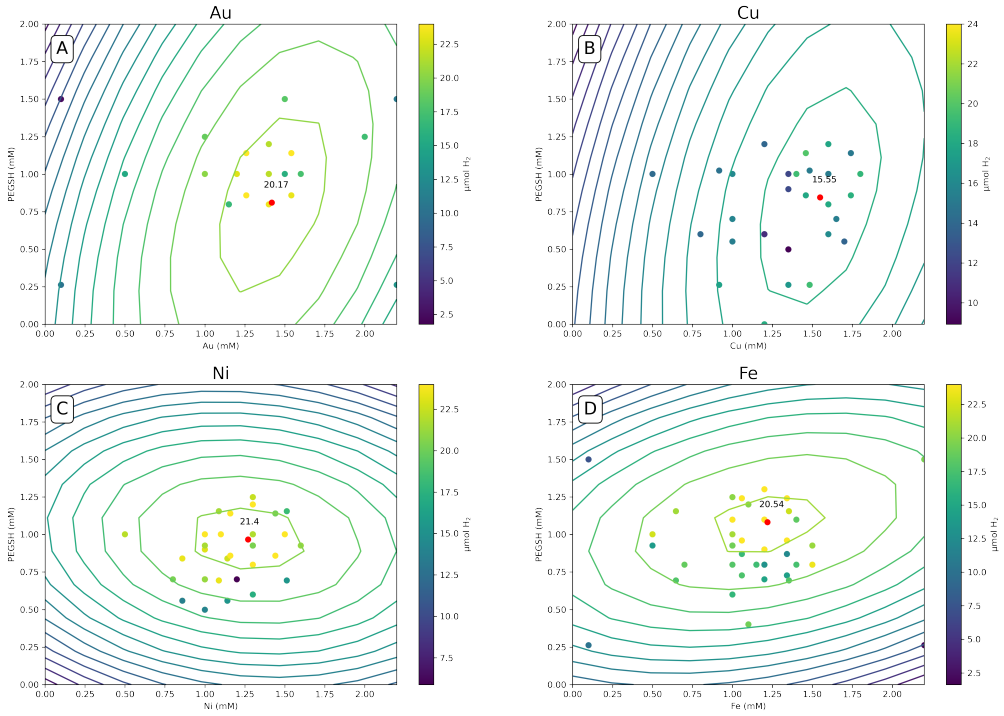


Figure 1: Model contour and raw data plots for the 4 metals. A. Au model and raw data. B. Cu model and raw data. C. Ni model and raw data. D. Fe model and raw data.

It is clear to see the similarities in the noble and non noble metals. Au and Cu have broader contours around ligand concentration whereas Ni and Fe have broader contours around metal concentration. In all cases, the brightest data points fall in or near the highest contour and optimal prediction. The distribution of high spots in all cases indicates there is a region in which high activity is likely to occur. Additionally, the activity variance is likely due to experimental error.

2 TEM Images and Nanoparticle Size Distributions

The TEM images and size distributions were within 2 nm for all four metals. Size distributions were generated using at least 200 nanoparticles per sample. With the addition of the PEGSH ligand, distinct nanoparticles were formed. After more than 45 hours of irradiation, the nanoparticles were imaged and sized as shown in Figure 2.

