Topics in Catalysis

Simulating temperature programmed desorption of oxygen on Pt(111) using DFT derived coverage dependent desorption barriers

Spencer D. Miller Vladimir V. Pushkarev
Andrew J. Gellman John R. Kitchin

January 30, 2016

Contents

1 Supporting information

This section describes the data used in preparation of the figures in the manuscript. We also describe the leading edge analysis we performed. The actual data is in the last sections, and largely consist of tables of data. When the native document format (org-mode) is used, it is straightforward to use these tables as data sources, which is done in the sections on figure generation and the leading edge analysis.

Manuscript file in native format:

Supporting information file in native format: Note that this attached file contains data not included in the pdf version because the size of the data tables leads to a 900 page supporting information file in PDF.

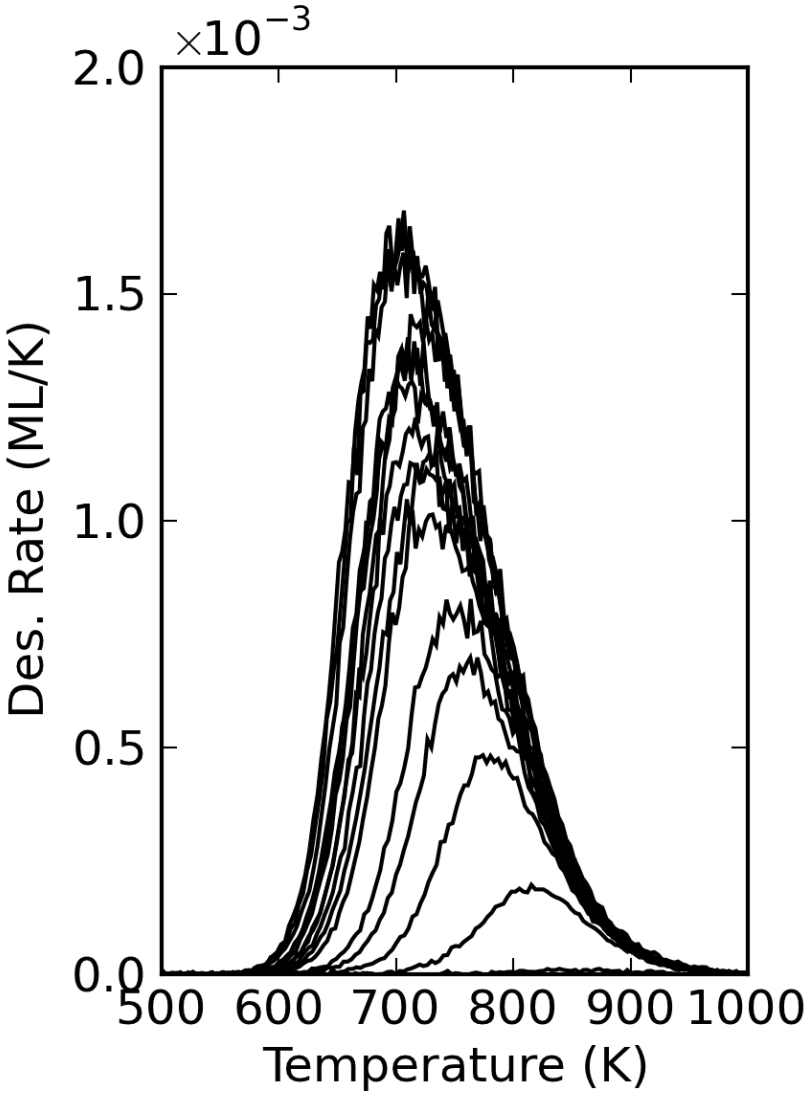
1.1 Manuscript figure generation

This section contains all of the code and data used to generate the figures for the manuscript.

1.1.1 Figure 1

```
from numpy import *
from pylab import *

import scipy.io

rcParams['font.size'] = 12
```

```python
rcParams['axes.formatter.limits'] = -3,4;
rcParams['figure.figsize'] = (3,4);
rcParams['lines.markersize'] = 7;
rcParams['figure.subplot.bottom'] = 0.12;
rcParams['figure.subplot.top'] = 0.91;
rcParams['figure.subplot.right'] = 0.9;
rcParams['figure.subplot.left'] = 0.22;
rcParams['axes.labelsize'] = 12;

axe = subplot(111)
axe.yaxis.set_major_formatter(ScalarFormatter(useMathText=True))

#Reads in matlab file creating a dictionary with the following entires
data = scipy.io.loadmat('analysis/coverage-normalized.mat')
for i in range(data['T'].shape[1]):

    t = data['T'][0,i][0]
    m = data['M'][0,i][0]
    plot(t,m,'k')

#Set Plot Axis;
ylim(0,2*10**-3);
xlim(500,1000);
axe = gca();

xlabel('Temperature (K)')
ylabel('Des. Rate (ML/K)')

for ext in ['png','eps','pdf']:
    savefig('figures/fig1.{0}'.format(ext), dpi=300)
show()
```

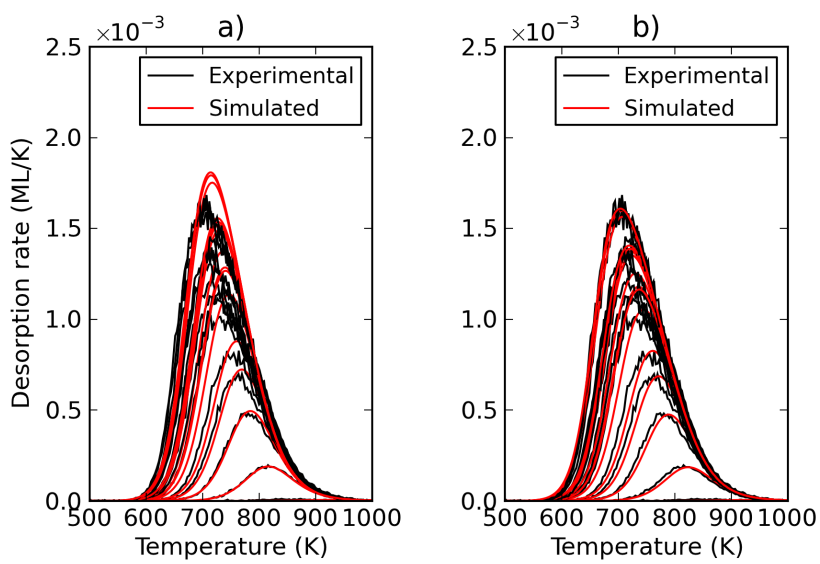


1.1.2 Figure 2

```
from scipy.io import loadmat, savemat
from scipy.integrate import odeint
from scipy.optimize import leastsq
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams

```

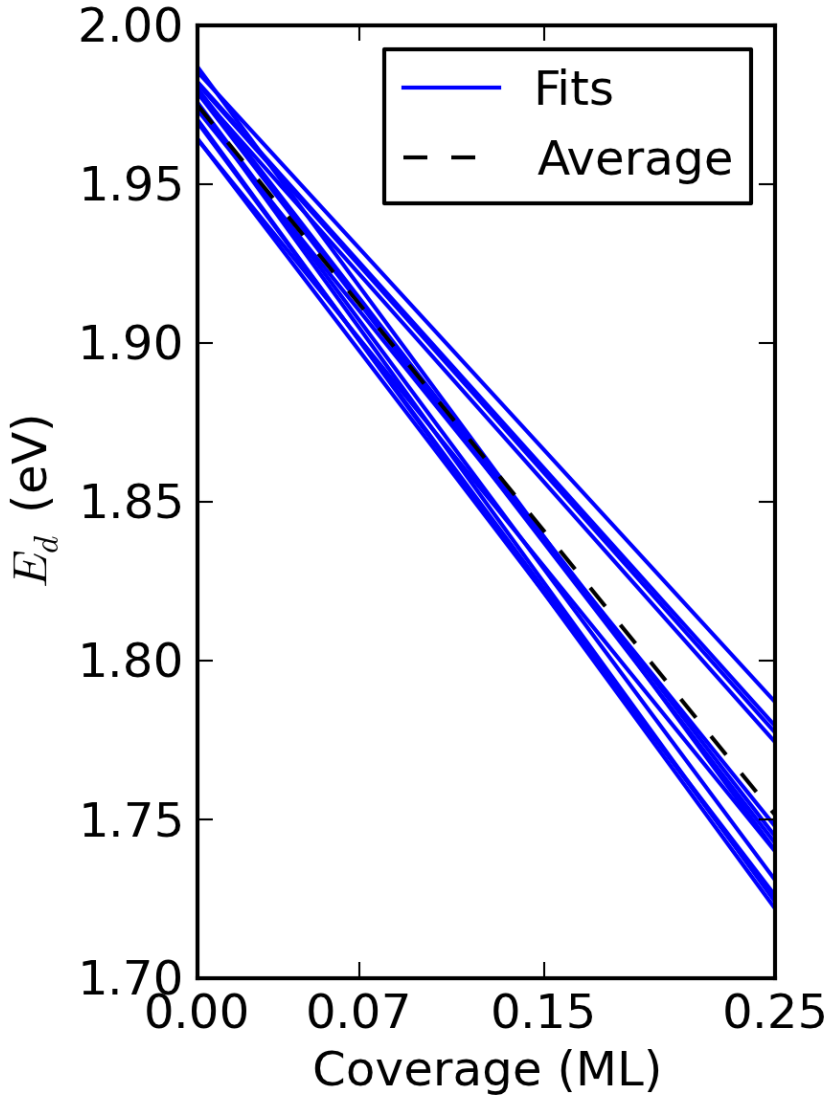
```python

rcParams['font.size'] = 12
rcParams['axes.formatter.limits'] = -3,4;
rcParams['figure.figsize'] = (6,4);
rcParams['lines.markersize'] = 7;
rcParams['figure.subplot.bottom'] = 0.12;
rcParams['figure.subplot.top'] = 0.91;
rcParams['figure.subplot.right'] = 0.9;
rcParams['figure.subplot.left'] = 0.22;
rcParams['axes.labelsize'] = 12;
rcParams['legend.fontsize'] = 11

data = loadmat('analysis/coverage-normalized.mat')

# Here is the ode that simulates 2nd order desorption
def myodefunc(theta, T,  Ed0, alpha):
    kb = 8.617e-5;                # gas constant
    beta = 2                      # heating rate
    Ed = Ed0 + alpha*theta        # coverage dependent desorption barrier
    k = 3 * 10**12*np.exp(-Ed / kb / T)   # rate constant for desorption
    dthetadT = -k / beta * theta**2
    return dthetadT

# a low coverage line
plt.figure()
axe = plt.subplot(121)
axe.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))
for i in range(16):
    theta0 = data['theta0'][0,i]
    T = data['T'][0, i][0]
    M = data['M'][0, i][0]

    # initial parameters
    Ed0 = 1.956
    alpha = -0.584

    X, infodict = odeint(myodefunc, theta0,
                         T, args=(Ed0, alpha),
                         full_output=True)

    # this is the solution
    theta = X[:, 0].T

    ## ### now, plot
    plt.title( 'a)')
    plt.plot(T, M, 'k')
    plt.plot(T, -myodefunc(theta, T, Ed0, alpha),'r')
    plt.ylim([0, 2.5e-3])
    plt.legend(['Experimental','Simulated'])

plt.xlabel('Temperature (K)')
plt.ylabel('Desorption rate (ML/K)')

# a high coverage line
axe = plt.subplot(122)
axe.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))
```

```python
for i in range(16):
    theta0 = data['theta0'][0,i]
    T = data['T'][0,i][0]
    M = data['M'][0,i][0]

    # initial parameters
    Ed0 = 1.979
    alpha = -0.819

    X, infodict = odeint(myodefunc, theta0,
                         T, args=(Ed0, alpha),
                         full_output=True)

    # this is the solution
    theta = X[:,0].T

    ## ### now, plot
    ## ##
    plt.plot(T,M,'k')
    plt.plot(T,-myodefunc(theta, T, Ed0, alpha),'r')
    plt.ylim([0, 2.5e-3])
    plt.legend(['Experimental','Simulated'])
    plt.title('b)')

plt.xlabel('Temperature (K)')
plt.subplots_adjust(left=0.14, right=0.95, wspace=0.47)

for ext in ['png','eps','pdf']:
    plt.savefig('figures/fig2.{0}'.format(ext), dpi=300)
plt.show()
```