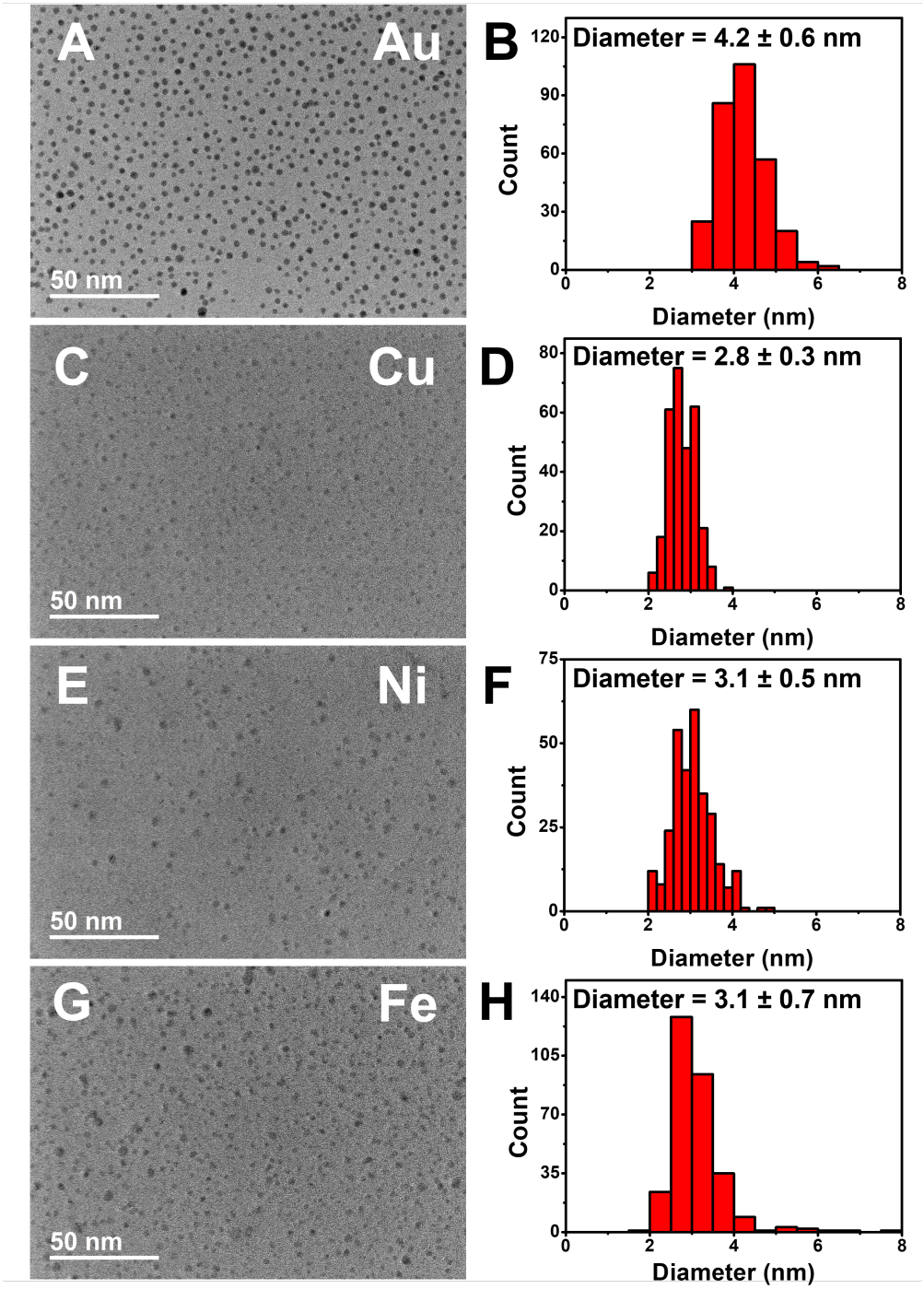


Figure 2: TEM images and nanoparticle size distributions of the four metals. A. Au TEM image with distinct nanoparticles. B. Au nanoparticle size distribution from this sample. C. Cu TEM image with distinct but fewer and smaller nanoparticles compared to Au. D. Cu nanoparticle size distribution. E. Ni TEM image with distinct particles. F. Ni nanoparticle size distribution. G. Fe TEM image with distinct particles. H. Fe nanoparticle size distribution.

The particles were all taken from wells at compositions near but not at the optimally predicted

composition. Though the compositions were not at exactly at the optimal, we have reason to believe their particle size would not be significantly different at these other compositions. The compositions of the wells that the images were taken from are in Table 1.

Table 1: The metal and ligand concentrations of the wells that were taken for TEM imaging and XPS spectra.

Metal	Metal Conc (mM)	PEGSH Conc (mM)	Avg Exp umolH
Au	1.4	1.00	20.81
Cu	1.6	1.00	14.63
Ni	1.3	1.00	20.19
Fe	1.2	1.10	20.63

3 Internal Standard

The internal standard was used to normalize experiments. On every plate, six wells were filled with the exact same composition of a molecular internal standard solution. The average activity and standard error of these wells for each experiment is plotted in Figure 3.

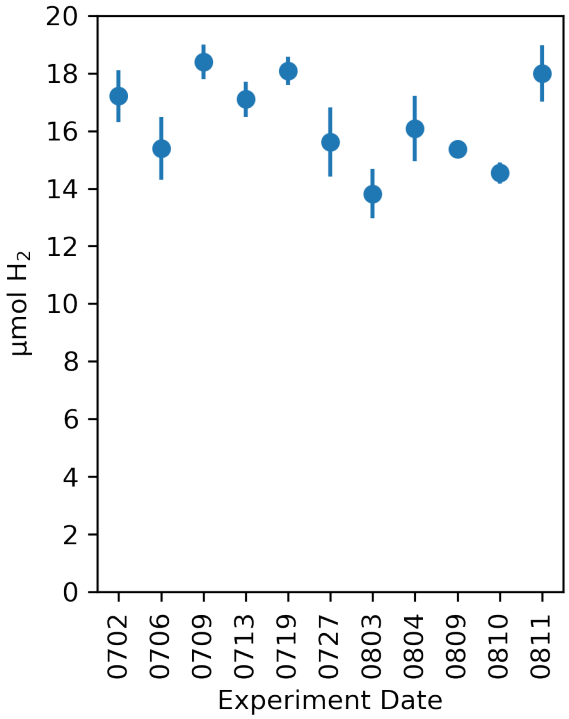


Figure 3: The average internal standard wells performance in µmol H_2 with standard error bars. Each experiment containing internal standard wells were plotted in here and the experiment dates are on the x axis.

Given the same reaction conditions, the internal standard should perform identically. The variation seen within a plate is a feature of the experimental error. The variation seen from

plate to plate is due to variability in the external conditions. These conditions could be due to temperature, humidity, light exposure, or others, and they affect the entire plate. As a result, the performance of the rest of the plate is compared to the internal standard performance to account for external variation.

4 Figure generation

4.1 Generating Figures for the Manuscript

The figures from the manuscript were generated in the following sections. Data was extracted from the raw data files into consolidated json data files that are included and used in the subsequent sections. The main analysis package used to process the data is a python module called doeanalysis. This module is a sub-module for `gespyranto`.

This section documents how the figures for the manuscript have been made with the exception of the TEM images, XPS spectra, and workflow. Each figure is represented in a section.

4.2 Data extraction

This code can only be run by people with access to the Shared GoogleDrive used in the project. This code is not broadly reproducible, but results in two json files that are included in this document which make the subsequent codes reproducible by others. We provide the code here to document how we extracted the data, and store it in a json file.

```
import sys
import os
import json

# Load gespyranto
from gespyranto.plate import Plate
from gespyranto.umolh import umolH
from gespyranto.doeanalysis import analysis

#This cell is used to generate the data dataframe that is used for the rest of this analysis
dir = !find /content/drive/Shareddrives/h2-data-science/data/generation-4 -maxdepth 1 -mindepth 1 -type d
dir = dir.sort()[:-2]

df = pd.DataFrame({})
for i in dir:
  p = Plate(i)
  umolH = p.data['umolH']
  df1 = umolH.dataframe()
  df = pd.concat([df, df1])
df = df.reset_index()
os.chdir('/content/drive/Shareddrives/h2-data-science/publications/3-pure-metal/')
data = df.to_json('data/data.json')

#Internal Standard Dataframe generation
avg = []
error = []
root = '/content/drive/Shareddrives/h2-data-science/data/'
plate_directories = []
for root, dirs, files in os.walk(root):
    for dir in dirs:
        if (os.path.isfile(os.path.join(root, dir, 'output/IntStand.JSON'))):
            plate_directories += [(os.path.join(root, dir))]
            path = (os.path.join(root, dir, 'output/IntStand.JSON'))
            df = pd.read_json(path)
            avg.append(df['umol H2'].max())
            max_ind = df[df['umol H2'] == df['umol H2'].max()].index.values[0]
```

```
            error.append(df.iloc[max_ind]['Std Err'])

paths = [str(i).split('/')[-1] for i in plate_directories]
df1 = pd.DataFrame({'umolH':avg, 'stddev':error, 'paths': paths})

intstand = df1.to_json('data/intstand.json')
```

The data is attached to this pdf here:

supporting-information.org

data.json

intstand.json

These files can be extracted and used in the subsequent sections.

4.3 Loading the data

```
import pandas as pd

df = pd.read_json('data/data.json')
dfint = pd.read_json('data/intstand.json')
```

4.4 Figure 2

```
from gespyranto.doeanalysis import analysis
import matplotlib.pyplot as plt

markers = ['.', 'o', ',', '>']
metals = ['Au', 'Cu',  'Fe', 'Ni']
colors = ['0.3', '0.5', '0.7', '0.1']
fig, axs= plt.subplots(1,1, figsize = (3.25, 4), sharex = True, sharey = True)
for i in range(len(metals)):
  m = analysis(df, [metals[i]], [metals[i], 'PS', 'PEGSH'], 'umolH_max')
  m1 = analysis(df, [metals[i]], [metals[i], 'PS', 'PEGSH'], 'umolH_max_rate')
  axs.scatter(m.y, m1.y, c=colors[i], marker=markers[i], s=10)

  axs.set_xlabel(u'Max \u03bcmol H$_2$')
  axs.set_ylabel(u'Max \u03bcmol H$_2$ Rate')

axs.legend(metals)
for ax in fig.get_axes():
    ax.label_outer()
plt.tight_layout()
for ext in ['png', 'pdf']:
  fig.savefig(f'figures/maxh2vsmaxrate.{ext}', dpi=600)
plt.close()

from pycse.orgmode import Figure
Figure('./figures/maxh2vsmaxrate.png',
       caption='Figure 2 in the manuscript.',
       attributes=(('org', ':width 300'),
                   ('latex', ':placement [H] :width 3.25in')))
```

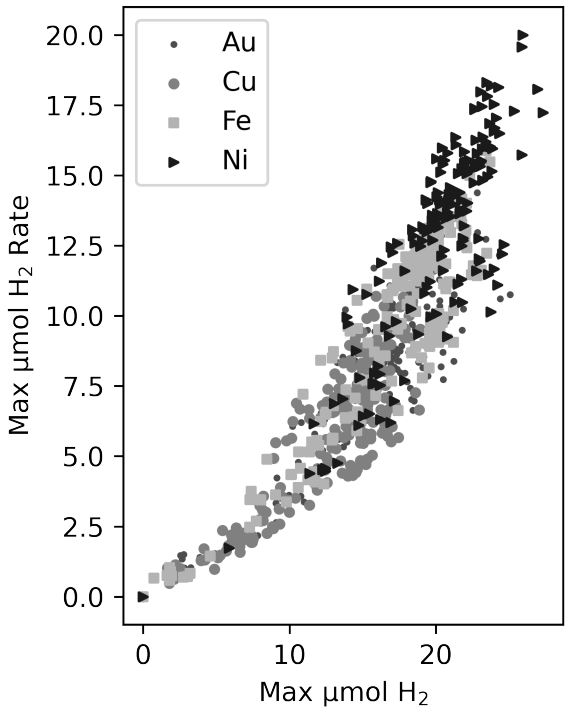


Figure 4: Figure 2 in the manuscript.

4.5 Figure 3

```
import numpy as np

path = '/content/drive/Shareddrives/h2-data-science/data/generation-4/0719CuFeNiAu'
df1  = df[df.directory == path]

mfe = analysis(df1, ['Fe'], ['Fe', 'PS', 'PEGSH'], 'umolH_max')
mni = analysis(df1, ['Ni'], ['Ni', 'PS', 'PEGSH'], 'umolH_max')
mcu = analysis(df1, ['Cu'], ['Cu', 'PS', 'PEGSH'], 'umolH_max')
mau = analysis(df1, ['Au'], ['Au', 'PS', 'PEGSH'], 'umolH_max')

def plot_ps(x, Y05, Y1, labels, met_conc):
  fig, (ax, ax1) = plt.subplots(2,1, figsize=(3.25,8))
  ax.plot(x, Y05[2], c='blue', label=labels[2])
  ax.plot (x, Y05[3], c='red', label=labels[3])
  ax.plot (x, Y05[0], c='green', label=labels[0])
  ax.plot (x, Y05[1], c='orange', label=labels[1])

  ax.axvline(x=0.5, color='k')
  ax.set_title(f'{met_conc[0]}mM Metal', fontsize=15)
  ax.set_ylabel(u'\u03bcmol H$_2$ max', fontsize=12)
  ax.set_xlim(0,1)
  ax.set_yticks(np.linspace(0,int(Y1.max())+6, 5))
  ax.legend(loc='upper center', ncol=2)
  ax.text(0.05,26, 'A', fontsize=20)

  ax1.plot(x, Y1[2], c='blue', label=labels[2])
  ax1.plot (x, Y1[3], c='red', label=labels[3])
  ax1.plot (x, Y1[0], c='green', label=labels[0])
  ax1.plot (x, Y1[1], c='orange', label=labels[1])

```

```
  ax1.axvline(x=0.5, color='k')
  ax1.set_title(f'{met_conc[1]}mM Metal', fontsize=15)
  ax1.set_xlabel('PS Concentration (mM)', fontsize=12)
  ax1.set_ylabel(u'\u03bcmol H$_2$ max', fontsize=12)
  ax1.set_xticks(np.linspace(0, 1, 5))
  ax1.set_xlim(0,1)
  ax1.set_yticks(np.linspace(0,int(Y1.max())+6, 5))
  ax1.text(0.05, 26,'B', fontsize=20)

  plt.legend(loc='upper center', ncol=2)
  plt.tight_layout()
  for ext in ['png', 'pdf']:
    plt.savefig(f'figures/ps_dependence_metal.{ext}', dpi=600)
  plt.close()
  return Figure('./figures/ps_dependence_metal.png',
                caption='Figure 3 in the manuscript',
                attributes=(('org', ':width 500'),
                            ('latex', ':placement [H] :width 3.25in')))

def get_y(m, metal, conc):
  df = m.avg_df()[m.avg_df()[metal] == conc]
  y = df.umolH_max_avg.values
  x = df.PS.values
  return x, y

mods = [mau, mcu, mni, mfe]
labels = ['Au', 'Cu', 'Ni', 'Fe']
Y05 = np.empty((0,6),int)
for i in range(len(mods)):
  Y05 = np.append(Y05, [get_y(mods[i], labels[i], 0.5)[1]], axis=0)

Y1 = np.empty((0, 6), int)
for i in range(len(mods)):
  Y1 = np.append(Y1, [get_y(mods[i], labels[i], 1)[1]], axis=0)

plot_ps(get_y(mods[0], labels[0], 1)[0], Y05, Y1, labels, [0.5,1])
```