1.1.3 Figure 3

```
from numpy import *
import scipy.io
from pylab import *

rcParams['font.size'] = 12
rcParams['axes.formatter.limits'] = -3,4;
rcParams['figure.figsize'] = (3,4);
rcParams['lines.markersize'] = 7;
rcParams['figure.subplot.bottom'] = 0.12;
rcParams['figure.subplot.top'] = 0.95;
rcParams['figure.subplot.right'] = 0.95;
rcParams['figure.subplot.left'] = 0.22;
rcParams['axes.labelsize'] = 12;
rcParams['legend.fontsize']=12

avgb = average(array(data)[1:,0])
avgm = average(array(data)[1:,1])

theta = np.linspace(0, 0.25)
for b,m in data[2:]:
    h1 = plot(theta, b + m*theta, 'b-')

h1[-1].set_label('Fits')

h2 = plot(theta, avgb + avgm*theta, 'k--',label='Average')
legend(loc='best')
xlabel('Coverage (ML)')
ylabel('$E_d$ (eV)')
subplots_adjust(left=0.25, right=0.92)

xticks([0, 0.07, 0.15, 0.25])
for ext in ['png','eps','pdf']:
    savefig('figures/fig3.{0}'.format(ext), dpi=300)
show()
```

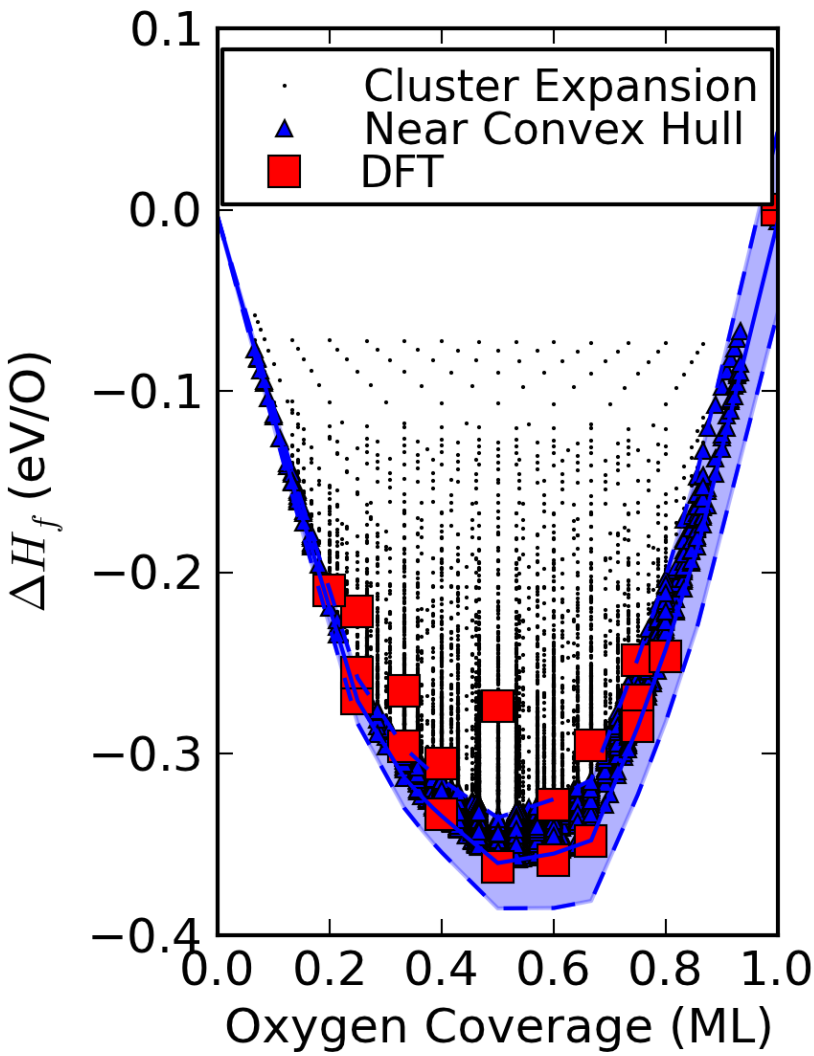


1.1.4 Figure 4

```
#This script will plot several aspects of the cluster expansion convex hull
#DFT based heats of formation
#cluster expansion based heats of formation
#cluster expansion based convex hull
#cluster expansion based heats of formation near the convex hull
#range of heats of formation considered "stable"

```

```python
import os
import sys
from pylab import *
rcParams['font.size'] = 12
rcParams['axes.formatter.limits'] = -3,4;
rcParams['legend.fontsize'] =  11;
rcParams['figure.figsize'] = (3,4);
rcParams['lines.markersize'] = 7;
rcParams['figure.subplot.bottom'] = 0.12;
rcParams['figure.subplot.top'] = 0.91;
rcParams['figure.subplot.right'] = 0.95;
rcParams['figure.subplot.left'] = 0.3;
rcParams['axes.labelsize'] = 12;

DFT = np.array(DFT)
DE = np.array(DE)
CHULL = np.array(CHULL)
SHF = np.array(SHF)

stableRange = 50./1000;

dftHfCoverage = DFT[:, 0];
dftHfEnergy = DFT[:, 1]/1000.;
ceHfCoverage = DE[:, 0];
ceHfEnergy = DE[:, 1]/1000.;
stableCeHfCoverage = SHF[:, 0];
stableCeHfEnergy = SHF[:, 1]/1000.;
convexHullCoverage = CHULL[:, 0];
convexHullEnergy = CHULL[:, 1]/1000.;
lowConvexHullCoverage = convexHullCoverage;
lowConvexHullEnergy = convexHullEnergy - stableRange*convexHullCoverage;
highConvexHullCoverage = convexHullCoverage;
highConvexHullEnergy =  convexHullEnergy + stableRange*convexHullCoverage;

#Plot heats of formation
axe = subplot(111)
axe.yaxis.set_major_formatter(ScalarFormatter(useMathText=True))

ceHfPlot = plot(ceHfCoverage,ceHfEnergy,'ko',markersize=0.5,label='Cluster Expansion');
stableCeHfPlot = plot(stableCeHfCoverage,stableCeHfEnergy,'b^',markersize=4,label='Near Convex Hull');
dftHfPlot = plot(dftHfCoverage, dftHfEnergy, 'rs', markersize=8, label='DFT');

convexHullPlot = plot(convexHullCoverage, convexHullEnergy,'b-');
lowConvexHullPlot = plot(lowConvexHullCoverage, lowConvexHullEnergy,'b--');
highConvexHullPlot = plot(highConvexHullCoverage, highConvexHullEnergy,'b--');
fill_between(convexHullCoverage,lowConvexHullEnergy, highConvexHullEnergy, color='b', alpha=0.3);

ylabel(r'$\Delta H_{f}$ (eV/O)');
xlabel("Oxygen Coverage (ML)");
xlim([0,1])
legend(loc='upper center',
       numpoints=1,
       labelspacing=0,
       columnspacing=0.01);

ylim(-0.400, 0.100)
```

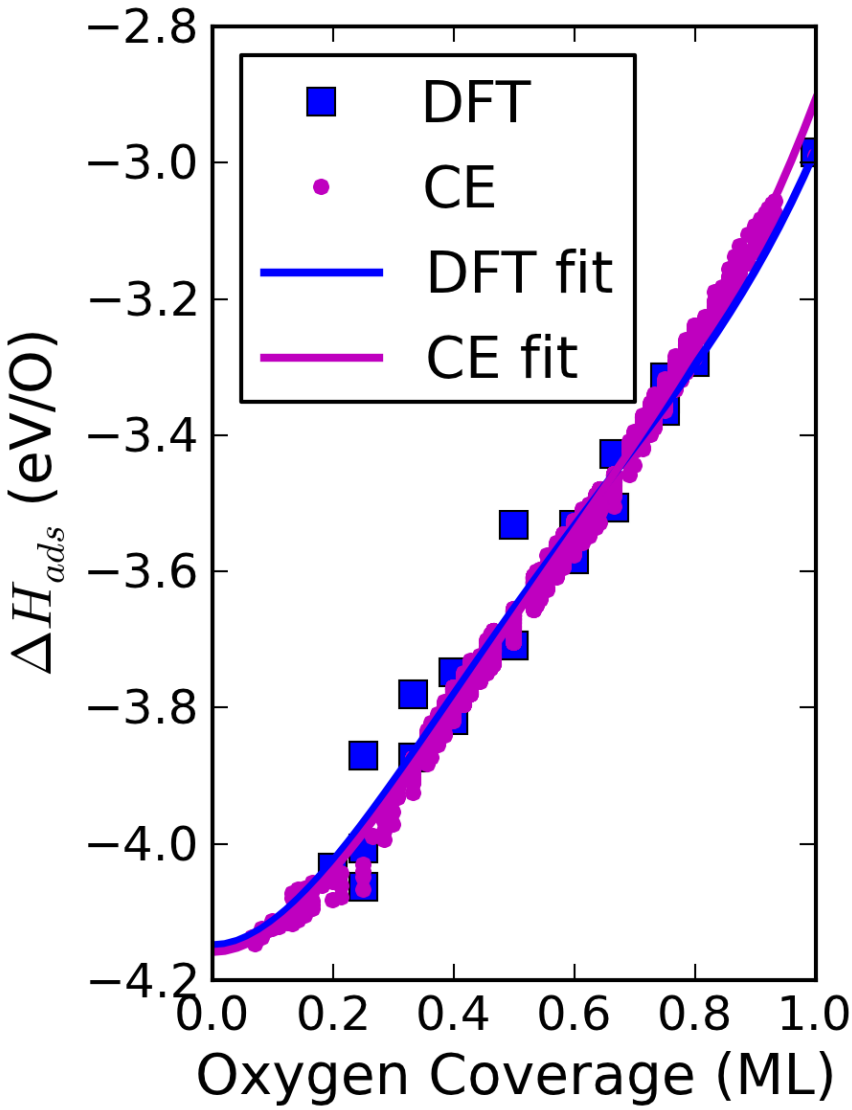
```
for ext in ['png','eps','pdf']:
    savefig('figures/fig4.{0}'.format(ext), dpi=300)
show()
```



1.1.5 Figure 5

```python
import numpy as np
import sys
import matplotlib
from pylab import *

rcParams['font.size'] = 12
rcParams['axes.formatter.limits'] = -3,4;
rcParams['figure.figsize'] = (3,4);
rcParams['lines.linewidth'] = 2;
rcParams['lines.markersize'] = 7;
rcParams['figure.subplot.bottom'] = 0.12;
rcParams['figure.subplot.top'] = 0.95;
rcParams['figure.subplot.right'] = 0.95;
rcParams['figure.subplot.left'] = 0.25;
rcParams['axes.labelsize'] = 14;

H1ML = -2.98497267465
# Hads = Hf/theta + Hads(1ML)
DE = np.array(DE)
ce_theta = DE[:, 0]
ce_hf = DE[:, 1]/1000. # convert to eV
ce_hads = ce_hf/ce_theta + H1ML

DFT = np.array(DFT)
dft_theta = DFT[:,0]
dft_hf = DFT[:,1] / 1000.
dft_hads = dft_hf / dft_theta +  H1ML

## Constrained fits of DFT
A = np.vstack([dft_theta**4, dft_theta**3, dft_theta**2, np.ones(len(dft_theta))]).T
dft_pars = np.linalg.lstsq(A, dft_hads)[0]
print 'DFT fit pars = ',dft_pars

## Constrained fits of CE
A = np.vstack([ce_theta**4, ce_theta**3, ce_theta**2, np.ones(len(ce_theta))]).T
ce_pars = np.linalg.lstsq(A, ce_hads)[0]
print 'CE fit pars = ',ce_pars

plot(dft_theta, dft_hads,'bs ', label='DFT')
plot(ce_theta, ce_hads,'m. ', label='CE')

# now plot fits
theta = np.linspace(0,1)
A = np.vstack([theta**4, theta**3, theta**2, np.ones(len(theta))]).T
plot(theta, np.dot(A, dft_pars),'b-', label='DFT fit')
plot(theta, np.dot(A, ce_pars),'m-', label='CE fit')
legend(loc='upper left')
ylim([-4.2, -2.8])

ylabel(r'$\Delta H_{ads}$ (eV/O)');
xlabel("Oxygen Coverage (ML)");

for ext in ['png','eps','pdf']:
    savefig('figures/fig5.{0}'.format(ext), dpi=300)
show()
```

```
DFT fit pars =  [ 2.67693764 -5.63713503  4.12922348 -4.14884994]
CE fit pars =  [ 2.55245771 -5.24686157  3.94694687 -4.15873743]
```



1.1.6 Figure 6

1. data fitting First, we do the fitting. We want to fit a single alpha and kappa to the data

```python
from scipy.io import loadmat, savemat
from scipy.integrate import odeint
from scipy.optimize import leastsq
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams

data = loadmat('analysis/coverage-normalized.mat')

# Here is the ode that simulates 2nd order desorption with coverage dependent adsorption energy
def myodefunc(theta, T,  alpha, kappa):
    kb = 8.617e-5;                # gas constant
    beta = 2                      # heating rate

    p0 = 2.6770;
    p1 = -5.6372;
    p2 = 4.1292;
    p3 = -4.1488;

    # this is the average adsorption energy!!!
    avg_Eads = p0 * theta**4 + p1 * theta**3 + p2 * theta**2 + p3;
    int_Eads = theta * avg_Eads

    diff_Eads = avg_Eads + (4 * p1 * theta**4
                           + 3 * p1 * theta**3
                           + 2 * p2 * theta**2)

    Ed = alpha * diff_Eads + kappa
    k = 3*10**12*np.exp(-Ed/kb/T)   # rate constant for desorption
    dthetadT = -k/beta*theta**2
    return dthetadT

def func(pars):
    alpha = pars[0]
    kappa = pars[1]

    errors = np.array([])

    for i in range(16):
        theta0 = data['theta0'][0,i]
        T = data['T'][0,i][0]
        M = data['M'][0,i][0]

        X, infodict = odeint(myodefunc, theta0,
                             T, args=(alpha, kappa),
                             full_output=True)
        theta = X.T

        error = M - (-myodefunc(theta,T,alpha, kappa))

        errors = np.append(errors, error[0])
    return errors

# initial parameters
alpha = -1.0
kappa = -1.9
```

```
x,cov_x, infodict, mesg, ier = leastsq(func, [alpha, kappa], full_output=True)

print x
```

```
[-0.46295336  0.01384663]
```

2. the figure

   We fit the average adsorption energy to an equation:

   $E_{ads}^{avg} = p_0\theta^4 + p_1\theta^3 + p_2\theta^2 + p_3$

   To get the integral adsorption energy, we multiply the average adsorption energy by θ.[2]

   $E_{ads}^{int} = \theta E^{avg}$

   Finally, the differential energy is the derivative of the integral energy. Using the chain rule this leads to:

   $E_{ads}^{diff} = E^{avg} + \theta \frac{dE^{avg}}{d\theta}$

   This leads to:

   $E_{ads}^{diff} = E_{ads}^{avg} + \theta(4p0\theta^3 + 3p1\theta^2 + 2p2\theta)$

   which finally results in:

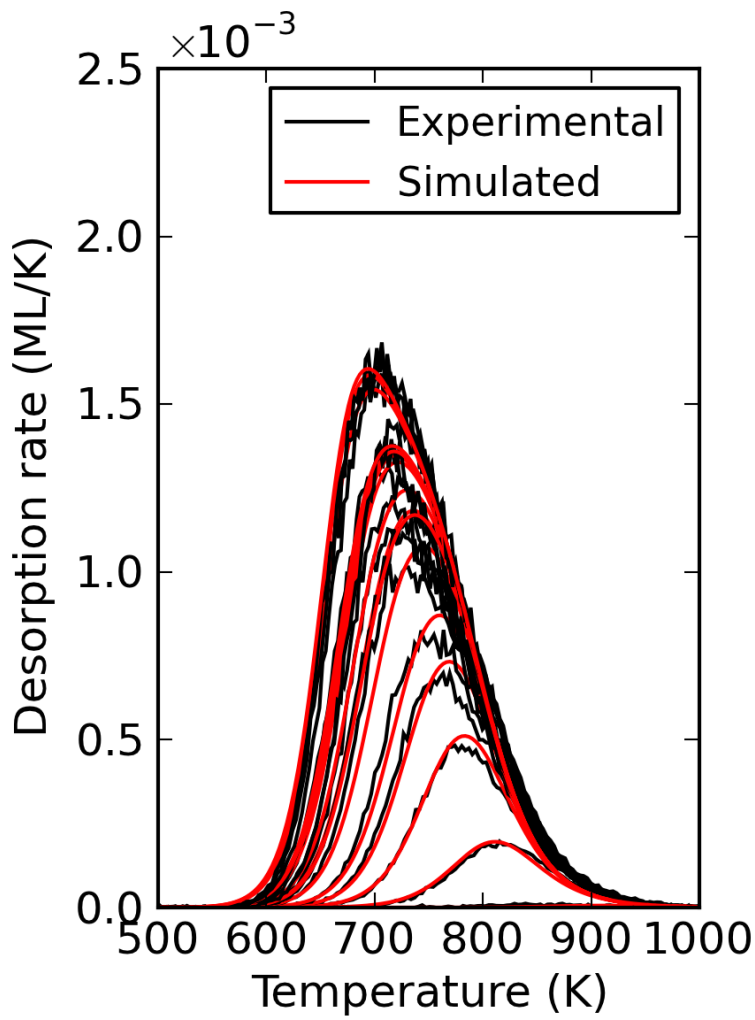
   $E_{ads}^{diff} = E_{ads}^{avg} + 4p0\theta^4 + 3p1\theta^3 + 2p2\theta^2$

```
from scipy.io import loadmat, savemat
from scipy.integrate import odeint
from scipy.optimize import leastsq
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams

rcParams['font.size'] = 12
rcParams['axes.formatter.limits'] = -3,4;
rcParams['figure.figsize'] = (3,4);
rcParams['lines.markersize'] = 7;
rcParams['figure.subplot.bottom'] = 0.12;
rcParams['figure.subplot.top'] = 0.91;
rcParams['figure.subplot.right'] = 0.9;
rcParams['figure.subplot.left'] = 0.22;
rcParams['axes.labelsize'] = 12;
rcParams['legend.fontsize'] = 11

data = loadmat('analysis/coverage-normalized.mat')

# Here is the ode that simulates 2nd order desorption with coverage dependent adsorption energy
def myodefunc(theta, T,  alpha, kappa):
```

```
    kb = 8.617e-5;                # gas constant
    beta = 2                      # heating rate

    p0 = 2.6770;
    p1 = -5.6372;
    p2 = 4.1292;
    p3 = -4.1488;

    avg_Eads = p0 * theta**4 + p1 * theta**3 + p2 * theta**2 + p3;

    int_Eads = theta * avg_Eads

    diff_Eads = avg_Eads + (4 * p1 * theta**4
                           + 3 * p1 * theta**3
                           + 2 * p2 * theta**2)

    Ed = alpha * diff_Eads + kappa

    k = 3*10**12*np.exp(-Ed/kb/T)  # rate constant for desorption
    dthetadT = -k / beta * theta**2
    return dthetadT

plt.figure()
axe = plt.subplot(111)
axe.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))

SSE = 0

for i in range(16):
    theta0 = data['theta0'][0,i]
    T = data['T'][0,i][0]
    M = data['M'][0,i][0]

    # initial parameters
    alpha, kappa = [-0.46295336,  0.01384663]
    X, infodict = odeint(myodefunc, theta0,
                         T, args=(alpha, kappa),
                         full_output=True)

    # this is the solution
    theta = X[:,0].T

    ## ### now, plot
    ## ##
    sim = -myodefunc(theta,T,alpha, kappa)
    plt.plot(T,M,'k')
    plt.plot(T, sim,'r')
    plt.ylim([0, 2.5e-3])
    plt.legend(['Experimental','Simulated'])
    SSE += np.sum((M - sim)**2)

print 'SSE = {0}'.format(SSE)

plt.xlabel('Temperature (K)')
plt.ylabel('Desorption rate (ML/K)')
for ext in ['png','eps','pdf']:
```

```
    plt.savefig('figures/fig6.{0}'.format(ext), dpi=300)
plt.show()
```