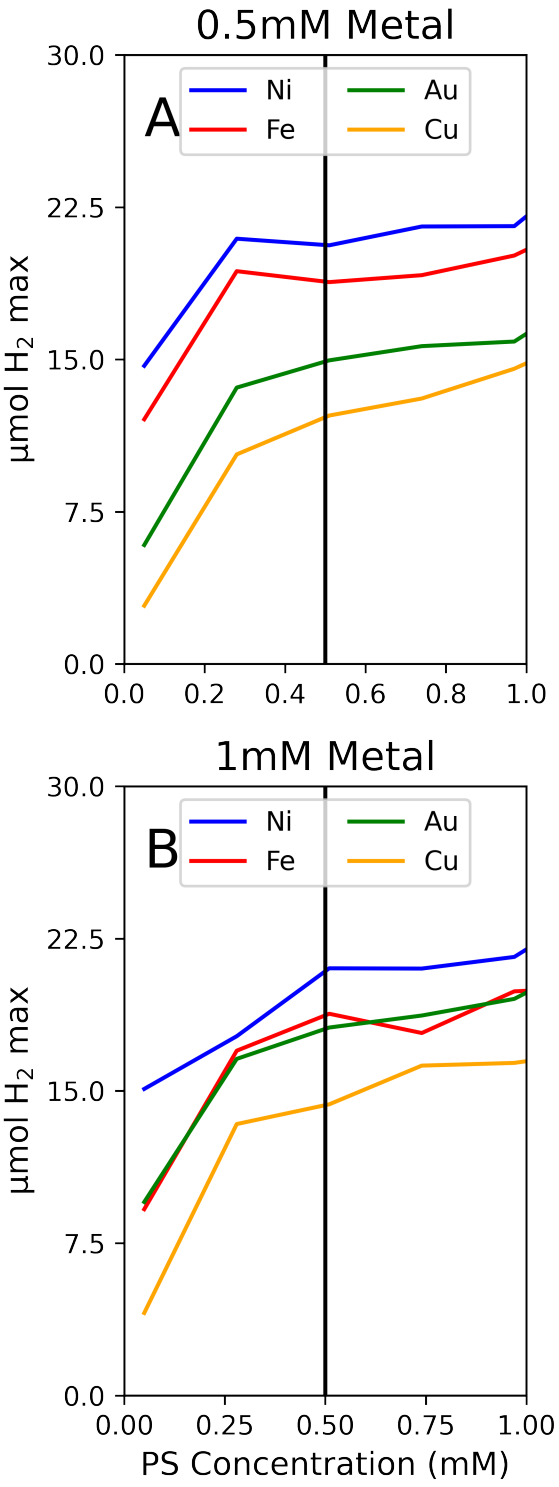


Figure 5: Figure 3 in the manuscript

4.6 Figure 4

```
dir = df.directory.unique()
df1 = df.copy()
df1['path'] = [str(i).split('/')[-1] for i in df1.directory]
df_in = df1[df1.attributes.str.contains('Internal Standard')].copy()

#get normalizing factor
avg = []
std = []
for i in df_in.path.unique():
  avg1 = np.mean(df_in[df_in.path == i].umolH_max)
  err1 = np.std(df_in[df_in.path == i].umolH_max)
  avg.append(avg1)
  std.append(err1)
multi_factor = [avg[0]/i for i in avg]
map = dict(zip(df_in.path.unique(), multi_factor))
df1['int_stand_base'] = 1.0
for i, v in map.items():
  df1['int_stand_base'][df1.path == i] = v
df1['normalize'] = df1.umolH_max * df1.int_stand_base

def get_plot_data(dir, metal, target,cmax, df):
  new_dir = df[df.directory.str.contains(metal)].directory.unique()
  new_df = df[df.directory.isin(new_dir)]

  m = analysis(new_df, [metal], [metal, 'PS', 'PEGSH'], target)

  # Get the data
  par = m.model.params.values
  x1 = np.linspace(0, 2.2, 10)
  x2 = np.linspace(0, 2 , 10)
  x3 = np.linspace(0,2, 10)
  X1,X2, X3 = np.meshgrid(x1, x2, x3)
  X1 = X1.flatten()
  X2 = X2.flatten()
  X3 = X3.flatten()
  def y(X1, X2, X3, par):
      y = (par[0] + par[1]*X1 + par[2]*X2 + par[3]*X3 +
            par[4]*X1**2 + par[5]*X1*X2 + par[6]*X1*X3 +
            par[7]*X2**2 + par[8]*X2*X3 + par[9]*X3**2)
      return y

  Y = y(X1, X2**0*0.5, X3, par)

  size = int(np.cbrt(len(X1)))
  Y = Y[0:size**2].reshape(size, size).T
  X1 = (X1[0::10][0:10])
  X2 = (X2[0::100])

  #plot raw data
  df = df[(df[metal]!=None) & (df[metal]>0)]
  df = df[(df.PS>=0.4)&(df.PS<=0.6)]
  x = df[metal]
  y = df.PEGSH
  z = df['normalize']

  #Optimal point
  opt = m.optimum().x
  return ((X1, X2, Y), (x, y, z), (opt, -m.objective(opt)[0]))


# Au
metal = 'Au'
au = get_plot_data(dir, metal, 'umolH_max', 24, df1)
model = au[0]
data= au[1]
```

```
opt = au[2]
fig, (ax, ax1) = plt.subplots(2,1, figsize = (3.25,8))
im = ax.contour(model[0], model[1], model[2],levels= 15, vmax = 24)
ax.set_xlabel(f'{metal} (mM)')
ax.set_ylabel('PEGSH (mM)')
ax.set_title(f'{metal}', fontsize = 20)
im = ax.scatter(data[0],data[1],c=data[2], vmax = 24)
ax.scatter(opt[0][0], opt[0][-1], c = 'r')
ax.annotate(f'{np.around(opt[1],2)}', (opt[0][0]-0.05, opt[0][-1]+0.1))
ax.text(0.07, 1.8,'A', fontsize = 14,
             bbox=dict(boxstyle='round',
            facecolor='white'))

# Ni
metal = 'Ni'
au = get_plot_data(dir, metal, 'umolH_max', 24, df1)
model = au[0]
data= au[1]
opt = au[2]
im = ax1.contour(model[0], model[1], model[2],levels= 15, vmax = 24)
ax1.set_xlabel(f'{metal} (mM)')
ax1.set_ylabel('PEGSH (mM)')
ax1.set_title(f'{metal}', fontsize = 20)
im = ax1.scatter(data[0],data[1],c=data[2], vmax = 24)
ax1.scatter(opt[0][0], opt[0][-1], c = 'r')
ax1.annotate(f'{np.around(opt[1],2)}', (opt[0][0]-0.05, opt[0][-1]+0.1))
ax1.text(0.07, 1.8,'B', fontsize = 14,
             bbox=dict(boxstyle='round',
            facecolor='white'))

cbar_ax = fig.add_axes([0.2, -0.05, 0.8, 0.04])
cbar = fig.colorbar(im, cax=cbar_ax, orientation = "horizontal")
cbar.set_label(u'\u03bcmol H$_2$')

plt.tight_layout()
for ext in ['png', 'pdf']:
  plt.savefig(f'figures/model_raw_opt.{ext}', dpi = 600)
plt.close()

Figure('./figures/model_raw_opt.png',
       caption='Figure 4 in the manuscript',
       attributes=(('org', ':width 500'),
                           ('latex', ':placement [H] :width 3.25in')))
```