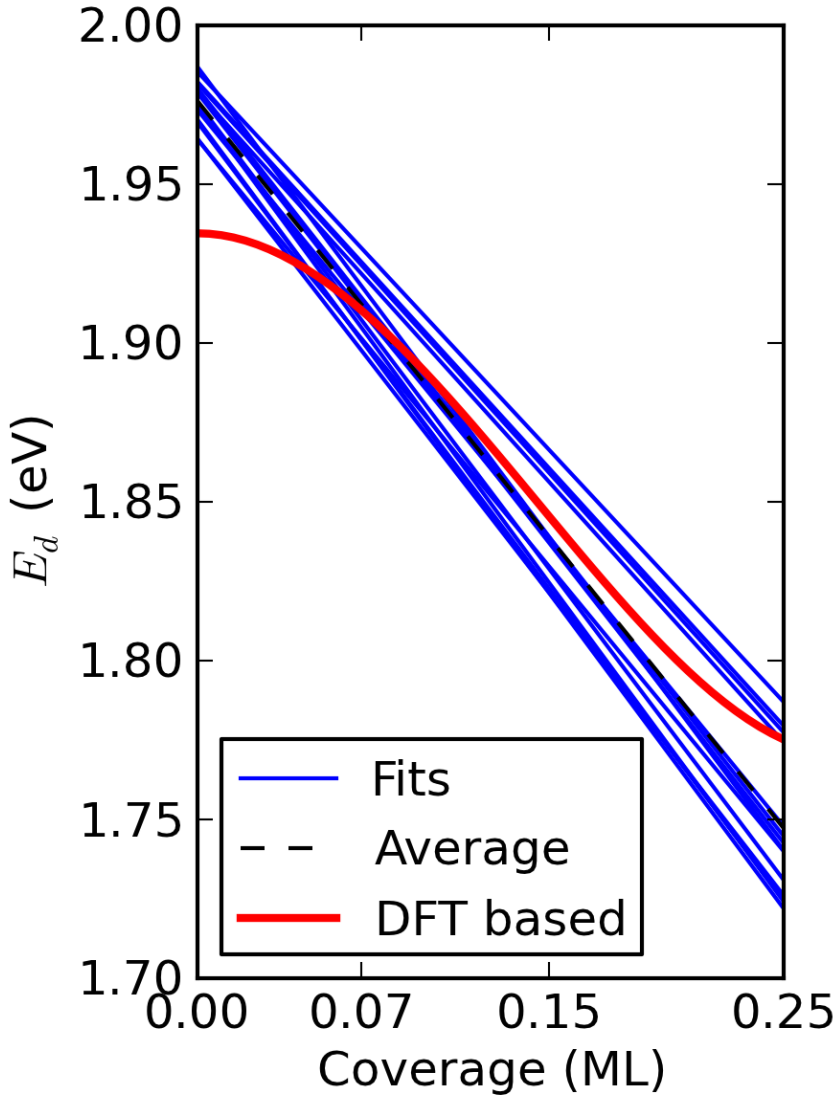
```
SSE = 6.17295283732e-06
```



1.1.7 Figure 7

```
from numpy import *
import scipy.io
from pylab import *
```

```python

rcParams['font.size'] = 12
rcParams['axes.formatter.limits'] = -3,4;
rcParams['figure.figsize'] = (3,4);
rcParams['lines.markersize'] = 7;
rcParams['figure.subplot.bottom'] = 0.12;
rcParams['figure.subplot.top'] = 0.95;
rcParams['figure.subplot.right'] = 0.95;
rcParams['figure.subplot.left'] = 0.22;
rcParams['axes.labelsize'] = 12;
rcParams['legend.fontsize']=12

avgb = average(array(data)[2:,0])
avgm = average(array(data)[2:,1])

theta = np.linspace(0, 0.25)
for b,m in data[2:]:
    h1 = plot(theta, b + m*theta, 'b-')

h1[-1].set_label('Fits')

h2 = plot(theta, avgb + avgm*theta, 'k--',label='Average')

p0 = 2.6770;
p1 = -5.6372;
p2 = 4.1292;
p3 = -4.1488;

alpha, kappa = [-0.46295336,  0.01384663]

avg_Eads = p0 * theta**4 + p1 * theta**3 + p2 * theta**2 + p3;
int_Eads = theta * avg_Eads

diff_Eads = avg_Eads + (4 * p1 * theta**4
                        + 3 * p1 * theta**3
                        + 2 * p2 * theta**2)

Ed = (alpha * diff_Eads + kappa)

plot(theta, Ed, 'r-', lw=2, label='DFT based')

legend(loc='best')
xlabel('Coverage (ML)')
ylabel('$E_d$ (eV)')
subplots_adjust(left=0.25, right=0.93)
xticks([0, 0.07, 0.15, 0.25])

for ext in ['png','eps','pdf']:
    savefig('figures/fig7.{0}'.format(ext), dpi=300)
show()
```

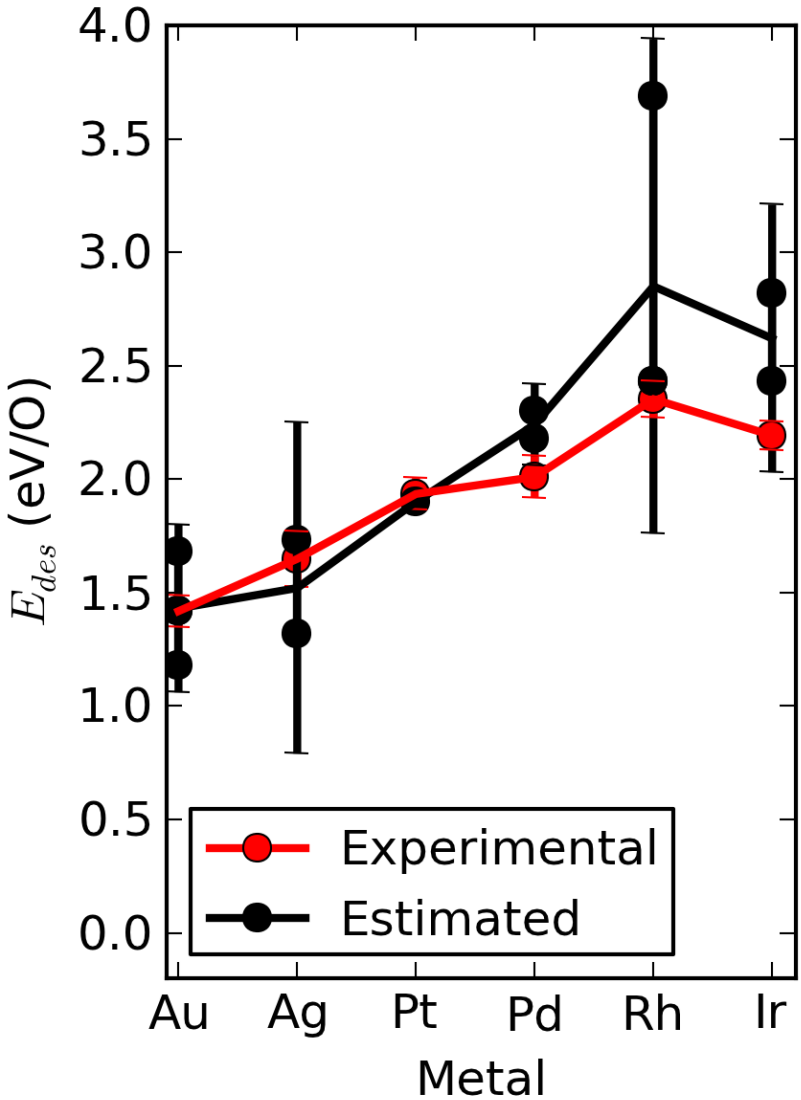


1.1.8 Figure 8

```
#!/usr/bin/env python
import sys
import matplotlib
from pylab import *
import matplotlib

rcParams['font.size'] = 12
```

```
rcParams['axes.formatter.limits'] = -3,4;
rcParams['figure.figsize'] = (3,4);
rcParams['lines.linewidth'] = 2;
rcParams['lines.markersize'] = 7;
rcParams['figure.subplot.bottom'] = 0.12;
rcParams['figure.subplot.top'] = 0.95;
rcParams['figure.subplot.right'] = 0.95;
rcParams['figure.subplot.left'] = 0.22;
rcParams['axes.labelsize'] = 12;
rcParams['legend.fontsize']=12
#DFT Simulated Barriers in Alphabetical Order these are the avg E_ads
# at theta=0, which is equivalent to the differential adsorption
# energy
dftEnergies = [-3.531, # Ag
               -3.033, # Au
               -4.706, # Ir
               -4.311, # Pd
               -4.149, # Pt
               -5.052] # Rh

alpha, kappa = [-0.46295336,  0.01384663]

dftBarriers = [alpha * energy + kappa for energy in dftEnergies]

#Experimental Barriers in alphabetical order, multiple barriers per metal
expBarriers = [[1.73, 1.32],
               [1.68, 1.42, 1.18],
               [2.82, 2.43],
               [2.3, 2.18],
               [1.9],
               [2.43, 2.43, 3.69]]

#Mean of experimental barriers in alphabetical order
expMeanBarriers = [1.52,
                   1.43,
                   2.62,
                   2.24,
                   1.9,
                   2.85]

#95%Confidence intervals for each metal in alphabetical order
simErrors = [0.1220,
             0.0708,
             0.0635,
             0.0914,
             0.0704,
             0.0797];

#95% Confidence intervals for experimental barriers in alphabetical order
expErrors = [0.73,
             0.37,
             0.59,
             0.18,
             0,
             1.09];
```

```
#Order of Metals with number equal to alphabetical order
metalOrder = [1, 0, 4, 3, 5, 2];

#Metals in alphabetical order
metalList = ['Ag','Au','Ir','Pd','Pt','Rh']

metalNameList = [];
simPlotBarrier = [];
expPlotBarrier = [];

for l in range(len(metalOrder)):

    i = metalOrder[l];
    j = [];
    for k in range(len(expBarriers[i])):
        j.append(l);

    plot(l,dftBarriers[i],color='red',marker='o')
    plot(j,expBarriers[i],color='black',marker='o');

    errorbar(l,expMeanBarriers[i],yerr=expErrors[i],color='black');
    errorbar(l,dftBarriers[i],yerr=simErrors[i],color='red');

    metalNameList.append(metalList[i]);

    simPlotBarrier.append(dftBarriers[i]);
    expPlotBarrier.append(expMeanBarriers[i]);

expPlot = plot(expPlotBarrier,color='black', label='Experimental');
dftPlot = plot(simPlotBarrier,color='red', label='Estimated');

ylim(ymin = -0.2)
xlim(xmin = -.1,xmax=5.2)
xticks(arange(len(metalList)),metalNameList)

ylabel(r'$E_{des}$ (eV/O)');
xlabel(r'Metal');

legend(['Experimental','Estimated'], loc='lower left')
for ext in ['png','eps','pdf']:
    savefig('figures/fig8.{0}'.format(ext), dpi=300)
show()
```



1.2 Comparison of our results to recent work by Karp

In the review article by Brown,[1] they suggest that $E_d = q_{st} - 0.5RT$ when all properties are temperature independent. In that equation q_{st} is the isosteric heat of adsorption. Karp et al.[3] correct their data by the factor $0.5RT$, and here we check on the magnitude of that correction. The value of T is ambiguous in a TPD, so we chose a low coverage limit of 850 K based on

our experiments.

kb = 0.008314 kJ/mol/K = 8.617e-5 eV/K 1 eV = 96.4853 kJ/mol

```
k = 8.617e-5
T = 850
print '{0:1.3f} eV'.format(0.5*k*T)
print '{0:1.3f} kJ/mol'.format(0.5*0.008314*T)
```

```
0.037 eV
3.533 kJ/mol
```

The correction is small in magnitude. We estimate a low coverage desorption barrier of 1.925 eV in Figure 7. So the correction is only about 2% of the estimated desorption barrier.

From our work (Figure 7) we estimate the following desorption barriers and here convert them to kJ/mol without the $0.5RT$ correction factor.

```
print 'Zero coverage desorption barrier = {0} kJ/mol'.format(1.925*96.4853)
print '0.25 ML coverage desorption barrier = {0} kJ/mol'.format(1.74*96.4853)
```

```
Zero coverage desorption barrier = 185.7342025 kJ/mol
0.25 ML coverage desorption barrier = 167.884422 kJ/mol
```

According to Figure 1 in,[3] the differential adsorption energy is -217 kJ/mol in the limit of zero coverage, and -179.25 kJ/mol at 0.25 ML. We underestimate these adsorption energies.

```
print '%error at 0ML = {0}'.format((-186. - (-217.))/(-217.)*100.)
print '%error at 0.25ML = {0}'.format((-167. - (-217.))/(-217.)*100.)
```

```
%error at 0ML = -14.2857142857
%error at 0.25ML = -23.0414746544
```

1.3 Leading edge analysis of the TPD spectra of oxygen on Pt(111)

Our aim here is to do a leading edge analysis to estimate a pre-exponential factor and estimate coverage dependence. We follow the analysis at http://www.chemie.fu-berlin.de/~pcprakt/tds.pdf.

We plot $\ln(r_{des})$ vs. $1/T$ for the leading edge, and we should get a line according to:

$\ln(r_{des}) = -\frac{\Delta E_{des}}{RT} + \ln(\nu_n) + n \ln \theta$.

From this equation we have: $\Delta E_{des} = -R * slope$

and $\ln(\nu_n) + n \ln \theta = intercept$, so $\nu_n = \exp(intercept - n \ln(\theta))$.

```
from scipy.io import loadmat, savemat
from scipy.integrate import trapz, cumtrapz
from scipy.optimize import curve_fit
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats.distributions import  t

data = loadmat('analysis/coverage-normalized.mat')

theta0s, afactors, barriers = [],[],[]
aerr, berr = [],[] # store 95% intervals

for i in range(2,16):

    T = data['T'][0,i][0]
    M = data['M'][0,i][0]
    theta0 = trapz(M,T) # initial coverage

    # compute coverage as a function of coverage
    theta = np.zeros(T.shape)
    for j in range(len(T)):
        theta[j] = theta0 - trapz(M[0:j], T[0:j])

    # now find region where coverage has changed less than 5%, and the
    # desorption rate is sufficiently large to not be noisy (rdes > 1e-10
    ind = (theta > 0.95*theta0) & (np.log(M) > -10)

    T1 = T[ind]
    lnr1 = np.log(M[ind])

    # fit a line to get initial guesses
    (m,b) = np.polyfit(1/T1, lnr1, 1)

    kb = 8.617e-5

    # now use the nonlinear fit to get confidence intervals
    def func(x, m, b):
        return m*x + b

    pars, pcov = curve_fit(func, 1/T1, lnr1, p0=[m,b])
    alpha = 0.05 # 95% confidence interval
    n = len(T1)    # number of data points
    p = len(pars) # number of parameters
    dof = max(0, n-p) # number of degrees of freedom
    tval = t.ppf(1.0-alpha/2., dof) # student-t value for the dof and
                                    # confidence level

    # barrier
```

```
    pm, pb = pars

    sigmapm = pcov[0,0]**0.5
    print '   Spectra {0}: theta0 = {1:1.3f}'.format(i, theta0)
    print '   The desorption barrier is {2:1.3f} [{0:1.3f} {1:1.3f}]'.format(-kb*(pm + sigmapm*tval),
                                                                            -kb*(pm - sigmapm*tval),
                                                                            -kb*pm)

    berr.append(-kb*sigmapm*tval)

    # note we do not consider the uncertainty in theta0 in this confidence interval
    sigmapb = pcov[1,1]**0.5

    a = np.exp(pb - 2*np.log(theta0))
    aupper = np.exp((pb + sigmapb*tval) - 2*np.log(theta0))
    alower = np.exp((pb - sigmapb*tval) - 2*np.log(theta0))

    print 'The preexponential factor is {2:1.3e} [{0:1.3e} {1:1.3e}]'.format(alower,
                                                                              aupper,
                                                                              a)
    aerr.append((a-alower, aupper-a))

    plt.plot(1/T1, lnr1, ' o', label='spectra{0}'.format(i))
    plt.plot(1/T1, func(1/T1, pm, pb))
    print

    theta0s.append(theta0)
    afactors.append(a)
    barriers.append(-kb*pm)

aerr = np.array(aerr).T

plt.xlabel('1/T (1/K)')
plt.ylabel('$\ln(r_{des})$')
plt.legend(loc='best')
plt.xlim([0.0012, 0.0017])
plt.savefig('analysis/leading-edge-analysis.png', dpi=300)

plt.figure()
plt.semilogy(theta0s, afactors,'bo')
plt.semilogy(theta0s, aerr[0,:],'*')
plt.semilogy(theta0s, aerr[1,:],'*')
plt.xlabel('Initial coverage (ML)')
plt.ylabel('Preexponential factor')
plt.savefig('analysis/LEA-preexponentials.png', dpi=300)

plt.figure()
plt.errorbar(theta0s, barriers, berr, fmt='bo')

plt.xlabel('Initial coverage (ML)')
plt.ylabel('Desorption barrier (eV)')
plt.savefig('analysis/LEA-barriers.png', dpi=300)
```

```
Spectra 2: theta0 = 0.056
The desorption barrier is 1.905 [1.640 2.170]
```

```
The preexponential factor is 1.064e+12 [1.359e+10 8.328e+13]

   Spectra 3: theta0 = 0.083
   The desorption barrier is 1.884 [1.769 1.998]
The preexponential factor is 1.174e+12 [1.686e+11 8.173e+12]

   Spectra 4: theta0 = 0.102
   The desorption barrier is 1.911 [1.809 2.014]
The preexponential factor is 2.418e+12 [4.128e+11 1.416e+13]

   Spectra 5: theta0 = 0.136
   The desorption barrier is 1.847 [1.764 1.930]
The preexponential factor is 1.348e+12 [3.095e+11 5.874e+12]

   Spectra 6: theta0 = 0.156
   The desorption barrier is 1.840 [1.760 1.919]
The preexponential factor is 1.631e+12 [3.891e+11 6.836e+12]

   Spectra 7: theta0 = 0.169
   The desorption barrier is 1.885 [1.802 1.968]
The preexponential factor is 4.315e+12 [9.538e+11 1.952e+13]

   Spectra 8: theta0 = 0.187
   The desorption barrier is 1.858 [1.735 1.982]
The preexponential factor is 3.026e+12 [3.165e+11 2.893e+13]

   Spectra 9: theta0 = 0.191
   The desorption barrier is 1.861 [1.770 1.951]
The preexponential factor is 3.370e+12 [6.361e+11 1.785e+13]

   Spectra 10: theta0 = 0.193
   The desorption barrier is 1.873 [1.772 1.975]
The preexponential factor is 4.439e+12 [6.843e+11 2.880e+13]

   Spectra 11: theta0 = 0.154
   The desorption barrier is 1.804 [1.723 1.885]
The preexponential factor is 6.769e+11 [1.594e+11 2.874e+12]

   Spectra 12: theta0 = 0.197
   The desorption barrier is 1.875 [1.788 1.963]
```

```
The preexponential factor is 2.614e+12 [5.333e+11 1.281e+13]

   Spectra 13: theta0 = 0.228
   The desorption barrier is 1.858 [1.809 1.907]
The preexponential factor is 3.330e+12 [1.334e+12 8.317e+12]

   Spectra 14: theta0 = 0.235
   The desorption barrier is 1.904 [1.815 1.993]
The preexponential factor is 9.156e+12 [1.745e+12 4.803e+13]

   Spectra 15: theta0 = 0.238
   The desorption barrier is 1.875 [1.797 1.952]
The preexponential factor is 5.790e+12 [1.367e+12 2.453e+13]
```

Leading edge analysis. The lines are the best fits to the data.

./analysis/leading-edge-analysis.png

Pre-exponential factors with 95%% confidence interval indicated by the

stars. The errorbars are not symmetric due to the nonlinear transform of the confidence interval on the intercept to a pre-exponential factor.

./analysis/LEA-preexponentials.png

Coverage dependent desorption barriers with 95% confidence intervals.

./analysis/LEA-barriers.png

The main takeaway points are that one cannot say with 95% certainty there is coverage dependence based on the leading edge analysis. The desorption barrier is about 1.85 eV. The pre-exponential factor is in the range of 1e12 to 1e13 over the whole coverage range.

1.4 Preparation of the TPD data

1.4.1 Convert the raw data in Excel sheets to tables for analysis here.

Here we convert the Excel sheets to org-tables that store the data in this file. This will enable others to reproduce the results here without access to the Excel files. It is not necessary to include the data in this form; the Excel sheets could also be included as supplementary data. Including them here makes this document completely portable. This is the only script a reader cannot run without access to the Excel sheets.

```python
import xlrd

exposures = []

for i in range(1,21):
    wb = xlrd.open_workbook('xls/tpd{0}.xls'.format(i))
    sh = wb.sheet_by_name(u'Sheet1')

    # there is header information in row 0
    T = sh.col_values(0,start_rowx=1)   #temperature
    I = sh.col_values(1,start_rowx=1)   #intensity

    # this is in the header of the second column
    # exposures are in Langmuirs
    exposure = sh.cell(rowx=0,colx=1).value

    exposures.append(exposure)

    # the goal is to write out a table
    print '**** tpd{0} data'.format(i)
    print '#+ATTR_LaTeX: longtable'
    print '#+tblname: tpd{0}'.format(i)
    print '| Temperature (K)| M.S. intensity (arb. units)|'
    print '|-'
    for t,i in zip(T,I):
        print '|{0}|{1}|'.format(t,i)
    print

# now we need to make a table of exposures
print '**** exposures'
print '#+tblname: exposures'
print '|spectrum number | exposure (L)'
print '|-'
for i,e in enumerate(exposures):
    print '|{0}|{1}|'.format(i,e)
print
```