/var/folders/3q/ht_2mtk52hl7ydxrcr87z2gr0000gn/T/ipykernel_6969/278245199.py:18: SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy df1['int_stand_base'][df1.path == i] = v /var/folders/3q/ht_2mtk52hl7ydxrcr87z2gr0000gn/T/ipykernel_6969/278245199.py:100: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect. plt.tight_layout()

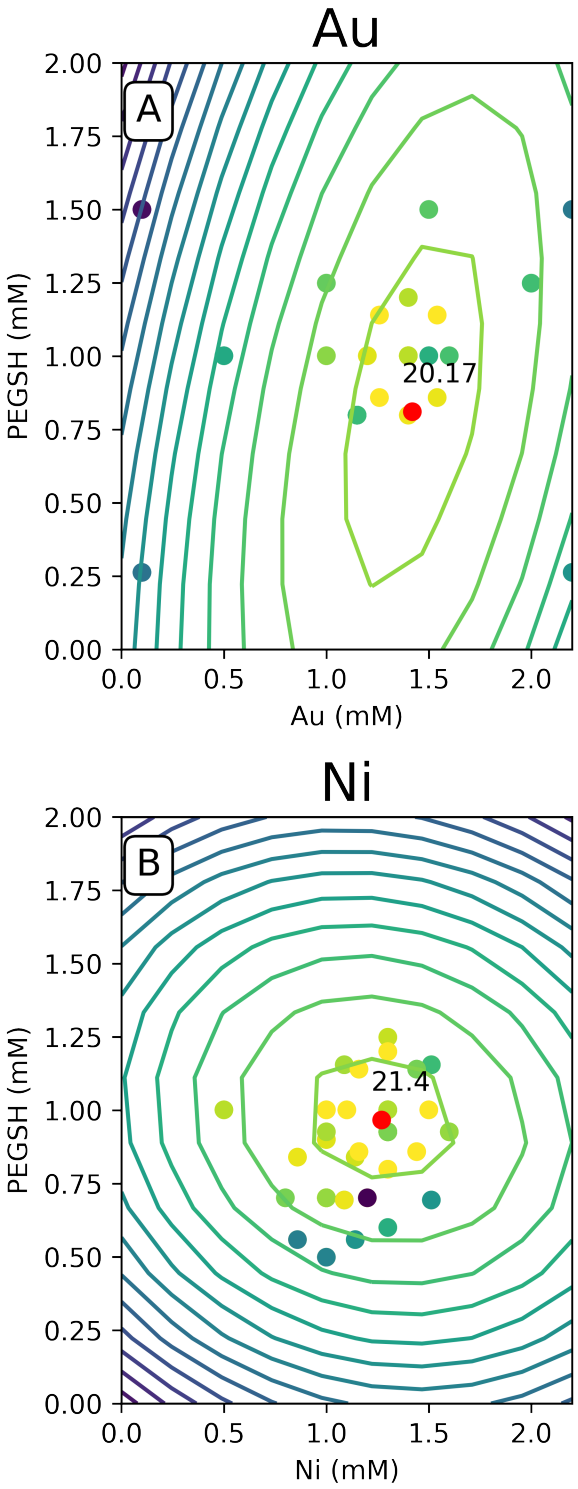


Figure 6: Figure 4 in the manuscript

4.7 Figure 5

```
import autograd.numpy as np
from autograd import elementwise_grad, hessian
from scipy.optimize import minimize
from scipy.stats.distributions import t
from sklearn.preprocessing import PolynomialFeatures

#dir = dir[:-1]
def get_error(m):
  X = m.X
  y = m.y
  _theta, res, rank, s = np.linalg.lstsq(X, y, rcond=-1)
  H = X.T @ X
  sse = 0.5 * np.sum((X @ _theta -  y)**2)
  mse = sse / len(X)

  X = [m.optimum().x]
  X = PolynomialFeatures(2).fit_transform(X)
  error = np.array([np.sqrt(mse * row @ np.linalg.pinv(H) @ row) for row in X])
  return error[0]

def plot_error(error = 'SE'):
  metals = ['Au', 'Cu', 'Fe', 'Ni']
  first = []
  last = []

  for i in metals:
    new_df = df[df.directory.str.contains(i)]
    new_dir = df[df.directory.str.contains(i)].directory.unique()

    # generate info for first plot
    m = analysis(new_df[new_df.directory.str.contains(f'{new_dir[0]}')], [i], [i, 'PS', 'PEGSH'], 'umolH_max')
    first.append(get_error(m))

    m1 = analysis(new_df, [i], [i, 'PS', 'PEGSH'], 'umolH_max')
    last.append(get_error(m1))

  x = np.arange(len(metals))  # the label locations
  width = 0.35  # the width of the bars

  fig, ax = plt.subplots(figsize = (3.25, 4))
  rects1 = ax.bar(x - width / 2, first, width, label='First Iteration', color='teal')
  rects2 = ax.bar(x + width / 2, last, width, label='Last Iteration', color='orange')

  # Add some text for labels, title and custom x-axis tick labels, etc.
  err_lab = u'Standard Error (\u03bcmolH$_2$)'

  ax.set_ylabel(err_lab)
  ax.set_xticks(x)
  ax.set_xticklabels(metals)
  ax.legend()

  fig.tight_layout()
  for ext in ['png', 'pdf']:
    plt.savefig(f'figures/optimal_{error}.{ext}', dpi=600)
  plt.close()
  return Figure(f'./figures/optimal_{error}.png',
                caption='Figure 5 in the manuscript.',
                attributes=(('org', ':width 500'),
                            ('latex', ':placement [H] :width 3.25in')))


plot_error('SE')
```