1.4.2 Convert all the tabular data to a single data file for convenient for analysis

Now we convert the tables in the previous sections to a binary data format that is more convenient for subsequent analysis. This script uses the tables in the previous section as a data source.

```python
import matplotlib.pyplot as plt
from scipy.io import savemat

exposures = [float(x[1]) for x in exposures]
AllT, AllI = [], [] #lists to save all spectra in

# these are the temperatures for each spectrum
AllT.append([float(x[0]) for x in tpd1])
AllT.append([float(x[0]) for x in tpd2])
```

```
AllT.append([float(x[0]) for x in tpd3])
AllT.append([float(x[0]) for x in tpd4])
AllT.append([float(x[0]) for x in tpd5])
AllT.append([float(x[0]) for x in tpd6])
AllT.append([float(x[0]) for x in tpd7])
AllT.append([float(x[0]) for x in tpd8])
AllT.append([float(x[0]) for x in tpd9])
AllT.append([float(x[0]) for x in tpd10])
AllT.append([float(x[0]) for x in tpd11])
AllT.append([float(x[0]) for x in tpd12])
AllT.append([float(x[0]) for x in tpd13])
AllT.append([float(x[0]) for x in tpd14])
AllT.append([float(x[0]) for x in tpd15])
AllT.append([float(x[0]) for x in tpd16])
AllT.append([float(x[0]) for x in tpd17])
AllT.append([float(x[0]) for x in tpd18])
AllT.append([float(x[0]) for x in tpd19])
AllT.append([float(x[0]) for x in tpd20])

# these are the MS intensities for each spectrum
AllI.append([float(x[1]) for x in tpd1])
AllI.append([float(x[1]) for x in tpd2])
AllI.append([float(x[1]) for x in tpd3])
AllI.append([float(x[1]) for x in tpd4])
AllI.append([float(x[1]) for x in tpd5])
AllI.append([float(x[1]) for x in tpd6])
AllI.append([float(x[1]) for x in tpd7])
AllI.append([float(x[1]) for x in tpd8])
AllI.append([float(x[1]) for x in tpd9])
AllI.append([float(x[1]) for x in tpd10])
AllI.append([float(x[1]) for x in tpd11])
AllI.append([float(x[1]) for x in tpd12])
AllI.append([float(x[1]) for x in tpd13])
AllI.append([float(x[1]) for x in tpd14])
AllI.append([float(x[1]) for x in tpd15])
AllI.append([float(x[1]) for x in tpd16])
AllI.append([float(x[1]) for x in tpd17])
AllI.append([float(x[1]) for x in tpd18])
AllI.append([float(x[1]) for x in tpd19])
AllI.append([float(x[1]) for x in tpd20])

# Make some figures of the raw data
for i in range(20):
    plt.plot(AllT[i], AllI[i], label='{0} L'.format(exposures[i]))

plt.xlabel('Temperature (K)')
plt.ylabel('M.S. intensity (arb. units)')
plt.legend(ncol=3, loc='best')

plt.savefig('analysis/raw-data.png', dpi=300)

plt.xlim([400,1000])
plt.ylim([0, 3e-7])

plt.savefig('analysis/raw-data-xlim.png', dpi=300)
plt.show()
```

```
# Finally, this is the data file we save for all subsequent analysis
savemat('analysis/raw-data.mat',{'T':AllT, 'M':AllI, 'exposures':exposures}, oned_as='row')
```

./analysis/raw-data.png

Figure 1: All of the data over the whole range.

1.4.3 Normalize the baseline of the raw data and narrow T range

The baselines of each spectrum is not at zero. Here we average the value of each spectrum between 400-500K and subtract that from the spectrum to make the baselines equal zero. Finally, we cut the spectra to be between 500-1000 K

```
from scipy.io import loadmat, savemat
import numpy as np
```

./analysis/raw-data-xlim.png

Figure 2: The data narrowed over the range of temperatures corresponding to chemisorption.

```
import matplotlib.pyplot as plt

data = loadmat('analysis/raw-data.mat')

newT, newM =  [], [] #to store baseline normalized spectra

for t, m in zip(data['T'], data['M']):
    ind1 = (t > 400) & (t < 500)
    baseline = np.average(m[ind1])

    ind2 = (t >= 500) & (t <= 1000) # narrow data set to this T range

    newm = m - baseline # subtract baseline

    newM.append(newm[ind2]) # baseline corrected
    newT.append(t[ind2])
```

```
    plt.plot(t[ind2], newm[ind2])

plt.xlabel('Temperature (K)')
plt.ylabel('M.S. intensity (arb. units)')

plt.ylim([0, 1.5e-7])
plt.savefig('analysis/baseline-normalized-data.png', dpi=300)
plt.show()

savemat('analysis/baseline-normalized.mat',{'T':np.array(newT),
                                 'M':newM,
                                 'exposures':data['exposures']},
                                 oned_as='row')
```

./analysis/baseline-normalized-data.png

Figure 3: Baseline normalized data.

1.4.4 Normalize to saturation coverage

Now, we normalize the spectra so that the integrated area is equal to the initial coverage for each spectrum. First, we plot the area as a function of exposure.

```
from scipy.io import loadmat, savemat
import numpy as np
import matplotlib.pyplot as plt

data = loadmat('analysis/baseline-normalized.mat')

exposures, areas = [], []

# data['T'].shape = (1,20) data is all in rows
# so data['T'][0,0] is the temperature data for the first spectrum
for i in range(data['T'].shape[1]):

    t = data['T'][0,i]
    m = data['M'][0,i]
    area = np.trapz(m, t)
    areas.append(area)

plt.semilogx(data['exposures'][0], areas, 'bo ')
plt.semilogx(data['exposures'][0], 1.5e-5*np.ones(data['exposures'][0].shape), 'k-')
plt.xlabel('Exposure (Langmuir)')
plt.ylabel('Integrated area (arb. units)')
plt.subplots_adjust(left=0.18)
plt.savefig('analysis/integrated-area.png', dpi=300)

plt.show()
```

We choose an area of 1.5e-5 as the area representing a saturated surface. We choose a saturation coverage of 0.25 ML for this area based on the analysis presented in the manuscript. Next we normalize the spectra so the integral will give the initial coverage.

```
from scipy.io import loadmat, savemat
import numpy as np
import matplotlib.pyplot as plt

data = loadmat('analysis/baseline-normalized.mat')

normT = []
normM    = [] # normalized spectra
theta_0 = [] # vector of initial coverages

theta_sat = 0.25 #ML
area_sat  = 1.5e-5 # area corresponding to saturated coverage

#Now, we will only examine the spectra up to a 5L exposure because we believe after that surface oxides have likely
```

./analysis/integrated-area.png

Figure 4: Integrated peak area as a function of exposure.

```
for i in range(16):
    t = data['T'][0,i][0]
    m = data['M'][0,i][0]

    area = np.trapz(m, t)

    coverage0 = area/area_sat*theta_sat
    theta_0.append(coverage0) # this is the initial coverage

    norm_m = m/area_sat*theta_sat

    normT.append(t)
    normM.append(norm_m)

    plt.plot(t, norm_m)

savemat('analysis/coverage-normalized.mat', {'T':normT,                # temperatures
```

```
                                    'M':normM,                      # normalized intensity
                                    'exposures':data['exposures'],  # exposures
                                    'theta0':theta_0},
                                    oned_as='row')              # initial coverages

plt.xlabel('Temperature (K)')
plt.ylabel('Integrated area (ML/K)')
plt.xlim([500, 1000])
plt.ylim([0,5e-3])
plt.savefig('analysis/coverage-normalized.png', dpi=300)
plt.show()
```

./analysis/coverage-normalized.png

Figure 5: Fully normalized spectra.

1. Double check the normalization

```python
from scipy.io import loadmat, savemat
import numpy as np
import matplotlib.pyplot as plt

data = loadmat('analysis/coverage-normalized.mat')

for i in range(data['T'].shape[1]):

    t = data['T'][0,i]
    m = data['M'][0,i]
    e = data['exposures'][0,i]
    area = np.trapz(m, t)[0]

    print 'spectrum {0:3d}: exposure = {1:5.2f} L  coverage = {2:1.3f}'.format( i,  e, area)
```

```
spectrum   0: exposure =  0.25 L  coverage = 0.001
spectrum   1: exposure =  0.50 L  coverage = 0.022
spectrum   2: exposure =  0.75 L  coverage = 0.056
spectrum   3: exposure =  1.00 L  coverage = 0.083
spectrum   4: exposure =  1.25 L  coverage = 0.102
spectrum   5: exposure =  1.50 L  coverage = 0.136
spectrum   6: exposure =  1.75 L  coverage = 0.156
spectrum   7: exposure =  2.00 L  coverage = 0.169
spectrum   8: exposure =  2.25 L  coverage = 0.187
spectrum   9: exposure =  2.50 L  coverage = 0.191
spectrum  10: exposure =  2.75 L  coverage = 0.193
spectrum  11: exposure =  3.00 L  coverage = 0.154
spectrum  12: exposure =  3.50 L  coverage = 0.197
spectrum  13: exposure =  4.00 L  coverage = 0.228
spectrum  14: exposure =  4.50 L  coverage = 0.235
spectrum  15: exposure =  5.00 L  coverage = 0.238
```

The coverage varies from approximately 0 ML to about 0.24 ML, which is the range it should vary over.

1.4.5 Show that a constant desorption barrier does not describe the data

```python
from scipy.io import loadmat, savemat
from scipy.integrate import odeint
from scipy.optimize import leastsq
import numpy as np
import matplotlib.pyplot as plt

data = loadmat('analysis/coverage-normalized.mat')
```

```python

# Here is the ode that simulates 2nd order desorption
def myodefunc(theta, Temperature,  Ed):
    kb = 8.617e-5;                # gas constant
    beta = 2                      # heating rate

    k = 10**12*np.exp(-Ed/kb/Temperature) # rate constant for desorption
    dthetadT = -k/beta*theta**2
    return dthetadT

for i in range(16):

    theta0 = data['theta0'][0,i]
    T = data['T'][0,i][0]
    M = data['M'][0,i][0]

    def func(Ed):
        '''returns error between expt data and simulated data'''
        X, infodict = odeint(myodefunc,
                             theta0, T,
                             args=(Ed,),
                             full_output=True)
        theta = X.T
        error = M - (-myodefunc(theta,T,Ed))
        return error[0]


    # initial parameters
    Ed = 1.95

    x, cov_x, infodict, mesg, ier = leastsq(func, Ed, full_output=True)
    X, infodict = odeint(myodefunc, theta0,
                         T, args=(x[0],),
                         full_output=True)

     # this is the solution
    theta = X[:,0].T

    ## ### now, plot
    plt.plot(T,M)
    plt.plot(T,-myodefunc(theta,T,x[0]),'r')
    plt.xlabel('Temperature (K)')
    plt.ylabel('Desorptionrate (ML/K)')
    plt.title('Best fit E_d = {0:1.3f} eV'.format(x[0]))
    plt.savefig('analysis/tpd-constant-ed-{0}.png'.format(i), dpi=300)
    print '**** Spectrum {0}'.format(i)
    print '[[./analysis/tpd-constant-ed-{0}.png]]'.format(i)
```