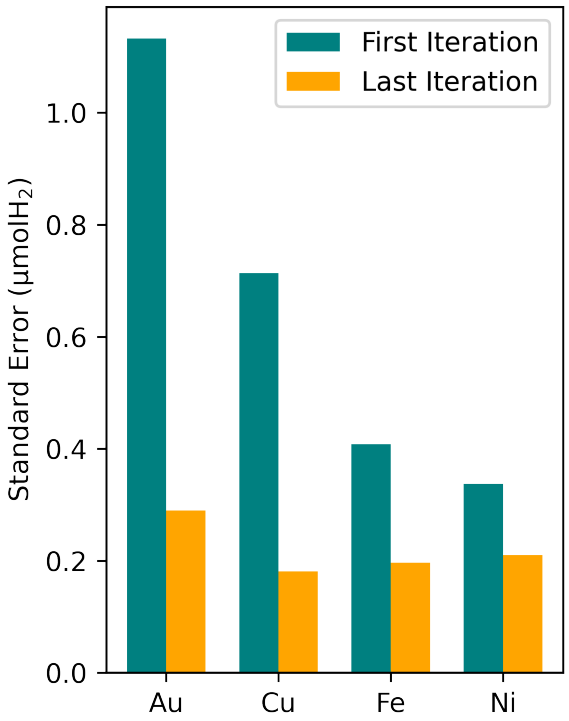


Figure 7: Figure 5 in the manuscript.

4.8 SI Figure for all contour plots

```
dir = df.directory.unique()
df1 = df.copy()
df1['path'] = [str(i).split('/')[-1] for i in df1.directory]
df_in = df1[df1.attributes.str.contains('Internal Standard')].copy()

#get normalizing factor
avg = []
std = []
for i in df_in.path.unique():
  avg1 = np.mean(df_in[df_in.path == i].umolH_max)
  err1 = np.std(df_in[df_in.path == i].umolH_max)
  avg.append(avg1)
  std.append(err1)
multi_factor = [avg[0]/i for i in avg]
map = dict(zip(df_in.path.unique(), multi_factor))
df1['int_stand_base'] = 1.0
for i, v in map.items():
  df1['int_stand_base'][df1.path == i] = v
df1['normalize'] = df1.umolH_max * df1.int_stand_base

def get_plot_data(dir, metal, target,cmax, df):
  new_dir = df[df.directory.str.contains(metal)].directory.unique()
  new_df = df[df.directory.isin(new_dir)]

  m = analysis(new_df, [metal], [metal, 'PS', 'PEGSH'], target)

  # Get the data
  par = m.model.params.values
  x1 = np.linspace(0, 2.2, 10)
  x2 = np.linspace(0, 2 , 10)
```

```
    x3 = np.linspace(0,2, 10)
    X1,X2, X3 = np.meshgrid(x1, x2, x3)
    X1 = X1.flatten()
    X2 = X2.flatten()
    X3 = X3.flatten()
    def y(X1, X2, X3, par):
        y = (par[0] + par[1]*X1 + par[2]*X2 + par[3]*X3 +
              par[4]*X1**2 + par[5]*X1*X2 + par[6]*X1*X3 +
              par[7]*X2**2 + par[8]*X2*X3 + par[9]*X3**2)
        return y

    Y = y(X1, X2**0*0.5, X3, par)

    size = int(np.cbrt(len(X1)))
    Y = Y[0:size**2].reshape(size, size).T
    X1 = (X1[0::10][0:10])
    X2 = (X2[0::100])

    #plot raw data
    df = df[(df[metal]!=None) & (df[metal]>0)]
    df = df[(df.PS>=0.4)&(df.PS<=0.6)]
    x = df[metal]
    y = df.PEGSH
    z = df['normalize']

    #Optimal point
    opt = m.optimum().x
    return ((X1, X2, Y), (x, y, z), (opt, -m.objective(opt)[0]))

# Au
metal = 'Au'
au = get_plot_data(dir, metal, 'umolH_max', 24, df1)
model = au[0]
data= au[1]
opt = au[2]
fig, ax = plt.subplots(2,2, figsize = (20, 14))
im = ax[0,0].contour(model[0], model[1], model[2],levels= 15, vmax = 24)
ax[0,0].set_xlabel(f'{metal} (mM)')
ax[0,0].set_ylabel('PEGSH (mM)')
ax[0,0].set_title(f'{metal}', fontsize = 20)
im = ax[0,0].scatter(data[0],data[1],c=data[2], vmax = 24)
ax[0,0].scatter(opt[0][0], opt[0][-1], c = 'r')
ax[0,0].annotate(f'{np.around(opt[1],2)}', (opt[0][0]-0.05, opt[0][-1]+0.1))
ax[0,0].text(0.07, 1.8,'A', fontsize = 20,
             bbox=dict(boxstyle='round',
            facecolor='white'))
cbar = fig.colorbar(im, ax = ax[0,0])
cbar.set_label(u'\u03bcmol H$_2$')

# Cu
metal = 'Cu'
au = get_plot_data(dir, metal, 'umolH_max', 24, df1)
model = au[0]
data= au[1]
opt = au[2]
im = ax[0,1].contour(model[0], model[1], model[2],levels= 15, vmax = 24)
ax[0,1].set_xlabel(f'{metal} (mM)')
ax[0,1].set_ylabel('PEGSH (mM)')
ax[0,1].set_title(f'{metal}', fontsize = 20)
im = ax[0,1].scatter(data[0],data[1],c=data[2], vmax = 24)
ax[0,1].scatter(opt[0][0], opt[0][-1], c = 'r')
ax[0,1].annotate(f'{np.around(opt[1],2)}', (opt[0][0]-0.05, opt[0][-1]+0.1))
ax[0,1].text(0.07, 1.8,'B', fontsize = 20,
             bbox=dict(boxstyle='round',
            facecolor='white'))
cbar = fig.colorbar(im, ax = ax[0,1])
cbar.set_label(u'\u03bcmol H$_2$')