1. Spectrum 0 ./analysis/tpd-constant-ed-0.png

2. Spectrum 1 ./analysis/tpd-constant-ed-1.png

3. Spectrum 2 ./analysis/tpd-constant-ed-2.png

4. Spectrum 3

./analysis/tpd-constant-ed-3.png

5. Spectrum 4 ./analysis/tpd-constant-ed-4.png

6. Spectrum 5 ./analysis/tpd-constant-ed-5.png

7. Spectrum 6 ./analysis/tpd-constant-ed-6.png

8. Spectrum 7 ./analysis/tpd-constant-ed-7.png

9. Spectrum 8 ./analysis/tpd-constant-ed-8.png

./analysis/tpd-constant-ed-9.png

10. Spectrum 9

11. Spectrum 10

./analysis/tpd-constant-ed-10.png

./analysis/tpd-constant-ed-11.png

12. Spectrum 11

13. Spectrum 12

./analysis/tpd-constant-ed-12.png

14. Spectrum 13

./analysis/tpd-constant-ed-13.png

15. Spectrum 14 ./analysis/tpd-constant-ed-14.png

./analysis/tpd-constant-ed-15.png

16. Spectrum 15

1.4.6 Fitting a linear coverage dependence to the data

Our goal is to fit a linear coverage dependent model to the TPD data. We will find the parameters E_{d0} and α in $E_d = E_{d0} + \alpha\theta$ that best fit all the equations. We will do that by integrating the TPD mass balance equations for each initial coverage, subtracting the simulated results from the experimental data, and minimizing the summed squared error.

```
from scipy.io import loadmat, savemat
from scipy.integrate import odeint
from scipy.optimize import leastsq
import numpy as np
import matplotlib.pyplot as plt

data = loadmat('analysis/coverage-normalized.mat')

# Here is the ode that simulates 2nd order desorption
def myodefunc(theta, T,  Ed0, alpha):
    kb = 8.617e-5;                # gas constant
    beta = 2                      # heating rate
```

```
    Ed = Ed0 + alpha*theta        # coverage dependent desorption barrier
    k = 3*10**12*np.exp(-Ed/kb/T)   # rate constant for desorption
    dthetadT = -k/beta*theta**2
    return dthetadT

s = []
# loop over the spectra
for i in range(16):
    theta0 = data['theta0'][0,i]
    T = data['T'][0,i][0]
    M = data['M'][0,i][0]

    def func(params):
        Ed0, alpha = params

        X, infodict = odeint(myodefunc, theta0,
                             T, args=(Ed0, alpha),
                             full_output=True)
        theta = X.T

        error = M - (-myodefunc(theta,T,Ed0,alpha))
        return error[0]

    # initial parameters
    Ed0 = 1.95
    alpha = -0.5

    x,cov_x, infodict, mesg, ier = leastsq(func, [Ed0, alpha], full_output=True)

    s.append(x)

    X, infodict = odeint(myodefunc, theta0,
                         T, args=(x[0], x[1]),
                         full_output=True)

    # this is the solution
    theta = X[:,0].T

    ## ### now, plot
    ## ##
    plt.figure()
    plt.plot(T,M)
    plt.plot(T,-myodefunc(theta,T,x[0],x[1]),'r')
    plt.title('$E_d$ = {0:1.3f} + {1:1.3f} $\\theta$'.format(x[0], x[1]))
    plt.xlabel('Temperature (K)')
    plt.ylabel('Desorption rate (ML/K)')
    plt.savefig('analysis/tpd-linear-ed-{0}'.format(i), dpi=300)
    print '**** Spectrum {0}'.format(i)
    print '[[./analysis/tpd-linear-ed-{0}.png]]'.format(i)

print '**** Fitted lines'
print '#+tblname: linear-fits'
print '| intercept | slope |'
for b,m in s:
    print '| {0} | {1} |'.format(b,m)
```

1. Spectrum 0 ./analysis/tpd-linear-ed-0.png

2. Spectrum 1 ./analysis/tpd-linear-ed-1.png

3. Spectrum 2 ./analysis/tpd-linear-ed-2.png

4. Spectrum 3 ./analysis/tpd-linear-ed-3.png

5. Spectrum 4 ./analysis/tpd-linear-ed-4.png

./analysis/tpd-linear-ed-5.png

6. Spectrum 5

./analysis/tpd-linear-ed-6.png

7. Spectrum 6

./analysis/tpd-linear-ed-7.png

8. Spectrum 7

9. Spectrum 8 ./analysis/tpd-linear-ed-8.png

./analysis/tpd-linear-ed-9.png

10. Spectrum 9

./analysis/tpd-linear-ed-10.png

11. Spectrum 10

./analysis/tpd-linear-ed-11.png

12. Spectrum 11

13. Spectrum 12 ./analysis/tpd-linear-ed-12.png

./analysis/tpd-linear-ed-13.png

14. Spectrum 13

15. Spectrum 14 ./analysis/tpd-linear-ed-14.png

./analysis/tpd-linear-ed-15.png

16. Spectrum 15
17. Fitted lines

intercept	slope
1.72991910987	110.599760148
1.95634872029	-0.584506977
1.96413691517	-0.896668320418
1.96431452738	-0.953072780296
1.96947364817	-0.97926930681
1.97032568924	-0.99352866073
1.9750469438	-1.00371851308
1.97574568707	-0.979287916309
1.97877725402	-0.944815102167
1.98014787754	-0.941725556664
1.97334933065	-0.901988056372
1.98695567948	-0.986765670065
1.98538770635	-0.793602804862
1.98222587397	-0.81111165715
1.98168919413	-0.817273534646
1.97908241437	-0.818973479364

1.4.7 Show no single linear fit works

```
from scipy.io import loadmat, savemat
from scipy.integrate import odeint
from scipy.optimize import leastsq
import numpy as np
import matplotlib.pyplot as plt

data = loadmat('analysis/coverage-normalized.mat')

# Here is the ode that simulates 2nd order desorption
def myodefunc(theta, T,  Ed0, alpha):
    kb = 8.617e-5;                # gas constant
    beta = 2                      # heating rate
    Ed = Ed0 + alpha*theta        # coverage dependent desorption barrier
    k = 3*10**12*np.exp(-Ed/kb/T)   # rate constant for desorption
    dthetadT = -k/beta*theta**2
    return dthetadT

low_SSE = 0
# a low coverage linear dependence
for i in range(16):
    theta0 = data['theta0'][0,i]
    T = data['T'][0,i][0]
    M = data['M'][0,i][0]

    # initial parameters # spectrum 2
    Ed0 = 1.956
    alpha = -0.584

```

```
    X, infodict = odeint(myodefunc, theta0,
                         T, args=(Ed0, alpha),
                         full_output=True)

    # this is the solution
    theta = X[:,0].T

    ## ### now, plot
    ## ##
    sim = -myodefunc(theta,T,Ed0, alpha)
    plt.plot(T, M)
    plt.plot(T, sim,'r')
    low_SSE += np.sum((M - sim)**2)


print 'LOW SSE = {0}'.format(low_SSE)

plt.xlabel('Temperature (K)')
plt.ylabel('Desorption rate (ML/K)')
plt.title('Low coverage fit')
plt.savefig('analysis/tpd-linear-low-coverage.png', dpi=300)

# a high coverage line # spectrum 15
high_SSE = 0
plt.figure()
for i in range(16):
    theta0 = data['theta0'][0,i]
    T = data['T'][0,i][0]
    M = data['M'][0,i][0]

    # initial parameters
    Ed0 = 1.979
    alpha = -0.819

    X, infodict = odeint(myodefunc, theta0,
                         T, args=(Ed0, alpha),
                         full_output=True)

    # this is the solution
    theta = X[:,0].T

    ## ### now, plot
    sim = -myodefunc(theta,T,Ed0, alpha)
    plt.plot(T, M)
    plt.plot(T, sim,'r')
    high_SSE += np.sum((M - sim)**2)

print 'HIGH SSE = {0}'.format(high_SSE)
plt.xlabel('Temperature (K)')
plt.ylabel('Desorption rate (ML/K)')
plt.title('High coverage fit')
plt.savefig('analysis/tpd-linear-high-coverage.png', dpi=300)
plt.show()
```

```
LOW SSE = 2.08183838751e-05
```

```
HIGH SSE = 6.12873753024e-06
```

./analysis/tpd-linear-low-coverage.png

Figure 6: Low coverage fitted linear dependence. The low coverage peaks are excellent, but the high coverage peaks are not as good.

We conclude that a nonlinear model is needed to fully capture the high and low coverage dependence.

1.5 Coverage dependent adsorption data from DFT calculations

All of this data is from[4] and details of the calculations can be found there.

./analysis/tpd-linear-high-coverage.png

Figure 7: High coverage fitted linear dependence. The high coverage peaks are fit pretty well, but the low coverage peaks are not fit quite as well, but the fit is not too bad.

1.5.1 Rh-O.out

coverage (ML)	ΔH_{ads} (eV/O)
1.0	-4.17854197081
0.5	-4.75799542429
0.333333333333	-4.92718837611
0.666666666667	-4.4789865732
0.333333333333	-4.77518229352
0.666666666667	-4.5288433656
0.25	-4.99086339781
0.75	-4.46730135777
0.25	-4.97292683148
0.5	-4.75524049382
0.75	-4.42405762672
0.25	-4.77559540919
0.5	-4.55195005977
0.75	-4.4335769363
0.2	-4.99296354842
0.4	-4.83677014856
0.4	-4.74275452874
0.6	-4.57663993907
0.6	-4.56568308015
0.8	-4.37255249662

```
import numpy as np
import matplotlib.pyplot as plt

data = np.array(data)

thetas = data[:,0]
hads = data[:,1]

A = np.vstack([thetas**4, thetas**3, thetas**2, np.ones(len(thetas))]).T
pars = np.linalg.lstsq(A,hads)[0]
print 'fitted pars = ',pars

plt.plot(thetas, hads, 'bo ', label='DFT data')

ft = np.linspace(0,1)
A = np.vstack([ft**4, ft**3, ft**2, np.ones(len(ft))]).T
print('At theta=0 E_ads={0}'.format(np.dot(A,pars)[0]))
plt.plot(ft, np.dot(A,pars), label='Fit')

plt.legend(loc='best')
plt.xlabel('Coverage (ML)')
plt.ylabel('$\Delta H_{ads}$ (eV/O)')
plt.title(pars)
plt.savefig('analysis/Rh-O-fit.png', dpi=300)
```

```
plt.show()
```

```
fitted pars =  [ 1.18430842 -2.98973673  2.6784023  -5.05184062]
At theta=0 E_ads=-5.05184062207
```

./analysis/Rh-O-fit.png

1.5.2 Ir-O.out

coverage (ML)	ΔH_{ads} (eV/O)
1.0	-3.87890916411
0.5	-4.49132386092
0.333333333333	-4.61250753253
0.666666666667	-4.2995072858
0.333333333333	-4.50708287467
0.666666666667	-4.28888765987
0.25	-4.66544464194
0.75	-4.20776610907
0.25	-4.63559153358
0.5	-4.49074331771
0.75	-4.19619127713
0.25	-4.53521471603
0.5	-4.33154964592
0.75	-4.18603137392
0.2	-4.6778774751
0.4	-4.56364149625
0.4	-4.49048241279
0.6	-4.33314244475
0.6	-4.36337676309
0.8	-4.14301050253

```
import numpy as np
import matplotlib.pyplot as plt

data = np.array(data)

thetas = data[:,0]
hads = data[:,1]

A = np.vstack([thetas**4, thetas**3, thetas**2, np.ones(len(thetas))]).T
pars = np.linalg.lstsq(A,hads)[0]
print 'fitted pars = ',pars

plt.plot(thetas, hads, 'bo ', label='DFT data')

ft = np.linspace(0,1)
A = np.vstack([ft**4, ft**3, ft**2, np.ones(len(ft))]).T
plt.plot(ft, np.dot(A,pars), label='Fit')

plt.legend(loc='best')
plt.xlabel('Coverage (ML)')
plt.ylabel('$\Delta H_{ads}$ (eV/O)')
plt.title(pars)
plt.savefig('analysis/Ir-O-fit.png', dpi=300)
plt.show()
```

```
fitted pars =  [ 0.83028234 -1.73469651  1.73345385 -4.70645872]
```

./analysis/Ir-O-fit.png

1.5.3 Pd-O.out

coverage (ML)	ΔH_{ads} (eV/O)
1.0	-3.11239224891
0.5	-3.92465836979
0.333333333333	-4.1049456028
0.666666666667	-3.61264246124
0.333333333333	-3.93833603536
0.666666666667	-3.59061190504
0.25	-4.25006933431
0.75	-3.54651079316
0.25	-4.18972509755
0.5	-3.92398068264
0.75	-3.5213138129
0.25	-3.96389270825
0.5	-3.64041862625
0.75	-3.46094819746
0.2	-4.22069651616
0.4	-4.02718126293
0.4	-3.89510242178
0.6	-3.68559624451
0.6	-3.72660348343
0.8	-3.42882838395