```

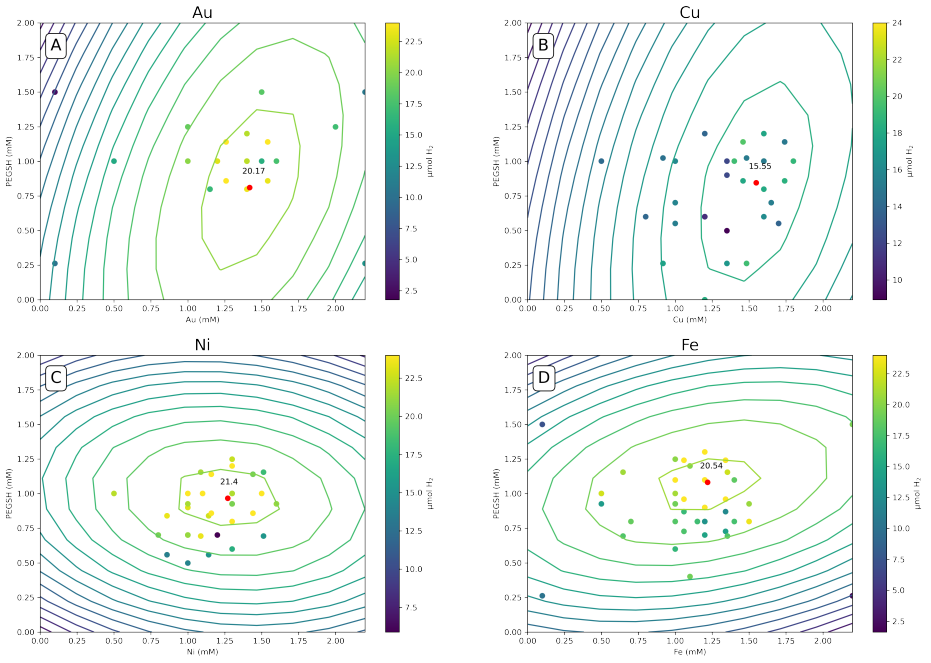
```
# Ni
metal = 'Ni'
au = get_plot_data(dir, metal, 'umolH_max', 24, df1)
model = au[0]
data= au[1]
opt = au[2]
im = ax[1,0].contour(model[0], model[1], model[2],levels= 15, vmax = 24)
ax[1,0].set_xlabel(f'{metal} (mM)')
ax[1,0].set_ylabel('PEGSH (mM)')
ax[1,0].set_title(f'{metal}', fontsize = 20)
im = ax[1,0].scatter(data[0],data[1],c=data[2], vmax = 24)
ax[1,0].scatter(opt[0][0], opt[0][-1], c = 'r')
ax[1,0].annotate(f'{np.around(opt[1],2)}', (opt[0][0]-0.05, opt[0][-1]+0.1))
ax[1,0].text(0.07, 1.8,'C', fontsize = 20,
             bbox=dict(boxstyle='round',
            facecolor='white'))
cbar = fig.colorbar(im, ax = ax[1,0])
cbar.set_label(u'\u03bcmol H$_2$')

# Fe
metal = 'Fe'
au = get_plot_data(dir, metal, 'umolH_max', 24, df1)
model = au[0]
data= au[1]
opt = au[2]
im = ax[1,1].contour(model[0], model[1], model[2],levels= 15, vmax = 24)
ax[1,1].set_xlabel(f'{metal} (mM)')
ax[1,1].set_ylabel('PEGSH (mM)')
ax[1,1].set_title(f'{metal}', fontsize = 20)
im = ax[1,1].scatter(data[0],data[1],c=data[2], vmax = 24)
ax[1,1].scatter(opt[0][0], opt[0][-1], c = 'r')
ax[1,1].annotate(f'{np.around(opt[1],2)}', (opt[0][0]-0.05, opt[0][-1]+0.1))
ax[1,1].text(0.07, 1.8,'D', fontsize = 20,
             bbox=dict(boxstyle='round',
            facecolor='white'))
cbar = fig.colorbar(im, ax = ax[1,1])
cbar.set_label(u'\u03bcmol H$_2$')

plt.savefig(f'figures/model_raw_opt_4_metals.png', dpi = 300)
plt.close()
Figure('./figures/model_raw_opt_4_metals.png',
       attributes=(('org', ':width 500'),
                   ('latex', ':placement [H] :width 6in')))
```

/var/folders/3q/ht_2mtk52hl7ydxrcr87z2gr0000gn/T/ipykernel_6969/2523530347.py:18: SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy df1['int_stand_base'][df1.path == i] = v



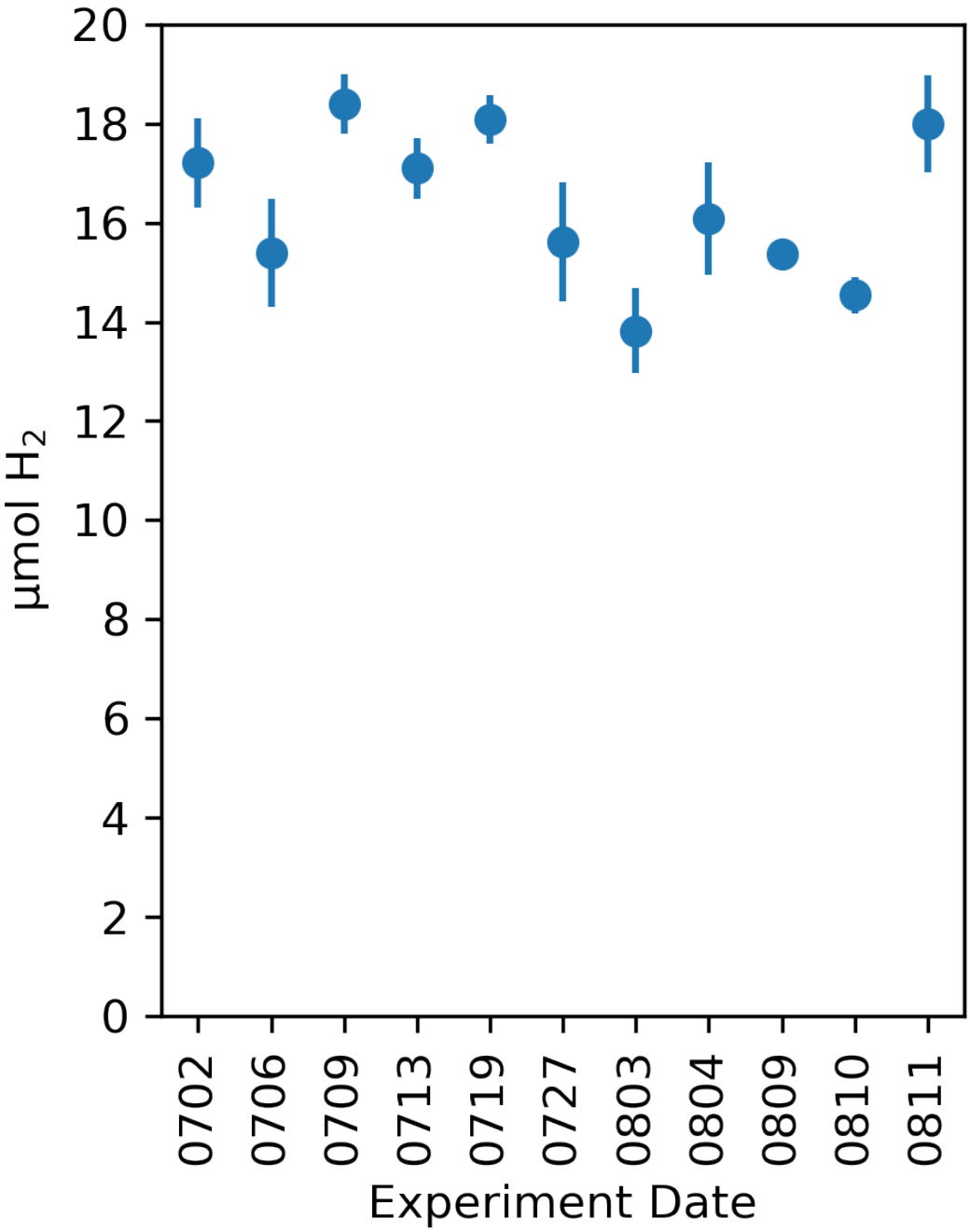
4.9 SI Figure on internal standards

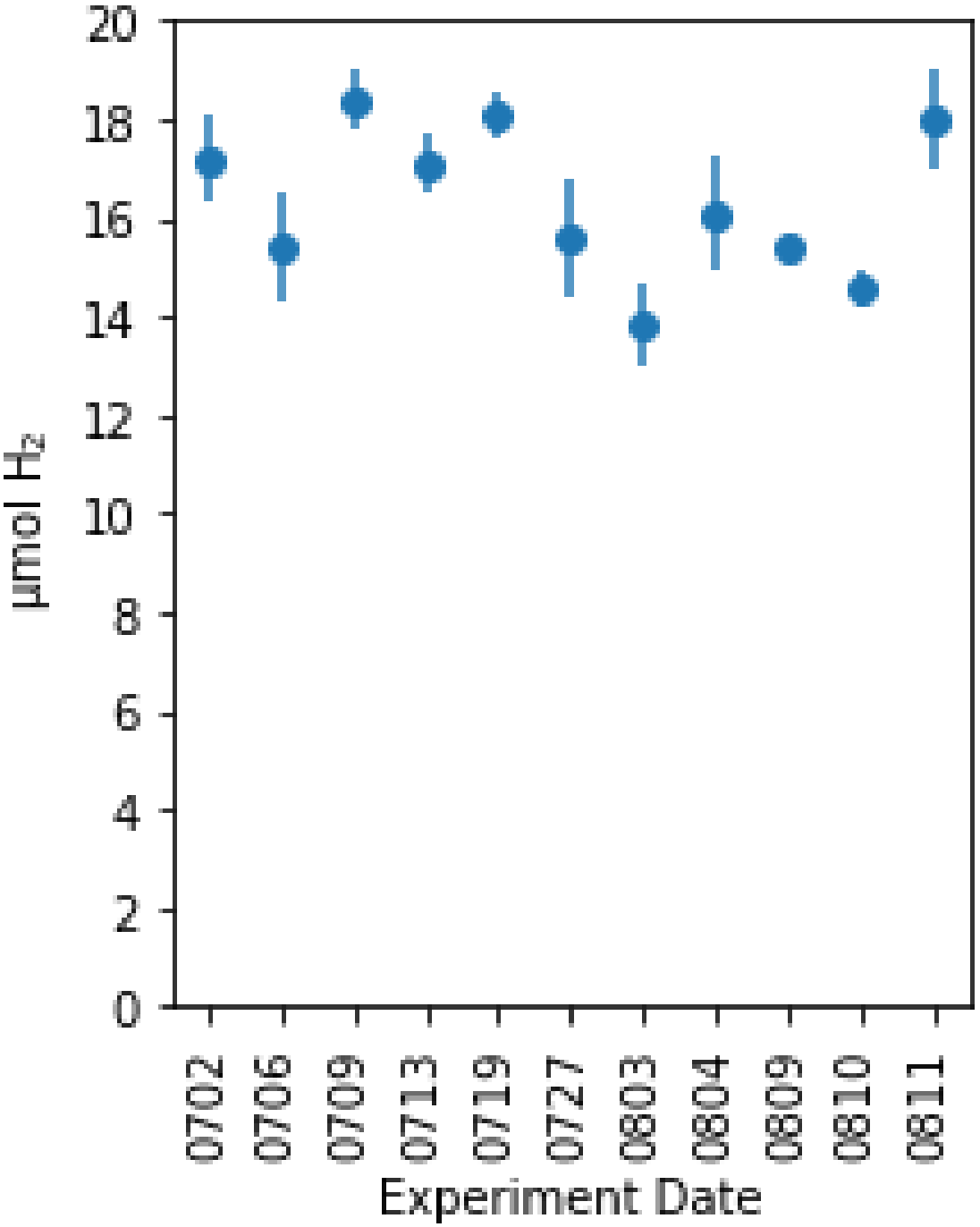
```
dfint['paths'] = [i[:4] for i in dfint.paths]
dfint.sort_values('paths', inplace=True)
dfint = dfint[~dfint.paths.str.contains('0820|0901')]

fig, ax = plt.subplots(figsize = (3.25,4))
ax.errorbar(x=dfint.paths, y = dfint.umolH, yerr = dfint.stddev, fmt="o")
ax.set_yticks(np.arange(0,22, 2))
ax.set_xticklabels(dfint.paths, rotation=90)
ax.set_ylabel(u'\u03bcmol H$_2$')
ax.set_xlabel('Experiment Date')
fig.tight_layout()
plt.savefig(f'figures/internal_standard.png', dpi = 300)

Figure('./figures/internal_standard.png',
       attributes=(('org', ':width 500'),
                   ('latex', ':placement [H] :width 6in')))
```

/var/folders/3q/ht_2mtk52hl7ydxrcr87z2gr0000gn/T/ipykernel_6969/2727707073.py:8: UserWarning: FixedFormatter should only be used together with FixedLocator ax.set_xticklabels(dfint.paths, rotation=90)


20
18
16
14
12
10
8
6
4
2
0
µmol H_2
0702
0706
0709
0713
0719
0727
0803
0804
0809
0810
0811
Experiment Date


20
18
16
14
12
10
8
6
4
2
0
μmol H₂
0702
0706
0709
0713
0719
0727
0803
0804
0809
0810
0811
Experiment Date