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

data = np.array(data)

thetas = data[:,0]
hads = data[:,1]

A = np.vstack([thetas**4, thetas**3, thetas**2, np.ones(len(thetas))]).T
pars = np.linalg.lstsq(A,hads)[0]
print 'fitted pars = ',pars

plt.plot(thetas, hads, 'bo ', label='DFT data')

ft = np.linspace(0,1)
A = np.vstack([ft**4, ft**3, ft**2, np.ones(len(ft))]).T
plt.plot(ft, np.dot(A,pars), label='Fit')

plt.legend(loc='best')
plt.xlabel('Coverage (ML)')
plt.ylabel('$\Delta H_{ads}$ (eV/O)')
plt.title(pars)
plt.savefig('analysis/Pd-O-fit.png', dpi=300)
plt.show()
```

```
fitted pars =  [ 2.2381205  -4.84159614  3.80350333 -4.31073989]
```

./analysis/Pd-O-fit.png

1.5.4 Pt-O.out

coverage (ML)	ΔH_{ads} (eV/O)
1.0	-2.98497267465
0.5	-3.70960646792
0.333333333333	-3.87388628502
0.666666666667	-3.50718561051
0.333333333333	-3.78095149571
0.666666666667	-3.42871278934
0.25	-4.06401377723
0.75	-3.34611018223
0.25	-4.00717277061
0.5	-3.70993332614
0.75	-3.36486082582
0.25	-3.87139490566
0.5	-3.53261093209
0.75	-3.31623022466
0.2	-4.03488768233
0.4	-3.8184615326
0.4	-3.7488555174
0.6	-3.53189394792
0.6	-3.58288972314
0.8	-3.29307464011

```
import numpy as np
import matplotlib.pyplot as plt

data = np.array(data)

thetas = data[:,0]
hads = data[:,1]

A = np.vstack([thetas**4, thetas**3, thetas**2, np.ones(len(thetas))]).T
pars = np.linalg.lstsq(A,hads)[0]
print 'fitted pars = ',pars

plt.plot(thetas, hads, 'bo ', label='DFT data')

ft = np.linspace(0,1)
A = np.vstack([ft**4, ft**3, ft**2, np.ones(len(ft))]).T
plt.plot(ft, np.dot(A,pars), label='Fit')

plt.legend(loc='best')
plt.xlabel('Coverage (ML)')
plt.ylabel('$\Delta H_{ads}$ (eV/O)')
plt.title(pars)
plt.savefig('analysis/Pt-O-fit.png', dpi=300)
plt.show()
```

```
fitted pars =  [ 2.67693764 -5.63713503  4.12922348 -4.14884994]
```

./analysis/Pt-O-fit.png

1.5.5 Cu-O.out

coverage (ML)	ΔH_{ads} (eV/O)
1.0	-2.5867482111
0.5	-3.76218392211
0.333333333333	-4.2783877723
0.666666666667	-3.33425808304
0.333333333333	-3.76878619534
0.666666666667	-3.59770017512
0.25	-4.39486104127
0.75	-3.08119119805
0.25	-4.38679913224
0.5	-3.77286532216
0.75	-3.11849907403
0.25	-3.80492647515
0.5	-3.63259206759
0.75	-3.38628934324
0.2	-4.3369809536
0.4	-4.10819127982
0.4	-3.84840666663
0.6	-3.77053044244
0.6	-3.4483064173
0.8	-3.00147004286

```
import numpy as np
import matplotlib.pyplot as plt

data = np.array(data)

thetas = data[:,0]
hads = data[:,1]

A = np.vstack([thetas**4, thetas**3, thetas**2, np.ones(len(thetas))]).T
pars = np.linalg.lstsq(A,hads)[0]
print 'fitted pars = ',pars

plt.plot(thetas, hads, 'bo ', label='DFT data')

ft = np.linspace(0,1)
A = np.vstack([ft**4, ft**3, ft**2, np.ones(len(ft))]).T
plt.plot(ft, np.dot(A,pars), label='Fit')

plt.legend(loc='best')
plt.xlabel('Coverage (ML)')
plt.ylabel('$\Delta H_{ads}$ (eV/O)')
plt.title(pars)
plt.savefig('analysis/Cu-O-fit.png', dpi=300)
plt.show()
```

```
fitted pars =  [ 1.12175057 -3.06868965  3.77240738 -4.39291491]
```

./analysis/Cu-O-fit.png

1.5.6 Ag-O.out

coverage (ML)	ΔH_{ads} (eV/O)
1.0	-1.93964906655
0.5	-2.9031492478
0.333333333333	-3.31126589417
0.666666666667	-2.56831850347
0.333333333333	-2.93461955103
0.666666666667	-2.53200882982
0.25	-3.44013252506
0.75	-2.34792655815
0.25	-3.39998767808
0.5	-2.90433750073
0.75	-2.38232347843
0.25	-2.96828405669
0.5	-2.60253306317
0.75	-2.35598935395
0.2	-3.40101048905
0.4	-3.15362422628
0.4	-2.93897131503
0.6	-2.60649495075
0.6	-2.6644333147
0.8	-2.290610442

```
import numpy as np
import matplotlib.pyplot as plt

data = np.array(data)

thetas = data[:,0]
hads = data[:,1]

A = np.vstack([thetas**4, thetas**3, thetas**2, np.ones(len(thetas))]).T
pars = np.linalg.lstsq(A,hads)[0]
print 'fitted pars = ',pars

plt.plot(thetas, hads, 'bo ', label='DFT data')

ft = np.linspace(0,1)
A = np.vstack([ft**4, ft**3, ft**2, np.ones(len(ft))]).T
plt.plot(ft, np.dot(A,pars), label='Fit')

print('At theta=0 E_ads={0}'.format(np.dot(A,pars)[0]))

plt.legend(loc='best')
plt.xlabel('Coverage (ML)')
plt.ylabel('$\Delta H_{ads}$ (eV/O)')
plt.title(pars)
```

```
plt.savefig('analysis/Ag-O-fit.png', dpi=300)
plt.show()
```

```
fitted pars =  [ 2.5743023  -6.34491459  5.36442986 -3.5306766 ]
At theta=0 E_ads=-3.53067659834
```

./analysis/Ag-O-fit.png

1.5.7 Au-O.out

coverage (ML)	ΔH_{ads} (eV/O)
1.0	-1.58970384746
0.5	-2.50745056768
0.333333333333	-2.75168883627
0.666666666667	-2.30180824844
0.333333333333	-2.67297967293
0.666666666667	-2.25905250859
0.25	-2.90650828264
0.75	-2.04335355204
0.25	-2.87688533076
0.5	-2.50721013495
0.75	-2.11395999424
0.25	-2.69357747219
0.5	-2.36639994059
0.75	-2.13939111675
0.2	-2.89593452282
0.4	-2.66930106683
0.4	-2.54289491982
0.6	-2.24701684753
0.6	-2.36991317359
0.8	-2.00347547071

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

data = np.array(data)

thetas = data[:,0]
hads = data[:,1]

A = np.vstack([thetas**4, thetas**3, thetas**2, np.ones(len(thetas))]).T
pars = np.linalg.lstsq(A,hads)[0]
print 'fitted pars = ',pars

plt.plot(thetas, hads, 'bo ', label='DFT data')

ft = np.linspace(0,1)
A = np.vstack([ft**4, ft**3, ft**2, np.ones(len(ft))]).T
plt.plot(ft, np.dot(A,pars), label='Fit')

plt.legend(loc='best')
plt.xlabel('Coverage (ML)')
plt.ylabel('$\Delta H_{ads}$ (eV/O)')
plt.title(pars)
plt.savefig('analysis/Au-O-fit.png', dpi=300)
plt.show()
```

```
fitted pars =  [ 3.41772673 -6.76426974  4.79529878 -3.03335965]
```

./analysis/Au-O-fit.png

1.6 Additional data used in the manuscript

This section contains tables of the Pt-oxygen coverage dependent DFT data used in this work. This data came from,[5] and the methods for calculating the data are in that reference.

1.6.1 Pt-O DFT data

Coverage (ML)	ΔH_f (meV/O)
1.0	0.0
0.5	-362.316896635
0.333333333333	-296.30453679
0.666666666667	-348.14195724
0.333333333333	-265.326273686
0.666666666667	-295.826743127
0.25	-269.760275645
0.75	-270.853130685
0.25	-255.55002399
0.5	-362.480325745
0.75	-284.916113378
0.25	-221.605557753
0.5	-273.81912872
0.75	-248.443162508
0.2	-209.983001536
0.4	-333.39554318
0.4	-305.5531371
0.6	-328.152763962
0.6	-358.750229094
0.8	-246.481572368

1.6.2 Pt-O Convex hull data

This data is used in Figure 4. It contains the convex hull points determined from the direct enumeration for O adsorption on Pt(111) in the fcc sites.

Coverage (ML)	ΔH_f (meV/O)
0.000000	-3.827100
0.083300	-95.673200
0.133300	-150.780900
0.142900	-161.277600
0.153800	-172.509500
0.250000	-270.788900
0.333300	-313.252300
0.363600	-323.382000
0.384600	-329.925000
0.400000	-334.316000
0.428600	-341.843000
0.500000	-360.194000
0.545500	-357.829300
0.555600	-357.303800
0.600000	-354.991700
0.615400	-353.311800
0.636400	-351.020900
0.642900	-350.311800
0.666700	-347.711800
0.727300	-302.067600
0.750000	-284.951000
0.777800	-261.164100
0.800000	-242.134600
0.857100	-185.825100
0.866700	-173.859000
0.916700	-111.037200
1.000000	-6.334000

2 Bibliography

References

[1] Brown, W.A., Kose, R., King, D.A.: Femtomole adsorption calorimetry on single-crystal surfaces. Chemical Reviews **98**(2), 797–832 (1998)

[2] Grabow, L.C., Hvolbæk, B., Nørskov, J.K.: Understanding trends in catalytic activity: The effect of adsorbate-adsorbate interactions for co oxidation over transition metals. Topics in Catalysis **53**(5-6), 298–310 (2010)

[3] Karp, E.M., Campbell, C.T., Studt, F., Abild-Pedersen, F., Nørskov, J.K.: Energetics of oxygen adatoms, hydroxyl species and water dissociation on pt(111). The Journal of Physical Chemistry C **116**(49), 25,772–25,776 (2012). DOI 10.1021/jp3066794. URL http://pubs.acs.org/doi/abs/10.1021/jp3066794

[4] Miller, S.D., Inoglu, N., Kitchin, J.R.: Configurational correlations in the coverage dependent adsorption energies of oxygen atoms on late transition metal fcc(111) surfaces. The Journal of Chemical Physics **134**(10), 104709 (2011)

[5] Miller, S.D., Kitchin, J.R.: Relating the coverage dependence of oxygen adsorption on Au and Pt fcc(111) surfaces through adsorbate-induced surface electronic structure effects. Surface Science **603**(5), 794–801 (2009)